Сетевые взаимодействия

Руководства

Настройка домена

Пример ресурса Domain (CR)

Создание домена через веб-консоль

Создание домена через CLI

Использование одного домена в нескол

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

Создание Ingress через CLI

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

Configure BGP Peers by using the web console

ıent

Настройка GatewayAPI Gateway

Обзор

Предварительные требования

Настройка

Обзор

Основы Gateway
Создание Gateway
Просмотр сведений Gateway
Справка по Listener и Route
Следующий шаг

Требования
Основы прикре
Сводка прикре
Создание поли

Настройка маршрута GatewayAPI

Обзор

Настройка ALB

ALB
Frontend
Rule
Логи и мониторинг

Configure CoreDNS

Overview
Configuration

How It Works
Configuration

Как сделать

Задачи для Ingress-Nginx

Предварительные требования
Максимальное количество соединений
Таймаут запроса
Сессионная аффинность (Sticky Sessior

Задачи для Envoy Gateway

Обзор
Предварительные требования
Продвинутые задачи
Связанная документация

Soft Data C

Предварительны
Процедура
Проверка

Устранение неполадок

Как решить проблемы межуз **Определение причины ошибки**

Руководства

Настройка домена

Пример ресурса Domain (CR)

Создание домена через веб-консоль

Создание домена через CLI

Использование одного домена в нескол

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

Создание Ingress через CLI

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

Configure BGP Peers by using the web console

ient

Настройка GatewayAPI Gateway

Обзор

Предварительные требования

Основы Gateway

Создание Gateway

Настройка

Обзор

Требования

Основы прикре

Просмотр сведений Gateway

Справка по Listener и Route

Следующий шаг

Сводка прикре

Создание поли

Настройка маршрута GatewayAPI

Обзор

Настройка ALB

ALB

Frontend

Rule

Логи и мониторинг

Configure CoreDNS

Overview

Configuration

How It Works

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

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

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

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

Содержание

Пример ресурса Domain (CR)

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

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

Создание домена через веб-консоль

1. Перейдите в раздел **Administrator**.

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

Параметр	Описание
Type	• Domain: Полное доменное имя, например, `developer.test.cn`. • Wildcard Domain: Подстановочный домен с символом подстановки (*), например, `*.test.cn`, который включает все поддомены домена `test.cn`.
Domain	Введите полное доменное имя или суффикс домена в зависимости от выбранного типа доменного имени.
Allocate Cluster	Если выделяется кластер, необходимо также выбрать проект, связанный с выделенным кластером, например, все проекты, связанные с этим 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 действительным и подписанным 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.

Параметр	Описание
	• После создания привязанного сертификата поддерживается обновление содержимого сертификата, но замена на другой сертификат не поддерживается.

5. Нажмите **Create**.

Создание домена через CLI

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

Использование одного домена в нескольких кластерах

Вы можете настроить использование одного и того же домена в нескольких кластерах, создав отдельные ресурсы Domain в глобальном кластере с одинаковым значением `spec.name`, но с разными метками `cluster.cpaas.io/name`.

Настройка через веб-консоль

1. Следуйте шагам из раздела Создание домена через веб-консоль.
2. Создайте два ресурса домена с **одинаковым доменным именем** (например, `app.example.com`).
3. Для каждого домена выберите разный **Allocate Cluster** (например, cluster-a и cluster-b).

Настройка через CLI

Создайте два ресурса Domain в глобальном кластере с одинаковым `spec.name`, но разными метками `cluster.cpaas.io/name`:

NOTE

Доменное имя (`spec.name`) должно быть одинаковым в обоих ресурсах Domain, но `metadata.name` ресурса должен быть уникальным. Оба ресурса создаются в глобальном кластере.

Домен для кластера A:

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

Домен для кластера B:

```yaml
apiVersion: crd.alauda.io/v2
kind: Domain
metadata:
  name: '${random-unique-name}'
  labels:
    cluster.cpaas.io/name: cluster-b
    project.cpaas.io/name: project-a
spec:
  name: app.example.com  # То же доменное имя, что и у кластера A
  kind: full
```

Последующие действия

- **Регистрация домена**: Зарегистрируйте домен, если созданный домен ещё 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

externalTrafficPolicy: Local

LoadBalancer с BGP

Шаги развертывания

Проверка

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. 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

Параметр	Описание
	по виртуальному IP (если включён) и TCP-порту *80*; запросы будут 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

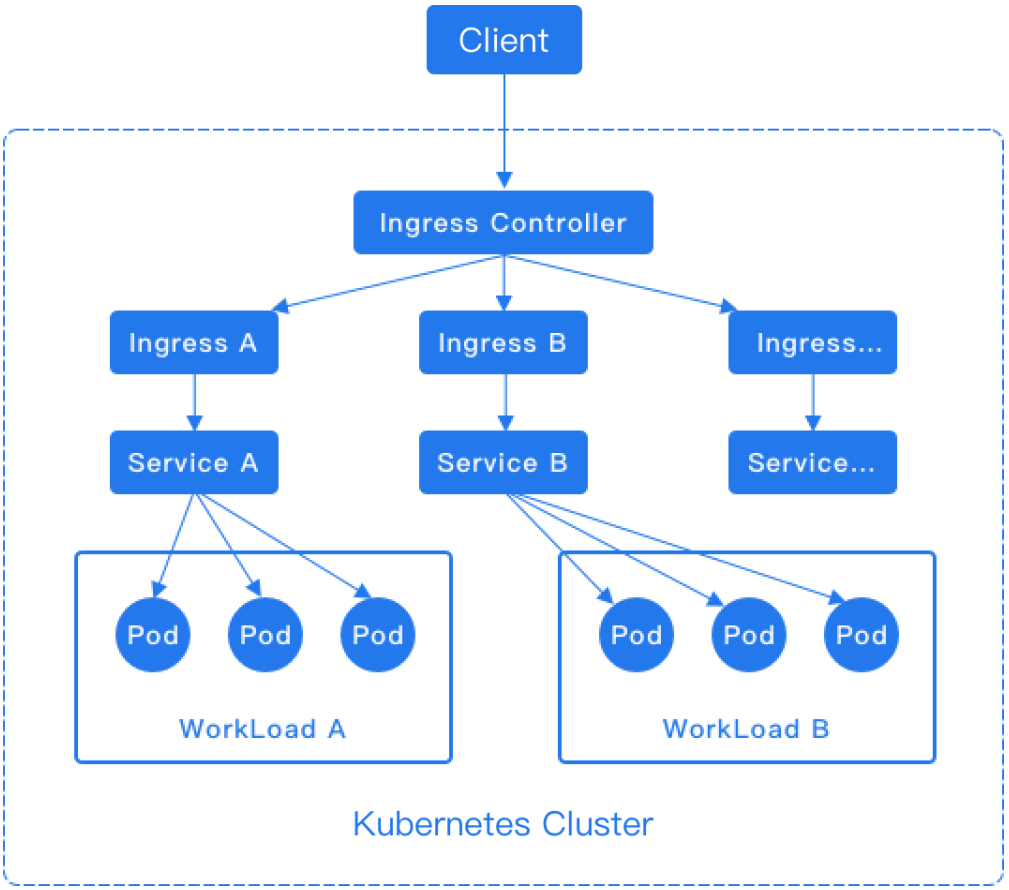
При создании нескольких ingress в одном namespace разные ingress **НЕ ДОЛЖНЫ** иметь одинаковые **Domain**, **Protocol** и **Path** (то есть дублирование точек доступа не допускается).

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. Если `default` true, используется инкапсуляция VXLAN.
2. См. параметры Encapsulation Mode и Encapsulation Protocol.

3. См. параметры Outbound Traffic NAT.

Создание подсети в сети Calico через веб-консоль

1. Перейдите в **Administrator**.
2. В левой навигационной панели 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 см. в Reference Content.
Encapsulation Protocol	Выберите протокол инкапсуляции. **IPIP** не поддерживается в режиме dual-stack. • **IPIP**: реализует межсегментную связь с помощью протокола IPIP. • **VXLAN (Alpha)**: реализует межсегментную связь с помощью протокола VXLAN. • **No Encapsulation**: прямое соединение через маршрутизацию.
Encapsulation Mode	При выборе протокола инкапсуляции **IPIP** или **VXLAN** необходимо указать режим инкапсуляции, по умолчанию — **Always**. • **Always**: всегда включать туннели IPIP / VXLAN. • **Cross Subnet**: включать туннели IPIP / VXLAN только при нахождении хоста в разных подсетях; при нахождении в одной подсети — прямое соединение через маршрутизацию.

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

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

Создание подсети в сети Calico через CLI

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

Reference Content

Динамическое соответствие между CIDR и blockSize приведено в таблице ниже.

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

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

> **NOTE**
>
> 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 узлами 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 в сеть. - **Centralized**: Все Pods кластера используют один или несколько конкретных хостов в качестве исходящих узлов, 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-шлюз обрабатывает трафик. При включении ECMP (Equal-Cost Multipath Routing) исходящий трафик маршрутизируется по нескольким равнозначным путям ко всем доступным узлам-шлюзам, что увеличивает общую пропускную способность шлюза. **Примечание**: Необходимо предварительно включить соответствующие функции.
Gateway Nodes	При использовании **Centralized** gateway выберите один или несколько конкретных хостов в качестве узлов-шлюзов.
Outbound Traffic NAT	Выберите, включать ли NAT для исходящего трафика. По умолчанию включено. Используется для задания адреса доступа, видимого во внешней сети при выходе Pods подсети в интернет. При включённом NAT в качестве адреса доступа используется IP хоста;

Параметр	Описание
	при отключённом NAT IP Pods подсети напрямую видны во внешней сети. В этом случае рекомендуется использовать централизованный шлюз.

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

Создание подсети в Overlay-сети Kube-OVN через CLI

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

Underlay-сеть

Создание подсетей в Underlay-сети Kube-OVN обеспечивает не только 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 перенаправлять 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 определённым 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 нести другой трафик, например SSH. Например, если в

мостовой сети три узла, планирующие eth0, eth0, eth1 для исключительного использования Underlay, то сетевая карта по умолчанию может быть eth0, а для третьего узла — eth1.

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

5. Нажмите **Add**.

Добавление Bridge Network через CLI

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

Добавление VLAN через веб-консоль (опционально)

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

1 VLAN ID.

2 Ссылка на мостовую сеть.

Платформа имеет преднастроенную виртуальную LAN **ovn-vlan**, которая подключается к мостовой сети **provider**. Также можно создать новую VLAN, подключённую к другим мостовым сетям, 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 мостовой сети для связи с физической сетью.

5. Нажмите **Add**.

Добавление VLAN через CLI

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

Пример custom resource (CR) подсети с 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 узлами и Pods в Overlay-режиме. 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 для вычислительных компонентов.

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

Создание подсети в Underlay-сети Kube-OVN через CLI

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

Связанные операции

При наличии в кластере как Underlay, так и Overlay подсетей можно при необходимости настроить Автоматическую связь между Underlay и Overlay подсетями.

Управление подсетями

Обновление шлюза через веб-консоль

Включает изменение метода исходящего трафика, узлов-шлюзов 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 см. в разделе 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, 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 см. в разделах назначение проектов и назначение пространств имён.

Удаление подсети через веб-консоль

> **NOTE**
>
> - При удалении подсети, если контейнерные группы всё ещё используют IP из этой подсети, они смогут продолжать работу с неизменными IP, но не смогут обмениваться данными по сети. Контейнерные группы можно пересоздать для использования IP из подсети по умолчанию или назначить новое подсеть пространству имён, где находятся контейнерные группы.
> - 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,

Parameter	Description
	итоговые доступные узлы — пересечение обоих.
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

Обзор

В этом документе объясняется, как настроить `Gateway` после того, как оператор Envoy Gateway и `EnvoyGatewayCtl` готовы. `Gateway` определяет, как трафик поступает в шлюз, в то время как сопутствующий `EnvoyProxy` контролирует, как развертывается базовая плоскость данных Envoy.

В рекомендуемом рабочем процессе этот документ следует после Envoy Gateway Operator и перед Configure GatewayAPI Route.

Предварительные требования

Пожалуйста, убедитесь, что вы выполнили следующие шаги перед продолжением:

1. Ознакомились с Envoy Gateway Operator, чтобы понять основные концепции и взаимосвязи ресурсов
2. Установили оператор Envoy Gateway и создали `EnvoyGatewayCtl`

> **NOTE**
>
> В этом документе сначала объясняются основные концепции Gateway, а затем показано, как создать `Gateway`. Если вы уже знакомы с этими концепциями, можете перейти к разделу Create Gateway.

Основы Gateway

Что такое Gateway

`Gateway` — это точка входа для трафика, поступающего в ваш кластер. Он определяет, как внешние запросы принимаются и маршрутизируются к вашим бэкенд-сервисам. `Gateway` в основном определяет:

- **Listeners**: определяют порты, протоколы и имена хостов, на которых слушает шлюз
- **GatewayClass**: выбирает, какой контроллер шлюза управляет этим `Gateway`
- **Ссылка на инфраструктуру**: ссылается на `EnvoyProxy`, который контролирует, как развертывается базовая плоскость данных Envoy

Экспозиция Gateway (тип сервиса)

Тип сервиса настраивает, как шлюз экспонируется через базовый сервис Envoy. Существует три режима: LoadBalancer, NodePort и ClusterIP.

В YAML эта настройка задаётся в сопутствующем ресурсе `EnvoyProxy` по пути `.spec.provider.kubernetes.envoyService.type`.

LoadBalancer (Рекомендуется)

Преимущество — простота использования и возможности балансировки нагрузки с высокой доступностью. Для использования LoadBalancer в кластере должна быть поддержка LoadBalancer, которую можно включить через MetalLB.

При использовании MetalLB можно указать статический VIP через аннотации сервиса. В веб-консоли используйте поле **Service Annotation**:

```
metallb.universe.tf/address-pool: ADDRESS_POOL_NAME
# Или укажите конкретный IP напрямую
metallb.universe.tf/loadBalancerIPs: VIP_IP
```

Подробнее см. How To Specify a VIP When Using MetalLB.

NodePort

Преимущество — не требует внешних зависимостей.

Однако у NodePort есть следующие недостатки:

- При использовании NodePort Kubernetes назначает номера портов NodePort, отличающиеся от портов самого сервиса. Для доступа нужно использовать порт NodePort, а не порт сервиса.
- Сервис доступен по IP-адресу любого узла в кластере, что может представлять потенциальные риски безопасности.

Как получить правильный порт при использовании NodePort

На странице сведений Gateway, когда тип сервиса — NodePort, в списке слушателей отображается столбец NodePort с назначенными номерами портов. Также можно использовать следующую команду:

```
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 внешняя экспозиция не требуется.

Конфигурация Listener

Listener определяет порт и протокол, на которых шлюз слушает трафик. В протоколах HTTP или HTTPS разные имена хостов могут рассматриваться как разные слушатели.

Нельзя создать слушателя с конфликтующими портом, протоколом или именем хоста.

В `Gateway` необходимо создать как минимум одного слушателя.

Порт и протокол

Каждый слушатель настраивается с номером порта и протоколом. Поддерживаемые протоколы: HTTP, HTTPS, TCP, UDP, TLS.

AllowRouteNS

По умолчанию маршруты (Routes) могут быть прикреплены только к Gateway в том же namespace (`Same`). Чтобы разрешить маршрутизацию между пространствами имён, используйте поле `Allowed Routes Namespace`:

- `Same`: разрешить маршрутам из того же namespace прикрепляться к этому слушателю.
- `All`: разрешить маршрутам из любого namespace прикрепляться к этому слушателю.
- `Selector`: разрешить маршрутам из namespace, соответствующих селектору, прикрепляться к этому слушателю.

В ACP проект идентифицируется метками на namespace, например `cpaas.io/project: <project-name>`. Если вы хотите, чтобы слушатель использовался только маршрутами из определённого проекта, используйте `Selector` и укажите метку проекта для целевых namespace.

Настройка `Allowed Routes Namespace` слушателя вместе с его протоколом определяет, какие слушатели доступны в веб-консоли маршрутов при публикации маршрута на слушатель.

Подробнее см. attach to gateway created in other ns.

Конфигурация TLS

Для протоколов HTTPS и TLS необходимо настроить параметры TLS.

Режимы TLS:

Режим TLS	Описание	Требуется сертификат
Terminate	Envoy завершает TLS и расшифровывает трафик перед передачей бэкендам	Да, необходимо выбрать TLS-сертификат

Режим TLS	Описание	Требуется сертификат
Passthrough	Envoy пропускает зашифрованный TLS-трафик напрямую к бэкендам без расшифровки	Нет

> **NOTE**
>
> - Протокол HTTPS поддерживает только режим **Terminate**
> - Протокол TLS поддерживает режимы **Terminate** и **Passthrough**
> - TLS слушатели в режиме **Passthrough** поддерживают TLSRoute
> - TLS слушатели в режиме **Terminate** поддерживают TCPRoute

Требования к сертификатам:

- По умолчанию можно использовать только секреты, созданные в том же namespace
- Секрет должен иметь тип `kubernetes.io/tls` и содержать ключи `tls.crt` и `tls.key`
- Для секретов из других namespace см. use secret created in other ns

EnvoyProxy (конфигурация развертывания)

Envoy Gateway использует ресурс `EnvoyProxy` для управления конфигурациями развертывания шлюза. Рекомендуется создавать отдельный ресурс `EnvoyProxy` для каждого Gateway и ссылаться на него через поле `.spec.infrastructure.parametersRef` Gateway.

При создании `Gateway` из веб-консоли с использованием `GatewayClass`, созданного `EnvoyGatewayCtl`, консоль автоматически создаёт сопутствующий ресурс `EnvoyProxy` с тем же именем и namespace.

При создании `Gateway` через применение YAML вы несёте ответственность за согласованность `.spec.infrastructure.parametersRef` Gateway и соответствующего ресурса `EnvoyProxy`.

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

- Количество реплик
- Лимиты и запросы ресурсов
- Селекторы узлов
- Тип сервиса и аннотации
- Репозиторий образов

Репозиторий образов

Репозиторий образов преднастроен со значением по умолчанию для вашего кластера. Не изменяйте его без необходимости.

Для других способов конфигурации развертывания см. deployment-mode↗.

Создание Gateway

Через веб-консоль

1. Перейдите в `Alauda Container Platform -> Networking -> Gateway -> Gateways`
2. Нажмите кнопку `Create Gateway`
3. На странице `Create Gateway` выберите `GatewayClass`, созданный вашим `EnvoyGatewayCtl`. В рекомендуемом примере по умолчанию из Envoy Gateway Operator это `envoy-gateway-operator-cpaas-default`.

Страница отображает следующие параметры конфигурации:

Поле	Описание	Путь в YAML
Name	Имя Gateway	gateway: `.metadata.name` envoyproxy: `.metadata.name`
GatewayClass	Какой GatewayClass	gateway: `.spec.gatewayClassName`

Поле	Описание	Путь в YAML
	использовать	
Service Type	Тип сервиса	envoyproxy: `.spec.provider.kubernetes.envoyService.t`
Service Annotation	Аннотации сервиса	envoyproxy: `.spec.provider.kubernetes.envoyService.a`
Resource Limits	Лимиты ресурсов развертывания	envoyproxy: `.spec.provider.kubernetes.envoyDeployment.containe`
Replicas	Количество реплик развертывания	envoyproxy: `.spec.provider.kubernetes.envoyDeploymer`
Node Labels	Селекторы узлов развертывания	envoyproxy: `.spec.provider.kubernetes.envoyDeployment.nodeSele`
Listener	Listener	gateway: `.spec.listeners`

WARNING

Форма веб-консоли поддерживает только GatewayClasses, созданные `EnvoyGatewayCtl`.

Для других GatewayClasses используйте редактор YAML.

NOTE

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

Конфигурация Listener

При создании или редактировании слушателя можно настроить следующие параметры:

Поле	Описание	Путь в YAML
Name	Имя слушателя	`.spec.listeners[].name`
Port	Номер порта, на котором слушает слушатель	`.spec.listeners[].port`
Protocol	Протокол слушателя. Опции: HTTP, HTTPS, TCP, UDP, TLS	`.spec.listeners[].protocol`
Hostname	Необязательно. Имя хоста для слушателя	`.spec.listeners[].hostname`
Allowed Routes Namespace	Контролирует, из каких namespace маршруты могут прикрепляться к слушателю	`.spec.listeners[].allowedRoutes.namespaces.from`

Конфигурация протокола HTTPS

При выборе протокола **HTTPS**:

Поле	Описание	Путь в YAML
Certificates	Обязательно. Выберите Kubernetes-секрет с TLS-сертификатом	`.spec.listeners[].tls.certificateRefs`

NOTE

- Протокол HTTPS поддерживает только режим **Terminate**

- Выбор сертификата обязателен для HTTPS слушателей
- По умолчанию можно использовать только секреты, созданные в том же namespace

Конфигурация протокола TLS

При выборе протокола **TLS**:

Поля конфигурации

Поле	Описание	Путь в YAML
TLS Mode	Выберите режим TLS. Опции: Terminate, Passthrough. По умолчанию: Terminate	`.spec.listeners[].tls.mode`
TLS Certificate	Обязательно только при режиме TLS Mode = Terminate. Выберите секрет с TLS-сертификатом	`.spec.listeners[].tls.certificateRefs`

Подробнее о режимах TLS см. в разделе TLS Configuration.

Через YAML

Замените `envoy-gateway-operator-cpaas-default` на `GatewayClass`, созданный вашим собственным `EnvoyGatewayCtl`, если вы не используете рекомендуемый пример по умолчанию.

Следующий минимальный пример создаёт HTTP `Gateway` и выделенный `EnvoyProxy`.

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo
  namespace: demo
spec:
  infrastructure:
    parametersRef:
      group: gateway.envoyproxy.io
      kind: EnvoyProxy
      name: demo
  gatewayClassName: envoy-gateway-operator-cpaas-default
  listeners:
    - name: http
      port: 80
      protocol: HTTP
      allowedRoutes:
        namespaces:
          from: Same
---
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: demo
  namespace: demo
spec:
  provider:
    kubernetes:
      envoyService:
        type: ClusterIP
      envoyDeployment:
        replicas: 1
        container:
          imageRepository: registry.alauda.cn:60080/acp/envoyproxy/envoy
          resources:
            limits:
              cpu: '1'
              memory: 1Gi
            requests:
              cpu: '1'
              memory: 1Gi
    type: Kubernetes
```

Полный пример с несколькими типами слушателей

Если вам нужны дополнительные типы слушателей, используйте следующий полный пример:

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

```
      protocol: TLS
      allowedRoutes:
        namespaces:
          from: Same
      tls:
        mode: Terminate
        certificateRefs:
          - name: demo-tls
    - name: tcp
      port: 8080
      protocol: TCP
      allowedRoutes:
        namespaces:
          from: Same
    - name: udp
      port: 8081
      protocol: UDP
      allowedRoutes:
        namespaces:
          from: Same
---
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: demo (8)
  namespace: demo
spec:
  provider:
    kubernetes:
      envoyService:
        type: ClusterIP (9)
      envoyDeployment:
        replicas: 1
        container:
          imageRepository: registry.alauda.cn:60080/acp/envoyproxy/envoy
(10)
          resources: (11)
            limits:
              cpu: '1'
              memory: 1Gi
            requests:
              cpu: '1'
              memory: 1Gi
    type: Kubernetes (12)
```

1. Ссылка на ресурс EnvoyProxy для конфигурации развертывания
2. Замените `envoy-gateway-operator-cpaas-default` на ваш собственный `GatewayClass`, если необходимо
3. `listeners` определяет, как трафик поступает в шлюз
4. `hostname` влияет на сопоставление маршрутизации по хосту со слушателем
5. `protocol` определяет, какие типы маршрутов могут прикрепляться к слушателю
6. `allowedRoutes` контролирует, из каких namespace маршруты могут прикрепляться
7. `tls` настраивает завершение TLS или пропуск для слушателей HTTPS и TLS
8. Имя `EnvoyProxy` должно совпадать с `.spec.infrastructure.parametersRef.name`
9. `envoyService.type` управляет экспозицией шлюза
10. Не изменяйте `imageRepository`, если это не требуется для вашей среды
11. `resources` настраивает лимиты и запросы ресурсов плоскости данных Envoy
12. Оставьте `provider.type` равным `Kubernetes`

Просмотр сведений Gateway

На странице сведений Gateway в списке слушателей отображается следующая информация:

Столбец	Описание
Name	Имя слушателя
Protocol	Протокол слушателя (HTTP, HTTPS, TCP, UDP, TLS)
Port	Настроенный номер порта
NodePort	Отображается, когда тип сервиса — NodePort. Показывает назначенный номер NodePort для доступа к слушателю.

NOTE

Когда тип сервиса Gateway — **NodePort**, в списке слушателей появляется дополнительный столбец **NodePort**. Для доступа к шлюзу используйте значение NodePort, а не порт сервиса. Подробнее см. How to Get the Correct Port When Using NodePort.

Справка по Listener и Route

Используйте следующие правила при прикреплении ресурсов `Route` к `Gateway`.

Имя хоста

Имя хоста в слушателе является уникальным идентификатором для слушателей с одинаковым протоколом. Нельзя добавить или обновить слушателя с конфликтующим именем хоста в одном шлюзе.

Правило пересечения имён хостов

При поступлении запроса он сопоставляется с **пересечением** имени хоста слушателя и имён хостов маршрута. Для маршрутизации используется только пересечение имён хостов.

Имя хоста слушателя	Имена хостов маршрута	Результат пересечения	Пример
Нет имени хоста	Нет имён хостов	Совпадает со всеми хостами	Любой входящий заголовок Host принимается
Нет имени хоста	Есть имена хостов (например, `api.example.com`)	Все имена хостов маршрута	Совпадают только запросы с `api.example.com`
Есть имя хоста (например, `api.example.com`)	Нет имён хостов	Все имена хостов слушателя	Совпадают только запросы с `api.example.com`

Имя хоста слушателя	Имена хостов маршрута	Результат пересечения	Пример
Есть имя хоста (например, `api.example.com`)	Есть точное совпадающее имя хоста	Точное совпадение имени хоста	Совпадают только запросы с `api.example.com`
Есть подстановочный знак (например, `*.example.com`)	Есть совпадающие имена хостов	Совпадающие конкретные имена хостов	Совпадают запросы с `api.example.com` или `web.example.com`
Есть имя хоста (например, `api.example.com`)	Есть несовпадающие имена хостов	**Нет пересечения** — статус маршрута аномальный	Маршрут не может обрабатывать трафик

NOTE

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

WARNING

Отсутствие пересечения означает, что статус маршрута становится аномальным и трафик не может обрабатываться.

Поддерживаемые типы маршрутов

Каждый слушатель поддерживает разные типы маршрутов в зависимости от протокола:

Протокол слушателя	Поддерживаемый тип маршрута
HTTP	HTTPRoute, GRPCRoute

Протокол слушателя	Поддерживаемый тип маршрута
HTTPS	HTTPRoute, GRPCRoute
TLS (режим Passthrough)	TLSRoute
TLS (режим Terminate)	TCPRoute
TCP	TCPRoute
UDP	UDPRoute

При настройке маршрутов убедитесь, что они соответствуют протоколу слушателя, к которому прикрепляются. Например, нельзя прикрепить `HTTPRoute` к слушателю с протоколом `TCP`.

Следующий шаг

После готовности `Gateway` продолжайте с Configure GatewayAPI Route. Если вам требуется продвинутый контроль трафика после прикрепления маршрутов, продолжайте с Configure GatewayAPI Policy.

Настройка политики GatewayAPI

Содержание

Обзор

В этом документе объясняется, как настроить ресурсы политики после того, как ресурсы `Gateway` и `Route` готовы. Политики используют паттерн прикрепления политики через `.spec.targetRefs` для добавления дополнительного поведения трафика, безопасности и бэкенда к поддерживаемым ресурсам.

В рекомендуемом рабочем процессе этот документ следует после Настройка маршрута GatewayAPI.

Envoy Gateway в настоящее время предоставляет четыре типа политик: `SecurityPolicy`, `BackendTLSPolicy`, `ClientTrafficPolicy` и `BackendTrafficPolicy`.

Требования

Пожалуйста, убедитесь, что вы выполнили следующие шаги перед продолжением:

1. Ознакомились с Настройкой GatewayAPI Gateway и Настройкой маршрута GatewayAPI
2. Создали целевой ресурс, к которому будет прикреплена политика, например `Gateway`, `Route` или `Service`

Основы прикрепления политики

Политики прикрепляются к другим ресурсам через `.spec.targetRefs`.

По умолчанию политика может быть прикреплена только к ресурсам в том же namespace.

Для целей `Gateway` можно использовать `sectionName` для указания конкретного слушателя, если тип политики это поддерживает. Для целей `Service` `sectionName` относится к имени порта Service.

Сводка прикрепления политики

Тип политики	Назначение	Поддержка в Web Console	Gateway	HTTPRou
SecurityPolicy	Аутентификация, авторизация, CORS и другие функции безопасности	API Key Auth, CORS	✅ (имя слушателя / `ALL`)	✅
BackendTLSPolicy	Настройка TLS между Envoy и бэкенд-сервисами	Поддерживается	❌	❌

Тип политики	Назначение	Поддержка в Web Console	Gateway	HTTPRou
ClientTrafficPolicy	Таймауты и поведение соединений на стороне клиента	Настройки таймаутов	✅ (имя слушателя / `ALL`)	❌
BackendTrafficPolicy	Таймауты и поведение соединений на стороне бэкенда	Настройки таймаутов	✅ (имя слушателя / `ALL`)	✅

`sectionName` используется для указания конкретного слушателя на `Gateway` или конкретного порта на `Service`. Если опущено или установлено в `ALL`, политика применяется ко всем слушателям или портам.

Создание политик в Web Console

Все типы политик создаются из одного пункта:

1. Перейдите в `Alauda Container Platform -> Networking -> Gateway -> Policies`
2. Выберите нужное значение в выпадающем списке `Policy Type`
3. Нажмите кнопку `Create Policy`

В следующих разделах рассматриваются только поля, специфичные для каждого типа политики.

SecurityPolicy

Настройка через Web Console

Общие поля (общие для всех политик):

Поле	Описание	Путь в YAML
Policy Type	Тип создаваемой политики	`.kind`
Attach To	Ресурсы Gateway API, к которым применяется эта политика. Поддерживаются Gateway, HTTPRoute и GRPCRoute. При прикреплении к Gateway можно указать имя слушателя или выбрать ALL слушателей.	`.spec.targetRefs`

Специфичные поля SecurityPolicy:

Поле	Описание	Путь в YAML
Authorization Type	Метод аутентификации/авторизации. Поддерживает множественный выбор: API Key Authentication, CORS Configuration	`.spec.apiKeyAuth`, `.spec.cors`

Аутентификация по API Key

Поле	Описание	Путь в YAML
Secrets	Kubernetes-секреты, содержащие API ключи для аутентификации	`.spec.apiKeyAuth.credentialRefs`
Extract From	Указывает, откуда извлекать API ключ (HTTP заголовки или параметры запроса)	`.spec.apiKeyAuth.extractFrom`

Конфигурация CORS

Поле	Описание	Путь в YAML
Allow Origins	Список разрешённых источников для CORS-запросов	`.spec.cors.allowOrigins`
Allow Methods	Список разрешённых HTTP методов	`.spec.cors.allowMethods`

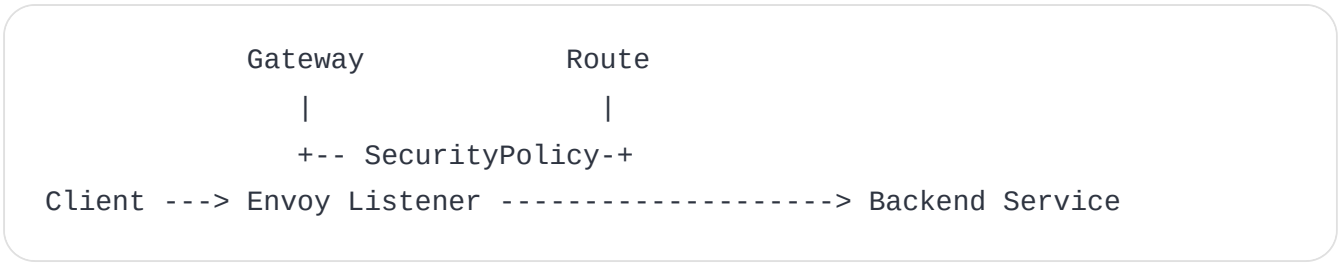
Поле	Описание	Путь в YAML
Allow Headers	Список разрешённых заголовков в CORS-запросах	`.spec.cors.allowHeaders`
Expose Headers	Список заголовков, доступных клиенту в ответе	`.spec.cors.exposeHeaders`
Max Age	Время кэширования preflight-ответа CORS	`.spec.cors.maxAge`
Allow Credentials	Разрешены ли учётные данные в CORS-запросах	`.spec.cors.allowCredentials`

Настройка через YAML

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

Справка

SecurityPolicy используется для настройки аутентификации, авторизации и других функций безопасности для вашего Gateway и маршрутов. Она предоставляет декларативный способ защиты сервисов путём проверки входящих запросов до того, как они достигнут ваших бэкенд-приложений.



Возможности

- **Аутентификация**: Проверка идентичности клиентов с помощью различных методов (API Key, JWT, OIDC, Basic Auth)
- **Авторизация**: Контроль доступа к ресурсам на основе проверенных учётных данных
- **Конфигурация CORS**: Управление политиками Cross-Origin Resource Sharing

Как это работает

1. Создайте SecurityPolicy с нужными правилами аутентификации/авторизации
2. Прикрепите её к конкретному Gateway, HTTPRoute или GRPCRoute
3. Envoy Gateway проверяет входящие запросы согласно политике
4. Валидные запросы перенаправляются к бэкенд-сервисам; невалидные отклоняются с соответствующими HTTP статусами

Примечания

1. В веб-консоли в настоящее время поддерживается настройка **API Key Authentication** и **CORS**. Для других методов аутентификации и расширенных функций безопасности необходимо использовать YAML-конфигурацию.
2. Каждый маршрут может быть связан только с одной SecurityPolicy.
3. Если SecurityPolicy ссылается на секрет без значений, все запросы к прикреплённому маршруту будут отклонены с кодом `401 Unauthorized`.
4. В веб-консоли по умолчанию поле `Extract From` установлено в `header`, а поле `Header Name` — в `authorization`.
5. Вы можете просмотреть, какие политики прикреплены к маршруту, перейдя на вкладку топологии маршрута в веб-консоли.

Официальная документация

- Спецификация SecurityPolicy ↗
- API Key Authentication ↗

BackendTLSPolicy

Настройка через Web Console

Общие поля:

Поле	Описание	Путь в YAML
Policy Type	Тип создаваемой политики	`.kind`
Attach To	Service, к которому применяется эта политика. Необходимо указать имя Service и имя порта (`sectionName`).	`.spec.targetRefs`

Специфичные поля BackendTLSPolicy:

Поле	Описание	Путь в YAML
Hostname	Обязательно. SNI (Server Name Indication), используемый при подключении Envoy к бэкенд-сервису	`.spec.validation.hostname`
Subject Alternative Names	Необязательно. Используется для валидации HTTPS-ответа бэкенда. Если не указано, по умолчанию используется значение hostname.	`.spec.validation.subjectAltNames`
Validation Type	Метод валидации TLS-сертификатов бэкенда. Опции: `CACertificateRefs` (использовать пользовательские CA-сертификаты),	`.spec.validation`

Поле	Описание	Путь в YAML
	`WellKnownCACertificates` (использовать системные CA)	

Конфигурация CACertificateRefs:

Поле	Описание	Путь в YAML
CA Certificate Secret	Kubernetes-секрет, содержащий CA-сертификат. Секрет должен содержать ключ `ca.crt` с PEM-кодированными TLS-сертификатами.	`.spec.validation.cACertificateRefs`

NOTE

При создании или выборе секрета с CA-сертификатом:

- Тип секрета должен подходить для CA-сертификатов
- Ключ должен быть `ca.crt`
- Можно импортировать файл сертификата, который должен начинаться с `-----BEGIN CERTIFICATE-----` и заканчиваться `-----END CERTIFICATE-----`
- При импорте некорректного формата сертификата будет показано сообщение об ошибке "must contain PEM-encoded TLS certificates"
- При выборе существующего секрета без ключа `ca.crt` будет показано сообщение об ошибке "must have ca.crt key"

Настройка через YAML

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

Справка

BackendTLSPolicy управляет настройками TLS между Envoy Gateway и бэкенд-сервисами. Она позволяет настроить:

- **SNI (Server Name Indication)**: имя хоста, используемое при установлении TLS-соединений с бэкендами
- **Валидация сертификатов**: способ проверки сертификатов серверов бэкенда
- **CA-сертификаты**: пользовательские CA-сертификаты для проверки сертификатов бэкенда

Возможности

- Настройка TLS для соединений с бэкенд-сервисами
- Поддержка пользовательских CA-сертификатов или системных известных CA-сертификатов
- Конфигурация SNI для корректного TLS-рукопожатия

Примечания

1. `sectionName` в `targetRefs` соответствует имени порта Service.
2. При использовании `WellKnownCACertificates` для валидации используются системные CA-сертификаты по умолчанию.
3. Поле hostname обязательно и используется как значение SNI при подключении Envoy к бэкенду.

Официальная документация

- Спецификация BackendTLSPolicy ↗

ClientTrafficPolicy

Настройка через Web Console

Общие поля:

Поле	Описание	Путь в YAML
Policy Type	Тип создаваемой политики	`.kind`
Attach To	Gateway, к которому применяется эта политика. Можно указать имя слушателя или выбрать ALL слушателей.	`.spec.targetRefs`

Настройка таймаутов (опции):

Поле	Описание	Путь в YAML
TCP Idle Timeout	Таймаут простоя TCP-соединения. Время простоя определяется как период, в течение которого не передаются данные ни в одном из направлений (upstream или downstream). По умолчанию: 1 час.	`.spec.settings.timeout.tcp.idleTimeout`
HTTP Request Received Timeout	Время ожидания Envoy полного получения запроса. Таймер запускается при начале запроса и останавливается при отправке последнего байта запроса upstream или начале ответа. По умолчанию: 1 час.	`.spec.settings.timeout.http.requestReceivedTimeout`
HTTP Idle Timeout	Таймаут простоя HTTP-соединения. Время простоя определяется как период, в течение которого в	`.spec.settings.timeout.http.idleTimeout`

Поле	Описание	Путь в YAML
	соединении нет активных запросов. По умолчанию: неограничено.	
HTTP Stream Idle Timeout	Таймаут простоя потока определяет время, в течение которого поток может существовать без активности upstream или downstream. По умолчанию: 5 минут.	`.spec.settings.timeout.http.streamIdleTimeout`

Настройка через YAML

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

Справка

ClientTrafficPolicy управляет поведением соединений от клиентов к Envoy Gateway. Она предоставляет тонкую настройку:

```
            Gateway
               |
               +-- ClientTrafficPolicy
Client ---> Envoy Listener --------------------> Backend Service
          применяется на стороне клиента ^
```

- **Настройки TCP**: таймауты и параметры keepalive на уровне соединения
- **Настройки HTTP**: таймауты запросов/ответов и поведение HTTP-протокола

Возможности

- Настройка таймаутов простоя TCP-соединений
- Контроль таймаутов получения HTTP-запросов

- Установка таймаутов простоя HTTP-соединений
- Настройка таймаутов простоя HTTP-потоков

Примечания

1. Значения таймаутов задаются в виде строк с длительностью (например, "30s", "5m", "1h").

Официальная документация

- Спецификация ClientTrafficPolicy ↗

BackendTrafficPolicy

Настройка через Web Console

Общие поля:

Поле	Описание	Путь в YAML
Policy Type	Тип создаваемой политики	`.kind`
Attach To	Ресурсы Gateway API, к которым применяется эта политика. Поддерживаются Gateway, HTTPRoute, GRPCRoute, TCPRoute, UDPRoute и TLSRoute. При прикреплении к Gateway можно указать имя слушателя или выбрать `ALL` слушателей. В YAML это настраивается через `sectionName` в `.spec.targetRefs`.	`.spec.targetRefs`

Настройка таймаутов (опции):

Поле	Описание	Путь в YAML
TCP Connection	Таймаут установления	`.spec.settings.timeout.tcp.connectionTimeout`

Поле	Описание	Путь в YAML
Timeout	сетевого соединения, включая TCP и TLS рукопожатия. По умолчанию: 10 секунд.	
HTTP Connection Idle Timeout	Таймаут простоя HTTP-соединения. Время простоя определяется как период, в течение которого в соединении нет активных запросов. По умолчанию: 1 час.	`.spec.settings.timeout.http.connectionIdleTimeout`
HTTP Max Connection Duration	Максимальная длительность HTTP-соединения. По умолчанию: неограничено.	`.spec.settings.timeout.http.maxConnectionDuration`
HTTP Request Timeout	Время до получения полного ответа от upstream. По умолчанию: 15 секунд. Поддерживается установка в неограниченное значение.	`.spec.settings.timeout.http.requestTimeout`

Настройка через YAML

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

Справка

BackendTrafficPolicy управляет поведением соединений от Envoy Gateway к бэкенд-сервисам. Она предоставляет тонкую настройку:

```
            Gateway / Route
                  |
                  +-- BackendTrafficPolicy
Client ---> Envoy Listener --------------------> Backend Service
                                  применяется здесь ^
```

- **Настройки TCP**: таймауты установления соединений
- **Настройки HTTP**: длительности соединений, таймауты простоя и таймауты запросов

Возможности

- Настройка таймаутов установления TCP-соединений

- Контроль таймаутов простоя HTTP-соединений
- Установка максимальной длительности HTTP-соединений
- Настройка таймаутов HTTP-запросов

Примечания

1. Значения таймаутов задаются в виде строк с длительностью (например, "30s", "5m", "1h").
2. Поле `requestTimeout` поддерживает установку значения "unlimited" для отключения таймаута.

Официальная документация

- Спецификация BackendTrafficPolicy ↗

Связанные задачи

После прикрепления политик продолжайте с Задачами для Envoy Gateway для получения дополнительных примеров эксплуатации и задач по расширенной настройке.

Настройка маршрута GatewayAPI

Содержание

Связанные задачи

Обзор

В этом документе объясняется, как настроить ресурсы `Route` после того, как `Gateway` готов. `Route` прикрепляется к одному или нескольким слушателям gateway и определяет, как совпадающий трафик перенаправляется на backend-сервисы.

В рекомендуемом рабочем процессе этот документ следует после Настройка GatewayAPI Gateway и перед Настройка GatewayAPI Policy.

Помимо операций создания и обновления, в этом документе также представлены дополнительные возможности просмотра маршрутов, предоставляемые ACP Web Console.

Предварительные требования

Пожалуйста, убедитесь, что вы выполнили следующие шаги перед продолжением:

1. Ознакомились с Настройка GatewayAPI Gateway для понимания слушателей, правил прикрепления и `EnvoyProxy`
2. Создали `Gateway`, к которому будет прикреплен ваш `Route`

> **NOTE**
>
> В этом документе сначала отдельно представлены каждый тип маршрута, затем приведены примеры YAML, и, наконец, объяснены общие концепции маршрутов в общей справочной секции.

Конфигурация

`Route` прикрепляется к одному или нескольким слушателям на `Gateway`. Выбор слушателя зависит от типа маршрута, протокола слушателя и настроек разрешённых пространств имён маршрутов слушателя.

Конфигурация через Web Console

1. Перейдите в `Alauda Container Platform -> Networking -> Gateway -> Routes`
2. Нажмите кнопку `Create Route`
3. Выберите тип маршрута (HTTPRoute, TCPRoute, UDPRoute, GRPCRoute или TLSRoute)

Создание HTTPRoute

Поле	Описание	Путь в YAML
Publish to Listener	публикация на слушатель	`.spec.parentRefs`
Hostnames	hostnames	`.spec.hostnames`
Matches	совпадения	`.spec.rules[].matches`
Filters	фильтры	`.spec.rules[].filters`
Backend Instance	backend	`.spec.rules[].backendRefs`
Options	опции	`.spec.rules[].filters`, `.spec.rules[].timeouts`, `.spec.rules[].retry`, `.spec.rules[].sessionPersistence`

Конфигурация опций

Поле Options позволяет настроить расширенные параметры управления трафиком:

Опция	Описание	Путь в YAML
Session Affinity	сохранение сессии	`.spec.rules[].sessionPersistence`

Опция	Описание	Путь в YAML
Timeout	настройки таймаута	`.spec.rules[].timeouts`
Retry	политика повторных попыток	`.spec.rules[].retry`

Создание TCP/UDP Route

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

Создание GRPCRoute

Поле	Описание	Путь в YAML
Publish to Listener	публикация на слушатель	`.spec.parentRefs`
Hostnames	hostnames	`.spec.hostnames`
Matches	grpc совпадения	`.spec.rules[].matches`
Filters	grpc фильтры	`.spec.rules[].filters`
Backend Instance	backend	`.spec.rules[].backendRefs`

Создание TLSRoute

Поле	Описание	Путь в YAML
Publish to Listener	публикация на слушатель	`.spec.parentRefs`
Hostnames	hostnames (опционально)	`.spec.hostnames`
Backend Instance	backend	`.spec.rules[].backendRefs`

Конфигурация через YAML

Следующий минимальный пример создаёт `HTTPRoute`, который прикрепляется к слушателю `https` в `Gateway` с именем `demo` и перенаправляет совпадающий трафик на backend `Service`.

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

Если вам нужны другие типы маршрутов и расширенные опции HTTPRoute, используйте следующий полный пример:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: demo-443
  namespace: demo
spec:
  hostnames: 1
    - example.com
  parentRefs: 2
    - group: gateway.networking.k8s.io
      kind: Gateway
      name: demo
      sectionName: https
  rules: 3
    - backendRefs: 4
        - group: ''
          kind: Service
          name: echo-resty
          namespace: demo-space
          port: 80
          weight: 100
      filters: [] 5
      matches: 6
        - path:
            type: Exact
            value: /a
      timeouts: 7
        request: '30s'
        backendRequest: '10s'
      retry: 8
        codes:
          - 503
        attempts: 3
        backoff: '100ms'
      sessionPersistence: 9
        type: Cookie
        sessionName: a
---
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: TCPRoute
metadata:
  name: tcp
  namespace: demo-space
```

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

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

1 Hostnames

2 Publish to listener

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

Справочник по полям маршрута

Каждый маршрут — это CR, определённый спецификацией `GatewayAPI`. Для подробной информации о полях и параметрах конфигурации каждого типа маршрута, пожалуйста, обратитесь к официальной документации:

- Спецификация HTTPRoute ↗
- Спецификация TCPRoute ↗
- Спецификация UDPRoute ↗
- Спецификация GRPCRoute ↗
- Спецификация TLSRoute ↗

Publish to Listener

В Web Console

В веб-консоли вы можете выбрать несколько слушателей для публикации маршрута. Доступные кандидаты на слушателей фильтруются по следующим критериям:

- **Права пользователя**: у вас должен быть доступ к пространству имён gateway (проект должен включать это пространство имён).
- **Белый список пространств имён маршрутов**: разрешённые пространства имён маршрутов слушателя gateway должны включать пространство имён маршрута.
- **Совпадение типа маршрута**: тип маршрута (HTTPRoute, GRPCRoute и т.д.) должен соответствовать разрешённым типам маршрутов слушателя.

Для более сложных сценариев с кросс-пространствами имён смотрите прикрепление к gateway, созданному в другом пространстве имён.

В YAML

- `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 пространстве имён.

Для прикрепления к gateway в другом пространстве имён смотрите прикрепление к gateway, созданному в другом пространстве имён.

Backend

Определяет целевой сервис(ы), куда должны перенаправляться совпадающие запросы.

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

Hostnames

Поле `hostnames` поддерживается `HTTPRoute`, `GRPCRoute` и `TLSRoute`. `TCPRoute` и `UDPRoute` это поле не используют.

`hostnames` — массив строк. Он следует правилам пересечения hostnames.

Rules

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

Matches

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

Правило может иметь несколько совпадений:

- Каждое совпадение состоит из нескольких условий (например, путь, заголовки, параметры запроса, метод)
- Условия **внутри** одного совпадения связаны логикой **И** (все должны быть выполнены)
- Совпадения **между собой** связаны логикой **ИЛИ** (достаточно, чтобы выполнилось любое совпадение)

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

Структура совпадения общая для всех типов маршрутов, но поддерживаемые условия совпадения зависят от типа маршрута. Например, `HTTPRoute` и `GRPCRoute` поддерживают разные наборы условий.

Filters

Определяет преобразования или модификации, применяемые к запросам или ответам.

Концепция фильтров общая для всех типов маршрутов, но поддерживаемые типы фильтров зависят от типа маршрута.

Справочник HTTPRoute

Для `HTTPRoute` используются следующие типы условий совпадения, фильтров и расширенных опций.

Типы условий совпадения

Объект	Метод	Типы значений	Описание	Требования к значению
Path				
	`Exact`	путь (строка)	Совпадение с URL-путём точно и с учётом регистра. Это значит, что	Должен начинаться с без последовательн `//`.

Объект	Метод	Типы значений	Описание	Требования к значению
			точное совпадение пути `/abc` сработает только для `/abc`, **НЕ** для `/abc/`, `/Abc` или `/abcd`.	
	`PathPrefix`	путь (строка)	Совпадение по префиксу URL-пути, разделённому `/`. Совпадение чувствительно к регистру и происходит **поэлементно по пути**. Например, пути `/abc`, `/abc/` и `/abc/def` совпадут с префиксом `/abc`, а `/abcd` — нет.	Должен начинаться с `/` без последовательных `//`.
	`RegularExpression`	путь (строка)	Регулярное выражение: движок RE2.	например, `/api/v1/.*`.
Header				
	`Exact`	имя (ключ заголовка) + значение	Точное совпадение	

Объект	Метод	Типы значений	Описание	Требования к значению
			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.	
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 ↗

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 (строка)	Добавляет заголовок к 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

Тип	Метод	Типы значений	Описание
	`Remove`	[]string	Удаляет указанные заголовки из ответа (без учёт регистра)
RequestRedirect	`Scheme`	строка	Схема для заголовка Location (http/https)
	`Hostname`	PreciseHostname	Хост для заголовка Location
	`ReplaceFullPath`	строка	Заменить весь путь запроса
	`ReplacePrefixMatch`	строка	Заменить совпадающий префикс пути
	`Port`	PortNumber	Порт для заголовка Location
	`StatusCode`	int	HTTP код перенаправлен
URLRewrite	`Hostname`	PreciseHostname	Хост для переписывания запросе

Тип	Метод	Типы значений	Описание
	`ReplaceFullPath`	строка	Заменить весь путь запроса
	`ReplacePrefixMatch`	строка	Заменить совпадающий префикс пути
CORS	`AllowOrigins`	[]string	Список разрешённых источников для CORS-запросов
	`AllowMethods`	[]HTTPMethod	Список разрешённых HTTP-методов
	`AllowHeaders`	[]string	Список разрешённых заголовков в CORS-запросах
	`ExposeHeaders`	[]string	Список заголовков, доступных клиенту в ответ
	`MaxAge`	Duration	Время кэширования для CORS preflight-ответа
	`AllowCredentials`	bool	Разрешены ли учётные данные в CORS-запрос

Примечания:

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

Options

Раздел Options предоставляет расширенные возможности управления трафиком для `HTTPRoute`, включая настройки таймаута, повторных попыток и сохранения сессии.

Таймауты

Поле	Описание	Путь в YAML
Request Timeout	Максимальное время, за которое gateway должен завершить HTTP-ответ	`.spec.rules[].timeouts.request`

Поле	Описание	Путь в YAML
	после получения полного запроса от клиента. Опции: Default (использует таймаут по умолчанию, обычно 15 секунд), Unlimited (устанавливает "0s" для снятия таймаута), Custom.	
Backend Request Timeout	Максимальное время для одного вызова gateway к backend, от начала отправки запроса до получения полного ответа от backend. Опции: Default (использует таймаут по умолчанию), Unlimited (устанавливает "0s"), Custom.	`.spec.rules[].timeouts.backendRequest`

NOTE

- Таймаут Request Timeout начинает отсчёт после полного получения запроса от клиента и охватывает всю транзакцию, которая может включать несколько вызовов backend при повторных попытках.
- Backend Request Timeout должен быть ≤ Request Timeout, если задан.
- При выборе "Default" поле устанавливается в nil (используется таймаут по умолчанию реализации).
- При выборе "Unlimited" поле устанавливается в "0s" (максимально возможное значение).

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

Retry

Поле	Описание	Путь в YAML
Status Codes	HTTP-коды статуса, при которых происходит повтор (например, 503, 502). Диапазон значений: 400-599.	`.spec.rules[].retry.codes`
Attempts	Количество попыток повторов.	`.spec.rules[].retry.attempts`
Backoff	Время ожидания перед повтором (например, "100ms", "1s").	`.spec.rules[].retry.backoff`

NOTE

- По умолчанию повторные попытки **отключены**. Если поле retry не настроено или пустое, gateway не будет повторять неудачные запросы.
- Для включения повторных попыток необходимо явно настроить и количество попыток, и условия повторов.
- При настройке retry в веб-консоли, если удалить все элементы конфигурации retry, поле устанавливается в nil.

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

Сохранение сессии

Настраивает параметры сохранения сессии, чтобы запросы от одного клиента направлялись на один и тот же backend.

Поле	Описание	Путь в YAML
Type	Тип сохранения сессии. Опции: Cookie, Header.	`.spec.rules[].sessionPersistence.type`

Поле	Описание	Путь в YAML
Session Name	Имя cookie или заголовка, используемого для отслеживания сессии.	`.spec.rules[].sessionPersistence.sessionName`

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

Справочник совпадений и фильтров GRPCRoute

Для `GRPCRoute` используются следующие типы совпадений и фильтров.

Совпадения GRPCRoute

`GRPCRoute` поддерживает следующие типы совпадений:

Объект	Метод	Типы значений	Описание
Method	-	тип (service/method) + значение	Совпадение gRPC метода. Тип может быть `service` (совпадение по имени сервиса) или `method` (по имени метода).
Headers	`Exact`	имя (ключ заголовка) + значение	Точное совпадение значения заголовка.
	`RegularExpression`	имя (ключ заголовка) + значение	Регулярное выражение: движок RE2.

Фильтры GRPCRoute

`GRPCRoute` поддерживает только фильтр `RequestHeaderModifier`:

Тип	Метод	Типы значений	Описание	Требования к значению
RequestHeaderModifier	`Set`	имя (строка) + значение (строка)	Перезаписывает заголовок запроса с указанным именем и значением	Макс 16 элементов, значение: 1-4096 символов
	`Add`	имя (строка) + значение (строка)	Добавляет заголовок к запросу, дописывая к существующим значениям	Макс 16 элементов, значение: 1-4096 символов
	`Remove`	[]string	Удаляет указанные заголовки из запроса (без учёта регистра)	Макс 16 элементов

NOTE

`GRPCRoute` не поддерживает опции, такие как таймаут, повторные попытки или сохранение сессии.

Справочник TLSRoute

Следующее поведение характерно для `TLSRoute`.

NOTE

- Для `TLSRoute` hostnames необязательны. Если у слушателя есть hostname, а у `TLSRoute` нет, `TLSRoute` автоматически наследует hostname слушателя.

- `TLSRoute` может быть прикреплен только к слушателям с протоколом `TLS` в режиме `Passthrough`.

Просмотр

Топология

Следующие функции предоставляют дополнительные возможности просмотра, доступные в ACP Web Console.

Вкладка топологии предоставляет визуальное представление маршрута и связанных с ним ресурсов. Она отображает все политики, прикреплённые к маршруту, а также их зависимые ресурсы, такие как секреты, на которые ссылается `SecurityPolicy`.

Эта функция в настоящее время доступна только для `HTTPRoute`.

Следующий шаг

После прикрепления маршрутов к слушателям продолжайте с Настройка GatewayAPI Policy, если вам нужны расширенные политики трафика или безопасности. Для дополнительных примеров эксплуатации смотрите Задачи для Envoy Gateway.

Связанные задачи

- Как прикрепиться к слушателю, созданному в другом пространстве имён

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 следит за всеми Frontends и Rules, прикрепленными к этому ALB, и маршрутизирует запросы на бэкенды на основе этих конфигураций.

Предварительные требования

Высокая доступность **Load Balancer** требует VIP. Пожалуйста, ознакомьтесь с разделом 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	можно 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 смогут найти и настроить этот ALB.
2. Этот ALB будет обрабатывать ingress ресурсы, принадлежащие этому проекту. См. ingress-sync.
3. В веб-интерфейсе правила, созданные в проекте X, не будут видны и не смогут быть настроены в проекте Y.

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

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

Настройка дополнительных параметров

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

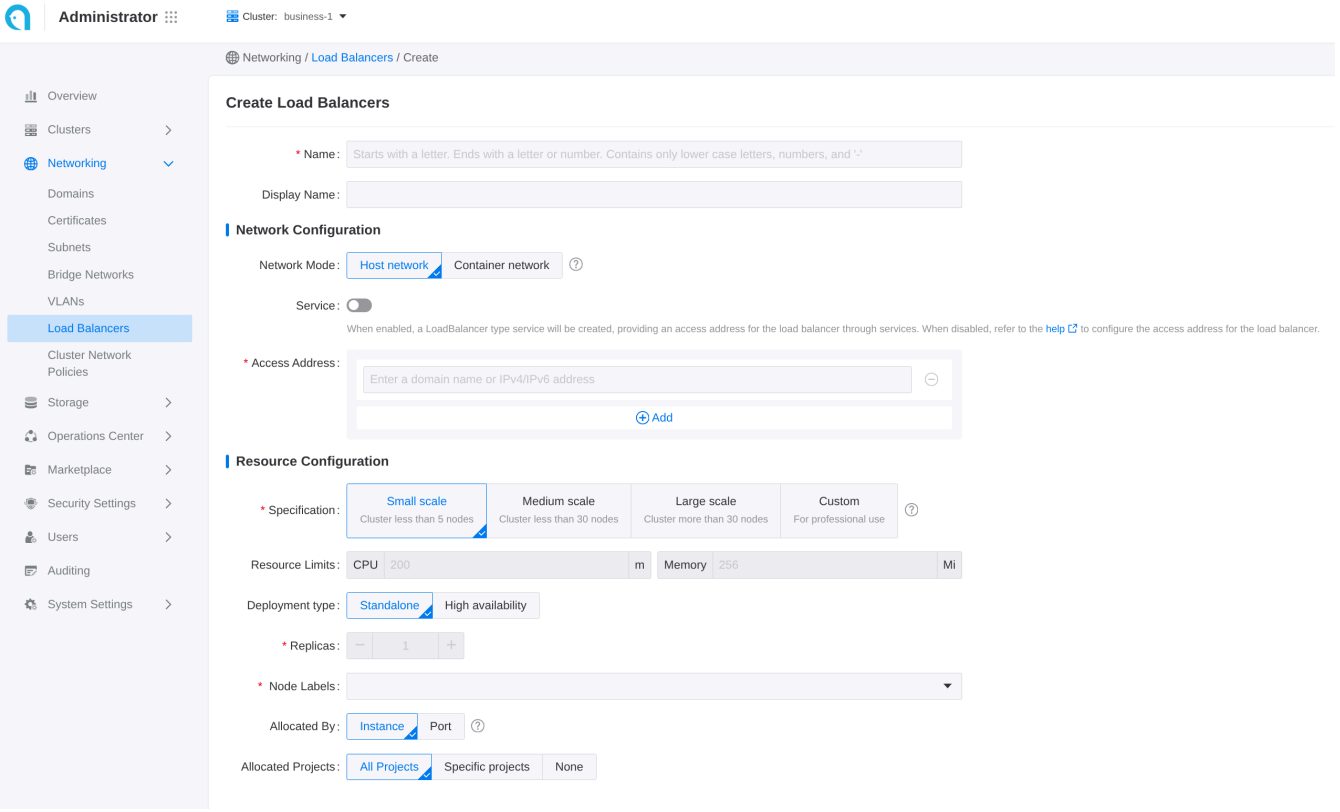
- Привязка NIC

- 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 обслуживания на 3–5 минут.
>
> Пожалуйста, выбирайте подходящее время для этой операции!

1. Войдите в **Administrator**.
2. В левой навигационной панели нажмите **Network Management** > **Load Balancer**.
3. Нажмите ⋮ > **Update**.
4. Обновите сетевые и ресурсные настройки по необходимости.
   - Пожалуйста, задавайте спецификации разумно в соответствии с бизнес-требованиями. Также можно обратиться к соответствующему руководству Как правильно распределять 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, а для настройки backend-сервиса используйте rule.

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

Предварительные требования

Сначала создайте ALB.

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; для протокола gRPC добавление сертификата необязательно.
- При выборе протокола gRPC backendProtocol по умолчанию gRPC, который не поддерживает сохранение сессии. Если для gRPC протокола задан сертификат, балансировщик нагрузки снимет сертификат gRPC и перенаправит незашифрованный трафик gRPC на backend-сервис.
- Если используется кластер Google GKE, балансировщик нагрузки одного **типа контейнерной сети** не может одновременно иметь протоколы слушателей TCP и UDP.

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

6 backendProtocol: протокол, используемый backend-сервисом.

7 По умолчанию `serviceGroup`:

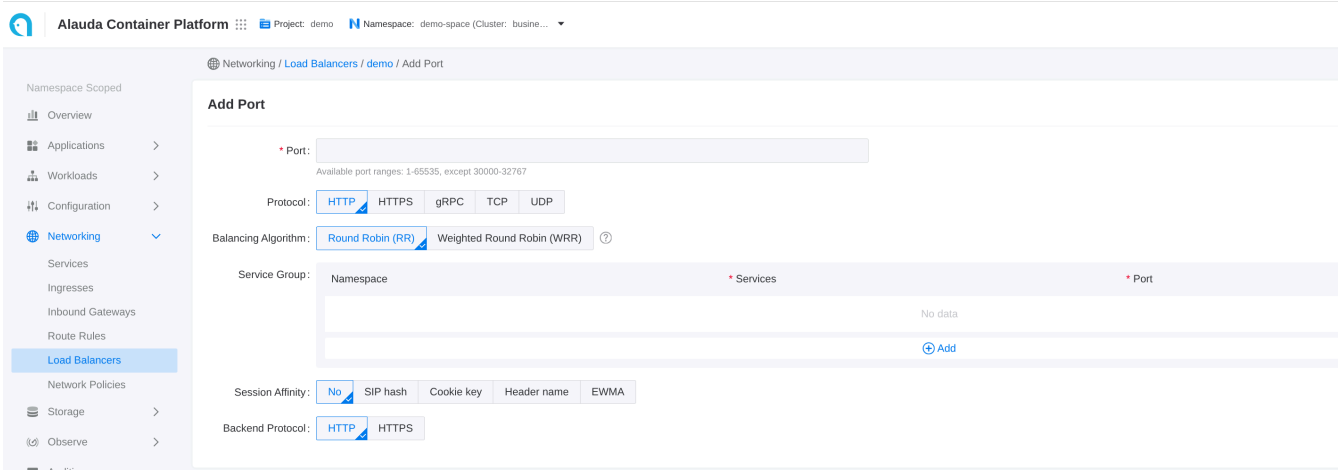
- L4 proxy: обязательно. ALB напрямую перенаправляет трафик в группу сервисов по умолчанию.
- L7 proxy: необязательно. ALB сначала сопоставляет правила на этом Frontend; если совпадений нет, используется группа сервисов по умолчанию.

8 session_affinity_policy

Операции с Frontend

Создание

Через веб-консоль



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

Каждое поле ввода в веб-интерфейсе соответствует полю CR

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

Через CLI

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

Последующие действия

Для трафика с портов HTTP, gRPC и HTTPS, помимо группы внутреннего роутинга по умолчанию, можно настроить более разнообразные правила сопоставления backend-сервисов rules. Балансировщик нагрузки сначала сопоставит соответствующий backend-сервис согласно заданным правилам; если совпадение не найдено, будет использован backend-сервис из упомянутой внутренней группы роутинга.

Связанные операции

Вы можете нажать на значок ⋮ справа на странице списка или нажать **Actions** в правом верхнем углу страницы деталей для обновления маршрута по умолчанию или удаления порта слушателя по необходимости.

> **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 понимания принципа использования 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 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 — целое число от 0 до 10, где меньшее значение означает более высокий приоритет. Для настройки приоритета правила в ingress можно использовать следующую аннотацию:

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

Для правил приоритет задается напрямую в `.spec.priority` целым числом.

Действия

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

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	Перезапись 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 требуется TLS-перенаправление, можно настроить HTTPS.

Группа сервисов и политика сохранения сессии

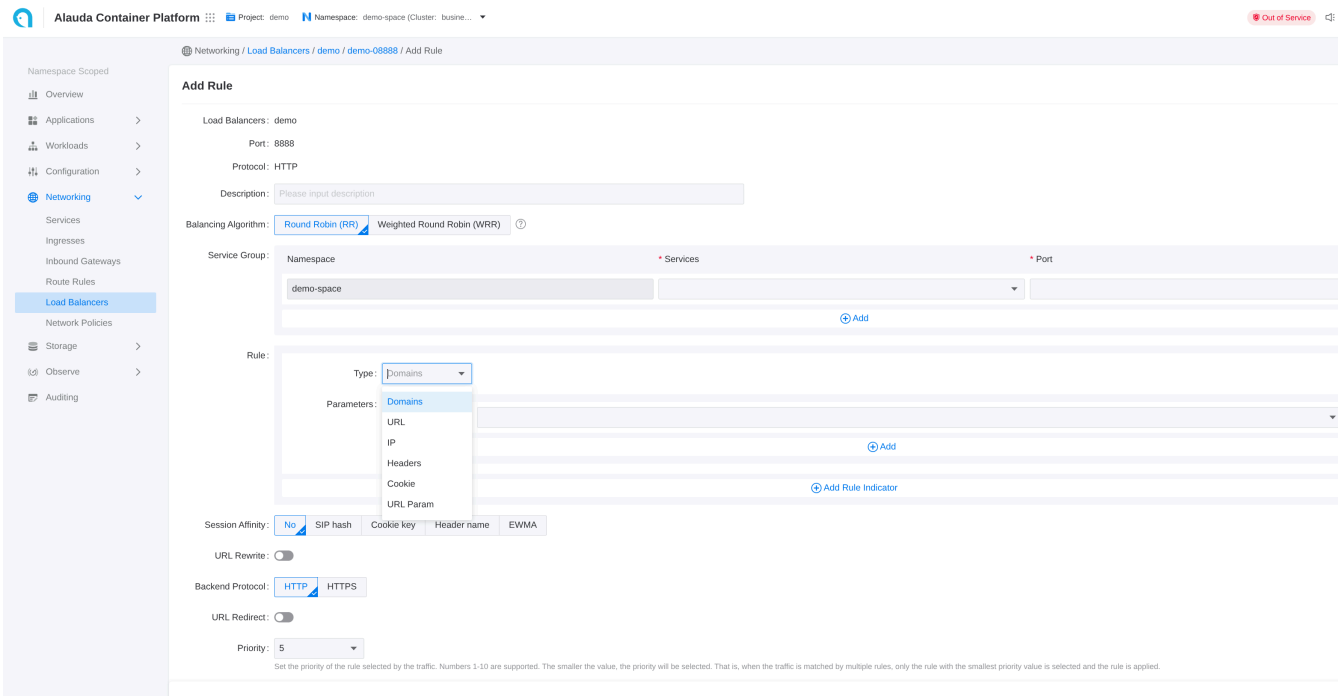
В правиле можно настроить один или несколько сервисов.

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

Подробнее см. Алгоритм балансировки.

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 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.

Сертификат можно указать либо в правиле, либо в ingress для сопоставления сертификата с конкретным портом.

Поддерживается терминация, а также повторное шифрование, если backend протокол HTTPS. Однако **нельзя** указывать сертификат для связи с backend-сервисом.

Аннотация сертификата в Ingress

Сертификаты могут ссылаться на секреты в разных 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 ft 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 ft 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 пределах одного проекта.

Если у ingress namespace есть метка вида `cpaas.io/project: demo`, это означает, что ingress принадлежит проекту `demo`.

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

> **NOTE**
>
> ALB слушает ingress и автоматически создает `Frontend` или Rule. Поле `source` определено следующим образом:
>
> 1. `spec.source.type` в настоящее время поддерживает только `ingress`.
> 2. `spec.source.name` — имя ingress.
> 3. `spec.source.namespace` — namespace 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

Сочетая логи и данные мониторинга, вы можете быстро выявлять и устранять проблемы балансировщика 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** просматривайте 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** просматривайте информацию о тенденциях метрик балансировщика нагрузки с точки зрения узла.
   - **Usage Rate**: текущее использование CPU и памяти балансировщиком 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 DNS-запросов и повышает стабильность кластера, кэшируя DNS-ответы локально на каждом узле и 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'ов**: После успешного развертывания плагина он не повлияет на уже запущенные Pod'ы, а вступит в силу только для вновь созданных Pod'ов. При использовании CNI Kube-OVN необходимо вручную добавить параметр "--node-local-dns-ip=(IP-адрес локального DNS-кэша)" в kube-ovn-controller.
4. **Настройка NetworkPolicy**: Если в кластере настроен NetworkPolicy, необходимо дополнительно разрешить трафик в обоих направлениях для node CIDR и nodeLocalDNSIP в networkPolicy, чтобы обеспечить корректную коммуникацию.
5. **Обновление кластера MicroOS**: Для кластеров MicroOS обновление выполняется путём пересоздания кластера, что приводит к потере изменений конфигурации kubelet. Чтобы сохранить конфигурацию NodeLocal DNS при обновлениях, необходимо добавить параметр `--cluster-dns` в `kubeletExtraArgs` в следующих трёх местах шаблона кластера:
   - `KubeadmControlPlane` → `initConfiguration` → `nodeRegistration` → `kubeletExtraArgs`
   - `KubeadmControlPlane` → `joinConfiguration` → `nodeRegistration` → `kubeletExtraArgs`
   - `KubeadmConfigTemplate` → `template` → `spec` → `joinConfiguration` → `nodeRegistration` → `kubeletExtraArgs`

Добавьте следующий параметр в каждый из указанных разделов `kubeletExtraArgs`:

```
cluster-dns: "<NodeLocal_DNS_IP>"  # например, 169.254.20.10
```

WARNING

Примечания по обновлению 4.2.x

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

До обновления:

- 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 до 4.3+ ручные действия не требуются.

Installation

Установка через Marketplace

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

Параметр	Описание	Пример значения
IP	IP-адрес локального DNS-кэша на узле. Для IPv4 рекомендуется использовать адрес из диапазона 169.254.0.0/16, предпочтительно 169.254.20.10. Для IPv6 рекомендуется использовать адрес из диапазона fd00::/8, предпочтительно fd00::10.	169.254.20.10

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

How It Works

Архитектура

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

Configuration

Настройка Network Policy

> **Важно**: Если в вашем кластере включён NetworkPolicy, необходимо настроить соответствующие правила, разрешающие DNS-трафик к NodeLocal DNSCache. Без этих правил Pod'ы 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 Envoy Gateway

Задачи для Envoy Gateway

Обзор

Предварительные требования

Продвинутые задачи

Связанная документация

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

Конфигура

Overview

Prerequisites

Шаги конфигур

Пример конфигурации

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

Установка Multus CNI

Создание подсетей

Создание Pod с несколькими сетевыми интерфейсами

Проверка создания двух сетевых интерфейсов

Дополнительные возможности

Настройка IPPool

Инструкции

Меры предосторожности

Автоматическое взаимное подключение подсетей Underlay и Overlay

weight: 13

Процедура

Изоляция между подсетями Underlay с включённым u2oInterconnection

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

Задача: Миграция с OCP Route на GatewayAPI Route

Задача: Миграция с OCP Route на GatewayAPI Route

Введение

Предварительные требования

Базовый HTTP Route

Таймауты маршрута

HTTP Strict Transport Security (HSTS)

Сессионная аффинность на основе cookie

Задачи для Ingress-Nginx

Содержание

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 ↗

действие	ссылка
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

используйте следующий yaml для развертывания ingress-nginx с сертификатом по умолчанию

```yaml
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo
spec:
  controller:
    extraArgs:
      default-ssl-certificate: $DEFAULT_CERT_NAMESPACE/$DEFAULT_CERT_NAME
```

пожалуйста, обратитесь к default-ssl-certificate ↗

Добавление аннотации Pod в IngressNginx

Добавление аннотации pod

Сохранение исходного IP

Когда трафик проходит 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 для каждого 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, который содержит 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 парсит 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 ошибкам соединения)

Как настроить

Настройте ваш HAProxy для отправки заголовков PROXY protocol, затем разверните ingress-nginx с включенной 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 пересылки трафика без обработки TLS-сертификатов. Поскольку PROXY protocol работает на уровне TCP, вы

можете позволить Ingress-Nginx обрабатывать HTTPS-терминацию и управление сертификатами напрямую, 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 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:

```
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
MetalLB
      ingress-nginx: "true"
```

Важно: `nodeSelector` должен совпадать с узлами в конфигурации пула адресов MetalLB, чтобы гарантировать, что поды Ingress-Nginx развернуты на узлах, которые могут принимать трафик от MetalLB.

Для подробностей смотрите документацию externalTrafficPolicy↗.

Задачи для Envoy Gateway

Содержание

Обзор

Этот документ расширяет основной путь конфигурации (operator → gateway → route → policy) и вводит продвинутые задачи, выходящие за рамки стандартных конфигураций ресурсов, рассмотренных в предыдущих документах.

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

1. **Изменение стандартных ресурсов Gateway API**: прямое редактирование `Gateway`, `HTTPRoute`, `TCPRoute`, `UDPRoute` и других основных ресурсов.
2. **Присоединение политик через PolicyAttachment**: использование ресурсов `SecurityPolicy`, `ClientTrafficPolicy`, `BackendTrafficPolicy` и других политик для расширения поведения Gateway и Route.
3. **Настройка глобальных параметров**: изменение `EnvoyGatewayCtl` для изменения поведения экземпляра `envoy-gateway` или других глобальных настроек, влияющих на все шлюзы.

Этот документ фокусируется на продвинутых задачах и специальных сценариях, которые не входят в основной путь конфигурации, включая маршрутизацию между пространствами имён, настройку наблюдаемости, кастомизацию развертывания и устранение неполадок.

Предварительные требования

1. Настройка EnvoyGatewayCtl
2. Настройка Gateway
3. Настройка Route
4. Настройка GatewayAPI Policy

Продвинутые задачи

OpenTelemetry (OTel)

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

Как присоединиться к Listener, созданному в другом Namespace

В конфигурации listener Gateway необходимо указать, какие пространства имён разрешено присоединять к нему маршруты.

```yaml
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 в том же namespace
    - name: http-82
      protocol: HTTP
      port: 82
      allowedRoutes:
        namespaces:
          from: Selector
          selector:
            matchLabels:
              team: frontend # разрешены маршруты только в namespace с ме
ткой team=frontend
```

Подробности см. в Cross-Namespace routing ↗.

Как использовать сертификат, созданный в другом Namespace

Чтобы использовать сертификат, созданный в другом namespace, создайте `ReferenceGrant` в namespace, где хранится сертификат. Пожалуйста, следуйте инструкциям в cross-namespace-certificate-references ↗ и referencegrant ↗.

> **NOTE**
>
> Вы не можете указывать отдельные ресурсы `secret`; необходимо разрешить весь namespace целиком.

Как использовать SSL Passthrough

Пожалуйста, следуйте инструкциям в

- tls ↗
- tls-passthrough ↗

Как изменить минимальную версию TLS

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
```

Поле `.spec.tls` в `ClientTrafficPolicy` соответствует clienttlssettings↗. Если необходимо настроить наборы шифров (cipher suites) помимо минимальной версии TLS, обратитесь к тому же upstream заданию и API-справочнику.

Как указать NodePort при использовании NodePort Service

При использовании NodePort сервиса Kubernetes назначает значение NodePort для каждого порта сервиса. При доступе к сервису через IP узла используйте назначенный NodePort вместо порта сервиса.

Есть два подхода:

Ручное получение назначения NodePort, следуя 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 listener Gateway и NodePort совпадали, порт listener также должен быть в диапазоне NodePort.

Как указать VIP при использовании MetalLB

При использовании MetalLB в качестве провайдера LoadBalancer вы можете указать статический VIP для сервиса Gateway через аннотации сервиса.

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: demo
  namespace: demo
spec:
  provider:
    type: Kubernetes
    kubernetes:
      envoyService:
        type: LoadBalancer
        annotations: (1)
          metallb.universe.tf/address-pool: production (2)
          metallb.universe.tf/loadBalancerIPs: VIP_IP (3)
```

(1) Добавьте аннотации MetalLB в поле `envoyService.annotations`

(2) Укажите имя пула адресов для выделения IP

(3) Или укажите конкретный IP-адрес (должен быть в диапазоне пула адресов)

Доступные аннотации:

Аннотация	Описание
`metallb.universe.tf/address-pool`	Выбор пула адресов для выделения IP
`metallb.universe.tf/loadBalancerIPs`	Указание конкретного IP-адреса (поддерживается несколько IP через запятую)

NOTE

- Указанный IP должен находиться в сконфигурированном пуле адресов MetalLB
- Убедитесь, что MetalLB корректно установлен и настроен перед указанием VIP
- Для настройки MetalLB смотрите Configure MetalLB

Как добавить аннотации Pod в Envoy Gateway

Добавление аннотаций pod

Как задать NodeSelector и Tolerations для `envoy-gateway-operator`

Обновите ресурс `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 ресурс `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 ресурс `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"
              }
            ]
          }
        }
      }
    }
  }
}'
```

Как использовать `hostNetwork` в `envoy-proxy`

Использование `hostNetwork: true` позволяет pod-ам Envoy proxy использовать сетевое пространство имён хоста напрямую. Это может быть полезно для:

- достижения лучшей сетевой производительности
- доступа к шлюзу напрямую через IP узла

Особенности:

- Pod-ы с hostNetwork будут привязываться напрямую к сетевым интерфейсам хоста
- Возможны конфликты портов, если несколько pod-ов пытаются использовать один и тот же порт на одном узле

- Снижается изоляция безопасности, так как pod-ы разделяют сетевое пространство имён хоста
- Рекомендуется использовать правила `nodeSelector` или `affinity` для контроля размещения pod-ов и избежания конфликтов портов

Существует два подхода к настройке `hostNetwork`, в зависимости от того, хотите ли вы использовать привилегированные порты (< 1024) напрямую:

Подход 1: Использование смещения портов (по умолчанию, рекомендуется)

Это подход по умолчанию и рекомендуется. Envoy Gateway автоматически добавляет смещение 10000 к привилегированным портам, чтобы избежать необходимости специальных разрешений.

Плюсы:

- Не требуются специальные разрешения или capabilities
- Более безопасно, так как запускается от имени непривилегированного пользователя без дополнительных прав
- Проще в настройке
- Работает из коробки

Минусы:

- Клиенты должны использовать порты со смещением (10080, 10443)

Конфигурация:

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: demo
  namespace: demo
spec:
  provider:
    type: Kubernetes
    kubernetes:
      envoyDeployment:
        patch:
          type: StrategicMerge
          value:
            spec:
              template:
                spec:
                  hostNetwork: true                    # Включить режим h
ostNetwork
                  dnsPolicy: ClusterFirstWithHostNet   # Необходимо для к
орректного разрешения DNS
        pod:
          nodeSelector:                                # Рекомендуется: к
онтролировать размещение pod для избежания конфликтов
            kubernetes.io/hostname: "demo"
```

Доступ:

- Порт 80 → доступ через `http://<node-ip>:10080`
- Порт 443 → доступ через `https://<node-ip>:10443`

Подход 2: Использование привилегированных портов напрямую с useListenerPortAsContainerPort

Этот подход позволяет Envoy привязываться напрямую к привилегированным портам (< 1024), таким как 80 и 443.

Плюсы:

- Можно использовать стандартные порты (80 и 443) напрямую
- Лучшая совместимость с клиентами, ожидающими стандартные порты

Минусы:

- Требуется capability NET_BIND_SERVICE
- Немного снижена безопасность по сравнению с подходом со смещением портов
- Более сложная конфигурация

Конфигурация:

```
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: demo
  namespace: demo
spec:
  provider:
    type: Kubernetes
    kubernetes:
      useListenerPortAsContainerPort: true  # Отключить смещение портов
      envoyDeployment:
        patch:
          type: StrategicMerge
          value:
            spec:
              template:
                spec:
                  hostNetwork: true
                  dnsPolicy: ClusterFirstWithHostNet
                  containers:
                  - name: envoy
                    command:
                      - /usr/local/bin/envoy-with-cap  # использовать env
oy с filecap
                    securityContext:
                      capabilities:
                        add:
                        - NET_BIND_SERVICE  # Требуется для привязки к пр
ивилегированным портам
        pod:
          nodeSelector:                                # Рекомендуется: к
онтролировать размещение pod для избежания конфликтов
            kubernetes.io/hostname: "demo"
```

Доступ:

- Порт 80 → доступ через `http://<node-ip>:80`
- Порт 443 → доступ через `https://<node-ip>:443`

Как получить доступ к LoadBalancer VIP изнутри кластера

По умолчанию Envoy Gateway создаёт сервисы LoadBalancer с `externalTrafficPolicy: Local`. Эта политика сохраняет исходный IP клиента, но имеет важное ограничение: запросы с узлов кластера без pod-ов Envoy Gateway будут неуспешны, так как трафик не перенаправляется на другие узлы.

Решение 1: Использовать ClusterIP сервиса (рекомендуется для доступа внутри кластера)

Для приложений, работающих внутри кластера, используйте ClusterIP сервиса вместо VIP LoadBalancer. Это полностью избегает ограничения маршрутизации.

Решение 2: Изменить политику внешнего трафика на Cluster

Если необходимо получить доступ к VIP LoadBalancer с любого узла кластера, измените `externalTrafficPolicy` на `Cluster`:

```
kubectl patch envoyproxy $GATEWAY_NAME -n $GATEWAY_NS --type='json' -p='[
  {"op": "replace", "path": "/spec/provider/kubernetes/envoyService/exter
nalTrafficPolicy", "value": "Cluster"}
]'
```

Связанная документация

- Настройка EnvoyGatewayCtl
- Настройка Gateway
- Настройка Route
- Настройка GatewayAPI Policy

Дополнительная конфигурация

Пожалуйста, смотрите 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 и стабильную работу 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 передачи 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 узла внешнего LB):
   - `ipvsadm`

- `container-runtime, например containerd`

3. Убедитесь, что `container-runtime` настроен на автоматический запуск при загрузке каждого хоста.
4. Убедитесь, что часы каждого узла-хоста синхронизированы.
5. Подготовьте образ Keepalived, используемый для запуска службы внешнего LB; платформа уже содержит этот образ. Адрес образа имеет следующий формат: `<image repository address>/tkestack/keepalived:<version suffix>`. Суффикс версии может незначительно отличаться в разных версиях. Получить адрес репозитория образов и суффикс версии можно следующим образом. В этом документе используется пример `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 узле внешнего 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. Отредактируйте 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 этого экземпляра, может содержать т
олько 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 интерфейса будут удалены.
      kube_lock:
        kubecfgs: # Список kube-config, используемый kube-lock для посл
едовательной попытки лидерства в Keepalived
          - "/live/cfg/kubecfg/kubecfg01.conf"
          - "/live/cfg/kubecfg/kubecfg02.conf"
          - "/live/cfg/kubecfg/kubecfg03.conf"
    ipvs: # Конфигурация для опции IPVS
      ips: [ "192.168.143.192", "192.168.138.100","192.168.129.100" ] #
IPVS backend, замените IP узлов k8s master на IP узлов ALB
      ports: # Настройка логики проверки здоровья для каждого порта VIP
        - port: 80 # Порт виртуального сервера должен совпадать с 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 лидерства 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 узлах внешнего 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 узел внешнего 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, применимые 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 в течение 20 секунд.
3. Используйте команду `curl` на узле, не являющемся LB, чтобы проверить, нормальна ли связь с VIP.

```
curl 192.168.128.118

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

<p>For online documentation and support please refer to <a href="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

Конфигура

- Overview
- Prerequisites
- Шаги конфигур

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

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

Настройка IPPool

- Инструкции

шлю

Меры предосторожности

Настройка

Автоматическое взаимное подключение подсетей Underlay и Overlay

MT

MTU

weight: 13

Процедура

Изоляция между подсетями Underlay с включённым u2oInterconnection

Понимание Kube-OVN CNI

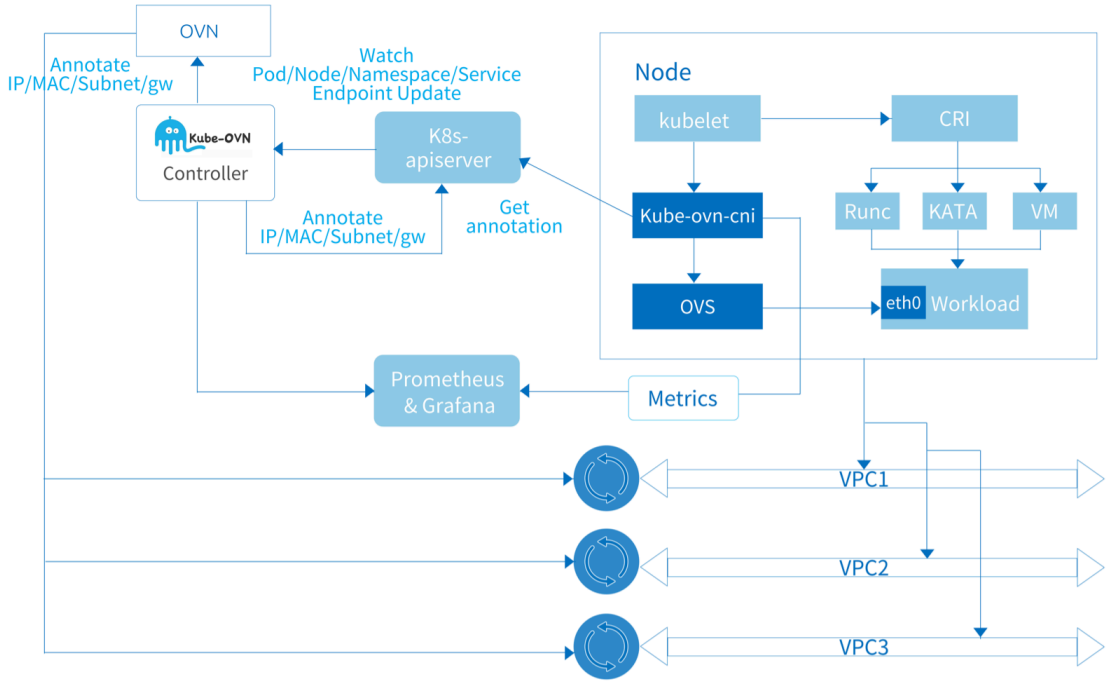
В этом документе описывается общая архитектура Kube-OVN, функциональность каждого компонента и их взаимодействие.

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.

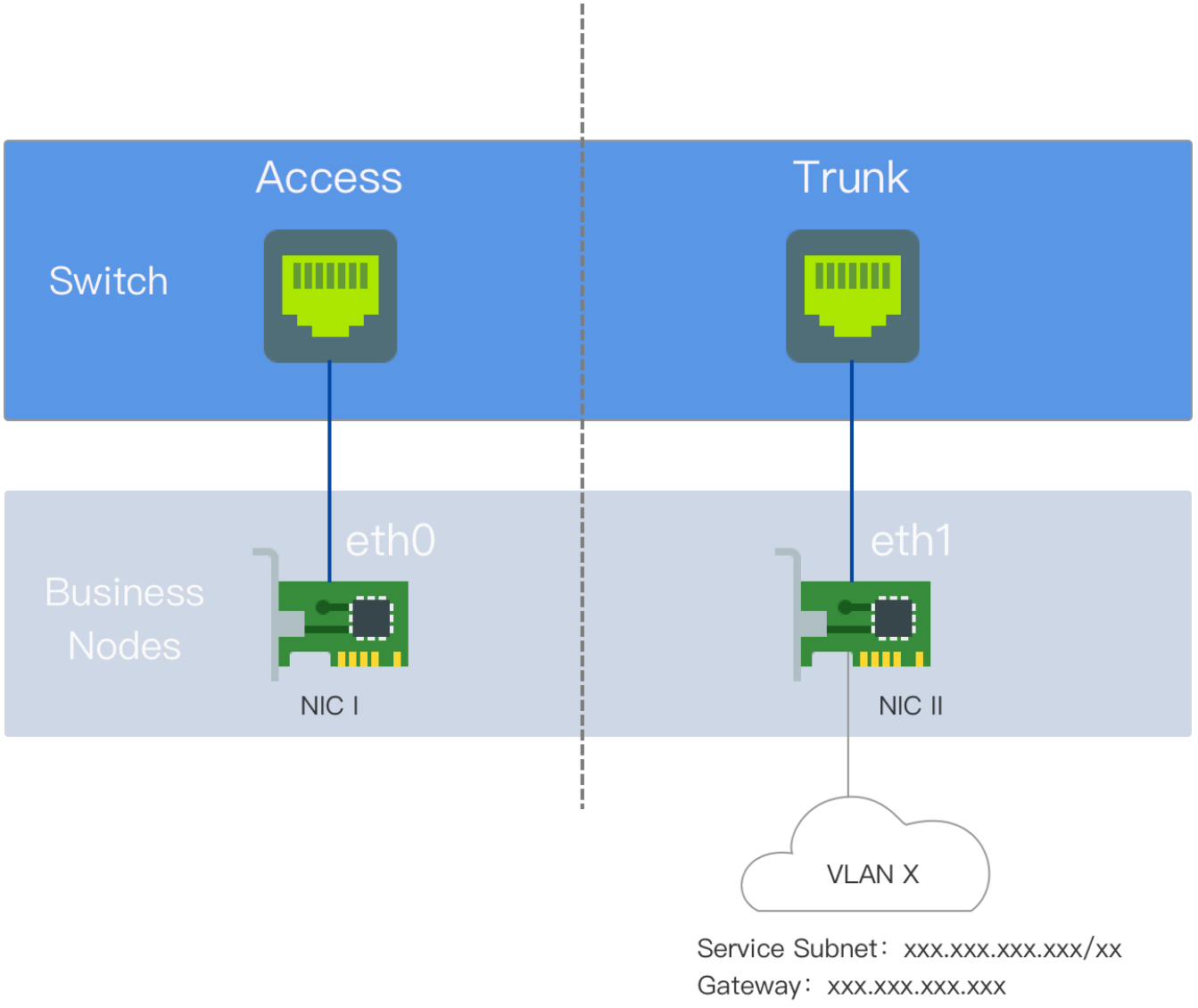
Содержание

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

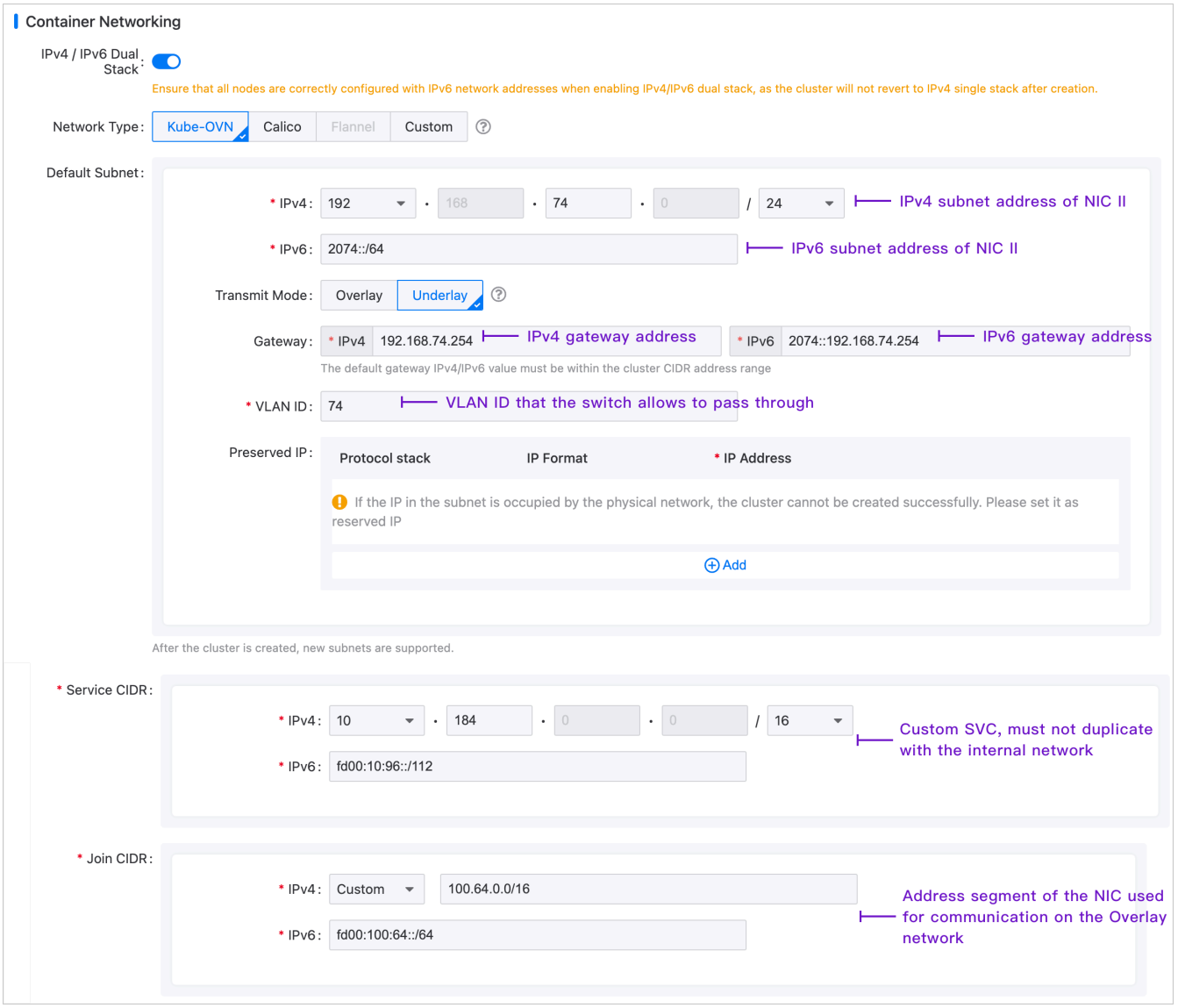
Проверьте, может ли NIC Два связаться с адресом шлюза:

```
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

Конфигурация сервиса LoadBalancer Kube-OVN Underlay + MetalLB

Содержание

Overview

Это решение предназначено для интеграции режима L2 MetalLB с сетью Kube-OVN Underlay. Оно позволяет использовать IP-адреса подсети Underlay в качестве VIP

сервиса LoadBalancer MetalLB, напрямую перенаправляя трафик на backend-бизнес Pod'ы.

> ⚠️ **Критично**: VIP LoadBalancer и IP-адреса backend Pod'ов **должны находиться в одной подсети Underlay**.

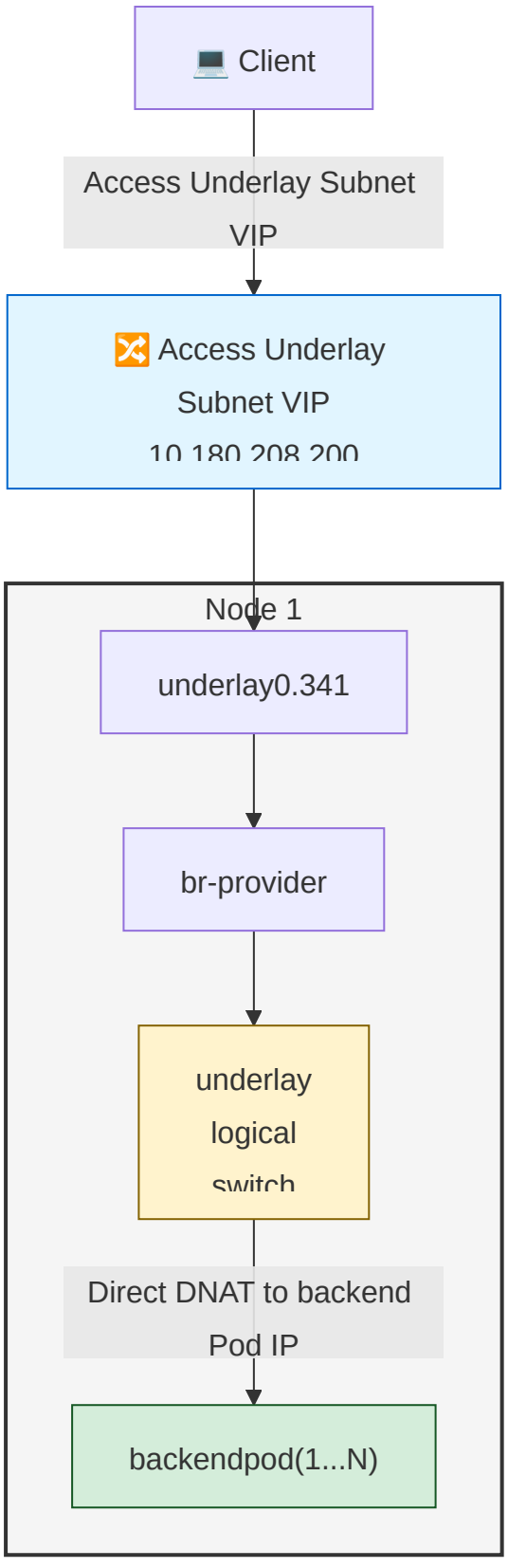
Prerequisites

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 субинтерфейсы (например, underlay0.341), если существует только родительский интерфейс (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
```

> ⚠️ **Предупреждение**: При отдельном изменении ресурсов `ProviderNetwork` или `Vlan` сетевое подключение Underlay будет прервано. Сеть восстановится только после полной настройки и синхронизации обоих ресурсов. Планируйте изменения конфигурации в окна обслуживания, чтобы минимизировать прерывание 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 **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 для резервирования диапазона IP-адресов 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 сервиса LB
  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: установите `spec.loadBalancerIP: 10.180.208.201`
spec:
  type: LoadBalancer
  externalTrafficPolicy: Local  # **ВАЖНО**: необходимо для сохранения исходного IP и прямой маршрутизации к 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** добавьте 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 подключение (Alpha)

Поддерживается настройка межкластерного подключения между кластерами, сетевой режим которых совпадает с Kube-OVN, чтобы Pods в кластерах могли обращаться друг к другу. Cluster Interconnect Controller — это расширяющий компонент, предоставляемый Kube-OVN; он отвечает за сбор сетевой информации между разными кластерами и соединение сетей нескольких кластеров путем распространения маршрутов.

Содержание

Предварительные требования

- Диапазоны CIDR подсетей разных кластеров не должны пересекаться.
- Должен быть набор машин, доступных по IP из kube-ovn-controller каждого кластера, чтобы развернуть контроллеры межкластерного подключения.
- Для каждого кластера должен существовать набор машин, доступных по IP из kube-ovn-controller, для межкластерного подключения; впоследствии они будут использоваться в качестве gateway-узлов.
- Эта функция доступна только для VPC по умолчанию; пользовательские VPC не могут использовать функцию межкластерного подключения.

Развернут многонодовый контроллер связности Kube-OVN

Доступны три способа развертывания: развертывание через Deploy, развертывание через Podman и развертывание через Containerd.

Развертывание через Deployment

Примечание: Этот способ развертывания поддерживается в платформе версии v3.16.0 и более поздних версиях.

Шаги выполнения

1. Выполните следующую команду на Master-узле кластера, чтобы получить установочный скрипт `install-ic-server.sh` из Pod `kube-ovn-controller`.

   ```
   kubectl -n kube-system cp $(kubectl get pods -n kube-system -l app=kube
   -ovn-controller -o custom-columns=NAME:.metadata.name --no-headers | he
   ad -1):/kube-ovn/install-ic-server.sh ./install-ic-server.sh
   ```

2. Откройте файл скрипта в текущем каталоге и измените параметры следующим образом.

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

Измененные значения параметров должны быть следующими:

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

3. Сохраните файл скрипта и выполните его с помощью следующей команды.

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

Развертывание через Podman и Containerd

1. Выберите **три или более узлов в любом кластере** для развертывания Interconnected Controller. В этом примере подготовлены три узла.
2. Выберите любой узел в качестве Leader и выполните следующие команды в зависимости от способа развертывания.
   Примечание: Перед настройкой проверьте, существует ли каталог ovn в `/etc`. Если нет, создайте его с помощью команды `mkdir /etc/ovn`.

   - **Команды для развертывания через container Примечание**: Выполните команду `podman images | grep ovn`, чтобы получить адрес образа Kube-OVN.

     - Команда для Leader-узла:

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

- Команды для двух других 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 address of the current node>" \   ## For example:
-e LOCAL_IP="192.168.39.24"
-e LEADER_IP="<IP address of the Leader node>" \  ## For example:
-e LEADER_IP="192.168.39.37"
-e NODE_IPS="<IP addresses of all nodes, separated by commas>" \
## For example: -e NODE_IPS="192.168.39.22,192.168.39.24,192.168.3
9.37"
<image repository address> bash start-ic-db.sh   ## For example: 1
92.168.39.10:60080/acp/kube-ovn:v1.8.8  bash start-ic-db.sh
```

- **Команды для развертывания через Containerd**

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

- Команда для Leader-узла:

```
ctr -n k8s.io run \
-d \
--env "ENABLE_OVN_LEADER_CHECK=false" \
--net-host \
--privileged \
--mount="type=bind,src=/etc/ovn/,dst=/etc/ovn,options=rbind:rw" \
--mount="type=bind,src=/var/run/ovn,dst=/var/run/ovn,options=rbin
d:rw" \
--mount="type=bind,src=/var/log/ovn,dst=/var/log/ovn,options=rbin
d:rw" \
--env="NODE_IPS=<IP addresses of all nodes, separated by commas>"
\   ## For example: --env="NODE_IPS="192.168.178.97,192.168.181.9
3,192.168.177.192""
--env="LOCAL_IP=<IP address of the current node>" \   ## For examp
le: --env="LOCAL_IP="192.168.178.97""
<image repository address> ovn-ic-db bash start-ic-db.sh   ## For
example: registry.alauda.cn:60080/acp/kube-ovn:v1.9.25 ovn-ic-db b
ash start-ic-db.sh
```

- Команды для двух других 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 addresses of all nodes, separated by commas>"
\   ## For example: --env="NODE_IPS="192.168.178.97,192.168.181.9
3,192.168.177.192"" \
--env="LOCAL_IP=<IP address of the current node>" \   ## For examp
le: --env="LOCAL_IP="192.168.181.93""
--env="LEADER_IP=<IP address of the Leader node>" \   ## For examp
le: --env="LEADER_IP="192.168.178.97""
<image repository address> ovn-ic-db bash start-ic-db.sh   ## For
example: registry.alauda.cn:60080/acp/kube-ovn:v1.9.25 ovn-ic-db b
ash start-ic-db.sh
```

Развертывание контроллера межкластерного подключения в глобальном кластере

На любом управляющем узле global замените следующие параметры в соответствии с комментариями и выполните следующую команду, чтобы создать ресурс ConfigMap.

Примечание: Для корректной работы не допускается изменять ConfigMap с именем ovn-ic в global. Если требуется изменить какой-либо параметр, сначала удалите ConfigMap и правильно настройте его заново, а затем примените ConfigMap.

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

Присоединение к межкластерному подключению

Добавьте кластер, сетевой режим которого — Kube-OVN, в межкластерное подключение.

Предварительные требования

Созданные подсети, **ovn-default** и **подсети присоединения** в кластере не должны конфликтовать ни с одним сегментом кластера в группе межкластерного подключения.

Порядок выполнения

1. В левом навигационном меню нажмите **Кластеры** > **Кластер кластеров**.
2. Нажмите на имя ***кластера***, который требуется добавить в межкластерное подключение.
3. В правом верхнем углу нажмите **Параметры** > **Межкластерное подключение**.
4. Нажмите **Присоединиться к межкластерному подключению**.
5. Выберите gateway-узел для кластера.
6. Нажмите **Присоединить**.

Связанные операции

Обновление информации о gateway-узле подключенного кластера

Обновите информацию о gateway-узлах кластера, которые присоединились к группе межкластерного подключения.

Порядок выполнения

1. В левом навигационном меню нажмите **Кластеры** > **Кластер кластеров**.
2. Нажмите ***Имя кластера***, для которого требуется обновить информацию о gateway-узле.
3. В правом верхнем углу нажмите **Операции** > **Межкластерное подключение**.
4. Для кластера, информацию о gateway-узле которого нужно обновить, нажмите **Обновить gateway-узел**.
5. Снова выберите gateway-узел для кластера.
6. Нажмите **Обновить**.

Выход из межкластерного подключения

Кластер, который присоединился к группе межкластерного подключения, выходит из межкластерного подключения и при этом отключает Pod кластера от внешнего Pod кластера.

Порядок выполнения

1. В левом навигационном меню нажмите **Кластеры** > **Кластер кластеров**.
2. Нажмите на имя ***кластера***, который требуется вывести из эксплуатации.
3. В правом верхнем углу нажмите **Параметры** > **Межкластерное подключение**.
4. Для кластера, который нужно вывести, нажмите **Выйти из межкластерного подключения**.
5. Корректно введите имя кластера.

6. Нажмите **Выйти**.

Очистка остаточных данных межкластерного подключения

Если кластер был удален без выхода из межкластерного подключения, на контроллере могут остаться некоторые остаточные данные. Если затем попытаться снова использовать эти узлы для создания кластера и присоединения к межкластерному подключению, могут возникнуть сбои. Подробную информацию об ошибке можно посмотреть в журнале контроллера (kube-ovn-controller) `/var/log/ovn/ovn-ic.log`. Некоторые сообщения об ошибках могут включать:

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

Шаги выполнения

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

   - Войдите в контейнер ovn-ic-db и выполните операцию очистки с помощью следующих команд.

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

     Затем выполните одну из следующих команд очистки:

     - Выполните очистку, указав имя исходного кластера. Замените *<cluster-name>* на **имя исходного кластера**:

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

- Выполните очистку, указав имя любого узла исходного кластера. Замените *<node-name>* на **имя любого узла исходного кластера**:

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

Способ 2: Выполнение в Pod ovn-ic-controller

- Войдите в Pod ovn-ic-controller и выполните операцию очистки с помощью следующих команд.

```
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:

- Выполните очистку, указав имя исходного кластера. Замените *<cluster-name>* на **имя исходного кластера**:

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

- Выполните очистку, указав имя любого узла исходного кластера. Замените *<node-name>* на **имя любого узла исходного кластера**:

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

Удаление межкластерного подключения

Примечание: Шаг 1 — Шаг 3 необходимо выполнить на всех **рабочих кластерах, которые присоединились к межкластерному подключению**.

Шаги выполнения

1. Выйдите из межкластерного подключения. Существует два конкретных способа выхода, выберите один в зависимости от ваших требований.

- Удалите ConfigMap с именем ovn-ic-config в рабочем кластере. Используйте следующую команду.

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

- Выйдите из межкластерного подключения через операции платформы.

2. Войдите в Leader Pod ovn-central с помощью следующей команды.

```
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 с помощью следующей команды.

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

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

- Команда для Podman:

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

- Команда 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 с помощью следующей команды.

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

Настройка высокой доступности gateway-узла кластера

Чтобы после присоединения к межкластерному подключению настроить высокую доступность gateway кластера, выполните следующие действия:

1. Войдите в кластер, который нужно преобразовать в High Availability Gateway, и выполните следующую команду, чтобы изменить значение поля `enable-ic` на `false`.
   Примечание: Изменение значения поля `enable-ic` на `false` нарушит межкластерное подключение до тех пор, пока оно снова не будет установлено в `true`.

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

2. Измените конфигурацию gateway-узла, обновив поле `gw-nodes` и разделив gateway-узлы английскими запятыми; также измените поле `enable-ic` на `true`.

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

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

3. Перейдите в Pod в кластере ovn-central и выполните команду `ovn-nbctl lrp-get-gateway-chassis {current cluster name}-ts`, чтобы убедиться, что конфигурация применяется.

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

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

Настройка Egress Gateway

Содержание

Обзор

Egress Gateway, также называемый VPC Egress Gateway, предоставляет стабильные исходящие адреса для Pods в overlay-сети. Он маршрутизирует выбранные workloads через выделенные gateway Pods, прежде чем трафик покидает cluster.

Используйте Egress Gateway, когда вам нужны:

- Стабильные source IP addresses для определённых workloads
- Контроль egress на уровне workload вместо контроля на уровне subnet
- Более высокая пропускная способность за счёт горизонтального масштабирования
- Более быстрое переключение при отказе для исходящего трафика

Основные возможности:

- Высокая доступность Active-Active через ECMP с горизонтальным масштабированием пропускной способности
- Быстрое переключение при отказе через BFD, обычно менее чем за 1 секунду
- Поддержка IPv4, IPv6 и dual-stack окружений
- Точное сопоставление трафика через NamespaceSelector и PodSelector
- Гибкое планирование через node selectors и tolerations

Текущие ограничения:

- Развёртывания с несколькими репликами требуют нескольких egress IPs
- Записи сопоставления Source NAT не сохраняются

Egress Gateway и Centralized Gateway

Используйте этот раздел, чтобы выбрать модель gateway, которая лучше всего подходит для вашего сценария. Сведения о centralized mode см. в Configure Centralized Gateway.

Dimension	Egress Gateway	Centralized Gateway
Granularity	Точный контроль через `selectors` и `policies` (Namespace/Pod/Subnet/IPBlock).	Применяется на уровне subnet (`gatewayType: centralized`).
Egress Address Source	Использует выделенные gateway Pods с external subnet IPs.	Использует назначенные gateway nodes; при `natOutgoing: true` egress использует node IPs.
Data Path	Трафик маршрутизируется к VPC Egress Gateway Pods, затем выполняется SNAT во внешнюю сеть.	Трафик маршрутизируется к назначенным nodes (`ovn0`), а затем пересылается правилами host routing/NAT.
HA and Failover	Active-Active ECMP; поддерживает быстрое переключение при отказе BFD (меньше секунды).	Поддерживает ECMP и primary-backup; переключение обычно медленнее, чем у Egress Gateway на BFD.
Scheduling Control	Поддерживает средства управления размещением gateway workloads (`nodeSelector` , `tolerations`).	Gateway nodes выбираются через `gatewayNode` или `gatewayNodeSelectors` .
Typical Use Cases	Изоляция egress на уровне tenant или workload, точный контроль политик, более высокая производительность и быстрое переключение при отказе.	Более простой фиксированный egress на уровне subnet, аудит, IP allowlists и управление source IP на firewall.

Рекомендации по выбору:

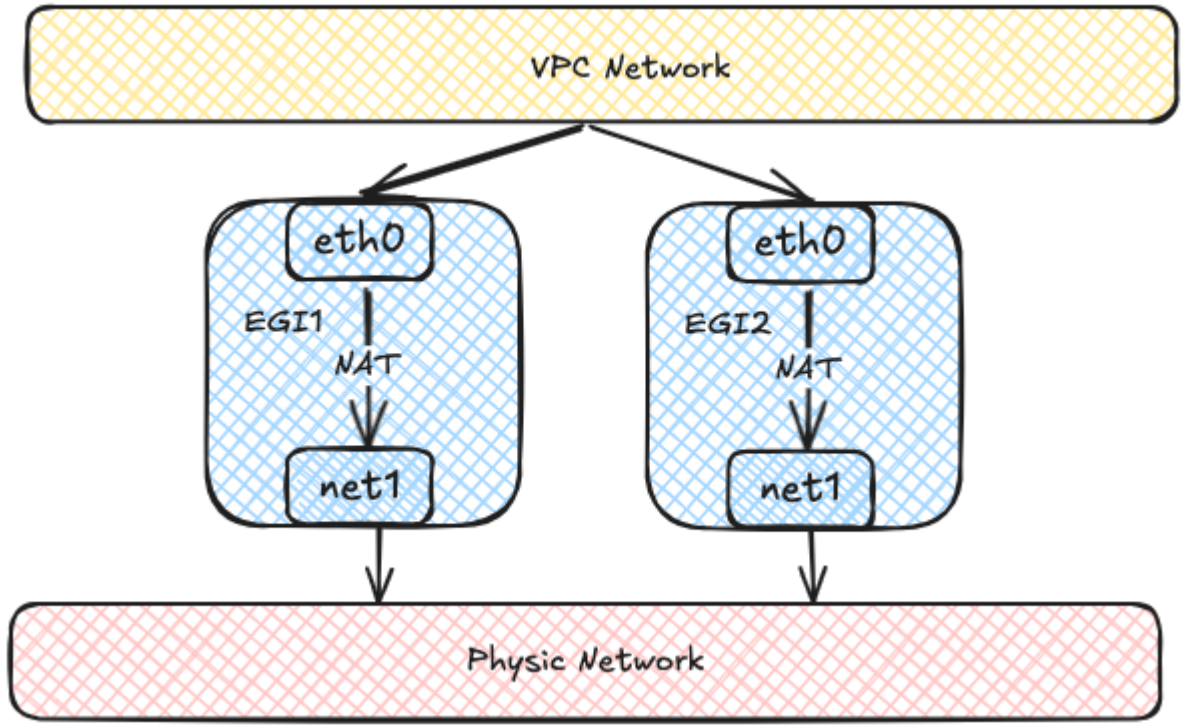
- Выбирайте **Egress Gateway**, если вам нужен контроль политик на уровне workload, масштабируемая пропускная способность и быстрое переключение при отказе.
- Выбирайте **Centralized Gateway**, если достаточно фиксированного egress на уровне subnet и простоты эксплуатации.

Как работает Egress Gateway

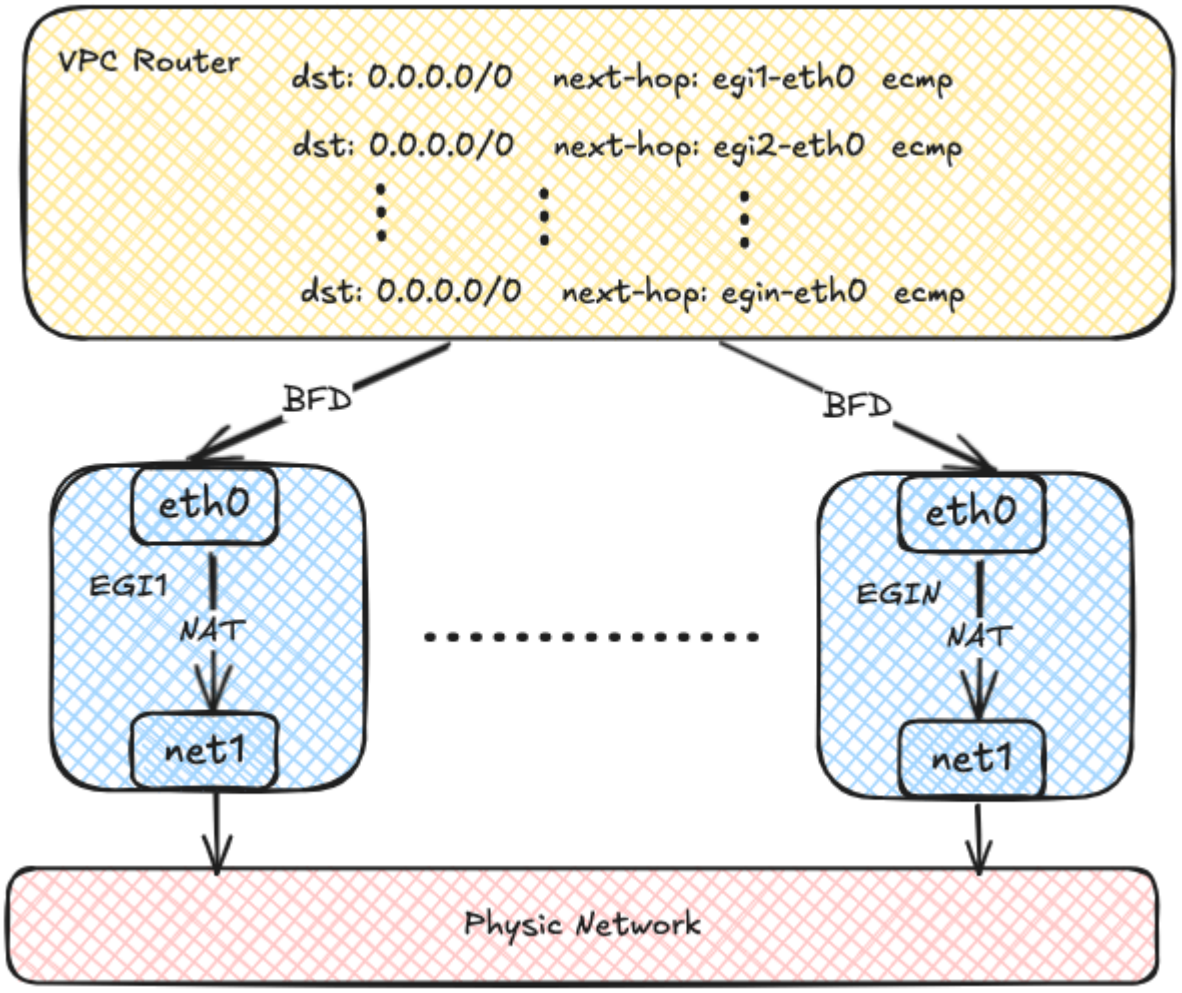
Egress Gateway работает как один или несколько Pods. Каждый Pod использует два сетевых интерфейса:

- Один интерфейс подключается к overlay-сети внутри cluster.
- Другой интерфейс подключается к внешней underlay-сети.

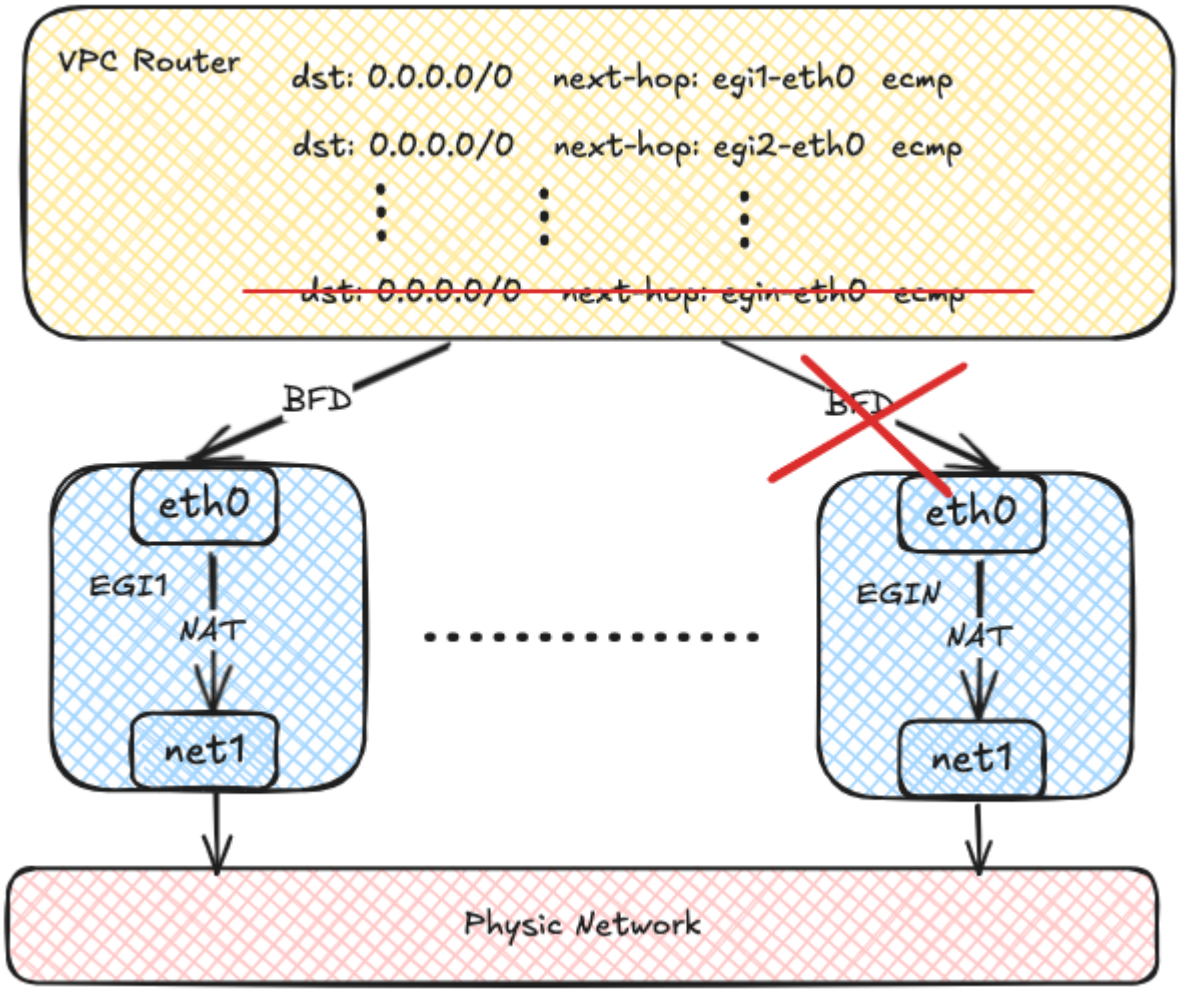
Трафик от выбранных Pods сначала пересылается на gateway Pods, а затем отправляется во внешнюю сеть через интерфейс underlay.



Каждый экземпляр Egress Gateway регистрирует свой адрес в таблице маршрутизации OVN. Когда Pod в overlay-сети обращается к внешней сети, OVN использует хеширование по source-address для распределения трафика между несколькими экземплярами gateway. Это обеспечивает балансировку нагрузки и позволяет масштабировать пропускную способность по горизонтали по мере добавления новых экземпляров.



OVN может использовать BFD для проверки нескольких экземпляров gateway. Если один экземпляр выходит из строя, OVN помечает соответствующий маршрут как недоступный и быстро перенаправляет трафик на рабочий экземпляр.



Перед началом

Перед началом убедитесь, что выполнены следующие предварительные условия:

- *Multus CNI Plugin* **ДОЛЖЕН** быть уже установлен в cluster.
- Внешняя underlay-сеть, VLAN и bridge network уже спланированы и доступны.
- Вы определили, какие workloads должны использовать gateway и требуется ли им SNAT.

Для установки *Multus CNI Plugin* см. Deploying the Multus CNI Plugin.

Процесс настройки

Настройте Egress Gateway в следующем порядке:

1. Подготовьте подключение внешней сети, включая subnet и Network Attachment Definition.
2. Создайте ресурс VPC Egress Gateway и укажите внешний subnet и policies.
3. Проверьте состояние ресурса, маршруты и пересылку трафика.
4. При необходимости масштабируйте реплики и включите быстрое переключение при отказе на основе BFD.

Шаг 1: Подготовка подключения внешней сети

Egress Gateway использует несколько интерфейсов для подключения как к внутренней, так и к внешней сети. Перед созданием gateway подготовьте следующие ресурсы:

- Внешний subnet
- Network Attachment Definition (NAD) для этого subnet

В следующем примере в качестве внешней сети используется underlay subnet Kube-OVN.

> **NOTE**
>
> В этом примере предполагается, что bridge network и ресурс VLAN `external-vlan` уже созданы для underlay-сети.

```yaml
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: underlay-ext
  namespace: default
spec:
  config: |-
    {
      "cniVersion": "0.3.0",
      "type": "kube-ovn", 1
      "server_socket": "/run/openvswitch/kube-ovn-daemon.sock", 2
      "provider": "underlay-ext.default.ovn" 3
    }
---
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: underlay-ext
spec:
  protocol: IPv4
  provider: underlay-ext.default.ovn 4
  cidrBlock: 172.17.0.0/16 5
  gateway: 172.17.0.1 6
  vlan: external-vlan 7
  excludeIps: 8
    - 172.17.0.11..172.17.0.20
```

1. Используйте плагин Kube-OVN CNI для secondary network.
2. Сокет демона Kube-OVN, используемый плагином CNI.
3. Имя provider в формате `<network attachment definition name>.<namespace>.ovn`.
4. Provider, используемый subnet. Это значение ДОЛЖНО совпадать с provider в NetworkAttachmentDefinition.
5. CIDR внешней underlay-сети.
6. Gateway внешней underlay-сети.
7. Ресурс VLAN, используемый underlay subnet.
8. Диапазон IP, исключённый из автоматического назначения. Подробности см. в Example Subnet custom resource (CR) with Kube-OVN Underlay Network.

TIP

Перед использованием underlay subnet убедитесь, что физическая сеть, VLAN и bridge network подготовлены корректно. Подробности планирования окружения см. в Preparing Kube-OVN Underlay Physical Network.

Шаг 2: Создание VPC Egress Gateway

Создайте ресурс VPC Egress Gateway и определите, какие workloads должны его использовать.

```yaml
apiVersion: kubeovn.io/v1
kind: VpcEgressGateway
metadata:
  name: gateway1
  namespace: default (1)
spec:
  replicas: 1 (2)
  internalSubnet: ovn-default (3)
  externalSubnet: underlay-ext (4)
  externalIPs: (5)
    - 172.17.0.11
    - 172.17.0.12
  resources: (6)
    requests:
      cpu: 100m
      memory: 128Mi
    limits:
      cpu: 200m
      memory: 256Mi
      ephemeral-storage: 2Gi
  nodeSelector: (7)
    - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
            - node1
            - node2
  tolerations: (8)
    - key: node-role.kubernetes.io/control-plane
      operator: Exists
      effect: NoSchedule
  selectors: (9)
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: ns1
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: ns2
      podSelector:
        matchLabels:
          app: myapp
  policies: (10)
    - snat: true (11)
```

```
    subnets: 12
      - subnet1
  - snat: false
    ipBlocks: 13
      - 10.18.0.0/16
```

1. Namespace, в котором создаются экземпляры VPC Egress Gateway.
2. Количество экземпляров VPC Egress Gateway.
3. Внутренний subnet, подключённый к внутренней сети. Subnet ДОЛЖЕН быть overlay subnet в том же VPC и иметь достаточно свободных IP для экземпляров gateway. Если не указан, gateway Pods будут использовать subnet по умолчанию для внутренней сети VPC.
4. Внешний subnet, подключённый к внешней сети.
5. Внешние IP, используемые gateway Pods в underlay-сети. Каждый экземпляр gateway получает один IP из этого списка. Эти IP ДОЛЖНЫ находиться в пределах CIDR внешнего subnet и должны быть включены в диапазон `excludeIps` этого subnet. Рекомендуется зарезервировать `.spec.replicas + 1` IP, чтобы gateway Pod мог получить IP даже в граничных случаях.
6. Запросы и лимиты ресурсов для каждого экземпляра VPC Egress Gateway. Если не указано, будут применены значения по умолчанию для requests и limits, определённые в контроллере VPC Egress Gateway.
7. Node selectors, используемые для размещения экземпляров VPC Egress Gateway.
8. Tolerations, используемые для размещения экземпляров VPC Egress Gateway.
9. Namespace selectors и Pod selectors, используемые для выбора Pods, которые обращаются к внешней сети через VPC Egress Gateway.
10. Policies для VPC Egress Gateway, включая SNAT и subnets/ipBlocks, которые нужно применить.
11. Следует ли включать SNAT для policy.
12. Subnets, к которым применяется policy.
13. IP blocks, к которым применяется policy.

Этот пример создаёт VPC Egress Gateway с именем *gateway1* в namespace `default`. Трафик, соответствующий selectors и policies, пересылается через внешний subnet *underlay-ext*. В этом примере это включает:

- Pods в namespace *ns1*
- Pods в namespace *ns2* с меткой `app: myapp`
- Трафик, относящийся к subnet *subnet1*
- Трафик, относящийся к CIDR *10.18.0.0/16*

> **NOTE**
>
> Pods, соответствующие *.spec.selectors*, всегда SNATed gateway.

Шаг 3: Проверка gateway

После создания gateway убедитесь, что он готов и пересылает трафик ожидаемым образом.

1. Проверка состояния ресурса

Начните с базового состояния ресурса:

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

Затем проверьте подробную информацию о gateway:

```
kubectl get veg gateway1 -o wide
NAME       VPC           REPLICAS   BFD ENABLED   EXTERNAL SUBNET   PHASE
READY   INTERNAL IPS     EXTERNAL IPS      WORKING NODES   AGE
gateway1   ovn-cluster   1          false         underlay-ext      Compl
eted   true    ["10.16.0.12"]   ["172.17.0.11"]   ["node1"]       82s
```

Наконец, убедитесь, что workloads gateway запущены:

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

$ kubectl get pod -n default -l ovn.kubernetes.io/vpc-egress-gateway=gate
way1 -o wide
NAME                        READY   STATUS    RESTARTS   AGE     IP
NODE    NOMINATED NODE   READINESS GATES
gateway1-b9f8b4448-76lhm   1/1     Running   0          4m48s   10.16.0.1
2   node1   <none>           <none>
```

2. Проверка сетевой конфигурации внутри gateway Pod

Проверьте IP addresses, записи маршрутизации и правила iptables внутри gateway Pod:

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

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

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

$ kubectl exec -n default gateway1-b9f8b4448-76lhm -c gateway -- ip route
```

```
show table 1000
default via 10.16.0.1 dev eth0

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

3. Подтверждение пересылки трафика

Снимите пакеты в gateway Pod, чтобы убедиться, что трафик проходит через gateway:

```
$ kubectl exec -n default gateway1-b9f8b4448-76lhm -c gateway -- tcpdump
-i any -nnve icmp and host 172.17.0.1
tcpdump: data link type LINUX_SLL2
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapsh
ot length 262144 bytes
06:50:58.936528 eth0  In  ifindex 17 92:26:b8:9e:f2:1c ethertype IPv4 (0x
0800), length 104: (tos 0x0, ttl 63, id 30481, offset 0, flags [DF], prot
o ICMP (1), length 84)
    10.17.0.9 > 172.17.0.1: ICMP echo request, id 37989, seq 0, length 64
06:50:58.936574 net1  Out ifindex 2 62:d8:71:90:7b:86 ethertype IPv4 (0x0
800), length 104: (tos 0x0, ttl 62, id 30481, offset 0, flags [DF], proto
ICMP (1), length 84)
    172.17.0.11 > 172.17.0.1: ICMP echo request, id 39449, seq 0, length
64
06:50:58.936613 net1  In  ifindex 2 02:42:39:79:7f:08 ethertype IPv4 (0x0
800), length 104: (tos 0x0, ttl 64, id 26701, offset 0, flags [none], pro
to ICMP (1), length 84)
    172.17.0.1 > 172.17.0.11: ICMP echo reply, id 39449, seq 0, length 64
06:50:58.936621 eth0  Out ifindex 17 36:7c:6b:c7:82:6b ethertype IPv4 (0x
0800), length 104: (tos 0x0, ttl 63, id 26701, offset 0, flags [none], pr
oto ICMP (1), length 84)
    172.17.0.1 > 10.17.0.9: ICMP echo reply, id 37989, seq 0, length 64
```

4. Подтверждение OVN routing policies

OVN Logical Router policies автоматически создаются для выбранного трафика:

```
$ kubectl ko nbctl lr-policy-list ovn-cluster
Routing Policies
     31000                             ip4.dst == 10.16.0.0/16   allow
     31000                             ip4.dst == 10.17.0.0/16   allow
     31000                           ip4.dst == 100.64.0.0/16   allow
     30000                              ip4.dst == 172.18.0.2  reroute  1
00.64.0.4
     30000                              ip4.dst == 172.18.0.3  reroute  1
00.64.0.3
     30000                              ip4.dst == 172.18.0.4  reroute  1
00.64.0.2
     29100                    ip4.src == $VEG.8ca38ae7da18.ipv4  reroute  1
0.16.0.12 1
     29100                     ip4.src == $VEG.8ca38ae7da18_ip4  reroute  1
0.16.0.12 2
     29000 ip4.src == $ovn.default.kube.ovn.control.plane_ip4  reroute  1
00.64.0.3
     29000       ip4.src == $ovn.default.kube.ovn.worker2_ip4  reroute  1
00.64.0.2
     29000        ip4.src == $ovn.default.kube.ovn.worker_ip4  reroute  1
00.64.0.4
     29000     ip4.src == $subnet1.kube.ovn.control.plane_ip4  reroute  1
00.64.0.3
     29000           ip4.src == $subnet1.kube.ovn.worker2_ip4  reroute  1
00.64.0.2
     29000            ip4.src == $subnet1.kube.ovn.worker_ip4  reroute  1
00.64.0.4
```

1 Logical Router Policy, используемая VPC Egress Gateway для пересылки трафика, соответствующего *.spec.policies*.

2 Logical Router Policy, используемая VPC Egress Gateway для пересылки трафика, соответствующего *.spec.selectors*.

Необязательно: включение балансировки нагрузки с несколькими репликами

NOTE

Перед масштабированием реплик убедитесь, что в внешнем subnet подготовлено достаточно external IPs и что они указаны в *.spec.externalIPs*.

Чтобы включить балансировку нагрузки ECMP и масштабировать пропускную способность по горизонтали, увеличьте *.spec.replicas*:

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

$ kubectl get veg -n default gateway1
NAME       VPC           REPLICAS   BFD ENABLED   EXTERNAL SUBNET   PHASE
READY   AGE
gateway1   ovn-cluster   2          false         underlay-ext      Compl
eted   true    39m

$ kubectl get pod -n default -l ovn.kubernetes.io/vpc-egress-gateway=gate
way1 -o wide
NAME                        READY   STATUS    RESTARTS   AGE   IP
NODE    NOMINATED NODE   READINESS GATES
gateway1-b9f8b4448-76lhm   1/1     Running   0          40m   10.16.0.12
node1   <none>           <none>
gateway1-b9f8b4448-zd4dl   1/1     Running   0          64s   10.16.0.13
node2   <none>           <none>

$ kubectl ko nbctl lr-policy-list ovn-cluster
Routing Policies
     31000                            ip4.dst == 10.16.0.0/16    allow
     31000                            ip4.dst == 10.17.0.0/16    allow
     31000                           ip4.dst == 100.64.0.0/16    allow
     30000                              ip4.dst == 172.18.0.2  reroute  1
00.64.0.4
     30000                              ip4.dst == 172.18.0.3  reroute  1
00.64.0.3
     30000                              ip4.dst == 172.18.0.4  reroute  1
00.64.0.2
     29100                  ip4.src == $VEG.8ca38ae7da18.ipv4  reroute  1
0.16.0.12, 10.16.0.13
     29100                   ip4.src == $VEG.8ca38ae7da18_ip4  reroute  1
0.16.0.12, 10.16.0.13
     29000 ip4.src == $ovn.default.kube.ovn.control.plane_ip4  reroute  1
00.64.0.3
     29000       ip4.src == $ovn.default.kube.ovn.worker2_ip4  reroute  1
00.64.0.2
     29000        ip4.src == $ovn.default.kube.ovn.worker_ip4  reroute  1
00.64.0.4
     29000     ip4.src == $subnet1.kube.ovn.control.plane_ip4  reroute  1
00.64.0.3
     29000           ip4.src == $subnet1.kube.ovn.worker2_ip4  reroute  1
00.64.0.2
```

```
     29000            ip4.src == $subnet1.kube.ovn.worker_ip4  reroute  1
00.64.0.4
```

Необязательно: включение высокой доступности на основе BFD

Переключение при отказе на основе BFD зависит от VPC BFD LRP. Включите его в следующем порядке.

1. Включение BFD Port на VPC

Сначала включите BFD Port на 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) Следует ли включать BFD Port.

(2) IP address BFD Port, который ДОЛЖЕН быть допустимым IP address, не конфликтующим с ЛЮБЫМИ другими IPs/Subnets.

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

> **TIP**
>
> Ресурс Vpc *ovn-cluster* существует по умолчанию. Вы можете отредактировать его напрямую, чтобы включить BFD Port.

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

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

1 BFD Port, созданный на OVN Logical Router.

2. Включение BFD на VPC Egress Gateway

Затем включите BFD на VPC Egress Gateway, установив *.spec.bfd.enabled* в *true*:

```yaml
apiVersion: kubeovn.io/v1
kind: VpcEgressGateway
metadata:
  name: gateway2
  namespace: default
spec:
  vpc: ovn-cluster (1)
  replicas: 2
  internalSubnet: ovn-default (2)
  externalSubnet: underlay-ext (3)
  externalIPs: (4)
    - 172.17.0.11
    - 172.17.0.12
    - 172.17.0.13
  bfd:
    enabled: true (5)
    minRX: 100 (6)
    minTX: 100 (7)
    multiplier: 5 (8)
  policies:
    - snat: true
      ipBlocks:
        - 10.18.0.0/16
```

(1) VPC, к которому относится Egress Gateway.

(2) Внутренний subnet, к которому подключены экземпляры Egress Gateway.

(3) Внешний subnet, к которому подключены экземпляры Egress Gateway.

(4) External IPs, назначенные экземплярам Egress Gateway.

(5) Следует ли включать BFD для Egress Gateway.

(6) Минимальный интервал приёма для BFD, в миллисекундах.

(7) Минимальный интервал передачи для BFD, в миллисекундах.

(8) Multiplier для BFD, определяющий количество пропущенных пакетов до объявления сбоя.

Этот пример создаёт VPC Egress Gateway с именем *gateway2* с двумя репликами и включённым BFD. Если один экземпляр выходит из строя, BFD session переходит в состояние down, OVN помечает маршрут как недоступный и перенаправляет трафик на рабочий экземпляр.

Время обнаружения отказа зависит от настроек BFD. Используйте следующую формулу: *break time = (multiplier + 1) * max(minRX, minTX)*. С этой примерной конфигурацией обнаружение отказа составляет примерно 500–600 ms.

> **NOTE**
>
> Существующие соединения могут быть прерваны во время переключения при отказе и потребовать повторного подключения. Новые соединения при этом по-прежнему могут устанавливаться нормально.

3. Проверка состояния BFD

Проверьте состояние VPC Egress Gateway:

```
$ kubectl get veg -n default gateway2 -o wide
NAME       VPC           REPLICAS   BFD ENABLED   EXTERNAL SUBNET   PHASE
READY   INTERNAL IPS                EXTERNAL IPS                   WOR
KING NODES       AGE
gateway2   ovn-cluster   2          true          underlay-ext      Compl
eted   true    ["10.16.0.12","10.16.0.13"]   ["172.17.0.11","172.17.0.1
2"]   ["node1","node2"]   58s

$ kubectl get pod -n default -l ovn.kubernetes.io/vpc-egress-gateway=gate
way2 -o wide
NAME                        READY   STATUS    RESTARTS   AGE     IP
NODE    NOMINATED NODE   READINESS GATES
gateway2-fcc6b8b87-8lgvx   1/1     Running   0          2m18s   10.16.0.1
3   node2   <none>           <none>
gateway2-fcc6b8b87-wmww6   1/1     Running   0          2m18s   10.16.0.1
2   node1   <none>           <none>

$ kubectl ko nbctl lr-policy-list ovn-cluster
Routing Policies
     31000                            ip4.dst == 10.16.0.0/16    allow
     31000                            ip4.dst == 10.17.0.0/16    allow
     31000                           ip4.dst == 100.64.0.0/16    allow
     30000                              ip4.dst == 172.18.0.2  reroute  1
00.64.0.4
     30000                              ip4.dst == 172.18.0.3  reroute  1
00.64.0.3
     30000                              ip4.dst == 172.18.0.4  reroute  1
00.64.0.2
     29100                    ip4.src == $VEG.8ca38ae7da18.ipv4  reroute  1
0.16.0.12, 10.16.0.13  bfd
     29100                     ip4.src == $VEG.8ca38ae7da18_ip4  reroute  1
0.16.0.12, 10.16.0.13  bfd
     29090                    ip4.src == $VEG.8ca38ae7da18.ipv4     drop
     29090                     ip4.src == $VEG.8ca38ae7da18_ip4     drop
     29000 ip4.src == $ovn.default.kube.ovn.control.plane_ip4  reroute  1
00.64.0.3
     29000       ip4.src == $ovn.default.kube.ovn.worker2_ip4  reroute  1
00.64.0.2
     29000        ip4.src == $ovn.default.kube.ovn.worker_ip4  reroute  1
00.64.0.4
     29000     ip4.src == $subnet1.kube.ovn.control.plane_ip4  reroute  1
00.64.0.3
     29000           ip4.src == $subnet1.kube.ovn.worker2_ip4  reroute  1
```

```
00.64.0.2
     29000            ip4.src == $subnet1.kube.ovn.worker_ip4  reroute  1
00.64.0.4

$ kubectl ko nbctl list bfd
_uuid               : 223ede10-9169-4c7d-9524-a546e24bfab5
detect_mult         : 5
dst_ip              : "10.16.0.12"
external_ids        : {af="4", vendor=kube-ovn, vpc-egress-gateway="defau
lt/gateway2"}
logical_port        : "bfd@ovn-cluster"
min_rx              : 100
min_tx              : 100
options             : {}
status              : up

_uuid               : b050c75e-2462-470b-b89c-7bd38889b758
detect_mult         : 5
dst_ip              : "10.16.0.13"
external_ids        : {af="4", vendor=kube-ovn, vpc-egress-gateway="defau
lt/gateway2"}
logical_port        : "bfd@ovn-cluster"
min_rx              : 100
min_tx              : 100
options             : {}
status              : up
```

Затем проверьте BFD sessions:

```
$ kubectl exec -n default gateway2-fcc6b8b87-8lgvx -c bfdd -- bfdd-contro
l status
There are 1 sessions:
Session 1
 id=1 local=10.16.0.13 (p) remote=10.255.255.255 state=Up

$ kubectl exec -n default gateway2-fcc6b8b87-wmww6 -c bfdd -- bfdd-contro
l status
There are 1 sessions:
Session 1
 id=1 local=10.16.0.12 (p) remote=10.255.255.255 state=Up
```

NOTE

Если все экземпляры gateway недоступны, egress traffic, обрабатываемый VPC Egress Gateway, отбрасывается.

Операции, которые могут прервать трафик

Следующие операции могут кратковременно прервать egress traffic, поскольку они удаляют или пересоздают экземпляры gateway:

1. Изменение количества реплик
2. Изменение конфигурации, например internal или external IPs, node selectors или BFD settings
3. Обновление или понижение версии Kube-OVN, если *.spec.image* не указан
4. Ручное удаление Egress Gateway Pod

Дополнительные ресурсы

- RFC 5880 - Bidirectional Forwarding Detection (BFD) ↗

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

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

Содержание

Установка Multus CNI

Развертывание плагина Multus CNI

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

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

Создание подсетей

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

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

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

После создания примените ресурс:

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

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

NOTE

- `spec.provider` должен совпадать с провайдером в NetworkAttachmentDefinition.
- Если необходимо использовать Underlay подсеть, установите `spec.vlan` подсети в имя VLAN CR, который вы хотите использовать. Настройте другие 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 проверить успешное создание двух сетевых интерфейсов:

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

Дополнительные возможности

Фиксированный IP

- **Основной сетевой интерфейс (первый интерфейс)**: если нужно 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.

- **Второй сетевой интерфейс (второй или другие интерфейсы)**: базовый способ аналогичен основному интерфейсу, но `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 `routers` в config NetworkAttachmentDefinition и заполните маршруты, которые нужно 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 централизованного шлюза

Централизованный шлюз позволяет Pod-ам внутри подсети получать доступ к внешней сети с использованием фиксированных IP-адресов. Это особенно полезно для задач безопасности, таких как аудит сети, белые списки IP и управление правилами файрвола, когда необходимо идентифицировать и контролировать трафик с конкретных исходных IP. В режиме централизованного шлюза весь исходящий трафик от Pod-ов маршрутизируется через назначенные узлы-шлюзы, что обеспечивает централизованное применение сетевых политик и мониторинг.

> **NOTE**
>
> Pod-ы в централизованной подсети не могут быть доступны через `hostport` или Service типа NodePort с `externalTrafficPolicy: Local`.

Если вы хотите, чтобы трафик внутри подсети выходил в внешнюю сеть с фиксированного IP для задач безопасности, таких как аудит и белые списки, вы можете установить тип шлюза в подсети как centralized.
В режиме централизованного шлюза пакеты от Pod-ов, обращающихся к внешней сети, сначала направляются на NIC `ovn0` конкретных узлов, а затем исходят через правила маршрутизации хоста.
Когда `natOutgoing` установлено в `true`, Pod будет использовать IP конкретного узла при доступе к внешней сети.

Пример централизованного шлюза приведён ниже, где поле `gatewayType` равно `centralized`,
а `gatewayNode` — это NodeName конкретной машины в Kubernetes.

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: centralized
spec:
  protocol: IPv4
  cidrBlock: 10.166.0.0/16
  default: false
  excludeIps:
  - 10.166.0.1
  gateway: 10.166.0.1
  gatewayType: centralized
  gatewayNode: "node1,node2"
  natOutgoing: true
```

- Если централизованный шлюз должен указать конкретный NIC машины для исходящего трафика,
  формат `gatewayNode` можно изменить на `kube-ovn-worker:172.18.0.2, kube-ovn-control-plane:172.18.0.3`.
- В определение CRD подсети с версии Kube-OVN v1.12.0 добавлено поле spec `enableEcmp` для переноса переключателя ECMP на уровень подсети.
  Вы можете включать или отключать режим ECMP для разных подсетей. Параметр `enable-ecmp` в развертывании `kube-ovn-controller` больше не используется.
  После обновления с предыдущей версии до v1.12.0 переключатель подсети автоматически унаследует значение из глобального параметра.

NOTE

В режиме ECMP централизованного шлюза kube-ovn-controller активно проверяет состояние узлов с помощью ping, обнаруживая сбои в течение 5 секунд и завершая переключение в течение 5-10 секунд, в течение которых часть трафика может не пройти.

В режиме primary-backup централизованного шлюза переключение происходит на основе статуса Node Ready и может занять несколько минут при отключении питания.

Содержание

Использование селекторов меток для указания узлов-шлюзов

Помимо прямого указания имён узлов, вы можете использовать `gatewayNodeSelectors` для динамического выбора узлов-шлюзов с помощью селекторов меток.
Этот подход более гибкий, особенно полезен, когда имена узлов не фиксированы или когда необходимо динамически выбирать шлюзы на основе меток.

> **NOTE**
>
> - Если `gatewayNode` не пуст, он имеет приоритет, и `gatewayNodeSelectors` игнорируется.
> - Несколько селекторов оцениваются по логике OR — узел, соответствующий любому селектору, становится узлом-шлюзом.
> - При изменении меток узлов система автоматически обновляет список узлов-шлюзов.

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: centralized-selector
spec:
  protocol: IPv4
  cidrBlock: 10.166.0.0/16
  default: false
  excludeIps:
  - 10.166.0.1
  gateway: 10.166.0.1
  gatewayType: centralized
  gatewayNodeSelectors:
    - matchLabels:
        role: gateway
    - matchExpressions:
        - key: node-type
          operator: In
          values: ["gateway", "egress"]
  natOutgoing: true
```

Настройка IPPool

IPPool — это более детализированная единица управления IPAM по сравнению с Subnet.
Вы можете разделить сегмент подсети на несколько единиц с помощью IPPool, и каждая единица будет привязана 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) Необязательные namespace, к которым привязан IP pool. Подам в привязанном namespace будут выделяться IP только из привязанных pool(ов), а не из других диапазонов в подсети(ях).

Использование IPPool

Чтобы назначать IP-адреса случайным образом из IP pool, просто привяжите IP pool к нужному namespace(ам).
При создании Pod 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, к которой принадлежит IP pool.

Также IP pool можно назначить Pod 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 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. IP-адреса вне диапазона подсети разрешены, 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 назначении IP из IP pool зарезервированные IP в диапазоне pool будут пропускаться.
6. При случайном назначении IP из подсети будут исключены 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 ключевое 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 соответственно устанавливает MTU для виртуальных сетевых интерфейсов.
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 MTU

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

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 настройки MTU применились равномерно во всем кластере.

Почему это важно?

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

1. Вы настраиваете 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 отбрасывается, если пакеты превышают MTU `ovn0`

Решение: После увеличения MTU необходимо пересоздать все Pods, чтобы обеспечить согласованную настройку MTU по всему сетевому пути.

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:

```
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 для конкретной подсети равным 1450, используйте команду:

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

Эта настройка переопределит глобальное значение MTU для указанной подсети.

Содержание

weight: 13

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 должны 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.

Процедура

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

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

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 этом обе имеют включённый `u2oInterconnection`, сначала нужно настроить параметр kube-ovn-controller, а затем настроить изоляцию подсети.

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 **необходимо** установить значение `false`. В противном случае трафик от шлюза к Pod будет прерываться (например, ping-запросы доходят до Pod, но ответы отбрасываются), поскольку 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:

```yaml
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 (правила с более высоким приоритетом оцениваются первыми; рекомендуемый диапазон: 1002-1899)

NOTE

- Поле `acls` обеспечивает оценку правил на основе приоритетов, 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.

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.
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": ""
      }
    }
  }
}'
```

Задача: Миграция с OCP Route на GatewayAPI Route

Содержание

Введение

Данное руководство содержит подробные инструкции по миграции с OpenShift Container Platform (OCP) Routes на Kubernetes Gateway API HTTPRoutes с использованием Envoy Gateway. Каждый раздел описывает конкретную функцию OCP Route и её эквивалентную конфигурацию в Gateway API.

Предварительные требования

1. Настройка EnvoyGatewayCtl
2. Настройка Gateway
3. Базовое понимание Gateway API Routes

Базовый HTTP Route

Конфигурация OCP Route

В OCP базовый HTTP маршрут создаётся с помощью ресурса Route:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  namespace: demo
spec:
  host: example.com 1
  to: 2
    kind: Service
    name: example-service
  port: 3
    targetPort: 8080
```

1. Имя хоста для маршрута
2. Ссылка на бэкенд-сервис

3 Целевой порт на бэкенд-сервисе

Конфигурация Gateway API

В Gateway API эквивалентная конфигурация использует HTTPRoute:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames: 1
    - example.com
  parentRefs: 2
    - name: demo-gateway
      namespace: demo
  rules:
    - backendRefs: 3
        - name: example-service
          port: 8080
```

1 Имя хоста, которое принимает этот маршрут (эквивалент поля `host` в OCP Route)

2 Ссылка на слушатель Gateway

3 Бэкенд-сервис и порт

Таймауты маршрута

Конфигурация OCP Route

В OCP таймауты настраиваются с помощью аннотаций:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/timeout: 30s (1)
    haproxy.router.openshift.io/timeout-tunnel: 1h (2)
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
```

(1) Общий таймаут для HTTP-запросов

(2) Таймаут для туннельных соединений (WebSocket, HTTP/2 и др.)

Конфигурация Gateway API

В Gateway API таймауты настраиваются в правилах HTTPRoute:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames:
    - example.com
  parentRefs:
    - name: demo-gateway
  rules:
    - backendRefs:
        - name: example-service
          port: 8080
      timeouts: (1)
        request: 30s
        backendRequest: 25s
```

(1) Конфигурация таймаута запроса

Подробнее см. Request Timeouts ↗.

> **NOTE**
>
> В Gateway API нет отдельного таймаута специально для туннельных соединений. Таймаут `request` применяется ко всем типам соединений.

HTTP Strict Transport Security (HSTS)

Конфигурация OCP Route

В OCP HSTS настраивается с помощью аннотации для маршрутов с edge-terminated или re-encrypt TLS:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=31536000;includeSubD
omains;preload (1)
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
  tls:
    termination: edge
```

(1) Конфигурация заголовка HSTS

Конфигурация Gateway API

В Gateway API HSTS настраивается с помощью модификации заголовков ответа:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames:
    - example.com
  parentRefs:
    - name: demo-gateway
  rules:
    - filters: (1)
        - type: ResponseHeaderModifier
          responseHeaderModifier:
            add:
              - name: Strict-Transport-Security
                value: max-age=31536000;includeSubDomains;preload
      backendRefs:
        - name: example-service
          port: 8080
```

(1) Добавление заголовка HSTS в ответ

Подробнее см. HTTP Header Modification ↗.

> **NOTE**
>
> В отличие от кластерного принудительного применения `requiredHSTSPolicies` в OCP, Gateway API требует явной настройки HSTS для каждого маршрута. Глобального механизма принудительного применения HSTS в Gateway API нет.

Сессионная аффинность на основе cookie

Конфигурация OCP Route

В OCP сессионная аффинность настраивается с помощью аннотаций:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/balance: source (1)
    haproxy.router.openshift.io/disable_cookies: "false" (2)
    router.openshift.io/cookie_name: my-cookie (3)
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
```

1. Алгоритм балансировки нагрузки
2. Включение сессионной аффинности на основе cookie
3. Имя cookie для сохранения сессии

Конфигурация Gateway API

В Gateway API сохранение сессии настраивается в правилах HTTPRoute:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames:
    - example.com
  parentRefs:
    - name: demo-gateway
  rules:
    - backendRefs:
        - name: example-service
          port: 8080
      sessionPersistence: 1
        type: Cookie
        sessionName: my-cookie
        cookieConfig:
          lifetimeType: Permanent 2
```

1 Конфигурация сохранения сессии

2 Тип времени жизни cookie: `Permanent` (сохраняется между сессиями браузера) или `Session` (истекает при закрытии браузера)

Подробнее см. Session Affinity/Sticky Sessions ↗.

Маршрутизация по пути

Конфигурация OCP Route

В OCP маршрутизация по пути задаётся полем `path`:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
spec:
  host: example.com
  path: /api 1
  to:
    kind: Service
    name: example-service
```

1 Префикс пути для сопоставления запросов

Конфигурация Gateway API

В Gateway API сопоставление пути настраивается в правилах HTTPRoute:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames:
    - example.com
  parentRefs:
    - name: demo-gateway
  rules:
    - matches: 1
        - path:
            type: PathPrefix
            value: /api
      backendRefs:
        - name: example-service
          port: 8080
```

1 Конфигурация сопоставления пути

Gateway API поддерживает несколько типов сопоставления:

- `PathPrefix`: сопоставление по префиксу пути (эквивалент поведения по умолчанию в OCP)
- `Exact`: точное совпадение пути
- `RegularExpression`: сопоставление с использованием регулярных выражений

Подробнее см. HTTPRoute Matches.

Модификация заголовков

Конфигурация OCP Route

В OCP модификация заголовков выполняется с помощью аннотаций:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/response-set-header: X-Custom-Header:valu
e (1)
    haproxy.router.openshift.io/request-set-header: X-Request-Header:valu
e (2)
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
```

(1) Установка заголовков ответа

(2) Установка заголовков запроса

Конфигурация Gateway API

В Gateway API модификация заголовков настраивается с помощью фильтров:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames:
    - example.com
  parentRefs:
    - name: demo-gateway
  rules:
    - filters:
        - type: RequestHeaderModifier (1)
          requestHeaderModifier:
            add:
              - name: X-Request-Header
                value: value
        - type: ResponseHeaderModifier (2)
          responseHeaderModifier:
            add:
              - name: X-Custom-Header
                value: value
      backendRefs:
        - name: example-service
          port: 8080
```

(1) Модификация заголовков запроса

(2) Модификация заголовков ответа

Подробнее см. HTTP Header Modification ↗.

Ограничения соединений

Конфигурация OCP Route

В OCP ограничения соединений настраиваются с помощью аннотаций:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/pod-concurrent-connections: "100" (1)
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
```

(1) Максимальное количество одновременных соединений на один под бэкенда

Конфигурация Gateway API

В Gateway API ограничения соединений настраиваются с помощью ClientTrafficPolicy, прикреплённой к Gateway:

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: ClientTrafficPolicy
metadata:
  name: connection-limit-policy
  namespace: demo
spec:
  targetRefs: (1)
    - group: gateway.networking.k8s.io
      kind: Gateway
      name: demo-gateway
  connection: (2)
    connectionLimit:
      value: 100
```

(1) Прикрепление политики к Gateway

(2) Конфигурация ограничения соединений

Подробнее см. Connection Limit↗.

NOTE

В отличие от ограничения на уровне пода бэкенда в OCP, ограничение соединений в Gateway API применяется на уровне Gateway и распределяется между экземплярами Envoy proxy.

Ограничение скорости (Rate Limiting)

Конфигурация OCP Route

В OCP ограничение скорости настраивается с помощью аннотаций:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/rate-limit-connections: "true" 1
    haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp: "1
0"
    haproxy.router.openshift.io/rate-limit-connections.rate-http: "100"
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
```

1 Конфигурация ограничения скорости

Конфигурация Gateway API

В Gateway API ограничение скорости настраивается с помощью BackendTrafficPolicy:

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: BackendTrafficPolicy
metadata:
  name: rate-limit-policy
  namespace: demo
spec:
  targetRefs:
    - group: gateway.networking.k8s.io
      kind: HTTPRoute
      name: example-route
  rateLimit: 1
    type: Local
    local:
      rules:
        - limit:
            requests: 100
            unit: Second
```

1 Конфигурация ограничения скорости

Подробнее см. Rate Limiting ↗.

> **NOTE**
>
> Ограничение скорости в Gateway API более гибкое, чем аннотации OCP Route, и поддерживает как локальные, так и глобальные механизмы ограничения.

Белый/чёрный список IP

Конфигурация OCP Route

В OCP белый список IP настраивается с помощью аннотаций:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/ip_allowlist: 192.168.1.0/24 10.0.0.1 1
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
```

1 Конфигурация белого списка IP

Конфигурация Gateway API

В Gateway API фильтрация IP реализуется с помощью SecurityPolicy с правилами авторизации:

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: ip-filter-policy
  namespace: demo
spec:
  targetRefs: 1
    - group: gateway.networking.k8s.io
      kind: HTTPRoute
      name: example-route
  authorization: 2
    defaultAction: Deny
    rules:
      - action: Allow
        principal:
          clientCIDRs: 3
            - 192.168.1.0/24
            - 10.0.0.1/32
```

1 Прикрепление политики к HTTPRoute

2. Конфигурация авторизации с действием по умолчанию Deny
3. Белый список IP в формате CIDR

Для чёрного списка IP (denylist) установите `defaultAction: Allow` и используйте `action: Deny`:

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: ip-blocklist-policy
  namespace: demo
spec:
  targetRefs:
    - group: gateway.networking.k8s.io
      kind: HTTPRoute
      name: example-route
  authorization:
    defaultAction: Allow (1)
    rules:
      - action: Deny (2)
        principal:
          clientCIDRs:
            - 192.168.100.0/24
```

1. Действие по умолчанию разрешает весь трафик
2. Запрет трафика с указанных IP

Подробнее см. IP Allowlist/Denylist ↗.

> **NOTE**
>
> Убедитесь, что вы правильно настроили определение IP клиента с помощью ClientTrafficPolicy, если ваш Gateway находится за балансировщиком нагрузки или прокси.

Перезапись URL

Конфигурация OCP Route

В OCP перезапись URL настраивается с помощью аннотации:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/rewrite-target: /new-path (1)
spec:
  host: example.com
  path: /old-path
  to:
    kind: Service
    name: example-service
```

(1) Целевой путь для перезаписи

Конфигурация Gateway API

В Gateway API перезапись URL настраивается с помощью фильтров:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames:
    - example.com
  parentRefs:
    - name: demo-gateway
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /old-path
      filters:
        - type: URLRewrite (1)
          urlRewrite:
            path:
              type: ReplacePrefixMatch
              replacePrefixMatch: /new-path
      backendRefs:
        - name: example-service
          port: 8080
```

(1) Фильтр перезаписи URL

Подробнее см. URL Rewrite ↗.

Разрешение маршрутов из других неймспейсов

Конфигурация OCP Route

В OCP политика допуска маршрутов контролирует возможность маршрутам из разных неймспейсов использовать один и тот же hostname. Это настраивается на уровне кластера через Ingress Operator.

Конфигурация Gateway API

В Gateway API доступ из других неймспейсов контролируется на уровне слушателя Gateway:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo-gateway
  namespace: gateway-ns
spec:
  gatewayClassName: envoy-gateway-operator-cpaas-default
  listeners:
    - name: http
      protocol: HTTP
      port: 80
      allowedRoutes: 1
        namespaces:
          from: All  # Разрешить маршруты из всех неймспейсов
```

1 Настройка, какие неймспейсы могут прикреплять маршруты

Варианты значения `allowedRoutes.namespaces.from`:

- `Same`: только маршруты из того же неймспейса, что и Gateway
- `All`: маршруты из любого неймспейса
- `Selector`: маршруты из неймспейсов, соответствующих селектору меток

Подробнее см. Cross-Namespace Routing.

> **NOTE**
>
> В отличие от кластерной политики допуска в OCP, Gateway API контролирует это на уровне каждого слушателя Gateway, обеспечивая более детальный контроль.

Сертификат TLS по умолчанию для Ingress

Конфигурация OCP Route

В OCP маршруты без TLS-конфигурации могут использовать сертификат по умолчанию, настроенный на уровне Ingress Controller.

Конфигурация Gateway API

В Gateway API настройте сертификат TLS по умолчанию на слушателе Gateway:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo-gateway
  namespace: demo
spec:
  gatewayClassName: envoy-gateway-operator-cpaas-default
  listeners:
    - name: https
      protocol: HTTPS
      port: 443
      tls: 1
        mode: Terminate
        certificateRefs:
          - name: default-tls-cert
```

1 Сертификат TLS по умолчанию для слушателя

Любой HTTPRoute, прикреплённый к этому слушателю без специфической TLS-конфигурации, будет использовать этот сертификат по умолчанию.

TLS Re-encrypt с пользовательским CA

Конфигурация OCP Route

В OCP маршруты с re-encrypt TLS завершают TLS на маршрутизаторе и повторно шифруют трафик к бэкенду с проверкой:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
  tls:
    termination: reencrypt (1)
    destinationCACertificate: | (2)
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
```

(1) Режим termination re-encrypt

(2) CA-сертификат для проверки бэкенда

Конфигурация Gateway API

В Gateway API проверка TLS бэкенда настраивается с помощью BackendTLSPolicy:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo-gateway
  namespace: demo
spec:
  gatewayClassName: envoy-gateway-operator-cpaas-default
  listeners:
    - name: https
      protocol: HTTPS
      port: 443
      tls:
        mode: Terminate (1)
        certificateRefs:
          - name: frontend-tls
---
apiVersion: gateway.networking.k8s.io/v1
kind: BackendTLSPolicy
metadata:
  name: backend-tls-policy
  namespace: demo
spec:
  targetRefs: (2)
    - group: ""
      kind: Service
      name: example-service
  validation: (3)
    caCertificateRefs:
      - name: backend-ca-cert
        kind: ConfigMap
    hostname: backend.example.com
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  parentRefs:
    - name: demo-gateway
  hostnames:
    - example.com
  rules:
```

```
    - backendRefs:
        - name: example-service
          port: 8443
```

1. Завершение TLS на Gateway
2. Применение политики к сервису бэкенда
3. Конфигурация проверки TLS бэкенда

Подробнее см. Backend TLS↗.

> **NOTE**
>
> CA-сертификат должен храниться в ConfigMap, а не в Secret.

Edge Termination с пользовательским сертификатом

Конфигурация OCP Route

В OCP edge termination настраивается через TLS-конфигурацию маршрута:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
  tls:
    termination: edge (1)
    certificate: | (2)
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
    key: | (3)
      -----BEGIN PRIVATE KEY-----
      ...
      -----END PRIVATE KEY-----
```

1. Режим termination edge
2. TLS-сертификат
3. Закрытый ключ TLS

Конфигурация Gateway API

В Gateway API TLS-termination настраивается на слушателе Gateway:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: example-tls
  namespace: demo
type: kubernetes.io/tls (1)
data:
  tls.crt: <base64-encoded-cert>
  tls.key: <base64-encoded-key>
---
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo-gateway
  namespace: demo
spec:
  gatewayClassName: envoy-gateway-operator-cpaas-default
  listeners:
    - name: https
      protocol: HTTPS (2)
      port: 443
      tls:
        mode: Terminate (3)
        certificateRefs: (4)
          - name: example-tls
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  parentRefs:
    - name: demo-gateway
      sectionName: https
  hostnames:
    - example.com
  rules:
    - backendRefs:
        - name: example-service
          port: 8080
```

1. Тип секрета TLS
2. Протокол HTTPS
3. Режим termination TLS
4. Ссылка на секрет TLS

TLS Passthrough

Конфигурация OCP Route

В OCP маршруты с passthrough не завершают TLS:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
  tls:
    termination: passthrough (1)
```

1. Режим termination passthrough

Конфигурация Gateway API

В Gateway API TLS passthrough реализуется с помощью TLSRoute:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo-gateway
  namespace: demo
spec:
  gatewayClassName: envoy-gateway-operator-cpaas-default
  listeners:
    - name: tls-passthrough
      protocol: TLS (1)
      port: 443
      tls:
        mode: Passthrough (2)
---
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: TLSRoute
metadata:
  name: example-route
  namespace: demo
spec:
  parentRefs:
    - name: demo-gateway
      sectionName: tls-passthrough
  hostnames: (3)
    - example.com
  rules:
    - backendRefs:
        - name: example-service
          port: 8443
```

(1) Протокол TLS

(2) Режим passthrough

(3) Имя хоста SNI для маршрутизации

Подробнее см. TLS Passthrough и TLSRoute specification ↗.

> **NOTE**
>
> TLSRoute использует SNI (Server Name Indication) для принятия решений о маршрутизации без завершения TLS. Это означает:

- Нет доступа к HTTP-заголовкам или содержимому запроса для маршрутизации
- Нельзя применять фильтры L7 (модификация заголовков, перезапись URL и т.д.)
- Маршрутизация основана исключительно на имени хоста SNI

Сводка сравнения функций

Функция	OCP Route	Gateway API
Базовая HTTP маршрутизация	spec маршрута	HTTPRoute
Маршрутизация по пути	`.spec.path`	`.spec.rules[].matches[].path`
Таймауты	Аннотации	HTTPRoute `.spec.rules[].timeouts`
HSTS	Аннотация	Фильтр ResponseHeaderModifier
Сессионная аффинность	Аннотации	HTTPRoute `.spec.rules[].sessionPersistence`
Модификация заголовков	Аннотации	Фильтры RequestHeaderModifier/ResponseHeaderModi
Ограничения соединений	Аннотация	ClientTrafficPolicy

Функция	OCP Route	Gateway API
Ограничение скорости	Аннотации	BackendTrafficPolicy
Белый/чёрный список IP	Аннотация	SecurityPolicy с авторизацией
Перезапись URL	Аннотация	Фильтр URLRewrite
Кросс-неймспейс	Политика на уровне кластера	Gateway `.spec.listeners[].allowedRoutes`
Сертификат TLS по умолчанию	На уровне контроллера	TLS слушателя Gateway
TLS Re-encrypt	`.spec.tls.termination: reencrypt`	BackendTLSPolicy
TLS Edge	`.spec.tls.termination: edge`	TLS слушателя Gateway с HTTPS
TLS Passthrough	`.spec.tls.termination: passthrough`	TLSRoute с режимом Passthrough

Стратегия миграции

При миграции с OCP Routes на Gateway API:

1. **Начните с Gateway**: Создайте ресурс Gateway с необходимыми слушателями для вашего сценария

- Разверните Gateway в том же неймспейсе или в выделенном неймспейсе для Gateway
- Настройте слушатели для всех нужных протоколов (HTTP, HTTPS, TLS, TCP, UDP)
- Убедитесь, что сервис Gateway создан и имеет доступную конечную точку

2. **Преобразуйте маршруты в HTTPRoutes**: Мигрируйте каждый OCP Route в HTTPRoute, начиная с простых маршрутов
   - Начинайте с непроизводственных или низконагруженных маршрутов для минимизации рисков
   - Проверьте конфигурации сопоставления hostname и пути
   - Протестируйте базовую доступность перед добавлением расширенных функций
3. **Примените политики**: Для функций, требующих политик (BackendTrafficPolicy, SecurityPolicy, ClientTrafficPolicy), создайте и прикрепите их после настройки базового маршрута
   - Добавляйте по одной политике и проверяйте работоспособность
   - Следите за неожиданным поведением или влиянием на производительность
4. **Тестируйте поэтапно**: Проверяйте каждый этап миграции перед переходом к следующему маршруту
   - **Мониторинг**: Проверяйте состояния Gateway и маршрутов на наличие ошибок
   - **Функциональное тестирование**: Убедитесь, что все маршруты работают корректно с помощью curl или автоматизированных тестов
   - **Тестирование производительности**: Сравните время отклика и пропускную способность с OCP Routes
   - **Проверки валидации**:
     - Проверьте корректность применения TLS-сертификатов
     - Тестируйте сессионную аффинность и поведение балансировки нагрузки
     - Проверьте ограничения скорости и политики безопасности
     - Проверьте модификацию заголовков и перезапись URL
5. **Обновите DNS**: После проверки обновите записи DNS, чтобы указывать на новый сервис Gateway

- **Период параллельной работы** (рекомендуется): Направьте часть трафика на новый Gateway, сохраняя OCP Routes активными для возможности отката
- Постепенно переключайте трафик с помощью взвешенного DNS или канареечных релизов
- Мониторьте уровень ошибок, задержки и метрики приложений во время переключения

6. **Процедуры отката**: Если после миграции обнаружены проблемы

- Верните DNS обратно на OCP Routes
- Держите OCP Routes активными минимум 24-48 часов после переключения DNS
- Документируйте любые различия в конфигурации, вызвавшие проблемы
- Имейте план коммуникации со стейкхолдерами на случай необходимости отката

Связанная документация

- Настройка GatewayAPI Gateway
- Настройка GatewayAPI Route
- Настройка GatewayAPI Policy
- Задачи для Envoy Gateway

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
a 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.
```