Сетевая подсистема в Kubernetes

Не забудь включить запись!

План

- Основные компоненты сетевой подсистемы
- Внутри одного пода
- Внутри одного хоста
- Внутри одного кластера
- Внешний доступ до ресурсов кластера
- Между кластерами

Основные компоненты сетевой подсистемы

Network Namespace

- Используются стандартные механизмы ядра
- Каждый Pod запускается в своём Namespace (оно общее для всех контейнеров Pod)
- Что входит в Namespace:
  - Таблица маршрутизации
  - Внешние сетевые интерфейсы + свой личный Loopback
  - Правила Firewall (iptables)
  - Некоторые сетевые `sysctl`

Container Networking Interface (CNI)

CNI - стандартизованный API для управления сетевой подсистемой.

Несмотря на то, что сетевые плагины описываются в конфигурации `kubelet`, фактически с ними взаимодействует Container Runtime.

> Подробный обзор CNI запланирован в Модуле 3

Domain Name System (DNS)

Через встроенный DNS можно получать информацию о подах и сервисах.

- Все сервисы имеют отображения в DNS, а Pod нет
  - В `PodSpec` можно указать `hostname` и `sub-domain`
  - Тогда для Poda будет создана A-запись
- Все сервисы (ClusterIP, Headless) имеют 3 записи в DNS
  - A: `web.default.svc.cluster.local.`
  - SRV: `_https._tcp.web.default.svc.cluster.local.`
  - PTR: `1.0.3.10.in-addr.arpa.`

Подробнее про `sub-domain` и `hostname` - тут

Domain Name System (DNS)

- Внутри кластера возможно обращение по следующим именам
  - Обращение к сервису внутри namespace: **service**
  - Обращение к сервису внутри кластера: **service.ns**
  - FQDN сервиса: **service.ns.svc.cluster.local.**
  - FQDN пода: **10-0-0-1.ns.pod.cluster.local** (Deprecated, но пока работает)

Описание текущей схемы DNS

Сравнение CoreDNS и kube-dns

CoreDNS	kube-dns
Несколько потоков, Go	dnsmasq, один поток, C
Один контейнер в Pod	Три контейнера в Pod
Negative caching by default	-
По умолчанию после v1.12	По умолчанию до v1.12

Доступ извне к DNS Kubernetes

Если необходимо, чтобы внешние клиенты могли резолвить адреса сервисов, есть несколько способов:

- Так себе:
    - Создание статических записей вручную в конфигурации DNS-сервера
    - Публикация CoreDNS используя LoadBalancer
- Нормальные:
    - Отдельный DNS-рекурсор, опубликованный наружу
    - ExternalDNS

dnsPolicy

Задает поведение пода при работе с DNS, есть следующие политики:

- **Default** - наследует `resolv.conf` с ноды
- **ClusterFirst** - зависит от того, настроены ли на kube–dns/CoreDNS upstream и stub:
  - нет - к `cluster.local` на внутренний DNS, остальное - на NSы с ноды
  - да - все запросы уходят на внутренний DNS кластера
- **ClusterFirstWithHostNet** - используется при `hostNetwork: true`
- **None** - настройка производится с помощью блока `dnsConfig`

☝️ По умолчанию используется **ClusterFirst**, а не **Default**

F...amous ndots problem

- По умолчанию, в `resolv.conf` добавляется опция `options ndots:5`
- Это значит, что DNS-запросы, где меньше 5 точек (или не FQDN) обрабатываются так:
  - дописываем домен из списка `search` и пробуем резолвить
  - если не вышло, берем следующий домен, снова пробуем
- Обычно, список такой

```
default.svc.cluster.local svc.cluster.local cluster.local
```

- Особенно больно это в Alpine (неполный список тут)

Подробно тут

Kube-proxy

- До версии 1.2 проксировал запросы через себя (userspace)
- "Слушает" все `NodePort` в root network namespace
- Следит за правилами iptables/ipvs, таким образом реальный трафик не попадает на `kube-proxy`

iptables vs IPVS

- **iptables** всегда выполняет фильтрацию трафика
- **ipvs** или **iptables** выполняют балансировку трафика
- **iptables** GA в 1.2, **IPVS** в 1.11 (пока не дефолт)
- Основное отличие в оптимизированной структуре хранения данных у LVS и iptables (Hashing vs Chains)
  - **iptables** дает ощутимую задержку роутинга при большом количестве правил (>1000 сервисов)
  - **iptables** при добавлении новых правил полностью перезаписывает структуру

iptables

- Пример балансировки с помощью **iptables**

```
-A KUBE-SVC-DRO76PSFYMKZCFJ3 -m statistic --mode random --probability 0.07691999990 -j
KUBE-SEP-HFCCAXWF7TEISIDM
... 10 ...
-A KUBE-SVC-DRO76PSFYMKZCFJ3 -m statistic --mode random --probability 0.50000000000 -j
KUBE-SEP-3VXKYXDWAVI52E5N
-A KUBE-SVC-DRO76PSFYMKZCFJ3 -j KUBE-SEP-6TJMS6HHN77ADPYT
```

- Количество правил **iptables** при 168 подах и 69 сервисах

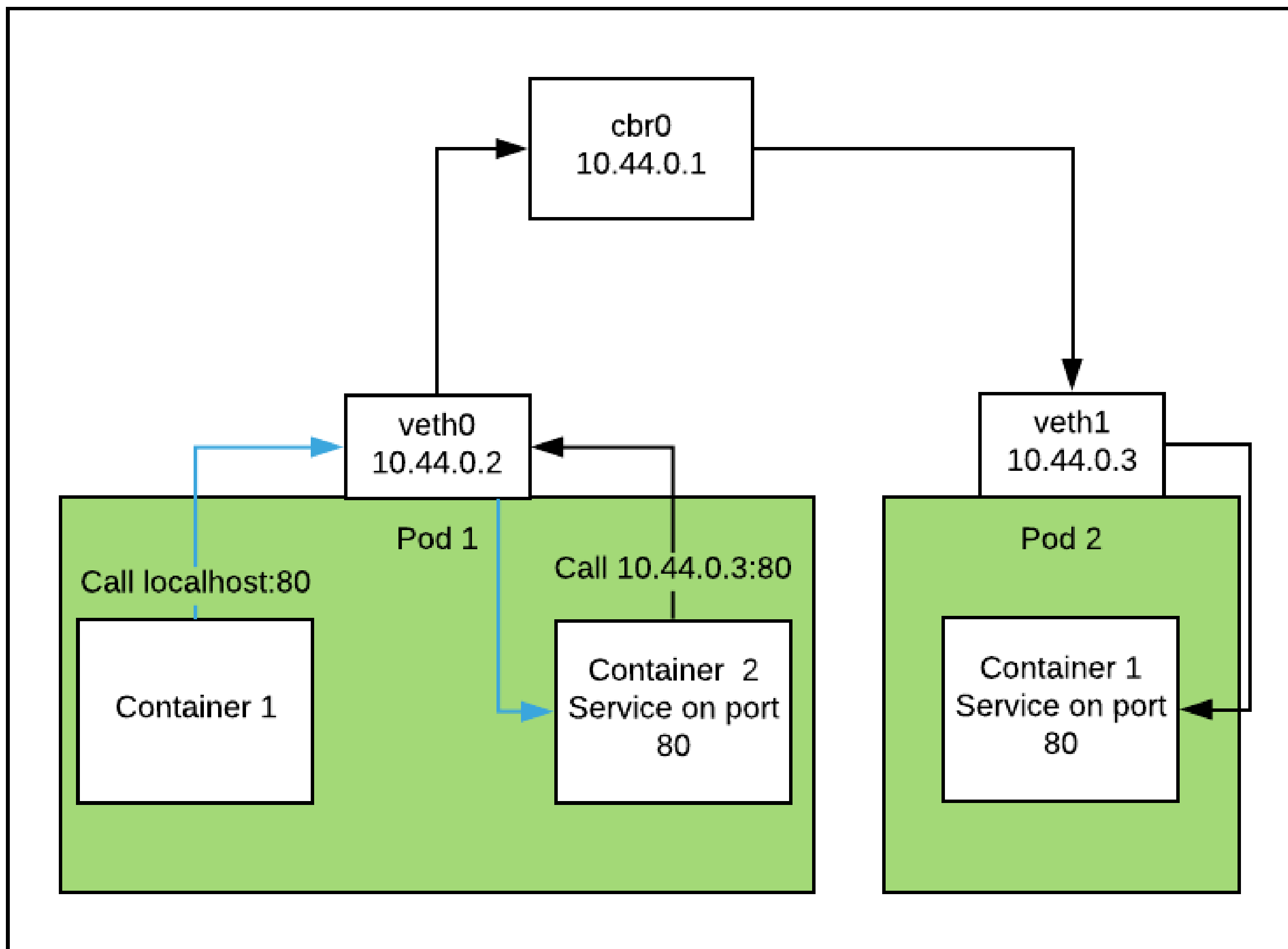
```
$ iptables-save | wc -l
902
```

Взаимодействие контейнеров

Внутри одного пода

- Pod смертен и может быть пересоздан в любой момент
- Pod всегда имеет IP-адрес, но он может меняться при смерти/переезде
- Контейнеры внутри одного pod могут общаться через
  - Loopback-интерфейс (он же `localhost`, он же `127.0.0.1`)
  - UNIX Sockets, `/dev/shm`, named pipes
  - Pod IP
- Для общения контейнеров внутри одного Pod не требуется описывать порты
- Все контейнеры могут видеть весь сетевой трафик пода 🦈

Внутри одного хоста



Внутри одного хоста

- Стандартный Linux bridge (или нет - зависит от CNI)
- По умолчанию всем разрешено всё
- Для ограничений используются `PodNetworkPolicy`
- Для связи между Pod требуется знать их IP-адреса или использовать `Service`
- Так же возможно использовать `hostNetwork`
- Внешний доступ с использованием SNAT на host IP (зависит от CNI)

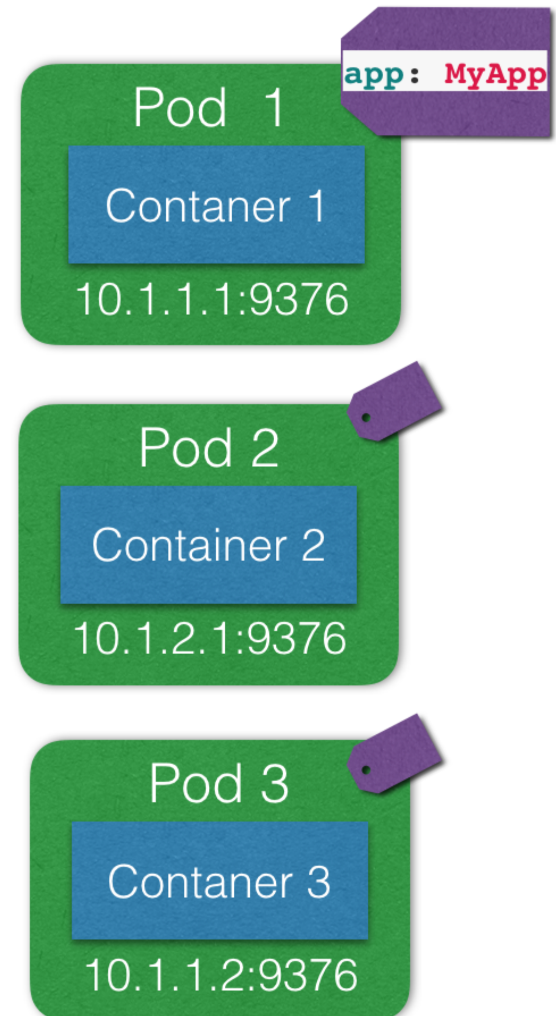
Внутри одного кластера

Service

- Абстракция, описывающая набор pod'ов и конфигурацию доступа к ним
- Позволяет отвязаться от использования конкретных pod'ов
- `Service` не содержит списка подов, а использует дополнительную абстракцию `Endpoints`
- `Endpoints` - список адресов, на которые `Service` может перенаправить трафик
- Основные типы `Service` используемые внутри кластера:
  - `ClusterIP`
  - `Headless`
  - `ExternalName`
- `sessionAffinity` - `None`, `ClientIP`

Service with selector

```yaml
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```



Service without selector

```yaml
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

```yaml
kind: Endpoints
apiVersion: v1
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 1.2.3.4
    ports:
      - port: 9376
```

Service on iptables

Пример сервиса реализованного iptables с 3 endpoint:

```
*nat

-A KUBE-SERVICES -d 10.11.249.128/32 -p tcp -m comment --comment "default/post: cluster
IP" -m tcp --dport 5000 -j KUBE-SVC-ES3BC673JESWV6HT

-A KUBE-SVC-ES3BC673JESWV6HT -m comment --comment "default/post:" -m statistic --mode
random --probability 0.33332999982 -j KUBE-SEP-O4GBVVV2IUYGM642

-A KUBE-SVC-ES3BC673JESWV6HT -m comment --comment "default/post:" -m statistic --mode
random --probability 0.50000000000 -j KUBE-SEP-RRVZNCBLSP23WDST

-A KUBE-SVC-ES3BC673JESWV6HT -m comment --comment "default/post:" -j KUBE-SEP-
TOVXRQYGCLRJPZKA

-A KUBE-SEP-O4GBVVV2IUYGM642 -p tcp -m comment --comment "default/post:" -m tcp -j DNAT
--to-destination 10.8.0.16:5000
```

ExternalName Service

- Редирект на уровне DNS (прокси не используется)
- Возвращает CNAME-запись

```
my-service.prod.svc.cluster.local => my.database.example.com
my-service => my.database.example.com
```

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

Headless Service

- Работает как Round Robin DNS
- Это `Service` с `ClusterIP: None`
- Возвращает адреса pod'ов (если задан селектор)
- Возвращает CNAME-записи (для типа `ExternalName`)
- Возвращает адреса всех одноименных сервису `Endpoints` (во всех остальных случаях)

Headless Service

Service:

```
$ nslookup web-service

Name:    web-service.default.svc.cluster.local
Address: 10.55.251.238
```

Headless Service:

```
$ nslookup web-service-headless

Name:    web-service-headless.default.svc.cluster.local
Address: 10.52.1.3
Name:    web-service-headless.default.svc.cluster.local
Address: 10.52.2.2
Name:    web-service-headless.default.svc.cluster.local
Address: 10.52.2.4
```

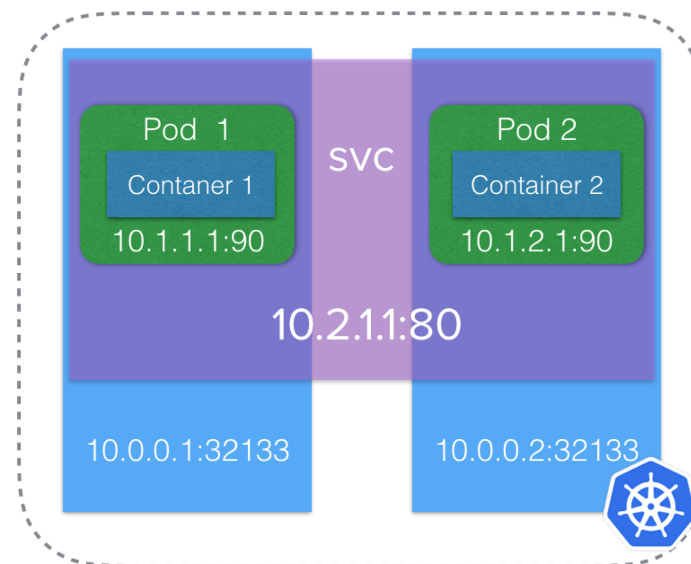
Внешний доступ к ресурсам кластера

- Взаимодействие внешнего мира с ресурсами кластера
- Используются следующие типы ресурсов `Service`
  - `NodePort`
  - `LoadBalancer`
- Или ресурсы типа `Ingress`
- Через `externalIPs` в конфигурации сервиса

NodePort

Тип сервиса NodePort является "надстройкой" над ClusterIP

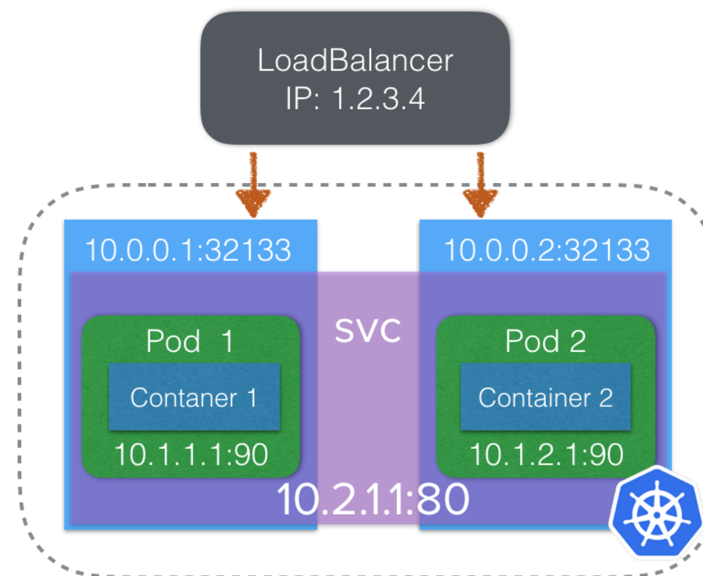
```yaml
kind: Service
apiVersion: v1
metadata:
  name: svc
spec:
  type: NodePort
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 90
    nodePort: 32133
```



LoadBalancer

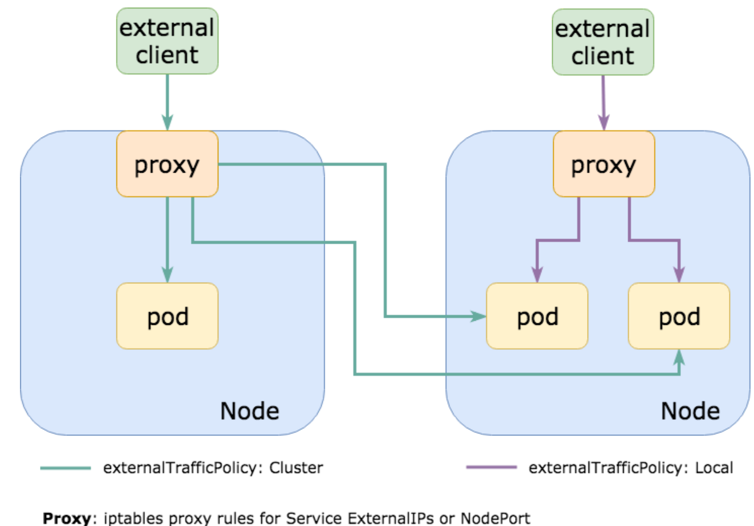
Тип сервиса LoadBalancer является "надстройкой" над NodePort

```
kind: Service
apiVersion: v1
metadata:
  name: svc
spec:
  type: LoadBalancer
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 90
```



LoadBalancer | externalTrafficPolicy

- **Cluster** (используется по умолчанию)
  - Размазывает "трафик" по всех хостам, которые размазывают по всем подам
- **Local**
  - Трафик приходит только на узлы, где запущены нужные поды
  - Реализовано через `kube-proxy` (проверки `/healthz`)



LoadBalancer | Облака

- AWS
  - Classic Load Balancer (default) - L3/L4/L7, работает поверх `NodePort`
  - Network Load Balancer - L3/L4, работает поверх `NodePort`
  - Application Load Balancer - L7, умеет балансировать напрямую в поды
- GKE
  - NetworkLoadBalancer - L3/L4, работает через `NodePort`
  - HTTPLoadBalancer - L7, работает через `Ingress`

LoadBalancer | Bare Metal

- Реализация через CNI
  - Не совсем LB, а обычный сервис доступный снаружи кластера
  - Например Calico, kube-router
- Честный `LoadBalancer`
  - MetalLB (Beta)

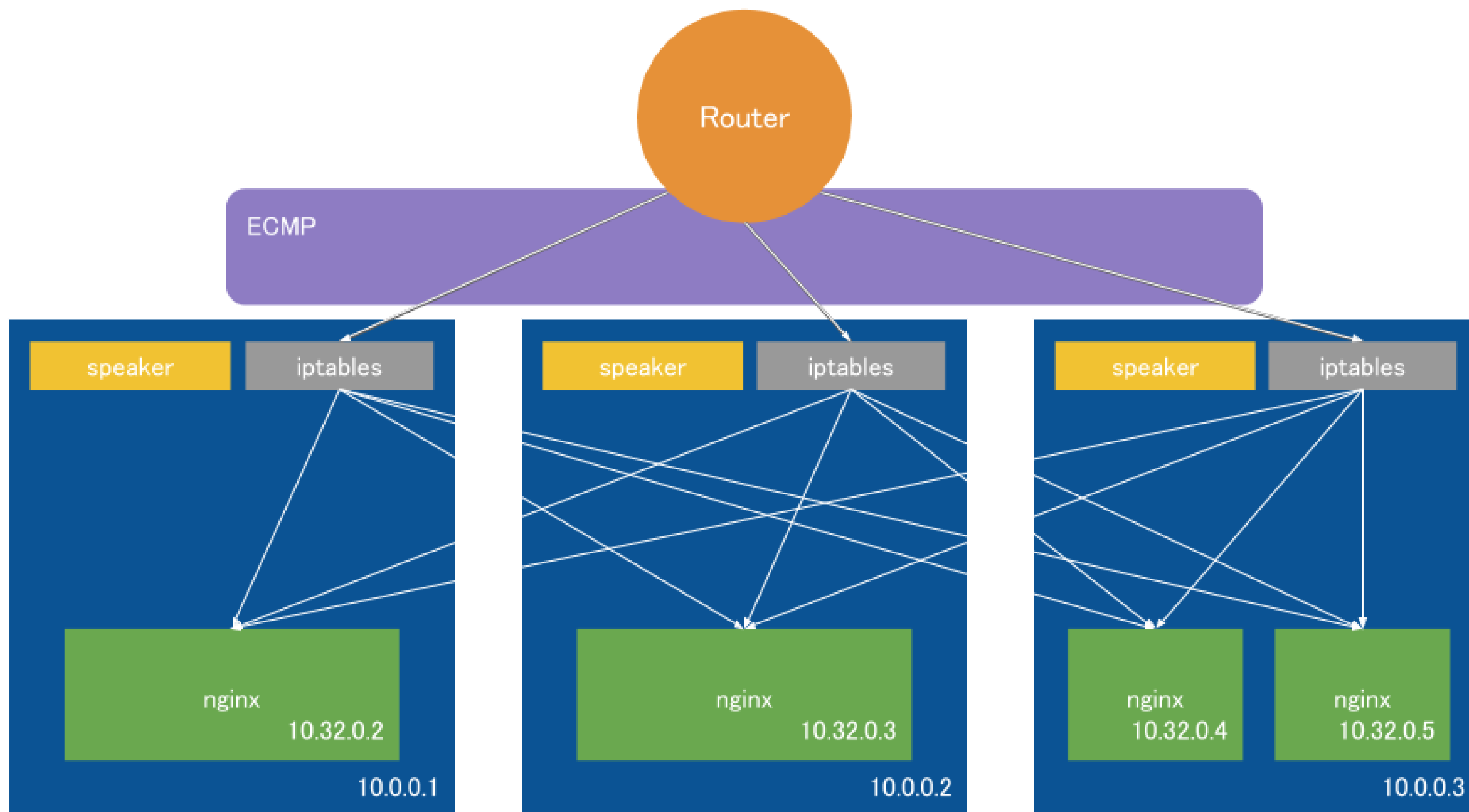
MetalLB

- Состоит из двух компонентов:
  - **Controller** - управляюший пулами адресов и Speaker-ами
  - **Speaker** - компонент запущенный на всех или части узлов кластера, отвечает за взаимодействие с внешними сетями
- Два режима работы:
  - **L2-режим** - похож на VRRP/HSRP. Среди спикеров выбирается "лидер" который отвечает на ARP-запросы к IP-адресу из пула.
  - **L3-режим** - каждый Speaker устанавливает BGP-сессию с сетевым оборудованием (например, ToR-коммутатором). Дальше работает ECMP (Equal-Cost Multipath)

MetalLB

- Используется собственная реализация BGP-протокола ("только на вход", никаких лишних плюшек)
- MetalLB отслеживает только сервисы с типом `LoadBalancer`
  - Он не следит (и не будет) обрабатывать ресурсы типа `Ingress`
  - Предполагается, что администратор сам установит и настроит Ingress-прокси (`ingress-nginx`, Kong и т.д.)
  - И вот для этих Ingress-прокси уже создается сервис `LoadBalancer`

MetalLB



Ingress

Объект управляющий внешним доступом к сервисам внутри кластера. Обеспечивает:

- Организацию единой точки входа в приложения снаружи
- Балансировку трафика
- Терминацию SSL
- Виртуальный хостинг на основе имен и т.д.

Работает на L7 уровне.

Ingress Controller

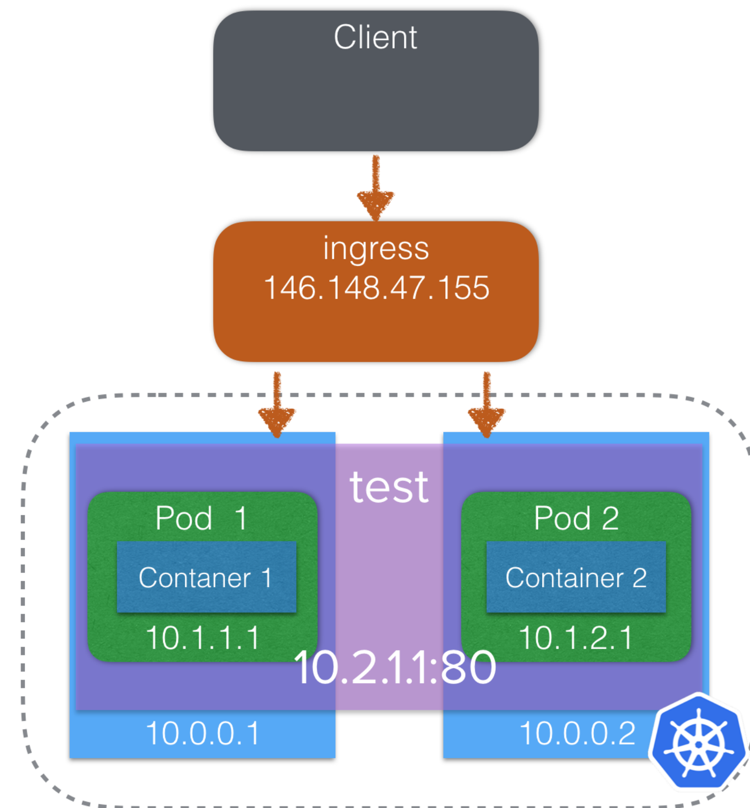
Ingress - набор правил внутри кластера Kubernetes.

Для применения данных правил нужен **Ingress Controller** - плагин который состоит из 2-х функциональных частей:

- Приложение, которое отслеживает через Kubernetes API новые объекты Ingress и обновляет конфигурацию балансировщика
- Балансировщик (nginx, HAProxy, traefik,...), который отвечает за управление сетевым трафиком

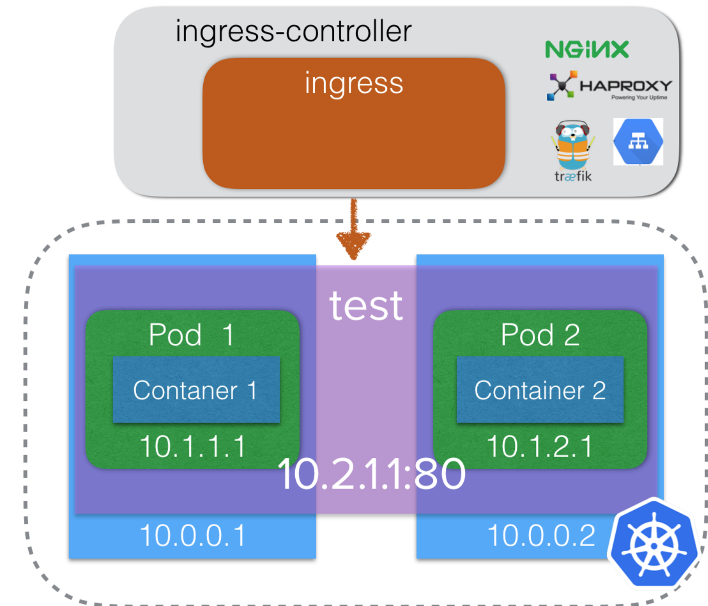
Ingress

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        backend:
          serviceName: test
          servicePort: 80
```



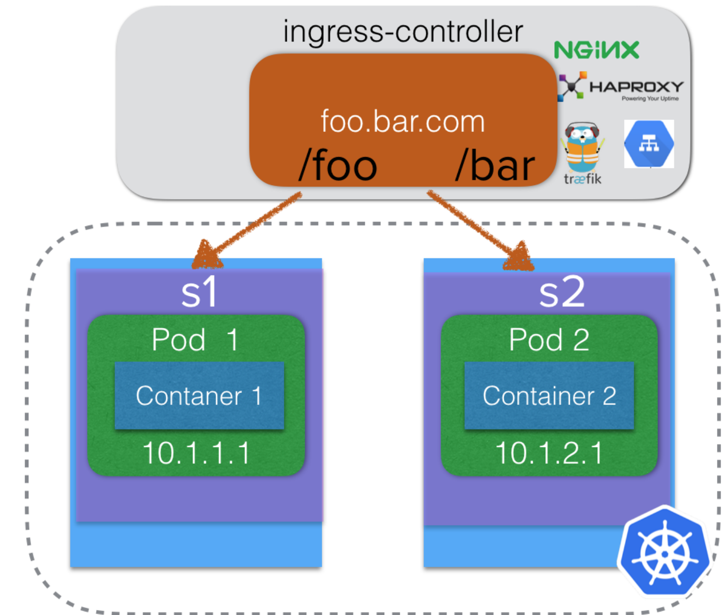
Single Service Ingress

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress
spec:
  backend:
    serviceName: test
    servicePort: 80
```



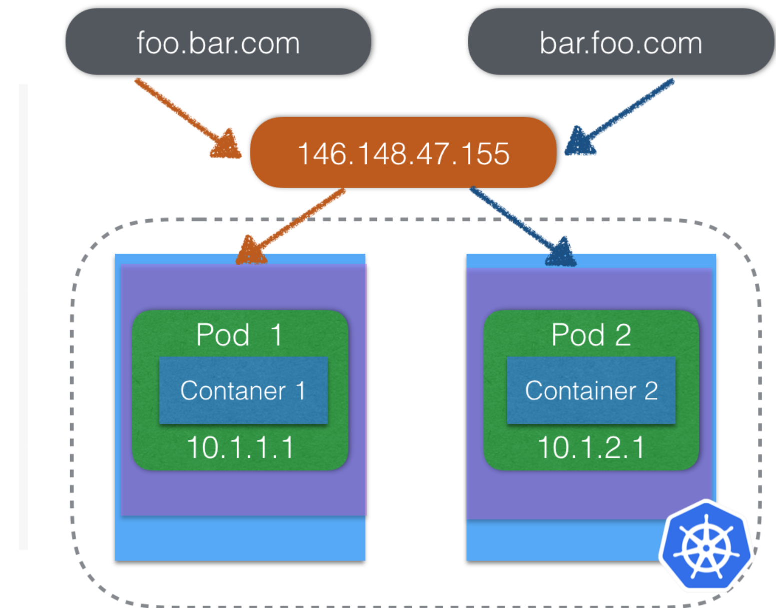
Simple Fanout

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: s1
          servicePort: 80
      - path: /bar
        backend:
          serviceName: s2
          servicePort: 80
```



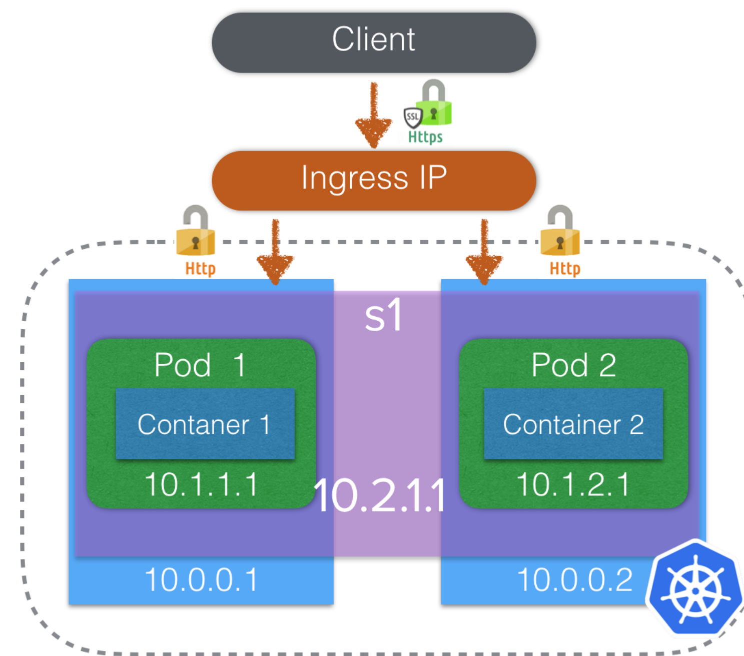
Name Based Virtual Hosting

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          serviceName: s1
          servicePort: 80
  - host: bar.foo.com
    http:
      paths:
      - backend:
          serviceName: s2
          servicePort: 80
```



TLS termination

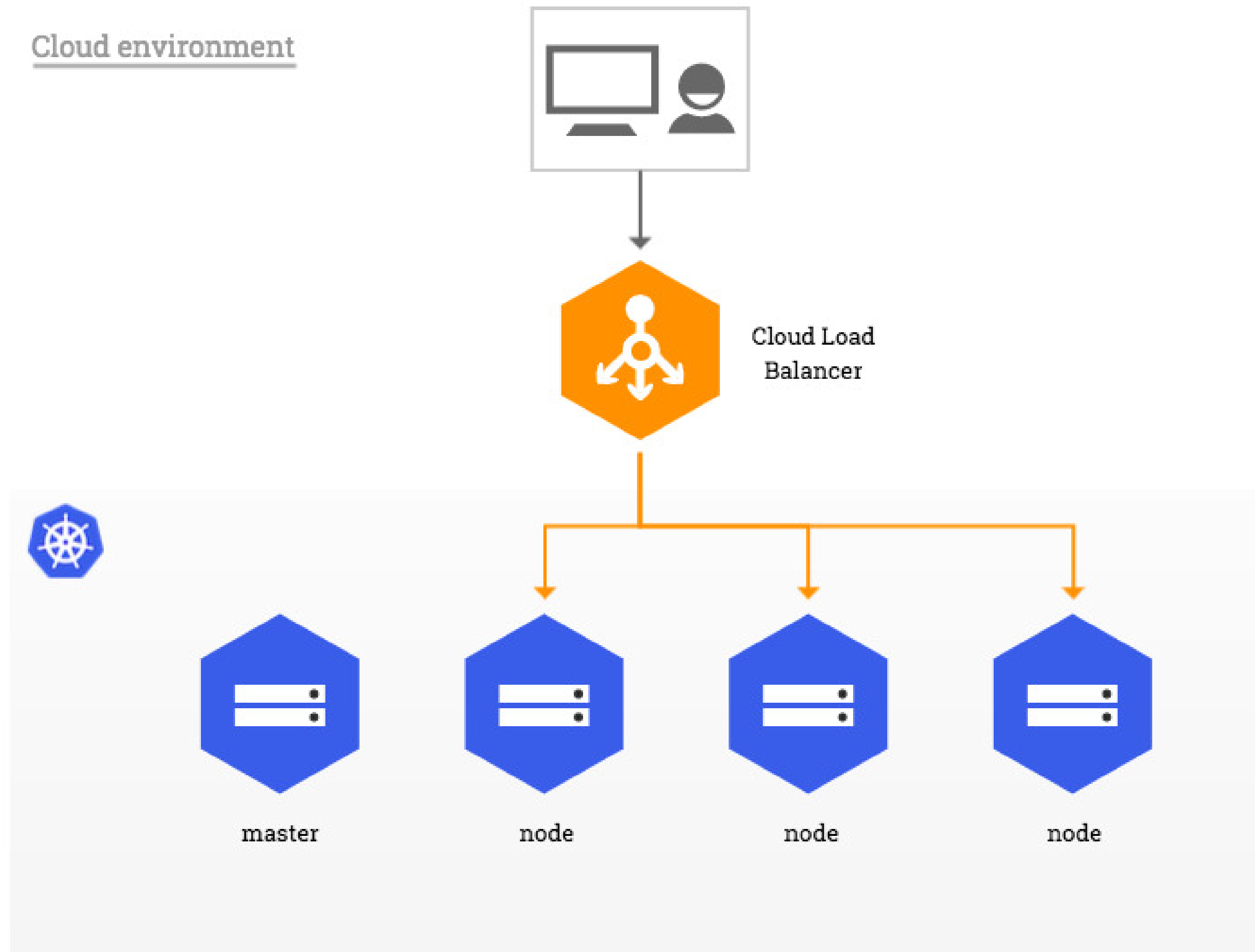
```
apiVersion: v1
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
kind: Secret
metadata:
  name: testsecret-tls
  namespace: default


apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: tls-example-ingress
spec:
  tls:
  - hosts:
    - sslexample.foo.com
    secretName: testsecret-tls
  rules:
  ...
```



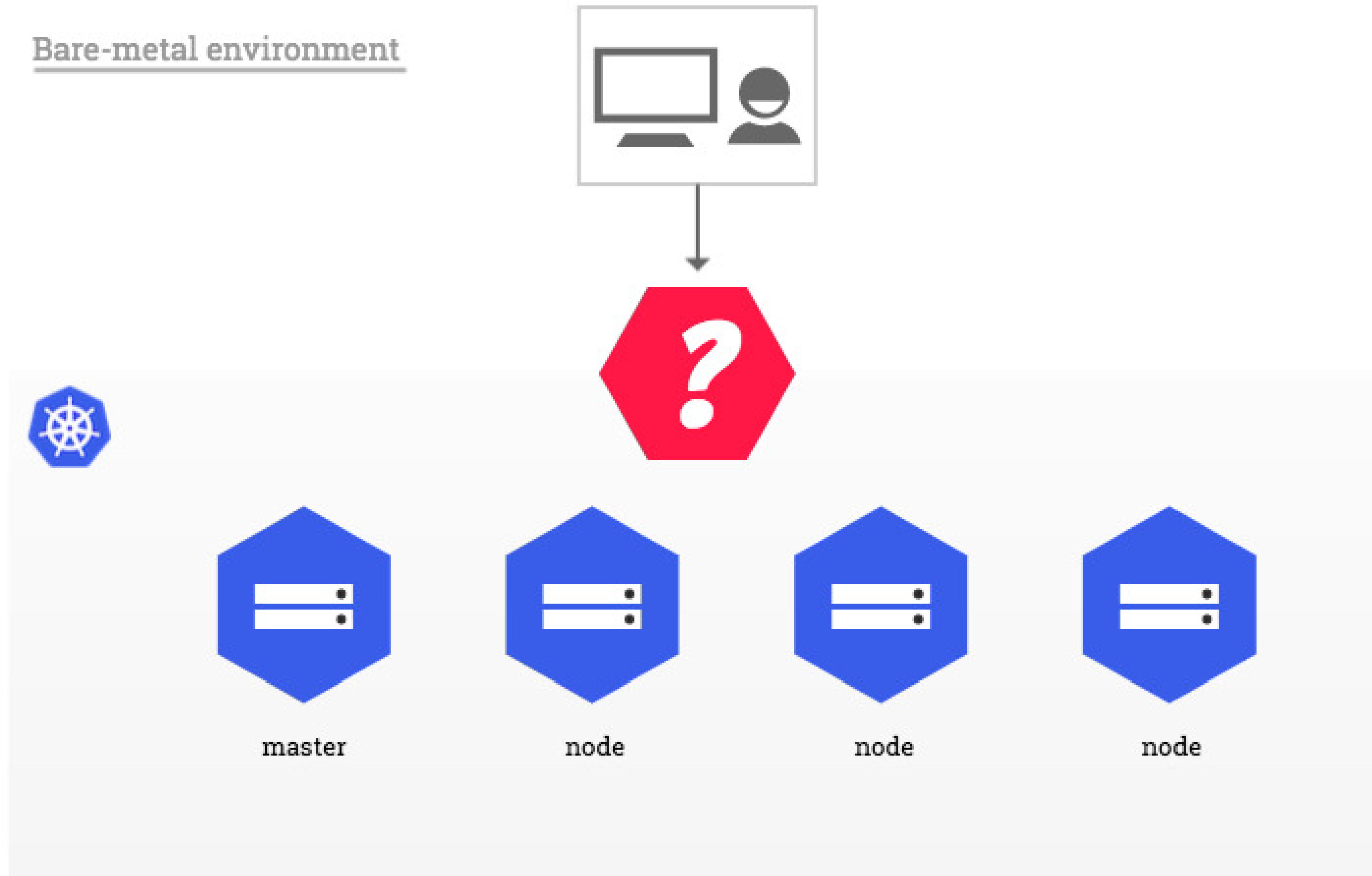
Ingress

[Краткий обзор различных Ingress и сводная таблица по их возможностям]

Ingress



Ingress

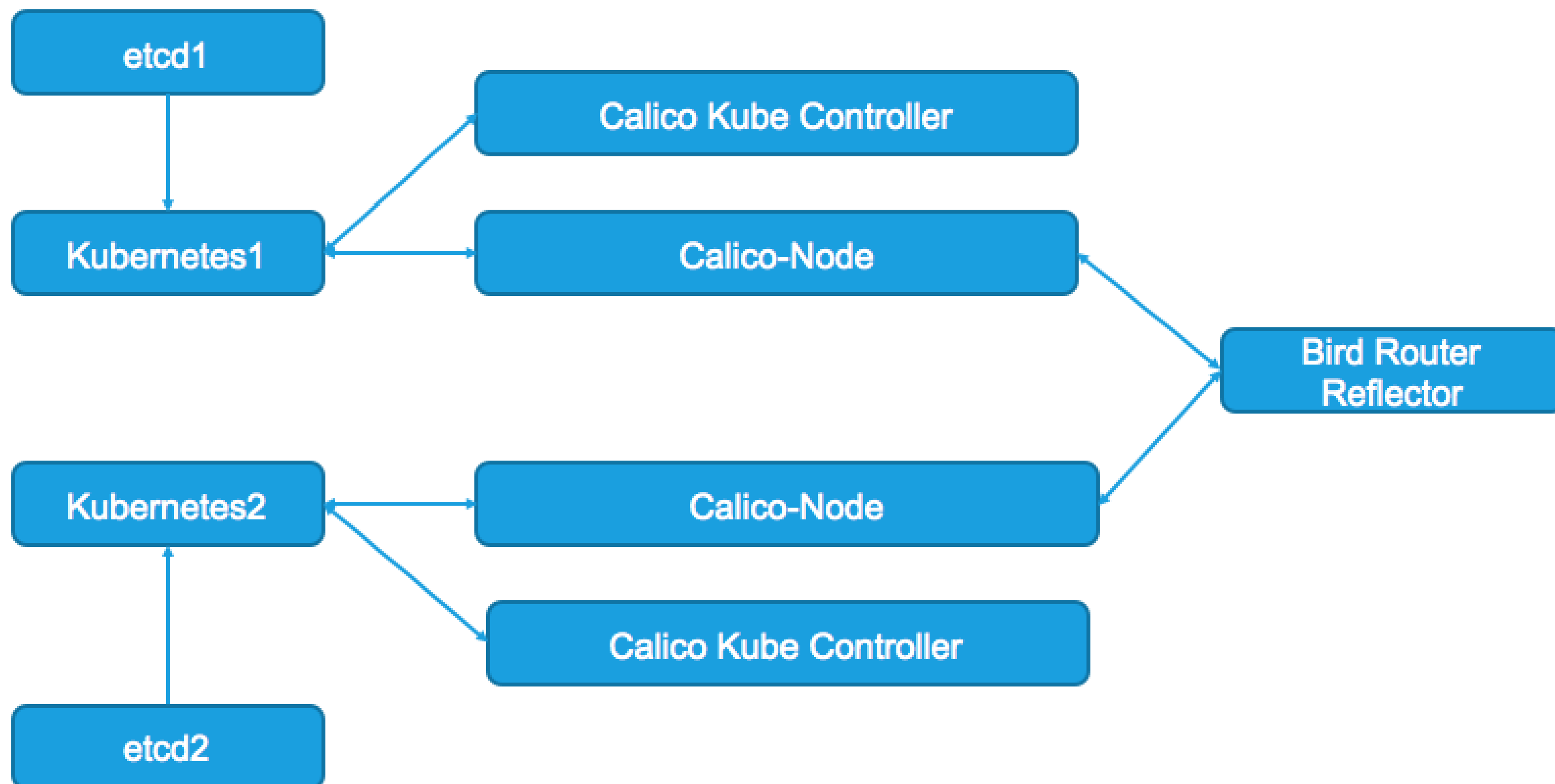


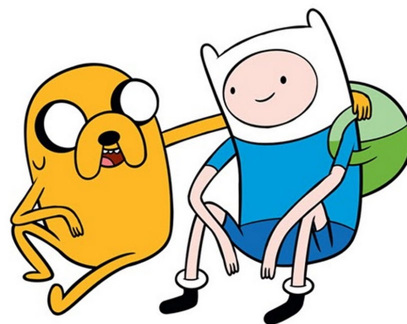
Связность нескольких кластеров

- По возможности используем стандартные средства LB и Ingress
  - Модно-молодежный вариант - Service Mesh Gateway
- Ручной способ - настройка маршрутов/туннелей между кластерами вручную
- Используя функционал CNI, например Calico
- В случае использования реальных IP для подов и сервисов - достаточно настроить CoreDNS

Вариант Service Mesh Gateway

Связность нескольких кластеров





Спасибо за внимание!

Время для ваших вопросов!