# Kubernetes.
# Сетевое взаимодействие.
# Хранение данных.

# План

- Сетевое взаимодействие в Kubernetes
- Хранение данных в Kubernetes

# Сетевое взаимодействие в Kubernetes

# Pods

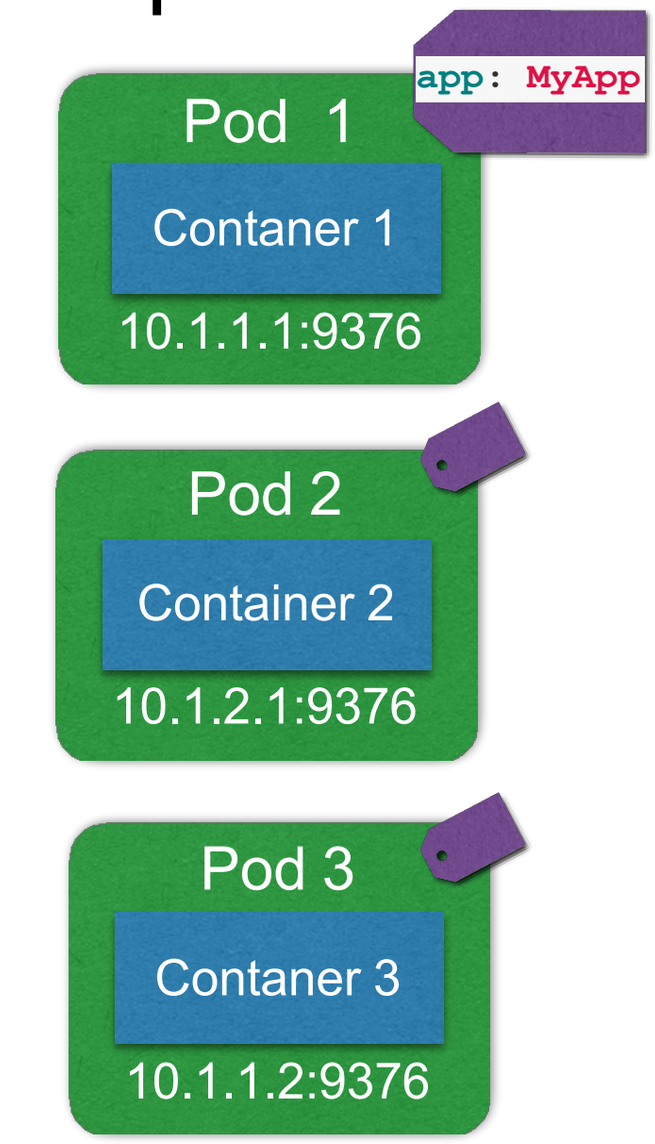
- Смертны
- Динамически создаются и удаляются
- IP-адреса появляются и исчезают вместе с Pod'ами
- Несколько Pod'ов могут выполнять одну и ту же функцию

# Service

- Абстракция, описывающая набор Pod'ов и конфигурацию доступа к ним
- Позволяет отвязаться от использования конкретных Pod'ов

# 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
```

# ExternalName service

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

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

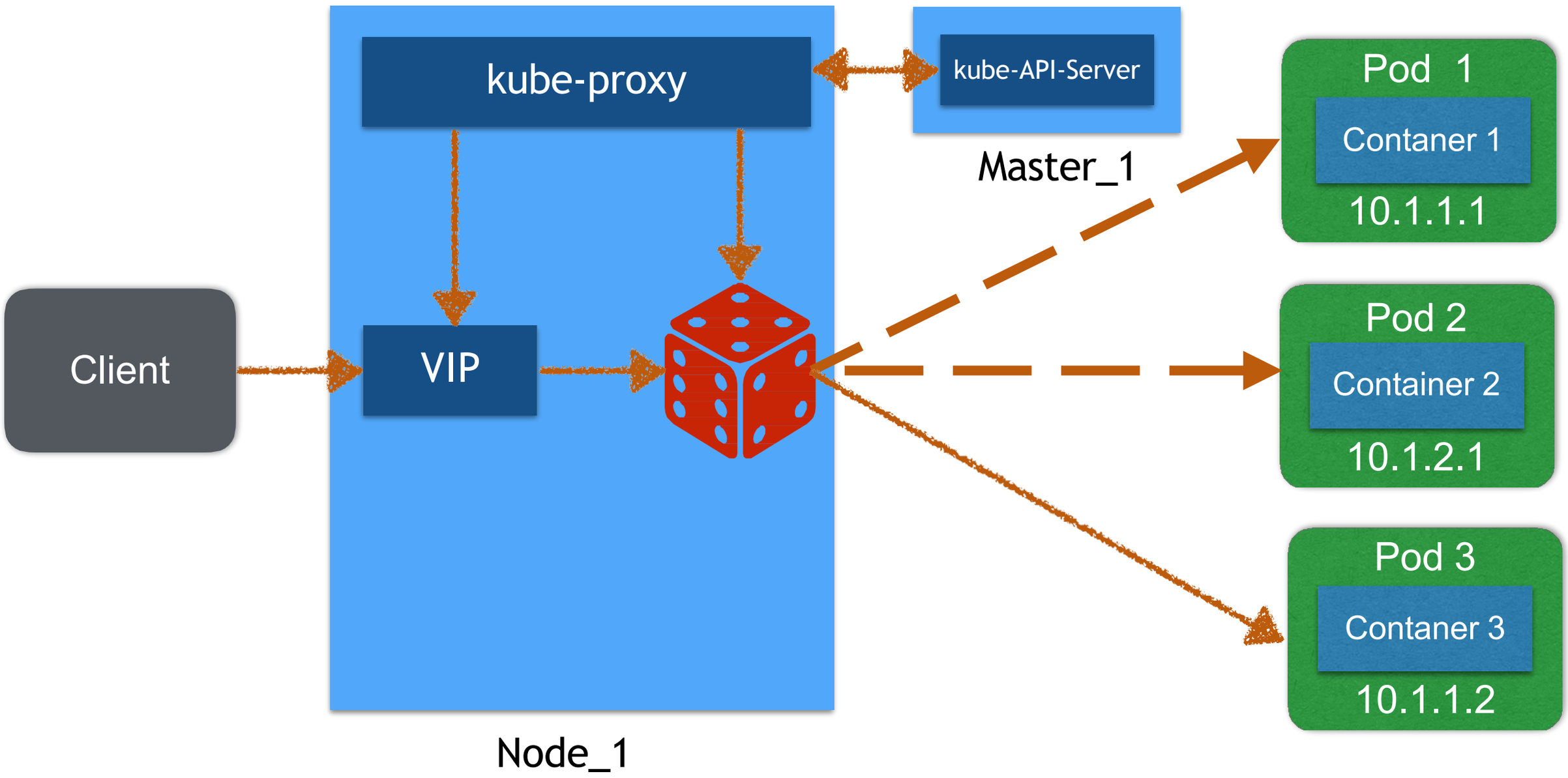
# Headless Service

- Service без Cluster IP
- DNS возвращает адреса Pod'ов (Service with Selector)
- Возвращает CNAME запись (для ExternalName Service)
- Возвращает адреса всех одноименных сервису Endpoint'ов (во всех остальных случаях)

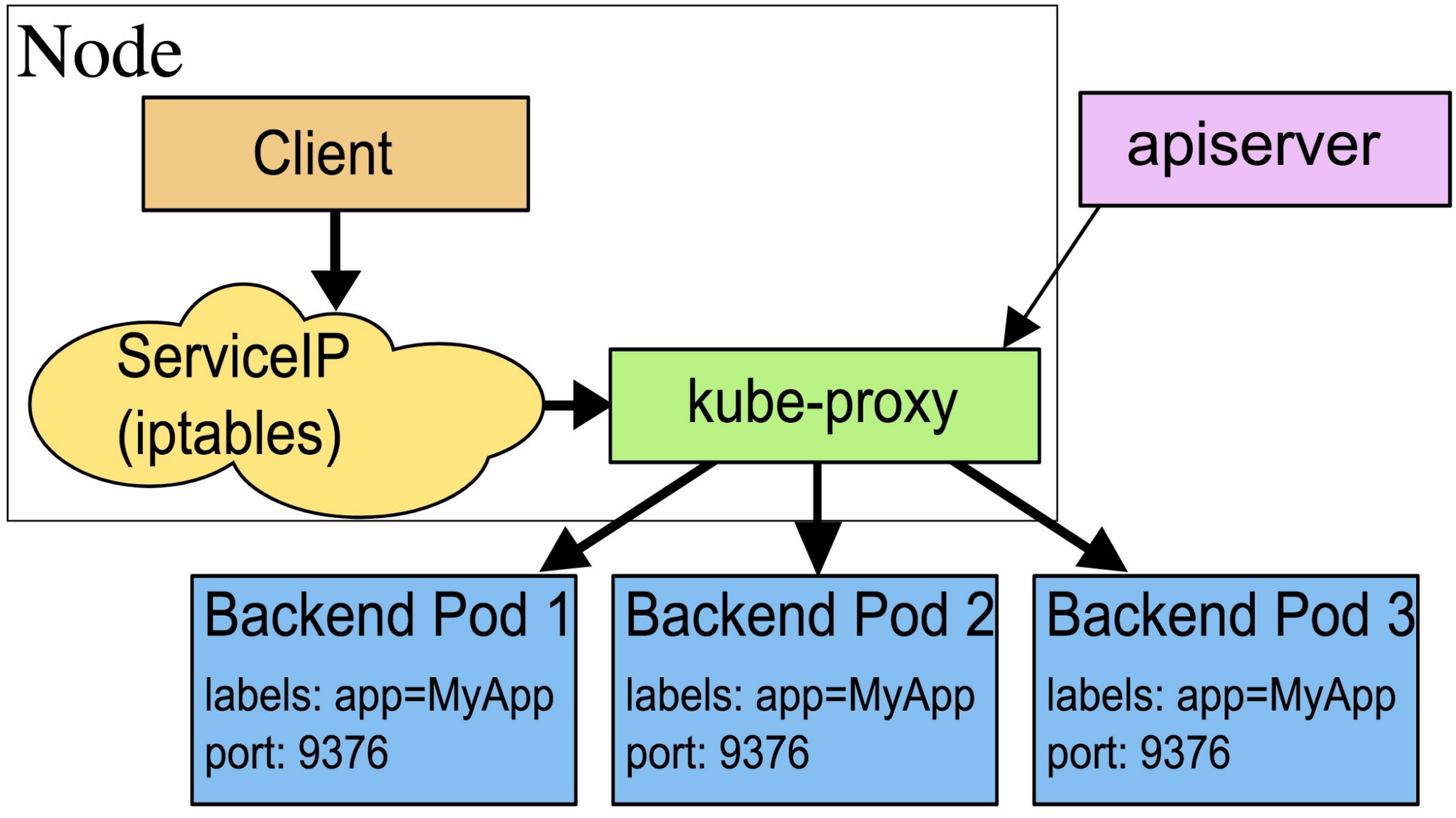
# kube-proxy

- Проксирует TCP и UPD (не HTTP)
- Используется только, для работы с сервисами
- Обеспечивает внутренюю балансировку
- Используется для внутренней коммуникации

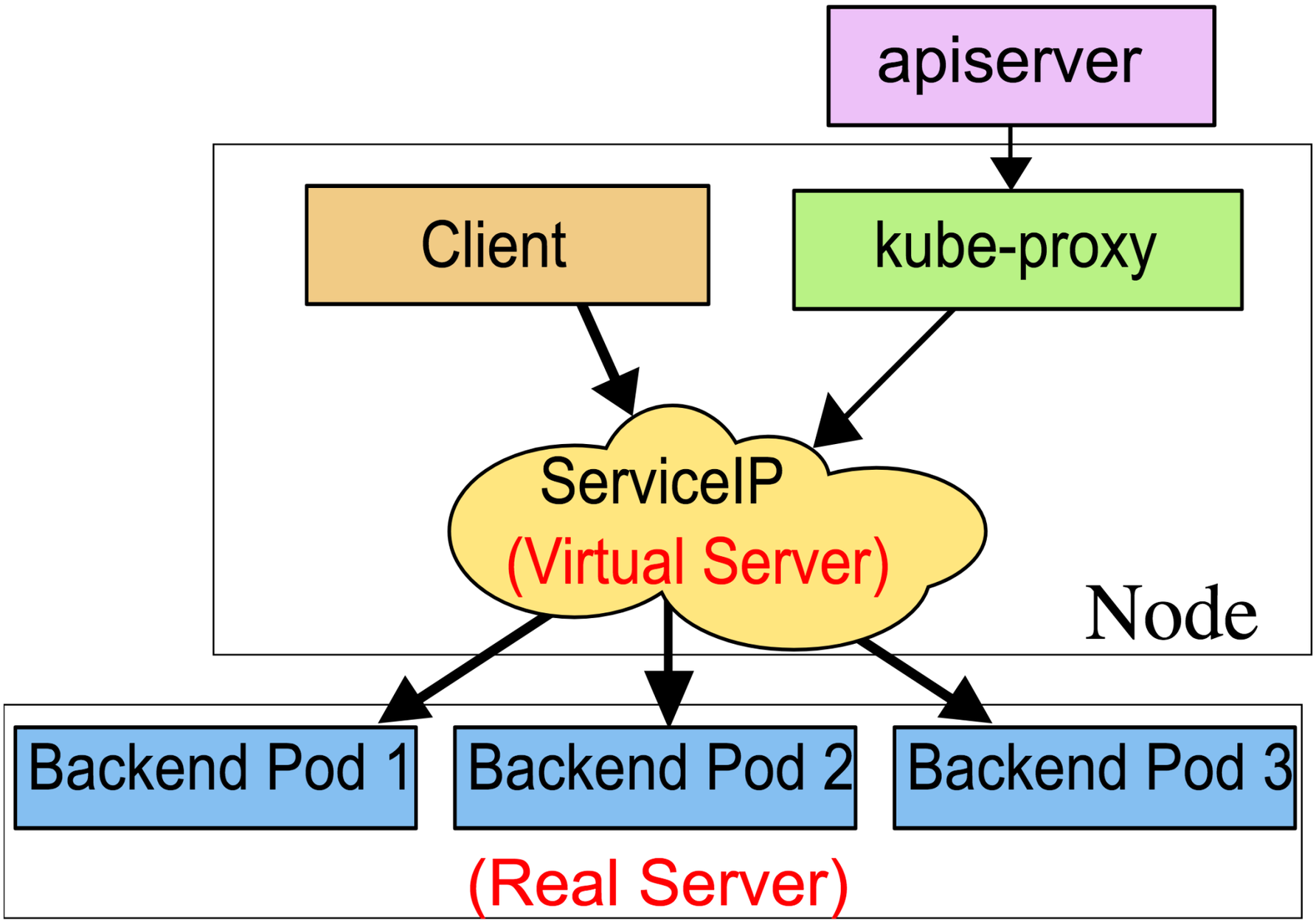
# kube-proxy


kube-proxy
kube-API-Server
Master_1
Client
VIP
Node_1
Pod 1
Contaner 1
10.1.1.1
Pod 2
Container 2
10.1.2.1
Pod 3
Contaner 3
10.1.1.2

# kube-proxy: Userspace



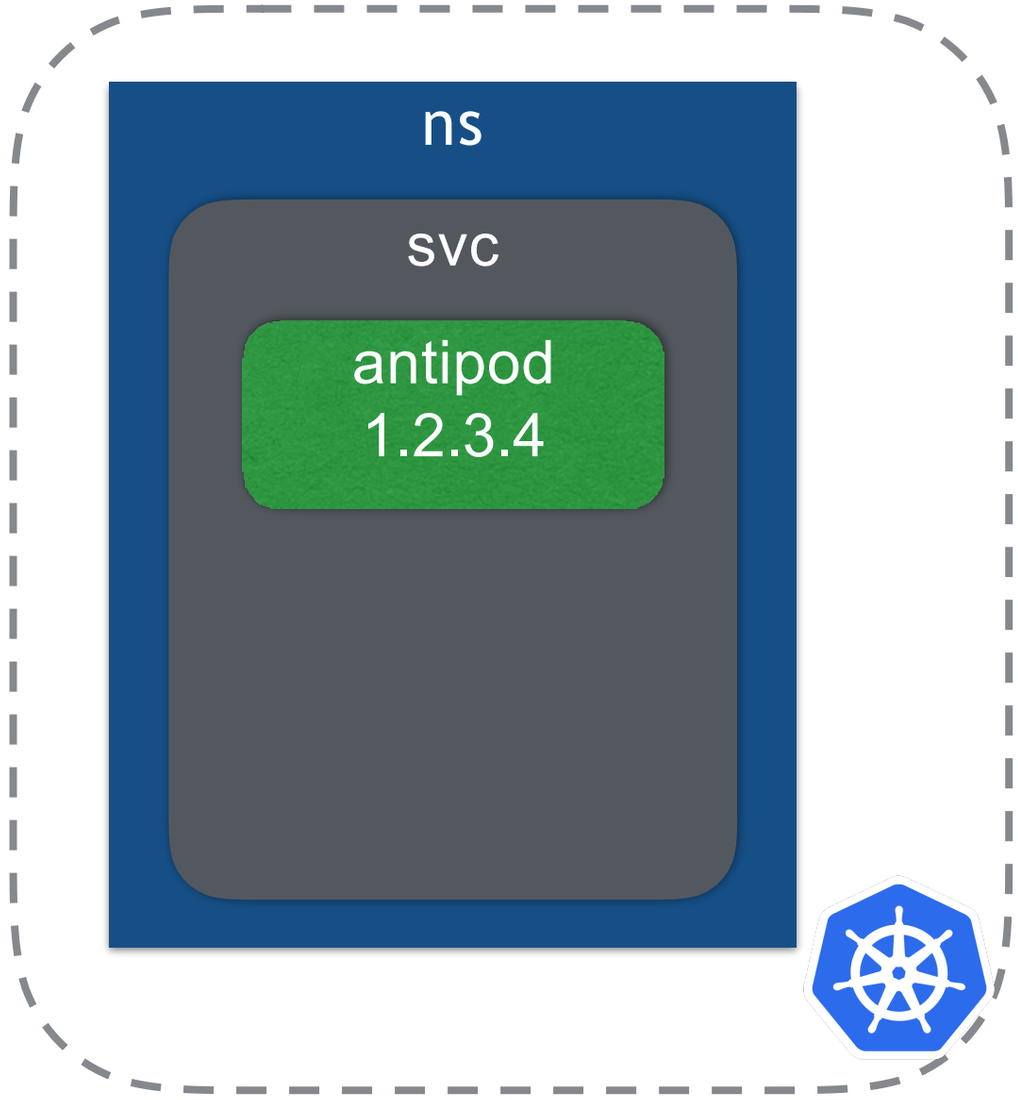
# kube-proxy: IPVS



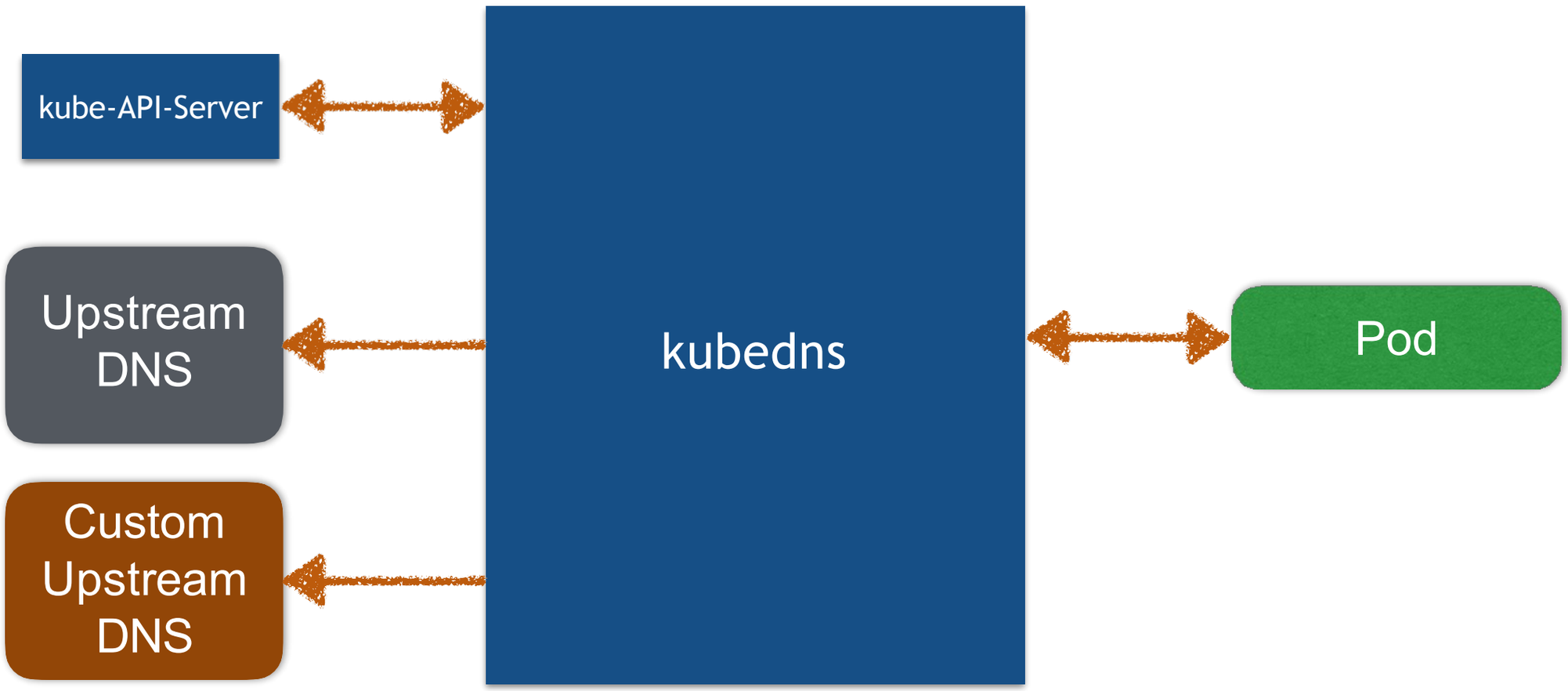

Подробнее: https://dustinspecker.com/posts/ipvs-how-kubernetes-services-direct-traffic-to-pods/

# kubedns

- Namespace
  - svc
- Cluster
  - svc.ns
- FQDN
  - svc.ns.svc.cluster.local
- Pod FQDN
  - 1-2-3-4.ns.pod.cluster.local



# kubedns


kube-API-Server
kubedns
Upstream DNS
Custom Upstream DNS
Pod

# kubedns config

```
apiVersion: v1
kind: ConfigMap
metadata:
   name: kube-dns
   namespace: kube-system
data:
   stubDomains: |
     {"acme.local": ["1.2.3.4"]}
   upstreamNameservers: |
     ["8.8.8.8", "8.8.4.4"].
```

# dnsPolicy

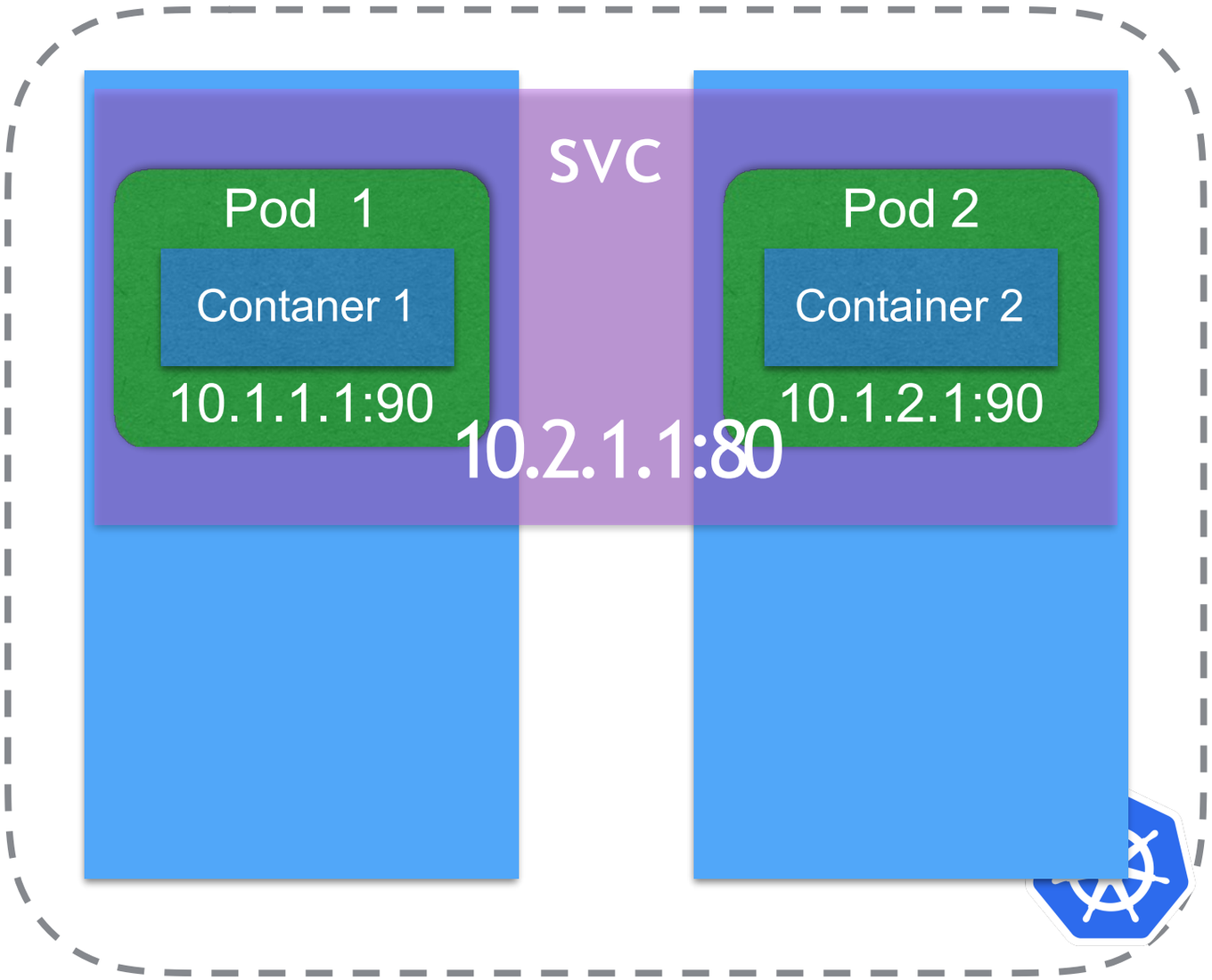
- Default - разрешение имен наследуется с хоста на котором запущен Pod.
  Нельзя использовать upstreamNameservers и stubDomains
- ClusterFirst - разрешение имен происходит через kubedns и upstream сервера

# Publishing Service

- ClusterIP - только внутри кластера
- NodePort - на порты нод кластера
- LoadBalancer - на внешнем сервисе балансировки
- ExternalIP - указатель на IP-адрес, по которому доступна одни или несколько Node кластера

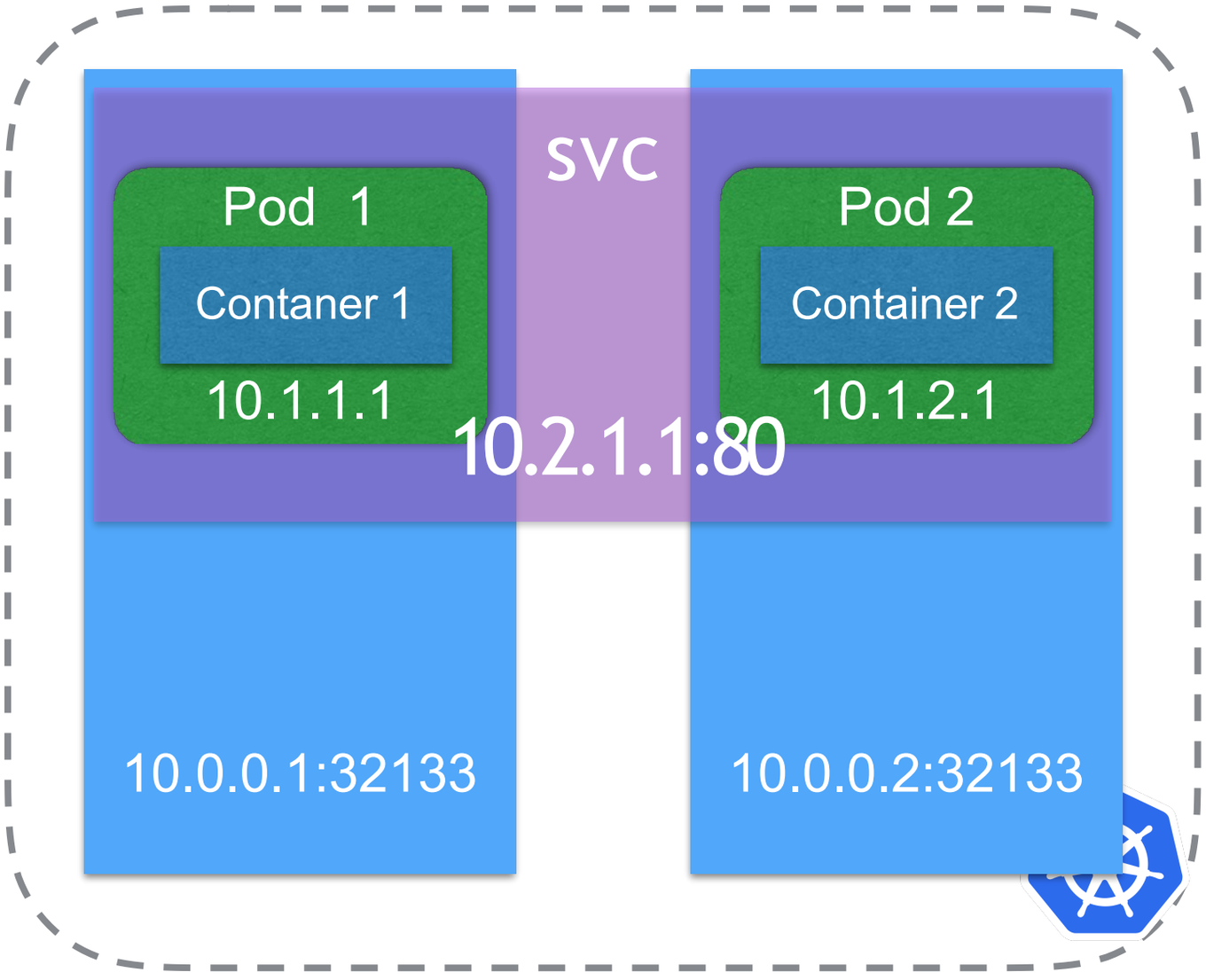
# ClusterIP

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



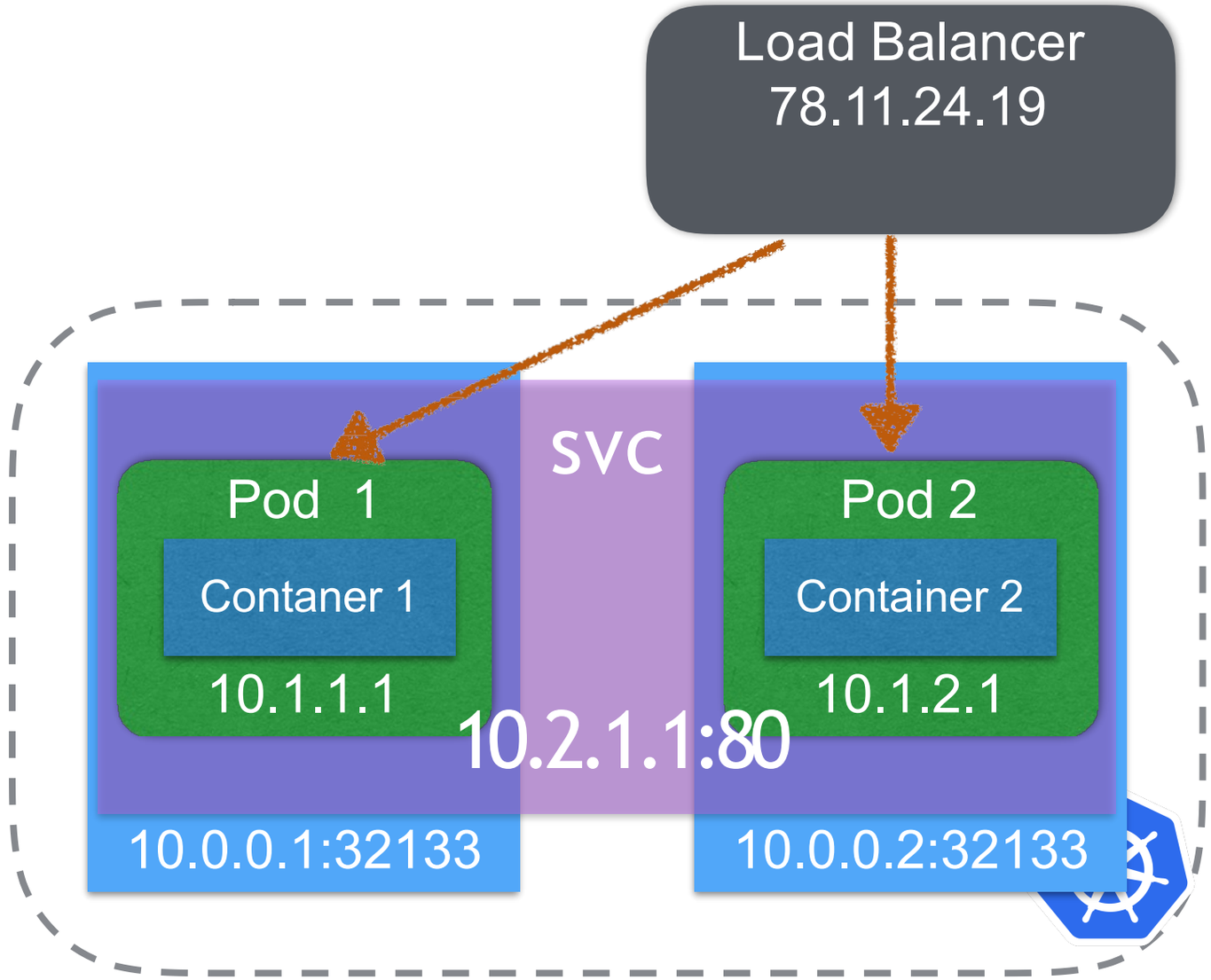
# NodePort

```yaml
kind: Service
apiVersion: v1
metadata:
  name: svc
spec:
  type: NodePort
  selector:
    app: MyApp
  ports:
  - port: 80
```



# LoadBalancer

```yaml
kind: Service
apiVersion: v1
metadata:
  name: svc
spec:
  type: LoadBalancer
  selector:
    app: MyApp
  ports:
  - port: 80
  clusterIP: 10.2.1.1
  loadBalancerIP: 78.11.24.19
```

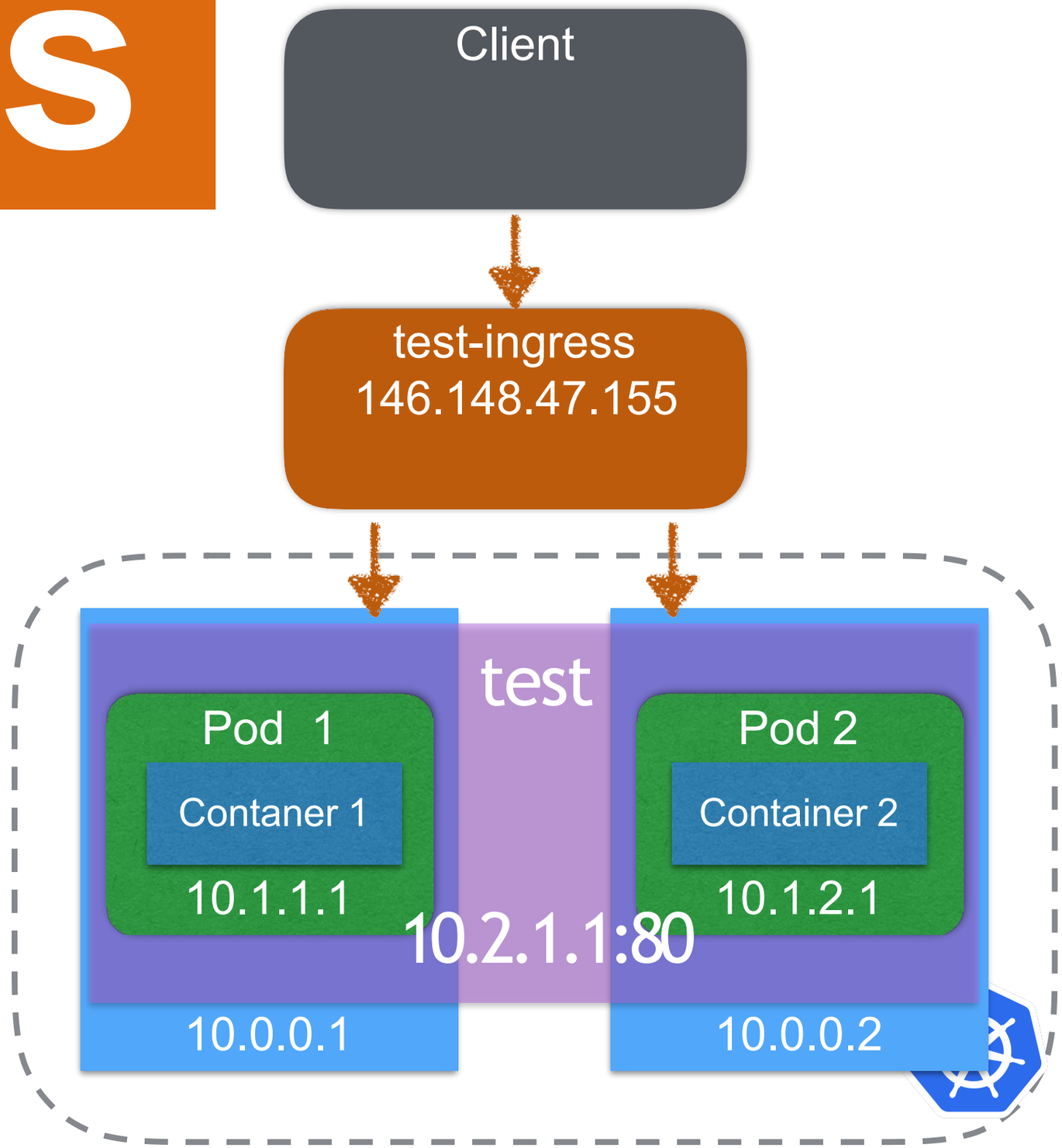


# Ingress

- Ingress - объект управляющий внешним доступом к сервисам внутри кластера. Обеспечивает:
  - Балансировку нагрузки
  - Терминацию SSL трафика
  - Name based virtual hosting
- Работает на 7 уровне OSI

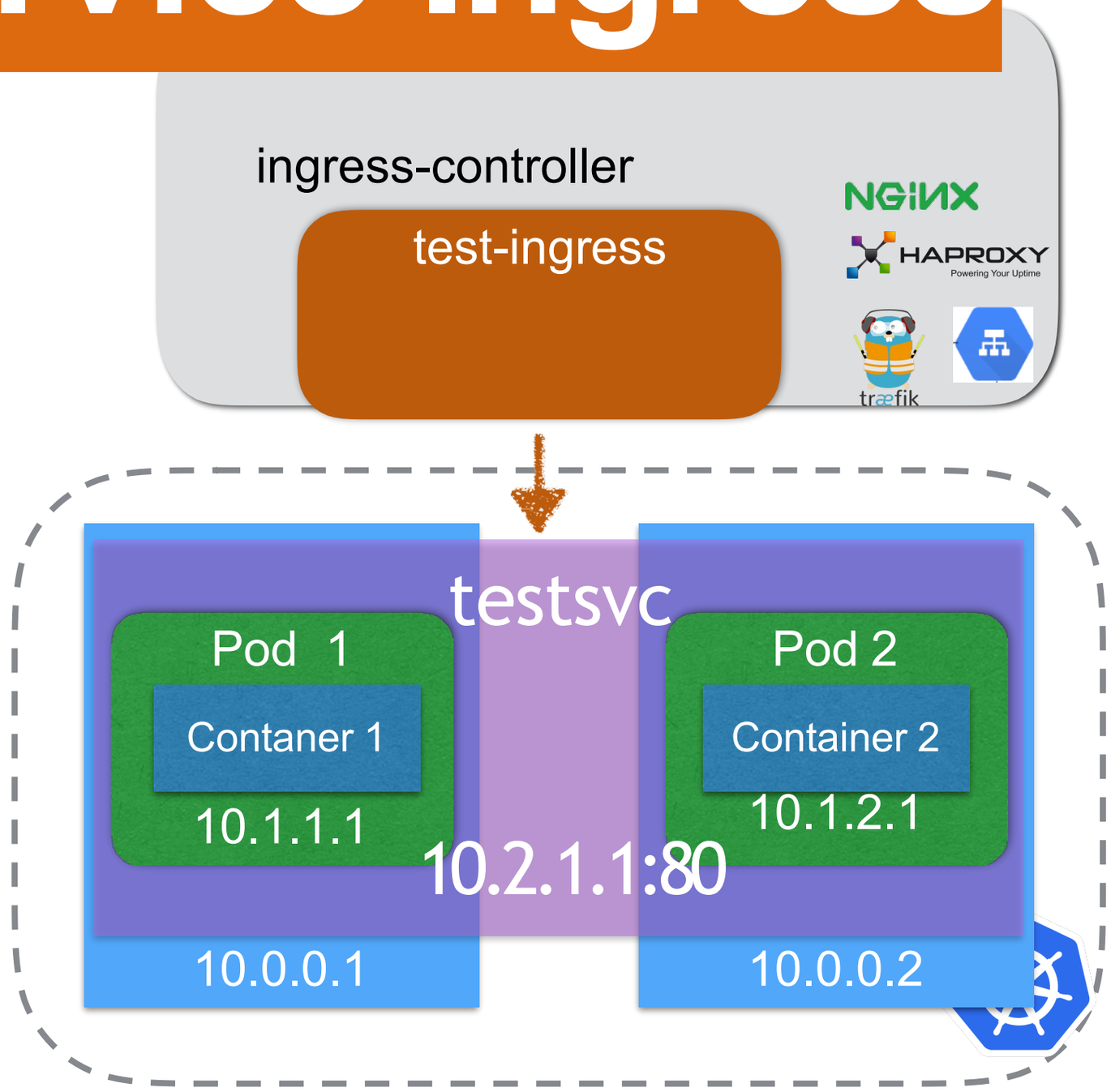
# Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-ingress
spec:
  rules:
  - host: aaa.ru
    http:
      paths:
      - path: /testpath
        backend:
          service:
            name: test
            port: 80
```



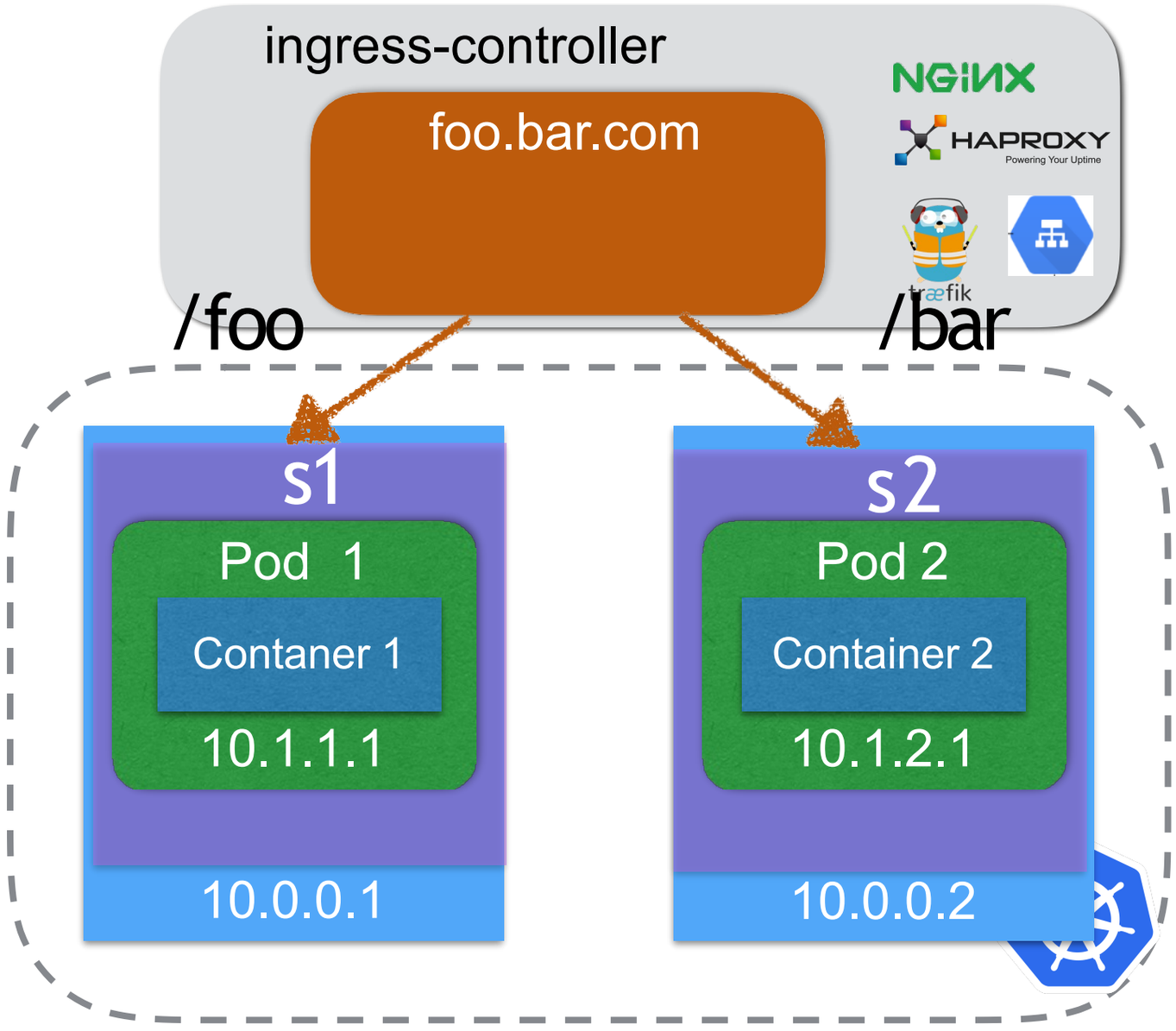
# Single Service Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-ingress
spec:
  defaultBackend:
     service:
         name: test
         port: 80
```



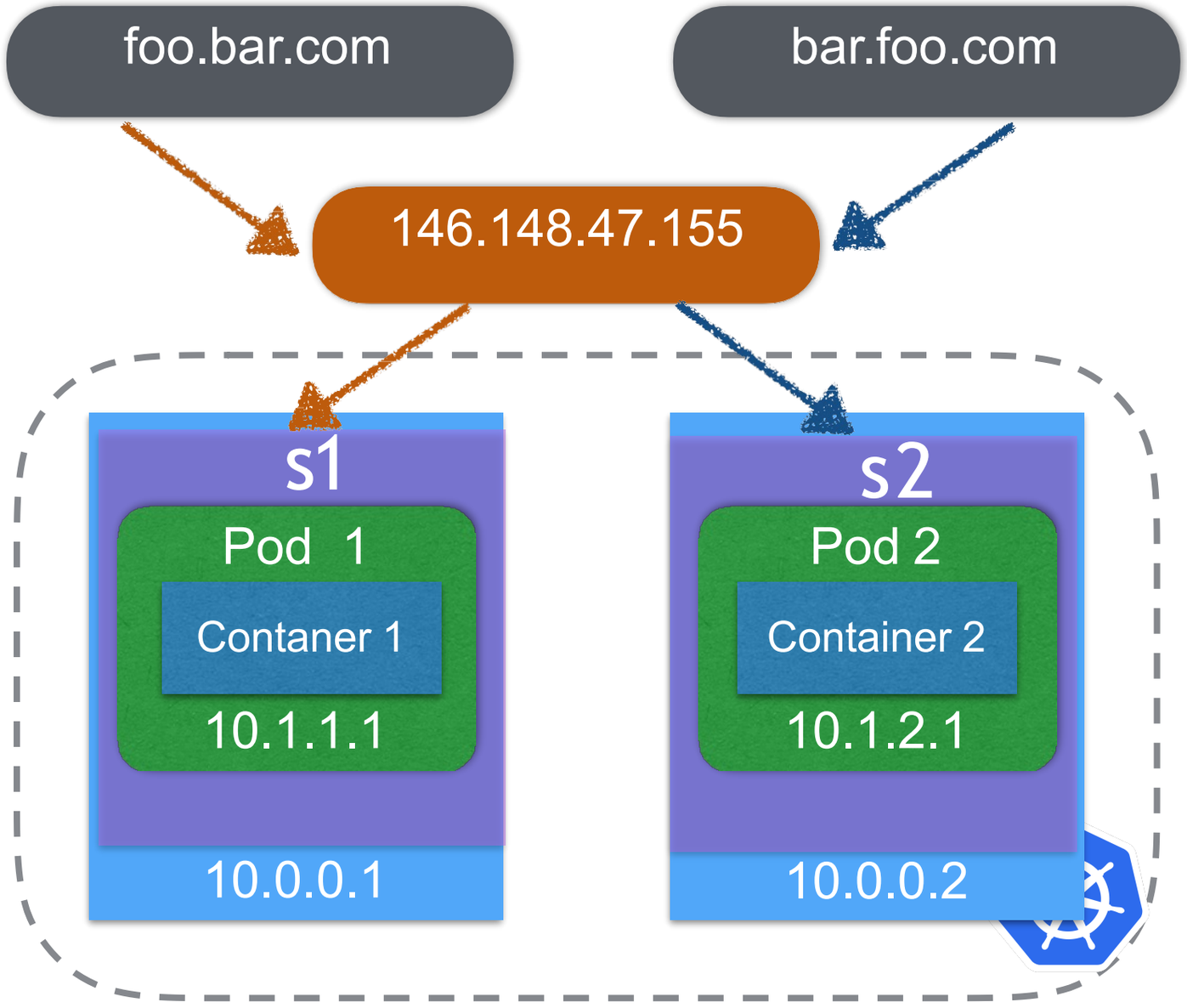
# Simple fanout

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-app
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        pathType: Prefix
        backend:
          service:
            name: s1
            port:
              number: 8080
      - path: /bar
        pathType: Prefix
        backend:
          service:
            name: s2
            port:
              number: 8080
```



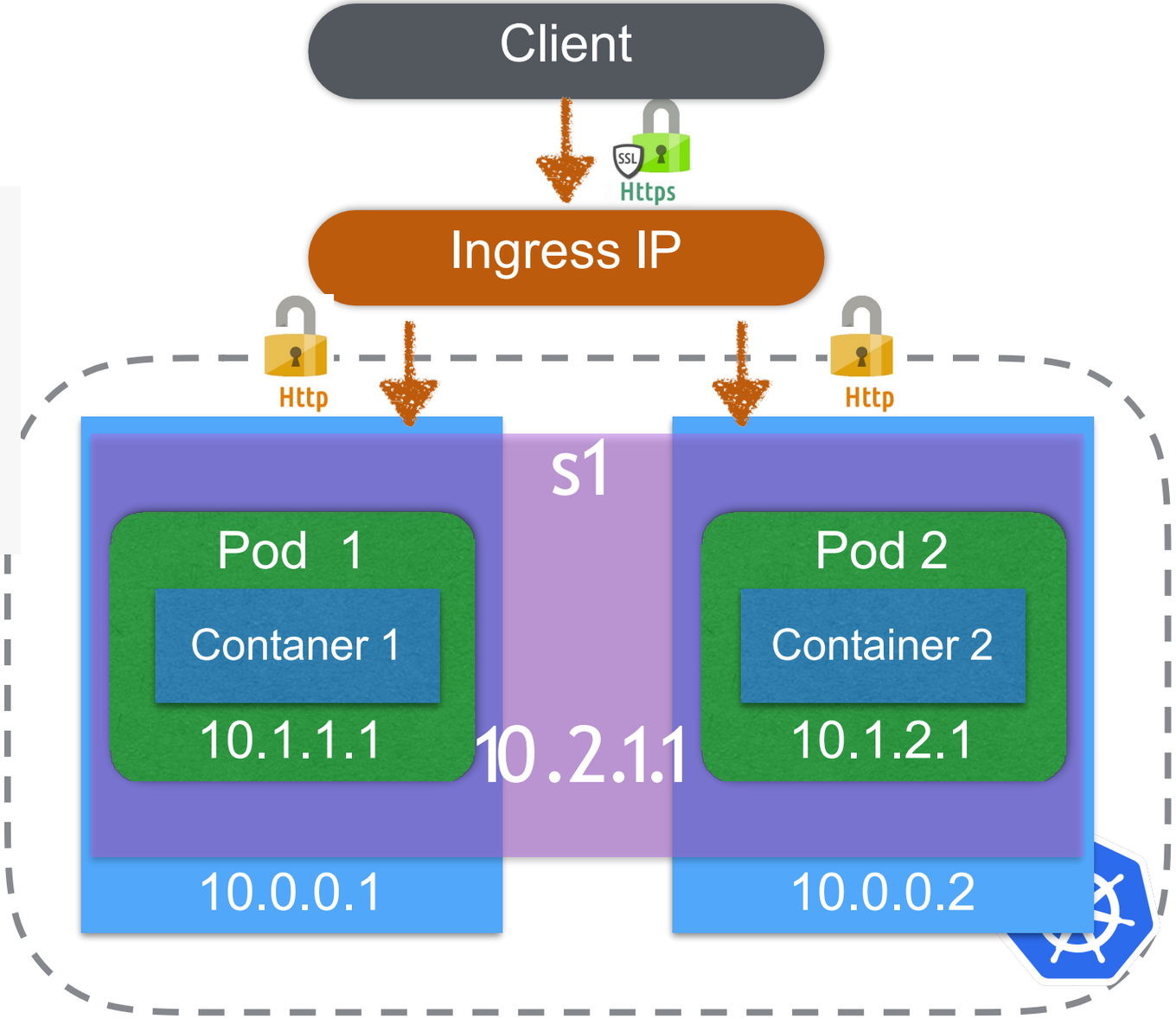
# Name based virtual hosting

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-app
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: s1
            port:
              number: 8080
- host: bar.foo.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: s2
            port:
              number: 8080
```



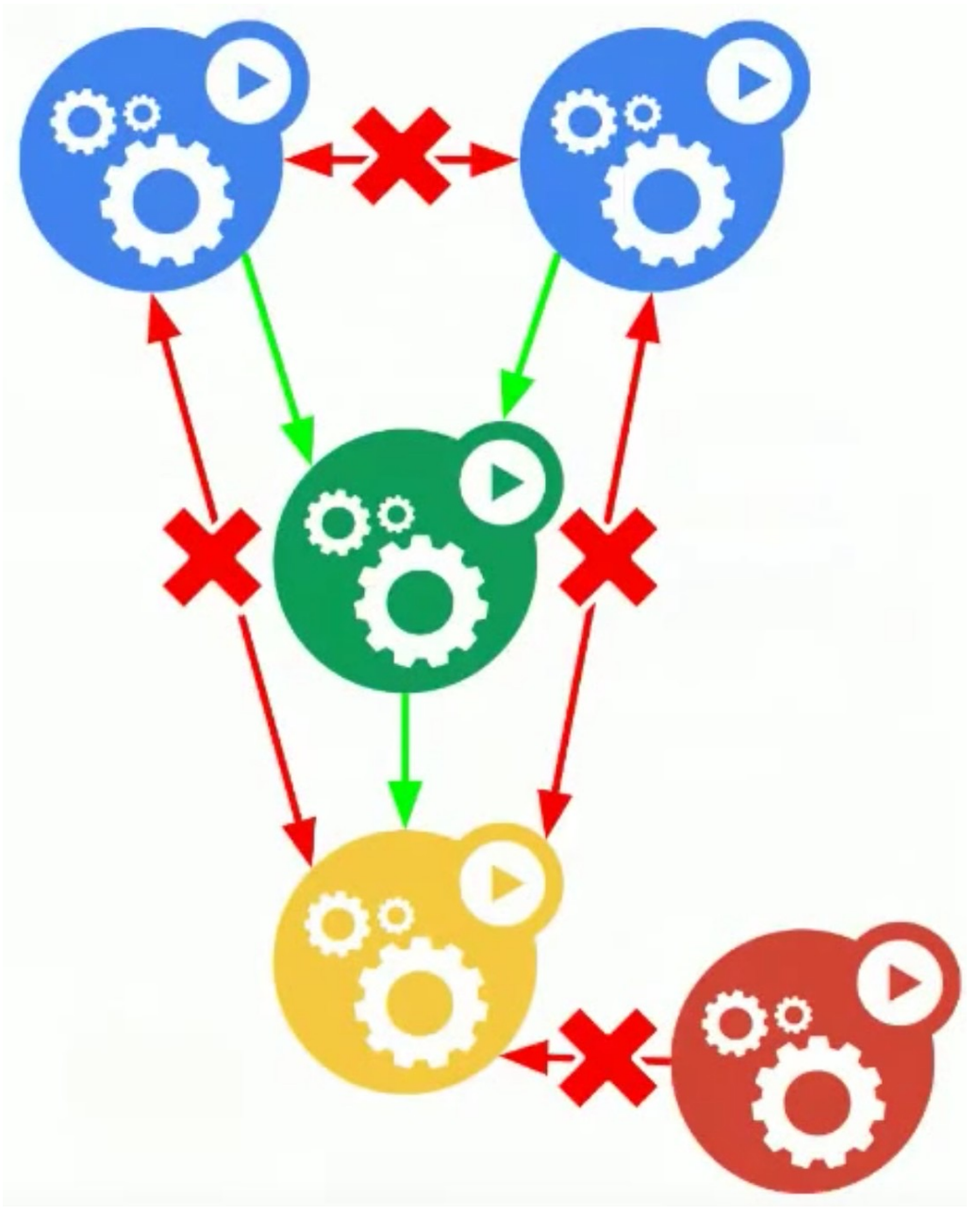
# TLS termination

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: no-rules-map
spec:
  tls:
  - secretName: testsecret
```

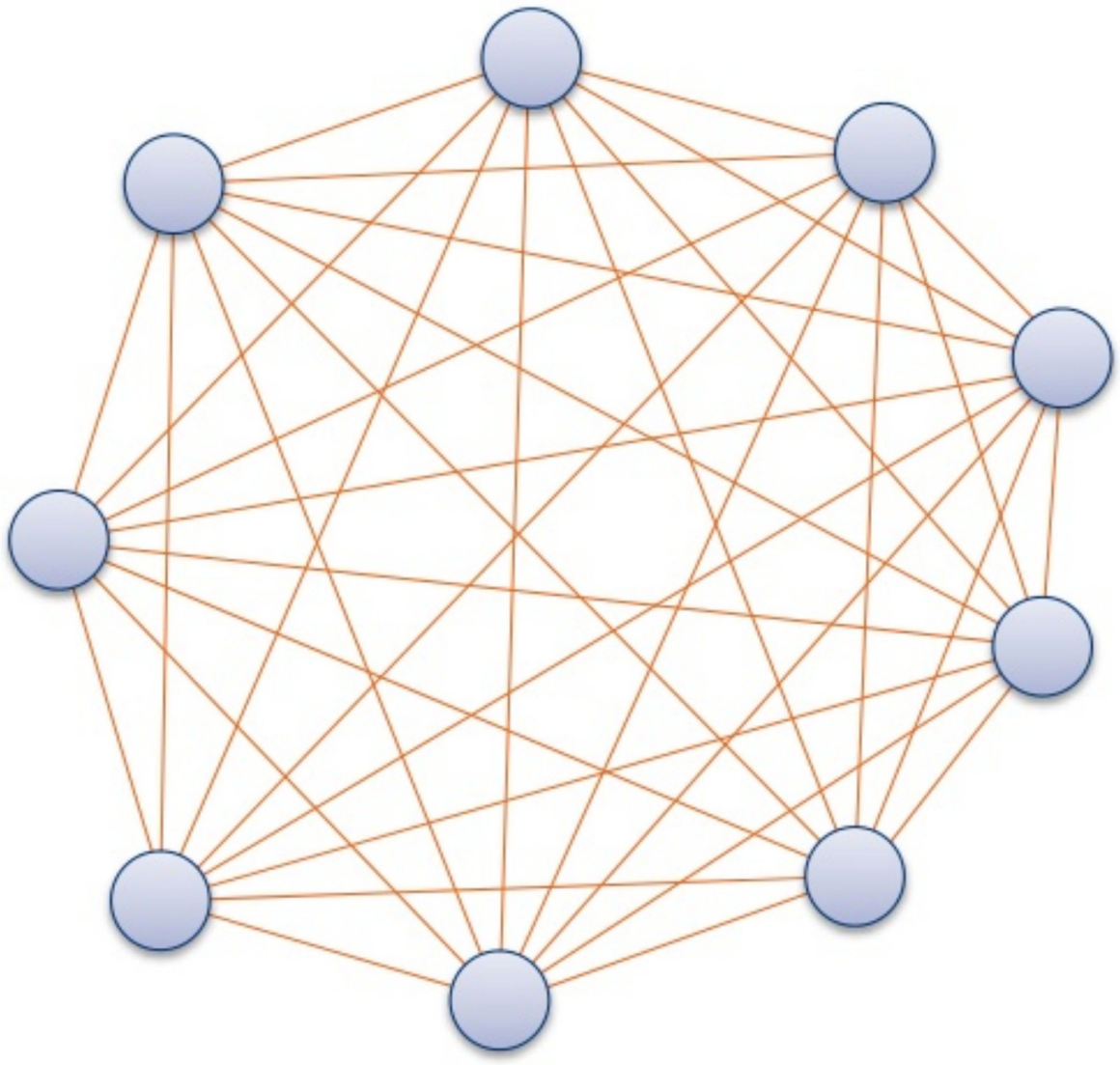


# Network Policies

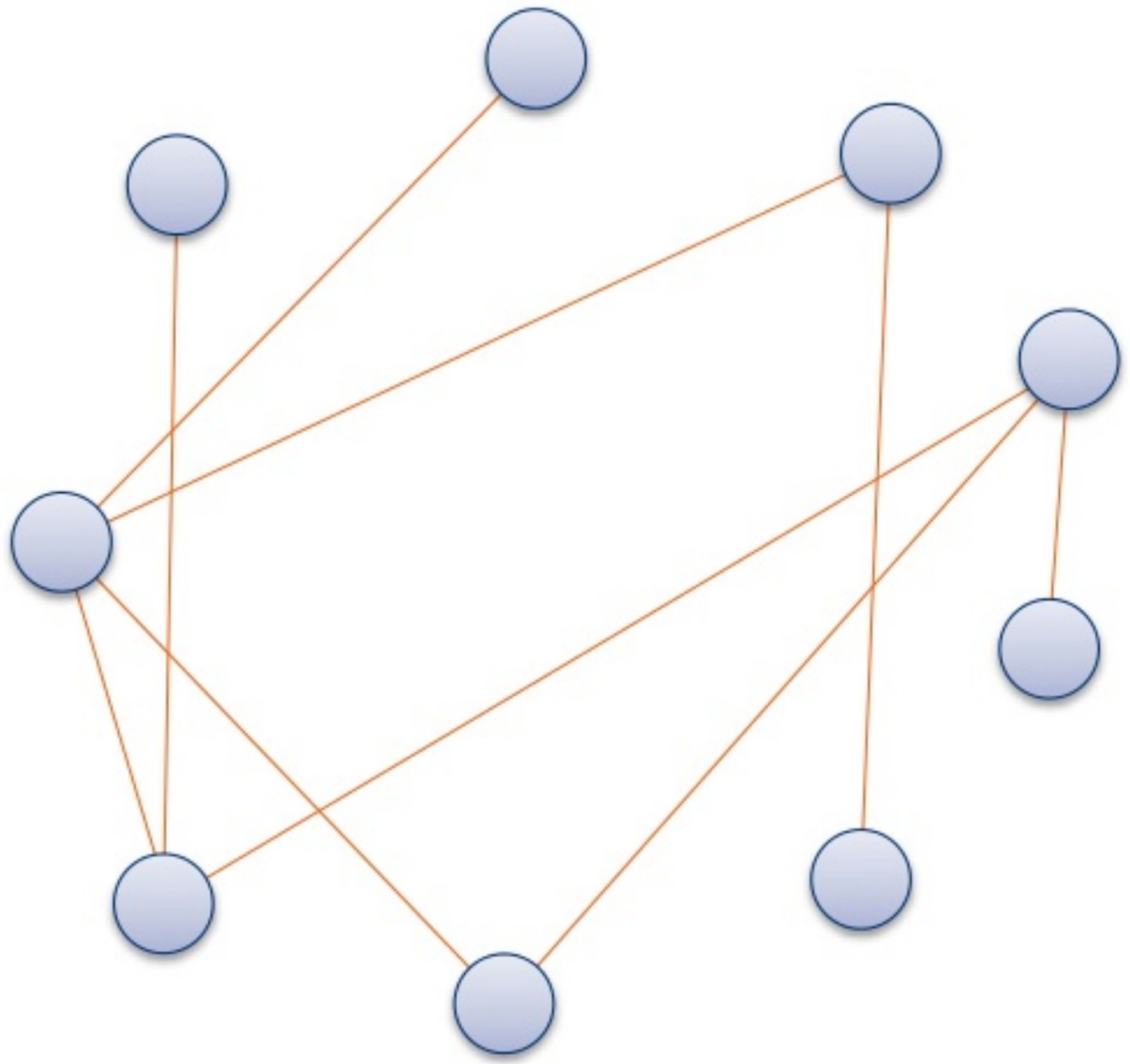
```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
   name: access-nginx
spec:
  podSelector:
    matchLabels:
      run: nginx
  ingress:
  - from:
    - podSelector:
        matchLabels:
          access: "true"
```



# Network isolation



# Network isolation

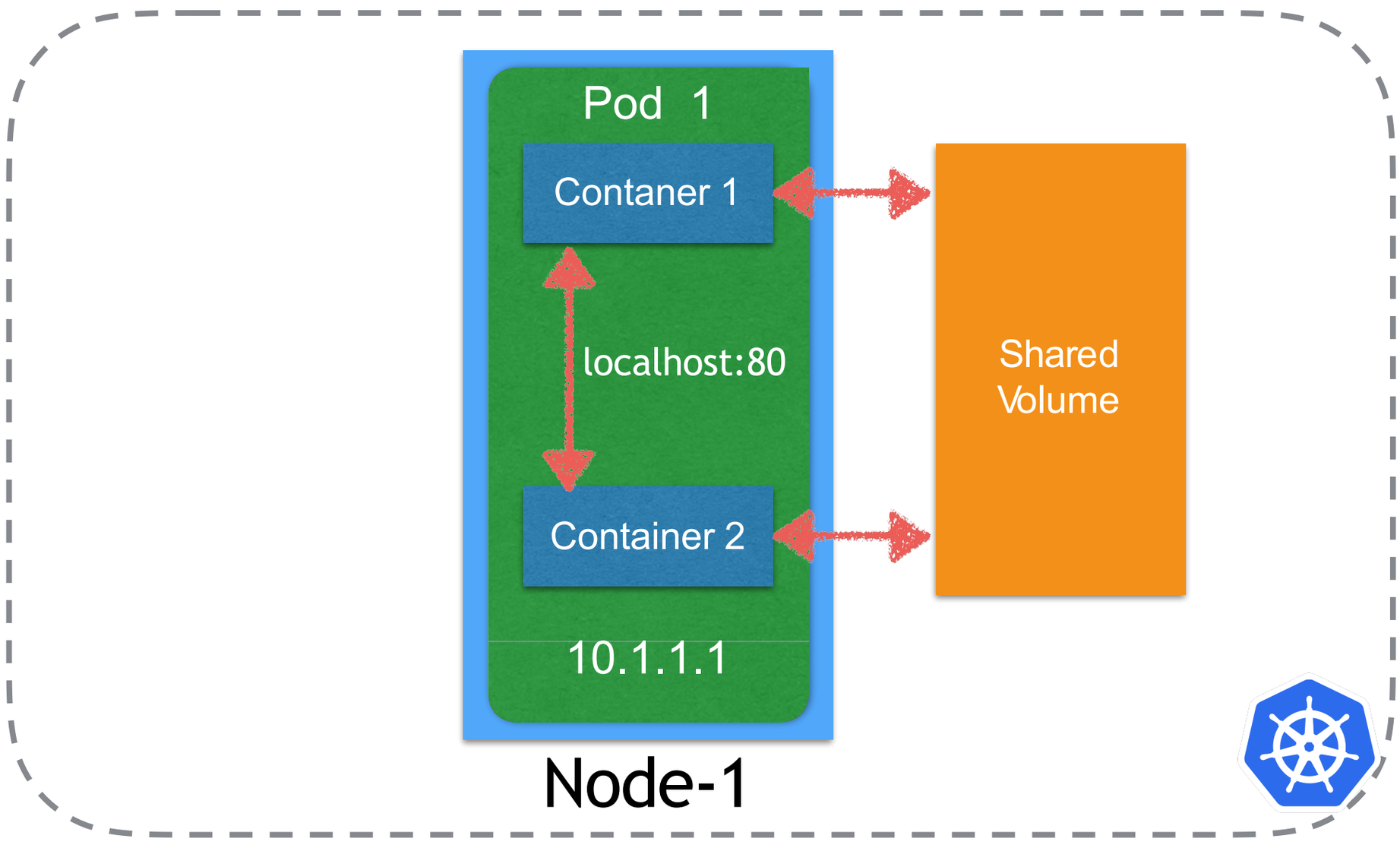


# Kubernetes networking

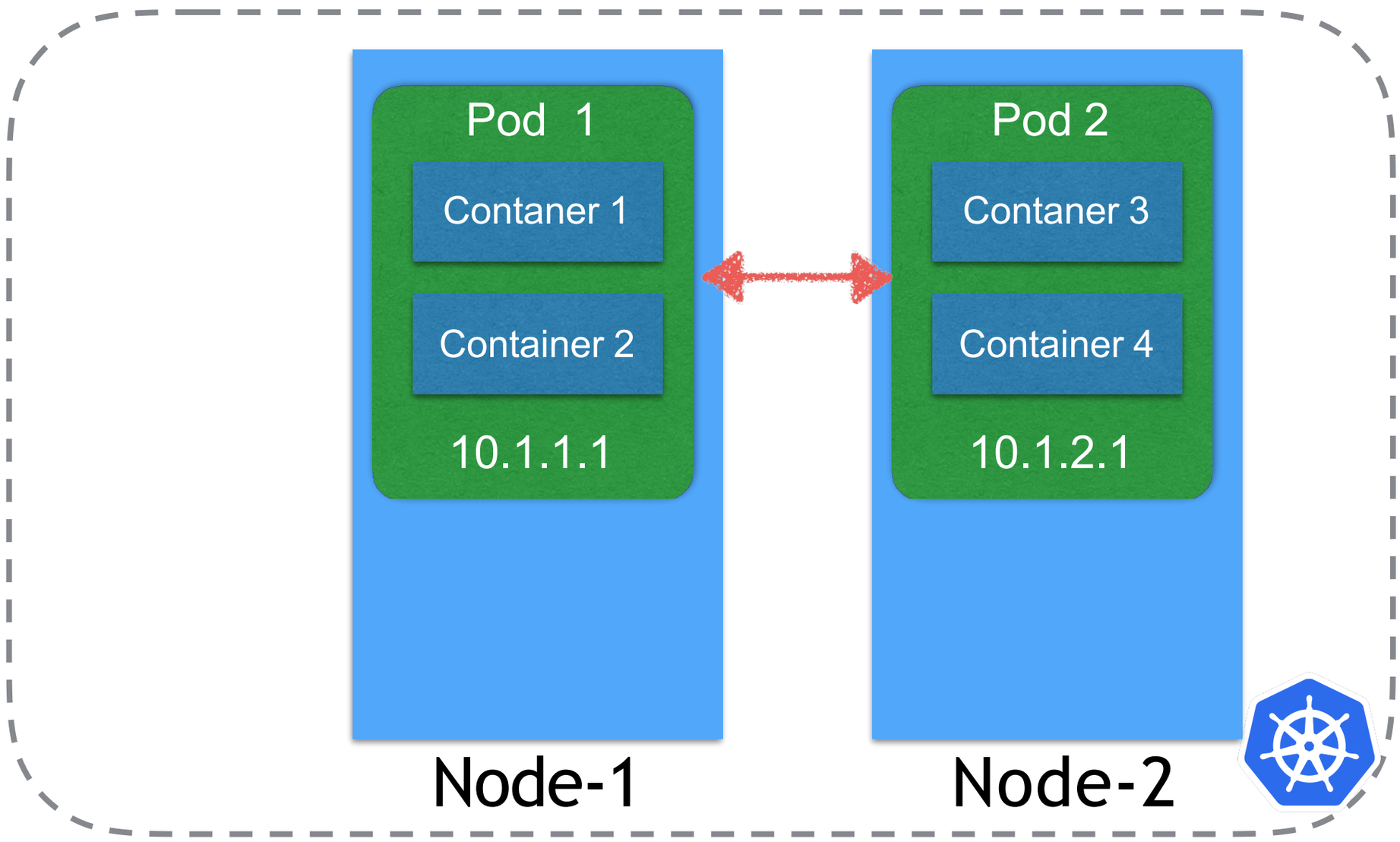
- Методы взаимодействия
  - Контейнер-Контейнер
  - Pod-Pod
  - Pod-Service
  - Внешний трафик

подробнее:
https://kubernetes.io/docs/concepts/services-networking/network-policies/

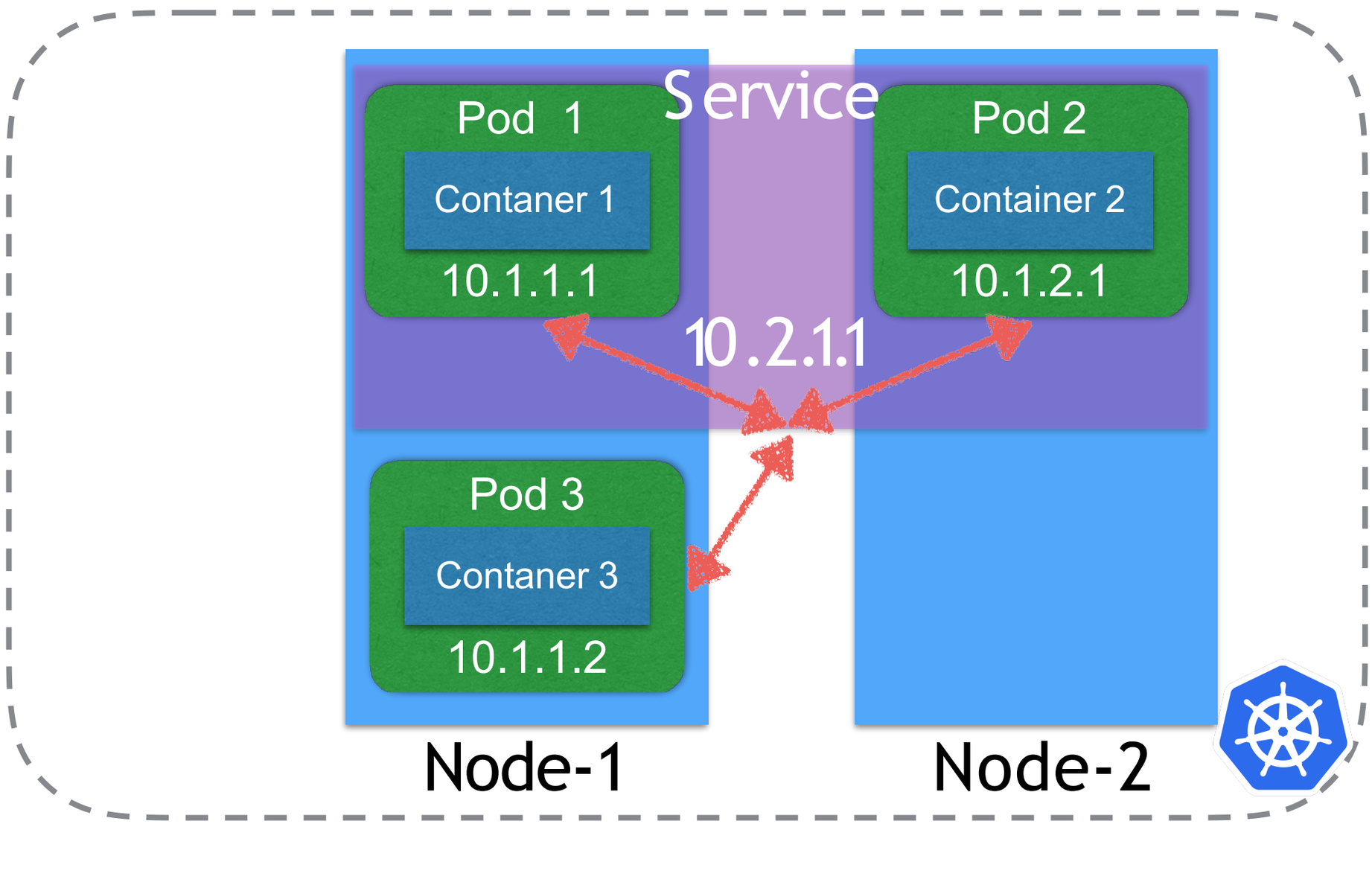
# Контейнер - Контейнер



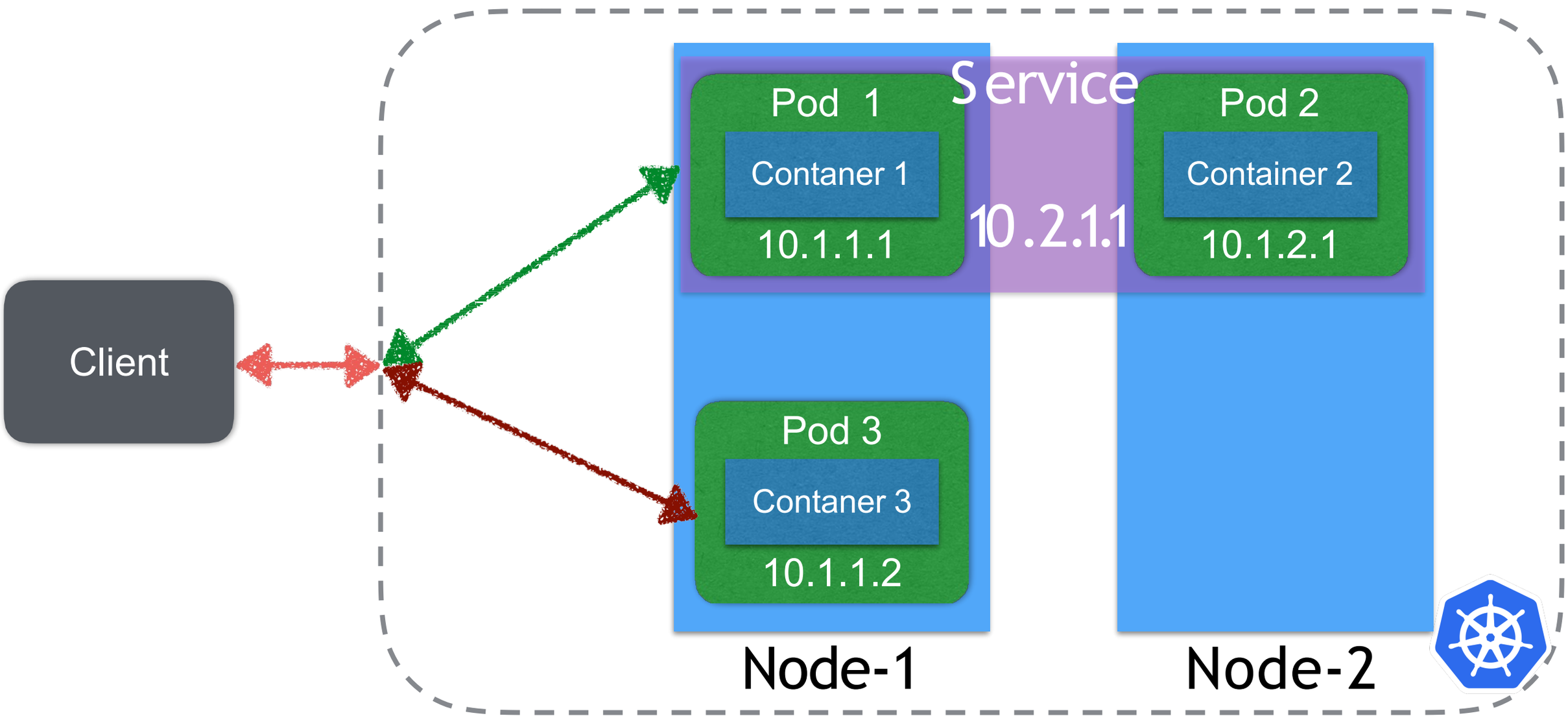
# Pod - Pod


Pod 1
Container 1
Container 2
10.1.1.1
Pod 2
Container 3
Container 4
10.1.2.1
Node-1
Node-2

# Pod - Service

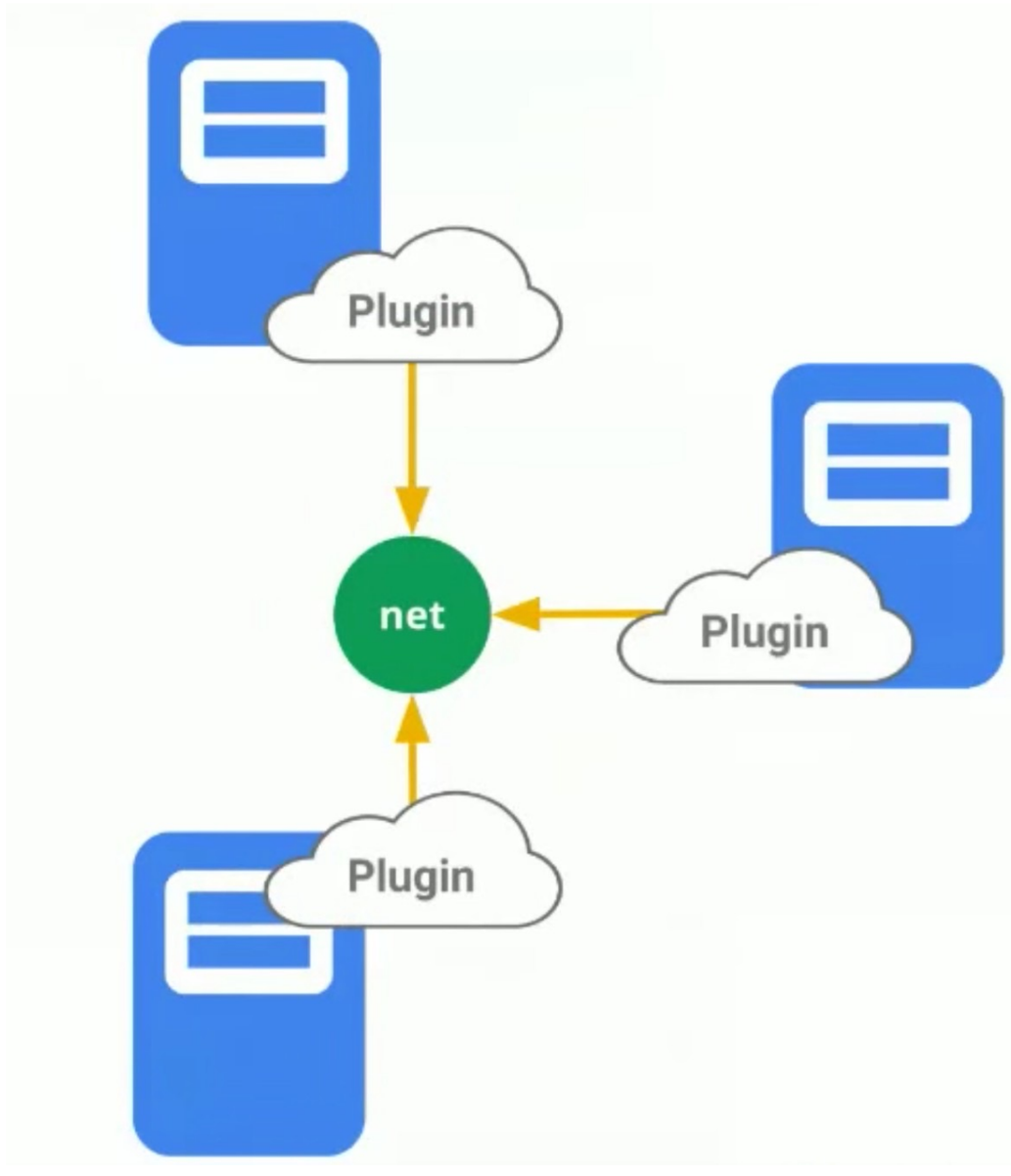

Service
Pod 1
Contaner 1
10.1.1.1
Pod 2
Container 2
10.1.2.1
10 .2.1.1
Pod 3
Contaner 3
10.1.1.2
Node-1
Node-2

# Внешний трафик



# Network plugins

- kubenet
- AWS: Route tables
- Weave
- **Calico**
- Flannel
- OVS
- OpenContrail
- Cisco Contiv



# Хранение данных в Kubernetes

# Volumes

- Меньше ограничений по сравнению с Docker'ом
- Full lifecycle с привязкой к Pod'у
- Pod может использовать несколько типов Volume'ов одновременно

# Local Volumes

- emptyDir - пустая директория, создается на Node'е и удаляется вместе с остановкой Pod'а
- hostPath - директория на Node'е
- local - локальное хранилище, диск, партиция или директория. Может быть создан только как PV

# emptyDir

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image:gcr.io/google_containers/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

# Volume Types

- cephfs
- configMap
- emptyDir
- hostPath
- nfs
- persistentVolumeClaim

Подробнее: https://kubernetes.io/docs/concepts/storage/volumes/#types-of-volumes

# Persistent Volumes

- Позволяет отделить жизненный цикл контейнеров от данных
- PersistentVolume
- PersistentVolumeClaim

# PersistentVolume

- Такой же ресурс кластера как и Node
- PV как обычные Volume'ы, но имеют отдельный от сервисов жизненный цикл
- Содержит всю информацию об имплементации хранилища

# PersistentVolume

```yaml
apiVersion: v1
 kind: PersistentVolume
 metadata:
    name: pv0003
 spec:
    capacity:
      storage: 5Gi
    accessModes:
    - ReadWriteOnce
    persistentVolumeReclaimPolicy: Recycle
    storageClassName: slow
    mountOptions:
      - hard
    - nfsvers=4.1
    nfs:
      path: /tmp
      server: 172.17.0.2
```

# PersistentVolume

- Capacity
- Access Modes - ReadWriteOnce/ReadOnlyMany/ ReadWriteMany
- Class
- Reclaim Policy - Retain/Recycle/Delete
- Mount Option
- Phase - Available/Bound/Released/Failed

# PersistentVolumeClaim

- Запрос на хранилище от пользователя
- PVC могут требовать определенный объем хранилища и прав доступа

# PersistentVolumeClaim

- Access Modes
- Resources
- Selector
- Class

# PersistentVolumeClaim

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
  storageClassName: slow
```

# PVC as Volume

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: myclaim
```

# PV portability

- В конфигурации описывайте PVC вместо PV
- Предпочтительно использование Storage Class
- Следите за не выполненными PVC

# Storage Classes

- Описание "классов" различных систем хранения
- Разные классы могут использоваться для:
  - QoS уровни
  - Произвольных политик
  - Для динамического provisioning

# Storage Classes

- Provisioner
- Reclaim Policy
- Mount Options

# Storage Classes

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
reclaimPolicy: Retain
mountOptions:
  - debug
```

# Dynamic Volume Provisioning

- Использует StorageClass для provisioning'а
- PersistentVolumeClaim содержит информацию о нужном StorageClass

# Lifecycle of a volume and claim

- Provisioning:
  - Static
  - Dynamic
- Binding
- Using
- Reclaiming