Kubernetes. Сетевое взаимодействие. Хранение данных.

Не забудь включить запись!

План

- Сетевое взаимодействие в Kubernetes
- Хранение данных в Kubernetes

Сетевое взаимодействие в Kubernetes

Pods

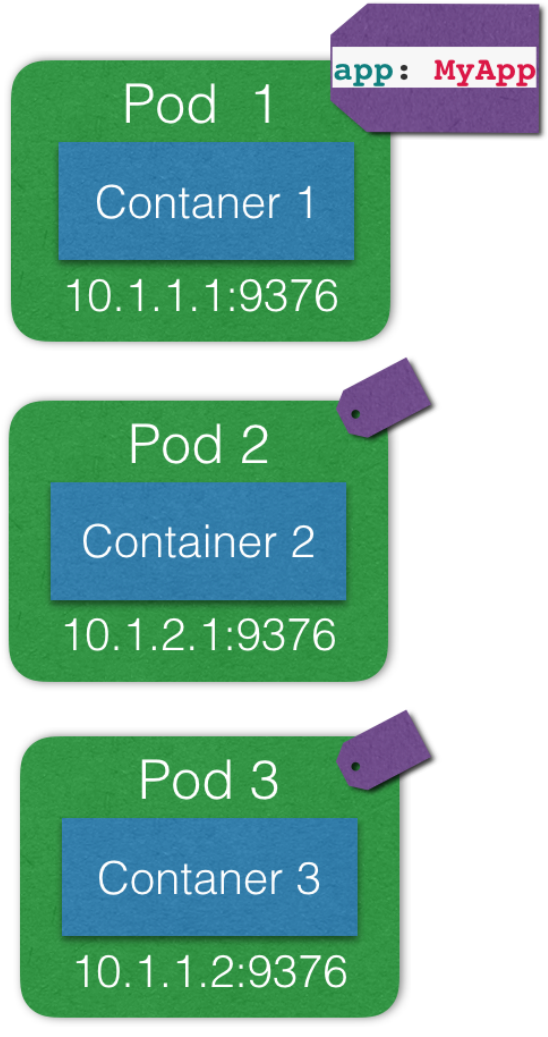
- Смертны
- Динамически создаются и удаляются
- IP-адреса появляются и исчезают вместе с Pod'ами
- Несколько Pod'ов могут выполнять одну и ту же функцию

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

- 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
```

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

Headless Service

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

Headless Service

Service:

```
$ nslookup ui-service

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

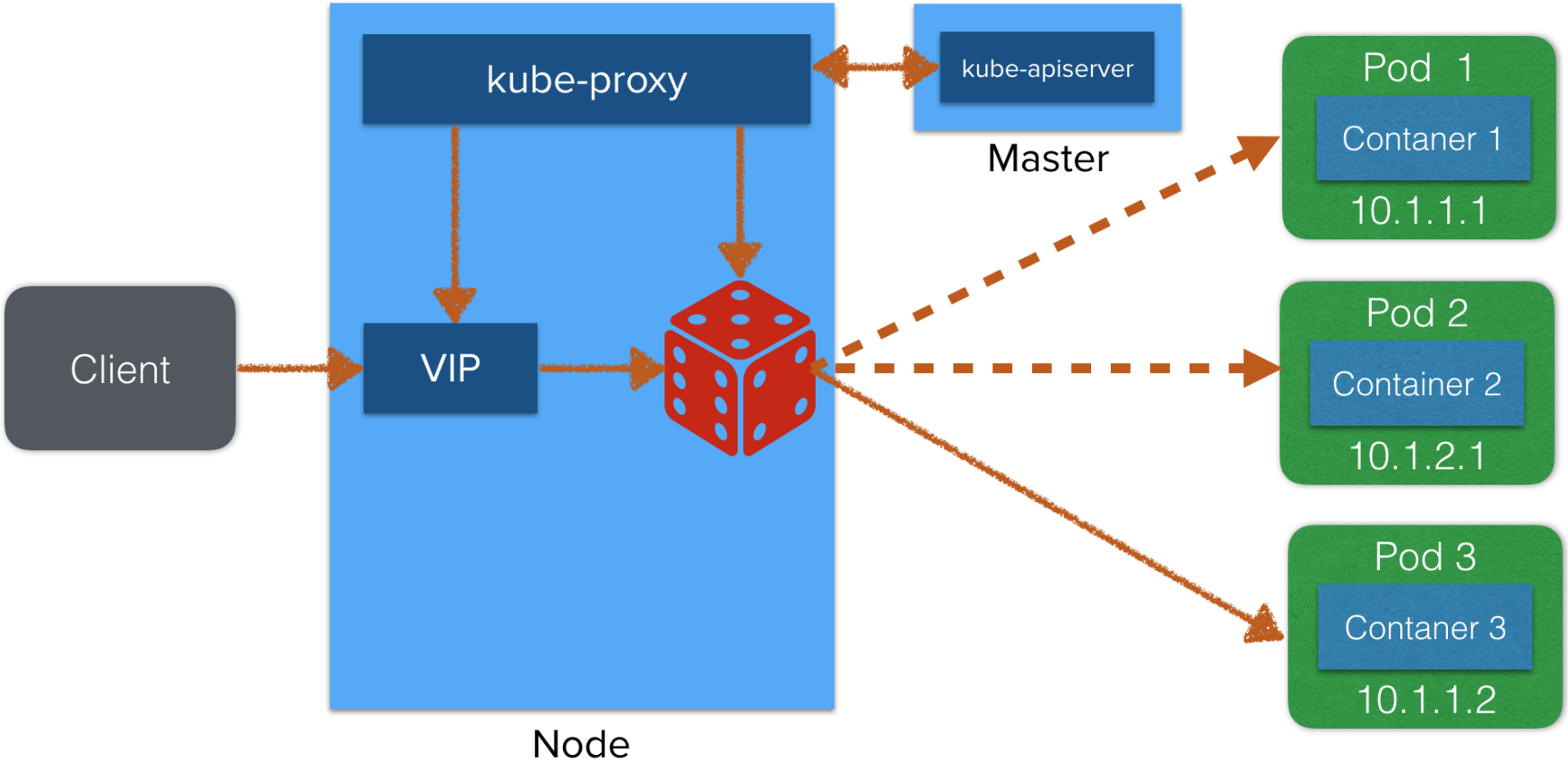
Headless Service:

```
$ nslookup ui-service-headless

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

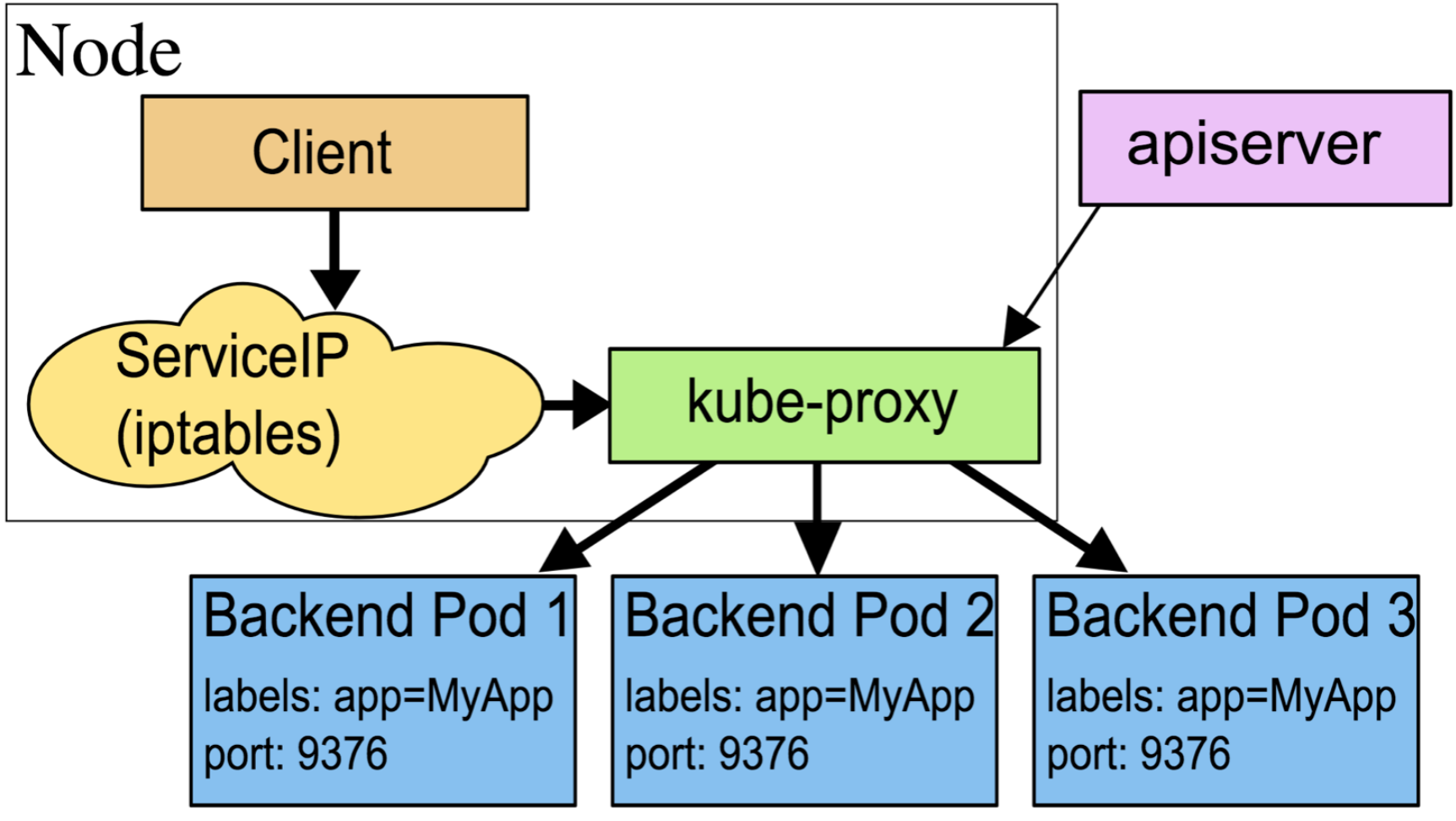
kube-proxy

- Проксирует TCP и UDP (не HTTP)
- Используется только для работы с сервисами
- Обеспечивает внутреннюю балансировку
- Используется для внутренней коммуникации

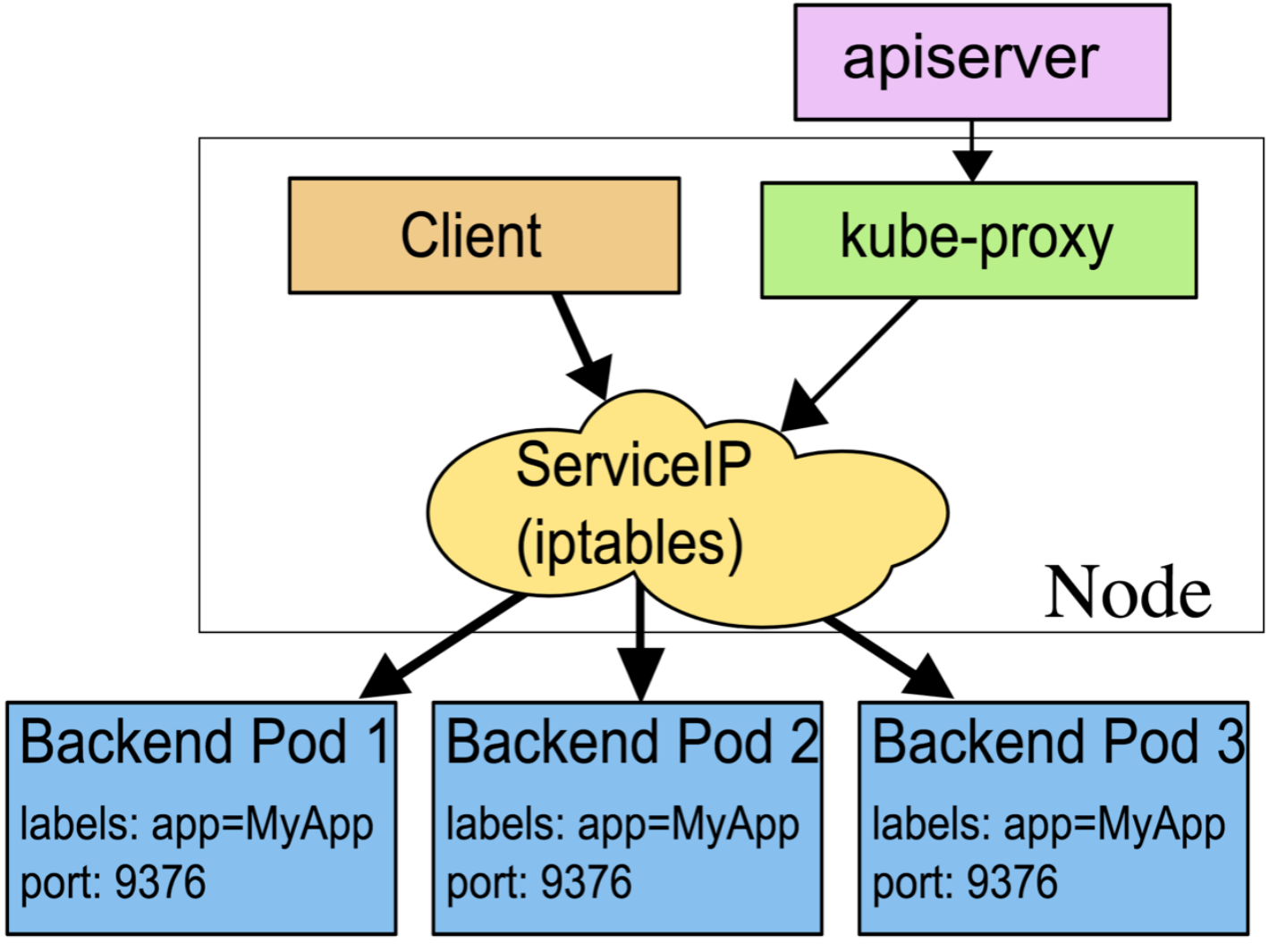
kube-proxy


kube-proxy
kube-apiserver
Master
Pod 1
Contaner 1
10.1.1.1
Client
VIP
Pod 2
Container 2
10.1.2.1
Pod 3
Contaner 3
10.1.1.2
Node

kube-proxy. Userspace (deprecated)



kube-proxy. Iptables (default)

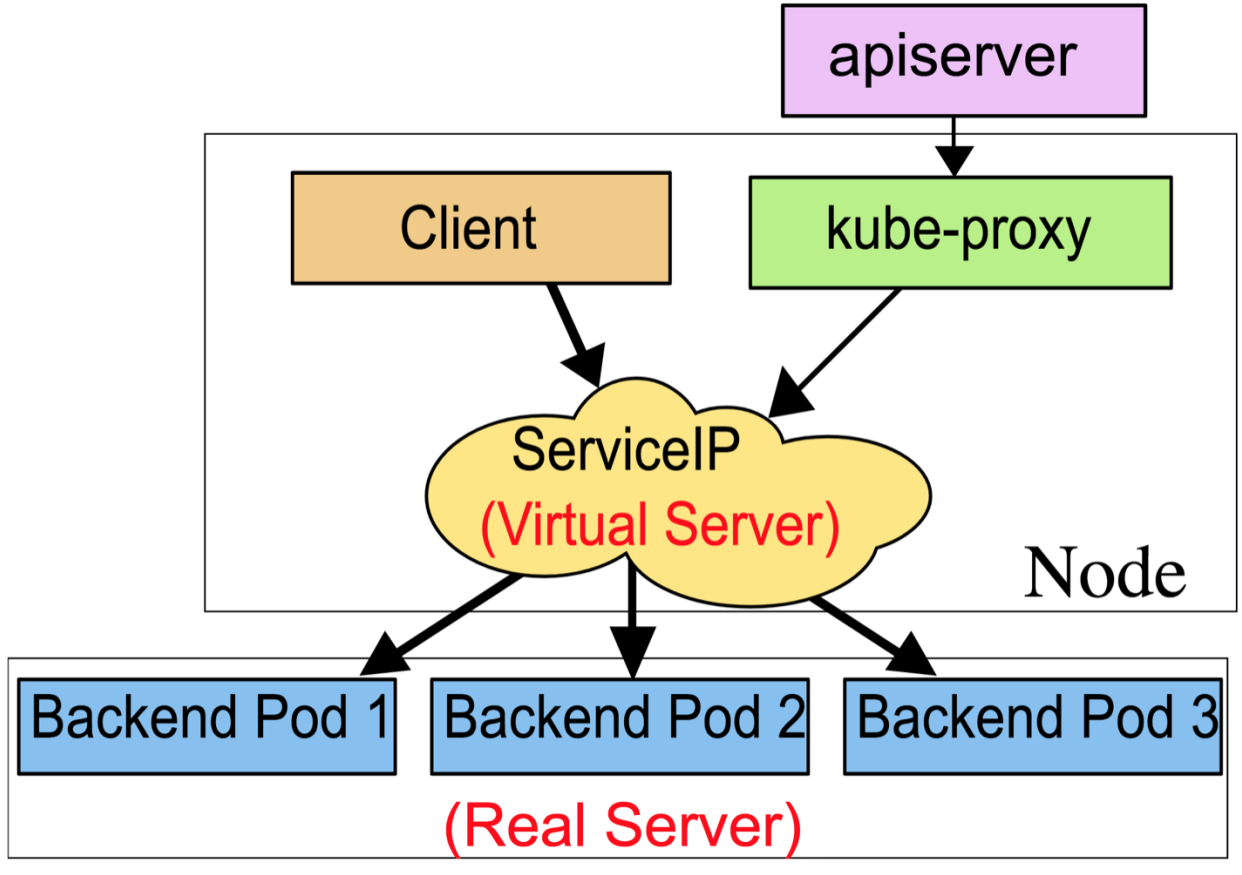


kube-proxy. Iptables (default)

```
*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
```

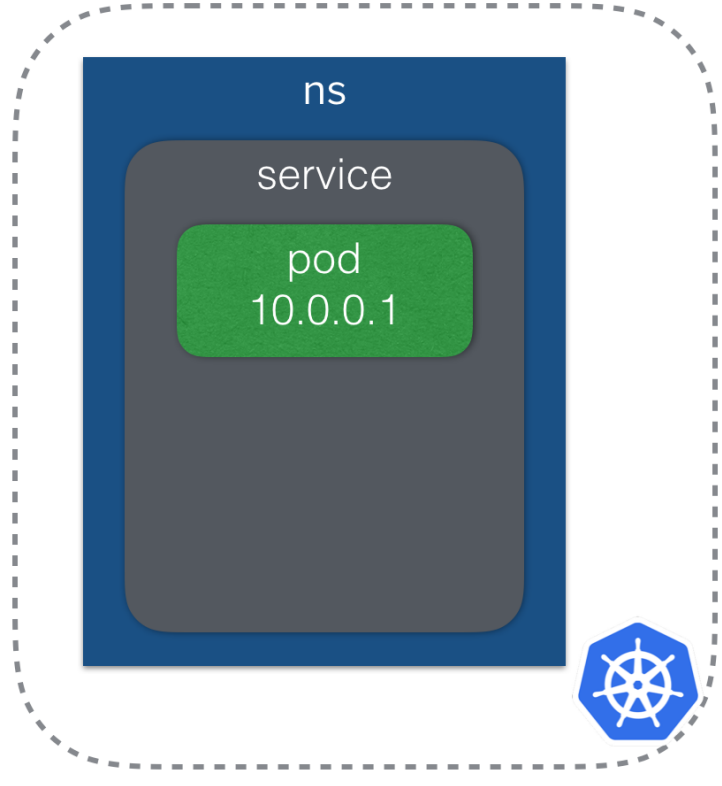
kube-proxy. IPVS (beta)

Сравнение iptables и IPVS

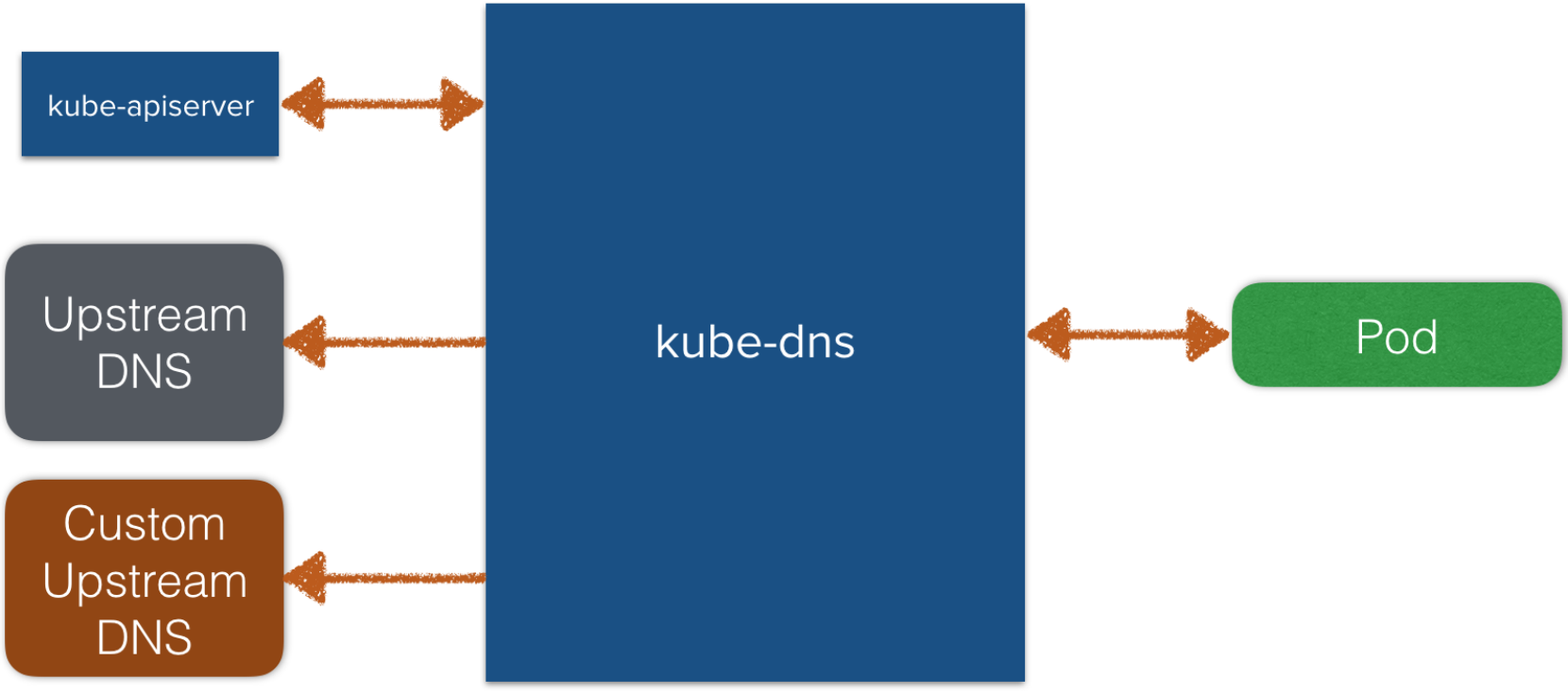


kube-dns

- 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**



kube-dns


kube-apiserver
Upstream DNS
Custom Upstream DNS
kube-dns
Pod

kube-dns

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-dns
  namespace: kube-system
data:
  upstreamNameservers: |
    ["8.8.8.8", "8.8.4.4"]
```

CoreDNS

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

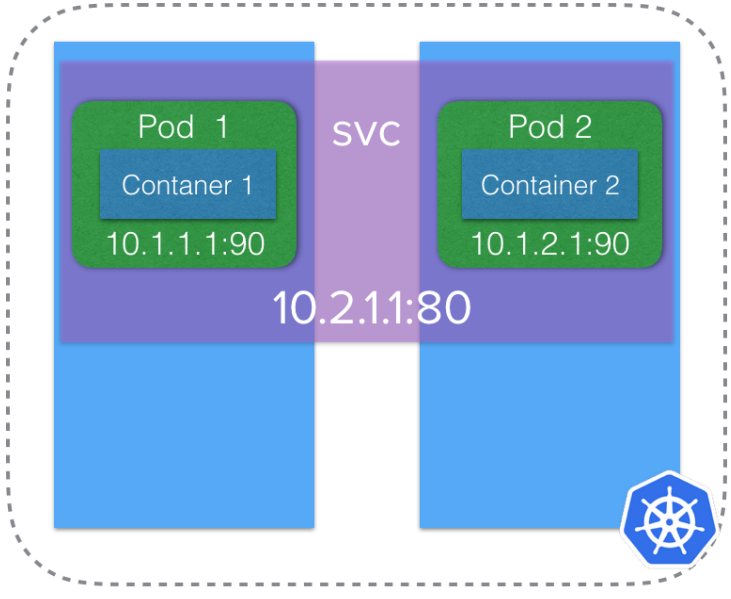
Как опубликовать приложение

Типы сервисов

- **ClusterIP** - доступ только изнутри кластера
- **NodePort** - доступ снаружи кластера через порты физических хостов
- **LoadBalancer** - доступ снаружи кластера с использованием внешнего сервиса балансировки
- **ExternalIP** - доступ снаружи по внешнему IP адресу, указывающему на одну из нод

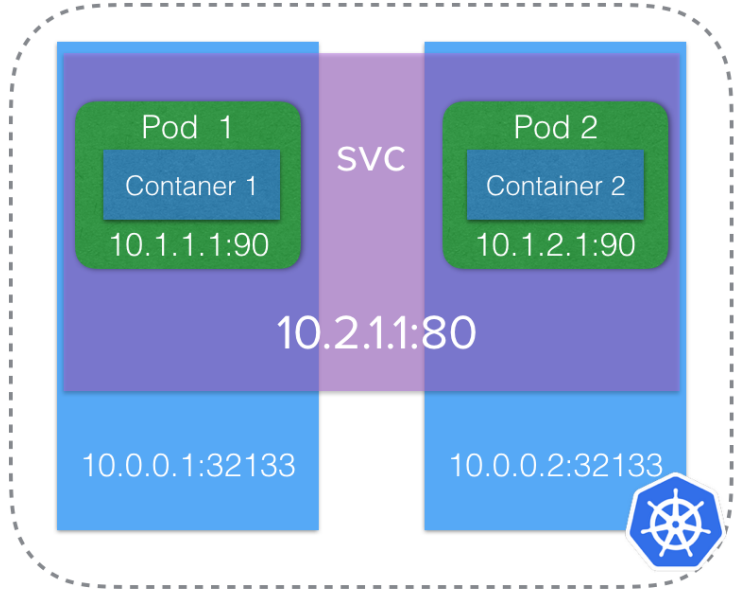
ClusterIP

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



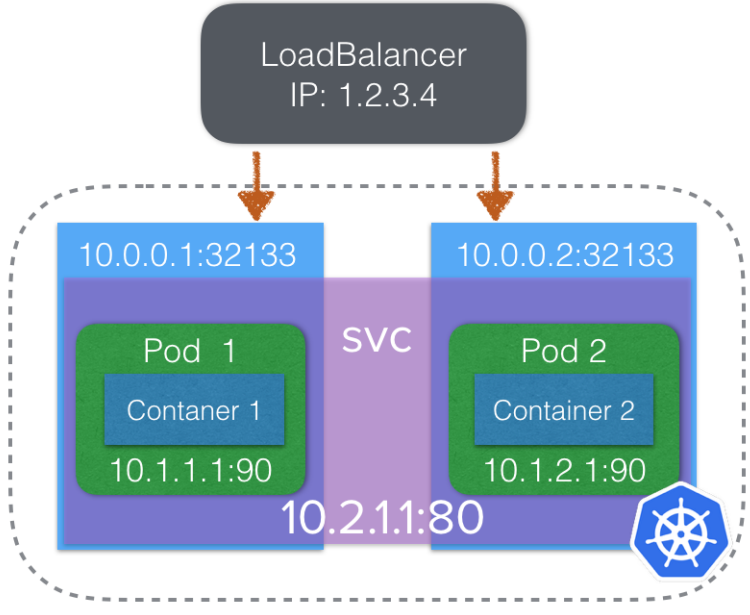
NodePort

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



LoadBalancer

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



ExternalIP

ClusterIP + ExternalIP

ExternalIP - IP адрес одного или нескольких физических хостов

```
kind: Service
apiVersion: v1
metadata:
  name: svc
spec:
  type: ClusterIP
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 90
  externalIPs:
  - 1.2.3.4
```

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

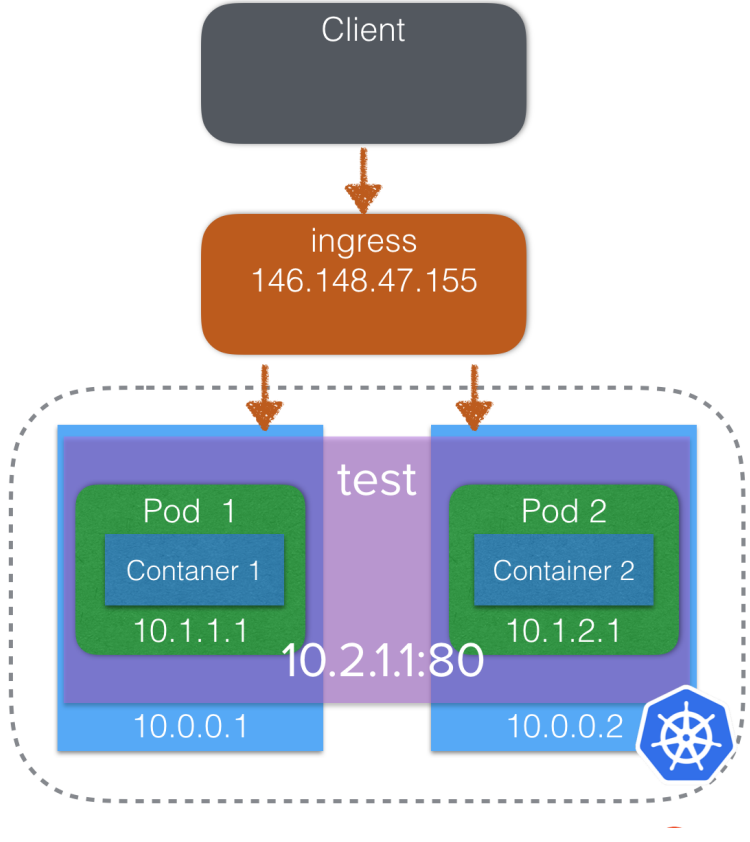
Ingress - набор правил внутри кластера Kubernetes.

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

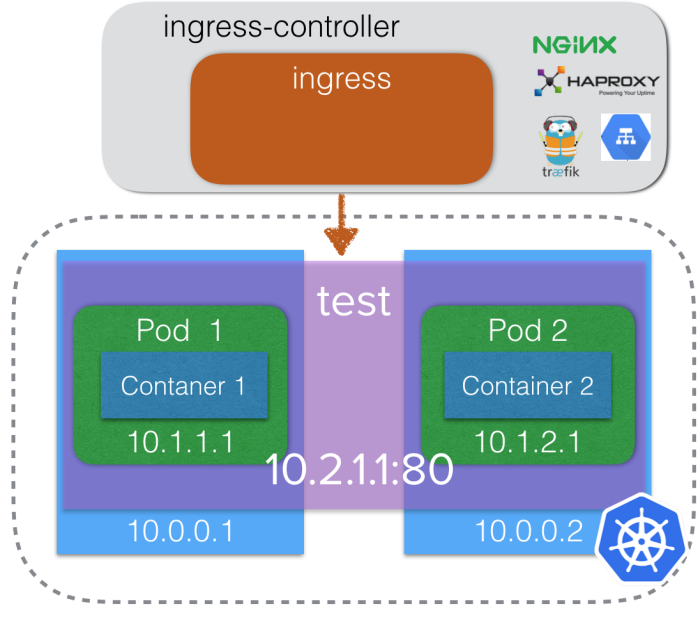
Ingress

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



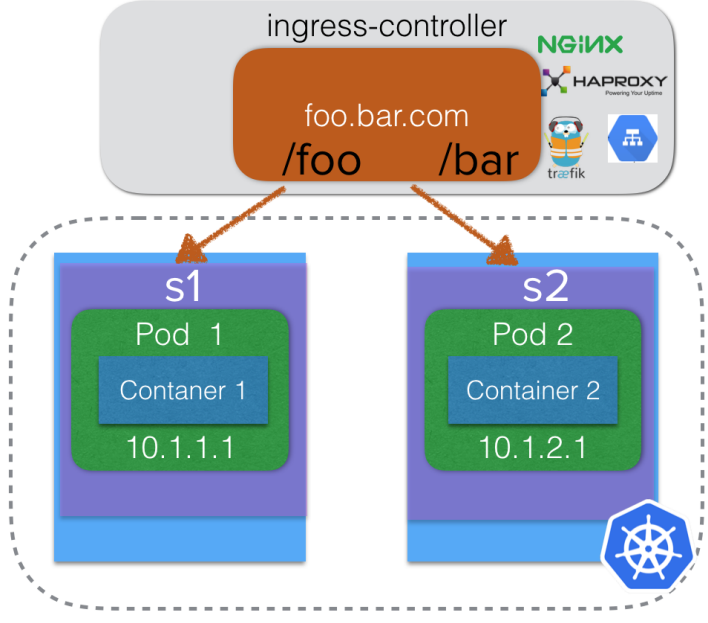
Single Service Ingress

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



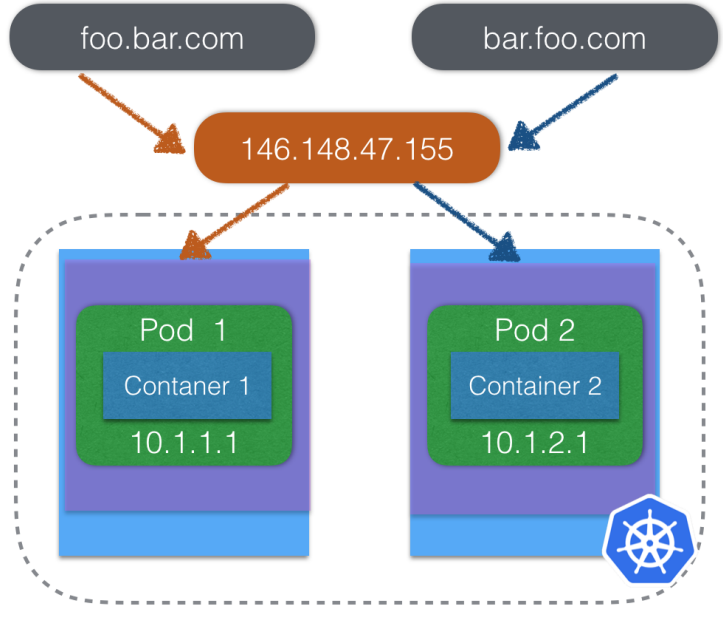
Simple Fanout

```
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
```



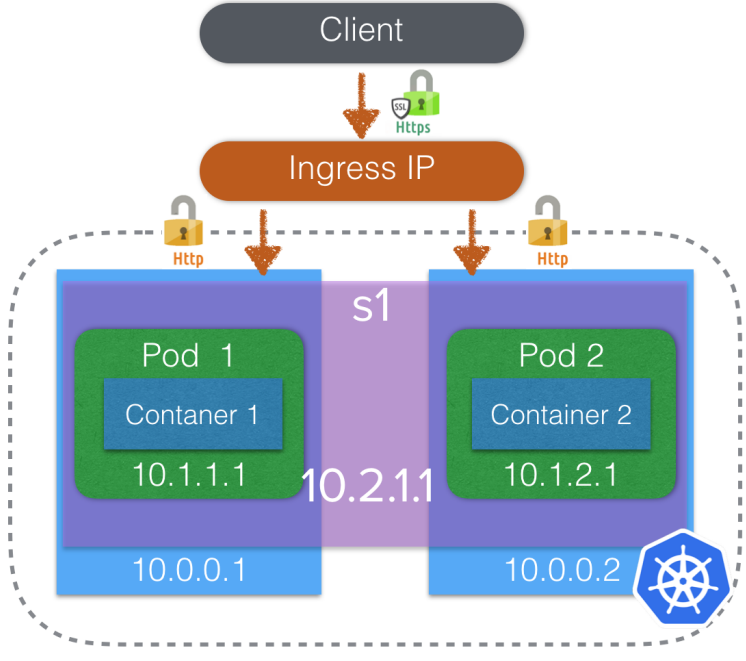
Name Based Virtual Hosting

```
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

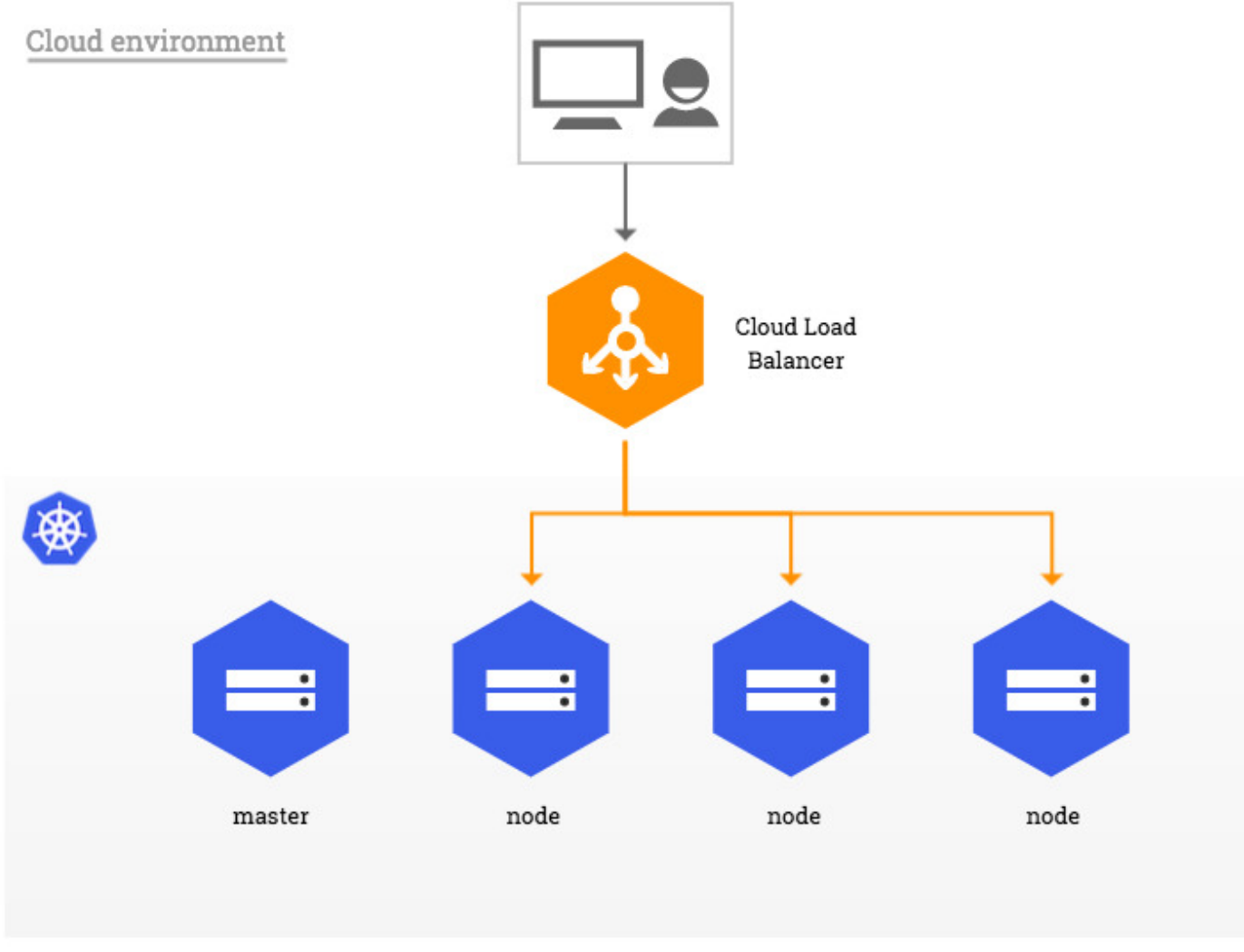
```
apiVersion: v1
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
kind: Secret
metadata:
  name: testsecret-tls
  namespace: default


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



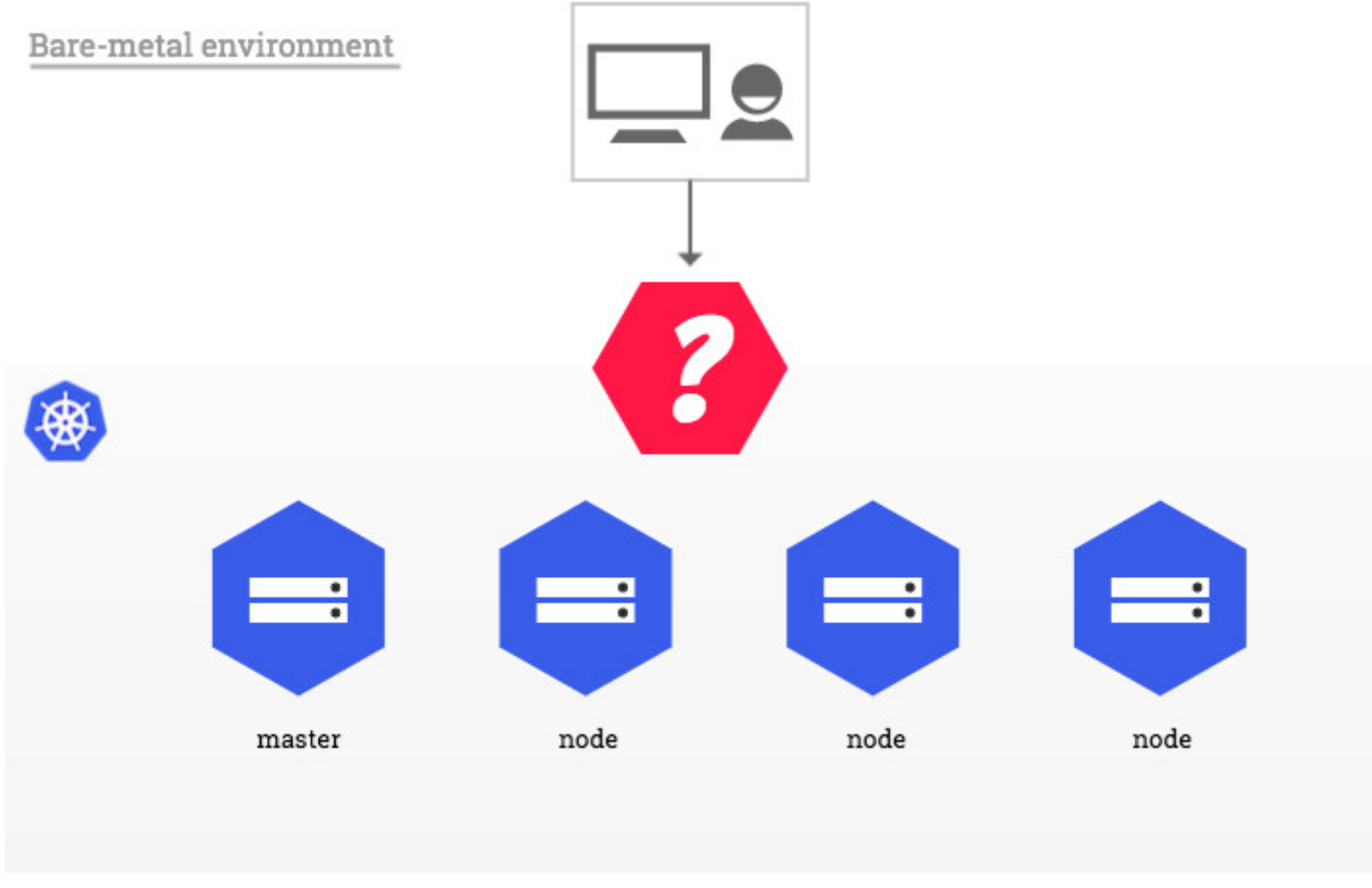
Ingress

Краткий обзор различных Ingress и сводная таблица по их возможностям

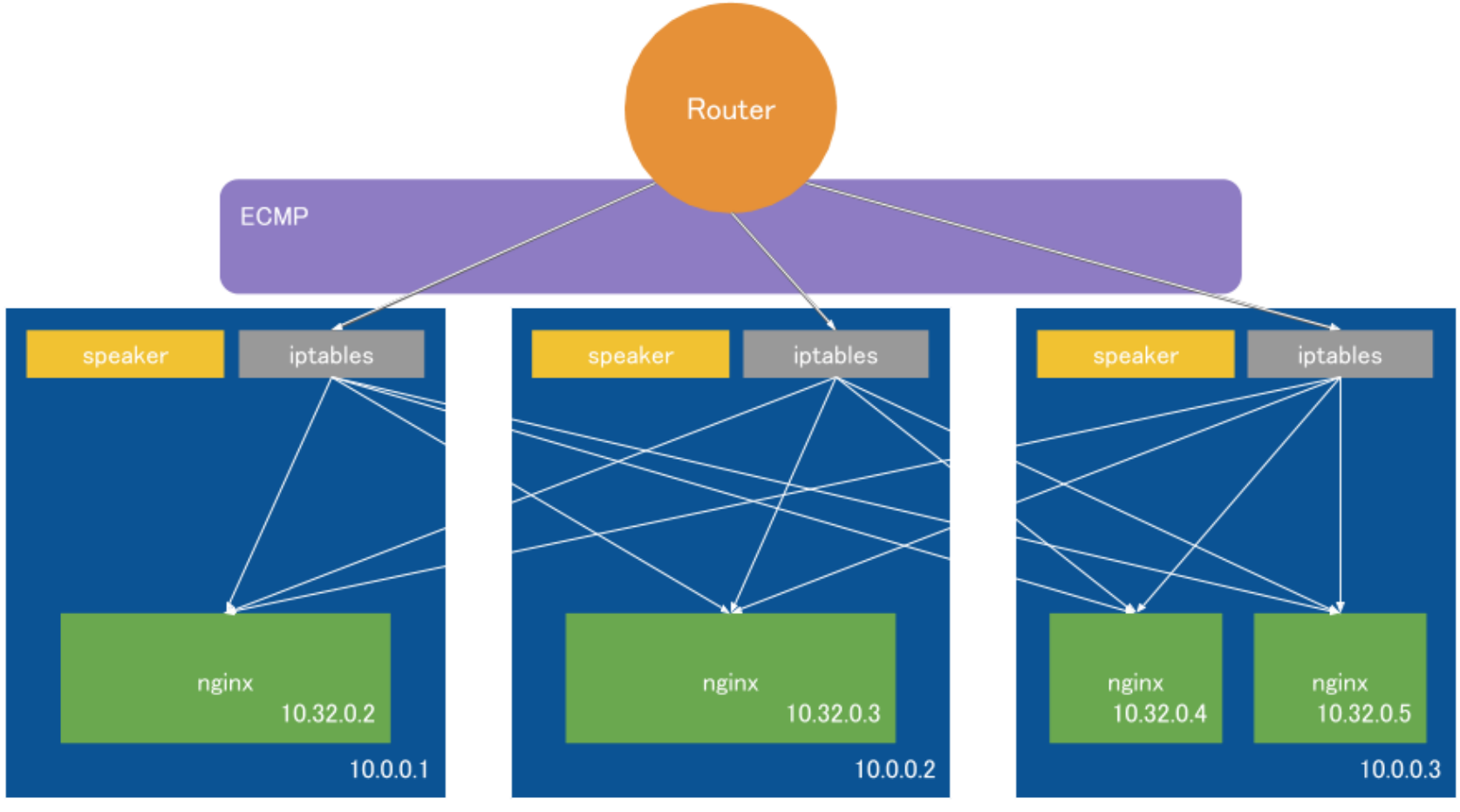
Ingress


Cloud environment
Cloud Load Balancer
master
node
node
node

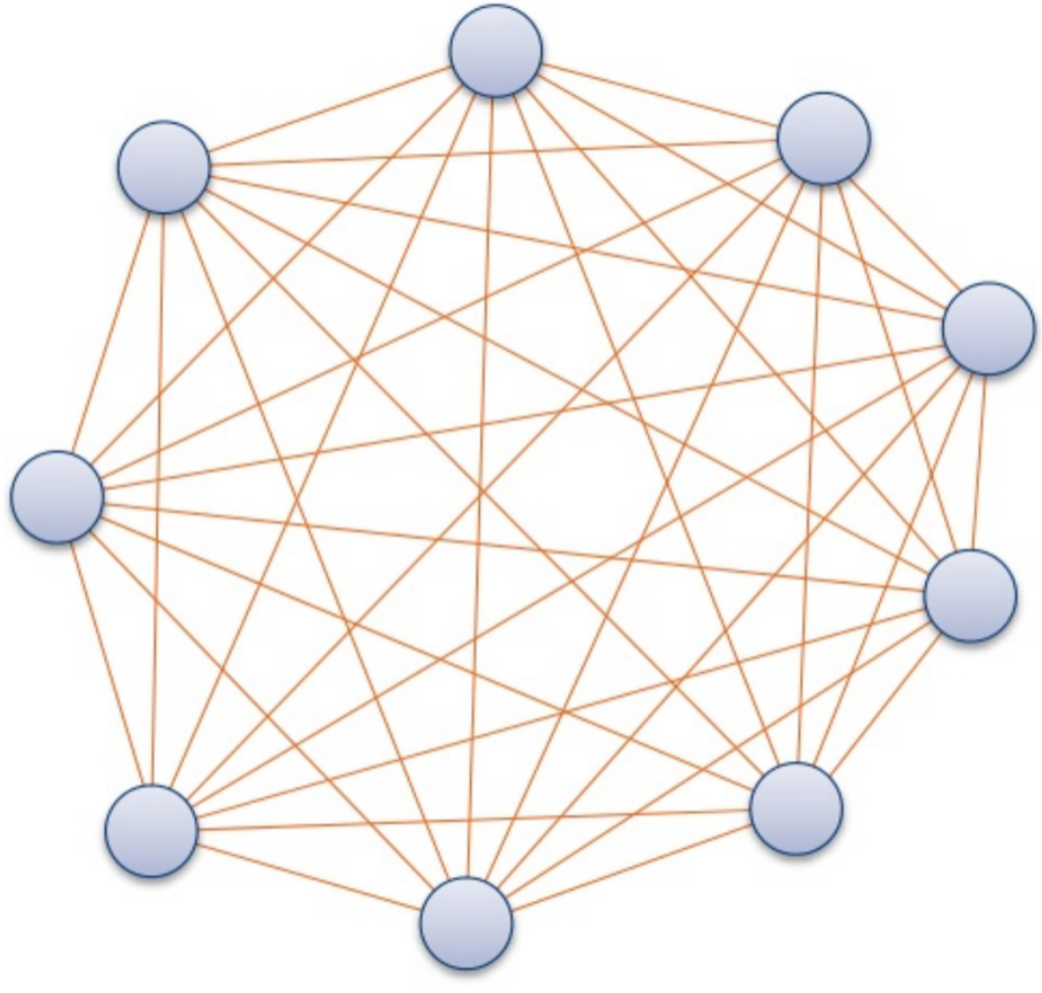
Ingress



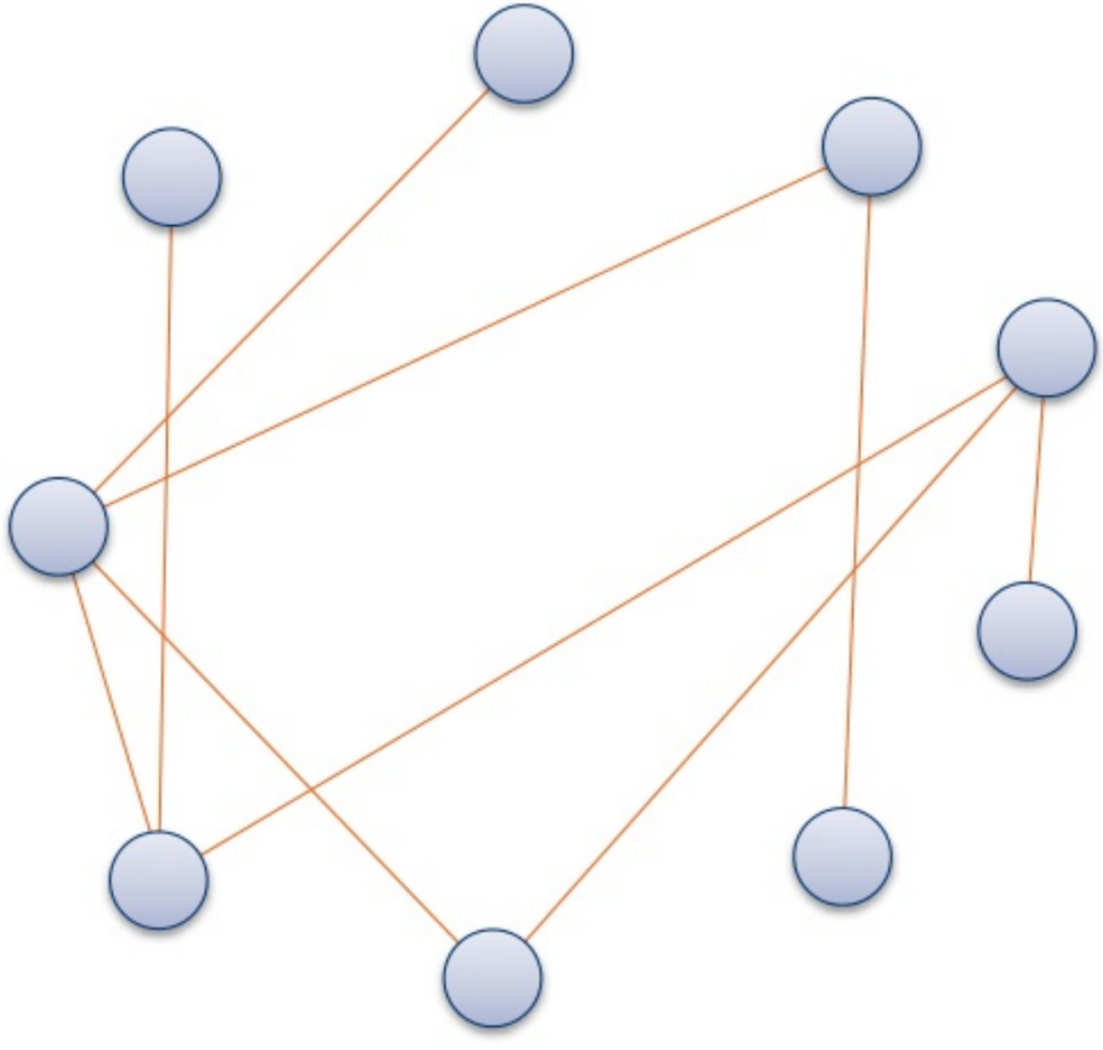
MetalLB


Router
ECMP
speaker
iptables
nginx
10.32.0.2
10.0.0.1
speaker
iptables
nginx
10.32.0.3
10.0.0.2
speaker
iptables
nginx
10.32.0.4
nginx
10.32.0.5
10.0.0.3

Network Isolation

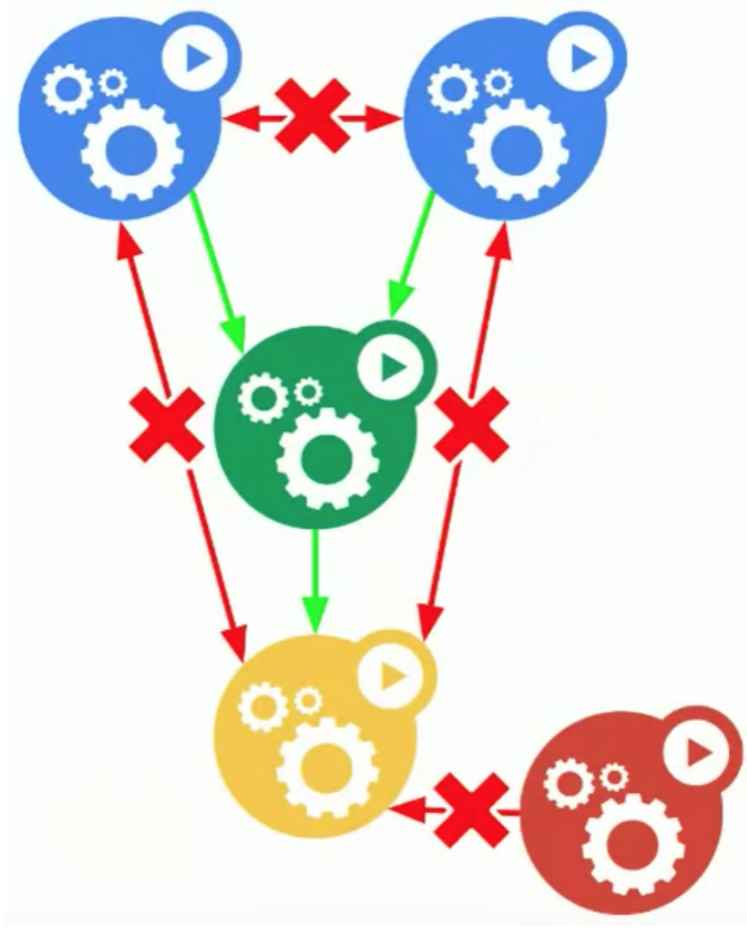


Network Isolation



Network Policies

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



Как изолировать namespace

Набор типовых network policies

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: isolated-ns
  name: deny-from-other-namespaces
spec:
  podSelector:
    matchLabels:
  ingress:
  - from:
    - podSelector: {}
```

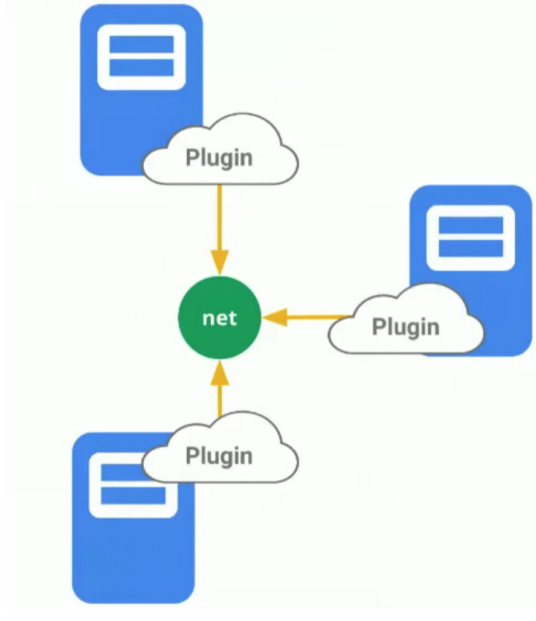
Network Providers

- All containers can communicate with all other containers without NAT
- All nodes can communicate with all containers (and vice-versa) without NAT
- The IP that a container sees itself as is the same IP that others see it as

Network Providers

Сравнение производительности некоторых сетевых решений для Kubernetes

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



Network Providers

Обзор network providers (2017 г.)

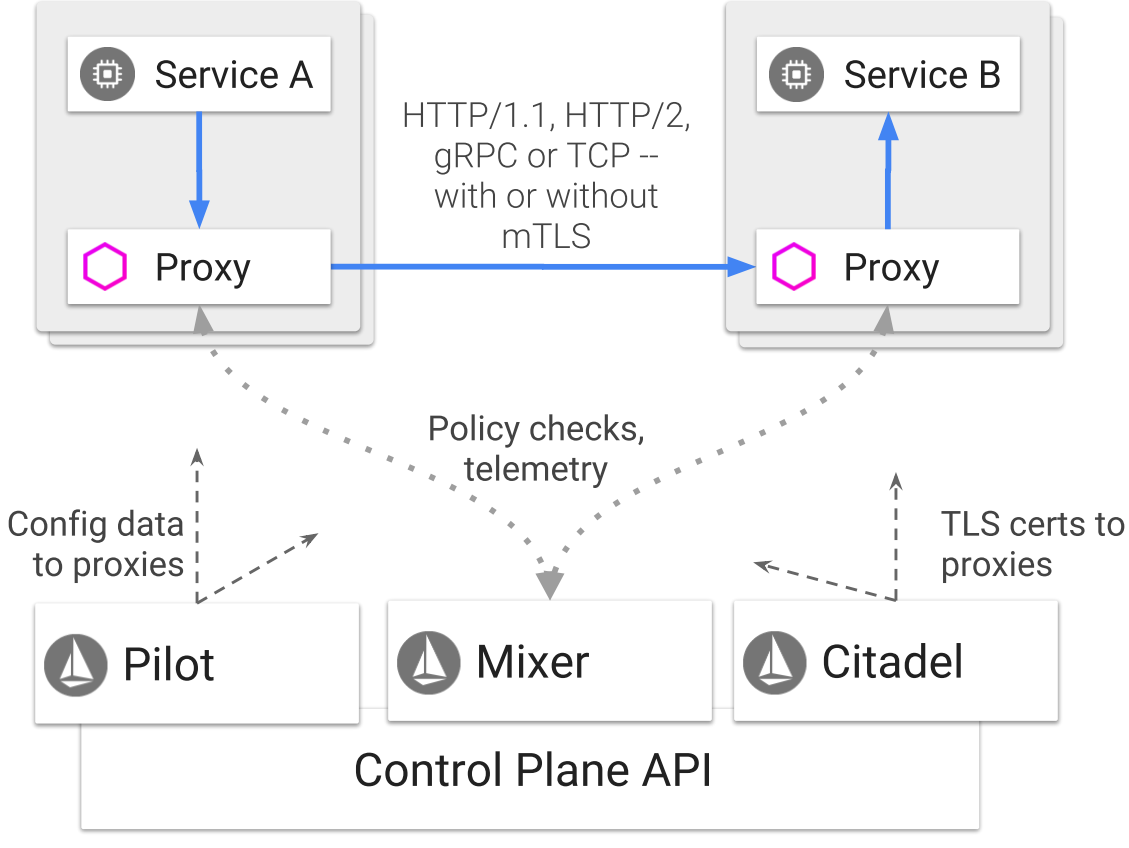
Provider	Network Model	Route Distribution	Network Policies	Mesh	External Datastore	Encryption	Ingress/Egress Policies	Commercial Support
Calico	Layer 3	Yes	Yes	Yes	Etcd [1]	Yes	Yes	Yes
Canal	Layer 2 vxlan	N/A	Yes	No	Etcd [1]	No	Yes	No
flannel	vxlan	No	No	No	None	No	No	No
kopeio-networking	Layer 2 vxlan [2]	N/A	No	No	None	Yes [3]	No	No
kube-router	Layer 3	BGP	Yes	No	No	No	No	No
romana	Layer 3	OSPF	Yes	No	Etcd	No	Yes	Yes
Weave Net	Layer 2 vxlan [4]	N/A	Yes	Yes	No	Yes	Yes [5]	Yes

Service Mesh

- Умная сетевая инфраструктура, выносим логику взаимодействия из приложения в инфраструктуру
- Observability
- Security
- Load balancing (canary releases, etc...)
- Circuit breakers
- Sidecar proxy
- Memory/latency overhead

Доклад Ивана Круглова (Booking.com) про Service Mesh

Istio


Service A
Proxy
HTTP/1.1, HTTP/2,
gRPC or TCP --
with or without
mTLS
Service B
Proxy
Policy checks,
telemetry
Config data
to proxies
TLS certs to
proxies
Pilot
Mixer
Citadel
Control Plane API

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 и удаляется вместе с удалением Pod. Используется для обмена данными между контейнерами внутри Pod.
- **hostPath** - директория на Node, не удаляется при удалении Pod. Используется для доступа к информации на Node, например к /var/lib/docker/

emptyDir

```
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: {}
```

Используем tmpfs для emptyDir:

```
emptyDir:
      medium: Memory
```

Cloud Volumes

- AWS EBS
- AzureDisk и AzureFile
- gcePersistentDisk

gcePersistentDisk

- Надо предварительно создать диск в GCP
- После удаления pod данные на диске останутся

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: nginx
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    gcePersistentDisk:
      pdName: my-data-disk
      fsType: ext4
```

Custom Volumes

- nfs
- CephFS и GlusterFS
- FC и iSCSI

Persistent Volumes

- PersistentVolume
- PersistentVolumeClaim
- StorageClass
- Local Volumes

Persistent Volumes

- Создаются на уровне кластера
- PV похожи на обычные Volume, но имеют отдельный от сервисов жизненный цикл

Persistent Volumes

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongodb-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  gcePersistentDisk:
    pdName: mongodb
    fsType: ext4
```

Persistent Volumes: Phase

- **Available** — доступен, не смонтирован
- **Bound** — смонтирован
- **Released** — размонтирован
- **Failed** — что-то сломалось при запросе

Persistent Volumes: Reclaim Policy

- **Retain** — не удалять
- **Recycle** — переиспользовать (deprecated)
- **Delete** (default) — удалять

PersistentVolumeClaim

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

PersistentVolumeClaim

- Access Modes
- Resources
- Selector
- Class

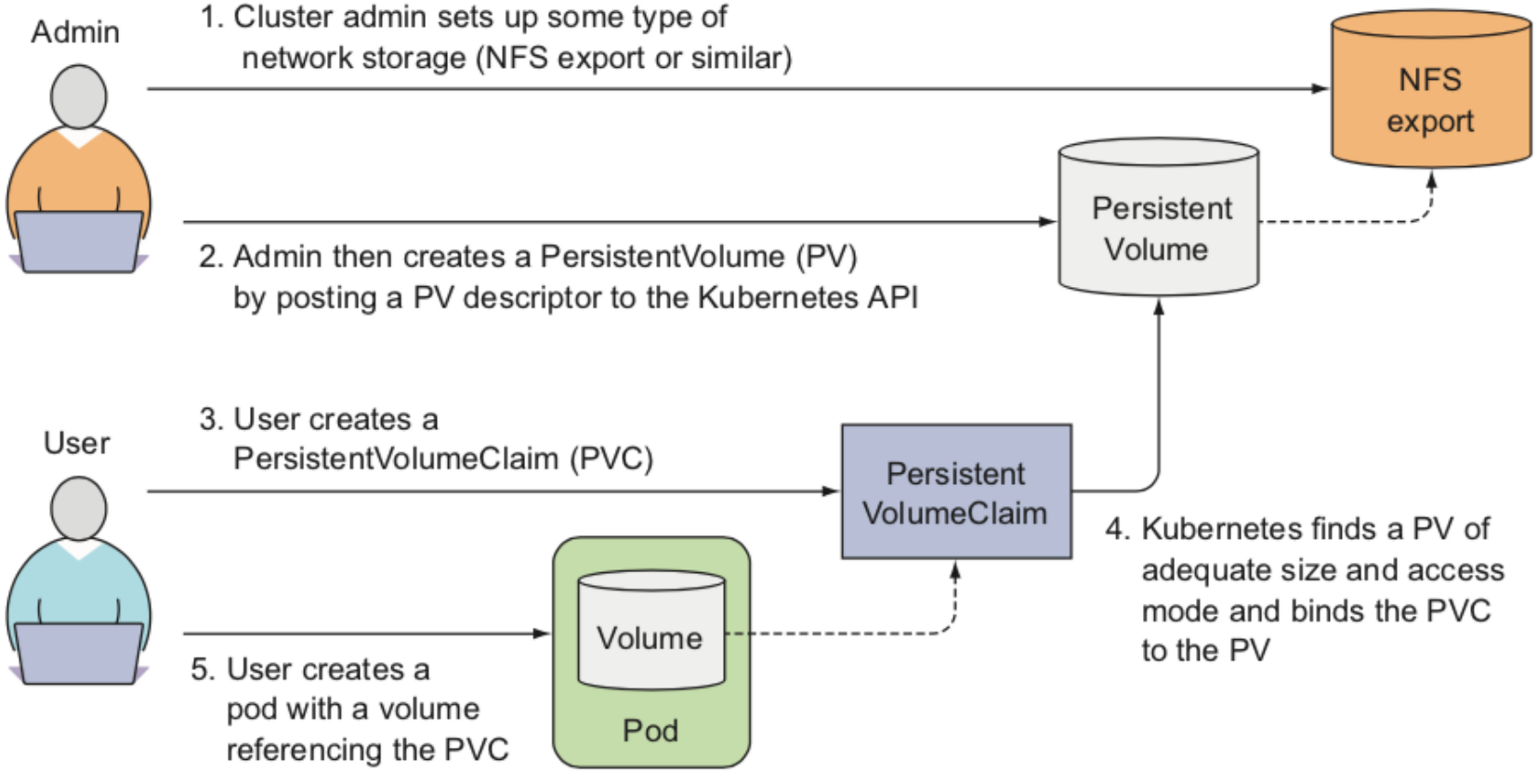
PersistentVolumeClaim

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
  - ReadWriteOnce
  selector:
    matchLabels:
      release: "stable"
```

PVC as Volume

```
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  containers:
  - image: mongo
    name: mongodb
    volumeMounts:
    - name: mongodb-data
      mountPath: /data/db
    ports:
    - containerPort: 27017
      protocol: TCP
  volumes:
  - name: mongodb-data
    persistentVolumeClaim:
      claimName: mongodb-pvc
```

Static Workflow



Storage Classes

- Описание "классов" различных систем хранения
- Разные классы могут использоваться для:
  - Произвольных политик
  - Динамического provisioning

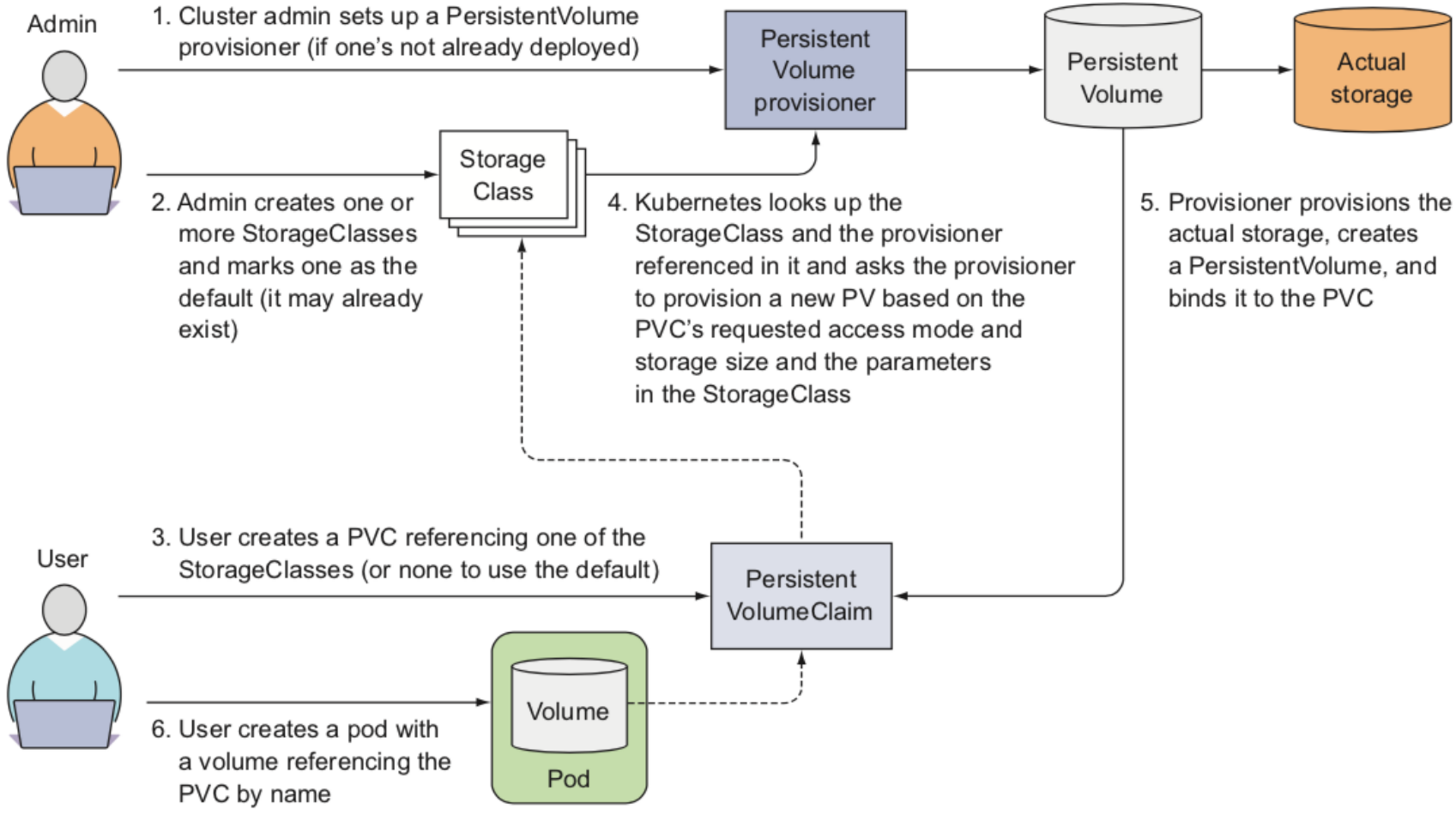
Storage Classes

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
  zone: europe-west1-c
```

PersistentVolumeClaim с StorageClass

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  storageClassName: fast
  resources:
    requests:
      storage: 100Mi
  accessModes:
    - ReadWriteOnce
```

Dynamic Workflow



Dynamic Volume Provisioning

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

Жизненный цикл PV

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

Базы данных и Kubernetes

В целом да, но...

Статья и доклад Дмитрия Столярова (Флант), Highload 2018