

Kubernetes.
Запуск кластера и
приложения.
Модель безопасности.

Подготовка

Создайте новую ветку в репозитории **Microservices** для выполнения данного ДЗ. Т.к. это третье задание по Kubernetes, то назовите ее **kubernetes-3**

Проверка данного ДЗ будет производиться через Pull Request (PR назовите **kubernetes-3**) ветки с ДЗ к ветке мастер и добавление в Reviewers пользователей **Nklya, chromko, vitkhab**.

После того, как один из преподавателей сделает approve пул реквеста, ветку с ДЗ можно смерджить.

План

- Ingress Controller
- Ingress
- Secret
- TLS
- LoadBalancer Service
- Network Policies
- PersistentVolumes
- PersistentVolumeClaims

Сетевое взаимодействие

В предыдущей работе нам уже довелось настраивать сетевое взаимодействие с приложением в Kubernetes с помощью **Service** - абстракции, определяющей конечные узлы доступа (Endpoint'ы) и способ коммуникации с ними (nodePort, LoadBalancer, ClusterIP). Разберем чуть подробнее что в реальности нам это дает.

Service

Service - определяет **конечные узлы доступа** (Endpoint'ы):

- селекторные сервисы (k8s сам находит POD-ы по label'ам)
- безселекторные сервисы (мы вручную описываем конкретные endpoint'ы)

и **способ коммуникации** с ними (тип (type) сервиса):

- ClusterIP - дойти до сервиса можно только изнутри кластера
- nodePort - клиент снаружи кластера приходит на опубликованный порт
- LoadBalancer - клиент приходит на облачный (aws elb, Google gclb) ресурс балансировки
- ExternalName - внешний ресурс по отношению к кластеру

Service

Вспомним, как выглядели Service'ы:

post-service.yml ([ссылка на gist](#))

```
---
apiVersion: v1
kind: Service
metadata:
  name: post
  labels:
    app: reddit
    component: post
spec:
  ports:
    - port: 5000
      protocol: TCP
      targetPort: 5000
  selector:
    app: reddit
    component: post
```

Это селекторный сервис
типа **ClusterIP** (тип не указан, т.к.
этот тип по-умолчанию).

Service

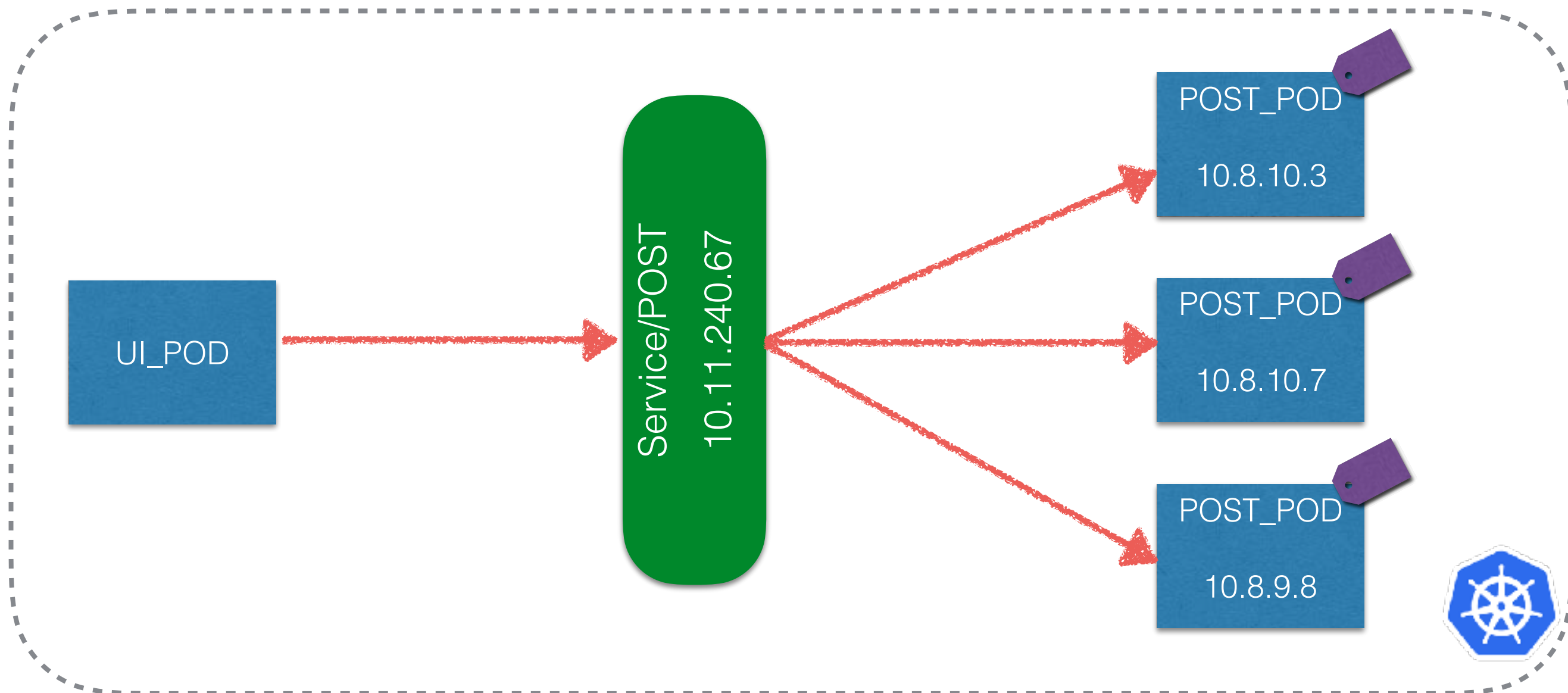
ClusterIP - это виртуальный (в реальности нет интерфейса, pod'а или машины с таким адресом) IP-адрес из диапазона адресов для работы внутри, скрывающий за собой IP-адреса реальных POD-ов. Сервису любого **типа** (кроме ExternalName) назначается этот IP-адрес.

```
$ kubectl get services -n dev
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
comment	ClusterIP	10.11.248.14	<none>	9292/TCP	1d
comment-db	ClusterIP	10.11.252.16	<none>	27017/TCP	1d
post	LoadBalancer	10.11.240.67	35.193.193.168	5000:30298/TCP	1d
post-db	ClusterIP	10.11.242.112	<none>	27000/TCP	18m
ui	NodePort	10.11.253.107	<none>	9292:30221/TCP	1d

Service

Схема взаимодействия



Kube-dns

Отметим, что **Service** - это лишь абстракция и описание того, как получить доступ к сервису. Но опирается она на реальные механизмы и объекты: DNS-сервер, балансировщики, iptables.

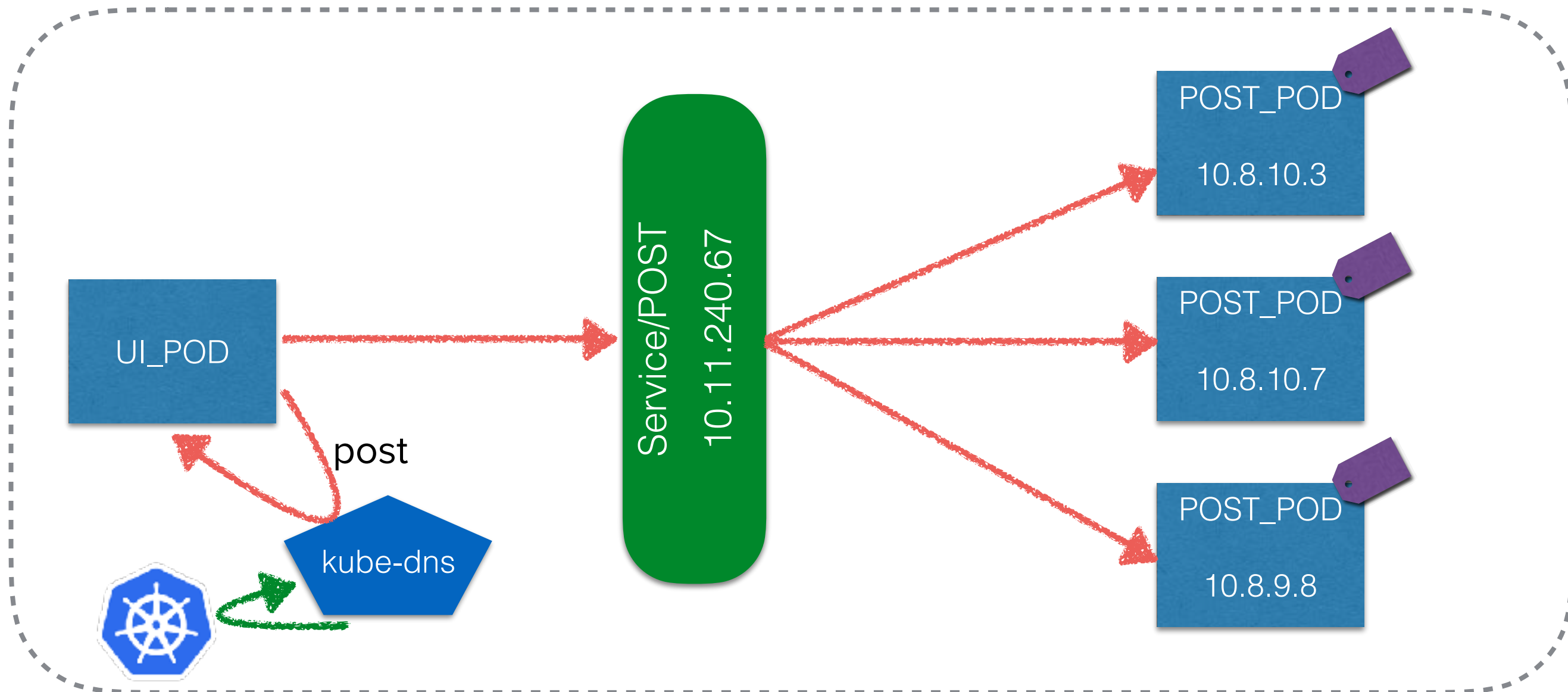
Для того, чтобы дойти до сервиса, нам нужно узнать его адрес по имени. Kubernetes не имеет своего собственного DNS-сервера для разрешения имен. Поэтому используется плагин **kube-dns** (это тоже Pod).

Его задачи:

- ходить в API Kubernetes'а и отслеживать Service-объекты
- заносить DNS-записи о Service'ах в собственную базу
- предоставлять DNS-сервис для разрешения имен в IP-адреса (как внутренних, так и внешних)

kube-dns

Схема приобретает следующий вид



Kube-dns

Можете убедиться, что при отключенном **kube-dns** сервисе связность между компонентами reddit-app пропадет и он перестанет работать.

1) Проскейлим в 0 сервис, который следит, чтобы dns-kube подов всегда хватало ([ссылка на gist](#))

```
$ kubectl scale deployment --replicas 0 -n kube-system kube-dns-autoscaler
```

2) Проскейлим в 0 сам kube-dns ([ссылка на gist](#))

```
$ kubectl scale deployment --replicas 0 -n kube-system kube-dns
```

Kube-dns

3) Попробуйте достучаться по имени до любого сервиса.
Например ([ссылка на gist](#)):

```
$ kubectl exec -ti -n dev <имя любого pod-a> ping comment
```

```
ping: bad address 'comment'  
command terminated with exit code 1
```

4) Вернем kube-dns-autoscale в исходную ([ссылка на gist](#))

```
$ kubectl scale deployment --replicas 1 -n kube-system kube-dns-autoscaler
```

5) Проверьте, что приложение заработало (в браузере)

Service

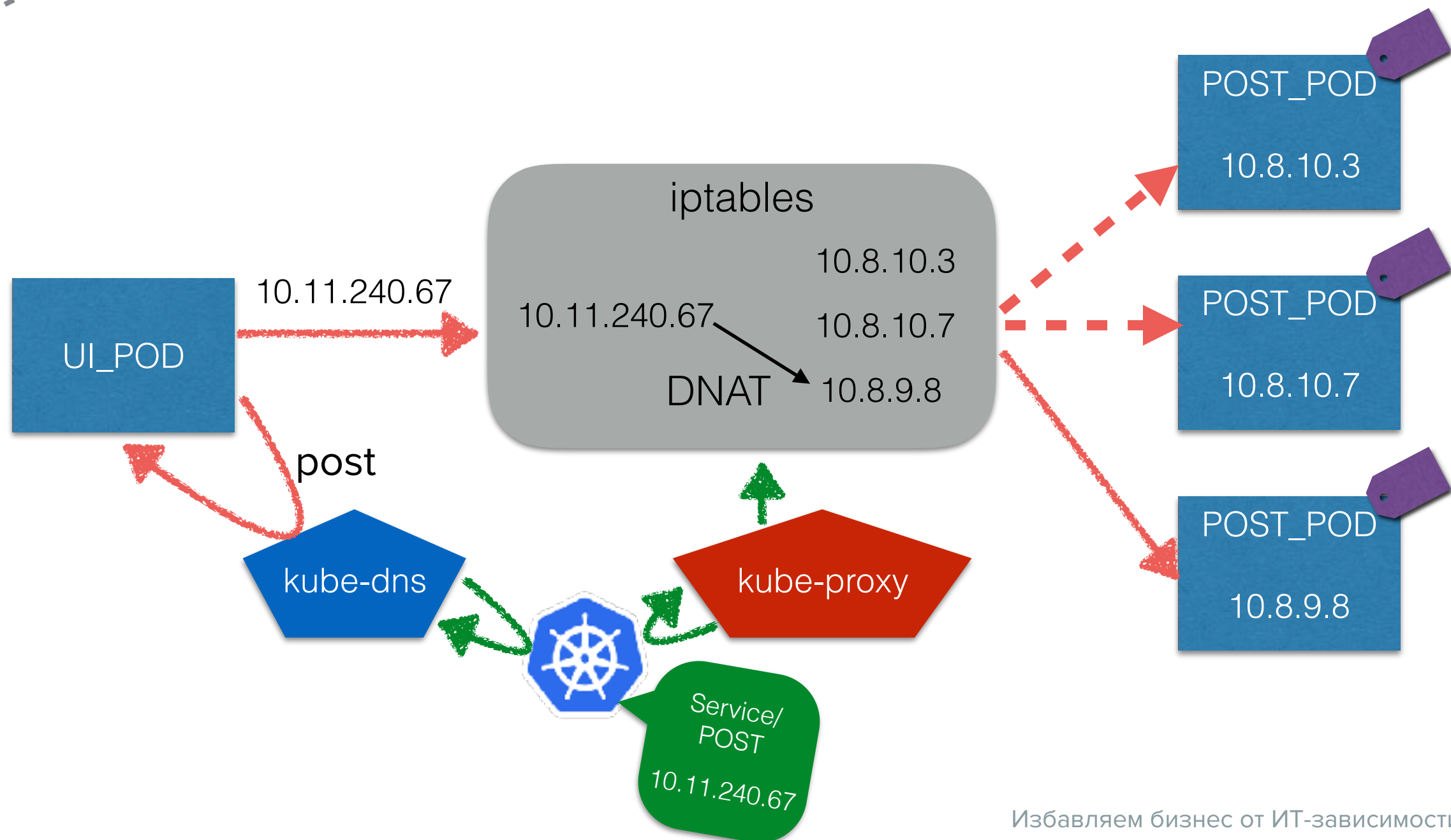
Как уже говорилось, **ClusterIP** - виртуальный и не принадлежит ни одной реальной физической сущности. Его чтением и дальнейшими действиями с пакетами, принадлежащими ему, занимается в нашем случае **iptables**, который настраивается утилитой **kube-proxy** (забирающей инфу с API-сервера).

Сам kube-proxy, можно настроить на прием трафика, но это устаревшее поведение и не рекомендуется его применять.

На любой из нод кластера можете посмотреть эти правила **IPTABLES** (это не задание).

kube-dns

Схема приобретает следующий вид



А в рамках кластера?

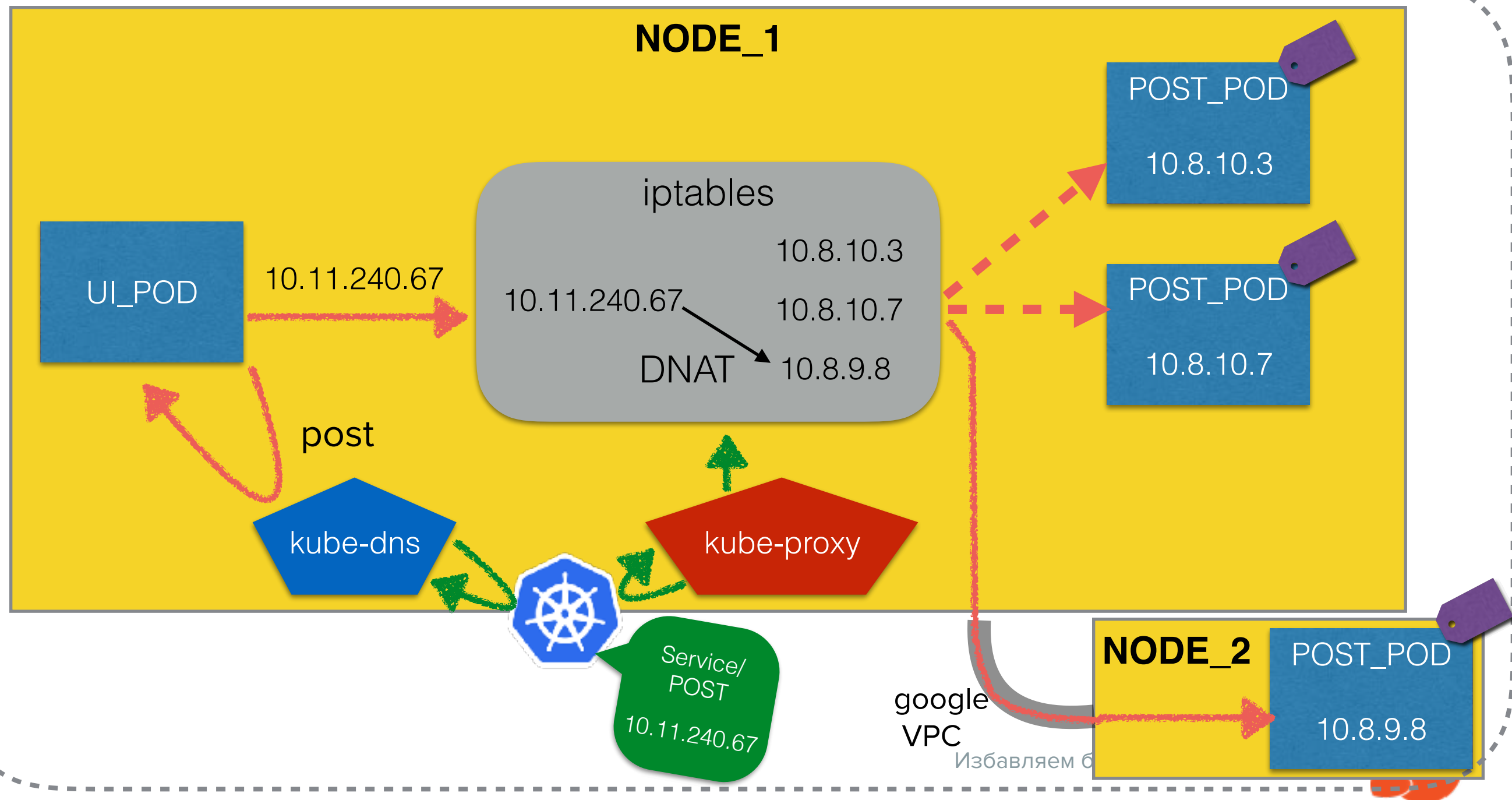
На самом деле независимо от того, на одной ноде находятся поды или на разных - трафик проходит через цепочку, изображенную на предыдущем слайде.

Kubernetes не имеет в комплекте механизма организации overlay-сетей (как у Docker Swarm). Он лишь предоставляет интерфейс для этого. Для создания Overlay-сетей используются отдельные аддоны: Weave, Calico, Flannel, В Google Kontainer Engine (GKE) используется собственный плагин **kubenet** (он - часть kubelet).

Он работает **только** вместе с платформой **GCP** и, по-сути занимается тем, что настраивает google-сети для передачи трафика Kubernetes. Поэтому в конфигурации Docker сейчас вы не увидите никаких Overlay-сетей.

kubenet

Схема приобретает следующий вид



А в рамках кластера?

Посмотреть правила, согласно которым трафик отправляется на ноды можно здесь:

<https://console.cloud.google.com/networking/routes/>

<input type="checkbox"/>	gke-cluster-1-b7bd426c-17c43352-d34c-11e7-a267-42010a80001b	10.8.11.0/24	1000	Het	gke-cluster-1-bigpool-8e29c447-r9s8 (зона us-central1-a)	default
<input type="checkbox"/>	gke-cluster-1-b7bd426c-f8f6cf94-d34b-11e7-a267-42010a80001b	10.8.9.0/24	1000	Het	gke-cluster-1-default-pool-f9c66281-pz6r (зона us-central1-a)	default
<input type="checkbox"/>	gke-cluster-1-b7bd426c-f8fbc1a1-d34b-11e7-a267-42010a80001b	10.8.10.0/24	1000	Het	gke-cluster-1-default-pool-f9c66281-xnwz (зона us-central1-a)	default

nodePort

Service с типом **NodePort** - похож на сервис типа **ClusterIP**, только к нему прибавляется прослушивание портов нод (всех нод) для доступа к сервисам **снаружи**. При этом ClusterIP также назначается этому сервису для доступа к нему изнутри кластера.

kube-proxy прослушивается либо заданный порт (nodePort: 32092), либо порт из диапазона 30000-32670.

Дальше IPTables решает, на какой Pod попадет трафик.

nodePort

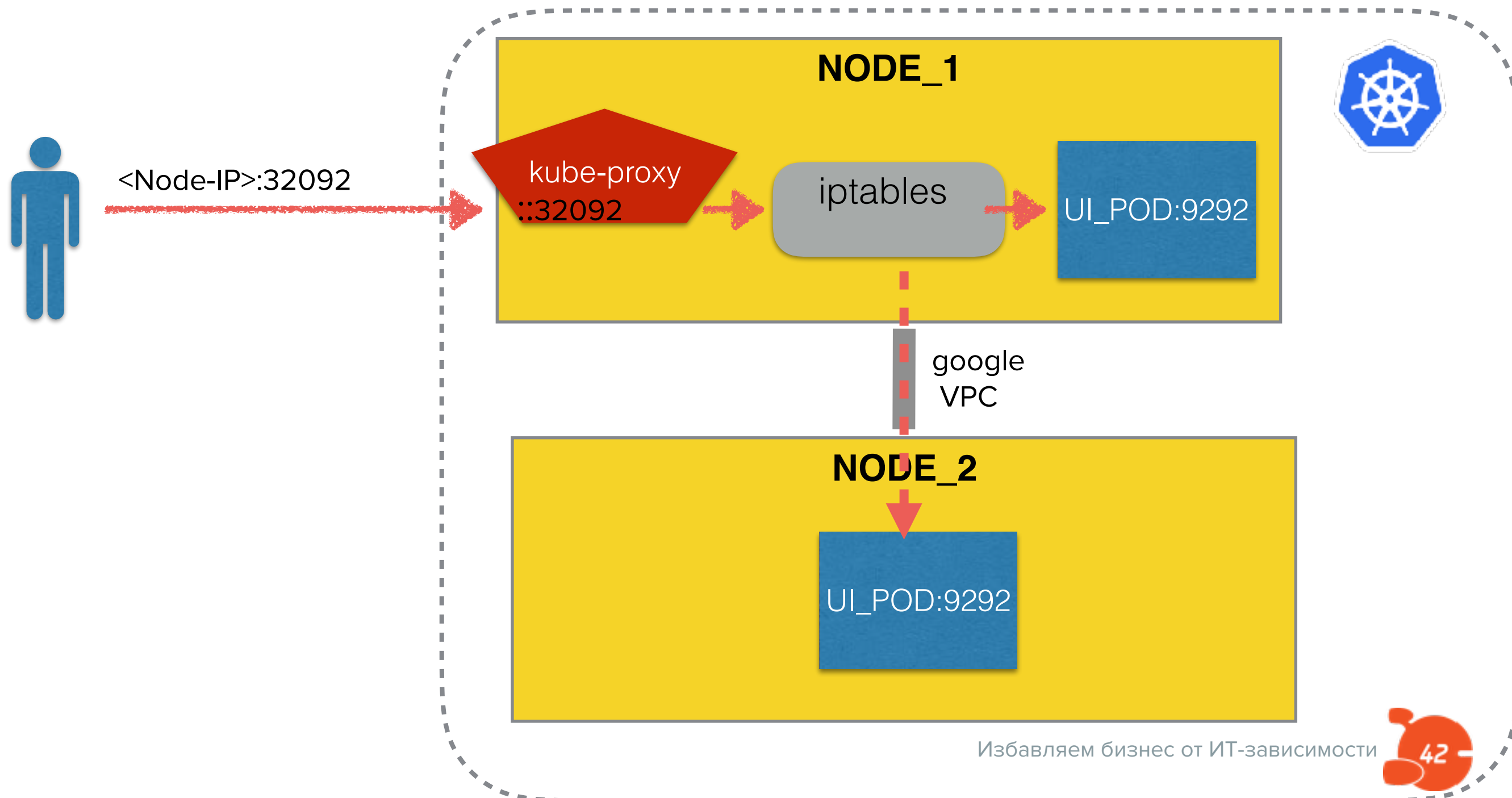
ui-service.yml ([ссылка на gist](#))

```
---
apiVersion: v1
kind: Service
metadata:
  name: ui
  labels:
    app: reddit
    component: ui
spec:
  type: NodePort
  ports:
    - port: 9292
      nodePort: 32092
      protocol: TCP
      targetPort: 9292
  selector:
    app: reddit
    component: ui
```

Сервис UI мы уже
публиковали наружу с
помощью NodePort

nodePort

Схема приобретает следующий вид



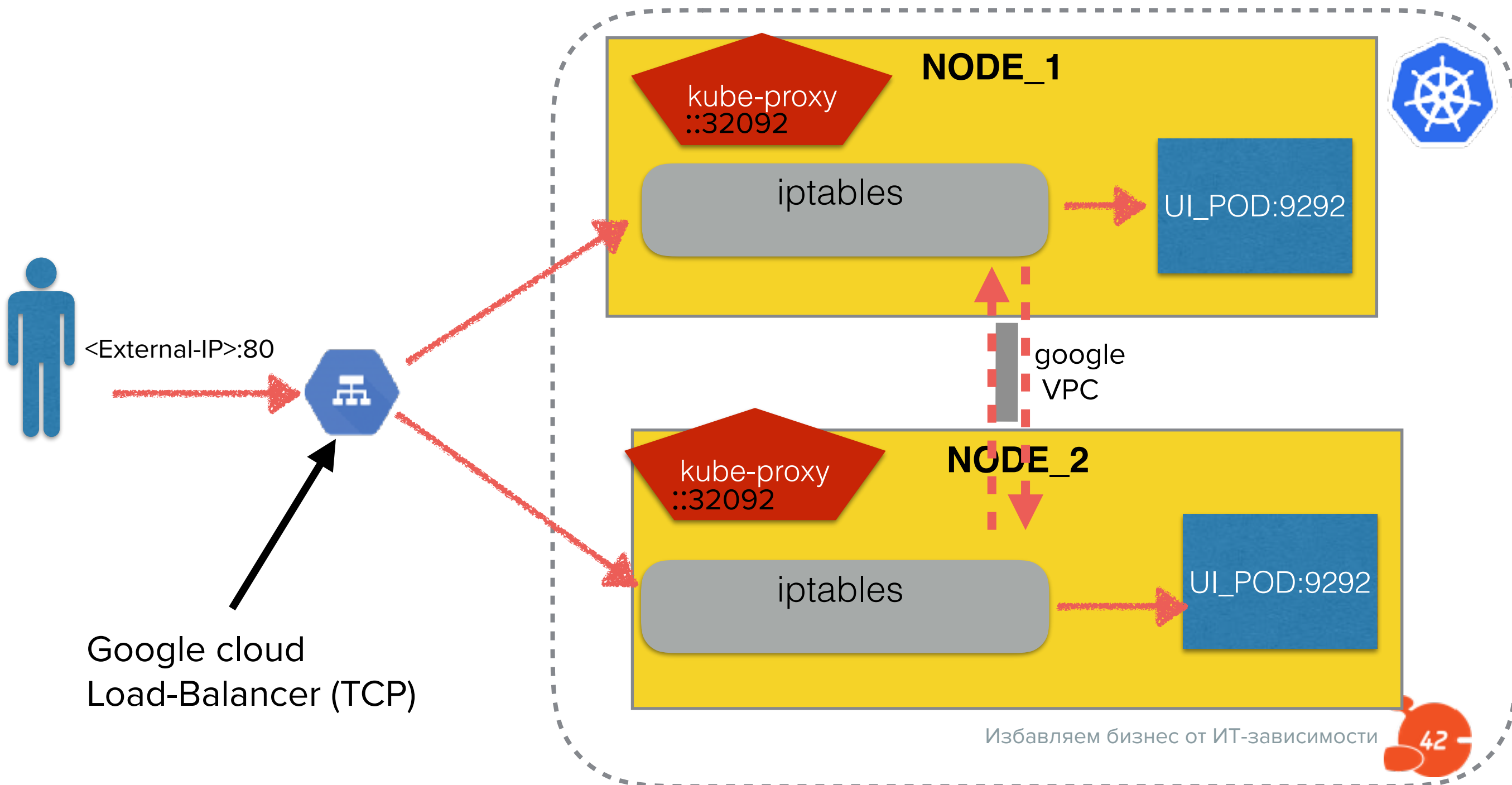
LoadBalancer

Тип NodePort хоть и предоставляет доступ к сервису снаружи, но открывать все порты наружу или искать IP-адреса наших нод (которые вообще динамические) не очень удобно.

Тип **LoadBalancer** позволяет нам использовать **внешний облачный** балансировщик нагрузки как единую точку входа в наши сервисы, а не полагаться на IPTables и не открывать наружу весь кластер.

LoadBalancer

Схема приобретает следующий вид



LoadBalancer

Настроим соответствующим образом Service UI

ui-service.yml ([ссылка на gist](#))

apiVersion: v1

kind: Service

metadata:

name: ui

labels:

app: reddit

component: ui

spec:

type: LoadBalancer

ports:

- port: 80

nodePort: 32092

protocol: TCP

targetPort: 9292

selector:

app: reddit

component: ui

1) Порт, который будет открыт на балансировщике

2) Также на ноде будет открыт порт, но нам он не нужен и его можно даже убрать

3) Порт POD-а

LoadBalancer

Настроим соответствующим образом Service UI

```
$ kubectl apply -f ui-service.yml -n dev
```

Посмотрим что там ([ссылка на gist](#))

```
$ kubectl get service -n dev --selector component=ui
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
ui	LoadBalancer	10.11.243.150	<pending>	80:31850/TCP	1s

Немного подождем (идет настройка ресурсов GCP)

```
$ kubectl get service -n dev --selector component=ui
```

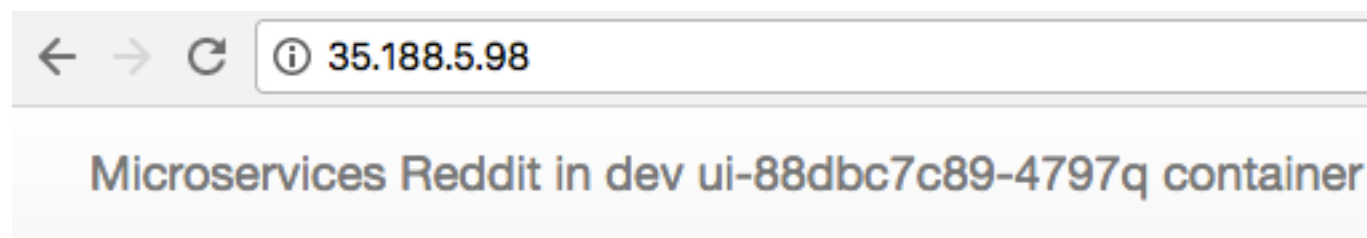
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
ui	LoadBalancer	10.11.243.150	35.188.5.98	80:31850/TCP	1s

Наш адрес: 35.188.5.98:80



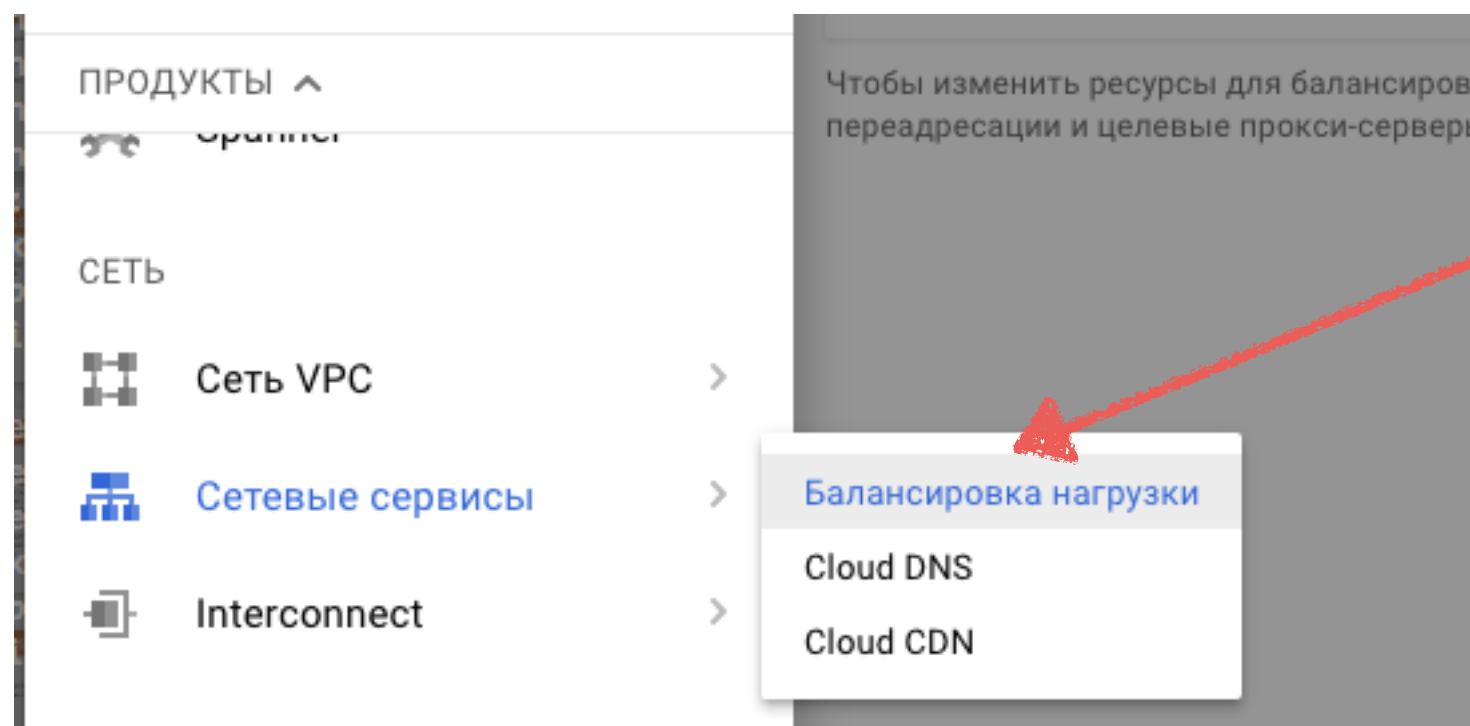
LoadBalancer

Проверим в браузере: `http://<external-ip>:port`



А что за кулисами? Откроем консоль GCP:

Сюда



LoadBalancer

Будет создано правило для балансировки.

Балансировщики нагрузки

Серверные VM

Интерфейсные VM

Балансировщик
нагрузки

Протокол

✓ a7e7cbfe5d43711e78c9842010a80001

Интерфейсная VM

Протокол	IP-адрес: порт
TCP	35.188.5.98:80

Серверная VM

Название: a7e7cbfe5d43711e78c9842010a80001 Регион: us-central1 Привязка сеанса: Н

Экземпляры	35.188.5.98
gke-cluster-1-default-pool-f9c66231-pz6r	✓
gke-cluster-1-default-pool-f9c66231-xnw7	✓
gke-cluster-1-bigpool-8e29c447-r9s8	✓

LoadBalancer

Балансировка с помощью Service типа LoadBalancing имеет ряд недостатков:

- нельзя управлять с помощью http URI (L7-балансировка)
- используются **только** облачные балансировщики (AWS, GCP)
- нет гибких правил работы с трафиком

Ingress

Для более удобного управления входящим снаружи трафиком и решения недостатков LoadBalancer можно использовать другой объект Kubernetes - **Ingress**.

Ingress – это набор правил внутри кластера Kubernetes, предназначенных для того, чтобы входящие подключения могли достичь сервисов (Services)

Сами по себе Ingress'ы это просто правила. Для их применения нужен **Ingress Controller**.

Ingress Controller

Для работы Ingress-ов необходим **Ingress Controller**.
В отличие остальных контроллеров k8s - он не запускается вместе с кластером.

Ingress Controller - это скорее плагин (а значит и отдельный POD), который состоит из 2-х функциональных частей:

- Приложение, которое отслеживает через k8s API новые объекты Ingress и обновляет конфигурацию балансировщика
- Балансировщик (Nginx, haproxy, traefik,...), который и занимается управлением сетевым трафиком

Ingress

Основные задачи, решаемые с помощью Ingress'ов:

- Организация единой точки входа в приложения снаружи
- Обеспечение балансировки трафика
- Терминация SSL
- Виртуальный хостинг на основе имен и т.д

Ingress

Поскольку у нас web-приложение, нам вполне было бы логично использовать L7-балансировщик вместо Service LoadBalancer.

Google в GKE уже предоставляет возможность использовать их собственные решения балансировщик в качестве Ingress controller-ов.

Перейдите в настройки кластера в веб-консоли gcloud

Ingress

Убедитесь, что встроенный Ingress включен.
Если нет - включите

[←](#) Kubernetes clusters [✎ ИЗМЕНИТЬ](#) [🗑 УДАЛИТЬ](#) [🔌 CONNECT ▼](#)

Сетевая политика	Отключена
Устаревшие права доступа	Включено
Перерыв на техническое обслуживание	В любое время

Ярлыки

Нет ярлыков

Дополнения

Панель управления Kubernetes	Включено
Балансировка нагрузки HTTP	Включено

⤴ Скрыть

⤵ Права доступа

Ingress

Создадим Ingress для сервиса UI

ui-ingress.yml ([ссылка на gist](#))

```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ui
spec:
  backend:
    serviceName: ui
    servicePort: 80
```

Это **Singe Service Ingress** - значит, что весь ingress контроллер будет просто балансировать нагрузку на Node-ы для одного сервиса (очень похоже на Service LoadBalancer)

Ingress

Зайдем в консоль GCP и увидим уже несколько правил

Балансировка нагрузки		+ СОЗДАТЬ БАЛАНСИРОВЩИК НАГРУЗКИ
Балансировщики нагрузки		Серверные VM Интерфейсные VM
Балансировщик нагрузки	Протокол	
✓ k8s-um-dev-ui--206a088ba6bb0cad	HTTP(S)	
✓ af68f09c1d45811e78c9842010a80001	TCP	

Чтобы изменить ресурсы для балансировки нагрузки, в том числе правила переадресации и целевые прокси-серверы, откройте [расширенное меню](#).

Ingress

Нас интересует 1-е

Правила обработки хостов и путей

Хосты ^	Пути	Серверная VM
Все незадаанные (по умолчанию)	Все незадаанные (по умолчанию)	k8s-be-30229--206a0

Серверная VM

Серверные службы

1. k8s-be-30229--206a088ba6bb0cad

Endpoint protocol: HTTP Именованный порт: port30229 ← Время ожидания в секундах: 30 П

Cloud CDN: **отключен**

Время запрета новых соединений: 0 сек. Прокси-сервер с идентификацией (IAP): **отключен**

Это NodePort
опубликованного сервиса

Т.е. для работы с Ingress в
GCP нам нужен минимум
Service с типом NodePort
(он уже есть)

Ingress

Посмотрим в сам кластер:

```
$ kubectl get ingress -n dev
```

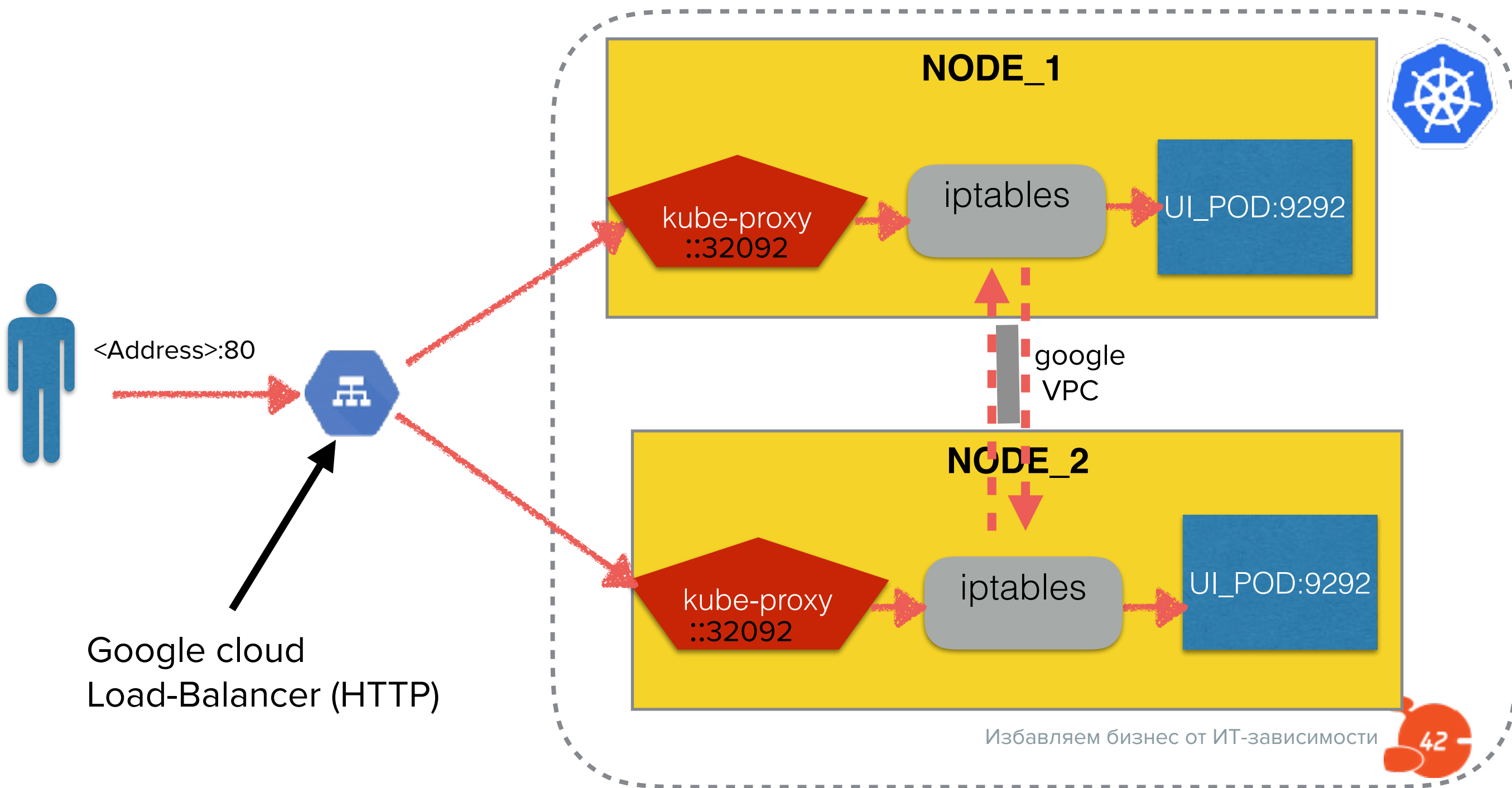
NAME	HOSTS	ADDRESS	PORTS	AGE
ui	*	35.190.66.90	80	23m

Адрес сервиса (если не появился, подождите)
<http://35.190.66.90:80>



Ingress

Схема приобретает следующий вид



Ingress

В текущей схеме есть несколько недостатков:

- у нас 2 балансировщика для 1 сервиса
- Мы не умеем управлять трафиком на уровне HTTP

Ingress

Один балансировщик можно спокойно убрать. Обновим сервис для UI

ui-service.yml ([ссылка на gist](#))

```
---
apiVersion: v1
kind: Service
metadata:
  name: ui
  labels:
    app: reddit
    component: ui
spec:
  type: NodePort
  ports:
    - port: 9292
      protocol: TCP
      targetPort: 9292
  selector:
    app: reddit
    component: ui
```

Применим

```
$ kubectl apply -f ... -n dev
```

Ingress

Заставим работать Ingress Controller как классический веб

ui-ingress.yml ([ссылка на gist](#))

```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ui
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: ui
          servicePort: 9292
```


Ingress

← → ↻ ⓘ 35.190.66.90

Microservices Reddit in dev ui-76ffdbbf6b-zs4rr container

Secret

Теперь давайте защитим наш сервис с помощью TLS.
Для начала вспомним Ingress IP

```
$ kubectl get ingress -n dev
```

NAME	HOSTS	ADDRESS	PORTS	AGE
ui	*	35.190.66.90	80	23m

Далее подготовим сертификат используя IP как CN

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout tls.key -out tls.crt -subj "/CN= 35.190.66.90"
```

И загрузит сертификат в кластер kubernetes

```
$ kubectl create secret tls ui-ingress --key tls.key --cert tls.crt -n dev
```

Проверить можно командой

```
$ kubectl describe secret ui-ingress -n dev
```

TLS Termination

Теперь настроим Ingress на прием только HTTPS траффика

ui-ingress.yml ([ссылка на gist](#))

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: ui

annotations:

kubernetes.io/ingress.allow-http: "false"

spec:

tls:

- secretName: ui-ingress

backend:

serviceName: ui

servicePort: 9292

Отключаем проброс HTTP

Подключаем наш сертификат

Применим

```
$ kubectl apply -f ui-ingress.yml -n dev
```

TLS Termination

Зайдем на страницу web console и увидим в описании нашего балансировщика только один протокол HTTPS

Frontend


Protocol ^	IP:Port	Certificate
HTTPS	35.227.230.121:443	k8s-ssl-dev-ui--8ffc067eddc9e0

Иногда протокол HTTP может не удалиться у существующего Ingress правила, тогда нужно его вручную удалить и пересоздать

```
$ kubectl delete ingress ui -n dev  
$ kubectl apply -f ui-ingress.yml -n dev
```

TLS Termination

Заходим на страницу нашего приложения по https, подтверждаем исключение безопасности (у нас сертификат самоподписанный) и видим что все работает



🔒 https://35.227.230.121

Microservices Reddit in ui-64c9489bd5-4d9ml container

Правила Ingress могут долго применяться, если не получилось зайти с первой попытки - подождите и попробуйте еще раз

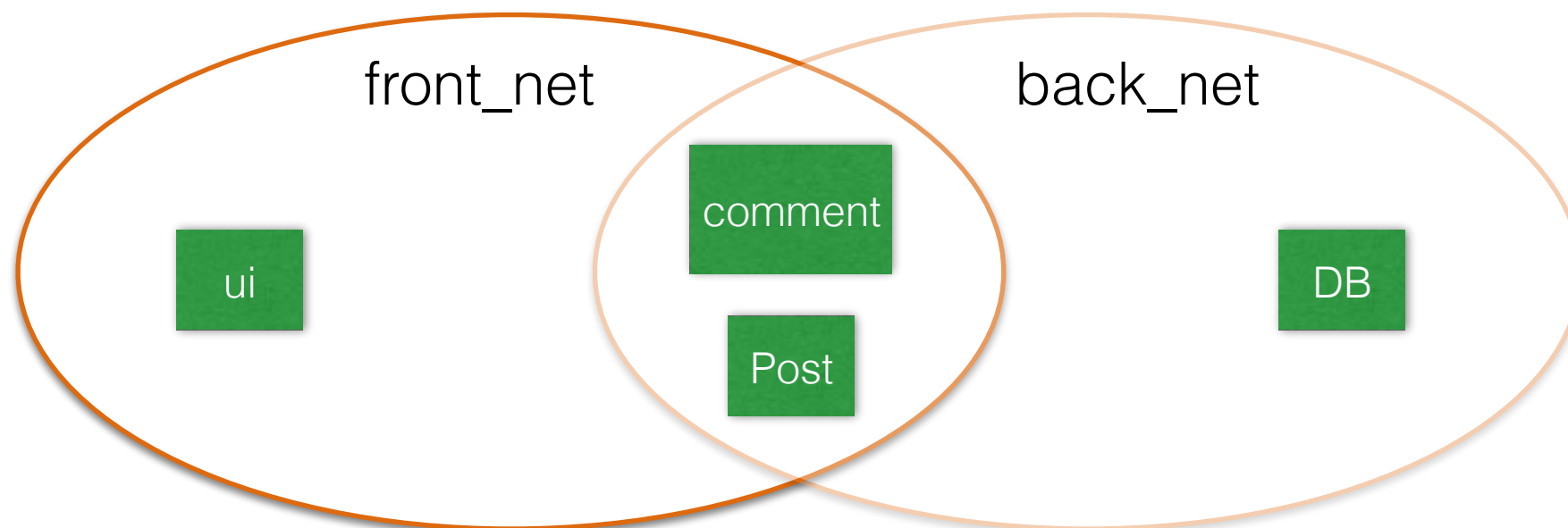
Задание



Опишите создаваемый объект Secret в виде Kubernetes-манифеста.

Network Policy

В прошлых проектах мы договорились о том, что хотелось бы разнести сервисы базы данных и сервис фронтенда по разным сетям, сделав их недоступными друг для друга. И приняли следующую схему сервисов.



В Kubernetes у нас так сделать не получится с помощью отдельных сетей, так как все POD-ы могут достигаться друг до друга по умолчанию.

Network Policy

Мы будем использовать **NetworkPolicy** - инструмент для декларативного описания потоков трафика. Отметим, что не все сетевые плагины поддерживают политики сети. В частности, у GKE эта функция пока в Beta-тесте и для её работы отдельно будет включен сетевой плагин **Calico** (вместо Kubenet).

Давайте ее протестируем.

Наша задача - ограничить трафик, поступающий на mongodb отовсюду, кроме сервисов post и comment.

Network Policy

Найдите имя кластера

```
$ gcloud beta container clusters list
```

NAME	ZONE	MASTER_VERSION	MASTER_IP	MACHINE_TYPE	NODE_VERSION	NUM_NODES	STATUS
cluster-1	us-central1-a	1.8.3-gke.0	35.202.145.198	g1-small	1.8.3-gke.0	3	RUNNING

Включим network-policy для GKE. ([ссылка на gist](#))

```
$ gcloud beta container clusters update <cluster-name> \
  --zone=us-central1-a --update-addons=NetworkPolicy=ENABLED
Updating cluster-1...
```

```
$ gcloud beta container clusters update <cluster-name> \
  --zone=us-central1-a --enable-network-policy
```

Дождитесь, пока кластер обновится

Вам может быть предложено добавить beta-функционал в gcloud - нажмите yes.

Network Policy

mongo-network-policy.yml ([ссылка на gist](#))

```
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-db-traffic
  labels:
    app: reddit
spec:
  podSelector:
    matchLabels:
      app: reddit
      component: mongo
  policyTypes:
    - Ingress
  ingress:
    - from:
      - podSelector:
          matchLabels:
            app: reddit
            component: comment
```

Выбираем объекты

Блок запрещающих направлений

Блок разрешающих правил
(белый список)

Network Policy

```
podSelector:  
  matchLabels:  
    app: reddit  
    component: mongo
```

Выбираем объекты политики (pod'ы с mongodb)

```
policyTypes:  
- Ingress
```

Запрещаем все входящие подключения
Исходящие разрешены

```
ingress:  
- from:  
  - podSelector:  
    matchLabels:  
      app: reddit  
      component: comment
```

Разрешаем все входящие подключения от
POD-ов с label-ами comment.

Network Policy

Применяем политику

```
$ kubectl apply -f mongo-network-policy.yml -n dev
```

Заходим в приложение

Microservices Reddit in dev ui-76ffdbbf6b-7zxsg container

Can't show blog posts, some problems with the post service. Refresh?

Post-сервис не может достучаться до базы.

Задание

Обновите **mongo-network-policy.yml** так, чтобы post-сервис дошел до базы данных.

Хранилище для базы

Рассмотрим вопросы хранения данных. Основной Stateful сервис в нашем приложении - это база данных MongoDB. В текущий момент она запускается в виде Deployment и хранит данные в стандартный Docker Volume-ах. Это имеет несколько проблем:

- при удалении POD-а удаляется и Volume
- потеря Nod'ы с mongo грозит потерей данных
- запуск базы на другой ноде запускает новый экземпляр данных

Хранилище для базы

mongo-deployment.yml ([ссылка на gist](#))

```
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: mongo
```

...

spec:

containers:

- image: mongo:3.2
- name: mongo

volumeMounts:

- name: mongo-persistent-storage
- mountPath: /data/db

volumes:

- name: mongo-persistent-storage
- emptyDir: {}

Подключаем Volume

Объявляем Volume

Volume

Сейчас используется тип Volume **emptyDir**. При создании пода с таким типом просто создается пустой docker volume. При остановке POD'а содержимое emptyDir удалится навсегда. Хотя в общем случае падение POD'а не вызывает удаления Volume'а.

Задание:

- 1) создайте под в приложении
- 2) удалите deployment для mongo
- 3) Создайте его заново

Вместо того, чтобы хранить данные локально на ноде, имеет смысл подключить удаленное хранилище. В нашем случае можем использовать Volume gcePersistentDisk, который будет складывать данные в хранилище GCE.

Volume

Создадим диск в Google Cloud (ссылка на gist)

```
$ gcloud compute disks create --size=25GB --zone=us-central1-a reddit-mongo-disk
```

Добавим новый Volume POD-у базы.

mongo-deployment.yml (ссылка на gist)

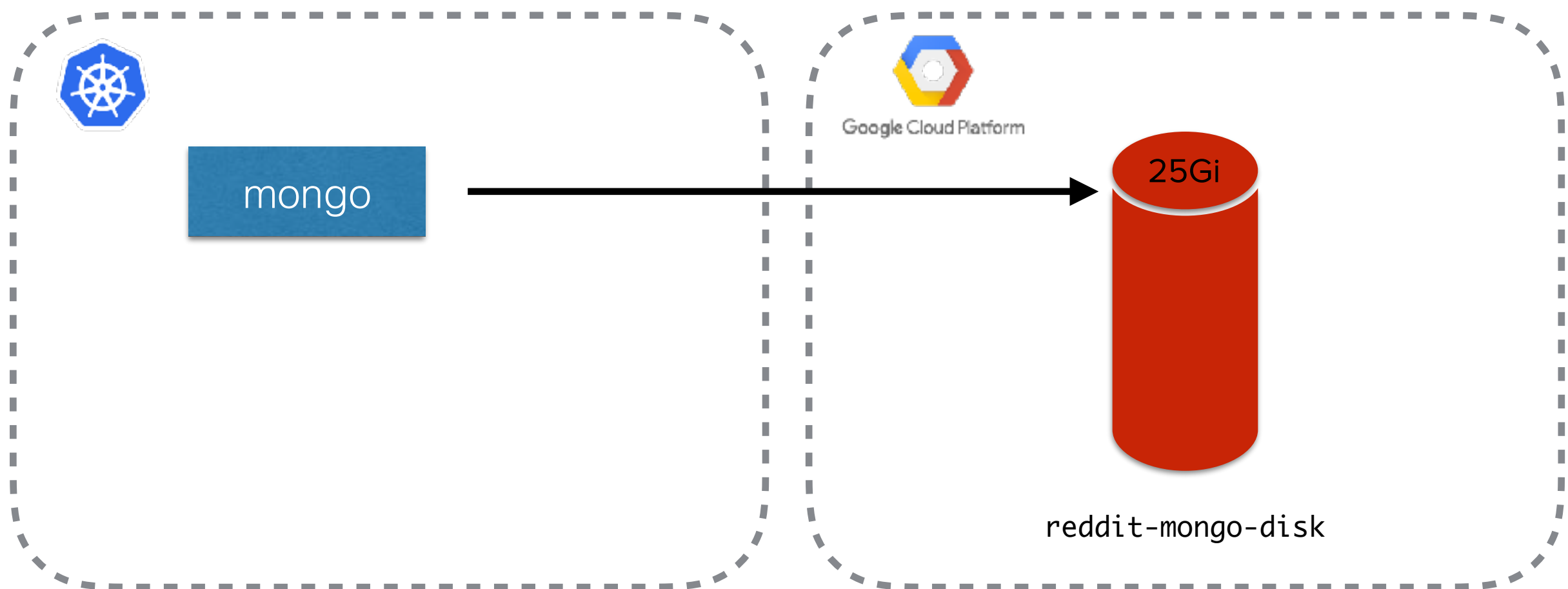
```
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: mongo
...

- image: mongo:3.2
  name: mongo
  volumeMounts:
    - name: mongo-gce-pd-storage
      mountPath: /data/db
  volumes:
    - name: mongo-gce-pd-storage
      gcePersistentDisk:
        pdName: reddit-mongo-disk
        fsType: ext4
```

Меняем Volume на другой тип

Volume

Монтируем выделенный диск к POD'у mongo



Volume

```
$ kubectl apply -f mongo-deployment.yml -n dev
```

Дождитесь, пересоздания Pod'a (занимает до 10 минут).
Зайдем в приложение и добавим пост

Microservices Reddit in dev ui-76ffdbbf6b-c4ftx container



5



<https://cloud.google.com/kubernetes-engine/docs/tutorials/http-balancer>

[Go to the link](#)

Удалим deployment

```
$ kubectl delete deploy mongo -n dev
```

Volume

Снова создадим деплой mongo.

```
$ kubectl apply -f mongo-deployment.yml -n dev
```

Microservices Reddit in dev ui-76ffdbbf6b-7zxsg container



5



<https://cloud.google.com/kubernetes-engine/docs/tutorials/http-balancer>

[Go to the link](#)

наш пост все еще на месте

Здесь можно посмотреть на созданный диск и увидеть какой машиной он используется

PersistentVolume

Используемый механизм Volume-ов можно сделать удобнее. Мы можем использовать не целый выделенный диск для каждого пода, а целый ресурс хранилища, общий для всего кластера.

Тогда при запуске Stateful-задач в кластере, мы сможем запросить хранилище в виде такого же ресурса, как CPU или оперативная память.

Для этого будем использовать механизм **PersistentVolume**.

PersistentVolume

Создадим описание PersistentVolume
mongo-volume.yml ([ссылка на gist](#))

```
-----
apiVersion: v1
kind: PersistentVolume
metadata:
  name: reddit-mongo-disk
spec:
  capacity:
    storage: 25Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  gcePersistentDisk:
    fsType: "ext4"
    pdName: "reddit-mongo-disk"
```

Имя PersistentVolume'a

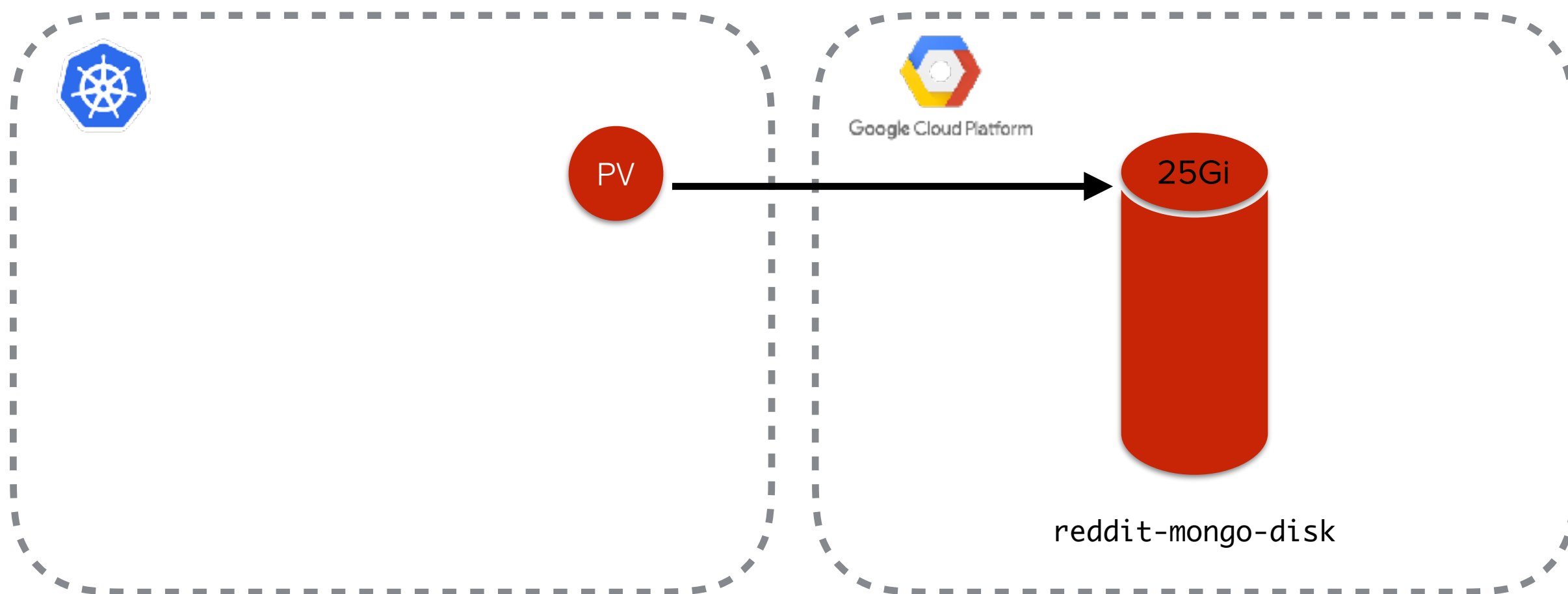
Имя диска в GCE

Добавим PersistentVolume в кластер

```
$ kubectl apply -f mongo-volume.yml -n dev
```

PersistentVolume

Мы создали PersistentVolume в виде диска в GCP.



PersistentVolumeClaim

Мы создали ресурс дискового хранилища, распространённый на весь кластер, в виде PersistentVolume.

Чтобы выделить приложению часть такого ресурса - нужно создать запрос на выдачу - **PersistentVolumeClaim**. Claim - это именно запрос, а не само хранилище.

С помощью запроса можно выделить место как из конкретного **PersistentVolume** (тогда параметры accessModes и StorageClass должны соответствовать, а места должно хватать), так и просто создать отдельный PersistentVolume под конкретный запрос.

PersistentVolumeClaim

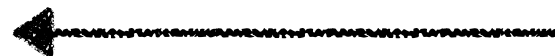
Создадим описание PersistentVolumeClaim (PVC)
mongo-claim.yml ([ссылка на gist](#))

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mongo-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 15Gi
```

Имя PersistentVolumeClaim'a



accessMode у PVC и у PV должен совпадать

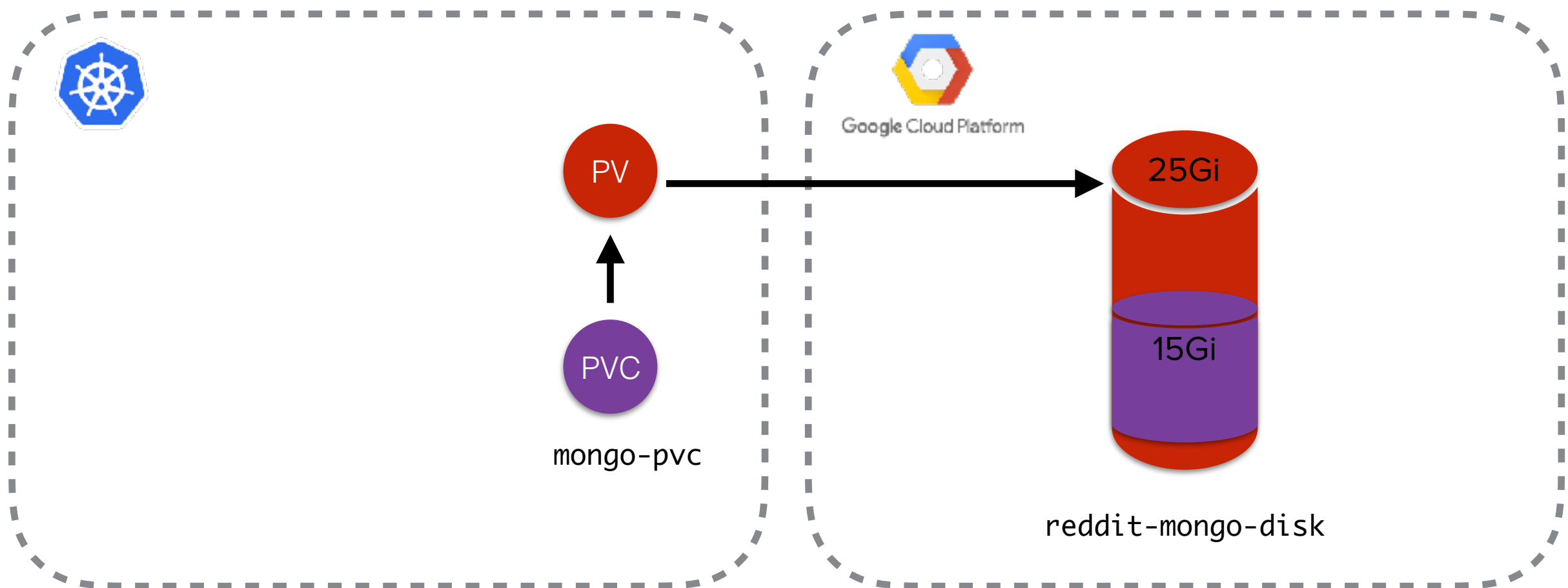


Добавим PersistentVolumeClaim в кластер

```
$ kubectl apply -f mongo-claim.yml -n dev
```

PersistentVolume

Мы выделили место в PV по запросу для нашей базы.
Одновременно использовать один PV можно только по **одному** Claim'у



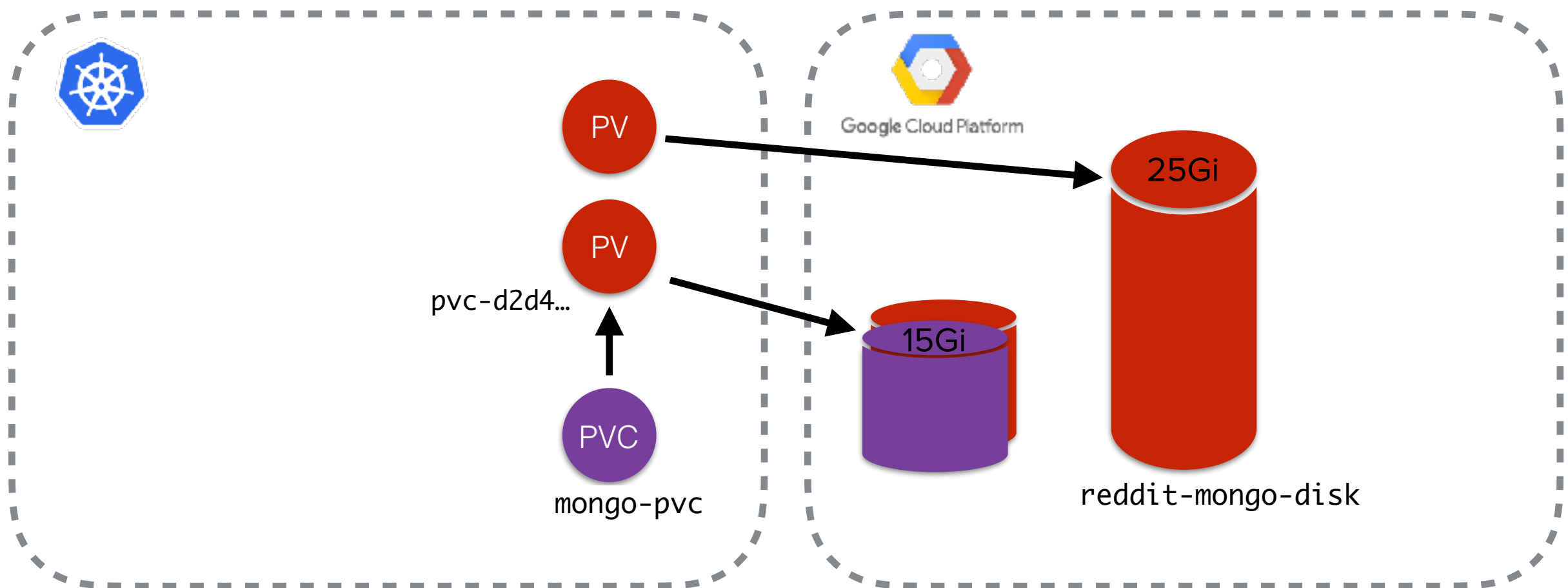
PersistentVolumeClaim

Если Claim не найдет по заданным параметрам PV внутри кластера, либо тот будет занят другим Claim'ом то он сам создаст нужный ему PV воспользовавшись стандартным StorageClass.

```
% kubectl describe storageclass standard -n dev
Name:                standard
IsDefaultClass:      Yes
Annotations:         storageclass.beta.kubernetes.io/is-default-class=true
Provisioner:         kubernetes.io/gce-pd
Parameters:          type=pd-standard
Events:              <none>
```

PersistentVolumeClaim

В нашем случае это обычный медленный Google Cloud Persistent Drive



Подключение PVC

Подключим PVC к нашим Pod'ам
mongo-deployment.yml ([ссылка на gist](#))

```
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: mongo
...
spec:
  containers:
  - image: mongo:3.2
    name: mongo
    volumeMounts:
    - name: mongo-persistent-storage
      mountPath: /data/db
  volumes:
  - name: mongo-persistent-storage
    persistentVolumeClaim:
      claimName: mongo-pvc
```

Имя PersistentVolumeClaim'a

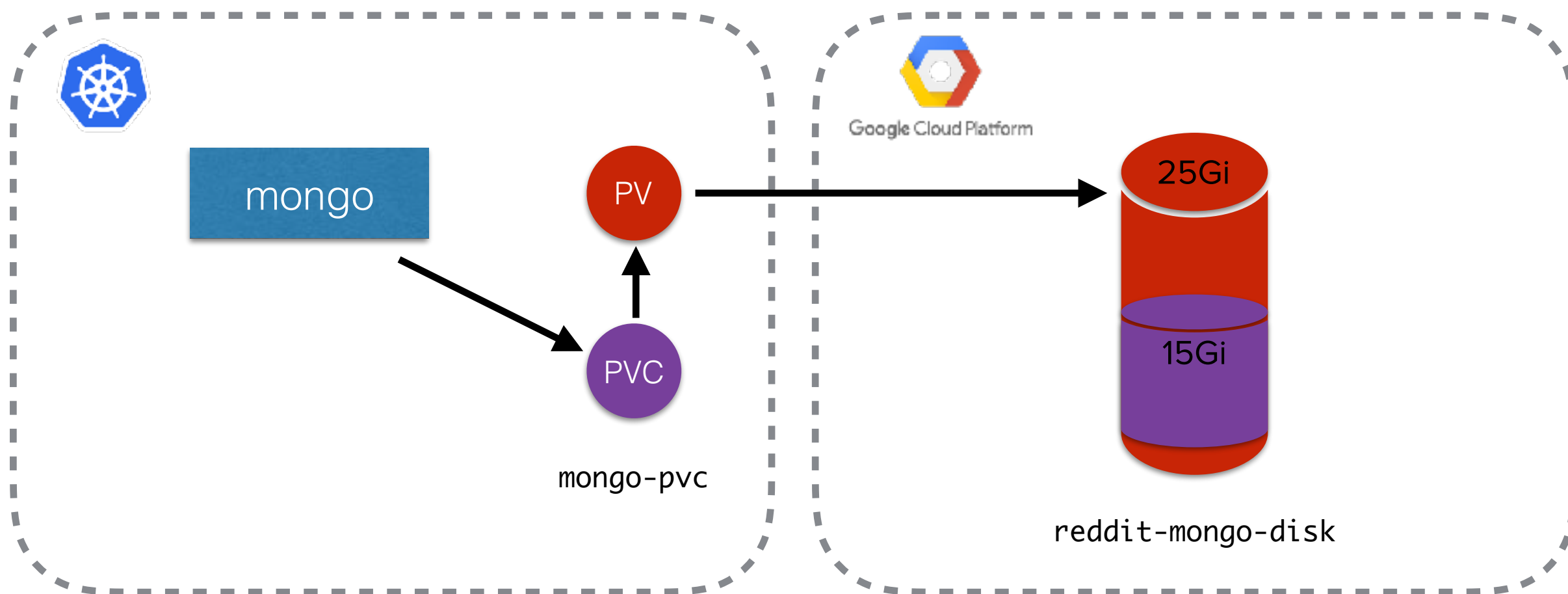


Обновим описание нашего Deployment'a

```
$ kubectl apply -f mongo-deployment.yml -n dev
```

PersistentVolume

Монтируем выделенное по PVC хранилище к POD'у mongo



Динамическое выделение Volume'ов

Создав PersistentVolume мы отделили объект "хранилища" от наших Service'ов и Pod'ов. Теперь мы можем его при необходимости переиспользовать.

Но нам гораздо интереснее создавать хранилища при необходимости и в автоматическом режиме. В этом нам помогут **StorageClass**'ы. Они описывают где (какой провайдер) и какие хранилища создаются.

В нашем случае создадим StorageClass **Fast** так, чтобы монтировались SSD-диски для работы нашего хранилища.

StorageClass

Создадим описание StorageClass'a
storage-fast.yml ([ссылка на gist](#))

```
---
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```

Имя StorageClass'a

Провайдер хранилища

Тип предоставляемого хранилища

Добавим StorageClass в кластер


```
$ kubectl apply -f storage-fast.yml -n dev
```


PVC + StorageClass

Создадим описание PersistentVolumeClaim **mongo-claim-dynamic.yml** ([ссылка на gist](#))

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mongo-pvc-dynamic
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: fast
  resources:
    requests:
      storage: 10Gi
```

Вместо ссылки на
созданный диск, теперь мы
ссылаемся на StorageClass



Добавим StorageClass в кластер

```
$ kubectl apply -f mongo-claim-dynamic.yml -n dev
```

Подключение динамического PVC

Подключим PVC к нашим Pod'ам
mongo-deployment.yml ([ссылка на gist](#))

```
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: mongo
...
spec:
  containers:
  - image: mongo:3.2
    name: mongo
    volumeMounts:
    - name: mongo-persistent-storage
      mountPath: /data/db
  volumes:
  - name: mongo-persistent-storage
    persistentVolumeClaim:
      claimName: mongo-pvc-dynamic
```

Обновим
PersistentVolumeClaim



Обновим описание нашего Deployment'a

```
$ kubectl apply -f mongo-deployment.yml -n dev
```

Подключение динамического PVC

Давайте посмотрит какие в итоге у нас получились PersistentVolume'ы

```
$ kubectl get persistentvolume -n dev
```

NAME	...	STATUS	CLAIM	STORAGE-CLASS
pvc-2884e35b-d5ee-11e7-8e22-42010a840111	...	Bound	dev/mongo-pvc-dynamic	fast
pvc-cc999fe0-d5df-11e7-8e22-42010a840111	...	Bound	dev/mongo-pvc	standard
reddit-mongo-disk	...	Available		

Статус PV по отношению к Pod'ам и Claim'ам

К какому Claim'у привязан данный PV

StorageClass данного PV

На созданные Kubernetes'ом диски можно посмотреть в web console

Подключение динамического PVC

