Kubernetes. Запуск кластера и приложения. Модель безопасности.

Подготовка

Создайте новую ветку в репозитории **Microservices** для выполнения данного ДЗ. Так как это второе задание по Kubernetes, то назовите ее **kubernetes-2**

Проверка данного ДЗ будет производиться через Pull Request (PR назовите **kubernetes-2**) ветки с ДЗ к ветке мастер и добавление в него kubernetes конфигурации

Добавьте Labels **Kubernetes** и **kubernetes-2** к вашему Pull Request

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

План

- Развернуть локальное окружение для работы с Kubernetes
- Развернуть Kubernetes в GKE
- Запустить reddit в Kubernetes

Разворачиваем Kubernetes локально

Для дальнейшей работы нам нужно подготовить локальное окружение, которое будет состоять из:

1) **kubectl** - фактически, главной утилиты для работы с Kubernetes API (все, что делает kubectl, можно сделать с помощью HTTP-запросов к API k8s)

2) Директории **~/.kube** - содержит служебную инфу для kubectl (конфиги, кеши, схемы API)

3) **minikube** - утилиты для разворачивания локальной инсталляции Kubernetes.

Kubectl

Необходимо установить **kubectl**:

Все способы установки доступны по ссылке

Установка Minikube

Для работы Minukube вам понадобится локальный гипервизор:

1. Для OS X: или xhyve driver, или VirtualBox, или VMware Fusion.

2. Для Linux: VirtualBox или KVM.

3. Для Windows: VirtualBox или Hyper-V.

Minikube

Инструкция по установке Minikube для разных ОС:
https://kubernetes.io/docs/tasks/tools/install-minikube/

Minikube

Запустим наш Minukube-кластер.

```
$ minikube start

Starting local Kubernetes v1.10.0 cluster...
Starting VM...
Downloading Minikube ISO
 140.01 MB / 140.01 MB [============================================] 100.00% 0s
Getting VM IP address...
Moving files into cluster...
Downloading localkube binary
 148.56 MB / 148.56 MB [============================================] 100.00% 0s
Setting up certs...
Connecting to cluster...
Setting up kubeconfig...
Starting cluster components...
Kubectl is now configured to use the cluster.
```

P.S. Если нужна конкретная версия kubernetes, указывайте флаг
--kubernetes-version <version> (v1.8.0)
P.P.S.По-умолчанию используется VirtualBox. Если у вас другой гипервизор, то ставьте флаг
--vm-driver=<hypervisor>

Kubectl

Наш Minikube-кластер развернут. При этом автоматически был настроен конфиг kubectl.

Проверим, что это так:

```
$ kubectl get nodes

NAME       STATUS    ROLES     AGE       VERSION
minikube   Ready     <none>    3h        v1.10.0
```

Kubectl

Конфигурация kubectl - это **контекст**.

Контекст - это комбинация:

1) **cluster** - API-сервер
2) **user** - пользователь для подключения к кластеру
3) **namespace** - область видимости (не обязательно, по-умолчанию default)

Информацию о контекстах kubectl сохраняет в файле `~/.kube/config`

Kubectl

Файл `~/.kube/config` - это такой же манифест kubernetes в YAML-формате (есть и Kind, и ApiVersion).

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /Users/chromko/.minikube/ca.crt
    server: https://192.168.99.100:8443
  name: minikube
contexts:
- context:
    cluster: minikube
    user: minikube
  name: minikube
current-context: minikube
kind: Config
preferences: {}
users:
- name: minikube
  user:
    as-user-extra: {}
    client-certificate: /Users/chromko/.minikube/client.crt
    client-key: /Users/chromko/.minikube/client.key
```

Список кластеров

Список контекстов

Список пользователей

Kubectl

Кластер (cluster) - это:

1) **server** - адрес kubernetes API-сервера
2) **certificate-authority** - корневой сертификат (которым подписан SSL-сертификат самого сервера), чтобы убедиться, что нас не обманывают и перед нами тот самый сервер

\+ **name** (Имя) для идентификации в конфиге

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /Users/chromko/.minikube/ca.crt
    server: https://192.168.99.100:8443
  name: minikube
```

Kubectl

Пользователь (**user**) - это:

1) Данные для аутентификации (зависит от того, как настроен сервер). Это могут быть:
   - username + password (Basic Auth
   - client key + client certificate
   - token
   - auth-provider config (например GCP)

\+ **name** (Имя) для идентификации в конфиге

```
users:
- name: minikube
  user:
    as-user-extra: {}
    client-certificate: /Users/chromko/.minikube/client.crt
    client-key: /Users/chromko/.minikube/client.key
```

Kubectl

Контекст (**контекст**) - это:

1) **cluster** - имя кластера из списка clusters
2) **user** - имя пользователя из списка users
3) **namespace** - область видимости по-умолчанию (не обязательно)

\+ **name** (Имя) для идентификации в конфиге

```
contexts:
- context:
    cluster: minikube
    user: minikube
  name: minikube
```

Kubectl

Обычно порядок конфигурирования kubectl следующий:

1) Создать cluster:

```
$ kubectl config set-cluster … cluster_name
```

2) Создать данные пользователя (credentials)

```
$ kubectl config set-credentials … user_name
```

3) Создать контекст

```
$ kubectl config set-context context_name \
--cluster=cluster_name \
--user=user_name
```

4) Использовать контекст

```
$ kubectl config use-context context_name
```

Kubectl

Таким образом kubectl конфигурируется для подключения к разным кластерам, под разными пользователями.

Текущий контекст можно увидеть так:

```
$ kubectl config current-context
minikube
```

Список всех контекстов можно увидеть так:

```
$ kubectl config get-contexts
CURRENT   NAME                          CLUSTER                       AUTHINFO    NAMESPACE
          kubernetes-the-hard-way       kubernetes-the-hard-way       admin
*         minikube                      minikube                      minikube
```

Запустим приложение

Для работы в приложения kubernetes, нам необходимо описать их желаемое состояние либо в YAML-манифестах, либо с помощью командной строки.

Всю конфигурацию поместите в каталог **./kubernetes/reddit** внутри вашего репозитория.

Deployment

Основные объекты - это ресурсы **Deployment**.

Как помним из предыдущего занятия, основные его задачи:

- Создание ReplicationSet (следит, чтобы число запущенных Pod-ов соответствовало описанному)
- Ведение истории версий запущенных Pod-ов (для различных стратегий деплоя, для возможностей отката)
- Описание процесса деплоя (стратегия, параметры стратегий)

ui-deployment.yml (ссылка на gist)

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: ui
  labels:
    app: reddit
    component: ui
spec:
  replicas: 3
  selector:
    matchLabels:
      app: reddit
      component: ui
  template:
    metadata:
      name: ui-pod
      labels:
        app: reddit
        component: ui
    spec:
      containers:
      - image: chromko/ui
        name: ui
```

Блок метаданных деплоя

Блок спецификации деплоя

Блок описания POD-ов

Не забудьте подставить свой образ

ui-deployment.yml

```
…
spec:
  replicas: 3
  selector:
    matchLabels:
      app: reddit
      component: ui
  template:
    metadata:
      name: ui-pod
      labels:
        app: reddit
        component: ui
    spec:
      containers:
      - image: chromko/ui
        name: ui
```

selector описывает, как ему отслеживать POD-ы.

В данном случае - контроллер будет считать POD-ы с метками:
app=reddit **И** component=ui.

Поэтому **важно** в описании POD-а задать нужные метки (labels)

P.S. Для более гибкой выборки вводим 2 метки (app и component).

Запустим в Minikube ui-компоненту.

```
$ kubectl apply -f ui-deployment.yml

deployment "ui" created
```

Убедитесь, что во 2,3,4 и 5 столбцах стоит число 3 (число реплик ui):

```
$ kubectl get deployment

NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
ui        3         3         3            3           1m
```

P.S. **kubectl apply -f <filename>** может принимать не только отдельный файл, но и папку с ними. Например:

```
$ kubectl apply -f ./kubernetes/reddit
```

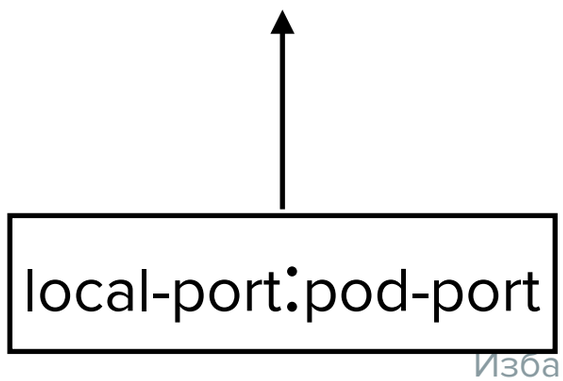

Пока что мы не можем использовать наше приложение полностью, потому что никак не настроена сеть для общения с ним.

Но **kubectl** умеет пробрасывать сетевые порты POD-ов на локальную машину

Найдем, используя selector, POD-ы приложения (ссылка на gist):

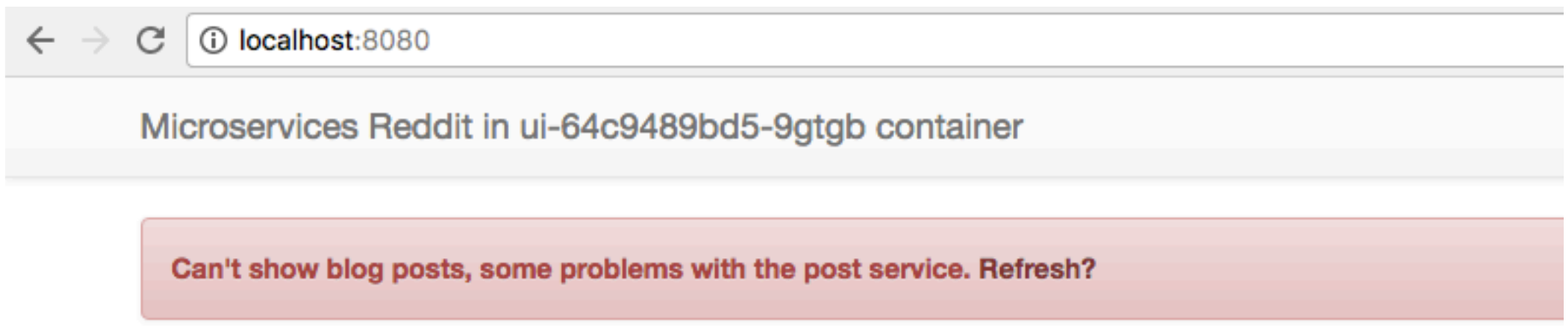
```
$ kubectl get pods --selector component=ui

$ kubectl port-forward <pod-name> 8080:9292
```



Зайдем в браузере на

http://localhost:8080



UI работает, подключим остальные компоненты

Comment

comment-deployment.yml (ссылка на gist)

```
---
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: comment
  labels:
    app: reddit
    component: comment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: reddit
      component: comment
  template:
    metadata:
      name: comment
      labels:
        app: reddit
        component: comment
    spec:
      containers:
      - image: chromko/comment
        name: comment
```

Компонент **comment** описывается похожим образом.
Меняется только имя образа и метки и применяем (kubectl apply)

Проверить можно так же, пробросив <local-port>: 9292 и зайдя на адрес

http://localhost:<local-port>/healthcheck

Задание

Deployment компонента **post** сконфигурируйте подобным же образом самостоятельно и проверьте его работу.

Не забудьте, что **post** слушает по-умолчанию на порту 5000

MongoDB

mongo-deployment.yml (ссылка на gist)

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: mongo
  labels:
    app: reddit
    component: mongo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: reddit
      component: mongo
  template:
    metadata:
      name: mongo
      labels:
        app: reddit
        component: mongo
   spec:
      containers:
      - image: mongo:3.2
        name: mongo
```

Разместим базу данных

Все похоже, но меняются только образы и значения label-ов

MongoDB

Также примонтируем стандартный Volume для хранения данных вне контейнера

`mongo-deployment.yml` (ссылка на gist)

```
apiVersion: apps/v1beta2
kind: Deployment
…
      containers:
      - image: mongo:3.2
        name: mongo
        volumeMounts:
        - name: mongo-persistent-storage
          mountPath: /data/db
      volumes:
      - name: mongo-persistent-storage
        emptyDir: {}
```

Точка монтирования в контейнере (не в POD-е)

Ассоциированные с POD-ом Volume-ы

Services

В текущем состоянии приложение не будет работать, так его компоненты ещё не знают как найти друг друга

Для связи компонент между собой и с внешним миром используется объект **Service** - абстракция, которая определяет набор POD-ов (Endpoints) и способ доступа к ним

Services

Для связи ui с post и comment нужно создать им по объекту Service.

`comment-service.yml` (ссылка на gist)

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

Когда объект service будет создан:

1) В DNS появится запись для comment
2) При обращении на адрес post:9292 изнутри любого из POD-ов **текущего namespace** нас переправит на 9292-ный порт одного из POD-ов приложения post, выбранных по label-ам

Services

По label-ам должны были быть найдены соответствующие POD-ы. Посмотреть можно с помощью:

```
$ kubectl describe service comment | grep Endpoints
Endpoints:         172.17.0.9:5000
```

А изнутри любого POD-а должно разрешаться:

```
$ kubectl exec -ti <pod-name> nslookup comment

nslookup: can't resolve '(null)': Name does not resolve

Name:      post
Address 1: 10.0.0.162 comment.default.svc.cluster.local
```

Задание

По аналогии создайте объект Service в файле **post-service.yml** для компонента **post** (не забудьте про label-ы и правильные tcp-порты).

Services

Post и Comment также используют mongodb, следовательно ей тоже нужен объект Service.

mongodb-service.yml (ссылка на gist)

```
---
apiVersion: v1
kind: Service
metadata:
  name: mongodb
  labels:
    app: reddit
    component: mongo
spec:
  ports:
  - port: 27017
    protocol: TCP
    targetPort: 27017
  selector:
    app: reddit
    component: mongo
```

По-сути все очень похоже, деплоим:

```
$ kubectl apply -f mongodb-service.yml
```

Services

Проверяем:

пробрасываем порт на **ui** pod

```
$ kubectl port-forward <pod-name> 9292:9292
```

Заходим на http://localhost:9292

Services

Посмотрим в логи, например, comment:

Подставьте свой POD

```
$ kubectl logs post-56bbbf6795-7btnm
D, [2017-11-23T11:58:14.036381 #1] DEBUG -- : MONGODB | Topology type 'unknown' initializing.
D, [2017-11-23T11:58:14.036584 #1] DEBUG -- : MONGODB | Server comment_db:27017 initializing.
D, [2017-11-23T11:58:14.041398 #1] DEBUG -- : MONGODB | getaddrinfo: Name does not resolve
D, [2017-11-23T11:58:14.090421 #1] DEBUG -- : MONGODB | getaddrinfo: Name does not resolve
```

Services

Приложение ищет совсем другой адрес: **comment_db**, а не **mongodb**
Аналогично и сервис comment ищет **post_db**.

Эти адреса заданы в их Dockerfile-ах в виде переменных окружения:

post/Dockerfile

```
…
ENV POST_DATABASE_HOST=post_db
```

comment/Dockerfile

```
…
ENV COMMENT_DATABASE_HOST=comment_db
```

Services

В Docker Swarm проблема доступа к одному ресурсу под разными именами решалась с помощью сетевых алиасов.

В Kubernetes такого функционала нет.
Мы эту проблему можем решить с помощью тех же Service-ов.

Services

Сделаем Service для БД comment.

comment-mongodb-service.yml (ссылка на gist)

```
---
apiVersion: v1
kind: Service
metadata:
  name: comment-db
  labels:
    app: reddit
    component: mongo
    comment-db: "true"
spec:
  ports:
  - port: 27017
    protocol: TCP
    targetPort: 27017
  selector:
    app: reddit
    component: mongo
    comment-db: "true"
```

В имени нельзя использовать “_”

добавим метку, чтобы различать сервисы

Отдельный лейбл для comment-db

P.S. булевые значения обязательно указывать в кавычках

Services

Так же придется обновить файл deployment для mongodb, чтобы новый Service смог найти нужный POD

mongo-deployment.yml (ссылка на gist)

```
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: mongo
  labels:
    app: reddit
    component: mongo
    comment-db: "true"
spec:
  replicas: 1
  selector:
    matchLabels:
      app: reddit
      component: mongo
  template:
    metadata:
      name: mongo
      labels:
        app: reddit
        component: mongo
        comment-db: "true"
...
```

Лейбл в deployment чтобы было понятно, что развернуто

label в pod, который нужно найти

Services

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: comment
…
      containers:
      - image: chromko/comment
        name: comment
        env:
        - name: COMMENT_DATABASE_HOST
          value: comment-db
```

Зададим pod-ам comment переменную окружения для обращения к базе (см слайд 34) (ссылка на gist)

Service

Мы сделали базу доступной для comment.

Проделайте аналогичные же действия для post-сервиса. Название сервиса должно post-db.

После этого снова сделайте **port-forwarding** на **UI** и убедитесь, что приложение запустилось без ошибок и посты создаются

Services

Создадим все новые объекты с помощью

```
$ kubectl apply -f …
```

Проверим логи **post** снова

```
D, [2017-11-23T13:00:40.110693 #1] DEBUG -- : MONGODB | comment-db:27017 | admin.listDatabases | STARTED |
{"listDatabases"=>1}
D, [2017-11-23T13:00:40.113945 #1] DEBUG -- : MONGODB | comment-db:27017 | admin.listDatabases | SUCCEEDED |
0.002974432s
```

Удалите объект mongodb-service

```
$ kubectl delete -f mongodb-service.yml
```

Или

```
$ kubectl delete service mongodb
```

Service

Нам нужно как-то обеспечить доступ к ui-сервису снаружи.
Для этого нам понадобится Service для UI-компоненты

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
```

Главное отличие - тип сервиса **NodePort.**

Service

По-умолчанию все сервисы имеют тип **ClusterIP** - это значит, что сервис располагается на внутреннем диапазоне IP-адресов кластера. Снаружи до него нет доступа.

Тип **NodePort** - на каждой ноде кластера открывает порт из диапазона **30000-32767** и переправляет трафик с этого порта на тот, который указан в **targetPort** Pod (похоже на стандартный expose в docker)

Теперь до сервиса можно дойти по <Node-IP>:<NodePort>
Также можно указать самим NodePort (но все равно **из диапазона**):

```
spec:
  type: NodePort
  ports:
  - nodePort: 32092
    port: 9292
    protocol: TCP
    targetPort: 9292
  selector:
…
```

Service

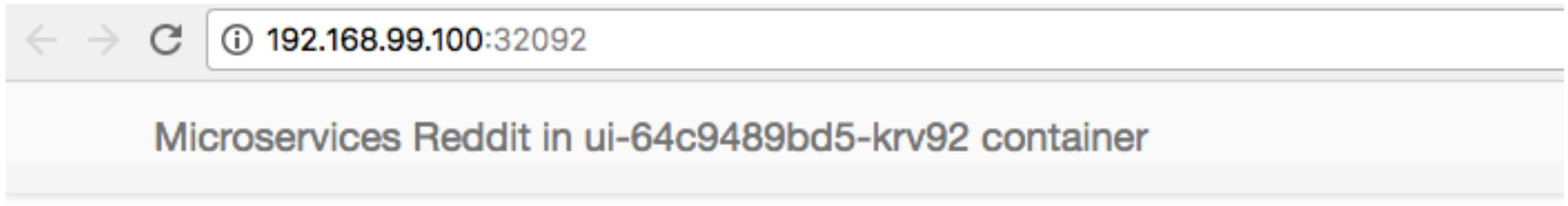
T.e. в описании service

NodePort - для доступа снаружи кластера
port - для доступа к сервису изнутри кластера

Minikube

Minikube может выдавать web-странцы с сервисами которые были помечены типом **NodePort**
Попробуйте:

```
$ minikube service ui
```



Minikube

Minikube может перенаправлять на web-странцы с сервисами которые были помечены типом **NodePort**
Посмотрите на список сервисов:

```
$ minikube services list
|-------------|-----------------------|-------------------------------|
|  NAMESPACE  |         NAME          |              URL              |
|-------------|-----------------------|-------------------------------|
| default     | comment               | No node port                  |
| default     | comment-db            | No node port                  |
| default     | kubernetes            | No node port                  |
| default     | post                  | No node port                  |
| default     | post-db               | No node port                  |
| default     | ui                    | http://192.168.99.100:32092 |
| kube-system | kube-dns              | No node port                  |
|-------------|-----------------------|-------------------------------|
```

Minikube

Minikube также имеет в комплекте несколько стандартных аддонов (расширений) для Kubernetes (kube-dns, dashboard, monitoring,...). Каждое расширение - это такие же PODы и сервисы, какие создавались нами, только они еще общаются с API самого Kubernetes

Получить список расширений:

```
$ minikube addons list
- addon-manager: enabled
- coredns: disabled
- dashboard: enabled
- default-storageclass: enabled
- efk: disabled
- freshpod: disabled
- heapster: disabled
- ingress: disabled
- kube-dns: enabled
- metrics-server: disabled
- registry: disabled
- registry-creds: disabled
- storage-provisioner: enabled
```

Minikube

Интересный аддон - dashboard. Это UI для работы с kubernetes. По умолчанию в новых версиях он включен. Как и многие kubernetes add-on'ы, dashboard запускается в виде pod'a.

Если мы посмотрим на запущенные pod'ы с помощью команды **kubectl get pods**, то обнаружим только наше приложение.

Потому что поды и сервисы для dashboard-а были запущены в **namespace** (пространстве имен) **kube-system**.
Мы же запросили пространство имен **default**.

Namespaces

Namespace - это, по сути, виртуальный кластер Kubernetes внутри самого Kubernetes. Внутри каждого такого кластера находятся свои объекты (POD-ы, Service-ы, Deployment-ы и т.д.), кроме объектов, общих на все namespace-ы (nodes, ClusterRoles, PersistentVolumes)

В разных namespace-ах могут находится объекты с одинаковым именем, но в рамках одного namespace имена объектов должны быть уникальны.

Namespaces

При старте Kubernetes кластер уже имеет 3 namespace:

- **default** - для объектов для которых не определен другой Namespace (в нем мы работали все это время)
- **kube-system** - для объектов созданных Kubernetes'ом и для управления им
- **kube-public** - для объектов к которым нужен доступ из любой точки кластера

Для того, чтобы выбрать конкретное пространство имен, нужно указать флаг **-n** <namespace> или **--namespace** <namespace> при запуске kubectl

Namespace

Найдем же объекты нашего dashboard

```
$ kubectl get all -n kube-system --selector k8s-app=kubernetes-dashboard

NAME                                        READY     STATUS    RESTARTS   AGE
pod/kubernetes-dashboard-598d75cb96-vnntk   1/1       Running   0          1h

NAME                           TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes-dashboard   ClusterIP   10.55.244.13   <none>        443/TCP   1h

NAME                                   DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/kubernetes-dashboard   1         1         1            1           1h

NAME                                              DESIRED   CURRENT   READY     AGE
replicaset.apps/kubernetes-dashboard-598d75cb96   1         1         1         1h
```

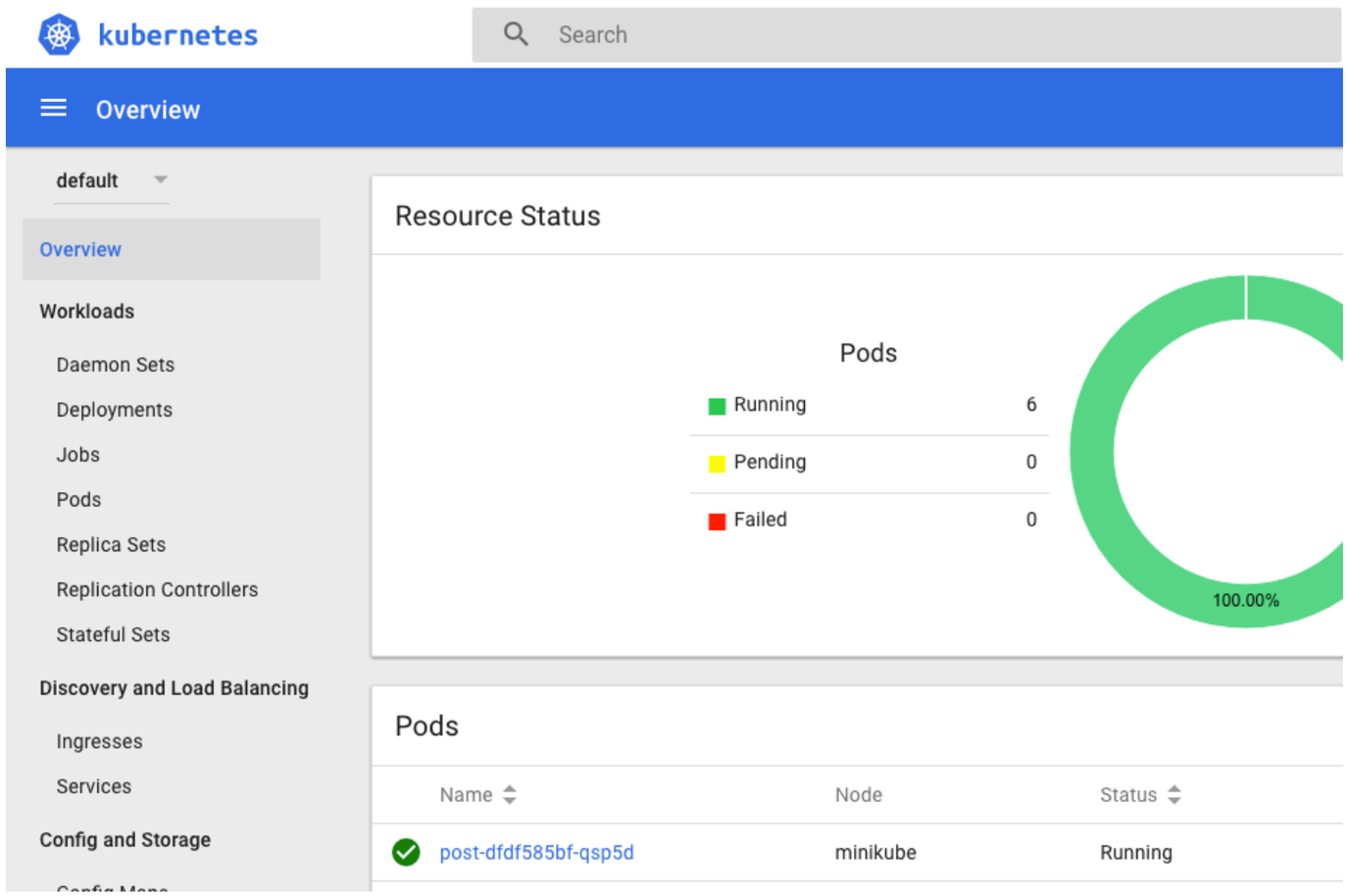
Мы вывели все объекты из неймспейса **kube-system**, имеющие label **app=kubernetes-dashboard**

Dashboard

Зайдем в Dashboard:

minikube тоже надо указывать namespace

```
$ minikube service kubernetes-dashboard -n kube-system
```



Dashboard

В самом Dashboard можно:

- отслеживать состояние кластера и рабочих нагрузок в нем
- создавать новые объекты (загружать YAML-файлы)
- Удалять и изменять объекты (кол-во реплик, yaml-файлы)
- отслеживать логи в Pod-ах
- при включении Heapster-аддона смотреть нагрузку на Pod-ах
- и т.д.

Ознакомьтесь, покликайте - в minikube не страшно ничего сломать (если что заново поднять).

Namespace

Используем же namespace в наших целях. Отделим среду для разработки приложения от всего остального кластера.
Для этого создадим свой Namespace **dev**

dev-namespace.yml(ссылка на gist)

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: dev
```

$ kubectl apply -f dev-namespace.yml

Namespace

Запустим приложение в dev неймспейсе:

```
$ kubectl apply -n dev -f …
```

Если возник конфликт портов у ui-service, то убираем из описания значение NodePort

Смотрим результат:

```
$ minikube service ui -n dev
```

Namespace

Давайте добавим инфу об окружении внутрь контейнера UI

`ui-deployment.yml` (ссылка на gist)

```
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: ui
…
   spec:
      containers:
      - image: chromko/ui
        name: ui
        env:
        - name: ENV
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
```

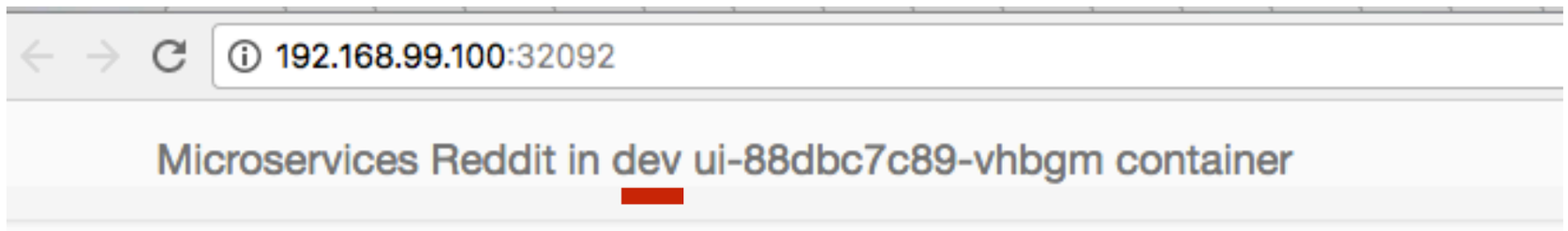
Извлекаем значения из контекста запуска

Подробнее здесь

Namespace

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

Смотрим результат:



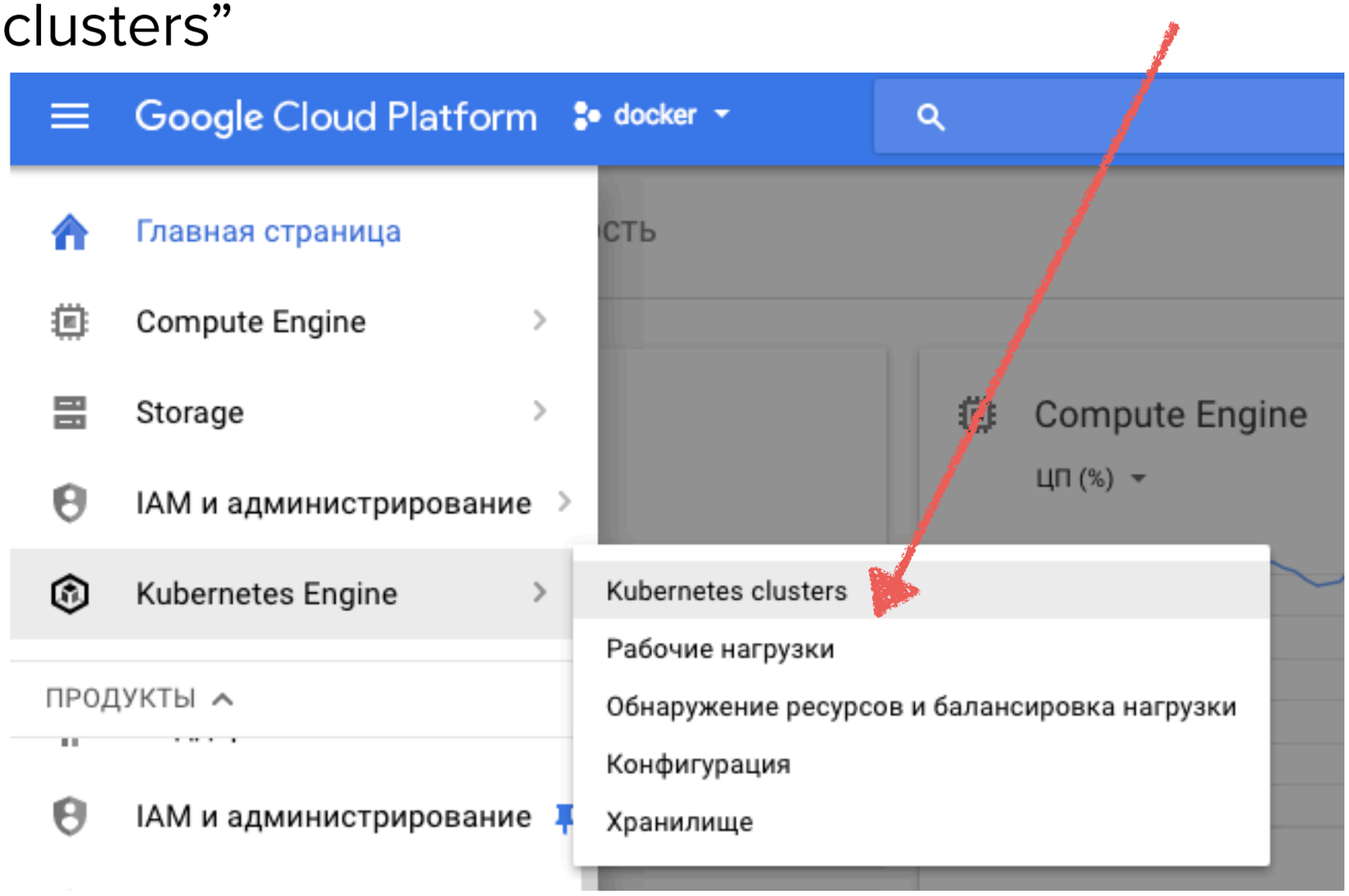
Разворачиваем Kubernetes

Мы подготовили наше приложение в локальном окружении. Теперь самое время запустить его на реальном кластере Kubernetes.

В качестве основной платформы будем использовать **Google Kubernetes Engine.**

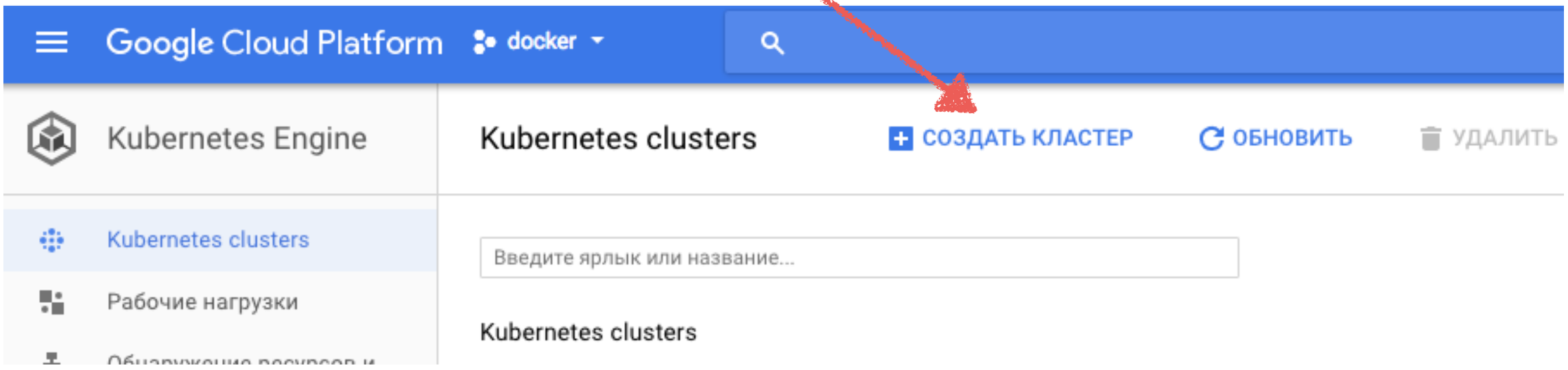
Разворачиваем Kubernetes

Зайдите в свою gcloud console, перейдите в “kubernetes clusters”



Разворачиваем Kubernetes

Нажмите “создать Cluster”



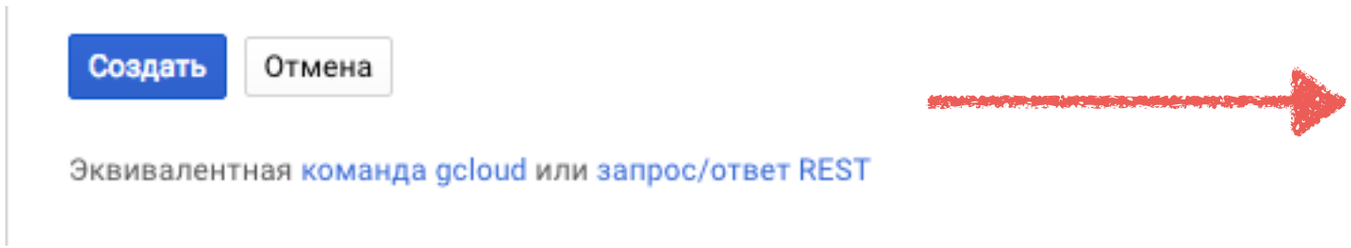
Разворачиваем Kubernetes

Укажите следующие настройки кластера:

- Тип машины - небольшая машина (1,7 ГБ) (для экономии ресурсов)
- Размер - 2
- Базовая аутентификация - отключена
- Устаревшие права доступа - отключено
- Панель управления Kubernetes - отключено
- Размер загрузочного диска - 20 ГБ (для экономии)

Разворачиваем Kubernetes

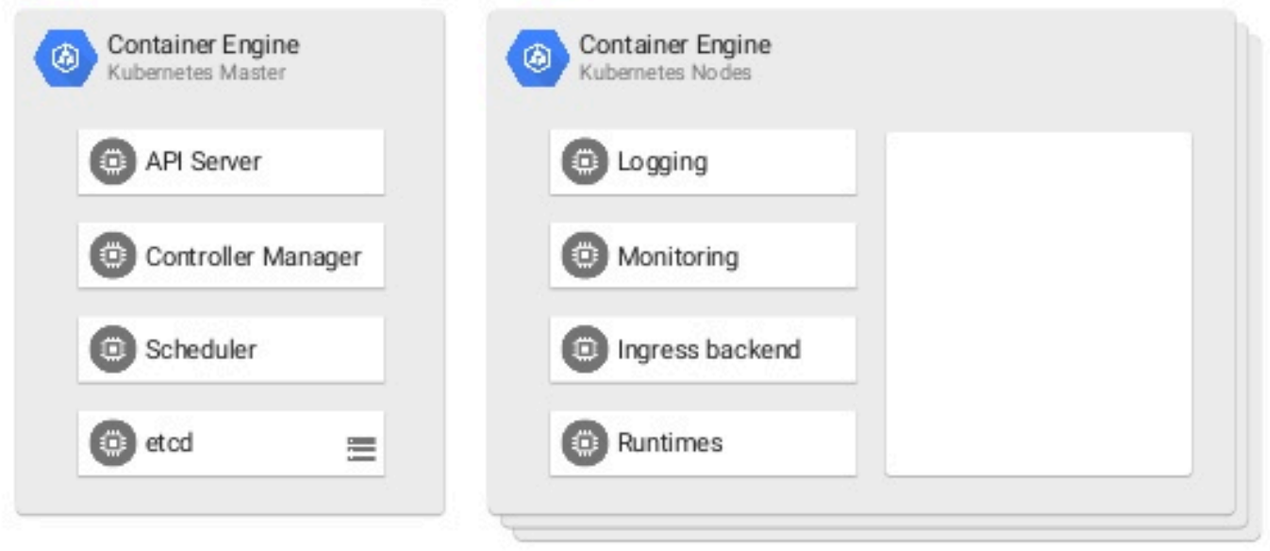
Жмем “Создать” и ждем, пока поднимется кластер



GKE

Компоненты управления кластером запускаются в container engine и управляются Google:

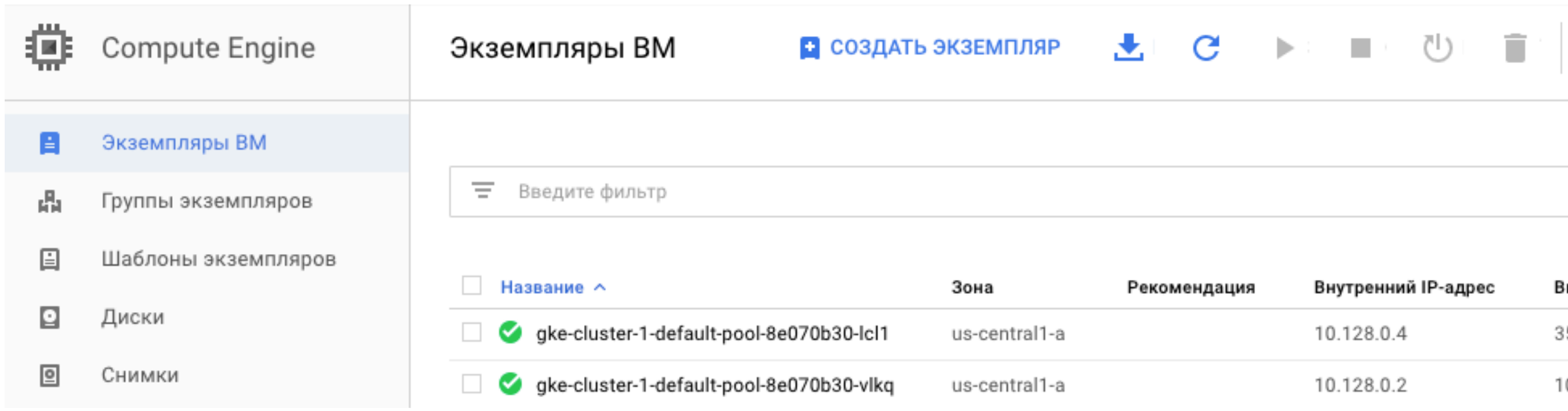
- kube-apiserver
- kube-scheduler
- kube-controller-manager
- etcd



Рабочая нагрузка (собственные POD-ы), аддоны, мониторинг, логирование и т.д. запускаются на **рабочих нодах**

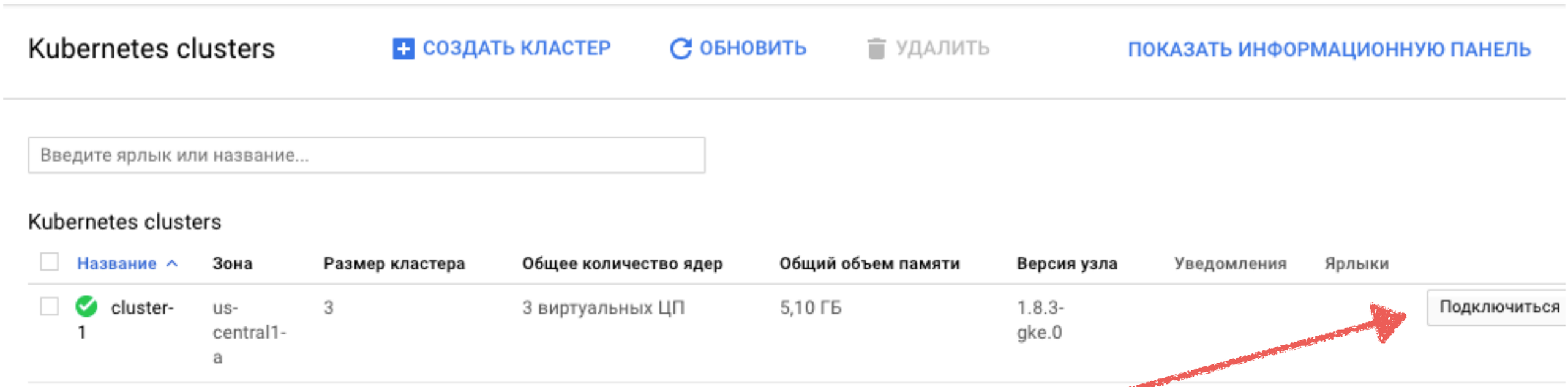
GKE

Рабочие ноды - стандартные ноды Google compute engine. Их можно увидеть в списке запущенных узлов.
На них всегда можно зайти по ssh
Их можно остановить и запустить.



GKE

Подключимся к GKE для запуска нашего приложения.



Нажмите и скопируйте команду вида:

```
$ gcloud container clusters get-credentials cluster-1 --zone
us-central1-a --project docker-182408
```

Введите в консоли скопированную команду.

В результате в файл `~/.kube/config` будут добавлены **user, cluster** и **context** для подключения к кластеру в GKE. Также текущий контекст будет выставлен для подключения к этому кластеру.
Убедиться можно, введя

```
$ kubectl config current-context
```

Запустим наше приложение в GKE

Создадим **dev** namespace

```
$ kubectl apply -f ./kubernetes/reddit/dev-namespace.yml
```

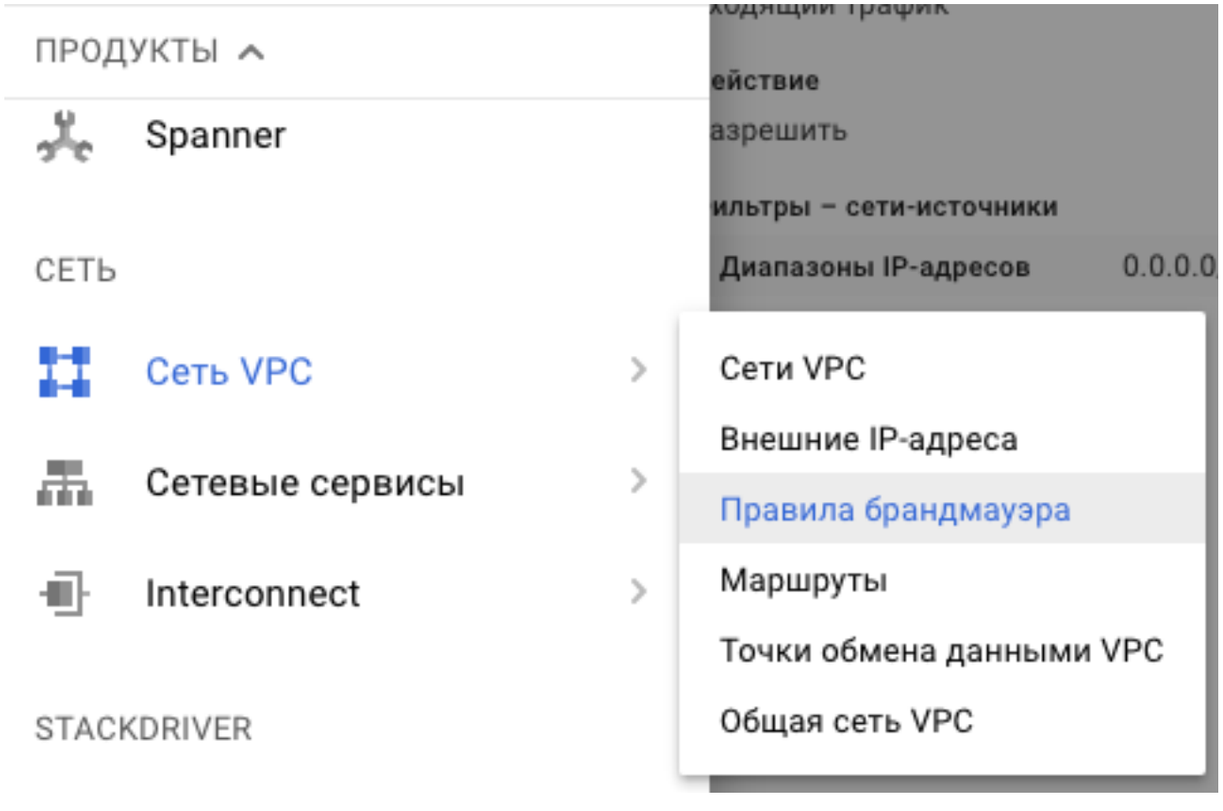
Задеплоим все компоненты приложения в namespace dev:

```
$ kubectl apply -f ./kubernetes/reddit/ -n dev
```

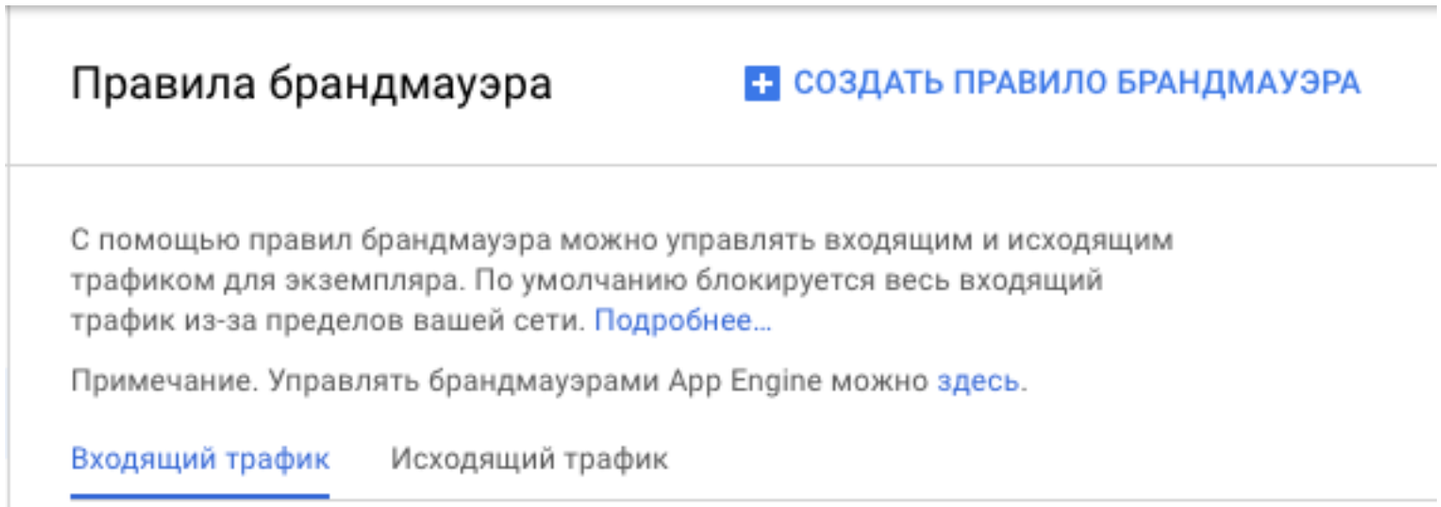

GKE

Откроем Reddit для внешнего мира:

Зайдите в “правила брандмауэра”



Нажмите “создать правило брандмауэра”

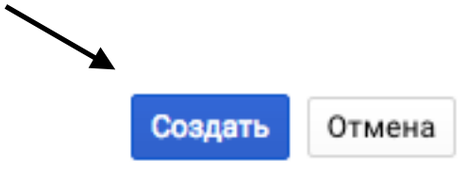


Откроем диапазон портов kubernetes для публикации сервисов

Настройте:

- Название - произвольно, но понятно
- Целевые экземпляры - все экземпляры в сети
- Диапазоны IP-адресов источников - 0.0.0.0/0
  Протоколы и порты - Указанные протоколы и порты
  tcp:**30000-32767**

Создать



GKE

Найдите внешний IP-адрес любой ноды из кластера либо в веб-консоли, либо **External IP** в выводе:

```
$ kubectl get nodes -o wide
```

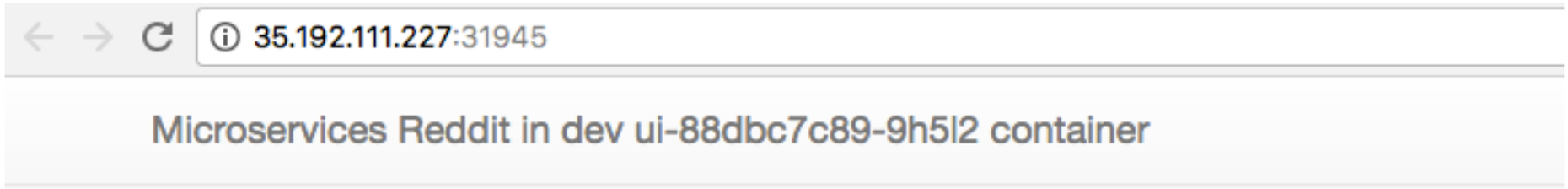
Найдите порт публикации сервиса ui

```
$ kubectl describe service ui  -n dev  | grep NodePort

    Type:                     NodePort
    NodePort:                 <unset>  31945/TCP
```

GKE

Идем по адресу `http://<node-ip>:<NodePort>`



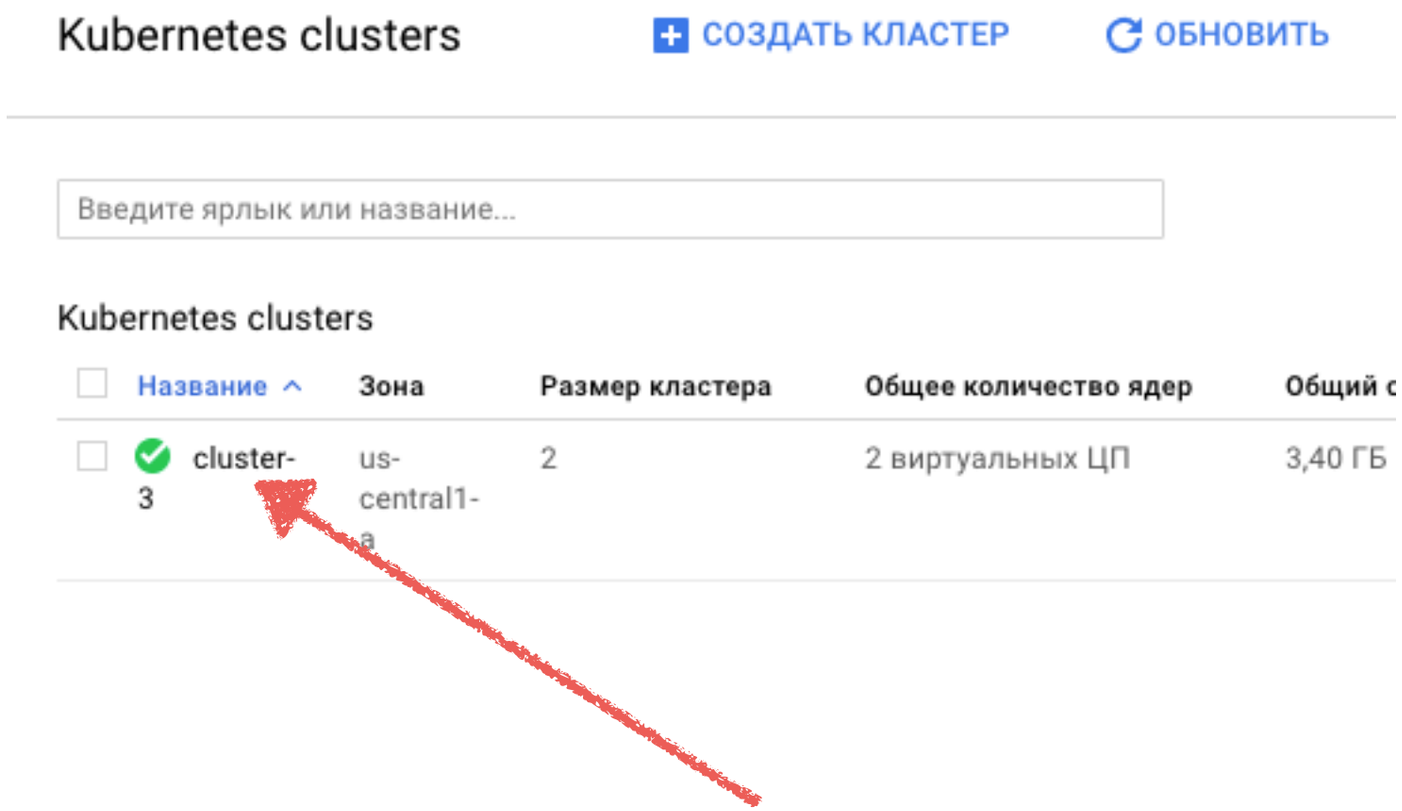
Задание

В PR должны присутствовать конфигурация для развертывания Reddit приложения в kubernetes (deployments, services, namespace) в отдельной директории.

К PR приложить **скриншот** веб-интерфейса приложения в GKE (по-желанию) **или ссылку** на него.

GKE

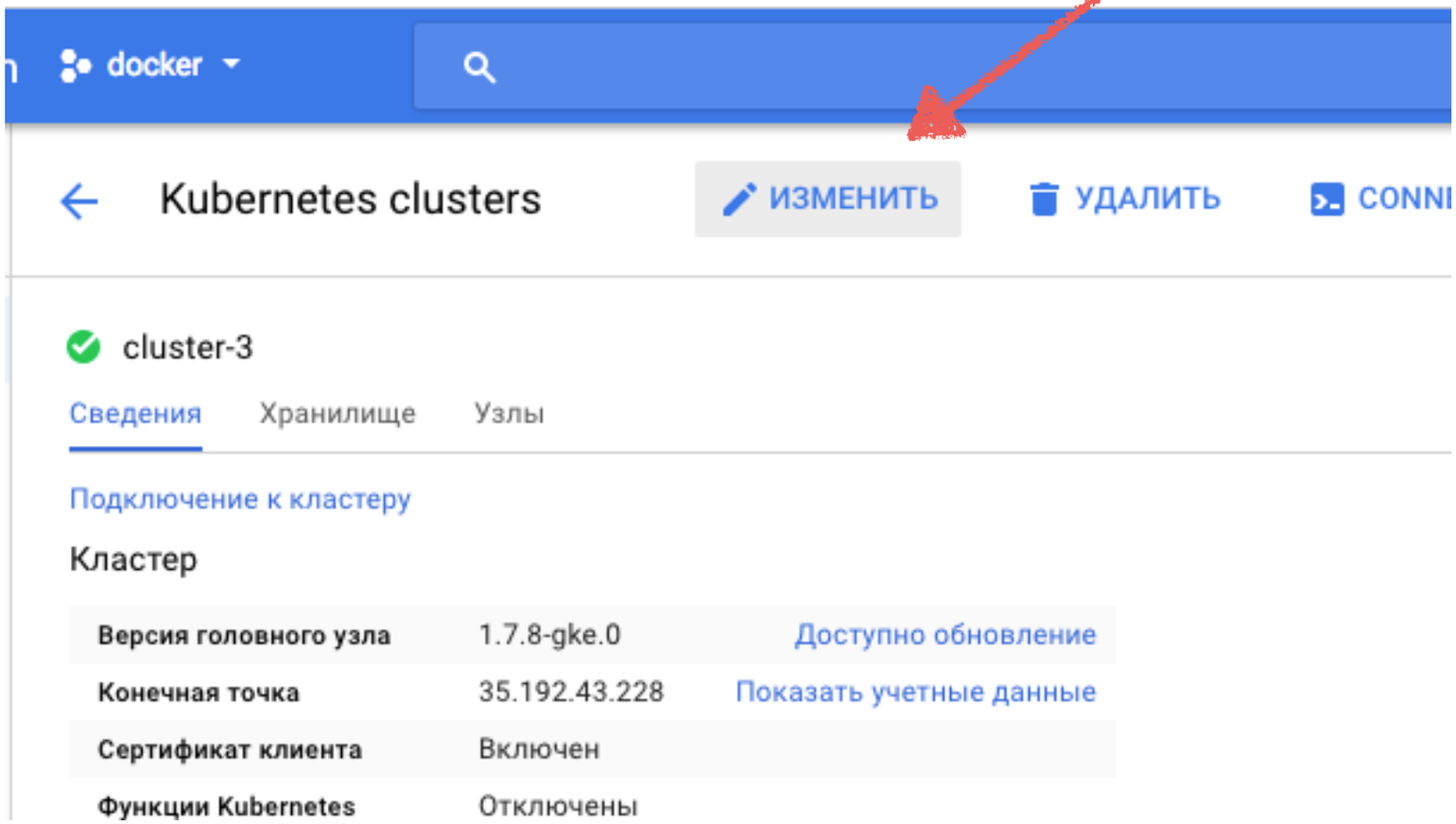
В GKE также можно запустить Dashboard для кластера.



Нажмите на имя кластера

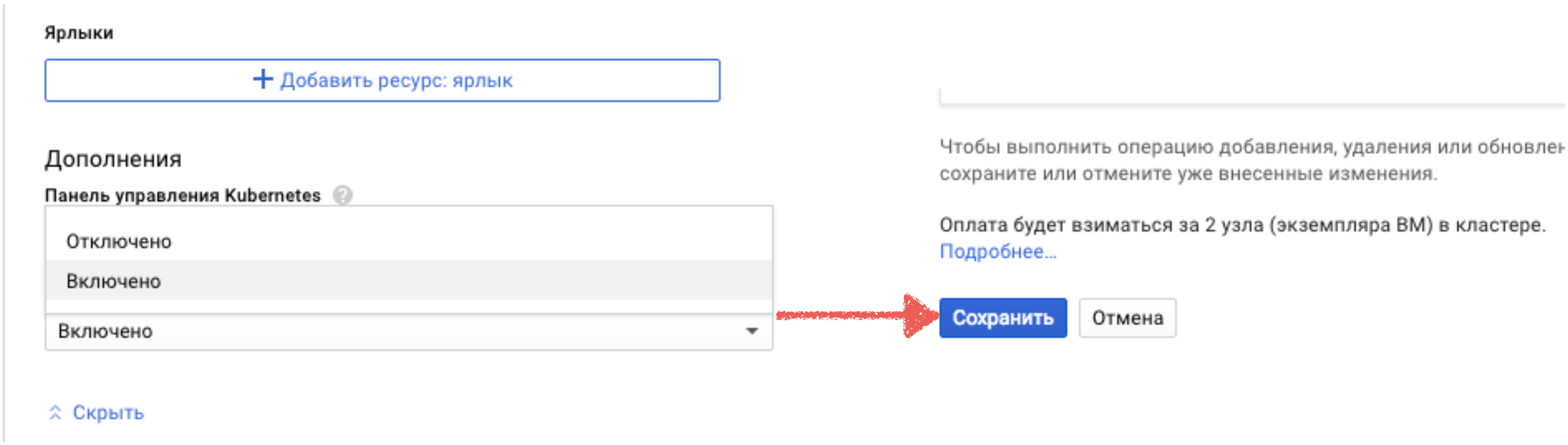
GKE

Изменить



GKE

В этом меню можно поменять конфигурацию кластера. Нам нужно включить дополнение
“Панель управления Kubernetes”



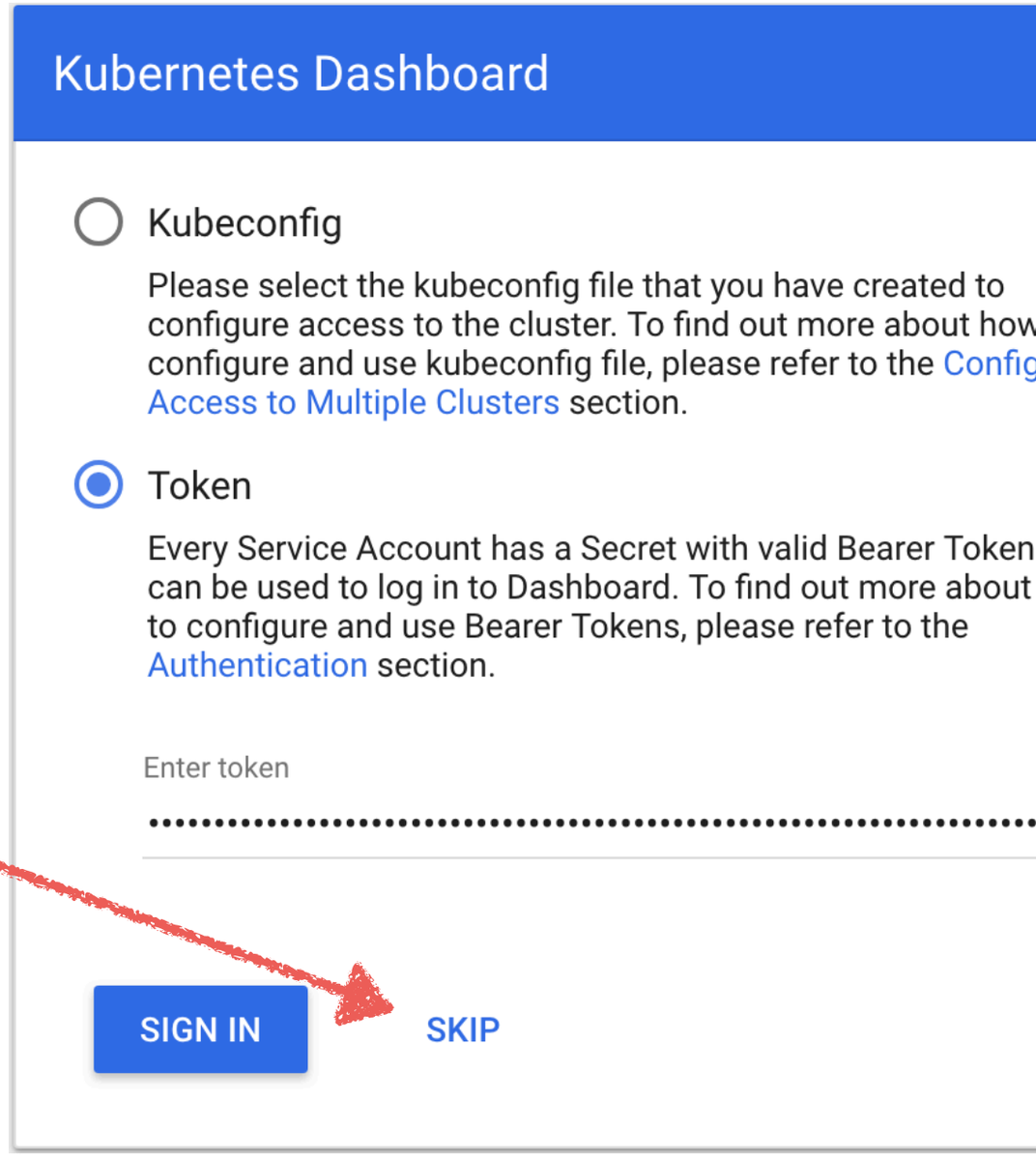
Ждем пока кластер загрузится

GKE

```
$ kubectl proxy
```

Заходим по адресу http://localhost:8001/ui

Нажимаем на SKIP

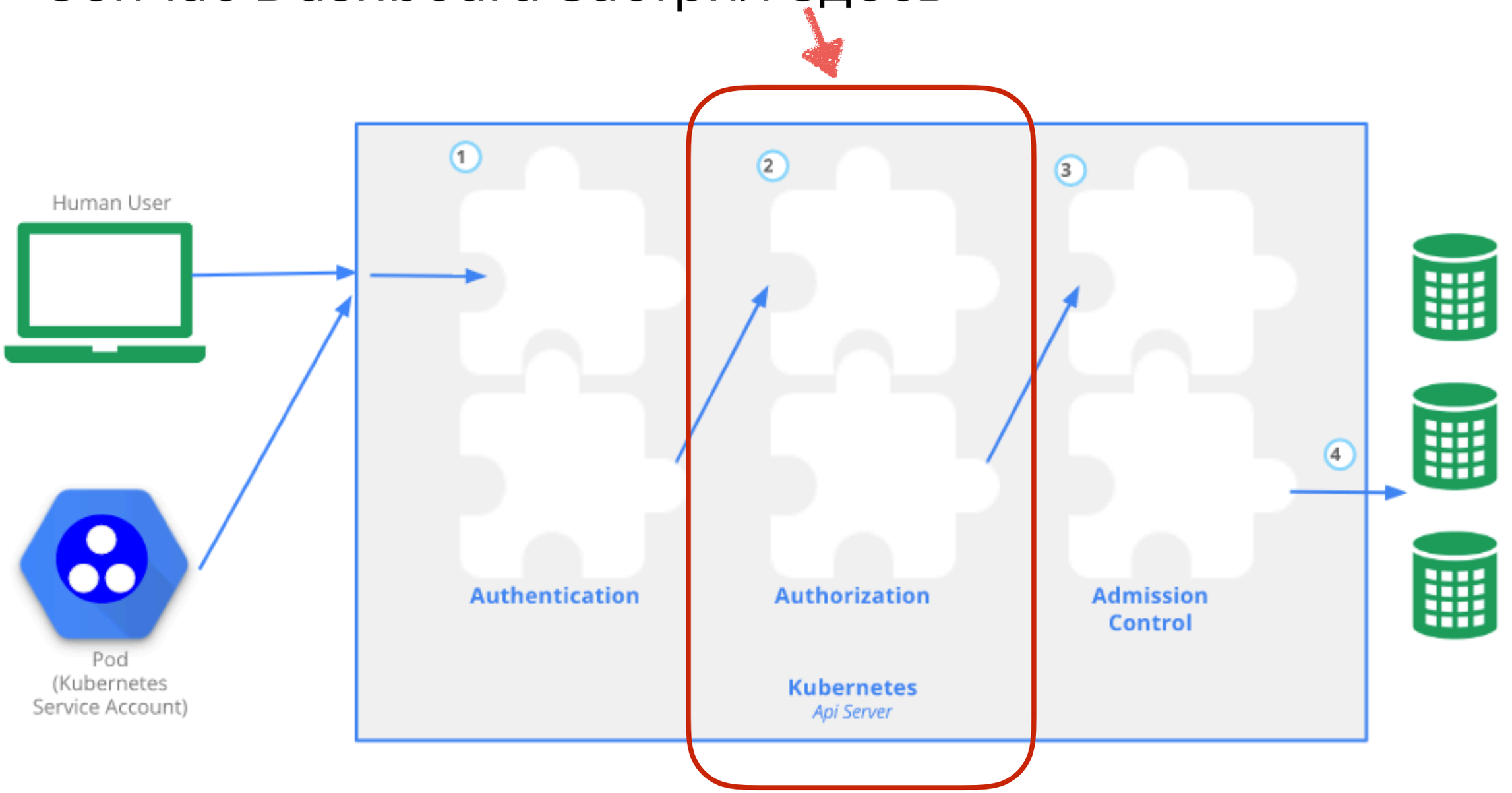


Security

У dashboard не хватает прав, чтобы посмотреть на кластер. Его не пускает **RBAC** (ролевая система контроля доступа). Нужно нашему Service Account назначить роль с достаточными правами на просмотр информации о кластере

Security

Сейчас Dashboard застрял здесь



Security

Нужно нашему Service Account назначить роль с достаточными правами на просмотр информации о кластере

В кластере уже есть объект ClusterRole с названием **cluster-admin**. Тот, кому назначена эта роль имеет полный доступ ко всем объектам кластера.

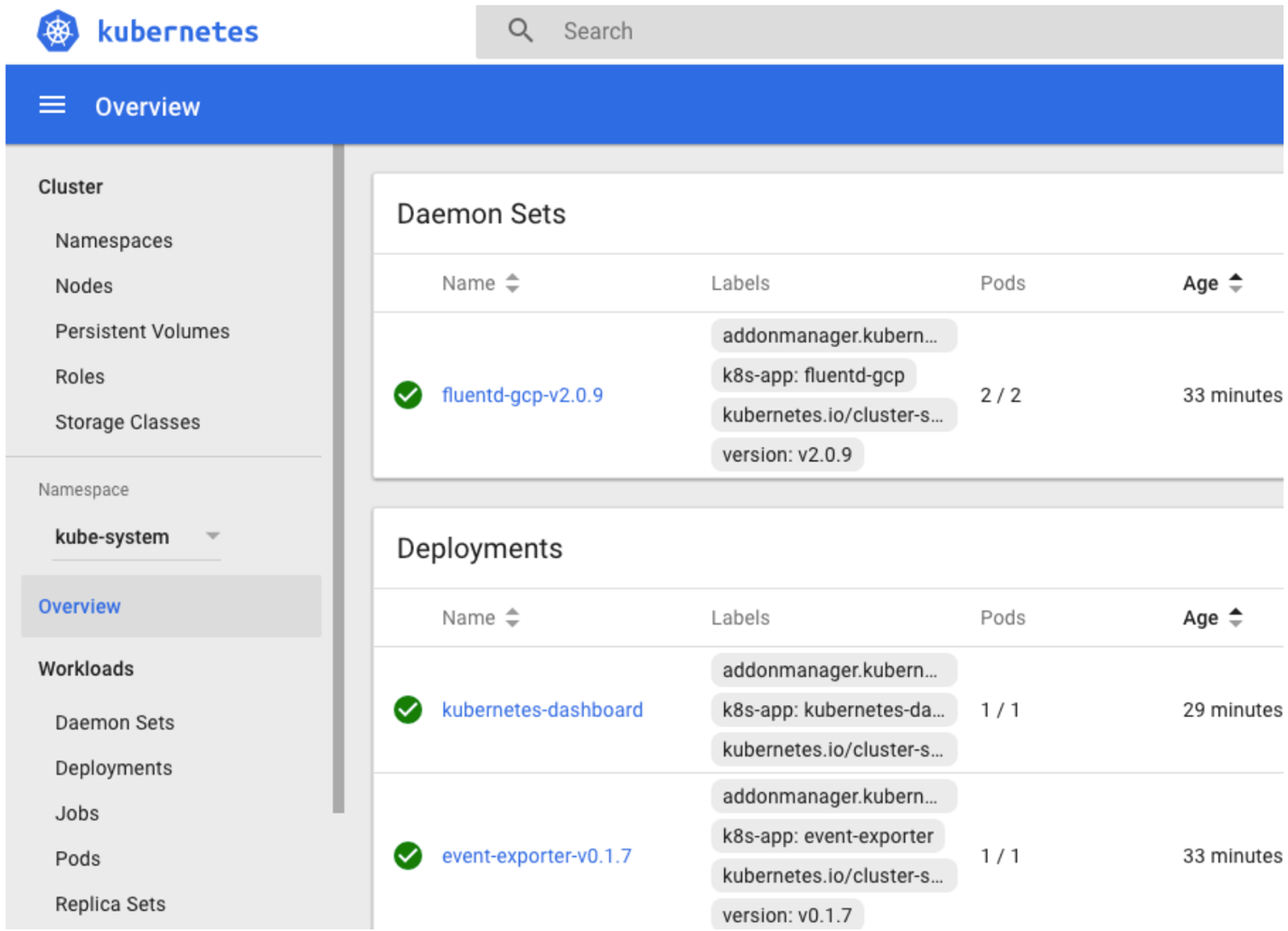
Давайте назначим эту роль service account-у dashboard-а (ссылка на gist) с помощью clusterrolebinding (привязки)

```
$ kubectl create clusterrolebinding kubernetes-dashboard
 --clusterrole=cluster-admin --serviceaccount=kube-system:kubernetes-dashboard
```

Для clusterrole, serviceaccount - это комбинация serviceaccount и namespace, в котором он создан

Security

Давайте снова http://localhost:8001/ui



Задание

*

1) Разверните Kubenetes-кластер в GKE с помощью Terraform модуля (https://www.terraform.io/docs/providers/google/r/container_cluster.html)

2) Создайте YAML-манифесты для описания созданных сущностей для включения dashboard.

3) Приложите конфигурацию к PR