CI/CD в Kubernetes

Подготовка

- Создайте новую ветку в репозитории Microservices для выполнения данного ДЗ. Т.к. это первое задание по Kubernetes, то назовите ее kubernetes-4
- Добавьте "Labels" Kubernetes и kubernetes-4 к вашему Pull Request
- Проверка данного ДЗ будет производиться через Pull Request ветки с ДЗ к ветке мастер
- После того, как один из преподавателей сделает approve пул реквеста, ветку с ДЗ можно смерджить.

План

- Работа с Helm
- Развертывание Gitlab в Kubernetes
- Запуск CI/CD конвейера в Kubernetes

Helm

Helm

Helm - пакетный менеджер для Kubernetes.

С его помощью мы будем:

1. Стандартизировать поставку приложения в Kubernetes
2. Декларировать инфраструктуру
3. Деплоить новые версии приложения

Helm - установка

Helm - клиент-серверное приложение. Установим его клиентскую часть - консольный клиент Helm

MacOS$ brew install kubernetes-helm

Linux, Windows,MacOS- выберите и скачайте пакет (2.9.1 на момент написания) https://github.com/kubernetes/helm/releases - распакуйте и разместите исполняемый файл helm в директории исполнения (/usr/local/bin/ , /usr/bin, …)

Helm - установка

Helm читает конфигурацию kubectl (`~/.kube/config`) и сам определяет текущий контекст (кластер, пользователь, неймспейс)

Если хотите сменить кластер, то либо меняйте контекст с помощью

```
$ kubectl config set-context
```

либо подгружайте helm'у собственный config-файл флагом `--kube-context`.

Helm - установка

Установим серверную часть Helm'а - *Tiller*

Tiller - это аддон Kubernetes, т.е. Pod, который общается с API Kubernetes.

> Для этого понадобится ему выдать *ServiceAccount* и назначить роли RBAC, необходимые для работы.

Helm - устанвка

Создайте файл tiller.yml и поместите в него манифест

```
$ kubectl apply -f tiller.yml
```

Теперь запустим tiller-сервер

```
$ helm init --service-account tiller
```

Проверим

```
$ kubectl get pods -n kube-system --selector app=helm
NAME                            READY     STATUS    RESTARTS   AGE
tiller-deploy-546cf9696c-j6k8w   1/1       Running   0          2m
```

Charts

Chart - это пакет в Helm.

Создайте директорию Charts в папке kubernetes со следующей структурой директорий:

```
├── Charts
    ├── comment
    ├── post
    ├── reddit
    └── ui
```

Charts

Начнем разработку Chart'а для компонента ui приложения Создайте файл-описание chart'а.

```
$ touch ui/Chart.yaml
```

Helm предпочитает .yaml

```
name: ui
version: 1.0.0
description: OTUS reddit application UI
maintainers:
  - name: Someone
    email: my@mail.com
appVersion: 1.0
```

Реально значимыми являются поля name и version. От них зависит работа Helm'а с Chart'ом. Остальное - описания.

Templates

Основным содержимым Chart'ов являются шаблоны манифестов Kubernetes.

1. Создайте директорию ui/templates
2. Перенесите в неё все манифесты, разработанные ранее для сервиса ui (ui-service, ui-deployment, ui-ingress)
3. Переименуйте их (уберите префикс "ui-") и поменяйте расширение на .yaml) - стилистические правки

```
└── ui
 ├── Chart.yaml
 ├── templates
 │   ├── deployment.yaml
 │   ├── ingress.yaml
 │   └── service.yaml
```

Templates

По-сути, это уже готовый пакет для установки в Kubernetes

1. Убедитесь, что у вас не развернуты компоненты приложения в kubernetes. Если развернуты - удалите их
2. Установим Chart

```
$ helm install --name test-ui-1 ui/
```

Передаем имя и путь до Chart'a соответсвенно 3)
Посмотрим, что получилось

```
$ helm ls
```

Templates

Теперь сделаем так, чтобы можно было использовать 1 Chart для запуска нескольких экземпляров (релизов). Шаблонизируем его. ui/templates/service.yaml (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
```

Templates

ui/templates/service.yaml (ссылка на gist)

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Release.Name }}-{{ .Chart.Name }}
  labels:
    app: reddit
    component: ui
    release: {{ .Release.Name }}
spec:
  type: NodePort
  ports:
  - port: {{ .Values.service.externalPort }}
    protocol: TCP
    targetPort: 9292
  selector:
    app: reddit
    component: ui
    release: {{ .Release.Name }}
```

Нам нужно уникальное имя запущенного ресурса

Помечаем, что сервис из конкретного релиза

Выбираем POD-ы только из этого релиза

Templates

name: {{ .Release.Name }}-{{ .Chart.Name }}

Здесь мы используем встроенные переменные

- .Release - группа переменных с информацией о релизе (конкретном запуске Chart'а в k8s)
- .Chart - группа переменных с информацией о Chart'е (содержимое файла Chart.yaml)

Также еще есть группы переменных:

- .Template - информация о текущем шаблоне (.Name и .BasePath)
- .Capabilities - информация о Kubernetes (версия, версии API)
- .Files.Get - получить содержимое файла

Templates

Шаблонизируем подобным образом остальные сущности *ui/templates/deployment.yaml*(ссылка на gist)

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: {{ .Release.Name }}-{{ .Chart.Name }}
  labels:
    app: reddit
    component: ui
    release: {{ .Release.Name }}
spec:
…
  selector:
    matchLabels:
      app: reddit
      component: ui
      release: {{ .Release.Name }}
  template:
    metadata:
      name: ui
      labels:
        app: reddit
        component: ui
        release: {{ .Release.Name }}
…
```

Важно, чтобы selector deployment'а нашел только нужные POD'ы.

Templates

Не забудьте поставить свои образы

```
containers:
- image: chromko/ui
  name: ui
```

Templates

Шаблонизируем подобным образом остальные сущности *ui/templates/ingress.yaml* (ссылка на gist)

```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: {{ .Release.Name }}-{{ .Chart.Name }}
  annotations:
    kubernetes.io/ingress.class: "gce"
spec:
  rules:
  - http:
      paths:
      - path: /*
        backend:
          serviceName: {{ .Release.Name }}-{{ .Chart.Name }}
          servicePort: 9292
```

Templates

Определим значения собственных переменных *ui/values.yaml* (ссылка на gist)

```
service:
  internalPort: 9292
  externalPort: 9292

image:
  repository: chromko/ui
  tag: latest
```

```
$ helm upgrade ui-1 ui/

$ helm upgrade ui-2 ui/

$ helm upgrade ui-3 ui/
```

Templates

Установим несколько релизов ui

```
$ helm install ui --name ui-1

$ helm install ui --name ui-2

$ helm install ui --name ui-3
```

Где ui-(1/2/3) - имена релизов

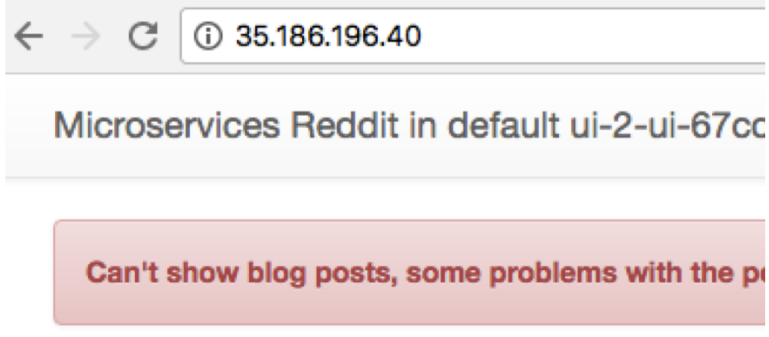
Должны появиться 3 ingress'a

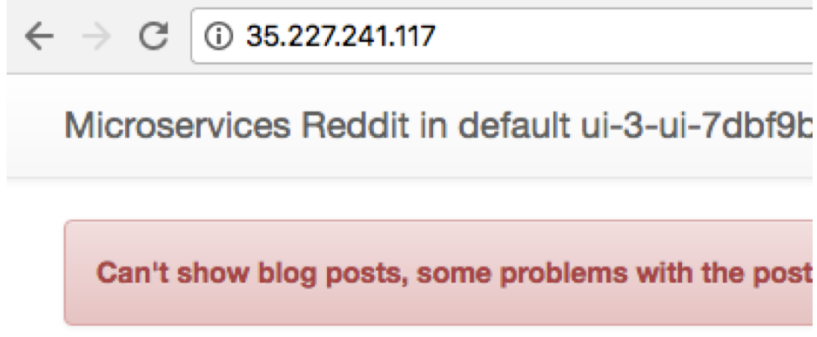
```
$ kubectl get ingress

NAME      HOSTS     ADDRESS           PORTS     AGE
ui-1-ui   *         35.190.86.49      80        13m
ui-2-ui   *         35.186.196.40     80        13m
ui-3-ui   *         35.227.241.117    80        13m
```

Templates

По IP-адресам можно попасть на разные релизы ui-приложений.P.S. подождите пару минут, пока ingress'ы станут доступными





Templates

Мы уже сделали возможность запуска нескольких версий приложений из одного пакета манифестов, используя лишь встроенные переменные. Кастомизируем установку своими переменными (образ и порт).

ui/templates/deployment.yaml (ссылка на gist)

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: {{ .Release.Name }}-{{ .Chart.Name }}
…
     containers:
      - image: "{{ .Values.image.repository }}/ui:{{ .Values.image.tag }}"
        name: ui
        ports:
        - containerPort: {{ .Values.service.internalPort }}
```

Templates

ui/templates/service.yaml (ссылка на gist)

```
apiVersion: v1
kind: Service
metadata:
…
spec:
  type: NodePort
  ports:
  - port: {{ .Values.service.externalPort }}
    protocol: TCP
    targetPort: {{ .Values.service.internalPort }}
  selector:
    app: reddit
    component: ui
    release: {{ .Release.Name }}
```

Templates

ui/templates/ingress.yaml (ссылка на gist)

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: {{ .Release.Name }}-{{ .Chart.Name }}
  annotations:
    kubernetes.io/ingress.class: "gce"
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: {{ .Release.Name }}-{{ .Chart.Name }}
          servicePort: {{ .Values.service.externalPort }}
```

Templates

Определим значения собственных переменных *ui/values.yaml* (ссылка на gist)

```
service:
  internalPort: 9292
  externalPort: 9292

image:
  repository: chromko/ui
  tag: latest
```

```
$ helm upgrade ui-1 ui/

$ helm upgrade ui-2 ui/

$ helm upgrade ui-3 ui/
```

Templates

Мы собрали Chart для развертывания ui-компоненты приложения. Он должен иметь следующую структуру

```
└── ui
    ├── Chart.yaml
    ├── templates
    │   ├── deployment.yaml
    │   ├── ingress.yaml
    │   └── service.yaml
    └── values.yaml
```

Осталось собрать пакеты для остальных компонент

Templates

post/templates/service.yaml (ссылка на gist)

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Release.Name }}-{{ .Chart.Name }}
  labels:
    app: reddit
    component: post
    release: {{ .Release.Name }}
spec:
  type: ClusterIP
  ports:
  - port: {{ .Values.service.externalPort }}
    protocol: TCP
    targetPort: {{ .Values.service.internalPort }}
  selector:
    app: reddit
    component: post
    release: {{ .Release.Name }}
```

Templates

post/templates/deployment.yaml (ссылка на gist)

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: {{ .Release.Name }}-{{ .Chart.Name }}
  labels:
    app: reddit
    component: post
    release: {{ .Release.Name }}
spec:
  replicas: 1
  selector:
    matchLabels:
      app: reddit
      component: post
      release: {{ .Release.Name }}
…
```

Templates

Обратим внимание на адрес БД

```
env:
        - name: POST_DATABASE_HOST
          value: postdb
```

Поскольку адрес БД может меняться в зависимости от условий запуска:

- бд отдельно от кластера
- бд запущено в отдельном релизе
- … , то создадим удобный шаблон для задания адреса БД.

```
env:
      - name: POST_DATABASE_HOST
        value: {{ .Values.databaseHost }}
```

Templates

Будем задавать бд через переменную databaseHost. Иногда лучше использовать подобный формат переменных вместо структур database.host, так как тогда прийдется определять структуру database, иначе helm выдаст ошибку.

Используем функцию default. Если databaseHost не будет определена или ее значение будет пустым, то используется вывод функции printf (которая просто формирует строку <имя-релиза>-mongodb)

```
value: {{ .Values.databaseHost | default (printf "%s-mongodb" .Release.Name) }}
```

```
release-name-mongodb
```

Templates

В итоге должно получиться (ссылка на gist)

```
        env:
         - name: POST_DATABASE_HOST
           value: {{ .Values.databaseHost | default (printf "%s-mongodb"
.Release.Name) }}
```

Теперь, если databaseHost не задано, то будет использован адрес базы, поднятой внутри релиза

Более подробная документация по шаблонизации и функциям

Templates

post/values.yaml (ссылка на gist)

```
service:
  internalPort: 5000
  externalPort: 5000

image:
  repository: chromko/post
  tag: latest

databaseHost:
```

Templates

Шаблонизируем сервис comment:

Здесь все очень похоже на сервис post:

- comment/templates/deployment.yaml (ссылка на gist
- comment/templates/service.yaml (ссылка на gist)
- comment/values.yaml (ссылка на gist)

не забудьте добавить Chart.yaml

Templates

Итоговая структура должна выглядеть так:

```
├── comment
│   ├── Chart.yaml
│   ├── charts
│   ├── templates
│   │   ├── deployment.yaml
│   │   └── service.yaml
│   └── values.yaml
├── post
│   ├── Chart.yaml
│   ├── templates
│   │   ├── deployment.yaml
│   │   └── service.yaml
│   └── values.yaml
└── ui
    ├── Chart.yaml
    ├── templates
    │   ├── deployment.yaml
    │   ├── ingress.yaml
    │   └── service.yaml
    └── values.yaml
```

Templates

Также стоит отметить функционал helm по использованию helper'ов и функции templates. *Helper* - это написанная нами функция. В функция описывается, как правило, сложная логика. Шаблоны этих функция распологаются в файле `_helpers.tpl`

Пример функции comment.fullname: `charts/comment/templates/_helpers.tpl`

```
{{- define "comment.fullname" -}}
{{- printf "%s-%s" .Release.Name .Chart.Name }}
{{- end -}}
```

которая в результате выдаст то же, что и:

```
{{ .Release.Name }}-{{ .Chart.Name }}
```

Templates

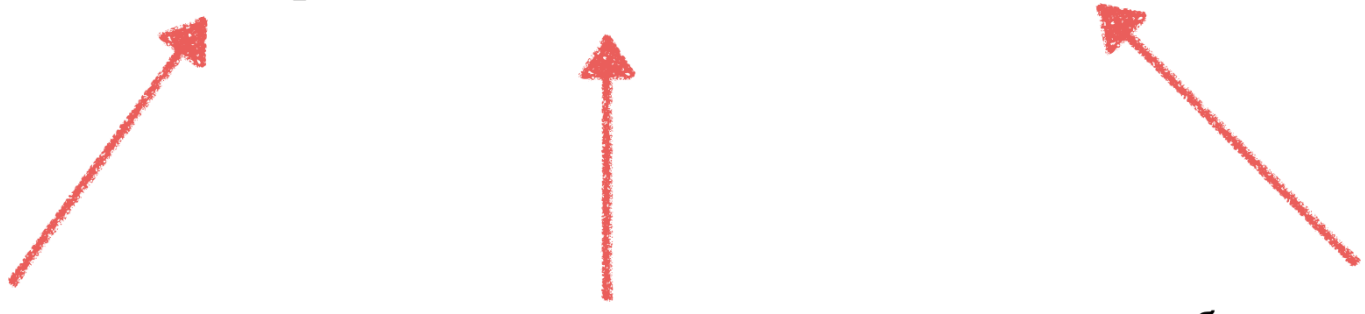
И заменим в соответствующие строчки в файле, чтобы использовать helper charts/comment/templates/service.yaml (ссылка на gist)

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Release.Name }}-{{ .Chart.Name }}
  labels:
```

Templates

Структура ипортирующей функции template

```
{{ template "comment.fullname" . }}
```



Функция template

Название функции для импорта

область видимости для импорта

“.”- вся область видимости всех переменных (можно передать .Chart, тогда .Values не будут доступны внутри функции)

Задание

1. Создать файл `_helpers.tpl` в папках templates сервисов ui, post и comment
2. вставить функцию “.fullname” в каждый `_helpers.tpl` файл. заменить на имя чарта соотв. сервиса
3. В каждом из шаблонов манифестов вставить следующую функцию там, где это требуется (большинство полей это name:)

```
{{ template "comment.fullname" . }}
```

Управление зависимостями

Структура становится следующая:

```
├── comment
│   ├── Chart.yaml
│   ├── charts
│   ├── templates
│   │   ├── _helpers.tpl
│   │   ├── deployment.yaml
│   │   └── service.yaml
│   └── values.yaml
├── post
│   ├── Chart.yaml
│   ├── templates
│   │   ├── _helpers.tpl
│   │   ├── deployment.yaml
│   │   └── service.yaml
│   └── values.yaml
└── ui
    ├── Chart.yaml
    ├── templates
    │   ├── _helpers.tpl
    │   ├── deployment.yaml
    │   ├── ingress.yaml
    │   └── service.yaml
    └── values.yaml
```

Управление зависимостями

Мы создали Chart'ы для каждой компоненты нашего приложения. Каждый из них можно запустить по-отдельности командой

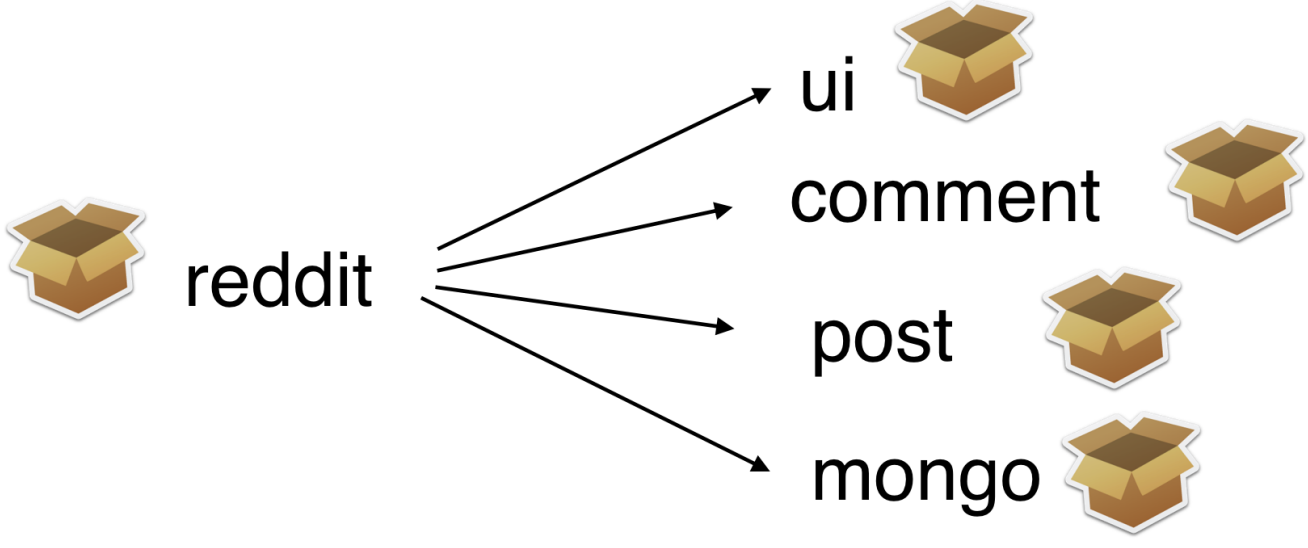
```
$ helm install <chart-path> <release-name>
```

Но они будут запускаться в разных релизах, и не будут видеть друг друга.

С помощью механизма управления зависимостями создадим единый Chart reddit, который объединит наши компоненты

Управление зависимостями

Структура приложения reddit



Управление зависимостями

1. Создайте `reddit/Chart.yaml` (ссылка на gist)

```
name: reddit
version: 0.1.0
description: OTUS sample reddit application
maintainers:
- name: Someone
  email: my@gmail.com
```

Заполните параметры name и email своими данными

2. Создайте пустой `reddit/values.yaml`

Управление зависимостями

В директории Chart'a reddit создадим файл `reddit/requirements.yaml`ссылка на gist

```
dependencies:
  - name: ui
    version: "1.0.0"
    repository: "file://../ui"
  - name: post
    version: "1.0.0"
    repository: file://../post
  - name: comment
    version: “1.0.0"
    repository: file://../comment
```

Имя и версия должны совпадать с содержанием ui/Chart.yml

Путь указывается относительно расположения самого requirements.yaml

Управление зависимостями

Нужно загрузить зависимости (когда Chart' не упакован в tgz архив)

```
$ helm dep update
```

Появится файл `requirements.lock` с фиксацией зависимостей Будет создана директория charts с зависимостями в виде архивов

Структура станет следующей:

```
reddit
├── Chart.yaml
├── charts
│   ├── comment-1.0.0.tgz
│   ├── post-1.0.0.tgz
│   └── ui-1.0.0.tgz
├── requirements.lock
├── requirements.yaml
└── values.yaml
```

Управление зависимостями

Chart для базы данных не будем создавать вручную. Возьмем готовый.

1. Найдем Chart в общедоступном репозитории

```
$ helm search mongo
NAME                            VERSION    DESCRIPTION
stable/mongodb                  0.4.18     NoSQL document-oriented database
that stores JS...
stable/mongodb-replicaset       2.1.3      NoSQL document-oriented database
that stores JS…
```

Управление зависимостями

2. добавим в reddit/requirements.yml (ссылка на gist)

```
dependencies:
…
- name: comment
 version: 1.0.0
 repository: file://../comment
- name: mongodb
 version: 0.4.18
 repository: https://kubernetes-charts.storage.googleapis.com
```

3. Выгрузим зависимости

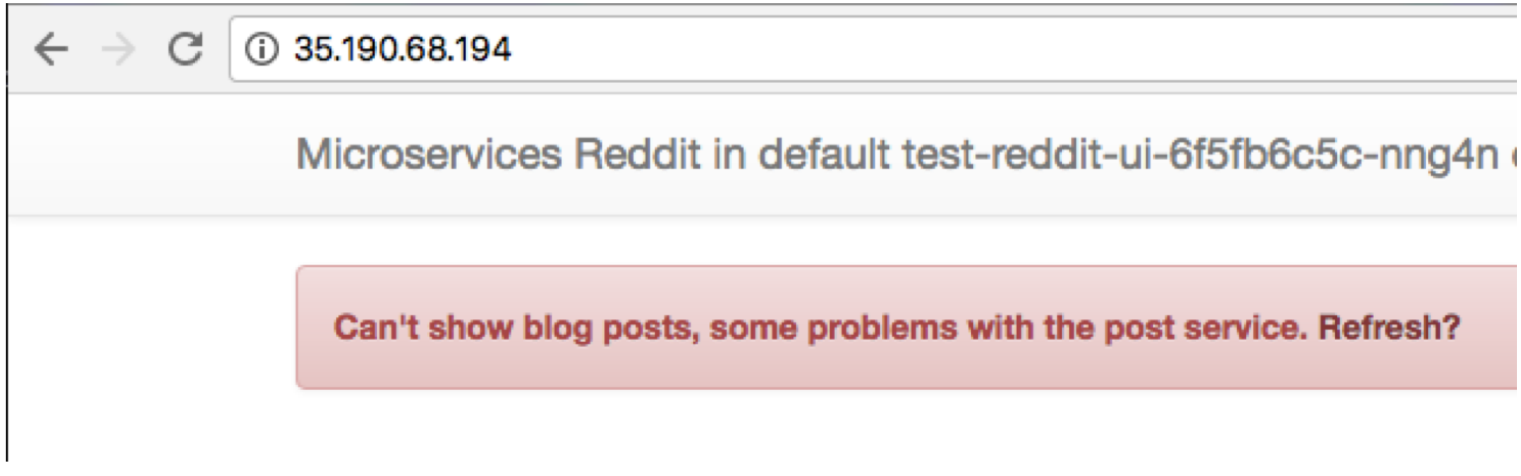
```
$ helm dep update
```

Управление зависимостями

Установим наше приложение:

```
/Charts] $ helm install reddit --name reddit-test
```

Найдите адрес ingress’a с помощью kubectl Подождать пока ingress обработается и …



Управление зависимостями

Есть проблема с тем, что UI-сервис не знает как правильно ходить в post и comment сервисы. Ведь их имена теперь динамические и зависят от имен чартов

В Dockerfile UI-сервиса уже заданы переменные окружения. Надо, чтобы они указывали на нужные бекенды

```
ENV POST_SERVICE_HOST post
ENV POST_SERVICE_PORT 5000
ENV COMMENT_SERVICE_HOST comment
ENV COMMENT_SERVICE_PORT 9292
```

Управление зависимостями

Добавим в ui/deployments.yaml (ссылка на gist)

```
   spec:
      containers:
}
…
        env:
        - name: POST_SERVICE_HOST
          value: {{  .Values.postHost | default (printf "%s-post" .Release.Name)
}}
        - name: POST_SERVICE_PORT
          value: {{  .Values.postPort | default "5000" | quote }}
        - name: COMMENT_SERVICE_HOST
          value: {{  .Values.commentHost | default (printf "%s-comment"
.Release.Name) }}
        - name: COMMENT_SERVICE_PORT
          value: {{  .Values.commentPort | default "9292" | quote }}
        - name: ENV
```

{{ .Values.commentPort | default "9292" | quote }} ! обратите внимание на функцию добавления кавычек. Для чисел и булевых значений это важно

Управление зависимостями

Добавим в `ui/values.yaml` (ссылка на gist)

```
  …
postHost:
postPort:
commentHost:
commentPort:
```

Можете даже закоментировать эти параметры или оставить пустыми. Главное, чтобы они были в конфигурации Chart'а в качестве документации

Управление зависимостями

Вы можете задавать теперь переменные для зависимостей прямо в `values.yaml` самого Chart'a reddit. Они перезаписывают значения переменных из зависимых чартов

`reddit/values.yaml` (ссылка на gist)

```
comment:
  image:
    repository: chromko/comment
    tag: latest
  service:
    externalPort: 9292

post:
  image:
    repository: chromko/post
    tag: latest
    service:
      externalPort: 5000
…
```

Ссылаемся на переменные чартов из зависимостей

Управление зависимостями

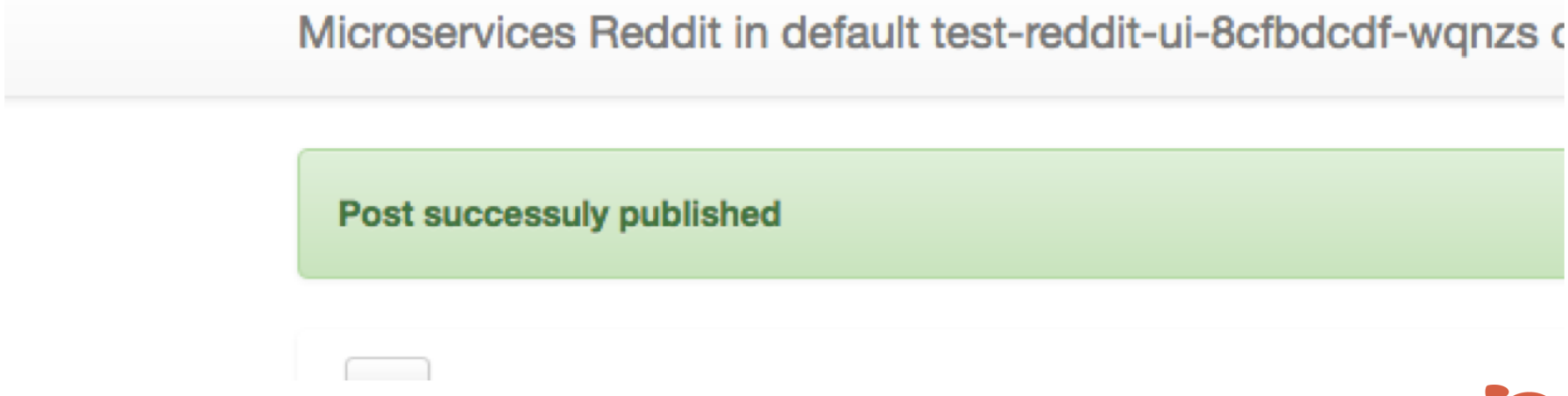
После обновления UI - нужно обновить зависимости чарта reddit.

```
$ helm dep update ./reddit
```

Обновите релиз, установленный в k8s

```
$ helm upgrade <release-name> ./reddit
```

снова проверьте UI

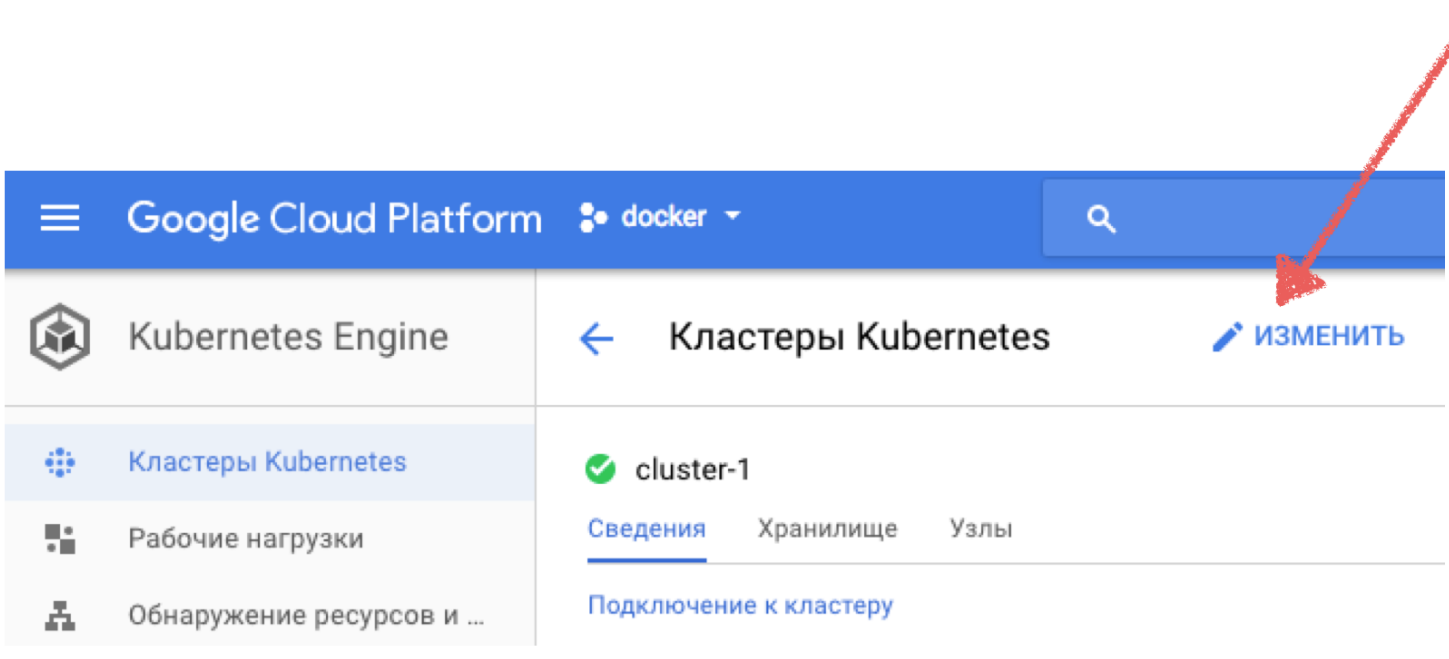


GitLab + Kubernetes

Установим GitLab

Подготовим GKE-кластер. Нам нужны машинки помощнее.

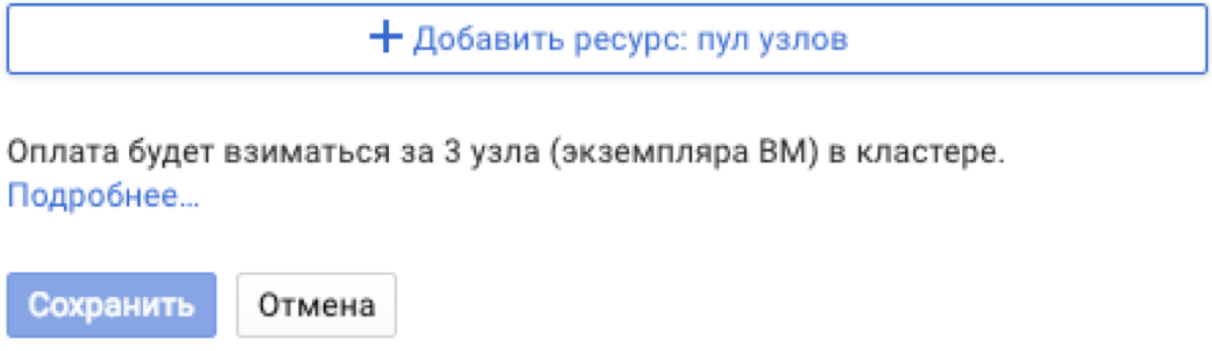
Зайдите в настройки своего кластера и нажмите “изменить”



Установим GitLab

Добавьте новый пул узлов:

- назовите его bigpool
- 1 узел типа n2-standard-2 (7,5 Гб, 2 виртуальных ЦП)
- Размер диска 20-40 Гб



С помощью пулов узлов можно добавлять в кластер новые машины разной мощности и комплектации.

P.S. подождите пока кластер будет готов к работе

Установим GitLab

Отключите RBAC для упрощения работы. Gitlab-Omnibus пока не подготовлен для этого, а самим это в рамках работы смысла делать нет.

Там же, в настройках кластера

Установим GitLab

Gitlab будем ставить также с помощью Helm Chart'а из пакета Omnibus.

1. Добавим репозиторий Gitlab

```
$ helm repo add gitlab https://charts.gitlab.io
```

2. Мы будем менять конфигурацию Gitlab, поэтому скачаем Chart

```
$ helm fetch gitlab/gitlab-omnibus  --version 0.1.37 --untar

$ cd gitlab-omnibus
```

Установим GitLab

1. Поправьте `gitlab-omnibus/values.yaml` (ссылка на gist)

```
baseDomain: example.com
legoEmail: you@example.com
```

2. Добавьте в `gitlab-omnibus/templates/gitlab/gitlab-svc.yaml` (ссылка на gist)

```
apiVersion: v1
kind: Service
metadata:
name: {{ template "fullname" . }}
…
selector:
 name: {{ template "fullname" . }}
ports:
…
- name: prometheus
   port: 9090
   targetPort: prometheus
 - name: web
   port: 80
   targetPort: workhorse
```

Установим GitLab

3. Поправить в `gitlab-omnibus/templates/gitlab-config.yaml` (ссылка на gist)

```
apiVersion: v1
kind: ConfigMap
…
heritage: "{{ .Release.Service }}"
data:
external_scheme: http
external_hostname: {{ template "fullname" . }}
```

```
4. Поправить в `gitlab-omnibus/templates/ingress/gitlab-ingress.yaml` ([ссылка на gist](https://raw.githubusercontent.com/express42/otus-snippets/master/kubernetes-4/gitlab/ingress.yml))
```yaml
apiVersion: extensions/v1beta1
kind: Ingress
…
spec:
  tls:
  …
 rules:
  - host: {{ template "fullname" . }}
```
```

Установим GitLab

```
$ helm install --name gitlab . -f values.yaml
```

Должно пройти несколько минут. Найдите выданный IP-адрес ingress-контроллера nginx.

```
$ kubectl get service  -n nginx-ingress nginx

NAME      TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
nginx     LoadBalancer   10.11.241.113   35.184.199.209
80:31782/TCP,443:30043/TCP,22:31194/TCP   23m
```

Поместите запись в локальный файл `/etc/hosts` (поставьте свой IP-адрес)

```
$ echo "35.184.199.209 gitlab-gitlab staging production" >> /etc/hosts
```

Установим GitLab

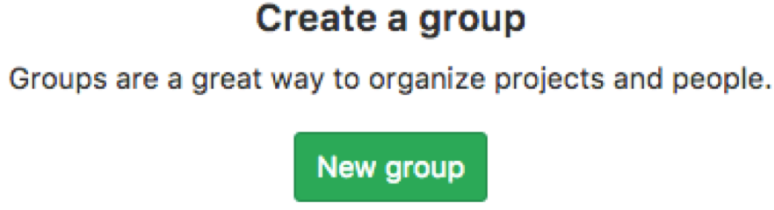
Ждем пока gitlab поднимется

```
kubectl get pods                                [17:39:22]
NAME                                 READY     STATUS    RESTARTS   AGE
gitlab-gitlab-847dc985b5-tfhd9       1/1       Running   0          1d
```

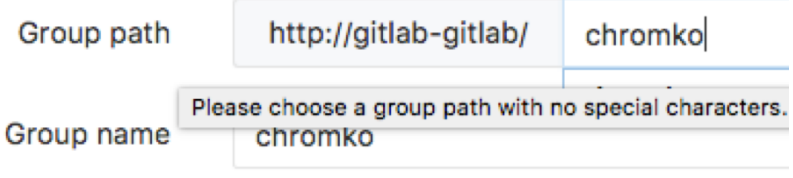
Идем по адресу http://gitlab-gitlab

Ставим собственный пароль. Логинимся под пользователем root и новым паролем otusgitlab.

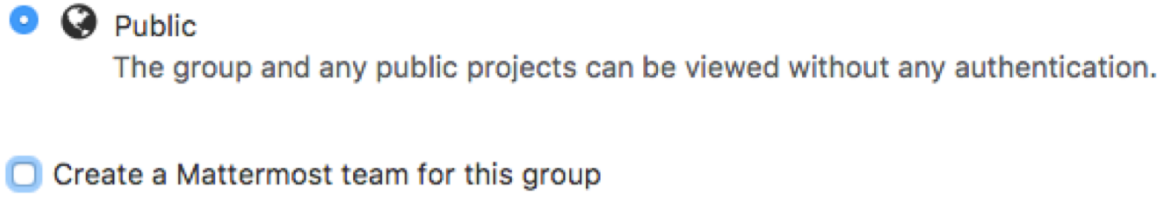
Запустим проект

1. 

Создайте группу

2. 

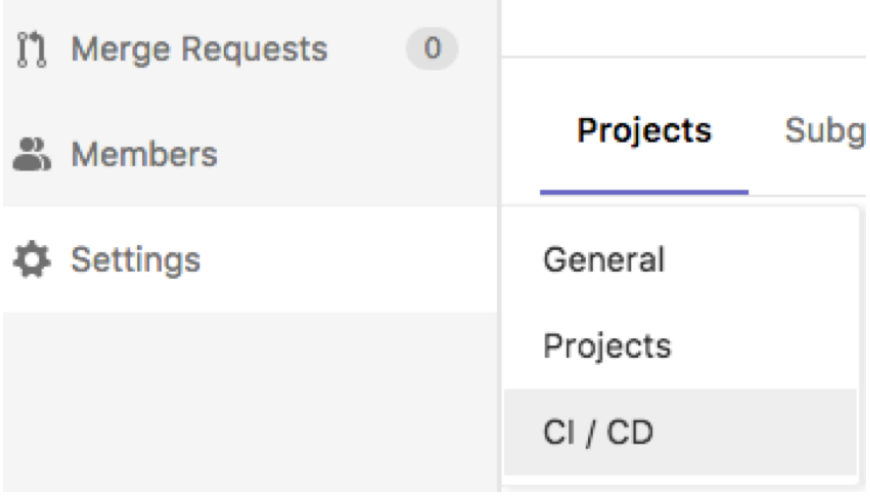
В качестве имени
Введите свой Docker ID

3. 

Запустим проект

http://gitlab-gitlab/имя_вашей_группы

В настройках группы выберите пункт CI/CD



Запустим проект

Добавьте 2 переменные - *CI_REGISTRY_USER* - логин в dockerhub *CI_REGISTRY_PASSWORD* - пароль от Docker Hub

Add new variable

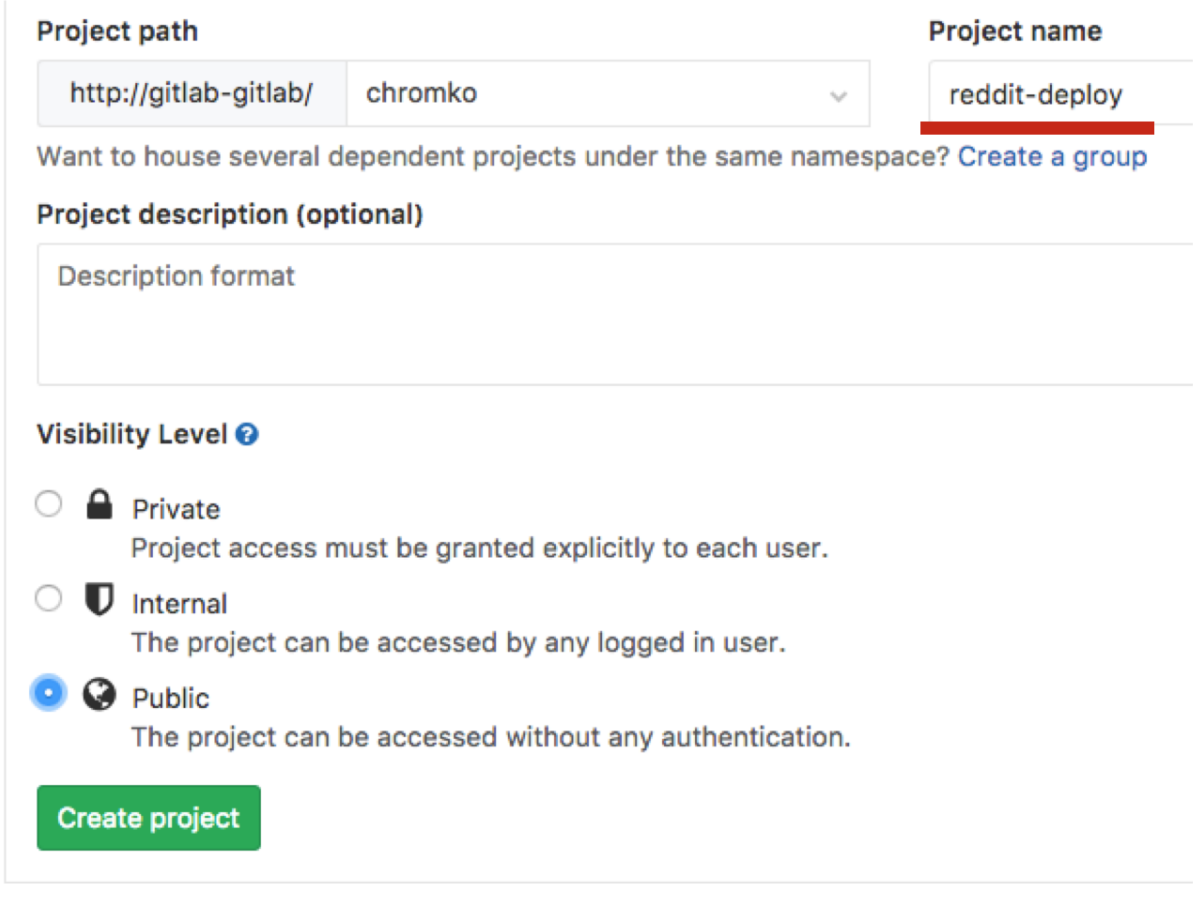
Your variables (2)

Key
CI_REGISTRY_PASSWORD
CI_REGISTRY_USER

Эти учетные данные будут использованы при сборке и релизе docker-образов с помощью Gitlab CI

Запустим проект

В группе создадим новый проект



Задание

Создайте еще 3 проекта: post, ui, comment (сделайте также их публичными) Структура проектов в группе должна выглядеть так:

Запустим проект

Локально у себя создайте директорию Gitlab_ci со следующей структурой директорий.

```
Gitlab_ci
├── comment
├── post
├── reddit-deploy
└── ui
```

Запустим проект

Перенесите исходные коды сервиса ui в Gitlab_ci/ui

Примерная структура директории будет похожа на:

```
ui
├── Dockerfile
├── Gemfile
├── Gemfile.lock
├── VERSION
├── config.ru
├── docker_build.sh
├── helpers.rb
├── middleware.rb
├── ui_app.rb
└── views
    ├── create.haml
    ├── index.haml
    ├── layout.haml
    └── show.haml
```

Запустим проект

В директории Gitlab_ci/ui:

1. Инициализируем локальный git-репозиторий

```
$ git init
```

2. Добавим удаленный репозиторий

```
$ git remote add origin http://gitlab-gitlab/chromko/ui.git
```

3. Закоммитим и отправим в gitlab

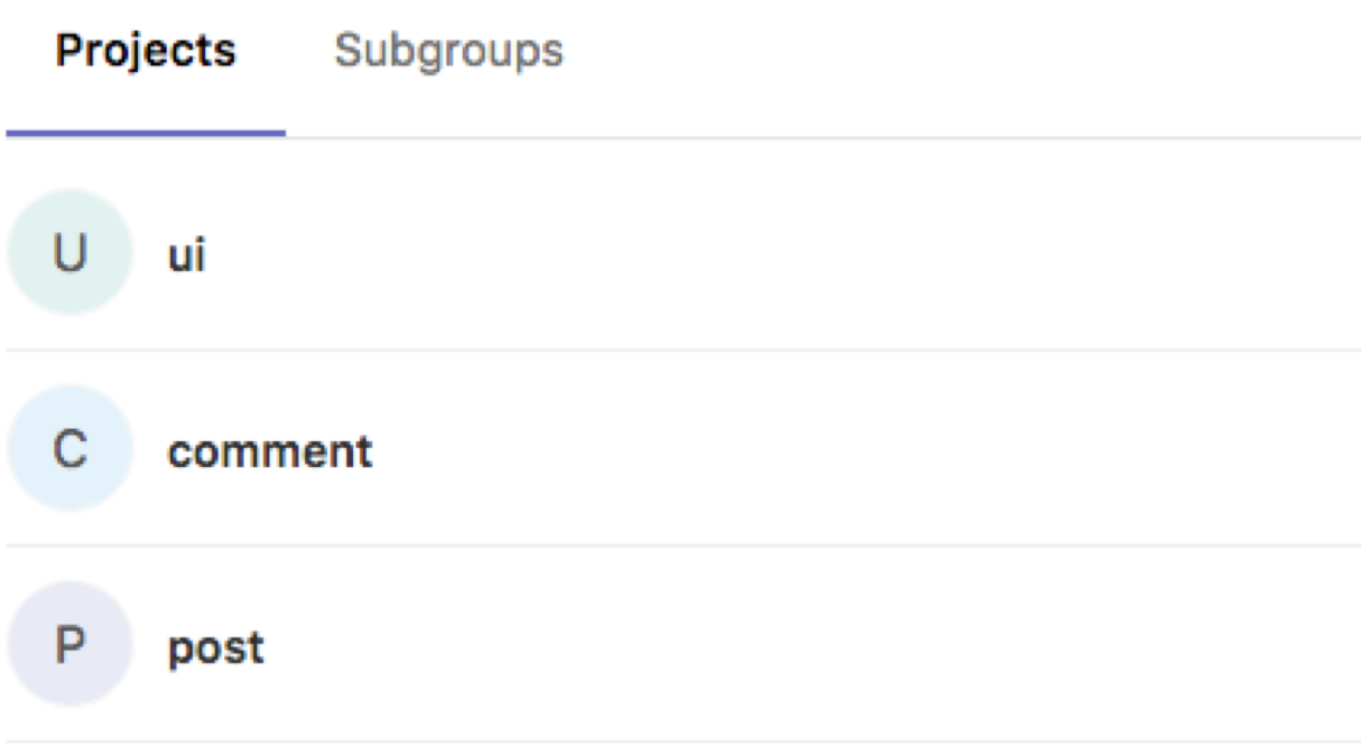
```
$ git add .
$ git commit -m “init”
$ git push origin master
```

Не забудьте подставить название своей группы

Задание

Для post и comment продейлайте аналогичные действия. Не забудьте указывать соответствующие названия репозиториев и групп.

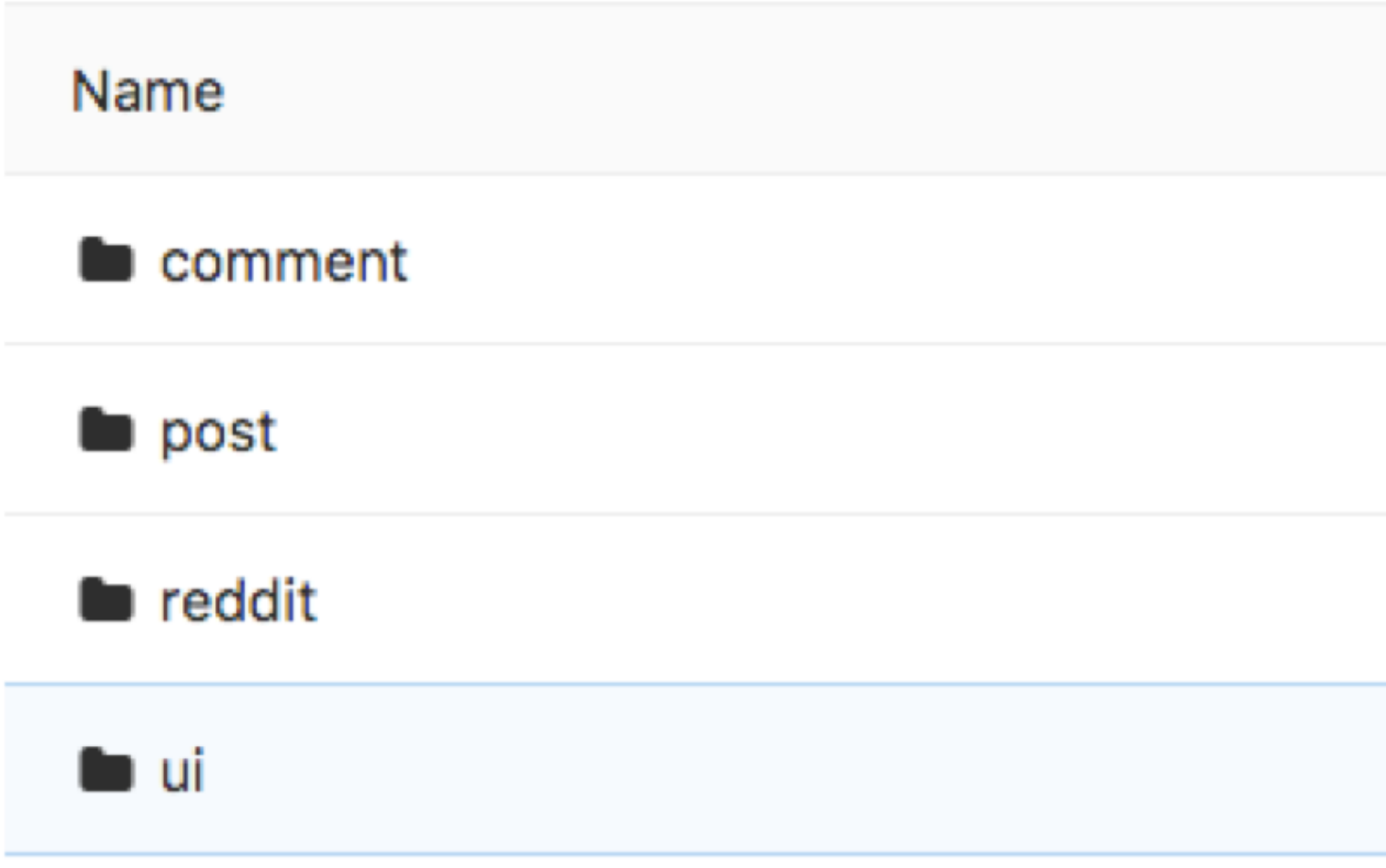
Структура группы



Задание

1. Перенести содержимое директории Charts (папки ui, post, comment, reddit) в Gitlab_ci/reddit-deploy
2. Запушить reddit-deploy в gitlab-проект reddit-deploy

Структура должна выглядеть так



Настроим CI

Настроим CI

В текущей конфигурации CI выполняет

1. Build: Сборку докер-образа с тегом master
2. Test: Фиктивное тестирование
3. Release: Смену тега с master на тег из файла VERSION и пуш docker-образа с новым тегом

Job для выполнения каждой задачи запускается в отдельном Kubernetes POD-е.

Настроим CI

Требуемые операции вызываются в блоках script:

```
script:
   - setup_docker
   - build
```

Описание самих операций производится в виде bash-функций в блоке .auto_devops

```
.auto_devops: &auto_devops |

 function setup_docker() {
…
  }

  function release() {
…
  }

  function build() {
…
}
```

Задание

Для Post и Comment также добавьте в репозиторий `.gitlab-ci.yml` и проследите, что сборки образов прошли успешно.

Настроим CI

Дадим возможность разработчику запускать отдельное окружение в Kubernetes по коммиту в feature-бранч.

Немного обновим конфиг ингресса для сервиса UI:

`reddit-deploy/ui/templates/ingress.yml` (ссылка на gist)

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: {{ template "ui.fullname" . }}
  annotations:
    kubernetes.io/ingress.class: {{ .Values.ingress.class }}
spec:
  rules:
  - host: {{ .Values.ingress.host | default .Release.Name }}
    http:
      paths:
      - path: /
…
```

Настроим CI

Обновим конфиг ингресса для сервиса UI:

`reddit-deploy/ui/templates/values.yml` (ссылка на gist):

```
---
service:
  internalPort: 9292
  externalPort: 9292
…

ingress:
  class: nginx
…
```

Будем использовать nginx-ingress, который был поставлен вместе с gitlab-ом (так быстрее и правила более гибкие, чем у GCP)

Настроим CI

Дадим возможность разработчику запускать отдельное окружение в Kubernetes по коммиту в feature-бранч.

1. Создайте новый бранч в репозитории ui

```
$ git checkout -b feature/3
```

2. Обновите `ui/.gitlab-ci.yml` файл (ссылка на gist)
3. Закоммитьте и запушьте изменения

```
$ git commit -am "Add review feature"
$ git push origin feature/3
```

Настроим CI

В коммитах ветки feature/3 можете найти сделанные изменения

Отметим, что мы добавили стадию review, запускающую приложение в k8s по коммиту в feature-бранчи (не master).

```
review:
  stage: review
  script:
    - install_dependencies
    - ensure_namespace
    - install_tiller
    - deploy
  variables:
    KUBE_NAMESPACE: review
    host: $CI_PROJECT_PATH_SLUG-$CI_COMMIT_REF_SLUG
  environment:
    name: review/$CI_PROJECT_PATH/$CI_COMMIT_REF_NAME
    url: http://$CI_PROJECT_PATH_SLUG-$CI_COMMIT_REF_SLUG
  only:
    refs:
      - branches
    kubernetes: active
  except:
    - master
```

Настроим CI

Мы добавили функцию deploy, которая загружает Chart из репозитория reddit-deploy и делает релиз в неймспейсе review с образом приложения, собранным на стадии build.

```
 function deploy() {
…
    git clone http://gitlab-gitlab/$CI_PROJECT_NAMESPACE/reddit-deploy.git

    echo "Download helm dependencies..."
    helm dep update reddit-deploy/reddit

    echo "Deploy helm release $name to $KUBE_NAMESPACE"
    helm upgrade --install \
      --wait \
      --set ui.ingress.host="$host" \
      --set $CI_PROJECT_NAME.image.tag=$CI_APPLICATION_TAG \
      --namespace="$KUBE_NAMESPACE" \
      --version="$CI_PIPELINE_ID-$CI_JOB_ID" \
      "$name" \
      reddit-deploy/reddit/
  }
```

Настроим CI

Можем увидеть какие релизы запущены

```
$ helm ls

NAME                            REVISION    UPDATED                         STATUS
CHART                           NAMESPACE
gitlab                          1           Fri Dec  8 03:03:55 2017     DEPLOYED
gitlab-omnibus-0.1.36           default
review-chromko-ui-p81rxy        1           Sat Dec  9 16:21:25 2017     DEPLOYED
reddit-0.1.1                    review
```

Настроим CI

Созданные для таких целей окружения временны, их требуется “убивать”, когда они больше не нужны.

Добавьте в `.gitlab-ci.yml` (ссылка на gist)

```
stop_review:
  stage: cleanup
  variables:
    GIT_STRATEGY: none
  script:
    - install_dependencies
    - delete
  environment:
    name: review/$CI_PROJECT_PATH/$CI_COMMIT_REF_NAME
    action: stop
  when: manual
  allow_failure: true
  only:
    refs:
      - branches
    kubernetes: active
  except:
    - master
```

Настроим CI

Добавьте также

```
stages:
  - build
  - test
  - review
  - release
  - cleanup

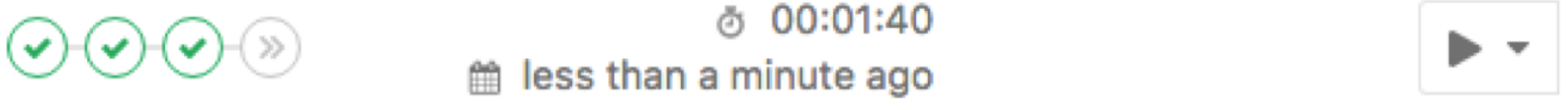
review:
  stage: review
  …
  environment:
    name: review/$CI_PROJECT_PATH/$CI_COMMIT_REF_NAME
    url: http://$CI_PROJECT_PATH_SLUG-$CI_COMMIT_REF_SLUG
    on_stop: stop_review
  …
```

Настроим CI

Добавьте функцию удаления окружения (ссылка на gist)

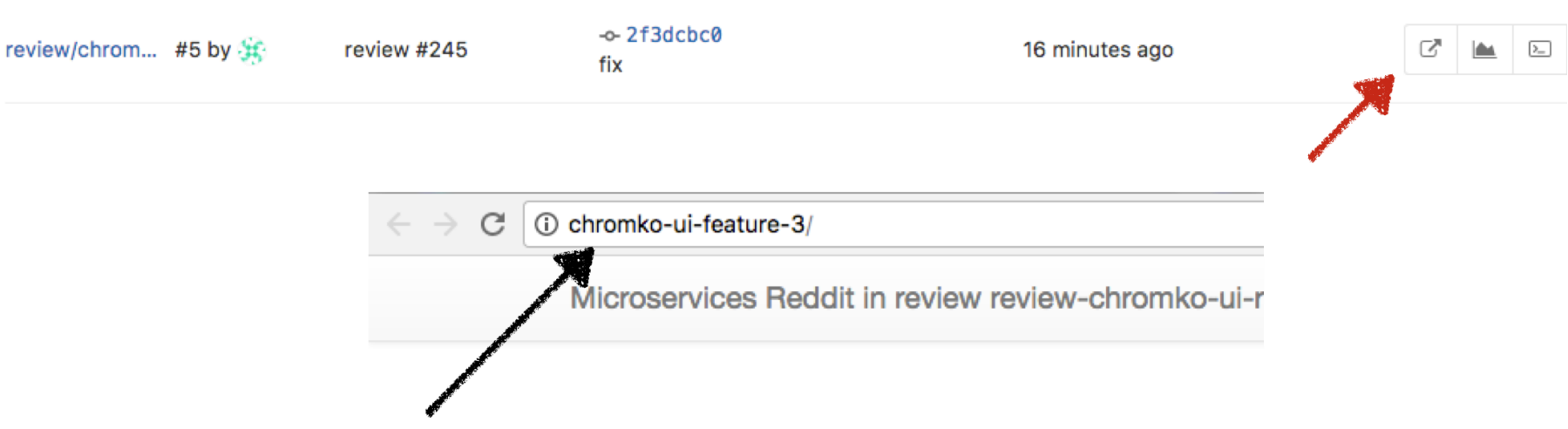
```
.auto_devops: &auto_devops |
…
  function delete() {
    track="${1-stable}"
    name="$CI_ENVIRONMENT_SLUG"
    helm delete "$name" --purge || true
  }
```

1. Запуште изменения в Git
2. зайдите в Pipelines ветки feature/3



Настроим CI

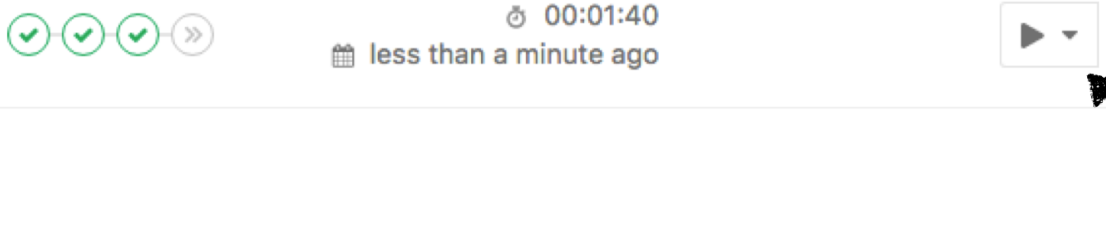
В Environments



Внесите этот адрес в /etc/hosts под адресом gitlab’a

Настроим CI

В Pipelines:



Запустите
удаление окружения

```
$ helm ls

NAME                    REVISION    UPDATED                     STATUS    CHART                   NAMESPACE
gitlab                  1           Fri Dec  8 03:03:55 2017    DEPLOYED  gitlab-omnibus-0.1.36   default
```

Задание

Скопировать полученный файл `.gitlab-ci.yml` для ui в репозитории для post и comment.

Проверить, что динамическое создание и удаление окружений работает и с ними как ожидалось

Деплоим

Теперь создадим staging и production среды для работы приложения

Создайте файл `reddit-deploy/.gitlab-ci.yml` (ссылка на gist)

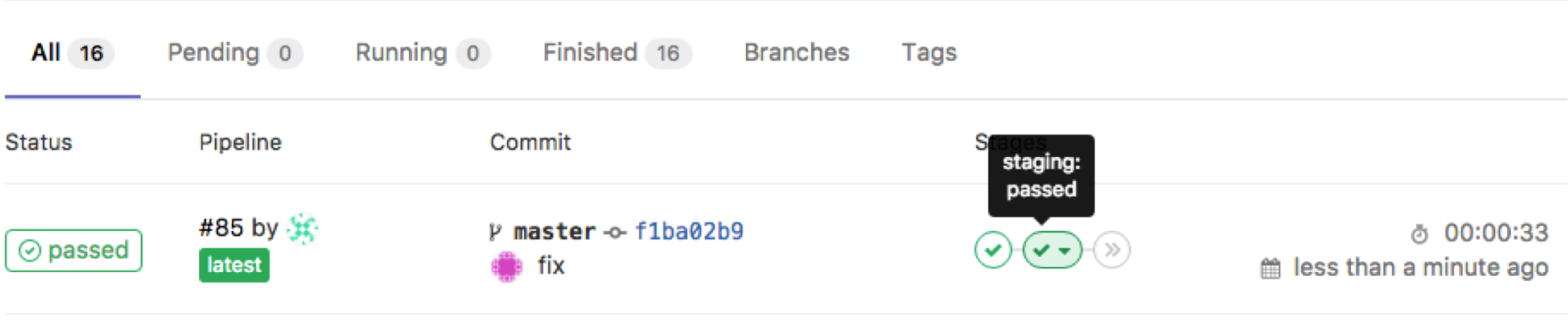
Запуште в репозиторий reddit-deploy ветку master

Этот файл отличается от предыдущих тем, что:

1. Не собирает docker-образы
2. Деплоит на статичные окружения (staging и production)
3. Не удаляет окружения

Деплоим

Удостоверьтесь, что staging успешно завершен

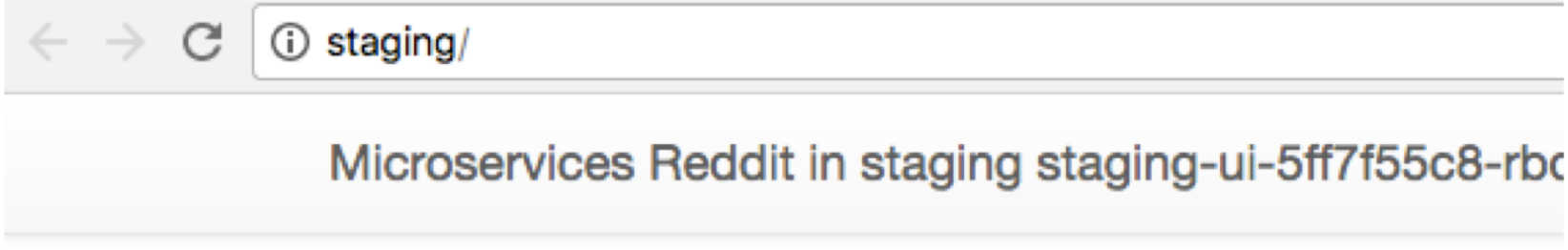


В Environments найдите staging

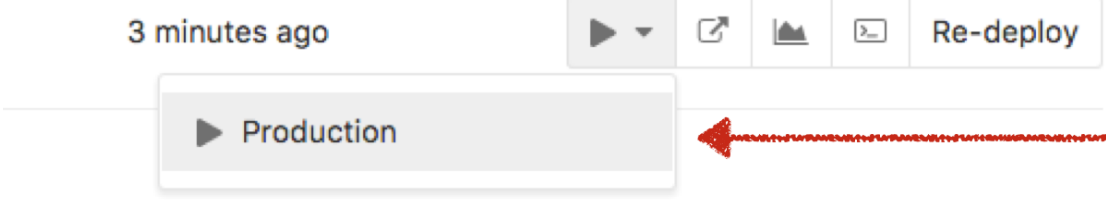
Перейдем по URL

Деплоим

Приложение работает!



Выкатываем на Production:



Ручной деплой

И ждем, пока пайплай пройдет

Деплоим

В Environments

Посмотрите что в helm также все видно

```
$ helm ls`
```

Задание

Файлы `.gitlab-ci.yml`, полученные в ходе работы, поместите в папку с исходниками для каждой компоненты приложения.

Файл `.gitlab-ci.yml` для reddit-deploy поместите в charts

Все изменения, которые были внесены в Chart'ы - перенести в папку charts, созданную вначале.

Папку Gitlab_ci - не комитить!

Задание со *

Сейчас у нас выкатка на staging и production - по нажатию кнопки

Свяжите пайплайны сборки образов и пайплайн деплоя на staging и production так, чтобы после релиза образа из ветки мастер - запускался деплой уже новой версии приложения на production