Возможные проблемы с CI/CD

Возможные проблемы с CI/CD

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: myapp:v1
        env:
        - name: DEBUG
          value: ???
```

DEBUG: true

DEV namespace

DEBUG: false

PROD namespace

Возможные проблемы с CI/CD

```
 1 apiVersion: apps/v1
 2 kind: Deployment
 3 metadata:
 4   name: myapp
 5 spec:
 6   replicas: 1
 7   template:
 8     metadata:
 9       labels:
10         app: myapp
11     spec:
12       containers:
13       - name: myapp
14         image: myapp:v1
15         env:
16         - name: DEBUG
17           value: {{
   .Values.debug.enabled }}
```

DEBUG: true

DEV namespace

DEBUG: false

PROD namespace

Возможные проблемы с CI/CD

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: myapp:v1
        volumeMounts:
        - name: ...
      volumes:
      - name: ...
```

Возможные проблемы с CI/CD

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: myapp:v1
{{- if .Values.persistentVolume.enabled }}
        volumeMounts:
        - name: ...
      volumes:
      - name: ...
{{ end }}
```

Возможные проблемы с CI/CD

Как выкатывать новую версию приложения?

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: myapp:latest
```

Возможные проблемы с CI/CD

Как выкатывать новую версию приложения?

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: myapp:{{ .Values.image.tag }}
```

> `image.tag=${CI_COMMIT_SHA}`

Helm

Helm

Причины использования

- Шаблонизация
- Откат
- Версионирование
- Переиспользование манифестов (в том числе публичных)

Helm

Возможности

- Упаковка нескольких манифестов Kubernetes в пакет - **Chart**
- Установка пакета в Kubernetes (установленный **Chart** называется **Release**)
- Шаблонизация во время установки пакета
- **Upgrade** (обновления) и **Rollback** (откаты) установленных пакетов
- Управление зависимостями между пакетами
- Хранение пакетов в удаленных репозиториях

Helm

Основные концепции

- Chart
- Values
- Release

Helm

Chart - пакет

- Метаданные
- Шаблоны описания ресурсов Kubernetes
- Конфигурация установки (**values.yaml**)
- Документация
- Список зависимостей

```
example/
  Chart.yaml          # описание пакета
  README.md
  requirements.yaml   # список зависимостей
  values.yaml         # переменные
  charts/             # загруженные зависимости
  templates/          # шаблоны
```

Helm

Chart.yaml - описание пакета

```yaml
apiVersion: v1
description: A Helm chart for ELK
home: https://www.elastic.co/products # Информация для community
name: elastic-stack # Название Chart
version: 1.7.0 # Версия Chart
appVersion: 6.0 # Версия приложения
maintainers: # Информация для community
- name: rendhalver
  email: pete.brown@powerhrg.com
- name: jar361
  email: jrodgers@powerhrg.com
- name: christian-roggia
  email: christian.roggia@gmail.com
```

Helm

Values - переменные

- Конфигурация для **Chart**
- Используются в шаблонах манифестов
- У пакета бывают значения по умолчанию
- Можно перезаписывать во время установки пакета
- Иерархическая структура

Helm

values.yaml

```
# Default values for elk.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

elasticsearch:
  enabled: true

kibana:
  enabled: true
  env:
    ELASTICSEARCH_URL: http://http.default.svc.cluster.local:9200
```

Helm

Release

- Установленный в Kubernetes **Chart**
- Хранятся в configMaps и Secrets
- **Chart** + **Values** = **Release**
- 1 **Upgrade** = 1 **Release**

Helm

Workflow

- Установка **Helm** на локальную машину (или на CI агента)
- **Для Helm 2:** Инициализация **Helm** в кластере Kubernetes (установка **Tiller**)
- Создание **Chart** или загрузка с Helm Hub
- Установка **Chart** в кластер Kubernetes (создание **Release**)
- Поддержание жизненного цикла приложения (upgrade, rollback, delete, etc.)

Helm

Workflow

```
$ helm install <chart_name> --name=<release_name> --namespace=<namespace> # Создание
release
$ kubectl get secrets -n <namespace> | grep <release_name>
sh.helm.release.v1.<release_name>.v1     helm.sh/release.v1    1    115m
```

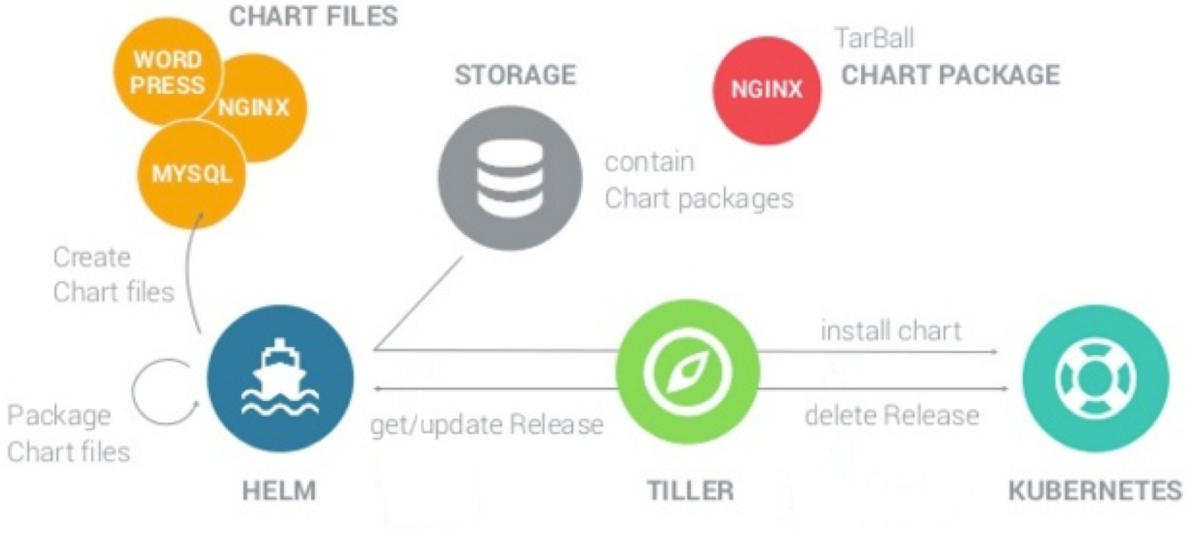
```
$ helm upgrade <release_name> <chart_name> --namespace=<namespace> # Обновление
release
$ kubectl get secrets -n <namespace> | grep <release_name>
sh.helm.release.v1.<release_name>.v1     helm.sh/release.v1    1    115m
sh.helm.release.v1.<release_name>.v2     helm.sh/release.v1    1    56m
```

```
$ helm upgrade --install <release_name> <chart_name> --namespace=<namespace> #
Создание или обновление release
$ kubectl get secrets -n <namespace> | grep <release_name>
sh.helm.release.v1.<release_name>.v1     helm.sh/release.v1    1    115m
sh.helm.release.v1.<release_name>.v2     helm.sh/release.v1    1    56m
sh.helm.release.v1.<release_name>.v3     helm.sh/release.v1    1    5s
```

Helm

Архитектура (v2)

- Helm client (**helm**) - CLI клиент на локальной машине
- **Tiller** server - сервер (pod) внутри кластера Kubernetes

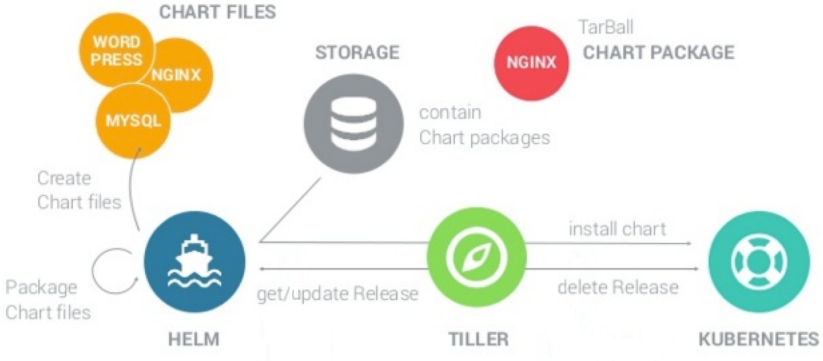


Helm v2

Helm Client

Используется для:

- Локальной работы с **Chart**
- Управления репозиториями
- Взаимодействия с **Tiller**
- Запроса:
  - Установки **Chart**
  - Обновления и удаления **Release**
  - Информации о текущих **Release**

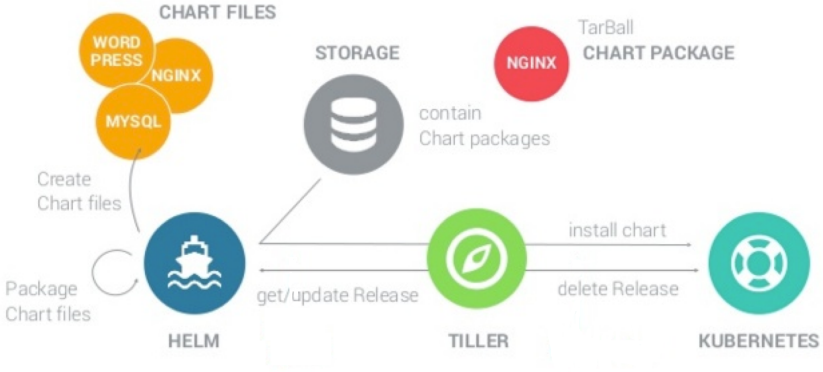


Helm v2

Tiller

Используется для:

- Ожидания входящих запросов от Helm Client
- Установки **Chart** в Kubernetes
- Обновления и удаления **Release**
- Взаимодействия с Kubernetes API



Хранит информацию о релизах в **configMaps** в соответствующем namespace

Helm v2

Tiller

Установка Tiller в кластер:

```
helm init
```

Установка Tiller с правами определенного service-account:

```
helm init --service-account=tiller
```

Проверка:

```
helm version
```

Helm 3

Что нового в Helm 3 (текущая версия - v3.3.0)

- Убран Tiller
- Release'ы теперь хранятся в namespace'ах (и namespace необходимо создавать)
- Зависимости указываются в Chart.yaml
- Экспериментальная поддержка OCI-совместимых registry для хранения Helm Charts
- Lua как альтернативный язык написания скриптов
- Переименованы некоторые команды клиента:
  - helm inspect -> helm show
  - helm fetch -> helm pull

Миграция v2 -> v3

Недостатки Helm

Helm 2: Tiller

В большинстве случаев устанавливается с правами `cluster-admin`

Как обойти проблему:

- Плагин **helm-tiller**
- Tiller per namespace
- Helm 3

> К чему может привести установка Tiller "самым простым путем"

Неприятные баги

Error: UPGRADE FAILED: "chart" has no deployed releases

Если первый release установился неуспешно - последующие также не будут установлены

- `helm upgrade --install --atomic` - при установке первого release
- `helm upgrade --install --force` - если первый release уже установлен
- `helm delete --purge && helm upgrade --install` - универсальный* способ (replaced with `helm uninstall`)
- **Helm 3** - появится команда `helm deploy` (сейчас нет)

Прочее

- Go/YAML Templating - на любителя
- Отступы (indent)
- `helm delete --purge` replaced by `helm uninstall`

> Полезные ссылки:
>
> - https://habr.com/ru/company/southbridge/blog/429340/
> - https://habr.com/ru/company/flant/blog/438814/

Templating

Templating | Манифест

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.name }}
  labels:
    app: nginx
spec:
  replicas: {{ .Values.nginx.replicas }}
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: {{ .Values.nginx.image.repository }}:{{ .Values.nginx.image.tag }}
        ports:
        - containerPort: 80
```

Templating

Встроенные переменные

- **Release** - информация об устанавливаемом release
- **Chart** - информация о chart, из которого происходит установка
- **Files** - возможность загружать в шаблон данные из файлов (например, в `configMap`)
- **Capabilities** - информация о возможностях кластера (например, версия Kubernetes)
- **Templates** - информация о манифесте, из которого был создан ресурс

> Подробное описание доступно по ссылке

Templating

Определяемые переменные

Задаются в файле **values.yaml**

```
1 nginx:
2   replicas: 1
3   image:
4     repository: my.private.registry/nginx
5     tag: latest
```

Helm

Workflow

Создание release с использованием переменных из файла `values.yaml`:

```
$ helm upgrade --install <release_name> <chart_name> -f values.yaml
```

Переопределение переменной в момент установки:

```
$ helm upgrade --install <release_name> <chart_name> -f values.yaml --set
nginx.image.tag=${CI_COMMIT_SHA}
```

> Рекомендуется использовать ключ `--wait` при создании или обновлении release. Подробное описание ключа

Templating

В основе Helm лежит шаблонизатор Go с 50+ встроенными функциями (документация)

Условия:

```
{{- if .Values.server.persistentVolume.enabled }}
    persistentVolumeClaim:
      ...
{{- else }}
```

Циклы:

```
{{- range $key, $value := .Values.server.annotations }}
  {{ $key: }} {{ $value }}
{{- end }}
```

Templating

Более сложный пример

```
{{- $files := .Files.Glob "dashboards/*.json" }}
{{- if $files }}
apiVersion: v1
kind: ConfigMapList
items:
{{- range $path, $fileContents := $files }}
{{- $dashboardName := regexReplaceAll "(^.*/)(.*)\\.json$" $path "${2}" }}
- apiVersion: v1
  kind: ConfigMap
  metadata:
    name: {{ $dashboardName | trunc 63 | trimSuffix "-" }}
    labels:
      grafana_dashboard: "1"
  data:
    {{ $dashboardName }}.json: {{ $.Files.Get $path | toJson }}
{{- end }}
{{- end }}
```

Полезные советы

- Указывайте все используемые в шаблонах переменные в `values.yaml`, выбирайте адекватные значения по умолчанию
- Используйте команду `helm create` для генерации структуры своего chart
- Пользуйтесь плагином **helm docs** для документирования своего chart

> Больше советов по написанию helm charts от разработчиков https://helm.sh/docs/chart_best_practices/

Хранение Helm Charts

В репозитории с приложением

- Удобно если манифесты пишут разработчики продукта
- Манифесты версионируются по тому же flow что и приложение
- Самый простой способ

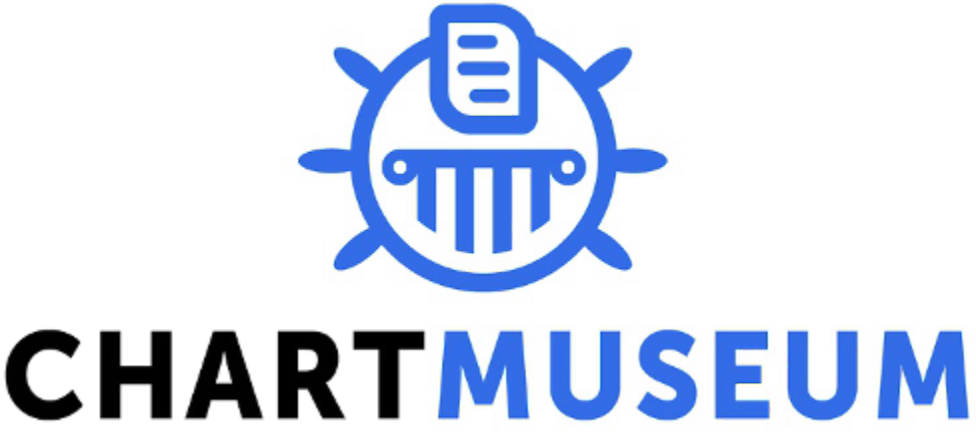
Внешний репозиторий

GitHub/GitLab/etc, либо хранилища артефактов

- Удобно если манифесты пишет выделенная команда (Infrastructure team)
- Проще переиспользовать наработки
- Полезно если есть необходимость в meta pipelines
- Сложнее привязать flow работы с кодом продукта к инфраструктурному коду (манифестам)

Хранилища артефактов

- **ChartMuseum**
- **Harbor** (внутри - ChartMuseum)
- **Artifactory**
- ...



git

- Необходим `index.yaml` файл с описанием всех chart в репозитории
- Необходима сборка в tgz перед публикацией (в идеале tgz архивы должны храниться вне репозитория)

```
helm package charts/
helm repo index .
helm-docs charts/
```

Пример репозитория

Helm Hooks

Hooks

Определенные действия, выполняемые в различные моменты жизненного цикла поставки. Hook, как правило, запускает Job (но это не обязательно).

Виды hooks:

- pre/post-install
- pre/post-delete
- pre/post-upgrade
- pre/post-rollback

> Документация

Hooks | Описание

Обычный манифест, но требуется добавить annotations.

Например, **pre-install** hook:

```
apiVersion: ...
kind: ....
metadata:
  annotations:
    "helm.sh/hook": "pre-install"
```

Hooks | Пример из жизни

Для развертывания продукта из feature branch во временное окружение (namespace) необходимо сделать следующие действия:

1. Добавить в namespace секреты для подключения к внешнему инстансу БД
2. Создать в инстансе новую БД
3. Выкатить миграции для БД (база используется более чем одним компонентом)
4. Установить Helm Chart с приложением

Hooks | Weight

Позволяют определить порядок выполнения одинаковых типов hooks

```
apiVersion: ...
kind: ....
metadata:
  annotations:
    "helm.sh/hook": "pre-install"
    "helm.sh/hook-weight": "10"
```

Чем меньше вес - тем раньше будет выполнен hook 🤷‍♂️

Hooks | Удаление

После выполнения hook можно (и в большинстве случаев нужно) удалять артефакты после него (например - **job**). Подробнее

Для этого придумана специальная annotation - `"helm.sh/hook-delete-policy"` Могут использоваься следующие значения:

- **"hook-succeeded"**
- **"hook-failed"**
- **"before-hook-creation"**

```
apiVersion: ...
kind: ....
metadata:
  annotations:
    "helm.sh/hook": "pre-install"
    "helm.sh/hook-weight": "10"
    "helm.sh/hook-delete-policy": "hook-succeeded"
```

Практика helm_1

Удалим все существующие ui ресурсы

```
kubectl delete deployment ui
kubectl delete svc ui
kubectl delete ingress ui
```

Полезные команды

```
# Показыват список установленных релизов
helm ls --namespace=<namespace>

# Показать статус релиза
helm status ui --namespace=<namespace>

# Удалить установленный релиз
helm delete <release name> --namespace=<namespace>

# Рендер шаблонов чарта или конкретного шаблона
helm template chart/
```

Проверка Helm

Убедимся, что установлен Helm последней версии:

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash
```

```
helm version
version.BuildInfo{Version:"v3.1.1",
GitCommit:"afe70585407b420d0097d07b21c47dc511525ac8", GitTreeState:"clean",
GoVersion:"go1.13.8"}
```

Так как мы работаем с `3` версией `Helm`, никаких элементов не нужно инициализировать в кластере.

Сразу за работу! 👷

Установка community чарта

Сегодня мы уже работали с `Helm3`, давайте вспомним что делали 💪

Руками добавляли `stable` репозиторий:

```
$ helm repo list
Error: no repositories to show


$ helm repo add stable https://charts.helm.sh/stable
$ helm repo update
"stable" has been added to your repositories
```

Выполняли поиск уже существующего чарта `nginx-ingress` в `stable` репозитории и устанавливали его:

```
$ helm search repo nginx
NAME                    CHART VERSION   APP VERSION   DESCRIPTION
stable/nginx-ingress    1.33.0          0.30.0        An nginx Ingress controller that uses
ConfigMap...

$ helm upgrade --install nginx-ingress stable/nginx-ingress --namespace=nginx-
ingress
```

Упаковка UI в Chart

- В `~/project/ui` выполнить `helm create chart`
- В `chart/Chart.yaml` поправьте название с `chart` на `ui`
- Удалить заготовки шаблонов `rm -rf ~/project/ui/chart/templates/*.yaml`
- Скопировать текущие ресурсы из `kubernetes/` в `chart/templates/`

Values services

`values.yaml` уже содержит набор параметров. Можно их удалить и написать свои с нуля, можно дополнить существующие.

```
replicaCount: 3 # выставляем 3

labels:
  app: reddit
  component: ui

image:
  repository: gcr.io/cd-k8s-236617/user-<your number>-ui
  tag: v0.0.1

service:
  type: NodePort
  port: 9292
  targetPort: 9292
```

> ❗ Укажите правильный тег образа

Deployment.yaml

В `templates/deployment.yaml` нужно заменить:

- `name: ui` должен быть заменен на переменную `.Chart.Name`
- `labels` на перебор значений переменной `.Values.labels`
- В качестве `image` использовать сочетание переменных `.Values.image.repository` и `.Values.image.tag`
- в `replicas` использовать переменную `.Values.replicasCount`

> Подсказка: Для подстановки .Values.labels в labels манифеста, можно использовать цикл, как на слайде Go templating

Deployment.yaml

Заменяем `name: ui` на переменную

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Chart.Name }}
```

Заменяем `labels` на перебор значений переменной

```
labels:
  {{- range $key, $value := .Values.labels }}
    {{ $key }}: {{ $value }}
  {{- end }}
```

Deployment.yaml

В качестве `image` используем сочетание переменных

```
image: {{ .Values.image.repository }}:{{ .Values.image.tag }}
```

В `replicas` используем переменную `.Values.replicasCount`

```
spec:
  replicas: {{ .Values.replicaCount }}
```

Устанавливаем пакет

```
# установка пакета
helm upgrade --install ui chart/ # namespace не указываем, будет установлен в
default

# посмотреть все релизы
helm ls

# сохраняем изменения
git add . && git commit -m 'create helm chart'
git push origin master
```

Поставка Chart'a

В `.gitlab-ci.yml`, в стадии `deploy` надо заменить

```
- kubectl apply -f kubernetes/
```

на следующие команды:

```
- curl -L https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3  |
bash
- helm upgrade ui -i chart/
```

Было

```
deploy:
  stage: deploy
  script:
    - apk add -U openssl curl tar gzip bash ca-certificates git
    - curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-
release/release/stable.txt)/bin/linux/amd64/kubectl
    - chmod +x ./kubectl && mv ./kubectl /usr/local/bin/kubectl
    - mkdir -p ~/.kube/ && echo $KUBE_CONFIG | base64 -d > ~/.kube/config
    - kubectl cluster-info
    - kubectl apply -f kubernetes/
```

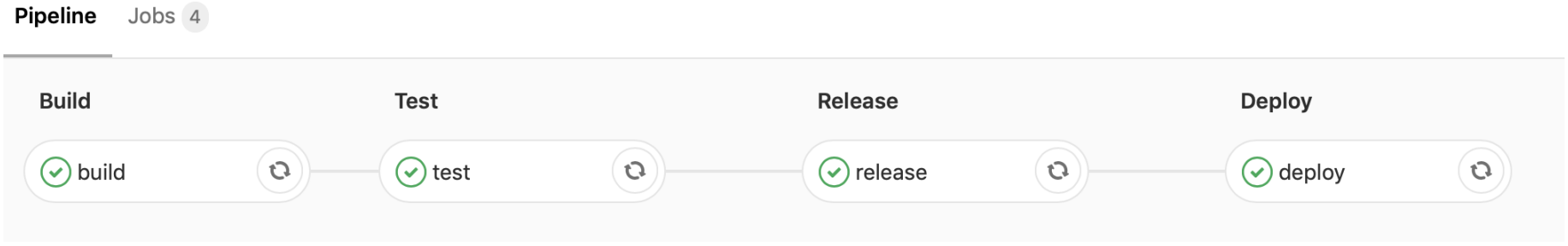
Стало

```
deploy:
  stage: deploy
  script:
    - apk add -U openssl curl tar gzip bash ca-certificates git
    - curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-
release/release/stable.txt)/bin/linux/amd64/kubectl
    - chmod +x ./kubectl && mv ./kubectl /usr/local/bin/kubectl
    - mkdir -p ~/.kube/ && echo $KUBE_CONFIG | base64 -d > ~/.kube/config
    - kubectl cluster-info
    - curl -L https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
| bash
    - helm upgrade ui -i chart/
```

Сохраняем изменения

```
git add .
git commit -m 'Use helm for deploy'
git push origin master
```

Проверяем, что пайплайн зеленый.



Шаблонизируем service и ingress

- Шаблонизировать `service.yaml` и `ingress.yaml` так же, как `deployment.yaml`.
- Запушить код
- Посмотреть информацию о релизе через `helm status ui` или в пайплайне
- **Убедиться, что приложение все еще работает :)**

Шаблонизируем service и ingress

В service.yaml не забыть заменить `port: 9292` и `targetPort: 9292`

```
port: {{ .Values.service.port }}
targetPort: {{ .Values.service.targetPort }}
```

В ingress.yaml не забыть заменить `servicePort: 9292`

```
servicePort: {{ .Values.service.port }}
```

Какая проблема осталась в пайплайне?

Деплой с тегом

Добавить `tag` в `.gitlab-ci.yml`

```
- helm upgrade ui -i --set-string image.tag=$CI_COMMIT_SHORT_SHA chart/
```

Запушьте изменения и, после прохождения пайплайна убедитесь, что было задеплоен образ с тегом коммита. Тег должен быть виден в поле `Image` сущности `Deployment`

```
kubectl describe deployment ui
```

```
…
Containers:
   ui:
    Image:        gcr.io/cd-k8s-236617/user00-ui:32e53330
…
```

Повторить для comment

Создайте `chart` для `comment`, добавьте установку `chart` в пайплайн поставки.

Базу данных тоже нужно параметризировать, например, так:

```
db:
  name: post-db
  replicasCount: 1
  labels:
    app: reddit
    component: post-db
  …
```

Эти значения будут доступны как `.Values.db.*`

> P.S. Перед описанием `chart`ов для `comment` и `post` нужно удалить их манифесты из кластера через `kubectl delete …`

Отличия от ui

- Подставьте в `values.yaml` подставить требуемые `labels`:
  - а именно `app: reddit` и `component: comment`
- Укажите в качестве репозитория `gcr.io/cd-k8s-236617/user-<ваш номер>-comment`

  А все остальное - аналогично сервису **ui**

Повторите для post

Самостоятельно повторите для **post**

В целом, все так же, только не забудьте подставить корректный порт (у comment это `5000`)

Задания со ⭐

Выносим базу данных в зависимости чартов `comment` и `post`

- Создайте репозиторий `db`
- Опишите в нем chart `db`, вынесите в него шаблоны запуска `mongodb` из `comment`/`post`
- Опишите chart `db` как зависимость в `requirements.yaml` через `file://...`
- Параметризируйте `db` через `values` чартов
- На шаге `deploy` в `.gitlab-ci.yml` добавьте команды `git clone <репозиторий с чартом db>` и `helm dep update`