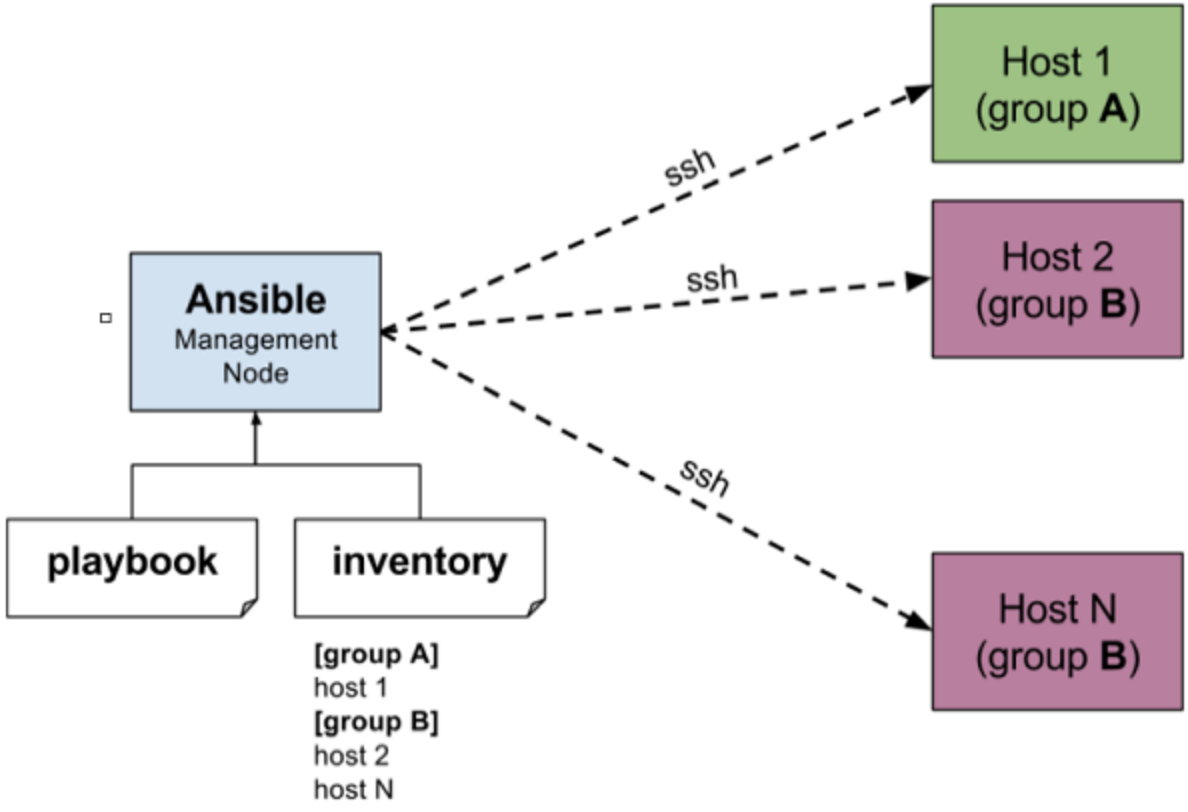
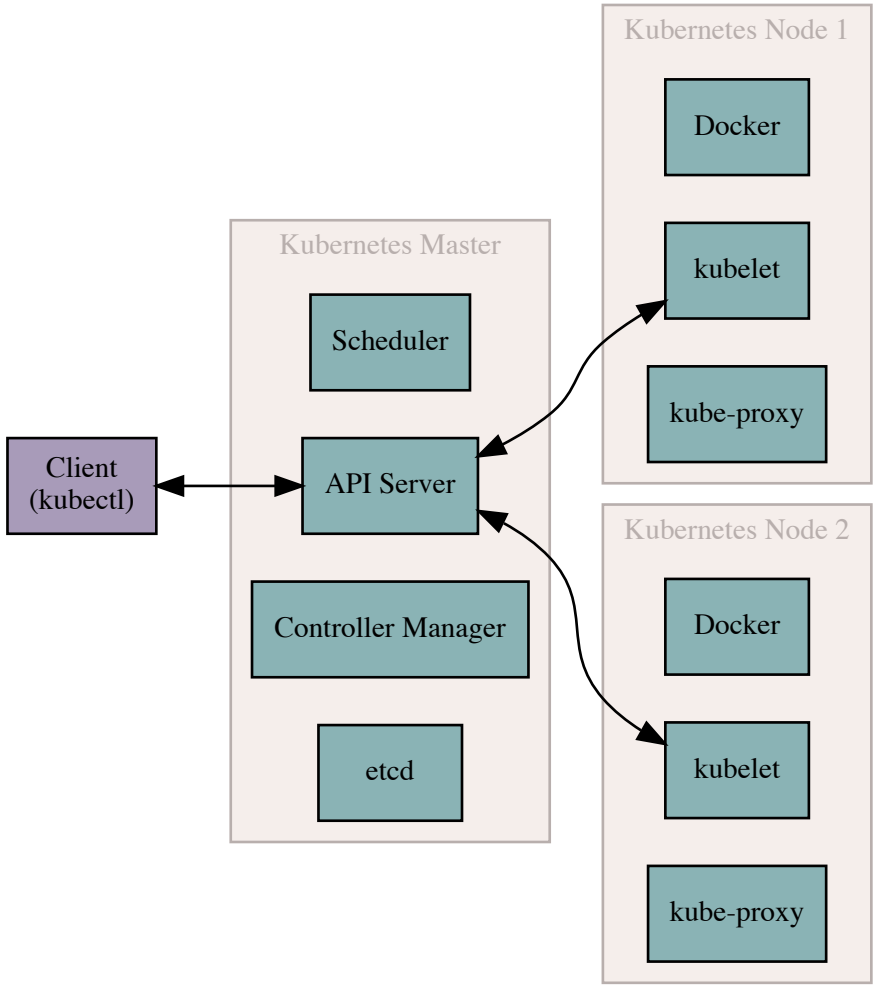
CI/CD в Apliteni

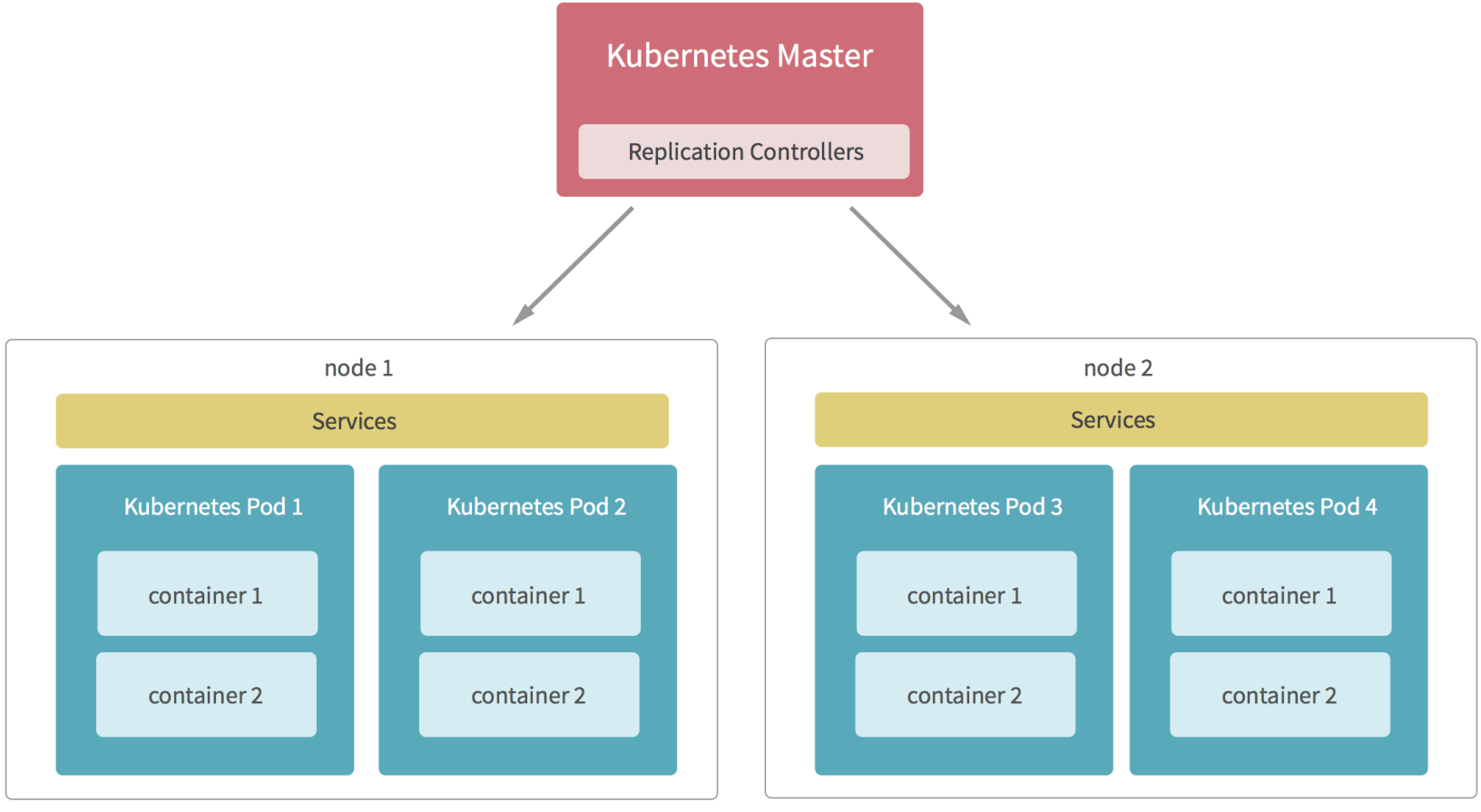
Как работает Ansible



Как работает Kubernetes



Слой Kubernetes



Когда использовать Ansible

- Изолированный проект
- Редкие деплои (1-2 раза в год)
- Нужна высокая скорость чтения-записи на диск

Когда использовать Kubernetes

- Во всех остальных случаях

Подключение к кластеру K8s

```
kubectl config set-credentials cluster-user \
        --client-certificate=~/.kube/user.crt \
        --embed-certs=true    # импортировать в конфиг
kubectl config set-cluster hz1 --server=https://1.2.3.4:6443
kubectl config set-context hz1 --user=cluster-user
```

Проверяем

```
$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
    server: https://1.2.3.4:6443
  name: hz1
```

```
$ kubectl cluster-info
Kubernetes master is running at https://1.2.3.4:6443
KubeDNS is running at https://1.2.3.4:6443/api/v1/namespaces/ku
```

```
$ kubectl get nodes
NAME              STATUS    ROLES     AGE       VERSION
hz1-master-01   Ready     master    20d       v1.10.2
hz1-worker-02   Ready     <none>    20d       v1.10.2
hz1-worker-03   Ready     <none>    20d       v1.10.2
hz1-worker-04   Ready     <none>    20d       v1.10.2
```

Деплой в K8s

```
$ kubectl apply -f app/ --namespace=app
pod "app" created
service "app" created
ingress "app" created
```

Содержимое app/

```
app/
    deployment.yaml
    service.yaml
    ingress.yaml
```

Пример spec'a

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: appx-deployment
spec:
  replicas: 2
  selector:
    template:
      metadata:
        labels:
          app: appx
    spec:
      containers:
      - name: appx
        image: apliteni/appx
        ports:
        - containerPort: 80
```

Версия спецификации

```
apiVersion: apps/v1
```

Тип объекта

```
kind: Deployment
```

Другие виды объектов:

Рабочие объекты: Pod, ReplicaSet, StatefulSet, JobCron

Discovery и балансировка: Service, Ingress

Конфиги: ConfigMap, Secret

Хранилище: StorageClass, PersistentVolumeClaim, Volume

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.10/

Детали объекта

```
spec:
  replicas: 2 # количество реплик
  selector:
    template:
      metadata:
        labels:
          app: appx   # ставит метку 'app' со значением 'appx'
    spec:
      containers:
      - name: appx     # запустить docker-контейнер
        image: apliteni/appx      # образ с dockerhub
        ports:
        - containerPort: 80
```

Недостатки выкатки через kubectl

- Все объекты в одну кучу
- Нет поддержки переменных

Helm

```
helm install app-chart/ --namespace=app --name=app-release
```

Helm Chart

```
app-chart/
    Chart.yaml
    values.yaml
    templates/
         deployment.yaml
         service.yaml
         ingress.yaml
```

Пример Chart.yaml

```yaml
apiVersion: v1
version: 0.1.1    # Версия chart'а, не приложения
description: Management system for GDBC
name: gdbc-management-api
```

Пример values.yaml

```yaml
resources:
  requests:
    memory: 512Mi
    cpu: 300m
    storage: 1Gi

image:
  repository: registry.domain.com:9999
  name: apliteni/gdbc/gdbc-management
  tag: ""
  pullPolicy: IfNotPresent

service:
  port: 80
  logLevel: DEBUG
```

Пример шаблона deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ template "project.fullname" . }}
  labels:
    name: {{ template "project.fullname" . }}
    ...
spec:
  replicas: {{ .Values.deploymnent.replicas }}
  selector:
    matchLabels:
      name: {{ template "project.fullname" . }}
  template:
    metadata:
      labels:
        name: {{ template "project.fullname" . }}
      containers:
        - name: {{ template "project.name" . }}
          image: "{{ .Values.image.repository }}/{{ .Values.ima
```

Хелперы

./app-chart/templates/_helpers.tpl

```
{{- define "project.name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimS
{{- end -}}
```

./app-chart/templates/deplyment.yaml

```
metadata:
  name: {{ template "project.name" . }}
```

Недостаки Helm

- Нельзя деплоить сразу в несколько namespace'ов

Что дает Helm

- Поддержка gotemplates
- Отслеживание состояния релиза `helm status RELEASE`
- Встроенное тестирование релиза `helm test RELEASE`
- Откат релиза `helm rollback`
- Наличие готовых chart'ов

Правильный деплой

```
git push origin master
```

Как он достигается?

- Dockerfile
- Helm chart
- gitlab-ci.yml

Пример проекта

```
app/
  .gitlab-ci.yaml
  helm/
    templates/...
    Chart.yaml
    values.yaml
  Dockerfile.pre_build
  Dockerfile.final_build
  ...
```

Контейнеризация

- Один проект - один образ (артефакт)
- Все настройки пробрасываются через ENV параметры
- Версионирование приложения и образа

ENV параметры

```
apiVersion: apps/v1
kind: Deployment
...
spec:
  template:
    spec:
      containers:
        - name: appx
          env
            - name: LOG_LEVEL
              value: {{ .Values.service.logLevel }}
```

```
import "os"

func getLogLevel() bool {
        return os.Getenv("LOG_LEVEL")
}
```

Версионирование

- COMMIT SHA
- Файл VERSION
- В исходном коде (например, в version.go)

Helm Chart

- Описание приложения для развертки в kubernetes
- Настройки в `values.yaml` или по окружениям `values.dev.yaml`, `values.staging.yaml`.

.gitlab-ci.yml

Описание стадий: `pre_build`, `test`, `build`, `deploy`, `test_release`

Авторизация в приватном Registry

```
before_script:
    - docker login
      -u gitlab-ci-token
      -p ${CI_JOB_TOKEN} ${CI_REGISTRY}
```

Стадия pre-build

```
build:
  stage: build
  script:
    - export APP_VERSION=app-${CI_COMMIT_SHA}
    - docker build --pull
      -t ${PRE_BUILD_IMAGE}
      --build-arg APP_VERSION=${APP_VERSION} # Версия
      --build-arg DATE=${DATE}
      --build-arg PROJECT_PATH=${PROJECT_PATH}
      .
      -f Dockerfile
    - docker push ${PRE_BUILD_IMAGE}
```

Пример Dockerfile.pre_build

```
FROM golang:1.9

ARG PROJECT_PATH
ARG APP_VERSION

RUN mkdir -p $PROJECT_PATH
WORKDIR $PROJECT_PATH

RUN go get -u github.com/golang/dep/...

COPY Gopkg.toml Gopkg.lock ./

RUN dep ensure -vendor-only

COPY ./ ./

RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo \
                -ldflags "-X \"main.version=${APP_VERSION}\" -X
                -o server .
```

Стадия test

```
test:
  stage: test
  script:
    - docker pull ${PRE_BUILD_IMAGE}
    - docker run -w ${PROJECT_PATH} ${REGISTRY_BUILD_IMAGE}
        make test
```

Интеграционные тесты (DinD)

```
test:
  stage: test
  tags:
    - docker        # запуск на runner'е с docker
  services:         # поднимет сервисы в контейнерах
    - postgres:9.8
    - redis:2.5
  script:
    - ping postgres
```

В логе в runner job:

```
$ ping postgres
PING postgres (10.2.2.2): 56 data bytes
64 bytes from 10.2.2.2: icmp_seq=0 ttl=53 time=95.219 ms
```

Стадия build

```
build:
  stage: build
  script:
    # Запуск prebuild-образа в контейнере
    - docker container create --name extract ${PRE_BUILD_IMAGE}

    # Вытягиваем бинарник из prebuild-образа
    - docker container cp extract:${PROJECT_PATH}/server ./serv
    - docker container rm -f extract

    # Собираем новый образ
    - docker build --no-cache
      -t ${CI_REGISTRY_IMAGE}:latest
      -t ${CI_REGISTRY_IMAGE}:${APP_VERSION}
      .
      -f Dockerfile.final_build
    - docker push ${CI_REGISTRY_IMAGE}
```

Почему не multi-stage из Docker?

Для тестов нужны исходники

Стадия deploy

```
deploy:
  stage: deploy
  image: "${CI_REGISTRY}/infra/kubernetes-helm"
  tags:
    - docker
  only:
    - master
  environment:
    name: production
  script:
    - helm init --client-only

    - export DEPLOYS=$(helm ls | grep ${CI_PROJECT_NAME} | wc -

    - if [ ${DEPLOYS}  -eq 0 ]; then
        helm install --name ${CI_PROJECT_NAME} --namespace=${NA
      else
        helm upgrade ${CI_PROJECT_NAME} --namespace=${NAMESPACE
      fi
```

Helm install / upgrade

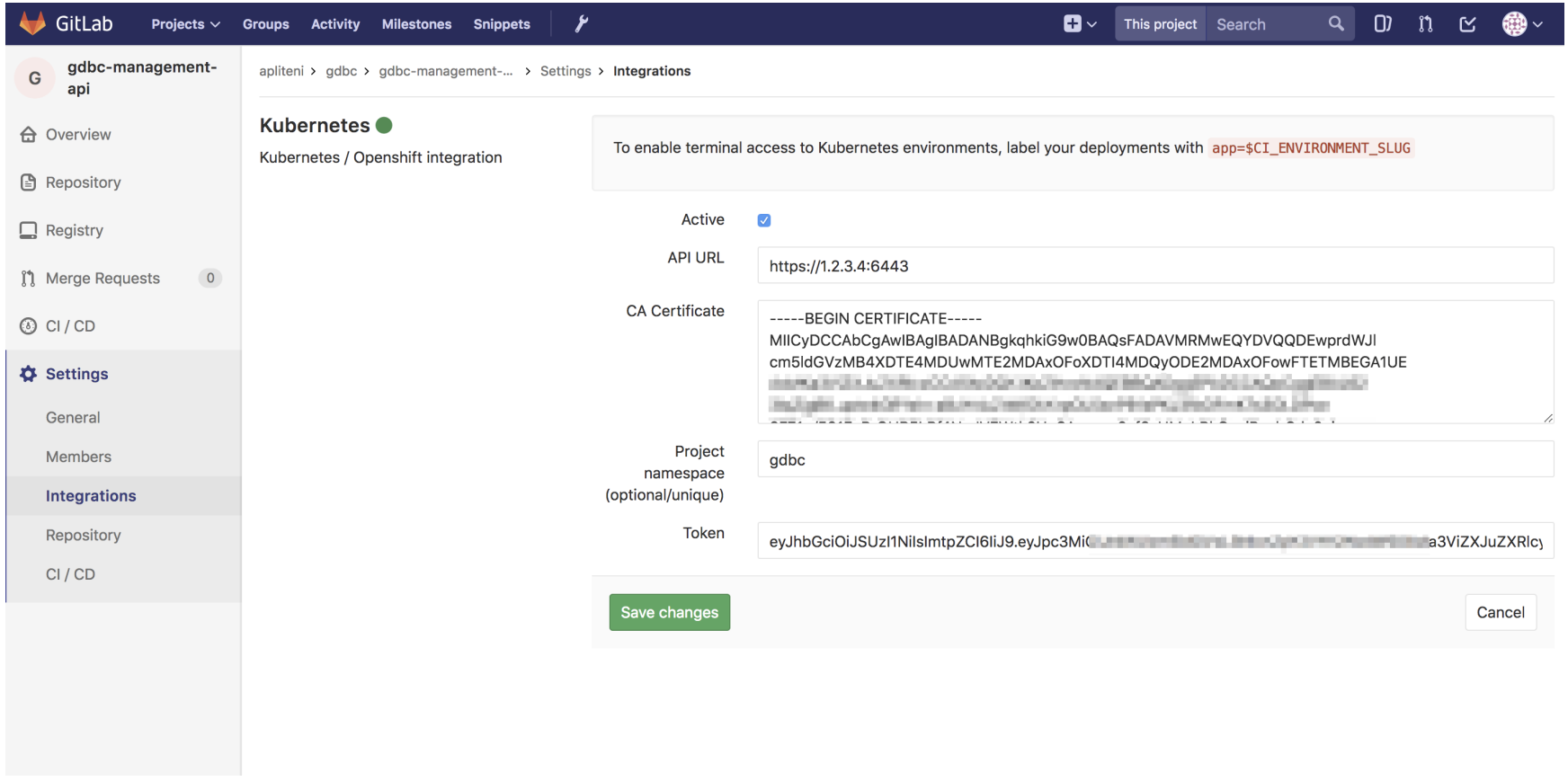
Есть баг с `helm upgrade --install`

https://github.com/kubernetes/helm/issues/3353

Обходное решение:

```
script:
  - export DEPLOYS=$(helm ls | grep ${CI_PROJECT_NAME} | wc -l)
  - if [ ${DEPLOYS}  -eq 0 ]; then
       helm install ...
    else
        helm upgrade ...
    fi
```

Авторизация CI/CD в кластере



Настройка Environment

```
deploy:
  ...
  environment:
    name: staging
```

Стадия test release

```
test_release:
  stage: test_release
  image: hypnoglow/kubernetes-helm
  tags:
    - docker
  only:
    - master
  environment:
    name: staging
  script:
    - helm init --client-only

    # Удалит старый pod с тестом, если он есть
    - ! kubectl delete pod ${CI_PROJECT_NAME}-test --namespace=

    # Запуск pod'а с тестом
    - helm test ${CI_PROJECT_NAME}
```

Пример теста для Helm

./helm/templates/tests/test-app.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: "{{ template "project.fullname" . }}-test"
  labels:
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version }}
    app: {{ template "project.name" . }}
  annotations:
    "helm.sh/hook": test-success
spec:
  containers:
    - name: curl
      image: radial/busyboxplus:curl
      command: ["/bin/sh","-c"]
      args: ['curl -s "{{ template "project.fullname" . }}:{{ .
  restartPolicy: Never
```

Как работает тест

- Запускает pod с контейнером
- Выполняет command

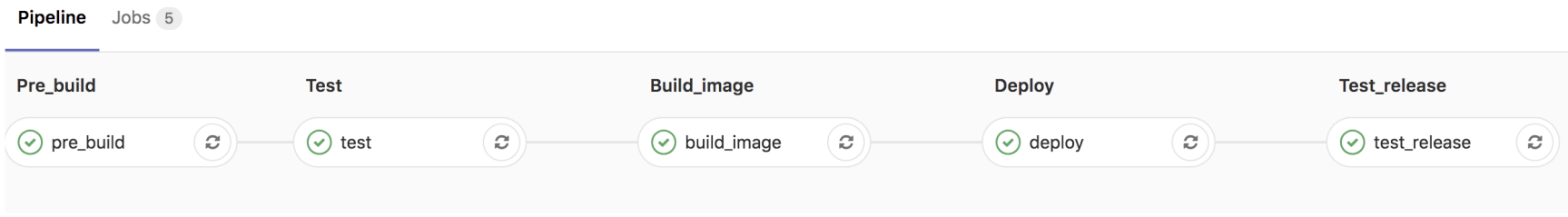
```
command: ['command']
```

- С параметрами:

```
args: ['...']
```

- Тест падает, если:
  - ошибка в `stderr`
  - exit(1) или exit(2)

Результат работы CI/CD




```
$ helm status gdbc-management-api
LAST DEPLOYED: Mon May 21 23:54:28 2018
NAMESPACE: gdbc
STATUS: DEPLOYED

RESOURCES:
==> v1/Service
NAME                TYPE       CLUSTER-IP      EXTERNAL-IP  PORT(S)  AGE
gdbc-management-api  ClusterIP  10.105.109.220  <none>       80/TCP   2d

==> v1/Deployment
NAME                DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
gdbc-management-api  1        1        1           1          2d

==> v1beta1/Ingress
NAME                HOSTS               ADDRESS  PORTS  AGE
gdbc-management-api [redacted]          80       2d

==> v1/Pod(related)
NAME                                READY  STATUS   RESTARTS  AGE
gdbc-management-api-5ff7c97f54-5v4xk  1/1    Running  0         12m


TEST SUITE:
Last Started: Mon May 21 23:54:45 2018
Last Completed: Mon May 21 23:54:48 2018

TEST                          STATUS          INFO    STARTED                          COMPLETED
gdbc-management-api-test      SUCCESS                 Mon May 21 23:54:45 2018         Mon May 21 23:54:48 2018
```

Текущие проблемы

- DinD не кэширует образы, следовательно, медленный https://gitlab.com/gitlab-org/gitlab-ce/issues/17861

Почитать

- https://docs.docker.com/engine/reference/builder/
- https://kubernetes.io/docs/home/
- https://docs.helm.sh/developers/
- https://docs.gitlab.com/ee/ci/yaml/

Посмотреть

- https://www.youtube.com/playlist?list=PL8QZN5SAvhBMb4aNj4GZ7-b3efWkuTYYG