KUB 35: Настройка CI/CD в Gitlab для Kubernetes

Описание:

Организация процесса CI/CD — это комплексная задача, которая ставит ряд проблем, на текущий момент, решаемых с помощью разных инструментов. Перед тем как начать решать ее, следует определить исходные данные:

1. Уникальные характеристики проекта:
- является ли код адаптированным для контейнеризации? Монолиты vs микросервисы;
- требования команды — одним проектам достаточно test/stage и prod окружения, другим необходимы динамические окружения, собственные песочницы для каждого сотрудника;
- сложность конвейера доставки кода, требования к организации тестов, глубина тестов и т.д.;
- вопросы безопасности, разграничения прав и т.д.;
2. Общие требования к инструментам:
- open source vs proprietary software;
- масштабируемость и совместимость;
- сложность эксплуатации.

Ответив на указанные вопросы, вы, может быть, сразу определитесь со стеком инструментов, который будет использоваться для решения поставленной задачи. В нашем случае для большей части вопросов (исключая enterprise-проекты) мы используем связку Kubernetes + Helm + Gitlab + Docker.

Если коротко, то на этапе сборки кода и хранения images мы используем docker + gitlab container registry. На этапе тестирования — gitlab-ci и заранее подготовленные images с необходимым для тестирования окружением. На этапе деплоя — gitlab-ci + helm для версионирования, шаблонизации, возможности rollback и иных механизмов, неразрывно связанных с обслуживанием жизненного цикла кодовой базы.

Разумеется, основой указанной схемы является gitlab. Для наших проектов мы предпочитаем собственные инсталляции gitlab, ответственность за обслуживание которых полностью берем на себя. Это разумное решение, если учитывать вопросы безопасности и самодостаточности. Изучение всех механизмов gitlab-ci — тема, которая выходит за рамки текущего практикума, так что данное задание мы сделаем модульным.

Давайте представим, что у нас есть проект, написанный на любом языке программирования. Представим его структуру в гит-репозитории:

```
./ - корневая директория проекта
  infra/ — директория для хранения helm-чарта и его настроек
  infra/values.dev.yaml — файл с настройками для dev-окружения
```

```
  infra/secrets.dev.yaml — файл с секретными данными для
dev-окружения
  infra/chart/ — директория с чартом нашего приложения
  Dockerfile — докер-файл, который будет использоваться для
сборки нашего приложения
  .gitlab-ci.yaml — файл с описанием процесса CI/CD
```

Итак, давайте приведем пример настройки gitlab-ci.yaml:

```
image: docker:19.03.12

variables:
  DOCKER_TLS_CERTDIR: "/certs"

services:
  - docker:19.03.12-dind

stages:
  - build
  - deploy

build:
  stage: build
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
$CI_REGISTRY
    - docker build -t
$CI_REGISTRY/$CI_PROJECT_NAMESPACE/$CI_PROJECT_NAME:$CI_COMMIT_S
HORT_SHA .
    - docker push
$CI_REGISTRY/$CI_PROJECT_NAMESPACE/$CI_PROJECT_NAME:$CI_COMMIT_S
HORT_SHA

.deploy_template:
  stage: deploy
  image: registry.gitlab.com/fevlake_ops/helm-deployer:latest
  script:
    # Setup Kubeconfig
    - mkdir -p $HOME/.kube
    - cat $KUBECONFIG > $HOME/.kube/config
    # Setup GPG for Helm secrets
    - gpg --allow-secret-key-import --import $HELM_SECRET
```

```
    # Deploy via Helm
    - cd .infra/
    - helm secrets upgrade --wait --install service --namespace
${CI_ENVIRONMENT_NAME} --values
secrets.${CI_ENVIRONMENT_NAME}.yaml --set
image.tag=${CI_COMMIT_SHORT_SHA} chart/

dev deploy:
  extends: .deploy_template
  environment:
    name: dev
  when: manual

prod deploy:
  extends: .deploy_template
  environment:
    name: prod
  when: manual
```

В данном примере мы сначала собираем контейнер и сохраняем его в registry, после этого запускаем helm с основными параметрами для разворачивания приложения в kubernetes кластер. Из интересного здесь стоит отметить, что мы берем секретный ключ и kubeconfig из переменных проекта (в гитлаб) и сохраняем их локально при деплое. Они будут использоваться для расшифровки секретов, а так же для подключения к kubernetes кластеру.
И еще один важный момент - мы используем специальный образ helm deployer, который является alpine линукс с установленными helm, kubectl и другими системными утилитами. Собирается он простым dockerfile:

```
FROM alpine:3.9

ARG KUBECTL_VERSION=v1.19.2
ARG HELM_VERSION=3.3.1
ARG KUBEDOG_VERSION=v0.4.0

ENV KUBECTL_VERSION=$KUBECTL_VERSION
ENV HELM_VERSION=$HELM_VERSION
ENV HELM_HOME=/helm/
ENV YC_HOME=/yc

ENV PATH $HELM_HOME:$YC_HOME/bin:$PATH

RUN apk --no-cache add \
```

```
        curl \
        python \
        py-crcmod \
        bash \
        libc6-compat \
        openssh-client \
        git \
        gnupg

RUN curl -LO
https://storage.googleapis.com/kubernetes-release/release/${KUBE
CTL_VERSION}/bin/linux/amd64/kubectl && \
    chmod +x ./kubectl && \
    mv ./kubectl /usr/local/bin/kubectl

RUN curl -O
https://get.helm.sh/helm-v${HELM_VERSION}-linux-amd64.tar.gz &&
\
    tar xzf helm-v${HELM_VERSION}-linux-amd64.tar.gz && \
    mv linux-amd64 $HELM_HOME && \
    rm helm-v${HELM_VERSION}-linux-amd64.tar.gz && \
    mkdir -p $HELM_HOME/plugins && \
    helm plugin install
https://github.com/futuresimple/helm-secrets

RUN curl
https://storage.yandexcloud.net/yandexcloud-yc/install.sh | \
    bash -s -- -i ${YC_HOME} -n

RUN apk add --no-cache python3 py3-pip \
    && pip3 install --upgrade pip \
    && pip3 install awscli \
    && rm -rf /var/cache/apk/*

RUN curl -L -o /usr/local/bin/kubedog
https://dl.bintray.com/flant/kubedog/${KUBEDOG_VERSION}/kubedog-
linux-amd64-${KUBEDOG_VERSION} && \
    chmod +x /usr/local/bin/kubedog

VOLUME ["/root/.config"]
```

Задание

1. Создайте репозиторий в группе https://gitlab.rebrainme.com/kubernetes_users_repos/<your_gitlab_id>/
2. Склонируйте код приложения расположенный в репозитории и сохрание его в созданный репозиторий в п.1
3. Настройте gitlab-ci в созданном репозитории:

- необходимо собирать образ контейнера и размещать его в Gitlab Registry.
- приложение из собранного контейнера необходимо деплоить в Kubernetes кластер.
- Деплой должен происходить с использованием сервисного аккаунта.
- Деплой так же должен происходить с использованием helm.

4. Приложение должно:

- деплоится в namespace test
- деплоймент должен называться application
- сервис для доступа к приложению должен называться app-svc
- образ контейнера для приложения должен запрашиваться с registry.rebrainme.com

5. После выполнения задания при формировании ответа добавьте вывод команд kubectl, который сможет подтвердить создание всех необходимых по заданию объектов kubernetes в назначенном по заданию namespace. Также добавьте в ответ ссылку на ваш репозиторий.
6. Отправьте задание на проверку куратору через кнопку ("Задать вопрос куратору").