Если вы видите что-то необычное, просто сообщите мне.

Gitlab

- Подготовка сервера для обновления версии приложения.
- Пример создания Pipeline на основе Gitlab-ci.
- Производство ПО как конвеер.

Подготовка сервера для обновления версии приложения.

В этой инструкции мы попытаемся объединить уже ранее собранные нами конфигурационные файлы во едино. Что для этого нужно

Подготовим север для запуска сервиса. Установим необходимое ПО

```
#!/bin/bash
apt update
apt install -y docker.io nginx
groupadd docker
usermod -aG docker $USER
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-
$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
 sudo chmod +x /usr/local/bin/docker-compose
 sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

Настроим docker-compose и запустим проект

Настройку проекта мы можем опустить, там как мы её уже изучали в разделе про docker-compose. Обычно к этому моменту мы должны уже иметь репозиторий, который должен работать локально. Его достаточно склонировать, и использовать для запуска.

В проекте не должны использоваться порты 80 и 443, так как это порты используемые nginx.

Для полноценной работы, нам необхдимо изменить адрес контейнера в docker-compose, так как теперь мы не будем собирать контейнер на сервер, за нас это делает pipeline, нам необходимо заменить директиву `build` на `image` таким образом, мы скажем докеру, что ему необходимо брать образ откуда то еще.

Настроим скрипт выливки на сервер и проверим его работоспособность.

Добавим в проект `deploy.sh` - скрипт который позволит нам актоматически равзертывать новую версию на сайте.

```
#!/usr/bin/env bash



#Не забываем авторизоваться в docker, если используем закрытый или частный репозиторий
docker login -u КАКОЙ_ТО-token -p КАКОЙ-ТО-ПАРОЛЬ какой-то.регистри.com

cd "АДРЕС/МЕСТОПОЛОЖЕНИЯ/ПАПКИ/ПРОЕКТА"
```

```
# Вытягиваем новую версию
docker-compose pull;
# Останавливаем старый сервис
docker-compose down;
# Запускаем новую версию
docker-compose up -d
```

Настраиваем nginx и получаем рабочий сервис.

Теперь мы дошли до того, чтобы настроить nginx. Nginx нужен для того, чтобы мы могли настроить работу сервиса по доменному имени.

Для работы нашего сервиса воспользуемся готовым конфигом nginx.conf

```
server{
       listen 80;
       server_name videomanager-test.garpix.com;

       location / {
            proxy_pass http://web:8080;
            proxy_set_header Host $http_host;
            proxy_set_header Connection "upgrade";
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header Upgrade $http_upgrade;
            proxy_connect_timeout       600;
            proxy_send_timeout          600;
            proxy_read_timeout          600;
            send_timeout                600;
        }
}
```

Пример создания Pipeline на основе Gitlab-ci.

Рассмотрим пример создания pipeline на основе gitlab-ci.yaml

Давайте посмотрим какие пункты мы можем внести gitlab.

В gitlab-ci.yaml будет внесены пукнты 3-5 прошлой статьи. То есть, gitlab-ci будет отвечать за создание, тестирование и выпуск образа и развертывание его в рабочем окружении.

1. Добавим в проект файл gitlab-ci.yml.
2. Заполним его, для этого воспользуемся готовым шаблоном:

```
image: docker:latest

variables:
  DOCKER_DRIVER: overlay
  CONTAINER_TEST_IMAGE: $CI_REGISTRY/$CI_PROJECT_PATH:$CI_COMMIT_REF_NAME
  CONTAINER_RELEASE_IMAGE: $CI_REGISTRY/$CI_PROJECT_PATH:latest

services:
  - docker:dind

stages:
  - build
  - test
  - release
  - deploy

build:
```

```
  stage: build
  script:
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY
    - docker build --pull -t $CONTAINER_TEST_IMAGE . --build-arg=secret_key=secret
    - docker push $CONTAINER_TEST_IMAGE
  only:
    - tags

 test:
   image: $CI_REGISTRY/$CI_PROJECT_PATH:$CI_COMMIT_REF_NAME
   stage: test
   services:
     - postgres:11-alpine
   variables:
     SECRET_KEY: runner
     POSTGRES_DB: runner
     POSTGRES_USER: runner
     POSTGRES_PASSWORD: runner
     POSTGRES_HOST: postgres
     POSTGRES_PORT: 5432
     TEST: 1
   script:
     - python3 /code/backend/manage.py qa
   only:
     - tags

release:
  stage: release
  script:
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY
    - docker pull $CONTAINER_TEST_IMAGE
    - docker tag $CONTAINER_TEST_IMAGE $CONTAINER_RELEASE_IMAGE
    - docker push $CONTAINER_RELEASE_IMAGE
  only:
    - tags

deploy:
  image: kroniak/ssh-client
  stage: deploy
  script:
```

```
    - eval $(ssh-agent -s)
    - ssh-add <(echo "$SSH_PRIVATE_KEY")
    - ssh -o StrictHostKeyChecking=no -p $TESTPORT $TESTUSER@$STAGINGHOST make --
directory=$TESTPATH deployfront TAG=$CI_COMMIT_REF_NAME
  only:
    - tags
#  when: manual
```

Для начала разберем переменные используемые в данном файле конфигурации, но которые необхдимо задать в окружении gitlab(Settings->CI\CD->Variables->"Add variables" уберите галочку "Protected variable" для ознакомления она вам не понадобится):

- SSH_PRIVATE_KEY = приватная часть ключа для доступа к удаленной машине
- TESTPORT = порт на котором слушает ssh сервер
- TESTUSER = пользователь у которого есть публичная часть доступа к серверу.
- TESTHOST = адрес хоста,(ip или dns имя)
- TESTPATH = путь до папки проекта.

Все остальные переменные перечислены в секции `variables` (используются в местах где они начинаются с символа `$`) геренируются в процессе запуска скрипта. Берутся они из gitlab окружения, можно найти в документации к gitlab.

Теперь рассмотрим стадии, в приведенном примере в разделе `stages` их 4 стадии:

- build - сбор образа
- test - тестирование
- release - выпуск
- deploy - развертывание

Рассмотрим их более подробно:

Для начала общие части

- image - образ внутри которого будет выполняться скрипт
- stage - стади к которой относится текущая секция

- services - дополнительные сервисы необходимые для проекта
- script - непосредственно выполнение логики.

Производство ПО как конвеер.

Что это за конвеер?

Создание программного обеспечения представялет из себя несколько этапов. Рабочий процесс может отличаться от текущего, но в основном подходы отличаются очередью выполнения.

1. Написание кода - не представляет ничего сложного:
    1. Забираем изменения из репозистория
    2. Вносим изменения и коммитим
    3. Пуши изменения в репозиторий
2. Публикация его в репозиторий - после того как вы его запушили в репозиторий
    1. Ваши изменения требуют мердж реквест в `test` ветку
    2. После того как мердж реквест пройден(изменения одобрил ведущий разрабочик)
    3. Дальше изменения переходят к пункту 3 и идут дальше.
3. Создание образа(рамках нашего проекта это будет docker)
    1. Подготавливается образ
    2. Образ пушится в облако
4. Тестирование образа и релиз
    1. Происходит тестирование
    2. Если тестирование успешно пройдено, то производится резил образа.
5. Развертывание образа
    1. Запускается скрипт, который тригерит обновления окружения.

Если пункт 5 завершился успешно, то теперь пункт 2 выполняется полностью для ветки `master`, И дальше опять пункты 3-5, только уже для производственного окружения.