САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет
Домашняя работа №6

Выполнил: Митурский Богдан Антонович

Группа: K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

Необходимо настроить автодеплой (с триггером на обновление кода в вашем репозитории, на определённой ветке) для вашего приложения на удалённый сервер с использованием Github Actions или Gitlab CI (любая другая CI-система также может быть использована).

Ход работы

Для настройки будем использовать Gitlab CI. Для начала подготовим конфигурационный файл для пайплайнов.

Подготовим несколько пайплайнов, которые будут срабатывать при коммите в main ветку.

build_prod - будет загружать свежую версию ветки на сервер, устанавливать модули и собирать проект

deploy_backend - будет обновлять докер бекенда (только в режиме manual активации, чтобы случайно не перезапустить сервер когда это будет не нужно)

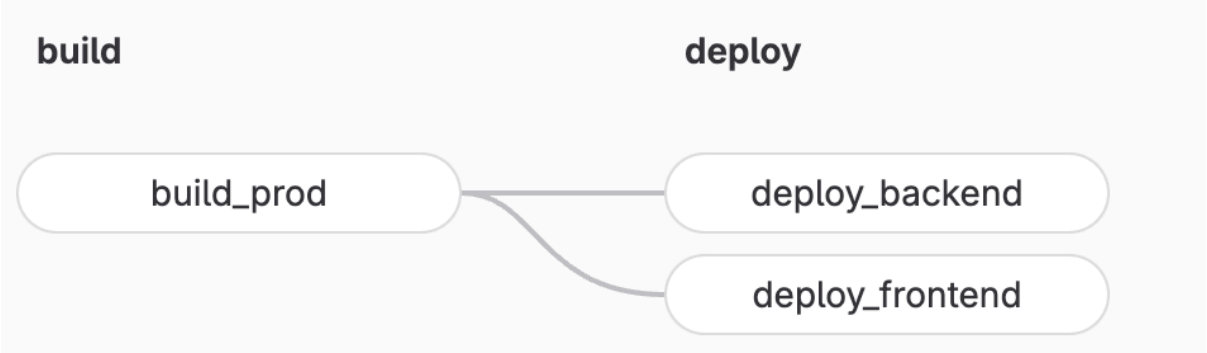
deploy_frontend - будет обновлять вызывать деплой фронтенда на сервере (только в режиме manual активации, чтобы случайно не перезалить клиент когда это будет не нужно)

```
stages:
 - build
 - deploy

.ssh-script-before: &ssh-script
 - apk add openssh-client
 - eval $(ssh-agent -s)
 - echo "$PROD_SSH_KEY" | tr -d '\r' | ssh-add -
 - mkdir -p ~/.ssh
 - chmod 700 ~/.ssh

build_prod:
 stage: build
 image: alpine
 only:
   - main
 allow_failure: false
 before_script:
```

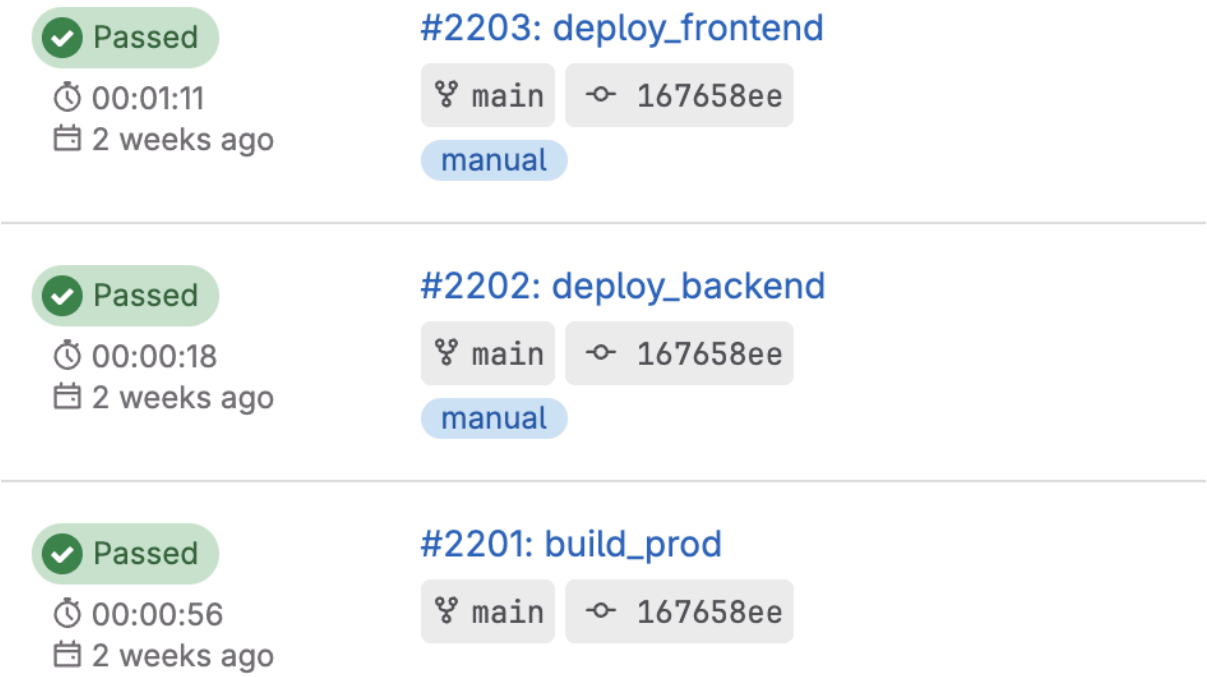
```
    - *ssh-script
  script: |
    echo "Start pipeline"
    ssh -o StrictHostKeyChecking=no $PROD_SSH_USERNAME@$PROD_SSH_HOST bash <<EOF
      cd $PROD_PATH
      source /root/.bashrc
      source ~/.nvm/nvm.sh;
      source script.sh
      section_start "git" "Update repository"
      git restore .
      git pull
      section_end "git"
      section_start "yarn_install" "Installing dependency"
      yarn install
      section_end "yarn_install"
      section_start "build" "Building application"
      yarn build --skip-nx-cache
      section_end "build"
    EOF

deploy_backend:
  stage: deploy
  image: alpine
  before_script:
    - *ssh-script
  script: |
    echo "Start pipeline"
    ssh -o StrictHostKeyChecking=no $PROD_SSH_USERNAME@$PROD_SSH_HOST bash <<EOF
      cd $PROD_PATH
      docker compose up -d --build
    EOF

  only:
    - main
  when: manual
  needs:
    - build_prod
  allow_failure: false

deploy_frontend:
  stage: deploy
  image: alpine
  before_script:
    - *ssh-script
  script: |
```

```
  echo "Start pipeline"
  ssh -o StrictHostKeyChecking=no $PROD_SSH_USERNAME@$PROD_SSH_HOST bash <<EOF
    cd $PROD_PATH
    source /root/.bashrc
    source ~/.nvm/nvm.sh;
    env MINI_APPS_ACCESS_TOKEN=$MINI_APPS_ACCESS_TOKEN yarn deploy
  EOF

only:
  - main
when: manual
needs:
  - build_prod
allow_failure: false
```

Получим следующую визуализацию пайплайнов.



Протестируем и убедимся что всё работает корректно.



Вывод

В ходе выполнения работы я настроил пайплайны через Gitlab CI для своего проекта, что позволило автоматизировать и упростить процессы деплоя. Автодеплой очень удобная и несложная в настройке вещь.