

Курс DevOps

Лабораторная работа №2

GitHub Actions

1 апреля 2026 г.

Общие инструкции

Рекомендации к выполнению

- Настоятельно рекомендуется выполнять команды из листинга отдельно, одну за другой.
- Рекомендуется копировать команды через промежуточную вставку в текстовый редактор во избежание лишних символов.
- Перед началом работы ознакомьтесь с [инструкцией по сдаче лабораторных работ](#).

Требования к выполняемой работе

- Добавьте в репозиторий все необходимые файлы: код приложения, файлы конфигурации и скрипты и т.д.

Внимание: Данная лабораторная работа является продолжением лабораторной работы №1. Для выполнения данной работы необходимо иметь подготовленный репозиторий из первой лабораторной работы.

1 Цель работы

Заменить самописный webhook обработчик из предыдущей лабораторной работы на CI/CD pipeline на платформе GitHub Actions. Сравнить подходы и изучить преимущества CI/CD платформ.

2 Содержание работы

В предыдущей работе был создан webhook обработчик, который автоматически разворачивал приложение при событии в репозитории. GitHub Actions позволяет решить ту же задачу без написания собственного обработчика и с дополнительными возможностями.

Основные компоненты GitHub Actions:

- **Workflow** — набор из одной или нескольких задач (Job) автоматизации
- **Job** — логическое объединение шагов (Step) автоматизации
- **Step** — непосредственно исполняемое действие автоматизации. Например, запуск shell скриптов или вызов готовых наборов задач — Action
- **Action** — заранее определенный набор задач (Job), предоставляемый GitHub или сообществом
- **Runner** — сервер, который выполняет workflows

Преимущества GitHub Actions перед webhook веб-сервером:

- Встроенная интеграция с GitHub
- Широкий выбор готовых actions
- Визуальный интерфейс и логи
- Безопасное хранение секретов
- Не требует собственной инфраструктуры для [небольших проектов](#)

2.1 Задание 1

Подготовить репозиторий из предыдущей лабораторной работы.

1. создайте отдельную ветку lab2 для работы с GitHub Actions;
2. временно отключите webhook: в настройках репозитория (Settings → Webhooks) снимите галочку Active у конкретного webhook;
3. создайте SSH ключи для доступа GitHub Actions к вашей виртуальной машине:

```
# Генерируем SSH ключи (на вашей виртуальной машине)
ssh-keygen -t rsa -C "github-actions" -f ~/.ssh/github_actions -N ""

# Добавляем публичный ключ в authorized_keys
cat ~/.ssh/github_actions.pub >> ~/.ssh/authorized_keys

# Выводим приватный ключ для добавления в GitHub Secrets
cat ~/.ssh/github_actions
```

4. скопируйте содержимое приватного ключа и добавьте его в GitHub Secrets:
 - Перейдите в Settings → Secrets and variables → Actions
 - Нажмите New repository secret
 - Name: SSH_PRIVATE_KEY
 - Value: вставьте содержимое приватного ключа целиком (включая -----BEGIN и -----END строки)
 - Сохраните секрет

2.2 Задание 2

Настроить SSH доступ до вашей виртуальной машины через FRP для работы CD workflow.

Внимание! Будьте осторожны, открывая SSH доступ всему миру! Позаботьтесь о том, что у вас стоит надежный пароль, а лучше всего чтобы на виртуальной машине была отключена аутентификация по паролю.

1. Обновите конфигурацию FRP для работы с GitHub Actions:

```
# Настройка переменных. Актуальные значения уточняйте в ведомости!
PROXY=course.prafdin.ru
TOKEN=token
ID=prafdin
SSH_PORT=2200

# Обновляем конфигурацию FRP - убираем webhook, добавляем SSH
# Команда запросит пароль текущего пользователя
sudo tee /etc/frp/frpc.toml > /dev/null <<EOF
serverAddr = "$PROXY"
serverPort = 7000
```

```
auth.method = "token"
auth.token = "$TOKEN"

# Проброс для приложения (остается как было)
[[proxies]]
name = "app-$ID"
type = "http"
localPort = 8181
customDomains = ["app.$ID.$PROXY"]

# Новый проброс для SSH доступа GitHub Actions
[[proxies]]
name = "ssh-$ID"
type = "tcp"
localPort = 22
remotePort = $SSH_PORT
EOF

# Перезапустите frpc
sudo systemctl restart frpc

# Проверьте SSH подключение к серверу через новый порт
ssh -p $SSH_PORT $USER@$PROXY
```

2.3 Задание 3

Создать CI workflow для проверки кода.

1. Создайте файл `.github/workflows/ci.yaml`;
2. Настройте workflow для запуска тестов при push нового коммита и создании pull request;
3. В workflow добавьте загрузку файл с отчетом запущенных тестов с названием `test_result` с помощью [actions/upload-artifact](#);
4. Убедитесь, что workflow запускается, успешно проходит и загружается файл с результатами тестов.

В этом workflow вся работа (сборка, тестирование) должна происходить на GitHub Runner. В рамках CI workflow подключения до виртуальной машины не требуется.

Полезная информация:

- [Список предустановленного ПО на GitHub Runners](#)
- Полезные GitHub Actions:
 - [actions/checkout](#);
 - [actions/setup-dotnet](#);
 - [actions/setup-python](#).

2.4 Задание 4

Создать CD workflow развертывания приложения.

1. создайте файл `.github/workflows/deploy.yaml`;
2. настройте запуск workflow по наступлению события [release](#);
3. реализуйте ту же логику развертывания, что была в webhook веб-сервере;
4. протестируйте развертывание;
5. убедитесь, что приложение обновляется и доступно по тому же адресу: `http://app.$ID.$PROXY`

Этот workflow должен включать в себя сборку и развертывание приложения. Сборку необходимо производить на GitHub Runner. Для доступа к виртуальной машине воспользуйтесь предварительно добавленным приватным ключом через [GitHub Secrets](#)

Полезная информация:

- Полезные GitHub Actions:
 - [appleboy/ssh-action@v1](#);
 - [webfactory/ssh-agent@v0.7.0](#);
 - [appleboy/scp-action@v1](#).

Примечание:

- Все необходимые зависимости приложения предварительно могут быть установлены вручную на виртуальной машине. Список зависимостей должен быть зафиксирован в документации или специальном файле, например, в `requirements.txt` в случае Python приложения;
- Разворачиваемое приложение должно быть доступно по адресу `http://app.$ID.$PROXY` и запущено на порту 8181.

3 Контрольные вопросы

1. Сравните GitHub Actions и webhook обработчик из предыдущей лабораторной работы. Какие преимущества и недостатки у каждого подхода?
2. Что такое Workflow в GitHub Actions и из каких основных компонентов он состоит?
3. Какие события могут запускать workflow в GitHub Actions? Приведите несколько примеров.
4. В чем разница между GitHub Secrets и GitHub Environment Variables? Когда использовать каждый?

5. В чем разница между GitHub-hosted runners и self-hosted runners? Какие плюсы у каждого типа?
6. Зачем использовать FRP в данной работе? Какие проблемы это решает?
7. В чем заключается различие между CI и CD задачами? Как эти практики реализованы в данной лабораторной работе?
8. Почему важно разделять этапы сборки, тестирования и развертывания в CI/CD pipeline? Какие преимущества это даёт?
9. Чем отличается trigger по событию push от триггера pull request? В каких сценариях используется каждый из них?
10. Как можно откатить приложение при настроенном CI/CD pipeline?