УДК 004

Автоматизация тестирования с помощью Github Action

Вихляев Дмитрий Романович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описывается метод создания автоматического тестирования проекта как часть решения процесса непрерывной интеграции. Инструментом реализации является Github Action, который расположен внутри веб-сервиса совместной разработки IT-проектов Github. В результате исследования создано действие автоматически запускающее тест при каждой загрузке новых изменений проекта на Github.
Ключевые слова: Github Action, CI/CD, автоматизация тестирования.

Test automation using Github Action

Vikhlyaev Dmitry Romanovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article describes a method for creating automated project testing as part of a continuous integration process solution. The implementation tool is Github Action, which is located inside the Github IT project collaboration web service. As a result of the research, an action will be created that automatically starts the test every time new project changes are uploaded to Github.
Keywords: Github Action, CI/CD, test automation.

1 Введение

1.1 Актуальность

Автоматизация тестирования в настоящее время играет ключевую роль в разработке программного обеспечения. Повышение эффективности и скорости благодаря быстрому выполнению множества тестов существенно сокращает время цикла разработки. Тесты выполняются однозначно и не подвержены человеческим ошибкам, уменьшая риск пропуска серьезных дефектов. Снижение затрат в долгосрочной перспективе за счёт экономии средств, так как автоматические тесты могут быть повторно использованы без необходимости постоянного вовлечения ресурсов вручную. Обеспечение более надежной и последовательной проверки функциональности, что помогает выявлять ошибки на ранних этапах разработки, увеличивает общее покрытие функциональности продукта. Автоматизация тестирования является

неотъемлемой частью CI/CD-процессов, обеспечивая автоматическую проверку кода при каждом изменении.

1.2Обзор исследований

К.С.Каратыгин анализировал распространенных программных систем автоматизации тестирования [1]. А.А.Котлячков провёл автоматизацию функционального тестирования сложных программных средств [2]. Д.В.Силаков описал автоматизацию тестирования web-приложений, основанных на скриптовых языках [3] И.А.Яковлев исследовал автоматизацию тестирования и контроля качества систем оптического распознавания символов с применением нейронных сетей [4]. В.П.Котляров, А.А.Голубев, А.Н.Карпов в своей статье показали метод автоматизации тестирования встроенных java-приложений с адаптацией к целевой платформе [5].

1.3Цель исследования

Цель исследования – используя веб-сервис Github и инструмент Github Action реализовать автоматический запуск тестов после загрузки любых изменений на удалённый репозиторий.

2 Материалы и методы

Для реализации используются язык программирования python и встроенный в стандартную библиотеку модуль unittest.

3 Результаты и обсуждения

Github Actions – это интегрированный сервис автоматизации для проектов, размещенных на платформе Github. Этот сервис позволяет создавать, настраивать и выполнять различные автоматические задачи в рамках рабочего репозитория.

Github Actions позволяет определить и настроить рабочий процесс (workflow), который состоит из шагов (steps) и действий (actions). Этот рабочий процесс может автоматизировать широкий спектр задач, от сборки и тестирования кода до развертывания приложений.

Рабочие процессы можно настроить для автоматического запуска при определенных событиях, таких как отправка изменений в репозиторий, создание нового тега, или другие пользовательские события. Можно создавать собственные действия для выполнения специфичных для проекта задач. Это позволяет максимально адаптировать автоматизацию под потребности.

В начале необходимо определить рабочий процесс (Workflow Definition). В корне репозитория создается директория .github/workflows, в которой размещаются файлы с описанием рабочих процессов. Эти файлы содержат инструкции для выполнения различных задач.

События определяют, на какие действий должен реагировать рабочий процесс. Например, это может быть событие отправки изменений в

репозиторий (push), создание нового тега (tag), или другие пользовательские события.

Каждый рабочий процесс состоит из шагов, представляющих собой отдельные задачи. Шаги могут включать в себя различные действия, такие как клонирование репозитория, сборка проекта, запуск тестов, развертывание и другие.

Шаги используют действия, которые могут быть предопределенными (поставляемыми Github) или собственными. Действия – это переиспользуемые блоки кода, предназначенные для выполнения конкретных задач.

Каждый шаг выполняется в своей среде выполнения, которая предоставляет определенные параметры окружения, такие как доступные версии языков, установленные пакеты и т.д.

В зависимости от конфигурации, Github Actions может уведомлять о результатах выполнения, например, отправлять уведомление в Slack или создавать Issue в репозитории.

Для настройки используется формата файлов YAML. Данный файл предназначается как файл конфигурации. В нём описывается, когда, где и что будет запускаться, какие зависимости должны быть установлены, а также сами команды запуска.

Начинается он с имени, имя можно выбрать любое. Затем идёт конструкции, которые объясняют, что нужно сделать.

Первая конструкция «on». Указывает, в какой ситуации будут запускаться действия. Например, запуск тестов произойдёт после команды «push», данная команда загружает проект на удалённый репозиторий, автоматически подхватывая изменения в проекте.

Алгоритм запуска тестов jobs (задача). Для каждого запуска создаётся контейнер, как виртуальная машина, внутри которого запускаются необходимые настройки и программы. Так как каждые решения требуют своей специфики, начиная от версии интерпретатора заканчивая операционной системой, Github Action предлагает гибкую настройку всего окружения.

В конструкции «runs-on» указывается на чём будет запускаться задача. Здесь указывается версия операционной системы или их список.

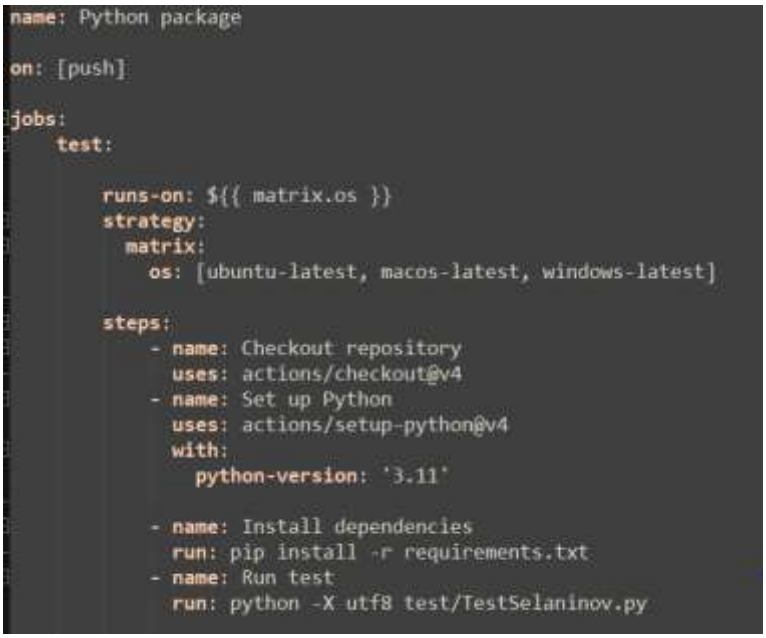
Чтобы проект был проверен внутри контейнера, нужно загрузить в него репозиторий. Github предоставляет Action специально под данный шаг.

Функция называется checkout. У каждой функции есть версия, на данный момент 4 версия данной функции является последней.

Для запуска тестов, также необходим интерпретатор python, у Github доступна специальная функция и под этот случай. Для некоторых action имеются дополнительные настройки, чтобы их применить необходимо, использовать строку «with». В данном случае указана версия python.

Помимо самого интерпретаторы, также возможно понадобится загрузить необходимые модули. В программе pip есть специальный метод для вывода всех установленных модулей. Проще всего переназначить их вывод в текстовый файл и поместить в папку проекта.

Запуск теста осуществляется как простой запуск скрипта python из консоли. Параметр «-X» указывает кодировку, в котором запускается скрипт. В тестируемой функции присутствует вывод предупреждения в стандартный поток вывода о, том, что требования параметров для корректного выполнения не соблюдены. Чтобы оставить это предупреждение на русском языке на операционной системе windows, нужно принудительно указать кодировку (рис. 1).



```yaml
name: Python package

on: [push]

jobs:
  test:

    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, macos-latest, windows-latest]

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'

      - name: Install dependencies
        run: pip install -r requirements.txt
      - name: Run test
        run: python -X utf8 test/TestSelaninov.py
```

Рис. 1. Структура yaml файла

В качестве примера тестируемой функции используется метод Селянинова, используемый для расчёта индекса засухи на основе значений температуры и осадков. Чтобы метод корректно выдал результат необходимо убедится, что размеры списка температуры и осадков равны. Второе менее очевидное условие, связано с возможным делением на ноль, в случае если размер хотя бы одного из списков равен нулю. Так как уже предусмотрено условие равенства длин, то достаточно проверить только один список на нулевую длину (рис. 2).

```python
def selaninov_drought_index(temperature, precipitation):
    """
    Вычисление индекса засухи Селянинова.

    Параметры:
    - temperature: Список ежемесячных значений температуры.
    - precipitation: Список ежемесячных значений осадков.

    Возвращаемое значение:
    - Значение индекса засухи Селянинова.
    """

    # Проверка, что длины списков температуры и осадков совпадают
    if len(temperature) != len(precipitation):
        print("Длины списков температуры и осадков должны быть одинаковыми.")
        return None
    elif len(precipitation)==0:
        print("Длины списка осадков должны быть больше 0.")
        return None

    # Вычисление средней температуры
    avg_temperature = sum(temperature) / len(temperature)

    # Вычисление суммы ежемесячных осадков
    total_precipitation = sum(precipitation)

    # Вычисление индекса засухи Селянинова
    kddi = avg_temperature / (total_precipitation / len(precipitation))

    return kddi
```

Рис. 2. Метод Селянинова на python

В стандартной библиотеке python вострен модуль unittest, обладающий достаточным набором методов для создания разных тестов средней сложности. Для примера используется простой метод проверки равенства между желаемым и полученным от проверяемой функции результатом. Приведены проверки, для получения хороших и плохих результатов (рис. 3).

```python
import unittest
import os, sys
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from drought_index.Selaninov import selaninov_drought_index

class TestSelaninovDroughtIndex(unittest.TestCase):

    def test_valid_input(self):
        temperature_data = [25, 26, 27, 24, 23, 22, 21, 20, 19, 18, 17, 16]
        precipitation_data = [50, 40, 30, 20, 10, 5, 5, 10, 15, 20, 25, 30]

        result = selaninov_drought_index(temperature_data, precipitation_data)
        expected_result = (
            sum(temperature_data)/len(temperature_data)
            ) / (
                sum(precipitation_data) / len(precipitation_data)
                )

        self.assertAlmostEqual(result, expected_result, places=5, msg="Invalid result")

    def test_invalid_input_length(self):
        temperature_data = [25, 26, 27, 24, 23, 22, 21, 20, 19, 18, 17]
        precipitation_data = [50, 40, 30, 20, 10, 5, 5, 10, 15, 20, 25, 30]


        result=selaninov_drought_index(temperature_data, precipitation_data)
        self.assertAlmostEqual(result, None)

    def test_invalid_input_zero_length(self):
        temperature_data = []
        precipitation_data = []

        result=selaninov_drought_index(temperature_data, precipitation_data)
        self.assertAlmostEqual(result, None )

if __name__ == '__main__':
    unittest.main()
```

Рис. 3. Содержание методов unittest

После загрузки проекта на Github, происходит автоматический запуск тестов на всех указанных платформах (рис. 4).

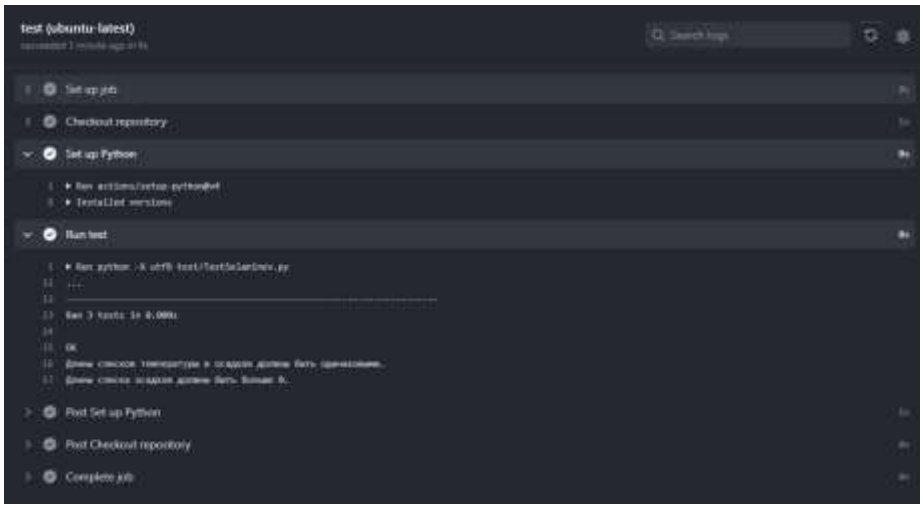


Рис. 4. Успешное прохождение теста

Если убрать второе условие из проверяемой функции и снова загрузить на удалённый сервер. Новый запуск тестов покажет ошибку при проверке деления на ноль (рис. 5).

Рис. 5. Вывод ошибки теста

Все тесты хранятся подобно камитам каждый из которых можно подробно посмотреть (рис. 6).

Рис. 6. История проведённых тестов проекта

В данной статье было разобрано подключение Github Action на удалённом репозитории и настройка необходимых действий для проведения теста. Описано создание простого теста на python загрузка и проверка выполнения автоматического тестирования после внесения новых изменений на удалённое хранилище проекта.

Библиографический список

1. Каратыгин К.С. Анализ распространенных программных систем автоматизации тестирования // Техника и технология. 2010. № 3. С. 22-31.
2. Котлячков А.А. Автоматизация функционального тестирования сложных программных средств // Автоматизация в промышленности. 2010. № 2. С. 32-36.
3. Силаков Д.В. Автоматизация тестирования web-приложений, основанных на скриптовых языках // Труды Института системного программирования РАН. 2008. Т. 14. № 2. С. 159-178.
4. Яковлев И.А. Автоматизация тестирования и контроля качества систем оптического распознавания символов с применением нейронных сетей // Сборник научных трудов SWorld. 2011. Т. 2. № 1. С. 39-42.
5. Котляров В.П., Голубев А.А., Карпов А.Н. Автоматизация тестирования

встроенных java-приложений с адаптацией к целевой платформе // Научно-технические ведомости СПбГТУ. 2006. № 5-1 (47). С. 117-124.
6. Преображенская Т.В., Симонов В.И. Автоматизированное тестирование в среде GITLAB CI/CD // В сборнике: Информатика: проблемы, методы, технологии. материалы XXIII Международной научно-практической конференции им. Э.К. Алгазинова. Воронеж, 2023. С. 1447-1451.
7. Сергейчев А.В., Коняева О.С. Инструменты автотестирования и их важность в процессе CI/CD // В книге: Проблемы техники и технологии телекоммуникаций. Оптические технологии в телекоммуникациях. Материалы XXIV Международной научно-технической конференции и материалы XX Международной научно-технической конференции. В 2-х томах. Уфа, 2023. С. 114-115.
8. Государев И.Б., Кочетыгов А.В. Анализ применения непрерывной интеграции среди проектов на Github // Современное образование: традиции и инновации. 2020. № 2. С. 99-102.