**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**Курсовая работа**

**по дисциплине «Технологии автоматизации процесса разработки программного обеспечения»**

**Тема: Реализация системы автоматизированного тестирования ИС ИОС.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9305 |  | Пак А.И. |
| Преподаватель |  | Корытов П.В. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Пак А.И. | | |
| Группа 9305 | | |
| Тема работы: Реализация системы автоматизированного тестирования ИС ИОС. | | |
| Исходные данные:  Необходимо реализовать docker-compose конфигурацию из двух узлов (не больше и не меньше):  app - контейнер с существующим демонстрационным веб-приложением  tester - контейнер для запуска всех тестов  Задача в написании Selenium-тестов - написать автотесты для нескольких форм ИС ИОТ | | |
| Содержание пояснительной записки:  Введение; Dockerfile, Docker-compose; Этапы тестирования; Заключение | | |
| Предполагаемый объем пояснительной записки:  Не менее 14 страниц. | | |
| Дата выдачи задания: 08.12.2024 | | |
| Дата сдачи реферата: 27.12.2024 | | |
| Дата защиты реферата: 27.12.2024 | | |
| Студент |  | Пак А.И. |
| Преподаватель |  | Корытов П.В. |

СОДЕРЖАНИЕ

[Введение 6](#_Toc186185063)

[1. Dockerfile, Docker-compose 7](#_Toc186185064)

[1.1. Dockerfile\_app 7](#_Toc186185065)

[1.2. Dockerfile\_tester 8](#_Toc186185066)

[1.3. Docker-compose 9](#_Toc186185067)

[2. Этапы тестирования 10](#_Toc186185068)

[2.1. Проверка на соответствие стилю кодирования 10](#_Toc186185069)

[2.2. Статический анализ 10](#_Toc186185070)

[2.3. Интеграционное тестирование 10](#_Toc186185071)

[2.4. Selenium-тесты 11](#_Toc186185072)

[2.5. Скрипт для запуска тестов 12](#_Toc186185073)

[Заключение 13](#_Toc186185074)

[Список использованных источников 14](#_Toc186185075)

**АННОТАЦИЯ**

В данной работе реализуется процесс автоматизированного тестирования информационной системы ИОТ. В процессе работы осуществляется настройка Docker и docker-compose для создания тестового окружения, конфигурируются два контейнера, с демонстрационным веб-приложением и контейнер для запуска всех тестов.

**SUMMARY**

This work implements the process of automated testing of the IOT information system. During the work, Docker and docker-compose are configured to create a test environment, two containers are configured, with a demo web application and a container for running all tests.

Введение

Целью работы является реализация процесса автоматизированного тестирования информационной системы (ИС) ИОТ с использованием Docker и Selenium. В рамках работы настраиваются необходимые компоненты для автоматизированного тестирования, включая конфигурацию Docker, docker-compose, написание тестовых сценариев с использованием фреймворка pytest. Реализуется интеграция в процесс автоматизированного тестирования различных этапов, таких как проверка на соответствие стиля кодирования, статический анализ, интеграционное тестирование, selenium-тесты.

1. Dockerfile, Docker-compose
   1. Dockerfile\_app

FROM ubuntu:22.04

RUN apt-get update && apt-get install -y \

python3=3.10.6-1~22.04.1 \

python3-pip=22.0.2+dfsg-1ubuntu0.5 \

git=1:2.34.1-1ubuntu1.11 \

openssh-server=1:8.9p1-3ubuntu0.10

RUN mkdir -p /root/.ssh && \

chmod 700 /root/.ssh && \

mkdir /var/run/sshd && \

sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin prohibit-password/' /etc/ssh/sshd\_config && \

sed -i 's/#PasswordAuthentication yes/PasswordAuthentication no/' /etc/ssh/sshd\_config

COPY ssh/authorized\_keys/docker\_rsa.pub /root/.ssh/authorized\_keys

RUN chmod 600 /root/.ssh/authorized\_keys

RUN git clone https://github.com/moevm/devops-examples.git

COPY requirements\_app.txt devops-examples/EXAMPLE\_APP/requirements.txt

WORKDIR devops-examples/EXAMPLE\_APP

RUN pip3 install -r requirements.txt

COPY host.diff .

RUN patch main.py < host.diff

CMD ["sh", "-c", "/usr/sbin/sshd -D & python3 main.py"]

Используется базовый образ ubuntu:22.04. В процессе сборки обновляются и устанавливаются необходимые пакеты, устанавливаются зависимости из скопированного файла requirements.txt (requirements\_tester.txt). Выполняется патч для внесения правок в код склонированного демонстрационного веб-приложения EXAMPLE\_APP.

Для работы с SSH устанавливается openssh-server, создается каталог .ssh с ограниченными правами доступа 700. Вход в систему для пользователя root ограничивается аутентификацией на основе заранее сгенерированных ключей (публичный ключ копируется в файл /root/.ssh/authorized\_keys), при этом отключается аутентификация по паролю.

Доступ к контейнеру происходит с использованием приватного ключа.

Внешний SSH доступ в контейнер:

ssh -i <путь\_до\_приватного\_ключа> -p 2222 root@127.0.0.1

* 1. Dockerfile\_tester

FROM ubuntu:22.04

WORKDIR /app

RUN apt-get update && DEBIAN\_FRONTEND=noninteractive apt-get install -y \

python3=3.10.6-1~22.04.1 \

python3-pip=22.0.2+dfsg-1ubuntu0.5 \

python3-tk=3.10.8-1~22.04 \

xvfb=2:21.1.4-2ubuntu1.7~22.04.12 \

wget=1.21.2-2ubuntu1.1 && \

wget -q https://dl.google.com/linux/direct/google-chrome-stable\_current\_amd64.deb && \

apt-get install -y ./google-chrome-stable\_current\_amd64.deb && \

rm google-chrome-stable\_current\_amd64.deb

COPY requirements\_tester.txt requirements.txt

RUN pip3 install -r requirements.txt

COPY tests ./tests

CMD ["python3", "-m", "http.server", "3000"]

Используется базовый образ ubuntu:22.04. В качестве рабочей директории выступает app. В процессе сборки обновляются и устанавливаются необходимые пакеты, устанавливаются зависимости из скопированного файла requirements.txt (requirements\_tester.txt). Помимо стандартных пакетов скачивается, распаковывается и устанавливается Google Chrome, далее установочный файл удаляется. Копируется директория с тестами tests, корнем дерева процессов выступает стандартный python http сервер.

* 1. Docker-compose

version: "3.5"

services:

example\_app:

restart: always

build:

dockerfile: Dockerfile\_app

ports:

- "127.0.0.1:${WEB\_SERVER\_PORT}:${WEB\_SERVER\_PORT}"

- "127.0.0.1:2222:22"

deploy:

resources:

limits:

cpus: "3"

tester:

restart: always

build:

dockerfile: Dockerfile\_tester

env\_file:

- ./.env

* Указана версия docker compose 3.5
* example\_app, tester: указаны пути к docker-файлам для сборки образов
* example\_app: открыт порт 2222 (22) для подключения к контейнеру по ssh и порт демонстрационного веб-приложения, полученный из переменной окружения
* example\_app: ограничено использование CPU – максимальное количество ядер для контейнера установлено на 3
* tester: передан env-файл для дальнейшей передачи переменных окружения внутрь контейнера

Сборка образов:

docker compose build

Запуск контейнеров:

docker compose up

1. Этапы тестирования
   1. Проверка на соответствие стилю кодирования

Проверка на соответствие стилю кодирования Python была произведена с помощью black (<https://github.com/psf/black>). Используется рекомендованная конфигурация по умолчанию.

Запуск проверки кода директории tests:

black --check tests

* 1. Статический анализ

Статический анализ в проекте выполняется с использованием pylint. В конфигурационном файле заданы 10 уникальных критериев проверки.

[MAIN]

[MESSAGES CONTROL]

disable=all

enable=missing-function-docstring,

missing-module-docstring,

invalid-name,

line-too-long,

deprecated-module,

logging-too-few-args,

condition-evals-to-constant,

return-in-finally,

lost-exception,

duplicate-key

[REPORTS]

output-format=text

reports=no

Запуск статического анализа кода директории tests:

pylint --rcfile=tests/.pylintrc tests

* 1. Интеграционное тестирование

Процесс интеграционного тестирования включал в себя тест загрузки файла в демонстрационное веб-приложение. Для тестирования использовался фреймворк pytest и библиотеки requests для http-запросов.

Сценарий тестирования включал в себя загрузку файла file.png по адресу <http://example_app:5000/upload> (POST-запрос), и получение этого файла по адресу http://example\_app:5000/download/file.png (GET-запрос), где example\_app – запущенный контейнер с демонстрационным веб-приложением.

* 1. Selenium-тесты

Было произведено тестирование заполнения вкладок 7 и НМ информационной системы ИОТ, настроен ChromeDriverManager, использован фреймбуфер xvfb для создания виртуального дисплея.

exec -a xvfb Xvfb :1 -screen 0 1920x1080x16 &> xvfb.log &

DISPLAY=:1.0

export DISPLAY

pytest tests/selenium\_tests/test\_opop.py > >(tee -a /proc/1/fd/1) 2>&1

rcode=$?

kill $(pgrep -f xvfb)

exit ${rcode}

Перед исполнением selenium тестов запускается процесс xvfb, создается виртуальный дисплей с номером 1, задается стандартное разрешение экрана и глубина цвета. Вывод (stdout и stderr) перенаправляется в файл xvfb.log.

Для организации процесса написания тестов был выбран и реализован упрощенный вариант паттерна Page Object, создана базовая страница, классы-локаторы, функции для непосредственного исполнения сценариев тестирования.

Этапы тестирования включают в себя:

1. Авторизация через ETU ID. Используются данные для авторизации из env файла (логин, пароль ETU ID).
2. Авторизация за пользования id=1305
3. Создание документа (или взятие из черновиков)
4. Заполнение вкладки 7. Происходит внесение данных в таблицу «Лист регистрации изменений» из тестового json-файла. Также проходит проверка выполнения операции путем сравнения тестового json-файла с соответствующим полем из «Превью JSON», предоставляемый системой.
5. Заполнение вкладки «Настройка модулей». Происходит копирование РП и заполнение формы случайным выбором вариантов.
   1. Скрипт для запуска тестов

Для упрощения процесса запуска различных этапов тестирования был реализован bash-скрипт скрипт run.sh. Вывод перенаправляется в файловый дескриптор fd/1 процесса с PID 1, основного процесса контейнера. К записям добавлена информация о дате и времени запуска конкретного этапа тестирования. По каждому этапу тестирования предоставляется развернутая информация о выполненном процессе.

Запуск всех этапов тестирования производится следующей командой:

run.sh –all

Запуск отдельных этапов:

run.sh --test <Этап>

Заключение

В данной работе был выполнен ряд задач, направленных на автоматизацию тестирования информационной системы (ИС) ИОТ. Создан Docker-контейнер (сервис) app для запуска демонстрационного веб-приложения и контейнер (сервис) tester для запуска всех тестов. Были написаны Docker-файлы, настроен Docker Compose для работы с Docker-контейнерами. Процесс автоматизации тестирования включал в себя различные этапы, такие как проверка на стиль кодирования с использованием black, статические анализ с pylint, этап интеграционного тестирования и selenium-тесты вкладки 7 и НМ информационной системы ОИТ. В качестве фреймворка для тестирования использовался pytest.

Список использованных источников

1. Docs.docker – URL: <https://docs.docker.com/build-cloud/> (дата обращения – 11.12.24).
2. Docs.docker/Docker Compose – URL: <https://docs.docker.com/compose/> (дата обращения – 13.12.24).
3. Psf/black – URL: <https://github.com/psf/black> (дата обращения – 13.12.24).
4. Docs.pytest – URL: <https://docs.pytest.org/en/stable/> (дата обращения – 13.12.24).
5. Автоматизация тестирования с помощью Selenium и Python – URL: <https://stepik.org/course/575/promo> (дата обращения – 13.12.24).
6. Доклад "Тестирование веб-приложений в Selenium" – URL: <https://www.youtube.com/watch?v=gLj6BMmI69I&ab_channel=OSLLvideos> (дата обращения – 13.12.24).