**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: Использование Docker**

**Вариант 1**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8306 |  | Азаров Д.Б. |
| Преподаватель |  | Заславский М.М. |

Санкт-Петербург

2023

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Азаров Д.Б. | | |
| Группа 8306 | | |
| Тема работы: Использование Docker | | |
| Исходные данные:  Требуется реализовать конфигурацию docker-compose, состоящую из двух контейнеров – с приложением и с тестами. | | |
| Содержание пояснительной записки:  Содержание, Введение, Постановка задачи, Описание Dockerfile, Описание скриптов запуска тестов, Описание конфигурации docker-compose, Заключение, Список использованных источников. Приложение. | | |
| Предполагаемый объем пояснительной записки:  Не менее 15 страниц. | | |
| Дата выдачи задания: | | |
| Дата сдачи реферата: | | |
| Дата защиты реферата: | | |
| Студент |  | Азаров Д.Б. |
| Преподаватель |  | Заславский М.М. |

**Аннотация**

В результате данной работы была реализована конфигурация docker-compose, состоящая из двух контейнеров, и соответствующие им dockerfile. Один контейнер запускает предоставленное приложение, другой – тесты. Тестирование включает анализатор кода pylint, запуск реализованных интеграционных тестов, проверка на pep8.

**Summary**

As a result of this work, a Docker Compose configuration was implemented, consisting of two containers, and their corresponding Dockerfiles. One container runs the provided application, while the other runs tests. Testing includes the code analyzer pylint, running implemented integration tests and checking for pep8 compliance.

Содержание

[Введение 5](#_Toc134885393)

[Постановка задачи 6](#_Toc134885394)

[1. Описание Dockerfile 8](#_Toc134885395)

[1.1. Dockerfile для app-контейнера 8](#_Toc134885396)

[1.2. Dockerfile для tester-контейнера 8](#_Toc134885397)

[2. Описание скриптов запуска тестов 10](#_Toc134885398)

[2.1. Проверка на pep8 10](#_Toc134885399)

[2.2. Статический анализ по 10 критериям 10](#_Toc134885400)

[2.3. Интеграционный тест 10](#_Toc134885401)

[2.4. Selenium тест 10](#_Toc134885402)

[2.5. Запуск тестов 10](#_Toc134885403)

[3. Описание Docker-compose конфигурации 12](#_Toc134885404)

[Заключение 13](#_Toc134885405)

[Список использованных источников 14](#_Toc134885406)

[Приложение А. Содержание созданных файлов 15](#_Toc134885407)

# Введение

В данной работе реализована конфигурация docker-compose, состоящая из двух контейнеров, и dockerfile для их создания. Один контейнер используется для запуска веб-сервера, другой – для запуска тестов. Для запуска тестов используется pytest.

# Постановка задачи

Необходимо реализовать конфигурацию docker-compose, состоящую из двух узлов: с приложением и с тестами для него. При этом необходимо выполнить следующие задачи:

1. Добавить в тестирование проверку на pep8;
2. Проанализировать по десяти существующим критериям статического анализа;
3. Написать интеграционный тест, проверяющий коды возврата.
4. Написать selenium тест проверяющий заполнение формы авторизации (включая проверку верстки страницы)
5. Обеспечить доступ по ssh ключу
6. Записывать каждый этап тестирования в docker log, а также оба потока в один файл
7. Добавить возможность передачи порта через файл .env;
8. Ограничить максимальное потребление ОЗУ в 110МБ.

Требования к docker-compose:

* Минимальная версия docker compose version 1.27.4, build 40524192
* Все должно собираться по команде docker-compose build без sudo
* Не использовать тип сети HOST
* Не отрывать лишних (непредусмотренных заданием) портов
* Не использовать порты хост-машины ⇐1024

Требования к Dockerfile:

* Минимальная версия докера Docker version 19.03.13, build 4484c46d9d
* Базовый образ ubuntu:22.04
* Не использовать Expose
* При установке любых пакетов и программ (в том числе в requirements) ВСЕГДА указывать версии
* Ограничить установку зависимостей apt одной строкой (один RUN)
* Если настройка одной части приложения состоит из нескольких команд → необходимо разместить их в одном слое (в одном RUN)

# Описание Dockerfile

## Dockerfile для app-контейнера

В качестве базового используется образ ubuntu:22.04.

Устанавливаются зависимости:

* git – для скачивания кода приложения;
* python – для его запуска;
* pip – для установки зависимостей.

Загружается код приложения, рабочей директорией устанавливается директория с кодом приложения.

Устанавливаются переменные окружения Flask – хост и файл сервера

(main.py).

В контейнер копируется файл со списком python-зависимостей, после чего с помощью pip выполняется установка указанных библиотек.

Указывается точка входа – код приложения.

## Dockerfile для tester-контейнера

В качестве базового используется образ ubuntu:22.04.

Устанавливаются зависимости:

* git – для скачивания кода приложения;
* python – для запуска тестов;
* pip – для установки зависимостей;
* wget – для скачивания firefox и geckodriver;
* xvfb – для запуска firefox без графического интерфейса;
* libgtk, libdbus и libasound – зависимости, необходимые для запуска firefox;

Выполняется установка firefox и geckodriver, необходимых для запуска selenium-тестов.

Загружается код приложения, рабочей директорией устанавливается директория с кодом приложения.

В контейнер копируются написанные тесты и файл с перечислением python-зависимостей для этих тестов. Выполняется установка библиотек.

Устанавливается точка входа – http-сервер.

# Описание скриптов запуска тестов

## Проверка на pep8

Запускается скрипт, проверяющий соответствие кода Python стандартам оформления PEP8. Результаты проверки логируются.

## Статический анализ по 10 критериям

В настройке pylint установлена проверка по критериям: main, basic, format, classes, design, imports, typecheck, variables, refactoring, string.

## Интеграционный тест

В интеграционном тесте проверяются коды ответа на различные запросы. Известно, что запрос GET /files должен возвращать код 200 (ок), запрос POST /login с некорректными логином и паролем – код 401 (не авторизован), запрос GET /wrong – код 404 (не найден), запрос GET /login – код 405 (некорректный тип запроса).

## Selenium тест

Задача не была реализована. Для успешной реализации следует сделать так:

Тестирование следует проводить с помощью драйвера браузера Firefox. Производить открытие веб-страницы приложения, запущенного в контейнере app, по короткой ссылке (через сеть docker). Далее с помощью JS-инъекции execute\_script выполнять модификацию HTML страницы – добавляется кнопка с уникальным id. Далее происходит проверка добавленного элемента путем сверки текста, который в нем должен содержаться.

## Запуск тестов

Запуск тестов производится с помощью .sh скриптов.

* start-all.sh– запускает все тесты
* start-pylint.sh– запускает статическую проверку по 10 критериям
* start-integration.sh– запускает интеграционный тест
* start-pep8.sh– запускает pep8 проверку

Вывод логов осуществляется в один файл из двух потоков:

2>&1 | tee -a /proc/1/fd/1 /logs/logs.txt

Также к логам добавляется дата и время.

Используются флаги Pytest для выключения предупреждений, а также вывода не только ошибок, но и списка тестов, которые прошли успешно, а также короткий формат сообщений «-rpP --tb=short».

# Описание Docker-compose конфигурации

Конфигурация docker-compose задается в файле docker-compose.yml. В нем описывается два узла: app и testser.

Узел app:

* Используется образ Docker\_app;
* Подключается файл с переменными среды .env;
* Соотносится внешний порт {PORT} с портом 5000 в контейнере;
* Ограничивается максимальное потребление ОЗУ в 110 мегабайт.

Узел tester:

* Устанавливается зависимость от сервиса app;
* Используется образ Docker\_tester;
* Устанавливается volume для передачи логов на хост машину - ./logs:/logs

# Заключение

В ходе выполнения данной работы были изучены возможности docker и docker-compose. Для этого была реализована конфигурация, состоящая из двух контейнеров: один – для запуска приложения, другой – для запуска тестов.

В ходе работы были реализованы различные виды тестов, решены задачи отображения вывода процессов в контейнере в docker log и ограничения максимального потребления оперативной памяти.

# Список использованных источников

1. Документация по Dockerfile : сайт. – URL: https://docs.docker.com/engine/reference/builder/ (дата обращения: 12.05.2023)
2. Документация по Docker-compose : сайт. – URL: https://docs.docker.com/compose/compose-file/ (дата обращения: 12.05.2023)
3. Документация pytest : сайт. – URL: https://docs.pytest.org/en/7.1.x/contents.html (дата обращения: 12.05.2023)
4. Документация pylint : сайт. – URL: https://docs.pylint.org/ (дата обращения: 12.05.2023)

# Приложение А. Содержание созданных файлов

**Файл docker-compose.yml:**

version: "3"

services:

app:

env\_file:

- .env

ports:

- "${PORT}:5000"

build:

dockerfile: Dockerfile\_app

context: ./app

deploy:

resources:

limits:

memory: 110M

tester:

build:

dockerfile: Dockerfile\_tester

context: ./tester

depends\_on:

- app

volumes:

- ./logs:/logs

**Файл Dockerfile\_app:**

FROM ubuntu:22.04

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

git=1:2.34.1-1ubuntu1.9 \

python3=3.10.6-1~22.04 \

python3-pip=22.0.2+dfsg-1ubuntu0.2 \

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

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

WORKDIR devops-examples/EXAMPLE\_APP

ADD ./app-req.txt requirements.txt

ENV FLASK\_APP main.py

ENV FLASK\_RUN\_HOST 0.0.0.0

RUN pip3 install -r requirements.txt

ENTRYPOINT ["flask", "run", "--host=0.0.0.0"]

**Файл Dockerfile\_tester:**

FROM ubuntu:22.04

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

git=1:2.34.1-1ubuntu1.9 \

python3=3.10.6-1~22.04 \

python3-pip=22.0.2+dfsg-1ubuntu0.2 \

wget=1.21.2-2ubuntu1 \

xvfb=2:21.1.3-2ubuntu2.9 \

devscripts=2.22.1ubuntu1 \

libgtk-3-0=3.24.33-1ubuntu2 \

libdbus-glib-1-2=0.112-2build1 \

libasound2=1.2.6.1-1ubuntu1

RUN wget -O firefox-setup.tar.bz2 "https://download.mozilla.org/?product=firefox-latest&os=linux64" \

&& tar -xvjf firefox-setup.tar.bz2 \

&& ln -s /usr/local/firefox/firefox /usr/bin/firefox

RUN wget https://github.com/mozilla/geckodriver/releases/download/v0.32.2/geckodriver-v0.32.2-linux32.tar.gz \

&& tar -xvzf geckodriver\* \

&& cp geckodriver /usr/bin/geckodriver

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

WORKDIR devops-examples/EXAMPLE\_APP/

ADD ./tester-req.txt requirements.txt

RUN pip3 install -r requirements.txt

ADD tests tests

ENTRYPOINT python3 -m http.server 3000

**Файл start-all.sh:**

#!/bin/sh

annotate-output "+%D %H:%M:%S" pep8 /devops-examples/EXAMPLE\_APP/. 2>&1 | tee /proc/1/fd/1 /logs/logs.txt

annotate-output "+%D %H:%M:%S" pytest --disable-warnings -rpP --tb=short -k "not selenium and not integration" --pylint --pylint-rcfile=./tests/pylintrc . 2>&1 | tee -a /proc/1/fd/1 /logs/logs.txt

annotate-output "+%D %H:%M:%S" pytest --disable-warnings -rpP --tb=short -k "not selenium" . 2>&1 | tee -a /proc/1/fd/1 /logs/logs.txt

**Файл start-integration.sh:**

#!/bin/sh

echo "Start integration"

annotate-output "+%D %H:%M:%S" pytest --disable-warnings -rpP --tb=short -k "not selenium" . 2>&1 | tee -a /proc/1/fd/1 /logs/logs.txt

**Файл start-pep8.sh:**

#!/bin/sh

echo "Start pep8"

annotate-output "+%D %H:%M:%S" pep8 /devops-examples/EXAMPLE\_APP/. 2>&1 | tee /proc/1/fd/1 /logs/logs.txt

**Файл start-pylint.sh:**

#!/bin/sh

echo "Start pylint"

annotate-output "+%D %H:%M:%S" pytest --disable-warnings -rpP --tb=short -k "not selenium and not integration" --pylint --pylint-rcfile=./tests/pylintrc . 2>&1 | tee -a /proc/1/fd/1 /logs/logs.txt

**Файл .pylintrc:**

[MAIN]

ignore=tests

[MESSAGES CONTROL]

disable=all

enable=main, basic, format, classes, design, imports, typecheck, variables, refactoring, string

**Файл integration\_test.py:**

import requests

URL = "http://app:5000"

class TestIntegration:

def test\_200\_status\_code(self):

try:

response = requests.get(f"{URL}/files")

assert response.status\_code == 200

except requests.exceptions.RequestException as e:

assert False, f"Exception occurred: {e}"

def test\_401\_status\_code(self):

try:

response = requests.post(f"{URL}/login", data={'name': 'wrong', 'password': 'wrong'})

assert response.status\_code == 401

except requests.exceptions.RequestException as e:

assert False, f"Exception occurred: {e}"

def test\_404\_status\_code(self):

try:

response = requests.get(f"{URL}/wrong")

assert response.status\_code == 404

except requests.exceptions.RequestException as e:

assert False, f"Exception occurred: {e}"

def test\_405\_status\_code(self):

try:

response = requests.get(f"{URL}/login")

assert response.status\_code == 405

except requests.exceptions.RequestException as e:

assert False, f"Exception occurred: {e}"