Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №18
дисциплины «Основы программной инженерии»

Выполнил:
Матвеев Александр Иванович
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и сопровождение программного обеспечения», очная форма обучения

_____
(подпись)

Проверил Воронкин Роман Александрович
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Основы работы с Dockerfile.

Цель работы: овладеть навыками создания и управления контейнерами Docker для разработки, доставки и запуска приложений. Понимание процесса создания Dockerfile, сборки и развертывания контейнеров Docker, а также оптимизации их производительности и безопасности.

Ход работы.

1. Создание простого веб-приложения на Python с использованием Dockerfile

```
→  docker git:(main) ✗ mkdir web-app
→  docker git:(main) ✗ cd web-app
→  web-app git:(main) ✗ python -m venv .venv
zsh: command not found: python
→  web-app git:(main) ✗ python3 -m venv .venv
```

Рисунок 1 – Установка виртуального окружения

```
→  web-app git:(main) ✗ source .venv/bin/activate
(.venv) →  web-app git:(main) ✗ pip3 install flask
```

Рисунок 2 – Активация venv и установка flask

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from flask import Flask, render_template


app = Flask(__name__, template_folder='templates')


@app.route("/user/<name>")
def hello_world(name):
    return render_template( template_name_or_list: 'index.html', usr=name)


if __name__ == "__main__":
    app.run(debug=True)
```

Рисунок 3 – main.py

```html
<!DOCTYPE html>
<html>
<head>
    <title>User Profile</title>
</head>
<body>
    <h1>Hello, {{ usr }}!</h1>
</body>
</html>
```

Рисунок 4 – index.html

```
FROM python:3.12-slim

RUN mkdir /usr/src/app
COPY . /usr/src/app

WORKDIR /usr/src/app

RUN pip3 install --no-cache-dir -r requirements.txt

EXPOSE 5000

CMD ["python3", "main.py"]
```

Рисунок 5 – Dockerfile

```
(.venv) ➜  web-app git:(main) ✗ docker build -t python-web-app .
[+] Building 35.3s (10/10) FINISHED
```

Рисунок 6 – Создание образа

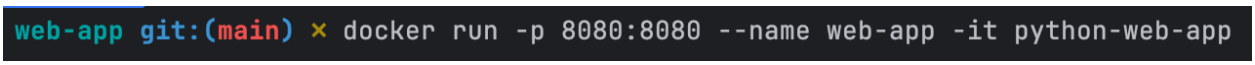

Рисунок 7 – Запуск контейнера

2. Установить дополнительный пакет, например библиотеку NumPy для Python, в образ Docker веб-приложения.

```
FROM python:3.12-slim as builder

RUN mkdir /usr/src/app
COPY . /usr/src/app

WORKDIR /usr/src/app

RUN pip3 install --no-cache-dir -r requirements.txt
RUN pip3 install numpy

FROM python:3.12-slim as runner

WORKDIR /usr/src/app

COPY --from=builder /usr/src/app/. .

EXPOSE 8000

CMD ["python3", "main.py"]
```

Рисунок 8 – Dockerfile

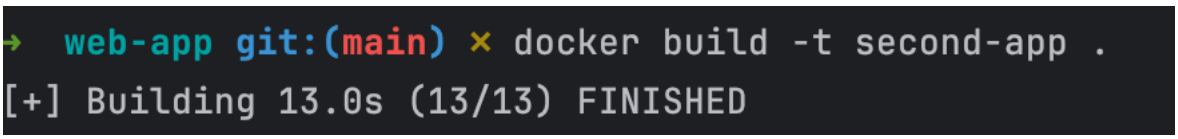

Рисунок 9 – Создание образа

```
→  web-app git:(main) ✗ docker run -p 8000:8000 --name python-second-app -it second-app
 * Serving Flask app 'main'
 * Debug mode: on
```

Рисунок 10 – Запуск контейнера

3. Настроить переменную среды, например URL базы данных, в образе Docker веб-приложения. Используйте команду ENV в Dockerfile для определения переменной среды и сделайте ее доступной для приложения.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os


def main():
    print(os.getenv("DATABASE_URL"))


if __name__ == "__main__":
    main()
```

Рисунок 11 – Файл main.py

```
FROM python:3.12-slim

RUN mkdir /usr/src/app
COPY . /usr/src/app

WORKDIR /usr/src/app

ENV DATABASE_URL="//user:password@localhost:5432/database"

CMD ["python3", "main.py"]
```

Рисунок 12 – Dockerfile

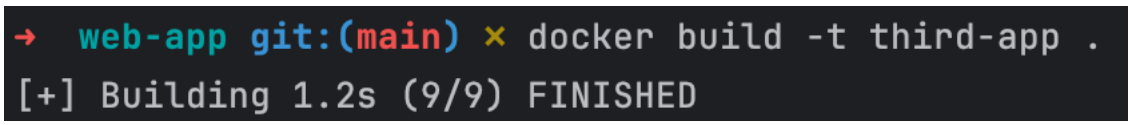

Рисунок 13 – Создание образа

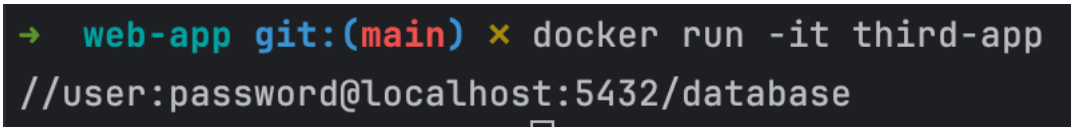

Рисунок 14 – Запуск контейнера

Контрольные вопросы:

1. Dockerfile — это текстовый файл, содержащий все команды, которые пользователь может вызвать в командной строке для создания образа Docker.

2. Основные команды Dockerfile включают: FROM: Эта строка указывает базовый образ, который будет использоваться для сборки нового образа. WORKDIR: Эта строка устанавливает рабочую директорию для контейнера. COPY: Эта строка копирует файлы из хоста в образ Docker. RUN: Эта строка выполняет команды в процессе сборки образа. EXPOSE: Эта строка указывает порты, которые должны быть открыты в контейнере. CMD: Эта строка указывает команду, которая будет выполняться при запуске контейнера.

3. FROM используется для указания базового образа, от которого начинается процесс сборки.

4. WORKDIR устанавливает рабочую директорию для инструкций RUN, CMD, ENTRYPOINT, COPY и ADD.

5. COPY копирует файлы и директории из контекста сборки в образ.

6. RUN выполняет команды внутри образа и создает новый слой образа с результатами.

7. CMD задает команду по умолчанию, которая выполняется при запуске контейнера.

8. EXPOSE указывает на то, что контейнер прослушивает определенный порт сети.

9. ENV устанавливает переменную среды в образе.

10. USER меняет пользователя, под которым выполняются последующие команды и при запуске контейнера.

11. HEALTHCHECK позволяет Docker определять состояние контейнера.

12. LABEL добавляет метаданные к образу.

13. ARG определяет переменные, которые могут быть переданы Docker во время сборки образа.

14. ONBUILD определяет команду, которая будет исполнена в дочернем образе, основанном на текущем образе.

15. Многоэтапная сборка в Docker позволяет использовать несколько FROM инструкций в одном Dockerfile для создания промежуточных образов, что улучшает кэширование и уменьшает размер конечного образа.

16. Преимущества многоэтапной сборки: Уменьшение размера конечного образа. Изоляция стадий сборки и запуска.

17. Недостатки многоэтапной сборки: Может быть более сложной в настройке и управлении. Возможное увеличение времени сборки из-за необходимости управления несколькими стадиями.

18. Базовый образ в Dockerfile определяется с помощью инструкции FROM.

19. Рабочая директория в Dockerfile устанавливается с помощью инструкции WORKDIR.

20. Копирование файлов в образ Docker производится с помощью инструкции COPY.

21. Выполнение команд при сборке образа Docker производится с помощью инструкции RUN.

22. Команда запуска контейнера указывается с помощью CMD.

23. Открытие портов в контейнере осуществляется с помощью EXPOSE.

24. Задание переменных среды в образе Docker выполняется с помощью ENV.

25. Изменение пользователя осуществляется с помощью USER.

26. Добавление проверки работоспособности к контейнеру выполняется с помощью HEALTHCHECK.

27. Добавление метки к контейнеру осуществляется через LABEL.

28. Передача аргументов при сборке образа Docker производится с помощью ARG.

29. Выполнение команды при первом запуске контейнера задается с помощью CMD или ENTRYPOINT.

30. Зависимости между образами Docker определяются в Docker Compose файле или через скрипты сборки, где одни образы используются в качестве базы (в инструкции FROM) для других.