МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

«Основы работы с Dockerfile»

Отчет по лабораторной работе

по дисциплине «Анализ данных»

Выполнил студент группы ИВТ-б-о-21-1

Стригалов Дмитрий.

« » _____ 2023г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2023

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.

Порядок выполнения работы:

Задача 1: Создание простого веб-приложения на Python с использованием Dockerfile

Цель: Создать простое веб-приложение на Python, которое принимает имя пользователя в качестве параметра URL и возвращает приветствие с именем пользователя. Используйте Dockerfile для сборки образа Docker вашего приложения и запустите контейнер из этого образа.

Описание:

- Создайте проект веб-приложения на Python, включая код приложения и необходимые файлы.

```
root@ReviOS:~/Docker3# mkdir python-web-app
root@ReviOS:~/Docker3# cd python-web-app
```

```
root@ReviOS:~/Docker3/python-web-app# python3 -m venv .venv
root@ReviOS:~/Docker3/python-web-app# source .venv/bin/activate
(.venv) root@ReviOS:~/Docker3/python-web-app# pip install Flask
Collecting Flask
  Downloading flask-3.0.0-py3-none-any.whl (99 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.7/99.7 KB 1.3 MB/s eta 0:00:00
Collecting blinker>=1.6.2
  Downloading blinker-1.7.0-py3-none-any.whl (13 kB)
Collecting itsdangerous>=2.1.2
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting Jinja2>=3.1.2
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 133.1/133.1 KB 4.7 MB/s eta 0:00:00
Collecting click>=8.1.3
  Downloading click-8.1.7-py3-none-any.whl (97 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 97.9/97.9 KB 6.6 MB/s eta 0:00:00
Collecting Werkzeug>=3.0.0
  Downloading werkzeug-3.0.1-py3-none-any.whl (226 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 226.7/226.7 KB 9.2 MB/s eta 0:00:00
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.1.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (25 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, Flask
Successfully installed Flask-3.0.0 Jinja2-3.1.2 MarkupSafe-2.1.3 Werkzeug-3.0.1 blinker-1.7.0 click-8.1.7 itsdangerous-2.1.2
```

```
(.venv) root@ReviOS:~/Docker3/python-web-app# pip freeze > .\requirements.txt
```

Рисунок 1 - Настройка виртуального окружения и установка Flask

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


from pathlib import Path
from flask import Flask, render_template, request


app = Flask(__name__, template_folder=str(Path(__file__).parent))


@app.route("/users/", methods=['GET','POST'])
def greet_user():
        user_name = request.args.get('user')
        return render_template("index.html", message = f"Hello, {user_name}!")


if __name__ == "__main__":
        app.run(host="0.0.0.0")
```

Рисунок 2 - Содержимое файла app.py

```html
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title>FlaskApp</title>
</head>
<body>
        <h2>{{ message }}</h2>
</body>
</html>
```

Рисунок 3 - Содержимое index.html

- Создайте Dockerfile для сборки образа Docker вашего приложения.
- Определите инструкции для сборки образа, включая копирование файлов, установку зависимостей и настройку команд запуска.

```
FROM python:3.10-slim

RUN mkdir /usr/src/app
COPY ./my-app /usr/src/app
COPY ./requirements.txt /usr/src/app

WORKDIR /usr/src/app

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

EXPOSE 5000

CMD [ "python", "app.py" ]
```

Рисунок 4 - Содержимое Dockerfile

- Соберите образ Docker с помощью команды docker build .

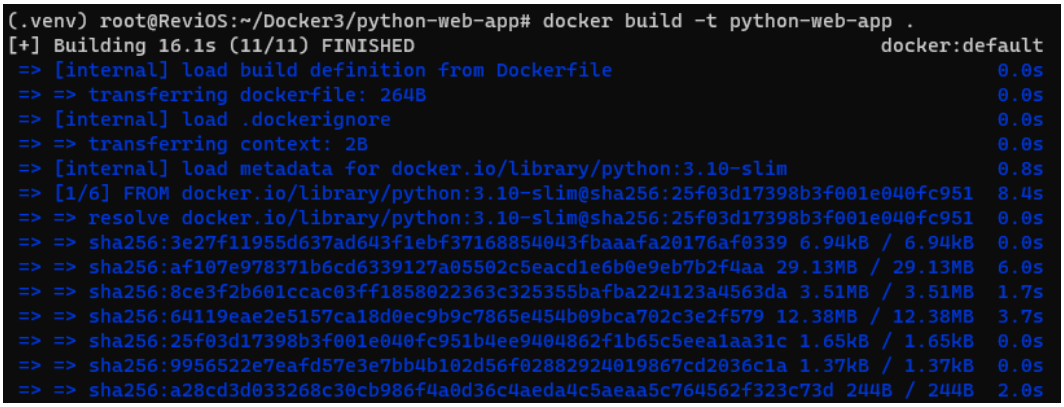

Рисунок 5 - Сборка образа при помощи docker build

- Запустите контейнер из образа Docker с помощью команды docker run.

```
(.venv) root@ReviOS:~/Docker3/python-web-app# docker run -p 5000:5000 --name my-python-app -d py
thon-web-app
b99b9308fbd3d933a9a1dae91925d77ab2ce0ffd3036a44b2dd2108fdcf68f19
```

Рисунок 6 – Запуск контейнера

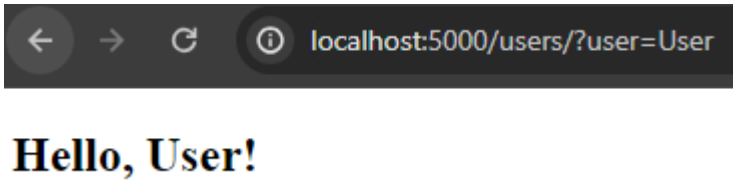

Рисунок 7 – Успешно запущенное приложение

Задача 2: Установка дополнительных пакетов в образ Docker

Цель: установить дополнительный пакет, например библиотеку NumPy для Python, в образ Docker веб-приложения.

Описание:

- Создайте многоэтапной Dockerfile, состоящий из двух этапов: этап сборки и этап выполнения.
- На этапе сборки установите дополнительный пакет, такой как библиотеку NumPy, используя команду RUN.
- На этапе выполнения скопируйте созданное приложение из этапа сборки и укажите команду запуска.

```
FROM python:3.10 as builder

WORKDIR /app

RUN pip install numpy
RUN mkdir /usr/src/app
COPY ./my-app /usr/src/app
COPY ./requirements.txt /usr/src/app

FROM python:3.10-slim as runner

WORKDIR /app

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

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

EXPOSE 5000

CMD [ "python", "app.py" ]
```

Рисунок 8 - Многоэтапный Dockerfile

- Соберите образ Docker с помощью команды docker build .
- Запустите контейнер из образа Docker с помощью команды docker run.

```
(.venv) root@ReviOS:~/Docker3/python-web-app# docker build -t python-web-app .
[+] Building 5.5s (16/16) FINISHED                                              docker:default
 => [internal] load .dockerignore                                                        0.1s
 => => transferring context: 2B                                                          0.0s
 => [internal] load build definition from Dockerfile                                     0.0s
 => => transferring dockerfile: 369B                                                     0.0s
 => [internal] load metadata for docker.io/library/python:3.10-slim                      0.9s
 => [internal] load metadata for docker.io/library/python:3.10                           0.8s
 => [builder 1/6] FROM docker.io/library/python:3.10@sha256:ba7e6f1feea05621dec8a6525e1bb  0.0s
 => [internal] load build context                                                        0.0s
 => => transferring context: 133B                                                        0.0s
 => [runner 1/4] FROM docker.io/library/python:3.10-slim@sha256:25f03d17398b3f001e040fc95  0.0s
 => CACHED [builder 2/6] WORKDIR /app                                                    0.0s
 => CACHED [builder 3/6] RUN pip install numpy                                           0.0s
 => CACHED [builder 4/6] RUN mkdir /usr/src/app                                          0.0s
 => CACHED [builder 5/6] COPY ./my-app /usr/src/app                                      0.0s
 => CACHED [builder 6/6] COPY ./requirements.txt /usr/src/app                            0.0s
 => CACHED [runner 2/4] WORKDIR /app                                                     0.0s
 => [runner 3/4] COPY --from=builder /usr/src/app/. .                                    0.2s
 => [runner 4/4] RUN pip install --no-cache-dir -r requirements.txt                      4.1s
 => exporting to image                                                                   0.1s
 => => exporting layers                                                                  0.1s
 => => writing image sha256:4a49d84670be427c38fa73180e5c903c09cedcb5acd602482601c69421c47  0.0s
```

```
(.venv) root@ReviOS:~/Docker3/python-web-app# docker run -p 5000:5000 --name my-python-app -d py
thon-web-app
da9a30fb37a326529f130289a69804ca11a9bfc8397318e6fb19a688cc04f482
```

Рисунок 9 - Сборка образа и запуск контейнера

Задача 3: Настройка переменных среды в образе Docker

Цель: 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.

Определите переменную среды, такую как URL базы данных, в Dockerfile с помощью команды ENV.

```
FROM python:3.10

WORKDIR /app

ENV IGNITE_DATABASE_URL jdbc:ignite:thin://localhost:10800/

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

Рисунок 10 - Содержимое Dockerfile

- Запустите контейнер из образа Docker с помощью команды docker run.

```
(.venv) root@ReviOS:~/Docker3/python-web-app# docker build -t python-db-app .
[+] Building 2.4s (6/6) FINISHED                                    docker:default
 => [internal] load .dockerignore                                           0.0s
 => => transferring context: 2B                                             0.0s
 => [internal] load build definition from Dockerfile                        0.0s
 => => transferring dockerfile: 170B                                        0.0s
 => [internal] load metadata for docker.io/library/python:3.10              2.3s
 => [1/2] FROM docker.io/library/python:3.10@sha256:ba7e6f1feea05621dec8a6525e1bb9edf30a3  0.0s
 => CACHED [2/2] WORKDIR /app                                               0.0s
 => exporting to image                                                      0.0s
 => => exporting layers                                                     0.0s
 => => writing image sha256:d7e85900ef7a49a74275fd7ce9f19a5b5bd02ccd089c94a48875810db9a1a  0.0s
 => => naming to docker.io/library/python-db-app                            0.0s
```

Рисунок 11 - Сборка образа и запуск контейнера

Задача 4: Копирование файлов в образ Docker

Цель: скопировать необходимые файлы, такие как статические файлы или конфигурационные файлы, в образ Docker веб-приложения.

Описание:

- Определите файлы для копирования в образ Docker с помощью команды COPY в Dockerfile.
- Укажите исходное расположение файлов и их местоположение в образе.

```
FROM python:3.10

WORKDIR /app

ENV IGNITE_DATABASE_URL jdbc:ignite:thin://localhost:10800/

COPY my-app ./

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

Рисунок 12 - Содержимое Dockerfile

Соберите образ Docker с помощью команды docker build .

- Запустите контейнер из образа Docker с помощью команды docker run.

```
(.venv) root@ReviOS:~/Docker3/python-web-app# docker build -t python-db-app .
[+] Building 2.4s (8/8) FINISHED                                                docker:default
 => [internal] load .dockerignore                                                         0.0s
 => => transferring context: 2B                                                           0.0s
 => [internal] load build definition from Dockerfile                                      0.0s
 => => transferring dockerfile: 174B                                                      0.0s
 => [internal] load metadata for docker.io/library/python:3.10                            2.3s
 => [1/3] FROM docker.io/library/python:3.10@sha256:ba7e6f1feea05621dec8a6525e1bb9edf30a3  0.0s
 => [internal] load build context                                                         0.0s
 => => transferring context: 98B                                                          0.0s
 => CACHED [2/3] WORKDIR /app                                                             0.0s
 => CACHED [3/3] COPY my-app ./                                                           0.0s
 => exporting to image                                                                    0.0s
 => => exporting layers                                                                   0.0s
```

```
(.venv) root@ReviOS:~/Docker3/python-web-app# docker run --name python-db -d python-db-app
cb69bb6e691c59b9c8941c39f598e87eb72cb620cd0fa021869739096397108b
```

Рисунок 13 - Сборка образа и запуск контейнера

Задача 5: Запуск команд при запуске контейнера

Цель: выполнить команды инициализации или настройки при запуске контейнера веб-приложения. Используйте команду RUN в Dockerfile для определения команд для выполнения и их параметров.

Описание:

- Определите команды для выполнения при запуске контейнера с помощью команды RUN в Dockerfile.
- Укажите команды и их параметры, например, создание конфигурационных файлов или выполнение скриптов инициализации.

```
FROM python:3.10 as builder

WORKDIR /app

RUN pip install numpy
RUN mkdir /usr/src/app
COPY ./my-app /usr/src/app
COPY ./requirements.txt /usr/src/app

FROM python:3.10-slim as runner

WORKDIR /app

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

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

EXPOSE 5000

CMD [ "python", "app.py" ]
```

Рисунок 14 - Dockerfile

Соберите образ Docker с помощью команды docker build.

- Запустите контейнер из образа Docker с помощью команды docker run.

```
(.venv) root@ReviOS:~/Docker3/python-web-app# docker build -t python-web-app .
[+] Building 1.1s (16/16) FINISHED                                                docker:default
 => [internal] load .dockerignore                                                          0.1s
 => => transferring context: 2B                                                            0.0s
 => [internal] load build definition from Dockerfile                                       0.0s
 => => transferring dockerfile: 369B                                                       0.0s
 => [internal] load metadata for docker.io/library/python:3.10-slim                        0.8s
 => [internal] load metadata for docker.io/library/python:3.10                             0.8s
 => [internal] load build context                                                          0.0s
 => => transferring context: 133B                                                          0.0s
 => [runner 1/4] FROM docker.io/library/python:3.10-slim@sha256:25f03d17398b3f001e040fc95  0.0s
 => [builder 1/6] FROM docker.io/library/python:3.10@sha256:ba7e6f1feea05621dec8a6525e1bb  0.0s
 => CACHED [runner 2/4] WORKDIR /app                                                       0.0s
 => CACHED [builder 2/6] WORKDIR /app                                                      0.0s
 => CACHED [builder 3/6] RUN pip install numpy                                             0.0s
 => CACHED [builder 4/6] RUN mkdir /usr/src/app                                            0.0s
 => CACHED [builder 5/6] COPY ./my-app /usr/src/app                                        0.0s
```

```
(.venv) root@ReviOS:~/Docker3/python-web-app# docker run --name python-app -d python-web-app
52e9b13dc6961f29550f226f7a53e93a3a59992be4f542807de9c713b829fcfa
```

Рисунок 15 - Сборка образа и запуск контейнера

Контрольные вопросы:

1. Что такое Dockerfile?

Dockerfile - это текстовый файл, который содержит инструкции для автоматизированного создания образа Docker. Dockerfile определяет, какие

операции и конфигурации должны быть выполнены внутри контейнера при его создании.

2. Какие основные команды используются в Dockerfile?

FROM - Указывает базовый образ, на основе которого будет создан новый образ.

RUN - Выполняет команды в процессе создания образа.

CMD - Указывает команду, которая будет выполняться при запуске контейнера из образа.

COPY - Копирует файлы из хоста в образ.

EXPOSE - Указывает порты, которые будут открыты в контейнере.

3. Для чего используется команда FROM?

FROM: Эта строка указывает базовый образ, который будет использоваться для сборки нового образа.

4. Для чего используется команда WORKDIR?

WORKDIR: Эта строка устанавливает рабочую директорию для контейнера.

5. Для чего используется команда COPY?

COPY: Эта строка копирует файлы из хоста в образ Docker.

6. Для чего используется команда RUN?

RUN: Эта строка выполняет команды в процессе сборки образа.

7. Для чего используется команда CMD?

CMD: Эта строка указывает команду, которая будет выполняться при запуске контейнера.

8. Для чего используется команда EXPOSE?

EXPOSE: Эта строка указывает порты, которые должны быть открыты в контейнере.

9. Для чего используется команда ENV?

Самый простой способ настроить переменную среды в образе Docker – это использовать команду EN .

10. Для чего используется команда USER?

Команда USER в Dockerfile указывает пользователя, от имени которого будет выполняться основная команда контейнера (CMD или ENTRYPOINT).

11. Для чего используется команда HEALTHCHECK?

HEALTHCHECK — инструкции, которые Docker может использовать для проверки работоспособности запущенного контейнера.

12. Для чего используется команда LABEL?

LABEL — описывает метаданные. Например — сведения о том, кто создал и поддерживает образ.

13. Для чего используется команда ARG?

ARG — задаёт переменные для передачи Docker во время сборки образа.

14. Для чего используется команда ONBUILD?

Инструкция ONBUILDдобавляет к образу инструкцию-триггер, которая будет выполнена позже, когда образ будет использоваться в качестве основы для другой сборки.

15. Что такое многоэтапная сборка?

Многоэтапный Dockerfile состоит из двух основных этапов:

1. Этап сборки: Этот этап отвечает за компиляцию и сборку приложения. Он использует базовый образ с необходимыми инструментами для сборки, такими как компилятор Golang и соответствующие зависимости.

2. Этап выполнения: Этот этап отвечает за запуск и выполнение приложения. Он использует более минимальный базовый образ, например, Alpine Linux, содержащий только необходимые библиотеки для выполнения приложения.

16. Какие преимущества использования многоэтапной сборки?

Уменьшение размера образа: при использовании многоэтапных сборок только необходимые файлы для выполнения приложения включаются в окончательный образ, что значительно уменьшает его размер.

Повышение безопасности: Многоэтапные сборки уменьшают риск уязвимостей безопасности, поскольку они изолируют этапы сборки и выполнения, ограничивая доступ к ненужным инструментам и зависимостям.

17. Какие недостатки использования многоэтапной сборки?

Сложность конфигурации и поддержки процесса сборки может возрасти. Необходимо следить за последовательностью этапов, управлять зависимостями и обеспечивать корректное выполнение каждого этапа. Это требует дополнительных знаний и времени на настройку, особенно для больших и сложных проектов.

Кроме того, многоэтапная сборка Docker требует наличия основного образа системы, который может быть достаточно большим и содержать лишние компоненты. Это может увеличить размер окончательного образа, что негативно отразится на скорости его развертывания и потреблении ресурсов.

18. Как определить базовый образ в Dockerfile?

FROM node:latest

19. Как определить рабочую директорию в Dockerfile?

WORKDIR /usr/src/app

20. Как скопировать файлы в образ Docker?

COPY ./my-app /usr/src/app/

21. Как выполнить команды при сборке образа Docker?

Команда RUN выполняет команды в процессе создания образа.

22. Как указать команду запуска контейнера?

Команда CMD в Dockerfile указывает команду, которая будет выполняться при запуске контейнера. Она может состоять из одной или нескольких команд, разделенных пробелами

Команда ENTRYPOINT в Dockerfile указывает исполняемый файл, который будет использоваться в качестве основной точки входа в контейнер.

23. Как открыть порты в контейнере?

Запуск образа с флагом -p перенаправляет общедоступный порт на частный порт внутри контейнера.

24. Как задать переменные среды в образе Docker?

Самый простой способ настроить переменную среды в образе Docker – это использовать команду ENV .

Вы также можете настроить переменные среды в образе Docker с помощью файла .env. Чтобы использовать файл .env для настройки переменных среды в образе Docker, вы должны добавить команду ADD .env /app/.env в Dockerfile.

Вы также можете настроить переменные среды при запуске контейнера. Для этого используйте флаг --env или -e .

25. Как изменить пользователя, от имени которого будет выполняться контейнер?

При помощи команды USER.

26. Как добавить проверку работоспособности к контейнеру?

При помощи команды HEALTHCHECK.

27. Как добавить метку к контейнеру?

При помощи команды LABEL.

28. Как передать аргументы при сборке образа Docker?

При помощи команды ARG.

29. Как выполнить команду при первом запуске контейнера?

При помощи команды ENTRYPOINT.

30. Как определить зависимости между образами Docker? При помощи команды ONBUILD.