

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

Мальцев Николай Артемович.

«3» декабря 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.

Ход работы:

Создание простого web-приложения на Python с использованием Dockerfile.

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

Настраиваем виртуальное окружение:

```
(env) root@LAPTOP-FN5I7B17:/home/nikolaym/python_web/AD_3_Docker/py_web# pip install Flask
Collecting Flask
  Downloading flask-3.0.0-py3-none-any.whl (99 kB)
    99.7/99.7 KB 269.6 kB/s eta 0:00:00
Collecting itsdangerous>=2.1.2
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting click>=8.1.3
  Downloading click-8.1.7-py3-none-any.whl (97 kB)
    97.9/97.9 KB 170.8 kB/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 253.5 kB/s eta 0:00:00
Collecting blinker>=1.6.2
  Downloading blinker-1.7.0-py3-none-any.whl (13 kB)
Collecting Jinja2>=3.1.2
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
    133.1/133.1 KB 362.7 kB/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
(env) root@LAPTOP-FN5I7B17:/home/nikolaym/python_web/AD_3_Docker/py_web# pip freeze > .\requirements.txt
(env) root@LAPTOP-FN5I7B17:/home/nikolaym/python_web/AD_3_Docker/py_web# |
```

Рисунок 1. Настройка виртуального окружения

Код программы на Python (файл app.py):

```
# app.py
from flask import Flask, request

app = Flask(__name__)

@app.route('/')
def hello():
    user_name = request.args.get('name', 'Guest')
    return f'Hello, {user_name}!'
```

```
if __name__ == '__main__':  
    app.run(host='0.0.0.0', port=5000)
```

Код html:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <title>FlaskApp</title>  
</head>  
<body>  
    <h1>{{ message }}</h1>  
</body>  
</html>
```

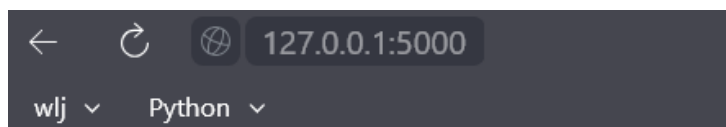
Создайте Dockerfile для сборки образа Docker вашего приложения.

Определите инструкции для сборки образа, включая копирование файлов, установку зависимостей и настройку команд запуска.

Код Dockerfile:

```
FROM python:3.10-slim as builder  
WORKDIR /usr/src/app  
RUN pip install numpy  
COPY . /usr/src/app  
COPY ./requirements.txt /usr/src/app  
FROM python:3.10-slim as runner  
WORKDIR /usr/src/app  
ENV DATABASE_URL postgres://user:password@localhost:5432/database  
COPY --from=builder /usr/src/app/. .  
RUN pip install --no-cache-dir -r requirements.txt  
EXPOSE 5000  
CMD [ "python", "app.py" ]
```

Результат работы программы:



Hello, nikolaym!

Рисунок 2. Результат работы программы в браузере

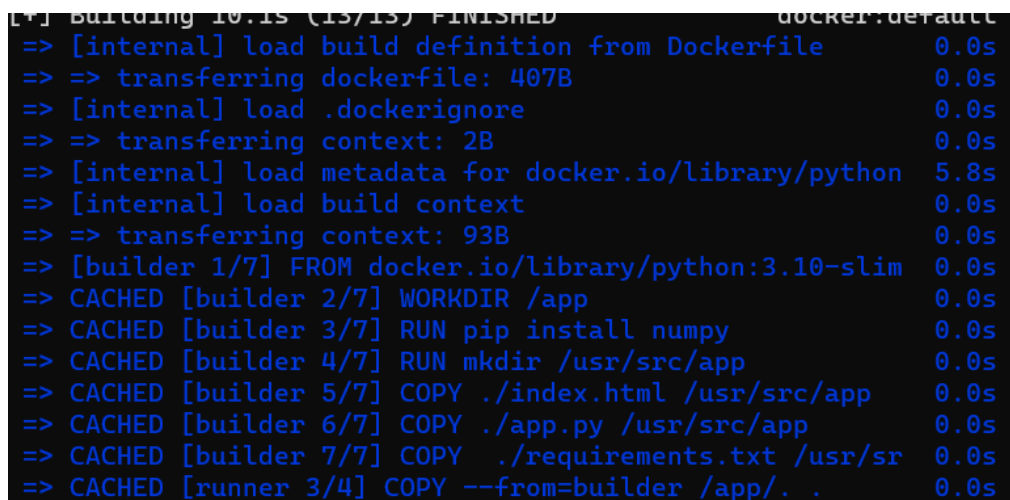
Установка дополнительных пакетов в образ Docker.

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.

Код Dockerfile:

```
# Dockerfile
FROM python:3.10-slim
WORKDIR /app
RUN pip install numpy
RUN mkdir /usr/src/app
COPY ./index.html /usr/src/app
COPY ./app.py /usr/src/app
COPY ./requirements.txt /usr/src/app
FROM python:3.10-slim as runner
WORKDIR /usr/src/app
COPY --from=builder /app/. .
RUN pip install --no-cache-dir -r requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

Сборка и запуск контейнера:



```
[+] Building 10.1s (13/13) FINISHED docker:default
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 407B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/python 5.8s
=> [internal] load build context 0.0s
=> => transferring context: 93B 0.0s
=> [builder 1/7] FROM docker.io/library/python:3.10-slim 0.0s
=> CACHED [builder 2/7] WORKDIR /app 0.0s
=> CACHED [builder 3/7] RUN pip install numpy 0.0s
=> CACHED [builder 4/7] RUN mkdir /usr/src/app 0.0s
=> CACHED [builder 5/7] COPY ./index.html /usr/src/app 0.0s
=> CACHED [builder 6/7] COPY ./app.py /usr/src/app 0.0s
=> CACHED [builder 7/7] COPY ./requirements.txt /usr/src/app 0.0s
=> CACHED [runner 3/4] COPY --from=builder /app/. . 0.0s
```

Рисунок 3. Сборка контейнера

Настройка переменных окружения среды в образе Docker.

Изменим Dockerfile:

```
# Dockerfile
```

```
FROM python:3.10
WORKDIR /app
ENV DATABASE_URL arangodb://user:password@localhost:8529
CMD ["python", "app.py"]
```

Сборка контейнера:

```
[+] Building 95.3s (6/6) FINISHED          docker:default
=> [internal] load build definition from Dockerfile      0.0s
=> => transferring dockerfile: 162B                      0.0s
=> [internal] load .dockerignore                        0.0s
=> => transferring context: 2B                            0.0s
=> [internal] load metadata for docker.io/library/pytho 11.4s
=> [1/2] FROM docker.io/library/python:3.10@sha256:ba7e 82.7s
=> => resolve docker.io/library/python:3.10@sha256:ba7e6 0.1s
=> => sha256:ba7e6f1feea05621dec8a6525e1 1.65kB / 1.65kB 0.0s
=> => sha256:c9a36876f4e4c133e47ecde425a 7.53kB / 7.53kB 0.0s
=> => sha256:b3f22817b332547c7757a39f47f 2.01kB / 2.01kB 0.0s
=> => sha256:bc0734b949dcdcab5bdfdf0c 49.56MB / 49.56MB 23.9s
=> => sha256:b5de22c0f5cd2ea2bb6c0524 24.05MB / 24.05MB 14.3s
=> => sha256:917ee5330e73737d6095a802 64.13MB / 64.13MB 32.7s
=> => sha256:b43bd898d5f5be0e1606380 211.10MB / 211.10MB 66.7s
=> => sha256:7fad4bfffde2444237b82386b9b 6.39MB / 6.39MB 28.4s
=> => extracting sha256:bc0734b949dcdcab5bdfdf0c8b9f4449 6.1s
=> => sha256:6ae8bd1942a2f1d9fb2331d0 17.15MB / 17.15MB 36.4s
=> => extracting sha256:b5de22c0f5cd2ea2bb6c0524478db95b 1.0s
=> => extracting sha256:917ee5330e73737d6095a802333d3116 6.4s
=> => sha256:7a7849a795a3b4bb5bc8c71e479572 242B / 242B 33.0s
=> => sha256:e2f45fa6aa0ffb40284233b2ff 3.08MB / 3.08MB 35.5s
=> => extracting sha256:b43bd898d5f5be0e1606380820047fd1 13.4s
=> => extracting sha256:7fad4bfffde2444237b82386b9b704d8a 0.5s
=> => extracting sha256:6ae8bd1942a2f1d9fb2331d04819e196 0.8s
=> => extracting sha256:7a7849a795a3b4bb5bc8c71e4795722e 0.0s
=> => extracting sha256:e2f45fa6aa0ffb40284233b2ff7959d8 0.4s
=> [2/2] WORKDIR /app                                0.9s
```

Рисунок 4. Сборка 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 ENV.

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 ONBUILD добавляет 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.

Вывод: в ходе работы были освоены навыки создания и управления контейнерами Docker для разработки, доставки и запуска приложений. Понимание процесса создания Dockerfile, сборки и развёртывания контейнеров Docker, а также оптимизации их производительности и безопасности.