

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 № 5**  
«Основы работы с Dockerfile»

Выполнила:  
Кузнецова Алена Валерьевна  
3 курс, группа ИВТ-б-о-21-1,  
09.03.01 «Информатика  
и вычислительная техника»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной техники и  
автоматизированных систем»,  
очная форма обучения

---

(подпись)

Ставрополь, 2023 г.

**Цель занятия:** 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.

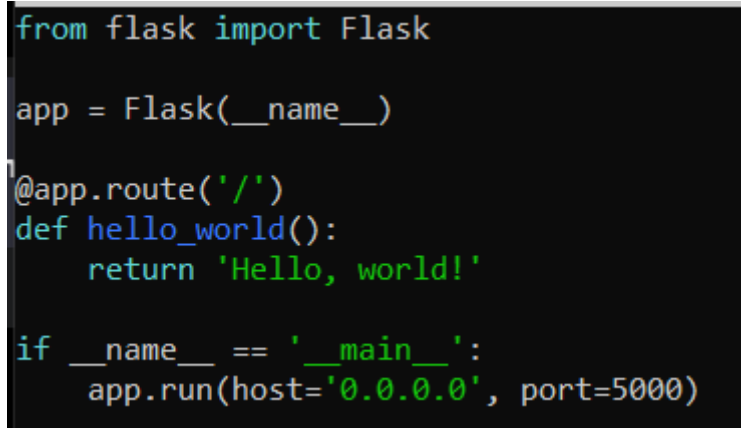
**Выполнение работы:**

**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.

A screenshot of a code editor with a dark background and light-colored text. The code is written in Python and uses syntax highlighting. It defines a Flask application with a single route for the root URL that returns 'Hello, world!'. The application is configured to run on all interfaces (0.0.0.0) at port 5000.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, world!'

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

Рисунок 1 – Файл app.py

```

alena@LAPTOP-ARKAM4ED:~$ pip install flask
Defaulting to user installation because normal site-packages is not writeable
Collecting flask
  Downloading flask-3.0.0-py3-none-any.whl (99 kB)
    99.7/99.7 KB 270.0 kB/s eta 0:00:00
Collecting blinker>=1.6.2
  Downloading blinker-1.7.0-py3-none-any.whl (13 kB)
Collecting Werkzeug>=3.0.0
  Downloading werkzeug-3.0.1-py3-none-any.whl (226 kB)
    226.7/226.7 KB 337.6 kB/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 351.1 kB/s eta 0:00:00
Collecting Jinja2>=3.1.2
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
    133.1/133.1 KB 414.5 kB/s eta 0:00:00
Collecting itsdangerous>=2.1.2
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
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
WARNING: The script flask is installed in '/home/alena/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed Jinja2-3.1.2 MarkupSafe-2.1.3 Werkzeug-3.0.1 blinker-1.7.0 click-8.1.7 flask-3.0.0 itsdangerous-2.1.2
alena@LAPTOP-ARKAM4ED:~$

```

Рисунок 2 – Установка Flask

```

alena@LAPTOP-ARKAM4ED:~/my-web-app$ pip freeze > ./requirements.txt
alena@LAPTOP-ARKAM4ED:~/my-web-app$ ls

```

Рисунок 3 – Создание файла requirements.txt

Создайте Dockerfile для сборки образа Docker вашего приложения.

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

```

# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Make port 5000 available to the world outside this container
EXPOSE 5000

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]

```

Рисунок 4 – Dockerfile

Соберите образ Docker с помощью команды `docker build`.

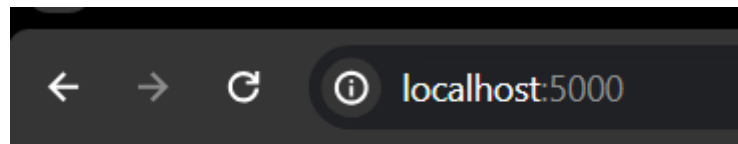
```
alena@LAPTOP-ARKAM4ED:~/my-web-app$ docker build -t my_web_app .
[+] Building 19.7s (10/10) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.1s
=> => transferring dockerfile: 672B                                0.0s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                       0.0s
=> [internal] load metadata for docker.io/library/python:3.9-slim 1.6s
=> [1/5] FROM docker.io/library/python:3.9-slim@sha256:96be08c44307e781fd9ce8e05b49c969b4cb902ec23594f904739c58d 0.0s
```

Рисунок 5 – Сборка образа

Запустите контейнер из образа Docker с помощью команды `docker run`

```
alena@LAPTOP-ARKAM4ED:~/my-web-app$ docker run -p 5000:5000 my_web_app
* Serving Flask app main (new loading) *
```

Рисунок 6 – Запуск контейнера



Hello, World!

Рисунок 7 – Результат

**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.

```

# Этап сборки
FROM python:3.8-slim as builder

RUN pip install --no-cache-dir numpy Flask

# Установка приложения
WORKDIR /app
COPY . /app

# Этап выполнения
FROM python:3.8-slim

# Копирование приложения из этапа сборки
COPY --from=builder /app /app

# Указание рабочей директории
WORKDIR /app

# Команда запуска приложения
CMD ["python", "app.py"]

```

Рисунок 8 - Dockerfile

Соберите образ Docker с помощью команды `docker build`.

```

alena@LAPTOP-ARKAM4ED:~/AD-5/my-web-app$ docker build -t my-python-app .
[+] Building 3.1s (11/11) FINISHED                                docker:default
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                    0.0s
=> [internal] load build definition from Dockerfile              0.0s
=> => transferring dockerfile: 516B                               0.0s

```

Рисунок 9 – Сборка образа

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

```

alena@LAPTOP-ARKAM4ED:~/AD-5/my-web-app$ docker run -p 5000:5000 --name my-python-app -d my-python-app
62db33bdc3fa523dd5bce0cdae43136d4b8d894eee3b97d51c94654ec448e54f

```

Рисунок 10 – 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.

#### Описание:

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

```

# Этап сборки
FROM python:3.8-slim as builder

# Установка приложения и других зависимостей
WORKDIR /app
COPY . /app

# Определение переменной среды
ENV DB_URL="your_database_url"

# Этап выполнения
FROM python:3.8-slim

# Копирование приложения из этапа сборки
COPY --from=builder /app /app

# Указание рабочей директории
WORKDIR /app

# Команда запуска приложения
CMD ["python", "app.py"]

```

Рисунок 11 – Dockerfile

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

Доступ к переменной среды из приложения с помощью соответствующей переменной окружения.

```

alena@LAPTOP-ARKAM4ED:~/AD-5/my-python-app$ docker run -e DB_URL="actual_database_url" my-python-app
Database URL: actual_database_url

```

Рисунок 12 – Результат

#### Задача 4: Копирование файлов в образ Docker

**Цель:** скопировать необходимые файлы, такие как статические файлы или конфигурационные файлы, в образ Docker веб-приложения. Используйте команду `COPY` в Dockerfile для определения файлов для копирования и их местоположения в образе.

##### Описание:

Определите файлы для копирования в образ Docker с помощью команды `COPY` в Dockerfile.

Укажите исходное расположение файлов и их местоположение в образе.

```

# Этап сборки
FROM python:3.8-slim as builder

# Копирование файлов приложения
COPY app.py /app/
COPY requirements.txt /app/

# Установка зависимостей
WORKDIR /app
RUN pip install --no-cache-dir -r requirements.txt

# Этап выполнения
FROM python:3.8-slim

# Копирование приложения из этапа сборки
COPY --from=builder /app /app

# Указание рабочей директории
WORKDIR /app

# Команда запуска приложения
CMD ["python", "app.py"]

```

Рисунок 13 - Dockerfile

Соберите образ Docker с помощью команды docker build.

```

alena@LAPTOP-ARKAM4ED:~/AD-5/my-python-app$ docker build -t my-python-app .
[+] Building 16.4s (12/12) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 622B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/python:3.8-slim

```

Рисунок 14 – Сборка образа

Запустите контейнер из образа Docker с помощью команды docker run.

```

alena@LAPTOP-ARKAM4ED:~/AD-5/my-python-app$ docker run --name app -d my-python-app
46fcd079888a431174bc283662e6fa46215f7f6c7dd3c8e54411871d9be115b8

```

Рисунок 15 – 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.

```
# Используйте базовый образ (в данном случае, python:3.8)
FROM python:3.8

# Создайте директорию для приложения
RUN mkdir /app

# Скопируйте файлы приложения в директорию /app внутри контейнера
COPY . /app

# Установите зависимости
RUN pip install --no-cache-dir -r /app/requirements.txt

# Выполните дополнительные команды, например, создание конфигурационных файлов
RUN echo "Config setting=value" > config.txt

# Укажите команду, которая будет выполнена при запуске контейнера
CMD ["python", "/app/app.py"]
```

Рисунок 16 – Dockerfile

Соберите образ Docker с помощью команды `docker build`.

```
alena@LAPTOP-ARKAM4ED:~/AD-5/my-python-app$ docker build -t my-python-app .
[+] Building 15.4s (10/10) FINISHED
=> [internal] load build definition from Dockerfile                                docker:default 0.0s
=> => transferring dockerfile: 814B                                              0.0s
=> [internal] load .dockerignore                                                 0.1s
=> => transferring context: 2B                                                  0.0s
```

Рисунок 17 – Сборка файла

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

```
alena@LAPTOP-ARKAM4ED:~/AD-5/my-python-app$ docker run --name python-2 -d my-python-app
7aba339b05ebf3d9baf5b7a7e8eb60cc2fc8b1c81e9e13fb490b2d261dc42502
```

Рисунок 18 – Запуск контейнера

**Вывод:** мы овладели навыками создания и управления контейнерами Docker для разработки, доставки и запуска приложений.

## Контрольные вопросы:

### 1. Что такое Dockerfile?

Dockerfile — это текстовый файл, который содержит инструкции для создания образа Docker. Образ Docker представляет собой автономный и исполняемый пакет, который включает в себя все необходимое для запуска приложения, включая код, среду выполнения, библиотеки, зависимости и настройки.

### 2. Какие основные команды используются в Dockerfile?



## Конструкции Dockerfile:

1. FROM: указывает базовый образ.
2. COPY и ADD: копируют файлы в образ.
3. RUN: выполняет команды внутри образа.
4. CMD: задает команду по умолчанию для контейнера.
5. ENTRYPOINT: определяет исполняемую команду при запуске контейнера.

6. EXPOSE: объявляет порт, который контейнер будет слушать.

7. ENV: устанавливает переменные среды.

8. ARG: определяет аргументы для сборки образа.

### **3. Для чего используется команда FROM?**

Команда FROM используется в Dockerfile для указания базового образа, на основе которого будет создаваться новый образ.

### **4. Для чего используется команда WORKDIR?**

Команда WORKDIR используется в Dockerfile для установки рабочего каталога внутри контейнера.

### **5. Для чего используется команда COPY?**

Команда COPY в Dockerfile используется для копирования файлов и директорий из вашей хост-системы в образ контейнера.

### **6. Для чего используется команда RUN?**

Команда RUN в Dockerfile используется для выполнения команд во время сборки образа контейнера.

### **7. Для чего используется команда CMD?**

Команда CMD в Dockerfile используется для указания команды, которая будет выполнена при запуске контейнера на основе этого образа.

### **8. Для чего используется команда EXPOSE?**

Команда EXPOSE используется для указания портов, которые контейнер будет слушать во время выполнения.

### **9. Для чего используется команда ENV?**

Команда ENV используется для определения переменных среды, которые будут доступны во время выполнения контейнера.

#### **10. Для чего используется команда USER?**

Команда USER используется для указания пользователя, от имени которого будет выполняться контейнер.

#### **11. Для чего используется команда HEALTHCHECK?**

Команда HEALTHCHECK используется для добавления проверки работоспособности контейнера.

#### **12. Для чего используется команда LABEL?**

Команда LABEL используется для добавления меток к образу Docker.

#### **13. Для чего используется команда ARG?**

Команда ARG используется для передачи аргументов при сборке образа Docker.

#### **14. Для чего используется команда ONBUILD?**

Команда ONBUILD используется в Dockerfile для определения команд, которые будут выполнены при использовании вашего образа в качестве базового образа для другого образа.

#### **15. Что такое многоэтапная сборка?**

Многоэтапная сборка в Docker позволяет создавать образы, используя несколько этапов, каждый из которых может выполнять определенные задачи.

#### **16. Какие преимущества использования многоэтапной сборки?**

Преимущества многоэтапной сборки включают уменьшение размера образа, улучшение безопасности и упрощение процесса сборки.

#### **17. Какие недостатки использования многоэтапной сборки?**

Недостатки многоэтапной сборки могут включать сложность настройки и поддержки для более сложных сценариев сборки.

#### **18. Как определить базовый образ в Dockerfile?**

Базовый образ определяется с помощью команды FROM в Dockerfile.

#### **19. Как определить рабочую директорию в Dockerfile?**

Рабочая директория определяется с помощью команды WORKDIR в Dockerfile.

## **20. Как скопировать файлы в образ Docker?**

Файлы копируются в образ Docker с помощью команды COPY или ADD в Dockerfile.

## **21. Как выполнить команды при сборке образа Docker?**

Команды выполняются при сборке образа Docker с помощью команды RUN в Dockerfile.

## **22. Как указать команду запуска контейнера?**

Команда запуска контейнера указывается с помощью команды CMD или ENTRYPOINT в Dockerfile.

## **23. Как открыть порты в контейнере?**

Порты открываются в контейнере с помощью команды EXPOSE в Dockerfile и опций при запуске контейнера.

## **24. Как задать переменные среды в образе Docker?**

Переменные среды задаются в образе Docker с помощью команды ENV в Dockerfile.

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

Пользователь изменяется с помощью команды USER в Dockerfile.

## **26. Как добавить проверку работоспособности к контейнеру?**

Проверка работоспособности добавляется к контейнеру с помощью команды HEALTHCHECK в Dockerfile.

## **27. Как добавить метку к контейнеру?**

Метка добавляется к контейнеру с помощью команды LABEL в Dockerfile.

## **28. Как передать аргументы при сборке образа Docker?**

Аргументы передаются при сборке образа Docker с помощью команды ARG в Dockerfile.

## **29. Как выполнить команду при первом запуске контейнера?**

Команда выполняется при первом запуске контейнера с помощью команды ONBUILD в Dockerfile.

### **30. Как определить зависимости между образами Docker?**

Зависимости между образами Docker определяются через инструкции в Dockerfile, такие как FROM для базового образа и COPY для копирования файлов из других образов.