МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Институт цифрового развития

ОТЧЁТ

по лабораторной работе №3

Дисциплина: «Конфигурационное распределенное управление ПО»

Тема: «Основы работы с Dockerfile»

Вариант 5

Выполнила: студентка 3 курса,

группы ИВТ-б-о-21-1

Диченко Дина Алексеевна

Ставрополь 2023

Цель: 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.

Практическая часть:

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

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

```
root@DESKTOP-1B0Q56B:~# mkdir my-web-app
root@DESKTOP-1B0Q56B:~# cd my-web-app
root@DESKTOP-1B0Q56B:~/my-web-app# python3 -m venv .venv
root@DESKTOP-1B0Q56B:~/my-web-app# source venv/bin/activate
-bash: venv/bin/activate: No such file or directory
root@DESKTOP-1B0Q56B:~/my-web-app# source .venv/bin/activate
(.venv) root@DESKTOP-1B0Q56B:~/my-web-app# pip install flask
Collecting flask
  Using cached flask-3.0.0-py3-none-any.whl (99 kB)
Collecting click>=8.1.3
  Using cached click-8.1.7-py3-none-any.whl (97 kB)
Collecting itsdangerous>=2.1.2
  Using cached itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting Werkzeug>=3.0.0
  Using cached werkzeug-3.0.1-py3-none-any.whl (226 kB)
Collecting blinker>=1.6.2
  Using cached blinker-1.7.0-py3-none-any.whl (13 kB)
Collecting Jinja2>=3.1.2
  Using cached Jinja2-3.1.2-py3-none-any.whl (133 kB)
Collecting MarkupSafe>=2.0
  Using cached 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 Jinja2-3.1.2 MarkupSafe-2.1.3 Werkzeug-3.0.1 blinker-1.7.0 click-8.1.7 flask-3.0.0 its
dangerous-2.1.2
(.venv) root@DESKTOP-1B0Q56B:~/my-web-app# pip freeze > .\requirements.txt
```

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

from datetime import datetime
from pathlib import Path

from flask import Flask, render_template


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


app.route("/user/<username>")
ef hello_world(username):
        return render_template("index.html", utc_dt=datetime.utcnow(), username=username)


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

```html
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title>Task 1</title>
</head>
<body>
        <h1>Hello, {{username}}</h1>
        <h2>Welcome to my first flask app!</h2>
        <h3>{{ utc_dt }}</h3>
</body>
</html>
```

Создайте Dockerfile для сборки образа Docker вашего приложения.

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

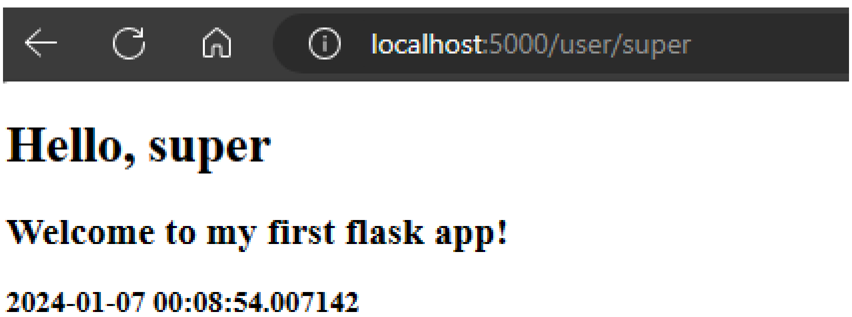
```dockerfile
FROM python:3.10-slim

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" ]
```

Соберите образ Docker с помощью команды docker build .\

```
(.venv) root@DESKTOP-1B0Q56B:~/my-web-app# docker build -t my-web-app .
[+] Building 12.8s (12/12) FINISHED                                    docker:default
 => [internal] load .dockerignore                                                0.0s
 => => transferring context: 2B                                                  0.0s
 => [internal] load build definition from Dockerfile                             0.0s
 => => transferring dockerfile: 262B                                             0.0s
 => [internal] load metadata for docker.io/library/python:3.10-slim              4.3s
 => [auth] library/python:pull token for registry-1.docker.io                    0.0s
 => [1/6] FROM docker.io/library/python:3.10-slim@sha256:25f03d17398b3f001e040fc951b4ee9404862f1b65c5eea1aa31c042  0.0s
```

Запустите контейнер из образа Docker с помощью команды docker run .

```
(.venv) root@DESKTOP-1B0Q56B:~/my-web-app# docker run -p 5000:5000 --name my-python-app -d my-web-app
c23ae3927f63cba6455593b9fd51fa4b23ee58f9b6d60127c72b747d1565e177
```



Задача 2: 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.

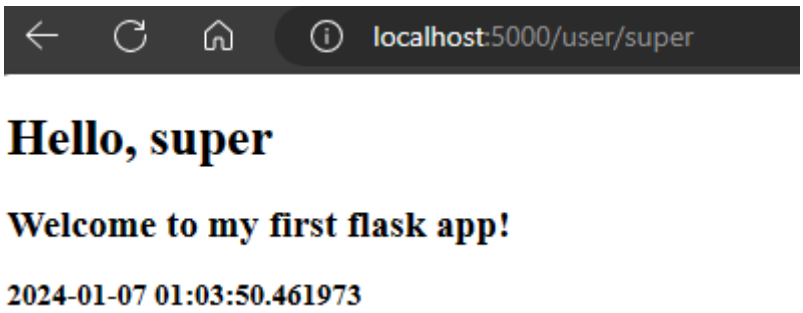
```
FROM python:3.10-slim AS builder

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
RUN pip install numpy

FROM python:3.10-slim AS runner

WORKDIR /usr/src/app

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

EXPOSE 5000

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

Соберите образ Docker с помощью команды docker build .

```
+] Building 65.3s (15/15) FINISHED                                                                                    docker:default
=> [internal] load build definition from Dockerfile                                                                             0.0s
=> => transferring dockerfile: 386B                                                                                             0.0s
=> [internal] load .dockerignore                                                                                                0.0s
=> => transferring context: 2B                                                                                                  0.0s
=> [internal] load metadata for docker.io/library/python:3.10-slim                                                              3.7s
=> [auth] library/python:pull token for registry-1.docker.io                                                                    0.0s
=> [internal] load build context                                                                                                0.0s
=> => transferring context: 133B                                                                                                0.0s
=> CACHED [builder 1/7] FROM docker.io/library/python:3.10-slim@sha256:25f03d17398b3f001e040fc951b4ee9404862f1b6                0.0s
=> [runner 2/3] WORKDIR /usr/src/app                                                                                            0.1s
=> CACHED [builder 2/7] RUN mkdir /usr/src/app                                                                                  0.0s
=> CACHED [builder 3/7] COPY ./my-app /usr/src/app                                                                              0.0s
=> CACHED [builder 4/7] COPY requirements.txt /usr/src/app                                                                      0.0s
=> CACHED [builder 5/7] WORKDIR /usr/src/app                                                                                    0.0s
```

Запустите контейнер из образа Docker с помощью команды docker run .

```
(.venv) root@DESKTOP-1B0Q56B:~/my-web-app# docker run -p 5000:5000 --name task2 -d my-web-app
fad4be9001b42df6c4c7fb2bd406f7a436cf7200a89c7040df4922ce3ba2cd7a
(.venv) root@DESKTOP-1B0Q56B:~/my-web-app#
```



Задача 3: Настройка переменных среды в образе Docker

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

```
FROM python:3.10-slim AS builder

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

WORKDIR /usr/src/app

ENV URL=postgres://user:password@localhost:5432/database

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

EXPOSE 5000

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

Запустите контейнер из образа Docker с помощью команды docker run .

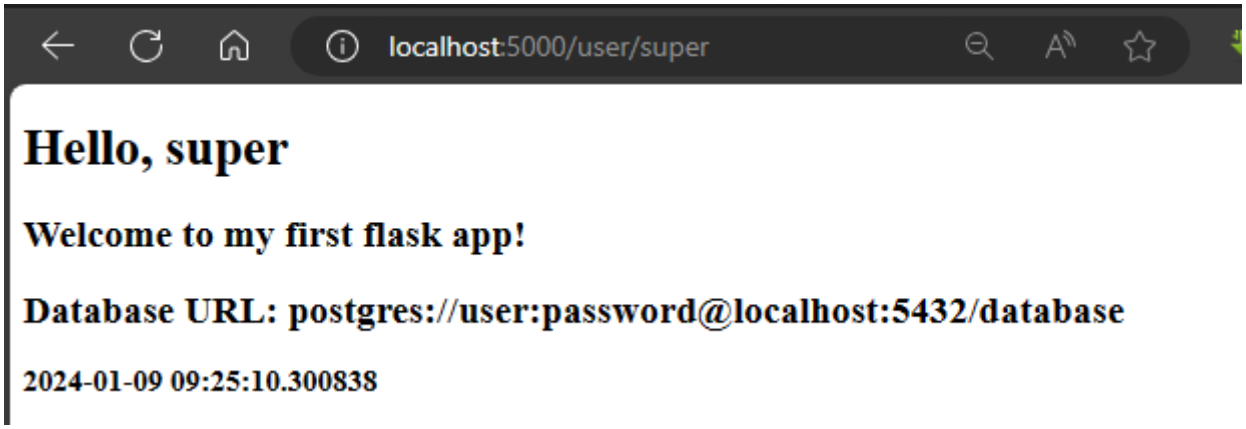
```
(.venv) root@DESKTOP-1B0Q56B:~/my-web-app# docker build -t my-web-app .
[+] Building 46.5s (12/12) FINISHED                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                           0.0s
 => => transferring dockerfile: 351B                                                                           0.0s
 => [internal] load .dockerignore                                                                              0.0s
 => => transferring context: 2B                                                                                0.0s
 => [internal] load metadata for docker.io/library/python:3.10-slim                                            1.6s
 => [1/7] FROM docker.io/library/python:3.10-slim@sha256:25f03d17398b3f001e040fc951b4ee9404862f1b65c5eea1aa31c042  0.0s
 => [internal] load build context                                                                              0.0s
 => => transferring context: 133B                                                                              0.0s
 => CACHED [2/7] RUN mkdir /usr/src/app                                                                        0.0s
 => CACHED [3/7] COPY ./my-app /usr/src/app                                                                    0.0s
 => CACHED [4/7] COPY requirements.txt /usr/src/app                                                            0.0s
```

```
(.venv) root@DESKTOP-1B0Q56B:~/my-web-app# docker run -p 5000:5000 --name task3 -d my-web-app
92692a3139c1c122d9bbc2e4c298873e34a23edbb12a9ecefeb9a08de99d3690
```

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

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

from datetime import datetime
from pathlib import Path

from flask import Flask, render_template
import os

app = Flask(__name__, template_folder=str(Path(__file__).parent))
url = os.environ.get('URL')


@app.route("/user/<username>")
def hello_world(username):
        return render_template("index.html", utc_dt=datetime.utcnow(), username=username, database_url=url)

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

```html
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title>Task 1</title>
</head>
<body>
        <h1>Hello, {{username}}</h1>
        <h2>Welcome to my first flask app!</h2>
        <h2>Database URL: {{ database_url }}<h2>
        <h3>{{ utc_dt }}</h3>
</body>
</html>
```



Задача 4: Копирование файлов в образ Docker

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

```
FROM python:3.10-slim AS builder

WORKDIR /usr/src/app

ENV URL=postgres://user:password@localhost:5432/database

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

RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

FROM python:3.10-slim as runner

WORKDIR /usr/src/app

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

EXPOSE 5000

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

Укажите исходное расположение файлов и их местоположение в образе.

Соберите образ Docker с помощью команды docker build .

```
(.venv) root@DESKTOP-1B0Q56B:~/my-web-app# docker build -t my-web-app .
[+] Building 12.1s (10/10) FINISHED                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                           0.0s
 => => transferring dockerfile: 298B                                                                           0.0s
 => [internal] load .dockerignore                                                                              0.0s
 => => transferring context: 2B                                                                                0.0s
 => [internal] load metadata for docker.io/library/python:3.10-slim                                           11.9s
 => [auth] library/python:pull token for registry-1.docker.io                                                  0.0s
 => [internal] load build context                                                                              0.0s
 => => transferring context: 98B                                                                               0.0s
 => [builder 1/3] FROM docker.io/library/python:3.10-slim@sha256:25f03d17398b3f001e040fc951b4ee9404862f1b65c5e  0.0s
 => CACHED [builder 2/3] WORKDIR /usr/src/app                                                                  0.0s
 => CACHED [builder 3/3] COPY my-app ./                                                                        0.0s
```

Запустите контейнер из образа Docker с помощью команды docker run.

```
.venv) root@DESKTOP-1B0Q56B:~/my-web-app# docker run -p 5000:5000 --name task4 -d my-web-app
737691d7bc227a466bf83dbdca8c28ac949f22d2bc9f49ed46e92d689983ea9
```

Задача 5: Запуск команд при запуске контейнера

Определите команды для выполнения при запуске контейнера с помощью команды RUN в Dockerfile.

```
FROM python:3.10-slim AS builder

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
RUN pip install numpy

ENV URL=postgres://user:password@localhost:5432/database

EXPOSE 5000

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

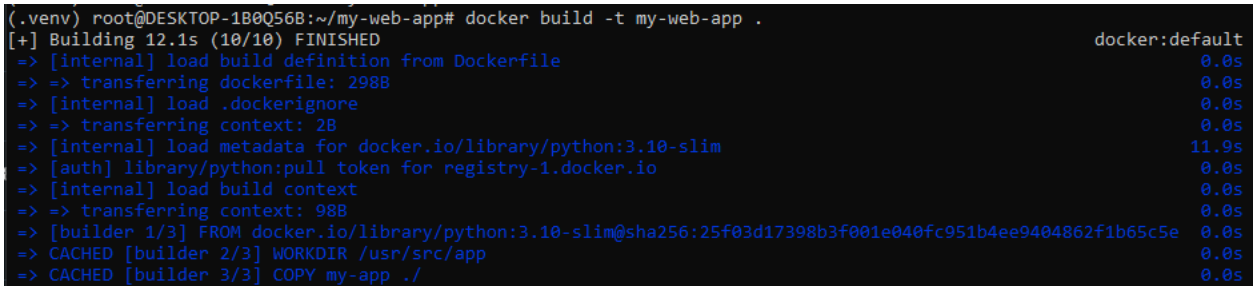
Укажите команды и их параметры, например, создание конфигурационных файлов или выполнение скриптов инициализации.

RUN mkdir /usr/src/app – создание директории

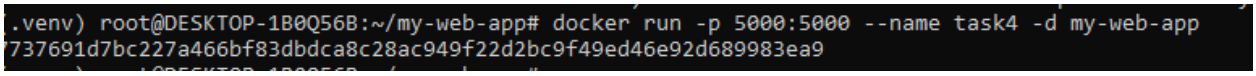
RUN pip install .. – установка библиотек python

CMD ["python", "app.py"] – запуск исполняемого файла

Соберите образ Docker с помощью команды docker build .



Запустите контейнер из образа Docker с помощью команды docker run.



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, run, cmd, copy, expose

3 Для чего используется команда FROM?

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

4 Для чего используется команда WORKDIR?

устанавливает каталог, который будет рабочим каталогом для приложения.

5 Для чего используется команда COPY?

Копирует файлы из хоста в образ.

6 Для чего используется команда RUN?

Выполняет команды в процессе создания образа.

7 Для чего используется команда CMD?

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

8 Для чего используется команда EXPOSE?

Указывает порты, которые будут открыты в контейнере.

9 Для чего используется команда ENV?

Настроить переменную среды в образе Docker.

10 Для чего используется команда USER?

указывает пользователя, от имени которого будет выполняться основная команда контейнера.

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

Вывод: в результате выполнения работы были освоены навыки создания и управления контейнерами Docker для разработки, доставки и запуска приложений.