

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

(подпись)

Доцент кафедры инфокоммуникаций
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.

Порядок выполнения работы

1. 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.

```
C:\Users\vлади\OneDrive\Рабочий стол\Основы программной инженерии\Docker>mkdir python-web-app
C:\Users\vлади\OneDrive\Рабочий стол\Основы программной инженерии\Docker>cd python-web-app
C:\Users\vлади\OneDrive\Рабочий стол\Основы программной инженерии\Docker\python-web-app>|
```

Рисунок 1.1 – Создание папки приложения

```
C:\Users\vлади\OneDrive\Рабочий стол\Основы программной инженерии\Docker\python-web-app>python -m venv .venv
```

Рисунок 1.2 – Создание виртуального окружения Python

```
C:\Users\vлади\OneDrive\Рабочий стол\Основы программной инженерии\Docker\python-web-app>.\venv\Scripts\activate.bat
(.venv) C:\Users\vлади\OneDrive\Рабочий стол\Основы программной инженерии\Docker\python-web-app>|
```

Рисунок 1.3 – Активация виртуального окружения

```
(.venv) C:\Users\vлади\OneDrive\Рабочий стол\Основы программной инженерии\Docker\python-web-app>pip install flask
Collecting flask
  Obtaining dependency information for flask from https://files.pythonhosted.org/packages/36/42/015c23096649b908c809c69388a805a571a3bea44362fe87e33fc3afa01f/flask-3.0.0-py3-none-any.whl.metadata
  Using cached flask-3.0.0-py3-none-any.whl.metadata (3.6 kB)
Collecting Werkzeug>=3.0.0 (from flask)
  Obtaining dependency information for Werkzeug>=3.0.0 from https://files.pythonhosted.org/packages/c3/fc/254c3e9b5feb89ff5b9076a23218dafbc99c96ac5941e900b71206e6313b/werkzeug-3.0.1-py3-none-any.whl.metadata
  Using cached werkzeug-3.0.1-py3-none-any.whl.metadata (4.1 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Using cached Jinja2-3.1.2-py3-none-any.whl (133 kB)
Collecting itsdangerous>=2.1.2 (from flask)
  Using cached itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting click>=8.1.3 (from flask)
  Obtaining dependency information for click>=8.1.3 from https://files.pythonhosted.org/packages/00/2e/d53fa4befbf2cfa713304affc7ca780ce4fc1fd8710527771b58311a3229/click-8.1.7-py3-none-any.whl.metadata
  Using cached click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting blinker>=1.6.2 (from flask)
  Obtaining dependency information for blinker>=1.6.2 from https://files.pythonhosted.org/packages/fa/2a/7f3714cbc6356aefec525ce7a0613d581072ed6eb53eb7b9754f33db807/blinker-1.7.0-py3-none-any.whl.metadata
  Using cached blinker-1.7.0-py3-none-any.whl.metadata (1.9 kB)
Collecting colorama (from click>=8.1.3->flask)
  Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->flask)
```

Рисунок 1.4 – Установка Flask

```
(.venv) C:\Users\vлади\OneDrive\Рабочий стол\Основы программной инженерии\Docker\python-web-app>pip freeze > .\requirements.txt
```

Рисунок 1.5 – Создание файла, содержащий список необходимых пакетов

```
app.py ×
my-app > app.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  from datetime import datetime
4  from pathlib import Path
5  from flask import Flask, render_template
6
7  app = Flask(__name__, template_folder=str(Path(__file__).parent))
8
9
10 @app.route("/")
11 def hello_world():
12     return render_template("index.html", utc_dt=datetime.utcnow())
13
14
15 if __name__ == "__main__":
16     app.run(host="0.0.0.0")
17
```

Рисунок 1.6 – Создание файла app.py

```
<> index.html ×
my-app > <> index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>FlaskApp</title>
6  </head>
7  <body>
8      <h1>Hello World!</h1>
9      <h2>Welcome to FlaskApp!</h2>
10     <h3>{{ utc_dt }}</h3>
11 </body>
12 </html>
```

Рисунок 1.7 – Создание файла index.html

```
Dockerfile X
Dockerfile
1 FROM python:3.10-slim
2
3 RUN mkdir /usr/src/app
4 COPY ./my-app /usr/src/app
5 COPY ./requirements.txt /usr/src/app
6
7 WORKDIR /usr/src/app
8
9 RUN pip install --no-cache-dir -r requirements.txt
10
11 EXPOSE 5000
12
13 CMD ["python", "app.py"]
```

Рисунок 1.8 – Создание Dockerfile

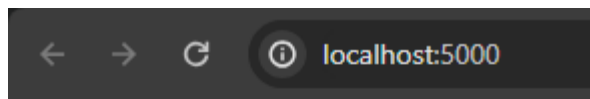
```
(.venv) C:\Users\vladi\OneDrive\Рабочий стол\Основы программной инженерии\docker\python-web-app>docker build -t python-web-app .
[+] Building 11.7s (12/12) FINISHED
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 275B
=> [internal] load metadata for docker.io/library/python:3.10-slim
=> [auth] library/python:pull token for registry-1.docker.io
=> [1/6] FROM docker.io/library/python:3.10-slim@sha256:25f03d17398b3f001e040fc951b4ee9404862f1b65c5eea1aa31c042dfdab527
=> resolve docker.io/library/python:3.10-slim@sha256:25f03d17398b3f001e040fc951b4ee9404862f1b65c5eea1aa31c042dfdab527
=> sha256:64119eae2e5157ca18d0ec9b9c7865e454b09bca702c3e2f579703deb0eaaeaf 12.38MB / 12.38MB
=> sha256:a28cd3d033268c30cb986f4a0d36c4aeda4c5aeaa5c764562f323c73d1f4f771 244B / 244B
=> sha256:25f03d17398b3f001e040fc951b4ee9404862f1b65c5eea1aa31c042dfdab527 1.65kB / 1.65kB
=> sha256:9956522e7eafd57e3e7bb4b102d56f02882924019867cd2036c1a7c3ee56b174 1.37kB / 1.37kB
=> sha256:3e27f11955d637ad643f1ebf37168854043fbafba20176af0339c01d51f1c1c 6.94kB / 6.94kB
=> sha256:8ce3f2b601ccac03ff1858022363c325355bafba224123a4563dade58bc8e70f 3.51MB / 3.51MB
=> sha256:2c1fe8bcf114038da9146a2528b2c3b606a09873aee7122f8529d45418fcd9e9 3.36MB / 3.36MB
=> extracting sha256:8ce3f2b601ccac03ff1858022363c325355bafba224123a4563dade58bc8e70f
=> extracting sha256:64119eae2e5157ca18d0ec9b9c7865e454b09bca702c3e2f579703deb0eaaeaf
=> extracting sha256:a28cd3d033268c30cb986f4a0d36c4aeda4c5aeaa5c764562f323c73d1f4f771
=> extracting sha256:2c1fe8bcf114038da9146a2528b2c3b606a09873aee7122f8529d45418fcd9e9
=> [internal] load build context
=> => transferring context: 899B
=> [2/6] RUN mkdir /usr/src/app
=> [3/6] COPY ./my-app /usr/src/app
=> [4/6] COPY ./requirements.txt /usr/src/app
=> [5/6] WORKDIR /usr/src/app
=> [6/6] RUN pip install --no-cache-dir -r requirements.txt
=> exporting to image
=> => exporting layers
=> => writing image sha256:71e6c9fb8cf0b0532876d0a298062df1513e3e7fb3c4709dcb6ec7300980dd04
=> => naming to docker.io/library/python-web-app

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
```

Рисунок 1.9 – Создание образа

```
(.venv) C:\Users\vladi\OneDrive\Рабочий стол\Основы программной инженерии\docker>docker run -p 5000:5000 --name my-python-app -d python-web-app
59c28527a86043b6cfc9b2a80243f4d63ea785ecde577b4685590325138906
```

Рисунок 1.10 – Запуск контейнера на основе созданного образа



Hello World!

Welcome to FlaskApp!

2023-12-29 15:29:49.141584

Рисунок 1.11 – Работа приложения

2. Установить дополнительный пакет, например библиотеку NumPy для Python, в образ Docker веб-приложения. 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. Соберите образ Docker с помощью команды docker build. Запустите контейнер из образа Docker с помощью команды docker run.

```
1 FROM python:3.10 as builder
2
3 RUN python -m venv /opt/venv
4 ENV PATH="/opt/venv/bin:$PATH"
5
6 COPY requirements.txt .
7
8 RUN pip install --no-cache-dir -r requirements.txt
9 RUN pip install --no-cache-dir numpy
10
11 RUN mkdir /usr/src/app
12 COPY ./my-app /usr/src/app
13
14 WORKDIR /usr/src/app
15
16 FROM python:3.10-slim as runner
17
18 WORKDIR /usr/src/app
19
20 COPY --from=builder /opt/venv /opt/venv
21 COPY --from=builder /usr/src/app/. .
22
23 EXPOSE 5000
24
25 ENV PATH="/opt/venv/bin:$PATH"
26
27 CMD [ "python", "app.py" ]
28
```

Рисунок 2.1- Dockerfile

```
(.venv) C:\Users\vladi\OneDrive\Рабочий стол\Основы программной инженерии\docker\python-web-app>docker build -t multi-stage-web-app .
[*] Building 2.3s (17/17) FINISHED
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 401B
=> [internal] load metadata for docker.io/library/python:3.10-slim
=> [internal] load metadata for docker.io/library/python:3.10
=> [auth] library/python:pull token for registry-1.docker.io
=> [builder 1/7] FROM docker.io/library/python:3.10@sha256:ba7e6f1feea85621dec8a6525e1bb9edf38a3897bc6fd847d72f860ff0b73786
=> [internal] load build context
=> => transferring context: 134B
=> [runner 1/3] FROM docker.io/library/python:3.10-slim@sha256:25f03d17398b3f001e040fc951b4ee9404862f1b65c5eea1aa31c042dfdaab527
=> CACHED [runner 2/3] WORKDIR /usr/src/app
=> CACHED [builder 2/7] RUN mkdir /usr/src/app
=> CACHED [builder 3/7] COPY ./requirements.txt /usr/src/app
=> CACHED [builder 4/7] COPY ./my-app /usr/src/app
=> CACHED [builder 5/7] WORKDIR /usr/src/app
=> CACHED [builder 6/7] RUN pip install --no-cache-dir -r requirements.txt
=> CACHED [builder 7/7] RUN pip install numpy
=> CACHED [runner 3/3] COPY --from=builder /usr/src/app .
=> exporting to image
=> => exporting layers
=> => writing image sha256:907b338dd05fecb3444b734cd9673621f5c8c7b41146e8ad0213594c20e1e9df
=> => naming to docker.io/library/multi-stage-web-app
```

Рисунок 2.2 – Создание образа

```
(.venv) C:\Users\vladi\OneDrive\Рабочий стол\Основы программной инженерии\docker\python-web-app>docker run -p 5000:5000 --name my-python-app-2 -d multi-stage-web-app
e159152eed41a9b1640de33d93c6089acbef15a4a72f1f77ca4999a7ae958c5b
```

Рисунок 2.3 – 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.

```
1 FROM python:3.10-slim
2 Set the baseImage to use for subsequent instructions. FROM must be the first
3 RUN mkdir /usr/src/app
4
5 COPY ./my-app /usr/src/app
6 COPY ./requirements.txt /usr/src/app
7
8 WORKDIR /usr/src/app
9
10 RUN pip install --no-cache-dir -r requirements.txt
11
12 ENV DATABASE_URL postgres://user:password@localhost:5432/database
13
14 EXPOSE 5000
15
16 CMD [ "python", "app.py" ]
17
```

Рисунок 3.1 – Dockerfile

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  import os
4
5
6  def main():
7      database_url = os.environ["DATABASE_URL"]
8      print(database_url)
9
10
11 if __name__ == "__main__":
12     main()
13

```

Рисунок 3.2 – Программа

```

(.venv) C:\Users\vladi\OneDrive\Рабочий стол\Основы программной инженерии\docker\python-web-app>docker build -t app-env .
[+] Building 7.2s (12/12) FINISHED
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 348B
=> [internal] load metadata for docker.io/library/python:3.10-slim
=> [auth] library/python:pull token for registry-1.docker.io
=> [1/6] FROM docker.io/library/python:3.10-slim@sha256:25f03d17398b3f00e040fc951b4ee9404862f1b65c5eealaa31c042dfdab527
=> [internal] load build context
=> => transferring context: 338B
=> CACHED [2/6] RUN mkdir /usr/src/app
=> [3/6] COPY ./my-app /usr/src/app
=> [4/6] COPY ./requirements.txt /usr/src/app
=> [5/6] WORKDIR /usr/src/app
=> [6/6] RUN pip install --no-cache-dir -r requirements.txt
=> exporting to image
=> => exporting layers
=> => writing image sha256:9389c119aa38c663c71e8b2204f768569ec23fcbdb1f58e51549ba9856e238b1
=> => naming to docker.io/library/app-env

View build details: docker-desktop://dashboard/build/default/default/urcr2k0tdqg8ozjzifqxqz2ys

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview

```

Рисунок 3.3 – Создание образа

```

(.venv) C:\Users\vladi\OneDrive\Рабочий стол\Основы программной инженерии\docker\python-web-app>docker run -d -e DATABASE_URL=postgres://user:password@localhost:5432/database app-env
3e0555069f6112173df2c0963fd37f2461a8028b3d7ad8607db26763eef9e5ce

```

Рисунок 3.4 – Запуск контейнера

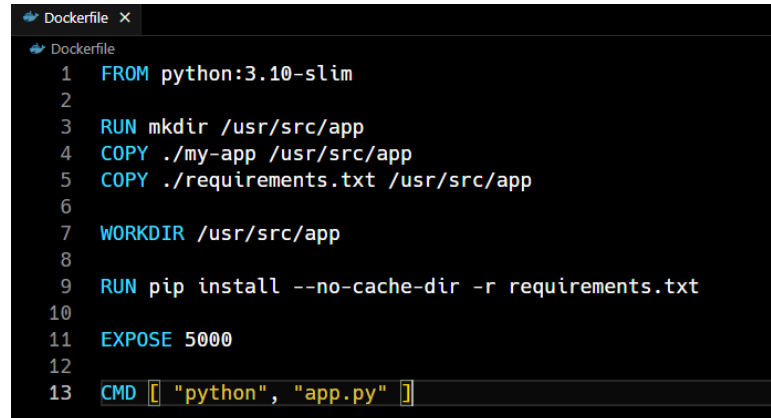
```

2023-12-29 21:15:20 postgres://user:password@localhost:5432/database

```

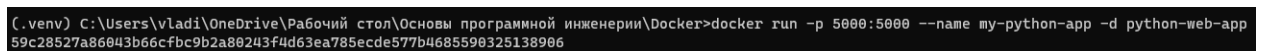
Рисунок 3.5 – Вывод

4. 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.



```
1 FROM python:3.10-slim
2
3 RUN mkdir /usr/src/app
4 COPY ./my-app /usr/src/app
5 COPY ./requirements.txt /usr/src/app
6
7 WORKDIR /usr/src/app
8
9 RUN pip install --no-cache-dir -r requirements.txt
10
11 EXPOSE 5000
12
13 CMD ["python", "app.py"]
```

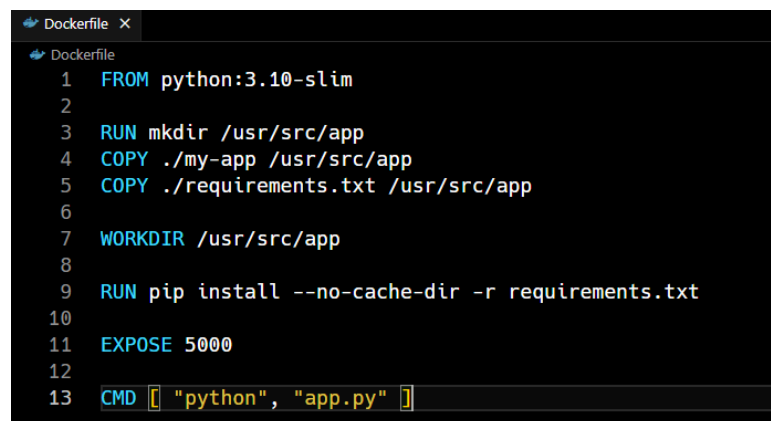
Рисунок 4.3 - Dockerfile



```
(.venv) C:\Users\vladi\OneDrive\Рабочий стол\Основы программной инженерии\Docker>docker run -p 5000:5000 --name my-python-app -d python-web-app
59c28527a86043b66cfbc9b2a80243f4d63ea785ecde577b4685590325138906
```

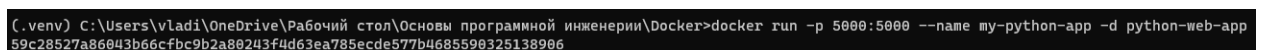
Рисунок 4.2 – Запуск контейнера

5. Выполнить команды инициализации или настройки при запуске контейнера вебприложения. Используйте команду RUN в Dockerfile для определения команд для выполнения и их параметров.



```
1 FROM python:3.10-slim
2
3 RUN mkdir /usr/src/app
4 COPY ./my-app /usr/src/app
5 COPY ./requirements.txt /usr/src/app
6
7 WORKDIR /usr/src/app
8
9 RUN pip install --no-cache-dir -r requirements.txt
10
11 EXPOSE 5000
12
13 CMD ["python", "app.py"]
```

Рисунок 5.1 - Dockerfile



```
(.venv) C:\Users\vladi\OneDrive\Рабочий стол\Основы программной инженерии\Docker>docker run -p 5000:5000 --name my-python-app -d python-web-app
59c28527a86043b66cfbc9b2a80243f4d63ea785ecde577b4685590325138906
```

Рисунок 5.2 – Запуск контейнера

Ответы на контрольные вопросы

1. Dockerfile - это конфигурационный файл, который используется для создания образов Docker. Он содержит инструкции, которые определяют, как собрать образ Docker.

2. Основные команды, используемые в Dockerfile, включают в себя:

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

- WORKDIR: устанавливает рабочую директорию для последующих команд в Dockerfile.

- COPY: копирует файлы из хост-системы в контейнер Docker.

- RUN: выполняет команды внутри контейнера Docker.

- CMD: указывает команду, которая будет выполнена при запуске контейнера Docker.

- EXPOSE: указывает порт, который будет использоваться для взаимодействия с контейнером Docker.

- ENV: устанавливает переменные среды в контейнере Docker.

3. Команда FROM используется для указания базового образа, на основе которого будет создан новый образ. Например, FROM python:3.9-slim-buster указывает, что мы хотим использовать образ Python 3.9 в качестве базового образа.

4. Команда WORKDIR устанавливает рабочую директорию для последующих команд в Dockerfile. Например, WORKDIR /app устанавливает /app в качестве рабочей директории.

5. Команда COPY копирует файлы из хост-системы в контейнер Docker. Например, COPY requirements.txt /app копирует файл requirements.txt из хост-системы в директорию /app в контейнере Docker.

6. Команда RUN выполняет команды внутри контейнера Docker. Например, RUN pip install -r requirements.txt устанавливает зависимости Python, перечисленные в файле requirements.txt.

7. Команда `CMD` указывает команду, которая будет выполнена при запуске контейнера `Docker`. Например, `CMD ["python", "app.py"]` указывает, что мы хотим запустить приложение на `Python`.

8. Команда `EXPOSE` указывает порт, который будет использоваться для взаимодействия с контейнером `Docker`. Например, `EXPOSE 5000` указывает, что мы хотим использовать порт `5000`.

9. Команда `ENV` устанавливает переменные среды в контейнере `Docker`. Например, `ENV MY_VAR=my_value` устанавливает переменную среды `MY_VAR` со значением `my_value`.

10. Команда `USER` указывает имя пользователя или `UID`, от которого будут выполняться команды внутри контейнера `Docker`.

11. Команда `HEALTHCHECK` позволяет проверять состояние контейнера `Docker` и определять, работает ли он должным образом.

12. Команда `LABEL` позволяет добавлять метаданные к образу `Docker`.

13. Команда `ARG` позволяет передавать аргументы в `Dockerfile` при сборке образа.

14. Команда `ONBUILD` позволяет задавать инструкции, которые будут выполнены при создании образа, который будет использоваться в качестве базового образа для другого образа.

15. Многоэтапная сборка - это метод, который позволяет создавать образы `Docker` с помощью нескольких этапов. Это позволяет уменьшить размер образа и ускорить процесс сборки.

16. Многоэтапная сборка позволяет создавать образы `Docker`, которые меньше по размеру и быстрее в работе, чем традиционные образы. Это достигается за счет того, что каждый этап сборки создает отдельный слой образа, который может быть использован в последующих этапах. Это позволяет уменьшить размер образа и ускорить процесс сборки.

17. Недостатком многоэтапной сборки является то, что она может быть сложнее в понимании и настройке, чем традиционная сборка. Кроме того,

многоэтапная сборка может потребовать больше времени на настройку и тестирование.

18. Для определения базового образа в Dockerfile используется команда FROM. Например, FROM python:3.9-slim-buster указывает, что мы хотим использовать образ Python 3.9 в качестве базового образа.

19. Для определения рабочей директории в Dockerfile используется команда WORKDIR. Например, WORKDIR /app устанавливает /app в качестве рабочей директории.

20. Для копирования файлов в образ Docker используется команда COPY. Например, COPY requirements.txt /app копирует файл requirements.txt из хост-системы в директорию /app в контейнере Docker.

21. Для выполнения команд при сборке образа Docker используется команда RUN. Например, RUN pip install -r requirements.txt устанавливает зависимости Python, перечисленные в файле requirements.txt.

22. Для указания команды запуска контейнера используется команда CMD. Например, CMD ["python", "app.py"] указывает, что мы хотим запустить приложение на Python.

23. Для открытия портов в контейнере используется команда EXPOSE. Например, EXPOSE 5000 указывает, что мы хотим использовать порт 5000.

24. Для задания переменных среды в образе Docker используется команда ENV. Например, ENV MY_VAR=my_value устанавливает переменную среды MY_VAR со значением my_value.

25. Для изменения пользователя, от имени которого будет выполняться контейнер, используется команда USER. Например, USER myuser устанавливает пользователя myuser в качестве пользователя, от имени которого будет выполняться контейнер.

26. Для добавления проверки работоспособности к контейнеру используется команда HEALTHCHECK. Например, HEALTHCHECK CMD

`curl --fail http://localhost:5000/health || exit 1` проверяет, что приложение доступно по адресу `http://localhost:5000/health`.

27. Для добавления метки к контейнеру используется команда `LABEL`. Например, `LABEL version="1.0"` добавляет метку `version` со значением `1.0` к контейнеру.

28. Для передачи аргументов при сборке образа `Docker` используется команда `ARG`. Например, `ARG MY_ARG=my_value` устанавливает аргумент `MY_ARG` со значением `my_value`.

29. Для выполнения команды при первом запуске контейнера используется команда `ONBUILD`. Например, `ONBUILD RUN python setup.py install` выполняет команду `python setup.py install` при первом запуске контейнера.

30. С помощью команды `FROM`.