

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение
высшего образования**

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра информационных систем и технологий

Отчет по лабораторной работе «основы работы с Docker».

Дисциплина: «Основы программной инженерии»

Выполнил:

Студент группы ПИЖ-б-о-22-1,

направление подготовки: 09.03.04

«Программная инженерия»

ФИО: Джараян Арег Александрович

Проверил:

Воронкин Р. А.

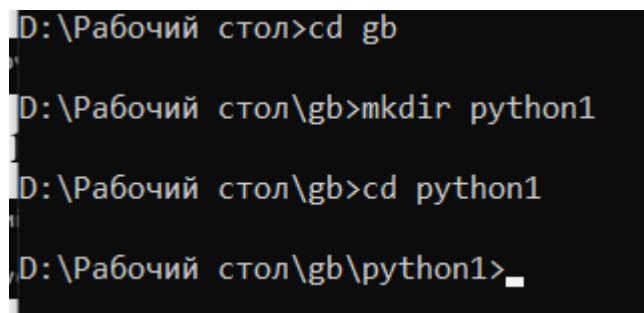
Ставрополь 2023

Тема: Лабораторная работа 3. Основы работы с Dockerfile

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.

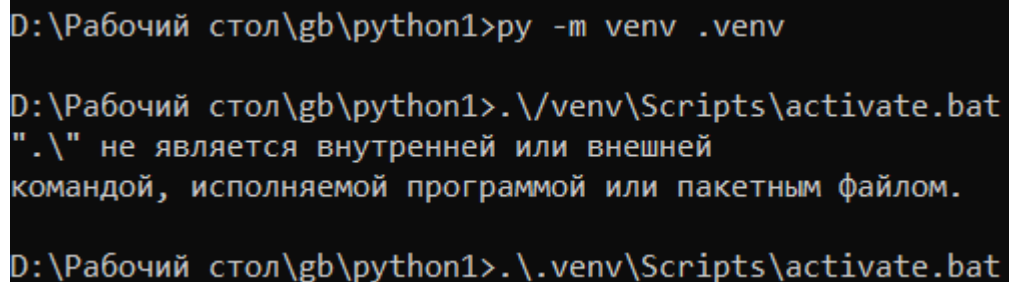
Выполнение работы: Задача 1: Создание простого веб-приложения на Python с использованием Dockerfile.

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.



```
D:\Рабочий стол>cd gb
D:\Рабочий стол\gb>mkdir python1
D:\Рабочий стол\gb>cd python1
D:\Рабочий стол\gb\python1>_
```

Рисунок 1.1 – Создание директории



```
D:\Рабочий стол\gb\python1>py -m venv .venv
D:\Рабочий стол\gb\python1>.\venv\Scripts\activate.bat
"." не является внутренней или внешней
командой, исполняемой программой или пакетным файлом.
D:\Рабочий стол\gb\python1>.\venv\Scripts\activate.bat
```

Рисунок 1.2 – Создание виртуального окружения и активизация виртуального окружения

```
(.venv) D:\Рабочий стол\gb\python1>pip install flask
Collecting flask
  Obtaining dependency information for flask from https://files.pytho
  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://f
  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://file
  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://fi
  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)
  Obtaining dependency information for MarkupSafe>=2.0 from https://f
etadate
  Using cached MarkupSafe-2.1.3-cp312-cp312-win_amd64.whl.metadata (3
Using cached flask-3.0.0-py3-none-any.whl (99 kB)
Using cached blinker-1.7.0-py3-none-any.whl (13 kB)
Using cached click-8.1.7-py3-none-any.whl (97 kB)
Using cached werkzeug-3.0.1-py3-none-any.whl (226 kB)
Using cached MarkupSafe-2.1.3-cp312-cp312-win_amd64.whl (16 kB)
Installing collected packages: MarkupSafe, itsdangerous, colorama, bl
Successfully installed Jinja2-3.1.2 MarkupSafe-2.1.3 Werkzeug-3.0.1 b

[notice] A new release of pip is available: 23.2.1 -> 23.3.2
[notice] To update, run: python.exe -m pip install --upgrade pip

(.venv) D:\Рабочий стол\gb\python1>pip install flask
```

Рисунок 1.3 – Установка Flask

```
(.venv) D:\Рабочий стол\gb\python1>pip freeze > .\requirements.txt
(.venv) D:\Рабочий стол\gb\python1>
```

Рисунок 1.4 – Создание requirements.txt

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

Рисунок 1.5 – Python файл

```
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>
13
```

Рисунок 1.6 – html файл

```
1 FROM python:3.11-slim
2 RUN mkdir /usr/src/app
3 COPY ./my-app /usr/src/app
4 COPY ./requirements.txt /usr/src/app
5 WORKDIR /usr/src/app
6 RUN pip install --no-cache-dir -r requirements.txt
7 EXPOSE 5000
8 CMD [ "python", "app.py" ]
```

Рисунок 1.7 – Dockerfile

```
(.venv) D:\Рабочий стол\gb\python1>docker build -t python1 .
[+] Building 21.6s (12/12) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 265B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/python:3.11-slim
=> [auth] library/python:pull token for registry-1.docker.io
=> [1/6] FROM docker.io/library/python:3.11-slim@sha256:8f64a67710f3d981cf3008d6f9f1dbe61accd7927f165f4e37ea3f8b883ccc3f
=> => resolve docker.io/library/python:3.11-slim@sha256:8f64a67710f3d981cf3008d6f9f1dbe61accd7927f165f4e37ea3f8b883ccc3f
=> => sha256:b05f8bf97bacb24c0f8ddcd5ea6764c49aff2a325c50071f9e2042b6e3d4286c 241B / 241B
=> => sha256:8f64a67710f3d981cf3008d6f9f1dbe61accd7927f165f4e37ea3f8b883ccc3f 1.65kB / 1.65kB
=> => sha256:233e4c53aebbd019c06fd927b93de5292cfc5091ae22bcb9d9630e608107c790 1.37kB / 1.37kB
=> => sha256:dd159e5400f1a750b77529314306da97f529325b74e18d56b0a4e98a4b9f9f0 6.93kB / 6.93kB
=> => sha256:9ce3f2b601ccac03ff1858022363c325355bafba224123a4563dada58bc8e70f 3.51MB / 3.51MB
=> => sha256:6ac7ac839d9ddb5b76a12d1c4e07d66079748ed2dc51a644b95d492902e34d 12.84MB / 12.84MB
=> => sha256:b5a30bc7f1f876ea3d8555c4c65117eee16c4e62241a1a64fd3f14ecb794edd5 3.39MB / 3.39MB
=> => extracting sha256:8ce3f2b601ccac03ff1858022363c325355bafba224123a4563dada58bc8e70f 0.4s
=> => extracting sha256:6ac7ac839d9ddb5b76a12d1c4e07d66079748ed2dc51a644b95d492902e34d 0.9s
=> => extracting sha256:b05f8bf97bacb24c0f8ddcd5ea6764c49aff2a325c50071f9e2042b6e3d4286c 0.5s
=> => extracting sha256:b5a30bc7f1f876ea3d8555c4c65117eee16c4e62241a1a64fd3f14ecb794edd5 0.5s
=> [internal] load build context
=> => transferring context: 909B
=> [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 layers
=> => exporting image sha256:0470472fc9694ec895045b3d9dd51bf5036e63eacafa2ebd162368a049e67090 0.2s
=> => writing image sha256:0470472fc9694ec895045b3d9dd51bf5036e63eacafa2ebd162368a049e67090 0.0s
=> => naming to docker.io/library/python1 0.0s

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

(.venv) D:\Рабочий стол\gb\python1>
```

Рисунок 1.8 – Создание образа

```
(.venv) D:\Рабочий стол\gb\python1>docker run -p 5000:5000 --name python1 -d python1
4ea4306320d8ce649d4bc46a92b5f737ea1df2eddaef4d28024f53a6c0080554
```

Рисунок 1.9 – Создание контейнера

<input type="checkbox"/>	Name	Tag	Status	Created	Size	Actions
<input type="checkbox"/>	python1					
<input type="checkbox"/>	0470472fc969	latest	In use	1 minute ago	146.06 MB	
<input type="checkbox"/>	mysql	latest	In use	11 days ago	619.02 MB	

Рисунок 1.10 – Отображение образа в приложении

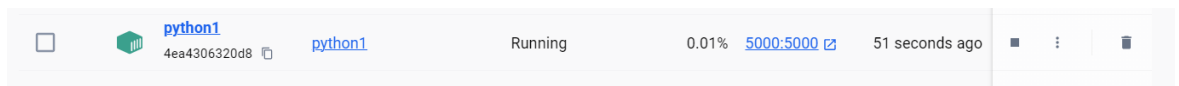


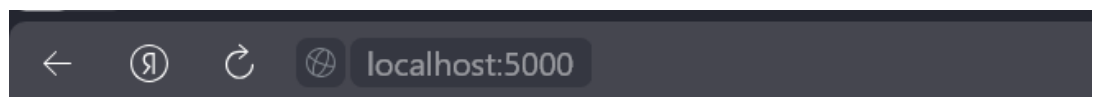
Рисунок 1.11 – Отображение контейнера в приложении

```
(.venv) D:\Рабочий стол\gb\python1>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
4ea4306320d8   python1   "python app.py"         About a minute Up            0.0.0.0:5000->5000/tcp             python1

(.venv) D:\Рабочий стол\gb\python1>docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
python1        latest   0470472fc969   2 minutes ago  146MB
mysql          latest   73246731c4b0   11 days ago    619MB
docker         latest   6091c7bd89fd   2 weeks ago    331MB
ubuntu         latest   174c8c134b2a   2 weeks ago    77.9MB
postgres       latest   398d34d3cc5e   2 weeks ago    425MB
alpine         latest   f8c20f8bbcb6   3 weeks ago    7.38MB
docker/welcome-to-docker latest   c1f619b6477e   7 weeks ago    18.6MB
nginx          latest   d453dd892d93   2 months ago   187MB
openjdk        latest   71260f256d19   10 months ago  470MB
docker/getting-started latest   3e4394f6b72f   12 months ago  47MB
postgres       9.5.20-alpine 14bacaedfd91   3 years ago    37.3MB

(.venv) D:\Рабочий стол\gb\python1>
```

Рисунок 1.12 – Отображение контейнера и образа в терминале



Hello World!

Welcome to FlaskApp!

2023-12-30 06:18:48.441031

Рисунок 1.13 – Работа приложения

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. Соберите

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

```
app.py  <> index.html  Dockerfile

1  FROM python:3.11 as builder
2  RUN mkdir /usr/src/app
3  COPY ./my-app /usr/src/app
4  COPY ./requirements.txt /usr/src/app
5  WORKDIR /usr/src/app
6  RUN pip install --no-cache-dir -r requirements.txt
7  RUN pip install numpy
8  FROM python:3.11-slim as runner
9  WORKDIR /usr/src/app
10 COPY --from=builder /usr/src/app/. .
11 EXPOSE 5000
12 CMD [ "python", "app.py" ]
```

Рисунок 2.1 – Изменение dockerfile

```
(.venv) D:\Рабочий стол\gb\python1\docker build -t python1
ERROR: "docker buildx build" requires exactly 1 argument.
See 'docker buildx build --help'.

Usage: docker buildx build [OPTIONS] PATH | URL | -

Start a build

[+] Building 111.9s (14/16)
=> CACHED [runner 1/3] FROM docker.io/library/python:3.11-slim@sha256:8f64a67710f3d981cf3008d6f9f1dbe61accd7927f165f4e37ea3f8b883ccc3f
[+] Building 113.7s (17/17) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 393B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/python:3.11-slim
=> [internal] load metadata for docker.io/library/python:3.11
=> [auth] library/python:pull token for registry-1.docker.io
=> CACHED [runner 1/3] FROM docker.io/library/python:3.11-slim@sha256:8f64a67710f3d981cf3008d6f9f1dbe61accd7927f165f4e37ea3f8b883ccc3f
=> [builder 1/7] FROM docker.io/library/python:3.11@sha256:4e5e9b05d4e8cf699084f20bb1d3463234446387fa0f7a45d90689c48e204c83
=> resolve docker.io/library/python:3.11@sha256:4e5e9b05d4e8cf699084f20bb1d3463234446387fa0f7a45d90689c48e204c83
=> sha256:4e5e9b05d4e8cf699084f20bb1d3463234446387fa0f7a45d90689c48e204c83 2.14kB / 2.14kB
=> sha256:22140cbb3b0c5bba69de94474622781790e217d05a6f34c6f254645e1e2dc9 7.53kB / 7.53kB
=> sha256:bc0734b949dcabc5bdf0c8b9f44491e0fce04cb10c9c6e76282b9f6abdf01 24.05MB / 24.05MB
=> sha256:b5de22c0f5cd2ea2bb6c0524478db95bffa294c99419ccda9d3ccc9873bad9 24.05MB / 24.05MB
=> sha256:917ee5330e73737d0095a80233d3116488959399ff2c06715089016e270f863 64.13MB / 64.13MB
=> sha256:718a8a89d1f719227b0d27a510a99024b5613a5467f2c3e267079e93bce 2.01kB / 2.01kB
=> sha256:b43b-a98d5f5be01606388820047f1c0b421722e9e63c12757474305bd6702 211.10MB / 211.10MB
=> sha256:7fad4bffe2444237b82386b9b704d8ac48a54eeec2c992e377a3a28da49b98d3 6.39MB / 6.39MB
=> extracting sha256:bc0734b949dcabc5bdf0c8b9f44491e0fce04cb10c9c6e76282b9f6abdf01
=> extracting sha256:b5de22c0f5cd2ea2bb6c0524478db95bffa294c99419ccda9d3ccc9873bad9
=> sha256:1f68ce6a3e62331ec49a8265d0ec7d7cd28317b9b53c1bcb35eb3bd8ea742478 19.80MB / 19.80MB
=> sha256:e27d998f416bd7b65491e40c3adbec0b854c064a921baaf08543374934d273 244B / 244B
=> extracting sha256:917ee5330e73737d0095a80233d3116488959399ff2c06715089016e270f863
=> sha256:fefdc9854bf74766b974cb4e3415b361e5c0d572f6774f0e6c53b7a2c599ee 3.11MB / 3.11MB
=> extracting sha256:b43bd898d5f5be01606388820047f1c0b421722e9e63c12757474305bd6702
=> extracting sha256:7fad4bffe2444237b82386b9b704d8ac48a54eeec2c992e377a3a28da49b98d3
=> extracting sha256:1f68ce6a3e62331ec49a8265d0ec7d7cd28317b9b53c1bcb35eb3bd8ea742478
=> extracting sha256:e27d998f416bd7b65491e40c3adbec0b854c064a921baaf08543374934d273
```

Рисунок 2.2 – Создание образа

```
(.venv) D:\Рабочий стол\gb\python1\docker run -p 5000:5000 --name python2 -d b38bcb2f7c533fa75daf07f6c219fe19ee8f2852aabe1e0ddcaf40c733ec78
f4473f41870e97eced2b14e9eb1368d1b9276c4156c52123042edd57bd37cfc
docker: Error response from daemon: driver failed programming external connectivity on endpoint python2 (26a18e0644f7817402a2eb787f478c86b0ccc2476daf45149cc8d4b0
757cd36): Bind for 0.0.0.0:5000 failed: port is already allocated.
```

Рисунок 2.3 – Создание 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. Запустите контейнер из образа Docker с помощью команды docker run. Доступ к переменной среды из приложения с помощью соответствующей переменной окружения.

```
1 import os
2
3 def main():
4     database_url = os.environ["DATABASE_URL"]
5     print(database_url)
6 if __name__ == "__main__":
7     main()
```

Рисунок 3.1 – Измененный файл python

```
1 >> FROM python:3.11 as builder
2 RUN mkdir /usr/src/app
3 COPY ./my-app /usr/src/app
4 COPY ./requirements.txt /usr/src/app
5
6 WORKDIR /usr/src/app
7
8 RUN pip install --no-cache-dir -r requirements.txt
9 ENV DATABASE_URL postgres://user:password@localhost:5432/database
10
11 EXPOSE 5000
12 CMD [ "python", "app.py" ]
```

Рисунок 3.2 – Измененный файл докер

```
(.venv) D:\Рабочий стол\gb\python1>docker build -t zadanie3 .
[+] Building 3.6s (9/11)
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 344B
=> [internal] load metadata for docker.io/library/python:3.11
=> [auth] library/python:pull token for registry-1.docker.io
=> [1/6] FROM docker.io/library/python:3.11@sha256:4e5e9b05dda9cf699084f20bb1d3463234446387fa0f7a45d90689c48e204c83
=> [internal] load build context
=> => transferring context: 281B
=> CACHED [2/6] RUN mkdir /usr/src/app
=> [3/6] COPY ./my-app /usr/src/app
=> [4/6] COPY ./requirements.txt /usr/src/app
```

Рисунок 3.3 – Создание образа


```
(.venv) D:\Рабочий стол\gb\python1>docker run -it -e DATABASE_URL=postgres://user:password@localhost:5432/database zadanie3  
postgres://user:password@localhost:5432/database
```

Рисунок 3.4 – Создание 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. Используйте команду COPY в Dockerfile для определения файлов для копирования и их местоположения в образе. Определите файлы для копирования в образ Docker с помощью команды COPY в Dockerfile. Укажите исходное расположение файлов и их местоположение в образе. Соберите образ Docker с помощью команды docker build. Запустите контейнер из образа Docker с помощью команды docker run0.

```
1 >> FROM python:3.11 as T builder
2 RUN mkdir /usr/src/app
3 COPY ./my-app /usr/src/app
4 COPY ./requirements.txt /usr/src/app
5
6 WORKDIR /usr/src/app
7
8 RUN pip install --no-cache-dir -r requirements.txt
9 EXPOSE 5000
10 CMD [ "python", "app.py" ]
```

Рисунок 4.1 - Dockerfile

```
(.venv) D:\Рабочий стол\gb\python1>docker build -t zadanie4 .  
[+] Building 2.0s (3/4)  
=> [internal] load .dockerignore  
=> => transferring context: 2B  
=> [internal] load build definition from Dockerfile  
=> => transferring dockerfile: 275B  
=> [internal] load metadata for docker.io/library/python:3.11  
=> [auth] library/python:pull token for registry-1.docker.io
```

Рисунок 4.2 – Создание образа

```
(.venv) D:\Рабочий стол\gb\python1>docker run -p 5000:5000 --name python4 -d zadanie4
2d8527ec0e4a7404721e4736bbd2dd0149bed8d5a06f4f5e52cc261901d90a20

(.venv) D:\Рабочий стол\gb\python1>
```

Рисунок 4.3 – Создание контейнера

Задача 5: Запуск команд при запуске контейнера

Цель: Выполнить команды инициализации или настройки при запуске контейнера вебприложения. Используйте команду RUN в Dockerfile для определения команд для выполнения и их параметров. 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. Соберите образ Docker с помощью команды docker build. Запустите контейнер из образа Docker с помощью команды docker run.

```
(.venv) D:\Рабочий стол\gb\python1>docker build -t zadanie5 .
[+] Building 0.9s (3/3)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 275B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/python:3.11
```

Рисунок 5.1 – Создание образа

```
(.venv) D:\Рабочий стол\gb\python1>docker run -p 5000:5000 --name python5 -d zadanie5
7e45da875e51d0e3e85b92680daf5dc11bca1dec40fb9d079d68dd50d5e00100
```

Рисунок 5.2 – Создание контейнера

6. В ходе выполнения работы были получены 5 контейнеров и 5 образов.

<input type="checkbox"/>	zadanie4 4d4d4cf192d7	latest	In use	27 minutes ago	1.02 GB	▶	:	🗑
<input type="checkbox"/>	zadanie5 4d4d4cf192d7	latest	In use	27 minutes ago	1.02 GB	▶	:	🗑
<input type="checkbox"/>	zadanie3 4c2ed84aea22	latest	In use	27 minutes ago	1.02 GB	▶	:	🗑
<input type="checkbox"/>	multy-python1 b38becb2f7c53	latest	In use	45 minutes ago	130.55 MB	▶	:	🗑
<input type="checkbox"/>	python1 0d70d472f6969	latest	In use	1 hour ago	146.06 MB	▶	:	🗑

Рисунок 6.1 – Образы



































<input type="checkbox"/>	Name	Image	Status	CPU (%)	Port(s)	Last start... ↓	Actions
<input type="checkbox"/>	 python5 7e45da875e51 	zadanie5	Exited (1)	N/A	5000:5000 	2 minutes ago	  
<input type="checkbox"/>	 python4 2d8527ec0e4a 	zadanie4	Exited (1)	N/A	5000:5000 	9 minutes ago	  
<input type="checkbox"/>	 vigilant_chaum 6c37d829b622 	zadanie3	Exited	N/A		24 minutes ago	  
<input type="checkbox"/>	 great_varahamihira 977ca2eaa9db 	zadanie3	Exited	N/A		25 minutes ago	  
<input type="checkbox"/>	 python2 f4473f41870e 	b38bcb2f7c533fa75dabf07	Exited (1)	N/A	5000:5000 	42 minutes ago	  
<input type="checkbox"/>	 python1 4ea4306320d8 	python1	Exited (137)	N/A	5000:5000 	1 hour ago	  

Рисунок 6.2 - Контейнеры

Контрольные вопросы:

1. Что такое Dockerfile?

Dockerfile - это текстовый файл, содержащий инструкции для автоматизированного создания Docker-образа. В Dockerfile определяются шаги и настройки, необходимые для сборки контейнера.

2. Какие основные команды используются в Dockerfile?

Основные команды в Dockerfile включают FROM, RUN, COPY, CMD, EXPOSE, WORKDIR, ENV, USER, HEALTHCHECK, LABEL, ARG, ONBUILD, и другие.

3. Для чего используется команда FROM?

Команда FROM определяет базовый образ, который будет использован для создания нового образа.

4. Для чего используется команда WORKDIR?

Команда WORKDIR устанавливает текущий рабочий каталог для следующих инструкций в Dockerfile.

5. Для чего используется команда COPY?

Команда COPY копирует файлы или директории из исходной директории (контекст сборки) в образ.

6. Для чего используется команда RUN?

Команда RUN выполняет команды внутри контейнера во время сборки образа.

7. Для чего используется команда CMD?

Команда CMD устанавливает команду по умолчанию, которая будет выполнена при запуске контейнера.

8. Для чего используется команда EXPOSE?

Команда EXPOSE указывает на порт, который контейнер будет слушать во время выполнения.

9. Для чего используется команда ENV?

Команда ENV устанавливает переменные среды в образе.

10. Для чего используется команда USER?

Команда USER устанавливает пользователя, от имени которого будут выполняться инструкции RUN, CMD, и ENTRYPOINT в Dockerfile.

11. Для чего используется команда HEALTHCHECK?

Команда HEALTHCHECK добавляет инструкции для проверки состояния контейнера во время выполнения.

12. Для чего используется команда LABEL?

Команда LABEL в Dockerfile используется для добавления метаданных к образу. Эти метаданные могут включать в себя информацию о версии, описании, авторе и другие пользовательские метки.

13. Для чего используется команда ARG?

Команда ARG в Dockerfile определяет переменные, которые могут быть переданы при сборке образа через флаг --build-arg. Это позволяет параметризовать сборку образа.

14. Для чего используется команда ONBUILD?

Команда ONBUILD в Dockerfile добавляет инструкции, которые будут выполнены автоматически при использовании созданного образа в качестве базового образа для другого Dockerfile. Это позволяет автоматизировать определенные шаги в сборке.

15. Что такое многоэтапная сборка?

Многоэтапная сборка (multi-stage build) - это подход, при котором Dockerfile содержит несколько этапов (блоков), каждый из которых может использовать разные базовые образы. Этот подход позволяет уменьшить размер конечного образа, так как в итоговый образ включаются только необходимые компоненты.

16. Какие преимущества использования многоэтапной сборки?

Преимущества многоэтапной сборки:

- Уменьшение размера образа.

- Лучшая чистота и безопасность, так как в конечный образ включаются только необходимые файлы.

- Упрощение сценариев сборки и управления зависимостями.

17. Какие недостатки использования многоэтапной сборки?

Усложнение конфигурации Dockerfile. Дополнительная работа по разделению и настройке этапов сборки.

18. Как определить базовый образ в Dockerfile?

Используйте команду FROM с указанием имени базового образа и его тега.

19. Как определить рабочую директорию в Dockerfile?

Используйте команду WORKDIR, указав путь к рабочей директории внутри контейнера.

20. Как скопировать файлы в образ Docker?

Используйте команду COPY в Dockerfile, указав путь к файлам в контексте сборки и путь внутри образа.

21. Как выполнить команды при сборке образа Docker?

Используйте команду RUN в Dockerfile для выполнения команд во время сборки образа.

22. Как указать команду запуска контейнера?

Используйте команду CMD в Dockerfile для установки команды, которая будет выполнена при запуске контейнера.

23. Как открыть порты в контейнере?

Используйте команду EXPOSE в Dockerfile для указания портов, которые контейнер будет слушать во время выполнения.

24. Как задать переменные среды в образе Docker?

Используйте команду ENV в Dockerfile для установки переменных среды.

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

Используйте команду USER в Dockerfile для указания пользователя, от имени которого будут выполняться следующие инструкции (RUN, CMD, ENTRYPOINT).

26. Как добавить проверку работоспособности к контейнеру?

Используйте команду HEALTHCHECK в Dockerfile для добавления проверки состояния контейнера во время выполнения.

27. Как добавить метку к контейнеру?

Используйте команду LABEL в Dockerfile для добавления метаданных (меток) к образу.

28. Как передать аргументы при сборке образа Docker?

Используйте команду ARG в Dockerfile для определения переменных, которые можно передавать при сборке образа через флаг --build-arg.

29. Как выполнить команду при первом запуске контейнера?

Используйте опцию --entrypoint при запуске контейнера с docker run, или определите команду в виде аргумента в CMD.

30. Как определить зависимости между образами Docker?

В Dockerfile можно определить зависимости, используя команду FROM для указания базового образа. Дополнительные зависимости могут быть выражены в виде инструкций, которые требуют выполнения определенных шагов при сборке.