



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации информационных  
технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 4  
по дисциплине

«Технологии разработки информационных систем»

Тема: « Docker »

Выполнил студент группы ИНБО-12-13

Александр И.В.

Принял

Петров И.А.

Практическая работа выполнена

«19» 04 2025 г.

А  
(подпись студента)

«Зачтено»

«30» 04 2025 г.

В  
(подпись руководителя)

Москва 2025

# СОДЕРЖАНИЕ

1. ПОСТАНОВКА ЗАДАЧИ.....	3
2. ОСНОВНАЯ ЧАСТЬ .....	4
3. ВЫВОДЫ.....	15

## 1. ПОСТАНОВКА ЗАДАЧИ

В практической работе необходимо выполнить все шаги из разделов 1–7. В отчёт должны быть включены ответы на вопросы, выделенные курсивом, результаты выполнения команд из разделов 1–7, а также выполненное индивидуальное задание (раздел 8): листинг Dockerfile, а также команды сборки и запуска контейнера.

Мой номер студента Албахтин И.В. по списку в группе равен  $N=3$ . Максимальный номер задания в таблице (число вариантов)  $M=15$ . Тогда номер задания  $V$  студента Албахтина И.В. будет равен остатку от деления  $V = \text{Ост}((N-1)/M)+1 = \text{Ост}((3-1)/15)+1 = \text{Ост}(2/15)+1 = 2+1 = 3$

Индивидуальный вариант № 3:

Вариант	Устанавливаемый пакет
3	zip

## 2. ОСНОВНАЯ ЧАСТЬ

### 2.1 Образы:

Проверим наличие имеющихся образов Docker:

```
C:\Windows\System32>docker images
REPOSITORY    TAG          IMAGE ID     CREATED      SIZE
```

Рисунок 2.1 — Проверка наличия образов Docker

Как мы видим, репозиторий пуст. Загрузим образ Ubuntu с помощью соответствующей команды:

```
C:\Windows\System32>docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
2726e237d1a3: Pull complete
Digest: sha256:1e622c5f073b4f6bfad6632f2616c7f59ef256e96fe78bf6a595d1dc4376ac02
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

Рисунок 2.2 — Загрузка образа Ubuntu

Проверим наличие образов еще раз:

```
C:\Windows\System32>docker images
REPOSITORY    TAG          IMAGE ID     CREATED      SIZE
ubuntu        latest       1e622c5f073b 3 weeks ago  117MB
```

Рисунок 2.3 — Проверка наличия образов Docker

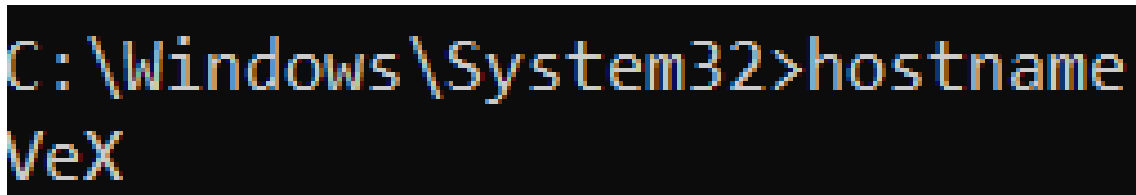
Посмотрим список контейнеров с помощью соответствующей команды:

```
C:\Windows\System32>docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Рисунок 2.4 — Список существующих контейнеров(пуст)

## 2.2 Изоляция:

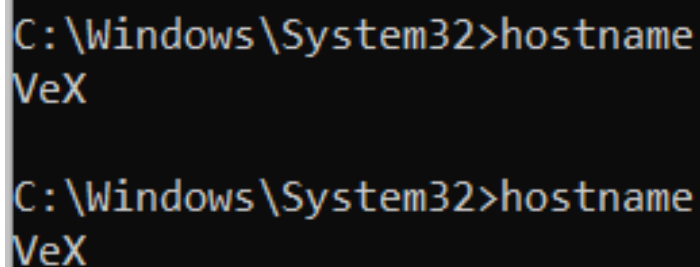
Посмотрим информацию о хостовой системе с помощью команды `hostname`:



```
C:\Windows\System32>hostname  
VeX
```

Рисунок 2.5 — Информация о хостовой системе

Сделаем это еще раз



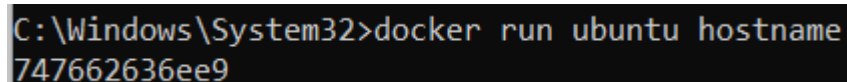
```
C:\Windows\System32>hostname  
VeX  
  
C:\Windows\System32>hostname  
VeX
```

Рисунок 2.6 — Информация о хостовой системе – повторный запрос

Вопрос: Одинаковый ли результат получился при разных запусках?

Ответ: Да, так как между запросами не проводилось никаких действий, влияющих на результат

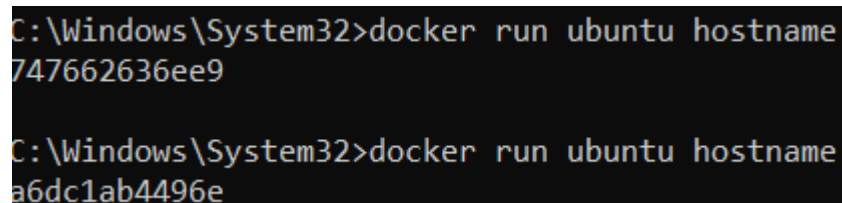
Попробуем запросить информацию о хостовой системе от скачанного образа Ubuntu:



```
C:\Windows\System32>docker run ubuntu hostname  
747662636ee9
```

Рисунок 2.7 — Информация о хостовой системе от Ubuntu

Попробуем еще раз:



```
C:\Windows\System32>docker run ubuntu hostname  
747662636ee9  
  
C:\Windows\System32>docker run ubuntu hostname  
a6dc1ab4496e
```

Рисунок 2.8 — Информация о хостовой системе от Ubuntu – повторный запрос

Вопрос: Одинаковый ли результат получился при разных запусках?

Ответ: Нет, так как из одного образа ubuntu были запущены два изолированных контейнера

Проверим нашу теорию, еще раз запросив список контейнеров:

```
C:\Windows\System32>docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a6dc1ab4496e	ubuntu	"hostname"	29 seconds ago	Exited (0) 28 seconds ago		gracious_driscoll
747662636ee9	ubuntu	"hostname"	42 seconds ago	Exited (0) 41 seconds ago		hungry_shirley

Рисунок 2.9 — Список существующих контейнеров

Действительно, мы видим два изолированных контейнера

Попробуем запустить bash в контейнере с помощью соответствующей команды:

```
C:\Windows\System32>docker run ubuntu bash
C:\Windows\System32>
```

Рисунок 2.10 — Попытка запуска bash

В этом случае ничего не произошло, так как интерактивные оболочки выйдут после выполнения любых скриптовых команд, если только они не будут 1 запущены в интерактивном терминале. Попробуем еще раз, отредактировав команду:

```
C:\Windows\System32>docker run -it ubuntu bash
root@87f44b5827cf:/# ^C
```

Рисунок 2.11 — Запуск bash

## 2.3 Работа с портами:

Для начала загрузим образ python с помощью соответствующей команды:

```
C:\Windows\System32>docker pull python
Using default tag: latest
latest: Pulling from library/python
ab89b3116421: Pull complete
ca513cad200b: Pull complete
c187b51b626e: Pull complete
39ca2d92e129: Pull complete
cf05a52c0235: Pull complete
63964a8518f5: Pull complete
776493ee5e4c: Pull complete
Digest: sha256:fdbba75f9d66e120e35bf89a384027518fdcf79902b751473cd80bc0d59adfda
Status: Downloaded newer image for python:latest
docker.io/library/python:latest
```

**Рисунок 2.12 — Загрузка образа python**

В качестве примера запустим встроенный в Python модуль веб-сервера из корня контейнера, чтобы отобразить содержание контейнера:

```
C:\Windows\System32>docker run -it python python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

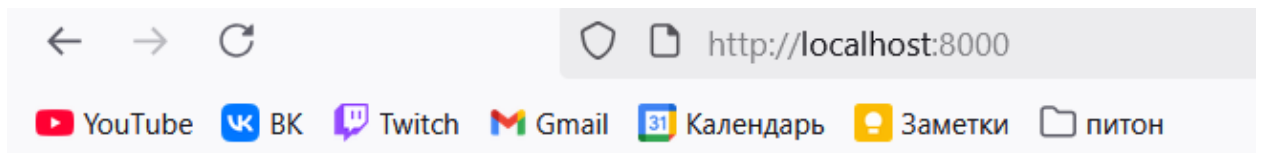
**Рисунок 2.13 — Запуск веб-сервера Python**

Однако, если открыть этот адрес, то ничего не будет видно, потому что порты не проброшены. Запустим сервер еще раз, учтя этот момент:

```
C:\Windows\System32>docker run -it -p8000:8000 python python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

**Рисунок 2.14 — Запуск веб-сервера Python с перебросом портов**

Проверим корректность работы веб-сервера:



## Directory listing for /

---

- [.dockerenv](#)
- [bin/](#)
- [boot/](#)
- [dev/](#)
- [etc/](#)
- [home/](#)
- [lib/](#)
- [lib64/](#)
- [media/](#)
- [mnt/](#)
- [opt/](#)
- [proc/](#)
- [root/](#)
- [run/](#)
- [sbin/](#)
- [srv/](#)
- [sys/](#)
- [tmp/](#)
- [usr/](#)
- [var/](#)

Рисунок 2.15 — Содержимое веб-страницы

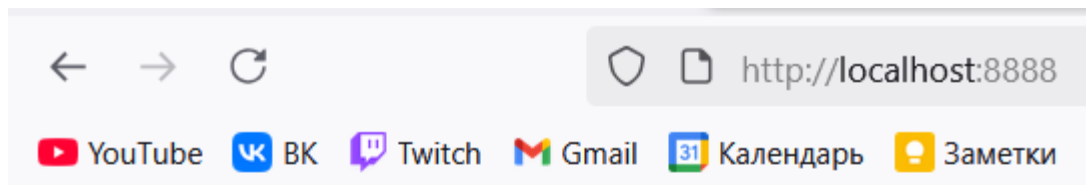
Во время работы сервера изменим его порт с помощью специальной команды:

```
C:\Windows\System32>docker run -it -p8888:8000 python python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Рисунок 2.16 — Изменение порта сервера

Проверим внесенные изменения:





## Directory listing for /

- [.dockerenv](#)
- [bin@](#)
- [boot/](#)
- [dev/](#)
- [etc/](#)
- [home/](#)
- [lib@](#)
- [lib64@](#)
- [media/](#)
- [mnt/](#)
- [opt/](#)
- [proc/](#)
- [root/](#)
- [run/](#)
- [sbin@](#)
- [srv/](#)
- [sys/](#)
- [tmp/](#)
- [usr/](#)
- [var/](#)

Рисунок 2.17 — Новый порт сервера

### 2. 4 Именованные контейнеры, остановка и удаление:

Запустим контейнер веб-сервера и переведем в фоновый режим с помощью соответствующих команд:

```
C:\Windows\System32>docker run -p8000:8000 --name pyserver -d python python -m http.server f13a89cd1703ac1df4e3568dc9e1a12b81ed09ab8a6456e61b4e8a1269286101
```

Рисунок 2.18 — Запуск сервера в фоновом режиме

Проверим, что сервер все еще работает:

```
C:\Windows\System32> docker ps | grep pyserver
782f4fe56ea4  python  "python -m http.serv..."  9 seconds ago  Up 9 seconds  0.0.0.0:8000->8000/tcp  pyserver
```

Рисунок 2.19 — Проверка функционирования сервера

Остановим процесс работы сервера с помощью соответствующей команды:

```
C:\Windows\System32>docker stop pyserver
pyserver
```

**Рисунок 2.20 — Завершение работы сервера**

Однако завершение процесса не означает удаление контейнера. Попробуем создать контейнер с тем же именем и увидим ошибку:

```
C:\Windows\System32>docker run -it -p8000:8000 --name pyserver -d python python -m http.server
docker: Error response from daemon: Conflict. The container name "/pyserver" is already in use by container "f13a89cd1703ac1df4e3568dc9e1a12b81ed09ab8a6456e61b4e8a1269286101". You have to remove (or rename) that container to be able to reuse that name.
Run 'docker run --help' for more information
```

**Рисунок 2.21 — Попытка создания дубликата сервера**

Чтобы полностью удалить контейнер, воспользуемся командой `docker rm`:

```
C:\Windows\System32>docker rm -f pyserver
pyserver
```

**Рисунок 2.22 — Удаление контейнера pyserver**

## 2.5 Постоянное хранение данных:

Для начала запустим сервер, который будет отображать информацию о файлах в директории `/mnt`

```
C:\Windows\System32>docker run -p8000:8000 --name pyserver --rm -d python python -m http.server -d /mnt
f2fd2583944ba75e44c149e5a8b14af32a7564952227aeb6e2e8133b3761d839
```

**Рисунок 2.23 — Запуск pyserver**

Вопрос: Что значат остальные флаги запуска? Где здесь команда, которая выполнится в контейнере?

Ответ: Флаги запуска:

`--rm` – процесс удалится после завершения.

`--name` – выдача имени контейнеру

Попадем в контейнер с помощью соответствующей команды и создадим текстовый файл:

```
C:\Windows\System32>docker exec -it pyserver bash
root@f2fd2583944b:/# cd /mnt && echo "hello world" > hi.txt
root@f2fd2583944b:/mnt# exit
exit
```

**Рисунок 2.24 — Создание текстового файла**

Проверим, появился ли созданный файл в контейнере:

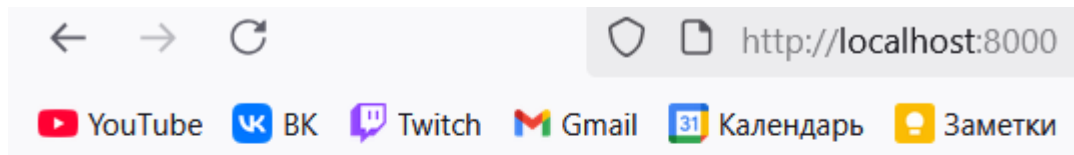


Рисунок 2.25 — Страница веб-сервера

Остановим контейнер и снова создадим. Проверим веб-сервер еще раз:

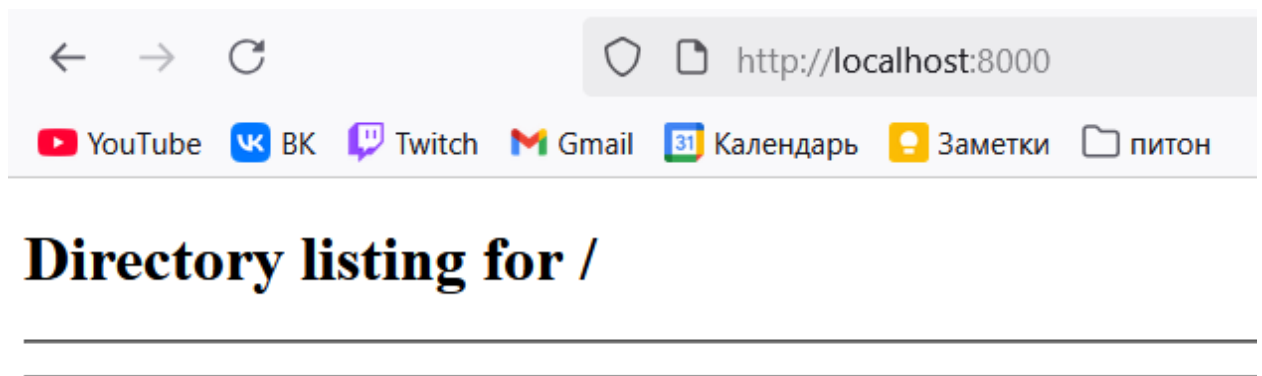


Рисунок 2.26 — Страница веб-сервера после перезапуска

Как мы видим, файл исчез – после остановки сервера он удалился, так как не был сохранен. Для сохранения файлов существуют два способа.

### 2.5.1 Способ №1:

Первый способ — это создать отдельный том с помощью ключа `-v myvolume:/mnt`, где `myvolume` — название тома, `/mnt` — директория в контейнере, где будут доступны данные.

```
C:\Windows\System32>docker run -p8000:8000 --rm --name pyserver -d -v /myfiles:/mnt python python -m http.server -d /mnt
4a5e11ff88b9c4ebf6600dcba57f73bee59a4654651ac73ccb119b4b733b3c03
```

Рисунок 2.27 — Создание контейнера с примонтированным томом

Затем создадим файл тем же способом:

```
C:\Windows\System32>docker exec -it pyserver bash
root@4a5e11ff88b9:/# cd mnt && echo "hello world" > hi.txt
root@4a5e11ff88b9:/mnt# exit
exit
```

Рисунок 2.28 — Создание текстового файла

Убедимся в том, что файл сохранился

```
C:\Windows\System32>docker inspect -f "{{json .Mounts }}" pyserver
[{"Type":"bind","Source":"/myfiles","Destination":"/mnt","Mode":"","RW":true,"Propagation":"rprivate"}]
```

Рисунок 2.29 — Путь до тома с файлом

## 2.5.2 Способ №2:

Второй способ - монтирование директорий и файлов.

```
C:\Windows\System32\myfiles>touch host.txt

C:\Windows\System32\myfiles>
```

Рисунок 2.30 — Создание текстового файла

Запустим контейнер с помощью соответствующей команды:

```
C:\Windows\System32\myfiles>docker run -p8000:8000 --rm --name pyserver -d -v /myfiles:/mnt python python -m http.server -d /mnt
2a31a13bce870590ea88d672c8174e7a7f7f2571bcfc0df325593fb11bbb285e
```

Рисунок 2.31 — Запуск конвейера

С помощью bash посмотрим содержимое директории /mnt.

```
C:\Windows\System32\myfiles>docker exec -it pyserver bash
root@1368733aadb9:/# cd /mnt
root@1368733aadb9:/mnt# ls
hi.txt
root@1368733aadb9:/mnt#
```

Рисунок 2.32 — Содержимое директории /mnt

Как мы видим, там содержится файл hi.txt.

## 2.6 Переменные окружения:

Для передачи переменных окружения внутрь контейнера используется ключ -e. Например, чтобы передать в контейнер переменную окружения MIREA во значение «ONE LOVE», нужно добавить ключ -e MIREA="ONE LOVE".

Проверим, выведя все переменные окружения, определённые в контейнере с помощью утилиты:

```
C:\Windows\System32\myfiles>docker run -it --rm -e MIREA="ONE LOVE" ubuntu env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=fab2a4a3ef1d
TERM=xterm
MIREA=ONE LOVE
HOME=/root
```

Рисунок 2.33 — Вывод всех переменных окружений

## 2.7 Dockerfile:

Соберем образ, в который будут установлены дополнительные пакеты, примонтируем директорию и установим команду запуска с помощью последовательности команд:

```
C:\Windows\System32\myfiles>touch Dockerfile
```

Рисунок 2.34 — Создание файла Dockerfile

```
FROM ubuntu:20.04
RUN apt update \
    && apt install -y python3 fortune \
    && cd /usr/bin \
    && ln -s python3 python
RUN /usr/games/fortune > /mnt/greeting-while-building.txt
ADD ./data /mnt/data
EXPOSE 80
CMD ["python", "-m", "http.server", "-d", "/mnt/", "80"]
```

Рисунок 2.35 — Заполнение файла Dockerfile

```
C:\Windows\System32\myfiles>docker build -t mycoolimage .
[+] Building 34.9s (9/9) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 308B
=> [internal] load metadata for docker.io/library/ubuntu:20.04
=> [internal] load .dockerignore
=> => transferring context: 2B
=> CACHED [1/4] FROM docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2fafe15388214
=> => resolve docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2fafe15388214
=> [internal] load build context
=> => transferring context: 26B
=> [2/4] RUN apt update && apt install -y python3 fortune && cd /usr/bin && ln -s python3 python
=> [3/4] RUN /usr/games/fortune > /mnt/greeting-while-building.txt
=> [4/4] ADD ./data /mnt/data
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:79a9de73c46293fbc44d58503d89464a39234b1322b6e709f3eec46244462e81
=> => exporting config sha256:5789688cfdb39866bc3968d1facb78e72ae5935dfe8bf6774112d60e7d3ecd9f
=> => exporting attestation manifest sha256:b3a1c36ffbc00a5181e96dca3bc365a4dd9671a68ce016aefaaab3ecf7443a
=> => exporting manifest list sha256:2f55b4b781f68aa2a492fb1eb39bdeceaa15c3567af2b3a72aa9bf33ef0276f8
=> => naming to docker.io/library/mycoolimage:latest
=> => unpacking to docker.io/library/mycoolimage:latest
```

Рисунок 2.36 — Успешный build

```
C:\Windows\System32\myfiles>docker run --rm -it -p8099:80 mycoolimage
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Рисунок 2.37 — Успешный запуск

## 2.8 Индивидуальный вариант:

Вариант	Устанавливаемый пакет
3	zip

```
C: > Windows > System32 > myfiles > Dockerfile
1 FROM ubuntu:20.04
2 RUN apt update && \
3     apt install -y python3 zip && \
4     ln -s /usr/bin/python3 /usr/bin/python
5 RUN mkdir -p /mnt/files
6 EXPOSE 80
7 CMD ["python", "-m", "http.server", "80", "-d", "/mnt/files"]
```

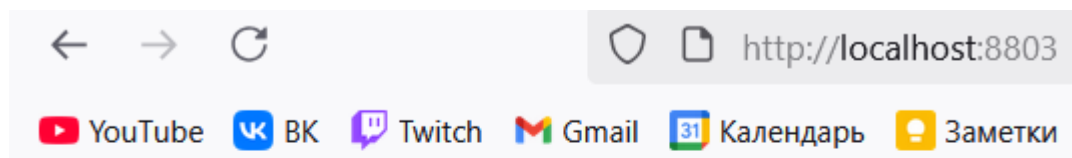
Рисунок 2.38 — Содержимое Dockerfile

```
C:\Windows\System32\myfiles>docker build -t variant3-image .
[+] Building 38.7s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 251B
=> [internal] load metadata for docker.io/library/ubuntu:20.04
=> [internal] load .dockerignore
=> => transferring context: 2B
=> CACHED [1/3] FROM docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2fafa15388214
=> => resolve docker.io/library/ubuntu:20.04@sha256:8feb4d8ca5354def3d8fce243717141ce31e2c428701f6682bd2fafa15388214
=> [2/3] RUN apt update && apt install -y python3 zip && ln -s /usr/bin/python3 /usr/bin/python
=> [3/3] RUN mkdir -p /mnt/files
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:683153b39ce4209e56debac8a1a8c4ed29f4caf30ec66185f8b0357252bd427b
=> => exporting config sha256:44c9e0d7cd747acdc1f8969a6d05217fec959fa6e95c015d490b24c60a99052
=> => exporting attestation manifest sha256:31333554ef2c744b10139dab158653ea1054204807bd0cc24c81828a116f0d44
=> => exporting manifest list sha256:82a78dbe71a3dbf17b34dbf0d02f15419c6423bef8e586915c9236918c417e60
=> => naming to docker.io/library/variant3-image:latest
=> => unpacking to docker.io/library/variant3-image:latest
```

Рисунок 2.39 — Успешный build

```
C:\Windows\System32\myfiles>docker run --rm -it -p8804:8804 variant_4
Serving HTTP on 0.0.0.0 port 8804 (http://0.0.0.0:8804/) ...
```

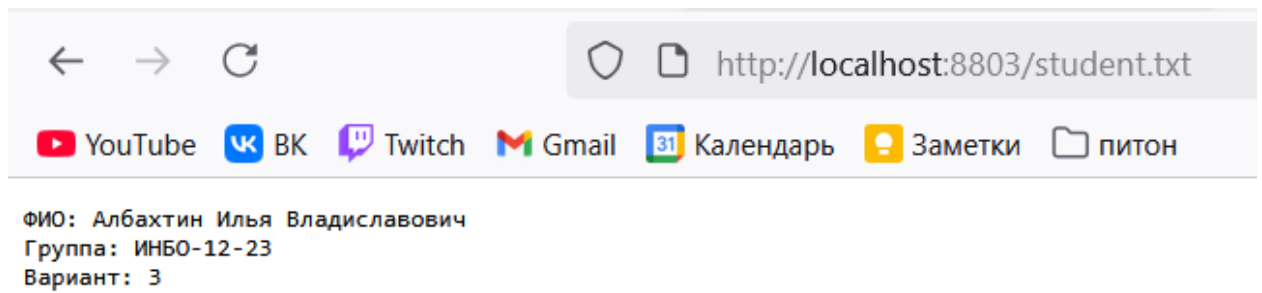
Рисунок 2.39 — Успешный запуск



## Directory listing for /

- [student.txt](#)

Рисунок 2.40 — Запущенный сервер



**Рисунок 2.41 — Содержимое файла student.txt**

### **3. ВЫВОДЫ**

Было произведено знакомство с программой Docker, управлением контейнерами и взаимодействием с прочими атрибутами системы.