|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

**Отчет по практической работе №4**

по дисциплине «Технологии разработки программных приложений»

**Тема практической работы:** «Docker»

|  |  |  |  |
| --- | --- | --- | --- |
| **Выполнил:**  Студент группыИКБО-41-23 | | Трофимов А.А. | |
| **Проверил:** | Доцент кафедры МОСИТ,  кандидат технических наук, доцент Жматов Д.В. | |

Москва 2025

**ОГЛАВЛЕНИЕ**

[**ЦЕЛЬ РАБОТЫ** 3](#_Toc195035904)

[**ЗАДАНИЯ** 4](#_Toc195035905)

[**1 Образы** 4](#_Toc195035906)

[**2 Изоляция** 5](#_Toc195035907)

[**3 Работа с портами** 7](#_Toc195035908)

[**4 Именованные контейнеры, остановка и удаление** 9](#_Toc195035909)

[**5 Постоянное хранение данных** 11](#_Toc195035910)

[**5.1 Тома** 14](#_Toc195035911)

[**5.2 Монтирование директорий и файлов** 15](#_Toc195035912)

[**6 Переменные окружения** 17](#_Toc195035913)

[**7 Dockerfile** 18](#_Toc195035914)

[**8 Индивидуальное задание** 20](#_Toc195035915)

[**ВЫВОД** 23](#_Toc195035916)

[**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ** 24](#_Toc195035917)

**ЦЕЛЬ РАБОТЫ**

Разработать и внедрить контейнеризированное решение с использованием Docker для обеспечения масштабируемости, переносимости и упрощённого развертывания приложения в изолированной среде.

# **ЗАДАНИЯ**

**1 Образы**

Посмотрите на имеющиеся образы: docker images.

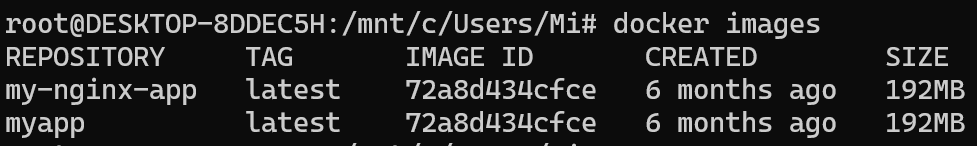


Рисунок 1 – образы докер

Загрузите образ: docker pull ubuntu будет загружен образ ubuntu:latest - последняя доступная версия. Для загрузки конкретной версии, нужно указать тег, например, 12.04: docker pull ubuntu:12.04.

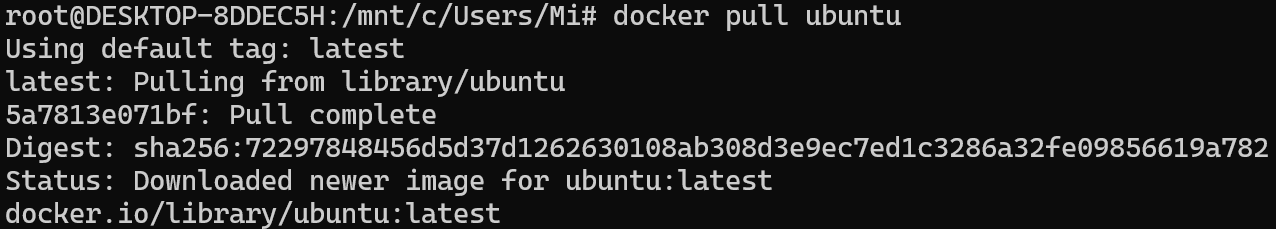


Рисунок 2 – загрузка образа ubuntu

Посмотрите на имеющиеся образы ещё раз: docker images должны появиться новые загруженные образы. Посмотрите список контейнеров, выполнив команду: docker ps -a

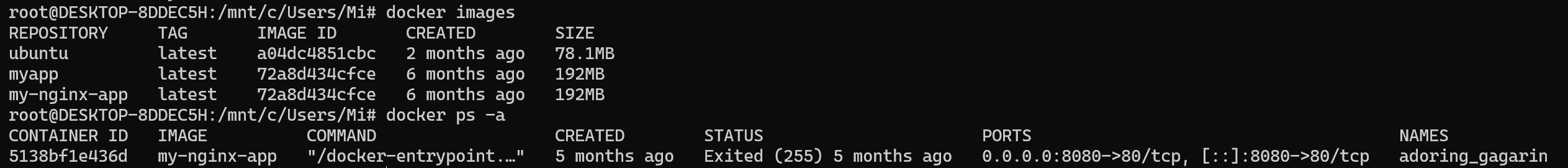


Рисунок 3 – список контейнеров

**2 Изоляция**

Посмотрим информацию о хостовой системе, выполнив команду hostname. Выполните её ещё один раз.

*Вопрос: одинаковый ли результат получился при разных запусках?*

***Ответ:*** *Если выполнить команду hostname дважды на хостовой системе (не в контейнере), результат будет* ***одинаковым****. Эта команда просто выводит имя текущей машины, которое не меняется между запусками (если только его не изменить вручную)*

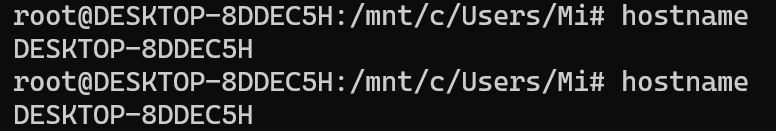


Рисунок 4 – информация о хостовой системе

Попробуем выполнить то же самое в контейнерах. Выполните два раза команду docker run ubuntu hostname.

*Вопрос: Одинаковый ли результат получился при разных запусках?*

***Ответ:*** *Если выполнить docker run ubuntu hostname дважды, результат будет* ***разным****. Потому что каждый запуск docker run создаёт новый контейнер, и Docker генерирует для него случайное уникальное имя (например, friendly\_curie, happy\_mayer и т. д.).*

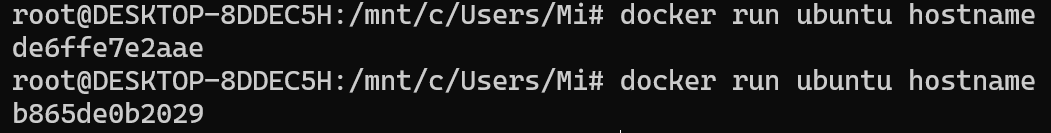
**

Рисунок 5 – информация о контейнерах

В случае запуска команды в контейнерах, ответ будет немного отличаться, будет разный hostname. Так происходит, потому что из одного образа ubuntu были запущены два изолированных контейнера, поэтому у них и был разный hostname.

Заново выполните docker ps -a - там должны появиться запущенные ранее контейнеры.

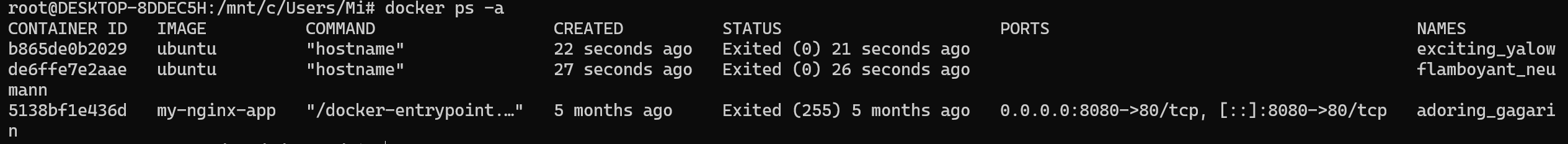


Рисунок 6 – запущенные ранее контейнеры

Запуск контейнеров производится командой: docker run --флаги --докера имя\_контейнера команда для запуска -и --флаги --запуска --программы.

Запустите bash в контейнере: docker run ubuntu bash. Ничего не произошло. Это не баг. Интерактивные оболочки выйдут после выполнения любых скриптовых команд, если только они не будут запущены в интерактивном терминале - поэтому для того, чтобы этот пример не завершился, вам нужно добавить флаги -i -t или сгруппированно -it: docker run -it ubuntu bash.

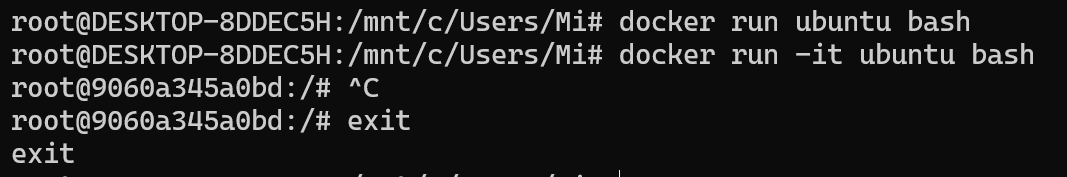


Рисунок 8 – запуск bash в контейнере

Выполняя запуск контейнера, указывая образ ubuntu, неявно указывался образ ubuntu:latest. Следовательно, следующие команды равнозначны:

• docker run ubuntu hostname

• docker run ubuntu:latest hostname

Если бы мы хотели запустить ubuntu:12.04, то нужно было бы выполнить команду docker run ubuntu:12.04 hostname

# **3 Работа с портами**

Для начала, загрузите образ python командой docker pull python.

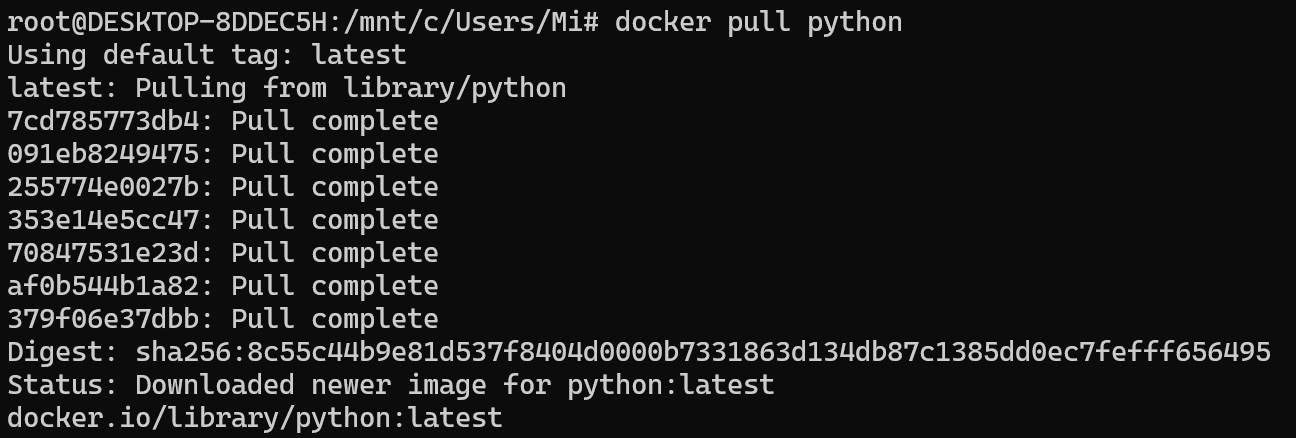


Рисунок 8

В качестве примера, запустите встроенный в Python модуль веб-сервера из корня контейнера, чтобы отобразить содержание контейнера. docker run -it python python -m http.server

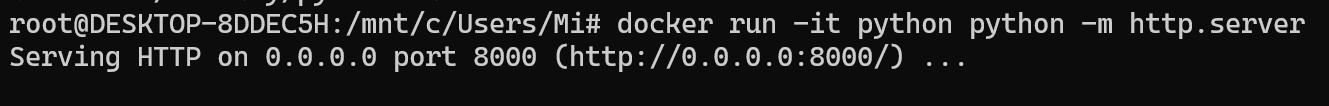


Рисунок 9

При запуске пишется, что сервер доступен по адресу http://0.0.0.0:8000/. Однако, если открыть этот адрес, то ничего не будет видно, потому что порты не проброшены. Завершите работу веб-сервера, нажав комбинацию клавиш Ctrl+C.

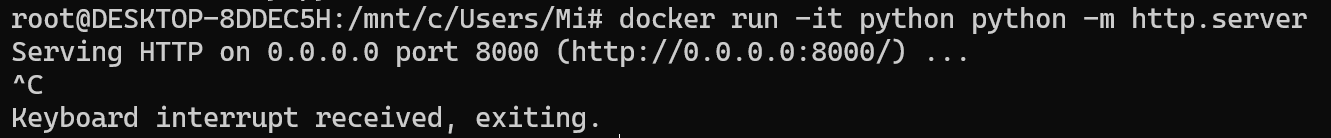


Рисунок 10

Для проброса портов используется флаг -p hostPort:containerPort

Добавьте его, чтобы пробросить порт 8000:

docker run -it -p8000:8000 python python -m http.server - теперь по адресу http://0.0.0.0:8000/ (если не открывается на Windows, то вместо 0.0.0.0 нужно указать localhost) открывается содержимое корневой директории в контейнере.



Рисунок 11

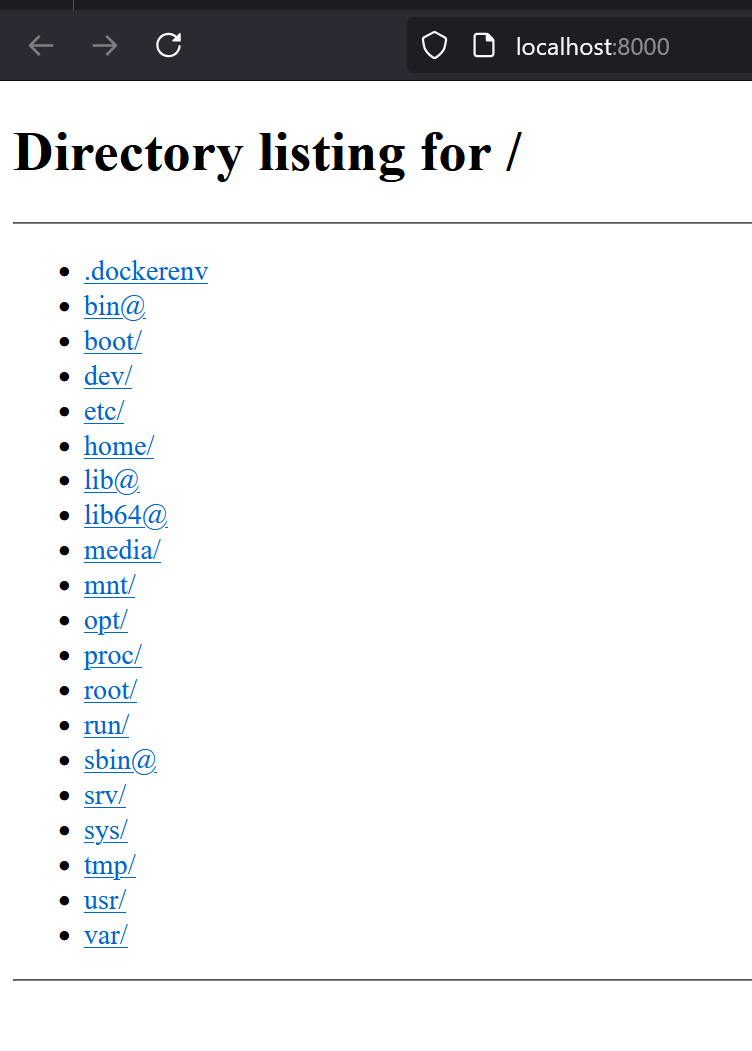


Рисунок 12

Для того, чтобы доступный в контейнере на порту 8000 веб-сайт в хостовой системе открывался на порту 8888, необходимо указать флаг -p 8888:8000: docker run -it -p8888:8000 python python -m http.server.

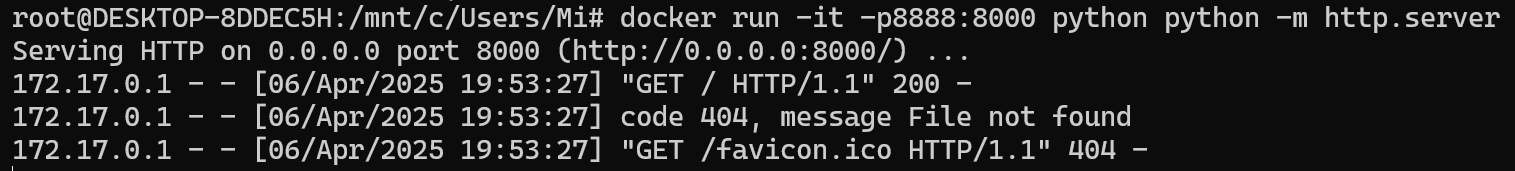


Рисунок 12

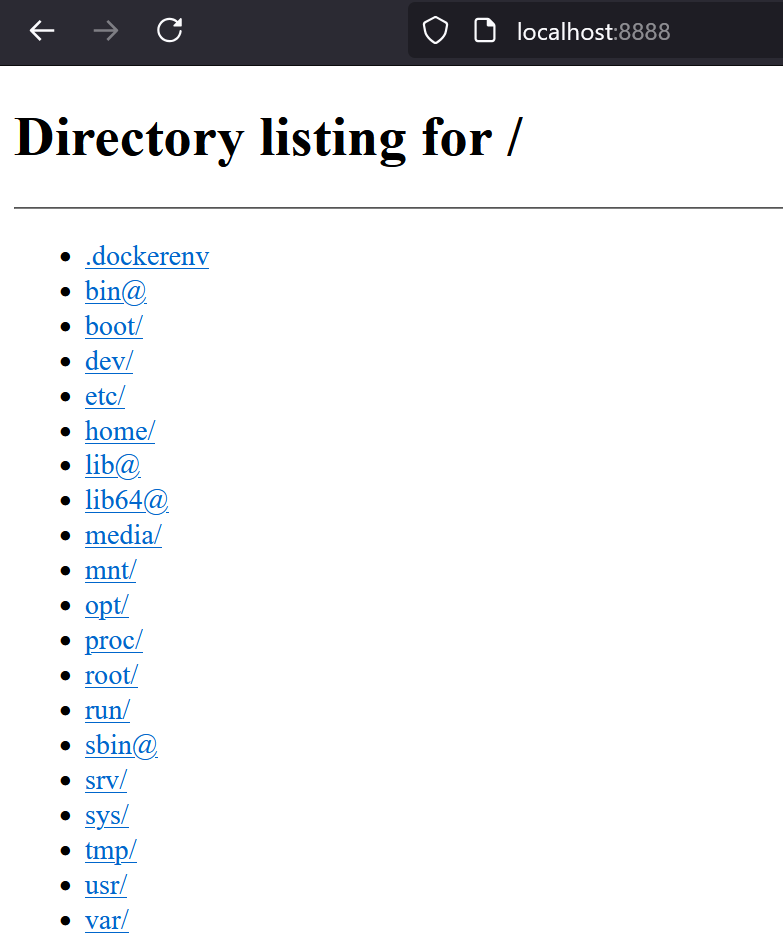


Рисунок 13

Завершите работу веб-сервера, нажав комбинацию клавиш Ctrl+C.

**4 Именованные контейнеры, остановка и удаление**

Запустите контейнер: docker run -it -p8000:8000 python python -m http.server. Нажмите Ctrl+C выполнение завершится. Для того, чтобы запустить контейнер в фоне, нужно добавить флаг -d/--detach. Также определим имя контейнеру, добавив флаг --name.

docker run -p8000:8000 --name pyserver -d python python -m http.server

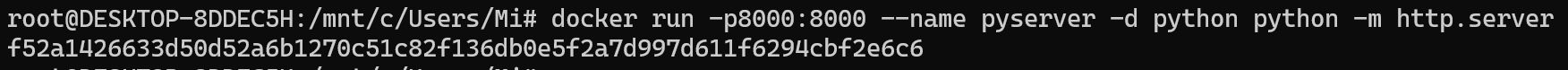
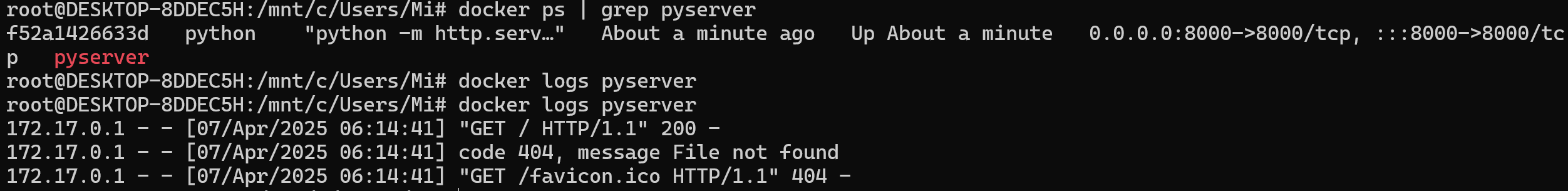


Рисунок 14

Убедитесь, что контейнер всё ещё запущен: docker ps | grep pyserver - вывод команды не должен быть пустым. Для просмотра логов контейнера, воспользуйтесь командой docker logs pyserver.

 Рисунок 15

Для того, чтобы остановить выполнение контейнера, существует команда docker stop pyserver. Однако, если снова попробовать запустить командой docker run -it -p8000:8000 --name pyserver -d python python -m http.server, то возникнет ошибка: контейнер с таким именем существует. Его нужно удалить docker rm pyserver.

Для остановки и удаления контейнера можно воспользоваться командой docker rm -f pyserver вместо выполнения двух отдельных команд stop и rm. После удаления контейнер с таким именем можно будет создать заново.

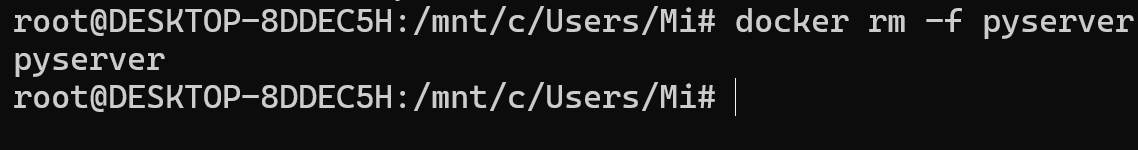


Рисунок 16

Для того, чтобы контейнер удалялся после завершения работы, нужно указать флаг –rm при его запуске - далее в работе мы будем использовать данный флаг:

docker run --rm -p8000:8000 --name pyserver -d python python -m http.server

**5 Постоянное хранение данных**

Запустите контейнер, в котором веб-сервер будет отдавать содержимое директории /mnt: docker run -p8000:8000 --name pyserver --rm -d python python -m http.server -d /mnt, где -d /mnt указывает модулю http.server какая директория будет корневой для отображения.

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

***Ответ:***

|  |  |
| --- | --- |
| Флаг | Описание |
| -p 8000:8000 | Пробрасывает порт:<хост\_порт>:<контейнер\_порт>. Веб-сервер в контейнере слушает 8000, и этот порт становится доступен на хосте тоже на 8000. |
| --name pyserver | Задаёт имя контейнера (pyserver). Без этого Docker присвоил бы случайное имя (например, funny\_gagarin). |
| --rm | Автоматически удаляет контейнер после его остановки. Без этого контейнер остался бы в системе (но в статусе Exited). |
| -d | |  |  | | --- | --- | |  | Запускает контейнер в **фоновом режиме** (detached), чтобы терминал не блокировался. | |

Часть python -m http.server -d /mnt — это **команда**, которая запускается внутри контейнера.

* python -m http.server — запускает встроенный в Python HTTP-сервер.
* -d /mnt — указывает серверу использовать /mnt как корневую директорию.

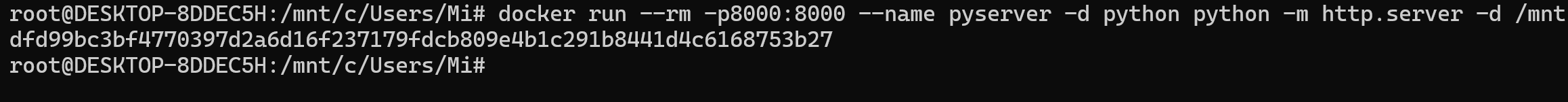


Рисунок 17

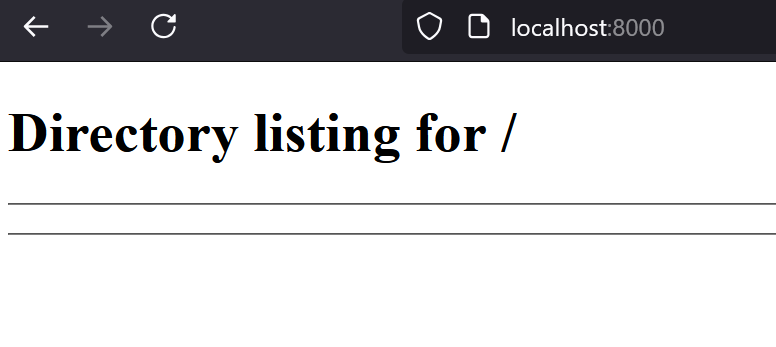


Рисунок 18

Для того, чтобы попасть в уже запущенный контейнер, существует команда docker exec -it pyserver bash - вы попадёте в оболочку bash в контейнере. Попав в контейнер, выполните команду cd mnt && echo "hello world" > hi.txt, а затем выйдите из контейнера, введя команду exit или нажав комбинацию клавиш Ctrl+D.

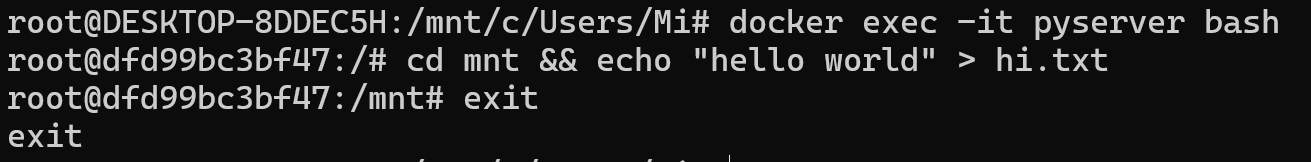


Рисунок 19

Если открыть http://0.0.0.0:8000/, там будет доступен файл hi.txt.

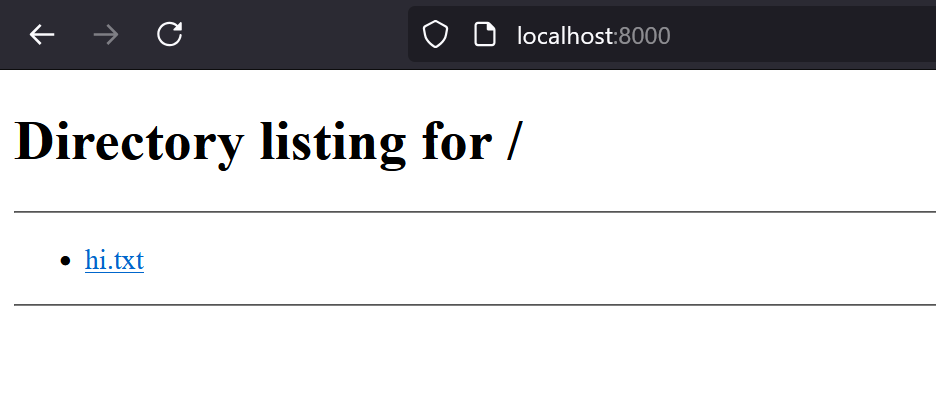


Рисунок 20

Остановим контейнер: docker stop pyserver, а затем снова запустим: docker run -p8000:8000 --name pyserver --rm -d python python -m http.server -d /mnt.

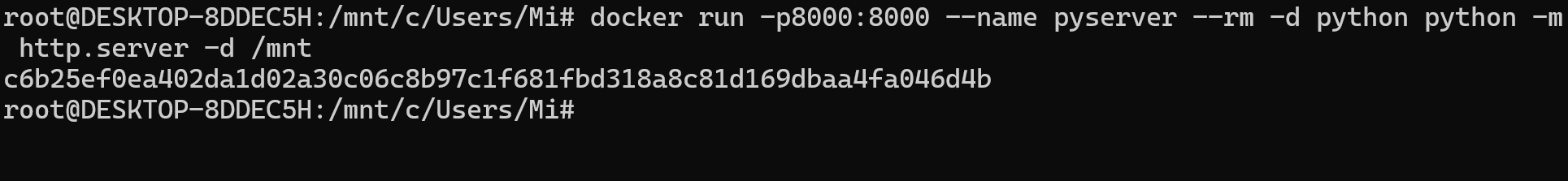


Рисунок 21

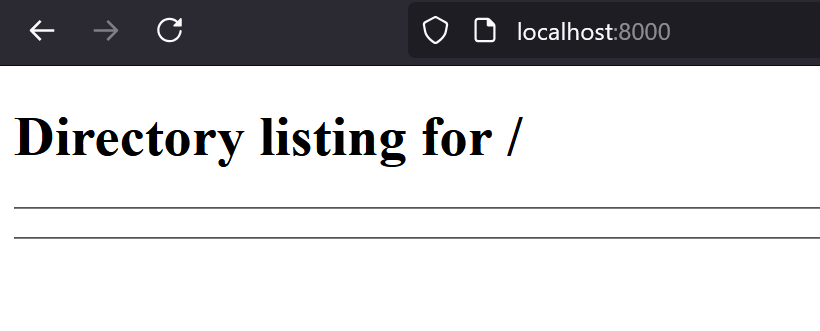


Рисунок 22

Как мы видим, файл hi.txt пропал – это неудивительно, ведь мы запустили другой контейнер, а старый был удалён после завершения работы (флаг --rm). Остановим контейнер: docker stop pyserver.

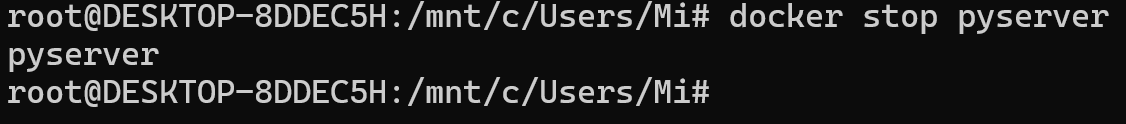


Рисунок 23

Для того, чтобы не терялись какие-то данные (например, если запущен контейнер с СУБД, то чтобы не терялись данные из неё) существует механизм монтирования.

**5.1 Тома**

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

Попробуйте снова создать контейнер, но уже с примонтированным томом:

docker run - p8000 :8000 - - rm - - name pyserver - d \ - v $ ( pwd )/ myfiles :/ mnt python python - m http . server - d / mnt

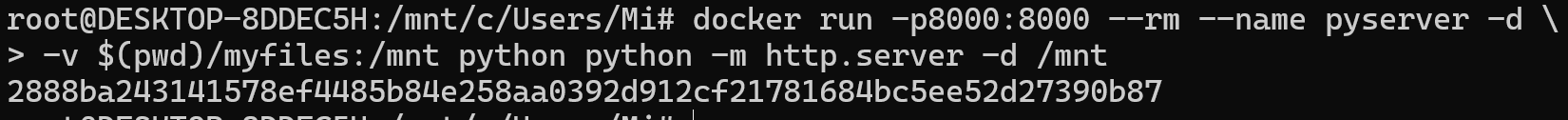


Рисунок 24

Затем, если создать файл (выполнить docker exec -it pyserver bash и внутри контейнера выполнить cd mnt && echo "hello world" > hi.txt), то даже после удаления контейнера данные в этом томе будут сохранены.

Чтобы узнать где хранятся данные, выполните команду docker inspect -f "{{json .Mounts }}" pyserver, в поле Source будет храниться путь до тома нахостовой машине.



Рисунок 25

Для управления томами существует команда docker volume, ознакомиться с которой предлагается самостоятельно.

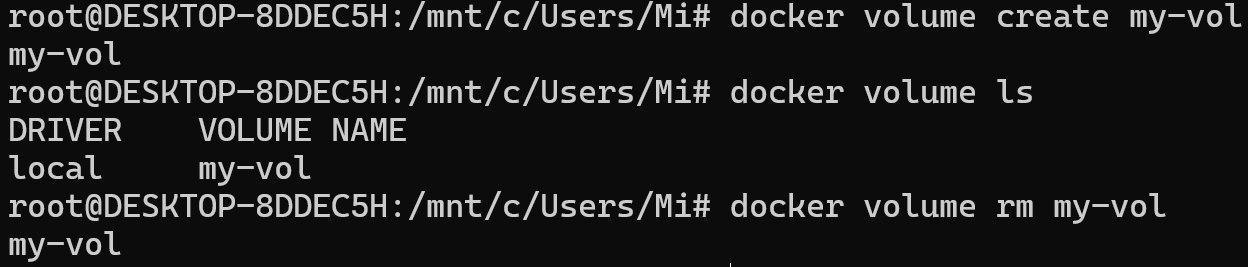


Рисунок 26

**5.2 Монтирование директорий и файлов**

Сперва, остановите контейнер, созданный на предыдущем шаге: docker stop pyserver.

Иногда требуется пробросить в контейнер конфигурационный файл или отдельную директорию. Для этого используется монтирование директорий и файлов.

Создадим директорию и файлы, которые будем монтировать. Часть из них нам понадобится дальше: создайте директорию: mkdir myfiles, в ней создайте файл host.txt: touch myfiles/host.txt

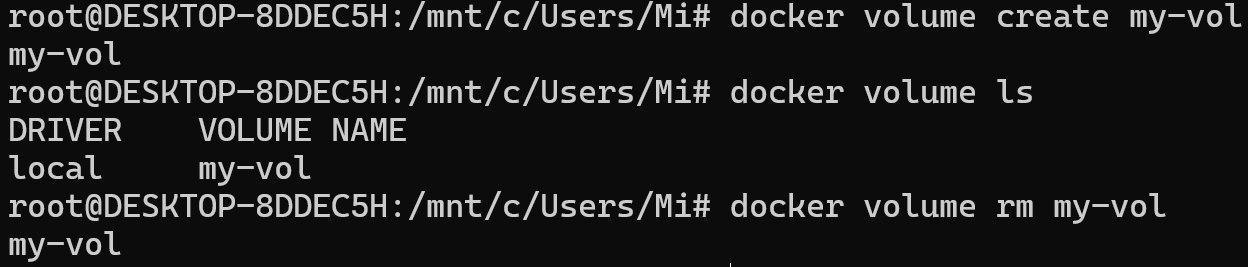


Рисунок 27

Запустите контейнер: docker run -p8000:8000 --rm --name pyserver -d -v $(pwd)/myfiles:/mnt python \ python -m http.server -d /mnt

Команда pwd - выведет текущую директорию, например: /home/user/dome-directory, в итоге получился абсолютный путь до файла: /home/user/dome-directory/myfiles. Обратный слеш (\) перед переводом строки экранирует символ перевода строки и позволяет написать одну команду в несколько строк.

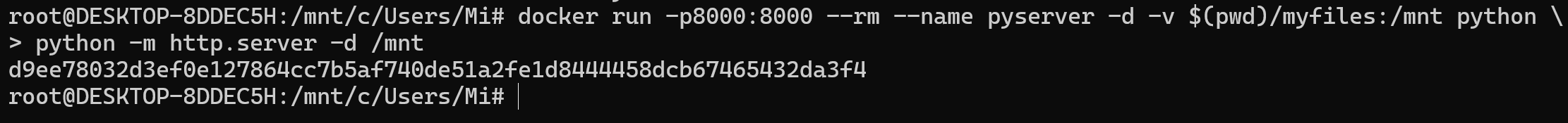


Рисунок 28

Затем, зайдите в контейнер: docker exec -it pyserver bash, перейдите в директорию /mnt командой cd /mnt. Если вывести список файлов командой ls, то там будет файл host.txt, примонтированный вместе с директорией myfiles



Рисунок 29

Создайте файл echo "hello world" > hi.txt, а затем выйдите из контейнера: exit. Теперь на хостовой машине в директории myfiles/ появится файл hi.txt. Проверить можно командой ls myfiles.

Остановите контейнер: docker stop pyserver.

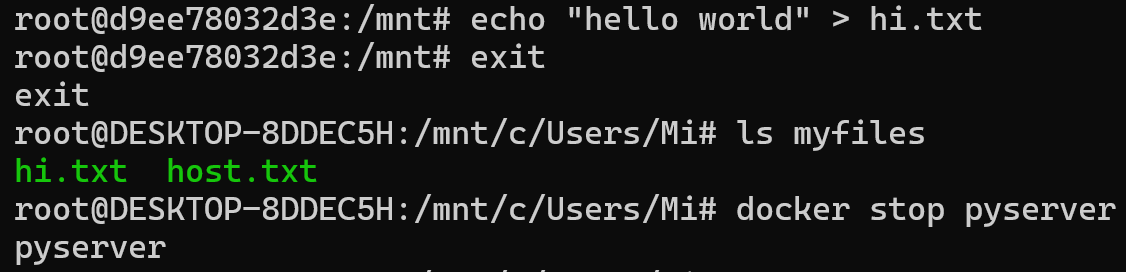


Рисунок 30

Для того, чтобы примонтировать один файл, нужно указать ключ -v, например: -v $(pwd)/myfiles/host.txt:/mnt/new-name-of-host.txt – файлу в контейнере присвоится другое имя: new-name-of-host.txt.

*Если на Windows возникают ошибки при монтировании, убедитесь, что вы используете bash, а не cmd.exe*.

**6 Переменные окружения**

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

Проверьте, выведя все переменные окружения, определённые в контейнере с помощью утилиты env: docker run -it --rm -e MIREA="ONE LOVE" ubuntu env. Среди списка переменных будет и MIREA

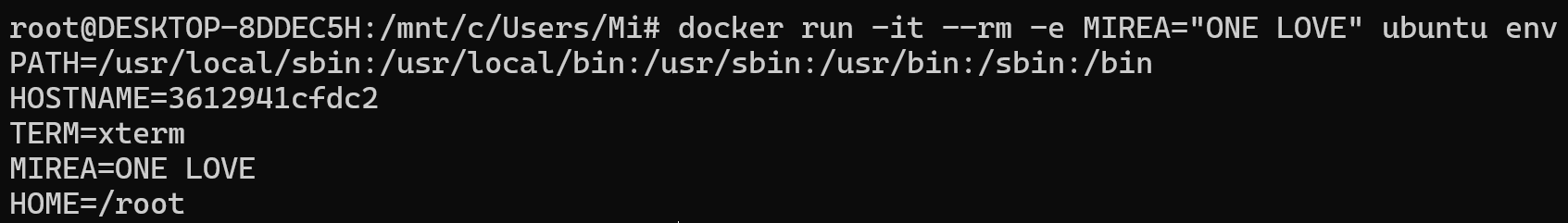


Рисунок 31

**7 Dockerfile**

Соберите образ, в который будут установлены дополнительные пакеты, примонтируйте директорию и установите команду запуска. Для этого создаётся файл Dockerfile (без расширения).

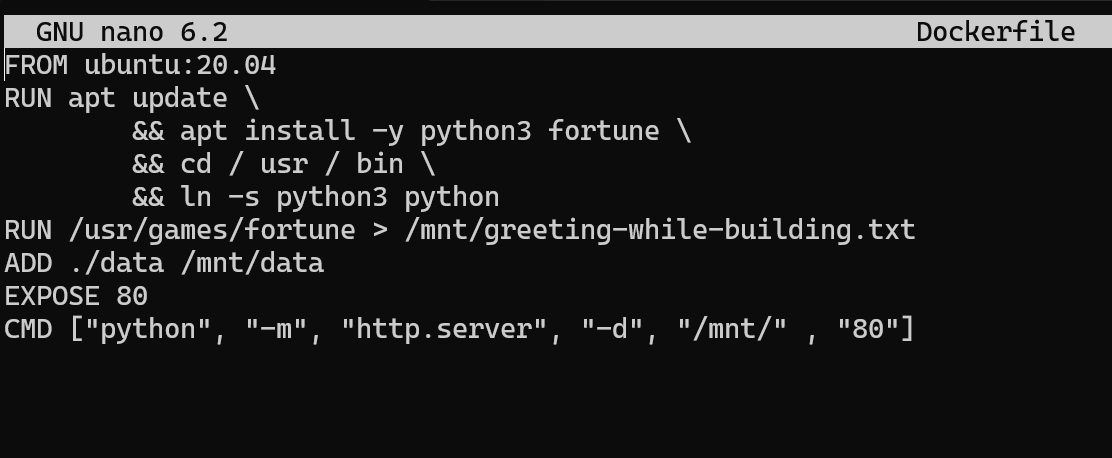


Рисунок 32

Соберите образ с тегом mycoolimage с помощью команды docker build -t mycoolimage . Точка в конце указывает на текущую директорию, где лежит Dockerfile.

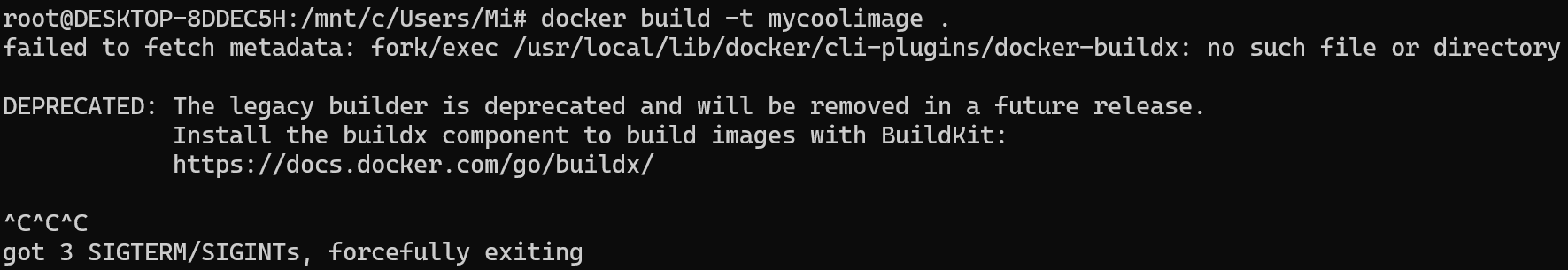


Рисунок 33 – Ошибка из-за использования build

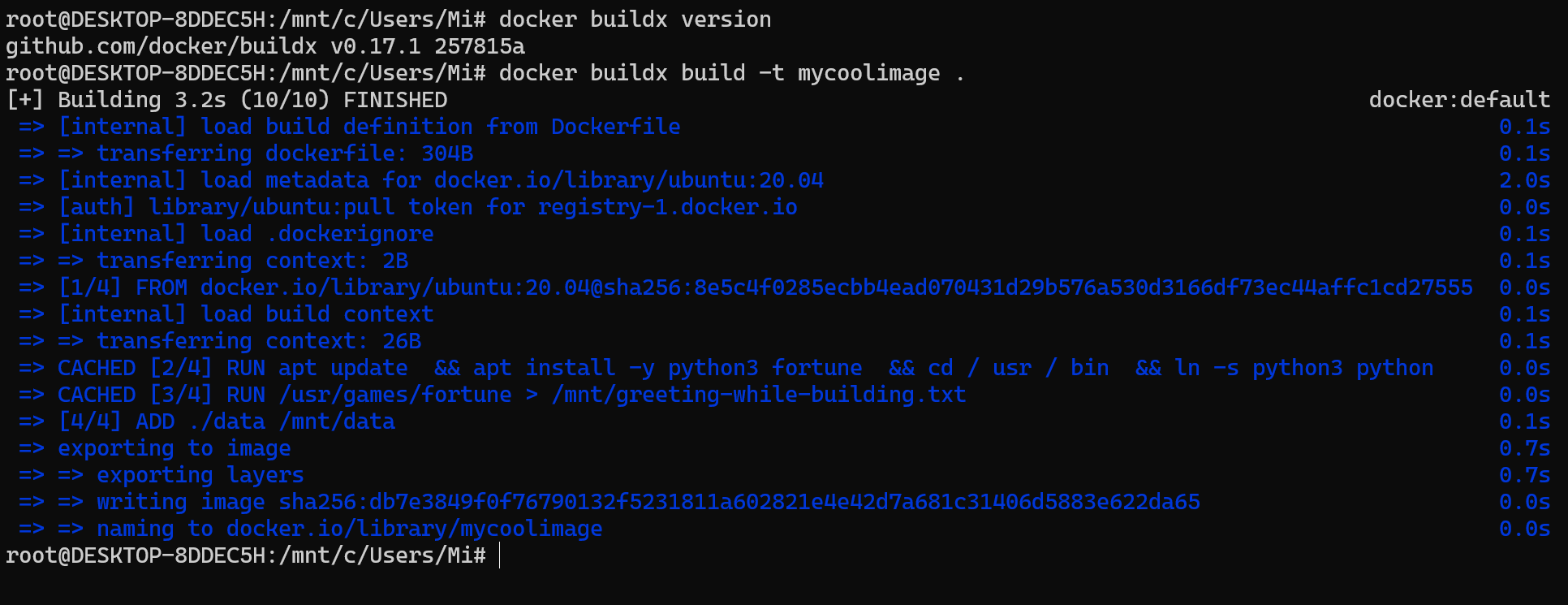


Рисунок 34 – Использование buildx

Запуск производится командой docker run --rm -it -p8099:80 mycoolimage, где порт 8099 – порт на хостовой машине.

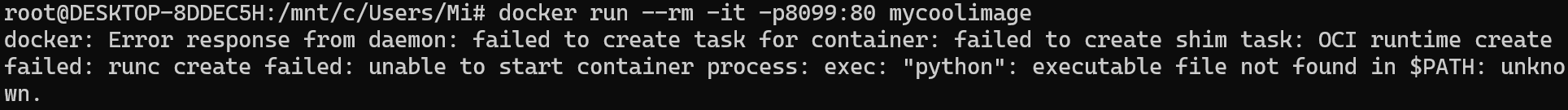


Рисунок 35

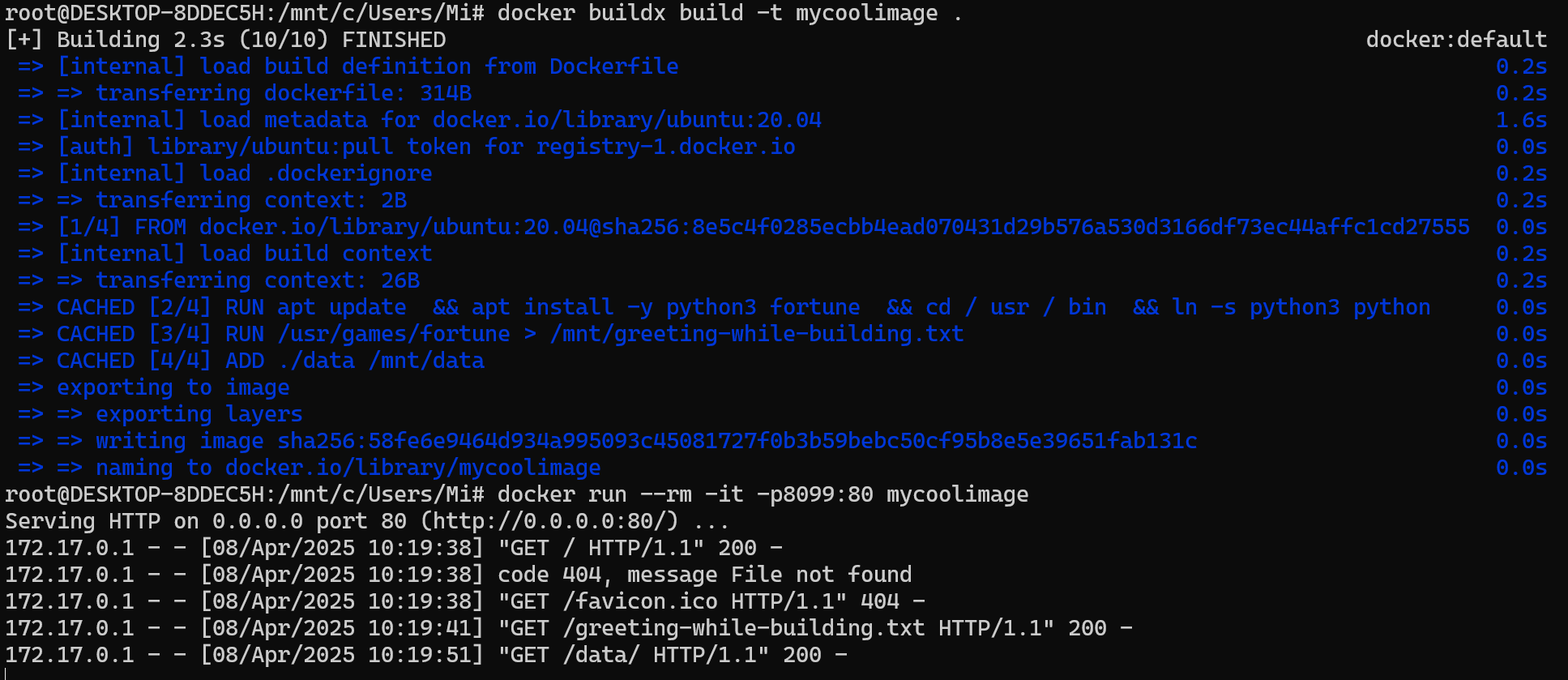


Рисунок 36

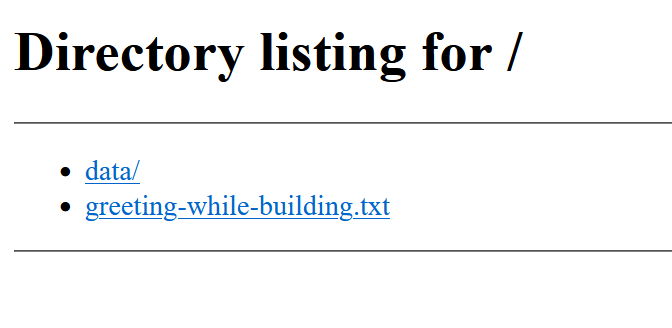


Рисунок 37

**8 Индивидуальное задание**

Написать Dockerfile, собрать образ, запустить контейнер (и записать команду для его запуска).

Для монтирования создайте директорию data и в ней файл student.txt, содержащий ФИО, название группы и номер варианта.

Для установки пакетов использовать команду apt install -y название-пакета. В качестве примера можно использовать Dockerfile из раздела 7.

• необходимо использовать базовый образ ubuntu:20.10

• примонтировать файл data/student.txt как /mnt/files/student.txt в контейнере.

Запустить веб-сервер, отображающий содержимое /mnt/files, в хостовой системе должен открываться на порту (8800 + номер варианта).

Установить пакет, согласно варианту: 8. postgresql-client

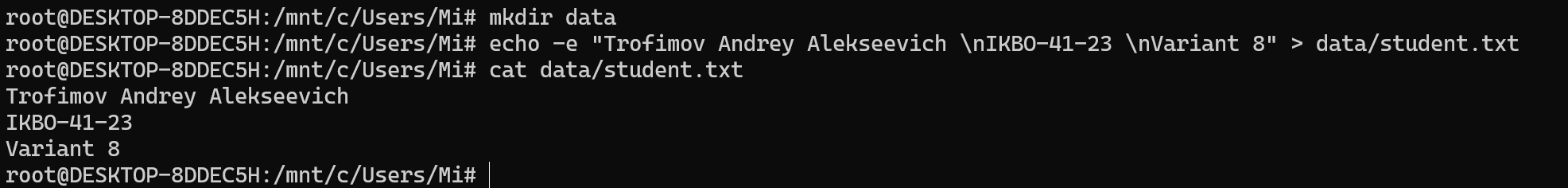


Рисунок 38

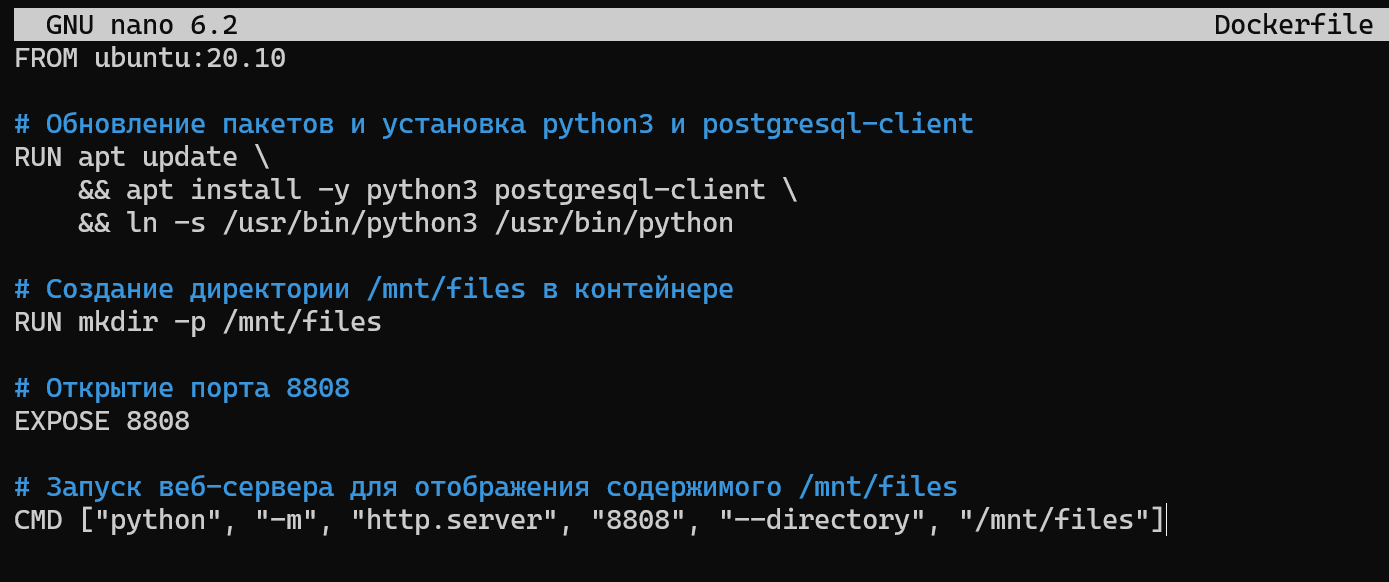


Рисунок 39

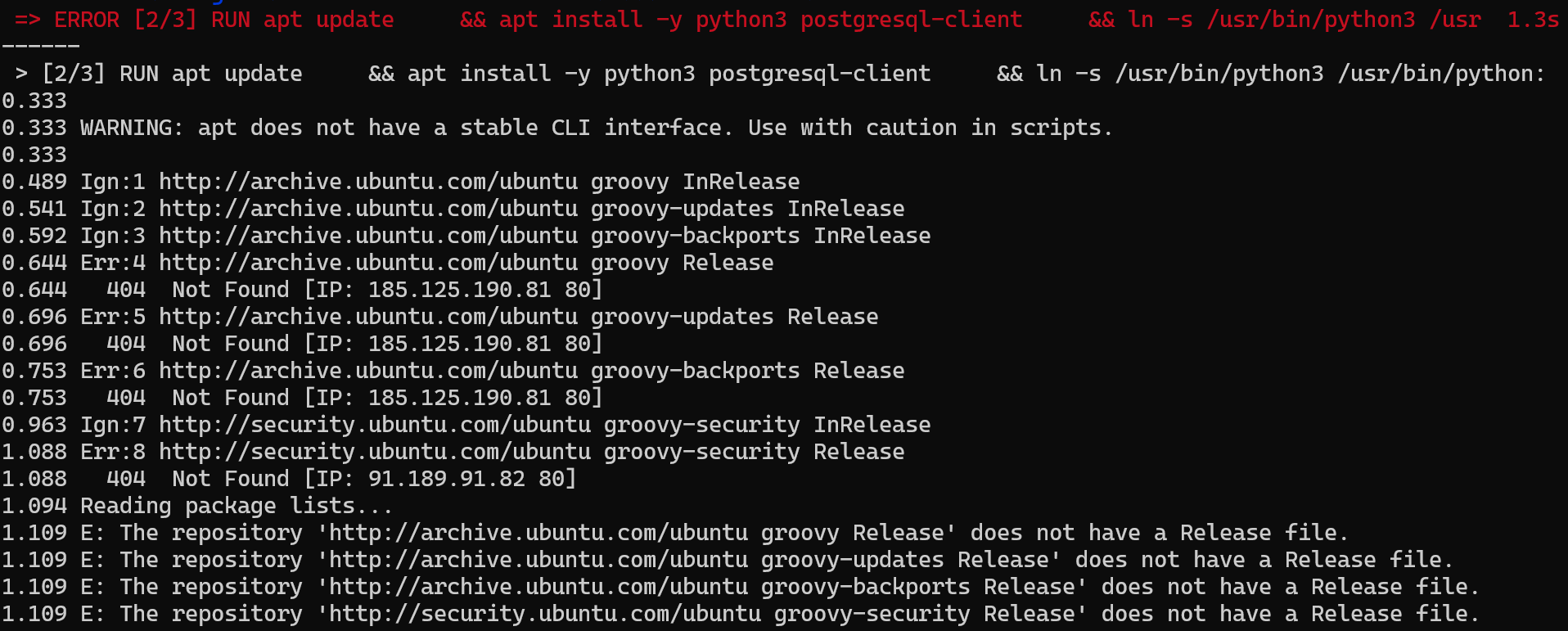


Рисунок 40 – Ошибка из-за устаревшей версии ubuntu

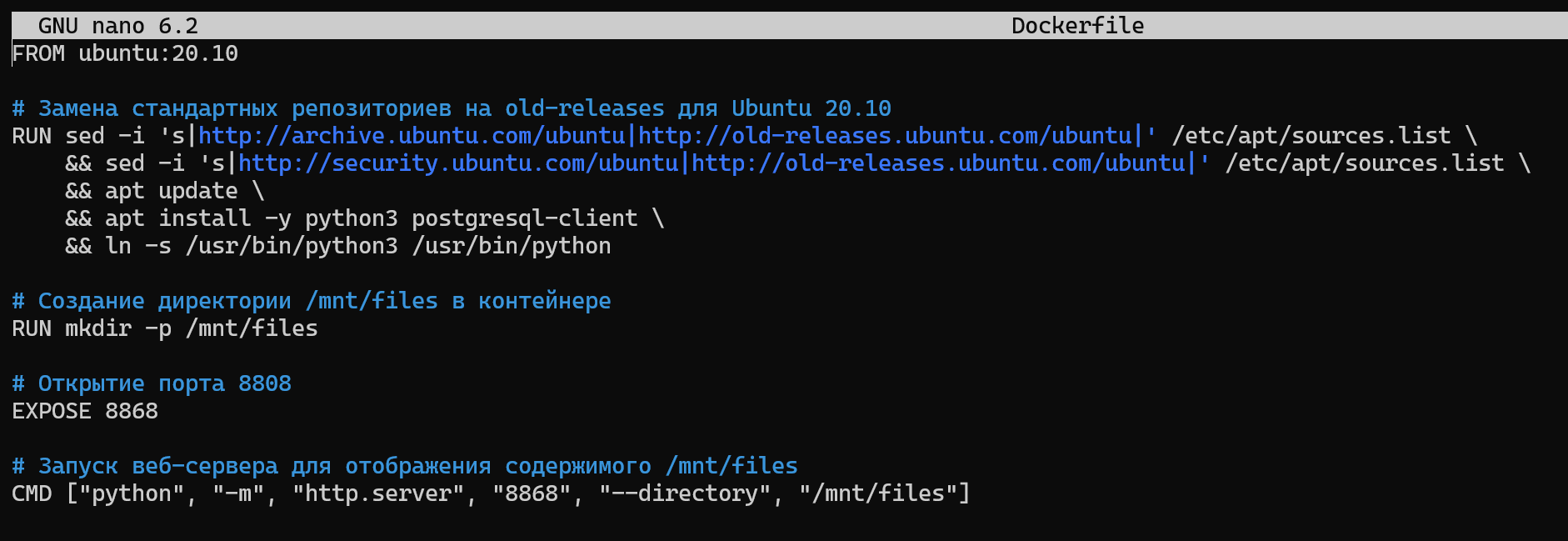


Рисунок 41

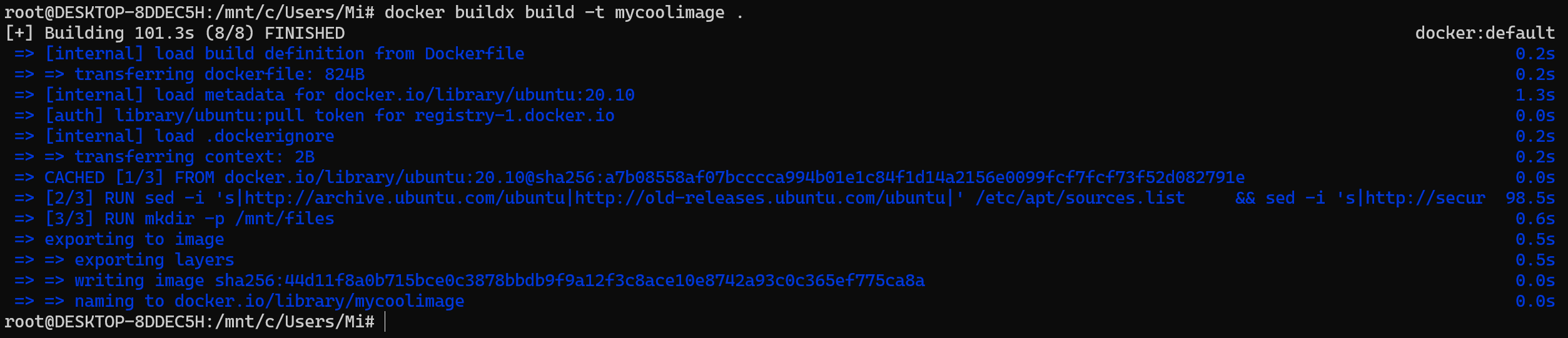


Рисунок 42

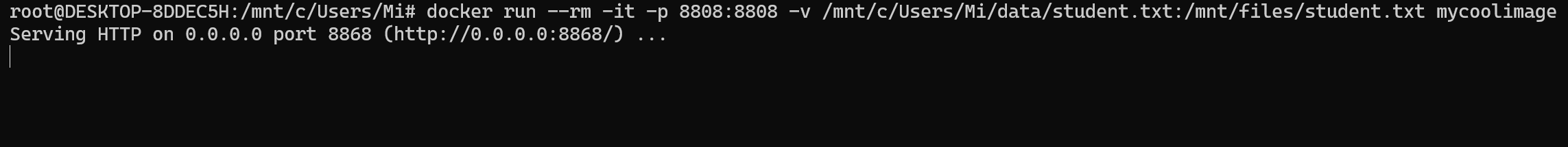


Рисунок 43

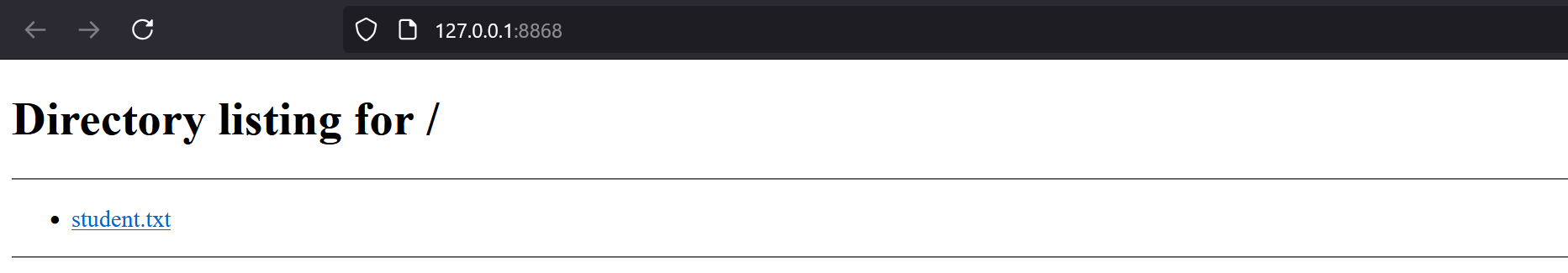


Рисунок 44

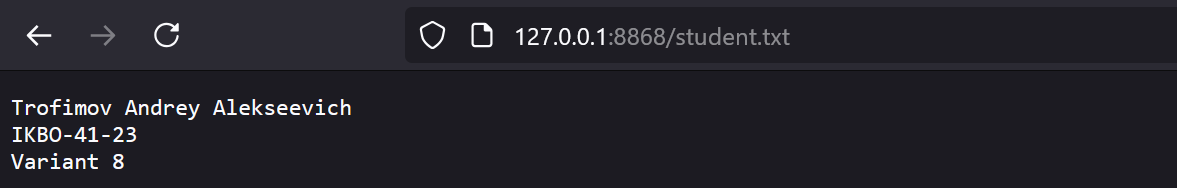


Рисунок 45

# **ВЫВОД**

В ходе работы было успешно продемонстрировано применение технологии Docker для создания, управления и развертывания контейнеров. Использование Docker позволило обеспечить независимость приложения от окружения, упростить процесс настройки и повысить эффективность разработки. Полученные результаты подтверждают, что Docker является мощным инструментом для автоматизации развертывания и управления приложениями, что делает его востребованным в современной разработке программного обеспечения.

**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. Турнецкая Е. Л., Аграновский А. В.Программная инженерия. Интеграционный подход к разработке» (Турнецкая, Е. Л. Программная инженерия. Интеграционный подход к разработке / Е. Л. Турнецкая, А. В. Аграновский. — Санкт-Петербург : Лань, 2023.
2. «Зубкова Т.М. Технология разработки программного обеспечения» (Зубкова, Т. М. Технология разработки программного обеспечения : учебное пособие / Т. М. Зубкова. — Санкт-Петербург : Лань, 2022.