

Оглавление

[1. Постановка задачи 3](#_Toc131591311)

[1.1. Цель работы 3](#_Toc131591312)

[1.2. Задание (Вариант 1) 3](#_Toc131591313)

[2. Ход работы 3](#_Toc131591314)

[2.1. Образы 3](#_Toc131591315)

[2.2. Изоляция 4](#_Toc131591316)

[2.3. Работа с портами 5](#_Toc131591317)

[2.4. Именованные контейнеры, остановка и удаление 7](#_Toc131591318)

[2.5. Постоянное хранение данных 8](#_Toc131591319)

[2.5.1. Тома 9](#_Toc131591320)

[2.5.2. Монтирование директорий и файлов 10](#_Toc131591321)

[2.6. Переменные окружения 11](#_Toc131591322)

[2.7. Dockerfile 12](#_Toc131591323)

[2.8. Индивидуальные задания 12](#_Toc131591324)

[3. Вывод 13](#_Toc131591325)

1. Постановка задачи
   1. Цель работы

Знакомство с контейнеризатором прилолежний Docker. Возможности Docker.

* 1. Задание (Вариант 1)

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.

**ВАРИАНТ 1:**

**Установить пакет, согласно варианту: 1. cowsay**

1. Ход работы
   1. Образы

Посмотрите на имеющиеся образы: docker images.

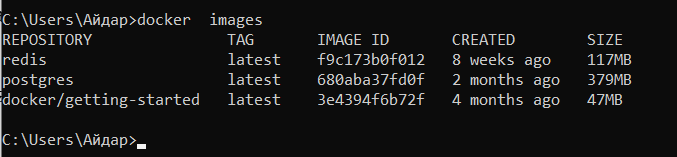


Рисунок 1 – Образы

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.

Посмотрите на имеющиеся образы ещё раз: docker images — должны появиться новые загруженные образы.

Посмотрите список контейнеров, выполнив команду: docker ps -a.

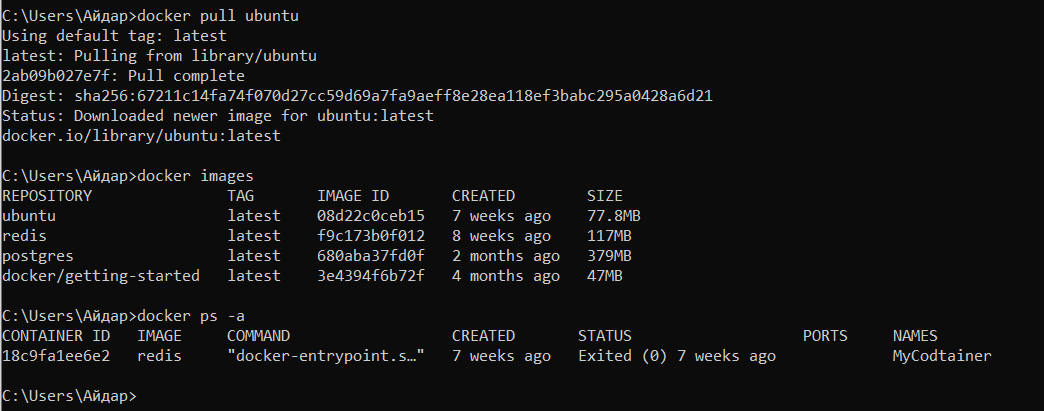


Рисунок 2 – Загрузка образа

* 1. Изоляция

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

Вопрос: одинаковый ли результат получился при разных запусках?

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

Вопрос: Одинаковый ли результат получился при разных запусках?

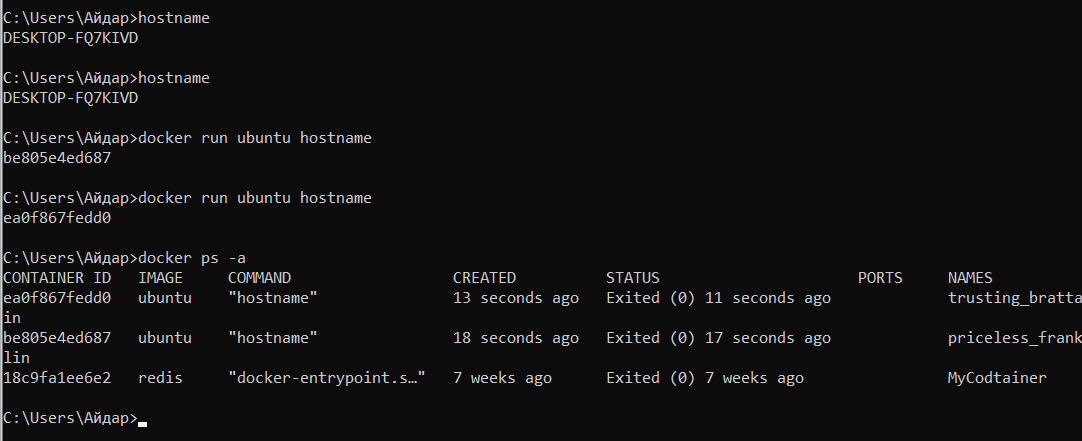


Рисунок 3 – Информация о хостовой системе

*Ответ на вопросы*: для моего ПК – одинаковый, для контейнеров в докере – каждый раз разный.

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.

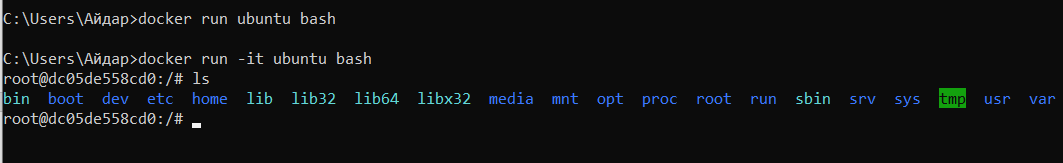


Рисунок 4 – Запуск контейнера с -it и без

* 1. Работа с портами

Для начала, загрузите образ python командой docker pull python.

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

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.

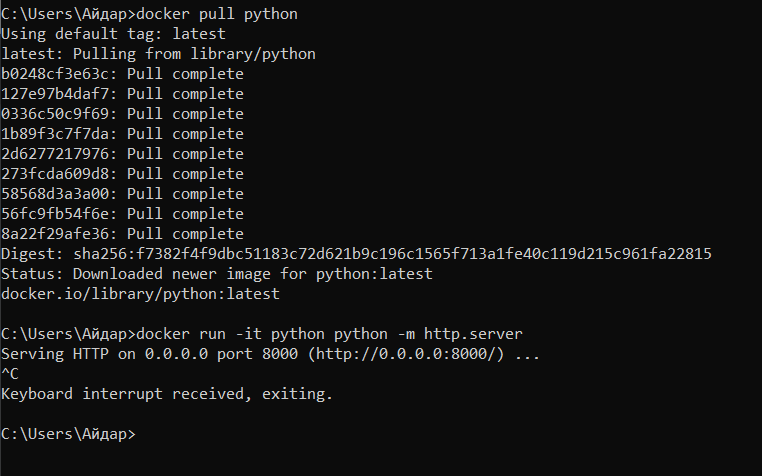


Рисунок 5 – Работа с образом python

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/> открывается содержимое корневой директории в контейнере.

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.

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

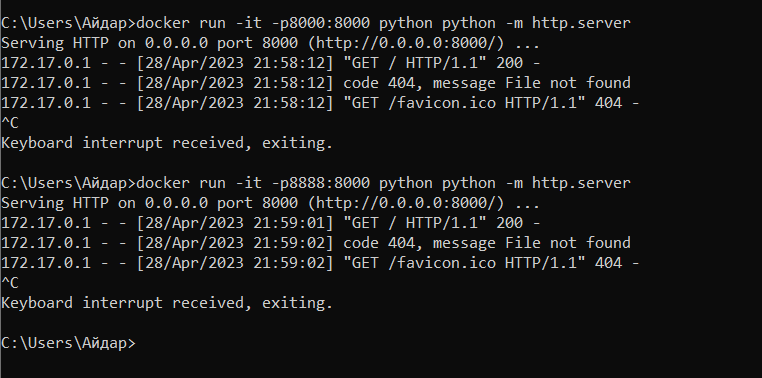


Рисунок 6 – Проброс порта



Рисунок 7 – Содержимое по адресу

* 1. Именованные контейнеры, остановка и удаление

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

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

Убедитесь, что контейнер всё ещё запущен: docker ps. Для просмотра логов контейнера, воспользуйтесь командой docker logs pyserver. Для того, чтобы остановить выполнение контейнера, существует команда docker stop pyserver.

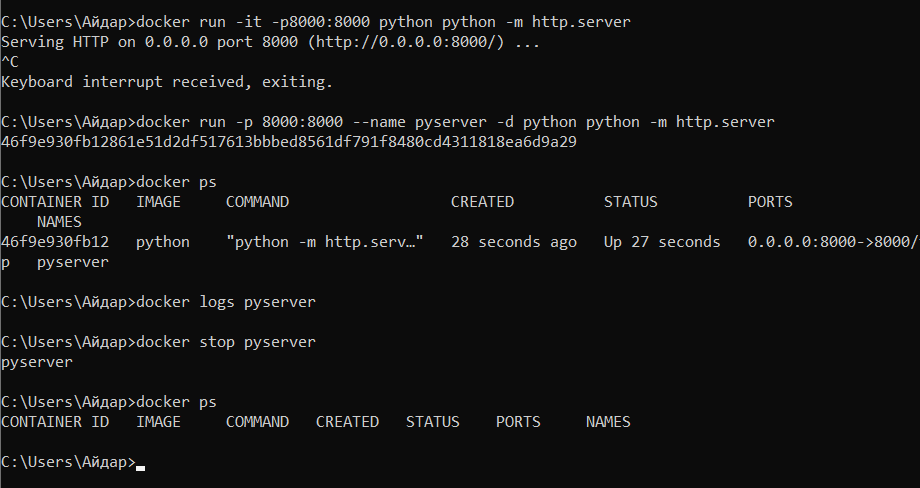


Рисунок 8 – Запуск контейнера в фоне и с именем

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.

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

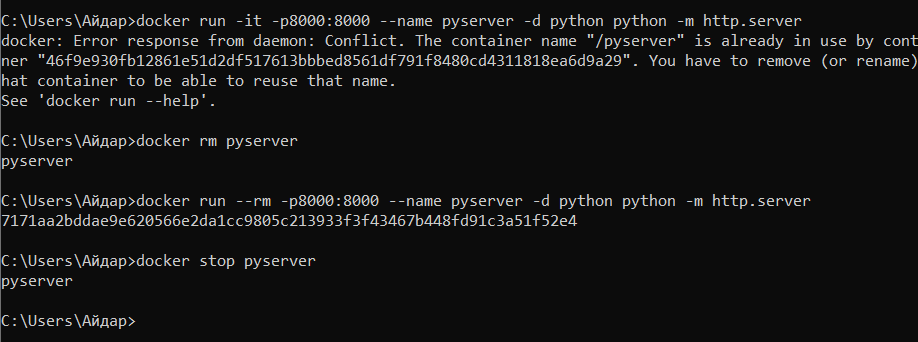
docker run --rm -p8000:8000 --name pyserver -d python python -m http.server 

Рисунок 9 – Запуск контейнера с флагом --rm

* 1. Постоянное хранение данных

Запустите контейнер, в котором веб-сервер будет отдавать содержимое директории /mnt.

docker exec -it pyserver bash

cd mnt && echo "hello world" > hi.txt

Выполните команды выше, а затем выйдите из контейнера, введя команду exit.

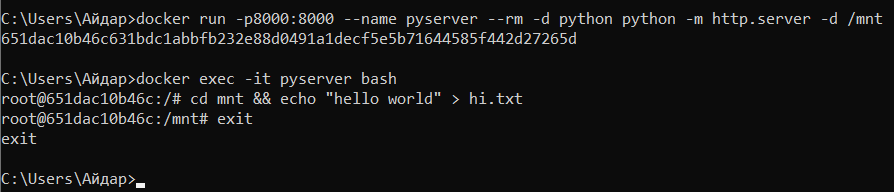


Рисунок 10 – Запуск контейнера и переход в оболочку bash

Остановим контейнер: docker stop pyserver, а затем снова запустим. Видно, что файл пропал, ведь прошлый контейнер удалился (флаг --rm).

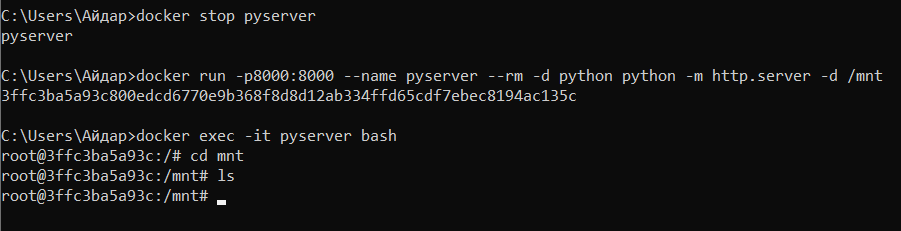


Рисунок 11 – Повторный запуск контейнера и переход в оболочку bash

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

*Ответ на вопрос:*

* p8000:8000 – пробрасывает порт 8000 из контейнера в порт 8000 на хосте, так веб-сервер, запущенный в контейнере, может быть доступным по адресу <http://localhost:8000>;
* --name pyserver – задает имя контейнеру, конкретно – pyserver;
* --rm – флаг для автоматического удаления контейнера после его остановки;
* -d – запускает контейнер в фоновом режиме;
* python python -m http.server -d /mnt – команда, которая выполнится в контейнере сразу после его запуска: запустит веб-сервер на порту 8000 с корневой директорией /mnt.
  + 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.

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.

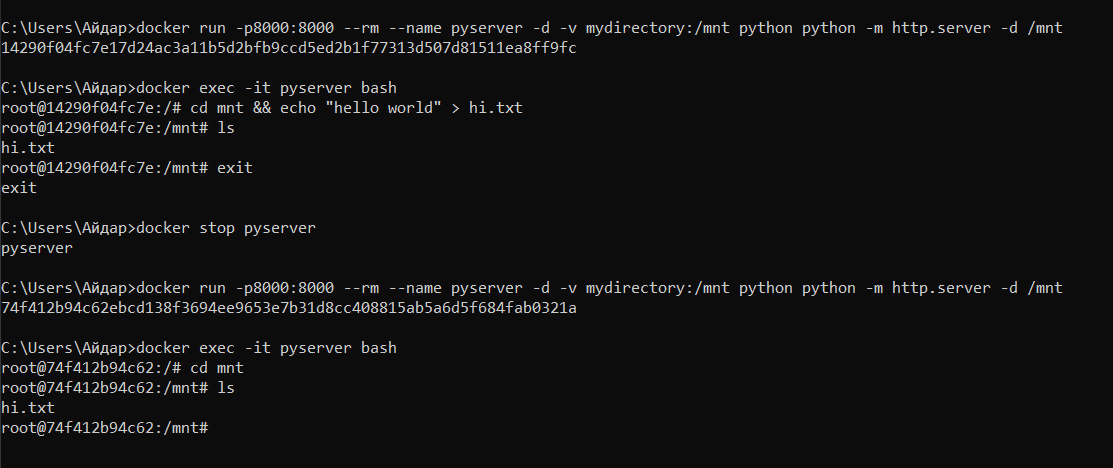


Рисунок 12 – Данные сохранились

Чтобы узнать, где хранятся данные, выполните команду

docker inspect -f "{{json .Mounts }}" pyserver, в поле Source будет храниться путь до тома на хостовой машине.

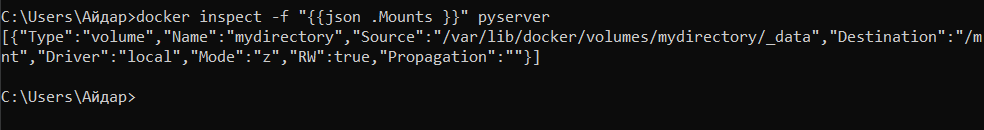


Рисунок 13 – Местоположение данных

* + 1. Монтирование директорий и файлов

Иногда требуется пробросить в контейнер конфигурационный файл или отдельную директорию. Для этого используется монтирование директорий и файлов. Создадим директорию и файлы, которые будем монтировать. Затем запустим контейнер и смонтируем созданный файл. Можно также создать файл в контейнере, даже после остановки и удаления контейнера – он будет находиться на хосте.

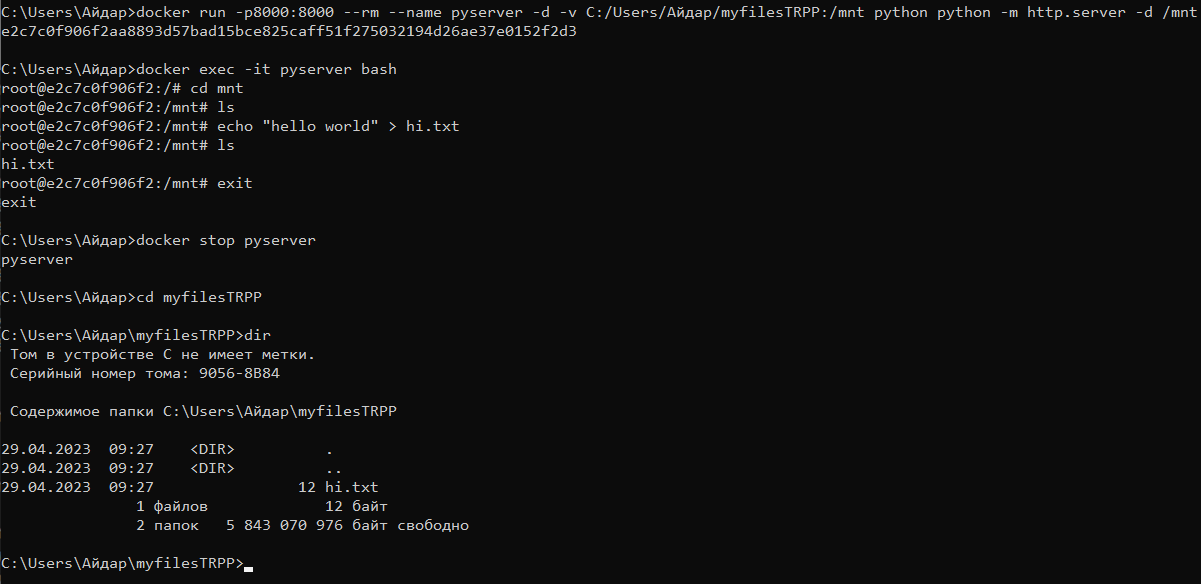


Рисунок 14 – Монтирование файла

* 1. Переменные окружения

Для передачи переменных окружения внутрь контейнера используется ключ -e. Например, чтобы передать в контейнер переменную окружения MIREA со 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.

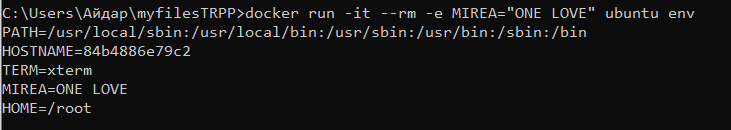


Рисунок 15 – Переменные окружения

* 1. Dockerfile

Соберите образ, в который будут установлены дополнительные пакеты, примонтируйте директорию и установите команду запуска. Для этого создаётся файл Dockerfile (без расширения). Соберите образ с тегом mycoolimage с помощью команды docker build -t mycoolimage . Точка в конце указывает на текущую директорию, где лежит Dockerfile.

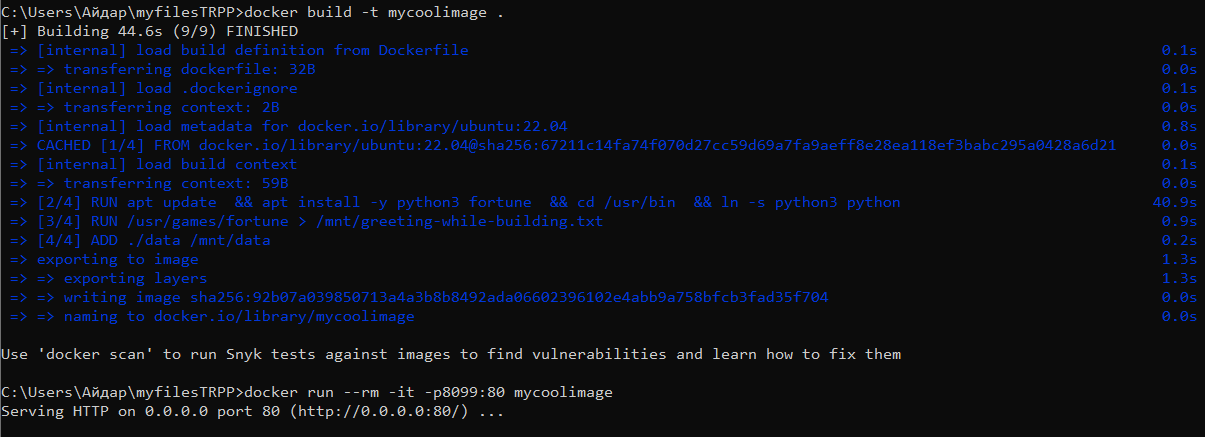


Рисунок 16 – Сборка образа

* 1. Индивидуальные задания

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.

Запустить веб-сервер, отображающий содержимое /mnt/files, в хостовой системе должен открываться на порту **8801**.

Установить пакет, согласно **варианту 1**:  **cowsay**

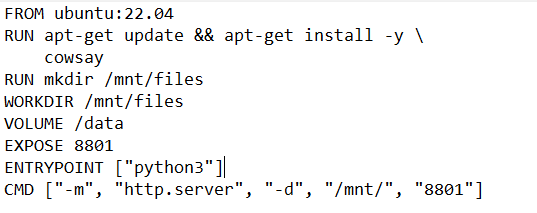


Рисунок 17 – Содержимое файла Dockerfile

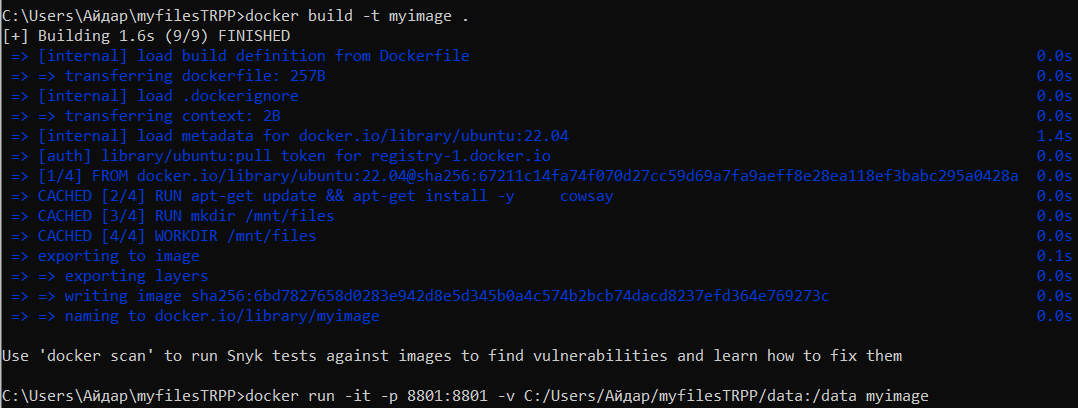


Рисунок 18 – Сборка образа по индивидуальному заданию

1. Вывод

В ходе выполнения данной практической работы были изучены основные команды Docker, а также были получены практические навыки по работе с контейнеризатором приложений Docker.