|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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

«**Технология разработки программных приложений**»

**Тема: «Docker»**

|  |  |
| --- | --- |
| **Выполнил:**  Студент группыИКБО-36-22 | Утенков Юрий Юрьевич |
| **Проверил:** | ассистент Петрова А.А. |

МОСКВА 2024 г.

**О Г Л А В Л Е Н И Е**

[Практическая работа № 4 3](#_Toc161441263)

[Цель работы 3](#_Toc161441264)

[Выполнение работы 3](#_Toc161441265)

[Образы 3](#_Toc161441266)

[Изоляция 4](#_Toc161441267)

[Работа с портами 5](#_Toc161441268)

[Именованные контейнеры, остановка и удаление 7](#_Toc161441269)

[Постоянное хранение данных 8](#_Toc161441270)

[Тома 10](#_Toc161441271)

[Монтирование директорий и файлов 11](#_Toc161441272)

[Переменные окружения 12](#_Toc161441273)

[Dockerfile 13](#_Toc161441274)

[Индивидуальное задание 14](#_Toc161441275)

[Выводы 15](#_Toc161441276)

# Практическая работа № 4

Цель работы

Знакомство с системой контейнеризации Docker. Возможности Docker. Управление контейнерами.

Выполнение работы

**Задание:**

В данной практической работе 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.

1. Образы
2. Изоляция
3. Работа с портами
4. Именованные контейнеры, остановка и удаление
5. Постоянное хранение данных
   1. Тома
   2. Монтирование директорий и файлов
6. Переменные окружения
7. Dockerfile
8. Индивидуальное задание

**Ход выполнения работы:**

Образы

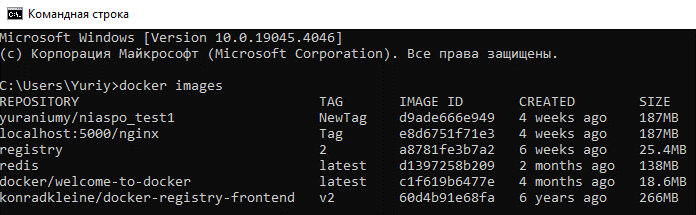


Рисунок 1 − Выполнение команды docker images

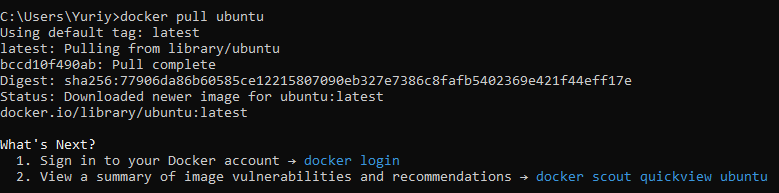


Рисунок 2 − Загрузка образа ubuntu

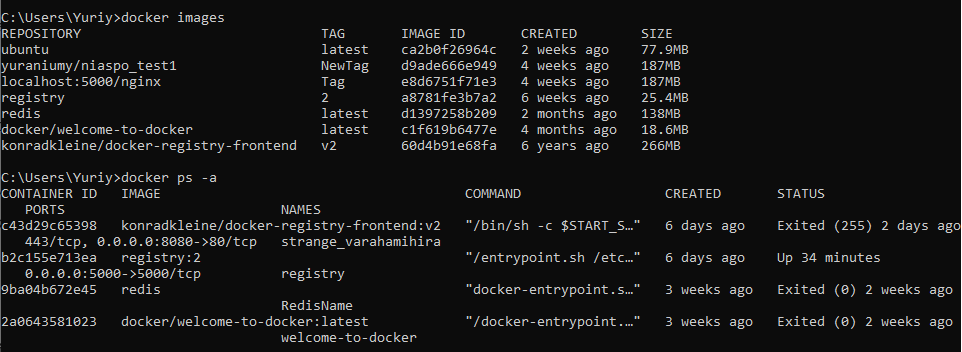


Рисунок 3 − Выполнение команды docker images и docker ps -a

Команда docker ps с опцией -a выводит все контейнеры, которые когда-либо были запущены на компьютере.

Изоляция

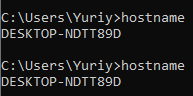


Рисунок 4 − Выполнение команды hostname

Вопрос: одинаковый ли результат получился при разных запусках?

Ответ: Результат одинаковый, т. к. в качестве хоста выступает непосредственно локальная машина.

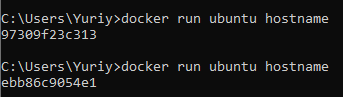


Рисунок 5 − Запуск контейнера на основе образа ubuntu

Вопрос: Одинаковый ли результат получился при разных запусках?

Ответ: нет, разный, т. к. при выполнении команды docker run, docker создаёт новый контейнер на основе образа. По умолчанию, docker устанавливает имя хоста контейнера как короткую версию идентификатора контейнера, который генерируется случайным образом, поэтому будет возвращён укороченный идентификатор контейнера.

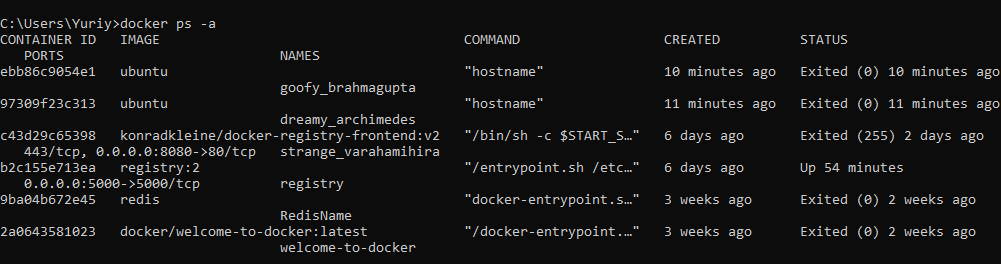


Рисунок 6 − Выполнение команды docker ps -a



Рисунок 7 − Запуск bash из контейнера на основе образа ubuntu

Контейнер представляет собой изолированную часть, в которой можно безопасно работать, или что-либо делать, т. к. контейнер виртуализирует ОС, и, если что-то пойдет не так, можно просто перезапустить контейнер без вреда для основной ОС.

Флаг -it позволяет интерактивно в данном терминале сразу запустить bash.

Работа с портами

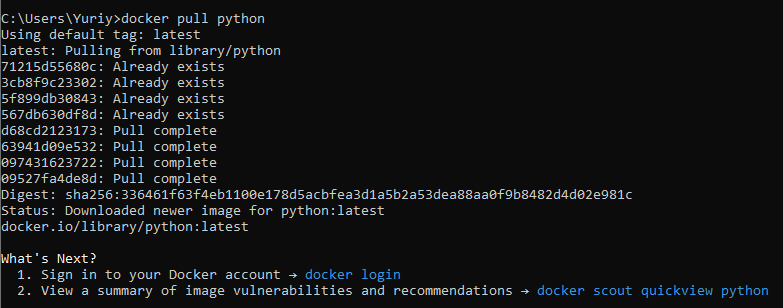


Рисунок 8 − Загрузка образа python из реестра docker

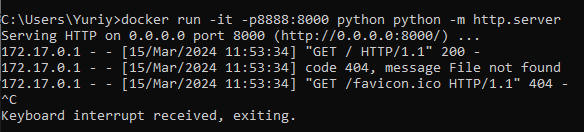


Рисунок 9 − Запуск контейнера с пробросом портов

Флаг -p позволяет пробрасывать порты между локальной машиной и контейнером (сервером) для того, чтобы можно было подключиться к серверу, флаг -p позволяет это сделать.

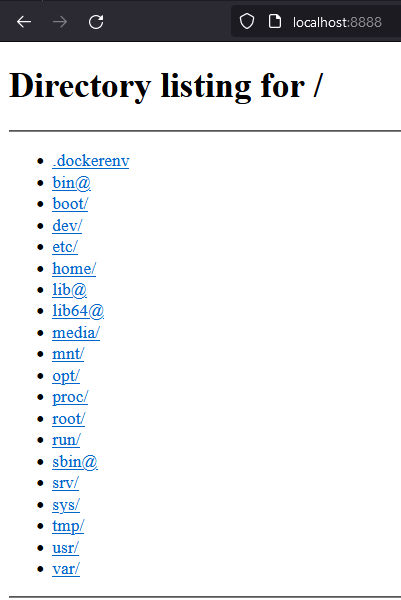


Рисунок 10 − Содержимое корневой директории в контейнере

Именованные контейнеры, остановка и удаление

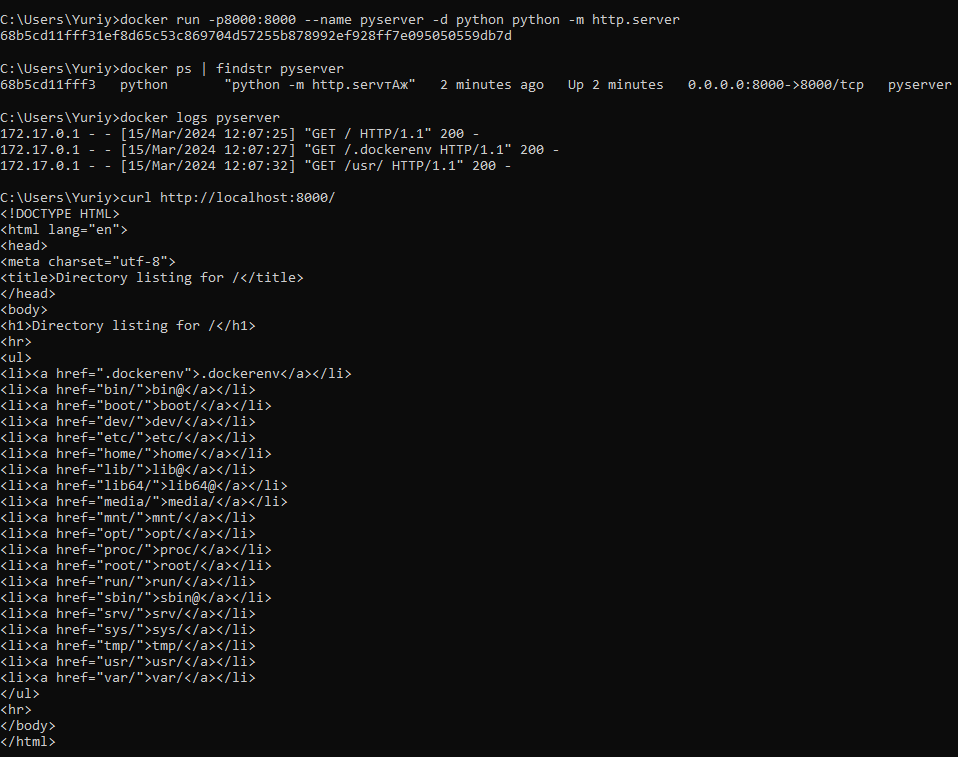


Рисунок 11 − Запуск контейнера в отдельном процессе и работа с ним

На рисунке 11 происходит запуск контейнера в отдельном процессе, это значит, что контейнер запустился и можно работать в другой среде. После, с помощью команды docker ps | findstr pyserver, которая ищет все контейнеры с именем puserver, происходит вывод информации о найденных контейнерах в консоль. Потом идёт выполнения команды docker logs pyserver, которая выводит список всех логов для данного контейнера. В конце я произвожу GET запрос на данный сервер, для отображения работоспособности.

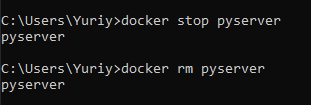


Рисунок 12 − Удаление данного контейнера

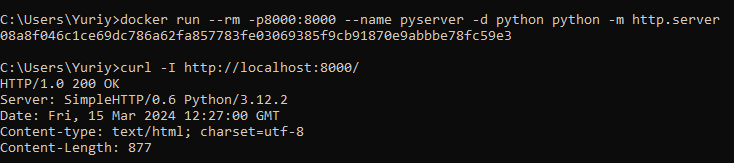


Рисунок 13 − Запуск контейнера с опциями –rm и –name

На рисунке 13 указан запуск контейнера с опциями –rm и –name. Опция –rm позволяет запускать контейнер единоразово, это удобно, если нужен запуск контейнера для каких-либо тестов, после остановки контейнера, он удалится. Опция –name позволяет задать имя для данного контейнера.

С помощью команды curl с флагом -I я вывожу краткую справку от сервера (заголовки).

Постоянное хранение данных

Запуск контейнера, в котором веб-сервер будет отдавать содержимое директории /mnt (Рисунок 14).



Рисунок 14 − Запуск контейнера

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

Ответ: флаг -p позволяет пробрасывать порты между локальной машиной и контейнером (сервером). Опция –name позволяет задать имя запускаемому контейнеру. Опция –rm позволяет удалить контейнер после завершения работы. Флаг -d позволяет запускать контейнер в отдельном процессе, т.е. в фоновом режиме. Флаг -m задаёт имя модуля. Все команды в контейнере выполнятся после его запуска.

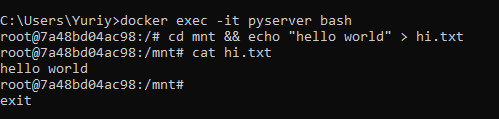


Рисунок 15 − Заход в запущенный контейнер

Команда docker exec позволяет попадать внутрь запущенного контейнера, а флаг -it позволяет запустить среду выполнения внутри контейнера непосредственно в данном терминале.

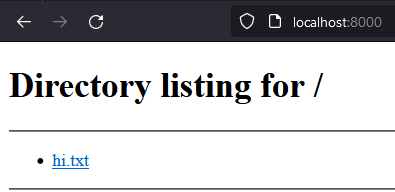


Рисунок 16 − Отображение созданного файла на сервере

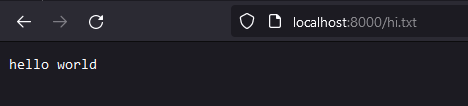


Рисунок 17 − Записанная информация в файле

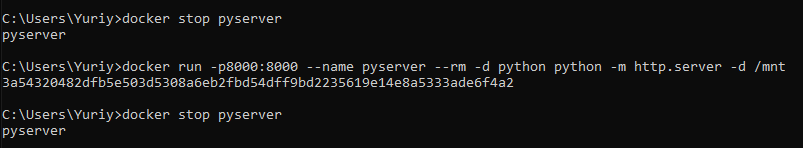


Рисунок 18 − Остановка и запуск нового контейнера с последующей остановкой

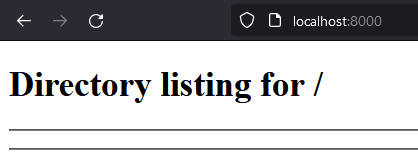


Рисунок 19 − Отображение директории в новом контейнере

Как можно заметить на рисунке 19, файл, созданный в прошлый раз, пропал, т. к. при запуске контейнера была указана опция –rm, которая удаляла контейнер при остановке его. В новом контейнере данного файла нет.

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

Тома

Тома в docker позволяют хранить данные в какой-то директории, и удобно взаимодействовать с различными контейнерами взаимно. Это решает проблему множественного создания файлов, вместо этого данные можно сохранять в томе, а не в контейнере.

Чтобы создать том во время старта контейнера, можно указать несколько флагов, один из которых -v, от слова volume. Но лучше всего указывать опцию –mount, т. к. она позволяет работать с сервисами или указывать параметры драйвера тома.

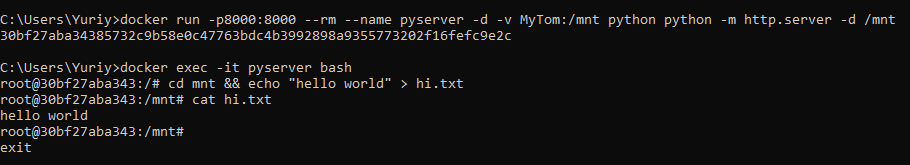


Рисунок 20 − Создание тома и запуск контейнера

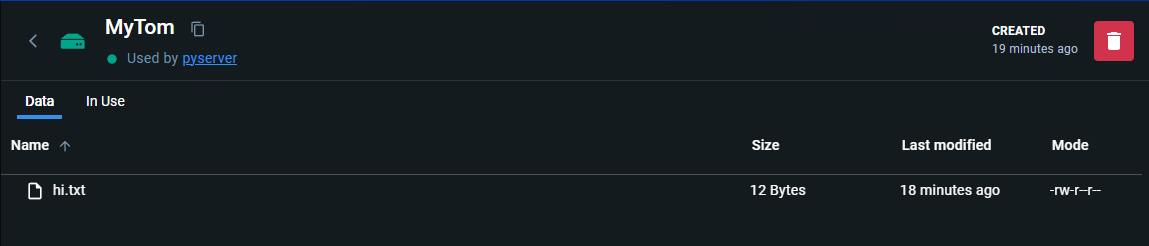


Рисунок 21 − Добавленный файл в том MyTom

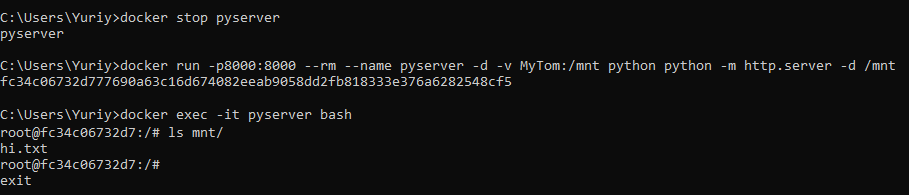


Рисунок 22 − Демонстрация работы тома

На рисунке 22 происходит удаление старого контейнера с файлом hi.txt, находящегося в томе MyTom, и после запуска нового контейнера, можно увидеть, что файл остался.

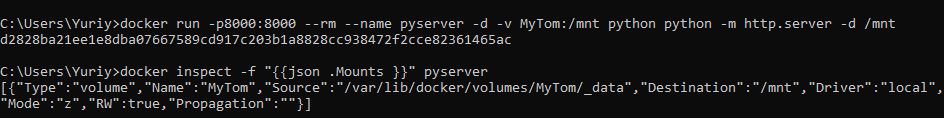


Рисунок 23 − Путь до тома

Путь до данного тома будет храниться в поле Source.

Монтирование директорий и файлов

Иногда требуется пробросить в контейнер конфигурационный файл или отдельную директорию. Для этого используется монтирование директорий и файлов. На рисунке 24 показан процесс монтированию директории и двух файлов.

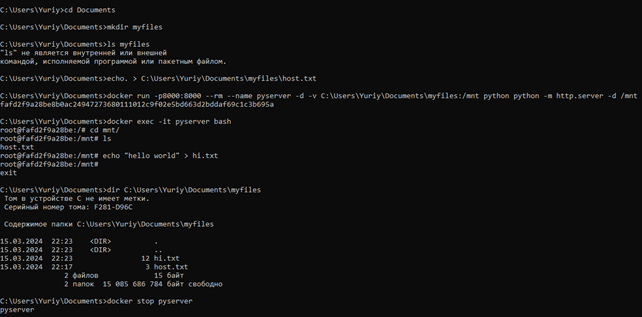


Рисунок 24 − Выполнение команд по монтированию директории и файлов

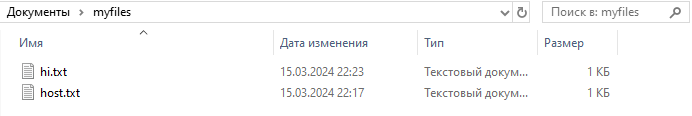


Рисунок 25 − Файлы на хостовой машине, созданные внутри контейнера

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". Можно проверить, 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 MIREA.

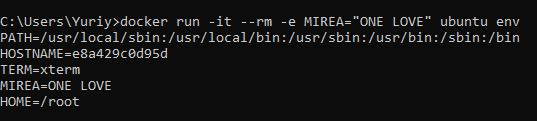


Рисунок 26 − Вывод всех переменных окружения в контейнере

Dockerfile

В Dockerfile содержатся необходимые инструкции, для сборки собственного образа, для этого достаточно указать основную информацию.

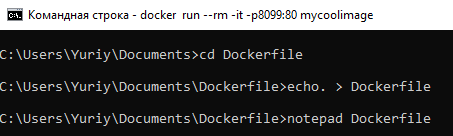


Рисунок 27 − Создание директории и файла Dockerfile

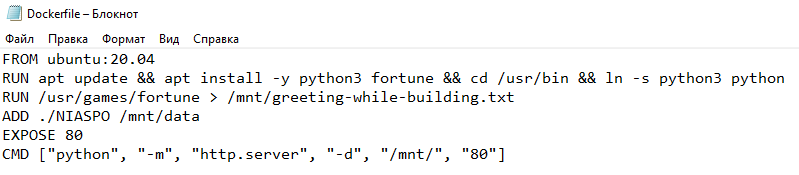


Рисунок 28 − Содержимое Dockerfile’а

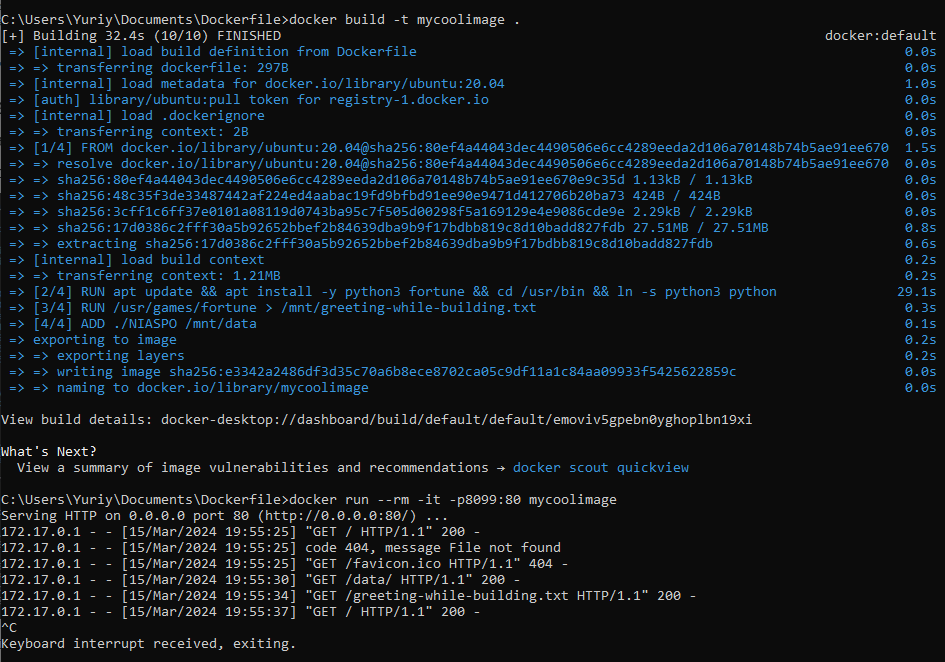


Рисунок 29 − Сборка собственного образа на основе Dockerfile’а и запуск контейнера

Индивидуальное задание

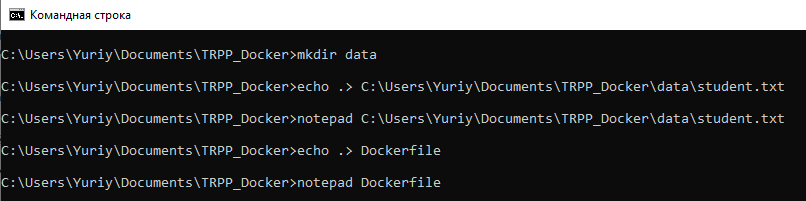


Рисунок 30 − Создание директории data и файла student.txt в этой директории

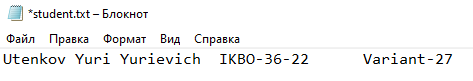


Рисунок 31 − Содержимое файла student.txt

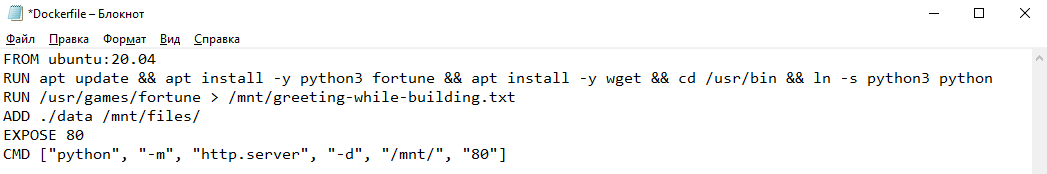


Рисунок 32 − Содержимое Dockerfile’а для данной задачи

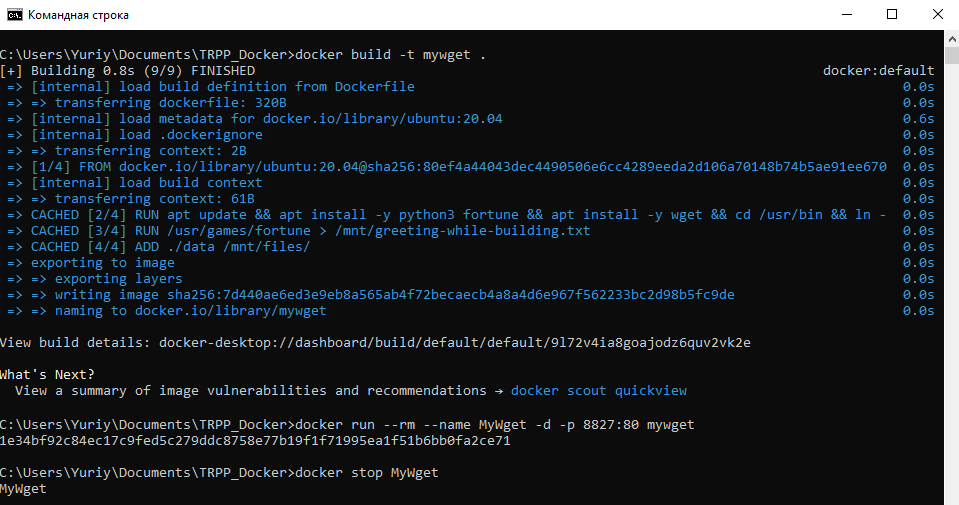


Рисунок 33 − Построение собственного образа на основе официального образа ubuntu, версии 20.04 и запуск контейнера

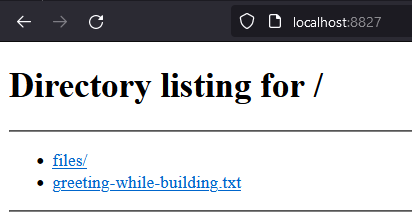


Рисунок 34 − Отображение содержимого директории /mnt/files на сервере

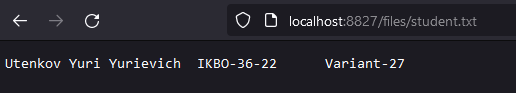


Рисунок 35 − Отображение файла student.txt на сервере

# Выводы

В ходе выполнения данной работы я вспомнил и узнал о новых функциях docker’а. Docker позволяет быстро и удобно развёртывать большие приложения с большим количеством зависимостей, т. к. всё необходимое для проекта находится непосредственно внутри контейнера. Контейнеры представляют собой ядро какой-либо ОС, они виртуализируют только необходимые компоненты и библиотеки для запуска приложения, за счёт этого они получаются очень легковесными, в отличие от ВМ.