|  |
| --- |
|  |
| 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 № 3** |
| **по дисциплине**  «ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ» |
| **Тема: «Docker»** |

|  |  |
| --- | --- |
| Выполнил студент группы ИКБО-32-21 | Быченков А.К. |
| Принял | Миронов А.И. |

|  |  |  |
| --- | --- | --- |
| Практическая работа выполнена | «\_\_»\_\_\_\_\_\_\_2023 г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2023 г. | *(подпись руководителя)* |

Москва 2023

**Ход работы**

1. **Образы**

Docker был установлен на локальную машину. Посмотрим на имеющиеся образы с помощью команды docker images:

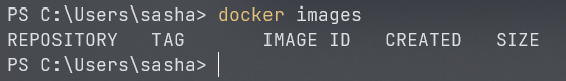
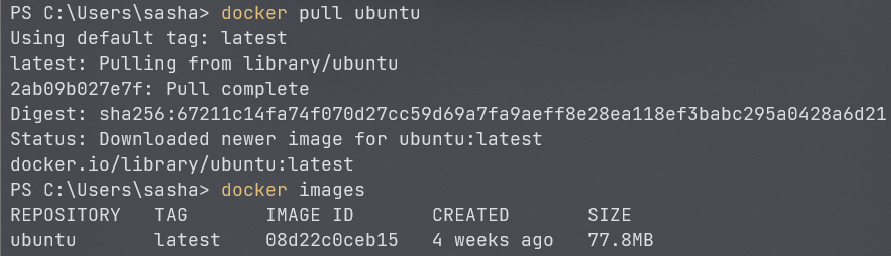


Рисунок 1. Образы docker

Docker-образов на локальной машине не найдено. Загрузим образ ubuntu командой docker pull ubuntu:

Рисунок 2. Загрузка образа ubuntu

1. **Изоляция**

Посмотрим информацию о хостовой системе, выполнив команду hostname:

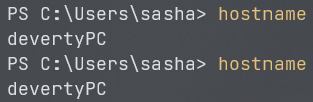


Рисунок 3. Команда hostname на локальной машине

Если выполнить команду hostname на локальной машине несколько раз, получим один и тот же результат, ведь локальная машина останется прежней.

Попробуем выполнить эту же команду в контейнерах. Для этого запустим команду docker run ubuntu hostname:

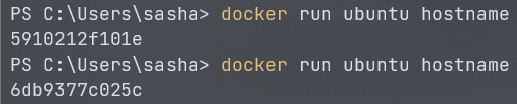


Рисунок 4. Команда hostname в контейнерах

Получили различный результат, так как были запущены два изолированных контейнера с разным именем хоста.

Выполним команду docker ps -a для просмотра запущенных ранее контейнеров:



Рисунок 5. Запущенные ранее контейнеры

Запустим контейнер в интерактивном режиме, указав флаги -it:

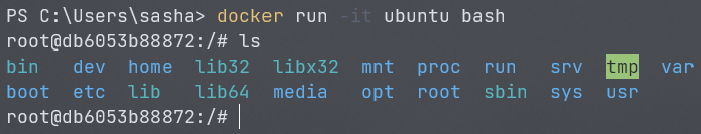


Рисунок 6. Запуск контейнера в интерактивном режиме

1. **Работа с портами**

Загрузим образ python командой docker pull python:

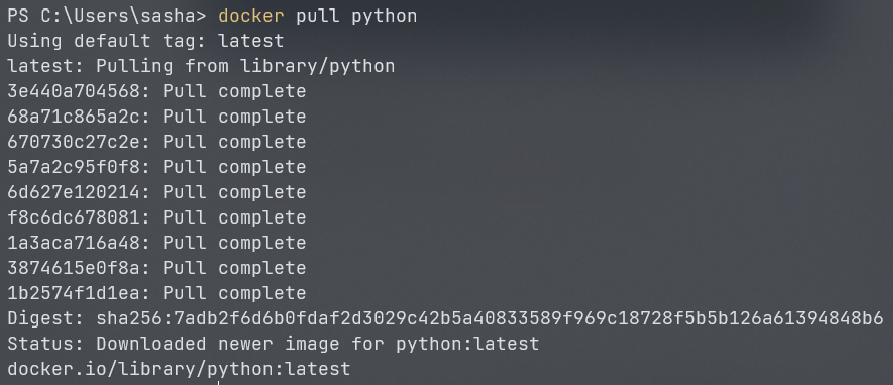


Рисунок 7. Загрузка образа python

Далее необходимо запустить контейнер с этим образом и выполнить в нем команду для запуска python-модуля веб-сервера. Чтобы доступный в контейнере веб-сервер был доступен на локальной машине, необходимо пробросить порты. Для этого при запуске контейнера укажем флаг -p, порт на локальной машине, порт в контейнере:

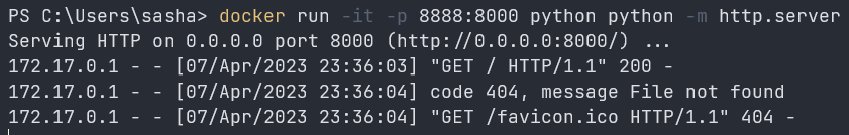


Рисунок 8. Запуск контейнера с веб-сервером

Была проверена доступность веб-сервера по адресу <http://localhost:8888>, и сервер был остановлен комбинацией клавиш Ctrl+C.

1. **Именованные контейнеры, остановка и удаление**

Для того, чтобы запустить контейнер в фоновом режиме, необходимо добавить флаг -d. Также определим имя контейнеру, добавив флаг –name. Запустим веб-сервер в фоновом режиме и дадим ему имя pyserver:

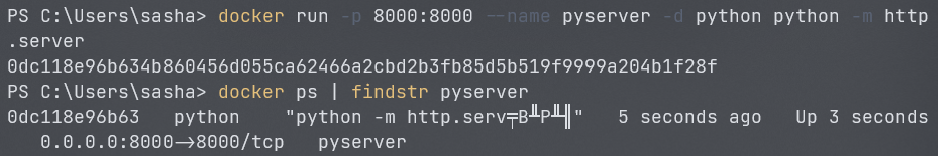


Рисунок 9. Запуск контейнера в фоновом режиме

Проверим, что контейнер все ещё запущен, командой docker ps | findstr pyserver:

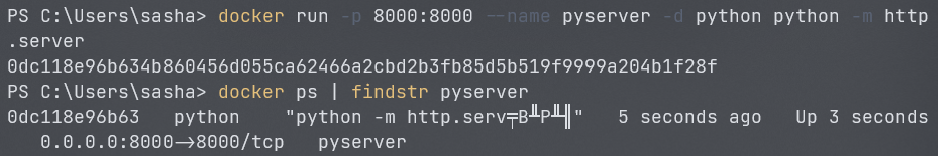


Рисунок 10. Запущенный контейнер

Просмотрим логи запущенного контейнера командой docker logs:

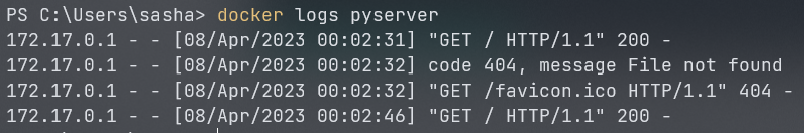


Рисунок 11. Логи запущенного контейнера

Теперь остановим и удалим этот контейнер командой docker rm -f:

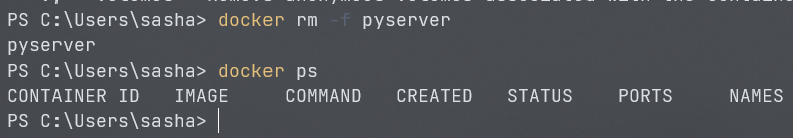


Рисунок 12. Удаление запущенного контейнера

1. **Постоянное хранение данных**

Запустим контейнер, в котором веб-сервер будет отдавать содержимое директории /mnt:

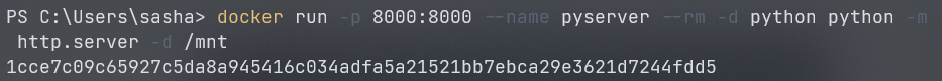


Рисунок 13. Запуск контейнера с python веб-сервером

При запуске контейнера были использованы следующие флаги:

1. -p – проброс порта. Используется для возможности доступа к веб-серверу в контейнере на локальной машине. Порт 8000 в контейнере связывается с портом 8000 на локальной машине.
2. С помощью флага –name устанавливается имя контейнера. Используется для удобной идентификации контейнера.
3. Флаг –rm указывает, что контейнер должен автоматически удаляться после остановки
4. Флаг -d указывает, что контейнер будет запущен в фоновом режиме

В запущенном контейнере с образом python выполнилась команда python -m http.server -d \mnt для запуска модуля веб-сервера. Используется флаг -d для указания корневой директории для отображения.

Теперь выполним в запущенном контейнере команду docker exec -it pyserver bash для открытия интерактивной оболочки bash:

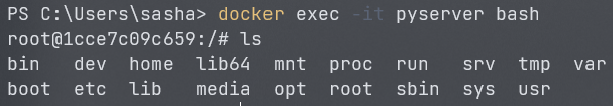


Рисунок 14. Запуск интерактивной оболочки bash

Далее создадим в директории mnt файл hi.txt и выйдем из контейнера комбинацией клавиш Ctrl+D:

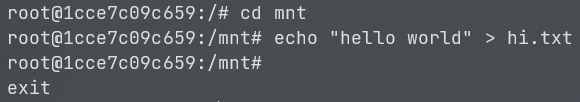


Рисунок 15. Создание файла hi.txt

Обратимся к веб-серверу по адресу localhost:8000 и увидим созданный файл hi.txt:

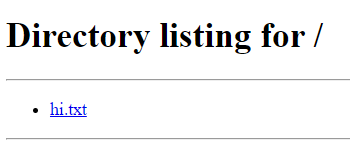


Рисунок 16. Созданный файл hi.txt

Теперь остановим контейнер и снова запустим:

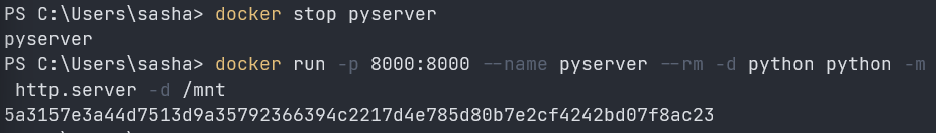


Рисунок 17. Остановка и повторный запуск контейнера

Файл hi.txt пропал, так как старый контейнер был удален после завершения работы:

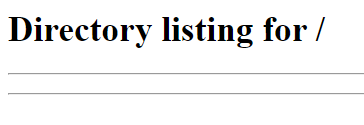


Рисунок 18. Вид корневой директории

Остановим контейнер командой docker stop pyserver:



Рисунок 19. Остановка контейнера

* 1. **Тома**

Попробуем создать контейнер с примонтированным томом, используя ключ -v:

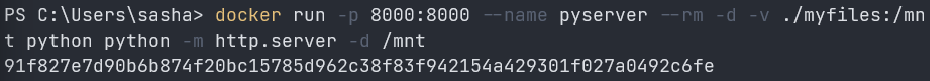


Рисунок 20. Создание контейнера с примонтированным томом

Далее создадим файл hi.txt в контейнере:

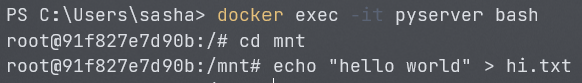


Рисунок 21. Создание файла hi.txt в контейнере

Посмотрим, где хранится том на хостовой машине командой docker inspect:

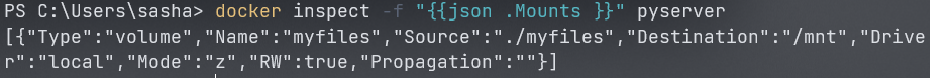


Рисунок 22. Информация о томе

* 1. **Монтирование директорий и файлов**

Сперва остановим контейнер командой docker stop pyserver. Далее создадим директорию и файлы, которые будем монтировать:

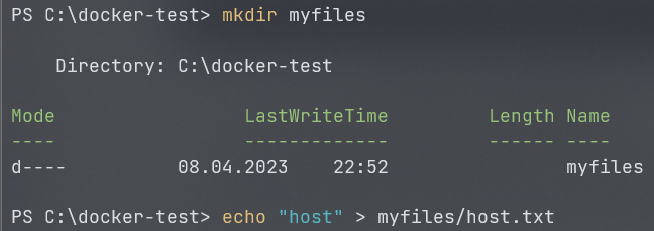


Рисунок 23. Создание директории для монтирования

Теперь запустим контейнер с примонтированной директорией:

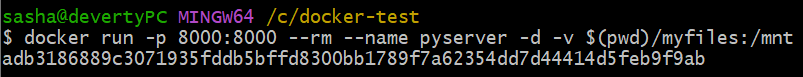


Рисунок 24. Запуск контейнера с примонтированной директорией

Далее зайдем в контейнер в интерактивном режиме, перейдем в директорию /mnt и выведем список файлов командой ls:

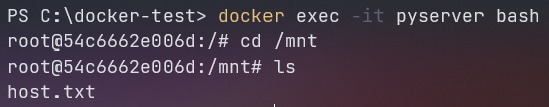


Рисунок 25. Список файлов в примонтированной директории

Создадим файл hi.txt, выйдем из контейнера и проверим созданный файл на хостовой машине:

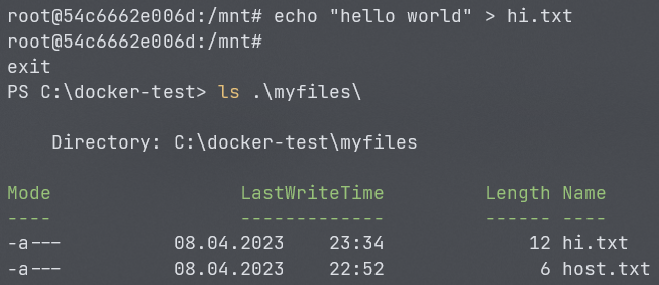


Рисунок 26. Созданный файл на хостовой машине

Остановим контейнер и попробуем примонтировать один файл:

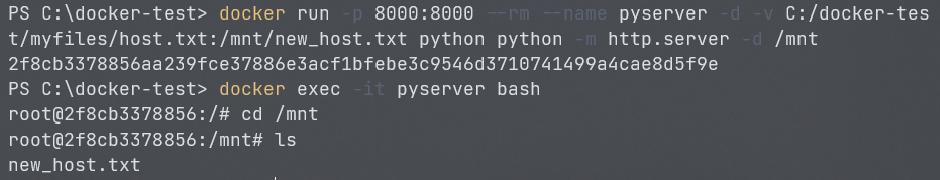


Рисунок 27. Монтирование файлов

1. **Переменные окружения**

Передадим в контейнер переменную окружения, используя ключ -e, и выведем все переменные окружения командой env:

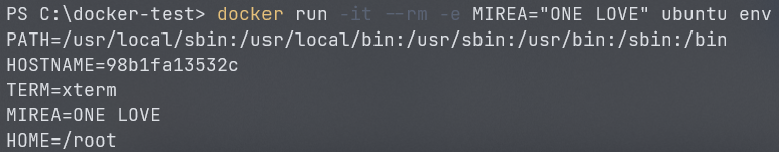


Рисунок 28. Передача переменных окружения

1. **Dockerfile**

Создадим Dockerfile для сборки образа на основе образа ubuntu с python веб-сервером:

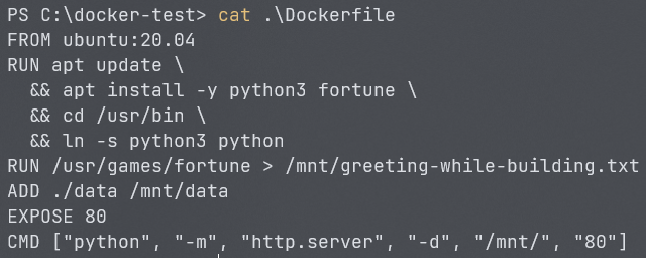


Рисунок 29. Содержимое Dockerfile

Соберем этот образ с помощью команды docker build:

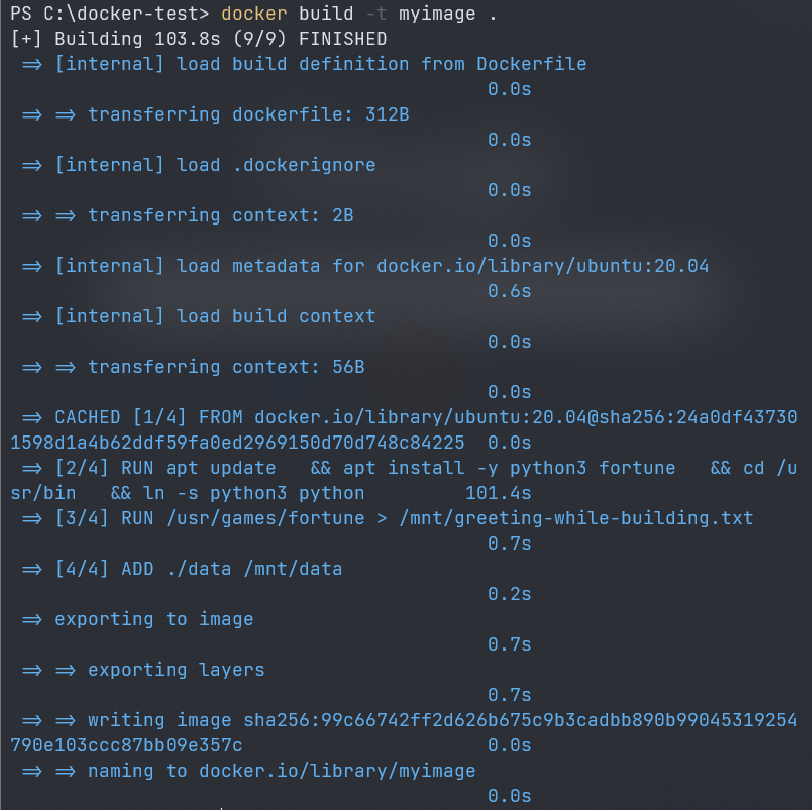


Рисунок 30. Сборка образа

Теперь запустим образ командой docker run:



Рисунок 31. Запуск собранного образа

1. **Индивидуальные задания**

Вариант 4.

Напишем Dockerfile согласно индивидуальному заданию. Необходимо собрать образ на основе ubuntu:20.10 (но версия ubuntu:20.10 поддерживалась до 22 июля 2021 и выполнить команду apt update не получится, поэтому в образе будет использоваться ubuntu:20.04 c LTS до 2025 года), примонтировать файл data/student.txt как /mnt/files/student.txt, запустить веб-сервер, отображающий содержимое /mnt/files, который в хостовой системе должен открываться на порту 8804 и установить пакет imagemagick:

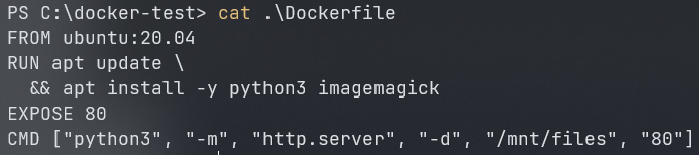


Рисунок 32. Dockerfile

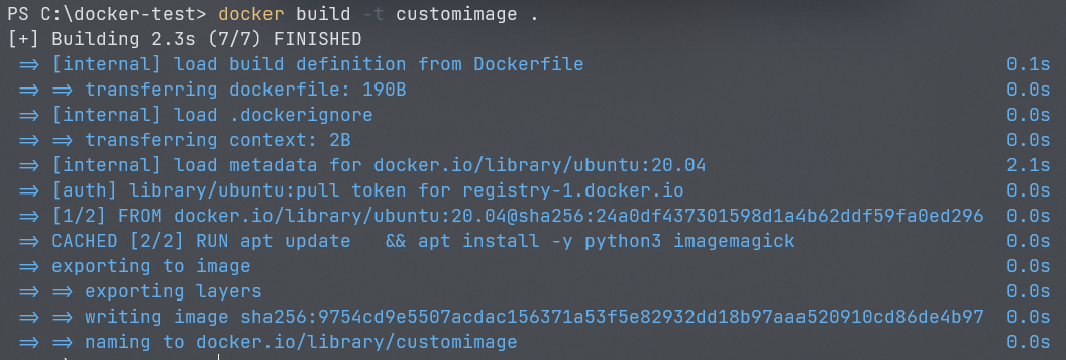


Рисунок 33. Сборка образа

Теперь запустим контейнер командой docker run, примонтируем необходимый файл:

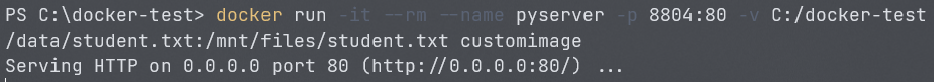


Рисунок 34. Запуск контейнера

Просмотрим содержимое файла student.txt на веб-сервере:

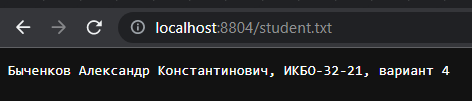


Рисунок 35. Содержимое файла student.txt

**Вывод**

В ходе выполнения данной практической работы я научился работать с Docker, запускать контейнеры с различными настройками, в том числе с монтированием файлов и директорий, собирать собственные Docker-образы.