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 группы ИКБО-04-22 Оганнисян Г.А.

Проверил: Исобекова О.А.

Москва 2024

Оглавление

Цель практической работы.....3
Выполнение практической работы.....4
Часть 1. Образы. 4
Часть 2. Изоляция.....5
Часть 3. Работа с портами. 7
Часть 4. Именованные контейнеры, остановка и удаление.....9
Часть 5. Постоянное хранение данных. 11
Часть 6. Переменные окружения.....17
Часть 7. Dockerfile.....18
Часть 8. Индивидуальное задание. Вариант 19.....20
Вывод.....220

Цель практической работы

Получить практические навыки для работы с Docker.

Выполнение практической работы

Часть 1. Образы.

Посмотрим на имеющиеся образы: docker images.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    d2c94e258dcb   11 months ago   13.3kB
```

Рисунок 1.1 — Просмотр образов

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

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
3c645031de29: Pull complete
Digest: sha256:1b8d8ff4777f36f19bfe73ee4df61e3a0b789caeff29caa019539ec7c9a57f95
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

Рисунок 1.2 — Загрузка образа

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

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
ubuntu        latest    7af9ba4f0a47   7 days ago      77.9MB
hello-world   latest    d2c94e258dcb   11 months ago   13.3kB
```

Рисунок 1.3 — Просмотр образов

Посмотрите список контейнеров, выполнив команду: docker ps -a

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
effc109f3ca6   hello-world   "/hello"   14 minutes ago   Exited (0) 14 minutes ago             modest_rubin
```

Рисунок 1.4 — Просмотр списка контейнеров

Часть 2. Изоляция.

Посмотрим информацию о хостовой системе, выполнив команду hostname.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>hostname
pcpi
```

Рисунок 2.1 — Просмотр информации о хостовой системе

Выполним её ещё один раз.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>hostname
pcpi
```

Рисунок 2.2 — Просмотр информации о хостовой системе

Вопрос: одинаковый ли результат получился при разных запусках?

Ответ: Результаты могут быть разными при разных запусках команды hostname. Это зависит от настроек и конфигурации хостовой системы.

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

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker run ubuntu hostname
d0bf99583d79
```

Рисунок 2.3 — Просмотр информации о хостовой системе

Вопрос: Одинаковый ли результат получился при разных запусках?

Ответ: При выполнении команды docker run ubuntu hostname в контейнерах результаты могут быть разными при каждом запуске. Каждый запуск команды создает новый контейнер на основе образа Ubuntu и выводит имя хоста контейнера. Поэтому результаты при разных запусках могут быть разными.

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

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker ps -a
CONTAINER ID   IMAGE         COMMAND      CREATED              STATUS                          PORTS     NAMES
d0bf99583d79   ubuntu        "hostname"   About a minute ago   Exited (0) About a minute ago             compassionate_t
uring
effc109f3ca6   hello-world   "/hello"     26 minutes ago       Exited (0) 26 minutes ago                 modest_rubin
```

Рисунок 2.4 — Выполнение docker ps -a

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

Запустим bash в контейнере: docker run ubuntu bash.

Рисунок 2.5 — Запуск bash в контейнере

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, нам нужно добавить флаги -i -t или сгруппировано -it: docker run -it ubuntu bash.

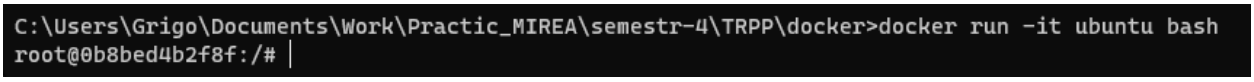

Рисунок 2.6 — Выполнение -it: docker run -it ubuntu bash

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

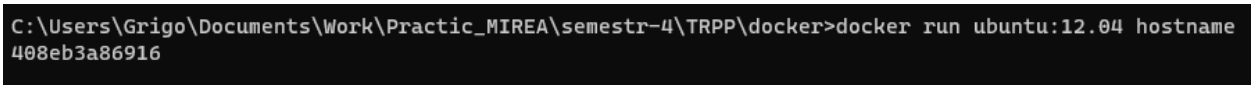

Рисунок 2.7 — Выполнение docker run ubuntu:12.04 hostname

Часть 3. Работа с портами.

Для начала, загрузим образ python командой docker pull python

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker pull python
Using default tag: latest
latest: Pulling from library/python
609c73876867: Pull complete
7247ea8d81e6: Pull complete
be374d06f382: Pull complete
b4580645a8e5: Pull complete
aa7e0aca67dd: Pull complete
84816cb735e2: Pull complete
85e25f7ceb91: Pull complete
849540060de4: Pull complete
Digest: sha256:e0e2713ebf0f7b114b8bf9fbcaba9a69ef80e996b9bb3fa5837e42c779dcdc0f
Status: Downloaded newer image for python:latest
docker.io/library/python:latest
```

Рисунок 3.1 — Загрузка образа python

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

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>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 3.2 — Запуск модуля веб-сервера

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.

Directory listing for /

Рисунок 3.3 — Открытие адреса http://0.0.0.0:8000/

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

```
^C
Keyboard interrupt received, exiting.
```

Рисунок 3.4 — Завершение работы веб-сервера

Для проброса портов используется флаг -p hostPort:containerPort Добавим 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.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>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/) ...
172.17.0.1 - - [18/Apr/2024 13:48:55] "GET / HTTP/1.1" 200 -
```

Рисунок 3.5 — Проброс порта 8000

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.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>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/) ...
172.17.0.1 - - [18/Apr/2024 13:49:55] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [18/Apr/2024 13:49:55] code 404, message File not found
172.17.0.1 - - [18/Apr/2024 13:49:55] "GET /favicon.ico HTTP/1.1" 404 -
```

Рисунок 3.6 — Проброс порта 8888

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

```
^C
Keyboard interrupt received, exiting.
```

Рисунок 3.7 — Завершение работы веб-сервера

Часть 4. Именованные контейнеры, остановка и удаление.

Запустим контейнер: docker run -it -p8000:8000 python python -m http.server.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>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/) ...
172.17.0.1 - - [18/Apr/2024 13:48:55] "GET / HTTP/1.1" 200 -
```

Рисунок 4.1 — Запуск контейнера

Нажмем Ctrl+C — выполнение завершится.

```
^C
Keyboard interrupt received, exiting.
```

Рисунок 4.2 — Завершение выполнения

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

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker run -p8000:8000 --name pyserver -d py
thon python -m http.server
569113cb597f40da6917caaea4828085528a2513bfd2e8e97ca5ea6d14db47fa
```

Рисунок 4.3 — Запуск контейнера в фоне

Убедимся, что контейнер всё ещё запущен: docker ps | findstr pyserver — вывод команды не должен быть пустым.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker ps | findstr pyserver
569113cb597f   python    "python -m http.servтАж"   3 minutes ago   Up About a minute   0.0.0.0:8000->8000/tcp
   pyserver
```

Рисунок 4.5 — Проверка запущенности контейнера

Для просмотра логов контейнера, воспользуемся командой docker logs pyserver.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker logs pyserver
```

Рисунок 4.6 — Просмотр логов контейнера

Для того, чтобы остановить выполнение контейнера, существует команда docker stop pyserver.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker stop pyserver
pyserver
```

Рисунок 4.7 — Остановка контейнера

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.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>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 "5
69113cb597f40da6917caaea4828085528a2513bfd2e8e97ca5ea6d14db47fa". You have to remove (or rename) that containe
r to be able to reuse that name.
See 'docker run --help'.
```

Рисунок 4.8 — Попытка запуска контейнера

Его нужно удалить docker rm pyserver.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker stop pyserver
pyserver
```

Рисунок 4.6 — Остановка контейнера

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker rm pyserver
pyserver
```

Рисунок 4.7 — Удаление контейнера

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

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker run --rm -p8000:8000 --name pyserver
-d python python -m http.server
37484302197eb85af1db7b475291e14f3bcab5071e744f018d24a1ea397ccc6c
```

Рисунок 4.8 — Создание контейнера с возможностью удаления после завершения работы

Часть 5. Постоянное хранение данных.

Запустим 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.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker run -p8000:8000 --name pyserver --rm
-d python python -m http.server -d /mnt
7448404fa7b9cc337ddebb385936658b37cafc2ca23e4bfdf1c3b58c70a51089
```

Рисунок 5.1 — Запуск контейнера, в котором веб-сервер будет отдавать содержимое директории /mnt

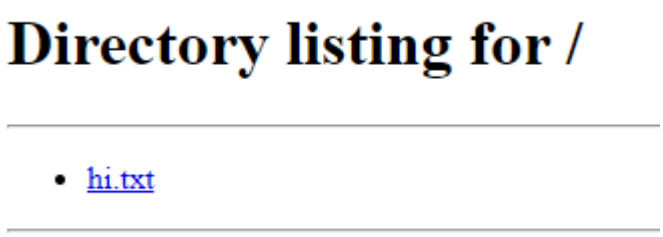

Рисунок 5.1 — Содержимое директории /mnt

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

1. -p 8000:8000: Этот флаг указывает Docker на проброс портов. Он говорит Docker, что порт 8000 в контейнере должен быть доступен с хостовой машины в порту 8000.
2. --name pyserver: Этот флаг задает имя контейнера (в данном случае "pyserver") для удобства обращения к нему в дальнейшем.
3. --rm: Этот флаг указывает Docker на удаление контейнера после его завершения. Это позволяет избежать накопления неиспользуемых контейнеров.
4. -d: Этот флаг запускает контейнер в фоновом режиме (detached mode), позволяя вам продолжать работу с командной строкой.
5. python python -m http.server -d /mnt: Эта команда указывает Docker на выполнение команды python -m http.server -d /mnt внутри контейнера. Эта команда запустит встроенный веб-сервер Python, который будет отдавать содержимое директории /mnt.

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

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker exec -it pyserver sh
# cd mnt && echo "hello world" > hi.txt
```

Рисунок 5.2 — Переход в оболочку bash в контейнере, выполнение команды cd mnt && echo "hello world" > hi.txt

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

Рисунок 5.2 — Доступность файла hi.txt

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 5.3 — Остановка и запуск контейнера

Как мы видим, файл hi.txt пропал — это неудивительно, ведь мы запустили другой контейнер, а старый был удалён после завершения работы (флаг --rm).

Рисунок 5.3 — Содержимое директории

Остановим контейнер: docker stop pyserver.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker stop pyserver
pyserver
```

Рисунок 5.4 — Остановка контейнера

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.

Попробуем 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

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker run -p8000:8000 --rm --name pyserver
-d -v C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker\myfiles:/mnt python python -m http.ser
ver -d /mnt
756ef5f35e88ba738a1eec4d36506a52cb1675a18663280c6f7a4370a30f61b3
```

Рисунок 5.5 — Создание контейнера 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.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker exec -it pyserver bash
root@756ef5f35e88:/# cd mnt && echo "hello world" > hi.txt
root@756ef5f35e88:/mnt#
```

Рисунок 5.5 — Создание файла

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

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker volume ls
DRIVER    VOLUME NAME
local     57d7ee6ee5266f17e063ff6f4d9bbfaacae91d43ba3843d283339df150ab1788
```

Рисунок 5.6 — Путь по которому хранятся данные

Для управления томами существует команда docker volume.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker volume create new
new

C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker volume ls
DRIVER    VOLUME NAME
local     57d7ee6ee5266f17e063ff6f4d9bbfaacae91d43ba3843d283339df150ab1788
local     new
```

Рисунок 5.7 — Создание Docker тома и
Вывод списка всех Docker томов

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker volume inspect 57d7ee6ee5266f17e063ff6f4d9bbfaa
cae91d43ba3843d283339df150ab1788
[
    {
        "CreatedAt": "2024-04-18T15:31:23Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/57d7ee6ee5266f17e063ff6f4d9bbfaacae91d43ba3843d283339df150ab1788/_data",
        "Name": "57d7ee6ee5266f17e063ff6f4d9bbfaacae91d43ba3843d283339df150ab1788",
        "Options": null,
        "Scope": "local"
    }
]
```

Рисунок 5.9 — Вывод подробной информации о Docker томе

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker volume rm new
new
```

Рисунок 5.10 — Удаление Docker тома

5.2 Монтирование директорий и файлов.

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

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker stop pyserver
pyserver
```

Рисунок 5.10 — Остановка контейнера

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: создадим директорию: mkdir myfiles, в ней создайте файл host.txt: touch myfiles/host.txt Запустим 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.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker\myfiles>docker run -p8000:8000 --rm --name pyserver -d
 -v C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker\myfiles:/mnt python  python -m http.server -d /mnt

dd1e9f8b08e9d86d8acd5b4651d3bd70b3ddb8561bd9b3295a79d12194693fbd
```

Рисунок 5.10 — Запуск контейнера в конфигурационный файл

Затем, зайдем в контейнер: docker exec -it pyserver bash, перейдем в директорию /mnt командой cd /mnt.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker\myfiles>docker exec -it pyserver bash
root@dd1e9f8b08e9:/# cd /mnt
root@dd1e9f8b08e9:/mnt#
```

Рисунок 5.11 — Переход в контейнер, в директорию /mnt

Если вывести список файлов командой ls, то там будет файл host.txt, примонтированный вместе с директорией myfiles

```
root@dd1e9f8b08e9:/mnt# ls
host.txt
```

Рисунок 5.12 — Вывод списка файлов

Создадим файл echo "hello world" > hi.txt, а затем выйдем из контейнера: exit.

```
root@dd1e9f8b08e9:/mnt# echo "hello world" > hi.txt
root@dd1e9f8b08e9:/mnt# exit
exit
```

Рисунок 5.12 — Создание файла и выход из контейнера

Теперь на хостовой машине в директории myfiles/ появится файл hi.txt.

Рисунок 5.12 — Файлы хостовой машины

Остановим контейнер: docker stop pyserver.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker\myfiles>docker stop pyserver
pyserver
```

Рисунок 5.13 — Остановка контейнера

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.

```
C:\Users\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker>docker run -p8000:8000 --rm --name pyserver -d -v C:\U
sers\Grigo\Documents\Work\Practic_MIREA\semestr-4\TRPP\docker\myfiles\host.txt:/mnt/new-name-of-host.txt python python -
m http.server -d /mnt
87da1dd43ecb691f81ce0c0435872003f4eec4df2baa05fb35d754404b6f79f6
```

Рисунок 5.14 — Присвоение файлу другого имени

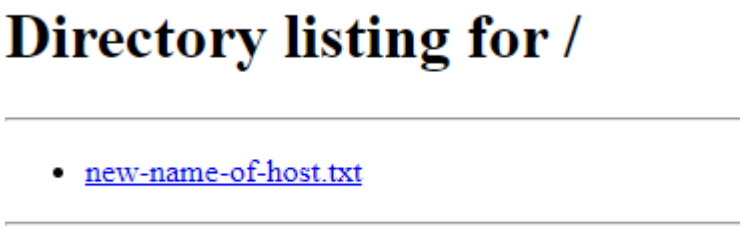

Рисунок 5.15 — Переименованный файл

Часть 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". Проверим, 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

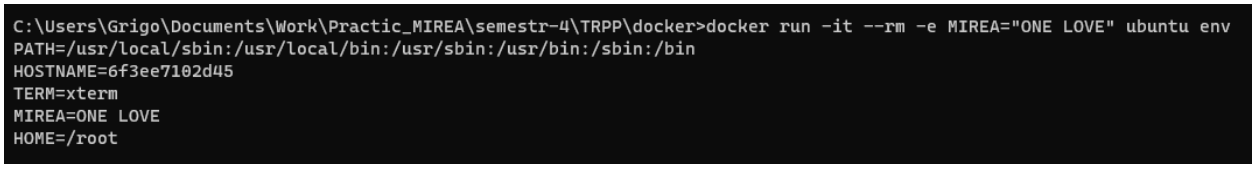

Рисунок 6.1 — Передача переменной окружения в контейнер

Часть 7. Dockerfile.

Соберем образ, в который будут установлены дополнительные пакеты, примонтируем директорию и установим команду запуска. Для этого создаётся файл Dockerfile (без расширения).

1 FROM ubuntu :20.04

2 RUN apt update \

3 && apt install - y python3 fortune \

4 && cd /usr/bin \

5 && ln - s python3 python

6 RUN /usr/games/fortune > /mnt/greeting - while - building . txt

7 ADD ./data / mnt/data 8 EXPOSE 80 9 CMD [" python " , " - m " , " http . server " , " - d " , "/ mnt /" , "80"]

В строке (1) указывается базовый образ, на основе которого будет строиться новый образ. В строках (2-5) указана команда, которая выполнится в процессе сборки. На самом деле, там выполняются несколько команд, соединённых && для того, чтобы создавать меньше слоёв в образе. В строках (6) тоже указана команда, которая сгенерирует случайную цитату и перенаправит вывод в файл /mnt/greeting-while-building.txt. Файл будет сгенерирован во время сборки образа. В строке (7) копируется всё содержимое директории ./data хостовой машины в директорию /mnt, которая будет доступна в контейнере. В строке (8) указывается, какой порт у контейнера будет открыт. В строке (9) указывается команда, которая будет выполнена при запуске, где 80 — порт, который будет слушать веб-сервер.

emestr-4 > TRPP > docker > Dockerfile > ...

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

Рисунок 7.1 — Содержимое файла Dockerfile

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

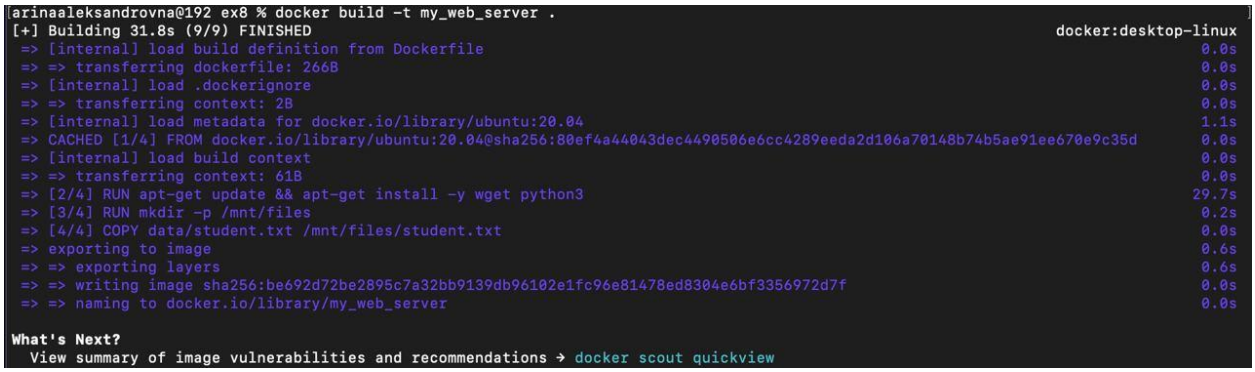

Рисунок 7.2 — Сборка образа

Часть 8. Индивидуальное задание. Вариант 19.

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. Чётные варианты:

- 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). Например, для 19-го варианта это порт 8819. Установить пакет, согласно варианту 19: imagemagick

```
/Practic_MIREA/semestr-4/TRPP/docker/part8/data$ nano student.txt
/Practic_MIREA/semestr-4/TRPP/docker/part8/data$ cd ..
/Practic_MIREA/semestr-4/TRPP/docker/part8$ nano Dockerfile
```

Рисунок 8.1 — Создание файлов

```
  GNU nano 6.2
19 Ogannisyan Grigor IKBO-15-22
```

Рисунок 8.2 — Содержимое файла student.txt

```
FROM ubuntu:20.10
RUN apt-get update && apt-get upgrade -y
RUN apt-get install -y imagemagick python3
RUN mkdir -p /mnt/files
COPY data/student.txt /mnt/files/student.txt
EXPOSE 8819
CMD ["python3", "-m", "http.server", "--directory", "/mnt/files", "8819"]
```

Рисунок 8.3 — Содержимое файла Dockerfile

```
piglin@pcpi:/mnt/c/Users/Grigo/Documents/Work/Practic_MIREA/semestr-4/TRPP/docker/part8$ docker build -t student_server .
[+] Building 165.9s (10/10) FINISHED                                                                                docker:default
 => [internal] load build definition from Dockerfile                                                                         0.0s
 => => transferring dockerfile: 296B                                                                                         0.0s
 => [internal] load metadata for docker.io/library/ubuntu:20.04                                                              0.5s
 => [internal] load .dockerignore                                                                                            0.0s
 => => transferring context: 2B                                                                                              0.0s
 => CACHED [1/5] FROM docker.io/library/ubuntu:20.04@sha256:71b82b8e734f5cd0b3533a16f40ca1271f28d87343972bb4cd6bd6           0.0s
 => [internal] load build context                                                                                            0.0s
 => => transferring context: 61B                                                                                             0.0s
 => [2/5] RUN apt-get update && apt-get upgrade -y                                                                          24.5s
 => [3/5] RUN apt-get install -y imagemagick python3                                                                       139.4s
 => [4/5] RUN mkdir -p /mnt/files                                                                                            0.3s
 => [5/5] COPY data/student.txt /mnt/files/student.txt                                                                       0.0s
 => exporting to image                                                                                                       1.1s
 => => exporting layers                                                                                                      1.1s
 => => writing image sha256:1ada2c86bb284f22a919f7147381a196f67047fb6b832dde6cf3db3ec76149eb                                 0.0s
 => => naming to docker.io/library/student_server                                                                            0.0s
```

Рисунок 8.2 — Запуск веб-сервера

```
piglin@pcpi:/mnt/c/Users/Grigo/Documents/Work/Practic_MIREA/semestr-4/TRPP/docker/part8$ docker run -d -p 8819:8819 stude
nt_server
92ed6509c45d4bb226267444f2a4b402a142c162f79c181effd582fb84c913a3
```

Рисунок 8.3 — Успешный запуск веб-сервера

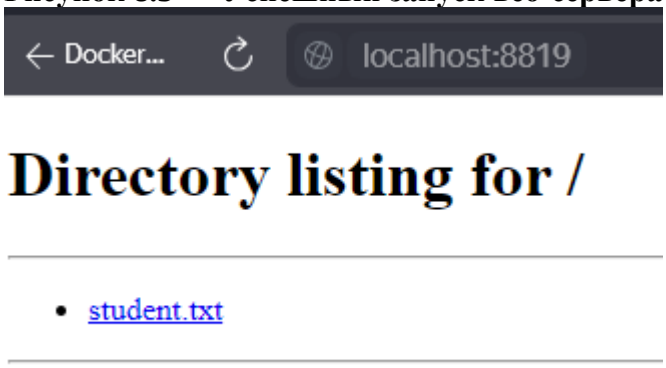

Рисунок 8.4 — Содержимое /mnt/files

Выводы

В ходе практической работы были получены практические и теоретические навыки работы с Docker.