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

«Системное программное обеспечение»

Тема практической работы:
«Системы конфигурационного управления»

Выполнил студент группы ИВБО-02-19

К. Ю. Денисов

Проверил ассистент

Ю. А. Вороноцов

Москва 2021

Содержание

1 Образы

С помощью данных команд проверим имеющиеся образы и контейнеры docker, загрузим образ ubuntu.

```
$ docker images
$ docker ps -a
$ docker pull ubuntu
```

2 Изоляция

Приведем ответ на вопрос *"Одинаковый ли результат получился при разных запусках?"*

При выполнении данной команды на локальной машине ответ ожидаемо один и тот же, но при выполнении данной команды в контейнере Docker, каждый раз ответ отличался. Это связано с тем, что из одного образа ubuntu были запущены два изолированных контейнера, поэтому у них и был разный hostname.

Запуск контейнеров производится командой:

```
$ docker run --flags --docker container_name commsnds -and --boot
    --flags --program
```

Запустим bash с возможностью входа в интерактивную оболочку, используя флаги `-if`:

```
$ docker run -it ubuntu bash.
```

Результаты выполнения команд пунктов 1 и 2 можно увидеть на рисунке 1 Приложении А.

3 Работа с портами

Загрузим образ `python` командой `docker pull python`. Запустим модуль веб-сервера из корня контейнера, чтобы отобразить содержание контейнера с помощью команд.

```
$ docker pull python
$ docker run -it python python m http.server
```

ля проброса портов используется флаг `-p hostPort:containerPort`. 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 -p8000:8000 python python -m http.server
```

4 Именованные контейнеры, остановка и удаление

Запустим контейнер в фоне, используя флаг `-d`, указав имя для него с помощью флага `-name`. После запуска контейнера можно удалить его, предварительно остановив используя команду `docker stop <name>`. Для того, чтобы контейнер удалялся после завершения работы, нужно указать флаг `-rm` при его запуске — далее в работе мы будем использовать данный флаг. Приведем команды, используемые в данном пункте:

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

Результаты выполнения команд из пунктов 3 и 4 можно увидеть на рис. 2 и 4 в Приложении А.

5 Постоянное хранение данных

Для того, чтобы иметь возможность сохранять результаты работы приложения в файлах, используем флаг `-d` после указания названия образа, который указывает модулю `http.server` какая директория будет корневой для отображения.

Выполним команду:

```
$ docker run -p8000:8000 --name pyserver --rm -d python python -m
    http.server -d /mnt
```

После выполнения данной команды запустим оболочку bash и создадим файл *hi.txt*, в который поместим строку *"hello world"*. Если Если открыть http://0.0.0.0:8000/, там будет доступен файл hi.txt.

После остановки контейнера и повторного запуска данный файл больше не будет доступен. Так как мы запустили новый контейнер, а старый вместе со всеми файлами, созданными им, был удален после завершения работы.

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

Приведем ответ на вопрос *"Что значат остальные флаги запуска? Где здесь команда, которая выполнится в контейнере?"*

- `--name` позволяет задавать имя для контейнера;
- `--rm` удалять контейнер после остановки;
- `-d` перед названием образа позволяет запускать контейнер в фоновом режиме;
- `-m` позволяет запустить модуль из корня контейнера;
- `-d` в конце указывает на то, какая директория будет корневой для отображения.

Команда, которая выполнится в контейнере — `python -m http.server -d /mnt`.

Результаты выполнения перечисленных команд можно увидеть на рис. 5 и 6 в Приложении А.

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.

Воспользуемся командой:

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

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 на хостовой машине. См. рис. 7 и 8 в Приложении А.

5.2 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. Часть из них нам понадобится дальше: создадим директорию: `mkdir myfiles`, в ней создадим файл *host.txt:* `touch myfiles/host.t`
Запустим контейнер:

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

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

6 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"`. 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. См. рис. 9.

7 Dockerfile

Соберем образ, в который будут установлены дополнительные пакеты, примонтируем директорию и установим команду запуска. Для этого создадим файл *Dockerfile*.

/home/denilai/gg/Dockerfile

```
FROM ubuntu:20.04
RUN apt update \
&& apt install -y python3 fortune \
&& cd / usr / bin \
&& ln -s python3 python
RUN /usr/games/fortune > /mnt/greeting-while-building.txt
ADD ./data /mnt/data
EXPOSE 80
CMD ["python3" , "-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 выпол- няются 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.

Соберем образ с тегом *mycoolimage* с помощью команды `docker build -t mycoolimage .`

Точка в конце указывает на текущую директорию, где лежит Dockerfile. Запуск производится командой `docker run -rm -it -p8099:80 mycoolimage`, где порт 8099 — порт на хостовой машине.

Результаты создания образа, запуска контейнера и обращения к серверу можно видеть на рисунках 10, **??** и **??** в Приложении А.

8 Индивидуальное задание

В качестве индивидуального задания предложено 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.

Условия согласно варианту:

- необходимо использовать базовый образ ubuntu:20.10;
- примонтировать файл data/student.txt как /mnt/files/student.txt в контейнере;

- установить пакет patch.

Запустить веб-сервер, отображающий содержимое /mnt/files, в хостовой системе должен открываться на порту (8800 + номер варианта). Например, для 22-го варианта это порт 8822.

Приведем содержание Dockerfile, который создает образ согласно вышеизложенным требованиям.

/home/denilai/data/Dockerfile

```
FROM ubuntu:20.04
RUN apt update \
&& apt install -y python3 fortune \
&& cd / usr / bin \
&& ln -s python3 python
RUN apt update && apt install -y patch
EXPOSE 80
CMD ["python3" , "-m" , "http.server" , "-d" , "/mnt/" , "80"]
```

Для сборки образа используем команду

```
$ docker build -t myvariant .
```

Для запуска контейнера по данному образу используем команду

```
$docker run --rm -it -p8806:80 -v
    \$(pwd)/student.txt:/mnt/files/student.txt myvariant
```

Результаты выполнения команд и открытие файлов на сервере можно увидеть на рисунках **??**, **??**, **??** и **??** в Приложении А.

9 Вывод

В ходе данной практической работы мы познакомились с командами по созданию Docker-образов, Docker-контейнеров, научились монтировать директории и отдельные файлы в контейнер, пробрасывать переменные окружения, настраивать внутренние и внешние порты, познакомились и разобрали структуру Dockerfile.

Применили все полученные знания для создания собственного образа и контейнера.

ПРИЛОЖЕНИЕ А

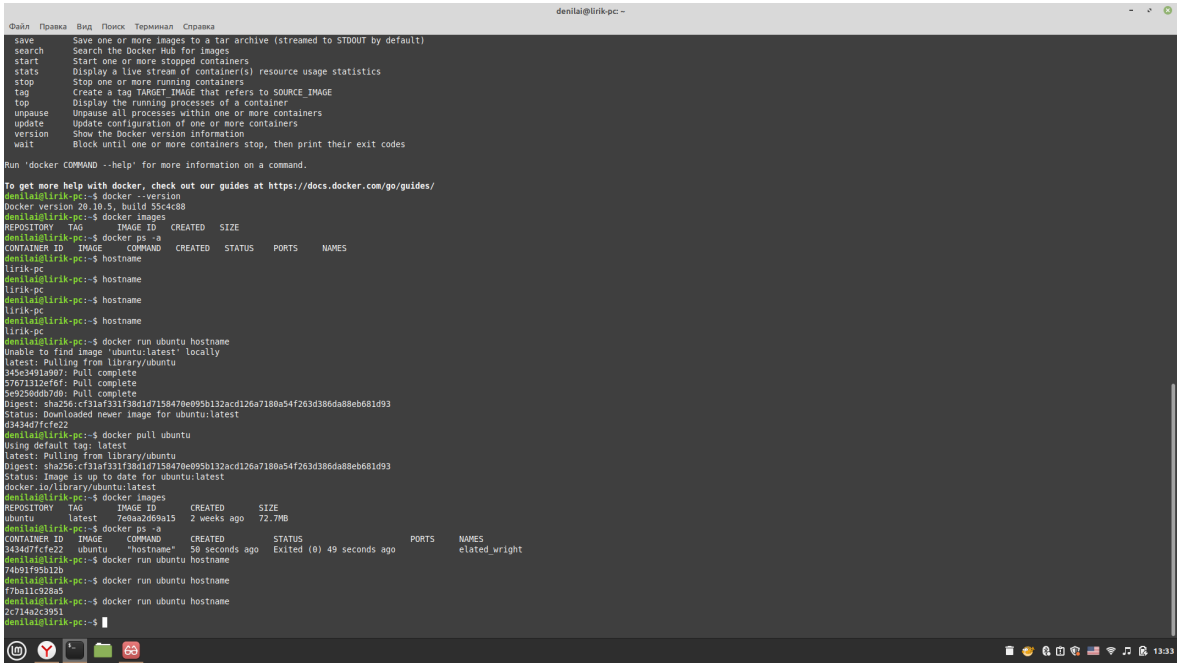

Рис. 1: Образы и изоляция

```
denilai@lirik-pc:~$ docker pull python
Using default tag: latest
latest: Pulling from library/python
bd8f6a7501cc: Pull complete
44718e6d535d: Pull complete
efe9738af0cb: Pull complete
f37aabde37b8: Pull complete
3923d444ed05: Pull complete
1ecef690e281: Pull complete
1c053581d9c9: Pull complete
81182ea718cf: Pull complete
83ebd4edf9af: Pull complete
Digest: sha256:0813df59b3d73a13fc581fd416d7733a2de6540d7e3f7633a1a9aabf9b201548
Status: Downloaded newer image for python:latest
docker.io/library/python:latest
denilai@lirik-pc:~$ docker run -it python python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
^C
Keyboard interrupt received, exiting.
denilai@lirik-pc:~$ 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 - - [08/May/2021 10:48:04] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [08/May/2021 10:48:05] code 404, message File not found
172.17.0.1 - - [08/May/2021 10:48:05] "GET /favicon.ico HTTP/1.1" 404 -
```

Рис. 2: Работа с портами

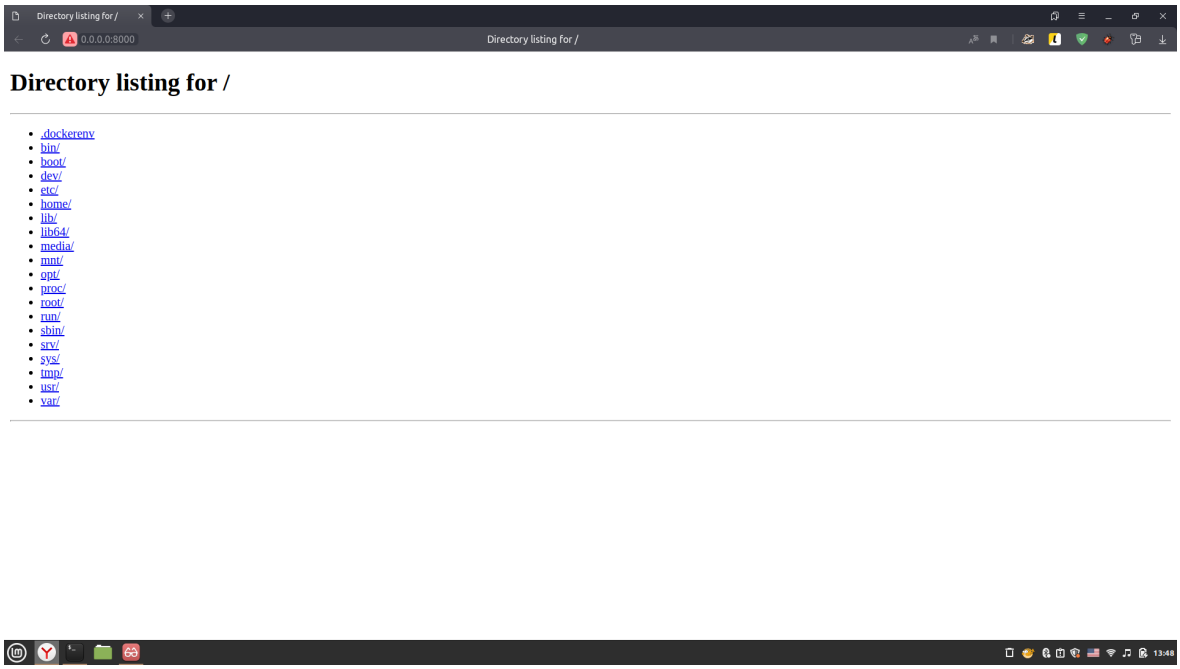

Рис. 3: Работа с портами. Веб-сервер

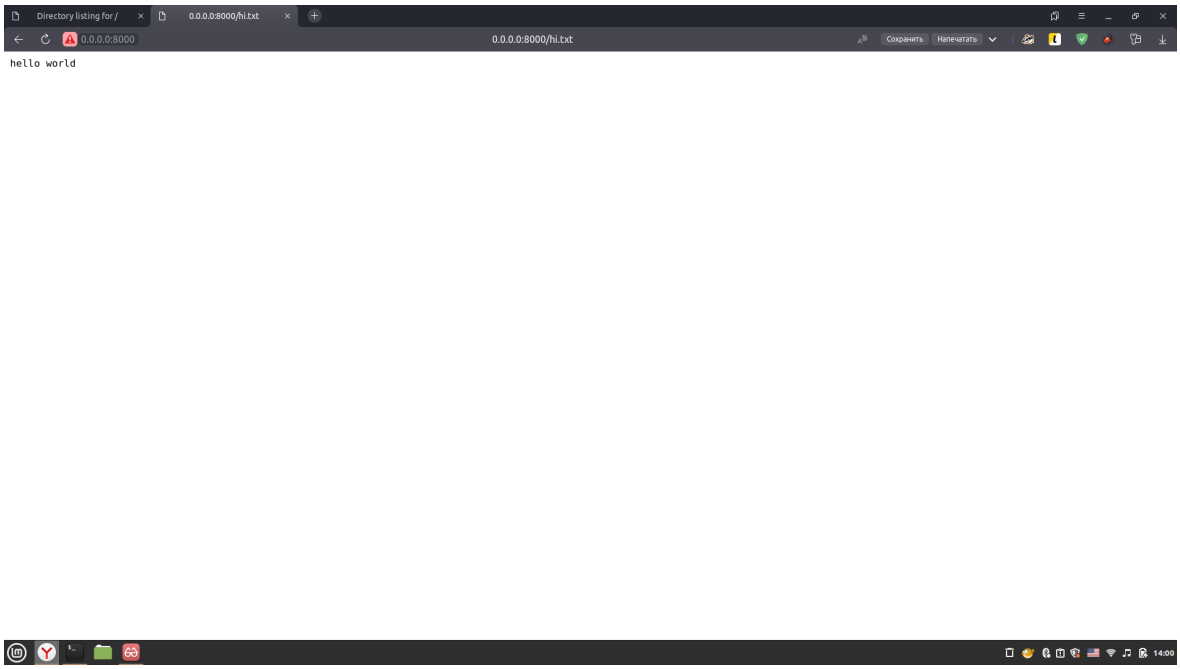

Рис. 4: Работа с портами. Файл

```
denilai@lirik-pc:~$ 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 - - [08/May/2021 10:48:04] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [08/May/2021 10:48:05] code 404, message File not found
172.17.0.1 - - [08/May/2021 10:48:05] "GET /favicon.ico HTTP/1.1" 404 -
^C
Keyboard interrupt received, exiting.
denilai@lirik-pc:~$ docker run -p8000:8000 --name pyserver -d python python -m http.server
fdbb5e1d7dd11ee212944e11a7f007edd27051170ab9a9b3d0c01de1571d841e
denilai@lirik-pc:~$ docker ps | grep pyserver
fdbb5e1d7dd1   python    "python -m http.serv…"   18 seconds ago   Up 17 seconds   0.0.0.0:8000->8000/tcp   pyserver
denilai@lirik-pc:~$ docker logs pyserver
denilai@lirik-pc:~$ docker rm -f pyserver
pyserver
denilai@lirik-pc:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      PORTS     NAMES
f3686f724b31   python    "python -m http.serv…"   2 minutes ago    Exited (0) 2 minutes ago              funny_mcclintock
3e10cc48da6e   python    "python -m http.serv…"   3 minutes ago    Exited (0) 3 minutes ago              stoic_mcclintock
9a64809ef617   ubuntu    "bash"                   15 minutes ago   Exited (0) 14 minutes ago             agitated_gates
2c714a2c3951   ubuntu    "hostname"               17 minutes ago   Exited (0) 17 minutes ago             elated_moser
f7ba11c928a5   ubuntu    "hostname"               17 minutes ago   Exited (0) 17 minutes ago             fervent_black
74b91f95b12b   ubuntu    "hostname"               17 minutes ago   Exited (0) 17 minutes ago             hungry_wilbur
3434d7fcfe22   ubuntu    "hostname"               19 minutes ago   Exited (0) 19 minutes ago             elated_wright
denilai@lirik-pc:~$
```

Рис. 5: Постоянное хранение

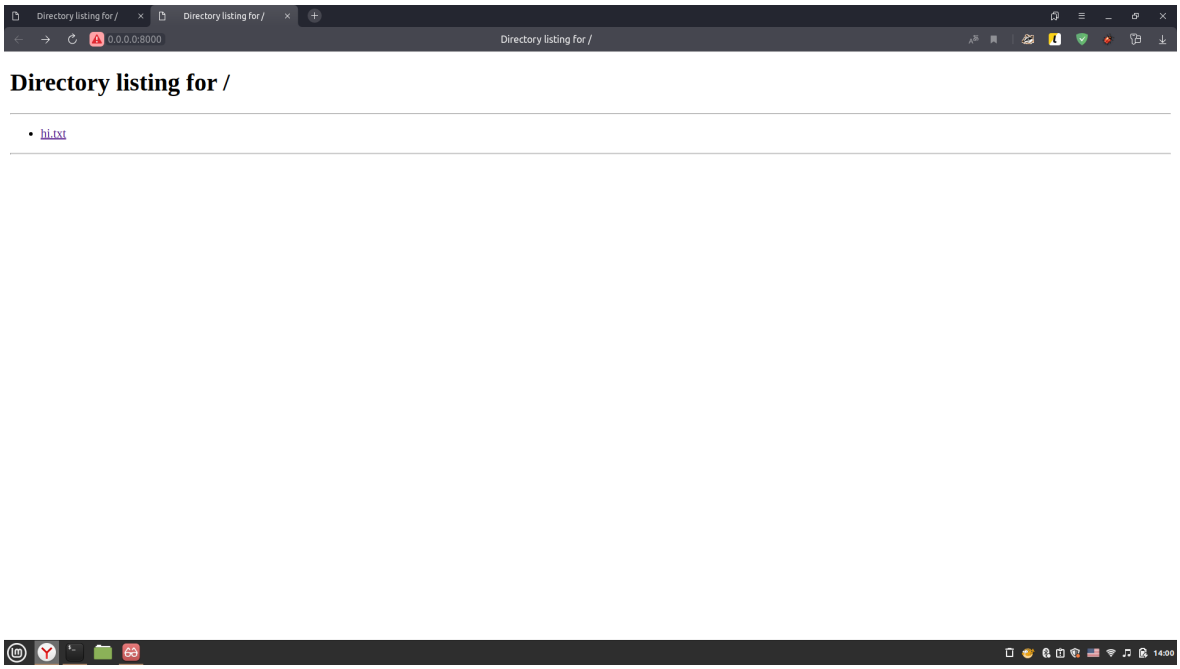

Рис. 6: Постоянное хранение. Веб-сервер

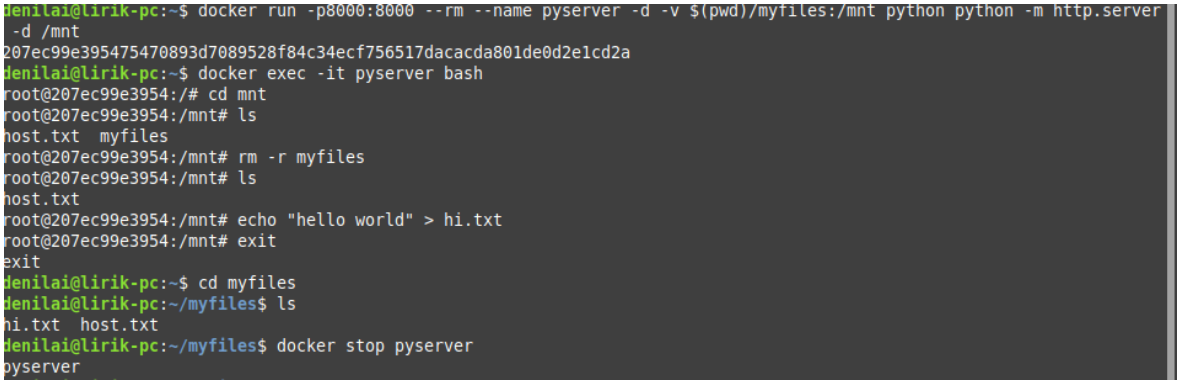

Рис. 7: Тома

```
denilai@lirik-pc:~$ docker run -p8000:8000 --rm --name pyserver -d -v $(pwd)/myfiles:/mnt python python -m http.server
 -d /mnt
c30c8be48deeab2166716c6ed031edce94535e4591506e7e2840333440ea4498
denilai@lirik-pc:~$ docker inspect -f "{{json .Mounts}}" pyserver
[{"Type":"bind","Source":"/home/denilai/myfiles","Destination":"/mnt","Mode":"","RW":true,"Propagation":"rprivate"}]
denilai@lirik-pc:~$
```

Рис. 8: Тома. Локальное место хранения

```
denilai@lirik-pc:~$ docker run -it --rm -e MIREA="ONE LOVE" ubuntu env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=d2bf04abe0b8
TERM=xterm
MIREA=ONE LOVE
HOME=/root
denilai@lirik-pc:~$
```

Рис. 9: Переменные окружения

```
denilai@lirik-pc:~/gg$ docker build -t mycoolimage .
Sending build context to Docker daemon  4.096kB
Step 1/6 : FROM ubuntu:20.04
 ---> 7e0aa2d69a15
Step 2/6 : RUN apt update && apt install -y python3 fortune && cd / usr / bin && ln -s python3 python
 ---> Using cache
 ---> be4cc969d994
Step 3/6 : RUN /usr/games/fortune > /mnt/greeting-while-building.txt
 ---> Using cache
 ---> 1f69d94b6de4
Step 4/6 : ADD ./data /mnt/data
 ---> Using cache
 ---> d74d6ce471df
Step 5/6 : EXPOSE 80
 ---> Using cache
 ---> 1f4e50d884ff
Step 6/6 : CMD ["python3" , "-m" , "http.server" , "-d" , "/mnt/" , "80"]
 ---> Running in 2b30ee4457cb
Removing intermediate container 2b30ee4457cb
 ---> 235fe9ed6085
Successfully built 235fe9ed6085
Successfully tagged mycoolimage:latest
denilai@lirik-pc:~/gg$ docker run --rm -it -p8099:80 mycoolimage
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
172.17.0.1 - - [08/May/2021 12:40:58] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [08/May/2021 12:40:59] code 404, message File not found
172.17.0.1 - - [08/May/2021 12:40:59] "GET /favicon.ico HTTP/1.1" 404 -
172.17.0.1 - - [08/May/2021 12:41:01] "GET /data/ HTTP/1.1" 200 -
172.17.0.1 - - [08/May/2021 12:41:04] "GET /greeting-while-building.txt HTTP/1.1" 200 -

 0  2h 59m  1 docker                                  100% | 15:44 | 08 мая  denilai  lirik-pc
```

Рис. 10: Dockerfile. Команды

,р,,п,,о,,н,,м,,л,

,р,,п,,о,,н,,м,,л,,р,,п,,о,,н,,м,,л,