Оглавление

1. ХОД РАБОТЫ

1.1 Образы

После установки докера необходимо посмотреть на все образы с помощью команды docker image (рис. 1).

Рисунок 1 – Просмотр образов

Далее с помощью команды docker pull Ubuntu была загружена Ubuntu последней версии (рис. 2).

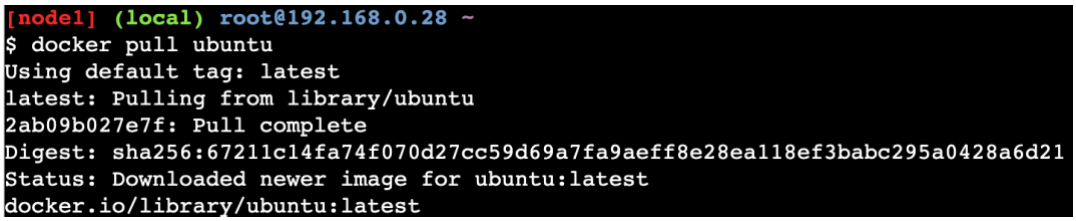

Рисунок 2 – загрузка последней версии Ubuntu

После установки последней Ubuntu версии необходимо посмотреть на все образы с помощью команды docker image (рис. 3).

Рисунок 3 – Просмотр образов

Просмотр список текущих контейнеров возможен с помощью команды docker ps -a (рис. 4)

Рисунок 4 – Просмотр текущих контейнеров

1.2 Изоляция

После требовалось вызвать hostname для хостовой системы и для контейнера с Ubuntu и посмотреть на результаты (рис. 5-6).

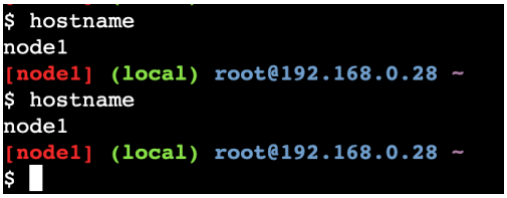

Рисунок 5 – Вызов hostname для хостовой системы

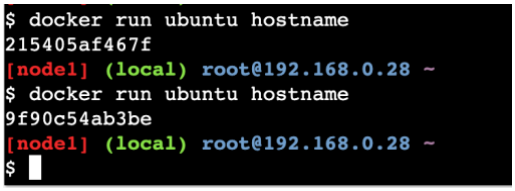

Рисунок 6 – Вызов hostname для контейнера с Ubuntu

В результате выполнения видно, что hostname для хостовой системы одинаковый, а для контейнеров разный. Это происходит потому, что при вызове hostname для контейнера мы открываем два контейнера, что видно на рисунке (рис. 7)

Рисунок 7 – Просмотр открытых контейнеров

Запуск bash из контейнера с Ubuntu производится с помощью команды docker run Ubuntu bash. Но при таком запуске ничего не будет выводиться. Для корректного запуска требуется указать флаги i и t, чтобы открылся интерактивный терминал.

Демонстрация запуска bash представлена на рисунке (рис. 8).

```
$ docker run ubuntu bash
[node1] (local) root@192.168.0.28 ~
$ docker run -it ubuntu bash
root@7e7ea015a111:/# docker run ubuntu hostname
bash: docker: command not found
root@7e7ea015a111:/# docker run ubuntu:latest hostname
bash: docker: command not found
root@7e7ea015a111:/#
```

Рисунок 8 – Запуск bash

1.3 Работа с портами

Загрузка образа python с помощью команды docker pull python и запуск встроенного в Python веб-сервера из корня контейнера (рис. 9).

```
[node1] (local) root@192.168.0.28 ~
$ docker pull python
Using default tag: latest
latest: Pulling from library/python
b0248cf3e63c: Pull complete
127e97b4daf7: Pull complete
0336c50c9f69: Pull complete
1b89f3c7f7da: Pull complete
2d6277217976: Pull complete
273fcda609d8: Pull complete
58568d3a3a00: Pull complete
56fc9fb54f6e: Pull complete
8a22f29afe36: Pull complete
Digest: sha256:f7382f4f9dbc51183c72d621b9c196c1565f713a1fe40c119d215c961fa22815
Status: Downloaded newer image for python:latest
docker.io/library/python:latest
[node1] (local) root@192.168.0.28 ~
$
```

Рисунок 9 – Загрузка образа python

После запуска при попытке входа на данный адрес ничего не будет видно, потому что не проброшен порт. 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/ (если не открывается на Windows, то вместо 0.0.0.0 нужно указать localhost) (рис. 10) открывается содержимое корневой директории в контейнере (рис. 11).

```
[node1] (local) root@192.168.0.28 ~
$ 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/) ...
172.18.0.1 - - [27/Apr/2023 16:19:05] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [27/Apr/2023 16:19:05] code 404, message File not found
172.18.0.1 - - [27/Apr/2023 16:19:05] "GET /favicon.ico HTTP/1.1" 404 -
```

Рисунок 10 – Запуск сервера с пробросой порта

Рисунок 11 – Просмотр содержимого каталога

1.4 Именованные контейнеры, остановка и удаление

Для запуска контейнера в фоновом режиме используется флаг –d. Для задания имени используется флаг –name. Для проверки запуска используется команда docker ps | grep servername. Для просмотра истории действий docker logs servername. Для остановки – docker stop servername. Для удаления из списка – docker rm servername.

Демонстрация выполнения команд представлена на рисунке (рис. 11 – 11.1).

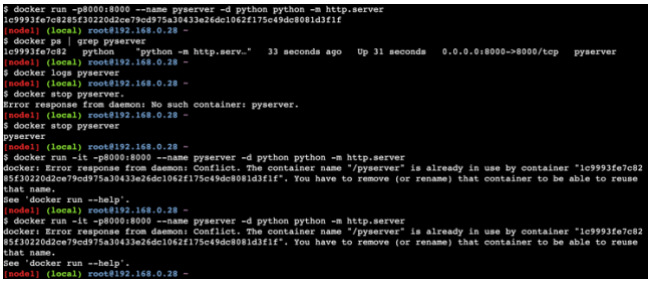

Рисунок 11 – Запуск контейнера в фоновом режиме с установленным именем

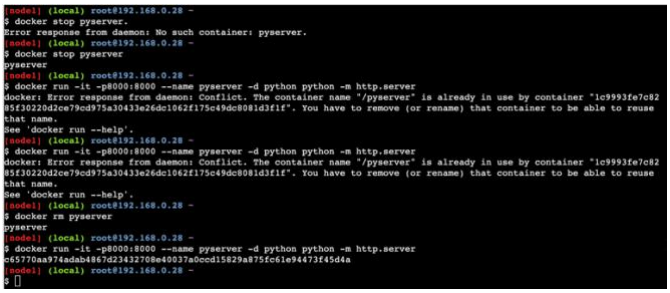

Рисунок 11.1 – Запуск контейнера в фоновом режиме с установленным именем

1.5 Постоянное хранение данных

Далее на рисунках (рис. 12-13.1) демонстрируется запуск контейнера, в котором веб-сервер будет отдавать содержимое директории /mnt, создание файла hi.txt, просмотр содержимого и остановка сервера.

```
[node1] (local) root@192.168.0.28 ~
$ docker run -p8000:8000 --name pyserver --rm -d python python -m http.server -d /mnt
298b8baac5412793629a1f649a252acc5fc0ddf852f0afa4416d39c4ddb7f613
[node1] (local) root@192.168.0.28 ~
$ docker exec -it pyserver bash
root@298b8baac541:/# cd mnt && echo "hello world" > hi.txt
root@298b8baac541:/mnt# exit
exit
[node1] (local) root@192.168.0.28 ~
$ docker stop pyserver
pyserver
[node1] (local) root@192.168.0.28 ~
$
```

Рисунок 12 – Запуск контейнера, создание файла и остановка контейнера

Directory listing for /

- hi.txt

Рисунок 13 – Просмотр содержимого контейнера

```
hello world
```

Рисунок 13.1 – Просмотр содержимого контейнера

1.5.1 Тома

После повторного запуска контейнера и просмотра содержимого мы обнаружим, что файл hi.txt пропал.

Для сохранения данных есть несколько способов. Первый – создание отдельного тома с помощью ключа –v. В таком случае даже после удаления контейнера созданные файлы будут сохраняться (рис. 14).

```
[node1] (local) root@192.168.0.13 ~
$ docker run -p8000:8000 --rm --name pyserver -d \
-v $(pwd)/myfiles:/mnt python python -m http.server -d /mnt
7cee0a3976214c5cba1c1f6979741f380314a324e009a52f4f2c2d324725f9ac
[node1] (local) root@192.168.0.13 ~
$ docker exec -it pyserver bash
root@7cee0a397621:/# cd mnt && echo "hello world" > hi.txt
root@7cee0a397621:/mnt# exit
exit
[node1] (local) root@192.168.0.13 ~
$ docker inspect -f "{{json .Mounts }}" pyserver
[{"Type":"bind","Source":"/root/myfiles","Destination":"/mnt","Mode":"","RW":true,"Propagation":"rprivate"}]
[node1] (local) root@192.168.0.13 ~
$
```

Рисунок 14 – Монтирование тома

1.5.2 Монтирование директорий и файлов

Второй способ – монтирование директорий и файлов. Для этого создается директория и файлы и монтируются при запуске контейнера (рис. 15).

```
[node1] (local) root@192.168.0.13 ~
$ docker stop pyserver
pyserver
[node1] (local) root@192.168.0.13 ~
$  mkdir myfiles
mkdir: can't create directory 'myfiles': File exists
[node1] (local) root@192.168.0.13 ~
$ docker exec -it pyserver bash
Error: No such container: pyserver
[node1] (local) root@192.168.0.13 ~
$ mkdir myfiles
mkdir: can't create directory 'myfiles': File exists
[node1] (local) root@192.168.0.13 ~
$ touch myfiles/host.txt
[node1] (local) root@192.168.0.13 ~
$ docker run -p8000:8000 --rm --name pyserver -d -v $(pwd)/myfiles:/mnt python \
python -m http.server -d /mnt
23de528968cd7ba6458313d4b662ae206f21106307d2ede5eb6a98824231d2e5
[node1] (local) root@192.168.0.13 ~
$ pwd
/root
[node1] (local) root@192.168.0.13 ~
$ docker exec -it pyserver bash
root@23de528968cd:/# cd /mnt
root@23de528968cd:/mnt# ls
hi.txt  host.txt
root@23de528968cd:/mnt# echo "hello world" > hi.txt
root@23de528968cd:/mnt# ls
hi.txt  host.txt
root@23de528968cd:/mnt# ls myfiles
ls: cannot access 'myfiles': No such file or directory
```

Рисунок 15 – Монтирование директорий и файлов

1.6 Переменные окружения

Передача переменных окружения внутрь контейнера делается с помощью ключа -e (рис. 16).

```
[node1] (local) root@192.168.0.13 ~
$ docker run -it --rm -e MIREA="ONE LOVE" ubuntu env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=fc5d1ae688cc
TERM=xterm
MIREA=ONE LOVE
HOME=/root
[node1] (local) root@192.168.0.13 ~
$
```

Рисунок 16 – Передача переменных окружения

1.7 Dockerfile

Создание докерфайла (рис. 17).

```
[node1] (local) root@192.168.0.13 ~
$ cat >> 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 ["python", "-m", "http.server", "-d", "/mnt/", "80"]
^C
```

Рисунок 17 – Создание докерфайла

Сборка образа, запуск и просмотр содержимого контейнера представлены на рисунках (рис. 18-19).

```
[node1] (local) root@192.168.0.18 ~
$ docker build -t mycoolimage .
Sending build context to Docker daemon   12.8kB
Step 1/7 : FROM ubuntu:20.04
 ---> 88bd68917189
Step 2/7 : RUN apt update     && apt install -y postgresql-client
 ---> Using cache
 ---> 15fbe1575141
Step 3/7 : RUN /usr/games/fortune > /mnt/greeting-while-building.txt
 ---> Using cache
 ---> 0fc186dbba18
Step 4/7 : ADD /data /mnt/files
 ---> Using cache
 ---> b505fd4e97c4
Step 5/7 : VOLUME /mnt/files
 ---> Using cache
 ---> 76a62d7e4fee
Step 6/7 : EXPOSE 8
 ---> Using cache
 ---> 5be807fa3070
Step 7/7 : CMD ["python3", "-m", "http.server", "-d", "/mnt/", "8"]
 ---> Using cache
 ---> 28093c07057f
Successfully built 28093c07057f
Successfully tagged mycoolimage:latest
```

Рисунок 18 – Сборка образа и запуск контейнера

Directory listing for /

- data/
- greeting-while-building.txt

Рисунок 19 – Просмотр содержимого контейнера

1.8 Индивидуальные задания

Персональный вариант: 2.

Создадим директорию data, содержащий файл student.txt (Рисунок 20-21).

Рисунок 20 – Новый директорий и файлы

```
[node1] (local) root@192.168.0.13 ~
$ mkdir data
```

Рисунок 20 – Создание директорию data

```
[node1] (local) root@192.168.0.13 ~
$ echo "Daurbecov Magomed Ibragimov, ikbo-04-21, variant - 2" > /data/student.txt
```

Рисунок 21 – Создание и заполнение файла student.txt

Выполнение индивидуального задания (рис. 22-25).

student.txt × Dockerfile ×

Save Reload

```
1  FROM ubuntu:20.04
2
3  RUN apt update \
4      && apt install -y figlet  \
5      && apt install -y python3 fortune \
6      && ln -s python3 puthon3
7
8  RUN /usr/games/fortune > /mnt/greeting-while-building.txt
9
10 ADD /data /mnt/files
11 VOLUME /mnt/files
12 EXPOSE 6
13
14 CMD ["python3", "-m", "http.server", "-d", "/mnt/", "6"]
```

Рисунок 22 – Создание dockerfile по заданию

```
Sending build context to Docker daemon   12.8kB
Step 1/7 : FROM ubuntu:20.04
 ---> 88bd68917189
Step 2/7 : RUN apt update     && apt install -y figlet      && apt install -y python3 fortune     && ln -s python3 puthon
3
 ---> Using cache
 ---> d0a6fb31f3b8
Step 3/7 : RUN /usr/games/fortune > /mnt/greeting-while-building.txt
 ---> Using cache
 ---> 69e09f2d9ed6
Step 4/7 : ADD /data /mnt/files
 ---> Using cache
 ---> 8a6a349c373f
Step 5/7 : VOLUME /mnt/files
 ---> Using cache
 ---> 38e9f5ece7f1
Step 6/7 : EXPOSE 6
 ---> Using cache
 ---> 0882161919b7
Step 7/7 : CMD ["python3", "-m", "http.server", "-d", "/mnt/", "6"]
 ---> Using cache
 ---> 07231fcb33db
Successfully built 07231fcb33db
Successfully tagged mycoolimage:latest
```

Рисунок 23 – Сборка образа

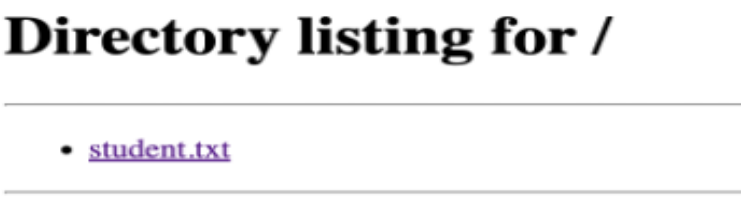

Рисунок 24 – Просмотр содержимого контейнера

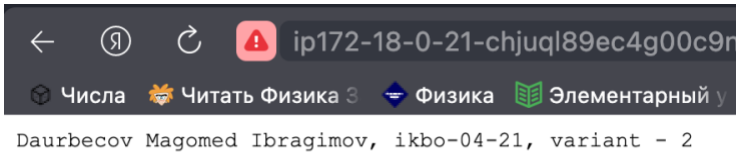

Рисунок 25 – Просмотр содержимого файла

Вывод

В ходе работы были получены навыки работы с Docker:

- Установка Docker;
- Скачивание и использование образов;
- Работа с контейнерами;
- Работа с переменными окружениями;
- Создание Dockerfile.

Список информационных источников

1. Методические указания по выполнению практической работы №3: Docke, МИРЭА – Российский технологический университет, 2023.
2. Docker – [Электронный ресурс] URL: https://www.docker.com/