

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»  
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ  
Кафедра компьютерной инженерии и моделирования

**ОТЧЕТ ПО ПРАКТИЧЕСКОМУ ЗАДАНИЮ №5**  
**«Основы контейнеризации»**

Практическая работа  
по дисциплине «Современные технологии программирования»  
студента 1 курса группы ПИ-б-о-233  
Иващенко Дениса Олеговича  
направления подготовки 09.03.04 «Программная инженерия»

Симферополь, 2024

**Цель:** ознакомиться с базовыми возможностями Docker и Podman. Приобрести опыт и навыки контейнеризации приложений.

**Ход выполнения работы:**

### Основы Docker

1) Для начала нужно определить название сетевого интерфейса через который машина выходит в интернет. В нашем случае этому интерфейсу выдаётся ip-адрес роутером (т.к. сеть VirtualBox в режиме сетевого моста). Введите команду `ip a` и найдите название интерфейса:

```
ivd@ivd:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether ba:d7:5c:0f:b0:6b brd ff:ff:ff:ff:ff:ff
    inet 192.168.89.77/24 metric 100 brd 192.168.89.255 scope global dynamic enp0s1
        valid_lft 338sec preferred_lft 338sec
    inet6 fe80::b8d7:5cff:fe0f:b06b/64 scope link
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:3c:50:5a:e3 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
```

2) Чтобы установить для данного интерфейса dns можно воспользоваться 2-мя вариантами (выберите любой):

- Временный вариант, он будет отменятся после перезагрузки сервера. Нужно ввести команду:  
`sudo resolvectl dns enp0s3 8.8.8.8`

Здесь мы добавляем в список dns-серверов dns-сервер google. Чтобы проверить, что команда выполнена успешно введите:  
`sudo resolvectl dns enp0s3`

В результате вы должны увидеть список dns-серверов для интерфейса "enp0s3".

```
ivd@ivd:~$ sudo resolvectl dns enp0s1 8.8.8.8
[sudo] password for ivd:
ivd@ivd:~$ sudo resolvectl dns enp0s1
Link 2 (enp0s1): 8.8.8.8 192.168.89.1 185.112.140.5 185.112.140.7
ivd@ivd:~$
```

3) Для удобства дальнейшей работы, подключитесь к машине по ssh;

```
denisivaschenko — ivd@ivd: ~ — ssh ivd@192.168.89.77 — 80x24

System load:  0.0          Processes:      123
Usage of /:   23.9% of 29.82GB Users logged in: 1
Memory usage: 49%         IPv4 address for docker0: 172.17.0.1
Swap usage:   0%          IPv4 address for enp0s1: 192.168.89.77

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
  just raised the bar for easy, resilient and secure K8s cluster deployment

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

3 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Sat May 18 08:56:31 2024
ivd@ivd:~$
```

4) Обновите индексы пакетов: `sudo apt-get update`.

```
Last login: Sat May 18 08:56:31 2024
ivd@ivd:~$ sudo apt-get update
[[sudo] password for ivd:
Hit:1 http://ports.ubuntu.com/ubuntu-ports
Get:2 https://download.docker.com/linux/ubuntu
Get:3 http://ports.ubuntu.com/ubuntu-ports
Get:4 https://download.docker.com/linux/ubuntu
kB]
Ign:5 https://pkg.jenkins.io/debian-stable
Get:6 https://pkg.jenkins.io/debian-stable
Get:7 https://pkg.jenkins.io/debian-stable
Get:8 http://ports.ubuntu.com/ubuntu-ports
Get:9 http://ports.ubuntu.com/ubuntu-ports
```

5) Устанавливать docker и docker compose будем из официальных репозитория Docker.

```
Context:          default

Server: Docker Engine - Community
Engine:
  Version:        26.1.3
  API version:    1.45 (minimum version 1.24)
  Go version:     go1.21.10
  Git commit:     8e96db1
  Built:          Thu May 16 08:39:57 2024
  OS/Arch:        linux/arm64
  Experimental:   false
containerd:
  Version:        1.6.31
  GitCommit:      e377cd56a71523140ca6ae87e30244719194a521
runc:
  Version:        1.1.12
  GitCommit:      v1.1.12-0-g51d5e94
docker-init:
  Version:        0.19.0
  GitCommit:      de40ad0
Docker Compose version v2.27.0
[ivd@ivd:~$ docker compose version
Docker Compose version v2.27.0
ivd@ivd:~$
```

6) Выполните команду:

docker —help

```
[ivd@ivd:~$ docker build --help
Start a build

Usage:  docker buildx build [OPTIONS] PATH | URL | -

Start a build

Aliases:
  docker buildx build, docker buildx b
```

7)Выполните команду:

```
docker image ls
```

Данная команда позволяет посмотреть список образовав которые есть у вас на локальной машине. Сейчас список должен быть пустой.

```
[ivd@ivd:~$ docker image ls
REPOSITORY          TAG          IMAGE ID      CREATED       SIZE
chameleoncode/qrcode a400d78     23dc8ed4f23b 3 weeks ago   94.1MB
qrcode              latest      23dc8ed4f23b 3 weeks ago   94.1MB
```

8)Выполните команду:

```
docker run hello-world
```

```
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (arm64v8)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

9)Ещё раз проверьте список локальных образов. Теперь там должен быть один образ - тот самый "hello-world".

```
[ivd@ivd:~$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
chameleoncode/qrcode	a400d78	23dc8ed4f23b	3 weeks ago	94.1MB
qrcode	latest	23dc8ed4f23b	3 weeks ago	94.1MB
hello-world	latest	ee301c921b8a	12 months ago	9.14kB

```
ivd@ivd:~$
```

10)Команда `run` запустила контейнер, который отработал, вывел на экран приветственное сообщение и тут же завершит свою работу (т.е. виртуальная машина выключилась). Чтобы проверить список контейнеров, которые когда либо были запущены воспользуйтесь командой:

`docker container ls -a`

```
ivd@ivd:~$ docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
373cd7f5ecab	hello-world	"/hello"	About a minute ago	Exited (0) About a minute ago
		busy_wilson		

Изучите информацию о контейнерах. Сейчас у вас в списке должен быть только один контейнер и в столбце STATUS должно значится Exited (0), т.е. контейнер остановился и его главный процесс вернул код ошибки 0.

Более короткий вариант команды: `docker ps -a`.

11)Запустите ещё несколько контейнеров из образа "hello-world" и изучите как изменился список контейнеров.

Теперь docker больше не будет скачивать образ, т.к. он уже есть на машине.

```
ivd@ivd:~$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
chameleoncode/qrcode	a400d78	23dc8ed4f23b	3 weeks ago	94.1MB
qrcode	latest	23dc8ed4f23b	3 weeks ago	94.1MB
hello-world	latest	ee301c921b8a	12 months ago	9.14kB

12)Посмотрите список контейнеров, но в этот раз добавьте к команде опцию -s. В результате у вас появится дополнительный столбец SIZE показывающий размер занимаемый контейнером на диске.

Как видно, собственный размер контейнеров 0 Байт, а значение указанное в скобочках - это размер образа на котором основаны контейнеры (образ один для всех наших контейнеров). Реально контейнер хранит только ту информацию, которая отличается в нём от базового образа.

```
ivd@ivd:~$ docker container ls -s
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES	SIZE
...	...	...	...	...	...	...	...

13)В результате работы у вас может накопиться довольно большое количество контейнеров. И хотя каждый из них занимает мало места на диске полезно уметь их удалять.

Введите команду и удалите один контейнер:

`docker rm {CONTAINER ID | CONTAINER NAME}`

```
ivd@ivd:~$ docker rm {CONTAINER ID | CONTAINER NAME}
Error response from daemon: No such container: {CONTAINER
Error response from daemon: No such container: ID
CONTAINER: command not found
ivd@ivd:~$
```

14)Когда образов накапливается очень много, то перечислять все их имена или id в команде rm не очень удобно, поэтому, все *остановленные* контейнеры можно удалить командой:

`docker container prune`

Удалите все оставшиеся контейнеры.

15) Образ "hello-world" нам больше не понадобится, поэтому удалите его командой:

```
docker rmi hello-world
```

```
ivd@ivd:~$ docker rmi hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:266b191e926f65542fa8daaec01a192c4d292bff79426f47300a046e1bc576fd
Deleted: sha256:ee301c921b8aad002973b2e0c3da17d701dcd994b606769a7e6eaa100b81d44
Deleted: sha256:12660636fe55438cc3ae7424da7ac56e845cdb52493ff9cf949c47a7f57f8b43
ivd@ivd:~$
```

16) Выполните команду:

```
docker pull nginx
```

Эта команда скачает образ nginx с DockerHub без запуска контейнера.

```
ivd@ivd:~$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
24c63b8dcb66: Pull complete
ac894f1d1dfb: Pull complete
2572d4eb2260: Pull complete
0ac3805c647c: Pull complete
da20f09652a8: Pull complete
2de21a3abd85: Pull complete
77cea143f3c3: Pull complete
Digest: sha256:a484819eb60211f5299034ac80f6a681b06f89e65866ce91f356ed7c72af059c
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

17) апустите новый контейнер из образа "nginx" при помощи команды run без параметров.

Эта команда запустит контейнер в режиме по умолчанию, который называется `attached mode`. Это означает, что стандартный ввод (stdin), стандартный вывод



(stdout) и стандартный вывод ошибок (stderr) контейнера подключены к сеансу текущей оболочки. Т.е. любой вывод из контейнера немедленно выводится на наш терминал, а любой ввод с терминала отправляется в контейнер. Однако мы не можем полноценно взаимодействовать с контейнером, потому что внутри контейнера не запущена командная оболочка.

```
[ivd@ivd:~$ docker run nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform c
onfiguration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.s
h
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/def
ault.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/d
efault.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/05/18 09:48:28 [notice] 1#1: using the "epoll" event method
2024/05/18 09:48:28 [notice] 1#1: nginx/1.25.5
2024/05/18 09:48:28 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/05/18 09:48:28 [notice] 1#1: OS: Linux 5.15.0-105-generic
2024/05/18 09:48:28 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/05/18 09:48:28 [notice] 1#1: start worker processes
2024/05/18 09:48:28 [notice] 1#1: start worker process 29
2024/05/18 09:48:28 [notice] 1#1: start worker process 30
```

18) Т.к. весь наш ввод отправляется к контейнеру, то мы не можем работать с хостовой системой. Остановите контейнер при помощи комбинации клавиш Ctrl + C. Эта комбинация завершит основной процесс контейнера и он автоматически остановится.

```
^C2024/05/18 09:48:53 [notice] 1#1: signal 2 (SIGINT) received, exiting
2024/05/18 09:48:53 [notice] 29#29: exiting
2024/05/18 09:48:53 [notice] 32#32: exiting
2024/05/18 09:48:53 [notice] 29#29: exit
2024/05/18 09:48:53 [notice] 32#32: exit
2024/05/18 09:48:53 [notice] 30#30: exiting
2024/05/18 09:48:53 [notice] 30#30: exit
2024/05/18 09:48:53 [notice] 31#31: exiting
2024/05/18 09:48:53 [notice] 31#31: exit
2024/05/18 09:48:53 [notice] 1#1: signal 17 (SIGCHLD) received from 30
2024/05/18 09:48:53 [notice] 1#1: worker process 30 exited with code 0
2024/05/18 09:48:53 [notice] 1#1: signal 29 (SIGIO) received
2024/05/18 09:48:53 [notice] 1#1: signal 17 (SIGCHLD) received from 32
2024/05/18 09:48:53 [notice] 1#1: worker process 31 exited with code 0
2024/05/18 09:48:53 [notice] 1#1: worker process 32 exited with code 0
2024/05/18 09:48:53 [notice] 1#1: signal 29 (SIGIO) received
2024/05/18 09:48:53 [notice] 1#1: signal 17 (SIGCHLD) received from 29
2024/05/18 09:48:53 [notice] 1#1: worker process 29 exited with code 0
2024/05/18 09:48:53 [notice] 1#1: exit
ivd@ivd:~$
```

19) Выполните у себя в терминале команду `tty`.

В результате вы должны увидеть, путь к виртуальному устройству с которым ассоциирован терминал.

```
[ivd@ivd:~$ tty
/dev/pts/0
```

20) Контейнер можно попросить запустить внутри себя команду указав саму команду и её аргументы после имени образа. Попросим его выполнить команду `tty`:

`docker run nginx tty`

```
[ivd@ivd:~$ docker run nginx tty
not a tty
```

21) Чтобы решить проблему захвата ввода-вывода нашего терминала контейнером запустите новый контейнер из образа "nginx" при помощи команды:

`docker run -d nginx`

```
[ivd@ivd:~$ docker run -d nginx
475b80e47529a0fddeac7bc8a641f3d50bf12a86d7614cb7e533c1eb03695ad2
```

22) Выполните команду:

`docker ps`

```
[ivd@ivd:~$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS   NAMES
475b80e47529   nginx     "/docker-entrypoint..." 39 seconds ago Up 38 seconds 80/tcp   lucid_proskuriakova
```

23) Работающий контейнер можно остановить командой `docker stop` и заново запустить командой `docker start`. Так же как и при удалении нужно

указать CONTAINER ID или CONTAINER NAME целевого контейнера.

Остановите контейнер используя его id.

Обратите внимание, команда run не используется для запуска остановленных контейнеров, она создаёт новый.

```
[ivd@ivd:~$ docker stop 475b80e47529  
475b80e47529
```

24) Последний, используемый на практике, способ запуска контейнера - интерактивный режим. В этом режиме внутри контейнера создаётся сеанс оболочки (shell session), который позволяет взаимодействовать с ним напрямую через терминал. Т.е. мы как-бы попадаем внутрь контейнера и работаем там.

Выполните команду:

`docker run -it nginx`

```
[ivd@ivd:~$ docker run -it nginx  
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration  
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/  
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh  
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf  
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf  
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh  
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh  
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh  
/docker-entrypoint.sh: Configuration complete; ready for start up  
2024/05/18 09:56:53 [notice] 1#1: using the "epoll" event method  
2024/05/18 09:56:53 [notice] 1#1: nginx/1.25.5  
2024/05/18 09:56:53 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)  
2024/05/18 09:56:53 [notice] 1#1: OS: Linux 5.15.0-105-generic  
2024/05/18 09:56:53 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576  
2024/05/18 09:56:53 [notice] 1#1: start worker processes  
2024/05/18 09:56:53 [notice] 1#1: start worker process 30  
2024/05/18 09:56:53 [notice] 1#1: start worker process 31  
2024/05/18 09:56:53 [notice] 1#1: start worker process 32  
2024/05/18 09:56:53 [notice] 1#1: start worker process 33
```

25) Для данного контейнера вы получите такую же самую "картину" как и для запуска без параметров, но для других контейнеров отличия будут (увидим далее). Остановите контейнер.

```
^C2024/05/18 09:57:21 [notice] 33#33: signal 2 (SIGINT) received, exiting
2024/05/18 09:57:21 [notice] 32#32: signal 2 (SIGINT) received, exiting
2024/05/18 09:57:21 [notice] 1#1: signal 2 (SIGINT) received, exiting
2024/05/18 09:57:21 [notice] 33#33: exiting
2024/05/18 09:57:21 [notice] 32#32: exiting
2024/05/18 09:57:21 [notice] 31#31: signal 2 (SIGINT) received, exiting
2024/05/18 09:57:21 [notice] 30#30: signal 2 (SIGINT) received, exiting
2024/05/18 09:57:21 [notice] 30#30: exiting
2024/05/18 09:57:21 [notice] 33#33: exit
2024/05/18 09:57:21 [notice] 30#30: exit
2024/05/18 09:57:21 [notice] 32#32: exit
2024/05/18 09:57:21 [notice] 31#31: exiting
2024/05/18 09:57:21 [notice] 31#31: exit
2024/05/18 09:57:21 [notice] 1#1: signal 17 (SIGCHLD) received from 32
2024/05/18 09:57:21 [notice] 1#1: worker process 32 exited with code 0
2024/05/18 09:57:21 [notice] 1#1: worker process 33 exited with code 0
2024/05/18 09:57:21 [notice] 1#1: signal 29 (SIGIO) received
2024/05/18 09:57:21 [notice] 1#1: signal 17 (SIGCHLD) received from 30
2024/05/18 09:57:21 [notice] 1#1: worker process 30 exited with code 0
2024/05/18 09:57:21 [notice] 1#1: signal 29 (SIGIO) received
2024/05/18 09:57:21 [notice] 1#1: signal 17 (SIGCHLD) received from 31
2024/05/18 09:57:21 [notice] 1#1: worker process 31 exited with code 0
2024/05/18 09:57:21 [notice] 1#1: exit
```

26) Убедимся, что при запуске контейнера в интерактивном режиме в нём есть терминал.

Выполните команду:

`docker run -it nginx tty`

```
[ivd@ivd:~$ docker run -it nginx tty
/dev/pts/0
ivd@ivd:~$
```

27) Удалите все созданные контейнеры.

28) В процессе работы у нас постоянно накапливаются ненужные контейнеры и их приходится постоянно удалять. Чтобы попросить контейнер автоматически удалиться после остановки, можно добавить к команде запуска опцию `--rm`.

Запустите контейнер:

`docker run --rm -d nginx`

```
[ivd@ivd:~$ docker run --rm -d nginx
0501173a08bdc986dd9cd415ba2a95b5fe6e12ecd1863f4e0d1a3a0cc1b2183f
```

29) Теперь остановите его при помощи команды `docker stop` и `id` контейнера, а затем проверьте список всех контейнеров. Он должен быть пуст.

```
[ivd@ivd:~$ docker stop 0501173a08bdc986  
0501173a08bdc986
```

30) Запустите новый контейнер nginx:

`docker run --rm -d nginx`

```
[ivd@ivd:~$ docker run --rm -d nginx  
4522815ba9b3303c1c372a2ee708f8db7d85187b958293850bfb02de826e9867
```

31) Выполните команду:

`docker exec -it {CONTAINER ID | CONTAINER NAME} bash`

```
[ivd@ivd:~$ docker exec -it 4522815ba bash  
root@4522815ba9b3:/#
```

32) Перейдите в каталог `"/usr/share/nginx/html"` и посмотрите его содержимое.

Там должно быть 2 файла, среди которых `"index.html"`.

```
root@4522815ba9b3:/# cd usr  
root@4522815ba9b3:/usr# cd share  
root@4522815ba9b3:/usr/share# cd nginx  
root@4522815ba9b3:/usr/share/nginx# cd html  
root@4522815ba9b3:/usr/share/nginx/html# ls  
50x.html  index.html  
root@4522815ba9b3:/usr/share/nginx/html#
```

33) Попробуйте открыть его при помощи текстового редактора nano.

Вы обнаружите, что этот редактор не установлен в контейнере.

```
root@4522815ba9b3:/usr/share/nginx/html# sudo nano index.html  
bash: sudo: command not found  
root@4522815ba9b3:/usr/share/nginx/html# nano index.html  
bash: nano: command not found  
root@4522815ba9b3:/usr/share/nginx/html#
```

34)Обновите индексы пакетов apt и установите nano (название пакета такое же).

35)Проверьте, что теперь файл успешно открывается.

```
[root@4522815ba9b3:/usr/share/nginx/html# sudo nano index.html
```

36)Отключитесь от контейнера при помощи exit.

Контейнер продолжит работать дальше и при необходимости мы в любое время снова сможем к нему подключиться.

Команда exec не требует обязательного входа в контейнер для запуска команд, например:

```
docker exec {CONTAINER ID | CONTAINER NAME} ls -al
```

```
[root@4522815ba9b3:/usr/share/nginx/html# exit  
exit
```

37)Ранее мы запускали контейнер с nginx, который, кроме прочего, используется и как web-сервер для простых сайтов. Запустите новый контейнер в detach mode и попробуйте получить от него ответ на GET-запрос (nginx прослушивает 80 порт):

```
curl 127.0.0.1:80
```

```
[ivd@ivd:~$ curl 127.0.0.1:80  
curl: (7) Failed to connect to 127.0.0.1 port 80 after 0 ms: Connection refused
```

39)Остановите контейнер и запустите новый, командой:

```
docker run -d --network=host nginx
```



40) Попробуйте запустить ещё один контейнер той же командой что и выше (останавливать первый не нужно!), а затем проверьте список *запущенных* контейнеров.

Вы должны обнаружить, что запущен по прежнему только первый.

```
[ivd@ivd:~$ docker run -d --network=host nginx
2ed717ed60743bdd01d770bde073d5b5fa3fde4f3d009fca167e9c52d2af9553
[ivd@ivd:~$ docker run -d --network=host nginx
c18d76c1a1d52787dfdc7a5f49f1da6d0464c8d03a11e7697f5095b5b863faa
```

41) Т.к. мы запустили второй контейнер в фоновом режиме, то никаких сообщений кроме его id на экране не появилось. Чтобы понять в чём проблема воспользуемся командой:

`docker logs {CONTAINER ID | CONTAINER NAME}`

```
[ivd@ivd:~$ docker logs 2ed717ed60743bdd01d770bde073d5b5fa3fde4f3d009fca167e9c52d2af9553
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/05/18 10:13:47 [notice] 1#1: using the "epoll" event method
2024/05/18 10:13:47 [notice] 1#1: nginx/1.25.5
2024/05/18 10:13:47 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/05/18 10:13:47 [notice] 1#1: OS: Linux 5.15.0-105-generic
2024/05/18 10:13:47 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/05/18 10:13:47 [notice] 1#1: start worker processes
2024/05/18 10:13:47 [notice] 1#1: start worker process 29
2024/05/18 10:13:47 [notice] 1#1: start worker process 30
2024/05/18 10:13:47 [notice] 1#1: start worker process 31
2024/05/18 10:13:47 [notice] 1#1: start worker process 32
```

42) Остановите первый контейнер.

```
[ivd@ivd:~$ docker stop 2ed717ed60743bdd01d770bde073d5b5fa3fde4f3d009fca167e9c52d2af9553
2ed717ed60743bdd01d770bde073d5b5fa3fde4f3d009fca167e9c52d2af9553
[ivd@ivd:~$
```

43) Более безопасным способом попасть внутрь сети контейнера является проброс портов. Введите команду:

`docker run -d -p 8080:80 nginx`

44) Запустите ещё один контейнер nginx, но в качестве порта на хосте укажите 80. Проверьте при помощи curl, что оба контейнера отвечают на запросы (ip остается 127.0.0.1).

```
[ivd@ivd:~]$ docker run -d --network=host nginx
4b6f24996fe6e05baed9860f945c1bd010181aed85dec0fa99948f083877332e
[ivd@ivd:~]$ curl
curl: try 'curl --help' or 'curl --manual' for more information
[ivd@ivd:~]$ curl 127.0.0.1:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

45) Откройте браузер в основной ОС (не на виртуальной машине). Введите в строке адреса ip виртуальной машины (тот который вы использовали для ssh-подключения).

В результате вы должны увидеть приветственную страницу "Welcome to nginx!", что свидетельствует о том, что трафик по 80-му порту попадает через виртуальную машину в docker-контейнер и обратно.



# Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](http://nginx.org).  
Commercial support is available at [nginx.com](http://nginx.com).

*Thank you for using nginx.*

46) Остановите все запущенные контейнеры.

47) Для того, чтобы проверить какие сети уже созданы docker-ом введите команду:

`docker network ls`

```
[ivd@ivd:~$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
85d468ff9bc5    bridge    bridge       local
481d93e7c9cf    host      host         local
af2ad5ca318f    none      null         local
```

48) Запустите ещё одно окно с ssh-подключением к серверу (в kitty можно щёлкнуть правой кнопкой мыши по окну и выбрать "Duplicate Session") и в каждом окне запустите по одному контейнеру "avenga/net-tools":

`docker run -it --name=one avenga/net-tools`

`docker run -it --name=two avenga/net-tools`

```
[ivd@ivd:~$ docker run -it --name=two avenga/net-tools
```

49)Выполните внутри контейнера one команду ip a.

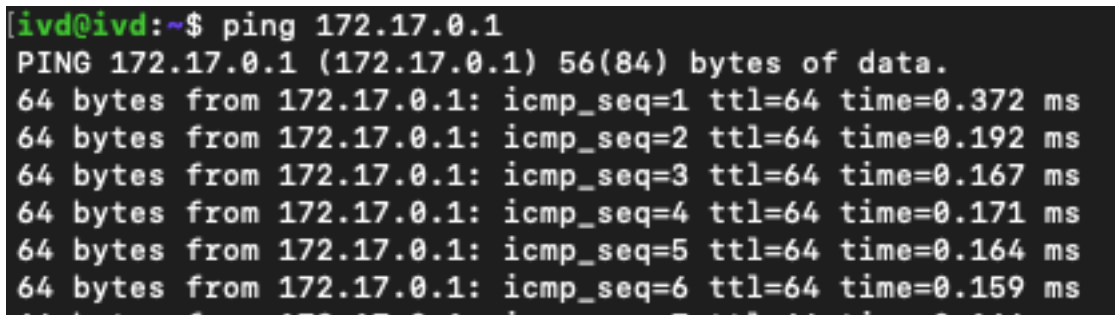
50)Повторите команду в контейнере two.

Как видно вывод отличается только ip-адресом (у меня он 172.17.0.3).

51)Как говорилось ранее эти контейнеры подключены к сети по умолчанию (bridge), а значит они должны "видеть" друг друга. Пропингуйте второй контейнер из первого:

ping 172.17.0.3

Если пинги идут, значит всё хорошо.



```
ivd@ivd:~$ ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1) 56(84) bytes of data.
64 bytes from 172.17.0.1: icmp_seq=1 ttl=64 time=0.372 ms
64 bytes from 172.17.0.1: icmp_seq=2 ttl=64 time=0.192 ms
64 bytes from 172.17.0.1: icmp_seq=3 ttl=64 time=0.167 ms
64 bytes from 172.17.0.1: icmp_seq=4 ttl=64 time=0.171 ms
64 bytes from 172.17.0.1: icmp_seq=5 ttl=64 time=0.164 ms
64 bytes from 172.17.0.1: icmp_seq=6 ttl=64 time=0.159 ms
```

55)Запустите новый контейнер в отдельной ssh-сессии:

docker run -it --name=three --link=one --link=two:alias\_for\_two avenga/net-tools

56)Получите список переменных окружения контейнера three:

Env

```
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:
=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ov
4:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=0
31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.
z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.
=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:
tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01
1:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cp
01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.
pg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp
1;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.
iff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=0
35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.
ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01
5:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.av
01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.
m=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36
*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc
0;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*
```

57)Посмотрите содержимое файла /etc/hosts.

58)Проверьте, что в контейнере three работает ping по имени контейнера one:  
ping one

```
[ivd@ivd:~]$ ping two
```

59)Перейдите в окно подключённое к контейнеру one и попробуйте пропинговать контейнер three в обратном направлении, сначала по ip, а затем по имени.

Пинг по имени сработать не должен, т.к. опция --link подействовала только на контейнер three.

1.

```
[ivd@ivd:~]$ ping three
```

60)Остановите и удалите все три контейнера. Для удаления используйте их имена.

Удалять контейнеры - обязательно, т.к. далее мы будем использовать имена one, two, three для новых контейнеров. Если будут существовать контейнеры с этими именами (даже не работающие) docker не позволит это сделать.

```
[ivd@ivd:~$ docker stop three
three
[ivd@ivd:~$ docker rm three
three
```

61) Более продвинутый вариант позволяющий использовать имена контейнеров как их доменные имена в сети заключается в работе через встроенный dns docker-a. Но он не работает в сети по умолчанию, поэтому нужно будет создать свою.

62) Введите команду:

`docker network create sky_net`

```
[ivd@ivd:~$ docker network create sky_net
506006663f335a22e9d2e2d5141536410a004663378dab7384fb98ecf7ada46d
ivd@ivd:~$
```

63) Проверьте, что сеть была успешно создана и присутствует в общем списке сетей.

Как видно, в качестве драйвера был выбран драйвер bridge.

```
[ivd@ivd:~$ docker network ls
NETWORK ID        NAME        DRIVER        SCOPE
85d468ff9bc5     bridge     bridge        local
481d93e7c9cf     host       host          local
af2ad5ca318f     none       null          local
506006663f33     sky_net    bridge        local
```

64) Запустите контейнер командой:

`docker run -it --name=one --network=sky_net avenga/net-tools`

```
[ivd@ivd:~$ docker run -it --name=one --network=sky_net avenga/net-tools
```

65) В другом окне запустите контейнер two такой же командой.

```
ivd@ivd:~$ docker run -it --name=two --network=sky_net avenga/net-tools
```

66) В третьем окне запустите контейнер three, но *без подключения* к сети "sky\_net":

`docker run -it --name=three avenga/net-tools`

```
[ivd@ivd:~$ docker run -it --name=three avenga/net-tools
```

67) Проверьте, что контейнер one пингует контейнер two по имени, а контейнер three нет.

```
[ivd@ivd:~$ ping two
```

```
[ivd@ivd:~$ ping three  
ping: three: Temporary failure in name resolution
```

68) Откройте четвёртое окно с ssh-подключением к серверу и введите команду:  
docker network inspect sky\_net

```
[ivd@ivd:~$ docker network inspect sky_net  
[  
  {  
    "Name": "sky_net",  
    "Id": "506006663f335a22e9d2e2d5141536410a004663378dab7384fb98ecf7ada46d"  
  },  
  {  
    "Created": "2024-05-18T10:36:20.012396367Z",  
    "Scope": "local",  
    "Driver": "bridge",  
    "EnableIPv6": false,  
    "IPAM": {  
      "Driver": "default",  
      "Options": {},  
      "Config": [  
        {  
          "Subnet": "172.18.0.0/16",  
          "Gateway": "172.18.0.1"  
        }  
      ]  
    }  
  },  
]
```

69) Найдите раздел "Containers" и убедитесь, что там присутствует информация только о двух подключённых контейнерах с именами one и two.

70) В этом же окне введите команду:  
docker network connect sky\_net three

```
[ivd@ivd:~$ docker network connect sky_net three
```

71)Посмотрите как изменился раздел "Containers" у сети "sky\_net".

72)Перейдите в окно подключённое к контейнеру one и пропингуйте контейнер three. В этот раз пинг должен работать.

```
ivd@ivd:~$ ping three
```

73)Вернитесь в 4-е окно и введите команду:

docker network disconnect sky\_net one

```
ivd@ivd:~$ docker network disconnect sky_net one
ivd@ivd:~$
```

74)Проверьте, что контейнер one больше не пингует контейнеры two и three.

75)Вернитесь в 4-е окно и введите команду:

docker network connect --alias=zero sky\_net one

```
ivd@ivd:~$ docker network connect --alias=zero sky_net one
ivd@ivd:~$
```

76)Перейдите во 2-е окно и проверьте, что контейнер one пингуется по именам "one" и "zero".

```
ivd@ivd:~$ ping zero
```

77) Остановите и удалите все контейнеры. Закройте все окна кроме одного.

78) Введите команду:

```
docker network rm sky_net
```

```
[ivd@ivd:~$ docker network rm sky_net  
sky_net
```

79) Для начала рассмотрим монтирование каталога внутрь контейнера.

80) Создайте в домашнем каталоге пользователя папку "content" и внутри неё файл "index.html" с текстом:

```
<h1>Hello from the local directory</h1>
```

```
[ivd@ivd:~$ mkdir content  
[ivd@ivd:~$ cd content  
[ivd@ivd:~/content$ touch index.html  
[ivd@ivd:~/content$ cat <<END>> index.html  
> <h1>Hello from the local directory</h1>  
> END  
[ivd@ivd:~/content$
```

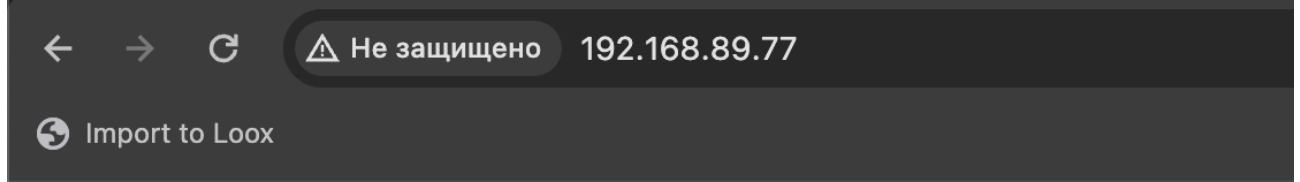
81) Введите команду:

```
docker run --rm --name=web -d -p 80:80 -v ~/content:/usr/share/nginx/html nginx
```

```
[ivd@ivd:~$ docker run --rm --name=web -d -p 80:80 -v ~/content:/usr/share/nginx/html nginx  
975be6597ea875543758f06177ddbcbdc620c2cc8a64e7cb1a141a18eb2ce360  
[ivd@ivd:~$
```

82) Откройте браузер в основной ОС и введите IP-виртуальной машины в строку адреса.

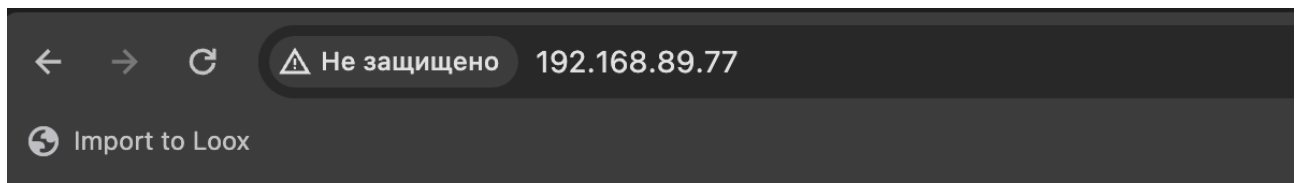
На экране должно появиться сообщение "Hello from the local directory".



# Hello from the local directory

83)Измените содержимое файла "index.html" в локальной папке "content" на следующее:

`<h1 style="color: red">Hello from the local directory</h1>`



# Hello from the local directory

85)Для начала посмотрите список созданных томов:

`docker volume ls`

```
ivd@ivd:~$ docker volume ls
DRIVER      VOLUME NAME
```



86)Создайте новый том командой:

`docker volume create data`

```
[ivd@ivd:~$ docker volume create data
data
```

87)Выполните команду:

`docker volume inspect data`

```
[ivd@ivd:~$ docker volume inspect data
[
  {
    "CreatedAt": "2024-05-18T11:44:07Z",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/data/_data",
    "Name": "data",
    "Options": null,
    "Scope": "local"
  }
]
```

88)Остановите предыдущий контейнер "web" и запустите новый, командой:

`docker run --rm --name=web -d -p 80:80 -v data:/usr/share/nginx/html:ro nginx`

```
[ivd@ivd:~$ docker run --rm --name=web -d -p 80:80 -v data:/usr/share/nginx/html:ro nginx
```

89)Введите команду:

`docker run -d -v data:/root/site -p 6080:6080 -p 5901:5901 -e PASSWORD=123456  
beneventsur/xubuntu:vscode-1.44.2`

```
[ivd@ivd:~$ docker run -d -v data:/root/site -p 6080:6080 -p 5901:5901 -e PASSWORD=123456 beneventsur/xubuntu:vscode-1.44.2
Unable to find image 'beneventsur/xubuntu:vscode-1.44.2' locally
vscode-1.44.2: Pulling from beneventsur/xubuntu
5bed26d33875: Pull complete
f11b29a9c730: Pull complete
930bda195c84: Pull complete
78bf9a5ad49e: Pull complete
2b4e130f1a45: Pull complete
ec89238d8b31: Pull complete
923eeeb914a7: Pull complete
f9815cee44df: Pull complete
ab6046805c0d: Pull complete
7b2fe5d9e8ed: Pull complete
b9ac4339e27f: Pull complete
492e900d1b4d: Pull complete
2d0ae8e36264: Pull complete
79573d8f23f4: Pull complete
```

90)Перейдите в браузер на основной ОС и введите в адресную строку:

`http://{ip виртуальной машины}:6080/vnc.html`



91)Откройте в браузере ещё одну страницу и введите в адресную строку ip виртуальной машины. В результате вы подключитесь к контейнеру с сервером `nginx` и получите от него приветственную страницу.

93)Закройте вкладки, остановите и удалите контейнеры.

Несмотря, на то, что контейнеры больше не существуют, файл "`index.html`" сохранён в томе "`data`".

94)Выполните команду:

`docker volume rm data`

A screenshot of a terminal window with a dark background. The prompt is 'ivd@ivd:~\$' in green. The command 'docker volume rm data' is entered in white text.

95)Кроме опции `-v` можно, для монтирования, можно воспользоваться опцией `--mount`. В этом случае нужно заменить:

`-v {локальный каталог или том}:{каталог в контейнере}` `--mount type={bind-для каталогов или volume-для томов},source={локальный каталог или том},target={каталог в контейнере}`

96)Монтирование `tmpfs` (похоже на том, только в оперативной памяти, а не на

диске) в данной работе рассмотрено не будет, в виду её специфичности и чтобы не усложнять работу.

97)апустите новый nginx контейнер с именем "web", в фоновом режиме, без монтирования томов и пробросив наружу 80-й порт.

98)Введите команду:

```
docker cp web:/usr/share/nginx/html/. ~/new_site
```

```
ivd@ivd:~$ docker cp web:/usr/share/nginx/html/. ~/new_site
Successfully copied 2.56kB to /home/ivd/new_site
```

99)Проверьте, что в "~/new\_site" появились 2 файла, затем внесите изменения в файл "index.html".

```
ivd@ivd:~$ cd ~/new_site
ivd@ivd:~/new_site$ ls
index.html
ivd@ivd:~/new_site$
```

100)Введите команду:

```
docker cp ~/new_site/. web:/usr/share/nginx/html
```

```
ivd@ivd:~$ docker cp ~/new_site/. web:/usr/share/nginx/html
Successfully copied 2.56kB to web:/usr/share/nginx/html
```

101)Проверьте, что в браузере отображается изменённая страница, а затем остановите и удалите контейнер "web".

```
ivd@ivd:~$ docker stop web
web
ivd@ivd:~$
```

102)В домашнем каталоге создайте каталог tutorial и зайдите в него;

103)Теперь создайте каталог flask и зайдите в него. Тут будет flask-приложение;

104)Проверьте, что в системе установлен интерпретатор python3 и его версию:

python3 --version

```
[ivd@ivd:~$ mkdir tutorial
[ivd@ivd:~$ cd tutorial
[ivd@ivd:~/tutorial$ mkdir flask
[ivd@ivd:~/tutorial$ cd flask
[ivd@ivd:~/tutorial/flask$ python3 --version
Python 3.10.12
ivd@ivd:~/tutorial/flask$
```

105)Создайте виртуальное окружение с именем "venv":

python3 -m venv venv

```
[ivd@ivd:~/tutorial/flask$ python3 -m venv venv
ivd@ivd:~/tutorial/flask$
```

106)Активируйте виртуальное окружение:

source venv/bin/activate

107)Установите пакет Flask с помощью pip:

pip install Flask

```
[ivd@ivd:~/tutorial/flask$ python3 -m venv venv
[ivd@ivd:~/tutorial/flask$ source venv/bin/activate
(venv) ivd@ivd:~/tutorial/flask$ pip install Flask
Collecting Flask
  Downloading flask-3.0.3-py3-none-any.whl (101 kB)
    |----- 101.7/101.7 KB 637.4 kB/s eta 0:00:00
Collecting itsdangerous>=2.1.2
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting blinker>=1.6.2
  Downloading blinker-1.8.2-py3-none-any.whl (9.5 kB)
Collecting Jinja2>=3.1.2
  Downloading jinja2-3.1.4-py3-none-any.whl (133 kB)
    |----- 133.3/133.3 KB 3.2 MB/s eta 0:00:00
Collecting click>=8.1.3
  Downloading click-8.1.7-py3-none-any.whl (97 kB)
    |----- 97.9/97.9 KB 462.1 kB/s eta 0:00:00
Collecting Werkzeug>=3.0.0
  Downloading werkzeug-3.0.3-py3-none-any.whl (227 kB)
    |----- 227.3/227.3 KB 2.4 MB/s eta 0:00:00
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.1.5-cp310-cp310-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (26 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, Flask
Successfully installed Flask-3.0.3 Jinja2-3.1.4 MarkupSafe-2.1.5 Werkzeug-3.0.3 blinker-1.8.2 click-8.1.7 itsdangerou
s-2.2.0
(venv) ivd@ivd:~/tutorial/flask$
```

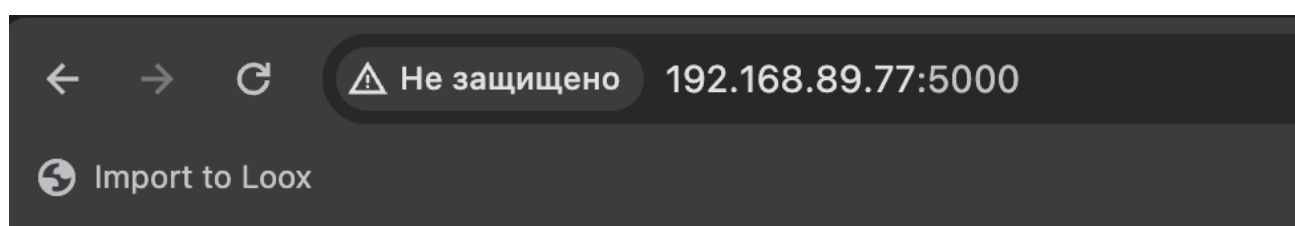
108)Создайте файл с именем app.py содержащий код из документации Flask для создания базового приложения:

```
from flask import Flask

2.
3. app = Flask(__name__)
4.
5. @app.route('/')
6. def hello_world():
7.     return 'Hello, World!'
8.
9. if __name__ == "__main__":
10.     app.run(debug=True, host='0.0.0.0')
```

109)Запустите python app.py и вы должны увидеть запуск сервера.

Перейдите в основную ОС и в браузере укажите ip-адрес виртуальной машины и порт 5000. Вы должны увидеть "Hello, World!»);



Hello, World!

110) Остановите приложение комбинацией клавиш: `Ctrl+C`;

111) Сохраните установленные пакеты (зависимости) в файл `requirements.txt`:

`pip freeze > requirements.txt`

```
[^C(venv) ivd@ivd:~/tutorial/flask$ pip freeze > requirements.txt
(venv) ivd@ivd:~/tutorial/flask$
```

112) Далее можно деактивировать виртуальное окружение командой: `deactivate`.

В принципе можно и не деактивировать, разницы нет. Мы будем работать с `docker`, а он ничего не знает о виртуальных окружениях, созданных командой `venv`.

113) Создайте файл с именем `Dockerfile` (`nano Dockerfile`) и скопируйте в него приведенный ниже код:

```
FROM python:3.10.6
WORKDIR /app
COPY / /app
RUN pip install -r requirements.txt
ENTRYPOINT [ "python" ]
CMD [ "app.py" ]
```

```
(venv) ivd@ivd:~/tutorial/flask$ touch Dockerfile
(venv) ivd@ivd:~/tutorial/flask$ cat <<END>> Dockerfile
> FROM python:3.10.12
WORKDIR /app
COPY / /app
RUN pip install -r requirements.txt
ENTRYPOINT [ "python" ]
CMD [ "app.py" ]
[> END
(venv) ivd@ivd:~/tutorial/flask$
```

114) Запустим сборку образа:

`docker build -t flask .`

```

[(venv) ivd@ivd:~/tutorial/flask$ docker build -t flask .
[+] Building 32.1s (4/8)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 159B
=> [internal] load metadata for docker.io/library/python:3.10.12
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/4] FROM docker.io/library/python:3.10.12@sha256:bac3a0e0d16125977e351c861e2f4b12ecafeaa6f72431dc9
=> => resolve docker.io/library/python:3.10.12@sha256:bac3a0e0d16125977e351c861e2f4b12ecafeaa6f72431dc9
=> => sha256:bac3a0e0d16125977e351c861e2f4b12ecafeaa6f72431dc970d0b9155103232 1.65kB / 1.65kB
=> => sha256:ecafac4bd535d1772f7f39ad71056a6480eb4f65651e6ea33b94aee3e7ac3209 2.01kB / 2.01kB
=> => sha256:715cea74ecbb15cb82efef1e77dd60c31d90b01d1286d6f39b4562afaebe75f3 23.57MB / 23.57MB
=> => sha256:e00e504a45c0c63e2b7061f1c3a1e489f07c77fe1d73693e0fba4c54079de217 7.54kB / 7.54kB
=> => sha256:a014e5e7d08c37cf1703b97e701ccdc850e4a18d0ee679f03aa875dcd520aa85 49.59MB / 49.59MB
=> => sha256:003f1109a21287fa17dc866e87e8c6685113960cbb0379fee8f42b83de63c647 45.09MB / 63.99MB
=> => sha256:a56ae3b61eb9574588be7e73e31c31798e2cbf75f53f1f824d855afdf2be6437 94.37MB / 202.42MB
=> => extracting sha256:a014e5e7d08c37cf1703b97e701ccdc850e4a18d0ee679f03aa875dcd520aa85
=> => sha256:c3668095a3a2c9b08668f4fdc90dde552a6405ce8e99f4b1acd906744092ab4b 6.47MB / 6.47MB
=> => extracting sha256:715cea74ecbb15cb82efef1e77dd60c31d90b01d1286d6f39b4562afaebe75f3
=> => sha256:eba8703016dd18bb6df83c035f5a36ac703d14dcb941c972f18b4fcd73c70880 16.84MB / 16.84MB
=> => sha256:a99285b5e2b909d4fcd532b58cfb59e1b996f170c1a8afa7535bc248afc1d06c 243B / 243B
=> => sha256:df7bb94220c1f8111998d8d602873b056d0db8974d1d331cd5e2615abb7ebae9 3.08MB / 3.08MB
=> [internal] load build context
=> => transferring context: 20.03MB

```

115) После того, как сборка завершилась введите проверьте список образов. В нём должен появиться наш образ flask:latest.

Может быть, что процесс сборки будет прерван из-за ошибки, и в списке появятся ещё несколько "не доделанных" образов с именем "<none>". В этом случае их следует удалить используя команду: `docker rmi` с указанием IMAGE ID удаляемого образа (возможно потребуется добавить ключ `-f`).

```

[(venv) ivd@ivd:~/tutorial/flask$ docker image ls
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
flask                latest          6957aad7d16b   About a minute ago  1.04GB
nginx               latest          8dd77ef2d82e   2 weeks ago     193MB
qrcode              latest          23dc8ed4f23b   3 weeks ago     94.1MB
chameleoncode/qrcode a400d78         23dc8ed4f23b   3 weeks ago     94.1MB
avenga/net-tools    latest          711a1d6e68c3   3 years ago     36.3MB
beneventsur/xubuntu vscode-1.44.2    05224dab2e89   4 years ago     1.41GB

```

116)Проверьте, что приложение работает корректно (через браузер или curl),  
затем остановите его:

`docker run --rm -p 80:5000 flask`

```
[(venv) ivd@ivd:~/tutorial/flask$ docker run --rm -p 80:5000 flask
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production
.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.3:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 993-536-849
192.168.89.22 - - [18/May/2024 12:10:11] "GET / HTTP/1.1" 200 -
192.168.89.22 - - [18/May/2024 12:10:14] "GET / HTTP/1.1" 200 -
^C(venv) ivd@ivd:~/tutorial/flask$
(venv) ivd@ivd:~/tutorial/flask$
```

117)Определите размер образа flask используя `docker image ls`.

Святая корова, он тяжелый! Что-то около 900 МБ для python, работающего с  
приложением Flask? Давайте изменим это.

```
[(venv) ivd@ivd:~/tutorial/flask$ docker image ls
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
flask                latest          6957aad7d16b   4 minutes ago  1.04GB
nginx               latest          8dd77ef2d82e   2 weeks ago    193MB
chameleoncode/qrcode a400d78         23dc8ed4f23b   3 weeks ago    94.1MB
qrcode              latest          23dc8ed4f23b   3 weeks ago    94.1MB
avenga/net-tools    latest          711a1d6e68c3   3 years ago    36.3MB
beneventsur/xubuntu vscode-1.44.2    05224dab2e89   4 years ago    1.41GB
```



119) Для начала избавимся о лишних файлов.

120) Запустите новый контейнер, а затем посмотрите что у него внутри:

```
docker run -d --rm --name=subject flask
```

```
docker exec subject ls
```

```
[(venv) ivd@ivd:~/tutorial/flask$ docker run -d --rm --name=subject flask
a265d11097f6ddf992aa13855823b7e68b15f1dbe0eee30fcc6f001335a3726c
[(venv) ivd@ivd:~/tutorial/flask$ docker exec subject ls
Dockerfile
app.py
requirements.txt
venv
```

121) Оценим сколько места занимают эти:

```
docker exec subject du -ha --max-depth=1
```

```
[(venv) ivd@ivd:~/tutorial/flask$ docker exec subject du -ha --max-depth=1
4.0K    ./app.py
24M     ./venv
4.0K    ./Dockerfile
4.0K    ./requirements.txt
24M     .
```

122) мы можем изменить Dockerfile и скопировать не все файлы из директории flask, а только те, которые нужны, т.е:

```
COPY requirements.txt /app
```

```
COPY app.py /app
```

123)Создайте в каталоге flask файл ".dockerignore" содержащий:

venv/Dockerfile

124)Пересоберите образ командой docker build, но назовите его flask:clean.

Здесь clean - это тег. Просто, чтобы иметь возможность отличить новый образ от старого. Если тег не указать, то будет установлено значение по умолчанию latest и новый образ затрёт старый.

```
(venv) ivd@ivd:~/tutorial/flask$ docker build -t flask:clean .
[+] Building 5.3s (9/9) FINISHED
=> [internal] load build definition from Dockerfile                                docker:default 0.0s
=> => transferring dockerfile: 159B                                              0.0s
=> [internal] load metadata for docker.io/library/python:3.10.12                1.3s
=> [internal] load .dockerignore                                                 0.0s
=> => transferring context: 57B                                                  0.0s
=> [1/4] FROM docker.io/library/python:3.10.12@sha256:bac3a0e0d16125977e351c861e2f4b12ecafeaa6f72431dc970d0b9 0.0s
=> [internal] load build context                                                0.0s
=> => transferring context: 118B                                                0.0s
=> CACHED [2/4] WORKDIR /app                                                    0.0s
=> [3/4] COPY / /app                                                            0.0s
=> [4/4] RUN pip install -r requirements.txt                                    3.8s
=> exporting to image                                                            0.1s
=> => exporting layers                                                            0.1s
=> => writing image sha256:b13d5e840cf233c5a195df61616c724b50681df47a7970094a87e19dab7958a3 0.0s
=> => naming to docker.io/library/flask:clean                                   0.0s
(venv) ivd@ivd:~/tutorial/flask$
```

125)Проверьте список образов и сравните размеры clean и latest.

```
(venv) ivd@ivd:~/tutorial/flask$ docker image ls
REPOSITORY          TAG          IMAGE ID       CREATED        SIZE
flask                clean        b13d5e840cf2  28 seconds ago 1.02GB
flask                latest       6957aad7d16b  8 minutes ago  1.04GB
nginx                latest       8dd77ef2d82e  2 weeks ago   193MB
chameleoncode/qrcode a400d78     23dc8ed4f23b  3 weeks ago   94.1MB
qrcode               latest       23dc8ed4f23b  3 weeks ago   94.1MB
avenga/net-tools     latest       711a1d6e68c3  3 years ago   36.3MB
beneventsur/xubuntu  vscode-1.44.2 05224dab2e89  4 years ago   1.41GB
(venv) ivd@ivd:~/tutorial/flask$
```

126) Теперь когда "мусор" в образе нет подумаем о замене базового образа для контейнера.

В качестве базового образа мы использовали образ "python:3.10.6" построенный на ubuntu. То есть полновесная ubuntu! Чтобы просто запустить Flask-приложение!! Есть много функций ubuntu, которые нам не нужны. Возможно, нам удастся найти более легкий дистрибутив Linux для запуска нашего приложения. Один из самых популярных - alpine. При этом, нам даже не придётся выполнять установку интерпретатора самостоятельно, реестре python на DockerHub есть python:3.10.6-alpine!

127) Откройте Dockerfile и замените первую строку на:

FROM python:3.10.6-alpine

```
(venv) ivd@ivd:~/tutorial/flask$ docker build -t flask:alpine .
[+] Building 5.6s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 81B
=> [internal] load metadata for docker.io/library/python:3.10.12-alpine
=> [internal] load .dockerignore
=> => transferring context: 57B
=> [internal] load build context
=> => transferring context: 28B
=> [1/2] FROM docker.io/library/python:3.10.12-alpine@sha256:5d1dafb473686c5435faed90387956d6f1fdd919e4ced85e
=> => resolve docker.io/library/python:3.10.12-alpine@sha256:5d1dafb473686c5435faed90387956d6f1fdd919e4ced85e
=> => sha256:5d1dafb473686c5435faed90387956d6f1fdd919e4ced85eb2aa5ac45d29a98c 1.65kB / 1.65kB
=> => sha256:c8f5b21532d1ecb7d1aa6aadbfe032276aca5b7f6b7e73375c694374e3de15d7 1.37kB / 1.37kB
=> => sha256:424efdac9c68d1e45964346274af15ee084174cc28696fd3bad9097dfbc0b445 6.28kB / 6.28kB
=> => sha256:9fda8d8052c61740409c4bea888859c141fd8cc3f58ac61943144ff6d1681b2d 3.33MB / 3.33MB
=> => sha256:e7ae3e644d565d1dbd9dfe33d6b9a021d4f05402864a4043e5632cb2397f1157 624.50kB / 624.50kB
=> => sha256:b5e51838445d740fc8753c2272b4d79b167d18822797e7140f803694ddcc5b5c 12.13MB / 12.13MB
=> => extracting sha256:9fda8d8052c61740409c4bea888859c141fd8cc3f58ac61943144ff6d1681b2d
=> => sha256:bb3cfbb259c2a81b148eb8d1abdcabb0ed163d27036668d9544de00389f33703 241B / 241B
=> => sha256:450c81b128cfcbd96958879ec4c383f1692291fa578b38d2bca8ca69d8e882be 3.08MB / 3.08MB
=> => extracting sha256:e7ae3e644d565d1dbd9dfe33d6b9a021d4f05402864a4043e5632cb2397f1157
=> => extracting sha256:b5e51838445d740fc8753c2272b4d79b167d18822797e7140f803694ddcc5b5c
=> => extracting sha256:bb3cfbb259c2a81b148eb8d1abdcabb0ed163d27036668d9544de00389f33703
=> => extracting sha256:450c81b128cfcbd96958879ec4c383f1692291fa578b38d2bca8ca69d8e882be
=> [2/2] COPY app.py /app
=> exporting to image
=> => exporting layers
=> => writing image sha256:f39a9b96d8394adb53e2ec58ba6957b65559146e9ecad81ffe8b6812c8fbac2
=> => naming to docker.io/library/flask:alpine
(venv) ivd@ivd:~/tutorial/flask$
```

128)Посмотрите список образов.

Новый образ должен весить что-то около 60 МБ. Всё еще немного великовато, но пока на этом остановимся.

```
[(venv) ivd@ivd:~/tutorial/flask$ docker image ls
REPOSITORY          TAG                IMAGE ID           CREATED            SIZE
flask                alpine            f39a9b96d839      2 minutes ago     55.2MB
flask                clean             b13d5e840cf2      5 minutes ago     1.02GB
flask                latest            6957aad7d16b      13 minutes ago    1.04GB
nginx                latest            8dd77ef2d82e      2 weeks ago       193MB
chameleoncode/qrcode a400d78           23dc8ed4f23b      3 weeks ago       94.1MB
qrcode              latest            23dc8ed4f23b      3 weeks ago       94.1MB
avenga/net-tools    latest            711a1d6e68c3      3 years ago       36.3MB
beneventsur/xubuntu vscode-1.44.2      05224dab2e89      4 years ago       1.41GB
(venv) ivd@ivd:~/tutorial/flask$
```

129)Удалите образы flask с тегами latest, clean.

```
[(venv) ivd@ivd:~/tutorial/flask$ docker rmi flask:clean
Untagged: flask:clean
Deleted: sha256:b13d5e840cf233c5a195df61616c724b50681df47a7970094a87e19dab7958a3
(venv) ivd@ivd:~/tutorial/flask$
```

130)Как упоминалось выше, если мы сейчас попробуем выполнить `docker run flask`, то получим ошибку, т.к. у нас нет образа `flask:latest` и с `dockerhub` его тоже скачать не получается. Следовательно, во всех командах придётся писать `flask:alpine`, что не всегда удобно. Исправим это.

131)Выполните команды:

```
docker image tag flask:alpine flask:latest
docker image tag flask:alpine my_app:latest
```

```
docker image ls
```

```

(env) ivd@ivd:~/tutorial/flask$ docker image tag flask:alpine flask:latest
(env) ivd@ivd:~/tutorial/flask$ docker image tag flask:alpine my_app:latest
(env) ivd@ivd:~/tutorial/flask$ docker image ls

```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
flask	alpine	f39a9b96d839	4 minutes ago	55.2MB
flask	latest	f39a9b96d839	4 minutes ago	55.2MB
my_app	latest	f39a9b96d839	4 minutes ago	55.2MB
<none>	<none>	6957aad7d16b	15 minutes ago	1.04GB
nginx	latest	8dd77ef2d82e	2 weeks ago	193MB
chameleoncode/qrcode	a400d78	23dc8ed4f23b	3 weeks ago	94.1MB
qrcode	latest	23dc8ed4f23b	3 weeks ago	94.1MB
avenga/net-tools	latest	711a1d6e68c3	3 years ago	36.3MB
beneventsur/xubuntu	vscode-1.44.2	05224dab2e89	4 years ago	1.41GB

```

(env) ivd@ivd:~/tutorial/flask$

```

132) Удалите my\_app:latest и flask:alpine командой docker rmi. При этом удалятся только имена, а сам образ останется под именем flask:latest.

```

(env) ivd@ivd:~/tutorial/flask$ docker rmi my_app:latest
Untagged: my_app:latest
(env) ivd@ivd:~/tutorial/flask$ docker rmi flask:alpine
Untagged: flask:alpine
(env) ivd@ivd:~/tutorial/flask$

```

133) Удалите my\_app:latest и flask:alpine командой docker rmi. При этом удалятся только имена, а сам образ останется под именем flask:latest.

```

(env) ivd@ivd:~/tutorial/flask$ docker run -d --name=flask_base flask
f06359bcdf6579e5c9892839d3b54458923e23994fe8f62b983a06c6c171593d
(env) ivd@ivd:~/tutorial/flask$

```

134) Запустите новый контейнер в фоновом режиме:

```
docker run -d --name=flask_base flask
```

```

(env) ivd@ivd:~/tutorial/flask$ docker run -d --name=flask_base flask
f06359bcdf6579e5c9892839d3b54458923e23994fe8f62b983a06c6c171593d
(env) ivd@ivd:~/tutorial/flask$

```

135) Установим в этом контейнере модуль для работы с mysql - "mysql-connector-python":

```
docker exec flask_base pip install mysql-connector-python
```

```
[(venv) ivd@ivd:~/tutorial/flask$ docker exec flask_base pip install mysql-connector-python]
```

136) Остановите контейнер, а затем скопируйте локальный файл app.py в контейнер при помощи команды:

```
docker cp app.py flask_base:/app/app.py
```

```
[(venv) ivd@ivd:~/tutorial/flask$ docker cp app.py flask_base:/app/app.py  
Successfully copied 4.1kB to flask_base:/app/app.py]
```

137) Мы изменили в контейнере всё, что хотели, теперь из него можно сделать образ (ну или просто запустить (docker start))

```
docker commit flask_base var_keeper:latest
```

```
[(venv) ivd@ivd:~/tutorial/flask$ docker commit flask_base var_keeper:latest  
sha256:f452c9399054093481a6f1666dfadac7b01b3aface496c3b1fb565faebefe506  
(venv) ivd@ivd:~/tutorial/flask$ ]
```

138) Убедитесь, что новый образ появился в списке.

139) Теперь проверим работоспособность получившегося образа, но для этого нам понадобится контейнер с базой данных mysql. Разумеется на DockerHub такой есть.

140) Создайте сеть под названием "net", а затем запустите контейнер с mysql как указано на странице образа:

```
docker run -d --name=db --network=net -v vars:/var/lib/mysql -e
```

```
MYSQL_ROOT_PASSWORD=123 mysql
```

```
[(venv) ivd@ivd:~/tutorial/flask$ docker run -d --name=db --network=net -v vars:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=123 mysql  
Unable to find image 'mysql:latest' locally  
latest: Pulling from library/mysql  
3c4da62cd991: Downloading [=====>] 34.31MB/47.65MB  
6246d8c92d7f: Download complete  
64e7eb422142: Download complete  
f787785dc518: Download complete  
7585e10db184: Download complete  
c69999011c03: Download complete  
1142f05d9006: Downloading [=====>] 13.25MB/46.08MB  
d598a5053b5c: Download complete  
3de4462059cc: Downloading [====>] 4.283MB/64.96MB  
cca926667410: Waiting  
]
```

141)Теперь запустите контейнер из образа var\_keeper:latest

`docker run -d --name=keeper --network=net -p 80:5000 var_keeper`

```
((venv) ivd@ivd:~/tutorial/flask$ docker run -d --name=keeper --network=net -p 80:5000 var_keeper
```

142)Проверьте работоспособность контейнера при помощи браузера. При отправке приложению POST запроса на роут /var/<имя переменной> значение должно добавляться в базу данных, а при отправке GET запроса - извлекаться из базы и выводиться на экран.

Для отправки POST запроса можно ввести в строку адреса:

`data:text/html,<form action=http://{ip_виртуальной_машины}:80/var/a  
method=post><input name=value></form>`

143)Затем отправьте GET запрос (просто перейдите по `http://`

`{ip_виртуальной_машины}/var/a`) в браузере должно появиться то значение, которое вы вводили ранее.

144)Зарегистрируетесь на <https://hub.docker.com> и зайдите под своей учёткой (возможно понадобится VPN);

145)Нажмите кнопку "Create repository" и в качестве имени укажите "var\_keeper".

---

chameleoncode / [Repositories](#) / [var\\_keeper](#) / [General](#)

---

**General**

Tags

Builds

Collaborators

Webhooks

Settings

---



146)Перейдите в терминал и введите команду:

docker login

```
((venv) ivd@ivd:~/tutorial/flask$ docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com/ to create one.
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grants better security and is required for organizations using SSO. Learn more at https://docs.docker.com/go/access-tokens/

Username: chameleoncode
Password:
WARNING! Your password will be stored unencrypted in /home/ivd/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
(venv) ivd@ivd:~/tutorial/flask$
```

147)Добавьте образу var\_keeper ещё одно имя <имя пользователя

DockerHub>/var\_keeper. В моём случае так:

docker tag var\_keeper:latest vladimirschabanov/var\_keeper:latest

Как обычно latest можно не указывать. Новое имя и тег могут быть любыми.

```
((venv) ivd@ivd:~/tutorial/flask$ docker tag var_keeper:latest chameleoncode/var_keeper:latest
(venv) ivd@ivd:~/tutorial/flask$
```

148)Отправьте образ на DockerHub командой docker push. В моём случае так:

docker push vladimirschabanov/var\_keeper

```
((venv) ivd@ivd:~/tutorial/flask$ docker push chameleoncode/var_keeper
Using default tag: latest
The push refers to repository [docker.io/chameleoncode/var_keeper]
5525a2acfd0f: Pushed
4596d9fc0e92: Pushed
506efcf0c90d: Mounted from library/python
490cc9826c89: Mounted from library/python
a4a998889b38: Mounted from library/python
07c9596e31dd: Mounted from library/python
b2191e2be29d: Mounted from library/python
latest: digest: sha256:35893cf11dc5ea76d5d04c3ca6d61fab600e898a7f28c5df2c45a3210e581746 size: 1784
(venv) ivd@ivd:~/tutorial/flask$
```



## Docker - многоконтейнерные приложения

1) Удалите все образы (кроме nginx) и все контейнеры которые у вас могли остаться с прошлой работы.

```
(venv) ivd@ivd:~/tutorial/flask$ docker rmi -f $(docker images | grep -v "nginx" | awk '{print $3}')
```

2) Выполните команду:

`docker compose --help`

```
(venv) ivd@ivd:~/tutorial/flask$ docker compose --help

Usage:  docker compose [OPTIONS] COMMAND

Define and run multi-container applications with Docker

Options:
  --all-resources          Include all resources, even those not used by services
  --ansi string            Control when to print ANSI control characters ("never"|"always"|"auto"
                           (default "auto")
  --compatibility          Run compose in backward compatibility mode
  --dry-run               Execute command in dry run mode
  --env-file stringArray  Specify an alternate environment file
  -f, --file stringArray  Compose configuration files
  --parallel int          Control max parallelism, -1 for unlimited (default -1)
  --profile stringArray   Specify a profile to enable
  --progress string       Set type of progress output (auto, tty, plain, quiet) (default "auto")
  --project-directory string Specify an alternate working directory
                           (default: the path of the, first specified, Compose file)
  -p, --project-name string Project name

Commands:
  attach  Attach local standard input, output, and error streams to a service's running container
  build   Build or rebuild services
  config  Parse, resolve and render compose file in canonical format
```

3) Изучите структуру формата yaml;

4) В домашнем каталоге создайте каталог "compose" и перейдите в него;

5) Создайте файл "compose.yaml" содержащий:

version: "3.9"

services:

web:

image: nginx

6) Запустите приложение командой:

`docker compose up`

7) В данном случае мы запустили сервис "web" в attached mode, поэтому он захватил ввод и вывод нашего терминала.

Остановите контейнер `Ctrl+C` и посмотрите список всех контейнеров `docker ps -a`. Как видно в списке присутствует один контейнер построенный на базе образа `nginx`.

```
(venv) ivd@ivd:~/compose$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
2ed0af79e59a	nginx	"/docker-entrypoint..."	34 seconds ago	Exited (0) 6 seconds ago		compose-web-1

8) Выполните команду:

`docker compose down`

```
[(venv) ivd@ivd:~/compose$ docker compose down
WARN[0000] /home/ivd/compose/compose.yaml: `version` is obsolete
[+] Running 2/2
 ✓ Container compose-web-1   Removed
 ✓ Network compose_default   Removed
(venv) ivd@ivd:~/compose$
```

9) Проверьте, что теперь список контейнеров пуст.

10) Модифицируйте "compose.yaml" следующим образом:

services:

web:

image: nginx

ports:

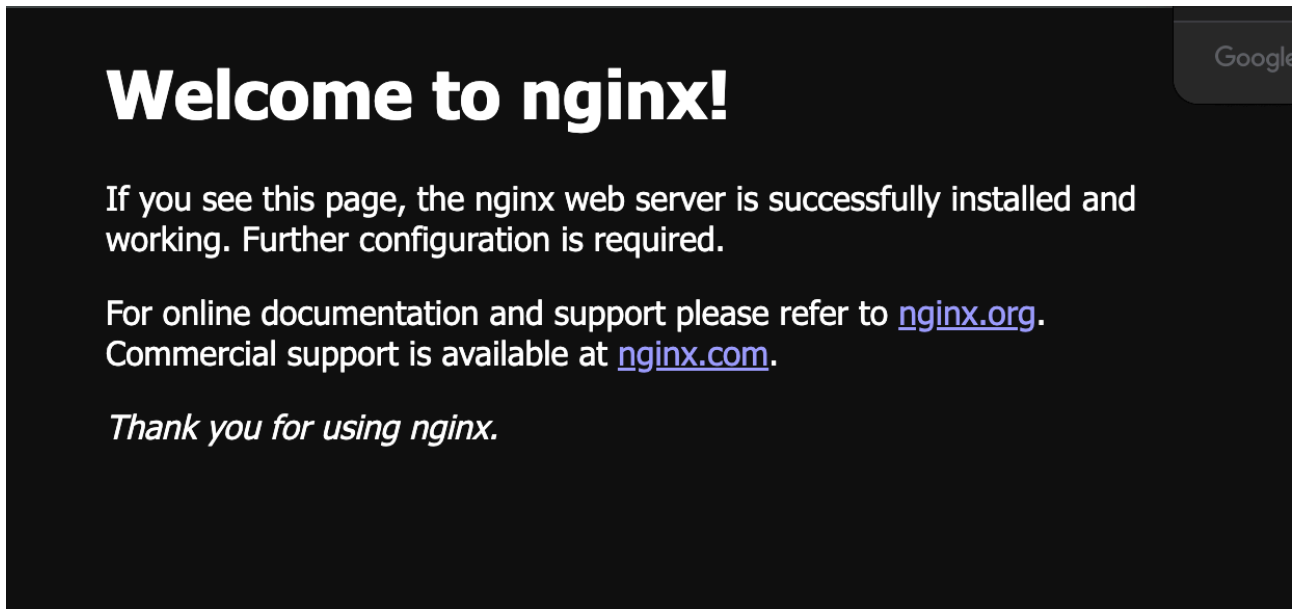
- "80:80"

и запустите приложение командой:

`docker compose up -d`

```
[(venv) ivd@ivd:~/compose$ docker compose up -d
[+] Running 2/2
 ✓ Network compose_default   Created
 ✓ Container compose-web-1   Started
(venv) ivd@ivd:~/compose$
```

11)Перейдите в браузер в основной ОС и введите в строку адреса ip виртуальной машины. Вы должны увидеть приветственную страницу от nginx.



12)Остановите приложение командой:

`docker compose stop`

```
[(venv) ivd@ivd:~/compose$ docker compose stop  
[+] Stopping 1/1  
✓ Container compose-web-1 Stopped
```

13)Запустите приложение командой:

`docker compose start`

```
[(venv) ivd@ivd:~/compose$ docker compose start  
[+] Running 1/1  
✓ Container compose-web-1 Started  
(venv) ivd@ivd:~/compose$
```

14)Остановите приложение при помощи команды:

`docker compose down`

```
(venv) ivd@ivd:~/compose$ docker compose down
[+] Running 2/2
 ✓ Container compose-web-1   Removed
 ✓ Network compose_default   Removed
(venv) ivd@ivd:~/compose$
```

15) Модифицируйте "compose.yaml" следующим образом:

services:

web:

container\_name: web

image: nginx

ports:

- "80:80"

volumes:

- ./content:/usr/share/nginx/html

16) Запустите приложение и проверьте, что контейнеру присвоено имя "web" и в текущем каталоге появился каталог "content".

```
(venv) ivd@ivd:~/compose$ docker compose up -d
[+] Running 2/2
 ✓ Network compose_default   Created
 ✓ Container web              Started
(venv) ivd@ivd:~/compose$ ls
compose.yaml  content
(venv) ivd@ivd:~/compose$
```

17) Как и ожидалось каталог пустой и принадлежит root. Поменяйте владельца каталога на своего пользователя и создайте в нём файл "index.html"

содержащий:

<h1>Hello from compose</h1>



Не защищено

192.168.89.77



Import to Loox

# Hello from compose

18) Остановите приложение (down) и убедитесь, что каталог "content" по-прежнему существует.

```
[(venv) ivd@ivd:~/compose/content$ docker compose down
[+] Running 2/2
 ✓ Container web          Removed
 ✓ Network compose_default Removed
(venv) ivd@ivd:~/compose/content$
```

19) Модифицируйте "compose.yaml" следующим образом:

services:

web:

image: nginx

container\_name: web

ports:

- "80:80"

volumes:

- data:/usr/share/nginx/html

volumes:

data:

```

[(venv) ivd@ivd:~/compose$ cat <<END>> compose.yaml
> services:
  web:
    image: nginx
    container_name: web
    ports:
      - "80:80"
    volumes:
      - data:/usr/share/nginx/html

volumes:
[  data:
[> END
[(venv) ivd@ivd:~/compose$

```

20) Запустите приложение и проверьте список томов.

Как видно там появился том, название которого состоит из названия текущего каталога и названия тома в yaml файле.

Если том с таким именем уже существует и он создан не compose, то вы получите предупреждение. Чтобы намеренно использовать существующий том, нужно создавать его таким образом:

volumes:

data:

external: true

21) Остановите приложение и убедитесь, что том по-прежнему существует.

```

[(venv) ivd@ivd:~/compose$ docker volume ls
DRIVER      VOLUME NAME
local       compose_data
local       data
local       vars

```

22) Модифицируйте "compose.yaml" следующим образом:

services:

web:

image: nginx

container\_name: web

ports:

- "80:80"

volumes:

- data:/usr/share/nginx/html

networks:

- local

tools:

image: avenga/net-tools

command: "sleep 10m"

networks:

- local

volumes:

data:

networks:

local:

23) Запустите приложение и посмотрите список сетей.

Как видно там появилась сеть, название которой состоит из названия текущего каталога и названия сети в yaml файле.

Для подключения к существующей сети также как и для тома используйте ключ external: true.

```
[+] Running 3/3
✓ tools Pulled
✓ 540db60ca938 Pull complete
✓ 1dca19b5623d Pull complete
[+] Running 2/3
✓ Network compose_local
Created0.1s
Container compose-tools-1
[+] Running 2/4
Starting0.1s
✓ Network compose_local
Created0.1s
Container compose-tools-1
[+] Running 3/4
Starting0.2s
✓ Network compose_local
Created0.1s
Container compose-tools-1
[+] Running 4/4
Started0.3s
✓ Network compose_local
Created0.1s
Container compose-tools-1
Started0.3s
Container web
Started0.3s
! tools The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8)
no specific platform was requested 0.0s
(venv) ivd@ivd:~/compose$ docker network ls

```

NETWORK ID	NAME	DRIVER	SCOPE
85d468ff9bc5	bridge	bridge	local
ffe56cb52f73	compose_local	bridge	local
481d93e7c9cf	host	host	local
af2ad5ca318f	none	null	local

```
(venv) ivd@ivd:~/compose$
```

24)Выполните команду:

`docker network inspect compose_local`

```
[(venv) ivd@ivd:~/compose$ docker network inspect compose_local
[
  {
    "Name": "compose_local",
    "Id": "ffe56cb52f73d79594b38cc65d3394129ec1c470a045a7eccd4a7ec91aa59c78",
    "Created": "2024-05-18T12:57:47.938282529Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.23.0.0/16",
          "Gateway": "172.23.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "25fda9f1cbd6aa2d15cdb2f5fa96056c72a60b62a4c69302976c6fbcdeef89ca": {
```

25)Выполните команду:



`docker compose exec tools curl {ip_контейнера_web}`

26) Остановите приложение и проверьте, что сети "compose\_local" больше нет в списке.

```
[(venv) ivd@ivd:~/compose$ docker compose down
[+] Running 3/3
 ✓ Container web                Removed
 ✓ Container compose-tools-1    Removed
 ✓ Network compose_local        Removed
[(venv) ivd@ivd:~/compose$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
85d468ff9bc5    bridge    bridge       local
481d93e7c9cf    host      host         local
af2ad5ca318f    none      null         local
(venv) ivd@ivd:~/compose$
```

27) В прошлой работе вы создавали приложение "var\_keeper" и загружали его на свой аккаунт в DockerHub. Используем этот образ и развернём приложение на локальной машине при помощи compose.

28) Модифицируйте "compose.yaml" следующим образом:

services:

keeper:

image: {ваш\_логин}/var\_keeper

ports:

- "80:5000"

networks:

- net

db:

image: mysql

environment:

- MYSQL\_ROOT\_PASSWORD=123

volumes:

- vars:/var/lib/mysql

networks:

- net

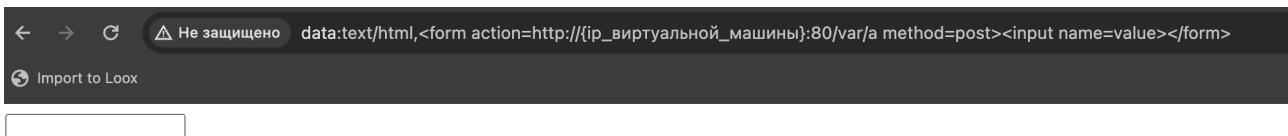
volumes:

vars:

networks:

net:

29) Запустите приложение и проверьте его работоспособность при помощи браузера. Чтобы послать POST-запрос сервису "keeper" введите в строке адреса: `data:text/html,<form action=http://{ip_виртуальной_машины}:80/var/a method=post><input name=value></form>`



30) В результате вы должны получить ошибку. Воспользуйтесь командой:  
`docker compose logs`

```
[(venv) ivd@ivd:~/compose$ docker compose logs
```

31) Приложение не работает, поэтому остановите его.

```
[(venv) ivd@ivd:~/compose$ docker compose down
```

32) Чтобы указать порядок запуска сервисов в compose есть специальный ключ depends\_on, который позволяет указать список сервисов, которые должны быть запущены перед текущим.

Модифицируйте "compose.yaml" следующим образом:

services:

keeper:

image: {ваш\_логин}/var\_keeper

ports:

- "80:5000"

networks:

- net

depends\_on:

- db

db:

image: mysql

environment:

- MYSQL\_ROOT\_PASSWORD=123

volumes:

- vars:/var/lib/mysql

networks:

- net

volumes:

vars:

networks:

net:

33) Приложение по прежнему не работает - остановите его.

35) Чтобы решить нашу задачу compose предоставляет возможность проверять "здоровье" (healthy) контейнера при помощи ключа healthcheck.

Модифицируйте "compose.yaml" следующим образом:

services:

keeper:

image: {ваш\_логин}/var\_keeper

ports:

- "80:5000"

networks:

- net

depends\_on:

db:

condition: service\_healthy

db:

image: mysql

environment:

- MYSQL\_ROOT\_PASSWORD=123

volumes:

- vars:/var/lib/mysql

networks:

- net

healthcheck:

test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]

timeout: 5s

retries: 10

volumes:

vars:

networks:

net:

36) Запустите приложение, дождитесь пока все контейнеры стартуют и затем проверьте логи.

37) Если в процессе работы приложения один или несколько контейнеров остановятся, то приложение перестанет работать нормально. Чтобы поддерживать его работоспособность можно назначить политику перезапуска при помощи ключа restart.

38) Остановите приложение и модифицируйте "compose.yaml" следующим образом:

services:

keeper:

image: {ваш\_логин}/var\_keeper

ports:

- "80:5000"

networks:

- net

depends\_on:

db:

condition: service\_healthy

restart: always

db:

image: mysql

environment:

- MYSQL\_ROOT\_PASSWORD=123

volumes:

- vars:/var/lib/mysql

networks:

- net

healthcheck:

test: ["CMD", "mysqladmin" ,"ping", "-h", "localhost"]

timeout: 5s

retries: 10

volumes:

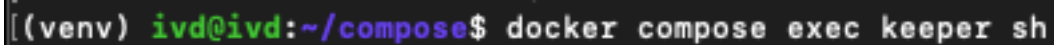
vars:

networks:

net:

39)Запустите приложение и зайдите в контейнер сервиса "keeper" при помощи команды:

`docker compose exec keeper sh`



```
[(venv) ivd@ivd:~/compose$ docker compose exec keeper sh
```

40)Проверьте список запущенных контейнеров и вы должны обнаружить, что "keeper" по прежнему на месте, но в столбце STATUS указано, что он был запущен совсем недавно.

41)Остановите приложение.

42)

```

Using cached jinja2-3.1.4-py3-none-any.whl (133 kB)
Collecting Werkzeug>=3.0.0
  Using cached werkzeug-3.0.3-py3-none-any.whl (227 kB)
Collecting charset-normalizer<4,>=2
  Downloading charset_normalizer-3.3.2-cp310-cp310-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (138 kB)
    138.2/138.2 KB 1.3 MB/s eta 0:00:00
Collecting idna<4,>=2.5
  Downloading idna-3.7-py3-none-any.whl (66 kB)
    66.8/66.8 KB 5.5 MB/s eta 0:00:00
Collecting certifi>=2017.4.17
  Downloading certifi-2024.2.2-py3-none-any.whl (163 kB)
    163.8/163.8 KB 5.6 MB/s eta 0:00:00
Collecting urllib3<3,>=1.21.1
  Downloading urllib3-2.2.1-py3-none-any.whl (121 kB)
    121.1/121.1 KB 6.8 MB/s eta 0:00:00
Collecting MarkupSafe>=2.0
  Using cached MarkupSafe-2.1.5-cp310-cp310-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (26 kB)
Installing collected packages: urllib3, MarkupSafe, itsdangerous, idna, click, charset-normalizer, certifi, blinker, Werkzeug, requests, Jinja2, flask
Successfully installed Jinja2-3.1.4 MarkupSafe-2.1.5 Werkzeug-3.0.3 blinker-1.8.2 certifi-2024.2.2 charset-normalizer-3.3.2 click-8.1.7 flask-3.0.3 idna-3.7 itsdangerous-2.2.0 requests-2.31.0 urllib3-2.2.1
[(venv) ivd@ivd:~/compose/front$ pip freeze > requirements.txt ]
[(venv) ivd@ivd:~/compose/front$ touch Dockerfile ]
[(venv) ivd@ivd:~/compose/front$ cat <<END>> Dockerfile ]
> FROM python:3.10.12-alpine
WORKDIR /app
COPY / /app
RUN pip install -r requirements.txt
ENTRYPOINT [ "python" ]
CMD [ "app.py" ]
> END
(venv) ivd@ivd:~/compose/front$ █

```

43) Теперь у нас есть готовый фронтенд и Dockerfile, чтобы упаковать его в контейнер.

Выйдите из каталога "front" и модифицируйте "compose.yaml" следующим образом:

services:

front:

build: ./front

image: var\_keeper\_front

ports:

- "80:80"

networks:

- net

depends\_on:

- keeper

restart: always

keeper:

image: {ваш\_логин}/var\_keeper

networks:

- net

```
depends_on:
  db:
    condition: service_healthy
restart: always
```

```
db:
  image: mysql
  environment:
    - MYSQL_ROOT_PASSWORD=123
  volumes:
    - vars:/var/lib/mysql
  networks:
    - net
  healthcheck:
    test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
    timeout: 20s
    retries: 10
```

volumes:

vars:

networks:

net:

44)Выполните команду:

docker compose build

```
[(venv) ivd@ivd:~/compose$ docker compose build
```



45)Проверьте список образов. Теперь в нём должен появиться образ "var\_keeper\_front:latest".

46)Выполните команду:  
docker compose up -d

```
[(venv) ivd@ivd:~/compose$ docker compose up -d
```

47)Перейдите в браузер и введите в строку адреса ip виртуальной машины. Воспользуйтесь приложением, чтобы сохранить две переменные, а затем проверьте, что их значения корректно воспроизводятся.

**Вывод:** сегодня на практической работе по дисциплине «Современные технологии программирования» я ознакомился с базовыми возможностями Docker и Podman. Приобрести опыт и навыки контейнеризации приложений.

### Контрольные вопросы

1)Как запустить контейнер в интерактивном режиме?

Запуск контейнера в интерактивном режиме:

```
docker run -it image_name /bin/bash
```

Флаг -i (interactive) обеспечивает интерактивный ввод, а -t (tty) выделяет псевдо-ТТУ терминал.

2)Что нужно сделать, чтобы при вводе команд для работы с контейнерами не нужно было вводить sudo?

Чтобы работать с Docker без sudo, нужно добавить текущего пользователя в группу Docker:

```
sudo usermod -aG docker $USER
```

3)Каким образом можно соединить 2 контейнера в отдельную для них сеть?

Для соединения двух контейнеров в отдельную сеть:

1. Создайте сеть:

```
docker network create custom_network
```

2. Запустите контейнеры, указав сеть:

```
docker run --network custom_network image1
```

```
docker run --network custom_network image2
```

4)Два контейнера хотят обмениваться файлами. Каким образом можно организовать такое взаимодействие?

Для обмена файлами между контейнерами можно использовать тома (volumes):

```
docker run -v /host/path:/container/path image
```

5)Мне нужен образ с wordpress на основе дистрибутива alpine. Приведите команду которой я могу скачать такой образ с dockerhub (версию wordpress и php выберите любую);

Чтобы скачать образ WordPress на базе Alpine:

```
docker pull wordpress:latest-php8.0-apache-alpine
```

6)Как указать имя контейнера (не образа) в docker-compose.yml файле?

. Имя контейнера в docker-compose.yml указывается через параметр container\_name:

services:

my-wordpress:

container\_name: my-wordpress-container

7)Как директива healthcheck влияет на работу контейнер?

Директива healthcheck позволяет Docker отслеживать состояние контейнера и перезапускать его, если проверка не пройдена.

8)Как указать статический IP адрес для контейнера в docker-compose.yml?

Статический IP-адрес контейнера задается через параметр ipv4\_address в секции networks:

services:

my-app:

# ...

networks:

custom\_network:

ipv4\_address: 172.16.238.10

9)Как depends\_on влияет на порядок запуска контейнеров ?

depends\_on определяет порядок запуска контейнеров - сначала запустятся зависимые контейнеры, а потом контейнеры, которые от них зависят.

10)В чем разница между up , run и start ?

- docker run запускает новый контейнер
- docker start запускает остановленный контейнер
- docker up создает и запускает контейнеры, описанные в docker-compose.yml