

#docker #linux #IT

Основные команды

⚠ Важно

Перед началом работы нужно убедиться, что docker установлен и docker daemon запущен. и добавить текущего пользователя в группу docker `sudo usermod -aG docker $USER`

проверка статуса сервиса docker:

```
service docker status
```

базовая команда для проверки работоспособности docker:

```
docker --version
```

запуск контейнера для проверки:

```
docker run helloworld
```

проверить запущенные контейнеры (покажет только **запущенные**):

```
docker ps
```

покажет **все** контейнеры:

```
docker ps -a
```

удалит *контейнер*:

```
docker rm name_of_cont/id_of_con
```

список всех образов:

```
docker images
```

удалить образ:

```
docker rmi id_of_im
```

скачать образ с docker hub:

```
docker pull name_of_image
```

запуск контейнера в фоновом режиме:

```
docker run -d --name Myfirst ubuntu
```

подключиться уже к существующему контейнеру:

```
docker start id\name_of_cont
```

поставить контейнер на паузу:

```
docker pause id
```

убрать контейнер с паузы:

```
docker unpause id
```

остановить/принудительно завершить процесс:

```
docker stop/kill id
```

после отработки контейнер автоматически удалится:

```
docker --rm -d ubuntu
```

посмотреть всю информацию о контейнере:

```
docker inspect id
```

список ресурсов контейнера:

```
docker stats id
```

просмотр логов контейнера:

```
docker logs id (можно с параметром -f)
```

подключиться к контейнеру:

```
docker exec -it id/name bin/bash
```

удалить все контейнеры, образы и прочее:

```
docker system prune -a --volumes
```

запуск контейнера с пробросом портов:

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

посмотреть содержит информацию о каждом объеме Docker:

```
docker volume ls
```

команда, позволяющая удалять ненужные volumes:

```
docker volume rm name_of_v
```

если необходимо посмотреть информацию о сети пишем команду:

```
docker network inspect name/id
```

мы можем указать какие ip адреса мы будем использовать:

```
docker network create -d bridge --subnet 192.168.10.0/24 --gateway 192.168.10.1 NAME
```

если необходимо удалить сети:

```
docker network rm name/id
```

чтобы собрать образ на основе Dockerfile пишем команду (Если необходимо будет запустить образ на docker hub то нужно называть образ так имя_гитхаб/название_репозитория:тег):

```
docker build -t NAMVE:version .
```

чтобы выложить образ на docker hub:

```
docker push NAMVE:version
```

чтобы взять образ с docker hub:

```
docker pull NAMVE:version
```

опция, которая позволяет удалить контейнер сразу, как только он отработал:

```
docker run --rm id/name
```

Работа с docker

Реестр для хранения **docker images**.

[Docker Hub](#)

alpine - самый легковесный linux дистрибутив.

В примере ниже мы скачиваем образ **ubuntu** (по умолчанию самую свежую).

```
> docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
6b851dcae6ca: Pull complete
Digest: sha256:6120be6a2b7ce665d0cbddc3ce6eae60fe94637c6a66985312d1f02f63cc0bcd
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

🕒 Все контейнеры, работающие без процессов, обрабатывают и сразу останавливаются.

Контейнеры можно запускать в фоновом режиме `docker run -d ubuntu`, в таком случае можно подключаться к контейнерам (например к терминалу). Чтобы не создавать новый контейнер, а подключиться уже к существующему можем использовать команду `docker start id\name_of_cont`.

🔗 Можно не прописывать id контейнера целиком достаточно указать первые 2-3 уникальных символа контейнера

```
> docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS          NAMES
7ea29ede0a6c   ubuntu    "sleep 5"               8 minutes ago Exited (0) 8 minutes ago           priceless_newton
3df1a62c0468   retrodance "/.server"             13 days ago   Exited (137) 13 days ago           beautiful_kilby
9025a0f5805b   retrodance "/.server"             13 days ago   Exited (137) 13 days ago           optimistic_goodall
9c16f53767a9   retrodance "/.server"             13 days ago   Created                                dazzling_shtern
cbd3a1a32d1a   timfej/exam_01_07_06_2023:1.0.0 "/mytcpserver"        2 weeks ago   Created                                infallible_aryabhata
487e62f34fed   timfej/exam_01_07_06_2023:1.0.0 "/mytcpserver"        2 weeks ago   Created                                exciting_davinci
> docker rm 7e 3d
7e
3d
```

Контейнеры можно ставить на паузу и запускать вновь командами `docker pause id` и `docker unpause id`. Также не менее полезным бывает остановка и фатальная остановка контейнера `docker stop/kill id`.

Иногда бывает полезно, чтобы сразу после отработки контейнер автоматически удалялся, для этого существует команда `docker --rm -d ubuntu`.

Чтобы посмотреть всю необходимую информацию о контейнере есть команда `docker inspect id`. (можно посмотреть как собирался контейнер, порты и т. д.).

Если необходимо узнать сколько ресурсов требует определенный контейнер `docker stats id`. (будет выведена следующая информация id контейнера, имя, сколько занимают памяти, нагрузка на сеть, нагрузка на диски и т. д.).

Узнать logs контейнера можно двумя способами `docker logs id` или если необходимо просматривать в live режиме, то используем `docker logs -f id`. Эта команда полезна для диагностики проблем, анализа производительности, мониторинга безопасности и анализа пользовательского поведения.

Чтобы подключиться к контейнеру можем использовать команду `docker exec -it id/name`.

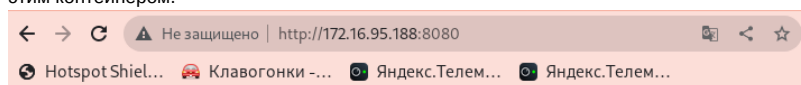
🔗 В docker контейнере могут работать не все команды.

Для выхода из контейнера используем комбинацию клавиш `ctrl + d`.

Управление портами (Port mapping)

Все запросы, которые будут поступать на сервер на данный порт, будут автоматически пробрасываться на контейнер.

Посмотреть все порты на сервере можно с помощью команды `netstat -tulpen`. Для переброса портов используем команду `docker run -d -p 8080:80 nginx`. В этом случае будет связан 80 порт на сервере и 80 порт на контейнере и все запросы приходящие на сервер на порт 80 будут обрабатываться этим контейнером.



Welcome to nginx!

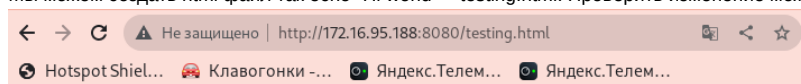
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Html файл с содержимым отображаемой страницы хранится на контейнере в папке `/usr/share/nginx/html`.

Мы можем создать html файл так `echo "Hi world" > testing.html`. Проверить изменение можно написав в браузере `ip/testing.html`.



Hi world!

Переменные. Environment Variables

Переменные на основе контейнеров mysql.

```
docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag
```

`env` - можем запросить все переменные linux системы.`

При запуске можно сразу запустить базу данных. Не обязательно сначала открывать `bash` оболочку, а потом подключаться к базе данных.

```
> docker exec -it 7a mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 8.0.33 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement
.

mysql>
```

Важно

Очень часто все необходимые переменные можно найти в официальной документации к образу на [git/docker hub](#)

Постоянные данные

Порой бывает необходимо завершать работу контейнера, например, чтобы установить более свежую версию `sql` или по другим причинам. Но перед этим нам необходимо остановить предыдущий контейнер. При вводе команд `docker stop/rm/kill mysql` теряются все данные (конфигурационные файлы или файлы которые были изменены в ходе работы приложения) в контейнере (также работает с приложением).

Чтобы сохранять данные, мы можем сохранять их на самом сервере или `host`. Это называется **persisting data** (сохраняющиеся данные)

Есть 3 способа как мы можем сохранить данные из контейнера.

1. Host Volumes

при запуске контейнера мы используем параметр `-v` и указываем данные. Предварительно нужно создать необходимые директории на сервере (хосте). Если мы работаем с сервером `nginx`, то нужно создавать `html` файл на сервере, иначе, при отсутствии файла, при запуске контейнера будет показывать ошибку, которая ссылается на то, что файл не найден.

Слева часть которая принадлежит самому серверу, то есть `/opt/mysql_data`, а справа директория в `docker` контейнере. Этим действием мы монтируем физический сервер в `docker` контейнер.

2. Anonymous Volumes

Не удобный метод, поскольку, если есть много `volumes`, то сложно понять, что к чему относится. Если мы удалим контейнер, то данные в папке `*/var/lib/docker/volumes/HASH*` тоже удалятся, то есть доступ к данным будет только до тех пор, пока запущен контейнер. В случае с `nginx` сервером `-v /usr/share/nginx/html`.

3. Named Volumes

На сервере в папке `/var/lib/docker/volumes` создается директория которую мы хотим добавить в контейнер.

чем отличаются Anonymous/Named Volumes?

Anonymous Volumes будет сгенерирован какой-то хеш (номер), а в Named Volumes мы пишем название директории самостоятельно. Данные при остановке контейнера в Anonymous не сохраняются, а в Named сохраняются.

`docker volumes ls` - содержит информацию о каждом объеме Docker, включая его идентификатор (ID), имя, драйвер хранения (storage driver) и точку монтирования (mountpoint).

Команда, позволяющая удалять ненужные `volumes`:

```
docker volume rm name_of_v
```

Пример команды в докер, с монтированием директории с сервера в контейнер `docker`:

```
docker run --name some-nginx -v /some/content:/usr/share/nginx/html:ro -d nginx
```

Опция `-v` в Docker используется для монтирования файлов или директорий между хост-системой и контейнером. Не важно, передается ли файл с хост-системы на контейнер или наоборот - данные будут скопированы в обоих направлениях, и изменения, сделанные в одном месте, будут отражены в другом.

Когда мы используем опцию `-v` для монтирования файлов или директорий в контейнер, любые изменения, сделанные внутри контейнера, будут отражены на хост-системе. То есть, если в контейнере созданы или изменены файлы, изменения будут видны и сохранены на хост-системе.

Исключение составляет опция `ro`, она указывается после пути в папке контейнера:

```
docker run -v /путь/к/файлу/на/хосте:/путь/к/файлу/в/контейнере:ro <image_name>
```

ro - read only, значит, что контейнер может только читать, и не может изменять.

Сетевые настройки (Docker Networks)

1. bridge

С помощью параметра `-p port:port` можно переадресовывать все запросы, которые поступают локально на сервер на контейнер. Если контейнер запустить с помощью команды `docker run cont` без параметров, то он попадет в default сеть. С помощью команды:

```
docker network create --driver/-d bridge NAME
```

мы можем создать свою сеть и в этой сети контейнеры могут общаться с помощью dns, то есть по именами которые указываются в параметре `--name` при создании контейнера. При запуске контейнера необходимо указать к какой сети мы подключаем контейнер. Параметр `--driver bridge` указывать не обязательно, он используется по умолчанию. Мы можем пинговать контейнеры в одной сети используя *только ip адреса*.

⚠ Контейнеры из разных сетей не могут общаться друг с другом, это может быть полезным для полной изоляции отдельных приложений.

2. host

С помощью параметра `-network=host`, контейнер получается ip адрес сервера, чтобы подключиться, достаточно указать ip адрес сервера и необходимый порт. При создании сети необходимо прописать `--driver host`:

```
docker network create --driver/-d host NAME
```

В таком случае чтобы подключиться к контейнеру необходимо написать ip адрес сервера. Мы можем пинговать контейнеры в одной сети используя **не** только **ip адреса**, но и **имена контейнеров**.

⚠ Нельзя создавать больше 1 сети типа host.

3. none

С помощью параметра `-network=none`, в данном случае мы никак не сможем подключиться к контейнеру из вне и локально, но сможем подключиться к контейнеру, подключившись непосредственно к самому контейнеру.

. При создании сети необходимо прописать `--driver none`:

```
docker network create --driver/-d null NAME
```

⚠ Нельзя создавать больше 1 сети типа null.

4. macvlan и ipvlan

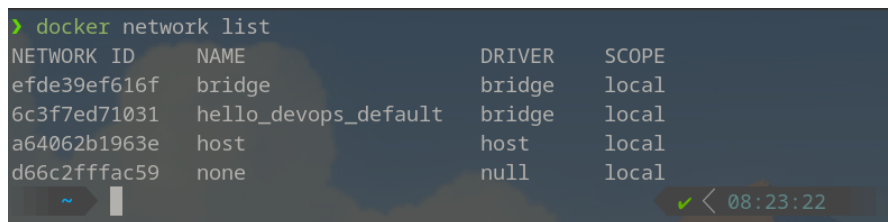
В случае с `macvlan` у каждому контейнеру присваивается свой mac адрес, а `ipvlan` mac адрес общий для сервера и контейнеров.

5. overlay

Используется тогда, когда docker запускается в cluster.

Чтобы посмотреть список существующих сетей пишем команду:

```
docker network list
```



```
> docker network list
NETWORK ID      NAME                DRIVER  SCOPE
efde39ef616f    bridge             bridge   local
6c3f7ed71031    hello_devops_default bridge   local
a64062b1963e    host               host     local
d66c2fffac59    none              null     local
```

Если необходимо посмотреть информацию о сети пишем команду:

```
docker network inspect name/id
```

Мы можем указать какие ip адреса мы будем использовать:

```
docker network create -d bridge --subnet 192.168.10.0/24 --gateway 192.168.10.1 NAME
```

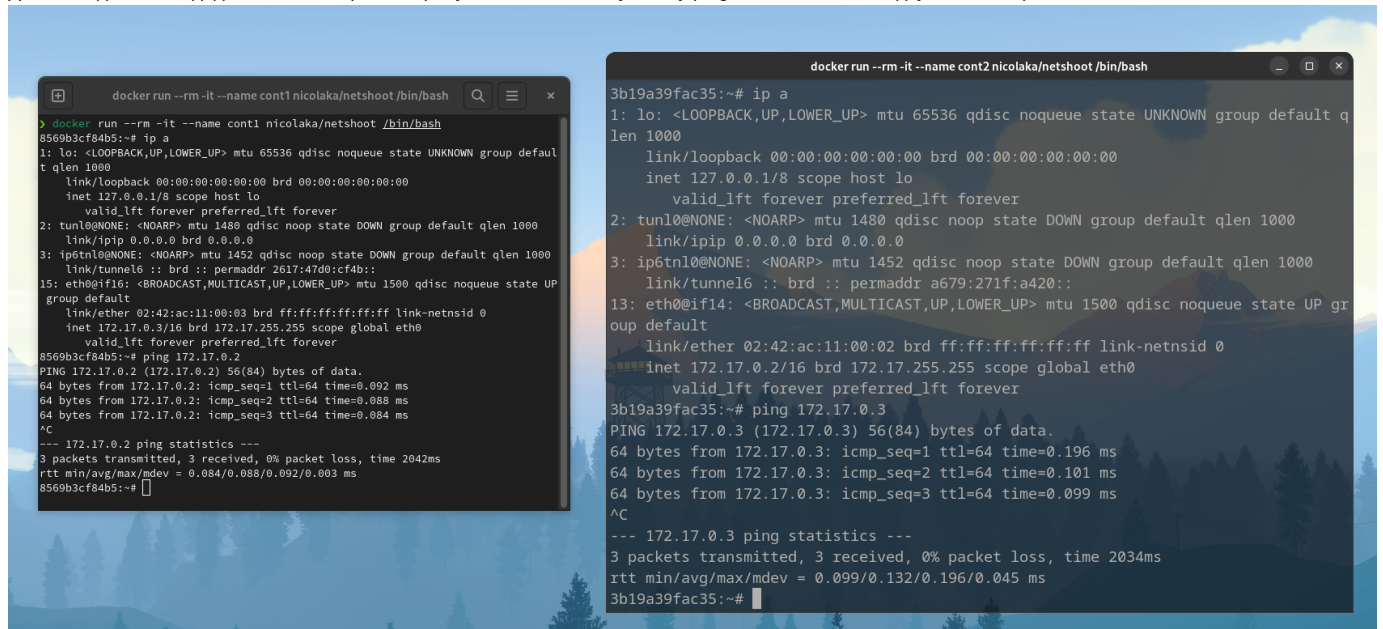
Если необходимо удалить сети:

```
docker network rm name/id
```

В контейнере `ubuntu` не будет работать команда `ip` а, поэтому для проверки этого функционала необходимо запустить контейнер, предназначенный для работы с сетью, например `nicolaka/netshoot`.

```
docker run --rm -it --name cont2 nicolaka/netshoot /bin/bash
```

Для наглядности создадим 2 контейнера и попробуем использовать утилиту ping в отношении этих двух контейнеров.



Мы не можем сделать пинг по dns, используя имя контейнера:

```
ping cont2
```

Создание контейнера в своей сети:

```
docker run --rm -it --name cont1 --network MyNetwork nicolaka/netshoot /bin/bash
```

Мы можем менять сеть в запущенных контейнерах:

```
docker network connect name_of_network name_of_cont
```

⚠ Если мы подключаем контейнер к другой сети у него сохраняется предыдущая сеть, чтобы отключить ее, пишем команду `docker inspect name_of_cont`. В консоль выведется параметры сети и какой id у каждой подключенной сети. Чтобы отключить контейнер от какой-либо сети пишем `docker network disconnect id(из параметров контейнера)`

Создание контейнера Dockerfile

Общий вид того, как выглядит dockerfile:

Файл с описанием инструкций для создания образа должен называться **Dockerfile**.

Dockerfile:

1. FROM, CMD, ENTRYPOINT

Инструкция FROM в Dockerfile используется для указания базового образа, на основе которого будет создаваться новый образ контейнера. Она является обязательной и должна быть первой инструкцией в Dockerfile.

CMD/ENTRYPOINT в Dockerfile задает команду, которая будет выполнена по умолчанию при запуске контейнера, если не указана другая команда в командной строке при запуске контейнера. Обычно в Dockerfile может быть только одна инструкция CMD/ENTRYPOINT, и она определяет команду, которая будет автоматически выполнена после запуска контейнера.

```
FROM ubuntu:22.04
CMD echo "hello world"
```

Правильнее писать в другом формате:

```
FROM ubuntu:22.04
CMD ["echo", "hello world"]
```

Чтобы собрать образ на основе этого Dockerfile пишем команду:

```
docker build -t NAMVE:version .
```

Для запуска контейнера используем команду `docker run NAMVE:version`

🔗 [Чем ENTRYPOINT отличается от CMD?](#)

ENTRYPOINT - это фиксированная команда, а CMD - можем перезаписать.

Например:

```
FROM ubuntu:22.04
ENTRYPOINT ["echo"]
CMD ["hello world"]
```

При сборке контейнера мы можем перезаписать вывод `docker run name_cont:v hello bruh`.

2. LABEL

В LABEL мы можем передавать информацию об образе, авторе и т. д.

```
LABEL author=Timofey
LABEL type=demo
LABEL platform=university
```

Информацию о LABEL можно посмотреть командой `docker image inspect name | grep Label`.

3. RUN

RUN в Dockerfile используется для выполнения команд во время сборки образа контейнера. Каждая инструкция RUN создает новый слой в образе Docker. Это позволяет устанавливать и обновлять пакеты, загружать и компилировать зависимости, копировать файлы и выполнять другие команды в процессе сборки контейнера.

```
RUN apt-get update
RUN apt-get install nginx -y
```

Параметр `-y` мы указываем для соглашения со скачиванием `nginx`.

🔗 Чем RUN отличается от ENTRYPOINT/CMD?

ENTRYPOINT/CMD определяет команду, которая будет выполнена по умолчанию при запуске контейнера, а RUN используется для выполнения команд во время сборки образа контейнера.

4. EXPOSE

Написание EXPOSE является хорошей манерой, поскольку при отсутствии должной документации позволяет понять, какой порт лучше открывать и какой протокол использовать:

```
EXPOSE 80/tcp
EXPOSE 443/tcp
```

А также это позволяет запускать разработчикам контейнер с default порт (задуманной другим разработчиком) с помощью команды `docker run -d -P id`. То есть будет автоматический проброс портов. Но порт на сервере будет выбран **рандомно**.

5. COPY

Команда COPY в Dockerfile используется для копирования файлов и директорий из исходной файловой системы (хостовой системы, где выполняется сборка образа) в образ Docker.

```
COPY filesfordocker/index.html /var/www/html
```

Сначала указывается путь до файла на сервере, а затем куда мы этот файл положим в контейнере.

5. WORKDIR

С помощью команды WORKDIR мы можем указать рабочую директорию, в которую мы будем попадать при подключении к контейнеру.

```
WORKDIR /var/www/html
```

Команда WORKDIR в Dockerfile используется для установки рабочей директории (working directory) для всех последующих инструкций в Dockerfile. Рабочая директория представляет собой директорию внутри контейнера, где будут выполняться команды, указанные в Dockerfile.

🔗 Если указанная в команде WORKDIR директория не существует в контейнере, Docker автоматически создаст эту директорию внутри контейнера при его выполнении. Если директория уже существует, команда WORKDIR просто устанавливает ее в качестве рабочей директории.

Также эта команда упрощает Dockerfile, поскольку нам не нужно будет указывать полный путь до рабочей директории, при копировании файлов с компьютера можно указать просто *точку*. Однако стоит помнить, что команда WORKDIR в таком случае должна идти раньше COPY.

```
WORKDIR /var/www/html
COPY files/index.html .
```

6. ENV

Команда ENV в Dockerfile используется для установки переменных среды (environment variables) в контейнере. Переменные среды представляют собой значения, которые могут использоваться внутри контейнера для настройки окружения, передачи конфигурационных параметров или определения других параметров, необходимых для работы приложения в контейнере.

```
ENV OWNER=Timofey
```

можем подключиться к контейнеру и посмотреть переменные среды командой `env`.

Менять переменные можно при запуске контейнера:

```
docker run -d --rm -e OWNER=TIM id
```

Если указать переменную которой нет в контейнере, то она создастся автоматички.

Контейнер с nginx:

```
FROM ubuntu:22.04

LABEL author=Timofey

RUN apt-get update

RUN apt-get install nginx -y

CMD ["nginx", "-g", "daemon off;"]
```

Мы можем подключиться к контейнеру и проверить запущенные процессы командой `ps xa`

```
> docker exec -it e4 /bin/bash
root@e4c459a48716:/# ps xa
  PID TTY          STAT       TIME COMMAND
   1 ?        Ss         0:00 nginx: master process nginx -g daemon off;
   7 ?        S          0:00 nginx: worker process
   8 ?        S          0:00 nginx: worker process
   9 ?        S          0:00 nginx: worker process
  10 ?        S          0:00 nginx: worker process
  11 pts/0    Ss         0:00 /bin/bash
  19 pts/0    R+         0:00 ps xa
```

Чтобы проверить открытые порты на сервере мы можем использовать команду `netstat -tulpen`.

🔗 Важно

При создании контейнера с nginx необходимо сделать проброс портов `-p 80:80`

Проверка успешного запуска сервера nginx:

```
> curl -Li http://localhost:8080
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Sun, 25 Jun 2023 21:57:38 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Sun, 25 Jun 2023 21:39:56 GMT
Connection: keep-alive
ETag: "6498b42c-264"
Accept-Ranges: bytes

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    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>
```

Команда `curl -Li` выполняет HTTP-запрос к указанному URL и выводит информацию о ответе от сервера. Здесь параметры `-L` и `-i` используются для дополнительной настройки вывода.

- `-L` (или `--location`) указывает `curl` 'у следовать перенаправлениям, если сервер отправляет ответ с кодом 3xx (например, 301 Moved Permanently или 302 Found). Это позволяет получить информацию о конечном URL после перенаправления.
- `-i` (или `--include`) указывает `curl` 'у включить заголовки ответа в вывод. Это включает заголовок HTTP-статуса, заголовки ответа сервера и другую мета-информацию.

Сочетание этих параметров позволяет получить информацию о сервере и ответе, включая заголовки, перенаправления и конечный URL, к которому был выполнен запрос.

Можно подключить volumes и вывести свой сайт `docker run --name nginx -d -p 80:80 -v /home/timofej/Рабочий\ стол/demo-docker/filesfordocker:/var/www/html id` В файле `*filesfordocker*` должен лежать файл `index.html`.

Docker Compose

- Используется для управления одним или несколькими контейнерами
- Содержит инструкции по запуску контейнера(ов)
- Упрощает автоматизацию запуска контейнеров
- Описывает в YAML файле, имеет название `docker-compose.yml`

`unless-stopped` - имеет тоже состояние что и до перезагрузки сервера
`always` - контейнер всегда будет перезапускаться

Чтобы запустить `docker-compose` нужно в месте нахождения файла использовать команду `docker-compose up (-d)`. С параметром `-d` это запуск в `background`.

Если необходимо запустить несколько контейнеров, то мы можем указать параметр `depends_on`, он означает, что приложение запустится только тогда, когда запустятся другие два контейнера это может быть полезно, когда для работы приложения необходима например база данных, которая должна быть запущена до самого приложения.

Чтобы посмотреть логи `docker compose logs -f`

Чтобы остановить контейнеры команда `docker compose stop`

Portainer – Web UI для управления Docker.

Приложение, которое упрощает работу с контейнерами.

Скачивать необходимо версию CE, подробнее о скачивании можно найти в официальной документации [документация](#)



Перед запуском приложения необходимо добавить volumes `docker volume create portainer_data`

Это приложение можно запустить с помощью команды:

```
docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:latest
```

можем подключиться к portainer по url <https://127.0.0.1:9443>

⚠ Warning

Если будет запрещен доступ к сайту, нужно нажать расширенные настройки и все равно перейти