

[Войти на сайт](#)
[Зарегистрироваться](#)

Форум

Блоги

Статьи

Интернет-магазин

 Статьи

Ваша корзина пуста

[Главная](#)[О компании](#)[ИТ услуги](#)рекламное место
СВОБОДНО Системный...

3 614 подписчиков



Подписат...

Основные команды для Docker

22.09.2022

Основные команды для Docker

Docker - Полный курс Docker Для Начинающих [3 ЧАСА]

Docker - Полный курс Docker Для Начинающих [3 ЧАСА]



Информация

docker info - Информация обо всём в установленном Docker

docker history - История образа

docker tag - Дать тег образу локально или в registry

docker login - Залогиниться в registry

docker search - Поиск образа в registry

docker pull - Загрузить образ из Registry себе на хост

docker push - Отправить локальный образ в registry

Управления контейнерами

docker ps -a - Посмотреть все контейнеры

docker start container-name - Запустить контейнер

docker run container-name - создает новый контейнер и сразу включает его. **(подробнее отдельно)**

docker kill/stop container-name - Убить (SIGKILL) /Остановить (SIGTERM) контейнер

docker logs --tail 100 container-name - Вывести логи контейнера, последние 100 строк

docker inspect container-name - Вся инфа о контейнере + IP

docker rm container-name - Удалить контейнер (поле каждой сборки Dockerfile)

docker rm -f \$(docker ps -aq) - Удалить все запущенные и остановленные контейнеры

docker events container-name

`docker port container-name` - Показать публичный порт контейнера

`docker top container-name` - Отобразить процессы в контейнере

`docker stats container-name` - Статистика использования ресурсов в контейнере

`docker diff container-name` - Изменения в ФС контейнера

Управления образами

`docker build -t my_app .` - Билд контейнера в текущей папке, Скачивает все слои для запуска образа

`docker images` / `docker image ls` - Показать все образы в системе

`docker image rm` / `docker rmi image` - Удалить image

`docker commit <containerName/ID> lepkov/debian11slim:version3` - Создает образ из контейнера

`docker insert URL` - вставляет файл из URL в контейнер

`docker save -o backup.tar` - Сохранить образ в backup.tar в STDOUT с тегами, версиями, слоями

`docker load` - Загрузить образ в .tar в STDIN с тегами, версиями, слоями

`docker import` - Создать образ из .tar

`docker image history --no-trunc` - Посмотреть историю слоёв образа

`docker system prune -f` - Удалит все, кроме используемого (лучше не использовать на проде, ещё кстати из-за старого кеша может собираться старая версия контейнера)

Подробнее "docker run"

Синтаксис следующий: **`docker run [опции] название образа [команды][аргументы]`**.

`docker run -d -p 80:80 -p 22:22 debian:11.1-slim sleep infinity` (`--rm` удалит после закрытия контейнера, `--restart unless-stopped` добавит автозапуск контейнера) - Запуск контейнера интерактивно или как демона/detached (`-d`), Порты: слева хостовая система, справа в контейнере, пробрасывается сразу 2 порта 80 и 22, используется легкий образ Debian 11 и команда бесконечный сон

`docker update --restart unless-stopped redis` - добавит к контейнеру правило перезапускаться при закрытии, за исключением команды стоп, автозапуск по-сути

`docker exec -it container-name /bin/bash` (ash для alpine) - Интерактивно подключиться к контейнеру для управления, exit чтобы выйти

`docker attach container-name` - Подключиться к контейнеру чтоб мониторить ошибки логи в реальном времени

Управления разделами (Volumes)

`docker cp file <containerID>:/` - Скопировать в корень контейнера file

`docker cp <containerID>:/file .` - Скопировать file из корня контейнера в текущую директорию командной строки

`docker volume create todo-db` - Создать volume для постоянного хранения файлов

`docker run -dp 3000:3000 --name=dev -v todo-db:/etc/todos container-name` - Добавить named volume todo-db к контейнеру (они ok когда мы не заморачиваемся где конкретно хранить данные)

`docker run -dp 3000:3000 --name=dev --mount source=todo-db,target=/etc/todos container-name` - тоже самое что команда сверху

`docker volume ls` - Отобразить список всех volume'ов

`docker volume inspect` - Инспекция volume'ов

`docker volume rm` - Удалить volume

Управления сетью (Network)

`docker network create todo-app` - Создать сеть

`docker network rm` - Удалить сеть

`docker network ls` - Отобразить все сети

`docker network inspect` - Вся информация о сети

`docker network connect` - Соединиться с сетью

`docker network disconnect` - Отсоединиться от сети

Пример 1: Hello world

Настало время запустить ваш первый контейнер:

`docker run ubuntu /bin/echo 'Hello world'`

Вывод консоли:

```
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
d54efb8db41d: Pull complete
f8b845f45a87: Pull complete
e8db7bf7c39f: Pull complete
9654c40e9079: Pull complete
6d9ef359eaaa: Pull complete
Digest: sha256:dd7808d8792c9841d0b460122f1acf0a2dd1f56404f8d1e56298048885e45535
Status: Downloaded newer image for ubuntu:latest
Hello world
```

`docker run` — команда, запускающая контейнер.

`ubuntu` — образ, который вы запускаете, например, образ ОС Ubuntu. Когда вы определяете его, Docker начинает поиск на Docker-хосте. Если образ не существует локально, тогда он будет вытянут из общественного реестра — Docker Hub.

`/bin/echo 'Hello world'` — команда, которая будет запущена внутри нового контейнера. Данный контейнер просто выведет «Hello world» и остановит выполнение.

Давайте попробуем создать интерактивную оболочку внутри Docker-контейнера:

`docker run -i -t -rm ubuntu /bin/bash`

- `t` — флаг, добавляющий псевдотерминал внутри нового контейнера.

- `i` — флаг, который позволяет установить интерактивное соединение, взяв стандартный ввод (STDIN) контейнера.

- `rm` — флаг, автоматически удаляющий контейнер после завершения процесса. По умолчанию контейнеры не удаляются. Этот контейнер существует, пока установлена текущая сессия оболочки, и уничтожается, когда мы покидаем сессию (например, SSH-сессию с удаленным сервером).

Если вы хотите оставить контейнер работающим после завершения сессии, превратите его в демон-процесс:

`docker run --name daemon ubuntu /bin/sh -c "while true; do echo hello world; sleep 1; done"`

--name `daemon` — устанавливает имя демона для нового контейнера. Если вы не определите имя, то Docker сгенерирует и присвоит его автоматически.

- `d` — флаг запускает контейнер в фоновом режиме (демонизирует его).

Давайте взглянем, какие контейнеры у нас есть на текущий момент:

`docker ps -a`

Вывод консоли:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1fc8cee64ec2	ubuntu	"/bin/sh -c 'while...'"	32 seconds ago	Up 30 seconds		daemon
c006f1a02edf	ubuntu	"/bin/echo 'Hello ...'"	About a minute ago	Exited (0) About a minute ago		gifted_nobel

`docker ps` — команда для вывода списка контейнеров.

- `a` — флаг, который показывает все контейнеры (без этого флага `ps` выведет только работающие контейнеры).

Команда **ps** показала, что у нас всего два контейнера:

gifted_nobel (имя для этого контейнера сгенерировано автоматически, оно будет другим на вашем компьютере) – это первый контейнер, который мы создали, выводит один раз «Hello world».
daemon – это третий контейнер, который мы создали, работает как фоновый процесс.

Замечание: здесь нет второго контейнера (с интерактивной оболочкой), потому что мы установили флаг **--rm**. Как следствие, этот контейнер автоматически удалится сразу после выполнения.

Давайте проверим логи и посмотрим, что фоновый контейнер делает прямо сейчас:

docker logs -f daemon

Вывод консоли:

```
. . .
hello world
hello world
hello world
```

docker logs – объединяет логи контейнера.

-f – флаг для слежения за выводом логов (работает так же, как **tail -f**).

Теперь остановим фоновый контейнер:

docker stop daemon

Удостоверимся, что контейнер остановился:

docker ps -a

Вывод консоли:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1fc8cee64ec2	ubuntu	"/bin/sh -c 'while...'"	5 minutes ago	Exited (137) 5 seconds ago		daemon
c006f1a02edf	ubuntu	"/bin/echo 'Hello ...'"	6 minutes ago	Exited (0) 6 minutes ago		gifted_nobel

Контейнер остановился. Мы можем запустить его заново:

docker start daemon

Удостоверимся, что он работает:

docker ps -a

Вывод консоли:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1fc8cee64ec2	ubuntu	"/bin/sh -c 'while...'"	5 minutes ago	Up 3 seconds		daemon
c006f1a02edf	ubuntu	"/bin/echo 'Hello ...'"	6 minutes ago	Exited (0) 7 minutes ago		gifted_nobel

Теперь остановим его опять и удалим все контейнеры вручную:

docker stop daemon

docker rm <your first container name>

docker rm daemon

Чтобы удалить все контейнеры, мы можем использовать следующую команду:

docker rm -f \$(docker ps -aq)

docker rm – команда для удаления контейнера.

-f – флаг для **rm**, который останавливает контейнер, если он работает (принудительное удаление).

-q – флаг для **ps**, который выводит только ID контейнеров.

Пример 2: Nginx

Настало время создать и запустить более полезный контейнер типа Nginx.

Сменим директорию на **examples/nginx**:

docker run -d --name test-nginx -p 80:80 -v \$(pwd):/usr/share/nginx/html:ro nginx:latest

Вывод консоли:

```
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
693502eb7dfb: Pull complete
6dec850d2bc: Pull complete
c3e19f087ed6: Pull complete
Digest: sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe335
Status: Downloaded newer image for nginx:latest
436a602273b0ca687c61cc843ab28163c720a1810b09005a36ea06f005b0c971
```

-p – преобразование портов HOST PORT:CONTAINER PORT.

-v – монтирование тома HOST DIRECTORY:CONTAINER DIRECTORY.

Важно: команда `run` принимает только абсолютные пути. В нашем примере мы использовали `$(pwd)`, чтобы установить абсолютный путь текущей директории.

Теперь вы можете открыть localhost в вашем браузере.

Или можно изменить `/example/nginx/index.html` (который смонтирован как том в `/usr/share/nginx/html` внутри контейнера) и обновить страницу.

Получим информацию о контейнере `test-nginx`:

docker inspect test-nginx

Эта команда показывает широкую системную информацию об установке Docker. Эта информация включает в себя версию ядра, количество контейнеров и образов, открытые порты, смонтированные тома и так далее.

Пример 3: написание Dockerfile

Чтобы создать Docker-образ, вам необходимо создать **Dockerfile**. Это просто текстовый файл с инструкциями и аргументами. Вот описание инструкций, которые мы будем использовать в нашем следующем примере:

FROM – устанавливает базовый образ.

RUN – выполняет команду в контейнере.

ENV – устанавливает переменную окружения.

WORKDIR – устанавливает рабочую директорию.

VOLUME – создает точку монтирования для тома.

CMD – устанавливает исполняемый файл для контейнера.

Вы можете просмотреть [справку по Dockerfile](#), чтобы узнать больше подробностей.

Теперь создадим образ, который будет получать содержимое сайта с помощью `curl` и записывать его в текстовый файл. Нам надо передать URL сайта через переменную окружения `SITE_URL`. Результирующий файл будет помещен в директорию, смонтированную в качестве тома:

```
FROM ubuntu:latest
RUN apt-get update
RUN apt-get install --no-install-recommends --no-install-suggests -y curl
ENV SITE_URL https://google.com/
WORKDIR /data
VOLUME /data
CMD sh -c "curl -L $SITE_URL > /data/results"
```

Dockerfile готов. Теперь настало время создать сам образ.

Перейдем в `examples/curl` и выполним следующую команду для создания образа:

```
docker build . -t test-curl
```

Вывод консоли:

```
Sending build context to Docker daemon 3.584 kB
Step 1/7 : FROM ubuntu:latest
----> 0ef2e08ed3fa
Step 2/7 : RUN apt-get update
----> Running in 4aa839bb46ec
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:2 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
...
Fetched 24.9 MB in 4s (5208 kB/s)
Reading package lists...
----> 35ac5017c794
Removing intermediate container 4aa839bb46ec
Step 3/7 : RUN apt-get install --no-install-recommends --no-install-suggests -y curl
----> Running in 3ca9384ecf8d
```

```

Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed...
--> f3c6d26b95e6
Removing intermediate container 3ca9384ecf8d
Step 4/7 : ENV SITE_URL https://google.com/
--> Running in 21b0022b260f
--> 9a733ee39a46
Removing intermediate container 21b0022b260f
Step 5/7 : WORKDIR /data
--> c024301ddfb8
Removing intermediate container 3bc973e5584c
Step 6/7 : VOLUME /data
--> Running in a9594a8958fe
--> 6802707a7114
Removing intermediate container a9594a8958fe
Step 7/7 : CMD sh -c "curl -L $SITE_URL > /data/results"
--> Running in 37503bc4e386
--> 5ebb2a65d771
Removing intermediate container 37503bc4e386
Successfully built 5ebb2a65d771

```

docker bulid – создает новый локальный образ.

-t – устанавливает именную метку для образа.

Теперь у нас есть новый образ, и мы можем посмотреть список существующих:

docker images

Вывод консоли:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
test-curl	latest	5ebb2a65d771	37 minutes ago	180 MB
nginx	latest	6b914bbcb89e	7 days ago	182 MB
ubuntu	latest	0ef2e08ed3fa	8 days ago	130 MB

Мы можем создать и запустить контейнер из образа. Попробуем сделать это со стандартными параметрами:

docker run --rm -v \$(pwd)/vol:/data:rw test-curl

Чтобы увидеть результаты, сохраненные в файле, выполним команду:

cat ./vol/results

Попробуем с facebook.com:

docker run --rm -e SITE_URL=https://facebook.com/ -v \$(pwd)/vol:/data:rw test-curl

И снова посмотрим результаты:

cat ./vol/results

Пример 4: Разворачиваем PROXY / ZABBIX / AGENT в docker контейнере

[Docker Compose](#) – это инструмент, который позволяет определить и запустить несколько связанных контейнеров Docker как одно приложение. Он основан на файле конфигурации в формате YAML, который описывает различные сервисы, сети и объемы данных, необходимые для запуска и настройки контейнеров. С помощью Docker Compose вы можете определить множество контейнеров, которые взаимодействуют друг с другом, и описать их конфигурацию в файле 'docker-compose.yml'. В этом файле вы указываете образы Docker, сети, порты, переменные окружения, объемы и другие параметры для каждого сервиса, который хотите запустить.

Docker Compose позволяет вам определить зависимости между сервисами, настроить сетевое взаимодействие между контейнерами и упростить процесс развертывания и управления многоконтейнерными приложениями. Он также предоставляет команды для создания, запуска и остановки всех контейнеров, описанных в файле 'docker-compose.yml'.

При использовании Docker Compose вы можете создавать и развертывать комплексные приложения, состоящие из нескольких сервисов, таких как веб-серверы, базы данных, кэш-серверы и другие. Это упрощает разработку, тестирование и развертывание приложений в среде Docker.

Данный пример представляет файл конфигурации Docker Compose (docker-compose-zabbix-proxy.yml) для развертывания двух сервисов: zabbix-proxy и zabbix-agent:

Создадим файл docker-compose.yml:

```
version: '3.5'
```

services:

proxy:

image: zabbix/zabbix-proxy-sqlite3:alpine-latest

ports:

- 10051:10051

volumes:

- /usr/lib/zabbix/externalscripts:/usr/lib/zabbix/externalscripts:ro

- /var/lib/zabbix/modules:/var/lib/zabbix/modules:ro

- /var/lib/zabbix/enc:/var/lib/zabbix/enc:ro

- /var/lib/zabbix/ssh_keys:/var/lib/zabbix/ssh_keys:ro

- /var/lib/zabbix/ssl/certs:/var/lib/zabbix/ssl/certs:ro

- /var/lib/zabbix/ssl/keys:/var/lib/zabbix/ssl/keys:ro

- /var/lib/zabbix/ssl/ssl_ca:/var/lib/zabbix/ssl/ssl_ca:ro

- /var/lib/zabbix/snmptraps:/var/lib/zabbix/snmptraps:ro

- /var/lib/zabbix/mibs:/var/lib/zabbix/mibs:ro

restart: always

environment:

- ZBX_HOSTNAME=zabbix-proxy.linuxshop

- ZBX_SERVER_HOST=zabbix.bsd.su

- ZBX_SERVER_PORT=10051

- ZBX_DEBUGLEVEL=1

- ZBX_PROXYMODE=1 # 0 - active proxy and 1 - passive proxy.

- ZBX_CONFIGFREQUENCY=60

- ZBX_STARTVMWARECOLLECTORS=2

- ZBX_VMWAREFREQUENCY=60

- ZBX_VMWARECACHESIZE=8M

agent:

image: zabbix/zabbix-agent:alpine-latest

ports:

- "10050:10050"

volumes:

- /etc/localtime:/etc/localtime:ro

- /etc/timezone:/etc/timezone:ro

- /proc:/proc

- /sys:/sys

- /dev:/dev

- /var/run/docker.sock:/var/run/docker.sock

privileged: true

pid: "host"

restart: always

depends_on:

- proxy

environment:

- ZBX_SERVER_HOST=proxy

- ZBX_DEBUGLEVEL=1

Данный пример представляет файл конфигурации Docker Compose (docker-compose-zabbix-проxy.yml) для разворачивания двух сервисов: proxy-zabbix и agent-zabbix. Давайте разберем каждый сервис по отдельности:

Сервис proxy-zabbix: имя для обращения внутри докера proxy

Использует образ [zabbix/zabbix-proxy-sqlite3:alpine-latest](#) из Docker Hub.

Открывает порт 10051 на хосте и перенаправляет его на порт 10051 внутри контейнера.

Примонтированы следующие тома:

```
/usr/lib/zabbix/externalscripts на /usr/lib/zabbix/externalscripts внутри контейнера в режиме только для чтения.
/var/lib/zabbix/modules на /var/lib/zabbix/modules внутри контейнера в режиме только для чтения.
/var/lib/zabbix/enc на /var/lib/zabbix/enc внутри контейнера в режиме только для чтения.
/var/lib/zabbix/ssh_keys на /var/lib/zabbix/ssh_keys внутри контейнера в режиме только для чтения.
/var/lib/zabbix/ssl/certs на /var/lib/zabbix/ssl/certs внутри контейнера в режиме только для чтения.
/var/lib/zabbix/ssl/keys на /var/lib/zabbix/ssl/keys внутри контейнера в режиме только для чтения.
/var/lib/zabbix/ssl/ssl_ca на /var/lib/zabbix/ssl/ssl_ca внутри контейнера в режиме только для чтения.
/var/lib/zabbix/snmptraps на /var/lib/zabbix/snmptraps внутри контейнера в режиме только для чтения.
/var/lib/zabbix/mibs на /var/lib/zabbix/mibs внутри контейнера в режиме только для чтения.
```

В случае сбоя контейнер будет автоматически перезапущен.

Определены следующие переменные окружения:

```
ZBX_HOSTNAME установлено в zabbix-proxy.linuxshop.
ZBX_SERVER_HOST установлено в zabbix.bsd.su
ZBX_SERVER_PORT установлено в 10051.
ZBX_DEBUGLEVEL установлено в 1.
ZBX_PROXYMODE установлено в 1 (режим пассивного прокси).
ZBX_CONFIGFREQUENCY установлено в 60.
```

Сервис agent-zabbix имя для обращения внутри докера agent:

Использует образ [zabbix/zabbix-agent:alpine-latest](#) из Docker Hub.

Открывает порт 10050 на хосте и перенаправляет его на порт 10050 внутри контейнера.

Примонтированы следующие тома:

```
/etc/localtime на /etc/localtime внутри контейнера в режиме только для чтения.
/etc/timezone на /etc/timezone внутри контейнера в режиме только для чтения.
/proc на /proc внутри контейнера.
/sys на /sys внутри контейнера.
/dev на /dev внутри контейнера.
/var/run/docker.sock на /var/run/docker.sock внутри контейнера.
```

Установлены следующие параметры:

```
privileged установлено в true.
pid установлено в "host".
```

В случае сбоя контейнер будет автоматически перезапущен.

Зависит от сервиса proxy.

Определены следующие переменные окружения:

```
ZBX_SERVER_HOST установлено в proxy. (отсылает данные на proxy внутри докер)
ZBX_DEBUGLEVEL установлено в 1.
```

Установка docker и docker-compose

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu `lsb_release -cs` test"
sudo apt update
apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

<https://docs.docker.com/engine/install/ubuntu/>

Старт zabbix-proxy

```
sudo docker-compose -f docker-compose-zabbix-proxy.yml up -d
```

up - это команда docker-compose, которая запускает приложение на основе файла конфигурации. Она создает и запускает контейнеры, сети и другие ресурсы, определенные в файле конфигурации.

-d - это флаг команды docker-compose, который указывает, чтобы приложение запустилось в фоновом режиме (detached mode). В этом режиме контейнеры будут работать в фоновом режиме, и вывод команды не будет отображаться в терминале.

Остановка


```
sudo docker-compose -f docker-compose-zabbix-proxy.yml down
```

Как обновить существующие образы с помощью docker-compose?

1. `docker-compose pull`
2. `docker-compose up -d --remove-orphans`
3. `docker image prune`

В вашей директории должен быть файл `docker-compose.yml` прим. выше.

Команда `'docker-compose pull'` используется для загрузки обновленных образов Docker, указанных в файле `'docker-compose.yml'`. Эта команда проверяет наличие более новых версий образов на Docker Hub или в другом удаленном реестре и загружает их на вашу локальную машину.

Команда `'docker-compose up -d --remove-orphans'` используется для запуска Docker-контейнеров на основе файла `'docker-compose.yml'` в фоновом режиме (detached mode) и удаления любых орфанных контейнеров. Вот что означают опции:

- `'-d'`: Эта опция указывает Docker Compose запустить контейнеры в фоновом режиме, без привязки к терминалу.
- `'--remove-orphans'`: Эта опция указывает Docker Compose удалить все контейнеры, которые не находятся в конфигурации `'docker-compose.yml'`. Она удаляет орфанные контейнеры, которые могут оставаться после предыдущего запуска или удаления контейнеров.

Команда `'docker image prune'` используется для удаления неиспользуемых образов Docker на вашем хост-системе. Она автоматически удаляет образы, которые не используются ни одним контейнером или другими образами. Эта команда помогает освободить дисковое пространство, занимаемое неактуальными образами.

Концепция Docker

У Docker есть некоторые ограничения и требования, которые зависят от архитектуры системы (приложений, которые вы упаковываете в контейнеры). Можно игнорировать эти требования или найти какие-нибудь пути обхода, но в этом случае вы не получите все преимущества использования Docker. Настоятельно рекомендуется следовать следующим советам:

- 1 приложение = 1 контейнер.
- Запускайте процесс на переднем плане (не используйте `systemd`, `upstart` или другие похожие инструменты).
- Для хранения данных вне контейнера используйте тома.
- Не используйте SSH (если вам надо залезть внутрь контейнера, используйте **docker exec**).
- Избегайте ручных настроек или действий внутри контейнера.
- Включайте только необходимый контекст – используйте `.dockerignore` файл (как `.gitignore` в git).
- Избегайте установки ненужных пакетов – это займет лишнее дисковое пространство.
- Используйте кэш. Добавьте контекст, который часто меняется, например, исходный код вашего проекта, в конец `Dockerfile` – кэш Docker будет использоваться более эффективно.
- Будьте осторожны с томами. Вы должны помнить, какие данные находятся в томах. Поскольку тома постоянны и не исчезают вместе с контейнерами, следующий контейнер будет использовать данные, которые были созданы предыдущим контейнером.
- Используйте переменные окружения: `RUN`, `EXPOSE`, `VOLUME`. Это сделает ваш `Dockerfile` более гибким.

[< Назад к списку новостей](#)

[Системный администратор.](#)

Контакты: +7 926 366 9191

mail: filippov.bsd@gmail.com



 Рейтинг@Mail.ru

[Карта сайта](#)