

Docker: команды

Интернет · 21.02.2026 · PDF-версия статьи с сайта cms.dolnais.ru

Шпаргалка с командами Docker

Перед тем как перейти к практике, важно познакомиться с базовыми понятиями, которые используются в Docker:

- **Контейнер** — это изолированная среда, в которой запускается приложение. Каждый контейнер содержит все необходимое для работы программы: код, библиотеки, зависимости и настройки. Благодаря изоляции, контейнеры не мешают друг другу и не влияют на основную систему.
- **Образ** — это шаблон для создания контейнера. В него входит приложение и все, что нужно для его запуска. Из одного образа можно запустить сколько угодно контейнеров. Готовые образы можно найти на Docker Hub или создать свои.
- **Dockerfile** — это текстовый файл с инструкциями, по которым собирается образ. В нем прописывается, какие компоненты нужно установить, какие файлы скопировать и какие команды выполнить для подготовки среды.
- **Docker Hub** — это онлайн-репозиторий, где хранятся тысячи готовых образов для различных программ и сервисов. Вы можете скачать нужный образ или загрузить собственный.
- **Репозиторий** — это пространство для хранения и управления образами. Как правило, репозитории размещаются на Docker Hub, но можно использовать и частные решения.
- **Контейнеризация** — процесс упаковки приложения и всех его зависимостей в контейнер. Так оно запускается одинаково на любой системе, независимо от настроек окружения.
- **Том** — это способ хранить данные вне контейнера. Даже если контейнер будет удален, данные в томе сохранятся. Подходит для работы с базами данных и другими постоянными файлами.

Создание контейнера

```
docker create -t -i eon01/infinite --name infinite
```

Первый запуск контейнера

```
docker run -it --name infinite -d eon01/infinite
```

Переименование контейнера

```
docker rename infinite infinity
```

Удаление контейнера

```
docker rm infinite
```

Обновление контейнера

```
docker update --cpu-shares 512 -m 300M infinite
```

Запуск остановленного контейнера

```
docker start nginx
```

Остановка

```
docker stop nginx
```

Перезагрузка

```
docker restart nginx
```

Пауза (приостановка всех процессов контейнера)

```
docker pause nginx
```

Снятие паузы

```
docker unpause nginx
```

Блокировка (до остановки контейнера)

```
docker wait nginx
```

Отправка SIGKILL (завершающего сигнала)

```
docker kill nginx
```

Отправка другого сигнала

```
docker kill -s HUP nginx
```

Подключение к существующему контейнеру

```
docker attach nginx
```

Информация о контейнерах

```
docker ps
```

```
docker ps -a
```

Логи контейнера

```
docker logs infinite
```

Информация о контейнере

```
docker inspect infinite
```

```
docker inspect --format '{{ .NetworkSettings.IPAddress }}' $(docker ps -q)
```

События контейнера

```
docker events infinite
```

Публичные порты

```
docker port infinite
```

Выполняющиеся процессы

```
docker top infinite
```

Использование ресурсов

```
docker stats infinite
```

Изменения в файлах или директориях файловой системы контейнера

```
docker diff infinite
```

Управление образами

Список образов

```
docker images
```

Создание образов

```
docker build .
```

```
docker build github.com/creack/docker-firefox
```

```
docker build - < Dockerfile
```

```
docker build - < context.tar.gz
```

```
docker build -t eon/infinite .
```

```
docker build -f myOtherDockerfile .
```

```
curl example.com/remote/Dockerfile | docker build -f - .
```

Удаление образа

```
docker rmi nginx
```

Загрузка репозитория в tar (из файла или стандартного ввода)

```
docker load < ubuntu.tar.gz
```

```
docker load --input ubuntu.tar
```

Сохранение образа в tar-архив

```
docker save busybox > ubuntu.tar
```

Просмотр истории образа

```
docker history
```

Создание образа из контейнера

```
docker commit nginx
```

Тегирование образа

```
docker tag nginx eon01/nginx
```

Push (загрузка в реестр) образа

```
docker push eon01/nginx
```

Создание сети

```
docker network create -d overlay MyOverlayNetwork
```

```
docker network create -d bridge MyBridgeNetwork
```

```
docker network create -d overlay \  
  --subnet=192.168.0.0/16 \  
  --subnet=192.170.0.0/16 \  
  --gateway=192.168.0.100 \  
  --gateway=192.170.0.100 \  
  --ip-range=192.168.1.0/24 \  
  --aux-address="my-router=192.168.1.5" --aux-address=  
"my-switch=192.168.1.6" \  
  --aux-address="my-printer=192.170.1.5" --aux-address="my-nas=192.170.1.6" \  
  MyOverlayNetwork
```

Удаление сети

```
docker network rm MyOverlayNetwork
```

Список сетей

```
docker network ls
```

Получение информации о сети

```
docker network inspect MyOverlayNetwork
```

Подключение работающего контейнера к сети

```
docker network connect MyOverlayNetwork nginx
```

Подключение контейнера к сети при его запуске

```
docker run -it -d --network=MyOverlayNetwork nginx
```

Отключение контейнера от сети

```
docker network disconnect MyOverlayNetwork nginx
```

Очистка Docker

Удаление работающего контейнера

```
docker rm nginx
```

Удаление контейнера и его тома (volume)

```
docker rm -v nginx
```

Удаление всех контейнеров со статусом exited

```
docker rm $(docker ps -a -f status=exited -q)
```

Удаление всех остановленных контейнеров

```
docker container prune
```

```
docker rm `docker ps -a -q`
```

Удаление контейнеров, остановленных более суток назад

```
docker container prune --filter "until=24h"
```

Удаление образа

```
docker rmi nginx
```

Удаление неиспользуемых (dangling) образов

```
docker image prune
```

```
docker rmi $(docker images -f dangling=true -q)
```

Удаление неиспользуемых (dangling) образов даже с тегами

```
docker image prune -a
```

Удаление всех образов

```
docker rmi $(docker images -a -q)
```

Удаление всех образов без тегов

```
docker rmi -f $(docker images | grep "^" | awk "{print $3}")
```

Остановка и удаление всех контейнеров

```
docker stop $(docker ps -a -q) && docker rm $(docker ps -a -q)
```

Удаление неиспользуемых (dangling) томов

```
docker volume prune
```

```
docker volume rm $(docker volume ls -f dangling=true -q)
```

Удаление неиспользуемых (dangling) томов по фильтру

```
docker volume prune --filter "label!=keep"
```

Удаление неиспользуемых сетей

```
docker network prune
```

Удаление всех неиспользуемых объектов

```
docker system prune
```

По умолчанию для Docker 17.06.1+ тома не удаляются. Чтобы удалились и они тоже:

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

Копирование файлов из контейнера Docker

Использование: `docker cp [OPTIONS] container:src_path dest_path`

- `container` — имя или идентификатор контейнера, из которого требуется скопировать.
- `src_path` — путь внутри контейнера к файлу или папке, которую требуется скопировать.
- `dest_path` — путь на хосте к файлу или папке, в которую требуется скопировать. Используйте `-` для указания стандартного вывода.

Например, скопируем файл `/etc/ssh/sshd_config` на хост (`f5732ee3d33c` — идентификатор контейнера):

```
docker cp f5732ee3d33c:/etc/ssh/sshd_config /root/ssh-changes  
Successfully copied 5.12kB to /root/ssh-changes
```

Скопируем папку `/var/log` на хост:

```
docker cp f5732ee3d33c:/var/log /root/ssh-changes  
Successfully copied 768kB to /root/ssh-changes
```

Скопируем файл `/etc/ssh/sshd_config` в `STDOUT`. Это просто как печать содержимого файла:

```
docker cp f5732ee3d33c:/etc/ssh/sshd_config -  
sshd_config00006440000000000000000000000000000000000000624714606114611011466 0ustar000000000000
```

Копирование файлов в контейнер Docker

Использование: `docker cp [OPTIONS] src_path container:dest_path`

- `container` — имя или идентификатор контейнера, в который требуется скопировать.
- `src_path` — путь к файлу или папке на хосте, которую требуется скопировать. Используйте - для обозначения стандартного ввода.
- `dest_path` — путь внутри контейнера к файлу или папке, в которую требуется скопировать.

Например, скопируем файл `/root/ssh-changes/sshd_config` в файл в контейнере `/etc/ssh/sshd_config` (f5732ee3d33c — идентификатор контейнера):

```
docker cp /root/ssh-changes/sshd_config f5732ee3d33c:/etc/ssh/sshd_config
Successfully copied 5.12kB to f5732ee3d33c:/etc/ssh/sshd_config
```

Скопируем папку `/root/ssh-changes/log` в папку контейнера `/tmp` (f5732ee3d33c — идентификатор контейнера):

```
docker cp /root/ssh-changes/log f5732ee3d33c:/tmp
Successfully copied 768kB to f5732ee3d33c:/tmp
```

Извлечём `tar` из хоста в контейнер (f5732ee3d33c — идентификатор контейнера):

```
docker cp - f5732ee3d33c:/tmp < /root/ssh-changes/apps.tar
Successfully copied 0B to f5732ee3d33c:/tmp
```

```
docker exec f5732ee3d33c ls -ld /tmp/apps
/tmp/apps
```

Опции

Опция: `-a, --archive`

С помощью этой опции команда `docker cp` может сохранять исходные разрешения, права, специальные атрибуты и временные метки копируемых файлов. Если копируется каталог, то всё содержимое каталога будет скопировано рекурсивно.

Опция: `-L, --follow-link`

С помощью опции команда `docker cp` может следовать символическим ссылкам (symlinks) во время процесса

копирования. При копировании каталога, содержащего символические ссылки, будет скопирован фактический файл или каталог, на который указывает символическая ссылка, а не сама символическая ссылка.

Опция: -q, --quiet

С помощью этой опции команда `docker cp` может автоматически подавлять вывод прогресса при копировании с подключением терминала. Вывод прогресса автоматически подавляется без подключения терминала. Это может быть удобно в сценарии или если вы не хотите видеть вывод прогресса.

Метод 2: Копирование с «хоста» на «хост»

Он обычно используется для копирования файлов или папок между контейнером и хостом. Но этот метод копирует только на хосте, например из одной папки хоста в другую папку.

На самом деле, файловая система контейнера также находится на хосте. Таким образом, мы можем получить доступ к файловой системе контейнера непосредственно с хоста. Но где находится файловая система контейнера на хосте?

Прежде чем начать, давайте посмотрим драйвер хранилища Docker, выполнив следующие команды:

```
docker info | grep Storage
Storage Driver: overlay2
```

Теперь запустим контейнер «hello».

```
docker run -dit --name hello ubuntu:latest
```

Выполним команду `docker inspect`:

```
# docker container inspect hello | jq '[0].GraphDriver'
{
  "Data": {
    "LowerDir": "/var/lib/docker/overlay2/1e04348e54577157f8945bfd16543b18910c6c9551e14ea2f3fd91cc29f200f6-init/diff:/ci_runner/dockerHome/overlay2/595b425d150311a9c2637426cdc200d9f7509a7f26653ee203b35c171e379e89/diff",
    "MergedDir": "/var/lib/docker/overlay2/1e04348e54577157f8945bfd16543b18910c6c9551e14ea2f3fd91cc29f200f6/merged",
    "UpperDir": "/var/lib/docker/overlay2/1e04348e54577157f8945bfd16543b18910c6c9551e14ea2f3fd91cc29f200f6/diff",
    "WorkDir": "/var/lib/docker/overlay2/1e04348e54577157f8945bfd16543b18910c6c9551e14ea2f3fd91cc29f200f6/work"
  },
  "Name": "overlay2"
}
```

Мы можем увидеть 4 вида слоёв данных:

- **LowerDir**: слои образа, доступные только для чтения, содержат оригинальное содержимое файловой системы, включая слои bootfs и rootfs. Эти слои образов могут быть собраны в стек. Слой LowerDir обеспечивает базовую структуру файловой системы контейнера.
- **UpperDir**: расположен над слоем LowerDir, доступен для чтения и записи. При запуске контейнера слой UpperDir автоматически создаётся поверх финального слоя образа. Все изменения данных контейнера происходят в этом слое, и он имеет собственное состояние данных, не влияя на базовый образ.
- **MergedDir**: объединяет LowerDir и UpperDir в единую структуру каталогов, обеспечивая единое представление контейнера. Изменения файлов в LowerDir или новые файлы, добавленные в UpperDir, будут перезаписаны или добавлены в MergedDir.
- **WorkDir**: определяет рабочий каталог по умолчанию в контейнере. Он может быть задан с помощью инструкции WORKDIR в Dockerfile. Определение рабочей директории может быть полезно для организации файлов внутри контейнера.

Эти 4 слоя в совокупности обеспечивают полную структуру файловой системы и рабочую среду для контейнеров. Обратим внимание на MergedDir, где мы можем увидеть все файлы внутри контейнера.

Перейдя

в `/var/lib/docker/overlay2/1e04348e54577157f8945bfd16543b18910c6c9551e14ea2f3fd91cc29f200f6/merged`, можно увидеть все файлы в контейнере.

```
cd /var/lib/docker/overlay2/1e04348e54577157f8945bfd16543b18910c6c9551e14ea2f3fd91cc29f200f6/merged
ls
bin    dev    home  lib32  libx32  mnt    proc   run    srv    tmp    var
boot   etc    lib   lib64  media   opt    root   sbin   sys    usr
```

Таким образом, копирование файла или папки аналогично операции копирования на хосте с помощью команд Linux `cp`, `гупс` и так далее.

Копирование файлов из контейнера Docker

Например: скопируем файл `/var/log/last` из контейнера на хост в `/tmp`:

```
cd /var/lib/docker/overlay2/1e04348e54577157f8945bfd16543b18910c6c9551e14ea2f3fd91cc29f200f6/merged
cp ./var/log/lastlog /tmp/
ls -l /tmp/lastlog
-rw-r--r-- 1 root root 29492 Apr 21 21:38 /tmp/lastlog
```

Копирование файлов в контейнер Docker

Например: скопируем файл из хоста в каталог `/root` контейнера:

```
echo "hello!" > /tmp/hello.test  
cd /var/lib/docker/overlay2/1e04348e54577157f8945bfd16543b18910c6c9551e14ea2f3  
fd91cc29f200f6/merged  
cp /tmp/hello.test ./root/
```

Давайте проверим это, выполнив команду:

```
docker exec -it hello cat /root/hello.test  
hello!
```

Отслеживание ресурсов контейнеров — `docker stats`

Чтобы узнать, сколько ресурсов (процессор, память, сеть и диск) использует каждый контейнер, используйте команду:

```
docker stats
```

Она покажет в реальном времени текущую нагрузку на систему от каждого запущенного контейнера.

Источник: <http://taskchat.ru/docker-komandy>