Управление данными в Docker

К докер контейнерам можно подключать внешние папки. Рассматриваем основные способы.

Время чтения: больше 15 мин

1. [Связанные папки (bind mounts)](https://doka.guide/tools/docker-data-management/#svyazannye-papki-bind-mounts)
   1. [Как пользоваться](https://doka.guide/tools/docker-data-management/#kak-polzovatsya)
2. [Тома (volumes)](https://doka.guide/tools/docker-data-management/#toma-volumes)
   1. [Как пользоваться](https://doka.guide/tools/docker-data-management/#kak-polzovatsya-1)
   2. [Использование драйверов](https://doka.guide/tools/docker-data-management/#ispolzovanie-drayverov)
   3. [Резервные копии](https://doka.guide/tools/docker-data-management/#rezervnye-kopii)
3. [Хранение в оперативной памяти](https://doka.guide/tools/docker-data-management/#hranenie-v-operativnoy-pamyati)
   1. [Как пользоваться](https://doka.guide/tools/docker-data-management/#kak-polzovatsya-2)
4. [На практике](https://doka.guide/tools/docker-data-management/#na-praktike)
   1. [Игорь Коровченко](https://doka.guide/tools/docker-data-management/#igor-korovchenko)

Авторы:

* [Игорь Коровченко](https://doka.guide/people/igsekor/)

Редакторы:

* [Ольга Алексашенко](https://doka.guide/people/tachisis/)

[Редактировать на  GitHub](https://github.com/doka-guide/content/blob/main/tools/docker-data-management)

Обновлено 17 мая 2022

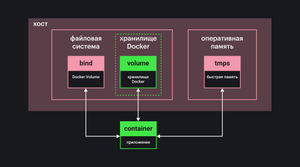
В этой статье мы поговорим про управление данными приложений в Docker. Узнать, что такое Docker, вы сможете из статьи «[Что такое Docker](https://doka.guide/tools/docker/)». Также вы можете почитать о [мультиконтейнерных приложениях и Docker Compose](https://doka.guide/tools/docker-compose/) и о том, [как устроен Dockerfile](https://doka.guide/tools/dockerfile/).

Итак, по умолчанию все данные приложения хранятся в контейнере Docker и после остановки контейнера теряются. Но это не единственный способ работать с данными. Можно использовать оперативную память и файловую систему компьютера, на котором установлен Docker Engine. Существует несколько типов хранилищ данных:

* связанные папки, примонтированные к контейнеру как внешние диски (*bind mounts*);
* тома (*volumes*);
* часть оперативной памяти для работы с данными (*tmpfs mounts* или *npipe mounts*).

Вне зависимости от того, какой тип хранилища вы выберете, данные для приложения будут храниться в заданной вами папке внутри контейнера. Технология работает бесшовно, но имеет свои накладные расходы для каждого конкретного типа.

Наглядная схема типов управления данными в Docker:



Рассмотрим каждый тип по отдельности.

**Связанные папки (bind mounts)**

[Секция статьи "Связанные папки (bind mounts)"](https://doka.guide/tools/docker-data-management/#svyazannye-papki-bind-mounts)

Связанные папки появились в Docker с самых первых релизов. Это удобный инструмент, но у него есть ограничения. Этот тип управления данными позволяет связать папку на компьютере пользователя (то есть хосте, на котором установлен Docker Engine) и папку в контейнере. Работать в контейнере и на хосте с такой папкой можно одновременно, все изменения будут отображаться и там, и там. Механизм bind mounts подразумевает, что данные могут быть изменены в любое время как из подключённого контейнера, так и непосредственно на хосте.

При создании связанной папки указывается полный путь к ней на хосте и путь внутри контейнера. Если папка не существует на хосте, Docker может создать её сам.

Связанные папки используются:

**Когда конфигурационные файлы на хосте и в контейнере одни и те же.** Именно этот тип использует сам Docker для автоматического монтирования конфигурации DNS хоста.

**Когда работаем с исходным кодом и артефактами сборок.** Можно использовать системы сборки для исходного кода внутри контейнера. Вы меняете код, [бандлер](https://doka.guide/tools/bundlers/), который находится внутри контейнера, это видит, и код попадает в новую сборку. Другой вариант использования — работа с уже собранными бандлами, например, для тестирования или отладки приложений.

**Когда необходимо обеспечить создание одной и той же файловой структуры на различных компьютерах.** Если папки на компьютере пользователя не существует, она будет создана при сборке образа и запуске контейнера.

**Как пользоваться**

[Секция статьи "Как пользоваться"](https://doka.guide/tools/docker-data-management/" \l "kak-polzovatsya)

Чтобы связать папку на хосте с папкой внутри контейнера, можно воспользоваться флагами -v или --mount. $(pwd) в командах ниже означает, что примонтируется текущая папка на хосте.

Пример с флагом -v:

docker run -d \

-it \

--name devtest \

-v "$(pwd)"/target:/app \

node:lts

Скопировать

Можно задать следующие опции: rprivate, private, rshared, shared, rslave, slave, ro, z и Z.

Первые шесть параметров позволяют управлять тем, как будут влиять изменения в одной точке монтирования тома на другие точки монтирования. По умолчанию используется rprivate, что означает — никак.

Последние три параметра могут быть указаны только для флага -v. Значение ro определяет режим только для чтения. Папка на хосте не может быть изменена внутри контейнера. Значение z обозначает, что папка на хосте может быть использована несколькими контейнерами. Значение Z обозначает, что папка используется только одним контейнером. Не указывайте значение Z для системных папок, например, */usr* или */home*. Это приведёт к тому, что работа операционной системы на хосте будет парализована. Будьте аккуратны!

Пример с флагом --mount:

docker run -d \

-it \

--name devtest \

--mount type=bind,source="$(pwd)"/target,target=/app \

node:lts

Скопировать

Ключ bind-propagation

Флаг --mount не поддерживает опции для управления метками [selinux](https://ru.wikipedia.org/wiki/SELinux) (z и Z).

Проверьте корректность работы хранилища с помощью команды:

docker inspect devtest

Скопировать

В соответствующей секции *Mounts* вы сможете найти исчерпывающую информацию. Например, если вы находились в папке */tmp/source/target* при запуске контейнера, то в этой секции будет указана примерно следующая информация:

"Mounts": [

{

"Type": "bind",

"Source": "/tmp/source/target",

"Destination": "/app",

"Mode": "",

"RW": true,

"Propagation": "rprivate"

}

],

Скопировать

Для разрыва связи между папками на хосте и в контейнере выполните команды остановки и удаления контейнера:

docker container stop devtest

docker container rm devtest

Скопировать

☝️

Помните:

1. Связанными папками нельзя управлять из Docker CLI.
2. Абсолютные пути на разных компьютерах могут быть разными.
3. Если в контейнере в примонтированной папке есть содержимое, то оно «перекроет» содержимое связанной папки на все время работы контейнера.
4. Использовать связанные папки для работы с конфигурационными файлами небезопасно.
5. Файловая система и структура папок могут сильно отличаться на разных компьютерах.
6. Правила описания путей к файлам могут отличаться при переходе от одной платформы к другой.
7. Вы можете столкнуться с ситуацией, когда приложение в контейнере получит доступ к системным папкам или удалит критически важные файлы.

**Тома (volumes)**

[Секция статьи "Тома (volumes)"](https://doka.guide/tools/docker-data-management/" \l "toma-volumes)

Тома — это лучший тип управления данных в Docker. Только объекты или службы Docker должны иметь права на изменение данных, расположенных в томах. На хосте данные хранятся в специальных папках, но без доступа администратора к ним не подобраться. В идеологии Docker тома — что-то вроде образа флэш-накопителя или CD/DVD.

Тома можно размещать не только на хосте. Можно, например, пользоваться облачными платформами для совместной работы с данными или для тестирования приложений. А ещё тома будут работать как с Linux-контейнерами, так и с Windows-контейнерами, поскольку файловая система томов одна и та же.

Когда том примонтирован к контейнеру, операционная система хоста не имеет к нему доступа. Docker управляет томами отдельно, позволяя подключаться одному или нескольким контейнерам одновременно. Плюсом является и то, что том существует самостоятельно и не зависит от жизненного цикла контейнеров.

Тома могут быть созданы при сборке контейнера (с помощью [Dockerfile](https://doka.guide/tools/docker/) или [Docker Compose](https://doka.guide/tools/docker-compose/)) или вручную с помощью Docker Engine. Тома могут иметь имя, назначенное пользователем (именованные тома, named volumes), а могут быть анонимными с именем, которое Docker устанавливает автоматически (анонимные тома, anonymous volumes).

Концепция драйвера позволяет преобразовывать данные в томах или влиять на потоки данных между томами и контейнерами. Например, это можно использовать для шифрования. Но чаще с помощью драйверов к контейнеру подключают тома, которые расположены не локально на хосте, а в облаке или на сервере. Это позволяет, не меняя логику работы приложения внутри контейнера, обрабатывать данные, которых на хосте нет.

Итак, возможности томов:

— миграция данных и создание резервных копий;  
— управление с помощью Docker CLI или Docker API;  
— тома работают и с Linux-, и с Windows-контейнерами;  
— данные легко и безопасно можно использовать в нескольких контейнерах;  
— существует механизм драйверов, который позволяет хранить данные не только на хосте, но и на сервере или в облаке, шифровать данные в томе или добавлять дополнительную функциональность;  
— новые тома могут создаваться с уже загруженными с помощью контейнера данными;  
— если на хосте установлены Mac или Windows, тома будут быстрее работать с Docker Desktop, чем связанные папки;  
— тома не увеличивают размер контейнера;  
— тома находятся вне жизненного цикла контейнера.

Тома используются:

**Когда нам нужно получить доступ к данным из разных контейнеров.** Том создаётся в первый раз либо вручную, либо при сборке контейнера. Уничтожается том всегда только с помощью Docker вручную. После остановки контейнера том будет продолжать работать, пока не будет удалён пользователем.

**Когда вы не уверены, что путь до папки будет одним и тем же на разных компьютерах.** Тома позволяют повысить уровень абстракции.

**Когда вы хотите хранить данные не только у себя на локальном компьютере, но и на сервере или в облаке.**

**Когда нужно создать резервную копию или перенести тома с одного компьютера на другой.** Тома хранятся в определённой папке на компьютере. Вы можете просто скопировать её, заархивировать и перенести на другой хост. Примерно так же создаётся и резервная копия.

**Если ваше приложение требует высокой скорости обмена данными на Mac и Windows.** Тома сохраняются на виртуальной машине Linux VM, на которой работают и контейнеры, поэтому скорость чтения и записи высокая. Нет лишних накладных расходов на доступ к файловой системе хоста.

**Когда важно, чтобы файловая система имела нативное поведение.** Например, база данных должна контролировать кэширование на диске для гарантии выполнения транзакций. Файловые системы на Mac и Windows работают не так, как на Linux. Это может привести к ошибкам работы некоторых приложений.

**Как пользоваться**

[Секция статьи "Как пользоваться"](https://doka.guide/tools/docker-data-management/#kak-polzovatsya-1)

Создать том можно с помощью флагов -v или --mount при запуске контейнера. Для флага -v можно указать параметр ro, который будет означать использование режима только для чтения. Для флага --mount есть ключ volume-opt, который устанавливает набор опций, разделённых запятыми. Не забывайте, что значения для этого ключа должны быть экранированы кавычками. Работа с томами такова, что изменения в одной точке монтирования в контейнере не будут отображаться в другой точке монтирования (параметр bind-propagation всегда выставлен в значение rprivate).

Подключить том с именем my-vol можно следующим образом.

С флагом --mount:

docker run -d \

--name devtest \

--mount source=my-vol,target=/app \

node:lts

Скопировать

С флагом -v:

docker run -d \

--name devtest \

-v my-vol:/app \

node:lts

Скопировать

Проверьте корректность результата выполнения команды:

docker inspect devtest

Скопировать

Чтобы удалить том, необходимо отключить связанный с ним контейнер и удалить сам контейнер:

docker container stop devtest

docker container rm devtest

docker volume rm my-vol

Скопировать

Управлять томами можно через Docker API с помощью Docker CLI и Docker Compose.

Чтобы создать новый том **с помощью Docker CLI**, используйте команду:

docker volume create my-vol

Скопировать

Получите список томов на хосте:

docker volume ls

Скопировать

Посмотрите информацию о томе:

docker volume inspect my-vol

Скопировать

Удалите том командой:

docker volume rm my-vol

Скопировать

Если том был анонимным, то можно удалить его сразу после завершения работы контейнера. Для этого при запуске контейнера вы можете прописать флаг --rm. Вместе с удалением контейнера в этом случае удалится и том:

docker run --rm -v /foo -v awesome:/bar container app

Скопировать

После завершения работы и последующего удаления контейнера анонимный том удалится, а именованный awesome продолжит работать.

Чтобы удалить все неиспользуемые тома, используйте команду:

docker volume prune

Скопировать

Для того, чтобы подключить том **с помощью Dockerfile**, необходимо использовать инструкцию VOLUME:

FROM node:lts

RUN useradd user

RUN mkdir /data && touch /data/x

RUN chown -R user:user /data

VOLUME /data

Скопировать

Интересно, что вы не сможете внести какие-либо изменения в данные на этапе сборки образа. Следующий *Dockerfile* правильно работать не будет:

FROM node:lts

RUN useradd user

VOLUME /data

RUN touch /data/x

RUN chown -R user:user /data

Скопировать

Том будет подключён только после создания образа на этапе запуска контейнера. Возможно, придётся использовать инструкции CMD или ENTRYPOINT. Подробнее описано в статье «[Как устроен Dockerfile](https://doka.guide/tools/dockerfile/)».

Запустить том для отдельного контейнера **с Docker Compose** можно с помощью следующей конфигурации:

services:

frontend:

image: node:lts

volumes:

- myapp:/home/node/app

volumes:

myapp:

Скопировать

Команда docker-compose up поднимет не только сам контейнер frontend, но и создаст том myapp. Если он уже был создан, Docker Compose подключит его к контейнеру, но надо указать это явно с помощью элемента external так:

services:

frontend:

image: node:lts

volumes:

- myapp:/home/node/app

volumes:

myapp:

external: true

Скопировать

Подробнее о формате конфигурации Docker Compose можно прочитать [в статье о Docker Compose](https://doka.guide/tools/docker-compose/).

**Использование драйверов**

[Секция статьи "Использование драйверов"](https://doka.guide/tools/docker-data-management/" \l "ispolzovanie-drayverov)

Когда приходит время масштабировать приложение, несколько сервисов должны работать с одним хранилищем данных. Для этого существует масса решений, и у Docker есть своё — драйверы для томов. Это лишь один пример использования драйверов. Можно организовать, например, пересылку данных между контейнерами с поддержкой шифрования или автоматическое шифрование и дешифровку всех данных в томе. Можно реализовать любой механизм обработки данных. Драйверы повышают уровень абстракции, позволяя отделить логику работы приложения от системы хранения данных.

Например, есть два компьютера — хост, на котором установлен Docker и запускаются контейнеры, и файловый сервер, который поставляет данные для них. Контейнеры ничего не знают про эту архитектуру: все запускалось изначально на локальном хосте. Драйвер vieux/sshfs позволяет использовать SSH-соединение для связи с файловым сервером, при этом данные будут представлены в виде тома Docker.

Для начала необходимо установить соответствующий плагин для Docker Engine:

docker plugin install --grant-all-permissions vieux/sshfs

Скопировать

Затем нужно создать том и прописать учётные данные:

docker volume create --driver vieux/sshfs \

-o sshcmd=test@node2:/home/test \

-o password=testpassword \

sshvolume

Скопировать

Если для связи по SSH между клиентом и сервером уже работают ключи доступа, то пароль можно опустить. Флаг -o указывает на опции, которые могут быть переданы драйверу. Набор доступных опций у каждого драйвера свой.

Можно создать том и другим способом, при запуске контейнера:

docker run -d \

--name sshfs-container \

--volume-driver vieux/sshfs \

--mount src=sshvolume,target=/app,volume-opt=sshcmd=test@node2:/home/test,volume-opt=password=testpassword \

nginx:latest

Скопировать

Если драйвер требует передачи опций, приходится использовать флаг --mount.

**Резервные копии**

[Секция статьи "Резервные копии"](https://doka.guide/tools/docker-data-management/#rezervnye-kopii)

Для того чтобы создать резервную копию тома, можно использовать механизм контейнеров Docker. Например, вы уже создали контейнер с именем dbstore на базе операционной системы Ubuntu и работаете с данными в томе dbdata. Для этого вы уже выполнили команду и получили доступ к терминалу контейнера:

docker run -v /dbdata --name dbstore node:lts /bin/bash

Скопировать

Как создать резервную копию данных в томе? Нужно:

— запустить новый контейнер и примонтировать том, который используется в контейнере dbstore;  
— примонтировать папку на хосте, чтобы потом в неё положить резервную копию;  
— зайти внутри контейнера в том, заархивировать данные и положить их в связанную папку.

Выполните команду:

docker run --rm --volumes-from dbstore -v $(pwd):/backup ubuntu tar cvf /backup/backup.tar /dbdata

Скопировать

После завершения архивации контейнер выключится и удалится, а резервная копия останется у вас в папке, из которой вы запускали команду.

Допустим, у вас возникла необходимость развернуть данные из сохранённой резервной копии внутри контейнера dbstore2. Нужно запустить его:

docker run -v /dbdata --name dbstore2 node:lts /bin/bash

Скопировать

Затем разархивировать данные в том:

docker run --rm --volumes-from dbstore2 -v $(pwd):/backup ubuntu bash -c "cd /dbdata && tar xvf /backup/backup.tar --strip 1"

Скопировать

**Хранение в оперативной памяти**

[Секция статьи "Хранение в оперативной памяти"](https://doka.guide/tools/docker-data-management/#hranenie-v-operativnoy-pamyati)

Хранение в оперативной памяти бывает двух типов: *tmpfs mounts* и *npipe mounts*.

Механизм *tmpfs mount* в операционной системе Linux позволяет выделить часть оперативной памяти хоста для хранения данных. Данные не сохраняются в файловой системе, и получается быстрое хранилище. Примонтированная папка *tmpfs* работает, пока запущен контейнер, поэтому не стоит использовать этот способ для хранения настроек и результатов работы приложения.

Для пользователей операционной системы Windows существует ещё один тип управления данными — *npipe mount*. Этот тип позволяет получить доступ к хосту Docker из контейнера и в основном используется для управления данными с Docker Engine API.

Используем оперативную память:

**Если вы не хотите оставлять данные после завершения работы приложения.**

**Как пользоваться**

[Секция статьи "Как пользоваться"](https://doka.guide/tools/docker-data-management/#kak-polzovatsya-2)

Этот раздел посвящён использованию только на Linux.

С помощью томов и связанных папок вы можете делиться файлами между хостом и контейнером. После остановки контейнера данные сохраняются. Но если на хосте используется операционная система Linux, то существует и третий тип работы с данными — *tmpfs*. Это временное файловое хранилище, которое располагается в оперативной памяти, присутствует во многих Unix-подобных системах. Когда вы создаёте контейнер, Docker может создать отдельный слой в оперативной памяти снаружи контейнера для хранения и обработки данных.

При использовании этого типа работы с данными в Docker есть два ограничения:

— операционной системой хоста может быть только Linux;  
— данные в *tmpfs* доступны лишь из одного контейнера.

*tmpfs* хорошо работает в случае хранения чувствительной информации: ключей шифрования, паролей, сертификатов доступа и тому подобного.

Чтобы запустить контейнер с *tmpfs*, используют команду:

docker run -d \

-it \

--name tmptest \

--mount type=tmpfs,destination=/app \

node:lts

Скопировать

С помощью ключа tmpfs-size можно определить максимальный размер хранилища в байтах. По умолчанию он не ограничен. Ключ tmpfs-mode служит для определения уровня доступа в восьмеричном формате. Например, значение по умолчанию 1777 обозначает, что любой пользователь или программа в контейнере имеют неограниченный доступ к данным, которые будут доступны и вне контейнера. Этот параметр работает также, как и для *[tmpfs](https://man.archlinux.org/man/tmpfs)*[в Unix-подобных операционных системах](https://man.archlinux.org/man/tmpfs).

Также есть альтернативная более короткая команда для управления *tmpfs mounts*:

docker run -d \

-it \

--name tmptest \

--tmpfs /app \

node:lts

Скопировать

Проверьте состояние контейнера, чтобы убедиться, что файловое хранилище создано корректно:

docker container inspect tmptest

Скопировать

В соответствующей секции будет доступна информация о примонтированной папке:

"Tmpfs": {

"/app": ""

},

Скопировать

Для удаления слоя с данными выполните команды остановки и удаления контейнера:

docker container stop tmptest

docker container rm tmptest

Скопировать

**На практике**

[Секция статьи "На практике"](https://doka.guide/tools/docker-data-management/#na-praktike)

[**Игорь Коровченко**](https://doka.guide/people/igsekor/)

[Секция статьи "Игорь Коровченко"](https://doka.guide/tools/docker-data-management/#igor-korovchenko)

✨ Вы можете получить данные из определённой папки контейнера, если примонтируете к ней **пустую** папку или **пустой** том на хосте. После монтирования данные автоматически окажутся доступны с хоста и останутся там после работы контейнера. Это удобный способ сделать бэкап или получить результаты работы приложения.

Если папка или том окажутся **не пустыми**, то при монтировании к контейнеру содержимое будет на время скрыто. Контейнер будет воспринимать эту папку как пустую, данные будут в неё сохраняться и будут доступны с хоста во время работы контейнера. После окончания работы контейнера или после того, как том или папка будут отмонтированы, данные из контейнера будут потеряны, поскольку снова будут доступны те файлы и подпапки, которые были скрыты при монтировании. Это тот же механизм, который Linux будет использовать, когда вы, например, примонтируете USB-накопитель к уже заполненной чем-то папке.

Top of Form

Статья была полезной?

Bottom of Form