Что такое Docker, и как его использовать? Подробно рассказываем

Разберем по косточкам, ведь Docker – это мощный инструмент, и огромное количество информации по работе с ним вряд ли уместится в брошюрку.

2[0](https://proglib.io/p/docker#comments)

Что такое Docker?

[Это ПО с открытым кодом](https://www.docker.com/), принцип работы которого проще всего сравнить с транспортными контейнерами. Только подумайте, ведь когда-то транспортные компании сталкивались с похожими проблемами:

1. Как перевозить разные (несовместимые) типы товаров вместе (например, продукты питания с химикатами или стекло с кирпичом)?
2. Как обрабатывать пакеты разных размеров одним и тем же транспортным средством?

С введением контейнеров стало возможным перевозить вместе кирпичи и стекло, химикаты и еду, а также многое другое. Груз разного размера может быть распределен по стандартизированным контейнерам, которые загружаются/выгружаются одним и тем же транспортным средством.

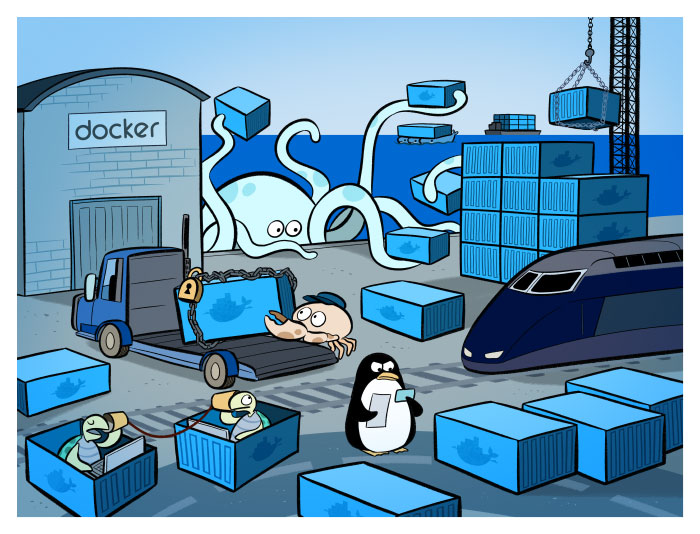
Но вернемся же к контейнерам. Когда вы разрабатываете приложение, вам нужно предоставить код вместе со всеми его составляющими, такими как библиотеки, сервер, базы данных и т. д. Вы можете оказаться в ситуации, когда приложение работает на вашем компьютере, но отказывается включаться на устройстве другого пользователя.

Эта проблема решается через создание независимости ПО от системы.

В чем отличие от виртуализации?

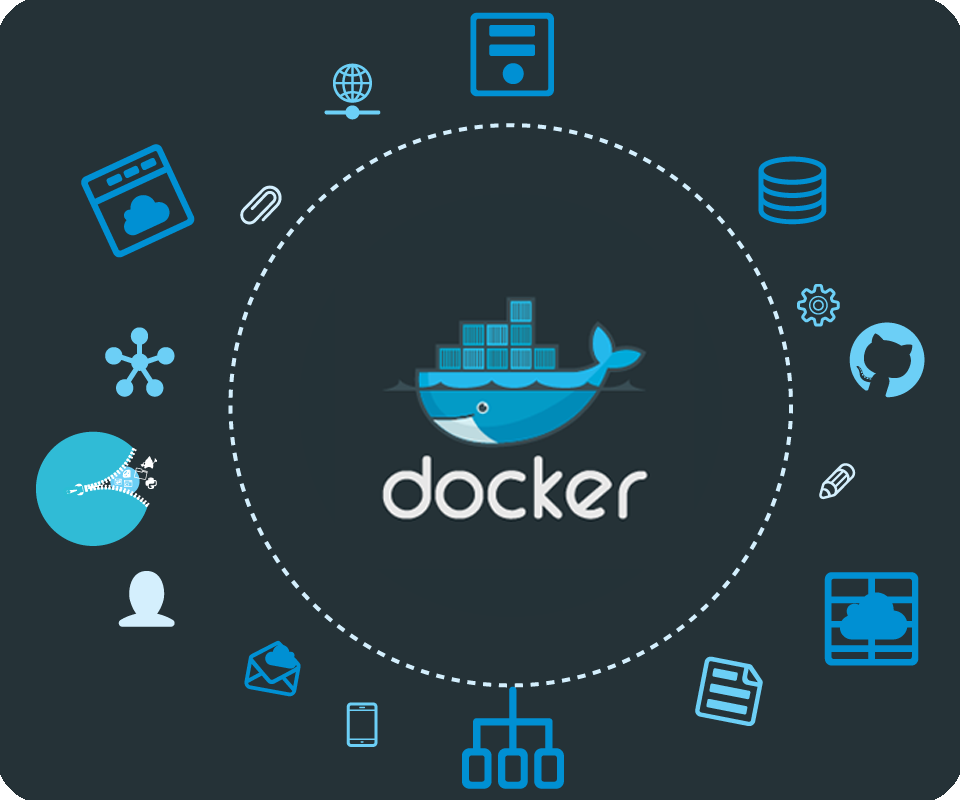
Изначально виртуализация была призвана избавить от подобных проблем, но в ней есть существенные недостатки:

* медленная загрузка;
* возможная плата за предоставление дополнительного пространства;
* не все виртуальные машины (VM) поддерживают совместимое использование;
* поддерживающие VM часто требуют сложной настройки;
* образ может быть слишком большим, так как «дополнительная ОС» добавляет гигабайт пространства в проект поверх операционной системы, а в большинстве случаев на сервер ставится несколько VM, которые занимают еще больше места.



Докер же просто разделяет ядро ОС на все контейнеры (Docker container), работающие как отдельные процессы. Это не единственная подобная платформа, но, бесспорно, одна из самых популярных и востребованных.

Какие очевидные плюсы?

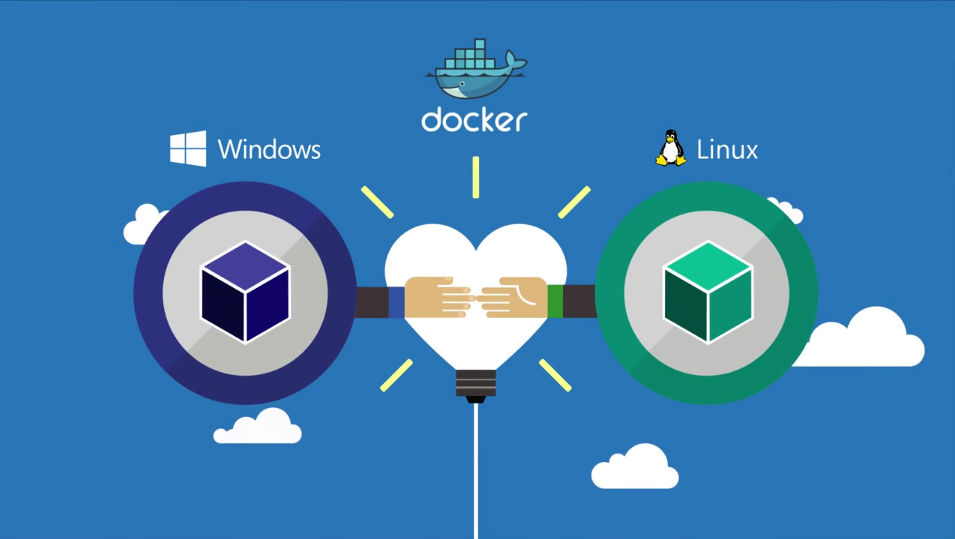


К его преимуществам относятся:

1. Ускоренный процесс разработки. Нет необходимости устанавливать вспомогательные инструменты вроде PostgreSQL, Redis, Elasticsearch: их можно запускать в контейнерах.
2. Удобная инкапсуляция приложений.
3. Понятный мониторинг.
4. Простое масштабирование.

Поддерживаемые платформы

Докер работает не только на его родной ОС, Linux, но также поддерживается Windows и macOS. Единственное отличие от взаимодействия с Linux в том, что на macOS и Windows платформа инкапсулируется в крошечную виртуальную машину. На данный момент Докер для macOS и Windows достиг значительного уровня удобства в использовании.



Кроме того, существует множество дополнительных приложений, таких как [Kitematic](https://kitematic.com/" \t "_blank) или Docker[Machine](https://docs.docker.com/machine/overview/" \t "_blank), которые помогают устанавливать и использовать Докер на платформах, отличных от Linux.

Установка

[Здесь](https://docs.docker.com/get-started/) можно посмотреть подробную инструкцию по установке. Если вы работаете с Докером на ОС Linux, вам нужно выполнить несколько несложных действий и повторно войти в систему:

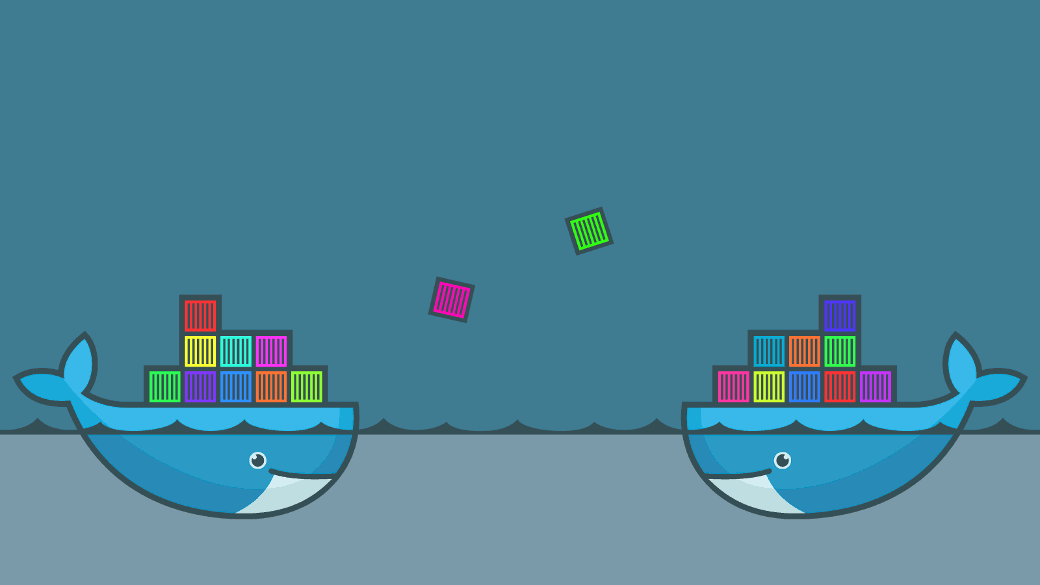
sudo usermod -aG docker $(whoami)

Терминология

1. Контейнер – это исполняемый экземпляр, который инкапсулирует требуемое программное обеспечение. Он состоит из образов. Его можно легко удалить и снова создать за короткий промежуток времени.

2. Образ – базовый элемент каждого контейнера. В зависимости от образа, может потребоваться некоторое время для его создания.

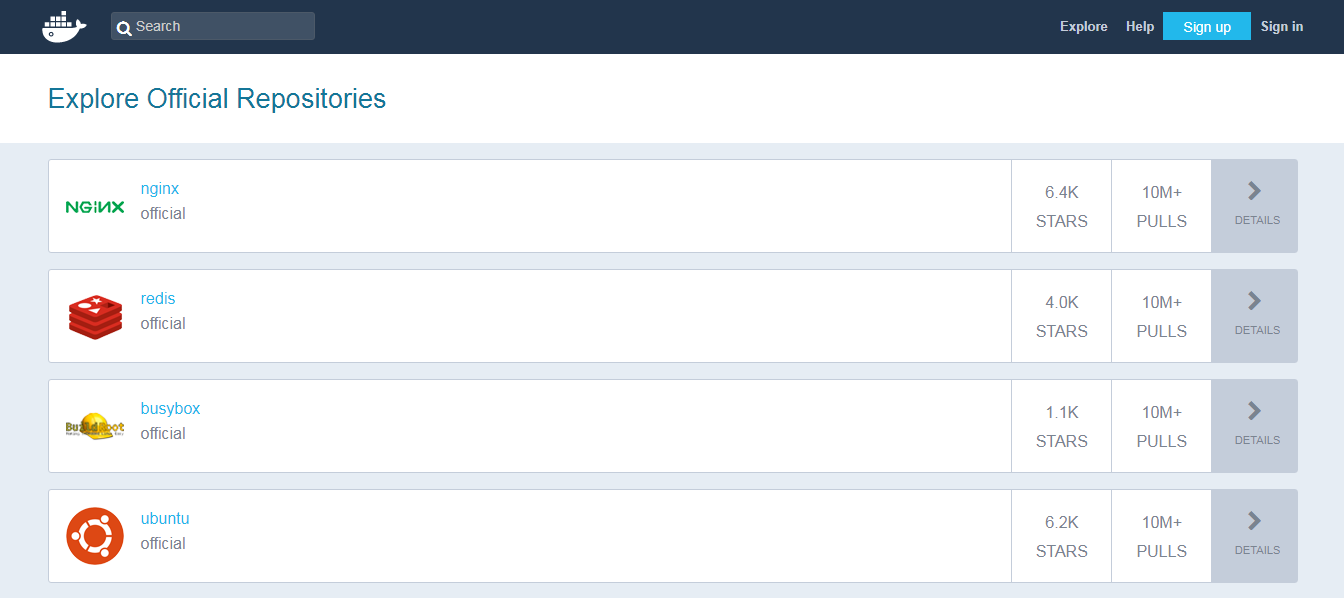
3. Порт – это порт TCP/UDP в своем первоначальном значении. Чтобы все было просто, предположим, что порты могут быть открыты во внешнем мире или подключены к контейнерам (доступны только из этих контейнеров и невидимы для внешнего мира).



4. Том – описывается как общая папка. Тома инициализируются при создании контейнера и предназначены для сохранения данных, независимо от жизненного цикла контейнера.

5. Реестр – это сервер, на котором хранятся образы. Сравним его с GitHub: вы можете вытащить образ из реестра, чтобы развернуть его локально, и так же локально можете вносить в реестр созданные образы.

6. Docker[Hub](https://hub.docker.com/explore/" \t "_blank) – публичный репозиторий с интерфейсом, предоставляемый Докер Inc. Он хранит множество образов. Ресурс является источником «официальных» образов, сделанных командой Докер или созданных в сотрудничестве с разработчиком ПО. Для официальных образов перечислены их потенциальные уязвимости. Эта информация открыта для любого зарегистрированного пользователя. Доступны как бесплатные, так и платные аккаунты.



Пример 1: Hello World

Пришло время запустить наш первый контейнер:

docker run ubuntu /bin/echo 'Hello world'

Консольный вывод:

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 – это команда запуска контейнера.
* ubuntu – образ, который вы запускаете (например, образ операционной системы Ubuntu). Когда вы его указываете, Докер сначала анализирует элемент в разрезе хоста.
* /bin/echo 'Hello world' – команда, которая будет запускаться внутри нового контейнера. Данный контейнер просто выводит «Hello world» и останавливает выполнение.

Теперь попробуем создать интерактивную оболочку внутри контейнера:

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

* -t присваивает псевдо-tty или терминал внутри нового контейнера.
* -i позволяет создавать интерактивное соединение, захватывая стандартный вход (STDIN) контейнера.
* --rm требуется для автоматического удаления контейнера при выходе из процесса. По умолчанию контейнеры не удаляются.

Если вы хотите, чтобы контейнер работал после окончания сеанса, вам необходимо его «демонизировать»:

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

* --name daemon назначает имя новому контейнеру. Если вы его не укажете, имя сгенерируется и назначится автоматически.
* -d запускает контейнер в фоновом режиме («демонизирует» его).

Давайте посмотрим, какие контейнеры у нас есть на данный момент:

docker ps -a

Консольный вывод:

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 перечисления контейнеров.
* -a показывает все контейнеры (без -a ps покажет только запущенные контейнеры).

ps показывает нам, что у нас есть два контейнера:

* gifted\_nobel (имя для этого контейнера генерировалось автоматически) – первый контейнер, который мы создали с набранным «Hello world».
* daemon – третий контейнер, который мы создали и «демонизировали».

Примечание: второй контейнер (с интерактивной оболочкой) отсутствует, потому что мы устанавливаем параметр -rm, в результате чего этот контейнер автоматически удаляется после выполнения.

Давайте проверим журналы и посмотрим, что делает контейнер-демон прямо сейчас:

docker logs -f daemon

Консольный вывод:

...

hello world

hello world

hello world

* docker logs получают журналы контейнера.
* -f следит за выходом журнала.

Теперь давайте остановим контейнер-демон:

docker stop daemon

Проверяем его остановку:

docker ps -a

Консольный вывод:

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 остановлен. Давайте запустим его снова:

docker start daemon

Убедимся, что он запущен:

docker ps -a

Консольный вывод:

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 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 удаления контейнера.
* -f (для rm) должен остановить контейнер, если он работает (принудительное удаление).
* -q (для ps) – это вывод только идентификаторов контейнера.

Пример 2: Nginx

Начиная с этого примера, вам понадобятся дополнительные файлы, которые вы можете найти в [репозитории GitHub](https://github.com/alexryabtsev/docker-workshop). Как вариант, загрузите образцы файлов по [ссылке](https://github.com/alexryabtsev/docker-workshop/archive/master.zip).

Пришло время создать и запустить более важный контейнер, такой как Nginx.

Измените каталог на examples/nginx:

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

Консольный вывод:

Unable to find image 'nginx:latest' locally

latest: Pulling from library/nginx

693502eb7dfb: Pull complete

6decb850d2bc: Pull complete

c3e19f087ed6: Pull complete

Digest: sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe335

Status: Downloaded newer image for nginx:latest

436a602273b0ca687c61cc843ab28163c720a1810b09005a36ea06f005b0c971

* -p – отображение портов HOST PORT: CONTAINER PORT.
* -v отвечает за HOST DIRECTORY:CONTAINER DIRECTORY.

Теперь проверьте [этот URL-адрес](http://127.0.0.1/) в своем веб-браузере.

Еще мы можем попробовать изменить /example/nginx/index.html (который добавляется в каталог /usr/share/nginx/html внутри контейнера) и обновить страницу.

Получим информацию о контейнере test-nginx:

docker inspect test-nginx

Эта команда отображает системную информацию об установке Докер. Она включает версию ядра, количество контейнеров и образов, открытые порты и т. д.

Пример 3: запись Dockerfile

Чтобы создать образ, сперва вам нужно создать Dockerfile: это текстовый файл с инструкциями и аргументами. Краткое описание инструкций, которые мы собираемся использовать в примере:

* FROM – задать базовый образ
* RUN – выполнить команду в контейнере
* ENV – задать переменную среды
* WORKDIR – установить рабочий каталог
* VOLUME – создать точку монтирования для тома
* CMD – установить исполняемый файл для контейнера

Более подробная информация [здесь](https://docs.docker.com/engine/reference/builder/).

Давайте создадим образ, который получит содержимое сайта и сохранит его в текстовом файле. Нам нужно передать URL-адрес через переменную SITE\_URL. Результирующий файл будет помещен в каталог, установленный как том:

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, пришло время создать образ.

Создание образа

Перейдите к examples/curl и выполните следующую команду:

docker build . -t test-curl

Консольный вывод:

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 build создает новый образ локально.
* -t устанавливает в образе метку имени.

Теперь у нас есть новый образ, и мы можем его увидеть в списке существующих:

docker images

Консольный вывод:

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 создавать и запускать контейнер из образа. Давайте попробуем сделать это с параметрами по умолчанию:

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

Чтобы просмотреть результаты, сохраненные в файле:

cat ./vol/results

Попробуем с facebook.com:

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

Чтобы просмотреть результаты, сохраненные в файле:

cat ./vol/results

Рекомендации по созданию образов

* Избегайте установки ненужных пакетов – они будут потреблять дополнительное дисковое пространство.
* Используйте кэш.
* Будьте осторожны с объемами. Вы должны помнить, какие данные в томах. Поскольку тома являются постоянными и не «умирают» с контейнерами – следующий контейнер будет использовать данные из тома, которые были созданы предыдущим контейнером.
* Используйте переменные среды (в RUN, EXPOSE, VOLUME). Это сделает ваш Dockerfile более гибким.

Соединение между контейнерами

Docker[compose](https://docs.docker.com/compose/overview/" \t "_blank) - это единственный правильный способ подключения контейнеров друг к другу.

Пример 4: Python + Redis

В этом примере мы подключим контейнеры Python и Redis.

version: '2'

services:

app:

build:

context: ./app

depends\_on:

- redis

environment:

- REDIS\_HOST=redis

ports:

- "5000:5000"

redis:

image: redis:3.2-alpine

volumes:

- redis\_data:/data

volumes:

redis\_data:

Перейдем к examples/compose и выполним команду:

docker-compose --project-name app-test -f docker-compose.yml up

[Консольный вывод](https://gist.github.com/anonymous/359cf5326a16b46f8375becede3290e8)

Текущий пример увеличит счетчик просмотров в Redis. Откройте [ссылку](http://127.0.0.1:5000/) и убедитесь в этом.

Использование docker-compose – это тема для целого учебника. Чтобы начать работу, вы можете поиграться с некоторыми образами из Docker Hub, а если хотите создать свои собственные – следуйте рекомендациям, перечисленным выше. Единственное, что можно добавить с точки зрения использования docker-compose – всегда давайте явные имена вашим томам. Это простое правило избавит вас от проблемы в будущем.

version: '2'

services:

...

redis:

image: redis:3.2-alpine

volumes:

- redis\_data:/data

volumes:

redis\_data:

В этом случае redis\_data будет именем внутри файла docker-compose.yml.

Смотрим выполнение тома:

docker volume ls

Консольный вывод:

DRIVER VOLUME NAME

local apptest\_redis\_data

Без явного имени тома будет UUID. И вот пример:

DRIVER VOLUME NAME

local ec1a5ac0a2106963c2129151b27cb032ea5bb7c4bd6fe94d9dd22d3e72b2a41b

local f3a664ce353ba24dd43d8f104871594de6024ed847054422bbdd362c5033fc4c

local f81a397776458e62022610f38a1bfe50dd388628e2badc3d3a2553bb08a5467f

local f84228acbf9c5c06da7be2197db37f2e3da34b7e8277942b10900f77f78c9e64

local f9958475a011982b4dc8d8d8209899474ea4ec2c27f68d1a430c94bcc1eb0227

local ff14e0e20d70aa57e62db0b813db08577703ff1405b2a90ec88f48eb4cdc7c19

local polls\_pg\_data

local polls\_public\_files

local polls\_redis\_data

local projectdev\_pg\_data

local projectdev\_redis\_data

В заключение

Докер стал одним из важнейших инструментов современного разработчика. Да, он имеет некоторые ограничения и требования в зависимости от архитектуры вашей системы, но немного усидчивости – и мир контейнеров обязательно будет приручен!