Что такое Докер?

Определение [Докера](https://www.docker.com/) в Википедии звучит так:

программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы; позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, а также предоставляет среду по управлению контейнерами.

Ого! Как много информации. Простыми словами, Докер это инструмент, который позволяет разработчикам, системными администраторам и другим специалистам деплоить их приложения в песочнице (которые называются *контейнерами*), для запуска на целевой операционной системе, например, Linux. Ключевое преимущество Докера в том, что он позволяет пользователям **упаковать приложение со всеми его зависимостями в стандартизированный модуль** для разработки. В отличие от виртуальных машин, контейнеры не создают такой дополнительной нагрузки, поэтому с ними можно использовать систему и ресурсы более эффективно.

Docker можно использовать для следующего:

* Изоляции зависимостей приложения
* Создания образа приложения и его тиражирования
* Создания готового для запуска приложения, которое легко распространять
* Возможности легкого и быстрого масштабирования отдельных экземпляров приложения
* Тестирования приложения и освобождения затраченных ресурсов после завершения тестирования

### Идея, стоящая за Docker, представляет собой создание переносимых легких контейнеров, применяемых для программ, которые могут быть запущены на любом компьютере с установленным Docker, причем, независимо от основной ОС, что напоминает грузовые контейнеры, используемые на судах. Довольно амбициозно и успешно.

### Почему я должен использовать их?

Взлет Докера был по-настоящему эпичным. Не смотря на то, что контейнеры сами по себе — не новая технология, до Докера они не были так распространены и популярны. Докер изменил ситуацию, предоставив стандартный API, который сильно упростил создание и использование контейнеров, и позволил сообществу вместе работать над библиотеками по работе с контейнерами

### Главные компоненты Docker

Docker состоит из двух главных компонент:

* Docker: платформа виртуализации с открытым кодом;
* Docker Hub: наша платформа-как-сервис для распространения и управления docker контейнерами.

*Примечание! Docker распространяется по Apache 2.0 лицензии.*

### Архитектура Docker

Docker использует архитектуру клиент-сервер. Docker клиент общается с демоном Docker, который берет на себя тяжесть создания, запуска, распределения ваших контейнеров. Оба, клиент и сервер могут работать на одной системе, вы можете подключить клиент к удаленному демону docker. Клиент и сервер общаются через сокет или через RESTful API.

#### **Docker-демон**

Как показано на диаграмме, демон за пускается на хост-машине. Пользователь не взаимодействует с сервером на прямую, а использует для этого клиент.

#### **Docker-клиент**

Docker-клиент, программа docker — главный интерфейс к Docker. Она получает команды от пользователя и взаимодействует с docker-демоном.

#### **Внутри docker-а**

Чтобы понимать, из чего состоит docker, вам нужно знать о трех компонентах:

* образы (images)
* реестр (registries)
* контейнеры

#### **Образы**

Docker-образ — это read-only шаблон. Например, образ может содержать операционку Ubuntu c Apache и приложением на ней. Образы используются для создания контейнеров. Docker позволяет легко создавать новые образы, обновлять существующие, или вы можете скачать образы созданные другими людьми. Образы — это компонента сборки docker-а.

#### **Реестр**

#### Docker-реестр хранит образы. Есть публичные и приватные реестры, из которых можно скачать либо загрузить образы. Публичный Docker-реестр — это [Docker Hub](http://hub.docker.com/). Там хранится огромная коллекция образов. Как вы знаете, образы могут быть созданы вами или вы можете использовать образы созданные другими. Реестры — это компонента распространения. **Как работает образ?**

Мы уже знаем, что образ — это read-only шаблон, из которого создается контейнер. Каждый образ состоит из набора уровней. Docker использует [union file system](http://en.wikipedia.org/wiki/UnionFS) для сочетания этих уровней в один образ. Union file system позволяет файлам и директориями из разных файловых систем (разным ветвям) прозрачно накладываться, создавая когерентную файловую систему.  
  
Одна из причин, по которой docker легковесен — это использование таких уровней. Когда вы изменяете образ, например, обновляете приложение, создается новый уровень. Так, без замены всего образа или его пересборки, как вам возможно придётся сделать с виртуальной машиной, только уровень добавляется или обновляется. И вам не нужно раздавать весь новый образ, раздается только обновление, что позволяет распространять образы проще и быстрее.  
  
В основе каждого образа находится базовый образ. Например, ubuntu, базовый образ Ubuntu, или fedora, базовый образ дистрибутива Fedora. Так же вы можете использовать образы как базу для создания новых образов. Например, если у вас есть образ apache, вы можете использовать его как базовый образ для ваших веб-приложений.  
  
*Примечание! Docker обычно берет образы из реестра Docker Hub.*

Docker образы могут создаться из этих базовых образов, шаги описания для создания этих образов мы называем инструкциями. Каждая инструкция создает новый образ или уровень. Инструкциями будут следующие действия:

* запуск команды
* добавление файла или директории
* создание переменной окружения
* указания что запускать когда запускается контейнер этого образа

Эти инструкции хранятся в файле Dockerfile. Docker считывает это Dockerfile, когда вы собираете образ, выполняет эти инструкции, и возвращает конечный образ.

#### **Как работает docker реестр?**

Реестр — это хранилище docker образов. После создания образа вы можете опубликовать его на публичном реестре Docker Hub или на вашем личном реестре.  
  
С помощью docker клиента вы можете искать уже опубликованные образы и скачивать их на вашу машину с docker для создания контейнеров.  
  
Docker Hub предоставляет публичные и приватные хранилища образов. Поиск и скачивание образов из публичных хранилищ доступно для всех. Содержимое приватных хранилищ не попадает в результат поиска. И только вы и ваши пользователи могут получать эти образы и создавать из них контейнеры.

#### **Контейнеры**

Контейнеры похожи на директории. В контейнерах содержится все, что нужно для работы приложения. Каждый контейнер создается из образа. Контейнеры могут быть созданы, запущены, остановлены, перенесены или удалены. Каждый контейнер изолирован и является безопасной платформой для приложения. Контейнеры — это компонента работы.

**Контейнеры** – это способ виртуализации операционной системы, позволяющий запускать приложение и его зависимости в виде процессов с изолированными ресурсами. Контейнеры позволяют легко упаковывать код приложений, их настройки и зависимости в простые в использовании компоновочные блоки, обеспечивающие единообразную среду, эффективность и производительность труда разработчиков, а также контроль версий.   
 У контейнеров есть одна особенность, которая существенно отличает их от виртуальных машин; они спроектированы так, что запускают только один процесс, они также не полностью эмулируют среду (если это то, что вам требуется, то воспользуйтесь LXC).

Каждый **контейнер** включает все необходимое для работы приложения: библиотеки, системные инструменты, код и среду исполнения. Благодаря Docker пользователи могут быстро развертывать и масштабировать свои приложения в любой среде и сохранять уверенность в том, что код будет работать.

**Docker** — это инструмент, предоставляющий удобный интерфейс для работы с LXC.

**LXC (Linux Containers)** — механизм виртуализации на уровне операционной системы,   
позволяющий исполнять   
множество изолированных   
Linux-систем (контейнеров)   
в одной системе.

**Ядро Linux** может изолировать ресурсы (процессор, память, ввод/вывод, сеть и так далее) при помощи cgroups, не прибегая для этого к использованию виртуальных машин. Посредством cgroups изолируются так же деревья процессов, сеть, пользователи и файловые системы.

LXC комбинирует cgroups и пространства имён (namespace).

На данный момент LXC использует следующие возможности ядра:

* Kernel namespaces (ipc, uts, mount, pid, network and user)
* Apparmor and SELinux profiles
* Seccomp policies
* Chroots (using pivot\_root)
* Kernel capabilities
* CGroups (control groups)
* **cgroups** (англ. *control group*) — механизм ядраLinux, который ограничивает и изолирует вычислительные ресурсы (процессорные, сетевые, ресурсы памяти, ресурсы ввода-вывода) для групп процессов

#### **Основные термины**

**Image (образ)** – собранная подсистема, необходимая для работы процесса, сохраненная в образе.  
**Container (контейнер)** – процесс, инициализированный на базе образа. То есть контейнер существует только когда запущен. Это как экземпляр класса, а образ это типа класс. Ну я думаю идея понятна.  
**Host (хост)** – среда, в которой запускается докер. Проще говоря – ваша локальная машина.  
**Volume** – это дисковое пространство между хостом и контейнером. Проще – это папка на вашей локальной машине примонтированная внутрь контейнера. Меняете тут меняется там, и наоборот, миракл.  
**Dockerfile** – файл с набором инструкций для создания образа будущего контейнера  
**Service (сервис)**– по сути это запущенный образ (один или несколько контейнеров), дополнительно сконфигурированный такими опциями как открытие портов, маппинг папок (volume) и прочее. Обычно это делается при помощи docker-compose.yml файла.  
**Docker-compose** (докер-композ, чаще композер, но не путать с php composer) – тулза, облегчающая сборку и запуск системы состоящей из нескольких контейнеров, связанных между собой.  
**Build (билд, билдить)** – процесс создания образа из набора инструкций в докерфайле, или нескольких докерфайлов, если билд делается с помощью композера  
В данной статье позже (завтра) я опишу процесс сборки связки nginx+mysql+php7-fpm с примерами и описаниями dockerfile и docker-compose файлов.

# **Как связать Docker-контейнеры, не заставляя приложение читать переменные окружения**

* [Виртуализация](https://habr.com/ru/hub/virtualization/)

Docker, если кто умудрился об этом ещё не слышать — фреймворк с открытым исходным кодом для управления контейнерной виртуализацией. Он быстрый, удобный, продуманный и модный. По сути он меняет правила игры в благородном деле управления конфигурацией серверов, сборки приложений, выполнения серверного кода, управления зависимостями и много ещё где.  
  
Архитектура, которую поощряет Docker — это изолированные контейнеры, каждый из которых выполняет одну команду. Эти контейнеры должны знать только как друг друга найти — другими словами, о контейнере нужно знать его fqdn и порт, или ip и порт, то есть, не более, чем о любой внешней службе.  
  
Рекомендованный способ сообщить такие координаты внутрь процесса, выполняемого в Docker — переменные окружения. Типичный пример этого подхода, не применительно к докеру — DATABASE\_URL, принятый во фреймворке Rails или NODE\_ENV принятый в фрейворке Nodejs.  
  
И вот переменные окружения позволяют приложению внутри контейнера удобно и непринуждённо найти базу данных. Но для этого, человек, который пишет приложение, должен об этом знать. И хотя [конфигурация приложения с помощью переменных окружения — это хорошо и правильно](http://12factor.net/config), иногда приложения написаны плохо, а запускать их как-то надо.

Технология Docker позволяет связать два контейнера и даёт механизм ссылок (Docker links).

1. Определение имя контейнеру при запуске: docker run -d --name db training/postgres.

Теперь можно ссылаться на этот контейнер по имени db.

2. Запускаем второй контейнер, связывая его с первым:

docker run -d -P --name web --link db:db training/webapp python app.py.

 --link name:alias. name — имя контейнера,

alias — имя, под которым этот контейнер будет   
известен запускаемому.

**В результате:**

во-первых, в контейнере web появится набор переменных окружения, указывающих на контейнер db,

во-вторых в /etc/hosts контейнера web появится имя db указывающий на ip, на котором мы запустили контейнер с базой данных. Набор переменных окружения, которые будут доступны в контейнере web вот такой: