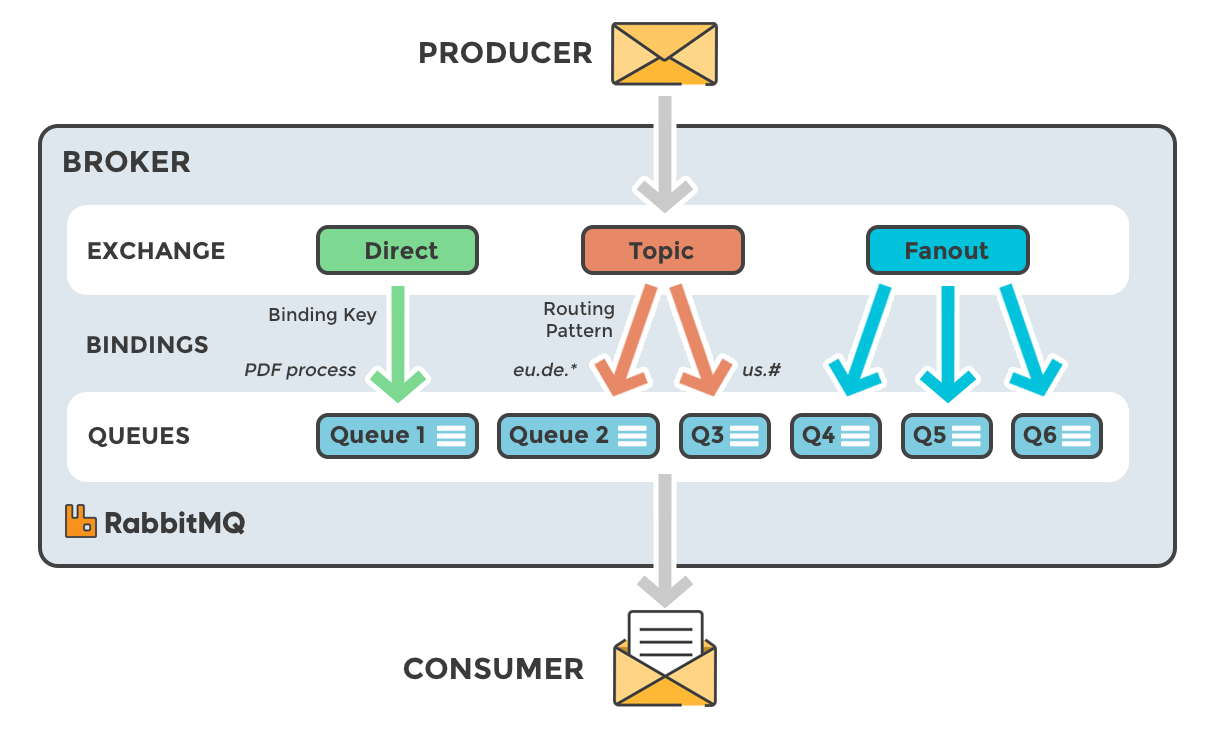
**Кратко про RabbitMQ**

[RabbitMQ](https://www.rabbitmq.com/) – это брокер сообщений с открытым исходным кодом. Он маршрутизирует собщения по всем базовым принципам протокола [AMQP](https://ru.wikipedia.org/wiki/AMQP) описанным в [спецификации](http://www.amqp.org/resources/download). Отправитель передает сообщение брокеру а тот доставляет его получателю. RabbitMQ реализует и дополняет протокол AMQP.

Основная идея модели обмена сообщениями в RabbitMQ заключается в том, что producer (издатель) не отправляет сообщения непосредственно в очередь. На самом деле и довольно часто издатель даже не знает, будет ли сообщение вообще доставлено в какую-либо очередь.

Вместо этого издатель может отправлять сообщения только на обмен. С одной стороны, обмен получает сообщения от издателей, а с другой — отправляет их в очереди. Обмен должен точно знать, что делать с полученным сообщением. Должно ли оно быть добавлено в определенную очередь? Должно ли оно быть добавлено в несколько очередей? Или сообщение нужно игнорировать.



Кратко работу RabbitMQ можно описать следующим образом:

1. Издатель отправляет сообщение определенному обменнику
2. Обменник, получив сообщение, маршрутизирует его в одну или несколько очередей в соответствии с правилами привязки между ним и очередью
3. Очередь хранит ссылку на это сообщение. Само сообщение хранится в оперативной памяти или на диске
4. Как только потребитель готов получить сообщение из очереди, сервер создает копию сообщения по ссылке и отправляет
5. Потребитель получает сообщение и отправляет брокеру подтверждение
6. Брокер, получив подтверждение, удаляет копию сообщения из очереди. Затем удаляет из оперативной памяти и с диска

**Понимая Docker**

### Что такое докер?

Докер — это открытая платформа для разработки, доставки и эксплуатации приложений. Docker разработан для более быстрого выкладывания ваших приложений. С помощью docker вы можете отделить ваше приложение от вашей инфраструктуры и обращаться с инфраструктурой как управляемым приложением. Docker помогает выкладывать ваш код быстрее, быстрее тестировать, быстрее выкладывать приложения и уменьшить время между написанием кода и запуска кода. Docker делает это с помощью легковесной платформы контейнерной виртуализации, используя процессы и утилиты, которые помогают управлять и выкладывать ваши приложения.  
  
В своем ядре docker позволяет запускать практически любое приложение, безопасно изолированное в контейнере. Безопасная изоляция позволяет вам запускать на одном хосте много контейнеров одновременно. Легковесная природа контейнера, который запускается без дополнительной нагрузки гипервизора, позволяет вам добиваться больше от вашего железа.  
  
Платформа и средства контейнерной виртуализации могут быть полезны в следующих случаях:

* упаковывание вашего приложения (и так же используемых компонент) в docker контейнеры;
* раздача и доставка этих контейнеров вашим командам для разработки и тестирования;
* выкладывания этих контейнеров на ваши продакшены, как в дата центры так и в облака.

### Для чего я могу использовать docker?

#### **Быстрое выкладывание ваших приложений**

Docker прекрасно подходит для организации цикла разработки. Docker позволяет разработчикам использовать локальные контейнеры с приложениями и сервисами. Что в последствии позволяет интегрироваться с процессом постоянной интеграции и выкладывания (continuous integration and deployment workflow).  
  
Например, ваши разработчики пишут код локально и делятся своим стеком разработки (набором docker образов) с коллегами. Когда они готовы, отравляют код и контейнеры на тестовую площадку и запускают любые необходимые тесты. С тестовой площадки они могут оправить код и образы на продакшен.

#### **Более простое выкладывание и разворачивание**

Основанная на контейнерах docker платформа позволят легко портировать вашу полезную нагрузку. Docker контейнеры могут работать на вашей локальной машине, как реальной так и на виртуальной машине в дата центре, так и в облаке.  
  
Портируемость и легковесная природа docker позволяет легко динамически управлять вашей нагрузкой. Вы можете использовать docker, чтобы развернуть или погасить ваше приложение или сервисы. Скорость docker позволяет делать это почти в режиме реального времени.

#### **Высокие нагрузки и больше полезных нагрузок**

Docker легковесен и быстр. Он предоставляет устойчивую, рентабельную альтернативу виртуальным машинам на основе гипервизора. Он особенно полезен в условиях высоких нагрузок, например, при создания собственного облака или платформа-как-сервис (platform-as-service). Но он так же полезен для маленьких и средних приложений, когда вам хочется получать больше из имеющихся ресурсов.

### 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.

### 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.

#### **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.

#### **Так как же работает Docker?**

Пока мы знаем, что:

* можем создавать образы, в которых находятся наши приложения;
* можем создавать контейнеры из образов, для запуска приложений;
* можем распространять образы через Docker Hub или другой реестр образов.

Давайте посмотрим, как эти компоненты сочетаются.

#### **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.

#### **Как работает контейнер?**

Контейнер состоит из операционной системы, пользовательских файлов и метаданных. Как мы знаем, каждый контейнер создается из образа. Этот образ говорит docker-у, что находится в контейнере, какой процесс запустить, когда запускается контейнер и другие конфигурационные данные. Docker образ доступен только для чтения. Когда docker запускает контейнер, он создает уровень для чтения/записи сверху образа (используя union file system, как было указано раньше), в котором может быть запущено приложение.

#### **Что происходит, когда запускается контейнер?**

Или с помощью программы docker, или с помощью RESTful API, docker клиент говорит docker демону запустить контейнер.  
  
$ sudo docker run -i -t ubuntu /bin/bash  
  
Давайте разберемся с этой командой. Клиент запускается с помощью команды docker, с опцией run, которая говорит, что будет запущен новый контейнер. Минимальными требованиями для запуска контейнера являются следующие атрибуты:

* какой образ использовать для создания контейнера. В нашем случае ubuntu
* команду которую вы хотите запустить когда контейнер будет запущен. В нашем случае /bin/bash

Что же происходит под капотом, когда мы запускаем эту команду?  
  
Docker, по порядку, делает следующее:

* **скачивает образ ubuntu:** docker проверяет наличие образа ubuntu на локальной машине, и если его нет — то скачивает его с [Docker Hub](https://hub.docker.com/). Если же образ есть, то использует его для создания контейнера;
* **создает контейнер:** когда образ получен, docker использует его для создания контейнера;
* **инициализирует файловую систему и монтирует read-only уровень:** контейнер создан в файловой системе и read-only уровень добавлен образ;
* **инициализирует сеть/мост:** создает сетевой интерфейс, который позволяет docker-у общаться хост машиной;
* **Установка IP адреса:** находит и задает адрес;
* **Запускает указанный процесс:** запускает ваше приложение;
* **Обрабатывает и выдает вывод вашего приложения:** подключается и логирует стандартный вход, вывод и поток ошибок вашего приложения, что бы вы могли отслеживать как работает ваше приложение.

Теперь у вас есть рабочий контейнер. Вы можете управлять своим контейнером, взаимодействовать с вашим приложением. Когда решите остановить приложение, удалите контейнер.

### Используемые технологии

Докер написан на Go и использует некоторые возможности ядра Linux, чтобы реализовать приведенный выше функционал.

#### **Пространство имен(namespaces)**

Docker использует технологию namespaces для организации изолированных рабочих пространств, которые мы называем контейнерами. Когда мы запускаем контейнер, docker создает набор пространств имен для данного контейнера.  
  
Это создает изолированный уровень, каждый аспект контейнера запущен в своем простанстве имен, и не имеет доступ к внешней системе.  
  
Список некоторых пространств имен, которые использует docker:

* **pid:** для изоляции процесса;
* **net:** для управления сетевыми интерфейсами;
* **ipc:** для управления IPC ресурсами. (ICP: InterProccess Communication);
* **mnt:** для управления точками монтирования;
* **utc:** для изолирования ядра и контроля генерации версий(UTC: Unix timesharing system).

#### **Control groups (контрольные группы)**

Docker также использует технологию cgroups или контрольные группы. Ключ к работе приложения в изоляции, предоставление приложению только тех ресурсов, которые вы хотите предоставить. Это гарантирует, что контейнеры будут хорошими соседями. Контрольные группы позволяют разделять доступные ресурсы железа и если необходимо, устанавливать пределы и ограничения. Например, ограничить возможное количество памяти контейнеру.

#### **Union File System**

Union File Sysem или UnionFS — это файловая система, которая работает создавая уровни, делая ее очень легковесной и быстрой. Docker использует UnionFS для создания блоков, из которых строится контейнер. Docker может использовать несколько вариантов UnionFS включая: AUFS, btrfs, vfs и DeviceMapper.

#### **Форматы контейнеров**

Docker сочетает эти компоненты в обертку, которую мы называем форматом контейнера. Формат, используемый по умолчанию, называется libcontainer. Так же docker поддерживает традиционный формат контейнеров в Linux c помощью [LXC](http://linuxcontainers.org/). В будущем Docker возможно будет поддерживать другие форматы контейнеров. Например, интегрируясь с BSD Jails или Solaris Zones.