



AG10 18 апреля 2018 в 08:05

Docker. Начало

Разработка веб-сайтов, .NET, Разработка под Linux, Учебный процесс в IT, Разработка под Windows

Примерно такие же эмоции я и мои коллеги испытывали, когда начинали работать с Docker. это происходило от недостатка понимания основных механизмов, поэтому его поведение как страсти поутихли и вспышки ненависти происходят все реже и все слабее. Более того, посты достоинства и он начинает нам нравиться... Чтобы снизить степень первичного отторжения использования, нужно обязательно заглянуть на кухню Docker'a и хорошенько там осмотреть

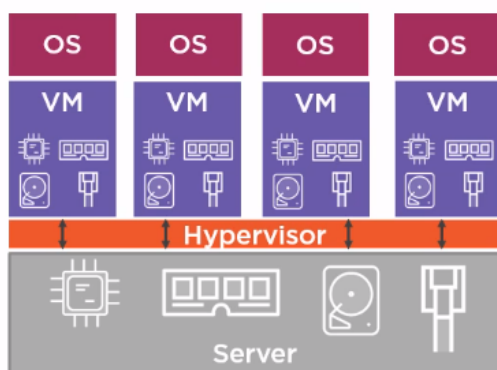
Начнем с того, для чего же нам нужен Docker:

1. изолированный запуск приложений в контейнерах
2. упрощение разработки, тестирования и деплоя приложений
3. отсутствие необходимости конфигурировать среду для запуска — она поставляется в комплекте
4. упрощает масштабируемость приложений и управление их работой с помощью систем

Предыстория

Для изоляции процессов, запущенных на одном хосте, запуска приложений, предназначенны ^{Реклама} виртуальные машины. Виртуальные машины делят между собой физические ресурсы хоста:

- процессор,
- память,
- дисковое пространство,
- сетевые интерфейсы.



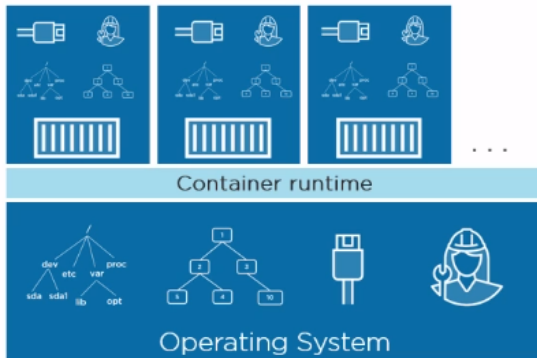
На каждой VM устанавливаем нужную ОС и запускаем приложения. Недостатком такого подхода является то, что значительная часть ресурсов хоста расходуется не на полезную нагрузку(работа приложений), а на работу нескольких ОС.

Контейнеры

Альтернативным подходом к изоляции приложений являются контейнеры. Само понятие контейнеров не ново и давно известно в Linux. Идея состоит в том, чтобы в рамках одной ОС выделить изолированную область и запускать в ней приложение. В этом случае говорим о виртуализации на уровне ОС. В отличие от VM контейнеры изолированно используют свой кусочек ОС:

- файловая система

- дерево процессов
- сетевые интерфейсы
- и др.



Т.о. приложение, запущенное в контейнере думает, что оно одно во всей ОС. Изоляция достигается за счет использования таких Linux-механизмов, как **namespaces** и **control groups**. Если говорить просто, то namespaces обеспечивают изоляцию в рамках ОС, а control groups устанавливают лимиты на потребление контейнером ресурсов хоста, чтобы сбалансировать распределение ресурсов между запущенными контейнерами.

Т.о. контейнеры сами по себе не являются чем-то новым, просто проект Docker, во-первых, скрыл сложные механизмы namespaces, control groups, а во-вторых, он окружен экосистемой, обеспечивающей удобное использование контейнеров на всех стадиях разработки ПО.

Образы

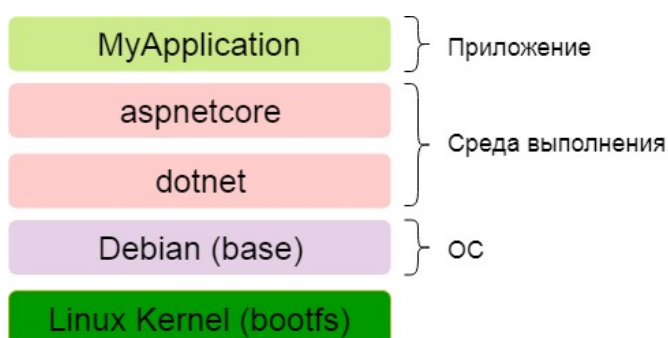
Образ в первом приближении можно рассматривать как набор файлов. В состав образа входит все необходимое для запуска и работы приложения на голой машине с докером: ОС, среда выполнения и приложение, готовое к развертыванию.

Но при таком рассмотрении возникает вопрос: если мы хотим использовать несколько образов на одном хосте, то будет нерационально как с точки зрения загрузки, так и с точки зрения хранения, чтобы каждый образ тащил все необходимое для своей работы, ведь большинство файлов будут повторяться, а различаться — только запускаемое приложение и, возможно, среда выполнения. Избежать дублирования файлов позволяет структура образа.

Образ состоит из слоев, каждый из которых представляет собой неизменяемую файловую систему, а по-простому набор файлов и директорий. Образ в целом представляет собой объединенную файловую систему (Union File System), которую можно рассматривать как результат слияния файловых систем слоев. Объединенная файловая система умеет обрабатывать конфликты, например, когда в разных слоях присутствуют файлы и директории с одинаковыми именами. Каждый следующий слой добавляет или удаляет какие то файлы из предыдущих слоев. В данном контексте «удаляет» можно рассматривать как «затеняет», т.е. файл в нижележащем слое остается, но его не будет видно в объединенной файловой системе.

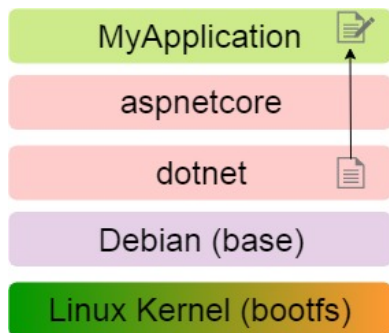
Можно провести аналогию с Git: слои — это как отдельные коммиты, а образ в целом — результат выполнения операции *squash*. Как мы увидим дальше, на этом параллели с Git не заканчиваются. Существуют различные реализации объединенной файловой системы, одна из них — AUFS.

Для примера рассмотрим образ произвольного .NET приложения MyApplication: первым слоем является ядро Linux, далее следуют слои ОС, среды исполнения и уже самого приложения.

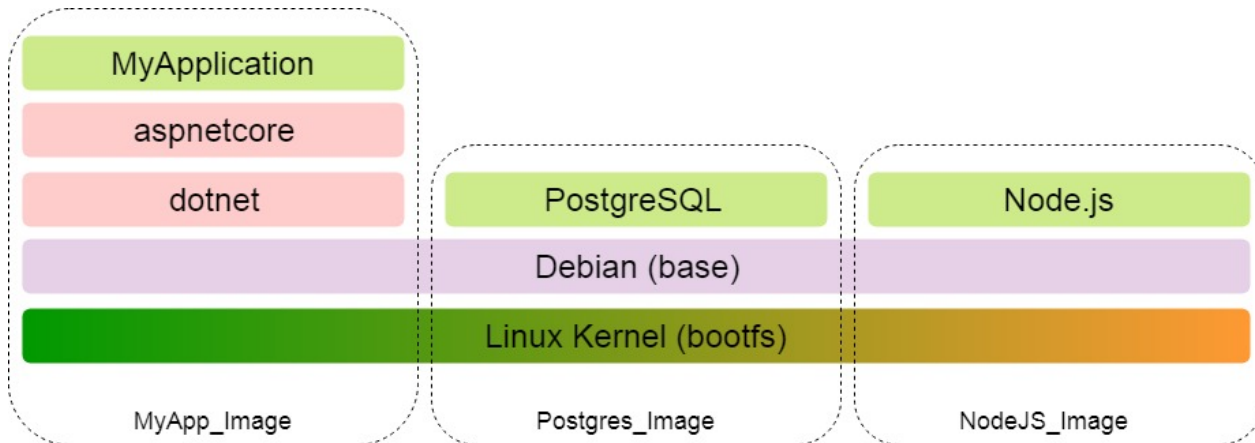


Слои являются read only и, если в слое MyApplication нужно изменить файл, находящийся в слое dotnet, то файл сначала

копируется в нужный слой, а потом в нем изменяется, оставаясь в исходном слое в первозданном виде.



Неизменяемость слоев позволяет использовать их всеми образами на хосте. Допустим MyApplication — это веб-приложение, которое использует БД и взаимодействует также с NodeJS сервером.



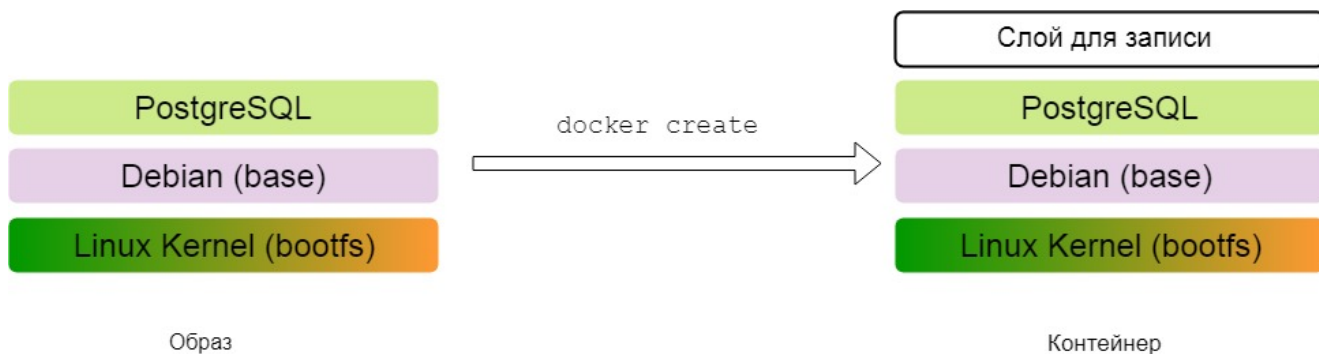
Совместное использование проявляется также и при скачивании образа. Первым загружается манифест, который описывает какие слои входят в образ. Далее скачиваются только те слои из манифеста, которых еще нет локально. Т.о. если мы для MyApplication уже скачали ядро и ОС, то для PostgreSQL и Node.js эти слои уже загружаться не будут.

Подытожим:

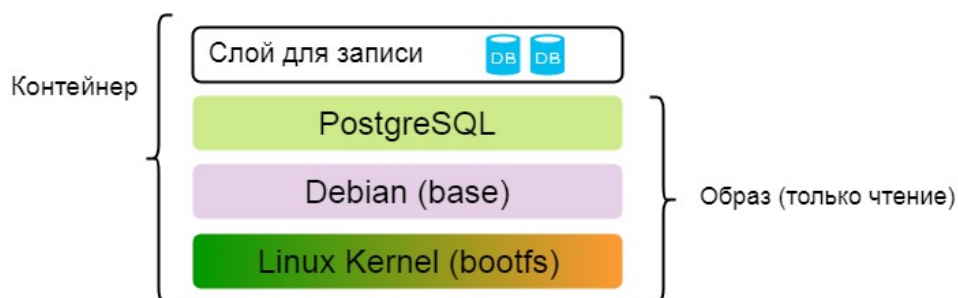
- Образ — это набор файлов, необходимых для работы приложения на голой машине с установленным Docker.
- Образ состоит из неизменяемых слоев, каждый из которых добавляет/удаляет/изменяет файлы из предыдущего слоя.
- Неизменяемость слоев позволяет их использовать совместно в разных образах.

Docker-контейнеры

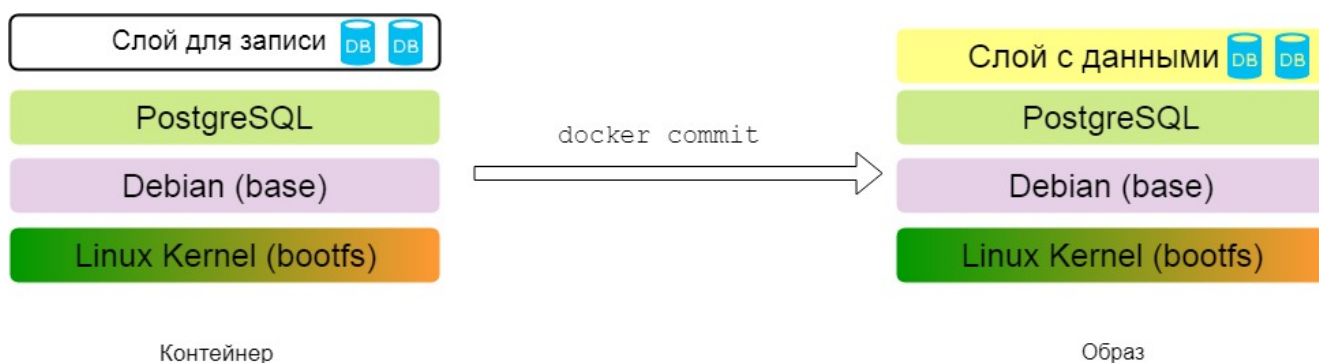
Docker-контейнер строится на основе образа. Суть преобразования образа в контейнер состоит в добавлении верхнего слоя, для которого разрешена запись. Результаты работы приложения (файлы) пишутся именно в этом слое.



Например, мы создали на основе образа с PostgreSQL сервером контейнер и запустили его. Когда мы создаем БД, то соответствующие файлы появляются в верхнем слое контейнера — слое для записи.



Можно провести и обратную операцию: из контейнера сделать образ. Верхний слой контейнера отличается от остальных только лишь разрешением на запись, в остальном это обычный слой — набор файлов и директорий. Делая верхний слой read only, мы преобразуем контейнер в образ.

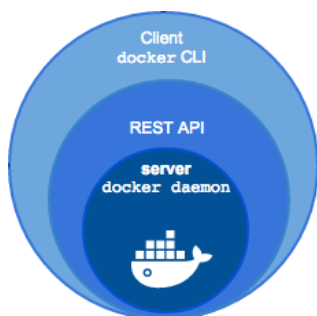


Теперь я могу перенести образ на другую машину и запустить. При этом на сервере PostgreSQL можно будет увидеть БД, созданные на предыдущем этапе. Когда при работе контейнера будут внесены изменения, то файл БД будет скопирован из неизменяемого слоя с данными в слой для записи и там уже изменен.



Docker

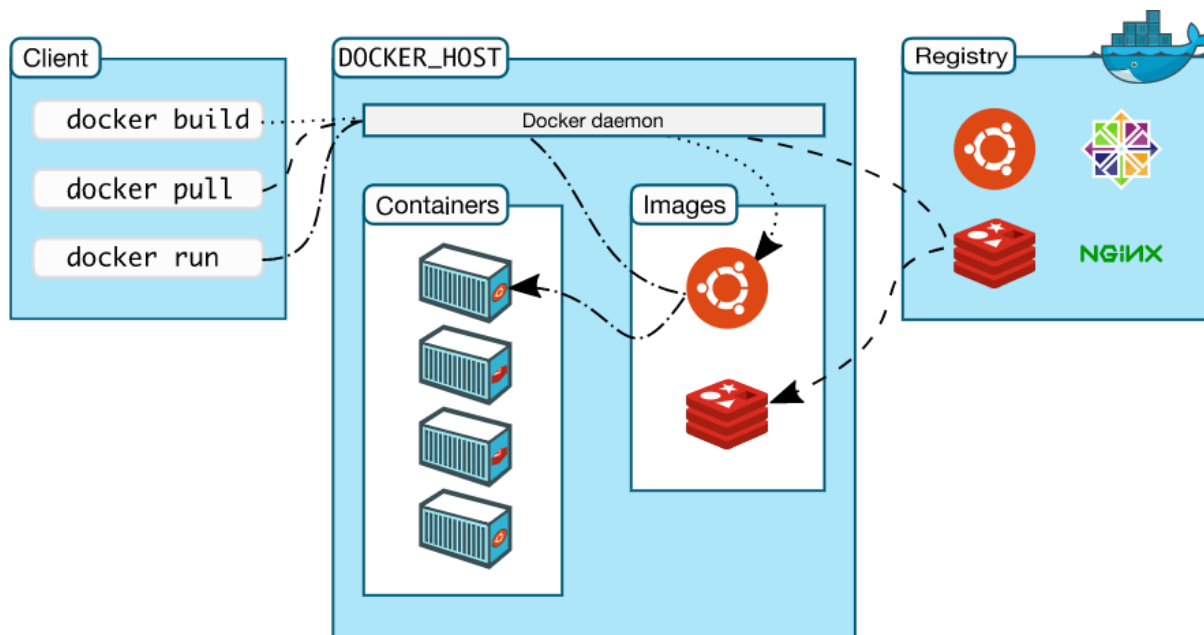
Когда мы устанавливаем докер на локальную машину, то получаем клиент (CLI) и http-сервер, работающий как демон. Сервер предоставляет REST API, а консоль просто преобразует введенные команды в http-запросы.



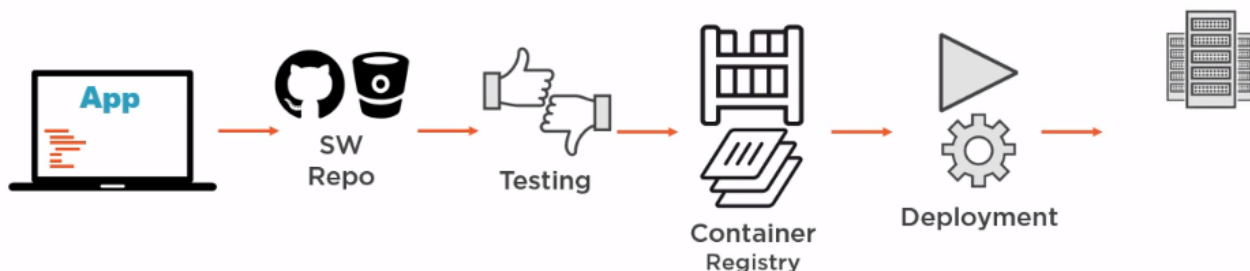
Registry

Registry — это хранилище образов. Самым известным является DockerHub. Он напоминает GitHub, только содержит образы, а не исходный код. На DockerHub также есть репозитории, публичные и приватные, можно скачивать образы (pull), заливать изменения

образов (push). Скачанные однажды образы и собранные на их основе контейнеры хранятся локально, пока не будут удалены вручную.



Существует возможность создания своего хранилища образов, тогда при необходимости Docker будет искать там образы, которых еще нет локально. Надо сказать, что при использовании Docker хранилище образов становится важнейшим звеном в CI/CD: разработчик делает коммит в репозиторий, запускаются тесты. Если тесты прошли успешно, то на основе коммита обновляется существующий или собирается новый образ с последующим деплоем. Причем в registry обновляются не целые образы, а только необходимые слои.



При этом важно не ограничивать восприятие образа как некой коробки в которой приложение просто доставляется до пункта назначения и потом запускается. Приложение может и **собираться внутри образа** (правильнее сказать внутри контейнера, но об этом чуть позже). На схеме выше сервер, занимающийся сборкой образов, может иметь только установленный Docker, а не различные среды, платформы и приложения, необходимые для сборки разных компонентов нашего приложения.

Dockerfile

Dockerfile представляет собой набор инструкций, на основе которых строится новый образ. Каждая инструкция добавляет новый слой к образу. Для примера рассмотрим Dockerfile, на основе которого мог бы быть создан образ рассмотренного ранее .NET-приложения MyApplication:

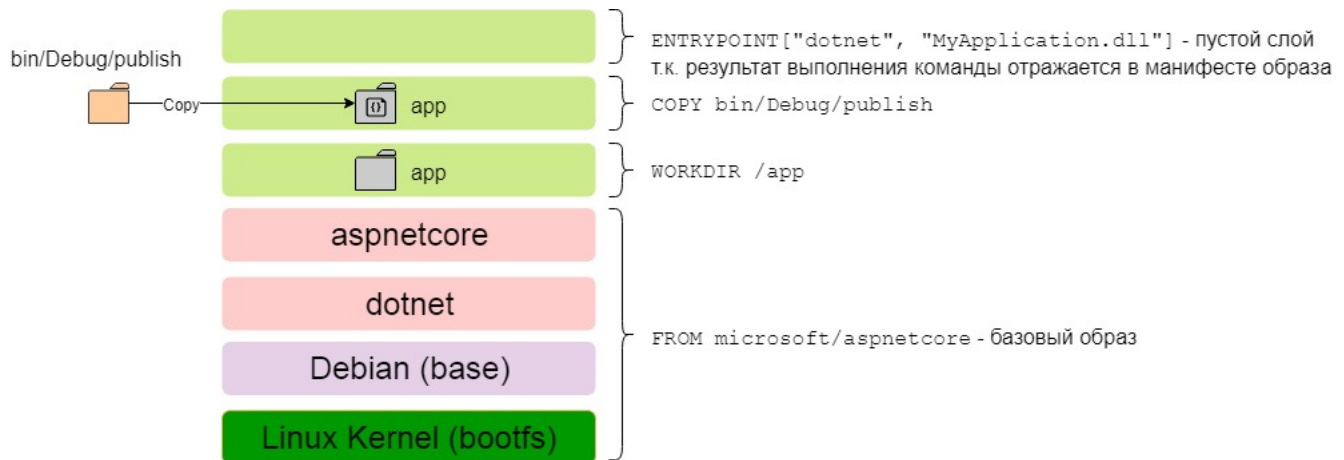
```
FROM microsoft/aspnetcore
WORKDIR /app
COPY bin/Debug/publish .
ENTRYPOINT["dotnet", "MyApplication.dll"]
```

Рассмотрим отдельно каждую инструкцию:

1. определяем базовый образ, на основе которого будем строить свой. В данном случае берем microsoft/aspnetcore — официальный образ от Microsoft, который можно найти на DockerHub
2. задаем рабочую директорию внутри образа

3. копируем предварительно спаблишенное приложение MyApplication в рабочую директорию внутри образа. Сначала пишется исходная директория — путь относительно контекста, указанного в команде `docker build`, а вторым аргументом — целевая директория внутри образа, в данном случае точка обозначает рабочую директорию
4. конфигурируем контейнер как исполняемый: в нашем случае для запуска контейнера будет выполнена команда `dotnet MyApplication.dll`

Если в директории с Dockerfile выполнить команду `docker build`, то мы получим образ на основе `microsoft/aspnetcore`, к которому будет добавлено еще три слоя.



Рассмотрим еще один Dockerfile, который демонстрирует прекрасную возможность Docker, обеспечивающую легковесность образов. Подобный файл генерирует VisualStudio 2017 для проекта с поддержкой контейнеров и он позволяет собирать образ из исходного кода приложения.

```
FROM microsoft/aspnetcore-build:2.0 AS publish
WORKDIR /src
COPY . .
RUN dotnet restore
RUN dotnet publish -o /publish

FROM microsoft/aspnetcore:2.0
WORKDIR /app
COPY --from=publish /publish .
ENTRYPOINT ["dotnet", "MyApplication.dll"]
```

Инструкции в файле разбиты на две секции:

1. Определение образа для сборки приложения: `microsoft/aspnetcore-build`. Данный образ предназначен для сборки, паблиша и запуска .NET приложений и согласно DockerHub с тегом 2.0 имеет размер **699 MB**. Далее происходит копирование исходных файлов приложения внутрь образа и внутри него выполняются команды `dotnet restore` и `dotnet build` с размещением результатов в директории `/publish` внутри образа.
2. Определяется базовый образ, в данном случае это `microsoft/aspnetcore`, который содержит в себе только среду исполнения и согласно DockerHub с тегом 2.0 имеет размер всего **141 MB**. Далее определяется рабочая директория и в нее копируется результат предыдущей стадии (ее имя указывается в аргументе `--from`), определяется команда запуска контейнера и все — образ готов.

В итоге изначально имея исходный код приложения, на основе тяжелого образа с SDK было спаблишено приложение, а потом результат размещен поверх легкого образа, содержащего только среду исполнения!

Напоследок хочу отметить, что намеренно для простоты оперировал понятием образ, рассматривая работу с Dockerfile. На самом деле изменения, вносимые каждой инструкцией происходят конечно же не в образе (ведь в нем только неизменяемые слои), а в контейнере. Механизм такой: из базового образа создается контейнер (добавляется ему слой для записи), выполняется инструкция в данном слое (она может добавлять файлы в слой для записи: `COPY` или нет: `ENTRYPOINT`), вызывается команда `docker commit` и получается образ. Процесс создания контейнера и коммита в образ повторяется для каждой инструкции в файле. В итоге в процессе формирования конечного образа создается столько промежуточных образов и контейнеров, сколько инструкций в файле. Все они автоматически удаляются после окончания сборки конечного образа.

Заключение

Конечно же Docker не панацея и его использование должно быть оправдано и мотивировано не только желанием использовать современную технологию, о которой многие говорят. При этом я уверен, что Docker, примененный грамотно и к месту, может принести много пользы на всех стадиях разработки ПО и облегчить жизнь всем участникам процесса.

Надеюсь смог раскрыть базовые моменты и заинтересовать к дальнейшему изучению вопроса. Конечно же для овладения Docker одной этой статьи недостаточно, но, надеюсь, она станет одним из элементов пазла для осознания общей картины происходящего в мире контейнеров под управлением Docker.


Ссылки

1. Документация Docker
2. Механизм namespaces
3. Механизм control groups
4. Статья о Docker



Теги: Docker, Linux контейнеры



Реклама


Комментарии 30

-  FUNNYDMAN





18 апреля 2018 в 10:37



 


 +1 

Спасибо большое за статью. Обновил некоторые моменты в памяти. Планируется ли написание еще статей? Было бы интересно прочитать про docker-compose и о том, как устроен докер на низком уровне.
-  AG10



18 апреля 2018 в 21:49



   


 0 

Пока не планировал продолжения, но, когда появится чем поделиться, обязательно это сделаю
-  berezuev



18 апреля 2018 в 13:39



 


 0 

Новичкам в docker всегда рекомендую потратить 10 баксов на The Docker Book.
Книга на очень понятном английском, материал разжеван с азов.
К тому же, книгу регулярно обновляют (на данный момент актуальная версия выпущена в феврале 2018, хотя покупалась книга еще в 2016 году)
-  Merkat0r



18 апреля 2018 в 16:14



 


 -1 

Чо, опять на рекламу докер отсыпал? Так поздно — поезд уехал :)
-  ptQa





18 апреля 2018 в 17:14



 

 0 

2k18
@
Docker уже одной ногой в могиле
@
Контейнеры даже в банках и прочих энтерпрайзах
@
На хабре появляются статьи "Что такое контейнеры и как запускать докер"
-  darkAlert

18 апреля 2018 в 19:19

 0 

а почему в docker одной ногой в могиле?

 leqa 18 апреля 2018 в 23:45 # 0

У docker как технологии все* прекрасно, проблема у компании докер, не знают как монетизировать, а без денег коммерческие компании долго не живут. Возможно в итоге докер перейдет в apache foundation или т.п.
Но за технологию переживать не стоит.

 ptQa 19 апреля 2018 в 00:08 # 0

1. Docker cloud не взлетел <https://docs.docker.com/docker-cloud/migration/>
2. Founder свалил <https://blog.docker.com/2018/03/au-revoir/>
3. Kubernetes выиграл войну оркестраторов, запили cri-o (<https://github.com/kubernetes-incubator/cri-o>), а последняя поддерживаемая версия докера в нем 1.12
4. Сейчас стандартизировали registry (<https://github.com/opencontainers/tob/blob/master/proposals/distribution.md>) и скоро видимо запилят ванильную реализацию вместо docker registry
5. Все что осталось у docker как технологии, это билд имаджей (но и ему не долго жить)
6. CFN и OCI толкают unix way, где каждый компонент это отдельный инструмент, и комбайнам типа docker в этом мире не место.
7. Умрет компания, никто не будет саппортить докер как продукт. С rethinkdb такое уже было.

 ptQa 19 апреля 2018 в 00:16 # 0

Все что осталось у docker как технологии, это билд имаджей (но и ему не долго жить)

Да и по поводу билдов, уже есть <https://github.com/projectatomic/buildah> и <https://github.com/openshift/source-to-image>, которые билдят OCI. С фидами, которые мы ждем в Docker уже 5-й год (типа mount volume во время билда). Короче закапывайте.

 stripe 19 апреля 2018 в 03:57 # 0

В таком случае, если чем-то начинать пользоваться в 2018 году — что выбрать? Все еще Docker? Или уже что-то еще?

 dimaborisovsky 19 апреля 2018 в 20:12 # 0

+1 к вопросу, хотелось бы услышать специалистов в данной области

 iproger 19 апреля 2018 в 23:04 # 0

Где почитать о 6 пункте?

 ptQa 20 апреля 2018 в 02:28 # 0

<https://www.opencontainers.org/faq#faq1>

The mission of the Open Container Initiative (OCI) is to promote a **set of common, minimal**, open standards and specifications around container technology.

Вот тут можно посмотреть набор каких стандартов и реализаций они уже сделали (каждая реализация — отдельный тул, а не комбайн!) и что планируют: <https://www.opencontainers.org/about/oci-scope-table>

Что касается CNF, то в kubernetes такая же история, для runtime они поддерживают любой OCI, для network поддерживают любой CNI (<https://github.com/containernetworking/cni>), опять таки никаких монолитных кусков.

 AG10 23 апреля 2018 в 13:12 # 0

Спасибо за ваш комментарий, который побудил разобраться подробнее в этом вопросе. Несмотря на проблемы Docker Inc сам проект, думаю, рановато хоронить.

1. Проект Docker является OpenSource и, вероятно, найдет поддержку даже в случае отсутствия таковой от Docker Inc. Тем более вы сами упомянули о повсеместном использовании проекта.
2. Структура Docker-образов и исполняемая среда контейнеров поддерживает OCI, а значит одинакова для всех проектов,

поддерживающих данный стандарт.

3. У меня нет практического опыта перевода приложения, работающего под управлением Kubernetes, с Docker на другую среду запуска контейнеров, но, например, инструкция по переходу с Docker на CRI-O занимает буквально одну страницу.



achempion 30 августа 2018 в 14:53



↑ 0 ↓

Не совсем так, проект Docker является закрытым проприетарным продуктом который использует некоторые компоненты с открытым исходным кодом, по такой же модели как например «VS Code»: сам билд редактора является закрытым продуктом с дополнительными компонентами к опенсорсному редактору «Code».

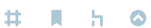
Поэтому вы не найдете никаких исходников программы «Docker for mac» и тд. Опенсорсная часть компонентов из которых состоит Docker вынесена в Mobyproject.

Для закрепления:

- Проект Docker не является OpenSource
- Проект Docker использует некоторые OpenSource компоненты, такие как Mobyproject
- Docker нельзя скачать без регистрации в докер клауд



lega 30 августа 2018 в 17:16



↑ 0 ↓

Для закрепления:

Исходники докера лежат на гитхабе, можете скачать и собрать, бинарники лежат [тут](#) без всякой регистрации.



achempion 30 августа 2018 в 19:14



↑ 0 ↓

А можно, пожалуйста, ссылочку на исходники докера.

Проблема со скачиванием в том, что прямую ссылку вы нигде не найдете. Само скачивание не требует регистрации, а вот чтобы узнать ресурс откуда скачивать — нужно зарегистрироваться.

Предлагаю еще ознакомиться с [этим](#) ишью на гитхабе в официальном репозитории вебсайта.



lega 30 августа 2018 в 22:00



↑ 0 ↓

github.com/docker/docker.git, в интернете так же можете найти пошаговую инструкцию для сборки из этих исходников.

Под linux все ссылки доступны без регистраций, под windows тоже проблем не вижу:

If you haven't already downloaded the installer (Docker for Windows Installer.exe), you can get it from download.docker.com.



achempion 30 августа 2018 в 22:06



↑ 0 ↓

Для мака все же требуется регистрация. Проект моба является опенсорсным, но сам докер нет. Docker и моба это не одно и то же (а ссылка выше на репозиторий ведет именно на исходники моба проекта).

Если по мануалам из моба что-то и соберу, то это не будет докером. Посмотреть и собрать бинарники докера, и сравнить их с теми что выложены на сайте не получится.



achempion 30 августа 2018 в 22:14



↑ 0 ↓

Еще немного добавлю, мне интересно посмотреть на исходники приложения «Docker for Mac», но к сожалению это невозможно сделать, так как это закрытый продукт.



cobbaka 18 апреля 2018 в 19:49



↑ 0 ↓

Не планировали у себя использовать. Но статья раскрыла много плюсов docker. Спасибо за статью



The_Pro 18 апреля 2018 в 19:49



↑ +1 ↓

Исправьте пожалуйста.

"На самом деле изменения, вносимые каждой инструкцией происходят **кончено** же не в образе



AG10 18 апреля 2018 в 19:49



0



спасибо, исправил



арпасу 18 апреля 2018 в 22:32



0



Раз уже сюда заходят специалисты по докеру есть вопрос. Если запускать на ubuntu контейнеры основанные на alpine то есть какой-то профит или наоборот противопоказания?

НЛО прилетело и опубликовало эту надпись здесь



qspor 19 апреля 2018 в 20:12



0



Если docker сливается то почему тогда досихпор OpenShift и K8s досихпор в базе используют его я уже не говорю о тиктонике и ранчере, все по прежнему смотрят на докер, там еще и сварм что-то трепыхается.



NickForHabr 19 апреля 2018 в 20:12



+1



Зачем в этой конструкции точка в конце?

```
COPY --from=publish /app .
```



AG10 19 апреля 2018 в 20:18



0



Я допустил ошибку, инструкция должна выглядеть следующим образом:

```
COPY --from=publish /publish .
```

Она говорит: «Скопируй со стадии *publish* содержимое папки */publish* в текущую рабочую директорию (в данном случае это */app*)»

Спасибо за комментарий!



win32niruh 30 августа 2018 в 14:12



0



Спасибо, хорошая статья. Вот только вопрос: неужели не нашлось нормальное слово

«предварительно **спаблишенное** приложение»?

Если нет, то тогда надо переводить и другие также:

превиоузли спаблишенная аппликация



AG10 5 сентября 2018 в 00:21



0



превиоузли спаблишенная аппликация

Интересный вариант :) ну а если серьезно, то, на мой взгляд, использование сленга это дискуссионный вопрос и применять его, безусловно, нужно дозировано. А крайние варианты, вроде «спаблишенный аппликейшн» и «опубликованное приложение» выглядят зачастую одинаково невнятно.

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.