электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51036. ISSN 2307-0595

Виртуальные контейнеры Docker: назначение и особенности применения

02, февраль 2015

Нанян С. М.[1,*], Ничушкина Т. Н.[2]

УДК: 004.457

[1]Россия, Яндекс
[2]МГТУ им. Н.Э. Баумана
[*]sarkis.nanyan@yandex.ru

Введение

Некоторое время назад, практически все приложения (например вебсайты, СУБД) разрабатывались используя ограниченный набор технологий и развертывались в единый, монолитный сервер – сервер на котором запущено множество различных сервисов и служб. Сегодня разработчики создают и распространяют приложения с использованием лучших доступных сервисов и технологий и должны подготовить приложения для развертывания в самые различные места: на физические серверы, в публичные и приватные облака. Со временем, были выявлены следующие проблемы, полное решение которых при данном подходе не является возможным:

- Разворачивание программного обеспечения на большом количестве различных серверов
- Масштабирование имеющейся инфраструктуры
- Доставка обновлений
- Реализация отказоустойчивости
- Тестирование

Решением данных проблем распределенных приложений является использование виртуализации. Виртуализация — предоставление набора вычислительных ресурсов или их логического объединения, абстрагированное от аппаратной реализации, и обеспечивающее при этом логическую изоляцию вычислительных процессов, выполняемых на одном физическом ресурсе. Кроме того, виртуализация позволяет вместо приобретения новых серверов, запускать виртуальные машины на уже имеющихся серверах, что позволяет:

- резко сократить затраты на охлаждение, поскольку виртуальные серверы не генерируют тепла
- снизить потребление электроэнергии
- упростить обеспечение бесперебойной работы
- добиться высокой технической маневренности

Существует большое количество различного программного обеспечения для виртуализации, однако преобладающее большинство существующих решений не подходит для использования в крупных компаниях с большим количеством серверов и множеством различных сервисов. Недостатком существующих систем виртуализации является использование метода полной виртуализации или метода паравиртуализации. Указанные методы вносят дополнительные накладные расходы в связи с наличием дополнительного уровня гипервизора между ядром исходной системы и виртуальными и виртуальной машиной. Также в каждой виртуальной машине находятся полноценные операционные системы. В случае с использованием метода паравиртуализации, достигается увеличение производительности, однако недостатком является необходимость модификации ядра операционной системы.

Еще одним из подходов виртуализации, является виртуализация на основе операционной системы. Данный подход позволяет вместо создания множественных виртуальных машин внутри физической системы, создавать множественные изолированные окружения прикладных программ, относящихся к одному и тому же ядру. Виртуализацию на основе операционной системы следует рассматривать как функциональную возможность ядра, а не отдельный уровень абстракции программного обеспечения. Указанный метод, реализованный в ядре Linux, значительно сокращает расходы и объем виртуальных машин, по сравнению с классическим методом виртуализации.

1. Устройство Docker

Благодаря возможностям, которое предоставляет ядро *Linux*, и преимуществам метода виртуализации на уровне операционной системы, было разработано программное обеспечение, позволяющее решить вышеописанные проблемы. Этим программным обеспечением является *Docker*, разработчиком которого является одноименная компания.

Docker является свободным программным обеспечением, базирующимся на ядре *Linux* и механизмах ядра *namespaces*, *cgroups* и *LXC*. Механизм *cgroups* позволяет изолировать процессорные, сетевые ресурсы, а также ресурсы памяти и ввода-вывода для групп процессов, что предоставляет следующие возможности:

- ограничение ресурсов;
- применение приоритетов – разным группам процессов можно выделить разное количество процессорного ресурса и пропускной способности подсистемы ввода-вывода;

- учет – подсчет затрат тех либо иных ресурсов группой процессов;
- изоляцию – разделение пространства имен для групп процессов таким образом, что одной группе недоступны процессы, сетевые соединения и файлы другой группы;
- управление – приостановку групп процессов, создание контрольных точек и их перезагрузку.

Использование указанных механизмов позволяет образовывать иерархические группы процессов с заданными ресурсными свойствами и обеспечивать программное управление ими, запускать несколько изолированных экземпляров системы *Linux* на одном узле. Данные экземпляры представляют собой виртуальные окружения, с собственным пространством процессов и сетевым стеком, что и является механизмом *LXC-Linux containers*. Использование таких окружений позволяет устранить проблему монолитных серверов, связанную с непредвиденным поведением процессов, приводящим к выходу из строя целого сервера. Распределение процессов в изолированных окружениях не позволяет при отказе одного приложения, вывести из строя остальные приложения и сервер в целом, обеспечивая при этом необходимую отказоустойчивость.

Решением, которое предлагает *Docker*, является использование виртуальных контейнеров, на основе изолированных окружений процессов, для расширения функционала и возможностей *LXC*, и создания полноценной платформы для решения спектра задач начиная с этапа разработки, заканчивая миграцией и обновлением ПО. На рисунке ниже представлено сравнение метода полной виртуализации и виртуальных контейнеров Docker.

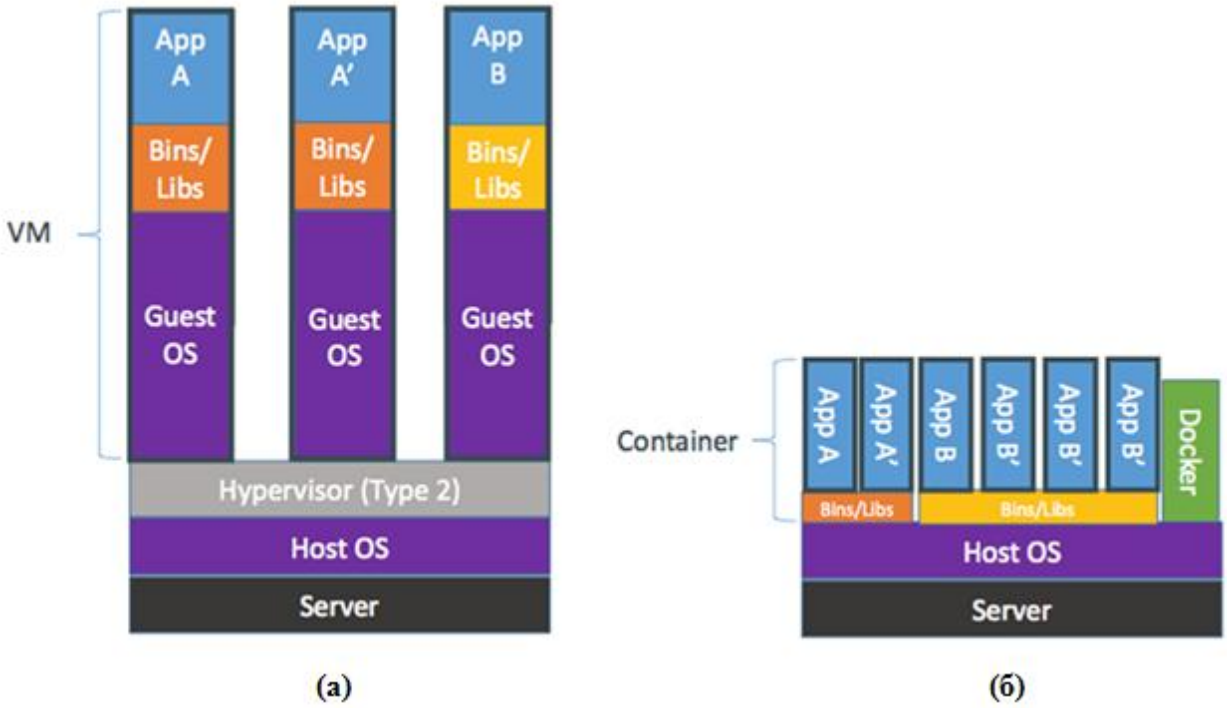

Рис. 1. Сравнение методов виртуализации: а – метод полной виртуализации; б – виртуальные контейнеры Docker

Рисунок 1 (а) демонстрирует недостатки метода полной виртуализации, и преимущества метода виртуализации на уровне операционной системы, на котором построен программный комплекс *Docker*. Метод полной виртуализации обладает избыточностью, каждая виртуальная машина содержит в себе полноценную операционную систему, и все разнообразные операционные системы виртуальных машин должны функционировать через слой гипервизора. Как видно из рисунка 1 (б), вышеописанные недостатки в нем отсутствуют, и виртуальные контейнеры несут в себе непосредственно только необходимую полезную нагрузку, разделяя при этом общие бинарные файлы и библиотеки. Причем, каждый виртуальный контейнер работает напрямую с ядром основной операционной системы. *Docker,* при этом, управляет контейнерами и их взаимодействием.

2. Модель и принцип работы

Docker предоставляет полноценную платформу для разработки, переноса, и запуска программного обеспечения, автоматизирует развертывание приложений в легковесные, переносимые самодостаточные виртуальные контейнеры, которые могут без изменений переноситься между серверами.

ПО *Docker* является клиент-серверным приложением. Клиенты взаимодействуют с сервером, который непосредственно выполняет всю работу. Работать с *Docker* можно с помощью одноименной утилиты командой строки и собственным *Remote API*. Базовым элементом работы с Docker являются образы. Образы представляют собой заготовки под различные версии ядер, на основе которых строятся контейнеры. Еще одним способом создания контейнера являются, так называемые, *Dockerfiles*, которые представляют собой аналог *Makefile*. Описав необходимую конфигурацию контейнера декларативным языком, можно запустить команду по сборке контейнера из этой конфигурации. Кроме того, неотъемлемой частью модели *Docker* является хранилище образов и контейнеров, некий публичный или частный реестр, куда загружаются готовые приложения в виде образа или контейнера. Таким образом, модель *Docker* состоит из следующих базовых элементов:

- Образ
- *Dockerfile*
- Контейнер
- Реестр

Рассмотрим основной принцип работы с технологией *Docker*, представленной на рисунке 2.

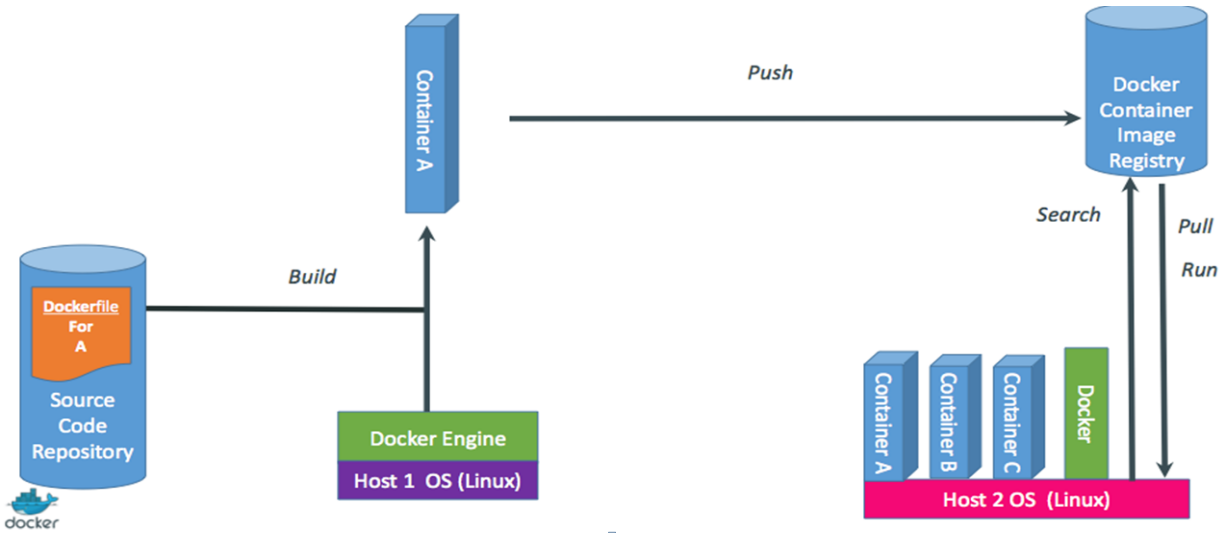

Рис. 2. Принцип работы с Docker

В начале проектировщик, разработав приложение, готовит для него виртуальный контейнер, например, описав *Dockerfile*. После чего выполняет сборку контейнера, на своем компьютере (*Host 1*) и готовый контейнер отправляет в реестр контейнеров и образов. Сервера могут проверить наличие необходимого контейнера в реестре и выполнить загрузку и подключение контейнера при его наличии. Похожим образом происходит доставка обновлений, при внесении изменений в контейнер, принцип доставки представлен на рисунке 3.

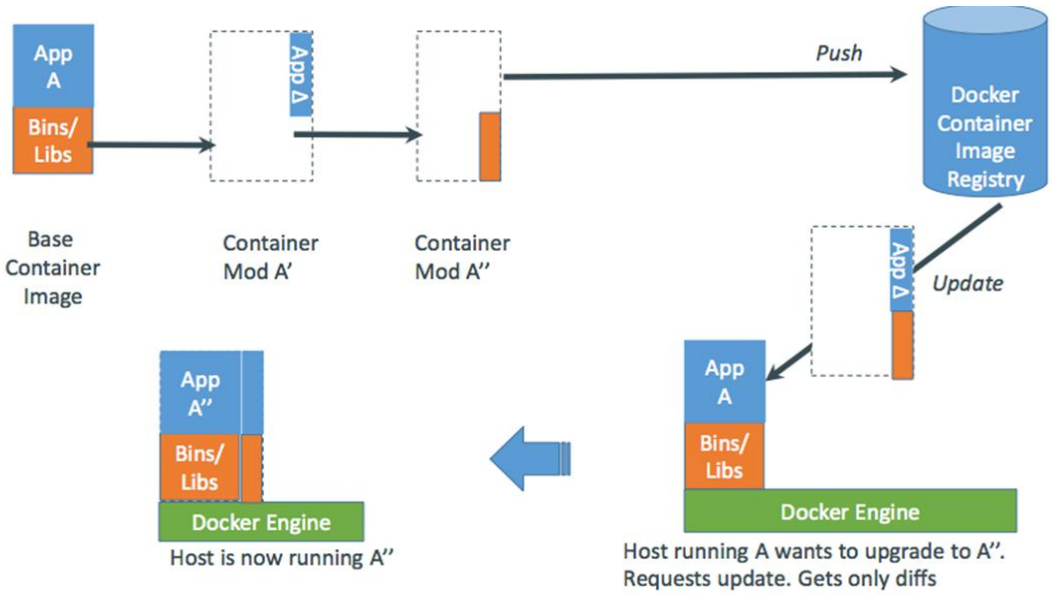

Рис. 3. Принцип доставки обновлений

Рассмотрим принцип доставки обновлений, представленный на рис.3. Пусть, разработчик добавил изменения в исходный код базового образа контейнера и новые необхо-

димые для работы приложения библиотеки и модули. Каждое из данных изменений сохраняется в отдельном образе, однако новые образы содержат только разницу между текущим и базовым образами, тем самым значительно сокращая объем образов и ускоряя доставку. Далее образы загружаются в реестр, откуда сервера могут запросить обновления. После запроса обновления происходит загрузка данного слепка двух образов, и наложение на имеющийся образ.

Заключение

Таким образом, применение описанного средства позволяет разработчикам сосредоточиться на своей цели, написании кода приложений, используя необходимые и привычные для них инструменты, языки программирования. После чего готовое решение, вместе со всеми необходимыми библиотеками, модулями, компиляторами упаковывается в виртуальный контейнер. Контейнеры изолированы, но разделяют одно ядро ОС и бинарные файлы с библиотеками. Результатом является существенно ускоренное развертывание, уменьшение накладных расходов, упрощенная миграция и быстрый перезапуск приложений.

Системным администраторам, в свою очередь, остается позаботиться только о внешнем взаимодействии контейнеров, организовать удаленный доступ, настроить сетевое взаимодействие и подготовить систему мониторинга. Работа с самостоятельными и независимыми контейнерами, реализующими отдельные сервисы, службы, упрощает задачу по автоматизации разворачивания программного обеспечения на распределенные системы и на множество серверов.

ПО *Docker* позволяет значительно ускорить и облегчить процесс разработки, автоматизировать процесс тестирования, автоматизировать процесс непрерывной интеграции и развертывания приложений на распределенные системы, сервера, создавать собственные легковесный облачные окружения и сервисы состоящие из множества независимых самостоятельных компонентов.

Docker является дополнительной программной оболочкой над *LXC*, которая предоставляет более удобный и легко осваиваемый инструментарий для автоматизации работы с большим количеством однообразных виртуальных машин. В этом и заключается основное отличие от *LXC*, *Docker* разрабатывался для упрощения и автоматизации работы с большим количеством виртуальных машин (свыше 100). *LXC* в свою очередь удобен для работы с несколькими десятками виртуальных машин, которые сильно отличаются конфигурацией и требованиями друг от друга. В таких случаях автоматизация не является выгодным решением, и ручная настройка позволяет предусмотреть все особенности каждой виртуальной машины.

В настоящее время единственным ограничением для работы с *Docker*, является необходимость использования идентичных ядер Linux. ПО *Docker* на текущий момент доступно только под операционные системы семейства *Linux*, в связи с вышерассмотренными особенностями реализации. Существует возможность запуска и под операционными сис-

темами семейства *Windows*, однако полноценная поддержка будет доступна в ближайшем будущем, о чем компания *Microsoft* заявила осенью текущего года.

Список литературы

1. Таненбаум Э. Современные операционные системы : пер. с англ. 3-е изд. СПб.: Питер, 2010. – 1120 с.
2. Немет Э., Снайдер Г., Хейн Т., Уэйли Б. Unix и Linux: руководство системного администратора : пер. с англ. 4-е издание / под ред. Д.А. Клюшина. М.: ООО «И.Д. Вильямс», 2012. –1312 с.
3. Turnbull J. The Docker Book. James Turnbull, 2014, p. 338.