Содержание

Предисловие

Контейнеры – это простые и переносимые
между платформами хранилища
для приложения и его зависимостей.

Написанное выше звучит сухо и скучно. Но возможности контейнеров вовсе не ограничиваются усовершенствованием процессов; при правильном использовании контейнеры способны изменить всю картину. Такое обоснование возможности использования контейнеров в архитектурах и рабочих процессах выглядит весьма убедительно, и похоже на то, что все крупные ИТ-компании, которые раньше никогда не слышали о контейнерах и Docker, буквально за год перешли к активным исследованиям и практическому применению этих технологий.

Рост популярности Docker был поразительным. Я не помню, чтобы какая-либо другая среда оказывала столь основательное и глубокое воздействие в области информационных технологий. Эта книга представляет собой попытку помочь читателям понять, почему контейнеры так важны, какую пользу принесет применение контейнеризации и, что самое главное, как перейти к этой технологии.

Для кого предназначена эта книга

В этой книге предпринята попытка дать целостный подход к программной среде Docker с обоснованием ее использования и описанием способов ее практического применения и внедрения в рабочий поток разработки программного обеспечения. Книга полностью охватывает весь жизненный цикл программного продукта – от разработки до промышленного тиражирования и сопровождения.

Я старался не делать каких-либо предположений об уровне знаний читателя об ОС Linux и о процессе разработки программного обеспечения в целом. К предполагаемому кругу читателей прежде всего относятся разработчики программного обеспечения, инженеры по эксплуатации и системные администраторы (особенно те, кто серьезно интересуется развитием методики DevOps). Однако руководители, обладающие достаточными техническими знаниями, и все интересующиеся этой темой также могут кое-что почерпнуть из этой книги.

Почему я написал эту книгу

Мне повезло оказаться среди тех, кто узнал о Docker и стал использовать эту программную среду в самом начале ее стремительного взлета. Когда появилась возможность написать книгу о Docker, я ухватился за нее обеими руками. Если мои труды помогут кому-то из вас понять эту технологию и проделать бóльшую часть

пути к контейнеризации, то я буду считать это более значимым достижением, чем годы разработки программного обеспечения.

Искренне надеюсь, что вы с удовольствием прочтете эту книгу и она поможет вам при переходе к практическому применению Docker в вашей организации.

Структура книги

Книга состоит из трех основных частей:

- часть I начинается с определения контейнеров и краткого описания их свойств, которые представляют интерес для читателей, далее следует глава-руководство по основам Docker. Первая часть завершается большой главой, в которой описаны главные концепции и технология, на которых основана программная среда Docker, включая обзор различных команд среды Docker;
- в части II рассматривается использование Docker в жизненном цикле разработки программного обеспечения. Описан процесс установки среды разработки, затем создание простого веб-приложения, которое используется в дальнейшем как пример на протяжении всей части II. Отдельная глава посвящена разработке, тестированию и объединению с существующей программной средой, также рассматриваются процедура развертывания контейнеров и способы эффективного наблюдения и журналирования в целевой рабочей системе;
- в части III все внимание сосредоточено на мельчайших подробностях, а также на инструментах и методиках, необходимых для безопасного и надежного функционирования многохостовых кластеров, состоящих из Docker-контейнеров. Если вы уже используете Docker и хотите понять, как выполнить масштабирование или решить проблемы безопасности и работы в сетевой среде, то эта часть предназначена для вас.

Условные обозначения и соглашения, принятые в книге

В книге используются следующие типографские соглашения:

Курсив – используется для смыслового выделения важных положений, новых терминов, имен команд и утилит, а также имен и расширений файлов и каталогов.

`Моноширинный шрифт` – используется для листингов программ, а также в обычном тексте для обозначения имен переменных, функций, типов, объектов, баз данных, переменных среды, операторов, ключевых слов и других программных конструкций и элементов исходного кода.

`Моноширинный полужирный шрифт` – используется для обозначения команд или фрагментов текста, которые пользователь должен ввести дословно без изменений.

`Моноширинный курсив` – используется для обозначения в исходном коде или в командах шаблонных меток-заполнителей, которые должны быть заменены соответствующими контексту реальными значениями.

Такая пиктограмма обозначает совет, указание или примечание общего характера.

Эта пиктограмма предупреждает о неочевидных и скрытых «подводных камнях» и «ловушках», которых следует всячески избегать.

В ОРИГИНАЛЕ ЕСТЬ ПИКТОГРАММА "ВОРОНА", НО НЕТ ОПИСАНИЯ К НЕЙ. НАДО БЫ ПРИДУМАТЬ ЕГО.

Примеры исходного кода

Дополнительные материалы (примеры исходного кода, учебные задания и т. п.) можно получить по ссылке https://github.com/using-docker/.

Эта книга написана для того, чтобы помочь вам в работе. Вообще говоря, вы можете использовать код из данной книги в своих программах и в документации. Если вы копируете для собственных нужд фрагмент исходного кода незначительного размера, то нет необходимости обращаться к автору и издателям для получения разрешения на это. Например, при включении в свою программу нескольких небольших фрагментов кода из книги вам не потребуется какое-либо специальное разрешение. Но для продажи или распространения CD-диска с примерами из книг издательства O'Reilly необходимо будет получить официальное разрешение на подобные действия. При ответах на вопросы можно цитировать текст данной книги и приводить примеры кода из нее без дополнительных условий. При включении крупных фрагментов исходного кода из книги в документацию собственного программного продукта также потребуется официальное разрешение.

При цитировании мы будем особенно благодарны за библиографическую ссылку на источник. Обычно ссылка на источник состоит из названия книги, имени автора, наименования издательства и номера по ISBN-классификации. Например: «Using Docker (Использование Docker) by Adrian Mouat (Эдриэн Моуэт) (O'Reilly). Copyright 2016 Adrian Mouat, 978-1-491-91576-9».

Если у вас возникли сомнения в легальности использования примеров исходного кода без получения специального разрешения при условиях, описанных выше, то без колебаний обращайтесь по адресу электронной почты: permissions@oreilly.com.

Онлайн-сервис Safari Books

Онлайн-сервис Safari Books (http://safaribooksonline.com) – это электронная библиотека с весьма быстрым обслуживанием заявок, которая предоставляет в формате книг и в видеоформате высококачественные материалы, созданные ведущими авторами с мировой известностью в технических дисциплинах и в сфере бизнеса.

Специалисты-профессионалы в различных областях техники, разработчики программного обеспечения, веб-дизайнеры, бизнесмены и творческие работники используют онлайн-сервис Safari Books как основной ресурс для научных исследований, решения профессиональных задач, обучения и подготовки к сертификации.

Онлайн-сервис Safari Books предлагает широкий ассортимент разнообразных подборок материалов с индивидуальным формированием цены каждой такой подборки для организаций, государственных учреждений, учебных заведений и частных лиц.

В рамках общей базы данных с единым механизмом полнотекстового поиска подписчики получают доступ к тысячам книг, учебных видеоматериалов и даже к еще не опубликованным рукописям, предоставленным такими известными издательствами, как O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, и многими другими. Более подробную информацию об онлайн-сервисе Safari Books можно получить на сайте https://www.safaribooksonline.com/.

От издательства

Замечания, предложения и вопросы по этой книге отправляйте по адресу:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в США или в Канаде)
707-829-0515 (международный или местный)
707-829-0104 (факс)

Для этой книги создана специальная веб-страница, на которой мы разместили список обнаруженных опечаток и ошибок, исходные коды примеров и другую дополнительную информацию. Сетевой адрес этой страницы: http://bit.ly/using-docker.

Комментарии и технические вопросы по теме данной книги отправляйте по адресу электронной почты: bookquestions@oreilly.com.

Для получения более подробной информации о книгах, учебных курсах, конференциях и новостях издательства O'Reilly посетите наш веб-сайт: http://www.oreilly.com/.

Издательство представлено:

- в соцсети Facebook: http://facebook.com/oreilly;
- в Twitter: http://twitter.com/oreillymedia;
- и в YouTube: http://www.youtube.com/oreillymedia.

Благодарности

Я в высшей степени благодарен за всю помощь, советы и критику, которые получал во время написания этой книги. Если в приведенном ниже списке я пропустил чье-то имя, примите мои извинения; ваш вклад оценен по достоинству вне зависимости от содержания этого списка.

За плодотворное активное сотрудничество благодарю Элли Хьюм (Ally Hume), Тома Сагдена (Tom Sugden), Лукаша Гумински (Lukasz Guminski), Тайлейе Элему (Tilaye Alemu), Себастиена Гусгоэна (Sebastien Goasduen), Максима Белоусова (Maxim Beloussov), Михаэля Белена (Michael Boelen), Ксению Бурлаченко (Ksenia Burlachenko), Карлоса Санчеса (Carlos Sanchez), Дэниэла Брайанта (Daniel Bryant), Кристофера Холмштедта (Christoffer Holmstedt), Майка Рэтбана (Mike Rathbun), Фабрицио Соппелза (Fabrizio Soppelsa), Юн-Чжин Ху (Yung-Jin Hu), Йоуни Миикки (Jouni Miikki) и Дэйла Бьюли (Dale Bewley).

За обсуждение технических вопросов и технологий, описываемых в этой книге, спасибо Эндрю Кеннеди (Andrew Kennedy), Питеру Уайту (Peter White), Алексу Поллитту (Alex Pollitt), Финтэну Райану (Fintan Ryan), Шону Крэмптону (Shaun Crampton), Спайку Кертису (Spike Curtis), Алексису Ричардсону (Alexis Richardson), Илье Дмитриченко (Ilya Dmitrichenko), Кейси Биссон (Casey Bisson), Тийсу Шнитгеру (Tijs Schnitger), Шен Лян (Sheng Liang), Тимо Дерштаппену (Timo Derstappen), Пуйа Аббасси (Puja Abbassi), Александру Ларссону (Alexander Larsson) и Келзи Хайтауэр (Kelsey Hightower). Отдельная благодарность Кевину Годэну (Kevin Gaudin) за разрешение неограниченного использования модуля monsterid.js.

За всю оказанную мне помощь благодарю коллектив издательства O'Reilly, особая благодарность моему редактору Брайану Андерсону (Brian Anderson) и Меган Бланшетт (Meghan Blanchette) за работу на начальном этапе процесса.

Диого Моника (Diogo Mónica) и Марк Коулмэн (Mark Coleman), спасибо вам обоим за то, что откликнулись на мою просьбу о срочной помощи.

Кроме того, следует особо отметить две компании: Container Solutions и CloudSoft. Джейми Добсон (Jamie Dobson) и Container Solutions поощряли ведение моего блога и выступления на соответствующих мероприятиях, а также обеспечивали связь с некоторыми людьми, оказавшими влияние на содержание этой книги. Компания CloudSoft любезно предоставила в мое распоряжение офис на время написания книги и провела семинар Edinburgh Docker – оба этих события были чрезвычайно важными для меня.

За терпеливую поддержку моей одержимости и моих роптаний во время работы над книгой спасибо моим друзьям и моей семье – вам хорошо известно, что вы значите для меня (хотя вряд ли прочтете когда-либо эти строки).

Наконец, спасибо ди-джеям BBC 6 Music, предоставившим саундтрек для этой книги, в том числе Лорен Лаверн (Lauren Laverne), Рэдклифф и Макони (Radcliffe and Maconie), Шону Кэвени (Shaun Keaveny) и Игги Поп (Iggy Pop).

Часть I

ПРЕДПОСЫЛКИ И ОСНОВЫ

В первой части книги вы узнаете, что такое контейнеры и почему они становятся столь широко распространенными. Далее следуют введение в технологию Docker и описание основных концепций, которые необходимо понимать, чтобы наилучшим образом использовать контейнеры.

Глава 1

Что такое контейнеры и для чего они нужны

Контейнеры коренным образом изменяют способ разработки, распространения и функционирования программного обеспечения. Разработчики могут создавать программное обеспечение на локальной системе, точно зная, что оно будет работать одинаково в любой операционной среде – в программном комплексе ИТ-отдела, на ноутбуке пользователя или в облачном кластере. Инженеры по эксплуатации могут сосредоточиться на поддержке работы в сети, на предоставлении ресурсов и на обеспечении бесперебойной работы и тратить меньше времени на конфигурирование окружения и на «борьбу» с системными зависимостями. Масштабы перехода к практическому применению контейнеров стремительно растут во всей промышленности информационных технологий, от небольших стартапов до крупных предприятий. Разработчики и инженеры по эксплуатации должны понимать, что необходимость постоянного использования контейнеров будет возрастать в течение нескольких следующих лет.

Контейнеры (containers) представляют собой средства инкапсуляции приложения вместе с его зависимостями. На первый взгляд контейнеры могут показаться всего лишь упрощенной формой виртуальных машин (virtual machines – VM) – как и виртуальная машина, контейнер содержит изолированный экземпляр операционной системы (ОС), который можно использовать для запуска приложений.

Но контейнеры обладают некоторыми преимуществами, обеспечивающими такие варианты использования, которые трудно или невозможно реализовать в обычных виртуальных машинах:

- контейнеры совместно используют ресурсы основной ОС, что делает их на порядок более эффективными. Контейнеры можно запускать и останавливать за доли секунды. Для приложений, запускаемых в контейнерах, накладные расходы минимальны или вообще отсутствуют, по сравнению с приложениями, запускаемыми непосредственно под управлением основной ОС;
- переносимость контейнеров обеспечивает потенциальную возможность устранения целого класса программных ошибок, вызываемых незначительными изменениями рабочей среды, – лишается обоснования древний довод разработчика: «но это работает на моем компьютере»;

- упрощенная сущность контейнера означает, что разработчики могут одновременно запускать десятки контейнеров, что дает возможность имитации работы промышленной распределенной системы. Инженеры по эксплуатации могут запустить на одном хосте намного больше контейнеров, чем при использовании отдельных виртуальных машин;
- кроме того, контейнеры предоставляют преимущества конечным пользователям и разработчикам без необходимости развертывания приложения в облаке. Пользователи могут загружать и запускать сложные приложения без многочасовой возни с конфигурированием и проблемами при установке и при этом не беспокоиться о каких-либо изменениях в их локальных системах. В свою очередь, разработчики подобных приложений могут избежать проблем, связанных с различиями в конфигурациях пользовательских сред и с доступностью зависимостей для этих приложений.

И что более важно, существуют принципиальные различия в целях использования виртуальных машин и контейнеров – целью применения виртуальной машины является полная эмуляция инородной программной (операционной) среды, тогда как цель применения контейнера – сделать приложения переносимыми и самодостаточными.

Сравнение контейнеров с виртуальными машинами

Несмотря на то что контейнеры и виртуальные машины на первый взгляд кажутся похожими, между ними существуют важные различия, которые проще всего продемонстрировать на графических схемах.

На рис. 1.1 показаны три приложения, работающих в отдельных виртуальных машинах на одном хосте. Здесь требуется гипервизор[1] для создания и запуска виртуальных машин, управляющий доступом к нижележащей ОС и к аппаратуре, а также при необходимости интерпретирующий системные вызовы. Для каждой виртуальной машины необходимы полная копия ОС, запускаемое приложение и все библиотеки поддержки.

В противоположность описанной схеме на рис. 1.2 показано, как те же самые три приложения могут работать в системе с контейнерами. В отличие от виртуальных машин, ядро[2] хоста совместно используется (разделяется) работающими кон-

---

[1] На диаграмме изображен гипервизор типа 2, такой как Virtualbox или VMWare Workstation, который работает поверх основной ОС. Существуют также гипервизоры типа 1, такие как Xen, работающие непосредственно на «голом железе».

[2] Ядро (kernel) является главным компонентом любой ОС и отвечает за предоставление приложениям важнейших системных функций (ресурсов), связанных с оперативной памятью, процессором и доступом к различным устройствам. Полноценная ОС состоит из ядра и разнообразных системных программ, таких как программа инициализации системы, компиляторы и оконные менеджеры.

тейнерами. Это означает, что контейнеры всегда ограничиваются использованием того же ядра, которое функционирует на хосте. Приложения Y и Z пользуются одними и теми же библиотеками и могут совместно работать с этими данными, не создавая избыточных копий. Внутренний механизм контейнера отвечает за пуск и остановку контейнеров так же, как гипервизор в виртуальной машине. Тем не менее процессы внутри контейнеров равнозначны собственным процессам ОС хоста и не влекут за собой дополнительных накладных расходов, связанных с выполнением гипервизора.

Как виртуальные машины, так и контейнеры можно использовать для изоляции приложений друг от друга при работе на одном хосте. Дополнительная степень изоляции в виртуальных машинах обеспечивается гипервизором, и виртуальные машины являются многократно проверенной в реальных условиях технологией. Контейнеры представляют собой сравнительно новую технологию, и многие организации не вполне доверяют методике изоляции функциональных компонентов в контейнерах, пока не увидят воочию весомых доказательств преимуществ контейнеров. По этой причине часто встречаются гибридные системы, в которых контейнеры работают внутри виртуальных машин для совмещения преимуществ обеих технологий.

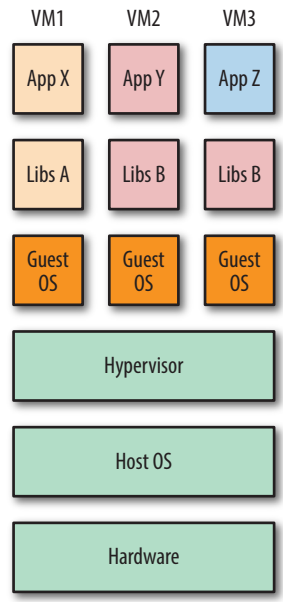

Рис. 1.1. Три виртуальные машины, работающие на одном хосте

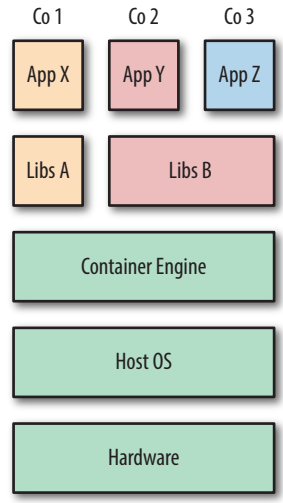

Рис. 1.2. Три контейнера, работающих на одном хосте

Docker и контейнеры

Контейнеры являются старой теоретической концепцией. Еще несколько десятков лет назад в Unix-системах существовала команда chroot, обеспечивающая простейшую форму изоляции части файловой системы. С 1998 года в ОС FreeBSD появилась утилита jail, распространяющая изоляционные возможности chroot на процессы. В 2001 году реализация Solaris Zones обеспечивала относительно полную технологию контейнеризации, но ее применение было ограничено только ОС Solaris. В том же 2001 году компания Parrallels Inc (в дальнейшем SWsoft) выпустила коммерческую версию технологии контейнеров Virtuozzo для ОС Linux, а в 2005 году открыла исходные коды ядра этой технологии под именем OpenVZ[1]. Затем компания Google начала разаработку CGroups для ядра ОС Linux и приступила к перемещению своей инфраструктуры в контейнеры. Проект Linux Containers (LXC) был создан в 2008 году и вскоре объединил CGroups, пространства имен ядра, технологию chroot и некоторые другие технологии, чтобы предоставить полностью завершенное решение по обеспечению контейнеризации. Наконец, в 2013 году Docker стал последним штрихом в общей картине состояния контейнеризации, и эта технология по праву заняла свое место как одно из главных направлений развития ИТ-индустрии.

В основу Docker была заложена существующая технология Linux-контейнеров с разнообразными обертками и расширениями – в основном использующими пе-

[1] Технология OpenVZ никогда не применялась в массовом порядке, возможно, из-за того, что для ее реализации требуется модифицированное ядро.

реносимые образы и удобный для пользователя интерфейс – для создания полностью готового к применению решения, обеспечивающего создание и распространение контейнеров. Платформа Docker состоит из двух отдельных компонентов: Docker Engine, механизма, отвечающего за создание и функционирование контейнеров, и Docker Hub, облачного сервиса для распространения контейнеров.

Механизм Docker Engine предоставляет эффективный и удобный интерфейс для запуска контейнеров. До этого для запуска контейнеров, использующих такую технологию, как, например, LXC, требовались изрядный запас специальных знаний в этой области и большой объем ручной работы. Docker Hub предоставляет огромное количество образов контейнеров с открытым доступом для загрузки, позволяя пользователям быстро начать работу с ними и избежать рутинной работы, ранее уже проделанной другими людьми. Несколько позже были разработаны инструментальные средства для Docker: Swarm – менеджер кластеров, Kitematic – графический пользовательский интерфейс для работы с контейнерами и Machine – утилита командной строки для поддержки работы Docker-хостов.

Ввиду открытости исходных кодов Docker Engine стали возможными создание большого сообщества приверженцев технологии Docker и его постоянный рост, а также вытекающие из этого преимущества массовой помощи в исправлении ошибок и внесении усовершенствований. Быстрый рост популярности этой технологии свидетельствует о том, что Docker действительно становится стандартом де-факто, и это уже привело к разработке независимых стандартов для функциональности времени выполнения и формата контейнеров. В 2015 году наивысшей точкой развития стало создание Open Container Initiative, «управляющей структуры», поддерживаемой Docker, Microsoft, CoreOS и многими другими известными организациями, целью которой является разработка промышленного стандарта. Основой для разработки служат формат и функциональные формы времени выполнения Docker-контейнера.

Темпы роста применения контейнеров в наибольшей степени обеспечили разработчики, которым с самого начала были предоставлены инструментальные средства для эффективного использования контейнеров. Минимальное время от начала разработки до ввода в эксплуатацию Docker-контейнеров чрезвычайно важно для разработчиков, которым необходимы быстрые итеративные циклы разработки с возможностью немедленного просмотра результатов изменений, внесенных в исходный код. Переносимость и изолированность контейнеров способствуют упрощению сотрудничества с другими разработчиками и инженерами по эксплуатации: разработчики могут быть вполне уверены в том, что их код будет работать в любых программных средах, а инженеры по эксплуатации могут сосредоточиться на организации и настройке работы контейнеров на хостах, не беспокоясь о том, какой код выполняется внутри контейнеров.

Новшества, внесенные с применением технологии Docker, существенно изменили способ разработки программного обеспечения. Без Docker контейнеры, вероятнее всего, еще долго оставались бы малозаметным направлением в развитии информационных технологий.

Метафорическая аналогия с доставкой грузов

Философию Docker часто описывают с помощью метафоры «доставки грузов в контейнерах», которая вполне очевидно объясняет происхождение имени Docker. Ниже приводится наиболее типичное изложение этой философии.

При транспортировке грузов используются разнообразные средства, включая трейлеры, автопогрузчики, краны, поезда и корабли. Эти средства должны обладать возможностями работы с широким спектром грузов различных размеров и соблюдать многочисленные специальные требования (например, при транспортировке упаковок кофе, бочек с опасными химикатами, коробок с электронными товарами, парков дорогих автомобилей и стеллажей с замороженными мясными продуктами). На протяжении многих лет это был тяжелый и дорогостоящий процесс, требующий больших трудозатрат многих людей, в том числе и портовых докеров, для погрузки и выгрузки вручную различных предметов в каждом транзитном пункте (рис. 1.3).

Рис. 1.3. Работа докеров в порту Бристоля (Англия) в 1940 году (снимок предоставлен отделом фотографии Министерства информации Великобритании)

Коренной переворот в транспортной промышленности произвело появление универсальных грузовых контейнеров. Эти контейнеры имели стандартные размеры и были специально спроектированы таким образом, чтобы для их перемещения между различными видами транспортных средств требовался минимум ручного труда. Все виды грузового транспорта создавались с учетом характеристик этих

контейнеров – от автопогрузчиков и кранов до грузовиков, поездов и кораблей. Контейнеры-рефрижераторы и изолированные контейнеры предназначены для перевозки грузов с жестко заданным температурным режимом, например некоторых продуктов питания и фармацевтических товаров. Кроме того, преимущества стандартизации распространились и на другие вспомогательные системы, такие как система маркировки грузов и система герметизации и опечатывания контейнеров. Это означает, что ответственность за содержимое контейнеров полностью перекладывается на производителей транспортируемых товаров, а работники транспорта могут полностью сосредоточиться на перевозке и хранении самих контейнеров.

Основная цель программной среды Docker – перенести преимущества стандартизации контейнеров в область информационных технологий. В последние годы программные системы отличаются впечатляющим разнообразием. Прошли времена технологии LAMP[1], реализованной на одном компьютере. Типичная современная система может состоять из фреймворков JavaScript, баз данных NoSQL, очередей сообщений, прикладных программных интерфейсов REST и внутренних компонентов, написанных на различных языках программирования. Такой стек должен частично или полностью работать на разнообразной аппаратуре – от личного компьютера или ноутбука разработчика и небольшого тестового кластера до крупного промышленного узла провайдера облачных сервисов. Все эти среды отличаются разнообразием, используют различные операционные системы с разными версиями библиотек и работают на различной аппаратуре. Короче говоря, проблема точно такая же, как и в транспортной промышленности, – для перемещения кода приложений между разными средами требуется значительный объем ручного труда. Подобно тому, как универсальные контейнеры упрощают перевозку грузов, контейнеры Docker упрощают перемещение (перенос) программных приложений. Разработчики могут полностью сосредоточиться на создании приложения, на проведении цикла тестирования и на вводе приложения в эксплуатацию, не беспокоясь о различиях в программных средах и обеспечении необходимых зависимостей. Инженеры по эксплуатации могут уделить все внимание специфическим вопросам обеспечения работы контейнеров, таким как распределение ресурсов, запуск и остановка контейнеров, перемещение контейнеров между серверами.

Краткая история Docker

В 2008 году Соломон Хайкс (Solomon Hykes) основал компанию dotCloud для реализации облачной технологии «платформа как услуга» (Platform-as-a-Service – PaaS), полностью независимой от какого-либо языка программирования. Решение вопроса независимости от языка являлось наиболее значимой отличительной характеристикой компании dotCloud, поскольку существующие «платформы как услуги» были привязаны к конкретным наборам языков (например, платформа Heroku поддерживала Ruby, платформа Google App Engine поддерживала Java и Python). В 2010 году компания dotCloud приняла участие в разработке программы Y Combinator accelerator, в результате чего установила связи с новыми

[1] LAMP – Linux, Apache, MySQL, PHP – основные компоненты любого веб-приложения.

партнерами и начала привлекать серьезных инвесторов. Самое важное событие произошло в марте 2013 года, когда dotCloud открыла исходные коды Docker, ключевого компонента, ядра программного сервиса dotCloud. В то время как некоторые компании опасались разглашения своих «чудо-секретов», компания dotCloud поняла, что Docker принесет гораздо больше пользы, если станет открытым проектом, управляемым свободным сообществом.

Ранние версии Docker представляли собой немного усовершенствованную обертку механизма LXC в сочетании с файловой системой с каскадно-объединенным монтированием, но при этом ввод в эксплуатацию и скорость разработки были потрясающе быстрыми. В течение шести месяцев проект заработал более 6700 звезд на сайте GitHub и привлек 175 добровольных участников. Это привело к изменению названия компании с dotCloud на Docker Inc. и к смене ориентации бизнес-модели. Спустя 15 месяцев после релиза 0.1, в июне 2014 года была представлена версия Docker 1.0. В этой версии было продемонстрировано значительное улучшение стабильности и надежности, и она была официально объявлена «готовой к промышленной эксплуатации», хотя до этого уже использовалась в полной мере некоторыми компаниями, в том числе Spotify и Baidu. В то же время началась работа по превращению из простого механизма контейнеров Docker в полноценную платформу, сопровождаемая запуском открытого репозитория для контейнеров Docker Hub.

Другие компании быстро оценили потенциальные возможности Docker. В сентябре 2013 года основным партнером становится компания Red Hat, которая начала использовать Docker в качестве движка своего облачного продукта OpenShift. Вскоре после этого Google, Amazon и DigitalOcean обеспечили поддержку Docker в своих облачных сервисах, а некоторые стартапы стали специализироваться на создании Docker-хостов, например компания StackDock. В октябре 2014 года корпорация Microsoft объявила о полной поддержке Docker в будущих версиях Windows Server, что свидетельствовало о существенном изменении позиционирования компании, ранее знаменитой своим массивным, «неповоротливым» корпоративным программным обеспечением.

На конференции DockerConEU в декабре 2014 года был представлен Docker Swarm, менеджер кластеров для Docker и Docker Machine, инструмент командной строки для поддержки работы Docker-хостов. Это стало очевидным подтверждением намерения компании Docker создать полное комплексное решение для обеспечения работы контейнеров, не ограничиваясь при этом одним лишь механизмом Docker.

В декабре того же года проект CoreOS объявил о разработке rkt, собственного механизма поддержки контейнеров, а также о разработке спецификации контейнеров appc. В июне 2015 года на конференции DockerCon в Сан-Франциско Соломон Хайкс (Docker) и Алекс Полви (Alex Polvi) (CoreOS) объявили о создании организации Open Container Initiative (OCI) (позже переименованной в Open Container Project) для разработки общего стандарта формата контейнеров и механизмов запуска.

Также в июне 2015 года проект FreeBSD заявил о поддержке Docker в ОС FreeBSD с использованием файловой системы ZFS и механизма обеспечения совместимости с ОС Linux. В августе 2015 года Docker и Microsoft совместно выпустили «предварительный технический обзор» реализации Docker Engine для Windows Server.

При выпуске версии Docker 1.8 компания Docker представила новую функцию управления надежностью содержимого контейнера, которая проверяет целостность Docker-образа и подлинность авторства стороны, опубликовавшей этот образ. Управление надежностью содержимого является важнейшим компонентом для создания проверенных рабочих процессов на основе образов, загружаемых из реестров Docker.

Дополнительные модули и надстройки

Docker Inc., как и любая другая компания, всегда тонко чувствовала, что своим успехом она обязана всей экосистеме в целом. Компания Docker Inc. сконцентрировала внимание на создании стабильной, готовой к промышленной эксплуатации версии механизма контейнеров, в то время как другие компании, например CoreOS, WeaveWorks и ClusterHQ, работали в смежных областях, таких как оркестрация и сетевая связанность контейнеров. Но сразу стало понятно, что Docker Inc. планирует представить полностью завершенную платформу «из коробки», включающую все возможности поддержки сетевой среды, хранения данных и организации работы в целом. Для поощрения непрерывного развития экосистемы и обеспечения доступа пользователей к решениям из широкого набора вариантов использования компания Docker Inc. заявила, что намерена создать модульную расширяемую программную среду для Docker, в которой все компоненты могут быть заменены на равноценные компоненты от независимых производителей («третьих сторон») или расширены с помощью функциональных компонентов независимых производителей. Компания Docker Inc. охарактеризовала эту философию фразой «батарейки включены в комплект, но их можно полностью заменить», означающей, что должно быть представлено полностью завершенное решение, но отдельные его части могут быть заменены[1].

На момент написания данной книги инфраструктура дополнительных подключаемых модулей уже доступна, хотя и находится на самой ранней стадии своего развития. Существует несколько дополнительных модулей для обеспечения сетевой связанности контейнеров и управления данными.

Кроме того, компания Docker следует так называемому «Манифесту наращивания функциональности инфраструктуры» («Infrastructure Plumbing Manifesto»),

---

[1] Автору никогда не нравилась эта фраза, поскольку все батарейки предоставляют в основном одну и ту же функциональность, но могут быть заменены только батарейками того же типоразмера и с одинаковой величиной напряжения. Автор полагает, что приведенная фраза происходит от философской концепции языка Python «все батарейки включены в комплект» как констатации того факта, что все стандартные библиотеки расширений всегда включены в дистрибутивный комплект Python.

в котором придается особое значение обязательному повторному использованию компонентов и улучшению их существующей инфраструктуры, когда есть возможность для этого, а также предоставление сообществу усовершенствованных повторно используемых компонентов при возникновении потребности в новых инструментах. Следствием этого является выделение исходного кода низкого уровня для поддержки работы контейнеров в проект runC, управляемый OCI, и этот код можно многократно использовать как основу для других контейнерных платформ.

64-битовая версия ОС Linux

Во время написания данной книги единственной стабильной и готовой к промышленному использованию платформой для Docker можно назвать только 64-битовую версию ОС Linux. Это означает, что для работы Docker необходимо установить на компьютер дистрибутив 64-битовой версии Linux, и все создаваемые контейнеры также будут образами 64-битовой версии этой ОС. Если вы работаете с ОС Windows или Mac OS, то можете запустить Docker в виртуальной машине. Поддержка «родных» контейнеровдля других платформ, включая BSD, Solaris и Windows Server, находится на различных стадиях разработки. Поскольку Docker сам по себе не обеспечивает реализацию любого типа виртуализации, контейнеры всегда должны соответствовать ядру хоста – контейнер на Windows Server может работать только на хосте под управлением ОС Windows Server, 64-битовый Linux-контейнер работает только на хосте с установленной 64-битовой версией ОС Linux.

Микросервисы и «монолиты»

Одним из наиболее часто встречающихся вариантов использования контейнеров, в наибольшей степени способствующим их широкому распространению, являются микросервисы (microservices).

Микросервисы представляют собой такой способ разработки и компоновки программных систем, при котором они формируются из небольших независимых компонентов, взаимодействующих друг с другом через сеть. Такая методика полностью противоположна традиционному «монолитному» способу разработки программного обеспечения, где создается одна большая программа, обычно написанная на C++ или на Java.

При необходимости масштабирования «монолитной» программы выбор, как правило, ограничен только вариантом вертикального масштабирования (scale up), и растущие потребности удовлетворяются использованием более мощного компьютера с большим объемом оперативной памяти и более производительным процессором. В противоположность такому подходу микросервисы предназначены для горизонтального масштабирования (scale out), когда рост потребностей удовлетворяется добавлением нескольких серверов с распределением нагрузки между ними. В архитектуре микросервисов возможно регулирование только тех ресурсов, которые требуются для конкретного сервиса, то есть можно сосредо-

точиться лишь на узких проблемных местах в системе. В «монолитной» системе масштабируется либо все, либо ничего, в результате ресурсы используются крайне нерационально.

С точки зрения сложности микросервисы подобны обоюдоострому клинку. Каждый отдельный микросервис должен быть простым для понимания и модификации. Но в системе, сформированной из десятков или даже сотен таких микросервисов, общая сложность возрастает из-за многочисленных взаимодействий между отдельными компонентами.

Простота и высокая скорость работы контейнеров позволяют считать их наиболее подходящими компонентами для реализации архитектуры микросервисов. По сравнению с виртуальными машинами, контейнеры намного меньше по размерам и гораздо быстрее развертываются, что позволяет использовать в архитектурах микросервисов минимум ресурсов и быстро реагировать на требуемые изменения.

Чтобы узнать больше о микросервисах, прочтите книги Building Microservices (англ.) Сэма Ньюмена (Sam Newman) и Microservice Resource Guide (англ.) Мартина Фаулера (Martin Fowler). Русский перевод вышел в издательстве «Питер» в 2016 году.