Содержание

Предисловие

В сентябре 2013 года, просматривая сайт *Hacker News*, я наткнулся на статью в *Wired* о новой технологии под названием «Docker». Когда я ее читал, я все больше волновался, осознавая революционный потенциал этой платформы.

Компания, на которую я работал более десяти лет, изо всех сил пыталась поставлять программное обеспечение достаточно быстро. Подготовка среды была дорогостоящим, трудоемким, ручным и неэлегантным занятием. Непрерывной интеграции почти не существовало, а настройка среды разработки требовала терпения. Поскольку название моей должности включало в себя слова «Менеджер DevOps», я был очень заинтересован в решении этих проблем!

Я привлек пару мотивированных коллег (один из них теперь мой соавтор) через список рассылки компании, и наша команда разработчиков трудилась над тем, чтобы превратить бета-инструмент в бизнес-преимущество, снижая высокую стоимость виртуальных машин и предлагая новые способы мышления относительно сборки и развертывания программного обеспечения. Мы даже собрали и открыли исходный код инструмента автоматизации (ShutIt) для удовлетворения потребностей нашей организации в доставке.

Docker дал нам упакованный и обслуживаемый инструмент, решавший многие проблемы, которые были бы фактически непреодолимыми, если бы мы взяли их решение на себя. Это был открытый исходный код в лучшем виде, который позволил нам принять вызов, используя свое свободное время, преодолевая технический долг и ежедневно обучаясь. Обучаясь не только Docker, но и непрерывной интеграции, доставке, упаковке, автоматизации и тому, как люди реагируют на быстрые и разрушительные технологические изменения.

Для нас Docker – удивительно обширный инструмент. Везде, где вы запускаете программное обеспечение с использованием Linux, Docker может влиять на него. Это затрудняет написание книги на эту тему, так как ландшафт настолько же широк, как и само программное обеспечение. Задача усложняется необычайной скоростью, с которой экосистема Docker производит решения для удовлетворения потребностей, возникающих в результате таких фундаментальных изменений в производстве программного обеспечения. Со временем форма проблем и решений стала нам знакомой, и в книге мы постарались передать данный опыт. Это позволит вам найти решения для ваших конкретных технических и бизнес-ограничений.

Выступая на встречах, мы поражены тем, насколько быстро Docker стал эффективным в организациях, готовых использовать его. Эта книга отражает то, как мы использовали Docker, переходя от настольных компьютеров через конвейер DevOps вплоть до производства. Следовательно, книга признана не всеми, но, будучи инженерами, мы считаем, что ,безупречность иногда уступает место практичности, особенно когда речь идет об экономии денег! Все в этой книге основано на реальных уроках в этой области, и мы надеемся, что вы извлечете пользу из нашего с трудом завоеванного опыта.

Иан Милл

Об этой книге

Docker, пожалуй, самый быстрорастущий программный проект в мире. Его код был открыт в марте 2013 года, к 2018 году он получил почти 50 000 звезд на GitHub и свыше 14 000 ответвлений. Он принял значительное количество запросов на принятие изменений от таких компаний, как Red Hat, IBM, Microsoft, Google, Cisco и VMWare.

Docker достиг этой критической массы, откликнувшись на насущную потребность многих программных организаций: способность создавать программное обеспечение открытым и гибким способом, а затем надежно и последовательно развертывать его в различных контекстах. Вам не нужно изучать новый язык программирования, покупать дорогостоящее аппаратное обеспечение или делать многое в плане установки или настройки, чтобы компактно собирать, поставлять и запускать приложения с помощью Docker.

Второе издание *Docker in Practice* знакомит вас с реальными примерами использования Docker с применением техник, которые мы брали в различных контекстах. Там, где это возможно, мы пытались объяснить эти методы, не требуя знания других технологий, прежде чем приступать к чтению. Мы предполагаем, что читатели знакомы с основными методами и концепциями, такими как способность разрабатывать структурированный код и знание процессов разработки и развертывания программного обеспечения. Кроме того, предполагаются знания основных идей управления исходным кодом и базовые знания основ сети, таких как TCP/IP, HTTP и порты. Менее важные направления мы будем объяснять по мере продвижения.

Начиная с краткого изложения основ Docker в первой части, во второй части мы заостряем внимание на использовании Docker при разработке на одном компьютере. В третьей части мы переходим к использованию Docker с конвейером DevOps, рассказывая о непрерывной интеграции, непрерывной доставке и тестировании.

В четвертой части речь идет о том, как запускать контейнеры Docker масштабируемым образом с помощью оркестровки.

В последней части описывается, как запустить Docker в производстве, особое внимание уделяется вариантам стандартных производственных операций, а также тому, что может пойти не так и как с этим бороться.

Docker – это настолько широкий, гибкий и динамичный инструмент, что не отставать от его стремительного развития – задача не для слабонервных. Мы постарались дать вам понимание критических концепций с помощью реальных приложений и примеров с целью предоставить возможность критически оценить будущие инструменты и технологии в экосистеме Docker с уверенностью. Мы надеялись сделать книгу приятной экскурсией по множеству увиденных нами способов, которыми Docker делает жизнь проще и даже веселее.

Погружение в Docker познакомило нас со многими интересными методиками программного обеспечения, охватывающими весь его жизненный цикл, и мы надеемся, что вы разделите этот опыт.

Дорожная карта

Эта книга состоит из 16 глав, разделенных на 5 частей.

Часть 1 закладывает основы для остальной книги, знакомя вас с Docker и предлагая выполнять некоторые основные команды Docker. Глава 2 посвящена знакомству с архитектурой Docker «клиент-сервер» и способам ее отладки, что может быть полезно для выявления проблем с нетрадиционными настройками Docker.

Часть 2 посвящена знакомству с Docker и максимально эффективному его использованию на вашем собственном компьютере. Аналогия с концепцией, которая, возможно, вам известна, – виртуальные машины – используется в качестве основы для главы 3, чтобы показать более простой способ по-настоящему приступить к использованию Docker. В главах 4, 5 и 6 подробно описывается несколько техник, которые мы ежедневно применяем для создания образов, их запуска и управления самим Docker. В последней главе этой части более подробно рассматривается тема создания образов с помощью методов управления конфигурацией.

В третьей части мы рассмотрим использование Docker в контексте DevOps – от его использования для автоматизации сборок и тестирования программного обеспечения до перемещения вашего собранного программного обеспечения в различные места. Эта часть завершается главой о виртуальной сети Docker, которая представляет Docker Compose и охватывает ряд более сложных тем, связанных с сетевой средой, таких как сетевое моделирование и сетевые плагины Docker.

Часть 4 исследует тему оркестровки контейнеров. Мы отправимся в путешествие от одного контейнера на одном хосте к платформе на основе Docker, работающей в «центре обработки данных в качестве операционной системы». Глава 13 – это расширенное обсуждение областей, которые необходимо учитывать при выборе платформы на основе Docker, и она также служит руководством к тому, что думают архитекторы предприятия при внедрении таких технологий.

Часть 5 охватывает ряд тем для эффективного использования Docker в рабочей среде. В главе 14 рассматривается важная тема безопасности, объясняется, как заблокировать процессы, происходящие внутри контейнера, и как ограничить доступ к внешнему демону Docker. В последних двух главах подробно рассматривается ключевая практическая информация для запуска Docker в производстве. Глава 15 демонстрирует, как применять классические знания системного администратора в контексте контейнеров, от логирования до ограничений ресурсов, а глава 16 рассматривает проблемы, с которыми вы можете столкнуться, и предоставляет шаги для отладки и решения.

В приложениях содержатся подробные сведения об установке, использовании и настройке Docker различными способами, в том числе на виртуальной машине и в Windows.

О коде

Исходный код для всех инструментов, приложений и образов Docker, которые мы создали для использования в этой книге, доступен на GitHub в организации «docker-in-practice»: https://github.com/docker-in-practice. Образы на Docker Hub под пользователем «dockerin-practice» (https://hub.docker.com/u/dockerinpractice/) обычно представляют собой автоматические сборки из одного из репозиториев GitHub. Там, где мы чувствовали, что читателю может быть интересно дальнейшее изучение исходного кода, лежащего в основе методики, в обсуждение методики была включена ссылка на соответствующий репозиторий. Исходный код также доступен на сайте издателя по адресу: www.manning.com/books/docker-in-practice-second-edition.

Значительное количество листингов кода в книге иллюстрирует терминальную сессию, которой должен следовать читатель, вместе с соответствующим выводом команд. Следует отметить пару вещей относительно этих сессий:

- длинные терминальные команды могут использовать символ продолжения строки оболочки (\), чтобы разделить команду на несколько строк. Хотя это будет работать в вашей оболочке, если вы напечатаете его, вы также можете опустить его и набрать команду в одной строке;
- если раздел вывода не предоставляет дополнительную полезную информацию для обсуждения, он может быть опущен, а вместо него вставлено многоточие ([...]).

Книжный форум

Приобретение *Docker in Practice* (2-е изд.) включает в себя бесплатный доступ к частному веб-форуму, организованному Manning Publications, где вы можете оставлять комментарии о книге, задавать технические вопросы и получать помощь от автора и других пользователей. Чтобы получить доступ к форуму, перейдите на страницу www.manning.com/books/docker-in-practice-second-edition. Вы также можете узнать больше о форумах Manning и правилах поведения по адресу: https://forums.manning.com/forums/about.

Обязательство Manning перед своими читателями состоит в том, чтобы обеспечить место, где может состояться содержательный диалог между отдельными читателями и между читателями и автором. Это не обязательство какого-либо конкретного участия со стороны автора, чей вклад в форум остается добровольным (и не оплачивается). Мы предлагаем вам задать автору несколько сложных вопросов, чтобы его интерес не пропал.

Часть 1

Основы Docker

Первая часть этой книги состоит из глав 1 и 2, которые знакомят вас с использованием Docker и его основами.

Глава 1 объясняет происхождение Docker, а также его основные понятия, такие как образы, контейнеры и слои. Наконец, вы займетесь практикой, создав первый образ с помощью файла Dockerfile.

Глава 2 знакомит с некоторыми полезными приемами, которые помогут вам глубже понять архитектуру Docker. Беря каждый основной компонент по очереди, мы рассмотрим взаимоотношения между демоном Docker и его клиентом, реестром Docker и Docker Hub.

К концу первой части вы освоите базовые концепции Docker и сможете продемонстрировать некоторые полезные приемы, заложив прочную основу для оставшейся части книги.

Глава 1

Знакомство с Docker

О чем рассказывается в этой главе:

- что такое Docker;
- использование Docker и как он может сэкономить вам время и деньги;
- различия между контейнерами и образами;
- слои Docker;
- сборка и запуск приложения с использованием Docker.

Docker – это платформа, которая позволяет «создавать, поставлять и запускать любое приложение повсюду». За невероятно короткое время она прошла большой путь и теперь считается стандартным способом решения одного из самых дорогостоящих аспектов программного обеспечения – развертывания.

До появления Docker в конвейере разработки обычно использовались комбинации различных технологий для управления движением программного обеспечения, такие как виртуальные машины, инструменты управления конфигурацией, системы управления пакетами и комплексные сети библиотечных зависимостей. Все эти инструменты должны были управляться и поддерживаться специализированными инженерами, и у большинства из них были свои собственные уникальные способы настройки.

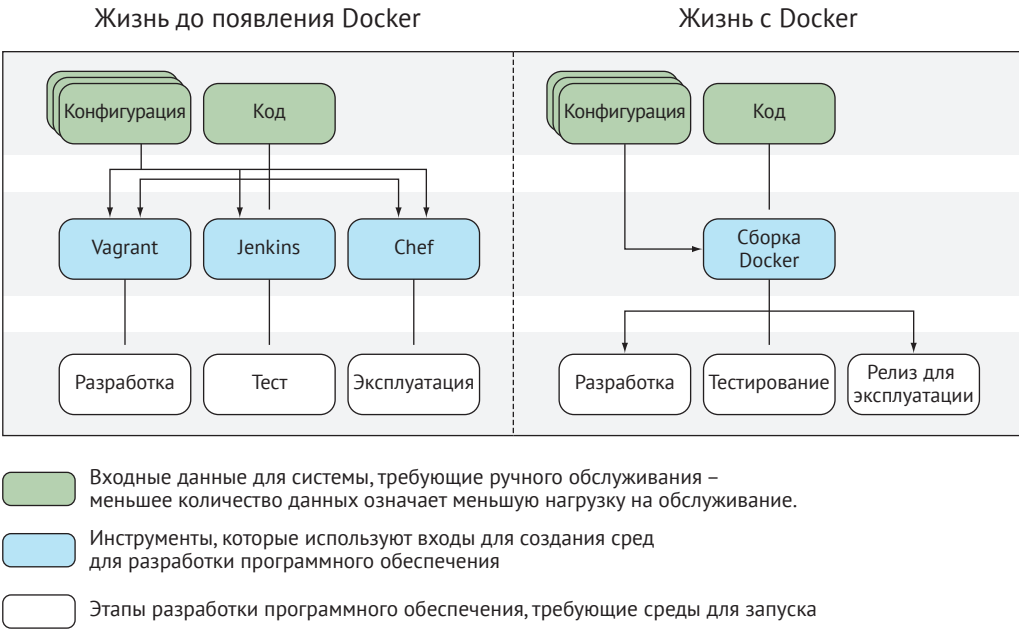


Рис. 1.1 ❖ Как Docker облегчил бремя обслуживания инструментов

Docker изменил все это, позволив различным инженерам, вовлеченным в этот процесс, эффективно говорить на одном языке, облегчая совместную работу. Все проходит через общий конвейер к одному выходу, который можно использовать для любой цели – нет необходимости продолжать поддерживать запутанный массив конфигураций инструментов, как показано на рис. 1.1.

В то же время нет необходимости выбрасывать существующий программный стек: если он работает – можно упаковать его в контейнер Docker как есть, чтобы другие могли его использовать. В качестве бонуса вы можете увидеть, как были созданы эти контейнеры, поэтому, если нужно покопаться в деталях, вы можете это сделать.

Данная книга предназначена для разработчиков среднего уровня, обладающих рядом знаний о Docker. Если вы знакомы с основами, не стесняйтесь переходить к последующим главам. Цель этой книги – раскрыть реальные проблемы, с которыми сталкивается Docker, и показать, как их можно преодолеть. Но сначала мы немного расскажем о самом Docker. Если вы хотите более подробно изучить его основы, обратите внимание на книгу Джеффа Николоффа *Docker в действии* (Manning, 2016).

В главе 2 вы познакомитесь с архитектурой Docker более подробно, с помощью ряда методов, демонстрирующих его мощь. В этой главе вы узнаете, что такое Docker, поймете, почему он важен, и начнете его использовать.

1.1 Что такое Docker и для чего он нужен

Прежде чем приступить к практике, мы немного обсудим Docker, чтобы вы поняли его контекст, то, откуда пришло название «Docker», и почему мы вообще его используем!

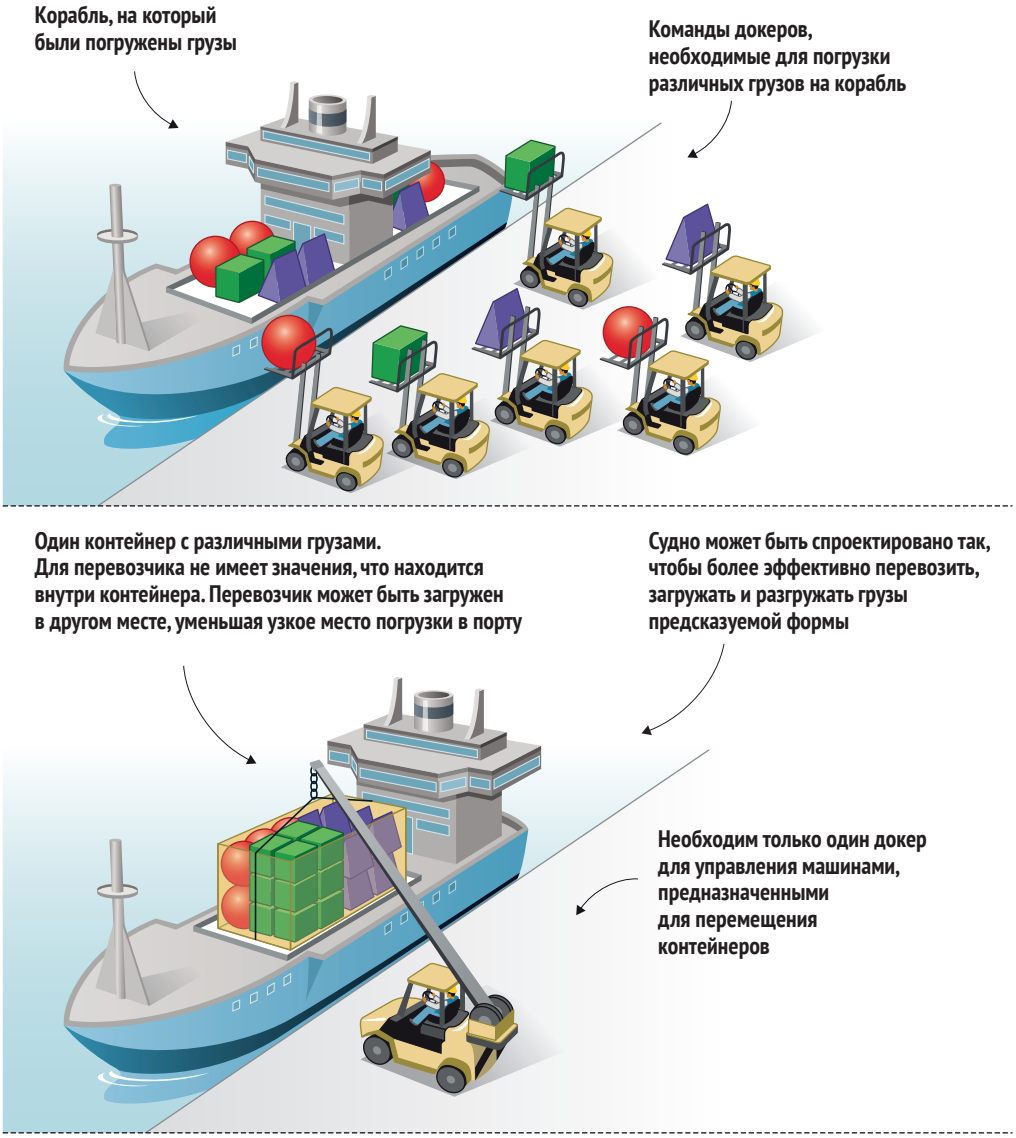


Рис. 1.2 ❖ Доставка до и после стандартизованных контейнеров

1.1.1 Что такое Docker?

Чтобы понять, что такое Docker, проще начать с метафоры, чем с технического объяснения, а метафора у Docker – мощная. *Докером (docker)* назывался рабочий, который переносил товары на суда и обратно, когда те стояли в портах. Там были ящики и грузы разных размеров и форм, и опытные докеры ценились за то, что они в состоянии вручную поместить товары на корабли экономически эффективным способом (см. рис. 1.2). Нанять людей для перемещения этих грузов было недешево, но альтернативы не было.

Это должно быть знакомо всем, кто работает в сфере программного обеспечения. Много времени и интеллектуальной энергии тратится на перенос программного обеспечения нечетной формы в метафорические корабли разного размера, заполненные другим программным обеспечением нечетной формы, чтобы их можно было продавать пользователям или предприятиям где бы то ни было.

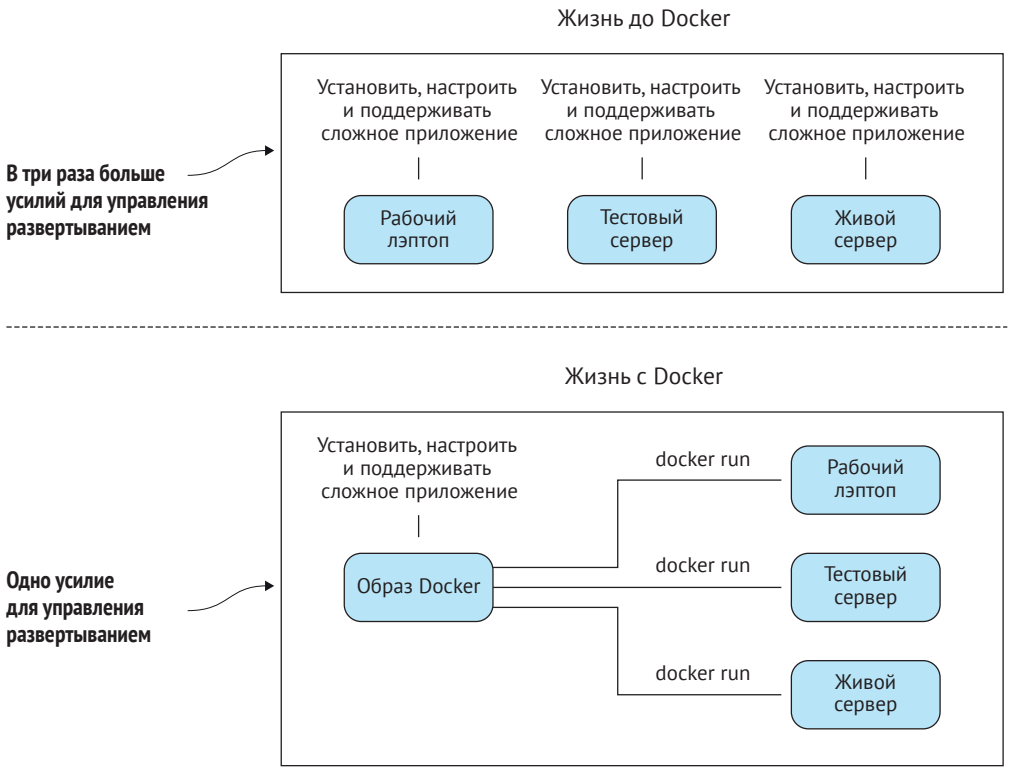


Рис. 1.3 ❖ Доставка программного обеспечения до и после Docker

На рис. 1.3 показано, как можно сэкономить время и деньги с помощью концепции Docker. До появления Docker развертывание программного обеспечения в различных средах требовало значительных усилий. Даже если вы не выполняли сценарии вручную для подготовки программного обеспечения

на разных компьютерах (а многие так и делают), вам все равно приходилось бороться с инструментами управления конфигурацией, которые регулируют состояние в быстро растущих средах, испытывающих нехватку ресурсов. Даже когда эти усилия были заключены в виртуальных машинах, много времени тратилось на управление развертыванием этих машин, ожидание их загрузки и управление накладными расходами на использование ресурсов, которые они создавали.

В Docker процесс конфигурирования отделен от управления ресурсами, а процесс развертывания тривиален: запустите `docker run`, и образ среды будет извлечен и готов к запуску, потребляя меньше ресурсов и не мешая другим средам.

Вам не нужно беспокоиться о том, будет ли ваш контейнер отправлен на компьютер с Red Hat, Ubuntu или образ виртуальной машины CentOS; до тех пор, пока там есть Docker, с ним все будет хорошо.

1.1.2 Чем хорош Docker?

Возникают некоторые важные практические вопросы: зачем вы используете Docker и для чего? Короткий ответ на вопрос «почему» заключается в том, что при минимальных усилиях Docker может быстро сэкономить вашему бизнесу много денег. Некоторые из этих способов (и далеко не все) обсуждаются в следующих подразделах. Мы видели все эти преимущества не понаслышке в реальных условиях работы.

Замена виртуальных машин

Docker может использоваться для замены виртуальных машин во многих ситуациях. Если вас волнует только приложение, а не операционная система, Docker может заменить виртуальную машину, а вы можете оставить заботу об операционной системе кому-то другому. Docker не только быстрее, чем виртуальная машина, предназначенная для запуска, он более легок в перемещении, и благодаря его многоуровневой файловой системе вы можете легче и быстрее делиться изменениями с другими. Он также прочно укоренен в командной строке и в высшей степени пригоден для написания сценариев.

Прототипирование программного обеспечения

Если вы хотите быстро поэкспериментировать с программным обеспечением, не нарушая существующую настройку и не проходя через трудности, связанные с подготовкой виртуальной машины, Docker может предоставить «песочницу» за миллисекунды. Освобождающий эффект этого процесса трудно понять, пока вы не испытаете его на себе.

Упаковка программного обеспечения

Поскольку образ Docker фактически не имеет зависимостей для пользователя Linux, это отличный способ для упаковки программного обеспечения. Вы можете создать свой образ и быть уверенным, что он может работать на любом современном компьютере с Linux – подумайте о Java без необходимости создания виртуальной машины Java.

Возможность для архитектуры микросервисов

Docker упрощает декомпозицию сложной системы на серию составных частей, что позволяет более детально рассуждать о своих сервисах. Это может позволить вам реструктурировать свое программное обеспечение, чтобы сделать его части более управляемыми и подключаемыми, не затрагивая целое.

Моделирование сетей

Поскольку вы можете запустить сотни (даже тысячи) изолированных контейнеров на одном компьютере, смоделировать сеть очень просто. Это может отлично подойти для тестирования реальных сценариев по разумной цене.

Возможность производительности полного стека в автономном режиме

Поскольку можно связать все части вашей системы в контейнеры Docker, вы можете реализовать их оркестровку для запуска на своем ноутбуке и работать в пути, даже в автономном режиме.

Сокращение неизбежных расходов на отладку

Сложные переговоры между различными командами относительно поставляемого программного обеспечения – обычное явление в данной отрасли. Мы лично пережили бесчисленные дискуссии по поводу испорченных библиотек, проблемных зависимостях, неправильного применения обновлений или их применения в неверном порядке (или даже их отсутствия), невоспроизводимых ошибок и т. д. Вероятно, вы тоже. Docker позволяет вам четко указать (даже в форме сценария) шаги для отладки проблемы в системе с известными свойствами, что значительно упрощает воспроизведение ошибок и среды и обычно позволяет отделить ее от среды хоста.

Документирование зависимостей программного обеспечения и точки взаимодействия

Создавая образы в структурированном виде, готовыми к перемещению в различные среды, Docker заставляет вас явно документировать свои программные зависимости с базовой отправной точки. Даже если вы решите не использовать Docker повсюду, эта документация может помочь вам установить программное обеспечение в других местах.

Возможность непрерывной доставки

Непрерывная доставка – это парадигма доставки программного обеспечения, основанная на конвейере, который перестраивает систему при каждом изменении, а затем осуществляет доставку в производство (или «вживую») посредством автоматизированного (или частично автоматизированного) процесса.

Поскольку вы можете более точно контролировать состояние среды сборки, сборки Docker более воспроизводимы чем традиционные методы сборки программного обеспечения. Это делает реализацию непрерывной доставки намного проще. Стандартные технологии непрерывной доставки, такие как развертывание Blue/Green (где «живое» и «последнее» развертывания

поддерживаются в реальном времени) и развертывание Phoenix (где целые системы создаются заново в каждом релизе), становятся тривиальными благодаря реализации воспроизводимого Docker-ориентированного процесса сборки.

Теперь вы немного знаете, как Docker может вам помочь. Прежде чем мы углубимся в реальный пример, давайте рассмотрим несколько основных концепций.

1.1.3 Ключевые концепции

В этом разделе мы рассмотрим некоторые ключевые концепции Docker, которые проиллюстрированы на рис. 1.4.

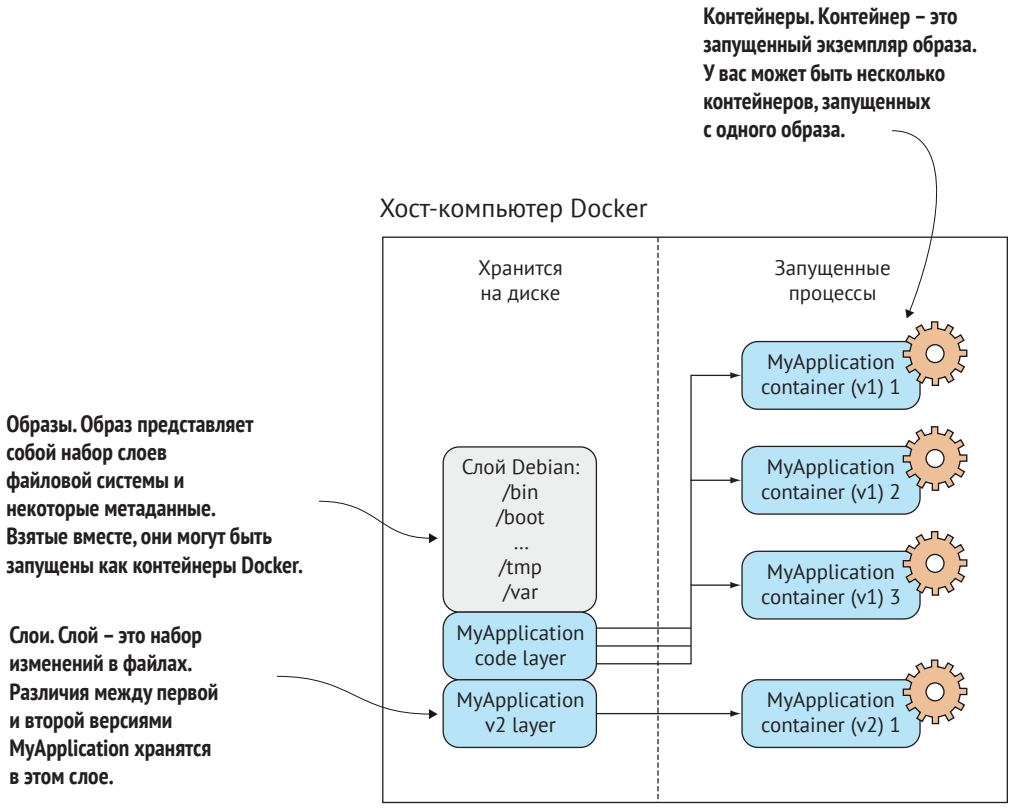


Рис. 1.4 ❖ Базовые концепции Docker

Прежде чем запускать команды Docker, лучше всего разобраться с понятиями образов, контейнеров и слоев. Говоря кратко, *контейнеры* запускают системы, определенные *образами*. Эти *образы* состоят из одного или нескольких *слоев* (или наборов различий) плюс некоторые метаданные Docker.

Давайте посмотрим на некоторые основные команды Docker. Мы превратим образы в контейнеры, изменим их и добавим слои к новым образам, которые сохраним. Не волнуйтесь, если все это звучит странно. К концу главы все станет намного понятнее.

Ключевые команды Docker

Основная функция Docker – создавать, отправлять и запускать программное обеспечение в любом месте, где есть Docker. Для конечного пользователя Docker – это программа с командной строкой, которую они запускают. Как и git (или любой другой инструмент управления исходным кодом), у этой программы есть подкоманды, которые выполняют различные операции. Основные подкоманды Docker, которые вы будете использовать на своем хосте, перечислены в табл. 1.1.

Таблица 1.1 ❖ Подкоманды Docker

Команда	Назначение
`docker build`	Собрать образ Docker
`docker run`	Запустить образ Docker в качестве контейнера
`docker commit`	Сохранить контейнер Docker в качестве образа
`docker tag`	Присвоить тег образу Docker

Образы и контейнеры

Если вы не знакомы с Docker, это может быть первый раз, когда вы встречаете слова «контейнер» и «образ» в данном контексте. Это, вероятно, самые важные концепции в Docker, поэтому стоит потратить немного времени, чтобы убедиться, что разница ясна. На рис. 1.5 вы увидите иллюстрацию этих концепций с использованием трех контейнеров, запущенных из одного базового образа.

Один из способов взглянуть на образы и контейнеры – это рассматривать их как программы и процессы. Точно так же процесс может рассматриваться как «выполняемое приложение», контейнер Docker может рассматриваться как образ, выполняемый Docker.

Если вы знакомы с принципами объектно-ориентированного программирования, еще один способ взглянуть на образы и контейнеры – это рассматривать образы как классы, а контейнеры – как объекты. Точно так же, как объекты представляют собой конкретные экземпляры классов, контейнеры являются экземплярами образов. Вы можете создать несколько контейнеров из одного образа, и все они будут изолированы друг от друга так же, как и объекты. Что бы вы ни изменили в объекте, это не повлияет на определение класса – это принципиально разные вещи.

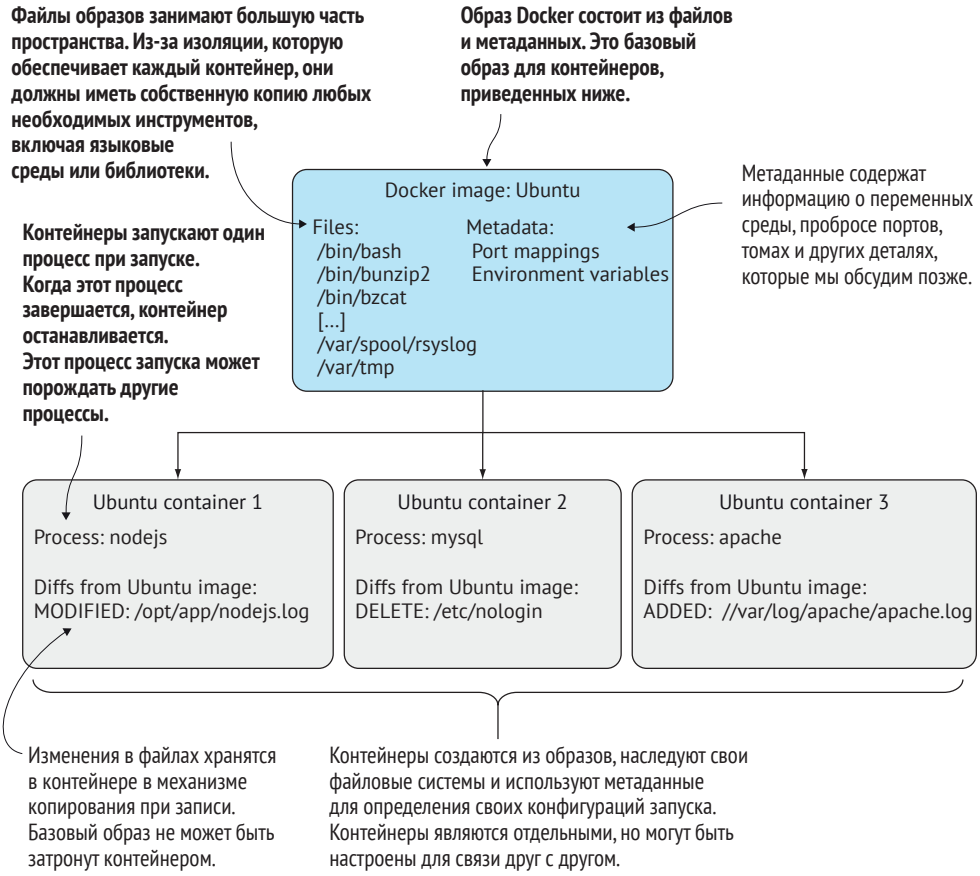


Рис. 1.5 ❖ Образы и контейнеры Docker

1.2 Создание приложения Docker

Теперь мы перейдем к практике, создав простой образ приложения в формате to-do с помощью Docker. В ходе этого процесса вы встретитесь с некоторыми ключевыми функциями Docker, такими как файлы Dockerfiles, повторное использование образов, отображение портов и автоматизация сборки. Вот что вы узнаете в течение следующих 10 минут:

- как создать образ Docker с помощью Dockerfile;
- как присвоить тег образу Docker для удобства пользования;
- как запустить свой новый образ Docker.

Приложение в формате to-do – это приложение, которое помогает вам отслеживать то, что вы хотите сделать. Приложение, создаваемое нами, будет хранить и отображать короткие строки информации, которые можно пометить как выполненные, представленные в простом веб-интерфейсе. На рис. 1.6 показано, чего мы добьемся благодаря этому.

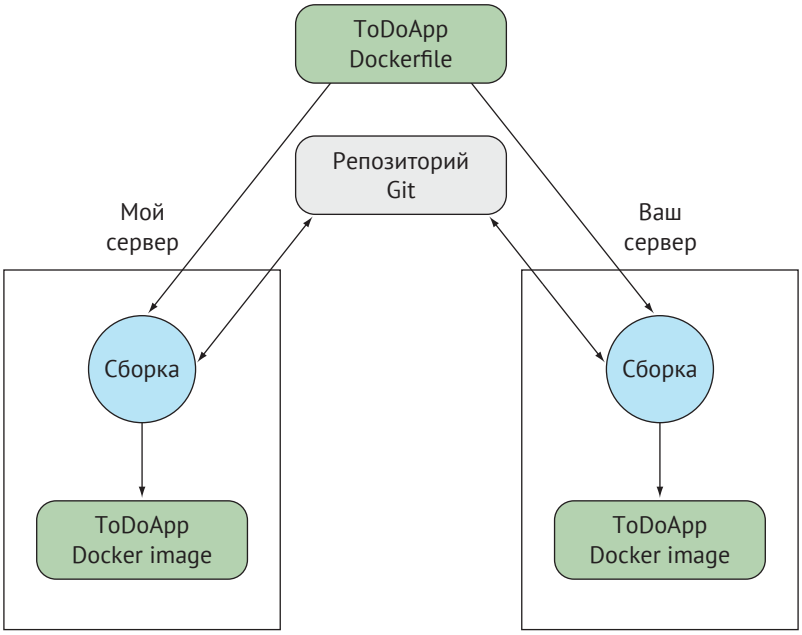


Рис. 1.6 ❖ Создание приложения Docker

Детали приложения не важны. Мы продемонстрируем, что из одного короткого файла Dockerfile, который мы собираемся вам дать, вы можете гарантированно создавать, выполнять, останавливать и запускать приложение как на своем, так и на нашем хосте, не беспокоясь об установке или зависимостях. Это ключевая часть того, что предлагает Docker – надежно воспроизводимые, легко управляемые и совместно используемые среды разработки. Это означает, что не нужно следовать более сложным или неоднозначным инструкциям по установке, в которых можно потеряться.

> **ПРИМЕЧАНИЕ.** Это приложение будет использоваться неоднократно на протяжении всей книги, и оно довольно полезно для экспериментов и демонстраций, так что стоит с ним ознакомиться.

1.2.1 Способы создания нового образа Docker

Существует четыре стандартных способа создания образов Docker. Они перечислены в табл. 1.2

Таблица 1.2 ❖ Методы создания образов Docker

Метод	Описание	Источник
Команды Docker/ «От руки»	Запустите контейнер с помощью `docker run` и введите команды для создания образа в командной строке. Создайте новый образ с помощью `docker commit`	См. метод 15
Dockerfile	Выполните сборку из известного базового образа и укажите ее с помощью ограниченного набора простых команд	Будет обсуждаться в скором времени
Dockerfile и инструмент управления конфигурацией	То же самое, что и Dockerfile, но вы передаете контроль над сборкой более сложному инструменту управления конфигурацией	См. метод 55
Стереть образ и импортировать набор файлов	Из пустого образа импортируйте файл TAR с необходимыми файлами	См. метод 11

Первый вариант «от руки» подойдет, если вы проверяете концепцию, чтобы увидеть, работает ли ваш процесс установки. В то же время вы должны вести записи о предпринимаемых вами шагах, чтобы при необходимости иметь возможность вернуться к нужной вам точке.

В какой-то момент вы захотите определить шаги для создания своего образа.

Это опция Dockerfile (и та опция, которую мы будем использовать здесь).

Для более сложных сборок вы можете выбрать третий вариант, особенно когда функции Dockerfile недостаточно сложны для вашего образа.

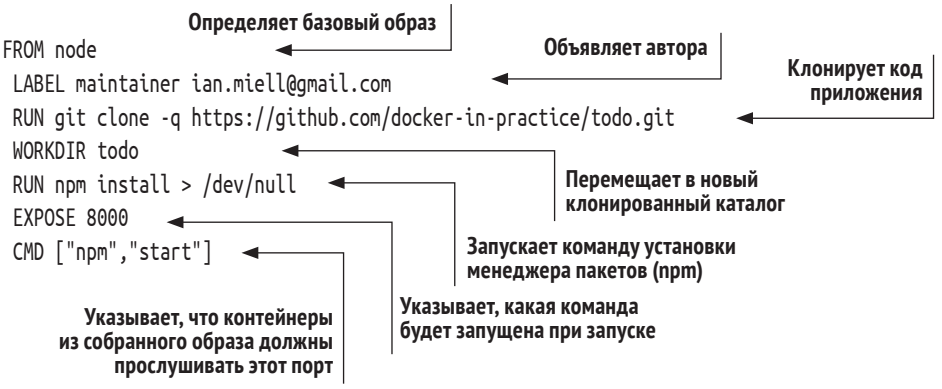
Последний вариант строится из нулевого образа путем наложения набора файлов, необходимых для запуска образа. Это полезно, если вы хотите импортировать набор автономных файлов, созданных в другом месте, но этот метод редко встречается при массовом использовании.

Сейчас мы рассмотрим метод Dockerfile; другие будут изучены позже.

1.2.2 Пишем Dockerfile

Dockerfile – это текстовый файл, содержащий серию команд. В листинге 1.1 приведен файл Dockerfile, который мы будем использовать в этом примере. Заведите новую папку, перейдите в нее и создайте файл с именем «Dockerfile» с таким содержимым:

Листинг 1.1. Файл Dockerfile



Файл Dockerfile начинается с определения базового образа с помощью команды FROM. В этом примере используется образ Node.js, поэтому у вас есть доступ к двоичным файлам Node.js. Официальный образ Node.js называется node.

Далее вы объявляете автора с помощью команды LABEL. В этом случае мы используем один из наших адресов электронной почты, но вы можете заменить его своей ссылкой, поскольку теперь это ваш файл Dockerfile. Эта строка не обязательна для создания рабочего образа Docker, но рекомендуется ее включить. На этом этапе сборка унаследовала состояние node-контейнера, и вы готовы работать над ним.

Затем вы клонируете код приложения с помощью команды RUN. Указанная команда используется для получения кода приложения, выполняя git внутри контейнера. В этом случае Git устанавливается внутри базового образа, но нельзя принимать это как должное.

Теперь вы перемещаетесь в новый клонированный каталог с помощью команды WORKDIR. Это не только меняет каталоги в контексте сборки, но и последняя команда WORKDIR определяет, в каком каталоге вы находитесь по умолчанию, когда запускаете свой контейнер из собранного образа.

Затем вы запускаете команду установки менеджера пакетов Node.js (npm). Она установит зависимости для вашего приложения. В этом примере вывод вас не интересует, поэтому вы перенаправляете его в /dev/null.

Поскольку порт 8000 используется приложением, вы применяете команду EXPOSE, чтобы сообщить Docker, что контейнеры из собранного образа должны прослушивать этот порт.

Наконец, вы используете команду CMD, чтобы сообщить Docker, какая команда будет выполнена при запуске контейнера.

Этот простой пример иллюстрирует несколько ключевых особенностей Docker и файлов Dockerfiles. Dockerfile – это простая последовательность ограниченного набора команд, выполняемых в строгом порядке. Это оказывает воздействие на файлы и метаданные полученного образа. Здесь команда RUN

влияет на файловую систему, проверяя и устанавливая приложения, а команды `EXPOSE`, `CMD` и `WORKDIR` – на метаданные образа.

1.2.3 Собираем образ Docker

Вы определили шаги сборки своего файла Dockerfile. Теперь вы собираетесь создать из него образ Docker, набрав команду, показанную на рис. 1.7. Вывод будет выглядеть примерно так:

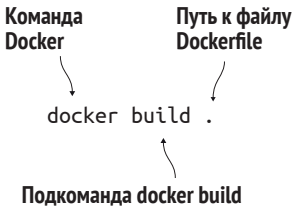


Рис. 1.7 ❖ Команда сборки

```
Sending build context to Docker daemon  2.048kB
Step 1/7 : FROM node
 ---> 2ca756a6578b
Step 2/7 : LABEL maintainer ian.miell@gmail.com
 ---> Running in bf73f87c88d6
 ---> 5383857304fc
Removing intermediate container bf73f87c88d6
Step 3/7 : RUN git clone -q https://github.com/docker-in-practice/todo.git
 ---> Running in 761baf524cc1
 ---> 4350cb1c977c
Removing intermediate container 761baf524cc1
Step 4/7 : WORKDIR todo
 ---> a1b24710f458
Removing intermediate container 0f8cd22fbe83
Step 5/7 : RUN npm install > /dev/null
 ---> Running in 92a8f9ba530a
npm info it worked if it ends with ok
[...]
npm info ok
 ---> 6ee4d7bba544
Removing intermediate container 92a8f9ba530a
Step 6/7 : EXPOSE 8000
 ---> Running in 8e33c1ded161
 ---> 3ea44544f13c
```

Каждая команда приводит к созданию нового образа с выводом его идентификатора

Docker загружает файлы и каталоги по пути, предоставленному команде docker build

Каждый шаг сборки последовательно нумеруется, начиная с 1, и выводится командой

Для экономии места каждый промежуточный контейнер удаляется, перед тем как продолжить

Отладка сборки выводится здесь (и редактируется из этого списка)

```
Removing intermediate container 8e33c1ded161
Step 7/7 : CMD npm start
 ---> Running in ccc076ee38fe
 ---> 66c76cea05bb
Removing intermediate container ccc076ee38fe
Successfully built 66c76cea05bb
```

Окончательный идентификатор образа для этой сборки, готовый к присвоению тега

Теперь у вас есть образ Docker со своим идентификатором («66c76cea05bb» в предыдущем примере, но ваш идентификатор будет другим). Возможно, к нему неудобно обращаться, поэтому вы можете присвоить ему тег для удобства, как показано на рис. 1.8.

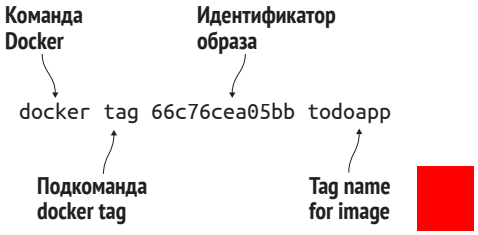


Рис. 1.8 ❖ Подкоманда docker tag

Введите предыдущую команду, заменив 66c76cea05bb сгенерированным для вас идентификатором образа.

Теперь вы можете собрать свою собственную копию образа Docker из файла Dockerfile, воспроизводя среду, определенную кем-то другим!

1.2.4 Запускаем контейнер Docker

Вы собрали свой образ Docker и присвоили ему тег. Теперь вы можете запустить его в качестве контейнера:

Листинг 1.2. Вывод docker run для todoapp

```
$ docker run -i -t -p 8000:8000 --name example1 todoapp
npm install
npm info it worked if it ends with ok
npm info using npm@2.14.4
npm info using node@v4.1.1
npm info prestart todomvc-swarm@0.0.1

> todomvc-swarm@0.0.1 prestart /todo
 > make all

npm install
```

Подкоманда docker run запускает контейнер, -p перенаправляет порт контейнера 8000 в порт 8000 на хост-компьютере, --name присваивает контейнеру уникальное имя, а последний аргумент – это образ

Вывод процесса запуска контейнера отправляется на терминал

```
npm info it worked if it ends with ok
npm info using npm@2.14.4
npm info using node@v4.1.1
npm WARN package.json todomvc-swarm@0.0.1 No repository field.
npm WARN package.json todomvc-swarm@0.0.1 license should be a valid SPDX
➥ license expression
npm info preinstall todomvc-swarm@0.0.1
npm info package.json statics@0.1.0 license should be a valid SPDX license
➥ expression
npm info package.json react-tools@0.11.2 No license field.
npm info package.json react@0.11.2 No license field.
npm info package.json node-
     jsx@0.11.0 license should be a valid SPDX license expression
npm info package.json ws@0.4.32 No license field.
npm info build /todo
npm info linkStuff todomvc-swarm@0.0.1
npm info install todomvc-swarm@0.0.1
npm info postinstall todomvc-swarm@0.0.1
npm info prepublish todomvc-swarm@0.0.1
npm info ok
if [ ! -e dist/ ]; then mkdir dist; fi
cp node_modules/react/dist/react.min.js dist/react.min.js

LocalTodoApp.js:9: // TODO: default english version
LocalTodoApp.js:84: fwdList = this.host.get('/TodoList#'+listId);
 // TODO fn+id sig
TodoApp.js:117: // TODO scroll into view
TodoApp.js:176: if (i>=list.length()) { i=list.length()-1; } // TODO
➥ .length
local.html:30: <!-- TODO 2-split, 3-split -->
model/TodoList.js:29: // TODO one op - repeated spec? long spec?
view/Footer.jsx:61: // TODO: show the entry's metadata
view/Footer.jsx:80: todoList.addObject(new TodoItem()); // TODO
➥ create default
view/Header.jsx:25: // TODO list some meaningful header (apart from the
➥ id)

npm info start todomvc-swarm@0.0.1

> todomvc-swarm@0.0.1 start /todo
> node TodoAppServer.js
```

```
Swarm server started port 8000
^Cshutting down http-server...
closing swarm host...
swarm host closed
npm info lifecycle todomvc-swarm@0.0.1~poststart: todomvc-swarm@0.0.1
npm info ok
$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b9db5ada0461 todoapp "npm start" 2 minutes ago Exited (0) 2 minutes ago
➥ example1
$ docker start example1
example1
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
➥ PORTS NAMES
b9db5ada0461 todoapp "npm start" 8 minutes ago Up 10 seconds
➥ 0.0.0.0:8000->8000/tcp example1
$ docker diff example1
C /root
C /root/.npm
C /root/.npm/_locks
C /root/.npm/anonymous-cli-metrics.json
C /todo
 A /todo/.swarm
 A /todo/.swarm/_log
A /todo/dist
A /todo/dist/LocalTodoApp.app.js
A /todo/dist/TodoApp.app.js
A /todo/dist/react.min.js
C /todo/node_modules
```

Нажмите здесь сочетание клавиш Ctrl-C, чтобы завершить процесс и контейнер

Запустите эту команду, чтобы увидеть контейнеры, которые были запущены и удалены, а также идентификатор и состояние (например, процесс)

Перезапустите контейнер, на этот раз в фоновом режиме

Run the ps command again to see the changed status.

Подкоманда docker diff показывает, какие файлы были затронуты с момента создания экземпляра образа как контейнера

Каталог /todo был изменен (C)

Добавлен каталог /todo/.swarm (A)

Подкоманда `docker run` запускает контейнер. Флаг `-p` перенаправляет порт контейнера 8000 в порт 8000 на хост-компьютере, поэтому теперь вы можете перейти в своем браузере по адресу http://localhost:8000 для просмотра приложения.

Флаг `-name` присваивает контейнеру уникальное имя, к которому вы можете обратиться позже для удобства. Последний аргумент – это имя образа.

Как только контейнер будет запущен, нажмите сочетание клавиш **Ctrl-C**, чтобы завершить процесс и контейнер. Вы можете выполнить команду `ps`, чтобы увидеть контейнеры, которые были запущены, но не были удалены. Обратите внимание, что у каждого контейнера есть свой идентификатор и статус, аналогичный процессу. Его статус `Exited`, но вы можете перезапустить его. После того, как вы это сделаете, обратите внимание, что статус изменился на `Up`, и теперь виден порт, перенаправляемый из контейнера в хост-компьютер.

Подкоманда `docker diff` показывает, какие файлы были затронуты с момента создания экземпляра образа как контейнера. В этом случае каталог todo был изменен (C), и были добавлены другие перечисленные файлы (A). Файлы не были удалены (D) – это другая возможность.

Как видно, тот факт, что Docker «содержит» вашу среду, означает, что вы способны рассматривать ее как сущность, над которой можно предсказуемо выполнять действия. Это дает Docker широкие возможности – влиять на жизненный цикл программного обеспечения от разработки до эксплуатации и обслуживания. Эти изменения – то, о чем пойдет речь в этой книге, показывая вам на практике, что можно сделать с помощью Docker.

Далее вы узнаете о слоях, еще одной ключевой концепции Docker.

1.2.5 Слои Docker

Слои Docker помогают справиться с большой проблемой, которая возникает, когда вы используете контейнеры в широком масштабе. Представьте себе, что произойдет, если вы запустите сотни или даже тысячи приложений, и каждому из них потребуется копия файлов для хранения в каком-либо месте.

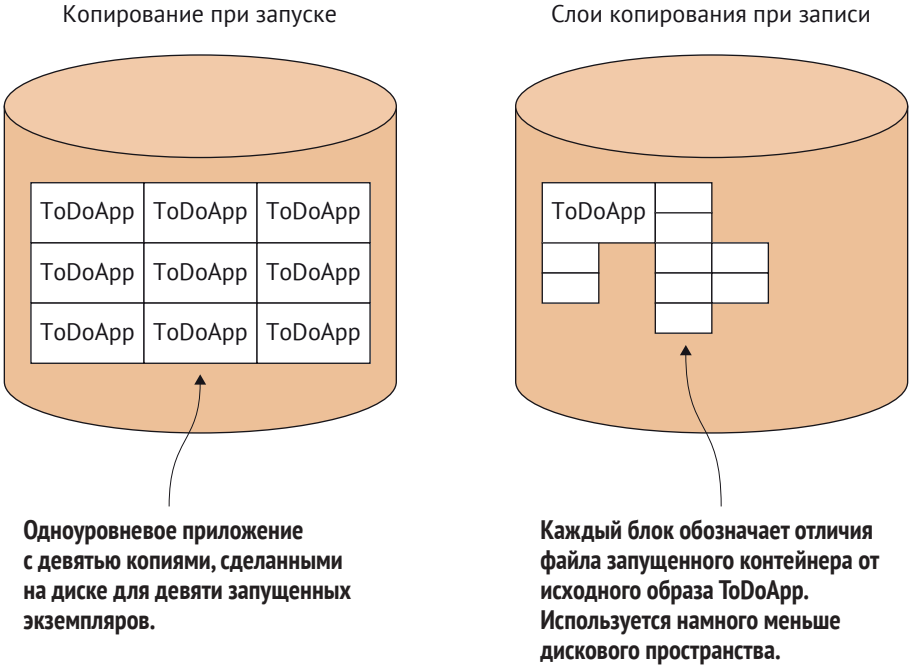


Рис. 1.9 ❖ Слои файловой системы Docker

Как вы можете себе представить, дисковое пространство закончится довольно быстро! По умолчанию Docker внутренне использует механизм копирования при записи, чтобы уменьшить объем требуемого дискового пространства (см. рис. 1.9). Всякий раз, когда работающему контейнеру необходимо

выполнить запись в файл, он записывает изменение, копируя элемент в новую область диска. После выполнения фиксации новая область диска замораживается и записывается как слой со своим собственным идентификатором.

Это отчасти объясняет, как контейнеры Docker могут так быстро запускаться – им нечего копировать, потому что все данные уже сохранены в виде образа.

> **ПОДСКАЗКА.** Копирование при записи – это стандартная стратегия оптимизации, используемая в вычислительной технике. Когда вы создаете новый объект (любого типа) из шаблона, а не копируете весь требуемый набор данных, вы копируете данные только после их изменения. В зависимости от варианта использования это может сэкономить значительные ресурсы.

На рис. 1.10 показано, что созданное вами приложение содержит три слоя, которые вам интересны. Слои статичны, поэтому, если вам нужно что-то изменить в более высоком слое, можно просто выполнить сборку поверх образа, который вы хотите взять в качестве ссылки. В своем приложении вы создали общедоступный node-образ и многоуровневые изменения сверху.

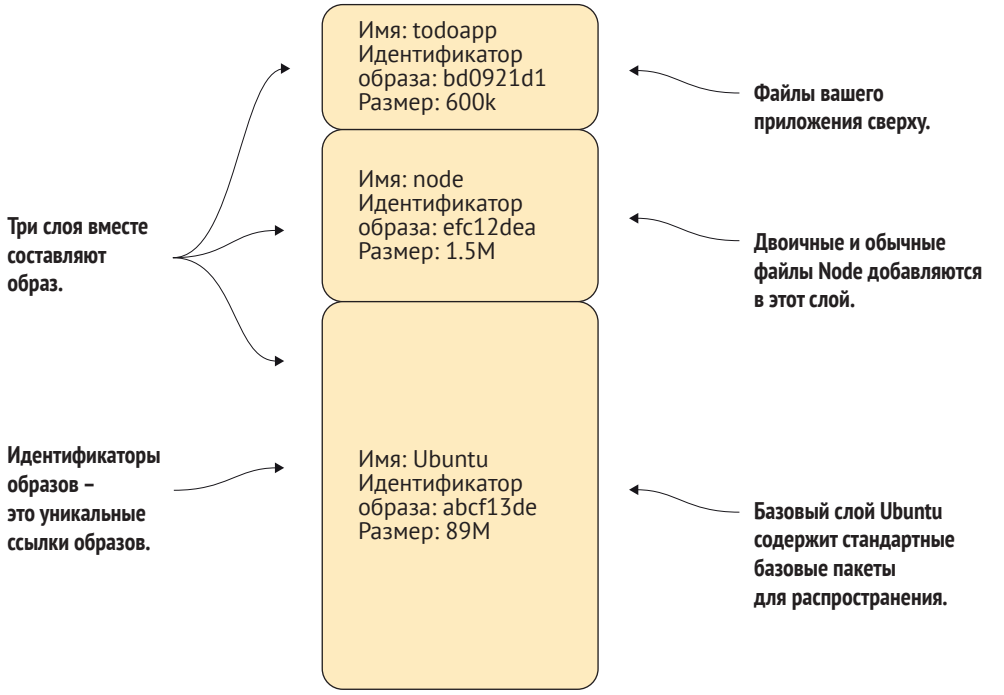


Рис. 1.10 ❖ Концепция слоев в файловой системе todoapp в Docker

Все три слоя могут совместно использоваться несколькими запущенными контейнерами, так же как общая библиотека может совместно использоваться

в памяти несколькими запущенными процессами. Это жизненно важная функция для операций, позволяющая запускать многочисленные контейнеры на основе разных образов на хост-компьютерах, не испытывая нехватки дискового пространства.

Представьте, что вы запускаете свое приложение как сервис для платных клиентов в режиме реального времени. Вы можете расширить свое предложение для большого количества пользователей. Если вы занимаетесь разработкой, то можно запустить много разных сред на локальном компьютере одновременно. Если вы проходите тестирование, можно одновременно выполнять гораздо больше тестов и значительно быстрее, чем раньше. Все это стало возможным благодаря слоям.

Создав и запустив приложение с помощью Docker, вы начали осознавать всю мощь, которую Docker может внести в рабочий процесс. Воспроизведение и совместное использование определенных сред и возможность их размещения в разных местах дают вам гибкость и контроль над разработкой.

Резюме

- Docker – это попытка сделать для программного обеспечения то, что контейнеризация сделала для судоходной отрасли: снизить стоимость локальных различий за счет стандартизации.
- Некоторые из применений Docker включают в себя программное обеспечение для создания прототипов, программное обеспечение для упаковки, снижение затрат на тестирование и отладку сред, а также использование методологий DevOps, таких как непрерывная доставка.
- Вы можете создавать и запускать приложение Docker из файла Dockerfile, используя команды `docker build` и `docker run`.
- Образ Docker – это шаблон для работающего контейнера. Это похоже на разницу между исполняемым файлом программы и запущенным процессом.
- Изменения в запущенных контейнерах можно сохранять и тегировать как новые образы.
- Образы создаются из многоуровневой файловой системы, что уменьшает пространство, используемое образами Docker на вашем хосте.