# 3. Контейнеры и Docker

Контейнеры (containers) - относительно новое слово и концепция, мгновенно захватившая мир разработки программного обеспечения за последние несколько лет. Это относительно новое достижение в попытке разработчиков и системных инженеров максимально использовать доступные им вычислительные ресурсы.

Если кратко вспомнить историю, то серверные приложения, сервисы и базы данных изначально располагались на выделенных для них физических серверах, в подавляющем большинстве случаев под управлением одного из вариантов операционной системы Unix (или ее клона Linux). С взрывным ростом вычислительных мощностей использовать один мощнейший сервер для одного приложения стало и расточительно, и неэффективно - на одном сервере стали работать несколько приложений, внутренних сервисов или даже баз данных. При этом незамедлительно возникли серьезные трудности - различные версии приложений использовали разные версии основных библиотек Unix, использовали разные несовместимые между собой пакеты расширений или дополнительные библиотеки, соревновались за одинаковые номера портов, особенно если они были широко используемы (HTTP 80, HTTPS 443 и т.п.)

Разработчики работали над своим продуктом и тестировали его на отдельно выделенных серверах для тестирования (среда разработки, development environment, или же дальше в среде тестирования, QA environment). На этих серверах сочетание приложений и сервисов было хаотичным и постоянно менялось в зависимости от этапа разработки, и как правило не совпадало с состоянием производственной (production) среды. Системным администраторам серверов пришлось особенно тяжело - совмещать созданные в изоляции приложения необходимо было развертывать и запускать в производственной среде (production), жонглируя при этом общим доступом к ресурсам, портам, настройкам и всему остальному. Надежность системы во многом зависела от качества настройки ее производственной среды.

Следующим решением, популярным и сейчас, стала виртуализация на уровне операционной системы. Основной идеей была работа независящих друг от друга операционных систем на одном физическом сервере. Обеспечивал разделение всех физических ресурсов, прежде всего процессорного времени, памяти и дисков с данными так называемый гипервизор (hypervisor). Делал он это прямо на аппаратном уровне (гипервизор первого типа) или же уже на уровне существующей базовой операционной системы (гипервизор второго уровня). Разделение на уровне операционной системы радикально улучшило и упростило настройку гетерогенных, разнородных систем в средах разработки и производства. Для работы отдельных приложений и баз данных на мощный сервер устанавливалась отдельная виртуальная операционная система, которую и стали называть виртуальной машиной (virtual machine), так как отличить ее “изнутри” от настоящей, работающей на аппаратном обеспечении ОС невозможно. На мощном сервере могут работать десятки виртуальных машин, имеющих соответственно в десятки раз меньше ресурсов, но при это совершенно независимые друг от друга. Приложения теперь свободны настраивать систему и ее библиотеки, зависимости и внутреннюю структуру как им вздумается, не задумываясь об ограничениях из-за присутствия других систем и сервисов. Виртуальные машины подсоединяются к общей сети, и могут иметь отдельный IP-адрес, не зависящий от адреса своего физического сервера.

Именно виртуальные машины являются краеугольным камнем облачных вычислений. Основные провайдеры облачных услуг Cloud (Amazon, Google, Microsoft и другие) обладают огромными вычислительными мощностями. Их центры обработки данных (data center) состоят из большого количества мощных серверов, соединенных между собой и основным Интернетом сетями с максимально возможной пропускной способностью (bandwidth). “Арендовать” себе целый сервер, постоянно работающий и присутствующий в сети Интернет, было бы очень дорого и особенно невыгодно для только начинающих компаний-стартапов, или проектов в стадии зарождения, которым не нужны большие мощности. Вместо этого провайдеры облака продают виртуальные машины разнообразных видов - начиная от самых микроскопических, по сути “слабее” чем любой современный смартфон - но этого зачастую более чем достаточно для небольшого сервера, обслуживающего не более чем несколько простых запросов в минуту или того меньше. Более того, виртуальные машины оптимизированы - новая виртуальная машина создается по запросу в течение нескольких минут, версия операционной системы всегда проверена на уязвимости и быстро обновляется. Если виртуальная машина больше не нужна, ее можно быстро остановить и перестать платить за использование облачных ресурсов.

Тем не менее, виртуальные машины, несмотря на то что выглядят совершенно универсальным инструментом облака, обеспечивающим полную изоляцию на уровне операционной системы, имеют существенный недостаток. Это по прежнему полноценная операционная система, и учитывая сложность аппаратного обеспечения, большое количество драйверов, поддержку сети, основных библиотек, встроенное управление пакетами расширения (package manager), и наконец интерпретатор команд (shell), все это выливается во внушительным размер системы, и достаточно долгое время первоначальной инициализации (минуты). Для некоторых случаев это может не являться препятствием - в том случае если ожидается что количество виртуальных машин фиксировано или скорость их инициализации не является критической, и на каждой из них работает большое приложение, останавливать и перезапускать которое часто нет необходимости.

Однако, подобное предположение все чаще является препятствием для приложений с высокой скоростью разработки и постоянным появлением новой функциональности, и особенно это критично для архитектуры на основе микросервисов.

* Микросервисы в идеале очень малы, и даже самая слабая виртуальная машина может быть слишком неэффективна для них, как по избыточной мощности, так и по цене
* Одно из основных теоретических преимуществ микросервисов - быстрое, почти мгновенное масштабирование при увеличении нагрузки на них. В мире где удачное приложение собирает миллионы запросов в секунду, ожидание нескольких минут для появления следующей копии сервиса просто недопустимо и практически лишает легкую масштабируемость смысла.
* Опять же из-за их малого размера микросервисы могут иметь намного меньше зависимостей и требований к операционной системе в которой они работают - полная операционная система, ограниченная гипервизором, является для них чрезмерным ресурсом.

Именно так на свет появилась следующая идея - контейнеры. Контейнеры позволяют перейти на следующий уровень разделения вычислительных ресурсов, не налагая на приложение, работающей в облаке, необходимость нести с собой операционную систему виртуальной машины и связанные с этим издержки.

## Контейнеры - это Linux

Давайте сразу определим для себя, что представляют собой контейнеры, и чем они отличаются от виртуальных машин, чтобы избежать путаницы которая часто случается между ними. Контейнер - это набор ограничений для запуска приложений, которые поддерживаются ядром (kernel) операционной системы Linux. Эти ограничения заставляют приложение исполняться в закрытой файловой системе, со своим пространством процессов (приложение не видит процессы вне своей группы), и с квотами на использование памяти, мощности процессоров CPU, дисков, и возможно сети. При этом у приложения в таком ограниченном пространстве существует свой сетевой IP-адрес и полный набор портов, а также полная поддержка ядра системы - устройств ввода/вывода, управление памятью и процессором, многозадачность, и наконец самое главное, возможность установить любые расширения и библиотеки, не беспокоясь о конфликтах с другими приложениями.

Можно сказать, что приложение, запускающееся в контейнере Linux, “видит” стандартное ядро (kernel) операционной системы, так, как если бы ничего кроме этого приложения, и этого ядра, больше не существовало. Файловая система пуста, нет никаких дополнительных пакетов и библиотек, интерпретаторов shell и тем более никакого намека на графический интерфейс GUI. Примерно так же “ощущает” себя приложение, запускающееся в виртуальной машине, на которой установлена операционная система Linux, только в крайне урезанном варианте.

Иметь только возможности ядра Linux для большинства современных приложений недостаточно, они как правило зависят от множества расширений и библиотек, а также имеют свои файлы с данными. Здесь контейнеры также хороши - приложение свободно распоряжаться своим пространством контейнера так, как если бы оно находилось на своем отдельном виртуальном (или реальном) сервере. Возможно установить любые пакеты, расширения, библиотеки и скопировать файлы и данные внутрь контейнера, не опасаясь конфликтов. Все это будет закрыто внутри контейнера и недоступно другим приложениям из других контейнеров.

Запуск и настройка контейнеров может вылиться в большое количество связанных между собой системных команд, и требует определенных знаний команд и архитектуры Linux. Здесь свою нишу занял инструмент Docker, который великолепно справляется с настройкой и запуском контейнеров, именно он стал еще одной причиной взрывной популярности контейнеров, которые отныне не требуют расширенных знаний Linux.

“Что же, если я использую Mac OS или Windows?” спросите вы? Благодаря инструментам Docker контейнеры Linux доступны на любых популярных операционных системах. Еще раз вспомним, что для работы контейнеров требуется доступ к минимальному ядру Linux - именно это и предоставляет Docker, как правило с помощью скрытой внутри него минимальной виртуальной машиной. Отличная возможность не только использовать контейнеры, но и получить доступ к настоящему ядру Linux за считанные секунды на любой операционной системе и лэптопе без запуска тяжелых виртуальных машин с полной операционной системой!

Хотя идея контейнеров и сама их суть - это операционная система Linux, их растущая популярность и прекрасно подходящая для облачных приложений архитектура не могла пройти мимо всех провайдеров облака, особенно облака Microsoft Azure с большим фокусом внимания на серверные варианты операционных систем Windows. Компания Microsoft уже поддерживает контейнеры (и Docker) в своих операционных системах без использования виртуальных машин. Конечно это уже не Linux, но если вы работаете с кодом .Net, это неоценимая возможность использовать архитектуру контейнеров не переписывая свои приложения.

## Docker

Контейнеры - отличная замена виртуальным машинам. Не требующие гипервизора, и всей тяжелой массы полной операционной системы, им нужна лишь уже установленная подходящая версия Linux с поддержкой контейнеров. Остается “только лишь” вопрос их настройки, запуска, остановки, управления ресурсами, файловыми системами, совместными данными и их томами, и многим другим. В принципе для этого не требуется ничего кроме инструментов Linux, но кривая обучения и время настройки всего этого будет весьма значимы. Все эти функции берет на себя инструмент Docker, и именно его легкость и логичность его команд и общей модели помогли контейнерам выйти на авансцену облачных вычислений намного быстрее.

Вы легко можете установить инструменты Docker на любые операционные системы, и он при необходимости обеспечит вам минимальную виртуальную машину с ядром Linux (смотрите сайт docker.com, вам понадобится бесплатная версия Docker CE, в ней есть все что необходимо для работы с любыми контейнерами).

После установки Docker можно начинать запускать контейнеры. Еще раз вспомним, что контейнер - ограниченная часть пространства операционной системы Linux, в которой есть ограниченный и прозрачный для приложения доступ ко всем функциям ядра (kernel) системы. Никаких интерпретаторов команд shell, известных всем команд ls, ps или curl там просто нет. Соответственно, выполнить там можно только приложение без зависимостей или библиотек (статически скомпонованное, static linking, что означает что все возможные зависимости и библиотеки находятся внутри исполняемого файла программы). Зачастую этого слишком мало, особенно если вы рассчитываете на отладку приложения, хотите исследовать его журналы (logs), или посмотреть список процессов и использование ими ресурсов.

Чтобы не копировать каждый раз нужные инструменты вручную или сложными скриптами, в Docker ключевую роль играет уже готовый набор файлов и библиотек, который можно одной командой перенести в свой контейнер и спокойно там получать к ним доступ. Это так называемые образы (image) контейнеров.

## Образы (image) Docker

Как мы уже поняли, сам контейнер в своей основе - лишь возможность вызывать системные функции и использовать сервисы ядра Linux, и в него необходимо перенести файлы с приложением, его зависимости, и возможно инструменты для отладки и диагностики приложения, включая интерпретатор команд shell, если понадобится взаимодействовать с контейнером в интерактивном режиме. Весь этот набор в архитектуре Docker хранится в образе (image). Образ - это статический набор файлов, инструментов, директорий, символических ссылок symlink, словом всего того, что требуется приложению и нам как его разработчикам, чтобы успешно его запустить и при необходимости отладить или диагностировать проблему.

Созданные один раз образы могут использоваться как заготовка для запуска контейнеров вновь и вновь. По умолчанию они хранятся в хранилище на вашем рабочем компьютере, а также могут иметь уникальные метки (tag, для отметки особенно важных или удачных образов). Все это удивительно напоминает систему контроля версий, и именно так работает хранение образов Docker на вашей машине. Более того, образы можно хранить в удаленном репозитории в сети, аналогично тому как храним версии и ветки исходного кода, например в GitHub. Создатели Docker поняли полную аналогию между хранением образов, необходимость разнообразных версий с помощью меток, и практически полностью скопировали команды Git для управления образами, а общедоступное для всех удаленное хранилище для них назвали конечно же… Docker Hub.

Чуть позже мы подробно узнаем про управление образами, а пока самое интересное для нас заключается в том, что в Docker Hub, также как в GitHub, есть открытые (public) образы, на основе которых можно запускать свои контейнеры, ничего не меняя в них, или же строить на основе них новые образы. Самые популярные образы как правило позволяют запустить в контейнере основные базы данных, кэши, веб-серверы и прокси-серверы, а также эмулировать популярные операционные системы, такие как Ubuntu или CentOS. Это прекрасный способ получить в свое распоряжение закрытое виртуальное пространства контейнера для любых экспериментов с использованием Linux, не волнуясь при этом за сохранность своей главной операционной системы.

Как и для репозиториев с исходным кодом на GitHub, образам на Docker Hub можно добавлять “звезды”, чтобы их отметить или выделить, а также увидеть сколько раз открытые для всех образы были скачаны (pull). Набравшие самое большое количество звезд и использований образы находятся в начале списка открытых образов, который вы можете увидеть на странице hub.docker.com - именно там вы увидите, что сейчас в мире наиболее популярно для запуска контейнеров, или в качестве базового образа для запуска своего приложения.

## Интерактивные контейнеры - запуск и управление на примере образа Ubuntu

Если мы зайдем на открытое хранилище образов Docker Hub и посмотрим самые популярные образы (страница Explore), мы сразу же увидим там образ операционной системы Ubuntu - один из самых используемых. Интересно, что это не отдельное приложение, не веб-сервер и не прокси-сервер и совершенно точно не база данных, то есть это не что-то, что обычно требуется для разработки приложений или сервисов. Какой же смысл запускать операционную систему из контейнера?

В основном этот образ используется для экспериментов - вы можете запустить свое приложение из окружения Ubuntu со всеми ее стандартными зависимостями, пакетами и библиотеками, а также интерпретатором команд shell и всеми удобными инструментами Linux просто для того, чтобы иметь возможность отладить и отследить его работу в комфортном окружении, или же для того чтобы просто проверить его на работу в Linux, если у вас к примеру лэптоп с Mac OS. Отсюда еще один популярный вариант использования образа Ubuntu - возможность работать с легким контейнером внутри Linux, который запускается в течение секунд, вместо тяжелой виртуальной машины с гипервизором, на системах где основной операционной системой не является Linux.

И повторим еще раз, основное, что стоит помнить при запуске контейнеров, особенно таких как Ubuntu, Debian или CentOS - это не совсем операционная система! Работать ваш контейнер будет с общим ядром Linux, или с основной операционной системой, если вы уже работаете на Linux, или с минимальной виртуальной машиной Linux. Просто он будет ограничен своей “песочницей” со своей закрытой файловой системой, ограниченным пространством процессов и пользователей, и, возможно, ресурсов. Все что вы увидите внутри такого контейнера - это набор команд, файлов, символических ссылок, характерных для Ubuntu или другой операционной системы, но ниже этого уровня будет единое ядро. Другими словами, если вы запустите контейнеры Ubuntu, CentOS и Debian одновременно, все они будут работать с ядром и под управлением основной операционной системы Linux, и таким образом это просто удобный инструмент для эмуляции их инструментов и окружения.

Зная это, приступим наконец к запуску контейнера. Все команды Docker выполняются инструментом командной строки docker, а для запуска контейнера нужно просто выполнить docker run <метка образа>.

docker run ubuntu

Эта команда запустит контейнер на основе самой последней метки образа Ubuntu (latest). Если это первый запуск, она к тому же загрузит файлы этого образа из хранилища Docker Hub.

Интересно, что после запуска контейнер практически мгновенно завершит свою работу. Это его базовое свойство - если внутри контейнера не исполняется работающее приложение, он мгновенно завершается - контейнер должен быть эффективным и сразу освобождать свои ресурсы как только выполнение программы внутри него заканчивается. При запуске же Ubuntu ничего кроме интерпретатора bash по умолчанию не исполняется.

Здесь нам пригодится интерактивный режим запуска (ключ -i) - он присоединяет к контейнеру консоль для ввода и вывода и пока они активны, контейнер будет продолжать работать даже если в нем ничего не исполняется. Для эмуляции стандартного терминала пригодится также ключ -t.

docker run -it ubuntu

На этот раз Docker запустит контейнер на основе образа Ubuntu, и предоставит нам консоль с эмуляцией терминала - вы сразу увидите привычное приветствие интерпретатора bash и можете работать с ним так, как если бы это была реальная операционная система Ubuntu. Пока интерактивный режим активен, контейнер будет работать. Завершить работу контейнера из терминала можно набрав команду exit, или использовать сочетание Ctrl-P-Q - отсоединение (detach) интерактивного режима от работающего контейнера без его остановки.

Если вы хотите изначально оставить контейнер с Ubuntu запущенным, или он понадобится вам для продолжения экспериментов чуть позже, вы можете запустить его в отсоединенном (detach) режим, используя ключ -d:

docker run -d -it ubuntu

В этом случае терминал сразу не открывается, а контейнер запускается и продолжает работать в фоновом режиме уже без вашего участия, даже без работающих в нем процессов. Увидеть работающие в данный момент под управлением Docker контейнеры можно командой ps, заимствующей свое имя из той же Linux:

docker ps  
…  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
8abb1b4a6886 ubuntu "/bin/bash" Less than a second ago Up 1 second determined\_lichterman

Вы видите примерный вывод команды ps, довольно полезный - указан образ, на основе которого был создан работающий контейнер, время его запуска и работы, а также два уникальных идентификатора - первый просто уникальный UUID, а второй некое сгенерированное забавное “имя”, которое чуть легче напечатать человеку.

Конечно, нет ничего проще подключить терминал к работающему контейнеру, если он уже работает в фоновом режиме - для этого предназначена команда attach. Ей необходимо передать уникальный идентификатор контейнера, к которому нужно присоединиться (один из двух):

docker attach determined\_lichterman

Или

docker attach 8abb1b4a6886

Запустив эту команду, вы вновь окажетесь в терминале своего контейнера и сможете делать там любые эксперименты. Как и в первом случае, когда мы сразу же открыли терминал при запуске контейнера, команда exit закончит его выполнение, а клавиши Ctrl-P-Q просто отключат терминал, оставляя сам контейнер работающим.

Когда работающий в фоновом режиме контейнер вам станет больше не нужен, вы можете остановить его командой stop, как нетрудно догадаться, вновь указав его уникальный идентификатор:

docker stop 8abb1b4a6886

Или

docker stop determined\_lichterman

Снова вызвав команду ps, вы увидите что контейнер был остановлен.

Можем себя поздравить - зная название популярных образов на Docker Hub и несколько только что опробованных команд, мы можем экспериментировать с эмуляцией популярных операционных систем Linux, базами данных и серверами, запуская их в контейнерах за считанные секунды, и экспериментируя с ними любым образом, не опасаясь мусора в своей основной операционной системе или проблем с другими приложениями - контейнеры Docker надежно изолируют все, что происходит внутри них.

## Открытие мира для контейнера - веб-сервер nginx и работа с портами

Итак, проводить эксперименты в контейнерах с различными версиями операционных систем Linux, разнообразными системами, работающими на них, и получать все это в пределах секунд, великолепно. Однако основная идея контейнера - изоляция именно вашего приложения или дополнительных сервисов, необходимых для его работы, чтобы эффективно разделить ресурсы, избежать сложностей с развертыванием, и мгновенно проводить горизонтальное масштабирование.

Вновь взглянув на самые популярные образы на сайте Docker Hub, мы увидим там знаменитый веб-сервер nginx. Он часто используется для поддержки уже готовых приложений в качестве просто веб-сервера или балансировщика нагрузки. Давайте запустим последнюю версию его образа в контейнере:

docker run -d nginx

Запускать его лучше сразу в отсоединенном режиме (detach, -d), так как при запуске этого образа, в отличие от образа Ubuntu, процесс nginx сразу запускается и работает пока мы его не остановим, и запуск образа без ключа -d приведет к тому что мы будем наблюдать консоль с выводом сообщений nginx и выйти из нее можно только вместе с самим nginx с помощью Ctrl-С.

Запустив команду docker ps, вы увидите что контейнер с работающим сервером nginx существует и увидите его уникальное имя. Что же дальше? Что если просто хотим использовать его как свой веб-сервер, только работающий из контейнера, чтобы не запускать его на основной операционной системе и иметь всю гибкость и безопасность контейнера? Достоверно известно, что любой веб-сервер обслуживает порт HTTP 80, можно ли получить к нему доступ, если он находится в контейнере?

Открытые (exposed) порты - основной путь взаимодействия между контейнерами, которые, как мы знаем, в общем случае максимальным образом изолированы и от основной операционной системы, и друг от друга. Именно через открытые порты можно получить доступ к работающим в контейнерах приложениям (не присоединяя к ним терминалы), и обеспечить взаимодействия множества контейнеров между друг другом. Многие образы, подобные nginx, автоматически указывают, какие порты будут активны в контейнере - в nginx это конечно будет порт HTTP 80.

Указать взаимодействие между портами основной операционной системы (системы, ресурсами которой пользуется Docker, то есть хоста), и контейнера, можно все той же командой run. Давайте остановим уже запущенный ранее контейнер (легкое упражнение на повторение предыдущего раздела), и запустим nginx снова, на этот раз указав на какой порт главной операционной системы мы переадресуем (forward) открытый в контейнере порт 80 (возьмем порт 8888):

docker run -d -p 8888:80 nginx

После запуска такого контейнера откройте в браузере адрес localhost:8888 или 0.0.0.0:8888 и вы увидите приветствие сервера nginx. Переадресовать открытый порт контейнера на порт основной операционной системы-хоста в общем случае можно так:

docker run -p [порт основной хост-системы]:[открытый порт контейнера] [имя образа]

Если вновь посмотреть список активных контейнеров командой ps, мы обнаружим что она теперь сообщает нам информацию об открытых портах и их переадресации:

docker ps  
…  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
5e0c7f30656d nginx "nginx -g 'daemon of…" 5 minutes ago Up 5 minutes 0.0.0.0:8888->80/tcp loving\_rosalind

Когда вы вполне насладитесь возможностями сервера nginx, не забудьте остановить работающий контейнер командой docker stop [имя].

Это основное предназначение и главное достижение контейнеров - вы можете запустить столько разнородных систем в отдельных контейнерах, сколько вам нужно, и на стольких серверах, как подсказывает вам здравый смысл, ваша архитектура и текущая нагрузка на приложение. Все они будут надежно изолированы друг от друга, все будут использовать свои собственные зависимости, библиотеки и пакеты, или виртуальные машины и интерпретаторы, и взаимодействовать будут через заранее оговоренные открытые порты.

## Резюме

Вычислительные ресурсы дороги, и зачастую запуск множества приложений на одном сервере грозит множеством конфликтов библиотек, зависимостей, сетевых адресов и портов, а также возможными уязвимостями безопасности. Мы кратко изучили эволюцию от виртуальных машин к контейнерам, и можем сделать следующие краткие выводы:

* Контейнеры - это изоляция приложения, основанная на возможностях операционной системы Linux (иногда Windows Server), позволяющих отделить процессы друг от друга, выделить им отдельную виртуальную файловую систему и ресурсы.
* Контейнеры - это не виртуальные машины. Они работают напрямую с ядром системы Linux, на которой запущены, и не имеют внутри себя никакой собственной операционной системы или гипервизора. Благодаря этому их запуск по скорости и удобству мало чем отличается от запуска приложения напрямую на основной операционной системе.
* Инструмент Docker организует эффективный запуск, остановку и управление контейнерами Linux, и позволяет работать с ними на популярных операционных системах посредством минимальных виртуальных машин.
* Зависимости и файлы, необходимые для работы контейнера, упаковываются в образы (image), которые как правило хранятся в общедоступном репозитории Docker Hub, имеющем управление и модель команд, схожую с репозитории исходного кода GitHub.
* Популярные образы, такие как Ubuntu или Nginx, часто используются для тестирования и экспериментов без запуска тяжелых виртуальных машин. Доступ к ним организован через открытые порты, переадресуемые на порты основной операционной системы (хоста), где работает Docker.