

Использование Kubernetes для развертывания приложений

В этой главе

- ✓ Преимущества упаковки приложений в контейнеры.
- ✓ Что делает Kubernetes идеальной платформой для развертывания контейнеров.
- ✓ Когда стоит использовать Kubernetes, а когда — нет.

Наступил вечер пятницы, и у пользователей вдруг проснулся необычайный интерес к тому приложению, над которым вы работали в течение последнего года. Вам нужно его масштабировать, и как можно скорее. Позволит ли ваше приложение и та платформа, на которой оно запущено, увеличить масштаб в 100 раз, обеспечив вам успех, или все ваши усилия окажутся напрасными из-за нестабильно работающего кода и негибкой платформы?

Вы создали отличное stateless-приложение на базе популярной платформы для создания приложений, позволяющей быстро проводить масштабирование, и все прекрасно работает... Но вот наступает момент, когда бизнес-требования изменяются, в связи с чем необходимо запустить специализированное приложение с сохранением состояния (stateful-приложение) для некоторых критически важных бизнес-данных или специально настроенный конвейер ночной пакетной обработки. Сможете ли вы без проблем добавить эти новые рабочие нагрузки к уже

существующим? Или потребуется разработать новое приложение или как-то объединить несколько разнородных систем?

Предлагая разработчикам решение этих и многих других проблем, Kubernetes быстро становится отраслевым стандартом, позволяющим выполнять рабочие нагрузки самых разных форм и размеров в любом необходимом масштабе. Эта платформа позволяет запускать контейнеризированные приложения, предоставляя возможность быстрого их масштабирования, используя при этом несколько сложных схем развертывания: от развертывания приложений без сохранения состояния до развертывания баз данных с сохранением состояния и пакетных заданий с эфемерным хранилищем. Это свободно распространяемое детище компании Google (<https://cloud.google.com/learn/what-is-kubernetes>), которым уже пользуются такие уважаемые организации, как Spotify (<https://kubernetes.io/case-studies/spotify/>), CapitalOne (<https://kubernetes.io/case-studies/capital-one/>) и OpenAI (<https://kubernetes.io/case-studies/openai/>), представляет собой открытую, не привязанную к конкретному поставщику надежную платформу, которая играет в мире облачного развертывания столь же важную роль, как, например, Linux в мире операционных систем.

Однако платой за такую гибкость является достаточно длительный срок освоения возможностей платформы. Будучи универсальным инструментом, Kubernetes позволяет использовать так много разных способов развертывания, что изучение платформы может показаться крайне сложной задачей. Здесь я должен сказать, что, во-первых, изучить ее не так сложно, как может показаться, и, во-вторых, ваши старания окупятся с лихвой. Если вы начнете с основ и будете постепенно добавлять новые конструкции (а именно такой подход описывается в книге), то освоение этой системы уже не будет выглядеть пугающе сложной задачей. Вы можете развернуть stateless-приложение, используя лишь несколько строк YAML-кода, и продолжить дальнейшее изучение, отталкиваясь от этой отправной точки.

Исходя из своего профессионального опыта, могу сказать, что при решении такой проблемы, как выбор наилучшего способа развертывания приложения, нужно иметь в виду следующий момент. Часто вместо того чтобы выбрать самый простой вариант, способный удовлетворить текущие требования, стоит потратить некоторое время на изучение системы, что сможет удовлетворить ваши потребности и сейчас, и в дальнейшем, а также позволит вам развивать свои навыки и расти профессионально по мере увеличения требований к приложению. Именно такой системой является Kubernetes. Вы можете за считанные часы реализовать одну из простейших схем развертывания, зная при этом, что данный инструмент таит в себе множество других функциональных возможностей, которые вы всегда можете освоить и применить в случае необходимости.

Если я уже убедил вас в необходимости использования Kubernetes, то вы можете пропустить оставшуюся часть этой главы и сразу перейти к главе 2, в которой рассказывается о создании Docker-образа, или к главе 3, если уже знаете, что такое Docker-контейнер, и хотите приступить к развертыванию приложений в Kubernetes. Далее в этой главе рассказывается о том, почему разработчики приложений все чаще предпочитают использовать Kubernetes и контейнеры.

1.1. ЗАЧЕМ ИСПОЛЬЗОВАТЬ КОНТЕЙНЕРЫ

Kubernetes — платформа развертывания для контейнеров. При развертывании на ней любого кода, например приложения, он сначала упаковывается в контейнер. Но что такое контейнеры и зачем вообще они нужны?

Контейнеры — современное средство упаковки и запуска приложений. Если вы не собираетесь запускать каждое приложение на отдельном хосте (что крайне неэффективно), то вам как-то нужно будет развернуть несколько приложений на одной машине или группе машин. Рассмотрим доступные варианты.

До появления виртуальных машин было принято устанавливать приложения в отдельные каталоги на общем хосте, выделяя для каждого из них отдельный порт. Этот подход чреват проблемами, поскольку приложения могут влиять друг на друга при совместном использовании зависимостей и таких ресурсов машины, как процессор, память и доступные порты. Кроме того, данная схема трудно поддается масштабированию: если одно из приложений вдруг начнет получать больше трафика, чем другие, то вы не сможете масштабировать только это приложение, оставив другие на прежнем уровне.

После появления виртуальных машин разработчики стали упаковывать каждое приложение в отдельную такую машину. При этом каждое приложение имеет собственную операционную среду с изоляцией зависимостей и выделенной для него частью ресурсов. Но каждая виртуальная машина эмулирует все функции отдельного хоста, поэтому вам приходится обеспечивать управление операционной системой и всеми пакетами для каждого приложения, что достаточно сложно и чревато большими накладными расходами.

Теперь перейдем к контейнерам. Подобно виртуальной машине, контейнер позволяет упаковать приложение и зависимости, необходимые ему для размещения в изолированной среде, но при этом не требует, чтобы вместе с приложением устанавливалась отдельная операционная система.

На рис. 1.1 показан процесс эволюции сервисов хостинга: от запуска нескольких рабочих нагрузок на общем хосте до запуска их в отдельных виртуальных машинах

и, наконец, в контейнерах. Благодаря использованию контейнеров можно получить многие из преимуществ виртуальной машины, обойдясь без накладных расходов на запуск дополнительного ядра операционной системы, что делает переход к их применению логичным следующим шагом.

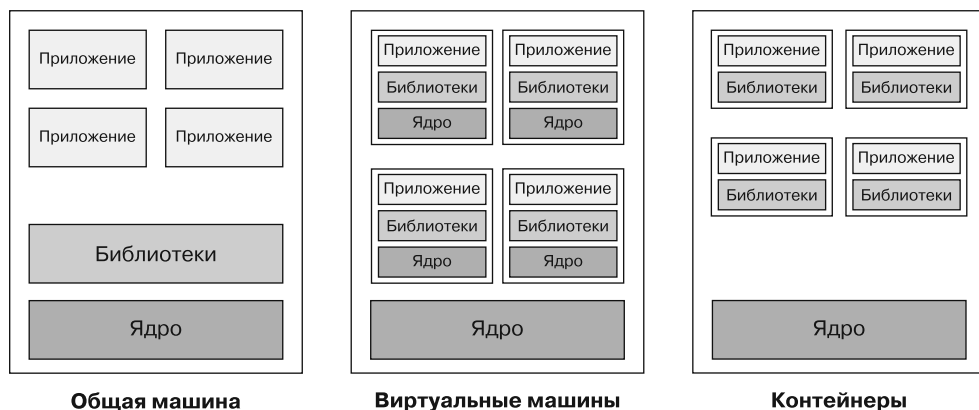


Рис. 1.1. Эволюция архитектур общего хостинга

1.1.1. Преимущества контейнеров

Основные доводы в пользу применения контейнеров:

- *языковая гибкость* — на контейнерной платформе можно использовать любой язык или среду;
- *легковесная изоляция* — можно исключить влияние рабочих нагрузок друг на друга, не используя виртуальные машины;
- *высокая эффективность процесса разработки* — степень различий между продуктивной средой и средой разработки уменьшается, а также упрощается настройка приложения;
- *воспроизводимость* — в сборочном файле контейнера описывается последовательность действий по созданию необходимой среды.

Языковая гибкость

Контейнеры исключают зависимость системы развертывания от конкретного языка или библиотек. Вы можете использовать любой язык и обновлять любой пакет. Вы больше не будете привязаны к конкретному языку или версии языка либо к устаревшей версии критически важной зависимости, установленной вместе с операционной системой много лет назад, как часто происходило при использовании традиционной PaaS-среды (platform as a service, платформа как услуга).

При запуске двух контейнеров на одном хосте они не будут иметь общих библиотек, а значит, конфигурация одного не будет влиять на конфигурацию другого. Вам нужно использовать разные версии языка Java или совершенно разные зависимости? Нет проблем! Изоляция не ограничивается одними библиотеками: разные контейнеры могут применять совершенно разные операционные системы и менеджеры пакетов: например, один пакет может использовать Ubuntu в сочетании с менеджером APT, а другой — CentOS в сочетании с менеджером RPM. Такая гибкость позволяет при необходимости легко объединить в одну систему несколько сервисов (используя так называемую *микросервисную* архитектуру), каждый из которых будет иметь свой набор зависимостей и используемых языков и будет поддерживаться отдельной командой разработчиков. Контейнеры изолируют друг от друга зависимости разных приложений, что упрощает их запуск на одном хосте (рис. 1.2).

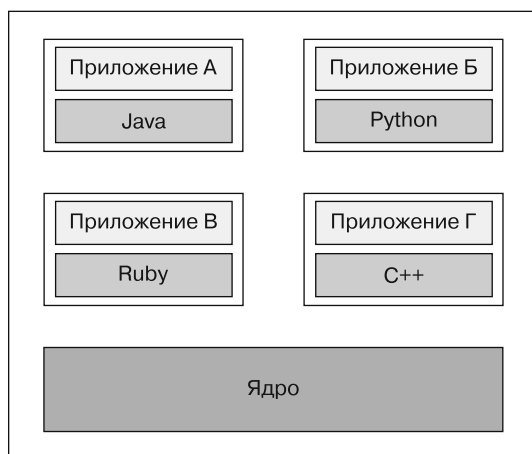


Рис. 1.2. На одном хосте запущены четыре контейнера, в которых используются разные языки

Изоляция без накладных расходов

Раньше для взаимной изоляции приложений, запускаемых на одном и том же хосте, использовались виртуальные машины. Это решение влияет и на размер образа, и на потребление ресурсов процессора и памяти, поскольку в каждой виртуальной машине приходится дублировать ядро и значительную часть операционной системы. Контейнеры являются более подходящим решением, обеспечивая практически те же преимущества изоляции ресурсов. Любой контейнер можно настроить на использование только части ресурсов хоста, и Kubernetes проследит за соблюдением данного ограничения. В конечном счете это позволяет размещать на одном хосте больше приложений, снижая уровень затрат на инфраструктуру (рис. 1.3).

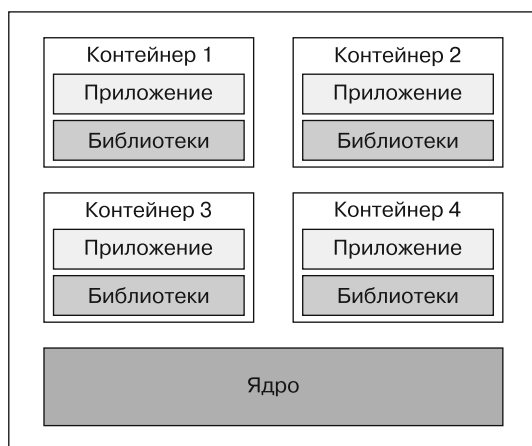


Рис. 1.3. Четыре контейнера, запущенных на одном хосте, полностью изолированы друг от друга, но используют общее ядро

Высокая эффективность процесса разработки

Изоляция зависимостей, возможная благодаря контейнерам, упрощает не только процесс развертывания в продуктивной среде, но и процесс разработки, поскольку вы можете разрабатывать на одной машине множество различных приложений, и при этом вам не придется учитывать в конфигурации хоста зависимости каждого из них (рис. 1.4). Наряду с разработкой Linux-приложений непосредственно в Linux с помощью Docker вы можете создавать контейнеры для Linux на рабочей станции с операционной системой macOS или Windows, не создавая нативную версию приложения для этих платформ, когда на этапе разработки приходится использовать соответствующую платформенно-зависимую конфигурацию.

Разработчикам больше не нужно использовать многостраничные инструкции по начальной настройке, поскольку теперь этот процесс сводится лишь к тому, чтобы установить Docker, проверить код, выполнить сборку и запуск. Упрощается и совместная работа над несколькими проектами в составе одной или нескольких команд разработчиков: каждый проект аккуратно изолируется в своем контейнере, и вам не нужно использовать для него специальную конфигурацию хоста.

Теперь приложение выглядит практически одинаково и в среде разработки, и в продуктивной среде, зачастую представляя собой один тот же контейнер. Благодаря этому можно забыть об особенностях среды разработки, таких как использование конкретной версии библиотеки MySQL в macOS или незначительные различия в способе упаковки кода для развертывания в продуктивной среде. Вам нужно выяснить, чем вызвана определенная проблема в продуктивной среде? Скачайте тот же контейнер и посмотрите, что с ним не так, запустив его в своей среде разработки (рис. 1.5).

Машина разработчика, использующего Docker

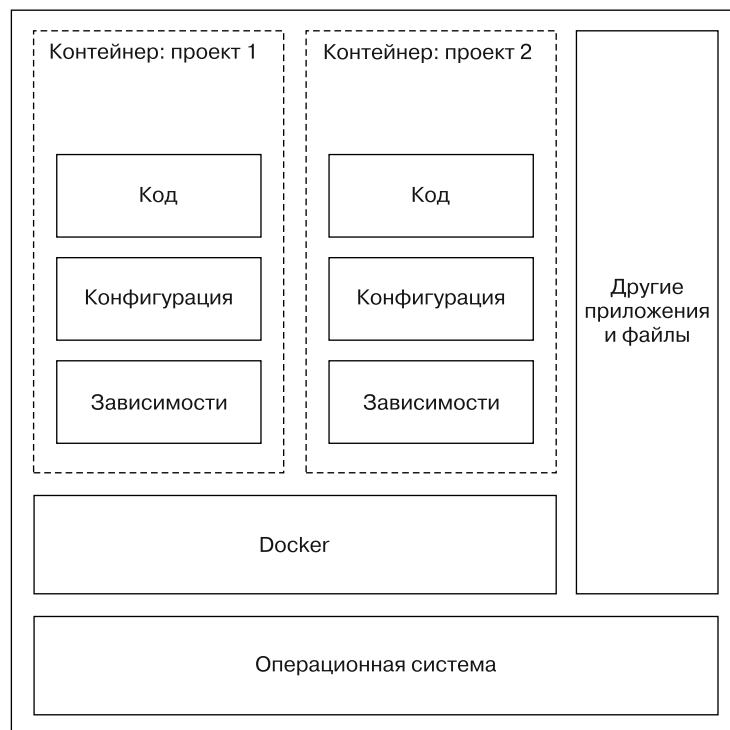


Рис. 1.4. Машина разработчика с двумя контейнеризированными проектами



Рис. 1.5. Развертывание одного и того же контейнера в продуктивной среде и среде разработки

Воспроизводимость

При использовании контейнеров проще воспроизвести необходимую для приложения среду. Допустим, вы развернули приложение на виртуальной машине, и вам нужно настроить протокол TLS (Transport Layer Security, протокол защиты транспортного уровня), чтобы обезопасить HTTP-соединения. Вы устанавливаете SSH-соединение с рабочим хостом и добавляете в одну из папок TLS-сертификат. Это не дает результата, и вы добавляете сертификат в другую папку. Результата снова нет, и вы добавляете сертификат в третью папку. Он срабатывает, и вы больше не вносите никаких изменений. Год спустя наступает момент, когда требуется обновить этот TLS-сертификат. Сможете ли вы теперь вспомнить, каким образом и в какой из трех папок нужно выполнить обновление?

Использование контейнеров позволяет решить эту проблему. Вместо того чтобы устанавливать SSH-соединение и вносить изменения в состояние рабочего хоста, вы можете выполнить добавление TLS-сертификата как один из этапов сборки контейнера. Если данный этап не дает результата, то вы можете скорректировать его так, чтобы он заработал, но, что важно, в конечном счете сохраните только ту его версию, которая действительно дает результат. Кроме того, те файлы, которые вы добавляете на данном этапе сборки, аккуратно изолируются от остальной части системы и поэтому могут представлять собой только приращение или отличия от исходного состояния системы — только те изменения, которые вам необходимо внести. Это означает следующее: когда подойдет время обновления сертификата, вам нужно будет просто заменить старый файл сертификата новым и заново выполнить процесс сборки контейнера, который поместит сертификат в нужное место.

В листинге 1.1 представлен пример псевдокода, описывающего содержимое файла Dockerfile, то есть код настройки контейнера, представленный на обычном языке (примеры реального содержимого этого файла будут рассмотрены в главе 2).

Листинг 1.1. Chapter01/1.1.1_Pseudocode Dockerfile

Использовать операционную систему Ubuntu
Скопировать и настроить TLS-сертификат
Скопировать приложение

ПРИМЕЧАНИЕ

Используя Docker в качестве инструмента для создания контейнеров, вы не сможете обеспечить идеальную воспроизводимость. При установке зависимостей с помощью таких команд, как `apt-get`, Docker взаимодействует с операционной системой в ее текущем виде и поэтому иногда выдает разный результат для одинаковых входных данных, поскольку такие зависимые системы, как репозиторий АРТ, могут изменяться от сборки к сборке. Некоторые инструменты, например, свободно распространяемая система сборки Bazel от компании Google, позволяют решить эту и другие проблемы, но имеют свои сложности и больше подходят для больших корпоративных схем развертывания. Несмотря на это ограничение, система сборки Docker все же обеспечивает намного больший уровень воспроизводимости по сравнению с необходимостью вспоминать, что именно вы делали в SSH-сеансе год назад, когда нужно было устранить проблему на определенной Linux-машине. В этом плане данная система вполне приемлема для большинства проектов.