Контейнер Docker

Docker — это программная платформа для разработки, доставки и запуска контейнерных приложений. Он позволяет создавать контейнеры, автоматизировать их запуск и развертывание, управляет жизненным циклом. С помощью Docker можно запускать множество контейнеров на одной хост-машине.

Что такое контейнеры

Контейнеризация — это способ упаковки приложения и всех его зависимостей в один образ, который запускается в изолированной среде, не влияющей на основную операционную систему. С помощью контейнера можно отделить приложение от инфраструктуры: не важно, в каком окружении оно будет работать, есть ли там нужные зависимости и настройки, — разработчикам достаточно создать программу и упаковать все зависимости и настройки в единый образ. Затем ее можно разворачивать и запускать на других системах. Это ускоряет процесс разработки, сокращает промежуток между написанием кода и его выкладкой.

Контейнеризация напоминает виртуализацию, но технологии отличаются друг от друга. Виртуализация работает как отдельный компьютер со своей операционной системой и виртуальным оборудованием. Внутри одной операционной системы можно запустить другую. При контейнеризации виртуальная среда может запускаться прямо из ядра основной ОС и не виртуализирует оборудование. Поэтому контейнер может работать только в той же операционной системе, что и основная. Так как контейнеры не виртуализируют оборудование, они потребляют меньше ресурсов.

Как устроен Docker

Docker — клиент-серверное приложение. Это означает, что оно состоит из двух частей: сервера и клиента.

Сервер еще называют Docker-движком или демоном (daemon). Это фоновый процесс, который непосредственно управляет контейнерами. Именно демон создает, разворачивает и запускает контейнеры. Его можно сравнить с двигателем машины.

Клиент — это программа-интерфейс для командной строки, с которой взаимодействует пользователь. Он отдает команды через терминал. Клиент сообщает нужные сведения демону и отдает ему указания. Если продолжать аналогию с машиной, клиент — это руль и педали.

Клиент и сервер могут находиться на одном устройстве, а могут — на разных. Во втором случае клиент подключают к удаленному серверу через сокеты или API.

Компоненты для контейнеризации

Docker работает со следующими компонентами:

Dockerfile. Это файл для предварительной работы, набор инструкций, который нужен для записи образа. В нем описывается, что должно находиться в образе, какие команды, зависимости и процессы он будет содержать.

Docker Image. Это образ — неизменяемый файл, из которого разворачивается контейнер.

Docker Registry. Это реестр, или репозиторий — открытая или закрытая база образов.

При запуске команды docker run программа сначала проверяет, есть ли нужный образ в локальном хранилище. Если его нет, она сама находит файл в репозитории и скачивает на компьютер.

Docker Container. Это уже готовый и развернутый контейнер, который находится на каком-либо устройстве.

Так пользователь запускает нужный образ через клиент Docker и ждет, пока платформа развернет его в полноценную среду или приложение-контейнер.

Docker контейнер

Docker контейнер — это стандартизированный, изолированный и портативный пакет программного обеспечения, который включает в себя все необходимое для запуска приложения, включая код, среду выполнения, системные инструменты, библиотеки и настройки. Контейнеры позволяют упаковать приложение и все его зависимости в единый объект, который может быть запущен на любой системе, поддерживающей Docker, без изменения среды выполнения.

Как хранятся данные в Docker

При остановке и перезапуске контейнера можно потерять часть рабочей информации, которая в нем записана. Чтобы этого избежать, программисты стремятся разрабатывать приложения с минимальным использованием хранилищ внутри контейнеров. Но обойтись без хранения данных получается не всегда, а от основной системы контейнер изолирован. Существует несколько способов решить проблему.

Docker Volumes. Это тома — способ хранения информации, который рекомендуют использовать разработчики платформы. Внутри лежат файлы и другие данные. Тома можно подключать к разным контейнерам, выбирать специальные драйверы и хранить информацию не на хосте, а в облаке или на удаленном сервере.

В Linux тома по умолчанию находятся в /var/lib/docker/volumes/. Другие программы могут получить к ним доступ только через контейнер, а не напрямую. Для создания и управления томами используются средства Docker: команда docker volume create, указание тома при создании контейнера в Dockerfile или docker-compose.yml.

Bind Mount. Более простой способ реализовать удаленное хранение памяти — папки, которые монтируются в контейнер прямо с хоста. Этот вариант используют для передачи конфигурационных файлов или в процессе разработки. Программист может писать код в среде хоста, а потом передавать его в контейнер.

Tmpfs и Named Pipes. Так называется особое файловое хранилище, которое есть только в системах Linux. Как правило, оно используется не для хранения файлов, а для обеспечения безопасности. Tmpfs — временное хранилище. Стоит остановить контейнер — данные будут потеряны. Доступ к Tmpfs очень быстрый, поэтому хранилище используют, чтобы оптимизировать работу контейнера.

Named Pipes — это именованные каналы. Через них с Docker могут работать только пользователи Windows.

Задачи, которые решает Docker

1. **Развертывание среды или приложения**

Docker позволяет перенести приложение со всеми зависимостями на другую систему с помощью пары команд в терминале. Настройка зависимостей вручную занимает больше времени. Также с помощью Docker можно быстро развернуть рабочую среду с определенными настройками. Существуют «системные контейнеры», которые содержат дистрибутивы ОС.

2. **Изолированный запуск**

Docker позволяет запустить приложение отдельно от всей системы без конфликтов с другими программами. Программа становится практически автономной и не вызывает ошибок зависимости.

3. **Контроль ресурсов**

Еще одна возможность Docker — распределение ресурсов между разными приложениями. Неизолированные процессы могут конкурировать за память и вычислительные мощности процессора. Изолированные друг от друга программы не делают этого. Docker позволяет эффективнее использовать ресурсы и не допускать конфликтов.

4. **Повышение безопасности**

Если код контейнерного приложения окажется небезопасным, это не навредит серверу-хосту. При правильной настройке контейнера деятельность кода не затронет основную систему. Даже фатальная ошибка не повлияет на работоспособность остальных служб и программ.

5. **Работа с микросервисами**

Микросервисная архитектура — такой тип организации ПО, при котором функции большого приложения разделяются на маленькие независимые программные модули. Они общаются друг с другом с помощью протоколов, но в целом работают автономно друг от друга. Docker подходит для реализации архитектуры этого типа: каждый микросервис упаковывается в отдельный контейнер, который можно настроить и протестировать, запустить или остановить отдельно от других.

6. **Ускорение цикла разработки**

Технологии контейнеризации помогают программировать быстрее. На настройку среды, разворачивание приложений под разными платформами тратится меньше времени. В результате повышается производительность всей команды.

7. **Управление сложными системами**

Для автоматизации большинства процессов со сложными контейнерными приложениями используются платформы оркестрации. Многие возможности специального ПО завязаны на контейнеризации и функциях Docker.

Например, платформы автоматизируют разворачивание контейнеров, их настройку и масштабирование. Это нужно, так как программная архитектура становится более сложной. Приложения могут состоять из сотен отдельных контейнеров, каждый из которых нужно развернуть и настроить. Поддержка таких приложений вручную занимает много времени.

8. **Масштабирование**

Это еще одна задача для платформ оркестрации. Во многих из них поддерживается автоматическое масштабирование систем под разные

площадки и условия. Пример такой платформы — Kubernetes, которая часто используется в связке с Docker.

Автомасштабирование помогает быстро оптимизировать сектор под повышенную нагрузку. Если сайт неожиданно получит больше трафика, чем обычно, система перераспределит ресурсы и сервисы адаптируются.