Доклад по Docker

Введение.

Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации, контейнеризатор приложений. Позволяет «упаковать» приложение со всем своим окружением и зависимостями в контейнер, который может быть развёрнут на любой Linux-системе с поддержкой контрольных групп в ядре, а также предоставляет набор команд для управления этими контейнерами. Изначально использовал возможности LXC, с 2015 года начал использовать собственную библиотеку, абстрагирующую виртуализационные возможности ядра Linux — **libcontainer**. С появлением Open Container Initiative начался переход от монолитной к модульной архитектуре.

Основные концепции

1. Контейнеры

Контейнеры – это легковесные, изолированные процессы, которые используют общую операционную систему. Они быстро загружаются и работают более эффективно, чем виртуальные машины.

2. Образы

Образ – это шаблон для создания Docker-контейнеров. Представляет собой исполняемый пакет, содержащий все необходимое для запуска приложения: код, среду выполнения, библиотеки, переменные окружения и файлы конфигурации. Docker-образ состоит из слоев. Каждое изменение записывается в новый слой.

3. Dockerfile

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. При запуске команды docker run программа сначала проверяет, есть ли нужный образ в локальном хранилище.

4. Docker Hub

Docker Hub - это размещенный реестр Docker, управляемый Docker. Docker Hub содержит более 100 000 образов контейнеров от поставщиков программного обеспечения, а также проекты с открытым исходным кодом и сообщества.

Преимущества Docker

К преимуществам Docker для разработки и развертывания программного обеспечения можно отнести масштабируемость, согласованность, переносимость, изолированность и эффективность использования ресурсов. Благодаря тому, что Docker изолирует зависимости, каждый контейнер может одинаково надежно работать в любой среде. Все это ценится сотрудниками самых разных групп в организации, например отделов разработки, эксплуатации и контроля качества.

1. Масштабируемость

Быстрый запуск контейнеров Docker позволяет без усилий развертывать приложения по запросу. При такой динамичности можно масштабировать приложения в зависимости от колебаний трафика или загруженности.

Например, во время распродаж «черной пятницы» приложение интернет-магазина может не выдержать наплыва покупателей. Чтобы справиться с такими пиками, можно прибегнуть к автоматическому масштабированию контейнеров Docker, в которых работают микрослужбы веб-сайта. С этим справится такой инструмент оркестрации, как Kubernetes. Его лишь нужно настроить для корректировки количества запущенных контейнеров в зависимости от спроса.

Инструмент оркестрации контейнеров предоставляет платформу для автоматического управления жизненным циклом контейнеров и архитектурой микрослужб в нужном масштабе. Он автоматизирует операционные усилия при эксплуатации контейнерных рабочих нагрузок и служб: выделяет ресурсы, а также выполняет развертывание, подключение к сети, масштабирование, балансировку нагрузки и не только.
По мере увеличения числа покупателей и транзакций этот инструмент создает новые контейнеры, чтобы равномерно распределить рабочую нагрузку. Благодаря этому веб-сайт будет работать без перебоев при любых всплесках трафика.

2. Единообразие

Docker обеспечивает согласованность разработки и развертывания. Инженеры могут создавать и дублировать пакеты без привязки к среде, а у пользователей есть возможность проверить точную версию необходимых библиотек и пакетов в контейнере, сводя к минимуму риск возникновения багов из-за слегка отличающихся редакций зависимостей.

Без такой согласованности устранение багов и тестирование кода заставляли бы команды тратить много времени и ресурсов. Из-за ряда несоответствий поставка программного обеспечения стала бы ненадежной.

3. Переносимость

Docker — это легковесный портативный программный инструмент, который упаковывает все необходимое для стабильной работы приложения в любой среде.

Такая автономность позволяет не привязывать контейнеры к предустановленному программному обеспечению или определенным конфигурациям хоста. Контейнеры можно легко подготавливать и разворачивать, где угодно.

4. Изоляция

Контейнер Docker изолирует код в автономной среде, не зависящей ни от других контейнеров, ни от операционной системы хост-компьютера. Благодаря этому тестирование кода становится безопаснее, поскольку оно не влияет на работу остального приложения. Docker также устраняет проблемы совместимости и конфликты зависимостей, которые могут возникнуть при непосредственном запуске приложений в различных средах или системах, поскольку предоставляет единую согласованную платформу для запуска.

Недостатки Docker

Несмотря на всё своё удобство и эффективность в решении поставленных задач, у этой платформы есть и недостатки.

Во-первых, самой по себе ее хватает на то, чтобы запускать всего несколько контейнеров. Если же приложение включает десятки сервисов, понадобится оркестратор (Kubernetes или OpenShift). А это означает дополнительное усложнение инфраструктуры.

При этом Docker уже довольно требователен к ресурсам. Решая, использовать его или нет, следует задаться вопросом, достаточно ли у вас мощностей. Если да, то его можно смело ставить, обеспечив себе удобство созданий новых версий программы и не опасаясь навредить системной среде. Если же нет, то лучше работать по старинке.

Ещё стоит отметить, что изначально эта платформа разрабатывалась для Linux. Чтобы запустить ее на Windows или MacOS, придётся сначала поднять виртуальную машину с Linux. Это, разумеется, повышает расход ресурсов и налагает кое-какие другие ограничения (например, не поддерживаются

некоторые виды сетей). При установке на Windows также иногда возникают конфликты с Virtual Box (хотя и не на всех устройствах).

Заключение

Использование контейнеров стало стандартом в современном программировании, позволяя разработчикам фокусироваться на коде, а не на инфоструктуре.