Методы и средства сборки и развертывания ПО, 25

ФИО преподавателя: Смирнов Михаил Вячеславович

e-mail: smirnovmgupi@gmail.com

Лекция 11

Контейнеры и инструменты оркестрации в CI/CD. Serverless архитектура

Контейнеризация, как альтернатива гипервизору

Контейнер - легковесный, исполняемый пакет программного обеспечения, который включает в себя все необходимое для его запуска: код, среду выполнения, системные инструменты, системные библиотеки и настройки.

Ключевая концепция **-** изоляция на уровне операционной системы. В отличие от виртуальных машин, контейнеры используют ядро хостовой ОС, что делает их весьма быстрыми в запуске.

Dockerfile **-** это «рецепт» сборки образа. Он описывает пошаговые инструкции: какую базовую среду взять, какие файлы скопировать, какие команды выполнить для установки зависимостей.

Фундаментальные преимущества контейнеров

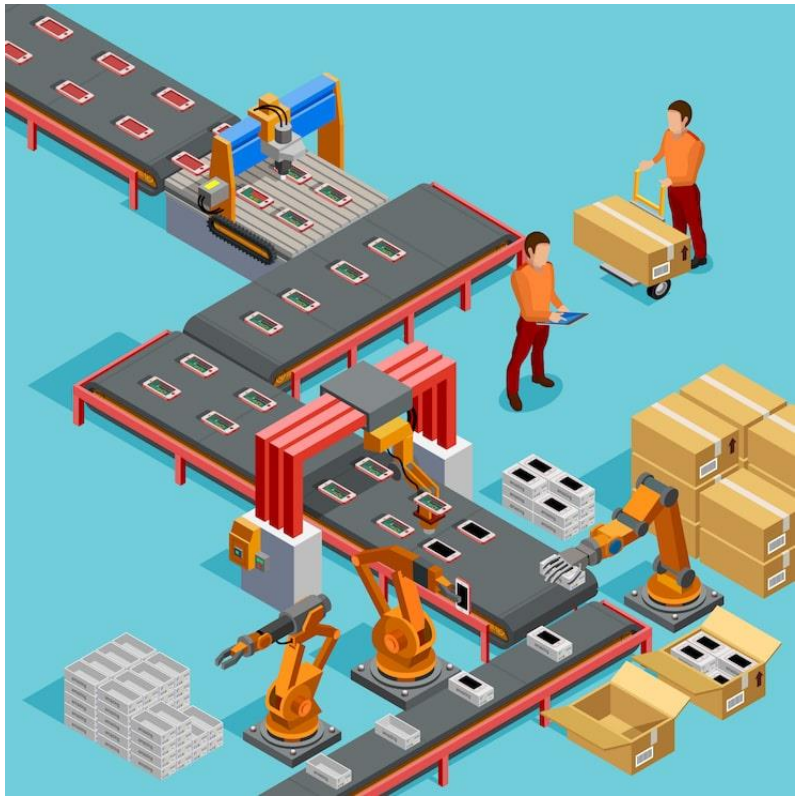
Переносимость. Так, например образ, собранный на macOS разработчика, без изменений запустится на Linux-сервере в дата-центре или Windows-машине инженера по тестированию.

Изоляция: Контейнеры обеспечивают изоляцию процессов, файловой системы и сети. Два контейнера на одном хосте могут использовать разные версии одной и той же библиотеки, не конфликтуя друг с другом.

Эффективность: Отсутствие гостевой ОС делает контейнеры существенно меньше и быстрее по сравнению с виртуальными машинами. Это позволяет «плотно» грузить их на одном сервере.

Контроль версий: Образы можно версионировать и хранить в реестре (Docker Registry). Это позволяет легко откатиться к предыдущей, рабочей версии приложения в любой момент.

«Конвейер контейнеров»



Сборка (Build). Этап, на котором исходный код компилируется и упаковывается в Docker-образ. Этот образ помечается уникальным тегом (обычно хэшем коммита). Образ собирается только один раз.

Тестирование (Test). Собранный образ запускается в контейнере, и внутри него выполняются все этапы тестирования: юнит-тесты, интеграционные тесты, проверки безопасности.

Развертывание (Deploy). Прошедший все проверки образ из реестра развертывается в тестовой и релизной средах. Поскольку среда внутри контейнера идентична среде тестирования, риск «поломки» при релизе минимален.

Проблема масштаба контейнеризации

Координация: Как обеспечить связь между контейнерами? Как управлять их конфигурациями?

Обнаружение сервисов: Как одному контейнеру найти другой, если их местоположение динамически меняется?

Масштабирование: Как вручную увеличить количество экземпляров приложения под нагрузкой и распределить между ними трафик?

Отказоустойчивость: Что произойдет, если упадет сервер? Как автоматически перезапустить контейнеры на другой здоровой машине?

Обновления: Как обновить приложение на всех контейнерах без простоев (zero-downtime deployment)?

Kubernetes (K8s) - система оркестрации

Принцип работы. YAML-конфигурация описывает текущие «хотелки» а Kubernetes сам решает как этого достичь.

Pod. Наименьшая единица в Kubernetes. Это «обертка» для одного или нескольких контейнеров, которые разделяют сетевое пространство и хранилище. Kubernetes не запускает контейнеры напрямую — он работает с подами.

Deployment. Декларативный способ описать желаемое состояние приложения (например, «должно быть запущено 3 реплики веб-приложения»). K8s автоматически стремится привести реальное состояние к желаемому.

Service. Абстракция, которая определяет постоянную конечную точку для доступа к разгруженной группе Pod'ов, обеспечивая сетевое взаимодействие и балансировку нагрузки.

Встройка Kubernetes в CI/CD конвейер

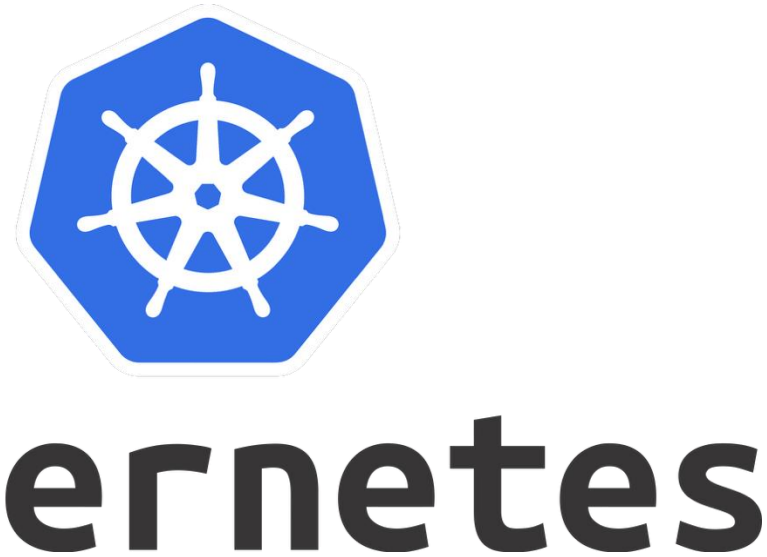
1. После успешного прохождения тестов конвейер помещает собранный Docker-образ в реестр контейнеров.

2. Далее конвейер выполняет команду для обновления конфигурации в кластере Kubernetes. Чаще всего это команда `kubectl set image deployment/my-app my-app=my-registry/app:v2.0`.

3. Kubernetes «замечает», что описание образа в манифесте изменилось.

4. Он запускает стратегию Rolling Update. Это означает, что он постепенно создает поды с новой версией приложения (v2.0) и удаляет поды со старой версией (v1.0), обеспечивая непрерывность обслуживания.

5. Service автоматически направляет трафик на готовые и живые поды новой версии.



Serverless модель

Serverless - модель выполнения, где облачный провайдер динамически управляет выделением и масштабированием машинных ресурсов.

Эволюция абстракции:

Виртуальные машины. Абстракция от физического железа. Управление всей «гостевой» ОС.

Контейнеры. Абстракция от ОС. Управление приложением и его средой.

Serverless (FaaS - Functions as a Service). Абстракция от среды выполнения и инфраструктуры. Управление только бизнес-логикой.

Детализация FaaS-модели

Функция - это автономный фрагмент бизнес-логики, написанный на поддерживаемом языке (Python, Node.js, Go, Java).

Триггеры «заставляют» функцию выполняться:

- HTTP-запросы (через API Gateway)
- события от облачных сервисов (загрузка файла в облачное хранилище, обновление в базе данных, сообщение в очереди)

Жизненный цикл выполнения**:** при поступлении события облачный провайдер подготавливает среду выполнения, выполняет вызванную функцию и возвращает результат. После завершения среда может быть «заморожена» или уничтожена.

AWS Lambda

AWS Lambda - лидер рынка FaaS.

Поддержка разных вариантов рантаймов: Python, Node.js, Java, C#, Go, Ruby, а также собственные рантаймы через Custom Runtime.

Универсальность артефактов: Развертывание в виде ZIP-архива с кодом и зависимостями или в виде **контейнерного образа** до 10 ГБ, что позволяет запускать сложные приложения с уникальными зависимостями.

Интеграция с экосистемой AWS: Более 200 встроенных триггеров от сервисов AWS, что делает Lambda центральным «клеем» для событийных архитектур.

Serverless в CI/CD конвейере

Serverless-функции выгодно использовать в процессе решения эпизодических задач внутри конвейера, например:

Динамические тестовые среды. Для каждого пул-реквеста GIT может запускаться функция, которая разворачивает изолированную копию приложения в контейнере, запускает тесты и затем уничтожает среду.

Интеграционные тесты. Функция может проверять качество кода, проводить статический анализ безопасности (SAST) или отправлять уведомления в Slack/Telegram о результатах сборки.

Обработка веб-хуков (механизм автоматической отправки HTTP-запросов при наступлении определённого события на сервере). Получение уведомлений от Git-репозитория и запуск соответствующего конвейера может быть реализовано через serverless-функцию, что избавляет от необходимости содержать постоянно работающий веб-сервер (Jenkins).

Доставка кода в FaaS

Разработка и сборка. Разработчик пишет код функции и описывает ее конфигурацию (триггеры, права доступа) с помощью инфраструктурного кода (Terraform, Serverless Framework, AWS SAM).

CI/CD-конвейер. Код функции и ее конфигурация помещаются в Git. Конвейер запускает модульные тесты, проверяет код на уязвимости и подготавливает артефакт для развертывания (ZIP-архив или контейнерный образ).

Развертывание. Конвейер с помощью инструментов (например, serverless deploy или terraform apply) развертывает функцию и всю необходимую периферию (API Gateway, очереди) в облаке провайдера.

Гибридный подход: Контейнеры + Serverless

Применимость Kubernetes:

- Долгоживущие (long-running) сервисы, которые должны быть постоянно доступны (сервисы аутентификации).
- Сложные stateful-приложения, требующих локального хранилища (базы данных, очереди сообщений).
- Сервисы со строгими требованиями к сетевой задержке, где «cold start» serverless недопустим.

Применимость Serverless:

- Фоновые и эпизодические задачи. Обработка видео/изображений, массовая рассылка email, выполнение ETL-задач по расписанию.
- Обработчики событий. Реакция на действия пользователей (например, генерация PDF-отчета по запросу).
- API с непостоянной и непредсказуемой нагрузкой.

Архитектурные сложности Контейнеры + Serverless

Kubernetes:

Высокий порог входа. Требует глубоких знаний для корректной настройки и эксплуатации.

Операционная нагрузка. Хотя K8s автоматизирует многое, сам кластер требует обновлений, мониторинга и обслуживания.

Serverless:

Vendor Lock-in. Код и архитектура приложения тесно связываются с сервисами конкретного облачного провайдера.

Холодный старт. Задержка при первом вызове функции после периода бездействия, связанная с подготовкой среды. Может быть критична для приложений реального времени.

Ограничения на время выполнения и ресурсы. Функции имеют лимиты на время выполнения (до 15 минут) и объем доступной памяти.

Референсная архитектура современного веб-приложения

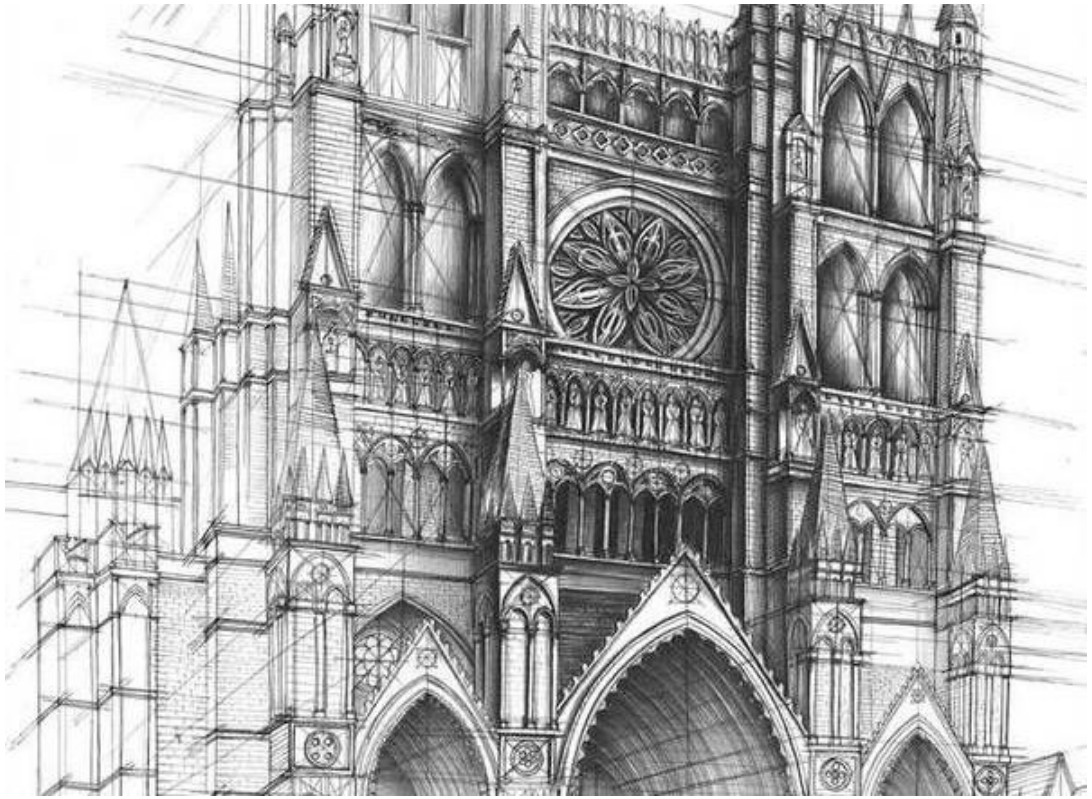
Фронтенд. Некоторое приложение на React. *Размещение:* Статические файлы в облачном хранилище (AWS S3) с доставкой через CDN (AWS CloudFront).

Бэкенд. Микросервисы, отвечающие за ядро бизнес-логики (пользователи, заказы, товары). *Размещение:* Кластер Kubernetes (требуют постоянной доступности, сложной маршрутизации и взаимодействия между собой).

Фоновые задачи (пример):

- **Обработка изображений.** При загрузке пользователем картинки срабатывает Lambda, которая создает превью разных размеров.
- **Генерация отчетов.** Еженедельно по расписанию запускается Lambda, которая формирует сводный отчет и рассылает его менеджерам.

CI/CD. Единый пайплайн, который тестирует, собирает и развертывает все компоненты в их целевые среды (K8s для бэкенда, S3 для фронтенда, Lambda для функций).



Спасибо за внимание!