

# Развёртывание микросервисов

Ранее мы изучали, что в процессе CI\CD сервисы находятся в Docker контейнерах. Но давайте рассмотрим все варианты развёртывания микросервисов.

Развертывание микросервисов может быть выполнено различными способами, и каждый из них имеет свои преимущества и недостатки в зависимости от контекста использования.

## 1. Multiple Service Instances per Host:

Описание:

- На одном хосте(физическом сервере) развертываются множество экземпляров сервиса.

Преимущества:

- Эффективное использование ресурсов хоста.
- Уменьшение затрат на оборудование и инфраструктуру.

Недостатки:

- Взаимное воздействие между сервисами может привести к проблемам производительности и стабильности.
- Может быть сложно управлять зависимостями и версиями сервисов.

## 2. Service Instance per Host:

Описание:

- Каждый экземпляр сервиса развертывается на отдельном хосте.

### **Преимущества:**

- Изоляция сервисов обеспечивает повышенную стабильность и безопасность.
- Упрощается управление зависимостями и версиями сервисов.

### **Недостатки:**

- Может быть неэффективным с точки зрения использования ресурсов.
- Выше затраты на оборудование и инфраструктуру.

## **3. Serverless Deployment:**

### **Описание:**

- Сервисы развертываются на платформе serverless, где ресурсы выделяются динамически.

### **Преимущества:**

- Отсутствие необходимости управления серверами.
- Оплата только за фактически использованные ресурсы.
- Автоматическое масштабирование.

### **Недостатки:**

- Ограничения платформ.
- Зависимость от поставщика услуг.
- Возможные проблемы с холодным стартом.

## **4. Service Instance per VirtualMachine:**

### **Описание:**

- Каждый экземпляр сервиса развертывается внутри своей виртуальной машины.

### **Преимущества:**

- Хорошая изоляция и безопасность.
- Гибкость в управлении ресурсами и конфигурацией.

### **Недостатки:**

- Выше затраты на ресурсы по сравнению с контейнерами.
- Занимает больше времени на старт и остановку по сравнению с контейнерами.

## **5. Service Instance per Container.**

### **Описание:**

- Каждый экземпляр сервиса развертывается внутри своего контейнера.

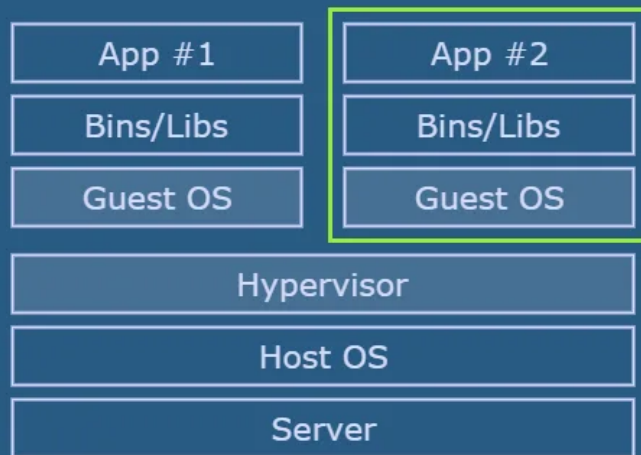
### **Преимущества:**

- Быстрое развертывание и масштабирование.
- Легкость в управлении зависимостями и конфигурацией.
- Эффективное использование ресурсов.

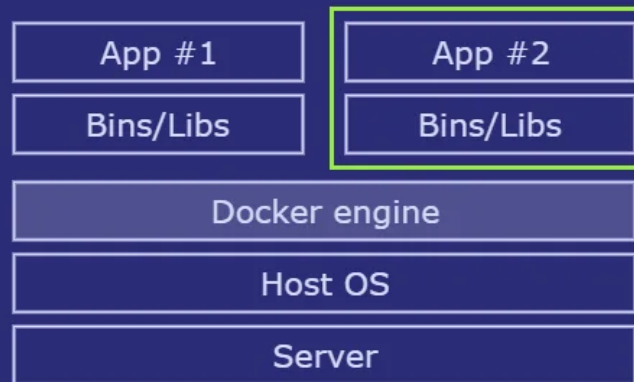
### **Недостатки:**

- Требуется дополнительный инструментарий для управления контейнерами.
- Возможны проблемы безопасности из-за меньшей степени изоляции по сравнению с VM.

## Виртуальные машины



## Контейнеры



разница

|   |  |
|---|--|
| Тяжеловесные                                    | Легковесные                                      |
| Ограниченная производительность                 | "Нативная" производительность                    |
| Каждая VM запускает свою собственную ОС         | Контейнер использует "нативную" ОС               |
| Hardware-level виртуализация                    | ОС виртуализация                                 |
| Запускается минутами                            | Запускается в секунды                            |
| Полностью изолирована, а значит более безопасна | Изоляция уровня процесса, значит менее безопасна |

В современных системах используется часто подход Service Instance per Container, который хорошо зарекомендовал себя в микросервисной архитектуре.

## Стратегия автономности

Стратегия автономности поставки — это концепция, в основе которой лежит принцип, что каждый микросервис должен быть самодостаточным в плане процессов CI/CD (Continuous Integration/Continuous Delivery), и, в идеале, каждый микросервис должен иметь свой собственный процесс поставки и развертывания. Это включает в себя уникальные для каждого сервиса пайплайны CI/CD,

контейнеризацию и автоматизацию тестирования и развертывания.

### 1. Один Сервис

- Каждый микросервис является независимым, собственным API и базой данных (при необходимости).

### 2. Один CI/CD Пайплайн

- У каждого микросервиса должен быть свой процесс непрерывной интеграции и доставки, что позволяет командам разработки работать автономно и уменьшает риск конфликтов.

### 3. Один Контейнер

- Каждый микросервис упаковывается в свой собственный контейнер, что обеспечивает изоляцию, консистентность и портативность.

## Преимущества:

- **Скорость Релизов:**

- Команды могут разрабатывать, тестировать и выпускать свои микросервисы независимо и быстро.

- **Масштабируемость:**

- Отдельные сервисы могут масштабироваться независимо друг от друга в соответствии с их индивидуальными требованиями к ресурсам.

- **Изоляция Рисков:**

- Ошибки в одном сервисе меньше вероятно повлияют на другие сервисы.

- **Улучшенное Тестирование и Отладка:**

- Меньший объем и сложность кода каждого отдельного сервиса облегчают проведение тестирований и отладки.

# Логи и метрики

Инфраструктурные паттерны для микросервисов, такие как сбор метрик приложения (Application Metrics) и агрегация логов (Logs Aggregation), важны в мониторинге, управлении и оптимизации именно микросервисных архитектур.

## 1. Application Metrics (Сбор Метрик Приложения)

Сбор метрик приложения включает в себя отслеживание и анализ данных, таких как использование CPU, использование памяти, количество запросов в секунду и время ответа. Это помогает разработчикам и администраторам системы отслеживать производительность и стабильность сервисов в реальном времени.

### Инструменты:

- **Prometheus** является одним из основных инструментов для сбора метрик, который собирает метрики в реальном времени.
- **Grafana** используется для визуализации данных, собранных Prometheus, позволяя пользователям создавать настраиваемые дашборды для анализа производительности.

### Применение:

Подключив Prometheus к микросервисам, вы можете собирать метрики и анализировать их с помощью Grafana, чтобы определить узкие места производительности или потенциальные проблемы, прежде чем они повлияют на пользователей.

## 2. Logs Aggregation (Агрегация Логов)

Агрегация логов включает в себя сбор, структурирование и хранение логов от различных сервисов в централизованном месте. Это упрощает процесс анализа логов для выявления ошибок, взлома или других проблем.

### Инструменты:

- Системы агрегации логов, такие как **ELK Stack** (Elasticsearch, Logstash, Kibana), позволяют собирать, индексировать и анализировать логи.
- **Kibana** предоставляет интерфейс для визуализации и анализа данных, собранных и индексированных через Elasticsearch.

## Применение:

Логи от каждого микросервиса собираются и агрегируются в Elasticsearch. С помощью Kibana, команды могут затем анализировать эти логи для мониторинга поведения сервисов, отладки и аудита безопасности.

## Хранение конфигураций и секретов (пароли для доступа к базам данных и т.д.)

В микросервисной архитектуре управление конфигурациями и хранение секретных ключей — особенно важно, так как они напрямую влияют на безопасность, стабильность и масштабируемость системы (есть много разных сервисов а значит больше "секрето"). Рассмотрим несколько популярных подходов к хранению конфигураций и секретов.

### 1. Централизованные Серверы Конфигурации:

- **Consul, etcd, Zookeeper:** Эти системы предоставляют централизованное хранилище для конфигураций и могут обеспечивать динамическое обновление конфигураций для микросервисов.

### 2. Сервисы Хранения Секретов:

- **HashiCorp Vault:** Это инструмент для хранения секретов и управления ими, который поддерживает различные бэкенды хранения и предоставляет множество политик контроля доступа.
- **AWS Secrets Manager, Azure Key Vault:** Это облачные сервисы для хранения секретов с расширенными возможностями управления и аудита.

### 3. Конфигурация как Код:

- **Git:** Конфигурации могут храниться в системе контроля версий, например, в Git. Это улучшает отслеживание изменений и версиюность, но не подходит для хранения секретов.

## 4. Environment Variables:

- **Переменные Окружения:** Они могут хранить конфигурационные параметры и секреты, которые можно внедрить в контейнеры при запуске. Однако для секретов это может быть небезопасно, и для них лучше использовать специализированные инструменты.

## 5. Контейнеризация и Оркестрация:

- **Kubernetes Secrets:** Kubernetes предоставляет средства для хранения секретов, которые могут быть связаны с конкретными подами.
- **ConfigMaps:** Это объекты Kubernetes, позволяющие хранить конфигурационные параметры, которые могут быть использованы контейнерами в поде.

## Лучшие практики которым можно следовать:

- **Шифрование:** Все секреты должны быть зашифрованы при хранении и передаваться по защищенным каналам.
- **Минимизация Доступа:** Доступ к секретам должен быть строго ограничен, и предоставляться только тем сервисам и пользователям, которым это действительно необходимо.
- **Регулярное Обновление Секретов:** Секреты должны регулярно обновляться, особенно после инцидентов безопасности.