# 11. Микросервисы: Архитектура, Spring Cloud, Docker & Kubernetes

## 1. Архитектура микросервисов

### REST vs gRPC

| Характеристика | REST (HTTP/JSON) | gRPC (HTTP/2 + Protobuf) |
|---|---|---|
| **Скорость** | Медленнее (текстовый JSON) | Быстрее (бинарный Protobuf) |
| **Поддержка языков** | Любые (HTTP-клиенты) | Множество (но требуется кодогенерация) |
| **Использование** | Веб-API, внешние интеграции | Внутренние сервисы, high-load системы |

**Пример REST (Spring Boot):**

```
@RestController
public class UserController {
    @GetMapping("/users/{id}")
    public User getUser(@PathVariable Long id) { ... }
}
```

**Пример gRPC (Protobuf + Java):**

```
service UserService {
    rpc GetUser (UserRequest) returns (UserResponse);
}
message UserRequest { int64 id = 1; }
```

---

### API Gateway

Единая точка входа для клиентов: маршрутизация, аутентификация, кэширование.

**Функции:**

- **Маршрутизация** (`/orders` → Order Service).
- **Агрегация данных** (объединение ответов от нескольких сервисов).
- **Rate Limiting** (ограничение запросов).

**Примеры:**

- **Spring Cloud Gateway** (современная замена Zuul).
- **Kong** / **Apigee** (продвинутые решения).

---

### Service Discovery

Автоматическое обнаружение сервисов в динамической среде (Docker/K8s).

**Как работает?**

1. Сервис регистрируется в **Eureka** (или Consul/Zookeeper).
2. Другие сервисы запрашивают адрес через Discovery Client.

**Пример (Spring Cloud Eureka):**

```
# application.yml
eureka:
  client:
    serviceUrl:
      defaultZone: http://eureka-server:8761/eureka/
```

---

# 2. Spring Cloud

Набор инструментов для упрощения разработки микросервисов.

## Eureka (Service Discovery)

- Сервер (`@EnableEurekaServer`) и клиенты (`@EnableDiscoveryClient`).
- Сервисы регистрируются при старте и отправляют heartbeat.

## Zuul / Spring Cloud Gateway (API Gateway)

- **Zuul** (устаревший, но простой):

```
@EnableZuulProxy
public class GatewayApplication { ... }
```

- **Spring Cloud Gateway** (актуальный, на WebFlux):

```
spring:
cloud:
    gateway:
      routes:
        - id: order-service
          uri: lb://order-service
          predicates:
            - Path=/orders/**
```

## Feign (REST-клиент)

Генерация HTTP-клиента на основе интерфейса.

**Пример:**

```
@FeignClient(name = "user-service")
public interface UserClient {
    @GetMapping("/users/{id}")
    User getUser(@PathVariable Long id);
}
```

## Hystrix (Circuit Breaker)

Защита от каскадных ошибок:

- **Fallback** – альтернативный ответ при недоступности сервиса.
- **Circuit Breaker** – временный отказ от запросов при частых ошибках.

**Пример:**

```
@HystrixCommand(fallbackMethod = "getDefaultUser")
public User getUser(Long id) { ... }

public User getDefaultUser(Long id) {
    return new User("default", 0);
}
```

> **Важно:** Hystrix deprecated, используйте **Resilience4j** или **Spring Cloud Circuit Breaker**.

---

# 3. Docker + Kubernetes

## Docker (Контейнеризация)

Упаковка сервиса в изолированное окружение со всеми зависимостями.

**Основные команды:**

```
docker build -t my-service .       # Собрать образ
docker run -p 8080:8080 my-service # Запустить контейнер
docker push my-repo/my-service     # Загрузить в реестр (Docker Hub)
```

**Dockerfile:**

```
FROM openjdk:17
COPY target/my-service.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

---

## Kubernetes (Оркестрация)

Управление кластером контейнеров: масштабирование, балансировка, self-healing.

**Основные сущности:**

- **Pod** – минимальная единица (1+ контейнер).
- **Deployment** – декларативное описание работы приложения.
- **Service** – постоянный IP для доступа к Pod’ам.

**Пример Deployment (order-service.yml):**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: order-service
spec:
  replicas: 3
  selector:
```

```
    matchLabels:
      app: order-service
  template:
    metadata:
      labels:
        app: order-service
    spec:
      containers:
        - name: order-service
          image: my-repo/order-service:latest
          ports:
            - containerPort: 8080
```

**Пример Service:**

```
apiVersion: v1
kind: Service
metadata:
  name: order-service
spec:
  selector:
    app: order-service
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
```

---

## Когда что использовать?

| Компонент | Решаемая проблема | Альтернативы |
|---|---|---|
| **API Gateway** | Единая точка входа, аутентификация | Kong, Apigee |
| **Service Discovery** | Динамическое обнаружение сервисов | Consul, Zookeeper |
| **Docker** | Контейнеризация для переносимости | Podman |
| **Kubernetes** | Оркестрация в продакшене | Docker Swarm, Nomad |

---

## Итог

1. **REST/gRPC** – выбор зависит от требований к скорости и совместимости.
2. **Spring Cloud** – упрощает создание микросервисов (Eureka, Feign, Hystrix).
3. **Docker+K8s** – стандарт для развертывания и масштабирования.

Микросервисы требуют инфраструктурных затрат, но дают гибкость и отказоустойчивость! 🚀