# Tổng hợp lý thuyết, khái niệm và ví dụ về kiến trúc Microservice

LexiLingo - DL-Model-Support

Ngày 29 tháng 1 năm 2026

## Mục lục

# 1 Giới thiệu

Kiến trúc microservice là một phong cách thiết kế hệ thống phần mềm trong đó ứng dụng được chia thành nhiều dịch vụ nhỏ, độc lập, mỗi dịch vụ chịu trách nhiệm cho một khả năng nghiệp vụ cụ thể. Mỗi dịch vụ có thể được phát triển, triển khai và mở rộng riêng biệt.

# 2 Khái niệm cốt lõi

## 2.1 Dịch vụ (Service)

Một dịch vụ là một đơn vị triển khai độc lập, sở hữu logic nghiệp vụ riêng và giao tiếp với các dịch vụ khác thông qua giao thức mạng (thường là HTTP/gRPC hoặc message broker).

## 2.2 Ranh giới ngữ cảnh (Bounded Context)

Mỗi microservice nên được thiết kế theo một ranh giới ngữ cảnh nghiệp vụ rõ ràng, tránh chia sẻ trực tiếp mô hình dữ liệu nội bộ với dịch vụ khác.

## 2.3 Tự trị (Autonomy)

Mỗi dịch vụ có vòng đời riêng (CI/CD, versioning, scaling), cơ sở dữ liệu riêng, và có thể sử dụng công nghệ phù hợp nhất cho bài toán.

## 2.4 Giao tiếp (Communication)

Có hai hình thức chính:

- **Đồng bộ:** REST/HTTP, gRPC.
- **Bất đồng bộ:** Message Queue (Kafka, RabbitMQ), Event Bus.

## 2.5 Triển khai (Deployment)

Microservice thường triển khai dưới dạng container (Docker), điều phối bởi Kubernetes hoặc hệ thống tương tự.

# 3 Ưu điểm và nhược điểm

## 3.1 Ưu điểm

- **Mở rộng độc lập:** Mỗi dịch vụ có thể scale theo nhu cầu.
- **Tăng tốc phát triển:** Teams độc lập, phát hành nhanh.
- **Độ bền:** Lỗi cục bộ không làm sập toàn hệ thống.
- **Đa dạng công nghệ:** Polyglot persistence và polyglot programming.

### 3.2 Nhược điểm

- **Độ phức tạp cao:** Quản lý nhiều dịch vụ, mạng, logging, tracing.
- **Nhất quán dữ liệu khó:** Giao dịch phân tán, eventual consistency.
- **Chi phí vận hành:** Yêu cầu hạ tầng giám sát và CI/CD mạnh.

## 4 Kiến trúc tổng quan

### 4.1 Thành phần điển hình

- **API Gateway:** Cổng vào duy nhất cho client, định tuyến, xác thực, giới hạn tốc độ.
- **Service Registry/Discovery:** Đăng ký và tìm kiếm dịch vụ (Consul, Eureka).
- **Configuration Service:** Cấu hình tập trung (Spring Cloud Config).
- **Observability:** Logging tập trung, metrics, tracing (ELK, Prometheus, Jaeger).
- **Message Broker:** Giao tiếp bất đồng bộ, event-driven.

### 4.2 Sơ đồ luồng cơ bản

Ví dụ luồng request từ client đến backend:

1. Client gửi request đến API Gateway.
2. Gateway xác thực, định tuyến đến dịch vụ phù hợp.
3. Dịch vụ xử lý và gọi các dịch vụ khác nếu cần.
4. Kết quả trả về client qua gateway.

## 5 Thiết kế dữ liệu

### 5.1 Database per Service

Mỗi dịch vụ sở hữu DB riêng để tránh coupling và tăng tính tự trị.

### 5.2 Event Sourcing và CQRS

- **CQRS:** Tách mô hình đọc và ghi.
- **Event Sourcing:** Lưu lại mọi thay đổi dưới dạng sự kiện.

### 5.3 Nhất quán dữ liệu

Các chiến lược phổ biến:

- **Saga Pattern:** Giao dịch phân tán theo chuỗi bước.
- **Outbox Pattern:** Đảm bảo event không bị mất.
- **Idempotency:** Chống trùng lặp xử lý.

## 6 Mẫu thiết kế phổ biến

### 6.1 API Gateway Pattern

Tập trung truy cập và bảo vệ hệ thống, giảm số lượng request trực tiếp đến dịch vụ.

### 6.2 Circuit Breaker

Ngăn chặn lỗi lan truyền bằng cách tạm ngưng gọi dịch vụ bị lỗi.

### 6.3 Bulkhead

Cô lập tài nguyên để lỗi ở một dịch vụ không gây ảnh hưởng toàn cục.

### 6.4 Service Mesh

Quản lý giao tiếp, bảo mật, monitoring ở tầng hạ tầng (Istio, Linkerd).

## 7 Ví dụ minh họa

### 7.1 Hệ thống thương mại điện tử

Các microservice cơ bản:

- **User Service**: Quản lý tài khoản.
- **Catalog Service**: Quản lý sản phẩm.
- **Order Service**: Quản lý đơn hàng.
- **Payment Service**: Xử lý thanh toán.
- **Shipping Service**: Vận chuyển.

Luồng đơn hàng (rút gọn):

1. User tạo đơn hàng tại Order Service.
2. Order Service gửi event cho Payment Service.
3. Payment Service xác nhận thành công và phát event.
4. Order Service cập nhật trạng thái và thông báo Shipping Service.

### 7.2 Ví dụ cấu hình API Gateway (pseudo)

```
route /api/orders -> OrderService
route /api/payments -> PaymentService
route /api/catalog -> CatalogService
rate_limit 1000 req/min
jwt_auth enabled
```

### 7.3 Ví dụ giao tiếp bất đồng bộ (pseudo)

```
Event: OrderCreated
Consumer: PaymentService
Action: charge payment and publish PaymentSucceeded
```

## 8 Best Practices

- Thiết kế dịch vụ theo domain-driven design.
- Tách rõ ràng trách nhiệm và sở hữu dữ liệu.
- Tự động hóa CI/CD.
- Implement logging, metrics, tracing ngay từ đầu.
- Dùng versioning cho API.
- Sử dụng hợp lý caching.

## 9 So sánh Microservice và Monolith

| **Tiêu chí** | **Monolith** | **Microservice** |
|---|---|---|
| Triển khai | Một ứng dụng duy nhất | Nhiều dịch vụ độc lập |
| Mở rộng | Scale toàn bộ | Scale từng dịch vụ |
| Phát triển | Team phụ thuộc | Team độc lập |
| Độ phức tạp | Thấp hơn | Cao hơn |
| Thay đổi công nghệ | Khó | Linh hoạt |

## 10 Khi nào nên dùng Microservice?

- Hệ thống lớn, nhiều team phát triển song song.
- Yêu cầu mở rộng linh hoạt từng phần.
- Khả năng vận hành và giám sát đủ mạnh.

# 11 Khi nào không nên dùng?

- Dự án nhỏ, team ít.
- Chi phí vận hành hạn chế.
- Yêu cầu triển khai nhanh, đơn giản.

# 12 Kết luận

Microservice đem lại tính linh hoạt và khả năng mở rộng cao, nhưng đồng thời tăng độ phức tạp vận hành và thiết kế. Cần cân nhắc kỹ quy mô, năng lực team và hạ tầng trước khi áp dụng.