Lesson 3

01

Giới thiệu việc xác định kích thước phù hợp (right-sizing) và ranh giới của microservices. Một ví dụ minh họa được đưa ra bằng việc chọn kích cỡ áo thun phù hợp, tương tự như cách phải chọn kích thước hợp lý cho microservices.

Hai phương pháp phổ biến để xác định kích thước microservices:

- Domain-Driven Sizing: Dựa trên ranh giới miền nghiệp vụ (business domains) và khả năng của doanh nghiệp. Tuy nhiên, phương pháp này mất nhiều thời gian và yêu cầu sự tham gia của các chuyên gia trong tổ chức.
- Event-Storming Sizing: Tổ chức các buổi thảo luận giữa các bên liên quan để xác định các sự kiện, lệnh và phản ứng trong hệ thống. Sau đó, chúng được nhóm lại theo miền nghiệp vụ để xác định microservices.

02

Tiếp tục chủ đề về cách xác định kích thước và ranh giới của các microservices trong hệ thống. Ví dụ được đưa ra là một BankApplication cần được xây dựng hoặc chuyển đổi sang kiến trúc microservices.

Ba nhóm được giao nhiệm vụ đề xuất mô hình phân chia microservices:

Team 1:

- Gộp saving account và trading account vào một microservice.
- Gộp cards và loans vào một microservice.
- Nhược điểm: Dẫn đến tight coupling, gây khó khăn khi mở rộng hoặc thay đổi từng phần.

Team 2:

- Tách saving account và trading account thành 2 microservices riêng biệt.
- Tách cards và loans thành 2 microservices riêng biệt.
- Tổng cộng: 4 microservices.
- Ưu điểm: Loose coupling, linh hoạt hơn trong việc mở rộng và nâng cấp.

Team 3:

- Chia nhỏ hơn nữa, có microservices riêng cho saving account, trading account, debit card, credit card, home loan, vehicle loan, personal loan...
- Ưu điểm: Phân tách chi tiết, mỗi dịch vụ độc lập.
- Nhược điểm: Quá nhiều microservices, gây overhead về vận hành.

Quyết định cuối cùng:

- Phương án của Team 2 được chọn vì cân bằng giữa sự phân tách hợp lý và tính vận hành dễ dàng.
- Nếu trong tương lai có nhu cầu tách nhỏ hơn, có thể điều chỉnh dựa trên tình hình thực tế.

Bài học rút ra:

- Không có kích thước microservices "chuẩn" ngay từ đầu.
- Việc điều chỉnh và tối ưu hóa kiến trúc microservices là quá trình liên tục, không phải quyết định cố định ngay từ ngày đầu tiên.

03

Trình bày về một tình huống di chuyển hệ thống của một startup thương mại điện tử từ kiến trúc monolithic sang kiến trúc microservices, đồng thời nhấn mạnh tầm quan trọng của việc xác định kích thước hợp lý (right-sizing) cho microservices.

1. Kiến trúc monolithic ban đầu:

- Startup ban đầu sử dụng một máy chủ monolithic chứa tất cả các module (định danh, danh mục sản phẩm, đơn hàng, hóa đơn, bán hàng, tiếp thị…).
- Có một cơ sở dữ liệu quan hệ (RDBMS) duy nhất.
- API phục vụ ứng dụng di động và web theo mô hình MVC.
- Ban đầu hệ thống hoạt động tốt vì lượng truy cập thấp.

2. Vấn đề gặp phải với hệ thống monolithic:

- Quá phức tạp: Không ai hiểu toàn bộ hệ thống do nhiều logic kinh doanh tích lũy qua thời gian.
- Ràng buộc chặt chẽ giữa các module, gây khó khăn khi thay đổi hoặc nâng cấp.
- Cập nhật khó khăn: Một thay đổi nhỏ cũng yêu cầu deploy lại toàn bộ hệ thống.
- Khả năng mở rộng kém: Lượng truy cập tăng cao khiến hệ thống quá tải.
- Khó áp dụng công nghệ mới.
- Không thể tổ chức nhóm Agile riêng lẻ vì mọi thứ liên kết chặt chẽ.

3. Giải pháp: Chuyển sang kiến trúc microservices

- Tách nhỏ các module thành các microservices riêng biệt (định danh, danh mục, đơn hàng, hóa đơn...).
- Sử dụng API Gateway để quản lý và điều phối request.
- Triển khai trên container (Docker, Kubernetes) giúp mở rộng dễ dàng.
- Từng microservice có database riêng, phù hợp với yêu cầu nghiệp vụ:
- NoSQL cho các dịch vụ lưu trữ dữ liệu lớn.
- RDBMS cho danh mục sản phẩm.
- Redis Cache cho tiếp thị, bán hàng.
- Tích hợp Event Streaming (Kafka, RabbitMQ) để xử lý bất đồng bộ:
- Gửi OTP qua SMS khi đăng nhập.
- Gửi thông báo khi đơn hàng được xác nhận.

4. Tầm quan trọng của việc xác định kích thước hợp lý (Right-Sizing)

- Nếu chia quá nhỏ: Dẫn đến quá tải về mặt vận hành, khó quản lý.
- Nếu chia quá lớn: Mất đi lợi ích của microservices.
- Cần cân bằng phù hợp, tùy thuộc vào nhu cầu thực tế của tổ chức.

➢ Tổng kết: Việc chuyển đổi sang microservices giúp startup mở rộng quy mô linh hoạt hơn, tối ưu vận hành và phát triển nhanh hơn, nhưng cần xác định kích thước hợp lý để tránh gánh nặng quản lý quá nhiều microservices nhỏ lẻ.

04

Strangler Fig Pattern là một mô hình thiết kế được sử dụng để di chuyển ứng dụng monolithic (nguyên khối) hoặc hệ thống cũ sang kiến trúc microservices theo từng bước mà không gây gián đoạn.

- Cách hoạt động: Ứng dụng cũ sẽ được thay thế dần dần bằng các microservices mới, tương tự như cách cây strangler fig phát triển quanh cây cũ và dần thay thế nó.
- Lợi ích:
  - Giảm rủi ro so với phương pháp Big Bang migration (chuyển đổi toàn bộ hệ thống cùng lúc).
  - Hệ thống cũ và mới có thể cùng tồn tại trong quá trình chuyển đổi.
  - Có thể kiểm tra, so sánh và điều chỉnh trước khi chuyển đổi hoàn toàn.
- Quy trình triển khai:
  1. Xác định các dịch vụ cần tách ra bằng Domain-Driven Design.
  2. Chuyển đổi từng phần của hệ thống monolithic sang microservices.
  3. Sử dụng API Gateway để định tuyến lưu lượng giữa hệ thống cũ và mới.
  4. Loại bỏ hệ thống cũ khi tất cả các chức năng đã được chuyển đổi thành microservices.

Đây là một mô hình phổ biến trong quá trình hiện đại hóa ứng dụng lớn và có thể được hỏi trong các buổi phỏng vấn về migration sang microservices.