**Tìm hiểu Microservices là gì?**

# **1. Tại sao áp dụng microservices?**

Để hiểu rõ microservices, bắt đầu với monolithic software (ứng dụng một khối).

Monolithic software thường sử dụng kiến trúc 3 tầng (three-tier):

+Presentation layer.

+ Business layer.

+ Data access layer.

Tất cả code của 3 tầng (presentation, business layer, and data access layer) sẽ được lưu trữ trong 1 codebase điều này gây ra nhiều vấn đề sau:

+ Vì có 1 codebase, nó sẽ phát sinh code ngày một lớn. Dev UI và Dev Bussiness đều commit lên 1 codebase, nó sẽ làm cho việc quản lý kém hiệu quả.

+ Vì có 1 codebase và các module phụ thuộc vào nhau nên khi có sự thay đổi nhỏ trong 1 module thì sẽ phải deploy lại.

+ Trong monolithic software, sẽ có 3 nhóm theo 3 lớp - UI developers, backend developers, database programmers. Khi có lỗi nghiêm trọng sẽ phải tốn thời gian xác định vấn đề xảy ra ở đâu và ai cần giải quyết vấn đề.

+ Khó có khả năng scale.

Các vấn đề này được giải quyết bằng phương pháp gọi là microservices.

# **2. Microservices là gì?**

Microservices là các service nhỏ, độc lập hoạt động cùng nhau.

Mỗi microservice chịu trách nhiệm riêng, các team có thể phát triển chúng độc lập với các microservice khác.

Sự phụ thuộc duy nhất giữa các microservice là sự giao tiếp giữa chúng.

Và người ta cũng đặt ra câu hỏi về kích thước service: “How small is small?”. Nếu 1 codebase có kích thước mà 1 team quản lý không hiệu quả, thì chúng ta sẽ cần phải tiếp tục chia nhỏ service.

# **3. Lợi ích**

## **3.1 Technology Heterogeneity**

Với microservices, mỗi service là độc lập, mỗi service là 1 project mới, mỗi service có thể được phát triển bằng bất kì ngôn ngữ lập trình nào phù hợp với yêu cầu.

## **3.2 Resilience**

Với microservices, nếu 1 component của hệ thống có vấn đề, nó sẽ không làm ảnh hưởng cả hệ thống, component đó sẽ bị cô lập và phần còn lại của hệ thống vẫn có thể hoạt động bình thương.

Với monilithic software, nếu service bị hỏng, mọi thứ sẽ ngừng hoạt động.

## **3.3 Scaling**

Với monolithic service, chúng ta sẽ phải mở rộng mọi thứ cùng nhau.

Với microservices, chúng ta chỉ cần mở rộng những service mà chúng ta cần phải mở rộng.

## **3.4 Ease of Deployment**

Thay đổi 1 dòng trong 1 ứng dụng có 1 triệu dòng sẽ yêu cầu cả ứng dụng sẽ phải deploy lại. Việc deploy có tác động lớn, rủi ro cao.

Với microservices, chúng ta có thể thực hiện thay đổi trong 1 service và deploy nó độc lập với phần còn lại của hệ thống. Điều này cho phép chúng ta deploy nhanh hơn. Nếu có lỗi xảy ra, service gặp lỗi sẽ được cô lập, giúp cho việc khôi phục nhanh chóng dễ dàng. Điều đó cũng có nghĩa chúng ta có thể đưa ra tính năng mới đến khách hàng nhanh hơn.

## **3.5 Organizational Alignment**

Nhiều người trong chúng ta đã gặp phải các vấn đề liên quan đến các team lớn và codebase lớn. Các vấn đề này có thể trầm trọng hơn khi nhóm bị phân tán. Chúng ta cũng biết rằng các team nhỏ làm việc với codebase nhỏ sẽ đạt hiệu quả cao hơn.

Microservices giúp chúng ta giảm thiểu số lượng người làm việc trên 1 codebase.

## **3.6 Optimizing for Replaceability**

Khi bạn làm việc trong 1 tổ chức vừa hoặc lớn, chắc hẳn bạn sẽ thấy 1 hệ thống nằm 1 góc, không ai muốn động đến chỉ vì nó được viết bằng ngôn ngữ Fortran và chỉ chạy trên phần cứng có tuổi thọ 25 tuổi. Tại sao không ai động đến nó? Vì đó là một công việc quá to lớn và mạo hiểm.

Với microservices, các service có kích thước nhỏ, khi muốn thay thế chúng ta có thể xóa đi và viết lại mà không phải lo lắng, vì các service có kích thước nhỏ, dễ quản lý.

Các team áp dụng microservices rất thoải mái với việc viết lại khi được yêu cầu, và diệt 1 service không còn sử dụng nữa. Khi các codebase chỉ vài trăm dòng, thì chi phí thay thế rất thấp.

# **3. Hạn chế**

Nhược điểm đầu tiên của Microservices cũng chính từ tên gọi của nó. Microservice nhấn mạnh kích thước nhỏ gọn của dịch vụ. Một số developer đề xuất dịch vụ siêu nhỏ cỡ dưới 100 dòng code. Service nhỏ là tốt, nhưng nó không phải mục tiêu chính của Microservice. Mục tiêu của Mircoservice là phân tích đầy đủ ứng dụng để tạo điều kiện phát triển và triển khai ứng dụng nhanh chóng.

Nhược điểm thứ hai của Microservices đến từ đặc điểm hệ thống phân tán. Developer cần phải lựa chọn phát triển mỗi dịch vụ nhỏ giao tiếp với các dịch vụ khác bằng cách nào messaging hay là RPC. Họ còn phải xử lý sự cố khi kết nối chậm, lỗi khi thông điệp không gửi được. Việc này phức tạp hơn nhiều so với ứng dụng nguyên khối nơi các module gọi nhau thông qua các method/procedure cấp ngôn ngữ.

Thứ ba, phải đảm bảo giao dịch phân tán (distributed transaction) cập nhật dữ liệu đúng đắn (all or none) vào nhiều dịch vụ nhỏ khác nhau khó hơn rất nhiều, đôi khi là không thể so với đảm bảo giao dịch cập nhật vào nhiều bảng trong một cơ sở dữ liệu trung tâm. Theo nguyên tắc CAP (CAP theorem) thì giao dịch phân tán sẽ không thể thỏa mãn cả 3 điều kiện:

+ Consistency: dữ liệu ở điểm khác nhau trong mạng phải giống nhau.

+ Availablity: yêu cầu gửi đi phải có phúc đáp.

+ Partition tolerance: hệ thống vẫn hoạt động được ngay cả khi mạng bị lỗi.

Những công nghệ cơ sở dữ liệu phi quan hệ (NoSQL) hay message broker tốt nhất hiện nay cũng chưa vượt qua nguyên tắc CAP.

Thứ tư, testing một dịch vụ trong kiến trúc microservices đôi khi yêu cầu phải chạy cả các dịch vụ nhỏ khác mà nó phụ thuộc. Do đó khi phân rã ứng dụng một khối thành microservices cần luôn kiểm tra mức độ ràng buộc giữa các dịch vụ. Nếu các dịch vụ nhỏ thiết kế phục thuộc vào nhau theo chuỗi. A gọi B, B gọi C, C gọi D. Nếu một mắt xích có giao tiếp API thay đổi, liệu các mắt xích khác có phải thay đổi theo không? Nếu có thì việc bảo trì, kiểm thử sẽ phức tạp tương tự ứng dụng một khối. Thiết kế dịch vụ tốt sẽ giảm tối đa ảnh hưởng lan truyền đến các dịch vụ khác.

Cuối cùng, việc triển khai microservices phức tạp hơn rất nhiều nếu làm thủ công theo cách đã làm với ứng dụng monolithic.