Microservices Thực Tiễn: Từ Thiết Kế Đến Triển Khai

27/09/2017 / Bởi **Ngô Thùy Linh** / trong **Microservice**

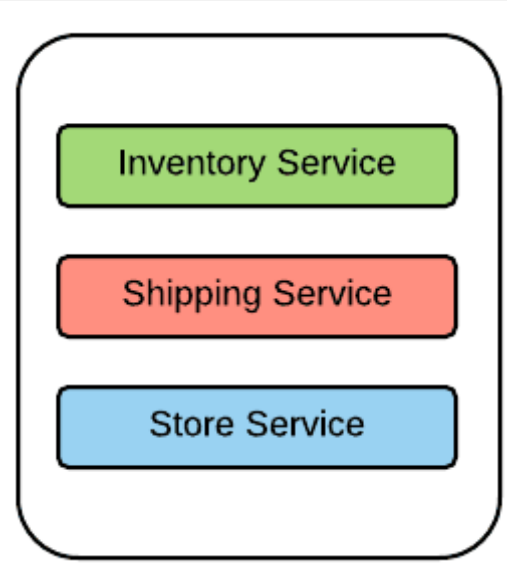
Hiện nay, "Microservices" là một trong những thuật ngữ hay từ khóa phổ biến nhất
Bạn có thể tìm thấy khá nhiều tài nguyên giới thiệu và nói về tính chất cũng như lợi
tài liệu hướng dẫn cách áp dụng microservices trong những hoàn cảnh thực tế.

Bài này chúng ta sẽ tìm hiểu về các khái niệm chính về kiến trúc microservices (kiế
nhỏ) và cách mà bạn sử dụng những nguyên lý này trong thực tiễn.

Kiến trúc một khối truyền thống (Monolithic Architecture)

Ứng dụng phần mềm doanh nghiệp được thiết kế để đáp ứng nhiều yêu cầu kinh doanh của doanh nghiệp. Do đó, các phần mềm cung cấp hàng trăm các tính năng và tất cả những tính năng này đều được gói trong một ứng dụng monolithic. Ví dụ, ERPs, CRMs hay nhiều phần mềm khác chứa hàng trăm tính năng. Việc triển khải, sử lỗi, mở rộng và nâng cấp những phần mềm khổng lồ này trở thành một cơn ác mộng.

Kiến trúc hướng dịch vụ (Service Oriented Architecture - SOA) được thiết kế để giải quyết một phần của vấn đề bằng cách giới thiệu khái niệm "service". Một dịch vụ là một nhóm tổng hợp các tính năng tương tự trong một ứng dụng. Do đó trong SOA, ứng dụng phần mềm được thiết kế như một tổ hợp của các dịch vụ. Tuy nhiên, với SOA, giới hạn hay phạm vi của một dịch vụ khá là rộng và được định nghĩa khá "thô" (coarse-grained). Việc này khiến các services cũng có thể trở nên quá to và phức tạp.



Hình 1: Monolithic architecture

Trong phần lớn trường hợp, dịch vụ trong SOA là độc lập nhưng chúng lại được triển khải chung. Tương tự như ứng dụng monolithic, những dịch vụ này to và phức tạp lên theo thời gian vì thường xuyên thêm các tính năng. Và thế là những ứng

dụng này lại trở thành một mớ các dịch vụ monolithic, cũng không còn khác mấy so với kiến trúc một khối thông thường. Hình 1 thể hiện một ứng dụng gồm nhiều services. Những services này được triển khai cùng một lúc vào 1 ứng dụng lớn. Dù bên trong có gồm các services thì đây là một ứng dụng monolithic. Một số tính chất của kiến trúc một khối:

- Được thiết kế, phát triển và triển khai theo một khối duy nhất
- Ứng dụng monolithic phức tạp và to gây khó khăn cho việc bảo trì, nâng cấp và thêm tính năng mới
- Khó áp dụng phát triển kiểu agile
- Phải triển khai lại toàn bộ hệ thống dù chỉ cập nhật hay nâng cấp một phần
- Mở rộng: phải mở rộng cả khối ứng dụng, gặp khó khăn nếu có các yêu cầu về tài nguyên khác nhau (ví dụ một service yêu cầu thêm CPU, service khác lại yêu cầu nhiều memory)
- Độ tin cậy: một service không ổn định có thể sập cả hệ thống
- Khó đổi mới: ứng dụng monolithic phải sử dụng chung công nghệ nên khó thay đổi hay áp dụng công nghệ mới

Những tính chất giới hạn trên của kiến trúc một khối dẫn tới sự phát triển của kiến trúc dịch vụ nhỏ (Microservices Architecture).

Kiến trúc dịch vụ nhỏ (Microservices Architecture)

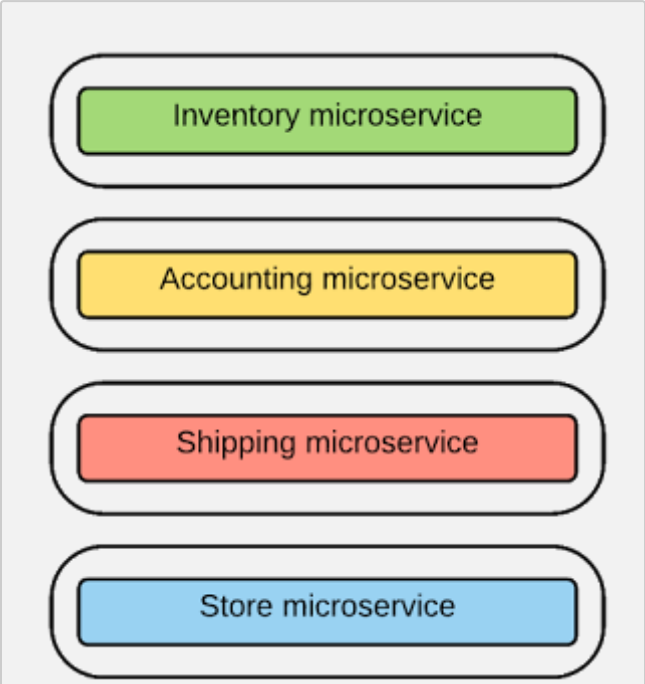
Nền tảng cả kiến trúc microservices là xây dựng một ứng dụng mà ứng dụng này là tổng hợp của nhiều services nhỏ và độc lập có thể chạy riêng biệt, phát triển và triển khai độc lập.

Một số khái niệm về microservices nói về quá trình chia tách ứng dụng monolithic
nhiên, theo quan điểm của tôi, microservices không chỉ về chia tách các servcies

Ý tưởng quan trọng chính là nhìn vào các tính năng trong một ứng dụng monolithi
cầu và khả năng cần thiết để đáp ứng một nghiệp vụ. Sau đó từng năng lực nghiệp
service nhỏ, độc lập. Những services này có thể sử dụng các nền tảng công nghệ
thể và có giới hạn.

Theo đó, ví dụ về hệ thống trong hình 1 có thể được chia theo microservices như trong hình 2. Đây có thể là một ứng dụng phần mềm bán hàng, với kiến trúc microservices, mỗi chức năng kinh doanh hay trong doanh ngiệp được tách

thành một microservices. Trong kiến trúc microservices, một service mới được tạo ra từ những services gốc trong kiến trúc monolithic.



Hình 2: Microservices Architecture

Tiếp theo, chúng ta sẽ nói về các nguyên tắc chính của kiến trúc microservices và
được sử dụng trong thực tế.

Thiết kế Microservices: kích cỡ, phạm vi và tí

Bạn có thể xây dựng ứng dụng mới với microservices hoặc chuyển đổi ứng dụng sẵn có sang microservices. Với cách nào thì việc quyết định kích cỡ, phạm vi và tính năng của microservices rất quan trọng. Có thể đây chính là phần khó nhất bạn gặp phải khi phát triển hệ thống microservices trong thực tế.

Dưới đây chúng ta sẽ xem xét một số vấn đề thực tế và những hiểu sai về kích cỡ, phạm vi và chức năng của microservices.

- Số dòng code/ kích cỡ của một đội lập trình là chỉ số tồi: có vài cuộc bàn luận về kích thước của một service dựa vào số lượng dòng code hay kích thước của đội phát triển service đó (ví dụ two-pizza team). Tuy nhiên, những cách đo đếm này không thực tiễn và không chính xác, vì ta có thể phát triển services với ít dòng code hoặc với một đội nhỏ nhưng hoàn toàn vi phạm các nguyên tắc trong kiến trúc microservices.
- "Micro" là một từ khóa dễ gây nhầm lẫn. Một số lập trình viên nghĩ rằng họ nên tạo ra services nhỏ hết mức. Điều này là một cách hiểu sai.
- Trong SOA, services thường trở thành các cục monolithic với nhiều hàm, chức năng khác hỗ trợ. Vì vậy, chỉ phát triển services kiểu SOA rồi dán nhãn microservices hoàn toàn lạc hướng và không mang lại bất kì lợi ích nào của kiến trúc microservices.

Do đó, câu hỏi là chúng ta nên thiết kế services trong kiến trúc microservices như thế nào thì phù hợp và đúng mực?

Một Số Chỉ Dẫn Khi Thiết Kế Microservices

- Single Responsibility Principle (SRP): một service với phạm vi và chức năng giới hạn, tập trung vào một nhiệm vụ giúp quá trình phát triển và triển khai dịch vụ trở nên nhanh chóng hơn.
- Trong quá trình thiết kế, ta nên xác định và giới hạn các services theo chức năng nghiệp vụ thực tế (theo Domain-Driven-Design)
- Đảm bảo microservices có thể phát triển và triển khai độc lập
- Mục tiêu của thiết kế là phạm vi của microservices phục vụ một nghiệp vụ chứ khô
  hơn. Kích thước hợp lý của một service là kích thước đủ để đáp ứng yêu cầu của m
- Khác với services trong SOA, một microservice không nên có quá nhiều hàm hay c
  dạng thông báo/ gửi tin (messaging) đơn giản.
- Một cách tốt là có thể bắt đầu với services to có phạm vi rộng rồi chia nhỏ dần (dự
  thống)

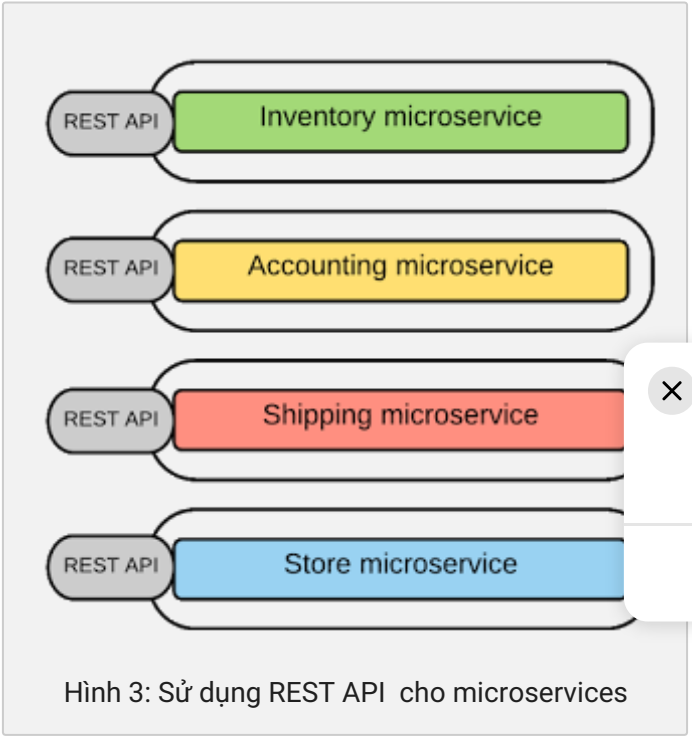
Với ví dụ hệ thống bán hàng trong hình, bạn có thể thấy ứng dụng monolithic được tách thành bốn microservices là "inventory", "accounting", "shipping" và "store". Chúng giải quyết một yêu cầu giới hạn nhất định nhưng tập trung vào chức năng đó. Theo đó, mỗi service tách biệt hoàn toàn khỏi nhau và đảm bảo tính linh hoạt và độc lập.

Liên lạc giữa Microservices

Trong ứng dụng monolithic, các chức năng khác nhau nằm trong các component khác nhau được kết nối bằng cách gọi hàm hay phương thức. Trong SOA, việc này được chuyển sang một chế độ tách rời hơn với kiểu nhắn tin qua các dịch vụ web (web service messaging), phần lớn dùng SOAP trên nền phương thức HTTP, JMS. Những webservices này khá phức tạp. Với microservices, yêu cầu là phải có một cơ chế truyền tin đơn giản và nhẹ.

Gửi Tin Đồng Bộ - REST, Thrift

Với truyền tin đồng bộ (người gửi - client sẽ chờ một khoảng thời gian để nhận kết quả từ service), REST là sự lựa chọn hàng đầu vì nó cung cấp hệ thống truyền tin đơn giản qua giao thức HTTP dạng request - response. Do đó, nhiều microservices sử dụng HTTP với API. Mỗi chức năng xuất ra API.



Hình 3: Sử dụng REST API cho microservices

Ngoài REST, Thrift cũng được sử dụng để truyền tin đồng bộ.

Gửi Tin Bất Đồng Bộ - AMQP, STOMP, MQTT

Trong một số hoàn cảnh, truyền tin bất đồng bộ là cần thiết (client không mong đợi response ngay lập tức, hoặc không cần response). Các giao thức truyền tin bất đồng bộ như AMQP, STOMP hay MQTT được sử dụng rộng rãi.

Các Kiểu Tin Nhắn - JSON, XML, Thrift, ProtoBuf, Avro

Quyết định kiểu tin nhắn phù hợp cho microservices cũng là một yếu tố quan trọng. Với phần lớn các ứng dụng microservices, họ sử dụng những kiểu tin nhắn dạng chữ như JSON và XML trên nền giao thức HTTP với API. Trong trường hợp cần truyền tin dạng nhị phân, microservices có thể dùng dạng Thrift, Proto hay Avro.

Service Contracts - Định Nghĩa Service Interfaces - Swagger, RAML

Khi bạn có một nghiệp vụ được xây dựng như một dịch vụ, bạn cần định nghĩa và thông báo hợp đồng dịch vụ (service contract thể hiện giao kèo của service).

Bởi vì chúng ta xây dựng microservices trên kiểu kiến trúc REST, ta có thể sử dụng cùng kiểu REST API để định nghĩa hợp đồng của microservices. Do đó, microservices sử dụng các ngôn ngữ định nghĩa REST API tiêu chuẩn như Swagger, RAML để định nghĩa hợp đồng dịch vụ.

Kết Nối Microservices (Giao Tiếp Giữa Các Se

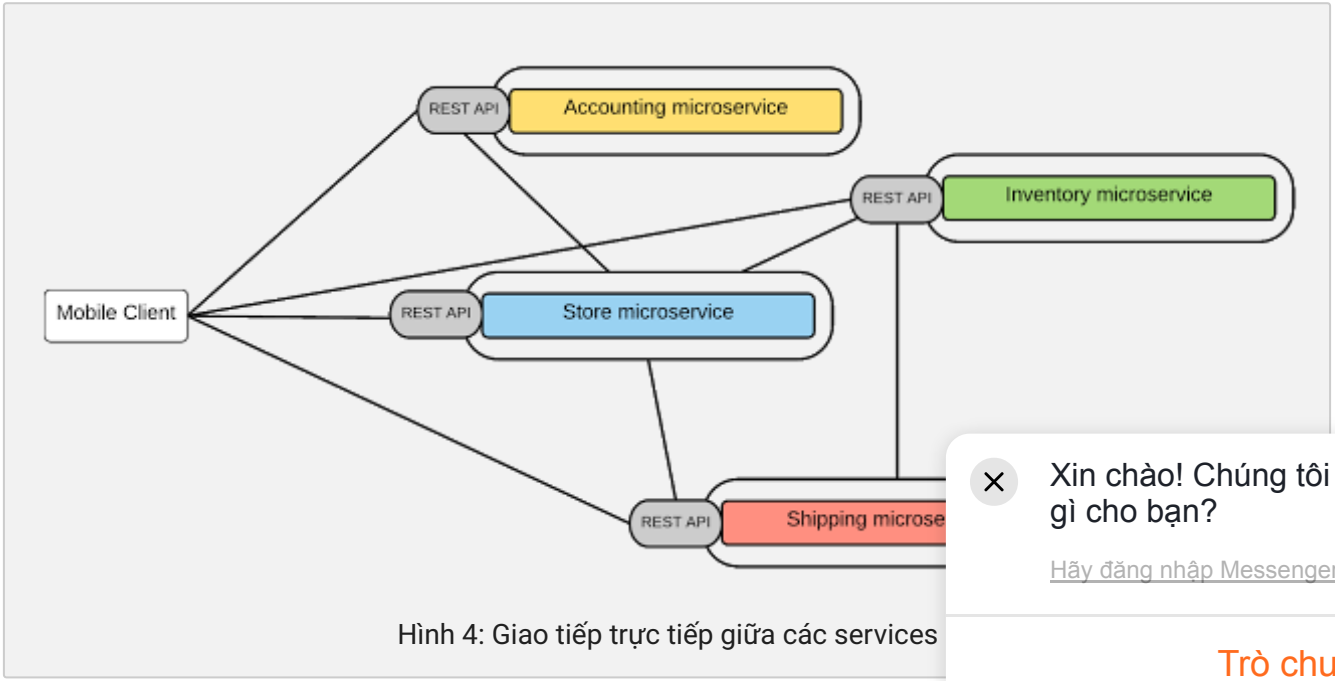
Trong kiến trúc microservices, ứng dụng phần mềm được cấu thành từ các dịch vụ
của người dùng trên phần mềm, kết nối và giao tiếp giữa các microservices là cần
khác nhau lên các services. Vì vậy, giao tiếp giữa các microservices là một vấn đề

Khi áp dụng SOA, giao tiếp giữa các services được tiến hành bởi một Enterprise Service Bus (ESB) và phần lớn logic nằm trong tầng trung gian này (định tuyến cho tin nhắn, chuyển đổi và điều phối tin nhắn). Tuy nhiên, kiến trúc microservices thúc đẩy việc loại trừ một tầng giao tiếp trung gian tập trung/ ESB và chuyển phần logic (business logic) sang các services (dạng Smart Endpoints).

Bởi vì microservices sử dụng giao thức tiêu chuẩn HTTP, JSON, ..., yêu cầu để kết nối một giao thức khác biệt được tối giản. Một lựa chọn khác cho giao tiếp giữa microservices là sử dụng một message bus nhẹ hay gateway với khả năng định tuyến tối thiểu hoạt động như những đường truyền đơn thuần không xử lý business logic gì hết (dump pipe). Dựa vào những phong cách này, có một số kiểu mẫu giao tiếp trong microservices như dưới đây.

Point-to-point - Kết Nối Trực Tiếp Giữa Các Services

Với kiểu điểm nối điểm, toàn bộ logic của việc định tuyến truyền tin nhắn nằm trong mỗi điểm cuối (endpoint) hay chính là các services. Và services nói chuyện trực tiếp với nhau. Mỗi service mở ra một REST APIs và bất kì service hay khách hàng bên ngoài nào cũng có thể gọi service qua REST API của nó.



Hình 4: Giao tiếp trực tiếp giữa các services

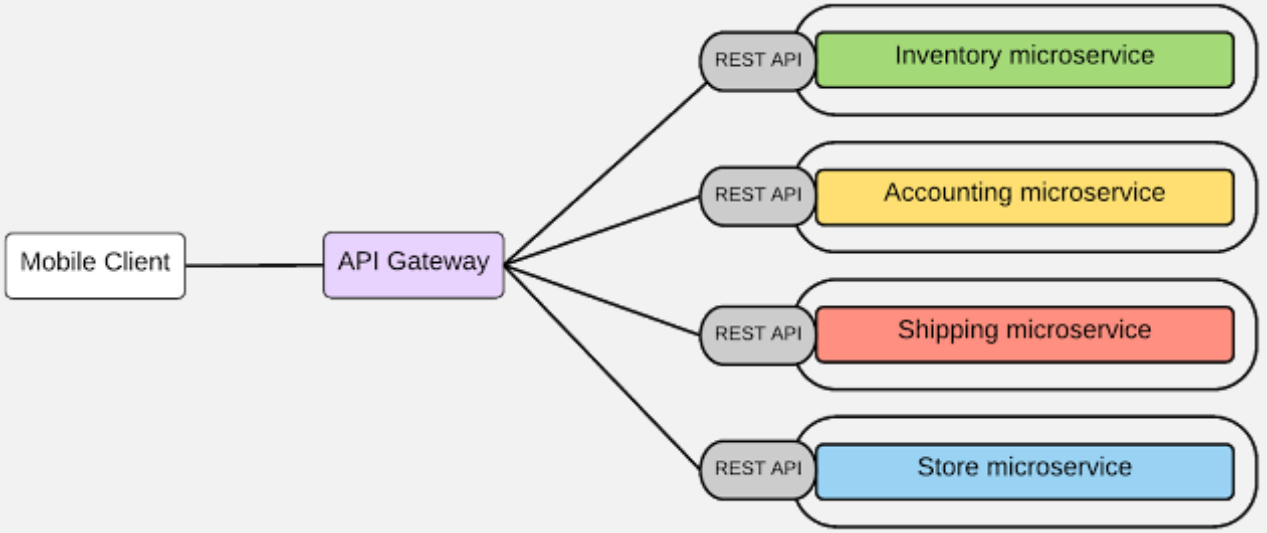
Rõ ràng là mô hình này đơn giản và hoạt động ổn với ứng dụng microservices tương đối nhỏ nhưng khi số lượng services tăng lên, việc giao tiếp trở nên cực phức tạp. Lí do này cũng chính là tác dụng của ESB trong SOA truyền thống, chính để loại bỏ kết nối trực tiếp phức tạp và rối rắm. Tổng hợp một số vấn đề của kiểu giao tiếp trực tiếp.

- Những yêu cầu như xác thực người dùng, điều tiết, giám sát,...phải được xây dựng tại tất cả microservices.
- Việc trên dẫn đến lập các tính năng chung, mỗi microservices có thể trở nên phức tạp.
- Không có cách quản lý, kiểm soát giao tiếp giữa các services
- Thường việc kết nối trực tiếp trong microservices được coi là anti-pattern khi áp dụng cho ứng dụng to.

Vì vậy, với những trường hợp phức tạp hơn, thay vì kết nối trực tiếp hay một ESB trung tâm, chúng ta có thể sử dụng 1 messaging bus trung tâm gọn nhẹ. Nó cung cấp một lớp trừu tượng hóa các microservices (an abstraction layer). Cách này được gọi là API Gateway.

API-Gateway

Ý tưởng chính của API Gateway là sử dụng một cổng truyền tin gọn nhẹ như một điểm vào chính cho tất cả khác hàng, người dùng và triển khai những chức năng chung mà không liên quan đến nghiệp vụ đặc thù ở cấp Gateway này. Nhìn chung, một API Gateway cho phép bạn sử dụng một API được quản lý qua REST/HTTP. Vì thế, chúng ta có thể cung cấp tất cả các chức năng nghiệp vụ được phát triển thành microservices qua API Gateway như một APIs tập trung được quản lý.



Hình 5: Tất cả microservices được xuất qua API Gateway

Trong ví dụ bán hàng của chúng ta, tất cả microservices đều được xuất qua API Gateway và đây là một chốt đầu vào duy nhất cho người dùng. Nếu một microservice muốn giao tiếp với một microservice khác thì cần đi qua API Gateway.

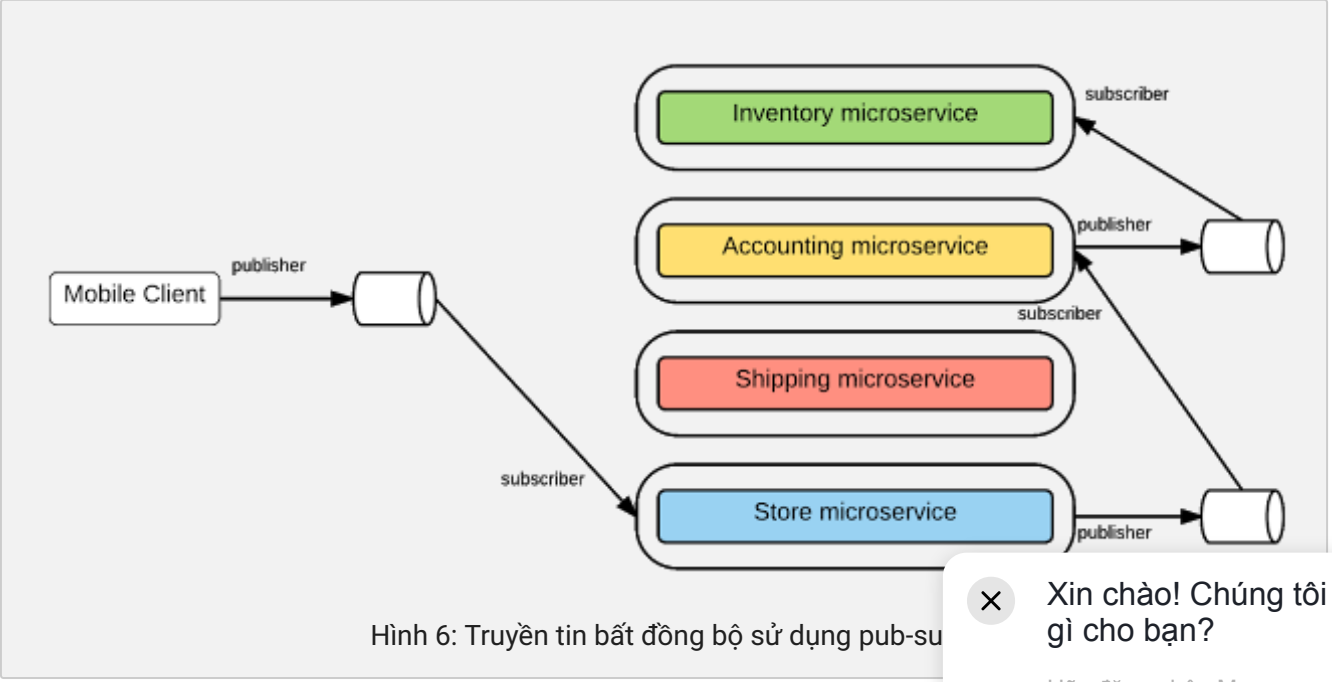
API Gateway cung cấp các lợi thế dưới đây:

- Cung cấp một lớp trừu tượng hóa các microservices. Ví dụ thay vì cung cấp một AI gateway có thể xuất ra hay hiển thị API khác nhau cho mỗi khách hàng.
- Định tuyến và chuyển đổi tin nhắn gọn nhẹ ở cấp gateway.
- Một điểm tập trung cho các chức năng chung không mang tính nghiệp vụ kinh doa tiết.
- Với API Gateway. microservices trở nên càng gọn nhẹ vì các chức năng chung không mang tính nghiệp vụ đều chuyển sang Gateway.

API Gateway có thể là kiểu mẫu được sử dụng rộng rãi nhất trong triển khai microservices.

Message Broker - Người truyền tin trung gian

Microservices có thể được kết nối qua truyền tin bất đồng bộ như yêu cầu một chiều (requests) hay thông báo/ đăng kí nhận thông báo (publish/ subscribe) qua queues hay topics. Với publish/ subscribe, một service có thể tạo tin nhắn và gửi bất đồng bộ đến một hàng chờ (queue hay topic). Sau đó service khác có thể nhận tin này từ queue hay topic. Phong cách này tách biệt người gửi và người nhận, và người truyền tin trung gian sẽ lưu tin nhắn đến khi người nhận có thể xử lý. Người gửi hoàn toàn không biết gì về người nhận.

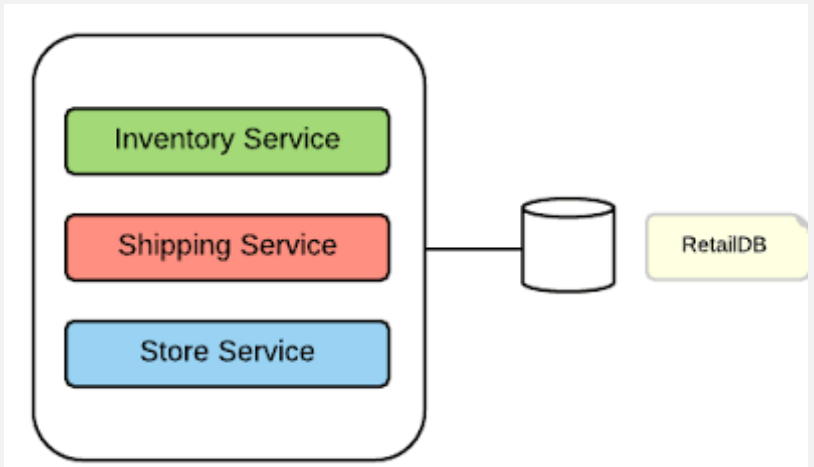


Hình 6: Truyền tin bất đồng bộ sử dụng pub-su

Giao tiếp giữa người gửi/ người nhận được tạo ra bởi message broker qua các tiêu
AMQP, MQTT,...

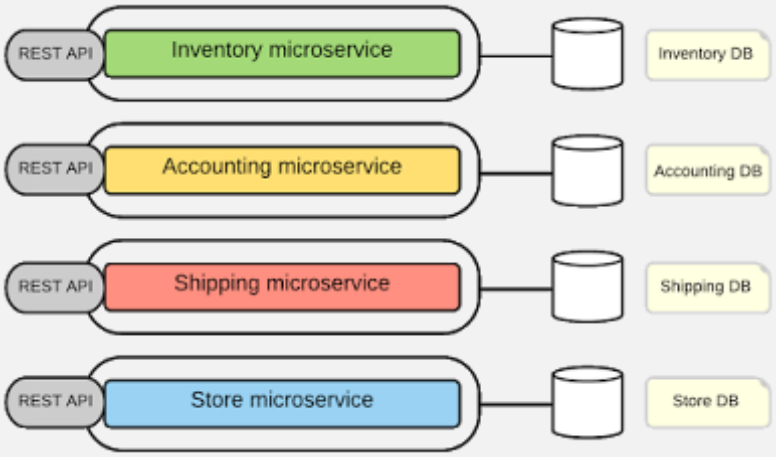
Quản Lý Cơ Sở Dữ Liệu Phân Tán

Trong cấu trúc monolithic, ứng dụng lưu dữ liệu vào một cơ sở dữ liệu (CSDL) tập trung duy nhất.



Hình 7: Ứng dụng monolithic với CSDL tập trung

Trong microservices, các chức năng được tách thành nhiều microservices và nếu sử dụng cùng một CSDL trung tâm thì microservices không còn độc lập. Thay đổi dữ liệu của một service có thể làm hỏng các services khác. Do đó, mỗi microservice phải có CSDL riêng.



Hình 8: Microservices với CSDL riêng biệt

Dưới đây là vài đặc điểm chính khi phát triển việc quản lý dữ liệu phân tán trong kiến trúc microservices.

- Mỗi service có thể có một CSDL riêng biệt để lưu trữ thông tin cần thiết cho nghiệp vụ của service đó.
- Một microservice chỉ có thể truy xuất vào CSDL riêng của nó mà không có quyền truy xuất vào CSDL của microservices khác.
- Trong một số hoàn cảnh, để hoàn thành một tác vụ bạn cần phải cập nhật nhiều CS
  khác chỉ nên được cập nhật qua API của những dịch vụ này (không trực tiếp thay đ

Việc phân tách quản lý dữ liệu cho phép bạn hoàn toàn phân tách các microservic
lý dữ liệu khác nhau (SQL hay NoSQL,...nhiều CSDL khác nhau cho các microservi
tạp liên quan đến nhiều microservices, giao dịch phải được thực hiện qua API.

Quản Trị Phân Tán

Kiến trúc microservices ủng hộ quản trị phân tán.

Nhìn chung, quản trị nghĩa là thiết lập và thực thi phương thức mà con người và các giải pháp làm việc cùng nhau để đạt được mục tiêu. Trong ngữ cảnh SOA, quản trị SOA (SOA governance) hướng dẫn việc phát triển các services có thể tái sử dụng, thiết lập cách services được thiết kế, phát triển và cách chúng thay đổi theo thời gian. Nó thiết lập một giao kèo giữa người cung cấp services và người sử dụng services, thông báo cho người sử dụng những thứ họ có thể nhận được và người sử dụng những thứ họ phải cung cấp. Trong quản trị SOA, có hai kiểu quản trị phổ biến:

- Quản trị khi thiết kế - định nghĩa và kiểm soát việc tạo services, thiết kế và thực thi các chính sách của services
- Quản trị khi triển khai/ chạy - khả năng đảm bảo các chính sách của services trong quá trình hoạt động

Vậy trong microservices thì quản trị là gì? Microservices được xây dựng động lập và phân tán với nền tảng công nghệ khác nhau. Do đó, việc thiết lập một tiêu chuẩn thiết kế và phát triển chung là không cần thiết. Chúng ta có thể kết luận về khả năng quản trị phân tán trong kiến trúc microservices như sau:

- Không cần phải có một hệ quản trị tập trung khi thiết kế
- Microservices có thể tự quyết định thiết kế và phát triển của nó
- Kiến trúc ủng hộ và hỗ trợ việc chia sẻ các services chung hay có thể tái sử dụng
- Một số mặt của quản trị trong quá trình hoạt động như SLAs. điều phối, giám sát, bảo mật hay tìm kiếm dịch vụ (service discovery) có thể được phát triển ở cấp API Gateway

Service Registry & Service Discorvery (Truy Tìm Dịch Vụ)

Trong microservices, số lượng services mà bạn cần xử lý khá lớn. Và địa điểm của
thường xuyên vì tính chất của việc phát triển microservices là nhanh. Nên bạn cần
service trong quá trình chạy. Giải pháp cho vấn đề này là Service Registry.

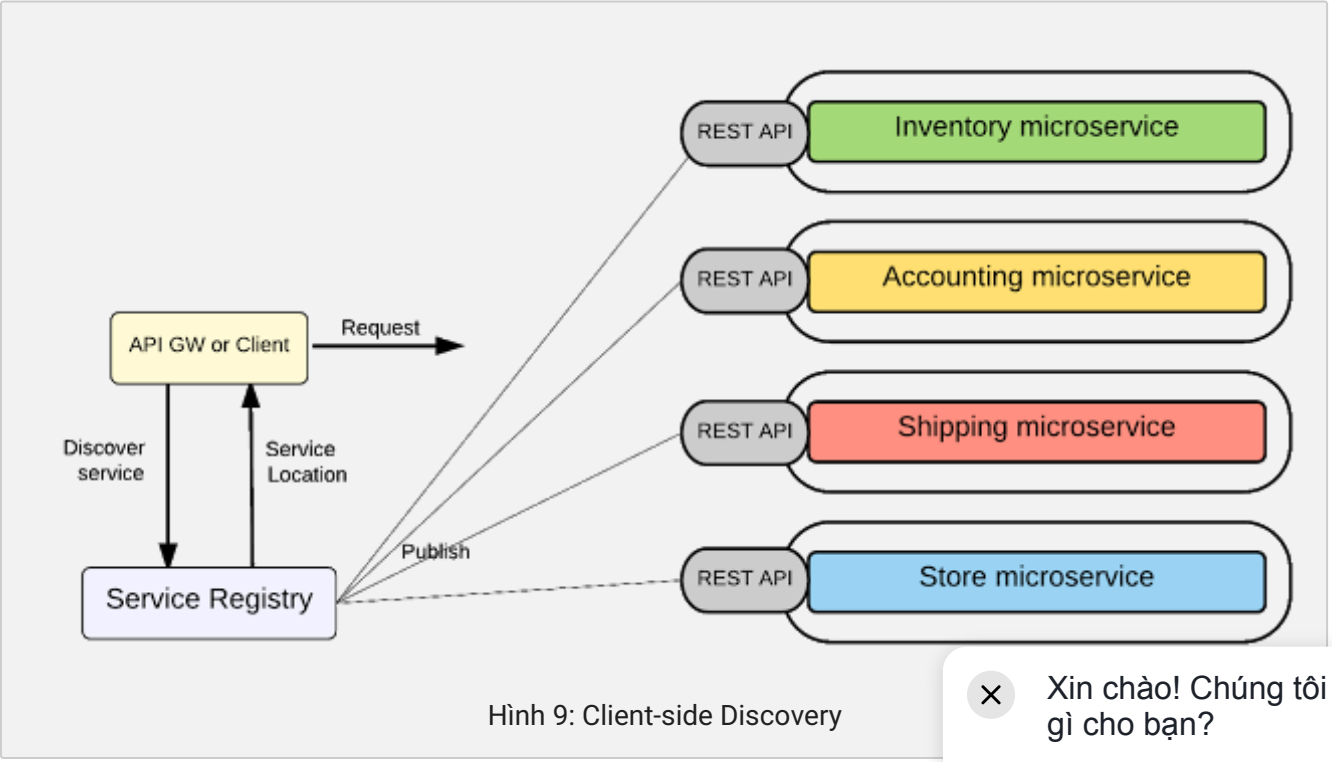
Service Registry

Service Registry giữ các thực thể microservices và địa chỉ của chúng. Thực thể mi
registery khi bắt đầu chạy và hủy đăng kí khi tắt. Người dùng có thể tìm các services đang tồn tại và địa chỉ của chúng qua service registry.

Service Discovery

Để tìm các microservices đang tồn tại và địa điểm của chúng, chúng ta cần một quy trình truy tìm dịch vụ. Có hai mô hình là Client-side Discovery và Server-side Discovery. Hãy cùng xem xét hai mô hình này.

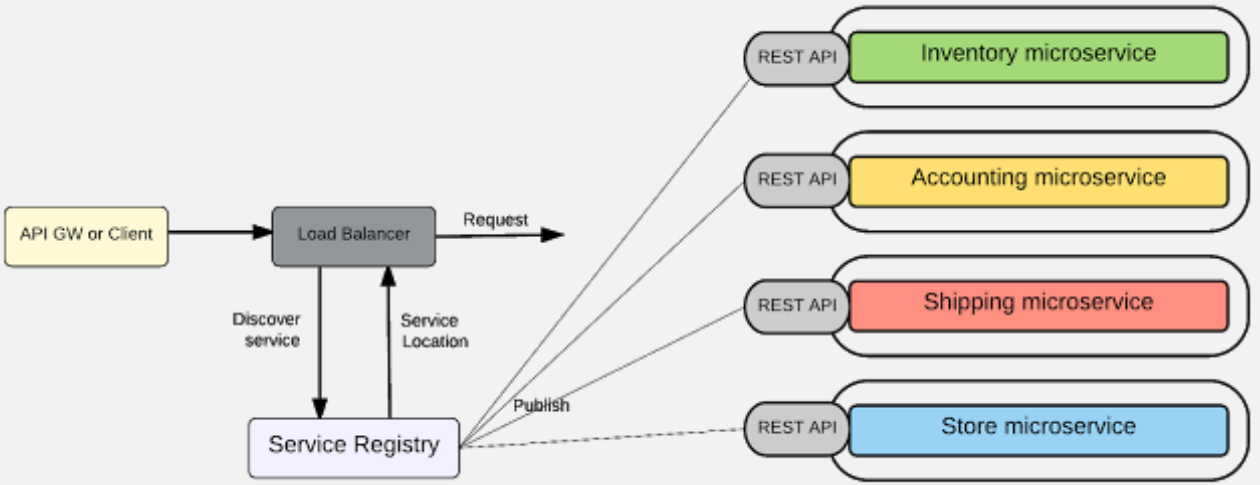
Client-side Discovery - Với mô hình này, client hay API Gateway lấy thông tin địa điểm của một thực thể service bằng cách truy vấn Service Registry.



Hình 9: Client-side Discovery

Client hay API Gateway phải thực hiện logic tìm kiếm service bằng cách gọi vào Se

Server-side Discovery - Với cách này, client hay API Gateway gửi yêu cầu lên một co
Balancer). Component này sẽ gọi Service Registry và quyết định địa điểm của các microservices.



Hình 10: Server-side Discovery

Một số giải pháp triển khai microservices như Kubernetes cung cấp mô hình server-side discovery.

Deployment

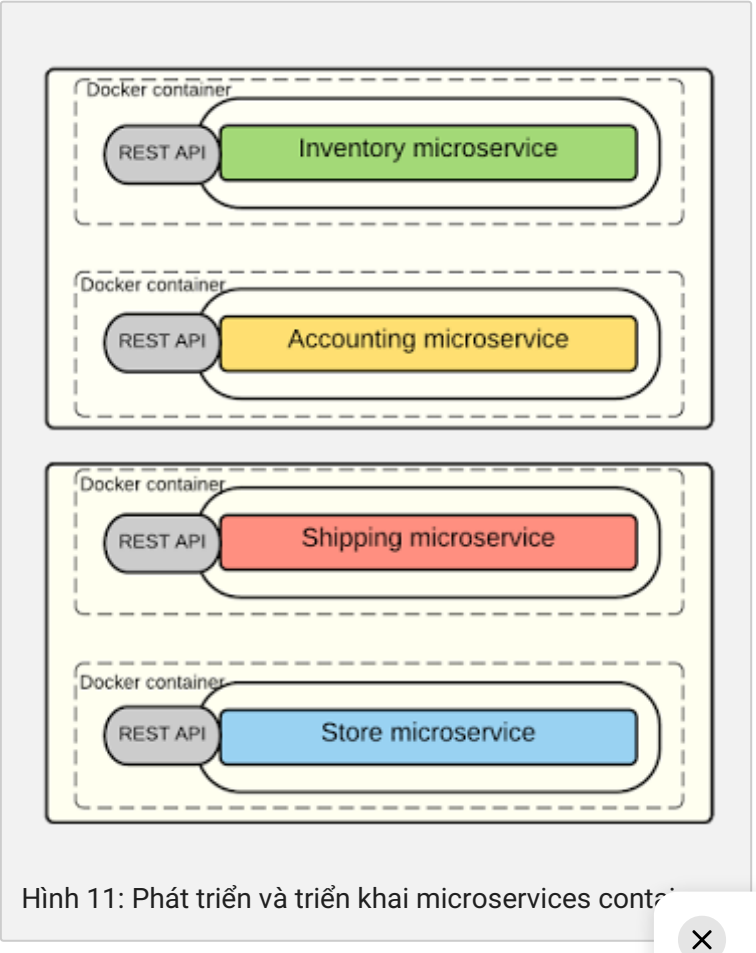
Việc triển khai các dịch vụ có một trò trọng yếu và gồm các yêu cầu sau:

- Khả năng triển khai/ gỡ xuống độc lập mà không ảnh hưởng đến dịch vụ khác
- Có thể mở rộng theo cấp microservices, chỉ mở rộng microservices cần thiết
- Phát triển và triển khai microservices nhanh chóng
- Một microservice ngắt kết nối hay sập thì không ảnh hưởng các dịch vụ khác

Docker (một công cụ mã nguồn mở cho phép lập trình viên và quản trị viên hệ thống triển khai các ứng dụng thành các containers trên môi trường Linux) cung cấp một công cụ tuyệt vời để triển khai microservices đáp ứng đủ các yêu cầu trên. Các bước chính gồm:

- Đóng gói mỗi microservice thành một ảnh Docker (docker image)
- Triển khai mỗi thực thể của service là một Docker container
- Mở dộng dựa vào số lượng thực thể
- Phát triển, triển khai và khởi động microservices trở nên nhanh hơn với Docker (nhanh hơn nhiều các máy ảo thông thường - VM)

Kubernetes mở rộng khả năng của Docker bằng cách quản lý một cụm các Linux container như một hệ thống duy nhất, quản lý và chạy các Docker containers trên nhiều hosts, cung cấp service discovery và kiểm soát việc nhân rộng. Như bạn có thể thấy, những tính năng này cần thiết cho kiến trúc microservices. Vì vậy sử dụng Kurbernetes (trên nền Docker) để triển khai microservices đang trở thành một phương pháp cực kì hữu dụng, đặc biệt với ứng dụng lớn.



Hình 11: Phát triển và triển khai microservices conta

Hình 11 thể hiện tổng quát việc triển khai các microservices. Mỗi thực thể của mic
container và có hai containers trên mỗi host. Bạn có thể thay đổi số lượng contain

Security - Bảo Mật

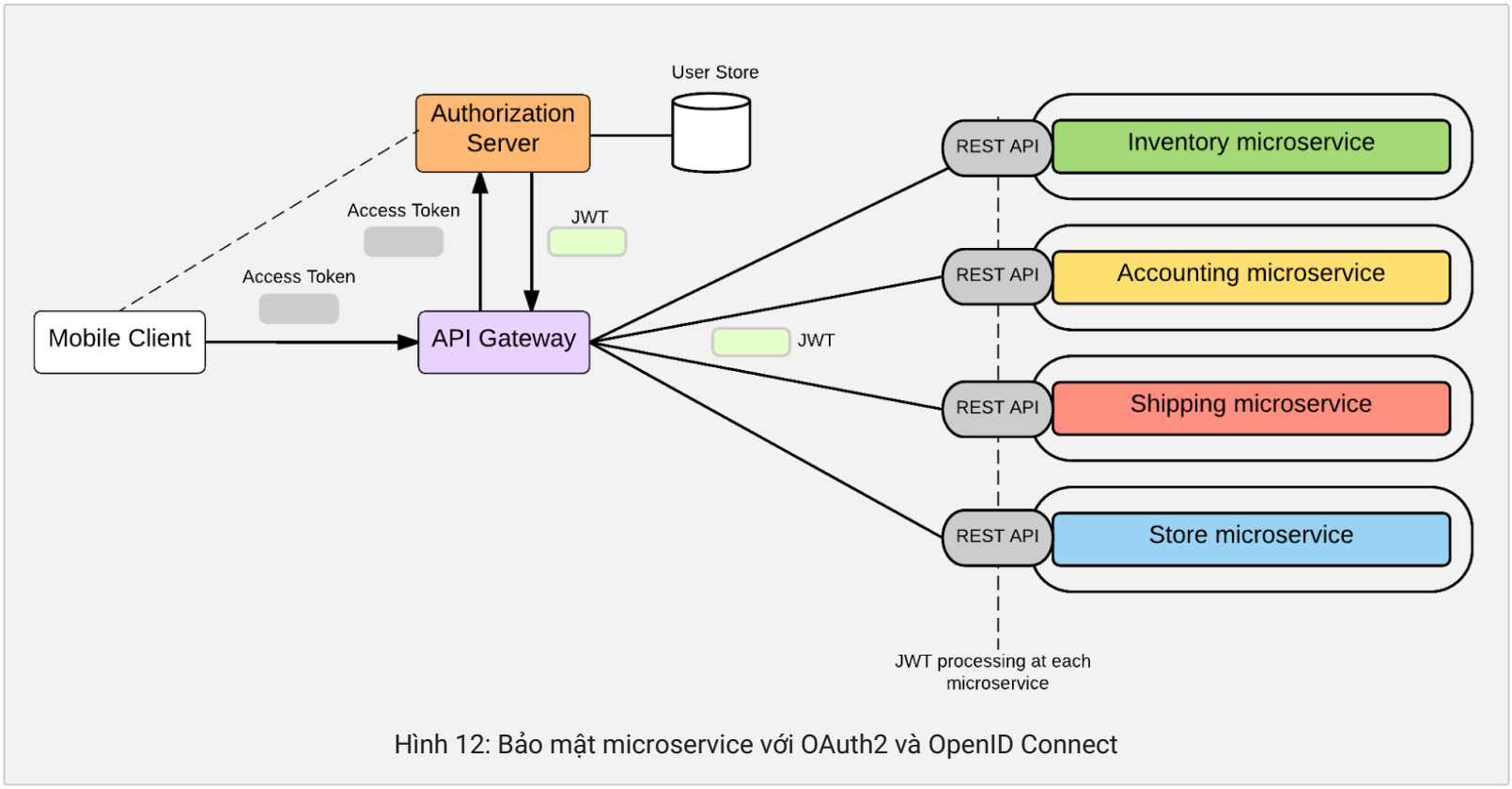
Bảo mật microservcies là một yêu cầu phổ biến trong thực tế. Trước khi nói đến bảo mật microservices, hãy nhìn lại cách chúng ta thường thực hiện bảo mật của ứng dụng monolithic.

- Bảo mật của một ứng dụng monolithic thường là về xác định xem "Ai là người gọi đến", "Người này có quyền hạn gì" và "Ứng dụng thông báo về nhưng thông tin này thế nào"
- Một component phụ trách bảo mật chung thường nằm ở đầu chuỗi xử lý yêu cầu và component sẽ trả về thông tin cần thiết

Vậy chúng ta có thể chuyển đổi trực tiếp mô hình này sang kiến trúc microservices không? Có nhưng điều này yêu cầu phát triển bảo mật ở từng microservices và kết nối với một cơ sở trung tâm về người dùng để lấy thông tin cần thiết. Đây là một cách khá rối rắm. Thay vào đó, chúng ta có thể tận dụng các API bảo mật tiêu chuẩn như OAuth2 và OpenID Connect để tìm một giải pháp tốt hơn cho vấn đề bảo mật. Trước khi nói sâu hơn, chúng ta sẽ tóm tắt mục tiêu của mỗi tiêu chuẩn và cách sử dụng chúng.

- OAuth2 - một phương thức chứng thực kiểu ủy quyền. Client xác thực với server cấp quyền (authorization server) và nhận một token gọi là "Access token". Access token không chứa bất kì thông tin gì về client. Nó chỉ là một tấm vé tham chiếu đến thông tin người dùng mà server cấp quyền có thể truy xuất đến. Do đó, đây cũng được gọi là token kiểu tham chiếu "by-reference token" và an toàn để sử dụng trên mạng lưới mở và internet.ư
- OpenID Connect hoạt động tương tự OAuth nhưng ngoài Access Token, server cấp quyền còn phát một ID token chứa thông tin người dùng. Token này thường dạng JWT (JSON Web Token) và được kí bởi server cấp quyền. JWT token do đó được coi là token kiểu tham trị "by-value token" bởi vì nó chức thông tin về người dùng và có thể trở nên không an toàn.

Bây giờ, hãy xem xét những tiêu chuẩn này có thể bảo mật microservices trong ví dụ bán hàng của chúng ta như thế nào.



Hình 12: Bảo mật microservice với OAuth2 và OpenID Connect

Như hình trên, các bước chính để thực hiện bảo mật microservices gồm:

- Chuyển việc xác thực cho server dạng OAuth và OpenID Connect (Authorization se
- Sử dụng API Gateway để có một điểm đầu vào duy nhất cho yêu cầu từ clients
- Client kết nối với server cấp quyền và nhận Access token. Sau đó gửi token này đến
- Dịch token tại Gateway - API Gateway lấy ra access token và gửi đến server cấp qu
- Gateway truyền JWT cùng với yêu cầu đến microservices
- JWT chứa thông tin người dùng cần thiết để lưu user sessions,....
- Ở mỗi lớp microservice, bạn có thể có một component xử lý JWT, việc này khác đơn giản

Design for Failures - Thiết Kế Cho Thất Bại

Kiến trúc microservices tăng khả năng xảy ra thất bại tại microservices. Một microservice có thể sập do một vấn đề mạng, nguồn tài nguyên bị thiếu hụt hoặc không sẵn sàng,... Một microservice không đáp ứng lại không nên dẫn đến toàn bộ ứng dụng microservices sập. Do đó, mỗi microservices nên có khả năng chịu lỗi và hồi phục khi có thể. Người dùng phải có trải nghiệm tốt, nhẹ nhàng và không nhận ra lỗi bên trong hệ thống.

Thêm nữa, vì các dịch vụ có thể gặp lỗi bất cứ lúc nào, việc phát hiện (giám sát thời gian thực) lỗi nhanh chóng và nếu có thể tự động phục hồi dịch vụ là cực kì quan trọng.

Có một vài kiểu mẫu xử lý lỗi trong microservices phổ biến.

Circuit Breaker

Khi có lời gọi từ bên ngoài đến một microservice, bạn có thể cấu hình một component giám sát lỗi với mỗi lời gọi. Component này đếm số yêu cầu thành công và thất bại, khi lỗi đạt một giới hạn nhất định thì component này sẽ dừng hoạt động của service (ngắt giao mạch).

Cách này hữu dụng để tránh tiêu tốn tài nguyên không cần thiết, yêu cầu bị chậm vì timeouts, cách này cũng giúp giám sát hệt thống.

Bulkhead

Vì một ứng dụng kiến trúc microservices bao gồm nhiều microservices, một servic
hưởng đến phần còn lại. Kiểu mẫu bulkhead tách biệt các phần của ứng dụng để l

Timeout

Kiểu mẫu timeout là một quy trình cho phép dừng một yêu cầu khi thời gian chờ đợ
hình khung thời gian tùy ý.

Vậy khi nào chúng ta sử dụng những kiểu mẫu này? Trong phần lớn trường hợp, những kiểu mẫu này được áp dụng ở lớp Gateway. Khi microservices không đáp lại hay không sẵn sàng, ở cấp Gateway chúng ta có thể quyết định có nên gửi yêu

cầu đến microservices sử dụng circuit breakers hay timeout. Hơn nữa, sử dụng bulkhead ở Gateway khá quan trọng vì đây là một điểm đầu vào cho tất cả yêu cầu từ clients, nên nếu lỗi một service cũng không ảnh hưởng đến việc gửi yêu cầu đến các services khác.

Thêm vào đó, Gateway có thể được sử dụng như một điểm trung tâm ta có thể lấy thông tin về trạng thái và theo dõi các microservices hoạt động.

Microservices, Tích Hợp Cho Doanh Nghiệp, Quản Lý API và hơn thế nữa

Chúng ta đã trao đổi các đặc tính của kiến trúc Microservices và cách bạn có thể thực hiện chúng. Tuy nhiên, bạn nên lưu ý rằng Microservices toa thuốc chữa bách bệnh. Việc áp dụng mù quáng những khái niệm đang nổi sẽ không xử lý vấn đề thực tế của bạn. Nếu bạn đã đọc toàn bộ bài này thì bạn sẽ thấy Microservices có nhiều lợi ích và chúng ta nên tận dụng. Tuy nhiên, bạn cần hiểu rằng mong chờ microservices giải quyết tất cả các vấn đề là không thực tiễn.

Mong rằng các bạn đã có một cái nhìn rõ hơn về Microservices.

Bài viết được dịch từ Microservices in Practice: From Architecture to Deployment

PHP

03 PHP/NodeJS Developers

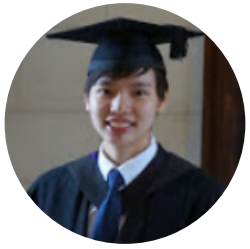
GUU JSC *Ha Noi* *Up to $1,000*

PHP NodeJS

Microservices một cách dễ hiểu
02/02/2018 Nguyễn Thành Long

BLOG HOME

Cấu trúc thư mục một project sử dụng Go-Micro
29/03/2018 Techmaster team



*Bởi **Ngô Thùy Linh***

Chuyên lập trình web. Tốt nghiệp đại học University College London (UCL) vương quốc .
Yêu thích đi du lịch và đọc truyện trinh thám ngoài thời gian dành cho đam mê lập trình.

Chủ sở hữu website

Công ty TNHH TechMaster Vietnam Ltd

Số ĐKDN: 0105392153

Ngày cấp: 4-7-2011

Nơi cấp: Sở kế hoạch - đầu tư Hà nội

Người đại diện pháp luật: Lê Minh Thu

Chịu trách nhiệm nội dung: Trịnh Minh Cường

Thông tin chung

Thông tin trung tâm

Giảng viên

Quy định

Hướng dẫn mua khóa học

Hoàn trả - Ưu đãi học phí

Chính sách bảo vệ thông tin khách hàng

Contact

Mr. Cường: 090 220 9011

cuong@techmaster.vn

Ms. Huyền : 0168 309 7229

huyen@techmaster.vn

Ms. Mai Anh: 096 247 1397

maianh2503@gmail.com

Địa chỉ

Số 14, ngõ 4, Nguyễn Đình Chiểu, Hai Bà Trưng, Hà Nội

Giờ mở cửa: **từ 9:30 - 18:00**