# Xây dựng Microservices: Sử dụng cổng kết nối API (API Gateway)

17/10/2015 / Bởi **Techmaster Team** / trong **Microservice**



Bài viết đầu tiên trong loạt 7 phần về thiết kế, xây dựng và triển khai các dịch vụ nh
hình kiến trúc Microservice. Bài viết đã bàn về những lợi ích và hạn chế khi sử dụn
qua được sự phức tạp của microservices, để nó trở thành một sự lựa chọn lý tưởn

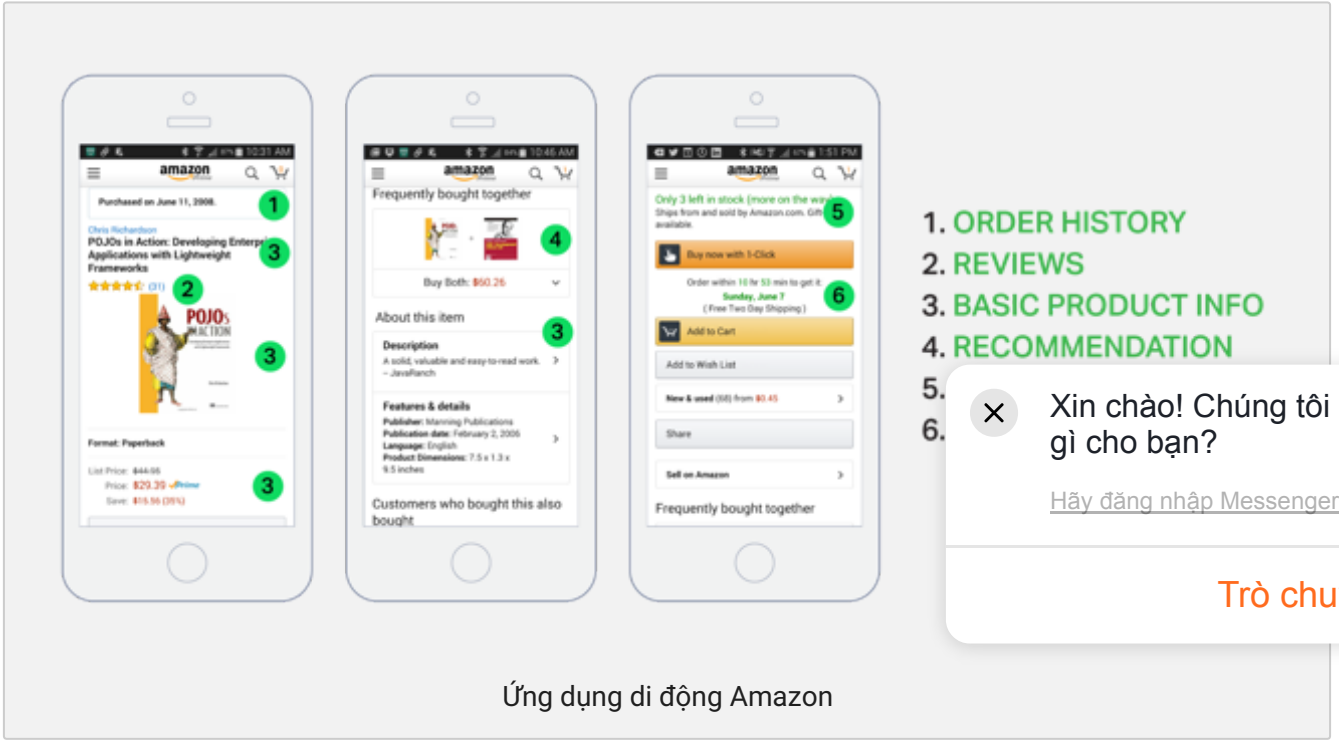
Khi bạn lựa chọn việc xây dựng ứng dụng dựa trên tập hợp các dịch vụ nhỏ, bạn cần quyết định cách các Client (web client, mobile client...) của ứng dụng đó tương tác với các microservice. Với một ứng dụng đơn khối (monolithic - thường đã được nhân bản ra nhiều nơi và cân bằng tải) thì các Client chỉ đơn thuần là tập hợp các thiết bị đầu cuối. Tuy nhiên, trong kiến trúc microservices, mỗi microservice sẽ tương tác với một tập hợp các thiết bị đầu cuối được phân phân bổ

một cách tinh tế (fine-grained). Bài viết này sẽ xem xét cách thức Client và ứng dụng tương tác với nhau, từ đó đề xuất cách tiếp cận sử dụng API Gateway.

## Giới thiệu

Hãy tưởng tượng bạn đang phát triển một ứng dụng chạy trên Mobile (mobile client) cho hệ thống mua sắm trực tuyến. Và rất có thể bạn cần làm một trang thông tin cho sản phẩm, hiển thị thông tin chi tiết về bất kỳ loại sản phẩm nào được cung cấp.

Để lấy ví dụ, sơ đồ dưới đây thể hiện những gì bạn sẽ nhìn thấy khi di chuyển trên màn hình thông tin chi tiết của sản phẩm trên ứng dụng di động của Amazon trên nền Android




Ứng dụng di động Amazon

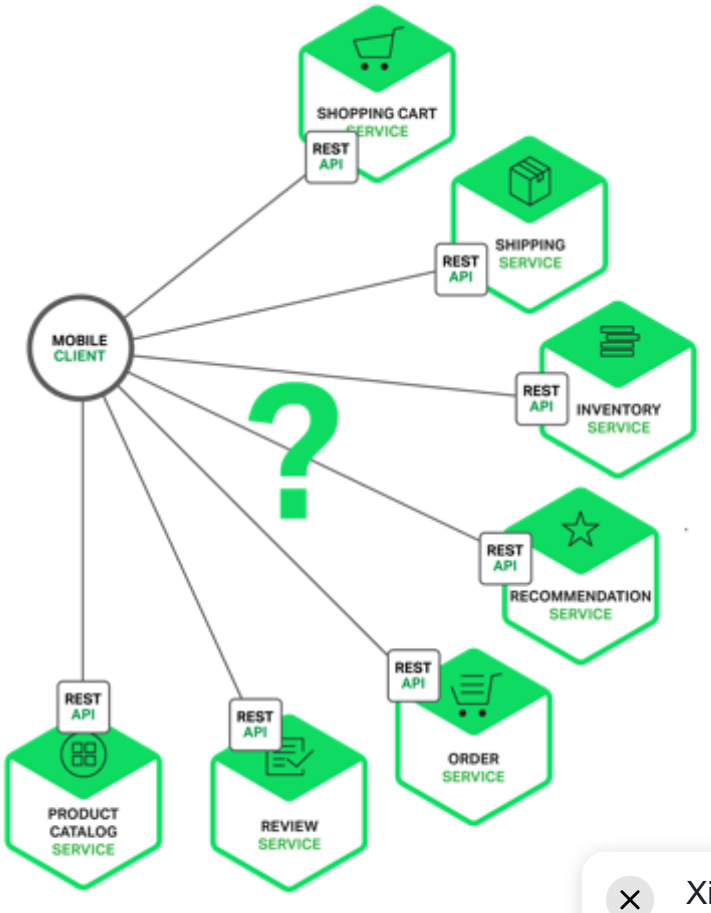
Mặc dù là một ứng dụng di động, trang thông tin sản phẩm lại hiển thị rất nhiều thông tin. Ví dụ trên không chỉ có các thông tin cơ bản của sản phẩm (tên gọi, mô tả và giá cả) mà nó còn hiển thị:

- Số lượng mặt hàng trong giỏ.
- Lịch sử đơn hàng.
- Phản hồi từ khách hàng.
- Cảnh báo khi sắp hết hàng.
- Các tùy chọn khi vận chuyển
- Một loạt gợi ý mua hàng bao gồm các mặt hàng thường được mua kèm với mặt hàng đang xem, các mặt hàng khác được mua bởi khách hàng đã mua sản phẩm này, và các mặt hàng khác được xem bởi người tiêu dùng đã mua sản phẩm.
- Các tùy chọn thanh toán.

Khi sử dụng kiến trúc khối đồng nhất (monolithic), ứng dụng di động (mobile client) sẽ nhận các dữ liệu này bằng cách gọi duy nhất một yêu cầu qua REST (GET api.company.com/productdetails/*productId*) đến ứng dụng. Cân bằng tải sẽ đưa yêu cầu này tới 1 trong nhiều bản sao của ứng dụng. Sau đó ứng dụng truy vấn vào nhiều bảng cơ sở dữ liệu khác nhau và trả về phản hồi cho Client.

Trái lại, khi sử dụng kiến trúc Microservice, dữ liệu hiển thị trên trang mua hàng thuộc về rất nhiều dịch vụ nhỏ khác nhau. Dưới đây là một vài microservice sở hữu các dữ liệu được hiển thị trên bảng thông tin chi tiết hàng hóa như ví dụ:

- Dịch vụ giỏ hàng – Số lượng mặt hàng trong giỏ.
- Dịch vụ đặt hàng – Lịch sử đơn hàng.
- Dịch vụ tổng kê (catalog) – Thông tin cơ bản của mặt hàng, ví dụ như tên hàng, hìn
- Dịch vụ đánh giá – Phản hồi từ khách hàng.
- Dịch vụ kho hàng – Cảnh báo khi sắp hết hàng.
- Dịch vụ vận chuyển – Hình thức vận chuyển, hạn gửi, và các chi phí sẽ được tính ri dịch vụ vận chuyển.
- Dịch vụ khuyến nghị – Các sản phẩm được khuyến nghị.




Để quyết định xem Mobile client của chúng ta làm thế nào để truy cập được tới cá
lựa chọn có thể dưới đây.

## Giao tiếp trực tiếp từ Client tới Microservice

Trên lý thuyết, mỗi Client có thể trực tiếp đưa ra nhiều yêu cầu tới một trong các microservice. Mỗi một microservice thường có một điểm truy cập đầu cuối công khai ( **https://*serviceName*.api.company.name**). Đường dẫn này sẽ chỉ đến bộ cân bằng tải của  microservice, có nhiệm vụ phân phối các lệnh yêu cầu tới các bản sao đang rảnh rỗi của microservice. Để nhận những thông tin về sản phẩm, mobile client sẽ phải gửi yêu cầu tới mỗi dịch vụ được liệt kê ở trên.

Không may, cách thức này vẫn còn nhiều hạn chế và khó khăn. Một trong số đó là tính không đồng nhất giữa nhu cầu của Client và các API phân tán được cung cấp bởi mỗi  microservice. Client trong ví dụ trên phải gửi đến 7 yêu cầu riêng biệt. Trong những phần mềm phức tạp hơn có thể phải gửi nhiều hơn. Ví dụ hoạt động của Amazon miêu tả hàng trăm dịch vụ khác nhau cùng tham gia việc hiển thị trang thông tin mua hàng. Trong khi một Client có thể dễ dàng gửi yêu cầu thông qua mạng LAN, thì việc này có thể không hiệu quả nếu thực hiện qua mạng Internet công cộng, thậm khí là vô cùng khó khăn nếu thực hiện qua mạng di động. Cách tiếp cận này khiến cho Code của Client trở nên phức tạp hơn.

Một vấn đề là khi Client gọi trực tiếp tới các microservice mà trong số đó sử dụng các giao thức không thân thiện với Web. Dịch vụ này có thể sử dụng giao thức Thrift binary RPC (một giao thức được Apache đề xướng) trong khi dịch vụ khác sử dụng giao thức tin nhắn AMQP (Một dạng Message Queue phổ biến hiện này). Không giao thức nào trong hai ví dụ trên là thuần túy về web hoặc thân thiện với tưởng lửa và sử dụng tốt trong nội tại của nó. Bên ngoài phạm vi tường lửa, một ứng dụng tốt nên sử dụng các giao thức như HTTP và WebSocket.

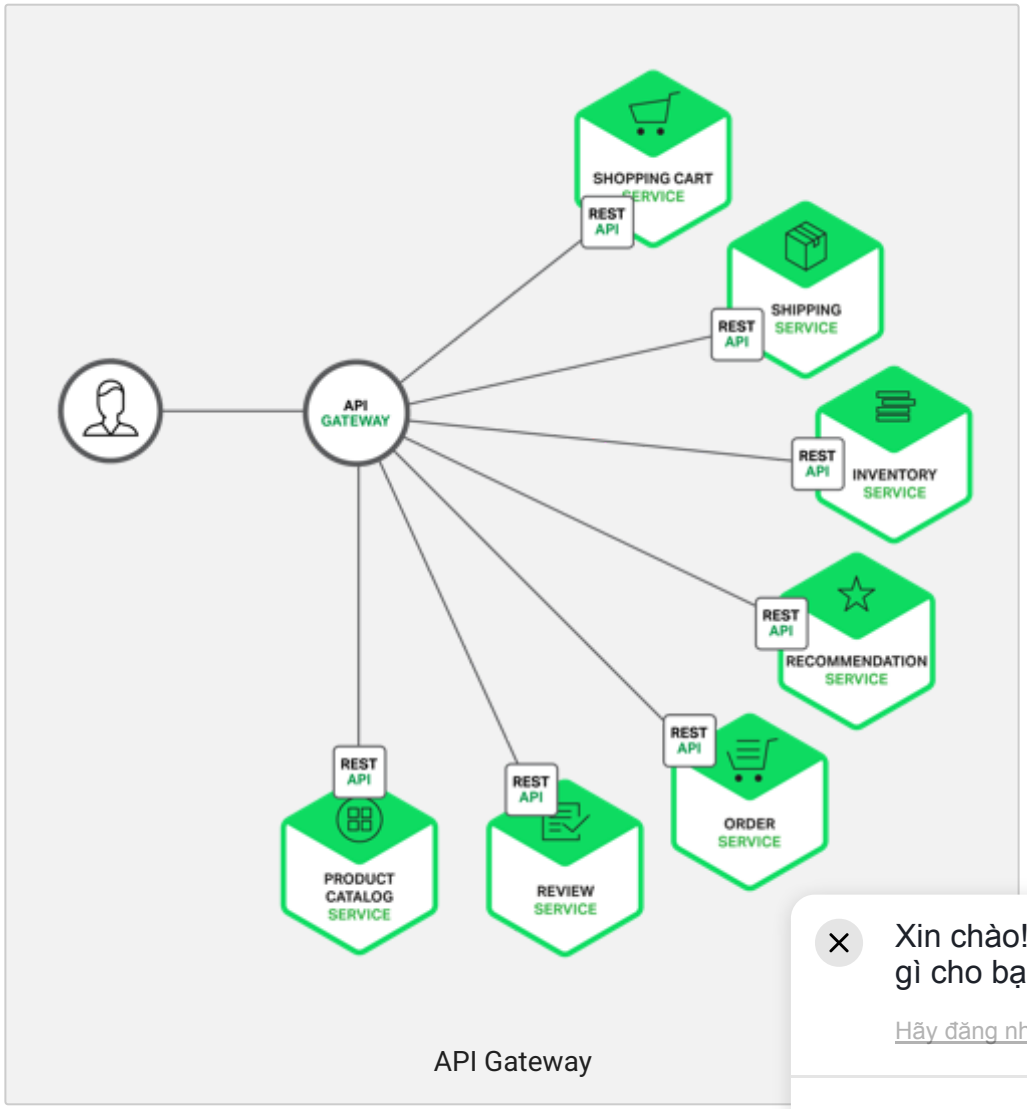
Nhược điểm khác của cách tiếp cận này là sự khó khăn trong việc tái cấu trúc (refactor) các Microservice. Theo thời gian, chúng ta sẽ muốn thay đổi cách hệ thống phân chia dịch vụ. Ta có thể hợp nhất hai dịch vụ hoặc phân chia một dịch vụ thành hai hoặc nhiều dịch vụ nhỏ hơn. Nếu Client giao tiếp trực tiếp với các dịch vụ [illegible] vô cùng khó khăn.

Những vấn đề được liệt kê trên khiến cho việc giao tiếp trực tiếp Client và các micr[illegible]

## Sử dụng cổng kết nối API (API Gateway)

Cổng kết nối API là phương pháp tiếp cận tốt hơn rất nhiều. Một cổng kết nối API là một máy chủ truy xuất duy nhất vào hệ thống. Nó cũng tương tự như mẫu thiết kế Facade dựa trên thiết kế hướng đối tượng. Cổng kết nối API che giấu đi thông tin kiến trúc hệ thống nội bộ và nó cung cấp các API tùy chỉnh cho mỗi Client. Cổng kết nối API còn có trách nhiệm xác thực, giám sát, cân bằng tải, caching, định hình yêu cầu và quản lí thông tin, xử lí phản hồi tĩnh.

Sơ đồ minh họa dưới đây cho thấy một API Gateway phù hợp trong kiến trúc ứng dụng.

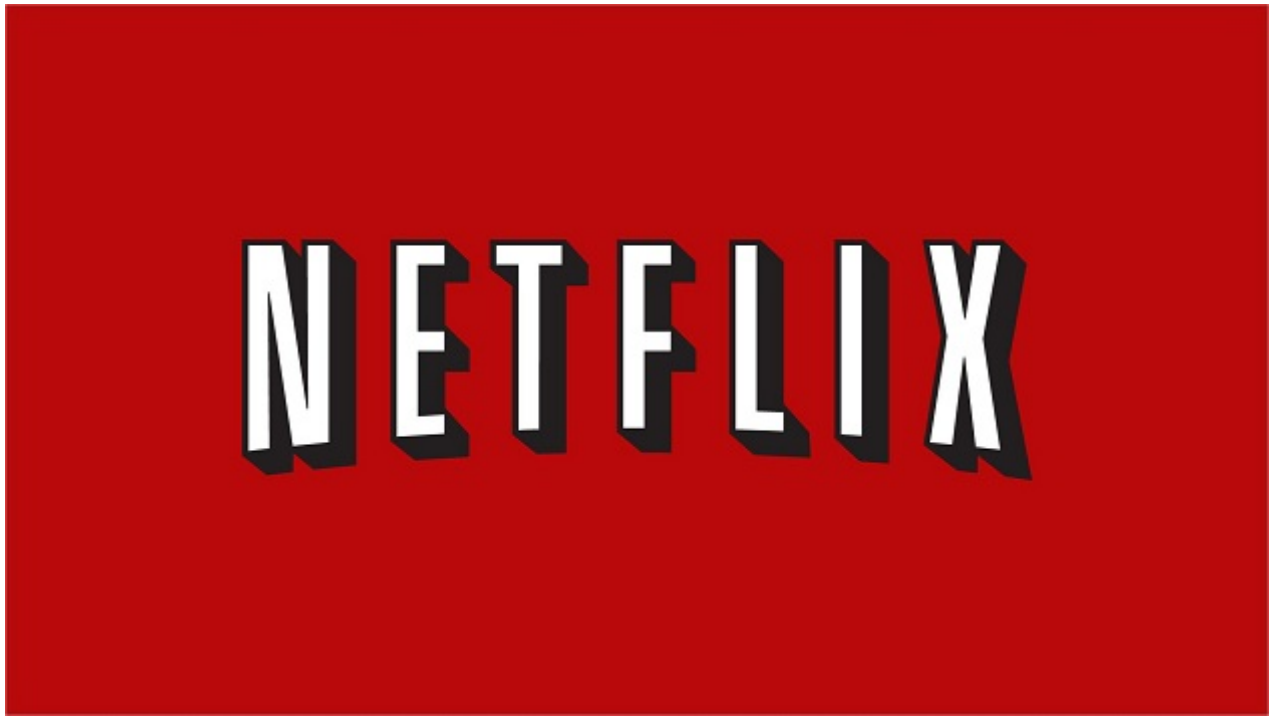



API Gateway

Cổng kết nối API làm nhiệm vụ định tuyến các yêu cầu, kết hợp và chuyển đổi các giao thức. Tất cả yêu cầu từ Client đều đi qua cổng kết nối API. Sau đó cổng kết nối API định tuyến các yêu cầu này tới microservice phù hợp. Cổng kết nối API Gateway sẽ xử lý một yêu cầu người dùng bằng cách gọi đến một loạt microservice rồi tổng hợp các kết quả. Nó có thể chuyển đổi giữa các giao thức web như HTTP, WebSocket và các giao thức nội bộ không thân thiện với web.

Cổng kết nối API cũng có thể cung cấp API tùy chỉnh cho mỗi Client. Nó cung cấp API "thô" (coarse-grained) cho mobile client. Cùng xem xét lại ví dụ về kịch bản trang thông tin chi tiết sản phẩm. Cổng kết nối API có thể cung cấp kết nối cuối (**/productdetails?productid=xxx**) cho phép mobile client nhận tất cả thông tin chi tiết của sản phẩm chỉ với một lệnh yêu cầu duy nhất. Cổng kết nối API xử lí lệnh yêu cầu bằng cách truy vấn đến các dịch vụ khác nhau như dịch vụ thông tin sản phẩm, dịch vụ khuyến nghị, dịch vụ đánh giá… rồi sau đó tổng hợp lại kết quả.

Cổng kết nối API của Netflix (trang cung cấp Video trực tuyến) là ví dụ điển hình. Dịch vụ streaming của Netflix có mặt trên hàng trăm các loại hình thiết bị khác nhau như TV, Smart-box, điện thoại thông minh, hệ thống chơi game, máy tính bảng, v.v… Ngay từ lúc đầu, tham vọng của Netflix là cung cấp một API đa nền tảng (one size fits all) cho dịch vụ streaming. Netflix nhận ra ý tưởng của họ hoạt động không hiệu quả vì sự đa dạng các thiết bị và tham vọng độc nhất của họ. Ngày nay, Netflix sử dụng một cổng kết nối API cung cấp API tùy chỉnh (API tailored) với các thiết bị khác nhau bằng cách chạy các đoạn Code phù hợp với từng bộ chuyển đổi thiết bị (device adapter). Bộ chuyển đổi xử lý các lệnh yêu cầu bằng cách truy vấn từ sáu đến bảy dịch vụ "back-end". Mỗi ngày cổng giao tiếp API của Netflix xử lý đến hàng tỉ yêu cầu.




## Lợi ích và nhược điểm của cổng kết nối API (API Gateway)

Việc sử dụng cổng kết nối API có lợi ích và nhược điểm riêng. Ích lợi lớn nhất khi s
đi cấu trúc bên trong của ứng dụng. Thay vì truy vấn đến các dịch vụ cụ thể, người
nối. Cổng kết nối API cung cấp API phù hợp với từng Client. Việc làm này giảm thiể
Client và ứng dụng, tối giản hóa mã nguồn phía Client.

Cổng kết nối API cũng có một vài nhược điểm. Nó là một trong các thành phần thi
khai và quản lý. Rủi ro khác là cổng kết nối API trở thành nút thắt cổ chai khi phát triển hệ thống (bottle-neck). Các Developer cần phải cập nhập cổng kết nối API để cung cấp cho các kết nối đầu cuối của microservice. Việc đảm bảo tiến trình cập nhập cổng kết nối API với dung lượng nhẹ nhất có thể vô cùng quan trọng. Nếu không thì Developer sẽ buộc

phải đợi đến lượt mới được cập nhập. Mặc dù có nhược điểm, nhưng hầu hết các ứng dụng thực tế sử dụng cổng kết nối API.

## Triển khai một cổng kết nối API

Sau khi đã xem qua lý do, điểm mạnh và yếu (trade-off) khi sử dụng cổng kết nối API, hãy cùng xem xét những vấn đề thiết kế API mà bạn nên cân nhắc.

**Hiệu năng và khả năng mở rộng.**

Chỉ một lượng nhỏ các công ty có tầm cỡ như Netflix mới phải xử lí đến hàng tỉ yêu cầu mỗi ngày. Dù vậy, với các ứng dụng, hiệu năng và khả năng mở rộng của cổng kết nối API thường vô cùng quan trọng. Vì vậy, điều này hoàn toàn phù hợp với việc xây dựng cổng kết nối API trên một nền tảng hỗ trợ xử lí bất đồng bộ và cơ chế non-blocking I/O. Có rất nhiều các công nghệ khác nhau được sử dụng để thực hiện mở rộng cổng kết nối API. Trong JVM (Java Virtual Machine) bạn có thể sử dụng các Framework dựa trên cơ chế NIO (Non-blocking IO) như Netty, Vertx, Spring Reactor hoặc Jboss Undertow. Một lựa chọn phổ biến không dùng đến JVM là Node.js, nền tảng xây dựng trên Chrome's JavaScript engine (V8 Engine). Lựa chọn khác, bạn có thể dùng NGINX Plus. (Một phút quảng cáo: NGINX Plus cung cấp hệ thống web server và proxy hoàn thiện, có khả năng mở rộng, hiệu năng cao, dễ dàng triển khai, tùy biến và lập trình. Công nghệ này quản lí tính xác thực, truy cập điều khiển, cân bằng tải lệnh yêu cầu, phản hồi đệm và cung cấp dịch vụ giám sát ứng dụng).

**Sử dụng mô hình lập trình tương tác.**

Cổng kết nối API xử lý một số yêu cầu bằng cách định tuyến (routing) chúng đến d
yêu cầu còn lại bằng cách truy vấn đến một loạt các dịch vụ back-end và tổng hợp
lệnh yêu cầu về chi tiết sản phẩm, các yêu cầu đến các dịch vụ back–end hoàn toà
gian phản hồi, cổng kết nối API nên thực hiện các yêu cầu độc lập trong cùng một
thuộc với nhau. Cổng kết nối API đầu tiên cần phải rà soát yêu cầu bằng cách gọi đến các dịch vụ xác thực trước khi định tuyến chúng đến dịch vụ back–end. Tương tự, để lấy thông tin về sản phẩm trong danh sách mong muốn của khách hàng,

cổng kết nối API trước hết phải nhận được hồ sơ khách hàng bao gồm thông tin, và sau đó lấy thông tin của mỗi sản phẩm. Một ví dụ thú vị về phân tán API là Netflix Video Grid.

Viết mã API thành phần sử dụng các callback bất đồng bộ truyền thống sẽ đưa bạn xuống địa ngục. Mã nguồn trở nên rối rắm, khó hiểu và thường xuyên gặp lỗi. Phương thức hiệu quả hơn là bạn viết mã nguồn cho cổng kết nối API dạng "Declarative style" sử dụng phương thức tương tác. Ví dụ về tương tác trừu tượng với Scala thì có Future, Java8 có CompletableFuture, JavaScript có Promise (các cơ chế xử lí bất đồng bộ). Ngoài ra còn có công nghệ Reactive Extensions (còn được gọi là Rx hoặc ReactiveX), ban đầu được phát triển bởi Microsoft cho nền tảng .NET. Netflix sau đó tạo ra RxJava cho JVM mục đích để sử dụng cho cổng kết nối API của họ. Đồng thời còn có RxJS trên JavaScript, chạy được trên cả trình duyệt và Node.js. Áp dụng phương pháp tương tác sẽ giúp bạn viết được một cổng giao tiếp API đơn giản mà hiệu quả.

**Truy vấn dịch vụ.**

Một ứng dụng dựa trên nền tảng microservices là một hệ thống phân tán và cần phải sử dụng cơ chế giao tiếp liên quá trình (inter-process communication mechanism). Có hai phong cách giao tiếp liên quá trình. Lựa chọn đầu tiên sử dụng cơ chế lập trình bất đồng bộ, dựa trên cơ chế truyền tin (messaging-based). Một số triển khai sử dụng các phần mềm truyền tin trung gian (message broker) như JMS hoặc AMQP. Số còn lại, ví dụ như Zeromq thì không phải sử dụng trung gian mà giao tiếp thẳng với các dịch vụ. Phong cách khác của giao tiếp liên quá trình là cơ chế đồng bộ như HTTP hoặc Thrift. Một hệ thống thường sử dụng cả 2 phong cách lập trình đồng bộ và bất đồng bộ. Nó còn có thể triển khai theo nhiều hướng cho mỗi phong cách. Do đó, cổng kết nối API sẽ cần hỗ trợ nhiều cơ c

**Phát hiện dịch vụ.**

Cổng kết nối API cần biết vị trí (địa chỉ IP và cổng) của từng microservice mà nó g
thống, bạn thường phải cố định các vị trí, nhưng hiện nay, với các microservice đư
toán đám mây thì việc xác định vị trí các Microservice trở thành một việc không hề
như khi sử dụng ứng dụng truyền tin trung gian (message broker) với các vị trí được cố định thì hoàn toàn có thể chỉ ra bằng các biến môi trường của hệ điều hành. Tuy nhiên, xác định vị trí của dịch vụ ứng dụng thì không đơn giản. Dịch vụ ứng dụng được gán vị trí động và vị trí bản sao của các dịch vụ cũng được thay đổi bởi khả năng tự điều chỉnh và nâng cấp. Do đó, một cổng kết nối API, cũng như bất kì dịch vụ Client nào trong hệ thống cũng đều cần sử dụng cơ chế phát

hiện dịch vụ như: Server-Side Discovery hoặc Client-Side Discovery. Bài viết kì sau sẽ mô tả chi tiết về phát hiện dịch vụ. Hiện tại, chúng ta đồng ý rằng nếu hệ thống sử dụng Client-Side Discovery thì cổng kết nối API phải được truy vấn đến Service Registry – cơ sở dữ liệu chứa tất cả các thể hiện (instances) của microservice và vị trí của chúng.

**Xử lý lỗi từng phần.**

Một vấn đề bạn cần lưu ý là vấn đề lỗi từng phần khi thực thi một cổng kết nối API. Vấn đề này phát sinh trong tất cả các hệ thống phân tán mỗi khi một dịch vụ gọi đến một dịch vụ khác không hoạt động hoặc phản hồi rất chậm. Cổng kết nối API không bao giờ được chặn vô thời hạn khi chờ đợi một dịch vụ downstream (bị mất kết nối). Dù vậy, cách xử lí lỗi phụ thuộc vào các hoàn cảnh cụ thể như dịch vụ nào đang gặp vấn đề. Ví dụ, nếu dịch vụ gợi ý mua hàng không phản hồi, cổng kết nối API nên trả lại các giá trị còn lại của mặt hàng cho khách hàng khi nó vẫn hữu ích cho người dùng. Dịch vụ gợi ý mua hàng có thể trống hoặc được thay thế bằng danh sách tốp 10 sản phẩm đắt hàng. Dù vậy, nếu như dịch vụ về thông tin sản phẩm mà không phản hồi thì cổng kết nối API nên trả lại thông báo lỗi cho khách hàng.

Cổng kết nối API nên trả lại bộ nhớ đệm nếu nó có sẵn. Ví dụ như giá cả hàng hóa hiếm khi thay đổi, cổng kết nối API có thể trả lại giá trị bộ nhớ đệm giá trị hàng hóa trong khi Dịch vụ giá cả không hoạt động. Dữ liệu có thể lự lưu đệm bằng chính cổng kết nối API hoặc được lưu trên bộ đệm mở rộng như Redis hoặc Memcached. Bằng cách trả lại dữ liệu mặc định hoặc dữ liệu đệm, cổng kết nối API đảm bảo rằng lỗi hệ thống sẽ không ảnh hưởng nhiều đến trải nghiệm của người sử dụng.

Công nghệ Hystrix của Netflix là một thư viện tuyệt hảo để viết code gọi các dịch vụ từ xa. Hystrix lần ra những lệnh gọi vượt quá ngưỡng qui định. Nó thực thi mô hình c*ircuit breaker*, giúp cho người dùn
dịch vụ không phản hồi. Nếu tỉ lệ lỗi trong một dịch vụ vượt quá ngưỡng qui định, H
cả lệnh yêu cầu trong thời gian nhất định sẽ thất bại ngay lập tức. Hystrix giúp bạn
một lệnh yêu cầu thất bại, ví dụ như đọc từ bộ đệm hoặc trả lại giá trị mặc định. Nế
sử dụng Hystrix. Và nếu bạn sử dụng một môi trường không JVM, thì bạn nên dùn



## Tổng kết

Hoàn toàn phù hợp khi triển khai cổng kết nối API làm điểm kết nối duy nhất với hệ thống ứng dụng trên nền tảng microservices. Cổng kết nối API có nhiệm vụ định tuyến yêu cầu, phân tán dữ liệu và chuyển đổi giao thức, cung cấp cho mỗi Client một API tùy chỉnh. Cổng kết nối API còn che giấu lỗi trên cách dịch vụ back–end bằng cách trả về giá trị bộ nhớ đệm hoặc dữ liệu mặc định. Bài viết kì tới, chúng ta sẽ nói về phương thức liên lạc giữa các dịch vụ.

*Người dịch: Phạm Đức Trung, lập trình viên Java - Java Spring tại Techmaster*
*Hiệu đính: Hoàng Giang Biển*

**DevOps**
23/05/2016 Techmaster team

BLOG HOME

**Một lập trình viên nên biết 6 công nghệ cần học trong 2013**
08/11/2013 Techmaster team

## 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

## Contact

Mr. Cường: 090 220 9011

cuong@techmaster.vn

Ms. Huyền : 0168 309 7229

huyen@techmaster.vn

## Địa chỉ

*Số 14, ngõ 4, Nguyễ*

Giờ mở cửa: **từ 9:30 - 18:00**

Hoàn trả - Ưu đãi học phí

Chính sách bảo vệ thông tin khách hàng

Ms. Mai Anh: 096 247 1397

maianh2503@gmail.com