# Kĩ thuật IPC (Inter Process Communication - liên lạc giữa các process) trong kiến trúc Microservice

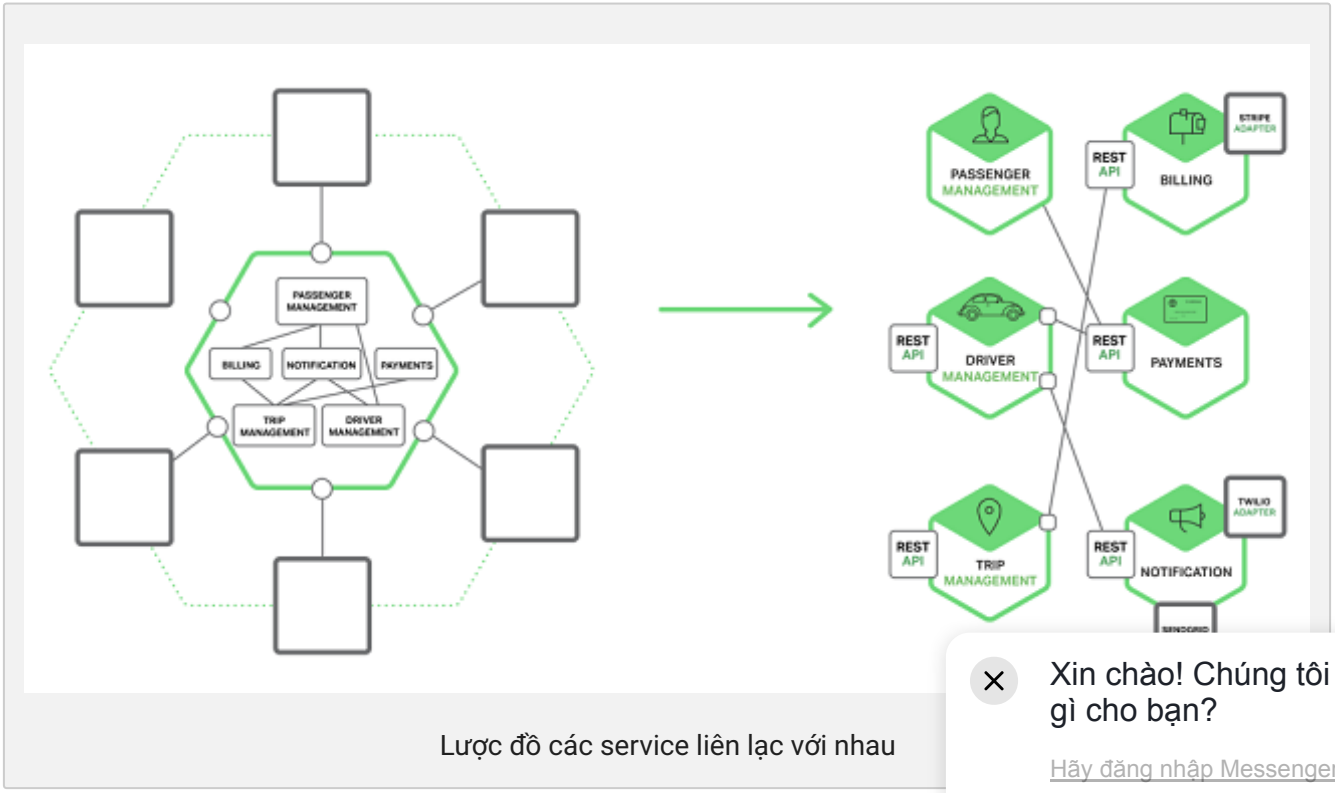
26/10/2015 / Bởi Techmaster Team / trong Micro



Đây là phần tiếp theo của series xây dựng ứng dụng với kiến trúc Microservice. Tô
trước để bạn đọc tiện theo dõi. Kì đầu tiên giới thiệu mô hình kiến trúc Microservic
sử dụng nó. Kì thứ 2 dành cho API Gateway, cầu nối giữa người dùng và các ứng dụng Microservices. Và ở phần 3 này, chúng ta sẽ tìm hiểu về sự tương tác giữa các dịch vụ trong hệ thống sử dụng mô hình này.

## Giới thiệu chung

Trong một ứng dụng đơn khối (monolithic), các components tương tác với nhau thông qua việc truy vấn method và function. Ngược lại, các ứng dụng Microservices là một hệ thống phân tán chạy trên nhiều máy. Mỗi service instance là một quá trình đặc trưng. Do đó, sự tương tác giữa các services cần đến kỹ thuật IPC, viết tắt của **inter-process communication**: hành động trao đổi dữ liệu giữa các tiến trình riêng biệt, sử dụng giao thức kết nối. Tham khảo thêm bài "Sự khác nhau giữa Process và Thread"




Lược đồ các service liên lạc với nhau

Có lẽ tôi sẽ để dành IPC cho phần sau, trước tiên, chúng ta cùng xem xét các vấn đ

## Cách tương tác

Khi lựa chọn kĩ thuật IPC cho một dịch vụ, ta cần hiểu các services tương tác với nhau như thế nào. Có vô số cách tương tác giữa client-service. Ta có thể chia chúng thành 2 nhóm theo 2 cách:

Cách 1: tương tác/ kết nối 1-1 hoặc 1-nhiều – gọi tắt là 1-n:

- 1-1: mỗi request từ client được xử lí bởi 1 service duy nhất
- 1-nhiều: mỗi request từ client được xử lí bởi nhiều service instance

Cách 2: kết nối đồng bộ (synchromous) hoặc bất đồng bộ (asynchronous)

- Với synchronous: clients gửi request rồi chờ phản hồi, trong lúc chờ có thể không chạy tiếp
- Với asynchronous: clients cũng gửi request rồi chạy tiếp, khi có kết quả (response) thì xử lí

Chúng ta có thể theo dõi tương quan các kiểu kết nối/tương tác qua bảng sau:

| | **1-1** | **1-n** |
|---|---|---|
| **Synchronous** | Request/response | |
| **Asynchronous** | Thông báo (notification)<br>Request/async response | Publish/subscribe<br>Publish/async responses |



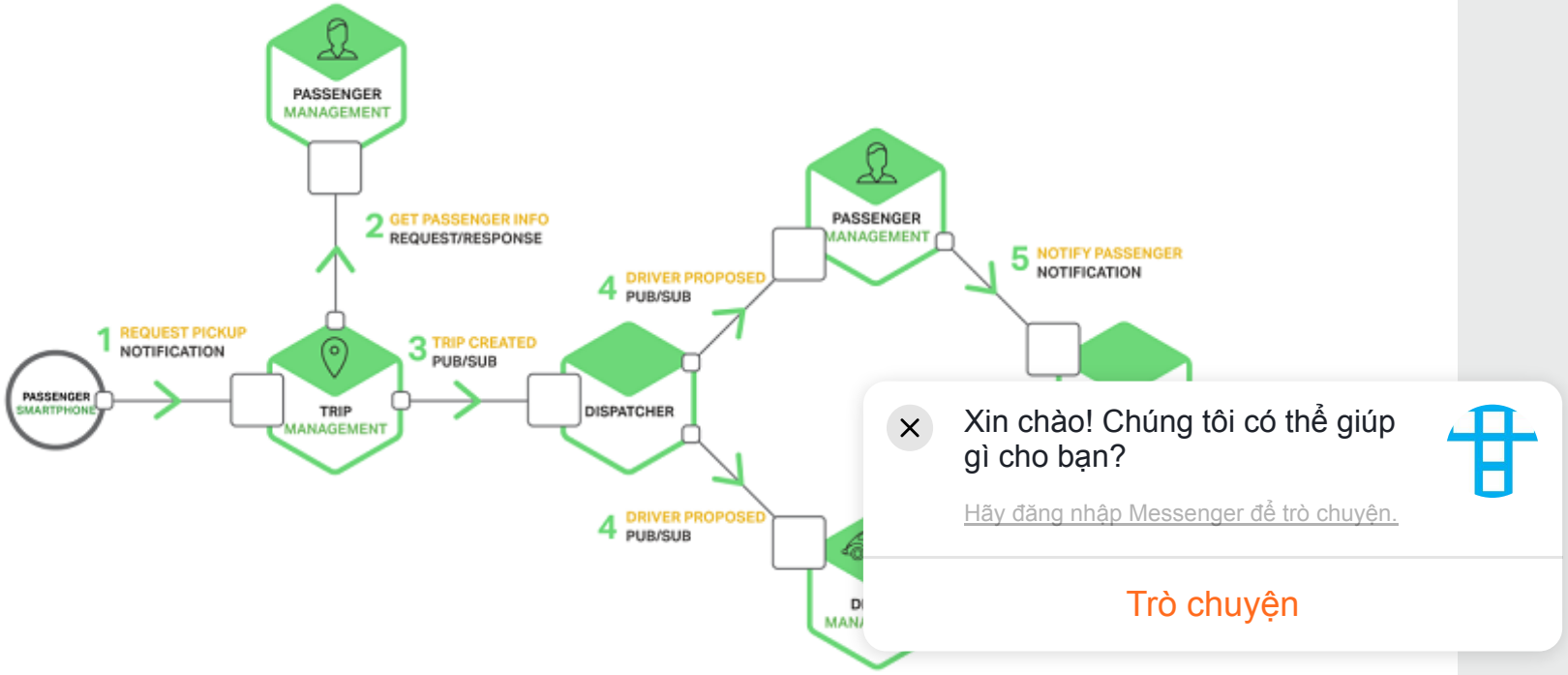
Với tương tác 1-1, ta có các hình thức:

- Request/response (yêu cầu/ hồi đáp): client gửi yêu cầu tới dịch vụ và đợi phản hồi dụng thread-based. Thread nào đang chạy yêu cầu (request) sẽ bị chiếm dụng tron
- Notification (Thông báo): client gửi yêu cầu (request) tới service nhưng không đợi p nào được gửi lại.
- Request/ async response (yêu cầu – hồi đáp bất đồng bộ): quá trình client gửi yêu cầu tới service và nhận phản hồi diễn ra không đồng thời. Client được thiết kế để hiểu rằng phản hồi sẽ không đến tức thì, do đó, client không bận trong suốt quá trình đợi phản hồi (trái ngược với hình thức request/response)

Với tương tác 1-n, ta có:

- Công bố/theo dõi (Publish/subscribe): client bắn ra một tin nhắn/thông báo, các services có ”hứng” với thông báo đó sẽ xâu xé nó.
- Công bố/ hồi đáp bất đồng bộ (Publish/async responses): client đưa ra yêu cầu (bằng tin nhắn hoặc thông báo) rồi đợi hồi đáp từ các services có “hứng”

Đa phần các service (dịch vụ) là tổ hợp của 3 phương pháp tương tác. Một tỉ lệ nhỏ chỉ cần một kĩ thuật IPC là đủ. Số còn lại thì cần sự kết hợp các kĩ thuật IPC. Biểu đồ sau cho thấy quá trình tương tác giữa ứng dụng và người dùng khi người dùng đặt một chuyến taxi:



Ta thấy một tổ hợp thông báo (notification) + yêu cầu/hồi đáp (request/response) + công bố/theo dõi (publish/subscribe).

Khách khởi động ứng dụng và thao tác trên smartphone, smartphone gửi 1 notification (thông báo) đến bộ phận quản lý hành trình (Trip Management) để gọi một chuyến taxi. Bộ phận quản lý (Trip Management) truy vấn bộ phận dịch vụ khách hàng (Passenger Service) thông qua hình thức request/response (yêu cầu/hồi đáp) để xác nhận trạng thái của tài khoản vừa gửi request (ở ví dụ này là active). Sau đó, bộ phận quản lý khởi tạo 1 chuyến đi mới và thông báo cho các bộ phận khác (bao gồm cả dispatcher) thông qua publish/subscribe (công bố/theo dõi). Bộ Dispatcher sẽ xác định một taxi tương ứng.

Trên đây là các phương thức tương tác, bây giờ chúng ta hãy bàn về APIs.

## Định nghĩa các API

Một API của dịch vụ là một hợp đồng giữa dịch vụ đó và các client của nó. Bất kể sự lựa chọn của bạn về cơ chế IPC, điều quan trọng là phải xác định chính xác API của dịch vụ bằng cách sử dụng một số loại ngôn ngữ định nghĩa giao diện (IDL - Interface Definition Language). Thậm chí có những lý lẽ tốt cho việc sử dụng một cách tiếp cận API đầu tiên (API-first approach) để định nghĩa các dịch vụ. Bạn bắt đầu phát triển một dịch vụ bằng cách viết định nghĩa giao diện và xem xét nó với các nhà phát triển client. Đó là chỉ sau khi lặp qua định nghĩa API mà bạn thực thi dịch vụ đó. Việc thực hiện trước thiết kế này làm tăng cơ hội của bạn trong việc xây dựng một dịch vụ đáp ứng nhu cầu của các client.

Ở các mục tiếp theo, ta sẽ thấy bản chất API phụ thuộc vào kĩ thuật IPC đang được
messaging, API sẽ chứa kiểu tin nhắn và các kênh tin nhắn. Nếu bạn dùng HTTP, A
response. Ở phần sau chúng ta sẽ mô tả một số IDL chi tiết hơn.



## Phát triển các API

API của service thay đổi theo thời gian. Với ứng dụng đơn khối, cập nhật API nhìn chung khá dễ dàng. Đối với ứng dụng kiểu microservices, nó khó gấp 10 lần. Ta không thể ép các clients update cùng thời điểm với dịch vụ. Do đó, việc cho ra

đời các phiên bản mới, đồng thời hỗ trợ các bản cũ là điều tất yếu. Việc đề ra chiến thuật hợp lý cho vấn đề này là rất quan trọng.

Chắc chắn ta nắm rõ quy mô thay đổi của API, vậy làm thế nào để xử lý hiệu quả?

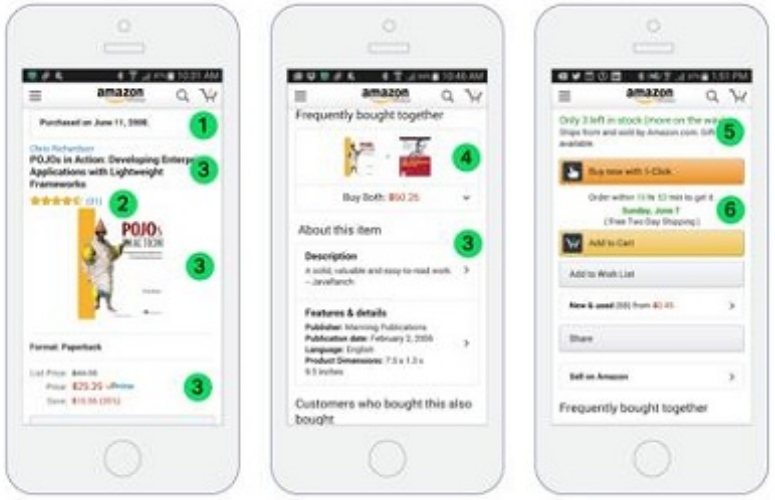
Với những thay đổi nhỏ và tính tương thích vẫn chưa bị phá vỡ. Ta có thể thêm các thuộc tính vào phần yêu cầu hoặc phần hồi đáp (lại thêm một lí do nữa để thiết kế clients và services theo chuẩn Robustness (thiết kế mạnh)). Các clients sử dụng API cũ vẫn làm việc bình thường với phiên bản mới của dịch vụ, tuy nhiên, với các request không được hỗ trợ, dịch vụ sẽ trả về các giá trị mặc định, tương tự, clients cũng sẽ bỏ qua các hồi đáp không tương thích với phiên bản của clients.

Với những thay đổi lớn, tính tương thích bị phá vỡ, khi clients chưa cập nhật, dịch vụ phải chấp nhận hỗ trợ phiên bản cũ một thời gian. Nếu bạn đang sử dụng một cơ chế dựa trên HTTP như REST, một cách tiếp cận là nhúng các số phiên bản trong URL. Mỗi instance dịch vụ có thể xử lý nhiều phiên bản cùng một lúc. Ngoài ra, bạn có thể triển khai các instance khác nhau mà mỗi cái xử lý một phiên bản riêng biệt.

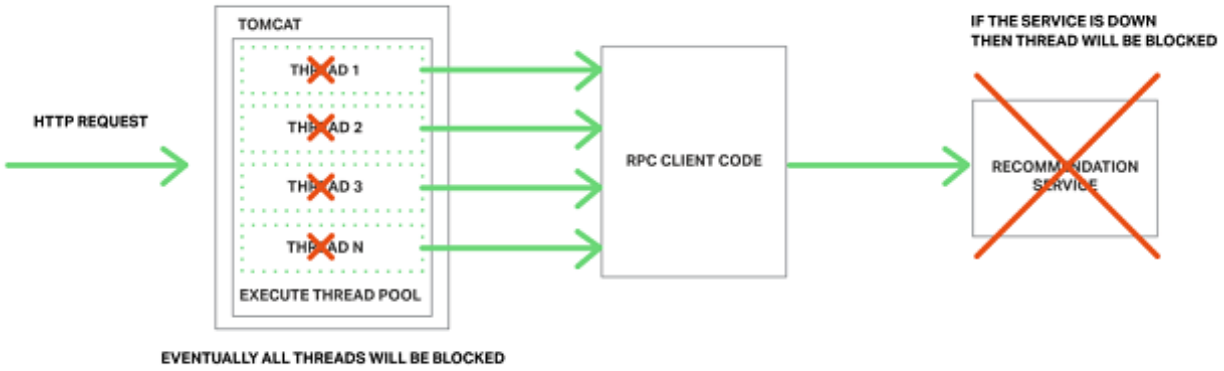
## Xử lý lỗi cục bộ

Vấn đề trên đã từng xuất hiện từ kì 2 của series này. Trong một hệ thống phân tán luôn tiềm ẩn các lỗi cục bộ, chẳng hạn, dịch vụ có thể không đáp ứng kịp yêu cầu, ngưng hoạt động tạm thời, bị quá tải...

Hãy cùng xem xét một kịch bản của sản phẩm ở kì 2 (ứng dụng shopping)




Giả sử dịch vụ gợi ý mua hàng (Recommendation) không phản hồi, nếu chẳng may client nào đó được cài đặt một cách "ngô nghê" đang kết nối với dịch vụ này, nó sẽ mãi mãi "đợi" phản hồi. Điều này khiến người dùng "bực" và hơn hết, nó chiếm dụng những thread liên quan. Cuối cùng, khi các thread bị chiếm dụng hết, toàn bộ chương trình sẽ bị treo. Bạn hãy theo dõi sơ đồ sau để có góc nhìn trực quan hơn:




Vậy, nhiệm vụ sống còn cho hệ thống của bạn là phải xử lý tốt những lỗi cục bộ kiểu này.

Vỏ quýt dày có móng tay nhọn, một phương pháp khá hiệu quả đã được mô tả chi tiết bởi Netflix. Chiến lược này gồm các điểm đáng chú ý sau:

- Network timeouts: timeout cho clients trong thời gian chờ hồi đáp, không block. Sử dụng timeouts đảm bảo tài nguyên client không bị chiếm dụng vô thời hạn.
- Giới hạn số lượng các yêu cầu còn tồn tại (chưa xử lí): đặt một ngưỡng đánh dấu số request tối đa mà mỗi client có thể gửi tới một dịch vụ. Quá con số này, mọi yêu cầu của client sẽ tự động hủy.
- Mô hình cầu giao ngắt mạch (Circuit breaker pattern): thống kê số yêu cầu thành c
  cầu lỗi vượt quá ngưỡng đã định, ngắt cầu giao (circuit breaker) để tất cả yêu cầu s
  yêu cầu bị lỗi vẫn tiếp tục tăng lên, sẽ có thông báo rằng dịch vụ không thể truy cập
  Sau 1 chu kì timeout, client có thể thử lại, nếu thành công, circuit breaker sẽ được
- Fallback: trả lại cached data hoặc giá trị mặc định (tập rỗng hoặc các khuyến cáo)

Netflix Hystrix là một thư viện mã nguồn mở hỗ trợ cài đặt mô hình chiến lược trên. Bạn đang dùng máy ảo Java (Java Virtual Machine)? Bạn nên cân nhắc Hystrix. Nếu không, bạn nên tìm một thư viện tương đương.


## Công nghệ IPC

Có hàng tá các công nghệ IPC khác nhau. Bạn có thể chọn các kỹ thuật tương tác đồng bộ, chẳng hạn giao thức HTTP dựa trên REST, hoặc Thrift. Một số lựa chọn khác thuộc kỹ thuật tương tác không đồng bộ: AMQP, STOMP. Bên cạnh đó, định dạng của thông báo (message) cũng vô cùng phong phú. Dịch vụ có thể sử dụng những định dạng gần gũi với ngôn ngữ con người như JSON, XML hoặc dạng nhị phân (không gần gũi tẹo nào, nhưng hiệu năng cao!). Các đại diện cho message dạng nhị phân: Avro, Protocol Buffers. Hãy để dành kỹ thuật IPC đồng bộ cho đoạn tiếp theo, và bây giờ, chúng ta hãy dành thời gian cho kỹ thuật IPC không đồng bộ (asynchronous IPC mechanisms.)

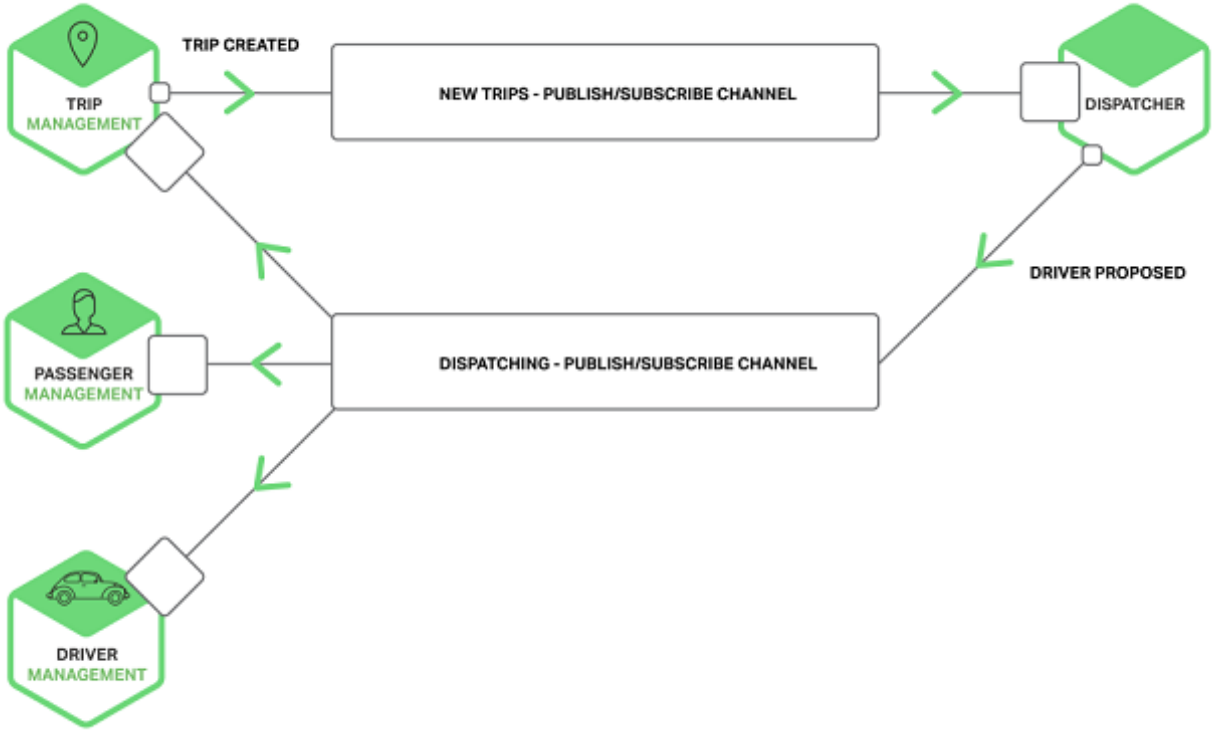
**Kỹ thuật tương tác không đồng bộ dựa trên hệ thống thông điệp:**

Khi sử dụng hệ thống thông điệp, các tiến trình tương tác nhờ trao đổi thông điệp (
dịch vụ bằng cách gửi thông điệp. Tương tự, bất kỳ consumer nào cũng có thể nhậ
kênh, ta có 2 loại: điểm-điểm (point-to-point) và công bố-theo dõi (publish-subscri



- Các kênh dạng điểm-điểm có trách nhiệm chuyển thông điệp đến consumer đang t
  kênh điểm-điểm được service tận dụng cho kiểu tương tác 1-1 giữa client-service
- Các kênh dạng công bố-theo dõi phân phát thông điệp tới một nhóm consumer được gắn kèm. Service dùng các kênh dạng này cho kiểu tương tác 1-n.

Biểu đồ dưới đây nói về việc áp dụng kênh loại 2 (publish-subscribe) trong ứng dụng gọi xe taxi:




Bộ phận quản lý tuyến (Trip Management service) khởi tạo một thông điệp với tiêu
kênh publish/subscribe với mục đích thông báo về một chuyến đi mới tới các serv
Sau đó, khối Dispatcher tìm kiếm một tài xế thỏa mãn điều kiện rồi thông báo tới c
thông điệp "DRIVER PROPOSED" và bắn nó lên kênh publish/subscribe.

Có rất nhiều hệ thống thông điệp cho sự lựa chọn của bạn. Mách nhỏ, bạn nên chọ
ngữ lập trình nhé!)... Một số hệ thống hỗ trợ các giao thức chuẩn như AMQP và STOMP: RabbitMQ, Apache Kafka,
Apache ActiveMQ, NSQ... là những đại diện cho các hệ thống sử dụng mã nguồn mở. Ở cấp độ cao, chúng đều hỗ trợ
nhiều định dạng thông điệp cũng như các loại kênh. Cuộc đua về hiệu năng, độ tin cậy và khả năng chịu tải giữa các hệ
thống này chưa bao giờ đến hồi kết. Dù vậy, vẫn có kha khá những khác biệt về cách thức làm việc giữa chúng.

Ta dễ dàng thấy được một loạt các ưu điểm khi sử dụng hệ thống thông điệp:

- Tách riêng client và service: client muốn tạo một request tới service thì chỉ việc gửi một message tới kênh thích hợp – thế là đủ
- Bộ đệm cho thông điệp: với giao thức request/response đồng bộ (HTTP là một ví dụ), cả client và service phải sẵn sàng (available) trong suốt quá trình trao đổi. Mặt khác, một hệ thống đóng vai trò trung gian sẽ đẩy message vào hàng đợi để ghi vào kênh cho đến khi consumer nhận được message này. Ví dụ, một shop online có thể nhận được đơn hàng từ người mua hàng ngay cả khi hệ thống đáp ứng đơn hàng không sẵn sàng, bởi lẽ, đơn hàng kia sẽ được xếp ngay vào hàng đợi khi hệ thống đáp ứng có vấn đề.
- Linh hoạt trong tương tác client-service: Như ta đã biết, hệ thống thông điệp (messaging) hỗ trợ hầu như tất cả các phương thức tương tác.
- IPC được phân biệt rõ: các kĩ thuật RPC(*) truy xuất một dịch vụ ở xa giống như dịch vụ cục bộ. Tuy nhiên qua những định luật vật lý và nguy cơ lỗi cục bộ, hai quá trình này lại khác nhau hoàn toàn. Hệ thống thông điệp lại càng đẩy chúng ra xa hơn, nhờ vậy, các dev sẽ tránh được nhầm lẫn đáng tiếc.

Chúng ta phải thừa nhận rằng, không thể tránh khỏi một số hạn chế của hệ thống thông điệp:

- Thao tác phụ phức tạp: Hệ thống thông điệp giống như bao component khác, nó cần được cài đặt, hiệu chỉnh, rồi thực thi. Điều thiết yếu là phải đảm bảo bộ phận trung gian (the message broker) luôn luôn sẵn sàng, nếu không, độ tin cậy của hệ thống sẽ giảm sút.
- Phức tạp trong việc thi hành kiểu tương tác request/response: kiểu tương tác request/response đòi hỏi một số công đoạn cài đặt. Mỗi request message phải có một module nhận diện kênh phản hồi (reply chanel) kèm với một module kiểm chứng ID tương ứng. Kéo theo đó, response từ service cần chứa ID tương ứng yếu tố này, client sẽ kiểm tra ID để đảm bảo request và response khớp nhau.

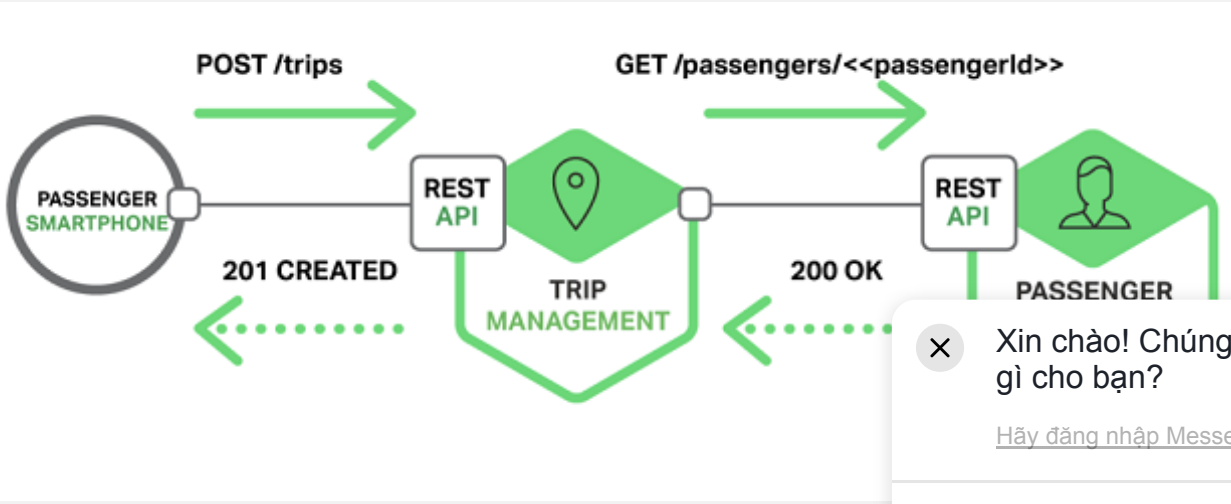
**Kỹ thuật tương tác đồng bộ dựa trên request/response IPC:**

Với kĩ thuật này, client gửi request đến service, service xử lí rồi gửi lại response. Ở request sẽ bị chiếm dụng suốt quá trình chờ phản hồi. Một số client khác có thể sử sự kiện (event-driven) mà có lẽ được đóng gói bởi Futures hoặc Rx Observables. Tuy nhiên, không giống như khi sử dụng tin nhắn (messaging), client giả định rằng response sẽ đến một cách kịp thời. Số lượng các giao thức có thể sử dụng cho kĩ thuật này cũng không phải là ít, trong đó hai giao thức phổ biến nhất là REST và Thrift.

*Đầu tiên, hãy để ý đến REST.*

Ngày nay, việc xây dựng APIs theo phong cách RESTful được coi là "mốt". REST là một kĩ thuật IPC thường dùng tới giao thức HTTP. Khái niệm chính của REST là một tài nguyên (resources), thường đại diện cho đối tượng của doanh nghiệp (sản phẩm, khách hàng...). REST thao tác với resource nhờ các HTTP method. Ví dụ, method GET sẽ trả về các miêu tả chung của đối tượng dưới dạng file XML hoặc đối tượng JSON, method POST khởi tạo resource mới, còn PUT sẽ ứng với update một resource.

"REST đem đến một tập hợp mang tính cấu trúc của các ràng buộc, nhằm nhấn mạnh tính đáp ứng của quá trình tương tác giữa các component; tính tổng quát của các interface; sự phát triển độc lập của các component; và các component trung gian để giảm độ trễ giữa các tương tác, tăng cường bảo mật, đóng gói các hệ thống di sản (encapsulate legacy systems)." - Roy Fielding, cha đẻ của REST




Đây là biểu đồ thể hiện một cách triển khai REST cho ứng



Ứng dụng yêu cầu một chuyến xe thông qua việc khởi tạo POST request và trỏ nó đến /trip resource thuộc bộ phận quản lý tuyến (Trip Management). Bộ phận này xử lý bằng cách bắn GET request đến khối quản lý khách hàng. Khối quản lý

khách hàng kiểm tra thông tin khách và gửi lại thông báo 200 nếu tài khoản hợp lệ. Tiếp đó, bộ phận quản lý tuyến tạo một lịch trình cho chuyến xe mới và gửi thông báo 201 tới smartphone của khách.

Nhiều Dev tự nhận các API xây dựng trên HTTP của họ là RESTful nhưng sự thực không phải là tất cả. Leonard Richardson đã vạch rõ mô hình hoàn thiện cho REST như sau:

- Level 0: client truy xuất tới service nhờ HTTP POST request. Mỗi request chỉ rõ hành động, mục tiêu (taget), một vài tham số đi kèm.
- Level 1: API ở level 1 hỗ trợ về ý đồ của resource. POST request lúc này sẽ chỉ rõ hành động và các tham số đi kèm.
- Level 2: API level 2 sử dụng HTTP methods để thực thi: GET: lấy thông tin, POST: khởi tạo, PUT: cập nhật.
- Level 3: kiến trúc của level 3 API dựa trên nguyên tắc HATEOAS (Hypertext As The Engine Of Application State)

Như thường lệ, ta hãy điểm qua các điểm mạnh của việc sử dụng giao thức trên nền HTTP:

- HTTP đơn giản và phổ biến
- Bạn có thể kiểm tra một API HTTP từ bên trong một trình duyệt bằng cách sử dụng một phần mở rộng như Postman hoặc từ dòng lệnh bằng cách sử dụng curl (giả sử JSON hoặc một số định dạng văn bản khác được sử dụng).
- Hỗ trợ trực tiếp kiểu tương tác request/response
- Đơn giản hóa kiến trúc hệ thống vì không cần dùng bộ phận trung gian.

Sau đây là nhược điểm:

- Server luôn phải gửi response dưới dạng HTTP
- Client và service phải luôn sẵn sàng trong quá trình trao đổi
- Client phải biết địa chỉ của service instance. Như chúng ta đã đề cập trong phần th
  hề đơn giản, client cần một bộ tìm kiếm để xác định chính xác service instance.



Gần đây, cộng đồng dev phát hiện ra giá trị của interface definition language - IDL
nổi bật của IDL gồm Swagger và RAML. Swagger và các IDL tương tự cho phép ta
messages. Trong khi đó, họ nhà RAML yêu cầu các bản ghi riêng biệt như JSON Schema. Các IDL cũng hỗ trợ các công cụ sinh ra client stubs và server skeletons từ một định nghĩa interface.

*Bây giờ, ta sẽ xét đến Thrift.*

Apache Thrift là một framework được tạo ra phục vụ cho cross-language RPC clients và servers. Trình biên dịch của Thrift dùng để sinh ra các client stubs và server skeletons. Trình biên dịch này sinh mã phù hợp với nhiều ngôn ngữ lập trình (C++, JAVA, Python, PHP, Ruby...).

Một interface của Thrift gồm một hoặc nhiều service. Định nghĩa của service ở đây tương tự như trong Java interface – là một tập các method. Các method này có thể trả về một giá trị hoặc được khai báo kiểu one-way. Method nào trả về giá trị sẽ thực thi request/response.

Về định dạng message, Thrift hỗ trợ rất nhiều: JSON, binary, compact binary. Dùng binary đạt hiệu năng cao hơn vì không tốn thời gian giải mã như JSON, compact binary tiết kiệm bộ nhớ, còn JSON thì thân thiện với người dùng.

Thrift mang đến 2 lựa chọn về giao thức vận chuyển: raw TCP và HTTP. Raw TCP và HTTP giống như binary với JSON vậy, một bên hiệu năng cao, một bên thân thiện với người dùng.

*Hãy dành những dòng cuối cùng của kì 3 cho Message Format.*

Nếu bạn đang sử dụng hệ thống thông điệp (messaging) hoặc REST, bạn phải chọn định dạng cho message và đặc biệt lưu ý tới tính đa ngôn ngữ của định dạng đó.

Có 2 loại định dạng chính cho message: kiểu text và binary.

<table>
<tr><th></th><th colspan="2">Text</th><th></th></tr>
<tr><td>Ví dụ</td><td>JSON</td><td>XML</td><td></td></tr>
<tr><td rowspan="2">Ưu</td><td colspan="2">Thân thiện với con người<br>Tự mô tả(self-describing)</td><td></td></tr>
<tr><td>Attribute = <name,value></td><td>Attribute = <name,ele></td><td></td></tr>
<tr><td>Nhược</td><td colspan="2">Hiệu năng thấp hơn binary<br>Message dễ bị dài dòng</td><td>Kém thân thiện với con người</td></tr>
</table>

Bảng trên thống kê sơ bộ một số ưu nhược điểm chính của 2 dạng text và binary.

Với định dạng text, việc biểu thị các tính chất của đối tượng thông qua những cặp <name,value> hoặc <name,ele> giúp consumer dễ dàng chọn ra được thứ mình cần trong một message.

Hiệu năng của định dạng text thấp hơn do nó cần thêm một bộ phân tích từ vựng.

## Tổng kết

Các microservices phải tương tác với nhau nhờ kĩ thuật IPC. Trong quá trình thiết kế phần này, ta cần xác định rõ những vấn đề như:

- Các service tương tác với nhau bằng con đường nào?
- Làm cách nào để vạch ra API cho mỗi service?
- Xây dựng các API như thế nào?
- Khắc phục lỗi cục bộ

Có 2 kĩ thuật IPC có thể áp dụng cho mô hình microservices: hệ thống thông điệp không đồng bộ và hệ thống request/response đồng bộ.

*Bản dịch của Đinh Công Minh, lập trình viên Java Spring tại Techmaste*
*Hiệu đính: Hồ Sỹ Hùng*

### Middle/ Senior PHP Developer (MySQL, Laravel)

**CÔNG TY CỔ PHẦN X – IDEAS VIỆT NAM** *Ha Noi* *Up to $1,400*

PHP MySQL Laravel

### 04 Junior/Senior PHP Developers

**GDC Group** *Ha Noi* *$500 - $1,000*

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

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**