Truy tìm dịch vụ trong một kiến trúc Microservices

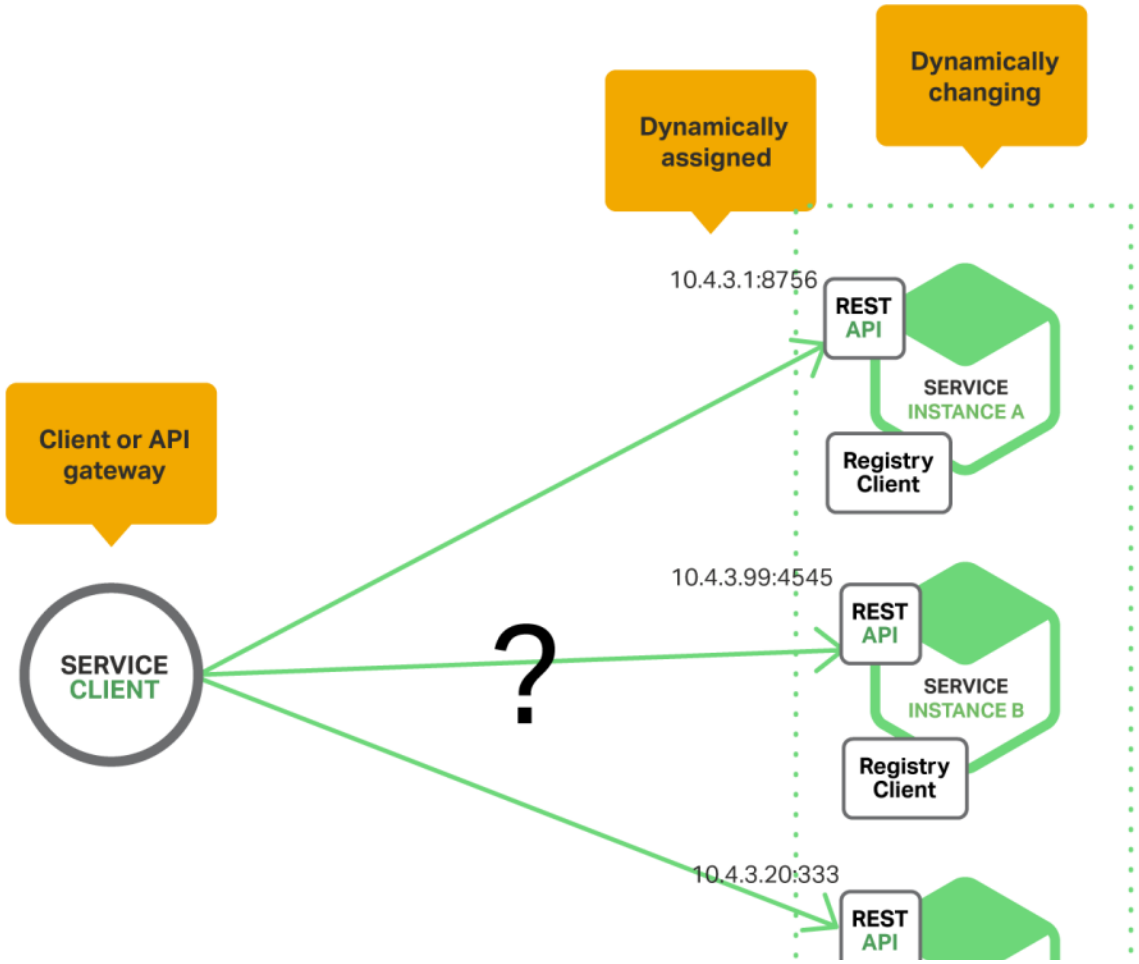
02/11/2015 / Bởi **Techmaster Team** / trong **Microservice**

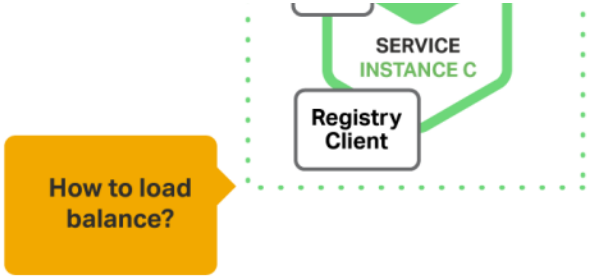
Đây là bài thứ tư trong loạt bài về thiết kế, xây dựng và triển khai các ứng dụng sử dụng kiến trúc Microservices. Trong bài đầu tiên, mô hình kiến trúc Microservices đã được giới thiệu và thảo luận về những ưu điểm và hạn chế trong việc ứng dụng và triển khai. Bài thứ hai mô tả về cách thức mà các client tương tác với các dịch vụ nhỏ thông qua một phương tiện trung gian gọi là API gateway. Bài thứ ba tìm hiểu về cách mà các dịch vụ nhỏ trong cùng một hệ thống giao tiếp với nhau. Trong bài viết này, chúng ta cùng làm sáng tỏ những vấn đề liên quan gần gũi đến truy tìm dịch vụ.

Tại sao cần truy tìm dịch vụ?

Tưởng tượng rằng bạn đang viết một đoạn mã để truy cập đến một dịch vụ mà dịch vụ này có REST API hoặc Thrift API. Đoạn mã đó cần có thông tin về vị trí mạng (địa chỉ IP và cổng) của một thực thể dịch vụ để gửi yêu cầu. Trong một ứng dụng truyền thống chạy trên phần cứng vật lý, các vị trí mạng của các instance service là tương đối tĩnh. Ví dụ, code của bạn có thể đọc các vị trí mạng từ một tập tin cấu hình được cập nhật thường xuyên.

Ngược lại, trong các ứng dụng microservices hiện đại được xây dựng và triển khai trên nền tảng điện toán đám mây. Do đó, việc truy tìm dịch vụ là một nhiệm vụ khó khăn hơn rất nhiều, thể hiện trong sơ đồ sau đây.



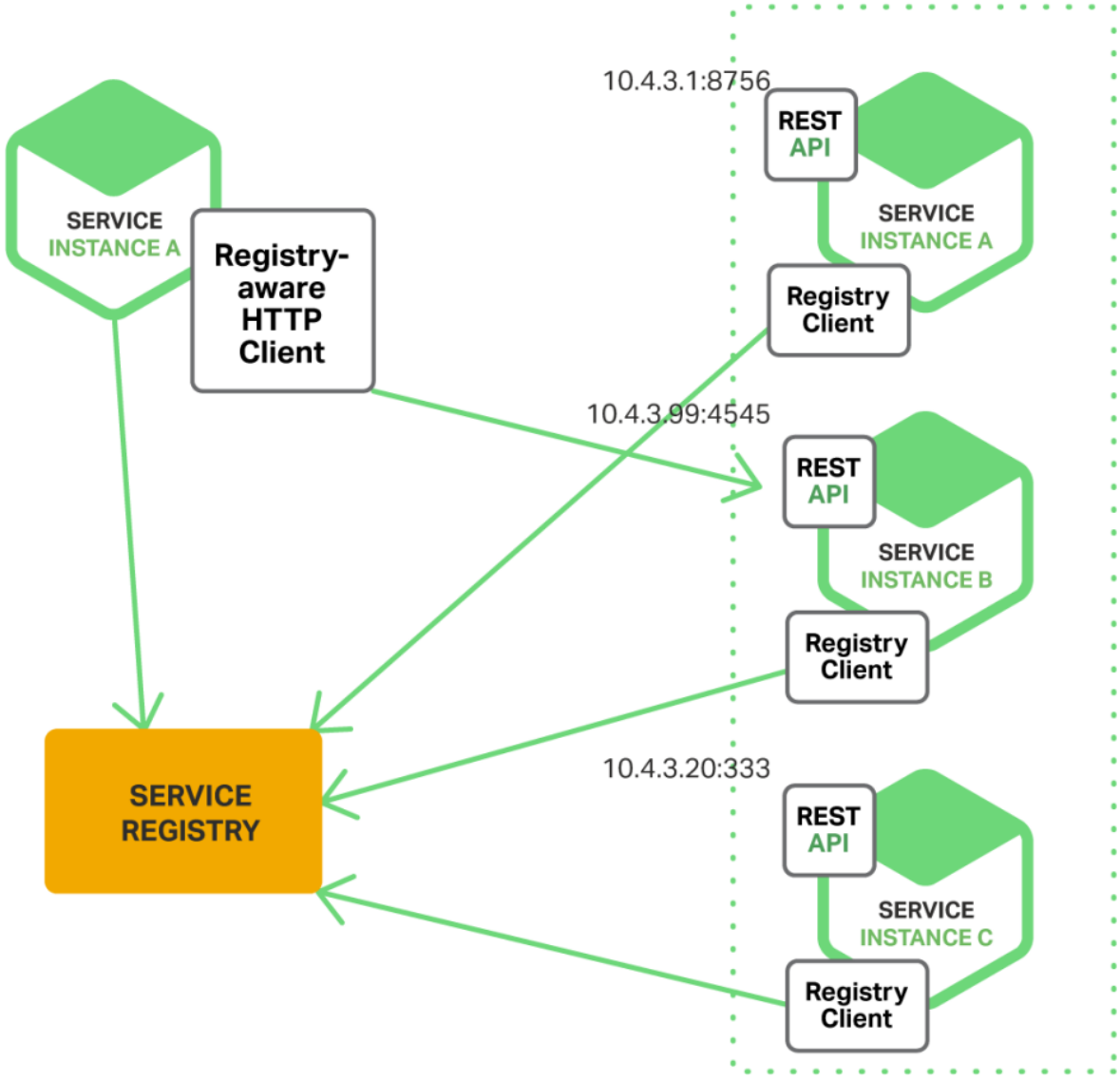


Mỗi thực thể dịch vụ trong một ứng dụng microservices đều được gán vị trí mạng và chúng có khuynh hướng thay đổi linh hoạt vì những tác động khác mang đến. Do đó, đoạn mã của client cần áp dụng một kỹ thuật tìm kiếm phức tạp hơn.

Có hai mô hình chủ yếu được sử dụng để truy tìm dịch vụ là: client-side discovery và server-side-discovery.

Mô hình client-side discovery

Mô hình client-side discovery quy định rằng các client sẽ phải xác định vị trí mạng của các dịch vụ khả dụng và yêu cầu cân bằng tải. Để làm được điều này client cần truy vấn một cơ sở dữ liệu gọi là Service Registry. Đây là một cơ sở dữ liệu chứa thông tin và vị trí mạng của các thực thể dịch vụ. Sau khi truy vấn được vị trí của dịch vụ, client sử dụng một thuật toán cân bằng tải giúp chọn ra một dịch vụ khả dụng để gửi yêu cầu.



Mô hình này đối phó với việc vị trí mạng không ổn định của mỗi dịch vụ bằng cách:

- Khi khởi động dịch vụ, ghi vị trí mạng vào Service Registry và xóa bỏ khi thực thể dịch vụ này kết thúc.
- Áp dụng kỹ thuật tín hiệu tuần hoàn (heartbeat mechanism) vào việc cập nhật vị trí mạng của các dịch vụ.

Netflix OSS cung cấp một ví dụ tuyệt vời của mô hình client-side discovery. Netflix Eureka là một service registry. Nó cung cấp một REST API cho việc quản lý đăng ký service-instance và cho việc truy vấn các instance có sẵn. Netflix Ribbon là một IPC client làm việc với Eureka để cân bằng tải các yêu cầu trên các service instance có sẵn. Chúng ta sẽ thảo luận về Eureka sâu hơn ở phần sau của bài viết này.

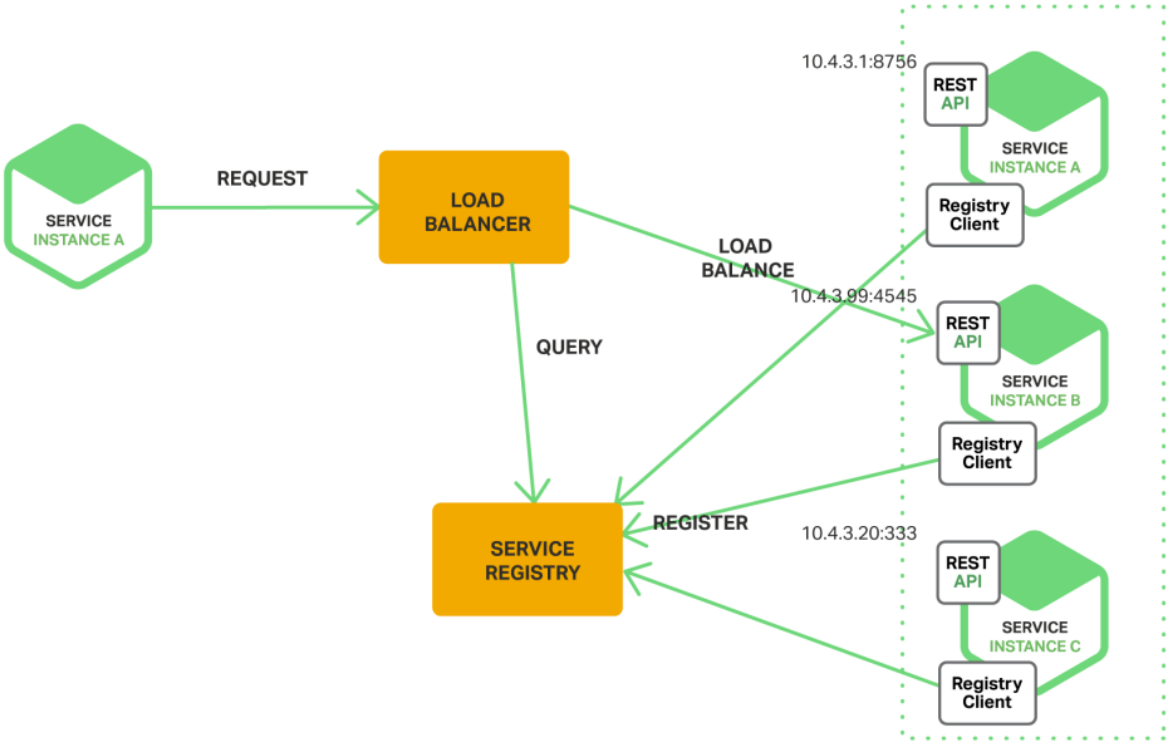
Mô hình client-side discovery có những lợi ích:

- Một mô hình đơn giản, ngoại trừ service registry thì không có gì phức tạp.

Những hạn chế:

- Sự ràng buộc giữa client với Service Registry
- Cần viết mã lệnh cho client đối với mỗi ngôn ngữ lập trình và framework

Mô hình server-side discovery



Client tạo ra một yêu cầu tới một dịch vụ thông qua một bộ cân bằng tải. Bộ cân bằng tải sẽ truy vấn Service Registry và định tuyến mỗi yêu cầu tới một instance service có sẵn. Cũng giống như mô hình client-side discovery, các instance service được đăng ký và hủy đăng ký với Service Registry.

Một AWS Elastic Load Balancer (ELB) là ví dụ về một bộ định tuyến server-side discovery. Một ELB thường được sử dụng để cân bằng tải lưu lượng truy cập từ bên ngoài Internet. Tuy nhiên, bạn cũng có thể dùng ELB để cân bằng tải lưu lượng truy cập bên trong một đám mây riêng ảo (Virtual Private Cloud). Một client tạo ra yêu cầu (HTTP hay TCP) thông qua ELB sử dụng tên DNS của nó. ELB cân bằng tải lưu lượng truy cập trong một tập hợp các instance Elastic Compute Cloud (EC2) đã đăng ký hoặc các container EC2 Container Service (ECS). Không có một Service Registry riêng biệt. Thay vào đó, các instance EC2 và các container ECS được đăng ký với chính ELB.

Các máy chủ HTTP và cân bằng tải như NGINX Plus và NGINX cũng có thể được sử dụng như là một cân bằng tải server-side discovery. Ví dụ, bài viết trên blog này mô tả cách sử dụng Consul Template để tự động cấu hình lại NGINX làm reverse proxying. Consul Template là một công cụ mà theo định kỳ tái tạo các tập tin cấu hình tùy ý từ dữ liệu cấu hình được lưu trữ trong Consul Service Registry. Nó chạy một lệnh shell tùy ý bất cứ khi nào các tập tin thay đổi. Ở ví dụ được mô tả trong bài viết blog đó, Consul Template tạo ra một tập tin **nginx.conf**, để cấu hình reverse proxying, và sau đó chạy một lệnh mà nói với NGINX tải lại cấu hình. Một thực thi tinh vi hơn có thể cấu hình động lại NGINX Plus bằng cách sử dụng API HTTP hoặc DNS của nó.

Trong một vài môi trường triển khai như Kubernetes và Marathon, proxy đóng vai trò như một cân bằng tải. Để gửi một yêu cầu đến một dịch vụ thì một client cần gửi yêu cầu thông qua proxy sử dụng địa chỉ IP của máy chủ và cổng dịch vụ đã được gán. Sau đó, proxy sẽ chuyển tiếp yêu cầu đến một thực thể dịch vụ khả dụng.

Server-side discovery có những lợi ích:

- Các client chỉ cần gửi yêu cầu tới bộ cân bằng tải, quá trình truy tìm tách biệt với client, phần mã lệnh của client cũng đơn giản hơn. Do đó, giảm đi mã lệnh cho client đối với mỗi ngôn ngữ lập trình và framework.
- Một vài môi trường triển khai cung cấp sẵn mô hình truy tìm dịch vụ này, trong đó có AWS Elastic Load Balancer.

Những hạn chế của mô hình server-side discovery:

- Nếu bộ cân bằng tải không được tích hợp sẵn vào môi trường triển khai thì bạn cần phải cài đặt và quản lý.

Service Registry

Service Registry (SR) là một bộ phận rất quan trọng trong truy tìm dịch vụ. Nó là một cơ sở dữ liệu bao gồm vị trí mạng của các thực thể dịch vụ, do đó khả năng cập nhật tức thời luôn là một yêu cầu cao. Về lý thuyết, các client có thể lưu vị trí mạng được lấy từ SR vào bộ nhớ tạm thời, nhưng chắc chắn rằng những thông tin này nhanh chóng vô dụng vì vị trí mạng thay đổi rất linh động và thường xuyên. Vậy nên, một SR bao gồm rất nhiều máy chủ sử dụng giao thức nhân bản để duy trì sự nhất quán dữ liệu.

Như đã đề cập ở trên, Netflix Eureka là một ví dụ điển hình của một SR. Nó cung cấp một REST API giúp bạn đăng ký và truy vấn các thực thể dịch vụ. Một thực thể dịch vụ sử dụng lệnh POST để đăng ký vị trí mạng của nó và cứ mỗi 30 giây nó tiếp tục dùng lệnh PUT để gửi yêu cầu cập nhật đến SR. Thông tin của một dịch vụ sẽ bị tự động xóa bỏ nếu không cập nhật sau 30 giây và nó cũng có thể bị xóa khi dùng lệnh HTTP DELETE. Client có thể sử dụng lệnh HTTP GET để lấy thông tin vị trí mạng của một dịch vụ.

Netflix có được tính khả dụng cao là nhờ chạy một hoặc nhiều máy chủ Eureka trong mỗi vùng AWS EC2 khả dụng. Mỗi máy chủ Eureka chạy trên một thực thể EC2 – có một địa chỉ IP Elastic. Các bản ghi của DNS TEXT được sử dụng để lưu trữ dữ liệu cấu hình của Eureka. Đó là một tấm bản đồ chứa thông tin các vùng khả dụng và danh sách các vị trí mạng của các máy chủ Eureka. Khi một máy chủ Eureka khởi động, nó truy vấn DNS để lấy thông tin cấu hình của nhóm Eureka, xác định vị trí các máy chủ khác và tự gán cho nó một địa chỉ IP Elastic chưa dùng đến.

Eureka clients- service và service – clients truy vấn DNS để tìm ra vị trí mạng của các máy chủ Eureka. Clients được khuyến khích dùng một máy chủ Eureka trong cùng một vùng khả dụng. Tuy nhiên, nếu không còn cái nào khả dụng thì client sẽ sử dụng một máy chủ Eureka ở một vùng khả dụng khác.

Một vài ví dụ khác về SR:

- etcd – một cơ sở dữ liệu key-value có tính khả dụng cao, phân phối và nhất quán sử dụng để tìm kiếm dịch vụ. Kubernetes và Cloud Foundry là hai dự án nổi tiếng có dùng etcd.
- consul – một công cụ giúp tìm kiếm và cấu hình các dịch vụ. Nó cung cấp một bộ API cho phép các client đăng ký và tìm kiếm dịch vụ. Consul có khả năng kiểm tra tính khả dụng của các dịch vụ.
- Apache Zookeeper – một dịch vụ được sử dụng rộng rãi, có khả năng phối hợp cao dành cho các ứng dụng phân tán. Apache Zookeeper ban đầu là một phần dự án của Hadoop nhưng giờ nó là một dự án quan trọng hàng đầu.

Cần nhắc lại rằng SR là một bộ phận được tích hợp trong hạ tầng dịch vụ, những hệ thống như Kubernetes, Marathon và AWS không tồn tại một SR hoàn chỉnh.

Các mô hình đăng ký thông tin dịch vụ

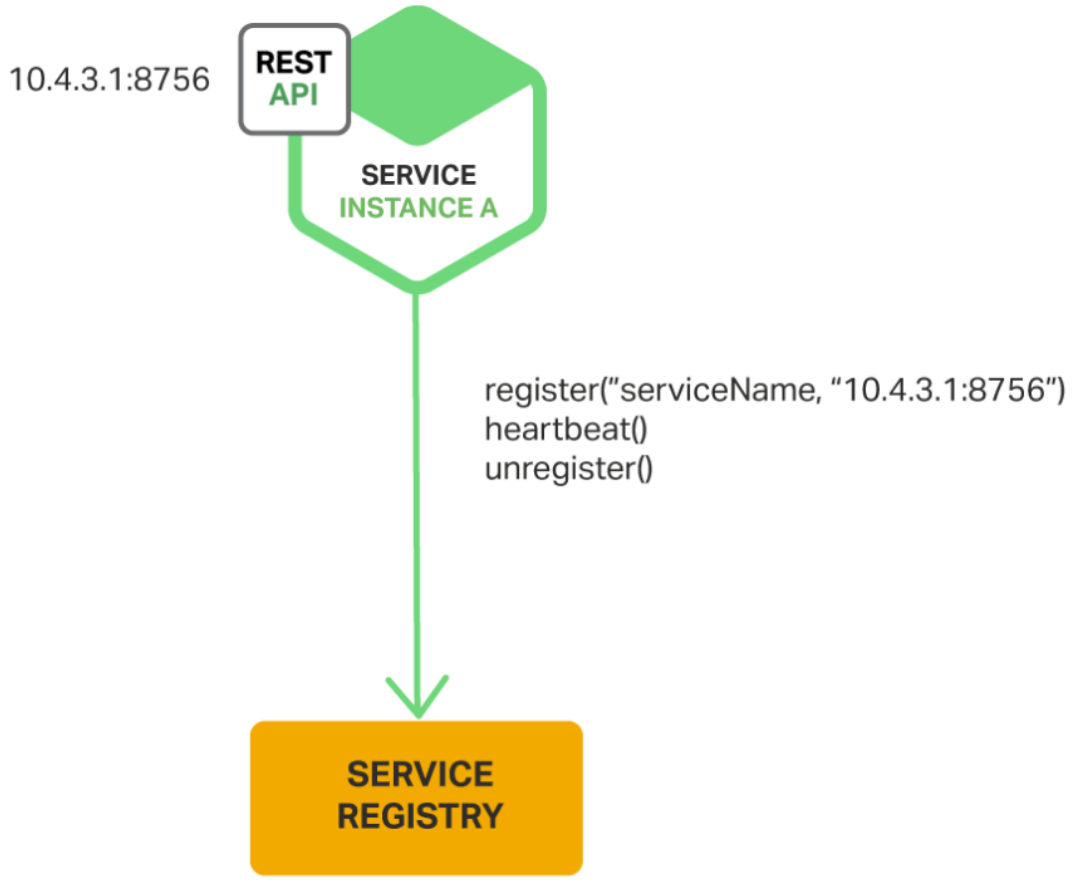
Bằng cách nào đó, thông tin của các thực thể dịch vụ phải được ghi và xóa trong SR. Có hai phương pháp để giải quyết vấn đề này:

Một là thực thể dịch vụ tự ghi thông tin lên SR (self-registration pattern)

Hai là sử dụng một công cụ hỗ trợ ghi và quản lý (third party registration pattern)

Mô hình dịch vụ tự ghi xóa thông tin

Trong mô hình này, mỗi thực thể dịch vụ sẽ chịu trách nhiệm trong việc ghi và xóa thông tin của chính nó trong SR. Đồng thời, mỗi thực thể cũng thường xuyên gửi các yêu cầu tuần hoàn để cập nhật thông tin.



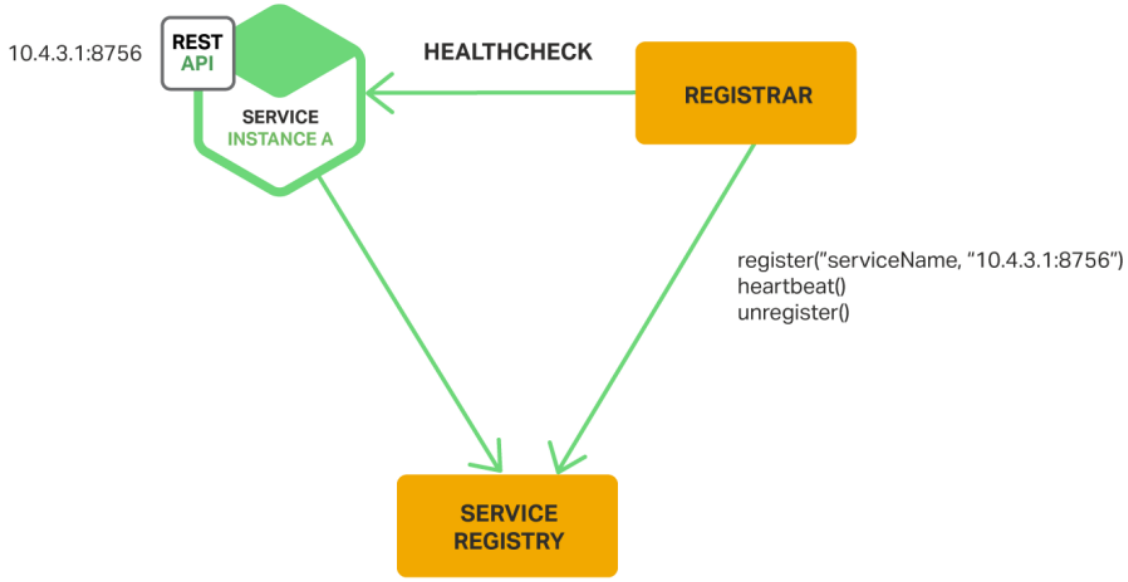
Netflix OSS Eureka client là một ví dụ minh họa cho mô hình này. Eureka client quản lý tất cả các vấn đề liên quan đến đăng ký và hủy đăng ký thông tin của các thực thể dịch vụ. Dự án Spring Cloud (có hỗ trợ truy tìm dịch vụ) giúp Eureka tự

động ghi thông tin dịch vụ một cách dễ dàng. Bạn chỉ cần chú thích class Java Configuration của mình với một chú thích @EnableEurekaClient.

Điểm lợi của mô hình này là nó tương đối đơn giản không yêu cầu sự trợ giúp của bất kỳ bên thứ ba nào. Tuy nhiên, việc các thực thể dịch vụ bị ràng buộc với SR lại là một hạn chế, bạn cần phải viết các mã lệnh hỗ trợ việc đăng ký cho các dịch vụ trong mỗi ngôn ngữ lập trình và framework.

Mô hình hỗ trợ đăng ký bên thứ ba

Trong mô hình này, các thực thể dịch vụ sẽ không tự đăng ký thông tin với SR, công việc sẽ được giao cho một thành phần bên thứ ba đó là một *Service Registrar*. Nó sẽ thường xuyên kiểm tra xem khi một thực thể dịch vụ khởi động lên, Service Registrar sẽ ghi thông tin vị trí mạng của dịch vụ đó vào SR. Khi dịch vụ dừng hoạt động, Registrar sẽ xóa thông tin về thực thể này khỏi SR.



Dự án nguồn mở Registrator là ví dụ về một service registrar. Nó tự động đăng ký và hủy đăng ký thông tin các thực thể dịch vụ được triển khai bởi Docker containers. Registrator hỗ trợ một vài SR trong đó có etcd và Consul.

Một ví dụ khác về service registrar là Netflix Prana. Mục đích chính của registrar này là hỗ trợ các dịch vụ không được viết bởi ngôn ngữ JVM (Java Virtual Machine), nó là một ứng dụng phụ chạy song song với một thực thể dịch vụ. Prana dùng Netflix Eureka để ghi và xóa thông tin của các thực thể dịch vụ.

Service registrar là một thành phần gắn liền với hạ tầng triển khai dịch vụ. Mỗi thực thể EC2 được tạo ra bởi Autoscaling Group có thể được tự động đăng ký bởi một ELB. Các dịch vụ của Kubernetes được tự động đăng ký và luôn khả dụng cho việc tìm kiếm.

Một ưu điểm rõ ràng của mô hình này là các dịch vụ không bị ràng buộc với SR, mã lệnh của dịch vụ sẽ ít phức tạp hơn so với mô hình đầu tiên. Công việc đăng ký thông tin cho dịch vụ được tập trung xử lý bởi một hệ thống của bên thứ ba. Một hạn chế của mô hình này đó là, bạn sẽ phải cài đặt và quản lý registrar nếu nó không là một phần của hạ tầng triển khai dịch vụ hoặc không là một bộ phận luôn khả dụng trong hệ thống.

Kết luận

Trong một ứng dụng microservices, xu hướng của các thực thể dịch vụ đang chạy là thay đổi linh động. Các thực thể được gán vị trí mạng không cố định. Do đó, các client cần áp dụng kỹ thuật truy tìm dịch vụ để gửi yêu cầu đến.

Một bộ phận rất quan trọng trong truy tìm dịch vụ đó là Service Registry. Đây là một cơ sở dữ liệu của các thực thể dịch vụ đang được kích hoạt. Service registry cung cấp các API giúp quản lý ghi xóa và truy vấn thông tin dịch vụ.

Client-side discovery và server-side discovery là hai mô hình tìm kiếm dịch vụ chủ yếu được áp dụng. Trong hệ thống với mô hình client-side discovery, các client truy vấn đến SR, chọn một thực thể khả dụng và gửi yêu cầu đi. Trong hệ thống áp dụng mô hình server-side discovery, các client gửi yêu cầu đến dịch vụ khả dụng thông qua một thiết bị định tuyến (router). Router có nhiệm vụ truy vấn đến SR và chuyển tiếp yêu cầu từ client đến thực thể đang hoạt động.

Self-registration và third-party registration là hai phương pháp chính để đăng ký và hủy đăng ký thông tin dịch vụ trong SR. Cách thứ nhất do các thực thể dịch vụ tự thực hiện các phép ghi xóa cập nhật, cách thứ hai do một hệ thống bên thứ ba đảm nhiệm cập nhật và quản lý.

Trong một vài môi trường triển khai, bạn phải sử dụng một Service Registry như Netflix Eureka, etcd hay Apache Zookeeper để tự cài đặt hệ thống truy tìm dịch vụ của mình. Trong một số môi trường triển khai khác, hệ thống này được tích hợp sẵn. Ví dụ, Kubernetes và Marathon điều khiển việc đăng ký và hủy đăng ký thông tin dịch vụ. Chúng cũng có khả năng chạy một proxy trên mỗi cụm máy chủ - với vai trò như một bộ định tuyến server-side discovery.

Một proxy đảo ngược HTTP và cân bằng tải như NGINX cũng có thể sử dụng như một cân bằng tải. Service Registry có thể đẩy thông tin đến NGINX và gọi ra một bản cập nhật cấu hình dễ dàng. Ví dụ, bạn có thể sử dụng Consul Template. NGINX Plus hỗ trợ kỹ thuật additional dynamic reconfiguration – nó sử dụng DNS để lấy thông tin dịch vụ từ Service Registry đồng thời cung cấp một API để giúp cấu hình lại thông tin dịch vụ.

Trong các bài viết tiếp, chúng ta sẽ tiếp tục đào sâu tìm hiểu những vấn đề khác của microservices.

Bản gốc tiếng Anh "Service Discovery in a Microservice Architecture"
Bản dịch của nhóm học viên lớp Java Spring dịch

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

Cấu trúc thư mục một project sử dụng Go-Micro

BLOG HOME

29/03/2018 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

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**