BAN CƠ YẾU CHÍNH PHỦ

HỌC VIỆN KỸ THUẬT MẬT MÃ

ĐỀ CƯƠNG

CHUYÊN ĐỀ AN TOÀN HỆ THỐNG THÔNG TIN

Nghiên cứu giải pháp đảm bảo an toàn cho Microservices trên Kubernetes

Ngành: An toàn thông tin

Sinh viên thực hiện:
Phương Văn Sơn
Mã sinh viên: AT160258
Lê Huy Dũng
Mã sinh viên: AT160211

Người hướng dẫn:
TS. Nguyễn Mạnh Thắng
Khoa An toàn thông tin - Học viện Kỹ thuật mật mã

Hà Nội, 2022

Mục lục

Lời cảm ơn

Nhóm chúng em xin chân thành cảm ơn các thầy cô trường Học viện Kỹ thuật Mật mã nói chung, quý thầy cô của khoa An toàn thông tin nói riêng đã tận tình dạy bảo, truyền đạt kiến thức cho chúng em trong suốt quá trình học.

Kính gửi đến Thầy Nguyễn Mạnh Thắng lời cảm ơn chân thành và sâu sắc nhất, cảm ơn thầy đã tận tình theo sát, chỉ bảo và hướng dẫn cho nhóm em trong quá trình thực hiện đề tài này. Thầy không chỉ hướng dẫn chúng em những kiến thức chuyên ngành, mà còn giúp chúng em học thêm những kĩ năng mềm, tinh thần học hỏi, thái độ khi làm việc nhóm.

Trong quá trình tìm hiểu nhóm chúng em xin cảm ơn các bạn sinh viên đã góp ý, giúp đỡ và hỗ trợ nhóm em rất nhiều trong quá trình tìm hiểu và làm đề tài.

Do kiến thức còn nhiều hạn chế nên không thể tránh khỏi những thiếu sót trong quá trình làm đề tài.Chúng em rất mong nhận được sự đóng góp ý kiến của quý thầy cô để đề tài của chúng em đạt được kết quả tốt hơn.

Chúng em xin chân thành cảm ơn!

Lời mở đầu

Nhiều năm trước, hầu hết các ứng dụng phần mềm đều được xây dựng với kiến trúc monolith hay còn gọi là kiến trúc 1 khối là mẫu thiết kế được dùng nhiều nhất trong giới lập trình web hiện nay bởi tính đơn giản của nó khi phát triển và khi triển khai. Các ứng dụng này chạy dưới dạng một tiến trình đơn lẻ hoặc số lượng nhỏ các tiến trình trên một số ít máy chủ. Chúng có khả năng cập nhật và nâng cấp chậm và yêu cầu nâng cấp thường xuyên. Trong trường hợp có sự cố như lỗi phần cứng hệ thống phần mềm này sẽ phải được di chuyển một cách thủ công sang các máy chủ còn hoạt động tốt.

Ngày nay các ứng dụng được xây dựng với kiến trúc lớn và phức tạp đang dần được chia thành các thành phần nhỏ hơn, có khả năng hoạt đông độc lập được gọi là microservices. Vì các Microservices tách biệt với nhau nên chúng có thể được phát triển, triển khai hay cập nhật và mở rộng quy mô một cách riêng lẻ. Nhờ khả năng này cho phép ta thay đổi các thành phần nhanh chóng và thường xuyên khi cần thiết để theo kịp với các yêu cầu thay đổi nhanh chóng thời nay.

Nhưng với số lượng lớn các thành phần cũng như cơ sở dữ liệu việc cấu hình, quản lý và giữ hệ thống hoạt động trơn tru ngày càng trở nên khó khăn đặc biệt trong việc tối ưu hiệu quả sử dụng tài nguyên. Kubernetes ra đời ể đáp ứng nhu cầu tự động hoá như lập lịch tự động, cấu hình tự động hay giám sát và xử lý lỗi.

Kubernetes cung cấp cho các nhà phát triển khả năng triển khai các ứng dụng một cách thường xuyên mà không cần thông qua nhóm vận hành. Không chỉ dừng lại ở đó Kuberbetes cũng giúp nhóm vận hành tự động theo dõi và khắc phục sự cố.

Đi cùng với sự phát triển lớn mạnh của kiến trúc Microservices cũng như Kubernetes đó là nhu cầu về việc đảm bảo tính an toàn cho các hệ thống này. Trong bài báo cáo này chúng em sẽ giới thiệu về giải pháp đảm bảo an toàn cho Microservices bằng Istio Service Mesh

Chương 1

Giới thiệu về công nghệ Container và kiến trúc Microservices

1.1 Giới thiệu về công nghệ Container

Xây dựng phần mềm theo xu hướng Cloud Native đang phát triển rất nhanh. Trong đó, công nghệ container đóng 1 vai trò rất quan trọng để theo đuổi cách triển khai này.

Công nghệ container, hay gọi đơn giản là container, là một phương pháp đóng gói ứng dụng để ứng dụng có thể chạy với các phụ thuộc của mình (gồm source code và library, runtime, framework...) một cách độc lập, tách biệt với các chương trình khác. Các nhà cung cấp dịch vụ đám mây lớn hiện nay đã cung cấp các dịch vụ dành cho việc quản lý container để hỗ trợ việc xây dựng ứng dụng sử dụng công nghệ container.

1.1.1 Hiểu về công nghệ Container

Container được đặt tên theo thuật ngữ container của ngành vận tải biển vì có cùng chung ý tưởng với nhau. Thay vì chỉ vận chuyển từng sản phẩm, hàng hóa được đặt vào các thùng hàng bằng thép, được thiết kế theo các tiêu chuẩn phù hợp về kích thước và trọng tải để có thể cẩu lên bến tàu và lắp vào con tàu. Như vậy, bằng cách tiêu chuẩn hóa quy trình và nhóm các thành phần liên quan lại với nhau, từng container sẽ được chuyển đi riêng lẻ và sẽ tốn ít chi phí hơn để làm theo cách này.

Trong công nghệ, tình huống cũng khá tương tự. Một chương trình chạy hoàn hảo trên một máy, nhưng khi chuyển sang máy khác thì lại không hoạt động được. Điều này có thể xảy ra khi di chuyển phần mềm từ PC của developer sang test server hoặc từ server vật lý sang cloud server. Các vấn đề phát sinh khi di chuyển phần mềm là do sự khác biệt giữa các môi trường máy tính, chẳng hạn như OS, thư viện SSL, storage, bảo mật và cấu trúc mạng trên các máy khác nhau sẽ khác nhau..

Container giải quyết vấn đề trên bằng cách tạo ra một môi trường bị cô lập (isolated) chứa mọi thứ mà phần mềm cần để có thể chạy được mà không bị các yếu tố liên quan đến môi trường hệ thống làm ảnh hưởng tới cũng như không làm ảnh hưởng tới các phần còn lại của hệ thống.

Giống như việc toàn bộ 1 container sẽ được nhấc lên tàu hoặc xe tải để vận chuyển, công nghệ container cũng như vậy. Container không chỉ chứa phần mềm mà còn chứa các phần phụ thuộc bao gồm các library, binary và file cấu hình cùng với nhau và chúng được di chuyển như một bộ phận, để tránh sự không tương thích và sự cố. Các container giúp việc triển khai phần mềm lên máy chủ thuận lợi hơn.

Như vậy, những container có tác dụng giúp cho một ứng dụng có thể vận hành một cách nhất quán và đáng tin cậy. Bất kể là môi trường hệ điều hành hay cơ sở hạ tầng

nào. Các container thực hiện điều này bằng cách đóng gói mọi thứ mà một dịch vụ cần để có thể chạy được (những thứ như code, runtime, các tool, thư viện và cài đặt), tạo ra một package linh động, độc lập, có khả năng thực thi được.

1.1.2 Container và Máy ảo

Trước khi các container dần được ưa chuộng, máy ảo VM là một phương pháp sử dụng phổ biến. Ở phương pháp này, một máy chủ vật lý có thể được sử dụng cho nhiều ứng dụng thông qua công nghệ ảo hóa, còn được gọi là virtual machine, trong đó mỗi máy ảo chứa toàn bộ hệ điều hành, cũng như các ứng dụng cần thiết để chạy.

Về cấu trúc

VM, hay Virtual Machine/máy ảo là một phiên bản tóm tắt của máy tính, từ hệ điều hành cho đến bộ nhớ và lưu trữ. Image dùng để tạo một VM có thể tương tự hệ điều hành để cài đặt ứng dụng lên hoặc có sẵn tất cả các ứng dụng bạn cần, như web server và database, thậm chí cả chính ứng dụng của bạn. Mỗi VM sẽ hoạt động độc lập hoàn toàn với máy chủ mà VM chạy trên đó, cũng như độc lập với bất kỳ VM nào khác trên máy chủ đó.

Trong khi đó, container sẽ chạy một phần của máy hiện có, chia sẻ kernel của máy chủ đó với bất kỳ container nào khác đang chạy trên hệ thống. Chỉ chứa vừa đủ hệ điều hành và bất kỳ thư viện hỗ trợ nào cần thiết để chạy code. Container được xây dựng từ những image bao gồm mọi thứ nó cần - và không có gì khác (trong TH lý tưởng nhất).

Nhu cầu tài nguyên

Do cấu trúc khác nhau nên nhu cầu để chạy VM và container có thể khác nhau đáng kể. Bởi vì về cơ bản VM tương đương với toàn bộ một máy tính, nên đương nhiên sẽ cần nhiều tài nguyên hơn là một container, trong khi container chỉ cần đến một phần nhỏ nhất của hệ điều hành. Tóm lại, việc mở rộng các container sẽ ít tốn tài nguyên, thời gian, công sức hơn và có thể "xếp" nhiều container hơn trên một máy chủ duy nhất.

Tuy nhiên, cần hết sức lưu ý là vì nhiều dịch vụ có thể "chia sẻ" tài nguyên của một máy ảo duy nhất, có thể có những trường hợp phức tạp trong đó cần thiết phải mở rộng nhiều container để thay thế một máy ảo duy nhất. Điều này dẫn đến việc kiểm soát tài nguyên không còn nhiều ý nghĩa. Ví dụ: nếu bạn tách một máy ảo đơn lẻ thành 50 dịch vụ khác nhau, thì đó là 50 bản sao một phần của hệ điều hành so với một bản sao đầy đủ. Vì vậy, điều quan trọng là cần hiểu chính xác để lựa chọn đúng.

Vậy Container và máy ảo thì đều là những "package". Mỗi container là một package bao gồm ứng dụng của bạn và mọi thứ nó cần để có thể chạy, ngoại trừ hệ điều hành. Máy ảo là một package ứng dụng và mọi thứ nó cần để chạy, bao gồm cả hệ điều hành của nó.

Bạn có thể chạy nhiều container trên một hệ điều hành. Và bạn có thể chạy nhiều máy ảo trên cùng một máy chủ vật lý. Bạn thậm chí có thể chạy container trên máy ảo.

Một lợi thế quan trọng của container so với máy ảo đó là chúng không bao gồm hệ điều hành, container cần ít tài nguyên hệ thống và ít chi phí hơn. Chúng cũng có xu hướng khởi động / tắt nhanh hơn và tính di động cao trong nhiều môi trường khác nhau. Nhưng chúng vẫn sử dụng công suất cơ sở hạ tầng khi không sử dụng, điều này có thể làm tăng chi phí không cần thiết.

1.1.3 Đặc điểm kỹ thuật của Container

Mô hình kiến trúc của container bao gồm các thành phần chính là Server (máy chủ vật lý hoặc máy ảo), host OS (hệ điều hành cài đặt trên server) và các container.

Mỗi một ứng dụng (App A và App B) sẽ có những sự phụ thuộc riêng của nó bao gồm cả về phần mềm (các dịch vụ hay thư viện) lẫn cả về phần cứng (CPU, bộ nhớ, lưu trữ).

Các ứng dụng này sẽ được Container Engine, một công cụ ảo hóa tinh gọn, được cài đặt trên host OS, nó sẽ cô lập sự phụ thuộc của các ứng dụng khác nhau bằng cách đóng gói chúng thành các container. Các tiến trình (process) trong một container bị cô lập với các tiến trình của các container khác trong cùng hệ thống tuy nhiên tất cả các container này đều chia sẻ kernel của host OS (dùng chung host OS).

Với mô hình trên, sự phụ thuộc của ứng dụng vào tầng OS cũng như cơ sở hạ tầng được loại bỏ giúp việc triển khai phương pháp "deploy anywhere" (triển khai ở bất kỳ nơi đâu) của container được hiệu quả hơn. Thêm vào đó, do chia sẻ host OS nên container có thể được tạo gần như một cách tức thì, giúp việc scale-up và scale-down theo nhu cầu được thực hiện một cách nhanh chóng.

1.1.4 Ứng dụng của Container trong thực tế

Các container đại diện cho tương lai của máy tính cùng với các công nghệ như DevOps, cloud native, AI, machine learning. Các trường hợp sử dụng phổ biến bao gồm:

- Hiện đại hóa các ứng dụng hiện có trên đám mây
- Tạo các ứng dụng mới tối đa hóa lợi ích của container
- Cô lập, triển khai, mở rộng quy mô và hỗ trợ microservices và các ứng dụng phân tán
- Tăng cường hiệu quả DevOps, hiệu quả thông qua việc build/test/triển khai được sắp xếp một cách hợp lý
- Cung cấp cho nhà phát triển một môi trường sản xuất nhất quán, tách biệt khỏi các ứng dụng và quy trình khác
- Đơn giản hóa và tăng tốc các chức năng có tính lặp đi lặp lại

Tạo điều kiện thuận lợi cho các môi trường máy tính kết hợp với multi-cloud, vì các container có thể chạy nhất quán ở bất kỳ đâu.

1.1.5 Về Containerization

Containerization là hành động tạo một container, bao gồm việc chỉ lấy ra ứng dụng hay dịch vụ mà bạn cần chạy, cùng với các cấu hình và những phần phụ thuộc của nó, đồng thời rút nó ra khỏi hệ điều hành và cơ sở hạ tầng bên dưới. Sau đó, cho ra kết quả là container image có thể chạy trên bất kỳ nền tảng container nào.

Nhiều container có thể được chạy trên cùng một máy chủ và chia sẻ cùng một hệ điều hành với các container khác, mỗi container chạy các quy trình biệt lập trong không gian được bảo mật riêng của nó. Bởi vì các container chia sẻ base OS (hệ điều hành), do vậy kết quả là có thể chạy mỗi container bằng cách sử dụng một lượng tài nguyên rất ít, ít hơn đáng kể so với việc sử dụng số lượng máy ảo (VM) riêng biệt.

Những lợi ích của Container

Tốn rất ít dung lượng

Các container chia sẻ kernel của máy chủ lưu trữ, chúng chỉ chứa các thành phần thực sự cần thiết với hệ điều hành và thư viện. Đồng thời các container thường cũng chỉ giới hạn ở một chức năng duy nhất, nên có kích thước rất nhỏ. Nhờ vậy, việc xây dựng, triển khai cực kỳ nhanh chóng.

Bởi vì chúng được tách biệt khỏi lớp OS nên việc container chạy hiệu quả và nhẹ về tài nguyên hơn so với máy ảo cũng là điều dễ hiểu.

Các Container có tính linh hoạt

Vì container bao gồm có tất cả các cấu hình cần thiết và các thành phần phụ thuộc, do vậy bạn có thể viết một lần và di chuyển giữa các môi trường. Có một câu thần chú nổi tiếng đó là "Build once, run everywhere".

- **Triển khai nhanh:**

  Do kích thước nhỏ, các container có thể chỉ cần vài giây để khởi động, thậm chí là ít hơn, nên rất thích hợp cho các ứng dụng cần được đẩy lên và xuống liên tục, chẳng hạn như các ứng dụng "serverless".

- **CI/CD:**

  CI là tên viết tắt của Continuous Integration, theo nghĩa tiếng Việt là tích hợp liên tục. Quá trình hoạt động cho phép các thành viên trong một team liên tục lưu trữ những mã mới vào một kho nhất định. Nhờ vào số lượng dữ liệu này, CI sẽ tự động chạy test và kiểm tra độ chính xác. Cùng lúc đó cũng hỗ trợ phát triển phần mềm một cách nhanh chóng hơn bằng việc báo lỗi sai và đưa ra gợi ý giải quyết.

  CD là tên viết tắt của Continuous Delivery, nghĩa là quá trình chuyển giao liên tục. Về cơ bản, CD cũng sở hữu những kỹ năng của CI, tuy nhiên sẽ phức tạp và nâng cao hơn một chút.

  Trong khi CI chỉ chạy và kiểm tra những code đã có sẵn, CD thậm chí còn tự sửa code đã được build và test nếu phát hiện lỗi sai. Ngoài ra, nó cũng tự động thay đổi môi trường testing hoặc staging để nâng cao chất lượng kiểm tra.

  Chính vì các container được thiết kế để có thể start và restart thường xuyên, nhờ vậy mà dễ dàng tiếp nhận các thay đổi tạo điều kiện vô cùng phù hợp để triển khai CI/CD.

- **Khả năng mở rộng:**

  Do kích thước nhỏ của chúng, các container có thể dễ dàng lấy ra, mở rộng quy mô trong quá trình vận hành, tắt đi khi không sử dụng và nhanh chóng khởi động lại khi cần thiết.

- **Tiết kiệm chi phí:**

  Thông qua việc giảm lượng nhu cầu về tài nguyên và mở rộng quy mô một cách thông minh, các container cung cấp một giải pháp linh hoạt, nhanh chóng và tiết kiệm chi phí.

Khả năng chịu lỗi cao

Các nhóm phát triển phần mềm sử dụng bộ chứa để xây dựng các ứng dụng có khả năng chịu lỗi cao. Họ sử dụng nhiều bộ chứa để chạy vi dịch vụ trên đám mây. Bởi vì các vi dịch vụ trong bộ chứa hoạt động trong không gian người dùng riêng biệt, một bộ chứa bị lỗi riêng lẻ sẽ không ảnh hưởng đến các bộ chứa khác. Điều này làm tăng khả năng phục hồi và tính khả dụng của ứng dụng.

Quản lý ít cơ sở hạ tầng hơn

Container buộc bạn phải nắm bắt được những gì mà bạn thực sự cần qua đó mang lại trải nghiệm tốt nhất cho khách hàng của bạn. Điều này giúp quản lý cơ sở hạ tầng tốt hơn vì đơn giản là có ít cơ sở hạ tầng hơn để quản lý.

Container tạo ra khả năng tập trung

Các teams IT sẽ dành ít thời gian hơn cho các hệ điều hành và phần cứng, điều đó cho phép họ tập trung hơn vào các dự án quan trọng của doanh nghiệp.

Thúc đẩy sự phát triển

Container cung cấp một môi trường ổn định, dễ dàng dự đoán được, nơi CPU/memory được tối ưu hóa và code thì được trừu tượng hóa từ cơ sở hạ tầng để có tính khả chuyển.

Tạo điều kiện cho những kiến trúc hiện đại

Sử dụng container, các nhà phát triển có thể chia các ứng dụng thành các microservices, điều này có thể tăng tốc độ phát triển và khi được triển khai thì được mở rộng một cách riêng biệt.

1.1.6 Những thách thức trong việc sử dụng Container

Container còn tương đối mới

Kubernetes được phát hành lần đầu tiền vào năm 2014 và nhanh chóng được thị trường đón nhận. Việc trở thành một "hot tech" có thể gây khó khăn trong việc tìm kiếm những người có kinh nghiệm và biết cách làm việc với nhiều môi trường container.

Không phải dịch vụ nào cũng được hỗ trợ Container hóa

Nếu ứng dụng của bạn đưa vào các dịch vụ không hỗ trợ container. Bạn có thể cần phải đầu tư nhiều để chuyển đổi nó thành một giải pháp container.

Container yêu cầu nhiều thay đổi về quy trình và kỹ năng

Container có thể đẩy nhanh quá trình chuyển đổi của bạn sang kiểu phát triển agile hay eficient, nhưng điều này cũng đồng nghĩa với với việc tạo ra một thay đổi lớn đối với các quy trình hiện có như quy trình phát triển, triển khai, review và giám sát. Dẫn đến việc các team mà tổ chức hiện có cần phải được điều chỉnh và đào tạo lại.

- **Có thể yêu cầu kết nối mạng phức tạp:**

  Thường thường các chức năng sẽ được chia thành nhiều container và cần phải giao tiếp với nhau. Việc số lượng rất nhiều các container phải giao tiếp với nhau

có thể phức tạp. Một số hệ thống điều phối như Kubernetes có các multi-container pods giúp việc trao đổi dễ dàng hơn một chút, nhưng được cho là vẫn phức tạp hơn so với sử dụng máy ảo. Thực tế thì mô hình mạng L3 trong Kubernetes đơn giản hơn nhiều so với mô hình L2 trong hạ tầng máy ảo OpenStack. Vì vậy, vấn đề nằm ở chỗ cần xác định được việc giao tiếp xảy ra giữa các chức năng hay giữa các máy ảo

- **Có thể cần thao tác nhiều hơn so với máy ảo:**

  Nếu sử dụng container, bạn sẽ phân tách ứng dụng thành các dịch vụ thành phần khác nhau, mặc dù việc này có ích lợi nhưng lại không cần thiết khi sử dụng VM.

- **Công nghệ đang phát triển với tốc độ chóng mặt:**

  Điều này không chỉ dành riêng cho container, nhưng về bản chất thì nhịp độ nhanh của công nghệ container có nghĩa là bạn cần mọi người (hoặc đối tác) có mặt để đưa ra quyết định đúng đắn, giảm rủi ro và đảm bảo việc triển khai không bị cản trở bởi tính trì trệ của công ty.

- **Container không phải là một viên đạn thần kỳ:**

  Đọc lướt qua một loạt các lợi ích thì trông có vẻ container là một thứ lý tưởng, nhưng hãy cẩn thận vì bất kỳ quá trình chuyển đổi nào cũng cần phải suy nghĩ nghiêm túc. Bạn phải biết mình đang làm gì, những gì sẽ mang lại hiệu quả và ngược lại. Hoặc tìm một ai có thể giúp bạn vượt trong sự chuyển đổi đó.

1.2 Giới thiệu về kiến trúc Microservices

Một vài năm trở lại đây, khái niệm kiến trúc Microservices hiện là chủ đề rất hot trong cộng động lập trình viên. Thật không khó để có thể tìm thấy một bài viết, một bản báo cáo hay một bài thuyết trình về chủ đề này. Vậy Microservices là gì? Ưu điểm và nhược điểm của kiến trúc Microservices ra sao?

1.2.1 Khái niệm Microservices

Trong tiếng anh, micro có nghĩa là nhỏ, vi mô. Vậy Microservice, như tên của nó, đó chính là chia một khối phần mềm thành các service nhỏ hơn, có thể triển khai trên các server khác nhau. Mỗi service sẽ xử lý từng phần công việc và được kết nối với nhau thông qua các các giao thức khác nhau, như http, SOA, socket, Message queue (Active MQ, Kafka)... để truyền tải dữ liệu.

Trước khi Microservices xuất hiện, các ứng dụng thường phát triển theo mô hình Monolithic architecture (Kiến trúc một khối). Có nghĩa là tất cả các module (view, business, database) đều được gộp trong một project, một ứng dụng được phát triển theo mô hình kiến trúc một khối thường được phân chia làm nhiều module. Nhưng khi được đóng gói và cài đặt sẽ thành một khối (monolithic). Lợi ích của mô hình kiến trúc một khối đó là dễ dàng phát triển và triển khai. Nhưng bên cạnh đó nó cũng có nhiều hạn chế ví dụ như khó khăn trong việc bảo trì, tính linh hoạt và khả năng mở rộng kém, đặc biệt với những ứng dụng doanh nghiệp có quy mô lớn. Đó chính là lí do ra đời của kiến trúc Microservices.

1.2.2 Những đặc điểm của Microservices

Decoupling: Các service trong một hệ thống phần lớn được tách rời. Vì vậy, toàn bộ ứng dụng có thể dễ dàng được xây dựng, thay đổi và thu nhỏ.

Componentization: Microservices được coi là các thành phần độc lập có thể dễ dàng thay thế và nâng cấp.

Business Capabilities: Mỗi một thành phần trong kiến trúc microservice rất đơn giản và tập trung vào một nhiệm vụ duy nhất.

Autonomy: Các lập trình viên hay các nhóm có thể làm việc độc lập với nhau trong quá trình phát triển.

Continous Delivery: Cho phép phát hành phần mềm thường xuyên, liên tục.

Decentralized Governance: Không có mẫu chuẩn hóa hoặc bất kỳ mẫu công nghệ nào. Được tự do lựa chọn các công cụ hữu ích tốt nhất để có thể giải quyết vấn đề.

Agility: Microservice hỗ trợ phát triển theo mô hình Agile.

1.2.3 Ưu điểm của Microservices

Kiến trúc Microservices được sinh ra để khắc phục những hạn chế của kiến trúc một khối.

- Hoạt động độc lập, linh hoạt, có tính chuyên biệt cao: Do không bị ràng buộc bởi những yêu cầu chung, nên mỗi service nhỏ có thể tự do lựa chọn công nghệ, nền tảng phù hợp. Tất cả các service có thể được phát triển dễ dàng dựa trên chức năng cá nhân của từng service. Có thể chia nhỏ để phát triển độc lập.
- Nâng cao khả năng xử lý lỗi: Với mô hình này, một service bất kỳ nào gặp lỗi sẽ không gây ra ảnh hưởng đối với những bộ phận còn lại. Việc khắc phục lỗi trên quy mô hẹp cũng sẽ được tiến hành một cách dễ dàng, khi một service của ứng dụng không hoạt động, hệ thống vẫn tiếp tục hoạt động.
- Independent Deployment: Có thể được triển khai riêng lẻ trong bất kỳ ứng dụng nào.
- Mixed Technology Stack: Các ngôn ngữ và công nghệ khác nhau có thể được sử dụng để xây dựng các service khác nhau của cùng một ứng dụng.
- Thuận tiện trong nâng cấp, mở rộng: Tương tự như trường hợp xử lý lỗi, việc nâng cấp, bảo trì service hoàn toàn độc lập sẽ không làm gián đoạn quá trình vận hành của cả phần mềm. Nhờ vậy, những phiên bản mới có thể được cập nhật thường xuyên.
- Đơn giản hóa trong quản lý và kiểm thử: Với từng service nhỏ, các bước quản lý, tính toán và kiểm soát, xử lý lỗi sẽ trở nên đơn giản và nhanh chóng hơn so với cả phần mềm.

Kiến trúc Microservices giúp đơn giản hóa hệ thống, chia nhỏ hệ thống ra làm nhiều service nhỏ lẽ dễ dàng quản lý và triển khai từng phần so với kiến trúc nguyên khối. Phân tách rõ ràng giữa các service nhỏ. Cho phép việc mỗi service được phát triển độc lập. Cũng như cho phép lập trình viên có thể tự do chọn lựa technology stack cho mỗi service

mình phát triển. mỗi service có thể được triển khai một cách độc lập (VD: Mỗi service có thể được đóng gói vào một docker container độc lập, giúp giảm tối đa thời gian deploy). Nó cũng cho phép mỗi service có thể được scale một cách độc lập với nhau. Việc scale có thể được thực hiện dễ dàng bằng cách tăng số instance cho mỗi service rồi phân tải bằng load balancer.

1.2.4 Các nhược điểm của kiến trúc Microservice

Kiến trúc Microservices đang là một xu hướng, nhưng nó cũng có nhược điểm của nó.

Microservice khuyến khích làm nhỏ gọn các service, nhưng khi chia nhỏ sẽ dẫn đến những thứ vụn vặt, khó kiểm soát. Hơn nữa chính từ đặc tính phân tán khiến cho các lập trình viên phải lựa chọn cách thức giao tiếp phù hợp khi xử lí request giữa các service. Trong trường hợp dự án quá lớn, số lượng service nhiều, chia nhỏ rời rạc, thiếu tính liên kết. Cùng với cách thức liên kết thông tin giữa qua môi trường mạng, việc trao đổi giữa các service càng trở nên khó khăn. Đôi khi, các lỗi kết nối cũng có thể xảy ra khiến việc trao đổi này bị gián đoạn.

Việc liên tục di chuyển qua các database khác nhau sẽ khiến dữ liệu bị đảo lộn, không còn nguyên vẹn, thậm chí phải đối mặt với nguy cơ an ninh, bị đánh cắp.

Hơn nữa việc quản lí nhiều database, và transaction giữa các service trong một hệ thống phân tán cũng là một khó khăn không nhỏ. Hay khi thực hiện test một service, bạn cũng cần test các service có liên quan. Gây khó khăn trong quá trình mở rộng, phát triển. Khi phần mềm được phát triển với quy mô lớn hơn, số lượng service cũng trở nên nhiều hơn. Các lập trình viên không chỉ mất thời gian tính toán chính xác kích thước của từng service, mà còn gặp khó khăn khi sử dụng những công cụ hỗ trợ tự động mã nguồn mở bên ngoài khác.

Triển khai microservice cũng sẽ phức tạp hơn so với ứng dụng kiến trúc một khối, cần sự phối hợp giữa nhiều service, điều này không đơn giản như việc triển khai WAR trong một ứng dụng kiến trúc một khối.

1.2.5 Những yêu cầu bắt buộc khi phát triển phần mềm theo kiến trúc Microservice

Để phát triển một phần mềm theo mô hình kiến trúc Microservice, lập trình viên cần đảm bảo một số yếu tố chính như sau:

- Xây dựng hệ cơ sở dữ liệu (database) độc lập.
- Xác định kích thước service phù hợp.
- Đề ra vai trò, chức năng cụ thể, riêng biệt cho từng service.

Việc phát triển một phần mềm theo mô hình kiến trúc Microservice chưa bao giờ là điều đơn giản.

1.3 Giới thiệu về Kubernetes

1.4 Một số vấn đề bảo mật Microservices trên Kubernetes

Kết luận Chương 1

Chương 2

Tổng quan về Istio

2.1 Tổng quan về Istio

2.1.1 Tổng quan về Service Mesh

2.1.2 Kiến trúc của Istio

2.1.3 Tổng quan về Envoy Proxy

2.2 Quản lý mạng giữa các Microservices với Istio

2.2.1 Tổng quan về Istio Ingress Gateway

2.2.2 Định tuyến trong Istio

2.2.3 Giải quyết các vấn đề về mạng trong Microservices

2.3 Giám sát các Microservices với Istio

2.3.1 Một số Metrics quan trọng của Istio

2.3.2 Giám sát các lưu lượng mạng qua Jaeger và Kiali

2.4 Bảo mật các Microservices bằng Istio

2.4.1 Xác thực giữa các Microservices với Istio

2.4.2 Phân quyền cho các Microservices với Istio

Kết luận chương 2

Chương 3

Triển khai Istio trên Kubernetes

3.1 Mô hình triển khai

3.2 Kịch bản triển khai

3.3 Thực nghiệm

3.4 Kết luận