Demo báo cáo - không

Truyen thong cong nghiep (Trường Đại học Công nghệ thông tin và truyền thông, Đại học Thái Nguyên)

BÁO CÁO TÌM HIỂU VỀ MICROSERVICES

I TỔNG QUAN VỀ ĐỀ TÀI

Giới thiệu dự án

Dự án "Tìm hiểu về Microservices" nhằm khám phá chi tiết về mô hình kiến trúc phần mềm tiên tiến này. Chúng tôi sẽ nghiên cứu các khía cạnh quan trọng như thiết kế, triển khai, tương tác giữa các dịch vụ, hiệu năng và quản lý lỗi, nhằm tạo ra cái nhìn toàn diện về việc áp dụng Microservices trong lĩnh vực công nghệ thông tin.

1.2. Thông tin chung

Tên dự án: Tìm hiểu về Microservices

Sinh viên thực hiện: Nhóm 8.

Lớp KTPM-K19B

Loại phần mềm: Kiến trúc phần mềm Microservice và Blog đặt hàng sản phẩm trực tuyến

Thời gian: … - …

1.3. Người tham gia

1.3.1. Giảng viên hướng dẫn

	Học hàm, họ và tên	Số điện thoại	E-mail
Giảng viên hướng dẫn	Ths.Nguyễn Thu Phương		

1.3.2. Sinh viên thực hiện

	Họ và tên	Số điện thoại	E-mail
Sinh viên thực hiện	Nguyễn Đức Tuấn		

	Nguyễn Anh Vũ		
	Hoàng Tùng Dương		
	Hoàng Huyền Diệp		
	Nguyễn Hải Đăng		
	Hà Trung Tuyến		

1.4 Đặt vấn đề

Vấn đề về Microservices là một chủ đề quan trọng trong lĩnh vực công nghệ thông tin, liên quan đến việc xây dựng kiến trúc phần mềm dựa trên các dịch vụ nhỏ và độc lập. Dưới đây là tóm tắt về vấn đề này:

Vấn đề: Microservices trong Kiến Trúc Phần Mềm Microservices là mô hình kiến trúc phần mềm chia ứng dụng thành các dịch vụ nhỏ, độc lập với khả năng thực hiện chức năng cụ thể. Điều này tạo ra một số vấn đề cần được tìm hiểu và nghiên cứu.

Các Khía Cạnh Quan Trọng:

Ưu điểm và Nhược điểm: Nghiên cứu về ưu điểm như tăng cường tích hợp linh hoạt, mở rộng dễ dàng, cùng với nhược điểm như quản lý phức tạp, sự phụ thuộc giữa các dịch vụ.

Tương Tác và Tích Hợp: Khám phá cách các dịch vụ Microservices tương tác, giao tiếp thông qua giao thức và API. Xem xét cách quản lý sự phụ thuộc và tương tác giữa chúng.

Hiệu Năng và Mở Rộng: Nghiên cứu cách tối ưu hiệu suất từng dịch vụ, cách quản lý tải và mở rộng hệ thống khi cần.

Bảo Mật và Quản Lý Lỗi: Đặt vấn đề về cách bảo mật dịch vụ, quản lý lỗi, khắc phục sự cố trong môi trường Microservices.

Kiến Thức và Kỹ Năng: Khám phá kiến thức và kỹ năng cần thiết cho việc phát triển và quản lý các dịch vụ Microservices.

Sự Phụ Thuộc và Độc Lập: Đặt vấn đề về cách quản lý sự phụ thuộc và đảm bảo tính độc lập giữa các dịch vụ.

Phù Hợp với Dự Án và Tổ Chức: Nghiên cứu về việc áp dụng Microservices trong các dự án và tổ chức khác nhau.

Tầm Quan Trọng: Vấn đề về Microservices không chỉ liên quan đến việc xây dựng ứng dụng mà còn đến việc tối ưu hóa, quản lý và phát triển hiệu quả. Sự hiểu biết sâu về các khía cạnh của Microservices sẽ đóng vai trò quan trọng trong việc tạo ra các ứng dụng phần mềm linh hoạt, dễ mở rộng và hiệu quả trong thế giới công nghệ ngày nay.

1.5 Lý do chọn đề tài

Kiến trúc Microservices đem lại nhiều lợi ích cho việc phát triển và quản lý ứng dụng hiện đại. Bằng cách phân tách ứng dụng thành các thành phần nhỏ hơn và độc lập, kiến trúc Microservices tăng tính linh hoạt, dễ dàng mở rộng, và quản lý. Việc triển khai kiến trúc Microservices yêu cầu xác định rõ các domain, chọn công nghệ phù hợp, xây dựng và triển khai từng microservice, sau đó tích hợp chúng lại thành một hệ thống hoàn chỉnh. Những trang web nổi tiếng và quen thuộc với tất cả chúng ta như Netflix, Amazon,… đều sử dụng kiến trúc Microservices để xây dựng và phát triển.

Ứng dụng của Microservices trong công nghệ thông tin nó dùng để phát các dự án app, website ví dụ như: App native, xây dựng và thiết kế web API, Phát triển, mở rộng và tích hợp với module IoT, sunteco cloud – scalable application platform, …

1.6 Mục tiêu nghiên cứu

Mục tiêu của bài báo cáo là nắm vững kiến thức về Microservices, tìm hiểu ưu điểm, nhược điểm, và phân tích cách triển khai hệ thống sử dụng kiến trúc Microservices. Hiểu rõ về kiến trúc Microservices và cách nó hoạt động.

Phân tích ưu điểm và nhược điểm của kiến trúc Microservices.

Thiết kế, triển khai và quản lý các dịch vụ Microservices độc lập.

Nghiên cứu cách tích hợp và giao tiếp giữa các dịch vụ Microservices.

Tối ưu hiệu năng và khả năng mở rộng của hệ thống dựa trên Microservices.

Tìm hiểu về bảo mật và quản lý lỗi trong môi trường Microservices.

Áp dụng thực tế và tìm hiểu các ví dụ cụ thể về việc sử dụng Microservices.

Đánh giá hiệu quả và tiềm năng phát triển của kiến trúc Microservices trong công nghệ thông tin.

1.7 Đối tượng nghiên cứu

Đối tượng nghiên cứu trong lĩnh vực Microservices là hệ thống kiến trúc phần mềm dựa trên mô hình Microservices. Điều này bao gồm:

Các dịch vụ Microservices: Các thành phần riêng lẻ của hệ thống, mỗi dịch vụ thực hiện một chức năng cụ thể. Đối tượng nghiên cứu bao gồm cách thiết kế, triển khai và quản lý các dịch vụ này.

Tương tác giữa dịch vụ: Cách các dịch vụ Microservices tương tác với nhau thông qua giao thức và API. Điều này liên quan đến tích hợp, giao tiếp và quản lý sự phụ thuộc giữa các dịch vụ.

1.8 Phạm vi nghiên cứu

Phạm vi nghiên cứu về Microservices bao gồm:

Xác định khái niệm và nguyên tắc cơ bản của kiến trúc Microservices.

Khám phá ưu điểm và nhược điểm của việc sử dụng Microservices trong phát triển phần mềm.

Nghiên cứu cách thiết kế, triển khai và quản lý các dịch vụ Microservices.

Tìm hiểu cách tích hợp và giao tiếp giữa các dịch vụ Microservices.

Đánh giá cách tối ưu hiệu năng và khả năng mở rộng của hệ thống dựa trên Microservices.

Nghiên cứu về bảo mật, quản lý lỗi và khả năng ổn định của hệ thống Microservices.

Áp dụng thực tế và tìm hiểu các ví dụ cụ thể về việc triển khai và sử dụng Microservices.

Đánh giá tiềm năng phát triển và ứng dụng của kiến trúc Microservices trong lĩnh vực công nghệ thông tin.

Phạm vi này giúp tập trung vào việc hiểu rõ, nghiên cứu và áp dụng kiến thức về Microservices trong các khía cạnh quan trọng của phát triển và quản lý phần mềm.

1.9 Phương pháp nghiên cứu

Phương pháp nghiên cứu về Microservices bao gồm:

Xác định mục tiêu nghiên cứu: Đặt ra mục tiêu nghiên cứu và xác định phạm vi cụ thể mà bạn muốn tập trung.

Thu thập tài liệu: Tìm hiểu về Microservices qua việc thu thập sách, bài báo, tài liệu hướng dẫn và nguồn trực tuyến.

Nghiên cứu thư viện công cụ: Hiểu về các công cụ như Docker, Kubernetes mà người ta thường sử dụng cho Microservices.

Nghiên cứu các dự án thực tế: Tìm hiểu về các dự án đã triển khai Microservices để hiểu cách họ ứng dụng và giải quyết thách thức.

Phân tích các ví dụ cụ thể: Nghiên cứu và phân tích các ví dụ về triển khai và quản lý Microservices để hiểu cách chúng tương tác.

Thực hiện thử nghiệm: Xây dựng các dịch vụ Microservices thử nghiệm để áp dụng kiến thức và kiểm tra hệ thống.

Phân tích kết quả thử nghiệm: Đánh giá hiệu suất, hiệu năng và khả năng mở rộng của hệ thống Microservices.

So sánh với các kiểu kiến trúc khác: Đối chiếu Microservices với các kiểu kiến trúc khác để đánh giá ưu điểm và nhược điểm.

Viết báo cáo nghiên cứu: Tạo báo cáo với cấu trúc gồm giới thiệu, phân tích, kết quả thử nghiệm, thảo luận và kết luận.

Kiểm tra và đánh giá: Đảm bảo báo cáo rõ ràng, logic và thuyết phục trước khi hoàn thành.

Phương pháp này giúp bạn nắm vững và phân tích chi tiết các khía cạnh quan trọng của kiến trúc Microservices trong nghiên cứu của mình.

1.10 Nội dung nghiên cứu

Nội dung nghiên cứu về Microservices bao gồm các khía cạnh cơ bản và chi tiết về kiến trúc này trong lĩnh vực công nghệ thông tin. Dưới đây là tóm tắt nội dung nghiên cứu của Microservices:

Khái niệm và nguyên tắc cơ bản:

Giới thiệu về Microservices là gì, lý do tại sao kiến trúc này được phát triển.

Đặc điểm cơ bản của Microservices: phân tách, độc lập, tương tác thông qua giao thức và API.

Ưu điểm và nhược điểm:

Phân tích ưu điểm của việc sử dụng Microservices: tăng tốc độ phát triển, tích hợp linh hoạt, mở rộng dễ dàng.

Đánh giá nhược điểm:

Quản lý phức tạp, khó khăn trong việc quản lý sự phụ thuộc.

Thiết kế và triển khai dịch vụ Microservices:

Xác định cách thiết kế dịch vụ Microservices độc lập, chia nhỏ chức năng cụ thể.

Nghiên cứu cách triển khai dịch vụ, sử dụng công cụ như Docker, Kubernetes để quản lý và mở rộng.

Tương tác và giao tiếp giữa các dịch vụ:

Khám phá cách tích hợp và giao tiếp giữa các dịch vụ Microservices qua giao thức và API.

Xem xét cách quản lý sự phụ thuộc và cách dịch vụ tương tác.

Tối ưu hiệu năng và khả năng mở rộng:

Nghiên cứu về cách tối ưu hiệu suất của từng dịch vụ và cách mở rộng hệ thống khi cần thiết.

Khám phá cách quản lý tải và đảm bảo hệ thống đáp ứng nhu cầu người dùng.

Bảo mật và quản lý lỗi:

Nghiên cứu cách đảm bảo an ninh cho các dịch vụ Microservices, bao gồm xác thực và phân quyền.

Xem xét cách xử lý lỗi và sự cố để đảm bảo tính ổn định của hệ thống.

Áp dụng thực tế và ví dụ cụ thể:

Tìm hiểu về các dự án thực tế sử dụng kiến trúc Microservices.

Nghiên cứu về những thử thách và thành công mà các tổ chức gặp phải khi triển khai Microservices.

Tiềm năng phát triển và ứng dụng tương lai:

Đánh giá tiềm năng phát triển và những xu hướng mới trong việc sử dụng kiến trúc Microservices.

Nội dung nghiên cứu này giúp bạn hiểu rõ và thấu hiểu về cách áp dụng, thiết kế, triển khai và quản lý Microservices trong các ứng dụng công nghệ thông tin.

II. TÌM HIỂU VỀ MICROSERVICES

2.1 Tổng quan về kiến trúc Microservices

Kiến trúc Microservices là một phương pháp xây dựng ứng dụng phần mềm bằng cách tách thành nhiều dịch vụ nhỏ và độc lập, được gọi là "microservices". Mỗi microservice thực hiện một nhiệm vụ cụ thể và có thể phát triển, triển khai, và quản lý độc lập. Đây là một tiếp cận tương phản với kiến trúc monolithic truyền thống, nơi toàn bộ ứng dụng được triển khai như một đơn vị duy nhất.

Trong kiến trúc Microservices, mỗi dịch vụ nhỏ tập trung vào một chức năng cụ thể và có thể sử dụng ngôn ngữ và công nghệ khác nhau. Các dịch vụ này tương tác thông qua giao diện API, cho phép hệ thống được linh hoạt, dễ dàng mở rộng và quản lý. Kiến trúc Microservices tạo thuận lợi cho việc phát triển song song, giảm thiểu tác động giữa các thành phần và cho phép sử dụng công nghệ mới một cách linh hoạt.

Tuy kiến trúc Microservices mang lại nhiều lợi ích, nhưng cũng đòi hỏi kiến thức về quản lý, tích hợp và triển khai hiệu quả. Việc phân chia dịch vụ và quản lý chúng đòi hỏi sự tập trung vào thiết kế đúng và việc quản lý nhiều thành phần. Mặc dù có những thách thức, kiến trúc Microservices đã trở thành một phương pháp phổ biến để xây dựng ứng dụng phức tạp và đa dạng, giúp cải thiện hiệu suất, mở rộng dễ dàng và đáp ứng linh hoạt với yêu cầu thay đổi.

2.2 Lịch sử và nguồn gốc phát triển

Nguồn Gốc và Tiền Thân:

Trước khi Microservices xuất hiện, kiến trúc phần mềm thường được xây dựng dưới dạng monolithic, trong đó toàn bộ ứng dụng được phát triển và triển khai như một thể thống nhất.

Sự phát triển của ứng dụng monolithic gặp phải một số vấn đề như khó khăn trong việc mở rộng, bảo trì và cập nhật.

Sự Xuất Hiện Của Microservices:

Ý tưởng về Microservices xuất phát từ ý niệm về "Service-Oriented Architecture" (SOA), một kiến trúc được đề xuất để tạo ra các dịch vụ phụ thuộc nhỏ hơn để phục vụ các chức năng cụ thể.

Microservices bắt đầu xuất hiện vào cuối những năm 2000, khi các công ty công nghệ lớn như Amazon, Netflix, và eBay bắt đầu áp dụng kiến trúc này để giải quyết các thách thức trong việc xây dựng và phát triển ứng dụng quy mô lớn.

Các Sự Kiện Quan Trọng:

Amazon Web Services (AWS) được ra mắt vào năm 2002, cung cấp khả năng tạo ra các dịch vụ web phục vụ cho ứng dụng.

Netflix chuyển từ kiến trúc monolithic sang Microservices vào năm 2009 để nâng cao khả năng mở rộng và độ tin cậy.

Martin Fowler, một chuyên gia hàng đầu về phát triển phần mềm, viết bài viết "Microservices" vào năm 2014, giới thiệu rộng rãi khái niệm và lợi ích của kiến trúc này.

Containerization và công nghệ như Docker được phát triển vào những năm gần đây, cung cấp môi trường cô lập để triển khai và vận hành các dịch vụ Microservices.

Lợi Ích và Phổ Biến Hóa:

Microservices giúp tách biệt các thành phần của ứng dụng, tạo thuận lợi cho việc phát triển độc lập và mở rộng từng dịch vụ riêng biệt.

Kiến trúc này giúp tăng tính linh hoạt, khả năng mở rộng, và tối ưu hóa việc triển khai.

Ngày nay, Microservices đã trở thành một phương pháp phổ biến trong việc xây dựng các ứng dụng phức tạp và quy mô lớn, được sử dụng rộng rãi trong các lĩnh vực công nghệ thông tin và phát triển phần mềm.

2.3 Định nghĩa và bản chất của Microservices

Không có định nghĩa phổ quát về thuật ngữ microservice. Định nghĩa đơn giản nhất của microservice, còn được gọi là kiến trúc microservice (microservice architecture), là một kiểu kiến trúc ứng dụng sử dụng các dịch vụ được liên kết lỏng lẻo. Các dịch vụ này có thể được phát triển, triển khai và duy trì độc lập.

Chúng hoạt động với tốc độ nhanh hơn và đáng tin cậy hơn nhiều so với các ứng dụng nguyên khối, phức tạp truyền thống. Sử dụng iến trúc microservice, một tổ chức có kích thước bất kỳ có thể phát triển các ngăn xếp công nghệ phù hợp với khả năng của họ.

Có nhiều lợi ích hữu hình khi sử dụng microservice, chúng ta sẽ thảo luận sau, nhưng vẫn còn một số tranh cãi về việc liệu các cơng ty có nên chuyển từ kiến trúc nguyên khối sang kiến trúc microservice hay không. Hãy xem xét sự khác biệt giữa hai kiến trúc để hiểu cuộc tranh luận.

2.4 Các tính năng của Microservices

Tách biệt - Các dịch vụ trong một hệ thống phần lớn được tách biệt. Vì vậy, tồn bộ ứng dụng có thể dễ dàng được xây dựng, thay đổi và mở rộng quy mơ

Thành phần hóa - Các dịch vụ vi mơ được coi như các thành phần độc lập có

thể dễ dàng thay thế và nâng cấp

Khả năng kinh doanh - Dịch vụ vi mô rất đơn giản và tập trung vào một khả năng duy nhất

Quyền tự chủ - Các nhà phát triển và nhóm có thể làm việc độc lập với nhau, do đó tăng tốc độ

Phân phối liên tục - Cho phép phát hành phần mềm thường xun, thơng qua hệ thống tự động hóa việc tạo, kiểm tra và phê duyệt phần mềm

Trách nhiệm - Microservices không tập trung vào các ứng dụng như các dự án. Thay vào đó, họ coi các ứng dụng là sản phẩm mà họ chịu trách nhiệm

Quản trị phi tập trung - Trọng tâm là sử dụng đúng công cụ cho đúng cơng việc. Điều đó có nghĩa là khơng có khn mẫu tiêu chuẩn hóa hoặc bất kỳ khn mẫu cơng nghệ nào. Các nhà phát triển có quyền tự do lựa chọn các cơng cụ hữu ích tốt nhất để giải quyết vấn đề của họ

Nhanh nhẹn - Mọi tính năng mới có thể được phát triển nhanh chóng và lại bị loại bỏ

2.5 Tại sao dùng kiến trúc Microservice

Khi phát triển phiên bản đầu tiên của ứng dụng, bạn thường không gặp phải các vấn

đề mà Microservices giải quyết. Hơn nữa, sử dụng một kiến trúc phân tán, phức tạp sẽ

làm chậm quá trình phát triển. Đây là một vấn đề rất lớn đối với các start-up vì họ cần phát triển nhanh mơ hình kinh doanh cùng ứng dụng đi kèm.
Vì vậy, theo tơi, trừ khi bạn có một hệ thống quá phức tạp để quản lý bằng Monolithic Architecture, hoặc bạn xác định tương lai của ứng dụng sẽ trở nên như vậy. Thì kiến trúc Monolithic vẫn đủ tốt đối với bạn.

2.6 Cân nhắc lựa chọn Microservice

Khi thực hiện dự án microservice cần tính tốn thật kỹ kích thước của 1 microservice
Về tính bảo mật: Khi người lập trình viên lựa chọn việc sử dụng mã nguồn mở hay sử dụng công cụ hỗ trợ khác

Về hiệu năng: Các microservice thường (nên) được triển khai bên trong docker container và giao tiếp với nhau qua REST API. Việc này làm hiệu năng của tồn bộ chương trình ứng dụng giảm xuống đáng kể do giới hạn tốc độ truyền tải của các giao thức và tốc độ mạng. Hơn nữa việc giao tiếp giữa các microservice có thể bị lỗi khi các kết nối bị lỗi.
Đây có phải là sản phẩm phát triển lâu dài hay không ?
Mức độ scale của hệ thống có lớn khơng (scale về lược người dùng và về dữ liệu) ?
Nhóm phát triển có sẵn sang đối đầu với những khó khan khi chuyển sang mơ hình này hay khơng ?
Các vẫn đề phức tạp bao gồm bảo trì, bảo mật, tính khả dụng của hệ thống từ xa, xác thực và ủy quyền truy cập từ xa ?

2.7 Các đặc điểm của mô hình Microservice

Tập hợp một nhóm nhỏ các service: mức độ chi tiết của một service là nhỏ và mỗi service này sẽ chịu một trách nhiệm cụ thể (single responsiblity) và chỉ tập trung

vào nhiệm vụ đó. Ví dụ: storage service sẽ chịu riêng trách nhiệm về lưu trữ

Việc phát triển và mở rộng một service là hoàn toàn độc lập. Điều này mang lại tính linh hoạt cho hệ thống . Q trình deliver feature, release version sẽ dễ dàng và nhanh chóng. Hơn nữa sẽ khơng cịn tình trạng bị block như ở mơ hình monolithic

Giảm tải được các mối quan ngại về công nghệ sử dụng. Chọn một công nghệ phù hợp với vấn đề của doanh nghiệp có thể được giải quyết dễ dàng. Các service giáp tiếp với nhau thông qua API, do vậy mỗi service có thể dùng một ngơn ngữ riêng biệt. Service A dùng Java, Service B dùng Javascript ...

Đối với team, microservice đem lại tính độc lập và tự quản lí cho team. Một team sẽ có trách nhiệm tồn bộ với life-cycle của một hay nhiều service. Họ làm việc trong việc context biệt lập, có thể tự quản lí các quyết định của mình.

2.8 Đặc Trưng của Kiến trúc Microservice

2.8.1. Micro-service.

Đặc trưng này được thể hiện ngay từ tên của kiến trúc. Nó là microservice chứ không phải là miniservice hay nanoservice. Trên thực tế khơng tồn tại mơ hình kiến trúc cho miniservice hay nanoservice. Từ microservice được sử dụng để giúp người thiết kế có cách tiếp cận đúng đắn. Một ứng dụng lớn cần được chia nhỏ ra thành nhiều

thành phần, các thành phần đó cần tách biệt về mặt dữ liệu (database) và phải đủ nhỏ cả về mặt kích cỡ và độ ảnh hưởng của nó trong hệ thống, khi thêm một microservice vào hệ thống cũng nên đảm bảo rằng nó đủ nhỏ để dễ dàng tháo gỡ, xóa bỏ khỏi hệ thống mà không ảnh hưởng nhiều tới các thành phần khác.

2.8.2. Tính độc lập.

Các microservice hoạt động tách biệt nhau trong hệ thống, do vậy việc build một microservice cũng độc lập với việc build các microservice khác. Thông thường, để tiện cho việc phát triển và duy trì các microservice, người phát triển nên viết các built script khác nhau cho mỗi microservice.

Do tính tách biệt này mà mỗi microservice đều dễ dàng thay thế và mở rộng. Hơn thế nữa, nó cịn giúp việc phát triển các microservice linh động hơn, các microservice có thể được phát triển bởi các team khác nhau, dùng các ngôn ngữ khác nhau và tiến độ phát triển dự án cũng nhanh hơn do khơng có sự phụ thuộc giữa các team, mỗi team có thể chủ động quản lý phần việc riêng của mình.

2.8.3. Tính chun biệt.

Mỗi microservice là một dịch vụ chuyên biệt, có thể hoạt động độc lập, thơng thường mỗi microservice đại diện cho một tính năng mà các công ty/ doanh nghiệp muốn cung cấp tới người dùng, do vậy người thiết kế hệ thống microservice cần hiểu rõ về hoạt động kinh doanh của công ty. Các đầu vào đầu ra và chức năng của mỗi microservice cần được định nghĩa rõ ràng.

2.8.4. Phòng chống lỗi.

Kiến trúc microservice sinh ra là để dành cho các hệ thống từ lớn đến vô cùng lớn. Nó áp dụng phương pháp chia để trị, phương pháp này giúp việc áp dụng các công cụ, kỹ thuật cho việc giám sát,phòng chống lỗi phần mềm, lỗi hệ thống hiệu quả.

Khi một thành phần trong hệ thống bị lỗi, nó có thể được thay thế bằng các thành phần dự phịng một cách dễ dàng, trong q trình thay thế thành phần bị lỗi, các thành

phần khác vẫn hoạt động bình thường, do vậy hoạt động của tồn bộ hệ thống sẽ khơng hoặc ít bị gián đoạn.

2.9 Sự khác biệt giữa Microservices và kiến trúc monolithic

Kiến trúc Microservices và kiến trúc Monolithic là hai mô hình khác nhau trong việc xây dựng ứng dụng phần mềm. Dưới đây là sự so sánh giữa hai kiến trúc này dựa trên các điểm khác biệt quan trọng:

Khái niệm:

Microservice: Với kiến trúc Microservices mỗi dịch vụ sẽ được chia nhỏ thành nhiều thành phần khác
nhau, mỗi thành phần sẽ hoạt động độc lập, được phát triển độc lập và chỉ xử lý các nghiệp vụ chức năng của nó. Mỗi thành phần cũng sẽ không lệ thuộc vào công nghệ phát triển với các thành phần khác.

Monolithic: Với cách triển khai truyền thống, sẽ sử dụng kiến trúc monolithic, mỗi dịch vụ sẽ hoạt động độc lập và đầy đủ các chức năng từ Xác thực, Định danh người dùng, Logging các request cho đến xử lý nghiệp vụ của dịch vụ. Mỗi dịch vụ được mở rộng bằng cách tạo thêm một node dịch vụ mới và phân tải request vào các node dịch vụ.

Phạm vi:

Microservice: Ứng dụng có phạm vi lớn và bạn xác định các tính năng sẽ được phát triển rất mạnh theo thời gian. Ví dụ: cửa hàng thương mại điện tử trực tuyến, dịch vụ truyền thông xã hội, dịch vụ truyền phát video với số lượng người dùng lơn, dịch vụ cung cấp API,…

Monolithic: Phạm vi ứng dụng là nhỏ và được xác định rõ. Bạn chắc chắn ứng dụng sẽ khơng phát triển mạnh về các tính năng. Ví dụ: blog, web mua sắm trực tuyến đơn giản, ứng dụng CRUD đơn giản…

Thành viên:

Microservice :Team-size lớn, có đủ thành viên để phát triển các component riêng lẻ một cách hiệu quả.

Monolithic :Team-size nhỏ, thường ít hơn 8 người.

Kĩ năng của thành viên:

Microservice :Mặt bằng kỹ năng của team tốt và các thành viên tự tin về các mẫu thiết kế microservice nâng cao.

Monolithic :Mặt bằng kỹ năng của các thành viên trong team thường không cao

Kiến trúc:

Microservices: Mô hình này chia ứng dụng thành các dịch vụ nhỏ độc lập, mỗi dịch vụ thực hiện một chức năng cụ thể. Các dịch vụ này có thể được triển khai, quản lý và mở rộng độc lập.

Monolithic: Toàn bộ ứng dụng được xây dựng và triển khai như một đơn vị duy nhất, các chức năng và thành phần gắn liền với nhau.

Tính linh hoạt:

Microservices: Có tính linh hoạt cao trong việc phát triển và triển khai. Các dịch vụ có thể được phát triển, thay đổi và triển khai độc lập, không ảnh hưởng đến các phần khác của ứng dụng.

Monolithic: Việc thay đổi một phần của ứng dụng có thể ảnh hưởng đến toàn bộ ứng dụng, cần phải xây dựng và triển khai lại toàn bộ ứng dụng.

Tích hợp và tái sử dụng:

Microservices: Tích hợp giữa các dịch vụ Microservices có thể khá phức tạp, nhưng mỗi dịch vụ có thể được sử dụng lại trong các ứng dụng khác.

Monolithic: Tích hợp tương đối dễ dàng vì tất cả các phần của ứng dụng được viết trong cùng một mã nguồn, nhưng việc tái sử dụng chức năng cụ thể có thể khó khăn hơn.

Quản lý và mở rộng:

Microservices: Quản lý và mở rộng dễ dàng hơn, vì bạn chỉ cần tập trung vào việc mở rộng các dịch vụ cần thiết thay vì cả ứng dụng.

Monolithic: Việc mở rộng có thể phức tạp hơn vì cần tăng cường tài nguyên cho toàn bộ ứng dụng.

Hiệu năng:

Microservices: Có thể tối ưu hiệu năng từng dịch vụ riêng lẻ, nhưng việc gọi qua lại giữa các dịch vụ có thể tạo ra độ trễ.

Monolithic: Hiệu năng được duy trì tốt hơn trong trường hợp ứng dụng không quá lớn và phức tạp.

Phát triển độc lập:

Microservices: Các đội phát triển có thể làm việc độc lập trên các dịch vụ khác nhau, không ảnh hưởng đến nhau.

Monolithic: Cần phải làm việc chung trên cùng một mã nguồn, dễ gây xung đột khi phát triển song song.

Phân chia quyền truy cập dữ liệu:

Microservices: Các dịch vụ có thể có quyền truy cập dữ liệu riêng tư, giảm rủi ro bảo mật.

Monolithic: Tất cả các phần của ứng dụng có thể có quyền truy cập dữ liệu chung.

Phân phối và độ tin cậy:

Microservices: Phân phối dịch vụ giúp ứng dụng có khả năng đảm bảo và dễ dàng khắc phục lỗi trong từng dịch vụ riêng lẻ.

Monolithic: Khó khăn hơn trong việc xử lý lỗi và khắc phục trong toàn bộ ứng dụng.

Tổng cộng, Microservices tập trung vào tích hợp các dịch vụ nhỏ và độc lập, trong khi kiến trúc Monolithic tạo ra một ứng dụng lớn đơn nhất. Sự lựa chọn giữa hai kiến trúc này phụ thuộc vào yêu cầu và tính chất cụ thể của ứng dụng.

2.10 Ưu điểm và nhược điểm của kiến trúc Microservices

2.10.1 Ưu điểm

Thứ nhất, giúp đơn giản hóa hệ thống. Với tổng số chức năng không đổi, kiến trúc Microservices chia nhỏ hệ thống ra làm nhiều dịch vụ nhỏ lẽ dễ dàng quản lý và triển khai từng phần so với kiến trúc nguyên khối. Mỗi dịch vụ thì được định nghĩa để giao tiếp với các dịch vụ khác thông qua RPC (Remote Procedure Call) hay message-driven API. Kiến trúc Microservices thúc đẩy việc phân tách rạch ròi các dịch vụ nhỏ, việc khó có thể làm nếu xây dựng ứng dụng bằng kiến trúc một khối. Trên hết với mỗi dịch vụ nhỏ, chúng ta sẽ có thời gian phát triển nhanh hơn, dễ nắm bắt cũng như bảo trì hơn.

Thứ hai, kiến trúc này cho phép việc mỗi dịch vụ được phát triển độc lập bởi những

team khác nhau. Cũng như cho phép developer có thể tự do chọn lựa technology stack cho mỗi dịch vụ mình phát triển. Dĩ nhiên tự do lựa chọn nhưng không phải là tạo ra một mớ hỗn độn về technology, điều này thì khơng có một dự án hay sản phẩm nào mong muốn cả. Tuy nhiên, sự tự do này có nghĩa là các developer khơng cịn phải bắt buộc phải sử dụng các cơng nghệ lỗi thời có thể đã tồn tại vào lúc bắt đầu dự án. Khi viết một dịch vụ mới, họ có tùy chọn của việc sử dụng công nghệ bắt kịp với xu thế. Hơn nữa, vì dịch vụ là tương đối nhỏ, việc viết lại một dịch vụ cũ dựa trên nền tảng cơng nghệ mới hơn là hồn tồn khả thi.

Thứ ba, kiến trúc Microservices cho phép mỗi dịch vụ có thể được triển khai một cách độc lập. Cùng với đó thì việc triển khai hệ thống theo kiểu continuous deployment là hồn tồn có thể.
Cuối cùng, kiến trúc Microservices cho phép mỗi dịch vụ có thể thực hiện việc scale một cách độc lập. Bạn có thể scale dễ dàng bằng cách tăng số instance phục vụ

cho mỗi dịch vụ lên và phân tải bằng load balancer. Ngoài ra bạn cũng có thể tối ưu chi phí vận hành dịch vụ bằng cách triển khai mỗi dịch vụ lên server có resource thích hợp.

2.10.2. Nhược điểm
Microservice nhấn mạnh kích thước nhỏ gọn của dịch vụ. Một số lập trình đề xuất dịch vụ siêu nhỏ cỡ dưới 100 dòng code. Nhưng làm sao để chia nhỏ, và chia làm sao để khi chia quá nhiều sẽ dẫn đến manh mún, vụn vặt, khó kiểm sốt.
Nhược điểm thứ hai của Microservices đến từ đặc điểm hệ thống phân tán (distributed system). Lập trình viên cần phải lựa chọn phát triển mỗi dịch vụ nhỏ giao tiếp với các dịch vụ khác bằng cách nào messaging hay là RPC. Hơn thế nữa, họ phải

xử lý sự cố khi kết nối chậm, lỗi khi thông điệp không gửi được hoặc thong điệp gửi đến nhiều đích đến vào các thời điểm khác nhau.
Thứ ba, phải đảm bảo giao dịch phân tán (distributed transaction) cập nhật dữ liệu đúng đắn (all or none) vào nhiều dịch vụ nhỏ khác nhau khó hơn rất nhiều, đôi khi là không thể so với đảm bảo giao dịch cập nhật vào nhiều bảng trong một cơ sở dữ liệu trung tâm. Theo nguyên tắc CAP (CAP theorem) thì giao dịch phân tán sẽ không thể thỏa mãn cả 3 điều kiện: consistency (dữ liệu ở điểm khác nhau trong mạng phải giống nhau), availablity (yêu cầu gửi đi phải có phúc đáp), partition tolerance (hệ thống vẫn hoạt động được ngay cả khi mạng bị lỗi). Những công nghệ cơ sở dữ liệu phi quan hệ (NoSQL) hay môi giới thông điệp (message broker) tốt nhất hiện nay cũng chưa vượt qua nguyên tắc CAP.
Thứ tư, testing một dịch vụ trong kiến trúc microservices đôi khi yêu cầu phải chạy cả các dịch vụ nhỏ khác mà nó phụ thuộc. Do đó khi phân rã ứng dụng một khối thành

microservices cần luôn kiểm tra mức độ ràng buộc giữa các dịch vụ. Nếu các dịch vụ nhỏ thiết kế phục thuộc vào nhau theo chuỗi. A gọi B, B gọi C, C gọi D. Nếu một mắt xích có giao tiếp API thay đổi, liệu các mắt xích khác có phải thay đổi theo khơng? Nếu có thì việc bảo trì, kiểm thử sẽ phức tạp tương tự ứng dụng một khối. Thiết kế

dịch vụ tốt sẽ giảm tối đa ảnh hưởng lan truyền đến các dịch vụ khác.
Cuối cùng, triển khai dịch vụ microservices nếu làm thủ công theo cách đã làm với ứng dụng một khối phức tạp hơn rất nhiều. Ứng dụng một khối bổ sung các server mới giống hệt nhau đằng sau load balancer. Trong khi ở kiến trúc microservices, các dịch vụ
nhỏ nằm trên nhiều máy ảo hay Docker container khác nhau, hoặc một dịch vụ có nhiều thực thể phân tán ra nhiều vị trí.

2.11 thiết kế phần mềm theo kiến trúc Microservice

2.11.1. Mỗi microservice nên có một database riêng biệt

Việc này đảm bảo cho microservice có tính đóng gói cao. Tuy vậy việc phân tách nơi chứa dữ liệu cho mỗi microservice không hề đơn giản, nó là phần khó nhất trong việc thiết kế ứng dụng phần mềm theo kiến trúc microservice . Do đó, rất nhiều người chọn lựa thiết kế sử dụng chung database cho nhiều microservice.

Cách này tồn tại một hạn chế là khi database schema cần thay đổi cho microservice1 thì nó sẽ làm ảnh hưởng và gây lỗi cho các microservice khác sử dụng chung database này. Để sửa lỗi, ta cần sửa đổi, cập nhật tồn bộ microservice cịn lại để chúng có thể hoạt động với schema mới.

Sơ đồ trên, microservice2 không truy cập trực tiếp vào database, thay vào đó nó truy cập database thơng qua microservice1. Do đó việc thay đổi schema của database sẽ không ảnh hưởng tới microservice2. Tuy nhiên với cách tiếp cận này thì

microservice2 phụ thuộc vào microservice1, khi microservice1 có lỗi thì microservice2 cũng bị lỗi theo.

Việc chọn cách tiếp cận nào phụ thuộc rất nhiều vào tình hình thực tế của dự án. Cần cân nhắc thiệt hơn của mỗi phương án để đưa ra lựa chọn cuối cùng.

2.11.2. Giữ source code của microservice ở mức hợp lý

Như đã đề cập ở phần ưu và nhược điểm của microservice. Kích thước source code của một microservice không nên quá nhỏ hoặc quá lớn. Tuy nhiên cái khó ở đây là

khơng có một con số định lượng cho kích thước của một microservice, nên thơng thường việc quyết định kích thước của một microservice là do kinh nghiệm, cảm tính.

2.11.3. Tạo build script cho mỗi microservice

Build script có thể là một file bash hoặc Dockerfile để đóng gói microservice vào bên trong một docker image.

2.11.4. Triển khai mỗi microservice bên trong một app (docker container)

Việc triển khai mỗi microservice trong một docker container đem lại rất nhiều lợi ích cho việc triển khai và mở rộng ứng dụng cũng như việc phân chia tài nguyên phần cứng cho mỗi microservice. Hiện nay có rất nhiều cơng cụ hỗ trợ cho việc liên tục tích hợp, liên tục triển khai hệ thống microservice. Các công cụ này giúp tăng hiệu quả làm việc cho các lập trình viên, giảm thời gian phơi phối sản phẩm phần mềm, và các cơng cụ này địi hỏi mỗi microservice được đóng gói trong một docker image và triển khai trên app.

2.11.5. Stateless server

Khi một yêu cầu được gửi đến server thì một phiên làm việc (session) được mở ra, kèm theo đó là các thơng tin của phiên. Stateless server là server khơng lưu thông tin của phiên. Mà thông tin về phiên được lưu ở một nơi khác, như caching server chẳng hạn.

Việc này rất quan trọng, bởi vì mỗi microservice thường được đóng gói thành một docker image. Khi muốn cập nhật một microservice, ta cập nhật docker image của nó, và khi chạy docker image mới (xóa bỏ container cũ và tạo container mới dựa trên image mới) thì tồn bộ thơng tin của các phiên hoạt động trên container cũ sẽ bị mất, thông tin phiên thường bao gồm thông tin mà client gửi tới server, mất thông tin này là một lỗi vô cùng nghiêm trọng. Nếu container là stateless thì nó khơng lưu thơng tin của các phiên nên khơng có gì để mất cả.

2.12. Ứng dụng của kiến trúc microservice

Hầu hết các công ty xây dựng hệ thống của họ bằng cách sử dụng kiến trúc nguyên khối. Họ bắt đầu theo cách này trong giai đoạn đầu của dự án, vì việc thiết lập một kiến trúc nguyên khối và đưa doanh nghiệp đi vào hoạt động sẽ nhanh hơn nhiều. Tuy

nhiên, dự đoán sự tăng trưởng trong tương lai, một số thương hiệu đã đưa ra các phương pháp mới để xử lý sự phức tạp.

1 số công ty sử dụng kiến trúc Microservice:

ϖ Amazon

Amazon là một trong những công ty đầu tiên mà microservices đóng vai trị quan trọng trong việc chuyển đổi toàn bộ hoạt động kinh doanh. Tất cả các dịch vụ và thành phần của nó được kết hợp chặt chẽ với nhau. Do đó, tất cả các thay đổi mã lớn đã bị

mắc kẹt trong quá trình triển khai trong nhiều tuần trước khi khách hàng có thể sử dụng chúng.

Amazon đã sử dụng microservices để chia nhỏ các cấu trúc thành một ứng dụng duy nhất. Điều này đã đơn giản hóa và rút ngắn quy trình cho phép các nhà phát triển nhận ra đâu là điểm nghẽn. Nó đã giúp họ xây dựng lại các ứng dụng dưới dạng kiến trúc hướng dịch vụ, mỗi ứng dụng có một nhóm nhỏ dành riêng cho một dịch vụ.

ϖ Coca Cola

Tập đoàn CNTT Toàn cầu của Coca Cola đã phải đối mặt với một thách thức lớn trong việc kết nối tất cả các thực thể trên các lục địa khác nhau và hỗ trợ sự phát triển

của họ. Những thay đổi nhanh chóng dường như là khơng thể do có nhiều giải pháp được triển khai trên toàn cầu. Họ quyết định tận dụng microservices bằng cách sử dụng mơ hình Dev-Ops và GIT . Coca Cola cho biết họ có thể giảm 50% lưu lượng dữ liệu đến mạng của mình và thời gian cần thiết để mở rộng quy mô hỗ trợ giảm từ vài tuần xuống còn vài phút.

ϖ Netflix

Netflix cũng là một trong những nhà cung cấp dịch vụ vi mô sớm nhất đã thiết lập kiến trúc của họ trên AWS .Quá trình chuyển đổi của họ diễn ra theo từng bước - đầu tiên, họ di chuyển mã hóa phim và các ứng dụng khơng phải của khách hàng. Sau đó, họ tách các yếu tố hướng tới khách hàng, như đăng ký tài khoản, chọn phim, chọn thiết bị và cấu hình. Ngày nay, ứng dụng Netflix tận dụng hơn 500 dịch vụ vi mô và Cổng API xử lý hơn 2 tỷ yêu cầu cạnh API mỗi ngày.

ϖ Spotify

Ngay khi Spotify gia nhập thị trường, họ đã phát triển khá nhanh chóng. Nó sớm bắt đầu tìm kiếm một giải pháp có thể hỗ trợ quy mơ cho hàng triệu người dùng, nhiều nền tảng và xử lý các quy tắc kinh doanh phức tạp. Nhóm cơng nghệ của họ đã tìm ra cách để đáp ứng những yêu cầu này bằng cách khởi chạy Microservices.

Hiện tại, Spotify đang chạy hơn 810 dịch vụ, được xây dựng cấu trúc linh hoạt có thể dễ dàng mở rộng và giải quyết các nút thắt trong thế giới thực trong thời gian ngắn.

ϖ 5Walmart

Walmart đã không thể xử lý 6 triệu lượt xem trang mỗi phút vào năm 2005 và khơng thể có được bất kỳ loại trải nghiệm người dùng tích cực nào.

Walmart đã cải tiến lại kiến trúc microservices vào năm 2012 và chuyển đổi đã tăng 20% chỉ sau một đêm, đơn đặt hàng trên thiết bị di động tăng 98% ngay lập tức và tiết kiệm 40% sức mạnh tính tốn và tiết kiệm 20–50% chi phí nói chung.

2.13 Sơ lược về tổ chức của kiến trúc phần mềm Microservices

Tổ chức của kiến trúc Microservices tập trung vào việc chia ứng dụng thành các dịch vụ nhỏ, độc lập với khả năng thực hiện các chức năng cụ thể. Dưới đây là một sơ lược về cách tổ chức kiến trúc Microservices:

Dịch vụ Độc Lập: Kiến trúc Microservices tập trung vào việc phân chia ứng dụng thành các dịch vụ riêng biệt, mỗi dịch vụ đảm nhiệm một chức năng hoặc tính năng cụ thể của ứng dụng. Mỗi dịch vụ hoạt động độc lập và có thể được phát triển, triển khai và quản lý riêng.

Tương Tác Qua Giao Thức và API: Các dịch vụ Microservices tương tác thông qua giao thức và API chuẩn, cho phép chúng giao tiếp và trao đổi dữ liệu một cách linh hoạt. Giao thức này có thể là HTTP, gRPC, MQTT hoặc các giao thức khác tùy thuộc vào yêu cầu của ứng dụng.

Phân Chia Chức Năng: Các dịch vụ được tổ chức dựa trên chức năng hoặc khả năng cụ thể. Ví dụ, một dịch vụ có thể quản lý thông tin người dùng, một dịch vụ khác có thể xử lý thanh toán, và còn các dịch vụ khác cho các chức năng khác.

Độc Lập Phát Triển và Triển Khai: Các đội phát triển có thể làm việc độc lập trên từng dịch vụ. Điều này cho phép phát triển song song và nhanh chóng hơn, mà không ảnh hưởng đến các dịch vụ khác.

Quản Lý Dịch Vụ: Mỗi dịch vụ được quản lý độc lập, bao gồm quản lý tài nguyên, mở rộng, theo dõi hiệu năng và xử lý lỗi. Điều này giúp đảm bảo tính ổn định và khả năng mở rộng của từng dịch vụ.

Tổng Hợp Dữ Liệu: Trong một ứng dụng Microservices, dữ liệu thường phải được tổng hợp từ nhiều dịch vụ khác nhau. Điều này đòi hỏi cơ chế để tổng hợp dữ liệu từ các dịch vụ và hiển thị cho người dùng cuối.

Bảo Mật và Quản Lý Định Danh: Vì các dịch vụ hoạt động độc lập, việc quản lý xác thực và phân quyền đòi hỏi sự quan tâm đặc biệt. Hệ thống cần có cơ chế bảo mật và quản lý định danh để đảm bảo tính an toàn.

Tích Hợp Tương Thích: Các dịch vụ có thể được xây dựng bằng ngôn ngữ và công nghệ khác nhau, miễn là chúng tuân theo các chuẩn giao tiếp được đặt ra. Điều này cho phép sự đa dạng về công nghệ và ngôn ngữ trong toàn bộ hệ thống.

III. PHÂN TÍCH YÊU CẦU

3.1 Xác định yêu cầu và chức năng của hệ thống

Phân tích các yêu cầu cơ bản và chức năng mà hệ thống cần đáp ứng.

3.2 Phân tích yêu cầu kỹ thuật và chức năng cho từng dịch vụ

Xác định yêu cầu chi tiết cho từng dịch vụ cụ thể, bao gồm cả giao diện API và tính năng.

3.3 Thiết kế mô hình dữ liệu cho các dịch vụ

Xây dựng mô hình dữ liệu cho từng dịch vụ, đảm bảo tích hợp hợp lý trong toàn bộ hệ thống.

IV. NHIỆM VỤ CỦA CÁC DỊCH VỤ MICROSERVICES

4.1 Xác định và mô tả các dịch vụ cụ thể

Mô tả từng dịch vụ, bao gồm tên gọi, mục tiêu và phạm vi của chúng.

4.2 Mô tả cách các dịch vụ hoạt động độc lập và tương tác với nhau

Mô tả cách các dịch vụ giao tiếp và tương tác trong toàn bộ hệ thống.

4.3 Phân tích cách mỗi dịch vụ đóng góp vào toàn bộ hệ thống

Xác định cách mỗi dịch vụ ảnh hưởng đến toàn bộ kiến trúc và quản lý hệ thống.

Mô tả cách mỗi dịch vụ ảnh hưởng đến toàn bộ hệ thống:

4.3.1 Dịch vụ Quản lý Người dùng

Dịch vụ này chịu trách nhiệm quản lý thông tin người dùng và xác thực. Nó cung cấp API để đăng nhập, đăng ký, và quản lý thông tin cá nhân. Dịch vụ này ảnh hưởng đến tính bảo mật và xác thực trong toàn bộ hệ thống.

4.3.2 Dịch vụ Quản lý Sản phẩm

Dịch vụ này quản lý thông tin về sản phẩm và dịch vụ mà hệ thống cung cấp. Nó cung cấp API để thêm, sửa đổi, và truy vấn thông tin sản phẩm. Dịch vụ này ảnh hưởng đến quản lý dữ liệu sản phẩm và tính năng liên quan đến gian hàng điện tử.

4.3.3 Dịch vụ Xử lý Đơn hàng

Dịch vụ này xử lý việc tạo, cập nhật và xác nhận đơn hàng từ khách hàng. Nó kết nối với dịch vụ Quản lý Sản phẩm để kiểm tra tính khả dụng của sản phẩm. Dịch vụ này ảnh hưởng đến quá trình mua hàng và giao hàng.

V. THIẾT KẾ KIẾN TRÚC MICROSERVICES

5.1 Thiết kế kiến trúc chung của hệ thống Microservices

Mô tả kiến trúc tổng quan của hệ thống, bao gồm sự tương tác giữa các dịch vụ và cách chúng giao tiếp.

5.2 Sử dụng các công nghệ và khung công việc phù hợp

Tìm hiểu các công nghệ như Docker để đóng gói dịch vụ, Kubernetes để quản lý và triển khai dịch vụ, và các khung công việc như Spring Boot để xây dựng dịch vụ.

5.3 Tích hợp các dịch vụ thông qua giao thức chuẩn

Mô tả cách các dịch vụ tương tác với nhau thông qua giao thức chuẩn như REST API hoặc giao thức message queue.

VI. TRIỂN KHAI VÀ QUẢN LÝ

6.1 Cách triển khai và quản lý từng dịch vụ

Mô tả quy trình triển khai dịch vụ, từ việc đóng gói đến việc triển khai trên môi trường sản xuất.

6.2 Mô hình triển khai tự động và liên tục (CI/CD)

Mô tả cách triển khai liên tục giúp tối ưu hóa quá trình phát triển và triển khai.

6.3 Giải pháp để theo dõi và khắc phục sự cố trong kiến trúc Microservices

Tìm hiểu cách giám sát và quản lý sự cố trong môi trường Microservices.

6.3.1 Giám sát hiệu suất

Sử dụng các công cụ giám sát hiệu suất như Prometheus để theo dõi hiệu suất của từng dịch vụ. Theo dõi các chỉ số như tài nguyên máy tính, thời gian phản hồi, và tải trọng để phát hiện vấn đề kịp thời.

6.3.2 Quản lý logs và ghi chú

Sử dụng các hệ thống ghi log như ELK Stack để thu thập và phân tích logs từ các dịch vụ. Điều này giúp tìm ra nguyên nhân của các sự cố và xác định hành động cần thực hiện.

6.3.3 Xử lý sự cố tự động

Triển khai các hệ thống tự động xử lý sự cố như Hystrix để xác định và tự động phản ứng với các sự cố trong dịch vụ. Các biện pháp như tránh gửi lời gọi nếu dịch vụ không khả dụng hoặc chuyển hướng yêu cầu sang dịch vụ khác.

VII. KẾT QUẢ VÀ NHẬN XÉT

7.1 Hiểu biết về Microservices sau quá trình nghiên cứu

Tổng hợp những hiểu biết và kiến thức về Microservices bạn đã thu thập trong quá trình nghiên cứu.

7.2 Đánh giá hiệu suất và khả năng mở rộng của hệ thống

Xác định hiệu suất của hệ thống sau khi triển khai kiến trúc Microservices và so sánh với kiến trúc trước đó. Đánh giá khả năng mở rộng của hệ thống trong tình huống tải cao.

7.3 Nhận xét về việc áp dụng kiến thức Microservices trong thực tế

Nhận xét về các thách thức, lợi ích và kinh nghiệm áp dụng kiến thức về Microservices vào việc xây dựng hệ thống thực tế.

VIII. HƯỚNG PHÁT TRIỂN TƯƠNG LAI

8.1 Đề xuất cách tối ưu hóa hệ thống Microservices

Đề xuất các biện pháp để tối ưu hóa hiệu suất và khả năng mở rộng của hệ thống Microservices.

8.2 Gợi ý các nâng cấp và mở rộng trong tương lai

Đề xuất các phát triển tiềm năng trong tương lai, bao gồm việc thêm tính năng mới và mở rộng hệ thống.

IX. TÀI LIỆU THAM KHẢO

Danh sách các tài liệu, sách, bài báo và nguồn tham khảo mà bạn đã sử dụng trong quá trình nghiên cứu và viết bài báo cáo.