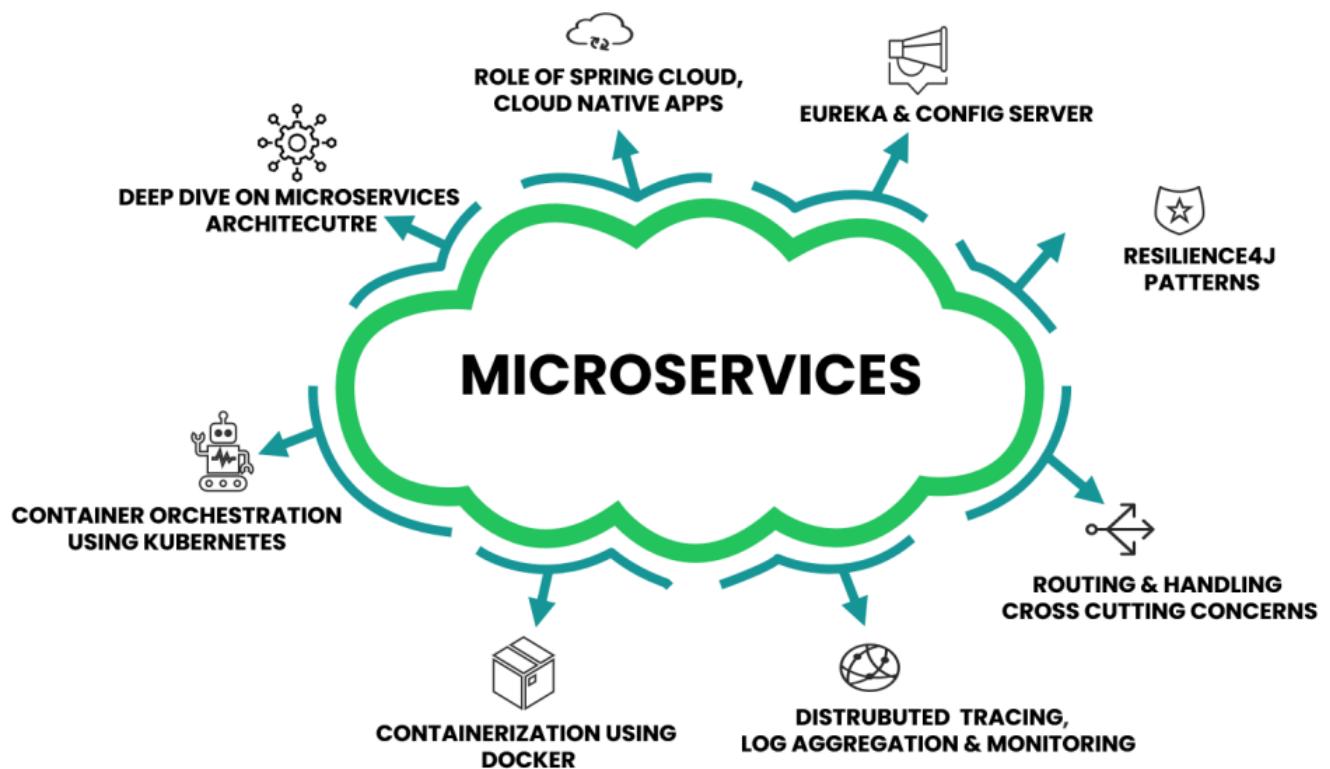




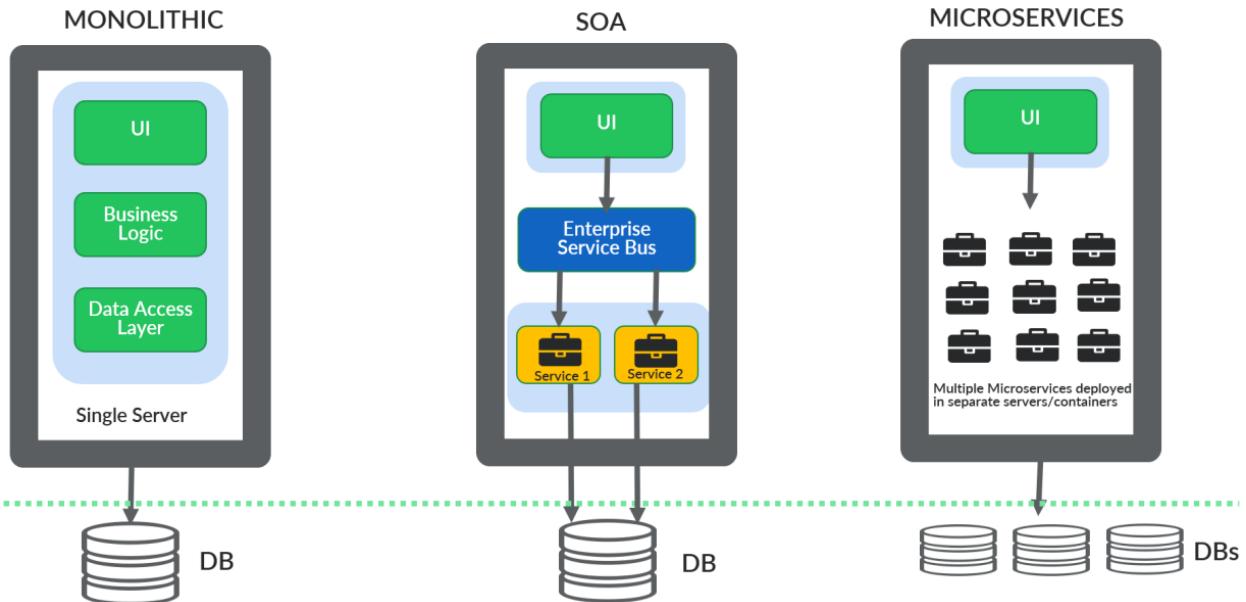
MICROSERVICES

USING SPRING, DOCKER, KUBERNETES



Section 1: Introduction to Microservices Architecture

1. Evolution of Microservices architecture



a) Kiến trúc nguyên khối (Monolithic Architecture)

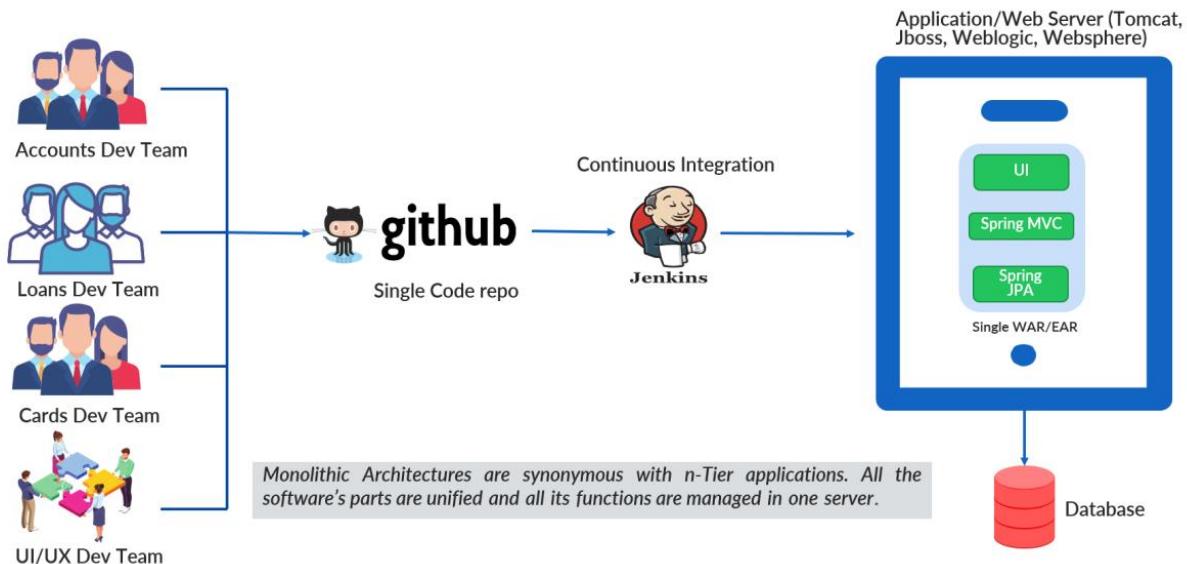
Monolith là một từ cổ để chỉ một khối đá khổng lồ. Trong công nghệ phần mềm, một mẫu nguyên khối đề cập đến một đơn vị phần mềm không thể chia tách. Khái niệm phần mềm nguyên khối nằm trong các thành phần khác nhau của ứng dụng được kết hợp thành một chương trình duy nhất trên một nền tảng duy nhất. Thông thường, ứng dụng nguyên khối bao gồm cơ sở dữ liệu, giao diện người dùng phía client và ứng dụng phía máy chủ. Tất cả các bộ phận của phần mềm được hợp nhất và tất cả các chức năng của phần mềm được quản lý ở một nơi. Chúng ta hãy xem xét cấu trúc của phần mềm nguyên khối một cách chi tiết.

Các thành phần của phần mềm nguyên khối được kết nối với nhau và phụ thuộc lẫn nhau, giúp phần mềm được khép kín. Kiến trúc này là một giải pháp truyền thống để xây dựng các ứng dụng, nhưng một số lập trình viên thấy nó đã lỗi thời. Tuy nhiên, trong nhiều trường hợp, một kiến trúc nguyên khối là một giải pháp hoàn hảo cho một số loại phần mềm.

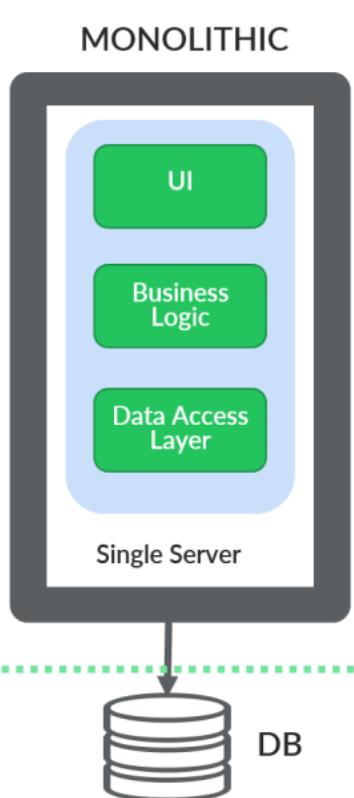
MONOLITHIC ARCHITECTURE

SAMPLE BANK APPLICATION

eazy
bytes



Monolithic Architectures đồng nghĩa với các ứng dụng n-Tier. Tất cả các phần của phần mềm được thống nhất và tất cả các chức năng của nó được quản lý trong một máy chủ..



Ưu điểm

- Phát triển và triển khai đơn giản
- Hiệu suất tốt hơn

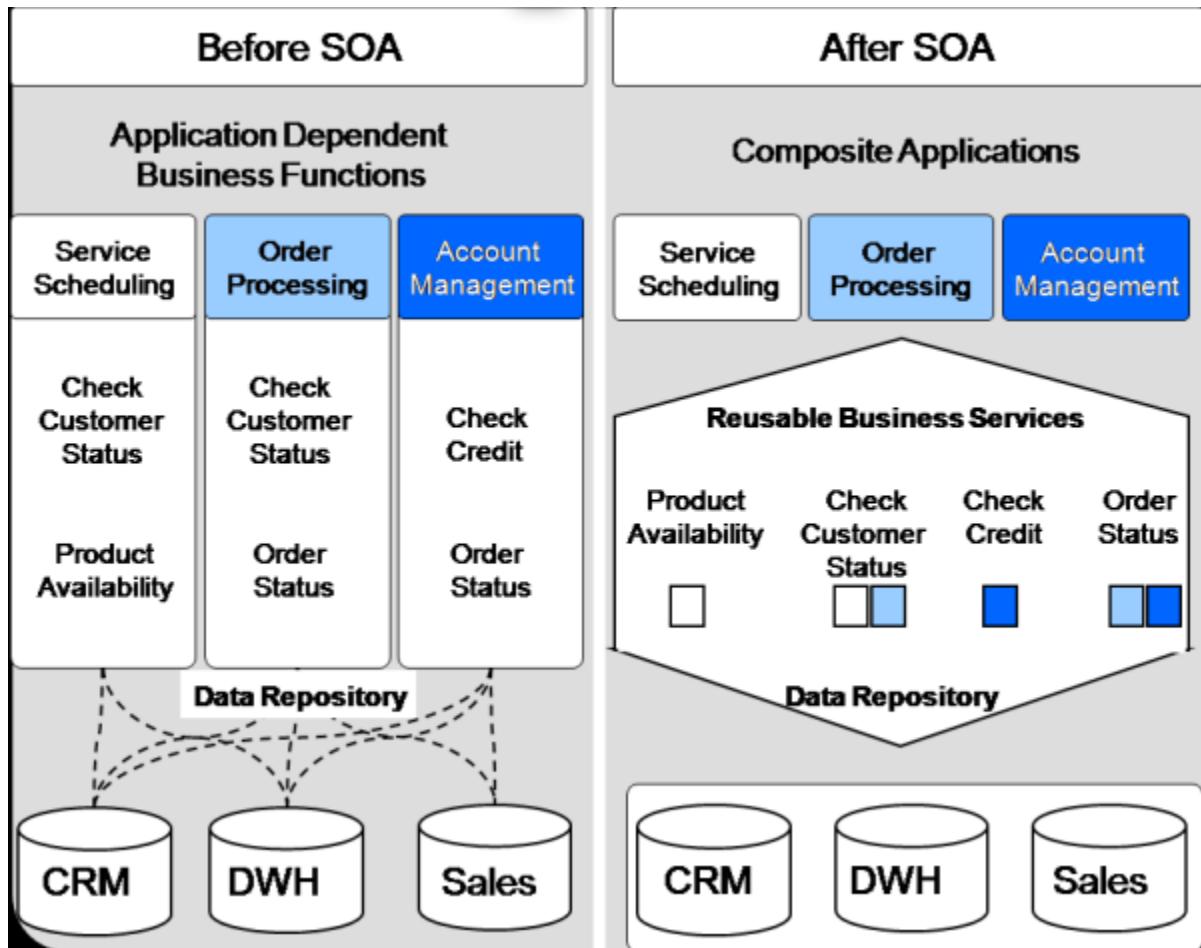
Nếu được xây dựng đúng cách, các ứng dụng nguyên khối thường có hiệu suất cao hơn các ứng dụng dựa trên microservice. Ứng dụng có kiến trúc microservice có thể cần thực hiện 40 lệnh gọi API đến 40 dịch vụ microservice khác nhau để load từng màn hình, điều này rõ ràng dẫn đến hiệu suất chậm hơn.

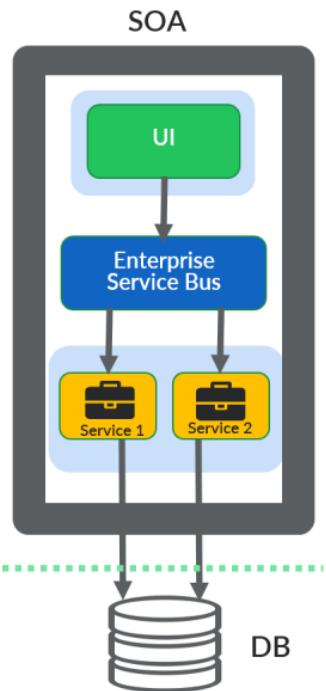
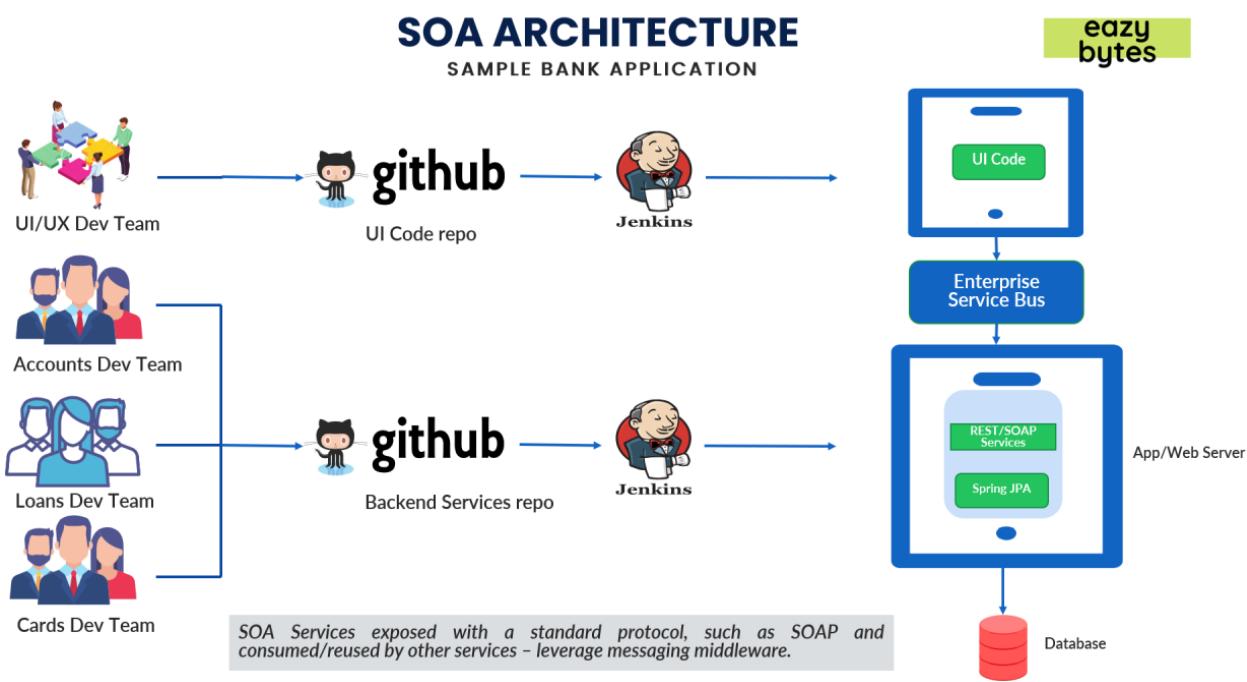
Nhược điểm

- Khó áp dụng công nghệ mới
- Hạn chế khi thay đổi
- Cơ sở mã đơn lẻ và khó bảo trì
- Cập nhật nhỏ và phát triển tính năng luôn cần triển khai đầy đủ

b) Kiến trúc hướng dịch vụ- SOA (Service-Oriented Architecture)

SOA (Service-Oriented Architecture) hay thường được hiểu chính là kiến trúc hướng dịch vụ và là một mẫu kiến trúc sắp xếp phân tách ứng dụng lớn thành các dịch vụ (service). SOA là cấp độ cao hơn của phát triển ứng dụng thường dùng trong quy trình nghiệp vụ và giao tiếp chuẩn để giúp che đi sự phức tạp kỹ thuật. Cùng so sánh việc trước và sau khi sử dụng SOA sẽ như thế nào bằng hình minh họa bên dưới:





Ưu điểm của SOA

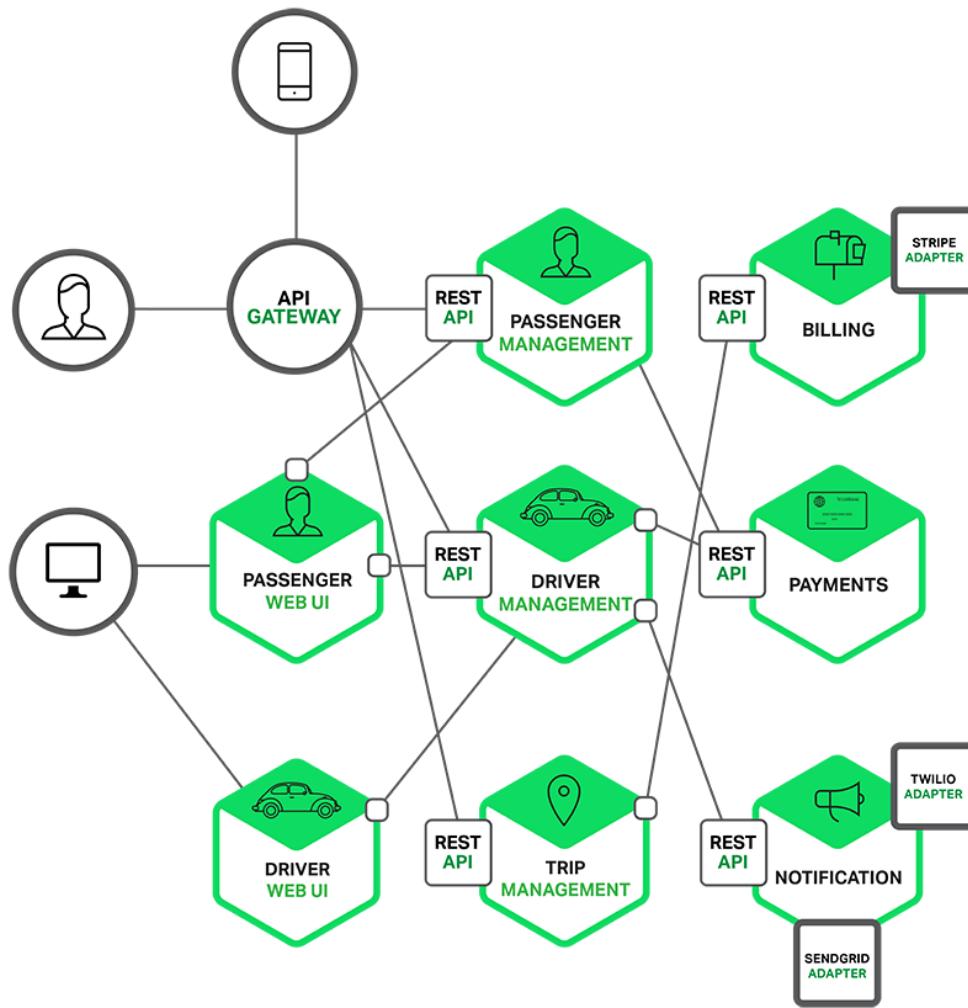
- Khả năng sử dụng lại dịch vụ
- Khả năng bảo trì tốt hơn
- Độ tin cậy cao hơn
- Phát triển song song

Nhược điểm của SOA

- Quản lý phức tạp
- Chi phí đầu tư cao
- Quá tải. Trong SOA, tất cả các đầu vào được xác nhận trước khi một dịch vụ tương tác với một dịch vụ khác. Khi sử dụng nhiều dịch vụ, điều này làm tăng thời gian phản hồi và giảm hiệu suất tổng thể.

c) Kiến trúc Microservice (Microservices Architecture)

Microservice là một loại kiến trúc phần mềm hướng dịch vụ, tập trung vào việc xây dựng một loạt các thành phần tự quản lý tạo nên ứng dụng. Không giống như các ứng dụng nguyên khối được xây dựng dưới dạng một đơn vị không thể chia tách, các ứng dụng microservice bao gồm nhiều thành phần độc lập output ra các API.

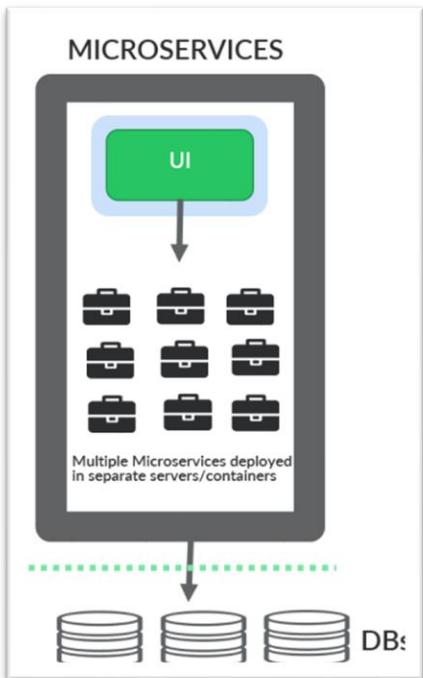
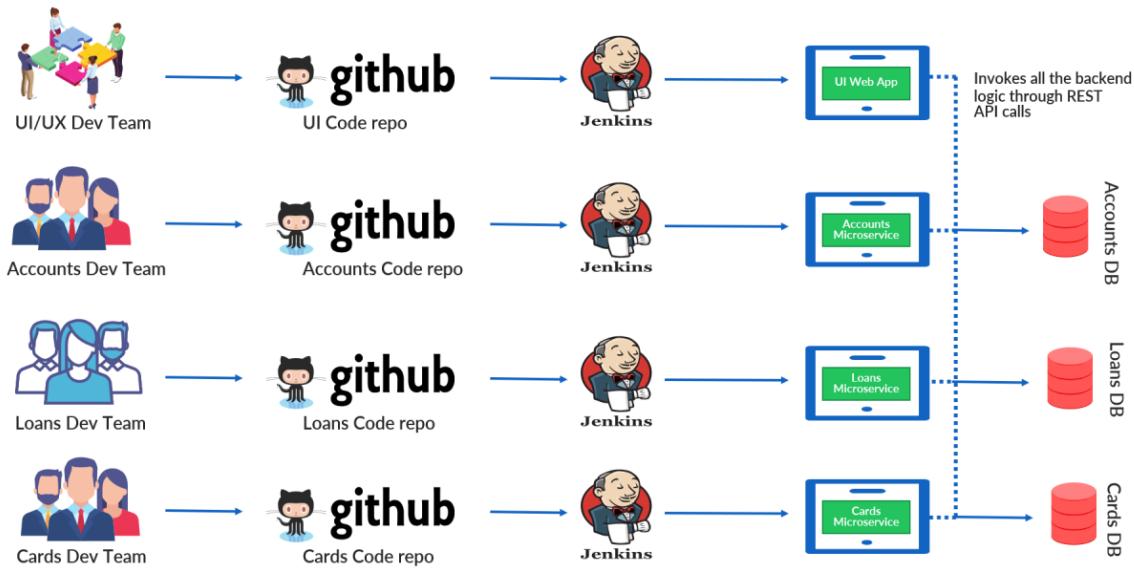


Kiến trúc Microservices đã trở thành một xu hướng quan trọng trong lĩnh vực phát triển phần mềm trong vài năm qua và vẫn tiếp tục phát triển mạnh mẽ. Đây là một kiến trúc phần mềm phân tán, trong đó ứng dụng được chia thành các thành phần nhỏ độc lập gọi là microservices. Mỗi microservice thực hiện một chức năng cụ thể và có thể được triển khai, quản lý và mở rộng độc lập.

MICROSERVICES ARCHITECTURE

SAMPLE BANK APPLICATION

eazy bytes



Ưu điểm

- Dễ phát triển, thử nghiệm và triển khai
- Tăng sự nhanh nhẹn
- Khả năng mở rộng theo chiều ngang
- Phát triển song song

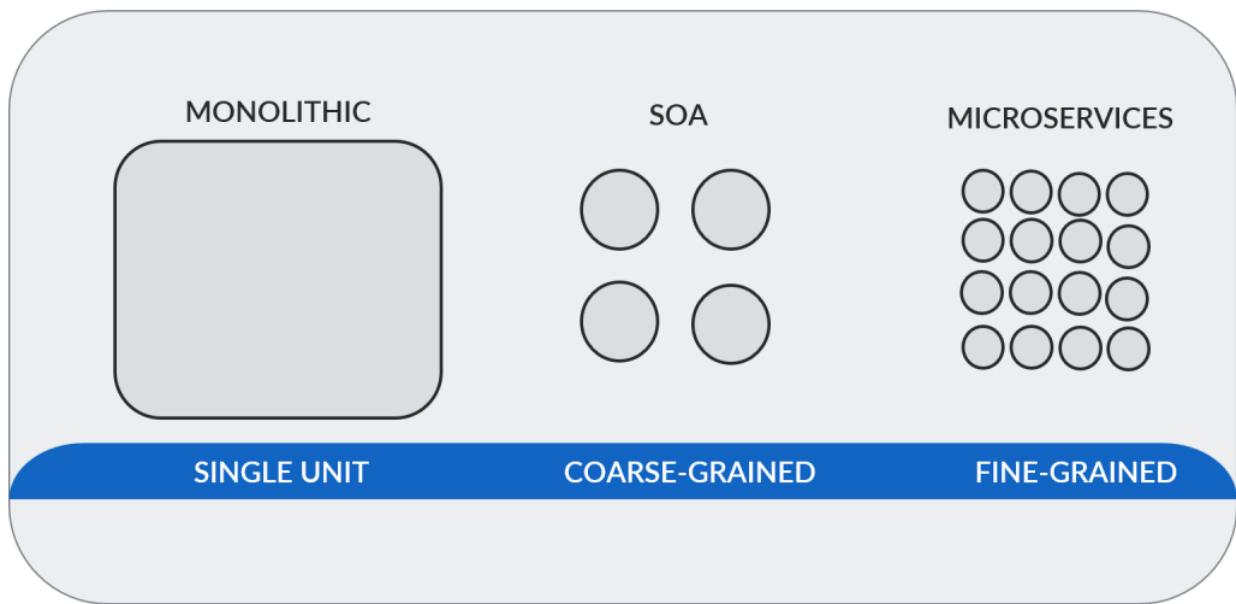
Nhược điểm

- Độ phức tạp
- Chi phí cơ sở hạ tầng
- Lo ngại về an ninh

d) Kết luận cuối cùng

- **Kiến trúc nguyên khối** (Monolithic Architecture) : Dự án Start-up nhỏ, Nguồn lực ít
- **Kiến trúc hướng dịch vụ** (SOA - Service Oriented Architecture) : Ứng dụng doanh nghiệp với nghiệp vụ phức tạp (Ngân hàng ...)
- **Kiến trúc Microservices** (Microservices Architecture) : Hệ thống cực lớn quy mô phức tạp, Đội ngũ phát triển đa năng

2. Monolithic Vs Soa Vs Microservices Comparision



Single Unit, Coarse-Grained và Fine-Grained đề cập đến cách chia nhỏ và tổ chức các thành phần trong kiến trúc phần mềm. Single Unit là sự tổ chức ứng dụng thành một đơn vị duy nhất, trong khi Coarse-Grained và Fine-Grained tập trung vào mức độ chi tiết của việc chia nhỏ thành phần. Coarse-Grained chia nhỏ thành phần thành các đơn vị lớn, trong khi Fine-Grained chia nhỏ thành phần thành các đơn vị nhỏ độc lập.

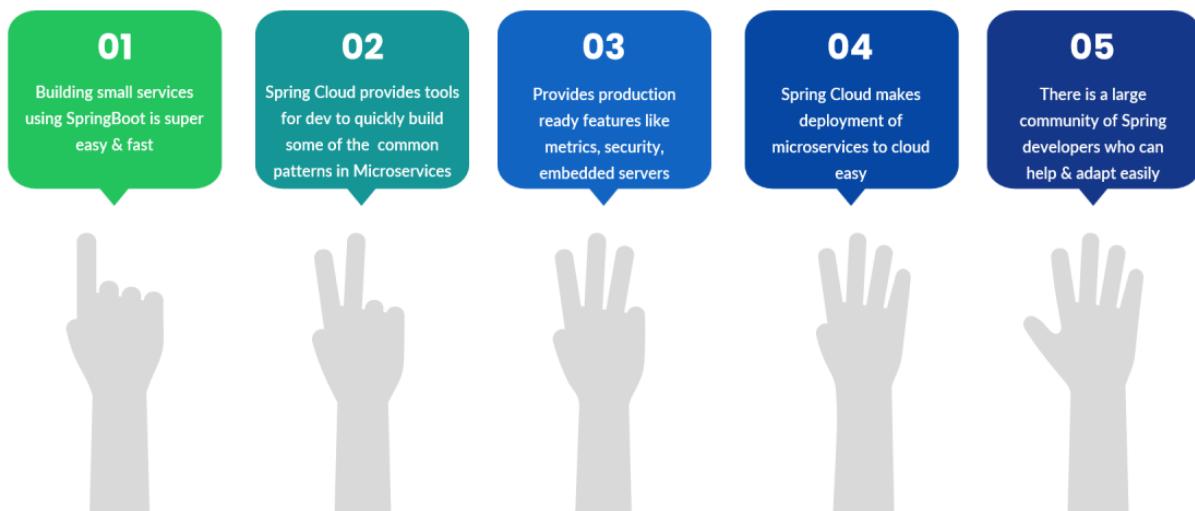
FEATURES	MONOLITHIC	SOA	MICROSERVICES
Parallel Development			
Agility			
Scalability			
Usability			
Complexity & Operational overhead			
Security Concerns & Performance			

Phát triển song song, Linh hoạt, Khả năng mở rộng, Khả năng sử dụng, Độ phức tạp và chi phí hoạt động, Mối quan tâm về bảo mật & Hiệu suất

Phần 2: [Optional] Microservices & Spring (Match Made in Heaven)

1. Why Spring is the best framework for building microservices

Spring là framework phát triển phổ biến nhất để xây dựng các ứng dụng và dịch vụ web dựa trên java. Từ Ngày đầu tiên, Spring đang làm việc để xây dựng mã của chúng tôi dựa trên các nguyên tắc như khớp nối lồng lèo bằng cách sử dụng phép nội xạ phụ thuộc. Trong những năm qua, Spring framework đang phát triển bằng cách duy trì sự phù hợp trên thị trường.



- **01:** Xây dựng các small service bằng SpringBoot cực kỳ dễ dàng & nhanh chóng
- **02:** Spring Cloud cung cấp công cụ cho dev xây dựng nhanh một số pattern phổ biến trong Microservices
- **03:** Cung cấp các tính năng sẵn sàng sản xuất như metrics, security, embedded servers
- **04:** Spring Cloud giúp triển khai microservice lên cloud dễ dàng
- **05:** Có một cộng đồng lớn các nhà phát triển Spring có thể trợ giúp và thích nghi dễ dàng

2. What is spring boot?

Spring Boot giúp dễ dàng tạo các ứng dụng dựa trên Spring cấp độ sản xuất(production-grade), độc lập mà bạn có thể "chỉ cần chạy".

- **STAND ALONE SPRING APPS:** Tạo các ứng dụng Spring độc lập/dịch vụ REST cực kỳ nhanh chóng và dễ dàng
- **NO NEED DEPLOY INTO A SERVER:** Máy chủ **Embed Tomcat, Jetty or Undertow** có sẵn và việc triển khai diễn ra trực tiếp
- **STARTER PROJECTS:** Starters projects là một tập hợp các bộ mô tả dependency thuận tiện mà bạn có thể sử dụng để xây dựng các ứng dụng Spring của mình.
- **AUTO CONFIGURATIONS:** Tự động cấu hình các thư viện/bean của Spring và bên thứ 3 bất cứ khi nào có thể
- **PROD READY FEATURES:** Hỗ trợ sẵn có các tính năng product như số liệu, kiểm tra tình trạng và cấu hình bên ngoài
- **SIMPLE CONFIGURATION:** Cung cấp nhiều annotation để thực hiện các cấu hình đơn giản và không yêu cầu cấu hình XML

3. What is spring cloud?

- Spring Cloud cung cấp các công cụ để các nhà phát triển nhanh chóng xây dựng một số pattern phổ biến của Microservices
- **SPRING CLOUD CONFIG:** Đảm bảo rằng bất kể bạn đưa ra bao nhiêu phiên bản microservice; chúng sẽ luôn có cùng cấu hình.
- **SERVICE REGISTRATION & DISCOVERY:** Các service mới sẽ được đăng ký và sau này consumer có thể gọi chúng thông qua tên hợp lý thay vì vị trí thực tế
- **ROUTING & TRACING:** Đảm bảo rằng tất cả lệnh gọi đến vi dịch vụ của bạn đều đi qua một "front door" (cửa trước) duy nhất trước khi service mục tiêu được gọi và service tương tự sẽ được truy tìm.
- **LOAD BALANCING:** Cân bằng tải phân phối hiệu quả lưu lượng mạng đến nhiều backend server hoặc nhóm server
- **SPRING CLOUD SECURITY:** Cung cấp các tính năng liên quan đến bảo mật dựa trên token trong các Microservice/Spring Boot
- **SPRING CLOUD NETFLIX:** Spring Cloud Netflix chủ yếu tập trung vào việc tích hợp các dịch vụ của Netflix vào các ứng dụng Spring. Ví dụ:
 - Service Discovery (Eureka): Là một dịch vụ đăng ký và phát hiện dịch vụ.
 - Circuit Breaker (Hystrix)
 - Intelligent Routing (Zuul): Là một cổng (gateway) và proxy dịch vụ. Nó cho phép quản lý, bảo mật và điều hướng các yêu cầu từ bên ngoài tới các microservice bên trong hệ thống.
 - Client Side Load Balancing (Ribbon): Là một thư viện cân bằng tải mặt trước (load balancing) cho các cuộc gọi giữa các microservice. Nó giúp phân phối tải đều giữa các phiên bản của một dịch vụ để tăng khả năng chịu tải và độ tin cậy.

4. Các phương pháp xác định ranh giới và kích thước phù hợp để xây dựng microservice

Một trong những khía cạnh thách thức nhất trong việc xây dựng một hệ thống microservice thành công là xác định các ranh giới microservice phù hợp và xác định kích thước của từng microservice

Dưới đây là các cách tiếp cận phổ biến nhất trong ngành:

- **Domain-Driven Sizing:** - Vì nhiều sửa đổi hoặc cải tiến của chúng tôi được thúc đẩy bởi nhu cầu kinh doanh, nên chúng tôi có thể xác định kích thước/xác định ranh giới của các microservice phù hợp chặt chẽ với khả năng Kinh doanh & Thiết kế theo hướng domain. Nhưng quá trình này mất rất nhiều thời gian và cần có kiến thức tốt về domain.
- **Event Storming Sizing:** Tiến hành một cuộc họp tương tác giữa các bên liên quan khác nhau để xác định danh sách các sự kiện quan trọng trong hệ thống như 'Completed Payment', 'Search for a Product', v.v. Dựa trên các sự kiện, chúng tôi có thể xác định 'Commands', 'Reactions' và có thể cố gắng nhóm chúng thành các domain-driven service.

5. Sizing & identifying boundaries with a Bank App use case

Bây giờ, hãy lấy một ví dụ về Bank application cần được xây dựng dựa trên kiến trúc vi dịch vụ và cố gắng xác định **ranh giới và kích thước** các dịch vụ.

Saving Account & Trading Account

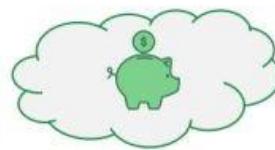


Cards & Loans



Sizing không chính xác khi có các module độc lập như Cards và Loans kết hợp với nhau

Saving Account



Trading Account



Cards



Loans



Saving Account



Trading Account



Debit Card



Credit Card



Home Loan



Vehicle Loan



Personal Loan

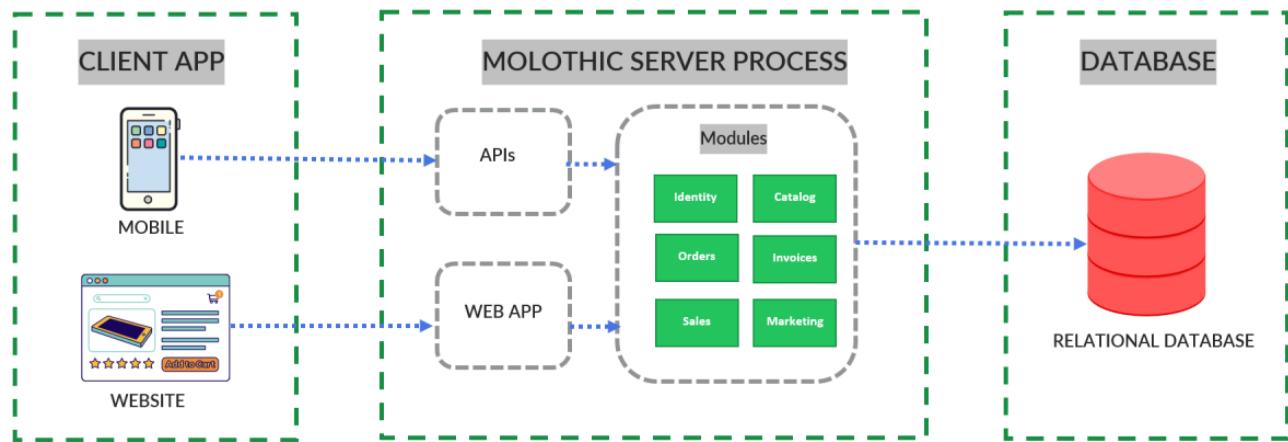


kích thước chính xác hợp lý nhất vì thế
thấy tất cả các mô-đun độc lập đều có
service riêng biệt duy trì liên kết lỏng lẻo
và có tính gắn kết cao

Sizing không chính xác khi có nhiều
service bên dưới Cards và Loans

6. Sizing & identifying boundaries with a Ecommerce migration use case

Bây giờ, hãy lấy một tình huống trong đó một E-Commerce startup (công ty khởi nghiệp Thương mại điện tử) đang tuân theo kiến trúc nguyên khối(monolithic) và cố gắng hiểu những thách thức với nó là gì.



Vấn đề mà nhóm **Ecommerce** đang gặp phải do thiết kế monolithic truyền thống

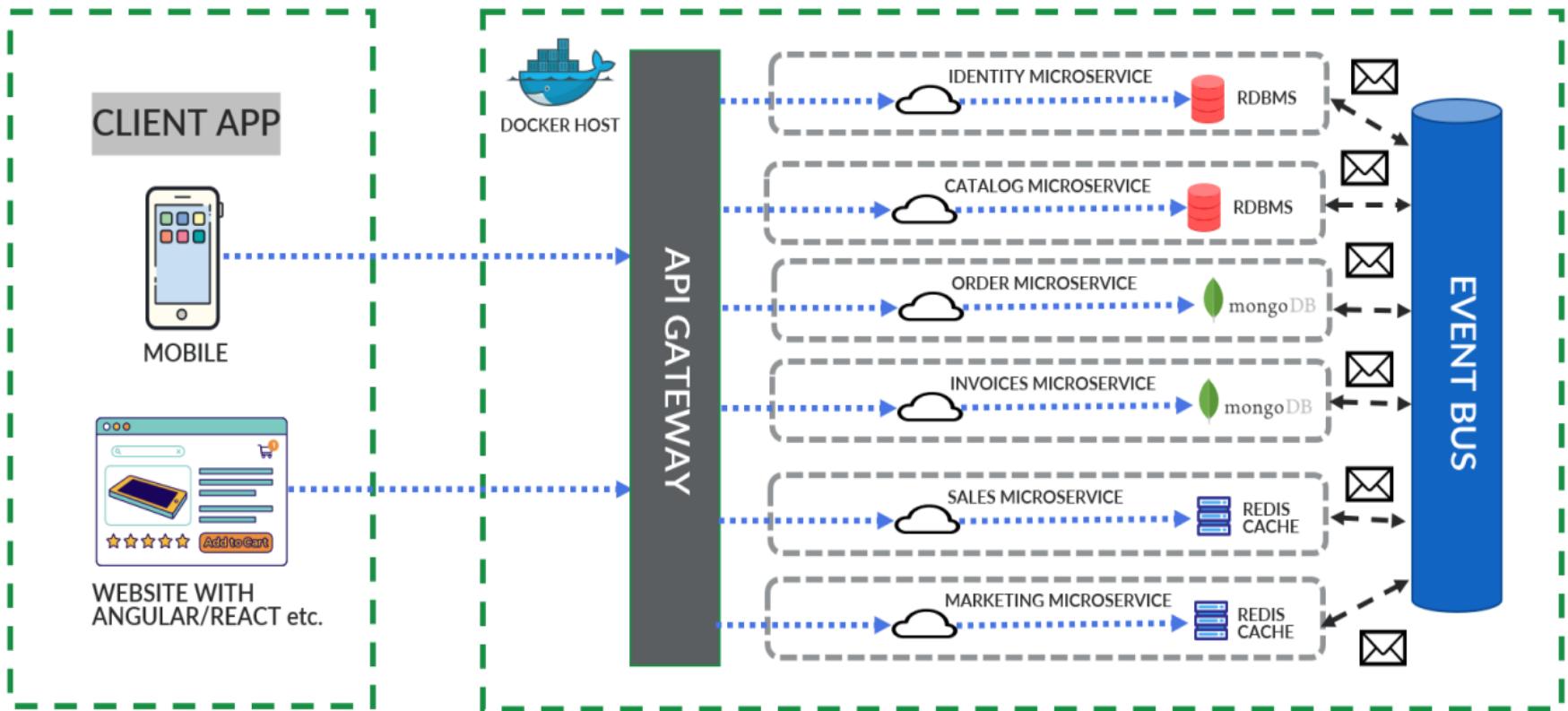
Ngày đầu tiên

- Dễ dàng xây dựng, thử nghiệm, triển khai, khắc phục sự cố và mở rộng quy mô trong quá trình khởi chạy và khi quy mô nhóm nhỏ

Sau vài ngày, ứng dụng/trang web này trở thành một siêu hit và bắt đầu phát triển rất nhiều. Bây giờ nhóm có vấn đề dưới đây,

- Ứng dụng đã trở nên cực kỳ phức tạp đến mức không một người nào hiểu được nó.
- Bạn sợ thực hiện các thay đổi - mỗi thay đổi đều có tác dụng phụ ngoài ý muốn và tốn kém.
- Các tính năng/bản sửa lỗi mới trở nên khó thực hiện, tốn thời gian và tốn kém.
- Mỗi lần phát hành càng nhỏ càng tốt và yêu cầu triển khai đầy đủ toàn bộ ứng dụng.
- Một thành phần không ổn định có thể làm hỏng toàn bộ hệ thống.
- Các công nghệ và khuôn khổ mới không phải là một lựa chọn.
- Rất khó để duy trì các nhóm nhỏ bị cô lập và thực hiện các phương pháp phân phối nhanh.

=> Vì vậy, **Ecommerce** đã quyết định và áp dụng thiết kế dựa trên cloud-native dưới đây bằng cách tận dụng kiến trúc Microservices để giúp dự án của họ trở nên dễ dàng và ít rủi ro hơn trước những thay đổi liên tục.

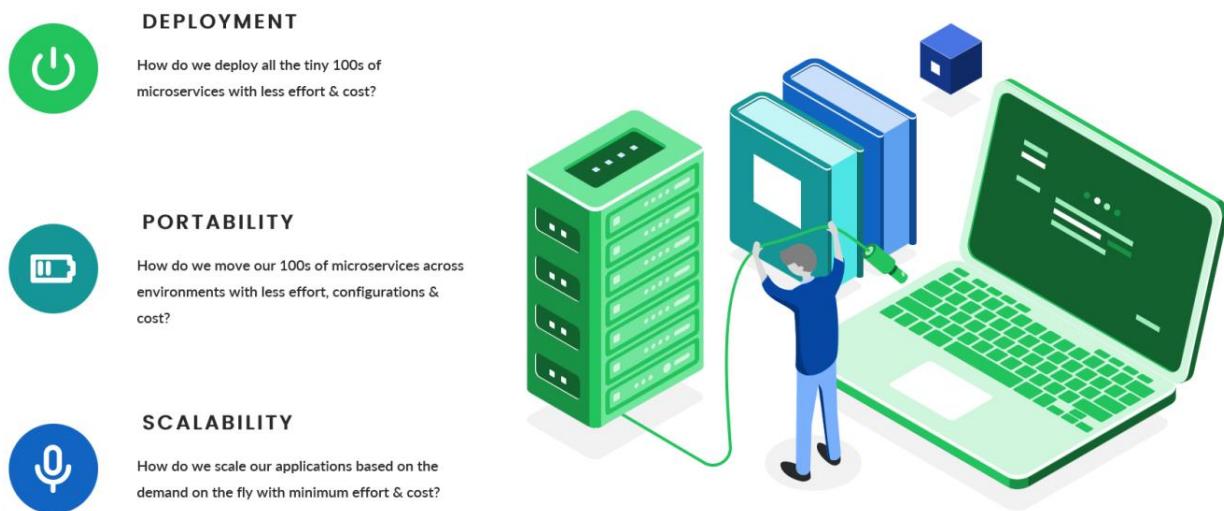


Session 3: How do we build, deploy, scale our microservices using Docker (Challenge 2 – code: example3)

1. Introduction to challenges while building, deploying microservices

Xây dựng và triển khai microservices có thể là một công việc thử thách nhưng cũng rất đáng để làm.

Dưới đây là một số thách thức thường gặp:



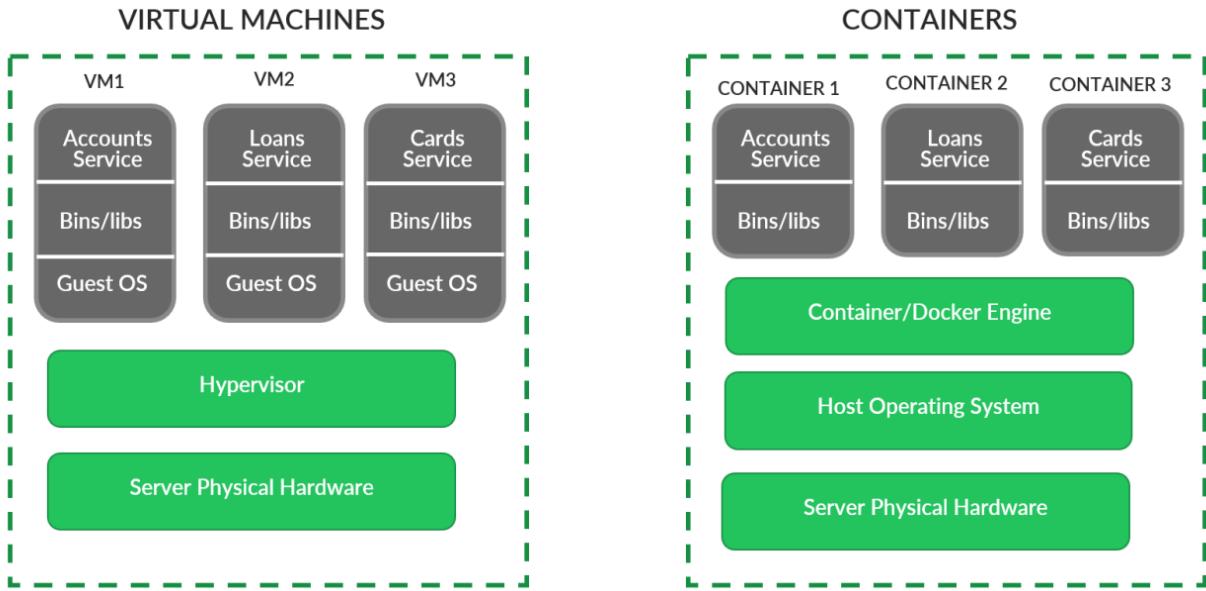
- **DEPLOYMENT:** Làm cách nào để chúng tôi triển khai tất cả 100 microservice với ít nỗ lực và chi phí hơn?
- **PORTABILITY:** Làm cách nào để chúng tôi di chuyển hơn 100 microservice qua các môi trường với ít nỗ lực, cấu hình và chi phí hơn?
- **SCALABILITY:** Làm cách nào để chúng tôi mở rộng quy mô các ứng dụng của mình dựa trên nhu cầu nhanh chóng với nỗ lực và chi phí tối thiểu?

2. Containerization technology

Công nghệ container (containerization technology) là một phương pháp giúp đóng gói ứng dụng và tất cả các thành phần phụ thuộc của nó, bao gồm các thư viện và cấu hình hệ thống, vào một gói duy nhất được gọi là container. Mỗi container hoạt động độc lập và cô lập với các container khác, cho phép ứng dụng chạy một cách đáng tin cậy và nhất quán trên nhiều môi trường máy tính.

Công nghệ container cho phép các ứng dụng chạy trên môi trường máy tính của bạn mà không cần lo ngại về sự khác biệt trong cấu hình hệ thống. Mỗi container bao gồm mọi thứ cần thiết để chạy ứng dụng, giúp tránh các xung đột giữa các phần mềm và đảm bảo tính ổn định của hệ thống.

Một trong những công nghệ container phổ biến nhất là Docker, nhưng có nhiều công nghệ container khác nhau như Podman, LXC và rkt. Containerization giúp tăng cường tính di động, linh hoạt và hiệu quả trong việc triển khai và quản lý ứng dụng, đồng thời cung cấp cách tiếp cận hiện đại để xây dựng và chạy các ứng dụng trên các môi trường máy tính hiện đại.



Sự khác biệt chính giữa virtual machines và container. Các Container không cần Guest Os cũng như trình ảo hóa(hypervisor) để chỉ định tài nguyên; thay vào đó, họ sử dụng container engine.

- Máy ảo: Máy ảo là một môi trường ảo hoàn chỉnh, bao gồm hệ điều hành và ứng dụng, chạy trên một lớp ảo hóa hypervisor. Mỗi VM có riêng lịch trình tài nguyên của nó (bộ nhớ, CPU, ổ cứng) và hoạt động như một máy tính độc lập.
- Container: Container là một phần cách ly ứng dụng và các phụ thuộc của nó, chia sẻ cùng một hạt nhân hệ điều hành với máy chủ chủ. Container chia sẻ hạt nhân, nhưng có môi trường gốc riêng biệt, nơi ứng dụng chạy.
- Container thường nhẹ nhàng, khởi động nhanh chóng và tiết kiệm tài nguyên hơn so với máy ảo, nhưng có cơ chế cách ly ít hơn. VMs cung cấp cách ly hoàn toàn và có thể chạy các hệ điều hành khác nhau, nhưng tốn nhiều tài nguyên hơn và khởi động chậm hơn

3. Definition of containers

Container là gì?

Container là một môi trường biệt lập lỏng lẻo cho phép xây dựng và chạy các **software package** (gói phần mềm). **Software package** này bao gồm mã và tất cả các thành phần phụ thuộc để chạy các ứng dụng một cách nhanh chóng và đáng tin cậy trên bất kỳ môi trường máy tính nào. Nó được gọi là **packages container image**.

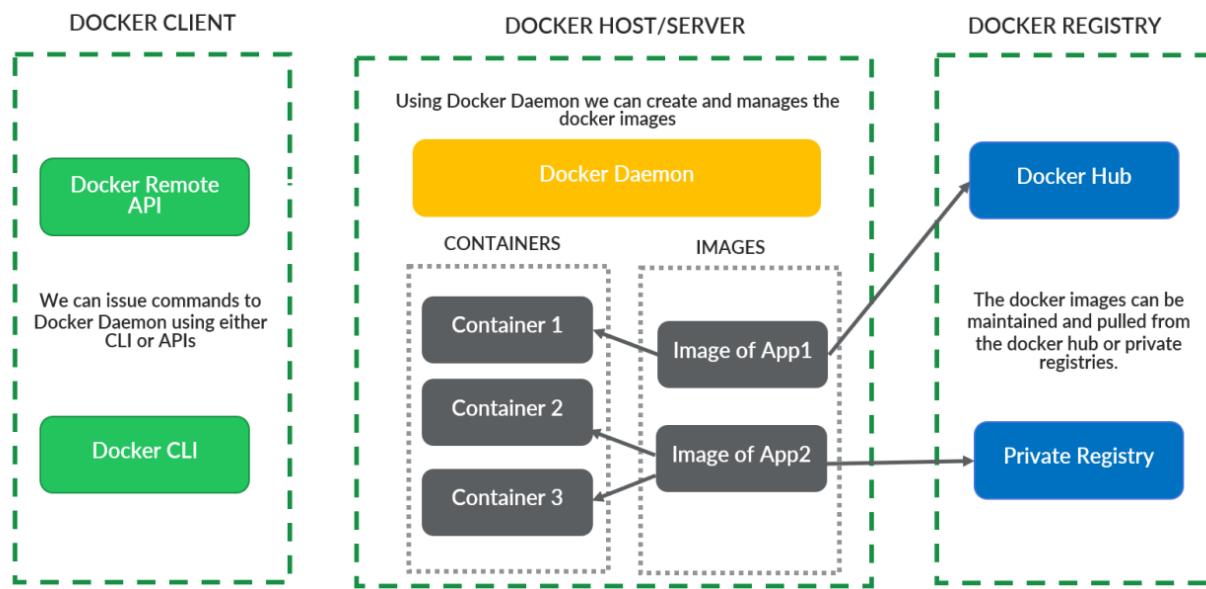
Software containerization

Software containerization (Bộ chứa phần mềm) là một phương pháp ảo hóa hệ điều hành được sử dụng để triển khai và chạy **container** mà không cần sử dụng máy ảo (VM). **Container** có thể chạy trên phần cứng vật lý, trên đám mây, máy ảo và trên nhiều hệ điều hành.

Docker là gì?

Docker là một trong những công cụ sử dụng ý tưởng về các tài nguyên bị cô lập để tạo ra một bộ công cụ cho phép đóng gói các ứng dụng với tất cả các phụ thuộc được cài đặt và chạy ở bất cứ đâu bạn muốn.

4. Introduction to Docker & its architecture



Docker là một công nghệ containerization (đóng gói ứng dụng vào container) phổ biến và được sử dụng rộng rãi trong việc triển khai ứng dụng trên các môi trường khác nhau. Docker cung cấp một cách đơn giản và hiệu quả để đóng gói ứng dụng và các thành phần liên quan của nó vào trong một container, giúp tăng tính di động, quản lý và triển khai.

Kiến trúc của Docker bao gồm các thành phần sau:

- **Docker daemon**: Là một tiến trình chạy trên máy chủ, quản lý các container và cung cấp các API để tương tác với Docker.
- **Docker client**: Là một công cụ dòng lệnh hoặc giao diện người dùng đồ họa (GUI) để tương tác với Docker daemon.
- **Docker image**: Là một tập hợp các lệnh và tập tin được đóng gói trong một định dạng chuẩn, định nghĩa cách để tạo ra một container.
- **Docker container**: Là một thực thể của Docker image, chứa tất cả các thành phần cần thiết để chạy ứng dụng.
- **Docker registry**: Là một kho lưu trữ để lưu trữ các Docker image, cho phép các nhà phát triển chia sẻ các image với nhau.

Khi một Docker image được tạo ra, nó có thể được lưu trữ trên Docker registry hoặc chia sẻ với các nhà phát triển khác. Khi một container được tạo ra từ một image, nó có thể được khởi chạy trên bất kỳ máy chủ nào có Docker daemon chạy trên đó.

Kiến trúc của Docker cung cấp tính di động và khả năng triển khai dễ dàng cho các ứng dụng, đồng thời giúp tối ưu hóa sử dụng tài nguyên của hệ thống.

5. Understanding Docker Hub & Installing Docker

Docker Hub là một dịch vụ kho lưu trữ ảnh Docker trực tuyến. Nó cung cấp cho người dùng một nơi để lưu trữ, quản lý và chia sẻ các ảnh Docker, và cũng là một nguồn tài nguyên phong phú để tìm kiếm các ảnh Docker chất lượng từ cộng đồng Docker.

Để sử dụng Docker Hub, người dùng cần đăng ký tài khoản và đăng nhập vào trang web Docker Hub. Sau đó, họ có thể tìm kiếm các ảnh Docker sẵn có trên Docker Hub hoặc tải lên các ảnh Docker của riêng mình để chia sẻ với cộng đồng.

Để cài đặt Docker, bạn có thể thực hiện các bước sau:

- Truy cập trang web chính thức của Docker và tải xuống phiên bản Docker phù hợp với hệ điều hành của bạn.
- Cài đặt Docker trên hệ thống của bạn bằng cách chạy tệp cài đặt và làm theo các hướng dẫn trên màn hình.
- Sau khi cài đặt xong, kiểm tra phiên bản Docker bằng cách chạy lệnh sau trong terminal hoặc command prompt:
`docker version`
- Nếu phiên bản Docker hiện tại được hiển thị, điều đó có nghĩa là Docker đã được cài đặt thành công trên hệ thống của bạn.

Sau khi cài đặt Docker, bạn có thể sử dụng Docker CLI (command-line interface) để quản lý các container và ảnh Docker trên hệ thống của bạn.

6. Creating Docker image definition using a Dockerfile

Để tạo Docker image cho dự án Spring Boot, bạn cần sử dụng Dockerfile. Dockerfile là một tệp văn bản đơn giản chứa các hướng dẫn để xây dựng Docker image của ứng dụng của bạn. Dưới đây là cách tạo một Dockerfile cho dự án Spring Boot cụ thể:

Bước 1: Chuẩn bị các tệp cần thiết:

Trước tiên, bạn nên đảm bảo rằng tệp .jar của ứng dụng Spring Boot đã được xây dựng thành công và có sẵn để triển khai. Đảm bảo rằng tệp .jar này được đặt trong thư mục gốc của dự án.

Vào thư mục của từng service:

```
mvn clean install
```

Bước 2: Tạo Dockerfile:

Tạo một tệp có tên "Dockerfile" trong thư mục gốc của dự án Spring Boot và thêm các hướng dẫn sau vào Dockerfile:

```
#Start with a base image containing Java runtime
FROM openjdk:17-jdk-slim as build

#Information around who maintains the image
MAINTAINER eazybytes.com

# Add the application's jar to the container
COPY target/accounts-0.0.1-SNAPSHOT.jar accounts-0.0.1-SNAPSHOT.jar

#execute the application
ENTRYPOINT ["java","-jar","/accounts-0.0.1-SNAPSHOT.jar"]
```

Bước 3: Xây dựng Docker image:

Sau khi bạn đã tạo Dockerfile, tiếp theo là xây dựng Docker image bằng cách chạy lệnh sau trong cửa sổ dòng lệnh:

```
docker build -t your-image-name:tag .
```

Trong đó:

- "your-image-name" là tên bạn muốn đặt cho image của bạn.
- "tag" là phiên bản hoặc tag của image (ví dụ: latest, v1.0, ...). Nếu không chỉ định tag, mặc định sẽ là "latest".
- Dấu chấm ở cuối lệnh chỉ định thư mục hiện tại làm nguồn cho việc xây dựng image.

```
docker build -t 1995mars/account .
```

Bước 4: Chạy container từ Docker image:

Khi Docker image đã được xây dựng, bạn có thể chạy container bằng cách chạy lệnh sau:

```
docker run -p 8080:8080 your-image-name:tag
```

Trong đó, "8080:8080" là cổng mà ứng dụng trong container sẽ lắng nghe và bạn có thể truy cập ứng dụng của mình qua cổng 8080 trên máy localhost.

```
docker run -p 8080:8080 1995mars/account
```

7. Deep dive of important Docker commands

Docker cung cấp một loạt các lệnh để quản lý các container, ảnh (image), mạng (network) và khối lưu trữ (volume). Sau đây là một số lệnh Docker quan trọng:

- **docker run:** Lệnh này được sử dụng để tạo và chạy một container từ một ảnh Docker. Ví dụ: docker run -it ubuntu bash sẽ tạo một container mới từ image Ubuntu và mở một phiên bản terminal bash bên trong container.
- **docker pull:** Lệnh này được sử dụng để tải về một image Docker từ Docker Hub hoặc một registry khác. Ví dụ: docker pull nginx sẽ tải về image Nginx mới nhất từ Docker Hub.
- **docker build:** Lệnh này được sử dụng để xây dựng một image Docker từ Dockerfile. Ví dụ: docker build -t myimage . sẽ xây dựng một image Docker từ Dockerfile được đặt trong thư mục hiện tại và đặt tên là "myimage".
- **docker ps:** Lệnh này được sử dụng để liệt kê tất cả các container đang chạy trên hệ thống. Ví dụ: docker ps sẽ liệt kê tất cả các container đang chạy trên hệ thống.
- **docker stop:** Lệnh này được sử dụng để dừng một container đang chạy. Ví dụ: docker stop mycontainer sẽ dừng container có tên "mycontainer".
- **docker rm:** Lệnh này được sử dụng để xóa một container. Ví dụ: docker rm mycontainer sẽ xóa container có tên "mycontainer".
- **docker rmi:** Lệnh này được sử dụng để xóa một image Docker. Ví dụ: docker rmi myimage sẽ xóa image Docker có tên "myimage".
- **docker network:** Lệnh này được sử dụng để quản lý các mạng Docker. Ví dụ: docker network create mynetwork sẽ tạo một mạng Docker mới có tên "mynetwork".

- **docker volume:** Lệnh này được sử dụng để quản lý các khối lưu trữ Docker. Ví dụ: docker volume create myvolume sẽ tạo một khối lưu trữ Docker mới có tên "myvolume".
- **docker exec:** Lệnh này được sử dụng để thực thi một lệnh bên trong một container đang chạy. Ví dụ: docker exec -it mycontainer bash sẽ mở một phiên bản terminal bash bên trong container có tên "mycontainer".

Trên đây là một số lệnh Docker quan trọng, tuy nhiên có nhiều lệnh khác nữa để quản lý các container và image Docker. Bạn nên tìm hiểu và sử dụng các lệnh này để quản lý các ứng dụng của mình trên Docker một cách hiệu quả.

8. Introduction to Buildpacks

Buildpacks là một công nghệ được sử dụng trong quá trình triển khai ứng dụng và các dịch vụ cloud để tự động hóa việc đóng gói ứng dụng và tạo các môi trường chạy mà không cần phải quan tâm đến cơ sở hạ tầng cụ thể. Buildpacks đảm bảo rằng ứng dụng của bạn được đóng gói một cách chính xác, có thể chạy trên các môi trường khác nhau một cách nhất quán, giúp giảm thiểu sự cố, tối ưu hóa hiệu suất và tăng tính di động.

Cơ chế hoạt động của Buildpacks hoạt động như sau:

- Phát hiện ngôn ngữ: Buildpacks xác định ngôn ngữ lập trình được sử dụng trong mã nguồn của ứng dụng.
- Đóng gói ứng dụng: Sau khi xác định ngôn ngữ, Buildpacks sẽ tạo ra môi trường chạy chính xác dựa trên yêu cầu của ứng dụng. Nó sẽ tự động tải xuống và cài đặt các phụ thuộc cần thiết, bao gồm phiên bản của ngôn ngữ lập trình, thư viện và công cụ cần thiết khác.
- Xây dựng ứng dụng: Sau khi xác định và chuẩn bị môi trường chạy, Buildpacks sẽ tiến hành xây dựng mã nguồn thành các thành phần thực thi có thể chạy được.
- Đóng gói: Cuối cùng, Buildpacks sẽ đóng gói ứng dụng đã xây dựng thành các container hoặc các định dạng triển khai ứng dụng khác nhau, sẵn sàng cho việc triển khai lên các môi trường cloud hoặc nền tảng PaaS (Platform as a Service).

Lợi ích chính của Buildpacks bao gồm:

- Tích hợp nền tảng: Buildpacks giúp bạn triển khai ứng dụng một cách dễ dàng và nhất quán trên các môi trường khác nhau, bao gồm các nhà cung cấp dịch vụ đám mây khác nhau như AWS, Azure, Google Cloud, Heroku, v.v.
- Tự động hóa: Buildpacks loại bỏ sự cần thiết phải cấu hình và tùy chỉnh môi trường chạy bằng cách thực hiện một phần lớn công việc tự động hóa, giảm nguy cơ sai sót do con người và tăng tính nhất quán.
- Độ tin cậy: Khi sử dụng Buildpacks, ứng dụng của bạn sẽ chạy trên một môi trường có độ tin cậy cao, giúp giảm thiểu sự cố và giữ cho ứng dụng của bạn luôn ổn định.
- Hiệu suất tối ưu: Các Buildpacks đảm bảo rằng các ứng dụng được xây dựng một cách tối ưu, điều này có thể dẫn đến hiệu suất tốt hơn và tiết kiệm tài nguyên.

Buildpacks đã trở thành một công nghệ phổ biến trong cộng đồng phát triển phần mềm và đang được sử dụng rộng rãi trong các dự án ứng dụng và hệ thống phức tạp.

Trong Spring Java, Buildpacks là một cơ chế để xây dựng và đóng gói ứng dụng Java thành các container hỗ trợ, chẳng hạn như Docker images. Buildpacks giúp tự động hóa quy trình xây dựng và triển khai ứng dụng, đồng thời đảm bảo rằng ứng dụng của bạn có thể chạy một cách đáng tin cậy và hiệu quả trên các môi trường khác nhau.

Các tính năng chính của Buildpacks trong Spring Java bao gồm:

- Xây dựng tự động: Buildpacks xác định ngôn ngữ lập trình và các thành phần phụ thuộc trong mã nguồn của ứng dụng Java và tự động tạo môi trường chạy phù hợp.
- Đóng gói container: Sau khi xây dựng ứng dụng, Buildpacks sẽ đóng gói ứng dụng thành các container như Docker images. Container là một cách phổ biến để triển khai ứng dụng đáng tin cậy và di động trên nhiều môi trường.
- Linh hoạt và nhất quán: Buildpacks cho phép bạn triển khai ứng dụng Java một cách nhất quán trên các nền tảng khác nhau, bao gồm các nhà cung cấp đám mây và các môi trường PaaS.
- Tối ưu hóa hiệu suất: Buildpacks giúp tối ưu hóa hiệu suất của ứng dụng Java bằng cách cung cấp môi trường chạy tối ưu và loại bỏ những thành phần không cần thiết.

Để sử dụng Buildpacks trong dự án Spring Java, bạn có thể sử dụng các công cụ hỗ trợ như Spring Boot CLI hoặc các framework triển khai ứng dụng Spring Boot như Spring Cloud, Spring Cloud Data Flow và Spring Cloud Services. Các công cụ này giúp bạn triển khai ứng dụng và sử dụng Buildpacks một cách dễ dàng và hiệu quả.

9. Creating docker image of Loans microservice using Buildpacks

Buildpacks tạo ra các image mà không cần phải tạo ra **Dockerfile**

Trong file pom.xml

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

Chạy lệnh command:

```
mvn spring-boot:build-image
```

Lệnh mvn spring-boot:build-image được sử dụng để build một Docker image từ một ứng dụng Spring Boot bằng cách sử dụng plugin "spring-boot-maven-plugin".

Khi chạy lệnh này, plugin sẽ đóng gói ứng dụng của bạn dưới dạng một file JAR và sau đó sử dụng Dockerfile được tạo ra tự động để build một image Docker. Image này sẽ có kích thước nhỏ hơn so với việc build image bằng cách sử dụng một hệ thống build khác, vì nó chỉ bao gồm các thành phần cần thiết để chạy ứng dụng Spring Boot.

10. Pushing Docker images from your local to remote Docker hub repository

Đăng nhập vào Docker Hub bằng lệnh:

```
docker login -u YOUR-USER-NAME
```

Trong command line, hãy thử chạy lệnh push mà bạn thấy trên Docker Hub. Lưu ý rằng lệnh của bạn sẽ sử dụng namespace của bạn, không phải "docker".

```
docker push 1995mars/account
# push image đến repository [docker.io/1995mars/ account]
# Một image không tồn tại cục bộ với tag: 1995mars/ account
```

11. Deep dive on docker-compose

Docker Compose là một công cụ giúp định nghĩa và chạy các ứng dụng Docker đa-container. Đây là một công cụ dòng lệnh sử dụng tệp YAML để khai báo các dịch vụ, mạng và khối lưu trữ cần thiết cho ứng dụng của bạn. Trong phần này, chúng ta sẽ khám phá các khía cạnh khác nhau của Docker Compose và cách nó có thể được sử dụng để đơn giản hóa việc quản lý các ứng dụng đa-container.

Cài đặt

Docker Compose được bao gồm trong cài đặt Docker, nhưng nó cũng có thể được cài đặt riêng biệt. Để cài đặt Docker Compose, bạn có thể làm theo các hướng dẫn được cung cấp trong tài liệu chính thức, phù hợp với hệ điều hành của bạn.

Để bắt đầu tất cả các docker image bằng một lệnh **docker compose** duy nhất, hãy cài đặt Docker compose bằng cách làm theo hướng dẫn được đề cập trong trang web:

<https://docs.docker.com/compose/install/>.

- Sau khi cài đặt docker compose, xác thực tương tự bằng cách chạy lệnh docker compose

```
docker-compose --version
```

Tệp YAML của Docker Compose

Docker Compose sử dụng tệp YAML để định nghĩa các services, networks, và volumes cần thiết cho ứng dụng của bạn. Tệp YAML có thể được cấu hình cao và có thể bao gồm một loạt các tùy chọn cho mỗi dịch vụ. Đây là một ví dụ về tệp YAML cơ bản:

```
# docker-compose.yml
version: "3.8"

services:

accounts:
  image: 1995mars/accounts:latest
  mem_limit: 700m
  ports:
    - "8080:8080"
networks:
```

```
- eazybank-network

loans:
  image: 1995mars/loans:latest
  mem_limit: 700m
  ports:
    - "8090:8090"
  networks:
    - eazybank-network

cards:
  image: 1995mars/cards:latest
  mem_limit: 700m
  ports:
    - "9000:9000"
  networks:
    - eazybank-network

networks:
  eazybank-network:
```

Chạy Docker Compose

Để chạy tệp Docker Compose, bạn có thể sử dụng lệnh docker-compose up. Lệnh này sẽ tạo và bắt đầu tất cả các dịch vụ được định nghĩa trong tệp YAML. Đây là một ví dụ về cách chạy tệp Docker Compose chúng ta đã định nghĩa trước đó:

docker-compose up

- Mở công cụ command line nơi chứa `docker-compose.yml` và chạy lệnh soạn thảo docker `"docker-compose up"` để khởi động tất cả các microservices container bằng một lệnh duy nhất. Tất cả các container đang chạy có thể được xác thực bằng cách chạy lệnh `docker ps`.
- Gọi API REST `http://localhost:8080/myAccount`, `http://localhost:8090/myLoans`, `http://localhost:9000/myCards` thông qua Postman bằng cách chuyển yêu cầu bên dưới ở định dạng JSON. Bạn sẽ nhận được phản hồi từ microservice tương ứng.

Quản lý Docker Compose

Docker Compose cung cấp một số lệnh để quản lý ứng dụng của bạn. Đây là một số lệnh thường được sử dụng nhất:

- `docker-compose up`: Tạo và bắt đầu tất cả các dịch vụ được định nghĩa trong tệp YAML.
- `docker-compose down`: Dừng và xóa tất cả các dịch vụ được tạo bởi `docker-compose up`.
- `docker-compose ps`: Liệt kê trạng thái của tất cả các dịch vụ được định nghĩa trong tệp YAML.
- `docker-compose logs`: Hiển thị nhật ký của tất cả các dịch vụ được định nghĩa trong tệp YAML.
- `docker-compose exec`: Chạy một lệnh trong một container đang chạy.

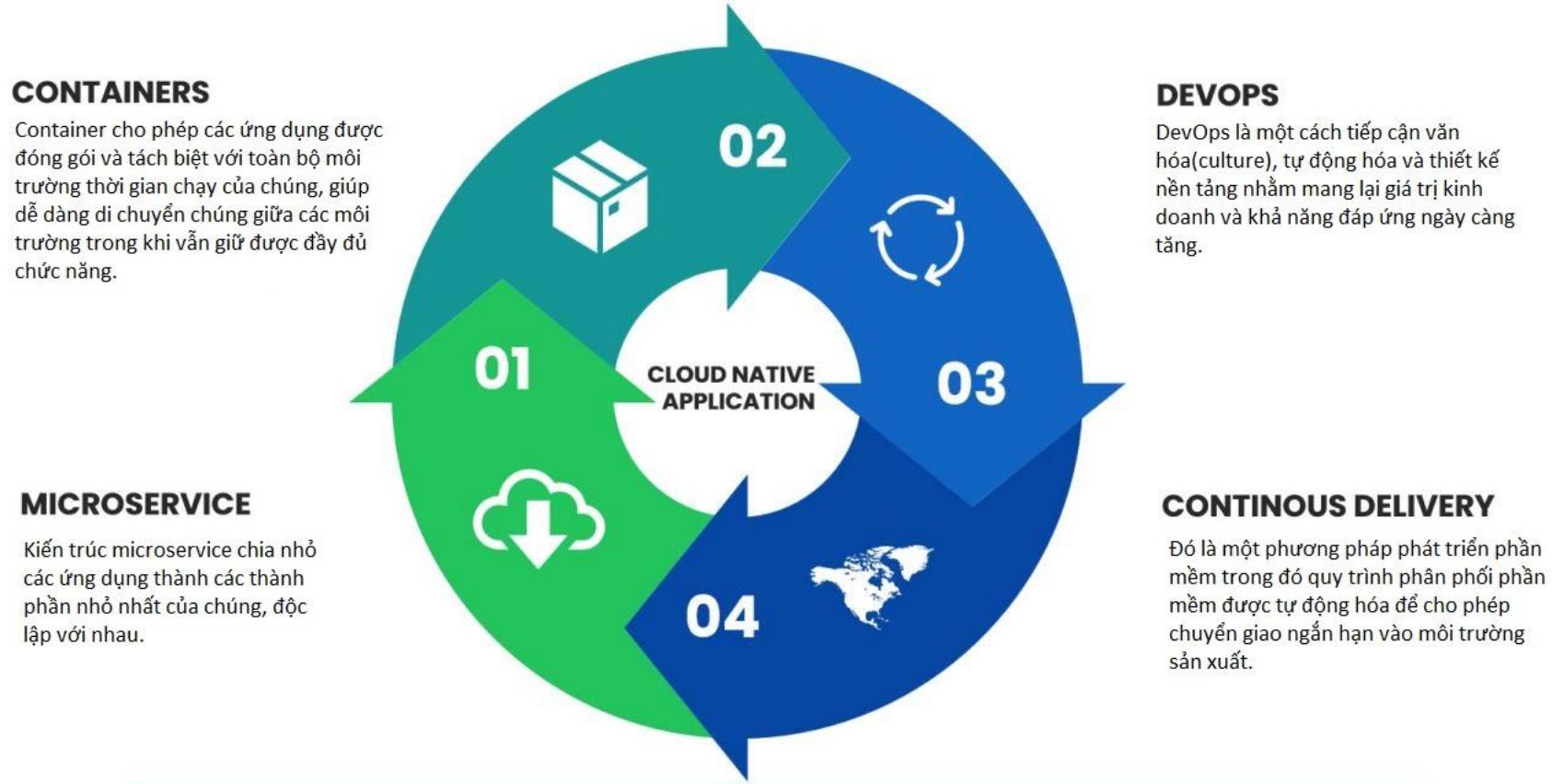
Section 4: [Optional] Deep Dive on Cloud Native Apps & 12factors

1. Introduction to Cloud-native applications

Giới thiệu

- Cloud-native applications là một tập hợp các dịch vụ nhỏ, độc lập và được liên kết lỏng lẻo. Chúng được thiết kế để mang lại business value (giá trị kinh doanh) đã được công nhận, chẳng hạn như khả năng kết hợp nhanh chóng phản hồi của người dùng để cải tiến liên tục. Mục tiêu của nó là cung cấp các ứng dụng mà người dùng muốn với tốc độ mà một doanh nghiệp cần.
- Nếu một ứng dụng là "cloud-native", thì ứng dụng đó được thiết kế đặc biệt để cung cấp trải nghiệm quản lý tự động và phát triển nhất quán trên các clouds riêng, công khai và kết hợp. Vì vậy, đó là về cách các ứng dụng được tạo và triển khai, chứ không phải ở đâu.
- Khi tạo các ứng dụng cloud-native, các nhà phát triển chia các chức năng thành các microservice, với các thành phần có thể mở rộng như container để có thể chạy trên một số máy chủ. Các dịch vụ này được quản lý bởi cơ sở hạ tầng ảo thông qua quy trình DevOps với quy trình phân phối liên tục. Điều quan trọng là phải hiểu rằng các loại ứng dụng này không yêu cầu bất kỳ thay đổi hoặc chuyển đổi nào để hoạt động trên cloud và được thiết kế để giải quyết tình trạng không có sẵn các thành phần hạ nguồn (downstream).

Nguyên tắc của ứng dụng đám mây



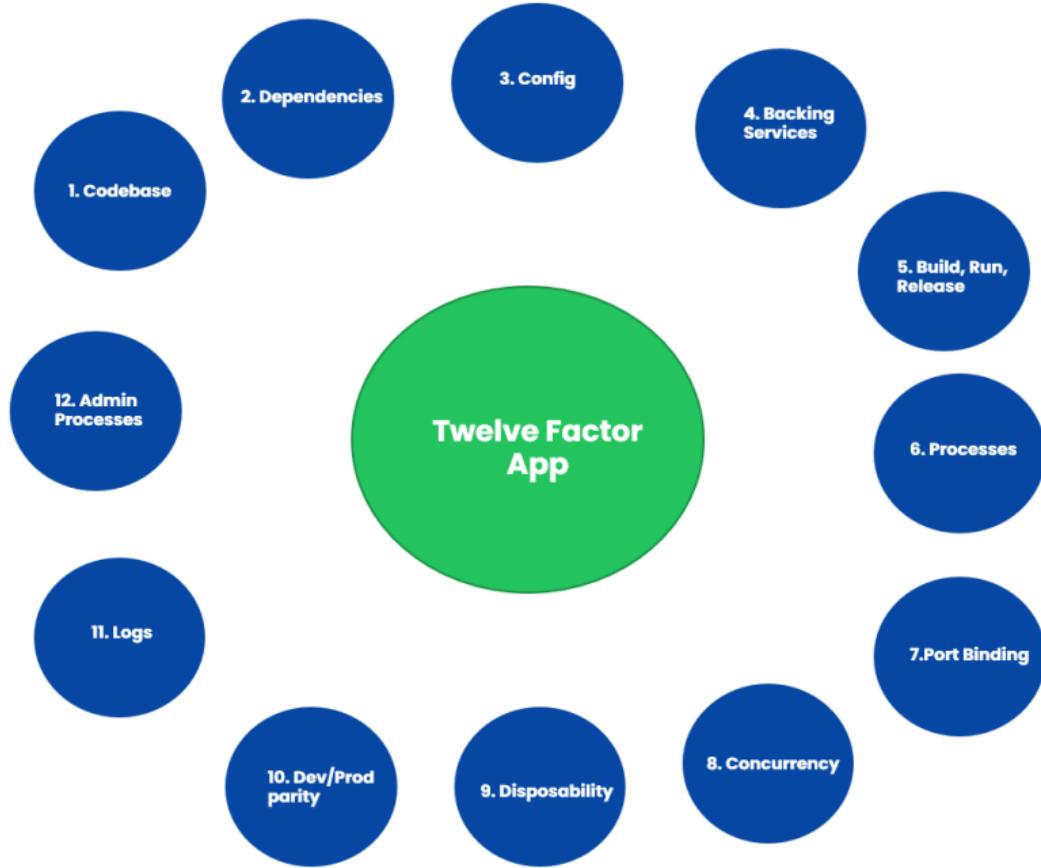
Chú ý:

DevOps Culture: Tiếp cận này kết hợp giữa phát triển và vận hành (operations), giúp các nhóm làm việc tăng tốc độ triển khai sản phẩm và dịch vụ, đồng thời giảm thiểu các vấn đề liên quan đến việc triển khai và vận hành.

2. Difference between cloud-native & traditional apps (khác nhau giữa ứng dụng truyền thống và đám mây)

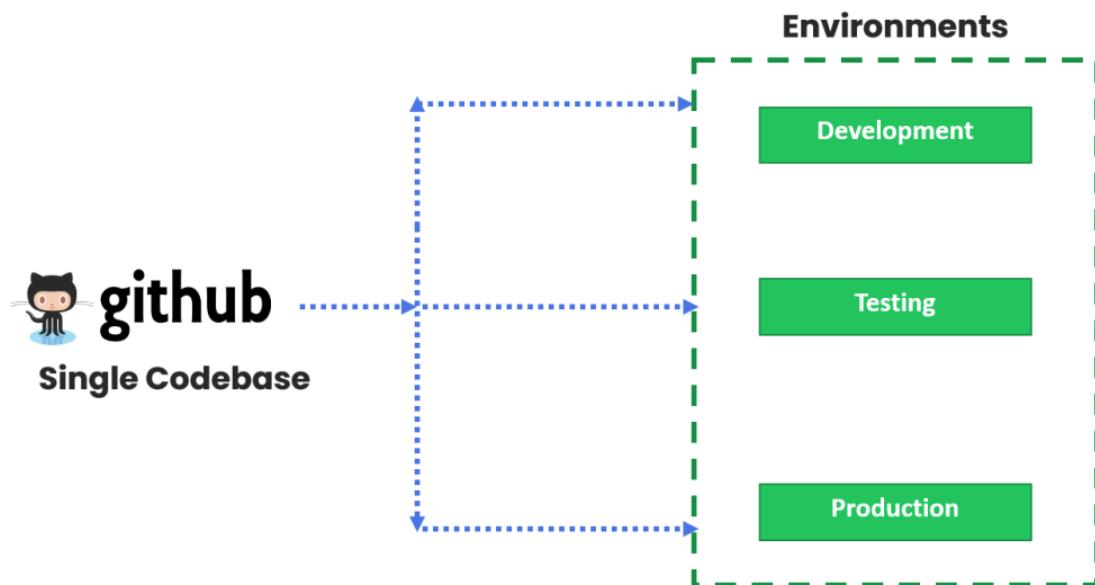
Cloud-native application	Traditional application
Hành vi dự đoán được (Predictable Behavior): <ul style="list-style-type: none"> Ứng dụng cloud-native được thiết kế để có thể đáp ứng được các yêu cầu về tài nguyên và tải lưu lượng một cách dự đoán được. Điều này giúp đảm bảo rằng ứng dụng luôn có thể hoạt động ổn định và đáp ứng được nhu cầu của người dùng. 	Hành vi không dự đoán được (Unpredictable Behavior) <ul style="list-style-type: none"> Ứng dụng truyền thống không được thiết kế để có thể đáp ứng được các yêu cầu về tài nguyên và tải lưu lượng một cách dự đoán được. Điều này có thể dẫn đến các sự cố và ảnh hưởng đến trải nghiệm của người dùng.
Trừu tượng hóa hệ điều hành (OS abstraction) <ul style="list-style-type: none"> Ứng dụng cloud-native được thiết kế để trừu tượng hóa hệ điều hành, giúp cho việc triển khai và quản lý ứng dụng được đơn giản hóa và dễ dàng hơn. 	Phụ thuộc vào hệ điều hành (OS dependent) <ul style="list-style-type: none"> Ứng dụng truyền thống phụ thuộc vào hệ điều hành để hoạt động, điều này làm cho việc triển khai và quản lý ứng dụng trở nên phức tạp hơn.
Quy mô phù hợp và độc lập (Right-sized capacity & Independent) <ul style="list-style-type: none"> Ứng dụng cloud-native có khả năng mở rộng theo nhu cầu và có khả năng hoạt động độc lập với các thành phần khác của hệ thống. Điều này giúp cho việc triển khai và quản lý ứng dụng trở nên dễ dàng hơn. 	Quy mô quá lớn và phụ thuộc vào các thành phần khác (Oversized capacity & Dependent) <ul style="list-style-type: none"> Ứng dụng truyền thống có quy mô quá lớn và phụ thuộc vào các thành phần khác của hệ thống, điều này có thể dẫn đến các vấn đề về hiệu suất và quản lý.
Triển khai liên tục (Continuous delivery) <ul style="list-style-type: none"> Ứng dụng cloud-native được thiết kế để triển khai liên tục, giúp cho việc cập nhật và phát triển ứng dụng trở nên nhanh chóng và thuận tiện hơn. 	Phát triển theo phong cách Waterfall (Waterfall development) <ul style="list-style-type: none"> Ứng dụng truyền thống được phát triển theo phương pháp Waterfall, điều này có nghĩa là các giai đoạn phát triển được thực hiện theo thứ tự tuyến tính, vì vậy việc cập nhật và phát triển ứng dụng trở nên chậm chạp và phức tạp hơn.
Khôi phục nhanh chóng và tự động mở rộng (Rapid recovery & Automated scalability) <ul style="list-style-type: none"> Ứng dụng cloud-native được thiết kế để có khả năng khôi phục nhanh chóng sau các sự cố và có khả năng mở rộng tự động theo nhu cầu. Điều này giúp cho hệ thống luôn hoạt động ổn định và đáp ứng được nhu cầu của người dùng. 	Khôi phục chậm (Slow recovery) <ul style="list-style-type: none"> Ứng dụng truyền thống có khả năng khôi phục chậm sau các sự cố, điều này có thể dẫn đến thời gian chết của hệ thống và ảnh hưởng đến trải nghiệm của người dùng.

3. Twelve factor app



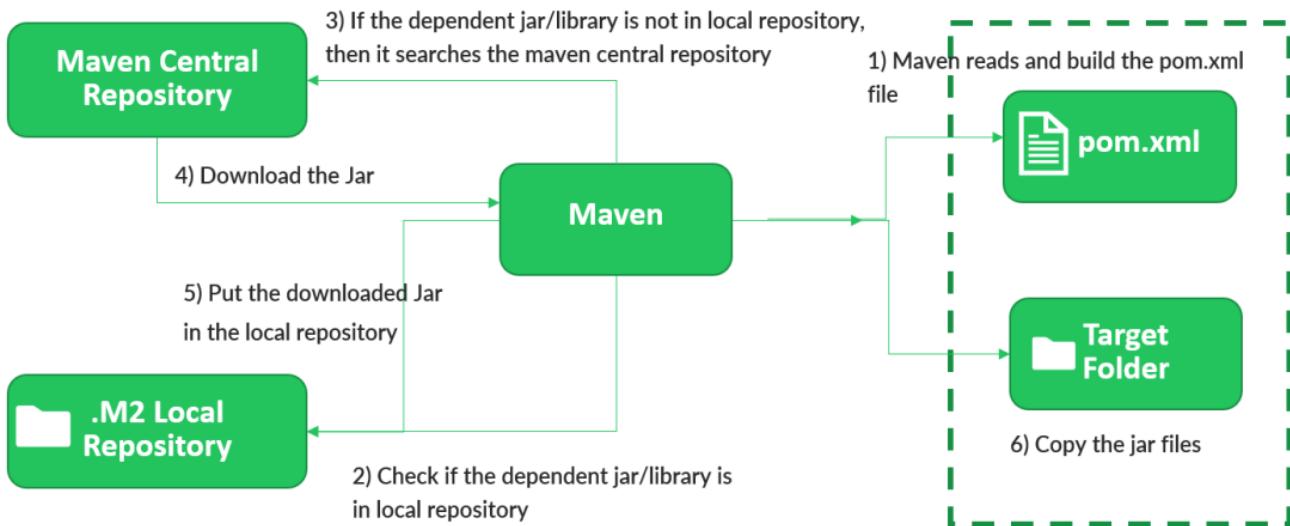
1. Codebase

Mỗi microservice phải có một cơ sở mã duy nhất, được quản lý trong source control. Cơ sở mã có thể có nhiều phiên bản môi trường triển khai như development, testing, staging, production, v.v. nhưng không được chia sẻ với bất kỳ microservice nào khác.



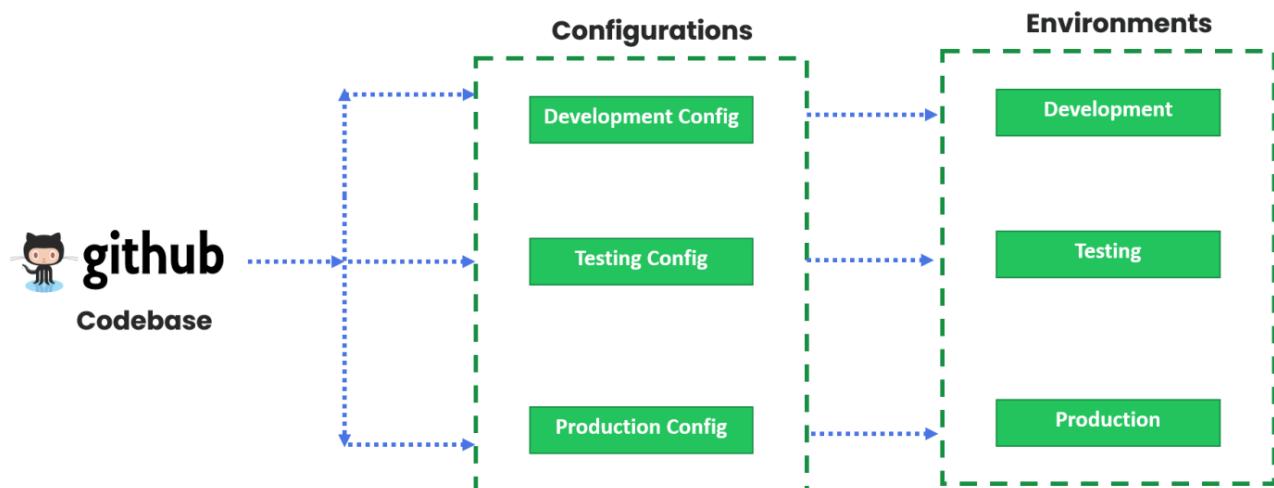
2. Dependencies

- Khai báo rõ ràng các dependency mà ứng dụng của bạn sử dụng thông qua các công cụ xây dựng như Maven, Gradle (Java). Sự phụ thuộc JAR của bên thứ ba phải được khai báo bằng cách sử dụng số phiên bản cụ thể của họ. Điều này cho phép microservice của bạn luôn được xây dựng bằng cùng một phiên bản thư viện.
- Một twelve-factor app không bao giờ phụ thuộc vào sự tồn tại ngầm định của các gói trên toàn hệ thống.



3. Config

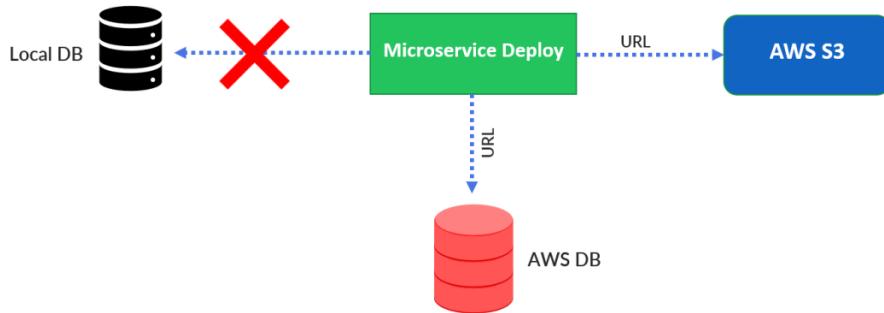
Lưu trữ cấu hình dành riêng cho môi trường độc lập với mã của bạn. Không bao giờ thêm cấu hình nhúng vào mã nguồn của bạn; thay vào đó, hãy duy trì cấu hình của bạn hoàn toàn tách biệt với microservice có thể triển khai của bạn. Nếu giữ nguyên cấu hình được đóng gói trong microservice, thì chúng tôi sẽ cần triển khai lại từng phiên bản trong số hàng trăm phiên bản để thực hiện thay đổi.



4. Backing services

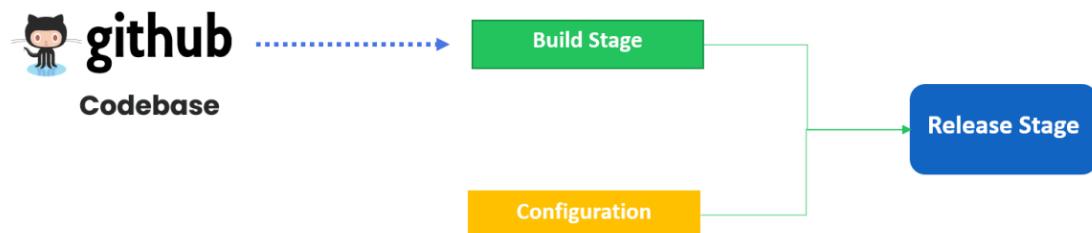
Phương pháp hay nhất về **Backing Services (Dịch vụ đằng sau)** chỉ ra việc triển khai microservice sẽ có thể hoán đổi giữa các kết nối cục bộ với bên thứ ba mà không có bất kỳ thay đổi nào đối với code.

Trong ví dụ dưới đây, chúng ta có thể thấy rằng local DB có thể được hoán đổi dễ dàng sang third-party DB là AWS DB tại đây mà không cần thay đổi mã.



5. Build, release, run

Tách biệt hoàn toàn các giai build, release, and run trong quá trình triển khai ứng dụng của bạn. Chúng ta sẽ có thể xây dựng các microservice độc lập với môi trường mà chúng đang chạy.



6. Processes

Thực thi ứng dụng dưới dạng một hoặc nhiều quy trình không trạng thái. Các quy trình twelve-factor là không trạng thái và không chia sẻ gì. Mọi dữ liệu cần duy trì phải được lưu trữ trong một dịch vụ sao lưu có trạng thái, điển hình là cơ sở dữ liệu.

Microservices có thể bị tắt và thay thế bất cứ lúc nào mà không sợ mất một phiên bản dịch vụ sẽ dẫn đến mất dữ liệu.



We can store the data of the loans microservice inside a SQL or NoSQL DB

7. Port binding(ràng buộc cổng)

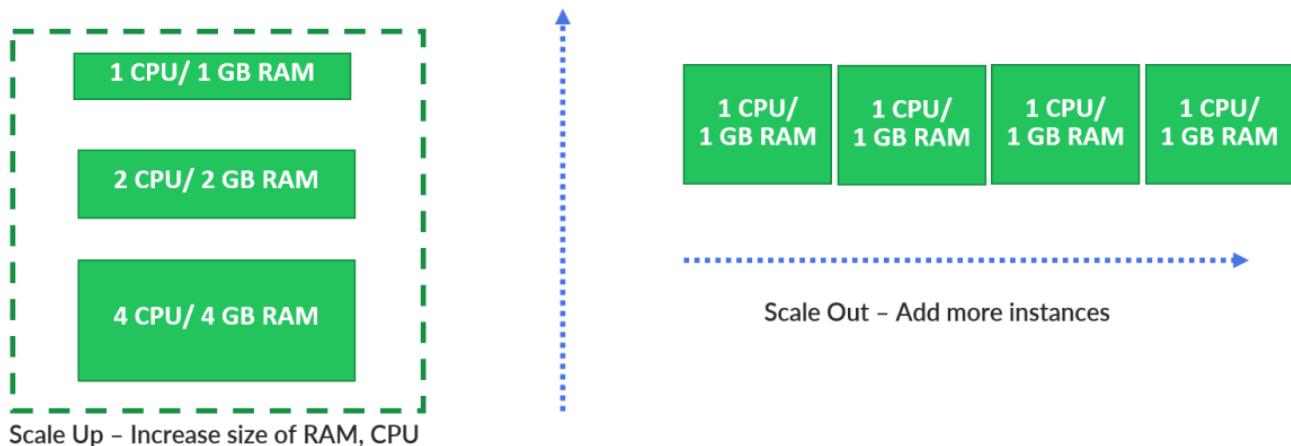
Các ứng dụng web đôi khi được thực thi bên trong webserver container (bộ chứa máy chủ web). Ví dụ: các ứng dụng PHP có thể chạy dưới dạng một mô-đun bên trong Apache HTTPD hoặc các ứng dụng Java có thể chạy bên trong Tomcat. Nhưng mỗi microservice phải độc lập với các giao diện và chức năng được hiển thị trên cổng riêng của nó. Làm như vậy sẽ tạo ra sự cách ly với các microservice khác.

Bạn có thể phát triển một ứng dụng bằng Spring Boot. Spring Boot, ngoài nhiều lợi ích khác, còn cung cấp cho chúng ta một máy chủ ứng dụng nhúng mặc định. Do đó, JAR mà chúng tôi đã tạo trước đó bằng Maven hoàn toàn có khả năng thực thi trong bất kỳ môi trường nào chỉ bằng cách có Java runtime environment tương thích.

8. Concurrency(đồng thời)

Các dịch vụ mở rộng quy mô trên một số lượng lớn các quy trình nhỏ giống hệt nhau (bản sao) thay vì mở rộng quy mô một phiên bản lớn duy nhất trên máy mạnh nhất hiện có.

Mở rộng quy mô theo chiều dọc (Scale up) đề cập đến việc tăng cơ sở hạ tầng phần cứng (CPU, RAM). Chia tỷ lệ theo chiều ngang (Scale out) đề cập đến việc thêm nhiều phiên bản của ứng dụng. Khi bạn cần mở rộng quy mô, hãy khởi chạy nhiều phiên bản microservice hơn và mở rộng quy mô chứ không tăng quy mô.



9. Disposability(Tính sẵn sàng)

Các phiên bản dịch vụ phải sẵn sàng, ưu tiên khởi động nhanh để tăng cơ hội mở rộng quy mô và tắt máy nhanh chóng để hệ thống ở trạng thái chính xác. Bộ chứa Docker cùng với bộ điều phối vốn đã đáp ứng yêu cầu này.

Ví dụ: nếu một trong các phiên bản của microservice bị lỗi do phần cứng cơ bản bị lỗi, chúng ta có thể tắt phiên bản đó mà không ảnh hưởng đến các microservice khác và bắt đầu một phiên bản khác ở nơi khác nếu cần.

10. Dev/prod parity (Tương đồng giữa phát triển và sản phẩm)

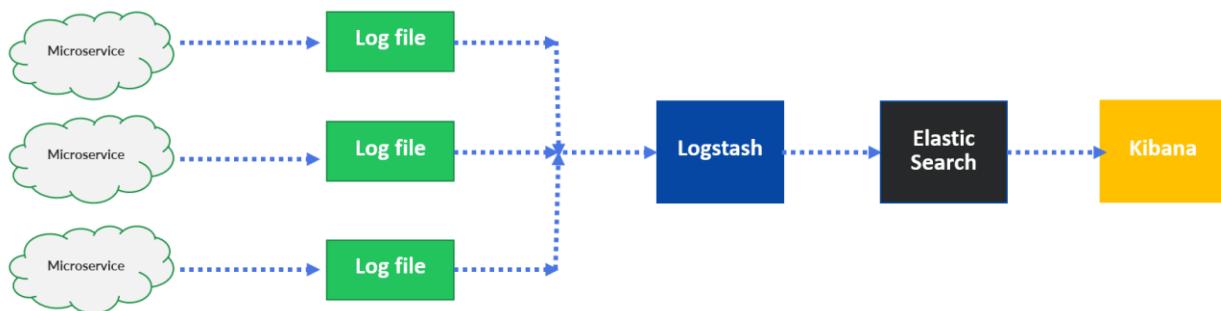
Giữ cho các môi trường trong vòng đời của ứng dụng càng giống nhau càng tốt, tránh các lỗi tắt tốn kém. Ở đây, việc áp dụng các container có thể đóng góp rất nhiều bằng cách thúc đẩy cùng một môi trường thực thi.

Ngay sau khi một code được commit, code đó phải được kiểm tra và sau đó xuc tiến càng nhanh càng tốt từ quá trình phát triển cho đến sản xuất. Hướng dẫn này rất cần thiết nếu chúng ta muốn tránh các lỗi triển khai. Việc có các môi trường sản xuất và phát triển tương tự cho phép chúng tôi kiểm soát tất cả các tình huống có thể xảy ra khi triển khai và thực thi.

11. Logs

Xử lý log do microservice tạo dưới dạng luồng sự kiện. Khi log được viết ra, chúng phải được quản lý bằng các công cụ, chẳng hạn như Logstash (<https://www.elastic.co/products/logstash>) sẽ thu thập log và ghi chúng vào một vị trí trung tâm.

Microservice không bao giờ nên quan tâm đến cơ chế xảy ra điều này như thế nào, chúng chỉ cần tập trung vào việc ghi các mục log vào thiết bị xuất chuẩn. Chúng ta sẽ thảo luận về cách cung cấp cấu hình tự động để gửi các nhật ký này tới ngăn xếp ELK (Elasticsearch, Logstash và Kibana).



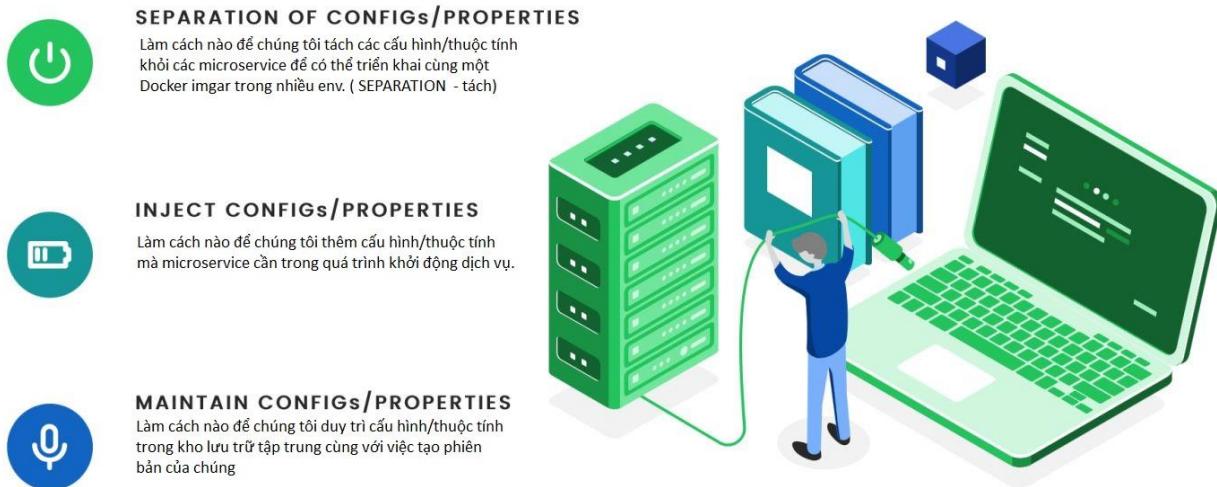
12. Admin processes (quy trình quản trị)

Chạy các tác vụ quản trị/quản lý dưới dạng quy trình một lần. Các nhiệm vụ có thể bao gồm dọn dẹp dữ liệu và lấy số liệu phân tích cho báo cáo. Các công cụ thực hiện các tác vụ này phải được gọi từ môi trường product, nhưng tách biệt với ứng dụng.

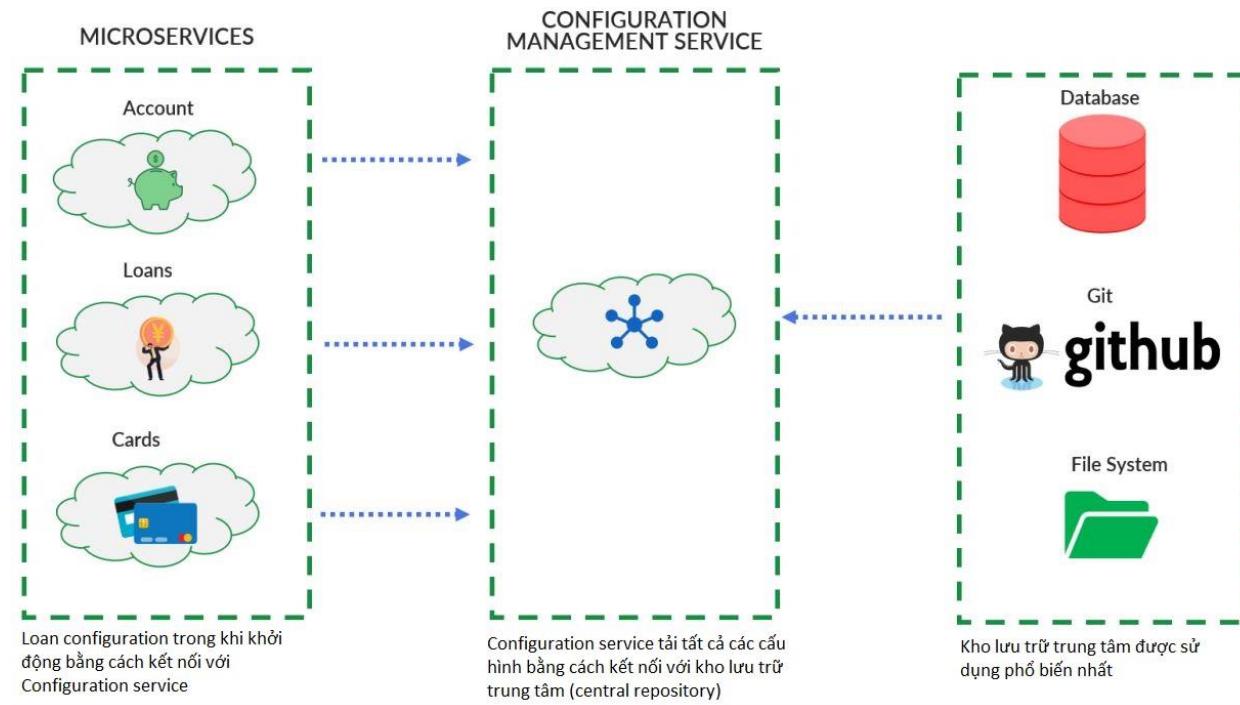
Các nhà phát triển thường sẽ phải thực hiện các tác vụ quản trị liên quan đến microservice của họ như Di chuyển dữ liệu, dọn dẹp các hoạt động. Các tác vụ này không bao giờ được thực hiện đột xuất và thay vào đó nên được thực hiện thông qua các tập lệnh được quản lý và duy trì thông qua kho lưu trữ mã nguồn(source code repository). Các tập lệnh này phải có thể lặp lại và không thay đổi trong từng môi trường mà chúng chạy trên đó. Điều quan trọng là phải xác định các loại tác vụ mà chúng tôi cần cân nhắc khi chạy vi dịch vụ của mình, trong trường hợp chúng tôi có nhiều microservice với các tập lệnh này, chúng tôi có thể thực thi tất cả các tác vụ quản trị mà không cần phải thực hiện thủ công.

Section 5: Configurations Management in Microservices (Challenge 3 – code: section5)

1. Introduction to Configurations Management challenges inside microservices

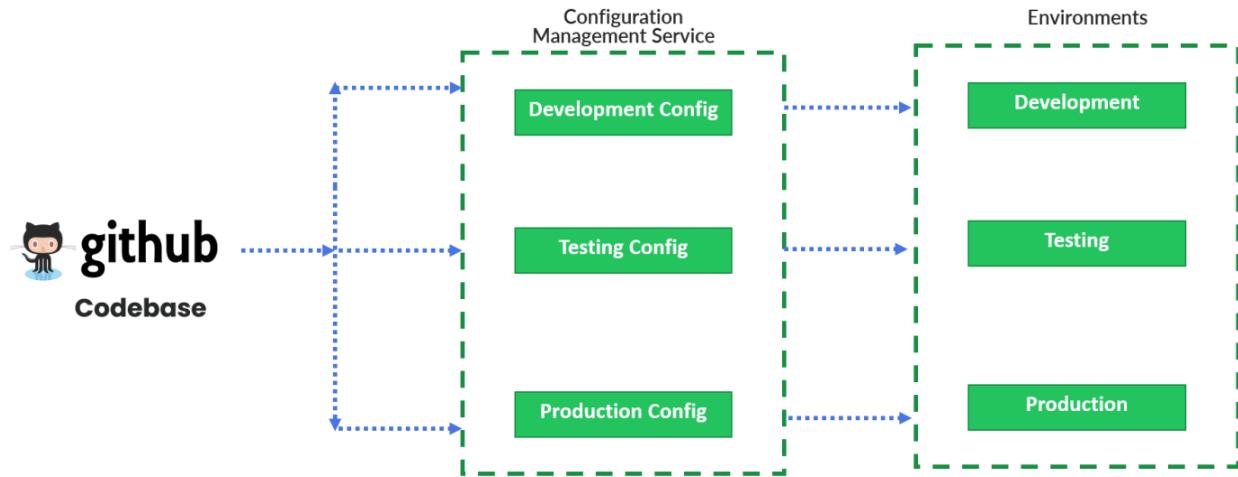


Architecture inside microservices



2. Deep dive of Spring Cloud Config for Configuration management

Spring Cloud Config cung cấp hỗ trợ phía server và client-side cho cấu hình bên ngoài trong một hệ thống phân tán. Với Config Server, bạn có một vị trí trung tâm để quản lý các thuộc tính bên ngoài cho các ứng dụng trên tất cả các môi trường.



Các tính năng của Spring Cloud Config Server:

- HTTP, API dựa trên tài nguyên cho cấu hình bên ngoài (cặp name-value hoặc nội dung YAML tương đương)
- Mã hóa và giải mã các giá trị thuộc tính
- Có thể dễ dàng nhúng vào ứng dụng Spring Boot application bằng cách sử dụng `@EnableConfigServer`

Các tính năng của Config Client (dành cho Microservices):

- Liên kết với Config Server và khởi tạo Spring Environment với các nguồn thuộc tính từ xa
- Mã hóa và giải mã các giá trị thuộc tính

3. Building Config Server service and load all the configurations from classpath

Để xây dựng dịch vụ Config Server và tải tất cả các cấu hình từ đường dẫn classpath, bạn có thể sử dụng Spring Cloud Config Server. Đây là một trong những công cụ phổ biến trong môi trường Spring để quản lý các cấu hình ứng dụng.

Bước 1: Chuẩn bị dự án

Trước tiên, bạn cần tạo một dự án Spring Boot mới. Bạn có thể sử dụng Spring Initializr hoặc cài đặt Maven hoặc Gradle để tạo dự án.

Bước 2: Thêm các phụ thuộc

Trong file pom.xml (hoặc build.gradle nếu bạn sử dụng Gradle), bạn cần thêm các phụ thuộc sau:

```
<dependencies>
    ...
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-server</artifactId>
    </dependency>
    ...
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Bước 3: Cấu hình Config Server

Tạo một lớp chứa cấu hình cho Config Server. Ví dụ:

```
@SpringBootApplication
@EnableConfigServer
public class ConfigserverApplication {

    public static void main(String[] args) {
        SpringApplication.run(ConfigserverApplication.class, args);
    }
}
```

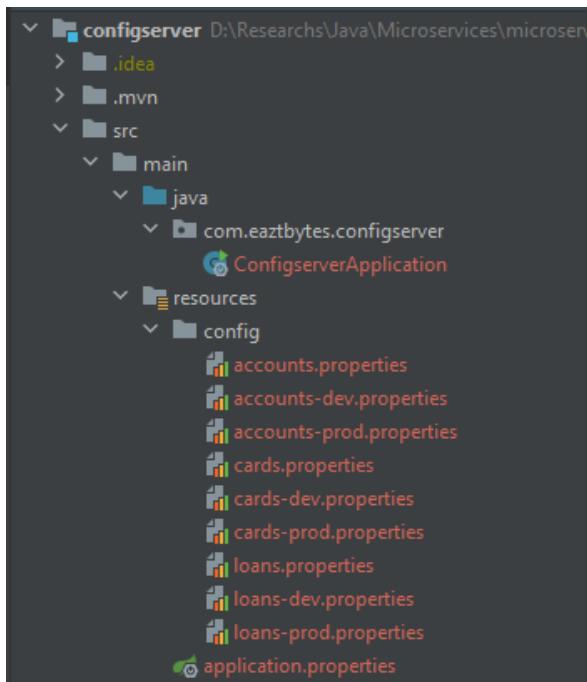
Bước 4: Cấu hình đường dẫn classpath cho cấu hình

Để tải cấu hình từ classpath, bạn chỉ cần đặt các file cấu hình của ứng dụng và các file cấu hình cho microservice (ví dụ: application.properties hoặc application.yml) trong thư mục resources trên classpath của dự án.

Ví dụ, trong thư mục resources, tạo một file có tên "application.properties" hoặc "application.yml" và đặt các cấu hình của bạn trong đó.

Cấu hình application.properties:

```
spring.application.name=configserver
spring.profiles.active=native
spring.cloud.config.server.native.searchLocations=classpath:/config
server.port=8071
```



Bước 5: Chạy ứng dụng Config Server

Sau khi cấu hình xong, bạn có thể chạy ứng dụng Config Server của mình. Khi nó đã hoạt động, Config Server sẽ tải cấu hình từ classpath và sẵn sàng phục vụ các ứng dụng khác thông qua HTTP endpoint.

<http://localhost:8071/accounts/default>

The screenshot shows a browser window with the URL `http://localhost:8071/accounts/default`. The page displays a JSON configuration object:

```
{
  "name": "accounts",
  "profiles": [
    "default"
  ],
  "label": null,
  "version": null,
  "state": null,
  "propertySources": [
    {
      "name": "classpath:/config/accounts.properties",
      "source": {
        "accounts.msg": "Welcome to the EazyBank Accounts Default application",
        "accounts.build-version": "3",
        "accounts.mailDetails.hostName": "default-accounts@eazybytes.com",
        "accounts.mailDetails.port": "9000",
        "accounts.mailDetails.from": "default-accounts@eazybytes.com",
        "accounts.mailDetails.subject": "Your Account Details from Eazy Bank Default Environment",
        "accounts.activeBranches[0)": "Mumbai",
        "accounts.activeBranches[1)": "London",
        "accounts.activeBranches[2)": "Washington"
      }
    }
  ]
}
```

Lưu ý: Đảm bảo rằng các ứng dụng khác trong hệ thống của bạn cũng được cấu hình để tìm kiếm cấu hình từ Config Server. Bạn có thể làm điều này bằng cách sử dụng các thư viện Spring Cloud Config Client trong các ứng dụng khác.

4. Reading configurations from a file system location

Cần thay đổi đường dẫn classpath cho cấu hình (`application.properties`)

```
spring.application.name=configserver
spring.profiles.active=native
spring.cloud.config.server.native.searchLocations=file:///C:/config
server.port=8071
```

Lưu ý: tạo các file cấu hình cho microservice trong đường dẫn

5. Reading configurations from a GitHub repository

Cần thay đổi đường dẫn classpath cho cấu hình (`application.properties`)

```
spring.application.name=configserver
spring.profiles.active=git

#Đường dẫn cấu hình
spring.cloud.config.server.git.uri=https://github.com/eazybytes/microservices-
config.git
spring.cloud.config.server.git.clone-on-start=true

#Brach-nhánh sử dụng
spring.cloud.config.server.git.default-label=main
server.port=8071
```

Ví dụ cấu hình được lưu trữ trong github:

The screenshot shows a GitHub repository page for 'eazybytes/microservices-config'. The main navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the navigation, there are buttons for Go to file, Add file, and Code. The 'Code' button is highlighted with a green background. The repository has 1 branch and 0 tags. A commit from user 'eazybytes' is shown, dated yesterday, with a commit message 'Thanks for choosing to learn from EazyBytes' and a SHA of '0fb09ec'. The commit details a list of files: README.md, accounts-dev.properties, accounts-prod.properties, accounts.properties, cards-dev.properties, cards-prod.properties, cards.properties, loans-dev.properties, loans-prod.properties, and loans.properties. Each file has a commit message 'Thanks for choosing to learn from EazyBytes' and is dated yesterday.

File	Commit Message	Date
README.md	Initial commit	yesterday
accounts-dev.properties	Thanks for choosing to learn from EazyBytes	yesterday
accounts-prod.properties	Thanks for choosing to learn from EazyBytes	yesterday
accounts.properties	Thanks for choosing to learn from EazyBytes	yesterday
cards-dev.properties	Thanks for choosing to learn from EazyBytes	yesterday
cards-prod.properties	Thanks for choosing to learn from EazyBytes	yesterday
cards.properties	Thanks for choosing to learn from EazyBytes	yesterday
loans-dev.properties	Thanks for choosing to learn from EazyBytes	yesterday
loans-prod.properties	Thanks for choosing to learn from EazyBytes	yesterday
loans.properties	Thanks for choosing to learn from EazyBytes	yesterday

6. Updating Accounts Microservice to read properties from Config Server (các microservice lấy cấu hình từ server config)

Dưới đây là hướng dẫn để xây dựng một microservice Config Client:

Bước 1: Tạo một dự án Spring Boot mới

Trước tiên, bạn cần tạo một dự án Spring Boot mới cho Config Client của mình. Bạn có thể sử dụng Spring Initializr hoặc cài đặt Maven hoặc Gradle để tạo dự án. (đã xây dựng trước đó)

Bước 2: Thêm các phụ thuộc

Trong file pom.xml (hoặc build.gradle nếu bạn sử dụng Gradle), bạn cần thêm các phụ thuộc sau để sử dụng Spring Cloud Config Client:

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

Bước 3: Cấu hình Config Client

Cấu hình Config Client để có thể tìm kiếm và đọc cấu hình từ Config Server. Bạn có thể làm điều này bằng cách chỉ định đường dẫn của Config Server trong file **application.properties**

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
server.port=8080

# Config Client
spring.application.name=accounts
spring.profiles.active=prod
spring.config.import=optional:configserver:http://localhost:8071/
management.endpoints.web.exposure.include=*
```

Bước 4: Đọc cấu hình từ Config Server

Khi bạn đã cấu hình Config Client để tìm kiếm Config Server, bạn có thể đọc cấu hình từ Config Server và sử dụng nó trong ứng dụng của mình. Điều này giúp bạn tránh việc cần phải cấu hình cứng trong mã nguồn của ứng dụng và có thể thay đổi cấu hình mà không cần phải triển khai lại ứng dụng.

Bạn có thể sử dụng các cấu trúc giống như @Value hoặc @ConfigurationProperties để đọc các thuộc tính cấu hình từ Config Server và sử dụng chúng trong mã của mình.

Ví dụ sử dụng @Value trong Spring Boot:

```
@Configuration
@ConfigurationProperties(prefix = "accounts")
@Getter @Setter @ToString
public class AccountsServiceConfig {

    private String msg;
    private String buildVersion;
    private Map<String, String> mailDetails;
    private List<String> activeBranches;

}
```

Bước 5: Xem cấu hình từ Config Server (ví dụ xem các config đã đọc)

Tạo RestController để trả về config đã lấy từ server config

```
@RestController
public class AccountsController {

    @Autowired
    private AccountsRepository accountsRepository;

    @Autowired
    AccountsServiceConfig accountsConfig;

    @GetMapping("/account/properties")
    public String getPropertyDetails() throws JsonProcessingException {
        ObjectWriter ow = new
ObjectMapper().writer().withDefaultPrettyPrinter();
        Properties properties = new Properties(accountsConfig.getMsg(),
accountsConfig.getBuildVersion(),
            accountsConfig.getMailDetails(),
accountsConfig.getActiveBranches());
        String jsonStr = ow.writeValueAsString(properties);
        return jsonStr;
    }
}
```

7. Generating Docker images after Config Server changes

Trong file pom.xml

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <image>
          <name>eazybytes/${project.artifactId}</name>
        </image>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Chạy lệnh command:

```
mvn spring-boot:build-image
```

Lệnh mvn spring-boot:build-image được sử dụng để build một Docker image từ một ứng dụng Spring Boot bằng cách sử dụng plugin "spring-boot-maven-plugin".

8. Pushing all the latest Docker images with Config server changes to DockerHub

Trong command line, hãy thử chạy lệnh push mà bạn thấy trên Docker Hub. Lưu ý rằng lệnh của bạn sẽ sử dụng namespace của bạn, không phải "docker".

```
docker push 1995mars/accounts
# push image đến repository [docker.io/1995mars/ account]
# Một image không tồn tại cục bộ với tag: 1995mars/ account
docker push 1995mars/cards
docker push 1995mars/loans
docker push 1995mars/configserver
```

9. Updating Docker Compose file to adapt Config Server changes

Thay đổi config trong: docker-compose.yml

```
version: "3.8"

services:

  configserver:
    image: eazybytes/configserver:latest
    mem_limit: 700m
    ports:
      - "8071:8071"
    networks:
      - eazybank
#Config của các microservice ...
```

10. Starting all the microservices using docker compose files based on the env

- Mở công cụ command line nơi chứa **docker-compose.yml** và chạy lệnh soạn thảo docker ***"docker-compose up"*** để khởi động tất cả các microservices container bằng một lệnh duy nhất.

11. Refreshing properties with @RefreshScope

Để cập nhật các thuộc tính cấu hình được load bởi Config Server trong ứng dụng Spring Boot mà không cần khởi động lại ứng dụng, bạn có thể sử dụng annotation `@RefreshScope`.

Đánh dấu các bean cần cập nhật lại cấu hình bằng annotation `@RefreshScope`.

```
@SpringBootApplication
@RefreshScope
@ComponentScans({@ComponentScan("com.eazybytes.accounts.controller")})
@EnableJpaRepositories("com.eazybytes.accounts.repository")
@EntityScan("com.eazybytes.accounts.model")
public class AccountsApplication {

    public static void main(String[] args) {
        SpringApplication.run(AccountsApplication.class, args);
    }

}
```

Ví dụ, nếu bạn có một bean MyService như sau:

```
@Service
@RefreshScope
public class MyService {
    @Value("${myapp.message}")
    private String message;

    public String getMessage() {
        return message;
    }
}
```

12. Encryption & Decryption of your properties inside Config server.

Nếu bạn muốn sử dụng một cách đơn giản hơn để mã hóa các thuộc tính trong Config Server, bạn có thể sử dụng cấu hình đơn giản hơn bằng việc chỉ định một giá trị cho thuộc tính `encrypt.key` trong file **application.properties** của Config Server.

```
encrypt.key=1995mars
```

Trong đó, **1995mars** là giá trị của khóa (key) dùng để mã hóa và giải mã các thuộc tính. Bạn có thể thay thế **1995mars** bằng bất kỳ giá trị nào bạn muốn sử dụng làm khóa.

Sau khi đã cấu hình `encrypt.key`, bạn có thể mã hóa các thuộc tính trong file cấu hình của Config Server bằng cách sử dụng prefix '{cipher}' như đã mô tả trong câu trả lời trước:

```
cards.msg
={cipher}c428e865ed8d16c86d273d0d9bb37cd300e68a76fa3c3b62c75aee4fb2e543e2ad6e
|eefb912a069535a674a626df207a56749d22cd076be5dd41fcac5be21e4a6b54ca99d2fc3531a
|d6d0e867b890cb3
```

Khi Config Server trả về các giá trị đã được mã hóa, Config Client (microservice) sẽ sử dụng giá trị của encrypt.key được cấu hình trên mình để giải mã các giá trị này.

Điều này giúp đơn giản hóa quá trình mã hóa và giải mã trong Config Server và Config Client, và bạn không cần phải tạo và quản lý keystore và khóa riêng. Tuy nhiên, hãy lưu ý rằng việc sử dụng giá trị cố định encrypt.key này có thể không an toàn nếu không được bảo vệ đúng cách. Trong môi trường thực tế, hãy đảm bảo giữ bí mật giá trị của encrypt.key và hãy cân nhắc sử dụng các cơ chế bảo mật phù hợp.

Lưu ý: trong product, bạn cần tạo một keystore (file chứa các khóa) và một khóa (key) để sử dụng cho mã hóa và giải mã. Bạn có thể sử dụng công cụ Java Keytool để tạo keystore và khóa. Điều này giúp bảo mật hơn.

Section 6: Service Discovery & Registration(Challenge 4)

1. Introduction to the Service Discovery & Registration inside microservices



HOW DO SERVICES LOCATE EACH OTHER INSIDE A NETWORK?

CÁC DỊCH VỤ ĐỊNH VỊ NHAU NHƯ THẾ NÀO BÊN TRONG MỘT MẠNG?

Mỗi phiên bản của một microservice hiển thị một API từ xa với máy chủ và cổng riêng của nó. Làm cách nào để các microservice và client khác biết về các dynamic endpoint URLs này để gọi chúng. Vậy dịch vụ của tôi ở đâu?



HOW DO NEW SERVICE INSTANCES ENTER INTO THE NETWORK?

CÁC TRƯỜNG HỢP DỊCH VỤ MỚI THAM GIA VÀO MẠNG NHƯ THẾ NÀO?

Nếu một microservice instance không thành công, các instance mới sẽ được đưa vào trực tuyến để đảm bảo tính khả dụng liên tục. Điều này có nghĩa là địa chỉ IP của các instance (phiên bản) có thể thay đổi liên tục. Vậy làm cách nào để những phiên bản mới này có thể bắt đầu phân phát cho khách hàng?



LOAD BALANCE, INFO SHARING B/W MICROSERVICE INSTANCES

Làm cách nào để chúng tôi đảm bảo cân bằng tải đúng cách với nhiều microservice instance, đặc biệt là một microservice đang gọi một microservice khác? Làm thế nào để một thông tin dịch vụ cụ thể được chia sẻ trên mạng?

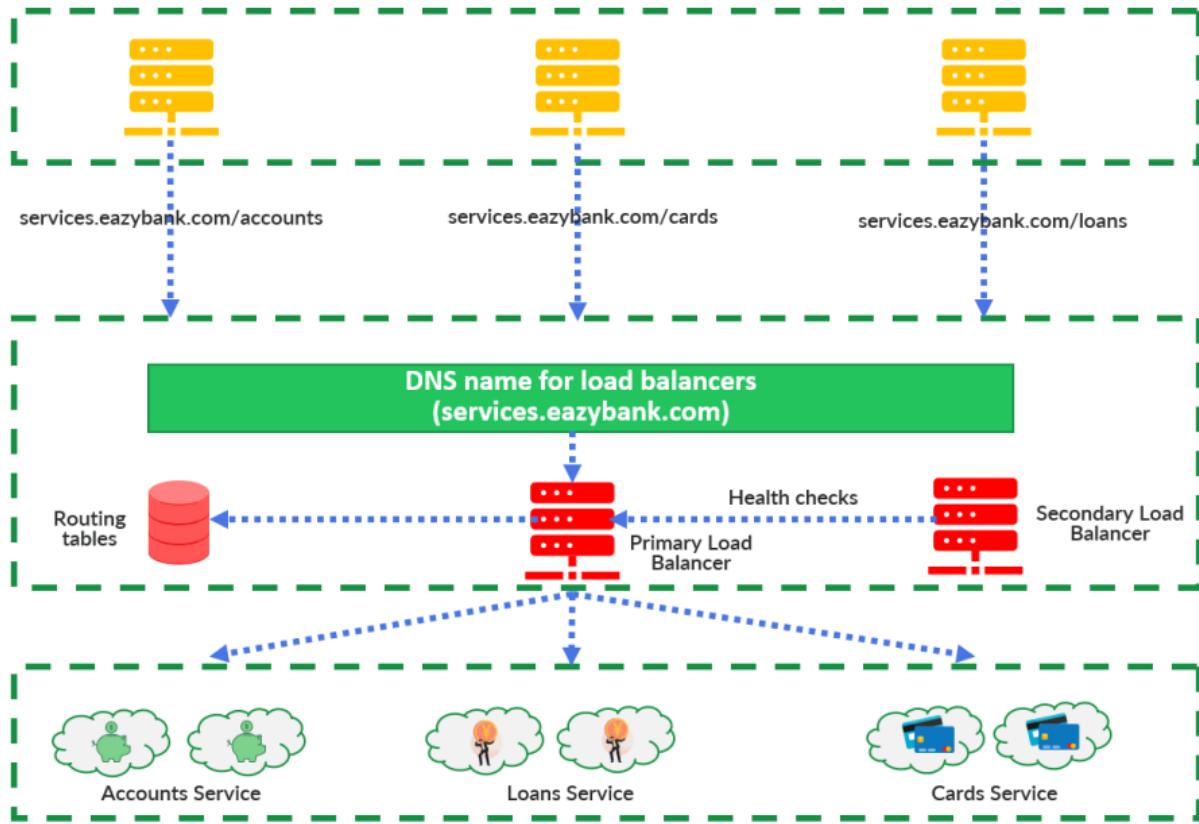


Service discovery & registration inside microservices network

- **Service discovery & registration** (Đăng ký và khám phá dịch vụ) giải quyết các vấn đề về cách các dịch vụ siêu nhỏ giao tiếp với nhau, tức là thực hiện các lệnh gọi API.
- Trong cấu trúc liên kết mạng truyền thống, các ứng dụng có vị trí mạng tĩnh. Do đó, địa chỉ IP của các vị trí bên ngoài có liên quan có thể được đọc từ tệp cấu hình vì những địa chỉ này hiếm khi thay đổi.
- Trong kiến trúc microservice hiện đại, việc biết đúng vị trí mạng của ứng dụng là một vấn đề phức tạp hơn nhiều đối với khách hàng vì các **service instance** (phiên bản dịch vụ) có thể có địa chỉ IP được gán động. Ngoài ra, số **instance** có thể thay đổi do autoscaling và failures..
- Microservices service discovery & registration là một cách để các ứng dụng và microservice xác định vị trí của nhau trên mạng. Điều này bao gồm:
 - ✓ Một máy chủ trung tâm (hoặc các máy chủ) duy trì chế độ xem địa chỉ toàn cầu.
 - ✓ Microservices/client kết nối với máy chủ trung tâm để đăng ký địa chỉ của chúng khi chúng bắt đầu và sẵn sàng
 - ✓ Microservices/client cần gửi heartbeats (nhịp tim) của họ đều đặn đến máy chủ trung tâm về tình trạng của họ

2. Why not traditional load balancers for Microservices (Tại sao không phải là cân bằng tải truyền thống cho Microservices)

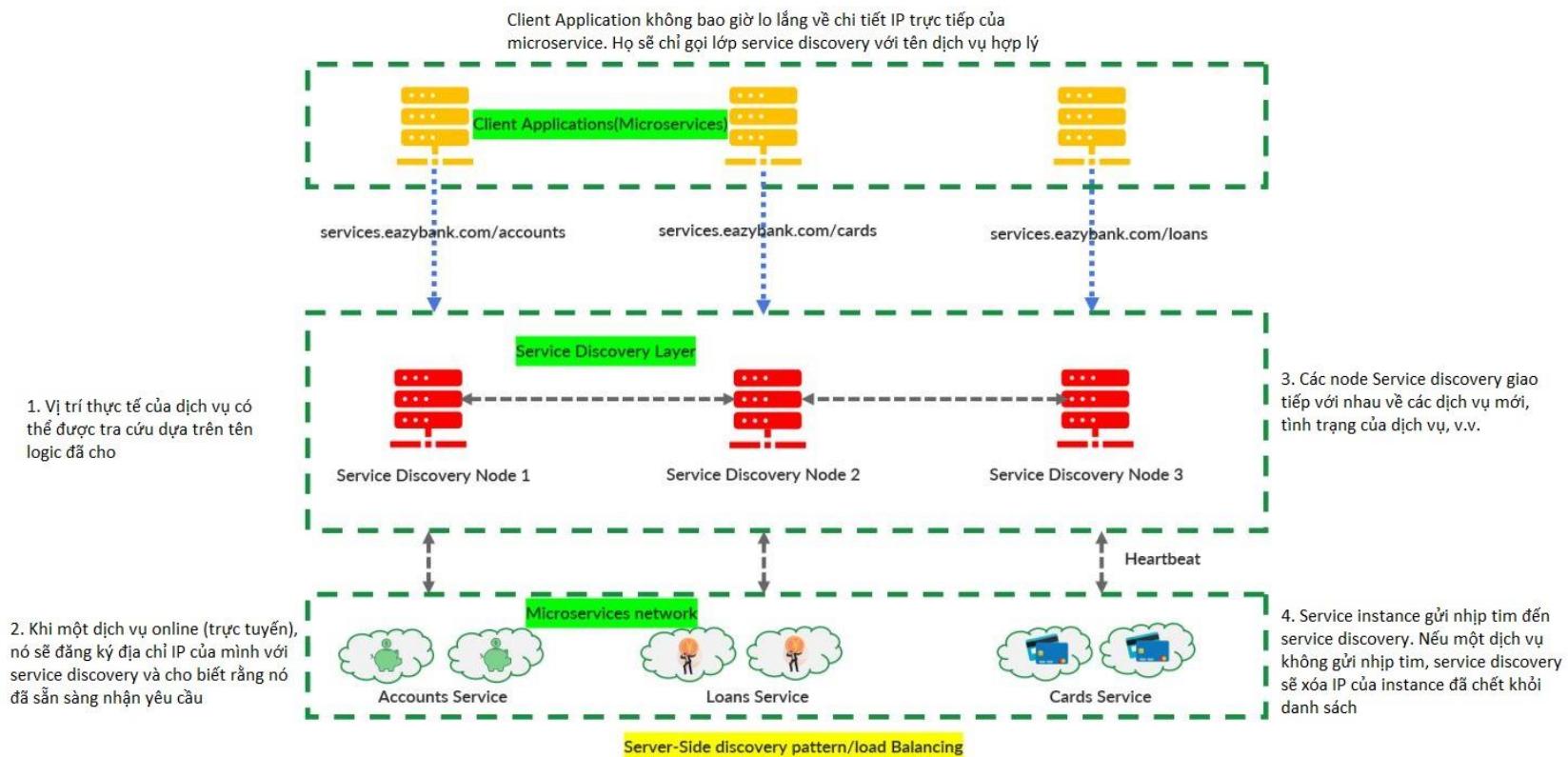
Các ứng dụng như giao diện người dùng hoặc các dịch vụ khác sử dụng DNS chung cùng với đường dẫn cụ thể của dịch vụ để gọi một dịch vụ cụ thể.



Kiến trúc **traditional load balancer** sử dụng DNS và bộ cân bằng tải

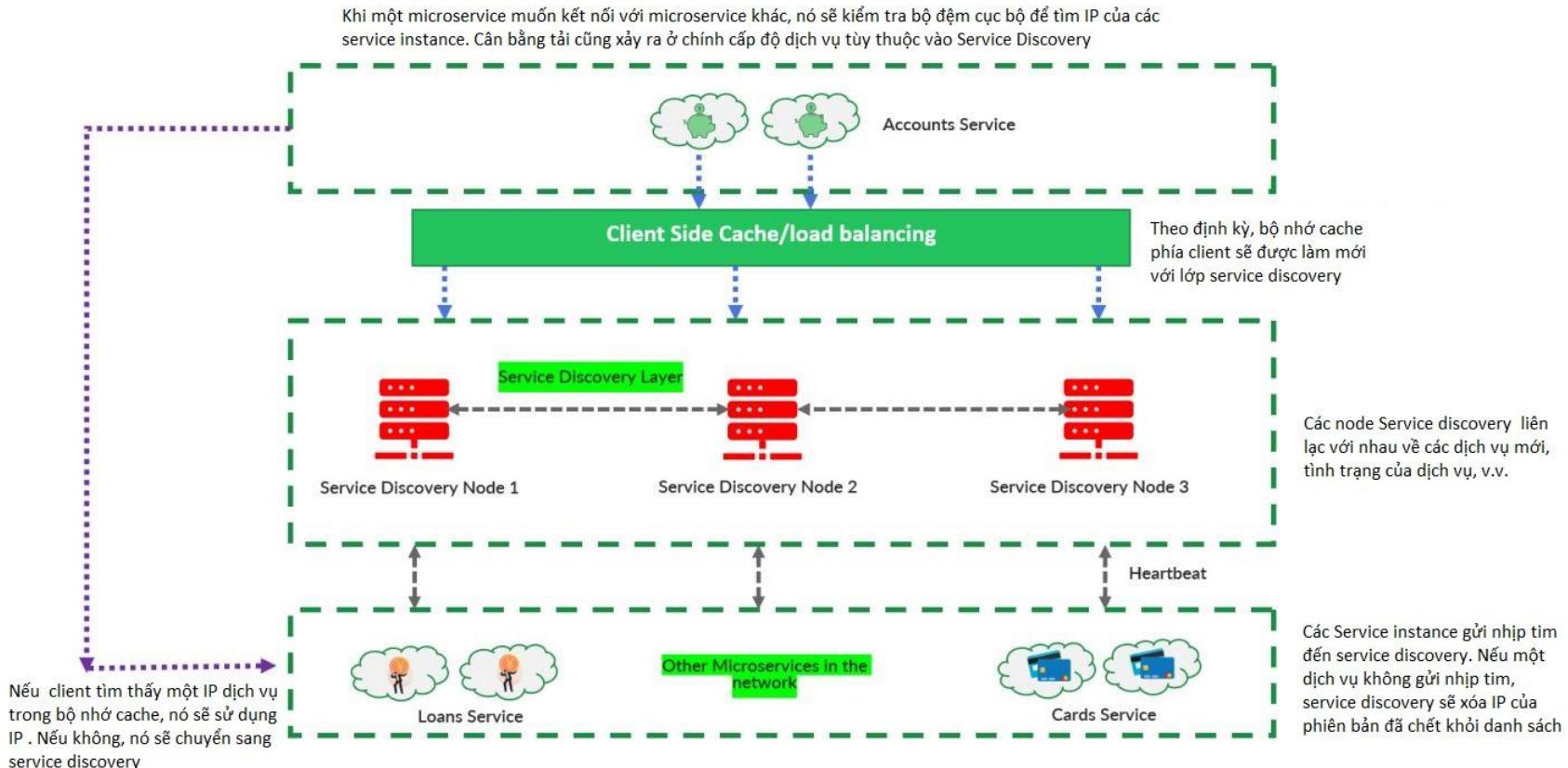
- Với cách tiếp cận truyền thống, từng instance của một dịch vụ thường được triển khai trong một hoặc nhiều máy chủ ứng dụng. Số lượng các máy chủ ứng dụng này thường là cố định và ngay cả trong trường hợp khôi phục, nó sẽ được khôi phục về trạng thái cũ với cùng một IP và các cấu hình khác.
- Mặc dù loại mô hình này hoạt động tốt với các ứng dụng dựa trên SOA và nguyên khối với số lượng dịch vụ tương đối nhỏ chạy trên một nhóm máy chủ tĩnh, nhưng nó không hoạt động tốt đối với các ứng dụng microservice trên cloud vì những lý do sau:
 - ✓ Khả năng mở rộng theo chiều ngang và chi phí licenses hạn chế
 - ✓ Điểm lỗi duy nhất & Điểm nghẽn tập trung
 - ✓ Quản lý thủ công để cập nhật mọi IP, cấu hình
 - ✓ Không thân thiện với container
 - ✓ Bản chất phức tạp

3. Architecture of Service Discovery inside microservices



- Các công cụ và mẫu **service discovery** được phát triển để vượt qua những thách thức với bộ cân bằng tải truyền thống.
- Service discovery bao gồm kho lưu trữ khóa-giá trị (Đăng ký dịch vụ) và API để đọc và ghi vào kho lưu trữ này. Các instances mới của ứng dụng được lưu vào sổ đăng ký dịch vụ này và bị xóa khi dịch vụ ngừng hoạt động hoặc không hoạt động tốt.
- Khách hàng muốn liên lạc với một dịch vụ nhất định phải tương tác với service registry để biết (các) vị trí mạng chính xác.
- Ưu điểm của phương pháp Khám phá dịch vụ,
 - ✓ Không giới hạn về tính khả dụng
 - ✓ Giao tiếp ngang hàng Service Discovery
 - ✓ IPs, configurations & Load balanced được quản lý động
 - ✓ Khả năng chịu lỗi & Khả năng phục hồi tự nhiên

4. Client Side load balancing between microservices (Cân bằng tải phía Client Side giữa các microservices)



5. Spring Cloud support for Service Discovery & Registration

- Dự án Spring Cloud làm cho việc thiết lập Service Discovery & Registration trở nên đơn giản để thực hiện với sự trợ giúp của các thành phần bên dưới:
 - ✓ **Spring Cloud Netflix's Eureka service** sẽ hoạt động như một tác nhân **service discovery**
 - ✓ **Spring Cloud Load Balancer library** để cân bằng tải phía client-side
 - ✓ Netflix Feign client để tìm kiếm dịch vụ microservice

* Mặc dù trong khóa học này, sử dụng Eureka vì nó được sử dụng chủ yếu nhưng có các service registry khác như etcd, Consul và Apache Zookeeper cũng tốt.

** Mặc dù Netflix Ribbon client-side cũng là một sản phẩm tốt và ổn định, nhưng sẽ sử dụng Spring Cloud Load Balancer để cân bằng tải phía máy khách. Điều này là do Ribbon đã chuyển sang chế độ bảo trì và rất tiếc, nó sẽ không được phát triển nữa.

6. Setup Service Discovery agent using Eureka server

Để cài đặt Service Discovery Agent bằng Eureka Server, bạn cần thực hiện các bước sau:

Bước 1: Tạo project

- Truy cập <https://start.spring.io/>
- Điền tất cả các chi tiết cần thiết để tạo dự án Spring Boot eurekaserver và thêm các phụ thuộc Eureka Server, Spring Boot Actuator, Config Client. Nhấp vào **GENERATE** sẽ tải xuống dự án maven eurekaserver ở định dạng zip
- Trích xuất dự án maven đã tải xuống của eurekaserver.

Bước 2: Thêm phụ thuộc Maven cho Eureka Server vào tệp pom.xml của dự án Spring Boot của bạn

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

Thêm chi tiết plugin spring-boot-maven-plugin cùng với chi tiết tên docker image bên trong. Chi tiết bổ sung về spring-boot-maven-plugin này sẽ giúp tạo docker image bằng Buildpacks một cách dễ dàng.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <image>
                    <name>eazybytes/${project.artifactId}</name>
                </image>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Bước 3: Tạo main class cho Eureka Server với chú thích `@EnableEurekaServer` để bật Eureka Server:

Mở SpringBoot main class EurekaserverApplication.java Trên main class này, vui lòng thêm chú thích '`@EnableEurekaServer`'. Chú thích này sẽ làm cho microservic của bạn hoạt động như một Spring Cloud Netflix Eureka Server. Sau khi thực hiện các thay đổi, lớp EurekaserverApplication.java của bạn sẽ như bên dưới:

```
@SpringBootApplication  
@EnableEurekaServer  
public class EurekaserverApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(EurekaserverApplication.class, args);  
    }  
  
}
```

Bước 4: Cấu hình Eureka Server

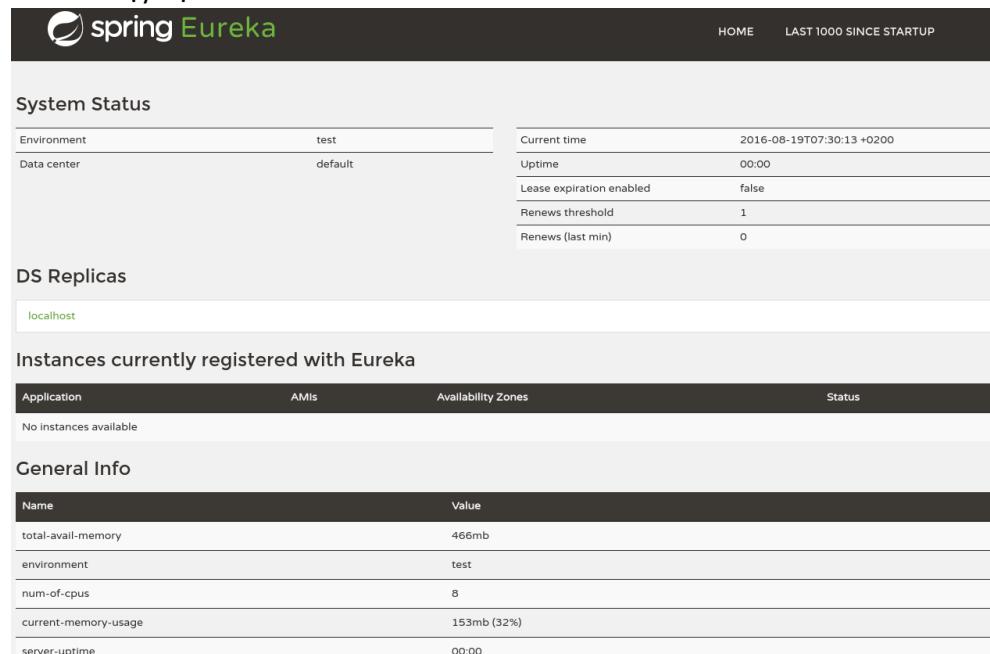
Tạo tệp application.properties hoặc application.yml trong thư mục resources của dự án và cấu hình các thuộc tính Eureka Server:

```
spring.application.name=eurekaserver  
spring.config.import=optional:configserver:http://localhost:8071/
```

Vui lòng đảm bảo tạo một eurekaserver.properties với nội dung bên dưới bên trong vị trí mà Config Server đang đọc các thuộc tính.

```
server.port=8070  
eureka.instance.hostname=localhost  
eureka.client.registerWithEureka=false  
eureka.client.fetchRegistry=false  
eureka.client.serviceUrl.defaultZone=http://${eureka.instance.hostname}:${server.port}/eureka/
```

Sau đó chạy dự án Eureka:



The screenshot shows the Eureka Server dashboard with the following sections:

- System Status:** Displays environment (test), data center (default), current time (2016-08-19T07:30:13 +0200), uptime (00:00), lease expiration enabled (false), renew threshold (1), and renew count (0).
- DS Replicas:** Shows a single instance at localhost.
- Instances currently registered with Eureka:** A table with columns Application, AMIs, Availability Zones, and Status. It shows "No instances available".
- General Info:** A table with columns Name and Value, listing system metrics like total-available-memory (466mb), environment (test), number of cpus (8), current memory usage (153mb (32%)), and server uptime (00:00).

Bước 5: Cài đặt Eureka Client

Thêm phụ thuộc Maven cho Eureka Client vào tệp pom.xml của dự án Spring Boot của bạn (các microservice như accounts, cads, loans .v.v)

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

Mở application.properties bên trong accounts microservices và thêm các mục bên dưới vào bên trong nó sẽ giúp tích hợp với eurekaserver

```
eureka.instance.preferIpAddress = true
eureka.client.registerWithEureka = true
eureka.client.fetchRegistry = true
eureka.client.serviceUrl.defaultZone = http://localhost:8070/eureka/

## Configuring info endpoint
info.app.name=Accounts Microservice
info.app.description=Eazy Bank Accounts Application
info.app.version=1.0.0
management.info.env.enabled = true

endpoints.shutdown.enabled=true
management.endpoint.shutdown.enabled=true
```

Bước 6: Tạo lớp chính cho Eureka Client và đánh dấu nó bằng @EnableDiscoveryClient.
@EnableDiscoveryClient không còn cần thiết nữa. Sau đó chạy project.

The screenshot shows the Spring Eureka dashboard. At the top, it says "HOME LAST 1000 SINCE STARTUP". The "System Status" section shows environment and data center details. The "DS Replicas" section lists an instance for the "ACCOUNTS" application. The "General Info" section provides memory usage statistics.

Name	Value
total-avail-memory	77mb
num-of-cpus	8
current-memory-usage	53mb (68%)

7. Feign Client to invoke other microservices

Feign Client là một thư viện được cung cấp bởi Netflix trong các ứng dụng sử dụng Spring Cloud để xây dựng các RESTful client một cách dễ dàng và tiện lợi. Thay vì phải viết mã HTTP client một cách thủ công, Feign giúp tự động tạo ra các proxy dựa trên các interface được định nghĩa trước để gọi các API từ các microservices khác.

Bạn có thể sử dụng Feign để giao tiếp giữa các microservices trong một kiến trúc dựa trên microservices. Feign là một thư viện trong Spring Cloud giúp tạo các RESTful client một cách dễ dàng, tiện lợi và hỗ trợ giao tiếp giữa các microservice.

Khi bạn triển khai kiến trúc microservices, các dịch vụ khác nhau thường cần giao tiếp và tương tác với nhau để hoàn thành một số tác vụ. Feign giúp bạn giảm độ phức tạp của việc gọi API giữa các microservice bằng cách tạo ra các proxy dựa trên các interface định nghĩa trước. Các proxy này tự động xử lý việc gọi các API từ các microservice khác, bạn không cần phải viết mã HTTP client một cách thủ công.

Một số lợi ích của việc sử dụng Feign trong việc giao tiếp giữa các microservices là:

- Đơn giản hóa giao tiếp: Feign giúp bạn tạo các proxy cho các microservices mục tiêu mà bạn muốn giao tiếp, giúp giảm độ phức tạp của việc tạo và quản lý các HTTP client.
- Tiết kiệm thời gian và công sức: Feign giúp bạn tự động tạo các cuộc gọi API, giúp tiết kiệm thời gian và công sức trong việc viết và bảo trì mã gọi API.
- Load balancing: Feign tích hợp với các giải pháp load balancing như Ribbon, giúp phân phối các cuộc gọi API đến nhiều phiên bản của cùng một dịch vụ để cải thiện độ tin cậy và khả năng mở rộng của hệ thống.
- Xử lý lỗi dễ dàng: Feign cung cấp cơ chế xử lý lỗi và đào tạo giúp bạn xử lý các trường hợp lỗi khi gọi các microservices không thành công.

Để thực hiện Feign Client để gọi các microservices khác trong một ứng dụng Spring Boot, bạn cần làm theo các bước sau:

Bước 1: Thêm Feign dependency

Thêm thư viện Feign vào file pom.xml của dự án:

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

Bước 2: Định nghĩa Feign Client Interface

Tạo một interface để định nghĩa các phương thức gọi API tới các microservices khác. Interface này sẽ được Spring Boot biên dịch thành một proxy để thực hiện các cuộc gọi API.

\accounts\src\main\java\com\easybytes\accounts\service\client\LoansFeignClient.java

```
@FeignClient("loans")
public interface LoansFeignClient {
    @RequestMapping(method = RequestMethod.POST, value = "myLoans", consumes
= "application/json")
    List<Loans> getLoansDetails(@RequestBody Customer customer);
}
```

Annotation @RequestMapping đánh dấu phương thức getLoansDetails sẽ được gọi khi có một cuộc gọi POST đến đường dẫn "/myLoans" từ microservice "loans". Điều này đồng nghĩa với việc phương thức getLoansDetails trong Feign Client Interface sẽ gọi endpoint "/myLoans" của microservice "loans" thông qua phương thức POST.

- method = RequestMethod.POST: Xác định phương thức HTTP sẽ được sử dụng khi gọi API (POST trong trường hợp này).
- value = "myLoans": Xác định đường dẫn của endpoint mục tiêu trên microservice "loans".
- consumes = "application/json": Xác định loại dữ liệu được gửi đi trong request body, trong trường hợp này là JSON.

Lưu ý value = "myLoans" là controller trong loans microservice

```
@PostMapping("/myLoans")
public List<Loans> getLoansDetails(@RequestBody Customer customer) {
    List<Loans> loans =
loansRepository.findByCustomerIdOrderByStartDtDesc(customer.getCustomerId());
    if (loans != null) {
        return loans;
    } else {
        return null;
    }
}
```

Bước 3: Sử dụng interface đã tạo

Để tìm nạp thông tin chi tiết về khoản vay bằng Feign client từ microservice, hãy cập nhật AccountsController.java để hiển thị API REST mới /myCustomerDetails

```
@RestController
public class AccountsController {

    @Autowired
    private AccountsRepository accountsRepository;

    @Autowired
    AccountsServiceConfig accountsConfig;

    @Autowired
    LoansFeignClient loansFeignClient;
```

```

    @PostMapping("/myCustomerDetails")
    public CustomerDetails myCustomerDetails(@RequestBody Customer customer)
    {
        Accounts accounts =
accountsRepository.findByCustomerId(customer.getCustomerId());
        List<Loans> loans = loansFeignClient.getLoansDetails(customer);

        CustomerDetails customerDetails = new CustomerDetails();
        customerDetails.setAccounts(accounts);
        customerDetails.setCards(cards);

        return customerDetails;
    }
}

```

8. Generating Docker images after Service Discovery changes

Trong file pom.xml

```

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <image>
                    <name>eazybytes/${project.artifactId}</name>
                </image>
            </configuration>
        </plugin>
    </plugins>
</build>

```

Chạy lệnh command:

```
mvn spring-boot:build-image
```

Lệnh mvn spring-boot:build-image được sử dụng để build một Docker image từ một ứng dụng Spring Boot bằng cách sử dụng plugin "spring-boot-maven-plugin".

9. Pushing all the latest Docker images with Eureka changes to Docker Hub

Trong command line, hãy thử chạy lệnh push mà bạn thấy trên Docker Hub. Lưu ý rằng lệnh của bạn sẽ sử dụng namespace của bạn, không phải "docker".

```

docker push 1995mars/accounts
docker push 1995mars/cards
docker push 1995mars/loans
docker push 1995mars/configserver
docker push 1995mars/eurekaserver

```

10. Updating Docker Compose file to adapt Service Discovery changes

Thay đổi config trong: docker-compose.yml

```
version: "3.8"

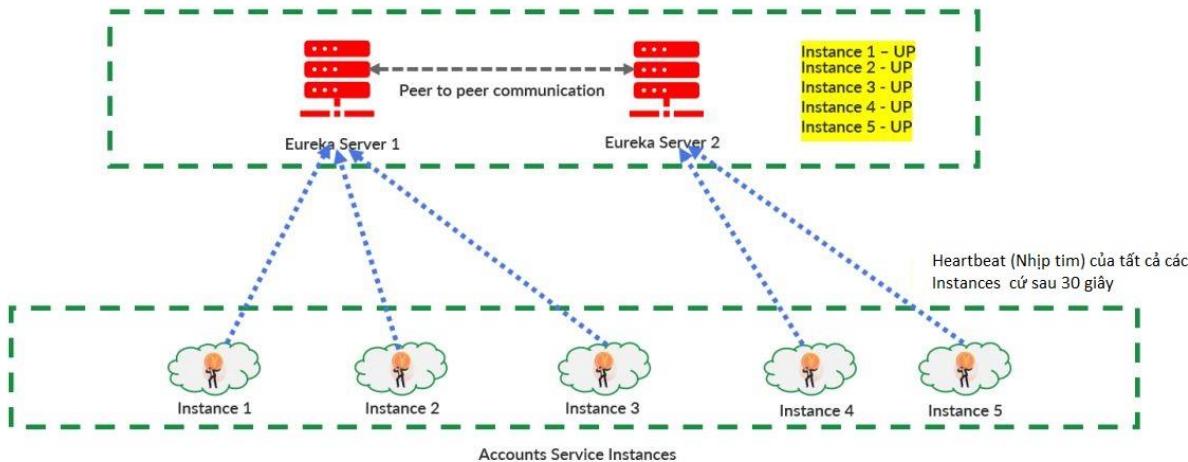
services:

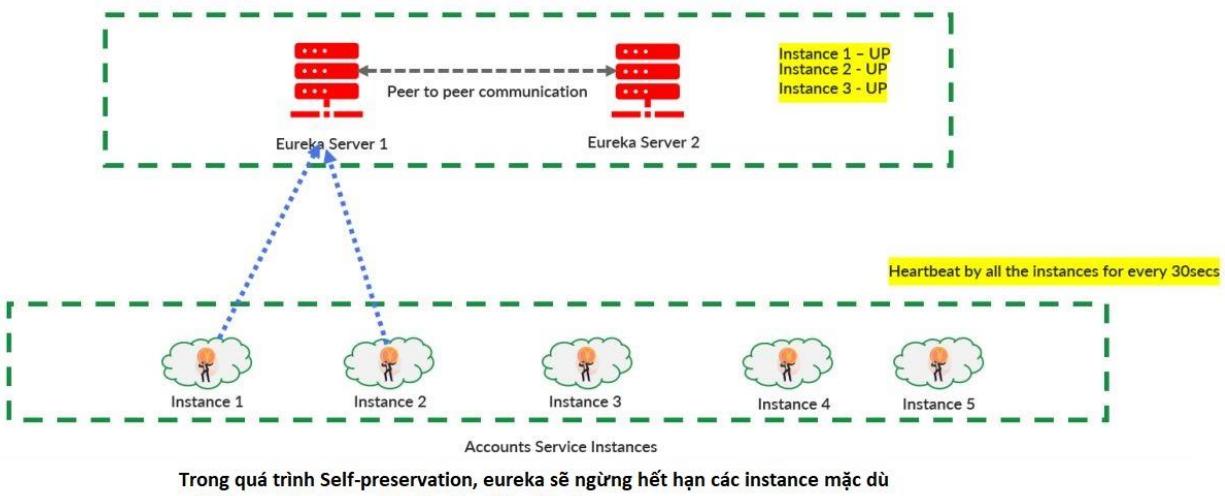
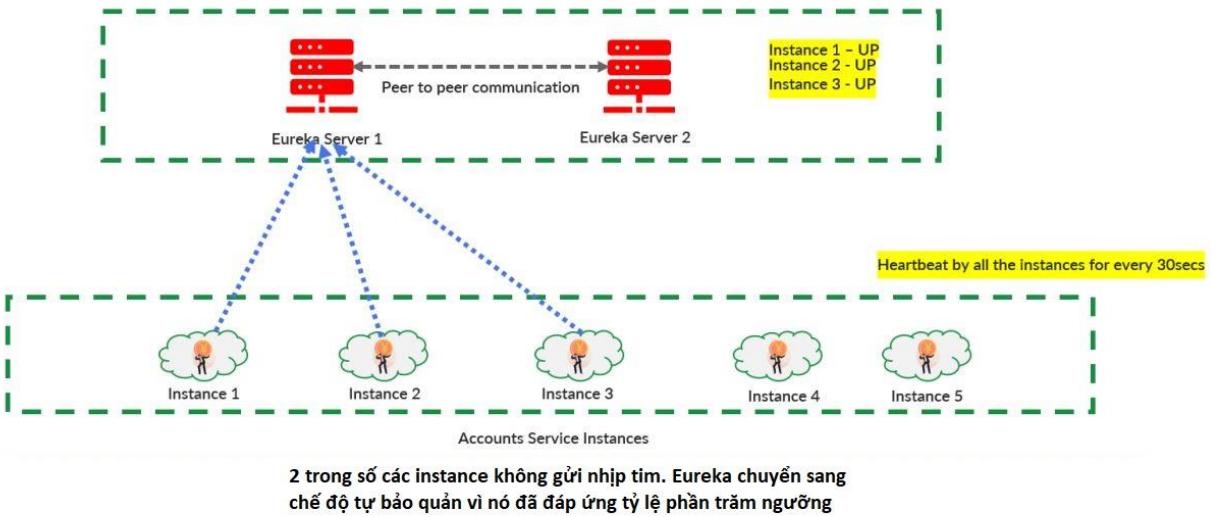
  configserver:
    image: eazybytes/configserver:latest
    mem_limit: 700m
    ports:
      - "8071:8071"
    networks:
      - eazybank
#Config của các microservice ...
```

11. Starting all the microservices using docker compose files based on the env

- Mở công cụ command line nơi chứa **docker-compose.yml** và chạy lệnh soạn thảo docker ***"docker-compose up"*** để khởi động tất cả các microservices container bằng một lệnh duy nhất.

12. Eureka Self-Preservation mode to avoid network trap issues (Chế độ tự bảo vệ Eureka để tránh các sự cố bẫy mạng)





- Lý do đằng sau chế độ self-preservation (tự bảo quản) trong eureka
 - Máy chủ không nhận được nhịp tim có thể do sự cố mạng kém nhưng không nhất thiết có nghĩa là clients ngừng hoạt động, điều này có thể được giải quyết sớm hơn. Vì vậy, nếu không có khả năng tự bảo quản, cuối cùng chúng ta sẽ không có instance nào với Eureka mặc dù các instance đó có thể đang hoạt động.
 - Mặc dù kết nối bị mất giữa các máy chủ và một số client, các client vẫn có thể có kết nối với nhau. Với chi tiết đăng ký bộ đệm cục bộ, họ có thể tiếp tục liên lạc với nhau
- Chế độ tự bảo quản không bao giờ hết hạn, cho đến khi và trừ khi các vi dịch vụ ngừng hoạt động được khôi phục hoặc trực trặc mạng được giải quyết. Điều này là do eureka sẽ không hết hạn các instance cho đến khi vượt quá giới hạn ngưỡng.
- Tính năng tự bảo quản sẽ là vị cứu tinh khi mạng trực trặc phổ biến và giúp chúng tôi xử lý các cảnh báo giả.

Có một số cấu hình trong Eureka Server sẽ tác động trực tiếp hoặc gián tiếp đến hành vi tự bảo vệ (self-preservation) của Eureka. Dưới đây là một số cấu hình quan trọng mà bạn nên quan tâm để điều chỉnh hành vi tự bảo vệ của Eureka:

eureka.server.enable-self-preservation = true

Đây là cấu hình chính để kích hoạt hoặc tắt chế độ tự bảo vệ. Nếu bạn muốn sử dụng tính năng tự bảo vệ của Eureka, hãy đảm bảo rằng giá trị của thuộc tính này là true. Nếu bạn muốn vô hiệu hóa tính năng tự bảo vệ, bạn có thể đặt giá trị là false.

eureka.server.renewal-percent-threshold=0.85

Đây là cấu hình để xác định ngưỡng phần trăm hồi sinh (renewal) mà Eureka sẽ sử dụng để xác định xem có tự bảo vệ hay không. Nếu tỷ lệ phần trăm các microservice hồi sinh (đăng ký lại) so với tổng số microservice đăng ký ban đầu dưới ngưỡng này, Eureka sẽ kích hoạt chế độ tự bảo vệ. Mục tiêu là giữ lại thông tin đăng ký của các microservice khi tỷ lệ các microservice hồi sinh thấp.

eureka.server.eviction-interval-timer-in-ms=60*1000

Đây là cấu hình để định nghĩa thời gian kiểm tra và xóa các thông tin đăng ký của các microservice không gửi heartbeat. Nếu một microservice không gửi heartbeat trong khoảng thời gian này, Eureka sẽ xem như nó không còn hoạt động và có thể xóa nó ra khỏi danh sách đăng ký. Tuy nhiên, trong chế độ tự bảo vệ, Eureka sẽ giữ lại các thông tin đăng ký này trong một khoảng thời gian nhất định để tránh xóa nhầm các microservice thực sự hoạt động nhưng tạm thời mất kết nối.

eureka.instance.lease-renewal-interval-in-second=30

Cho biết tần suất client gửi nhịp tim đến máy chủ để cho biết máy vẫn còn hoạt động.

eureka.instance.lease-expiration-duration-in-second=90

Cho biết khoảng thời gian máy chủ đợi kể từ khi nhận được nhịp tim cuối cùng trước khi có thể trục xuất một instance

eureka.server.renewal- threshold-update-interv-a-ms=15*60*1000

Một bộ lập lịch được chạy ở tần suất này để tính toán nhịp tim dự kiến mỗi phút

Section 7: Making Microservices Resilient (Challenge 5)

1. Introduction to the need of Resiliency inside microservices (Giới thiệu về sự cần thiết của *Khả năng phục hồi* trong microservice)



HOW DO WE AVOID CASCADING FAILURES?

Làm thế nào để chúng tôi tránh được các thất bại xếp tầng(**cascading failures**)?
Cascading failures là tình trạng một service trong hệ thống của bạn bị lỗi dẫn đến các services khác bị lỗi theo, thậm chí có thể dẫn đến cả hệ thống bị down. Giống như trong các tình huống có nhiều microservice đang giao tiếp với nhau, chúng ta cần đảm bảo rằng toàn bộ chuỗi microservice không bị lỗi do lỗi của một microservice duy nhất



HOW DO WE HANDLE FAILURES GRACEFULLY WITH FALLBACKS?

Làm thế nào để chúng tôi xử lý lỗi một cách tốt bằng dự phòng(**fallbacks**)?
Trong một chuỗi nhiều **microservice**, làm cách nào để xây dựng cơ chế dự phòng nếu một trong các microservice không hoạt động. Giống như trả về giá trị mặc định hoặc trả về giá trị từ bộ đệm hoặc gọi dịch vụ/DB khác để tìm nạp kết quả, v.v.



HOW TO MAKE OUR SERVICES SELF-HEALING CAPABLE

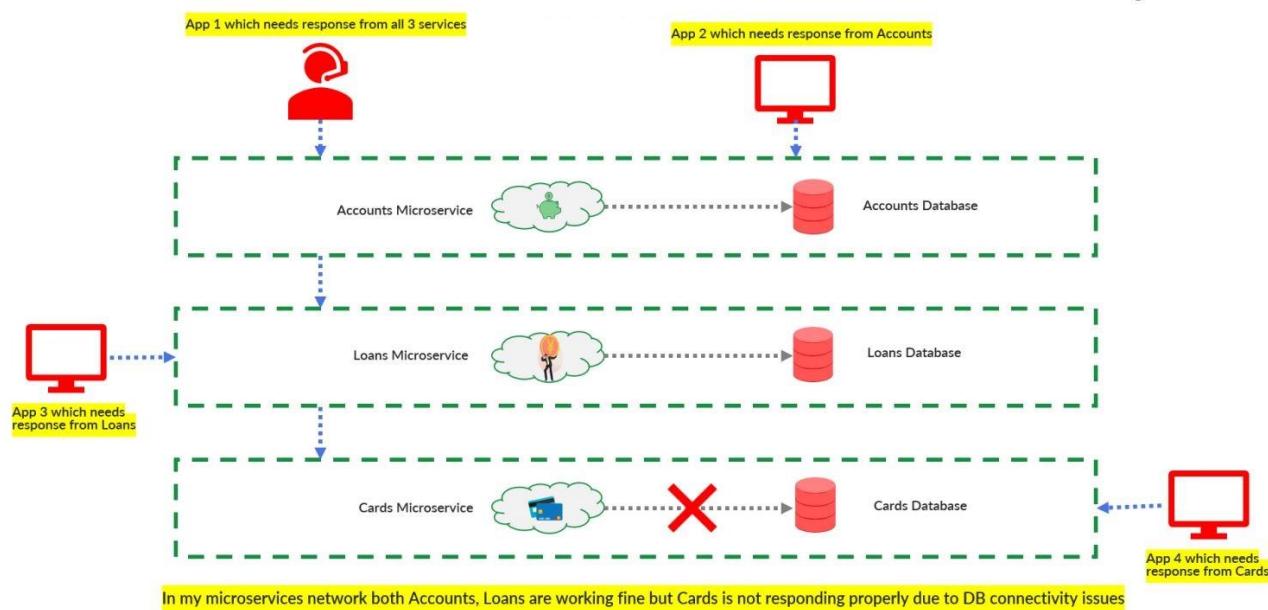
Cách làm cho các dịch vụ của chúng tôi có **khả năng tự phục hồi**
Trong trường hợp dịch vụ hoạt động chậm, làm cách nào để định cấu hình thời gian chờ, thử lại và dành thời gian để dịch vụ bị lỗi tự phục hồi.



Spring hỗ trợ cho khả năng phục hồi bằng Resilience4j

- Resilience4j là một thư viện nhẹ, dễ sử dụng lấy cảm hứng từ Netflix Hystrix, nhưng được thiết kế cho Java 8 và lập trình chức năng. Nhẹ, vì thư viện chỉ sử dụng Vavr, không có bất kỳ phụ thuộc thư viện bên ngoài nào khác. Ngược lại, Netflix Hystrix có phần phụ thuộc biên dịch vào Archaius có nhiều phần phụ thuộc thư viện bên ngoài hơn như Guava và Apache Commons Configuration..
- Resilience4j cung cấp các mẫu sau để tăng khả năng chịu lỗi do sự cố mạng hoặc lỗi của bất kỳ dịch vụ nào trong số nhiều dịch vụ:
 - ✓ **Circuit breaker - Bộ ngắt mạch:** Được sử dụng để dừng thực hiện yêu cầu khi dịch vụ được gọi không thành công.
 - ✓ **Fallback - Dự phòng:** Đường dẫn thay thế cho các yêu cầu không thành công.
 - ✓ **Retry - Thủ lại:** Được sử dụng để thử lại khi một dịch vụ tạm thời bị lỗi.
 - ✓ **Rate limit - Giới hạn tốc độ:** Giới hạn số lượng cuộc gọi mà một dịch vụ nhận được trong một thời gian.
 - ✓ **Bulkhead - Vách ngăn:** Giới hạn số lượng yêu cầu gửi đi đồng thời cho một dịch vụ để tránh quá tải.
- Trước khi sử dụng Resilience4j, Nhà phát triển đã từng sử dụng Hystrix, một trong những thư viện java phổ biến nhất để triển khai các mẫu khả năng phục hồi trong vi dịch vụ. Nhưng hiện tại Hystrix đang ở chế độ bảo trì và không có tính năng mới nào được phát triển. Vì lý do này, bây giờ mọi người đều sử dụng Resilience4j có nhiều tính năng hơn Hystrix.

2. Typical use case or scenario for the need of Resiliency- Trường hợp hoặc kịch bản sử dụng điển hình cho nhu cầu về Resiliency – Tự phục hồi

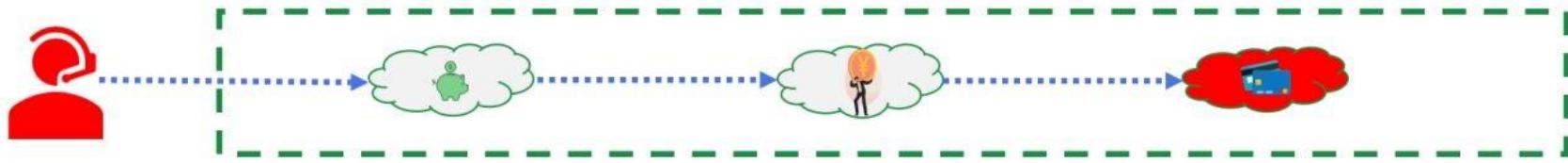


3. Deep dive on Circuit Breaker pattern in microservices

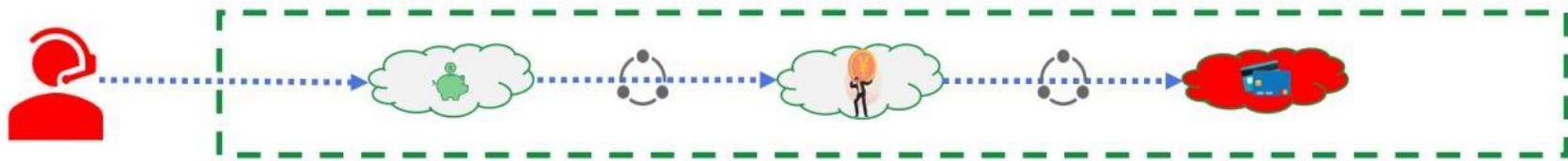
- Trong môi trường phân tán, các cuộc gọi đến tài nguyên và dịch vụ từ xa có thể không thành công do các lỗi nhất thời, chẳng hạn như kết nối mạng chậm, hết thời gian chờ hoặc tài nguyên bị quá tải hoặc tạm thời không khả dụng. Những lỗi này thường tự khắc phục sau một khoảng thời gian ngắn và bạn nên chuẩn bị một ứng dụng đâm mê mạnh mẽ để xử lý chúng.
- Circuit Breaker pattern** lấy cảm hứng từ bộ ngắt mạch điện sẽ giám sát các cuộc gọi từ xa. Nếu các cuộc gọi diễn ra quá lâu, bộ ngắt mạch sẽ can thiệp và hủy cuộc gọi. Ngoài ra, **circuit breaker** - bộ ngắt mạch sẽ giám sát tất cả các cuộc gọi đến một tài nguyên từ xa và nếu đủ các cuộc gọi không thành công, việc triển khai ngắt mạch sẽ bật lên, không thành công nhanh chóng và ngăn các cuộc gọi trong tương lai đến tài nguyên từ xa bị lỗi.
- Circuit Breaker** cũng cho phép ứng dụng phát hiện xem lỗi đã được giải quyết chưa. Nếu sự cố dường như đã được khắc phục, ứng dụng có thể cố gắng thực hiện thao tác.
- Ưu điểm của circuit breaker pattern là
 - ✓ Thất bại nhanh chóng
 - ✓ thất bại một cách tuyệt vời
 - ✓ Phục hồi liền mạch

Trong Resilience4j, bộ ngắt mạch được triển khai thông qua
máy trạng thái hữu hạn(finite state machine) với các trạng thái sau

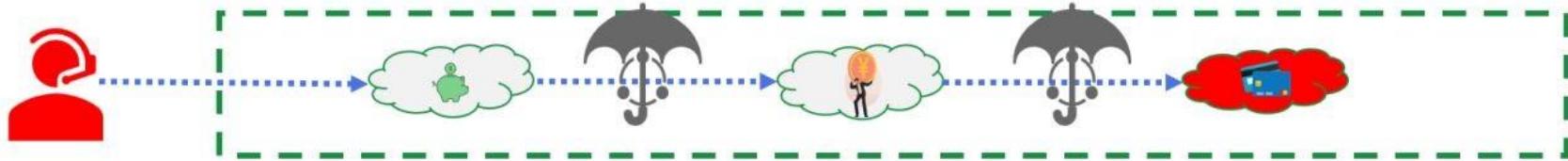




Tình huống 1 - Nếu Cards microservice đang phản hồi chậm, thì khi không có bộ ngắt mạch, microservice này sẽ bắt đầu ngốn hết tất cả các chuỗi tài nguyên trên Loans và Accounts microservice, điều này cuối cùng sẽ khiến chúng cũng chậm lại/sập xuống



Tình huống 2 - Nếu Cards microservice đang phản hồi chậm, thì với các bộ ngắt mạch ở giữa, nó sẽ bắt đầu hoạt động và làm hỏng service nhanh chóng với các trạng thái OPEN, HALF_OPEN, CLOSED. Bằng cách này, ít nhất các Accounts and Loans services sẽ không gặp bất kỳ sự cố nào đối với các Ứng dụng khác.



Tình huống 3 - Nếu Cards microservice đang phản hồi chậm thì với bộ ngắt mạch và cơ chế dự phòng, chúng ta có thể chắc chắn rằng ít nhất chúng ta đang bị lỗi một cách nhẹ nhàng với một số phản hồi mặc định đang được trả về, đồng thời các dịch vụ khác sẽ không bị ảnh hưởng

4. Implementing Circuit Breaker pattern

Để triển khai Resilience4j trong Spring Boot, làm theo các bước sau:

Bước 1: Thêm dependency trong file pom.xml:

```
<dependency>
    <groupId>io.github.resilience4j</groupId>
    <artifactId>resilience4j-spring-boot2</artifactId>
</dependency>
<dependency>
    <groupId>io.github.resilience4j</groupId>
    <artifactId>resilience4j-circuitbreaker</artifactId>
</dependency>
<dependency>
    <groupId>io.github.resilience4j</groupId>
    <artifactId>resilience4j-timelimiter</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

Bước 2 Đánh dấu các method cần bảo vệ bằng @CircuitBreaker:

```
@PostMapping("/myCustomerDetails")
@CircuitBreaker(name =
"detailsForCustomerSupportApp", fallbackMethod="myCustomerDetailsFallBack")
public CustomerDetails myCustomerDetails(@RequestBody Customer customer) {
    Accounts accounts =
accountsRepository.findByCustomerId(customer.getCustomerId());
    List<Loans> loans = loansFeignClient.getLoansDetails(customer);
    List<Cards> cards = cardsFeignClient.getCardDetails(customer);

    CustomerDetails customerDetails = new CustomerDetails();
    customerDetails.setAccounts(accounts);
    customerDetails.setLoans(loans);
    customerDetails.setCards(cards);

    return customerDetails;
}

private CustomerDetails myCustomerDetailsFallBack(Customer customer,
Throwable t) {
    Accounts accounts =
accountsRepository.findByCustomerId(customer.getCustomerId());
    List<Loans> loans = loansFeignClient.getLoansDetails(customer);
    CustomerDetails customerDetails = new CustomerDetails();
    customerDetails.setAccounts(accounts);
    customerDetails.setLoans(loans);
    return customerDetails;
}
```

- **@CircuitBreaker:** Đây là một annotation của Resilience4j, được sử dụng để triển khai mô hình Circuit Breaker pattern cho phương thức myCustomerDetails. Nó giúp bảo vệ ứng dụng khỏi những tình huống lỗi từ các tác vụ ngoài hệ thống, như việc gọi các API từ xa. Nếu số lượng lỗi vượt quá một ngưỡng được cấu hình, mô hình Circuit Breaker sẽ mở, và các yêu cầu mới sẽ không thực hiện, thay vào đó sẽ sử dụng fallback method (myCustomerDetailsFallBack) để trả về kết quả dự phòng.
- **myCustomerDetailsFallBack:** Đây là phương thức fallback được sử dụng khi @CircuitBreaker hoặc @Retry chạy vào trạng thái mở và không thực hiện được phương thức gốc. Phương thức này sẽ trả về kết quả dự phòng (fallback result) để giúp ứng dụng duy trì tính ổn định và không gặp lỗi.

Bước 3: Cấu hình trong file application.properties (hoặc application.yml)

```
resilience4j.circuitbreaker.configs.default.registerHealthIndicator= true
resilience4j.circuitbreaker.instances.detailsForCustomerSupportApp.minimumNumberOfCalls= 5
resilience4j.circuitbreaker.instances.detailsForCustomerSupportApp.failureRateThreshold= 50
resilience4j.circuitbreaker.instances.detailsForCustomerSupportApp.waitDurationInOpenState= 30000
resilience4j.circuitbreaker.instances.detailsForCustomerSupportApp.permittedNumberOfCallsInHalfOpenState=2

resilience4j.retry.configs.default.registerHealthIndicator= true
resilience4j.instances.retryForCustomerDetails.maxRetryAttempts=3
resilience4j.instances.retryForCustomerDetails.waitDuration=2000

resilience4j.ratelimiter.configs.default.registerHealthIndicator= true
resilience4j.ratelimiter.instances.sayHello.timeoutDuration=5000
resilience4j.ratelimiter.instances.sayHello.limitRefreshPeriod=5000
resilience4j.ratelimiter.instances.sayHello.limitForPeriod=1
```

5. Deep dive on Retry pattern in microservices

Retry pattern sẽ thực hiện nhiều lần thử lại đã được định cấu hình khi một dịch vụ tạm thời bị lỗi. Pattern này rất hữu ích trong các trường hợp như gián đoạn mạng khi yêu cầu của client có thể thành công sau khi thử lại.

Đối với pattern thử lại, chúng tôi có thể cấu hình các giá trị sau:

- **maxAttempts:** Số lần thử tối đa
- **waitDuration:** Khoảng thời gian chờ cố định giữa các lần thử lại
- **retryExceptions:** Cấu hình danh sách các lớp Throwable được ghi lại là lỗi và do đó được thử lại.
- **ignoreExceptions:** Cấu hình danh sách các lớp Throwable bị bỏ qua và do đó không được thử lại.

Chúng tôi cũng có thể xác định cơ chế dự phòng nếu cuộc gọi dịch vụ không thành công ngay cả sau nhiều lần thử lại. Dưới đây là một cấu hình mẫu:

```
@Retry(name = "retryForCustomerDetails", fallbackMethod = "myCustomerDetailsFallBack")
```

6. Implementing Retry Pattern in microservices

```
@PostMapping("/myCustomerDetails")
@Retry(name = "retryForCustomerDetails", fallbackMethod =
"myCustomerDetailsFallBack")
public CustomerDetails myCustomerDetails(@RequestBody Customer customer) {
    Accounts accounts =
accountsRepository.findByCustomerId(customer.getCustomerId());
    List<Loans> loans = loansFeignClient.getLoansDetails(customer);
    List<Cards> cards = cardsFeignClient.getCardDetails(customer);

    CustomerDetails customerDetails = new CustomerDetails();
    customerDetails.setAccounts(accounts);
    customerDetails.setLoans(loans);
    customerDetails.setCards(cards);

    return customerDetails;
}

private CustomerDetails myCustomerDetailsFallBack(Customer customer,
Throwable t) {
    Accounts accounts =
accountsRepository.findByCustomerId(customer.getCustomerId());
    List<Loans> loans = loansFeignClient.getLoansDetails(customer);
    CustomerDetails customerDetails = new CustomerDetails();
    customerDetails.setAccounts(accounts);
    customerDetails.setLoans(loans);
    return customerDetails;
}
```

- **@Retry:** Đây là một annotation khác của Resilience4j, được sử dụng để triển khai mô hình Retry pattern cho phương thức myCustomerDetails. Nó giúp xử lý các trường hợp gọi API từ xa thất bại bằng cách thực hiện retry (thử lại) một số lần (có thể cấu hình số lần retry). Nếu sau các lần retry mà vẫn không thành công, fallback method (myCustomerDetailsFallBack) sẽ được gọi để trả về kết quả dự phòng.

7. Deep dive on Rate Limiter pattern in microservices

Rate limiter sẽ giúp ngừng quá tải dịch vụ với nhiều cuộc gọi hơn mức có thể sử dụng trong một thời gian nhất định. Đây là một kỹ thuật bắt buộc để chuẩn bị cho API của chúng tôi có tính khả dụng và độ tin cậy cao.

Mẫu này bảo vệ các API và điểm cuối dịch vụ khỏi các tác động có hại, chẳng hạn như từ chối dịch vụ, lỗi xếp tầng.

Đối với Rate limiter, chúng tôi có thể định cấu hình các giá trị sau:

- timeoutDuration - Thời gian chờ mặc định mà một chuỗi chờ sự cho phép
- limitForPeriod - Số lượng quyền khả dụng trong một khoảng thời gian làm mới giới hạn
- limitRefreshPeriod - Khoảng thời gian làm mới giới hạn. Sau mỗi khoảng thời gian rate limiter đặt số quyền của nó trở lại giá trị limitForPeriod

Có thể xác định cơ chế dự phòng nếu cuộc gọi dịch vụ không thành công do cấu hình rate limiter. Dưới đây là cấu hình mẫu:

```
@Retry(name = "retryForCustomerDetails", fallbackMethod =
"myCustomerDetailsFallBack")
```

8. Implementing Rate Limiter Pattern in microservices

```
@Retry(name = "retryForCustomerDetails", fallbackMethod =
"myCustomerDetailsFallBack")
public CustomerDetails myCustomerDetails(@RequestBody Customer customer) {
    ...
    return customerDetails;
}

private CustomerDetails myCustomerDetailsFallBack(Customer customer,
Throwable t) {
    ...
    return customerDetails;
}

@GetMapping("/sayHello")
@RateLimiter(name = "sayHello", fallbackMethod = "sayHelloFallback")
public String sayHello() {
    return "Hello, Welcome to EazyBank";
}

private String sayHelloFallback(Throwable t) {
    return "Hi, Welcome to EazyBank";
}
```

- **@RateLimiter:** Đây là một annotation khác của Resilience4j, được sử dụng để triển khai mô hình Rate Limiter pattern cho phương thức sayHello. Mô hình này giới hạn số lượng yêu cầu được thực hiện trong một khoảng thời gian nhất định, để tránh quá tải và bảo vệ hệ thống. Nếu số lượng yêu

cầu vượt quá mức đã cấu hình, fallback method (sayHelloFallback) sẽ được gọi để trả về kết quả dự phòng.

- sayHelloFallback: Đây là phương thức fallback được sử dụng khi @RateLimiter giới hạn số lượng yêu cầu. Nếu đã vượt quá mức giới hạn, phương thức này sẽ trả về kết quả dự phòng (fallback result) để ứng dụng có thể tiếp tục xử lý mà không bị gián đoạn.

9. Deep dive on Bulk head pattern in microservices

Một con tàu được chia thành nhiều khoang nhỏ bằng vách ngăn. Các vách ngăn được sử dụng để bít kín các bộ phận của con tàu nhằm ngăn chặn toàn bộ con tàu bị chìm trong trường hợp lũ lụt.

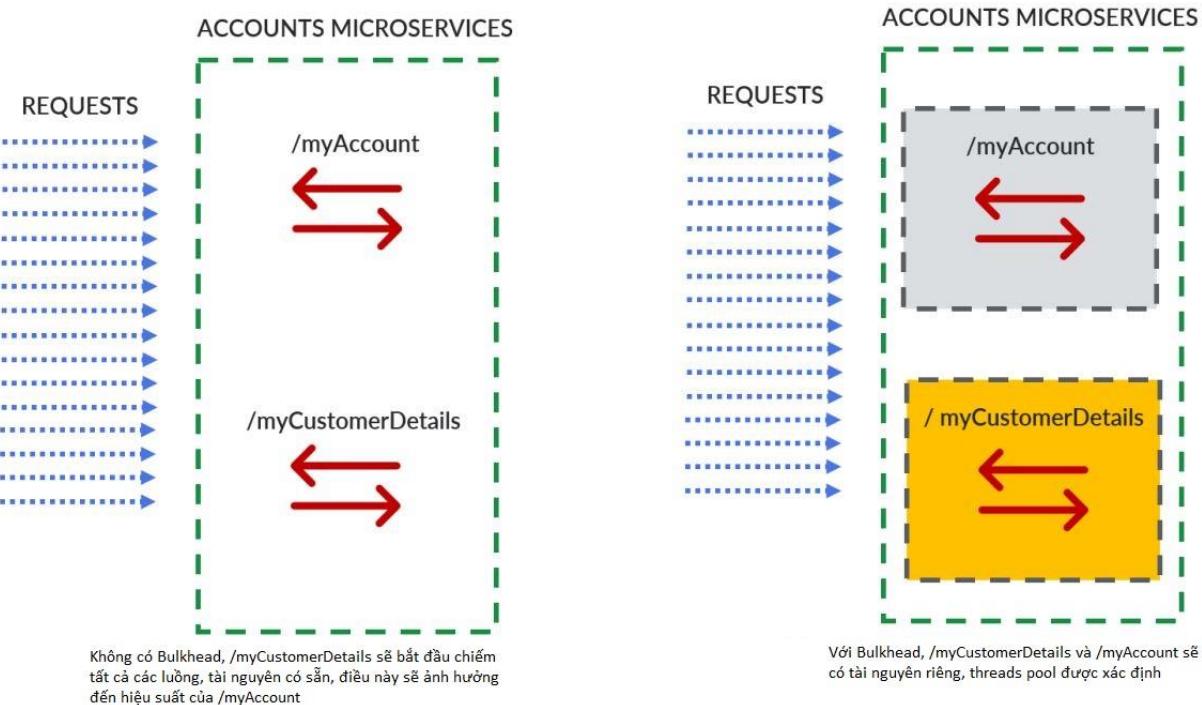
Tương tự như vậy, các tài nguyên microservice nên được cách ly theo cách sao cho lỗi của một thành phần không ảnh hưởng đến toàn bộ microservice.

Mô hình vách ngăn giúp phân bổ giới hạn tài nguyên có thể được sử dụng cho các dịch vụ cụ thể. Vì vậy, sự cạn kiệt tài nguyên có thể được giảm bớt.

Đối với mẫu Bulkhead, chúng ta có thể định cấu hình các giá trị sau:

- **maxConcurrentCalls** - Số lần thực thi song song tối đa mà vách ngăn cho phép
- **maxWaitDuration** - Lượng thời gian tối đa mà một thread sẽ bị chặn khi cố gắng vào vách ngăn bão hòa.

```
@Bulkhead(name = "bulkheadAccount", fallbackMethod =  
"bulkheadAccountsFallBack")
```



Section 8: Handling Routing & Cross cutting concerns in Microservices (Challenge 6)

1. Introduction to the challenges with Routing & Cross cutting concerns



HOW DO WE ROUTE BASED ON CUSTOM REQUIREMENTS

Làm thế nào để chúng tôi định tuyến dựa trên request tùy chỉnh ?

Nếu chúng tôi có một request tùy chỉnh để định tuyến các request đến đến đích thích hợp theo cả cách tĩnh và động, chúng tôi sẽ thực hiện điều đó như thế nào?



HOW DO WE HANDLE CROSS CUTTING CONCERNs?

Làm thế nào để chúng tôi xử lý những mối quan tâm xuyên suốt?

Trong kiến trúc microservice phân tán, làm cách nào để chúng tôi đảm bảo có các mối quan tâm xuyên suốt được thực thi nhất quán như logging, auditing, tracing, security và thu thập số liệu trên nhiều microservice ?



HOW DO WE BUILD A SINGLE GATEKEEPER?

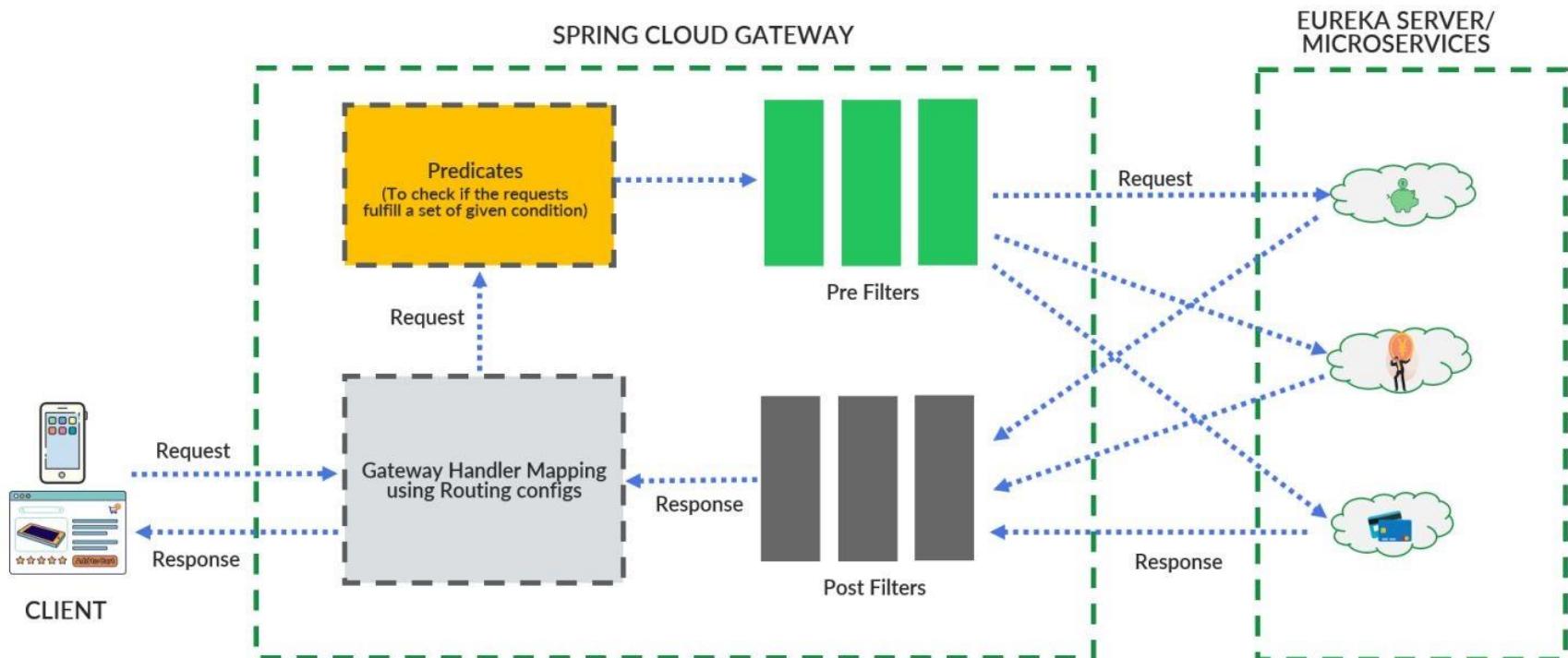
Làm cách nào để chúng tôi xây dựng single gatekeeper (một người gác cổng duy nhất) cho tất cả lưu lượng truy cập đến các vi dịch vụ của mình. Cơ quan này sẽ đóng vai trò là Policy Enforcement Point (PEP - Điểm thực thi chính sách) trung tâm cho tất cả các lệnh gọi dịch vụ?



2. Introduction to Spring Cloud Gateway

- **Spring Cloud Gateway** là API Gateway được nhóm Spring Cloud triển khai trên hệ sinh thái Spring. Nó cung cấp một cách đơn giản và hiệu quả để định tuyến các yêu cầu đến đích thích hợp bằng cách sử dụng Gateway Handler Mapping.
- **Cổng dịch vụ(service gateway)** đóng vai trò là người gác cổng cho tất cả lưu lượng truy cập đến các lệnh gọi microservice trong ứng dụng của chúng ta. Với một service gateway tại chỗ, các service client của chúng tôi không bao giờ gọi trực tiếp URL của một dịch vụ riêng lẻ mà thay vào đó, thực hiện tất cả các lệnh gọi đến service gateway.
- **Service gateway** nằm giữa tất cả các cuộc gọi từ client đến các dịch vụ riêng lẻ, nó cũng hoạt động như một Policy Enforcement Point trung tâm (PEP Điểm thực thi chính sách trung tâm) như bên dưới cho các cuộc gọi dịch vụ.
 - ✓ Định tuyến (Cả Tĩnh & Động)
 - ✓ Bảo mật (Xác thực & Ủy quyền)
 - ✓ Logging, Auditing and Metrics
- Spring Cloud Gateway là một thư viện để xây dựng cổng API, vì vậy nó trông giống như bất kỳ ứng dụng Spring Boot nào khác. Nếu bạn là Spring developer, bạn sẽ thấy rất dễ dàng để bắt đầu với Spring Cloud Gateway chỉ với một vài dòng mã.
- Spring Cloud Gateway nhằm mục đích nằm giữa người yêu cầu và tài nguyên đang được yêu cầu, nơi nó chặn, phân tích và sửa đổi mọi yêu cầu. Điều đó có nghĩa là bạn có thể định tuyến các yêu cầu dựa trên ngữ cảnh của chúng. Yêu cầu có bao gồm tiêu đề cho biết phiên bản API không? Chúng tôi có thể định tuyến yêu cầu đó đến backend phiên bản phù hợp. Yêu cầu có yêu cầu phiên dính không? Gateway có thể theo dõi phiên của từng người dùng.
- Spring Cloud Gateway thay thế Zuul vì những lý do và ưu điểm sau:
 - ✓ Spring Cloud Gateway là triển khai API gateway ưa thích từ Nhóm Spring Cloud. Nó được xây dựng trên Spring 5, Reactor và Spring WebFlux. Không chỉ vậy, nó còn tích hợp circuit breaker, service discovery với Eureka
 - ✓ Spring Cloud Gateway về bản chất là không chặn. Mặc dù Zuul 2 sau này cũng hỗ trợ nó, nhưng Spring Cloud Gateway vẫn có lợi thế hơn ở đây
 - ✓ Spring Cloud Gateway có hiệu suất vượt trội so với Zuul.

3. Deep dive on Spring Cloud Gateway internal architecture



Khi client đưa ra yêu cầu tới **Spring Cloud Gateway**, trước tiên, **Gateway Handler Mapping** sẽ kiểm tra xem yêu cầu có khớp với một **route** hay không. Matching này được thực hiện bằng cách sử dụng các **vị từ - predicate**. Nếu nó khớp với predicate thì yêu cầu sẽ được gửi đến các bộ lọc. **Post filter** nó sẽ gửi đến **microservice** thực tế hoặc **Eureka Server**.

- **Routes và Route Predicates:** Yếu tố trung tâm của Spring Cloud Gateway là khái niệm "routes" (đường đi). Routes được định nghĩa như một sự kết hợp giữa các "predicates" và "filters" (bộ lọc). Một predicate là một điều kiện để phù hợp các yêu cầu đến dựa trên tiêu chí như phương thức HTTP, tiêu đề hoặc đường dẫn. Khi một yêu cầu phù hợp với predicate cụ thể, route liên kết sẽ được chọn và yêu cầu được chuyển qua các bộ lọc tương ứng.
- **Bộ lọc (Filters):** Các bộ lọc có nhiệm vụ xử lý các yêu cầu và phản hồi khi chúng đi qua Gateway. Chúng được sử dụng cho các mục đích khác nhau, như ghi nhật ký, xác thực, thay đổi tiêu đề yêu cầu/ phản hồi, và nhiều tác vụ khác. Spring Cloud Gateway cung cấp sẵn các bộ lọc tích hợp, và bạn cũng có thể tạo các bộ lọc tùy chỉnh bằng cách triển khai giao diện `GatewayFilter` hoặc sử dụng `GlobalFilter` cho các bộ lọc toàn cầu áp dụng cho tất cả các route.

4. Building Spring Cloud Gateway service

Bước 1: Tạo project

- Truy cập <https://start.spring.io/>
- Điền tất cả các chi tiết cần thiết để tạo dự án Spring Boot của gatewayserver và thêm các phụ thuộc Spring Cloud Routing, Spring Boot Actuator, Eureka Discovery Client, Config Client, Spring Boot DevTools. Nhấp vào **GENERATE** sẽ tải xuống dự án maven máy chủ cổng ở định dạng zip
- Thêm chi tiết plugin spring-boot-maven-plugin cùng với chi tiết tên hình ảnh docker bên trong như chúng ta đã thảo luận trong khóa học. Chi tiết bổ sung về spring-boot-maven-plugin này sẽ giúp chúng tôi tạo hình ảnh docker bằng Buildpacks một cách dễ dàng.

Bước 2: Cấu hình Spring Cloud Gateway

Cấu hình file application.properties bên trong gatewayserver microservices

```
spring.application.name=gatewayserver

spring.config.import=optional:configserver:http://localhost:8071/

management.endpoints.web.exposure.include=*

## Configuring info endpoint
info.app.name=Gateway Server Microservice
info.app.description=Eazy Bank Gateway Server Application
info.app.version=1.0.0
management.info.env.enabled = true
management.endpoint.gateway.enabled=true
spring.cloud.gateway.discovery.locator.enabled=true
spring.cloud.gateway.discovery.lowerCaseServiceId=true

logging.level.com.eaztbytes.gatewayserver: DEBUG
```

Vui lòng đảm bảo tạo một gatewayserver.properties với nội dung bên dưới bên trong vị trí Config Server đang đọc các properties.

```
server.port=8072
eureka.instance.preferIpAddress = true
eureka.client.registerWithEureka = true
eureka.client.fetchRegistry = true
eureka.client.serviceUrl.defaultZone = http://localhost:8070/eureka/
```

5. Implementing Custom Routing using Spring Cloud Gateway

Tạo Bean trong main class hoặc có thể tạo file config:

```
@SpringBootApplication
public class GatewayserverApplication {

    public static void main(String[] args) {
        SpringApplication.run(GatewayserverApplication.class, args);
    }

    @Bean
    public RouteLocator myRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
            .route(p -> p
                .path("/eazybank/accounts/**")
                .filters(f ->
f.rewritePath("/eazybank/accounts/(?<segment>.*)", "/${segment}")
                    .addResponseHeader("X-Response-Time", new
Date().toString())
                    .uri("lb://ACCOUNTS"))
            .route(p -> p
                .path("/eazybank/loans/**")
                .filters(f ->
f.rewritePath("/eazybank/loans/(?<segment>.*)", "/${segment}")
                    .addResponseHeader("X-Response-Time", new
Date().toString()))
                    .uri("lb://LOANS"))
            .route(p -> p
                .path("/eazybank/cards/**")
                .filters(f ->
f.rewritePath("/eazybank/cards/(?<segment>.*)", "/${segment}")
                    .addResponseHeader("X-Response-Time", new
Date().toString()))
                    .uri("lb://CARDS")).build();
    }
}
```

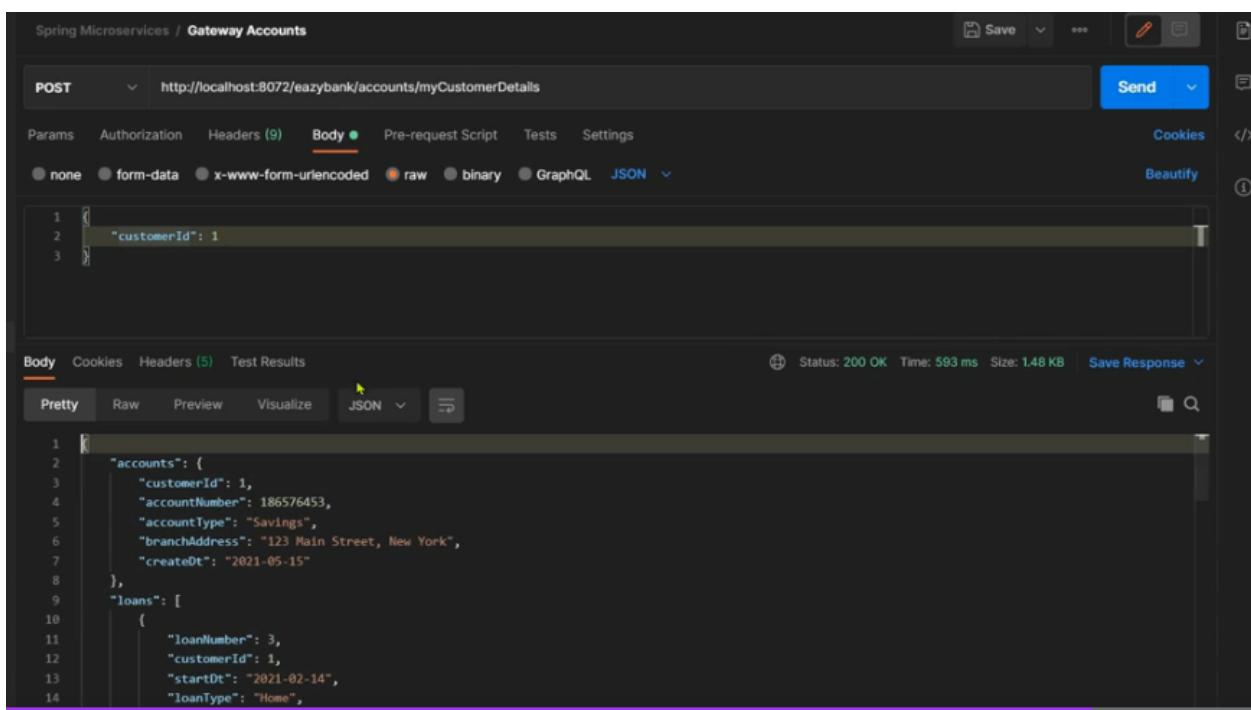
Mã trên là một ví dụ về cách triển khai tùy chỉnh định tuyến (Custom Routing) bằng Spring Cloud Gateway. Đoạn mã sử dụng Spring @Bean để tạo một **RouteLocator** dựa trên cấu hình được xây dựng bởi **RouteLocatorBuilder**.

Trong đó:

- **RouteLocator:** Là một interface trong Spring Cloud Gateway, giúp xác định cách định tuyến các yêu cầu đến các URL cụ thể.
- **RouteLocatorBuilder:** Là một lớp hỗ trợ giúp xây dựng các định tuyến (routes) cho RouteLocator.

Có 3 Route sau:

- **Route /eazybank/accounts/**:**
path: Tiêu chí để xác định các yêu cầu phù hợp với route này là có đường dẫn bắt đầu bằng /eazybank/accounts/.
filters: Điều này cho phép bạn thêm các bộ lọc cho route. Trong trường hợp này, đoạn mã thêm một bộ lọc để tái viết đường dẫn yêu cầu từ /eazybank/accounts/{segment} thành /{segment} và thêm một header X-Response-Time với giá trị là thời gian hiện tại.
uri: Đây là địa chỉ của dịch vụ backend cụ thể mà route này sẽ chuyển tiếp các yêu cầu tới. Trong trường hợp này, nó sẽ chuyển tiếp các yêu cầu đến dịch vụ có tên ACCOUNTS (được quản lý bởi Spring Cloud Load Balancer).
- **Route /eazybank/loans/**:** Tương tự như route trên, nhưng áp dụng cho đường dẫn bắt đầu bằng /eazybank/loans/ và chuyển tiếp yêu cầu tới dịch vụ có tên LOANS.
- **Route /eazybank/cards/**:** Tương tự như hai route trước, nhưng áp dụng cho đường dẫn bắt đầu bằng /eazybank/cards/ và chuyển tiếp yêu cầu tới dịch vụ có tên CARDS.



The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** <http://localhost:8072/eazybank/accounts/myCustomerDetails>
- Body (JSON):**

```
1 [
2   "customerId": 1
3 ]
```
- Response Status:** 200 OK
- Response Body (Pretty JSON):**

```
1 {
2   "accounts": [
3     {
4       "customerId": 1,
5       "accountNumber": 186576453,
6       "accountType": "Savings",
7       "branchAddress": "123 Main Street, New York",
8       "createDt": "2021-05-15"
9     }
10    ],
11    "loans": [
12      {
13        "loanNumber": 3,
14        "customerId": 1,
15        "startDt": "2021-02-14",
16        "loanType": "Home",
17        "amount": 100000
18      }
19    ]
20 }
```

Định tuyến đến microservice tương ứng.

6. Implementing Cross cutting concern Tracing & Logging using Gateway Server

Để triển khai các mối quan tâm xuyên suốt bên trong microservice của bạn, vui lòng tạo các lớp TraceFilter.java, ResponseTraceFilter.java, FilterUtility.java. Tất cả chúng sẽ giống như dưới đây,

TraceFilter.java

```
@Order(1)
@Component
public class TraceFilter implements GlobalFilter {

    private static final Logger logger =
LoggerFactory.getLogger(TraceFilter.class);

    @Autowired
    FilterUtility filterUtility;

    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain
chain) {
        HttpHeaders requestHeaders = exchange.getRequest().getHeaders();
        if (isCorrelationIdPresent(requestHeaders)) {
            logger.debug("EazyBank-correlation-id found in tracing filter: {}",
",
                filterUtility.getCorrelationId(requestHeaders));
        } else {
            String correlationID = generateCorrelationId();
            exchange = filterUtility.setCorrelationId(exchange, correlationID);
            logger.debug("EazyBank-correlation-id generated in tracing filter:
{}.", correlationID);
        }
        return chain.filter(exchange);
    }

    private boolean isCorrelationIdPresent(HttpHeaders requestHeaders) {
        if (filterUtility.getCorrelationId(requestHeaders) != null) {
            return true;
        } else {
            return false;
        }
    }

    private String generateCorrelationId() {
        return java.util.UUID.randomUUID().toString();
    }
}
```

Đoạn mã trên triển khai một Global Filter trong Spring Cloud Gateway để thực hiện theo dõi (tracing) các yêu cầu và thêm hoặc trích xuất mã theo dõi (correlation ID) cho các yêu cầu.

- **@Order(1):** Đánh dấu lớp TraceFilter là một GlobalFilter và chỉ định độ ưu tiên của nó trong việc thực hiện các bộ lọc. Với @Order(1), bộ lọc này sẽ được thực hiện trước các bộ lọc khác có độ ưu tiên cao hơn.

- **FilterUtility:** Là một lớp tiện ích (utility class) có nhiệm vụ hỗ trợ các phương thức liên quan đến việc trích xuất và thiết lập mã theo dõi (correlation ID) trong yêu cầu.
- **filter():** Phương thức này được triển khai từ interface GlobalFilter. Nó được thực hiện mỗi khi có một yêu cầu đến Gateway.
- **exchange:** Đại diện cho trao đổi thông tin giữa Gateway và client/service. Nó chứa thông tin về yêu cầu đến và phản hồi đi.
- **isCorrelationIdPresent():** Phương thức này kiểm tra xem mã theo dõi đã có trong tiêu đề yêu cầu hay chưa. Nếu đã có, nó sẽ trả về true, ngược lại trả về false.
- **generateCorrelationId():** Phương thức này sinh ra một mã theo dõi mới bằng cách sử dụng UUID (Unique Universal Identifier). UUID là một số duy nhất có độ dài 128 bit.

Trong phương thức filter(), bộ lọc TraceFilter kiểm tra nếu mã theo dõi đã có trong yêu cầu thì ghi log để theo dõi mã theo dõi đã tìm thấy. Nếu không có mã theo dõi, nó sẽ tạo một mã mới bằng cách gọi phương thức generateCorrelationId() và đặt mã mới vào yêu cầu thông qua filterUtility.setCorrelationId(). Sau đó, nó ghi log để theo dõi mã theo dõi đã được tạo.

Cuối cùng, phương thức filter() trả về chain.filter(exchange); để tiếp tục tiến hành định tuyến yêu cầu thông qua các bộ lọc khác và chuyển tiếp đến dịch vụ backend (nếu có).

ResponseTraceFilter.java

```
@Configuration
public class ResponseTraceFilter {

    private static final Logger logger =
LoggerFactory.getLogger(ResponseTraceFilter.class);

    @Autowired
    FilterUtility filterUtility;

    @Bean
    public GlobalFilter postGlobalFilter() {
        return (exchange, chain) -> {
            return chain.filter(exchange).then(Mono.fromRunnable(() -> {
                HttpHeaders requestHeaders = exchange.getRequest().getHeaders();
                String correlationId =
filterUtility.getCorrelationId(requestHeaders);
                logger.debug("Updated the correlation id to the outbound
headers. {}", correlationId);

                exchange.getResponse().getHeaders().add(filterUtility.CORRELATION_ID,
correlationId);
            }));
        };
    }
}
```

Đoạn mã trên triển khai một Global Filter (Bộ lọc toàn cầu) trong Spring Cloud Gateway để cập nhật mã theo dõi (correlation ID) trong các tiêu đề phản hồi (response headers) của các yêu cầu.

- **GlobalFilter: postGlobalFilter()** triển khai một GlobalFilter. Điều này cho phép bạn thực hiện xử lý các yêu cầu và phản hồi trên toàn bộ hệ thống Spring Cloud Gateway.

- **postGlobalFilter():** Trong phương thức này, bạn triển khai một bộ lọc toàn cầu (Global Filter) được gọi là postGlobalFilter(). Khi bộ lọc này được thực hiện, nó sẽ chạy sau khi các bộ lọc khác đã hoàn thành xử lý yêu cầu và đang sắp hoàn thành xử lý phản hồi.
- **return (exchange, chain) -> { ... }:** Đây là một lambda expression triển khai GlobalFilter. Nó nhận vào hai tham số exchange và chain, và trả về một Mono.
- **chain.filter(exchange):** Gọi phương thức filter() trên chain để chuyển tiếp yêu cầu đến các bộ lọc tiếp theo trong chuỗi lọc.
- **.then(Mono.fromRunnable(() -> { ... })):** Sau khi hoàn thành xử lý yêu cầu và phản hồi, lúc này then() sẽ thực thi một Runnable để cập nhật mã theo dõi vào tiêu đề phản hồi (response headers).
- **exchange.getRequest().getHeaders():** Lấy danh sách tiêu đề yêu cầu từ ServerWebExchange.
- **filterUtility.getCorrelationId(requestHeaders):** Sử dụng filterUtility để trích xuất mã theo dõi từ tiêu đề yêu cầu.
- **exchange.getResponse().getHeaders().add(filterUtility.CORRELATION_ID, correlationId):** Thêm mã theo dõi vào tiêu đề phản hồi (response headers) bằng cách sử dụng filterUtility.CORRELATION_ID làm tên tiêu đề. Điều này giúp đồng bộ hóa mã theo dõi giữa yêu cầu và phản hồi.

Cuối cùng, bộ lọc này sẽ chạy sau khi yêu cầu đã được xử lý và sẽ thêm mã theo dõi vào phản hồi để theo dõi từ đầu đến cuối trong quá trình định tuyến các yêu cầu qua Gateway.

FilterUtility.java

```
@Component
public class FilterUtility {

    public static final String CORRELATION_ID = "eazybank-correlation-id";

    public String getCorrelationId(HttpServletRequest requestHeaders) {
        if (requestHeaders.get(CORRELATION_ID) != null) {
            List<String> requestHeaderList =
requestHeaders.get(CORRELATION_ID);
            return requestHeaderList.stream().findFirst().get();
        } else {
            return null;
        }
    }

    public ServerWebExchange setRequestHeader(ServerWebExchange exchange,
String name, String value) {
        return
exchange.mutate().request(exchange.getRequest().mutate().header(name,
value).build()).build();
    }

    public ServerWebExchange setCorrelationId(ServerWebExchange exchange,
String correlationId) {
        return this.setRequestHeader(exchange, CORRELATION_ID, correlationId);
    }
}
```

Đoạn mã trên triển khai một lớp tiện ích (utility class) FilterUtility trong Spring Cloud Gateway. Lớp này cung cấp các phương thức hỗ trợ cho việc trích xuất và thiết lập mã theo dõi (correlation ID) trong tiêu đề yêu cầu (request headers) của các yêu cầu.

- **public String getCorrelationId(HttpServletRequest requestHeaders):** Phương thức này nhận vào HttpServletRequest của yêu cầu và trả về mã theo dõi (correlation ID) từ tiêu đề yêu cầu nếu có. Nếu không tìm thấy mã theo dõi, phương thức sẽ trả về null.
- **public ServerWebExchange setRequestHeader(ServerWebExchange exchange, String name, String value):** Phương thức này thay đổi tiêu đề yêu cầu bằng cách thêm hoặc cập nhật một tiêu đề với tên name và giá trị value. Nó nhận vào ServerWebExchange của yêu cầu và trả về một phiên bản ServerWebExchange mới với tiêu đề yêu cầu đã được thay đổi.
- **public ServerWebExchange setCorrelationId(ServerWebExchange exchange, String correlationId):** Phương thức này sử dụng setRequestHeader() để thiết lập mã theo dõi vào tiêu đề yêu cầu với tên là CORRELATION_ID.

Các phương thức trong lớp FilterUtility đều có mục đích hỗ trợ cho việc trích xuất và thiết lập mã theo dõi. Việc trích xuất mã theo dõi từ tiêu đề yêu cầu cho phép các bộ lọc sau này có thể sử dụng mã theo dõi này để đảm bảo theo dõi yêu cầu từ đầu đến cuối trong hệ thống.

Chỉnh sửa Controller của các microservice

Cập nhật tất cả các **Controller** quan trọng như AccountsController.java, LoansController.java, CardsController.java để chấp nhận chuỗi tương quan @RequestHeader("eazybank-correlation-id") làm đầu vào bên trong các tham số của phương thức.

```
@RestController
public class AccountsController {

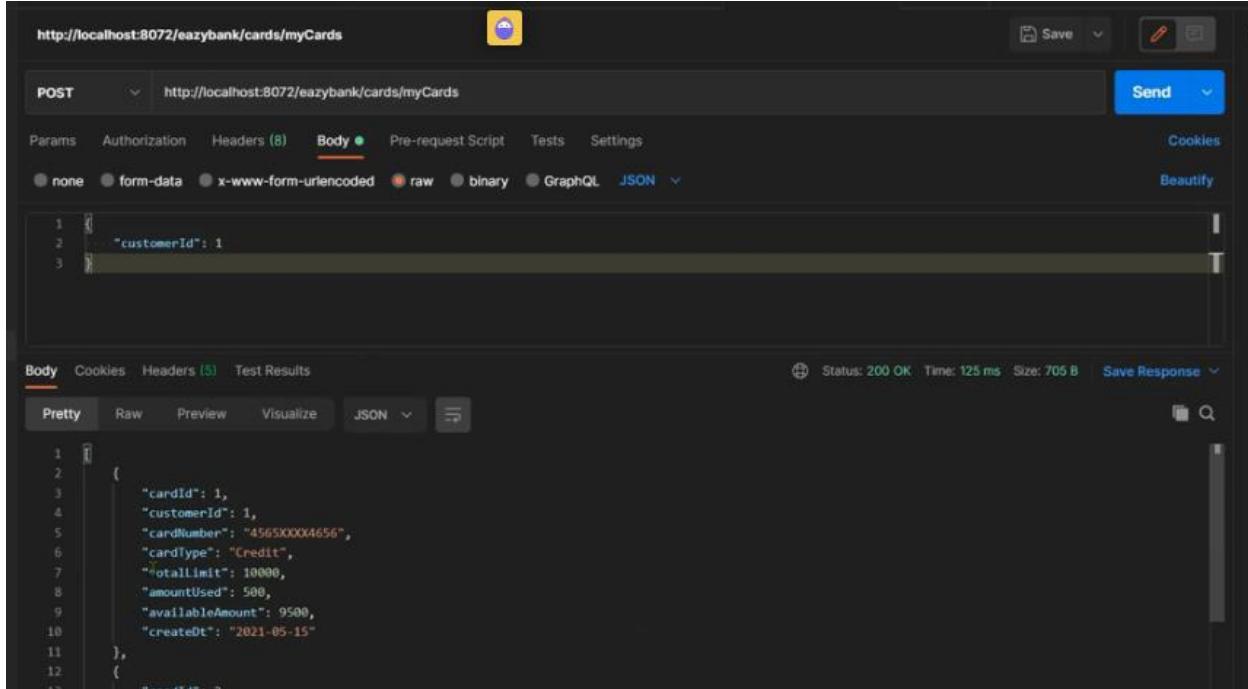
    @PostMapping("/myCustomerDetails")
    @CircuitBreaker(name =
"detailsForCustomerSupportApp", fallbackMethod="myCustomerDetailsFallBack")
    @Retry(name = "retryForCustomerDetails", fallbackMethod =
"myCustomerDetailsFallBack")
    public CustomerDetails myCustomerDetails(@RequestHeader("eazybank-
correlation-id") String correlationid, @RequestBody Customer customer) {
        Accounts accounts =
accountsRepository.findByCustomerId(customer.getCustomerId());
        List<Loans> loans =
loansFeignClient.getLoansDetails(correlationid, customer);
        List<Cards> cards =
cardsFeignClient.getCardDetails(correlationid, customer);

        CustomerDetails customerDetails = new CustomerDetails();
        customerDetails.setAccounts(accounts);
        customerDetails.setLoans(loans);
        customerDetails.setCards(cards);

        return customerDetails;
    }
}
```

Khởi động lại vi dịch vụ máy chủ cổng của bạn và gọi API REST

<http://localhost:8072/eazybank/accounts/myCustomerDetails> thông qua Postman bằng cách chuyển yêu cầu bên dưới ở định dạng JSON, sẽ nhận được phản hồi từ vi dịch vụ tài khoản có tất cả các chi tiết liên quan đến tài khoản, khoản vay và thẻ.



```
POST http://localhost:8072/eazybank/cards/myCards
```

Body (JSON)

```
{"customerId": 1}
```

Body (Pretty)

```
1 {
2     "cardId": 1,
3     "customerId": 1,
4     "cardNumber": "4565XXXX4656",
5     "cardType": "Credit",
6     "totalLimit": 10000,
7     "amountUsed": 500,
8     "availableAmount": 9500,
9     "createDt": "2021-05-15"
10 }
```

Console (Java Application) C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (May 15, 2021, 2:25:23 PM)

```
2021-05-15 14:35:33.529 INFO 22496 --- [ restartedMain] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [After]
2021-05-15 14:35:33.529 INFO 22496 --- [ restartedMain] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [Before]
2021-05-15 14:35:33.529 INFO 22496 --- [ restartedMain] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [Between]
2021-05-15 14:35:33.529 INFO 22496 --- [ restartedMain] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [Cookie]
2021-05-15 14:35:33.529 INFO 22496 --- [ restartedMain] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [Header]
2021-05-15 14:35:33.529 INFO 22496 --- [ restartedMain] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [Method]
2021-05-15 14:35:33.529 INFO 22496 --- [ restartedMain] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [Path]
2021-05-15 14:35:33.529 INFO 22496 --- [ restartedMain] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [Query]
2021-05-15 14:35:33.529 INFO 22496 --- [ restartedMain] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [ReadBody]
2021-05-15 14:35:33.529 INFO 22496 --- [ restartedMain] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [RemoteAddr]
2021-05-15 14:35:33.529 INFO 22496 --- [ restartedMain] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [Weight]
2021-05-15 14:35:33.529 INFO 22496 --- [ restartedMain] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [Weighted]
2021-05-15 14:35:33.529 INFO 22496 --- [ restartedMain] o.s.b.a.e.EndpointLinksResolver : Exposing 17 endpoint(s) beneath base path '/actuator'
2021-05-15 14:35:33.889 INFO 22496 --- [ restartedMain] o.s.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2021-05-15 14:35:33.892 INFO 22496 --- [ restartedMain] DiscoveryClientOptionalArgsConfiguration : Eureka HTTP Client uses RestTemplate.
2021-05-15 14:35:33.924 WARN 22496 --- [ restartedMain] IgniteCloudLoadBalancerCaffeineWarnLogger : Spring Cloud LoadBalancer is currently working with the default cache. You can switch to using Caffeine
2021-05-15 14:35:33.935 INFO 22496 --- [ restartedMain] o.s.c.n.eureka.InstanceInfoFactory : Setting initial instance status as: STARTING
2021-05-15 14:35:33.935 INFO 22496 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Initializing Eureka in region us-east-1
2021-05-15 14:35:33.989 INFO 22496 --- [ restartedMain] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration
2021-05-15 14:35:33.989 INFO 22496 --- [ restartedMain] c.n.d.s.r.aws.ConfigClusterResolver : Double check if configuration has changed
2021-05-15 14:35:33.995 INFO 22496 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Single vip registry refresh property : null
2021-05-15 14:35:33.995 INFO 22496 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Force full registry fetch : false
2021-05-15 14:35:33.995 INFO 22496 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Application is null : false
2021-05-15 14:35:33.995 INFO 22496 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Registered Applications size is zero : true
2021-05-15 14:35:33.995 INFO 22496 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Application version is -1: true
2021-05-15 14:35:33.995 INFO 22496 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Getting all instance registry info from the eureka server
2021-05-15 14:35:33.998 INFO 22496 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : The response status is 200
2021-05-15 14:35:33.998 INFO 22496 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Starting consistency check, renew interval is: 30
2021-05-15 14:35:33.998 INFO 22496 --- [ restartedMain] c.n.d.s.e.InstanceInfoReplicator : InstanceInfoReplicator onDemand update allowed rate per min is 4
2021-05-15 14:35:33.998 INFO 22496 --- [ restartedMain] com.netflix.discovery.InstanceInfoReplicator : Discovery Client initialized at timestamp 1621069533095 with initial instances count: 4
2021-05-15 14:35:33.998 INFO 22496 --- [ restartedMain] o.s.c.n.e.s.EurekaServiceRegistry : Registering application GATEWAYSERVER with eureka with status UP
2021-05-15 14:35:33.998 INFO 22496 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Saw local status change event StatusChangeEvent@{[timestamp=1621069533095, current=UP, previous=STARTING} : DiscoveryClient_GATEWAYSERVER/host.docker.internal:gatewayserver:8072: registering service...
2021-05-15 14:35:34.018 INFO 22496 --- [infoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_GATEWAYSERVER/host.docker.internal:gatewayserver:8072 - registration status: 204
2021-05-15 14:35:34.054 INFO 22496 --- [ restartedMain] o.s.w.e.embedded.netty.NettyWebServer : Netty started on port 8072
2021-05-15 14:35:34.054 INFO 22496 --- [ restartedMain] o.s.c.n.e.s.EurekaServiceRegistration : Updated gatewayserver:8072
2021-05-15 14:35:34.376 INFO 22496 --- [ restartedMain] c.n.d.s.r.aws.ConfigClusterResolver : Started GatewayserviceReplication in 2.946 seconds (JVM running for 608.666)
2021-05-15 14:35:34.388 INFO 22496 --- [ restartedMain] c.n.d.s.r.aws.ConfigClusterResolver : Condition evaluation unchanged
2021-05-15 14:40:33.968 INFO 22496 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration
2021-05-15 14:45:33.982 INFO 22496 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration
```

=> Generating and pushing Docker images, Updating Docker Compose file tương tự các section trên.

Section 9: Distributed tracing & Log aggregation in Microservices (Challenge 7)

1. Introduction to the challenges related to Distributed tracing & Log aggregation

HOW DO WE DEBUG WHERE A PROBLEM IN MICROSERVICES?

Làm thế nào để chúng tôi debug vấn đề trong các dịch vụ vi mô?
Làm cách nào để chúng tôi theo dõi một hoặc nhiều giao dịch trên nhiều service, máy vật lý, và lưu trữ dữ liệu khác nhau. Và cố gắng tìm chính xác vấn đề hoặc lỗi ở đâu?

HOW DO WE AGGREGATE ALL APPLICATION LOGS?

Làm thế nào để chúng tôi tổng hợp tất cả log ứng dụng?
Làm cách nào để chúng tôi kết hợp tất cả log từ nhiều dịch vụ vào một vị trí trung tâm nơi chúng có thể được indexed, searched, filtered, và grouped để tìm ra các lỗi góp phần gây ra sự cố?

HOW DO WE MONITOR OUR CHAIN OF SERVICE CALLS?

Làm thế nào để chúng tôi theo dõi chuỗi cuộc gọi dịch vụ của mình?
Làm cách nào để chúng tôi hiểu đối với một cuộc gọi dịch vụ chuỗi cụ thể, đường dẫn mà dịch vụ đó di chuyển trong microservice của chúng tôi, thời gian thực hiện tại mỗi microservice, v.v.?



2. Introduction to Spring Cloud Sleuth & Zipkin

Spring Cloud Sleuth

- Spring Cloud Sleuth cung cấp khả năng tự động cấu hình Spring Boot để theo dõi phân tán.
- Nó thêm id theo dõi và khoảng thời gian vào tất cả log, vì vậy bạn chỉ có thể trích xuất từ một dấu vết hoặc khoảng thời gian nhất định trong trình tổng hợp log.
- Nó thực hiện điều này bằng cách thêm các bộ lọc và tương tác với các thành phần khác của Spring để cho phép các ID tương quan được tạo đi qua tất cả các lệnh gọi hệ thống.

Zipkin

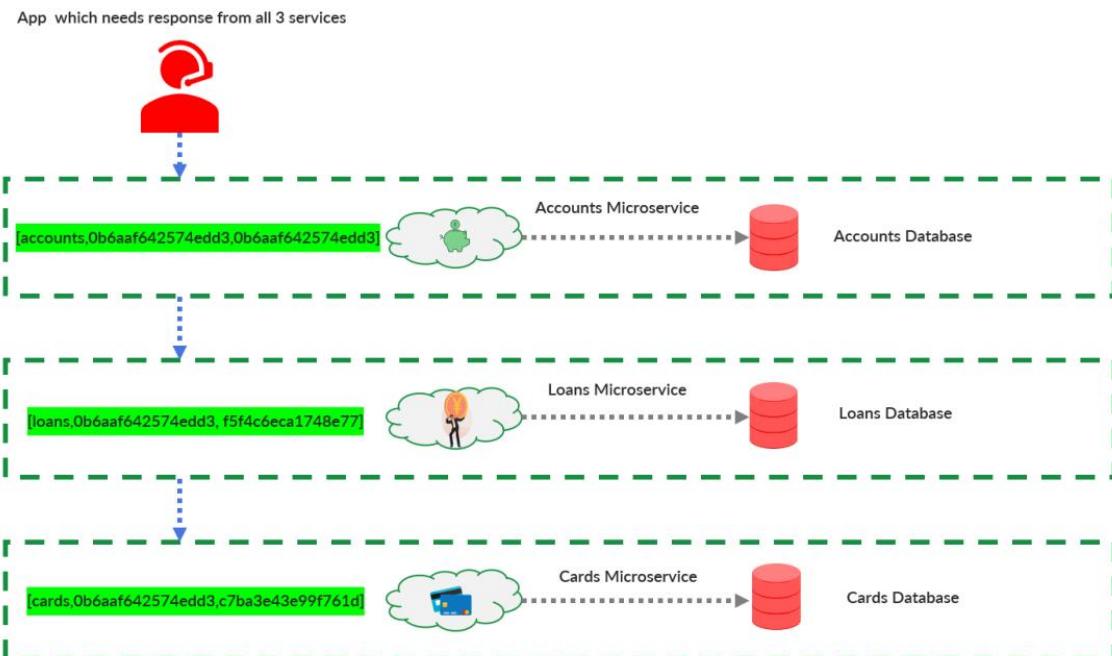
- Zipkin là một công cụ trực quan hóa dữ liệu nguồn mở có thể giúp tổng hợp tất cả log và thu thập dữ liệu thời gian cần thiết để khắc phục sự cố về độ trễ trong kiến trúc microservice.
- Nó cho phép chúng tôi chia nhỏ một giao dịch thành các phần cấu thành của nó và xác định một cách trực quan nơi có thể có các điểm nóng về hiệu suất. Do đó, giảm thời gian phân loại bằng cách bối cảnh hóa các lỗi và sự chậm trễ.

3. Deep dive on Spring Cloud Sleuth & it's tracing format

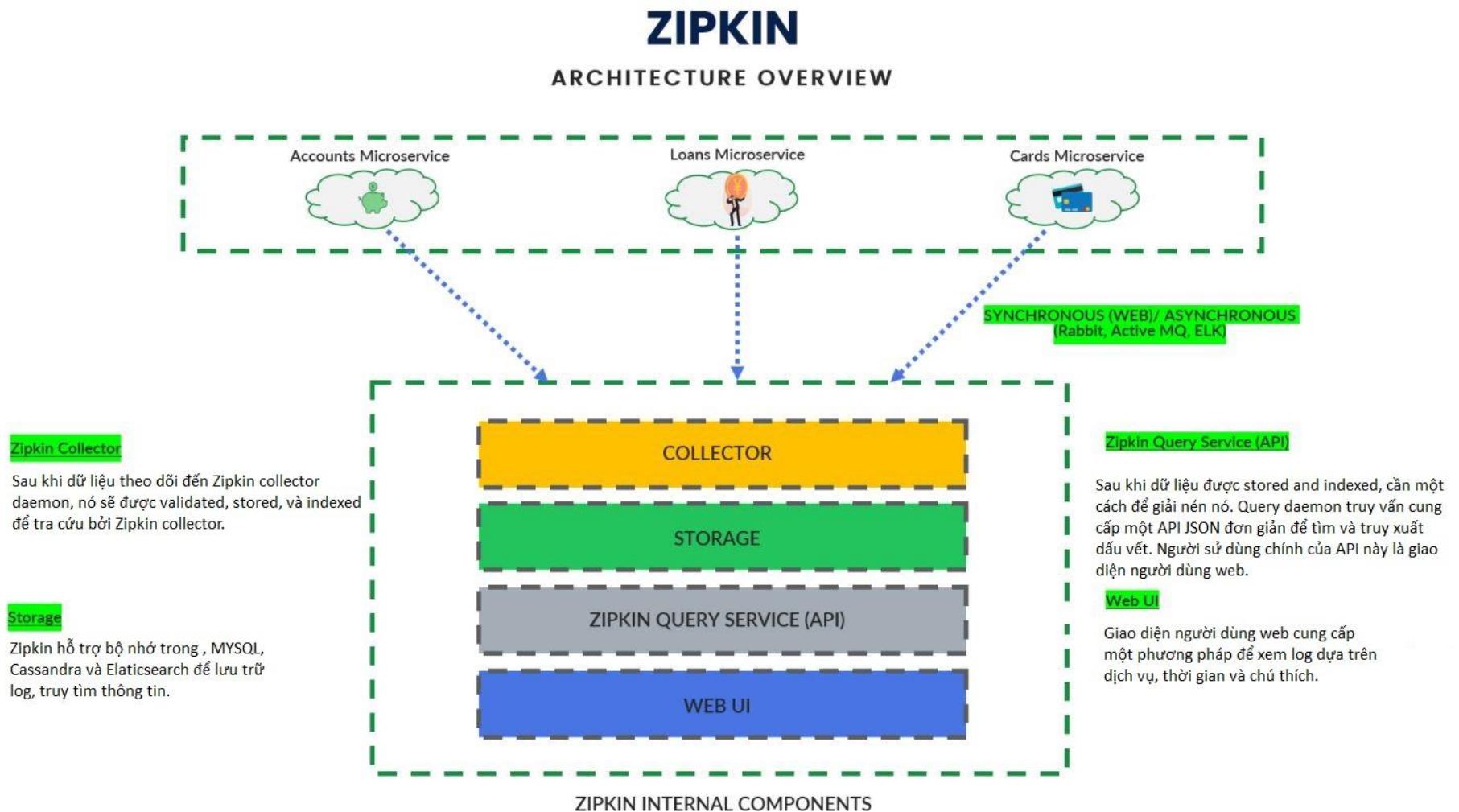
Spring Cloud Sleuth sẽ thêm ba phần thông tin vào tất cả log được ghi bởi một microservice.

[<App Name>,<Trace ID>,].

- App Name - Tên ứng dụng của dịch vụ: Đây sẽ là tên ứng dụng thực hiện mục log. Spring Cloud Sleuth lấy tên này từ thuộc tính 'spring.application.name'.
- Trace ID: Trace ID là thuật ngữ tương đương với ID tương quan(correlation ID). Đó là một số duy nhất đại diện cho toàn bộ giao dịch-transaction.
- Span ID: Span ID là một ID duy nhất đại diện cho một phần của giao dịch tổng thể. Mỗi dịch vụ tham gia trong giao dịch sẽ có span ID riêng. Span ID đặc biệt phù hợp khi bạn tích hợp với Zipkin để trực quan hóa các giao dịch của mình.



4. Deep dive on Zipkin internal architecture



Kiến trúc nội bộ của Zipkin bao gồm các thành phần và quá trình hoạt động chính như sau:

Các thành phần:

- Collector: Thành phần này nhận dữ liệu truy vấn từ các dịch vụ khác nhau. Nó chấp nhận dữ liệu ở nhiều định dạng như JSON, Thrift, hoặc Protobuf thông qua HTTP hoặc giao thức như Kafka, RabbitMQ, v.v.
- Storage: Dữ liệu truy vấn được thu thập được lưu trữ trong một hệ thống lưu trữ sau khi được xử lý. Có thể sử dụng các hệ thống lưu trữ phổ biến như Elasticsearch, MySQL, Cassandra, v.v.
- Index: Thành phần này tạo các chỉ mục phù hợp dựa trên metadata của các truy vấn để tối ưu hóa thời gian tìm kiếm trace.
- Query: Thành phần này xử lý các truy vấn đến từ giao diện người dùng Zipkin hoặc các ứng dụng khác. Nó truy xuất dữ liệu cần thiết từ hệ thống lưu trữ, xử lý và trả về kết quả cho người dùng.
- UI: Giao diện người dùng cung cấp một trang web cho phép người dùng xem và phân tích các traces và spans.

Mô hình Trace và Span:

- Trace: Trace đại diện cho hành trình toàn bộ của một yêu cầu qua nhiều dịch vụ. Nó bao gồm một trace ID duy nhất và chứa nhiều spans.
- Span: Span đại diện cho một hoạt động cụ thể trong một trace. Nó có một span ID, một parent span ID (nếu nó thuộc về một span khác), timestamp (thời gian bắt đầu và kết thúc), và các thông tin metadata khác như tên dịch vụ, các tag và log.

Quá trình thu thập Trace:

- Công cụ Zipkin client được tích hợp vào ứng dụng và dịch vụ để tạo và gửi dữ liệu trace. Các thư viện client tự động tạo và quản lý spans trong quá trình thực thi mã.
- Quá trình sampling được hỗ trợ để tránh quá tải hệ thống bằng việc giới hạn việc thu thập dữ liệu trace chỉ vào một phần trăm nhất định dựa trên các quy tắc xác định trước.

Quá trình thu thập dữ liệu:

- Dữ liệu trace được tạo ra trong ứng dụng qua quá trình xử lý yêu cầu.
- Thư viện Zipkin client tạo và gắn thông tin vào spans liên quan.
- Các spans được gửi đến thành phần Collector thông qua một điểm cuối HTTP hoặc message broker.
- Collector xử lý các spans và chuyển tiếp chúng đến thành phần Storage để lưu trữ dữ liệu.

Lưu trữ dữ liệu và tạo chỉ mục:

- Zipkin hỗ trợ nhiều hệ thống lưu trữ, chẳng hạn như Elasticsearch, MySQL, Cassandra, v.v.
- Các spans được lưu trữ và tạo chỉ mục dựa trên metadata của trace và span để truy vấn hiệu quả.
- Chỉ mục giúp tìm kiếm nhanh các traces phù hợp với các tiêu chí như tên dịch vụ, tên hoạt động hoặc khoảng thời gian cụ thể.

Truy vấn và hiển thị trace:

- Giao diện người dùng Zipkin hoặc các ứng dụng khác thực hiện các truy vấn đến thành phần Query để truy xuất các trace cụ thể.
- Thành phần Query truy xuất dữ liệu liên quan từ hệ thống lưu trữ và xử lý.
- Giao diện người dùng hiển thị các trace, hiển thị cấu trúc tổng thể của trace và span, thời gian thực hiện và bất kỳ vấn đề hiệu suất hoặc lỗi nào có thể xảy ra.

5. Implementing Distributed tracing with Spring Cloud Sleuth

Bước 1: Thêm phụ thuộc

Mở tệp pom.xml của tất cả các microservices accounts, loans, cards, configserver, eurekaserver, gatewayserver và đảm bảo thêm phần phụ thuộc bắt buộc bên dưới của Spring Cloud Sleuth vào tất cả chúng.

Từ Spring Cloud 2022.0.0 & Spring Boot 3, dự án Sleuth đã bị xóa và cốt lõi của dự án này đã chuyển sang Micrometer Tracing. Trong trường hợp nếu bạn đang sử dụng micrometer, vui lòng thêm các phụ thuộc maven bên dưới vào bên trong pom.xml của tất cả các dự án.

```
<dependency>
    <groupId>io.github.openfeign</groupId>
    <artifactId>feign-micrometer</artifactId>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-tracing-bridge-otel</artifactId>
</dependency>
<dependency>
    <groupId>io.opentelemetry</groupId>
    <artifactId>opentelemetry-exporter-zipkin</artifactId>
</dependency>
```

Bước 2: Thay đổi các file controller (AccountsController.java, LoansController.java , CardsController.java)

```
@RestController
public class AccountsController {

    private static final Logger logger =
LoggerFactory.getLogger(AccountsController.class);
    @PostMapping("/myCustomerDetails")
    public CustomerDetails myCustomerDetails(@RequestHeader("eazybank-
correlation-id") String correlationid, @RequestBody Customer customer) {
        logger.info("myCustomerDetails() method started");
        Accounts accounts =
accountsRepository.findByCustomerId(customer.getCustomerId());
        List<Loans> loans =
loansFeignClient.getLoansDetails(correlationid, customer);
        List<Cards> cards =
cardsFeignClient.getCardDetails(correlationid, customer);

        CustomerDetails customerDetails = new CustomerDetails();
        customerDetails.setAccounts(accounts);
        customerDetails.setLoans(loans);
        customerDetails.setCards(cards);
        logger.info("myCustomerDetails() method ended");
        return customerDetails;
    }
}
```

Bước 3: Chạy lại các service sau khi thay đổi:

The image consists of three vertically stacked screenshots illustrating the execution of a service call and its corresponding logs.

Top Screenshot (Postman): A screenshot of the Postman application interface. It shows a list of four pending requests at the top. Below is a detailed view of a single POST request to `http://localhost:8072/accounts/myCustomerDetails`. The "Body" tab is selected, displaying a JSON payload:

```
1 [ {  
2     "customerId": 1  
3 } ]
```

The "Test Results" tab shows a successful response with status 200 OK, time 65 ms, and size 1.44 KB. The response body is displayed in pretty JSON format:

```
1 {  
2     "accounts": [  
3         {"customerId": 1,  
4             "accountNumber": "1865/6453,  
5                 "accountType": "Savings",  
6                 "branchAddress": "123 Main Street, New York",  
7                 "createDt": "2021-05-29"  
8             },  
9             "loans": [  
10                 {  
11                     "loanNumber": 3,  
12                     "customerId": 1,  
13                     "loanDt": "2021-07-31"  
14             }  
15         ]  
16     }  
17 }
```

Middle Screenshot (Eclipse IDE Console): A screenshot of the Eclipse IDE's Console view. It displays two lines of DEBUG-level log output from a "GatewayserverApplication" process:

```
2021-05-29 20:55:08.799 DEBUG [gatewayserver,4f5a143f9ad88bc,4f5a143f9ad88bc] 21812 --- [ctor-http-nio-3] c.e.g.filters.RequestTracefilter : EasyBank-correlation-id generated in tracing filter:  
2021-05-29 20:55:08.798 DEBUG [gatewayserver,,] 21812 --- [ctor-http-nio-3] c.e.g.filters.ResponseTracefilter : Updated the correlation id to the outbound headers. 66e9377f-e872-41ac-a5db-7c4a5b2
```

Bottom Screenshot (Java Application Log): A screenshot of the Eclipse IDE's Console view showing log output from a Java application named "AccountsApplication". It includes timestamp, log level, logger, message, and thread information:

```
2021-05-29 20:55:08.755 INFO [accounts,4f5a143f9ad88bc,2c2991495970cb] 5884 --- [lo-8888-exec-10] c.e.a.controller.AccountsController : myCustomerDetails() method started  
2021-05-29 20:55:08.788 INFO [accounts,4f5a143f9ad88bc,2c2991495970cb] 5884 --- [lo-8888-exec-10] c.e.a.controller.AccountsController : myCustomerDetails() method ended
```

6. Implementing Log aggregation with Zipkin Server

Bây giờ để sử dụng theo dõi log bằng Zipkin, hãy chạy lệnh docker 'docker run -d -p 9411:9411 openzipkin/zipkin'. Lệnh docker này sẽ khởi động bộ chứa docker zipkin bằng docker image được cung cấp. (<https://zipkin.io/>)

Để xác thực xem zipkin server có khởi động thành công hay không, hãy truy cập URL <http://localhost:9411/zipkin> bên trong trình duyệt của bạn. Bạn sẽ có thể xem trang chủ zipkin.

Bước 1: Thêm phụ thuộc

Mở tệp pom.xml của tất cả các microservices accounts, loans, cards, configserver, eurekaserver, gatewayserver và đảm bảo thêm phần phụ thuộc bắt buộc của Zipkin vào tất cả chúng nếu bạn đang sử dụng Spring Cloud Sleuth.

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-sleuth-zipkin</artifactId>
</dependency>
```

Mặt khác, nếu bạn đang sử dụng micrometer, vui lòng thêm phần phụ thuộc bắt buộc bên dưới của Zipkin.

```
<dependency>
    <groupId>io.opentelemetry</groupId>
    <artifactId>opentelemetry-exporter-zipkin</artifactId>
</dependency>
```

Bước 2: thêm application.properties

Mở application.properties của tất cả các microservices accounts, loans, cards, configserver, eurekaserver, gatewayserver và đảm bảo thêm các thuộc tính/cấu hình bên dưới vào tất cả chúng dựa trên việc bạn đang sử dụng Sleuth hay Micrometer.

```
# Micrometer related properties
management.tracing.sampling.probability=1.0
management.zipkin.tracing.endpoint=http://localhost:9411/api/v2/spans
management.metrics.distribution.percentiles-
histogram.http.server.requests=true
logging.pattern.level=%5p [${spring.application.name:-},%X{traceId:-},
,%X{spanId:-}]

# Sleuth related properties
spring.sleuth.sampler.percentage=1
spring.zipkin.baseUrl=http://localhost:9411/
```

Bước 3: Bắt đầu tất cả các microservice

The screenshot shows two windows side-by-side. The top window is a Postman request to `http://localhost:8072/accounts/myCustomerDetails`. The Body tab contains the following JSON:

```
1 | {
2 |   "customerId": 1
3 | }
```

The response status is `200 OK` with a duration of `70 ms` and a size of `1.44 KB`. The bottom window is a browser displaying the Zipkin UI at `localhost:9411/zipkin/?serviceName=accounts&lookback=15m&endTs=1622303744316&limit=10`. It shows a table of traces for the `accounts` service. There are three results, all labeled `gatewayserver: post`, with start times ranging from 'a few seconds ago' to 'a minute ago'. Each trace has 7 spans and a duration between 2.555s and 63.182ms.

Start Time	Spans	Duration
a minute ago (05/29 21:24:42:183)	7	2.555s
a few seconds ago (05/29 21:25:42:423)	7	63.507ms
a minute ago (05/29 21:24:59:184)	7	63.182ms

C localhost:9411/zipkin/?serviceName=accounts&lookback=15m&endTs=1622303744316&limit=10

Zipkin

Find a trace Dependencies ENGLISH Search by trace

serviceName accounts X + RUN QUERY

3 Results EXPAND ALL COLLAPSE ALL Service filters

	Root	Start Time	Spans	Duration
▼	gatewayserver: post	a minute ago (05/29 21:24:42:183)	7	2.555s SHOW
^	gatewayserver: post	a few seconds ago (05/29 21:25:42:423)	7	63.507ms SHOW
	Trace ID: 44d9caebee7be9e2 accounts (3) gatewayserver (2) loans (1) cards (1)			
▼	gatewayserver: post	a minute ago (05/29 21:24:59:184)	7	63.182ms SHOW

Quickstart - OpenZipkin Zipkin localhost:9411/zipkin/traces/44d9caebee7be9e2 Guest

Zipkin

Find a trace Dependencies ENGLISH Search by trace ID

GATEWAYSERVER: post Duration: 63.507ms Services: 4 Depth: 5 Total Spans: 7 Trace ID: 44d9caebee7be9e2 DOWNLOAD JSON

GATEWAYSERVER: post [0ms] [21.169ms] [42.338ms] [63.507ms]

- GATEWAYSERVER: post [0ms] [49.308ms] [post /mycustomerdetails [47.172ms]] [post /myloans [8.935ms]] [post /mycards [8.767ms]]
- ACCOUNTS: post /mycustomerdetails [47.172ms]
- ACCOUNTS: post /myloans [8.935ms]
- LOANS: post /mycards [8.767ms]
- CARDS: post /mycards [8.767ms]

GATEWAYSERVER: post

Span ID: 44d9caebee7be9e2 Parent ID: None

Annotations

Tags

- http.method POST
- http.path /accounts/myCustomerDetails
- Client Address [::1]:64798

SHOW ALL ANNOTATIONS

7. Pushing Sleuth message into RabbitMQ

- Bây giờ để thiết lập Rabbit MQ server, hãy chạy lệnh docker 'docker run -it --name Rabbitmq -p 5672:5672 -p 15672:15672 Rabbitmq:3-manager'. Lệnh docker này sẽ khởi động bộ chứa docker liên quan đến Rabbit MQ bằng hình ảnh docker được cung cấp.
- Để xác thực xem Rabbit MQ server có khởi động thành công hay không, hãy truy cập URL <http://localhost:15672> bên trong trình duyệt của bạn và đăng nhập bằng tên người dùng/mật khẩu với tư cách là guest.

Bước 1: Thêm phụ thuộc

```
<dependency>
    <groupId>org.springframework.amqp</groupId>
    <artifactId>spring-rabbit</artifactId>
</dependency>
```

Bước 2: thêm application.properties

Lưu ý: Thư viện Micrometer Tracing mới, thay thế cho Sleuth, không hỗ trợ đẩy log không đồng bộ vào queues giống như RabbitMQ.

- Mở application.properties của tất cả các microservices accounts, loans, cards, configserver, eurekaserver, gatewayserver và đảm bảo thêm các thuộc tính/cấu hình bên dưới:

```
spring.zipkin.sender.type=rabbit
spring.zipkin.rabbitmq.queue=zipkin
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

Cần tạo 1 queue mới:

The screenshot shows the RabbitMQ Management UI with the 'Queues' tab selected. A new queue named 'zipkin' is being created. The queue details are as follows:

Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
zipkin	classic	D Args		Nan	Nan	Nan			

Below the table, there are additional configuration options for the queue, including 'Message TTL', 'Auto expire', 'Max length', 'Max length bytes', 'Overflow behaviour', 'Dead letter exchange', 'Dead letter routing key', 'Single active consumer', 'Maximum priority', 'Lazy mode', and 'Master locator'.

Bước 3: Bắt đầu tất cả các microservice

Spring Microservices / http://localhost:8072/accounts/myAccount

POST http://localhost:8072/accounts/myCustomerDetails

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies </> Beautify ⓘ

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 [
2   ...
3   "customerId": 1
4 ]

```

Body Cookies Headers (4) Test Results Status: 200 OK Time: 3.34 s Size: 1.44 KB Save Response

Pretty Raw Preview Visualize JSON

```

1 [
2   ...
3   "accounts": [
4     {
5       "customerId": 1,
6       "accountNumber": 186576453,
7       "accountType": "Savings",
8       "branchAddress": "123 Main Street, New York",
9       "createDt": "2021-05-29"
10    },
11    "loans": [
12      {
13        "loanNumber": 3,
14        "customerId": 1,
15        "startDt": "2021-02-14",
16        "loanType": "Home"
17      }
18    ]
19  ]
20 ]

```

Messaging that just works — Rail Zipkin RabbitMQ Management localhost:15672/#/connections

RabbitMQ™ RabbitMQ 3.8.16 Erlang 23.3.4.1 Refreshed 20

Overview Connections Channels Exchanges Queues Admin

Connections

All connections (6)

Pagination Page 1 of 1 - Filter: Regex ?

Overview		Details			Network		+/-
Name	User name	State	SSL / TLS	Protocol	Channels	From client	To client
172.17.0.1:45932	guest	green running	○	AMQP 0-9-1	2	0 B/s	0 B/s
172.17.0.1:45952	guest	green running	○	AMQP 0-9-1	2	146 iB/s	0 B/s
172.17.0.1:45984	guest	green running	○	AMQP 0-9-1	2	208 iB/s	0 B/s
172.17.0.1:46006	guest	green running	○	AMQP 0-9-1	2	0 B/s	0 B/s
172.17.0.1:46026	guest	green running	○	AMQP 0-9-1	2	95 iB/s	0 B/s
172.17.0.1:46070	guest	green running	○	AMQP 0-9-1	2	0 B/s	0 B/s

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

RabbitMQ™ RabbitMQ 3.8.16 Erlang 23.3.4.1

Overview Connections Channels Exchanges Queues Admin

Queue zipkin

- Overview

Queued messages last minute ?

Ready	28
Unacked	0
Total	28

Message rates last minute ?

Publish	0.20/s
Deliver (manual ack)	0.00/s
Deliver (auto ack)	0.00/s
Consumer ack	0.00/s
Redelivered	0.00/s
Get (manual ack)	0.00/s
Get (auto ack)	0.00/s
Get (empty)	0.00/s

Details

Features	arguments: x-queue-type: classic durable: true	State	running	Total	28	Ready	28	Una
Policy		Consumers	0	Messages	?			
Operator policy		Consumer capacity	?	Message body bytes	?	13 kB	13 kB	
Effective policy definition			0%	Process memory	?	108 kB		

RabbitMQ™ RabbitMQ 3.8.16 Erlang 23.3.4.1

Overview Connections Channels Exchanges Queues Admin

22:12:00 22:12:10 22:12:20 22:12:30 22:12:40 22:12:50

Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s
--------------------	--------	------------------	--------

Details

Features	arguments: x-queue-type: classic durable: true	State	running	Total	29	Ready	29	Una
Policy		Consumers	0	Messages	?			
Operator policy		Consumer capacity	?	Message body bytes	?	13 kB	13 kB	
Effective policy definition			0%	Process memory	?	88 kB		

- Consumers
- Bindings
- Publish message
- Get messages

Warning: getting messages from a queue is a destructive action. ?

Ack Mode: Nack message requeue true

Encoding: Auto string / base64

Messages: 54

Get Message(s)

- Move messages
- Delete

=> Generating and pushing Docker images, Updating Docker Compose file tương tự các section trên.

Section 10: Monitoring Microservices Metrics & Health (Challenge 8)

1. Introduction to the challenges related to monitoring microservices



HOW DO WE MONITOR SERVICES METRICS?

Làm cách nào để chúng tôi theo dõi các số liệu như mức sử dụng CPU, số liệu JVM, v.v. cho tất cả các ứng dụng microservice mà chúng tôi có trong mạng của mình một cách dễ dàng và hiệu quả?



HOW DO WE MONITOR SERVICES HEALTH?

Làm cách nào để chúng tôi theo dõi trạng thái/sức khỏe của tất cả các ứng dụng microservice mà chúng tôi có trong mạng của mình ở một nơi duy nhất?



HOW DO WE CREATE ALERTS BASED ON MONITORING ?

Làm cách nào để chúng tôi tạo cảnh báo/thông báo cho bất kỳ hành vi bất thường nào của dịch vụ?



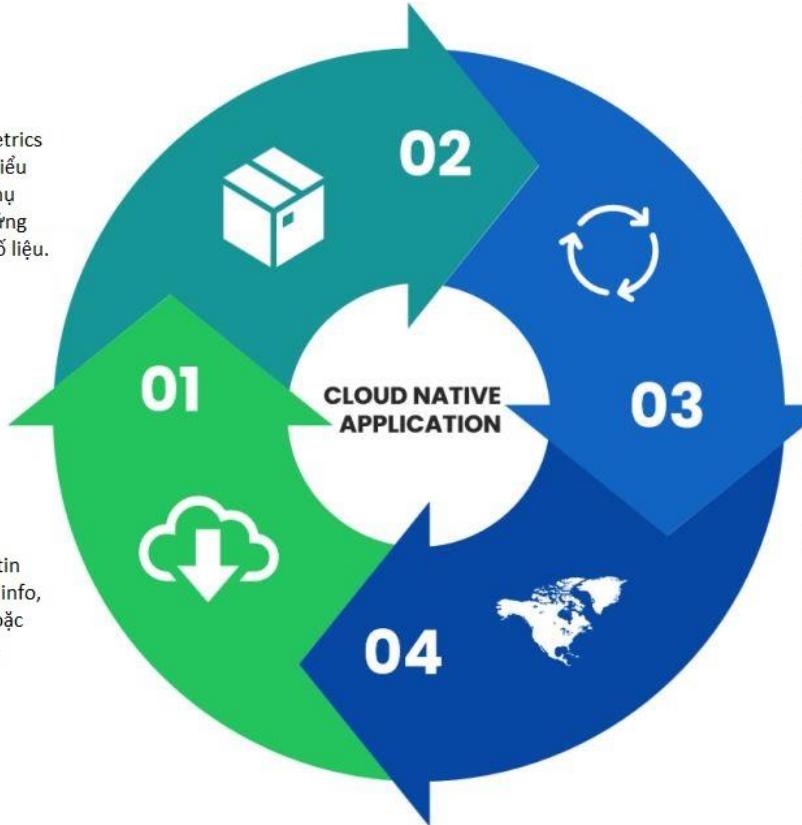
2. Different approaches to monitor microservices- Các cách tiếp cận khác nhau để giám sát microservice

MICROMETER

Micrometer tự động hiển thị dữ liệu/actuator/metrics thành thứ mà hệ thống giám sát của bạn có thể hiểu được. Tất cả những gì bạn cần làm là đưa phần phụ thuộc micromet của nhà cung cấp cụ thể đó vào ứng dụng của bạn. Hãy nghĩ về SLF4J, nhưng đổi với số liệu.

ACTUATOR

Actuator chủ yếu được sử dụng để hiển thị thông tin vận hành về ứng dụng đang chạy —health, metrics, info, dump, env, v.v. Nó sử dụng các HTTP endpoints hoặc JMX bean để cho phép chúng tôi tương tác với nó.



PROMETHEUS

Đây là cơ sở dữ liệu time-series database lưu trữ dữ liệu số liệu của chúng tôi bằng cách lấy dữ liệu đó (sử dụng công cụ quét dữ liệu tích hợp) theo định kỳ qua HTTP. Nó cũng có giao diện người dùng đơn giản, nơi chúng tôi có thể trực quan hóa/truy vấn tất cả các số liệu đã thu thập.

GRAFANA

Grafana có thể lấy dữ liệu từ nhiều nguồn dữ liệu khác nhau như Prometheus và cung cấp giao diện người dùng phong phú, nơi bạn có thể nhanh chóng xây dựng biểu đồ tùy chỉnh và tạo trang tổng quan từ nhiều biểu đồ ngay lập tức. Nó cũng cho phép bạn đặt cảnh báo dựa trên quy tắc cho thông báo.

3. Setup of micrometer inside microservices

Bước 1: Thêm phụ thuộc

Mở tệp pom.xml của tất cả các microservices accounts, loans, cards và đảm bảo thêm các thành phần phụ thuộc bắt buộc bên dưới của micrometer,prometheus vào tất cả chúng.

```
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

Bước 2: tạo một bean TimedAspect

Mở AccountsApplication.java và tạo một bean TimedAspect.

```
@SpringBootApplication
@EnableFeignClients
@RefreshScope
@ComponentScans({ @ComponentScan("com.eazybytes.accounts.controller") })
@EnableJpaRepositories("com.eazybytes.accounts.repository")
@EntityScan("com.eazybytes.accounts.model")
public class AccountsApplication {
    public static void main(String[] args) {
        SpringApplication.run(AccountsApplication.class, args);
    }

    @Bean
    public TimedAspect timedAspect(MeterRegistry registry) {
        return new TimedAspect(registry);
    }
}
```

Bước 3: Mở AccountsController.java và custom metric cho API "/myAccount" với sự trợ giúp của chú thích @Timed

```
@RestController
public class AccountsController {

    private static final Logger logger =
LoggerFactory.getLogger(AccountsController.class);
    @PostMapping("/myAccount")
    @Timed(value = "getAccountDetails.time", description = "Time taken to
return Account Details")
    public Accounts getAccountDetails(@RequestBody Customer customer) {
        ...
    }
}
```

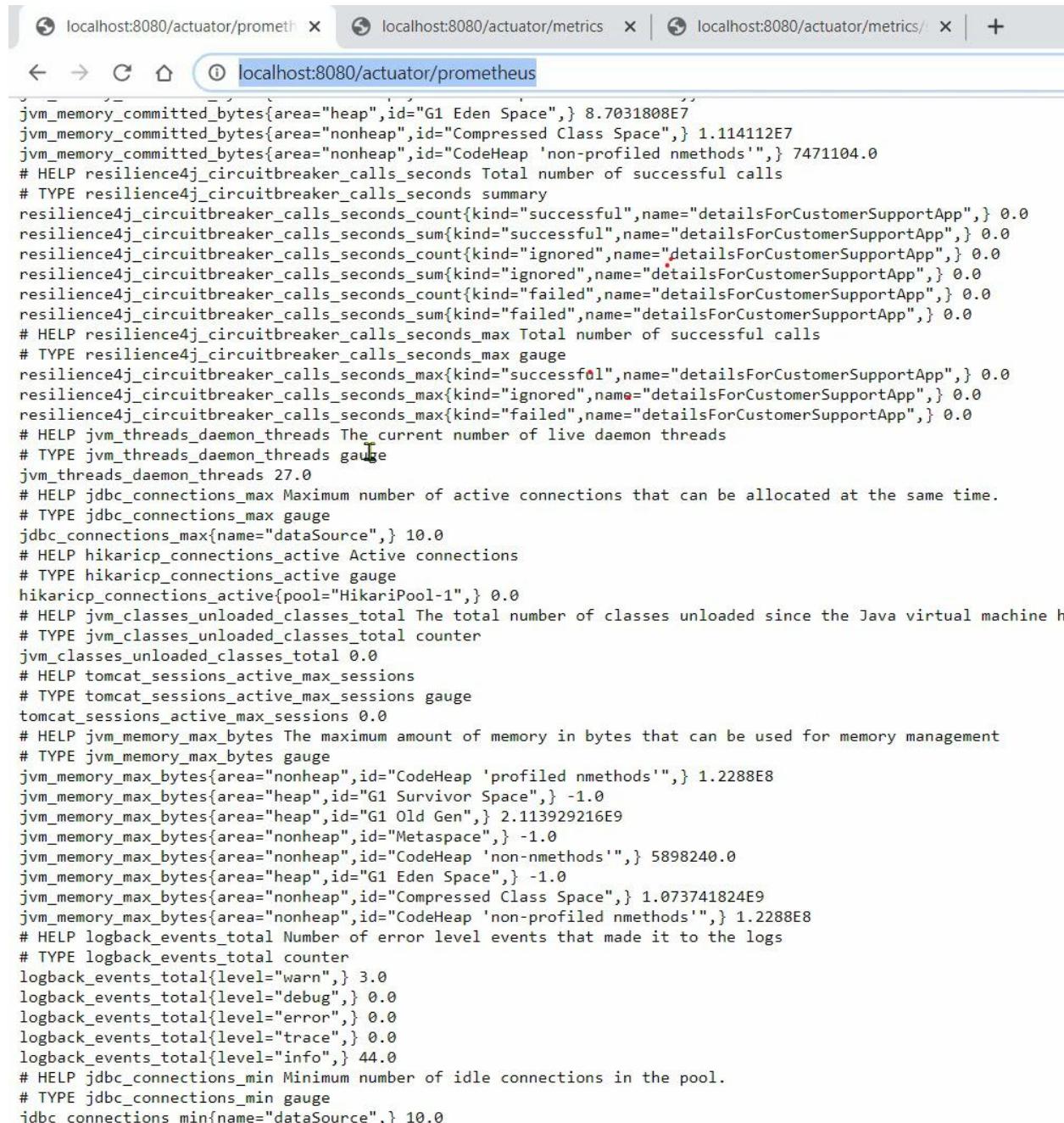
Bước 4: Start các microservice theo thứ tự configserver, eurekaserver, accounts.

Bước 5: Kiểm tra

Sau khi bắt đầu tất cả các vi dịch vụ bắt buộc, hãy truy cập URL `http://localhost:8080/myAccount` thông qua Postman bằng cách chuyển yêu cầu bên dưới ở định dạng JSON.

```
{  
    "customerId": 1  
}
```

Mở URL `http://localhost:8080/actuator/prometheus` để xác thực xem custom metric 'getAccountDetails.time' mà đã tạo có hiển thị trong thông tin chỉ số hay không.



```
jvm_memory_committed_bytes{area="heap",id="G1 Eden Space",} 8.7031808E7  
jvm_memory_committed_bytes{area="nonheap",id="Compressed Class Space",} 1.114112E7  
jvm_memory_committed_bytes{area="nonheap",id="CodeHeap 'non-profiled nmETHODS'",} 7471104.0  
# HELP resilience4j_circuitbreaker_calls_seconds Total number of successful calls  
# TYPE resilience4j_circuitbreaker_calls_seconds summary  
resilience4j_circuitbreaker_calls_seconds_count{kind="successful",name="detailsForCustomerSupportApp",} 0.0  
resilience4j_circuitbreaker_calls_seconds_sum{kind="successful",name="detailsForCustomerSupportApp",} 0.0  
resilience4j_circuitbreaker_calls_seconds_count{kind="ignored",name="detailsForCustomerSupportApp",} 0.0  
resilience4j_circuitbreaker_calls_seconds_sum{kind="ignored",name="detailsForCustomerSupportApp",} 0.0  
resilience4j_circuitbreaker_calls_seconds_count{kind="failed",name="detailsForCustomerSupportApp",} 0.0  
resilience4j_circuitbreaker_calls_seconds_sum{kind="failed",name="detailsForCustomerSupportApp",} 0.0  
# HELP resilience4j_circuitbreaker_calls_seconds_max Total number of successful calls  
# TYPE resilience4j_circuitbreaker_calls_seconds_max gauge  
resilience4j_circuitbreaker_calls_seconds_max{kind="successful",name="detailsForCustomerSupportApp",} 0.0  
resilience4j_circuitbreaker_calls_seconds_max{kind="ignored",name="detailsForCustomerSupportApp",} 0.0  
resilience4j_circuitbreaker_calls_seconds_max{kind="failed",name="detailsForCustomerSupportApp",} 0.0  
# HELP jvm_threads_daemon_threads The current number of live daemon threads  
# TYPE jvm_threads_daemon_threads gauge  
jvm_threads_daemon_threads 27.0  
# HELP jdbc_connections_max Maximum number of active connections that can be allocated at the same time.  
# TYPE jdbc_connections_max gauge  
jdbc_connections_max{name="dataSource",} 10.0  
# HELP hikaricp_connections_active Active connections  
# TYPE hikaricp_connections_active gauge  
hikaricp_connections_active{pool="HikariPool-1",} 0.0  
# HELP jvm_classes_unloaded_classes_total The total number of classes unloaded since the Java virtual machine has started  
# TYPE jvm_classes_unloaded_classes_total counter  
jvm_classes_unloaded_classes_total 0.0  
# HELP tomcat_sessions_active_max_sessions  
# TYPE tomcat_sessions_active_max_sessions gauge  
tomcat_sessions_active_max_sessions 0.0  
# HELP jvm_memory_max_bytes The maximum amount of memory in bytes that can be used for memory management  
# TYPE jvm_memory_max_bytes gauge  
jvm_memory_max_bytes{area="nonheap",id="CodeHeap 'profiled nmETHODS'",} 1.2288E8  
jvm_memory_max_bytes{area="heap",id="G1 Survivor Space",} -1.0  
jvm_memory_max_bytes{area="heap",id="G1 Old Gen",} 2.113929216E9  
jvm_memory_max_bytes{area="nonheap",id="Metaspace",} -1.0  
jvm_memory_max_bytes{area="nonheap",id="CodeHeap 'non-nmETHODS'",} 5898240.0  
jvm_memory_max_bytes{area="heap",id="G1 Eden Space",} -1.0  
jvm_memory_max_bytes{area="nonheap",id="Compressed Class Space",} 1.073741824E9  
jvm_memory_max_bytes{area="nonheap",id="CodeHeap 'non-profiled nmETHODS'",} 1.2288E8  
# HELP logback_events_total Number of error level events that made it to the logs  
# TYPE logback_events_total counter  
logback_events_total{level="warn",} 3.0  
logback_events_total{level="debug",} 0.0  
logback_events_total{level="error",} 0.0  
logback_events_total{level="trace",} 0.0  
logback_events_total{level="info",} 44.0  
# HELP jdbc_connections_min Minimum number of idle connections in the pool.  
# TYPE jdbc_connections_min gauge  
jdbc_connections_min{name="dataSource",} 10.0
```

4. Setup of Prometheus to monitor microservices

Tạo docker image cho accounts, loans, cards của microservice và đẩy chúng vào Docker Hub

Tạo một prometheus.yml bên trong đường dẫn và nội dung như hình bên dưới:

```
# \accounts\docker-compose\monitoring\prometheus.yml

global:
  scrape_interval:      5s # Set the scrape interval to every 5 seconds.
  evaluation_interval: 5s # Evaluate rules every 5 seconds.
scrape_configs:
  - job_name: 'accounts'
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ['accounts:8080']
  - job_name: 'loans'
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ['loans:8090']
  - job_name: 'cards'
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ['cards:9000']
```

Bây giờ trong cùng một thư mục chứa prometheus.yml, hãy tạo một tệp docker-compose.yml với nội dung sau,

```
version: "3.8"

services:
  .....
  prometheus:
    image: prom/prometheus:latest
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    networks:
      - eazybank
  .....
networks:
  eazybank:
```

Mở công cụ command line có chứa docker-compose.yml và chạy lệnh soạn thảo docker "**docker-compose up**" để khởi động tất cả các microservice bằng một lệnh duy nhất. Tất cả các container đang chạy có thể được xác thực bằng cách chạy lệnh docker "docker ps".

Mở URL <http://localhost:9090/targets/> bên trong trình duyệt và xác thực tất cả các chi tiết, biểu đồ có trong prometheus.

Prometheus Time Series Collection | localhost:8080/actuator/prometheus | localhost:8080/actuator/metrics/ | +

localhost:9090/targets

Prometheus Alerts Graph Status Help Classic UI

Targets

All Unhealthy Collapse All

accounts (1/1 up) show less

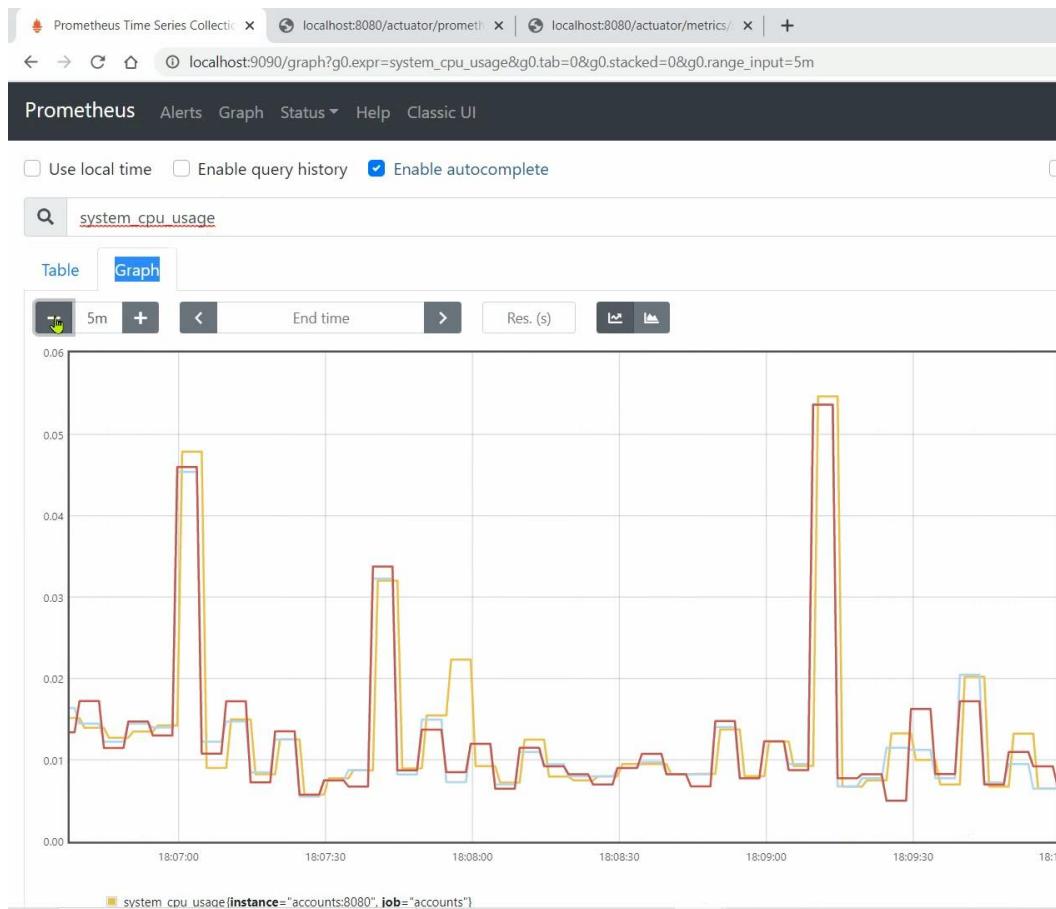
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://accounts:8080/actuator/prometheus	UP	instance="accounts:8080" job="accounts"	12.943s ago	5.309ms	

cards (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://cards:9000/actuator/prometheus	UP	instance="cards:9000" job="cards"	13.621s ago	5.069ms	

loans (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://loans:8090/actuator/prometheus	UP	instance="loans:8090" job="loans"	13.876s ago	5.161ms	



5. Setup of Grafana to monitor microservices with inbuilt dashboards

Chỉnh sửa docker-compose.yml với thêm nội dung sau:

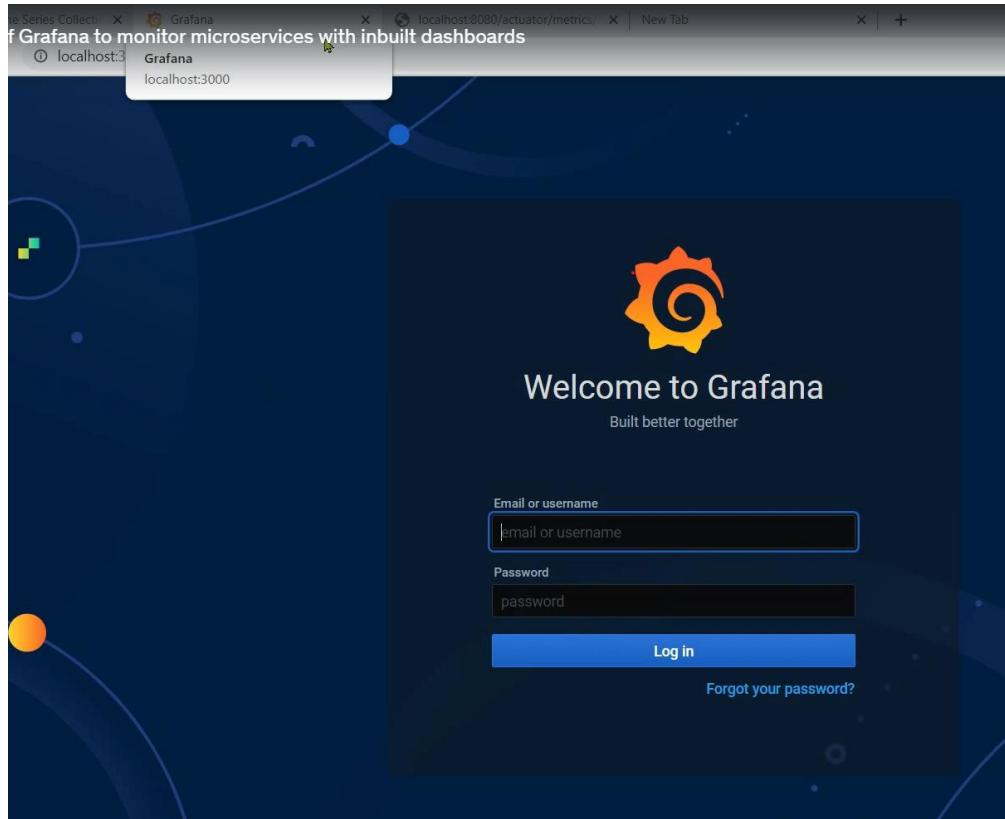
```
version: "3.8"

services:

  grafana:
    image: "grafana/grafana:latest"
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=password
    networks:
      - eazybank
    depends_on:
      - prometheus
```

Mở công cụ command line có chứa docker-compose.yml và chạy lệnh soạn thảo docker "**docker-compose up**" để khởi động tất cả các microservice bằng một lệnh duy nhất. Tất cả các container đang chạy có thể được xác thực bằng cách chạy lệnh docker "docker ps".

Mở URL <http://localhost:3000/login/> bên trong trình duyệt và nhập chi tiết đăng nhập (admin/password) của Grafana. Bên trong Grafana cung cấp thông tin chi tiết về prometheus, xây dựng bảng điều khiển tùy chỉnh, cảnh báo.



have no prior experience. This process and covers the "Data right.

DATA SOURCES
Add your first data source

 Learn how in the docs ↗

DASHBOARDS
Create your first dashboard

 Learn how in the docs ↗

Remove this

Latest from the blog

Monitor and alert on essential RabbitMQ cluster metrics with the new Grafana Cloud

Filter by name or type

Cancel

Time series databases

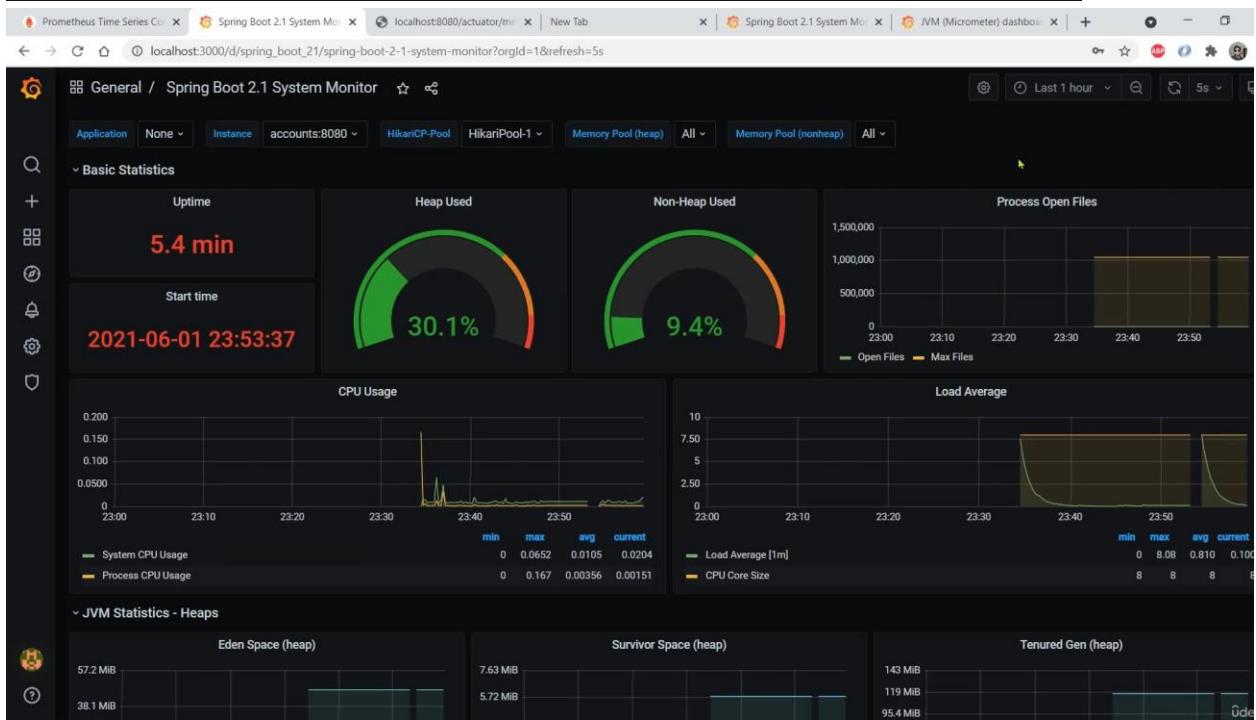
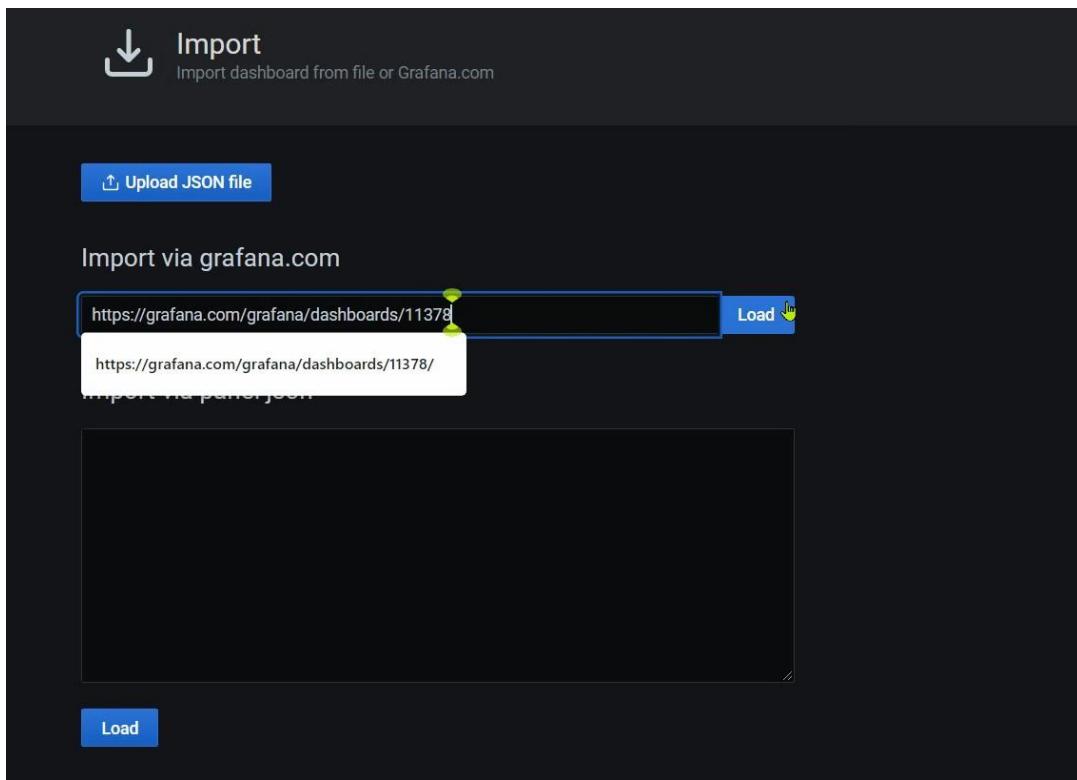
 Prometheus Open source time series database & alerting <input type="button" value="Core"/>	<input type="checkbox"/> Learn more <input type="button" value="Select"/>
 Graphite Open source time series database <input type="button" value="Core"/>	
 OpenTSDB Open source time series database <input type="button" value="Core"/>	
 InfluxDB Open source time series database <input type="button" value="Core"/>	

Configure your Prometheus data source below

Or skip the effort and get Prometheus (and Loki) as fully managed, scalable and hosted data sources from Grafana [Grafana Cloud plan](#).

Name	<input type="text" value="Prometheus"/>	Default	<input checked="" type="checkbox"/>
HTTP			
URL	<input type="text" value="http://prometheus:9090"/>		
Access	<input type="text" value="Server (default)"/>	Help >	
Whitelisted Cookies	<input type="text" value="New tag (enter key to add)"/>	Add	
Auth			
Basic auth	<input type="checkbox"/>	With Credentials	<input type="checkbox"/>
TLS Client Auth	<input type="checkbox"/>	With CA Cert	<input type="checkbox"/>
Skip TLS Verify	<input type="checkbox"/>		
Forward OAuth Identity	<input type="checkbox"/>		
Custom HTTP Headers			

The screenshot shows the Grafana interface. At the top, a dark blue header features the title "Welcome to Grafana". Below the header is a navigation bar with several icons: a magnifying glass for search, a plus sign for creating new items, a dashboard icon for "Dashboard", a folder icon for "Folder", and an import icon for "Import", which is highlighted with a yellow box and a cursor arrow. To the right of the import icon, a text box contains the message: "The steps below will guide you to quickly finish setting up your Grafana installation." Further down the page, there are sections titled "TUTORIAL", "DATA SOURCE AND DASHBOARDS", and "Grafana fundamentals", each with a corresponding orange button. A large orange gear icon is centered on the page. At the bottom, there are links for "Dashboards" and "Starred dashboards".

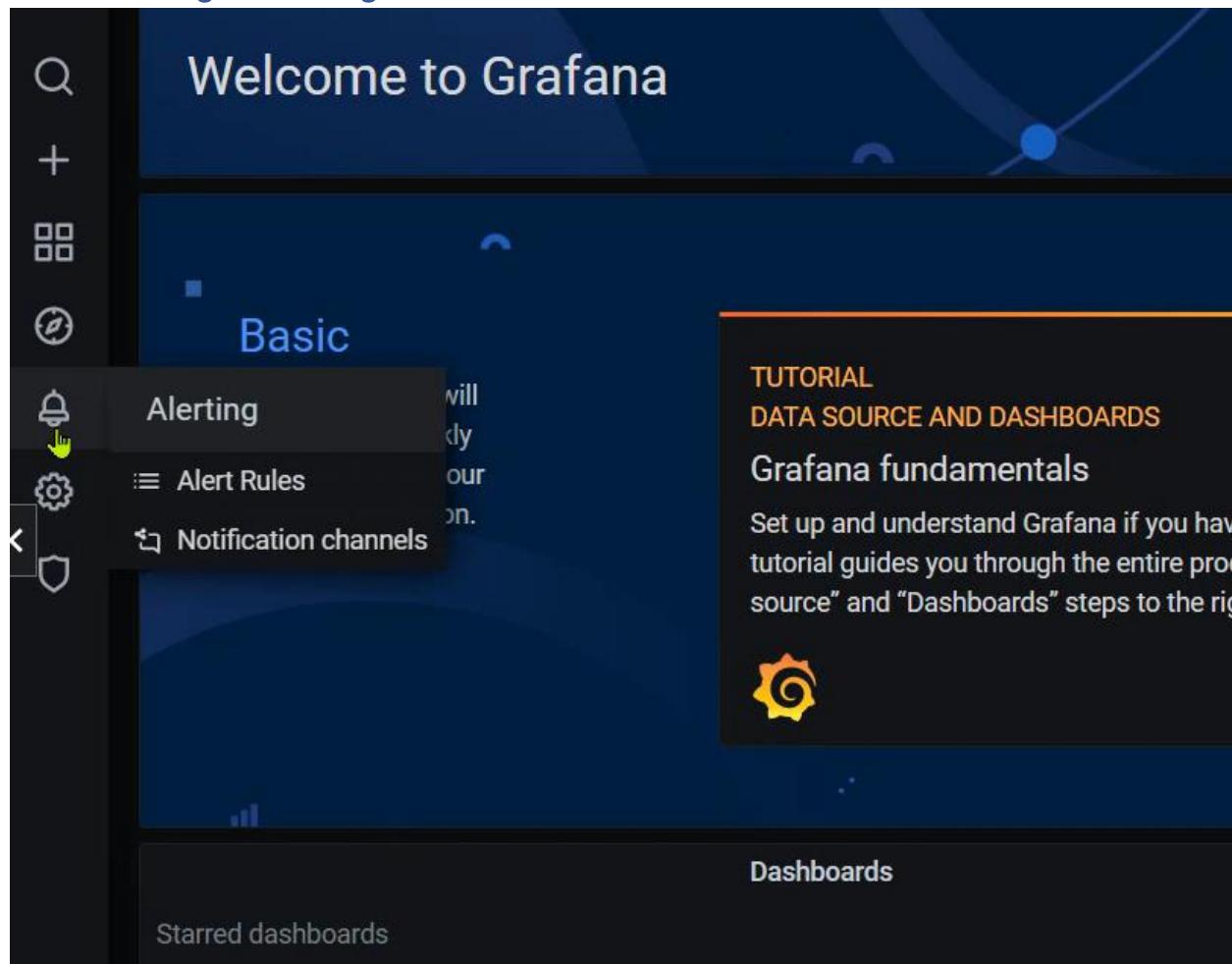


6. Building custom dashboards inside Grafana- xây dựng các bảng điều khiển tùy chỉnh trong Grafana

- Tạo một bảng điều khiển mới: Đăng nhập vào giao diện quản trị của Grafana và tạo một bảng điều khiển mới. Bạn có thể chọn loại biểu đồ mà bạn muốn hiển thị, như đồ thị dòng, đồ thị cột, đồ thị hình tròn, vv.
- Thiết kế bảng điều khiển: Tùy chỉnh bảng điều khiển của bạn bằng cách thêm các truy vấn và hiển thị các thông số bạn muốn giám sát. Bạn có thể chọn nguồn dữ liệu từ bước 2 và viết các truy vấn để truy xuất dữ liệu từ nguồn đó.
- Tùy chỉnh giao diện: Grafana cho phép bạn tùy chỉnh giao diện của bảng điều khiển, chẳng hạn như thay đổi màu sắc, kích thước, định dạng văn bản và hiển thị biểu đồ.
- Lưu và chia sẻ bảng điều khiển: Khi bạn hoàn thành việc tạo bảng điều khiển tùy chỉnh, hãy lưu nó lại và chia sẻ với người dùng khác nếu cần thiết.

Có rất nhiều tùy chọn và tính năng trong Grafana để tùy chỉnh bảng điều khiển của bạn theo nhu cầu cụ thể của từng dự án. Bạn có thể khám phá thêm trong tài liệu hướng dẫn của Grafana hoặc cộng đồng người dùng để tìm hiểu thêm về các tính năng và phương pháp tùy chỉnh khác nhau.

7. Sending alerts using Grafana when service is down





Alerting

Alert rules & notifications

≡ Alert Rules

Notification channels

New notification channel

Name

Type

webhook

Url

Optional Webhook settings >

Notification settings >

Save

Test

Back

Cần tạo 1 webhook:

The screenshot shows the Pipedream Request Bin interface. At the top, it displays the URL <https://enr93rip67qbf.x.pipedream.net/>. Below the URL, there's a message saying "Waiting for first request...". A "Generate Test Events" button is present, along with a list of supported languages: CURL, JAVASCRIPT, NODE, PYTHON 2, PYTHON 3, PHP, GO, and RUBY. A code snippet for generating a test event using cURL is shown:

```
curl -d '{ "name": "Han solo" }' \
-H "Content-Type: application/json" \
https://enr93rip67qbf.x.pipedream.net/
```

At the bottom, there's a "NEW" section with the heading "Preview the new Request Bin — now with an API and more sources!" and a "Create HTTP Source" button.

The screenshot shows the Alerting interface. The title bar says "Alerting" and "Alert rules & notifications". There are two tabs: "Alert Rules" and "Notification channels", with "Notification channels" being active. The main area is titled "New notification channel". It contains fields for "Name" (empty), "Type" (set to "webhook"), and "Url" (set to <https://enr93rip67qbf.x.pipedream.net/>). Below these fields is a section titled "Optional Webhook settings" with a "Save" button highlighted with a yellow arrow. Other buttons include "Test" and "Back".

Khi 1 microservice bị shutdown

The screenshot shows the Pipedream platform interface. At the top, there's a header with 'pipedream' and a sign-in link. Below the header, there's a navigation bar with 'Untitled' and a 'public' dropdown. The main area has tabs for 'LIVE' and 'PAUSE'. A search bar says 'Type to search...'. On the left, a sidebar shows 'Today' with two log entries: '12:29:23 AM POST /' and '12:21:00 AM POST /'. The right side is a large panel titled 'Headers (6) headers' with tabs for 'RAW', 'PRETTY', and 'STRUCTURED'. The 'STRUCTURED' tab is selected, displaying a nested JSON object. The JSON structure includes fields like 'root', 'dashboardId', 'evalMatches', 'tags', and 'message'. The 'message' field contains the text 'Alarm ! Your Accounts Service is down.' Below this panel, a banner says 'NEW Preview the new Request Bin — now with an API and more sources!'. The overall interface is clean and modern, designed for managing and visualizing API requests.

=> Có thể sử dụng thông báo bằng mail

Section 11: Automatic self-healing, scaling, deployments using Kubernetes- Tự động phục hồi, mở rộng quy mô, triển khai bằng Kubernetes

1. Introduction to the challenges related to container orchestration- Giới thiệu về những thách thức liên quan đến điều phối container

HOW DO WE AUTOMATE THE DEPLOYMENTS, ROLLOUTS & ROLLBACKS?



Làm cách nào để chúng tôi tự động hóa việc triển khai các container thành một cluster env phức tạp và thực hiện triển khai các phiên bản mới của container mà không mất thời gian ngừng hoạt động cùng với tùy chọn khôi phục tự động trong trường hợp có bất kỳ sự cố nào?

HOW DO WE MAKE SURE OUR SERVICES ARE SELF-HEALING?



Làm thế nào để chúng tôi đảm bảo dịch vụ của chúng tôi là tự chữa lành?
Làm cách nào để chúng tôi tự động khởi động lại các container bị lỗi, thay thế các container, hủy các container không phản hồi kiểm tra tình trạng do người dùng xác định và không quảng cáo các container đó cho khách hàng cho đến khi chúng sẵn sàng.

HOW DO WE AUTO SCALE OUR SERVICES ?



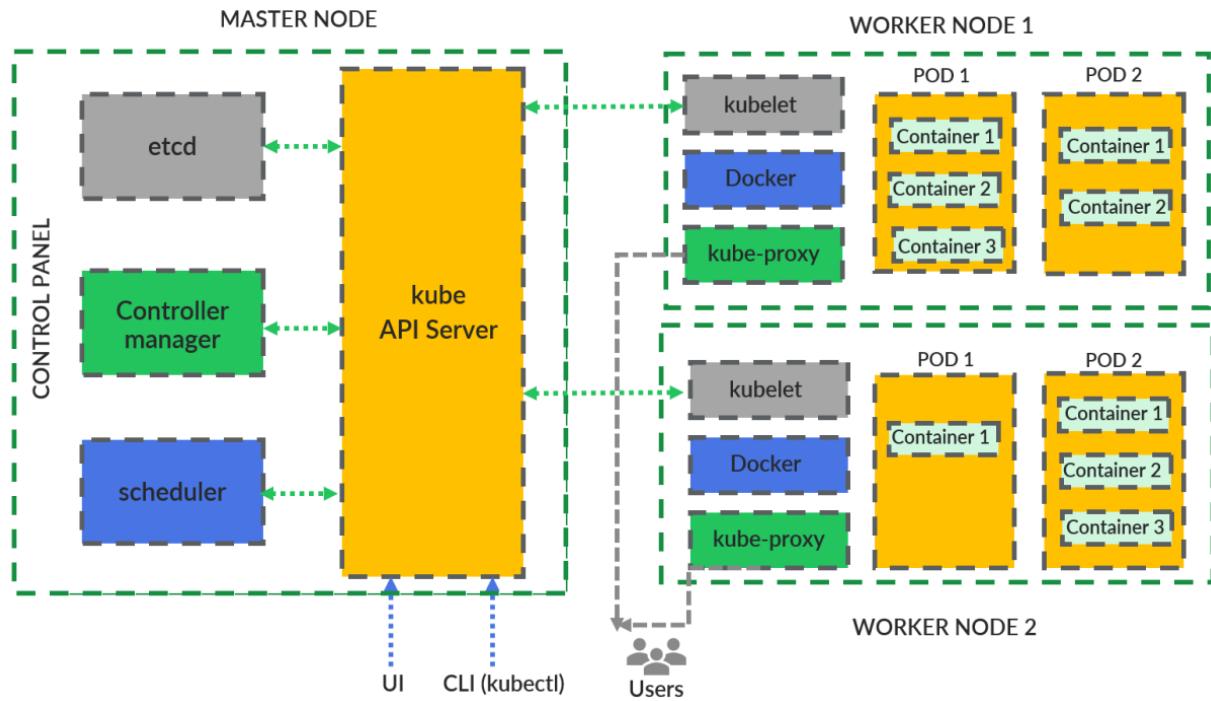
Làm cách nào để chúng tôi giám sát các dịch vụ của mình và tự động mở rộng quy mô dựa trên các số liệu như CPU Utilization, v.v.?



2. Introduction to Kubernetes

- Kubernetes, là một hệ thống mã nguồn mở để tự động triển khai, mở rộng quy mô và quản lý các ứng dụng được chứa trong container. Đây là nền tảng điều phối nổi tiếng nhất và nó là cloud neutral (đám mây trung lập).
- Cái tên Kubernetes bắt nguồn từ tiếng Hy Lạp, có nghĩa là người lái xe hoặc phi công. K8 là kết quả viết tắt của việc đếm tám chữ cái giữa chữ "K" và chữ "s".
- Google đã cung cấp mã nguồn mở cho dự án Kubernetes vào năm 2014. Kubernetes kết hợp hơn 15 năm kinh nghiệm của Google khi chạy khối lượng công việc sản xuất trên quy mô lớn với các ý tưởng và phương pháp hay nhất từ cộng đồng.
- Kubernetes cung cấp cho bạn một framework để chạy các hệ thống phân tán một cách linh hoạt. Nó đảm nhiệm việc mở rộng quy mô và chuyển đổi dự phòng cho ứng dụng của bạn, cung cấp các mẫu triển khai, v.v. Nó cung cấp cho bạn:
 - ✓ Service discovery and load balancing
 - ✓ Storage orchestration - phối hợp lưu trữ
 - ✓ Automated rollouts(giới thiệu) and rollbacks
 - ✓ Automatic bin packing (tự động đóng gói)
 - ✓ Self-healing
 - ✓ Secret and configuration management

3. Dee dive of Kubernetes internal architecture



Control Plane(Master Node)

Control Plane(Master Node) chính chịu trách nhiệm quản lý toàn bộ cluster. Nó giám sát, kiểm tra sức khỏe của tất cả các node trong cụm cluster lưu trữ thông tin của các thành viên liên quan đến các node khác nhau, lập kế hoạch cho các vùng chưa được lên lịch cho các node nhất định, giám sát các container và node, v.v. Vì vậy, khi một node bị lỗi, **Master Node** chính sẽ di chuyển khối lượng công việc từ node bị lỗi sang node khỏe mạnh khác.

Kubernetes master chịu trách nhiệm lên lịch, cung cấp, định cấu hình và hiển thị API cho client. Vì vậy, tất cả những điều này được thực hiện bởi một **Master Node** sử dụng các thành phần của **control plane**.

Kubernetes đảm nhiệm việc khám phá dịch vụ, mở rộng quy mô, cân bằng tải, tự phục hồi leader election, v.v. Do đó, các nhà phát triển không còn phải xây dựng các dịch vụ này bên trong ứng dụng.

Nó bao gồm một số thành phần:

- **etcd:** một kho lưu trữ key-value phân tán lưu trữ dữ liệu cấu hình và trạng thái của toàn bộ cụm. Tất cả các thành phần Kubernetes đọc và ghi vào etcd để duy trì tính nhất quán.
- **kube-apiserver:** máy chủ API Kubernetes cung cấp API Kubernetes cho người dùng và các thành phần khác. Nó xác minh và xử lý các yêu cầu API, cập nhật etcd và giao tiếp với các thành phần khác để duy trì trạng thái mong muốn của cụm.
- **kube-scheduler:** thành phần lập lịch pod (đơn vị nhỏ nhất có thể triển khai trong Kubernetes) tới các node dựa trên yêu cầu tài nguyên, các quy tắc liên quan/phi liên quan và các ràng buộc khác.
- **kube-controller-manager:** một bộ điều khiển giám sát trạng thái của cụm, phát hiện và phản hồi các thay đổi và phù hợp với trạng thái mong muốn với trạng thái thực tế. Các bộ điều khiển bao gồm bộ điều khiển node, bộ điều khiển sao chép, bộ điều khiển điểm cuối (endpoint controller) và bộ điều khiển tài khoản dịch vụ (service account controller).

Nodes

Worker node không gì khác ngoài một máy ảo (VM) chạy trên đám mây hoặc tại chỗ (một máy chủ vật lý chạy bên trong trung tâm dữ liệu). Vì vậy, bất kỳ phần cứng nào có khả năng chạy container runtime đều có thể trở thành **Worker node**. Các **node** này hiển thị khả năng tính toán, lưu trữ và kết nối mạng cơ bản cho các ứng dụng.

Các worker node thực hiện công việc nặng nhọc cho ứng dụng chạy bên trong cụm Kubernetes. Cùng với nhau, các **node** này tạo thành một **cluster** - **master node** chính sẽ phân công khối lượng công việc cho chúng, tương tự như cách người quản lý giao nhiệm vụ cho thành viên nhóm. Bằng cách này, sẽ có thể đạt được khả năng chịu lỗi và sao chép.

Pods là đơn vị triển khai nhỏ nhất trong Kubernetes cũng như container là đơn vị triển khai nhỏ nhất trong Docker. Nói một cách dễ hiểu, chúng ta có thể nói rằng pod chẳng qua là những VM nhẹ trong thế giới ảo. Mỗi pod bao gồm một hoặc nhiều container. Các pod có bản chất phù du khi chúng đến và đi, trong khi các container có bản chất không trạng thái. Thông thường, chúng tôi chạy một container bên trong một pod. Có một số trường hợp chúng tôi sẽ chạy nhiều container phụ thuộc vào nhau trong một pod. Mỗi khi một pod quay lên(spins up), nó sẽ nhận được một địa chỉ IP mới với dải IP ảo được chỉ định bởi giải pháp kết nối mạng pod(nhóm).

Chúng bao gồm một số thành phần:

- **kubelet**: chương trình chính chạy trên mỗi node và giao tiếp với Control Plane. Nó có trách nhiệm khởi động và dừng container, giám sát tình trạng của chúng và báo cáo lại cho Control Plane.
- **kube-proxy**: kube-proxy là một proxy mạng chạy trên mỗi node trong cluster của bạn, triển khai một phần của khái niệm Dịch vụ Kubernetes. kube-proxy duy trì các quy tắc mạng trên các node. Các quy tắc mạng này cho phép giao tiếp mạng với các Pods của bạn từ các phiên mạng bên trong hoặc bên ngoài lúter của bạn.
- **container runtime**: phần mềm chạy các container. Kubernetes hỗ trợ nhiều runtime, bao gồm Docker, containerd và CRI-O.

Networking

Kubernetes cung cấp mô hình mạng phẳng cho các pod, trong đó mỗi pod có địa chỉ IP riêng của nó. Điều này được đạt được thông qua mô hình mạng Kubernetes, bao gồm một số thành phần:

- **CNI (Container Network Interface)**: một phần mở rộng cho các plugin mạng để tích hợp với Kubernetes. Các plugin CNI được trách nhiệm tạo và quản lý các giao diện mạng của các pod.
- **kube-proxy**: như đã đề cập ở trên, kube-proxy quản lý kết nối mạng giữa các pod và dịch vụ.
- **Service**: một trừu tượng định nghĩa một tập hợp logic của các pod và một chính sách để truy cập chúng. Dịch vụ cung cấp một địa chỉ IP và tên DNS ổn định để truy cập các pod.

Storage

- Kubernetes cung cấp một số tùy chọn cho lưu trữ liên tục (persistent storage), bao gồm:
- **Persistent Volumes (PV)**: một trừu tượng lưu trữ tách biệt cấu hình lưu trữ từ đặc tả pod. PV có thể được cấu hình động bởi các lớp lưu trữ (storage classes) hoặc được cấu hình tĩnh bởi quản trị viên.
- **Persistent Volume Claims (PVC)**: yêu cầu lưu trữ từ một pod. PVC có thể được liên kết động với một PV phù hợp bởi Kubernetes.

- Storage Classes: cách để xác định các lớp lưu trữ khác nhau với các đặc tính hiệu suất và khả năng sẵn có khác nhau. Quản trị viên có thể xác định các lớp lưu trữ và ánh xạ chúng với các PV cụ thể.

Kết luận

Kiến trúc nội bộ của Kubernetes là phức tạp nhưng rất linh hoạt và có thể mở rộng. Control Plane quản lý trạng thái tổng thể của cụm, trong khi các node công việc chạy các ứng dụng được đóng gói trong container. Mô hình mạng và lưu trữ cung cấp một nền tảng linh hoạt và có thể mở rộng để triển khai và quản lý các ứng dụng được đóng gói trong container ở quy mô lớn.

4. Cloud providers support for Kubernetes

- Kubernetes có tính mô-đun, linh hoạt và có thể mở rộng đến mức nó có thể được triển khai tại chỗ, trong trung tâm dữ liệu của bên thứ ba, trong bất kỳ nhà cung cấp cloud phổ biến nào và thậm chí trên nhiều nhà cung cấp cloud .
- Việc tạo và duy trì cụm K8S có thể rất khó khăn tại chỗ. Do đó, nhiều doanh nghiệp tìm kiếm các nhà cung cấp đám mây sẽ giúp công việc của họ dễ dàng duy trì kiến trúc microservice của họ bằng Kubernetes. Dưới đây là các nhà cung cấp đám mây nổi tiếng khác nhau và sự hỗ trợ của họ đối với Kubernetes với các tên khác nhau:
 - ✓ Google Cloud Platform (GCP): Google Kubernetes Engine (GKE).
 - ✓ Amazon Web Services (AWS): Amazon Elastic Kubernetes Service (Amazon EKS).
 - ✓ Microsoft Azure: Azure Kubernetes Service (AKS).

5. GCP Account Setup and creating a K8s cluster

The screenshot shows the Google Cloud Platform's 'Free Tier products' page. At the top, there is a navigation bar with links for Why Google, Solutions, Products, Pricing, Getting Started, Contact Us, Docs, Support, English, and Help. Below the navigation bar, the title 'Free Tier products' is centered. There are four cards representing different services:

- Compute Engine**: Scalable, high-performance virtual machines. 1 F1-micro instance per month.
- Cloud Storage**: Best-in-class performance, reliability, and pricing for all your storage needs. 5 GB-months Standard Storage.
- BigQuery**: Fully managed, petabyte scale, analytics data warehouse. 1 TB queries per month.
- Google Kubernetes Engine**: One-click container orchestration via Kubernetes clusters, managed by Google. One Autopilot or Zonal cluster per month.

A green arrow points to the 'Google Kubernetes Engine' card.

- Sử dụng qua Goole Cloud Platform Console

The screenshot shows the Google Cloud Platform console interface. At the top, there are three tabs: "Home – Microservices – Google" (active), "Free Trial and Free Tier | Google", and "Quickstart: Getting started with GCP". Below the tabs, the URL is "console.cloud.google.com/home/dashboard?project=microservices-316111&_ga=2.28704766.391089178.1624103975-2001831314.1623065506". The main navigation bar includes "Google Cloud Platform" and "Microservices" dropdowns, and "DASHBOARD", "ACTIVITY", and "RECOMMENDATIONS" buttons.

A search bar at the top right contains the text "kuber". A sidebar on the left titled "Project info" displays project details: Project name (Microservices), Project ID (microservices-316111), and Project number (523929512636). Below this is a "ADD PEOPLE TO THIS PROJECT" button and a link to "Go to project settings".

The main content area shows search results for "kubernetes". The results are grouped under "PRODUCTS & PAGES" and "RESOURCES". Under "PRODUCTS & PAGES", there are four items: "Kubernetes Engine" (with sub-options "Clusters", "Applications", and "Configuration"), "Kubernetes clusters" (under "Kubernetes Engine"), and "Firewall" (under "Kubernetes Engine"). Under "RESOURCES", there are three items: "k8s-75cdc83bfc4340f9-node-http-hc" (under "Firewall – microservices-316111"), "k8s-fw-a5e7a466533474cb79b7fff2aa543673" (under "Firewall – microservices-316111"), and "k8s-fw-a7d462e9c04554f79b49d70a36b33f3c" (under "Firewall – microservices-316111").

A modal window titled "Create cluster" is open in the foreground. It asks "Select the cluster mode that you want to use." It provides two options: "Standard" and "Autopilot". The "Standard" section describes it as a "Kubernetes cluster with node configuration flexibility and pay-per-node." It includes a "Learn more" link and a large blue "CONFIGURE" button with a yellow hand cursor icon pointing to it. The "Autopilot" section describes it as an "Optimised Kubernetes cluster with a hands-off experience and pay-per-pod." It also includes a "Learn more" link and a blue "CONFIGURE" button. At the bottom right of the modal is a "CANCEL" button.

A cluster designed for

Name cluster-1 ?

Location type Zonal Regional

Zone us-central1-c ▼ ?

Specify default node locations ?
Current default: us-central1-c

Control plane version

Choose Release Channel to get automatic GKE upgrades as new versions are ready. Choose a static version to upgrade manually in the future. [Learn more](#).

Static version Release channel

Release channel Regular channel (default) ▼

Version 1.19.10-gke.1600 (default) ▼

These versions have passed internal validation and are considered production quality but don't have enough historical data to guarantee their stability. Known issues generally have known workarounds. [Release notes](#)

CREATE CANCEL Equivalent [REST](#) or [COMMAND LINE](#)

Kubernetes clusters		+CREATE	+DEPLOY	REFRESH	DELETE	OPERATIONS ▾	HIDE INFO PANEL	LEARN
Filter Enter property name or value								
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Name ▲	Location	Number of nodes	Total vCPUs	Total memory	Notifications	? ☰
<input type="checkbox"/>	<input checked="" type="checkbox"/>	cluster-1	us-central1-c	3	6	12 GB		? No clusters selected

6. Exploring K8S cluster and establish connection with it- Khám phá cụm K8S và thiết lập kết nối

- Danh sách các node

Name	Status	CPU requested	CPU allocatable	Memory requested	Memory allocatable	Storage requested
gke-cluster-1-default-pool-e5bfde1d-c2h0	Ready	463 mCPU	940 mCPU	377.49 MB	2.96 GB	0 B
gke-cluster-1-default-pool-e5bfde1d-mltz	Ready	349 mCPU	940 mCPU	584.06 MB	2.96 GB	0 B
gke-cluster-1-default-pool-e5bfde1d-scn6	Ready	493 mCPU	940 mCPU	408.94 MB	2.96 GB	0 B

- Note detail (Enable: Cloud Logging API để hiện thị log và Stackdriver Monitoring API để thu thập và theo dõi dữ liệu liên quan đến hoạt động và hiệu suất của các tài nguyên)

Name	Status	CPU requested	Memory requested	Storage requested	Namespace	Restarts	Created on
fluentbit-gke-977gw	Running	100 mCPU	209.72 MB	0 B	kube-system	0	19 Jun 2021, 21:11:34
ndes-node-flrnq	Running	0 CPU	0 B	0 B	kube-system	0	10 Jun 2021

Thiết lập kết nối đến cụm Kubernetes:

The screenshot shows the Google Cloud Platform Microservices interface. At the top, there's a search bar with the text 'kube'. Below it, a navigation bar has 'Microservices' selected. Underneath, a table lists 'Kubernetes clusters'. The first cluster entry is 'cluster-1' located in 'us-central1-c' with 3 nodes, 6 vCPUs, and 12 GB of memory. To the right of the cluster table is a context menu with three options: 'Edit', 'Connect', and 'Delete'. A yellow arrow points to the 'Connect' option.

- Copy command – line access

This screenshot shows a 'Connect to the cluster' dialog box. It includes two main sections: 'Command-line access' and 'Cloud Console dashboard'. The 'Command-line access' section provides a command-line configuration step and a 'RUN IN CLOUD SHELL' button. The 'Cloud Console dashboard' section provides a link to view workloads. At the bottom right of the dialog is an 'OK' button.

Connect to the cluster

You can connect to your cluster via command-line or using a dashboard.

Command-line access

Configure [kubectl](#) command line access by running the following command:

```
$ gcloud container clusters get-credentials cluster-1 --zone us-central1-c --project microservices-316111
```

RUN IN CLOUD SHELL

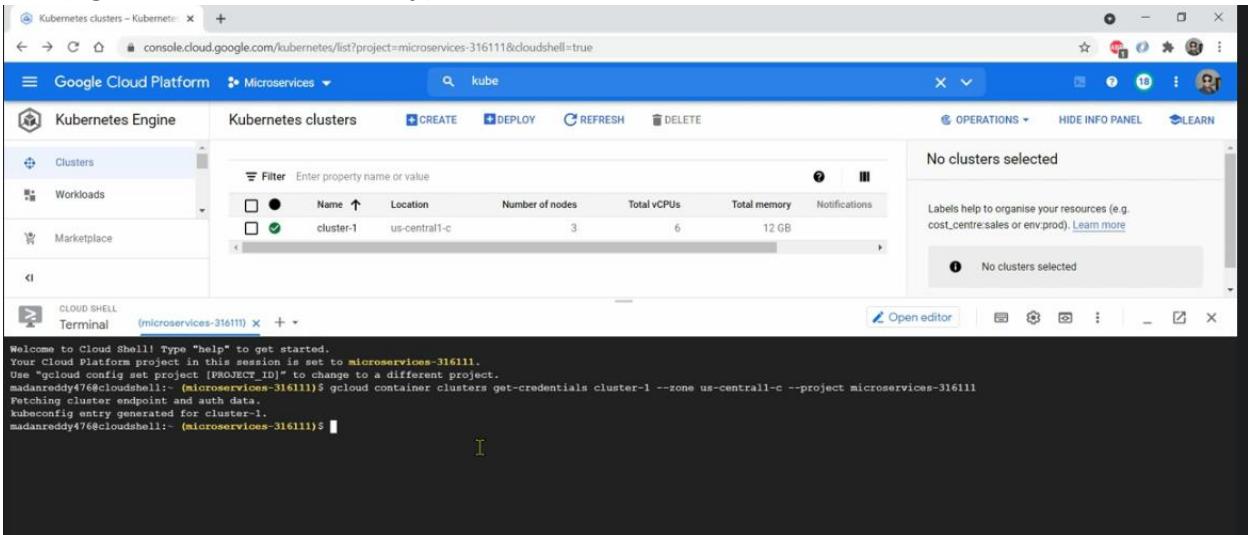
Cloud Console dashboard

You can view the workloads running in your cluster in the Cloud Console [Workloads dashboard](#).

[OPEN WORKLOADS DASHBOARD](#)

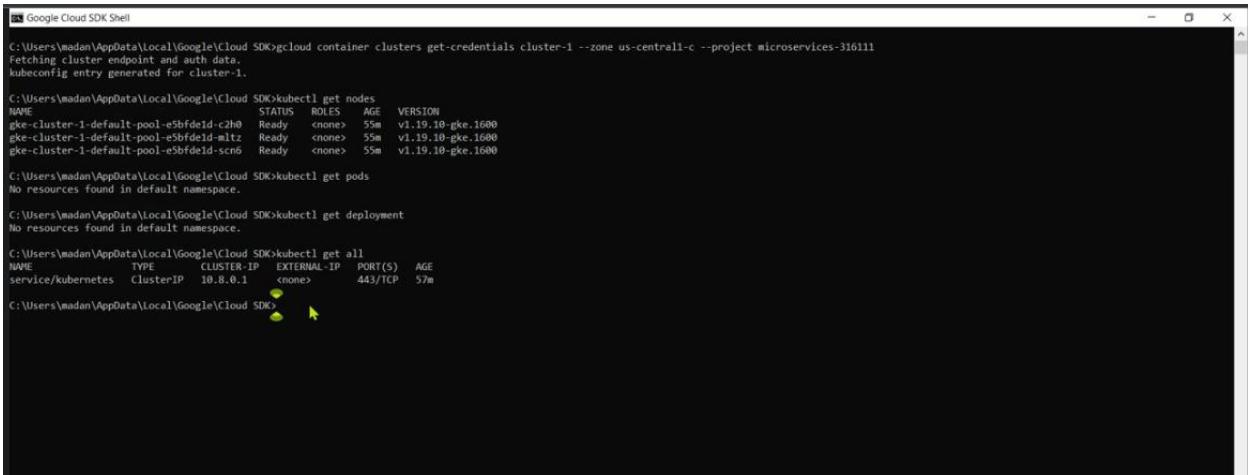
OK

Sử dụng Cloud Shell trên trình duyệt



The screenshot shows the Google Cloud Platform interface for Kubernetes Engine. On the left, there's a sidebar with 'Clusters', 'Workloads', and 'Marketplace'. The main area is titled 'Kubernetes clusters' with buttons for 'CREATE', 'DEPLOY', 'REFRESH', and 'DELETE'. A search bar at the top right says 'kube'. Below the buttons is a table with columns: Name, Location, Number of nodes, Total vCPUs, Total memory, and Notifications. One row is visible for 'cluster-1' located in 'us-central1-c' with 3 nodes, 6 vCPUs, and 12 GB memory. To the right of the table is a panel titled 'No clusters selected' with a note about labels. At the bottom, a terminal window titled '(microservices-316111)' shows a welcome message and some command-line output related to cluster configuration.

Sử dụng Google Cloud SDK:



The screenshot shows a Windows command-line interface (cmd) window titled 'Google Cloud SDK'. It displays several commands related to Kubernetes clusters:

```
C:\Users\madan\AppData\Local\Google\Cloud SDK>gcloud container clusters get-credentials cluster-1 --zone us-central1-c --project microservices-316111
Fetching cluster endpoint and auth data.
kubecfg entry generated for cluster-1.

C:\Users\madan\AppData\Local\Google\Cloud SDK>kubectl get nodes
NAME           STATUS   ROLES   AGE    VERSION
gke-cluster-1-default-pool-e5bfde1d-c2h0  Ready    <none>   55m   v1.19.10-gke.1600
gke-cluster-1-default-pool-e5bfde1d-mltz  Ready    <none>   55m   v1.19.10-gke.1600
gke-cluster-1-default-pool-e5bfde1d-scn6  Ready    <none>   55m   v1.19.10-gke.1600

C:\Users\madan\AppData\Local\Google\Cloud SDK>kubectl get pods
No resources found in default namespace.

C:\Users\madan\AppData\Local\Google\Cloud SDK>kubectl get deployment
No resources found in default namespace.

C:\Users\madan\AppData\Local\Google\Cloud SDK>kubectl get all
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.8.0.1       <none>        443/TCP   57m
```

7. Deep dive on Kubernetes YAML configurations

Một tệp YAML cơ bản bao gồm các cặp key-value (khóa-giá trị), và tất cả các cặp này nằm trong cấu trúc thụt lề (indentation) để xác định mối quan hệ giữa chúng.

Ví dụ đơn giản về cấu hình Pod trong YAML(tương tự cấu hình cho configserver, microservice, eurekaserver):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: zipkin-deployment
  labels:
    app: zipkin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: zipkin
  template:
    metadata:
      labels:
        app: zipkin
    spec:
      containers:
        - name: zipkin
          image: openzipkin/zipkin
          ports:
            - containerPort: 9411
apiVersion: v1
kind: Service
metadata:
  name: zipkin-service
spec:
  selector:
    app: zipkin
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 9411
      targetPort: 9411
```

Tệp cấu hình trên là một ví dụ về YAML để triển khai ứng dụng Zipkin trên Kubernetes sử dụng một Deployment và một Service.

Phần Deployment:

- apiVersion: apps/v1: Xác định phiên bản API của Kubernetes cho tài nguyên Deployment.
- kind: Deployment: Định nghĩa loại tài nguyên là Deployment, được sử dụng để triển khai ứng dụng trên Kubernetes.
- metadata: Chứa thông tin về tài nguyên, bao gồm tên và nhãn (labels). Trong trường này, tên Deployment được đặt là "zipkin-deployment" và có một nhãn được gán là "app: zipkin".
- spec: Xác định cấu hình và cài đặt của tài nguyên Deployment.
 - replicas: 1: Xác định số lượng replica (bản sao) của Pods mà Deployment sẽ triển khai, trong trường này là 1 replica.
 - selector: Định nghĩa bộ chọn Pods mà Deployment sẽ quản lý. Trong trường này, nó sử dụng nhãn "app: zipkin" để chọn các Pods.
 - template: Xác định template cho các Pods mà Deployment triển khai.
 - ✓ metadata: Chứa các nhãn của Pods. Trong trường này, nó sử dụng nhãn "app: zipkin" cho các Pods.
 - ✓ spec: Xác định cấu hình của các containers trong Pods.
 - containers: Định nghĩa danh sách các containers trong Pod.
 - ◆ name: zipkin: Đặt tên cho container là "zipkin".

- image: openzipkin/zipkin: Xác định hình ảnh container, trong trường này sử dụng hình ảnh "openzipkin/zipkin".
- ports: Định nghĩa cổng mà container lắng nghe.
 - containerPort: 9411: Xác định cổng 9411 mà container lắng nghe.

Phần Service:

- apiVersion: v1: Xác định phiên bản API của Kubernetes cho tài nguyên Service.
- kind: Service: Định nghĩa loại tài nguyên là Service, được sử dụng để tạo một cơ chế định tuyến và cung cấp truy cập vào các Pods.
- metadata: Chứa thông tin về tài nguyên, bao gồm tên. Trong trường này, tên Service được đặt là "zipkin-service".
- spec: Xác định cấu hình và cài đặt của tài nguyên Service.
 - selector: Định nghĩa bộ chọn Pods mà Service sẽ định tuyến yêu cầu tới. Trong trường này, nó sử dụng nhãn "app: zipkin" để định tuyến đến các Pods có nhãn tương ứng.
 - type: LoadBalancer: Xác định kiểu dịch vụ là LoadBalancer, cho phép Kubernetes tự động tạo một Load Balancer trên môi trường đám mây công cộng như GCP, AWS để cung cấp truy cập từ bên ngoài vào dịch vụ.
 - ports: Định nghĩa cổng mà Service sẽ lắng nghe.
 - ✓ protocol: TCP: Xác định giao thức sử dụng cho cổng, trong trường này là TCP.
 - ✓ port: 9411: Xác định cổng 9411 của Service.
 - ✓ targetPort: 9411: Xác định cổng 9411 của Pods mà Service sẽ định tuyến yêu cầu tới.

Phần "Deployment" cấu hình cho ứng dụng Zipkin. Nó chỉ định số lượng bản sao (replicas) là 1 và chọn các pods có nhãn "app: zipkin". Nó cũng chỉ định hình ảnh (image) của container là "openzipkin/zipkin", đây là hình ảnh chính thức của Zipkin được cung cấp trên Docker Hub, và cung cấp cổng (port) 9411.

Phần "Service" định nghĩa một dịch vụ Kubernetes để phục vụ cho ứng dụng Zipkin. Nó thiết lập loại dịch vụ là LoadBalancer, tạo ra một bộ cân bằng tải bên ngoài để phân phối lưu lượng truy cập đến các pods Zipkin, và chỉ định cổng đích (target port) là 9411. Selector khớp với nhãn (label) của deployment Zipkin, đảm bảo rằng dịch vụ định tuyến lưu lượng truy cập đến các pods đúng.

Tổng quan, khi áp dụng tệp cấu hình này vào một cụm Kubernetes, nó sẽ tạo ra một bản sao của deployment Zipkin, và phục vụ nó thông qua dịch vụ LoadBalancer trên cổng 9411, từ đó có thể truy cập vào bảng điều khiển Zipkin và xem dữ liệu theo dõi phân tán.

8. Create environment variables inside K8S cluster using ConfigMap

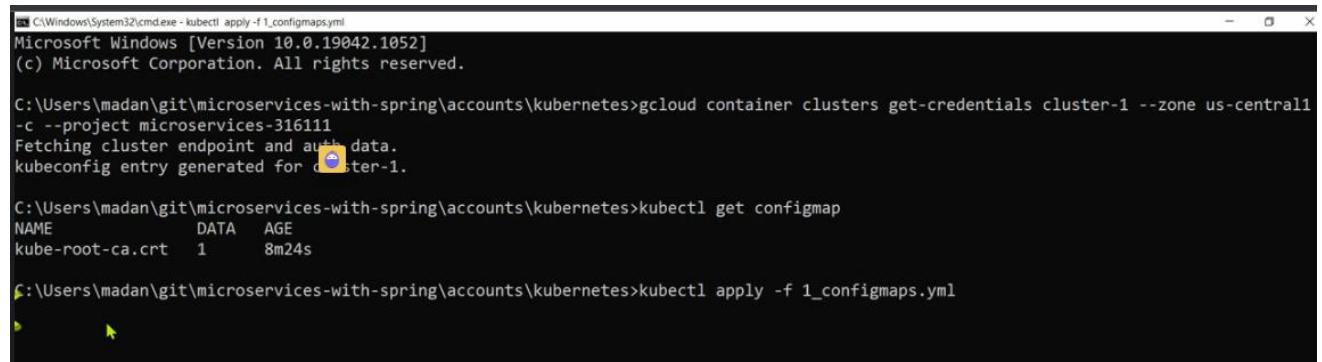
Để tạo biến môi trường trong cụm Kubernetes bằng cách sử dụng ConfigMap, bạn có thể làm như sau:

Bước 1: Tạo một tệp YAML để định nghĩa ConfigMap. Ví dụ, tạo một tệp có tên là configmap.yaml với nội dung như sau:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: eazybank-configmap
data:
  # SPRING_ZIPKIN_BASEURL: http://zipkin-service:9411/
  MANAGEMENT_ZIPKIN_TRACING_ENDPOINT: http://zipkin-service:9411/api/v2/spans
  SPRING_PROFILES_ACTIVE: prod
  SPRING_CONFIG_IMPORT: configserver:http://configserver-service:8071/
  EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://eurekaserver-
service:8070/eureka/
  EUREKA_APPLICATION_NAME: eurekaserver
  ACCOUNTS_APPLICATION_NAME: accounts
```

Bước 2: Triển khai ConfigMap vào cụm Kubernetes bằng lệnh kubectl:

```
kubectl apply -f configmap.yaml
```



The screenshot shows a Windows Command Prompt window with the following command history:

```
C:\Windows\System32\cmd.exe - kubectl apply -f 1_configmaps.yaml
Microsoft Windows [Version 10.0.19042.1052]
(c) Microsoft Corporation. All rights reserved.

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>gcloud container clusters get-credentials cluster-1 --zone us-central1
-c --project microservices-316111
Fetching cluster endpoint and auth data.
kubeconfig entry generated for cluster-1.

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get configmap
NAME        DATA   AGE
kube-root-ca.crt   1    8m24s

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl apply -f 1_configmaps.yaml
▶
```

Google Cloud Platform Microservices Search products and resources

Kubernetes Engine	Configuration	REFRESH	DELETE	
Clusters	Cluster	Namespace	RESET SAVE	
Workloads	Secrets are sensitive pieces of information, such as passwords, keys and tokens. ConfigMaps are designed to store information that is not sensitive, such as environment variables, command-line arguments and configuration files.			
Services & Ingress	Secrets respect access control and are not visible to users without read permissions			
Applications				
Configuration	Filter Is system object : False Filter secrets and config maps			
Storage	Name ↑	Type	Namespace	
Object Browser			Cluster	
Migrate to containers	<input type="checkbox"/> default-token-sm6ph	Secret	kube-node-lease	cluster-1
	<input type="checkbox"/> eazybank-configmap	Config Map	default	cluster-1
	<input type="checkbox"/> kube-root-ca.crt	Config Map	kube-node-lease	cluster-1
	<input type="checkbox"/> kube-root-ca.crt	Config Map	default	cluster-1
	<input type="checkbox"/> kube-root-ca.crt	Config Map	kube-public	cluster-1

← Config map details REFRESH EDIT DELETE KUBECTL

eazybank-configmap

DETAILS	YAML
Cluster	cluster-1
Namespace	default
Created	20 Jun 2021, 10:22:24
Labels	No labels set
Annotations	SHOW ANNOTATIONS

Data

EUREKA_CLIENT_SERVICEURL_DEFAULTZONE	http://eurekaserver-service:8070/eureka/
SPRING_CONFIG_IMPORT	configserver:http://configserver-service:8071/
SPRING_PROFILES_ACTIVE	prod
SPRINGZIPKIN_BASEURL	http://zipkin-service:9411/

9. Deploying our microservices to Kubernetes cluster

Chạy các lệnh:

```
kubectl apply -f 2_zipkin.yml  
kubectl apply -f 3_configserver.yml  
kubectl apply -f 4_eurekaserver.yml  
kubectl apply -f 5_accounts.yml
```

Để triển khai các microservice hiện hành vào cụm kubernetes. Điều tương tự cũng có thể được xác thực bằng cách chạy các lệnh kubectl get pods, kubectl get deployments, kubectl get services, kubectl get replicaset.

```
C:\Windows\System32\cmd.exe  
  
C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods  
NAME READY STATUS RESTARTS AGE  
accounts-deployment-66d9858848-bt7v4 1/1 Running 0 19s  
configserver-deployment-6c949c5bc9-h8g7k 1/1 Running 0 108s  
eurekaserver-deployment-6d5444fd6f-7bs4v 1/1 Running 0 72s  
zipkin-deployment-789f9f965f-d4h46 1/1 Running 0 2m11s  
  
C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get deployments  
NAME READY UP-TO-DATE AVAILABLE AGE  
accounts-deployment 1/1 1 1 70s  
configserver-deployment 1/1 1 1 2m39s  
eurekaserver-deployment 1/1 1 1 2m3s  
zipkin-deployment 1/1 1 1 3m2s  
  
C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get services  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
accounts-service LoadBalancer 10.8.2.40 35.188.50.218 8080:30198/TCP 104s  
configserver-service LoadBalancer 10.8.5.25 35.192.202.210 8071:30530/TCP 3m12s  
eurekaserver-service LoadBalancer 10.8.13.120 34.71.30.92 8070:30818/TCP 2m36s  
kubernetes ClusterIP 10.8.0.1 <none> 443/TCP 41m  
zipkin-service LoadBalancer 10.8.7.218 35.232.24.41 9411:30332/TCP 3m36s  
  
C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get replicaset  
NAME DESIRED CURRENT READY AGE  
accounts-deployment-66d9858848 1 1 1 3m9s  
configserver-deployment-6c949c5bc9 1 1 1 4m38s  
eurekaserver-deployment-6d5444fd6f 1 1 1 4m2s  
zipkin-deployment-789f9f965f 1 1 1 5m1s
```

Microservices ▾

Search products and resources

REFRESH CREATE INGRESS DELETE

Cluster Namespace RESET SAVE PREVIEW

SERVICES INGRESS

Services are sets of pods with a network endpoint that can be used for discovery and load balancing. Ingresses are collections of rules for routing external HTTP(S) traffic to Services.

Filter Is system object : False Filter services and ingresses

<input type="checkbox"/>	Name ↑	Status	Type	Endpoints	Pods	Namespace	Clusters
<input type="checkbox"/>	accounts-service	OK	External load balancer	35.188.50.218:8080 ↗	1/1	default	cluster-1
<input type="checkbox"/>	configserver-service	OK	External load balancer	35.192.202.210:8071 ↗	1/1	default	cluster-1
<input type="checkbox"/>	eurekaserver-service	OK	External load balancer	34.71.30.92:8070 ↗	1/1	default	cluster-1
<input type="checkbox"/>	zipkin-service	OK	External load balancer	35.232.24.41:9411 ↗	1/1	default	cluster-1

Microservices ▾

Search products and resources

REFRESH DEPLOY DELETE

Cluster Namespace RESET SAVE PREVIEW

Workloads

Workloads are deployable units of computing that can be created and managed in a cluster.

Filter Is system object : False Filter workloads

<input type="checkbox"/>	Name ↑	Status	Type	Pods	Namespace	Cluster
<input type="checkbox"/>	accounts-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	configserver-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	eurekaserver-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	zipkin-deployment	OK	Deployment	1/1	default	cluster-1

Microservices ▾

Search products and resources

Deployment details REFRESH EDIT DELETE ACTIONS KUBECTL

accounts-deployment

OVERVIEW DETAILS REVISION HISTORY EVENTS LOGS YAML

1 hour 6 hours 12 hours 1 day 2 days 4 d

CPU Memory Disk

Cluster: cluster-1 Namespace: default Labels: app: accounts Logs: Container logs, Audit logs

Google Cloud Platform Microservices ▾

Search products and resources

Kubernetes Engine

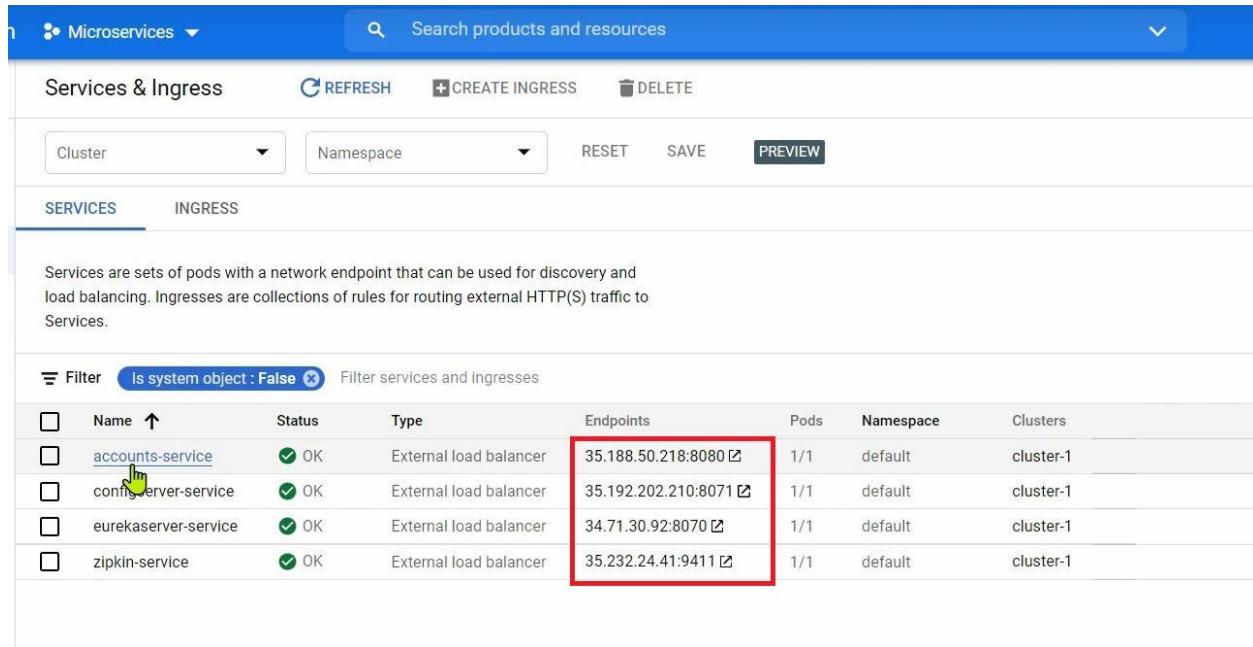
Clusters Workloads Services & Ingress Applications Configuration Storage Object Browser Migrate to containers

Node details

REFRESH EDIT CORDON KUBECTL

NAME	READY	CPU	MEMORY
pdcsl-node-zjkjk	Running	0 CPU	0 B
kube-proxy-gke-cluster-1-default-pool-a3d7a18e-30tm	Running	100 mCPU	0 B
fluentbit-gke-q6s8c	Running	100 mCPU	209.72 MB
gke-metrics-agent-v2pfh	Running	3 mCPU	52.43 MB
zipkin-deployment-789f9f965fd4h46	Running	0 CPU	0 B
configserver-deployment-6c949c5bc9-h8g7k	Running	0 CPU	0 B
accounts-deployment-66d9858848-bt7v4	Running	0 CPU	0 B

10. Validating our microservices deployed into K8s cluster- Xác thực các microservices đã triển khai vào cụm K8



The screenshot shows the Microservices dashboard with the following interface elements:

- Header: "Microservices" dropdown, "Search products and resources" input field, and a dropdown menu.
- Toolbar: "REFRESH", "CREATE INGRESS", "DELETE", "RESET", "SAVE", and a "PREVIEW" button.
- Filter bar: "Filter" dropdown set to "Cluster", "Namespace" dropdown set to "Namespace", and buttons for "RESET", "SAVE", and "PREVIEW".
- Tab navigation: "SERVICES" (selected) and "INGRESS".
- Description: "Services are sets of pods with a network endpoint that can be used for discovery and load balancing. Ingresses are collections of rules for routing external HTTP(S) traffic to Services."
- Table: A list of services with columns: Name, Status, Type, Endpoints, Pods, Namespace, and Clusters. The "Endpoints" column for the "accounts-service" row is highlighted with a red box and a green cursor icon pointing to it.

<input type="checkbox"/>	Name ↑	Status	Type	Endpoints	Pods	Namespace	Clusters
<input type="checkbox"/>	accounts-service	✓ OK	External load balancer	35.188.50.218:8080 [edit]	1/1	default	cluster-1
<input type="checkbox"/>	configserver-service	✓ OK	External load balancer	35.192.202.210:8071 [edit]	1/1	default	cluster-1
<input type="checkbox"/>	eurekasperver-service	✓ OK	External load balancer	34.71.30.92:8070 [edit]	1/1	default	cluster-1
<input type="checkbox"/>	zipkin-service	✓ OK	External load balancer	35.232.24.41:9411 [edit]	1/1	default	cluster-1

11. Automatic Self healing inside Kubernetes cluster

Sử dụng ReplicaSets và Deployments để đảm bảo rằng số lượng replica (bản sao) của các Pod được duy trì theo số lượng mong muốn. Nếu có bất kỳ Pod nào bị lỗi hoặc không khỏe mạnh, ReplicaSets sẽ tự động khởi tạo các Pod mới để thay thế.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: accounts-deployment
  labels:
    app: accounts
spec:
  replicas: 2 #tăng số lượng replica (bản sao) của các Pod
  selector:
    matchLabels:
      app: accounts
  template:
    metadata:
      labels:
        app: accounts
```

```
C:\Windows\System32\cmd.exe
C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get replicaset
NAME          DESIRED  CURRENT  READY  AGE
accounts-deployment-66d9858848  1        1        1      29m
configserver-deployment-6c949c5bc9  1        1        1      30m
eurekaserver-deployment-6d5444fd6f  1        1        1      30m
zipkin-deployment-789f9f965f    1        1        1      30m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods
NAME          READY  STATUS  RESTARTS  AGE
accounts-deployment-66d9858848-bt7v4  1/1   Running  0      30m
configserver-deployment-6c949c5bc9-h8g7k  1/1   Running  0      32m
eurekaserver-deployment-6d5444fd6f-7bs4v  1/1   Running  0      31m
zipkin-deployment-789f9f965f-d4h46    1/1   Running  0      32m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl apply -f 5_accounts.yml
deployment.apps/accounts-deployment configured
service/accounts-service unchanged

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods
NAME          READY  STATUS  RESTARTS  AGE
accounts-deployment-66d9858848-bt7v4  1/1   Running  0      31m
accounts-deployment-66d9858848-trq2f  1/1   Running  0      16s
configserver-deployment-6c949c5bc9-h8g7k  1/1   Running  0      32m
eurekaserver-deployment-6d5444fd6f-7bs4v  1/1   Running  0      32m
zipkin-deployment-789f9f965f-d4h46    1/1   Running  0      33m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get replicaset
NAME          DESIRED  CURRENT  READY  AGE
accounts-deployment-66d9858848  2        2        2      31m
configserver-deployment-6c949c5bc9  1        1        1      33m
eurekaserver-deployment-6d5444fd6f  1        1        1      32m
zipkin-deployment-789f9f965f    1        1        1      33m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>
```

```
117. Automatic Self healing inside Kubernetes cluster
C:\Windows\System32\cmd.exe
C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods
NAME          READY  STATUS  RESTARTS  AGE
accounts-deployment-66d9858848-bt7v4  1/1   Running  0      32m
accounts-deployment-66d9858848-trq2f  1/1   Running  0      107s
configserver-deployment-6c949c5bc9-h8g7k  1/1   Running  0      34m
eurekaserver-deployment-6d5444fd6f-7bs4v  1/1   Running  0      33m
zipkin-deployment-789f9f965f-d4h46    1/1   Running  0      34m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl delete pod accounts-deployment-66d9858848-bt7v4
pod "accounts-deployment-66d9858848-bt7v4" deleted

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods
NAME          READY  STATUS  RESTARTS  AGE
accounts-deployment-66d9858848-trq2f  1/1   Running  0      2m39s
accounts-deployment-66d9858848-vxrcj  1/1   Running  0      21s
configserver-deployment-6c949c5bc9-h8g7k  1/1   Running  0      35m
eurekaserver-deployment-6d5444fd6f-7bs4v  1/1   Running  0      34m
zipkin-deployment-789f9f965f-d4h46    1/1   Running  0      35m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get replicaset
NAME          DESIRED  CURRENT  READY  AGE
accounts-deployment-66d9858848  2        2        2      34m
configserver-deployment-6c949c5bc9  1        1        1      35m
eurekaserver-deployment-6d5444fd6f  1        1        1      35m
zipkin-deployment-789f9f965f    1        1        1      36m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>
```

Microservices ▾ Search products and resources

Services & Ingress REFRESH CREATE INGRESS DELETE

Cluster Namespace RESET SAVE PREVIEW

SERVICES **INGRESS**

Services are sets of pods with a network endpoint that can be used for discovery and load balancing. Ingresses are collections of rules for routing external HTTP(S) traffic to Services.

Filter Is system object : False Filter services and ingresses

<input type="checkbox"/>	Name ↑	Status	Type	Endpoints	Pods	Namespace	Clusters
<input type="checkbox"/>	accounts-service	✓ OK	External load balancer	35.188.50.218:8080 ↗	2/2	default	cluster-1
<input type="checkbox"/>	configserver-service	✓ OK	External load balancer	35.192.202.210:8071 ↗	1/1	default	cluster-1
<input type="checkbox"/>	eurekaserver-service	✓ OK	External load balancer	34.71.30.92:8070 ↗	1/1	default	cluster-1
<input type="checkbox"/>	zipkin-service	✓ OK	External load balancer	35.232.24.41:9411 ↗	1/1	default	cluster-1

Microservices ▾ Search products and resources

← Service details REFRESH EDIT DELETE KUBECTL ▾

Cluster	cluster-1
Namespace	default
Labels	No labels set
Logs	accounts-deployment
Type	LoadBalancer
External endpoints	35.188.50.218:8080 ↗

Load balancer

Cluster IP	10.8.2.40
Load balancer IP	35.188.50.218
Load balancer	a45b446378089405a8fa9888bbc7c7c6

Deployments

Name	Status	Pods
accounts-deployment	✓ OK	2/2

Serving pods

Name	Status	Endpoints	Restarts	Created on ↑
accounts-deployment-66d9858848-trq2f	✓ Running	10.4.0.4	0	20 Jun 2021, 11:22:34
accounts-deployment-66d9858848-vxrcj	✓ Running	10.4.2.7	0	20 Jun 2021, 11:24:52



12. Automatic Rollout & Rollback inside Kubernetes cluster

Trong cụm Kubernetes trên Google Cloud Platform (GCP), bạn có thể triển khai tự động (Automatic Rollout) và quay trở lại tự động (Automatic Rollback) bằng cách sử dụng tính năng Deployment. Đây là một cơ chế quản lý phiên bản ứng dụng trong Kubernetes, cho phép bạn tự động triển khai phiên bản mới và quay trở lại phiên bản trước đó nếu có lỗi.

Dưới đây là mô tả về cách thực hiện Automatic Rollout và Automatic Rollback trong Kubernetes trên GCP:

Automatic Rollout (Triển khai tự động):

- Bước 1:** Tạo một tệp YAML để định nghĩa Deployment cho ứng dụng của bạn. Trong tệp YAML này, bạn xác định thông tin về hình ảnh container, số lượng bản sao (replicas), cấu hình tự động triển khai và các thông tin khác về ứng dụng.
- Bước 2:** Sử dụng lệnh kubectl apply để triển khai tệp YAML vào cụm Kubernetes. Deployment sẽ tự động tạo ra các ReplicaSet và Pod mới để triển khai phiên bản mới của ứng dụng.
- Bước 3:** Deployment sẽ kiểm tra sự khả dụng và sức khỏe của phiên bản mới. Nếu phiên bản mới đáp ứng các điều kiện xác định (ví dụ: số lượng replicas đạt mức mong muốn và các liveness/readiness probes trả về trạng thái là "success"), nó sẽ tiến hành triển khai tự động và chuyển từ phiên bản cũ sang phiên bản mới.

Ví dụ dùng google console để tăng replicas

```
C:\Windows\System32\cmd.exe

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get deployments
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
accounts-deployment   2/2     2          2          49m
configserver-deployment 1/1     1          1          51m
eurekaserver-deployment 1/1     1          1          50m
zipkin-deployment    1/1     1          1          51m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl scale deployment accounts-deployment --replicas=3
deployment.apps/accounts-deployment scaled

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
accounts-deployment-66d9858848-d76ls   1/1     Running   0          5s
accounts-deployment-66d9858848-trq2f   1/1     Running   0          20m
accounts-deployment-66d9858848-vxrcj   1/1     Running   0          18m
configserver-deployment-6c949c5bc9-h8g7k 1/1     Running   0          53m
eurekaserver-deployment-6d5444fd6f-7bs4v 1/1     Running   0          52m
zipkin-deployment-789f9f965f-d4h46     1/1     Running   0          53m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>
```

```
C:\Windows\System32\cmd.exe - kubectl describe pod accounts-deployment-66d9858848-d76ls

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
accounts-deployment-66d9858848-d76ls   1/1     Running   0          39s
accounts-deployment-66d9858848-trq2f   1/1     Running   0          21m
accounts-deployment-66d9858848-vxrcj   1/1     Running   0          19m
configserver-deployment-6c949c5bc9-h8g7k 1/1     Running   0          53m
eurekaserver-deployment-6d5444fd6f-7bs4v 1/1     Running   0          53m
zipkin-deployment-789f9f965f-d4h46     1/1     Running   0          54m
```

Nếu cỗ gắng phát hành docker image mới sẽ không thành công nếu không tìm thấy image đó.

```
C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl set image deployment accounts-deployment accounts=eazybytes/accounts:k8suytptyr
deployment.apps/accounts-deployment image updated

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
accounts-deployment-66d9858848-d76ls   1/1     Running   0          3m57s
accounts-deployment-66d9858848-trq2f    1/1     Running   0          24m
accounts-deployment-66d9858848-vxrcj    1/1     Running   0          22m
accounts-deployment-78cd89d69c-475m5    0/1     ErrImagePull 0          13s
configserver-deployment-6c949c5bc9-h8g7k 1/1     Running   0          57m
eurekaserver-deployment-6d5444fd6f-7bs4v 1/1     Running   0          56m
zipkin-deployment-789f9f965f-d4h46      1/1     Running   0          57m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>
```

Tạo các pod mới nếu phát hành thành công image:

```
C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl set image deployment accounts-deployment accounts=eazybytes/accounts:k8s
deployment.apps/accounts-deployment image updated

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
accounts-deployment-66d9858848-d76ls   1/1     Running   0          5m49s
accounts-deployment-66d9858848-trq2f    1/1     Running   0          26m
accounts-deployment-66d9858848-vxrcj    1/1     Running   0          24m
accounts-deployment-7796fc4bdc-6psqf    0/1     ContainerCreating 0          7s
configserver-deployment-6c949c5bc9-h8g7k 1/1     Running   0          59m
eurekaserver-deployment-6d5444fd6f-7bs4v 1/1     Running   0          58m
zipkin-deployment-789f9f965f-d4h46      1/1     Running   0          59m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
accounts-deployment-66d9858848-d76ls   1/1     Terminating 0          6m1s
accounts-deployment-66d9858848-trq2f    1/1     Terminating 0          26m
accounts-deployment-7796fc4bdc-6h7rm    1/1     Running   0          4s
accounts-deployment-7796fc4bdc-6psqf    1/1     Running   0          19s
accounts-deployment-7796fc4bdc-g7xln    1/1     Running   0          11s
configserver-deployment-6c949c5bc9-h8g7k 1/1     Running   0          59m
eurekaserver-deployment-6d5444fd6f-7bs4v 1/1     Running   0          58m
zipkin-deployment-789f9f965f-d4h46      1/1     Running   0          59m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>
```

Automatic Rollback (Quay trở lại tự động):

- Nếu có lỗi xảy ra trong phiên bản mới hoặc phiên bản mới không đáp ứng yêu cầu, Deployment sẽ tự động thực hiện quay trở lại tự động.
- Khi quay trở lại tự động được kích hoạt, Deployment sẽ tự động quay lại phiên bản trước đó của ứng dụng, loại bỏ phiên bản mới không thành công.

```
C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl rollout history deployment accounts-deployment
deployment.apps/accounts-deployment
REVISION  CHANGE-CAUSE
1          <none>
2          <none>
3          <none>

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl rollout undo deployment accounts-deployment --to-revision=1
deployment.apps/accounts-deployment rolled back

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
accounts-deployment-66d9858848-42h49  1/1     Running   0          15s
accounts-deployment-66d9858848-77kdb  1/1     Running   0          12s
accounts-deployment-66d9858848-srjls  1/1     Running   0          17s
accounts-deployment-7796fc4bdc-6h7rm  0/1     Terminating   0          6m13s
accounts-deployment-7796fc4bdc-6psqf  0/1     Terminating   0          6m28s
configserver-deployment-6c949c5bc9-h8g7k  1/1     Running   0          65m
eurekaserver-deployment-6d5444fd6f-7bs4v  1/1     Running   0          64m
zipkin-deployment-789f9f965f-d4h46   1/1     Running   0          65m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>
```

```
C:\Windows\System32\cmd.exe

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl rollout history deployment accounts-deployment
deployment.apps/accounts-deployment
REVISION  CHANGE-CAUSE
1          <none>
2          <none>
3          <none>

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl rollout undo deployment accounts-deployment --to-revision=1
deployment.apps/accounts-deployment rolled back

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
accounts-deployment-66d9858848-42h49  1/1     Running   0          15s
accounts-deployment-66d9858848-77kdb  1/1     Running   0          12s
accounts-deployment-66d9858848-srjls  1/1     Running   0          17s
accounts-deployment-7796fc4bdc-6h7rm  0/1     Terminating   0          6m13s
accounts-deployment-7796fc4bdc-6psqf  0/1     Terminating   0          6m28s
configserver-deployment-6c949c5bc9-h8g7k  1/1     Running   0          65m
eurekaserver-deployment-6d5444fd6f-7bs4v  1/1     Running   0          64m
zipkin-deployment-789f9f965f-d4h46   1/1     Running   0          65m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
accounts-deployment-66d9858848-42h49  1/1     Running   0          31s
accounts-deployment-66d9858848-77kdb  1/1     Running   0          28s
accounts-deployment-66d9858848-srjls  1/1     Running   0          33s
configserver-deployment-6c949c5bc9-h8g7k  1/1     Running   0          65m
eurekaserver-deployment-6d5444fd6f-7bs4v  1/1     Running   0          65m
zipkin-deployment-789f9f965f-d4h46   1/1     Running   0          66m
```

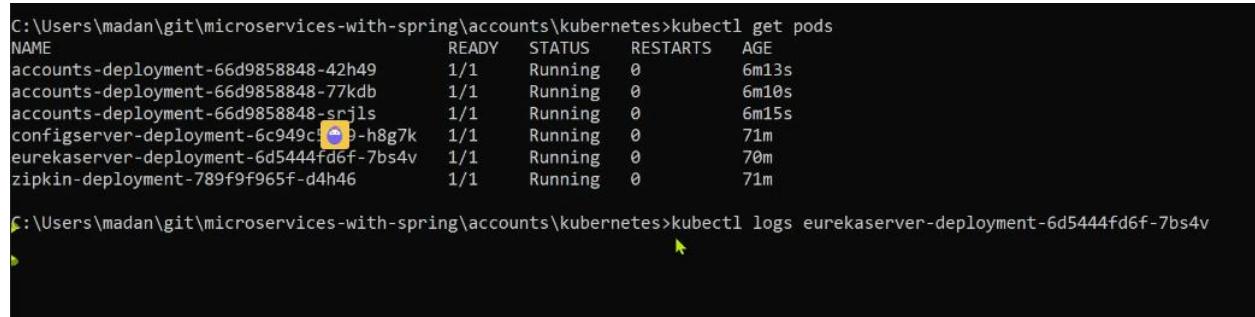
Việc triển khai và quay trở lại tự động giúp đảm bảo rằng ứng dụng của bạn luôn ổn định và khả dụng trong cụm Kubernetes. Nó cũng giúp giảm thiểu thời gian chết và đảm bảo rằng các lỗi triển khai không ảnh hưởng đến khách hàng và người dùng cuối.

13. Logging & Monitoring inside Kubernetes cluster

Logging và Monitoring là hai khía cạnh quan trọng trong việc quản lý và giám sát cụm Kubernetes. Chúng giúp bạn theo dõi hiệu suất, sức khỏe và hoạt động của các ứng dụng và hạ tầng Kubernetes.

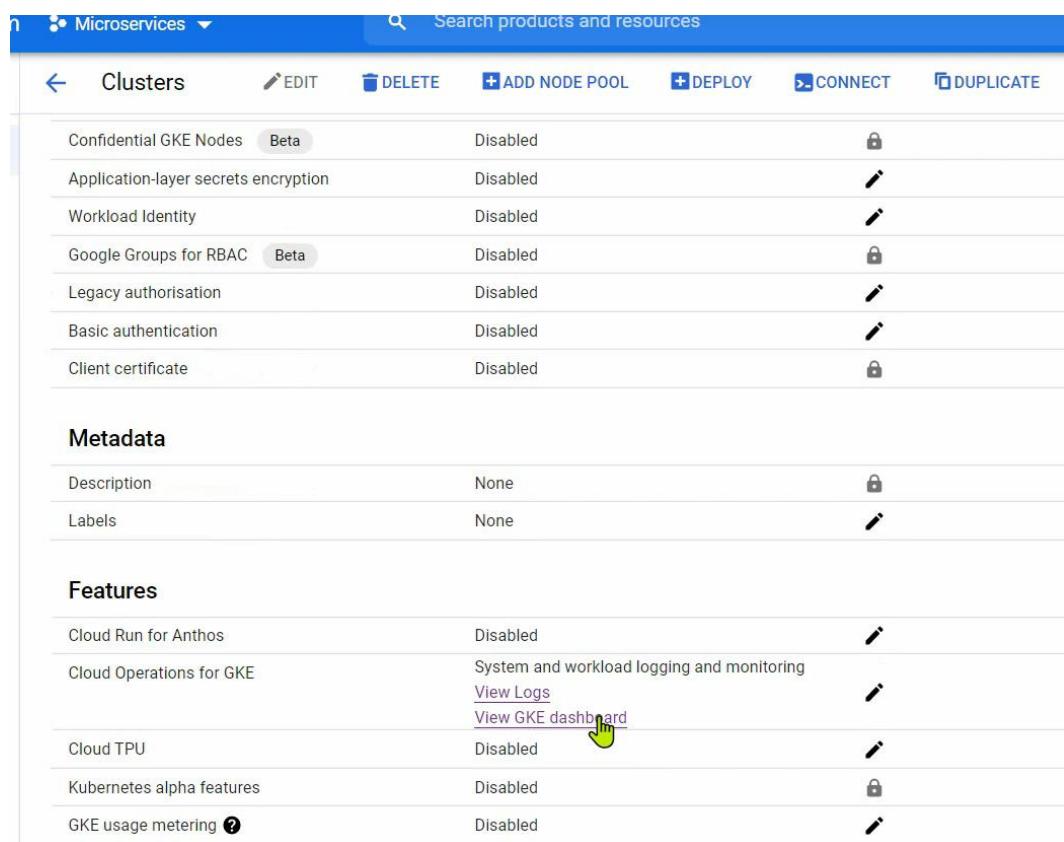
Để triển khai Logging và Monitoring trong Kubernetes trên GCP, bạn có thể sử dụng các dịch vụ có sẵn như sau:

- Logging: Sử dụng Stackdriver Logging để tự động thu thập và lưu trữ các log từ các container và các dịch vụ khác trong cụm Kubernetes. Bạn có thể sử dụng Stackdriver Logging để tìm kiếm và phân tích các log để giải quyết các sự cố và theo dõi hiệu suất của các ứng dụng.
- Monitoring: Sử dụng Stackdriver Monitoring để giám sát các thông số hiệu suất và tài nguyên trong cụm Kubernetes. Bạn có thể theo dõi các metric liên quan đến CPU, RAM, lưu lượng mạng và các metric tùy chỉnh khác. Stackdriver Monitoring cũng tích hợp với Prometheus để thu thập các metric từ các Pod và Service.



```
C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
accounts-deployment-66d9858848-42h49   1/1     Running   0          6m13s
accounts-deployment-66d9858848-77kdb   1/1     Running   0          6m10s
accounts-deployment-66d9858848-srjls   1/1     Running   0          6m15s
configserver-deployment-6c949cf9-h8g7k  1/1     Running   0          71m
eurekaserver-deployment-6d5444fd6f-7bs4v 1/1     Running   0          70m
zipkin-deployment-789f9f965f-d4h46    1/1     Running   0          71m

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl logs eurekaserver-deployment-6d5444fd6f-7bs4v
```



Feature	Status	Action
Cloud Run for Anthos	Disabled	Edit
Cloud Operations for GKE	System and workload logging and monitoring View Logs View GKE dashboard	Edit
Cloud TPU	Disabled	Edit
Kubernetes alpha features	Disabled	Edit
GKE usage metering ?	Disabled	Edit

cluster-1 – Clusters – Kubernetes | Logs explorer – Logging – Microservices | Google Cloud Platform | 35.188.50.218:8080/sayHello | Docker Hub | + | X

console.cloud.google.com/logs/query?query=resource.type%3D"k8s_cluster"%0Aresource.labels.project_id%3D"microservices-316111"%0Aresource.labels.location%3D"us-central1-c"%0Aresource.labels.cluste...

Logs explorer

REFINE SCOPE Project

SHARE LINK LAST 1 HOUR PAGE LAYOUT LEARN

New features are available in the Logs explorer.

Query Recent (5) Saved (0) Suggested (1)

resource.type="k8s_cluster" resource.labels.project_id="microservices-316111" resource.labels.location="us-central1-c" resource.labels.cluste...

Save Stream logs Run query Edit query

Histogram

Query results

SEVERITY TIMESTAMP IST SUMMARY

- 2021-06-20 12:02:29.595 IST cluster-1 k8s.io io.k8s.authorization.rbac.v1.clusterrolebindings.patch -
- 2021-06-20 12:02:29.602 IST cluster-1 k8s.io io.k8s.authorization.rbac.v1.clusterrolebindings.patch -
- 2021-06-20 12:02:29.627 IST cluster-1 k8s.io io.k8s.authorization.rbac.v1.clusterrolebindings.patch -
- 2021-06-20 12:02:29.637 IST cluster-1 k8s.io io.k8s.authorization.rbac.v1.clusterrolebindings.patch -
- 2021-06-20 12:02:29.647 IST cluster-1 k8s.io io.k8s.authorization.rbac.v1.clusterrolebindings.patch -
- 2021-06-20 12:02:29.657 IST cluster-1 k8s.io io.k8s.authorization.rbac.v1.clusterrolebindings.patch -
- 2021-06-20 12:02:29.673 IST cluster-1 k8s.io io.k8s.authorization.rbac.v1.clusterrolebindings.patch -
- 2021-06-20 12:02:29.680 IST cluster-1 k8s.io io.k8s.authorization.rbac.v1.clusterrolebindings.patch -
- 2021-06-20 12:02:29.686 IST cluster-1 k8s.io io.k8s.authorization.rbac.v1.clusterrolebindings.patch -
- 2021-06-20 12:02:29.699 IST cluster-1 k8s.io io.k8s.apps.v1.daemonsets.patch apps/v1/namespaces/kube-system/daemonsets/kube-proxy system:addon-manager aud...
- 2021-06-20 12:02:29.750 IST cluster-1 k8s.io io.k8s.authorization.rbac.v1beta1.clusterrolebindings.patch -
- 2021-06-20 12:02:29.759 IST cluster-1 k8s.io io.k8s.authorization.rbac.v1beta1.clusterrolebindings.patch -

https://console.cloud.google.com/logs/query?project=microservices-316111

cluster-1 – Clusters – Kubernetes | Logs explorer – Logging – Microservices | GKE dashboard – Monitoring – | 35.188.50.218:8080/sayHello | Docker Hub | + | X

console.cloud.google.com/monitoring/dashboards/resourceList/kubernetes?timeDomain=1m&project=microservices-316111&pageState={"interval":0}

Monitoring

GKE dashboard SEND FEEDBACK

Service Level Objectives for GKE

Use your Service Level Objectives (SLOs) to monitor the health of your resources within the GKE dashboard.

LEARN MORE

1 – 4 of 4

Nodes No active alerts NUM_ENTITIES.plural =0(0 (PLURAL_ENTITY_NAME) with active alerts=1(1 (SINGULAR_ENTITY_NAME) with active alerts)

Name	Alerts	Container restarts	Error logs	CPU utilisation	Memory utilisation	Disk utilisation
gke-cluster-1-default-pool...	0	Location: us-centra... +1	0	51.41% of 0.62 CPU	14.05% of 687 MiB	0% of 486.83 G
gke-cluster-1-default-pool...	0	Location: us-centra... +1	0	16.58% of 0.22 CPU	25.89% of 260 MiB	0% of 203.97 G
gke-cluster-1-default-pool...	0	Location: us-centra... +1	0	14.27% of 0.46 CPU	12.43% of 360 MiB	0% of 294.41 G

1 – 3 of 3

Workloads No active alerts NUM_ENTITIES.plural =0(0 (PLURAL_ENTITY_NAME) with active alerts=1(1 (SINGULAR_ENTITY_NAME) with active alerts)

Name	Alerts	Labels	Container restarts	Error logs	CPU utilisation	Memory utilisation
accounts-deployment	0	Cluster: cluster-1 Location: us-centra...	+3	0	0.01 CPU	832.4 MiB
configserver-deployment	0	Cluster: cluster-1 Location: us-centra...	+3	0	0 CPU	319.06 MiB
eurekaserver-deployment	0	Cluster: cluster-1 Location: us-centra...	+3	0	0 CPU	313.88 MiB
zipkin-deployment	0	Cluster: cluster-1 Location: us-centra...	+3	0	0 CPU	212.79 MiB
event-exporter-gke	0	Cluster: cluster-1 Location: us-centra...	+3	0	0 CPU	26.3 MiB

1 – 5 of 18

accounts-deployment - Deployment | Logs explorer - Logging - Microservices | GKE dashboard - Monitoring - | 35.188.50.218:8080/sayHello | Docker Hub | + | - | X

console.cloud.google.com/kubernetes/deployment/us-central1-c/cluster-1/default/accounts-deployment/overview?project=microservices-316111

Google Cloud Platform | Microservices | Search products and resources | SHOW INFO PANEL | OPERATIONS

Kubernetes Engine

Deployment details

accounts-deployment

OVERVIEW DETAILS REVISION HISTORY EVENTS LOGS YAML

1 hour 6 hours 12 hours 1 day 2 days 4 days 7 days 14 days 30 days

CPU Memory Disk

Cluster: cluster-1 Namespace: default Labels: app: accounts Logs: Container logs, Audit logs Replicas: 3 updated, 3 ready, 3 available, 0 unavailable Pod specification: Revision 4, containers: accounts

Active revisions

Revision	Name	Status	Summary	Created on	Pods running/Pods total
1	accounts-deployment-74458d46-2	Up-to-date	Deployment up-to-date with the latest commit.	2021-06-20T06:26:21.261339044Z	1/3

accounts-service - Service details | Logs explorer - Logging - Microservices | GKE dashboard - Monitoring - | 35.188.50.218:8080/sayHello | Docker Hub | + | - | X

console.cloud.google.com/kubernetes/service/us-central1-c/cluster-1/default/accounts-service/logs?project=microservices-316111

Google Cloud Platform | Microservices | Search products and resources | SHOW INFO PANEL | OPERATIONS

Kubernetes Engine

Service details

accounts-service

OVERVIEW DETAILS EVENTS LOGS YAML

Severity: Default Filter: Filter logs

Service logs: Showing 31 messages

```

2021-06-20T06:26:21.261339044Z 2021-06-20 06:26:21.268 INFO [accounts,,] 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Force full registry fetch : -
2021-06-20T06:26:21.261422294Z 2021-06-20 06:26:21.268 INFO [accounts,,] 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Application is null : false
2021-06-20T06:26:21.261488044Z 2021-06-20 06:26:21.261 INFO [accounts,,] 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Registered Applications size_-
2021-06-20T06:26:21.261501862Z 2021-06-20 06:26:21.261 INFO [accounts,,] 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Application version is -1: f_-
2021-06-20T06:26:21.261510207Z 2021-06-20 06:26:21.261 INFO [accounts,,] 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Getting all instance registr_-
2021-06-20T06:26:21.288936627Z 2021-06-20 06:26:21.288 INFO [accounts,,] 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : The response status is 200
2021-06-20T06:26:24.110982857Z 2021-06-20 06:26:24.118 INFO [accounts,,] 1 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[/] : Initializing Spring DispatcherS...
2021-06-20T06:26:24.11100873Z 2021-06-20 06:26:24.118 INFO [accounts,,] 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcher...
2021-06-20T06:26:24.115367174Z 2021-06-20 06:26:24.115 INFO [accounts,,] 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 3 ms
2021-06-20T06:30:36.923136233Z 2021-06-20 06:30:36.922 INFO [accounts,,] 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via...
2021-06-20T06:30:41.481333514Z 2021-06-20 06:30:41.488 INFO [accounts,,] 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via...
2021-06-20T06:30:58.442445787Z 2021-06-20 06:30:58.441 INFO [accounts,,] 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via...
2021-06-20T06:35:36.924632492Z 2021-06-20 06:35:36.923 INFO [accounts,,] 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via...
2021-06-20T06:35:41.482865879Z 2021-06-20 06:35:41.482 INFO [accounts,,] 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via...
2021-06-20T06:35:58.443714233Z 2021-06-20 06:35:58.443 INFO [accounts,,] 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via...

```

No newer entries found matching current filter.

<https://console.cloud.google.com/kubernetes/service/us-central1-c/cluster-1/default/accounts-service/events?project=microservices-316111>

14. Autoscaling inside Kubernetes cluster using HPA

Trong Kubernetes, Autoscaling (tự động mở rộng) cho phép cụm tự động điều chỉnh số lượng bản sao (replicas) của các Pod dựa trên lượng tải công việc và các metric được định nghĩa. Horizontal Pod Autoscaler (HPA) là phép bạn triển khai tự động mở rộng số lượng Pod theo các ngưỡng tải được định nghĩa, giúp cân bằng tải và đáp ứng nhu cầu thay đổi của ứng dụng.

Cách triển khai Autoscaling trong Kubernetes bằng cách sử dụng Horizontal Pod Autoscaler (HPA):

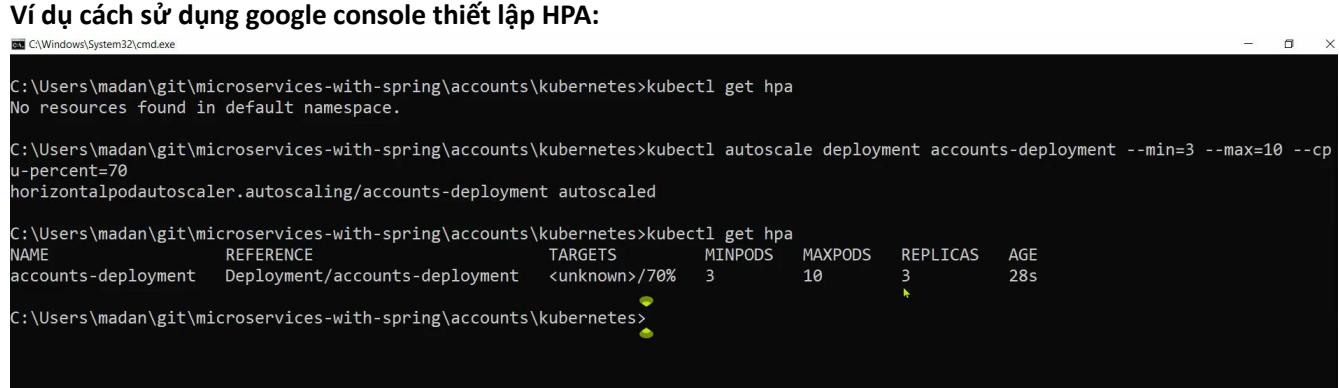
- Tạo Horizontal Pod Autoscaler (HPA): Tạo một tệp YAML mới để định nghĩa HPA. Trong tệp YAML này, cần chỉ định Deployment hoặc ReplicaSet mà bạn muốn áp dụng HPA và các thiết lập liên quan như số lượng bản sao tối thiểu và tối đa, các metric CPU và Memory để theo dõi, các điều kiện mở rộng.

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: my-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-deployment
  minReplicas: 2
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        targetAverageUtilization: 50
```

Trong ví dụ này, HPA sẽ theo dõi metric CPU của Pod trong Deployment **my-deployment**. Nếu CPU đạt trung bình 50%, HPA sẽ điều chỉnh số lượng bản sao để đáp ứng mức tải yêu cầu.

- Triển khai Horizontal Pod Autoscaler: Sử dụng lệnh kubectl apply để triển khai tệp YAML của HPA vào cụm Kubernetes.
`kubectl apply -f hpa.yaml`
- Khi HPA được triển khai, nó sẽ tự động theo dõi các metric và điều chỉnh số lượng Pod theo các ngưỡng được cấu hình.
- Quan sát Autoscaling: Bạn có thể sử dụng lệnh kubectl get hpa để kiểm tra trạng thái của HPA và số lượng bản sao hiện tại của Pod.
`kubectl get hpa`

Ví dụ cách sử dụng google console thiết lập HPA:



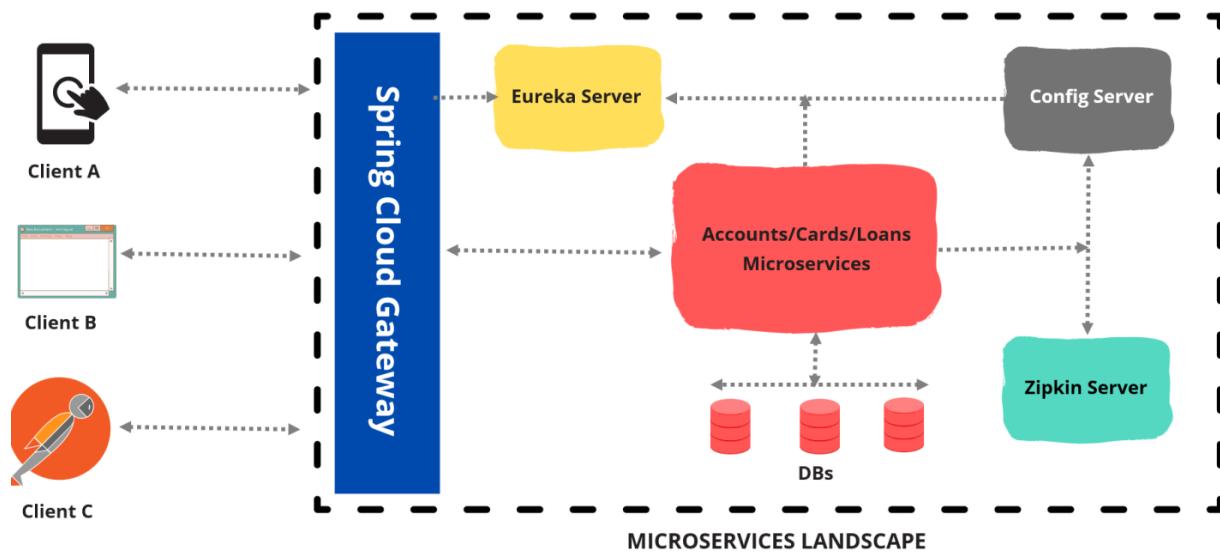
```
C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get hpa
No resources found in default namespace.

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl autoscale deployment accounts-deployment --min=3 --max=10 --cpu-percent=70
horizontalpodautoscaler.autoscaling/accounts-deployment autoscaled

C:\Users\madan\git\microservices-with-spring\accounts\kubernetes>kubectl get hpa
NAME          REFERENCE          TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
accounts-deployment  Deployment/accounts-deployment <unknown>/70%  3          10         3          28s
```

Section 12: Deploying all the microservices into K8s cluster

1. Deploying all the microservices into K8s cluster



Triển khai các microservices vào cụm Kubernetes (K8s) là một quá trình quan trọng và phức tạp trong việc xây dựng hệ thống ứng dụng dựa trên kiến trúc microservices. Dưới đây là mô tả tổng quan về quá trình triển khai các microservices vào cụm Kubernetes:

- Containerization: Trước khi triển khai vào Kubernetes, các microservices cần được đóng gói trong các container Docker. Containerization giúp đảm bảo rằng các microservices được đóng gói kín và độc lập với môi trường, đồng thời cho phép dễ dàng di chuyển và triển khai vào cụm Kubernetes.
- Kubernetes Cluster: Bạn cần sẵn sàng một cụm Kubernetes hoạt động trên môi trường bạn muốn triển khai các microservices. Cụm này có thể ở trong môi trường đám mây công cộng như Google Cloud Platform (GCP), Amazon Web Services (AWS), Microsoft Azure, hoặc ở trong môi trường riêng (on-premises).
- Dịch vụ Discovery và Load Balancing: Để triển khai các microservices, bạn cần một giải pháp dịch vụ discovery và load balancing để các microservices có thể tìm thấy và giao tiếp với nhau một cách độc lập với các vị trí vật lý của chúng. Kubernetes cung cấp dịch vụ DNS mặc định để giải quyết vấn đề này.
- Kubernetes Manifests: Triển khai các microservices vào Kubernetes được thực hiện thông qua việc sử dụng các tệp YAML được gọi là Kubernetes manifests. Trong các manifests, bạn định nghĩa các đối tượng như Deployments, Services, ConfigMaps, Secrets và các tài nguyên khác cần thiết để chạy các microservices.
- Deployment: Sử dụng Deployment để định nghĩa các microservices bạn muốn triển khai. Deployment là một đối tượng trong Kubernetes cho phép bạn quản lý các bản sao của các Pod, cập nhật phiên bản và xử lý quá trình triển khai.
- Service: Sử dụng Service để tạo một điểm cuối ổn định cho các microservices. Service giúp tự động phát hiện các Pod mới được tạo bởi Deployment và cân bằng tải các yêu cầu đến các Pod này.

- ConfigMaps và Secrets: ConfigMaps và Secrets cho phép bạn quản lý cấu hình ứng dụng và thông tin nhạy cảm như mật khẩu hoặc khóa bí mật mà các microservices cần để hoạt động.
- Monitoring và Logging: Đảm bảo rằng bạn đã cấu hình giám sát và ghi log cho cụm Kubernetes và các microservices để theo dõi hiệu suất và sức khỏe của hệ thống.
- Continuous Integration/Continuous Deployment (CI/CD): Khi triển khai các microservices vào Kubernetes, bạn nên xem xét việc sử dụng các quy trình CI/CD để tự động hóa quá trình triển khai. CI/CD giúp đảm bảo rằng các microservices được triển khai một cách liên tục và đáng tin cậy.
- Thử nghiệm và Phân phối: Trước khi đưa vào hoạt động, hãy đảm bảo rằng bạn đã kiểm tra và thử nghiệm từng microservice một cách kỹ lưỡng. Khi triển khai, cân nhắc sử dụng các phương pháp phân phối theo giai đoạn (phần trăm người dùng) để giảm thiểu tác động đối với người dùng cuối trong trường hợp xảy ra vấn đề không mong muốn.

2. Creating the K8s yaml config files for all microservices

Ví dụ Accounts microservice (các microservice khác tương tự)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: accounts-deployment
  labels:
    app: accounts
spec:
  replicas: 1
  selector:
    matchLabels:
      app: accounts
  ---
...
apiVersion: v1
kind: Service
metadata:
  name: accounts-service
spec:
  selector:
    app: accounts
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
```

3. How Deployment and Service are tied together inside K8s

Trong Kubernetes, Deployment và Service có mối liên kết chặt chẽ và phụ thuộc vào nhau để triển khai và cung cấp các ứng dụng trong cụm.

Deployment (Triển khai):

- Deployment là một tài nguyên trong Kubernetes cho phép bạn quản lý một tập hợp các Pod chạy một ứng dụng cụ thể trong cụm.
- Deployment hỗ trợ việc triển khai và quản lý các bản sao của các Pod. Nó cho phép bạn định nghĩa phiên bản ứng dụng cụ thể và tự động xử lý quá trình triển khai các bản cập nhật mới.
- Khi bạn thay đổi phiên bản của ứng dụng (ví dụ: cập nhật hình ảnh container), Deployment sẽ tự động tạo và phân phối các bản sao mới của Pod và sau đó loại bỏ các bản sao cũ khi mới được xác định là ổn định.

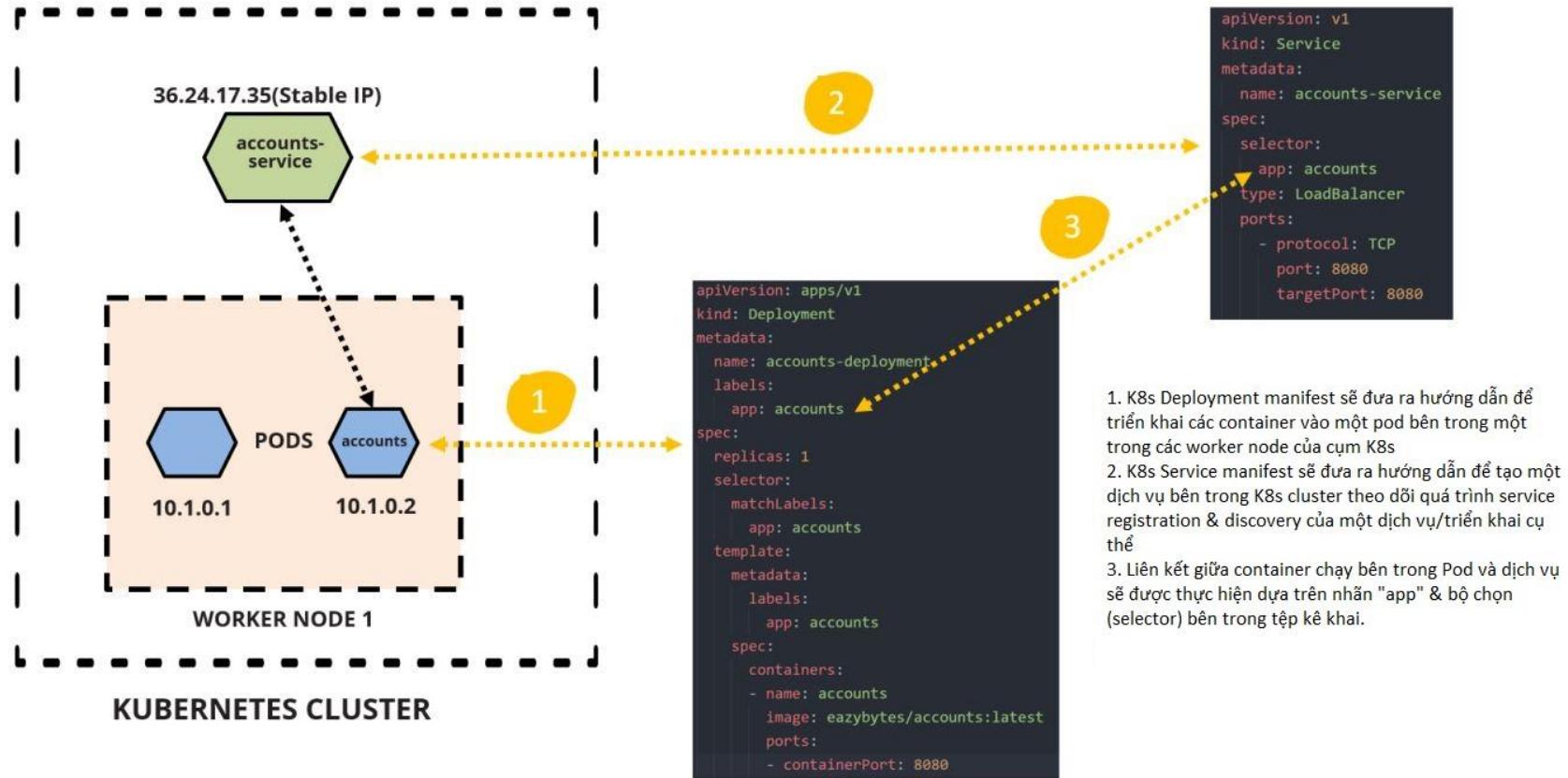
Service (Dịch vụ):

- Service là một tài nguyên trong Kubernetes dùng để tạo một điểm cuối ổn định (stable endpoint) cho các Pod.
- Service giúp các ứng dụng hoặc dịch vụ khác trong cụm truy cập các Pod một cách dễ dàng mà không cần biết vị trí cụ thể của từng Pod.
- Service sử dụng label selector để xác định các Pod mà nó sẽ cung cấp điểm cuối. Bất kỳ yêu cầu nào gửi tới Service sẽ được cân bằng tải đến các Pod phù hợp với label selector.

Liên kết giữa Deployment và Service:

- Liên kết giữa Deployment và Service xảy ra thông qua label selector.
- Trong tệp YAML của Deployment, bạn cần chỉ định các label cho Pod template của mình. Những label này sẽ được sử dụng để phân biệt các Pod mà Deployment tạo ra.
- Trong tệp YAML của Service, bạn cần chỉ định label selector để chọn các Pod mà Service sẽ cung cấp điểm cuối.
- Label selector trong Service phải khớp với label trong Pod template của Deployment để đảm bảo rằng Service sẽ chọn đúng các Pod đang chạy phiên bản của ứng dụng được định nghĩa bởi Deployment.

Kết hợp Deployment và Service trong Kubernetes cho phép bạn triển khai và cung cấp các ứng dụng một cách linh hoạt và tự động. Deployment đảm bảo rằng các Pod của ứng dụng luôn duy trì phiên bản ứng dụng mong muốn, trong khi Service cung cấp điểm cuối ổn định cho các ứng dụng trong cụm, cho phép các ứng dụng hoặc dịch vụ khác truy cập vào chúng một cách dễ dàng và nhất quán.



1. K8s Deployment manifest sẽ đưa ra hướng dẫn để triển khai các container vào một pod bên trong một trong các worker node của cụm K8s
2. K8s Service manifest sẽ đưa ra hướng dẫn để tạo một dịch vụ bên trong K8s cluster theo dõi quá trình service registration & discovery của một dịch vụ/triển khai cụ thể
3. Liên kết giữa container chạy bên trong Pod và dịch vụ sẽ được thực hiện dựa trên nhãn "app" & bộ chọn (selector) bên trong tệp khai

4. Deploying all the microservices into K8s cluster

```
C:\Windows\System32\cmd.exe

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section14\kubernetes>kubectl apply -f 1_configmaps.yml
configmap/eazybank-configmap created

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section14\kubernetes>kubectl apply -f 2_zipkin.yml
deployment.apps/zipkin-deployment created
service/zipkin-service created

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section14\kubernetes>kubectl apply -f 3_configserver.yml
deployment.apps/configserver-deployment created
service/configserver-service created

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section14\kubernetes>kubectl apply -f 4_eurekaserver.yml
deployment.apps/eurekaserver-deployment created
service/eurekaserver-service created

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section14\kubernetes>kubectl apply -f 5_accounts.yml
deployment.apps/accounts-deployment created
service/accounts-service created

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section14\kubernetes>kubectl apply -f 6_loans.yml
deployment.apps/loans-deployment created
service/loans-service created

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section14\kubernetes>kubectl apply -f 7_cards.yml
deployment.apps/cards-deployment created
service/cards-service created

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section14\kubernetes>kubectl apply -f 8_gateway.yml
deployment.apps/gatewayserver-deployment created
service/gatewayserver created

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section14\kubernetes>
```

Workloads REFRESH DEPLOY DELETE

Cluster Namespace RESET SAVE

Workloads are deployable units of computing that can be created and managed in a cluster.

OVERVIEW COST OPTIMIZATION

Filter Is system object : False Filter workloads

<input type="checkbox"/>	Name	Status	Type	Pods	Namespace	Cluster
<input type="checkbox"/>	accounts-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	cards-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	configserver-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	eurekaserver-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	gatewayserver-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	loans-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	zipkin-deployment	OK	Deployment	1/1	default	cluster-1

accounts-deployment - Deployments

console.cloud.google.com/kubernetes/deployment/us-central1-c/cluster-1/default/accounts-deployment/overview?project=microservices-346709

Google Cloud Platform Microservices Search Products, resources, docs (/)

Kubernetes Engine Deployment details REFRESH EDIT DELETE ACTIONS KUBECTL SHOW INFO PANEL OPERA

Clusters Namespace default

Workloads Labels app: accounts

Logs Container logs, Audit logs

Replicas 1 updated, 1 ready, 1 available, 0 unavailable

Pod specification Revision 1, containers: accounts

Active revisions

Revision ↓	Name	Status	Summary	Created on	Pods running/Pods total
1	accounts-deployment-66d985848	OK	accounts: eazybytes/accounts:latest	May 5, 2022, 8:25:19 PM	1/1

Managed pods

Revision	Name	Status	Restarts	Created on ↑
1	accounts-deployment-66d985848-jsq4z	Running	0	May 5, 2022, 8:25:19 PM

Exposing services

Name ↑	Type	Endpoints
accounts-service	Load balancer	35.225.173.213:8080

Marketplace

5. Problems with manually created Kubernetes manifest files- Sự cố với tệp kê khai Kubernetes được tạo thủ công

Khi triển khai nhiều microservices vào Kubernetes thủ công, có một số vấn đề phổ biến và phức tạp mà bạn có thể gặp phải. Dưới đây là một số vấn đề và khó khăn trong quá trình triển khai này:

- Số lượng tệp YAML lớn: Với nhiều microservices, bạn cần tạo nhiều tệp YAML manifest để triển khai các ứng dụng. Điều này dẫn đến sự phức tạp và khó quản lý khi số lượng tệp YAML tăng lên. Nếu bạn có hàng trăm microservices, việc quản lý hàng trăm tệp YAML có thể trở nên rất khó khăn và dễ dàng gây ra lỗi.
- Quá trình triển khai và hủy triển khai thủ công: Khi triển khai hoặc hủy triển khai các microservices, bạn phải thực hiện thủ công từng bước cho từng tệp YAML. Điều này rất tốn thời gian và dễ dàng gây ra sai sót khi áp dụng nhiều lần.
- Quản lý môi trường: Khi triển khai các ứng dụng vào nhiều môi trường khác nhau (ví dụ: development, staging, production), bạn cần tạo các tệp YAML riêng biệt cho từng môi trường. Điều này làm cho quá trình triển khai và quản lý các môi trường trở nên phức tạp.
- Quản lý phiên bản ứng dụng: Khi cần cập nhật ứng dụng lên một phiên bản mới, bạn phải sửa đổi tệp YAML tương ứng cho từng microservice. Điều này dễ dàng gây ra lỗi và tốn thời gian.
- Phân phối một số lượng lớn các thay đổi: Khi bạn muốn triển khai nhiều thay đổi cùng một lúc (ví dụ: cập nhật nhiều microservices), việc triển khai thủ công có thể trở nên phức tạp và dễ gây ra lỗi.
- Không thể tái sử dụng được: Với việc tạo thủ công các tệp YAML cho từng microservice, không thể tái sử dụng được cấu hình cho các ứng dụng tương tự hoặc các phiên bản khác nhau của cùng một ứng dụng.

Để giải quyết những vấn đề này, Helm ra đời như một công cụ quản lý gói cho Kubernetes. Helm giúp bạn quản lý, tái sử dụng và triển khai các ứng dụng trong Kubernetes một cách dễ dàng và tự động hơn. Helm cho phép bạn tạo các mô tả ứng dụng (charts) có thể tái sử dụng cho các microservices và môi trường khác nhau. Thay vì tạo thủ công từng tệp YAML, bạn có thể sử dụng các charts đã được định nghĩa sẵn và chỉ cần cung cấp giá trị cấu hình cụ thể cho mỗi ứng dụng và môi trường.

Helm cũng hỗ trợ các phiên bản của các charts, giúp bạn quản lý các thay đổi và triển khai phiên bản mới một cách tự động. Điều này giúp giảm thiểu các lỗi trong quá trình triển khai và cải thiện tính ổn định của hệ thống.

Với Helm, bạn có thể tự động triển khai hàng trăm microservices vào các môi trường khác nhau một cách dễ dàng và tiết kiệm thời gian, giúp cải thiện hiệu suất và quản lý của dự án.

Section 13: Deep Dive on Helm

1. Introduction to Helm

Helm được biết đến rộng rãi là "the package manager for K8s". Mục tiêu của Helm là cung cấp cho người dùng một cách tốt hơn để quản lý tất cả các tệp Kubernetes YAML mà tạo trên các dự án Kubernetes.

- Khi không có Helm, cần duy trì tất cả các tệp khai Kubernetes cho Deployment, Service, ConfigMap, v.v. cho mỗi microservice.
- Nếu không có Helm, nhóm Devops cần chạy tất cả các tệp K8s YAML manifest files thông qua kubectl.
- Cách mà Helm đã thực hiện để giải quyết vấn đề này là tạo Helm Chart. Mỗi Chart là một hoặc nhiều tệp Kubernetes manifest — một chart cũng có thể có chart con và chart phụ thuộc. Điều này có nghĩa là Helm cài đặt toàn bộ dependency tree của một dự án chỉ bằng một lệnh duy nhất.

2. Problems that Helm solves

Khi không có Helm, chúng ta cần duy trì các YAML files/manifest riêng biệt của K8 cho tất cả các microservice bên trong một dự án như bên dưới. Nhưng phần lớn nội dung bên trong chúng trông giống nhau ngoại trừ một vài giá trị động.

```
apiVersion: v1
kind: Service
metadata:
  name: accounts-service
spec:
  selector:
    app: accounts
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
```

accounts service manifest

```
apiVersion: v1
kind: Service
metadata:
  name: loans-service
spec:
  selector:
    app: loans
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 8090
      targetPort: 8090
```

loans service manifest

```
apiVersion: v1
kind: Service
metadata:
  name: cards-service
spec:
  selector:
    app: cards
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 9000
      targetPort: 9000
```

cards service manifest

Với Helm, chúng ta có thể tạo một template yaml mẫu như bên dưới. Chỉ các giá trị động mới được đưa vào trong quá trình thiết lập dịch vụ của K8 dựa trên các giá trị được đề cập bên trong các values.yaml có trong mỗi service/chart.

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.deploymentLabel }}
spec:
  selector:
    app: {{ .Values.deploymentLabel }}
  type: {{ .Values.service.type }}
  ports:
    - protocol: TCP
      port: {{ .Values.service.port }}
      targetPort: {{ .Values.service.targetPort }}
```

```
deploymentLabel: accounts
service:
  type: ClusterIP
  port: 8080
  targetPort: 8080
```

values.yaml

Helm service template

HELM SUPPORTS PACKAGING OF YAML FILES



Với sự trợ giúp của Helm, có thể đóng gói tất cả các YAML manifest file thuộc về một ứng dụng thành **Chart**. Điều tương tự có thể được phân phối vào các public hoặc private repository.



Helm đóng vai trò là người quản lý gói cho K8s(package manager). Giống như package manager là tập hợp các công cụ phần mềm tự động hóa quá trình cài đặt, nâng cấp, quản lý phiên bản và xóa chương trình máy tính cho máy tính một cách nhất quán. Tương tự, Helm tự động hóa quá trình cài đặt, khôi phục, nâng cấp nhiều tệp K8s manifest với một lệnh duy nhất.

HELM SUPPORTS EASIER INSTALLATION



Với sự trợ giúp của Helm, chúng ta có thể set up/upgrade/rollback/remove toàn bộ ứng dụng microservice vào K8s cluster chỉ bằng 1 lệnh. Không cần chạy lệnh áp dụng kubectl theo cách thủ công cho từng manifest file (từng microservice)

HELM SUPPORTS RELEASE/VERSION MANAGEMENT



Helm tự động duy trì lịch sử phiên bản của các tệp manifest đã cài đặt. Do đó, việc khôi phục toàn bộ cụm K8 về trạng thái làm việc trước đó chỉ bằng một lệnh duy nhất.

3. Installing Helm

<https://helm.sh/docs/intro/install/>

4. Creating our first Helm Chart

Sử dụng lệnh dưới đây để tạo một Helm Chart mới có tên là "eazybank-common":

```
helm create eazybank-common
```

helm create <chart-name> sẽ tạo cấu trúc thư mục biểu đồ như bên dưới.

Thư mục eazybank-common là tên của chart đã cung cấp khi tạo biểu đồ.

eazybank-common/
|---Chart.yaml
|---values.yaml
|---charts/
|---templates/

Chart.yaml sẽ có thông tin meta về helm.
values.yaml sẽ có các giá trị động (dynamic) cho chart

charts folder sẽ có các chart khác mà chart hiện tại phụ thuộc vào.

templates folder chứa các tệp manifest template yaml

5. Installing the Default Helm chart into K8s cluster

Kết nối với cụm Kubernetes

Sử dụng kubectl để kết nối với cụm Kubernetes. Để thực hiện việc này, chạy lệnh sau và làm theo hướng dẫn:

```
gcloud container clusters get-credentials CLUSTER_NAME --zone ZONE --project PROJECT_ID
```

Cài đặt Default Helm Chart

Cuối cùng, bạn có thể cài đặt Default Helm Chart vào cụm Kubernetes. Như đã đề cập ở trên, bạn có thể sử dụng các Chart mẫu có sẵn hoặc tùy chỉnh các Chart theo yêu cầu của ứng dụng.

```
helm install sample-deployment eazybank-common
```

Trong đó:

- "sample-deployment" là tên của release (phiên bản triển khai) bạn muốn tạo.
- "eazybank-common" là tên của Helm Chart mà bạn muốn cài đặt.

```

C:\Windows\System32\cmd.exe
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm>helm install sample-deployment eazybank-common
NAME: sample-deployment
LAST DEPLOYED: Sun May  8 21:19:12 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=eazybank-common,app.kubernetes.io/instance=sample-deployment" -o jsonpath="{.items[0].metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath=".spec.containers[0].ports[0].containerPort")
  echo "Visit http://127.0.0.1:$CONTAINER_PORT to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT

```

The screenshot shows the Microservices dashboard with the 'Services & Ingress' tab selected. The table lists a single service entry:

	Name	Status	Type	Endpoints	Pods	Namespace	Clusters
<input type="checkbox"/>	sample-deployment-eazybank-common	OK	Cluster IP	10.8.10.157	1/1	default	cluster-1

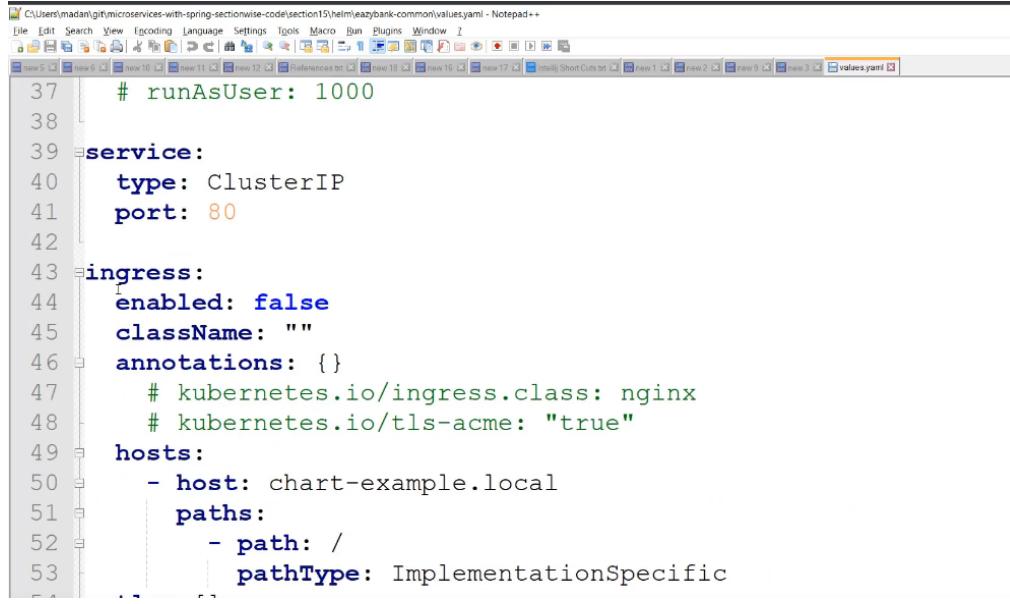
Ingress là một tài nguyên trong Kubernetes, được sử dụng để quản lý truy cập bên ngoài vào các dịch vụ trong cụm Kubernetes. Nó cho phép bạn định tuyến các yêu cầu HTTP và HTTPS đến các dịch vụ trong cụm dựa trên các quy tắc và đường dẫn xác định. Ingress hoạt động như một bộ định tuyến bên ngoài cho cụm Kubernetes, giúp dễ dàng mở rộng và quản lý việc truy cập vào các ứng dụng trong cụm.

Khi bạn triển khai một ứng dụng trong Kubernetes bằng Helm, bạn có thể bao gồm các tài nguyên Ingress trong Chart của mình. Điều này cho phép bạn định nghĩa cách truy cập bên ngoài vào ứng dụng của mình thông qua tài nguyên Ingress trong khi cài đặt ứng dụng bằng Helm.

The screenshot shows the Microservices dashboard with the 'INGRESS' tab selected. A large callout box provides information about Ingresses:

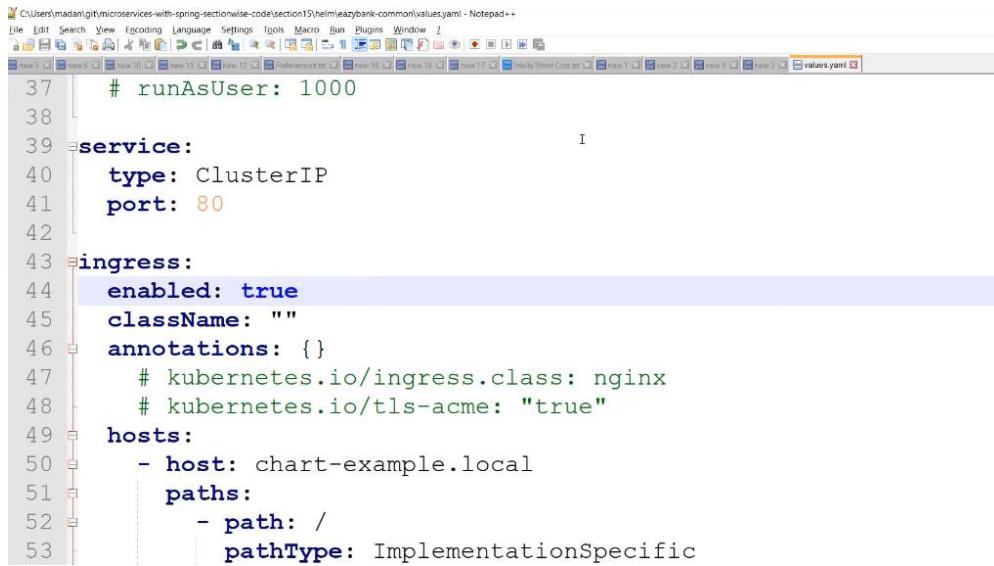
Kubernetes Engine
Discovery & load balancing
 Kubernetes Ingresses are collections of rules for routing external HTTP(S) traffic to Services. You can create Ingresses using the Services tab.

[LEARN MORE](#) [GO TO SERVICES](#)



```
37 # runAsUser: 1000
38
39 service:
40   type: ClusterIP
41   port: 80
42
43 ingress:
44   enabled: false
45   className: ""
46   annotations: {}
47     # kubernetes.io/ingress.class: nginx
48     # kubernetes.io/tls-acme: "true"
49 hosts:
50   - host: chart-example.local
51     paths:
52       - path: /
53         pathType: ImplementationSpecific
```

- Chuyển enabled: true



```
37 # runAsUser: 1000
38
39 service:
40   type: ClusterIP
41   port: 80
42
43 ingress:
44   enabled: true
45   className: ""
46   annotations: {}
47     # kubernetes.io/ingress.class: nginx
48     # kubernetes.io/tls-acme: "true"
49 hosts:
50   - host: chart-example.local
51     paths:
52       - path: /
53         pathType: ImplementationSpecific
```

```
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm>helm upgrade sample-deployment eazybank-common
Release "sample-deployment" has been upgraded. Happy Helm-ing!
NAME: sample-deployment
LAST DEPLOYED: Sun May  8 21:25:24 2022
NAMESPACE: default
STATUS: deployed
REVISION: 2
NOTES:
1. Get the application URL by running these commands:
  http://chart-example.local/
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm>
```

Name	Status	Type	Frontends	Services	Namespace	Clusters
sample-deployment-eazybank-common	OK	External HTTP(S) LB	chart-example.local/	sample-deployment-eazybank-common	default	cluster-1

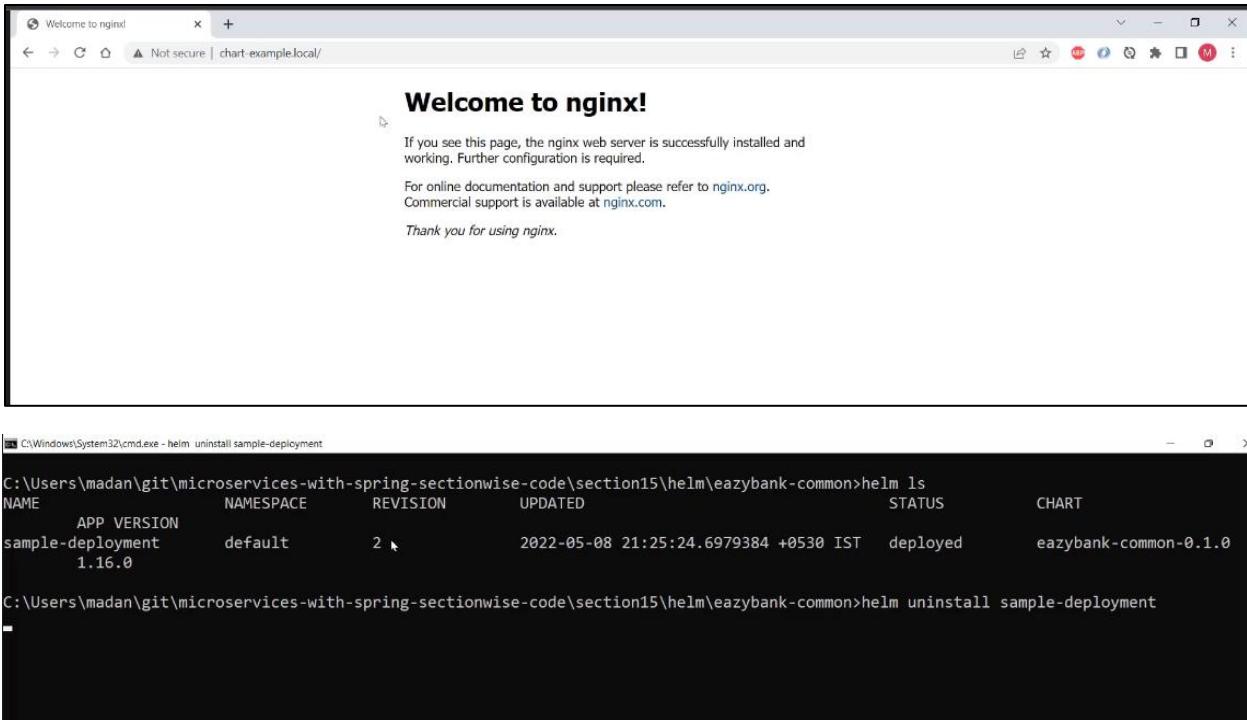
- Bạn có thể truy cập Charts thông qua font-end. Nhưng cần setting file host

Annotation	Value
ingress.kubernetes.io/backends	{"k8s-be-31699--5e9b59985b33cce5": "HEALTHY", "k8s1-5e9b5998-default-sample-deplc9b3cd4b2": "HEALTHY"}
ingress.kubernetes.io/forwarding-rule	k8s2-fr-jy0ik32a-default-sample-deployment-eazybank-co-1y0718to
ingress.kubernetes.io/target-proxy	k8s2-tp-jy0ik32a-default-sample-deployment-eazybank-co-1y0718to
ingress.kubernetes.io/url-map	k8s2-um-jy0ik32a-default-sample-deployment-eazybank-co-1y0718to
meta.helm.sh/release-name	sample-deployment
meta.helm.sh/release-namespace	default

```

14 # For example:
15 #
16 #      102.54.94.97      rhino.acme.com      # sou
17 #      38.25.63.10      x.acme.com          # x c
18 # localhost name resolution is handled within DNS its
19 #      127.0.0.1      localhost
20 #      ::1              localhost
21 # 127.0.0.1 kubernetes.docker.internal
22 # Added by Docker Desktop
23 192.168.0.136 host.docker.internal
24 192.168.0.136 gateway.docker.internal
25 # To allow the same kube context to work on the host
26 127.0.0.1 kubernetes.docker.internal
27 # End of section
28 34.149.177.220 chart-example.local/

```



6. Exploring the default Helm chart content- Khám phá nội dung default Helm chart

Khi bạn tạo một Helm Chart mới bằng lệnh helm create, nó sẽ tạo ra một cấu trúc thư mục mẫu với một số tệp mẫu và cấu hình mặc định. Hãy khám phá nội dung mặc định của một Helm Chart:

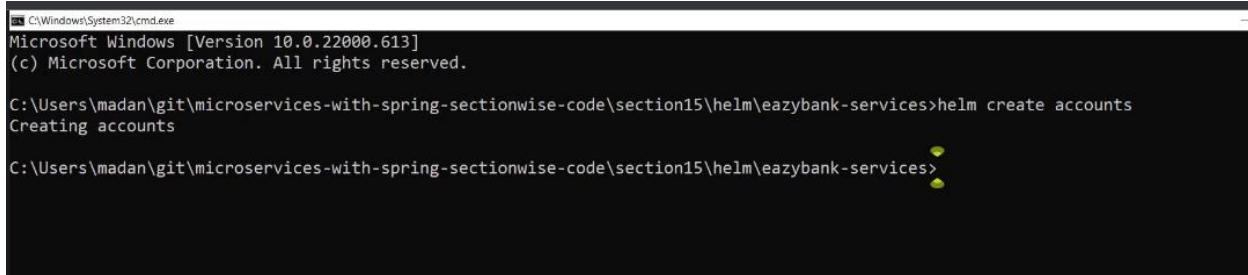
- **Chart.yaml:** Tệp này chứa các thông tin về Helm Chart, bao gồm tên Chart, phiên bản, mô tả và các chi tiết khác.
- **values.yaml:** Tệp này chứa các giá trị cấu hình mặc định cho Chart. Bạn có thể ghi đè lên các giá trị này bằng cách cung cấp tệp values.yaml riêng của bạn khi cài đặt Chart.
- **charts/:** Thư mục này được sử dụng để lưu trữ các Chart phụ thuộc mà Chart chính của bạn phụ thuộc vào. Nếu Chart chính có các phụ thuộc, chúng sẽ được tải xuống và giải nén trong thư mục này.
- **templates/:** Thư mục này chứa các tệp mẫu cho các tài nguyên Kubernetes. Các tệp mẫu này sử dụng ngôn ngữ mẫu Go để tạo các tệp YAML dựa trên cấu hình được chỉ định trong values.yaml. Các tệp mẫu mặc định bao gồm:
 - deployment.yaml: Một tệp mẫu để tạo tài nguyên Kubernetes Deployment.
 - service.yaml: Một tệp mẫu để tạo tài nguyên Kubernetes Service.
 - ingress.yaml: Một tệp mẫu để tạo tài nguyên Kubernetes Ingress.
 - NOTES.txt: Tệp này chứa các ghi chú hữu ích sẽ được hiển thị sau khi Chart được cài đặt.
- **templates/_helpers.tpl:** Đây là một tệp trợ giúp được sử dụng bởi các tệp mẫu trong thư mục templates/. Nó có thể chứa các chức năng và định nghĩa mẫu Go có thể tái sử dụng.
- **templates/tests/:** Thư mục này được sử dụng để lưu trữ các tệp kiểm tra cho Chart. Nó có thể bao gồm các tệp YAML với các trường hợp kiểm tra để xác minh tính chính xác của Chart trong quá trình phát triển.

Khi bạn tạo một Helm Chart bằng `helm create`, nội dung mặc định là một điểm khởi đầu mà bạn có thể tùy chỉnh dựa trên yêu cầu cụ thể của ứng dụng. Bạn có thể thay đổi các tệp mẫu, thêm nhiều tài nguyên Kubernetes khác và định nghĩa thêm các giá trị trong `values.yaml` để phù hợp với nhu cầu của mình.

Hãy nhớ rằng Helm rất linh hoạt và cho phép bạn tạo các gói có thể tái sử dụng, có tham số và kiểm soát phiên bản cho các triển khai Kubernetes của bạn. Nội dung mặc định của Helm Chart được thiết kế như một điểm khởi đầu, và bạn có thể điều chỉnh nó khi yêu cầu của ứng dụng và hạ tầng thay đổi.

7. Creating Helm chart for Accounts microservice

Bước 1: Tạo Helm chart

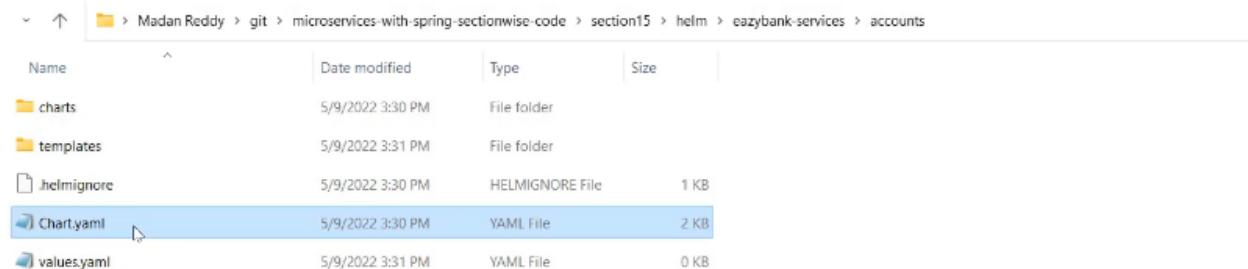


```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.613]
(c) Microsoft Corporation. All rights reserved.

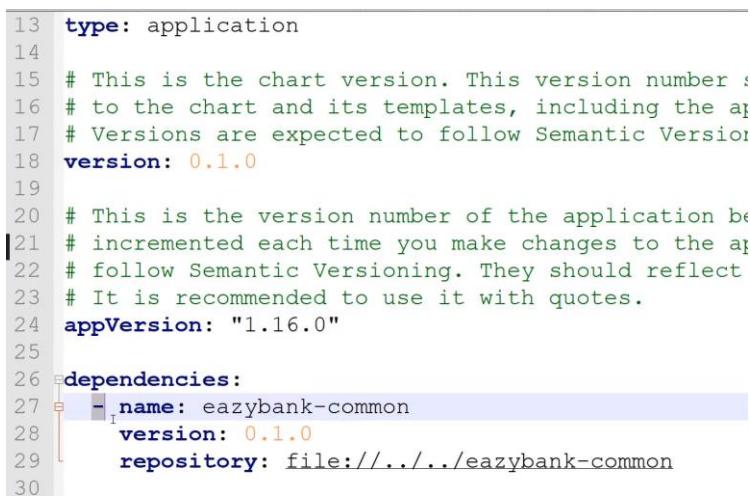
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\eazybank-services>helm create accounts
Creating accounts

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\eazybank-services>
```

- Thêm phụ thuộc vào Chart:



Name	Date modified	Type	Size
charts	5/9/2022 3:30 PM	File folder	
templates	5/9/2022 3:31 PM	File folder	
helmignore	5/9/2022 3:30 PM	HELMIGNORE File	1 KB
Chart.yaml	5/9/2022 3:30 PM	YAML File	2 KB
values.yaml	5/9/2022 3:31 PM	YAML File	0 KB



```
13 type: application
14
15 # This is the chart version. This version number :
16 # to the chart and its templates, including the ap
17 # Versions are expected to follow Semantic Versioni
18 version: 0.1.0
19
20 # This is the version number of the application be
21 # incremented each time you make changes to the ap
22 # follow Semantic Versioning. They should reflect
23 # It is recommended to use it with quotes.
24 appVersion: "1.16.0"
25
26 dependencies:
27 - name: eazybank-common
28   version: 0.1.0
29   repository: file://.../eazybank-common
30
```

Mục đích là xác định các phụ thuộc (dependencies) của chart. Cụ thể, đoạn mã này đang định nghĩa một phụ thuộc tên eazybank-common có phiên bản 0.1.0, được lấy từ một repository cục bộ (file://..../eazybank-common).

Khi bạn triển khai chart hiện tại (được định nghĩa trong file Chart.yaml) mà có phụ thuộc eazybank-common, Helm sẽ tự động tải và triển khai cả hai chart. Điều này giúp bạn quản lý phụ thuộc và đơn giản hóa việc triển khai ứng dụng có sử dụng các chart chung hoặc tái sử dụng trong các dự án khác nhau.

Nếu bạn muốn sử dụng phụ thuộc trong chart của mình, bạn cần chắc chắn rằng chart eazybank-common đã được đóng gói và có sẵn trong repository hoặc đường dẫn đã định nghĩa.

Bước 2: Tạo các tệp mẫu Kubernetes

Thư mục "templates" chứa các tệp mẫu Kubernetes (ví dụ: tệp YAML) cho các tài nguyên mà bạn muốn triển khai trong Kubernetes. Những tệp mẫu này chứa các đối tượng Kubernetes như Deployment, Service, ConfigMap, Secret, ServiceAccount, và nhiều tài nguyên khác.

Tạo hai file là deployment.yaml và service.yaml

- **deployment.yaml:**

Tệp deployment.yaml được sử dụng để định nghĩa một tài nguyên Deployment trong Kubernetes. Deployment là một tài nguyên quan trọng trong Kubernetes, cho phép bạn triển khai và quản lý các bản sao của các ứng dụng trên cụm Kubernetes.
Trong tệp deployment.yaml, bạn sẽ xác định các thông số của Deployment như số lượng bản sao (replicas), thông tin về container và ảnh (containers và image), các cổng và giao thức mà ứng dụng lắng nghe (ports), cài đặt sẵn các biến môi trường và tài nguyên cần thiết khác để triển khai ứng dụng của bạn.

- **service.yaml:**

Tệp service.yaml được sử dụng để định nghĩa một tài nguyên Service trong Kubernetes. Service là một tài nguyên quan trọng giúp các Pod (các bản sao của ứng dụng) trong cụm Kubernetes có thể tương tác và truy cập lẫn nhau.
Trong tệp service.yaml, bạn sẽ xác định loại dịch vụ (Service Type), cổng dịch vụ và cổng Pod được liên kết (ports), các cặp nhãn (labels) để chọn các Pod mục tiêu và kiểu cơ chế cân bằng tải nếu có (selector và type). Dịch vụ sẽ cung cấp một địa chỉ IP tĩnh hoặc tên miền (tùy thuộc vào loại dịch vụ) để ứng dụng có thể truy cập từ bên ngoài cụm Kubernetes.

Bước 3: xác định các giá trị cần thiết cho template

values.yaml

```
deploymentName: accounts-deployment
deploymentLabel: accounts
appName: accounts

replicaCount: 1

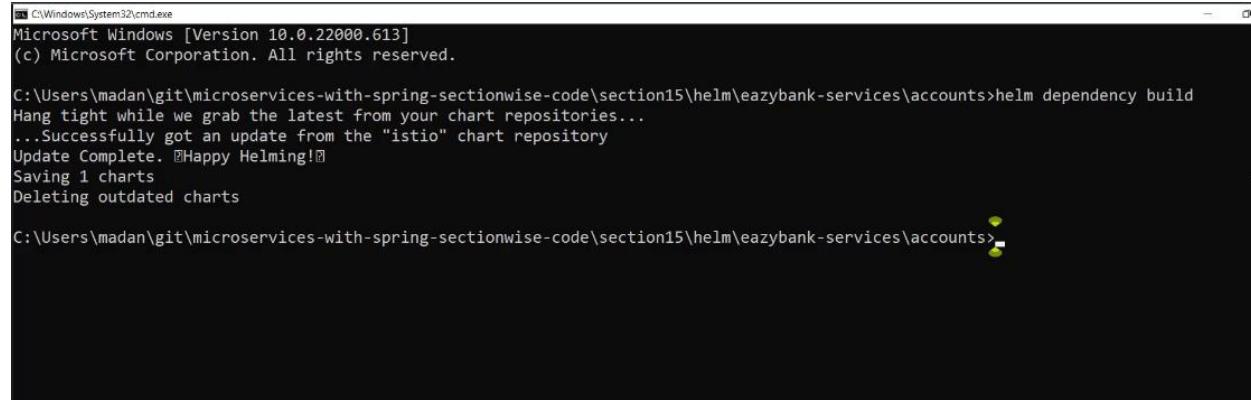
image:
  repository: eazybytes/accounts
  tag: latest
```

```
containerPort: 8080

service:
  type: LoadBalancer
  port: 8080
  targetPort: 8080

config_enabled: true
zipkin_enabled: true
profile_enabled: true
eureka_enabled: true
appname_enabled: true
```

Bước 4: Compile my chart



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.613]
(c) Microsoft Corporation. All rights reserved.

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\easybank-services\accounts>helm dependency build
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "istio" chart repository
Update Complete. Happy Helming!
Saving 1 charts
Deleting outdated charts

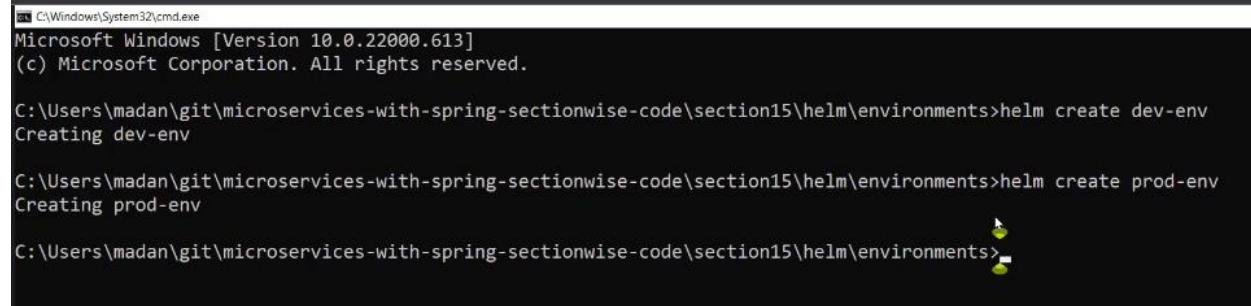
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\easybank-services\accounts>
```

=> Tương tự tạo Helm chart cho các microservice khác

8. Creating Helm chart for Dev and Prod environment

Môi trường phát triển và môi trường sản xuất thường có các yêu cầu và cấu hình khác nhau. Bằng cách tạo Helm chart riêng cho mỗi môi trường, bạn có thể tách biệt rõ ràng giữa các cấu hình và đảm bảo rằng các thay đổi trong môi trường Dev không làm ảnh hưởng đến môi trường Prod và ngược lại.

Bước 1: Tạo Helm chart cho hai môi trường:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.613]
(c) Microsoft Corporation. All rights reserved.

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm create dev-env
Creating dev-env

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm create prod-env
Creating prod-env

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>
```

Bước 2: Cấu hình Chart, values, template (product và dev)

Chart.yaml (thêm các phụ thuộc)

```
dependencies:
  - name: eazybank-common
    version: 0.1.0
    repository: file://.../eazybank-common

  - name: zipkin
    version: 0.1.0
    repository: file://.../eazybank-services/zipkin

  - name: configserver
    version: 0.1.0
    repository: file://.../eazybank-services/configserver

  - name: eurekaserver
    version: 0.1.0
    repository: file://.../eazybank-services/eurekaserver

  - name: accounts
    version: 0.1.0
    repository: file://.../eazybank-services/accounts

  - name: cards
    version: 0.1.0
    repository: file://.../eazybank-services/cards

  - name: loans
    version: 0.1.0
    repository: file://.../eazybank-services/loans

  - name: gatewayserver
    version: 0.1.0
    repository: file://.../eazybank-services/gatewayserver
```

values.yaml

```
global:
  configMapName: eazybankprod-configmap
  zipkinBaseURL: http://zipkin:9411/api/v2/spans
  activeProfile: prod
  configServerURL: configserver:http://configserver:8071/
  eurekaServerURL: http://eurekaserver:8070/eureka/
```

configmap.yaml

```
{ { - template "common.configmap" . - } }
```

```
C:\Windows\System32\cmd.exe

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments\dev-env>helm dependencies build
Error: directory ..\..\eazybank-services\config-server not found

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments\dev-env>helm dependencies build
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "istio" chart repository
Update Complete. 🎉Happy Helm-ing!🎉
Saving 8 charts
Deleting outdated charts

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments\dev-env>cd ..

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>cd prod-env

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments\prod-env>helm dependencies build
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "istio" chart repository
Update Complete. 🎉Happy Helm-ing!🎉
Saving 8 charts
Deleting outdated charts

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments\prod-env>
```

9. Installing Helm charts into K8s cluster

Kết nối với cụm Kubernetes

Sử dụng kubectl để kết nối với cụm Kubernetes. Để thực hiện việc này, chạy lệnh sau và làm theo hướng dẫn:

```
gcloud container clusters get-credentials CLUSTER_NAME --zone ZONE --project PROJECT_ID
```

Install helm chart

```
C:\Windows\System32\cmd.exe

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm ls
NAME      NAMESPACE      REVISION      UPDATED STATUS   CHART      APP VERSION
NAME: dev-deployment
LAST DEPLOYED: Mon May  9 21:07:17 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>
```

Google Cloud Platform Microservices

Kubernetes Engine Configuration

REFRESH DELETE

Clusters Workloads Services & Ingress Applications Secrets & ConfigMaps Storage Object Browser Migrate to containers Backup for GKE Config Management

Secrets are sensitive pieces of information, such as passwords, keys, and tokens. ConfigMaps are designed to store information that is not sensitive, such as environment variables, command-line arguments, and configuration files.

Secrets respect access control and are not visible to users without read permissions

Filter Is system object : False Filter secrets and config maps

Name ↑	Type	Namespace	Cluster
default-token-tx2wz	Secret	kube-node-lease	cluster-1
eazybankdev-configmap	Config Map	default	cluster-1
kube-root-ca.crt	Config Map	default	cluster-1
kube-root-ca.crt	Config Map	kube-public	cluster-1
kube-root-ca.crt	Config Map	kube-node-lease	cluster-1
sh.helm.release.v1.dev-deployment.v1	Secret	default	cluster-1

Google Cloud Platform Microservices

Kubernetes Engine Config map details

REFRESH EDIT DELETE KUBECTL

Clusters Workloads Services & Ingress Applications Secrets & ConfigMaps Storage Object Browser Migrate to containers Backup for GKE Config Management

eazybankdev-configmap

DETAILS YAML

Cluster	cluster-1
Namespace	default
Created	May 9, 2022, 9:07:23 PM
Labels	app.kubernetes.io/managed-by: Helm
Annotations	meta.helm.sh/release-name: dev-deployment meta.helm.sh/release-namespace: default

Data

EUREKA_CLIENT_SERVICEURL_DEFAULTZONE	http://eurekaserver:8070/eureka/
SPRING_CONFIG_IMPORT	configserver:http://configserver:8071/
SPRING_PROFILES_ACTIVE	dev
SPRING_ZIPKIN_BASEURL	http://zipkin:9411/

Name	Status	Type	Endpoints	Pods	Namespace	Clusters
accounts	OK	External load balancer	34.121.106.31:8080	1/1	default	cluster-1
cards	OK	External load balancer	34.66.216.179:9000	1/1	default	cluster-1
configserver	OK	External load balancer	34.68.173.232:8071	1/1	default	cluster-1
eurekaserver	OK	External load balancer	35.232.247.230:8070	1/1	default	cluster-1
gatewayserver	OK	External load balancer	173.255.115.109:8072	1/1	default	cluster-1
loans	OK	External load balancer	35.226.226.219:8090	1/1	default	cluster-1
zipkin	OK	External load balancer	104.197.130.115:9411	1/1	default	cluster-1

10. Demo of helm upgrade command

- Thay đổi cấu hình của microservice

```

values.yaml - Notepad
File Edit View

# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

deploymentName: cards-deployment
deploymentLabel: cards

replicaCount: 2

image:
  repository: eazybytes/cards
  tag: latest

containerPort: 9000

service:
  type: LoadBalancer
  port: 9000
  targetPort: 9000

config_enabled: true
zipkin_enabled: true
profile_enabled: true
eureka_enabled: true

```

- Build lại các phụ thuộc:

```
C:\Windows\System32\cmd.exe
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm ls
NAME      NAMESPACE      REVISION      UPDATED STATUS      CHART      APP VERSION
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm install dev-deployment dev-env
NAME: dev-deployment
LAST DEPLOYED: Mon May  9 21:07:17 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>cd dev-env

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments\dev-env>helm dependencies build
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "istio" chart repository
Update Complete. Happy Helming!
Saving 8 charts
Deleting outdated charts

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments\dev-env>cd..

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>cd prod-env

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments\prod-env>helm dependencies build
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "istio" chart repository
Update Complete. Happy Helming!
Saving 8 charts
Deleting outdated charts
```

- Thực hiện upgrade:

```
C:\Windows\System32\cmd.exe
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments\prod-env>helm ls
NAME      NAMESPACE      REVISION      UPDATED STATUS      CHART      APP VERSION
dev-deployment  default      1          2022-05-09 21:07:17.2267414 +0530 IST  deployed  dev-env-0.1.0  1.16.0

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments\prod-env>cd..

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm upgrade dev-deployment dev-env
Release "dev-deployment" has been upgraded. Happy Helming!
NAME: dev-deployment
LAST DEPLOYED: Mon May  9 21:27:17 2022
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>
```

Name	Status	Type	Pods	Namespace	Cluster
accounts-deployment	OK	Deployment	2/2	default	cluster-1
cards-deployment	OK	Deployment	2/2	default	cluster-1
configserver-deployment	OK	Deployment	1/1	default	cluster-1
eurekaserver-deployment	OK	Deployment	1/1	default	cluster-1
gatewayserver-deployment	OK	Deployment	1/1	default	cluster-1
loans-deployment	OK	Deployment	2/2	default	cluster-1
zipkin-deployment	OK	Deployment	1/1	default	cluster-1

11. Demo of helm history and rollback commands

```
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm history dev-deployment
REVISION      UPDATED             STATUS        CHART          APP VERSION   DESCRIPTION
1            Mon May  9 21:07:17 2022  superseded   dev-env-0.1.0  1.16.0        Install complete
2            Mon May  9 21:27:17 2022  deployed     dev-env-0.1.0  1.16.0        Upgrade complete
```

```
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm rollback dev-deployment 1
Rollback was a success! Happy Helming!
```

```
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>
```

```
C:\Windows\System32\cmd.exe
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm ls
NAME        NAMESPACE    REVISION    UPDATED             STATUS        CHART          APP VERSION
dev-deployment  default      3          2022-05-09 21:34:10.0136724 +0530 IST  deployed   dev-env-0.1.0  1.16.0

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm history dev-deployment
REVISION      UPDATED             STATUS        CHART          APP VERSION   DESCRIPTION
1            Mon May  9 21:07:17 2022  superseded   dev-env-0.1.0  1.16.0        Install complete
2            Mon May  9 21:27:17 2022  superseded   dev-env-0.1.0  1.16.0        Upgrade complete
3            Mon May  9 21:34:10 2022  deployed     dev-env-0.1.0  1.16.0        Rollback to 1

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>
```

Name	Status	Type	Pods	Namespace	Cluster
accounts-deployment	OK	Deployment	1/1	default	cluster-1
cards-deployment	OK	Deployment	1/1	default	cluster-1
configserver-deployment	OK	Deployment	1/1	default	cluster-1
eurekaserver-deployment	OK	Deployment	1/1	default	cluster-1
gatewayserver-deployment	OK	Deployment	1/1	default	cluster-1
loans-deployment	OK	Deployment	1/1	default	cluster-1
zipkin-deployment	OK	Deployment	1/1	default	cluster-1

12. Demo of helm uninstall command

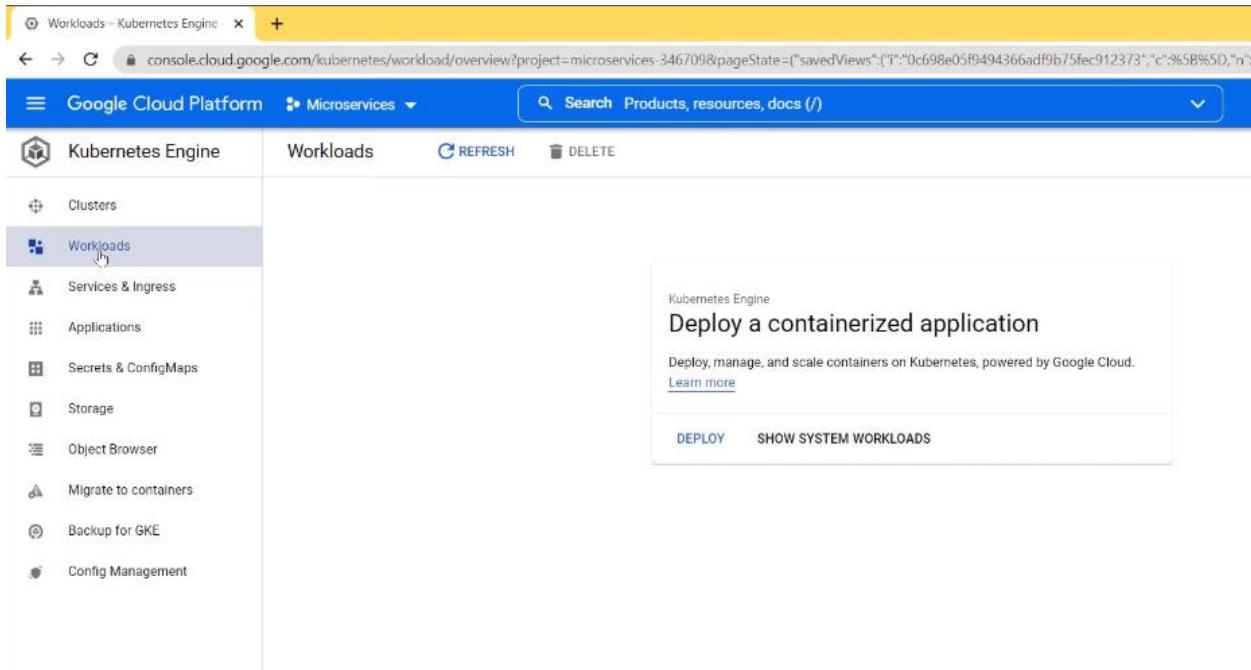
```
C:\Windows\System32\cmd.exe
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm ls
NAME      NAMESPACE   REVISION   UPDATED     STATUS      CHART          APP VERSION
dev-deployment  default      3        2022-05-09 21:34:10.0136724 +0530 IST  deployed  dev-env-0.1.0  1.16.0

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm history dev-deployment
REVISION  UPDATED      STATUS      CHART          APP VERSION  DESCRIPTION
1        Mon May 9 21:07:17 2022  superseded  dev-env-0.1.0  1.16.0    Install complete
2        Mon May 9 21:27:17 2022  superseded  dev-env-0.1.0  1.16.0    Upgrade complete
3        Mon May 9 21:34:10 2022  deployed   dev-env-0.1.0  1.16.0    Rollback to 1

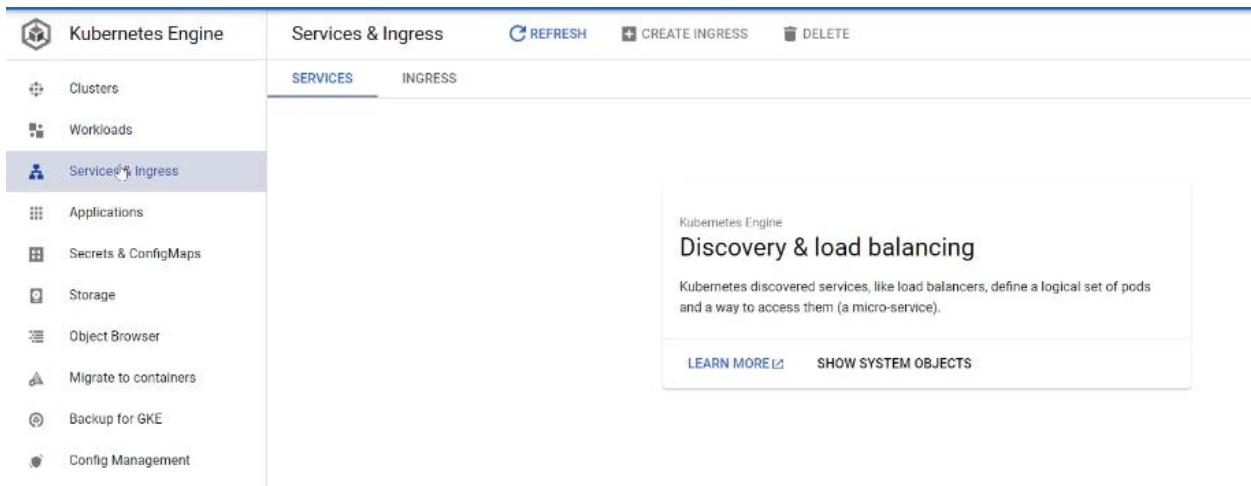
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm uninstall dev-deployment
release "dev-deployment" uninstalled

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>helm ls
NAME      NAMESPACE   REVISION   UPDATED STATUS  CHART          APP VERSION

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section15\helm\environments>
```



The screenshot shows the Google Cloud Platform interface for the Kubernetes Engine. The left sidebar has a 'Kubernetes Engine' section with various options: Clusters, Workloads (which is selected and highlighted in blue), Services & Ingress, Applications, Secrets & ConfigMaps, Storage, Object Browser, Migrate to containers, Backup for GKE, and Config Management. The main right panel is titled 'Deploy a containerized application' under 'Kubernetes Engine'. It contains a brief description: 'Deploy, manage, and scale containers on Kubernetes, powered by Google Cloud.' with a 'Learn more' link. Below the description are two buttons: 'DEPLOY' and 'SHOW SYSTEM WORKLOADS'.



The screenshot shows the Google Cloud Platform interface for the Services & Ingress section under 'Kubernetes Engine'. The left sidebar has a 'Kubernetes Engine' section with various options: Clusters, Workloads (which is selected and highlighted in blue), Services & Ingress (which is also highlighted in blue), Applications, Secrets & ConfigMaps, Storage, Object Browser, Migrate to containers, Backup for GKE, and Config Management. The main right panel is titled 'Discovery & load balancing' under 'Kubernetes Engine'. It contains a brief description: 'Kubernetes discovered services, like load balancers, define a logical set of pods and a way to access them (a micro-service).' with a 'LEARN MORE' link. Below the description are two buttons: 'LEARN MORE' and 'SHOW SYSTEM OBJECTS'.

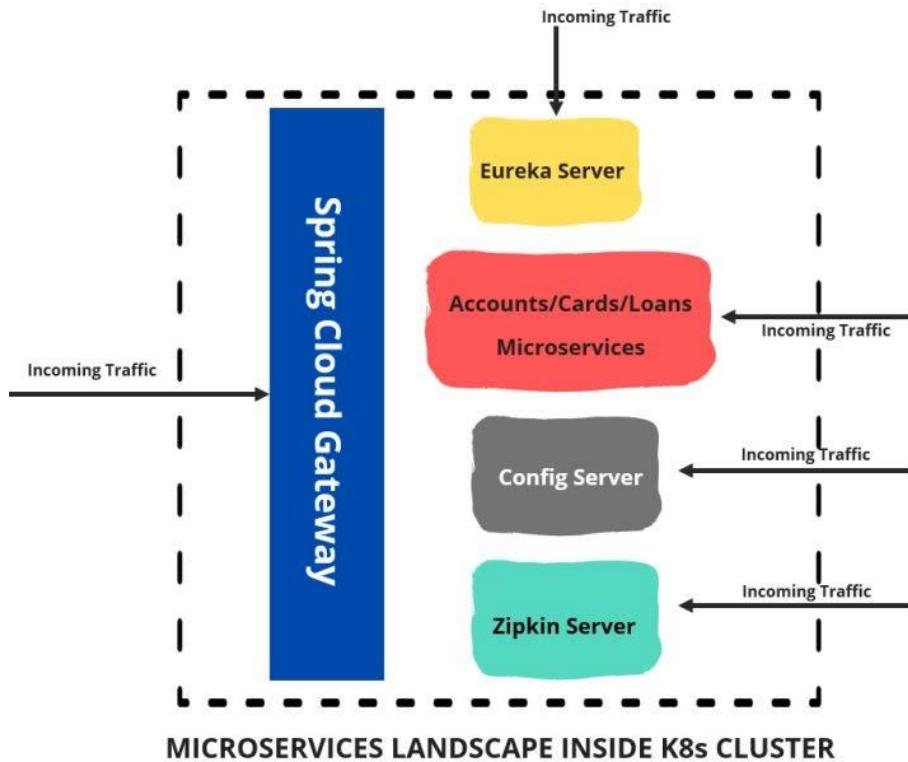
13. Revision of important helm commands

Helm commands

Command	Description
helm create eazybank	Create a blank chart with the name eazybank. Inside the eazybank folder, we can see Charts.yaml, values.yaml, Charts folder, templates folder etc.
helm dependencies build	Build is used to reconstruct a chart's dependencies
helm install [NAME] [CHART]	Install the manifests mentioned in the [CHART] with a given release name inside [NAME]
helm upgrade [NAME] [CHART]	Upgrades a specified release to a new version of a chart
helm history [NAME]	Prints historical revisions for a given release.
helm rollback [NAME] [REVISION]	Roll back a release to a previous revision. The first argument of the rollback command is the name of a release, and the second is a revision (version) number. If this argument is omitted, it will roll back to the previous release.
helm uninstall [NAME]	Removes all of the resources associated with the last release of the chart as well as the release history
helm template [NAME] [CHART]	Render chart templates locally along with the values and display the output.
helm list	This command lists all of the releases for a specified namespace

Section 14: Securing Microservices using K8s Service

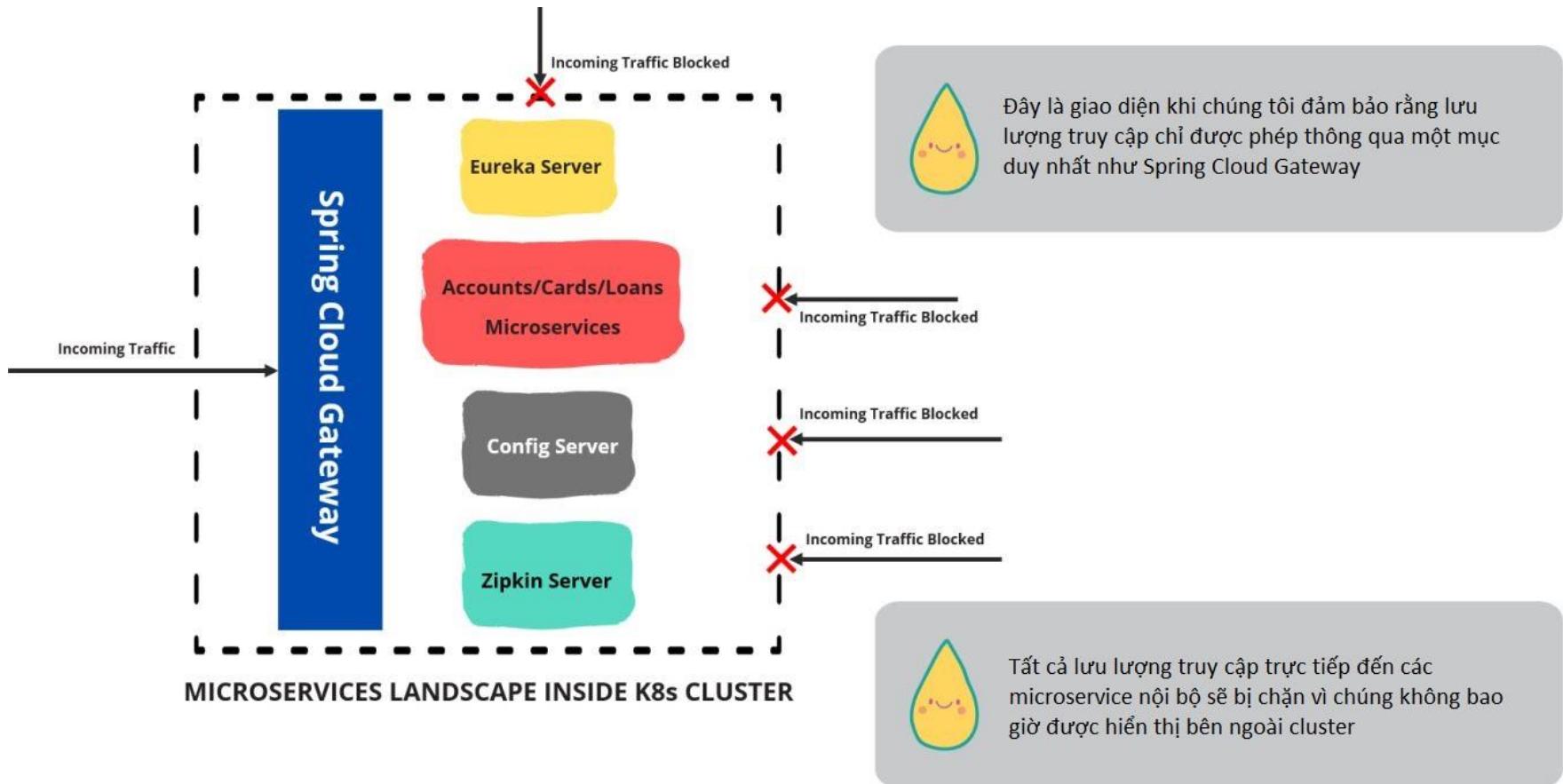
1. Problem with Kubernetes LoadBalancer Service



Hiện tại với K8s LoadBalancer Service, tất cả các microservice của được tiếp xúc với bên ngoài. Đây không phải là một thiết lập được khuyến nghị trong product.

Tất cả lưu lượng truy cập vào K8s cluster chỉ cần đến từ một mục duy nhất như Spring Cloud Gateway hoặc thành phần K8s Ingress.

Là bước đầu tiên hướng tới microservices security, chúng ta không nên để lộ microservice của mình ra công chúng và tất cả hoạt động giao tiếp sẽ diễn ra thông qua API Gateway.



2. Introduction to types of K8s Services

Dưới đây là 3 loại Dịch vụ Kubernetes được sử dụng chủ yếu bên trong các cụm K8.

ClusterIP Service

Đây là dịch vụ mặc định sử dụng Cluster IP nội bộ để hiển thị các Pods. Trong ClusterIP, các dịch vụ không khả dụng đối với quyền truy cập từ bên ngoài của cluster và được sử dụng để liên lạc nội bộ giữa các Pods hoặc microservice khác nhau trong cluster.

NodePort Service

Dịch vụ này hiển thị bên ngoài và cho phép lưu lượng truy cập bên ngoài kết nối với K8s Pods thông qua node port là cổng được mở ở cuối Node. Các Pod có thể được truy cập từ bên ngoài bằng cách sử dụng <Nodelp>:<Nodeport>

LoadBalancer Service

Dịch vụ này được hiển thị giống như trong NodePort nhưng tạo bộ cân bằng tải trong đám mây nơi K8s đang chạy để nhận các yêu cầu bên ngoài đối với dịch vụ. Sau đó, nó sẽ phân phối chúng giữa các cluster node bằng NodePort.

3. Deep dive on ClusterIP Service

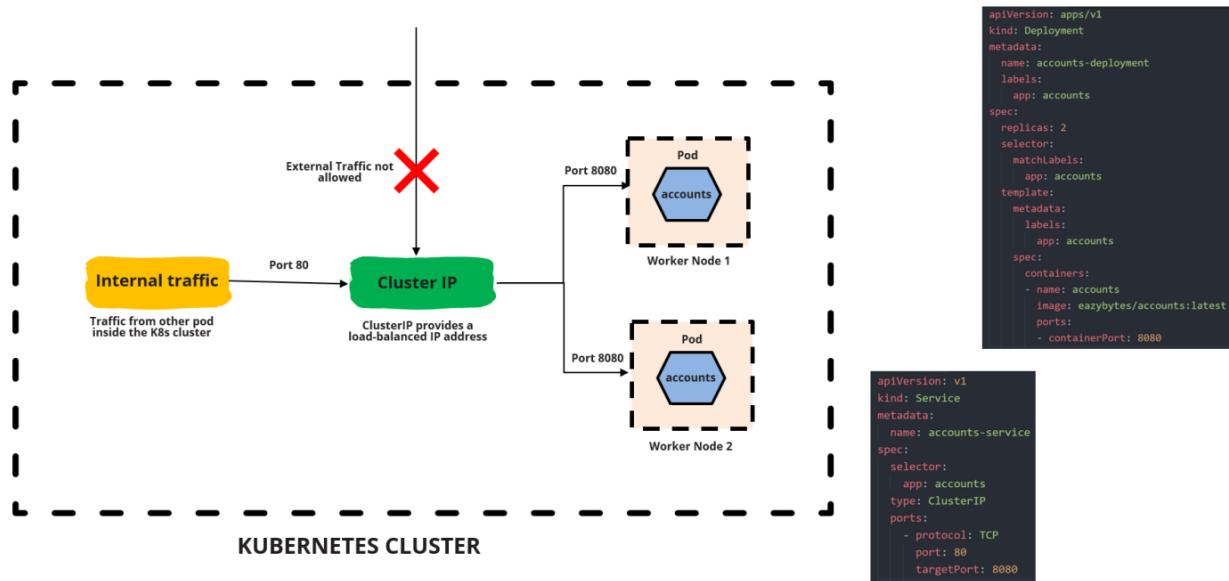
Dịch vụ ClusterIP là một trong các loại dịch vụ mạng trong Kubernetes, một hệ thống quản lý container phổ biến. ClusterIP Service cung cấp một địa chỉ IP ảo (Virtual IP) duy nhất trong cụm Kubernetes để chuyển hướng lưu lượng mạng đến các Pod thuộc một Service cụ thể. Điều này cho phép các ứng dụng trong cụm truy cập dễ dàng các dịch vụ mà không cần biết vị trí chính xác của các Pod cụ thể đang thực thi công việc.

Dịch vụ ClusterIP hoạt động bằng cách gán một địa chỉ IP trong phạm vi cụ thể của cụm Kubernetes cho dịch vụ đó. Các Pod của dịch vụ này sau đó được sử dụng để xử lý các yêu cầu từ ứng dụng bên ngoài hoặc từ các Pod khác trong cụm. Khi một yêu cầu được gửi đến địa chỉ IP ClusterIP, cơ chế kube-proxy trong Kubernetes sẽ chuyển hướng yêu cầu đó đến một trong các Pod của dịch vụ đó.

Một số đặc điểm chính của dịch vụ ClusterIP bao gồm:

- Phạm vi hoạt động trong cụm: Địa chỉ IP ClusterIP chỉ có thể truy cập được từ bên trong cụm Kubernetes. Nó không thể được truy cập từ bên ngoài cụm hoặc từ mạng ngoài.
- Mô hình lưu lượng mạng: ClusterIP hướng lưu lượng mạng đến các Pod của dịch vụ theo mô hình Round-robin (chia sẻ công bằng). Điều này đảm bảo rằng các yêu cầu được phân chia đều và không gây tải không cân đối trên các Pod.
- Các cổng và giao thức: ClusterIP hỗ trợ các cổng và giao thức khác nhau, cho phép bạn tạo dịch vụ với nhiều ứng dụng hoặc chức năng khác nhau.

ClusterIP Service tạo địa chỉ IP nội bộ để sử dụng trong K8s cluster. Tốt cho các ứng dụng chỉ dành cho nội bộ hỗ trợ các khối lượng công việc khác trong cluster.



4. Deep dive on ClusterIP Service – Demo

- Thay đổi values.yaml của các microservice (trừ gatewayserver)

*C:\Users\madan\git\microservices-with-spring-sectionwise-code\section16\helm\easybank-services\accounts\values.yaml - Notepad++

```

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window 2
new 5 new 6 new 10 new 11 new 12 References bt new 18 new 16 new 17 IntelliJ Short Cuts bt new 1 new 2 new 9 values.yaml values.yaml

4 deploymentName: accounts-deployment
5 deploymentLabel: accounts
6
7 replicaCount: 1
8
9 image:
10   repository: eazybytes/accounts
11   tag: latest
12
13 containerPort: 8080
14
15 service:
16   type: Cluster
17   port: 8080
18   targetPort: 8080
19
20 config_enabled: true
21 zipkin_enabled: true
22 profile_enabled: true
23 eureka_enabled: true

```

YAML Ain't Markup Language length: 408 lines: 23 Ln: 16 Col: 16 Pos: 284

- Helm build

```
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section16\helm\environments\dev-env>helm dependencies build
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "istio" chart repository
Update Complete. Happy Helming!
Saving 8 charts
Deleting outdated charts
```

```
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section16\helm\environments\prod-env>helm dependencies build
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "istio" chart repository
Update Complete. Happy Helming!
Saving 8 charts
Deleting outdated charts
```

```
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section16\helm\environments\prod-env>cls
```

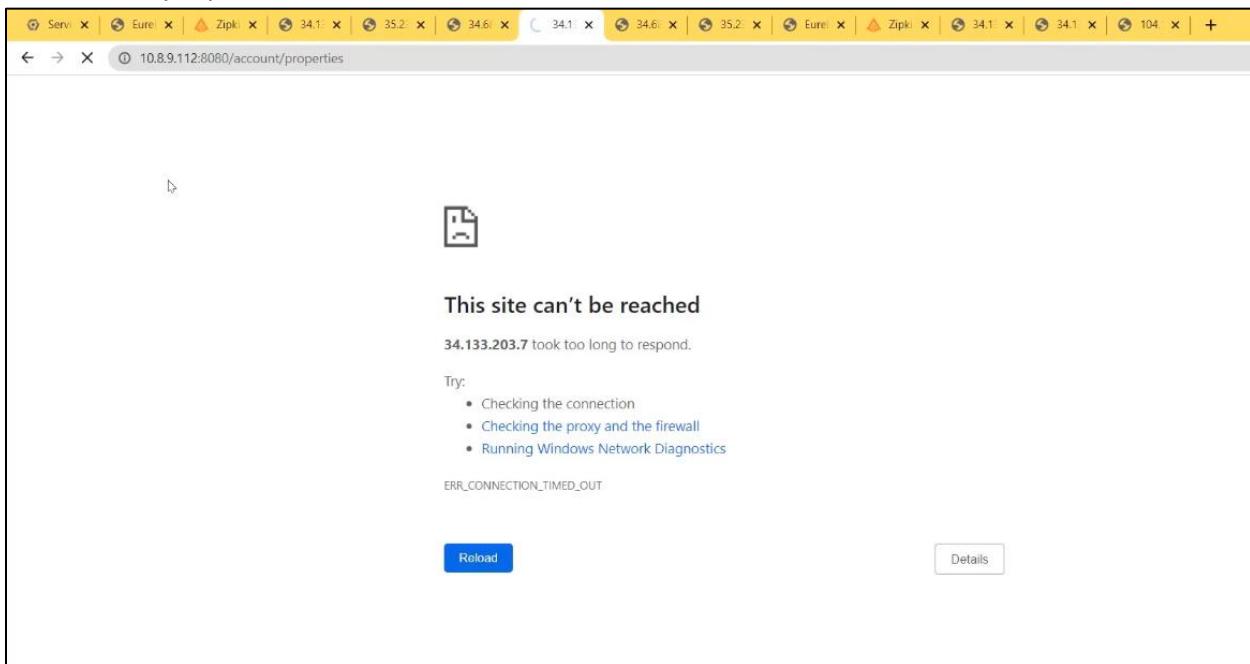
```
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section16\helm\environments\prod-env>cd..
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section16\helm\environments>helm upgrade dev-deployment dev-env
Release "dev-deployment" has been upgraded. Happy Helming!
NAME: dev-deployment
LAST DEPLOYED: Wed May 18 16:46:14 2022
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None
```

```
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section16\helm\environments>
```

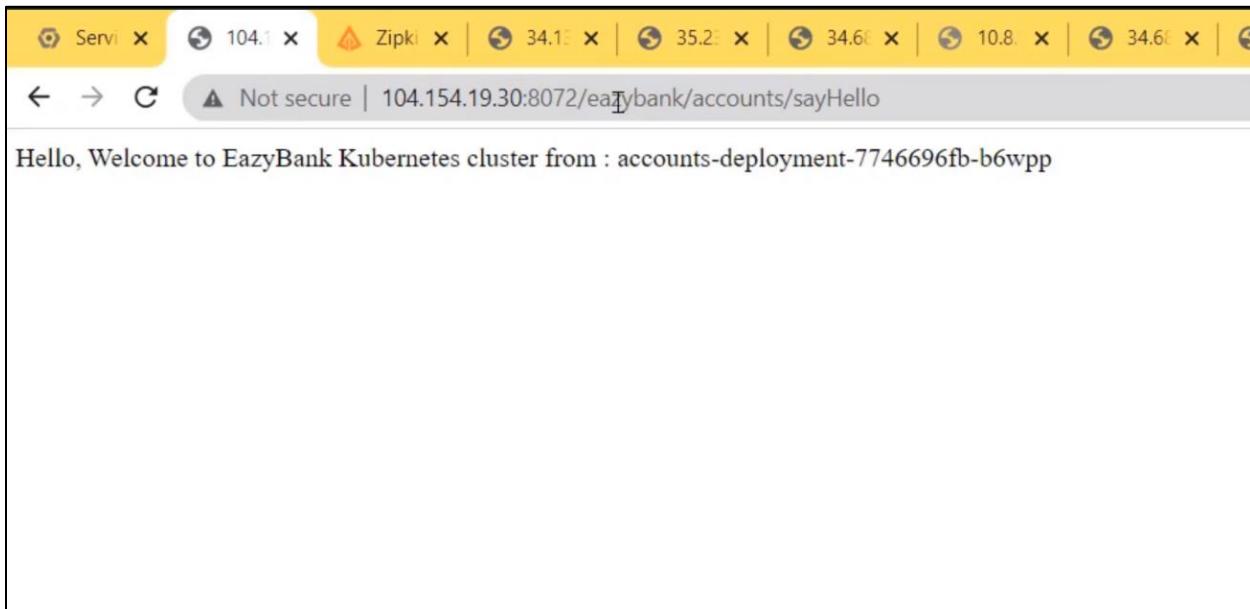
The screenshot shows the Microservices dashboard interface. At the top, there's a navigation bar with a dropdown menu set to 'Microservices', a search bar, and a dropdown for selecting a cluster or namespace. Below the header, there are two tabs: 'SERVICES' (which is selected) and 'INGRESS'. Under the 'SERVICES' tab, there's a brief description about services and ingresses. A filter bar allows for filtering by system objects (checkbox 'Is system object : False'). The main area is a table listing various services:

<input type="checkbox"/>	Name	Status	Type	Endpoints	Pods	Namespace	Clusters
<input type="checkbox"/>	accounts	OK	Cluster IP	10.8.9.112	1/1	default	cluster-1
<input type="checkbox"/>	cards	OK	Cluster IP	10.8.15.180	1/1	default	cluster-1
<input type="checkbox"/>	configserver	OK	Cluster IP	10.8.4.167	1/1	default	cluster-1
<input type="checkbox"/>	eurekaserver	OK	Cluster IP	10.8.7.110	1/1	default	cluster-1
<input type="checkbox"/>	gatewayserver	OK	External load balancer	104.154.19.30:8072	1/1	default	cluster-1
<input type="checkbox"/>	loans	OK	Cluster IP	10.8.4.0	1/1	default	cluster-1
<input type="checkbox"/>	zipkin	OK	Cluster IP	10.8.1.178	1/1	default	cluster-1

- Kiểm tra truy cập vào các microservice:



- Kiểm tra truy cập vào các microservice thông qua gatewayservice:



5. Deep dive on NodePort Service – Theory

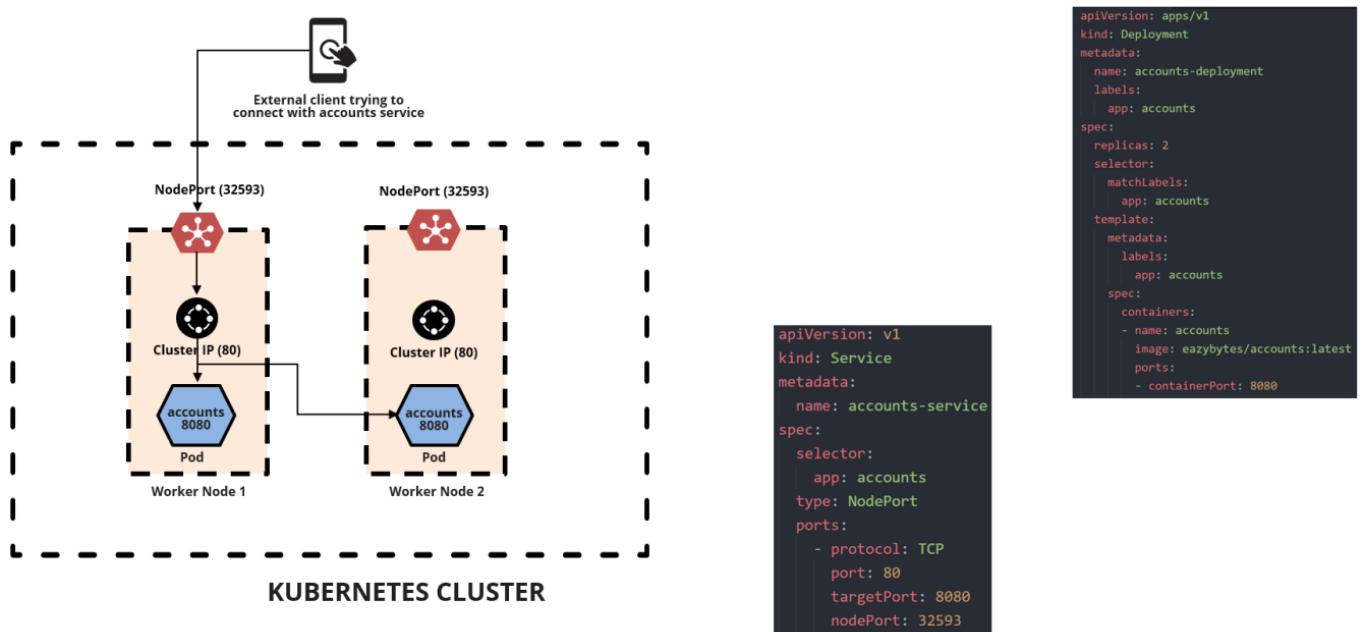
Dịch vụ NodePort là một trong các loại dịch vụ mạng trong Kubernetes, một hệ thống quản lý container phổ biến. Dịch vụ NodePort cho phép truy cập một ứng dụng từ bên ngoài cụm Kubernetes thông qua các cổng được mở trên tất cả các Node (nút) trong cụm. Điều này cho phép ứng dụng của bạn có thể được truy cập từ bất kỳ máy tính nào trong mạng hoặc từ Internet thông qua địa chỉ IP public của các Node.

Cách hoạt động của dịch vụ NodePort như sau:

- Khi tạo một dịch vụ NodePort, Kubernetes sẽ tự động chọn một cổng ngẫu nhiên trong phạm vi 30000-32767 (có thể cấu hình lại) để gắn vào dịch vụ đó. Nếu không có số cổng nào được chỉ định thì Kubernetes sẽ tự động chọn một cổng miễn phí.
- Kube-proxy cục bộ chịu trách nhiệm lắng nghe cổng trên node và chuyển tiếp lưu lượng máy khách trên NodePort tới ClusterIP.
- Mỗi Node trong cụm Kubernetes sẽ chuyển tiếp lưu lượng giao thông nhận được trên cổng được chọn đến cổng đích của các Pod mà dịch vụ đang nhắm tới.
- Các Pod của dịch vụ sẽ nhận lưu lượng giao thông từ Node mà nó được đính kèm, và ứng dụng trong Pod xử lý các yêu cầu từ lưu lượng đó.

Một số đặc điểm chính của dịch vụ NodePort bao gồm:

- Phạm vi hoạt động: Dịch vụ NodePort cho phép truy cập từ bên ngoài cụm Kubernetes. Điều này làm cho ứng dụng của bạn trở nên truy cập được từ Internet hoặc từ các máy tính trong mạng nội bộ.
- Cổng ngẫu nhiên: Kubernetes sẽ tự động chọn cổng ngẫu nhiên cho dịch vụ NodePort, nhưng bạn có thể cấu hình cổng theo ý muốn (nằm trong phạm vi 30000-32767).
- Mô hình cân bằng tải: Nếu một Node có nhiều Pod thuộc cùng một dịch vụ, lưu lượng giao thông sẽ được cân bằng tải giữa các Pod này.



6. Deep dive on NodePort Service – Demo

- Thay đổi values.yaml của các microservice (trừ gatewayserver)

```
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section16\helm\easybank-services\accounts\values.yaml - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window 
new 5 new 6 new 7 new 8 new 9 new 10 new 11 new 12 References br new 13 new 14 new 15 new 16 new 17 new 18 new 19 new 20 new 21 new 22 new 23 new 24 new 25 new 26 new 27 new 28 new 29 new 30 new 31 new 32 new 33 new 34 new 35 new 36 new 37 new 38 new 39 new 40 new 41 new 42 new 43 new 44 new 45 new 46 new 47 new 48 new 49 new 50 new 51 new 52 new 53 new 54 new 55 new 56 new 57 new 58 new 59 new 60 new 61 new 62 new 63 new 64 new 65 new 66 new 67 new 68 new 69 new 70 new 71 new 72 new 73 new 74 new 75 new 76 new 77 new 78 new 79 new 80 new 81 new 82 new 83 new 84 new 85 new 86 new 87 new 88 new 89 new 90 new 91 new 92 new 93 new 94 new 95 new 96 new 97 new 98 new 99 new 100 new 101 new 102 new 103 new 104 new 105 new 106 new 107 new 108 new 109 new 110 new 111 new 112 new 113 new 114 new 115 new 116 new 117 new 118 new 119 new 120 new 121 new 122 new 123 new 124 new 125 new 126 new 127 new 128 new 129 new 130 new 131 new 132 new 133 new 134 new 135 new 136 new 137 new 138 new 139 new 140 new 141 new 142 new 143 new 144 new 145 new 146 new 147 new 148 new 149 new 150 new 151 new 152 new 153 new 154 new 155 new 156 new 157 new 158 new 159 new 160 new 161 new 162 new 163 new 164 new 165 new 166 new 167 new 168 new 169 new 170 new 171 new 172 new 173 new 174 new 175 new 176 new 177 new 178 new 179 new 180 new 181 new 182 new 183 new 184 new 185 new 186 new 187 new 188 new 189 new 190 new 191 new 192 new 193 new 194 new 195 new 196 new 197 new 198 new 199 new 200 new 201 new 202 new 203 new 204 new 205 new 206 new 207 new 208 new 209 new 210 new 211 new 212 new 213 new 214 new 215 new 216 new 217 new 218 new 219 new 220 new 221 new 222 new 223 new 224 new 225 new 226 new 227 new 228 new 229 new 230 new 231 new 232 new 233 new 234 new 235 new 236 new 237 new 238 new 239 new 240 new 241 new 242 new 243 new 244 new 245 new 246 new 247 new 248 new 249 new 250 new 251 new 252 new 253 new 254 new 255 new 256 new 257 new 258 new 259 new 260 new 261 new 262 new 263 new 264 new 265 new 266 new 267 new 268 new 269 new 270 new 271 new 272 new 273 new 274 new 275 new 276 new 277 new 278 new 279 new 280 new 281 new 282 new 283 new 284 new 285 new 286 new 287 new 288 new 289 new 290 new 291 new 292 new 293 new 294 new 295 new 296 new 297 new 298 new 299 new 299 values.yaml
```

4 **deploymentName:** accounts-deployment
5 **deploymentLabel:** accounts
6
7 **replicaCount:** 2
8
9 **image:**
10 **repository:** eazybytes/accounts
11 **tag:** latest
12
13 **containerPort:** 8080
14
15 **service:**
16 **type:** NodePort
17 **port:** 8080
18 **targetPort:** 8080|
19
20 **config_enabled:** true
21 **zipkin_enabled:** true

```
C:\Windows\System32\cmd.exe
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section16\helm\environments>cd dev-env
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section16\helm\environments>dev-env>helm dependencies build
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "istio" chart repository
Update Complete. Happy Helming!
Saving 8 charts
Deleting outdated charts

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section16\helm\environments>cd..
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section16\helm\environments>helm upgrade dev-deployment dev-env
Release "dev-deployment" has been upgraded. Happy Helming!
NAME: dev-deployment
LAST DEPLOYED: Wed May 18 17:27:40 2022
NAMESPACE: default
STATUS: deployed
REVISION: 4
TEST SUITE: None

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section16\helm\environments>
```

Services & Ingress

REFRESH CREATE INGRESS DELETE

Cluster Namespace RESET SAVE

SERVICES INGRESS

Services are sets of Pods with a network endpoint that can be used for discovery and load balancing. Ingresses are collections of rules for routing external HTTP(S) traffic to Services.

Filter Is system object : False Filter services and Ingresses

<input type="checkbox"/>	Name	Status	Type	Endpoints	Pods	Namespace	Clusters
<input type="checkbox"/>	accounts	✓ OK	Node Port	10.8.9.112:8080 TCP	2/2	default	cluster-1
<input type="checkbox"/>	cards	✓ OK	Cluster IP	10.8.15.180	1/1	default	cluster-1
<input type="checkbox"/>	configserver	✓ OK	Cluster IP	10.8.4.167	1/1	default	cluster-1
<input type="checkbox"/>	eurekaserver	✓ OK	Cluster IP	10.8.7.110	1/1	default	cluster-1
<input type="checkbox"/>	gatewayserver	✓ OK	External load balancer	104.154.19.30:8072	1/1	default	cluster-1
<input type="checkbox"/>	loans	✓ OK	Cluster IP	10.8.4.0	1/1	default	cluster-1
<input type="checkbox"/>	zipkin	✓ OK	Cluster IP	10.8.1.178	1/1	default	cluster-1

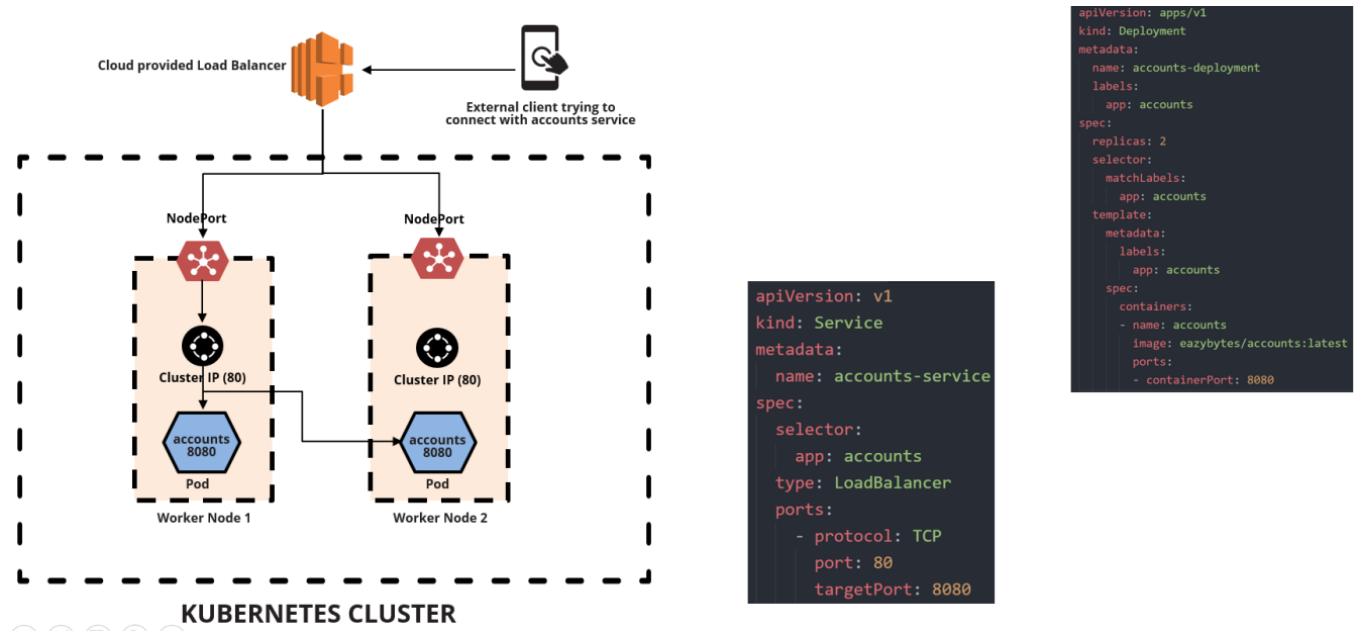
Services & Ingress - Kubernetes | 104.154.19.30:8072/eazybank/accounts | ↗

← → C Not secure | 104.154.19.30:8072/eazybank/accounts/sayHello

Hello, Welcome to EazyBank Kubernetes cluster from : accounts-deployment-7746696fb-b6wpp

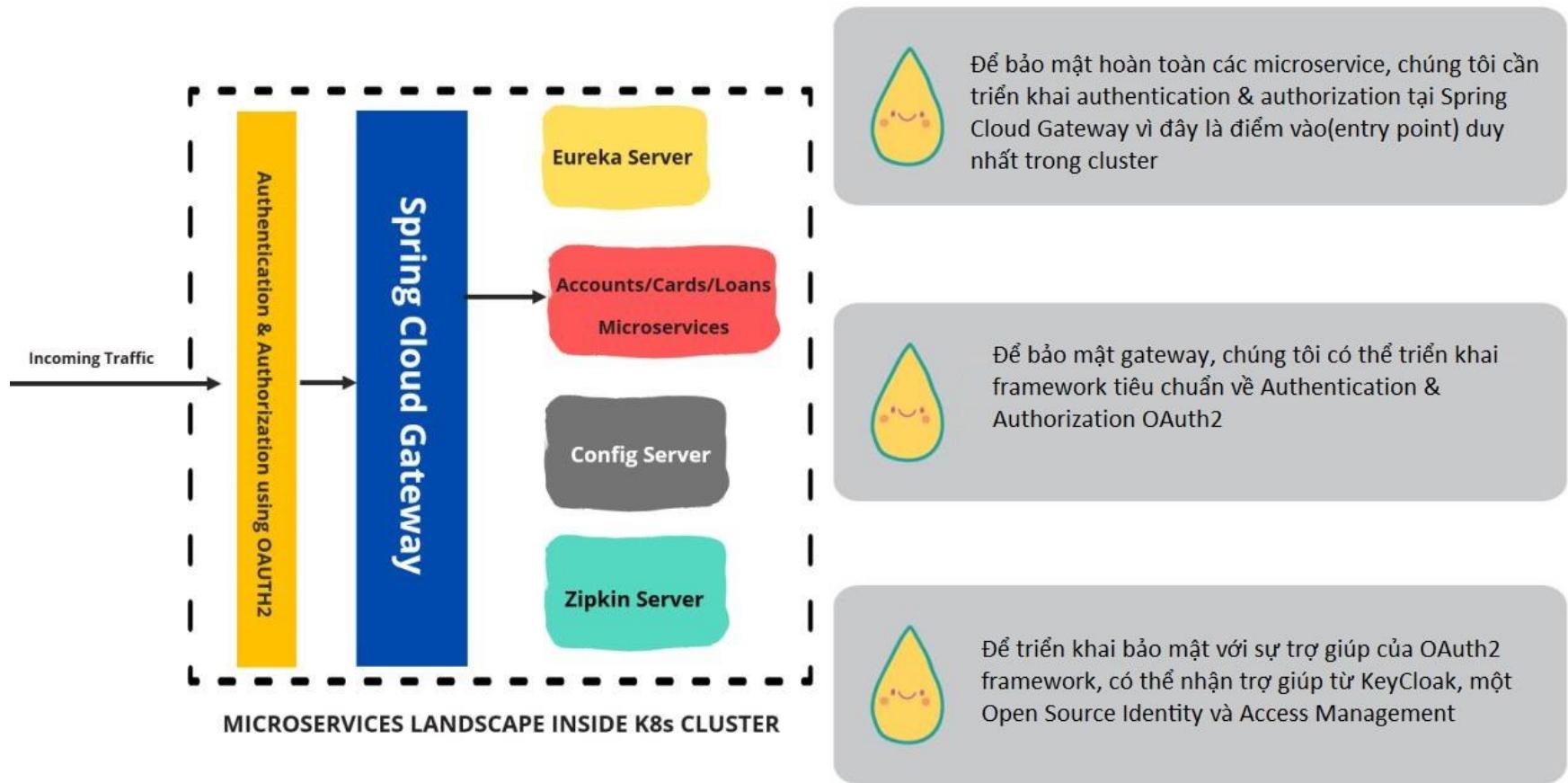
7. Deep dive on LoadBalancer Service- Theory

LoadBalancer Service được xây dựng dựa trên NodePort service bằng cách cung cấp và định cấu hình bộ cân bằng tải bên ngoài từ các public và private cloud provider. Nó hiển thị các dịch vụ đang chạy trong cụm bằng cách chuyển tiếp lưu lượng lớp 4 tới các worker node. Đây là một cách năng động để triển khai một trường hợp liên quan đến các bộ cân bằng tải bên ngoài và các dịch vụ loại NodePort.



Section 15: Securing Microservices using OAuth2 client credentials grant flow

1. Introduction to securing Spring Cloud Gateway with OAuth2



2. Quick intro to OAuth2 framework

Why oauth2?

Tại sao chúng ta nên sử dụng OAUTH2 framework để triển khai bảo mật bên trong các microservice của mình? Tại sao chúng ta không thể sử dụng basic authentication? Để trả lời câu hỏi này, trước tiên hãy cố gắng hiểu về basic authentication và những nhược điểm của nó.

Các trang web ban đầu thường yêu cầu thông tin xác thực thông qua HTML form mà trình duyệt sẽ gửi đến server. Server xác thực thông tin và ghi giá trị session vào cookie; miễn là session vẫn được đánh dấu là hoạt động, người dùng có thể truy cập các tính năng và tài nguyên được bảo vệ.

Hạn chế của basic authentication

- Backend server hoặc business logic được kết hợp chặt chẽ với Authentication/Authorization logic. Không thân thiện với luồng mobile/API REST
- Basic authentication** không phù hợp với trường hợp sử dụng khi người dùng của một sản phẩm hoặc dịch vụ muốn cấp cho khách hàng bên thứ ba quyền truy cập vào thông tin của họ.

Ưu điểm của OAuth2

Hỗ trợ tất cả các loại Ứng dụng

- OAuth2 là authorization framework hỗ trợ nhiều trường hợp sử dụng giải quyết các khả năng khác nhau của thiết bị. Nó hỗ trợ các ứng dụng từ server-to-server apps, browser-based apps, mobile/native apps, IoT devices và consoles/TVs..
- OAuth2 có nhiều luồng cấp quyền khác nhau như cấp Authorization Code grant, Client Credentials Grant Type , v.v. để hỗ trợ tất cả các loại giao tiếp ứng dụng,

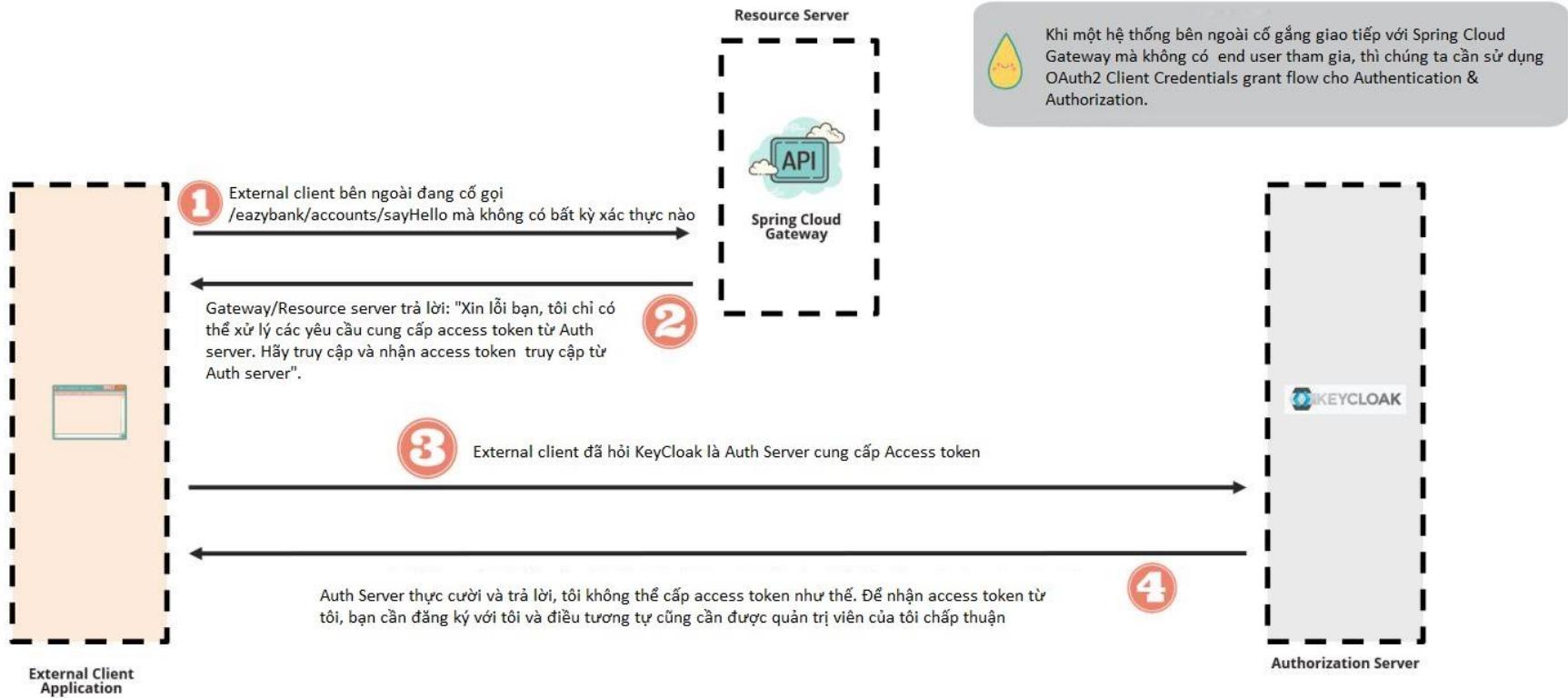
Tách Auth logic

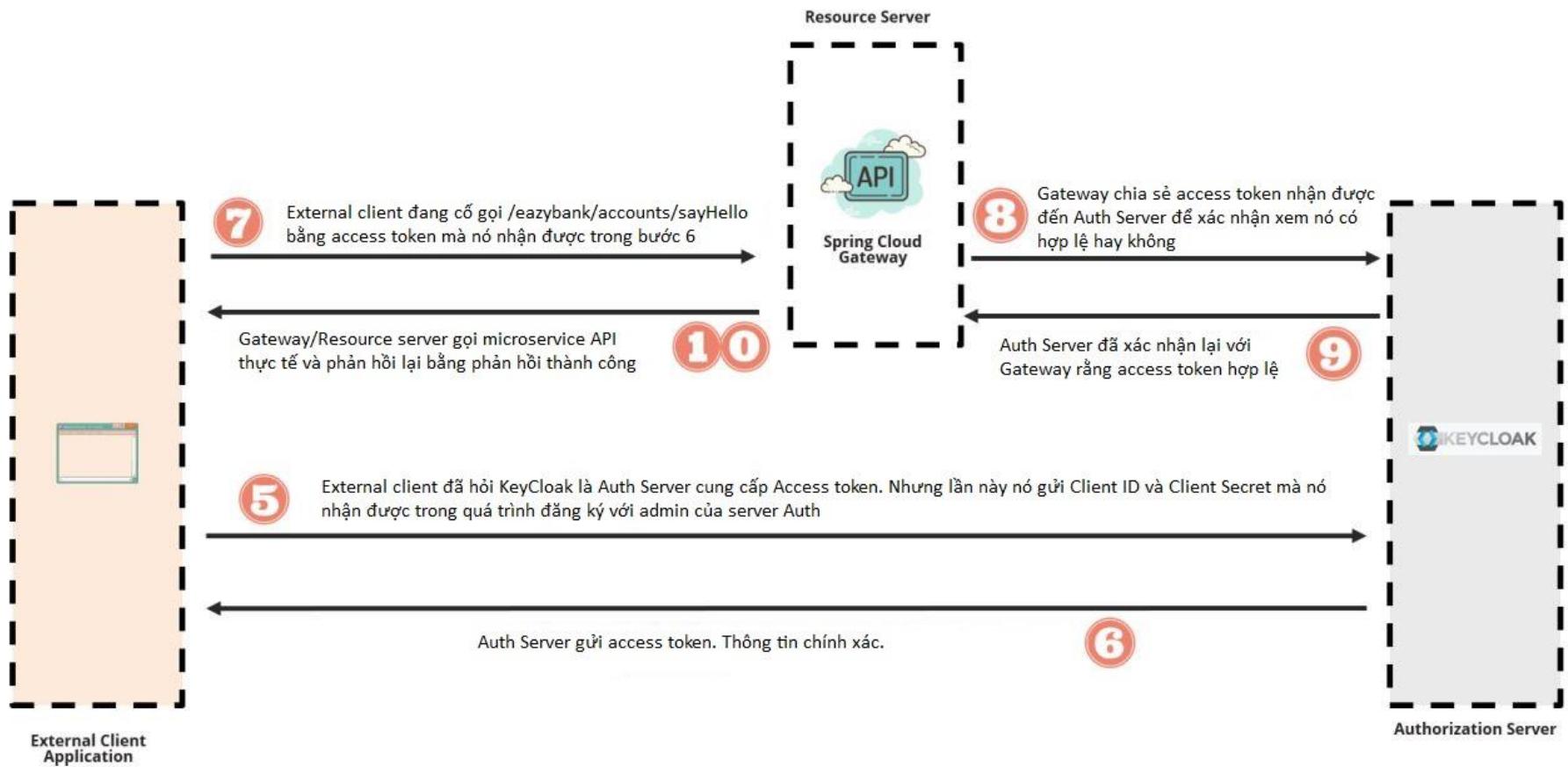
- Bên trong OAuth2, có Authorization Server nhận yêu cầu từ Client về Access Tokens và cấp chúng khi xác thực thành công. Điều này cho phép chúng tôi duy trì tất cả logic bảo mật ở một nơi duy nhất. Bất kể một tổ chức có bao nhiêu ứng dụng, tất cả chúng đều có thể kết nối với Auth server để thực hiện thao tác đăng nhập.
- Tất cả user credentials và client application credentials sẽ được duy trì ở một vị trí duy nhất bên trong Auth Server.

Không cần chia sẻ thông tin xác thực

- Nếu bạn dự định cho phép các ứng dụng và dịch vụ của bên thứ ba truy cập vào tài nguyên của mình thì bạn không cần phải chia sẻ thông tin đăng nhập của mình.
- Theo nhiều cách, bạn có thể coi OAuth2 token là "thẻ ra vào" tại bất kỳ văn phòng/khách sạn nào. Các token này cung cấp quyền truy cập hạn chế cho ai đó mà không chuyển giao toàn quyền kiểm soát dưới dạng master key.

3. Deep dive on OAuth2 Client Credentials grant flow

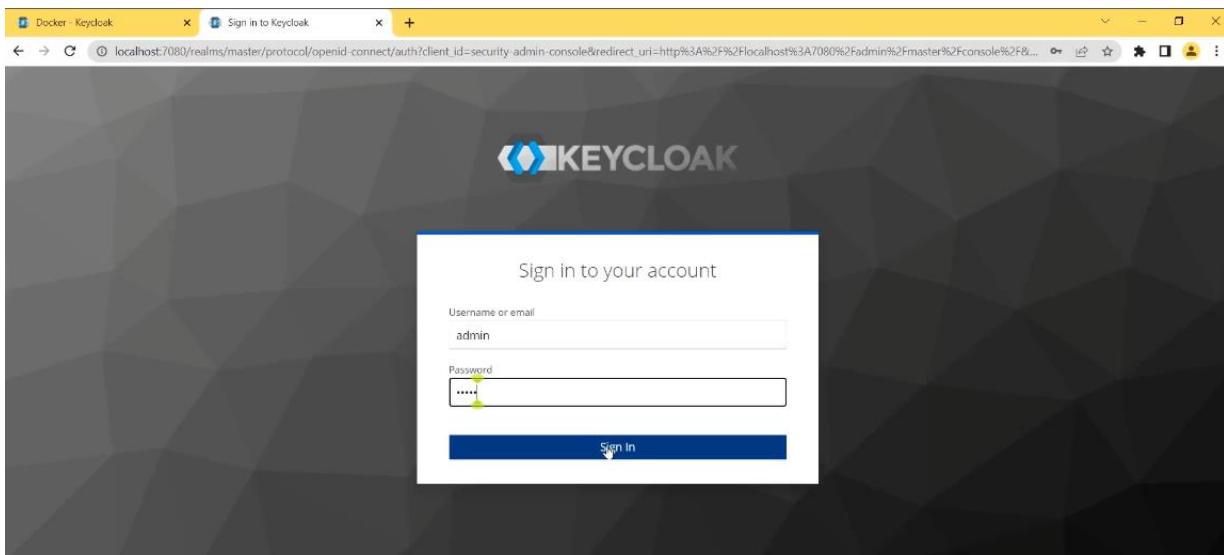
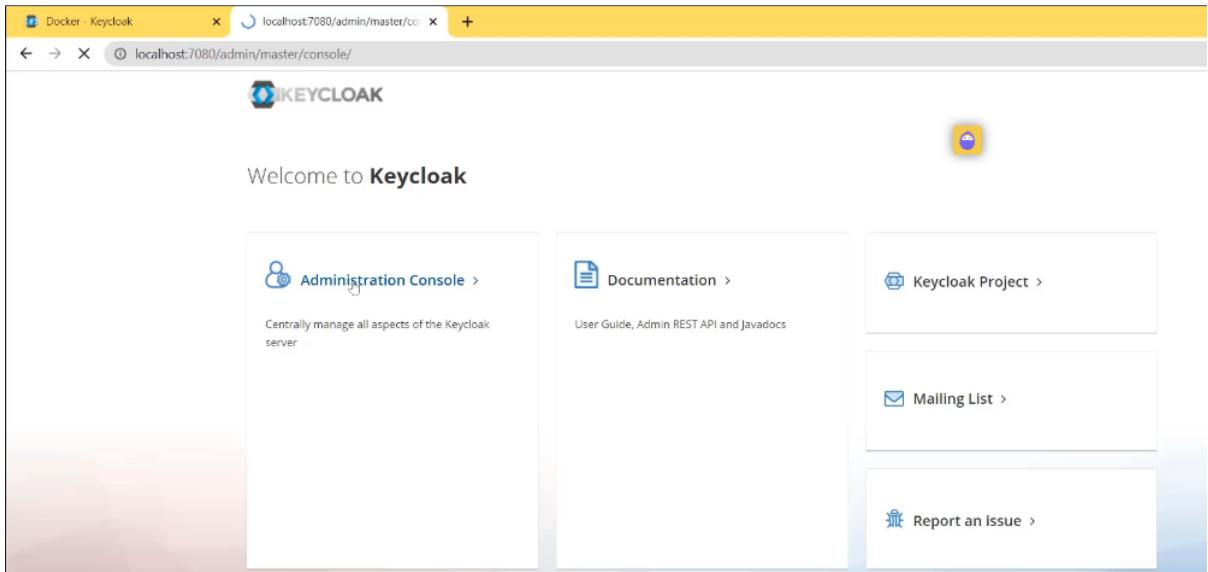




4. Keycloak Auth Server installation and setup using Docker command

<https://www.keycloak.org/getting-started/getting-started-docker>

```
C:\Windows\system32\cmd.exe - docker run -p 7080:8080 -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin quay.io/keycloak/keycloak:18.0.0 start-dev  
C:\Users\madan>docker run -p 7080:8080 -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin quay.io/keycloak/keycloak:18.0.0 start-dev  
Unable to find image 'quay.io/keycloak/keycloak:18.0.0' locally  
18.0.0: Pulling from keycloak/keycloak  
4752687a61a9: Downloading [=====> ] 36.15MB/36.44MB  
0344366a246a: Download complete  
bf8a68204bb: Downloading [=====> ] 45.26MB/172.6MB  
c4db0a36467a: Downloading [=====> ] 49.26MB/87.71MB
```



Docker - Keycloak Keycloak Admin Console

localhost:7080/admin/master/console/#/realms/master

KEYCLOAK

Master

Configure

- Realm Settings
- Clients
- Client Scopes
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

- Groups
- Users
- Sessions
- Events
- Import
- Export

General Login Keys Email Themes Localization Cache Tokens Client Registration Client Policies Security Defenses

* Name: master

Display name: Keycloak

HTML Display name: <div class="kc-logo-text">Keycloak</div>

Frontend URL:

Enabled: ON

User-Managed Access: OFF

Endpoints:

- OpenID Endpoint Configuration
- SAML 2.0 Identity Provider Metadata

Save Cancel

5. Register Client details inside KeyCloak Auth server

Add Client

Client ID: easybank-callcenter

Client Protocol: openid-connect

Root URL: (empty)

Save Cancel

Consent Required: OFF

Login Theme: (empty)

Client Protocol: openid-connect

Access Type: confidential

Standard Flow Enabled: OFF

Implicit Flow Enabled: OFF

Direct Access Grants Enabled: ON

Service Accounts Enabled: ON

OAuth 2.0 Device Authorization Grant Enabled: OFF

OIDC CIBA Grant Enabled: OFF

Authorization Enabled: OFF

Front Channel Logout: OFF

Root URL: (empty)

Base URL: (empty)

Admin URL: (empty)

Clients > easybank-callcenter

Eazybank-callcenter

Settings Credentials Keys Roles Client Scopes Mappers Scope Revocation Sessions Offline Access Clustering Installation

Service Account Roles

Client Authenticator: Client Id and Secret

Secret: LFDIDWxsVkh1FmTVQ2XyC5RBCSAOxEC02

Regenerate Secret

Registration access token: (empty)

Regenerate registration access token

6. Getting Access token from Auth Server using Client details

The screenshot shows a Postman interface with a POST request to `http://localhost:7080/realm/master/protocol/openid-connect/token`. The request body is set to `form-data` and contains three fields: `client_secret`, `scope`, and `grant_type`. The response status is `200 OK`, and the JSON response is:

```
1 {
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOIA1S1dUTiwa2IkTIA6ICJtbEMtS1pVOE8tNG9PnAwW1JpLTBUVhdYU1d2NFV6TjF5RFk2bzRRV1F3In0.eyJleHAiOiE2NTM0MDcwNjcsIiwiY2lkIjoiMSIiLCJpZGVudGl0eSI6ImF1dGhvcml0eSIsInN1YiI6IjIwMjAxMjIwMTQyNjAiLCJpYXQiOjE1NjUxOTUyNjMsImV4cCI6MTUxMjA5ODIyfQ",
  "expires_in": 60,
  "refresh_expires_in": 0,
  "token_type": "Bearer",
  "id_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOIA1S1dUTiwa2IkTIA6ICJtbEMtS1pVOE8tNG9PnAwW1JpLTBUVhdYU1d2NFV6TjF5RFk2bzRRV1F3In0.eyJleHAiOiE2NTM0MDcwNjcsIiwiY2lkIjoiMSIiLCJpZGVudGl0eSI6ImF1dGhvcml0eSIsInN1YiI6IjIwMjAxMjIwMTQyNjAiLCJpYXQiOjE1NjUxOTUyNjMsImV4cCI6MTUxMjA5ODIyfQ",
  "not-before-policy": 0,
  "scope": "openid email profile"
}
```

7. Making code changes inside Spring Cloud Gateway to secure the APIs

Mở file pom.xml của microservice gatewayserver và đảm bảo thêm các dependency bên dưới của Spring Security, OAuth2

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-resource-server</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-jose</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

Để làm cho Spring Cloud Gateway hoạt động như một Resource server và xử lý cả Authentication & Authorization, tạo các lớp SecurityConfig.java, KeycloakRoleConverter.java.

SecurityConfig.java

```
@Configuration
@EnableWebFluxSecurity
public class SecurityConfig {

    @Bean
    public SecurityWebFilterChain
    springSecurityFilterChain(ServerHttpSecurity http) {
        http.authorizeExchange(exchanges ->
    exchanges.pathMatchers("/eazybank/accounts/**").hasRole("ACCOUNTS")
            .pathMatchers("/eazybank/cards/**").authenticated()
            .pathMatchers("/eazybank/loans/**").permitAll()
            .oauth2ResourceServer(oauth2ResourceServerCustomizer ->
                oauth2ResourceServerCustomizer.jwt(jwtCustomizer ->
                    jwtCustomizer.jwtAuthenticationConverter(grantedAuthoritiesExtractor())));
        http.csrf((csrf) -> csrf.disable());
        return http.build();
    }

    Converter<Jwt, Mono<AbstractAuthenticationToken>>
    grantedAuthoritiesExtractor() {
        JwtAuthenticationConverter jwtAuthenticationConverter =
            new JwtAuthenticationConverter();
        jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter
            (new KeycloakRoleConverter());
        return new
    ReactiveJwtAuthenticationConverterAdapter(jwtAuthenticationConverter);
    }

}
}
```

KeycloakRoleConverter.java

KeycloakRoleConverter là một custom converter để trích xuất thông tin về vai trò từ JWT Token, dựa trên nền tảng Keycloak (một Authorization Server). Bạn cần triển khai **KeycloakRoleConverter** để phù hợp với cấu trúc của JWT Token từ Keycloak.

```
public class KeycloakRoleConverter implements Converter<Jwt,
Collection<GrantedAuthority>> {
    @Override
    public Collection<GrantedAuthority> convert(Jwt jwt) {
        Map<String, Object> realmAccess = (Map<String, Object>)
    jwt.getClaims().get("realm_access");
        if (realmAccess == null || realmAccess.isEmpty()) {
            return new ArrayList<>();
        }
        Collection<GrantedAuthority> returnValue = ((List<String>)
    realmAccess.get("roles"))
            .stream().map(roleName -> "ROLE_" + roleName)
            .map(SimpleGrantedAuthority::new)
            .collect(Collectors.toList());
        return returnValue;
    }
}
```

```
}
```

Mở application.properties bên trong gatewayserver microservice và thêm mục nhập sau. Ở đây đang cung cấp URI Keycloak nơi resource server của tôi có thể xác thực access token nhận được.

```
spring.security.oauth2.resource-server.jwt.jwk-set-uri =  
http://localhost:7080/realm/master/protocol/openid-connect/certs
```

8. Demo of Spring Cloud Gateway security inside local system

Vui lòng đảm bảo khởi động tất cả các microservice của bạn bao gồm Zipkin, Keycloak

Truy cập URL <http://localhost:8072/eazybank/accounts/sayHello> thông qua trình duyệt và có thể mong đợi phản hồi 401 vì đường dẫn API tài khoản được bảo mật.

The screenshot shows a Postman collection interface. A specific request is selected for execution:

- Method:** GET
- URL:** <http://localhost:8072/eazybank/cards/cards/properties>
- Headers:** (6) - This tab is currently active.
- Body:** Empty
- Tests:** Empty
- Settings:** Empty

Below the request details, the response section is visible:

- Status:** 401 Unauthorized
- Time:** 715 ms
- Size:** 420 B
- Save Response:** Button

The response body is displayed in "Pretty" format:

```
1
```

- Thêm token khi truy cập (access token)

The screenshot shows the Postman interface with a GET request to `http://localhost:8072/eazybank/accounts/sayHello`. In the Headers tab, there is a `Bearer` header with a long token value. The response body contains the text: `Hello, Welcome to EazyBank Kubernetes cluster from :`

Truy cập URL `http://localhost:8072/eazybank/loans/loans/properties` thông qua trình duyệt và có thể mong đợi phản hồi thành công vì không có bảo mật cho các đường dẫn API.

9. Installation of KeyCloak into K8s cluster using Helm chart

Bạn có thể sử dụng Helm repository của Bitnami để tải Helm chart của Keycloak. Chạy lệnh sau để thêm repository và cài đặt Keycloak:

```
helm repo add bitnami https://charts.bitnami.com/bitnami  
helm install keycloak bitnami/keycloak
```

Khi cài đặt Keycloak bằng Helm chart, dịch vụ Keycloak sẽ được triển khai với loại dịch vụ là ClusterIP. ClusterIP chỉ cho phép truy cập từ bên trong cụm Kubernetes, không thể truy cập từ bên ngoài.

The screenshot shows the Google Cloud Platform Kubernetes Engine Services & Ingress page. The sidebar has options like Clusters, Workloads (selected), Services & Ingress, Applications, Secrets & ConfigMaps, Storage, Object Browser, Migrate to containers, Backup for GKE, Config Management, and Protect. The main area shows a table of services:

Name	Status	Type	Endpoints	Pods	Namespace	Clusters
keycloak	Creating Service Endpoints	External load balancer	None	1/1	default	cluster-1
keycloak-headless	OK	Cluster IP	None	1/1	default	cluster-1
keycloak-postgresql	OK	Cluster IP	10.8.13.115	1/1	default	cluster-1
keycloak-postgresql-hl	OK	Cluster IP	None	1/1	default	cluster-1

Google Cloud Platform Microservices

Kubernetes Engine Workloads

Clusters Workloads Services & Ingress Applications Secrets & ConfigMaps Storage Object Browser Migrate to containers Backup for GKE

Workloads Cluster Namespace RESET SAVE

Workloads are deployable units of computing that can be created and managed in a cluster.

OVERVIEW COST OPTIMIZATION

Filter is system object : False Filter workloads

Name	Status	Type	Pods	Namespace	Cluster
keycloak	⚠️ Pods have warnings	Stateful Set	1/1	default	cluster-1
keycloak-postgresql	OK	Stateful Set	1/1	default	cluster-1

Google Cloud Platform Microservices

Kubernetes Engine Configuration

Clusters Workloads Services & Ingress Applications Secrets & ConfigMaps Storage Object Browser Migrate to containers Backup for GKE Config Management Protect NEW

Configuration Cluster Namespace RESET SAVE

Secrets are sensitive pieces of information, such as passwords, keys, and tokens. ConfigMaps are designed to store information that is not sensitive, such as environment variables, command-line arguments, and configuration files.

Secrets respect access control and are not visible to users without read permissions

Filter is system object : False Filter secrets and config maps

Name	Type	Namespace	Cluster
default-token-cnxn	Secret	kube-node-lease	cluster-1
keycloak	Secret	default	cluster-1
keycloak-env-vars	Config Map	default	cluster-1
keycloak-postgresql	Secret	default	cluster-1
keycloak-token-drn	Secret	default	cluster-1
kube-root-ca.crt	Config Map	kube-node-lease	cluster-1
kube-root-ca.crt	Config Map	default	cluster-1
kube-root-ca.crt	Config Map	kube-public	cluster-1
sh.helm.release.v1.keycloak.v1	Secret	default	cluster-1

Google Cloud Platform Microservices

Kubernetes Engine Secret details

Clusters Workloads Services & Ingress Applications Secrets & ConfigMaps Storage Object Browser Migrate to containers Backup for GKE Config Management Protect NEW

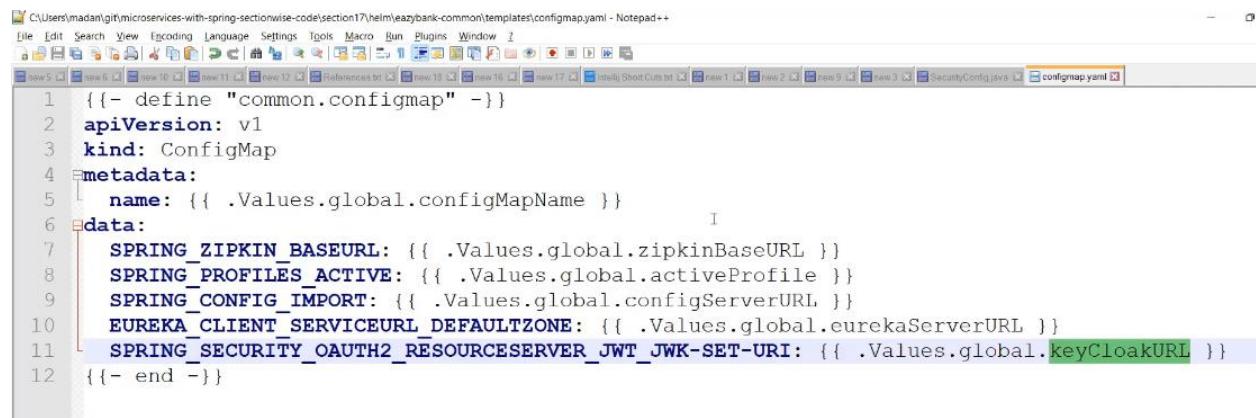
Secret details Cluster default Namespace default Created May 25, 2022, 6:45:36 PM Labels app.kubernetes.io/component: keycloak app.kubernetes.io/instance: keycloak app.kubernetes.io/managed-by: Helm app.kubernetes.io/name: keycloak helm.sh/chart: keycloak:9.0.4 Annotations meta.helm.sh/release-name: keycloak meta.helm.sh/release-namespace: default

Data Sensitive data fields are not displayed in the console.

admin-password	*****
management-password	*****

10. Updating Helm charts of microservices

```
{-- define "common.configmap" --}
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Values.global.configMapName }}
data:
  MANAGEMENT_ZIPKIN_TRACING_ENDPOINT: {{ .Values.global.zipkinBaseURL }}
  SPRING_PROFILES_ACTIVE: {{ .Values.global.activeProfile }}
  SPRING_CONFIG_IMPORT: {{ .Values.global.configServerURL }}
  EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: {{ .Values.global.eurekaServerURL }}
  SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_JWK-SET-URI: {{ .Values.global.keycloakURL }}
{{- end -}}
```



helm/environments/dev-env/values.yaml (làm tương tự với pro-env)

```
global:
  configMapName: eazybankdev-configmap
  zipkinBaseUrl: http://zipkin:9411/api/v2/spans
  activeProfile: dev
  configServerURL: configserver:http://configserver:8071/
  eurekaServerURL: http://eurekaserver:8070/eureka/
  keycloakURL: http://keycloak:80/realms/master/protocol/openid-connect/certs
```

helm/eazybank-common/templates/deployment.yaml

```
{-- if .Values.keycloak_enabled --}
- name: SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_JWK-SET-URI
  valueFrom:
    configMapKeyRef:
      name: {{ .Values.global.configMapName }}
      key: SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_JWK-SET-URI
{{- end -}}
```

helm/eazybank-services/accounts/values.yaml

```
config_enabled: true
zipkin_enabled: true
profile_enabled: true
eureka_enabled: true
keycloak_enabled: false
apname_enabled: true
```

- Helm build

```
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section17\helm\eazybank-services\gatewayserver>helm dependencies build
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "bitnami" chart repository
Update Complete. Happy Helming!
Saving 1 charts
Deleting outdated charts

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section17\helm\eazybank-services\gatewayserver>cd..

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section17\helm\eazybank-services\loans>cd loans

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section17\helm\eazybank-services\loans>helm dependencies build
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "bitnami" chart repository
Update Complete. Happy Helming!
Saving 1 charts
Deleting outdated charts

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section17\helm\eazybank-services\loans>cd..

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section17\helm\eazybank-services>cd zipkin

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section17\helm\eazybank-services\zipkin>helm dependencies build
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "bitnami" chart repository
Update Complete. Happy Helming!
Saving 1 charts
Deleting outdated charts

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section17\helm\eazybank-services\zipkin>cd..

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section17\helm\eazybank-services>cd..
```

11. Deploying all microservices into K8s and validating security changes

```
C:\Windows\System32\cmd.exe

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section17\helm\environments>cd..

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section17\helm\environments>helm install dev-deployment dev-env
NAME: dev-deployment
LAST DEPLOYED: Wed May 25 20:21:15 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section17\helm\environments>
```

Google Cloud Platform Microservices Search Products, resources, docs (/)

Kubernetes Engine Configuration

Clusters Workloads Services & Ingress Applications Secrets & ConfigMaps Storage Object Browser Migrate to containers Backup for GKE Config Management Protect NEW

REFRESH DELETE RESET SAVE

Secrets are sensitive pieces of information, such as passwords, keys, and tokens. ConfigMaps are designed to store information that is not sensitive, such as environment variables, command-line arguments, and configuration files.

Secrets respect access control and are not visible to users without read permissions

Filter Is system object : False Filter secrets and config maps

Name ↑	Type	Namespace	Cluster
default-token-cnxn	Secret	kube-node-lease	cluster-1
eazybankdev-configmap	Config Map	default	cluster-1
keycloak	Secret	default	cluster-1
keycloak-env-vars	Config Map	default	cluster-1
keycloak-postgresql	Secret	default	cluster-1
keycloak-token-rcdmn	Secret	default	cluster-1
kube-root-ca.crt	Config Map	kube-node-lease	cluster-1
kube-root-ca.crt	Config Map	default	cluster-1

Google Cloud Platform Microservices Search Products, resources, docs (/)

Kubernetes Engine Config map details

Clusters Workloads Services & Ingress Applications Secrets & ConfigMaps Storage Object Browser Migrate to containers Backup for GKE Config Management Protect NEW

REFRESH EDIT DELETE KUBECTL

eazybankdev-configmap

DETAILS YAML

Cluster: cluster-1 Namespace: default Created: May 25, 2022, 8:21:21 PM Labels: app.kubernetes.io/managed-by: Helm Annotations: meta.helm.sh/release-name: dev-deployment meta.helm.sh/release-namespace: default

Data

EUREKA_CLIENT_SERVICEURL_DEFAULTZONE	http://eurekaserver:8070/eureka/
SPRING_CONFIG_IMPORT	configserver:http://configserver:8071/
SPRING_PROFILES_ACTIVE	dev
SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_JWK_SET-URI	http://keycloak:80/realm/master/protocol/openid-connect/certs
SPRING_ZIPKIN_BASEURL	http://zipkin:9411/

Google Cloud Platform Microservices Search Products, resources, docs (/)

Kubernetes Engine Services & Ingress

Clusters Workloads Services & Ingress Applications Secrets & ConfigMaps Storage Object Browser Migrate to containers Backup for GKE Config Management Protect NEW

REFRESH CREATE INGRESS DELETE

SERVICES INGRESS

Services are sets of Pods with a network endpoint that can be used for discovery and load balancing. Ingresses are collections of rules for routing external HTTP(S) traffic to Services.

Filter Is system object : False Filter services and ingresses

Name ↑	Status	Type	Endpoints	Pods	Namespace	Clusters
accounts	OK	Cluster IP	10.8.7.100	2/2	default	cluster-1
cards	OK	Cluster IP	10.8.15.173	1/1	default	cluster-1
configserver	OK	Cluster IP	10.8.13.161	1/1	default	cluster-1
eurekaserver	OK	Cluster IP	10.8.6.108	1/1	default	cluster-1
gatewayserver	OK	External load balancer	34.70.121.12:80	1/1	default	cluster-1
keycloak	OK	External load balancer	35.238.37.119:80	1/1	default	cluster-1
keycloak-headless	OK	Cluster IP	None	1/1	default	cluster-1
keycloak-postgresql	OK	Cluster IP	10.8.13.115	1/1	default	cluster-1
keycloak-postgresql-hl	OK	Cluster IP	None	1/1	default	cluster-1
loans	OK	Cluster IP	10.8.6.37	1/1	default	cluster-1

```

1 // 2022052502435
2 // http://34.70.121.12:8072/eazybank/loans/loans/properties
3
4 {
5   "msg": "Welcome to the EazyBank Loans Dev application",
6   "buildVersion": "2",
7   "mailDetails": {
8     "hostName": "dev-loans@eazybytes.com",
9     "port": "9021",
10    "from": "dev-loans@eazybytes.com",
11    "subject": "Your Loan Details from Eazy Bank Dev Environment"
12  },
13  "activeBranches": [
14    "Chennai",
15    "Berlin",
16    "Indianapolis"
17  ]
18 }

```

http://35.238.37.119:80/realms/master/protocol/openid-connect/token

POST http://35.238.37.119:80/realms/master/protocol/openid-connect/token

Body (x-www-form-urlencoded)

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> client_id	eazybank-callcenter	
<input checked="" type="checkbox"/> client_secret	LFIDWxsVkf1FmTVQ2XyC5RBCSAOxEC9Z	
<input checked="" type="checkbox"/> scope	openid	

Status: 200 OK Time: 121 ms Size: 2.86 KB Save Response

```

1 cyLR6s2B-qBD-fq81EcrtX2m1D-NVR3_a8AVf3ucQO2aXp8M2yCM0z1gzH8ppA1aLo5ceXoBzVOZy78DrANlyj2BnT2Fc2QtFhPfAM1qrP0b4X18Ma8M_ZXGNnsTswJuXeX2FnV8gkJFPInq_q61g,
2
3
4
5

```

http://34.70.121.12:8072/eazybank/accounts/sayHello

GET http://34.70.121.12:8072/eazybank/accounts/sayHello

Headers (7)

Key	Value	Description
<input checked="" type="checkbox"/> User-Agent	PostmanRuntime/7.29.0	
<input checked="" type="checkbox"/> Accept	*/*	
<input checked="" type="checkbox"/> Accept-Encoding	gzip, deflate, br	
<input checked="" type="checkbox"/> Connection	keep-alive	
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJSUzI1NiIsInR5cCigOiAiSlDUlwia2lkIiA...	

Status: 200 OK Time: 1351 ms Size: 537 B Save Response

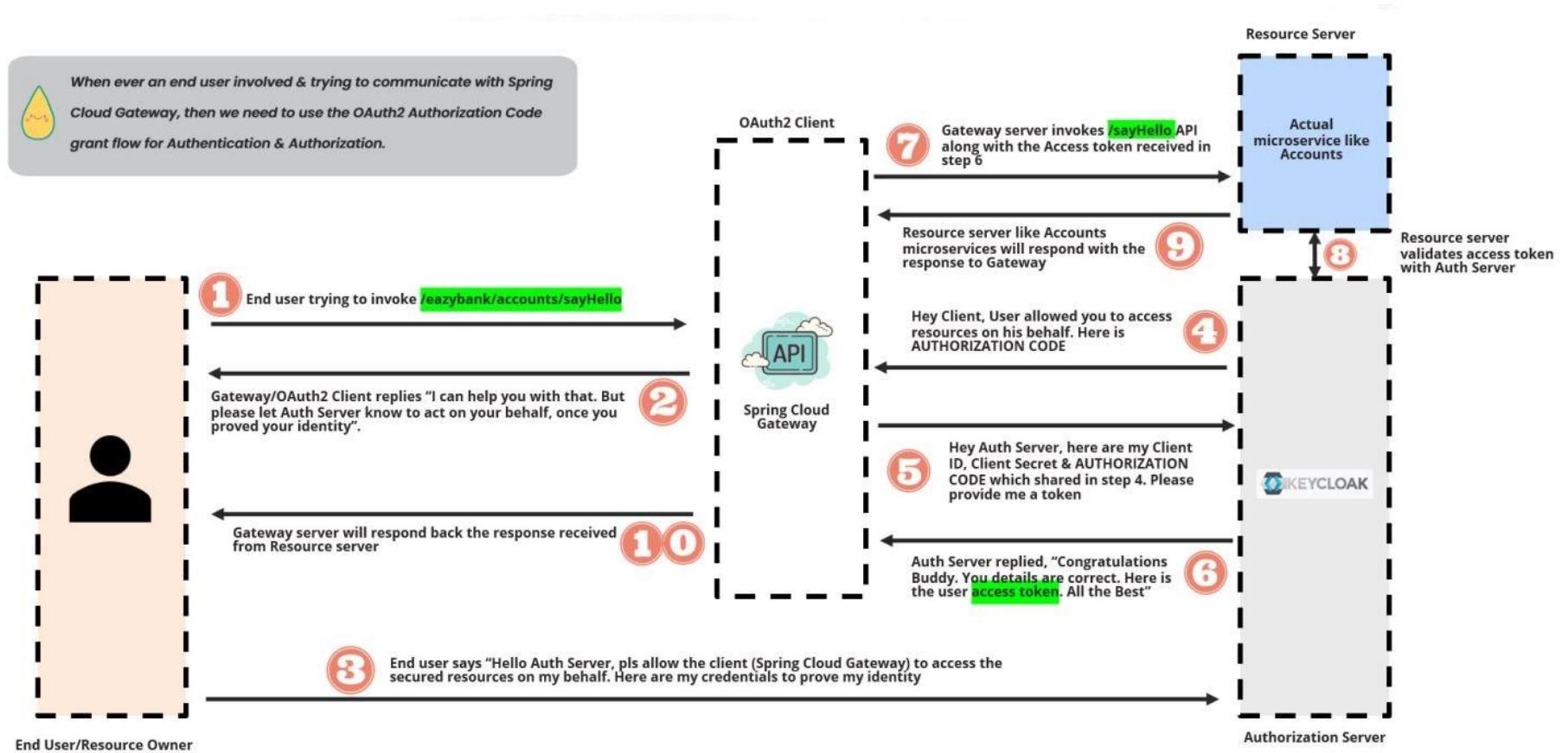
```

1 Hello, Welcome to EazyBank Kubernetes cluster from : Optional[accounts-deployment-7746696fb-92pp5]
2
3

```

Section 16: Securing Microservices using OAuth2 Authorization code grant flow

1. Introduction to OAuth2 Authorization code grant flow



2. Making code changes inside Accounts microservice to secure the APIs

Mở tệp pom.xml của microservice accounts và đảm bảo thêm các dependency bắt buộc của Spring Security, OAuth2.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-resource-server</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-jose</artifactId>
</dependency>
<dependency>
```

Để làm cho accounts microservice của hoạt động như một resource server và xử lý cả Authentication và Authorization, vui lòng tạo các lớp SecurityConfig.java, KeycloakRoleConverter.java.

SecurityConfig.java

\accounts\src\main\java\com\easybytes\accounts\config\SecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain web(HttpSecurity http) throws Exception {
        JwtAuthenticationConverter jwtAuthenticationConverter = new
        JwtAuthenticationConverter();
        jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(new
        KeycloakRoleConverter());

        http.authorizeHttpRequests(authorize ->
        authorize.requestMatchers("/sayHello").hasRole("ACCOUNTS")
            .anyRequest().authenticated()
            .oauth2ResourceServer(oauth2ResourceServerCustomizer ->
                oauth2ResourceServerCustomizer.jwt(jwtCustomizer ->
                    jwtCustomizer.jwtAuthenticationConverter(jwtAuthenticationConverter)));
        http.csrf((csrf) -> csrf.disable());
        return http.build();
    }
}
```

KeycloakRoleConverter.java.

\accounts\src\main\java\com\easybytes\accounts\config\KeycloakRoleConverter.java

```
public class KeycloakRoleConverter implements Converter<Jwt,
Collection<GrantedAuthority>> {

    @Override
    public Collection<GrantedAuthority> convert(Jwt jwt) {
        Map<String, Object> realmAccess = (Map<String, Object>)
jwt.getClaims().get("realm_access");
        if (realmAccess == null || realmAccess.isEmpty()) {
            return new ArrayList<>();
        }
        Collection<GrantedAuthority> returnValue = ((List<String>)
realmAccess.get("roles"))
            .stream().map(roleName -> "ROLE_" + roleName)
            .map(SimpleGrantedAuthority::new)
            .collect(Collectors.toList());
        return returnValue;
    }
}
```

Mở application.properties bên trong accounts microservice và thêm mục nhập sau. Ở đây cung cấp URI Keycloak nơi resource server của tôi có thể xác thực các access token nhận được.

\accounts\src\main\resources\application.properties

```
spring.security.oauth2.resource-server.jwt.jwk-set-uri =
http://localhost:7080/realm/master/protocol/openid-connect/certs
```

3. Register Client details inside KeyCloak Auth server for Spring Cloud Gateway

The screenshot shows the Keycloak Admin Console interface. The top navigation bar indicates the URL: `localhost:7080/admin/master/console/#/create/client/master`. The left sidebar menu is visible, showing options like 'Realm Settings', 'Clients', 'Client Scopes', 'Roles', 'Identity Providers', 'User Federation', and 'Authentication'. The main content area is titled 'Add Client'.

Add Client Form:

- Import:** A 'Select file' button with a file icon.
- Client ID ***: eazybank-gateway-ui
- Client Protocol**: openid-connect
- Root URL**: (empty field)
- Save** and **Cancel** buttons.

Client Configuration Options (Enabled/Disabled):

- Consent Required: OFF
- Login Theme: (dropdown menu)
- Client Protocol: openid-connect
- Access Type: confidential
- Standard Flow Enabled: ON
- Implicit Flow Enabled: OFF
- Direct Access Grants Enabled: ON
- Service Accounts Enabled: OFF
- OAuth 2.0 Device Authorization Grant Enabled: OFF
- OIDC CIBA Grant Enabled: OFF
- Authorization Enabled: OFF

Clients List:

Client ID	Enabled	Base URL	Actions
account	True	http://localhost:7080/realm/master/account/	Edit Export Delete
account-console	True	http://localhost:7080/realm/master/account/	Edit Export Delete
admin-cli	True	Not defined	Edit Export Delete
broker	True	Not defined	Edit Export Delete
eazybank-callcenter	True	Not defined	Edit Export Delete
eazybank-gateway-ui	True	Not defined	Edit Export Delete
master-realm	True	Not defined	Edit Export Delete
security-admin-console	True	http://localhost:7080/admin/master/console/	Edit Export Delete

4. Making code changes inside Spring Cloud Gateway

Xóa tất cả các thay đổi liên quan đến resource server đã thực hiện trong Spring Cloud Gateway

Mở tệp pom.xml của microservice gatewayserver và đảm bảo thêm các dependency bắt buộc của Spring Security, OAuth2

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

Để làm cho microservice gatewayserver hoạt động như một client OAuth2, vui lòng tạo lớp SecurityConfig.java. Nó sẽ giống như dưới đây,

\gatewayserver\src\main\java\com\eaaztbytes\gatewayserver\config\SecurityConfig.java

```
@Configuration
@EnableWebFluxSecurity
public class SecurityConfig {
    @Bean
    public SecurityWebFilterChain
    springSecurityFilterChain(ServerHttpSecurity http) {
        http.authorizeExchange(exchanges ->
            exchanges.pathMatchers("/eazybank/accounts/**").authenticated()
                .pathMatchers("/eazybank/cards/**").authenticated()
                .pathMatchers("/eazybank/loans/**").permitAll())
            .oauth2Login(Customizer.withDefaults());
        http.csrf((csrf) -> csrf.disable());
        return http.build();
    }
}
```

Mở application.properties bên trong các microservice gatewayserver và thêm các thuộc tính sau:

```
spring.security.oauth2.client.provider.keycloak.token-
uri=http://localhost:7080/realm/master/protocol/openid-connect/token
spring.security.oauth2.client.provider.keycloak.authorization-
uri=http://localhost:7080/realm/master/protocol/openid-connect/auth
spring.security.oauth2.client.provider.keycloak.userInfo-
uri=http://localhost:7080/realm/master/protocol/openid-connect/userinfo
spring.security.oauth2.client.provider.keycloak.user-name-
attribute=preferred_username
spring.security.oauth2.client.registration.eazybank-gateway.provider=keycloak
spring.security.oauth2.client.registration.eazybank-gateway.client-
id=eazybank-gateway-ui
spring.security.oauth2.client.registration.eazybank-gateway.client-
secret=zfoalJ1P4e4uTkkIJYQtm9MviTYJ6sqn
spring.security.oauth2.client.registration.eazybank-gateway.authorization-
grant-type=authorization_code
spring.security.oauth2.client.registration.eazybank-gateway.redirect-
uri={baseUrl}/login/oauth2/code/keycloak
```

Thay đổi @Bean RouteLocator:

```
@SpringBootApplication
public class GatewayserverApplication {

    public static void main(String[] args) {
        SpringApplication.run(GatewayserverApplication.class, args);
    }

    @Autowired
    private TokenRelayGatewayFilterFactory filterFactory;

    @Bean
    public RouteLocator myRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
            .route(p -> p
                .path("/eazybank/accounts/**")
                .filters(f -> f.filters(filterFactory.apply()))

            .rewritePath("/eazybank/accounts/(?<segment>.*)", "/${segment}")
                .removeRequestHeader("Cookie")
                .uri("lb://ACCOUNTS"))
            .route(p -> p
                .path("/eazybank/loans/**")
                .filters(f -> f.filters(filterFactory.apply()))

            .rewritePath("/eazybank/loans/(?<segment>.*)", "/${segment}")
                .removeRequestHeader("Cookie")
                .uri("lb://LOANS"))
            .route(p -> p
                .path("/eazybank/cards/**")
                .filters(f -> f.filters(filterFactory.apply()))

            .rewritePath("/eazybank/cards/(?<segment>.*)", "/${segment}")
                .removeRequestHeader("Cookie")
                .uri("lb://CARDS")).build();
    }
}
```

5. Demo of OAuth2 Authorization code grant flow inside local system

Users > adam

Adam

Details Attributes **Credentials** Role Mappings Groups Consents Sessions

ID	fd36bc6d-a774-4cf9-ada1-5881f2ced755
Created At	5/29/22 4:26:05 PM
Username	adam
Email	
First Name	
Last Name	
User Enabled	ON
Email Verified	OFF
Required User Actions	Select an action...
Impersonate user	Impersonate

Users > adam

Adam

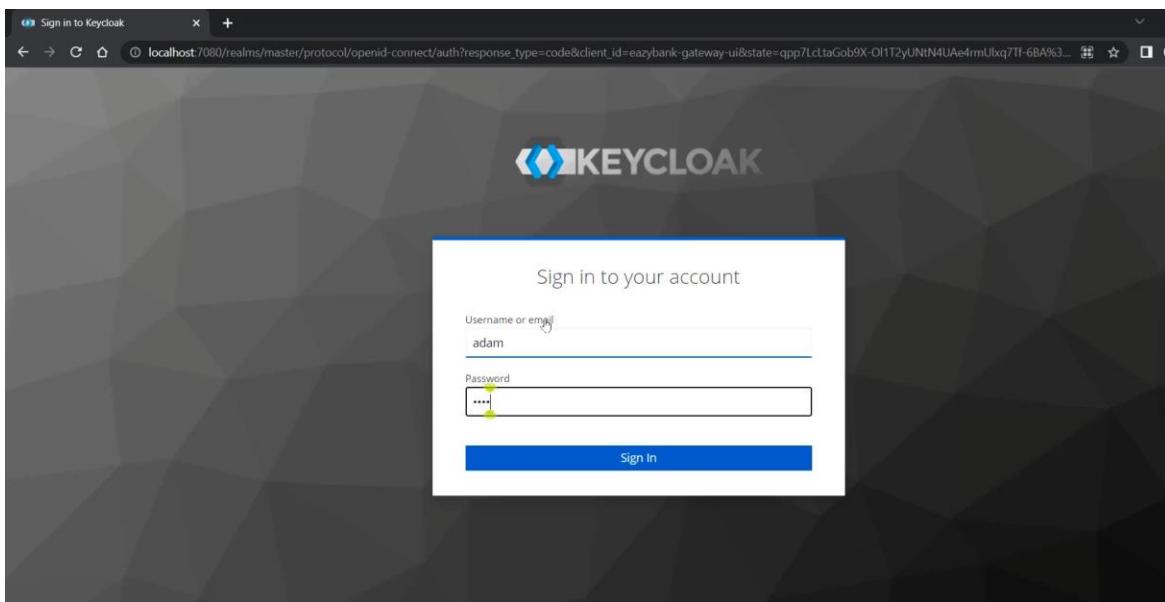
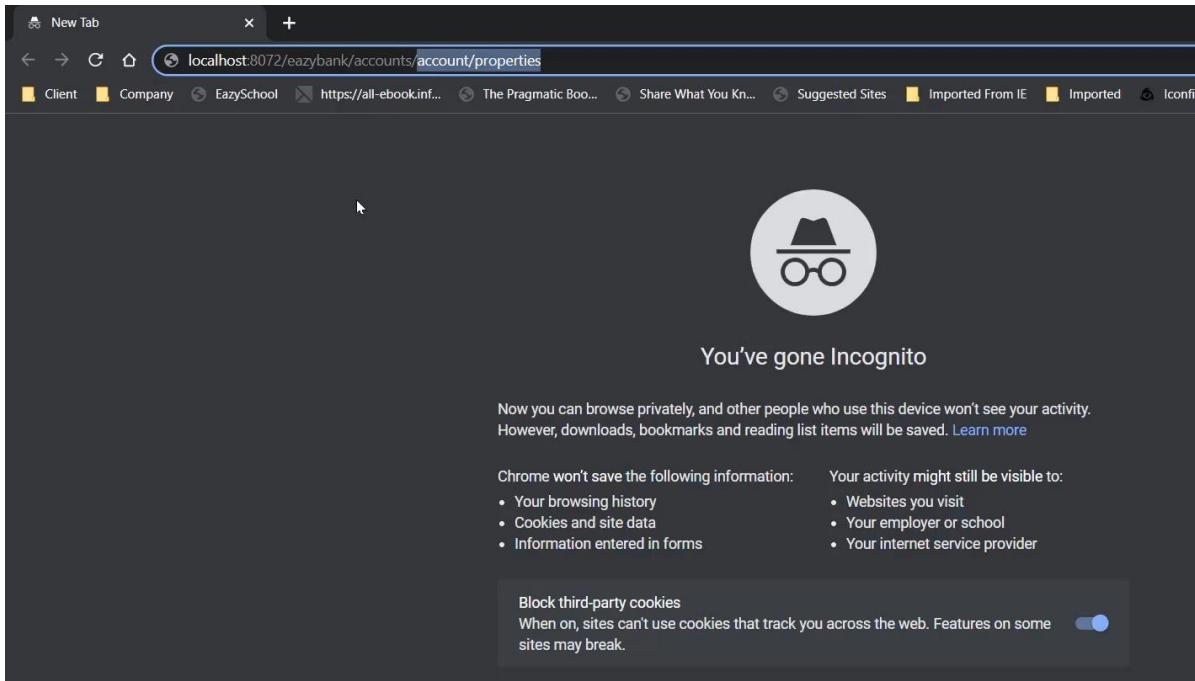
Details Attributes **Credentials** Role Mappings Groups Consents Sessions

Manage Credentials

Position	Type	User Label	Data
----------	------	------------	------

Set Password

Password	*****	
Password Confirmation	*****	
Temporary	OFF	



- Tạo thêm user mới

The screenshot shows the 'Role Mappings' tab selected for the user 'john'. The interface is divided into four main sections: 'Realm Roles', 'Available Roles', 'Assigned Roles', and 'Effective Roles'.

- Realm Roles:** A list containing 'admin', 'create-realm', 'offline_access', and 'uma_authorization'.- Available Roles:** A list containing 'admin', 'create-realm', 'offline_access', and 'uma_authorization'. Below this list is a button labeled 'Add selected >'.
- Assigned Roles:** A list containing 'ACCOUNTS' and 'default-roles-master'. Below this list is a button labeled '« Remove selected'.
- Effective Roles:** A list containing 'ACCOUNTS', 'default-roles-master', 'offline_access', and 'uma_authorization'.

The screenshot shows a browser window with the URL `localhost:8072/eazybank/accounts/sayHello`. The page content displays the message: "Hello, Welcome to EazyBank Kubernetes cluster from :".

6. Updating Helm charts of microservices

/helm/environments/prod-env/values.yaml (tương tự dev-env)

```
global:
  configMapName: eazybankprod-configmap
  zipkinBaseURL: http://zipkin:9411/api/v2/spans
  activeProfile: prod
  configServerURL: configserver:http://configserver:8071/
  eurekaServerURL: http://eurekaserver:8070/eureka/
  keycloakURL: http://keycloak:80/realms/master/protocol/openid-connect/certs
  tokenuri: http://keycloak:80/realms/master/protocol/openid-connect/token
  authorizationuri: http://keycloak:80/realms/master/protocol/openid-
connect/auth
  userinfouri: http://keycloak:80/realms/master/protocol/openid-
connect/userinfo
  secretName: eazybankprod-secret
  keycloakclientId: eazybank-gateway-ui
  keycloakclientsecret: MdbTzbRiOHoJAx5CTJsAC8UfkwfgIJZP
```

/helm/eazybank-common/templates/configmap.yaml

```
{-- define "common.configmap" --}
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Values.global.configMapName }}
data:
  MANAGEMENT_ZIPKIN_TRACING_ENDPOINT: {{ .Values.global.zipkinBaseURL }}
  SPRING_PROFILES_ACTIVE: {{ .Values.global.activeProfile }}
  SPRING_CONFIG_IMPORT: {{ .Values.global.configServerURL }}
  EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: {{ .Values.global.eurekaServerURL }}
  SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_JWK-SET-URI: {{ .
  Values.global.keycloakURL }}
  SPRING_SECURITY_OAUTH2_CLIENT_PROVIDER_KEYCLOAK_TOKEN-URI: {{ .
  Values.global.tokenuri }}
  SPRING_SECURITY_OAUTH2_CLIENT_PROVIDER_KEYCLOAK_AUTHORIZATION-URI: {{ .
  Values.global.authorizationuri }}
  SPRING_SECURITY_OAUTH2_CLIENT_PROVIDER_KEYCLOAK_USERINFO-URI: {{ .
  Values.global.userinfouri }}
{{- end -}}
```

/helm/eazybank-common/templates/deployment.yaml

```
{-- if .Values.keycloak_enabled --}
- name: SPRING_SECURITY_OAUTH2_CLIENT_PROVIDER_KEYCLOAK_AUTHORIZATION-URI
  valueFrom:
    configMapKeyRef:
      name: {{ .Values.global.configMapName }}
      key: SPRING_SECURITY_OAUTH2_CLIENT_PROVIDER_KEYCLOAK_AUTHORIZATION-URI
{{- end --}}
{-- if .Values.keycloak_enabled --}
- name: SPRING_SECURITY_OAUTH2_CLIENT_PROVIDER_KEYCLOAK_USERINFO-URI
  valueFrom:
    configMapKeyRef:
      name: {{ .Values.global.configMapName }}
      key: SPRING_SECURITY_OAUTH2_CLIENT_PROVIDER_KEYCLOAK_USERINFO-URI
```

```

{{- end --}}
{{- if .Values.keycloak_enabled --}}
- name: SPRING_SECURITY_OAUTH2_CLIENT_REGISTRATION_EAZYBANK-GATEWAY_CLIENT-ID
  valueFrom:
    secretKeyRef:
      name: {{ .Values.global.secretName }}
      key: keycloakclientid
{{- end --}}
{{- if .Values.keycloak_enabled --}}
- name: SPRING_SECURITY_OAUTH2_CLIENT_REGISTRATION_EAZYBANK-GATEWAY_CLIENT-
SECRET
  valueFrom:
    secretKeyRef:
      name: {{ .Values.global.secretName }}
      key: keycloakclientsecret
{{- end --}}

```

Thay đổi cấu hình helm trong microservice

/helm/eazybank-services/accounts/values.yaml

```

deploymentName: accounts-deployment
deploymentLabel: accounts
appName: accounts

replicaCount: 2

image:
  repository: eazybytes/accounts
  tag: latest

containerPort: 8080

service:
  type: ClusterIP
  port: 8080
  targetPort: 8080

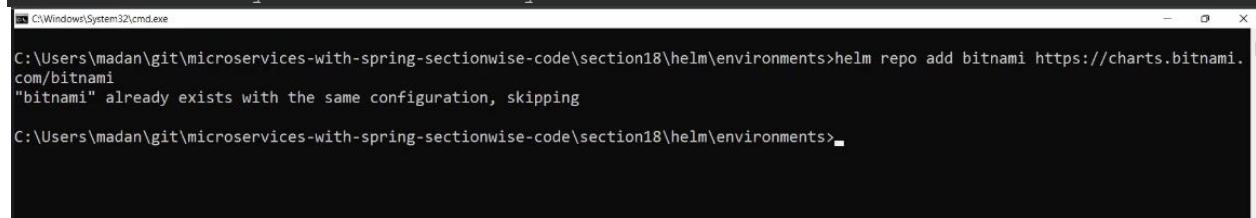
config_enabled: true
zipkin_enabled: true
profile_enabled: true
eureka_enabled: true
keycloak_enabled: false
resouceserver_enabled: true
appname_enabled: true

```

7. Deploy all microservices into K8s cluster and demo of Authorization code flow

Cài đặt Keycloak Auth Server bên trong K8s cluster bằng các lệnh bên dưới:

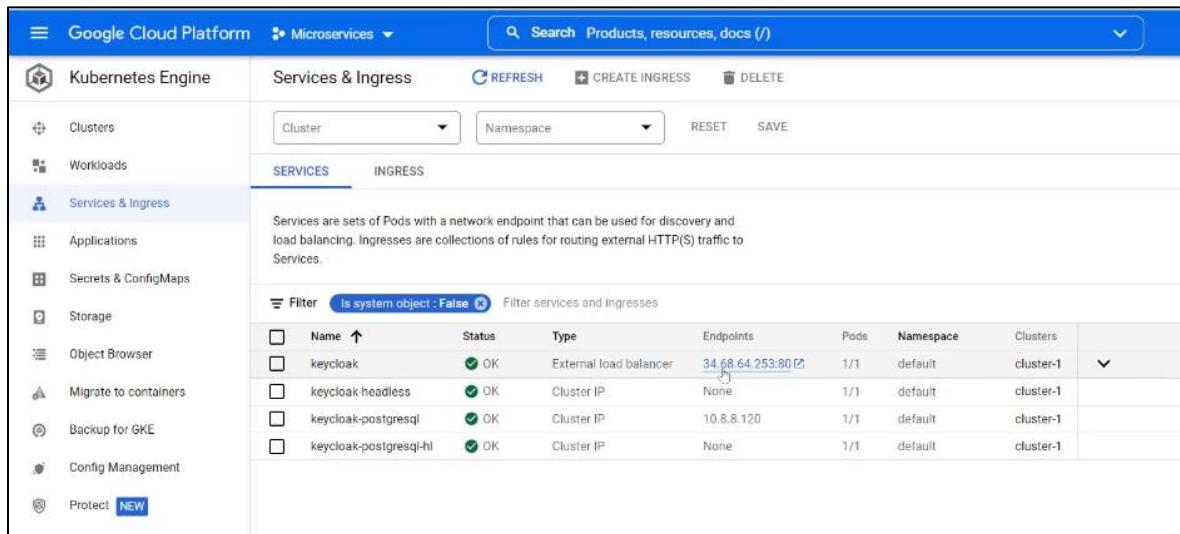
```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm install keycloak bitnami/keycloak
```



The screenshot shows a Windows command prompt window titled 'C:\Windows\System32\cmd.exe'. It displays the command 'helm repo add bitnami https://charts.bitnami.com/bitnami' followed by the output: "'bitnami' already exists with the same configuration, skipping". Below this, the command 'helm install keycloak bitnami/keycloak' is shown.

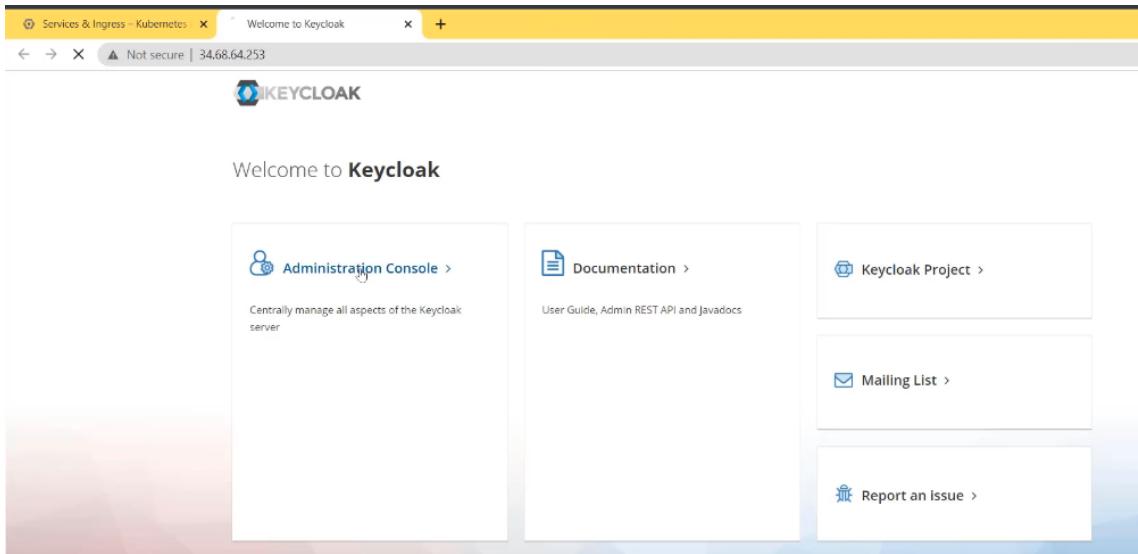


The screenshot shows a Windows command prompt window titled 'C:\Windows\System32\cmd.exe - helm install keycloak --set auth.adminPassword=password bitnami/keycloak'. It displays the command 'helm install keycloak --set auth.adminPassword=password bitnami/keycloak' followed by the output: '#=password bitnami/keycloak'.



The screenshot shows the 'Services & Ingress' page in the Google Cloud Platform Kubernetes Engine interface. The sidebar on the left is collapsed. The main area shows a table of services. The table has columns: Name, Status, Type, Endpoints, Pods, Namespace, and Clusters. The data in the table is as follows:

Name	Status	Type	Endpoints	Pods	Namespace	Clusters
keycloak	OK	External load balancer	34.68.64.253:80	1/1	default	cluster-1
keycloak-headless	OK	Cluster IP	None	1/1	default	cluster-1
keycloak-postgresql	OK	Cluster IP	10.8.8.120	1/1	default	cluster-1
keycloak-postgresql-hl	OK	Cluster IP	None	1/1	default	cluster-1



The screenshot shows a web browser window with the URL '34.68.64.253'. The title bar says 'Welcome to Keycloak'. The page content includes the Keycloak logo and the text 'Welcome to Keycloak'. Below this, there are links for 'Administration Console', 'Documentation', 'Keycloak Project', 'Mailing List', and 'Report an Issue'.

KEYCLOAK

Master

Configure

- Realm Settings
- Clients**
- Client Scopes
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

- Groups
- Users

Clients > Add Client

Add Client

Import

Client ID *

Client Protocol

Root URL



Identity Providers

User Federation

Authentication

Manage

- Groups
- Users
- Sessions
- Events
- Import
- Export

Enabled

Always Display in Console

Consent Required

Login Theme

Client Protocol

Access Type

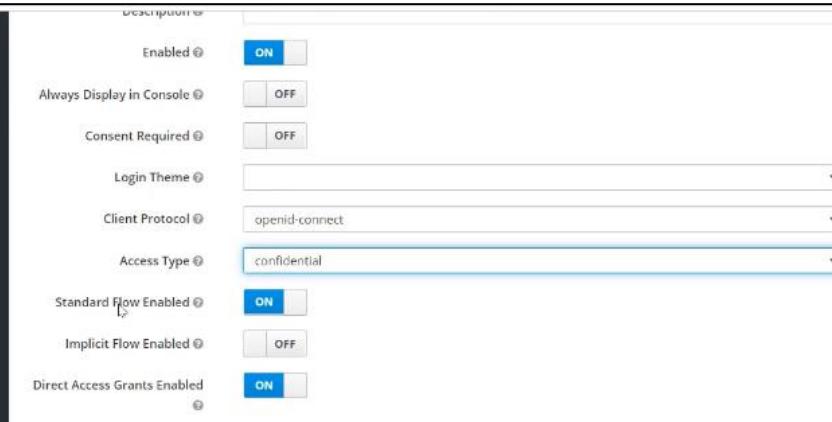
Standard Flow Enabled

Implicit Flow Enabled

Direct Access Grants Enabled

Service Accounts Enabled

OAuth 2.0 Device Authorization Grant Enabled



Master

Configure

- Realm Settings
- Clients**
- Client Scopes
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

- Groups
- Users
- Sessions

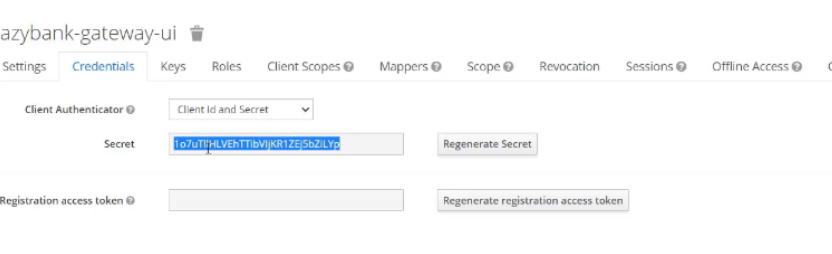
Clients > eazybank-gateway-ui

Settings Credentials Keys Roles Client Scopes Mappers Scope Sessions Offline Access Clustering Installation

Client Authenticator

Secret

Registration access token



```
C:\Windows\System32\cmd.exe - helm dependencies build
Microsoft Windows [Version 10.0.22000.675]
(c) Microsoft Corporation. All rights reserved.

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section18\helm>cd environments

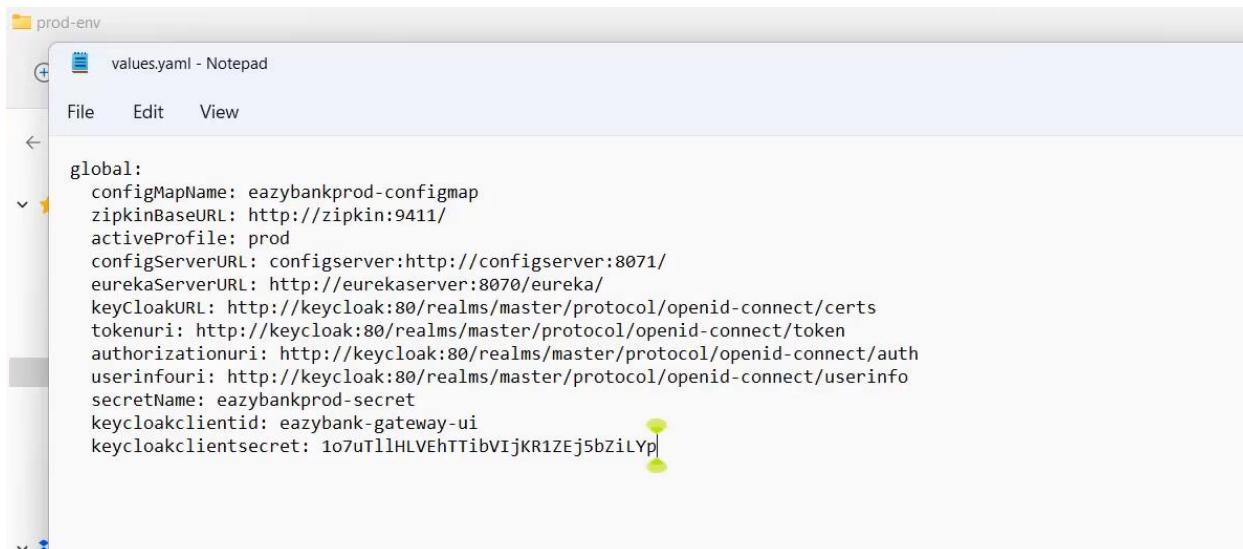
C:\Users\madan\git\microservices-with-spring-sectionwise-code\section18\helm\environments>cd dev-env

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section18\helm\environments\dev-env>helm dependencies build
Hang tight while we grab the latest from your chart repositories...   ↴
...Successfully got an update from the "bitnami" chart repository
Update Complete. 🎉Happy Helming!🎉
Saving 8 charts
Deleting outdated charts

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section18\helm\environments\dev-env>cd..

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section18\helm\environments>cd prod-env

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section18\helm\environments\prod-env>helm dependencies build
Hang tight while we grab the latest from your chart repositories...
```



```
M175. Deploy all microservices into K8s cluster and demo of Authorization code flow
(c) Microsoft Corporation. All rights reserved.

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section18\helm>cd environments

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section18\helm\environments>cd dev-env

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section18\helm\environments\dev-env>helm dependencies build
Hang tight while we grab the latest from your chart repositories...   ↴
...Successfully got an update from the "bitnami" chart repository
Update Complete. 🎉Happy Helming!🎉
Saving 8 charts
Deleting outdated charts

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section18\helm\environments\dev-env>cd..

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section18\helm\environments>cd prod-env

C:\Users\madan\git\microservices-with-spring-sectionwise-code\section18\helm\environments\prod-env>helm dependencies build
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "bitnami" chart repository
Update Complete. 🎉Happy Helming!🎉
```

Google Cloud Platform Microservices

Kubernetes Engine

Workloads

REFRESH DEPLOY DELETE

Clusters Workloads Services & Ingress Applications Secrets & ConfigMaps Storage Object Browser Migrate to containers Backup for GKE Config Management Protect NEW

Cluster Namespace RESET SAVE

OVERVIEW COST OPTIMIZATION

Filter Is system object : False Filter workloads

<input type="checkbox"/>	Name ↑	Status	Type	Pods	Namespace	Cluster
<input type="checkbox"/>	accounts-deployment	OK	Deployment	2/2	default	cluster-1
<input type="checkbox"/>	cards-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	configserver-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	eurekaserver-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	gatewayserver-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	keycloak	OK	Stateful Set	1/1	default	cluster-1
<input type="checkbox"/>	keycloak-postgresql	OK	Stateful Set	1/1	default	cluster-1
<input type="checkbox"/>	loans-deployment	OK	Deployment	1/1	default	cluster-1
<input type="checkbox"/>	zipkin-deployment	OK	Deployment	1/1	default	cluster-1

Google Cloud Platform Microservices

Kubernetes Engine

Services & Ingress

REFRESH CREATE INGRESS DELETE

Clusters Workloads Services & Ingress Applications Secrets & ConfigMaps Storage Object Browser Migrate to containers Backup for GKE Config Management Protect NEW Marketplace

Cluster Namespace RESET SAVE

SERVICES INGRESS

Filter Is system object : False Filter services and ingresses

<input type="checkbox"/>	Name ↑	Status	Type	Endpoints	Pods	Namespace	Clusters
<input type="checkbox"/>	accounts	OK	Cluster IP	10.8.10.69	2/2	default	cluster-1
<input type="checkbox"/>	cards	OK	Cluster IP	10.8.5.201	1/1	default	cluster-1
<input type="checkbox"/>	configserver	OK	Cluster IP	10.8.1.62	1/1	default	cluster-1
<input type="checkbox"/>	eurekaserver	OK	Cluster IP	10.8.10.137	1/1	default	cluster-1
<input type="checkbox"/>	gatewayserver	OK	External load balancer	35.184.193.83:8070	1/1	default	cluster-1
<input type="checkbox"/>	keycloak	OK	External load balancer	34.68.64.253:80	1/1	default	cluster-1
<input type="checkbox"/>	keycloak-headless	OK	Cluster IP	None	1/1	default	cluster-1
<input type="checkbox"/>	keycloak-postgresql	OK	Cluster IP	10.8.8.120	1/1	default	cluster-1
<input type="checkbox"/>	keycloak-postgresql-hl	OK	Cluster IP	None	1/1	default	cluster-1
<input type="checkbox"/>	loans	OK	Cluster IP	10.8.0.168	1/1	default	cluster-1
<input type="checkbox"/>	zipkin	OK	Cluster IP	10.8.8.107	1/1	default	cluster-1

Section 17: Introduction to K8s Ingress & Service Mesh (Istio)

1. Introduction to Kubernetes Ingress

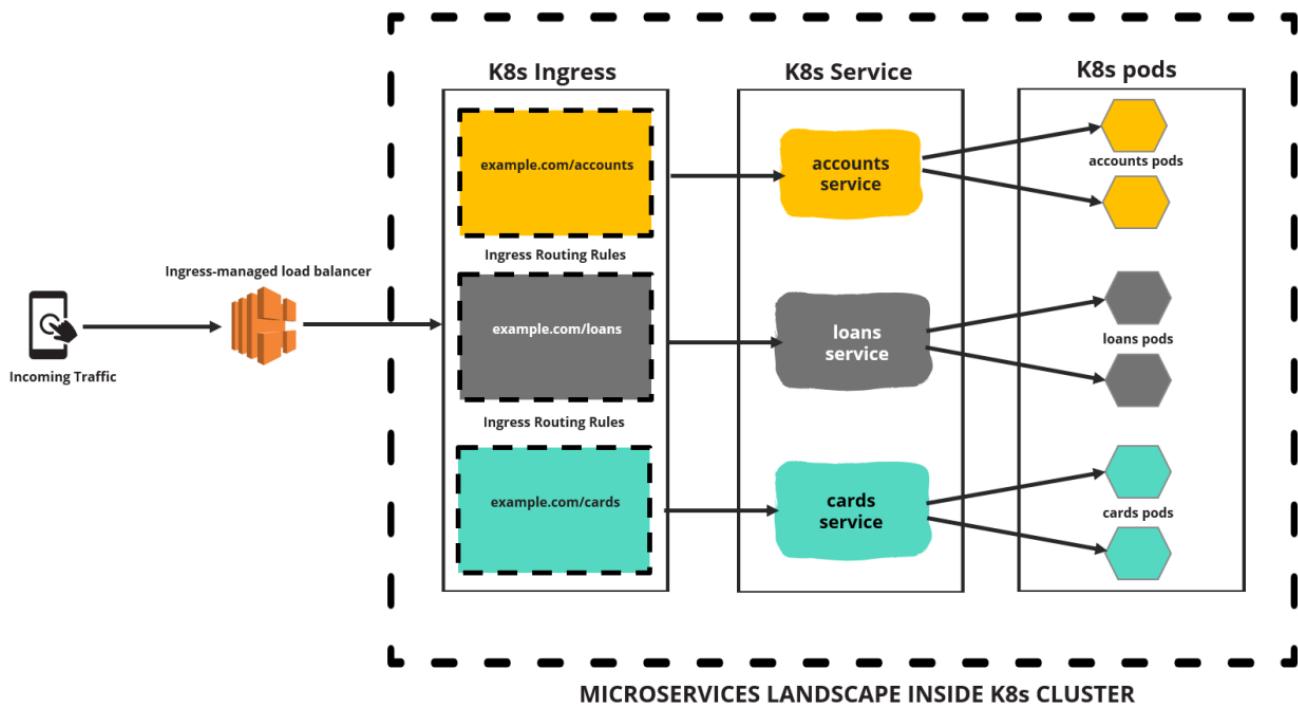
Kubernetes Ingress là một khái niệm quan trọng trong việc quản lý và điều phối lưu lượng truy cập vào các ứng dụng chạy trên Kubernetes. Nó cung cấp một cách linh hoạt để điều hướng yêu cầu HTTP và HTTPS từ bên ngoài đến các dịch vụ (Service) chạy trong cluster Kubernetes.

Trong Kubernetes, các dịch vụ (Service) đại diện cho các ứng dụng của bạn và được liên kết với một IP và port. Tuy nhiên, việc quản lý cách các yêu cầu từ bên ngoài được chuyển tiếp đến các dịch vụ này có thể phức tạp, đặc biệt là khi có nhiều dịch vụ chạy trên cluster.

Đây là khi Kubernetes Ingress xuất hiện. Ingress là một tài nguyên trong Kubernetes, cho phép bạn định nghĩa các quy tắc để điều hướng yêu cầu HTTP/HTTPS đến các dịch vụ tương ứng trong cluster dựa trên các tiêu chí như tên miền, URL path, hoặc header.

Ví dụ, bạn có thể cấu hình Ingress để chuyển hướng yêu cầu từ "www.example.com" đến dịch vụ A và yêu cầu từ "api.example.com" đến dịch vụ B. Điều này giúp quản lý lưu lượng truy cập và tạo điều kiện để triển khai các quy tắc định tuyến linh hoạt mà không cần phải sửa đổi cấu hình của các dịch vụ riêng lẻ.

Để sử dụng Ingress, bạn cần một Ingress Controller, là một thành phần riêng biệt xử lý và thực hiện các quy tắc định tuyến được định nghĩa trong Ingress. Có nhiều Ingress Controller được hỗ trợ, như Nginx Ingress Controller, Traefik, HAProxy, và mỗi Ingress Controller có cách cấu hình riêng.



- K8s Ingress hiển thị các tuyến(route) HTTP và HTTPS từ bên ngoài cluster đến các dịch vụ trong cluster. Traffic routing (Định tuyến lưu lượng) được kiểm soát bởi các quy tắc được xác định trên Ingress resource.
- K8s Ingress đánh giá tất cả các quy tắc định tuyến và quản lý các chuyển hướng. Nhưng nó không thể xử lý tất cả các loại định tuyến phức tạp như Spring Cloud Gateway.
- K8s Ingress có thể xử authentication, authorization, SSL/TLS termination.
- Bạn có thể cần triển khai ngress controller chẳng hạn như ingress-nginx. Bạn có thể chọn từ một số Ingress controllers có sẵn trên thị trường.
- Cũng giống như Spring Cloud Gateway, K8s ingress có thể hoạt động như một entry point/edge server của K8s cluster.
- Việc thiết lập K8s Ingress thuộc trách nhiệm của nhóm DevOps hơn là nhóm Dev.

2. Introduction to Service mesh

- Service mesh (Lưới dịch vụ) giúp các tổ chức chạy các ứng dụng phân tán, dựa trên microservice ở mọi nơi. Tại sao lại sử dụng nó? Nó cho phép các tổ chức bảo mật, kết nối và giám sát các microservice.
- Service mesh có thể giúp bảo mật ở cấp ứng dụng với xác thực, ủy quyền và mã hóa dựa trên danh tính mạnh mẽ.
- Service mesh có thể giúp giám sát bằng metrics, logs và trace cho tất cả lưu lượng truy cập trong một cluster, bao gồm cả đầu vào và đầu ra của cluster.
- Service mesh quản lý các luồng lưu lượng giữa các dịch vụ, thực thi các chính sách truy cập và tổng hợp dữ liệu đo từ xa mà không yêu cầu thay đổi mã ứng dụng.
- Service mesh có thể giúp kiểm soát chi tiết hành vi lưu lượng với các quy tắc định tuyến phong phú, thử lại, chuyển đổi dự phòng và chèn lỗi.
- Service mesh có thể bảo mật giao tiếp service-to-service nội bộ trong một cluster bằng TLS tương hỗ (mTLS)

3. Deep dive on Service mesh and Istio

Sự cố mà Service Mesh đang cố gắng giải quyết

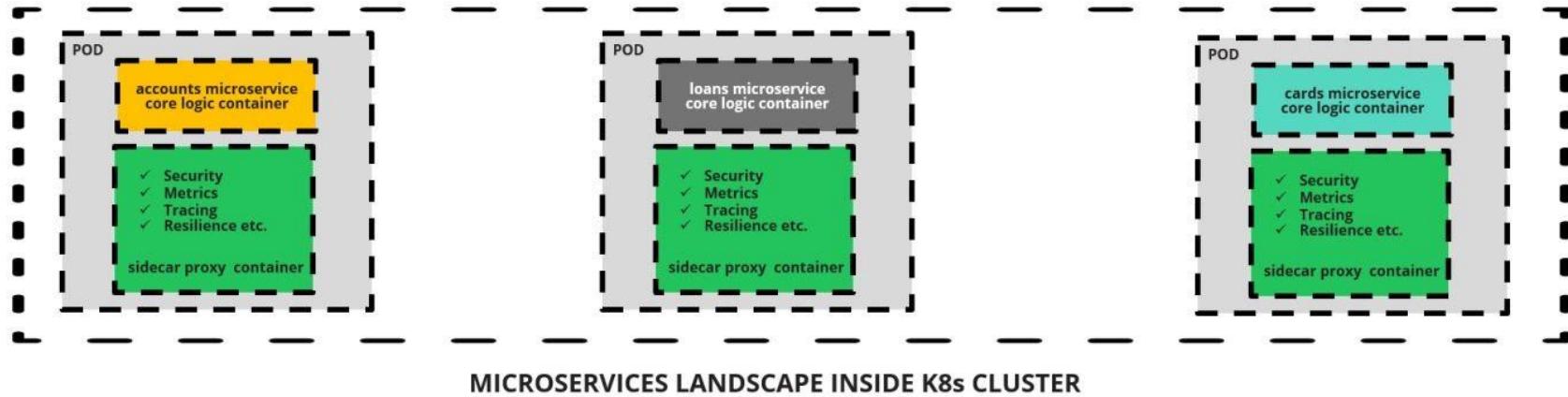


Mỗi microservice có nhiều mã/cấu hình liên quan đến Security, tracing, v.v. không liên quan đến logic nghiệp vụ. Tất cả các mã/cấu hình bổ sung có trong mỗi microservice sẽ khiến chúng trở nên phức tạp.



Developer cần quản lý những thay đổi này một cách nhất quán trong tất cả các microservice, điều này không phù hợp với trọng tâm chính của họ là xây dựng logic nghiệp vụ.

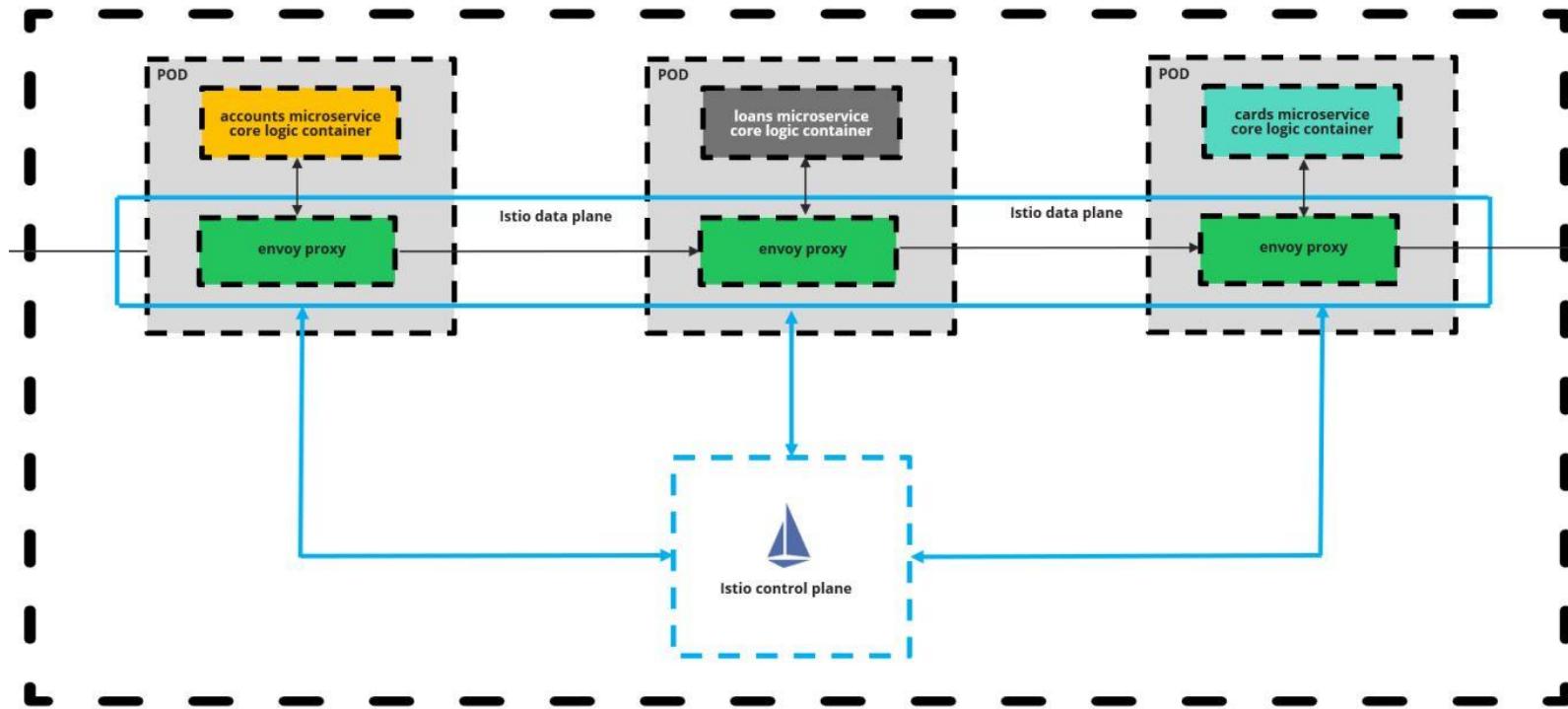
Cách Service Mesh giúp dễ dàng khi xây dựng microservice



Sidecar proxy container được triển khai cùng với mỗi dịch vụ mà bạn bắt đầu trong cluster. Đây còn được gọi là mô hình Sidecar. Pattern này được đặt tên là Sidecar vì nó giống như một chiếc sidecar gắn vào xe máy. Trong pattern, sidecar được gắn vào ứng dụng gốc và cung cấp các tính năng hỗ trợ cho ứng dụng. Sidecar cũng chia sẻ vòng đời giống như ứng dụng gốc, được tạo và ngừng hoạt động cùng với ứng dụng gốc.



Một sidecar độc lập với ứng dụng chính của nó về mặt môi trường thời gian chạy và ngôn ngữ lập trình, vì vậy bạn không cần phải phát triển một sidecar cho mỗi ngôn ngữ. Các nhà phát triển cũng sẽ không phải phát triển tất cả các thành phần liên quan đến non business logic .



Istio service mesh hoạt động như thế nào

- Istio có hai thành phần: data plane và control plane
- Một phẳng dữ liệu(data plane) là giao tiếp giữa các dịch vụ. Nếu không có service mesh, mạng sẽ không hiểu lưu lượng được gửi qua và không thể đưa ra bất kỳ quyết định nào dựa trên loại lưu lượng đó là gì, hoặc nó đến từ ai.
- Service mesh sử dụng proxy để chặn tất cả lưu lượng truy cập mạng của bạn, cho phép một loạt các tính năng nhận biết ứng dụng dựa trên cấu hình.
- Proxy Envoy được triển khai cùng với từng dịch vụ mà bạn bắt đầu trong cluster hoặc chạy cùng với các dịch vụ chạy trên máy ảo.
- Control plane lấy cấu hình mong muốn và chế độ xem các dịch vụ của nó, đồng thời lập trình động các máy chủ proxy, cập nhật chúng khi các quy tắc hoặc môi trường thay đổi.