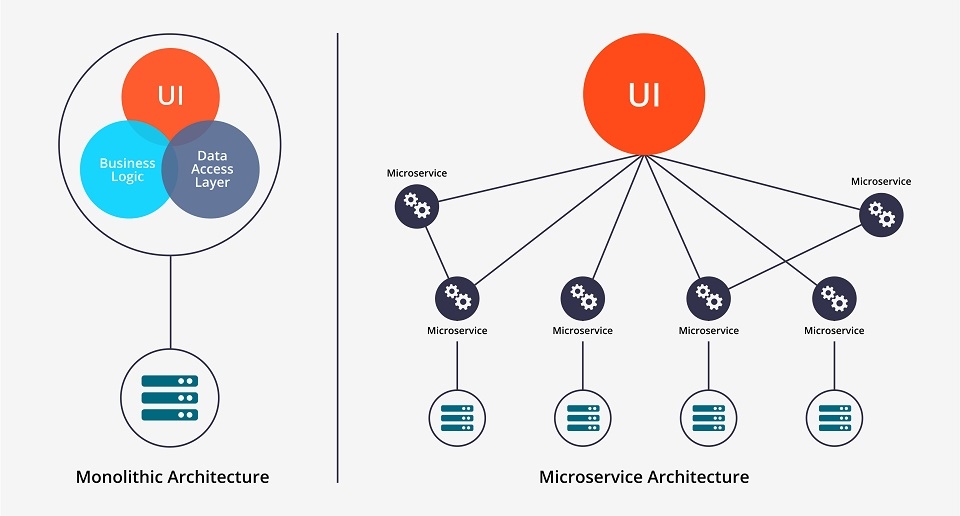
**Microservice**

1. **Microservices là gì?**



Microservices là một phương pháp để phát triển phần mềm, trong đó ứng dụng được xây dựng thành một tập hợp các dịch vụ nhỏ, độc lập nhau. Mỗi dịch vụ đảm nhận một chức năng cụ thể và có thể triển khai độc lập. Các dịch vụ này có thể được viết bằng nhiều ngôn ngữ lập trình khác nhau và sử dụng các công nghệ lưu trữ dữ liệu khác nhau. Các dịch vụ vi mô thường giao tiếp với nhau thông qua API, đồng thời có thể sử dụng các công cụ và giải pháp đã tồn tại trong hệ sinh thái dịch vụ web và RESTful.

Microservices giúp giải quyết những khó khăn của kiến trúc hệ thống lớn bằng cách chia nhỏ ứng dụng thành các thành phần nhỏ hơn, dễ quản lý và phát triển. Điều này mang lại tính linh hoạt, khả năng mở rộng và khả năng module hóa trong việc phát triển phần mềm. Nó cũng cho phép các nhóm phát triển làm việc độc lập và triển khai các phần của ứng dụng mà không phụ thuộc vào nhau.

Microservices đã được áp dụng rộng rãi trong các ứng dụng công nghệ nổi tiếng như Netflix, eBay, Amazon và PayPal, với mục tiêu tăng cường khả năng linh hoạt, mở rộng và hiệu suất của hệ thống phần mềm.

Ưu điểm:

* Triển khai và phân phối ứng dụng lớn và phức tạp một cách dễ dàng.
* Tăng cường khả năng bảo trì nhờ các Microservice có quy mô nhỏ, dễ hiểu và dễ thay đổi.
* Kiểm thử dễ dàng và phát hiện lỗi sớm do quy mô nhỏ của các Microservice.
* Khả năng triển khai độc lập của từng Microservice.
* Khả năng phát triển nhanh chóng với sự phân công công việc cho các đội phát triển riêng biệt.
* Tính cách ly lỗi, khi một Microservice gặp sự cố, chỉ ảnh hưởng đến service đó, không làm ảnh hưởng đến toàn bộ hệ thống.
* Dễ dàng thay đổi công nghệ mới trong từng Microservice.

Nhược điểm:

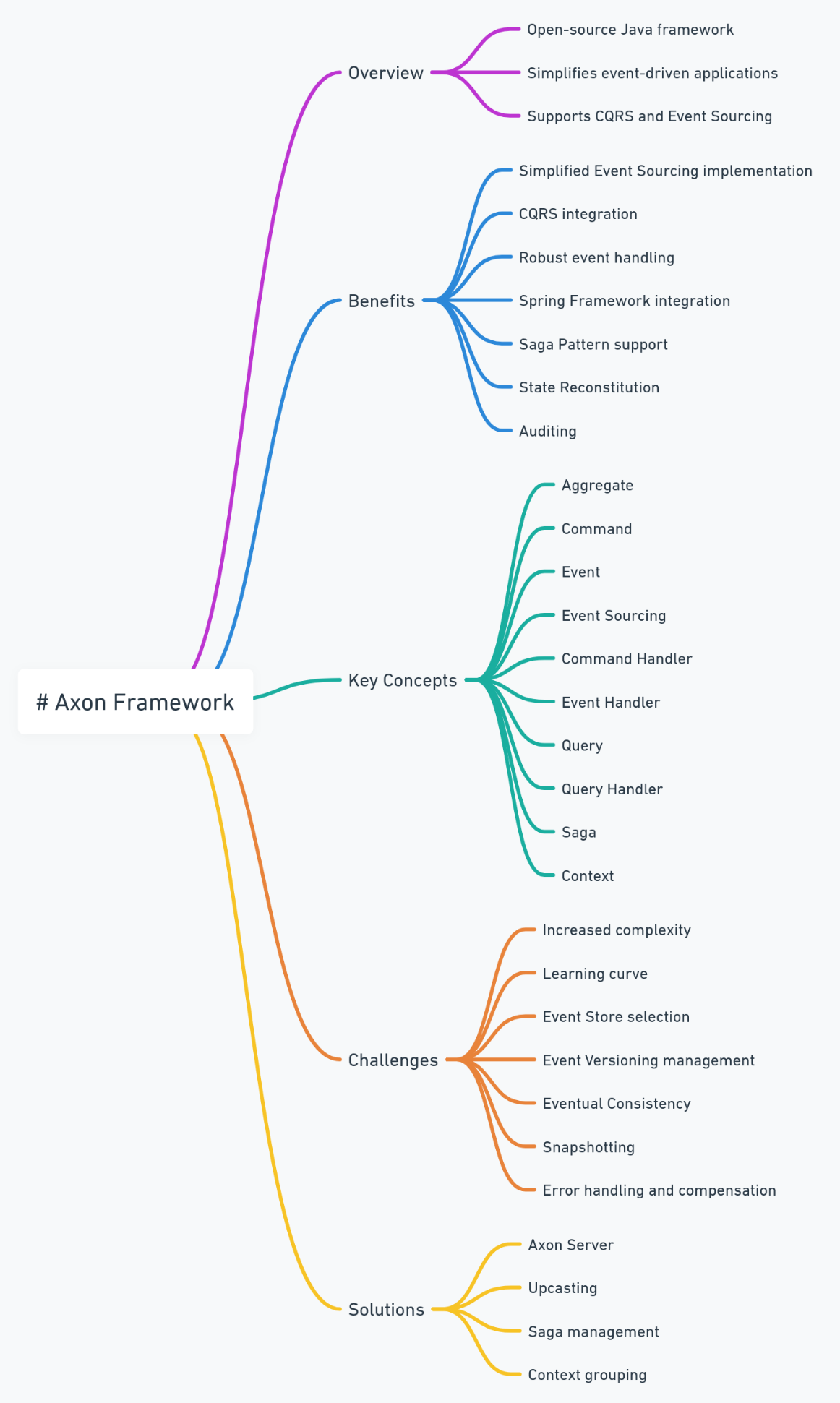
* Phức tạp trong việc xây dựng hệ thống phân tán.
* Đòi hỏi việc quản lý và xử lý giao tiếp giữa các Microservice.
* Xử lý sự cố một cách phức tạp do luồng xử lý đi qua nhiều Microservice.
* Đòi hỏi sự phối hợp giữa các đội phát triển khi thực hiện các yêu cầu trải rộng trên nhiều Microservice.
* Khó khăn trong việc đảm bảo toàn vẹn dữ liệu khi áp dụng cấu trúc phân vùng.
* Quản lý và triển khai Microservices có thể phức tạp nếu thực hiện thủ công.
* Xử lý các vấn đề liên quan đến kết nối chậm, lỗi và đồng bộ thông điệp giữa các Microservice.

Nguồn: <https://fptshop.com.vn/tin-tuc/danh-gia/microservices-171658>

1. **Eureka/Discovery Server**

Eureka Server là một ứng dụng lưu trữ thông tin về tất cả các ứng dụng dịch vụ khách hàng. Mọi dịch vụ sẽ đăng ký vào máy chủ Eureka và máy chủ Eureka biết tất cả các ứng dụng khách chạy trên mỗi cổng và địa chỉ IP.

1. **Axon Framework**



Axon framework là một framework hiện đại được ứng dụng phổ biến trong microservice để xây dựng các ứng dụng hướng dự kiện. Axon Framework dựa trên các nguyên tắc thiết kế như CQRS ( Command Query Responsibility Segregation ) và DDD (Domain Driven Design).

1. **Rate Limiter**

Rate Limit tức là hạn chế (limit) số lượng request gửi/nhận (rate) đến hệ thống. nói thì nghe đơn gỉản vậy thôi. Trên thực tế, người ta phải sử dụng 1 số thuật toán để đảm bảo chạy nhanh, chính xác mà lại ít tốn bộ nhớ. Giả sử như hệ thống của chúng ta nhận được hàng nghìn request nhưng mà trong số đó chỉ xử lí được trăm request/s chẳng hạn, và số request còn lại thì bị lỗi (do CPU hệ thống đang quá tải không thể xử lí được).

Để giải quyết được vấn đề này thì cơ chế Rate Limiting đã ra đời. Mục đích của nó chỉ cho phép nhận 1 số lượng request nhất định trong 1 đơn vị thời gian. Nếu quá sẽ trả về response lỗi.

1. **Message Queue**

Message queue là một kiến trúc cung cấp giao tiếp không đồng bộ. Ý nghĩa của queue ở đây chính là 1 hàng đợi chứa message chờ để được xử lý tuần tự theo cơ chế vào trước thì ra trước (FIFO - First In First Out). Một message là các dữ liệu cần vận chuyển giữa người gửi và người nhận. Vậy có thể hiểu đơn giản, message queue giống như một hòm thư email của chúng ta. Email có lẽ là ví dụ tốt nhất về giao tiếp không đồng bộ. Khi một email được gửi đi, người gửi tiếp tục xử lý những thứ khác mà không cần phản hồi ngay lập tức từ người nhận. Cách xử lý tin nhắn này tách người gửi khỏi người nhận để họ không cần phải tương tác với hàng đợi tin nhắn cùng một lúc.

A line of mails and a few envelopes

Description automatically generated with medium confidence

A diagram of a diagram

Description automatically generated

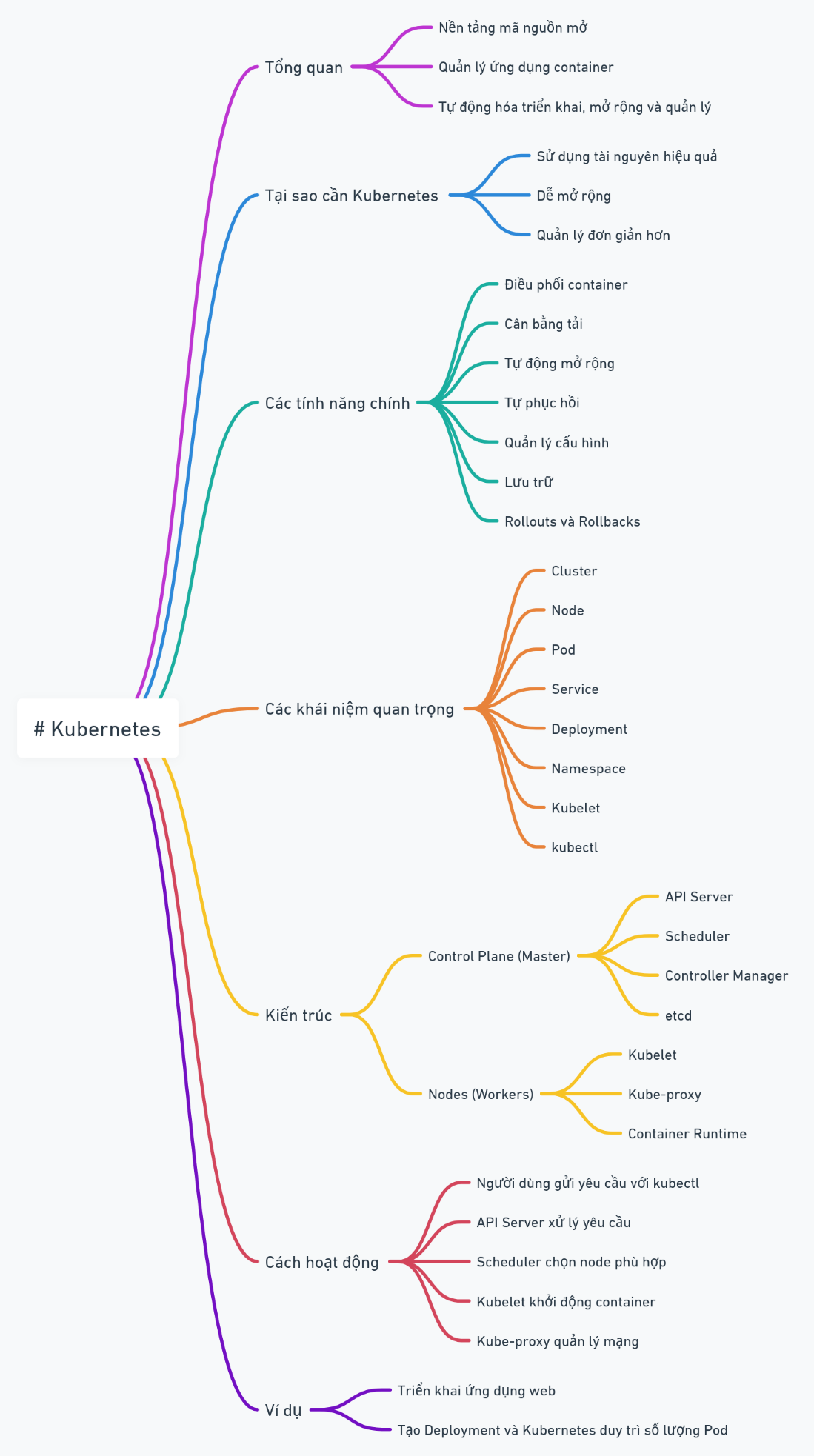
Ưu điểm:

* **Dự phòng**: Sau khi lưu trữ tin nhắn, hàng đợi đảm bảo tin nhắn sẽ chỉ bị loại bỏ khi quá trình đọc nó xác nhận thành công trong việc đọc và xử lý. Nếu có gì sai sót, tin nhắn sẽ không bị mất và được xử lý lại sau này.
* **Nhắn tin không đồng bộ**: Trong trường hợp ứng dụng của bạn không yêu cầu phản hồi đúng cách cho một quy trình, các consumer có thể nhận và xử lý theo logic và tốc độ riêng của nó.
* **Khả năng phục hồi**: Ví dụ: giả sử hệ thống của bạn bao gồm 2 microservice, một để xử lý đơn đặt hàng và một dịch vụ khác để gửi email. Có một hàng đợi cho biết rằng một email cần được gửi đi có nghĩa là ngay cả khi hệ thống email của bạn không hoạt động, dịch vụ xử lý đơn đặt hàng vẫn có thể tiếp tục nhận và xử lý đơn đặt hàng. Khi dịch vụ email trực tuyến trở lại, nó có thể bắt đầu đọc tin nhắn và gửi email.
* **Khả năng mở rộng**: Hàng đợi cho phép bạn tách hệ thống của mình thành các microservices khác nhau và chia tỷ lệ chúng theo nhu cầu.

Nhược điểm:

* **Làm phức tạp hệ thống**
* **Phải có message format:** Từ 2 phía producer và consumer cần phải thống nhất format message để có thể gửi và nhận message.
* **Monitor Queue là cần thiết:** Bạn cần phải theo dõi queue của mình để đảm bảo queue không quá nhiều hay đầy queue.
* **Khó xử lý đồng bộ:** Không phải lúc nào message queue cũng là lựa chọn hàng đầu khi chúng ta xây dựng hệ thống. Sẽ có nhiều trường hợp hệ thống bắt buộc cần phải xử lý đồng bộ giữa các service, khi đó ta sẽ cần lựa chọn những cơ chế phù hợp hơn như Remote Procedure Invocation (RPI).

1. **Kurbernetes**



**\* Các bước để apply một image vào k8s:**

1. **Tạo một file container chứa các image cần lưu**

Vd:

version: "3.8"

services:

    discoverserver:

        build:

            context: .

            dockerfile: discoverserver/Dockerfile

        ports:

            - "8761:8761"

        networks:

            - event-sourcing-mcs

networks:

    event-sourcing-mcs:

        driver: bridge

1. **Truy cập vào minikube để pushing container**

& minikube -p minikube docker-env --shell powershell | Invoke-Expression

1. **Sau đó đẩy image vừa làm ở file trên vào k8s**

docker compose up -d

1. **Tạo một file k8s-deployment để deployment image**

Vd:

# Deployment

apiVersion: apps/v1

kind: Deployment

metadata:

name: discoverserver-deployment

labels:

app: discoverserver

spec:

replicas: 3

selector:

matchLabels:

app: discoverserver

template:

metadata:

labels:

app: discoverserver

spec:

containers:

- name: discoverserver

image: eventsourcingmicroservice-discoverserver:latest

imagePullPolicy: IfNotPresent

ports:

- containerPort: 8761

---

# Service

apiVersion: v1

kind: Service

metadata:

name: discoverserver

spec:

selector:

app: discoverserver

ports:

- protocol: TCP

port: 8761

targetPort: 8761

type: NodePort

1. **Sau đó sử dụng lệnh để apply cấu hình**

kubectl apply -f k8s-deployment.yml

1. **Sử dụng minikube để bên ngoài có thể giao tiếp**

minikube service discoverserver