# Lesson summary

## Section 1: Fundamental Ideas Around Microservice

### What is microservice?

* Ở Monolithic server sẽ bao gồm tất cả các middleware, router, business logic, database access,… nói lại là gom hết vào một cục để implement nhiều features.
* Còn ở microservice thì sẽ chia ra nhiều cục, mỗi cục chịu trách nhiệm một tính năng riêng biệt. “A single microservice contains Routing, Middleware Business Logic, Database access to implement **one feature** of our app”. vì mỗi cục độc lập vơi nhau nên khi 1 cục bị crash hay chạy sai thì các cục còn lại không bị ảnh hưởng

### Data in Microservice

* Quản lý data giữa các service. Đây là một vấn đề lớn của các kiến trúc microservice. Cách để lưu data bên trong 1 service và cách để giao tiếp data giữa các service khác với nhau
* Cần tuân thủ 2 luật:
  + Mỗi service sẽ có 1 DB riêng (nếu nó cần 1)
  + Các service sẽ không bao giờ truy cập trực tiếp đến database của service khác
* Lý do DB cho mỗi service:
  + Chúng ta muốn mỗi service chạy độc lập với các service khác
  + DB schema/structure có thể thay đổi không mong đợi
  + Một số service hoạt động tốt hơn với loại DB khác (sql vs nosql)

### Big problems with Data

* Mỗi lần tạo một service mới hay chỉnh sửa một service khác và cần access đến một database đã tồn tại là một vấn đề lớn với kiến trúc này vì mỗi service chạy độc lập với nhau và không được tiếp cần đến DB của service khác để đảm bảo tính độc lập

### Sync & Async communication Between Service

* Two very general strategies mà ta có thể solve vấn đề ở mục 3
* **Sync**: Các service giao tiếp với mỗi service khác bằng cách dùng trực tiếp các request.
  + Upsides:
* khái niệm dễ hiểu
* service không cần có database, chỉ việc gửi request đến các service khác để lấy data cần thiết
  + Downsides:
* có sự phụ thuộc giữa các service
* nếu có bất kì request bị fail, thì toàn bộ request đều fail
* tổng thời gian request nhanh nhất = 1 request chậm nhất
* can easily introduce webs of requests (ex: gửi req đến service A B C trong service A B C lại gửi req đến các service khác)
* **Async**: Các service giao tiếp với mỗi service khác bằng cách dùng các event. “just like the database-per-service pattern, async communication is going to seem bizarre and inefficient”. Thằng này có 2 cách
* **Event-Based Communication**: khá củ chuối và nó khá giống Sync, thay vì dùng req thì ta dùng event, thằng này có đủ ưu và nhược giống Sync luôn.
* **Crazy way of String Data**: Là tạo 1 database cho service đó. Với cách này thì câu hỏi là làm sao mà thông tin được nhận chính xác và đồng bộ với các database khác, trả lời câu hỏi này ta dùng event để làm. Ex: service A có tính năng đáng ký người dùng và lưu thông tin người dùng vào DB và service B có tính năng dựa vào id người dùng để kiểm tra xem họ đã tiêm 2 mũi vaccine chưa, thì khi có req gửi đến service A để tạo tài khoản thì service A tạo thông tin user và lưu vào DB sau đó emit một event lên event bus nói rằng “hey tao mới tạo thông tin user mới mày xem coi có thằng service nào cần thông tin này không thì báo cho nó”. Event bus nhận emit và gửi thông tin mà service A gửi lên cho service B vì service B cần thông tin của các user để lưu vào DB để mốt có req gửi lên thì biết trong ứng dụng có user này và nó đã tiêm mấy mũi vaccine.

### Pros and Cons of Async communication

* Pros:
  + Không có sự phụ thuộc với các service khác
  + Phản hồi sẽ được nhanh hơn
* Cons:
  + Data duplication
  + Khó hiểu hơn

## Section 2: A Mini-Microservices App

### Event Bus overview

* Many different implement. RabbitMQ, Kafka, NATS,…
* Receives events, publishes them to listeners (nhận thông tin từ các service khác rồi gửi thông tin đó cho những service đang lắng nghe emit)
* Many different subtle features that make async communication way easier or way harder

## Section 3: Running Services with Docker

### ?

### ?

## Section 4:

## Section 5:

## Section 6:

## Section 7:

## Section 8:

## Section 9:

## Section 10:

## Section 11:

## Section 12:

## Section 13:

## Section 14:

## Section 15:

## Section 16:

## Section 17:

## Section 18:

## Section 19:

## Section 20:

## Section 21:

## Section 22:

## Section 23:

## Section 24:

## Section 25:

## Section 26:

# Keyword research

## Database-per-service pattern