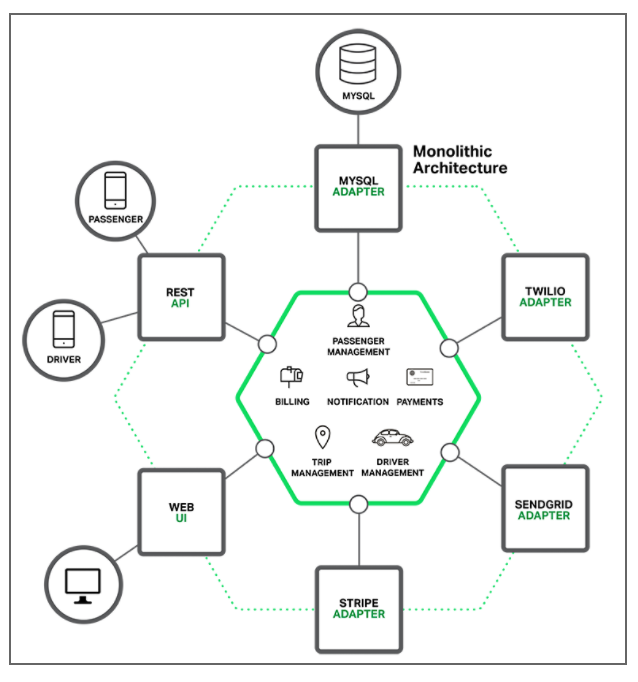
So sách Monolithic Architecture và MicroServicess Architecture

1. Monolithic Architecture

Ví dụ: giả sử bạn xây dựng một ứng dụng gọi xe

Đối với cấu trúc nguyên khối (Monolithic Architecture)



Các dịch vụ (Driver Management, Trip Management, Billing, Payment) sẽ được gom về một khối duy nhất - chạy trên 1 docker hoặc server nào đó, Để các dịch vụ này tương tác được với các người dùng, nó sẽ cần có các REST API, hoặc các ADAPTER

Nhược điểm:

Lúc mới đầu dự án, sẽ có thể chỉ có vài dịch vụ như trên, tuy nhiên các ứng dụng nếu thành công thì thướng sẽ tiếp tục phát triển thêm các dịch vụ và dần dần hình thành một khối khổng lồ ở mọi nơi như Services, các API, Adapter,… .Một ứng dụng quá lớn khiến cho việc hiểu đầy đủ ứng dụng đó là rất khó. Do đó, việc sửa lỗi và phát triển thêm các tính năng mới cũng là một điều rất khó.

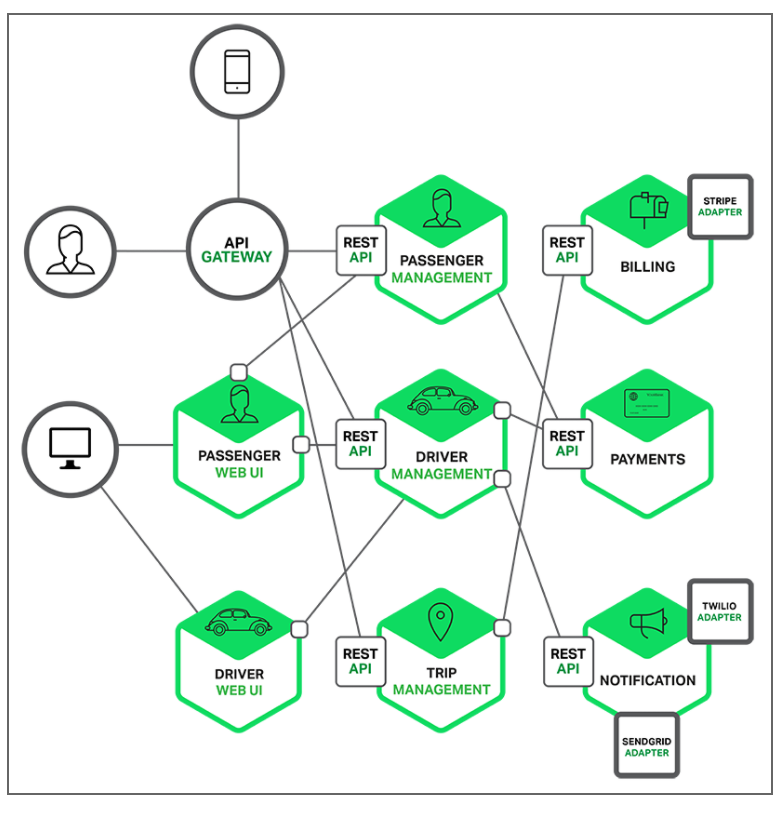
Thêm vào đó, vì ứng dụng lớn khiến cho việc khởi động cũng rất lâu ở mọi mặt khiến cho công ty bị ảnh hưởng về năng suất lao động

Hơn nữa, xu hướng phát triển ngày nay là luôn luôn thay đổi, những giá trị của ngày hôm nay thì có thể ngày mai sẽ không còn giá trị nữa, nên các ứng dụng (đặc biệc dạng Saas) phải liên tục phát triển à thích ứng với các yêu cầu của người dùng nếu chúng không muốn trở thành các ứng dụng cũ, vì vậy các Developer phải liên tục phát triển sản phẩm của mình theo từng ngày để đáp ứng được nhu cầu cho khách hàng. Đây là một điều rất khó thực hiện đối với các kiến trúc kiểu Monolithic (vì bạn phải triển khai lại toàn bộ ứng dụng nếu muốn cập nhật bất kỳ phần nào).

Độ tin cậy trong Monolithic cũng là một vấn đề, bởi vì tất cả các module đang chạy trong cùng một quy trình, một lỗi trong bất kỳ module nào cũng có thể ảnh hưởng đến toàn bộ quy trình

Một điểm cũng cực kỳ bất lợi nếu bạn sử dụng Monolithic Architecture đó là framework, vì ứng dụng của bạn là lớn nên bạn rất khó để di chuyển code từ framework bạn đang sử dụng, sang franmework mới tốt hơn.

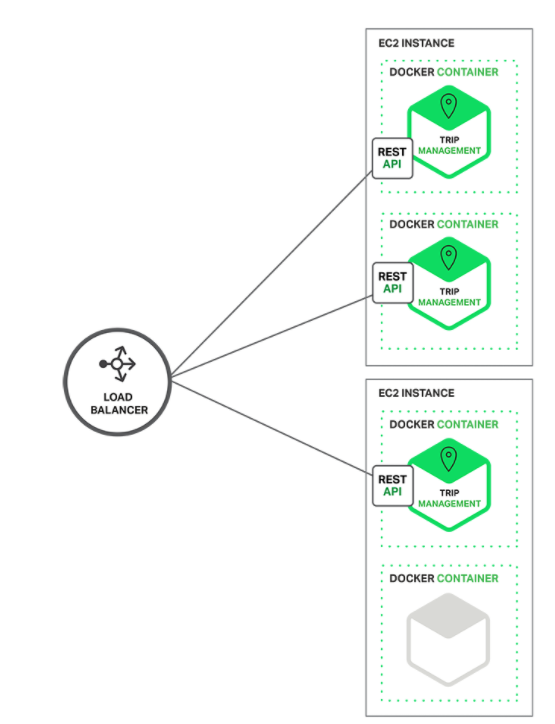
1. Microservices



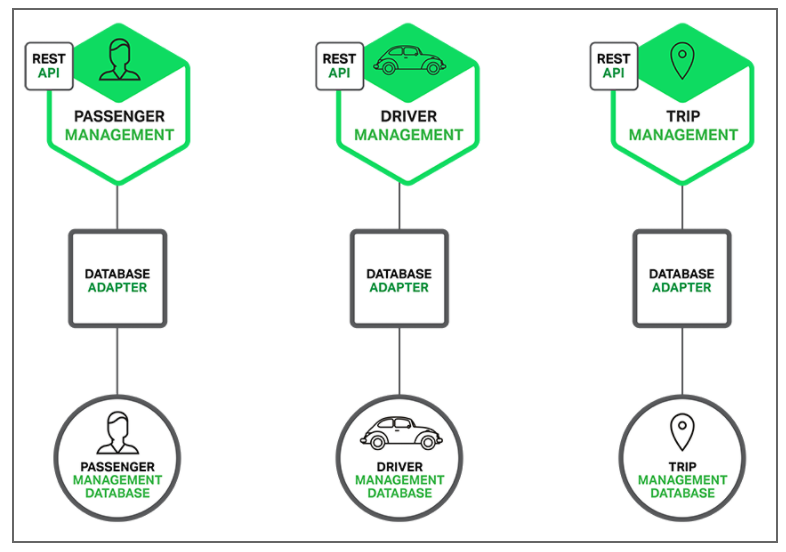
Mỗi service (driver management, passenger management, payments,…) đều được thực hiện bởi microservice của riêng nó. Hầu hết các service sử dụng các API do các service khác cung cấp

Vd : Driver management sử dụng máy chủ Notification để thông báo cho driver các chuyến đi, hoặc Passenger WEB UI sử dụng các service khác để render web pages (Passenger management, Driver Management)

Một vài REST API được hiển thị cho người dùng (driver , passenger) sử dụng, tuy nhiên các người dùng không thể trực tiếp truy cập vào các backend services mà thay vào đó nó phải thông qua API Gateway, API gateway chịu trách nhiệm cân bằng tải, truyền request đến dúng Service để thực hiện tác vụ, cache, access control,… .



Đặc biệt, mô hình Microservices có ảnh hưởng rất nhiều đến mối quan hệ giữa ứng dụng và Database. Đối với Monolithic Architecture,toàn bộ service sẽ dùng chung một database schema, tuy nhiên đối với Microservices mỗi service sẽ có một database schema riêng (mục đích chính của việc tách riêng schema là để các dịch vụ không phải phụ thuộc với nhau, mỗi dịch vụ nhỏ có thể tự chạy riêng mà không cần đến cơ sở dữ liệu của dịch vụ khác)



Nhờ sự tách biệt của database mà môi dịch vụ có thể sử dụng một loại database phù hợp nhất đối với nhu cầu.

Vd: Driver Management tìm tài xế gần nhất cho khách hàng, phải sử dụng các database hỗ trợ được tính năng truy vấn địa lý hiệu quả

// khác biệc với SOA

On the surface, the Microservices Architecture pattern is similar to SOA. With both approaches, the architecture consists of a set of services. However, one way to think about the Microservices Architecture pattern is that it’s SOA without the commercialization and perceived baggage of web service specifications (WS‑\*) and an Enterprise Service Bus (ESB). Microservice‑based applications favor simpler, lightweight protocols such as REST, rather than WS‑\*. They also very much avoid using ESBs and instead implement ESB‑like functionality in the microservices themselves. The Microservices Architecture pattern also rejects other parts of SOA, such as the concept of a canonical schema.

//

-Lợi ích của mô hình MicroServices mang lại:

Giải quyết vấn đề phức tạp của các ứng dụng lơn có nhiều dịch vụ, nó chia thành các ứng dụng nhỏ hơn tách biệt để dễ dàng quản lý và chỉnh sửa

Các Developer có thể chọn các framework tùy ý mà không còn bị hạn chế như Monolithic Architecture

Cho phép các Service triển khai độc lập trên các máy chủ khác nhau

Giúp các Service mở rộng một các độc lập, không phụ thuộc lẫn nhau

Tham khảo : https://www.nginx.com/blog/introduction-to-microservices/