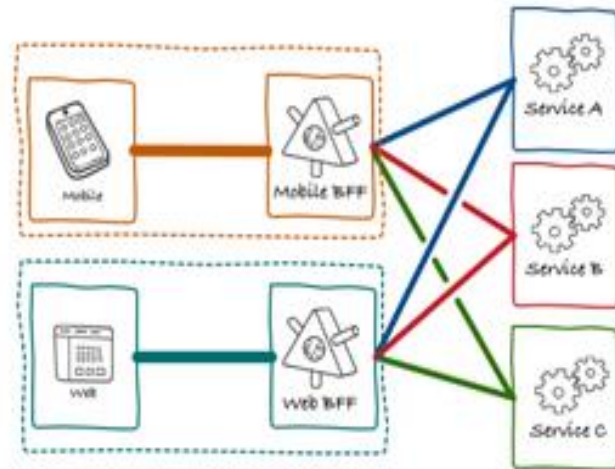


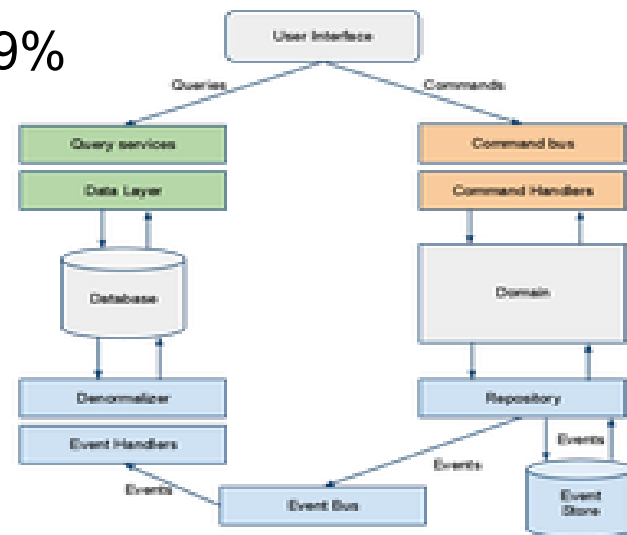
# Kiến trúc



# I. Yêu cầu kiến trúc

Những yêu cầu đòi hỏi kiến trúc của hệ thống phải đáp ứng được

1. Đảm bảo dữ liệu an toàn 99.9999999%
2. Tính sẵn sàng, chịu tải tốt 99.99%
3. Nhất quán
4. Có khả năng mở rộng)





## 1. Durability

- Đây là yếu tố đầu tiên và cũng là quan trọng nhất của hệ thống
- Đảm bảo rằng không bao giờ xảy ra chuyện mất dữ liệu được đưa lên hệ thống

## 2. Availability

- Là tính sẵn sàng, 99.99% các request phải được trả về
- Những hệ thống chịu tải thấp không thể đáp ứng được → kiến trúc hợp lý
- Kiến trúc có cơ chế tốt, để có thể chuyển sever ngay



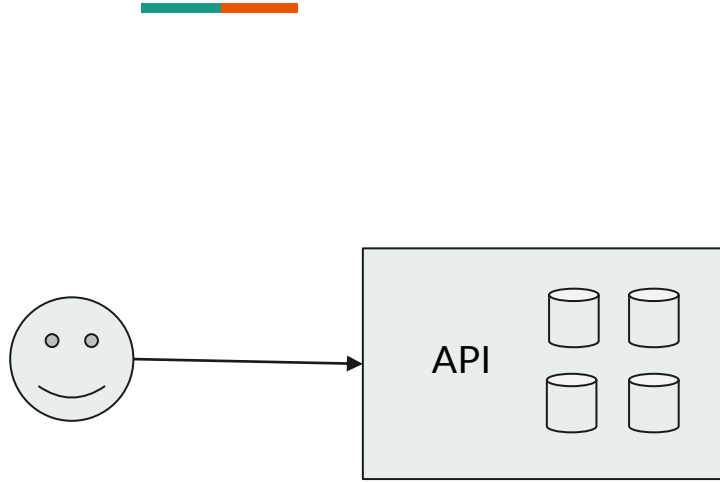
### 3. Multi-tenancy

- Tính nhất quán, điều này có nghĩa là: khi một người sử dụng dịch vụ họ không quan tâm bên dưới như thế nào (1 sever hay nhiều , dữ liệu ở đâu,...)
- Chỉ cần vào một địa chỉ duy nhất.

### 4. Scalable (Có thể mở rộng)

- Kiến trúc của bạn phải sẵn sàng cho việc mở rộng

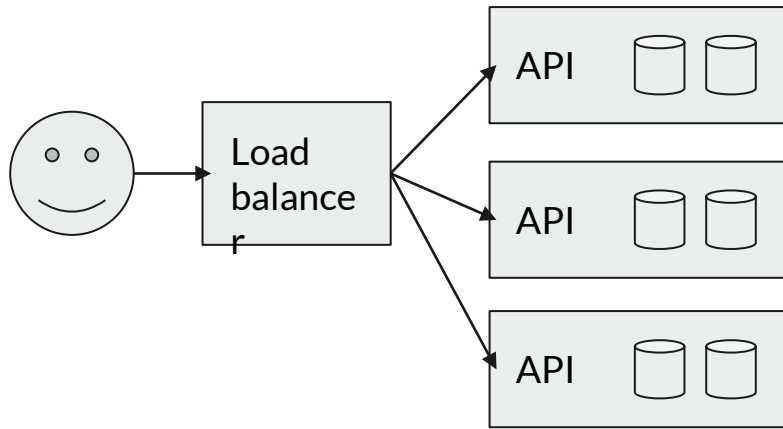
## II. Kiến trúc High level



- Đây là kiến trúc đơn giản nhất mà ta có
- Được cài đặt toàn bộ trên 1 server
- Lập trình theo mô hình mvc, 3 lớp,...

→ Mô hình không khả thi nếu có số lượng người dùng vượt quá khả năng chịu tải của server.

→ Chỉ phù hợp với những website nhỏ, có lượng truy cập thấp.



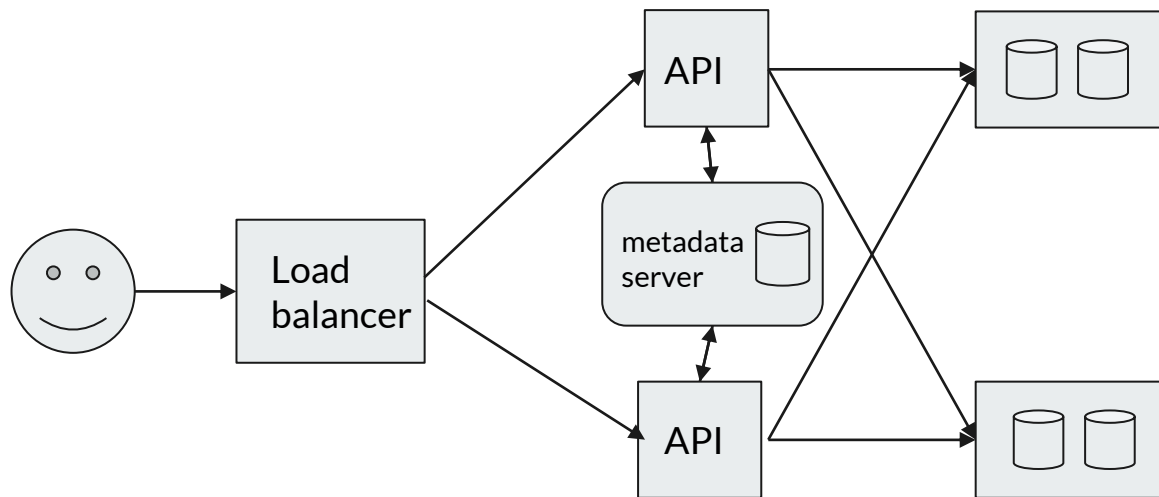
- Nâng cấp hơn kiến trúc đầu tiên. Ta tăng số lượng server lên và dùng một con server cân bằng tải (Load Balance) để phân chia truy cập từ người dùng đến các máy chủ khác nhau.




→ Đã có khả năng chịu tải , tuy nhiên những vấn đề khác phát sinh

- ví dụ bạn vào lập tài khoản tại máy chủ A, nhưng hôm sau được phân vào máy chủ B → không thấy tài khoản
- mô hình này chỉ phù hợp cho những website có lượng truy cập lớn, nhưng không có nhiều tính năng (phimmoi, vnexpress,...)

→ Tách server thành 2 phần riêng là API và Storage





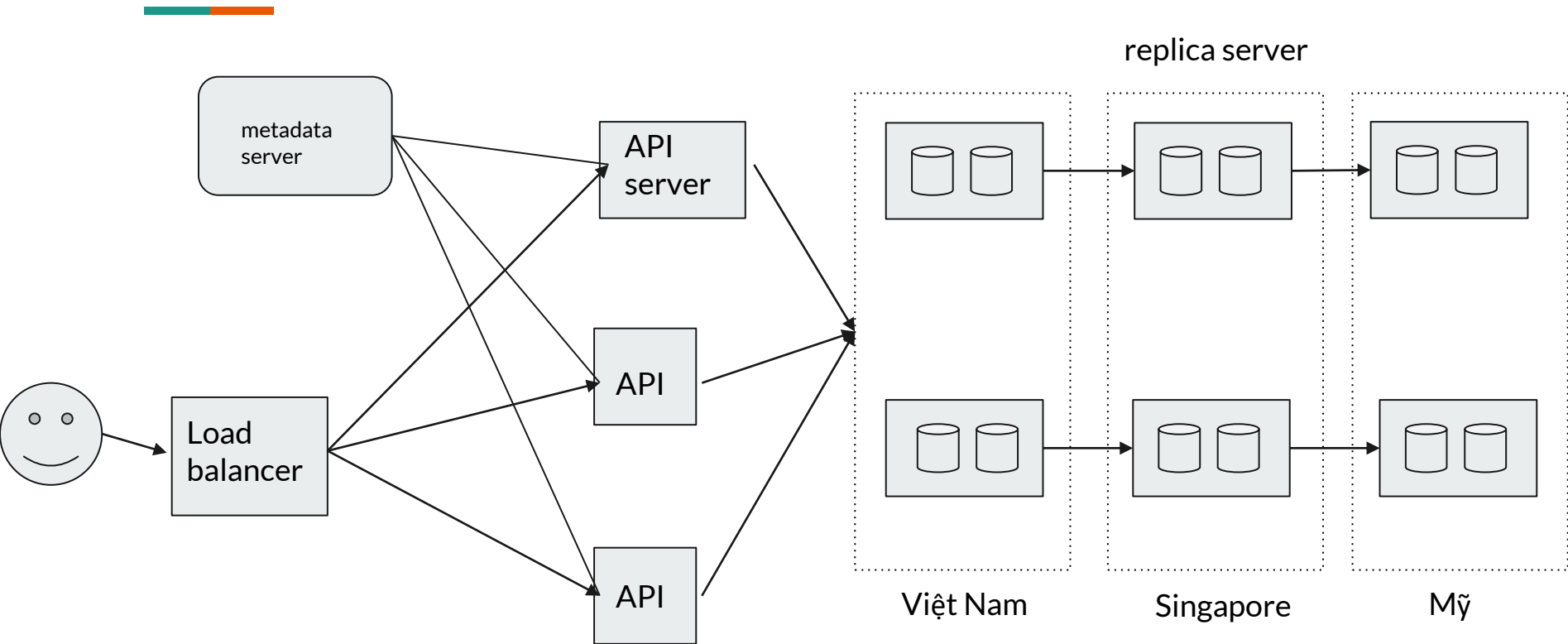
- 
- Metadata server: Đây là 1 microservice
  - Nó một cơ sở dữ liệu lưu trữ lại file nào nằm ở server nào
  - Một request diễn ra như thế nào: từ api server sẽ gửi một request đến metadata, meta trả về kết quả là server dữ liệu nào.


→ trả lời được các yêu cầu 2,3,4

- Hệ thống có khả năng chịu tải lớn
- Multi-tenancy
- Có khả năng mở rộng

→ Nhưng server database chết sẽ mất dữ liệu

## Tạo ra những con server replica ở các địa điểm khác



- 
- Tạo ra những con server replica trên các địa điểm khác nhau.
  - Những server này được đồng bộ từ server chính, theo một cơ chế nhất định
  - Nếu như request đến database không được phản hồi, API sẽ hỏi lại metadata, metadata sẽ cho biết database thay thế

### III. Công nghệ



#### 1. Load Balancer (cân bằng tải)

- Ứng dụng tương tự : Nginx Plus
- Ngôn ngữ lập trình: nodejs
- + thư viện : http-proxy
- Thuật toán cân bằng: Least connection, với thuật toán này LB sẽ điều hướng request đến server đang có ít request nhất, nhờ đó mà bảo đảm các server không bị mất cân bằng và phục vụ số request bằng nhau.

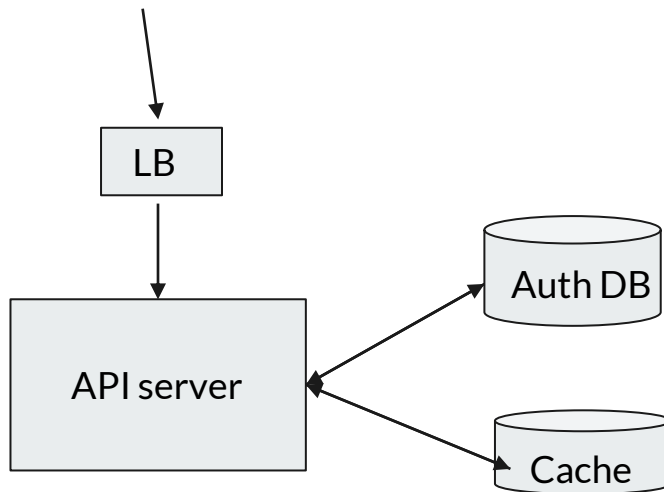
## 2. API server



Ngôn ngữ lập trình nodejs

Auth DB: postgresql

restAPI



### 3. metadata server



Dùng nodejs

Database: mongo db

```
Bucket
{
  _id: "88fq4381jf48345491"
  user : nguyenbao
  data: {
    folder1: {
      name : file1.txt
      whewe : .....//block storage
    }
  }
}
```

