BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT

BÁO CÁO SEMINAR

TÊN ĐỀ TÀI:

TRIỂN KHAI ỨNG DỤNG THEO MÔ HÌNH MICROSERVICE

Người thực hiện: TS. Nguyễn Thế Lộc (Mã cán bộ: 0801-02)

Đơn vị: Bộ môn Công nghệ phần mềm
Khoa Công nghệ Thông tin

Hà Nội - 2022

MỤC LỤC

DANH MỤC HÌNH VẼ

MỞ ĐẦU

Tích hợp hệ thống là một hoạt động giữ vai trò quan trọng trong quy trình phát triển phần mềm và là một trong các hoạt động cuối cùng không thể thiếu trong các dự án sản xuất hoặc gia công phần mềm. Chính vì thế mà tích hợp hệ thống đã trở thành công đoạn quyết định sự thành công của dự án.

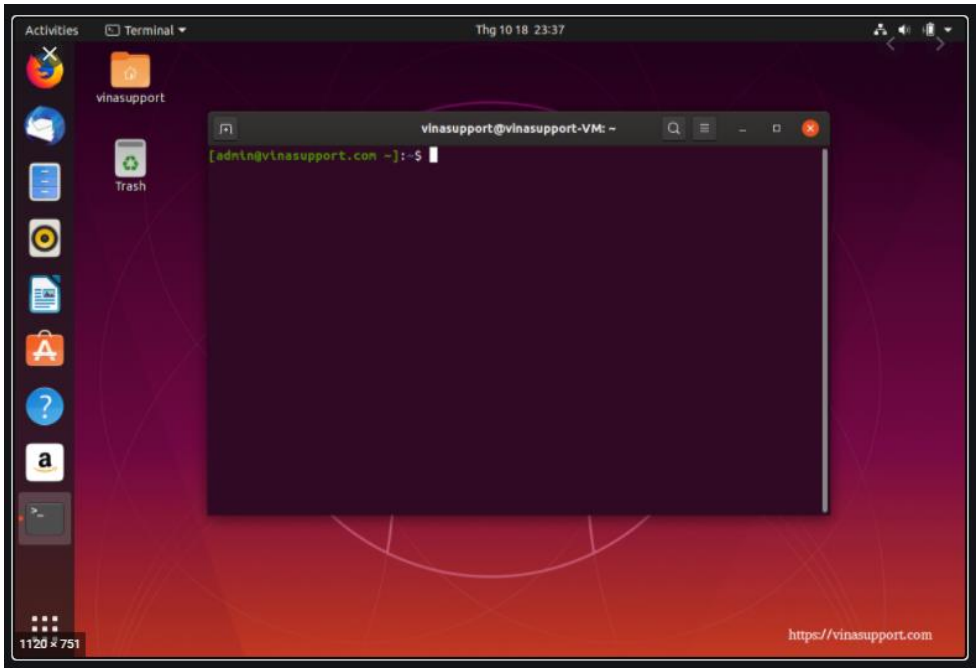
Chất lượng của hệ thống hỗ trợ ra quyết định được đánh giá qua một mô hình chất lượng cụ thể. Hệ hỗ trợ ra quyết định được phân tách theo cấp bậc vào một phần mềm với những tiêu chí chính và những tiêu chí con, sao cho có thể sử dụng chúng như một danh sách để kiểm tra những vấn đề phát sinh liên quan đến chất lượng. Trong nội dung này, nhóm nghiên cứu sẽ tập trung trình bày về vấn đề tích hợp và triển khai hệ thống. Đồng thời cũng cần phải kiểm tra toàn bộ hệ thống, với các bộ dữ liệu khác nhau để đánh giá một cách tổng thể từ đó có những cập nhật, chỉnh sửa và bổ sung phù hợp giúp cho hệ thống vận hành một cách tối ưu, đáp ứng được yêu cầu người dùng.

CHƯƠNG 1. TỔNG QUAN VỀ CÔNG NGHỆ TRIỂN KHAI

Khi xây dựng một hệ thống yêu cầu cần phải có hiệu năng cao, tốc độ nhanh và quan trọng là ổn định, hoạt động liên tục thì sẽ cần những công nghệ phù hợp nhất (về tài nguyên, hạ tầng, nhân sự). Những công nghệ được đề xuất sử dụng là Docker, Traefik được triển khai theo mô hình Microservice trên hệ điều hành Ubuntu. Những công nghệ này khi được kết hợp với nhau sẽ tạo ra một hệ thống có khả năng chịu tải tốt nhằm phục vụ được số lượng người dùng lớn thông qua cơ chế cân bằng tải (Load Balancing) và dễ dàng mở rộng.

1.1 Ubuntu

Ubuntu (phát âm IPA uːˈbuːntuː) là một hệ điều hành máy tính dựa trên Debian GNU/Linux, một bản phân phối Linux thông dụng. Tên của nó bắt nguồn từ "ubuntu" trong tiếng Zulu, có nghĩa là "tình người", mô tả triết lý ubuntu: "Tôi được là chính mình nhờ có những người xung quanh," một khía cạnh tích cực của cộng đồng.



Hình 1.1. Giao diện màn hình Ubuntu Desktop

Ubuntu được tài trợ bởi công ty Canonical Ltd (chủ sở hữu là một người Nam Phi Mark Shuttleworth). Thay vì bán Ubuntu, Canonical tạo ra doanh thu bằng cách bán hỗ trợ kĩ thuật.

Canonical phát triển Ubuntu thành 2 dòng sản phẩm chính:

Ubuntu Desktop: cài đặt cho các máy tính cá nhân và phục vụ những người dùng thông thường.

Ubuntu Server: cài đặt cho các máy chủ để phục vụ các dịch vụ trên internet cũng như mạng doanh nghiệp.

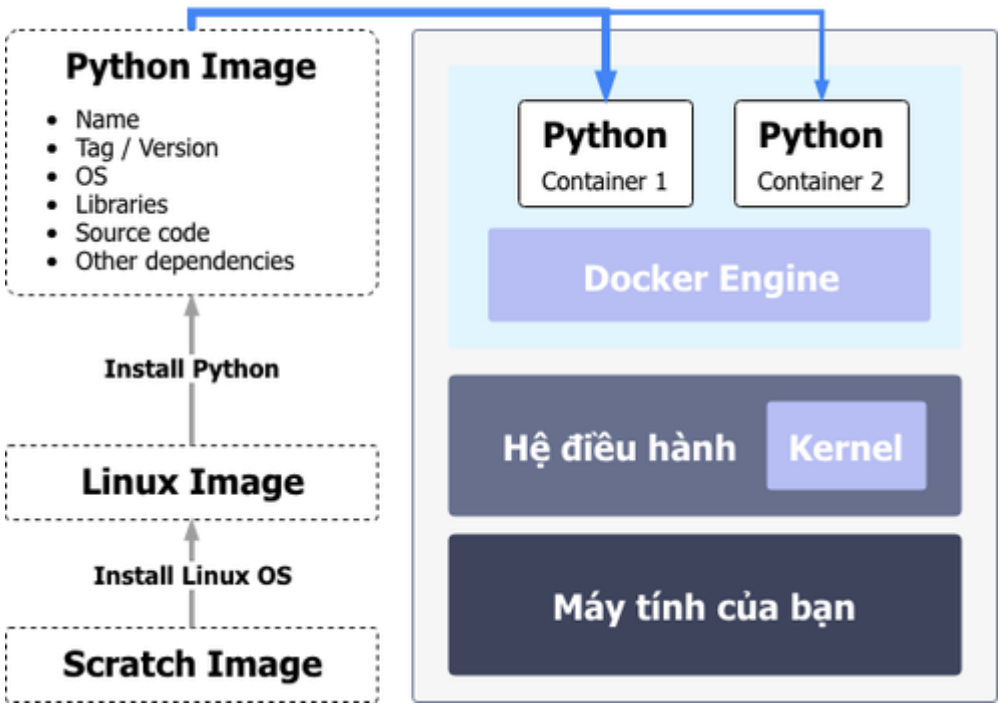
1.2 Docker

Docker là một open platform cung cấp cho người sử dụng những công cụ và service để người sử dụng có thể đóng gói và chạy chương trình của mình trên các môi trường khác nhau một cách nhanh nhất. Docker là một một phương thức để đóng gói và sắp xếp phần mềm giúp các nhà phát triển phần mềm tự động triển khai các ứng dụng Linux và Windows vào trong các container ảo hóa.

Docker khiến cho việc dựng và chạy các kiến trúc vi dịch vụ được phân phối, triển khai mã của bạn với quy trình tích hợp và phân phối liên tục được tiêu chuẩn hóa, xây dựng các hệ thống xử lý dữ liệu có quy mô cực kỳ linh hoạt cũng như tạo ra các nền tảng được quản lý đầy đủ dễ dàng hơn cho các nhà phát triển.

Container là một thuật ngữ để chỉ những Process được quản lý bởi một Container Enginer (Ngoài Docker Engine ra thì còn một số công cụ khác như Solaris Zones, LXC, etc). Với Docker Engine (Hay còn gọi tắt là Docker), chúng ta có thể tạo ra và thực thi những Container. Lợi ích của việc “Container hoá” (Containerized) các Process này sẽ giúp chúng ta quản lý sử dụng nguồn tài nguyên máy và tách biệt (isolating) môi trường giữa các Process.

Một khái niệm khác trong Docker: Image – Những mẫu (template) chứa những dòng quy định một Container chạy lên như thế nào. Nếu hiểu theo lập trình hướng đối tượng thì ta có thể xem Image như một Class và những Containers được tạo từ Image này như một Object Instance của Class đó..



Hình 1.2. Minh họa Docker Image

Ở mô hình trên, ta có thể thấy container 1 và 2 được khởi tạo từ Image có tên là Python. Một Docker Image thường được build từ một Base Image khác (Thường sẽ bắt nguồn từ các phân nhánh của Linux như Ubuntu, Alpine, Slim, etc). Những Base Image này cũng có Base Image cho chính nó và gốc cuối cùng là Scratch Image.

Ngoài Docker Container và Docker Image, chúng ta cần biết thêm về:

- Dockerfile: Là một tập tin chứa những mô tả để build một Docker Image
- Docker Registry: Là một kho chứa các Docker Image, về cơ bản chúng ta sẽ có sẵn 2 Docker Registry sẵn sàng để sử dụng:
- Local Registry: Những Image được kéo về máy chủ hay được build tại máy chủ sẽ nằm tại đây. Cũng giống như source code khi bạn clone về máy từ GitHub.
- Docker Hub: Đây là Public Registry và là Registry mặc định của Docker. Đa số các Image sẽ đuộc pull (kéo về) từ đây.

- Volume / Binding Volume: Khi một Container được khởi tạo, nó sẽ có một Volume (Ổ chứa dữ liệu) tách biệt để chứa các tập tin hệ thống hay source code, nếu Container đó ngừng chạy, chúng ta sẽ không còn access được những tập tin này nữa. Do đó, việc binding giữa Docker Volume và một ổ chứa của máy chủ (Host machine) là cần thiết nếu chúng ta muốn lưu giữ những tập tin sinh ra từ Container (ví dụ như log files, configuration files, etc)

- Container Port / Binding Ports: Tương tự như Volume, Container cũng có riêng những Network ports, và để access những Port này từ máy chủ, chúng ta cũng phải bind các port trên máy chủ với Containers.

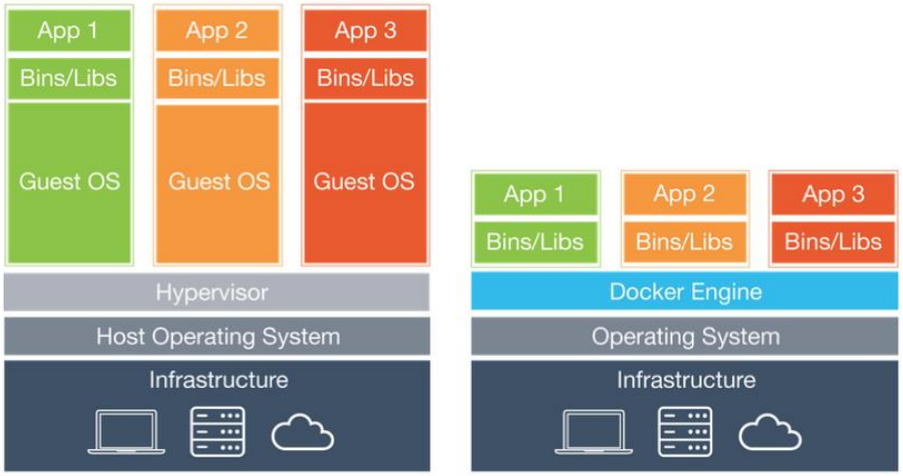
- Docker Client: công cụ chính để chúng ta giao tiếp với Docker Engine.

- Docker Daemon: đóng vai trò như Listener để thực thi các lệnh quản lý Docker như build Image, run Container, get Image list và get Container list, etc.

Sự khác biệt giữa Docker và Hypervisors:

Hypervisor là ảo hỏa nằm ở tầng Hardware (phần cứng), tức là mô phỏng phần cứng và chạy những OS con trên phần cứng đó (Các công cụ Hypervisor như Virtual Box, VMware..). Virtual Machine (Hypervisors): Cần thêm một Guest OS cho nên sẽ tốn tài nguyên hơn và làm chậm máy thật khi sử dụng. Thời gian khởi động trung bình là 20s có thể lên đến hàng phút, thay đổi phụ thuộc vào tốc độ của ổ đĩa.

Docker: Dùng chung kernel, chạy độc lập trên Host Operating System và có thể chạy trên bất kì hệ điều hành nào cũng như cloud. Khởi động và làm cho ứng dụng sẵn sàng chạy trong 500ms, mang lại tính khả thi cao cho những dự án cần sự mở rộng nhanh.

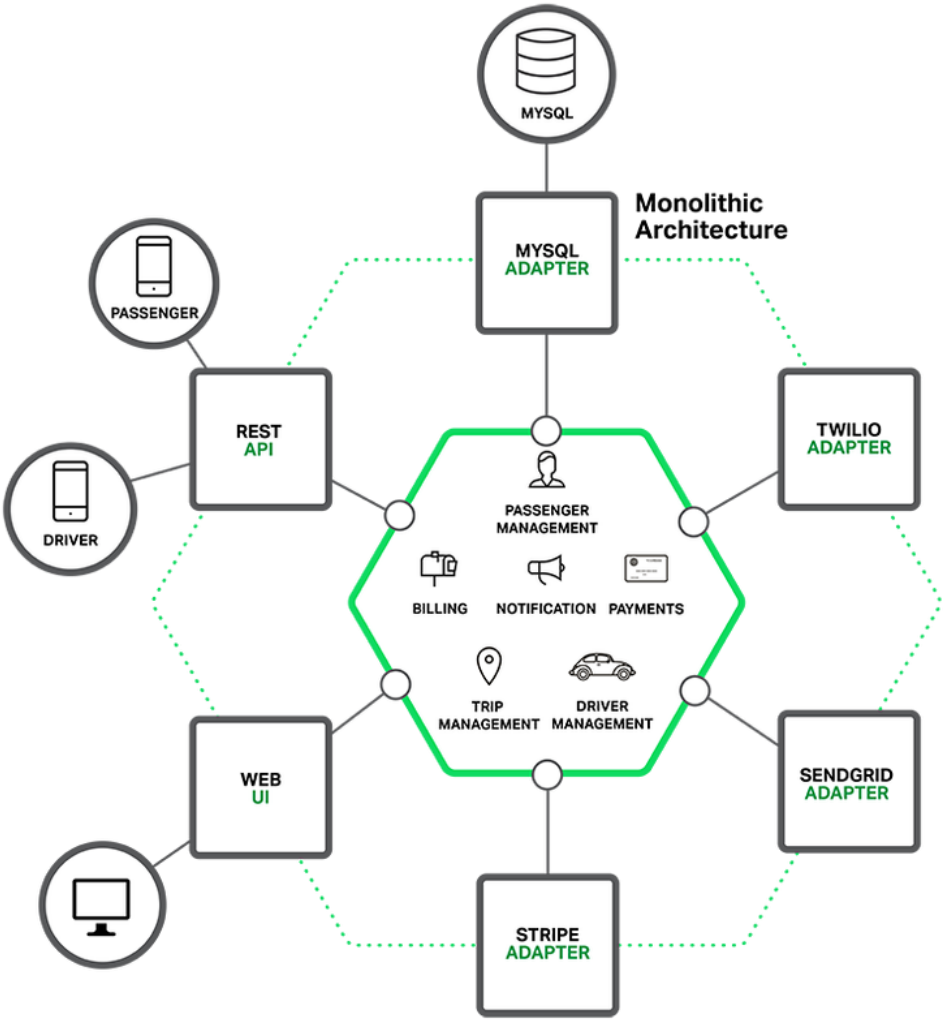


Hình 1.3. Kiến trúc của Docker và Hypervisors

1.3 Microservice

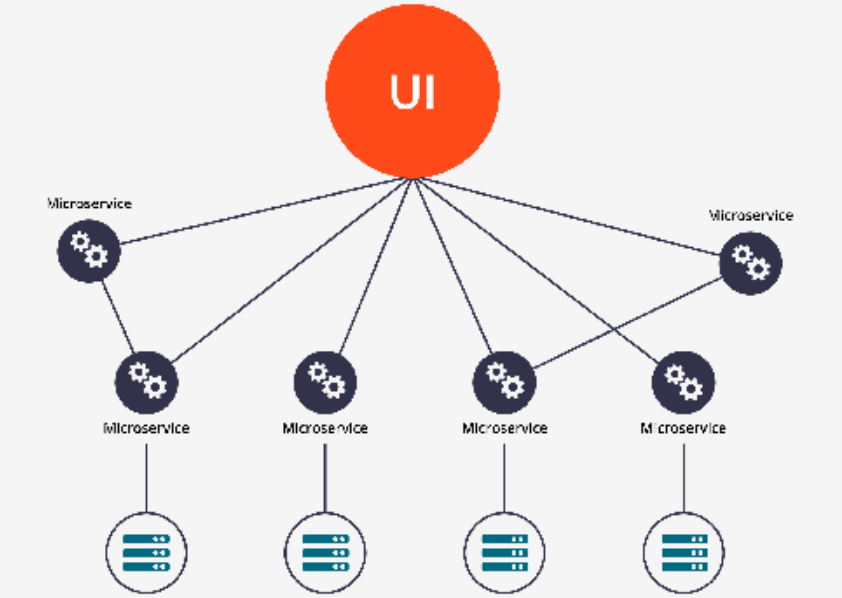
Trong tiếng anh, micro có nghĩa là nhỏ, vi mô. Vậy Microservice, như tên của nó, đó chính là chia một khối phần mềm thành các service nhỏ hơn, có thể triển khai trên các server khác nhau. Mỗi service sẽ xử lý từng phần công việc và được kết nối với nhau thông qua các các giao thức khác nhau, như http, SOA, socket, Message queue (Active MQ, Kafka)... để truyền tải dữ liệu.

Trước khi Microservices xuất hiện, các ứng dụng thường phát triển theo mô hình Monolithic architecture (Kiến trúc một khối). Có nghĩa là tất cả các module (view, business, database) đều được gộp trong một project, một ứng dụng được phát triển theo mô hình kiến trúc một khối thường được phân chia làm nhiều module. Nhưng khi được đóng gói và cài đặt sẽ thành một khối (monolithic). Lợi ích của mô hình kiến trúc một khối đó là dễ dàng phát triển và triển khai. Nhưng bên cạnh đó nó cũng có nhiều hạn chế ví dụ như khó khăn trong việc bảo trì, tính linh hoạt và khả năng mở rộng kém, đặc biệt với những ứng dụng doanh nghiệp có quy mô lớn. Đó chính là lí do ra đời của kiến trúc Microservices.



Microservice nó là một kiểu kiến trúc hệ thống để dựng lên application của chúng ta. Tại đó, application được định nghĩa dưới dạng một tập hợp các services. Mỗi service sẽ đảm nhiệm một phần chức năng trong hệ thống.

Hiểu theo một cách đơn giản thì microsevice là việc chia nhỏ ứng dụng lớn thành các ứng dụng nhỏ, mỗi ứng dụng đảm nhiệm một chức năng riêng biệt và được kết nối với nhau một cách hài hòa để đáp ứng yêu cầu nghiệp vụ của sản phẩm.

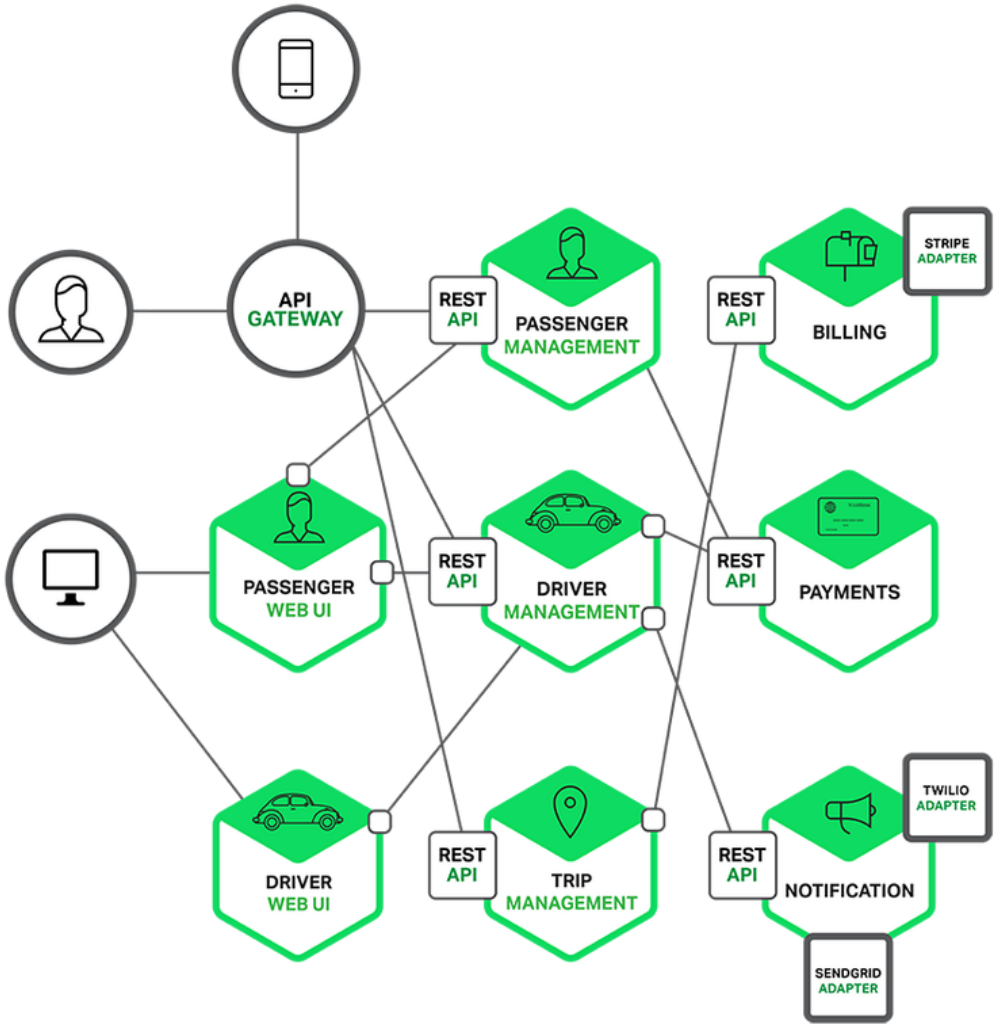


Hình 1.4. Mô hình kiến trúc Microservice

Microservice cho phép việc phát triển một ứng dụng lớn qua nhiều giai đoạn và nhiều nhóm có thể làm việc độc lập với nhau về ngôn ngữ, con người, tiêu chuẩn kết nối. Điều này là cần thiết với việc thị trường đang thay đổi từng ngày một cách nhanh chóng và sản phẩm phải liên tục được sửa đổi và nâng cấp để đáp ứng. Trong khi kiến trúc nguyên khối qua một thời gian cập nhật và sửa đổi bắt đầu để lộ ra những khó khăn về thời gian, con người, … vì việc nâng cấp đòi hỏi người phát triển phải hiểu toàn bộ hệ thống hiện tại, thì kiến trúc microservices được ra đời để giải quyết những khó khăn này. Với microservices thì việc nâng cấp và phát triển thêm tính năng cho sản phẩm đang sử dụng chỉ cần phát triển modun mới cho tính năng cần cập nhật và giao tiếp với các ứng dụng lõi khác đã có thông qua các interface được thiết kế bởi các dịch vụ đã tồn tại trước đó. Cách những dịch vụ nhỏ kết nối với nhau như sau:

- Cơ chế kết nối:
  - Xử lý đồng bộ (Synchronous): Gửi yêu cầu và chờ đợi trả lời rồi mới tiếp tục xử lý.
  - Bất đồng bộ (Asynchronous): Gửi yêu cầu và tiếp tục xử lý ngay, khi có câu trả lời sẽ được thông báo để xử lý kết quả.
- Về chuẩn giao tiếp:

- HTTP (RESTFUL API, SOAP)
- RPC (remote procedure call -lệnh gọi từ xa)
- MQ (message queue)



Những đặc điểm của microservice:

- Decoupling - Các service trong một hệ thống phần lớn được tách rời. Vì vậy, toàn bộ ứng dụng có thể dễ dàng được xây dựng, thay đổi và thu nhỏ.
- Componentization - Microservices được coi là các thành phần độc lập có thể dễ dàng thay thế và nâng cấp.
- Business Capabilities - mỗi một thành phần trong kiến trúc microservice rất đơn giản và tập trung vào một nhiệm vụ duy nhất.

- Autonomy - các lập trình viên hay các nhóm có thể làm việc độc lập với nhau trong quá trình phát triển.
- Continous Delivery - Cho phép phát hành phần mềm thường xuyên, liên tục.
- Responsibility .
- Decentralized Governance - không có mẫu chuẩn hóa hoặc bất kỳ mẫu công nghệ nào. Được tự do lựa chọn các công cụ hữu ích tốt nhất để có thể giải quyết vấn đề.
- Agility - microservice hỗ trợ phát triển theo mô hình Agile.

Ưu điểm:

Kiến trúc Microservices được sinh ra để khắc phục những hạn chế của kiến trúc một khối:

- Independent Development - Tất cả các service có thể được phát triển dễ dàng dựa trên chức năng cá nhân của từng service. Có thể chia nhỏ để phát triển độc lập.
- Independent Deployment - Có thể được triển khai riêng lẻ trong bất kỳ ứng dụng nào.
- Fault Isolation - khi một service của ứng dụng không hoạt động, hệ thống vẫn tiếp tục hoạt động.
- Mixed Technology Stack - Các ngôn ngữ và công nghệ khác nhau có thể được sử dụng để xây dựng các service khác nhau của cùng một ứng dụng.
- Granular Scaling

Kiến trúc Microservices giúp đơn giản hóa hệ thống, chia nhỏ hệ thống ra làm nhiều service nhỏ lẽ dễ dàng quản lý và triển khai từng phần so với kiến trúc nguyên khối. Phân tách rõ ràng giữa các service nhỏ. Cho phép việc mỗi service

được phát triển độc lập. Cũng như cho phép lập trình viên có thể tự do chọn lựa technology stack cho mỗi service mình phát triển. mỗi service có thể được triển khai một cách độc lập (VD: Mỗi service có thể được đóng gói vào một docker container độc lập, giúp giảm tối đa thời gian deploy). Nó cũng cho phép mỗi service có thể được scale một cách độc lập với nhau. Việc scale có thể được thực hiện dễ dàng bằng cách tăng số instance cho mỗi service rồi phân tải bằng load balancer.

Nhược điểm:

Kiến trúc Microservices đang là một xu hướng, nhưng nó cũng có nhược điểm của nó. Microservice khuyến khích làm nhỏ gọn các service, nhưng khi chia nhỏ sẽ dẫn đến những thứ vụn vặt, khó kiểm soát. Hơn nữa chính từ đặc tính phân tán khiến cho các lập trình viên phải lựa chọn cách thức giao tiếp phù hợp khi xử lí request giữa các service.

Một nhược điểm lớn khác của microservices là sự phức tạp phát sinh từ thực tế là một ứng dụng microservices là một hệ thống phân tán. Các developer cần phải lựa chọn và thực hiện một cơ chế giao tiếp giữa các process dựa trên messaging hoặc RPC. Hơn nữa, họ cũng phải viết mã để xử lý việc thất bại giữa chừng (partial failure) vì điểm đến của request có thể chậm hoặc không khả dụng. Rõ ràng là nó phức tạp hơn nhiều so với trong một ứng dụng nguyên khối nơi các mô-đun gọi nhau thông qua các cuộc gọi phương thức / thủ tục mức ngôn ngữ.

Một thách thức khác với microservices là kiến trúc cơ sở dữ liệu phân vùng (partitioned database architecture). Các business transactions cập nhật nhiều business entities là điều khá phổ biến. Các loại transaction này thường được thực hiện khá dễ dàng trong một ứng dụng nguyên khối vì chỉ có một cơ sở dữ liệu duy nhất. Tuy nhiên, trong một ứng dụng dựa trên microservices, bạn cần cập nhật nhiều cơ sở dữ liệu thuộc sở hữu của các dịch vụ khác nhau. Sử dụng các distributed transactions thường không phải là một lựa chọn, và không chỉ vì định lý CAP. Đơn giản là sự toàn vẹn về cơ sở dữ liệu không được hỗ trợ bởi các dạng cơ sở dữ liệu như NoSQL (có khả năng mở rộng cao) và các messaging brokers.

Chúng ta cuối cùng phải sử dụng một phương pháp tiếp cận dựa trên sự nhất quán cuối cùng (eventual consistency), điều này cũng gây không tí khó khăn cho các developer.

Test một ứng dụng microservices cũng phức tạp hơn nhiều. Ví dụ, với một framework hiện đại như Spring Boot, rất đơn giản để viết một lớp thử nghiệm khởi động một ứng dụng web nguyên khối và kiểm tra API REST của nó. Ngược lại, một lớp thử nghiệm tương tự cho một dịch vụ sẽ cần phải khởi chạy dịch vụ đó và bất kỳ dịch vụ nào mà nó phụ thuộc (hoặc ít nhất là cấu hình các stubs cho các dịch vụ đó). Một lần nữa là không đánh giá thấp sự phức tạp của việc này.

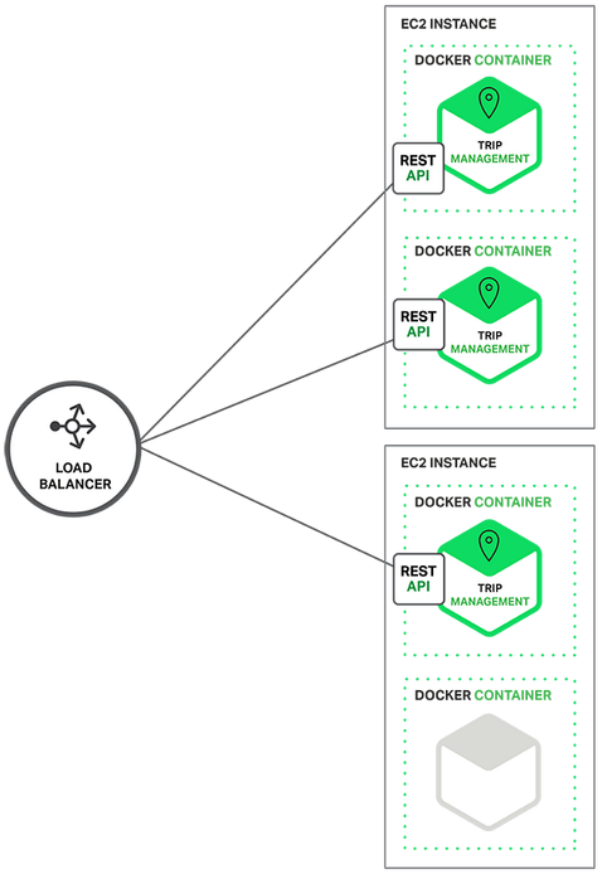
Một thách thức lớn khác với mô hình Microservices là thực hiện các thay đổi trải rộng trên nhiều dịch vụ. Ví dụ, hãy tưởng tượng rằng bạn đang thực hiện câu chuyện yêu cầu thay đổi đối với dịch vụ A, B và C, trong đó A phụ thuộc vào B và B phụ thuộc vào C. Trong ứng dụng nguyên khối, bạn có thể thay đổi mô-đun tương ứng, tích hợp các thay đổi, và triển khai chúng trong một lần. Ngược lại, trong Microservices, bạn cần phải lập kế hoạch và điều phối cẩn thận việc triển khai các thay đổi cho từng dịch vụ. Ví dụ, bạn sẽ cần cập nhật dịch vụ C, tiếp theo là dịch vụ B, và cuối cùng là dịch vụ A. May thay, hầu hết các thay đổi thường chỉ tác động đến một dịch vụ và các thay đổi dịch vụ đa yêu cầu phối hợp tương đối hiếm.

Triển khai một ứng dụng dựa trên microservices cũng phức tạp hơn nhiều. Một ứng dụng nguyên khối được triển khai đơn giản trên một tập hợp các máy chủ giống hệt nhau phía sau bộ cân bằng tải truyền thống. Mỗi cá thể ứng dụng được cấu hình với các vị trí (host and ports) của các dịch vụ cơ sở hạ tầng như cơ sở dữ liệu và message broker. Ngược lại, một ứng dụng microservice thường bao gồm một số lượng lớn các dịch vụ. Ví dụ, Hailo có 160 dịch vụ khác nhau và Netflix có hơn 600 dịch vụ. Mỗi dịch vụ sẽ có nhiều phiên bản running. Có nhiều bộ phận cần được cấu hình, triển khai, thu nhỏ và giám sát. Ngoài ra, bạn cũng sẽ cần triển khai một cơ chế khám phá dịch vụ (service discovery) (được thảo luận trong bài sau) cho phép một dịch vụ khám phá các vị trí (host and ports) của bất kỳ dịch vụ nào khác mà nó cần giao tiếp. Các cách tiếp cận thủ công truyền thống dựa trên

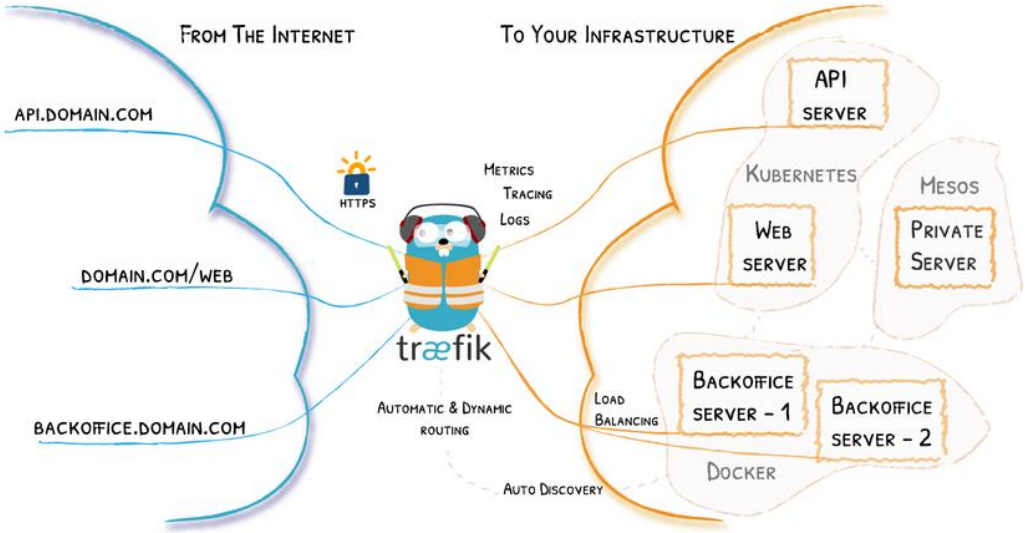
ticket-based và thủ công không thể mở rộng đến mức độ phức tạp này. Do đó, việc triển khai thành công ứng dụng microservices yêu cầu các developer kiểm soát các phương thức triển khai và mức độ tự động hóa cao hơn.

1.4 Traefik

Traefik là một Reverse-proxy đời mới, và cũng là load balancer để làm cho việc deploy hệ thống microservice được trở lên dễ dàng hơn. Tích hợp rất nhiều các thành phần infrastructure như Docker, Swarm mode, Kubernetes, Marathon, Consul, Etcd, Rancher, Amazon ECS... Và tính tự động là điểm quan trọng nhất trong các config với traefik.



Triển khai mô hình Microservice kết hợp với Docker đem lại nhiều lợi ích. Để người dùng truy cập vào hệ thống microservice thì ta cần một reverse proxy. Và reverse proxy chính là một trong số các vấn đề mà chúng ta gặp phải trong kiến trúc này. Traefik sinh ra để giải quyết việc đó.



Hình 1.5. Mô hình hoạt động của Traefik

1.5 Đăng nhập một lần - Single sign-on (SSO)

Đăng nhập một lần (SSO) là dịch vụ xác thực phiên và người dùng cho phép người dùng cuối nhập một bộ thông tin đăng nhập (có thể gồm tên và mật khẩu) để có quyền truy cập vào nhiều ứng dụng.

Hình 1.6. Mô hình hoạt động của SSO

OAuth, được phát âm là "oh-auth", là nền tảng framework cho phép các dịch vụ bên thứ ba sử dụng thông tin tài khoản của người dùng cuối, chẳng hạn như

Facebook, mà không để lộ mật khẩu của người dùng. OAuth hoạt động như một trung gian đại diện cho người dùng cuối thông qua một mã token truy cập, mã này sẽ cho phép thông tin tài khoản cụ thể được chia sẻ. Khi người dùng cố gắng truy cập một ứng dụng từ nhà cung cấp dịch vụ, nhà cung cấp dịch vụ sẽ gửi yêu cầu đến bên nhận dạng thứ 3 để xác thực. Nhà cung cấp dịch vụ sau đó sẽ xác minh xác thực và cho phép người dùng đăng nhập.

CHƯƠNG 2. CÁC BƯỚC TÍCH HỢP VÀ TRIỂN KHAI

Chương này sẽ trình bày các bước tích hợp và triển khai một hệ thống học tập trực tuyến dựa trên mã nguồn mở Moodle theo mô hình microservice.

2.1 Chuẩn bị

- Máy chủ cấu hình cao, dung lượng lớn (CPU 8 cores, RAM 8 GB, HDD 300 GB);

- Hệ điều hành Ubuntu server phiên bản được cài đặt sẵn bởi quản trị hệ thống;

- File kịch bản docker-compose.yml (để tự động cài các dịch vụ) được tạo sẵn bởi nhóm nghiên cứu;

- File cấu hình traefik.yml (để cấu hình reverse proxy), dynamic_conf.yml (để cấu hình cân bằng tải – load balancing) được tạo sẵn bởi nhóm nghiên cứu.

Nội dung file docker-compose.yml:

```
version: '2'
services:
  mariadb:
    image: 'mariadb:10'
    container_name: db
    environment:
      MYSQL_RANDOM_ROOT_PASSWORD: 1
      MYSQL_DATABASE: moodle
      MYSQL_USER: moodle
      MYSQL_PASSWORD: moodlePassword
    volumes:
      - $HOME/volumes/mysql:/var/lib/mysql

  moodle:
    image: 'bitnami/moodle:3'
    container_name: moodle
    environment:
      MOODLE_USERNAME: programster
      MOODLE_PASSWORD: thisIsMyMoodleLoginPassword
      MOODLE_EMAIL: admin@programster.org
      MARIADB_HOST: db
      MARIADB_PORT_NUMBER: 3306
      MOODLE_DATABASE_USER: moodle
      MOODLE_DATABASE_NAME: moodle
      MOODLE_DATABASE_PASSWORD: moodlePassword
      ALLOW_EMPTY_PASSWORD: "no"
```

```
    ports:
      - '80:80'
      - '443:443'
    volumes:
      - $HOME/volumes/moodle:/bitnami'
    depends_on:
      - mariadb

  traefik:
    image: traefik:v2.0
    restart: always
    volumes:
      - "./traefik.yml:/etc/traefik/traefik.yml"
      - "./conf:/conf"
      - "./secure-config:/secure-config"
    ports:
      - "80:80"
      - "443:443"
      - "8888:8888"

volumes:
  mariadb_data:
    driver: local
  moodle_data:
    driver: local
```

Nội dung file traefik.yml:

```
api:
  insecure: false
  dashboard: true
  debug: true
log:
  level: DEBUG #INFO
  format: json
entryPoints:
  web:
    address: ":80"
  web-secure:
    address: ":443"
  api:
    address: ":8888"
providers:
  file:
    directory: "/conf"
    filename: "dynamic_conf.yml"
    watch: true
```

Nội dung file dynamic_conf.yml:

```
http:
  routers:
    api:
      entryPoints:
```

```
      - "api"
    rule: PathPrefix(`/`)
    service: api@internal
  moodle-secure:
    entryPoints:
      - "web-secure"
    rule: "Host(`elearning.humg.edu.vn`)"
    service: moodle-secure
    tls: {}
  moodle:
    entryPoints:
      - "web"
    rule: "Host(`elearning.humg.edu.vn`)"
    service: moodle
middlewares:
  moodle-stripprefix:
    stripPrefix:
      prefixes:
        - "/moodle"
      forceSlash: false
services:
  moodle-secure:
    loadBalancer:
      servers:
        - url: http://moodle:8080
      passHostHeader: true
      sticky:
        cookie: {}
  moodle:
    loadBalancer:
      servers:
        - url: http://moodle:8080
      passHostHeader: true
      sticky:
        cookie: {}
```

2.2 Các bước cài đặt

Mở terminal trên Ubuntu và chạy các lệnh sau để thực hiện việc cài đặt Docker, Docker compose và thực thi file script docker-compose.yml.

- Cài đặt Docker:

- Cập nhật kho phần mềm Unix: sudo apt-get update
- Gỡ bỏ phiên bản Docker hiện tại (nếu có): sudo apt-get remove docker docker-engine docker.io
- Cài phiên bản Docker mới nhất: sudo apt install docker.io
- Thiết lập chế độ chạy tự động cho Docker: sudo systemctl start docker & sudo systemctl enable docker

- Kiểm tra lại việc cài đặt (xem phiên bản Docker): docker --version

- Cài đặt Docker compose:

- Tải phiên bản mới nhất hiện tại của Docker compose: sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
- Gán quyền thực thi cho các tệp tin vừa tải xuống: sudo chmod +x /usr/local/bin/docker-compose
- Kiểm tra lại việc cài đặt (xem phiên bản): docker-compose --version

- Chạy file kịch bản docker-compose.yml:

- Di chuyển đến thư mục chứa file kịch bản docker-compose.yml
- Chạy lệnh: docker-compose up -d (có thể scale để tăng hiệu năng)

Kết quả cài đặt trên máy chủ:

```
        Name                      Command               State                              Ports
-------------------------------------------------------------------------------------------------------------------------------------
tracker_mariadb_1      /opt/bitnami/scripts/maria ...   Up      3306/tcp
tracker_moodle_1       /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_10      /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_11      /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_12      /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_13      /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_14      /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_15      /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_16      /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_17      /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_18      /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_19      /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_2       /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_20      /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_3       /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_4       /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_5       /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_6       /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_7       /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_8       /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_moodle_9       /opt/bitnami/scripts/moodl ...   Up      8080/tcp, 8443/tcp
tracker_phpmyadmin_1   /opt/bitnami/scripts/phpmy ...   Up      0.0.0.0:8080->8080/tcp, 0.0.0.0:8443->8443/tcp
tracker_traefik_1      /entrypoint.sh traefik           Up      0.0.0.0:443->443/tcp, 0.0.0.0:80->80/tcp, 0.0.0.0:8888->8888/tcp
```

2.3 Các bước cấu hình

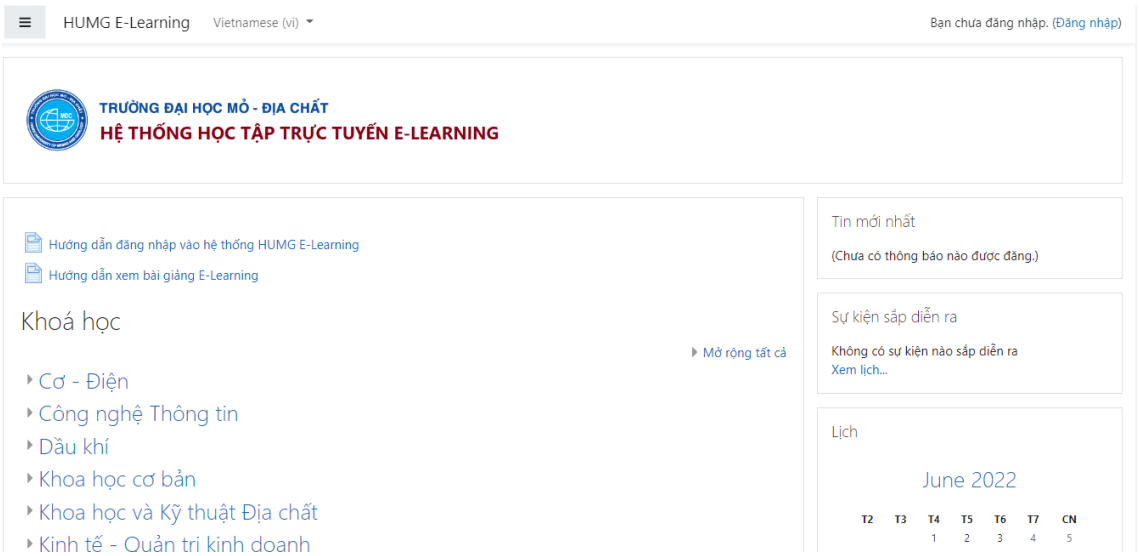
Để hệ thống có thể hoạt động được trên hạ tầng của cơ quan với tên miền elearning.humg.edu.vn, cần phải thực hiện các bước cấu hình sau:

- Tạo bản ghi A record với tên subdomain là elearning trên hệ thống quản lý tên miền (domain) humg.edu.vn của Trường Đại học Mỏ - Địa chất hiện đang duy trì tại công ty PA Vietnam tại địa chỉ: https://access.pavietnam.vn

- Cấu hình trên Sophos firewall của Nhà trường để ánh xạ tên miền trên đến đúng địa chỉ IP và port của dịch vụ đã được nhóm nghiên cứu cài đặt ở trên.

2.4 Kết quả

Hệ thống sau khi được triển khai trên máy chủ sẽ hoạt động tại địa chỉ được nêu ở trong file cấu hình phía trên.



KẾT LUẬN

Kiến trúc microservice không nhỏ như cái tên của nó. Mà nó là một kiến trúc dành cho các ứng dụng lớn. Nếu ứng dụng của bạn chưa đủ lớn và cũng không có ý định mở rộng trong tương lai thì hướng phát triển theo kiến trúc nguyên khối vẫn là một lựa chọn hợp lý.

Kiến trúc microservice giải quyết được rất nhiều vấn đề của kiến trúc nguyên khối, nó tỏ ra vượt trội kiến trúc nguyên khối ở nhiều khía cạnh. Tuy nhiên kiến trúc này cũng có không ít những hạn chế.

TÀI LIỆU THAM KHẢO

[1] https://smartbear.com/learn/api-design/what-are-microservices/

[2] https://www.edureka.co/blog/what-is-microservices/

[3] https://searchmicroservices.techtarget.com/definition/business-capability

[4] https://smartbear.com/learn/api-design/what-are-microservices/

[5] The What, Why, and How of a Microservices Architecture

[6] https://blog.newrelic.com/technology/microservices-what-they-are-why-to-use-them/

[7] https://whatis.techtarget.com/definition/business-logic

[8] https://www.quora.com/Should-each-microservice-have-its-own-database

[9] http://blog.christianposta.com/microservices/the-hardest-part-about-microservices-data/

[10] Best Practices for Building a Microservice Architecture

[11] https://blog.runscope.com/posts/5-reasons-not-to-use-microservices