**TRƯỜNG ĐẠI HỌC TÀI NGUYÊN VÀ MÔI TRƯỜNG**

**THÀNH PHỐ HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO ĐỒ ÁN MÔN HỌC**

**BẢO MẬT MẠNG MÁY TÍNH & HỆ THỐNG**

**ỨNG DỤNG KIẾN TRÚC MICROSERVICE VÀ CONTAINER HÓA TRONG XÂY DỰNG VÀ TRIỂN KHAI HỆ THỐNG QUẢN LÝ PHÒNG THỰC HÀNH TẠI TRƯỜNG HỌC**

Giảng viên: **ThS. Phạm Trọng Huynh**

Lớp: **10\_ĐH\_CNPM3**

Nhóm sinh viên thực hiện:

**Họ và tên Mã số sinh viên**

1. Nguyễn Bá Hùng 1050080264
2. Độ Nhật Nam 105008027

***TP. Hồ Chí Minh, 25 tháng 04 năm 2025***

**TRƯỜNG ĐẠI HỌC TÀI NGUYÊN VÀ MÔI TRƯỜNG**

**THÀNH PHỐ HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO ĐỒ ÁN MÔN HỌC**

**BẢO MẬT MẠNG MÁY TÍNH & HỆ THỐNG**

**ỨNG DỤNG KIẾN TRÚC MICROSERVICE VÀ CONTAINER HÓA TRONG XÂY DỰNG VÀ TRIỂN KHAI HỆ THỐNG QUẢN LÝ PHÒNG THỰC HÀNH TẠI TRƯỜNG HỌC**

Giảng viên: **ThS. Phạm Trọng Huynh**

Lớp: **10\_ĐH\_CNPM3**

Nhóm sinh viên thực hiện:

**Họ và tên Mã số sinh viên**

1. Nguyễn Bá Hùng 1050080264
2. Đỗ Nhật Nam 1050080278

***TP. Hồ Chí Minh, 25 tháng 04 năm 2025***

**LỜI CẢM ƠN**

**NHẬN XÉT**

**(Của giảng viên giảng dạy)**

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

………….., ngày….tháng….năm……

CÁN BỘ GIẢNG DẠY

*(ký tên)*

**MỤC LỤC**

[DANH MỤC CÁC BẢNG, SƠ ĐỒ, HÌNH 6](#_Toc196150263)

[LỜI MỞ ĐẦU 1](#_Toc196150264)

[Chương 1: TỔNG QUAN 1](#_Toc196150265)

[1.1 Mục tiêu nghiên cứu **Error! Bookmark not defined.**](#_Toc196150266)

[1.2 Nội dung nghiên cứu 3](#_Toc196150267)

[1.3 Phạm vi đạt được 3](#_Toc196150268)

[Chương 2: CƠ SỞ LÝ THUYẾT 3](#_Toc196150269)

[2.1 Tìm hiểu các phương pháp máy học 3](#_Toc196150270)

[2.1.1 Khái niệm 3](#_Toc196150271)

[2.1.2 Quy trình làm việc Machine Learning 3](#_Toc196150272)

[2.1.3 Phân loại các phương pháp Machine Learning 4](#_Toc196150273)

[2.2 Thuật toán Naïve Bayes 9](#_Toc196150274)

[2.2.1 Khái niệm: 9](#_Toc196150275)

[2.2.2 Định lý Naïve Bayes 9](#_Toc196150276)

[2.3 Tìm hiểu tổng về SDN 12](#_Toc196150277)

[2.3.1 Khái niệm 12](#_Toc196150278)

[2.3.2 Lợi ích 12](#_Toc196150279)

[2.3.3 Thách thức 13](#_Toc196150280)

[2.3.4 Kiến trúc mạng và cách hoạt động của SDN 13](#_Toc196150281)

[2.3.5 So sánh kiến trúc mạng SDN và mạng truyền thống 15](#_Toc196150282)

[2.3.6 Ứng dụng của SDN 15](#_Toc196150283)

[2.3.7 SDN dành cho mạng doanh nghiệp 16](#_Toc196150284)

[2.4 Tìm hiểu về giao thức OpenFlow 17](#_Toc196150285)

[2.4.1 Khái niệm 17](#_Toc196150286)

[2.4.2 Lịch sử và sự phát triển của OpenFlow 17](#_Toc196150287)

[2.4.3 Các đặc trưng của OpenFlow 18](#_Toc196150288)

[2.4.4 Phân loại phần cứng Openflow 18](#_Toc196150289)

[2.4.5 Chế độ hoạt động của OpenFlow 18](#_Toc196150290)

[2.4.6 Kiến trúc OpenFlow 19](#_Toc196150291)

[2.4.7 Các loại thông điệp của OpenFlow 22](#_Toc196150292)

[2.4.8 Cách hoạt động của Openflow 23](#_Toc196150293)

[2.4.9 Hành động OpenFlow gửi đến Controller 25](#_Toc196150294)

[2.4.10 Ưu điểm OpenFlow 25](#_Toc196150295)

[Chương 3: TRIỂN KHAI THỰC NGHIỆM 27](#_Toc196150296)

[3.1 Mô hình 27](#_Toc196150297)

[3.2 Kịch bản 27](#_Toc196150298)

[3.3 Triển khai 27](#_Toc196150299)

[Chương 4: KẾT LUẬN 33](#_Toc196150300)

[4.1 Kết luận 33](#_Toc196150301)

[4.2 Hướng phát triển 33](#_Toc196150302)

[TÀI LIỆU THAM KHẢO 34](#_Toc196150303)

# DANH MỤC CÁC BẢNG, SƠ ĐỒ, HÌNH

**Mục lục hình ảnh:**

[Hình 2.1 Quy trình làm việc Machine Learning 3](#_Toc196145799)

[Hình 2.2 Supervised Learning 5](#_Toc196145800)

[Hình 2.3 Unsupervised Learning 6](#_Toc196145801)

[Hình 2.4 Reinforcement learning 7](#_Toc196145802)

[Hình 2.5 Semi-Supervised Learning 7](#_Toc196145803)

[Hình 2.6 Self-Supervised Learning 8](#_Toc196145804)

[Hình 2.7 Transfer Learning 8](#_Toc196145805)

[Hình 2.8 Software-Defined Network 12](#_Toc196145806)

[Hình 2.9 Kiến trúc mạng SDN 14](#_Toc196145807)

[Hình 2.10 Ứng dụng phổ biến SDN 16](#_Toc196145808)

[Hình 2.11 Flow table 19](#_Toc196145809)

[Hình 2.12 Header Fields 20](#_Toc196145810)

[Hình 2.13 Per Table Counter 20](#_Toc196145811)

[Hình 2.14 OpenFlow 21](#_Toc196145812)

[Hình 2.15 OpenFlow Controller 22](#_Toc196145813)

# LỜI MỞ ĐẦU

Trong kỷ nguyên số hiện đại, việc quản lý và tối ưu hóa hạ tầng mạng doanh nghiệp đang trở thành một thách thức lớn. Sự gia tăng không ngừng của dữ liệu và nhu cầu băng thông đã thúc đẩy các doanh nghiệp tìm kiếm những giải pháp công nghệ tiên tiến nhằm đảm bảo hiệu suất và an toàn cho hệ thống mạng. Công nghệ mạng định nghĩa bằng phần mềm (Software-defined network - SDN) đã được phát triển như một công cụ mạnh mẽ, cho phép các quản trị viên mạng linh hoạt điều chỉnh cấu hình và quản lý tài nguyên hiệu quả hơn bao giờ hết.

Kiến trúc của SDN cho phép tách biệt lớp điều khiển khỏi lớp dữ liệu, từ đó giúp đơn giản hóa quá trình quản lý mạng. Sự phân tách này không chỉ làm giảm độ phức tạp trong việc cấu hình mà còn nâng cao khả năng tự động hóa và tối ưu hóa hoạt động của mạng. Nhờ vào SDN, doanh nghiệp có thể dễ dàng điều chỉnh các chính sách mạng để phù hợp với nhu cầu thực tế, từ đó nâng cao hiệu suất và giảm thiểu chi phí vận hành.

Đồng thời, thuật toán Naïve Bayes đã chứng tỏ được giá trị của mình trong việc phân tích và dự đoán dữ liệu. Dựa trên nguyên lý xác suất, thuật toán này cho phép phân loại dữ liệu một cách nhanh chóng và hiệu quả. Trong lĩnh vực an ninh mạng, Naïve Bayes có thể được áp dụng để phát hiện các mối đe dọa tiềm ẩn, bao gồm cả phân tích lưu lượng mạng.

Nhận thức được vài trò quan trọng của SDN kết hợp với thuật toán Naïve Bayes trong mô hình mạng doanh nghiệp nên chúng em quyết định lựa chọn đề tài “**ỨNG DỤNG SDN VÀ THUẬT TOÁN NAÏVE BAYES VÀO MÔ HÌNH MẠNG DOANH NGHIỆP**”. Mục tiêu là phân tích dữ liệu, nhận diện và thông báo về các cuộc tấn công, trong đó có cuộc tấn công DDoS làm ví dụ minh họa. Chúng em sẽ khảo sát các phương pháp tích hợp hai công nghệ này để xây dựng một mô hình mạng vừa linh hoạt vừa an toàn.

Bài báo cáo gồm có thành 4 chương:

Chương 1: Tổng quan

Chương 2: Cơ sở lý thuyết

Chương 3: Triển khai thực nghiệm

Chương 4: Kết luận

# Chương 1: TỔNG QUAN

Trong bối cảnh chuyển đổi số mạnh mẽ trong giáo dục, việc quản lý hiệu quả cơ sở vật chất, đặc biệt là các phòng thực hành, đóng vai trò quan trọng trong nâng cao chất lượng dạy và học. Kiến trúc Microservice và công nghệ container hóa (Docker, Kubernetes) đã nổi lên như giải pháp tối ưu, giúp xây dựng hệ thống quản lý linh hoạt, dễ bảo trì và có khả năng mở rộng cao.

Microservice cho phép phân tách hệ thống thành các dịch vụ độc lập (quản lý phòng, đặt lịch, theo dõi thiết bị, báo cáo), giúp phát triển và triển khai từng module một cách nhanh chóng. Trong khi đó, container hóa đảm bảo tính nhất quán giữa các môi trường (development, testing, production), đồng thời tối ưu hóa tài nguyên server và đơn giản hóa quy trình CI/CD.

Việc ứng dụng hai công nghệ này không chỉ giúp nhà trường triển khai hệ thống quản lý phòng thực hành một cách hiệu quả, tiết kiệm chi phí mà còn dễ dàng tích hợp các tính năng mới trong tương lai (IoT để giám sát thiết bị, AI để dự đoán nhu cầu sử dụng phòng). Ngoài ra, khả năng tự động scale của container cũng phù hợp với lưu lượng truy cập biến động theo thời khóa biểu, đảm bảo hệ thống luôn ổn định.

## Tính cấp thiết của đề tài

Trong bối cảnh chuyển đổi số giáo dục mạnh mẽ hiện nay, việc ứng dụng kiến trúc Microservice và Container hóa vào xây dựng hệ thống quản lý phòng thực hành tại trường học là yêu cầu cấp thiết. Các hệ thống quản lý truyền thống theo kiến trúc nguyên khối (Monolithic) đang bộc lộ nhiều hạn chế: khó mở rộng khi số lượng phòng học và người dùng tăng cao, khó bảo trì do phải triển khai lại toàn bộ ứng dụng khi cập nhật, và thiếu linh hoạt trong việc tích hợp các công nghệ mới như AI hay IoT.

Kiến trúc Microservice kết hợp Container hóa sẽ khắc phục triệt để những vấn đề này bằng cách chia hệ thống thành các dịch vụ độc lập (quản lý đặt phòng, điểm danh, theo dõi thiết bị...), mỗi dịch vụ có thể phát triển, triển khai và mở rộng riêng biệt. Công nghệ Container hóa với Docker và Kubernetes cho phép triển khai hệ thống linh hoạt, tự động mở rộng tài nguyên khi cần và đảm bảo tính sẵn sàng cao. Đặc biệt, giải pháp này còn mở đường cho việc tích hợp các tính năng hiện đại như điểm danh bằng AI nhận diện khuôn mặt hay giám sát thiết bị qua cảm biến IoT.

Việc áp dụng kiến trúc hiện đại này không chỉ nâng cao hiệu quả quản lý, tiết kiệm chi phí vận hành mà còn tạo nền tảng vững chắc cho các trường học trong quá trình chuyển đổi số, đáp ứng yêu cầu ngày càng cao của giáo dục thông minh trong kỷ nguyên 4.0. Đây chính là xu hướng tất yếu để xây dựng hệ thống quản lý phòng thực hành hiệu quả, linh hoạt và bền vững cho các cơ sở giáo dục hiện nay.

Từ những phân tích trên, nhóm nghiên cứu quyết định lựa chọn đề tài "Ứng dụng kiến trúc Microservice và Container hóa trong xây dựng và triển khai hệ thống quản lý phòng thực hành tại trường học" như một giải pháp tối ưu để giải quyết các thách thức trong quản lý phòng học truyền thống. Việc áp dụng kiến trúc Microservice sẽ mang lại sự linh hoạt, khả năng mở rộng và dễ bảo trì, trong khi Container hóa đảm bảo tính nhất quán và hiệu quả trong triển khai.

## Mục tiêu chọn đề tài

Đề tài "Ứng dụng kiến trúc Microservice và Container hóa trong xây dựng và triển khai hệ thống quản lý phòng thực hành tại trường học" hướng đến ba nhóm mục tiêu chính. Về mặt khoa học, đề tài tập trung nghiên cứu và đánh giá hiệu quả của kiến trúc Microservice kết hợp công nghệ Container hóa (Docker và Kubernetes) so với hệ thống quản lý truyền thống, đồng thời đề xuất mô hình thiết kế linh hoạt có khả năng tích hợp các công nghệ mới như AI và IoT trong tương lai.

Về mặt công nghệ, đề tài nhằm xây dựng hệ thống quản lý với các chức năng cốt lõi gồm đặt lịch phòng, điểm danh sinh viên, quản lý thiết bị và báo cáo tự động, được triển khai trên nền tảng phân tán với khả năng tự động mở rộng và phục hồi khi gặp sự cố. Hệ thống sẽ sử dụng API Gateway để quản lý truy cập và Service Mesh để giám sát giao tiếp giữa các dịch vụ.

Về ứng dụng thực tế, đề tài hướng tới nâng cao hiệu quả quản lý phòng thực hành thông qua tự động hóa quy trình, cải thiện trải nghiệm người dùng và tối ưu chi phí vận hành. Kết quả của đề tài sẽ tạo tiền đề cho việc ứng dụng công nghệ hiện đại trong giáo dục và có thể nhân rộng cho các trường học khác, góp phần thúc đẩy quá trình chuyển đổi số trong lĩnh vực giáo dục.

## Nội dung nghiên cứu

Đề tài "Ứng dụng kiến trúc Microservice và Container hóa trong xây dựng và triển khai hệ thống quản lý phòng thực hành tại trường học" tập trung vào các nội dung nghiên cứu chính sau:

Nghiên cứu tổng quan về kiến trúc Microservice và công nghệ Container hóa, bao gồm các nguyên lý hoạt động, ưu nhược điểm và ứng dụng thực tế. Phân tích yêu cầu hệ thống quản lý phòng thực hành hiện tại tại các trường học, bao gồm các chức năng cốt lõi như đăng ký sử dụng phòng, điểm danh sinh viên, quản lý thiết bị và báo cáo. Thiết kế hệ thống theo kiến trúc Microservice với các dịch vụ độc lập như Booking Service, Attendance Service, Device Management Service và Reporting Service, kết hợp với API Gateway để quản lý tập trung.

Triển khai hệ thống bằng công nghệ Container hóa với Docker và quản lý bằng Kubernetes, bao gồm các công việc như đóng gói dịch vụ, cấu hình tự động mở rộng và quản lý giao tiếp giữa các service. Đánh giá hiệu quả hệ thống thông qua kiểm thử hiệu năng, bảo mật và so sánh với hệ thống cũ. Cuối cùng, triển khai thử nghiệm tại môi trường thực tế để thu thập phản hồi và tối ưu hóa hệ thống.

## Phạm vi đề tài

Đề tài "Ứng dụng kiến trúc Microservice và Container hóa trong xây dựng và triển khai hệ thống quản lý phòng thực hành tại trường học" được giới hạn trong các phạm vi cụ thể sau: Về công nghệ, nghiên cứu tập trung vào kiến trúc Microservice với các công nghệ chính gồm Docker cho container hóa và Kubernetes để triển khai, sử dụng Spring Boot/Node.js cho phát triển dịch vụ và PostgreSQL/MongoDB cho hệ quản trị cơ sở dữ liệu. Về chức năng, hệ thống bao gồm các module cốt lõi như quản lý đặt phòng, điểm danh, theo dõi thiết bị và báo cáo, nhưng không bao gồm các hệ thống phức tạp như thanh toán hay tích hợp ERP.

Về triển khai, đề tài giới hạn ở quy mô thử nghiệm với cụm Kubernetes (3-5 node) và kiểm thử hiệu năng ở mức 100-500 người dùng đồng thời. Đối tượng áp dụng ban đầu là một số phòng thực hành cụ thể trong trường với các nhóm người dùng chính gồm quản trị viên, giảng viên và sinh viên. Thời gian thực hiện được xác định trong phạm vi 6 tháng với nguồn lực của nhóm nghiên cứu. Xử lý tối ưu nhất có thể khi có nhiều request

Phạm vi này được xác định nhằm đảm bảo tính khả thi của đề tài trong khi vẫn đáp ứng được các yêu cầu cơ bản của hệ thống quản lý phòng thực hành, đồng thời tạo tiền đề cho việc mở rộng và phát triển thêm các tính năng trong tương lai.

# Chương 2: CƠ SỞ LÝ THUYẾT

## Kiến trúc về Microservice

### **2.1.1 Khái niệm và đặc điểm**

a. Khái niệm

Kiến trúc Microservice (Microservices Architecture - MSA) là một phương pháp phát triển phần mềm trong đó ứng dụng được chia thành nhiều dịch vụ nhỏ, độc lập, mỗi dịch vụ thực hiện một chức năng nghiệp vụ cụ thể. Các dịch vụ này giao tiếp với nhau thông qua các API (REST, gRPC, message queue) và có thể được triển khai, mở rộng và quản lý riêng biệt.

b. Đặc điểm

Tính độc lập (Decentralized): Mỗi microservice có cơ sở dữ liệu riêng (Database per Service), codebase riêng và có thể được phát triển bằng các ngôn ngữ khác nhau.

Tự quản lý (Autonomous): Mỗi service có thể được triển khai, nâng cấp và mở rộng mà không ảnh hưởng đến các service khác.

Giao tiếp qua API: Các microservice thường sử dụng RESTful API, gRPC hoặc message brokers (Kafka, RabbitMQ) để trao đổi dữ liệu.

Khả năng chịu lỗi (Fault Isolation): Nếu một service bị lỗi, nó không làm sập toàn bộ hệ thống.

### **2.1.2 So sánh với kiến trúc nguyên khối (Monolithic)**

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Monolithic Architecture** | **Microservice Architecture** |
| Cấu trúc ứng dụng | Một khối (single codebase), tất cả chức năng được tích hợp chặt chẽ. | Ứng dụng được chia thành nhiều dịch vụ nhỏ, độc lập, mỗi service phụ trách một chức năng riêng. |
| Triển khai (Deployment) | Phải deploy lại toàn bộ ứng dụng dù chỉ thay đổi một tính năng nhỏ. | Có thể deploy từng service riêng lẻ mà không ảnh hưởng đến hệ thống. |
| Khả năng mở rộng (Scalability) | Khó scale từng phần, thường phải scale toàn bộ ứng dụng. | Dễ dàng scale theo từng service dựa trên nhu cầu tải (ví dụ: Service A cần 10 instances, Service B chỉ cần 2). |
| Công nghệ sử dụng | Bị giới hạn trong một ngôn ngữ/framework chính. | Linh hoạt sử dụng đa ngôn ngữ (Java, Python, Go...) và database (SQL, NoSQL) tùy theo service. |
| Hiệu suất (Performance) | Giao tiếp nội bộ nhanh do chạy trong cùng một process. | Có độ trễ (latency) do giao tiếp qua mạng (API/RPC), cần tối ưu hóa. |
| Khả năng chịu lỗi (Fault Tolerance) | Lỗi ở một module có thể làm sập toàn bộ hệ thống. | Cô lập lỗi tốt hơn, một service hỏng không ảnh hưởng đến các service khác (nhờ Circuit Breaker, Retry). |
| Bảo trì & Nâng cấp | Khó khăn khi codebase phình to, rủi ro cao khi thay đổi. | Dễ bảo trì, có thể cập nhật từng service mà không sợ phá vỡ hệ thống. |
| Phát triển (Development) | Phù hợp với team nhỏ, dự án đơn giản. | Phù hợp với team lớn, mỗi team phụ trách một service riêng. |
| Quản lý dữ liệu | Dùng chung một database, dễ đảm bảo tính nhất quán (ACID). | Mỗi service có database riêng, khó đồng bộ → cần Saga Pattern, Event Sourcing. |
| Phức tạp hệ thống | Đơn giản ban đầu, nhưng càng phát triển càng khó kiểm soát. | Phức tạp ngay từ đầu (cần Service Mesh, API Gateway, Monitoring). |
| Phù hợp | Ứng dụng nhỏ, startup, yêu cầu đơn giản. | Hệ thống lớn, cần high availability (Netflix, Amazon), cloud-native. |

### **2.1.3 Ưu nhược điểm của kiến trúc Microservice**

Ưu điểm:

Kiến trúc Microservice mang lại nhiều lợi ích nổi bật cho phát triển phần mềm hiện đại. Tính độc lập cao là ưu điểm quan trọng nhất, khi mỗi service có thể được phát triển, triển khai và nâng cấp riêng biệt mà không ảnh hưởng đến toàn hệ thống.

Nhược điểm:

Mặc dù có nhiều ưu điểm, kiến trúc Microservice cũng tồn tại một số hạn chế đáng kể. Độ phức tạp cao là thách thức lớn nhất, đòi hỏi nhiều công cụ hỗ trợ như Docker, Kubernetes, API Gateway, Service Mesh... Việc gỡ lỗi (debug) cũng trở nên khó khăn hơn do tính chất phân tán của hệ thống, cần đến các công cụ đặc biệt như Distributed Tracing (Jaeger).

Vấn đề giao tiếp giữa các service cũng là điểm yếu cần lưu ý. Giao thức API/RPC làm tăng độ trễ (latency) so với lời gọi hàm nội bộ. Đảm bảo tính nhất quán dữ liệu giữa các service đòi hỏi phải áp dụng các mẫu thiết kế phức tạp như Saga Pattern hay Event Sourcing.

## Container hóa và Docker

### **2.2.1 Khái niệm container và Docker**

Container là một công nghệ đóng gói ứng dụng cùng với tất cả các thành phần phụ thuộc (thư viện, biến môi trường, cấu hình) vào một gói độc lập, giúp ứng dụng chạy nhất quán trên mọi môi trường từ development tới production. Khác với máy ảo truyền thống (VM), container sử dụng chung kernel của hệ điều hành host nên nhẹ hơn, khởi động nhanh hơn (chỉ vài giây) và tiêu thụ ít tài nguyên hơn.

Docker là nền tảng mã nguồn mở phổ biến nhất để làm việc với container. Nó cung cấp các công cụ để đóng gói ứng dụng thành Docker image (như một khuôn mẫu), chia sẻ image qua Docker Hub, và khởi chạy các container từ những image này. Kiến trúc Docker bao gồm Docker Engine (phần core để build và run container), Docker Image (file chứa ứng dụng đã đóng gói), và Docker Container (instance đang chạy của image).

### **2.2.2 Lợi ích của container hóa trong phát triển phần mềm**

Container hóa đã mang lại những ưu điểm vượt trội cho quá trình phát triển phần mềm hiện đại. Đầu tiên, công nghệ này giải quyết triệt để vấn đề "chạy trên máy tôi thì được" bằng cách đóng gói ứng dụng cùng toàn bộ dependencies vào một container duy nhất. Nhờ vậy, ứng dụng sẽ hoạt động nhất quán từ môi trường development cho đến production mà không gặp các lỗi liên quan đến khác biệt về cấu hình hệ thống.

Về hiệu xuất: Container mang lại lợi thế lớn so với máy ảo truyền thống. Các container sử dụng chung kernel của hệ điều hành host nên tiêu thụ ít tài nguyên hơn, khởi động nhanh hơn (chỉ vài giây thay vì vài phút như VM). Điều này đặc biệt hữu ích khi cần scale hệ thống nhanh chóng để đáp ứng biến động về lưu lượng.

Về mặt bảo mật: Container cung cấp cơ chế cô lập giữa các ứng dụng. Mỗi container có không gian làm việc riêng biệt về filesystem và network, giảm thiểu nguy cơ xung đột.

### **2.2.3 Docker Swarm và quản lý container trong môi trường sản xuất**

Docker Swarm là công cụ orchestration tích hợp sẵn trong Docker Engine, giúp quản lý cụm các Docker host như một hệ thống duy nhất. Khác với Kubernetes phức tạp, Docker Swarm được thiết kế đơn giản, dễ triển khai và phù hợp cho các hệ thống vừa và nhỏ. Swarm sử dụng kiến trúc master-worker, trong đó các node manager chịu trách nhiệm điều phối trong khi node worker thực thi container.

Khi triển khai container vào môi trường production, Docker Swarm cung cấp các tính năng quan trọng như service discovery, load balancing và rolling updates. Các service được định nghĩa thông qua docker-compose file có thể dễ dàng mở rộng bằng lệnh đơn giản. Swarm tự động cân bằng tải giữa các container instance và đảm bảo high availability bằng cách tự động khởi chạy lại container bị lỗi.

Swarm nổi bật với tính đơn giản và dễ sử dụng, tích hợp chặt chẽ với hệ sinh thái Docker. Việc triển khai chỉ cần vài lệnh cơ bản để thiết lập cluster. Swarm cũng tiêu thụ ít tài nguyên hơn Kubernetes, phù hợp cho hệ thống không yêu cầu scaling phức tạp. Tính năng built-in load balancing và secret management giúp bảo mật thông tin nhạy cảm.

Docker Swarm có một số hạn chế so với Kubernetes như khả năng auto-scaling hạn chế, ít options cho storage management, và cộng đồng hỗ trợ nhỏ hơn. Swarm cũng thiếu các công cụ monitoring mạnh mẽ như Kubernetes. Điều này khiến Swarm phù hợp hơn với các ứng dụng không yêu cầu scaling phức tạp hoặc hệ thống đã quen thuộc với Docker.

Trong môi trường production, Docker Swarm thường được sử dụng kết hợp với các công cụ bổ sung như Portainer cho giao diện quản lý, Traefik làm reverse proxy, và Prometheus để giám sát. Việc rolling update được thực hiện dễ dàng thông qua docker service update, giúp ứng dụng luôn available trong quá trình deploy phiên bản mới.

## Giới thiệu hệ thống quản lý phòng thực hành

Hệ thống quản lý phòng thực hành là giải pháp công nghệ được thiết kế để số hóa và tự động hóa quy trình quản lý, sử dụng các phòng học thực hành tại các cơ sở giáo dục. Hệ thống giúp thay thế phương pháp quản lý thủ công truyền thống bằng các công nghệ hiện đại, mang lại hiệu quả cao trong việc sắp xếp lịch học, theo dõi thiết bị và quản lý tài nguyên.

Hệ thống cung cấp bộ tính năng toàn diện bao gồm: đăng ký và quản lý lịch sử dụng phòng, điểm danh sinh viên, theo dõi tình trạng thiết bị, cảnh báo hỏng hóc và báo cáo thống kê. Giảng viên có thể dễ dàng đặt lịch phòng thực hành thông qua giao diện web hoặc ứng dụng di động, trong khi bộ phận quản trị có thể nắm bắt toàn bộ hoạt động sử dụng phòng theo thời gian thực.

Việc ứng dụng hệ thống quản lý phòng thực hành mang lại nhiều lợi ích thiết thực: tối ưu hóa việc sử dụng tài nguyên phòng học, giảm thiểu thời gian chết, nâng cao trải nghiệm người dùng cho cả giảng viên và sinh viên. Hệ thống còn giúp nhà trường tiết kiệm chi phí vận hành, giảm thiểu sai sót trong quản lý và cung cấp dữ liệu để phân tích, cải tiến chất lượng đào tạo.

Hệ thống được xây dựng dựa trên kiến trúc Microservice và công nghệ Container hóa, cho phép dễ dàng mở rộng và tích hợp các tính năng mới trong tương lai. Việc sử dụng Docker và Kubernetes giúp hệ thống hoạt động ổn định, có khả năng chịu tải cao và dễ dàng triển khai trên nhiều môi trường khác nhau.

### **2.3.1 Đặc điểm và nhu cầu quản lý phòng thực hành**

Hệ thống được xây dựng dựa trên kiến trúc Microservice và công nghệ Container hóa mang những đặc điểm kỹ thuật nổi bật. Mỗi chức năng quản lý như đặt phòng, điểm danh hay theo dõi thiết bị được phát triển thành các microservice độc lập, giao tiếp thông qua API. Các service này được đóng gói trong container Docker, cho phép triển khai linh hoạt trên nhiều môi trường khác nhau từ cloud đến hệ thống vật lý của trường họ

Nhu cầu quản lý phòng thực hành tập trung vào các vấn đề chính: quản lý lịch sử dụng, theo dõi tình trạng thiết bị và đảm bảo an toàn trong quá trình sử dụng. Các trường học cần hệ thống giúp phân bổ phòng hợp lý theo nhu cầu của giảng viên và môn học, tránh tình trạng xung đột lịch sử dụng. Đồng thời, hệ thống cần theo dõi được tình trạng các thiết bị quan trọng, lịch sử bảo trì và cảnh báo khi có sự cố hỏng hóc.

Bài toán đặt ra yêu cầu quản lý đa dạng các loại phòng thực hành với đặc thù riêng về thiết bị và cách thức sử dụng. Hệ thống cần giải quyết vấn đề tối ưu hóa lịch sử dụng tránh xung đột, đồng thời tích hợp khả năng giám sát thiết bị theo thời gian thực thông qua các cảm biến IoT. Nhu cầu báo cáo chi tiết về hiệu quả sử dụng phòng và thiết bị cũng là yêu cầu quan trọng đối với công tác quản lý.

Đề tài đề xuất giải pháp tổng thể ứng dụng các công nghệ hiện đại. Kiến trúc Microservice giúp hệ thống dễ dàng mở rộng và bảo trì từng thành phần độc lập. Công nghệ Container hóa với Docker đảm bảo tính nhất quán khi triển khai trên nhiều môi trường khác nhau. Kubernetes được sử dụng để tự động điều phối và mở rộng các container theo nhu cầu tải thực tế.

Hệ thống được thiết kế với khả năng tích hợp dễ dàng với các hệ thống quản lý đào tạo hiện có của nhà trường thông qua API Gateway. Kiến trúc module hóa cho phép bổ sung các tính năng mới như nhận diện khuôn mặt cho điểm danh hay giám sát thiết bị qua IoT mà không ảnh hưởng đến hoạt động của các service hiện có. Đây là nền tảng quan trọng để phát triển thành hệ thống quản lý thông minh trong tương lai.

### **2.3.2 Đặc điểm và nhu cầu quản lý phòng thực hành**

**Quản lý phòng thực hành:**

Thêm/Sửa/Xóa thông tin phòng: Cho phép quản trị viên cập nhật danh sách phòng thực hành (thông tin cơ bản, loại phòng, sức chứa, thiết bị đi kèm).

Theo dõi tình trạng phòng: Hiển thị trạng thái phòng theo thời gian thực (sẵn sàng, đang sử dụng, bảo trì) qua giao diện trực quan.

Lập lịch sử dụng phòng: Hỗ trợ đăng ký lịch sử dụng phòng tự động, kiểm tra xung đột lịch, gửi thông báo nhắc nhở.

Thống kê tần suất sử dụng: Xuất báo cáo định kỳ về hiệu quả sử dụng phòng (theo tuần/tháng/học kỳ).

**Quản lý thiết bị:**

Theo dõi trạng thái thiết bị: Cập nhật tình trạng hoạt động của từng thiết bị (máy tính, máy chiếu, dụng cụ thí nghiệm...).

Quản lý cấu hình máy tính: Lưu trữ thông số kỹ thuật, phần mềm đã cài đặt, lịch sử nâng cấp.

Lịch sử bảo trì/sửa chữa: Ghi nhận chi tiết các lần bảo trì, thay thế linh kiện, chi phí phát sinh.

Cảnh báo sự cố: Tự động thông báo khi phát hiện thiết bị hỏng hóc hoặc đến hạn bảo dưỡng.

**Quản lý người dùng:**

Phân quyền chi tiết:

Admin: Toàn quyền quản lý hệ thống.

Giáo viên: Đặt lịch phòng, xem báo cáo thiết bị.

Sinh viên: Đăng ký sử dụng phòng, xem lịch sử cá nhân.

Thông tin cá nhân: Lưu trữ hồ sơ người dùng (ảnh, chuyên ngành, quyền truy cập).

Lịch sử sử dụng: Theo dõi hoạt động của từng người dùng (phòng đã sử dụng, thiết bị đã mượn).

**Ứng dụng hỗ trợ:**

Kết nối máy tính: Ghép nối toàn bộ máy tính trong phòng thực hành với hệ thống web quản lý.

Quản lý máy tính sinh viên: Giám sát máy tính đang được sử dụng (thời gian, người dùng, phần mềm đang chạy).

Truyền file: Hỗ trợ chia sẻ tài liệu giữa máy giáo viên và sinh viên trong cùng phòng.

Giám sát từ xa: Cho phép quản trị viên theo dõi và điều khiển máy tính trong phòng thực hành khi cần thiết.

**Công nghệ áp dụng**

Frontend:

React: Thư viện UI

TypeScript: Ngôn ngữ lập trình

Mantine UI: Component library

Vite: Công cụ xây dựng frontend hiện đại

Backend:

NestJS: Framework Node.js để xây dựng microservices

TCP/Microservices: Giao thức giao tiếp giữa các service

PostgreSQL: Cơ sở dữ liệu quan hệ

Prisma: ORM để tương tác với database

Jest: Framework kiểm thử

Database: PostgreSQL cho dữ liệu quan hệ

DevOps:

Docker & Docker Compose: Container hóa và điều phối dịch vụ

Nginx: Reverse proxy và cân bằng tải

Redis: Cache và message broker

### **2.3.3 Lý do chọn kiến trúc Microservice và Docker**

Kiến trúc Microservice được lựa chọn do phù hợp với yêu cầu của hệ thống quản lý phòng thực hành - một hệ thống đa chức năng và phức tạp. Microservice cho phép tách biệt các chức năng chính như quản lý đặt phòng, điểm danh và theo dõi thiết bị thành các service độc lập, giúp việc phát triển và bảo trì trở nên dễ dàng hơn. Mỗi service có thể được phát triển bằng ngôn ngữ phù hợp nhất và scale độc lập theo nhu cầu thực tế.

Docker được chọn để triển khai các microservice nhờ khả năng đóng gói ứng dụng cùng môi trường chạy vào container, đảm bảo tính nhất quán từ môi trường phát triển đến production. Công nghệ container hóa giúp giải quyết triệt để vấn đề "chạy trên máy tôi thì được" thường gặp. Docker còn tối ưu hiệu suất khi cho phép chạy nhiều service cùng lúc trên một máy chủ với mức tiêu thụ tài nguyên thấp hơn nhiều so với máy ảo.

Sự kết hợp giữa Microservice và Docker mang lại lợi ích vượt trội cho hệ thống. Kiến trúc này cho phép từng service được triển khai, nâng cấp và scale độc lập, trong khi Docker đảm bảo tính đồng nhất và khả năng di chuyển giữa các môi trường. Đặc biệt, khi kết hợp với Kubernetes, hệ thống có thể tự động mở rộng tài nguyên theo nhu cầu và tự phục hồi khi gặp sự cố, đáp ứng tốt yêu cầu về tính sẵn sàng cao của hệ thống quản lý phòng thực hành.

# Chương 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

## Phân tích yêu cầu nghiệp vụ kịch bản

### **3.1.1 Quản lý thông tin phòng máy**

Quản lý thông tin phòng máy cần cung cấp đầy đủ các thông tin cơ bản về phòng thực hành bao gồm tên phòng, vị trí, sức chứa và loại phòng (phòng máy tính, lab hóa học, xưởng cơ khí). Hệ thống phải quản lý được danh sách các thiết bị đi kèm mỗi phòng và trạng thái hiện tại (đang hoạt động, bảo trì hoặc ngừng sử dụng), đồng thời cho phép quản trị viên thêm mới, chỉnh sửa hoặc vô hiệu hóa các phòng máy khi cần thiết.

### **3.1.2 Quản lý trạng thái thiết bị**

Quản lý trạng thái thiết bị đòi hỏi hệ thống phải theo dõi chi tiết tình trạng hoạt động của từng thiết bị (đang sử dụng, sẵn sàng hoặc hỏng hóc), bao gồm cả thông số kỹ thuật và phiên bản phần mềm. Hệ thống cần lưu trữ lịch sử bảo trì, sửa chữa và có khả năng tự động cảnh báo khi thiết bị gần hết hạn bảo hành hoặc cần bảo dưỡng định kỳ.

### **3.1.3 Quản lý tình trạng phòng và lịch sử sử dụng**

Quản lý tình trạng phòng và lịch sử sử dụng yêu cầu hệ thống phải quản lý chặt chẽ lịch sử dụng phòng, bao gồm các chức năng đăng ký, hủy lịch và theo dõi tình trạng thực tế (trống, đang sử dụng hoặc bảo trì). Hệ thống cần cung cấp các báo cáo thống kê về tần suất sử dụng và hiệu suất khai thác phòng theo nhiều tiêu chí khác nhau.

### **3.1.3 Quản lý tài khoản người dùng và phân quyền**

Quản lý tài khoản người dùng và phân quyền cần phân loại rõ ràng các nhóm người dùng (quản trị viên, giảng viên, sinh viên) với các quyền hạn tương ứng. Hệ thống phải có cơ chế bảo mật mạnh (xác thực đa yếu tố, mã hóa thông tin) và ghi lại đầy đủ nhật ký truy cập để theo dõi các hoạt động quan trọng.

## Thiết kế kiến trúc Microservice

### **3.2.1 API Gateway và vai trò điều phối**

A screenshot of a computer screen

AI-generated content may be incorrect.

Hình 1: API Gateway và vai trò điều phối

### **3.2.2 User Service: Quản lý người dùng và xác thực**

A diagram of a computer

AI-generated content may be incorrect.

Hình 2: Quản lý người dùng và xác thực

### **3.2.3 Room Service: Quản lý thông tin phòng thực hành**

A diagram of a computer service

AI-generated content may be incorrect.

Hình 3: Quản lý thông tin phòng thực hành

### **3.2.4 Computer Service: Quản lý thông tin và trạng thái máy tính**

A diagram of a computer server

AI-generated content may be incorrect.

Hình 4: Quản lý thông tin và trạng thái máy tính

### **3.2.5 Giao tiếp giữa các Microservice thông qua NATS**

A diagram of a service

AI-generated content may be incorrect.

Hình 5: Giao tiếp giữa các Microservice thông qua NATS

### **3.2.6 Sơ đồ kiến trúc tổng thể hệ thống**

A diagram of a computer

AI-generated content may be incorrect.

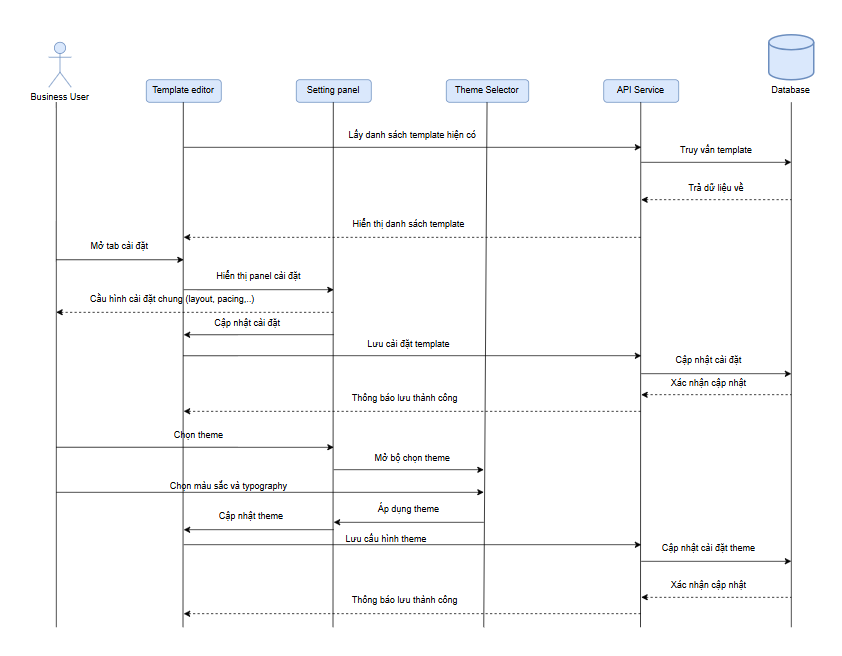
Hình 6: Sơ đồ kiến trúc tổng thể hệ thống

### **3.2.7 Sơ đồ tuần tự (Sequence Diagram) tương tác giữa các service**

A diagram of a diagram

AI-generated content may be incorrect.

Hình 7: Sơ đồ tuần tự



Hình 8: Sơ đồ tuần tự

A diagram of a project

AI-generated content may be incorrect.

Hình 9: Sơ đồ tuần tự

A diagram with text and lines

AI-generated content may be incorrect.

Hình 10: Sơ đồ tuần tự

A diagram of a service

AI-generated content may be incorrect.

Hình 11: Sơ đồ tuần tự

A diagram with text and images

AI-generated content may be incorrect.

Hình 12: Sơ đồ tuần tự

A diagram with text and symbols

AI-generated content may be incorrect.

Hình 13: Sơ đồ tuần tự

A diagram of a diagram

AI-generated content may be incorrect.

Hình 14: Sơ đồ tuần tự

A diagram with text and arrows

AI-generated content may be incorrect.

Hình 15: Sơ đồ tuần tự

A diagram with text and arrows

AI-generated content may be incorrect.

Hình 16: Sơ đồ tuần tự

A diagram of a diagram

AI-generated content may be incorrect.

Hình 17: Sơ đồ tuần tự

A diagram with text and arrows

AI-generated content may be incorrect.

Hình 18: Sơ đồ tuần tự

A diagram with text and images

AI-generated content may be incorrect.

Hình 19: Sơ đồ tuần tự

### **3.2.8 Sơ đồ Component Diagram của các Microservice**

A diagram of a computer

AI-generated content may be incorrect.

Hình 20: Sơ đồ thành phần

# Chương 4: KẾT LUẬN

## Kết luận

## Hướng phát triển

# TÀI LIỆU THAM KHẢO

* **Open Networking Foundation (ONF)**. (n.d.). *What is SDN?*. Retrieved from [https://opennetworking.org](https://opennetworking.org/)
* Goransson, P., & Black, C. (2014). *Software Defined Networks: A Comprehensive Approach*. Morgan Kaufmann.
* Kreutz, D., Ramos, F. M. V., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). "Software-defined networking: A comprehensive survey." *Proceedings of the IEEE*, 103(1), 14-76. doi:10.1109/JPROC.2014.2371999
* Cisco Systems. (n.d.). *Software-Defined Networking (SDN) Solutions*. Retrieved from [https://www.cisco.com](https://www.cisco.com/)
* VMware. (n.d.). *VMware NSX: The Network Virtualization and Security Platform*. Retrieved from [https://www.vmware.com](https://www.vmware.com/)
* Hu, F., Hao, Q., & Bao, K. (2014). "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation." *IEEE Communications Surveys & Tutorials*, 16(4), 2181-2206. doi:10.1109/COMST.2014.2326417
* **Juniper Networks**. (n.d.). *SDN for Modern Networking*. Retrieved from [https://www.juniper.net](https://www.juniper.net/)
* Difference between SDN and SDN- WAN, https://www.geeksforgeeks.org/difference-between-sdn-and-sdn-wan/
* https://www.securityweek.com/security-challenges-sdn-and-cloud-critical-role-visibility/
* Mạng SDN là gì? Tìm hiểu về Software Defined Networking, Hưng Nguyễn, https://vietnix.vn/sdn-la-gi/