

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN



LÝ TUẤN AN – 52000620
PHÙNG PHÚC HẬU – 52000443

TRIỂN KHAI DỊCH VỤ WEB CÓ KHẢ NĂNG MỞ RỘNG SỬ DỤNG DOCKER SWARM

DỰ ÁN CÔNG NGHỆ THÔNG TIN KỸ THUẬT PHẦN MỀM

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN



LÝ TUẤN AN – 52000620
PHÙNG PHÚC HẬU – 52000443

TRIỂN KHAI DỊCH VỤ WEB
CÓ KHẢ NĂNG MỞ RỘNG
SỬ DỤNG DOCKER SWARM

DỰ ÁN CÔNG NGHỆ THÔNG TIN
KỸ THUẬT PHẦN MỀM

Người hướng dẫn
ThS. Mai Văn Mạnh

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

LỜI CẢM ƠN

Lời đầu tiên, chúng tôi xin gửi lời cảm ơn chân thành và sâu sắc nhất đến Thầy Mai Văn Mạnh, người đã tận tình giải đáp các thắc mắc và chia sẻ các kiến thức mới trong quá trình chúng tôi tìm hiểu và thực hiện dự án công nghệ thông tin. Sự hỗ trợ quý báu của thầy đã giúp chúng tôi vượt qua nhiều khó khăn và hoàn thành bài dự án một cách tốt đẹp.

Lời cảm ơn thứ hai, chúng tôi xin gửi tới khoa Công nghệ thông tin và các thầy cô bộ môn tại trường Đại học Tôn Đức Thắng đã tạo điều kiện cho chúng tôi được lựa chọn và thực hiện đề tài này. Thông qua bài dự án công nghệ thông tin, chúng tôi đã có cơ hội tiếp thu nhiều kiến thức mới mẻ, từ đó trang bị cho chúng tôi những hành trang quý báu để tự tin bước vào môi trường làm việc thực tế ở các doanh nghiệp.

Một lần nữa, chúng tôi xin chân thành cảm ơn.

TP. Hồ Chí Minh, ngày 01 tháng 08 năm 2024

Tác giả

(Ký tên và ghi rõ họ tên)



Lý Tuân An



Phùng Phúc Hậu

PHIẾU ĐÁNH GIÁ CỦA GIẢNG VIÊN HƯỚNG DẪN

Tên giảng viên hướng dẫn:.....

Ý kiến nhận xét:

.....

.....

.....

Điểm tổng theo phiếu đánh giá rubrik:

TP. Hồ Chí Minh, ngày tháng năm 2024

Giảng viên hướng dẫn

(Ký tên và ghi rõ họ tên)

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH

TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Nhóm chúng tôi xin cam đoan đây là công trình nghiên cứu của riêng chúng tôi và được sự hướng dẫn khoa học của ThS. Mai Văn Mạnh. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong báo cáo còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào nhóm chúng tôi xin hoàn toàn chịu trách nhiệm về nội dung Báo cáo Dự án CNTT của mình. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do chúng tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 01 tháng 08 năm 2024

Tác giả

(Ký tên và ghi rõ họ tên)



Lý Tuân An



Phùng Phúc Hậu

TRIỂN KHAI DỊCH VỤ WEB CÓ KHẢ NĂNG MỞ RỘNG

SỬ DỤNG DOCKER SWARM

TÓM TẮT

Dự án này triển khai hệ thống quản lý nhà hàng theo cấu trúc Microservice, bao gồm các tính năng cơ bản (đăng nhập, đăng kí, đăng xuất, ...) đến các tính năng nâng cao (đặt đơn hàng, gửi mail thông báo, ...).

Dự án sử dụng các công nghệ như:

- RabbitMQ xử lý và truyền tải dữ liệu không đồng bộ giữa các Microservice.
- Redis dùng để cache dữ liệu tăng tốc độ truy xuất, giảm tải cho cơ sở dữ liệu chính.
- JWT xác thực và phân quyền người dùng, đảm bảo an toàn khi truy cập hệ thống.
- Docker đóng gói các Microservice vào container, đảm bảo mỗi dịch vụ hoạt động độc lập và nhất quán trên mọi môi trường.
- Docker Swarm triển khai và quản lý các Microservice, đảm bảo tính mở rộng và khả năng chịu lỗi của hệ thống.

Kết quả đạt được sau khi làm xong dự án là một hệ thống web quản lý nhà hàng được triển khai trên Docker Swarm, chạy trên các máy chủ EC2 của Amazon Web Services (AWS). Từ đó chúng tôi đánh giá được độ mở rộng, thích ứng và hiệu suất khi triển khai web lên Docker Swarm.

DEPLOYING SCALABLE WEB SERVICES WITH DOCKER SWARM

ABSTRACT

This project implements a restaurant management system using a microservice architecture, covering both basic features (login, registration, logout, etc.) and advanced features (order placement, email notifications, etc.).

The project utilizes the following technologies:

- RabbitMQ: Handles and transmits asynchronous data between microservices.
- Redis: Used for caching data to speed up retrieval and reduce the load on the main database.
- JWT (JSON Web Token): Authenticates and authorizes users, ensuring security when accessing the system.
- Docker: Packages microservices into containers, ensuring each service operates independently and consistently across all environments.
- Docker Swarm: Deploys and manages microservices, ensuring scalability and fault tolerance of the system.

The outcome of the project is a web-based restaurant management system deployed on Docker Swarm, running on Amazon Web Services (AWS) EC2 instances. This allowed us to evaluate the scalability, adaptability, and performance of deploying the web system on Docker Swarm.

MỤC LỤC

DANH MỤC HÌNH VẼ	ix
DANH MỤC BẢNG BIỂU	xi
DANH MỤC CÁC CHỮ VIẾT TẮT.....	xiii
CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI.....	1
1.1 Lý do chọn đề tài.....	1
1.2 Mục tiêu thực hiện đề tài.....	1
1.3 Phạm vi đề tài	2
1.4 Ý nghĩa đề tài	3
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT.....	4
2.1 Docker	4
2.2 Docker Swarm.....	6
2.3 Node.js & Express.js	11
2.4 MySQL & Sequelize	13
2.5 WebSocket & WebAPI	15
2.6 Microservice.....	17
2.7 RabbitMQ.....	19
2.8 Redis.....	23
2.9 JWT, Access Token & Refresh Token.....	25
2.10 AWS & EC2.....	28
CHƯƠNG 3. PHÂN TÍCH THIẾT KẾ	30
3.1 Phân tích quy trình kinh doanh	30
3.1.1 <i>Tìm hiểu quy trình thực tế.....</i>	30

3.1.2 Quy trình nghiệp vụ trong hệ thống.....	32
3.2 Phân tích yêu cầu	35
3.2.1 Đặc tả hệ thống	35
3.2.2 Đặc tả yêu cầu	36
3.2.3 Context Diagram.....	39
3.2.4 Xác định tác nhân và usecase	39
3.2.5 Phân tích Usecase.....	42
3.3 Thiết kế hệ thống.....	60
3.3.1 Kiến trúc hệ thống.....	60
3.3.2 Thiết kế cơ sở dữ liệu	64
3.3.3 Thiết kế API.....	69
CHƯƠNG 4. THỰC NGHIỆM	70
4.1 Triển khai	70
4.1.1 Sơ đồ triển khai tổng quát.....	70
4.1.2 Cài đặt Docker Swarm.....	72
4.1.3 Cấu hình dịch vụ	76
4.2 Kết quả thực nghiệm	82
4.2.1 Kiểm tra chức năng.....	82
4.2.2 Kiểm tra khả năng giám sát	83
4.2.3 Kiểm tra tính năng độ trễ thấp (Low Latency)	83
4.2.4 Kiểm tra tính năng cân bằng tải (Load Balancing).....	84
4.2.5 Kiểm tra khả năng mở rộng (Scalable).....	84
4.2.6 Kiểm tra khả năng khắc phục sự cố.....	88

4.3 Nhữn̄g mặt hạn ché̄ c̄a Docker Swarm.....	89
CHƯƠNG 5. TỔNG KẾT	90
5.1 Kết quả đạt được	90
5.2 Ưu điểm và Nhược điểm của dự án	91
5.3 Ưu điểm và Nhược điểm của Docker Swarm	92
5.4 So sánh gīa Docker Swarm và Kubernetes	93
5.5 Dùng Docker Swarm trong trường hợp nào?.....	94
5.5.1 <i>Khi nào n̄en d̄ng Docker Swarm?</i>	94
5.5.2 <i>Khi nào khōng n̄en d̄ng Docker Swarm?</i>	95
5.6 Thuận lợi và Khó khăn.....	96
5.7 Hướng phát triển trong tương lai	96
TÀI LIỆU THAM KHẢO	99

DANH MỤC HÌNH VẼ

Hình 2.1 Các thành phần chính của Docker.....	4
Hình 2.2 Cụm Docker Swarm.....	7
Hình 2.3 Nút trong Docker Swarm	8
Hình 2.4 Cách hoạt động của Express.js.....	12
Hình 2.5 Cách hoạt động của Sequelize	14
Hình 2.6 Cách hoạt động của WebSocket.....	16
Hình 2.7 Cách hoạt động của WebAPI	17
Hình 2.8 Ví dụ về mô hình Microservice.....	18
Hình 2.9 Cách hoạt động của RabbitMQ	20
Hình 2.10 Cách hoạt động của Redis	24
Hình 2.11 Các thành phần chính của JWT.....	25
Hình 2.12 Ví dụ Refresh Token	27
 Hình 3.1 Minh họa quy trình thực tế tại nhà hàng	30
Hình 3.2 Quy trình mở bàn phục vụ - Tạo mới đơn hàng.....	32
Hình 3.3 Quy trình thêm món ăn vào bàn phục vụ	32
Hình 3.4 Quy trình xử lý món ăn	33
Hình 3.5 Quy trình thiếu nguyên liệu món ăn.....	33
Hình 3.6 Quy trình chuyển bàn	34
Hình 3.7 Quy trình thanh toán.....	34
Hình 3.8 Sơ đồ ngũ cành.....	39
Hình 3.9 Sơ đồ usecase tổng quát	42
Hình 3.10 Kiến trúc tổng thể.....	63

Hình 3.11 Sơ đồ quan hệ tổng quát.....	69
Hình 4.1 Sơ đồ triển khai EC2	70
Hình 4.2 Sơ đồ triển khai cụm Docker Swarm	71
Hình 4.3 Khởi tạo EC2 instances	72
Hình 4.4 Cài đặt Docker trên từng node	73
Hình 4.5 Khởi tạo cụm Swarm.....	73
Hình 4.6 Thêm các manager và worker nodes vào swarm	74
Hình 4.7 Danh sách các manager và worker nodes	74
Hình 4.8 Phần cấu hình về mạng và khám phá dịch vụ	77
Hình 4.9 Phần cấu hình về phân bổ tài nguyên.....	78
Hình 4.10 Phần cấu hình về giám sát.....	79
Hình 4.11 Phần cấu hình về chiến lược triển khai	80
Hình 4.12 Phần cấu hình về biến môi trường	82
Hình 4.13 Thống kê và biểu đồ trong kịch bản ‘đông khách’ của 1 replicas	85
Hình 4.14 Thống kê và biểu đồ trong kịch bản ‘đông khách’ của 7 replicas	85
Hình 4.15 Thống kê và biểu đồ trong kịch bản ‘giờ cao điểm’ của 1 replicas	86
Hình 4.16 Thống kê và biểu đồ trong kịch bản ‘giờ cao điểm’ của 7 replicas	86
Hình 4.17 Thống kê và biểu đồ trong kịch bản ‘tăng đột biến’ của 1 replicas	87
Hình 4.18 Thống kê và biểu đồ trong kịch bản ‘tăng đột biến’ của 7 replicas	87

DANH MỤC BẢNG BIỂU

Bảng 3.1 Các tác nhân trong hệ thống	39
Bảng 3.2 Các usecase trong hệ thống	40
Bảng 3.3 Usecase đăng nhập.....	43
Bảng 3.4 Usecase đăng ký tài khoản	44
Bảng 3.5 Usecase kích hoạt tài khoản.....	45
Bảng 3.6 Usecase gửi mail kích hoạt tài khoản	46
Bảng 3.7 Usecase đăng xuất.....	47
Bảng 3.8 Usecase đổi mật khẩu	48
Bảng 3.9 Usecase tạo đơn hàng	49
Bảng 3.10 Usecase chuyển bàn	50
Bảng 3.11 Usecase thêm món ăn vào phục vụ.....	51
Bảng 3.12 Usecase thanh toán	52
Bảng 3.13 Usecase quản lý phiếu tính tiền	53
Bảng 3.14 Usecase quản lý nhân viên.....	54
Bảng 3.15 Usecase quản lý phụ phí	55
Bảng 3.16 Usecase quản lý món ăn	56
Bảng 3.17 Usecase quản lý bàn phục vụ.....	57
Bảng 3.18 Usecase xử lý món ăn	58
Bảng 3.19 Usecase quản lý nguyên liệu	59
Bảng 3.20 Mô tả thực thể Category	64
Bảng 3.21 Mô tả thực thể Item.....	64
Bảng 3.22 Mô tả thực thể Order.....	65

Bảng 3.23 Mô tả thực thể OrderItem	65
Bảng 3.24 Mô tả thực thể Surcharge.....	66
Bảng 3.25 Mô tả thực thể Payment.....	66
Bảng 3.26 Mô tả thực thể PaymentSurcharge.....	67
Bảng 3.27 Mô tả thực thể Table	67
Bảng 3.28 Mô tả thực thể Role	68
Bảng 3.29 Mô tả thực thể User	68
Bảng 5.1 So sánh giữa Docker Swarm và Kubernetes	93

DANH MỤC CÁC CHỮ VIẾT TẮT

JWT	Json Web Token
ORM	Object-Relational Mapping
AWS	Amazon Web Services
VPC	Virtual Private Cloud
RDS	Relational Database Service
ERD	Entity Relationship Diagram

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Lý do chọn đề tài

Ngày nay, các hệ thống phần mềm ngày càng trở nên phức tạp và yêu cầu khả năng mở rộng cao để đáp ứng nhu cầu của người dùng. Với sự phát triển của Internet và các ứng dụng web, việc xây dựng các hệ thống phân tán không chỉ là xu hướng mà còn là yêu cầu bắt buộc đối với các doanh nghiệp. Mô hình Microservice đã nổi lên như một giải pháp hiệu quả, cho phép các ứng dụng được chia thành các dịch vụ nhỏ, độc lập, có thể phát triển và triển khai một cách linh hoạt.

Trong bối cảnh đó, việc quản lý và điều phối các Microservices trở nên vô cùng quan trọng. Containerization với Docker đã mang lại một cách tiếp cận hiệu quả để đóng gói và triển khai các Microservices. Tuy nhiên, để quản lý hiệu quả các container, cần có những công cụ quản trị mạnh mẽ như Docker Swarm.

Chính vì lý do đó, chúng tôi đã chọn đề tài "Triển khai dịch vụ web có khả năng mở rộng sử dụng Docker Swarm". Đề tài này không chỉ giúp chúng tôi hiểu rõ hơn về các khái niệm liên quan đến Containerization và Microservice, mà còn cung cấp cho chúng tôi kỹ năng quản lý và điều phối các dịch vụ trong một hệ thống phân tán. Việc nắm vững Docker Swarm sẽ trang bị cho chúng tôi nền tảng vững chắc để đổi mới với những thách thức trong ngành công nghiệp phần mềm hiện đại, nơi mà khả năng mở rộng và tính sẵn sàng của hệ thống là những yếu tố then chốt cho sự thành công.

1.2 Mục tiêu thực hiện đề tài

Nghiên cứu và áp dụng Docker Swarm: Hiểu rõ và áp dụng Docker Swarm trong việc quản lý và điều phối container, bao gồm các khái niệm như service replication, rolling updates, và health checks.

Thiết lập và cấu hình môi trường Docker Swarm: Thiết lập một Docker Swarm cluster, có thể là trên máy tính cá nhân hoặc trên đám mây, để triển khai và kiểm thử

các dịch vụ web. Sử dụng thành thạo các lệnh Docker để xây dựng, chạy và quản lý container.

Triển khai dịch vụ web theo mô hình microservice: Triển khai các dịch vụ web theo mô hình microservice, đảm bảo khả năng mở rộng và tính khả dụng của hệ thống. Áp dụng các nguyên tắc phát triển dịch vụ web như HTTP, RESTful APIs và kiến trúc microservice.

Quản lý mạng và cấu hình dịch vụ: Hiểu và áp dụng các khái niệm mạng như service discovery, load balancing và routing trong môi trường phân tán. Cấu hình và tối ưu hóa các dịch vụ để đảm bảo hiệu suất và tính sẵn sàng.

Đánh giá và cải thiện: Đánh giá hiệu quả của việc triển khai dịch vụ web sử dụng Docker Swarm, từ đó áp dụng các phương pháp tốt nhất cho việc thiết kế, triển khai và bảo trì dịch vụ web, bao gồm phân bổ tài nguyên, giám sát và xử lý sự cố.

1.3 Phạm vi đề tài

Trong dự án này, chúng tôi tập trung vào việc phát triển và triển khai hệ thống quản lý nhà hàng, với mục tiêu đáp ứng các quy trình trực tiếp tại một nhà hàng. Cụ thể, chúng tôi sẽ:

- Triển khai các dịch vụ web chính: Phát triển và triển khai các dịch vụ web cần thiết cho hệ thống quản lý nhà hàng, bao gồm quản lý thực đơn, đặt bàn, xử lý đơn hàng, thanh toán, và các chức năng khác liên quan đến hoạt động của nhà hàng.
- Sử dụng Docker Swarm: Ứng dụng Docker Swarm để quản lý và điều phối các container cho các dịch vụ web này, nhằm đảm bảo tính khả dụng, mở rộng và hiệu suất của hệ thống.

Tuy nhiên, đề tài không bao gồm việc phát triển các giao diện người dùng chi tiết hoặc tích hợp với các hệ thống bên ngoài không thuộc phạm vi của hệ thống quản lý nhà hàng.

1.4 Ý nghĩa đề tài

Trước hết, về mặt học thuật, việc nghiên cứu và ứng dụng Docker Swarm trong triển khai dịch vụ web giúp mở rộng kiến thức về containerization và hệ thống phân tán. Đề tài này cung cấp cái nhìn sâu sắc về cách quản lý và điều phối các container trong môi trường sản xuất, đồng thời làm quen với các công nghệ và phương pháp hiện đại trong quản lý dịch vụ web. Điều này không chỉ nâng cao khả năng kỹ thuật mà còn tạo nền tảng vững chắc cho các nghiên cứu và dự án tương lai liên quan đến DevOps và phát triển phần mềm.

Về mặt thực tiễn, hệ thống quản lý nhà hàng được triển khai có khả năng mở rộng và đảm bảo tính sẵn sàng, giúp các nhà hàng quản lý hoạt động một cách hiệu quả hơn. Docker Swarm cung cấp giải pháp linh hoạt và mạnh mẽ để triển khai các dịch vụ web, đáp ứng nhu cầu mở rộng khi nhà hàng phát triển hoặc khi có sự thay đổi trong quy mô hoạt động. Điều này giúp tối ưu hóa hiệu suất hệ thống, giảm thiểu thời gian gián đoạn và nâng cao trải nghiệm khách hàng.

Ngoài ra, dự án cũng tạo điều kiện cho việc nghiên cứu và thực hành các kỹ thuật quản lý container và dịch vụ phân tán, góp phần phát triển kỹ năng và kinh nghiệm của sinh viên trong lĩnh vực công nghệ phần mềm. Từ đó, đề tài đóng góp vào việc cải tiến công nghệ quản lý nhà hàng, cung cấp nền tảng vững chắc cho các ứng dụng và dịch vụ tương lai, đồng thời đáp ứng nhu cầu của thị trường và nâng cao hiệu quả hoạt động của các cơ sở dịch vụ ăn uống.

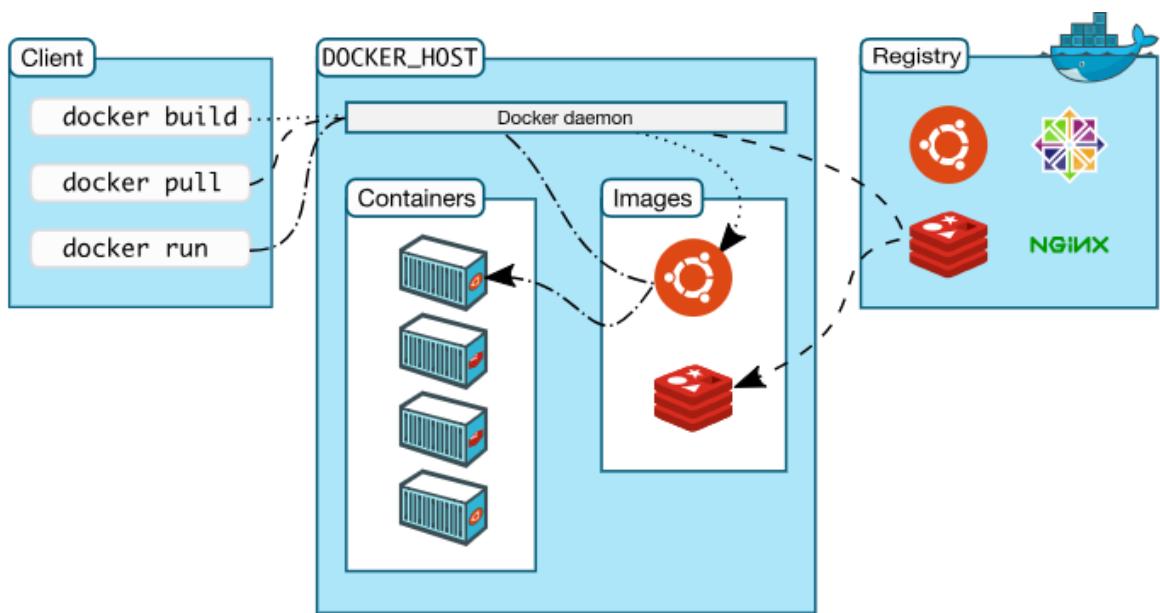
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Docker

2.1.1 Tổng quan về Docker

Docker là một nền tảng mã nguồn mở giúp tạo, triển khai và vận hành các ứng dụng thông qua việc sử dụng container. Container cung cấp môi trường cách ly, nhẹ và di động, đảm bảo rằng ứng dụng có thể chạy nhất quán từ máy phát triển đến môi trường sản xuất. Docker cho phép đóng gói ứng dụng cùng với tất cả các phụ thuộc vào một container, giúp đơn giản hóa quá trình phát triển, thử nghiệm và triển khai.

2.1.2 Các thành phần chính



Hình 2.1 Các thành phần chính của Docker

Docker Daemon: Chạy trên Docker Host và quản lý việc xây dựng, chạy và điều hành các container Docker. Docker Daemon lắng nghe các yêu cầu từ Docker Client và thực hiện các tác vụ tương ứng.

Docker Client: Giao diện dòng lệnh (CLI) hoặc giao diện người dùng đồ họa (GUI) để tương tác với Docker Daemon. Docker Client sử dụng Docker API để gửi các lệnh đến Docker Daemon.

Docker Images: Các template bất biến để tạo container. Mỗi image được xây dựng từ một Dockerfile và chứa tất cả các thành phần cần thiết để chạy ứng dụng, bao gồm mã nguồn, thư viện và các thiết lập môi trường. Docker Images có thể được chia sẻ và sử dụng lại, giúp duy trì tính nhất quán giữa các môi trường khác nhau.

Docker Containers: Các instance hoạt động của Docker Images. Containers cung cấp môi trường cách ly để chạy ứng dụng, đảm bảo rằng các ứng dụng hoạt động đồng nhất và tách biệt trong từng container, ngay cả khi chúng chia sẻ cùng một hệ điều hành.

Docker Networks: Cung cấp các phương thức kết nối giữa các container và với các dịch vụ bên ngoài. Docker hỗ trợ nhiều kiểu mạng, bao gồm bridge (cho kết nối giữa các container trên cùng một host), host (sử dụng mạng của host), và overlay (kết nối giữa các container trên nhiều host khác nhau).

Docker Volumes: Phương thức lưu trữ dữ liệu bền vững, không phụ thuộc vào vòng đời của container. Volumes cho phép chia sẻ dữ liệu giữa các container và giữa container với hệ thống máy chủ, đồng thời duy trì dữ liệu ngay cả khi container bị xóa hoặc khởi động lại.

Dockerfile: Tập tin chứa các chỉ thị để xây dựng Docker Images. Dockerfile định nghĩa các bước cần thiết để tải phần mềm, cài đặt các gói phụ thuộc và cấu hình ứng dụng, từ đó tạo ra một Docker Image có thể được sử dụng để khởi chạy container.

Docker Compose: Công cụ để định nghĩa và chạy nhiều container đồng thời. Docker Compose cho phép mô tả cấu hình của nhiều dịch vụ trong một tập tin "docker-compose.yml" và tự động hóa việc triển khai chúng, giúp quản lý các ứng dụng phức tạp với nhiều container dễ dàng hơn.

Docker Registry: Nơi lưu trữ Docker Images. Docker Hub là registry công cộng phổ biến nhất, ngoài ra còn có các registry riêng tư để lưu trữ images nội bộ.

2.1.3 Các tính năng nổi bật

Khả năng di động: Docker cho phép đóng gói ứng dụng và tất cả các phụ thuộc vào một container, đảm bảo rằng ứng dụng có thể chạy đồng nhất trên bất kỳ môi trường nào từ máy phát triển đến môi trường sản xuất. Điều này giúp dễ dàng di chuyển ứng dụng giữa các hệ thống mà không gặp phải các vấn đề tương thích.

Hiệu quả tài nguyên: Containers sử dụng ít tài nguyên hệ thống hơn so với các máy ảo truyền thống vì chúng chia sẻ cùng một kernel hệ điều hành. Điều này cải thiện hiệu suất hệ thống và giảm chi phí tài nguyên, giúp tối ưu hóa việc sử dụng phần cứng.

Triển khai nhanh chóng: Docker giúp tăng tốc quá trình triển khai ứng dụng bằng cách cho phép tạo và khởi động container trong vài giây. Điều này cho phép các nhà phát triển và quản trị viên hệ thống nhanh chóng triển khai các ứng dụng và dịch vụ, đồng thời dễ dàng phản ứng với các yêu cầu và thay đổi.

Quản lý phiên bản và cập nhật dễ dàng: Docker hỗ trợ quản lý phiên bản của các Docker Images và cho phép cập nhật ứng dụng một cách linh hoạt thông qua Dockerfile và Docker Compose. Người dùng có thể dễ dàng quay lại phiên bản trước đó nếu cần thiết và triển khai các bản cập nhật mới một cách hiệu quả.

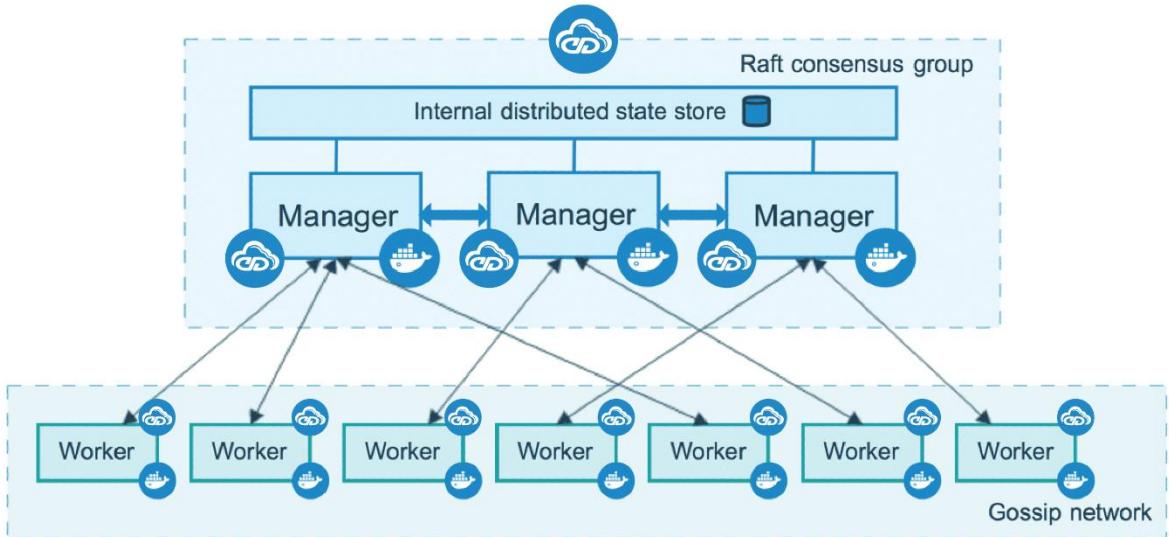
Tính nhất quán môi trường: Docker đảm bảo rằng ứng dụng chạy với cùng một cấu hình và phụ thuộc trên tất cả các môi trường, từ phát triển đến sản xuất. Điều này giúp giảm thiểu sự khác biệt giữa các môi trường và tránh các lỗi phát sinh do sự không tương thích.

2.2 Docker Swarm

2.2.1 Tổng quan về Docker Swarm

Docker Swarm là một công cụ mạnh mẽ cho việc quản lý cụm (cluster) và điều phối các container trên nhiều máy chủ Docker. Nó cho phép các dịch vụ container được triển khai, quản lý và mở rộng một cách dễ dàng trên nhiều nút trong một hệ thống phân tán. Docker Swarm được tích hợp trực tiếp vào Docker Engine từ phiên bản 1.12, giúp đơn giản hóa việc triển khai và quản lý các dịch vụ.

2.2.2 Cụm Docker Swarm (Swarm Cluster)



Hình 2.2 Cụm Docker Swarm

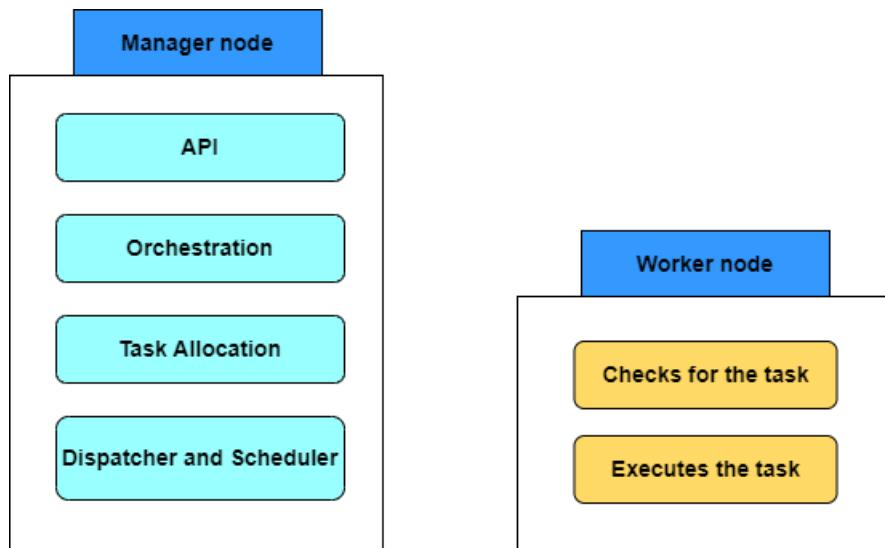
Một cụm Docker Swarm bao gồm nhiều máy chủ Docker chạy ở chế độ Swarm mode với các manager quản lý thành viên và phân chia công việc, và các worker thực thi các dịch vụ. Một máy chủ Docker có thể đóng vai trò là manager, worker, hoặc thực hiện cả hai vai trò cùng một lúc. Khi tạo dịch vụ, bạn định nghĩa trạng thái tối ưu – số lượng bản sao, tài nguyên mạng và lưu trữ, công dịch vụ, v.v. Docker sẽ duy trì trạng thái này. Nếu một worker node bị lỗi, Docker sẽ chuyển các tác vụ của nó sang các node khác. Một task là container đang chạy thuộc dịch vụ swarm và được swarm manager quản lý.

Một trong những ưu điểm chính của dịch vụ swarm so với các container độc lập là có thể thay đổi cấu hình của dịch vụ, bao gồm các mạng và volumes mà nó được kết nối, mà không cần phải khởi động lại dịch vụ thủ công. Docker sẽ cập nhật cấu hình, dừng các tác vụ dịch vụ có cấu hình lỗi thời, và tạo ra các tác vụ mới phù hợp với cấu hình mong muốn.

Khi Docker đang chạy ở chế độ Swarm mode, bạn vẫn có thể chạy các container độc lập trên bất kỳ Docker host nào tham gia vào swarm, cũng như các dịch vụ swarm. Một điểm khác biệt quan trọng giữa container độc lập và dịch vụ swarm là chỉ có swarm manager mới có thể quản lý swarm, trong khi các container độc lập

có thể được khởi động trên bất kỳ daemon nào. Các Docker daemon có thể tham gia vào một swarm với vai trò manager, worker, hoặc cả hai.

2.2.3 Nút trong Docker Swarm (Nodes)



Hình 2.3 Nút trong Docker Swarm

Một nút (node) là một phiên bản của Docker Engine tham gia vào cụm Swarm. Mỗi nút có thể là một máy tính vật lý hoặc máy chủ trên đám mây. Trong môi trường sản xuất, các nút thường được phân bố trên nhiều máy vật lý và đám mây để đảm bảo tính khả dụng và độ tin cậy.

Manager Node đảm nhận việc quản lý cụm, duy trì trạng thái mong muốn của các dịch vụ (service) và phân công nhiệm vụ (task) cho các worker node. Có thể có nhiều manager node trong một cluster để tăng tính sẵn sàng. Sử dụng thuật toán đồng thuận Raft (Raft Consensus Algorithm) để duy trì tính nhất quán trong toàn bộ cluster nhằm thực hiện việc bầu chọn một leader để thực hiện các nhiệm vụ điều phối.

Worker Node nhận và thực thi các nhiệm vụ từ manager node. Theo mặc định, manager node cũng sẽ chạy các dịch vụ như một worker node, tuy nhiên, ta có thể cấu hình để chúng chỉ thực hiện các nhiệm vụ quản lý và không chạy các dịch vụ, tức là chỉ đóng vai trò của một manager node mà thôi. Mỗi worker node có một agent chạy bên trong, agent này có nhiệm vụ báo cáo về các tác vụ được giao cho nó.

Worker node sẽ thông báo cho manager node về trạng thái hiện tại của các tác vụ được giao, giúp manager node duy trì trạng thái mong muốn của mỗi worker node.

2.2.4 Dịch vụ và nhiệm vụ (Services and Tasks)

Dịch vụ (Services) là cấu trúc trung tâm trong hệ thống Swarm, là điểm tương tác chính của người dùng với cụm. Khi tạo một dịch vụ, ta sẽ chỉ định Container Images và các lệnh để thực thi bên trong container đang chạy.

Nhiệm vụ (Tasks) là đơn vị lập lịch cơ bản của swarm. Một task chứa một Docker container và các lệnh để chạy bên trong container đó. Manager node sẽ phân công các task cho worker node dựa trên số lượng bản sao (replicas) được thiết lập từ trước. Khi một task đã được gán cho một node, nó không thể di chuyển sang node khác. Task chỉ có thể chạy trên node đã được chỉ định hoặc thất bại.

2.2.5 Cân bằng tải (Load Balancing)

Docker Swarm cung cấp khả năng cân bằng tải thông qua hai hình thức:

- *Cân bằng tải đầu vào:* Được sử dụng để công khai các dịch vụ ra bên ngoài cụm. Swarm manager có thể tự động gán một cổng cho dịch vụ hoặc bạn có thể chỉ định một cổng cụ thể. Mọi nút trong cụm đều có thể định tuyến kết nối đầu vào đến một phiên bản nhiệm vụ đang chạy.
- *Cân bằng tải nội bộ:* Sử dụng DNS nội bộ của Swarm để phân phối các yêu cầu giữa các dịch vụ trong cụm, dựa trên tên DNS của dịch vụ.

2.2.6 Các tính năng nổi bật

Quản lý cụm tích hợp với Docker Engine: Docker Swarm tích hợp trực tiếp với Docker Engine, cho phép tạo một cụm Docker Engine và triển khai các dịch vụ ứng dụng mà không cần phần mềm điều phối bổ sung.

Thiết kế phi tập trung: Với thiết kế này, ta có thể xây dựng toàn bộ swarm từ một hình ảnh đĩa duy nhất. Điều này có nghĩa là chỉ cần một hình ảnh hệ điều hành hoặc một hình ảnh Docker để triển khai cả các node manager và worker. Việc này

giúp đơn giản hóa quá trình triển khai, vì không cần phải tạo và cấu hình các node khác nhau từ các hình ảnh riêng biệt.

Mô hình dịch vụ khai báo: Docker Engine sử dụng cách tiếp cận khai báo, cho phép định nghĩa trạng thái mong muốn của các dịch vụ trong ngăn xếp ứng dụng. Ví dụ, ta có thể mô tả một ứng dụng bao gồm dịch vụ frontend web với các dịch vụ hàng đợi tin nhắn và backend cơ sở dữ liệu.

Khả năng mở rộng: Có thể khai báo số lượng task muốn chạy cho mỗi dịch vụ. Khi mở rộng hoặc thu hẹp, swarm manager tự động điều chỉnh bằng cách thêm hoặc loại bỏ các task để duy trì trạng thái mong muốn.

Đổi chiều trạng thái mong muốn: Swarm manager liên tục theo dõi trạng thái cụm và điều chỉnh những khác biệt giữa trạng thái thực tế và trạng thái mong muốn. Ví dụ, nếu một máy worker gặp sự cố và hai bản sao của container bị mất, swarm manager sẽ tạo lại hai bản sao mới để thay thế.

Mạng đa máy chủ: Có thể chỉ định một mạng overlay cho các dịch vụ của mình. Swarm manager tự động gán địa chỉ cho các container trên mạng overlay khi khởi tạo hoặc cập nhật ứng dụng.

Khám phá dịch vụ (Service discovery): Swarm manager gán mỗi dịch vụ trong swarm một tên DNS duy nhất và cân bằng tải các container đang chạy. Ta có thể truy vấn mọi container trong swarm thông qua máy chủ DNS tích hợp.

Cân bằng tải (Load balancing): Có thể công khai các cổng dịch vụ ra một bộ cân bằng tải bên ngoài. Bên trong, swarm cho phép chỉ định cách phân phối các container dịch vụ giữa các node.

Bảo mật mặc định: Mỗi node trong swarm bắt buộc phải sử dụng xác thực và mã hóa TLS để bảo mật giao tiếp giữa nó và các node khác. Ta có thể sử dụng chứng chỉ gốc tự ký hoặc từ một CA gốc tùy chỉnh.

Cập Nhật Rolling (Rolling updates): Khi triển khai dịch vụ, ta có thể áp dụng các cập nhật dịch vụ dần dần đến các node khác nhau. Swarm manager cho phép ta kiểm soát thời gian trễ giữa các đợt triển khai dịch vụ. Nếu có bất kỳ vấn đề nào xảy ra, bạn có thể quay lại phiên bản trước đó của dịch vụ.

2.3 Node.js & Express.js

2.3.1 Giới thiệu về Node.js

Node.js là một nền tảng (platform) mã nguồn mở, đa nền tảng, được xây dựng trên công cụ JavaScript V8 của Google. Node.js cho phép thực thi mã JavaScript trên phía server, thay vì chỉ trên phía client như các trình duyệt web truyền thống.

Các đặc điểm chính của Node.js

- *Hiệu suất cao:* Node.js sử dụng event-driven, non-blocking I/O model, giúp xử lý các tác vụ đồng thời một cách hiệu quả và tối ưu hóa hiệu suất.
- *Single-threaded nhưng có thể mở rộng:* Node.js chạy trên một luồng đơn (single-threaded) nhưng có thể xử lý nhiều kết nối đồng thời nhờ vào kiến trúc event-driven.
- *Môi trường JavaScript trên server:* Node.js cho phép sử dụng JavaScript để viết các ứng dụng server-side, thống nhất ngôn ngữ lập trình trên cả phía client và server.
- *Hệ sinh thái phong phú:* Node.js có một hệ sinh thái module rất phong phú và mạnh mẽ, với npm (Node Package Manager) là hệ thống quản lý package lớn nhất thế giới.

Các ứng dụng của Node.js

- *Ứng dụng web:* Phù hợp cho việc xây dựng các ứng dụng web thời gian thực như chat, game online, và các ứng dụng truyền thông.
- *API services:* Thường được sử dụng để xây dựng các RESTful API và microservices.
- *Ứng dụng mạng:* Tạo các ứng dụng mạng như proxy, load balancer, và các công cụ mạng khác.

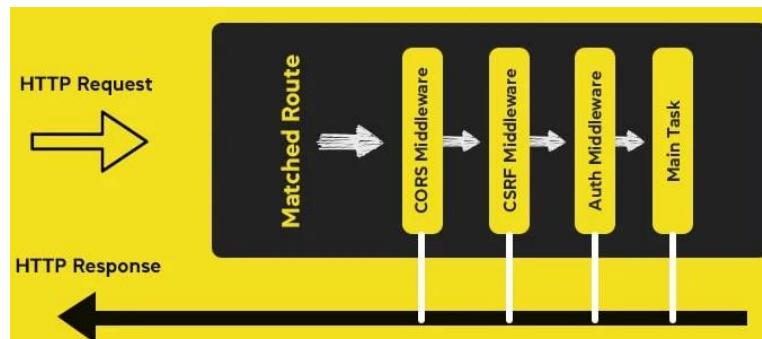
2.3.2 Giới thiệu về Express.js

Express.js là một framework web nhẹ và linh hoạt cho Node.js, cung cấp một bộ công cụ mạnh mẽ để xây dựng các ứng dụng web và API nhanh chóng và dễ dàng hơn. Đây là một trong những framework web phổ biến nhất cho Node.js nhờ vào sự đơn giản, linh hoạt và hiệu suất cao.

Các đặc điểm chính của Express.js

- *Nhẹ và linh hoạt*: Express.js là một framework minimalistic, cung cấp các tính năng cơ bản và cho phép mở rộng theo nhu cầu của ứng dụng.
- *Routing mạnh mẽ*: Cung cấp hệ thống định tuyến linh hoạt, giúp quản lý các route và điều hướng các yêu cầu HTTP một cách dễ dàng.
- *Middleware*: Express.js hỗ trợ sử dụng middleware, cho phép thêm các tính năng như xử lý lỗi, phân tích dữ liệu request, và quản lý xác thực một cách dễ dàng.
- *Tích hợp dễ dàng*: Dễ dàng tích hợp với các cơ sở dữ liệu, công cụ xác thực, và các thư viện khác, giúp xây dựng các ứng dụng web hoàn chỉnh và mạnh mẽ.
- *Hỗ trợ RESTful*: Express.js là lựa chọn phổ biến để xây dựng các RESTful API, nhờ vào khả năng xử lý các phương thức HTTP và quản lý route hiệu quả.

Cách hoạt động của Express.js



Hình 2.4 Cách hoạt động của Express.js

- *Thiết lập server:* Tạo một server Node.js sử dụng Express.js và lắng nghe các yêu cầu HTTP từ client.
- *Định nghĩa route:* Sử dụng các phương thức HTTP như GET, POST, PUT, DELETE để định nghĩa các route và xử lý các yêu cầu tương ứng.
- *Sử dụng middleware:* Thêm các middleware để xử lý dữ liệu request, xác thực người dùng, quản lý lỗi, và thực hiện các tác vụ khác trước khi gửi phản hồi tới client.
- *Gửi phản hồi:* Xử lý yêu cầu và gửi phản hồi về cho client dưới dạng JSON, HTML, hoặc các định dạng khác.

2.4 MySQL & Sequelize

2.4.1 Giới thiệu về MySQL

MySQL là một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS - Relational Database Management System) mã nguồn mở, được phát triển và duy trì bởi Oracle Corporation. Nó được thiết kế để quản lý dữ liệu trong các bảng, cho phép truy vấn và xử lý dữ liệu một cách hiệu quả.

Các đặc điểm chính của MySQL

- *Hiệu suất cao:* MySQL được tối ưu hóa cho hiệu suất cao và xử lý hàng triệu truy vấn mỗi ngày trong các hệ thống lớn.
- *Khả năng mở rộng:* MySQL có khả năng mở rộng từ các ứng dụng nhỏ đến các hệ thống doanh nghiệp lớn với khối lượng dữ liệu khổng lồ.
- *Bảo mật:* MySQL cung cấp các tính năng bảo mật mạnh mẽ, bao gồm xác thực người dùng, mã hóa dữ liệu và quản lý quyền truy cập.
- *Hỗ trợ đa nền tảng:* MySQL có thể chạy trên nhiều hệ điều hành khác nhau như Windows, Linux, và macOS.

2.4.2 Giới thiệu về Sequelize

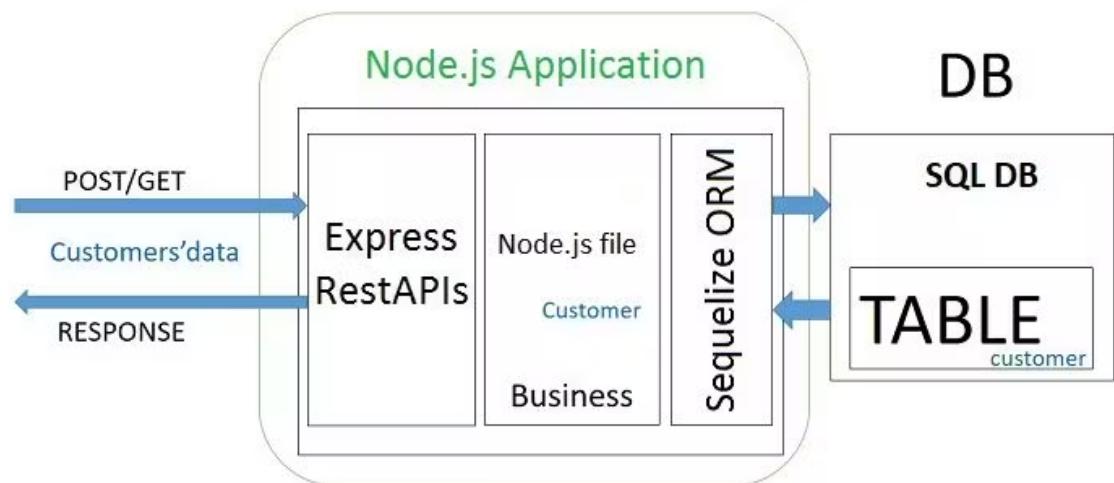
Sequelize là một ORM (Object-Relational Mapping) cho Node.js, giúp kết nối ứng dụng với cơ sở dữ liệu MySQL, PostgreSQL, SQLite và Microsoft SQL Server.

Nó cung cấp một API đơn giản để thao tác với cơ sở dữ liệu, đồng thời hỗ trợ các tính năng như migrations và seeding.

Các tính năng chính của Sequelize

- *Định nghĩa mô hình:* Sequelize cho phép định nghĩa các mô hình dữ liệu bằng cách sử dụng các lớp (classes) và thuộc tính (attributes), giúp quản lý và tương tác với dữ liệu một cách có cấu trúc.
- *Quản lý kết nối:* Cung cấp các phương thức để kết nối, đồng bộ hóa và quản lý cơ sở dữ liệu.
- *Tạo và thực hiện truy vấn:* Cung cấp API để tạo và thực hiện các truy vấn SQL mà không cần viết mã SQL thủ công.
- *Migrations:* Hỗ trợ di chuyển cơ sở dữ liệu bằng cách tạo và thực hiện các thay đổi cấu trúc cơ sở dữ liệu theo thời gian.
- *Seeding:* Cho phép thêm dữ liệu mẫu vào cơ sở dữ liệu để phục vụ cho mục đích phát triển và thử nghiệm.
- *Quan hệ giữa các mô hình:* Hỗ trợ định nghĩa và quản lý các quan hệ giữa các mô hình dữ liệu như one-to-one, one-to-many và many-to-many.

Cách hoạt động của Sequelize



Hình 2.5 Cách hoạt động của Sequelize

- *Kết nối cơ sở dữ liệu:* Thiết lập kết nối tới cơ sở dữ liệu thông qua các thông tin cấu hình như hostname, username, password và database.
- *Định nghĩa mô hình:* Tạo các mô hình dữ liệu bằng cách sử dụng các lớp và thuộc tính, định nghĩa các quan hệ giữa các mô hình.
- *Thao tác với dữ liệu:* Sử dụng các phương thức của Sequelize để thực hiện các thao tác CRUD (Create, Read, Update, Delete) và các truy vấn phức tạp khác.
- *Quản lý schema:* Sử dụng migrations để quản lý các thay đổi trong cấu trúc cơ sở dữ liệu và seeding để thêm dữ liệu mẫu.

2.5 WebSocket & WebAPI

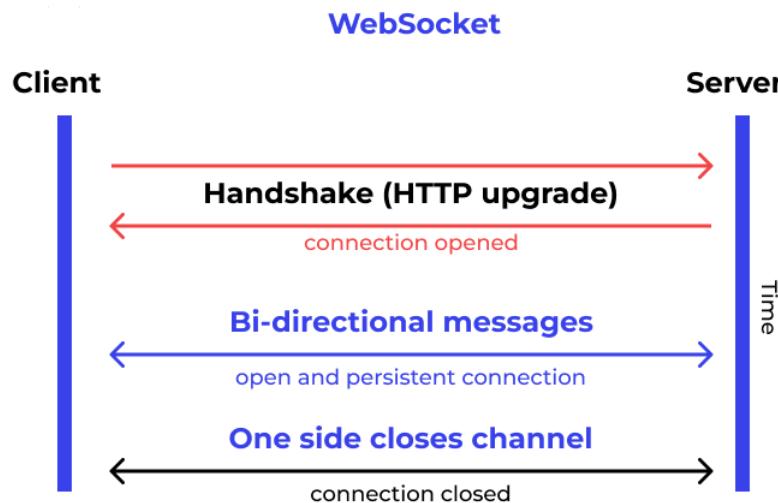
2.5.1 Giới thiệu về WebSocket

WebSocket là một giao thức giao tiếp hai chiều được thiết kế để thiết lập các kết nối liên tục giữa client và server thông qua một kết nối TCP duy nhất. WebSocket cho phép truyền dữ liệu nhanh chóng và hiệu quả, giảm thiểu độ trễ so với các phương pháp truyền thống như HTTP.

Các đặc điểm chính của WebSocket

- *Giao tiếp hai chiều:* Cả client và server đều có thể gửi và nhận dữ liệu đồng thời, không cần phải chờ đợi lẫn nhau.
- *Kết nối liên tục:* Kết nối WebSocket được duy trì cho đến khi client hoặc server đóng kết nối, giúp giảm thiểu overhead do việc thiết lập và phá vỡ kết nối liên tục.
- *Hiệu suất cao:* WebSocket sử dụng ít băng thông hơn do không cần gửi các tiêu đề HTTP lặp đi lặp lại.
- *Thích hợp cho ứng dụng thời gian thực:* WebSocket rất phù hợp cho các ứng dụng yêu cầu cập nhật dữ liệu thời gian thực như trò chuyện trực tuyến, trò chơi trực tuyến, và các ứng dụng theo dõi thị trường chứng khoán.

Cách hoạt động của WebSocket



Hình 2.6 Cách hoạt động của WebSocket

- **Handshake:** Kết nối WebSocket bắt đầu bằng một yêu cầu HTTP từ client tới server để thiết lập kết nối WebSocket. Server phản hồi bằng cách chấp nhận kết nối này.
- **Giao tiếp:** Sau khi kết nối được thiết lập, dữ liệu có thể được gửi qua lại giữa client và server mà không cần thiết lập lại kết nối.
- **Đóng kết nối:** Kết nối có thể được đóng bởi client hoặc server khi không còn cần thiết nữa.

2.5.2 Giới thiệu về WebAPI

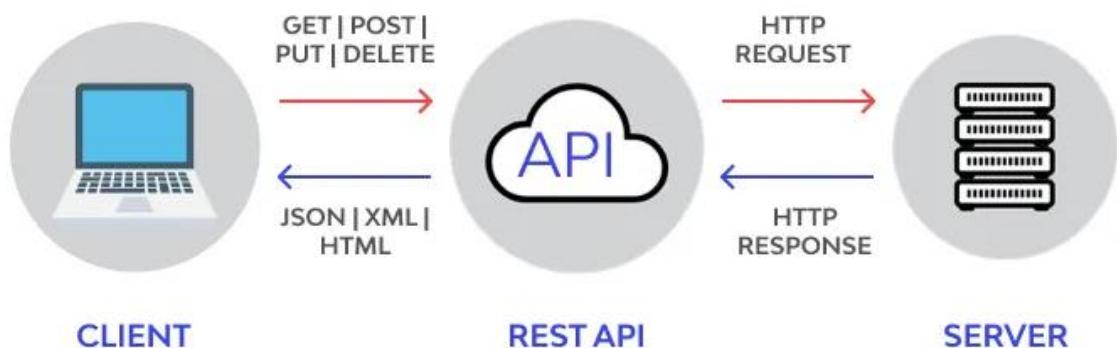
WebAPI (Application Programming Interface) là một tập hợp các giao diện và phương thức được cung cấp qua giao thức HTTP(S) để cho phép các ứng dụng tương tác với nhau. WebAPI cho phép các ứng dụng khác nhau trao đổi dữ liệu và thực hiện các chức năng thông qua các yêu cầu HTTP.

Các đặc điểm chính của WebAPI

- **Giao tiếp thông qua HTTP:** WebAPI sử dụng các phương thức HTTP như GET, POST, PUT, DELETE để thao tác với tài nguyên.
- **Tương thích đa nền tảng:** WebAPI có thể được sử dụng trên nhiều nền tảng khác nhau, bao gồm web, di động và các ứng dụng máy tính để bàn.

- *Dữ liệu trả về:* Dữ liệu được trao đổi thông qua WebAPI thường được định dạng dưới dạng JSON hoặc XML, giúp dễ dàng phân tích và xử lý.
- *Mở rộng và tích hợp:* WebAPI cho phép mở rộng chức năng của ứng dụng bằng cách tích hợp với các dịch vụ bên ngoài như thanh toán, bản đồ, và mạng xã hội.

Cách hoạt động của WebAPI



Hình 2.7 Cách hoạt động của WebAPI

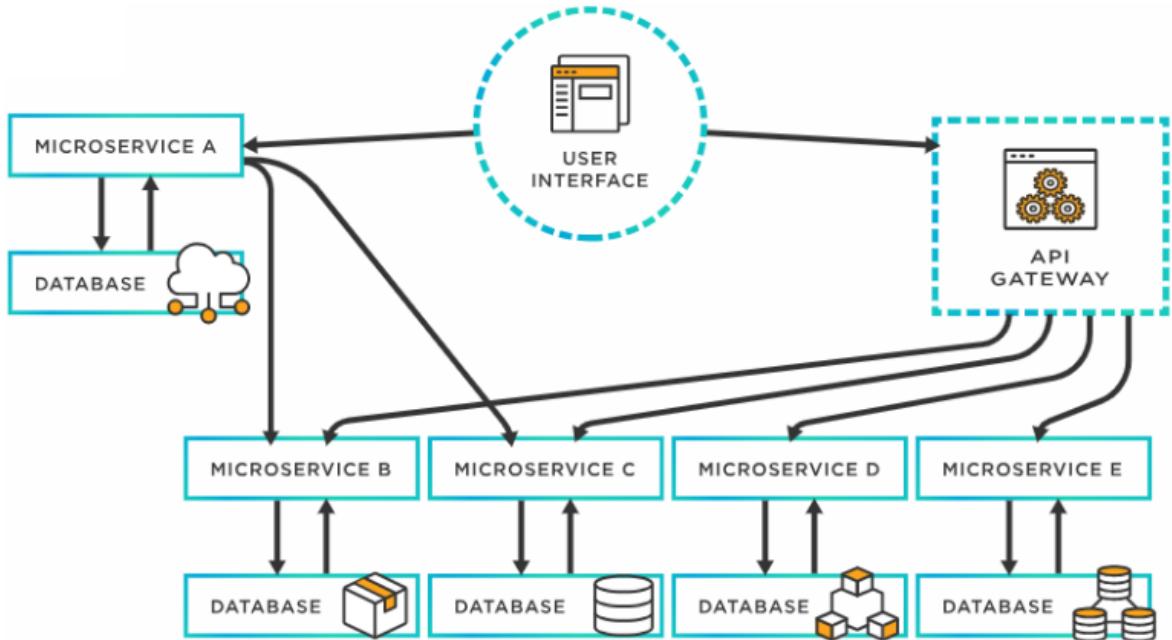
- *Yêu cầu:* Client gửi một yêu cầu HTTP tới một endpoint của API để thực hiện một hành động hoặc lấy dữ liệu.
- *Xử lý:* Server nhận yêu cầu, thực hiện các xử lý cần thiết (như truy vấn cơ sở dữ liệu, tính toán) và chuẩn bị dữ liệu phản hồi.
- *Phản hồi:* Server gửi lại một phản hồi HTTP cho client, thường chứa mã trạng thái và dữ liệu yêu cầu.

2.6 Microservice

2.6.1 Giới thiệu về Microservice

Microservices là một kiến trúc phần mềm trong đó một ứng dụng lớn được chia thành nhiều dịch vụ nhỏ, độc lập, và có thể triển khai riêng biệt. Mỗi dịch vụ trong kiến trúc microservice chịu trách nhiệm cho một chức năng cụ thể và có thể giao tiếp với các dịch vụ khác thông qua các giao thức nhẹ như HTTP/HTTPS hoặc

message queue. Mỗi microservice có thể được phát triển bằng công nghệ và ngôn ngữ lập trình khác nhau, và thường có cơ sở dữ liệu riêng để quản lý dữ liệu.



Hình 2.8 Ví dụ về mô hình Microservice

Các đặc điểm chính của Microservice

- *Độc lập và tự chủ*: Mỗi dịch vụ có thể được phát triển, triển khai, và mở rộng độc lập mà không ảnh hưởng đến các dịch vụ khác.
- *Tập trung vào một chức năng cụ thể*: Mỗi microservice đảm nhận một phần chức năng rõ ràng của ứng dụng, giúp dễ dàng quản lý và phát triển.
- *Giao tiếp nhẹ*: Các dịch vụ giao tiếp với nhau thông qua các giao thức nhẹ như RESTful API hoặc message queue, giảm thiểu độ phức tạp trong việc tích hợp.
- *Khả năng mở rộng*: Dễ dàng mở rộng các phần của ứng dụng bằng cách nhân bản các dịch vụ cần thiết mà không ảnh hưởng đến toàn bộ hệ thống.
- *Độc lập về công nghệ*: Mỗi microservice có thể được viết bằng một ngôn ngữ lập trình và công nghệ khác nhau, phù hợp với yêu cầu cụ thể của từng dịch vụ.

2.6.2 Giao tiếp giữa các Microservices

Cách các dịch vụ giao tiếp, bao gồm:

- *API:* Các microservices thường giao tiếp với nhau qua API, thường là HTTP/REST hoặc gRPC. API cung cấp các điểm cuối (endpoints) mà các dịch vụ có thể gọi để thực hiện các thao tác hoặc lấy dữ liệu.
- *Message Queues:* Message queues như RabbitMQ hoặc Kafka được sử dụng để giao tiếp bất đồng bộ giữa các dịch vụ. Điều này cho phép gửi và nhận thông điệp mà không cần thiết lập kết nối trực tiếp.

Các mô hình giao tiếp, bao gồm:

- *Synchronous:* Trong mô hình đồng bộ, các dịch vụ giao tiếp qua API, và các yêu cầu cần phải được xử lý ngay lập tức. Ví dụ, một dịch vụ A gửi yêu cầu đến dịch vụ B và chờ đợi phản hồi trước khi tiếp tục.
- *Asynchronous:* Trong mô hình bất đồng bộ, các dịch vụ giao tiếp qua message queues. Các dịch vụ gửi thông điệp vào queue và không cần chờ đợi phản hồi ngay lập tức. Điều này giúp cải thiện khả năng mở rộng và giảm độ trễ, vì các dịch vụ có thể xử lý thông điệp khi có sẵn tài nguyên.

2.7 RabbitMQ

2.7.1 Giới thiệu về RabbitMQ

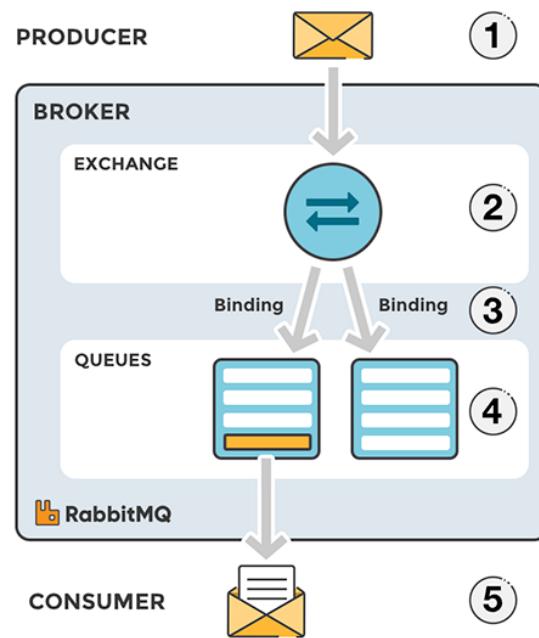
RabbitMQ là một hệ thống quản lý hàng đợi tin nhắn mã nguồn mở (message broker), được thiết kế để quản lý và điều phối các thông điệp giữa các hệ thống và ứng dụng. RabbitMQ dựa trên giao thức AMQP (Advanced Message Queuing Protocol), nhưng cũng hỗ trợ nhiều giao thức khác như MQTT và STOMP, giúp linh hoạt trong việc tích hợp với nhiều loại ứng dụng khác nhau.

Các đặc điểm chính của RabbitMQ

- *Giao thức AMQP:* Hỗ trợ AMQP, một giao thức chuẩn cho message-oriented middleware, đảm bảo tính tương thích và tính di động của ứng dụng.

- *Đa giao thức:* Ngoài AMQP, RabbitMQ còn hỗ trợ các giao thức khác như MQTT và STOMP, giúp tích hợp dễ dàng với các ứng dụng IoT và các hệ thống khác.
- *Độ tin cậy cao:* RabbitMQ đảm bảo độ tin cậy và an toàn cho thông điệp, với các tính năng như xác nhận thông điệp (message acknowledgment), duy trì thông điệp (message persistence), và chính sách quản lý hàng đợi (queue policies).
- *Quản lý hàng đợi:* Hỗ trợ các cơ chế quản lý hàng đợi linh hoạt, bao gồm việc tạo và xóa hàng đợi động, điều chỉnh kích thước hàng đợi, và cấu hình ưu tiên hàng đợi.
- *Khả năng mở rộng:* Có khả năng mở rộng ngang (horizontal scaling) bằng cách thêm các nút (node) vào cluster, giúp tăng khả năng xử lý và độ sẵn sàng của hệ thống.
- *Giao diện quản lý:* Cung cấp giao diện quản lý web và API quản lý, giúp theo dõi và quản lý các hàng đợi, kết nối, và các nút trong cluster một cách dễ dàng.

Cách hoạt động của RabbitMQ



Hình 2.9 Cách hoạt động của RabbitMQ

- *Producers và Consumers:* Producers gửi thông điệp tới RabbitMQ, nơi các thông điệp được lưu trữ trong các hàng đợi (queues). Consumers lấy thông điệp từ hàng đợi và xử lý chúng.
- *Exchanges và Queues:* RabbitMQ sử dụng exchanges để định tuyến thông điệp tới các hàng đợi. Có nhiều loại exchanges khác nhau như direct, topic, fanout, và headers, mỗi loại có cách định tuyến thông điệp riêng.
- *Bindings:* Bindings xác định quy tắc định tuyến giữa exchanges và queues, quyết định thông điệp nào sẽ được gửi tới hàng đợi nào.
- *Xác nhận thông điệp:* RabbitMQ hỗ trợ cơ chế xác nhận thông điệp, đảm bảo rằng thông điệp đã được nhận và xử lý thành công bởi consumer trước khi xóa khỏi hàng đợi.
- *Quản lý và giám sát:* Sử dụng giao diện quản lý web hoặc API để theo dõi và quản lý trạng thái của các hàng đợi, exchanges, bindings, và các nút trong hệ thống.

Các ứng dụng của RabbitMQ

- *Phân tán công việc:* RabbitMQ có thể được sử dụng để phân tán các tác vụ công việc giữa các worker, giúp cải thiện hiệu suất và khả năng chịu tải của hệ thống.
- *Xử lý thông điệp:* Sử dụng RabbitMQ để xử lý và định tuyến thông điệp giữa các hệ thống, đảm bảo tính nhất quán và liên tục của dữ liệu.
- *Ứng dụng thời gian thực:* RabbitMQ có thể được sử dụng trong các ứng dụng thời gian thực như chat, thông báo, và các dịch vụ truyền thông.
- *Tích hợp hệ thống:* Sử dụng RabbitMQ để tích hợp các hệ thống khác nhau, giúp trao đổi dữ liệu và thông điệp một cách hiệu quả và tin cậy.

2.7.2 Các mô hình giao tiếp

RabbitMQ hỗ trợ nhiều mô hình giao tiếp khác nhau để định tuyến và quản lý thông điệp giữa các thành phần của hệ thống. Các mô hình này giúp tối ưu hóa việc truyền tải thông điệp và đảm bảo tính linh hoạt trong cách các ứng dụng tương tác với nhau.

Các mô hình giao tiếp chính của RabbitMQ

- *Point-to-Point (Direct Exchange)*: Trong mô hình này, thông điệp được gửi từ một producer đến một consumer qua một hàng đợi (queue) cụ thể. Phù hợp cho các tác vụ yêu cầu sự xử lý đơn lẻ hoặc phân công công việc cụ thể giữa các worker.
- *Publish/Subscribe (Fanout Exchange)*: Thông điệp được gửi từ một producer đến tất cả các consumers đã đăng ký nhận thông điệp từ một exchange cụ thể. Mỗi hàng đợi được kết nối với exchange này sẽ nhận một bản sao của thông điệp. Phù hợp cho các hệ thống thông báo, nơi một thông điệp cần được phát tới nhiều người nhận cùng một lúc.
- *Routing (Topic Exchange)*: Thông điệp được gửi dựa trên các khóa routing (routing keys). Các queues sẽ nhận thông điệp dựa trên sự phù hợp của các khóa routing với các pattern định trước. Phù hợp cho các hệ thống mà thông điệp cần được định tuyến một cách linh hoạt dựa trên các tiêu chí khác nhau.
- *Header Exchange*: Thông điệp được định tuyến dựa trên các thuộc tính (headers) thay vì các khóa routing. Các thuộc tính này được kiểm tra để xác định hàng đợi nhận thông điệp. Phù hợp cho các trường hợp cần kiểm soát chi tiết các thuộc tính của thông điệp để định tuyến.
- *Request/Reply*: Đây là mô hình mà một producer gửi yêu cầu và chờ đợi một phản hồi từ consumer. Thông điệp yêu cầu được gửi đến một queue và consumer sẽ xử lý và gửi phản hồi lại qua một queue khác. Phù hợp cho các dịch vụ yêu cầu phản hồi ngay lập tức sau khi xử lý yêu cầu.

Cách sử dụng các mô hình giao tiếp trong RabbitMQ

- *Cấu hình Exchanges và Queues:* Sử dụng các lệnh hoặc giao diện quản lý của RabbitMQ để tạo và cấu hình các exchanges và queues phù hợp với mô hình giao tiếp.
- *Bindings:* Định nghĩa các bindings giữa exchanges và queues để thiết lập các quy tắc định tuyến thông điệp.
- *Producers và Consumers:* Cấu hình các ứng dụng để gửi thông điệp tới exchanges và nhận thông điệp từ các queues dựa trên mô hình giao tiếp đã chọn.

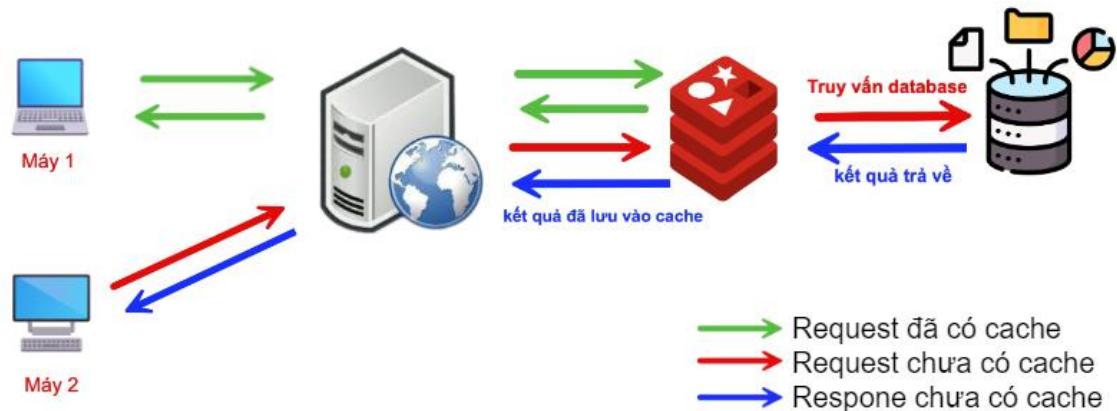
2.8 Redis

Redis là một hệ quản trị cơ sở dữ liệu dạng key-value, lưu trữ dữ liệu trong bộ nhớ (in-memory), cho phép truy xuất dữ liệu với tốc độ cực nhanh nhờ việc giữ tất cả dữ liệu trong bộ nhớ RAM. Redis được sử dụng rộng rãi trong các ứng dụng yêu cầu hiệu suất cao và thời gian phản hồi nhanh. Ngoài việc lưu trữ các cặp key-value đơn giản, Redis còn hỗ trợ nhiều cấu trúc dữ liệu phức tạp như strings, hashes, lists, sets, sorted sets, bitmaps, hyperloglogs và các geospatial indexes.

Các đặc điểm chính của Redis

- *In-memory:* Dữ liệu được lưu trữ hoàn toàn trong bộ nhớ, giúp truy xuất và xử lý dữ liệu với tốc độ cực nhanh.
- *Persistent Storage:* Redis hỗ trợ các cơ chế lưu trữ dữ liệu xuống đĩa để đảm bảo tính bền vững của dữ liệu.
- *Transactions:* Hỗ trợ các thao tác transaction với các lệnh MULTI, EXEC, DISCARD và WATCH.
- *Pub/Sub:* Hỗ trợ cơ chế publish/subscribe để xây dựng các ứng dụng real-time.

Cách hoạt động của Redis



Hình 2.10 Cách hoạt động của Redis

- *Client Request*: Khi một client gửi yêu cầu đến Redis, Redis nhận yêu cầu qua giao thức TCP.
- *Command Parsing*: Redis phân tích lệnh và xác định loại dữ liệu và thao tác cần thực hiện.
- *Data Processing*: Redis thực hiện các thao tác trên dữ liệu trong bộ nhớ hoặc ghi dữ liệu vào đĩa nếu cần.
- *Response*: Redis gửi kết quả của thao tác trở lại client.

Các ứng dụng của Redis

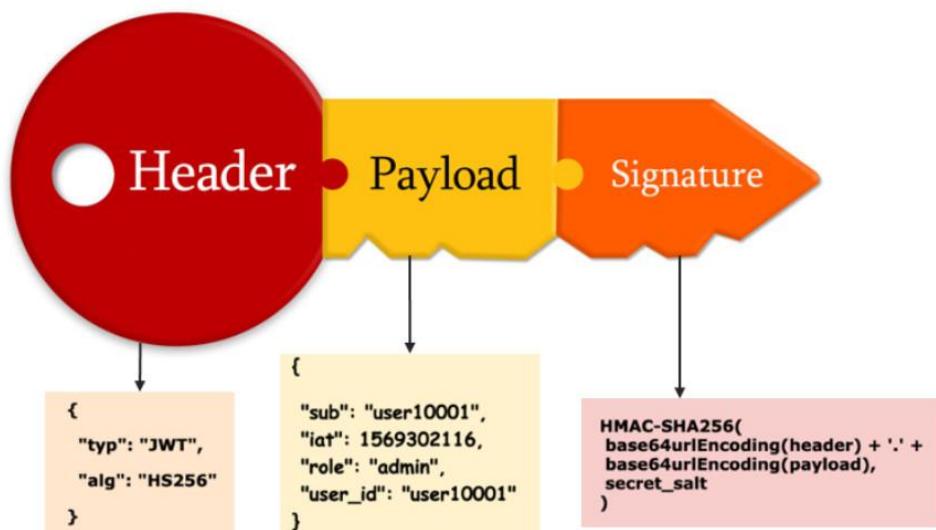
- *Caching*: Redis thường được sử dụng để làm bộ nhớ đệm (cache) nhằm giảm tải cho cơ sở dữ liệu chính và tăng tốc độ truy xuất dữ liệu.
- *Session Storage*: Lưu trữ phiên làm việc (session) của người dùng trong các ứng dụng web.
- *Real-time Analytics*: Xử lý và lưu trữ các dữ liệu phân tích thời gian thực.
- *Messaging*: Sử dụng cơ chế Pub/Sub để xây dựng các hệ thống thông báo và chat real-time.
- *Leaderboards/Counters*: Lưu trữ và quản lý các bảng xếp hạng, bộ đếm.
- *Data Streams*: Xử lý và quản lý các luồng dữ liệu.

2.9 JWT, Access Token & Refresh Token

2.9.1 Giới thiệu về JWT (JSON Web Token)

JWT là một tiêu chuẩn mở (RFC 7519) để truyền thông tin giữa các phần của hệ thống một cách an toàn và tin cậy. JWT thường được sử dụng trong các hệ thống xác thực và phân quyền để đại diện cho một đối tượng người dùng hoặc một yêu cầu.

Các thành phần chính của JWT



Hình 2.11 Các thành phần chính của JWT

- **Header (Phần tiêu đề):** Chứa thông tin về loại token và thuật toán mã hóa được sử dụng (như HMAC SHA256 hoặc RSA).
- **Payload (Phần nội dung):** Chứa dữ liệu (claims) mà ta muốn truyền đi. Claims có thể là thông tin về người dùng, quyền truy cập hoặc thông tin khác.
- **Signature (Chữ ký):** Được tạo ra bằng cách mã hóa phần header và payload với một khóa bí mật (hoặc khóa public/private nếu sử dụng thuật toán RSA) và thuật toán mã hóa đã chỉ định. Chữ ký đảm bảo rằng token không bị thay đổi và xác thực nguồn gốc của nó.

2.9.2 Giới thiệu về Access Token

Access Token là một loại token được cấp cho người dùng sau khi họ xác thực thành công. Access Token chứa thông tin về quyền truy cập của người dùng và được gửi kèm trong các yêu cầu đến các dịch vụ bảo mật để xác thực và phân quyền.

Các đặc điểm chính của Access Token

- *Thời gian sống ngắn:* Access Token thường có thời gian sống ngắn để giảm thiểu rủi ro khi token bị lộ.
- *Được sử dụng để truy cập tài nguyên:* Được gửi trong header của các yêu cầu HTTP để xác thực và cấp quyền truy cập vào các tài nguyên hoặc dịch vụ.
- *Có thể được mã hóa bằng JWT:* Access Token có thể là một JWT, chứa các claims về quyền truy cập và thông tin người dùng.

Ví dụ: Khi người dùng đăng nhập vào một ứng dụng, server tạo một Access Token và gửi về cho người dùng. Token này sau đó được sử dụng để thực hiện các yêu cầu đến API để truy cập tài nguyên.

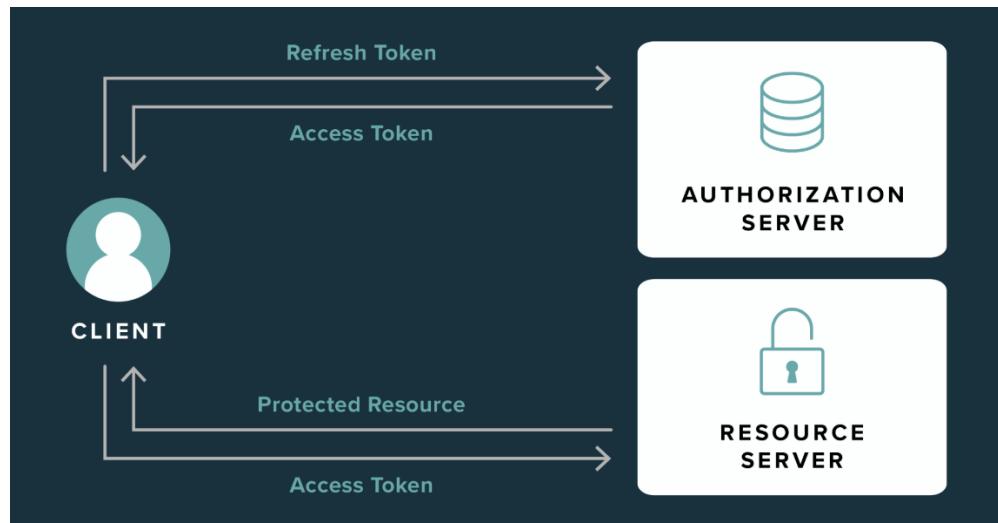
2.9.3 Giới thiệu về Refresh Token

Refresh Token là một loại token được sử dụng để yêu cầu một Access Token mới khi Access Token hiện tại hết hạn. Refresh Token thường có thời gian sống dài hơn so với Access Token và không được sử dụng trực tiếp để truy cập tài nguyên.

Các đặc điểm chính của Refresh Token

- *Thời gian sống dài hơn:* Refresh Token thường có thời gian sống lâu hơn để cho phép người dùng duy trì trạng thái đăng nhập mà không cần phải đăng nhập lại thường xuyên.
- *Chỉ được sử dụng để làm mới Access Token:* Refresh Token không được gửi cùng với các yêu cầu API để truy cập tài nguyên, mà chỉ được gửi đến endpoint làm mới token để yêu cầu một Access Token mới.
- *Bảo mật cao:* Refresh Token cần được bảo mật cẩn thận vì nếu bị lộ, nó có thể dẫn đến việc người dùng bị mất quyền kiểm soát tài khoản.

Ví dụ: Khi Access Token của người dùng hết hạn, client gửi Refresh Token đến server để yêu cầu Access Token mới. Nếu Refresh Token còn hợp lệ, server tạo và gửi Access Token mới về cho client.



Hình 2.12 Ví dụ Refresh Token

2.9.4 Tại sao cần JWT, Access Token & Refresh Token trong hệ thống

JWT, Access Token và Refresh Token đóng vai trò quan trọng trong việc quản lý phiên làm việc và bảo mật hệ thống. JWT giúp lưu trữ thông tin phiên làm việc trong token, giảm tải cho server và đơn giản hóa việc quản lý trạng thái. Access Token cung cấp cơ chế xác thực quyền truy cập vào các tài nguyên, giúp kiểm soát ai có quyền thực hiện hành động nào trong hệ thống. Khi Access Token hết hạn, Refresh Token cho phép làm mới Access Token mà không yêu cầu người dùng phải đăng nhập lại thường xuyên, cải thiện trải nghiệm người dùng và duy trì trạng thái đăng nhập liên tục.

Việc phân tách giữa Access Token và Refresh Token giúp quản lý thời gian sống của token một cách hiệu quả: Access Token có thời gian sống ngắn hơn để giảm thiểu rủi ro bảo mật, trong khi Refresh Token có thời gian sống dài hơn để duy trì phiên làm việc. Điều này không chỉ giúp giảm tải cho server, vì không cần lưu trữ trạng thái phiên làm việc, mà còn nâng cao bảo mật và cải thiện hiệu suất hệ thống.

2.10 AWS & EC2

2.10.1 Giới thiệu về AWS

Amazon Web Services (AWS) là một nền tảng điện toán đám mây toàn diện và được sử dụng rộng rãi trên toàn cầu. AWS cung cấp một loạt các dịch vụ điện toán, lưu trữ, cơ sở dữ liệu, phân tích, mạng, bảo mật, và nhiều dịch vụ khác, giúp doanh nghiệp triển khai, quản lý và mở rộng ứng dụng của họ một cách linh hoạt và hiệu quả. Các dịch vụ của AWS được cung cấp trên cơ sở pay-as-you-go, cho phép doanh nghiệp chỉ trả tiền cho những tài nguyên và dịch vụ mà họ thực sự sử dụng.

2.10.2 Giới thiệu về Amazon EC2

Amazon Elastic Compute Cloud (EC2) là một dịch vụ quan trọng của AWS, cung cấp khả năng mở rộng điện toán trên đám mây. EC2 cho phép người dùng triển khai và quản lý các máy chủ ảo, gọi là instances, với nhiều cấu hình phàn cứng và hệ điều hành khác nhau.

Các tính năng chính của Amazon EC2

- *Tính Linh Hoạt:* EC2 cho phép lựa chọn từ nhiều loại instance với các cấu hình khác nhau để phù hợp với yêu cầu của ứng dụng, từ các instance với khả năng tính toán cao đến các instance tối ưu hóa cho bộ nhớ và lưu trữ.
- *Khả Năng Mở Rộng:* EC2 hỗ trợ tự động mở rộng (auto-scaling) để điều chỉnh số lượng instances dựa trên nhu cầu sử dụng. Điều này giúp hệ thống luôn sẵn sàng và hiệu quả về chi phí.
- *Quản Lý Dễ Dàng:* AWS Management Console và các công cụ dòng lệnh giúp quản lý và cấu hình instances dễ dàng. EC2 cũng hỗ trợ các công cụ DevOps để tự động hóa triển khai và quản lý.
- *Tính Bảo Mật:* EC2 tích hợp với các dịch vụ bảo mật của AWS như Virtual Private Cloud (VPC) và Security Groups, cung cấp khả năng kiểm soát truy cập và bảo mật cho các instances.

- *Tích Hợp Với Các Dịch Vụ AWS Khác:* EC2 có thể được tích hợp với các dịch vụ khác của AWS như RDS (Relational Database Service), S3 (Simple Storage Service), và CloudWatch (giám sát và cảnh báo), giúp xây dựng và quản lý các hệ thống phức tạp một cách dễ dàng.

2.10.3 Sử dụng EC2 cho hệ thống microservice

Khi triển khai hệ thống microservice trên EC2, ta có thể tận dụng khả năng mở rộng và linh hoạt của EC2 để quản lý và điều chỉnh các dịch vụ của mình. Mỗi microservice có thể được triển khai trên các instance riêng biệt hoặc nhiều instance, tùy thuộc vào yêu cầu về tài nguyên và hiệu suất. EC2 cung cấp nền tảng vững chắc cho việc triển khai, mở rộng và duy trì hệ thống microservice, đồng thời hỗ trợ tích hợp với các dịch vụ AWS khác để xây dựng một hệ thống mạnh mẽ và hiệu quả.

CHƯƠNG 3. PHÂN TÍCH THIẾT KẾ

3.1 Phân tích quy trình kinh doanh

3.1.1 Tìm hiểu quy trình thực tế



Hình 3.1 Minh họa quy trình thực tế tại nhà hàng

Mô tả quy trình phục vụ thực khách trực tiếp tại nhà hàng

- Khi khách hàng đến nhà hàng, nhân viên tiếp đón sẽ chào đón và hướng dẫn khách đến bàn phù hợp với số lượng người.
- Nhân viên phục vụ sẽ chuẩn bị và mở bàn ăn dựa trên số lượng thực khách. Bàn ăn sẽ được sắp xếp và chuẩn bị đầy đủ để đáp ứng nhu cầu của khách hàng.
- Order món

Trường hợp 1 – Khách hàng tự order món thông qua Digital Menu

- + Nhân viên phục vụ sẽ cung cấp Digital Menu (như iPad hoặc Tablet) cho khách hàng.
- + Khách hàng có thể tự do thêm hoặc điều chỉnh các món ăn mà họ muốn chọn qua thiết bị số.

Trường hợp 2 – Khách hàng order món thông qua nhân viên phục vụ

- + Nhân viên phục vụ sẽ đưa menu cứng cho khách hàng để họ chọn món.

- + Khách hàng thực hiện chọn món và đặt order trực tiếp với nhân viên phục vụ.
- + Nhân viên phục vụ sẽ nhập đơn hàng vào thiết bị làm việc (như Smartphone) để gửi thông tin đơn hàng đến hệ thống.
- Đơn hàng cùng với các món ăn sẽ được gửi đến thiết bị làm việc của nhân viên bếp. Nhân viên bếp tiếp nhận thông báo và bắt đầu chuẩn bị các món ăn theo yêu cầu. Sau khi các món ăn đã được chuẩn bị xong, nhân viên bếp sẽ đánh dấu món ăn đã hoàn tất.
- Thông báo hoàn tất sẽ được gửi đến nhân viên phục vụ. Nhân viên phục vụ sẽ nhận thông báo và mang các món ăn đã hoàn tất đến vị trí bàn của khách hàng.

Quy trình thanh toán

- Sau khi hoàn tất bữa ăn, khách hàng sẽ gọi nhân viên phục vụ để yêu cầu thanh toán.
- Nhân viên phục vụ sẽ chuyển tiếp thông tin đơn hàng cần thanh toán cho người quản lý.
- Nhân viên phục vụ sẽ chuyển tiếp thông tin đơn hàng cần thanh toán cho người quản lý.
- Khách hàng sẽ kiểm tra hóa đơn và thực hiện thanh toán theo phương thức lựa chọn.

Quy trình tổng kết và thống kê

- Cuối ngày làm việc, nhân viên quản lý sẽ kiểm kê số lượng hóa đơn đã thanh toán trong ngày để tổng hợp số liệu.
- Nhân viên quản lý sẽ tổng hợp các dữ liệu về doanh thu, số lượng đơn hàng, và các số liệu khác để báo cáo lên cấp trên.

3.1.2 Quy trình nghiệp vụ trong hệ thống

Dựa trên quy trình cung cấp dịch vụ thực tế, chúng tôi đã thiết kế các quy trình nghiệp vụ cho hệ thống quản lý nhà hàng. Dưới đây là mô tả chi tiết về các quy trình nghiệp vụ đã được thiết kế.

Mở bàn phục vụ – Tạo mới đơn hàng



Hình 3.2 Quy trình mở bàn phục vụ - Tạo mới đơn hàng

- Bước 1: Khi có khách hàng vào nhà hàng, nhân viên phục vụ chọn bàn phù hợp dựa trên số lượng khách hàng.
- Bước 2: Nhân viên tiến hành mở bàn (tạo đơn hàng mới) và hệ thống cập nhật lại trạng thái của bàn hiện tại thành 'đang sử dụng'.
- Bước 3: Hệ thống tạo mã đơn hàng duy nhất cho bàn để theo dõi quá trình phục vụ và thanh toán.

Thêm món ăn vào bàn phục vụ



Hình 3.3 Quy trình thêm món ăn vào bàn phục vụ

- Bước 1: Khách hàng chọn món ăn từ menu (menu cứng hoặc Digital Menu)
- Bước 2: Khách hàng hoặc nhân viên phục vụ thực hiện thêm và điều chỉnh món ăn đã chọn vào hệ thống.
- Bước 3: Hệ thống cập nhật và lưu trữ thông tin món ăn vào đơn hàng tương ứng.

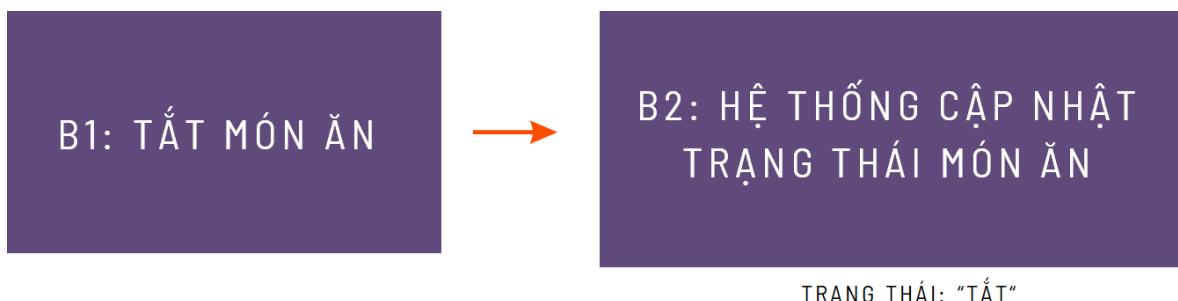
Xử lý món ăn



Hình 3.4 Quy trình xử lý món ăn

- Bước 1: Nhân viên bếp tiếp nhận đơn hàng và các món ăn qua hệ thống.
- Bước 2: Nhân viên cập nhật trạng thái qua từng giai đoạn thực hiện (ví dụ: 'đang chuẩn bị', 'đã hoàn tất') trên hệ thống.
- Bước 3: Hệ thống thông báo trạng thái món ăn cho nhân viên phục vụ.

Thiêu nguyên liệu nấu món ăn



Hình 3.5 Quy trình thiêu nguyên liệu món ăn

- Bước 1: Nhân viên bếp tắt trạng thái món ăn.
- Bước 2: Hệ thống cập nhật trạng thái món ăn đã hết nguyên liệu và gửi thông báo tới hệ thống menu.
- Bước 3: Khách hàng nhận biết món ăn đã hết trên Digital Menu hoặc nhân viên phục vụ thông báo trực tiếp với khách hàng về các món ăn đã hết và giới thiệu những món khác.

Chuyển bàn



Hình 3.6 Quy trình chuyển bàn

- Bước 1: Trong quá trình ăn uống, khách hàng có nhu cầu muốn chuyển sang bàn khác. Nhân viên phục vụ truy cập vào bàn phục vụ hiện tại và nhấn nút "Chuyển bàn".
- Bước 2: Một danh sách bàn ăn còn trống được hiển thị trên hệ thống, nhân viên chọn một bàn trống phù hợp với yêu cầu của khách hàng.
- Bước 3: Sau khi hoàn tất việc chuyển bàn, hệ thống cập nhật trạng thái của bàn mới thành 'đang phục vụ' và bàn cũ thành 'trống'.
- Bước 4: Hệ thống chuyển toàn bộ thông tin đơn hàng sang bàn mới.

Thanh toán



Hình 3.7 Quy trình thanh toán

- Bước 1: Nhân viên phục vụ nhấn nút yêu cầu thanh toán trên hệ thống.
- Bước 2: Quản lý kiểm tra và tạo một hóa đơn cho bàn cần thanh toán.
- Bước 3: Sau khi thanh toán thành công, hệ thống cập nhật trạng thái của bàn thành 'trống'.
- Bước 4: Hệ thống lưu trữ thông tin thanh toán và cập nhật số liệu thống kê doanh thu.

3.2 Phân tích yêu cầu

3.2.1 Đặc tả hệ thống

Đối với khách hàng, hệ thống cung cấp các chức năng tiện ích bao gồm khả năng chọn và thêm món ăn vào đơn hàng thông qua menu, bao gồm cả menu cứng và Digital Menu. Khi bữa ăn kết thúc, khách hàng có thể yêu cầu thanh toán và thực hiện thanh toán hóa đơn ngay tại bàn.

Nhân viên phục vụ có nhiệm vụ tạo đơn hàng mới khi khách hàng đến nhà hàng. Điều này bao gồm việc chọn bàn và mở đơn hàng mới, với hệ thống tự động cập nhật trạng thái của bàn thành 'đang sử dụng'. Khi khách hàng yêu cầu chuyển bàn, nhân viên phục vụ sẽ chọn bàn mới và cập nhật thông tin trên hệ thống. Họ cũng theo dõi và quản lý trạng thái món ăn, đảm bảo rằng trạng thái món ăn được cập nhật đúng khi món ăn được phục vụ cho khách.

Nhân viên bếp chịu trách nhiệm xử lý các đơn hàng. Khi đơn hàng có trạng thái "Created", nhân viên bếp tiếp nhận đơn hàng và cập nhật trạng thái món ăn từ "Pending" sang "In progress" khi bắt đầu chế biến. Sau khi món ăn được chế biến xong, nhân viên bếp cập nhật trạng thái món ăn từ "In progress" sang "Finished", và món ăn được đưa lên bàn phục vụ cho khách. Nhân viên bếp cũng cần quản lý nguyên liệu bằng cách cập nhật tình trạng món ăn khi nguyên liệu hết, nhằm tránh tình trạng khách hàng đặt món không còn sẵn có.

Nhân viên quản lý có nhiệm vụ xuất hóa đơn thanh toán cho khách hàng và giao cho nhân viên phục vụ để thực hiện thanh toán. Họ cũng quản lý lịch sử phiếu tính tiền, một chức năng chỉ dành cho nhân viên quản lý. Ngoài ra, nhân viên quản lý quản lý và phân công công việc cho các nhân viên trong ca trực.

Admin, thường là CEO hoặc Giám đốc của nhà hàng, có quyền quản lý các yếu tố quan trọng của hệ thống. Họ có thể cài đặt và điều chỉnh các khoản phụ phí liên quan đến dịch vụ, thêm, sửa đổi hoặc xóa món ăn trong menu, và quản lý thông tin về các bàn phục vụ, bao gồm số lượng và trạng thái của các bàn.

3.2.2 Đặc tả yêu cầu

3.2.2.1 Yêu cầu kinh doanh

Vấn đề hiện tại

- *Quy trình thủ công:* Các quy trình thực hiện như ghi nhận đơn hàng, quản lý nguyên liệu và thanh toán hiện đang được thực hiện bằng thủ công, dẫn đến hiệu suất làm việc thấp và dễ xảy ra sai sót.
- *Ghi nhận order bằng giấy:* Việc ghi lại đơn hàng của khách hàng bằng giấy không chỉ tốn thời gian mà còn dễ bị mất mát hoặc nhầm lẫn.
- *Thiếu nguyên liệu không cập nhật kịp thời:* Khi nguyên liệu hết, thông tin không được cập nhật nhanh chóng trên menu, dẫn đến việc khách hàng đợi lâu và không nhận được món ăn đã chọn.
- *Quản lý order chậm:* Quy trình ghi nhận và xử lý đơn hàng thêm món ăn hiện tại quá chậm vì yêu cầu phải thông báo qua nhiều bước, từ nhân viên phục vụ đến quản lý.

Cơ hội với hệ thống mới

- *Cải thiện quy trình và giảm thao tác thủ công:* Sử dụng hệ thống quản lý web sẽ tự động hóa các quy trình, giảm thiểu lỗi do thao tác thủ công và cải thiện hiệu quả công việc.
- *Quản lý order nhanh chóng và chính xác:* Hệ thống giúp quản lý đơn hàng hiệu quả hơn, giảm thiểu nhầm lẫn giữa các bàn và cải thiện tốc độ xử lý đơn hàng.
- *Cập nhật nguyên liệu nhanh chóng:* Hệ thống sẽ tự động cập nhật trạng thái nguyên liệu, giúp khách hàng biết ngay món ăn nào còn sẵn và lựa chọn món khác nếu cần.
- *Tăng cường sự hài lòng của khách hàng:* Khách hàng sẽ được phục vụ nhanh chóng và chính xác hơn, nâng cao trải nghiệm và sự hài lòng.

- *Xử lý order thêm nhanh chóng:* Thông tin về đơn hàng thêm món sẽ được cập nhật ngay lập tức, cho phép nhân viên bếp xử lý nhanh hơn và lược bỏ các thao tác không cần thiết.
- *Giảm tác động đến sức khỏe nhân viên:* Giảm khối lượng công việc thủ công giúp giảm căng thẳng và cải thiện sức khỏe cho nhân viên.

3.2.2.2 Yêu cầu chức năng

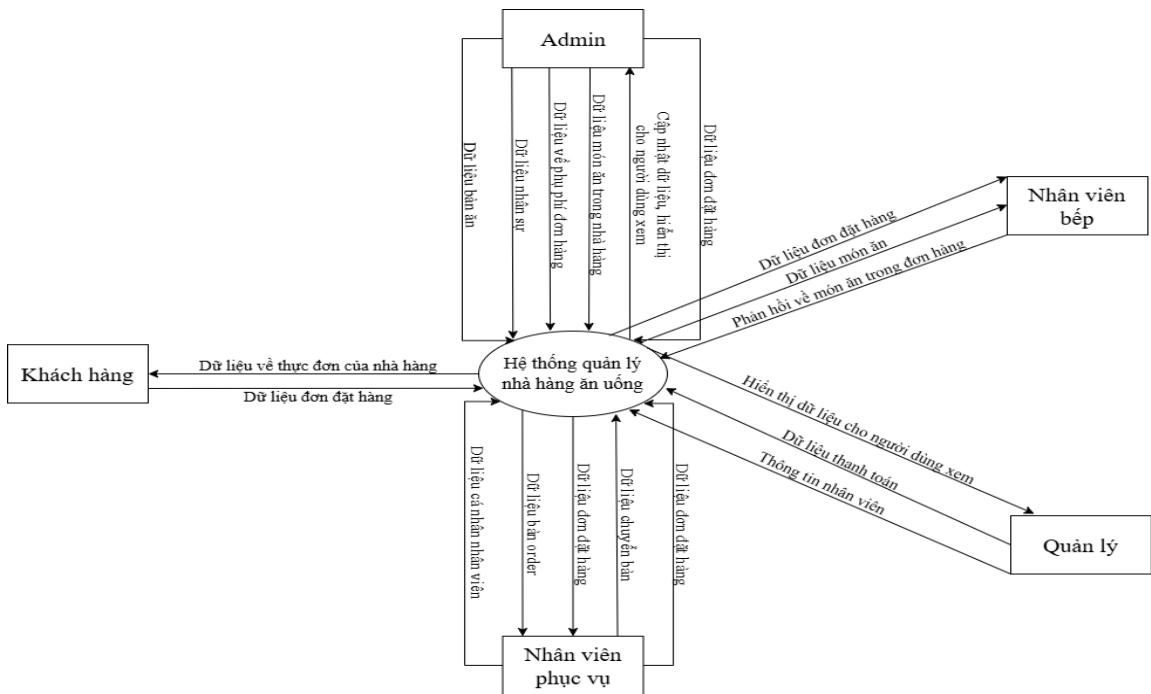
- *Xem Menu:* Khách hàng có thể xem nhanh menu của nhà hàng để chọn món ăn. Menu cần được hiển thị rõ ràng, bao gồm các thông tin cần thiết về món ăn như tên, mô tả, giá cả, và trạng thái (còn sẵn hoặc hết).
- *Thêm và Điều chỉnh Món Ăn:* Khách hàng có thể thêm món ăn vào đơn hàng và điều chỉnh các món đã chọn qua giao diện Digital Menu hoặc menu cứng. Điều chỉnh có thể bao gồm thay đổi số lượng hoặc thêm yêu cầu đặc biệt.
- *Tạo Order (Mở Bàn):* Nhân viên phục vụ có khả năng tạo đơn hàng mới khi khách hàng đến nhà hàng, bao gồm việc chọn bàn và ghi nhận các món ăn đã chọn. Hệ thống sẽ cập nhật trạng thái của bàn từ 'trống' thành 'đang sử dụng'.
- *Chuyển Bàn:* Nhân viên phục vụ có thể chuyển khách hàng sang bàn khác nếu khách hàng yêu cầu. Nhân viên phục vụ chọn bàn mới và cập nhật thông tin trên hệ thống để phản ánh trạng thái bàn mới và cũ.
- *Xử Lý Món Ăn:* Nhân viên bếp có thể tiếp nhận đơn hàng và xử lý các món ăn mới. Họ sẽ cập nhật trạng thái của món ăn từ "Created" sang "In progress" khi bắt đầu chế biến và từ "In progress" sang "Finished" khi món ăn hoàn tất.
- *Quản Lý Nguyên Liệu:* Nhân viên bếp có thể quản lý nguyên liệu của món ăn bằng cách cập nhật trạng thái của món ăn khi nguyên liệu đã hết. Hệ thống sẽ tự động cập nhật trạng thái món ăn trong menu để khách hàng không thể đặt những món không còn sẵn có.
- *Quản Lý Thanh Toán:* Nhân viên quản lý có thể xử lý thanh toán bằng cách tạo và xuất hóa đơn cho các đơn hàng. Họ sẽ giao hóa đơn cho nhân viên phục vụ để thực hiện thanh toán với khách hàng.

- *Quản Lý Nhân Viên:* Nhân viên quản lý có thể quản lý và phân công công việc cho các nhân viên trong ca trực. Điều này bao gồm việc theo dõi hiệu suất và sắp xếp lịch làm việc.
- *Quản Lý Phiếu Tính Tiền:* Nhân viên quản lý có thể theo dõi và quản lý lịch sử phiếu tính tiền của ngày, bao gồm việc xem lại các hóa đơn đã được tạo và thanh toán.
- *Quản Lý Phí Phụ Phí:* Admin (Giám đốc/CEO) có quyền quản lý các khoản phụ phí liên quan đến dịch vụ, bao gồm việc cài đặt và điều chỉnh các khoản phụ phí có thể áp dụng trên hóa đơn.
- *Quản Lý Món Ăn:* Admin (Giám đốc/CEO) có thể thêm, sửa đổi hoặc xóa các món ăn trong menu của nhà hàng. Điều này bao gồm việc cập nhật thông tin về món ăn như tên, mô tả, giá cả, và trạng thái.
- *Quản Lý Bàn Phục Vụ:* Admin (Giám đốc/CEO) có quyền quản lý thông tin về các bàn phục vụ trong nhà hàng, bao gồm việc cập nhật số lượng, trạng thái, và bố trí của các bàn.

3.2.2.3 Yêu cầu phi chức năng

- *Hiệu năng:* Cải thiện thời gian phản hồi và khả năng xử lý đồng thời để hệ thống hoạt động mượt mà và hiệu quả trong các giờ cao điểm.
- *Độ tin cậy:* Đảm bảo hệ thống có độ sẵn sàng cao và khả năng khôi phục sau sự cố nhanh chóng.
- *Bảo mật ứng dụng:* Hệ thống phải có cơ chế để phát hiện và xử lý lỗi bảo mật kịp thời. Đảm bảo các phiên làm việc (session) được quản lý an toàn để tránh các cuộc tấn công giả mạo hoặc lạm dụng.
- *Khả Năng Mở Rộng:* Hệ thống cần được thiết kế để có khả năng mở rộng dễ dàng khi số lượng người dùng hoặc khối lượng dữ liệu tăng lên.
- *Dễ bảo trì:* Đảm bảo mã nguồn được tổ chức và tài liệu hóa tốt để dễ dàng bảo trì và cập nhật.
- *Tính đa dạng thiết bị:* Đảm bảo hệ thống hoạt động tốt trên nhiều loại thiết bị khác nhau.

3.2.3 Context Diagram



Hình 3.8 Sơ đồ ngũ cảnh

3.2.4 Xác định tác nhân và usecase

3.2.4.1 Xác định tác nhân

Bảng 3.1 Các tác nhân trong hệ thống

STT	Tác nhân	Đặc tả
1	Người dùng	Là người có những chức năng được sử dụng hệ thống web quản lý nhà hàng ăn uống
2	Khách hàng	Là người bị giới hạn chỉ sử dụng được các chức năng như: Thêm và Điều chỉnh món vào bàn phục vụ
3	Nhân viên phục vụ	Là người đảm nhiệm các chức năng theo thiên hướng phục vụ thực khách như: Tạo đơn hàng, chuyển bàn.
4	Nhân viên bếp	Là người đảm nhiệm các chức năng về món ăn như: xử lý món ăn mới, quản lý nguyên liệu món ăn
5	Quản lý	Là người có vai trò quan trọng với các chức năng như: Thanh toán, quản lý phiếu tính tiền, quản lý nhân viên.
6	Admin	Là người có thể can thiệp toàn bộ hệ thống và một số chức năng chỉ người quản trị có thể sử dụng như: quản lý phụ phí, quản lý món ăn, quản lý bàn phục vụ.

3.2.4.2 Xác định usecase

Bảng 3.2 Các usecase trong hệ thống

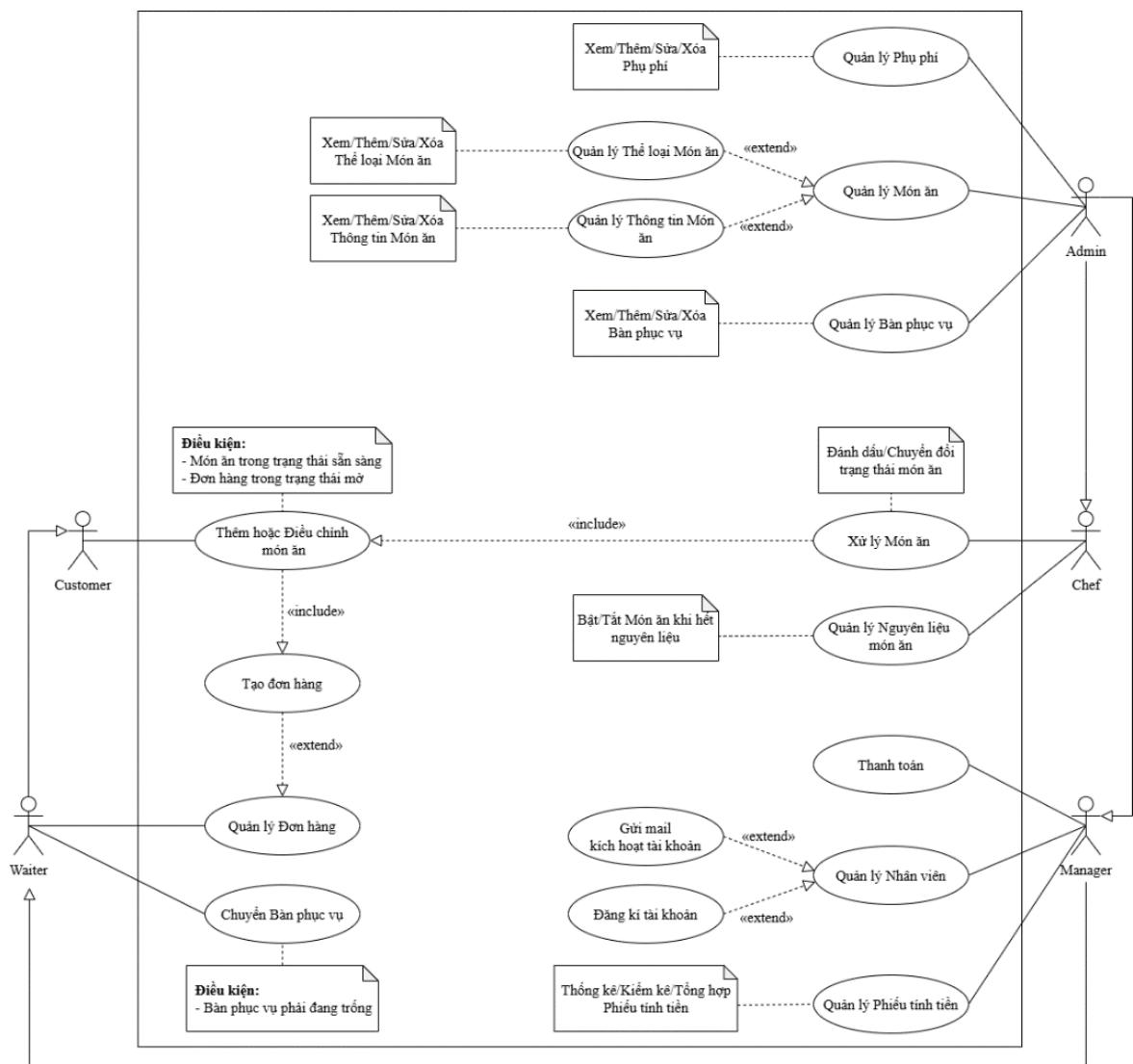
ID	Tên use case	Mô tả	Actor
UC01	Đăng nhập	Chức năng đăng nhập tài khoản cho phép các actor có thể đăng nhập vào hệ thống. Tùy từng loại actor mà có quyền truy cập khác nhau	Admin Quản lý Nhân viên phục vụ Nhân viên bếp
UC02	Đăng ký tài khoản	Chức năng cho phép actor khi truy cập vào có thể tạo tài khoản cho nhân viên	Admin Quản lý
UC03	Kích hoạt tài khoản	Chức năng cho phép actor khi truy cập vào đường link sẽ thực hiện kích hoạt tài khoản	Quản lý Nhân viên phục vụ Nhân viên bếp
UC04	Gửi mail kích hoạt tài khoản	Chức năng cho phép actor khi truy cập vào có thể gửi mail chứa link kích hoạt tài khoản cho nhân viên	Admin Quản lý
UC05	Đăng xuất	Chức năng cho phép các actor khi truy cập vào tài khoản có thể đăng xuất khỏi hệ thống	Admin Quản lý Nhân viên phục vụ Nhân viên bếp
UC06	Đổi mật khẩu	Chức năng cho phép các actor có thể đổi mật khẩu tài khoản	Admin Quản lý Nhân viên phục vụ Nhân viên bếp
UC07	Tạo đơn hàng	Hệ thống cho phép các actor có quyền thực hiện tạo mới đơn hàng khi có khách hàng đến ăn tại nhà hàng	Nhân viên phục vụ
UC08	Chuyển bàn	Hệ thống cho phép các actor có quyền thực hiện chuyển bàn khi	Nhân viên phục vụ

		khách hàng có nhu cầu thay đổi bàn ăn	
UC09	Thêm và điều chỉnh món ăn vào bàn phục vụ	Hệ thống cho phép các actor có quyền thêm hoặc điều chỉnh món ăn vào order cụ thể	Khách hàng
UC10	Thanh toán	Hệ thống cho phép actor có quyền tạo và in hóa đơn thanh toán	Quản lý
UC11	Quản lý phiếu tính tiền	Hệ thống cho phép actor có quyền tổng hợp phiếu tính tiền, xem thống kê, xem lịch sử của phiếu tính tiền theo ngày, tháng, năm	Quản lý
UC12	Quản lý nhân viên	Hệ thống cho phép actor có quyền quản lý tài khoản của nhân viên	Admin Quản lý
UC13	Quản lý phụ phí	Hệ thống cho phép actor có quyền quản lý phụ phí: xem, thêm, xóa, sửa phụ phí cho hóa đơn	Admin
UC14	Quản lý món ăn	Hệ thống cho phép actor có quyền quản lý thực đơn bao gồm thẻ thoại món ăn và thông tin món ăn	Admin
UC15	Quản lý bàn phục vụ	Hệ thống cho phép actor có quyền xem, thêm, sửa, xóa bàn phục vụ	Admin
UC16	Xử lý món ăn	Hệ thống cho phép nhân viên bếp đánh dấu món ăn đã phục vụ	Nhân viên bếp
UC17	Quản lý nguyên liệu làm món ăn	Hệ thống cho phép nhân viên bếp có quyền bật/tắt món ăn khi hết nguyên liệu chế biến	Nhân viên bếp

3.2.5 Phân tích Usecase

3.2.5.1 Sơ đồ usecase tổng quát

Sau khi đã xác định được các usecase có trong hệ thống, chúng tôi tiến hành xây dựng sơ đồ usecase tổng quát để thể hiện mối quan hệ giữa người dùng và các chức năng chính của hệ thống. Sơ đồ này cung cấp cái nhìn toàn diện về cách người dùng tương tác với hệ thống, từ đó làm cơ sở cho việc phân tích và thiết kế chi tiết hơn trong các bước tiếp theo.



Hình 3.9 Sơ đồ usecase tổng quát

3.2.5.2 Đặc tả usecase

Bảng 3.3 Usecase đăng nhập

<i>Usecase:</i> Đăng nhập	<i>ID:</i> UC01	<i>Priority:</i> High		
<i>Actor:</i> Admin, quản lý, nhân viên phục vụ, nhân viên bếp				
<i>Description:</i> Khi actor được phân quyền muốn truy cập vào hệ thống web quản lý nhà hàng ăn uống				
<i>Trigger:</i> Actor được phân quyền phải đăng nhập vào hệ thống thì mới được sử dụng các chức năng				
<i>Precondition:</i> tài khoản của các actor phân quyền đã được kích hoạt				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Actor chọn đăng nhập	1.1 Hệ thống hiển thị màn hình chưa form đăng nhập			
2. Actor nhập username và password	2.1 Hệ thống hiển thị dữ liệu actor vừa nhập + Username hiển thị rõ, password hiển thị theo kiểu ẩn			
3. Actor nhấn nút Đăng nhập	3.1 Hệ thống hiển thị thông báo thành công và chuyển hướng người dùng đến trang chức năng của hệ thống			
<i>Postconditions:</i> Đăng nhập thành công vào hệ thống, actor được chuyển hướng đến trang chức năng của hệ thống				
<i>Exception:</i>				
<ul style="list-style-type: none"> • Người dùng nhập sai username hoặc password hoặc cả 2 trường hợp • Người dùng không nhấn nút Đăng nhập sau khi nhập thông tin • Tài khoản của người dùng chưa được kích hoạt 				

Bảng 3.4 Usecase đăng kí tài khoản

<i>Usecase:</i> Đăng kí	<i>ID:</i> UC02	<i>Priority:</i> High		
<i>Actor:</i> Admin, quản lý				
<i>Description:</i> Khi actor được phân quyền muốn tạo tài khoản cho actor khác để truy cập vào hệ thống web quản lý nhà hàng ăn uống				
<i>Trigger:</i> Khi có nhân viên làm việc mới tại nhà hàng tham gia vào hệ thống				
<i>Precondition:</i> Tài khoản của các actor phân quyền đã được kích hoạt				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Actor chọn đăng kí	1.1 Hệ thống hiển thị màn hình chưa form đăng kí			
2. Actor nhập fullName, nationalId, phone, gmail và chọn gender	2.1 Hệ thống hiển thị dữ liệu actor vừa nhập và chọn + Fullname, nationalId, phone, gmail, gender hiển thị rõ dữ liệu mà người dùng nhập và chọn			
3. Actor nhấn nút Đăng kí	3.1 Hệ thống hiển thị thông báo thành công và chuyển hướng đến trang chức năng quản lý nhân viên			
<i>Postconditions:</i>				
<ul style="list-style-type: none"> • Đăng kí thành công tài khoản trên hệ thống • Hệ thống gửi link kích hoạt tài khoản mới đến gmail đăng ký 				
<i>Exception:</i>				
<ul style="list-style-type: none"> • Người dùng nhập và chọn thiếu một trong các trường dữ liệu (fullName, gender, nationalId, phone, gmail) • Một trong các trường bị trùng dữ liệu với tài khoản khác (nationalId, phone, gmail) • Người dùng không nhấn nút Đăng kí sau khi nhập thông tin 				

Bảng 3.5 Usecase kích hoạt tài khoản

<i>Usecase:</i> Kích hoạt tài khoản	<i>ID:</i> UC03	<i>Priority:</i> High		
<i>Actor:</i> Quản lý, nhân viên phục vụ, nhân viên bếp				
<i>Description:</i> Khi actor được phân quyền muốn truy cập vào hệ thống bằng tài khoản thì tài khoản phải được kích hoạt				
<i>Trigger:</i> Khi người dùng cần kích hoạt tài khoản để có thể truy cập vào hệ thống				
<i>Precondition:</i> Tài khoản của các actor phân quyền đã được tạo				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Actor chọn vào link kích hoạt mà hệ thống đã gửi qua email tài khoản đã đăng ký	1.1 Hệ thống hiển thị thông báo đã kích hoạt tài khoản thành công			
<i>Postconditions:</i> Kích hoạt thành công tài khoản trên hệ thống				
<i>Exception:</i> Người dùng không nhấn link kích hoạt tài khoản				

Bảng 3.6 Usecase gửi mail kích hoạt tài khoản

<i>Usecase:</i> Gửi mail kích hoạt tài khoản	<i>ID:</i> UC04	<i>Priority:</i> High		
<i>Actor:</i> Admin, quản lý				
<i>Description:</i> Khi actor được phân quyền muốn gửi lại mail chứa link kích hoạt				
<i>Trigger:</i> Khi nhân viên cần nhận lại link kích hoạt tài khoản				
<i>Precondition:</i> Tài khoản của các actor phân quyền đã được kích hoạt.				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Actor nhấn nút “Gửi mail kích hoạt” theo email muốn gửi	1.1 Hệ thống hiển thị thông báo đã gửi mail kích hoạt tài khoản thành công cho email đã chọn			
<i>Postconditions:</i> Gửi mail kích hoạt thành công cho email				
<i>Exception:</i> Người dùng không nhấn nút “Gửi mail kích hoạt”				

Bảng 3.7 Usecase đăng xuất

<i>Usecase:</i> Đăng xuất	<i>ID:</i> UC05	<i>Priority:</i> High		
<i>Actor:</i> Admin, quản lý, nhân viên phục vụ, nhân viên bếp				
<i>Description:</i> Khi người dùng được phân quyền đã sử dụng xong hệ thống và có nhu cầu thoát khỏi hệ thống để thực hiện bảo mật				
<i>Trigger:</i> Người dùng nhấn chọn đăng xuất				
<i>Precondition:</i> Người dùng đã đăng nhập thành công vào hệ thống				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Người dùng chọn nút đăng xuất	1.1 Hệ thống đăng xuất thông tin của người dùng và trở về màn hình đăng nhập			
<i>Postconditions:</i> Đăng xuất thành công, khi người dùng muốn vào lại hệ thống bắt buộc phải đăng nhập				
<i>Exception:</i> Không có				

Bảng 3.8 Usecase đổi mật khẩu

<i>Usecase:</i> Đổi mật khẩu	<i>ID:</i> UC06	<i>Priority:</i> High		
<i>Actor:</i> Admin, quản lý, nhân viên phục vụ, nhân viên bếp				
<i>Description:</i> Khi người dùng được phân quyền muốn đổi mật khẩu tài khoản để đảm bảo bảo mật				
<i>Trigger:</i> Người dùng nhấn chọn đổi mật khẩu				
<i>Precondition:</i> Người dùng đã đăng nhập thành công vào hệ thống				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Người dùng chọn nút đổi mật khẩu	1.1 Hệ thống hiển thị form chứa thông tin đổi mật khẩu			
2. Người dùng nhập mật khẩu cũ, mật khẩu mới, xác nhận mật khẩu mới	2.1 Hệ thống hiển thị thông tin người dùng nhập. +Mật khẩu cũ, mật khẩu mới, xác nhận mật khẩu hiển thị dữ liệu dạng ẩn với kí tự mà người dùng nhập			
3. Người dùng nhấn nút Xác nhận	3.1 Hệ thống hiển thị thông báo thành công và chuyển hướng đến trang đăng nhập.			
<i>Postconditions:</i> Đổi mật khẩu thành công.				
<i>Exception:</i>				
<ul style="list-style-type: none"> • Người dùng nhập và chọn thiếu một trong các trường dữ liệu (mật khẩu cũ, mật khẩu mới, xác nhận mật khẩu mới). • Người dùng nhập sai mật khẩu cũ. • Người dùng nhập xác nhận mật khẩu mới không trùng khớp mới mật khẩu mới. • Người dùng không nhấn nút Xác nhận sau khi nhập thông tin. • Tài khoản người dùng chưa được kích hoạt. • Người dùng chưa đăng nhập vào hệ thống. 				

Bảng 3.9 Usecase tạo đơn hàng

<i>Usecase:</i> Tạo đơn hàng	<i>ID:</i> UC07	<i>Priority:</i> High		
<i>Actor:</i> Nhân viên phục vụ				
<i>Description:</i> Khi có khách hàng mới đến cửa hàng ăn uống.				
<p><i>Trigger:</i></p> <ul style="list-style-type: none"> Khách hàng muốn ăn uống tại cửa hàng thì phải thực hiện order Nhân viên phục vụ là người tạo ra đơn hàng cho khách hàng 				
<i>Precondition:</i> Bàn phục vụ ở trạng thái trống				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Người dùng chọn nút thêm mới order trên màn hình tablet	1.1 Hệ thống hiển thị form chứa thông tin cần thiết cho một order.			
2. Trên màn hình order, khách hàng chọn các món ăn trên màn hình mà khách muốn ăn				
3. Khách hàng nhấn nút ok	3.1 Hệ thống hiển thị các món ăn mà khách đã chọn.			
4. Nhân viên phục vụ nhấn nút gửi order	4.1 Màn hình hiển thị đơn hàng đã được tạo thành công.			
<i>Postconditions:</i> Tạo đơn hàng thành công				
<i>Exception:</i>				
<ul style="list-style-type: none"> Khách hàng không chọn món ăn cho đơn hàng Nhân viên phục vụ cung cấp thiếu các thông tin bắt buộc trong form tạo Nhân viên phục không nhấn nút gửi đơn hàng đi 				

Bảng 3.10 Usecase chuyển bàn

<i>Usecase:</i> Chuyển bàn	<i>ID:</i> UC08	<i>Priority:</i> High
<i>Actor:</i> Nhân viên phục vụ		
<i>Description:</i> Khi khách hàng có nhu cầu thay đổi bàn ăn hoặc số lượng khách đến sau của bàn tăng lên quá nhiều, không đủ chỗ ngồi		
<i>Trigger:</i> Khách hàng yêu cầu chuyển bàn		
<i>Precondition:</i>		
<ul style="list-style-type: none"> Đơn hàng ở trạng thái đang phục vụ Bàn cần chuyển ở trạng thái trống 		
<i>Normal Course</i>		<i>Information for steps</i>
1. Nhân viên phục vụ xem bàn phục vụ đang trống		1.1 Hệ thống hiển thị danh sách bàn phục vụ có trạng thái trống
2. Nhân viên phục vụ truy cập vào bàn phục vụ đang ở có khách hàng đang yêu cầu chuyển bàn		2.1 Hệ thống hiển thị màn hình bàn phục vụ vừa chọn
3. Nhân viên phục vụ nhấn nút chuyển bàn		3.1 Hệ thống hiển thị popup chứa các bàn đang đóng
4. Nhân viên phục vụ chọn bàn đủ sức chứa để chuyển		4.1 Hệ thống hiển thị thông báo chuyển bàn thành công
<i>Postconditions:</i>		
<ul style="list-style-type: none"> Chuyển bàn thành công Bàn phục vụ cũ ở trạng thái trống Bàn phục vụ mới chuyển ở trạng thái đang phục vụ 		
<i>Exception:</i> Nhân viên chọn bàn đã phục vụ khác.		

Bảng 3.11 Usecase thêm món ăn vào phục vụ

<i>Usecase:</i> Thêm món ăn	<i>ID:</i> UC09	<i>Priority:</i> High		
<i>Actor:</i> Khách hàng				
<i>Description:</i> Khách hàng chọn món ăn để order				
<i>Trigger:</i> Khách hàng nhấn gửi danh sách món đã chọn				
<p><i>Precondition:</i></p> <ul style="list-style-type: none"> • Đơn hàng ở trạng thái đang phục vụ • Danh sách món ăn là hợp lệ 				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Khách hàng chọn các món ăn trong menu trên màn hình tablet	1.1 Hệ thống hiển thị màn hình chứa danh sách món ăn hiện có của nhà hàng			
2. Khách hàng nhấn chọn món ăn cần order	2.1 Hệ thống tick vào các món ăn mà khách hàng đã chọn			
3. Khách hàng nhấn nút Xác nhận	3.1 Hệ thống cập nhật các món ăn đã chọn vào đơn hàng hiện tại			
<i>Postconditions:</i> Thêm món ăn vào đơn hàng thành công				
<i>Exception:</i> Khách hàng không chọn bất kỳ món ăn nào				

Bảng 3.12 Usecase thanh toán

<i>Usecase:</i> Thanh toán	<i>ID:</i> UC10	<i>Priority:</i> High		
<i>Actor:</i> Quản lý				
<i>Description:</i> Khách hàng có yêu cầu thanh toán sau khi dùng bữa xong.				
<i>Trigger:</i> Khách hàng yêu cầu thanh toán				
<i>Precondition:</i> Đơn hàng ở trạng thái đang phục vụ				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Quản lý chọn đơn hàng của bàn phục vụ khách hàng đang yêu cầu thanh toán	1.1 Màn hình hiển thị thông tin chi tiết của đơn hàng			
2. Quản lý chọn các phụ phí cần thiết cho đơn hàng	2.1 Hệ thống hiển thị danh sách các loại phụ phí hiện có			
3. Quản lý nhấn nút thanh toán	3.1 Hệ thống hiển thị màn hình popup có thông tin chi tiết hóa đơn			
4. Quản lý nhấn nút Xuất hóa đơn	4.1 Hệ thống kết nối với máy in và thông báo xuất thành công			
<i>Postconditions:</i> Thanh toán thành công				
<i>Exception:</i> Kết nối máy in có vấn đề, không xuất được hóa đơn				

Bảng 3.13 Usecase quản lý phiếu tính tiền

<i>Usecase:</i> Quản lý phiếu tính tiền	<i>ID:</i> UC11	<i>Priority:</i> High		
<i>Actor:</i> Quản lý				
<i>Description:</i> Quản lý phiếu tính tiền giúp người quản lý có thể thống kê lại các hóa đơn của cửa hàng				
<i>Trigger:</i> Khi người quản lý báo cáo lại tổng hợp phiếu tính tiền cho cấp trên				
<i>Precondition:</i> Các hóa đơn đã được thanh toán thành công gọi là phiếu tính tiền				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Quản lý chọn chức năng Quản lý phiếu tính tiền	1.1 Hệ thống hiển thị màn hình chưa danh sách các phiếu tính tiền			
<i>Postconditions:</i> Danh sách các phiếu tính tiền hiển thị trên màn hình.				
<i>Exception:</i> Không có				

Bảng 3.14 Usecase quản lý nhân viên

<i>Usecase:</i> Quản lý nhân viên	<i>ID:</i> UC12	<i>Priority:</i> High		
<i>Actor:</i> Quản lý				
<i>Description:</i> Quản lý thực hiện quản lý các nhân viên làm việc tại nhà hàng				
<p><i>Trigger:</i></p> <ul style="list-style-type: none"> • Quản lý thực hiện quản lý các tài khoản liên quan đến nhà hàng khi nhân viên làm việc tại nhà hàng • Quản lý thông tin nhân viên 				
<i>Precondition:</i> Tài khoản của nhân viên đã tồn tại				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Quản lý chọn chức năng quản lý nhân viên	1.1 Hệ thống hiển thị màn hình chứa danh sách các tài khoản nhân viên hiện có và thông tin nhân viên			
2. Quản lý thực hiện các tùy chọn đối với nhân viên cụ thể	2.1 Hệ thống hiển thị màn hình chứa thông tin mà quản lý đã tùy chọn người dùng đã cập nhật			
<i>Postconditions:</i>				
<ul style="list-style-type: none"> • Danh sách nhân viên hiển thị • Quản lý có thể xem/ sửa thông tin nhân viên. 				
<i>Exception:</i> Không có				

Bảng 3.15 Usecase quản lý phụ phí

<i>Usecase:</i> Quản lý phụ phí	<i>ID:</i> UC13	<i>Priority:</i> High		
<i>Actor:</i> Admin				
<i>Description:</i> Admin thực hiện quản lý các khoản phụ phí có thể phát sinh.				
<i>Trigger:</i> Phụ phí được đính kèm theo quy định bàn hành của nhà nước.				
<i>Precondition:</i> Nhà hàng đã thực hiện đăng ký thuế với nhà nước.				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Admin chọn quản lý phụ phí	1.1 Hệ thống hiển thị màn hình thông tin các loại phụ phí hiện có			
2. Admin thực hiện các tùy chọn trong quản lý thuế	2.1 Hệ thống hiển thị màn hình chứa thông tin mà Admin đã tùy chọn cập nhật.			
<i>Postconditions:</i> Admin có thể tùy chỉnh phụ phí Xem/Thêm/Sửa/Xóa				
<i>Exception:</i> Không có				

Bảng 3.16 Usecase quản lý món ăn

<i>Usecase:</i> Quản lý món ăn	<i>ID:</i> UC14	<i>Priority:</i> High		
<i>Actor:</i> Admin				
<i>Description:</i> Admin thực hiện quản lý các món ăn trong nhà hàng				
<i>Trigger:</i> Danh sách món ăn giúp khách hàng dễ dàng hơn trong việc order món				
<i>Precondition:</i> Các món ăn trước đó đã có trong hệ thống				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Admin chọn quản lý món ăn	1.1 Hệ thống hiển thị màn hình thông tin chứa danh sách các món ăn hiện có trong hệ thống			
2. Admin thực hiện các tùy chọn trong quản lý món ăn	2.1 Hệ thống hiển thị màn hình chứa thông tin mà Admin đã tùy chọn cập nhật			
<i>Postconditions:</i> Admin có thể tùy chỉnh món ăn Xem/Thêm/Sửa/Xóa				
<i>Exception:</i> Không có				

Bảng 3.17 Usecase quản lý bàn phục vụ

<i>Usecase:</i> Quản lý bàn phục vụ	<i>ID:</i> UC15	<i>Priority:</i> High		
<i>Actor:</i> Admin				
<i>Description:</i> Admin thực hiện quản lý các bàn phục vụ trong nhà hàng				
<i>Trigger:</i> Danh sách bàn phục vụ giúp nhân viên phục vụ có thể linh hoạt trong việc chọn bàn phù hợp với số lượng khách hàng				
<i>Precondition:</i> Bàn phục vụ đã có tồn tại trước đó				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Admin chọn quản lý bàn phục vụ	1.1 Hệ thống hiển thị màn hình thông tin chứa danh sách các bàn phục vụ hiện có ở nhà hàng			
2. Admin thực hiện các tùy chọn trong quản lý bàn phục vụ	2.1 Hệ thống hiển thị màn hình chứa thông tin mà Admin đã tùy chọn cập nhật			
<i>Postconditions:</i> Admin có thể tùy chỉnh bàn phục vụ Xem/Thêm/Sửa/Xóa				
<i>Exception:</i> Không có				

Bảng 3.18 Usecase xử lý món ăn

<i>Usecase:</i> Xử lý món ăn	<i>ID:</i> UC16	<i>Priority:</i> High		
<i>Actor:</i> Nhân viên bếp				
<i>Description:</i> Nhân viên bếp đánh dấu món ăn phục vụ cho khách hàng				
<p><i>Trigger:</i></p> <ul style="list-style-type: none"> • Khi có một order cụ thể từ khách hàng • Nhân viên bếp đánh dấu món ăn đã phục vụ giúp cho nhân viên phục vụ có thể thông tin được khi nào món ăn hoàn tất và mang lên bàn phục vụ cho thực khách 				
<i>Precondition:</i> Khi có một order từ khách hàng				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Nhân viên bếp xem danh sách món ăn của một đơn hàng	1.1 Hệ thống hiển thị màn hình thông tin chứa danh sách món ăn trong đơn hàng			
2. Nhân viên bếp tiến hành chế biến món ăn				
3. Nhân viên bếp chế biến xong món ăn và đánh dấu đã phục vụ vào hệ thống.	3.1 Hệ thống hiển thị thông tin món ăn trong đơn hàng đã được thực hiện xong cho nhân viên phục vụ			
<i>Postconditions:</i> Nhân viên bếp đánh dấu món ăn đã phục vụ thành công.				
<i>Exception:</i> Không có				

Bảng 3.19 Usecase quản lý nguyên liệu

<i>Usecase:</i> Quản lý nguyên liệu	<i>ID:</i> UC17	<i>Priority:</i> High		
<i>Actor:</i> Nhân viên bếp				
<i>Description:</i>				
<ul style="list-style-type: none"> • Nhân viên bếp thực hiện quản lý nguyên liệu chế biến món ăn • Nhân viên bếp sẽ bật / tắt khi nguyên liệu còn hoặc hết đối với món ăn cụ thể 				
<i>Trigger:</i> Nhân viên bếp bật / tắt món ăn khi hết hoặc còn nguyên liệu				
<i>Precondition:</i> Nhân viên bếp kiểm tra nguyên liệu trước khi chế biến món ăn.				
<i>Normal Course</i>	<i>Information for steps</i>			
1. Nhân viên bếp chọn quản lý nguyên liệu	1.1 Hệ thống hiển thị màn hình thông tin chưa danh sách các món ăn			
2 Nhân viên bếp nhấn bật/ tắt món ăn	2.1 Hệ thống hiển thị màn hiển thị món ăn (bật) hoặc ẩn món ăn (tắt) đi			
<i>Postconditions:</i> Nhân viên bếp bật/ tắt được món ăn khi hết nguyên liệu				
<i>Exception:</i> Không có				

3.3 Thiết kế hệ thống

3.3.1 Kiến trúc hệ thống

3.3.1.1 Phân chia dịch vụ

Dưới đây là mô tả chi tiết về cách phân chia các dịch vụ trong hệ thống quản lý nhà hàng của chúng tôi, được xây dựng theo kiến trúc microservices. Kiến trúc này cho phép các dịch vụ hoạt động độc lập và tương tác với nhau thông qua các giao thức tiêu chuẩn. Mô hình microservices không chỉ tăng tính linh hoạt và khả năng mở rộng mà còn nâng cao khả năng bảo trì của hệ thống, đảm bảo các dịch vụ luôn hoạt động hiệu quả và dễ dàng thích ứng với các thay đổi.

Catalog Service

- *Quản lý danh mục:* Chức năng này cho phép tạo mới, cập nhật, xóa bỏ và truy xuất thông tin chi tiết về các danh mục sản phẩm trong hệ thống, giúp dễ dàng quản lý các nhóm món ăn một cách hiệu quả.
- *Quản lý sản phẩm:* Dịch vụ này chịu trách nhiệm về việc tạo mới, cập nhật, xóa bỏ và lấy thông tin chi tiết về các món ăn. Bao gồm cả thông tin về giá cả, mô tả, hình ảnh, và các thông tin liên quan khác, giúp khách hàng có cái nhìn rõ ràng về sản phẩm.
- *Quản lý trạng thái của sản phẩm:* Chức năng này cho phép bật hoặc tắt trạng thái của một hoặc nhiều món ăn trong hệ thống, giúp dễ dàng quản lý sự sẵn có của các sản phẩm theo thời gian thực.
- *Tìm kiếm và lọc:* Dịch vụ cung cấp khả năng tìm kiếm và lọc các món ăn dựa trên nhiều tiêu chí khác nhau như danh mục, giá cả, hoặc các đặc điểm khác, giúp người dùng nhanh chóng tìm thấy sản phẩm mong muốn.

Order Service

- *Quản lý đơn hàng:* Đây là chức năng chính để tạo mới, cập nhật, xóa bỏ và truy xuất thông tin chi tiết về các đơn hàng. Nó đảm bảo rằng mọi đơn hàng được ghi nhận và xử lý một cách chính xác.

- *Thay đổi bàn:* Chức năng này cho phép quản lý việc thay đổi bàn cho đơn hàng, phù hợp với yêu cầu của khách hàng trong quá trình đặt chỗ và sắp xếp chỗ ngồi.
- *Hủy đơn hàng:* Quản lý việc hủy đơn hàng một cách hiệu quả, đảm bảo rằng các yêu cầu hủy đơn hàng được xử lý đúng quy trình và cập nhật trong hệ thống.
- *Quản lý các món ăn trong đơn hàng:* Chức năng này cho phép thêm, cập nhật và xóa các món ăn trong đơn hàng, giúp linh hoạt trong việc quản lý và điều chỉnh thực đơn cho khách hàng.

Payment Service

- *Quản lý thanh toán và giao dịch:* Dịch vụ này giúp tạo mới, cập nhật và xóa bỏ các giao dịch thanh toán trong hệ thống, đảm bảo quá trình thanh toán diễn ra thuận lợi và an toàn.
- *Quản lý phụ phí:* Chức năng này cho phép thêm, cập nhật và xóa bỏ các phụ phí trong các giao dịch thanh toán, đảm bảo rằng mọi khoản phí phát sinh đều được ghi nhận một cách chính xác.
- *Thông kê:* Dịch vụ cung cấp các báo cáo chi tiết về các thanh toán dựa trên các tiêu chí khác nhau như ngày, tháng, năm, giúp quản lý có cái nhìn toàn diện về hoạt động tài chính của nhà hàng.

Table Service

- *Quản lý bàn nhà hàng:* Chức năng này cho phép tạo mới, cập nhật, xóa bỏ và lấy thông tin chi tiết về các bàn trong nhà hàng, giúp quản lý dễ dàng kiểm soát và sắp xếp chỗ ngồi cho khách hàng.
- *Quản lý sơ đồ chỗ ngồi:* Dịch vụ cung cấp sơ đồ chỗ ngồi của nhà hàng và cho phép phân bổ chỗ ngồi một cách hiệu quả, đảm bảo tối ưu hóa không gian và đáp ứng nhu cầu của khách hàng.

User Service

- *Quản lý thông tin người dùng:* Dịch vụ này cho phép đăng ký, cập nhật, xóa bỏ và truy xuất thông tin chi tiết về người dùng, đảm bảo rằng mọi thông tin cá nhân đều được quản lý một cách an toàn và bảo mật.
- *Quản lý tài khoản:* Chức năng bao gồm việc đăng nhập, đăng xuất, kích hoạt tài khoản, gửi lại email kích hoạt tài khoản, đặt lại mật khẩu và thay đổi mật khẩu, đảm bảo người dùng có thể quản lý tài khoản của mình một cách thuận tiện.

Kitchen Service

- *Xử lý các yêu cầu đặt món từ khách hàng:* Dịch vụ này nhận và quản lý các đơn đặt hàng từ khách hàng, truyền tải thông tin đến các thiết bị của nhân viên bếp qua WebSocket, giúp quá trình chế biến món ăn diễn ra nhanh chóng và chính xác.

Waiter Service

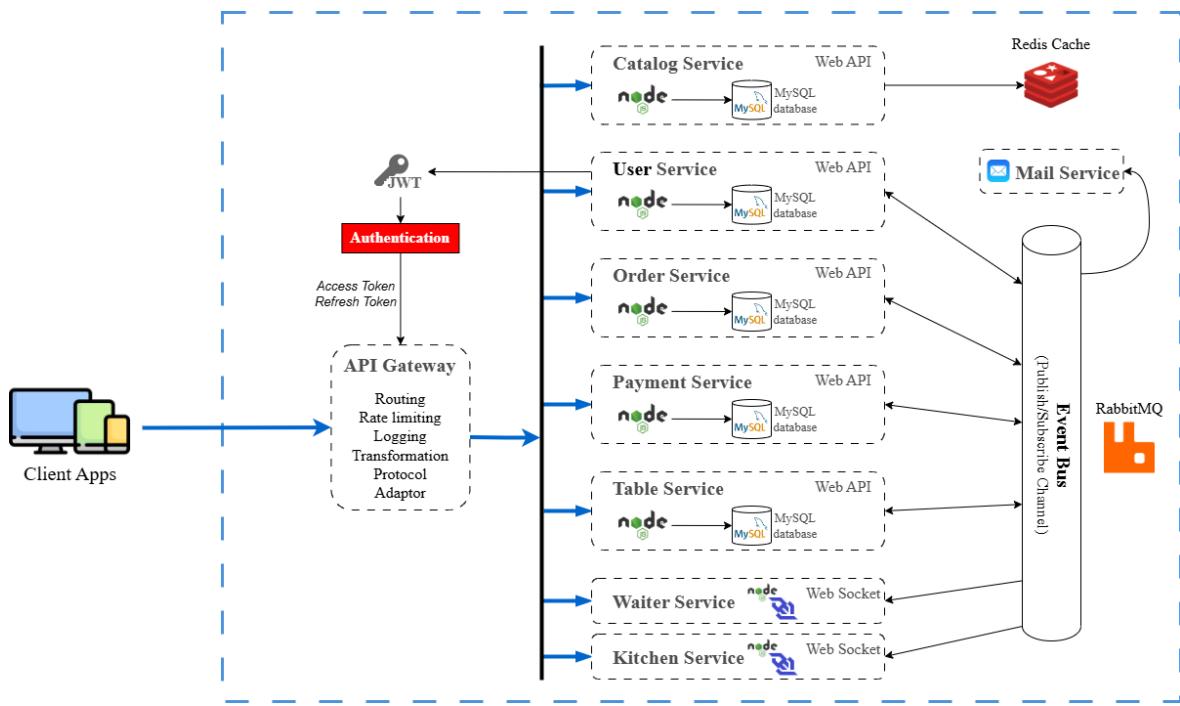
- *Giao tiếp với nhà bếp:* Chức năng này giúp truyền tải các đơn hàng và yêu cầu từ khách hàng đến các thiết bị của nhân viên phục vụ qua WebSocket, đảm bảo sự phối hợp chặt chẽ giữa nhân viên phục vụ và bếp.

Mail Service

- *Gửi email thông báo:* Dịch vụ này chịu trách nhiệm gửi các thông báo quan trọng như xác nhận đăng ký tài khoản, đặt lại mật khẩu hoặc các thông báo khác, đảm bảo khách hàng luôn được cập nhật thông tin kịp thời.

3.3.1.2 Sơ đồ kiến trúc tổng thể

Sau khi xác định và phân chia hệ thống thành các dịch vụ nhỏ hơn, chúng tôi xây dựng sơ đồ kiến trúc tổng thể để minh họa cách các dịch vụ này tương tác và phối hợp với nhau. Sơ đồ này cung cấp cái nhìn toàn diện về cấu trúc của hệ thống, bao gồm cách các thành phần được phân bố, kết nối và tích hợp, đảm bảo tính ổn định, khả năng mở rộng và dễ bảo trì.



Hình 3.10 Kiến trúc tổng thể

3.3.1.3 Giao tiếp giữa các dịch vụ

Các microservices trong hệ thống giao tiếp với nhau bằng cách gọi API trực tiếp qua mạng nội bộ hoặc sử dụng RabbitMQ để xử lý các tác vụ không đồng bộ. Khi gọi API trực tiếp, các dịch vụ có thể tương tác với nhau thông qua địa chỉ mạng nội bộ, giúp tối ưu hóa tốc độ phản hồi và giảm độ trễ so với việc phải đi qua một API Gateway trung gian. RabbitMQ được sử dụng để xử lý các tác vụ không đồng bộ và truyền thông điệp giữa các dịch vụ, cho phép các dịch vụ gửi và nhận thông điệp từ các hàng đợi RabbitMQ, đảm bảo tính linh hoạt và khả năng chịu tải cao.

3.3.1.4 Các công nghệ sử dụng

- *Framework:* Node.js (phiên bản 21.4.0) với Express.js (phiên bản 4.19.2).
- *WebSocket:* Gói thư viện 'ws' (phiên bản 8.17.1).
- *Authentication:* Gói thư viện 'jsonwebtoken' (phiên bản 9.0.2).
- *Cơ sở dữ liệu:* MySQL (phiên bản 9.0).
- *Message Broker:* RabbitMQ (phiên bản '3.13-management-alpine').
- *Caching:* Redis (phiên bản '7.2.5-alpine').

3.3.2 Thiết kế cơ sở dữ liệu

3.3.2.1 Phân chia cơ sở dữ liệu

Cơ sở dữ liệu trong hệ thống quản lý nhà hàng được thiết kế theo kiến trúc microservices, mỗi service có cơ sở dữ liệu riêng được phân chia như bên dưới.

Catalog Service

Bảng 3.20 Mô tả thực thể Category

Thuộc tính	Mô tả	Kiểu dữ liệu
ID (PK)	ID danh mục	INTEGER
Name	Tên danh mục	STRING
Description	Mô tả danh mục	STRING
Active	Trạng thái hoạt động của danh mục	BOOLEAN
CreatedAt	Ngày tạo danh mục	DATE
UpdatedAt	Ngày cập nhật danh mục	DATE

Bảng 3.21 Mô tả thực thể Item

Thuộc tính	Mô tả	Kiểu dữ liệu
ID (PK)	ID món ăn	INTEGER
CategoryID (FK)	ID danh mục liên kết	INTEGER
Name	Tên món ăn	STRING
Price	Giá món ăn	FLOAT
Image	Hình ảnh món ăn	STRING
Description	Mô tả món ăn	STRING
Available	Trạng thái phục vụ của món ăn	BOOLEAN
Active	Trạng thái hoạt động của món ăn	BOOLEAN
CreatedAt	Ngày tạo món ăn	DATE
UpdatedAt	Ngày cập nhật món ăn	DATE

Order Service

Bảng 3.22 Mô tả thực thể Order

<i>Thuộc tính</i>	<i>Mô tả</i>	<i>Kiểu dữ liệu</i>
ID (PK)	ID đơn hàng	INTEGER
UserID (FK)	ID người dùng liên kết	INTEGER
TableID (FK)	ID bàn liên kết	INTEGER
Status	Trạng thái phục vụ của đơn hàng ("đang phục vụ", "hoàn tất", "đã hủy")	ENUM
Active	Trạng thái hoạt động của đơn hàng	BOOLEAN
CreatedAt	Ngày tạo đơn hàng	DATE
UpdatedAt	Ngày cập nhật đơn hàng	DATE

Bảng 3.23 Mô tả thực thể OrderItem

<i>Thuộc tính</i>	<i>Mô tả</i>	<i>Kiểu dữ liệu</i>
ID (PK)	ID mục đơn hàng	INTEGER
OrderID (FK)	ID đơn hàng liên kết	INTEGER
ItemID (FK)	ID món ăn liên kết	INTEGER
Quantity	Số lượng món ăn	INTEGER
Price	Giá món ăn tại thời điểm đặt	FLOAT
Amount	Tổng số tiền cho món ăn (Quantity x Price)	FLOAT
Status	Trạng thái phục vụ của mục đơn hàng ("đang chờ", "đang chuẩn bị", "hoàn tất", "đã hủy")	ENUM
Active	Trạng thái hoạt động của đơn hàng	BOOLEAN
CreatedAt	Ngày tạo mục đơn hàng	DATE
UpdatedAt	Ngày cập nhật mục đơn hàng	DATE

Payment Service

Bảng 3.24 Mô tả thực thể Surcharge

<i>Thuộc tính</i>	<i>Mô tả</i>	<i>Kiểu dữ liệu</i>
ID (PK)	ID phụ phí	INTEGER
Name	Tên phụ phí	STRING
IsPercent	Có phải là phần trăm không	BOOLEAN
Value	Giá trị phụ phí	FLOAT
Description	Mô tả phụ phí	STRING
Active	Trạng thái hoạt động của phụ phí	BOOLEAN
CreatedAt	Ngày tạo phụ phí	DATE
UpdatedAt	Ngày cập nhật phụ phí	DATE

Bảng 3.25 Mô tả thực thể Payment

<i>Thuộc tính</i>	<i>Mô tả</i>	<i>Kiểu dữ liệu</i>
ID (PK)	ID thanh toán	INTEGER
UserID (FK)	ID người dùng liên kết	INTEGER
OrderID (FK)	ID đơn hàng liên kết	INTEGER
SubAmount	Số tiền phụ	FLOAT
TotalSurcharge	Tổng số tiền phụ phí	FLOAT
TotalDiscount	Tổng số tiền giảm giá	FLOAT
TotalAmount	Tổng số tiền thanh toán	FLOAT
Note	Ghi chú thanh toán	STRING
CreatedAt	Ngày tạo thanh toán	DATE
UpdatedAt	Ngày cập nhật thanh toán	DATE

Bảng 3.26 Mô tả thực thể PaymentSurcharge

Thuộc tính	Mô tả	Kiểu dữ liệu
ID (PK)	ID mục phụ phí của thanh toán	INTEGER
PaymentID (FK)	ID thanh toán liên kết	INTEGER
SurchargeID (FK)	ID phụ phí liên kết	INTEGER
Value	Giá trị phụ phí tại thời điểm áp dụng	FLOAT
Amount	Số tiền phụ phí	FLOAT
CreatedAt	Ngày tạo mục phụ phí của thanh toán	DATE
UpdatedAt	Ngày cập nhật mục phụ phí của thanh toán	DATE

Table Service

Bảng 3.27 Mô tả thực thể Table

Thuộc tính	Mô tả	Kiểu dữ liệu
ID (PK)	ID bàn	INTEGER
No	Số bàn	STRING
Capacity	Số chỗ ngồi	INTEGER
IsVip	Có phải là bàn VIP không	BOOLEAN
Status	Trạng thái phục vụ của bàn ("trống", "đang sử dụng", "đã đặt")	ENUM
Active	Trạng thái hoạt động của bàn	BOOLEAN
CreatedAt	Ngày tạo bàn	DATE
UpdatedAt	Ngày cập nhật bàn	DATE

User Service

Bảng 3.28 Mô tả thực thể Role

Thuộc tính	Mô tả	Kiểu dữ liệu
ID (PK)	ID vai trò	INTEGER
Name	Tên vai trò	STRING
Active	Trạng thái hoạt động của vai trò	BOOLEAN
CreatedAt	Ngày tạo vai trò	DATE
UpdatedAt	Ngày cập nhật vai trò	DATE

Bảng 3.29 Mô tả thực thể User

Thuộc tính	Mô tả	Kiểu dữ liệu
ID (PK)	ID người dùng.	INTEGER
RoleID (FK)	ID vai trò liên kết	INTEGER
FullName	Họ và tên	STRING
Gender	Giới tính	STRING
Phone	Số điện thoại	STRING
NationalID	Số CMND/CCCD	STRING
Gmail	Địa chỉ gmail của người dùng	STRING
Password	Mật khẩu người dùng	STRING
Active	Trạng thái hoạt động của người dùng	BOOLEAN
CreatedAt	Ngày tạo người dùng	DATE
UpdatedAt	Ngày cập nhật người dùng	DATE

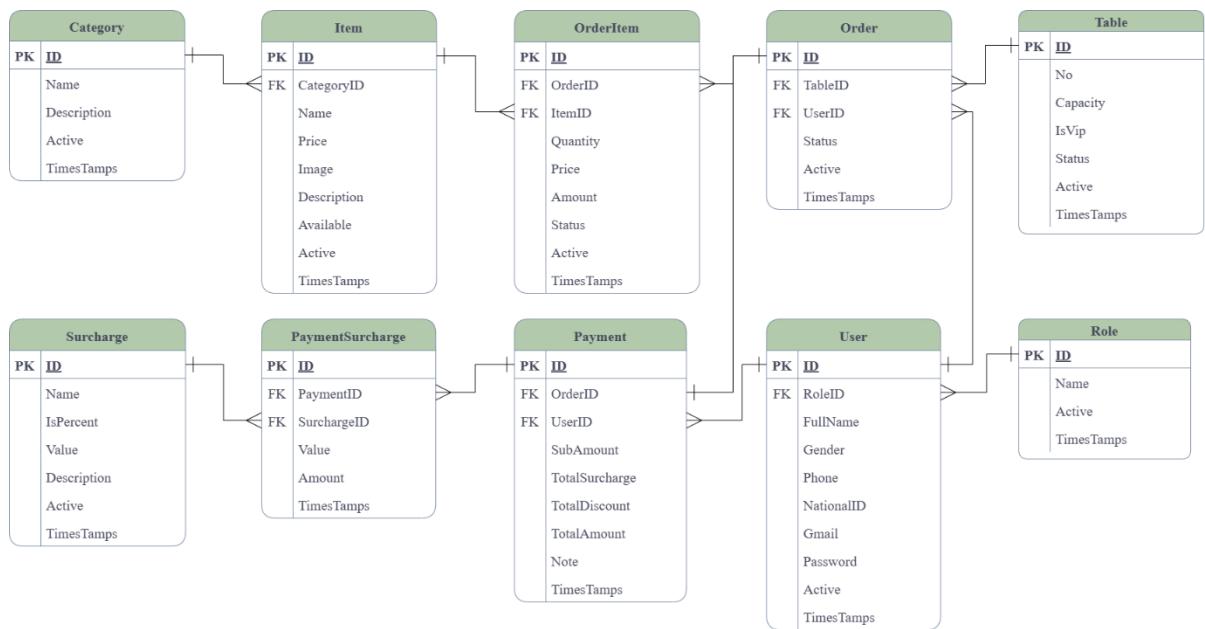
Kitchen Service: Không có bảng dữ liệu, xử lý các đơn hàng qua WebSocket.

Waiter Service: Không có bảng dữ liệu, giao tiếp với nhà bếp qua WebSocket.

Mail Service: Không có bảng dữ liệu, xử lý việc gửi email thông báo.

3.3.2.2 Sơ đồ quan hệ (ERD) tổng quát

Sau khi phân chia cơ sở dữ liệu theo các dịch vụ, chúng tôi xây dựng sơ đồ quan hệ (ERD) tổng quát để thể hiện các bảng và mối quan hệ giữa chúng. Sơ đồ này cung cấp cái nhìn tổng quan về cấu trúc dữ liệu, giúp đảm bảo tính nhất quán và hiệu quả trong việc quản lý dữ liệu của hệ thống.



Hình 3.11 Sơ đồ quan hệ tổng quát

3.3.3 Thiết kế API

Trong phần này, chúng tôi trình bày thiết kế các API cần thiết cho hệ thống quản lý nhà hàng. Các API được thiết kế nhằm đảm bảo giao tiếp hiệu quả giữa các microservices, cũng như với các ứng dụng khách. Mỗi API được mô tả chi tiết về các phương thức, điểm cuối (endpoints), và các tham số cần thiết, giúp các nhà phát triển dễ dàng tích hợp và sử dụng. Để tìm hiểu thêm chi tiết về thiết kế và tài liệu liên quan đến các API, vui lòng tham khảo tài liệu API theo liên kết dưới đây:

<https://documenter.getpostman.com/view/35381032/2sA3s4kVKE>

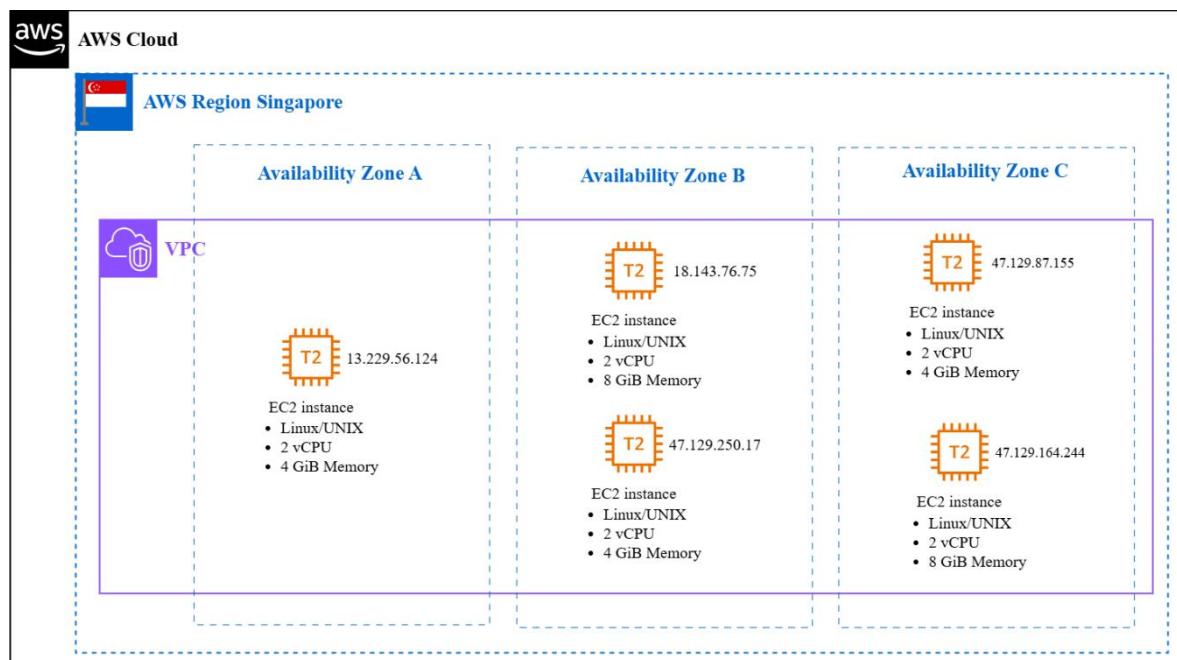
CHƯƠNG 4. THỰC NGHIỆM

4.1 Triển khai

4.1.1 Sơ đồ triển khai tổng quát

4.1.1.1 Triển khai EC2

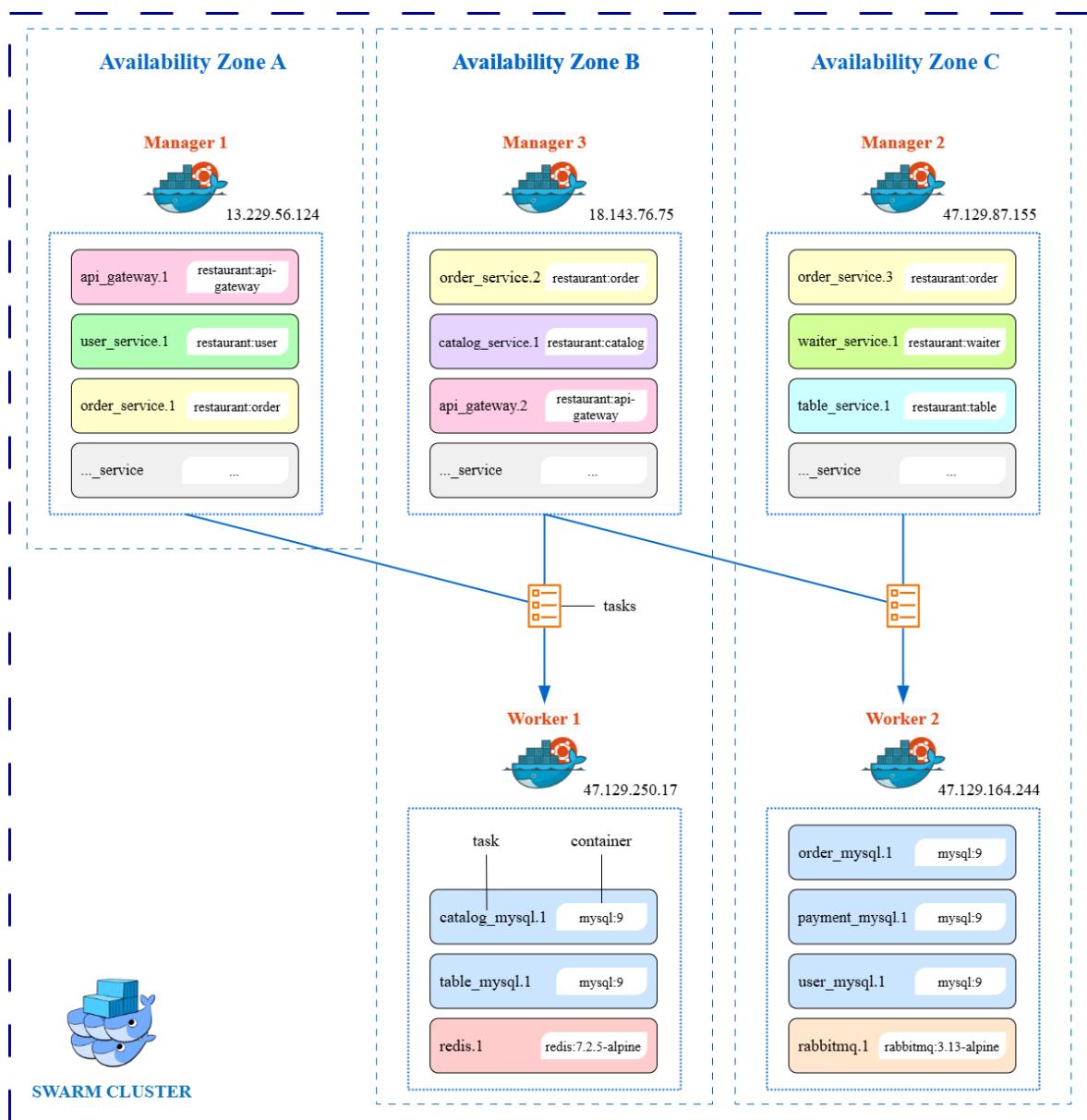
Phần này giới thiệu mô hình triển khai hệ thống trên Amazon EC2, với cấu trúc phân bổ tài nguyên trong 'AWS Cloud'. Hình ảnh dưới đây cho thấy việc triển khai trong 'AWS Region Singapore', bao gồm ba 'Availability Zones', mỗi zone chứa các EC2 instances. Điều này cho phép hệ thống đạt được sự phân tán cao và khả năng chịu lỗi tốt, đồng thời đảm bảo hiệu suất và tính sẵn sàng của dịch vụ.



Hình 4.1 Sơ đồ triển khai EC2

4.1.1.2 Triển khai cụm Docker Swarm

Phần này trình bày mô hình triển khai mong muốn cho cụm Docker Swarm trên Amazon EC2. Hình ảnh mô tả sự phân bố của các nodes trong cụm Swarm trong 'AWS Region Singapore', tương tự như cách triển khai EC2. Các manager và worker nodes được phân phối trong các 'Availability Zones' khác nhau để tối ưu hóa khả năng mở rộng và độ tin cậy. Triển khai này tận dụng cơ sở hạ tầng đám mây AWS để hỗ trợ việc quản lý và triển khai các dịch vụ microservices hiệu quả.



Hình 4.2 Sơ đồ triển khai cụm Docker Swarm

4.1.2 Cài đặt Docker Swarm

Trong phần này, chúng tôi sẽ mô tả quá trình cài đặt Docker Swarm trên các instances Amazon EC2, bao gồm việc khởi tạo cụm swarm và thêm các worker nodes vào cluster. Ngoài ra, phần này sẽ giải thích lý do tại sao sử dụng 3 node làm manager với thuật toán đồng thuận Raft và lý do phân bổ 3 manager node này vào các availability zone khác nhau.

4.1.2.1 Mô tả quá trình cài đặt Docker Swarm

Bước 1: Khởi tạo các EC2 instances

Trước tiên, chúng tôi thuê các EC2 instances tại vùng Asia Pacific (Singapore) với ID vùng là ap-southeast-1. Trong thực nghiệm này, chúng tôi sử dụng 5 EC2 instances, được phân bổ như sau:

- + Lead - Manager 1: ap-southeast-1a
- + Manager 2: ap-southeast-1c
- + Manager 3: ap-southeast-1b
- + Worker 1: ap-southeast-1b
- + Worker 2: ap-southeast-1c

Điều này có nghĩa là chúng tôi sẽ thiết lập một Swarm cluster gồm 5 nodes với 3 manager nodes và 2 worker nodes:

Name	Instance ID	Instance state	Instance type	Availability Zone	Public IPv4 address
Leader Manager 1	i-0682823...	Running	t2.medium	ap-southeast-1a	13.229.56.124
Manager 2	i-0418529...	Running	t2.medium	ap-southeast-1c	47.129.87.155
Manager 3	i-008a06c7...	Running	t2.large	ap-southeast-1b	18.143.76.75
Worker 1	i-04cd10fe...	Running	t2.medium	ap-southeast-1b	47.129.250.17
Worker 2	i-0c32ab7a...	Running	t2.large	ap-southeast-1c	47.129.164.244

Hình 4.3 Khởi tạo EC2 instances

Bước 2: Cài đặt Docker trên mỗi EC2 instance

Sau khi khởi tạo các EC2 instances, chúng tôi tiến hành cài đặt Docker trên từng instance và kiểm tra phiên bản Docker:

```

root@manager-2:/home/ubuntu# docker info
Client:
  Version: 24.0.7
  Context: default
  Debug Mode: false
  Server:
    root@leader-manager-1:/home/ubuntu# docker info
    Client:
      Version: 24.0.7
      Context: default
      Debug Mode: false
      Images:
        Server:
          Containers: 0
          Running: 0
          Paused: 0
          Stopped: 0
          Images: 0
          Server Version: 24.0.7
          Storage Driver: overlay2
          Backing Filesystem: extfs
          Supports d_type: true
          Using metacopy: false
          Paused: 0
          Stopped: 0
          Images: 0
    Client:
      Version: 24.0.7
      Context: default
      Debug Mode: false
      Images:
        Server:
          Containers: 0
          Running: 0
          Paused: 0
          Stopped: 0
          Images: 0
          Server Version: 24.0.7
          Storage Driver: overlay2
          Backing Filesystem: extfs
          Supports d_type: true
          Using metacopy: false
          Paused: 0
          Stopped: 0
          Images: 0
    Client:
      Version: 24.0.7
      Context: default
      Debug Mode: false
      Images:
        Server:
          Containers: 0
          Running: 0
          Paused: 0
          Stopped: 0
          Images: 0
          Server Version: 24.0.7
          Storage Driver: overlay2
          Backing Filesystem: extfs
          Supports d_type: true
          Using metacopy: false
          Paused: 0
          Stopped: 0
          Images: 0
  
```

Hình 4.4 Cài đặt Docker trên từng node

Bước 3: Khởi tạo cụm Docker Swarm

Chúng tôi xác định node để khởi tạo swarm là Leader - Manager 1. Trên node này, ta thực hiện lệnh tạo một cụm Docker Swarm mới và chỉ định node hiện tại làm manager node đầu tiên. Docker sẽ trả về một lệnh docker swarm join mà chúng ta sẽ sử dụng để thêm các node khác vào cụm Swarm:

```

root@leader-manager-1:/home/ubuntu
Swarm initialized: current node (2fy...fpt) is now a manager.

To add a worker to this swarm, run the following command:

  docker swarm join --token SWMTKN-1-5agj199ez5ib7vwbgyd388uipcfis1jph5nkyj1pgk8uxp0lut
  -2i1jpgzbvn16vzzwyhm8ieb3l 13.229.56.124:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

root@leader-manager-1:/home/ubuntu# docker swarm join-token manager
To add a manager to this swarm, run the following command:

  docker swarm join --token SWMTKN-1-5agj199ez5ib7vwbgyd388uipcfis1jph5nkyj1pgk8uxp0lut
  -bxwula4a0i6fabvmy88j6pcs4 13.229.56.124:2377
  
```

Hình 4.5 Khởi tạo cụm Swarm

Bước 4: Thêm các manager và worker nodes vào swarm

Trên các node còn lại (bao gồm 2 manager nodes và 2 worker nodes), ta chạy lệnh docker swarm join với token do node manager đầu tiên cung cấp để có thể tham gia vào cụm Swarm:

```

root@manager-2:/home/ubuntu# docker swarm join --token SWMTKN-1-5agjl99ez5ib7vwbgyd388uipcfis1jp h5nkyj1pgk8uxp0lut-bxwula4a0i6fabvmy88j6pcs4 13.229.56.124:2377
This node joined a swarm as a manager.
root@manager-2:/home/ubuntu# 

root@manager-3:/home/ubuntu# docker swarm join --token SWMTKN-1-5agjl99ez5ib7vwbgyd388uipcfis1jp h5nkyj1pgk8uxp0lut-bxwula4a0i6fabvmy88j6pcs4 13.229.56.124:2377
This node joined a swarm as a manager.
root@manager-3:/home/ubuntu# 

root@worker-1:/home/ubuntu# docker swarm join --token SWMTKN-1-5agjl99ez5ib7vwbgyd388uipcfis1jph5nkyj1pgk8uxp0lut-2i1jpgzbnl6vzzwyhm8ieb31 13.229.56.124:2377
This node joined a swarm as a worker.
root@worker-1:/home/ubuntu# 

root@worker-2:/home/ubuntu# docker swarm join --token SWMTKN-1-5agjl99ez5ib7vwbgyd388uipcfis1jph5nkyj1pgk8uxp0lut-2i1jpgzbnl6vzzwyhm8ieb31 13.229.56.124:2377
This node joined a swarm as a worker.
root@worker-2:/home/ubuntu# 

```

Hình 4.6 Thêm các manager và worker nodes vào swarm

Bước 5: Kiểm tra

Sau khi tất cả các node đã được thêm vào, chúng ta có thể kiểm tra trạng thái của swarm trên bất kỳ manager node nào. Lệnh này sẽ liệt kê tất cả các nodes trong swarm và chỉ ra node nào đang hoạt động với vai trò gì (manager hoặc worker):

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE V
2fyti... * (leader)	leader-manager-1	Ready	Active	Leader	24.0.7
arxeoc... (regular)	manager-2	Ready	Active	Reachable	24.0.7
10pncj... (regular)	manager-3	Ready	Active	Reachable	24.0.7
amivx... (regular)	worker-1	Ready	Active		24.0.7
n8a5nbhb... (regular)	worker-2	Ready	Active		24.0.7

Hình 4.7 Danh sách các manager và worker nodes

4.1.2.2 Available zones và lợi ích của việc phân bổ node

Khi triển khai một Docker Swarm cluster, đặc biệt là trong môi trường sản xuất, điều quan trọng là đảm bảo tính khả dụng cao và khả năng chịu lỗi. Trong thực nghiệm này, chúng ta đã phân bổ 3 manager nodes vào 3 availability zone khác nhau (ap-southeast-1a, ap-southeast-1b, ap-southeast-1c).

Lợi ích của việc này bao gồm:

- *Tăng khả năng chịu lỗi:* Bằng cách phân bổ các manager nodes vào các availability zone khác nhau, chúng ta giảm thiểu nguy cơ bị gián đoạn dịch vụ do lỗi hạ tầng tại một vùng cụ thể. Nếu một availability zone gặp sự cố, các manager nodes ở các zone khác vẫn có thể duy trì hoạt động của toàn bộ cluster.
- *Cải thiện độ tin cậy và tính nhất quán:* Sự phân bổ này đảm bảo rằng Docker Swarm cluster có thể tiếp tục hoạt động ngay cả khi một hoặc nhiều zone gặp sự cố, miễn là vẫn có đủ số lượng manager node để duy trì đa số (quorum).

4.1.2.3 Cơ chế Raft và lý do sử dụng 3 manager nodes

Raft Consensus Algorithm

Thuật toán Raft yêu cầu một đa số (quorum) các manager nodes đồng ý trước khi thực hiện bất kỳ thay đổi nào về trạng thái của swarm. Quorum thường được xác định là $(N/2) + 1$, trong đó N là số lượng manager nodes trong cluster.

Khi Docker Engine chạy ở chế độ Swarm, các manager nodes sử dụng thuật toán Raft để đảm bảo rằng tất cả các manager chịu trách nhiệm quản lý và lập lịch cho các tác vụ trong cluster đều lưu trữ cùng một trạng thái nhất quán. Điều này có nghĩa là, trong trường hợp có sự cố xảy ra, bất kỳ manager node nào cũng có thể tiếp nhận các tác vụ và khôi phục dịch vụ về trạng thái ổn định.

Ví dụ, nếu Leader Manager - node chịu trách nhiệm lập lịch tác vụ trong cluster - đột ngột gặp sự cố, bất kỳ manager node nào khác cũng có thể tiếp nhận nhiệm vụ lập lịch và cân bằng lại các tác vụ để đảm bảo trạng thái mong muốn.

Tính chịu lỗi và yêu cầu về quorum

Raft đảm bảo rằng trạng thái của cluster vẫn nhất quán trong trường hợp có lỗi bằng cách yêu cầu một đa số các node phải đồng ý về các giá trị được đề xuất cho cluster. Thuật toán này có thể chịu được $(N-1)/2$ lỗi, tức là nếu có 5 manager nodes, hệ thống có thể chịu được sự cố của 2 nodes mà vẫn duy trì được quorum.

Điều này có nghĩa là trong một cluster gồm 5 manager nodes chạy Raft, nếu 3 node không khả dụng, hệ thống sẽ không thể xử lý thêm bất kỳ yêu cầu nào để lập lịch các tác vụ mới. Các tác vụ hiện có vẫn tiếp tục chạy, nhưng trình lập lịch không thể cân bằng lại các tác vụ để đối phó với các sự cố nếu tập hợp các manager không ở trạng thái tốt.

Lý do sử dụng 3 manager nodes

Sử dụng 3 manager nodes là một lựa chọn phổ biến vì nó cung cấp sự cân bằng giữa tính sẵn sàng và chi phí tài nguyên. Với 3 manager nodes, cluster có thể chịu được sự thất bại của 1 node mà vẫn duy trì được quorum. Cụ thể:

- Nếu chỉ có 1 manager node, sự cố xảy ra sẽ làm mất toàn bộ cluster.
- Với 2 manager nodes, sự cố ở 1 node sẽ khiến cluster mất quorum, và không thể thực hiện bất kỳ thay đổi nào cho đến khi node bị lỗi trở lại hoạt động.
- Với 3 manager nodes, ngay cả khi 1 node bị lỗi, cluster vẫn duy trì được quorum với 2 node còn lại, cho phép tiếp tục hoạt động mà không bị gián đoạn.

4.1.3 Cấu hình dịch vụ

Trong dự án của chúng tôi, Docker Compose đã được sử dụng để cấu hình và triển khai các dịch vụ khác nhau như MySQL, RabbitMQ, Redis, và các microservice của hệ thống quản lý nhà hàng. Tệp docker-compose.yml này giúp quản lý toàn bộ hệ thống một cách dễ dàng, cho phép khởi động tất cả các dịch vụ chỉ bằng một lệnh.

4.1.3.1 Networking & Service Discovery (Mạng và Khám phá dịch vụ)

Trong Docker Compose, khái niệm networking rất quan trọng, đặc biệt là trong một hệ thống phân tán như microservices. Docker Compose tự động tạo ra một mạng lưới riêng (default network) cho các dịch vụ trong cùng một tệp Compose, giúp các container có thể giao tiếp với nhau thông qua tên dịch vụ (service name). Điều này giúp cho việc triển khai hệ thống trở nên dễ dàng hơn, vì ta không cần phải biết trước địa chỉ IP của các container, mà chỉ cần sử dụng tên dịch vụ để kết nối giữa các container với nhau.

Service Discovery là cơ chế mà Docker sử dụng để các container trong cùng một mạng có thể tìm và kết nối với nhau bằng cách sử dụng tên dịch vụ thay vì địa chỉ IP cụ thể. Khi một container mới được khởi chạy trong mạng của Docker Compose, nó sẽ được đăng ký vào hệ thống DNS nội bộ, cho phép các container khác có thể kết nối tới nó thông qua tên dịch vụ mà bạn đã định nghĩa trong tệp Compose.

Giải thích cơ chế hoạt động trong docker-compose.yml:

```

networks:
  restaurant_net:
    driver: overlay

services:
  catalog_mysql:
    image: mysql:9
    networks:
      - restaurant_net

  catalog_service:
    image: deviannnn/restaurant:catalog
    environment:
      - NODE_ENV=production
      - PORT=5000
      - PROD_REDIS_HOST=redis_service
      - PROD_DB_USERNAME=root
      - PROD_DB_PASSWORD=123456
      - PROD_DB_NAME=restaurant_system_catalog
      - PROD_DB_HOSTNAME=catalog_mysql
    networks:
      - restaurant_net

```

Hình 4.8 Phần cấu hình về mạng và khám phá dịch vụ

- networks: Mạng lưới riêng (ví dụ: *restaurant_net*) được tạo ra để các container trong cùng một mạng có thể giao tiếp với nhau. Mạng overlay là loại mạng thường được sử dụng trong Docker Swarm, giúp kết nối các container trên các node khác nhau trong cùng một Swarm.
- Trong phần cấu hình này, dịch vụ *catalog_mysql* và *catalog_service* được kết nối vào mạng *restaurant_net*. Điều này có nghĩa là *catalog_service* có thể giao tiếp với *catalog_mysql* riêng và các dịch vụ khác cũng nằm trong mạng *restaurant_net* nói chung.

Trong hệ thống của chúng tôi, tất cả các dịch vụ đều được kết nối qua mạng *restaurant_net*. Ví dụ, dịch vụ *catalog_service* có thể kết nối tới cơ sở dữ liệu MySQL của nó thông qua tên dịch vụ *catalog_mysql* trong định nghĩa ‘environment’, mà không cần phải biết địa chỉ IP cụ thể của container đó.

4.1.3.2 Resource Allocation (Phân bổ tài nguyên)

Docker Compose cho phép cấu hình phân bổ tài nguyên cho các container, đảm bảo mỗi dịch vụ chỉ sử dụng một lượng tài nguyên nhất định. Điều này giúp tối ưu hóa hiệu suất, tránh thiếu hụt tài nguyên, và đảm bảo hoạt động ổn định cho các dịch vụ bằng cách đặt giới hạn và mức ưu tiên cho CPU và bộ nhớ.

Giải thích cơ chế hoạt động trong docker-compose.yml:

```

deploy:
  resources:
    reservations:
      cpus: '0.3'
      memory: 384M
    limits:
      cpus: '0.5'
      memory: 512M
    placement:
      constraints:
        - node.role == worker
  
```

Hình 4.9 Phần cấu hình về phân bổ tài nguyên

- *resources*: Cấu hình tài nguyên cho dịch vụ, ngăn chặn việc tiêu thụ quá mức ảnh hưởng đến các dịch vụ khác.
 - + *reservations*: Định nghĩa tài nguyên tối thiểu yêu cầu. Trong phần cấu hình này, dịch vụ yêu cầu ít nhất 0.3 CPUs và 384MB bộ nhớ.
 - + *limits*: Đặt giới hạn tối đa tài nguyên sử dụng. Trong phần cấu hình này, dịch vụ không được sử dụng quá 0.5 CPUs và 512MB bộ nhớ.
- *placement*: Xác định node chạy dịch vụ để cân bằng tải.
 - + *constraints*: Đặt các điều kiện để dịch vụ chỉ chạy trên các node đáp ứng yêu cầu cụ thể. Trong phần cấu hình này, dịch vụ dịch vụ bắt buộc phải được phân phối các container trên các node "worker".

4.1.3.3 Monitoring (Giám sát)

Monitoring là yếu tố quan trọng để duy trì sức khỏe hệ thống. Trong Docker Compose, chúng tôi thiết lập các chính sách khởi động lại và kiểm tra sức khỏe cho container để tự động khởi động lại dịch vụ khi lỗi và đảm bảo các dịch vụ hoạt động ổn định.

Giải thích cơ chế hoạt động trong docker-compose.yml:

```

services:
  catalog_mysql:
    image: mysql:9
    deploy:
      restart_policy:
        condition: any
        delay: 0s
    healthcheck:
      test: [ "CMD-SHELL", "mysqladmin ping -h127.0.0.1 -uroot -p123456" ]
      interval: 30s
      timeout: 10s
      retries: 3

  catalog_service:
    image: deviannnn/restaurant:catalog
    deploy:
      restart_policy:
        condition: on-failure
        delay: 5s
    healthcheck:
      test: [ "CMD", "curl", "-f", "http://localhost:5000/health" ]
      interval: 30s
      timeout: 10s
      retries: 3

```

Hình 4.10 Phần cấu hình về giám sát

- *restart_policy*: Định nghĩa chính sách khởi động lại dịch vụ khi gặp sự cố.
Trong phần cấu hình này:
 - + ‘*condition: any*’: Container sẽ được khởi động lại trong mọi tình huống, bất kể lý do dừng.
 - + ‘*condition: on-failure*’: Container khởi động lại khi gặp lỗi và dừng hoạt động.
- *healthcheck*: Kiểm tra sức khỏe của container. Trong phần cấu hình này:
 - + Đối với *catalog_mysql*: Dùng lệnh mysqladmin ping để kiểm tra MySQL; nếu không phản hồi sau 3 lần thử với khoảng cách 30 giây, container sẽ bị đánh dấu là không hoạt động.
 - + Đối với *catalog_service*: Dùng lệnh curl để kiểm tra dịch vụ tại <http://localhost:5000/health>; nếu không phản hồi sau 3 lần thử, container sẽ bị đánh dấu là không hoạt động.

4.1.3.4 Deployment Strategy (Chiến lược triển khai)

Chiến lược triển khai trong Docker Compose xác định cách triển khai, cập nhật và khôi phục dịch vụ. Docker Compose cho phép kiểm soát số lượng bản sao, hành động khi cập nhật thất bại, và cách khởi động hoặc dừng dịch vụ mới.

Giải thích cơ chế hoạt động trong docker-compose.yml:

```
deploy:
  replicas: 3
  update_config:
    parallelism: 2
    delay: 10s
    failure_action: rollback
    monitor: 10s
    max_failure_ratio: 0.1
    order: start-first
  rollback_config:
    parallelism: 2
    delay: 0s
    failure_action: continue
    monitor: 0s
    max_failure_ratio: 0.2
    order: stop-first
```

Hình 4.11 Phân cấu hình về chiến lược triển khai

- *replicas*: Phần này cho phép ta định nghĩa số lượng bản sao của dịch vụ được triển khai. Trong phần cấu hình này, 3 bản sao của dịch vụ sẽ được khởi chạy.
- *update_config*: Phần này cho phép ta cấu hình cho quá trình cập nhật dịch vụ. Trong phần cấu hình này:
 - + *parallelism*: Cho phép cập nhật 2 bản sao dịch vụ song song.
 - + *delay*: Đợi 10 giây giữa các đợt cập nhật bản sao.
 - + *failure_action*: Nếu cập nhật thất bại, dịch vụ sẽ quay lại phiên bản trước.
 - + *monitor*: 10 giây sẽ là khoảng thời gian giám sát cập nhật để xác định sự cố.
 - + *max_failure_ratio*: Tỷ lệ thất bại tối đa là 10% trước khi cập nhật bị coi là thất bại.
 - + *order*: Bản sao mới sẽ được khởi động trước khi dừng bản sao cũ.
- *rollback_config*: Phần này cho phép ta cấu hình cho quá trình khôi phục nếu cập nhật thất bại. Trong phần cấu hình này:
 - + *parallelism*: Rollback sẽ được thực hiện song song trên 2 bản sao.
 - + *delay*: Không đợi giữa các đợt rollback.
 - + *failure_action*: Nếu rollback gặp lỗi, quá trình rollback vẫn tiếp tục.
 - + *monitor*: Không giám sát quá trình rollback.
 - + *max_failure_ratio*: Tỷ lệ thất bại tối đa cho rollback là 20%.
 - + *order*: Bản sao cũ sẽ dừng trước khi khởi động bản sao mới trong rollback.

4.1.3.5 Environment Variables (Biến môi trường)

Biến môi trường giúp cấu hình các dịch vụ một cách linh hoạt, dễ dàng thay đổi mà không cần sửa đổi tệp cấu hình ứng dụng. Điều này rất hữu ích khi bạn chuyển từ môi trường phát triển sang môi trường sản xuất.

Giải thích cơ chế hoạt động trong docker-compose.yml:

```
order_service:
  environment:
    - NODE_ENV=production
    - PORT=5000
    - PROD_RABBITMQ_USERNAME=admin
    - PROD_RABBITMQ_PASSWORD=admin
    - PROD_RABBITMQ_HOSTNAME=rabbitmq_service
    - PROD_DB_USERNAME=root
    - PROD_DB_PASSWORD=123456
    - PROD_DB_NAME=restaurant_system_order
    - PROD_DB_HOSTNAME=order_mysql
    - CATALOG_SERVICE_HOSTNAME=catalog_service
    - TABLE_SERVICE_HOSTNAME=table_service
    - CATALOG_SERVICE_PORT=5000
    - TABLE_SERVICE_PORT=5000
```

Hình 4.12 Phản ứng về môi trường

- *environment*: Định nghĩa các biến môi cho dịch vụ *order_service*.
- NODE_ENV: Thiết lập môi trường chạy là production.
- PORT: Dịch vụ sẽ chạy trên cổng 5000.
- PROD_RABBITMQ_HOSTNAME: Tên host của RabbitMQ service.
- PROD_DB_HOSTNAME: Tên host của MySQL service.
- CATALOG_SERVICE_HOSTNAME: Tên host của catalog_service.
- TABLE_SERVICE_HOSTNAME: Tên host của table_service.
- Các biến môi trường còn lại ...

4.2 Kết quả thực nghiệm

4.2.1 Kiểm tra chức năng

Kịch bản kiểm tra:

- Quản lý người dùng: Đăng ký, đăng nhập, quản lý hồ sơ người dùng,...
- Quản lý thực đơn: Hiển thị thực đơn theo nhiều định dạng, quản lý trạng thái của món ăn,...
- Quản lý đơn hàng: Tạo đơn hàng, thêm món ăn, cập nhật món ăn,...
- Thanh toán: Xử lý thanh toán, quản lý thống kê thanh toán,...

Kết quả:

- Tất cả các chức năng này đều hoạt động ổn định, các service có thể giao tiếp với nhau thông qua mạng nội bộ hoặc rabbitmq mà không gặp lỗi trong quá trình kiểm tra.

4.2.2 Kiểm tra khả năng giám sát

Để đảm bảo tính ổn định và dễ dàng quản lý, hệ thống đã được tích hợp tính năng kiểm tra sức khỏe (health checks) cho các dịch vụ.

Kịch bản kiểm tra:

- Kiểm tra logs của các service sau khi deploy stack
- Kiểm tra logs của các service xuyên suốt quá trình chạy chương trình

Kết quả:

- Hệ thống giám sát và khởi động lại các dịch vụ không phản hồi kịp thời, đảm bảo tính sẵn sàng cao.

4.2.3 Kiểm tra tính năng độ trễ thấp (Low Latency)

Rolling Updates: Docker Swarm cho phép cập nhật dịch vụ mà không cần dừng toàn bộ hệ thống. Bằng cách sử dụng cấu hình update_config trong docker-compose.yml, các dịch vụ được cập nhật theo cách tuần tự với các tham số như parallelism và delay. Điều này đảm bảo rằng không có downtime trong suốt quá trình cập nhật, và các dịch vụ vẫn tiếp tục phục vụ yêu cầu của người dùng.

Kịch bản kiểm tra:

- Cập nhật dịch vụ order_service với phiên bản mới.
- Kiểm tra phản hồi của dịch vụ trong suốt quá trình cập nhật.

Kết quả:

- Các dịch vụ được cập nhật mà không gây ra bất kỳ sự gián đoạn nào, đảm bảo độ trễ thấp cho người dùng cuối.

4.2.4 Kiểm tra tính năng cân bằng tải (Load Balancing)

Load Balancing: Docker Swarm thực hiện cân bằng tải giữa các replicas của dịch vụ để đảm bảo phân phối công việc đồng đều. Điều này giúp hệ thống có khả năng chịu tải tốt hơn và giảm tải cho từng container.

Kịch bản kiểm tra:

- Gửi một lượng lớn các yêu cầu API tới order_service.
- Kiểm tra sự phân phối yêu cầu giữa các replicas.

Kết quả:

- Yêu cầu được phân phối đồng đều giữa các replicas, hệ thống hoạt động trơn tru đảm bảo sự cân bằng tải.

4.2.5 Kiểm tra khả năng mở rộng (Scalable)

Service Replication: Docker Swarm cho phép tăng số lượng bản sao của một dịch vụ để cải thiện khả năng chịu tải và tính sẵn sàng. Trong phần này chúng tôi sẽ thực hiện so sánh hiệu suất giữa 1 với 7 bản sao của order_service trong các kịch bản kiểm tra như bên dưới.

Kịch bản kiểm tra:

- Đóng khách: Thực hiện kiểm tra tải với 200 người dùng, tăng thêm 2 người dùng mỗi giây và gửi yêu cầu liên tục trong 3 phút.
- Giờ cao điểm: Thực hiện kiểm tra tải với 1200 người dùng, tăng thêm 10 người dùng mỗi giây và gửi yêu cầu liên tục cho đến khi đạt đủ lượng người dùng.
- Tăng đột biến: Thực hiện kiểm tra tải với 1200 người dùng, tăng thêm 100 người dùng mỗi giây và gửi yêu cầu liên tục cho đến khi đạt đủ số lượng người dùng.

Kết quả:

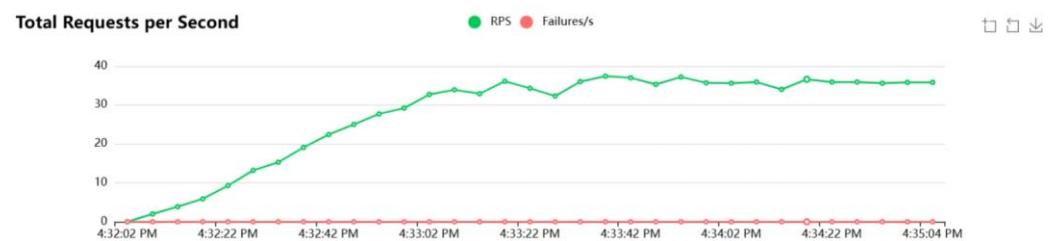
- Đóng khách: Với 7 bản sao, hệ thống xử lý gần gấp đôi tổng số lượng yêu cầu và Requests Per Second (RPS) so với 1 bản sao. Thời gian phản hồi trung bình của hệ thống với 7 bản sao là 124.47 ms, thấp hơn đáng kể so với 1,5 giây của hệ thống chỉ với 1 bản sao.

Locust Test Report

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/api/orders/5/add-items	4410	0	1583.18	93	3299	756.71	29.17	0
	Aggregated	4410	0	1583.18	93	3299	756.71	29.17	0

Charts



Hình 4.13 Thống kê và biểu đồ trong kịch bản ‘đóng khách’ của 1 replicas

Locust Test Report

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/api/orders/5/add-items	8421	0	124.47	89	1133	758.34	46.78	0
	Aggregated	8421	0	124.47	89	1133	758.34	46.78	0

Charts



Hình 4.14 Thống kê và biểu đồ trong kịch bản ‘đóng khách’ của 7 replicas

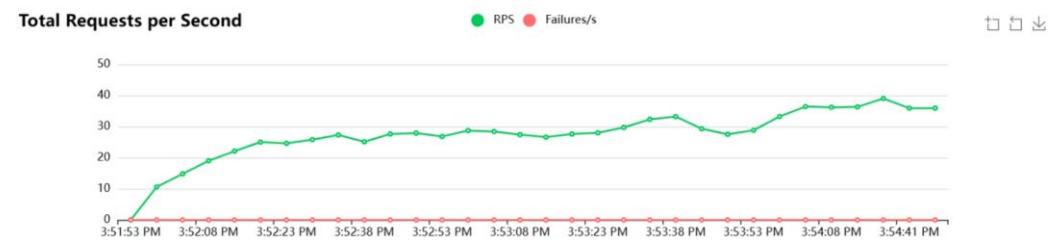
- Giờ cao điểm: Với 7 bản sao, hệ thống xử lý nhiều hơn gấp ba tổng số lượng yêu cầu và Requests Per Second (RPS) so với 1 bản sao. Thời gian phản hồi trung bình giảm từ 18 giây xuống dưới 4 giây. Tuy nhiên, hệ thống gặp phải 60 lỗi trên 12,877 yêu cầu khi sử dụng 7 bản sao.

Locust Test Report

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/api/orders/5/add-items	4275	0	18373.59	108	31028	752.43	28.59	0
	Aggregated	4275	0	18373.59	108	31028	752.43	28.59	0

Charts



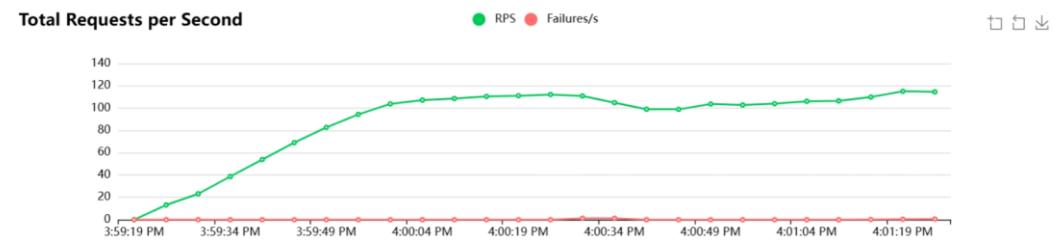
Hình 4.15 Thống kê và biểu đồ trong kịch bản ‘giờ cao điểm’ của 1 replicas

Locust Test Report

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/api/orders/5/add-items	12877	60	3701.9	93	23072	749.75	95.47	0.44
	Aggregated	12877	60	3701.9	93	23072	749.75	95.47	0.44

Charts



Hình 4.16 Thống kê và biểu đồ trong kịch bản ‘giờ cao điểm’ của 7 replicas

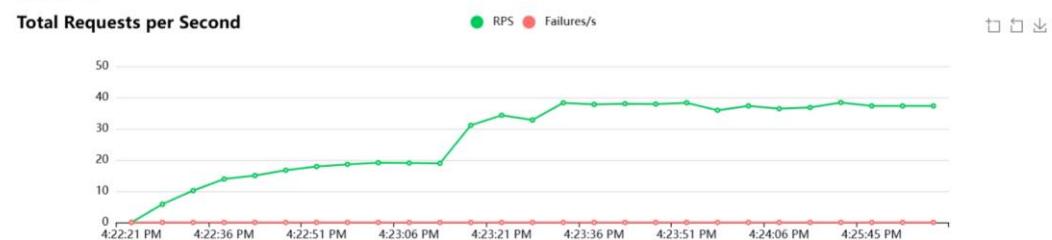
- Tăng đột biến: Trong tình huống này, hệ thống gặp phải độ trễ lớn trong việc đáp ứng yêu cầu. Tuy nhiên, với 7 bản sao, hệ thống vẫn xử lý nhiều yêu cầu hơn so với 1 bản sao. Dưới đây là hình ảnh và biểu đồ minh họa hiệu suất của hệ thống trong tình huống này.

Locust Test Report

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/api/orders/5/add-items	3398	0	32680.21	2058	60888	754	28.22	0
	Aggregated	3398	0	32680.21	2058	60888	754	28.22	0

Charts



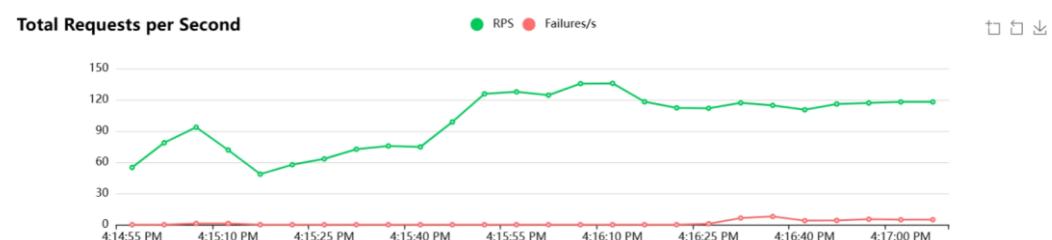
Hình 4.17 Thống kê và biểu đồ trong kịch bản ‘tăng đột biến’ của 1 replicas

Locust Test Report

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/api/orders/5/add-items	12171	193	8025.76	108	35911	742.12	101.09	1.6
	Aggregated	12171	193	8025.76	108	35911	742.12	101.09	1.6

Charts



Hình 4.18 Thống kê và biểu đồ trong kịch bản ‘tăng đột biến’ của 7 replicas

4.2.6 Kiểm tra khả năng khắc phục sự cố

Raft Consensus & Node Failures: Kiểm tra cơ chế Raft và khả năng phục hồi khi có sự cố liên quan đến các node manager và worker.

Kịch bản kiểm tra:

- Tắt Node Worker: Xem xét cách Docker Swarm xử lý việc tắt một node worker và ảnh hưởng đến các dịch vụ stateless và stateful.
- Tắt Node Manager: Kiểm tra cách Docker Swarm xử lý việc phân phối task và bầu ra leader mới khi tắt một hoặc hai node manager.

Kết quả:

- Tắt Node Worker:
 - + Dịch vụ Stateless: Các dịch vụ này hoạt động tốt sau khi được khôi phục trên node khác, không gặp gián đoạn và duy trì tính sẵn sàng cao.
 - + Dịch vụ Stateful: Mất đi tài nguyên và dữ liệu trước đó do sự phụ thuộc vào trạng thái lưu trữ cục bộ. Sau khi dịch vụ được khôi phục trên node khác, dữ liệu trước đó không còn và cần phải xử lý lại từ đầu.
- Tắt Node Manager:
 - + Tắt Một Node Manager: Docker Swarm có thể bầu ra một leader mới để duy trì hoạt động của cluster và đảm bảo sự phân phối task đúng cách. Các container ở các node còn lại vẫn còn hoạt động bình thường.
 - + Tắt Hai Node Manager: Khi tắt hai node manager, cơ chế Raft sẽ không còn khả năng duy trì đồng thuận do thiếu số lượng node cần thiết. Điều này có thể dẫn đến khả năng mất quản lý cluster và không phân phối được task một cách chính xác. Mặc dù cluster không còn khả năng quản lý và phân phối task, các container trên các node còn lại vẫn tiếp tục hoạt động.

4.3 Những mặt hạn chế của Docker Swarm

4.3.1 Hạn chế về mất mát dịch vụ

Vấn đề: Việc khắc phục sự cố mất kiểm soát cluster khi các node manager không hoạt động là rất quan trọng. Trong khi các node manager bị sập, các task không được phân chia cho các node còn hoạt động dẫn đến khả năng sẵn sàng không còn cao, thậm chí ảnh hưởng đến các dịch vụ cần kết nối đến các task này.

Giải pháp tạm thời: Thiết lập các dịch vụ quan trọng hoạt động trên các node worker thay vì thiết lập trên các node manager để hạn chế mất mát dịch vụ, đồng thời giảm khả năng chết của các dịch vụ liên quan cần kết nối đến các dịch vụ này.

4.3.2 Hạn chế khi khởi động lại Node hoặc thêm Node mới

Vấn đề: Khi một node bị sập và sau đó được khởi động lại, hoặc khi thêm một node mới vào cluster, Docker Swarm không tự động phân chia lại các task cho các node mới. Điều này có thể dẫn đến tình trạng một số node phải xử lý quá tải trong khi các node khác không có task nào.

Giải pháp tạm thời: Người quản trị hệ thống cần can thiệp thủ công để phân chia lại các task.

CHƯƠNG 5. TỔNG KẾT

5.1 Kết quả đạt được

Là sinh viên chuyên ngành Kỹ thuật phần mềm, việc chọn đề tài "*Triển khai dịch vụ web có khả năng mở rộng sử dụng Docker Swarm*" mang lại nhiều lợi ích thiết thực trong việc phát triển kiến thức và kỹ năng. Đầu tiên, đề tài này giúp chúng tôi tiếp cận và nắm vững các khái niệm về containerization, một công nghệ đang ngày càng phổ biến và quan trọng trong ngành công nghiệp phần mềm. Việc hiểu và sử dụng Docker Swarm sẽ cung cấp cho chúng tôi kiến thức sâu sắc hơn về cách quản lý và điều phối các container, một kỹ năng cần thiết cho sự nghiệp tương lai.

Hơn nữa, triển khai dịch vụ web theo mô hình microservice sử dụng Docker Swarm giúp chúng tôi hiểu rõ hơn về cách xây dựng và quản lý các hệ thống phân tán. Đây là một lĩnh vực quan trọng trong phát triển phần mềm hiện đại, đặc biệt là khi các ứng dụng ngày càng yêu cầu khả năng mở rộng cao. Thông qua việc thực hiện đề tài này, chúng tôi có cơ hội học cách thiết lập và quản lý các cluster Docker Swarm, cũng như cách tối ưu hóa hiệu suất và bảo đảm tính sẵn sàng của hệ thống.

Trong quá trình thực hiện dự án, chúng tôi đã nghiên cứu và hiểu rõ cách thức Docker hoạt động, từ việc container hóa ứng dụng đến việc triển khai các dịch vụ trên Docker Swarm. Bên cạnh đó, chúng tôi cũng đã thực nghiệm thành công trong việc tạo lập, quản lý các dịch vụ containerized, và triển khai hệ thống phân tán trên Docker Swarm, đảm bảo tính linh hoạt và khả năng mở rộng của hệ thống.

Ngoài ra, đề tài giúp chúng tôi nâng cao kỹ năng lập trình và tìm hiểu sâu về kiến trúc Microservice nhiều hơn, làm việc với nhiều công cụ và công nghệ khác nhau, cải thiện khả năng giải quyết vấn đề. Cuối cùng, kinh nghiệm thực tiễn từ đề tài này sẽ là điểm cộng lớn trong hồ sơ, giúp chúng tôi chuẩn bị tốt hơn cho công việc sau khi tốt nghiệp.

5.2 Ưu điểm và Nhược điểm của dự án

5.2.1 Ưu điểm

Sử dụng cấu trúc Microservice: Cấu trúc Microservice trong dự án mang lại nhiều lợi ích nổi bật so với kiến trúc Monolithic. Microservice cho phép mở rộng linh hoạt các dịch vụ độc lập, triển khai và cập nhật dễ dàng, và bảo trì hệ thống hiệu quả hơn. Mỗi dịch vụ có thể được phát triển và quản lý riêng biệt, sử dụng công nghệ và ngôn ngữ lập trình khác nhau, đồng thời tăng cường khả năng xử lý sự cố và quản lý tài nguyên hiệu quả. Những ưu điểm này giúp cải thiện khả năng mở rộng, hiệu suất, và độ tin cậy của hệ thống, mang lại sự linh hoạt và tối ưu hóa cho dự án.

Ứng dụng nhiều công nghệ tiên tiến: Dự án áp dụng các công nghệ tiên tiến như API Gateway, JWT, Redis, và RabbitMQ, giúp quản lý lưu lượng truy cập, xác thực người dùng, cache dữ liệu, và xử lý tác vụ không đồng bộ. Những công nghệ này góp phần nâng cao hiệu suất và bảo mật hệ thống, đảm bảo hoạt động trơn tru và hiệu quả.

Ứng dụng Docker và Docker Swarm: Dự án sử dụng Docker và Docker Swarm để container hóa các dịch vụ và triển khai hệ thống phân tán một cách linh hoạt. Docker Swarm đảm bảo tính mở rộng và hiệu suất tối ưu của hệ thống, giúp quản lý và vận hành các dịch vụ dễ dàng hơn trong môi trường phân tán.

5.2.2 Nhược điểm

Thiếu cấu hình master-slave: Việc thiếu cấu hình master-slave trong dự án có thể giảm hiệu suất hệ thống do tất cả các yêu cầu phải xử lý bởi cùng một cơ sở dữ liệu, hạn chế khả năng mở rộng và làm tăng rủi ro về độ tin cậy. Điều này cũng gây khó khăn trong việc quản lý dữ liệu và áp dụng các chiến lược sao lưu, phục hồi, dẫn đến khả năng khôi phục dữ liệu kém hơn khi gặp sự cố.

Sử dụng load-balancing có sẵn trong Docker Swarm: chỉ hỗ trợ HTTP(S)/TCP, mở rộng hạn chế, tính năng cơ bản, cấu hình ít tùy chỉnh, không hỗ trợ TLS termination, hiệu suất có thể kém hơn công cụ chuyên dụng.

5.3 Ưu điểm và Nhược điểm của Docker Swarm

5.3.1 Ưu điểm

Dễ triển khai và quản lý: Docker Swarm dễ dàng cài đặt và cấu hình. Bạn có thể nhanh chóng chuyển đổi một Docker Engine đơn lẻ thành một cụm Swarm với các lệnh đơn giản, giúp tiết kiệm thời gian và công sức.

Tích hợp tốt với Docker: Docker Swarm tích hợp chặt chẽ với Docker, sử dụng các cú pháp và công cụ quen thuộc của Docker. Điều này làm cho việc chuyển từ môi trường phát triển sang môi trường sản xuất trở nên dễ dàng và không đòi hỏi phải học thêm nhiều công cụ mới.

Tự động cân bằng tải: Docker Swarm tự động phân phối các container trên các nút trong cụm, giúp đảm bảo tải được cân bằng đều và tài nguyên được sử dụng hiệu quả. Cơ chế này cũng giúp giảm thiểu tình trạng quá tải ở một số nút nhất định.

Khả năng mở rộng: Docker Swarm cho phép bạn dễ dàng mở rộng hoặc thu hẹp quy mô cụm của mình bằng cách thêm hoặc loại bỏ các nút. Điều này giúp hệ thống linh hoạt trong việc đáp ứng các nhu cầu tải tăng giảm khác nhau.

Bảo mật: Docker Swarm cung cấp các tính năng bảo mật như TLS để mã hóa truyền thông giữa các nút, và hỗ trợ các chính sách bảo mật có thể định cấu hình để kiểm soát quyền truy cập. Tính năng mã hóa này giúp bảo vệ dữ liệu khỏi các mối đe dọa tiềm ẩn trong quá trình truyền tải.

Khả năng chịu lỗi: Docker Swarm có khả năng tự phục hồi và tái khởi động các container nếu một nút bị hỏng, giúp duy trì tính sẵn sàng của ứng dụng. Việc này giúp hệ thống duy trì hoạt động ngay cả khi một phần của cụm gặp sự cố.

Hỗ trợ đa nền tảng: Docker Swarm hỗ trợ triển khai trên nhiều nền tảng khác nhau như Linux, Windows, và MacOS, giúp bạn có thể triển khai trên các hệ thống đa dạng.

5.3.2 Nhược điểm

Hạn chế tính năng so với Kubernetes: So với Kubernetes, Docker Swarm thiếu một số tính năng cao cấp như quản lý trạng thái phức tạp, hỗ trợ các plugin, và khả

năng tùy chỉnh mạnh mẽ. Điều này có thể hạn chế khả năng của Docker Swarm trong các ứng dụng phức tạp.

Quản lý mạng phức tạp hơn: Mặc dù Docker Swarm hỗ trợ mạng overlay, nhưng việc cấu hình và quản lý mạng có thể trở nên phức tạp khi số lượng nút tăng lên. Điều này có thể gây khó khăn trong việc duy trì sự nhất quán và hiệu suất của mạng.

Hỗ trợ công đồng và tài liệu: Docker Swarm có ít tài liệu và hỗ trợ từ cộng đồng hơn so với Kubernetes. Điều này có thể gây khó khăn khi bạn gặp vấn đề hoặc cần tìm hiểu sâu hơn, đặc biệt là khi triển khai các giải pháp phức tạp.

Khả năng mở rộng giới hạn: Docker Swarm hoạt động tốt với các cụm nhỏ đến trung bình, nhưng có thể gặp khó khăn khi mở rộng quy mô lên hàng nghìn nút, điều mà Kubernetes xử lý tốt hơn. Điều này làm cho Docker Swarm không phải là lựa chọn lý tưởng cho các ứng dụng lớn yêu cầu khả năng mở rộng cao.

Khả năng tích hợp và mở rộng hạn chế: So với Kubernetes, Docker Swarm ít có khả năng tích hợp với các công cụ và dịch vụ bên ngoài, cũng như hạn chế về khả năng mở rộng với các plugin và các công cụ của bên thứ ba.

5.4 So sánh giữa Docker Swarm và Kubernetes

Docker Swarm và Kubernetes là hai nền tảng phổ biến nhất hiện nay để quản lý container trong môi trường sản xuất. Mỗi nền tảng có những ưu điểm và nhược điểm riêng, phù hợp với các nhu cầu khác nhau. Dưới đây là sự so sánh chi tiết giữa Docker Swarm và Kubernetes:

Bảng 5.1 So sánh giữa Docker Swarm và Kubernetes

Tiêu chí	Docker Swarm	Kubernetes
Triển khai và quản lý	Dễ dàng triển khai và quản lý với các lệnh đơn giản.	Phức tạp hơn trong triển khai và quản lý, yêu cầu nhiều thời gian và kỹ năng.
Cân bằng tải	Tự động cân bằng tải các container trên các nút.	Cung cấp các giải pháp cân bằng tải mạnh mẽ hơn với nhiều tùy chọn cấu hình.

Mở rộng quy mô	Dễ dàng mở rộng hoặc thu hẹp quy mô, nhưng giới hạn ở quy mô nhỏ đến trung bình.	Khả năng mở rộng mạnh mẽ, hỗ trợ các cụm lớn với hàng nghìn nút.
Khả năng chịu lỗi	Có khả năng tự phục hồi và tái khởi động container khi nút bị hỏng.	Khả năng chịu lỗi cao với các cơ chế tự động phát hiện và khắc phục sự cố.
Tính năng phong phú	Hạn chế hơn về tính năng so với Kubernetes, thiếu các tính năng cao cấp.	Cung cấp nhiều tính năng phong phú và linh hoạt, hỗ trợ nhiều loại workload khác nhau.
Yêu cầu tài nguyên	Tiêu tốn ít tài nguyên hơn, phù hợp với hệ thống nhỏ và vừa.	Yêu cầu tài nguyên cao hơn, phù hợp với các hệ thống lớn.

5.5 Dùng Docker Swarm trong trường hợp nào?

Docker Swarm là một công cụ quản lý container mạnh mẽ, nhưng không phải lúc nào cũng là lựa chọn phù hợp cho mọi tình huống.

5.5.1 Khi nào nên dùng Docker Swarm?

Triển khai ứng dụng quy mô vừa và nhỏ: Docker Swarm rất phù hợp cho các ứng dụng có quy mô vừa và nhỏ, nơi yêu cầu quản lý cụm đơn giản và dễ triển khai. Với giao diện thân thiện và các lệnh tương tự như Docker, Docker Swarm cho phép triển khai nhanh chóng mà không cần cấu hình phức tạp.

Môi trường phát triển và thử nghiệm: Do tính dễ sử dụng và khả năng tích hợp chặt chẽ với Docker, Docker Swarm là một lựa chọn tốt cho các môi trường phát triển và thử nghiệm. Nó cho phép các nhà phát triển dễ dàng tạo lập và kiểm tra các cụm container, đồng thời giúp đảm bảo rằng môi trường phát triển tương đồng với môi trường sản xuất.

Cụm nhỏ với số lượng nút hạn chế: Docker Swarm hoạt động hiệu quả với các cụm nhỏ, có số lượng nút hạn chế. Trong các tình huống này, Docker Swarm cung cấp đầy đủ các tính năng cần thiết mà không yêu cầu các cấu hình phức tạp và không cần đến các tính năng mở rộng cao cấp.

Cần một giải pháp quản lý container đơn giản và nhanh chóng: Nếu doanh nghiệp của bạn cần một giải pháp quản lý container đơn giản, có thể triển khai nhanh

và dễ dàng bảo trì, Docker Swarm là một lựa chọn lý tưởng. Nó giúp bạn dễ dàng kiểm soát các dịch vụ, quản lý tài nguyên và đảm bảo hiệu suất ổn định.

Khả năng chịu lỗi và tự phục hồi ở mức cơ bản: Với cơ chế tự phục hồi và khả năng chịu lỗi, Docker Swarm có thể đảm bảo rằng các ứng dụng của bạn sẽ duy trì hoạt động ngay cả khi có sự cố xảy ra ở một hoặc vài nút trong cụm. Điều này đặc biệt hữu ích trong các ứng dụng không yêu cầu độ sẵn sàng cao nhưng vẫn cần một mức độ bảo mật và tin cậy nhất định.

Yêu cầu tích hợp nhanh chóng vào các hệ thống hiện có sử dụng Docker: Nếu bạn đã sử dụng Docker trong hệ thống hiện tại của mình, Docker Swarm sẽ là một lựa chọn tự nhiên để mở rộng sang quản lý cụm container. Việc chuyển đổi từ Docker sang Docker Swarm diễn ra liền mạch và không yêu cầu học thêm các công cụ hoặc quy trình mới.

Ngân sách và tài nguyên giới hạn: Khi các tổ chức có hạn chế về ngân sách hoặc không cần các tính năng phức tạp của Kubernetes, Docker Swarm cung cấp một giải pháp quản lý container có chi phí thấp và ít phức tạp hơn, giúp tiết kiệm thời gian và nguồn lực.

5.5.2 Khi nào không nên dùng Docker Swarm?

Yêu cầu mở rộng quy mô lớn: Nếu bạn cần quản lý các cụm container lớn, với hàng nghìn nút và yêu cầu tính năng phức tạp, Kubernetes có thể là lựa chọn tốt hơn. Kubernetes cung cấp các công cụ và tính năng mạnh mẽ hơn để quản lý và mở rộng các cụm quy mô lớn.

Cần tích hợp với các công cụ DevOps phức tạp: Nếu bạn đang sử dụng một hệ sinh thái DevOps phức tạp, hoặc cần tích hợp với các công cụ CI/CD, giám sát, và quản lý logging chuyên sâu, Kubernetes thường cung cấp nhiều tùy chọn và tính năng linh hoạt hơn so với Docker Swarm.

Yêu cầu tính năng bảo mật và kiểm soát chi tiết: Khi cần kiểm soát bảo mật chi tiết và cần quản lý các chính sách bảo mật phức tạp, Kubernetes với hệ thống RBAC (Role-Based Access Control) và nhiều plugin bảo mật sẽ là sự lựa chọn tốt hơn.

5.6 Thuận lợi và Khó khăn

5.6.1 Thuận lợi

Kiến thức nền tảng vững chắc: Chúng tôi được trang bị kiến thức vững vàng về lập trình và công nghệ thông qua quá trình học tập và thực hành, giúp dễ dàng tiếp cận và áp dụng các công nghệ mới như Docker, Docker Swarm, Microservices, Redis, RabbitMQ,...

Sự hỗ trợ từ tài liệu và cộng đồng: Các công nghệ và công cụ sử dụng trong dự án đều có tài liệu phong phú và cộng đồng người dùng rộng lớn. Điều này giúp chúng tôi giải quyết nhanh chóng các vấn đề phát sinh trong quá trình triển khai và phát triển hệ thống.

5.6.2 Khó khăn

Khả năng kết hợp nhiều công nghệ: Dự án yêu cầu tích hợp nhiều công nghệ khác nhau như Docker, RabbitMQ, Redis, và API Gateway. Việc kết hợp chúng đòi hỏi chúng tôi phải có sự hiểu biết sâu rộng và khả năng xử lý các vấn đề phát sinh khi các công nghệ này tương tác với nhau.

Quản lý các microservices: Việc triển khai và quản lý một hệ thống phân tán với nhiều microservices đòi hỏi sự cẩn trọng trong việc điều phối, theo dõi, và khắc phục sự cố. Đôi khi trong quá trình triển khai hoặc kiểm thử dễ gây ra rắc rối nếu không được thực hiện cẩn thận.

Thời gian thực hiện dự án: Dự án yêu cầu một lượng thời gian đáng kể để nghiên cứu, thử nghiệm và triển khai. Việc quản lý thời gian hiệu quả là một thách thức lớn, đặc biệt khi phải cân bằng giữa các hoạt động cá nhân và nhiệm vụ trong dự án để đảm bảo tiến độ.

5.7 Hướng phát triển trong tương lai

5.7.1 Cấu hình cơ chế Master-Slave

Cấu hình master-slave là một phương pháp phổ biến trong quản lý các dịch vụ stateful, đặc biệt là cơ sở dữ liệu, nhằm đảm bảo tính sẵn sàng và độ tin cậy cao. Mô

hình này phân chia nhiệm vụ giữa các node, trong đó một node master xử lý các yêu cầu ghi, còn các node slave xử lý các yêu cầu đọc. Việc sao chép dữ liệu từ master đến slave đảm bảo tính nhất quán, và khi master gặp sự cố, một node slave có thể tự động chuyển thành master để duy trì hoạt động liên tục.

5.7.2 Cấu hình Cơ chế Auto Scale

Auto scale (tự động mở rộng) là một cơ chế quan trọng giúp hệ thống tự động điều chỉnh số lượng instances của các dịch vụ dựa trên nhu cầu thực tế. Khi tải trọng tăng, hệ thống sẽ tự động mở rộng để duy trì hiệu suất; ngược lại, khi tải giảm, hệ thống sẽ thu nhỏ lại để tiết kiệm tài nguyên.

5.7.3 Bổ sung công cụ giám sát

Các công cụ như Prometheus và Grafana sẽ được triển khai để theo dõi hiệu suất hệ thống theo thời gian thực. Việc cấu hình chi tiết cho các công cụ này sẽ giúp theo dõi và ghi lại các chỉ số quan trọng của cả dịch vụ stateful và stateless, đảm bảo tính minh bạch và khả năng khắc phục sự cố nhanh chóng.

Việc ghi log và phân tích log sẽ được quản lý bởi ELK Stack (Elasticsearch, Logstash, Kibana). Điều này cho phép lưu trữ và truy vấn log một cách hiệu quả, đồng thời cung cấp giao diện trực quan để phân tích dữ liệu log, giúp nhanh chóng xác định nguyên nhân của các vấn đề phát sinh.

5.7.4 Tối ưu hóa hiệu suất hệ thống

Hệ thống sẽ tiếp tục được cải tiến thông qua việc nghiên cứu và áp dụng các kỹ thuật tối ưu hóa mã nguồn, điều chỉnh cấu hình dịch vụ, và sử dụng các công cụ giám sát để phát hiện và xử lý các điểm nghẽn.

5.7.5 Mở rộng quy mô ứng dụng

Để đáp ứng nhu cầu ngày càng tăng, các công nghệ và phương pháp mới như Kubernetes hoặc các dịch vụ quản lý container khác sẽ được nghiên cứu và triển khai. Việc này giúp mở rộng quy mô ứng dụng một cách linh hoạt và đảm bảo tính khả dụng cao.

Quy trình triển khai liên tục (CI/CD) sẽ được cải tiến nhằm đảm bảo các bản cập nhật được đưa vào hệ thống một cách nhanh chóng và an toàn, đồng thời giảm thiểu thời gian gián đoạn dịch vụ khi mở rộng quy mô.

5.7.6 Cải thiện giao diện người dùng và trải nghiệm

Giao diện người dùng sẽ được đầu tư nâng cấp để cải thiện trải nghiệm người dùng (UX). Các tính năng mới sẽ được bổ sung dựa trên phản hồi của người dùng, giúp hệ thống thân thiện và dễ sử dụng hơn.

5.7.7 Tích hợp công nghệ mới

Hệ thống sẽ tiếp tục được cải tiến bằng cách theo dõi và tích hợp các công nghệ mới như trí tuệ nhân tạo (AI) hoặc học máy (machine learning). Các công nghệ này sẽ được sử dụng để cải thiện khả năng dự đoán, phân tích dữ liệu, nhằm nâng cao trải nghiệm thực khách.

TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] TopDev, “Docker là gì? Tìm hiểu về Docker,” [Trực tuyến]. Available:

[https://topdev.vn/blog/docker-la-gi/.](https://topdev.vn/blog/docker-la-gi/)

[2] TopDev, “Redis là gì? Ưu điểm của nó và ứng dụng,” [Trực tuyến]. Available:

[https://topdev.vn/blog/redis-la-gi/.](https://topdev.vn/blog/redis-la-gi/)

[3] Itnavi, “Microservices là gì?,” [Trực tuyến]. Available:

<https://itnavi.com.vn/blog/microservices-la-gi>.

[4] 200lab, “RabbitMQ là gì? Cách hoạt động?,” [Trực tuyến]. Available:

[https://200lab.io/blog/rabbitmq-la-gi/.](https://200lab.io/blog/rabbitmq-la-gi/)

Tiếng Anh

[5] Sequelize, "Document Sequelize V6," [Online]. Available:

<https://sequelize.org/docs/v6/>.

[6] IBM, "Docker Swarm vs. Kubernetes: A Comparison," [Online]. Available:

<https://www.ibm.com/blog/docker-swarm-vs-kubernetes-a-comparison/>.

[7] Docker, "Document of Docker Swarm," [Online]. Available:

<https://docs.docker.com/engine/swarm/>.

[8] Docker, "Document of Docker," [Online]. Available:

<https://docs.docker.com/compose/compose-file/>.