

**ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**



Bùi Minh Quang

**XÂY DỰNG GIẢI PHÁP PAAS HỖ TRỢ TRIỂN KHAI
BACKEND TRÒ CHƠI DỰA TRÊN KIẾN TRÚC
SERVERLESS**

**ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY
Ngành: Mạng máy tính và truyền thông dữ liệu**

HÀ NỘI – 2025

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Bùi Minh Quang

XÂY DỰNG GIẢI PHÁP PAAS HỖ TRỢ TRIỂN KHAI
BACKEND TRÒ CHƠI DỰA TRÊN KIẾN TRÚC
SERVERLESS

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY
Ngành: Mạng máy tính và truyền thông dữ liệu

Cán bộ hướng dẫn: TS. Phạm Mạnh Linh

HÀ NỘI – 2025

LỜI CAM ĐOAN

Em xin cam đoan rằng đồ án tốt nghiệp về đề tài "Xây dựng giải pháp PaaS hỗ trợ triển khai backend trò chơi dựa trên kiến trúc serverless" là sản phẩm của sự nỗ lực và công sức nghiên cứu cá nhân của em. Tất cả các ý kiến, kết quả, và phân tích trong đồ án này được thực hiện bởi em một cách độc lập và trung thực, không bị ảnh hưởng bởi bất kỳ nguồn thông tin nào mà không được trích dẫn. Em cam kết tuân thủ nguyên tắc đạo đức và trách nhiệm trong nghiên cứu khoa học, không sao chép hoặc lạm dụng thông tin từ bất kỳ nguồn nào mà không được sự chấp thuận của tác giả hoặc không trích dẫn cụ thể.

Em hiểu rõ tầm quan trọng của tính minh bạch và trung thực trong việc nghiên cứu, và cam kết giữ vững nguyên tắc này trong toàn bộ quá trình làm đồ án tốt nghiệp. Em sẽ tuân thủ mọi quy định và hướng dẫn của Trường Đại học Công nghệ - ĐHQGHN về nghiên cứu và viết báo cáo khoa học, em sẵn lòng chịu trách nhiệm về bất kỳ hành vi vi phạm nào liên quan đến nội dung và phương pháp nghiên cứu trong đồ án của mình.

Hà Nội, Ngày Tháng Năm

Sinh viên

Bùi Minh Quang

LỜI CẢM ƠN

Em xin bày tỏ lòng biết ơn chân thành đến Khoa Công nghệ Thông tin – Trường Đại học Công nghệ, ĐHQGHN, nơi đã trang bị cho em nền tảng kiến thức vững chắc cùng một môi trường học tập và nghiên cứu lý tưởng trong suốt quá trình học tập tại trường.

Đặc biệt, em xin gửi lời cảm ơn sâu sắc đến thầy Phạm Mạnh Linh – người thầy đã tận tình hướng dẫn, đồng hành và hỗ trợ em trong suốt quá trình thực hiện đồ án. Sự tận tâm và nhiệt huyết của thầy là nguồn động lực to lớn giúp em vượt qua những khó khăn, thử thách và hoàn thành đồ án một cách tốt nhất.

Em cũng xin gửi lời cảm ơn đến các thầy cô trong khoa, cùng các anh chị và bạn bè đã luôn sẵn sàng chia sẻ kiến thức, kinh nghiệm quý báu và hỗ trợ em trong hành trình học tập, nghiên cứu cũng như phát triển bản thân.

Quãng thời gian học tập và thực hiện đồ án không chỉ là thử thách mà còn là cơ hội để em rèn luyện, tích lũy kinh nghiệm và trưởng thành hơn. Em xin chân thành cảm ơn tất cả mọi người đã đồng hành và ủng hộ em trong suốt chặng đường này. Chúc mọi người luôn mạnh khỏe, hạnh phúc và gặt hái nhiều thành công trong cuộc sống.

TÓM TẮT

Đồ án đề xuất và hiện thực hóa một giải pháp PaaS hỗ trợ triển khai backend cho trò chơi trực tuyến, được xây dựng trên kiến trúc serverless của AWS nhằm tối ưu khả năng mở rộng, linh hoạt và hiệu quả chi phí. Giải pháp hướng đến việc tiêu chuẩn hóa quy trình phát triển backend trò chơi thông qua việc cung cấp các dịch vụ nền tảng như ghép trận, xếp hạng người chơi, trò chuyện, quản lý bạn bè, theo dõi trận đấu, đồng thời đảm nhiệm vai trò quản lý vòng đời máy chủ trò chơi theo yêu cầu hoạt động thực tế.

Hệ thống được thiết kế theo hướng dịch vụ hóa, cho phép các nhà phát triển tích hợp backend dễ dàng thông qua giao diện API. Việc tận dụng toàn bộ hệ sinh thái serverless của AWS—bao gồm Lambda, API Gateway, DynamoDB, AppSync, Cognito, SQS, SNS và Fargate—giúp nền tảng đạt được hiệu suất cao, độ sẵn sàng tốt và khả năng vận hành linh hoạt mà không cần quản lý hạ tầng vật lý truyền thống.

Giải pháp được triển khai và kiểm chứng qua một nền tảng trò chơi mẫu (cờ vua), hỗ trợ đến 10.000 người chơi đồng thời. Các kịch bản kiểm thử hiệu suất được thực hiện bằng công cụ mô phỏng k6 cho thấy hệ thống duy trì độ trễ thấp, hoạt động ổn định và không xuất hiện nút thắt cổ chai đáng kể ngay cả trong điều kiện tải cao.

Kết quả đạt được cho thấy tính khả thi và hiệu quả thực tiễn của hướng tiếp cận serverless trong việc xây dựng backend trò chơi hiện đại. Giải pháp không chỉ rút ngắn thời gian phát triển, giảm chi phí vận hành mà còn mở ra tiềm năng mở rộng hỗ trợ cho nhiều thể loại trò chơi trong tương lai.

Từ khóa: backend trò chơi, serverless, AWS, PaaS, máy chủ trò chơi, matchmaking, xếp hạng, Fargate, auto-scaling.

BẢNG KÝ HIỆU, CHỮ VIẾT TẮT

Ký hiệu/Viết tắt	Diễn giải
API	Application Programming Interface – Giao diện lập trình ứng dụng
API Gateway	Dịch vụ quản lý và định tuyến API của AWS (REST/WebSocket)
AppSync	Dịch vụ GraphQL serverless của AWS, hỗ trợ realtime
AWS	Amazon Web Services – Nền tảng điện toán đám mây của Amazon
BaaS	Backend as a Service – Backend dưới dạng dịch vụ
CDN	Content Delivery Network – Mạng phân phối nội dung
CloudWatch	Dịch vụ giám sát logs, chỉ số và cảnh báo hệ thống của AWS
Cold start	Trạng thái khởi động ban đầu của Lambda hoặc Fargate (gây trễ khởi tạo)
Cognito	Dịch vụ xác thực và quản lý danh tính người dùng của AWS
DynamoDB	Cơ sở dữ liệu NoSQL serverless của AWS
ECS	Elastic Container Service – Dịch vụ chạy container của AWS
ECS Fargate	Cách chạy container không cần quản lý hạ tầng trong ECS
ECR	Elastic Container Registry – Kho lưu trữ Docker Image của AWS
EventBridge	Dịch vụ bus sự kiện serverless của AWS
FaaS	Function as a Service – Hàm dưới dạng dịch vụ

IaC	Infrastructure as Code – Hạ tầng dưới dạng mã
JWT	JSON Web Token – Chuỗi mã hóa xác thực người dùng
Lambda	Dịch vụ tính toán serverless của AWS chạy theo sự kiện
NIST	National Institute of Standards and Technology
Matchmaking	Ghép trận – cơ chế tìm đối thủ phù hợp cho người chơi
PaaS	Platform as a Service – Nền tảng dưới dạng dịch vụ
QoS	Quality of Service – Chất lượng dịch vụ
RD	Rating Deviation – Độ lệch xếp hạng (trong Glicko)
RPS	Requests Per Second – Số lượng yêu cầu mỗi giây
S3	Simple Storage Service – Dịch vụ lưu trữ đối tượng của AWS
SAM	Serverless Application Model – Mô hình ứng dụng serverless của AWS
SDK	Software Development Kit – Bộ công cụ phát triển phần mềm
Serverless	Kiến trúc không máy chủ – tính toán không cần quản lý hạ tầng
Spectating	Chế độ theo dõi trận đấu (không tham gia trực tiếp)
SQS	Simple Queue Service – Dịch vụ hàng đợi tin nhắn serverless của AWS
TTL	Time To Live – Thời gian tồn tại của bản ghi dữ liệu
VPC	Virtual Private Cloud – Mạng đám mây riêng ảo trong AWS

MỤC LỤC

Danh sách hình ảnh.....	1
Danh sách bảng biểu.....	3
Chương 1: Tổng quan về đề tài	4
1.1 Đặt vấn đề	4
1.2 Lý do chọn đề tài	5
1.2.1 Nhu cầu thực tiễn từ thị trường game.....	5
1.2.2 Ưu điểm vượt trội của kiến trúc serverless	5
1.2.3 Xu thế phát triển tất yếu	6
1.3 Đối tượng nghiên cứu.....	6
1.4 Phạm vi nghiên cứu	7
1.5 Phương pháp nghiên cứu	8
1.6 Mục tiêu nghiên cứu	9
Chương 2: Cơ sở lý thuyết.....	10
2.1 Nền tảng dưới dạng dịch vụ	10
2.1.1 Giới thiệu về điện toán đám mây	10
2.1.2 Khái niệm và đặc trưng của PaaS	11
2.1.3 Các thành phần của mô hình PaaS	11
2.1.4 Lợi ích và hạn chế của mô hình PaaS.....	12
2.1.5 Các nhà cung cấp dịch vụ PaaS phổ biến.....	12
2.1.6 PaaS trong triển khai backend trò chơi.....	12
2.1.7 Kết luận	13
2.2 Kiến trúc điện toán phi máy chủ.....	13
2.2.1 Khái niệm và đặc trưng	13
2.2.2 Một số giải pháp Serverless cốt lõi của AWS	14
2.2.3 So sánh các mô hình triển khai backend trò chơi	15
2.2.4 Kết luận	17
2.3 Cơ sở hạ tầng dưới dạng mã	17

2.3.1 Khái niệm	17
2.3.1 AWS SAM.....	18
2.3.2 Kết luận	18
2.4 Hệ thống xếp hạng người chơi	19
2.4.1 Giới thiệu	19
2.4.2 Một số hệ thống xếp hạng phổ biến	20
2.4.3 Kết luận	23
Chương 3: Thiết kế backend trò chơi dựa trên kiến trúc serverless	24
3.1 Thiết kế hệ thống	24
3.1.1 Kiến trúc tổng quát	24
3.1.2 Kiến trúc Serverless.....	26
3.1.3 Cơ sở dữ liệu.....	28
3.1.4 Bảo mật hệ thống.....	35
3.2 Máy chủ trò chơi.....	36
3.2.1 Phát triển máy chủ	36
3.2.2 Mô hình triển khai	37
3.2.3 Chiến lược mở rộng tự động	38
3.3 Các hệ thống con trong backend trò chơi.....	41
Chương 4: Giải pháp hỗ trợ triển khai backend trò chơi.....	47
4.1 Mô hình giải pháp.....	47
4.1.1 Tổng quan mô hình.....	47
4.1.2 Mục tiêu của mô hình	47
4.1.3 Kiến trúc tổng thể	48
4.1.4 Luồng hoạt động của hệ thống	50
4.2 Bộ công cụ triển khai backend trò chơi.....	51
4.2.1 API tích hợp các dịch vụ backend	51
4.2.2 SDK hỗ trợ triển khai máy chủ trò chơi	53
4.3 Quy trình triển khai mẫu.....	55

Chương 5: Triển khai và đánh giá kết quả	61
5.1 Mục tiêu đánh giá	61
5.2 Môi trường thử nghiệm	61
5.2.1 Triển khai nền tảng chơi cờ vua trực tuyến.....	62
5.2.2 Kiểm thử bằng công cụ mô phỏng	66
5.2.3 Ước tính chi phí bằng AWS Pricing Calculator	76
5.3 Kết quả thử nghiệm	80
5.4 Đánh giá.....	83
5.5 So sánh với các giải pháp hiện có.....	84
Chương 6: Kết luận, tổng kết	87
6.1 Kết luận.....	87
6.2 Hướng nghiên cứu trong tương lai	88
Phụ lục	90
Tài liệu tham khảo	102

Danh sách hình ảnh

Hình 2.1 Mô hình dịch vụ điện toán đám mây	10
Hình 3.1 Kiến trúc tổng quát của hệ thống.....	24
Hình 3.2 Kiến trúc Serverless của hệ thống	27
Hình 3.3 Bảng Connections.....	29
Hình 3.4 Bảng UserRatings	29
Hình 3.5 Bảng UserProfiles.....	30
Hình 3.6 Bảng MatchmakingTickets.....	30
Hình 3.7 Bảng ActiveMatches	31
Hình 3.8 Bảng UserMatches.....	32
Hình 3.9 Bảng MatchStates	32
Hình 3.10 Bảng MatchRecords	33
Hình 3.11 Bảng MatchResults.....	33
Hình 3.12 Bảng Messages	34
Hình 3.13 Bảng SpectatorConversations.....	34
Hình 3.14 Bảng Friendships	35
Hình 3.15 Bảng FriendRequests	35
Hình 3.16 Mô hình triển khai ECS Fargate	37
Hình 3.17 Chiến lược auto-scaling dựa trên phân phối trận đấu.....	40
Hình 3.18 Luồng hoạt động của hệ thống xếp hạng.....	42
Hình 3.19 Luồng hoạt động của hệ thống ghép trận	43
Hình 3.20 Luồng hoạt động của hệ thống theo dõi trận đấu	44
Hình 3.21 Luồng hoạt động của hệ thống bạn bè.....	45
Hình 3.22 Luồng hoạt động của hệ thống trò chuyện	46
Hình 4.1 Kiến trúc giải pháp PaaS	48
Hình 4.2 Luồng hoạt động của hệ thống PaaS	50
Hình 4.3 Định nghĩa lớp xử lý thông qua giao diện ServerHandler.....	54
Hình 4.4 Khởi tạo máy chủ trò chơi thông qua SDK	54
Hình 4.5 Build và lưu trữ Container Image.....	55
Hình 4.6 Triển khai backend trò chơi.....	56

Hình 4.7 Các template định nghĩa tài nguyên	57
Hình 4.8 Trạng thái triển khai của backend	58
Hình 4.9 Danh sách API Endpoints cụ thể cho từng backend.....	59
Hình 4.10 Giao diện giám sát tài nguyên và trạng thái hoạt động của backend	59
Hình 4.11 Giao diện cập nhật backend.....	60
Hình 5.1 Cài đặt logic xử lý khi trận đấu được tạo	62
Hình 5.2 Lưu trữ container image trên Amazon ECR.....	62
Hình 5.3 Triển khai backend trò chơi cờ vua	63
Hình 5.4 Các màn hình của ứng dụng chơi cờ vua	65
Hình 5.5 Giao diện công cụ k6	66
Hình 5.6 Kịch bản kiểm thử 100 người dùng	67
Hình 5.7 Kiểm thử tải với 100 người dùng	68
Hình 5.8 Kịch bản kiểm thử 1000 người dùng.....	68
Hình 5.9 Kiểm thử tải với 1000 người dùng	69
Hình 5.10 Kịch bản kiểm thử 10000 người dùng.....	69
Hình 5.11 Kiểm thử tải với 10000 người dùng	70
Hình 5.12 Kiểm thử tải máy chủ trò chơi với 1000 người chơi đồng thời.....	71
Hình 5.13 Độ trễ đồng bộ trạng thái của người chơi.....	72
Hình 5.14 Mức sử dụng CPU	72
Hình 5.15 Mức sử dụng bộ nhớ	73
Hình 5.16 Kịch bản kiểm thử khả năng mở rộng của máy chủ trò chơi	73
Hình 5.17 Kết quả gửi yêu cầu ghép trận	74
Hình 5.18 Kết quả ghép trận thành công	75
Hình A.1 Tập tin template.yaml (1)	90
Hình A.2 Tập tin template.yaml (2)	91
Hình A.3 Tập tin template.yaml (3)	92
Hình B.1 Tập tin customization.yaml (1)	93
Hình B.2 Tập tin customization.yaml (2)	94
Hình C.1 Kịch bản 100 người dùng API đồng thời.....	95
Hình C.2 Kịch bản 1000 người dùng API đồng thời.....	96

Hình C.3 Kịch bản 10000 người dùng API đồng thời.....	97
Hình C.4 Kịch bản kiểm thử hiệu suất máy chủ trò chơi (1)	98
Hình C.5 Kịch bản kiểm thử hiệu suất máy chủ trò chơi (2)	99
Hình C.6 Kịch bản kiểm thử khả năng mở rộng của máy chủ trò chơi (1)	100
Hình C.7 Kịch bản kiểm thử khả năng mở rộng của máy chủ trò chơi (2)	101

Danh sách bảng biếu

Bảng 2.1 So sánh các mô hình kiến trúc backend trò chơi.....	16
Bảng 2.2 So sánh các hệ thống xếp hạng	22
Bảng 3.1 Tham số trong chiến lược mở rộng tự động.....	39
Bảng 4.1 Các dịch vụ trong hệ thống backend trò chơi	51
Bảng 4.2 API Endpoint cho backend trò chơi	51
Bảng 4.3 Tích hợp giữa SDK và các dịch vụ AWS	55
Bảng 5.1 Chi phí HTTP API	76
Bảng 5.2 Chi phí Websocket API.....	76
Bảng 5.3 Chi phí AWS Lambda	77
Bảng 5.4 Chi phí AWS Cognito	77
Bảng 5.5 Chi phí Amazon DynamoDB	77
Bảng 5.6 Chi phí Amazon S3	78
Bảng 5.7 Chi phí AWS AppSync Real-Time	78
Bảng 5.8 Chi phí AWS SNS.....	79
Bảng 5.9 Chi phí AWS Fargate	79
Bảng 5.10 So sánh thời gian triển khai backend trò chơi	80
Bảng 5.11 Hiệu suất API	81
Bảng 5.12 Chi phí vận hành	82
Bảng 5.13 So sánh giải pháp đê tài với các nền tảng triển khai khác	84

Chương 1: Tổng quan về đề tài

1.1 Đặt vấn đề

Ngành công nghiệp trò chơi đã chứng kiến sự phát triển vượt bậc trong thập kỷ qua, không chỉ về quy mô thị trường mà còn về công nghệ nền tảng. Theo báo cáo của Newzoo (2023), tổng doanh thu toàn cầu từ các trò chơi đạt 187,7 tỷ USD, trong đó trò chơi di động chiếm 50%, PC chiếm 22%, và console chiếm 28%. Sự bùng nổ của các nền tảng phân phối kỹ thuật số như Steam, Epic Games Store, Google Play và App Store đã giúp các nhà phát triển game tiếp cận hàng trăm triệu người dùng dễ dàng hơn. Tuy nhiên, đi kèm với cơ hội này là những thách thức lớn trong việc xây dựng và vận hành hệ thống backend ổn định, linh hoạt và có khả năng mở rộng để đáp ứng số lượng người chơi biến động theo thời gian.

Một trong những vấn đề cốt lõi mà các nhà phát triển trò chơi phải đối mặt là chi phí và độ phức tạp trong việc triển khai, mở rộng và duy trì backend cũng như máy chủ trò chơi. Theo khảo sát của Game Developers Conference (GDC 2023), khoảng 40% các studio trò chơi nhỏ và độc lập gặp khó khăn trong việc duy trì hạ tầng hệ thống do chi phí cao và rào cản kỹ thuật. Các kiến trúc truyền thống như monolithic hoặc microservices tuy có thể đáp ứng một phần nhu cầu nhưng vẫn gặp nhiều giới hạn khi quy mô mở rộng.

- Mô hình monolithic: Đơn giản khi bắt đầu nhưng khó mở rộng, cập nhật gây gián đoạn dịch vụ.
- Mô hình microservices: Dù linh hoạt hơn nhưng yêu cầu chi phí vận hành cao và kỹ năng chuyên sâu để quản lý hạ tầng phức tạp.

Trong bối cảnh đó, kiến trúc serverless nổi lên như một giải pháp tiềm năng, cho phép loại bỏ phần lớn công việc vận hành hạ tầng nhờ tính mở rộng tự động và mô hình tính phí theo mức sử dụng. Các dịch vụ như AWS Lambda, API Gateway, DynamoDB, ECS Fargate, SNS/SQS và AppSync cung cấp nền tảng linh hoạt để triển khai cả backend lẫn máy chủ trò chơi mà không cần thiết lập cụm máy chủ truyền thống.

Với mục tiêu tận dụng các đặc điểm nổi bật của serverless, đề tài hướng đến việc xây dựng một giải pháp PaaS hỗ trợ triển khai backend cho trò chơi điện tử trực tuyến. Giải pháp này cung cấp đầy đủ các dịch vụ backend thiết yếu (ghép trận, xếp hạng, người chơi, trò chuyện...) đồng thời hỗ trợ triển khai, giám sát và tự động mở rộng máy chủ trò chơi. Qua đó, nhà phát triển có thể nhanh chóng đưa trò chơi vào vận hành mà không

cần xây dựng hạ tầng từ đầu, tập trung tối đa vào phát triển nội dung và trải nghiệm người chơi.

1.2 Lý do chọn đề tài

Ý tưởng về đề tài "Xây dựng giải pháp PaaS hỗ trợ triển khai backend trò chơi dựa trên kiến trúc serverless" xuất phát từ những nhu cầu cấp thiết và xu hướng phát triển mạnh mẽ trong ngành công nghiệp trò chơi hiện đại. Cụ thể:

1.2.1 Nhu cầu thực tiễn từ thị trường game

Ngành công nghiệp trò chơi toàn cầu đang trải qua giai đoạn tăng trưởng nhanh chóng, đặc biệt ở phân khúc game di động và thể thao điện tử (eSports). Theo thống kê từ App Annie, số lượng trò chơi di động có hơn 1 triệu người dùng hàng tháng đã tăng 35% trong giai đoạn 2020–2023, đặt ra yêu cầu rất cao về khả năng mở rộng và vận hành linh hoạt của backend.

Đối với các studio nhỏ và độc lập, việc xây dựng toàn bộ hệ thống backend và máy chủ trò chơi từ đầu thường vượt quá nguồn lực hiện có. Một khảo sát của Unity cho thấy 65% trò chơi indie lựa chọn sử dụng backend từ bên thứ ba để tiết kiệm thời gian và chi phí. Tuy nhiên, các giải pháp sẵn có thường thiếu khả năng tùy chỉnh hoặc không đáp ứng đủ linh hoạt về mặt triển khai server. Điều này tạo ra nhu cầu cấp thiết về một giải pháp PaaS chuyên biệt, không chỉ cung cấp các dịch vụ backend tích hợp sẵn mà còn cho phép tùy biến triển khai, giám sát và điều phối máy chủ trò chơi hiệu quả.

1.2.2 Ưu điểm vượt trội của kiến trúc serverless

Kiến trúc serverless mang lại nhiều lợi thế thiết thực để giải quyết các thách thức kể trên:

- Về chi phí, mô hình "trả theo mức sử dụng" giúp giảm đáng kể chi phí vận hành so với các mô hình truyền thống như EC2 hoặc Kubernetes. Với cùng quy mô khoảng 10.000 người dùng/ngày, backend dựa trên AWS Lambda có thể duy trì chi phí dưới 100 USD/tháng, trong khi hệ thống dùng EC2 hoặc EKS có thể tốn 300–500 USD/tháng.
- Về kỹ thuật, serverless hỗ trợ:
 - Tự động mở rộng tài nguyên để xử lý các đợt tải tăng đột biến trong các sự kiện như ra mắt game hay giải đấu.

- Tích hợp dễ dàng với hệ sinh thái AWS, cho phép xây dựng hệ thống backend và máy chủ trò chơi bằng cách kết hợp các dịch vụ như Lambda, API Gateway, DynamoDB, Cognito, SNS/SQS, và Fargate mà không cần triển khai hoặc cấu hình hạ tầng vật lý.

Nhờ đó, giải pháp serverless đặc biệt phù hợp với xu hướng phát triển phần mềm hiện đại – tập trung vào logic nghiệp vụ thay vì hạ tầng kỹ thuật – giúp các studio trò chơi tối ưu hóa nguồn lực, rút ngắn thời gian ra mắt sản phẩm, và nâng cao trải nghiệm người chơi.

1.2.3 Xu thế phát triển tất yếu

Với sự bùng nổ của trò chơi trực tuyến và eSports, nhu cầu về hệ thống backend có khả năng tối ưu hóa tự động và mở rộng đàn hồi đã trở thành yêu cầu sống còn. Kiến trúc serverless không chỉ đáp ứng những yêu cầu này mà còn mở ra khả năng:

- Triển khai multi-region dễ dàng để giảm độ trễ cho người chơi toàn cầu
- Tích hợp các công nghệ tiên tiến như AI/ML cho các dịch vụ như ghép cặp người chơi, phân tích hành vi và chống gian lận.
- Hỗ trợ chơi đa nền tảng giữa mobile, PC và console

Bằng cách kết hợp những ưu thế của serverless với nhu cầu thực tế của ngành xuất bản trò chơi, giải pháp PaaS được đề xuất sẽ trở thành nền tảng lý tưởng cho thế hệ phát triển trò chơi tiếp theo - nơi mà khả năng mở rộng, hiệu suất cao và chi phí tối ưu trở thành những yếu tố không thể tách rời của thành công.

1.3 Đối tượng nghiên cứu

Đối tượng nghiên cứu của đề tài là một nền tảng hỗ trợ triển khai backend cho trò chơi trực tuyến theo mô hình PaaS, cho phép các nhà phát triển dễ dàng xây dựng, triển khai và vận hành backend mà không cần tự thiết lập hạ tầng phức tạp. Nền tảng này hướng tới việc chuẩn hóa và tự động hóa toàn bộ quá trình triển khai backend trò chơi, từ cung cấp các dịch vụ cốt lõi đến quản lý máy chủ trò chơi, nhằm giảm thiểu thời gian phát triển và tối ưu chi phí vận hành.

Cụ thể, nghiên cứu tập trung vào hai nhóm chức năng chính:

- Nhóm dịch vụ backend, bao gồm các chức năng cần thiết để hỗ trợ vận hành một trò chơi trực tuyến hiện đại như: ghép trận, xếp hạng, quản lý thông tin người chơi, giao tiếp trong và ngoài trận, theo dõi trận đấu và các cơ chế phân tích dữ liệu trò chơi.

- Nhóm chức năng quản lý máy chủ trò chơi, bao gồm việc triển khai máy chủ từ các hình ảnh container, thiết lập kết nối với backend, giám sát trạng thái hoạt động, và mở rộng tự động dựa trên lưu lượng thực tế.

Toàn bộ hệ thống được phát triển dựa trên kiến trúc serverless của nền tảng AWS, sử dụng các dịch vụ như Lambda, API Gateway, DynamoDB, AppSync, Cognito, SQS, SNS và Fargate để đảm bảo khả năng mở rộng linh hoạt, hiệu suất cao và giảm thiểu chi phí hạ tầng.

Ngoài ra, một bộ SDK mẫu cũng được xây dựng nhằm hỗ trợ xây dựng và tích hợp máy chủ trò chơi với backend. SDK này cung cấp sẵn các chức năng kết nối backend, quản lý người chơi và phiên chơi, giúp các nhà phát triển có thể tập trung tối đa vào việc xây dựng trải nghiệm trò chơi thay vì xử lý các vấn đề kỹ thuật phía hạ tầng.

1.4 Phạm vi nghiên cứu

Phạm vi nghiên cứu của đề tài giới hạn trong việc thiết kế và triển khai một giải pháp PaaS nhằm hỗ trợ các nhà phát triển trò chơi điện tử triển khai backend một cách nhanh chóng, hiệu quả và tiết kiệm chi phí. Nền tảng hướng đến việc chuẩn hóa quy trình triển khai backend cho các trò chơi trực tuyến, với trọng tâm là cung cấp các dịch vụ thiết yếu và khả năng quản lý máy chủ trò chơi một cách linh hoạt.

Đề tài không đi sâu vào phát triển frontend trò chơi, thiết kế đồ họa hay cơ chế gameplay cụ thể, mà tập trung vào tầng backend – nơi đảm nhận các chức năng hạ tầng quan trọng như ghép trận, xếp hạng, quản lý người chơi, tương tác xã hội, giao tiếp thời gian thực, theo dõi trận đấu và vận hành máy chủ trò chơi. Các dịch vụ này được tích hợp thành một nền tảng thống nhất, cung cấp giao diện lập trình ứng dụng (API) đơn giản và SDK hỗ trợ để giảm thiểu công sức tích hợp cho nhà phát triển.

Trong phạm vi triển khai thực nghiệm, đề tài xây dựng một nền tảng mẫu cho trò chơi cờ vua với khả năng phục vụ đồng thời lên đến 10.000 người chơi. Hệ thống backend và máy chủ trò chơi được triển khai trên hạ tầng serverless của AWS, sử dụng các dịch vụ như Lambda, API Gateway, DynamoDB, AppSync, SQS, SNS, Cognito và Fargate để đảm bảo tính mở rộng tự động, hiệu suất ổn định và độ sẵn sàng cao.

Đề tài cũng tích hợp các cơ chế giám sát và kiểm thử hiệu suất toàn diện nhằm đánh giá mức độ chịu tải và khả năng mở rộng của hệ thống trong điều kiện vận hành thực tế. Các kịch bản mô phỏng được xây dựng để phản ánh hành vi của hàng nghìn người chơi đồng thời, từ đó kiểm tra độ bền vững của kiến trúc cũng như hiệu quả vận hành về mặt chi phí và độ trễ.

1.5 Phương pháp nghiên cứu

Đồ án áp dụng phương pháp nghiên cứu ứng dụng kết hợp với phân tích thực nghiệm, nhằm thiết kế, triển khai và đánh giá một giải pháp PaaS phục vụ triển khai backend cho trò chơi trực tuyến dựa trên kiến trúc serverless. Quá trình nghiên cứu được tiến hành theo lộ trình tuần tự từ phân tích lý thuyết, thiết kế hệ thống, triển khai thực nghiệm đến đánh giá hiệu quả và khả năng mở rộng.

Giai đoạn đầu tiên tập trung vào việc nghiên cứu cơ sở lý thuyết thông qua tổng hợp tài liệu học thuật, báo cáo kỹ thuật và tài liệu chính thức từ AWS. Các chủ đề trọng tâm bao gồm: kiến trúc serverless, tổ chức backend trò chơi, cơ chế triển khai hạ tầng dưới dạng mã, quản lý máy chủ trò chơi, và các dịch vụ nền tảng như xác thực người dùng, ghép trận, xếp hạng, trò chuyện và theo dõi trạng thái trận đấu. Kiến thức nền này là cơ sở để định hình hướng tiếp cận thiết kế hệ thống phù hợp với các yêu cầu thực tiễn trong ngành công nghiệp game.

Dựa trên cơ sở lý thuyết đã tổng hợp, đồ án tiến hành thiết kế kiến trúc tổng thể của hệ thống PaaS với các thành phần dịch vụ được phân lớp rõ ràng: tầng dịch vụ backend và tầng điều phối máy chủ trò chơi. Quá trình thiết kế nhấn mạnh tính mô-đun, khả năng mở rộng, tính sẵn sàng cao và tối ưu chi phí. Kiến trúc được xây dựng sao cho nhà phát triển có thể lựa chọn các dịch vụ cần thiết và triển khai linh hoạt dựa trên nhu cầu của từng trò chơi.

Hệ thống sau đó được hiện thực hóa và triển khai thử nghiệm trên nền tảng điện toán đám mây AWS, sử dụng các dịch vụ serverless chủ đạo như Lambda, API Gateway, DynamoDB, AppSync, Cognito, SQS, SNS và Fargate. Việc triển khai được tự động hóa bằng công cụ AWS SAM theo mô hình hạ tầng dưới dạng mã để đảm bảo khả năng tái sử dụng và kiểm soát phiên bản cấu hình. Một bộ SDK mẫu cũng được phát triển bằng ngôn ngữ Go để đơn giản hóa việc tích hợp máy chủ trò chơi với nền tảng backend.

Sau khi triển khai hệ thống mẫu cho trò chơi cờ vua, đồ án tiến hành kiểm thử hiệu suất bằng công cụ mô phỏng k6, với các kịch bản giả lập lên đến 10.000 người chơi đồng thời. Các chỉ số như độ trễ, khả năng mở rộng, mức tiêu thụ tài nguyên và chi phí vận hành được thu thập, phân tích và sử dụng để tinh chỉnh cấu hình hệ thống.

Cuối cùng, hệ thống được đánh giá toàn diện thông qua việc so sánh với các mô hình triển khai backend truyền thống. Các tiêu chí như chi phí vận hành, độ phức tạp triển khai, thời gian đưa sản phẩm ra thị trường và khả năng mở rộng theo tải được phân tích nhằm làm rõ lợi thế của giải pháp đề xuất. Trên cơ sở đó, đồ án đưa ra các nhận

định tổng quan và định hướng phát triển trong tương lai cho các nền tảng triển khai backend trò chơi hiện đại.

1.6 Mục tiêu nghiên cứu

Mục tiêu chính của đề tài là thiết kế và triển khai một giải pháp PaaS phục vụ cho việc triển khai backend trò chơi điện tử, với cấu trúc mô-đun, linh hoạt và có thể tái sử dụng cho nhiều thể loại game khác nhau. Giải pháp cho phép các nhà phát triển trò chơi dễ dàng tích hợp và vận hành các dịch vụ backend thiết yếu như ghép trận, xếp hạng, quản lý người chơi, giao tiếp thời gian thực, bạn bè và theo dõi trận đấu mà không cần xây dựng lại hệ thống từ đầu.

Một mục tiêu quan trọng của đề tài là hiện thực hóa nền tảng này trên môi trường điện toán đám mây AWS, sử dụng kiến trúc serverless nhằm đạt được khả năng mở rộng tự động, độ sẵn sàng cao và tối ưu chi phí. Bên cạnh đó, hệ thống cần đảm bảo khả năng quản lý máy chủ trò chơi ở mức độ linh hoạt – từ triển khai container, điều phối phiên chơi đến giám sát trạng thái và tài nguyên – giúp rút ngắn quy trình triển khai sản phẩm thực tế.

Đề tài cũng hướng đến việc xây dựng một bộ SDK mẫu để hỗ trợ nhà phát triển tích hợp nhanh máy chủ trò chơi với hệ thống backend, đồng thời cung cấp giao diện lập trình ứng dụng đơn giản và rõ ràng. Các thành phần dịch vụ được tổ chức theo mô hình mô-đun để người dùng có thể lựa chọn, cấu hình và mở rộng độc lập theo đặc thù của từng trò chơi.

Ngoài việc thiết kế và triển khai, một mục tiêu trọng tâm khác là đánh giá định lượng toàn diện hiệu quả vận hành của nền tảng. Thông qua các thử nghiệm mô phỏng tải lớn với hàng nghìn người chơi đồng thời, đề tài sẽ phân tích các chỉ số quan trọng như độ trễ, khả năng chịu tải, mức tiêu thụ tài nguyên và chi phí vận hành nhằm kiểm chứng tính khả thi của giải pháp trong điều kiện thực tế.

Cuối cùng, đề tài đặt mục tiêu cung cấp một nền tảng mẫu hoàn chỉnh, có khả năng mở rộng và dễ áp dụng trong thực tiễn, giúp các nhóm phát triển trò chơi – đặc biệt là các studio nhỏ và vừa – có thể tiết kiệm đáng kể thời gian, công sức và chi phí khi triển khai backend trò chơi, từ đó tập trung hơn vào việc xây dựng trải nghiệm cho người chơi.

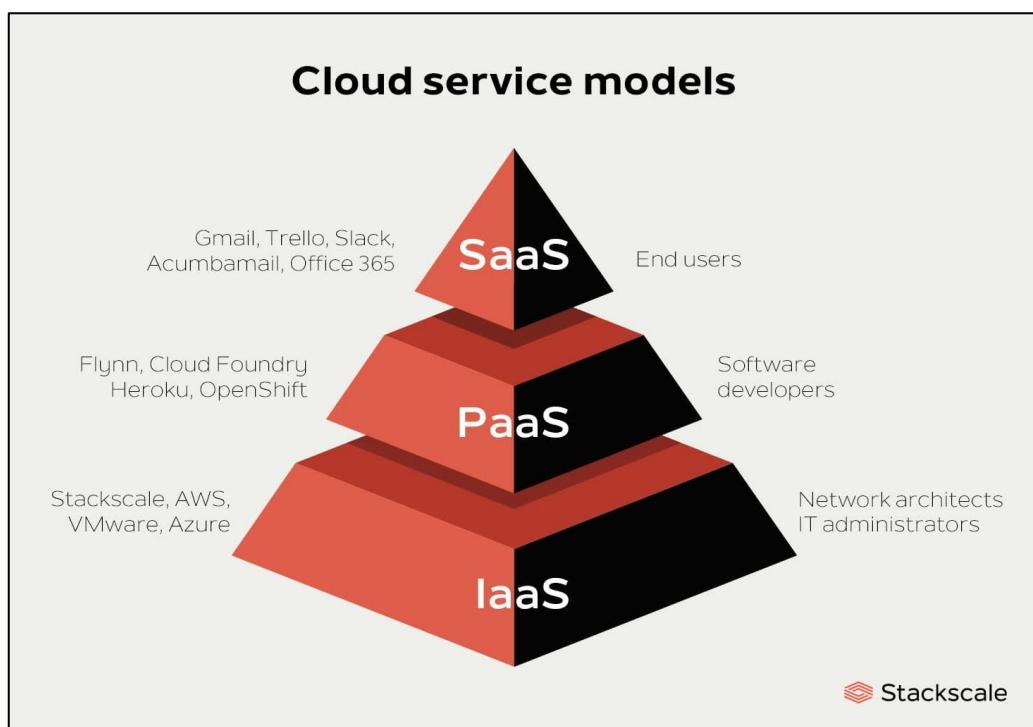
Chương 2: Cơ sở lý thuyết

2.1 Nền tảng dưới dạng dịch vụ

2.1.1 Giới thiệu về điện toán đám mây

Điện toán đám mây là mô hình cung cấp tài nguyên điện toán như máy chủ, lưu trữ, cơ sở dữ liệu, mạng và phần mềm thông qua internet. Thay vì phải đầu tư vào hạ tầng vật lý đắt đỏ và duy trì hệ thống nội bộ, người dùng có thể tận dụng các dịch vụ đám mây để truy cập tài nguyên theo nhu cầu.

Theo định nghĩa của NIST, điện toán đám mây có năm đặc điểm chính, bao gồm khả năng tự phục vụ theo yêu cầu, truy cập thông qua mạng, tài nguyên được gộp chung, khả năng mở rộng linh hoạt và mô hình thanh toán theo mức sử dụng.



Hình 2.1 Mô hình dịch vụ điện toán đám mây

Dịch vụ điện toán đám mây được chia thành ba mô hình chính:

- IaaS (Infrastructur-as-a-Service): Cung cấp tài nguyên phần cứng ảo hóa như máy chủ, lưu trữ và mạng (ví dụ: AWS EC2, Google Compute Engine).
- PaaS (Platform-as-a-Service): Cung cấp nền tảng phát triển ứng dụng mà không cần quản lý hạ tầng (ví dụ: AWS Lambda, Google App Engine).
- SaaS (Software-as-a-Service): Cung cấp phần mềm sẵn sàng sử dụng thông qua internet (ví dụ: Google Drive, Dropbox).

2.1.2 Khái niệm và đặc trưng của PaaS

Nền tảng dưới dạng dịch vụ (Platform as a Service - PaaS) là mô hình dịch vụ điện toán đám mây cung cấp nền tảng phát triển ứng dụng hoàn chỉnh, cho phép người dùng phát triển, triển khai, quản lý, và vận hành ứng dụng mà không cần quan tâm tới việc quản lý cơ sở hạ tầng (phần cứng, hệ điều hành, mạng) bên dưới.

Các đặc trưng chính của mô hình PaaS bao gồm:

- Tự động hóa triển khai: Hỗ trợ việc triển khai ứng dụng một cách nhanh chóng, đơn giản thông qua các công cụ hoặc giao diện lập trình ứng dụng (API).
- Tính linh hoạt và khả năng mở rộng (Scalability): Có khả năng điều chỉnh tài nguyên tự động theo tải thực tế.
- Tính đa nền tảng: Hỗ trợ đa ngôn ngữ lập trình và framework, giúp nhà phát triển linh hoạt trong việc lựa chọn công nghệ phù hợp.
- Tích hợp dễ dàng: Có khả năng tích hợp các dịch vụ cơ sở dữ liệu, phân tích dữ liệu, bảo mật, và các dịch vụ điện toán khác sẵn có trên đám mây.
- Khả năng quản lý tập trung: Cung cấp công cụ để quản lý các vòng đời ứng dụng một cách tập trung, giảm thiểu công sức và chi phí quản trị hạ tầng.

2.1.3 Các thành phần của mô hình PaaS

Một nền tảng PaaS điển hình bao gồm các thành phần sau:

- Cơ sở hạ tầng: Máy chủ, mạng, lưu trữ được quản lý hoàn toàn bởi nhà cung cấp dịch vụ đám mây.
- Middleware: Các framework, thư viện, môi trường runtime, hỗ trợ triển khai và chạy ứng dụng.
- Công cụ phát triển và triển khai: Công cụ hỗ trợ việc lập trình, kiểm thử, quản lý phiên bản mã nguồn, tự động hóa triển khai (CI/CD).
- Quản lý cơ sở dữ liệu: Hỗ trợ các loại cơ sở dữ liệu phổ biến (SQL, NoSQL), đảm bảo việc lưu trữ và truy xuất dữ liệu nhanh chóng, hiệu quả.
- Công cụ giám sát và bảo mật: Cung cấp các công cụ giám sát hiệu năng, logging, phân tích sự kiện, và bảo mật ứng dụng toàn diện.

2.1.4 Lợi ích và hạn chế của mô hình PaaS

Lợi ích

- Rút ngắn thời gian phát triển ứng dụng: Nhà phát triển chỉ cần tập trung vào logic ứng dụng, giảm thời gian và chi phí cho việc quản lý cơ sở hạ tầng.
- Chi phí thấp và tối ưu: Thanh toán theo mức độ sử dụng, không mất chi phí đầu tư lớn ban đầu.
- Khả năng mở rộng và đàn hồi: PaaS tự động scale tài nguyên theo nhu cầu, giúp ứng dụng luôn hoạt động ổn định.
- Tăng hiệu suất phát triển và cộng tác: Các công cụ phát triển và tích hợp liên tục được tích hợp sẵn, giúp nâng cao năng suất làm việc.

Hạn chế

- Giới hạn tùy biến hạ tầng: Người dùng có thể bị hạn chế trong việc tùy chỉnh hệ điều hành hoặc cấu hình chi tiết hạ tầng.
- Rủi ro khóa trói (Vendor lock-in): Việc sử dụng PaaS của một nhà cung cấp duy nhất có thể gây khó khăn trong việc chuyển đổi sang nền tảng khác sau này.

2.1.5 Các nhà cung cấp dịch vụ PaaS phổ biến

Một số nền tảng PaaS phổ biến trên thị trường hiện nay:

- Heroku: Nền tảng đơn giản, dễ sử dụng, hỗ trợ nhiều ngôn ngữ lập trình như Ruby, Python, Node.js, Java, PHP.
- Google App Engine (GAE): Giải pháp của Google hỗ trợ các ứng dụng web, tự động scale và quản lý hoàn toàn.
- AWS Elastic Beanstalk: Tích hợp sâu với hệ sinh thái AWS, hỗ trợ triển khai nhiều ứng dụng web và dịch vụ backend khác nhau.
- Microsoft Azure App Service: Nền tảng mạnh mẽ từ Microsoft, tích hợp tốt với hệ sinh thái .NET, Azure DevOps, hỗ trợ nhiều framework và ngôn ngữ lập trình.
- IBM Cloud Foundry: Nền tảng mã nguồn mở, linh hoạt cao, hỗ trợ nhiều ngôn ngữ và dịch vụ đa dạng.

2.1.6 PaaS trong triển khai backend trò chơi

Trong lĩnh vực trò chơi trực tuyến, việc áp dụng mô hình PaaS giúp các nhà phát triển tập trung hoàn toàn vào xây dựng logic gameplay, đồng thời khai thác tối đa sức

mạnh của điện toán đám mây. Các lợi ích khi triển khai backend game trên mô hình PaaS bao gồm:

- Khả năng scale linh hoạt: Backend game tự động mở rộng theo lượng người chơi, đáp ứng tải bất thường khi có sự kiện game hoặc thời gian cao điểm.
- Giảm chi phí vận hành: Tài nguyên chỉ được sử dụng và trả phí khi cần thiết, giảm chi phí vận hành hệ thống so với mô hình máy chủ truyền thống.
- Giảm thời gian ra mắt game: Việc triển khai ứng dụng trở nên nhanh chóng nhờ các công cụ tích hợp và tự động hóa triển khai.
- Tích hợp dễ dàng các dịch vụ cần thiết: Các tính năng như matchmaking, ranking, chat, phân tích dữ liệu người chơi, được cung cấp và quản lý dễ dàng bởi nền tảng PaaS.

Việc áp dụng PaaS cho backend trò chơi không chỉ tối ưu về mặt kỹ thuật mà còn phù hợp với xu thế phát triển của ngành công nghiệp game ngày càng thiên về dịch vụ và sự linh hoạt trong vận hành.

2.1.7 Kết luận

PaaS là mô hình cung cấp nền tảng để phát triển và vận hành ứng dụng nhanh chóng, hiệu quả, giúp các nhà phát triển tập trung vào giá trị cốt lõi của ứng dụng thay vì quản lý hạ tầng. Với những ưu điểm nổi bật về khả năng mở rộng, chi phí, tốc độ triển khai, mô hình PaaS ngày càng phổ biến và trở thành xu hướng tất yếu trong việc triển khai backend cho các ứng dụng hiện đại, đặc biệt là trong lĩnh vực game trực tuyến.

2.2 Kiến trúc điện toán phi máy chủ

2.2.1 Khái niệm và đặc trưng

Điện toán phi máy chủ (serverless) là một mô hình điện toán đám mây trong đó người dùng triển khai và thực thi ứng dụng mà không cần quản lý máy chủ. Toàn bộ hạ tầng và khả năng mở rộng được nhà cung cấp dịch vụ đám mây đảm nhiệm, giúp giảm bớt gánh nặng vận hành hệ thống. Trong kiến trúc này, các thành phần ứng dụng được triển khai dưới dạng các hàm nhỏ (Function-as-a-Service - FaaS), có khả năng phản ứng theo sự kiện và chỉ chạy khi được kích hoạt.

Cốt lõi của serverless là các hàm sự kiện hoạt động độc lập và có thể được gọi từ nhiều nguồn khác nhau như API Gateway, dịch vụ lưu trữ, hàng đợi tin nhắn hoặc cơ sở dữ liệu. Một trong những dịch vụ phổ biến nhất cho mô hình này là AWS Lambda, cho phép thực thi mã mà không cần cấu hình hoặc quản lý hạ tầng máy chủ.

Mô hình serverless có những ưu điểm đáng kể như khả năng tự động mở rộng theo tải hệ thống, tối ưu hóa chi phí do chỉ cần trả tiền cho thời gian thực thi thực tế, giảm tải việc quản lý hạ tầng, và tăng tốc độ triển khai phần mềm. Tuy nhiên, nó cũng có một số hạn chế như độ trễ khởi động (cold start), giới hạn thời gian thực thi của mỗi hàm, và độ phức tạp trong việc giám sát hệ thống phân tán.

Các thành phần chính trong kiến trúc serverless bao gồm:

- FaaS (Function-as-a-Service): Cho phép chạy các đoạn mã nhỏ khi có sự kiện kích hoạt mà không cần duy trì máy chủ.
- BaaS (Backend-as-a-Service): Cung cấp các dịch vụ backend như cơ sở dữ liệu, xác thực người dùng, lưu trữ tệp và xử lý hàng đợi tin nhắn mà không cần máy chủ chuyên dụng.
- Dịch vụ quản lý API: API Gateway đóng vai trò trung gian giữa người dùng và các hàm serverless, giúp định tuyến yêu cầu và quản lý xác thực.
- Cơ sở dữ liệu không máy chủ: Hệ thống cơ sở dữ liệu như AWS DynamoDB hoặc Firebase Firestore được thiết kế để hoạt động liền mạch với serverless, cung cấp khả năng lưu trữ dữ liệu mà không yêu cầu máy chủ liên tục.

Về mặt vận hành, serverless giúp các hệ thống dễ dàng mở rộng theo lưu lượng thực tế mà không cần quản lý tài nguyên máy chủ. Khi một yêu cầu đến hệ thống, dịch vụ serverless sẽ khởi tạo môi trường thực thi, chạy hàm tương ứng và tự động giải phóng tài nguyên sau khi hoàn tất. Do đó, kiến trúc này phù hợp với các ứng dụng có tải không ổn định hoặc yêu cầu xử lý theo sự kiện, chẳng hạn như xử lý dữ liệu thời gian thực, API backend và tự động hóa tác vụ.

Mặc dù kiến trúc serverless mang lại nhiều lợi ích, việc triển khai cần xem xét các yếu tố như thời gian khởi động, tối ưu hóa hiệu suất, và tích hợp với các dịch vụ truyền thống. Vì vậy, trong thực tế, mô hình này thường được kết hợp với các kiến trúc khác như microservices để tạo ra hệ thống linh hoạt và hiệu quả.

2.2.2 Một số giải pháp Serverless cốt lõi của AWS

AWS cung cấp nhiều dịch vụ serverless hỗ trợ xây dựng ứng dụng mà không phải lo về cơ sở hạ tầng. Dưới đây là một số giải pháp serverless cốt lõi của AWS:

Tính toán

- AWS Lambda: Thực thi mã theo sự kiện mà không cần quản lý máy chủ.

- AWS Fargate: Triển khai container không máy chủ, hỗ trợ các máy chủ trò chơi độc lập hoặc theo phiên.
- AWS App Runner: Triển khai các ứng dụng web hoặc API container hóa dễ dàng.

Cơ sở dữ liệu & lưu trữ

- Amazon DynamoDB: NoSQL serverless, hiệu suất cao, dùng để lưu trạng thái ván chơi, người dùng, kết quả...
- Amazon S3: Lưu trữ đối tượng serverless, phục vụ ảnh đại diện, dữ liệu huấn luyện, file game logs.
- Amazon Aurora Serverless: Dành cho cơ sở dữ liệu quan hệ không liên tục.

Truyền tin và điều phối sự kiện

- Amazon EventBridge: Kết nối sự kiện giữa các dịch vụ (ví dụ: trigger deploy khi người dùng tạo backend mới).
- Amazon SQS / SNS: Xử lý bất đồng bộ và gửi thông báo sự kiện (ví dụ: ghép trận thành công).
- AWS Step Functions: Điều phối quy trình nhiều bước như auto deploy backend, gỡ lỗi, và rollback tự động.

API và realtime

- Amazon API Gateway: Tạo REST API hoặc WebSocket API cho các dịch vụ trò chơi.
- AWS AppSync: Cung cấp API GraphQL realtime, phục vụ cập nhật trạng thái ván chơi và trò chuyện.

Giám sát & phân tích

- AWS CloudWatch: Thu thập logs và metrics hệ thống.
- AWS X-Ray: Phân tích và trace luồng hoạt động giữa các thành phần serverless.

2.2.3 So sánh các mô hình triển khai backend trò chơi

Trong lĩnh vực phát triển backend cho trò chơi trực tuyến, hai mô hình kiến trúc phổ biến nhất hiện nay là Monolithic và Microservices. Đây là hai cách tiếp cận truyền thống, phù hợp với các giai đoạn phát triển và quy mô sản phẩm khác nhau. Monolithic đơn giản và dễ tiếp cận trong giai đoạn đầu, trong khi Microservices đem lại khả năng mở rộng và bảo trì linh hoạt hơn nhờ phân tách rõ ràng giữa các thành phần chức năng.

Gần đây, kiến trúc serverless đã và đang trở thành một xu hướng nổi bật nhờ khả năng mở rộng tự động, chi phí vận hành tối ưu và sự đơn giản trong quản lý hạ tầng. Với sự phát triển nhanh chóng của công nghệ đám mây, serverless ngày càng phù hợp với các hệ thống backend có lưu lượng truy cập biến động mạnh — một đặc điểm điển hình trong các trò chơi trực tuyến hiện nay.

Trong đồ án này, kiến trúc serverless được lựa chọn làm nền tảng cho giải pháp PaaS, hỗ trợ triển khai backend trò chơi một cách hiệu quả. Để làm rõ cơ sở lựa chọn này, bảng sau đây trình bày so sánh giữa ba mô hình kiến trúc backend: Monolithic, Microservices và Serverless – mỗi mô hình mang theo những ưu điểm và hạn chế riêng, ảnh hưởng trực tiếp đến khả năng triển khai, mở rộng và duy trì hệ thống trong thực tiễn

Bảng 2.1 So sánh các mô hình kiến trúc backend trò chơi

Tiêu chí	Monolithic	Microservices	Serverless
Tính mở rộng	Kém (toàn khói)	Tốt (dịch vụ độc lập)	Rất tốt (tự động theo nhu cầu)
Quản lý hạ tầng	Đơn giản	Phức tạp (Kubernetes, CI/CD, v.v.)	Tối giản (hạ tầng do AWS quản lý)
Khả năng cập nhật	Khó khăn (ảnh hưởng toàn hệ thống)	Dễ (triển khai riêng lẻ)	Dễ (cập nhật từng hàm, dịch vụ nhỏ)
Chi phí khi tải thấp	Thấp	Trung bình	Rất thấp (trả theo thời gian thực thi)
Chi phí khi tải cao	Cao (scale toàn bộ)	Cao (phải scale nhiều service cùng lúc)	Cân bằng tốt nhờ khả năng scale động
Khả năng giám sát, đo lường	Hạn chế	Phức tạp nhưng mạnh nếu đủ công cụ	Tích hợp sẵn (CloudWatch, X-Ray...)
Phù hợp cho trò chơi	Game nhỏ, offline, casual	Game AAA, MMO, eSports lớn	Game online vừa & nhỏ, dễ nhân rộng và tái sử dụng

2.2.4 Kết luận

Kiến trúc serverless đóng vai trò then chốt trong việc hiện thực hóa một giải pháp PaaS hiện đại cho backend trò chơi. Với khả năng tự động mở rộng, tối ưu chi phí vận hành và triển khai nhanh chóng, serverless đặc biệt phù hợp với môi trường phát triển game – nơi mà lưu lượng người dùng thường dao động mạnh và yêu cầu khắt khe về hiệu năng, độ trễ cũng như thời gian ra mắt sản phẩm.

Khác với mô hình monolithic khó mở rộng hoặc microservices đòi hỏi đầu tư lớn về hạ tầng, kiến trúc serverless cho phép chia nhỏ các chức năng backend thành những thành phần độc lập, phản ứng theo sự kiện và được quản lý hoàn toàn bởi nhà cung cấp đám mây.

Trong phạm vi đồ án, kiến trúc này được áp dụng để xây dựng một nền tảng PaaS có khả năng hỗ trợ triển khai các dịch vụ backend thiết yếu như xác thực người dùng, ghép trận, xếp hạng, trò chuyện và phân tích, đồng thời điều phối các máy chủ trò chơi container hóa bằng ECS Fargate. Việc tích hợp sâu với các dịch vụ như Lambda, API Gateway, DynamoDB, AppSync, cùng hệ thống hàng đợi và giám sát (SQS, SNS, CloudWatch) tạo nên một nền tảng linh hoạt, dễ mở rộng và dễ sử dụng đối với cả các studio nhỏ lẫn các nhà phát triển độc lập.

Do đó, serverless không chỉ là một lựa chọn công nghệ, mà còn là nền tảng cốt lõi định hình kiến trúc tổng thể của hệ thống, mở đường cho một hướng tiếp cận mới trong việc phát triển backend trò chơi có khả năng tái sử dụng, dễ vận hành và phù hợp với xu thế cloud-native hiện đại.

2.3 Cơ sở hạ tầng dưới dạng mã

2.3.1 Khái niệm

Cơ sở hạ tầng dưới dạng mã (Infrastructure as Code - IaC) là một phương pháp quản lý và cung cấp hạ tầng thông qua mã hóa thay vì cấu hình thủ công. Điều này cho phép các tài nguyên như máy chủ, cơ sở dữ liệu, mạng và dịch vụ được định nghĩa dưới dạng mã có thể tái sử dụng, kiểm soát phiên bản và tự động triển khai. IaC giúp giảm thiểu sai sót do thao tác thủ công, đảm bảo tính đồng nhất giữa các môi trường và tăng cường khả năng mở rộng hệ thống.

IaC có thể được triển khai theo hai cách chính: mô hình khai báo (declarative) và mô hình mệnh lệnh (imperative). Trong mô hình khai báo, người dùng định nghĩa trạng thái mong muốn của hạ tầng, và công cụ IaC sẽ tự động thực hiện các bước cần thiết để

đạt được trạng thái đó. Ngược lại, mô hình mệnh lệnh yêu cầu chỉ rõ từng bước để cấu hình hạ tầng.

Các công cụ phổ biến trong IaC bao gồm:

- AWS CloudFormation: Giúp quản lý tài nguyên AWS bằng cách sử dụng các tệp YAML hoặc JSON để mô tả cơ sở hạ tầng.
- Terraform: Một công cụ mã nguồn mở hỗ trợ nhiều nhà cung cấp dịch vụ đám mây, sử dụng ngôn ngữ HashiCorp Configuration Language (HCL) để định nghĩa tài nguyên.
- AWS SAM (Serverless Application Model): Hỗ trợ triển khai các ứng dụng serverless trên AWS bằng cách mở rộng CloudFormation với cú pháp đơn giản hơn.
- Pulumi: Cung cấp khả năng viết IaC bằng các ngôn ngữ lập trình như Python, Go, TypeScript thay vì chỉ YAML/JSON.

Trong kiến trúc serverless, IaC đặc biệt quan trọng vì nó giúp triển khai và quản lý các hàm, API Gateway, cơ sở dữ liệu, hệ thống định danh và các dịch vụ liên quan một cách tự động, đảm bảo tính đồng nhất và có thể kiểm soát được toàn bộ hệ thống.

2.3.1 AWS SAM

AWS Serverless Application Model (AWS SAM) là một framework mở rộng của AWS CloudFormation, giúp đơn giản hóa việc triển khai và quản lý các ứng dụng serverless. AWS SAM cung cấp cú pháp đơn giản để định nghĩa các tài nguyên như:

- AWS Lambda: Xử lý logic nghiệp vụ mà không cần quản lý server.
- API Gateway: Cung cấp API RESTful và WebSocket để giao tiếp với client.
- DynamoDB: Lưu trữ dữ liệu phi quan hệ với hiệu suất cao.
- AWS Step Functions: Điều phối luồng xử lý nghiệp vụ phức tạp.
- AWS SAM sử dụng tệp ‘template.yaml’ để định nghĩa hạ tầng, tuân theo cú pháp YAML. Cấu trúc tệp bao gồm:
 - Transform: Xác định sử dụng AWS SAM.
 - Globals: Cấu hình mặc định cho tất cả tài nguyên.
 - Resources: Định nghĩa tài nguyên như Lambda, API Gateway, DynamoDB.

2.3.2 Kết luận

Việc áp dụng IaC đóng vai trò quan trọng trong việc đảm bảo tính nhất quán, tự động hóa và khả năng mở rộng của hệ thống. Với kiến trúc serverless, việc triển khai và

quản lý tài nguyên trên AWS đòi hỏi một phương pháp tiếp cận có hệ thống để giảm thiểu sai sót thủ công và đảm bảo tái sử dụng cấu hình.

AWS SAM được lựa chọn làm công cụ chính để triển khai hạ tầng serverless nhờ khả năng đơn giản hóa việc định nghĩa các tài nguyên như AWS Lambda, API Gateway, DynamoDB và các dịch vụ liên quan. Điều này không chỉ giúp rút ngắn thời gian triển khai mà còn tạo điều kiện thuận lợi cho việc kiểm soát phiên bản, nâng cấp hệ thống và mở rộng dịch vụ một cách linh hoạt.

Ngoài ra, IaC còn có đóng góp lớn trong việc hỗ trợ triển khai backend như một giải pháp PaaS. Thay vì phải thiết lập từng thành phần hạ tầng một cách thủ công, IaC cho phép tự động hóa toàn bộ quy trình triển khai, giúp lập trình viên tập trung vào việc phát triển logic ứng dụng thay vì quản lý hạ tầng. Với IaC, hệ thống backend có thể được triển khai nhanh chóng với các dịch vụ tích hợp sẵn, giúp giảm đáng kể thời gian đưa sản phẩm ra thị trường.

Việc sử dụng IaC không chỉ giúp tái tạo hạ tầng một cách nhất quán giữa các môi trường (phát triển, kiểm thử, sản xuất), mà còn tối ưu hóa chi phí vận hành, nâng cao khả năng mở rộng tự động và giảm thiểu rủi ro triển khai. Trong bối cảnh các ứng dụng serverless và kiến trúc cloud-native ngày càng trở nên phổ biến, IaC sẽ tiếp tục là một công cụ quan trọng giúp quản lý hạ tầng một cách hiệu quả và linh hoạt.

2.4 Hệ thống xếp hạng người chơi

2.4.1 Giới thiệu

Hệ thống xếp hạng người chơi là một thành phần quan trọng trong các trò chơi có yếu tố cạnh tranh, giúp ước lượng trình độ của người chơi dựa trên lịch sử thi đấu và kết quả đối đầu. Việc duy trì một hệ thống xếp hạng hiệu quả không chỉ giúp đảm bảo sự công bằng trong ghép trận, mà còn nâng cao trải nghiệm người chơi thông qua việc phản ánh sát với thực lực và quá trình cải thiện kỹ năng theo thời gian.

Các thuật toán phổ biến như Elo [13], Glicko [14] và TrueSkill [15] đều sử dụng mô hình xác suất để đánh giá năng lực tương đối giữa các người chơi, từ đó điều chỉnh điểm xếp hạng sau mỗi trận đấu. Mỗi thuật toán mang đến các đặc điểm riêng: Elo đơn giản và dễ triển khai, Glicko bổ sung thêm độ tin cậy và biến động, còn TrueSkill hỗ trợ hiệu quả cho các trò chơi nhiều người chơi hoặc theo đội nhóm.

Trong khuôn khổ án này, hệ thống xếp hạng được thiết kế dưới dạng một dịch vụ độc lập, có thể cấu hình và tái sử dụng linh hoạt. Nhà phát triển có thể lựa chọn thuật

toán phù hợp với đặc thù của trò chơi — ví dụ như dùng Glicko cho các game đối kháng 1v1 như cờ vua, hoặc TrueSkill cho game MOBA hay FPS có nhiều người chơi. Việc đóng gói xếp hạng thành mô-đun riêng biệt giúp tăng tính mở rộng, dễ bảo trì và dễ tích hợp với hệ thống ghép trận hoặc bảng thành tích tổng hợp của trò chơi.

Cách tiếp cận này phù hợp với định hướng của nền tảng backend trò chơi dạng PaaS, nơi mỗi trò chơi có thể tùy chọn dịch vụ xếp hạng theo yêu cầu, đồng thời tận dụng hạ tầng serverless để đảm bảo hiệu suất cao và khả năng mở rộng tự động theo tải thực tế.

2.4.2 Một số hệ thống xếp hạng phổ biến

Hệ thống Elo

Thuật toán Elo là một phương pháp xếp hạng đơn giản nhưng hiệu quả, được phát triển bởi Arpad Elo và ban đầu áp dụng trong cờ vua. Hiện nay, Elo đã trở thành một trong những thuật toán phổ biến nhất để đánh giá trình độ người chơi trong nhiều hệ thống trò chơi cạnh tranh.

Hệ thống Elo hoạt động dựa trên nguyên lý xác suất: mỗi người chơi được gán một mức điểm số, và từ điểm số đó có thể tính được xác suất chiến thắng trước đối thủ. Sau mỗi trận đấu, điểm số của người chơi được điều chỉnh dựa trên kết quả thực tế và kết quả kỳ vọng. Nếu một người chơi chiến thắng trước đối thủ mạnh hơn, họ sẽ nhận được nhiều điểm hơn so với khi thắng một đối thủ yếu hơn. Ngược lại, thất bại trước đối thủ yếu sẽ khiến họ mất nhiều điểm hơn.

Công thức cập nhật điểm Elo được tính như sau:

$$R' = R + K * (S - E)$$

Trong đó:

- R là điểm hiện tại của người chơi
- R' là điểm sau khi cập nhật
- S là kết quả thực tế (1 nếu thắng, 0 nếu thua, 0.5 nếu hòa)
- E là kết quả kỳ vọng (dựa trên chênh lệch điểm Elo giữa hai người chơi)
- K là hệ số điều chỉnh (hệ số K càng lớn thì điểm số thay đổi càng mạnh)

Công thức tính kết quả kỳ vọng E là:

$$E = 1 / (1 + 10^{((R_{opponent} - R_{player})/400)})$$

Thuật toán Elo có những ưu điểm như đơn giản, dễ triển khai và dễ hiểu với người dùng. Tuy nhiên, nó cũng tồn tại một số hạn chế:

- Không thể hiện được độ tin cậy trong điểm số (ví dụ: người chơi mới chưa có nhiều trận đấu)
- Không phản ánh được sự thay đổi nhanh trong phong độ
- Không phù hợp cho các hệ thống có biến động mạnh về số lượng trận hoặc người chơi

Mặc dù vậy, Elo vẫn là lựa chọn phù hợp cho nhiều trò chơi có tính đối kháng một đối một (1v1), đặc biệt khi sự đơn giản và hiệu quả là ưu tiên hàng đầu trong giai đoạn triển khai ban đầu.

Hệ thống Glicko

Thuật toán Glicko là một cải tiến của hệ thống Elo, được phát triển bởi Mark Glickman nhằm khắc phục các hạn chế của Elo trong việc đo lường độ tin cậy và sự biến động trong kỹ năng người chơi theo thời gian. Glicko không chỉ đánh giá điểm số (rating) mà còn bổ sung thêm hai yếu tố quan trọng: độ tin cậy (rating deviation - RD) và độ biến động (volatility).

Các thành phần chính trong hệ thống Glicko:

- Rating (R): tương tự như điểm Elo, phản ánh trình độ hiện tại của người chơi.
- Rating Deviation (RD): đại diện cho độ không chắc chắn của điểm rating. Người chơi có RD thấp là người có điểm số ổn định (do thi đấu thường xuyên), còn người có RD cao thường là người ít hoạt động hoặc mới tham gia hệ thống.
- Volatility (σ): đo lường mức độ biến động trong kỹ năng của người chơi theo thời gian.

Sau mỗi chu kỳ thi đấu (ví dụ: một hoặc nhiều trận), hệ thống Glicko sẽ cập nhật lại cả ba giá trị trên. Nếu một người chơi thi đấu thường xuyên với kết quả ổn định, RD sẽ giảm dần, phản ánh mức độ tin cậy tăng lên. Ngược lại, nếu người chơi nghỉ dài hạn hoặc có phong độ thất thường, RD và σ sẽ tăng lên.

Ưu điểm của Glicko so với Elo:

- Phân biệt được người chơi ổn định và người chơi thiếu dữ liệu.
- Phản ánh tốt hơn sự thay đổi trong phong độ thi đấu.

- Tạo điều kiện xếp hạng hợp lý hơn cho người chơi mới mà không cần khởi tạo điểm số bằng tay.

Hệ thống Glicko đặc biệt phù hợp với các nền tảng trò chơi có tính chất cạnh tranh liên tục và có số lượng lớn người chơi, trong đó độ tin cậy và sự thay đổi kỹ năng đóng vai trò quan trọng trong việc ghép trận công bằng. Trong thực tế, phiên bản nâng cấp Glicko thường được sử dụng do tích hợp tốt hơn với biến động và khả năng hội tụ nhanh.

Hệ thống TrueSkill

TrueSkill là hệ thống xếp hạng được phát triển bởi Microsoft, chủ yếu để sử dụng trong các nền tảng trò chơi trực tuyến có nhiều người chơi như Xbox Live. TrueSkill mở rộng các ý tưởng của Elo và Glicko để xử lý không chỉ trận đấu 1v1, mà còn các trận đấu có nhiều người chơi và nhiều đội tham gia cùng lúc.

Trong TrueSkill, mỗi người chơi được mô hình hóa bằng hai thông số:

- Mu (μ): kỳ vọng điểm số thực sự của người chơi (tương tự như rating trong Elo).
- Sigma (σ): độ lệch chuẩn, biểu thị mức độ không chắc chắn về kỹ năng thực tế của người chơi.

Sau mỗi trận đấu, hệ thống sử dụng kỹ thuật suy luận Bayes để cập nhật giá trị μ và σ của từng người chơi dựa trên kết quả thi đấu và thông tin ban đầu. Người chơi có σ nhỏ sẽ có điểm số ổn định hơn, trong khi người chơi có σ lớn sẽ có sự thay đổi mạnh sau mỗi trận đấu.

Ưu điểm của TrueSkill:

- Hỗ trợ nhiều người chơi và nhiều đội trong cùng một trận đấu.
- Phân biệt rõ ràng giữa kỹ năng thực sự và độ không chắc chắn về kỹ năng.
- Tối ưu hóa việc ghép trận nhanh chóng mà vẫn đảm bảo cân bằng.

Nhờ khả năng mở rộng tốt, TrueSkill rất phù hợp cho các nền tảng trò chơi trực tuyến hiện đại, nơi số lượng người chơi tham gia đông và đa dạng. TrueSkill cũng cung cấp cơ sở vững chắc cho việc xây dựng hệ thống xếp hạng động, thích ứng tốt với sự thay đổi liên tục trong hệ sinh thái người chơi.

Bảng 2.2 So sánh các hệ thống xếp hạng

Thuật toán	Ưu điểm	Hạn chế	Phù hợp cho trò chơi

Elo	Dễ triển khai, phô biến, tính toán đơn giản	Không phản ánh rõ sự biến động trình độ nhanh, độ tin cậy không cao	Game casual, game không quá cạnh tranh, chơi 1v1
Glicko	Phản ánh tốt độ biến động, độ tin cậy, phù hợp game đấu trí (như cờ vua)	Công thức phức tạp hơn Elo một chút	Cờ vua, cờ vây, cờ tướng, và game đối kháng trí tuệ khác
TrueSkill	Rất mạnh trong multiplayer, ghép trận theo đội nhóm	Phức tạp trong tính toán, không cần thiết với các game đấu trí 1v1	Các game MOBA, FPS, Battle Royale

2.4.3 Kết luận

Các thuật toán xếp hạng đóng vai trò nền tảng trong việc xây dựng hệ thống đánh giá và ghép trận công bằng cho người chơi. Những mô hình như Elo, Glicko, hay TrueSkill cung cấp các cơ chế xác suất đáng tin cậy để ước lượng trình độ và phản ánh sự thay đổi năng lực theo thời gian.

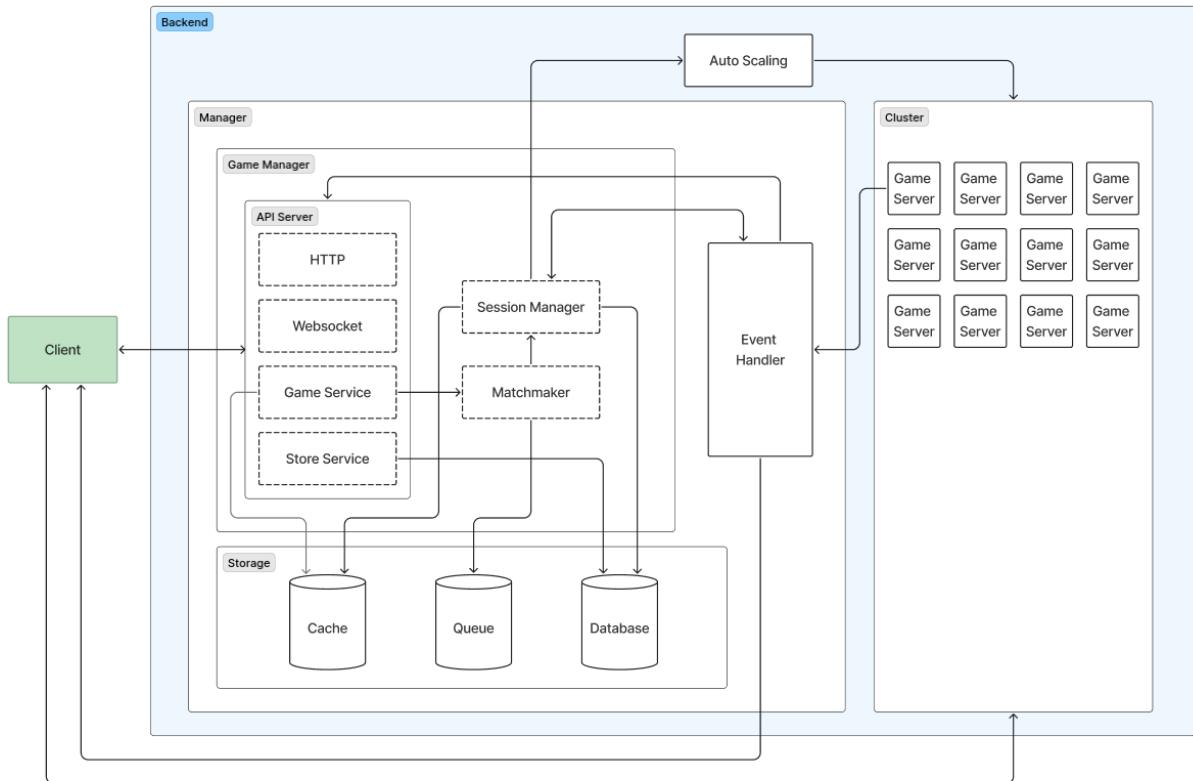
Trong đồ án này, hệ thống xếp hạng không được triển khai cố định theo một thuật toán cụ thể, mà được thiết kế dưới dạng một dịch vụ mô-đun có khả năng cấu hình linh hoạt. Cách tiếp cận này cho phép nhà phát triển lựa chọn thuật toán phù hợp với đặc điểm của trò chơi — ví dụ: Glicko cho trò chơi đối kháng 1v1 như cờ vua, hoặc TrueSkill cho các game có nhiều người chơi, chơi theo đội nhóm.

Việc đóng gói hệ thống xếp hạng thành một mô-đun độc lập giúp dễ bảo trì, mở rộng và tích hợp vào các trò chơi khác nhau trong cùng một nền tảng. Bên cạnh đó, thiết kế này cũng tạo điều kiện thuận lợi để bổ sung các thuật toán xếp hạng mới trong tương lai, hoặc kết hợp với các mô hình học máy nhằm phân tích hành vi người chơi ở mức độ sâu hơn. Nhờ đó, nền tảng có thể phục vụ nhiều thể loại game với yêu cầu khác nhau mà vẫn duy trì được tính nhất quán, linh hoạt và hiệu quả.

Chương 3: Thiết kế backend trò chơi dựa trên kiến trúc serverless

3.1 Thiết kế hệ thống

3.1.1 Kiến trúc tổng quát



Hình 3.1 Kiến trúc tổng quát của hệ thống

Hình 3.1 mô tả kiến trúc tổng quát của một nền tảng trò chơi, bao gồm các thành phần như sau:

A - Game Client

Tùy vào triển khai của trò chơi, client có thể là ứng dụng mobile, web hoặc desktop.

B - Game Manager

Game Manager trong hệ thống có vai trò trung tâm trong việc điều phối và quản lý các hoạt động liên quan đến trò chơi, bao gồm xử lý yêu cầu từ người chơi, quản lý phiên chơi, ghép cặp đối thủ và xử lý các sự kiện hệ thống. Thành phần cốt lõi của Game Manager là API Server, nơi tiếp nhận và xử lý các yêu cầu từ client thông qua các giao thức HTTP và WebSocket. HTTP được sử dụng để xử lý các yêu cầu RESTful như đăng nhập, truy xuất thông tin người chơi, hoặc lưu trữ kết quả trận đấu. Trong khi đó,

WebSocket đảm bảo kết nối thời gian thực giữa server và client, cho phép trao đổi thông tin liên tục mà không cần yêu cầu lặp lại, giúp giảm độ trễ trong các hoạt động như cập nhật kết quả trận đấu hoặc gửi thông báo tức thời.

Bên trong API Server có hai thành phần quan trọng: Game Service và Store Service. Game Service chịu trách nhiệm quản lý logic trò chơi, bao gồm điều phối các sự kiện trong game, xác định trạng thái trò chơi, và đảm bảo sự nhất quán của dữ liệu giữa các người chơi tham gia. Store Service thực hiện nhiệm vụ lưu trữ và truy xuất dữ liệu trò chơi, chẳng hạn như kết quả trận đấu, thông tin người chơi, và lịch sử chơi game.

Để quản lý các phiên chơi hiệu quả, hệ thống sử dụng Session Manager, một thành phần theo dõi và duy trì trạng thái của từng trận đấu. Session Manager làm việc chặt chẽ với Matchmaker, thành phần chịu trách nhiệm ghép cặp người chơi dựa trên các tiêu chí như điểm ELO, độ trễ mạng, hoặc sở thích chơi game. Matchmaker có thể sử dụng hàng đợi tin nhắn để xử lý các yêu cầu ghép cặp một cách công bằng và tối ưu.

Ngoài ra, Game Manager Service cũng bao gồm Event Handler, một mô-đun quan trọng đảm bảo rằng các sự kiện hệ thống được xử lý một cách hợp lý. Event Handler tiếp nhận và xử lý các sự kiện như trạng thái trò chơi được cập nhật, kết thúc phiên chơi, xin hòa, đầu hàng. Nó có thể thực hiện các tác vụ như thông báo đến các thành phần liên quan hoặc cập nhật trạng thái phiên chơi trong cơ sở dữ liệu.

Để tối ưu hóa hiệu suất và độ tin cậy, Game Manager tích hợp với ba loại hệ thống lưu trữ: Cache, Queue, và Database. Cache được sử dụng để lưu trữ dữ liệu tạm thời nhằm tăng tốc truy vấn, giảm tải cho cơ sở dữ liệu. Queue đóng vai trò quan trọng trong việc xử lý các tác vụ không đồng bộ, chẳng hạn như ghép cặp người chơi hoặc thông báo hệ thống. Database là kho lưu trữ dữ liệu lâu dài, đảm bảo rằng thông tin như tài khoản người chơi, lịch sử trận đấu và kết quả thi đấu được duy trì và truy xuất một cách bền vững.

C - Game Server

Ở mô hình này, máy chủ trò chơi là một máy chủ có thẩm quyền (authoritative server), nghĩa là mọi đầu vào từ client đều được xử lý và xác thực trên máy chủ. Sau đó, client trò chơi sẽ kiểm tra dữ liệu đã được máy chủ xác thực để hiển thị trạng thái chính xác của trò chơi.

Ngoài ra, máy chủ sẽ được triển khai dưới dạng vi máy chủ (micro-sized server), nghĩa là mỗi máy chủ chỉ quản lý tối đa một lượng nhỏ người chơi để giảm thiểu ảnh hưởng giữa các phiên trò chơi do việc sử dụng tài nguyên hoặc lỗi hệ thống. Ví dụ, nếu

một trận đấu gặp sự cố và gây crash máy chủ, các trận đấu ở máy chủ khác vẫn tiếp tục hoạt động bình thường và những trận đấu ở máy chủ bị lỗi sẽ nhanh chóng được khôi phục và chạy trên các máy chủ khác. Việc sử dụng micro-server cũng giúp hệ thống scale dễ dàng và tối ưu chi phí hơn nhờ sử dụng cấu hình máy chủ ở mức thấp nhất có thể và chỉ tăng tài nguyên theo từng lượng nhỏ, tránh lãng phí tài nguyên không sử dụng.

Mô hình này nhắm đến triển khai cả những trò chơi cần xử lý thời gian thực hoặc có yếu tố chính xác về thời gian trong logic. Do đó máy chủ cần lưu trữ trạng thái trò chơi, xử lý đầu vào ngay lập tức và kết nối trực tiếp với client để tối thiểu hóa độ trễ. Để giao tiếp với server, có thể sử dụng các giao thức thời gian thực như UDP, TCP, WebSocket kết hợp với phương thức xác thực người dùng.

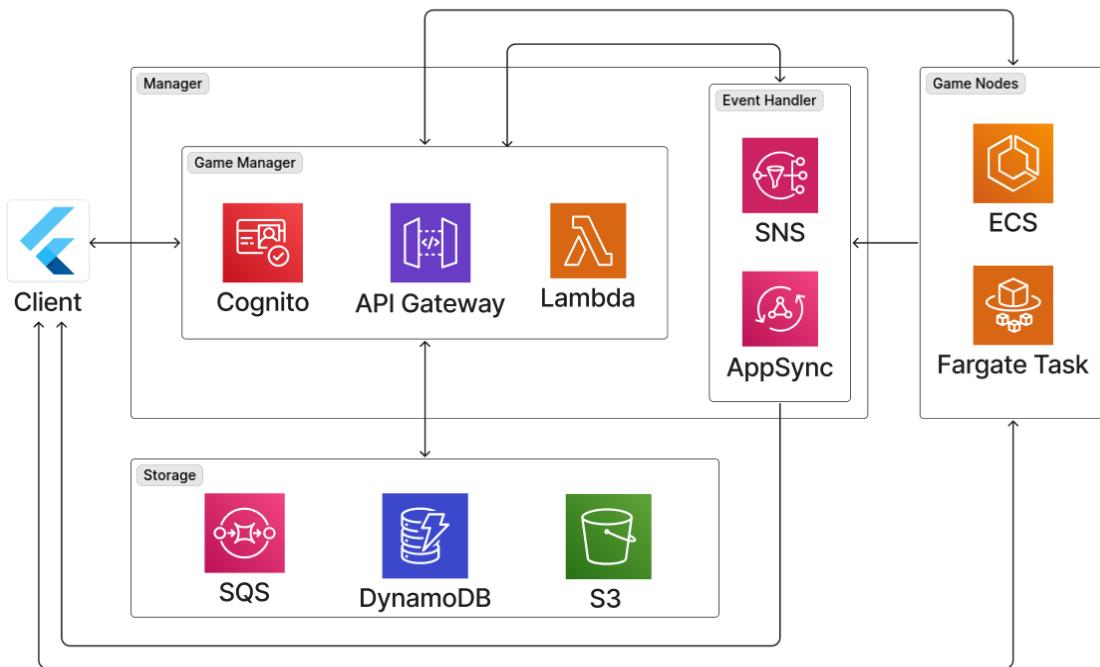
D - Auto Scaling

Auto Scaling trong hệ thống có vai trò quan trọng trong việc tối ưu hóa hiệu suất, đảm bảo khả năng mở rộng động cho các máy chủ trò chơi dựa trên tải thực tế. Cấu trúc của Auto Scaling gồm hai thành phần: Controller và Cluster. Controller có nhiệm vụ theo dõi và điều chỉnh số lượng máy chủ trò chơi trong cụm (Cluster) dựa trên các yếu tố như:

- Lượng người chơi hoạt động: Khi số lượng người chơi tăng lên, cần mở rộng thêm các Game Server để xử lý. Khi ít người chơi, số lượng server sẽ giảm để tiết kiệm tài nguyên.
- Mức sử dụng tài nguyên: Nếu các máy chủ hiện tại đạt mức sử dụng CPU/memory cao, Auto Scaling sẽ khởi tạo thêm các Game Server mới.
- Hàng đợi ghép trận: Nếu hàng đợi trong Matchmaker chứa nhiều yêu cầu chưa xử lý, Auto Scaling có thể khởi tạo thêm máy chủ trò chơi để phục vụ các trận đấu mới.
- Sự kiện trong hệ thống: Khi một sự kiện đặc biệt xảy ra (như giải đấu, cập nhật lớn), hệ thống có thể mở rộng trước để đảm bảo hiệu suất.
- Một số giải pháp có sẵn có thể dùng để triển khai cho thành phần này là Kubernetes HPA kết hợp với Kubernetes Clusters, AWS Auto Scaling Group hoặc AWS Fargate Auto Scaling kết hợp với ECS Cluster.

3.1.2 Kiến trúc Serverless

Kiến trúc tổng quát ở trên có thể được cụ thể hóa tùy theo kiến trúc và công nghệ được dùng để triển khai. Đề án xây dựng hệ thống backend trò chơi dựa trên kiến trúc Serverless và các giải pháp điện toán đám mây của AWS.



Hình 3.2 Kiến trúc Serverless của hệ thống

Kiến trúc serverless của hệ thống được thiết kế nhằm tận dụng tối đa các dịch vụ do nền tảng đám mây cung cấp, giúp giảm tải công tác quản trị hạ tầng, tự động mở rộng linh hoạt theo nhu cầu và tối ưu chi phí vận hành. Trong mô hình này, các thành phần chính bao gồm quản lý người chơi, xử lý sự kiện, điều phối trò chơi, và lưu trữ dữ liệu, tất cả đều được triển khai trên các dịch vụ serverless của AWS.

Quản lý trò chơi

Quản lý trò chơi đóng vai trò trung tâm trong việc tiếp nhận và xử lý yêu cầu từ client. Để đảm bảo hiệu suất và khả năng mở rộng, API Gateway được sử dụng làm cổng giao tiếp chính, hỗ trợ cả HTTP API và WebSocket API.

- Cognito giúp xác thực người chơi và quản lý danh tính, cung cấp cơ chế đăng nhập an toàn mà không cần duy trì máy chủ xác thực riêng.
- Lambda đảm nhiệm xử lý các yêu cầu API một cách linh hoạt, chỉ kích hoạt khi có yêu cầu và tự động ngừng khi không cần thiết, giúp giảm chi phí đáng kể so với việc sử dụng máy chủ truyền thống.

Xử lý sự kiện và giao tiếp thời gian thực

Hệ thống sử dụng SNS và AppSync để đảm bảo thông tin được truyền tải một cách nhanh chóng giữa các thành phần.

- SNS hỗ trợ phân phối sự kiện theo mô hình publish-subscribe, giúp thông báo trạng thái trò chơi hoặc cập nhật hệ thống đến nhiều người chơi một cách đồng bộ.
- AppSync đóng vai trò như một GraphQL API server, cho phép các client đăng ký và nhận dữ liệu theo thời gian thực, đảm bảo trải nghiệm liền mạch mà không cần client liên tục gửi yêu cầu truy vấn dữ liệu.

Điều phối và thực thi trò chơi

Hệ thống sử dụng ECS để triển khai các Fargate Task, đảm nhiệm vai trò xử lý logic trò chơi.

- Mỗi máy chủ trò chơi được chạy trên một Fargate Task với cấu hình phù hợp để tối ưu chi phí và việc mở rộng.
- Khi không có trận đấu diễn ra trong một khoảng thời gian, hệ thống có thể tự động giảm số lượng Fargate Task về 0 để tiết kiệm tài nguyên, và chỉ kích hoạt lại khi có người chơi tham gia.
- Điều này được điều phối bởi Lambda, giúp kiểm tra trạng thái hệ thống và khởi động lại các máy chủ trò chơi khi cần.

Lưu trữ dữ liệu

Hệ thống sử dụng các dịch vụ lưu trữ serverless của AWS để tối ưu hóa hiệu suất:

- DynamoDB đóng vai trò là cơ sở dữ liệu chính, lưu trữ thông tin người chơi, trạng thái trò chơi và lịch sử trận đấu với độ trễ thấp và khả năng mở rộng tự động.
- S3 được sử dụng để lưu trữ các dữ liệu tĩnh như logs trò chơi, ảnh đại diện người chơi hoặc các bản ghi trận đấu.
- SQS giúp quản lý hàng đợi các tác vụ như ghép cặp người chơi, đảm bảo xử lý công bằng và tối ưu hiệu suất hệ thống.

3.1.3 Cơ sở dữ liệu

Hệ thống sử dụng AWS DynamoDB để lưu trữ và truy xuất dữ liệu thời gian thực một cách hiệu quả nhờ tính sẵn sàng cao, khả năng mở rộng linh hoạt và độ trễ thấp, đảm bảo hiệu suất tối ưu cho nhu cầu thời gian thực của hệ thống.

Trong DynamoDB, cơ sở dữ liệu được tổ chức thành các bảng. Đối với mỗi bảng, chỉ cần định nghĩa các thuộc tính chính, bao gồm:

- Primary Key: là bắt buộc, có thể ở dạng Partition Key đơn (HASH key) hoặc kết hợp Partition Key và Sort Key (HASH + RANGE).
- Các thuộc tính này sẽ định danh duy nhất từng bản ghi trong bảng và cũng đóng vai trò trong việc chia dữ liệu đều trên các phân vùng vật lý để tối ưu hiệu suất.

Các thuộc tính khác hoàn toàn không cần định nghĩa trước trong DynamoDB. Điều này cho phép hệ thống linh hoạt trong việc lưu trữ dữ liệu bổ sung theo nhu cầu mà không cần phải thay đổi cấu trúc bảng hay thực hiện thao tác migration phức tạp như trong các cơ sở dữ liệu quan hệ.

Nhờ đặc điểm này, hệ thống có thể:

- Tự do mở rộng schema theo thời gian thực.
- Hỗ trợ các trường hợp sử dụng động như lưu trữ metadata tùy biến.
- Tối ưu hóa hiệu suất bằng cách chỉ lưu những trường thực sự cần thiết, giảm kích thước payload.

Connections

Connections	
Id	string
UserId	string

Hình 3.3 Bảng Connections

Bảng Connections lưu ID kết nối của người chơi, dùng để gửi thông báo theo thời gian thực. Dữ liệu lưu trữ của bảng gồm:

- Id - định danh của kết nối (PK).
- UserId - định danh của người chơi.

UserRatings

UserRatings	
UserId	string

Hình 3.4 Bảng UserRatings

Bảng UserRatings lưu trữ chỉ số xếp hạng và các tham số cần thiết để tính toán xếp hạng sau ván đấu. Mỗi người dùng sẽ có một bản ghi UserRating cho một trò chơi cụ thể. Thuộc tính chính của bảng gồm:

- UserId - định danh của người chơi (PK).

Các thông tin khác về tham số xếp hạng sẽ được thêm vào tùy theo hệ thống xếp hạng cụ thể.

UserProfiles

UserProfiles	
UserId	string
Username	string
Avatar	string
Phone	string
Locale	string
CreatedAt	timestamp

Hình 3.5 Bảng UserProfiles

Bảng UserProfiles lưu trữ thông tin hồ sơ người chơi. Mỗi người dùng sẽ có một hồ sơ người chơi cho một trò chơi cụ thể. Dữ liệu lưu trữ của bảng gồm

- UserId – định danh của người chơi (PK - Partition Key).
- Username – tên hiển thị của người chơi.
- Avatar – URL ảnh đại diện.
- Phone – số điện thoại (nếu người dùng cung cấp).
- Locale – mã ngôn ngữ/địa phương (ví dụ: vi-VN, en-US)
- CreatedAt – thời điểm tạo hồ sơ người dùng.

MatchmakingTickets

MatchmakingTickets	
UserId	string
IsRanked	bool
UserRating	float
MinRating	float
MaxRating	float
GameMode	string

Hình 3.6 Bảng MatchmakingTickets

Bảng MatchmakingTickets lưu trữ yêu cầu ghép trận của người chơi. Dữ liệu lưu trữ của bảng gồm:

- UserId – định danh của người chơi gửi yêu cầu (PK – Partition Key).
- IsRanked – giá trị boolean xác định đây là trận đấu xếp hạng hay không.
- UserRating – chỉ số hiện tại của người chơi (tùy theo thuật toán xếp hạng).
- MinRating – xếp hạng tối thiểu của đối thủ có thể ghép trận.
- MaxRating – xếp hạng tối đa của đối thủ có thể ghép trận.
- GameMode – chế độ chơi mà người dùng lựa chọn.

Nếu không sử dụng xếp hạng, các trường liên quan tới xếp hạng sẽ không được thêm vào.

ActiveMatches

ActiveMatches	
MatchId	string
ConversationId	string
Players	list
GameMode	string
Server	string
StartedAt	timestamp
CreatedAt	timestamp

Hình 3.7 Bảng ActiveMatches

Bảng ActiveMatches lưu trữ các trận đấu đang diễn ra. Dữ liệu lưu trữ của bảng gồm:

- MatchId – định danh duy nhất của trận đấu (PK – Partition Key).
- ConversationId – mã phòng trò chuyện liên kết với trận đấu.
- Players – danh sách người chơi tham gia trận đấu.
- GameMode – chế độ chơi của trận đấu.
- Server – địa chỉ IP của máy chủ trò chơi.
- StartedAt – thời điểm trận đấu bắt đầu.
- CreatedAt – thời điểm tạo bản ghi.

UserMatches

UserMatches		ActiveMatches	
UserId	string	MatchId	string
MatchId	string	ConversationId	string

Hình 3.8 Bảng UserMatches

Bảng UserMatches thể hiện liên kết giữa người chơi với một trận đấu đang diễn ra. Mỗi quan hệ 1-1 với bảng ActiveMatches thể hiện người chơi chỉ có thể tham gia một trận đấu tại một thời điểm nhất định. Dữ liệu lưu trữ của bảng gồm:

- UserId - định danh của người chơi (PK).
- MatchId - định danh của trận đấu.

MatchStates

MatchStates	
Id	string
MatchId	string
PlayerStates	json
GameState	json
Move	json
Timestamp	timestamp

Hình 3.9 Bảng MatchStates

Bảng MatchStates lưu trữ trạng thái của trận đấu để phục vụ đồng bộ dữ liệu với người xem hoặc khôi phục dữ liệu khi người chơi mất kết nối hoặc máy chủ xảy ra lỗi. Dữ liệu lưu trữ của bảng gồm:

- Id – định danh duy nhất của bản ghi trạng thái (PK – Partition Key).
- MatchId – định danh của trận đấu liên quan đến trạng thái này.

- PlayerStates – trạng thái chi tiết của những người chơi.
- GameState – trạng thái trò chơi.
- Move – đầu vào trò chơi.
- Timestamp – thời điểm bản ghi trạng thái này được ghi nhận.

MatchRecords

MatchRecords	
MatchId	string
PlayerRecords	json
GameRecord	json
StartedAt	timestamp
EndedAt	timestamp

Hình 3.10 Bảng MatchRecords

Bảng MatchRecords lưu trữ thông tin chi tiết của trận đấu. Dữ liệu lưu trữ của bảng bao gồm:

- MatchId - định danh của trận đấu (PK).
- PlayerRecords - thông tin của người chơi.
- GameRecord - dữ liệu trận đấu.
- StartedAt - thời gian trận đấu bắt đầu.
- EndedAt - thời gian trận đấu kết thúc.

MatchResults

MatchResults	
UserId	string
Timestamp	timestamp

Hình 3.11 Bảng MatchResults

Bảng MatchResults lưu trữ kết quả trận đấu của người chơi trong các trò chơi để phục vụ tính toán xếp hạng. Thuộc tính chính của bảng bao gồm:

- UserId - định danh của người chơi (PK - Primary Key).
- Timestamp - thời gian trận đấu kết thúc (SK- Sort Key).

Những thông tin khác sẽ được thêm vào tùy theo hệ thống xếp hạng cụ thể

Messages

Messages	
Id ↗	string
ConversationId	string
SenderId	string
Content	string
CreatedAt	string

Hình 3.12 Bảng Messages

Bảng Messages lưu trữ những tin nhắn được gửi trong hệ thống trò chuyện. Dữ liệu lưu trữ của bảng bao gồm:

- Id - định danh của tin nhắn (PK).
- ConversationId - định danh của hội thoại (Index).
- SenderId - định danh của người gửi
- Content - nội dung tin nhắn
- CreatedAt - thời gian tin nhắn được tạo

SpectatorConversations

SpectatorConversations	
MatchId ↗	string
ConversationId ↗	string

Hình 3.13 Bảng SpectatorConversations

Bảng SpectatorConversations lưu trữ hội thoại giữa những người xem trận đấu với nhau. Dữ liệu lưu trữ của bảng bao gồm:

- MatchId - định danh của tin nhắn (PK).
- ConversationId - định danh của hội thoại.

Friendships

Friendships	
UserId	string
FriendId	string
ConversationId	string
StartedAt	timestamp

Hình 3.14 Bảng Friendships

Bảng Friendships lưu trữ thông tin kết bạn với người chơi khác. Dữ liệu lưu trữ của bảng bao gồm:

- UserId - định danh của người dùng (PK).
- FriendId - định danh người dùng của bạn bè (PK)
- ConversationId - định danh của hội thoại giữa hai người dùng.
- Content - nội dung tin nhắn
- CreatedAt - thời gian tin nhắn được tạo

FriendRequests

FriendRequests	
SenderId	string
ReceiverId	string
CreatedAt	timestamp

Hình 3.15 Bảng FriendRequests

Bảng FriendRequests lưu trữ yêu cầu kết bạn với người chơi khác. Dữ liệu lưu trữ của bảng bao gồm:

- SenderId - định danh của người gửi (PK).
- ReceiverId - định danh của người nhận (PK).
- CreatedAt - thời gian tạo yêu cầu.

3.1.4 Bảo mật hệ thống

Bảo mật hệ thống đóng vai trò quan trọng hàng đầu trong kiến trúc triển khai backend trò chơi, đặc biệt khi sử dụng kiến trúc serverless và các dịch vụ điện toán đám mây.

mây. Giải pháp bảo mật của hệ thống được thiết kế dựa trên mô hình phòng thủ nhiều lớp, đảm bảo an toàn từ mức hạ tầng cho đến tầng ứng dụng và dữ liệu.

Ở tầng hạ tầng, hệ thống sử dụng các dịch vụ quản lý định danh và truy cập của AWS, cụ thể là AWS Cognito. Dịch vụ này cung cấp khả năng xác thực người dùng thông qua các phương thức như đăng nhập bằng tên người dùng và mật khẩu, đăng nhập bằng bên thứ ba (Google, Facebook) và các phương thức xác thực MFA. Việc ứng dụng AWS Cognito giúp giảm thiểu rủi ro từ các cuộc tấn công dựa trên định danh.

Đối với bảo mật ứng dụng, các dịch vụ như AWS API Gateway và AWS AppSync được cấu hình để kiểm soát và quản lý truy cập API thông qua các chính sách IAM. Các chính sách này cho phép chỉ những người dùng hoặc dịch vụ được ủy quyền mới có thể gọi API, hạn chế tối đa các truy cập trái phép.

Bảo mật dữ liệu là một trong những ưu tiên cao nhất của hệ thống. Tất cả dữ liệu lưu trữ trong DynamoDB đều được mã hóa ở trạng thái nghỉ thông qua AWS Key Management Service. Đồng thời, dữ liệu trong quá trình truyền đi cũng được mã hóa thông qua các giao thức bảo mật như HTTPS và TLS. Hệ thống cũng áp dụng các biện pháp phòng chống tấn công SQL injection và các hình thức tấn công dựa vào dữ liệu khác.

Giám sát và cảnh báo bảo mật là thành phần không thể thiếu, được triển khai thông qua AWS CloudWatch và AWS CloudTrail. Các dịch vụ này giúp hệ thống thu thập, phân tích và lưu trữ nhật ký hoạt động của người dùng và ứng dụng, từ đó phát hiện sớm và ứng phó nhanh chóng với các sự cố bảo mật.

Việc tích hợp các phương pháp bảo mật này tạo nên một hệ thống an toàn, đáng tin cậy, đáp ứng yêu cầu khắt khe của môi trường triển khai trò chơi trực tuyến hiện đại.

3.2 Máy chủ trò chơi

3.2.1 Phát triển máy chủ

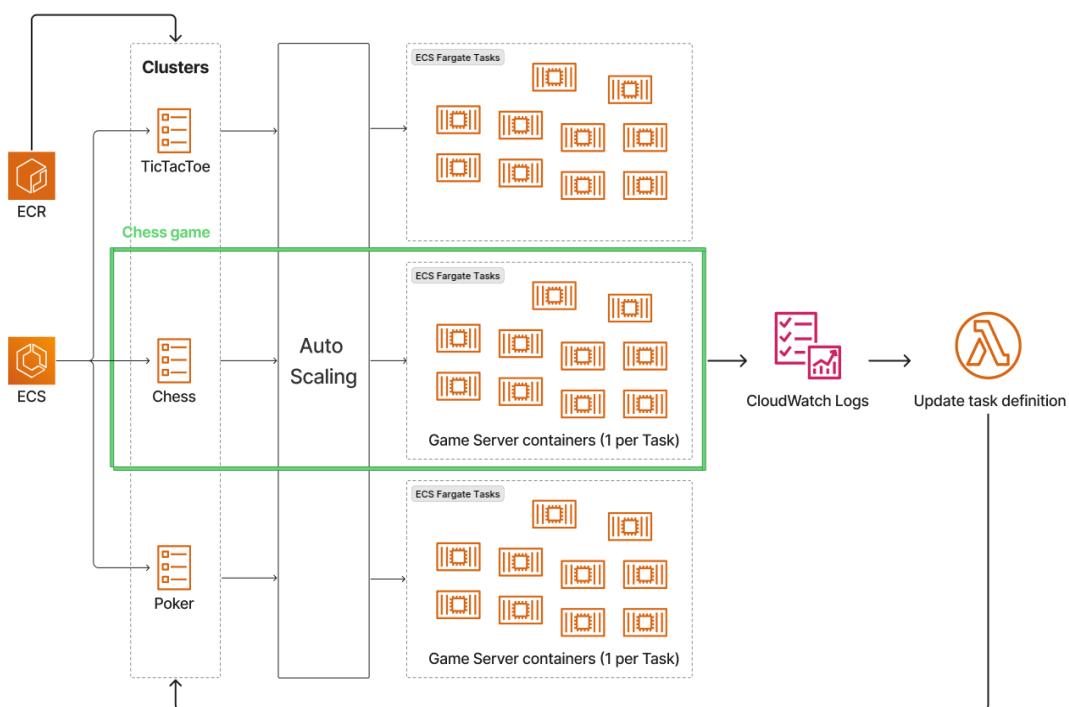
Máy chủ trò chơi là thành phần trung tâm, thực hiện nhiều chức năng cốt lõi trong toàn bộ hệ thống:

- Quản lý phiên chơi: Mỗi phiên chơi tương ứng với một trận đấu hoặc phòng chơi giữa hai hoặc nhiều người chơi. Máy chủ khởi tạo, theo dõi và huỷ các phiên chơi dựa trên trạng thái trò chơi và hành vi của người dùng.

- Quản lý người chơi: Lưu trữ trạng thái phiên của từng người chơi (đang trong trận, thoát, kết nối lại, bị loại, v.v.), và đảm bảo tính nhất quán dữ liệu xuyên suốt phiên chơi.
- Xử lý logic trò chơi: Mọi logic liên quan đến quy tắc của trò chơi đều được xử lý trên máy chủ để đảm bảo tính trung lập và công bằng.
- Giao tiếp thời gian thực: Máy chủ trò chơi cung cấp khả năng giao tiếp thời gian thực giữa các người chơi, thông qua các giao thức như WebSocket hoặc WebRTC.
- Đồng bộ trạng thái và lưu trữ: Đảm bảo đồng bộ hóa trạng thái trò chơi giữa các client và backend. Máy chủ cũng sử dụng các dịch vụ lưu trữ để lưu trạng thái phiên chơi nhằm hỗ trợ khả năng khôi phục khi người chơi bị mất kết nối.

Để tăng tính mô đun hóa và tái sử dụng, logic giao tiếp thời gian thực, quản lý phiên chơi và người chơi, đồng bộ trạng thái và lưu trữ đã được đóng gói thành một bộ kit phát triển phần mềm. Người phát triển trò chơi do đó chỉ cần tập trung vào logic của trò chơi mà không cần quan tâm đến logic kết nối các thành phần bên dưới.

3.2.2 Mô hình triển khai



Hình 3.16 Mô hình triển khai ECS Fargate

Hình 3.16 mô tả mô hình triển khai máy chủ trò chơi sử dụng Amazon ECS Fargate, với một điểm khác biệt quan trọng là mỗi backend trò chơi sẽ được cấp phát

một ECS Cluster riêng. Cách tiếp cận này cho phép quản lý tài nguyên một cách độc lập và linh hoạt giữa các backend, đồng thời nâng cao khả năng kiểm soát, giám sát và tối ưu hóa hiệu suất cho từng nền tảng trò chơi cụ thể.

Trong hệ thống này, mỗi trò chơi được triển khai dưới dạng một ECS Service trong ECS Cluster riêng biệt, đi kèm với một Task Definition riêng. Việc phân tách ECS Cluster theo từng backend giúp đảm bảo khả năng mở rộng đa trò chơi, đồng thời tránh xung đột tài nguyên giữa các dịch vụ trò chơi khác nhau.

Quá trình triển khai máy chủ trò chơi tuân theo mô hình container hóa, trong đó image của container được đóng gói và lưu trữ trên Amazon ECR. Khi cần khởi chạy một trò chơi, ECS sẽ tự động kéo image từ ECR và khởi tạo một Fargate Task trong cluster tương ứng, chia container chạy máy chủ trò chơi.

Cấu hình của Fargate Task có thể được điều chỉnh linh hoạt theo yêu cầu của từng trò chơi. Ví dụ:

- Đối với các trò chơi không yêu cầu xử lý thời gian thực (như board game, puzzle, chiến thuật), hệ thống có thể sử dụng cấu hình tài nguyên thấp để tối ưu hóa chi phí.
- Với các trò chơi yêu cầu cao về tài nguyên và thời gian phản hồi (như MOBA, FPS, MMORPG), hệ thống sẽ cấp phát nhiều CPU và bộ nhớ hơn để đảm bảo hiệu suất và trải nghiệm người dùng.

Đặc biệt, hệ thống có khả năng tự động điều chỉnh tài nguyên theo lưu lượng người dùng. Trong các khung giờ cao điểm, thay vì chỉ mở rộng quy mô bằng cách tăng số lượng task nhỏ, hệ thống có thể chủ động cập nhật Task Definition với cấu hình cao hơn, khởi chạy ít task nhưng mạnh hơn để giảm độ trễ mở rộng và tối ưu hiệu suất tổng thể.

Ngoài ra, hệ thống có thể tích hợp mô hình học máy để phân tích lịch sử lưu lượng người chơi, từ đó dự báo nhu cầu tài nguyên. Dựa trên các dự báo này, hệ thống sẽ tự động điều chỉnh cấu hình ECS Service và Fargate Task phù hợp với từng backend, đảm bảo cân bằng giữa hiệu suất và chi phí vận hành.

3.2.3 Chiến lược mở rộng tự động

Chiến lược mở rộng tự động đóng vai trò then chốt trong việc đảm bảo hiệu suất và khả năng phục vụ liên tục của hệ thống backend trò chơi. Đặc biệt, với các trò chơi trực tuyến, việc người chơi đồng thời tham gia tăng đột biến là thường xuyên xảy ra, do đó hệ thống cần một chiến lược mở rộng nhanh, linh hoạt và hiệu quả.

Trong đồ án này, chiến lược mở rộng tự động được xây dựng dựa trên việc theo dõi và giám sát chặt chẽ ba chỉ số quan trọng là độ sử dụng CPU, bộ nhớ, và số trận đấu đồng thời trên mỗi máy chủ. Việc thiết lập các ngưỡng này được xác định dựa trên kết quả thử nghiệm thực tế để đảm bảo sự cân bằng giữa hiệu suất tối ưu và chi phí vận hành hợp lý.

Bảng dưới đây tổng hợp rõ ràng các thông số quan trọng được sử dụng trong việc ra quyết định mở rộng quy mô máy chủ trò chơi:

Bảng 3.1 Tham số trong chiến lược mở rộng tự động

Thông số theo dõi	Nguadroing mở rộng (Threshold)	Thời gian Cooldown	Hành động mở rộng
CPU trung bình của ECS Fargate task	$\geq 70\%$ duy trì liên tục trong 2 phút	2 phút	Tăng thêm 1 Fargate Task
Bộ nhớ (RAM) của ECS Fargate task	$\geq 80\%$ duy trì liên tục trong 2 phút	2 phút	Tăng thêm 1 Fargate Task
Số lượng trận đấu đồng thời trên mỗi máy chủ	≥ 100 trận/server (có thể tùy chỉnh)	30 giây	Tăng thêm 1 Fargate Task

Các thông số trên được lựa chọn dựa trên thử nghiệm thực tế cũng như các khuyến nghị từ AWS trong tài liệu hướng dẫn về triển khai auto-scaling trên ECS Fargate [3]. Cụ thể như sau:

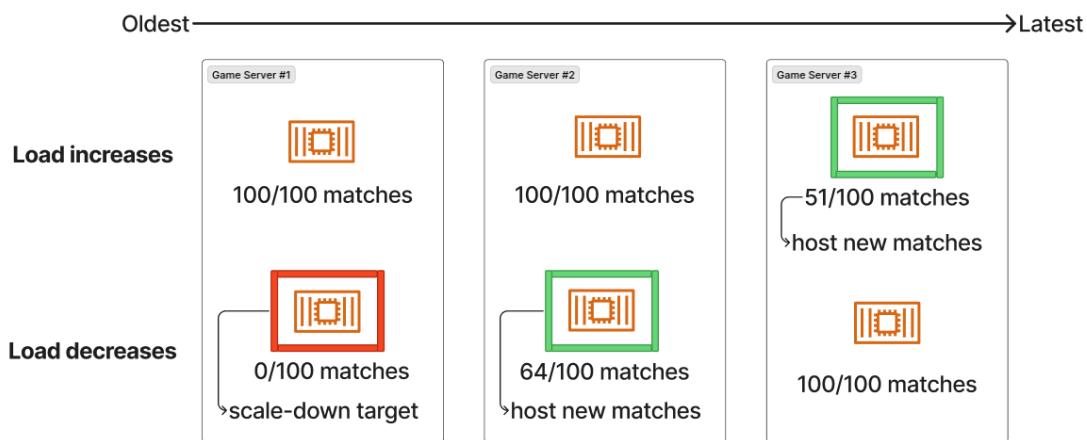
- Khi CPU trung bình vượt ngưỡng 70% hoặc bộ nhớ RAM vượt ngưỡng 80% liên tục trong 2 phút, hệ thống sẽ chủ động thêm ngay một task Fargate mới vào ECS Cluster để cân bằng lại tải. Lựa chọn này xuất phát từ kinh nghiệm thực tế, nhằm đảm bảo hệ thống phản ứng nhanh và phù hợp với mức tải ổn định thực tế, tránh tình trạng phản ứng thái quá với các biến động tải ngắn hạn. Đồng thời, nó cũng đảm bảo trải nghiệm người chơi luôn ổn định và không bị gián đoạn bởi vấn đề hiệu suất.
- Số lượng trận đấu đồng thời trên mỗi máy chủ được giới hạn nhằm đảm bảo độ ổn định cao nhất cho từng trận đấu. Khi số lượng trận đấu trên mỗi máy chủ đạt tới ngưỡng giới hạn (mặc định là 100 trận, tuy nhiên có thể tùy chỉnh linh hoạt bởi người dùng giải pháp), một máy chủ mới sẽ được kích hoạt ngay lập tức trong vòng 30 giây. Ngưỡng này được xác định từ các kết quả benchmark thực tế, thể

hiện điểm tối ưu để cân bằng giữa hiệu năng xử lý và trải nghiệm mượt mà của người chơi. Việc thiết lập cooldown ngắn (30 giây) cho ngưỡng này là cần thiết vì đây là thông số trực tiếp và rõ ràng nhất phản ánh nhu cầu mở rộng cấp thiết.

Việc thiết lập thời gian cooldown là một giải pháp cần thiết và hiệu quả. Với cooldown 2 phút cho chỉ số CPU và bộ nhớ, hệ thống tránh được tình trạng mở rộng quá thường xuyên dẫn đến dư thừa tài nguyên. Tuy nhiên, cooldown cho số lượng trận đấu đồng thời được thiết lập ngắn hơn (30 giây), vì số lượng trận đấu tăng cao là dấu hiệu rõ ràng nhất thể hiện việc cần mở rộng ngay lập tức để duy trì chất lượng trải nghiệm người chơi.

Ngoài chiến lược mở rộng dựa trên tài nguyên như CPU, RAM hay số lượng trận đấu đồng thời, hệ thống còn sử dụng chiến lược phân phối trận đấu ưu tiên vào máy chủ mới được khởi tạo để tối ưu quy trình thu hẹp (scale-in) và vận hành tài nguyên hiệu quả.

Cụ thể, khi hệ thống bắt đầu ghép trận, các trận mới sẽ được ưu tiên phân phối vào các máy chủ mới nhất (vừa được tạo ra do mở rộng). Trong khi đó, các máy chủ cũ hơn dần ngừng nhận thêm trận mới, chỉ duy trì để hoàn tất các trận đang diễn ra. Khi không còn trận đấu nào đang xử lý, các máy chủ cũ này sẽ tự động được thu hẹp để tiết kiệm tài nguyên.



Hình 3.17 Chiến lược auto-scaling dựa trên phân phối trận đấu

Hình 3.17 minh họa quá trình này theo trực thời gian từ trái qua phải:

- Khi tải tăng, một máy chủ mới (Game Server #3) được khởi tạo để xử lý các trận mới. Các máy chủ cũ (Game Server #1 và #2) vẫn tiếp tục xử lý các trận cũ nhưng không nhận thêm.

- Dần dần, các trận đấu hoàn thành và số lượng trận đấu trong các máy chủ cũ giảm xuống.
- Khi tải giảm mạnh, máy chủ cũ có thể không còn trận đấu nào, trở thành mục tiêu ưu tiên thu hẹp (scale-down target).

Chiến lược phân phối này đảm bảo hệ thống:

- Tối ưu hóa chi phí vận hành, tránh duy trì máy chủ không cần thiết.
- Duy trì hiệu suất ổn định, vì máy chủ mới thường có ít tải và độ phản hồi tốt hơn.
- Tự động thu hẹp một cách có kiểm soát, không ảnh hưởng đến các trận đấu đang diễn ra.

Cơ chế này đóng vai trò hỗ trợ quan trọng cho chiến lược mở rộng và thu hẹp tài nguyên tự động, giúp hệ thống game backend duy trì độ tin cậy, phản hồi linh hoạt và đạt hiệu quả chi phí cao — vốn là mục tiêu cốt lõi của giải pháp PaaS dựa trên kiến trúc serverless được đề xuất trong đồ án.

3.3 Các hệ thống con trong backend trò chơi

Để đáp ứng đầy đủ các yêu cầu chức năng đã đặt ra, hệ thống backend trò chơi được thiết kế gồm nhiều thành phần nhỏ, mỗi thành phần đảm nhiệm một chức năng cụ thể và có thể hoạt động độc lập. Các hệ thống con được tổ chức theo hướng module hóa, giúp dễ dàng triển khai, bảo trì và tái sử dụng trong nhiều trò chơi khác nhau. Dưới đây là các hệ thống con chính:

Hệ thống xác thực người dùng

Hệ thống xác thực là thành phần đầu tiên và bắt buộc trong toàn bộ chuỗi tương tác với backend trò chơi. Việc xác thực người dùng được tích hợp hoàn toàn với dịch vụ Amazon Cognito, nơi quản lý toàn bộ quá trình đăng nhập, lưu trữ thông tin định danh, và cấp phát mã thông báo JWT. Mỗi yêu cầu từ client gửi tới hệ thống backend đều được kiểm tra token để đảm bảo người dùng đã được xác thực hợp lệ. Ngoài ra, các thông tin định danh như user ID, email, hoặc vai trò (nếu có) sẽ được trích xuất từ token và sử dụng trong suốt phiên chơi. Bằng việc sử dụng Cognito, hệ thống đảm bảo tính bảo mật, khả năng mở rộng, và giảm thiểu chi phí vận hành các thành phần xác thực thủ công.

Hệ thống xếp hạng người chơi

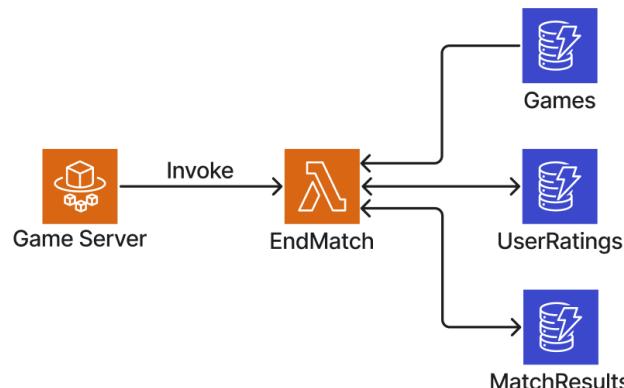
Hệ thống xếp hạng sử dụng các thuật toán như Glicko hoặc TrueSkill để tính toán điểm số của người chơi dựa trên kết quả các trận đấu. Đây là thành phần quan trọng nhằm đánh giá trình độ và giúp cân bằng trải nghiệm chơi game. Hệ thống xếp hạng

trong nền tảng được thiết kế dưới dạng dịch vụ độc lập, có thể kích hoạt hoặc vô hiệu tùy theo nhu cầu của từng trò chơi.

Các yếu tố có trong hệ thống xếp hạng bao gồm:

- Điểm xếp hạng: Mỗi người chơi có một điểm số phản ánh kỹ năng hiện tại. Điểm số này được cập nhật sau mỗi trận đấu để phản ánh sự tiến bộ hoặc tụt lùi.
- Tham số xếp hạng: Một tập các tham số đặc trưng ảnh hưởng đến cách tính điểm của hệ thống.
- Tùy biến theo trò chơi: Các nhà phát triển có thể lựa chọn và cấu hình thuật toán xếp hạng cho phù hợp với thể loại và cơ chế trò chơi.

Ngoài ra, hệ thống cũng hỗ trợ tích hợp với hệ thống ghép trận, giúp phân phối người chơi vào các trận đấu một cách công bằng. Việc sử dụng hệ thống xếp hạng là hoàn toàn tùy chọn. Các trò chơi giải trí, đơn giản, hoặc không mang tính đối kháng có thể bỏ qua dịch vụ này để tối ưu tài nguyên và đơn giản hóa triển khai.



Hình 3.18 Luồng hoạt động của hệ thống xếp hạng

Hình 3.18 mô tả luồng hoạt động của hệ thống xếp hạng như sau

1. Trận đấu ở trạng thái kết thúc, máy chủ trò chơi kích hoạt hàm xử lý kết thúc trận đấu
2. Hệ thống truy vấn thông tin về trò chơi để biết kiểu hệ thống xếp hạng được tích hợp với trò chơi.
3. Nếu trò chơi không tích hợp hệ thống xếp hạng, luồng dừng ở đây

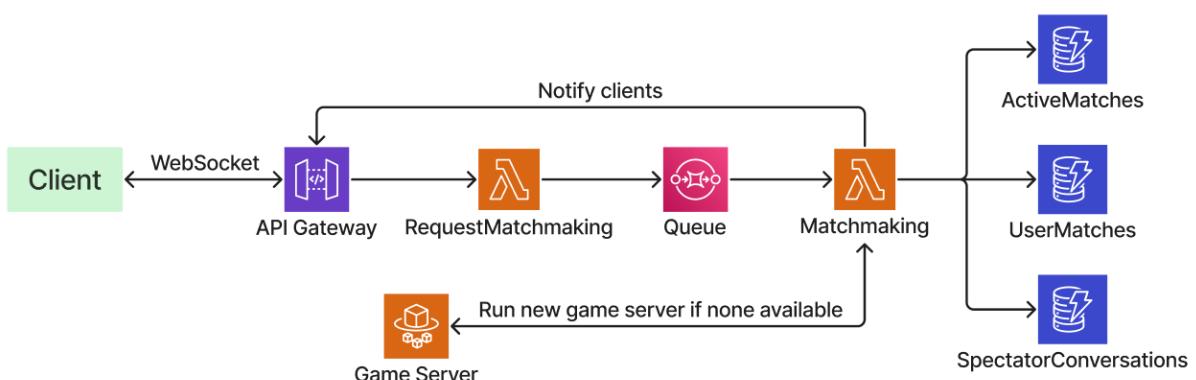
Nếu trò chơi có tích hợp hệ thống, hệ thống truy vấn xếp hạng hiện tại của người chơi và các tham số cần thiết để tính toán xếp hạng mới.

4. Hệ thống lưu kết quả trận đấu, xếp hạng người chơi và các tham số tính toán mới vào cơ sở dữ liệu.

Hệ thống ghép trận

Hệ thống ghép trận giúp tìm người và người chơi lại với nhau trong các trận đấu. Người phát triển trò chơi có thể tự định nghĩa các thuộc tính liên quan tới trò chơi để ghép trận. Các trò chơi có yếu tố cạnh tranh có thể sử dụng cùng với hệ thống xếp hạng để tìm các đối thủ có kỹ năng tương đồng, giúp đảm bảo công bằng và tăng độ kích thích khi chơi.

Hệ thống cũng đảm bảo trận đấu được tạo ra trong một khoảng thời gian hợp lý để người chơi đợi quá lâu. Khi cùng thỏa mãn tiêu chí ghép trận, người chơi gửi yêu cầu trước sẽ được ghép trận trước.



Hình 3.19 Luồng hoạt động của hệ thống ghép trận

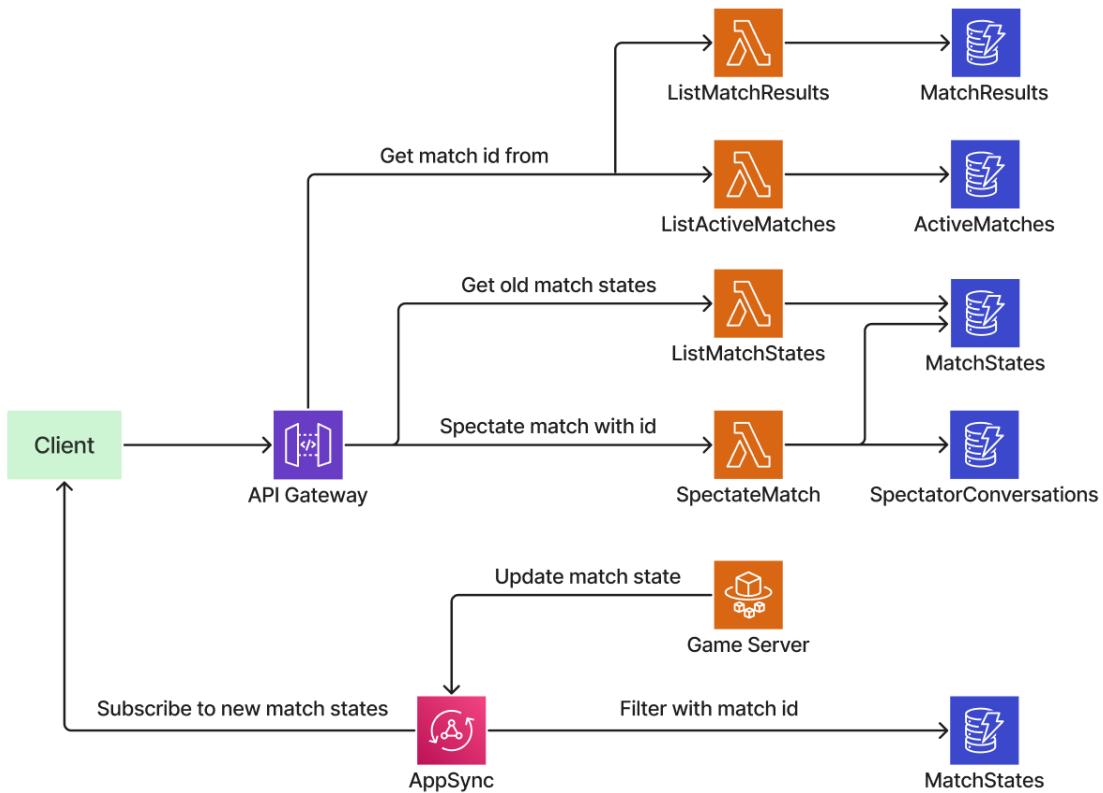
Hình 3.19 mô tả luồng hoạt động của hệ thống ghép trận như sau

1. Người chơi kết nối tới backend và gửi yêu cầu ghép trận
2. Hệ thống kiểm tra thông tin người chơi, xác nhận và đưa yêu cầu vào hàng đợi.
3. Khi tìm được các đối thủ phù hợp, hệ thống:
 - Gửi thông báo đến tất cả người chơi.
 - Chạy máy chủ trò chơi mới nếu không có máy chủ nào khả dụng.
4. Người chơi kết nối đến máy chủ trò chơi để bắt đầu trận đấu.

Hệ thống theo dõi trực tiếp trận đấu

Hệ thống theo dõi trực tiếp trận đấu người xem theo dõi diễn biến trận đấu theo thời gian thực. Đây là một tính năng hữu ích cho những người chơi muốn xem các trận đấu của người khác hoặc muốn cải thiện kỹ năng của mình thông qua việc quan sát các trận đấu chuyên nghiệp. Các yếu tố trong hệ thống này bao gồm:

- Trực tiếp trận đấu: Người xem có thể theo dõi trận đấu giữa những người chơi khác mà không cần tham gia.
- Chế độ phát lại: Cung cấp các tùy chọn phát lại các trận đấu đã diễn ra để người chơi học hỏi hoặc giải trí.
- Hiển thị thông tin chi tiết: Các số liệu thống kê về trận đấu (số lần di chuyển, điểm số, hiệu suất của người chơi...) được cập nhật và hiển thị trực tiếp.



Hình 3.20 Luồng hoạt động của hệ thống theo dõi trận đấu

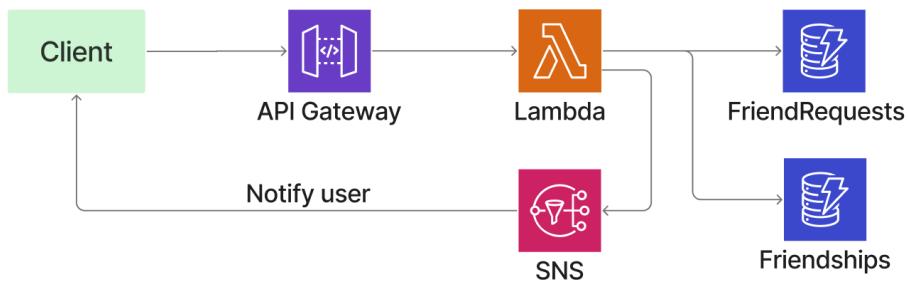
Hình 3.20 mô tả luồng hoạt động của hệ thống theo dõi trận đấu như sau

1. Lấy định danh trận đấu từ các luồng khác của hệ thống
2. Theo dõi trận đấu
 - Khi đã có định danh của trận đấu, Client gửi yêu cầu theo dõi trận đấu
 - Hệ thống trả lại lịch sử trạng thái gần nhất của hệ thống. Client có thể truy vấn thêm nếu cần thiết.
3. Client đăng ký nhận cập nhật trạng thái trận đấu mới thông qua AWS AppSync.
4. Khi có thay đổi trong trận đấu, máy chủ trò chơi gửi cập nhật trạng thái. AppSync lọc trạng thái mới dựa trên định danh trận đấu và gửi thông báo cập nhật đến Client.

Hệ thống bạn bè

Hệ thống bạn bè giúp người chơi kết nối và tạo mối quan hệ xã hội trong nền tảng trò chơi trực tuyến. Người chơi có thể kết bạn, trò chuyện và tham gia cùng nhau trong các trận đấu. Các tính năng trong hệ thống bạn bè bao gồm:

- Gửi lời mời kết bạn: Người chơi có thể gửi và nhận lời mời kết bạn từ các người chơi khác.
- Danh sách bạn bè: Người chơi có thể xem danh sách bạn bè và dễ dàng mời họ tham gia vào trận đấu.
- Chế độ chơi cùng bạn bè: Cho phép bạn chơi cùng bạn bè trong các chế độ chơi đặc biệt.
- Tình trạng trực tuyến: Người chơi có thể biết được ai đang trực tuyến và sẵn sàng tham gia chơi cùng họ.



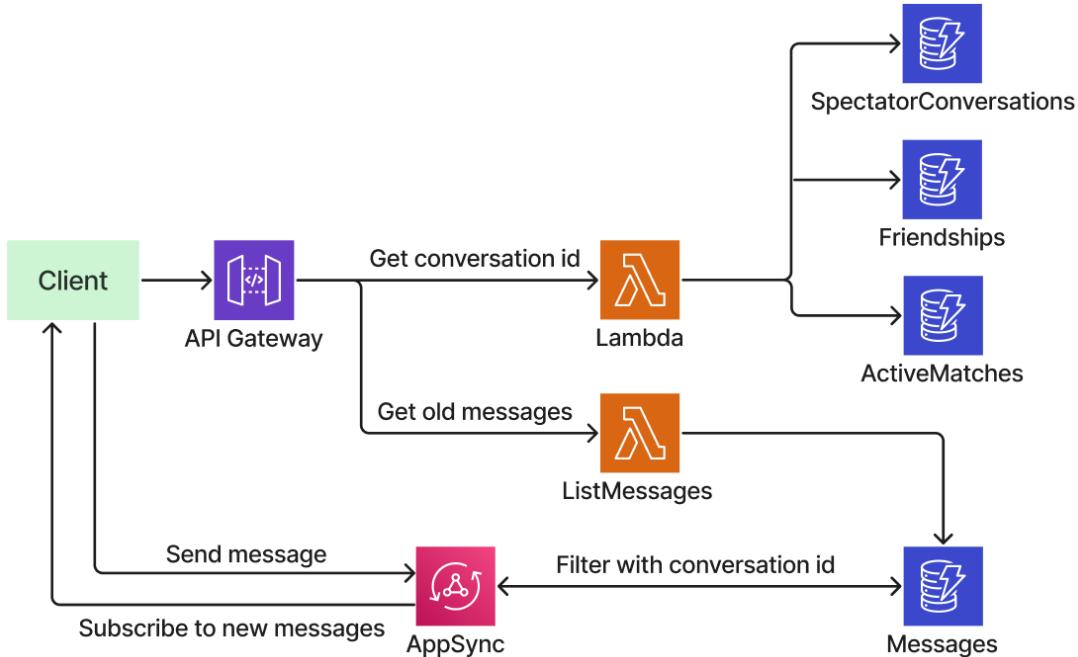
Hình 3.21 Luồng hoạt động của hệ thống bạn bè

Hình 3.21 mô tả luồng hoạt động của hệ thống bạn bè như sau

1. Người dùng gửi yêu cầu kết bạn. Hệ thống cập nhật cơ sở dữ liệu và báo cho người nhận thông qua AWS SNS.
2. Người nhận lựa chọn đồng ý hoặc từ chối kết bạn
 - Nếu từ chối, yêu cầu kết bạn được xóa khỏi hệ thống
 - Nếu đồng ý, hệ thống tạo bản ghi về thông tin kết bạn giữa hai người dùng.
3. Người dùng sau đó có thể xóa bạn bè nếu muốn.

Hệ thống trò chuyện

Hệ thống trò chuyện cho phép người chơi nhắn tin riêng với nhau và nhắn tin vào nhóm chung trong trận đấu hoặc khi cùng xem trực tiếp một trận đấu đang diễn ra. Để đảm bảo trải nghiệm tích cực của người dùng, hệ thống sử dụng bộ lọc từ ngữ chống lại ngôn từ thô tục hoặc vi phạm nội quy.



Hình 3.22 Luồng hoạt động của hệ thống trò chuyện

Hình 3.22 mô tả luồng hoạt động của hệ thống diễn ra như sau:

1. Người dùng lấy định danh của cuộc trò chuyện thông qua các luồng hoạt động khác.
2. Người dùng truy vấn lịch sử tin nhắn trong cuộc trò chuyện nếu có.
3. Người dùng đăng ký nhận tin nhắn mới và gửi tin nhắn tới cuộc trò chuyện thông qua AppSync

Các hệ thống con đã được trình bày ở trên – bao gồm hệ thống xác thực, xếp hạng, ghép trận, theo dõi trận đấu, bạn bè và trò chuyện – đều được thiết kế như các dịch vụ độc lập, có thể hoạt động riêng lẻ hoặc tích hợp theo yêu cầu. Nhờ cấu trúc module hóa, mỗi hệ thống có thể được đóng gói và cung cấp dưới dạng dịch vụ tích hợp sẵn, cho phép các nhà phát triển trò chơi tích hợp trực tiếp thông qua các API tiêu chuẩn mà không cần triển khai lại toàn bộ backend. Thiết kế này không chỉ nâng cao tính linh hoạt trong phát triển game mà còn giảm đáng kể chi phí vận hành và thời gian đưa sản phẩm ra thị trường.

Chương 4: Giải pháp hỗ trợ triển khai backend trò chơi

4.1 Mô hình giải pháp

4.1.1 Tổng quan mô hình

Giải pháp được phát triển trong khuôn khổ đề tài là một nền tảng triển khai backend trò chơi hiện đại, xây dựng dựa trên kiến trúc serverless và tích hợp sâu với hệ sinh thái dịch vụ AWS. Nền tảng này hướng đến việc chuẩn hóa và tự động hóa quy trình triển khai backend cho trò chơi, từ đó giúp tiết kiệm thời gian phát triển, tối ưu chi phí vận hành và đảm bảo khả năng mở rộng linh hoạt.

Giải pháp được thiết kế theo mô hình PaaS chuyên biệt cho backend trò chơi, cho phép nhà phát triển triển khai các máy chủ trò chơi tùy chỉnh dưới dạng container thông qua dịch vụ ECS Fargate. Bên cạnh đó, nền tảng còn cung cấp sẵn các dịch vụ backend tích hợp như xác thực người dùng, ghép trận, xếp hạng, trò chuyện và xem trận đấu trực tiếp. Các dịch vụ này được thiết kế để cấu hình linh hoạt, dễ tích hợp vào phía client hoặc game engine, đồng thời đảm bảo tính thống nhất và hiệu quả trong vận hành.

Khác với các nền tảng đóng gói toàn bộ dịch vụ trong dạng SDK cố định, giải pháp cho phép nhà phát triển toàn quyền kiểm soát kiến trúc triển khai thông qua các công cụ hạ tầng như AWS Serverless Application Model (SAM) hoặc CloudFormation. Điều này giúp kết hợp giữa sự chủ động trong việc thiết kế hệ thống với khả năng tái sử dụng các thành phần dịch vụ sẵn, từ đó tối ưu thời gian phát triển và khả năng mở rộng.

Cách tiếp cận này mang lại sự cân bằng giữa tự do kiến trúc và tiện ích tích hợp, phù hợp với nhiều đối tượng sử dụng — từ nhà phát triển độc lập, startup cho đến các studio trò chơi chuyên nghiệp. Nền tảng không chỉ hỗ trợ triển khai backend trò chơi một cách nhanh chóng, mà còn đảm bảo tính ổn định, hiệu năng cao và khả năng mở rộng linh hoạt trong môi trường cloud-native hiện đại.

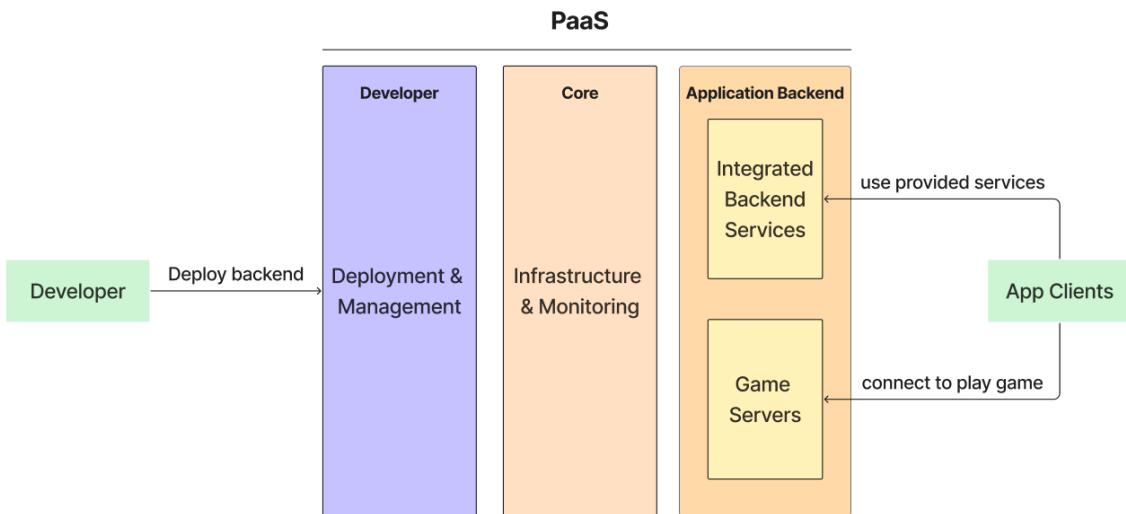
4.1.2 Mục tiêu của mô hình

Mục tiêu chính của mô hình là đơn giản hóa quy trình phát triển backend trò chơi, đồng thời cung cấp một nền tảng đủ linh hoạt để đáp ứng cả nhu cầu triển khai nhanh chóng. Cụ thể, mô hình hướng tới việc:

- Tự động hóa toàn bộ quá trình triển khai backend trò chơi từ cấu hình đến vận hành, giảm thiểu công việc thủ công.
- Cho phép tái sử dụng các dịch vụ backend phổ biến giữa nhiều trò chơi nhằm tối ưu chi phí và tài nguyên.

- Hỗ trợ cả hình thức tích hợp backend đơn giản (qua API) và hình thức triển khai backend đầy đủ theo yêu cầu riêng.
- Đảm bảo khả năng mở rộng linh hoạt, hiệu suất cao, và chi phí thấp thông qua kiến trúc serverless.

4.1.3 Kiến trúc tổng thể



Hình 4.1 Kiến trúc giải pháp PaaS

Giải pháp được thiết kế dựa trên một kiến trúc phân tầng rõ ràng, bao gồm ba tầng chính: Developer Layer, PaaS Core, và Application Backend Layer. Cách phân chia này giúp phân tách vai trò, giảm độ phức tạp trong triển khai, đồng thời tăng tính tái sử dụng và khả năng mở rộng theo từng thành phần.

Tầng Developer là nơi nhà phát triển tương tác với hệ thống để triển khai, cấu hình và giám sát backend trò chơi. Thông qua giao diện cấu hình hoặc SDK, các nhà phát triển có thể gửi yêu cầu triển khai backend, lựa chọn dịch vụ phù hợp và khởi tạo hệ thống backend mới. Quá trình triển khai được tự động hóa thông qua các công cụ như AWS SAM và CloudFormation, kết hợp với pipeline CI/CD sử dụng AWS Batch hoặc Lambda. Điều này giúp đảm bảo sự thống nhất trong quá trình phát hành backend mới, giảm thiểu lỗi thủ công và rút ngắn thời gian đưa vào vận hành.

Tầng PaaS Core đóng vai trò là lớp điều phối hạ tầng trung tâm của nền tảng. Tại tầng này, hệ thống thực hiện các chức năng giám sát trạng thái hoạt động, ghi nhận log, xử lý sự kiện và quản lý quá trình mở rộng hoặc thu hẹp tài nguyên dựa trên tải thực tế. Các cơ chế tự động tại tầng này đảm bảo rằng tài nguyên được phân phối linh hoạt theo nhu cầu sử dụng, đồng thời hỗ trợ phản ứng kịp thời với các thay đổi trong quá trình vận hành. Việc tách biệt lớp điều phối hạ tầng ra khỏi tầng thực thi giúp nâng cao tính ổn

định và khả năng bảo trì của toàn bộ hệ thống, đồng thời tạo điều kiện thuận lợi cho việc mở rộng theo chiều ngang một cách hiệu quả.

Việc tách riêng lớp điều phối hạ tầng khỏi logic nghiệp vụ giúp đảm bảo tính ổn định của nền tảng, đồng thời hỗ trợ khả năng mở rộng theo chiều ngang mà không ảnh hưởng đến quá trình xử lý phía người dùng.

Tầng Application Backend là nơi tập trung toàn bộ các thành phần backend thực thi trực tiếp cho người dùng cuối. Tầng này bao gồm:

- Các dịch vụ backend tích hợp: như xác thực người dùng, hệ thống ghép trận, bảng xếp hạng, trò chuyện và theo dõi trận đấu. Các dịch vụ này được thiết kế dưới dạng mô-đun có thể cấu hình, giao tiếp thông qua HTTP API hoặc WebSocket, và dễ dàng tích hợp vào ứng dụng client.
- Máy chủ trò chơi: được triển khai bằng container sử dụng AWS Fargate, có nhiệm vụ xử lý logic trò chơi, duy trì trạng thái ván đấu và đồng bộ theo thời gian thực giữa các người chơi.

Ứng dụng phía client sẽ tương tác với hệ thống thông qua hai hướng chính:

- Gửi yêu cầu tới các dịch vụ backend để thực hiện các thao tác như đăng nhập, tìm trận, gửi tin nhắn hoặc xem bảng xếp hạng.
- Kết nối trực tiếp đến game server để tham gia ván đấu, nhận và gửi dữ liệu thời gian thực.

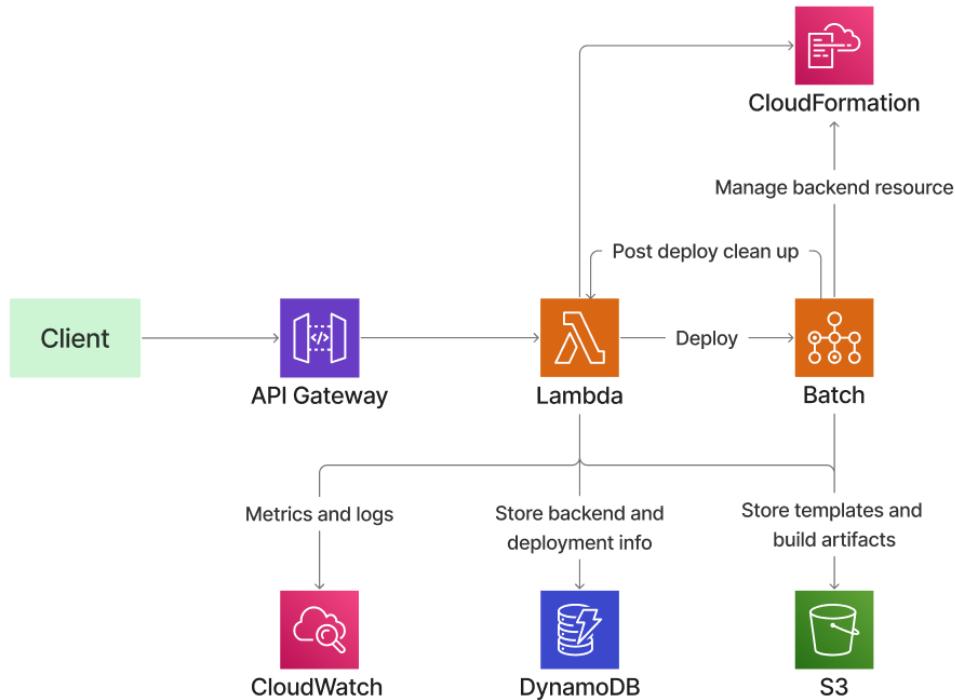
Cách tổ chức này cho phép backend và game server có thể mở rộng độc lập. Ví dụ, khi nhu cầu ghép trận tăng nhưng số lượng ván đấu không thay đổi nhiều, hệ thống chỉ cần mở rộng backend mà không phải tạo thêm game server, từ đó tối ưu tài nguyên và chi phí vận hành.

Tổng thể kiến trúc được triển khai theo mô hình cloud-native, kết hợp giữa serverless và container hóa, tận dụng toàn diện các dịch vụ AWS để:

- Tự động mở rộng theo tải thực tế
- Duy trì hiệu suất ổn định khi tải tăng cao
- Giảm chi phí trong thời gian thấp điểm
- Rút ngắn thời gian triển khai và vận hành sản phẩm mới

Đây là một kiến trúc hiện đại, phù hợp cho các nền tảng trò chơi yêu cầu khả năng mở rộng linh hoạt, vận hành bền vững và tối ưu hóa hiệu suất trong môi trường sản xuất thực tế.

4.1.4 Luồng hoạt động của hệ thống



Hình 4.2 Luồng hoạt động của hệ thống PaaS

Luồng hoạt động của hệ thống PaaS triển khai backend trò chơi được tổ chức theo mô hình sự kiện bắt đồng bộ, đảm bảo khả năng tự động hóa, mở rộng linh hoạt và vận hành hiệu quả. Quá trình xử lý từ khi client gửi yêu cầu cho đến khi backend trò chơi được triển khai hoàn chỉnh có thể được mô tả theo các bước sau:

Đầu tiên, yêu cầu triển khai backend trò chơi được gửi đến thông qua API Gateway. Khi nhận được yêu cầu từ API Gateway, hàm Lambda tương ứng thực hiện xử lý nghiệp vụ bao gồm kiểm tra hợp lệ yêu cầu, lưu trữ thông tin cấu hình backend, và quyết định các bước triển khai tiếp theo. Lambda đồng thời ghi nhận sự kiện, trạng thái và thông số kỹ thuật của backend vào DynamoDB để phục vụ cho việc theo dõi và tra cứu sau này.

Để thực hiện các công việc triển khai phức tạp như build artifact hoặc khởi tạo tài nguyên backend, Lambda gửi một nhiệm vụ tới AWS Batch. Batch chịu trách nhiệm thực hiện các tác vụ nặng như:

- Xây dựng artifact từ các đoạn mã nguồn để sử dụng trong hàm Lambda cho backend và đẩy lên Amazon S3.
- Lưu trữ các template tài nguyên và đẩy artifact đã build lên Amazon S3.
- Tạo các template định nghĩa tài nguyên dựa trên cấu hình của nhà phát triển và yêu cầu AWS CloudFormation triển khai tài nguyên dựa trên các template đó.

CloudFormation tiếp nhận yêu cầu từ Batch và khởi tạo hoặc cập nhật tài nguyên backend. Sau khi quá trình triển khai hoàn tất, Batch cập nhật trạng thái nhiệm vụ và kích hoạt Lambda xử lý dọn dẹp sau triển khai.

4.2 Bộ công cụ triển khai backend trò chơi

4.2.1 API tích hợp các dịch vụ backend

Để hỗ trợ triển khai một nền tảng trò chơi, các hệ thống con trong backend trò chơi đã trình bày trước đó sẽ được cung cấp dưới dạng các dịch vụ tích hợp sẵn. Trong đó, mỗi dịch vụ đảm nhiệm một chức năng riêng biệt và có thể hoạt động độc lập hoặc phối hợp linh hoạt tùy theo nhu cầu cụ thể của từng trò chơi. Việc thiết kế theo hướng module hóa giúp backend đạt được tính tái sử dụng, khả năng mở rộng và dễ bảo trì, đồng thời cho phép nhà phát triển lựa chọn kích hoạt hoặc vô hiệu hóa từng dịch vụ backend trong quá trình cấu hình hệ thống.

Bảng 4.1 Các dịch vụ trong hệ thống backend trò chơi

Dịch vụ	Mô tả chức năng chính
Xác thực	Quản lý đăng nhập, xác thực và định danh người dùng
Ghép trận	Ghép cặp người chơi theo điểm xếp hạng hoặc chế độ chơi
Xếp hạng	Quản lý điểm số, bảng xếp hạng (ELO, Glicko, v.v.)
Bạn bè	Quản lý danh sách bạn bè, lời mời kết bạn, trạng thái
Trò chuyện	Hỗ trợ người chơi gửi và nhận tin nhắn trong trận đấu hoặc khi theo dõi trận, với kiểm duyệt nội dung
Theo dõi trận đấu	Cho phép người chơi theo dõi diễn biến trận đấu theo thời gian thực, đăng ký nhận cập nhật trạng thái mới

Các API được triển khai thông qua Amazon API Gateway (HTTP và WebSocket), AppSync (GraphQL) với Lambda xử lý logic nghiệp vụ và DynamoDB làm tầng lưu trữ và đồng bộ dữ liệu.

Bảng 4.2 API Endpoint cho backend trò chơi

API	Endpoint	Phương thức	Mô tả
Ghép trận	/matchmaking	POST	Thực hiện tìm trận

	/queuing	WebSocket	Chờ tìm trận
Xếp hạng	/userRatings	GET	Truy vấn bảng xếp hạng người chơi
Bạn bè	/friends	GET	Lấy danh sách bạn bè
	/friend/{id}	DELETE	Xóa kết bạn
	/friend/request	POST	Gửi lời mời kết bạn
	/friend/accept	POST	Chấp nhận lời mời kết bạn
	/friend/reject	POST	Từ chối lời mời kết bạn
	/friendRequests/received	GET	Lấy danh sách lời mời đã nhận
	/friendRequests/sent	GET	Lấy danh sách lời mời đã gửi
Trò chuyện	sendMessage	GraphQL Mutation	Gửi tin nhắn trong nhóm hoặc cho người dùng cụ thể
	onMessageSent	GraphQL Subscription	Nhận tin nhắn thời gian thực
Theo dõi trận đấu	updateMatchState	GraphQL Mutation	Cập nhật trạng thái trận đấu
	onMatchStateUpdated	GraphQL Subscription	Nhận cập nhật trạng thái trận đấu thời gian thực
	matchSpectate	GET	Lấy thông tin trận đấu để theo dõi
Khác	/user	GET	Lấy thông tin người dùng
	/avatar	POST	Tải ảnh người dùng lên hệ thống
	/activeMatches	GET	Lấy danh sách các trận đấu đang diễn ra
	/matchRecord	GET	Lấy thông tin lịch sử một trận đấu cụ thể
	/matchStates	GET	Lấy các trạng thái của trận đấu

	/matchResults	GET	Lấy kết quả các trận đấu
	/match/{id}/restore	POST	Khôi phục trận đấu khi có lỗi

4.2.2 SDK hỗ trợ triển khai máy chủ trò chơi

Trong hệ thống được xây dựng, việc triển khai các máy chủ trò chơi là một thành phần quan trọng, nhằm phục vụ các trò chơi có tính chất thời gian thực và hỗ trợ nhiều người chơi cùng lúc. Để đơn giản hóa quá trình triển khai, vận hành và mở rộng các máy chủ trò chơi, sinh viên đã phát triển một SDK với mục tiêu:

- Cung cấp bộ công cụ hỗ trợ lập trình viên dễ dàng tích hợp với backend serverless đã xây dựng.
- Tự động xử lý các nghiệp vụ phổ biến như kết nối người chơi, đồng bộ trạng thái ván đấu, ghi nhận kết quả.
- Hỗ trợ triển khai các máy chủ trò chơi dựa trên chuẩn giao tiếp WebSocket.
- Tách biệt rõ ràng phần logic trò chơi với phần hạ tầng xử lý, từ đó tăng tính tái sử dụng và dễ bảo trì.

Ngôn ngữ Go được lựa chọn để xây dựng SDK nhờ ưu thế vượt trội trong xử lý đồng thời, hiệu suất cao, và độ trễ thấp khi vận hành máy chủ thời gian thực. So với Node.js, Python hoặc Java, Go cung cấp mô hình concurrency dựa trên goroutine rất nhẹ, giúp máy chủ xử lý đồng thời hàng ngàn kết nối với footprint tài nguyên nhỏ. Điều này phù hợp đặc biệt cho máy chủ backend trò chơi trực tuyến.

SDK được tổ chức thành nhiều thành phần, mỗi thành phần đảm nhiệm một chức năng riêng biệt. Các thành phần chính bao gồm:

- Server: Khởi tạo máy chủ WebSocket, xử lý xác thực người dùng, quản lý các ván đấu đang hoạt động.
- Match: Đại diện cho một ván đấu, bao gồm danh sách người chơi, kênh xử lý lượt đi, và các callback xử lý khi ván đấu kết thúc hoặc bị huỷ.
- Player: Đại diện cho một người chơi đang kết nối, bao gồm thông tin định danh, trạng thái kết nối và kênh gửi nhận dữ liệu.
- Move: Gói dữ liệu biểu diễn một lượt đi hoặc thao tác do người chơi gửi lên.
- 'MatchHandler' và 'ServerHandler': Giao diện cho phép nhà phát triển định nghĩa logic xử lý trò chơi, từ xử lý lượt đi cho đến ghi nhận kết quả trận đấu.

Mỗi quan hệ giữa các thành phần được thiết kế theo hướng mở rộng dễ dàng, cho phép nhà phát triển chỉ cần cài đặt các phương thức trong MatchHandler và ServerHandler để xây dựng trò chơi riêng của mình, mà không cần can thiệp vào phần hạ tầng xử lý WebSocket, xác thực, lưu trữ dữ liệu hoặc tương tác với các dịch vụ AWS.

Cách sử dụng SDK

Để xây dựng một trò chơi mới, nhà phát triển chỉ cần định nghĩa lớp xử lý logic bằng cách cài đặt các phương thức cần thiết trong 'ServerHandler' và 'MatchHandler', chẳng hạn:

```
/*
 * Implement ServerHandler interface
 */
type MyServerHandler struct{}

func (h *MyServerHandler) OnMatchCreate(match entities.ActiveMatch) (*server.Match, error) {
    return nil, nil
}
```

Hình 4.3 Định nghĩa lớp xử lý thông qua giao diện ServerHandler

Hình 4.3 mô tả cài đặt riêng của nhà phát triển trò chơi cho logic tạo trận đấu. SDK sẽ lo phần truy vấn dữ liệu trận đấu và lưu trữ thông tin liên quan cùng với quản lý các trận đấu. Nhà phát triển chỉ cần quan tâm tới chi tiết cụ thể để tạo trận đấu tương ứng với trò chơi của mình.

Sau khi cài đặt đầy đủ các phương thức xử lý, máy chủ trò chơi có thể được khởi tạo thông qua SDK như minh họa dưới đây. Quá trình khởi tạo yêu cầu truyền vào các lớp xử lý (handler) phù hợp với từng trò chơi cụ thể:

```
// Run server
func main() {
    serverHandler := NewServerHandler()
    matchHanndler := NewMatchHandler()
    cfg := server.NewConfig("7202", serverHandler, matchHanndler)
    srv := server.NewFromConfig(cfg)
    logging.Fatal("server runtime error", zap.Error(srv.Start()))
}
```

Hình 4.4 Khởi tạo máy chủ trò chơi thông qua SDK

SDK sẽ tự động đảm nhiệm các chức năng sau:

- Quản lý kết nối WebSocket từ người chơi đến máy chủ;

- Xác thực người dùng thông qua mã thông báo JWT;
- Quản lý vòng đời của các ván đấu bao gồm tạo, lưu, và kết thúc;
- Tích hợp với các dịch vụ backend như DynamoDB, Lambda, AppSync;
- Uỷ quyền xử lý logic trò chơi cho các handler do nhà phát triển định nghĩa.

Tích hợp với backend serverless

SDK được thiết kế để hoạt động chặt chẽ với các dịch vụ AWS, đảm nhiệm phần tích hợp backend phức tạp như sau:

Bảng 4.3 Tích hợp giữa SDK và các dịch vụ AWS

Dịch vụ AWS	Vai trò trong SDK
Amazon Cognito	Xác thực người dùng bằng JWT
Amazon DynamoDB	Lưu trạng thái ván đấu và khôi phục khi cần thiết
AWS Lambda	Ghi nhận kết quả, cập nhật thông tin trận đấu
AWS AppSync	Đồng bộ trạng thái ván đấu theo thời gian thực

Qua các dịch vụ này, SDK đảm bảo rằng việc lưu trữ, ghi nhận kết quả và đồng bộ trạng thái được thực hiện một cách tự động, an toàn và tiết kiệm chi phí, đúng với định hướng kiến trúc serverless.

4.3 Quy trình triển khai mẫu

Để làm rõ tính khả thi và tiện ích của giải pháp PaaS được đề xuất, mục này trình bày một quy trình triển khai mẫu, mô phỏng toàn bộ các bước mà một nhà phát triển trò chơi sẽ thực hiện khi sử dụng nền tảng để triển khai backend cho một trò chơi trực tuyến. Quy trình này minh họa rõ ràng khả năng tự động hóa, khả năng mở rộng và sự tiện lợi mà nền tảng mang lại trong thực tế.

Bước 1: Phát triển máy chủ trò chơi

The screenshot shows a user interface for managing artifacts. At the top, there are dropdown menus for 'Image tag' (set to 'latest'), 'Artifact type' (set to 'Image'), 'Pushed at' (set to 'April 11, 2025, 01:46:45 (UTC+07)'), 'Size (MB)' (set to '12.98'), 'Image URI' (with a copy icon), and 'Digest' (with a copy icon). Below this, a table lists the artifact details: '3453b60e7c752ef4940c4b58624300f454e4084b, latest' under 'Image tag', 'Image' under 'Artifact type', and 'sha256:ba5c8bbb4477f9...' under 'Digest'. A note at the bottom right says 'April 11,'.

Hình 4.5 Build và lưu trữ Container Image

Với mỗi trò chơi, nhà phát triển xây dựng máy chủ trò chơi sử dụng SDK được cung cấp. Máy chủ trò chơi sau khi phát triển cần được build thành container image và lưu trữ trên các container registry như AWS ECR, Docker Hub.

Bước 2: Cài đặt triển khai

Deploy New Game Backend

Stack name
Enter a stack name

Backend Services

Authentication (Required)

Chat

Friend

Matchmaking (Required)

Ranking

Match Spectating

Matchmaking Configuration

Players per match
2

Server Configuration

Server image URI
e.g., 123456789012.dkr.ecr.ap-southeast-1.amazonaws.com/mygame-server:latest

Private registry

Max concurrent matches (per server)
100

Processor (vCPU)
0.5

Memory (GB)
1

Deploy

Hình 4.6 Triển khai backend trò chơi

Sau khi đăng nhập vào giao diện web của nền tảng giải pháp, nhà phát triển tiến hành triển khai backend mới dành riêng cho trò chơi của mình. Trong quá trình khởi tạo, người dùng được cung cấp giao diện cấu hình để lựa chọn các dịch vụ backend cần thiết cho nền tảng, bao gồm:

- Dịch vụ xác thực người chơi.
- Dịch vụ ghép trận.

- Dịch vụ xếp hạng.
- Dịch vụ trò chuyện.
- Dịch vụ theo dõi trận đấu.

Việc lựa chọn dịch vụ được thực hiện thông qua giao diện đồ họa trực quan và hệ thống sẽ tự động cấu hình các tham số tương ứng cho từng dịch vụ.

Nhà phát triển cũng thực hiện các cấu hình chuyên biệt cho trò chơi, như:

- Chọn thuật toán ghép trận dựa trên ELO, với sai lệch tối đa 100 điểm.
- Bật tính năng theo dõi trận đấu cho phép khán giả xem thời gian thực.
- Áp dụng hệ thống xếp hạng Glicko để phản ánh biến động điểm chính xác hơn.
- Điện URL để kéo Docker Image của máy chủ trò chơi về khi triển khai.
- Cấu hình tài nguyên cho máy chủ trò chơi.

Với mỗi trò chơi, nhà phát triển xây dựng máy chủ trò chơi sử dụng SDK được cung cấp. SDK sẽ cung cấp các cấu trúc và giao diện cần thiết để xử lý vòng đời trận đấu, tương tác với người chơi, và tích hợp với hệ thống backend serverless.

Máy chủ trò chơi trò chơi sau khi phát triển cần được build thành Container Image và lưu trữ trên AWS ECR, Docker Hub hoặc các dịch vụ tương tự.

Bước 3: Sinh tệp hạ tầng triển khai

Name	Type	Last modified	Size	Storage class
appsync.yaml	yaml	April 28, 2025, 17:20:47 (UTC+07:00)	5.5 KB	Standard
auth.yaml	yaml	April 28, 2025, 17:20:47 (UTC+07:00)	2.8 KB	Standard
compute.yaml	yaml	April 28, 2025, 17:20:47 (UTC+07:00)	10.4 KB	Standard
customization.yaml	yaml	April 28, 2025, 17:20:41 (UTC+07:00)	1.9 KB	Standard
httpApi.yaml	yaml	April 28, 2025, 17:20:47 (UTC+07:00)	22.3 KB	Standard
log.yaml	yaml	April 28, 2025, 17:20:46 (UTC+07:00)	550.0 B	Standard
storage.yaml	yaml	April 28, 2025, 17:20:47 (UTC+07:00)	13.9 KB	Standard
template.yaml	yaml	April 28, 2025, 17:20:47 (UTC+07:00)	5.6 KB	Standard
websocketApi.yaml	yaml	April 28, 2025, 17:20:47 (UTC+07:00)	6.8 KB	Standard

Hình 4.7 Các template định nghĩa tài nguyên

Sau khi hoàn tất cấu hình, hệ thống sẽ tự động sinh ra các tệp template theo định dạng của AWS SAM, mô tả toàn bộ kiến trúc hạ tầng dưới dạng mã. Các tệp này định nghĩa các tài nguyên như:

- Các hàm Lambda phục vụ API.
- Các bảng DynamoDB cho lưu trữ trạng thái.
- Cấu hình API Gateway (REST/WebSocket) hoặc AWS AppSync.
- Cấu hình ECS Service sử dụng Fargate cho trò chơi.
- Các trigger, quyền truy cập (IAM), và biến môi trường tương ứng.

Bước 4: Hệ thống tự động triển khai

The screenshot shows the Ludofy Platform's Deployment History page. On the left, there is a sidebar with navigation links: Home, Deploy Backend, Backends, Deployments (which is highlighted in blue), Help, and Settings. The main content area is titled "Deployment History". It lists four deployment entries for the service "chessworld":

- Deployment ID: f4fdd010-26ed-4eb5-ac87-18b055360fb5 (successful, View / Manage)
- Deployment ID: f4fdd010-26ed-4eb5-ac87-18b055360fb5 (successful, View / Manage)
- Deployment ID: f4fdd010-26ed-4eb5-ac87-18b055360fb5 (successful, View / Manage)
- Deployment ID: f4fdd010-26ed-4eb5-ac87-18b055360fb5 (failed)

Each entry includes details like Server Image, Services Enabled, Customization, Matchmaking Configuration, and Server Configuration.

Hình 4.8 Trạng thái triển khai của backend

Sau khi sinh tệp template.yaml, hệ thống nộp yêu cầu lên AWS Batch Job để đợi triển khai backend. Quá trình triển khai diễn ra tự động và không yêu cầu nhà phát triển thao tác thủ công với từng dịch vụ điện toán đám mây riêng lẻ.

Bước 5: Kết nối phía client và thử nghiệm trò chơi

The screenshot shows a list of API endpoints under the heading "API Endpoints". There are three items listed:

- ActiveMatchListEndpointUrl
GET <https://z863s8oqkh.execute-api.ap-southeast-2.amazonaws.com/dev/activeMatches?limit=5&startKey=<START-KEY>>
- AppSyncGraphQLApiUrl
<https://q6ra2sztpne3njwbm2ojzym23i.appsync-api.ap-southeast-2.amazonaws.com/graphql>
- AppSyncRealtimeApiUrl
<wss://q6ra2sztpne3njwbm2ojzym23i.appsync-realtime-api.ap-southeast-2.amazonaws.com/graphql>

Hình 4.9 Danh sách API Endpoints cụ thể cho từng backend

Sau khi triển khai hoàn tất, hệ thống cung cấp các endpoint cụ thể để trò chơi client có thể kết nối. Nhà phát triển chỉ cần cập nhật URL trong client để kết nối với các backend service vừa triển khai. Các trận đấu có thể diễn ra ngay sau đó với đầy đủ các chức năng hỗ trợ từ phía backend.

Bước 6: Giám sát hoạt động của backend

The screenshot shows the "Backend Detail" page. On the left is a sidebar with navigation links: Home, Deploy Backend, Backends, Deployments, Help, and Settings. The main area displays the following information:

- ID:** 78115531-1c1f-4ef3-aad9-19f56314de14
- Stack Name:** chesster
- Created At:** 4/28/2025, 1:51:43 AM
- Active Matches:** 0
- Avg CPU Usage:** 0.03 %
- Avg Memory Usage:** 0.99 %

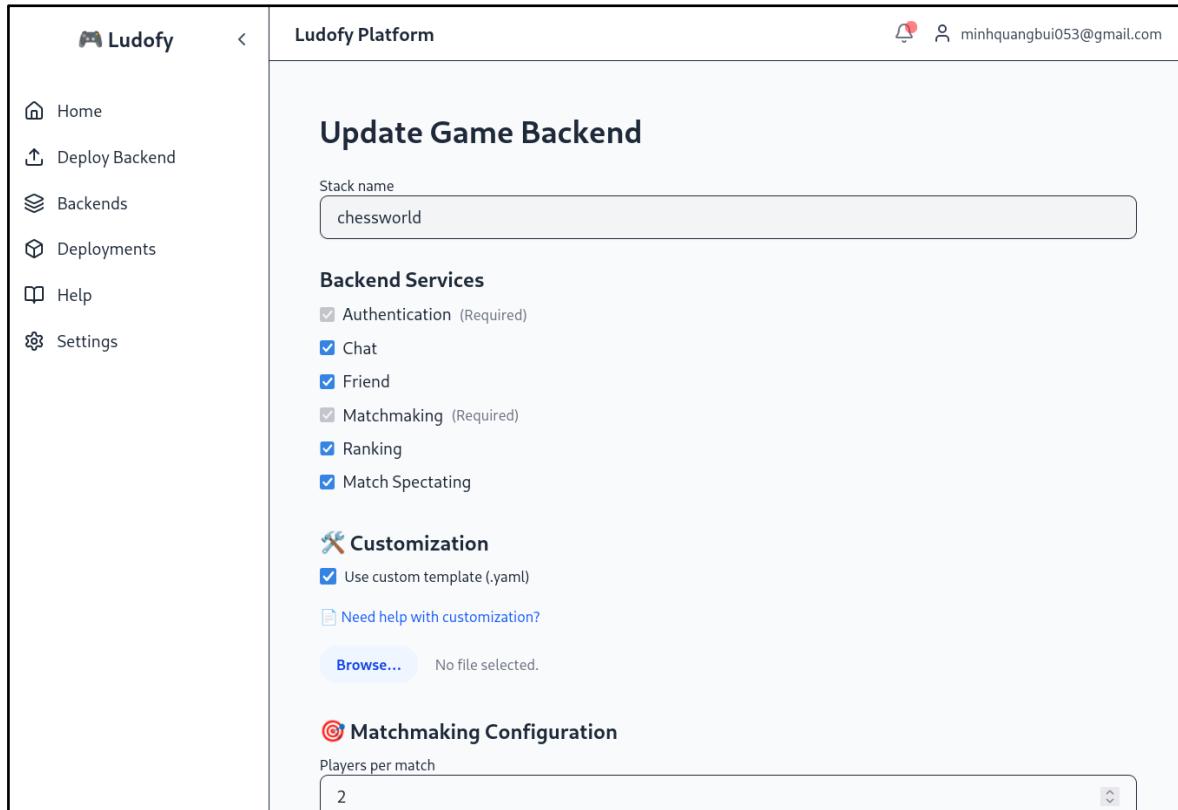
Below this is a chart titled "Resource Utilization" showing CPU and Memory usage over the last week. The Y-axis is "Usage (%)" from 0 to 2.0. The X-axis shows dates from 23:59:00 to 04:47:00. The CPU usage is consistently at 0%, while Memory usage fluctuates between 0.8% and 1.0%.

Hình 4.10 Giao diện giám sát tài nguyên và trạng thái hoạt động của backend

Sau khi trò chơi đi vào hoạt động, nền tảng cung cấp một bảng điều khiển giám sát để theo dõi các thông số quan trọng như:

- Số lượng người dùng đồng thời.
- Số lượng máy chủ đang chạy
- Mức tiêu thụ CPU/RAM.
- Thống kê số lượt ghép trận, số trận đang hoạt động, độ trễ trung bình,...

Bước 7: Cập nhật và mở rộng backend



Hình 4.11 Giao diện cập nhật backend

Sau khi backend trò chơi đã được triển khai, người dùng có thể dễ dàng cập nhật hoặc mở rộng backend nếu muốn. Người dùng có thể bật tắt các dịch vụ tùy theo nhu cầu và thay đổi các cấu hình đã cài đặt trước đó. Ngoài ra, hệ thống cung cấp khả năng tùy chỉnh và mở rộng backend thông qua tập tin template tùy chỉnh.

Khi cập nhật, hệ thống sẽ giữ nguyên dữ liệu và trạng thái hiện tại của backend trò chơi, đồng thời đảm bảo rollback an toàn nếu quá trình cập nhật gặp lỗi và đưa backend về trạng thái ổn định gần nhất, tránh gây gián đoạn hệ thống đang hoạt động.

Cách triển khai này giúp đảm bảo rằng hệ thống backend luôn mở rộng linh hoạt, không ngắt quãng hoạt động, đồng thời vẫn đảm bảo tính an toàn và tự động hóa cao trong quá trình vận hành.

Chương 5: Triển khai và đánh giá kết quả

5.1 Mục tiêu đánh giá

Mục tiêu của chương này là phân tích và đánh giá mức độ hiệu quả, tính khả thi và các ưu điểm của giải pháp PaaS xây dựng backend cho trò chơi dựa trên kiến trúc serverless. Các tiêu chí đánh giá bao gồm:

- **Khả năng triển khai:** Đo lường thời gian và công sức cần thiết để triển khai một backend trò chơi hoàn chỉnh.
- **Khả năng mở rộng:** Đánh giá khả năng tự động mở rộng và co dãn của hệ thống tùy theo lưu lượng người chơi.
- **Tối ưu chi phí:** Đánh giá chi phí khi sử dụng mô hình serverless so với mô hình truyền thống.
- **Khả năng cấu hình và tái sử dụng dịch vụ:** Đánh giá khả năng tùy chỉnh và tái sử dụng các dịch vụ chung như matchmaking, ranking, chat,... cho nhiều game.
- **Hiệu suất:** Đo lường độ trễ và thời gian phản hồi khi người dùng tương tác với hệ thống.

5.2 Môi trường thử nghiệm

Mô hình trả phí theo mức sử dụng (pay-per-use) của kiến trúc serverless giúp tối ưu chi phí vận hành trong các hệ thống xử lý theo sự kiện rời rạc, nơi hành động của người dùng xảy ra không liên tục. Do đó, kiến trúc này đặc biệt phù hợp với các thể loại trò chơi có tập trạng thái hữu hạn như trò chơi theo lượt, trò chơi bàn cờ, thẻ bài, giải đố hoặc trò chơi gia tăng tự động. Đặc điểm chung của những trò chơi này là hành động của người chơi diễn ra cách quãng và không yêu cầu xử lý liên tục với tải cao.

Bên cạnh đó, các trò chơi dạng này thường không tiêu tốn nhiều tài nguyên xử lý. Nhờ vậy, hệ thống có thể triển khai các “vi máy chủ” (micro-sized servers) với cấu hình thấp, phục vụ một số lượng giới hạn trận đấu đồng thời. Cách tiếp cận này giúp tối ưu hóa chi phí vận hành, đồng thời cho phép mở rộng linh hoạt theo sát số lượng người chơi hoạt động thực tế.

Trong đồ án, sinh viên triển khai một nền tảng chơi cờ vua dựa trên giải pháp đã đề xuất để đánh giá tính phù hợp và hiệu quả của giải pháp trong thực tế. Môi trường thử nghiệm bao gồm:

- **Nền tảng cloud:** AWS
- **Hệ thống thử nghiệm:** Nền tảng chơi cờ vua triển khai thông qua giải pháp PaaS.

- Số lượng người dùng mô phỏng: 100 - 10.000 người chơi đồng thời
- Công cụ mô phỏng: k6

5.2.1 Triển khai nền tảng chơi cờ vua trực tuyến

Xây dựng máy chủ trò chơi

Phần cốt lõi của quá trình triển khai nằm ở việc phát triển logic trò chơi và sử dụng SDK của giải pháp PaaS để tích hợp với máy chủ trò chơi. Ví dụ như sau:

```
type MyServerHandler struct{}
```

```
func (h *MyServerHandler) OnMatchCreate(match entities.ActiveMatch) (server.Match, error) {
    cfg, err := ConfigForGameMode(match.GameMode)
    if err != nil {
        return nil, fmt.Errorf("failed to get config: %w", err)
    }
    players := make(map[string]server.Player, len(match.Players))
    for i, player := range match.Players {
        players[player.Id] = &Player{
            Player: server.NewDefaultPlayer(player.Id, match.MatchId),
            Clock:  cfg.MatchDuration,
            Side:   i%2 == 0,
        }
    }
    return Match{
        Match: server.NewDefaultMatch(match.MatchId, players),
        cfg:   cfg,
    }, nil
}
```

Hình 5.1 Cài đặt logic xử lý khi trận đấu được tạo

Hình 5.1 mô tả cài đặt mẫu cho logic xử lý khi trận đấu được tạo. Nhà phát triển có thể tự định nghĩa và cấu hình trận đấu theo logic trò chơi của mình. Với trò chơi cờ vua, trận đấu có thêm các thông tin như đồng hồ đếm ngược cho từng người chơi, chia người chơi thành hai phe trắng hoặc đen... Đối tượng trận đấu cụ thể do nhà phát triển định nghĩa được trả về để sử dụng trong hệ thống.

Sau khi hoàn thiện tất cả các logic xử lý, mã nguồn được xây dựng thành container image và đẩy lên Amazon ECR.

ludofy/server-example	 588738569386.dkr.ecr.ap-southeast-2.amazonaws.com/ludofy/server-example
---------------------------------------	---

Hình 5.2 Lưu trữ container image trên Amazon ECR

Triển khai backend trò chơi

Deploy New Game Backend

Stack name
chessworld

Backend Services

Authentication (Required)

Chat

Friend

Matchmaking (Required)

Ranking

Match Spectating

Matchmaking Configuration

Players per match
2

Rating algorithm
Glicko

Initial rating
1200

Server Configuration

Server image URI
588738569386.dkr.ecr.ap-southeast-2.amazonaws.com/ludofy/server-example:latest

Private registry

Max concurrent matches (per server)
100

Processor (vCPU)
0.25

Memory (GB)
0.5

Deploy

Hình 5.3 Triển khai backend trò chơi cờ vua

Backend trò chơi cờ vua được triển khai với cấu hình chi tiết như sau:

- Tên stack: "chessworld" – tên này sẽ được dùng làm định danh cho toàn bộ tài nguyên của backend triển khai trên AWS.
- Các dịch vụ backend mong muốn:
 - Authentication: xác thực người chơi.
 - Chat: trò chuyện trong trận.
 - Friend: kết bạn và quản lý danh sách bạn.

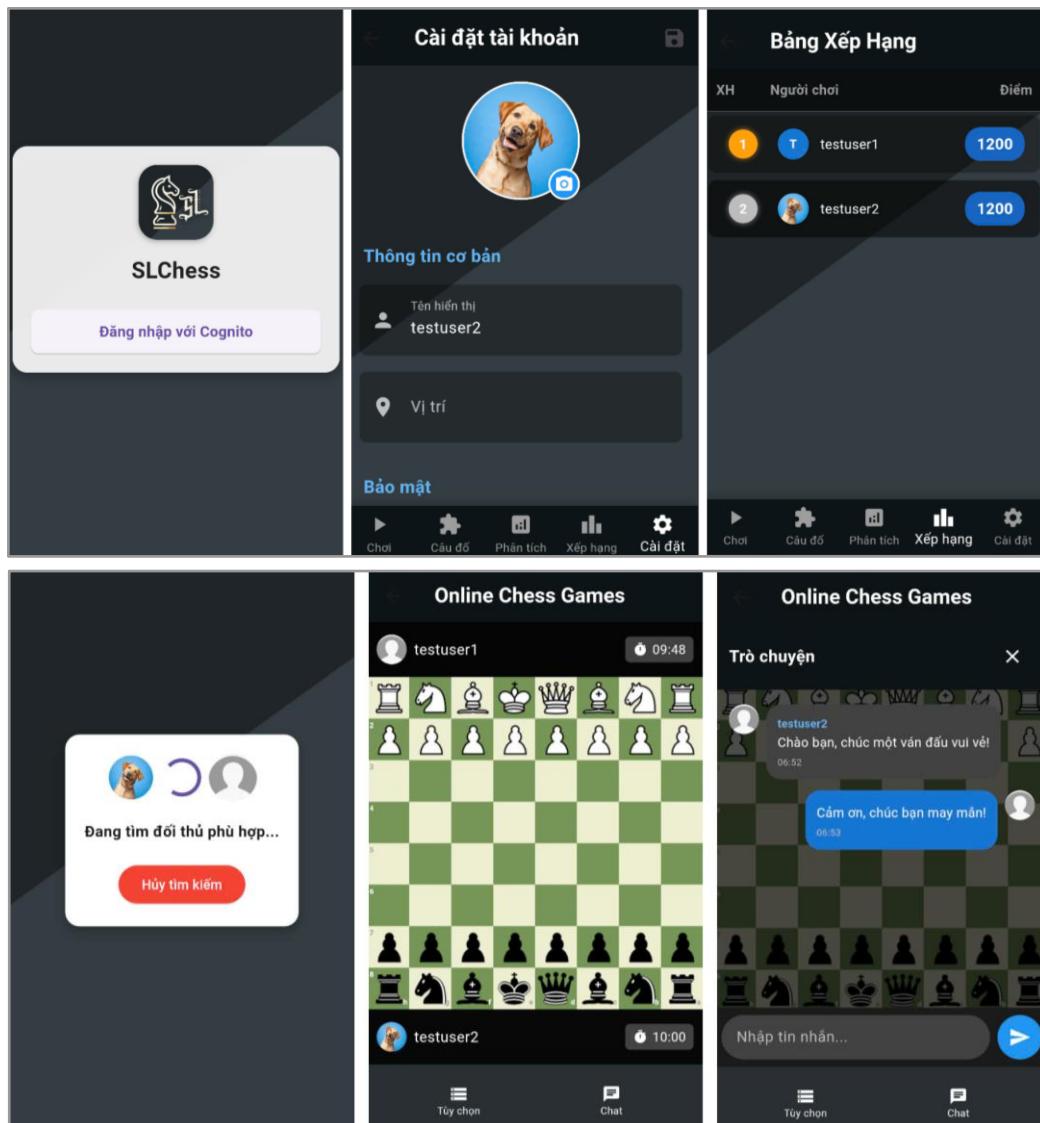
- Matchmaking: ghép trận tự động.
 - Ranking: bảng xếp hạng người chơi.
 - Match Spectating: hỗ trợ người dùng theo dõi trận đấu.
- Cấu hình hệ thống ghép trận:
 - Số người chơi mỗi trận: 2.
 - Thuật toán xếp hạng: Glicko.
 - Mức điểm ban đầu: 1200.
- Cấu hình máy chủ trò chơi:
 - Đường dẫn đến container image của máy chủ trò chơi, được lưu trữ trên Amazon ECR.
 - Tài nguyên máy chủ: 0.25 vCPU, 0.5 GB bộ nhớ và tối đa 100 người chơi đồng thời trên một máy chủ.

Phát triển ứng dụng chơi cờ vua

Sau khi triển khai backend, một ứng dụng di động mẫu đã được phát triển để kết nối và tương tác trực tiếp với các dịch vụ backend. Ứng dụng hỗ trợ các tính năng chính sau:

- Chế độ chơi cờ trực tuyến và ngoại tuyến, cho phép người dùng ghép trận và tham gia thi đấu online qua kết nối WebSocket hoặc chơi offline với hai người dùng chung thiết bị.
- Tính năng giải câu đố cho phép người chơi luyện tập các thế cờ dựa trên cấp độ xếp hạng hiện tại.
- Ứng dụng tích hợp khả năng phân tích ván đấu nhờ engine Stockfish, hỗ trợ đánh giá các nước đi của người chơi.
- Tính năng xếp hạng người chơi sử dụng thuật toán Glicko nhằm đảm bảo việc tính điểm và phân hạng một cách chính xác. Hệ thống sẽ điều chỉnh điểm số dựa trên kết quả thi đấu, giúp duy trì tính cạnh tranh và công bằng trong cộng đồng người chơi.
- Quản lý tài khoản người dùng thông qua quy trình đăng ký, đăng nhập và xác thực tài khoản tích hợp với AWS Amplify. Điều này đảm bảo sự an toàn cho thông tin cá nhân và cho phép đăng nhập dễ dàng trên nhiều thiết bị khác nhau.

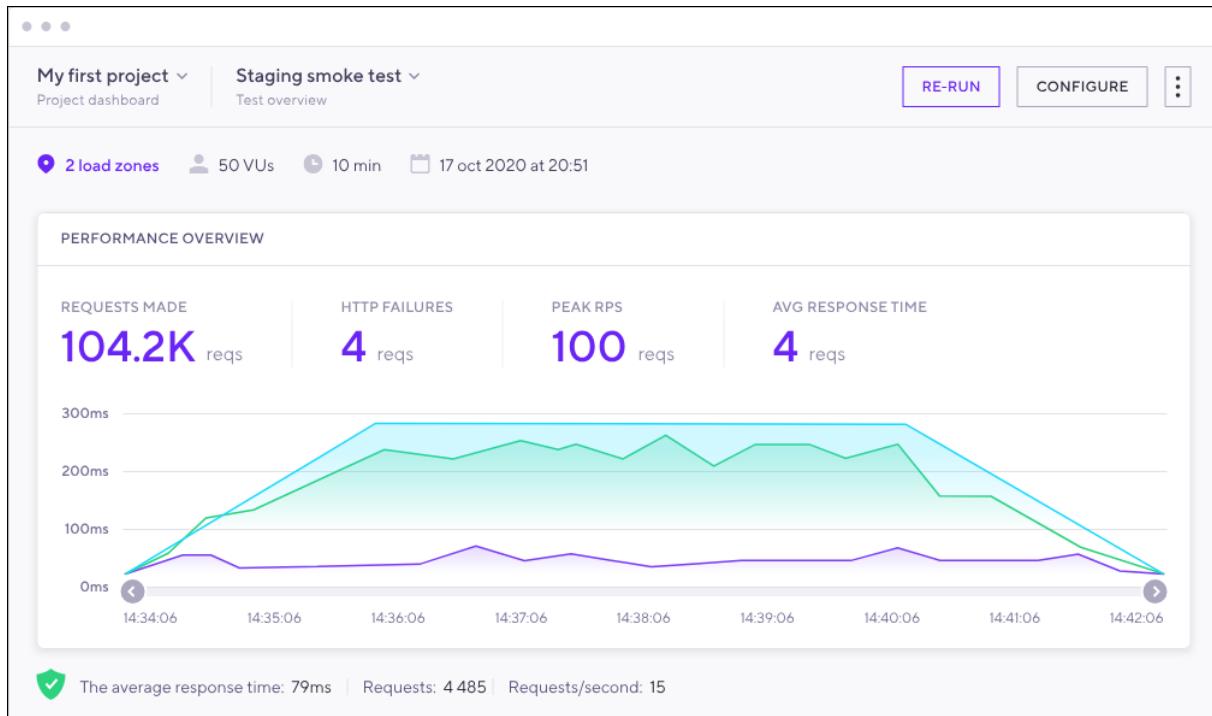
- Truy cập lịch sử các trận đấu đã chơi, xem lại toàn bộ diễn biến ván cờ và phân tích các nước đi. Tính năng này hỗ trợ người chơi theo dõi quá trình cải thiện kỹ năng cũng như rút kinh nghiệm sau từng trận.
- Tính năng kết bạn với người chơi khác, cho phép người dùng gửi lời mời kết bạn, chấp nhận hoặc từ chối lời mời, và quản lý danh sách bạn bè để dễ dàng kết nối, thi đấu hoặc trò chuyện với những đối thủ quen thuộc.
- Tính năng trò chuyện được hỗ trợ trong cả quá trình thi đấu lẫn ngoài trận đấu. Người chơi có thể trao đổi, thảo luận chiến thuật hoặc giao lưu qua hệ thống nhắn tin thời gian thực, nâng cao mức độ tương tác và trải nghiệm cộng đồng.
- Hệ thống thông báo đẩy, cho phép người dùng nhận thông báo về các sự kiện quan trọng như lời mời chơi, tin nhắn mới, cập nhật bảng xếp hạng hoặc các sự kiện hệ thống, đảm bảo người chơi luôn được kết nối và cập nhật kịp thời.



Hình 5.4 Các màn hình của ứng dụng chơi cờ vua

Hình 5.4 mô tả một số màn hình của ứng dụng chơi cờ vua. Nhờ tập hợp đầy đủ các tính năng kể trên, ứng dụng chơi cờ vua không chỉ đáp ứng nhu cầu thi đấu cơ bản mà còn mang đến một hệ sinh thái chơi cờ trực tuyến hiện đại, chuyên nghiệp và giàu tính tương tác.

5.2.2 Kiểm thử bằng công cụ mô phỏng



Hình 5.5 Giao diện công cụ k6

K6 là công cụ kiểm thử hiệu suất mã nguồn mở, được sử dụng để mô phỏng lượng lớn người dùng truy cập đồng thời và đánh giá khả năng chịu tải của hệ thống. Trong phạm vi đề tài, K6 được sử dụng để kiểm thử toàn diện cả backend dịch vụ và máy chủ trò chơi thời gian thực sau khi triển khai.

Cụ thể, K6 được cấu hình để gửi yêu cầu liên tục đến các API backend. Đồng thời, công cụ này cũng được sử dụng để mô phỏng người chơi kết nối tới máy chủ trò chơi qua WebSocket, thực hiện các hành động trong ván đấu như di chuyển, gửi nước đi và nhận trạng thái ván chơi theo thời gian thực.

Vệc kiểm thử đồng thời cả backend và máy chủ trò chơi cho phép đánh giá mức độ phối hợp giữa các thành phần hệ thống trong điều kiện tải cao. Công cụ thu nhập những chỉ số quan trọng như:

- Tốc độ xử lý yêu cầu (Requests per second - RPS)
- Thời gian phản hồi trung bình và P95

- Tỷ lệ lỗi HTTP hoặc lỗi kết nối WebSocket

Các chỉ số này giúp xác định hiệu suất tổng thể, phát hiện điểm nghẽn và đánh giá khả năng mở rộng của hệ thống trò chơi.

Các kịch bản kiểm thử chi tiết đã được trình bày trong Phụ lục C, giúp đảm bảo tính minh bạch và khả năng tái kiểm thử trong các nghiên cứu hoặc triển khai sau này.

5.2.2.1 Kiểm thử API

Trong các kịch bản kiểm thử API, mỗi người dùng ảo sẽ thực hiện truy cập ngẫu nhiên đến một trong các endpoint backend sau:

- GET /user: truy xuất thông tin tài khoản người dùng
- GET /userRatings: lấy điểm số hoặc cấp bậc xếp hạng
- GET /matchResults: xem kết quả các trận đấu đã hoàn thành
- GET /activeMatches: kiểm tra các trận đấu đang diễn ra
- GET /friends: truy vấn danh sách bạn bè

Tất cả các request đều sử dụng phương thức GET, chủ yếu mang tính đọc dữ liệu (read-only). Danh sách endpoint này được áp dụng thống nhất cho tất cả các kịch bản kiểm thử, nhằm đảm bảo đánh giá được hiệu suất chung của hệ thống dưới các tải khác nhau, mà không bị ảnh hưởng bởi sự khác biệt về logic request.

Kịch bản 1: 100 người dùng đồng thời

```
export const options = {
  stages: [
    { duration: '5m', target: 100 }, // Ramp up to 100 users over 5 minutes
    { duration: '10m', target: 100 }, // Hold 100 users for 10 minutes
    { duration: '5m', target: 0 }, // Gradually scale down to 0 users
  ],
  thresholds: {
    http_req_failed: ['rate<0.02'], // Less than 2% errors
    http_req_duration: ['p(95)<2000'], // 95% of requests < 2s
  },
};

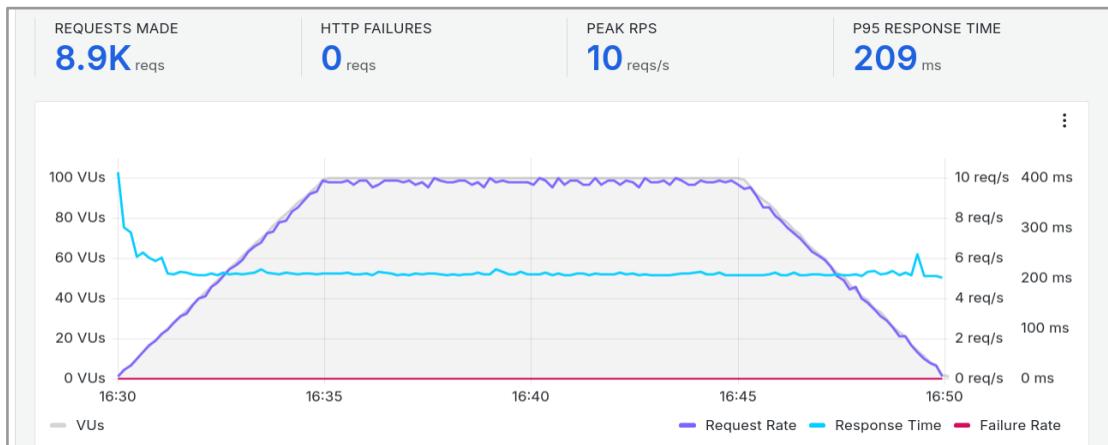
const endpoints = [
  '/user',
  '/userRatings',
  '/matchResults',
  '/activeMatches',
  '/friends',
];

```

Hình 5.6 Kịch bản kiểm thử 100 người dùng

Hình 5.6 mô tả kịch bản kiểm thử hiệu năng với 100 người dùng đồng thời, được thiết kế kéo dài trong 20 phút và chia thành 3 giai đoạn chính:

- Giai đoạn 1 (5 phút đầu): Tăng dần số lượng người dùng từ 0 lên 100, nhằm mô phỏng quá trình hệ thống bắt đầu nhận tải và dần đạt mức tải tối đa.
- Giai đoạn 2 (10 phút tiếp theo): Duy trì ổn định 100 người dùng đồng thời để đánh giá khả năng xử lý liên tục của hệ thống dưới áp lực cao.
- Giai đoạn 3 (5 phút cuối): Giảm dần số lượng người dùng từ 100 về 0, mô phỏng tình huống người dùng rời khỏi hệ thống.



Hình 5.7 Kiểm thử tải với 100 người dùng

Kết quả kiểm thử cho thấy hệ thống đã xử lý 8.881 yêu cầu thành công, không ghi nhận bất kỳ lỗi HTTP nào.

Thông lượng cực đại (peak RPS) đạt mức 10 yêu cầu/giây, trong khi thời gian phản hồi P95 (95% yêu cầu nhanh hơn) là 209ms — cho thấy hệ thống phản hồi ổn định và nhanh chóng trong suốt bài kiểm thử.

Kịch bản 2: 1,000 người dùng đồng thời

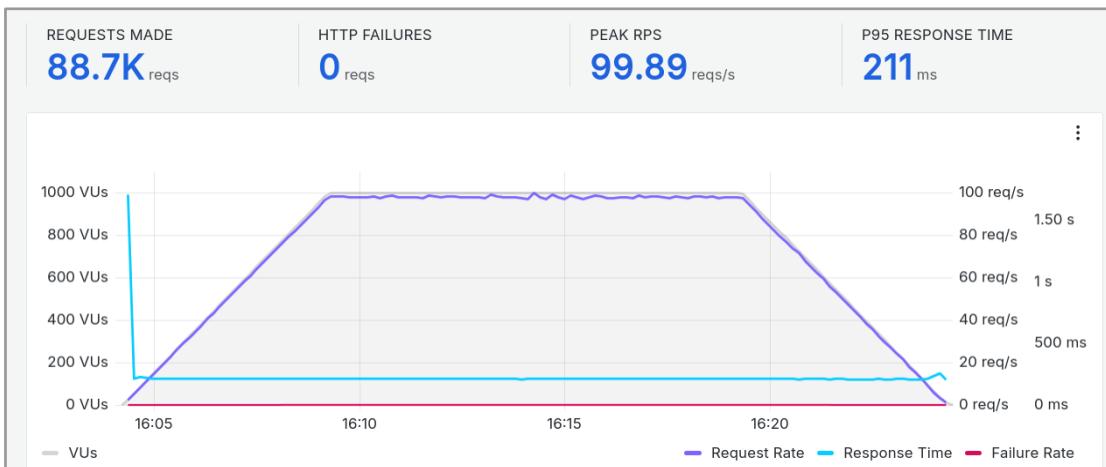
```
export const options = {
  stages: [
    { duration: '5m', target: 1000 }, // Ramp up to 1000 users over 5 minutes
    { duration: '10m', target: 1000 }, // Hold 1000 users for 10 minutes
    { duration: '5m', target: 0 }, // Gradually scale down to 0 users
  ],
  thresholds: {
    http_req_failed: ['rate<0.02'], // Less than 2% errors
    http_req_duration: ['p(95)<2000'], // 95% of requests < 2s
  },
};

const endpoints = [
  '/user',
  '/userRatings',
  '/matchResults',
  '/activeMatches',
  '/friends',
];
```

Hình 5.8 Kịch bản kiểm thử 1000 người dùng

Hình 5.8 minh họa kịch bản kiểm thử hiệu năng với 1000 người dùng đồng thời, được thực hiện trong khoảng thời gian 20 phút và chia thành ba giai đoạn như sau:

- Giai đoạn 1 (5 phút đầu): Tăng dần số lượng người dùng từ 0 lên 1000, mô phỏng việc hệ thống bắt đầu nhận tải lớn trong thời gian ngắn.
- Giai đoạn 2 (10 phút giữa): Duy trì mức tải ổn định với 1000 người dùng đồng thời, cho phép đánh giá khả năng chịu tải lâu dài của hệ thống.
- Giai đoạn 3 (5 phút cuối): Giảm dần số lượng người dùng về 0, mô phỏng tình huống người dùng dần rời khỏi hệ thống.



Hình 5.9 Kiểm thử tải với 1000 người dùng

Kết quả kiểm thử cho thấy hệ thống đã xử lý 88.673 yêu cầu thành công, không ghi nhận bất kỳ lỗi HTTP nào.

Thông lượng cực đại đạt mức 99.89 yêu cầu/giây, trong khi thời gian phản hồi P95 là 211ms, cho thấy độ trễ phản hồi vẫn được giữ ổn định.

Kịch bản 3: 10,000 người dùng đồng thời

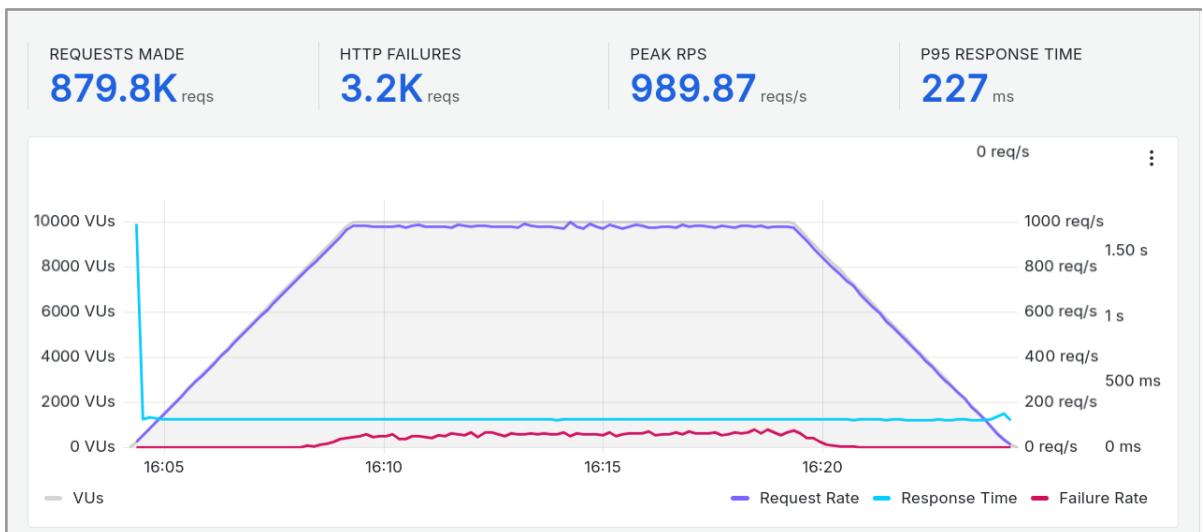
```
export const options = {
  stages: [
    { duration: '5m', target: 10000 }, // Ramp up to 10000 users over 5 minutes
    { duration: '10m', target: 10000 }, // Hold 10000 users for 10 minutes
    { duration: '5m', target: 0 }, // Gradually scale down to 0 users
  ],
  thresholds: {
    http_req_failed: ['rate<0.02'], // Less than 2% errors
    http_req_duration: ['p(95)<2000'], // 95% of requests < 2s
  },
};

const endpoints = [
  '/user',
  '/userRatings',
  '/matchResults',
  '/activeMatches',
  '/friends',
];
```

Hình 5.10 Kịch bản kiểm thử 10000 người dùng

Hình 5.10 mô tả kịch bản kiểm thử hiệu năng với 10000 người dùng đồng thời, được thiết kế kéo dài trong 20 phút và chia thành 3 giai đoạn chính:

- Giai đoạn 1 (5 phút đầu): Tăng dần số lượng người dùng từ 0 lên 10000, nhằm mô phỏng quá trình hệ thống bắt đầu nhận tải và dần đạt mức tải tối đa.
- Giai đoạn 2 (10 phút tiếp theo): Duy trì ổn định 10000 người dùng đồng thời để đánh giá khả năng xử lý liên tục của hệ thống dưới áp lực cao.
- Giai đoạn 3 (5 phút cuối): Giảm dần số lượng người dùng từ 10000 về 0, mô phỏng tình huống người dùng rời khỏi hệ thống.



Hình 5.11 Kiểm thử tải với 10000 người dùng

Kết quả kiểm thử cho thấy hệ thống đã xử lý 879.864 yêu cầu, trong đó có 3.246 yêu cầu không thành công, tương đương tỷ lệ lỗi khoảng 0.36%, cho thấy hệ thống vẫn duy trì ổn định dưới tải lớn.

Thông lượng cực đại đạt mức 989.87 yêu cầu/giây, trong khi thời gian phản hồi P95 là 227ms, cho thấy độ trễ phản hồi vẫn được giữ ổn định khi đã mở rộng.

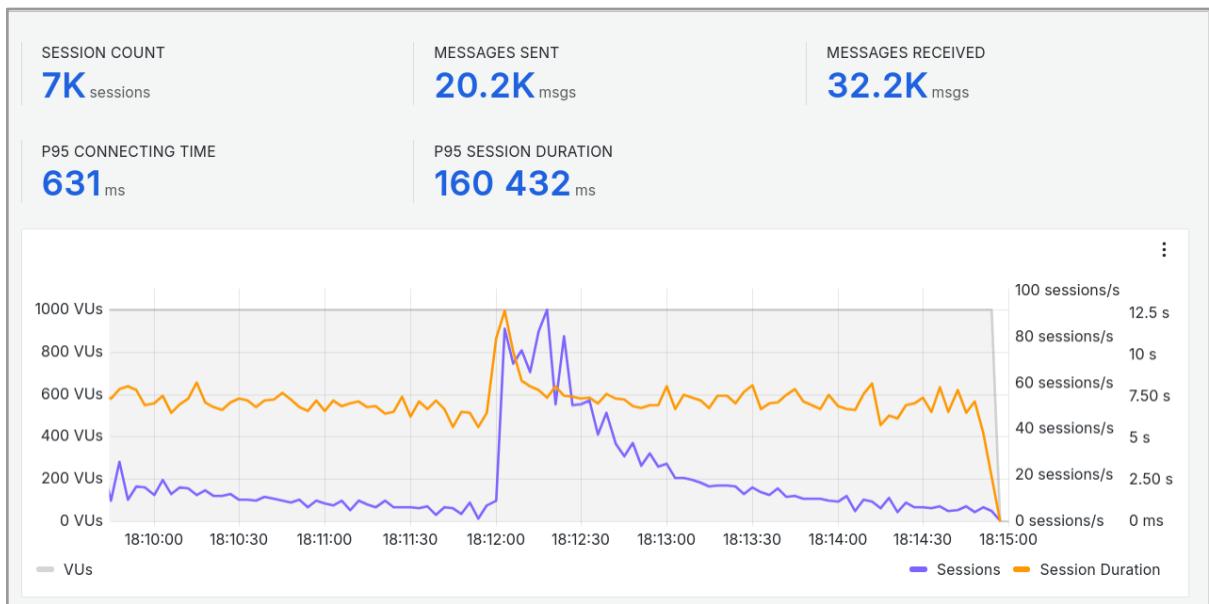
5.2.2.2 Kiểm thử hiệu suất máy chủ trò chơi

Kịch bản kiểm thử được xây dựng nhằm mô phỏng tương tác thời gian thực của 1000 người chơi đồng thời trên một máy chủ trò chơi thuộc nền tảng cờ vua trực tuyến đã triển khai. Những người dùng ảo sẽ được ghép thành từng cặp đấu với nhau. Sau khi kết nối và đợi máy chủ tạo trận đấu, hai người chơi luân phiên thực hiện các nước đi trong một ván cờ có kịch bản định sẵn.

Sau mỗi lần gửi nước đi, thời điểm gửi được lưu lại và khi nhận được phản hồi từ server, độ trễ được tính toán dựa trên chênh lệch thời gian. Các độ trễ này được lưu trữ

bằng metric 'move_to_gamestate_latency' của k6, cho phép hệ thống thống kê các giá trị như độ trễ trung bình, cực trị và phân佈 phân vị.

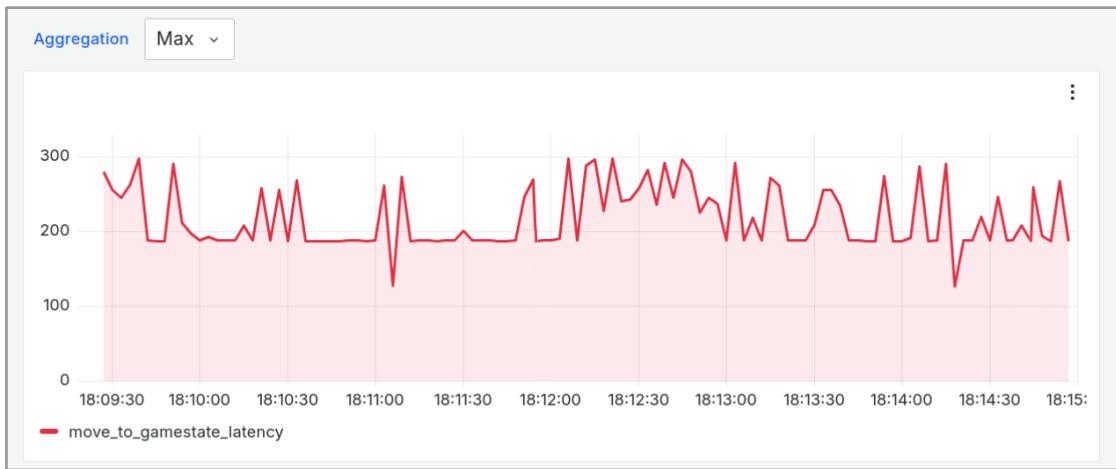
Ngoài ra, để mô phỏng hành vi thực tế của người chơi, mỗi người dùng ảo sẽ chờ từ 0.5 đến 2 giây trước khi gửi nước đi tiếp theo, và nghỉ từ 1 đến 3 giây sau khi kết thúc một ván để bắt đầu ván đấu mới. Kịch bản này giúp đánh giá khả năng xử lý kết nối WebSocket đồng thời, tốc độ phản hồi của server, và hiệu suất tổng thể của hệ thống backend trong môi trường tải cao, từ đó phục vụ cho việc tối ưu kiến trúc hạ tầng và cải thiện trải nghiệm người dùng trong các trò chơi trực tuyến.



Hình 5.12 Kiểm thử tải máy chủ trò chơi với 1000 người chơi đồng thời

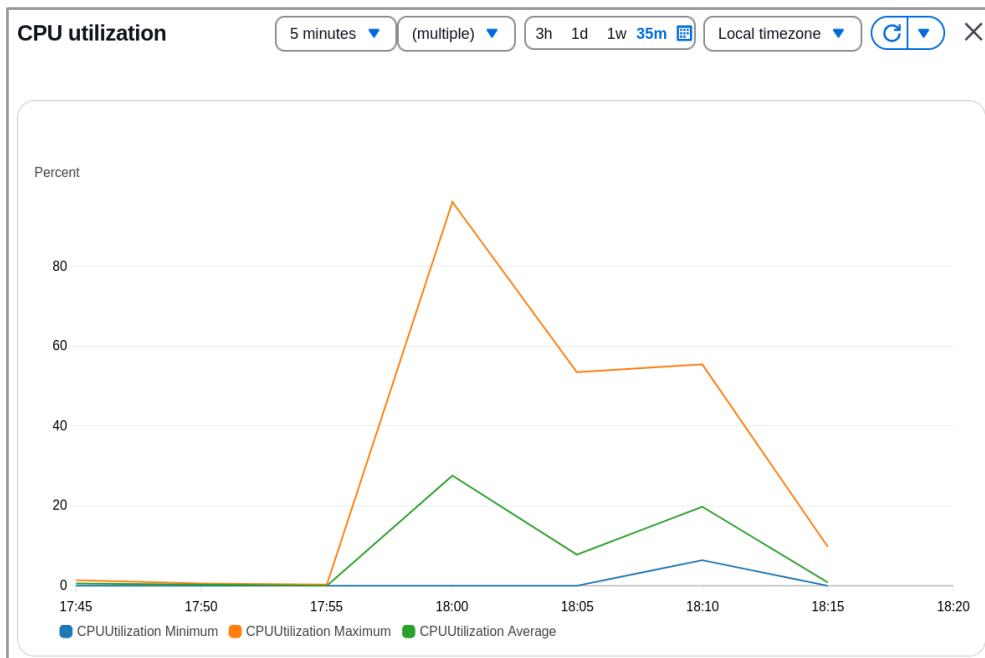
Hình 5.12 thể hiện kết quả kiểm thử tải máy chủ trò chơi khi phục vụ đồng thời 1000 người chơi. Trong quá trình kiểm thử, hệ thống ghi nhận khoảng 7.000 phiên kết nối WebSocket tương đương với khoảng 3500 trận đấu đã diễn ra, với hơn 20.000 tin nhắn được gửi và hơn 32.000 tin nhắn được nhận.

Thời gian trung bình để thiết lập kết nối và đợi tạo trận đấu là 631 mili giây và độ dài trung bình của mỗi phiên là khoảng 160 giây. Biểu đồ cho thấy số lượng người dùng đồng thời duy trì ổn định trong phần lớn thời gian kiểm thử, với một giai đoạn đỉnh điểm ngắn khi số lượng phiên tăng vọt rồi nhanh chóng ổn định trở lại. Điều này cho thấy hệ thống có khả năng tiếp nhận và xử lý tải lớn trong thời gian ngắn mà không xảy ra hiện tượng sụp đổ hoặc gián đoạn kết nối.



Hình 5.13 Độ trễ đồng bộ trạng thái của người chơi

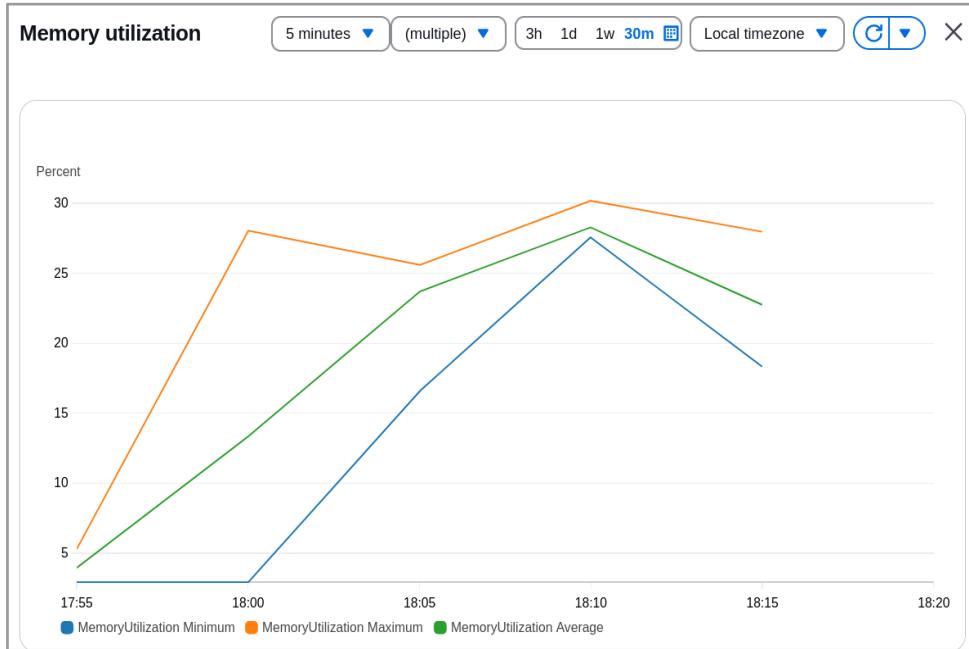
Hình 5.13 biểu diễn độ trễ đồng bộ trạng thái của người chơi thông qua chỉ số 'move_to_gamestate_latency'. Biểu đồ thể hiện giá trị độ trễ tối đa tại các thời điểm khác nhau trong suốt quá trình kiểm thử, phần lớn dao động từ 150 đến gần 300 mili giây. Kết quả này cho thấy phần lớn phản hồi trạng thái từ phía máy chủ diễn ra với độ trễ thấp và ổn định, đáp ứng được yêu cầu thời gian thực của trò chơi trong điều kiện tải cao.



Hình 5.14 Mức sử dụng CPU

Hình 5.14 thể hiện mức sử dụng CPU của máy chủ trong suốt quá trình kiểm thử tải. Tại thời điểm cao điểm, CPU đạt mức sử dụng tối đa 96%, trong khi mức trung bình dao động khoảng 30–40%. Trước và sau giai đoạn kiểm thử, mức sử dụng CPU giảm

mạnh về gần 0%, cho thấy CPU chỉ hoạt động cao trong thời gian máy chủ chịu tải. Biểu đồ này phản ánh rõ ràng khả năng mở rộng tài nguyên của hệ thống, đồng thời cho thấy rằng server có đủ khả năng xử lý đồng thời một lượng lớn kết nối khi cần thiết



Hình 5.15 Mức sử dụng bộ nhớ

Hình 5.15 trình bày mức sử dụng bộ nhớ của máy chủ trong cùng khoảng thời gian. Mức sử dụng bộ nhớ tăng dần từ khoảng 5% lên đến gần 30% khi tải hệ thống tăng lên, sau đó giảm nhẹ khi các phiên kết thúc. Trung bình bộ nhớ sử dụng trong giai đoạn cao điểm dao động quanh mức 25%, cho thấy hệ thống không gặp tình trạng tràn bộ nhớ và vẫn giữ được sự ổn định. Cả ba đường (giá trị tối thiểu, trung bình và tối đa) đều có xu hướng đồng nhất, chứng tỏ sự phân phối bộ nhớ giữa các tiến trình tương đối cân bằng.

5.2.2.3 Kiểm thử khả năng mở rộng của máy chủ trò chơi

```
export const options = {
  stages: [
    { duration: '30s', target: 200 }, // ramp to 200 VUs
    { duration: '30s', target: 400 }, // ramp to 400 VUs
    { duration: '30s', target: 600 }, // ramp to 600 VUs
    { duration: '30s', target: 800 }, // ramp to 800 VUs
    { duration: '30s', target: 1000 }, // ramp to 1000 VUs
    { duration: '30s', target: 1000 }, // hold at 1000 VUs
    { duration: '0s', target: 0 }, // gradually scale down
  ],
};

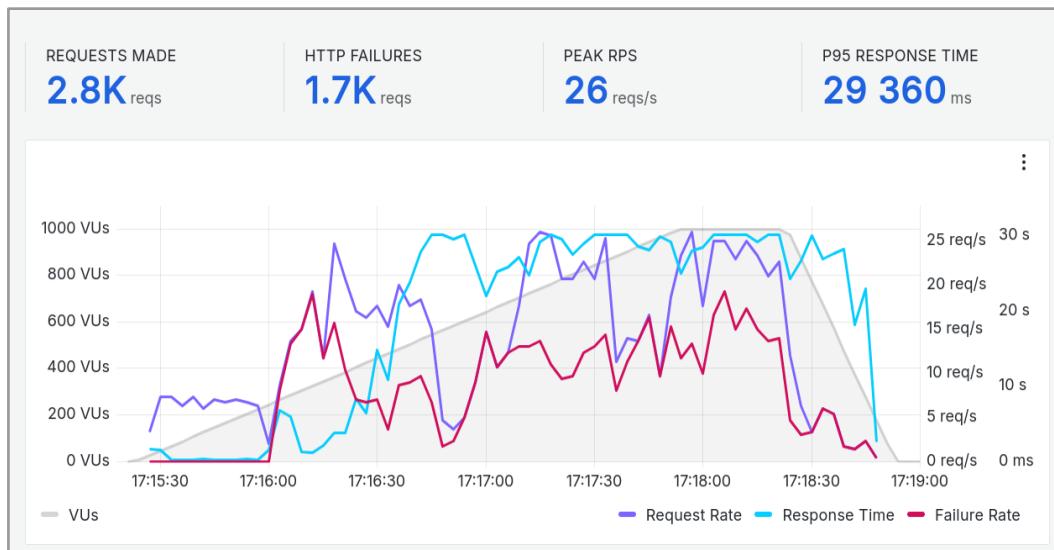
const MATCHMAKING_API = `${URL}/matchmaking`;
```

Hình 5.16 Kịch bản kiểm thử khả năng mở rộng của máy chủ trò chơi

Kịch bản này mô phỏng 1.000 người dùng ảo thực hiện quá trình ghép trận trong vòng 3 phút. Cụ thể, kể từ thời điểm bắt đầu kiểm thử, sau mỗi 30 giây, hệ thống sẽ tiếp nhận thêm 200 người dùng ảo cho đến khi đạt tổng cộng 1.000 người chơi, tương ứng với 500 trận đấu được yêu cầu khởi tạo.

Để kích hoạt cơ chế mở rộng một cách rõ ràng, hệ thống máy chủ trò chơi được cấu hình chỉ cho phép tổ chức tối đa 100 trận đấu đồng thời trên mỗi máy chủ, tương đương với 200 kết nối WebSocket tại một thời điểm. Với thiết lập giới hạn này, hệ thống buộc phải mở rộng quy mô hạ tầng trong thời gian ngắn để kịp thời phục vụ sự gia tăng đột ngột về số lượng người chơi.

Kịch bản kiểm thử trên giúp đánh giá khả năng mở rộng ngang (horizontal scaling) của hệ thống và mức độ hiệu quả của các cơ chế phân phối tài nguyên, bao gồm hàng đợi ghép trận, phân phối trận đấu, và kiểm soát tải trên tầng kết nối WebSocket.



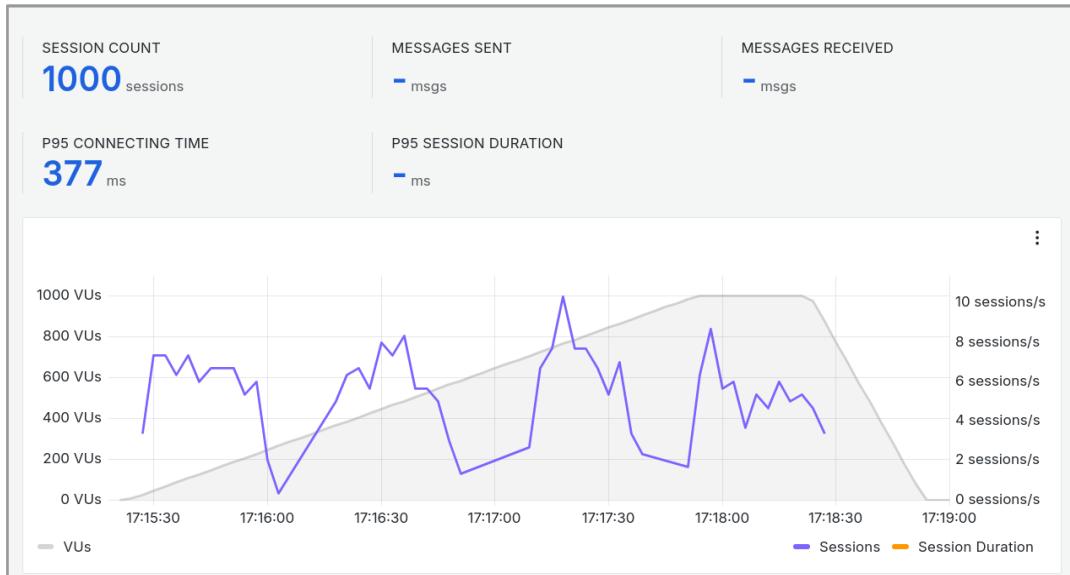
Hình 5.17 Kết quả gửi yêu cầu ghép trận

Hình 5.17 minh họa quá trình người dùng gửi yêu cầu ghép trận trong kịch bản kiểm thử. Trong suốt quá trình, tổng cộng có 2.800 yêu cầu ghép trận được gửi đến hệ thống. Trong đó, có khoảng 1.700 yêu cầu phản hồi không thành công (trạng thái HTTP 204 hoặc lỗi), chiếm một tỷ lệ lớn.

Tuy nhiên, kết quả này hoàn toàn nằm trong dự đoán. Hệ thống được thiết lập giới hạn số lượng trận đấu hoạt động đồng thời để đảm bảo ổn định cho tầng máy chủ trò chơi. Khi số lượng yêu cầu vượt quá giới hạn này, hệ thống sẽ phản hồi với mã 204 (chờ ghép trận) hoặc lỗi tạm thời – đây là một cơ chế kiểm soát tải chủ động. Nhờ vậy, các trận đấu đang diễn ra không bị ảnh hưởng, và hệ thống có thời gian để mở rộng thêm tài nguyên phục vụ các yêu cầu tiếp theo.

Để hỗ trợ điều này, phía người dùng có thể triển khai cơ chế retry – tức là gửi lại yêu cầu sau một khoảng thời gian chờ. Cách tiếp cận này cho phép hệ thống phục vụ khối lượng người dùng lớn vượt quá khả năng xử lý đồng thời, mà vẫn giữ được độ ổn định.

Thời gian ghép trận trung bình đạt mức tương đối cao là xấp xỉ 29 giây, phản ánh tình trạng chờ đợi trong giai đoạn hệ thống bị giới hạn về số lượng tài nguyên phục vụ.



Hình 5.18 Kết quả ghép trận thành công

Hình 5.18 thể hiện kết quả quá trình thiết lập kết nối thành công sau khi người chơi được ghép trận. Trong tổng số 1.000 người dùng, hệ thống đã thiết lập đủ 1.000 phiên kết nối WebSocket, chứng tỏ rằng mọi người chơi đều đã vào trận đấu thành công.

Trong suốt quá trình kiểm thử, hệ thống đã ghi nhận quá trình co giãn tài nguyên hoạt động đúng như kỳ vọng. Cụ thể, khi số lượng trận đấu đồng thời vượt ngưỡng cho phép trên mỗi máy chủ, hệ thống tự động khởi tạo thêm máy chủ mới trong vòng 30 giây.

Sau khi số lượng người chơi bắt đầu giảm và không còn nhiều yêu cầu mới, hệ thống bắt đầu thu hẹp tài nguyên tự động. Các máy chủ không còn trận đấu hoạt động sẽ bị đánh dấu và tự động dừng lại nhằm tiết kiệm chi phí vận hành. Trong giai đoạn tải đạt đỉnh, hệ thống đã mở rộng từ 1 lên 5 máy chủ trong vòng 2 phút, sau đó giảm về 1 máy chủ khi toàn bộ trận đấu kết thúc.

Mặc dù cơ chế ghép trận chưa được tối ưu hoàn toàn về tốc độ và độ trễ, hệ thống vẫn cho thấy năng lực mở rộng hiệu quả. Quá trình xử lý gia tăng đột ngột về lưu lượng

người chơi đã được giải quyết thông qua việc mở rộng hạ tầng và kiểm soát kết nối hợp lý. Điều này cho thấy hệ thống hoàn toàn có khả năng đáp ứng với các tình huống tải cao trong môi trường thực tế.

5.2.3 Ước tính chi phí bằng AWS Pricing Calculator

Amazon API Gateway

Bảng 5.1 Chi phí HTTP API

Số lượng user hoạt động (người/ngày)	500	5000	50000
Số lượng yêu cầu (mỗi ngày)	10,000	100,000	1,000,000
Kích cỡ yêu cầu trung bình (KB)	128	128	128
Chi phí (USD/tháng)	0.39	3.92	39.24

Bảng 5.2 Chi phí Websocket API

Số lượng user hoạt động (người/ngày)	500	5000	50000
Số lượng tin nhắn gửi trong một kết nối (tin nhắn/phút)		5	
Kích cỡ tin nhắn trung bình (KB)		128	
Thời lượng kết nối trung bình (giây)		15	
Số lượng kết nối (kết nối/ngày)	5,000	50,000	500,000
Chi phí (USD/tháng)	1.15	1.26	2.38

AWS Lambda

Bảng 5.3 Chi phí AWS Lambda

Số lượng user hoạt động (người/ngày)	500	5000	50000
Số lần gọi (mỗi ngày)	50,000	500,000	5,000,000
Thời gian xử lý (ms)	300		
Bộ nhớ (mb)	128		
Lưu trữ tạm thời (MB)	512		
Chi phí (USD/tháng)	1.25	12.55	125.47

AWS Cognito

Bảng 5.4 Chi phí AWS Cognito

Số lượng người dùng độc nhất (người/tháng)	2000	20,000	200,000
Chi phí (USD/tháng)	0.0	55.00	955.00

Amazon DynamoDB

Bảng 5.5 Chi phí Amazon DynamoDB

Số lượng user hoạt động (người/ngày)	500	5000	50000
Lưu trữ dữ liệu	Khối lượng lưu trữ (GB)	5	50
	Kích cỡ item trung bình (KB)	1	
Số lượng yêu cầu ghi (triệu/tháng)	2	20	200
Số lượng yêu cầu đọc	0.75	7.5	75

(triệu/tháng)			
Chi phí (USD/tháng)	2.90	28.98	289.84

Amazon Simple Storage Service

Bảng 5.6 Chi phí Amazon S3

Số lượng user hoạt động (người/ngày)	500	5000	50000
Ước tính khói lượng lưu trữ (GB/tháng)	20	200	2,000
Số lượng yêu cầu ghi	100,000	1,000,000	10,000,000
Số lượng yêu cầu đọc	500,000	5,000,000	50,000,000
Chi phí (USD/tháng)	1.27	12.70	127.00

AWS AppSync GraphQL Real-Time

Bảng 5.7 Chi phí AWS AppSync Real-Time

Số lượng user hoạt động (người/ngày)	500	5000	50000
Số lượng subscriber (người)	500	5000	50000
Tổng thời gian kết nối trung bình (giờ/tháng)	30,000	300,000	3,000,000
Số lượng tin nhắn inbound (tin nhắn/ngày)	35,000	350,000	3,500,000
Số lượng tin nhắn outbound (tin nhắn/ngày)	70,000	700,000	7,000,000
Chi phí (USD/tháng)	8.67	86.62	866.62

AWS SNS

Bảng 5.8 Chi phí AWS SNS

Số lượng user hoạt động (người/ngày)	500	5000	50000
Số lượng yêu cầu (triệu/tháng)	0.05	0.5	5
Số lượng thông báo đầy cho mobile (triệu/tháng)	0.25	2.5	25
Số lượng thông báo đầy cho HTTP (triệu/tháng)	0.25	2.5	25
Chi phí (USD/tháng)	0	2.14	29.44

AWS Fargate

Bảng 5.9 Chi phí AWS Fargate

Số lượng user hoạt động (người/ngày)	500	5000	50000
Số lượng người chơi đồng thời (người/ngày)	100	1,000	10,000
Số lượng máy chủ (máy/ngày)	1	10	100
Thời gian chạy máy chủ (giờ/ngày)		24	
Số lượng vCPU		0.25	
Bộ nhớ (GB)		0.5	
Chi phí (USD/tháng)	10.80	108.04	1,080.40

Tổng chi phí vận hành của hệ thống được ước tính dao động chỉ từ khoảng 50–100 USD/tháng khi lưu lượng thấp (dưới 1.000 người dùng/ngày) và lên đến khoảng 3.500–4.000 USD/tháng ở mức cao điểm với khoảng 50.000 người dùng hàng ngày và 10.000 người chơi đồng thời. Trong đó, ba dịch vụ chiếm tỷ trọng lớn nhất bao gồm AWS

Fargate (vận hành các máy chủ trò chơi), AppSync (duy trì kết nối thời gian thực), và DynamoDB (lưu trữ và truy xuất trạng thái).

Kiến trúc serverless giúp hệ thống chỉ tiêu tốn chi phí khi có người dùng, từ đó tối ưu hóa vận hành đáng kể so với các mô hình truyền thống luôn duy trì máy chủ hoạt động liên tục.

5.3 Kết quả thử nghiệm

Thời gian triển khai backend trò chơi

Phương pháp triển khai của hệ thống ảnh hưởng trực tiếp đến khả năng sẵn sàng, cấu hình hạ tầng và hiệu quả vận hành. Để đánh giá hiệu quả của giải pháp PaaS được đề xuất, sinh viên đã tiến hành thử nghiệm thực tế, so sánh thời gian triển khai backend trò chơi theo hai phương pháp: triển khai thủ công và triển khai tự động thông qua nền tảng PaaS.

Các số liệu dưới đây được tổng hợp từ quá trình thực nghiệm trong khuôn khổ đề tài. Dù không đại diện cho toàn bộ các trường hợp trong ngành công nghiệp game, chúng phản ánh tương đối rõ sự chênh lệch về thời gian hoàn thành giữa hai phương pháp, đặc biệt trong bối cảnh các hệ thống hiện đại yêu cầu tính linh hoạt và tốc độ triển khai cao.

Bảng 5.10 So sánh thời gian triển khai backend trò chơi

Phương pháp triển khai	Thời gian trung bình	Ghi chú
Triển khai thủ công	~1-2 tuần (ước lượng)	Phát triển API, kết nối cơ sở dữ liệu, triển khai phương án scaling, backup.
Qua giải pháp PaaS	~30 phút	Lựa chọn các dịch vụ theo nhu cầu. Đơn giản hóa việc cấu hình dịch vụ.

Kết quả cho thấy, việc triển khai backend qua giải pháp PaaS giúp rút ngắn đáng kể thời gian chuẩn bị hệ thống – từ nhiều ngày xuống chỉ còn khoảng nửa giờ, đồng thời loại bỏ nhiều bước cấu hình lặp lại. Đây là một lợi thế quan trọng đối với các nhà phát triển cần nhanh chóng kiểm thử, cập nhật hoặc phát hành trò chơi mới trong môi trường thực tế.

Hiệu suất hệ thống và khả năng mở rộng

Khả năng mở rộng tự động là một trong những tính năng nổi bật của kiến trúc serverless. Khi tải người dùng tăng lên, hệ thống tự động mở rộng mà không cần sự can thiệp thủ công, giúp duy trì chất lượng dịch vụ và giảm thiểu downtime. Các thử nghiệm về khả năng mở rộng dưới các mức tải người dùng khác nhau đã được thực hiện để đánh giá độ trễ và hiệu suất của hệ thống.

Bảng 5.11 Hiệu suất API

Số lượng người dùng đồng thời	Độ trễ trung bình (ms)	Ghi chú
100	209	Hệ thống phản hồi nhanh, phù hợp cho thử nghiệm chức năng.
1.000	216	Hệ thống duy trì hiệu suất ổn định với tải vừa phải.
10.000	227	Tăng nhẹ độ trễ, cho thấy khả năng mở rộng tốt dưới tải lớn.

Giải pháp serverless đã chứng minh khả năng duy trì hiệu suất ổn định với độ trễ thấp ngay cả khi tải tăng lên. Hệ thống có thể xử lý hàng nghìn người dùng đồng thời mà không gặp phải tình trạng nghẽn cổ chai, đảm bảo trải nghiệm người chơi mượt mà. Mặc dù độ trễ tăng lên khi tải cao, nhưng sự gia tăng này vẫn nằm trong ngưỡng chấp nhận được đối với các trò chơi trực tuyến.

Đáng chú ý hơn, trong thử nghiệm với 1.000 người chơi đồng thời, chỉ một máy chủ trò chơi duy nhất đã đủ khả năng xử lý toàn bộ khối lượng công việc mà không bị gián đoạn. Mức sử dụng CPU cao nhất chỉ đạt khoảng 96%, cho thấy hệ thống đã khai thác hiệu quả tài nguyên mà vẫn duy trì được tính ổn định. Đồng thời, độ trễ khi đồng bộ trạng thái giữa các người chơi – một yếu tố quan trọng trong các trò chơi thời gian thực – duy trì ở mức trung bình từ 150 đến 300 mili giây. Kết quả này cho thấy hệ thống không chỉ xử lý tốt khối lượng kết nối lớn, mà còn đảm bảo độ mượt trong trải nghiệm người chơi ở cả khía cạnh phản hồi và đồng bộ trạng thái trận đấu.

Ngoài hiệu suất ổn định, kết quả thử nghiệm còn khẳng định hiệu quả của chiến lược mở rộng máy chủ trò chơi. Khi 1.000 người dùng ảo thực hiện ghép trận liên tục trong thời gian ngắn, hệ thống vẫn đảm bảo mọi người chơi đều được ghép thành công

và trận đấu được phân bổ hợp lý cho các máy chủ. Việc khởi tạo các máy chủ trò chơi mới cũng được thực hiện nhanh chóng, hỗ trợ tốt cho quá trình mở rộng tức thì. Với khả năng phục vụ nhiều trận đấu trên mỗi máy chủ, hệ thống hoàn toàn có thể trì hoãn việc mở rộng khi chưa thật sự cần thiết, qua đó hạn chế tối đa tác động đến trải nghiệm người dùng và tối ưu chi phí vận hành.

Chi phí vận hành

Tối ưu hóa chi phí là một yếu tố quan trọng trong việc duy trì hệ thống lâu dài, đặc biệt đối với các trò chơi có lượng người chơi không ổn định. So với các mô hình truyền thống sử dụng EC2, kiến trúc serverless mang lại những lợi ích rõ rệt trong việc giảm chi phí vận hành, đặc biệt trong những giai đoạn không có người chơi.

Chi phí vận hành cho một nền tảng trò chơi trực tuyến trong vòng một tháng được ước tính như sau.

Bảng 5.12 Chi phí vận hành

Số lượng người dùng hàng ngày	Số lượng người chơi trung bình ở mọi thời điểm	Chi phí hàng tháng	
		EC2	Serverless
500	100	~\$120	~\$27
5,000	1,000	~\$900	~\$310
50,000	10,000	~\$6.000	~\$3515

Chi phí EC2 được tính dựa trên việc duy trì các máy chủ hoạt động 24/7 để đảm bảo độ sẵn sàng, bao gồm chi phí cho instance, lưu trữ, băng thông và dữ phòng hệ thống. Điều này đồng nghĩa với việc hệ thống vẫn phải trả phí ngay cả trong những khung giờ không có người chơi truy cập. Ví dụ, để phục vụ khoảng 10.000 người chơi đồng thời, hệ thống cần duy trì hàng chục instance EC2 loại trung bình (như t3.medium hoặc c5.large), dẫn đến chi phí có thể lên tới 6.000 USD/tháng.

Ngược lại, kiến trúc serverless chỉ tính phí dựa trên lệnh gọi API, thời gian duy trì kết nối, lượng dữ liệu truy vấn/ghi, và thời gian thực thi các hàm Lambda hoặc container. Các dịch vụ như AWS Lambda, Fargate, AppSync, DynamoDB và API Gateway có thể tự động mở rộng hoặc thu hẹp theo tải, giúp tiết kiệm chi phí đáng kể trong các thời điểm tải thấp hoặc vắng người chơi.

Tổng thể, hệ thống sử dụng serverless giúp giảm từ 50–80% chi phí so với EC2 ở cả mức tải thấp lẫn cao. Điều này đặc biệt phù hợp cho các trò chơi trực tuyến có thời điểm cao điểm rõ rệt (ví dụ: buổi tối, cuối tuần), hoặc các trò chơi theo mùa. Với mô hình pay-as-you-go, chi phí vận hành được tối ưu mà vẫn đảm bảo khả năng mở rộng tức thì và độ ổn định cao trong suốt thời gian hoạt động.

5.4 Đánh giá

Giải pháp PaaS được xây dựng nhằm triển khai backend cho trò chơi dựa trên kiến trúc serverless đã chứng minh tính khả thi và hiệu quả trong việc cung cấp nền tảng phát triển backend cho các trò chơi trực tuyến. Với mục tiêu tối ưu hóa về hiệu suất, khả năng mở rộng và chi phí, hệ thống đã thực hiện thành công nhiệm vụ cung cấp các dịch vụ linh hoạt và hiệu quả, đặc biệt trong việc xử lý yêu cầu ở nhiều mức tải người dùng khác nhau.

Thông qua các thử nghiệm thực tế, hệ thống cho thấy khả năng duy trì chất lượng trải nghiệm người chơi ổn định, kể cả khi số lượng người dùng tăng đột ngột từ 100 đến 10.000 người dùng đồng thời. Độ trễ trung bình trong các thử nghiệm dao động từ 209ms đến 227ms, thể hiện khả năng giữ ổn định thời gian phản hồi, đồng thời không phát sinh tình trạng nghẽn cổ chai.

Các kịch bản kiểm thử cũng ghi nhận thời gian phản hồi trung bình của API dao động từ 200–250ms, chứng minh rằng các dịch vụ con hoạt động ổn định ngay cả dưới tải cao. Đặc biệt, độ trễ khi đồng bộ trạng thái giữa các người chơi – yếu tố quan trọng trong trò chơi thời gian thực – luôn được duy trì trong khoảng 150–300ms, đảm bảo sự đồng bộ và nhất quán giữa các bên tham gia trận đấu.

Một điểm nổi bật khác là hệ thống đã xử lý hiệu quả quá trình ghép trận với 1.000 người chơi trong thời gian ngắn, cho thấy khả năng phản hồi ổn định ngay cả khi tải tăng theo từng đợt. Cơ chế kiểm soát tải chủ động kết hợp với chiến lược retry phía client giúp đảm bảo mọi người chơi đều được ghép trận thành công mà không xảy ra gián đoạn. Đồng thời, hệ thống mở rộng tài nguyên đúng thời điểm, tận dụng tốt tài nguyên hiện có để duy trì hiệu suất ổn định và tối ưu chi phí.

Về chi phí, mô hình serverless mang lại lợi thế rõ rệt nhờ cơ chế pay-as-you-go. Trong các thời điểm không có người chơi, hệ thống hầu như không phát sinh chi phí. Ngay cả khi tải cao, chi phí vẫn thấp hơn đáng kể so với các mô hình truyền thống như EC2 hoặc Fargate – vốn yêu cầu duy trì máy chủ thường trực. Điều này giúp hệ thống

linh hoạt thích ứng với biến động lưu lượng người dùng, đặc biệt trong các trò chơi có tần suất truy cập không ổn định.

Không chỉ tối ưu về hiệu suất và chi phí, hệ thống còn cho thấy tính linh hoạt cao trong cấu hình và khả năng tái sử dụng các dịch vụ backend. Các module như xác thực người dùng, ghép trận, xếp hạng, trò chuyện, kết bạn... đều có thể cấu hình lại dễ dàng theo đặc thù của từng trò chơi. Điều này cho phép các nhà phát triển tùy chỉnh các tham số như thuật toán xếp hạng, giới hạn người chơi hoặc thời gian chờ – mà không cần xây dựng lại toàn bộ dịch vụ từ đầu.

Tổng kết lại, giải pháp PaaS này đã đạt được các mục tiêu cốt lõi về hiệu suất, khả năng mở rộng và tối ưu chi phí. Hệ thống vừa đảm bảo trải nghiệm người chơi ổn định, vừa hỗ trợ mở rộng linh hoạt và tiết kiệm tài nguyên. Với khả năng cấu hình tùy biến và tái sử dụng dịch vụ, đây là một nền tảng mạnh mẽ, phù hợp cho các đội ngũ phát triển muốn tập trung vào logic trò chơi mà không phải lo ngại về hạ tầng backend phức tạp.

5.5 So sánh với các giải pháp hiện có

Để đánh giá khách quan tính hiệu quả của giải pháp đề tài, mục này tiến hành so sánh với ba nền tảng backend trò chơi phổ biến hiện nay: PlayFab, Nakama, và AWS GameLift. Đây đều là những nền tảng có khả năng triển khai backend trò chơi ở quy mô lớn, hỗ trợ đa người chơi và vận hành thời gian thực.

Các tiêu chí được sử dụng để so sánh là:

- Mô hình triển khai: Serverless, self-hosted, managed service...
- Mức độ kiểm soát và tùy biến: Khả năng thiết kế, chỉnh sửa hệ thống backend.
- Khả năng mở rộng: Khả năng đáp ứng nhu cầu người dùng tăng cao.
- Chi phí vận hành: Khả năng tối ưu hóa chi phí sử dụng trong kịch bản có lượng người dùng lớn.
- Hỗ trợ real-time multiplayer: Khả năng vận hành game yêu cầu đồng bộ thời gian thực.

Bảng 5.13 So sánh giải pháp đề tài với các nền tảng triển khai khác

Tiêu chí	Giải pháp đề tài	AWS GameLift	PlayFab	Nakama
Mô hình triển khai	Serverless	VM/Server truyền thống	SaaS/BaaS	Tự host VM

Kiểm soát & tùy biến	Cao (tùy chỉnh hạ tầng và dịch vụ)	Trung bình (giới hạn API AWS)	Thấp (giới hạn API PlayFab)	Cao (mã nguồn mở)
Khả năng mở rộng	Rất cao (tự động theo nhu cầu)	Cao (tự động nhưng tốn tài nguyên)	Có nhưng phụ thuộc dịch vụ tích hợp	Giới hạn (phụ thuộc cấu hình tự triển khai)
Chi phí khi nhàn rỗi	Thấp (gần như không tốn chi phí)	Cao (luôn chạy máy chủ)	Trung bình (có phí duy trì)	Trung bình đến cao (duy trì máy chủ)
Chi phí vận hành hàng tháng (10k CCU)	~3400-3700 USD	~4700-5000 USD	~3500–3700 USD	~2200–4500 USD
Độ phức tạp triển khai	Trung bình	Cao	Thấp	Cao
Hiệu suất realtime	Tốt (WebSocket, AppSync, Fargate)	Rất tốt (máy chủ vật lý)	Trung bình	Tốt (tùy cấu hình)

Bảng 5.13 so sánh các giải pháp triển khai backend trò chơi dựa trên những tiêu chí đã đề ra. Đặc điểm của các giải pháp được phân tích chi tiết như sau:

- **PlayFab:** Là một dịch vụ BaaS mạnh mẽ trong quản lý người chơi, authentication và dữ liệu. Tuy nhiên, PlayFab không cung cấp trực tiếp gameplay server cho các trò chơi real-time. Để vận hành game thời gian thực, nhà phát triển cần tích hợp thêm Azure Multiplayer Servers hoặc tự xây dựng server riêng, dẫn đến chi phí và độ phức tạp tăng cao.
- **Nakama:** Là giải pháp mã nguồn mở có cộng đồng lớn và hỗ trợ nhiều tính năng mạnh như multiplayer real-time, ranking, storage. Tuy nhiên, việc triển khai Nakama yêu cầu đội ngũ kỹ thuật có kinh nghiệm vận hành server (kể cả high availability, backup, auto scaling). Việc mở rộng quy mô sẽ gặp rủi ro nếu không có năng lực DevOps mạnh.
- **AWS GameLift:** Là dịch vụ chuyên biệt cho hosting game server với khả năng auto-scaling, matchmaking (FlexMatch), tích hợp bảo vệ DDoS. Phù hợp với các game real-time nặng như MOBA, FPS. Nhược điểm là chi phí cao và khó tích hợp nếu backend không nằm trong AWS.

- **Giải pháp đề tài:** Với việc sử dụng kiến trúc serverless và triển khai tài nguyên dưới dạng code, hệ thống có khả năng:

- Tự động mở rộng linh hoạt.
- Tối ưu hóa chi phí với mô hình pay-per-use.
- Cho phép nhà phát triển kiểm soát và tùy chỉnh mọi thành phần backend.

Chi phí ước tính trong bảng dựa trên hệ thống mẫu triển khai cho một nền tảng chơi cờ vua trực tuyến, với giả định có trung bình 10.000 người chơi hoạt động đồng thời (CCU) ở mọi thời điểm. Các chỉ số chi phí bao gồm vận hành toàn bộ backend, hạ tầng máy chủ trò chơi, chi phí lưu trữ và mạng. Với các thể loại game nặng hơn hoặc có yêu cầu real-time cao hơn, chi phí thực tế có thể thay đổi đáng kể.

Qua quá trình so sánh, có thể thấy giải pháp đề tài cung cấp sự cân bằng tối ưu giữa khả năng kiểm soát, khả năng mở rộng và tối ưu chi phí so với các nền tảng hiện có.

Chương 6: Kết luận, tổng kết

6.1 Kết luận

Đồ án đã đề xuất và hiện thực hóa một giải pháp PaaS hỗ trợ triển khai backend trò chơi dựa trên kiến trúc serverless, nhằm giải quyết các bài toán thực tiễn mà các nhà phát triển game thường gặp phải như chi phí vận hành cao, thời gian triển khai kéo dài và khó khăn trong việc mở rộng hệ thống theo tải thực tế.

Nền tảng được thiết kế theo hướng mô-đun hóa và tái sử dụng, cung cấp sẵn các dịch vụ backend tích hợp như ghép trận, xếp hạng, xác thực người dùng, trò chuyện và xem trận đấu – tất cả đều có thể cấu hình linh hoạt theo nhu cầu của từng tựa game. Bên cạnh đó, hệ thống hỗ trợ triển khai các máy chủ trò chơi tùy chỉnh dưới dạng container thông qua AWS Fargate, cho phép chạy logic trò chơi độc lập và mở rộng theo phiên đấu thực tế.

Toàn bộ giải pháp được xây dựng trên nền tảng hạ tầng cloud-native hiện đại, tận dụng sâu các dịch vụ serverless trong hệ sinh thái AWS như Lambda, API Gateway, AppSync, DynamoDB, SQS, SNS, Cognito, và Fargate để đạt được khả năng mở rộng tự động, giảm thiểu chi phí vận hành và đơn giản hóa quy trình triển khai. Hệ thống còn tích hợp khả năng giám sát, thu thập log và phản ứng theo sự kiện thông qua CloudWatch và EventBridge, giúp nâng cao khả năng vận hành ổn định trong môi trường sản xuất thực tế.

Một điểm nổi bật của giải pháp là khả năng tái sử dụng và mở rộng cho nhiều tựa game khác nhau. Nền tảng cho phép nhà phát triển lựa chọn dịch vụ phù hợp, cấu hình backend và triển khai tự động mà không cần kiến thức chuyên sâu về hạ tầng. Ngoài ra, nền tảng hỗ trợ sẵn SDK mẫu giúp việc tích hợp giữa backend và máy chủ trò chơi trở nên dễ dàng và thống nhất hơn.

Kết quả kiểm thử trên hệ thống trò chơi mẫu (cờ vua) đã chứng minh tính khả thi và hiệu quả của giải pháp: hệ thống có thể phục vụ lên đến 10.000 người chơi đồng thời với độ trễ thấp, hiệu suất ổn định và chi phí hợp lý. Các dịch vụ backend và máy chủ trò chơi có thể mở rộng độc lập theo nhu cầu thực tế, mà không cần can thiệp thủ công – cho thấy tiềm năng ứng dụng rộng rãi trong thực tế.

Tuy nhiên, đồ án cũng ghi nhận một số giới hạn tự nhiên của kiến trúc serverless trong các trường hợp đặc thù, chẳng hạn như những trò chơi yêu cầu phản hồi tức thì (low-latency) như game hành động hoặc FPS. Trong các trường hợp này, cần xem xét

kết hợp kiến trúc với các thành phần truyền thống như máy chủ chuyên dụng hoặc hybrid architecture để đảm bảo hiệu năng và độ tin cậy.

Tổng kết lại, đồ án đã hoàn thành đầy đủ các mục tiêu đề ra: từ thiết kế kiến trúc, hiện thực nền tảng mẫu, xây dựng các dịch vụ backend tích hợp, cho đến triển khai và kiểm thử toàn diện. Giải pháp thể hiện được tính mở rộng linh hoạt, hiệu quả chi phí, khả năng tùy biến cao và tiềm năng tái sử dụng thực tiễn, mở ra hướng phát triển thành một sản phẩm thương mại hoặc nền tảng triển khai backend trò chơi quy mô lớn trong tương lai.

6.2 Hướng nghiên cứu trong tương lai

Sau khi xây dựng thành công giải pháp PaaS hỗ trợ triển khai backend trò chơi dựa trên kiến trúc serverless, nhiều hướng nghiên cứu và phát triển tiềm năng đã được xác định để tiếp tục nâng cao khả năng ứng dụng thực tiễn và mở rộng quy mô nền tảng. Dưới đây là một số định hướng cụ thể:

Tích hợp trí tuệ nhân tạo (AI/ML) vào hệ thống ghép trận

Hiện tại, các thuật toán xếp hạng như Elo hoặc Glicko được sử dụng như tùy chọn mặc định. Trong tương lai, có thể tích hợp AI/ML để cải thiện hiệu quả và độ thông minh của hệ thống ghép trận, cá nhân hóa trải nghiệm và học hỏi hành vi người chơi theo thời gian. Ví dụ, Amazon SageMaker có thể được sử dụng để huấn luyện mô hình dự đoán khả năng chiến thắng, hoặc tạo đối thủ ảo cân bằng cho người chơi mới.

Phân tích dữ liệu hành vi người chơi và tối ưu vận hành

Nền tảng có thể mở rộng theo hướng tích hợp các công cụ xử lý dữ liệu lớn (Big Data) để thu thập, phân tích và khai thác dữ liệu hành vi người chơi như lịch sử thi đấu, thời gian hoạt động, hành vi trong trận đấu hoặc nội dung trò chuyện. Các dịch vụ như Kinesis, Firehose và Athena có thể được sử dụng để hỗ trợ ra quyết định, phát hiện gian lận, hoặc gợi ý cải tiến gameplay.

Chuẩn hóa SDK và hỗ trợ đa nền tảng

Việc phát triển SDK cho các nền tảng game phổ biến như Unity, Unreal Engine, Godot... sẽ giúp mở rộng đối tượng sử dụng, đồng thời giảm công sức tích hợp backend cho các nhà phát triển độc lập. SDK cũng có thể cung cấp giao diện cấu hình dịch vụ trực quan, từ đó tạo thành một hệ sinh thái backend plug-and-play thực sự.

Hỗ trợ đa vùng địa lý và tối ưu cho realtime

Nền tảng có thể được mở rộng để hỗ trợ triển khai phân tán theo nhiều vùng địa lý nhằm giảm độ trễ và nâng cao tính sẵn sàng toàn cầu. Đồng thời, cần nghiên cứu các mô hình hybrid (kết hợp serverless và dedicated servers) để tối ưu cho các game yêu cầu thời gian thực cao, sử dụng các công nghệ như WebRTC, UDP hoặc relay server chuyên dụng cho các kịch bản realtime phức tạp.

Phụ lục

A - Mẫu template định nghĩa tài nguyên backend

```
AWSTemplateFormatVersion: 2010-09-09
Transform: AWS::Serverless-2016-10-31
Description: Root Stack for Slchess Backend

Parameters:
  ServerImageUri:
    Type: String
    Description: "URI of the Docker image for game server in ECR"
  DeploymentStage:
    Type: String
    Default: dev

Resources:
  StorageStack:
    Type: AWS::CloudFormation::Stack
    Properties:
      TemplateURL: ./templates/storage.yaml
      Parameters:
        StackName: !Ref AWS::StackName
        DeploymentStage: !Ref DeploymentStage

  AuthStack:
    Type: AWS::CloudFormation::Stack
    Properties:
      TemplateURL: ./templates/auth.yaml
      Parameters:
        StackName: !Ref AWS::StackName
        DeploymentStage: !Ref DeploymentStage
    DependsOn: StorageStack

  AppSyncStack:
    Type: AWS::CloudFormation::Stack
    Properties:
      TemplateURL: ./templates/appsync.yaml
      Parameters:
        StackName: !Ref AWS::StackName
        DeploymentStage: !Ref DeploymentStage
    DependsOn:
      - StorageStack
      - AuthStack
```

Hình A.1 Tập tin template.yaml (1)

```

ComputeStack:
  Type: AWS::CloudFormation::Stack
  Properties:
    TemplateURL: ./templates/compute.yaml
  Parameters:
    StackName: !Ref AWS::StackName
    ServerImageUri: !Ref ServerImageUri
    DeploymentStage: !Ref DeploymentStage
  DependsOn:
    - StorageStack
    - AuthStack
    - AppSyncStack

AnalysisStack:
  Type: AWS::CloudFormation::Stack
  Properties:
    TemplateURL: ./templates/analysis.yaml
  Parameters:
    StackName: !Ref AWS::StackName
    DeploymentStage: !Ref DeploymentStage

WebsocketApiStack:
  Type: AWS::CloudFormation::Stack
  Properties:
    TemplateURL: ./templates/websocketApi.yaml
  Parameters:
    StackName: !Ref AWS::StackName
    DeploymentStage: !Ref DeploymentStage
  DependsOn:
    - StorageStack
    - AuthStack
    - ComputeStack
    - AnalysisStack

HttpApiStack:
  Type: AWS::CloudFormation::Stack
  Properties:
    TemplateURL: ./templates/httpApi.yaml
  Parameters:
    StackName: !Ref AWS::StackName
    DeploymentStage: !Ref DeploymentStage
  DependsOn:
    - StorageStack
    - AuthStack
    - AppSyncStack
    - ComputeStack

```

Hình A.2 Tập tin template.yaml (2)

```
LogStack:  
  Type: AWS::CloudFormation::Stack  
  Properties:  
    TemplateURL: ./templates/log.yaml  
  Parameters:  
    StackName: !Ref AWS::StackName  
    DeploymentStage: !Ref DeploymentStage  
  
CustomizationStack:  
  Type: AWS::CloudFormation::Stack  
  Properties:  
    TemplateURL: ./templates/customization.yaml  
  Parameters:  
    StackName: !Ref AWS::StackName  
    DeploymentStage: !Ref DeploymentStage  
  DependsOn:  
    - StorageStack  
    - AuthStack  
    - AppSyncStack  
    - ComputeStack  
    - WebsocketApiStack  
    - HttpApiStack  
  
Outputs:
```

Hình A.3 Tập tin template.yaml (3)

B - Mẫu template tùy chỉnh backend

```
AWSTemplateFormatVersion: "2010-09-09"
Transform: AWS::Serverless-2016-10-31
Description: Customization Stack

Globals:
  Function:
    Timeout: 5
    MemorySize: 128
    Architectures:
      - arm64

Parameters:
  StackName:
    Type: String
  DeploymentStage:
    Type: String

Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: nodejs20.x
    Properties:
      FunctionName: !Sub "${StackName}-${DeploymentStage}-HelloWorldFunction"
      Runtime: nodejs20.x
      Handler: index.handler
      InlineCode: |
        exports.handler = async (event) => {
          return {
            statusCode: 200,
            body: "Hello World!",
          };
        };

```

Hình B.1 Tập tin customization.yaml (1)

```

HelloWorldIntegration:
  Type: AWS::ApiGatewayV2::Integration
  Properties:
    ApiId:
      Fn::ImportValue: !Sub "${StackName}-HttpApiId"
    IntegrationType: AWS_PROXY
    IntegrationUri: !Sub arn:aws:apigateway:${AWS::Region}:lambda:path/2015-
    PayloadFormatVersion: "2.0"

HelloWorldRoute:
  Type: AWS::ApiGatewayV2::Route
  Properties:
    ApiId:
      Fn::ImportValue: !Sub "${StackName}-HttpApiId"
    RouteKey: GET /helloWorld
    Target: !Sub
      - integrations/${IntegrationId}
      - IntegrationId: !Ref HelloWorldIntegration

HelloWorldLambdaPermission:
  Type: AWS::Lambda::Permission
  Properties:
    Action: lambda:InvokeFunction
    FunctionName: !GetAtt HelloWorldFunction.Arn
    Principal: apigateway.amazonaws.com
    SourceArn: !Sub
      - arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${HttpApiId}/*
      - HttpApiId:
          Fn::ImportValue: !Sub "${StackName}-HttpApiId"

Outputs:
  HelloWorldEndpointUrl:
    Value: !Sub
      - "GET ${HttpApiEndpoint}/helloWorld"
      - HttpApiEndpoint:
          Fn::ImportValue: !Sub "${StackName}-HttpApiEndpoint"

```

Hình B.2 Tập tin customization.yaml (2)

C - Kịch bản kiểm thử K6

Kiểm thử API

```
import { check } from 'k6';
import http from 'k6/http';
import { sleep } from 'k6';

// Configuration: Number of virtual users and duration of the test for 100 users
export const options = {
  stages: [
    { duration: '5m', target: 100 }, // Ramp up to 1000 users over 5 minutes
    { duration: '10m', target: 100 }, // Hold 1000 users for 10 minutes
    { duration: '5m', target: 0 }, // Gradually scale down to 0 users
  ],
  thresholds: {
    http_req_failed: ['rate<0.02'], // Less than 2% errors
    http_req_duration: ['p(95)<2000'], // 95% of requests < 2s
  },
};

const endpoints = [
  '/user',
  '/userRatings',
  '/matchResults',
  '/activeMatches',
  '/friends',
];
const baseUrl = 'https://33hiq368dh.execute-api.ap-southeast-2.amazonaws.com/dev';
const token = 'eyJraWQiOiJiNSt2S1A3QTF6SHdEMGNrSG1BZHJIV3dFaDlwbmF3K0JZZWhPZ21kTUxvPSIsImFsZyI

// List of endpoints to hit

export default function main() {
  const randomPath = endpoints[Math.floor(Math.random() * endpoints.length)];
  const url = baseUrl + randomPath;

  const headers = {
    'Authorization': `${token}`,
    'Content-Type': 'application/json',
  };

  const res = http.get(url, { headers });

  check(res, {
    'status is 200': (r) => r.status === 200,
    'response time < 500ms': (r) => r.timings.duration < 500,
  });

  sleep(10); // Simulate think time
}
```

Hình C.1 Kịch bản 100 người dùng API đồng thời

```

import { check } from 'k6';
import http from 'k6/http';
import { sleep } from 'k6';

// Configuration: Number of virtual users and duration of the test for 100 users
export const options = {
  stages: [
    { duration: '5m', target: 1000 }, // Ramp up to 1000 users over 5 minutes
    { duration: '10m', target: 1000 }, // Hold 1000 users for 10 minutes
    { duration: '5m', target: 0 }, // Gradually scale down to 0 users
  ],
  thresholds: {
    http_req_failed: ['rate<0.02'], // Less than 2% errors
    http_req_duration: ['p(95)<2000'], // 95% of requests < 2s
  },
};

const endpoints = [
  '/user',
  '/userRatings',
  '/matchResults',
  '/activeMatches',
  '/friends',
];
const baseUrl = 'https://33hiq368dh.execute-api.ap-southeast-2.amazonaws.com/dev';
const token = 'eyJraWQiOiJiNSt2S1A3QTF6SHdEMGNrSG1BZHJV3dFaDlwbmF3K0JZZWhPZ21kTUxvPSIsImFsZyI

// List of endpoints to hit

export default function main() {
  const randomPath = endpoints[Math.floor(Math.random() * endpoints.length)];
  const url = baseUrl + randomPath;

  const headers = {
    'Authorization': `${token}`,
    'Content-Type': 'application/json',
  };

  const res = http.get(url, { headers });

  check(res, {
    'status is 200': (r) => r.status === 200,
    'response time < 500ms': (r) => r.timings.duration < 500,
  });

  sleep(10); // Simulate think time
}

```

Hình C.2 Kịch bản 1000 người dùng API đồng thời

```

import { check } from 'k6';
import http from 'k6/http';
import { sleep } from 'k6';

// Configuration: Number of virtual users and duration of the test
export const options = {
  stages: [
    { duration: '5m', target: 10000 }, // Ramp up to 1000 users over 5 minutes
    { duration: '10m', target: 10000 }, // Hold 10,000 users for 10 minutes
    { duration: '5m', target: 10000 }, // Hold 10,000 users for another 5 minutes
    { duration: '5m', target: 0 }, // Gradually scale down to 0 users
  ],
  thresholds: {
    http_req_failed: ['rate<0.02'], // http errors should be less than 2%
    http_req_duration: ['p(95)<2000'], // 95% of requests should be below 2 seconds
  },
  cloud: {
    distribution: {
      'amazon:us:ashburn': { loadZone: 'amazon:us:ashburn', percent: 100 },
    },
  },
}

export default function main() {
  // Define your API endpoint(s)
  const url = 'https://33hiq368dh.execute-api.ap-southeast-2.amazonaws.com/dev';

  // Bearer Token for Authorization
  const token = 'your-token-here';

  // Headers for Authorization
  let headers = {
    'Authorization': `${token}`,
    'Content-Type': 'application/json', // Optional, based on your API
  };

  // Test the Matchmaking endpoint with Authorization
  let res = http.get(url + '/user', { headers: headers });

  // Check if the response status is 200 and response time is less than 500ms
  check(res, {
    'matchmaking status is 200': (r) => r.status === 200,
    'response time is less than 500ms': (r) => r.timings.duration < 500,
  });

  sleep(1); // Sleep to simulate user think time (adjust as needed)
}

```

Hình C.3 Kịch bản 10000 người dùng API đồng thời

Kiểm thử hiệu suất máy chủ trò chơi

```
import { sleep } from 'k6';
import ws from 'k6/ws';
import { Trend } from 'k6/metrics';

export const options = {
  vus: 1000,
  duration: '5m',
};

const moveLatency = new Trend('move_to_gamestate_latency', true);

// 8 moves total, ends in checkmate
const fullGame = [
  "e2e4", "e7e5", "f1c4", "b8c6", "d1h5", "g8f6", "h5f7"
];

function getColor(userId) {
  return userId % 2 === 0 ? 'white' : 'black';
}

function getMoves(color) {
  return fullGame.filter((_, i) => (color === 'white' ? i % 2 === 0 : i % 2 === 1));
}

function getMatchId(userId, round) {
  const playerGroup = Math.floor(userId / 2);
  return `match-${round}-${playerGroup}`;
}

export default function () {
  const userId = __VU;
  const color = getColor(userId);
  const moves = getMoves(color);
  const playerId = color === 'white' ? 'PLAYER_1' : 'PLAYER_2';
  let round = 0;

  while (true) {
    const matchId = getMatchId(userId, round);
    const serverIp = '3.27.132.51';
    const url = `ws://${serverIp}:7202/game/${matchId}?playerId=${playerId}`;
    let moveIndex = 0;
    let lastMoveSentAt = null;
```

Hình C.4 Kịch bản kiểm thử hiệu suất máy chủ trò chơi (1)

```

const res = ws.connect(url, {}, (socket) => {
  console.log(`[${color}] Connected to ${matchId}`);

  socket.on('open', () => {
    if (color === 'white') {
      sendMove(socket);
    }
  });

  socket.on('message', (msg) => {
    const data = JSON.parse(msg);

    if (data.type === 'gameState') {
      if (lastMoveSentAt !== null) {
        const latency = Date.now() - lastMoveSentAt;
        if (Math.random() < 0.4 && latency < 300) {
          moveLatency.add(latency); // 40% sampling
        }
        lastMoveSentAt = null;
      }

      if (moveIndex < moves.length) {
        sendMove(socket);
      } else {
        console.log(`[${color}] Game finished for ${matchId}`);
        socket.close();
      }
    }
  });
}

socket.on('close', () => {
  console.log(`[${color}] Closed ${matchId}`);
});

socket.on('error', (e) => {
  console.error(`[${color}] Error in ${matchId}:`, e.error());
});

function sendMove(socket) {
  const move = moves[moveIndex++];
  const thinkTime = Math.random() * 1.5 + 0.5; // 0-2s think time
  sleep(thinkTime);

  lastMoveSentAt = Date.now();

  socket.send(JSON.stringify({
    type: 'gameData',
    data: {
      action: 'move',
      move: move,
    },
    createdAt: new Date().toISOString(),
  }));

  console.log(`[${playerId}] ${matchId} move: ${move} (after ${thinkTime.toFixed(2)}s`)
}
};

sleep(Math.random() * 2 + 1); // 1-3s cooldown between matches
round++;
}
}

```

Hình C.5 Kịch bản kiểm thử hiệu suất máy chủ trò chơi (2)

Kiểm thử khả năng mở rộng của máy chủ trò chơi

```
import http from 'k6/http';
import ws from 'k6/ws';
import { sleep } from 'k6';

export const options = {
  stages: [
    { duration: '30s', target: 200 }, // ramp to 200 VUs
    { duration: '30s', target: 400 }, // ramp to 400 VUs
    { duration: '30s', target: 600 }, // ramp to 600 VUs
    { duration: '30s', target: 800 }, // ramp to 800 VUs
    { duration: '30s', target: 1000 }, // ramp to 1000 VUs
    { duration: '60s', target: 1000 },
    { duration: '60s', target: 0 },
  ],
};

const MATCHMAKING_API = 'https://ky9s1s2sla.execute-api.ap-southeast-2.amazonaws.com/dev/match'
const sampleMoves = ['e2e4', 'e7e5'];

export default function () {
  const userId = `user-${__VU}-${__ITER}`;
  let matchInfo = null;
  const payload = JSON.stringify({
    minRating: 1000,
    maxRating: 1300,
    gameMode: '10+0',
  });
  const headers = {
    'Content-Type': 'application/json',
  };

  // Keep trying until we get a match (HTTP 200)
  while (true) {
    const initialTime = Math.random() * 1.5 + 0.5;
    sleep(initialTime);
    const res = http.post(` ${MATCHMAKING_API}?userId=${userId}`, payload, { headers });
    if (res.status === 200) {
      matchInfo = res.json();
      console.log(`[${userId}] Matched successfully.`);
      break;
    } else if (res.status === 204) {
      console.log(`[${userId}] Still waiting for match... retrying`);
      sleep(1 + Math.random()); // wait before retry
    } else {
      console.error(`[${userId}] Unexpected matchmaking response: ${res.status}`);
      const initialTime = Math.random() * 4.5 + 0.5;
      sleep(initialTime);
    }
  }
}
```

Hình C.6 Kịch bản kiểm thử khả năng mở rộng của máy chủ trò chơi (1)

```

// Extract and connect to game server
const matchId = matchInfo.matchId;
const serverIp = matchInfo.server;
const playerId = Math.random() < 0.5 ? 'PLAYER_1' : 'PLAYER_2';
const color = playerId === 'PLAYER_1' ? 'white' : 'black';
const moves = color === 'white' ? sampleMoves : [...sampleMoves].reverse();

const gameUrl = `ws://${serverIp}:7202/game/${matchId}?playerId=${playerId}`;

ws.connect(gameUrl, {}, (socket) => {
  console.log(`[${userId}] Connected to ${matchId} on ${serverIp} as ${playerId}`);

  let moveIndex = 0;

  socket.on('open', () => {
    sendMove(socket);
  });

  socket.on('message', (msg) => {
    const data = JSON.parse(msg);
    if (data.type === 'gameState' && moveIndex < 2) {
      sendMove(socket);
    }
  });
});

socket.on('error', (e) => {
  console.error(`[${userId}] WebSocket error:`, e.error());
});

socket.on('close', () => {
  console.log(`[${userId}] Closed connection to match ${matchId}`);
});

function sendMove(socket) {
  const move = moves[moveIndex++];
  const thinkTime = Math.random() * 1.5 + 0.5;
  sleep(thinkTime);

  socket.send(JSON.stringify({
    type: 'gameData',
    data: {
      action: 'move',
      move,
    },
    createdAt: new Date().toISOString(),
  }));
}

console.log(`[${userId}] Sent move: ${move}`);
}

sleep(300); // 5 minutes
});
}

```

Hình C.7 Kịch bản kiểm thử khả năng mở rộng của máy chủ trò chơi (2)

Tài liệu tham khảo

Tiếng Anh

- [1] Amazon Web Services, *Serverless Architectures with AWS Lambda*, AWS Whitepaper, 2023. [Online]. Available: <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>
- [2] Amazon Web Services, *AWS AppSync Developer Guide*. [Online]. Available: <https://docs.aws.amazon.com/appsync/>
- [3] Amazon Web Services, *Amazon ECS on AWS Fargate Documentation*. [Online]. Available: <https://docs.aws.amazon.com/AmazonECS/latest/userguide/what-is-fargate.html>
- [4] Newzoo, *Global Games Market Report 2023*. [Online]. Available: <https://newzoo.com/insights/trend-reports/global-games-market-report-2023>
- [5] Game Developers Conference (GDC), *GDC State of the Game Industry 2023 Report*. [Online]. Available: <https://gdconf.com>
- [6] Amazon Web Services, *Amazon ECS on AWS Fargate Documentation*. [Online]. Available: <https://docs.aws.amazon.com/AmazonECS/latest/userguide/what-is-fargate.html>
- [7] M. Fowler, “Serverless Architectures.” [Online]. Available: <https://martinfowler.com/articles/serverless.html>
- [8] Amazon Web Services, *Building Modern Applications on AWS*, AWS Architecture Blog, 2023.
- [9] J. Gough, “Serverless Gaming: Where It Works and Where It Doesn’t,” Serialized.net, Mar. 2021. [Online]. Available: https://serialized.net/2021/03/serverless_gaming_limits/
- [10] D. Momente, “Developing a Serverless Gaming Backend With Real-Time Features,” MomentsLog, 2021. [Online]. Available: <https://www.momentslog.com/development/web-backend/developing-a-serverless-gaming-backend-with-real-time-features>
- [11] AWS Compute Blog, “Building a Serverless Multiplayer Game That Scales,” AWS Blog, 2022. [Online]. Available: <https://aws.amazon.com/blogs/compute/building-a-serverless-multiplayer-game-that-scales/>
- [12] AWS Architecture Center, *Multiplayer Session-Based Game Hosting on AWS*. [Online]. Available: <https://docs.aws.amazon.com/architecture-diagrams/latest/multiplayer-session-based-game-hosting-on-aws/multiplayer-session-based-game-hosting-on-aws.html>
- [13] A. E. Elo, *The Rating of Chessplayers, Past and Present*. Arco Pub., 1978.

- [14] M. E. Glickman, “Parameter estimation in large dynamic paired comparison experiments,” *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 48, no. 3, pp. 377–394, 1999.
- [15] R. Herbrich, T. Minka, and T. Graepel, “TrueSkill™: A Bayesian skill rating system,” in *Advances in Neural Information Processing Systems*, vol. 20, 2007, pp. 569–576.