

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÀNH PHỐ HỒ CHÍ MINH
KHOA : CÔNG NGHỆ THÔNG TIN



BÁO CÁO ĐỒ ÁN HỌC PHẦN: CƠ SỞ DỮ LIỆU NÂNG CAO

(Học kì 2, năm học 2025-2026)

Đề tài: Tự động cung cấp tài nguyên theo nhu cầu Serverless

GVHD: ThS. Nguyễn Văn Danh

Nhóm SV thực hiện:

STT	HỌ VÀ TÊN	MSSV	LỚP
1	LÂM QUỐC ĐẠT	2011060180	22DTHB5
2	TRẦN THIỆN NHÂN	2280602189	22DTHB5
3	PHAN THANH PHONG	2280602380	22DTHB5
4	TRẦN NHƯ KHÁNH	2280601478	22DTHB5

TP.Hồ Chí Minh, tháng 4 năm 2025

LỜI CẢM ƠN

Chúng em xin bày tỏ lòng biết ơn chân thành tới quý thầy cô trường Đại Học Công Nghệ Thành Phố Hồ Chí Minh vì đã tạo điều kiện tuyệt vời cho chúng em thực hiện và hoàn thành đề tài nghiên cứu của chúng em. Sự hỗ trợ và sự đồng hành của quý thầy đã đóng góp quan trọng vào quá trình nghiên cứu và giúp chúng em tiến bộ trong học tập.

Đặc biệt, chúng em xin gửi lời cảm ơn sâu sắc và lòng biết ơn tới thầy Nguyễn Văn Danh - giáo viên bộ môn Cơ Sở Dữ Liệu Nâng Cao ngành Công nghệ thông tin. Thầy đã tận tình hướng dẫn, chỉ bảo và giúp đỡ chúng em trong suốt quá trình thực hiện đề tài. Sự kiên nhẫn và sự truyền cảm hứng từ thầy đã thúc đẩy chúng em vượt qua những khó khăn và phát triển kỹ năng nghiên cứu của mình.

Chúng em cũng muốn gửi lời cảm ơn chân thành tới tất cả các thầy cô giáo trong và ngoài khoa công nghệ thông tin. Được học tập và trang bị kiến thức từ quý thầy cô là một phần không thể thiếu trong quá trình chuẩn bị và thực hiện đề tài của chúng em. Những kiến thức và kỹ năng mà chúng em đã học được từ các môn học trước đó đã cung cấp nền tảng vững chắc để chúng em tiến xa hơn trong nghiên cứu của mình.

Chúng em hiểu rằng với thời gian và năng lực có hạn, không thể tránh khỏi những sai sót trong quá trình thực hiện đề tài nghiên cứu của mình. Tuy nhiên, chúng em hy vọng nhận được sự góp ý bổ sung từ thầy Nguyễn Văn Danh và các giáo viên khác để đề tài của chúng em ngày càng hoàn thiện hơn. Sự góp ý và đánh giá từ quý thầy cô sẽ giúp chúng em nắm bắt được những khía cạnh mới, cải thiện phương pháp nghiên cứu và đảm bảo chất lượng của đề tài.

Cuối cùng, chúng em xin chân thành cảm ơn mọi người đã đồng hành và đóng góp vào thành công của đề tài nghiên cứu này. Sự giúp đỡ và hỗ trợ của quý thầy cô đã truyền cảm hứng cho chúng em và khuyến khích chúng em tiếp tục theo đuổi sự nghiên cứu và học tập. Chúng em sẽ không ngừng cố gắng để phát triển và đóng góp vào lĩnh vực công nghệ thông tin cũng như xã hội.

MỤC LỤC

CHƯƠNG 1. TỔNG QUAN	5
1.1 Giới thiệu Serverless	5
1.2 Khái niệm Serverless	5
1.3 Lợi ích của Serverless	5
1.4 Phân chia công việc	6
<i>Bảng 1-1 Phân chia công việc</i>	6
CHƯƠNG 2. KIẾN TRÚC HỆ THỐNG SERVERLESS	7
2.1 Mô hình kiến trúc Event-driven	7
2.2 So sánh Serverless với kiến trúc máy chủ truyền thống	7
2.3 Các thành phần chính trong hệ thống Serverless	8
2.3.1 Function-as-a-Service (FaaS):	8
2.3.2 Backend-as-a-Service (BaaS):	8
2.4 Ví dụ thực tế trong hệ sinh thái AWS	8
2.5 Mô hình minh họa kiến trúc Serverless	8
2.6. Kết luận chương	8
CHƯƠNG 3. CƠ CHẾ CUNG CẤP TÀI NGUYÊN VỚI CÔNG NGHỆ AMAZON AWS	9
3.1 Giới thiệu chung	9
3.2 Tổng quan về cơ chế cấp phát tài nguyên trên AWS	9
3.3 Aurora Serverless v2 – Mô hình cấp phát linh hoạt	9
3.4 Amazon RDS và cơ chế cấp phát truyền thống	10
3.5 Cơ chế Auto Scaling trên AWS	10
3.6 So sánh Aurora Serverless v2 và Amazon RDS	11
CHƯƠNG 4. KẾT QUẢ THỰC NGHIỆM	12
4.1 Mục tiêu thực nghiệm:	12
4.2 Giới thiệu Amazon Aurora và Amazon RDS	12
4.2.1 Amazon RDS (Relational Database Service)	12

4.2.2 Amazon Aurora Serverless	12
4.3 Ứng dụng Aurora Serverless và RDS trong Demo	12
4.4 Thiết lập môi trường thực nghiệm	13
4.4.1 Công cụ / Dịch vụ triển khai: Sử dụng Amazon RDS và Aurora Serverless	13
4.4.2 Tạo môi trường:	13
4.5 Cấu hình kết nối và truy cập	15
4.6 Kết nối DBeaver tới Aurora Serverless	16
4.7 Nhập dữ liệu và thực thi thử nghiệm	17
4.8 Theo dõi tài nguyên mở rộng (ACUUtilization)	19
4.9 Kết luận thực nghiệm	20
CHƯƠNG 5. TÀI LIỆU THAM KHẢO	21

CHƯƠNG 1. TỔNG QUAN

1.1 Giới thiệu Serverless

Trong thời đại số hóa, các doanh nghiệp và nhà phát triển liên tục tìm kiếm những giải pháp giúp đơn giản hóa việc vận hành hạ tầng, đồng thời tối ưu hóa chi phí và hiệu năng. Một trong những xu hướng công nghệ nổi bật hiện nay chính là **Serverless Computing** – một mô hình cho phép các lập trình viên tập trung hoàn toàn vào logic nghiệp vụ mà không cần quan tâm đến việc quản lý máy chủ vật lý hoặc ảo. **Tính khả thi của bài toán**

Serverless không có nghĩa là "không có máy chủ", mà đúng hơn là việc "**ẩn máy chủ khỏi người dùng**". Trong mô hình này, việc khởi tạo, cấu hình, mở rộng và quản lý tài nguyên đều được tự động hóa bởi nhà cung cấp dịch vụ đám mây (như AWS, Azure, GCP). Người dùng chỉ cần viết và triển khai các **function nhỏ (FaaS – Function-as-a-Service)**, các function này sẽ được tự động gọi khi có sự kiện tương ứng xảy ra.

1.2 Khái niệm Serverless

Serverless là một mô hình kiến trúc trong đó việc thực thi ứng dụng được xử lý dưới dạng các hàm (function), và hạ tầng để chạy các hàm này được tự động khởi tạo, mở rộng và hủy khi không còn sử dụng.

Một số đặc điểm chính:

- **Event-driven:** Hệ thống chỉ hoạt động khi có sự kiện xảy ra (như HTTP request, thông điệp hàng đợi, file mới được upload,...).
- **Tự động mở rộng (Auto-scaling):** Khả năng mở rộng theo thời gian thực tùy theo lượng truy cập.
- **Không cần quản lý server:** Người dùng không phải cấu hình, vá lỗi hay duy trì máy chủ.
- **Thanh toán theo mức sử dụng:** Chỉ trả tiền cho thời gian thực thi thực tế của hàm.

1.3 Lợi ích của Serverless

- **Tiết kiệm chi phí:** Serverless chỉ tính phí khi có function được gọi, giúp giảm lãng phí tài nguyên.
- **Khả năng mở rộng tự động:** Khả năng đáp ứng lượng truy cập lớn mà không cần thêm thao tác thủ công.
- **Không cần quản lý cơ sở hạ tầng:** Loại bỏ chi phí và rủi ro liên quan đến vận hành máy chủ vật lý hoặc ảo.
- **Triển khai nhanh chóng:** Các ứng dụng và chức năng có thể được triển khai chỉ với vài dòng mã.
- **Tăng cường bảo mật:** Do cơ sở hạ tầng được cập nhật và bảo trì bởi các nhà cung cấp cloud.

Serverless cung cấp một mô hình linh hoạt, hiệu quả chi phí và phù hợp cho những ứng dụng có tải không đều hoặc cần phản hồi nhanh, như API, xử lý ảnh/video, chatbot, hoặc hệ thống đặt vé trực tuyến.

1.4 Phân chia công việc

MSSV	Họ Tên	Công việc đã làm
2011060180	Lâm Quốc Đạt	Kiến trúc hệ thống
2280602189	Trần Thiện Nhân	Kết quả thực nghiệm (Demo)
2280602380	Phan Thanh Phong	Giới thiệu, khái niệm, đặc điểm, lợi ích
2280601478	Trần Như Khánh	Cơ chế hoạt động cung cấp tài nguyên

Bảng 1-1 Phân chia công việc

CHƯƠNG 2. KIẾN TRÚC HỆ THỐNG SERVERLESS

2.1 Mô hình kiến trúc Event-driven

Kiến trúc Serverless hoạt động chủ yếu dựa trên mô hình event-driven – tức là các thành phần trong hệ thống sẽ tự động phản hồi lại các sự kiện (event) mà không cần máy chủ chạy liên tục.

Các bước hoạt động cơ bản của mô hình:

1. Người dùng hoặc hệ thống tạo ra sự kiện (ví dụ: gửi HTTP request, tải tệp lên, thêm dữ liệu mới...).
2. Sự kiện này được định tuyến tới một dịch vụ trung gian như API Gateway hoặc EventBridge.
3. Function (chức năng) tương ứng được kích hoạt, xử lý sự kiện (như tính toán, lưu dữ liệu, gửi email...).
4. Dữ liệu được lưu trữ lại hoặc phản hồi trả về người dùng.

Ví dụ: Khi người dùng tải ảnh lên S3, Lambda tự động được kích hoạt để xử lý ảnh (nén, resize) và lưu sang thư mục khác – tất cả diễn ra tự động và không cần máy chủ truyền thống.

Nguyên lý hoạt động:

1. Một sự kiện xảy ra (ví dụ: người dùng gửi HTTP request, tải ảnh lên S3, dữ liệu mới thêm vào DB).
2. Sự kiện này sẽ kích hoạt một function serverless – ví dụ như AWS Lambda.
3. Function xử lý logic và trả về kết quả hoặc lưu trữ dữ liệu ra backend.

Mô hình event-driven giúp hệ thống phản ứng nhanh, linh hoạt và tối ưu hiệu năng bằng cách chỉ sử dụng tài nguyên khi cần thiết.

2.2 So sánh Serverless với kiến trúc máy chủ truyền thống

Tiêu chí	Server-based truyền thống	Kiến trúc Serverless
Quản lý máy chủ	Người dùng phải cấu hình, bảo trì	Không cần quản lý máy chủ
Chi phí tài nguyên	Trả phí liên tục (dù không sử dụng)	Trả phí theo lượt gọi (on-demand)
Mở rộng hệ thống	Phải scale thủ công, dễ sai sót	Tự động mở rộng theo tải
Thời gian triển khai	Lâu, phức tạp	Nhanh, dễ triển khai
Độ phức tạp	Cần DevOps, sysadmin chuyên biệt	Tập trung vào logic ứng dụng

Kết luận: Kiến trúc Serverless mang lại sự linh hoạt, đơn giản hóa quản trị, và giảm chi phí vận hành so với kiến trúc truyền thống.

2.3 Các thành phần chính trong hệ thống Serverless

2.3.1 Function-as-a-Service (FaaS):

- Là thành phần cốt lõi của mô hình Serverless.
- Cho phép bạn viết các function nhỏ thực hiện nhiệm vụ cụ thể và chỉ chạy khi được kích hoạt.

Ví dụ: AWS Lambda, Azure Functions, Google Cloud Functions.

2.3.2 Backend-as-a-Service (BaaS):

- Là những dịch vụ backend được cung cấp sẵn, giúp bạn không cần tự xây dựng từ đầu.

Ví dụ: xác thực người dùng (Firebase Auth), cơ sở dữ liệu realtime (Firebase Realtime DB), push notification,...

Sự kết hợp giữa FaaS + BaaS tạo nên một hệ sinh thái linh hoạt, nơi bạn có thể triển khai toàn bộ ứng dụng chỉ bằng các function và dịch vụ sẵn có.

2.4 Ví dụ thực tế trong hệ sinh thái AWS

AWS Lambda (FaaS):

- Dịch vụ function serverless. Mỗi function thực thi một đoạn mã khi có trigger.
- Tự động scale theo số lượng request.

Amazon API Gateway:

- Tạo và quản lý API REST hoặc HTTP.
- Làm cầu nối giữa người dùng và Lambda.

Amazon DynamoDB:

- Cơ sở dữ liệu NoSQL serverless của AWS.
- Scale tự động, truy cập với độ trễ thấp, không cần cấu hình server.

2.5 Mô hình minh họa kiến trúc Serverless

[User Request]



[API Gateway] → xác thực & định tuyến request



[AWS Lambda Function] → thực hiện logic (ví dụ xử lý đơn hàng)



[DynamoDB] → lưu trữ dữ liệu kết quả

2.6. Kết luận chương

Kiến trúc Serverless không chỉ đơn thuần là “không có server”, mà là sự thay đổi hoàn toàn cách tiếp cận triển khai ứng dụng – tập trung vào sự kiện, hàm xử lý nhỏ gọn và tận dụng các dịch vụ backend có sẵn. Sự kết hợp giữa các thành phần như Lambda, API Gateway, DynamoDB đã mở ra một kỷ nguyên ứng dụng hiệu quả, linh hoạt và tiết kiệm chi phí trên đám mây.

CHƯƠNG 3. CƠ CHẾ CUNG CẤP TÀI NGUYÊN VỚI CÔNG NGHỆ AMAZON AWS

3.1 Giới thiệu chung

Amazon Web Services (AWS) là nền tảng điện toán đám mây phổ biến và mạnh mẽ hàng đầu thế giới, cung cấp hàng trăm dịch vụ đa dạng để phục vụ các nhu cầu hạ tầng, lưu trữ, tính toán, trí tuệ nhân tạo và quản lý dữ liệu của doanh nghiệp. Một trong những điểm khác biệt quan trọng của AWS so với phương pháp triển khai truyền thống là khả năng **cấp phát tài nguyên linh hoạt theo nhu cầu**, giúp tối ưu hiệu suất, giảm chi phí vận hành và tăng cường khả năng mở rộng hệ thống.

Trong môi trường **serverless**, người dùng không cần quan tâm đến việc cài đặt, quản lý hoặc điều chỉnh hạ tầng vật lý hoặc ảo. Thay vào đó, AWS sẽ tự động phân bổ tài nguyên tính toán khi có yêu cầu và giải phóng khi không còn truy cập, giúp hệ thống vận hành hiệu quả hơn mà không cần giám sát liên tục.

3.2 Tổng quan về cơ chế cấp phát tài nguyên trên AWS

Việc cấp phát tài nguyên là nền tảng cho hoạt động của các dịch vụ như **EC2**, **Lambda**, **RDS**, **Aurora**, **ECS**, và nhiều dịch vụ khác. Cơ chế này cho phép AWS phân phối các tài nguyên tính toán như **CPU**, **RAM**, **lưu trữ ổ cứng (EBS)** và **băng thông mạng** một cách tối ưu nhất.

AWS cung cấp ba mô hình cấp phát tài nguyên chính:

- **Cấp phát thủ công:** Người dùng tự chọn loại máy chủ (instance type), cấu hình và dung lượng phù hợp với ứng dụng. Phương pháp này phù hợp với các hệ thống cố định, ít thay đổi.
- **Cấp phát tự động (Auto Scaling):** Dựa trên các chính sách định nghĩa sẵn, hệ thống sẽ tự động tăng hoặc giảm số lượng tài nguyên đang sử dụng. Đây là mô hình phổ biến với các ứng dụng có lưu lượng dao động.
- **Cấp phát serverless:** Hệ thống chỉ cung cấp tài nguyên khi có yêu cầu thực thi (on-demand). Ví dụ điển hình là **AWS Lambda** và **Aurora Serverless**, nơi tài nguyên được cấp phát tự động theo mức sử dụng thực tế.

3.3 Aurora Serverless v2 – Mô hình cấp phát linh hoạt

Amazon Aurora Serverless v2 là một biến thể của Amazon Aurora – hệ quản trị cơ sở dữ liệu tương thích với MySQL và PostgreSQL, nhưng tối ưu hiệu suất hơn đáng kể. Aurora Serverless v2 hoạt động mà không cần xác định trước kích thước máy chủ cơ sở dữ liệu. Thay vào đó, nó sử dụng đơn vị đo tài nguyên riêng gọi là **ACU (Aurora Capacity Unit)** để tự động điều chỉnh tài nguyên phù hợp với khối lượng truy vấn.

Ưu điểm nổi bật:

- **Tự động mở rộng từ 0.5 đến hàng trăm ACU chỉ trong vài giây.**

- **Tạm dừng hoạt động khi không có truy vấn và tự động khởi động lại khi có truy vấn mới.**
- **Chi phí tính theo giây**, dựa trên tài nguyên sử dụng thực tế.
- **Tích hợp các tính năng nâng cao** như Global Database, Multi-AZ, Backtrack...

Không gây gián đoạn khi mở rộng, không cần thao tác nâng cấp thủ công.

3.4 Amazon RDS và cơ chế cấp phát truyền thống

Amazon RDS (Relational Database Service) là dịch vụ cơ sở dữ liệu có quản lý được nhiều người sử dụng từ trước khi Aurora xuất hiện. RDS hỗ trợ nhiều hệ quản trị phổ biến như MySQL, PostgreSQL, MariaDB, Oracle và SQL Server.

Tuy nhiên, RDS sử dụng mô hình **cấp phát tài nguyên cố định**, nghĩa là người dùng cần chọn một cấu hình cụ thể (ví dụ t2.medium, r5.large...) khi khởi tạo instance. Nếu sau đó tải truy cập tăng vượt khả năng đáp ứng của instance, người dùng buộc phải **nâng cấp thủ công lên phiên bản cao hơn**, dẫn tới **downtime tạm thời** và **chi phí cao hơn**.

Đặc điểm chính của RDS:

- Sử dụng máy chủ EC2 làm host cơ sở dữ liệu.
- Không thể tạm dừng hoạt động để tiết kiệm chi phí.
- Có hỗ trợ Auto Scaling thông qua Read Replica nhưng không linh hoạt.
- Phù hợp với hệ thống cố định, yêu cầu cấu hình ổn định, ít thay đổi.

3.5 Cơ chế Auto Scaling trên AWS

AWS cung cấp dịch vụ **Auto Scaling** cho nhiều loại tài nguyên. Tính năng này cho phép tự động điều chỉnh quy mô tài nguyên theo thời gian thực nhằm đảm bảo hiệu suất hệ thống và tiết kiệm chi phí.

Các phương pháp Auto Scaling phổ biến:

- **Target Tracking Scaling:** Giữ một chỉ số cụ thể (ví dụ CPU usage) ở mức mục tiêu.
- **Step Scaling:** Tăng/giảm tài nguyên theo từng bậc khi vượt ngưỡng.
- **Scheduled Scaling:** Đặt lịch tăng/giảm tài nguyên theo khung giờ.

Dịch vụ hỗ trợ Auto Scaling:

- Amazon EC2 Auto Scaling Groups
- AWS Lambda (Provisioned Concurrency)
- Amazon ECS (Elastic Container Service)
- Aurora Serverless (tự động điều chỉnh ACU)
- DynamoDB, SQS, v.v.

3.6 So sánh Aurora Serverless v2 và Amazon RDS

Tiêu chí	Aurora Serverless v2	Amazon RDS
Cấp phát tài nguyên	Tự động theo nhu cầu truy vấn	Cố định theo cấu hình instance
Khả năng mở rộng	Linh hoạt theo thời gian thực (0.5 ACU trở lên)	Thủ công, dễ gây downtime khi nâng cấp
Xử lý trạng thái idle	Tự động tạm dừng và khôi phục	Luôn hoạt động, không tạm dừng được
Chi phí vận hành	Tính theo giây, đúng với tài nguyên thực tế sử dụng	Tính theo giờ, ít linh hoạt hơn
Tính năng nâng cao	Đầy đủ: Multi-AZ, Global DB, Backtrack...	Giới hạn, tùy theo hệ quản trị
Thời gian khởi động	Vài giây khi có truy vấn	Không cần khởi động vì luôn bật
Phù hợp với	Ứng dụng tải biến động, không thường xuyên, tiết kiệm	Ứng dụng tải ổn định, cần hiệu năng cao liên tục

CHƯƠNG 4. KẾT QUẢ THỰC NGHIỆM

4.1 Mục tiêu thực nghiệm:

Mục tiêu của phần thực nghiệm là kiểm chứng khả năng tự động điều chỉnh tài nguyên của Aurora Serverless v2. Mình muốn xem khi có tải truy cập tăng lên, hệ thống có tự mở rộng (scale out), và khi hết tải thì có tự thu hẹp lại (scale in) hay không.

4.2 Giới thiệu Amazon Aurora và Amazon RDS

Để ứng dụng Serverless vào hệ thống quản lý dữ liệu, các dịch vụ cơ sở dữ liệu cloud cũng cần khả năng tự động mở rộng tài nguyên như function.

4.2.1 Amazon RDS (Relational Database Service)

Amazon RDS là một dịch vụ cơ sở dữ liệu được quản lý toàn phần, hỗ trợ các hệ quản trị như MySQL, PostgreSQL, Oracle và SQL Server. Tuy không hoàn toàn là Serverless, RDS vẫn hỗ trợ các tính năng mở rộng tài nguyên như:

- Tự động scale read replica.
- Tăng giảm IOPS theo nhu cầu.
- Backup, patching và failover tự động.

RDS phù hợp với các ứng dụng cần hệ cơ sở dữ liệu truyền thống nhưng muốn giảm chi phí vận hành.

4.2.2 Amazon Aurora Serverless

Aurora Serverless là một phiên bản mở rộng của Amazon Aurora (tương thích MySQL và PostgreSQL), được thiết kế riêng cho các ứng dụng có tải không đều, với khả năng:

- **Tự động scale capacity (vCPU, RAM)** theo thời gian thực.
- **Khởi tạo và dừng theo nhu cầu:** Khi không có query, hệ thống sẽ “ngủ” để tiết kiệm chi phí.
- **Tính phí theo giây** dựa trên mức tài nguyên sử dụng (Aurora Capacity Unit – ACU).

Đây là một lựa chọn lý tưởng cho các ứng dụng cần tính năng serverless cho cả phần xử lý logic (Lambda) lẫn phần lưu trữ dữ liệu.

4.3 Ứng dụng Aurora Serverless và RDS trong Demo

Trong phần demo của đề tài, có thể triển khai:

- Một **API Serverless bằng AWS Lambda + API Gateway**.
- Kết nối đến **Amazon Aurora Serverless** để lưu trữ và truy xuất dữ liệu động.

- Quan sát quá trình auto-scale khi tạo nhiều request song song bằng công cụ giả lập như **Artillery** hoặc **ab**.

Aurora Serverless cho phép bạn kiểm chứng rõ việc **tự động cấp phát và thu hồi tài nguyên backend theo lưu lượng thực tế**, một tính năng cốt lõi trong hệ thống Serverless hiện đại.

4.4 Thiết lập môi trường thực nghiệm

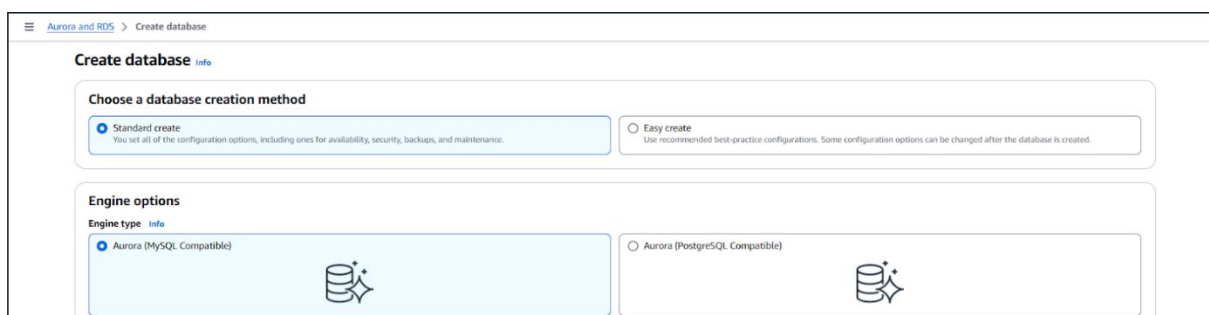
4.4.1 Công cụ / Dịch vụ triển khai: Sử dụng Amazon RDS và Aurora Serverless

Amazon RDS (Relational Database Service) là dịch vụ quản lý cơ sở dữ liệu quan hệ trên nền tảng đám mây AWS. Aurora là một engine cơ sở dữ liệu tối ưu do Amazon phát triển, tương thích với MySQL và PostgreSQL, có hiệu suất cao và khả năng tự động mở rộng tài nguyên.

Aurora Serverless là phiên bản không máy chủ (serverless) của Aurora, cho phép hệ thống tự động cấp phát hoặc thu hồi tài nguyên dựa theo mức sử dụng thực tế. Điều này đặc biệt phù hợp với các ứng dụng có lưu lượng truy cập không ổn định.


4.4.2 Tạo môi trường:

Bước 1: Chọn 'Standard create' để có toàn quyền cấu hình chi tiết và chọn 'Aurora (MySQL Compatible)' làm engine để tận dụng tính năng serverless.



The screenshot shows the 'Create database' page in the AWS Aurora console. Under 'Choose a database creation method', 'Standard create' is selected. Under 'Engine options', 'Aurora (MySQL Compatible)' is selected.

Bước 2: Chọn template 'Dev/Test' vì đây là môi trường thử nghiệm, không yêu cầu cấu hình cao như sản phẩm thật (production).



The screenshot shows the 'Templates' section with 'Dev/Test' selected. The description for 'Dev/Test' is: 'This instance is intended for development use outside of a production environment.'

Bước 3: Đặt tên cho cụm cơ sở dữ liệu và tạo tài khoản quản trị (username: admin). Chọn 'Self managed' để tự quản lý mật khẩu thay vì dùng Secrets Manager.

Settings

DB cluster identifier [Info](#)
Enter a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.

database-4

The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 63 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

▼ **Credentials Settings**

Master username [Info](#)
Type a login ID for the master user of your DB instance.

admin

1 to 32 alphanumeric characters. The first character must be a letter.

Credentials management
You can use AWS Secrets Manager or manage your master user credentials.

☐ Managed in AWS Secrets Manager - most secure
RDS generates a password for you and manages it throughout its lifecycle using AWS Secrets Manager.

☒ Self managed
Create your own password or have RDS create a password that you manage.

☐ Auto generate password
Amazon RDS can generate a password for you, or you can specify your own password.

Master password [Info](#)

Password strength **Very strong**

Minimum constraints: At least 8 printable ASCII characters. Can't contain any of the following symbols: / * @

Confirm master password [Info](#)

Bước 4: Chọn loại instance là 'Serverless v2'. Đây là phiên bản có khả năng mở rộng linh hoạt. Thiết lập mức tài nguyên tối thiểu là 0.5 ACU (1 GiB RAM) và tối đa là 2 ACU (4 GiB RAM).

Instance configuration
The DB instance configuration options below are limited to those supported by the engine that you selected above.

DB instance class [Info](#)

▼ **Hide filters**

☐ Include previous generation classes

☒ Serverless v2

☐ Memory optimized classes (includes r classes)

☐ Burstable classes (includes t classes)

Serverless v2
Instant scaling for even the most demanding workloads.

Capacity range [Info](#)
Database capacity is measured in Aurora Capacity Units (ACUs). 1 ACU provides 2 GiB of memory and corresponding compute and networking.

Minimum capacity (ACUs) **Maximum capacity (ACUs)**

0.5 (1 GiB) 2 (4 GiB)

0.5 to 128 in increments of 0.5 1 to 128 in increments of 0.5

Pause after inactivity [Info](#)
Clusters with a minimum capacity setting of 0 ACUs can be paused during inactivity. Specify the amount of time the DB instance can be idle before pausing. For more information, see [Pause Aurora Serverless DB Instances](#).

00:05:00

Enter a time from 5 minutes to 24 hours

☒ Selected Aurora MySQL version doesn't support pausing Aurora Serverless v2
Choose version 3.08.0 or higher to use.

Bước 5: Chọn 'Public access = Yes' để cho phép các dịch vụ như EC2 hoặc máy bên ngoài có thể kết nối đến cơ sở dữ liệu.

Public access [Info](#)

☒ Yes
RDS assigns a public IP address to the cluster. Amazon EC2 instances and other resources outside of the VPC can connect to your cluster. Resources inside the VPC can also connect to the cluster. Choose one or more VPC security groups that specify which resources can connect to the cluster.

☐ No
RDS doesn't assign a public IP address to the cluster. Only Amazon EC2 instances and other resources inside the VPC can connect to your cluster. Choose one or more VPC security groups that specify which resources can connect to the cluster.

Sau khi hoàn tất các bước, nhấn 'Create database' và đợi vài phút để Aurora Serverless được khởi tạo thành công.

4.5 Cấu hình kết nối và truy cập

Để EC2 có thể kết nối đến cơ sở dữ liệu Aurora Serverless, cần đảm bảo thiết lập đúng các quyền truy cập mạng thông qua Security Group. Security Group là một tường lửa ảo trong AWS, cho phép kiểm soát lưu lượng mạng đi vào và đi ra từ các tài nguyên như EC2 hoặc RDS.

Trong môi trường thực nghiệm này, chúng tôi sử dụng một Security Group mặc định ('default') có ID là 'sg-05f0dd560d8b7ab74'. Security Group này được gán cho cả EC2 instance và Aurora Serverless cluster nhằm cho phép chúng giao tiếp với nhau. Hình dưới đây minh họa Security Group đang được sử dụng:



Name	Security group ID	Security group name	VPC ID	Description	Owner	Inbound rules
-	sg-05f0dd560d8b7ab74	default	vpc-0ab6bf1d96bf4955f	default VPC security group	746669231402	1 Permission ent

Cần đảm bảo rằng Security Group cho Aurora Serverless mở cổng 3306 (MySQL) cho phép nhận kết nối từ EC2. Nếu bạn sử dụng chung Security Group cho cả EC2 và RDS, bạn có thể thiết lập rule như sau:

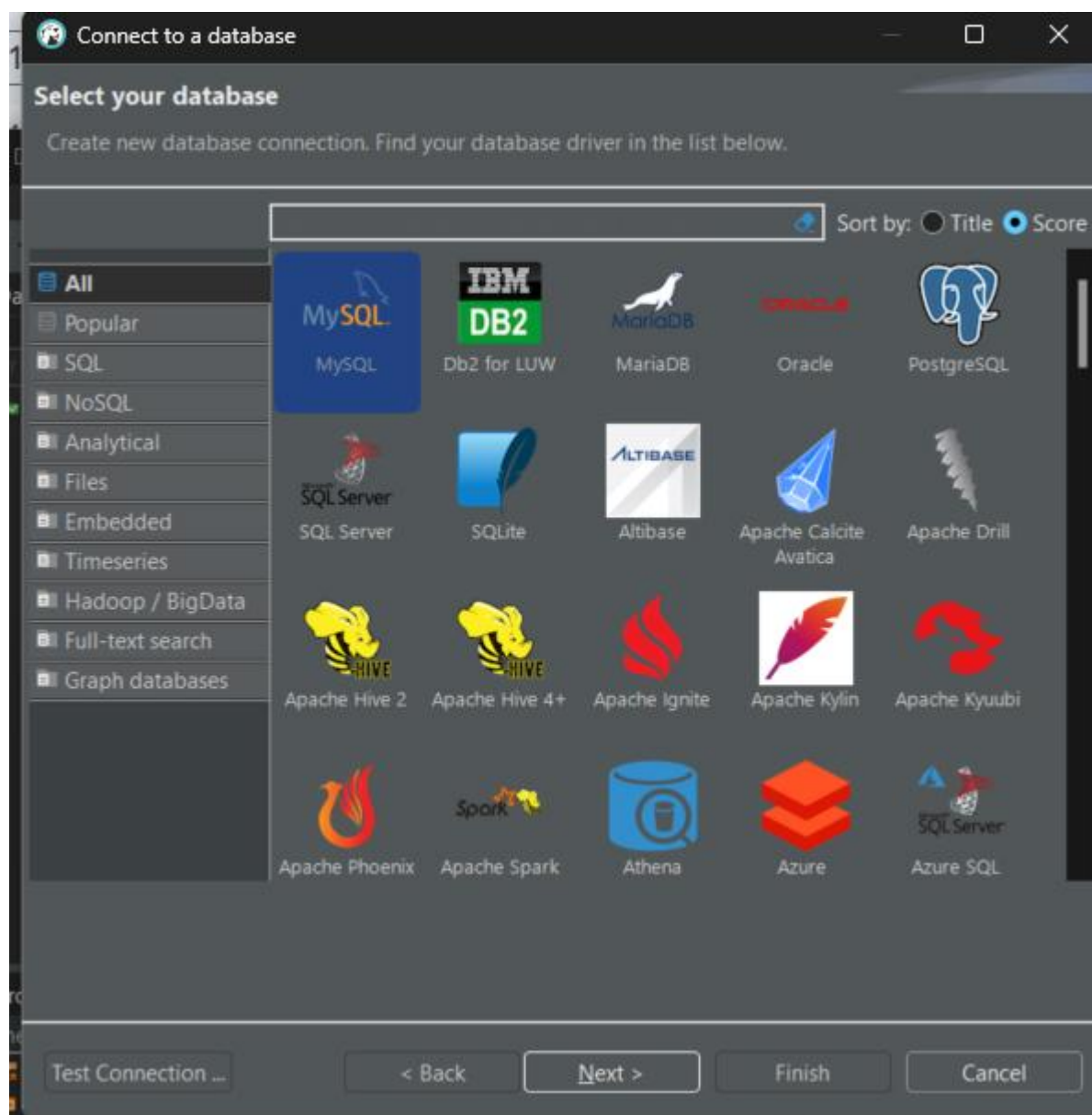
- Type: MySQL/Aurora
- Protocol: TCP
- Port Range: 3306
- Source: Security Group của EC2 hoặc 0.0.0.0/0 (tạm thời để test)

Sau khi thiết lập, EC2 instance sẽ có thể kết nối tới Aurora Serverless bằng MySQL client thông qua endpoint được cung cấp.

4.6 Kết nối DBeaver tới Aurora Serverless

Để thuận tiện trong việc quản lý cơ sở dữ liệu Aurora Serverless, công cụ DBeaver được sử dụng nhằm thiết lập giao diện kết nối trực quan và thao tác dễ dàng.

Bước 1: Mở DBeaver và tạo kết nối mới bằng cách nhấn vào dấu '+' (New Database Connection), sau đó chọn MySQL.



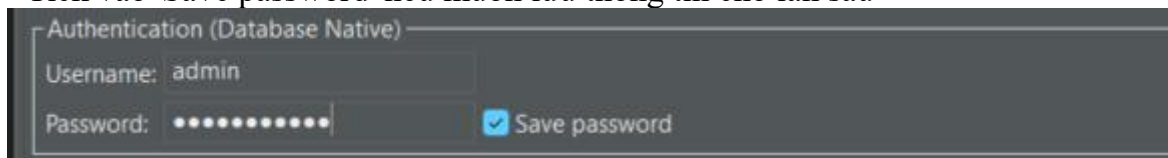
Bước 2: Trong phần 'Connection Settings', điền thông tin kết nối như sau:

- Server Host: dán endpoint của Aurora Serverless (ví dụ: `database-2.cluster-xxxxxx.us-east-1.rds.amazonaws.com`)
- Port: 3306 (cổng mặc định của MySQL)
- Database: có thể để trống hoặc nhập `mysql` ban đầu



Bước 3: Trong phần 'Authentication', nhập thông tin đăng nhập đã tạo khi khởi tạo Aurora:

- Username: admin (hoặc tài khoản bạn đã đặt)
- Password: mật khẩu tương ứng
- Tick vào 'Save password' nếu muốn lưu thông tin cho lần sau



Bước 4: Nhấn 'Test Connection' để kiểm tra kết nối. Nếu được yêu cầu, nhấn 'Download Driver' để tải driver MySQL. Sau khi kiểm tra thành công, nhấn 'Finish' để hoàn tất quá trình thiết lập.

Sau khi kết nối thành công, người dùng có thể thao tác với cơ sở dữ liệu Aurora Serverless như một hệ quản trị MySQL thông thường qua giao diện DBeaver.

4.7 Nhập dữ liệu và thực thi thử nghiệm

Để kiểm chứng khả năng tự động điều chỉnh tài nguyên của Aurora Serverless, một lượng lớn dữ liệu đã được sinh ra và truy vấn thông qua các thủ tục (stored procedures). Cụ thể, chúng tôi đã tạo bảng `students` và `marks`, sau đó sinh dữ liệu với quy mô 100.000 sinh viên và 500.000 điểm số tương ứng.

Quá trình thực thi bao gồm 2 bước chính: sinh dữ liệu lớn và mô phỏng truy vấn đồng thời để tạo tải cho hệ thống.

Bước 1: Tạo bảng và sinh dữ liệu lớn:

```

• -- Sử dụng database
USE demo_serverless;

• -- Tạo bảng
CREATE TABLE students (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100),
  student_code VARCHAR(20)
);

• CREATE TABLE marks (
  id INT AUTO_INCREMENT PRIMARY KEY,
  student_id INT,
  subject VARCHAR(50),
  score DECIMAL(4,2),
  FOREIGN KEY (student_id) REFERENCES students(id)
);

-- Tạo 100.000 sinh viên và 500.000 điểm:
DELIMITER $$

• CREATE PROCEDURE generate_data()
BEGIN
  DECLARE i INT DEFAULT 1;
  WHILE i <= 100000 DO
    INSERT INTO students(name, student_code)
    VALUES (CONCAT('Student ', i), CONCAT('HS', LPAD(i, 5, '0')));

    • INSERT INTO marks(student_id, subject, score) VALUES
      (i, 'Math', ROUND(RAND() * 10, 2)),
      (i, 'English', ROUND(RAND() * 10, 2)),
      (i, 'Physics', ROUND(RAND() * 10, 2)),
      (i, 'Chemistry', ROUND(RAND() * 10, 2)),
      (i, 'History', ROUND(RAND() * 10, 2));

    • SET i = i + 1;
  END WHILE;
END;
DELIMITER ;

```

Bước 2: Mô phỏng truy vấn đồng thời

```

• -- Gọi sinh dữ liệu
CALL generate_data();

-- Thủ tục này giả lập 500 sinh viên đang xem điểm cùng lúc.

DELIMITER $$

• CREATE PROCEDURE simulate_traffic()
BEGIN
  DECLARE i INT DEFAULT 1;
  WHILE i <= 100 DO
    SELECT s.name, s.student_code, m.subject, m.score
    FROM students s
    JOIN marks m ON s.id = m.student_id
    WHERE s.id = FLOOR(1 + RAND() * 100000)
    ORDER BY m.score DESC
    LIMIT 50;
    SET i = i + 1;
  END WHILE;
END;
DELIMITER ;

• -- Gọi tạo tải
CALL simulate_traffic();

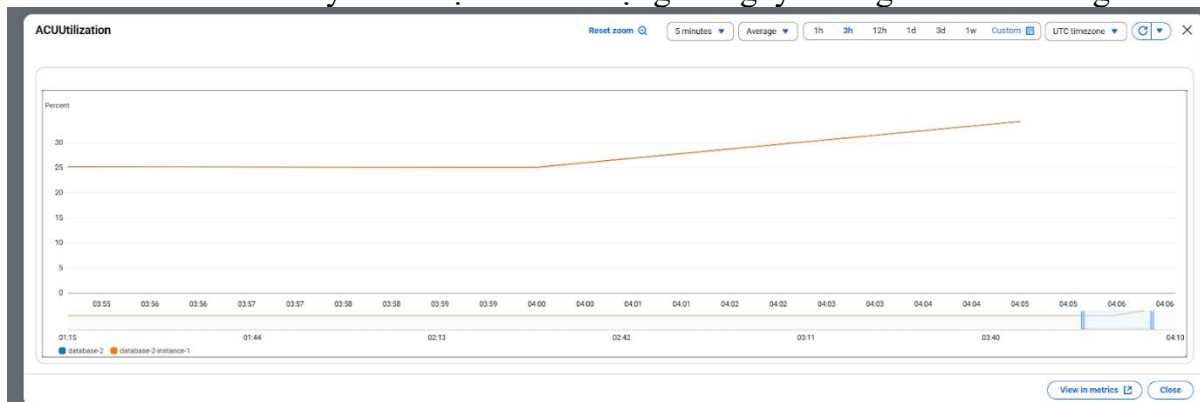
```

4.8 Theo dõi tài nguyên mở rộng (ACUUtilization)

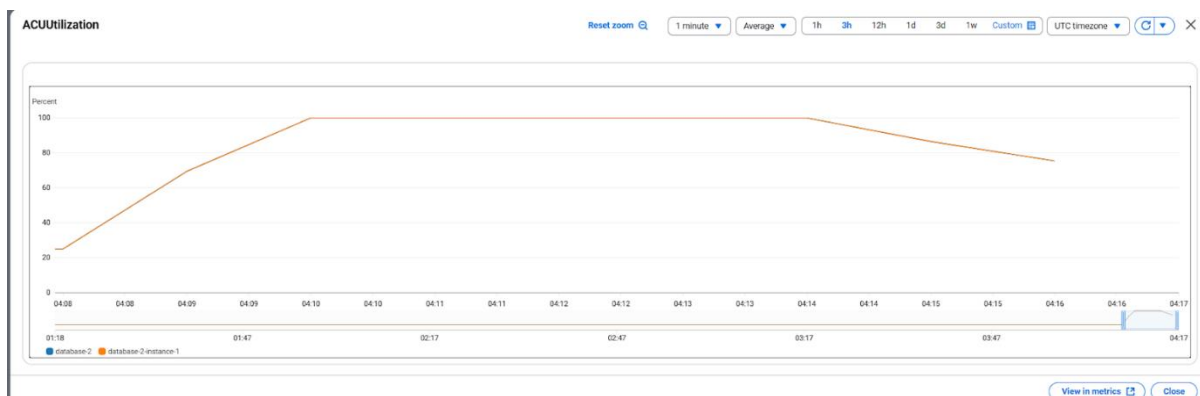
Sau khi thực hiện sinh dữ liệu và mô phỏng truy cập đồng thời bằng thủ tục `simulate_traffic`, hệ thống được theo dõi thông qua biểu đồ ACUUtilization trong phần giám sát của Aurora. Chỉ số này thể hiện mức sử dụng tài nguyên (Aurora Capacity Unit) thực tế tại từng thời điểm.

Ban đầu, mức sử dụng duy trì ở mức thấp (~25%), tuy nhiên khi truy vấn đồng loạt bắt đầu, biểu đồ cho thấy sự gia tăng rõ rệt tài nguyên sử dụng. Aurora Serverless tự động mở rộng để đáp ứng tải, thể hiện đúng chức năng auto-scaling của dịch vụ.

Biểu đồ dưới đây minh họa mức sử dụng tài nguyên tăng dần theo thời gian:



Biểu đồ dưới đây minh họa mức sử dụng tài nguyên giảm dần sau khi hết truy cập:



Từ 04:08 → 04:10: Tải truy cập tăng dần → ACU tăng từ 25% lên 100%.

Từ 04:10 → 04:14: Aurora giữ nguyên tài nguyên ở mức cao để xử lý truy vấn.

Từ 04:14 → 04:17: Tải truy cập giảm → hệ thống tự động scale in, ACU giảm xuống ~70%.

Sau khi hết truy cập hoặc truy cập ít, hệ thống tự thu hẹp tài nguyên để tiết kiệm chi phí, không cần can thiệp thủ công.

Điều này chứng minh Aurora Serverless v2 có khả năng co giãn tài nguyên hiệu quả theo mức độ truy cập thực tế, phù hợp với các hệ thống có lưu lượng không ổn định.

4.9 Kết luận thực nghiệm

Sau quá trình thiết lập môi trường, nhập dữ liệu lớn và thực hiện truy vấn mô phỏng đồng thời, chúng tôi đã tiến hành theo dõi hoạt động của Aurora Serverless v2 để kiểm chứng khả năng tự điều chỉnh tài nguyên.

Các kết quả quan sát được:

- Aurora Serverless hoạt động ổn định trong suốt quá trình sinh dữ liệu lớn và truy vấn đồng thời.
- Hệ thống phản ứng nhanh khi có tải truy cập tăng đột ngột, thể hiện qua biểu đồ ACUUtilization tăng từ 25% lên 100% chỉ trong vài phút.
- Sau khi tải giảm, Aurora tự động thu hẹp tài nguyên để tiết kiệm chi phí vận hành (ACU giảm dần về mức thấp hơn).
- Không cần cấu hình lại hoặc khởi động lại cơ sở dữ liệu – việc mở rộng và thu hẹp được thực hiện hoàn toàn tự động.

Kết luận:

Kết quả thực nghiệm cho thấy Aurora Serverless v2 là lựa chọn phù hợp với các hệ thống có lưu lượng truy cập biến động như hệ thống tra cứu điểm, dịch vụ mùa vụ, hoặc ứng dụng startup giai đoạn đầu. Tính năng tự động mở rộng và thu hẹp giúp cân bằng hiệu suất và chi phí, trong khi vẫn đảm bảo trải nghiệm người dùng được duy trì ổn định.

Ngoài ra, biểu đồ ACUUtilization đã cho thấy một cách trực quan quá trình mở rộng và thu hẹp tài nguyên của hệ thống. Aurora Serverless v2 tự động mở rộng khi có lượng truy vấn lớn, và thu hẹp lại khi không còn nhu cầu sử dụng cao. Điều này khẳng định tính năng tự động cung cấp tài nguyên theo nhu cầu (auto-scaling) hoạt động hoàn toàn chính xác và hiệu quả.

CHƯƠNG 5. TÀI LIỆU THAM KHẢO

1. <https://vinahost.vn/serverless-computing-la-gi/>
2. <https://trungquandev.com/xin-chao-serverless-chung-ta-lam-quen-voi-nhau-nhe/>
3. <https://2coffee.dev/bai-viet/mot-bai-viet-chi-tiet-hon-ve-serverless>
4. <https://aws.amazon.com/vi/serverless/>
5. <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>
6. <https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-serverless.html>
7. <https://learn.microsoft.com/en-us/azure/azure-functions/>
8. <https://viblo.asia/p/kien-truc-serverless-loi-ich-thach-thuc-va-huong-dan-trien-khai-obA46vZw4Kv>