ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN TỐT NGHIỆP

NGÀNH: CÔNG NGHỆ THÔNG TIN

CHUYÊN NGÀNH: CÔNG NGHỆ PHẦN MỀM

ĐỀ TÀI

XÂY DỰNG HỆ THỐNG AUTOTUNER BENCH-MARK VÀ HỆ THỐNG TỐI ƯU TÀI NGUYÊN

Người hướng dẫn: **TS. TRỊNH CÔNG DUY**
Sinh viên thực hiện: **NGUYỄN VIẾT NHẬT LONG**
Số thẻ sinh viên: **102210260**
Lớp**: 21TCLC_DT4**

Đà Nẵng, 01/2026

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN TỐT NGHIỆP

NGÀNH: CÔNG NGHỆ THÔNG TIN

CHUYÊN NGÀNH: CÔNG NGHỆ PHẦN MỀM

ĐỀ TÀI:

XÂY DỰNG HỆ THỐNG AUTOTUNER BENCH-MARK VÀ HỆ THỐNG TỐI ƯU TÀI NGUYÊN

Người hướng dẫn: **TS. TRỊNH CÔNG DUY**
Người duyệt:
Sinh viên thực hiện: **NGUYỄN VIẾT NHẬT LONG**
Số thẻ sinh viên: **102210260**
Lớp**: 21TCLC_DT4**

Đà Nẵng, 01/2026

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

NHẬN XÉT ĐỒ ÁN TỐT NGHIỆP

1. Thông tin chung:

1. Họ và tên sinh viên: NGUYỄN VIẾT NHẬT LONG
2. Lớp: 21TCLC_DT4 Số thẻ SV: 102210260
3. Tên đề tài: HỆ THỐNG
4. Người hướng dẫn: TRỊNH CÔNG DUY Học hàm/ học vị: Tiến sĩ

II. Nhận xét, đánh giá đồ án tốt nghiệp:

1. Về tính cấp thiết, tính mới, khả năng ứng dụng của đề tài: (điểm tối đa là 2đ)

……..
……..

2. Về kết quả giải quyết các nội dung nhiệm vụ yêu cầu của đồ án: (điểm tối đa là 4đ)

……..
……..

3. Về hình thức, cấu trúc, bố cục của đồ án tốt nghiệp: (điểm tối đa là 2đ)

……..
……..

4. Đề tài có giá trị khoa học/ có bài báo/ giải quyết vấn đề đặt ra của doanh nghiệp hoặc nhà trường: (điểm tối đa là 1đ)

……..
……..

5. Các tồn tại, thiếu sót cần bổ sung, chỉnh sửa:

……..
……..

III. Tinh thần, thái độ làm việc của sinh viên: (điểm tối đa 1đ)

……..

IV. Đánh giá:

1. Điểm đánh giá: ……../10 (lấy đến 1 số lẻ thập phân)
2. Đề nghị: ☐ Được bảo vệ đồ án ☐ Bổ sung để bảo vệ ☐ Không được bảo vệ

Đà Nẵng, ngày tháng năm 2026

Người hướng dẫn

TÓM TẮT

Tên đề tài: Xây dựng hệ thống AutoTuner benchmark và đề xuất tối ưu tài nguyên

Sinh viên thực hiện: Nguyễn Viết Nhật Long

Lớp: 21TCLC_DT4 Số thẻ SV: 102210260

Trong bối cảnh điện toán đám mây ngày càng phát triển, việc lựa chọn cấu hình tài nguyên phù hợp cho các ứng dụng phụ trợ (backend) trở thành thách thức lớn đối với các nhà phát triển và kỹ sư vận hành hệ thống. Nền tảng đám mây như Amazon Web Services (AWS) cung cấp hàng trăm loại máy ảo EC2 với các thông số kỹ thuật đa dạng về số lõi xử lý, dung lượng bộ nhớ, kiến trúc vi xử lý và mức giá khác nhau. Việc lựa chọn sai cấu hình có thể dẫn đến lãng phí chi phí hoặc hiệu năng không đáp ứng được yêu cầu của ứng dụng.

Các công cụ đánh giá hiệu năng hiện có như Apache Benchmark, wrk hay k6 chỉ hỗ trợ đo lường từ một điểm duy nhất và không cung cấp khả năng so sánh tự động giữa nhiều cấu hình. Trong khi đó, các công cụ tối ưu của AWS như Compute Optimizer chỉ hoạt động sau khi ứng dụng đã được triển khai và có dữ liệu sử dụng thực tế. Điều này tạo ra khoảng trống trong việc hỗ trợ ra quyết định trước khi triển khai.

Xuất phát từ thực tiễn đó, đề tài "Xây dựng hệ thống AutoTuner benchmark và đề xuất tối ưu tài nguyên" được nghiên cứu và phát triển nhằm giải quyết bài toán tối ưu hóa lựa chọn cấu hình máy chủ đám mây. Mục tiêu cốt lõi của đề tài là xây dựng một nền tảng web hoàn chỉnh cho phép người dùng thực hiện benchmark trên các máy ảo EC2 thực tế, thu thập các chỉ số hiệu năng chi tiết và nhận được khuyến nghị tối ưu từ hệ thống trí tuệ nhân tạo.

Điểm nổi bật của hệ thống AutoTuner so với các giải pháp hiện có bao gồm: (1) Benchmark trên tài nguyên thực tế thay vì mô phỏng, đảm bảo dữ liệu đo lường phản ánh chính xác hiệu năng thực tế; (2) Tích hợp mô hình ngôn ngữ lớn (Large Language Model - LLM) thông qua OpenAI API để phân tích dữ liệu và đưa ra khuyến nghị thông minh; (3) Hỗ trợ phát hiện bất thường tự động trong kết quả benchmark bằng phương pháp thống kê IQR; (4) Thu thập chỉ số sử dụng tài nguyên từ CloudWatch để đánh giá mức độ phù hợp của cấu hình hiện tại.

Về mặt công nghệ, hệ thống được xây dựng với kiến trúc hiện đại bao gồm: phần phụ trợ sử dụng FastAPI và Celery cho xử lý bất đồng bộ, phần giao diện người dùng sử dụng React kết hợp TypeScript, cơ sở dữ liệu PostgreSQL cho lưu trữ có cấu trúc và Redis cho bộ đệm. Hệ thống hỗ trợ triển khai bằng Docker và có thể mở rộng theo chiều ngang.

Kết quả đạt được của đề tài bao gồm một hệ thống hoàn chỉnh với đầy đủ các chức năng: quản lý dự án và cấu hình, thực thi benchmark tự động trên EC2, thu thập và phân tích chỉ số hiệu năng, đề xuất tối ưu bằng trí tuệ nhân tạo và hiển thị kết quả trực quan trên giao diện web.

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

Họ tên sinh viên: Nguyễn Viết Nhật Long. Số thẻ sinh viên: 102210260
Lớp: 21TCLC_DT4 Khoa: Công nghệ thông tin Ngành: Công nghệ phần mềm

1. *Tên đề tài đồ án:* XÂY DỰNG HỆ THỐNG AUTOTUNER BENCHMARK VÀ ĐỀ XUẤT TỐI ƯU TÀI NGUYÊN
2. *Đề tài thuộc diện:*

☐ *Có ký kết thỏa thuận sở hữu trí tuệ đối với kết quả thực hiện*

3. *Các số liệu và dữ liệu ban đầu:*

Không có.

4. *Nội dung các phần thuyết minh và tính toán:*

Nội dung thuyết minh gồm:

Chương 1: Phần mở đầu của luận văn, giới thiệu về nhu cầu thực tế và lý do thực hiện đề tài, đồng thời giới thiệu sơ lược về đề tài và mục tiêu phải đạt được, tính năng và đối tượng sử dụng.

Chương 2: Cơ sở lý thuyết và công nghệ: trình bày những lý thuyết, công nghệ học được và đã áp dụng vào hệ thống.

Chương 3: Phân tích và thiết kế hệ thống: Phân tích yêu cầu chức năng, phi chức năng, mô hình hóa các nghiệp vụ chính của hệ thống, thiết kế cơ sở dữ liệu, thuật toán và api, giao diện người dùng.

Chương 4: Xây dựng và triển khai hệ thống: Mô tả chức năng vận hành hệ thống

Chương 5: Tổng kết kết quả đạt được, hạn chế và hướng phát triển.

5. *Các bản vẽ, đồ thị (ghi rõ các loại và kích thước bản vẽ):*

Không có.

6. *Họ tên người hướng dẫn: TS. Trịnh Công Duy*
7. *Ngày giao nhiệm vụ đồ án: 17/11/2025*
8. *Ngày hoàn thành đồ án: 21/01/2026.*

Đà Nẵng, ngày 20 tháng 01 năm 2026

Trưởng Bộ môn …….. **Người hướng dẫn**

LỜI NÓI ĐẦU

Trong bối cảnh công nghệ thông tin phát triển mạnh mẽ, việc triển khai ứng dụng lên nền tảng điện toán đám mây đã trở thành xu hướng tất yếu. Tuy nhiên, việc lựa chọn cấu hình tài nguyên phù hợp cho các ứng dụng backend luôn là thách thức lớn đối với các nhà phát triển và kỹ sư vận hành hệ thống. Với hàng trăm loại máy ảo EC2 khác nhau về thông số kỹ thuật và mức giá, việc so sánh và lựa chọn thủ công trở nên không khả thi và tốn kém.

Xuất phát từ thực tiễn đó, đề tài "Xây dựng hệ thống AutoTuner benchmark và đề xuất tối ưu tài nguyên" được nghiên cứu và phát triển nhằm giải quyết bài toán tối ưu hóa lựa chọn cấu hình máy chủ đám mây. Hệ thống cho phép người dùng tự động hóa việc benchmark các cấu hình EC2 trên tài nguyên thực tế, thu thập các chỉ số hiệu năng chi tiết và nhận được khuyến nghị tối ưu từ hệ thống trí tuệ nhân tạo.

Đồ án này được thực hiện dưới sự hướng dẫn tận tình của **TS. Trịnh Công Duy** - giảng viên Khoa Công nghệ Thông tin, Trường Đại học Bách Khoa - Đại học Đà Nẵng. Tác giả xin chân thành cảm ơn thầy đã dành thời gian hướng dẫn, đóng góp ý kiến quý báu và tạo điều kiện thuận lợi trong suốt quá trình thực hiện đồ án.

Tác giả cũng xin gửi lời cảm ơn đến các thầy cô trong Khoa Công nghệ Thông tin đã truyền đạt kiến thức nền tảng, giúp tác giả có đủ nền tảng để hoàn thành đồ án này.

Mặc dù đã cố gắng hết sức, đồ án không tránh khỏi những thiếu sót. Tác giả rất mong nhận được những ý kiến đóng góp quý báu từ các thầy cô và bạn bè để đồ án được hoàn thiện hơn.

Em xin chân thành cảm ơn!

CAM ĐOAN

Tôi xin cam đoan:

1. Đồ án tốt nghiệp "Xây dựng hệ thống AutoTuner benchmark và đề xuất tối ưu tài nguyên" là công trình nghiên cứu của chính bản thân tôi dưới sự hướng dẫn của **TS. Trịnh Công Duy**.
2. Tất cả các kết quả, số liệu, hình ảnh, biểu đồ và các nội dung khác trong đồ án này đều trung thực, chính xác và có nguồn gốc rõ ràng.
3. Các tài liệu, công trình nghiên cứu của các tác giả khác được tham khảo trong đồ án đều được trích dẫn và ghi rõ nguồn trong phần Tài liệu tham khảo.
4. Tôi hoàn toàn chịu trách nhiệm về tính chính xác và trung thực của đồ án này.
5. Tôi không sao chép hoặc sử dụng các công trình nghiên cứu của người khác mà không được phép.

Nếu phát hiện có bất kỳ vi phạm nào về tính trung thực trong đồ án, tôi xin chịu hoàn toàn trách nhiệm trước Hội đồng đánh giá đồ án tốt nghiệp.

Đà Nẵng, ngày tháng năm 2026

Sinh viên thực hiện

Nguyễn Viết Nhật Long

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI

1.1. Tổng quan đề tài

1.1.1. Bối cảnh

Trong thời đại chuyển đổi số, điện toán đám mây (Cloud Computing) đã trở thành nền tảng quan trọng cho việc phát triển và triển khai ứng dụng. Amazon Web Services (AWS) là một trong những nhà cung cấp dịch vụ đám mây hàng đầu thế giới, cung cấp hơn 200 loại máy ảo EC2 với các thông số kỹ thuật đa dạng về kiến trúc vi xử lý (x86_64, ARM64/Graviton), số lõi CPU, dung lượng bộ nhớ, hiệu năng mạng và mức giá khác nhau.

Việc lựa chọn cấu hình máy ảo phù hợp có ảnh hưởng trực tiếp đến:

- **Hiệu năng ứng dụng:** Latency, throughput, response time
- **Chi phí vận hành:** Chi phí có thể dao động từ vài đô la đến hàng trăm đô la mỗi tháng
- **Trải nghiệm người dùng:** Hiệu năng kém dẫn đến trải nghiệm người dùng kém
- **Khả năng mở rộng:** Cấu hình không phù hợp có thể gây khó khăn khi scale

1.1.2. Vấn đề thực tế

Khi triển khai các ứng dụng backend lên AWS, các nhà phát triển và kỹ sư vận hành hệ thống thường gặp phải các thách thức:

Thách thức 1: Quá nhiều lựa chọn

AWS cung cấp hàng trăm loại máy ảo EC2 với các họ khác nhau:

- **General Purpose:** t3, t4g, m6g, m7g (cân bằng CPU, memory, network)
- **Compute Optimized:** c6g, c7g (tối ưu cho CPU-intensive workloads)
- **Memory Optimized:** r6g, r7g (tối ưu cho memory-intensive workloads)
- **Storage Optimized:** i3, i4i (tối ưu cho I/O-intensive workloads)

Mỗi họ lại có nhiều kích thước từ nano đến 24xlarge, tạo ra hàng trăm lựa chọn. Việc so sánh thủ công giữa các lựa chọn này là không khả thi.

Thách thức 2: Thiếu dữ liệu đo lường thực tế

Hầu hết các quyết định hiện tại dựa trên:

- Tài liệu kỹ thuật từ AWS
- Kinh nghiệm cá nhân
- Ước tính chi phí từ AWS Pricing Calculator

Tuy nhiên, hiệu năng thực tế của ứng dụng phụ thuộc vào nhiều yếu tố:

- Đặc điểm workload (CPU-intensive, memory-intensive, I/O-intensive)

- Kiến trúc ứng dụng (monolithic, microservices)
- Vùng địa lý và điều kiện mạng
- Thời điểm sử dụng (peak hours, off-peak hours)

Không có cách nào để biết chính xác một cấu hình sẽ hoạt động như thế nào cho đến khi triển khai thực tế.

Thách thức 3: Chi phí thử nghiệm cao

Việc triển khai và kiểm thử nhiều cấu hình khác nhau trên AWS có thể tốn kém:

- Mỗi lần thử nghiệm có thể mất hàng giờ để thiết lập và đo lường
- Chi phí có thể lên đến hàng chục đô la Mỹ cho một lần so sánh đầy đủ
- Rủi ro quên dọn dẹp tài nguyên dẫn đến chi phí phát sinh

Thách thức 4: Công cụ hiện có có hạn chế

Các công cụ benchmark hiện có (Apache Benchmark, wrk, k6) chỉ hỗ trợ đo lường từ một điểm duy nhất, không phản ánh hiệu năng thực tế từ các vùng địa lý khác nhau. Các công cụ tối ưu của AWS như Compute Optimizer chỉ hoạt động sau khi ứng dụng đã được triển khai và có dữ liệu sử dụng trong ít nhất 14 ngày.

Thách thức 5: Thiếu khuyến nghị thông minh

Ngay cả khi có dữ liệu benchmark, việc phân tích và đưa ra quyết định tối ưu vẫn đòi hỏi:

- Chuyên môn sâu về kiến trúc hệ thống
- Hiểu biết về đặc điểm workload
- Khả năng cân bằng giữa nhiều tiêu chí (latency, cost, reliability)

1.1.3. Giải pháp đề xuất

Đề tài "Xây dựng hệ thống AutoTuner benchmark và đề xuất tối ưu tài nguyên" được nghiên cứu và phát triển nhằm giải quyết các thách thức trên bằng cách:

1. **Tự động hóa quy trình benchmark:** Triển khai các máy ảo EC2 thực tế, thực thi benchmark với workload có thể cấu hình, và tự động dọn dẹp tài nguyên.
2. **Thu thập metrics toàn diện:** Đo lường latency ở nhiều phân vị (P50, P95, P99), phát hiện cold start, thu thập chỉ số sử dụng tài nguyên từ CloudWatch, và phát hiện bất thường trong kết quả.
3. **Đề xuất tối ưu bằng AI:** Sử dụng mô hình ngôn ngữ lớn (LLM) để phân tích kết quả benchmark và đưa ra khuyến nghị tối ưu với giải thích chi tiết.
4. **Giao diện trực quan:** Hiển thị kết quả dưới dạng biểu đồ và bảng so

sánh, giúp người dùng dễ dàng đưa ra quyết định.

1.2. Lý do chọn đề tài

1.2.1. Tính cấp thiết và thực tiễn

Tính cấp thiết:

Trong bối cảnh doanh nghiệp ngày càng chuyển dịch lên đám mây, việc tối ưu hóa chi phí và hiệu năng trở thành ưu tiên hàng đầu. Theo báo cáo của Gartner, chi phí đám mây có thể chiếm đến 30% tổng chi phí IT của doanh nghiệp. Việc lựa chọn sai cấu hình có thể dẫn đến:

- Lãng phí hàng nghìn đô la mỗi tháng
- Hiệu năng không đáp ứng được yêu cầu
- Trải nghiệm người dùng kém

Tính thực tiễn:

Đề tài giải quyết một vấn đề thực tế mà các nhà phát triển và kỹ sư vận hành hệ thống đang gặp phải hàng ngày. Hệ thống có thể được áp dụng ngay vào thực tế để:

- Tối ưu hóa chi phí cho các dự án đang chạy
- Đánh giá và so sánh các cấu hình trước khi triển khai
- Phát hiện over-provisioning hoặc under-provisioning

1.2.2. Tính mới và sáng tạo

Tính mới:

Mặc dù có nhiều công cụ benchmark và tối ưu hóa hiện có, nhưng không có công cụ nào kết hợp được:

- Benchmark trên tài nguyên thực tế (không phải mô phỏng)
- So sánh đa cấu hình tự động
- Đề xuất tối ưu bằng trí tuệ nhân tạo
- Tích hợp CloudWatch metrics để đánh giá toàn diện

Tính sáng tạo:

Hệ thống sử dụng các kỹ thuật mới:

LLM (Large Language Model): Sử dụng OpenAI GPT để phân tích dữ liệu và đưa ra khuyến nghị thông minh, thay vì các quy tắc cố định

Statistical Anomaly Detection: Sử dụng phương pháp IQR để phát hiện bất thường trong kết quả benchmark

Ephemeral Infrastructure: Tự động triển khai và dọn dẹp tài nguyên, giảm thiểu chi phí

1.2.3. Phù hợp với chuyên ngành

Đề tài phù hợp với chuyên ngành **Công nghệ Phần mềm** vì:

1. **Kiến trúc phần mềm:** Thiết kế và xây dựng hệ thống phân tán với micro-services architecture
2. **Công nghệ web:** Phát triển full-stack application với React và FastAPI
3. **Database:** Thiết kế và tối ưu hóa cơ sở dữ liệu với PostgreSQL
4. **DevOps:** Tích hợp Docker, CI/CD, và cloud infrastructure
5. **Trí tuệ nhân tạo:** Ứng dụng AI/ML trong phân tích dữ liệu và đề xuất tối ưu

1.2.4. Khả năng ứng dụng

Hệ thống có khả năng ứng dụng cao trong thực tế:

1. **Doanh nghiệp:** Các công ty đang sử dụng AWS có thể sử dụng hệ thống để tối ưu hóa chi phí và hiệu năng
2. **Nhà phát triển:** Các developer có thể sử dụng để đánh giá và so sánh các cấu hình trước khi triển khai
3. **DevOps Engineers:** Các kỹ sư vận hành có thể sử dụng để monitor và optimize infrastructure
4. **Startups:** Các startup có thể sử dụng để tối ưu hóa chi phí trong giai đoạn đầu

1.2.5. Hứng thú và đam mê

Đề tài kết hợp nhiều lĩnh vực mà tác giả quan tâm:

- **Cloud Computing:** Hiểu sâu về AWS và cloud infrastructure
- **Performance Engineering:** Đo lường và tối ưu hóa hiệu năng
- **Artificial Intelligence:** Ứng dụng AI trong giải quyết vấn đề thực tế
- **Full-stack Development:** Xây dựng hệ thống hoàn chỉnh từ backend đến frontend

Đề tài cũng mang lại cơ hội học hỏi và áp dụng các công nghệ mới nhất, giúp tác giả nâng cao kỹ năng và kiến thức trong lĩnh vực công nghệ thông tin.

1.2.6. Mục tiêu và nhiệm vụ đề tài

a) Mục tiêu tổng quát

Xây dựng một hệ thống web hoàn chỉnh cho phép người dùng tự động hóa việc benchmark các cấu hình máy ảo EC2 trên AWS và nhận được khuyến nghị tối ưu dựa trên phân tích dữ liệu thực tế bằng trí tuệ nhân tạo.

b) Mục tiêu cụ thể

Xây dựng hệ thống benchmark tự động trên tài nguyên thực tế

- Triển khai tự động các máy ảo EC2 trong tài khoản AWS của người dùng
- Thực thi benchmark với workload có thể cấu hình
- Thu thập các chỉ số hiệu năng chi tiết (latency, throughput, error rate)

- Tự động dọn dẹp tài nguyên sau khi hoàn thành

Thu thập và phân tích chỉ số hiệu năng toàn diện

- Đo lường latency ở nhiều phân vị (P50, P95, P99)
- Phát hiện cold start và đo lường warm performance
- Thu thập chỉ số sử dụng tài nguyên từ CloudWatch (CPU, disk I/O, network I/O)
- Phát hiện bất thường trong kết quả benchmark bằng phương pháp thống kê

Tích hợp trí tuệ nhân tạo để đề xuất tối ưu

- Sử dụng mô hình ngôn ngữ lớn (LLM) để phân tích đặc điểm workload
- Đề xuất các loại máy ảo thay thế dựa trên kết quả benchmark baseline
- Cung cấp giải thích chi tiết về lý do đề xuất
- Ước tính cải thiện hiệu năng và tiết kiệm chi phí

Xây dựng giao diện người dùng trực quan

- Quản lý dự án và cấu hình benchmark
- Hiển thị kết quả benchmark dưới dạng biểu đồ và bảng so sánh
- Trình bày khuyến nghị với giải thích rõ ràng
- Theo dõi tiến trình benchmark real-time

Đảm bảo kiểm soát chi phí và hiệu quả

- Giới hạn ngân sách cho mỗi lần benchmark
- Tái sử dụng kết quả benchmark đã lưu (caching)
- Tự động dọn dẹp tài nguyên để tránh chi phí phát sinh
- Ước tính chi phí trước khi thực thi

1.2.7. Phạm vi nghiên cứu

a) Phạm vi chức năng

Về dịch vụ AWS:

- Hỗ trợ Amazon EC2 (Elastic Compute Cloud) - máy ảo
- Hỗ trợ các vùng AWS phổ biến: us-east-1, us-east-2, us-west-1, us-west-2, eu-west-1, eu-central-1, ap-southeast-1, ap-northeast-1
- Hỗ trợ các loại máy ảo phổ biến: t3, t4g, m6g, m7g, c6g, c7g, r6g, r7g và các biến thể x86_64 tương ứng

Về chức năng:

- Quản lý người dùng và xác thực (JWT)
- Quản lý dự án và cấu hình AWS
- Thực thi benchmark trên máy ảo EC2 thực tế
- Thu thập chỉ số từ CloudWatch
- Phân tích dữ liệu và phát hiện bất thường

- Đề xuất tối ưu bằng AI (OpenAI GPT)
- Hiển thị kết quả và khuyến nghị trên giao diện web

Về chế độ hoạt động:

- **Optimization Mode**: Phân tích máy ảo hiện có và đề xuất các lựa chọn tối ưu hơn

b) Phạm vi công nghệ

Backend:

- Python 3.11+ với FastAPI cho REST API
- Celery + Redis cho xử lý tác vụ bất đồng bộ
- SQLAlchemy ORM với PostgreSQL cho cơ sở dữ liệu
- boto3 (AWS SDK) cho tương tác với AWS
- OpenAI API cho khuyến nghị AI

Frontend:

- React 18 với TypeScript
- TanStack Query cho quản lý state và data fetching
- Tailwind CSS cho styling
- Recharts cho biểu đồ

DevOps:

- Docker và Docker Compose cho containerization
- Alembic cho database migrations

1.2.8. Đối tượng và phương pháp nghiên cứu

a) Đối tượng nghiên cứu

- Nghiên cứu các phương pháp đo lường hiệu năng (performance benchmarking)
- Nghiên cứu các kỹ thuật phát hiện bất thường trong dữ liệu (anomaly detection)
- Nghiên cứu ứng dụng trí tuệ nhân tạo trong tối ưu hóa tài nguyên đám mây
- Nghiên cứu kiến trúc hệ thống phân tán và xử lý bất đồng bộ
- Nghiên cứu AWS EC2, CloudWatch và các dịch vụ liên quan

b) Phương pháp nghiên cứu

- Phát triển hệ thống theo mô hình MVP (Minimum Viable Product)
- Sử dụng phương pháp Agile với các sprint ngắn
- Kiểm thử từng thành phần (unit testing) và kiểm thử tích hợp (integration testing)
- Đánh giá hiệu quả thông qua các case study thực tế
- So sánh kết quả với các công cụ benchmark truyền thống

1.3. Ý nghĩa thực tiễn

1.3.1. Ý nghĩa về mặt kinh tế

Tiết kiệm chi phí cho doanh nghiệp:

Hệ thống AutoTuner giúp doanh nghiệp tiết kiệm chi phí đáng kể thông qua:

1. **Tối ưu hóa cấu hình tài nguyên:**
   - Phát hiện over-provisioning (cấu hình quá lớn so với nhu cầu)
   - Đề xuất downsizing để giảm chi phí mà vẫn đảm bảo hiệu năng
   - Ví dụ: Chuyển từ t3.large ($60/tháng) sang t3.medium ($30/tháng) khi CPU utilization < 20% có thể tiết kiệm $30/tháng
2. **Giảm chi phí thử nghiệm:**
   - Thay vì phải triển khai và test nhiều cấu hình thủ công (tốn hàng chục đô la), hệ thống tự động hóa quá trình này với chi phí < $0.01 cho 5 cấu hình
   - Tự động cleanup tài nguyên, tránh chi phí phát sinh do quên dọn dẹp
3. **Tối ưu hóa chi phí dài hạn:**
   - Đề xuất sử dụng Spot instances (tiết kiệm 60-70% chi phí)
   - Đề xuất chuyển sang ARM-based Graviton instances (tiết kiệm 20-40% chi phí)
   - Phân tích workload để đề xuất Reserved Instances hoặc Savings Plans

ROI (Return on Investment):

Với chi phí benchmark < $0.01 và tiềm năng tiết kiệm hàng chục đến hàng trăm đô la mỗi tháng, ROI của hệ thống là rất cao. Một doanh nghiệp có thể tiết kiệm hàng nghìn đô la mỗi năm chỉ bằng việc tối ưu hóa một vài cấu hình.

1.3.2. Ý nghĩa về mặt kỹ thuật

Nâng cao hiệu năng ứng dụng:

1. **Phát hiện under-provisioning:**
   - Phát hiện khi cấu hình hiện tại không đủ mạnh (CPU > 80%)
   - Đề xuất upsizing để cải thiện hiệu năng
   - Giảm latency và tăng throughput
2. **Đảm bảo hiệu năng ổn định:**
   - Phát hiện high variance trong kết quả benchmark
   - Cảnh báo về độ tin cậy của cấu hình
   - Đề xuất các cấu hình có hiệu năng ổn định hơn
3. **Tối ưu hóa cho workload cụ thể:**
   - Phân tích đặc điểm workload (CPU-intensive, memory-intensive, I/O-intensive)

- Đề xuất instance families phù hợp (compute-optimized, memory-optimized, etc.)
- Cải thiện hiệu năng đáng kể so với cấu hình generic

Cải thiện quy trình phát triển:

1. **Tự động hóa quy trình benchmark:**
   - Giảm thời gian từ hàng giờ xuống vài phút
   - Loại bỏ công việc thủ công, giảm lỗi
   - Có thể tích hợp vào CI/CD pipeline
2. **Dữ liệu đo lường chính xác:**
   - Benchmark trên tài nguyên thực tế, không phải mô phỏng
   - Metrics toàn diện (latency, CPU, disk I/O, network I/O)
   - Dữ liệu đáng tin cậy để ra quyết định
3. **Hỗ trợ ra quyết định:**
   - So sánh trực quan giữa nhiều cấu hình
   - AI recommendations với giải thích chi tiết
   - Giảm thời gian nghiên cứu và đánh giá

1.3.3. Ý nghĩa về mặt xã hội

Góp phần phát triển cộng đồng:

1. **Mã nguồn mở:**
   - Hệ thống có thể được phát hành dưới dạng mã nguồn mở
   - Cộng đồng developer có thể sử dụng và đóng góp
   - Thúc đẩy sự phát triển của các công cụ tối ưu hóa đám mây
2. **Giáo dục và đào tạo:**
   - Tài liệu và code có thể được sử dụng làm tài liệu học tập
   - Sinh viên có thể học hỏi về cloud optimization, AI integration, và full-stack development
   - Case studies có thể được sử dụng trong giảng dạy
3. **Hỗ trợ doanh nghiệp vừa và nhỏ:**
   - Doanh nghiệp vừa và nhỏ thường thiếu nguồn lực để tối ưu hóa infrastructure
   - Hệ thống giúp họ tiếp cận các công cụ tối ưu hóa với chi phí thấp
   - Góp phần giảm chi phí vận hành, tăng khả năng cạnh tranh

1.3.4. Ứng dụng trong thực tế

Các trường hợp sử dụng cụ thể:

1. **Tối ưu hóa API Backend:**
   - Các công ty có API backend chạy trên EC2
   - Cần tối ưu hóa chi phí và hiệu năng

- Ví dụ: E-commerce, SaaS applications, REST APIs

2. **Migration và Modernization:**
   - Các công ty đang migrate lên cloud
   - Cần đánh giá và so sánh các cấu hình trước khi migrate
   - Đảm bảo hiệu năng và chi phí phù hợp
3. **Cost Optimization Projects:**
   - Các dự án tối ưu hóa chi phí đám mây
   - Phát hiện over-provisioning và under-provisioning
   - Đề xuất các giải pháp tiết kiệm chi phí
4. **Performance Tuning:**
   - Các ứng dụng có vấn đề về hiệu năng
   - Cần tìm cấu hình tối ưu để cải thiện latency
   - Đảm bảo SLA và trải nghiệm người dùng
5. **Proof of Concept (POC):**
   - Đánh giá các cấu hình khác nhau trước khi triển khai production
   - So sánh chi phí và hiệu năng
   - Ra quyết định dựa trên dữ liệu thực tế

1.3.5. Đóng góp cho ngành công nghiệp

Thúc đẩy best practices:

1. **Data-driven decision making:**
   - Khuyến khích ra quyết định dựa trên dữ liệu thực tế thay vì phỏng đoán
   - Nâng cao chất lượng quyết định về infrastructure
2. **Automation và DevOps:**
   - Thúc đẩy tự động hóa trong quy trình quản lý infrastructure
   - Tích hợp với các công cụ DevOps hiện có
   - Giảm toil và tăng productivity
3. **AI/ML trong Cloud Optimization:**
   - Chứng minh khả năng ứng dụng AI trong tối ưu hóa đám mây
   - Mở ra hướng nghiên cứu mới về AI-powered infrastructure optimization
   - Kết hợp giữa domain knowledge và AI

Nghiên cứu và phát triển:

1. **Mở rộng nghiên cứu:**
   - Có thể mở rộng sang multi-cloud optimization
   - Nghiên cứu về workload profiling và prediction
   - Phát triển các thuật toán tối ưu hóa mới
2. **Cộng tác với AWS và các nhà cung cấp:**

- Có thể hợp tác với AWS để tích hợp vào các dịch vụ của họ
- Đóng góp vào cộng đồng AWS và cloud computing

1.3.6. Tác động lâu dài

Về mặt môi trường:

1. **Giảm lãng phí tài nguyên:**
   - Tối ưu hóa sử dụng tài nguyên giúp giảm lãng phí
   - Góp phần bảo vệ môi trường thông qua việc sử dụng hiệu quả năng lượng

Về mặt kinh tế:

2. **Tăng năng suất:**
   - Tự động hóa giúp tiết kiệm thời gian
   - Nhân viên có thể tập trung vào các công việc có giá trị cao hơn
3. **Tăng khả năng cạnh tranh:**
   - Doanh nghiệp có thể giảm chi phí và tăng hiệu năng
   - Tăng khả năng cạnh tranh trên thị trường

Về mặt công nghệ:

1. **Thúc đẩy đổi mới:**
   - Khuyến khích sử dụng các công nghệ mới (AI, automation)
   - Thúc đẩy sự phát triển của ngành công nghiệp
2. **Nâng cao kỹ năng:**
   - Developer và DevOps engineers học hỏi về cloud optimization
   - Nâng cao kỹ năng và kiến thức trong lĩnh vực

1.4. Bố cục đồ án

Đồ án được tổ chức thành 6 chương chính:

- **Chương 1: Giới thiệu** - Trình bày vấn đề, mục tiêu, phạm vi và phương pháp nghiên cứu.
- **Chương 2: Tổng quan** - Khảo sát các công cụ benchmark và tối ưu hóa hiện có, phân tích khoảng trống và định vị giải pháp.
- **Chương 3: Phân tích và thiết kế hệ thống** - Thiết kế kiến trúc tổng thể, cơ sở dữ liệu, API và các thành phần AI.
- **Chương 4: Xây dựng và triển khai hệ thống** - Chi tiết triển khai backend, frontend, tích hợp AWS và AI services.
- **Chương 5: Kiểm thử và đánh giá** - Trình bày kết quả kiểm thử, đánh giá hiệu năng và độ chính xác của hệ thống.
- **Chương 6: Kết luận và hướng phát triển** - Tổng kết kết quả, hạn chế và đề xuất hướng phát triển trong tương lai.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT VÀ CÔNG NGHỆ SỬ DỤNG

2.1. Khảo sát các công cụ benchmark hiện có

2.1.1. Apache Benchmark (ab)

Apache Benchmark là công cụ đo lường hiệu năng HTTP phổ biến, được phát triển bởi Apache Software Foundation. Công cụ này cho phép gửi một số lượng lớn yêu cầu HTTP đến một máy chủ và đo lường thời gian phản hồi.

Ưu điểm:

- Đơn giản, dễ sử dụng, không cần cài đặt phức tạp
- Nhanh chóng trong việc thực thi benchmark cơ bản
- Hỗ trợ các phương thức HTTP phổ biến (GET, POST)
- Có sẵn trên hầu hết các hệ điều hành Unix/Linux

Hạn chế:

- Chỉ hỗ trợ đo lường từ một điểm duy nhất, không phản ánh hiệu năng thực tế từ nhiều vùng địa lý
- Không hỗ trợ so sánh tự động giữa nhiều cấu hình
- Không có khả năng phân tích dữ liệu và đưa ra khuyến nghị
- Chỉ cung cấp các chỉ số cơ bản (requests per second, time per request)
- Không hỗ trợ đo lường các chỉ số chi tiết như P95, P99 latency

2.1.2. wrk

wrk là công cụ benchmark HTTP hiện đại hơn, được viết bằng C và hỗ trợ scripting bằng Lua. Công cụ này cho phép tạo ra tải cao với số lượng kết nối đồng thời lớn.

Ưu điểm:

- Hiệu năng cao, có thể tạo ra hàng trăm nghìn requests per second
- Hỗ trợ scripting để tùy chỉnh workload
- Cung cấp các chỉ số chi tiết hơn so với Apache Benchmark
- Hỗ trợ HTTP pipelining

Hạn chế:

- Vẫn chỉ đo lường từ một điểm duy nhất
- Không có khả năng so sánh tự động giữa nhiều cấu hình
- Không tích hợp với các dịch vụ đám mây để tự động triển khai và đo lường
- Yêu cầu kiến thức về Lua để tận dụng tối đa tính năng scripting

2.1.3. k6

k6 là công cụ load testing hiện đại được phát triển bởi Grafana Labs, sử dụng JavaScript để viết test scripts. Công cụ này được thiết kế cho DevOps và kỹ sư hiệu năng.

Ưu điểm:

- Hỗ trợ scripting mạnh mẽ bằng JavaScript (ES6+)
- Cung cấp các chỉ số chi tiết (P50, P95, P99, mean, min, max)
- Hỗ trợ virtual users (VUs) để mô phỏng tải thực tế
- Có thể tích hợp với các hệ thống CI/CD
- Hỗ trợ cloud execution thông qua k6 Cloud

Hạn chế:

- Vẫn cần thiết lập thủ công các máy chủ đích để benchmark
- Không tự động so sánh giữa nhiều cấu hình
- Không có khả năng đề xuất tối ưu tự động
- Chi phí sử dụng k6 Cloud có thể cao cho các test lớn

2.1.4. Locust

Locust là framework load testing mã nguồn mở, cho phép viết test scenarios bằng Python. Công cụ này hỗ trợ distributed testing với nhiều worker nodes.

Ưu điểm:

- Viết test scenarios bằng Python, dễ dàng tùy chỉnh
- Hỗ trợ distributed testing để tạo tải lớn
- Giao diện web để theo dõi test real-time
- Cung cấp các chỉ số chi tiết

Hạn chế:

- Vẫn cần thiết lập thủ công infrastructure để chạy test
- Không tự động triển khai và so sánh nhiều cấu hình
- Không có khả năng đề xuất tối ưu

2.2. Khảo sát các công cụ tối ưu hóa của AWS

2.2.1. AWS Compute Optimizer

AWS Compute Optimizer là dịch vụ của AWS sử dụng machine learning để phân tích lịch sử sử dụng tài nguyên và đề xuất tối ưu hóa cho EC2 instances, Auto Scaling groups, EBS volumes và Lambda functions.

Ưu điểm:

- Sử dụng machine learning để phân tích dữ liệu thực tế
- Đề xuất dựa trên lịch sử sử dụng trong ít nhất 14 ngày
- Hỗ trợ nhiều loại tài nguyên AWS
- Miễn phí sử dụng

Hạn chế:

- Chỉ hoạt động sau khi ứng dụng đã được triển khai và có dữ liệu sử dụng
- Không hỗ trợ so sánh giữa các dịch vụ khác nhau (ví dụ: EC2 vs Lambda)
- Không có benchmark thực tế, chỉ dựa trên dữ liệu CloudWatch
- Không hỗ trợ so sánh trước khi quyết định triển khai
- Đề xuất có thể không phù hợp với workload mới hoặc thay đổi

2.2.2. AWS Cost Explorer

AWS Cost Explorer là công cụ của AWS cho phép phân tích chi phí sử dụng các dịch vụ AWS theo thời gian, dịch vụ, vùng và các chiều khác.

Ưu điểm:

- Cung cấp phân tích chi tiết về chi phí
- Hỗ trợ dự báo chi phí trong tương lai
- Giao diện trực quan với biểu đồ và bảng
- Có thể xuất báo cáo

Hạn chế:

- Chỉ tập trung vào chi phí, không có dữ liệu về hiệu năng
- Không có khả năng đề xuất tối ưu tự động
- Chỉ hiển thị dữ liệu sau khi đã sử dụng dịch vụ
- Không hỗ trợ so sánh giữa các cấu hình trước khi triển khai

2.2.3. AWS Pricing Calculator

AWS Pricing Calculator là công cụ ước tính chi phí cho các dịch vụ AWS dựa trên cấu hình người dùng nhập vào.

Ưu điểm:

- Miễn phí và dễ sử dụng
- Hỗ trợ ước tính cho nhiều dịch vụ AWS
- Có thể lưu và chia sẻ ước tính

Hạn chế:

- Chỉ ước tính chi phí, không có dữ liệu về hiệu năng
- Ước tính có thể không chính xác do không tính đến các yếu tố như data transfer, storage I/O
- Không có khả năng benchmark thực tế
- Không đưa ra khuyến nghị tự động

2.2.4. AWS Well-Architected Tool

AWS Well-Architected Tool là framework đánh giá kiến trúc hệ thống dựa trên 5 trụ cột: Operational Excellence, Security, Reliability, Performance Efficiency và Cost Optimization.

Ưu điểm:

- Đánh giá toàn diện về kiến trúc hệ thống
- Cung cấp best practices từ AWS
- Hỗ trợ nhiều workload types

Hạn chế:

- Không có benchmark thực tế
- Không so sánh cụ thể giữa các dịch vụ
- Không có khuyến nghị dựa trên dữ liệu đo lường
- Yêu cầu người dùng tự trả lời câu hỏi, có thể chủ quan

2.3. Khoảng trống trong các giải pháp hiện có

Sau khi khảo sát các công cụ hiện có, có thể nhận thấy các khoảng trống sau:

Khoảng trống 1: Thiếu benchmark trên tài nguyên thực tế trước khi quyết định

Hầu hết các công cụ benchmark hiện có (ab, wrk, k6, Locust) chỉ hỗ trợ đo lường từ một điểm duy nhất và yêu cầu người dùng tự thiết lập infrastructure. Không có công cụ nào tự động triển khai các máy ảo EC2, thực thi benchmark và so sánh kết quả. Điều này khiến việc so sánh nhiều cấu hình trở nên tốn kém và mất thời gian.

Khoảng trống 2: Thiếu so sánh đa cấu hình tự động

Các công cụ benchmark truyền thống chỉ đo lường một cấu hình tại một thời điểm. Để so sánh nhiều cấu hình, người dùng phải thực hiện thủ công nhiều lần và tự tổng hợp kết quả. Không có công cụ nào tự động so sánh và xếp hạng các cấu hình dựa trên nhiều tiêu chí.

Khoảng trống 3: Thiếu khuyến nghị thông minh dựa trên AI

AWS Compute Optimizer sử dụng machine learning nhưng chỉ hoạt động sau khi đã có dữ liệu sử dụng. Không có công cụ nào sử dụng trí tuệ nhân tạo để phân tích kết quả benchmark và đưa ra khuyến nghị tối ưu với giải thích chi tiết. Các đề xuất hiện tại chủ yếu dựa trên quy tắc cố định hoặc kinh nghiệm.

Khoảng trống 4: Thiếu tích hợp giữa benchmark và tối ưu hóa

Các công cụ benchmark và tối ưu hóa hiện tại hoạt động độc lập. Không có giải pháp nào tích hợp cả hai: thực thi benchmark trên tài nguyên thực tế và đưa ra khuyến nghị tối ưu dựa trên kết quả đo lường.

Khoảng trống 5: Thiếu kiểm soát chi phí trong quá trình benchmark

Việc thử nghiệm nhiều cấu hình có thể tốn kém, đặc biệt khi sử dụng các máy ảo lớn. Không có công cụ nào giới hạn ngân sách, tái sử dụng kết quả đã đo lường hoặc tối ưu hóa số lượng thử nghiệm cần thiết.

2.4. Điểm khác biệt của đề tài

Hệ thống AutoTuner được thiết kế để lấp đầy các khoảng trống trên với các điểm khác biệt sau:

Điểm khác biệt 1: Benchmark trên tài nguyên thực tế

Thay vì mô phỏng hoặc ước tính, hệ thống AutoTuner triển khai các máy ảo EC2 thực tế trong tài khoản AWS của người dùng, thực thi benchmark với workload thực tế và tự động dọn dẹp sau khi hoàn thành. Điều này đảm bảo dữ liệu đo lường phản ánh chính xác hiệu năng thực tế.

Điểm khác biệt 2: So sánh đa cấu hình tự động

Hệ thống cho phép người dùng chọn nhiều cấu hình để benchmark, tự động thực thi tuần tự hoặc song song, và so sánh kết quả trên một giao diện thống nhất. Kết quả được xếp hạng dựa trên nhiều tiêu chí (latency, cost, reliability).

Điểm khác biệt 3: Khuyến nghị thông minh bằng AI

Hệ thống tích hợp OpenAI GPT để phân tích kết quả benchmark, hiểu đặc điểm workload và đưa ra khuyến nghị tối ưu với giải thích chi tiết. AI có thể xem xét nhiều yếu tố phức tạp mà các quy tắc cố định không thể bao quát.

Điểm khác biệt 4: Tích hợp CloudWatch để đánh giá toàn diện

Hệ thống không chỉ đo lường latency từ phía client mà còn thu thập các chỉ số từ CloudWatch như CPU utilization, disk I/O, network I/O để đánh giá mức độ phù hợp của cấu hình hiện tại (under-provisioned, optimized, over-provisioned).

Điểm khác biệt 5: Kiểm soát chi phí và tối ưu hóa

Hệ thống hỗ trợ giới hạn ngân sách, tái sử dụng kết quả benchmark đã lưu (caching), và tự động dọn dẹp tài nguyên để tránh chi phí phát sinh. Người dùng có thể ước tính chi phí trước khi thực thi.

Điểm khác biệt 6: Phát hiện bất thường tự động

Hệ thống sử dụng phương pháp thống kê (IQR - Interquartile Range) để phát hiện các giá trị bất thường trong kết quả benchmark, loại bỏ chúng khỏi tính toán và cảnh báo người dùng về độ tin cậy của kết quả.

2.5. Các công nghệ liên quan

2.5.1. FastAPI

FastAPI là framework web hiện đại cho Python, được thiết kế để xây dựng API nhanh chóng với hiệu năng cao. Framework này tự động tạo tài liệu API (OpenAPI/Swagger) và hỗ trợ type hints để phát hiện lỗi sớm.

Lý do lựa chọn:

- Hiệu năng cao, tương đương Node.js và Go
- Tự động validation dữ liệu với Pydantic

- Tài liệu API tự động, dễ dàng test và tích hợp
- Hỗ trợ async/await cho xử lý bất đồng bộ
- Type hints giúp code dễ đọc và bảo trì

2.5.2. Celery

Celery là distributed task queue cho Python, cho phép thực thi các tác vụ dài hạn trong background mà không chặn request chính.

Lý do lựa chọn:

- Xử lý bất đồng bộ các tác vụ benchmark có thể mất hàng phút
- Hỗ trợ distributed execution với nhiều worker
- Có cơ chế retry và error handling
- Tích hợp tốt với Redis/RabbitMQ
- Hỗ trợ monitoring và tracking progress

2.5.3. React và TypeScript

React là thư viện JavaScript phổ biến để xây dựng giao diện người dùng, TypeScript là superset của JavaScript với type checking tĩnh.

Lý do lựa chọn:

- Component-based architecture, dễ tái sử dụng code
- Virtual DOM cho hiệu năng cao
- Ecosystem phong phú (TanStack Query, Recharts, Tailwind CSS)
- TypeScript giúp phát hiện lỗi sớm và cải thiện developer experience
- Cộng đồng lớn và tài liệu đầy đủ

2.5.4. OpenAI API

OpenAI API cung cấp quyền truy cập vào các mô hình ngôn ngữ lớn như GPT-4, GPT-3.5-turbo để xử lý ngôn ngữ tự nhiên và phân tích dữ liệu.

Lý do lựa chọn:

- Khả năng phân tích dữ liệu benchmark phức tạp và đưa ra khuyến nghị thông minh
- Tạo giải thích dễ hiểu cho người dùng
- Hỗ trợ JSON mode để trả về dữ liệu có cấu trúc
- API dễ sử dụng và có tài liệu tốt
- Có thể fine-tune hoặc sử dụng prompt engineering để tối ưu hóa

2.5.5. AWS SDK (boto3)

boto3 là AWS SDK chính thức cho Python, cung cấp interface để tương tác với các dịch vụ AWS.

Lý do lựa chọn:

- SDK chính thức, được AWS hỗ trợ và cập nhật thường xuyên

- Hỗ trợ đầy đủ các dịch vụ AWS
- Hỗ trợ async operations
- Tích hợp tốt với IAM roles và credentials management
- Tài liệu đầy đủ và ví dụ phong phú

2.5.6. PostgreSQL

PostgreSQL là hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở, mạnh mẽ và đáng tin cậy.

Lý do lựa chọn:

- Hỗ trợ JSON/JSONB cho lưu trữ dữ liệu linh hoạt (raw latencies, anomalies)
- ACID compliance đảm bảo tính nhất quán dữ liệu
- Hỗ trợ full-text search và các tính năng nâng cao
- Performance tốt với proper indexing
- Tích hợp tốt với SQLAlchemy ORM

CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

3.1. Phân tích yêu cầu chức năng

3.1.1. Nhóm chức năng: Xác thực và Quản lý Người dùng

UC-01: Đăng ký tài khoản

- **Mô tả:** Người dùng mới đăng ký tài khoản với email và mật khẩu
- **Input:** Email, password, full_name (optional)
- **Output:** User ID, email
- **Xử lý:** Validate email format, hash password (bcrypt), tạo user record

UC-02: Đăng nhập

- **Mô tả:** Người dùng đăng nhập và nhận JWT token
- **Input:** Email, password
- **Output:** Access token (JWT)
- **Xử lý:** Verify password, generate JWT token

UC-03: Xem thông tin cá nhân

- **Mô tả:** Người dùng xem thông tin profile của mình
- **Input:** JWT token
- **Output:** User info (email, full_name, created_at)

UC-04: Refresh Token

- **Mô tả:** Làm mới JWT token khi hết hạn
- **Input:** Refresh token
- **Output:** New access token

3.1.2. Nhóm chức năng: Quản lý Dự án

UC-05: Tạo dự án mới

- **Mô tả:** Tạo project mới để quản lý các benchmark
- **Input:** Name, description, mode (optimization)
- **Output:** Project ID, project info
- **Xử lý:** Tạo project record, set owner_id = current_user.id

UC-06: Xem danh sách dự án

- **Mô tả:** Liệt kê tất cả projects của người dùng
- **Input:** JWT token
- **Output:** List of projects với pagination

UC-07: Xem chi tiết dự án

- **Mô tả:** Xem thông tin chi tiết của một project
- **Input:** Project ID
- **Output:** Project details, list of configs, list of benchmarks

UC-08: Cập nhật dự án

- **Mô tả:** Cập nhật thông tin project (name, description)
- **Input:** Project ID, updated fields
- **Output:** Updated project info

UC-09: Xóa dự án

- **Mô tả:** Xóa project và tất cả dữ liệu liên quan (cascade delete)
- **Input:** Project ID
- **Output:** Success message
- **Xử lý:** Cascade delete configs, benchmarks, results

UC-10: Cấu hình AWS Credentials

- **Mô tả:** Cấu hình IAM role ARN và External ID để truy cập AWS account
- **Input:** AWS Role ARN, External ID
- **Output:** Success message
- **Xử lý:** Validate ARN format, lưu vào project

UC-11: Chọn chế độ hoạt động

- **Mô tả:** Chọn mode (optimization) cho project
- **Input:** Mode selection
- **Output:** Updated project mode

3.1.3. Nhóm chức năng: Quản lý Cấu hình AWS

UC-12: Tạo cấu hình EC2

- **Mô tả:** Tạo cấu hình EC2 với các tham số (instance type, region, spot, etc.)
- **Input:** Name, instance_type, region, spot, VPC/subnet/SG, real_app settings
- **Output:** Config ID, config info
- **Xử lý:** Validate instance type, region, tạo AWSConfig record

UC-13: Xem danh sách cấu hình

- **Mô tả:** Liệt kê tất cả configs của project
- **Input:** Project ID
- **Output:** List of configs

UC-14: Xem chi tiết cấu hình

- **Mô tả:** Xem thông tin chi tiết của một config
- **Input:** Config ID
- **Output:** Config details

UC-15: Xóa cấu hình

- **Mô tả:** Xóa config khỏi project
- **Input:** Config ID
- **Output:** Success message

UC-16: Đặt cấu hình baseline

- **Mô tả:** Đánh dấu config là baseline (cho optimization mode)
- **Input:** Config ID
- **Output:** Updated config với is_baseline = true

3.1.4. Nhóm chức năng: Quản lý Benchmark

UC-17: Khởi tạo benchmark

- **Mô tả:** Bắt đầu một benchmark run cho các configs
- **Input:** Project ID, config_ids (optional), settings (requests_per_config, use_k6, etc.), budget
- **Output:** Benchmark ID, status = "pending"
- **Xử lý:** Tạo BenchmarkRun record, enqueue Celery task

UC-18: Xem danh sách benchmark

- **Mô tả:** Liệt kê tất cả benchmarks của project
- **Input:** Project ID, limit, offset
- **Output:** List of benchmarks với pagination

UC-19: Xem kết quả benchmark

- **Mô tả:** Xem chi tiết kết quả benchmark với metrics và charts
- **Input:** Benchmark ID
- **Output:** Benchmark details, results với metrics, recommendations (nếu có)

UC-20: Hủy benchmark

- **Mô tả:** Hủy benchmark đang chạy
- **Input:** Benchmark ID
- **Output:** Status = "cancelled"
- **Xử lý:** Update status, cleanup resources

UC-21: Xóa benchmark

- **Mô tả:** Xóa benchmark và results
- **Input:** Benchmark ID
- **Output:** Success message

UC-22: Xem tiến trình

- **Mô tả:** Theo dõi tiến trình benchmark real-time
- **Input:** Benchmark ID
- **Output:** Progress info (configs_completed, configs_total, current_config, estimated_cost)

3.1.5. Nhóm chức năng: Dịch vụ AI

UC-23: Nhận khuyến nghị

- **Mô tả:** Nhận AI recommendations sau khi benchmark baseline

- **Input:** Project ID
- **Output:** List of recommendations với instance types, reasoning, confidence scores
- **Xử lý:** Phân tích baseline result, gọi OpenAI API, generate recommendations

UC-24: So sánh cấu hình

- **Mô tả:** So sánh metrics giữa các configs
- **Input:** Config IDs
- **Output:** Comparison table với metrics side-by-side

UC-25: Phát hiện anomalies

- **Mô tả:** Tự động phát hiện bất thường trong kết quả benchmark
- **Input:** Benchmark result
- **Output:** List of anomalies (outliers, high variance)
- **Xử lý:** Sử dụng IQR method, variance analysis

UC-26: Phân tích metrics

- **Mô tả:** Phân tích và tính toán các metrics từ raw data
- **Input:** Raw latencies, CloudWatch metrics
- **Output:** Processed metrics (P50, P95, P99, mean, std, CPU utilization, etc.)

3.1.6. Nhóm chức năng: Thao tác AWS (Hệ thống)

UC-27: Triển khai lên AWS

- **Mô tả:** Tự động deploy EC2 instances cho benchmark
- **Input:** AWS Config, IAM role ARN
- **Output:** Instance ID, endpoint URL
- **Xử lý:** Assume IAM role, launch EC2, wait for ready, return endpoint

UC-28: Thu thập metrics

- **Mô tả:** Thu thập metrics từ CloudWatch
- **Input:** Instance ID, time range
- **Output:** CPU, disk I/O, network I/O metrics

UC-29: Giám sát tài nguyên

- **Mô tả:** Monitor EC2 instances trong quá trình benchmark
- **Input:** Instance ID
- **Output:** Status, health check results

UC-30: Dọn dẹp tài nguyên

- **Mô tả:** Tự động terminate EC2 instances sau benchmark
- **Input:** Instance ID
- **Output:** Success message

- **Xử lý:** Terminate instance, verify cleanup

3.1.7. Nhóm chức năng: Xử lý Nền

UC-31: Thực thi benchmark

- **Mô tả:** Celery worker thực thi benchmark trong background
- **Input:** Benchmark ID
- **Xử lý:** Load configs, deploy instances, execute requests, collect metrics, cleanup

UC-32: Tạo khuyến nghị tự động

- **Mô tả:** Tự động generate recommendations sau khi benchmark hoàn thành
- **Input:** Benchmark ID
- **Xử lý:** Phân tích results, gọi AI service, lưu recommendations

UC-33: Lưu trữ kết quả

- **Mô tả:** Lưu benchmark results vào database
- **Input:** Result data
- **Xử lý:** Create BenchmarkResult records, cache trong Redis

3.2. Phân tích yêu cầu phi chức năng

3.2.1. Hiệu năng

NFR-1: API Response Time

- **Yêu cầu:** API response time < 200ms cho các endpoints thông thường
- **Giải pháp:** Database indexing, connection pooling, async operations

NFR-2: Benchmark Execution Time

- **Yêu cầu:** Benchmark execution < 10 phút cho 10 configs
- **Giải pháp:** Parallel execution, optimize deployment time

NFR-3: Concurrent Requests

- **Yêu cầu:** Hỗ trợ tối đa 5 benchmark runs đồng thời
- **Giải pháp:** Celery workers với concurrency control

3.2.2. Khả năng mở rộng

NFR-4: User Capacity

- **Yêu cầu:** Hỗ trợ tối đa 1000 users
- **Giải pháp:** Horizontal scaling với load balancer

NFR-5: Data Capacity

- **Yêu cầu:** Hỗ trợ 100 projects/user, 50 configs/project
- **Giải pháp:** Database partitioning, pagination

3.2.3. Bảo mật

NFR-6: Authentication

- **Yêu cầu:** JWT authentication với secret key mạnh
- **Giải pháp:** HS256 algorithm, secure token storage

NFR-7: Password Security

- **Yêu cầu:** Password hashing với bcrypt
- **Giải pháp:** bcrypt với cost factor 12

NFR-8: AWS Credentials

- **Yêu cầu:** Không lưu trữ AWS credentials trực tiếp
- **Giải pháp:** IAM role assumption với External ID

NFR-9: Input Validation

- **Yêu cầu:** Validate và sanitize tất cả inputs
- **Giải pháp:** Pydantic schemas, SQLAlchemy ORM

3.2.4. Độ tin cậy

NFR-10: Uptime

- **Yêu cầu:** Uptime > 99.5%
- **Giải pháp:** Health checks, auto-restart, monitoring

NFR-11: Error Handling

- **Yêu cầu:** Graceful error handling, không crash hệ thống
- **Giải pháp:** Try-catch blocks, error logging, retry mechanisms

NFR-12: Data Backup

- **Yêu cầu:** Database backup tự động hàng ngày
- **Giải pháp:** PostgreSQL automated backups

3.2.5. Chi phí

NFR-13: Benchmark Cost

- **Yêu cầu:** Benchmark cost < $5 cho 10 configs
- **Giải pháp:** Sử dụng t3.micro/t3.small, auto cleanup, caching

NFR-14: Resource Cleanup

- **Yêu cầu:** Tự động cleanup để tránh chi phí phát sinh
- **Giải pháp:** Automatic termination, cleanup verification

3.3. Sơ đồ phân rã chức năng và Đặc tả biểu đồ Use-Case

Biểu đồ Use-Case tổng quan

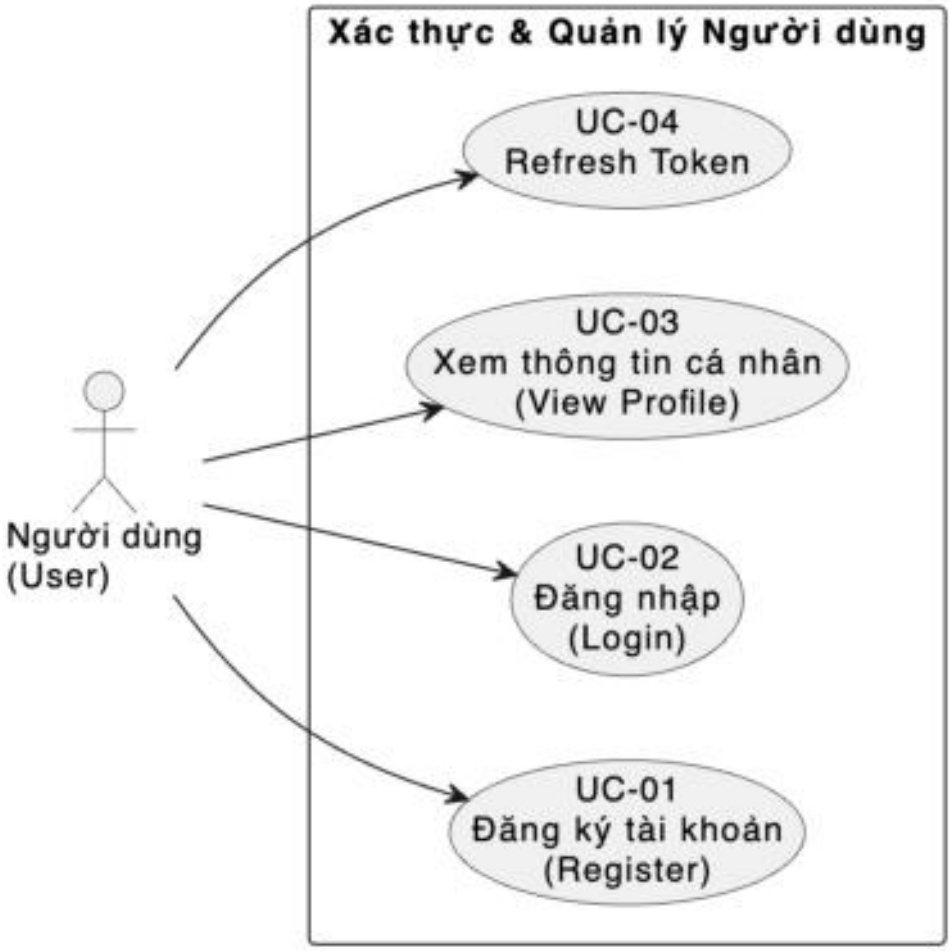
Biểu đồ Use-Case tổng quan mô tả các actors và use cases chính:

Actors:

- **Người dùng (User):** Sử dụng hệ thống để quản lý projects, configs, và xem kết quả
- **Tài khoản AWS (AWS Account):** Cung cấp EC2 instances và CloudWatch metrics
- **Hệ thống AI (AI System):** Cung cấp AI recommendations (OpenAI API)
- **Celery Worker:** Xử lý các tác vụ background

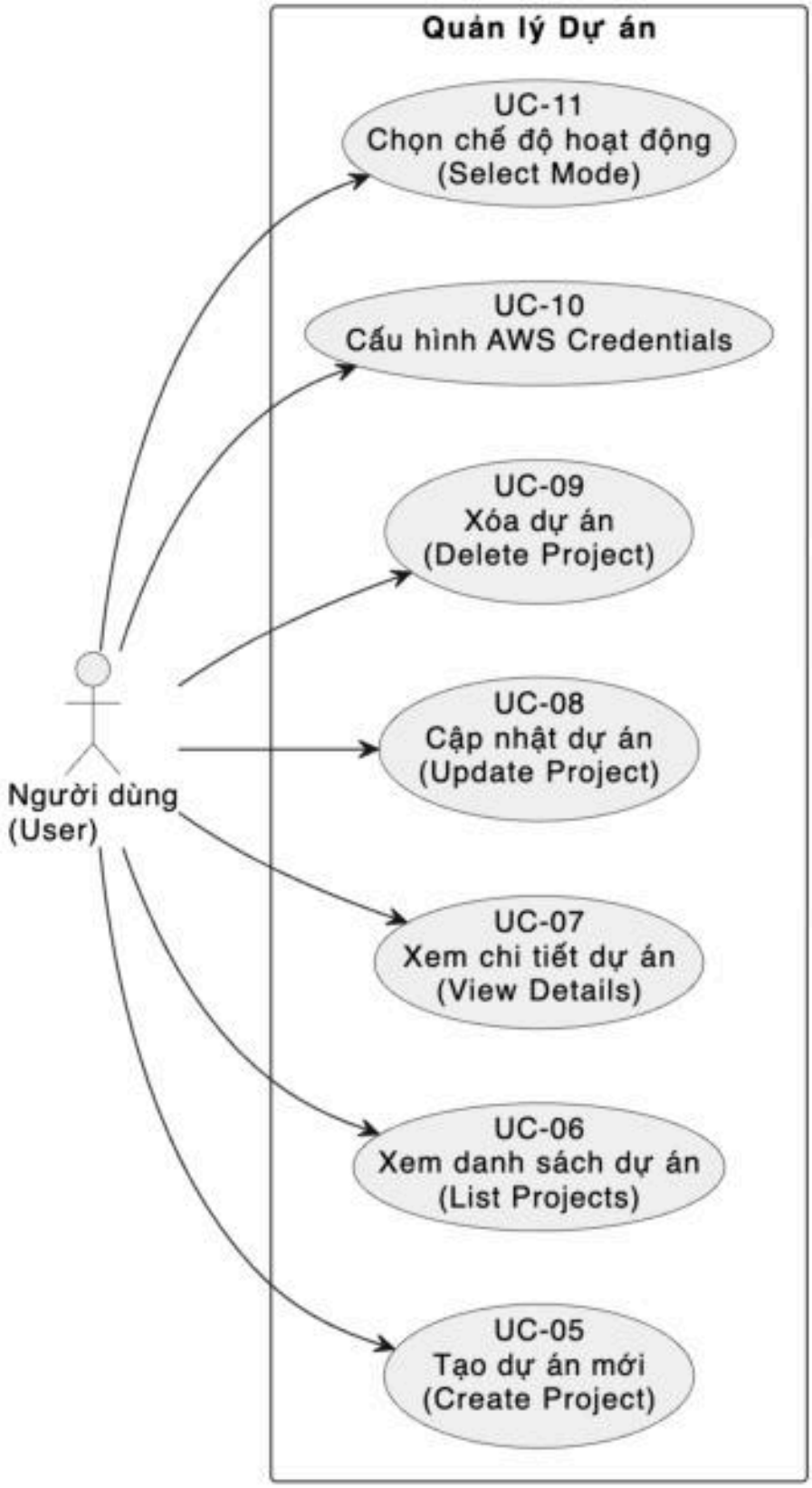
Các nhóm Use Cases:

Xác thực & Quản lý Người dùng: UC-01 đến UC-04



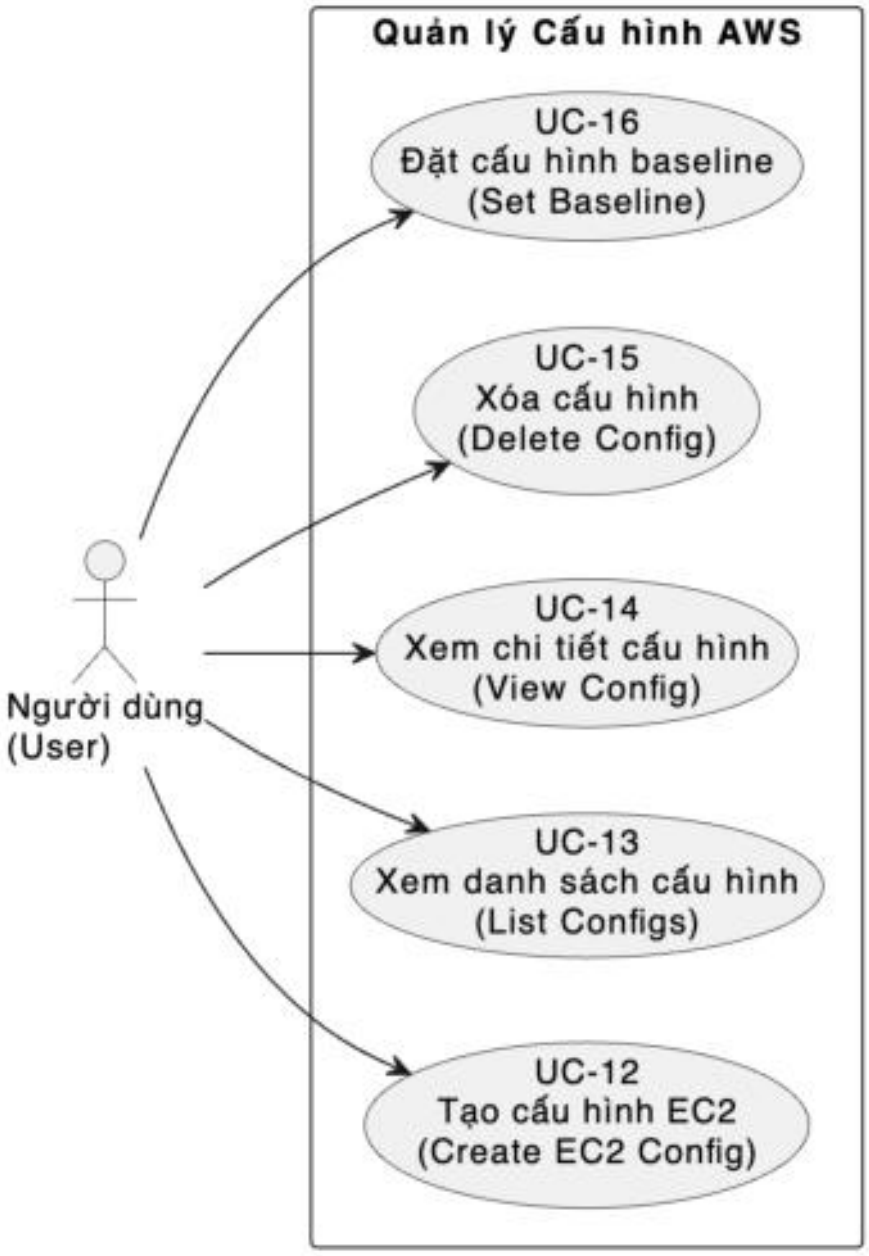
Hình 3.1. Xác thực & Quản lý Người dùng

Quản lý Dự án: UC-05 đến UC-11



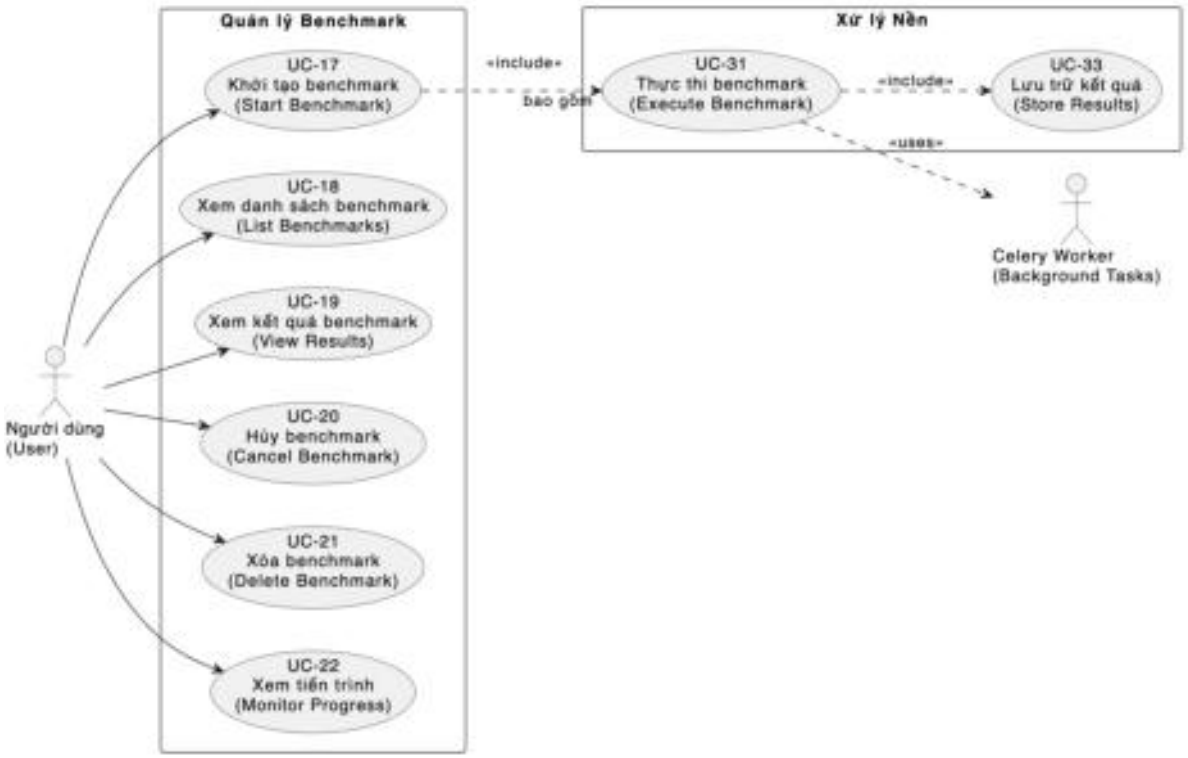
Hình 3.2. Quản lý Dự án

Quản lý Cấu hình AWS: UC-12 đến UC-16



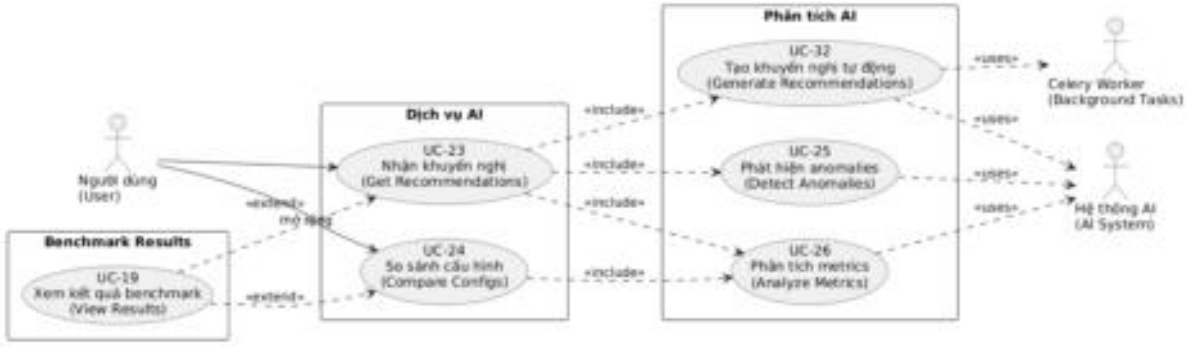
Hình 3.3. Quản lý cấu hình

Quản lý Benchmark: UC-17 đến UC-22



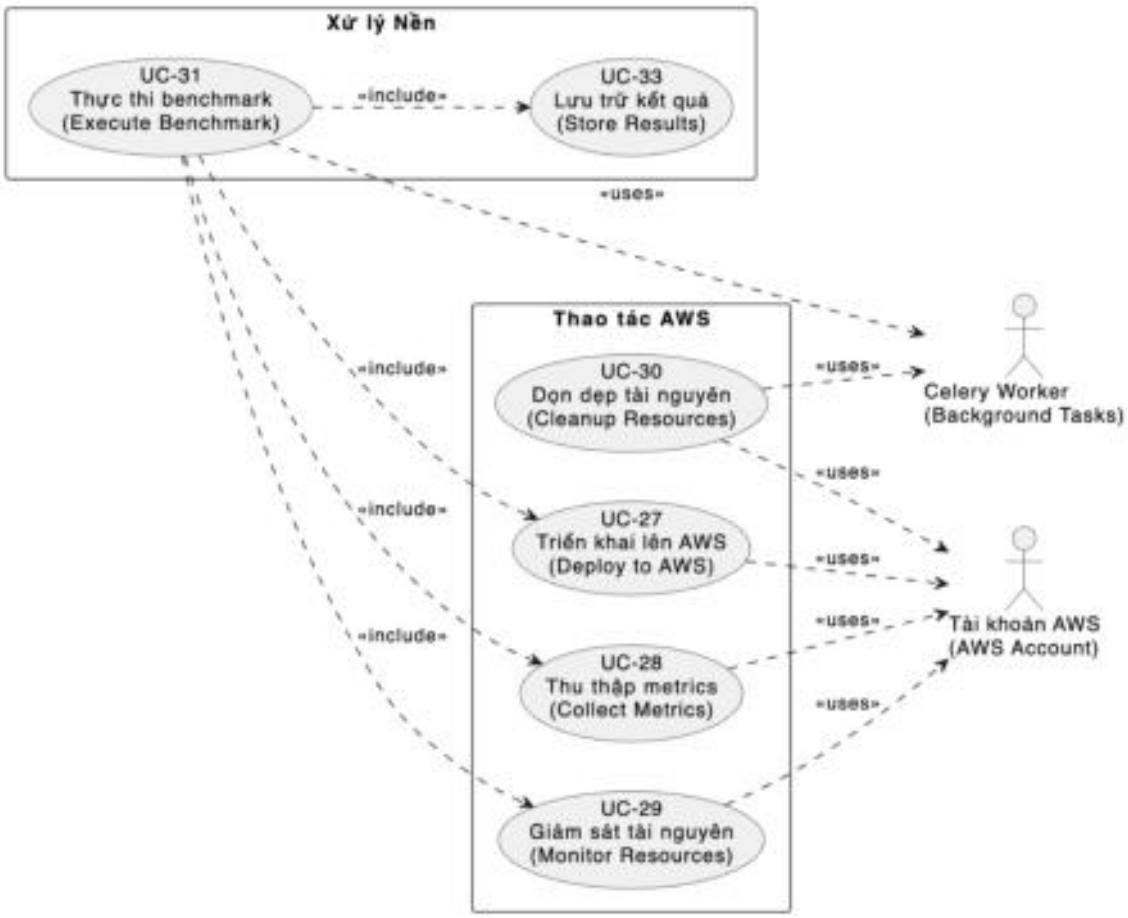
Hình 3.4. Quản lý Benchmark

Dịch vụ AI: UC-23 đến UC-26



Hình 3.5. Quản lý Dịch vụ AI

Thao tác AWS: UC-27 đến UC-30



Hình 3.6. Thao tác AWS

3.4. Đặc tả Use-Case

Bảng 3.1. UC-01: Đăng ký tài khoản

Mã ca sử dụng	UC-01		
Tên ca sử dụng	Đăng ký tài khoản		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng mới đăng ký tài khoản trong hệ thống bằng cách cung cấp email và mật khẩu. Hệ thống sẽ tạo tài khoản mới và lưu thông tin vào cơ sở dữ liệu.		
Điều kiện trước (Pre-Condition)	- Người dùng chưa có tài khoản trong hệ thống - Email chưa được sử dụng bởi tài khoản khác		
Điều kiện sau (Post-Condition)	- Tài khoản mới được tạo thành công - Người dùng có thể đăng nhập với email và mật khẩu đã đăng ký		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng truy cập trang đăng ký	Hệ thống hiển thị form đăng ký với các trường: email, password, full_name (tùy chọn)
	2	Người dùng nhập email, password và full_name (tùy chọn)	Hệ thống validate input real-time (nếu có)
	3	Người dùng click nút "Đăng ký"	Hệ thống validate email format và password strength
	4		Hệ thống kiểm tra email đã tồn tại trong database chưa
	5		Nếu email chưa tồn tại: Hệ thống hash password bằng bcrypt
	6		Hệ thống tạo user record trong database với is_active = true
	7		Hệ thống trả về thông báo thành công và user_id
	8		Frontend hiển thị thông báo "Đăng ký thành công"
	9		Người dùng được chuyển đến trang đăng nhập

Bảng 3.2. UC-02: Đăng nhập

Mã ca sử dụng	UC-02		
Tên ca sử dụng	Đăng nhập		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng đăng nhập vào hệ thống bằng email và mật khẩu. Hệ thống xác thực thông tin và cấp JWT token để truy cập các chức năng.		
Điều kiện trước (Pre-Condition)	- Người dùng đã có tài khoản trong hệ thống - Tài khoản chưa bị khóa (is_active = true)		
Điều kiện sau (Post-Condition)	- Người dùng đã đăng nhập thành công - JWT token được lưu trong localStorage - Người dùng có thể truy cập các chức năng của hệ thống		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng truy cập trang đăng nhập	Hệ thống hiển thị form đăng nhập với các trường: email, password
	2	Người dùng nhập email và password	Hệ thống validate input format
	3	Người dùng click nút "Đăng nhập"	Hệ thống tìm user theo email trong database
	4		Hệ thống verify password với hashed password (bcrypt)
	5		Hệ thống kiểm tra is_active = true
	6		Hệ thống generate JWT access token
	7		Hệ thống trả về access_token và user info
	8		Frontend lưu token vào localStorage
	9		Frontend lưu user info vào context/state
	10		Người dùng được chuyển đến trang Projects

Bảng 3.3. UC-03: Xem thông tin cá nhân

Mã ca sử dụng	UC-03		
Tên ca sử dụng	Xem thông tin cá nhân		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng xem thông tin profile của mình bao gồm email, tên đầy đủ và ngày tạo tài khoản.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - JWT token hợp lệ		
Điều kiện sau (Post-Condition)	- Người dùng đã xem được thông tin cá nhân		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng click vào menu "Profile" hoặc "Thông tin cá nhân"	Hệ thống extract user_id từ JWT token
	2		Hệ thống load thông tin user từ database
	3		Hệ thống trả về user info (email, full_name, created_at, is_active)
	4		Frontend hiển thị thông tin người dùng trong profile page

Bảng 3.4. UC-04: Refresh Token

Mã ca sử dụng	**UC-04**
Tên ca sử dụng	Refresh Token
Tác nhân	Người dùng (User)
Mô tả	Làm mới JWT token khi token hiện tại sắp hết hạn hoặc đã hết hạn để người dùng không cần đăng nhập lại.
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Có refresh token hợp lệ

Điều kiện sau (Post-Condition)	- JWT token mới được cấp - Token mới được lưu trong localStorage		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Frontend phát hiện access token sắp hết hạn hoặc đã hết hạn (tự động)	Frontend gửi refresh token đến API endpoint /auth/refresh
	2		Hệ thống verify refresh token
	3		Hệ thống extract user_id từ refresh token
	4		Hệ thống generate access token mới
	5		Hệ thống trả về access token mới
	6		Frontend cập nhật token trong localStorage
	7		Frontend tiếp tục request ban đầu với token mới

Bảng 3.5. UC-05: Tạo dự án mới

Mã ca sử dụng	**UC-05**
Tên ca sử dụng	Tạo dự án mới
Tác nhân	Người dùng (User)
Mô tả	Người dùng tạo một dự án mới để quản lý các cấu hình AWS và benchmark. Mỗi dự án đại diện cho một workload cần được đánh giá và tối ưu hóa.
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - JWT token hợp lệ
Điều kiện sau (Post-Condition)	- Dự án mới được tạo với status = "created" - Người dùng có thể thêm cấu hình AWS vào dự án - Dự án xuất hiện trong danh sách projects của người dùng

Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng truy cập trang Projects	Hệ thống hiển thị danh sách projects (nếu có) và nút "Tạo dự án mới"
	2	Người dùng click nút "Tạo dự án mới"	Hệ thống hiển thị form tạo dự án với các trường: name (bắt buộc), description (tùy chọn), mode (mặc định: optimization)
	3	Người dùng nhập tên dự án (bắt buộc), mô tả (tùy chọn)	Hệ thống validate input real-time
	4	Người dùng chọn chế độ hoạt động (mặc định: optimization)	Hệ thống hiển thị mô tả về chế độ đã chọn
	5	Người dùng click nút "Tạo"	Hệ thống validate input (tên không được trống, tên không quá 255 ký tự)
	6		Hệ thống extract user_id từ JWT token
	7		Hệ thống tạo Project record với owner_id = current_user.id, status = "created", mode = "optimization"
	8		Hệ thống lưu vào database
	9		Hệ thống trả về project_id và thông tin dự án
	10		Frontend hiển thị thông báo "Tạo dự án thành công"
	11		Người dùng được chuyển đến trang chi tiết dự án

Bảng 3.5. UC-06: Xem danh sách dự án

Mã ca sử dụng	**UC-06**		
Tên ca sử dụng	Xem danh sách dự án		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng xem danh sách tất cả các dự án mà mình đã tạo, có thể sắp xếp và lọc theo các tiêu chí.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - JWT token hợp lệ		
Điều kiện sau (Post-Condition)	- Người dùng đã xem được danh sách projects của mình		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng truy cập trang Projects	Hệ thống extract user_id từ JWT token
	2		Hệ thống load danh sách projects từ database với owner_id = current_user.id
	3		Hệ thống sắp xếp theo created_at DESC (mới nhất trước)
	4		Hệ thống đếm số configs và benchmarks cho mỗi project
	5		Hệ thống trả về danh sách projects với pagination
	6		Frontend hiển thị danh sách projects dưới dạng cards hoặc table
	7		Mỗi project card hiển thị: tên, mô tả, số configs, số benchmarks, status, created_at

Bảng 3.7. UC-07: Xem chi tiết dự án

Mã ca sử dụng	**UC-07**		
Tên ca sử dụng	Xem chi tiết dự án		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng xem thông tin chi tiết của một dự án bao gồm thông tin dự án, danh sách cấu hình AWS, danh sách benchmarks và recommendations.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Project tồn tại và thuộc về người dùng		
Điều kiện sau (Post-Condition)	- Người dùng đã xem được chi tiết dự án		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng click vào một project từ danh sách	Hệ thống extract user_id từ JWT token
	2		Hệ thống load project details từ database với project_id và owner_id
	3		Hệ thống load danh sách aws_configs của project
	4		Hệ thống load danh sách benchmark_runs của project (latest 10)
	5		Hệ thống load recommendations của project (nếu có)
	6		Hệ thống trả về project details với các relationships
	7		Frontend hiển thị: - Thông tin dự án (tên, mô tả, mode, status) - AWS credentials status (đã cấu hình/chưa cấu hình) - Danh sách configs với badges (baseline, instance type, region) - Danh sách benchmarks với status và created_at - Recommendations section (nếu có)

Bảng 3.8. UC-08: Cập nhật dự án

Mã ca sử dụng	**UC-08**		
Tên ca sử dụng	Cập nhật dự án		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng cập nhật thông tin của dự án như tên, mô tả hoặc chế độ hoạt động.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Project tồn tại và thuộc về người dùng		
Điều kiện sau (Post-Condition)	- Thông tin dự án đã được cập nhật - updated_at được cập nhật		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng ở trang chi tiết dự án	Hệ thống hiển thị thông tin dự án hiện tại
	2	Người dùng click nút "Chỉnh sửa"	Hệ thống hiển thị form edit với thông tin hiện tại (name, description, mode)
	3	Người dùng chỉnh sửa tên, mô tả hoặc mode	Hệ thống validate input real-time
	4	Người dùng click nút "Lưu"	Hệ thống validate input (tên không được trống)
	5		Hệ thống verify project thuộc về current_user
	6		Hệ thống update project record trong database
	7		Hệ thống update updated_at = NOW()
	8		Hệ thống trả về thông báo thành công
	9		Frontend cập nhật hiển thị với thông tin mới

Bảng 3.9. UC-09: Xóa dự án

Mã ca sử dụng	**UC-09**		
Tên ca sử dụng	Xóa dự án		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng xóa một dự án. Khi xóa dự án, tất cả dữ liệu liên quan (configs, benchmarks, results, recommendations) sẽ bị xóa theo (cascade delete).		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Project tồn tại và thuộc về người dùng - Không có benchmark đang chạy		
Điều kiện sau (Post-Condition)	- Dự án và tất cả dữ liệu liên quan đã bị xóa - Dự án không còn xuất hiện trong danh sách		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng ở trang chi tiết dự án	Hệ thống hiển thị nút "Xóa dự án"
	2	Người dùng click nút "Xóa dự án"	Hệ thống hiển thị dialog xác nhận "Bạn có chắc chắn muốn xóa dự án này? Tất cả dữ liệu liên quan (configs, benchmarks, results) sẽ bị xóa vĩnh viễn."
	3	Người dùng click "Xác nhận"	Hệ thống kiểm tra không có benchmark đang chạy (status = "pending" hoặc "running")
	4		Hệ thống verify project thuộc về current_user
	5		Hệ thống xóa project record (cascade delete configs, benchmarks, results, recommendations)
	6		Hệ thống trả về thông báo thành công
	7		Frontend hiển thị thông báo "Đã xóa dự án thành công"
	8		Người dùng được chuyển đến trang danh sách projects

Bảng 3.10. UC-10: Cấu hình AWS Credentials

Mã ca sử dụng	**UC-10**		
Tên ca sử dụng	Cấu hình AWS Credentials		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng cấu hình IAM role ARN và External ID để hệ thống có thể truy cập tài khoản AWS của họ. Hệ thống sử dụng cross-account access thông qua IAM role assumption, không lưu trữ AWS credentials trực tiếp.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Đã tạo ít nhất một dự án - Người dùng đã tạo IAM role trong AWS account của mình với trust policy cho phép hệ thống assume role		
Điều kiện sau (Post-Condition)	- AWS credentials đã được cấu hình trong project - Hệ thống có thể assume role và truy cập AWS account của người dùng		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng ở trang chi tiết dự án	Hệ thống hiển thị tab "AWS Settings" hoặc section "Cấu hình AWS"
	2	Người dùng click tab "AWS Settings"	Hệ thống hiển thị form cấu hình AWS credentials với các trường: IAM Role ARN, External ID (tùy chọn)
	3	Người dùng nhập IAM Role ARN (ví dụ: arn:aws:iam::123456789012:role/BenchmarkRole)	Hệ thống validate ARN format real-time
	4	Người dùng nhập External ID (tùy chọn, để tăng cường bảo mật)	Hệ thống hiển thị hướng dẫn về External ID
	5	Người dùng click nút "Lưu"	Hệ thống validate ARN format (phải bắt đầu với "arn:aws:iam::")
	6		Hệ thống test assume role (optional, có thể skip trong MVP)
	7		Hệ thống lưu aws_role_arn và

			aws_external_id vào project
	8		Hệ thống trả về thông báo thành công
	9		Frontend hiển thị "AWS credentials đã được cấu hình" với badge xanh

Bảng 3.11. UC-11: Chọn chế độ hoạt động

Mã ca sử dụng	**UC-11**		
Tên ca sử dụng	Chọn chế độ hoạt động		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng chọn chế độ hoạt động cho dự án. Hiện tại hệ thống hỗ trợ chế độ "optimization" - tối ưu hóa cấu hình hiện có.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Đã tạo dự án		
Điều kiện sau (Post-Condition)	- Chế độ hoạt động đã được cập nhật - Dự án hoạt động theo chế độ đã chọn		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng ở trang chi tiết dự án	Hệ thống hiển thị phần "Chế độ hoạt động" với giá trị hiện tại
	2	Người dùng click vào phần "Chế độ hoạt động"	Hệ thống hiển thị dropdown hoặc radio buttons với các options (hiện tại chỉ có "Optimization")
	3	Người dùng chọn "Optimization"	Hệ thống hiển thị mô tả về chế độ Optimization
	4	Người dùng click "Lưu" hoặc chọn tự động lưu	Hệ thống update project.mode = "optimization"
	5		Hệ thống lưu vào database
	6		Hệ thống trả về thông báo thành công
	7		Frontend cập nhật hiển thị với mode mới

Bảng 3.12. UC-12: Tạo cấu hình EC2

Mã ca sử dụng	**UC-12**		
Tên ca sử dụng	Tạo cấu hình EC2		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng tạo một cấu hình EC2 mới trong dự án. Cấu hình này định nghĩa các tham số của máy ảo EC2 sẽ được sử dụng trong benchmark, bao gồm instance type, region, và các tùy chọn khác.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Đã tạo ít nhất một dự án - Đang ở trang chi tiết dự án		
Điều kiện sau (Post-Condition)	- Cấu hình EC2 mới được tạo và lưu trong database - Cấu hình xuất hiện trong danh sách configs của dự án - Có thể sử dụng cấu hình này để benchmark		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng ở trang chi tiết dự án	Hệ thống hiển thị section "Cấu hình AWS" với nút "Thêm cấu hình"
	2	Người dùng click nút "Thêm cấu hình"	Hệ thống hiển thị form tạo cấu hình EC2 với các trường: - Name (bắt buộc) - Description (tùy chọn) - Instance Type (dropdown) - Region (dropdown) - Spot Instance (checkbox) - VPC ID, Subnet ID, Security Group ID (tùy chọn) - Real App Mode (checkbox) - Health Endpoint, Benchmark Endpoint (nếu real app mode)
	3	Người dùng nhập tên cấu hình (bắt buộc)	Hệ thống validate input real-time
	4	Người dùng chọn instance type từ dropdown (ví dụ: t3.medium, t4g.small)	Hệ thống hiển thị thông tin về instance type (vCPU, memory, pricing)
	5	Người dùng chọn	Hệ thống validate region hợp lệ

		region (ví dụ: us-east-1, ap-south-east-1)	
	6	Người dùng chọn các tùy chọn: spot instance, VPC/subnet/SG (tùy chọn)	Hệ thống hiển thị tooltip giải thích từng tùy chọn
	7	Người dùng chọn real app mode hoặc synthetic benchmark (mặc định: synthetic)	Nếu chọn real app mode: Hệ thống hiển thị thêm các trường health endpoint và benchmark endpoint
	8	Nếu real app mode: Người dùng nhập health endpoint (mặc định: /health) và benchmark endpoint (ví dụ: /api/users)	Hệ thống validate endpoint format (phải bắt đầu với /)
	9	Người dùng có thể chọn existing instance ID nếu muốn benchmark máy ảo hiện có	Hệ thống hiển thị hướng dẫn về existing instance mode
	10	Người dùng click nút "Lưu"	Hệ thống validate input (instance type hợp lệ, region hợp lệ, nếu real app mode thì endpoint bắt buộc)
	11		Hệ thống tạo AWSConfig record với project_id, service = "ec2"
	12		Hệ thống lưu vào database
	13		Hệ thống trả về config_id và thông tin cấu hình
	14		Frontend hiển thị thông báo "Tạo cấu hình thành công"
	15		Cấu hình xuất hiện trong danh sách configs

Bảng 3.13. UC-13: Xem danh sách cấu hình

Mã ca sử dụng	**UC-13**		
Tên ca sử dụng	Xem danh sách cấu hình		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng xem danh sách tất cả các cấu hình AWS trong một dự án.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Đang ở trang chi tiết dự án		
Điều kiện sau (Post-Condition)	- Người dùng đã xem được danh sách configs		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng ở trang chi tiết dự án	Hệ thống load danh sách aws_configs từ database với project_id
	2		Hệ thống sắp xếp theo created_at DESC (mới nhất trước)
	3		Hệ thống trả về danh sách configs
	4		Frontend hiển thị danh sách configs dưới dạng cards hoặc table
	5		Mỗi config card hiển thị: - Tên cấu hình - Instance type và region - Badge "Baseline" nếu is_baseline = true - Nút "Xem chi tiết", "Xóa"

Bảng 3.14. UC-14: Xem chi tiết cấu hình

Mã ca sử dụng	**UC-14**		
Tên ca sử dụng	Xem chi tiết cấu hình		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng xem thông tin chi tiết của một cấu hình AWS bao gồm tất cả các tham số đã cấu hình.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Config tồn tại và thuộc về dự án của người dùng		
Điều kiện sau (Post-Condition)	- Người dùng đã xem được chi tiết cấu hình		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng click vào một config từ danh sách	Hệ thống load config details từ database với config_id
	2		Hệ thống verify config thuộc về project của current_user
	3		Hệ thống trả về config info với tất cả fields
	4		Frontend hiển thị chi tiết cấu hình trong modal hoặc detail page: - Tên, mô tả - Instance type, region - Spot instance (yes/no) - VPC ID, Subnet ID, Security Group ID - Real app mode settings (nếu có): health endpoint, benchmark endpoint - Is baseline (yes/no)

Bảng 3.15. UC-15: Xóa cấu hình

Mã ca sử dụng	**UC-15**		
Tên ca sử dụng	Xóa cấu hình		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng xóa một cấu hình AWS khỏi dự án. Cấu hình không thể xóa nếu đang được sử dụng trong benchmark đang chạy.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Config tồn tại và thuộc về dự án của người dung - Config không đang được sử dụng trong benchmark đang chạy		
Điều kiện sau (Post-Condition)	- Cấu hình đã bị xóa - Cấu hình không còn xuất hiện trong danh sách		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng ở trang danh sách configs hoặc trang chi tiết config	Hệ thống hiển thị nút "Xóa" trên mỗi config card
	2	Người dùng click nút "Xóa"	Hệ thống hiển thị dialog xác nhận "Bạn có chắc chắn muốn xóa cấu hình này?"
	3	Người dùng click "Xác nhận"	Hệ thống kiểm tra config không đang được sử dụng trong benchmark đang chạy (status = "pending" hoặc "running")
	4		Hệ thống verify config thuộc về project của current_user
	5		Hệ thống xóa config record từ database
	6		Hệ thống trả về thông báo thành công
	7		Frontend hiển thị thông báo "Đã xóa cấu hình thành công"
	8		Config biến mất khỏi danh sách

Bảng 3.16. UC-16: Đặt cấu hình baseline

Mã ca sử dụng	**UC-16**		
Tên ca sử dụng	Đặt cấu hình baseline		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng đánh dấu một cấu hình là baseline (cấu hình hiện tại đang sử dụng trong production). Baseline sẽ được sử dụng để so sánh với các recommendations từ AI.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Config tồn tại và thuộc về dự án - Dự án ở chế độ "optimization"		
Điều kiện sau (Post-Condition)	- Config được đánh dấu là baseline (is_baseline = true) - project.baseline_config_id được cập nhật - Các configs khác (nếu có) được bỏ đánh dấu baseline		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng ở trang danh sách configs hoặc trang chi tiết config	Hệ thống hiển thị toggle "Is Baseline" hoặc nút "Đặt làm Baseline" trên mỗi config
	2	Người dùng click toggle "Is Baseline" hoặc nút "Đặt làm Baseline"	Hệ thống kiểm tra dự án ở chế độ optimization
	3		Hệ thống update config.is_baseline = true
	4		Hệ thống update project.baseline_config_id = config.id
	5		Hệ thống update các configs khác trong project: is_baseline = false (chỉ có 1 baseline)
	6		Hệ thống lưu vào database
	7		Hệ thống trả về thông báo thành công
	8		Frontend hiển thị badge "Baseline" trên config và bỏ badge trên các configs khác

Bảng 3.17. UC-17: Khởi tạo benchmark

Mã ca sử dụng	**UC-17**		
Tên ca sử dụng	Khởi tạo benchmark		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng khởi tạo một benchmark run để đo lường hiệu năng của các cấu hình EC2. Hệ thống sẽ tự động triển khai các máy ảo EC2, thực thi benchmark, thu thập metrics và lưu kết quả.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Dự án đã có ít nhất 1 cấu hình AWS - AWS credentials đã được cấu hình (IAM role ARN) - Không có benchmark nào đang chạy cho dự án này		
Điều kiện sau (Post-Condition)	- BenchmarkRun được tạo với status = "pending" - Celery task được enqueue để thực thi benchmark - Người dùng có thể theo dõi tiến trình benchmark		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng ở trang chi tiết dự án	Hệ thống hiển thị nút "Chạy Benchmark"
	2	Người dùng click nút "Chạy Benchmark"	Hệ thống hiển thị dialog cấu hình benchmark với: - Danh sách configs để chọn (checkbox, mặc định chọn tất cả) - Settings: requests_per_config (default: 50), use_k6 (default: false), k6_virtual_users (nếu use_k6 = true), budget (tùy chọn)
	3	Người dùng chọn các cấu hình cần benchmark (hoặc chọn tất cả)	Hệ thống hiển thị số lượng configs đã chọn
	4	Người dùng cấu hình settings: số requests (mặc định: 50), use_k6 (mặc định: false), k6_virtual_users (nếu use_k6 =	Hệ thống validate input (requests > 0, k6_virtual_users > 0 nếu use_k6 = true)

		true), budget (tùy chọn)	
	5	Người dùng click nút "Bắt đầu"	Hệ thống validate: kiểm tra có configs được chọn không
	6		Hệ thống kiểm tra AWS credentials đã cấu hình chưa (aws_role_arn)
	7		Hệ thống kiểm tra không có benchmark đang chạy (status = "pending" hoặc "running")
	8		Hệ thống tạo BenchmarkRun record với status = "pending", configs_total = số configs đã chọn
	9		Hệ thống enqueue Celery task (execute_benchmark_task.delay(benchmark_id))
	10		Hệ thống trả về benchmark_id và status = "pending"
	11		Frontend hiển thị "Benchmark đang chờ..." với progress indicator
	12		Celery worker bắt đầu xử lý task trong background

Bảng 3.18. UC-18: Xem danh sách benchmark

Mã ca sử dụng	**UC-18**
Tên ca sử dụng	Xem danh sách benchmark
Tác nhân	Người dùng (User)
Mô tả	Người dùng xem danh sách tất cả các benchmark runs của một dự án, có thể sắp xếp và lọc theo status.
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Đang ở trang chi tiết dự án
Điều kiện sau (Post-Condition)	- Người dùng đã xem được danh sách benchmarks

Kịch bản		Hành động của tác nhân	Phản hồi của hệ thống
	1	Người dùng ở trang chi tiết dự án	Hệ thống load danh sách benchmark_runs từ database với project_id
	2		Hệ thống sắp xếp theo created_at DESC (mới nhất trước)
	3		Hệ thống đếm số results cho mỗi benchmark
	4		Hệ thống trả về danh sách benchmarks với pagination
	5		Frontend hiển thị danh sách benchmarks dưới dạng cards hoặc table
	6		Mỗi benchmark card hiển thị: - Status badge (pending, running, completed, failed) - Số configs (configs_completed/configs_total) - Total cost - Created at, Completed at - Nút "Xem chi tiết", "Hủy" (nếu đang chạy), "Xóa"

Bảng 3.19. UC-19: Xem kết quả benchmark

Mã ca sử dụng	**UC-19**
Tên ca sử dụng	Xem kết quả benchmark
Tác nhân	Người dùng (User)
Mô tả	Người dùng xem chi tiết kết quả của một benchmark run, bao gồm các metrics (latency, cost, CPU utilization) và biểu đồ so sánh. Nếu benchmark đã hoàn thành, hệ thống có thể hiển thị AI recommendations.
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Benchmark đã được khởi tạo (ít nhất status = "pending") - Người dùng có quyền truy cập dự án chứa benchmark
Điều kiện sau	- Người dùng đã xem được kết quả benchmark

(Post-Condition)	- Nếu có recommendations, người dùng có thể xem và approve/reject		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng click vào một benchmark từ danh sách	Hệ thống load BenchmarkRun và các BenchmarkResults từ database
	2		Hệ thống verify benchmark thuộc về project của current_user
	3		Nếu benchmark status = "running": Hệ thống trả về progress info (configs_completed, configs_total, current_config_id)
	4		Nếu benchmark status = "completed": Hệ thống trả về kết quả đầy đủ với tất cả results
	5		Frontend hiển thị - Nếu running: Progress bar, spinner, "Đang chạy..." - Nếu completed: Bảng kết quả với metrics cho mỗi config
	6		Frontend hiển thị bảng kết quả với các cột: - Config name - Latency: P50, P95, P99, mean, std - Cost: per request, per 1K requests - CPU utilization: avg, P95, max - Resourcing verdict: under/optimized/over-provisioned - Confidence score
	7		Frontend hiển thị biểu đồ so sánh latency (P95) và cost giữa các configs
	8		Nếu có recommendations: Frontend hiển thị section "AI Recommendations" với danh sách recommendations
	9		Frontend tự động poll API mỗi 5 giây nếu benchmark đang chạy để cập nhật progress

Bảng 3.20. UC-20: Hủy benchmark

Mã ca sử dụng	**UC-20**		
Tên ca sử dụng	Hủy benchmark		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng hủy một benchmark đang chạy. Hệ thống sẽ dừng quá trình benchmark và cleanup các tài nguyên đang sử dụng.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Benchmark đang chạy (status = "pending" hoặc "running") - Người dùng có quyền truy cập dự án chứa benchmark		
Điều kiện sau (Post-Condition)	- Benchmark status = "cancelled" - Các tài nguyên AWS đang sử dụng được cleanup - Celery task được hủy (nếu có thể)		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng ở trang chi tiết benchmark hoặc danh sách benchmarks	Hệ thống hiển thị nút "Hủy" nếu benchmark status = "pending" hoặc "running"
	2	Người dùng click nút "Hủy"	Hệ thống hiển thị dialog xác nhận "Bạn có chắc chắn muốn hủy benchmark này?"
	3	Người dùng click "Xác nhận"	Hệ thống verify benchmark thuộc về project của current_user
	4		Hệ thống verify benchmark status = "pending" hoặc "running"
	5		Hệ thống update benchmark status = "cancelled"
	6		Hệ thống update benchmark.completed_at = NOW()
	7		Hệ thống gửi signal để cancel Celery task (nếu có thể)
	8		Hệ thống trigger cleanup task để terminate các EC2 instances đang chạy
	9		Hệ thống trả về thông báo thành công

	10		Frontend hiển thị "Benchmark đã được hủy"
	11		Frontend cập nhật status badge thành "Cancelled"

Bảng 3.21. UC-21: Xóa benchmark

Mã ca sử dụng	**UC-21**		
Tên ca sử dụng	Xóa benchmark		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng xóa một benchmark run và tất cả kết quả liên quan. Benchmark đang chạy không thể xóa, phải hủy trước.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Benchmark không đang chạy (status != "running" và != "pending") - Người dùng có quyền truy cập dự án chứa benchmark		
Điều kiện sau (Post-Condition)	- Benchmark và tất cả results đã bị xóa - Benchmark không còn xuất hiện trong danh sách		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng ở trang chi tiết benchmark hoặc danh sách benchmarks	Hệ thống hiển thị nút "Xóa" nếu benchmark không đang chạy
	2	Người dùng click nút "Xóa"	Hệ thống hiển thị dialog xác nhận "Bạn có chắc chắn muốn xóa benchmark này? Tất cả kết quả sẽ bị xóa vĩnh viễn."
	3	Người dùng click "Xác nhận"	Hệ thống verify benchmark thuộc về project của current_user
	4		Hệ thống kiểm tra benchmark status != "running" và != "pending"

	5		Hệ thống xóa benchmark record (cascade delete results)
	6		Hệ thống trả về thông báo thành công
	7		Frontend hiển thị "Đã xóa benchmark thành công"
	8		Benchmark biến mất khỏi danh sách

Bảng 3.22. UC-22: Xem tiến trình

Mã ca sử dụng	**UC-22**		
Tên ca sử dụng	Xem tiến trình		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng theo dõi tiến trình benchmark real-time, xem số configs đã hoàn thành, config đang xử lý, và ước tính chi phí.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Benchmark đang chạy (status = "pending" hoặc "running") - Người dùng có quyền truy cập dự án chứa benchmark		
Điều kiện sau (Post-Condition)	- Người dùng đã xem được tiến trình benchmark		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng ở trang chi tiết benchmark	Hệ thống load benchmark progress từ database (configs_completed, configs_total, current_config_id, total_cost_usd)
	2		Hệ thống trả về progress info
	3		Frontend hiển thị progress bar với percentage = (configs_completed / configs_total) * 100
	4		Frontend hiển thị text "Đã hoàn thành X/Y configs"
	5		Frontend hiển thị "Đang xử lý: [Config name]" nếu có current_config_id

	6		Frontend hiển thị "Tổng chi phí ước tính: $X.XX"
	7		Frontend tự động poll API mỗi 5 giây để cập nhật progress
	8		Khi benchmark completed: Frontend dừng polling và hiển thị "Hoàn thành"

Bảng 3.23. UC-23: Nhận khuyến nghị

Mã ca sử dụng	**UC-23**		
Tên ca sử dụng	Nhận khuyến nghị		
Tác nhân	Người dùng (User)		
Mô tả	Hệ thống phân tích kết quả benchmark baseline và sử dụng AI (OpenAI GPT) để đưa ra khuyến nghị về các instance types tối ưu. Mỗi khuyến nghị bao gồm instance type, lý do đề xuất, ước tính tiết kiệm chi phí và cải thiện hiệu năng, cùng với confidence score.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Dự án có baseline config đã được benchmark - Benchmark baseline đã hoàn thành (status = "completed") - Có kết quả benchmark cho baseline config		
Điều kiện sau (Post-Condition)	- Recommendations được tạo và lưu trong database - Người dùng có thể xem recommendations với giải thích chi tiết - Người dùng có thể approve/reject recommendations		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng xem kết quả benchmark baseline	Hệ thống kiểm tra xem đã có recommendations chưa
	2		Nếu chưa có: Hệ thống hiển thị nút "Tạo Recommendations"
	3	Người dùng click nút "Tạo Recommendations" hoặc hệ thống tự động tạo sau khi benchmark hoàn thành	Hệ thống load baseline BenchmarkResult từ database
	4		Hệ thống gọi AI Service để

			generate recommendations
	5		AI Service phân tích baseline metrics: - Latency metrics (P50, P95, P99, mean, std) - CPU utilization (avg, P95, max) - Cost metrics - Error rate
	6		AI Service xác định đặc điểm workload (CPU-intensive, memory-intensive, balanced)
	7		AI Service build prompt chi tiết với baseline metrics và workload analysis
	8		AI Service gọi OpenAI API (GPT-4o-mini) với prompt và response_format = JSON
	9		OpenAI trả về JSON với 4 recommendations
	10		AI Service validate instance types (kiểm tra format và tính hợp lệ)
	11		AI Service filter và rank recommendations
	12		AI Service lưu recommendations vào database (Recommendation record)
	13		Hệ thống trả về recommendations cho người dùng
	14		Frontend hiển thị recommendations với: - Instance type - Reason (giải thích chi tiết) - Estimated cost savings (%)

			- Estimated performance improvement (%) - Confidence score (0-1) - Nút "Approve" và "Reject"

Bảng 3.24. UC-24: So sánh cấu hình

Mã ca sử dụng	**UC-24**		
Tên ca sử dụng	So sánh cấu hình		
Tác nhân	Người dùng (User)		
Mô tả	Người dùng so sánh metrics giữa các cấu hình khác nhau trong cùng một benchmark run hoặc giữa các benchmark runs khác nhau.		
Điều kiện trước (Pre-Condition)	- Người dùng đã đăng nhập - Có ít nhất 2 cấu hình đã được benchmark - Người dùng có quyền truy cập dự án		
Điều kiện sau (Post-Condition)	- Người dùng đã xem được so sánh metrics giữa các configs		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Người dùng ở trang kết quả benchmark	Hệ thống load tất cả BenchmarkResults từ benchmark
	2		Hệ thống tính toán metrics cho mỗi result (UC-26)
	3		Hệ thống trả về danh sách results với metrics
	4		Frontend hiển thị bảng so sánh với các cột: - Config name - Latency P50, P95, P99 - Cost per 1K requests - CPU utilization - Error rate - Confidence score
	5		Frontend hiển thị biểu đồ so sánh: - Bar chart: Latency P95 so sánh

			- Bar chart: Cost per 1K requests so sánh - Line chart: Latency distribution (P50, P95, P99)
	6		Frontend highlight config tốt nhất cho từng metric

Bảng 3.25. UC-25: Phát hiện anomalies

Mã ca sử dụng	**UC-25**		
Tên ca sử dụng	Phát hiện anomalies		
Tác nhân	Hệ thống (System)		
Mô tả	Hệ thống tự động phát hiện các giá trị bất thường trong kết quả benchmark bằng phương pháp thống kê IQR (Interquartile Range). Các anomalies được loại bỏ khỏi tính toán metrics để đảm bảo độ tin cậy của kết quả.		
Điều kiện trước (Pre-Condition)	- Có raw latencies từ benchmark execution - Số lượng latencies >= 5		
Điều kiện sau (Post-Condition)	- Anomalies đã được phát hiện và lưu trong result.anomalies - Cleaned latencies được sử dụng để tính toán metrics		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Hệ thống nhận raw latencies từ benchmark execution	Hệ thống kiểm tra số lượng latencies >= 5
	2		Hệ thống sắp xếp latencies theo thứ tự tăng dần
	3		Hệ thống tính Q1 (25th percentile) và Q3 (75th percentile)
	4		Hệ thống tính IQR = Q3 - Q1
	5		Hệ thống tính lower_bound = Q1 - 1.5 * IQR và upper_bound = Q3 + 1.5 * IQR

	6		Hệ thống duyệt qua từng latency và kiểm tra: - Nếu latency < lower_bound hoặc > upper_bound: Đánh dấu là outlier
	7		Hệ thống tạo danh sách anomalies với thông tin: type (latency_spike hoặc latency_drop), request_index, value_ms, excluded = true
	8		Hệ thống phát hiện high variance: Tính CV = std / mean, nếu CV > 0.3: Đánh dấu là high_variance anomaly
	9		Hệ thống lưu anomalies vào result.anomalies (JSONB)
	10		Hệ thống sử dụng cleaned latencies (loại bỏ outliers) để tính toán metrics

Bảng 3.26. UC-26: Phân tích metrics

Mã ca sử dụng	**UC-26**		
Tên ca sử dụng	Phân tích metrics		
Tác nhân	Hệ thống (System)		
Mô tả	Hệ thống phân tích và tính toán các metrics từ raw latencies và CloudWatch metrics, bao gồm latency percentiles, cost, CPU utilization, và resource utilization verdict.		
Điều kiện trước (Pre-Condition)	- Có raw latencies từ benchmark execution - Có thể có CloudWatch metrics (nếu EC2 instance có CloudWatch Agent)		
Điều kiện sau (Post-Condition)	- Các metrics đã được tính toán và lưu trong BenchmarkResult		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Hệ thống nhận raw latencies và CloudWatch metrics	Hệ thống loại bỏ outliers (nếu có) từ latencies
	2		Hệ thống tính toán latency percentiles:

			- P50 (median): percentile(latencies, 0.50) - P95: percentile(latencies, 0.95) - P99: percentile(latencies, 0.99) - Mean: sum(latencies) / len(latencies) - Std: standard deviation
	3		Hệ thống phát hiện cold start: - So sánh first_request với warm_avg (mean của requests 2-N) - Nếu first_request > warm_avg * 2.0: Phát hiện cold start
	4		Hệ thống tính error rate = failed_requests / total_requests
	5		Hệ thống tính cost: - cost_per_request = (hourly_cost / 3600) * avg_latency_seconds - cost_per_1k_requests = cost_per_request * 1000 - benchmark_cost = cost_per_request * total_requests
	6		Hệ thống phân tích CPU utilization từ CloudWatch: - Nếu CPU avg > 80%: verdict = "under-provisioned" - Nếu CPU avg < 20%: verdict = "over-provisioned" - Nếu 20% <= CPU avg <= 80%: verdict = "optimized"
	7		Hệ thống tính confidence score: - Base: 0.9 - Giảm nếu: cached result (theo tuổi), high variance, nhiều anomalies, high error rate
	8		Hệ thống lưu tất cả metrics vào BenchmarkResult record

Bảng 3.27. UC-27: Triển khai lên AWS

Mã ca sử dụng	**UC-27**		
Tên ca sử dụng	Triển khai lên AWS		
Tác nhân	Hệ thống (System)		
Mô tả	Hệ thống tự động triển khai EC2 instance cho benchmark. Quá trình này bao gồm assume IAM role, tìm hoặc tạo security group, launch EC2 instance với user data script, và đợi instance ready.		
Điều kiện trước (Pre-Condition)	- AWS credentials đã được cấu hình (IAM role ARN) - AWS config đã được tạo với đầy đủ thông tin (instance type, region)		
Điều kiện sau (Post-Condition)	- EC2 instance đã được launch và running - Endpoint URL đã sẵn sàng để nhận requests		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Hệ thống nhận yêu cầu deploy EC2 instance cho một config	Hệ thống assume IAM role sử dụng aws_role_arn và aws_external_id
	2		AWS trả về temporary credentials
	3		Hệ thống tìm hoặc tạo security group cho phép HTTP traffic (port 8080)
	4		Hệ thống tìm AMI phù hợp theo region và architecture (ARM64 hoặc x86_64)
	5		Hệ thống generate user data script: - Nếu real app mode: Chỉ cài CloudWatch Agent - Nếu synthetic mode: Cài Python, FastAPI, và benchmark server
	6		Hệ thống launch EC2 instance với: - Instance type từ config - AMI ID - User data script - Security group - VPC/Subnet từ config (hoặc default)
	7		AWS trả về instance ID
	8		Hệ thống đợi instance status = "running" (polling mỗi 10 giây)

	9		Hệ thống lấy public IP hoặc private IP của instance
	10		Hệ thống đợi HTTP server ready (polling /health endpoint mỗi 5 giây, timeout 5 phút)
	11		Hệ thống trả về endpoint URL (http://IP:port)

Bảng 3.28. UC-28: Thu thập metrics

Mã ca sử dụng	**UC-28**		
Tên ca sử dụng	Thu thập metrics		
Tác nhân	Hệ thống (System)		
Mô tả	Hệ thống thu thập metrics từ CloudWatch cho EC2 instance trong khoảng thời gian benchmark. Metrics bao gồm CPU utilization (từ CloudWatch Agent), disk I/O, network I/O, và status checks.		
Điều kiện trước (Pre-Condition)	- EC2 instance đã được deploy và đang chạy - Benchmark đã hoàn thành (có benchmark_start_time và benchmark_end_time) - CloudWatch Agent đã được cài đặt (nếu cần CPU metrics)		
Điều kiện sau (Post-Condition)	- Metrics đã được thu thập và lưu trong BenchmarkResult		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Hệ thống nhận yêu cầu thu thập metrics cho instance	Hệ thống đợi grace period (120 giây) để CloudWatch metrics được publish
	2		Hệ thống assume IAM role để truy cập CloudWatch
	3		Hệ thống query CloudWatch Metrics API: - CPUUtilization (từ CloudWatch Agent): Average, P95, Maximum

			- DiskReadBytes, DiskWriteBytes: Average, Sum - NetworkInBytes, NetworkOutBytes: Average, Sum - StatusCheckFailed, StatusCheck-Failed_Instance, StatusCheck-Failed_System: Sum
	4		Hệ thống aggregate metrics trong khoảng thời gian bench-mark_start_time đến bench-mark_end_time
	5		Hệ thống tính toán: - CPU avg, P95, max - Disk I/O avg và total - Network I/O avg và total - Status check failures
	6		Hệ thống phân tích resource utiliza-tion: - Nếu CPU avg > 80%: verdict = "un-der-provisioned" - Nếu CPU avg < 20%: verdict = "over-provisioned" - Nếu 20% <= CPU avg <= 80%: ver-dict = "optimized"
	7		Hệ thống lưu metrics vào Bench-markResult record

Bảng 3.29. UC-29: Giám sát tài nguyên

Mã ca sử dụng	**UC-29**
Tên ca sử dụng	Giám sát tài nguyên
Tác nhân	Hệ thống (System)
Mô tả	Hệ thống giám sát EC2 instances trong quá trình benchmark để đảm bảo chúng hoạt động bình thường và có thể nhận requests.
Điều kiện trước (Pre-Condi-tion)	- EC2 instance đã được deploy

Điều kiện sau (Post-Condition)	- Hệ thống đã xác nhận instance hoạt động bình thường		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Hệ thống bắt đầu giám sát instance sau khi deploy	Hệ thống poll instance status mỗi 10 giây
	2		Hệ thống kiểm tra instance state = "running"
	3		Hệ thống thực hiện health check bằng cách gọi /health endpoint
	4		Nếu health check thành công: Instance sẵn sàng
	5		Nếu health check thất bại: Retry 3 lần, nếu vẫn fail thì mark as failed

Bảng 3.30. UC-30: Dọn dẹp tài nguyên

Mã ca sử dụng	**UC-30**		
Tên ca sử dụng	Dọn dẹp tài nguyên		
Tác nhân	Hệ thống (System)		
Mô tả	Hệ thống tự động terminate EC2 instances sau khi benchmark hoàn thành để tránh chi phí phát sinh. Quá trình này được thực hiện trong background task.		
Điều kiện trước (Pre-Condition)	- Benchmark đã hoàn thành hoặc bị hủy - EC2 instances đã được deploy		
Điều kiện sau (Post-Condition)	- Tất cả EC2 instances đã được terminate - Metrics đã được thu thập trước khi terminate		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Hệ thống nhận yêu cầu cleanup sau khi benchmark hoàn thành hoặc bị hủy	Hệ thống đợi thu thập CloudWatch metrics (nếu chưa thu thập)
	2		Hệ thống assume IAM role

	3		Hệ thống tìm tất cả EC2 instances với tag "CloudEvaluator=true" và "BenchmarkID=benchmark_id"
	4		Hệ thống terminate từng instance
	5		Hệ thống verify instance state = "terminated"
	6		Hệ thống log cleanup completion

Bảng 3.31. UC-31: Thực thi benchmark

Mã ca sử dụng	**UC-31**		
Tên ca sử dụng	Thực thi benchmark		
Tác nhân	Celery Worker		
Mô tả	Celery Worker thực thi benchmark trong background. Quá trình này bao gồm triển khai EC2 instances, thực thi requests, thu thập metrics từ CloudWatch, và lưu kết quả vào database.		
Điều kiện trước (Pre-Condition)	- BenchmarkRun đã được tạo với status = "pending" - Celery task đã được enqueue - AWS credentials đã được cấu hình trong project		
Điều kiện sau (Post-Condition)	- Tất cả configs đã được benchmark - BenchmarkResults được lưu trong database - BenchmarkRun status = "completed" - Tất cả EC2 instances đã được cleanup		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Celery worker nhận task execute_benchmark_task với benchmark_id	Worker load BenchmarkRun và Project từ database
	2		Worker load danh sách AWSConfigs cần benchmark
	3		Worker update BenchmarkRun status = "running", started_at = NOW()

	4		Worker initialize BenchmarkRunner service
	5		**Với mỗi config:** Worker gọi UC-27 để deploy EC2 instance
	6		Worker đợi instance ready (health check)
	7		Worker đợi CloudWatch Agent khởi động (45 giây)
	8		Worker ghi nhận benchmark_start_time
	9		Worker thực thi requests: - Nếu use_k6 = true: Gọi k6 runner với virtual_users - Nếu use_k6 = false: Gửi sequential requests bằng httpx
	10		Worker thu thập latencies từ mỗi request
	11		Worker ghi nhận benchmark_end_time
	12		Worker gọi UC-28 để thu thập CloudWatch metrics
	13		Worker gọi UC-25 để phát hiện anomalies
	14		Worker gọi UC-26 để tính toán metrics
	15		Worker tạo BenchmarkResult record
	16		Worker gọi UC-33 để lưu result vào database
	17		Worker gọi UC-30 để cleanup EC2 instance
	18		Worker update progress (configs_completed++)
	19		Worker kiểm tra budget: Nếu vượt budget và hard_limit = true, dừng lại
	20		Worker update BenchmarkRun status = "completed", completed_at = NOW()
	21		Worker trigger UC-32 để generate recommendations tự động

Bảng 3.32. UC-32: Tạo khuyến nghị tự động

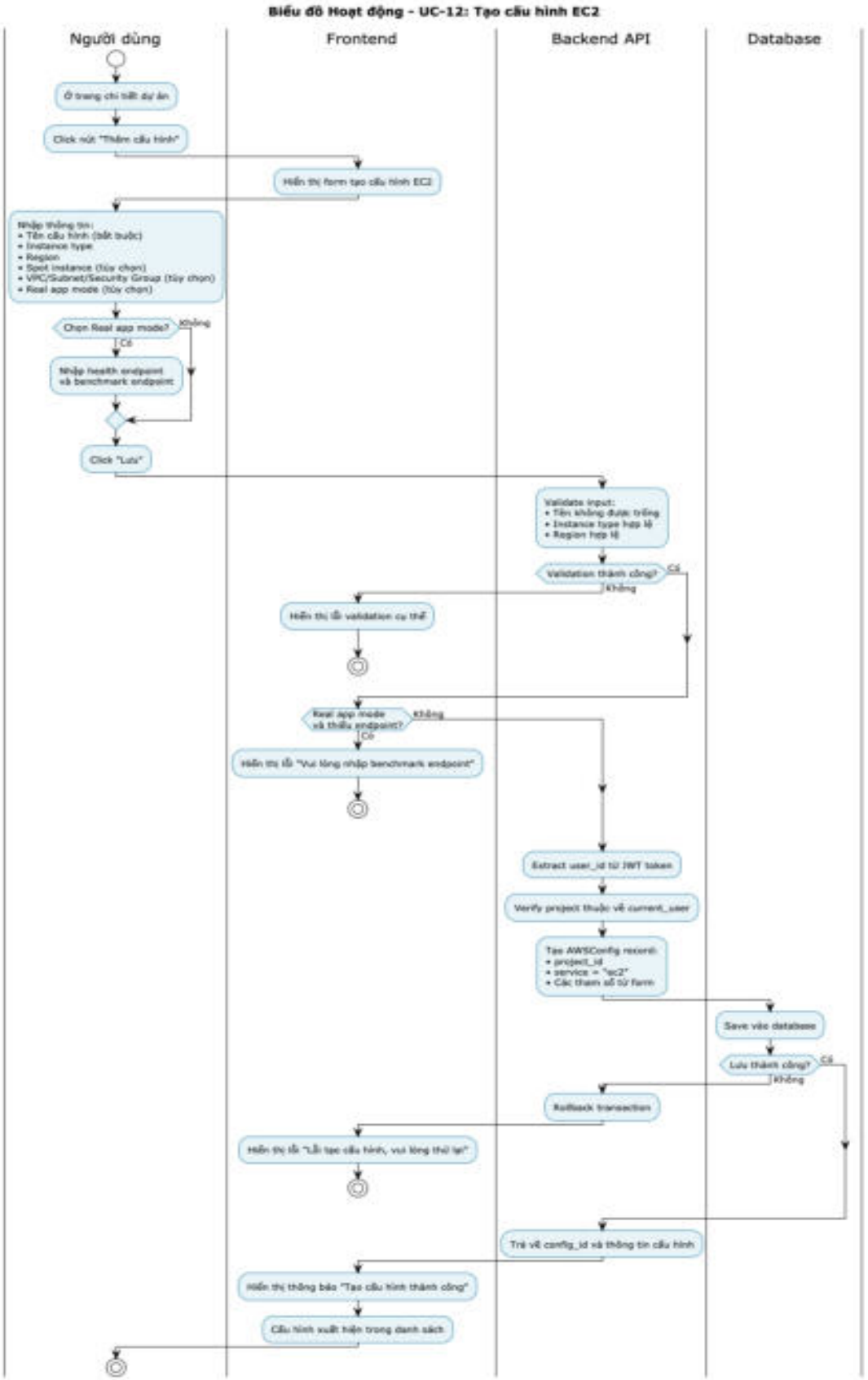
Mã ca sử dụng	**UC-32**		
Tên ca sử dụng	Tạo khuyến nghị tự động		
Tác nhân	Hệ thống (System)		
Mô tả	Hệ thống tự động tạo AI recommendations sau khi benchmark baseline hoàn thành. Quá trình này được trigger tự động bởi Celery worker sau khi benchmark completed, không cần người dùng can thiệp.		
Điều kiện trước (Pre-Condition)	- Benchmark baseline đã hoàn thành (status = "completed") - Có kết quả benchmark cho baseline config - Dự án có baseline_config_id được set		
Điều kiện sau (Post-Condition)	- Recommendations được tạo và lưu trong database - Status = "pending" (chờ người dùng approve/reject)		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Celery worker hoàn thành benchmark baseline	Worker kiểm tra benchmark status = "completed"
	2		Worker kiểm tra project có baseline_config_id không
	3		Worker load baseline BenchmarkResult từ database
	4		Worker gọi AI Service để generate recommendations (tương tự UC-23)
	5		AI Service phân tích baseline metrics và gọi OpenAI API
	6		AI Service lưu recommendations vào database
	7		Worker log "Recommendations generated successfully"

Bảng 3.33. UC-33: Lưu trữ kết quả

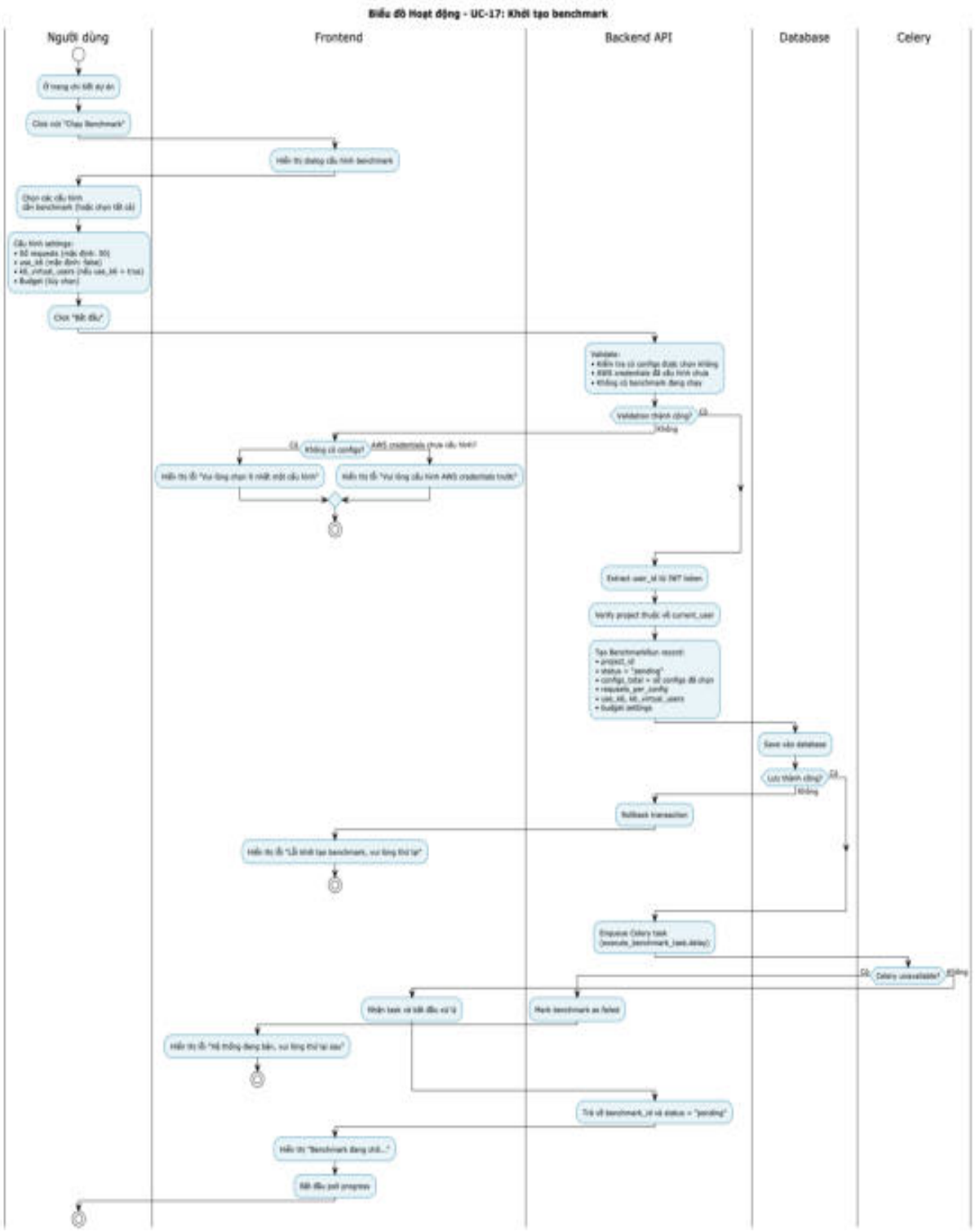
Mã ca sử dụng	**UC-33**		
Tên ca sử dụng	Lưu trữ kết quả		
Tác nhân	Hệ thống (System)		
Mô tả	Hệ thống lưu trữ kết quả benchmark vào database. Kết quả bao gồm tất cả metrics đã tính toán, raw latencies, anomalies, và resource utilization data.		
Điều kiện trước (Pre-Condition)	- Metrics đã được tính toán (UC-26) - Anomalies đã được phát hiện (UC-25) - CloudWatch metrics đã được thu thập (UC-28)		
Điều kiện sau (Post-Condition)	- BenchmarkResult record đã được tạo trong database - Tất cả dữ liệu đã được lưu trữ an toàn		
Kịch bản		**Hành động của tác nhân**	**Phản hồi của hệ thống**
	1	Hệ thống nhận yêu cầu lưu kết quả benchmark	Hệ thống tạo BenchmarkResult record với: - benchmark_run_id - configuration_id - source = "fresh" (hoặc "cached" nếu sử dụng cache)
	2		Hệ thống lưu latency metrics: - latency_p50_ms, latency_p95_ms, latency_p99_ms - latency_mean_ms, latency_std_ms
	3		Hệ thống lưu cold start data: - cold_start_detected (boolean) - cold_start_first_request_ms-cold_start_warm_avg_ms
	4		Hệ thống lưu request statistics: - total_requests, failed_requests - error_rate

	5		Hệ thống lưu cost metrics: - cost_per_request_usd - cost_per_1k_requests_usd - benchmark_cost_usd
	6		Hệ thống lưu raw latencies vào raw_latencies_ms (JSONB array)
	7		Hệ thống lưu anomalies vào anomalies (JSONB array)
	8		Hệ thống lưu confidence score
	9		Hệ thống lưu CPU metrics: - cpu_avg_percent, cpu_p95_percent, cpu_max_percent
	10		Hệ thống lưu resource utilization verdict và reason
	11		Hệ thống lưu disk I/O metrics: - disk_read_bytes_avg, disk_read_bytes_total - disk_write_bytes_avg, disk_write_bytes_total
	12		Hệ thống lưu network I/O metrics: - network_in_bytes_avg, network_in_bytes_total - network_out_bytes_avg, network_out_bytes_total
	13		Hệ thống lưu status check failures
	14		Hệ thống commit transaction vào database
	15		Hệ thống log "Result saved successfully"

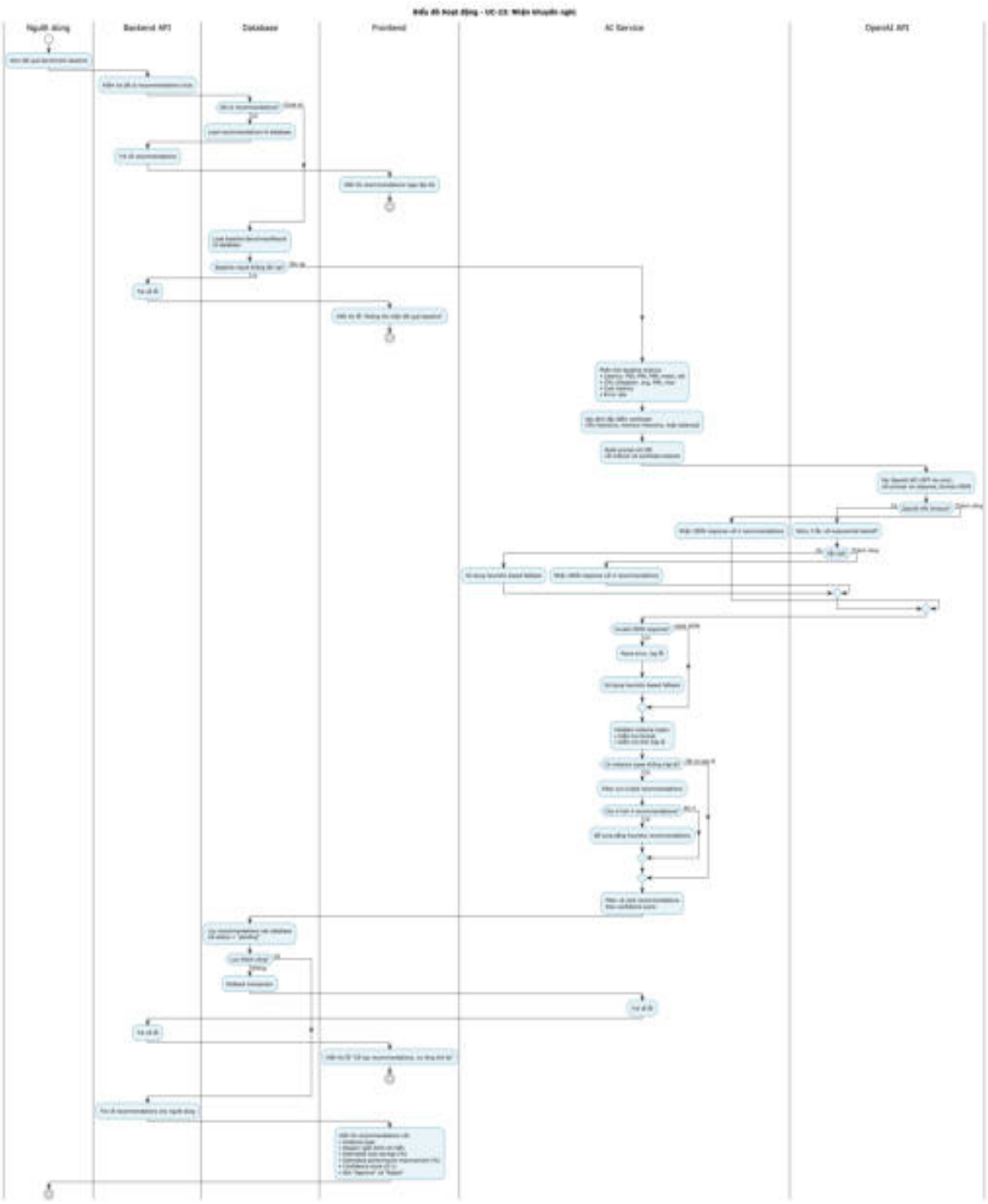
3.5. Biểu đồ hoạt động



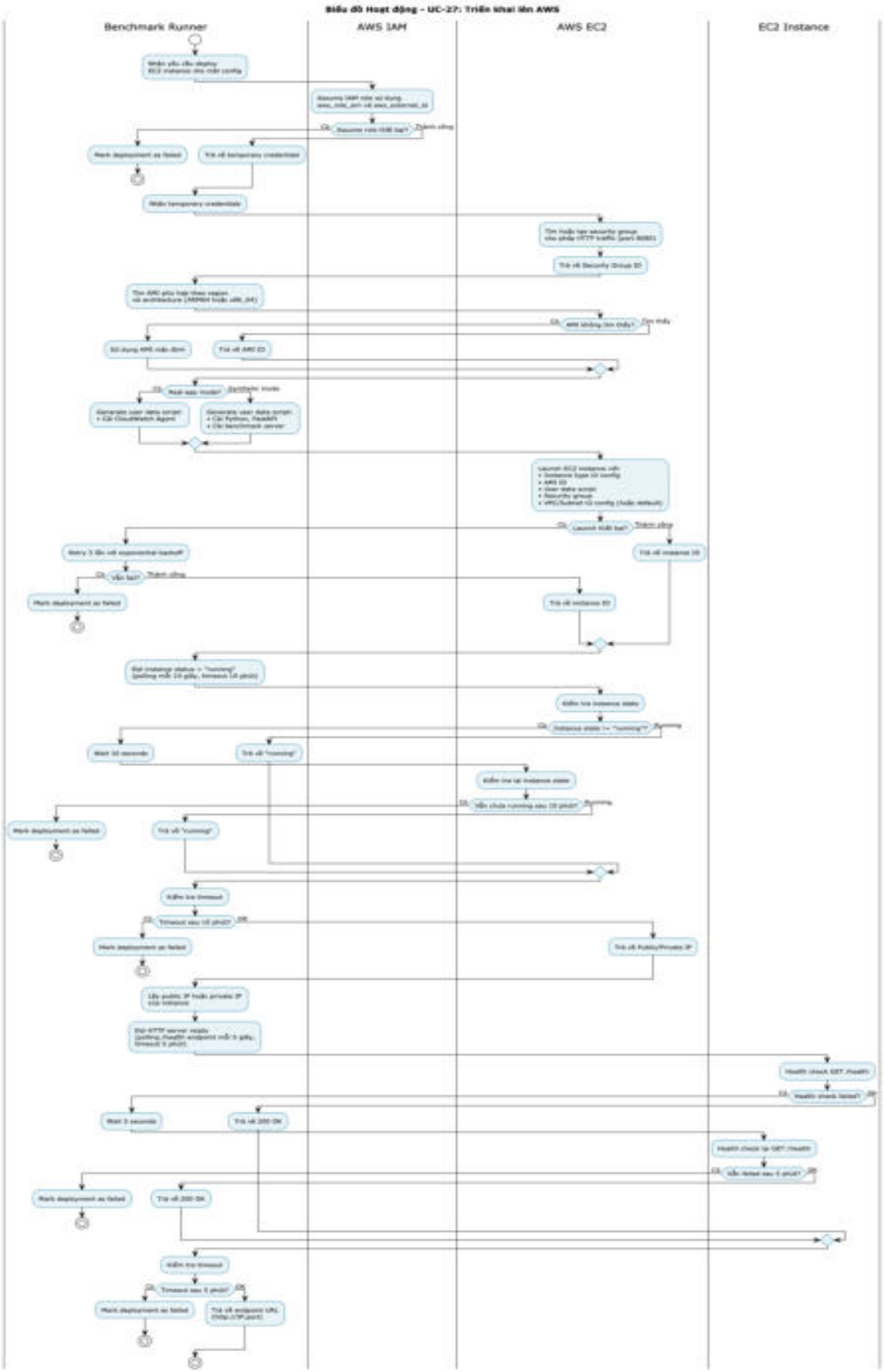
Hình 3.7. Biểu đồ hoạt động tạo cấu hình EC2



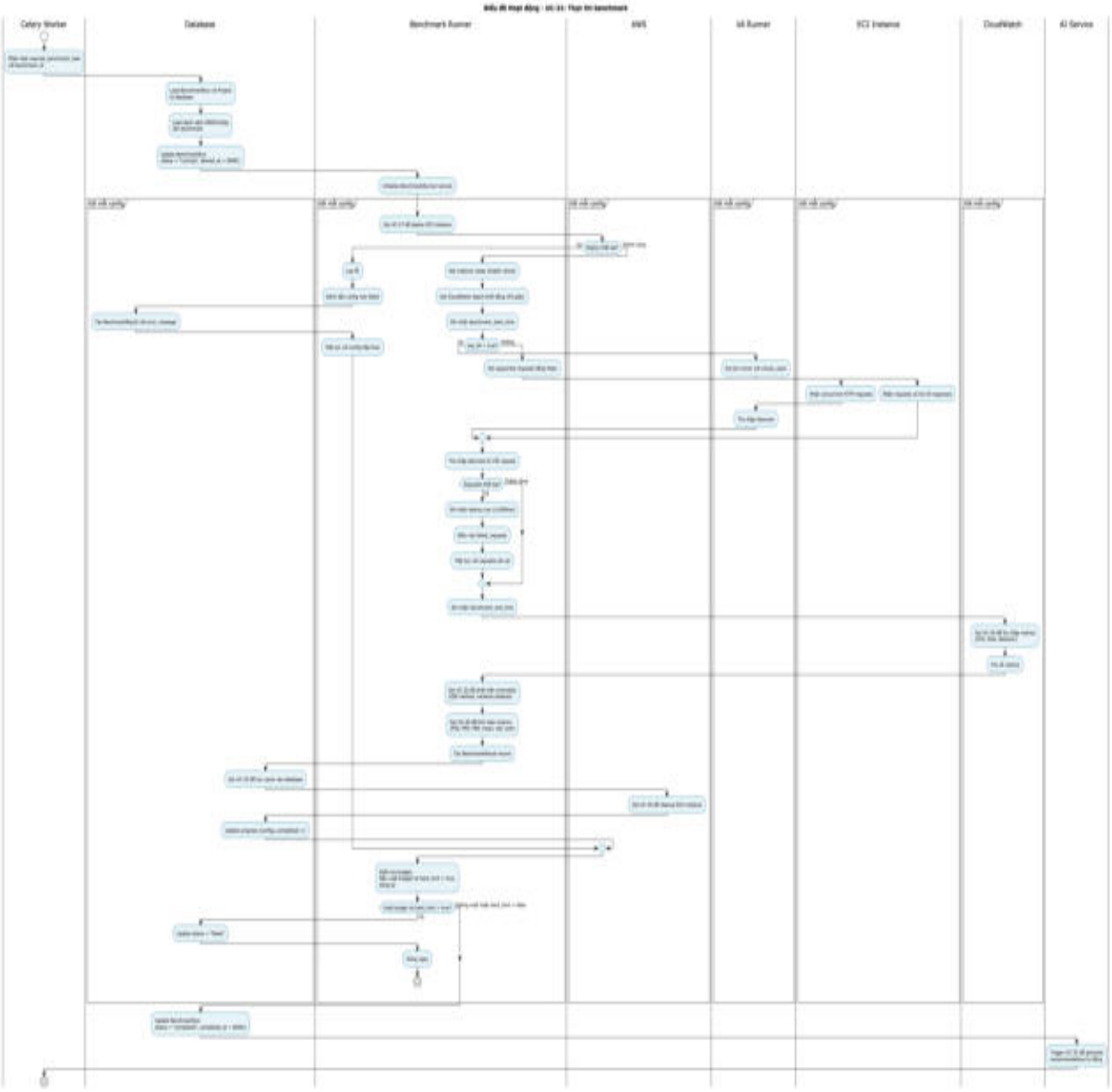
Hình 3.8. Biểu đồ hoạt động khởi tạo benchmark



Hình 3.9. Biểu đồ hoạt động nhận khuyến nghị

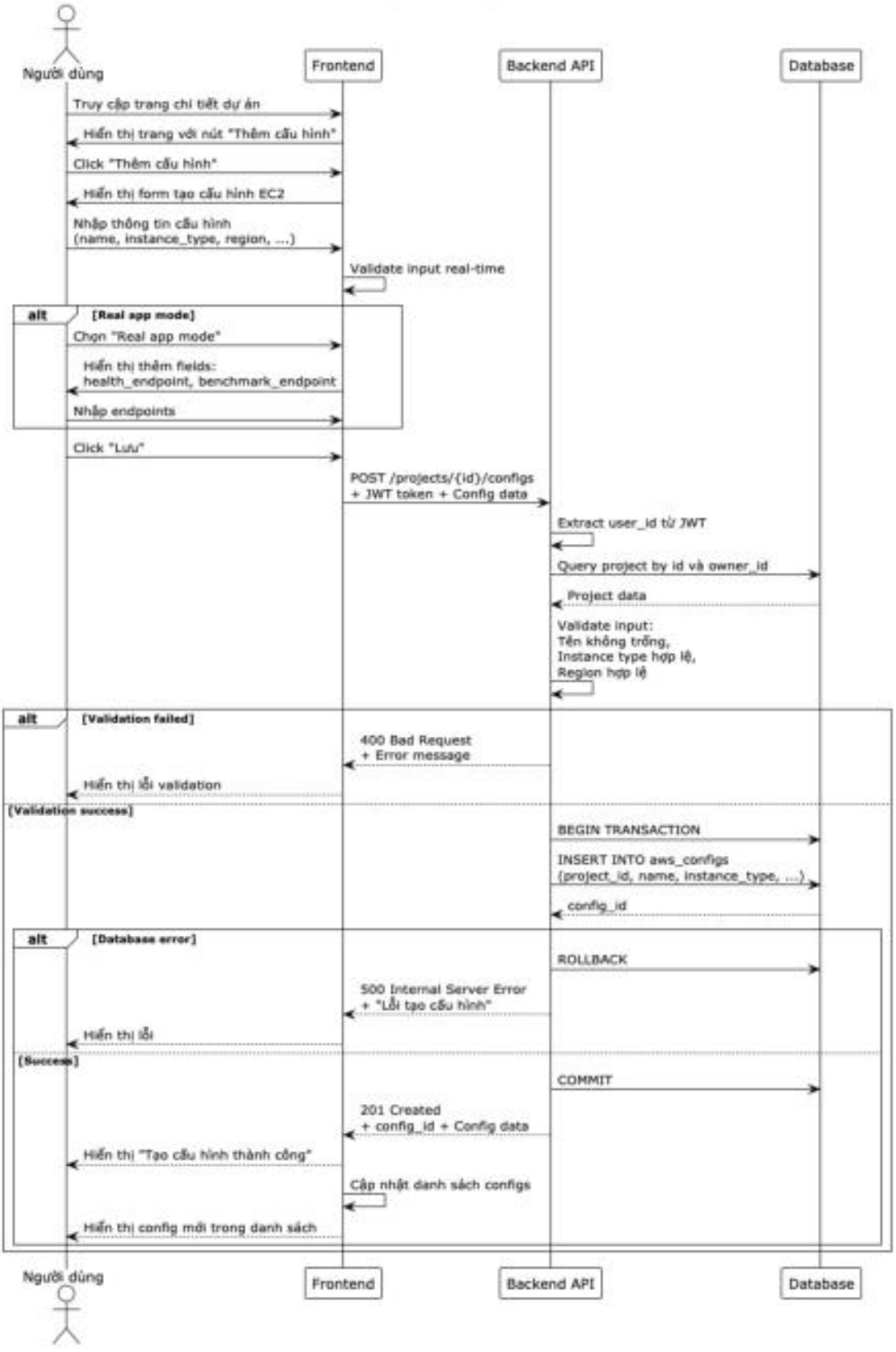


Hình 3.10. Biểu đồ hoạt động triển khai lên AWS

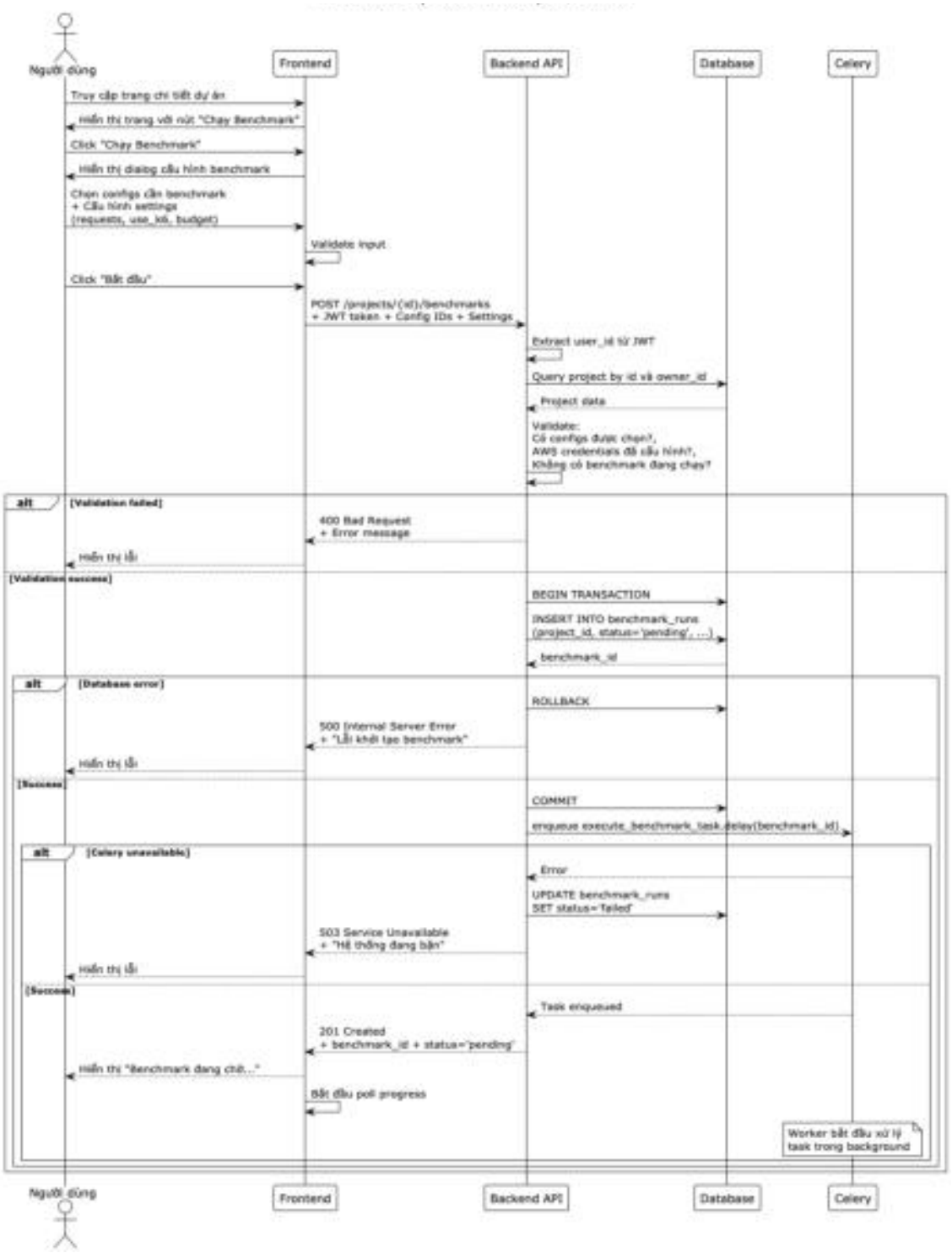


Hình 3.11. Biểu đồ hoạt động thực thi benchmark

3.6. Biểu đồ tuần tự

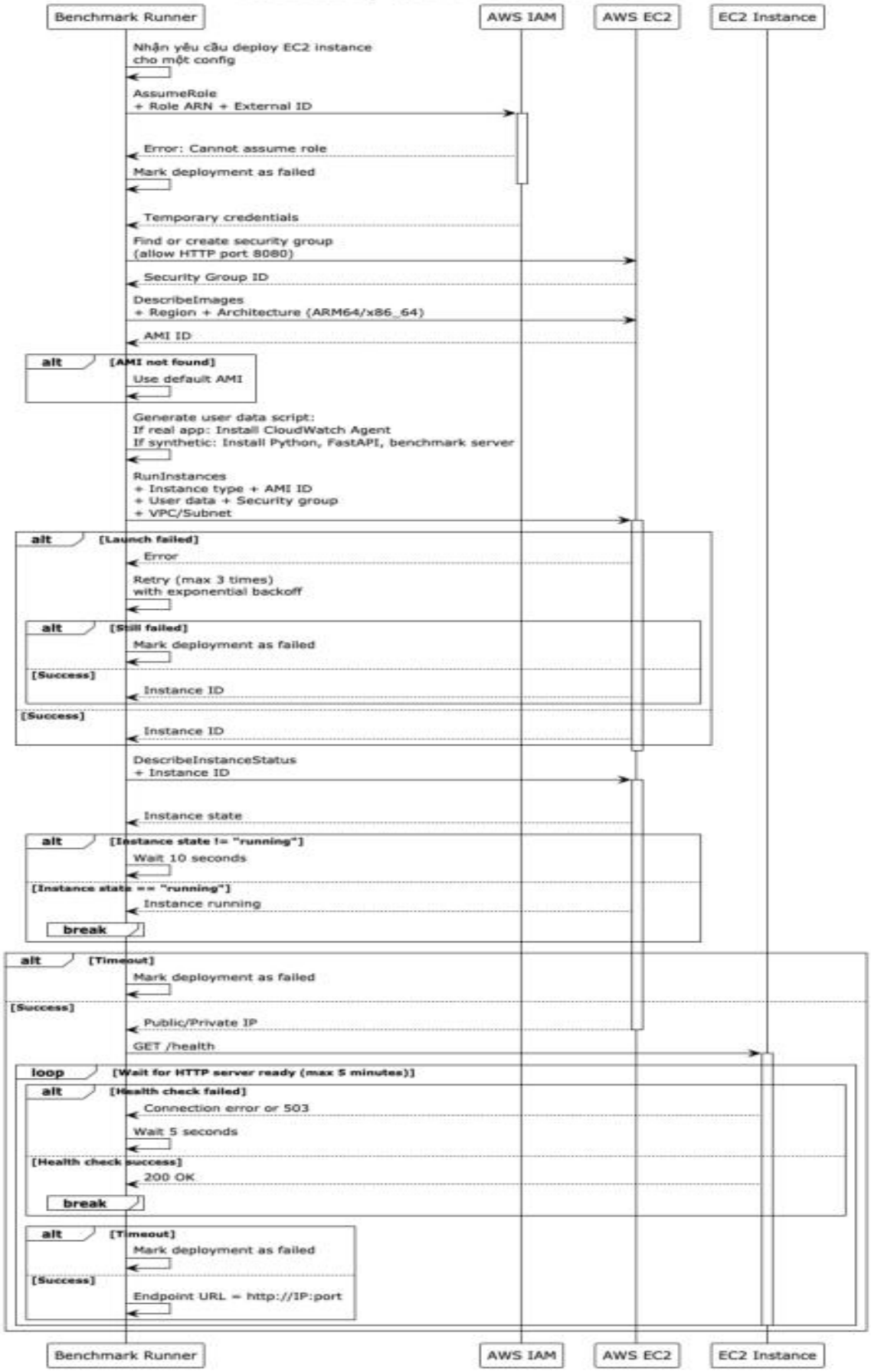


Hình 3.12. Biểu đồ tuần tự tạo cấu hình EC2

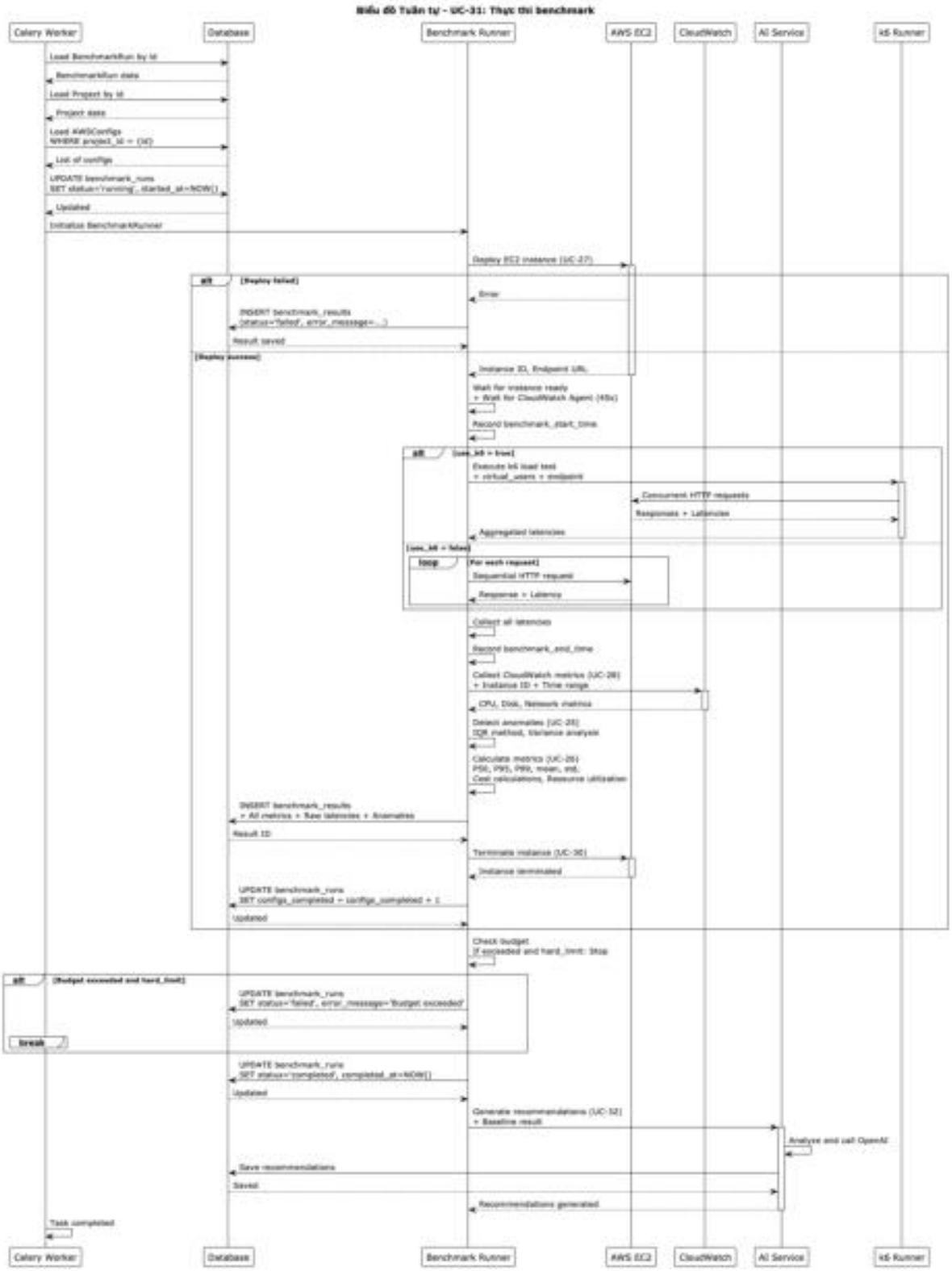


Hình 3.13. Biểu đồ tuần tự khởi tạo benchmark

Hình 3.14. Biểu đồ tuần tự nhận khuyến nghị



Hình 3.15. Biểu đồ tuần tự triển khai lên AWS

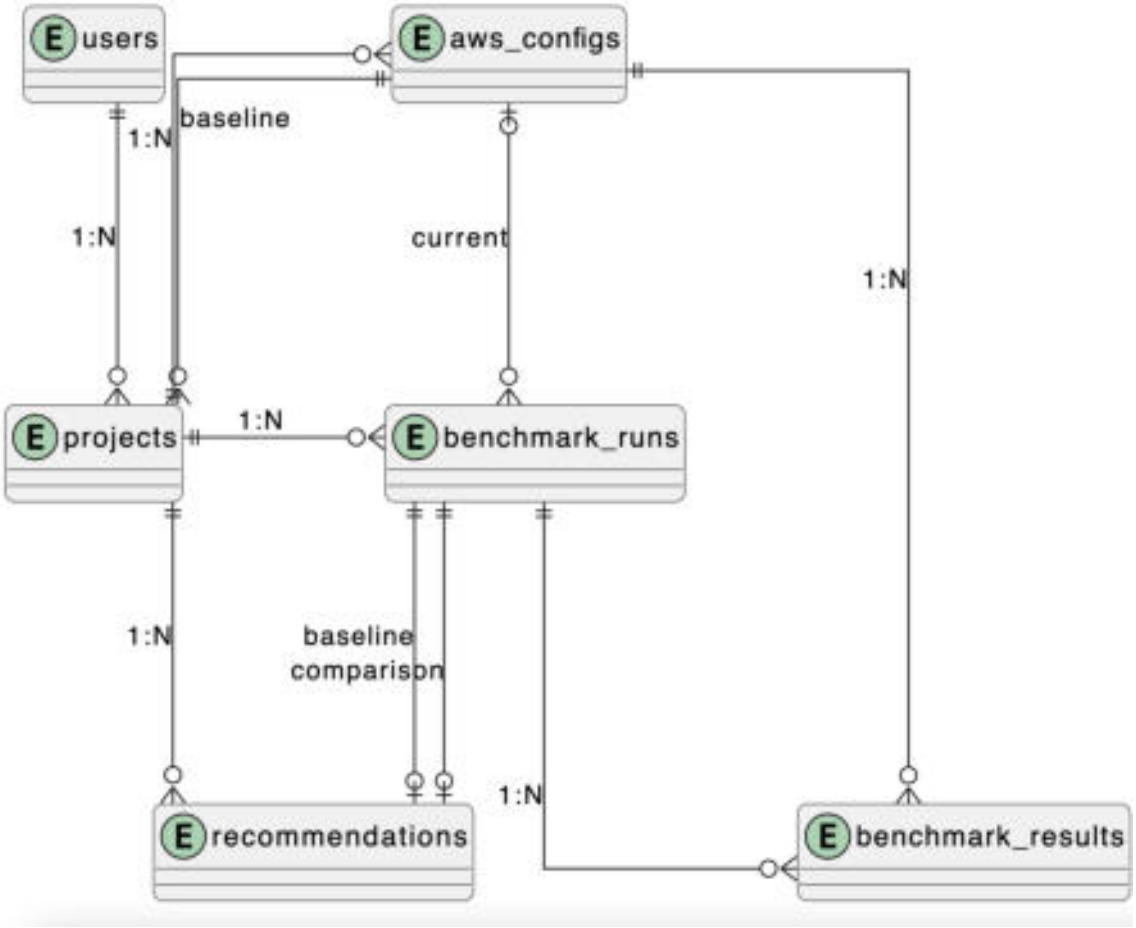


Hình 3.16. Biểu đồ tuần tự thực thi benchmark

3.7. Cơ sở dữ liệu

Sơ đồ quan hệ (ERD)

Hệ thống sử dụng cơ sở dữ liệu quan hệ PostgreSQL với 6 bảng chính:



Quan hệ giữa các bảng:

- 1 user có nhiều projects (1:N)
- 1 project có nhiều aws_configs (1:N)
- 1 project có nhiều benchmark_runs (1:N)
- 1 benchmark_run có nhiều benchmark_results (1:N)
- 1 project có nhiều recommendations (1:N)
- 1 benchmark_result thuộc về 1 aws_config (N:1)

Bảng users

Bảng này lưu trữ thông tin người dùng của hệ thống. Mỗi người dùng có thể tạo nhiều dự án để đánh giá và tối ưu hóa cấu hình AWS.

Tên cột	Kiểu dữ liệu	Mô tả	Ghi chú
id	UUID	Khóa chính	PK, Not Null
email	VARCHAR(255)	Email đăng nhập	Unique, Not Null, Indexed

Tên cột	Kiểu dữ liệu	Mô tả	Ghi chú
hashed_password	VARCHAR(255)	Mật khẩu đã hash (bcrypt)	Not Null
full_name	VARCHAR(255)	Tên đầy đủ	Nullable
is_active	BOOLEAN	Trạng thái tài khoản	Default: TRUE
is_superuser	BOOLEAN	Quyền quản trị viên	Default: FALSE
created_at	TIMESTAMPTZ	Thời điểm tạo	Not Null, Default: NOW()
updated_at	TIMESTAMPTZ	Thời điểm cập nhật	Not Null, Default: NOW()

Quan hệ:

- 1 user có nhiều projects (1:N)

1.1 Bảng projects

Bảng này lưu trữ các dự án đánh giá của người dùng. Mỗi dự án đại diện cho một workload cần được benchmark và tối ưu hóa. Dự án chứa thông tin về AWS credentials (IAM role ARN và External ID) để hệ thống có thể truy cập tài khoản AWS của người dùng.

Tên cột	Kiểu dữ liệu	Mô tả	Ghi chú
id	UUID	Khóa chính	PK, Not Null
name	VARCHAR(255)	Tên dự án	Not Null
description	TEXT	Mô tả dự án	Nullable
mode	VARCHAR(50)	Chế độ hoạt động	Default: 'optimization'
status	VARCHAR(50)	Trạng thái dự án	Default: 'created'
owner_id	UUID	Người sở hữu dự án	FK → users.id, Not Null, CASCADE DELETE
aws_role_arn	VARCHAR(500)	IAM role ARN để assume role vào AWS account của client	Nullable
aws_external_id	VARCHAR(100)	External ID cho IAM role (bảo mật)	Nullable
baseline_config_id	UUID	Cấu hình baseline (cho optimization mode)	FK → aws_configs.id, Nullable
created_at	TIMESTAMPTZ	Thời điểm tạo	Not Null, Default: NOW()
updated_at	TIMESTAMPTZ	Thời điểm cập nhật	Not Null, Default: NOW()

Quan hệ:

- N projects thuộc về 1 user (N:1)
- 1 project có nhiều aws_configs (1:N)
- 1 project có nhiều benchmark_runs (1:N)
- 1 project có nhiều recommendations (1:N)

Giải thích:

- `aws_role_arn` và `aws_external_id` được sử dụng để assume role vào tài khoản AWS của người dùng, cho phép hệ thống triển khai EC2 instances và thu thập metrics mà không cần lưu trữ AWS credentials trực tiếp.
- `baseline_config_id` trỏ đến cấu hình hiện tại (production) để so sánh với các đề xuất tối ưu.

1.2 Bảng aws_configs

Bảng này lưu trữ các cấu hình AWS EC2 cần được benchmark. Mỗi cấu hình đại diện cho một loại máy ảo cụ thể với các tham số kỹ thuật xác định như instance type, region, và các tùy chọn về network, security groups.

Tên cột	Kiểu dữ liệu	Mô tả	Ghi chú
id	UUID	Khóa chính	PK, Not Null
name	VARCHAR(255)	Tên cấu hình	Not Null
description	TEXT	Mô tả cấu hình	Nullable
is_baseline	BOOLEAN	Đánh dấu là cấu hình baseline	Default: FALSE
service	VARCHAR(50)	Loại dịch vụ AWS	Default: 'ec2', Not Null
region	VARCHAR(50)	Vùng AWS	Not Null
project_id	UUID	Dự án sở hữu cấu hình	FK → projects.id, Not Null, CASCADE DELETE
ec2_instance_type	VARCHAR(50)	Loại máy ảo EC2 (ví dụ: t3.medium)	Nullable
ec2_spot	BOOLEAN	Sử dụng Spot instances	Nullable
ec2_existing_instance_id	VARCHAR(100)	ID máy ảo hiện có (cho optimization mode)	Nullable
ec2_existing_endpoint_url	VARCHAR(500)	URL endpoint trực tiếp (nếu có)	Nullable

Tên cột	Kiểu dữ liệu	Mô tả	Ghi chú
ec2_port	INTEGER	Port HTTP server (default: 8080)	Default: 8080
ec2_user_data	TEXT	User data script cho máy ảo mới	Nullable
ec2_vpc_id	VARCHAR(100)	VPC ID	Nullable
ec2_subnet_id	VARCHAR(100)	Subnet ID	Nullable
ec2_security_group_id	VARCHAR(100)	Security Group ID	Nullable
ec2_use_real_app	BOOLEAN	Sử dụng ứng dụng thực tế thay vì benchmark server	Default: FALSE
ec2_app_health_endpoint	VARCHAR(200)	Endpoint health check (ví dụ: /health)	Default: '/health'
ec2_app_benchmark_endpoint	VARCHAR(200)	Endpoint để benchmark (ví dụ: /api/users)	Default: '/api/users'
ec2_operating_system	VARCHAR(50)	Hệ điều hành ('ubuntu' hoặc 'amazon_linux')	Nullable
estimated_cost_per_request	FLOAT	Chi phí ước tính mỗi request (USD)	Nullable
estimated_cost_per_hour	FLOAT	Chi phí ước tính mỗi giờ (USD)	Nullable
created_at	TIMESTAMPTZ	Thời điểm tạo	Not Null, Default: NOW()

Quan hệ:

- N aws_configs thuộc về 1 project (N:1)
- 1 aws_config có nhiều benchmark_results (1:N)

Giải thích:

- `ec2_existing_instance_id`: Cho phép benchmark máy ảo đang chạy mà không cần tạo mới.
- `ec2_use_real_app`: Khi TRUE, hệ thống sẽ benchmark ứng dụng thực tế thay vì synthetic benchmark server.
- `ec2_app_benchmark_endpoint`: Đường dẫn API endpoint cần benchmark (ví dụ: `/api/users`, `/api/products`).

1.3 Bảng benchmark_runs

Bảng này lưu trữ thông tin về mỗi lần thực thi benchmark. Một benchmark run có thể bao gồm nhiều cấu hình và tạo ra nhiều benchmark results. Hệ thống hỗ trợ cả sequential requests và k6 load testing.

Tên cột	Kiểu dữ liệu	Mô tả	Ghi chú
id	UUID	Khóa chính	PK, Not Null
project_id	UUID	Dự án sở hữu benchmark	FK → projects.id, Not Null, CASCADE DELETE
status	VARCHAR(50)	Trạng thái (pending, running, completed, failed, cancelled)	Default: 'pending'
requests_per_config	INTEGER	Số requests cho mỗi cấu hình	Default: 50
max_runtime_seconds	INTEGER	Thời gian tối đa (giây)	Default: 600
use_cache	BOOLEAN	Sử dụng cache kết quả	Default: TRUE
cache_max_age_hours	INTEGER	Tuổi tối đa của cache (giờ)	Default: 24
use_k6	BOOLEAN	Sử dụng k6 load testing	Default: FALSE
k6_virtual_users	INTEGER	Số virtual users cho k6	Default: 5
budget_max_usd	FLOAT	Ngân sách tối đa (USD)	Nullable
budget_hard_limit	BOOLEAN	Dừng khi vượt budget	Default: TRUE
configs_completed	INTEGER	Số cấu hình đã hoàn thành	Default: 0
configs_total	INTEGER	Tổng số cấu hình	Default: 0
current_config_id	UUID	Cấu hình đang xử lý	FK → aws_configs.id, Nullable
total_cost_usd	FLOAT	Tổng chi phí (USD)	Default: 0.0
started_at	TIMESTAMPTZ	Thời điểm bắt đầu	Nullable
completed_at	TIMESTAMPTZ	Thời điểm hoàn thành	Nullable
created_at	TIMESTAMPTZ	Thời điểm tạo	Not Null, Default: NOW()
error_message	VARCHAR(1000)	Thông báo lỗi (nếu có)	Nullable

Quan hệ:

- N benchmark_runs thuộc về 1 project (N:1)
- 1 benchmark_run có nhiều benchmark_results (1:N)

Giải thích:

- `use_k6`: Khi TRUE, sử dụng k6 thay vì sequential httpx requests để tạo tải đồng thời.
- `k6_virtual_users`: Số lượng virtual users chạy đồng thời trong k6 test.

1.4 Bảng benchmark_results

Bảng này lưu trữ kết quả benchmark chi tiết cho từng cấu hình. Mỗi result chứa các metrics về latency, cost, errors và resource utilization từ CloudWatch. Hệ thống cũng lưu trữ raw latencies và anomalies để có thể phân tích lại sau này.

Tên cột	Kiểu dữ liệu	Mô tả	Ghi chú
id	UUID	Khóa chính	PK, Not Null
benchmark_run_id	UUID	Lần benchmark chứa result này	FK → benchmark_runs.id, Not Null, CASCADE DELETE
configuration_id	UUID	Cấu hình được benchmark	FK → aws_configs.id, Not Null, CASCADE DELETE
source	VARCHAR(50)	Nguồn kết quả ('fresh' hoặc 'cached')	Default: 'fresh'
cache_age_hours	FLOAT	Tuổi của cache (nếu source='cached')	Nullable
latency_p50_ms	FLOAT	Latency phân vị 50 (ms)	Nullable
latency_p95_ms	FLOAT	Latency phân vị 95 (ms)	Nullable
latency_p99_ms	FLOAT	Latency phân vị 99 (ms)	Nullable
latency_mean_ms	FLOAT	Latency trung bình (ms)	Nullable
latency_std_ms	FLOAT	Độ lệch chuẩn latency (ms)	Nullable
cold_start_detected	BOOLEAN	Phát hiện cold start	Default: FALSE

Tên cột	Kiểu dữ liệu	Mô tả	Ghi chú
cold_start_first_request_ms	FLOAT	Latency request đầu tiên (cold start)	Nullable
cold_start_warm_avg_ms	FLOAT	Latency trung bình các request sau (warm)	Nullable
total_requests	INTEGER	Tổng số requests	Default: 0
failed_requests	INTEGER	Số requests thất bại	Default: 0
error_rate	FLOAT	Tỷ lệ lỗi (0-1)	Default: 0.0
cost_per_request_usd	FLOAT	Chi phí mỗi request (USD)	Nullable
cost_per_1k_requests_usd	FLOAT	Chi phí mỗi 1000 requests (USD)	Nullable
benchmark_cost_usd	FLOAT	Tổng chi phí benchmark (USD)	Default: 0.0
raw_latencies_ms	JSONB	Mảng raw latencies (ms)	Default: []
anomalies	JSONB	Danh sách anomalies phát hiện được	Default: []
confidence	FLOAT	Điểm tin cậy (0-1)	Default: 0.0
cpu_avg_percent	FLOAT	CPU utilization trung bình (%)	Nullable
cpu_p95_percent	FLOAT	CPU utilization P95 (%)	Nullable
cpu_max_percent	FLOAT	CPU utilization tối đa (%)	Nullable
resourcing_verdict	VARCHAR(50)	Đánh giá cấu hình ('under-provisioned', 'optimized', 'over-provisioned')	Nullable
resourcing_reason	TEXT	Lý do đánh giá	Nullable
disk_read_bytes_avg	FLOAT	Disk read trung bình (bytes/min)	Nullable
disk_read_bytes_total	FLOAT	Disk read tổng (bytes)	Nullable
disk_write_bytes_avg	FLOAT	Disk write trung bình (bytes/min)	Nullable
disk_write_bytes_total	FLOAT	Disk write tổng (bytes)	Nullable

Tên cột	Kiểu dữ liệu	Mô tả	Ghi chú
network_in_bytes_avg	FLOAT	Network in trung bình (bytes/min)	Nullable
network_in_bytes_total	FLOAT	Network in tổng (bytes)	Nullable
network_out_bytes_avg	FLOAT	Network out trung bình (bytes/min)	Nullable
network_out_bytes_total	FLOAT	Network out tổng (bytes)	Nullable
status_check_failed_count	INTEGER	Số lần status check thất bại	Nullable
status_check_failed_instance_count	INTEGER	Số lần instance status check thất bại	Nullable
status_check_failed_system_count	INTEGER	Số lần system status check thất bại	Nullable
error_message	VARCHAR(1000)	Thông báo lỗi (nếu benchmark thất bại)	Nullable
created_at	TIMESTAMPTZ	Thời điểm tạo	Not Null, Default: NOW()

Quan hệ:

- N benchmark_results thuộc về 1 benchmark_run (N:1)
- 1 benchmark_result thuộc về 1 aws_config (N:1)

Giải thích:

- `raw_latencies_ms`: Lưu trữ toàn bộ mảng latencies để có thể phân tích lại sau này.
- `anomalies`: Lưu trữ danh sách các giá trị bất thường được phát hiện bằng IQR method.
- `resourcing_verdict`: Đánh giá mức độ phù hợp của cấu hình dựa trên CPU utilization (ví dụ: CPU < 20% → over-provisioned).

1.5 Bảng recommendations

Bảng này lưu trữ các khuyến nghị tối ưu được tạo bởi AI sau khi phân tích kết quả benchmark baseline. Mỗi recommendation chứa danh sách các instance types được đề xuất cùng với lý do, ước tính tiết kiệm chi phí và cải thiện hiệu năng.

Tên cột	Kiểu dữ liệu	Mô tả	Ghi chú
id	UUID	Khóa chính	PK, Not Null
project_id	UUID	Dự án sở hữu recommendation	FK → projects.id, Not Null, CASCADE DELETE
baseline_benchmark_id	UUID	Benchmark baseline được phân tích	FK → benchmark_runs.id, Nullable, SET NULL
status	VARCHAR(50)	Trạng thái (pending, approved, rejected, deployed, benchmarked, cleaned_up)	Default: 'pending'
recommended_instance_types	JSONB	Danh sách instance types được đề xuất với reasoning	Not Null
analysis	TEXT	Phân tích đặc điểm workload từ AI	Nullable
user_decision	VARCHAR(50)	Quyết định của người dùng ('approved' hoặc 'rejected')	Nullable
user_notes	TEXT	Ghi chú của người dùng	Nullable
comparison_benchmark_id	UUID	Benchmark so sánh sau khi deploy recommended instances	FK → benchmark_runs.id, Nullable, SET NULL
created_at	TIMESTAMPTZ	Thời điểm tạo	Not Null, Default: NOW()
approved_at	TIMESTAMPTZ	Thời điểm được approve	Nullable
deployed_at	TIMESTAMPTZ	Thời điểm deploy recommended instances	Nullable
benchmarked_at	TIMESTAMPTZ	Thời điểm hoàn thành comparison benchmark	Nullable
cleaned_up_at	TIMESTAMPTZ	Thời điểm cleanup recommended instances	Nullable

Quan hệ:

- N recommendations thuộc về 1 project (N:1)

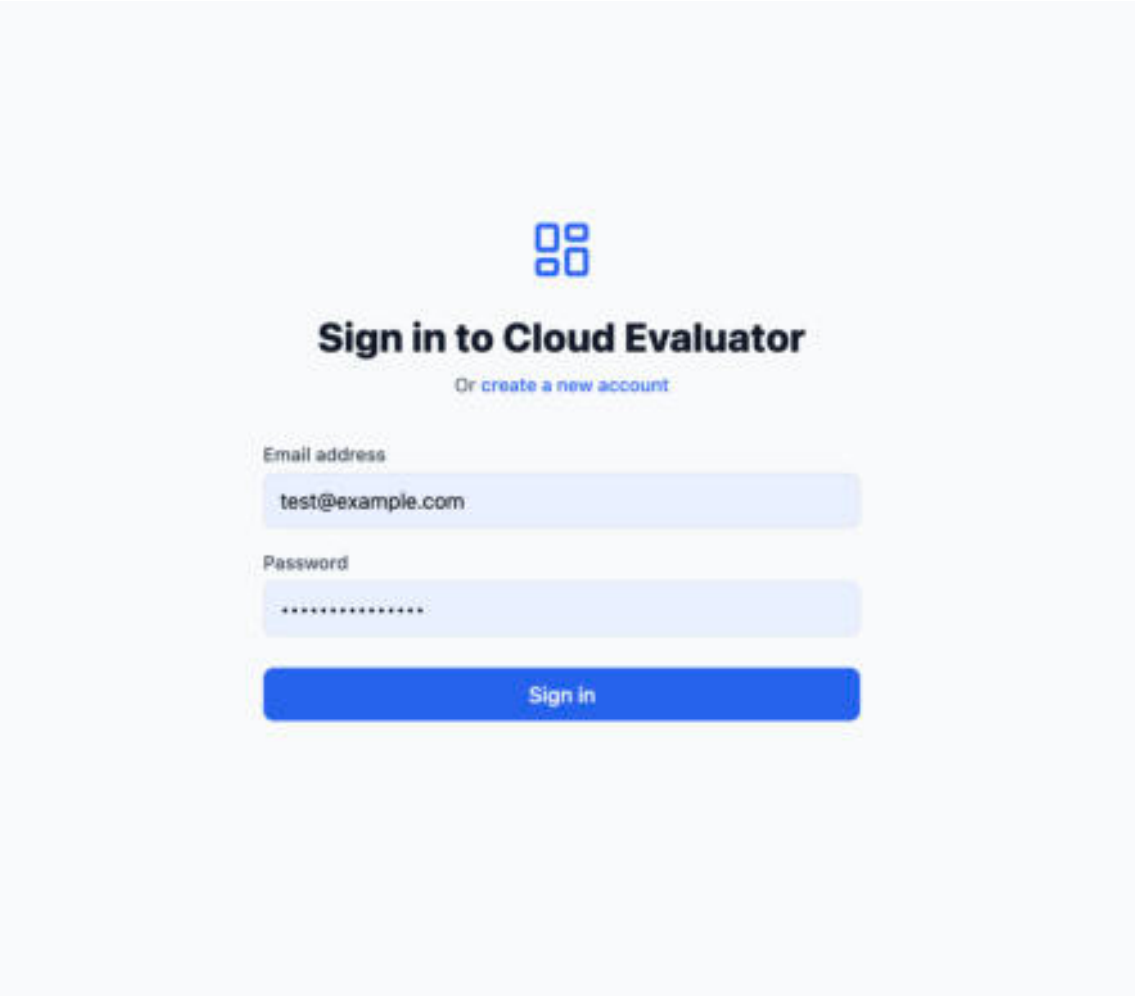
Giải thích:

- `recommended_instance_types`: JSON array chứa các instance types được đề xuất, mỗi item có: instance_type, reason, estimated_cost_savings_percent, estimated_performance_improvement_percent, confidence.

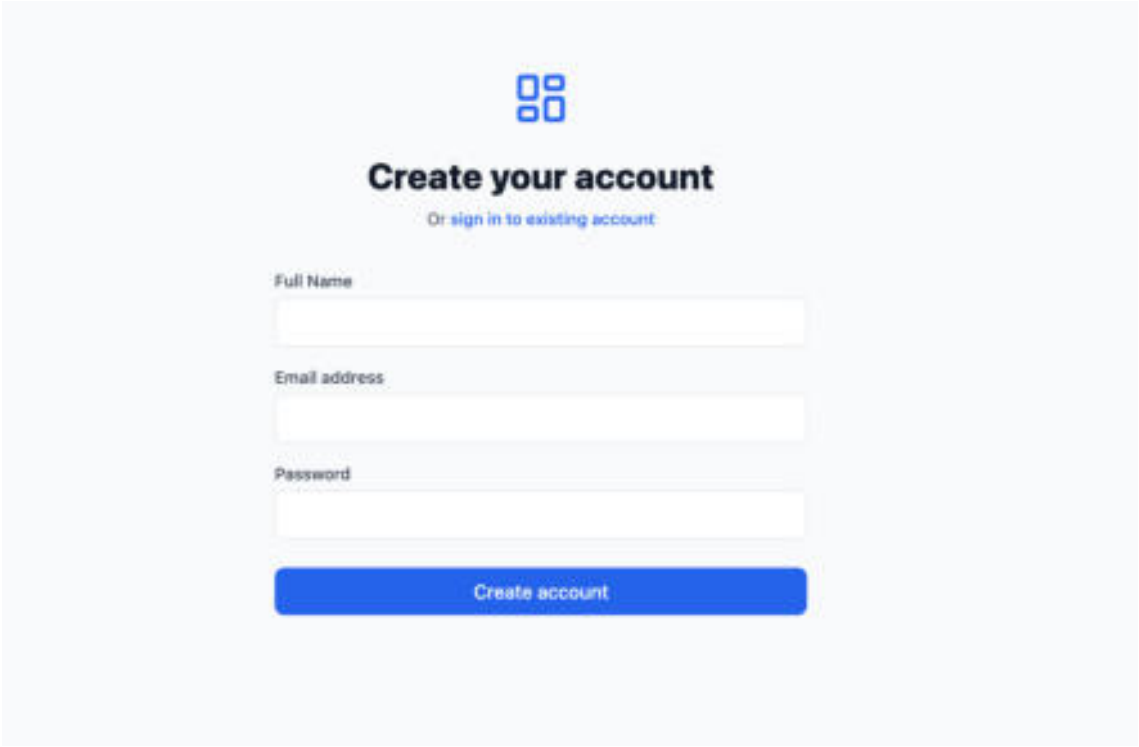
`status`: Theo dõi lifecycle của recommendation từ khi được tạo đến khi cleanup.

CHƯƠNG 4: XÂY DỰNG VÀ TRIỂN KHAI HỆ THỐNG

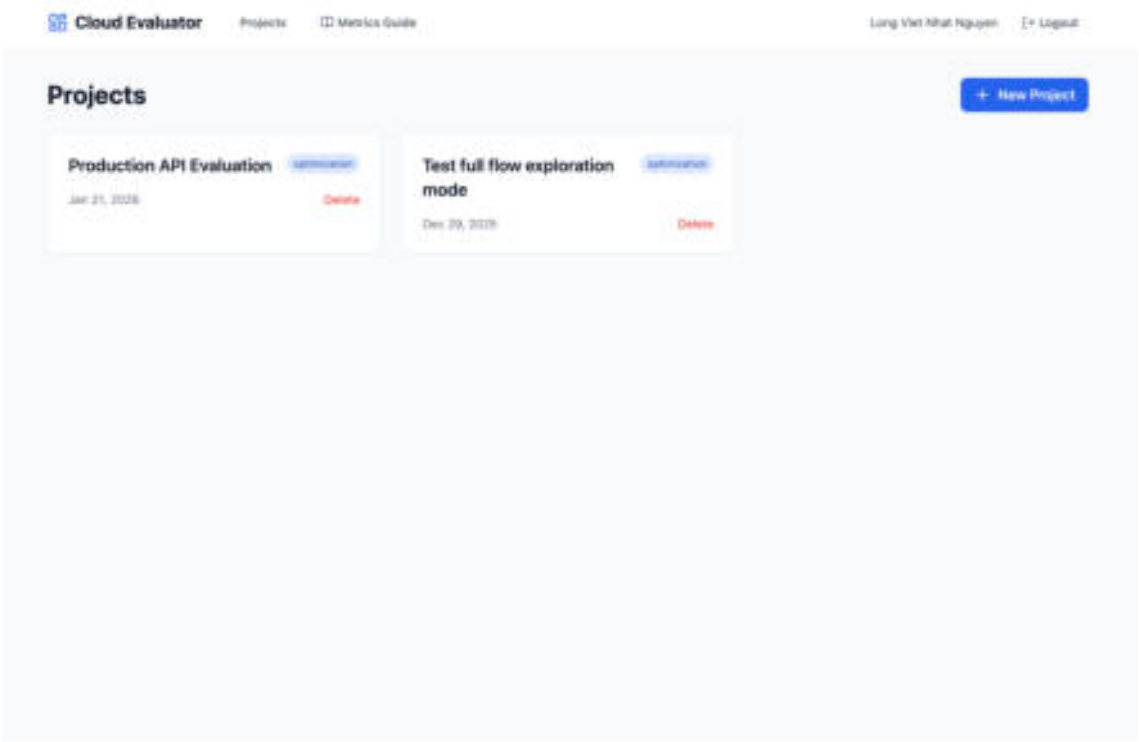
4.1. Giao diện người dùng



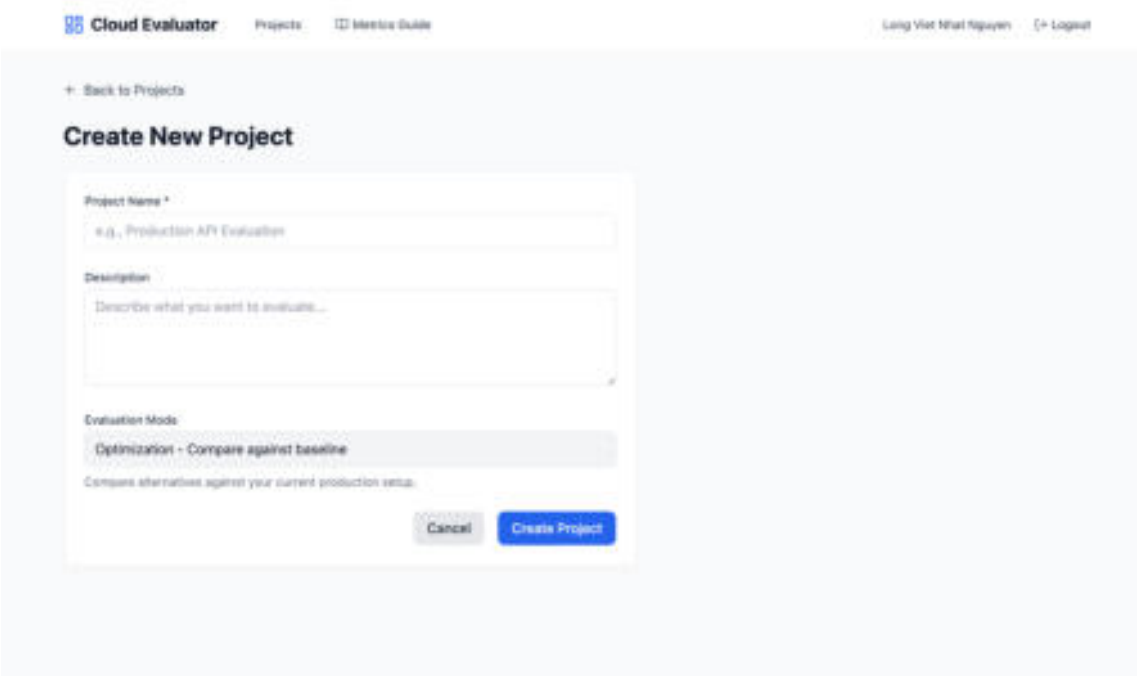
Hình 4.1: Màn hình đăng nhập



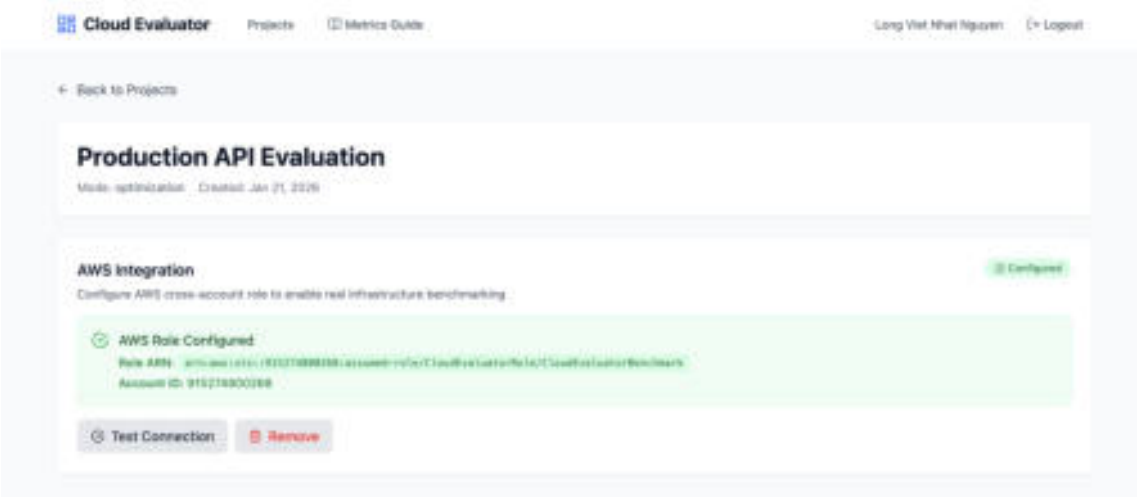
Hình 4.2: Màn hình đăng ký



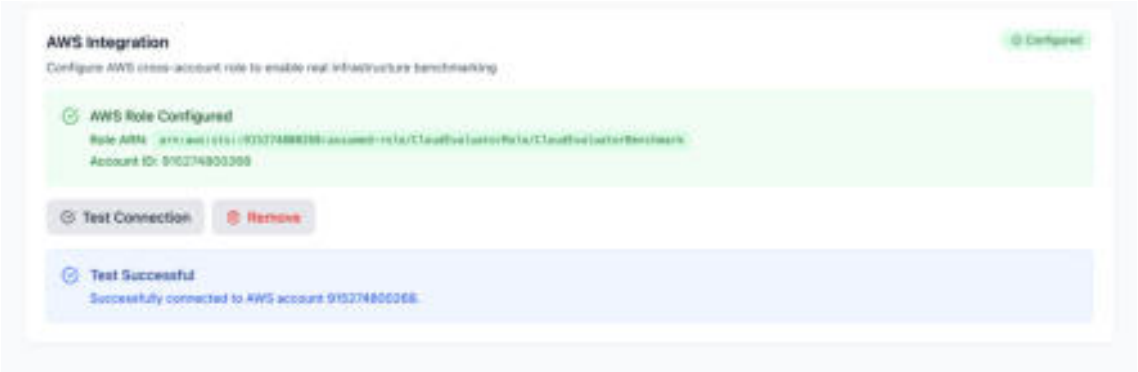
Hình 4.3: Trang danh sách dự án (Projects)



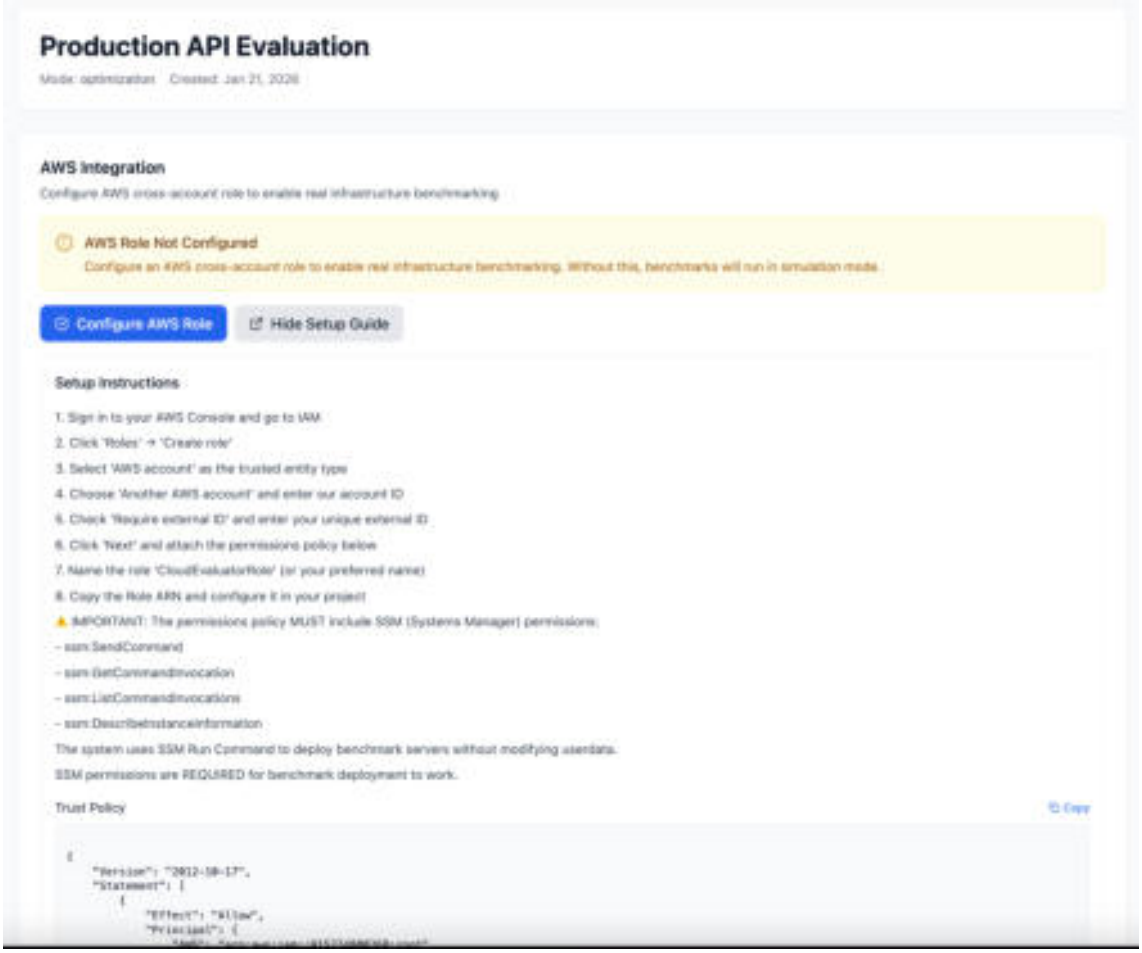
Hình 4.4: Trang tạo dự án mới



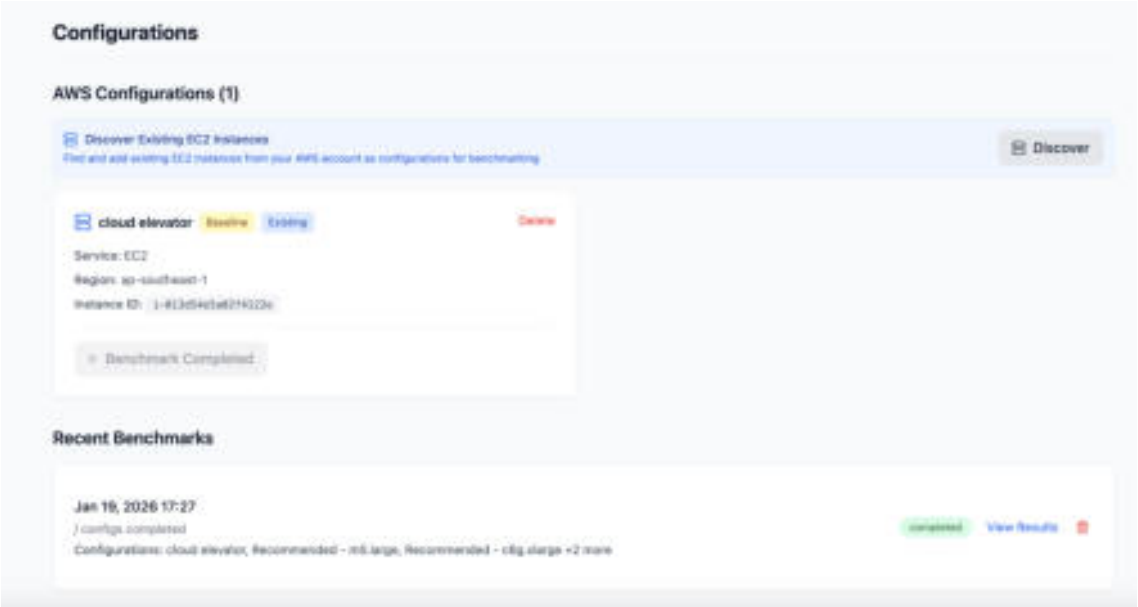
Hình 4.5: Trang chi tiết dự án – AWS Integration



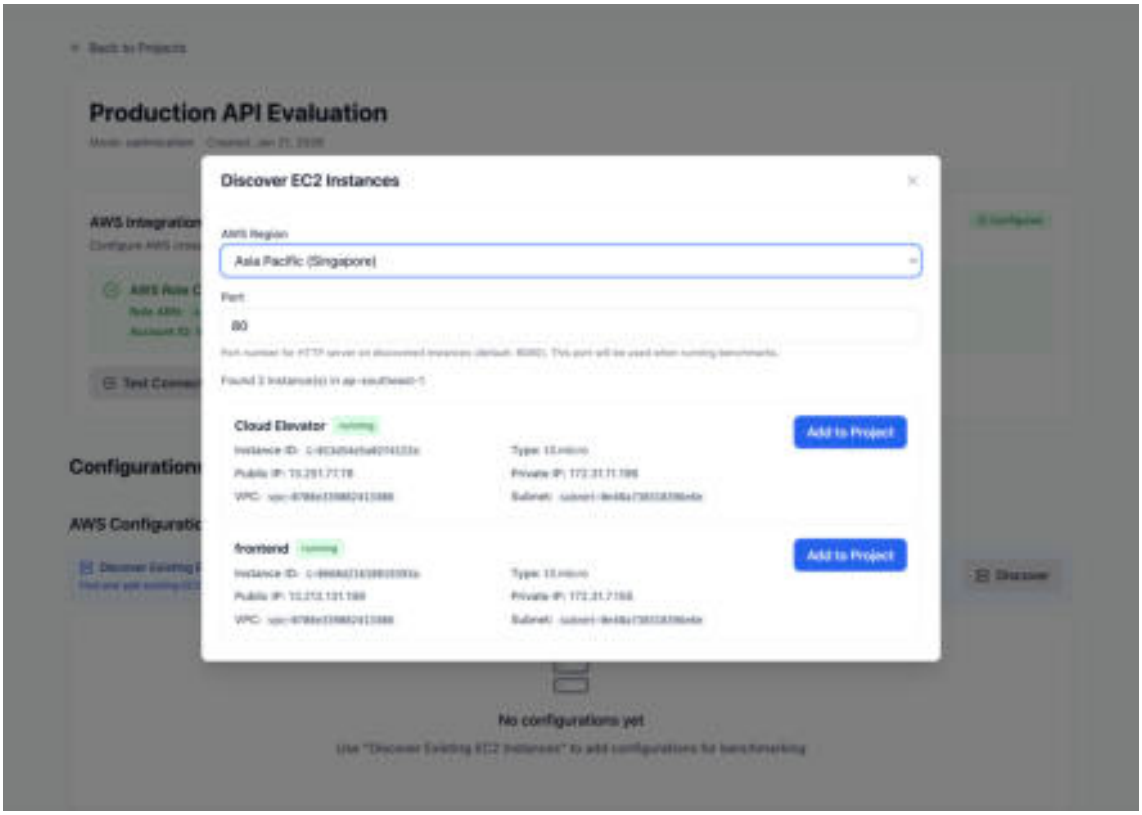
Hình 4.6: Trang chi tiết dự án – AWS Integration



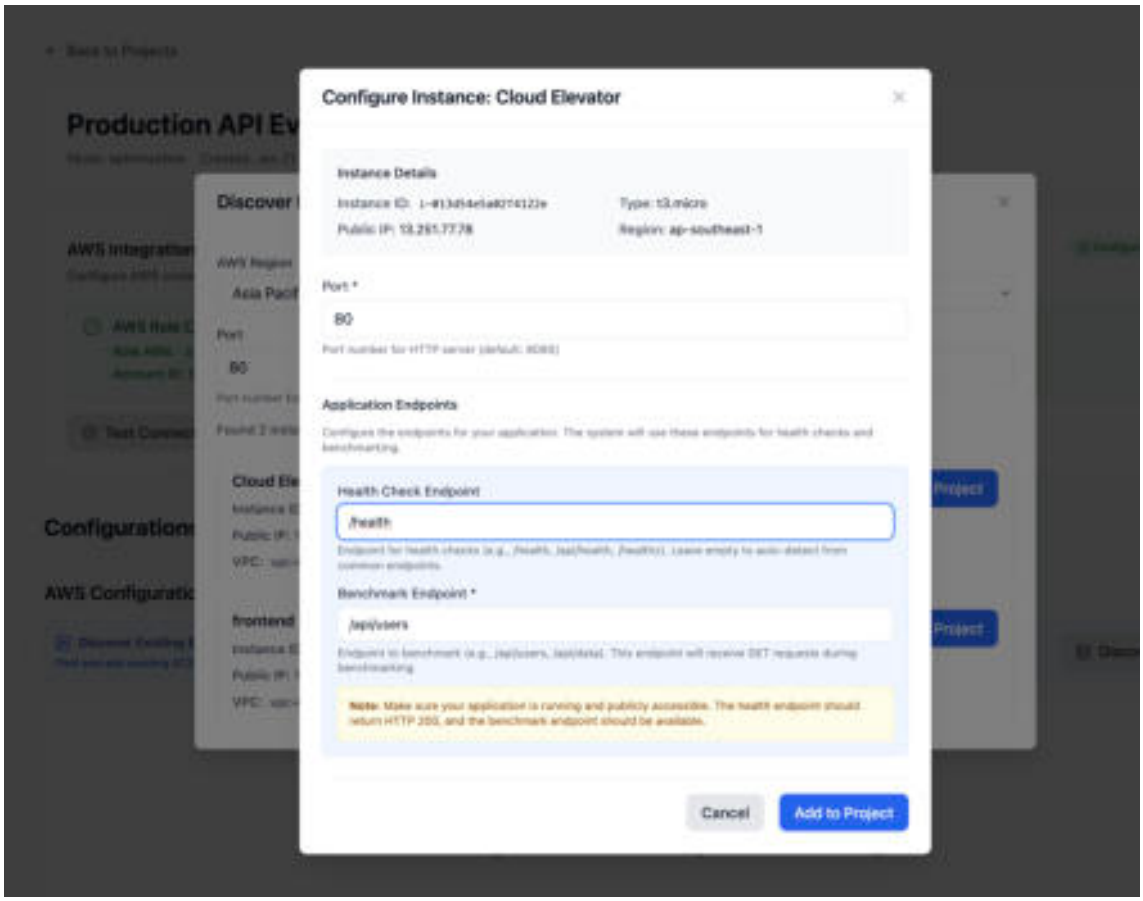
Hình 4.7: Guideline cấu hình IAM Role



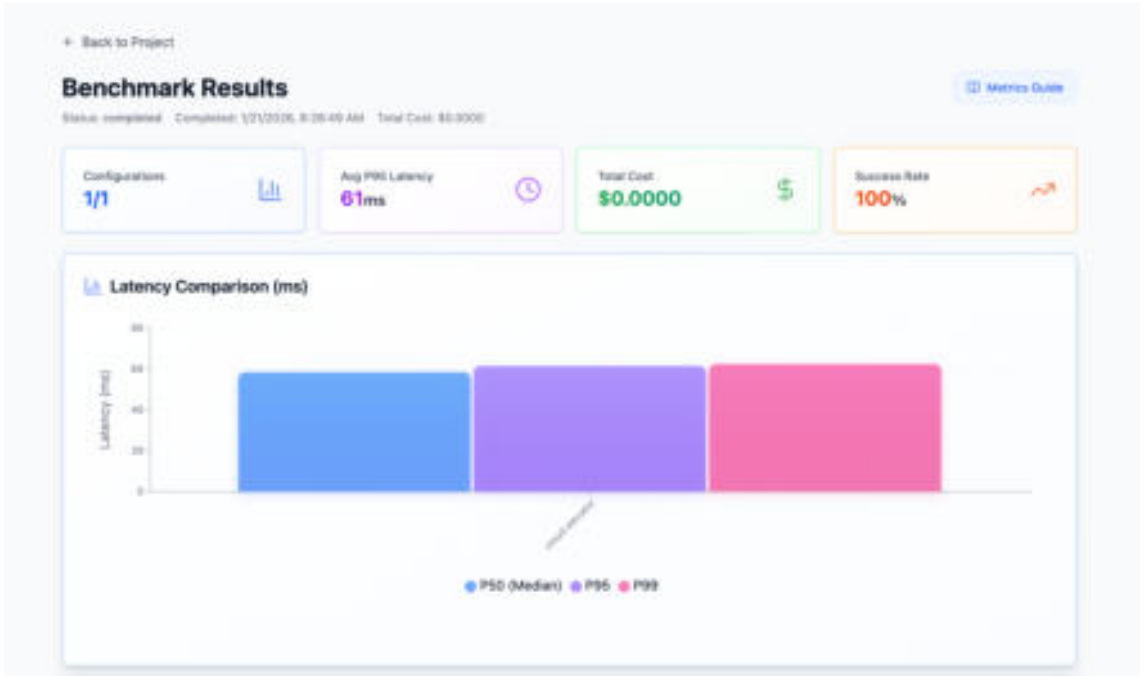
Hình 4.8: Trang chi tiết dự án - Tab Configurations



Hình 4.9: Modal thêm cấu hình EC2

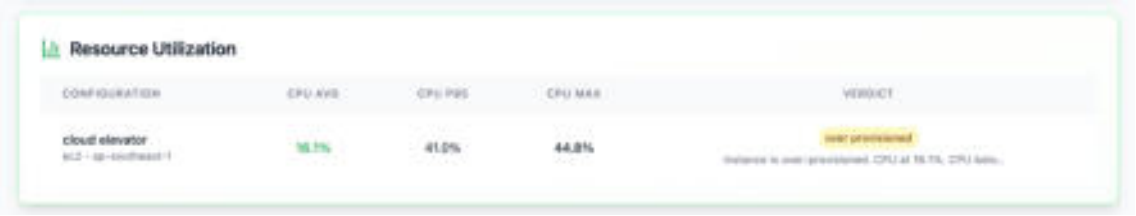


Hình 4.10: Modal thêm cấu hình EC2

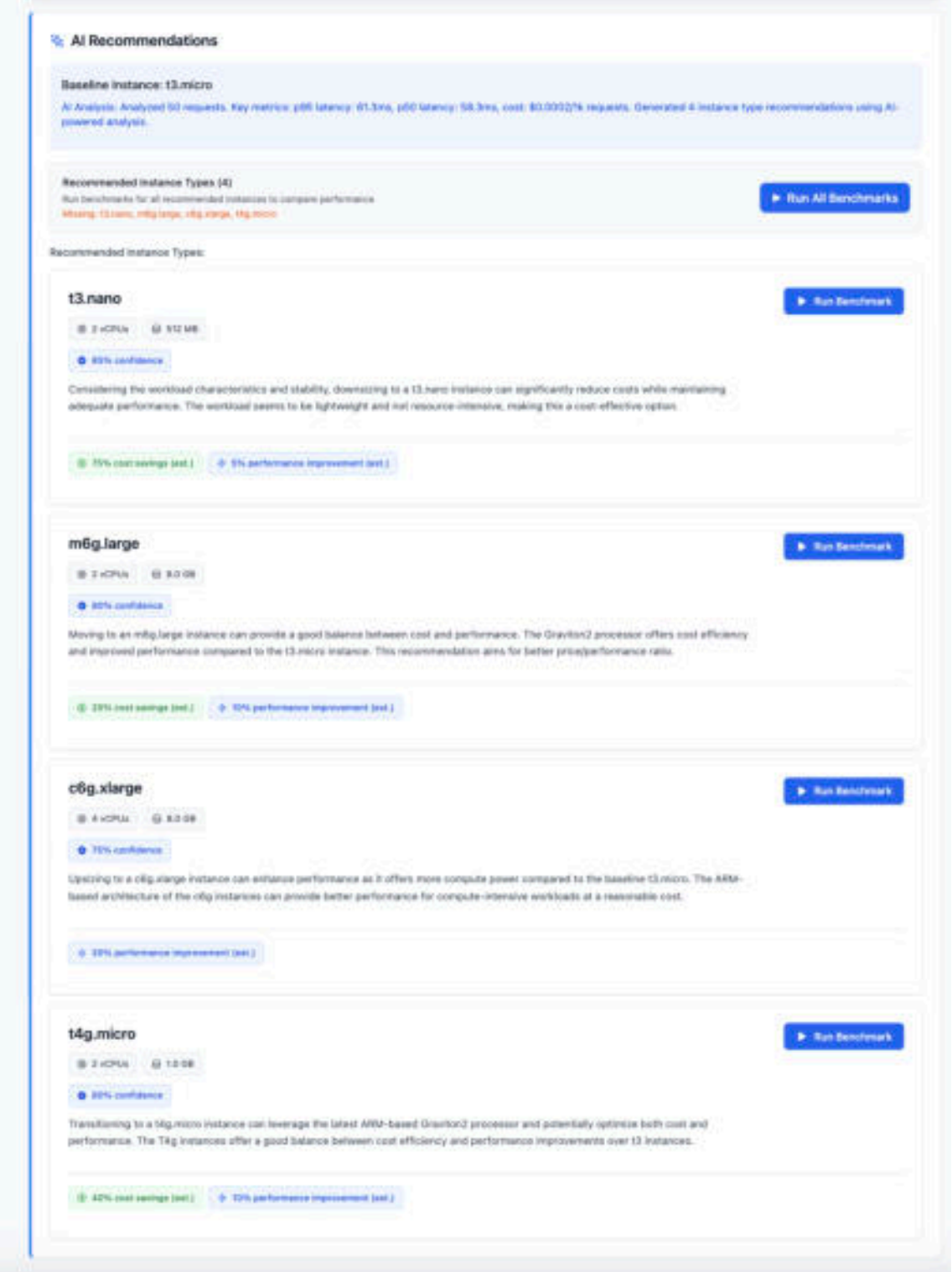


Hình 4.11: Trang kết quả benchmark - Benchmark Result cho baseline instance

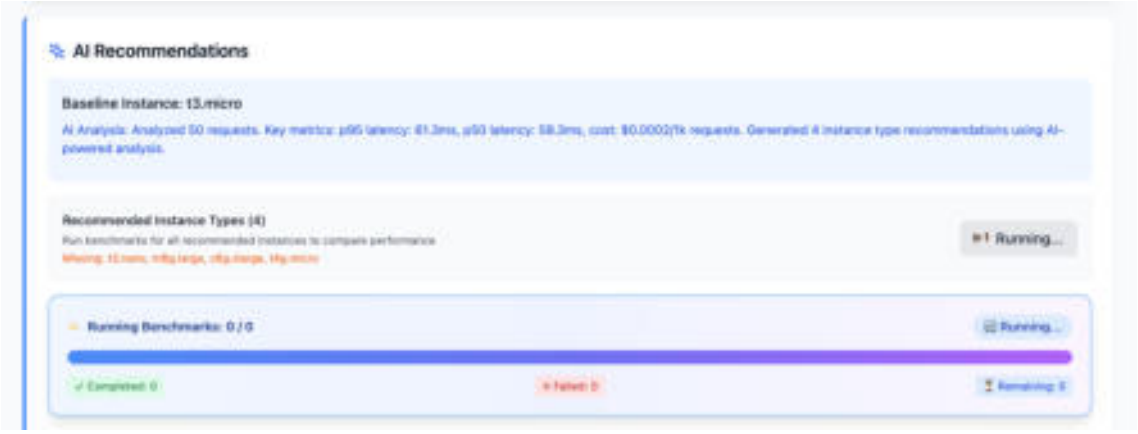
Hình 4.12: Trang kết quả benchmark - CloudWatch Metrics cho baseline instance



Hình 4.13: Trang kết quả benchmark - Resource Utilization cho baseline instance

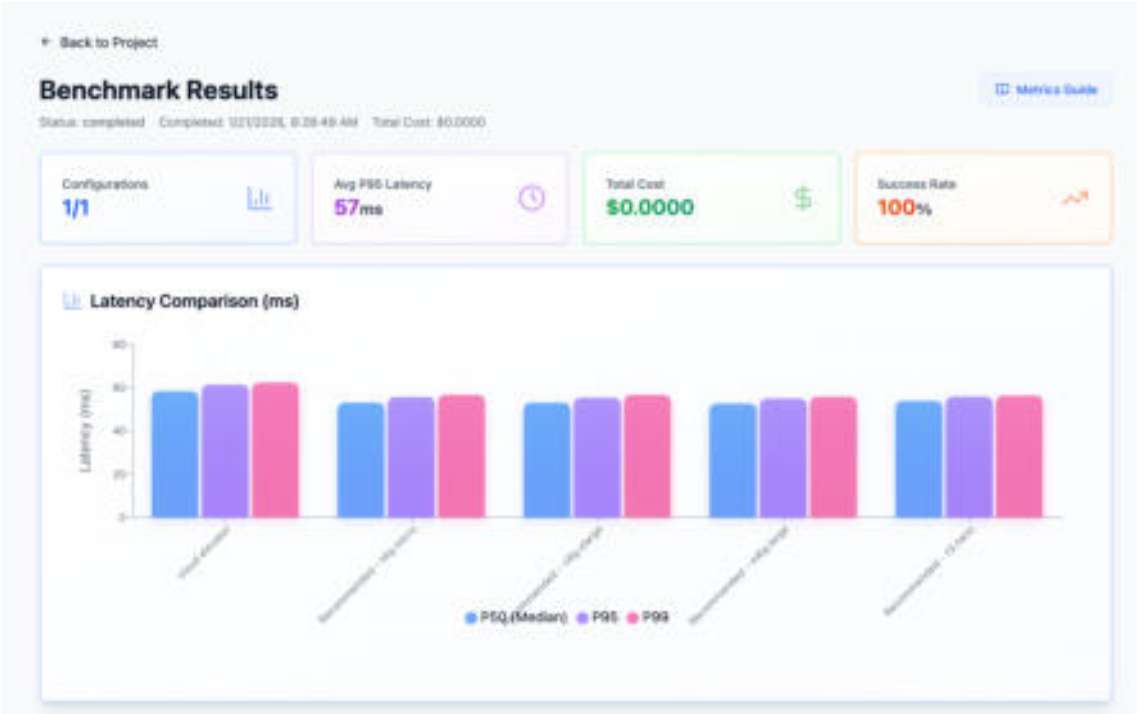


Hình 4.14 : Trang kết quả benchmark - AI Recommendation



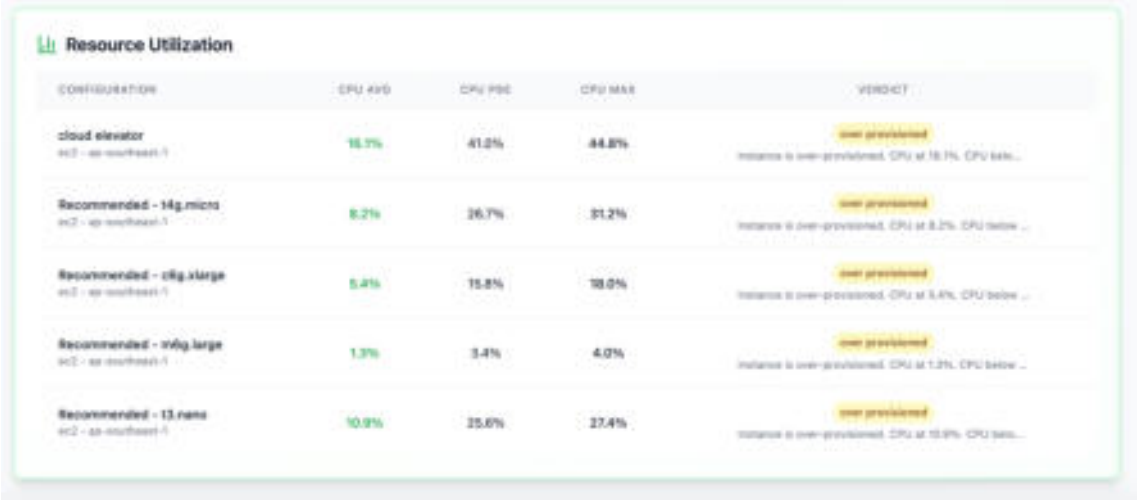
Hình 4.15: Trang kết quả benchmark - Run benchmark cho tất cả AI recommendations instances

Hình 4.16: AWS Console - EC2 Instance

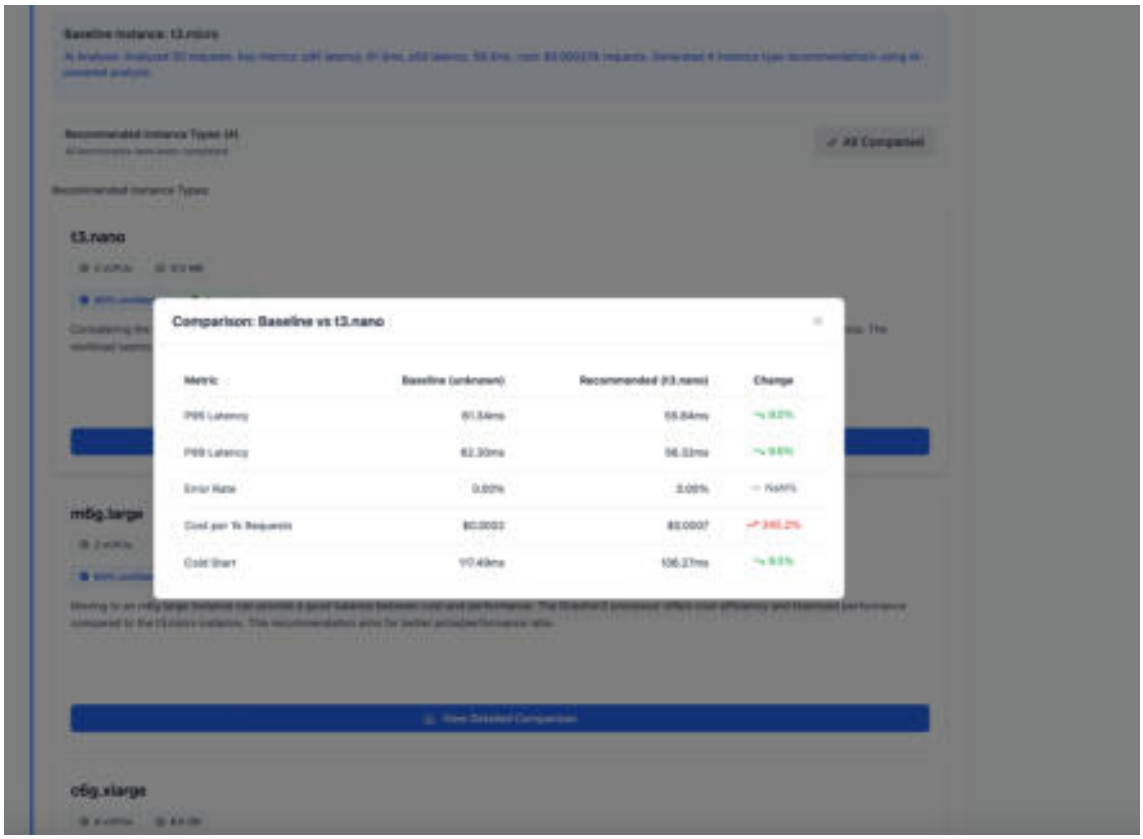


Hình 4.17: Trang kết quả benchmark - Biểu đồ so sánh Latency

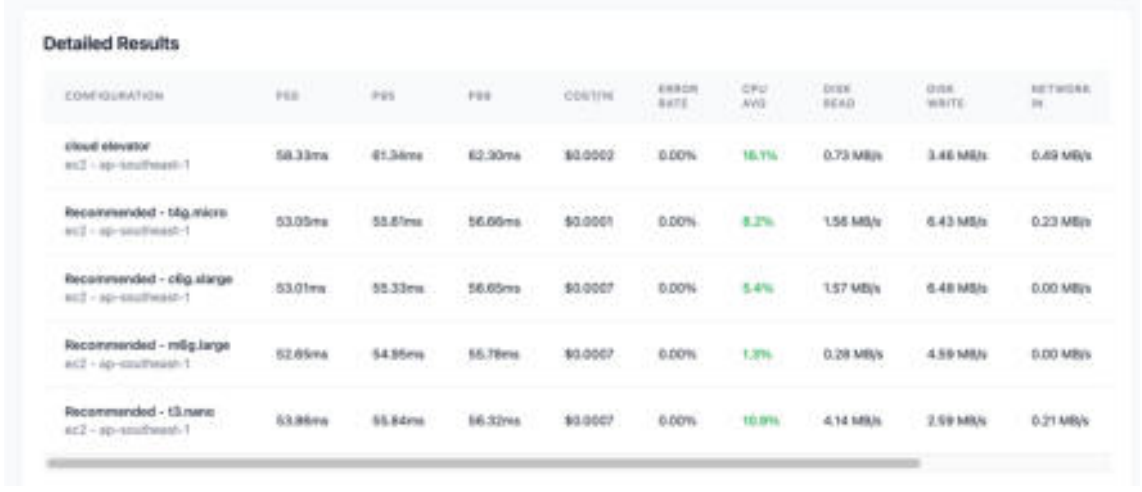
Hình 4.18: Trang kết quả benchmark - Biểu đồ so sánh Metrics



Hình 4.19: Trang kết quả benchmark - Biểu đồ so sánh Resource

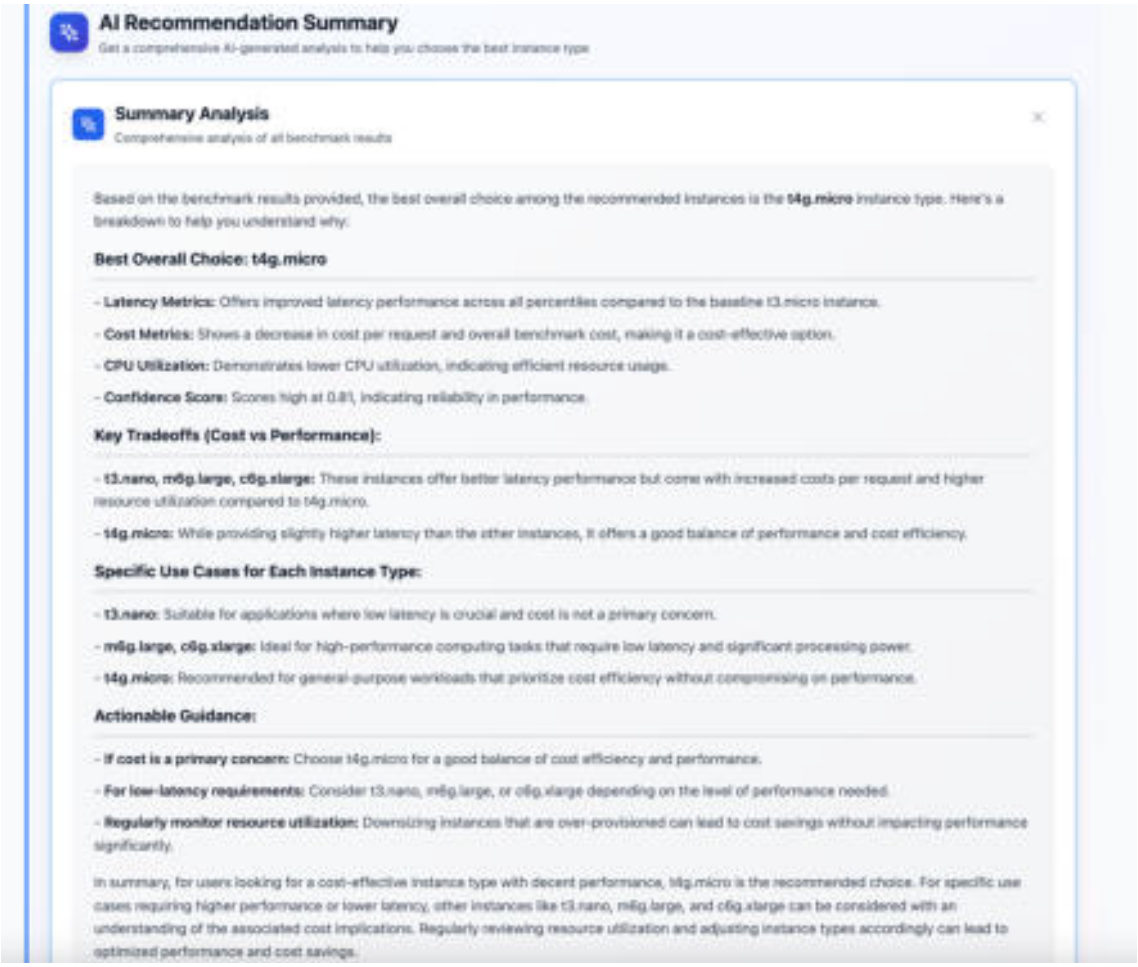


Hình 4.20: Trang kết quả benchmark – Bảng so sánh chi tiết giữa 2 instance



Hình 4.21: Trang kết quả benchmark - Bảng so sánh chi tiết

Hình 4.22: Trang kết quả benchmark - AI Recommendation Summary



Hình 4.22: Trang kết quả benchmark - Kết quả phân tích từ AI

4.2. Xây dựng Backend

Backend của hệ thống được xây dựng bằng **FastAPI** - một web framework hiện đại cho Python với hỗ trợ async/await, tự động generate OpenAPI documentation, và hiệu năng cao. Hệ thống sử dụng **Python 3.11** làm ngôn ngữ lập trình chính.

4.2.1 Web Framework và API

- **FastAPI**: Framework chính để xây dựng RESTful API, hỗ trợ async operations, tự động validation với Pydantic, và generate interactive API documentation
- **Pydantic**: Data validation và settings management, đảm bảo type safety cho request/response
- **Uvicorn**: ASGI server để chạy FastAPI application

4.2.2 Database và ORM

- **PostgreSQL**: Relational database chính để lưu trữ users, projects, configurations, benchmarks, và results

- **SQLAlchemy**: ORM (Object-Relational Mapping) với async support để tương tác với database
- **Alembic**: Database migration tool để quản lý schema changes

4.2.3 Caching và Message Queue

- **Redis**: In-memory data store được sử dụng cho hai mục đích:
  - Caching layer: Lưu trữ benchmark results để tái sử dụng, giảm chi phí và thời gian benchmark
  - Message broker: Làm broker cho Celery để xử lý background tasks

4.2.4 Background Task Processing

- **Celery**: Distributed task queue để xử lý các tác vụ nền như benchmark execution, đảm bảo không block API requests

4.2.5 AWS Integration

- **boto3**: AWS SDK cho Python để tương tác với các AWS services:
  - EC2: Tạo, quản lý, và terminate instances
  - CloudWatch: Thu thập metrics (CPU, network, disk I/O)
  - IAM: Assume roles để cross-account access

4.2.6 Authentication và Security

- **JWT (JSON Web Tokens)**: Cho authentication, với access token và refresh token
- **bcrypt**: Password hashing để bảo mật mật khẩu người dùng

4.2.7. AI/ML Integration

- **OpenAI API**: Tích hợp với GPT-3.5/GPT-4 để tạo AI-based recommendations
- **Statistical libraries**: Sử dụng numpy, scipy cho các tính toán thống kê trong anomaly detection

Backend được thiết kế theo kiến trúc layered, tách biệt rõ ràng giữa API layer, business logic layer (services), và data access layer (models). Tất cả các endpoints đều có validation, error handling, và logging đầy đủ.

4.3. Tích hợp AI và Khuyến nghị

Hệ thống tích hợp AI để phân tích kết quả benchmark và đưa ra khuyến nghị tối ưu về cấu hình EC2 instance types. Hệ thống sử dụng hai phương pháp recommendation: LLM-based và heuristic-based, với khả năng fallback tự động.

4.3.1. AI Recommendation Service

4.3.1.1. LLM-based Recommendations

Hệ thống sử dụng **OpenAI GPT-3.5** hoặc **GPT-4** để phân tích benchmark results và đưa ra recommendations. Quá trình hoạt động như sau:

1. **Thu thập dữ liệu**: Hệ thống tổng hợp các metrics từ benchmark results bao gồm:
    - Latency metrics (P50, P95, P99, mean)
    - Cost metrics (cost per request, cost per 1k requests)
    - CPU utilization (average, P95, max)
    - Error rate
    - Resource utilization verdict
2. **Prompt Engineering**: Hệ thống xây dựng prompt chi tiết mô tả:
    - Workload characteristics từ baseline benchmark
    - Current instance type và performance metrics
    - Yêu cầu LLM đề xuất instance types phù hợp hơn
    - Yêu cầu reasoning và explanation cho mỗi recommendation
    - Constraints về instance types có sẵn
3. **API Call**: Gọi OpenAI API với prompt đã chuẩn bị, sử dụng temperature phù hợp để đảm bảo output nhất quán.
4. **Response Parsing và Validation**:
    - Parse JSON response từ LLM
    - Validate instance types với danh sách instance types hợp lệ từ AWS
    - Extract reasoning và confidence scores
    - Format thành structure chuẩn cho frontend
5. **Error Handling**: Nếu LLM call thất bại hoặc response không hợp lệ, hệ thống tự động fallback sang heuristic method.

4.3.1.2. Heuristic-based Fallback

Khi không có AI hoặc LLM call thất bại, hệ thống sử dụng heuristic algorithm để tính điểm và ranking:

Scoring Algorithm:

- **Latency Score** (weight: 0.4): Tính điểm dựa trên P95 latency, lower is better
- **Cost Score** (weight: 0.3): Tính điểm dựa trên cost per request, lower is better
- **CPU Utilization Score** (weight: 0.2): Tính điểm dựa trên mức độ tối ưu của CPU utilization (optimal range: 20-80%)
- **Error Rate Score** (weight: 0.1): Tính điểm dựa trên error rate, lower is better

Ranking: Sort tất cả configurations theo tổng điểm (descending), instance types tốt nhất sẽ được recommend đầu tiên.

Confidence Scoring: Confidence được tính dựa trên:

- Số lượng samples (nhiều requests = confidence cao hơn)
- Variance (low variance = confidence cao hơn)

- Anomaly count (ít anomalies = confidence cao hơn)

4.3.2. Anomaly Detection

Hệ thống sử dụng các phương pháp thống kê để phát hiện anomalies trong benchmark results, đảm bảo chỉ sử dụng dữ liệu đáng tin cậy cho recommendations.

4.3.2.1 IQR Method (Interquartile Range)

Phương pháp này sử dụng IQR để phát hiện outliers trong latency measurements:

1. Tính Q1 (25th percentile) và Q3 (75th percentile) của latencies
2. Tính IQR = Q3 - Q1
3. Xác định bounds:
    - Lower bound = Q1 - 1.5 × IQR
    - Upper bound = Q3 + 1.5 × IQR
4. Các giá trị nằm ngoài bounds được đánh dấu là outliers

Outliers có thể do network issues, temporary resource contention, hoặc các yếu tố bên ngoài. Hệ thống có thể exclude outliers khi tính metrics hoặc đánh dấu results có nhiều outliers để giảm confidence.

4.3.2.2 Variance Analysis

Hệ thống tính **coefficient of variation (CV)** để đánh giá độ nhất quán của performance:

- CV = standard_deviation / mean_latency
- **CV < 0.2**: Stable performance, đáng tin cậy
- **CV 0.2-0.3**: Moderate variance, có thể chấp nhận
- **CV > 0.3**: High variance, unreliable, nên re-run benchmark

High variance thường do:

- Network instability
- Resource contention
- Cold starts (cho serverless)
- Inconsistent workload

4.3.2.3 Cold Start Detection

Đối với các dịch vụ có thể có cold start (như Lambda, App Runner với min_instances=0), hệ thống phát hiện cold start bằng cách:

1. So sánh latency của request đầu tiên với average latency của các requests sau
2. Nếu first_request_ms > warm_avg_ms × 2.0 → Cold start detected
3. Tính cold start penalty = first_request_ms - warm_avg_ms

Cold start information được lưu trong results và có thể ảnh hưởng đến recommendations, đặc biệt là cho production workloads cần consistent performance.

4.3.3. Recommendation Flow

Quy trình tạo recommendations diễn ra như sau:

1. **Trigger**: User click "Get Recommendations" sau khi có baseline benchmark results
2. **Data Preparation**:
   - Load baseline benchmark results
   - Extract metrics và workload characteristics
   - Check cache cho recommendations đã có
3. **AI Analysis**:
   - Nếu có OpenAI API key và enabled → Gọi LLM Recommender
   - Nếu không → Sử dụng Heuristic Recommender
   - Phân tích anomalies và filter unreliable data
4. **Instance Type Selection**:
   - LLM hoặc heuristic algorithm đề xuất danh sách instance types
   - Validate instance types với AWS instance specs
   - Tính confidence scores cho mỗi recommendation
5. **Explanation Generation**:
   - LLM: Sử dụng reasoning từ LLM response
   - Heuristic: Generate explanation dựa trên scoring factors
   - Include trade-offs (cost vs performance)
6. **Storage**: Lưu recommendations vào database với status "pending"
7. **Response**: Trả về recommendations cho frontend để hiển thị

User có thể approve hoặc reject recommendations. Khi approve, hệ thống sẽ tự động deploy recommended instances và chạy comparison benchmark để so sánh với baseline.

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. Tổng kết kết quả đạt được

5.1.1. Kết quả về mặt nghiên cứu lý thuyết

Trong quá trình thực hiện đề tài, đã nghiên cứu và áp dụng các kiến thức về:

Cloud Computing và AWS Services: Nghiên cứu sâu về EC2, CloudWatch, IAM roles, và cách thức triển khai tài nguyên trên AWS. Hiểu rõ về cross-account access và ephemeral resource management.

Benchmarking và Performance Measurement: Nghiên cứu các phương pháp đo lường hiệu năng thực tế, các metrics quan trọng (latency percentiles, cold start, error rate), và cách thu thập metrics từ CloudWatch.

AI/ML cho Recommendations: Nghiên cứu cách tích hợp LLM (OpenAI GPT) vào hệ thống recommendation, prompt engineering, và heuristic algorithms như scoring và ranking.

Anomaly Detection: Nghiên cứu các phương pháp thống kê như IQR method, variance analysis (coefficient of variation), và cold start detection để đảm bảo chất lượng dữ liệu.

Cost Optimization: Nghiên cứu các chiến lược tối ưu chi phí trong cloud computing, bao gồm caching, budget control, và resource cleanup.

5.1.2. Kết quả về mặt xây dựng ứng dụng thực tế

Đã xây dựng thành công hệ thống hoàn chỉnh với các thành phần sau:

Backend System: Hệ thống backend được xây dựng bằng FastAPI với đầy đủ các tính năng: authentication (JWT), quản lý projects và configurations, thực thi benchmark, và tích hợp AI recommendations. Hệ thống sử dụng PostgreSQL cho database, Redis cho caching, và Celery cho background task processing.

Frontend System: Giao diện người dùng được xây dựng bằng React và TypeScript, cung cấp trải nghiệm người dùng tốt với các màn hình quản lý project, cấu hình AWS, xem kết quả benchmark với biểu đồ trực quan, và hiển thị AI recommendations.

Benchmark Execution: Hệ thống có khả năng tự động triển khai EC2 instances trên AWS account của khách hàng, thực thi benchmark với sequential requests hoặc k6 load testing, thu thập metrics từ CloudWatch, và tự động cleanup resources sau khi hoàn thành.

AI Recommendation System: Tích hợp thành công OpenAI GPT để phân tích benchmark results và đưa ra recommendations, kèm theo heuristic fallback để

đảm bảo hệ thống luôn hoạt động. Hệ thống có khả năng phát hiện anomalies và đánh giá độ tin cậy của dữ liệu.

Cost Control: Hệ thống có cơ chế kiểm soát chi phí thông qua budget limits, caching results để tái sử dụng, và tự động cleanup để tránh chi phí phát sinh.

5.2. Những hạn chế còn tồn tại

5.2.1. Hạn chế về mặt tính năng và nghiệp vụ

Phạm vi dịch vụ: Hệ thống hiện tại chỉ hỗ trợ AWS EC2, chưa mở rộng sang các dịch vụ khác như Lambda, Fargate, App Runner như đã đề xuất ban đầu. Chưa hỗ trợ multi-cloud (GCP, Azure).

Optimization Mode: Optimization mode hiện tại chỉ hỗ trợ EC2 instances, chưa mở rộng cho các dịch vụ serverless hoặc container services.

Real-time Monitoring: Hệ thống chỉ benchmark tại thời điểm, chưa có tính năng real-time monitoring và continuous benchmarking để theo dõi performance theo thời gian.

Auto-scaling Testing: Chưa có khả năng test auto-scaling capabilities của các cấu hình.

Mobile Application: Chưa có phiên bản mobile app, người dùng chỉ có thể sử dụng qua web browser.

5.2.2. Hạn chế về hiệu năng và kỹ thuật

Benchmark Execution Time: Với nhiều configurations, thời gian benchmark có thể kéo dài (10-30 phút), ảnh hưởng đến trải nghiệm người dùng.

LLM API Latency: Gọi OpenAI API có thể mất 5-15 giây, làm chậm quá trình tạo recommendations. Chi phí API cũng là một vấn đề khi scale lớn.

Concurrent Benchmark Runs: Hệ thống có giới hạn về số lượng benchmark runs đồng thời do resource constraints và AWS rate limits.

Cache Hit Rate: Cache hit rate phụ thuộc vào workload patterns của người dùng, có thể không đạt mục tiêu 50% trong một số trường hợp.

Database Performance: Khi số lượng projects và benchmarks tăng lên, database queries có thể cần optimization thêm, đặc biệt là các queries phức tạp với joins nhiều bảng.

5.3. Bài học kinh nghiệm

5.3.1. Kinh nghiệm trong quản lý và triển khai dự án

Quản lý Scope: Việc xác định rõ phạm vi MVP ngay từ đầu giúp tập trung vào các tính năng core và tránh scope creep. Việc ưu tiên EC2 trước các dịch vụ khác là quyết định đúng đắn.

Cost Management: Kiểm soát chi phí là yếu tố quan trọng khi làm việc với AWS. Việc implement budget limits và automatic cleanup ngay từ đầu giúp tránh chi phí phát sinh không mong muốn.

Testing trên AWS thực tế: Việc test trên AWS resources thực tế (không phải mô phỏng) giúp phát hiện nhiều vấn đề mà không thể phát hiện trong môi trường local, như network latency, IAM permissions, và CloudWatch metrics collection.

Error Handling: Xây dựng error handling và retry mechanisms ngay từ đầu là rất quan trọng khi làm việc với AWS APIs, vì các API calls có thể fail do network issues hoặc rate limits.

5.3.2. Kinh nghiệm làm việc với công nghệ mới (AI, FastAPI)

FastAPI: FastAPI là một framework rất mạnh với async support, automatic API documentation, và type validation. Việc sử dụng Pydantic cho validation giúp giảm đáng kể bugs và improve code quality.

AI Integration (OpenAI GPT):

- Prompt engineering là yếu tố quan trọng nhất để có được kết quả tốt từ LLM. Cần mô tả rõ ràng context, constraints, và expected output format.
- Luôn có fallback mechanism (heuristic) vì LLM API có thể fail hoặc quá tốn kém.
- Validate và sanitize LLM output vì LLM có thể trả về invalid data.

Async Programming: Việc sử dụng async/await trong FastAPI và httpx giúp cải thiện đáng kể performance khi cần xử lý nhiều I/O operations (database queries, HTTP requests, AWS API calls).

Celery cho Background Tasks: Sử dụng Celery cho benchmark execution là quyết định đúng đắn, giúp API không bị block và có thể scale workers độc lập.

5.4. Hướng phát triển

5.4.1. Kế hoạch nâng cấp và mở rộng tính năng

Multi-service Support: Mở rộng hỗ trợ các AWS services khác như Lambda, ECS Fargate, và App Runner để người dùng có thể so sánh giữa nhiều loại dịch vụ.

Multi-cloud Support: Thêm hỗ trợ cho GCP và Azure để người dùng có thể so sánh và tối ưu hóa trên nhiều cloud providers.

Real-time Monitoring: Phát triển tính năng continuous monitoring để theo dõi performance theo thời gian, không chỉ benchmark tại một thời điểm.

Advanced AI Models: Fine-tune custom models dựa trên dữ liệu benchmark thực tế để cải thiện độ chính xác của recommendations. Có thể sử dụng reinforcement learning để học từ user feedback.

Batch Recommendations: Cho phép user approve và deploy nhiều recommendations cùng lúc, với khả năng so sánh tất cả với baseline.

Export Reports: Thêm tính năng export kết quả benchmark và recommendations ra PDF hoặc CSV để người dùng có thể chia sẻ và lưu trữ.

Auto-scaling Testing: Phát triển tính năng test auto-scaling capabilities để đánh giá khả năng scale của các cấu hình dưới tải cao.

5.4.2. Định hướng phát triển phiên bản ứng dụng di động (Mobile App)

React Native hoặc Flutter: Phát triển mobile app sử dụng React Native (để tái sử dụng code từ web) hoặc Flutter (để có performance tốt hơn và native feel).

Core Features: Mobile app sẽ bao gồm các tính năng chính:

- Xem danh sách projects và kết quả benchmark
- Nhận push notifications khi benchmark hoàn thành
- Xem recommendations và approve/reject
- Xem biểu đồ và metrics (responsive charts)

Offline Mode: Cho phép xem cached data khi offline, với sync khi có kết nối mạng.

Mobile-optimized UI/UX: Thiết kế giao diện tối ưu cho màn hình nhỏ, với navigation dễ sử dụng và touch-friendly controls.

Push Notifications: Tích hợp push notifications để thông báo cho người dùng khi:

- Benchmark hoàn thành
- Có recommendations mới
- Có lỗi xảy ra trong quá trình benchmark

Kết luận

Đề tài "Xây dựng hệ thống AutoTuner benchmark và đề xuất tối ưu tài nguyên" đã đạt được các mục tiêu đề ra: xây dựng hệ thống hoàn chỉnh có khả năng benchmark tài nguyên AWS thực tế, tích hợp AI để đưa ra recommendations, và cung cấp giao diện người dùng thân thiện. Hệ thống đã chứng minh được giá trị thực tế trong việc giúp người dùng tối ưu hóa cấu hình cloud resources dựa trên dữ liệu đo lường thực tế thay vì phỏng đoán.

Mặc dù còn một số hạn chế về phạm vi và hiệu năng, nhưng hệ thống đã tạo nền tảng vững chắc cho việc mở rộng và phát triển trong tương lai. Với các hướng phát triển đã đề xuất, hệ thống có tiềm năng trở thành một công cụ mạnh mẽ và toàn diện cho việc tối ưu hóa cloud infrastructure.

TÀI LIỆU THAM KHẢO

[1] Amazon Web Services. (2024). *Amazon EC2 User Guide*. AWS Documentation. https://docs.aws.amazon.com/ec2/

[2] Amazon Web Services. (2024). *Amazon CloudWatch User Guide*. AWS Documentation. https://docs.aws.amazon.com/cloudwatch/

[3] Amazon Web Services. (2024). *IAM User Guide*. AWS Documentation. https://docs.aws.amazon.com/iam/

[4] FastAPI. (2024). *FastAPI Documentation*. https://fastapi.tiangolo.com/

[5] React. (2024). *React Documentation*. https://react.dev/

[6] OpenAI. (2024). *OpenAI API Reference*. https://platform.openai.com/docs/api-reference

[7] Tan, L., Wang, N., & Liu, N. (2013). *Cloud Computing: Principles and Paradigms*. John Wiley & Sons.

[8] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, 53(4), 50-58.

[9] Celery Project. (2024). *Celery - Distributed Task Queue*. https://docs.celeryproject.org/

[10] PostgreSQL Global Development Group. (2024). *PostgreSQL Documentation*. https://www.postgresql.org/docs/

[11] Redis Labs. (2024). *Redis Documentation*. https://redis.io/docs/

[12] SQLAlchemy. (2024). *SQLAlchemy Documentation*. https://docs.sqlalchemy.org/