HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



Nguyễn Tuấn Hiệp

TRIỂN KHAI HỆ THỐNG PHÂN TÍCH, HỖ TRỢ KHẮC PHỤC SỰ CỐ GIÁN ĐOẠN TRONG MÔI TRƯỜNG ĐIỆN TOÁN ĐÁM MÂY

ĐỀ ÁN TỐT NGHIỆP THẠC SĨ KỸ THUẬT

(*Theo định hướng ứng dụng*)

HÀ NỘI - NĂM 2025

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



Nguyễn Tuấn Hiệp

TRIỂN KHAI HỆ THỐNG PHÂN TÍCH, HỖ TRỢ KHẮC PHỤC SỰ CỐ GIÁN ĐOẠN TRONG MÔI TRƯỜNG ĐIỆN TOÁN ĐÁM MÂY

Chuyên ngành: Hệ thống thông tin

Mã số: 8.48.01.04 (Hệ thống thông tin)

ĐỀ ÁN TỐT NGHIỆP THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

NGƯỜI HƯỚNG DẪN KHOA HỌC: TS. QUẢN TRỌNG THÉ

HÀ NỘI - NĂM 2025

BỘ KHOA HỌC VÀ CÔNG NGHỆ
HỌC VIỆN CÔNG NGHỆ
BƯU CHÍNH VIỄN THÔNG

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

BÁO CÁO GIẢI TRÌNH
SỬA CHỮA, HOÀN THIỆN ĐỀ ÁN TỐT NGHIỆP THẠC SĨ

Họ và tên học viên: Nguyễn Tuấn Hiệp

Chuyên ngành: Hệ thống thông tin

Khóa: .2024 – 2026 (Đợt 1)

Tên đề tài: Triển khai hệ thống phân tích, hỗ trợ khắc phục sự cố gián đoạn trong môi trường điện toán đám mây

Người hướng dẫn khoa học: TS. Quản Trọng Thế

Ngày bảo vệ: 22/01/2026

Các nội dung học viên đã sửa chữa, bổ sung trong đề án tốt nghiệp theo ý kiến đóng góp của Hội đồng chấm đề án tốt nghiệp:

TT	Ý kiến hội đồng	Sửa chữa của học viên
1	Rà soát và làm rõ, chỉnh sửa về mô hình và các thành phần triển khai hệ thống. Sửa các lỗi chính tả, trình bày, đảm bảo văn phong học thuật đồng nhất.	Bổ sung cập nhật hình vẽ (Hình 2.1), chỉnh sửa lỗi chính tả, văn phong đảm bảo tính học thuật, đồng nhất trong toàn bộ đề án.
2	Rà soát và làm rõ, chỉnh sửa về tiêu chí, phương pháp và kết quả đánh giá, có kết quả thống kê.	Bổ sung Chương 3 – Mục 3.3. Tiêu chí đánh giá – Làm rõ định nghĩa và thang đo để có cơ sở đối chiếu. Bổ sung nội dung mục 3.4. Phương pháp thử nghiệm. Làm rõ kết quả thống kê chi phí (Hình 3.10)
3	Rà soát, bổ sung danh mục thuật ngữ.	Bổ sung chú giải thuật ngữ GKE, Prometheus, MTTR... trong bảng danh mục thuật ngữ trang "vi"
4	Giải thích rõ lý do chọn RAG thay vì tinh chỉnh (Fine-tuning) mô hình LLM cho dữ liệu hệ thống	Bổ sung giải thích lý do tại Chương 1 – Mục 1.3, phần

	đặc thù.	(4) "Tạo sinh tăng cường truy xuất"
5	Làm rõ cơ chế xác nhận (Human-in-the-loop) trước khi hệ thống thực thi lệnh.	Bổ sung làm rõ cơ chế xác nhận tại Chương 2 – Mục 2.2.1 "Đặc tả yêu cầu chức năng"
6	Làm rõ cơ chế đảm bảo an toàn cho các dữ liệu nhạy cảm khi gửi lên API bên ngoài như Gemini.	Bổ sung làm rõ cơ chế đảm bảo an toàn dựa trên mã tại Chương 2 – Mục 2.3.2.a "Tác tử tiếp nhận và chuẩn hoá cảnh báo - AICA"
7	Làm rõ cách hệ thống xử lý khi gặp tình trạng "bão cảnh báo" (Alert Storm) với hàng trăm nghìn cảnh báo đổ về cùng lúc.	Bổ sung làm rõ tại Chương 2 – Mục 2.2.2 "Đặc tả yêu cầu phi chức năng"

Hà Nội, ngày..... tháng .... năm 20...

Ký xác nhận của

CHỦ TỊCH HỘI ĐỒNG CHẤM ĐỀ ÁN	THƯ KÝ HỘI ĐỒNG	NGƯỜI HƯỚNG DẪN KHOA HỌC	HỌC VIÊN
GS.TS. Từ Minh Phương	TS. Đào Ngọc Phong	TS. Quản Trọng Thế	Nguyễn Tuấn Hiệp

LỜI CAM ĐOAN

Tôi xin khẳng định rằng toàn bộ nội dung trong đề án này được hình thành từ quá trình nghiên cứu và thực hiện của chính tôi. Công trình này chưa từng được nộp, sử dụng hoặc trình bày để xét nhận bất kỳ học vị nào tại các cơ sở đào tạo đại học hay sau đại học.

Mọi nguồn tài liệu được tôi sử dụng trong quá trình nghiên cứu đều đã được trích dẫn rõ ràng và tuân thủ đầy đủ các quy định về trích dẫn học thuật.

Các kết quả phân tích, thử nghiệm và dữ liệu được trình bày trong đề án là do tôi trực tiếp thực hiện dưới sự hướng dẫn của TS. Quản Trọng Thế. Những thông tin và số liệu này được thu thập từ quá trình triển khai thực tế và được xử lý độc lập nhằm bảo đảm tính khách quan.

Trong quá trình xây dựng đề án, tôi có sử dụng một số công cụ hỗ trợ dựa trên trí tuệ nhân tạo để tra cứu thông tin, rà soát ngôn ngữ và nâng cao chất lượng diễn đạt. Tuy nhiên, mọi nội dung đánh giá, phân tích và kết luận đều do tôi tự nghiên cứu và hoàn toàn chịu trách nhiệm.

Tôi xin hoàn toàn chịu trách nhiệm trước pháp luật và trước nhà trường về tính trung thực, nguyên bản và tính chính xác của toàn bộ đề án này.

Tác giả



Nguyễn Tuấn Hiệp

Hà Nội, ngày 29 tháng 11 năm 2025

LỜI CẢM ƠN

Tôi xin bày tỏ lòng biết ơn chân thành tới TS. Quản Trọng Thế, Khoa An toàn thông tin, người đã tận tình hướng dẫn và hỗ trợ tôi trong suốt quá trình thực hiện đề án. Những nhận xét chuyên môn, định hướng nghiên cứu cũng như sự theo sát trong từng giai đoạn - từ việc lựa chọn tên đề tài, triển khai các nội dung đến hoàn thiện bản thảo báo cáo - đều là nguồn động lực quan trọng giúp tôi hoàn thành công trình này.

Tôi cũng xin trân trọng gửi lời cảm ơn đến các thầy, cô giáo của Khoa Sau đại học. Những kiến thức, sự tận tâm và nỗ lực giảng dạy của quý thầy cô trong suốt thời gian tôi học tập đã tạo nền tảng vững chắc cho quá trình nghiên cứu và phát triển của bản thân.

Xin chân thành cảm ơn tất cả sự đồng hành và hỗ trợ quý báu đó.

Tác giả

Nguyễn Tuấn Hiệp

Hà Nội, ngày 29 tháng 11 năm 2025

MỤC LỤC

LỜI CAM ĐOAN i
LỜI CẢM ƠN ii
DANH MỤC CÁC THUẬT NGỮ, CHỮ VIẾT TẮT v
DANH SÁCH HÌNH VẼ vii
MỞ ĐẦU 1
CHƯƠNG 1 - CƠ SỞ LÝ LUẬN 7
1.1. Tổng quan về điện toán đám mây 7
1.2. Tổng quan về sự cố gián đoạn 9
1.2.1. Sự cố máy chủ ngừng hoạt động 10
1.2.2. Sự cố thiếu hụt tài nguyên hệ thống 12
1.2.3. Sự cố phát sinh từ lỗi trong chuỗi dịch vụ phụ thuộc 14
1.3. Tổng quan về các công nghệ 16
1.4. Các yếu tố ảnh hưởng tới kết quả của hệ thống 20
CHƯƠNG 2 - THIẾT KẾ HỆ THỐNG 21
2.1. Mô hình hoá bài toán nghiên cứu 21
2.2. Đặc tả hệ thống 22
2.2.1. Đặc tả yêu cầu chức năng 22
2.2.2. Đặc tả yêu cầu phi chức năng 23
2.3. Thiết kế hệ thống 24
2.3.1. Kiến trúc tổng quan 24
2.3.2. Thiết kế hệ thống đa tác tử 25
2.3.3. Luồng dữ liệu 29
2.3.4. Luồng tích hợp LLM và RAG 32
CHƯƠNG 3 - TRIỂN KHAI, THỬ NGHIỆM VÀ ĐÁNH GIÁ 36
3.1. Môi trường và điều kiện triển khai hệ thống 36
3.1.1. Môi trường thử nghiệm 36
3.1.2. Công cụ và công nghệ triển khai 38

3.1.3. Tập dữ liệu cảnh báo và tài liệu kỹ thuật mẫu 39
3.2. Kết quả triển khai 39
3.3. Tiêu chí đánh giá 42
3.4. Thử nghiệm hệ thống 44
3.4.1. Kịch bản 1 – Sự cố đơn giản: Pod CrashLoopBackOff 44
3.4.2. Kịch bản 2 – Sự cố trung bình: Resource Exhaustion 47
3.4.3. Kịch bản 3 – Sự cố phức tạp: Microservice Dependency Failure 51
3.5. Đánh giá, nhận xét 55
KẾT LUẬN VÀ KIẾN NGHỊ 58
DANH MỤC TÀI LIỆU THAM KHẢO 59

DANH MỤC CÁC THUẬT NGỮ, CHỮ VIẾT TẮT

Viết tắt	Tiếng Anh	Tiếng Việt
ADK	Agent Developer Kit	Bộ công cụ phát triển tác tử (Google ADK)
AICA	Alert Ingestion & Contextual Analysis Agent	Tác tử tiếp nhận & chuẩn hoá cảnh báo
AIOps	Artificial Intelligence for IT Operations	Trí tuệ nhân tạo cho vận hành CNTT
Alert	Alert	Tín hiệu cảnh báo của hệ thống
Anomaly Score	Anomaly Score	Điểm bất thường
ChatOps	Chat-based Operations	Vận hành qua nền tảng chat
ChromaDB	Chroma Vector Database	CSDL vector dùng cho RAG
CrashLoopBackOff	CrashLoopBackOff	Trạng thái lỗi container lặp lại
Embedding	Embedding Representation	Biểu diễn vector hoá
Embedding Chunk	Knowledge Chunk Embedding	Mảnh tri thức đã vector hoá
HTTP 5xx	HTTP 5xx Error Codes	Lỗi máy chủ 5xx
IaC	Infrastructure as Code	Hạ tầng như mã
JSON	JavaScript Object Notation	Định dạng dữ liệu JSON
K8s / Kubernetes	Kubernetes	Nền tảng điều phối container
KREA	Knowledge Retrieval & Enrichment Agent	Tác tử truy xuất & tăng cường tri thức
LLM	Large Language Model	Mô hình ngôn ngữ lớn
LLM Gateway	LLM Gateway	Cổng trung gian kết nối tác tử–LLM

Log	Log	Nhật ký hệ thống
Metric	Metric	Chỉ số giám sát
Microservice	Microservice Architecture	Kiến trúc dịch vụ vi mô
MTTR	Mean Time to Recovery	Thời gian khôi phục trung bình
Nearest-Neighbor Search	Nearest Neighbor Search	Tìm kiếm láng giềng gần nhất
Observability	Observability	Quan sát hệ thống
Public Cloud	Public Cloud	Đám mây công cộng
RAG	Retrieval-Augmented Generation	Tạo sinh tăng cường truy xuất
RCARA	Root Cause Analysis & Remediation Agent	Tác tử phân tích nguyên nhân gốc
Runbook	Runbook	Tài liệu xử lý sự cố
SLA	Service-Level Agreement	Cam kết mức dịch vụ
SLO	Service-Level Objective	Mục tiêu mức dịch vụ
SRE	Site Reliability Engineering	Kỹ thuật đảm bảo độ tin cậy hệ thống
Temporal Pattern	Temporal Pattern	Mẫu hành vi theo thời gian
Trace	Distributed Tracing	Theo dõi yêu cầu phân tán
Unified Schema	Unified Schema	Lược đồ dữ liệu thống nhất
Vector Similarity	Vector Similarity	Tương đồng vector
GKE	Google Kubernetes Engine	Dịch vụ Kubernetes của Google
Prometheus	Prometheus	Công cụ giám sát mã nguồn mở.

DANH SÁCH HÌNH VẼ

Hình 2.1 Kiến trúc tổng quan của hệ thống 24
Hình 2.2 Thiết kế các tác tử của hệ thống 26
Hình 2.3 Sơ đồ mô tả quá trình suy luận của AICA 27
Hình 2.4 Luồng tích hợp giữa hệ thống đa tác tử và RAG 35
Hình 3.1 Mô hình triển khai hệ thống thử nghiệm 37
Hình 3.2 Cụm Kubernetes trên khai trên GKE 40
Hình 3.3 Giao diện giám sát trạng thái trên Grafana 40
Hình 3.4 Giao diện Alertmanager với một số cảnh báo 41
Hình 3.5 Giao diện hệ thống phân tích và hỗ trợ xử lý sự cố 41
Hình 3.6 Giao diện cảnh cáo trên Telegram 42
Hình 3.7 Minh hoạ kịch bản sự cố 1 45
Hình 3.8 Minh hoạ kịch bản sự cố 2 48
Hình 3.9 Minh hoạ kịch bản sự cố 3 52
Hình 3.10 Mức độ tiêu thụ token của thử nghiệm 56

MỞ ĐẦU

1. Lý do chọn đề tài

Trong thập kỷ qua, điện toán đám mây đã trở thành hạ tầng mặc định của hầu hết dịch vụ số. Quy mô triển khai mở rộng nhanh, cấu trúc liên vùng, đa tầng dịch vụ khiến hệ thống ngày các khó quan sát toàn diện. Khi một mắt xích - dù chỉ là một nút mạng hay một dịch vụ vi mô gặp sự cố - có thể là mất kết nối mạng, quá tải tài nguyên, lỗi ứng dụng ngừng hoạt động - thì hiệu ứng dây chuyền có thể làm ngừng trệ cả hệ thống, kéo theo tổn thất kinh tế và uy tín nghiêm trọng.

Báo cáo của ITIC cho thấy 90% doanh nghiệp mất trên 300.000 USD mỗi giờ ngừng dịch vụ [1], còn Gartner ước tính mức trung bình 5.600 USD mỗi phút [2][3]. Năm 2024 – 2025 chứng kiến nhiều sự cố lớn: ChatGPT sập dịch vụ hơn bốn giờ liên tiếp [4][5] và Google Cloud làm gián đoạn 33 dịch vụ toàn cầu trong gần ba giờ [6][7].

Các sự kiện này đã bộc lộ hạn chế của quy trình khắc phục sự cố thủ công: Tốn thời gian khoanh vùng nguyên nhân, phụ thuộc vào chuyên gia, khó khăn trong việc đáp ứng các thoả thuận mức độ dịch vụ (Service Level Agreement - SLA) khắt khe. Vì vậy, nhu cầu cấp thiết là phát triển một hệ thống hỗ trợ phân tích, suy luận và đề xuất phương án khắc phục sự cố, rút ngắn thời gian trung bình để khôi phục (Mean Time to Recovery - MTTR) và đảm bảo tính liên tục của dịch vụ. Đó chính là lý do để tác giả thực hiện đề án nghiên cứu "*Triển khai hệ thống phân tích, hỗ trợ khắc phục sự cố gián đoạn trong môi trường điện toán đám mây*".

2. Tổng quan về vấn đề nghiên cứu

Hiện nay, các hệ thống giám sát, nhật ký và cảnh báo đã được áp dụng rộng rãi để hỗ trợ giám sát hoạt động của hệ thống, nhưng các giải pháp này chủ yếu cung cấp tín hiệu hoặc cảnh báo chứ chưa có khả năng phân tích nguyên nhân gốc rễ hoặc đề xuất hành động khôi phục một cách thông minh. Việc phân tích và xây dựng phương án khắc phục sự cố vẫn phụ thuộc nhiều vào kinh nghiệm của kỹ sư vận hành, dẫn đến MTTR kéo dài và ảnh hưởng đến các cam kết SLA của dịch vụ.

Trước thực trạng đó, đề án này đề xuất xây dựng và triển khai một hệ thống hỗ trợ phân tích và gợi ý phương án khắc phục sự cố gián đoạn trong môi trường đám mây.

Cụ thể, hệ thống đề xuất được xây dựng dựa trên mô hình kiến trúc đa tác tử (multi-Agent), trong đó mỗi tác tử đảm nhiệm một vai trò chuyên biệt như tổng hợp cảnh báo, phân tích cảnh báo, truy xuất tri thức, tổng hợp ngữ cảnh, đề xuất giải pháp.

Cốt lõi của hệ thống là việc tích hợp với mô hình ngôn ngữ lớn (Large Language Model - LLM) nhằm tăng cường khả năng hiểu ngữ nghĩa của cảnh báo và các thông số giám sát, từ đó hỗ trợ phân tích nguyên nhân sự cố theo cách gần giống với chuyên gia. Nguồn dữ liệu được sử dụng là các dữ liệu nhật ký (Log), chỉ số giám sát (Metric), các tin hiệu cảnh báo (Alert) được phát sinh trong quá trình vận hành thực tế. Các dự liệu này sẽ được tổng hợp, trích xuất các thông tin quan trọng sau đó gửi tới LLM để thực hiện quá trình phân tích.

Kỹ thuật tạo sinh dựa trên truy xuất tăng cường (Retrieval-Augmented Generation - RAG) sẽ được tích hợp để cho phép truy xuất tri thức từ các nguồn dữ liệu bán cấu trúc như tài liệu kỹ thuật nội bộ, các tài liệu hướng dẫn vận hành, khắc phục sự cố.

Hệ thống sẽ được thiết kế và triển khai để tích hợp với các hệ thống giám sát truyền thống, nhằm tận dụng các thông số giám sát sẵn có. Ngoài ra, các kênh giao tiếp cũng được tích hợp để cung cấp giao diện tương tác với kỹ sư vận hành, giúp hiển thị các kết quả phân tích, đề xuất được trực quan.

Tổng thể, đề án không chỉ tập trung vào khía cạnh lý thuyết mà còn đặt trọng tâm vào việc thiết kế, xây dựng và triển khai một hệ thống hoàn chỉnh, có khả năng vận hành thực tế trong môi trường đám mây. Kết quả đầu ra kỳ vọng của đề án là một hệ thống nguyên mẫu (Prototype) có thể tiếp nhận cảnh báo, phân tích sự cố theo thời gian thực, gửi báo cáo phân tích và các đề xuất phương án khắc phục tới người vận hành.

3. Mục tiêu nghiên cứu

Mục tiêu chính của đề án này là tận dụng các kỹ thuật AI để nâng cao đáng kể khả năng phân tích và giải quyết các sự cố gián đoạn trong môi trường điện toán đám mây. Cụ thể, nghiên cứu nhằm đạt được các mục tiêu sau:

- *Mục tiêu lý thuyết:* Xây dựng khung phương pháp luận kết hợp đa tác tử, LLM và RAG trong phân tích sự cố; xác lập mô hình tri thức và đánh giá hiệu quả suy luận so với phương pháp truyền thống.
- *Mục tiêu thực tiễn:* Triển khai nguyên mẫu hệ thống hỗ trợ phân tích và đề xuất giải pháp khắc phục sự cố cho các ứng dụng trên nền tảng đám mây có khả năng rút ngắn thời gian khoanh vùng sự cố, áp dụng thử nghiệm trên môi trường đám mây công cộng.

4. Đối tượng và phạm vi nghiên cứu

Đề án nghiên cứu này tập trung vào thiết kế, phát triển, triển khai một hệ thống thông minh để phân tính, đề xuất phương án khắc phục sự cố trong các giới hạn cụ thể nhằm đảm bảo tính khả thi và chiều sâu.

Đối tượng

Đối tượng chính của nghiên cứu này là các sự cố gián đoạn (Interruption Incidents) ảnh hưởng trực tiếp đến khả năng sẵn sàng, hiệu suất và chức năng của các ứng dụng được triển khai trong môi trường điện toán đám mây. Điều này bao gồm các vấn đề dẫn đến ngừng hoạt động hoặc suy giảm hiệu suất đáng kể.

Hệ thống sẽ đặc biệt nhắm mục tiêu vào các giai đoạn phân tích và giải quyết vòng đời quản lý sự cố.

Phạm vi

Đề án này sẽ tập trung vào các ứng dụng được xây dựng bằng kiến trúc Microservice và Container hoá được triển khai trong môi trường đám mây. Lựa chọn này được thúc đẩy bởi sự phức tạp vốn có và bản chất phân tán của các kiến trúc hiện đại này, vốn đặt ra thách thức đáng kể cho việc quản lý sự cố truyền thống.

Hệ thống sẽ được thiết kế mà thử nghiệm trong môi trường đám mây công cộng Public Cloud. Các công nghệ AI tiên tiến cũng sẽ được sử dụng để phân tích dữ liệu hoạt động như nhật ký, số liệu, hay các tài liệu nội bộ được viết để vận hành các hệ thống đặc thù.

5. Phương pháp nghiên cứu

Để đạt được mục tiêu nghiên cứu đã nêu, đề án này sẽ sử dụng một phương pháp luận đa diện kết hợp các phương pháp sau:

- *Nghiên cứu tổng quan tài liệu:* Tìm hiểu các tri thức về Điện toán đám mây, giám sát hệ thống (Observability), AIOps, các công nghệ LLM, multi-Agent, RAG, cũng như các giải pháp khắc phục sự cố hiện có và các tài liệu về các sự cố lớn để xây dựng cơ sở lý thuyết và yêu cầu hệ thống.
- *Phân tích và thiết kế hệ thống:* Xác định yêu cầu chức năng và phi chức năng của hệ thống. Thiết kế kiến trúc tổng thể theo mô hình multi-Agent, phát triển các modules hỗ trợ, tích hợp với các hệ thống giám sát và phát triển giao diện tương tác qua các kênh giao tiếp, tin nhắn.
- *Triển khai và thử nghiệm nguyên mẫu:* Phát triển nguyên mẫu hệ thống bằng cách sử dụng ngôn ngữ lập trình và các công cụ phù hợp. Triển khai các thành phần giám sát và dữ liệu mẫu. Thực nghiệm hệ thống với các kịch bản gián đoạn giả lập hoặc dữ liệu từ thực tế.
- *Đánh giá kết quả:* Đánh giá hiệu suất của hệ thống thông qua các chỉ số như MTTR, độ chính xác trong việc phân tích nguyên nhân, mức độ hoàn thiện giải pháp đề xuất so với phương pháp truyền thống. Phân tích và so sánh kết quả với các mục tiêu đặt ra để rút ra kết luận về tính hiệu quả và khả năng ứng dụng của giải pháp đề xuất.

6. Nội dung

CHƯƠNG 1: CƠ SỞ LÝ LUẬN

Trình bày lý do và vai trò của việc xây dựng cơ sở lý luận cho đề án, làm tiền đề cho các phân tích và thiết kế hệ thống trong các chương sau. Bao gồm các nội

dung: (i) Các khái niệm cơ bản liên quan đến điện toán đám mây, dịch vụ ứng dụng, và các yêu cầu đảm bảo độ tin cậy trong hệ thống phân tán; (ii) Tổng quan về sự cố trong các hệ thống hiện tại, điển hình là sự cố gián đoạn; (iii) Tổng quan lý thuyết và công nghệ liên quan đến giám sát hệ thống, tự động hoá vận hành, LLM, multi-Agent, RAG; (iv) Xác định các nhân tố ảnh hưởng đến hiệu quả phân tích và gợi ý khắc phục sự cố trong môi trường điện toán đám mây, bao gồm tính phức tạp của hệ thống, sự phân tán, độ trễ; (v) Giả thiết nghiên cứu: Việc kết hợp các kỹ thuật học sâu, truy xuất thông tin và kiến trúc ứng dụng đa tác tử có thể hỗ trợ rút ngắn thời gian khắc phục sự cố, tăng độ chính xác trong khoanh vùng nguyên nhân và từ đó cải thiện độ tin cậy, độ ổn định của hệ thống trong môi trường điện toán đám mây.

CHƯƠNG 2: THIẾT KẾ HỆ THỐNG

Trình bày định hướng và nguyên lý thiết kế hệ thống. Bao gồm các nội dung: (i) Tổng quan thực trạng vận hành các hệ thống đám mây hiện nay và những khó khăn khi xảy ra sự cố (cả về phát hiện, khoanh vùng và khắc phục); (ii) Phân tích số liệu, báo cáo và sự cố gián đoạn nổi bật như ChatGPT, Google Cloud, v.v., để chứng minh tính cấp thiết của đề án; (iii) Mô hình hoá bài toán nghiên cứu: Đầu vào (Alert, Metrics, Log), yêu cầu phân tích, đầu ra là các đề xuất, phương án khắc phục cho kỹ sư vận hành; (iv) Đánh giá các công cụ và phương pháp hiện có (Truyền thống hoặc bán tự động) cùng các điểm còn hạn chế; (v) Xác định rõ yêu cầu đặt ra đối với hệ thống cần nghiên cứu: Tính chính xác, thời gian phản hồi, khả năng tích hợp; (vi) Thiết kế kiến trúc tổng thể cho hệ thống phân tích và hỗ trợ khắc phục sự cố sử dụng LLM, kiến trúc multi-Agent và RAG; (vii) Phân tách vai trò các tác tử trong kiến trúc multi-Agent; (viii) Thiết kế luồng dữ liệu, giao tiếp giữa các hệ thống với các thành phần khác của hệ thống giám sát; (ix) Xây dựng sơ đồ, biểu đồ luồng thông tin, sơ đồ trình tự mô phỏng hoạt động hệ thống.

CHƯƠNG 3: TRIỂN KHAI, THỬ NGHIỆM VÀ ĐÁNH GIÁ

Mô tả quá trình triển khai hệ thống nguyên mẫu trên môi trường thử nghiệm, thực hiện thử nghiệm, phân tích và đánh giá kết quả. Bao gồm các nội dung: (i) Môi trường triển khai: Môi trường đám mây thử nghiệm, công cụ phát triển, tập dữ liệu

cảnh báo sử dụng để mô phỏng; (ii) Thiết lập các Agent và tích hợp LLM vào pipeline xử lý; (iii) Triển khai module RAG cho truy xuất tri thức từ kho tài liệu kỹ thuật mẫu; (iv) Triển khai cơ chế giao tiếp giữa các Agent; (v) Xây dựng giao diện tương tác cho kỹ sư thông qua các kênh giao tiếp; (vi) Mô phỏng các kịch bản sự cố trong tập mẫu và thực hiện thử nghiệm; (vii) Các phương pháp được sử dụng để đánh giá kết quả; (viii) Xây dựng bộ tiêu chí đánh giá: MTTR, độ chính xác khi phân tích nguyên nhân, độ phù hợp của đề xuất khắc phục; (ix) Dữ liệu thử nghiệm, kết quả thu được qua các tình huống giả lập; (x) Phân tích hiệu quả hệ thống dưới nhiều góc độ: Tính ứng dụng, khả năng mở rộng, độ ổn định, tính tự động; (xi) So sánh với các phương pháp thủ công hoặc các công cụ hiện có; (xii) Nhận xét các điểm mạnh, điểm yếu và những điều chỉnh cần thiết.

CHƯƠNG 1 - CƠ SỞ LÝ LUẬN

1.1. Tổng quan về điện toán đám mây

Trong bối cảnh chuyển đổi số diễn ra mạnh mẽ, điện toán đám mây (Cloud Computing) đã trở thành nền tảng hạ tầng cốt lõi cho hầu hết các hệ thống thông tin và dịch vụ số hiện đại. Điện toán đám mây được hiểu là mô hình cung cấp tài nguyên tính toán – bao gồm máy chủ, lưu trữ, cơ sở dữ liệu, mạng, phần mềm và các dịch vụ khác – thông qua mạng Internet, cho phép người dùng truy cập và sử dụng tài nguyên một cách linh hoạt, tự động và được tính phí dựa trên mức độ tiêu thụ thực tế.

Theo định nghĩa của Viện tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ (NIST – Nation Institute of Standards and Technology), điện toán đám mây được đặc trưng bởi năm thuộc tính cốt lõi: (i) Khả năng tự phục vụ theo nhu cầu (*On-demand self-service*); (ii) Khả năng truy cập rộng qua mang *(Broad network access)*; (iii) Gộp tài nguyên linh hoạt (*Resource pooling*); (iv) Khả năng mở rộng nhanh chóng *(Rapid elasticity)* và (v) Dịch vụ được đo lường, giám sát *(measured service)*. Những đặc tính này cho phép người dùng triển khai ứng dụng nhanh chóng mà không cần đầu tư lớn vào cơ sở hạ tầng vật lý, đồng thời nâng cao khả năng mở rộng và tính sẵn sàng của dịch vụ [8][9].

Về mô hình cung cấp, điện toán đám mây được chia thành ba lớp dịch vụ chính, bao gồm: Cơ sở hạ tầng như một dịch vụ (Infrastructure as a Service - IaaS), cung cấp tài nguyên ảo hoá như máy chủ, lưu trữ và mạng. Mức trung gian là nền tảng như một dịch vụ (Platform as a Service - PaaS), cho phép các nhà phát triển triển khai và quản lý ứng dụng mà không cần xử lý các vấn đề hạ tầng. Mức cao nhất là Phần mềm như một dịch vụ (Software as a Service - SaaS), cung cấp các ứng dụng hoàn chỉnh cho người dùng cuối thông qua Internet. Bên cạnh các mô hình dịch vụ, các tổ chức có thể lựa chọn mô hình triển khai khác nhau tuỳ theo nhu cầu bảo mật, chi phí và khả năng kiểm soát, bao gồm: Đám mây công cộng (Public Cloud), đám mây riêng (Private Cloud) và đám mây lai (Hybrid Cloud).

Sự phát triển của điện toán đám mây không chỉ tạo ra bước tiến lớn về mặt công nghệ mà còn làm thay đổi cách thức các tổ chức xây dựng mô hình kinh doanh và vận hành hạ tầng CNTT. Nhờ khả năng mở rộng linh hoạt, cách thức phân bổ tài nguyên theo nhu cầu và giảm đáng kể chi phí đầu tư ban đầu, các doanh nghiệp có thể triển khai và phát triển dịch vụ với tốc độ nhanh hơn, hiệu quả hơn. Tuy nhiên, chính sự phức tạp và mức độ phân tán ngày càng cao của các hệ thống đám mây lại đặt ra nhiều yêu cầu mới đối với việc đảm bảo độ tin cậy, khả năng quan sát và năng lực xử lý sự cố. Những thách thức này trở thành động lực thúc đẩy các hướng nghiên cứu về hệ thống giám sát, phân tích và hỗ trợ vận hành.

Xét về bản chất, nền tảng đám mây là một hệ thống phân tán quy mô lớn, nơi nhiều thành phần độc lập - từ máy chủ, container cho đến các dịch vụ ứng dụng - tương tác và phối hợp để duy trì một dịch vụ thống nhất ở tầng người dùng. Trong bối cảnh đó, độ tin cậy trở thành tiêu chí cốt lõi, phản ánh khả năng hệ thống duy trì hoạt động ổn định và liên tục ngay cả khi đối mặt với những điều kiện không chắc chắn, lỗi cục bộ hoặc các sự cố bất thường trong môi trường vận hành. Đây chính là nền tảng lý luận quan trọng để hướng tới các giải pháp phân tích và xử lý sự cố tự động trong những chương tiếp theo.

Theo mô hình Site Reliability Engineering (SRE) do Google phát triển, độ tin cậy của hệ thống được định lượng thông qua các chỉ số như:

- *Khả năng sẵn sàng (Availability)*: phần trăm thời gian hệ thống có thể phục vụ bình thường.

- *Độ trễ (Latency)*: thời gian phản hồi của dịch vụ đối với yêu cầu người dùng.

- *Tính bền vững (Durability)*: khả năng bảo toàn dữ liệu và trạng thái trong các tình huống lỗi.

- Khả năng mở rộng *(Scalability)*: năng lực mở rộng tài nguyên mà không làm giảm chất lượng dịch vụ.

Những yếu tố này thường được ràng buộc bởi Thỏa thuận mức độ dịch vụ *(Service Level Agreement – SLA)* giữa nhà cung cấp và khách hàng. Tuy nhiên, độ tin

cậy trong hệ thống phân tán không thể được đảm bảo tuyệt đối, do đặc tính không đồng nhất và bất định *(non-determinism)* của môi trường đám mây. Mạng truyền thông có thể gián đoạn, nút tính toán có thể ngừng hoạt động đột ngột, và các thành phần ảo hóa có thể bị quá tải tài nguyên. Vì vậy, thiết kế hệ thống cần hướng đến khả năng chịu lỗi *(Fault Tolerance)* và tự phục hồi *(Self-Healing)* - tức là hệ thống có thể phát hiện, cô lập và khôi phục lỗi một cách tự động mà không làm gián đoạn toàn bộ dịch vụ [10].

1.2. Tổng quan về sự cố gián đoạn

Trong các hệ thống điện toán đám mây, tính phân tán, khả năng mở rộng linh hoạt và mức độ phụ thuộc lẫn nhau giữa các thành phần dịch vụ ngày càng trở thành đặc tính cốt lõi của kiến trúc. Cách tiếp cận này mang lại lợi ích đáng kể về khả năng chịu tải, linh hoạt và tính sẵn sàng, nhưng đồng thời làm gia tăng đáng kể độ phức tạp của hoạt động vận hành. Bất kỳ sai lệch nhỏ nào ở một trong các lớp hạ tầng, nền tảng hoặc ứng dụng đều có khả năng lan truyền và gây ra các trạng thái bất ổn vượt ngoài dự đoán của hệ thống. Trong bối cảnh đó, sự cố gián đoạn được xem là nhóm sự cố nghiêm trọng nhất, do có khả năng làm ngừng trệ một phần hoặc toàn bộ chức năng của dịch vụ, ảnh hưởng trực tiếp đến người dùng và hoạt động kinh doanh.

Sự cố gián đoạn không phải là kết quả của một nguyên nhân đơn nhất, mà thường hình thành từ sự tương tác phức tạp giữa nhiều yếu tố kỹ thuật. Các báo cáo vận hành của Google SRE, AWS Well-Architected Framework và CNCF đều cho thấy rằng các sự cố gây gián đoạn có bản chất "đa tầng" (multi-layer) và "đa nhân tố" (multi-factor). Điều này hàm ý rằng lỗi có thể khởi phát tại tầng vật lý, nhưng chỉ biểu hiện rõ rệt khi tác động lên các tầng cao hơn như bộ điều phối container, cơ chế cân bằng tải hoặc chuỗi dịch vụ phụ thuộc. Đặc tính này làm tăng đáng kể độ khó trong việc phát hiện sớm và phân tích nguyên nhân gốc rễ, bởi hệ thống quan sát thường chỉ ghi nhận các triệu chứng cuối cùng, trong khi tầng gây ra lỗi nằm ở sâu hơn trong cấu trúc hệ thống.

Một đặc điểm quan trọng của sự cố gián đoạn là phạm vi ảnh hưởng rộng và mang tính dây chuyền. Khi một thành phần cốt lõi nhất mất khả năng đáp ứng, các

dịch vụ phía trên có thể ngay lập tức rơi vào trạng thái quá tải, lỗi logic hoặc sai lệch về luồng dữ liệu. Hiệu ứng lan truyền (Cascading failure) thường khiến sự cố ban đầu nhanh chóng mở rộng quy mô, dẫn đến việc dịch vụ ngừng đáp ứng. Hậu quả của sự cố không chỉ dừng lại ở khía cạnh kỹ thuật, mà còn tác động trực tiếp đến chỉ số kinh doanh, cam kết SLA, uy tín thương hiệu và chi phí vận hành khắc phục.

Mặc dù phạm vi về các loại sự cố thuộc nhóm sự cố gián đoạn rất đa dạng, tuy nhiên để đảm bảo chất lượng, đề án sẽ tập trung nghiên cứu vào ba loại sự cố, bao gồm: *(i) Sự cố máy chủ ngừng hoạt động; (ii) Sự cố thiếu hụt tài nguyên hệ thống; (iii) Sự cố phát sinh từ lỗi trong chuỗi dịch vụ phụ thuộc*. Đây là ba loại sự cố gián đoạn có tần suất xuất hiện cao trong các môi trường đám mây của các tổ chức và là nguyên nhân hàng đầu dẫn đến các đợt gián đoạn quy mô lớn.

1.2.1. Sự cố máy chủ ngừng hoạt động

Sự cố "Server Down" được xem là một trong những dạng gián đoạn cơ bản nhưng có tác động rộng nhất trong các hệ thống phân tán và đám mây. Khái niệm này mô tả tình huống trong đó một thực thể đảm nhiệm nhiệm vụ xử lý - bao gồm máy chủ vật lý, máy ảo, node trong cụm Kubernetes, container hoặc pod - đột ngột mất khả năng hoạt động, không thể duy trì tiến trình thực thi hoặc không còn khả năng xử lý các yêu cầu gửi tới. Mặc dù cách gọi khác nhau tùy theo nền tảng, như NodeDown trong Kubernetes, Instance Termination trong các hạ tầng ảo hóa hay PodDown trong hệ thống điều phối container, bản chất của sự cố đều là sự gián đoạn đột ngột của một thành phần tính toán cốt lõi, dẫn đến sự ngưng trệ ở tầng dịch vụ.

Sự cố máy chủ ngừng hoạt động thường biểu hiện dưới các dạng triệu chứng rõ rệt: thành phần không phản hồi yêu cầu, thời gian chờ tăng cao, tiến trình giám sát không thể thu thập dữ liệu, hoặc các tác vụ định kỳ không thể được thực thi. Tuy nhiên, mặc dù triệu chứng có thể dễ quan sát, việc xác định nguyên nhân gốc rễ lại không đơn giản, bởi trạng thái ngừng hoạt động có thể phát sinh từ nhiều yếu tố, trải dài từ tầng vật lý đến tầng ứng dụng. Một máy chủ có thể ngừng hoạt động do lỗi phần cứng, hư hỏng thiết bị lưu trữ, sự cố nguồn điện hoặc lỗi mạng nội bộ. Tại tầng hệ điều hành, lỗi kernel, xung đột phiên bản thư viện, lỗi hệ thống tệp hoặc sự cố

trong quá trình cập nhật có thể khiến tiến trình bị dừng đột ngột. Ở tầng ứng dụng hoặc runtime, hiện tượng crash do quản lý bộ nhớ kém, lỗi logic không được kiểm soát hoặc sử dụng tài nguyên vượt giới hạn cũng có thể dẫn đến việc thực thể triển khai không thể tiếp tục chạy. Các sự cố liên quan đến quy trình triển khai, chẳng hạn như cấu hình sai, bản phát hành lỗi hoặc pipeline triển khai bị gián đoạn, cũng là nguồn gốc phổ biến của tình huống Server Down.

Trong môi trường Kubernetes, sự cố máy chủ ngừng hoạt động thường biểu hiện thông qua các trạng thái đặc trưng mà hệ thống điều phối ghi nhận. Ví dụ, Pod rơi vào trạng thái CrashLoopBackOff khi tiến trình ứng dụng liên tục khởi động thất bại, Container bị OOMKilled khi vượt quá hạn mức bộ nhớ định trước, Node chuyển sang trạng thái NotReady khi thành phần kubelet không thể duy trì kết nối với control plane, hoặc Deployment bị đánh dấu là không sẵn sàng khi số lượng replica hoạt động thấp hơn mức cam kết. Mặc dù mỗi dạng biểu hiện mang một nguyên nhân trực tiếp khác nhau, điểm chung là chúng đều khiến thành phần thực thi bị loại khỏi dòng xử lý chính của hệ thống, từ đó gây ảnh hưởng tức thời đến các dịch vụ phụ thuộc.

Điểm đáng lưu ý là sự cố máy chủ ngừng hoạt động thường gây gián đoạn hoàn toàn nếu hệ thống không có các cơ chế chịu lỗi phù hợp. Trong các kiến trúc không có dự phòng, việc một thực thể ngừng hoạt động có thể khiến toàn bộ dịch vụ bị tê liệt. Ngay cả trong các hệ thống có khả năng mở rộng theo chiều ngang, việc thiếu phân phối tải hoặc cấu hình không đồng nhất giữa các replica có thể khiến downtime kéo dài hơn dự kiến. Trong môi trường đám mây, cơ chế tự phục hồi như tự động tạo lại pod, tự động thay thế node hoặc tự động cân bằng tải góp phần giảm mức độ nghiêm trọng của sự cố, nhưng các cơ chế này chỉ hiệu quả khi nguyên nhân không mang tính hệ thống. Khi sự cố lặp lại ở nhiều thực thể cùng lúc, hoặc có nguồn gốc từ lỗi cấu hình hoặc lỗi triển khai diện rộng, các cơ chế tự động này có thể trở nên vô hiệu hoặc thậm chí làm trầm trọng thêm tình hình.

Tác động của sự cố Server Down không chỉ dừng lại ở việc gián đoạn xử lý yêu cầu. Do hầu hết các dịch vụ trong kiến trúc microservices đều có mối liên hệ phụ thuộc, sự cố tại một thực thể có thể dẫn đến tình trạng yêu cầu bị treo, gửi lại nhiều

lần, hoặc gây áp lực lên các thành phần dự phòng. Điều này dễ dẫn đến hiệu ứng lan truyền trong toàn bộ hệ thống, khiến các dịch vụ trước đó vẫn hoạt động ổn định cũng rơi vào trạng thái quá tải hoặc mất đáp ứng. Chính vì vậy, đây là một loại sự cố đòi hỏi hệ thống phải triển khai các biện pháp giảm thiểu đa tầng, bao gồm khả năng phát hiện nhanh, cô lập thành phần lỗi, tự động tái tạo phiên bản thay thế, điều phối lại tải, hoặc chuyển vùng sang các node hoặc khu vực khác.

Do tính phổ biến, mức độ ảnh hưởng lớn và khả năng lan truyền của nó, sự cố máy chủ ngừng hoạt động được xem là một trong những dạng sự cố nghiêm trọng nhất cần được hệ thống tự động hoá phân tích và xử lý. Nó cũng là một trường hợp điển hình để đánh giá các hệ thống đa tác tử và mô hình ngôn ngữ lớn trong khả năng triage, suy luận nguyên nhân và đề xuất biện pháp phục hồi phù hợp.

1.2.2. Sự cố thiếu hụt tài nguyên hệ thống

Sự cố gián đoạn do thiếu hụt tài nguyên là một trong những nhóm sự cố xuất hiện phổ biến nhất trong các hệ thống đám mây quy mô lớn, đặc biệt trong các môi trường có lưu lượng truy cập biến động mạnh và cấu trúc triển khai đa tầng. Khác với sự cố ngừng hoạt động đột ngột của máy chủ, nhóm sự cố này thường diễn tiến theo hướng suy giảm hiệu năng trước khi dẫn đến mất khả năng phục vụ hoàn toàn. Điều này khiến quá trình phát hiện sớm trở nên khó khăn hơn và thường kéo dài thời gian khắc phục sự cố.

Về bản chất, sự cố gián đoạn do tài nguyên xuất hiện khi nguồn lực tính toán hoặc truyền tải dữ liệu của hệ thống không đủ để đáp ứng khối lượng công việc thực tế, hoặc bị phân bổ không hợp lý giữa các thành phần dịch vụ. Trong các môi trường container hóa và điều phối tự động như Kubernetes, tài nguyên được quản lý thông qua cơ chế định mức (resource requests/limits). Khi cấu hình không chính xác, việc phân bổ thiếu có thể khiến ứng dụng liên tục rơi vào trạng thái cạn kiệt tài nguyên, trong khi phân bổ dư thừa lại làm giảm hiệu suất sử dụng tổng thể và ảnh hưởng đến khả năng mở rộng của hệ thống. Sự mất cân đối này là nguồn gốc quan trọng của nhiều sự cố gián đoạn liên quan đến tài nguyên.

Ba loại tài nguyên có mức độ ảnh hưởng cao nhất trong nhóm sự cố này bao gồm CPU, bộ nhớ và tài nguyên mạng. Trong trường hợp CPU bị quá tải, ứng dụng phải cạnh tranh luồng xử lý, dẫn đến thời gian thực thi bị kéo dài, độ trễ tăng đột biến và các tác vụ quan trọng không được xử lý kịp thời. Hiện tượng này thường xảy ra khi lưu lượng tăng đột ngột, khi ứng dụng thực hiện các tiến trình nặng bất thường hoặc khi vòng lặp không kiểm soát làm tiêu thụ tài nguyên liên tục. Ở khía cạnh bộ nhớ, thiếu hụt tài nguyên khiến hệ thống phải kích hoạt cơ chế loại bỏ tiến trình (OOMKill) để bảo vệ node, dẫn đến việc container hoặc pod bị chấm dứt đột ngột. Đây là dạng lỗi vừa mang tính phá vỡ hoạt động dịch vụ vừa khó khắc phục triệt để nếu ứng dụng có hiện tượng rò rỉ bộ nhớ hoặc phân bổ bộ nhớ không hợp lý. Đối với mạng và I/O, nghẽn băng thông, độ trễ tăng hoặc số lượng kết nối vượt giới hạn có thể khiến dịch vụ không thể giao tiếp với các thành phần phụ thuộc, gây ra lỗi kết nối dây chuyền hoặc làm tắc nghẽn các tuyến dịch vụ quan trọng.

Một đặc điểm đáng chú ý của nhóm sự cố này là quá trình diễn tiến thường bắt đầu từ những thay đổi nhỏ, khó nhận biết trong chỉ số vận hành, chẳng hạn như độ trễ trung bình tăng nhẹ, tỷ lệ lỗi phân tán theo thời gian hoặc mức sử dụng tài nguyên tăng dần theo chu kỳ. Những tín hiệu này dễ bị bỏ qua nếu không có cơ chế giám sát sâu hoặc hệ thống cảnh báo có ngưỡng điều chỉnh phù hợp. Khi sự cố đạt đến ngưỡng biểu hiện rõ rệt, hệ thống thường đã rơi vào trạng thái mất ổn định, khiến thời gian phục hồi kéo dài hơn nhiều so với các dạng sự cố khác.

Tính lan truyền của nhóm sự cố này đặc biệt nghiêm trọng trong các kiến trúc microservices, nơi mỗi dịch vụ đóng vai trò trong một chuỗi xử lý liên tục. Khi một dịch vụ bị suy giảm hiệu năng do thiếu tài nguyên, thời gian phản hồi tăng lên, dẫn đến việc các dịch vụ phía trên phải chờ đợi lâu hơn hoặc gửi lại yêu cầu nhiều lần. Điều này làm tăng tải tổng thể, thúc đẩy hiện tượng nghẽn cổ chai và có thể khiến nhiều dịch vụ vốn hoạt động ổn định cũng rơi vào trạng thái quá tải. Vòng phản hồi tiêu cực này khiến hiệu ứng lan truyền trở nên mạnh hơn và dẫn đến sự cố gián đoạn ở phạm vi rộng.

Các nguyên nhân thường gặp của dạng sự cố này có thể bắt nguồn từ cấu hình giới hạn tài nguyên không phù hợp, từ hiện tượng tăng lưu lượng bất thường ngoài dự kiến, hoặc từ lỗi trong ứng dụng như rò rỉ bộ nhớ, các vòng lặp tiêu thụ CPU bất thường, hoặc việc mở kết nối mạng mà không được giải phóng đúng cách. Vì các dấu hiệu cảnh báo thường âm thầm và khó quan sát trực tiếp, việc chẩn đoán sự cố chủ yếu dựa vào khả năng phân tích dữ liệu quan sát, tức là kết hợp thông tin từ logs, metrics và traces. Tuy nhiên, trong nhiều trường hợp, dữ liệu quan sát bị phân tán theo dịch vụ hoặc bị nhiễu bởi các chỉ báo không liên quan, khiến kỹ sư vận hành gặp khó khăn trong việc xác định đúng nguyên nhân gốc rễ.

Chính vì đặc thù này, nhóm sự cố gián đoạn do thiếu hụt tài nguyên thường là nguyên nhân trực tiếp kéo dài MTTR và làm gia tăng chi phí vận hành. Các hệ thống phân tích tự động, đặc biệt là các mô hình có khả năng học từ dữ liệu quan sát và suy luận theo ngữ cảnh, được xem là giải pháp tiềm năng giúp phát hiện sớm, nhận diện mẫu bất thường theo chiều thời gian và dự báo các tình huống có nguy cơ cao trước khi sự cố lan rộng. Điều này cũng lý giải vì sao nhóm sự cố này là một trong những đối tượng trọng tâm trong các hệ thống vận hành thông minh và các nghiên cứu ứng dụng mô hình đa tác tử trong phân tích sự cố.

1.2.3. Sự cố phát sinh từ lỗi trong chuỗi dịch vụ phụ thuộc

Trong các hệ thống ứng dụng hiện đại được xây dựng theo kiến trúc microservices, mỗi dịch vụ hiếm khi hoạt động độc lập mà thường phụ thuộc vào một hoặc nhiều thành phần khác trong chuỗi cung ứng dịch vụ. Các thành phần này có thể là API nội bộ, gateway, dịch vụ xác thực, hệ thống lưu trữ dữ liệu, message broker hoặc các dịch vụ do bên thứ ba cung cấp. Sự liên kết và phụ thuộc chặt chẽ như vậy mang lại tính linh hoạt và khả năng mở rộng, nhưng đồng thời cũng làm gia tăng nguy cơ xuất hiện các dạng gián đoạn mang tính dây chuyền, trong đó một lỗi tại một thành phần phụ thuộc có khả năng gây suy giảm hoặc gián đoạn dịch vụ ở một thực thể hoàn toàn khác. Đây được xem là một trong những nhóm sự cố phức tạp nhất trong hệ thống phân tán, do cơ chế lan truyền khó dự đoán và tính ẩn của nguyên nhân gốc rễ.

Bản chất của sự cố trong chuỗi phụ thuộc nằm ở việc dịch vụ trực tiếp chịu ảnh hưởng không phải là nơi khởi phát lỗi, mà chỉ là điểm cuối cùng trong chuỗi truyền lan. Một dịch vụ có thể hoạt động hoàn toàn bình thường từ góc nhìn nội bộ, nhưng vẫn không thể đáp ứng yêu cầu của người dùng do thành phần mà nó phụ thuộc không phản hồi hoặc phản hồi quá chậm. Điều này thường khiến kỹ sư vận hành mất nhiều thời gian để xác định nguyên nhân thực sự, bởi các triệu chứng ban đầu của sự cố thường xuất hiện tại dịch vụ cuối cùng trong chuỗi, trong khi lỗi gốc nằm ở một dịch vụ trung gian hoặc ở tầng hạ tầng giao tiếp.

Hiện tượng này trở nên đặc biệt phổ biến trong các tình huống mà dịch vụ phụ trợ đóng vai trò trung tâm của nhiều luồng xử lý. Ví dụ, sự chậm trễ hoặc mất đáp ứng từ dịch vụ xác thực có thể khiến tất cả các dịch vụ yêu cầu chứng thực người dùng bị ảnh hưởng, dẫn đến việc hệ thống không thể xử lý yêu cầu ngay cả khi phần lõi nghiệp vụ vẫn hoạt động bình thường. Tương tự, một cơ sở dữ liệu bị quá tải hoặc gặp lỗi khóa (deadlock) có thể khiến tất cả dịch vụ tiêu thụ dữ liệu bị treo, tạo ra sự gián đoạn trên diện rộng. Các message broker như Kafka hoặc RabbitMQ khi bị nghẽn hàng đợi cũng có thể làm trì hoãn toàn bộ luồng sự kiện, khiến các dịch vụ phản ứng chậm hoặc rơi vào trạng thái timeout. Ngay cả những dịch vụ hạ tầng tưởng như nhỏ bé như DNS, load balancer hoặc network overlay khi gặp lỗi cấu hình cũng có thể làm đứt gãy kết nối giữa các microservice, gây ra sự cố khó truy vết vì bản thân ứng dụng không gặp lỗi logic nào.

Một đặc điểm quan trọng của nhóm sự cố này là khả năng tạo ra hiệu ứng dây chuyền (cascading failure). Khi một dịch vụ phụ thuộc chậm hoặc không phản hồi, các dịch vụ phía trên thường kích hoạt cơ chế tái thử mặc định. Nếu được cấu hình không hợp lý, quá trình tái thử với tần suất cao có thể dẫn đến hiện tượng "retry storm", trong đó một lượng lớn yêu cầu tái thử đồng thời đổ về dịch vụ gặp sự cố. Thay vì giúp hệ thống phục hồi, cơ chế này làm tăng áp lực lên dịch vụ gốc, khiến nó càng khó đáp ứng và dẫn đến sự lan rộng của gián đoạn trên nhiều tầng dịch vụ. Hiệu ứng lan truyền này thường mở rộng rất nhanh và có thể ảnh hưởng tới toàn bộ hệ thống chỉ trong thời gian ngắn.

Thách thức lớn nhất khi xử lý những sự cố loại này nằm ở việc chúng mang tính chất "ẩn", không xuất hiện trực tiếp ở dịch vụ gặp lỗi mà thể hiện thông qua các dấu hiệu gián tiếp như tăng độ trễ, tỷ lệ timeout cao hoặc lưu lượng bất thường. Việc chẩn đoán nguyên nhân gốc rễ do đó đòi hỏi năng lực tổng hợp từ nhiều nguồn dữ liệu quan sát khác nhau, bao gồm logs phân tán, traces liên dịch vụ, metrics theo thời gian thực và bản đồ phụ thuộc dịch vụ (service dependency graph). Các công cụ giám sát truyền thống dựa trên chỉ báo độc lập thường khó phát hiện được toàn cảnh sự cố, bởi việc quan sát một dịch vụ đơn lẻ không đủ để tái hiện mối quan hệ phụ thuộc đằng sau. Đây là lý do khiến các giải pháp AIOps hiện đại và các hệ thống phân tích sự cố dựa trên mô hình đa tác tử ngày càng chú trọng tới khả năng tương quan giữa các tín hiệu rời rạc, tìm kiếm pattern bất thường theo chiều sâu và mô phỏng luồng lan truyền của sự cố trong kiến trúc dịch vụ.

Ba nhóm sự cố được trình bày – bao gồm sự cố máy chủ ngừng hoạt động, sự cố thiếu hụt tài nguyên và sự cố trong chuỗi dịch vụ phụ thuộc – phản ánh ba cơ chế gián đoạn chính trong môi trường đám mây hiện đại. Chúng không chỉ khác nhau về nguồn gốc và đặc tính kỹ thuật, mà còn khác biệt đáng kể về biểu hiện quan sát, tốc độ lan truyền và phương pháp xử lý phù hợp. Việc nghiên cứu sâu ba nhóm sự cố này không chỉ giúp làm rõ tính đa dạng của các kịch bản gián đoạn trong hệ thống phân tán, mà còn tạo cơ sở quan trọng cho việc thiết kế một hệ thống hỗ trợ phân tích, suy luận và đề xuất giải pháp khắc phục dựa trên mô hình đa tác tử – đúng với mục tiêu nghiên cứu trọng tâm của đề án

1.3. Tổng quan về các công nghệ

Trong bối cảnh các hệ thống điện toán đám mây ngày càng phức tạp, việc đảm bảo khả năng giám sát, phân tích và hỗ trợ khắc phục sự cố đòi hỏi sự kết hợp của nhiều công nghệ tiên tiến. Phần này trình bày tổng quan về các công nghệ trọng tâm liên quan trực tiếp đến hướng nghiên cứu của đề án: *(1) Giám sát hệ thống và tự động hoá vận hành, (2) Kiến trúc đa tác tử, (3) Mô hình ngôn ngữ lớn, (4) Kỹ thuật tạo sinh tăng cường truy xuất.*

(1) Hệ thống giám sát và tự động hoá vận hành

Giám sát hệ thống là nền tàng để đánh giá tình trạng, hiệu năng và độ ổn định của hệ thống. Mục tiêu của giám sát là thu thập, phân tích và hiển thị dữ liệu vận hành (Như log, metric, trace) nhằm phát hiện sớm các bất thường và cảnh báo kịp thời cho đội ngũ vận hành.

Theo triết lý Site Reliability Engineering (SRE) của Google, quan sát là yếu tố cốt lõi để đạt được các chỉ tiêu độ tin cậy (SLO/SLA). Ba trụ cột chính của giám sát hệ thống gồm: (i) Logging - Ghi lại các sự kiện chi tiết trong quá trình hoạt động; (ii) Metrics - Đo lường hiệu năng hệ thống theo thời gian; và (iii) Tracing - Theo dõi luồng yêu cầu giữa các dịch vụ phân tán [7].

Cùng với giám sát hệ thống, tự động hoá vận hành đóng vai trò then chốt trong việc giảm phụ thuộc vào thao tác thủ công và rủi ro con người. Các công cụ hiện đại cho phép giám sát, cấu hình và triển khai hạ tầng dưới dạng mã *(Infrastructure as Code - IaC)*, từ đó rút ngắn thời gian phản hồi khi xảy ra sự cố.

Sự kết hợp giữa giám sát hệ thống và tự động hoá vận hành đã dẫn đến hình thành khái niệm AIOps *(Artificial Intelligence for IT Operations)* – Hướng tiếp cận ứng dụng học máy và xử lý ngôn ngữ tự nhiên để phân tích log, dự đoán sự cố và tự động đưa ra các gợi ý khắc phục [8]

(2) Hệ thống đa tác tử

Tác tử (Agent) trong ngôn ngữ của trí tuệ nhân tạo được hiểu là một thực thể phần mềm có khả năng cảm nhận môi trường, xử lý thông tin và thực hiện hành động nhằm hướng tới một mục tiêu xác định. Theo quan niệm kinh điển của Russell và Norvig, tác tử là "một hệ thống có tính tự chủ, hoạt động dựa trên việc quan sát và tương tác với môi trường để tối ưu hoá hành vi theo mục tiêu" [1]. Khả năng chủ động ra quyết định, thích ứng và duy trì trạng thái hoạt động là những điểm khác biệt căn bản của tác tử so với các chương trình điều khiển truyền thống.

Hệ thống đa tác tử (Multi-Agent System – MAS) là một cấu trúc gồm nhiều tác tử cùng hoạt động trong một môi trường chung, có thể phối hợp hoặc cạnh tranh nhằm đạt được mục tiêu chung hoặc mục tiêu cục bộ của từng tác tử. MAS kế thừa

tư tưởng duy nhất chịu trách nhiệm điều phối toàn bộ hoạt động, nhiều tác tử đảm nhiệm các vai trò chuyên biệt, tương tác với nhau thông qua khả năng phân tách nhiệm vụ, xử lý song song và tận dụng tri thức phân tán của từng tác tử để giải quyết những vấn đề phức tạp vượt ngoài năng lực của một tác tử đơn lẻ [2].

Các nghiên cứu nền tảng trong lĩnh vực [2][3][4] thống nhất rằng một tác tử thông minh thường được đặc trưng bởi bốn thuộc tính cốt lõi.

1. Tự chủ (Autonomy): Tác tử có khả năng kiểm soát hành vi và trạng thái nội tại, không phụ thuộc vào chỉ thị liên tục từ bên ngoài. Tính tự chủ cho phép tác tử vận hành bền vững trong môi trường phức tạp.

2. *Khả năng xã hội (Social Ability)*: Tác tử có thể tương tác với các tác tử khác hoặc với con người thông qua các giao thức chuẩn hoá, đảm bảo việc trao đổi tri thức và phối hợp hành động.

3. Phản ứng (Reactivity): Đây là khả năng của tác tử nhận biết những thay đổi trong môi trường và phản ứng kịp thời và thích hợp. Khả năng thích ứng theo thời gian thực này rất quan trọng để hoạt động trong các môi trường nhiều biến động và khó dự đoán.

4. Chủ động (Proactiviness): Trong khi phản ứng là về việc đáp lại các thay đổi, chủ động liên quan tới việc dự đoán nhu cầu và thực hiện các hành động hướng đến mục tiêu. Các tác tử chủ động không chỉ chờ đợi mọi thứ xảy ra; chúng chủ động để đạt được mục tiêu của mình.

Ngoài bốn đặc tính cốt lõi, các hệ thống hiện đại thường yêu cầu thêm những thuộc tính như khả năng thích nghi, năng lực xử lý đồng thời, tính phân tán và tính di động. Những đặc tính này tăng cường bản chất linh hoạt và phù hợp của tác tử đối với các môi trường có mức độ phức tạp cao, điển hình như hạ tầng điện toán đám mây.

(3) Mô hình ngôn ngữ lớn

Mô hình Ngôn ngữ lớn *(Large Language Model – LLM)* là loại mô hình học sâu được huấn luyện trên khối lượng văn bản khổng lồ nhằm học các quy luật ngữ

nghĩa, cú pháp và kiến thức tổng quát của ngôn ngữ tự nhiên. Các LLM Như GPT (OpenAI), Gemini (Google DeepMind) hay Claude (Anthropic) có khả năng hiểu và sinh ngôn ngữ tự nhiên với mức độ chính xác và linh hoạt vượt trội, giúp chúng có thể tham gia vào các tác vụ phức tạp như tóm tắt văn bản, phân tích dữ liệu hoặc đề xuất các giải pháp kỹ thuật.

Về mặt kiến trúc, hầu hết các mô hình ngôn ngữ lớn hiện nay được xây dựng trên mô hình Transformer [9], trong đó có cơ chế self-attention cho phép mô hình nắm bắt được mối quan hệ ngữ cảnh giữa các từ trong chuỗi dài. Năng lực này đặc biệt hữu ích trong môi trường vận hành hệ thống, nơi các chuỗi log, cảnh báo và tài liệu kỹ thuật đều mang tính ngữ cảnh liên kết cao.

Mô hình ngôn ngữ lớn đang được ứng dụng ngày càng rộng rãi trong lĩnh vực AIOps và DevOps, hỗ trợ phân tích nguyên nhân sự cố, gợi ý khắc vụ và sinh tài liệu hướng dẫn kỹ thuật. Tuy nhiên, nhược điểm của LLM là thiếu khả năng truy xuất thông tin thời gian thực và dễ "ảo giác" nội dung (hallucination), dẫn tới nhu cầu kết hợp với cơ chế tạo sinh tăng cương truy xuất để đảm bảo độ chính xác.

(4) Tạo sinh tăng cường truy xuất

Tạo sinh tăng cường truy xuất *(Retrieval-Augmented Generation – RAG)* là kỹ thuật kết hợp giữa mô hình truy xuất thông tin (Information Retrieval) và mô hình ngôn ngữ tạo sinh (Generative Model) nhằm nâng cao độ chính xác và tính cập nhật của đầu ra (Lewis et al., 2020). Trong quy trình RAG, khi người dùng hoặc hệ thống gửi truy vấn, module *retriever* sẽ tìm kiếm các đoạn thông tin liên quan trong kho tri thức, sau đó *generator* (thường là LLM) sẽ tổng hợp và sinh câu trả lời dựa trên cả truy vấn và tài liệu được truy xuất.

So với phương pháp tinh chỉnh (Fine-tuning) mô hình, RAG được lựa chọn là kỹ thuật chủ đạo trong đề án vì các lý do sau: Thứ nhất, Fine-tuning khiến tri thức bị "đóng gói" cứng trong trọng số mô hình, khó cập nhật khi các hệ thống đặc thù thay đổi quy trình vận hành. Thứ hai, RAG cho phép hệ thống cung cấp các dẫn chứng cụ thể từ kho tri thức nội bộ, giúp người vận hành dễ dàng kiểm chứng tính đúng đắn

của đề xuất – điều mà các mô hình tinh chỉnh thuần tuý thường gặp khó khăn do hiện tượng "hộp đen" và "ảo giác".

Một yếu tố quan trọng khác là việc triển khai RAG giúp tối ưu hoá chi phí tính toán do không cần quy trình huấn luyện lại phức tạp, đảm bảo tính khả thi cho một hệ thống nguyên mẫu.

Gần đây, các framework như LangChain, LlamaIndex, và tiêu chuẩn Model Context Protocol (MCP) (LangChain, 2024) đã định hình cách thức tích hợp RAG vào hệ thống AI nhiều tác tử, giúp tăng tính tương tác giữa các mô-đun và khả năng mở rộng trong môi trường đám mây.

1.4. Các yếu tố ảnh hưởng tới kết quả của hệ thống

Hiệu quả của hệ thống phân tích và đề xuất khắc phục sự cố trong môi trường đám mây phụ thuộc đồng thời vào nền tảng kỹ thuật, chất lượng dữ liệu quan sát, cấu trúc tri thức, cơ chế phối hợp của kiến trúc đa tác tử và mức độ tương tác của con người. Các yếu tố này quyết định trực tiếp tính chính xác, tính kịp thời và độ tin cậy của hệ thống.

Trước hết, sự phân tán và độ phức tạp của hạ tầng đám mây làm tăng khối lượng dữ liệu và độ trễ truyền thông, đòi hỏi kiến trúc xử lý phân tán để giảm tải cho thành phần trung tâm. Hiệu quả phân tích phụ thuộc lớn vào chất lượng dữ liệu quan sát; dữ liệu nhiễu, thiếu ngữ cảnh hoặc không đồng bộ sẽ làm giảm độ chính xác, do đó yêu cầu cơ chế chuẩn hoá và lọc nhiễu.

Cấu trúc tri thức cũng giữ vai trò then chốt, đặc biệt đối với hệ thống kết hợp LLM và RAG, nhằm đảm bảo truy xuất theo ngữ cảnh và hạn chế sai lệch của mô hình sinh. Trong kiến trúc đa tác tử, giá trị phân tích không đến từ từng tác tử riêng lẻ mà từ khả năng phối hợp, trao đổi và hợp nhất thông tin; vì vậy, giao thức truyền thông và cơ chế điều phối phải được thiết kế chặt chẽ. Bên cạnh đó, LLM dù mạnh về suy luận vẫn có nguy cơ tạo ra kết quả sai, nên cần bổ sung cơ chế kiểm chứng dựa trên dữ liệu sự cố thực tế.

Cuối cùng, vai trò con người vẫn mang tính quyết định nhằm đảm bảo tính minh bạch, khả năng giải thích và kiểm soát trong quá trình vận hành. Tóm lại, hiệu quả hệ thống là kết quả tổng hợp của dữ liệu, mô hình, tri thức, năng lực phối hợp của tác tử và sự tham gia của con người; quản trị tốt các yếu tố này là nền tảng để xây dựng một hệ thống phân tích và hỗ trợ khắc phục sự cố đáng tin cậy trong môi trường đám mây.

CHƯƠNG 2 - THIẾT KẾ HỆ THỐNG

2.1. Mô hình hoá bài toán nghiên cứu

Sự phát triển của kiến trúc Cloud-Native dựa trên các công nghệ ảo hoá như Container và các nền tảng điều phối mang lại tính linh hoạt và khả năng mở rộng vượt trội, nhưng song song với đó là sự gia tăng đáng kể độ phức tạp vận hành. Trong các hệ thống đám mây quy mô lớn, lượng dữ liệu quan sát được tạo ra từ nhiều nguồn, nanh chóng vượt quá khả năng xử lý của kỹ sư vận hành, khiến các bước phát hiện, khoanh vùng và khắc phục sự cố vẫn kéo dài và phụ thuộc vào kinh nghiệm cá nhân.

Các hệ thống giám hiện hữu (Prometheus, ELK, Datadog, v.v.) vẫn chủ yếu dừng ở mức thu thập và trực quan hoá tín hiệu, thiếu khả năng suy luận ngữ nghĩa hoặc liên kết đa nguồn dữ liệu. Giải pháp Rule-based truyền thống không còn phù hợp trong một môi trường mà các thành phần bên trong biến động liên tục. Trong khi đó, các nền tảng AIOps hiện đại mới chỉ đạt mức tự động hoá cục bộ, chưa hình thành được mô hình thống nhất kết hợp phân tích ngữ nghĩa, truy xuất tri thức và phối hợp hành động.

Trên cơ sở khoảng trống này, bài toán nghiên cứu được mô hình hoá như một hệ thống nhận đầu vào là tập dữ liệu quan sát bao gồm Alert, Metrics và Logs từ các nền tảng giám sát. Hệ thống cần thực hiện phân tích ngữ nghĩa, suy luận mối tương quan, và khai thác tri thức quá khứ nhằm tạo ra các đầu ra gồm:

(1) Báo cáo phân tích nguyên nhân gốc rễ

(2) Đề xuất hành động khắc phục

Quá tình xử lý có thể được khái quát dưới mô hình toán học sau:

$$Output = f(Alert, Metrics, Logs, KnowledgeBase)$$

Trong đó:

- f: là quá trình phân tích ngữ nghĩa, suy luận từ hệ thống
- $Output$: Gồm tập các phân tích, đề xuất khắc phục có độ chính xác cao.

Hệ thống được kỳ vọng rút ngắn đáng kể thời gian phát hiện (MTTD) và thời gian khôi phục (MTTR), đồng thời giảm gánh nặng vận hành cho đội ngũ kỹ sư vận hành.

Tóm lại, bài toán nghiên cứu tập trung vào việc thiết kế một hệ thống phân tích và hỗ trợ khắc phục sự cố tự động trong môi trường đám mây, có khả năng hiểu ngữ nghĩa – học từ kinh nghiệm – và phối hợp hành động giữa các tác tử thông minh. Đây là bước tiến cần thiết để chuyển từ giai đoạn *giám sát thụ động* sang *vận hành chủ động và có trí tuệ*, mở đường cho các thế hệ hệ thống vận hành tự trị (autonomous operations) trong tương lai.

2.2. Đặc tả hệ thống

2.2.1. Đặc tả yêu cầu chức năng

Để đáp ứng mục tiêu nghiên cứu, hệ thống cần đảm bảo một số chức năng cốt lõi. Trước hết, hệ thống phải có khả năng thu thập và hợp nhất dữ liệu từ các nguồn hiện hữu như hệ thống giám sát, nhật ký hay các hệ thống vận hành. Dữ liệu sau khi thu thập được chuẩn hoá theo một cấu trúc thống nhất (Unified Schema), nhằm đảm bảo tính nhất quán, toàn vẹn và khả năng truy vết giữa các nguồn khác nhau.

Tiếp theo, hệ thống thực hiện phân tích và tổng hợp ngữ cảnh khi có cảnh báo phát sinh. Trong giai đoạn này, các dữ liệu liên quan trong cùng khoảng thời gian được truy vấn và đối chiếu nhằm phát hiện các mối tương quan bất thường giữa các thành phần của hệ thống. Kết quả được mô hình hoá dưới dạng đối tượng ngữ cảnh, phản ánh toàn diện diễn biến, phạm vi ảnh hưởng và các yếu tố tiềm ẩn liên quan đến sự cố.

Dựa trên ngữ cảnh đã xây dựng, hệ thống tiến hành suy luận nguyên nhân và đề xuất khắc phục. Song song với đó, hệ thống cần hỗ trợ truy xuất tri thức kỹ thuật từ kho tài liệu nội bộ, bao gồm tài liệu vận hành, kiến trúc, triển khai và cấu hình.

Cuối cùng, hệ thống phải hỗ trợ tương tác thông qua việc tích hợp với các nền tảng giao tiếp nhóm. Điều này cho pehps tự động gửi báo cáo phân tích, cảnh báo và các đề xuất xử lý dưới dạng thông báo tương tác, hỗ trợ kỹ sư vận hành ra quyết định kịp thời.

Các chức năng trên hình thành nên năng lực cốt lõi của hệ thống và đóng vai trò làm nền tảng cho quá trình thiết kế kiến trúc chi tiết.

2.2.2. Đặc tả yêu cầu phi chức năng

Hệ thống cần đáp ứng các yêu cầu phi chức năng nhằm đảm bảo khả năng vận hành ổn định trong môi trường đám mây phân tán. Trước hết là độ tin cậy và tính sẵn sàng cao, trong đó hệ thống phải có khả năng tự phục hồi và duy trì hoạt động liên tục ngày cả khi một số thành phần hoặc tác tự gặp sự cố.

Về hiệu năng và khả năng mở rộng, hệ thống phải xử lý được dữ liệu cảnh báo và nhật ký thời gian thực, với độ trễ thấp và thông lượng đủ lớn. Kiến trúc phi tập trung và khả năng mở rộng theo chiều ngang cho phép bổ sung tài nguyên mà không ảnh hưởng đến hoạt động cụ thể. Hệ thống phải có khả năng chống chịu bão cảnh báo trong tình huống phát sinh hàng nghìn cảnh báo do lỗi dây chuyền. Tầng tiếp nhận phải đảm bảo thông lượng lớn và khả năng lọc trùng ngay tại biên để không làm quá tải các tác tử suy luận. Độ trễ xử lý phải duy trì ở mức thấp để cung cấp cảnh báo kịp thời trong lúc sự cố đang leo thang.

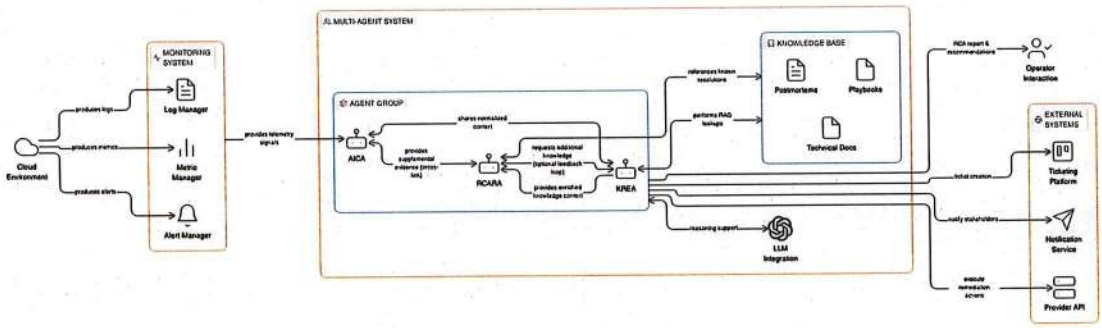
Bên cạnh đó, bảo mật và an toàn thông tin là yêu cầu trọng yếu. Mọi kênh trao đổi dữ liệu phải được mã hoá, kết hợp với cơ chế xác thực chặt chẽ, đặc biệt là trong quá trình tương tác với các mô hình ngôn ngữ lớn nhằm hạn chế nguy cơ rò rỉ thông tin nội bộ. Hệ thống phải triển khai cơ chế khử định danh dữ liệu trước khi gửi thông tin ra bên ngoài.

Cuối cùng, tính thân thiện trong tương tác giữa người vận hành và hệ thống cũng được đặt ra. Hệ thống phải tuân thủ nghiêm ngặt cơ chế con người trong vòng lặp (Human-in-the-loop), đảm bảo mọi đề xuất khắc phục phải được trình bày rõ ràng kèm theo các đánh giá rủi ro khi thực hiện. Hệ thống không thể tự ý can thiệp vào hạ tầng đám mây nếu chưa được lệnh xác nhận từ người vận hành, nhằm ngăn chặn các hành động sai sót có thể xảy ra gây gián đoạn nghiêm trọng hơn.

2.3. Thiết kế hệ thống

2.3.1. Kiến trúc tổng quan

Kiến trúc tổng quan của hệ thống được triển khai nhằm hỗ trợ phân tích và khắc phục sự cố gián đoạn trong môi trường điện toán đám mây được minh hoạ trong hình dưới đây.



Hình 2.1 Kiến trúc tổng quan của hệ thống

Kiến trúc này được thiết kế theo mô hình nhiều tầng, bảo đảm sự liên thông giữa các nguồn dữ liệu giám sát, các tác tử xử lý thông minh và các hệ thống vận hành bên ngoài, qua đó hình thành một chu trình phân tích – suy luận – phản hồi mang tính tự động hoá cao.

Ở tầng ngoài cùng là môi trường đám mây, nơi phát sinh toàn bộ dữ liệu phục vụ cho quá trình giám sát và phân tích. Các thành phần thu thập dữ liệu như hệ thống quản lý nhật ký, hệ tống quản lý chỉ số vận hành và điều phối cảnh báo thực hiện chức năng tiếp nhận, tập hợp và chuyển tiếp dữ liệu. Những dữ liệu này phản ánh trung thực trạng thái của hạ tầng và ứng dụng, cũng như là đầu vào trực tiếp cho toàn bộ chu trình phân tích sự cố.

Dữ liệu sau khi được tiếp nhận sẽ đi vào tầng xử lý đầu tiên của hệ thống, nơi thực hiện chức năng tiếp nhận (Ingestion) và chuẩn hoá. Việc chuẩn hoá nhằm giảm

thiểu sai lệch, tránh trùng lặp và bảo đảm cho các tác tử xử lý ở tầng tiếp theo có thể truy vấn và phân tích trên một nền dữ liệu nhất quán.

Trung tâm của kiến trúc là hệ thống đa tác tử (Multi-Agent System). Đây là lớp xử lý có tính chất quyết định, trong đó mỗi tác tử đảm nhiệm một vai trò chuyên biệt nhưng phối hợp chặt chẽ với nhau dưới sự điều phối của tác tử giám sát. Chi tiết về việc thiết kế, định nghĩa các tác tử sẽ được mô tả trong phần nội dung tiếp theo.

Bên cạnh cơ chế đa tác tử, hệ thống còn tích hợp với mô hình ngôn ngữ lớn để tăng cường khả năng hiểu ngữ nghĩa và rút ra quan hệ nhân – quả từ dữ liệu quan sát. LLM được sử dụng kết hợp với cơ chế truy xuất tri thức tăng cường, giúp vừa khai thác khả năng suy luận của mô hình, vừa đảm bảo tính chính xác của thông tin bằng cách dựa trên kho tri thức đặc thù của hệ thống.

Các kết quả phân tích và đề xuất được truyền tới các hệ thống bên ngoài, thông qua đó để xây dựng các luồng tự động tạo yêu cầu xử lý hay gửi thông báo về tình trạng sự cố hoặc đưa ra các phương án giải quyết cụ thể cho kỹ sư.

Như vậy, về mặt tổng quan, hệ thống có tính phân lớp rõ ràng, khả năng phối hợp dữ liệu, tác tử và mô hình suy luận, cùng với cơ chế tương tác với môi trường vận hành bên ngoài. Thiết kế này cho phép hệ thống không chỉ phát hiện và phân tích sự cố nhan chóng, mà còn đưa ra các phương án khắc phục phù hợp.

2.3.2. Thiết kế hệ thống đa tác tử

Thiết kế kiến trúc đa tác tử là phần trọng tâm của hệ thống, quyết định khả năng tiếp nhận tín hiệu cảnh báo, mở rộng ngữ cảnh bằng tri thức bổ sung và thực hiện phân tích để rút ra nguyên nhân gốc cũng như đề xuất giải pháp khắc phục.

Trong khuôn khổ của đề án tốt nghiệp, các giới hạn và ràng buộc đã đề cập ở trên, kiến trúc đa tác tử được thiết kế tối giản nhưng vẫn đảm bảo bao phủ đầy đủ chuỗi xử lý sự cố từ đầu vào đến đầu ra. Kiến trúc được tổ chức theo mô hình "Phân tầng chức năng", trong đó mỗi tác tử đảm nhiệm một vai trò độc lập nhưng liên kết chặt chẽ thông qua luồng dữ liệu và giao tiếp nội bộ.

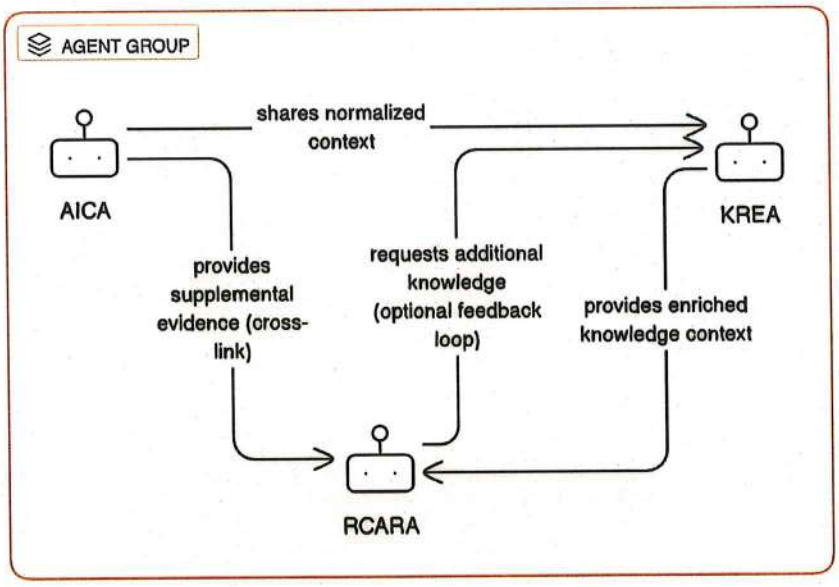
Việc thiết kế tác tử trong hệ thống dựa trên ba nguyên tắc:

(i) Mỗi tác tử chỉ thực hiện một nhóm chức năng chuyên biệt, tránh trùng lặp vai trò.

(ii) Các tác tử phải giao tiếp bằng ngữ cảnh được chuẩn hoá, bảm đảm quá trình truyền thông tin diễn ra nhất quán.

(iii) Toàn bộ luồng xử lý phải hướng tới kết quả cuối cùng là báo cáo nguyên nhân gốc và gợi ý khắc phục có thể áp dụng triển khai.

Các nguyên tắc này đảm bảo hệ thống có cấu trúc rõ ràng, dễ mở rộng khi cần tích hợp thêm các tác tử mới.



Hình 2.2 Thiết kế các tác tử của hệ thống

Hệ thống được thiết kế bao gồm ba tác tử chính, được mô tả như trên. Trong các phần nội dung tiếp theo, đề án sẽ phân tích chi tiết về từng tác tử.

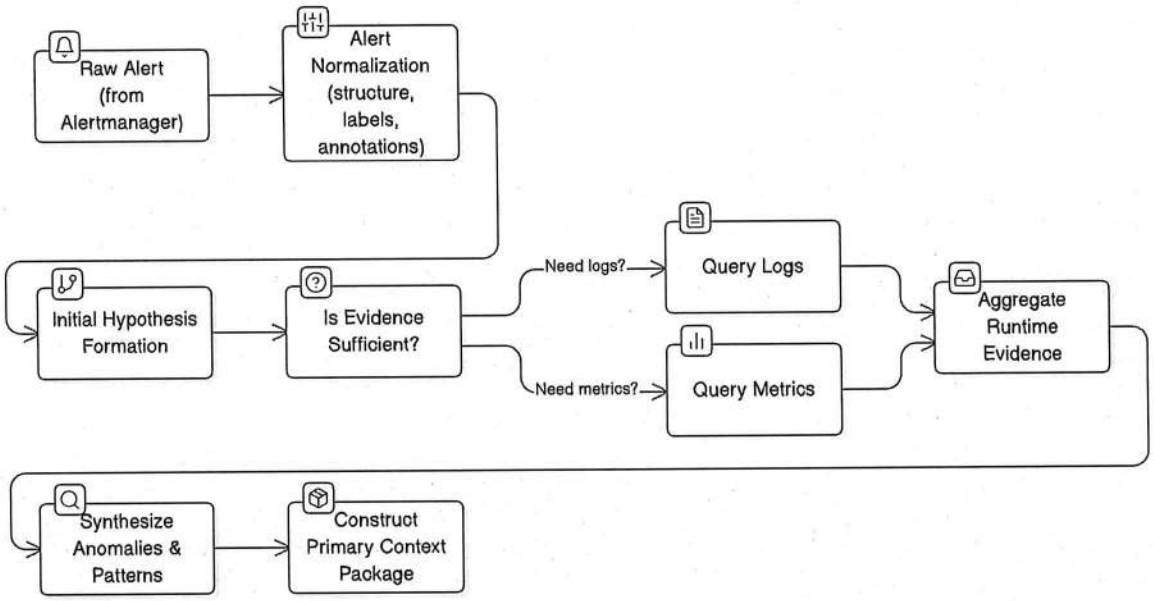
a. Tác tử tiếp nhận và chuẩn hoá cảnh báo – Alert Ingestion & Context Agent (AICA)

AICA được thiết kế như điểm khởi đầu của toàn bộ đường ống phân tích sự cố. Khi một tín hiệu cảnh báo được sinh ra từ môi trường giám sát, tác tử này tiếp nhận thông tin nguyên bản, tiến hành chuẩn hoá dữ liệu và triển khai quá trình suy luận từng bước nhằm hình thành gói ngữ cảnh sơ cấp.

Giai đoạn chuẩn hoá của AICA không chỉ dừng lại ở việc định dạng dữ liệu mà còn bao gồm bước làm mờ các thông tin nhạy cảm. Các đoạn mã được xây dựng sử dụng biểu thức chính quy (Regex) để nhận diện các chuỗi văn bản nhạy cảm. Các

thông tin này sẽ được thay thế bằng các mã đại diện (Ví dụ: <INTERNAL_IP>, <USER_ID>) trước khi được gửi tới các mô hình ngôn ngữ lớn để tiến hành suy luận. Quá trình này giúp giữ lại logic để mô hình phân tích nhưng vẫn bảo vệ được các thông tin nhạy cảm.

Quá trình nhận thức, suy luận của AICA được mô tả như sau:



Hình 2.3 Sơ đồ mô tả quá trình suy luận của AICA

Khác với các mô hình tiếp nhận tín hiệu mang tính bị động, AICA chủ động thực hiện quá trình suy luận nội tại theo hướng khám phá (Exploratory reasoning). Trong suốt tiến trình này, tác tử đánh giá từng giả thuyết tiềm năng về nguyên nhân trước khi chuyển sang giai đoạn thu thập bằng chứng.

Điểm nhấn quan trọng của AICA là khả năng sử dụng các công cụ bổ trợ trong quá trình suy luận. Chẳng hạn, sau khi phân tích sơ bộ thông tin cảnh báo, AICA có thể kết luận rằng dữ liệu hiện tại chưa đủ để đánh giá đúng mức độ ảnh hưởng hoặc xác định dạng sự cố tiềm năng. Khi đó, tác tử chủ động gọi tới các dịch vụ truy vấn log hoặc metric nhằm bổ sung bằng chứng. Các đoạn log bất thường, sự thay đổi đột ngột về tài nguyên, hay tri thức từ các sự kiến thời gian thực đều được AICA tích hợp vào phân tích.

AICA tổ chức hoạt động theo một chu trình gồm bốn giai đoạn chính: chuẩn hoá tín hiệu đầu vào, khởi tạo suy luận và phân rã giả thuyết, thu thập bằng chứng bổ sung bằng cách sử dụng các công cụ tích hợp, và cuối cùng tổng hợp tất cả các chứng cứ để hình thành gói ngữ cảnh sơ cấp *(Chi tiết cấu trúc trong phần Phụ lục)*. Gói dữ liệu này chứa toàn bộ thông tin đã được chuẩn hoá, các bằng chứng mở rộng và cấu trúc suy luận ban đầu, đóng vai trò nền tảng cho tác tử ở giai đoạn tiếp theo.

b. Tác tử truy xuất tri thức và tăng cường ngữ cảnh phân tích – Knowledge Retrieval & Enrichment Agent (KREA)

Sau khi AICA hoàn tất giai đoạn chuẩn hoá và thu thập bằng chứng, KREA tiếp nhận gói ngữ cảnh sơ cấp và triển khai quá trình mở rộng tri thức. KREA được thiết kế để khai thác tối đa nguồn tri thức tĩnh của hệ thống như tài liệu kỹ thuật, runbook, báo cáo sự cố trước đây, và các mô tả chuyên môn khác thông qua cơ chế truy xuất tăng cường.

Hoạt động nội tại của tác tử bao gồm xây dựng không gian tri thức liên quan tới sợ cố thông qua việc phân tích ngữ cảnh sơ cấp, tìm kiếm các tri thức liên quan dựa trên mức độ tương đồng ngữ nghĩa, và kết hợp tri thức với bằng chứng trong thời gian thực. Việc kết hợp tri thức này không đơn thuần là bổ sung thông tin mà còn nhằm tạo ra một lớp suy luận nâng cao, trong đó các quy tắc hoạt động, mẫu sự cố đã gặp trong quá khư và các hướng dẫn khắc phục được chuẩn hoá và lồng ghép vào ngữ cảnh vận hành.

Kết quả của quá trình này là gói ngữ cảnh tăng cường *(Chi tiết cấu trúc trong phần Phụ lục)* – Một cấu trúc thông tin chứa toàn bộ bằng chứng từ môi trường vận hành, tri thức từ RAG và sự liên kết logic giữa hai dạng thông tin này. Đây là đầu vào thiết yếu để thực hiện suy luận nguyên nhân gốc ở mức độ sâu hơn.

c. Tác tử tiếp nhận và chuẩn hoá cảnh báo – Root Cause Analysis & Remediation Agent (RCARA)

RCARA là tác tử thực hiện giai đoạn phân tích có chiều sâu nhất, trong đó nó xây dựng và đánh giá chuỗi quan hệ giữa triệu chứng và các giả thuyết nguyên nhân.

Bằng cách dựa trên cả bằng chứng thời gian thực và tri thức trích xuất từ KREA, RCARA kiến tạo một chu trình suy luận có cấu trúc nhằm xác định nguyên nhân gốc của sự cố.

Phương pháp phân tích của RCARA thường bao gồm đánh giá chuỗi nhân quả, đối chiếu các mẫu hỏng hóc đã biết, áp dụng mô hình lỗi phổ biến và phân tích vòng lặp dẫn đến sự cố. Tác tử này cũng đảm bảo toàn bộ quá trình suy luận đều được ghi nhận trong dạng "reasoning trace", nhằm hỗ trợ khả năng giải thích và phục vụ kiểm chứng trong các pha vận hành thực tế.

Kết quả cuối cùng mà RCARA tạo ra là một báo cáo RCA hoàn chỉnh, bao gồm mô tả sự cố, nguyên nhân gốc, diễn giải lập luận và bộ giải pháp khắc phục kèm theo các khuyến nghị phòng ngừa nhằm giải khả năng tái diễn.

2.3.3. Luồng dữ liệu

Trong kiến trúc đa tác tử được thiết kế cho hệ thống phân tích và hỗ trợ khắc phục sự cố trong môi trường điện toán đám mây, dữ liệu không chỉ đóng vai trò là nguyên liệu đầu vào cho quá trình suy luận mà còn quyết định mức độ chính xác của các kết luận được hình thành. Luồng dữ liệu, vì vậy, là yếu tố trung tâm phản ánh cách thức thông tin được tiếp nhận, chuẩn hoá, chuyển đổi, tăng cường và cuối cùng trở thành tri thức vận hành. Phần nội dung này trình bày hai góc nhìn bổ sung cho việc hiểu hệ thống: Luồng dữ liệu tổng thể từ góc độ vòng đời của sự cố và luồng dữ liệu giữa các tác tử.

a. Luồng dữ liệu tổng thể

Luồng dữ liệu tổng thể mô tả vòng đời của dữ liệu từ môi trường vận hành đến khi tri thức được hoàn thiện và phản hồi quay trở lại người vận hành. Dòng chảy này bắt đầu tại hệ thống giám sát, nơi một tín hiệu cảnh báo được sinh ra dưới dạng sự kiện (Event) và được chuyển tiếp thông qua API hoặc webhook.

Khi tín hiệu này đến AICA, hệ thống bước vào giai đoạn chuẩn hoá và trích xuất đặc trưng. Trong suốt quá trình này, hệ thống cắt giảm độ nhiễu, đồng thời giữ

lại các phần tử mang ngữ nghĩa vận hành quan trọng. Gói ngữ cảnh sơ cấp sau đó được chuyển đến KREA, nơi tri thức được tăng cường nhờ mô hình RAG.

Một điểm đặc trưng của luồng dữ liệu tổng thể là việc duy trì cấu trúc tuần tự nhưng không hoàn toàn tuyến tính; dữ liệu có thể được tái sử dụng cho nhiều mục đích khác nhau như giám sát lịch sử, phân tích thống kê, hoặc huấn luyện mô hình trong tương lai. Khi RCARA hoàn thành quá trình suy luận nguyên nhân gốc, dữ liệu đầu ra được chuyển đến các kênh giao tiếp, tạo nên vòng đời phản hồi hoàn chỉnh giữa máy và con người.

b. Luồng dữ liệu giữa các tác tử

Luồng dữ liệu giữa các tác tử được hình thành theo cấu trúc tăng trưởng ngữ cảnh (Context Accumulation Pipeline), trong đó, mỗi tác tử là một mô-đun xử lý tri thức độc lập, tiếp nhận dữ liệu đã được chuẩn hoá từ pha trước và chuyển nó thành cấu trúc tri thức giàu hơn. Chính sự kế tiếp này đảm bảo rằng các tác tử không chỉ truyền dữ liệu mà còn "gia tăng giá trị tri thức" cho từng giai đoạn.

Quá trình bắt đầu ở tác tử AICA, nơi các tín hiệu cảnh báo từ hệ thống giám sát được tiếp nhận dưới dạng JSON bán cấu trúc. Dữ liệu cảnh báo thường chứa các thông tin định danh *(alert_id, severity)*, các nhãn vận hành, cùng với các annotation mô tả ngữ nghĩa sự cố. Một cảnh báo tiêu biểu có thể được hệ thống nhận diện với cấu trúc:

```
{
  "alert_id": "ALRT-98231",
  "severity": "critical",
  "starts_at": "2025-11-20T10:32:21Z",
  "labels": {
    "service": "payment-api",
    "pod": "payment-api-7c9d8f"
  },
  "annotations": {
    "summary": "High memory usage",
    "description": "Memory usage > 90% for 5 minutes"
  }
}
```

Dữ liệu này được AICA chuẩn hoá và chuyển đổi thành gói ngữ cảnh sơ cấp, một cấu trúc có dạng trừu tượng hơn nhằm thể hiện những đặc trưng có ý nghĩa đối

với phân tích sự cố. Gói ngữ cảnh sơ cấp không chỉ sao chép dữ liệu giám sát mà còn tái cấu trúc chúng thành các "đặc trưng vận hành" (Operational features), như "anomaly score" hay "temporal pattern". Ví dụ, một gói ngữ cảnh sơ cấp có thể được mô tả như sau:


```
{
  "context_id": "CTX-20251120-12301",
  "affected_components": { "service": "payment-api", "pod": "payment-api-7c9d8f" },
  "evidence": { "metrics": { ... }, "logs_ref": ["log_9821"] },
  "feature_set": {
    "anomaly_score": 0.87,
    "temporal_pattern": "sustained_spike"
  }
}
```


Khi gói ngữ cảnh sơ cấp được chuyển đến KREA, dữ liệu bắt đầu bước sang giai đoạn tăng cường tri thức. Tác tử này trích xuất tri thức từ runbooks, tài liệu kỹ thuật, lịch sử sự cố và các mẫu vận hành tương tự được lưu trữ trong cơ sở dữ liệu vector. KREA hợp nhất những đoạn tri thức này vào gói ngữ cảnh, hình thành gói ngữ cảnh tăng cường.


```
{
  "context_id": "CTX-20251120-12301",
  "knowledge_evidence": [
    {
      "doc_id": "runbook-23",
      "relevance_score": 0.81,
      "snippet": "Restarting the payment-api pod resolves memory leak issues..."
    }
  ],
  "topology_context": {
    "upstream": ["auth-service"],
    "downstream": ["order-service"]
  }
}
```


Cuối cùng, RCARA tiếp nhận gói ngữ cảnh tăng cường và sử dụng LLM để hình thành luận giải nguyên nhân gốc kèm theo chuỗi lập luận (reasoning chain). Đầu ra của RCARA bao gồm một báo cáo suy luận về nguyên nhân sự cố và danh sách các phương án khắc phục được đánh giá theo mức độ rủi ro và khả năng áp dụng:

```
{
  "root_cause": {
    "hypothesis": "Memory leak in payment-api",
    "confidence": 0.78,
    "reasoning_chain": [
      "High memory usage detected for >5 minutes",
      "Historical incidents suggest pattern matches memory leak",
      "Runbook 23 indicates restart resolves similar conditions"
    ]
  },
  "recommended_actions": [
    { "action": "restart_pod", "risk_level": "low" }
  ]
}
```

Luồng dữ liệu giữa các tác tử, theo đó, phản ánh sự tiến hoá từ dữ liệu thô sang tri thức giải thích được, vốn là đặc trưng cơ bản của hệ thống đa tác tử định hướng tri thức trong môi trường phân tích sự cố.

2.3.4. Luồng tích hợp LLM và RAG

a. Luồng tích hợp giữa hệ thống đa tác tử và mô hình ngôn ngữ lớn

Trong kiến trúc đa tác tử được đề xuất, mô hình ngôn ngữ lớn đóng vai trò như một "bộ xử lý tri thức phổ quát", hỗ trợ các tác tử thực hiện các nhiệm vụ phân tích, suy luận và tổng hợp thông tin mà các thuật toán truyền thống không thể đảm đương một cách hiệu quả. Tuy nhiên, các tả tử không tương tác với LLM một cách đồng nhất; thay vào đó, mỗi tác tử định hình các phương thức tương tác riêng dựa trên mục tiêu vận hành và cấu trúc dữ liệu mà nó xử lý.

Đối với tác tử AICA, mô hình ngôn ngữ lớn giữ vai trò hỗ trợ phân tích ngôn ngữ tự nhiên trong phần "annotation" của cảnh báo, trích xuất các thực thể liên quan (Affected service, triệu chứng chính, điều kiện xảy ra), và bổ sung các đặc trưng có tính mô tả dữ liệu metrics/log vốn không thể biểu đạt.

Ví dụ, từ "annotation" nguyên bản:

```
"Memory usage > 90% for 5 minutes, possible degradation in payment-api module"
```

LLM giúp AICA trích xuất các thực thể:

```
{
  "symptom": "memory usage spike",
  "component": "payment-api",
  "possible_impact": "service degradation",
  "time_window": "5 minutes"
}
```

Kết quả phân tích này được hợp nhất vào gói ngữ cảnh sơ cấp, làm tăng độ sắc nét của dữ liệu đầu vào trước khi chuyển tới KREA.

KREA sử dụng LLM như một bộ máy tổng hợp tri thức, nhằm giải thích các đoạn runbook hoặc tài liệu truy xuất từ RAG theo cách phù hợp với ngữ cảnh sự cố đang xử lý. Ví dụ, sau khi RAG trả về đoạn tri thức:

```
"Restarting the service resolves memory leak conditions in most cases."
```

KREA yêu cầu LLM: (i) Diễn đạt lại theo ngữ cảnh cụ thể của *payment-api*; (ii) Đánh giá mức độ tương thích giữa tri thức tìm được và dữ liệu cảnh báo; (iii) Trừ tượng hoá tri thức thành dạng có thể gắn với biểu diễn feature-level.

LLM Trả về:

```
{
  "interpretation": "The memory spike in payment-api is consistent with known memory
leak patterns.",
  "relevance": 0.82,
  "recommended_probe": "Check GC activity or memory allocation patterns."
}
```

Đầu ra này giúp hình thành gói ngữ cảnh tăng cường, tạo nên nền tảng tri thức vững chắc cho tác tử suy luận phía sau.

Đối với RCARA, mô hình ngôn ngữ lớn đóng vai trò quan trọng nhất, giúp tổng hợp toàn bộ ngữ cảnh, chuyển chúng thành một reasoning prompt có cấu trúc kỹ thuật cao và yêu cầu mô hình ngôn ngữ lớn:

- *Xây dựng giả thuyết nguyên nhân gốc*
- *Hình thành chuỗi lập luận (Reasoning Chain)*
- *Đánh gía độ tin cậy*
- *Đề xuất phương án khắc phục khả thi.*

Đầu ra sau đó được sử dụng để báo cáo, gửi đến cho các kỹ sư vận hành qua các kênh giao tiếp.

b. Luồng tích hợp giữa hệ thống đa tác tử và RAG

Việc tích hợp RAG vào hệ thống đa tác tử giữ vai trò quan trọng trong việc đảm bảo rằng quá trình suy luận nguyên nhân gốc dựa trên tri thức chính xác, bối cảnh háo và phù hợp với hệ thống thực tế. RAG được sử dụng bởi tác tử KREA, vốn là cầu nối giữa tri thức tĩnh (runbooks, tài liệu kỹ thuật, lịch sử sự cố) và dữ liệu động (metrics, logs, alert).

Khi gói ngữ cảnh sơ cấp được AICA cung cấp, KREA tiến hành trích xuất các thành phần thông tin có giá trị tìm kiếm như tên dịch vụ, loại bất thường, mẫu hành vi (temporal pattern) và đặc trưng độ lệch chuẩn (anomaly score). Từ những tín hiệu này, KREA hình thành một truy vấn tìm kiếm ngữ nghĩa để gửi đến hệ thống RAG. Một truy vấn tiêu biểu có dạng:

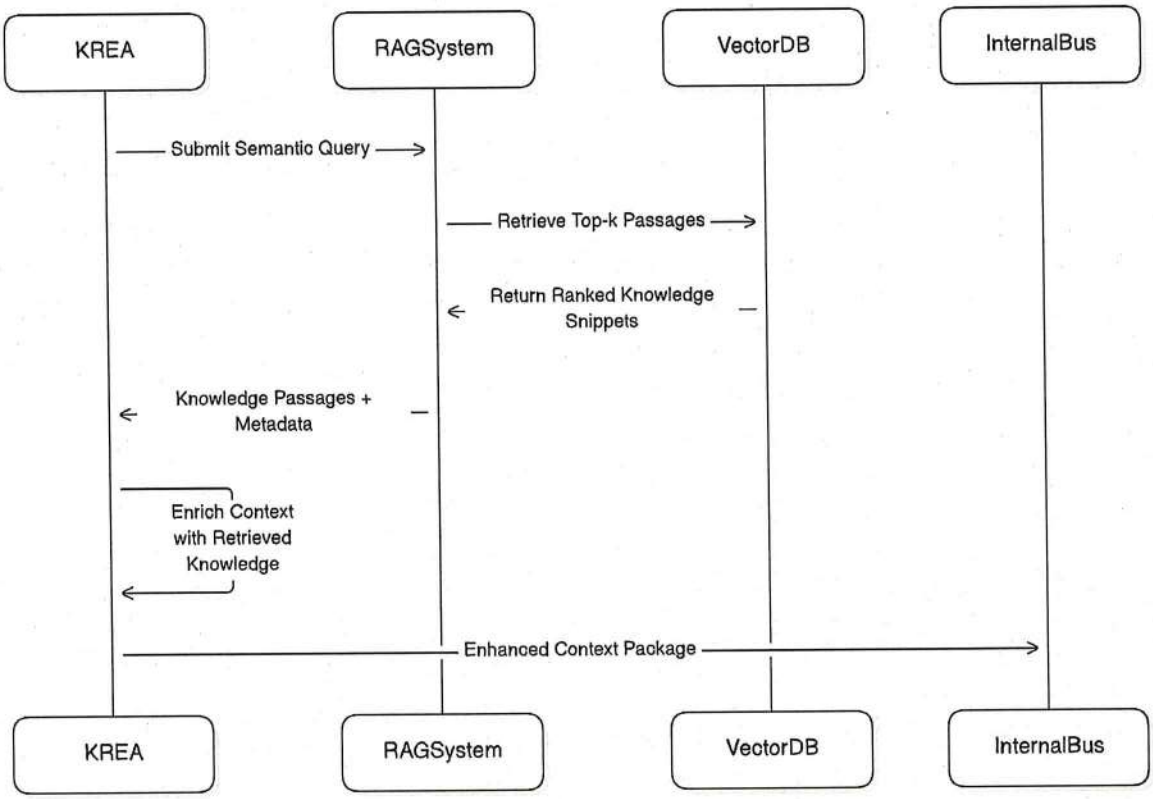
```
{
  "query": "memory spike sustained for 5 minutes in payment-api pod",
  "context_features": {
    "anomaly_score": 0.87,
    "temporal_pattern": "sustained_spike"
  }
}
```

Hệ thống RAG tiếp nhận truy vấn và thực hiện hai nhiệm vụ: (i) Truy xuất tri thức từ cơ sở dữ liệu vector bằng cơ chế "nearest-neighbor seach", và (ii) Khôi phục các đoạn văn, runbook hoặc mẫu sự cố có độ tương đồng cao nhất. Các tài liệu trả về có dạng:

```
{
  "doc_id": "runbook-23",
  "relevance_score": 0.81,
  "snippet": "Restarting the payment-api pod resolves memory leak issues..."
}
```

KREA sau đó thực hiện bước tăng cường ngữ cảnh bằng cách hợp nhất tri thức này vào gói ngữ cảnh, tạo nên gói ngữ cảnh tăng cường. Tác tử này đánh giá mức liên quan giữa dữ liệu thực và tri thức, đảm bảo rằng chỉ có những tài liệu có giá trị thực tiễn cao mới được đưa vào quá trình suy luận của RCARA.

Luồng tích hợp giữa hệ thống đa tác tử và RAG được mô tả bằng sơ đồ trình tự dưới đây:



Hình 2.4 Luồng tích hợp giữa hệ thống đa tác tử và RAG

Cơ chế tích hợp RAG giúp hệ thống vượt qua giới hạn của các mô hình học sâu thuần tuý, vốn thường thiếu tri thức đặc thù của từng môi trường triển khai. Nhờ đó, quá trình phân tích không chỉ dựa trên dữ liệu quan sát hiện tại mà còn có sự củng cố bởi kinh nghiệm vận hành được tích luỹ từ trước. Điều này nâng cao độ chính xác của suy luận và tránh việc LLM đưa ra các giải pháp sai lệch do thiếu tri thức kỹ thuật nền tảng.

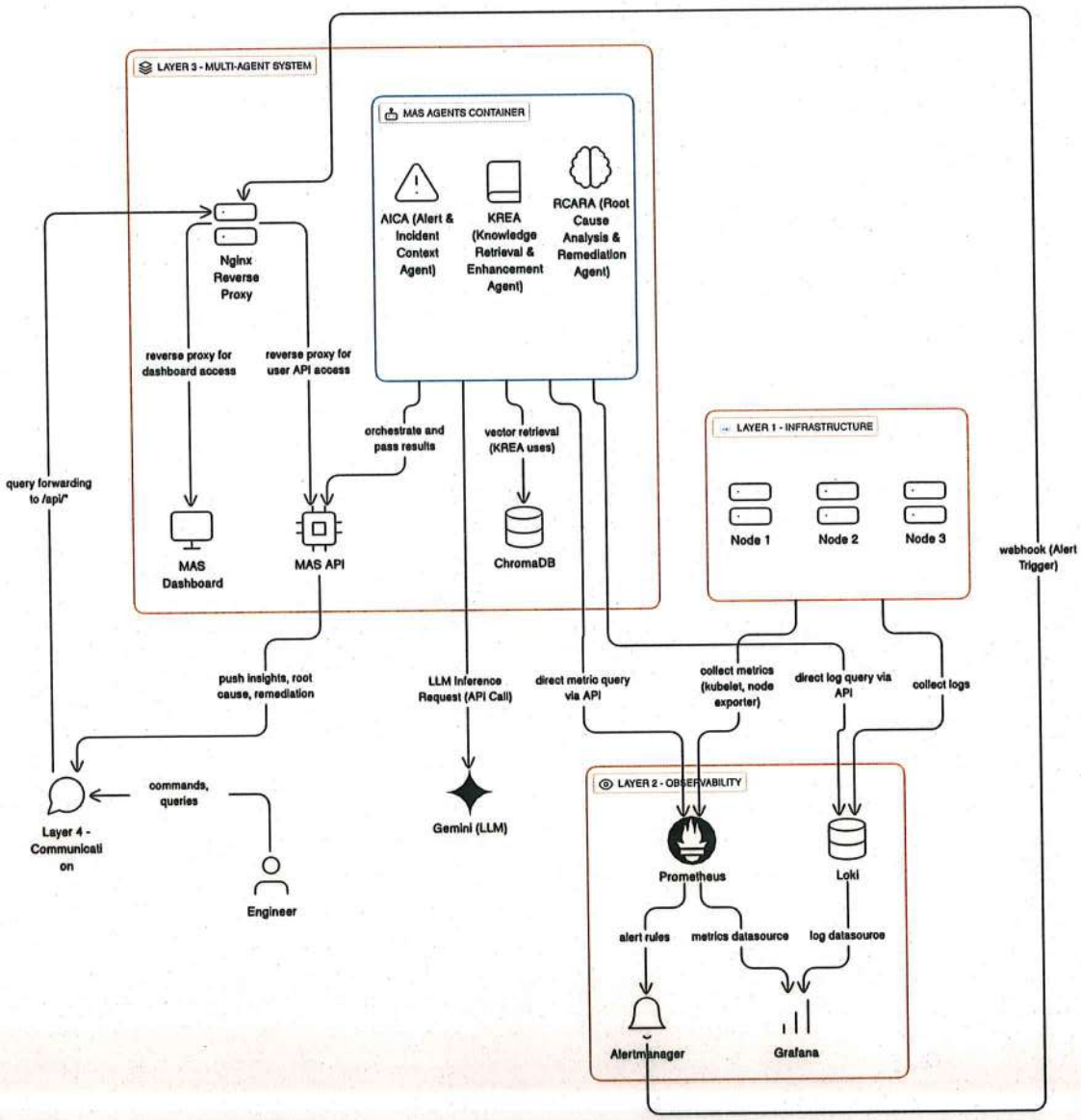
CHƯƠNG 3 - TRIỂN KHAI, THỬ NGHIỆM VÀ ĐÁNH GIÁ

3.1. Môi trường và điều kiện triển khai hệ thống

Việc triển khai hệ thống nguyên mẫu trong phạm vi nghiên cứu yêu cầu một môi trường kiểm thử có tính đại diện cao cho các hệ thống đám mây hiện đại. Mục tiêu là đảm bảo khả năng tái hiện các điều kiện hoạt động thực tế, đồng thời duy trì tính kiểm soát cần thiết để đánh giá hiệu năng của từng thành phần trong hệ thống đa tác tử. Trong các phần nội dung tiếp theo, đề án sẽ mô tả chi tiết các thành phần bao gồm: Môi trường đám mây thử nghiệm, công cụ phát triển và vận hành, dữ liệu cảnh báo mô phỏng và bộ tài liệu kỹ thuật phục vụ mô-đun RAG.

3.1.1. Môi trường thử nghiệm

Kiến trúc triển khai thử nghiệm của hệ thống được thiết kế theo mô hình đa lớp, mô phỏng đầy đủ một môi trường vận hành dịch vụ vi mô trên môi trường điện toán đám mây. Các lớp này bao gồm: (i) Hạ tầng Kubernetes chạy trên Google Kubernetes Engine; (ii) Hệ thống giám sát và thu thập dữ liệu vận hành; (iii) Lớp hệ thống phân tích và đề xuất khắc phục sự cố dựa trên kiến trúc đa tác tử; (iv) Lớp giao tiếp hỗ trợ qua nền tảng Telegram.



Hình 3.1 Mô hình triển khai hệ thống thử nghiệm

Lớp hạ tầng Kubernetes

Lớp hạ tầng được triển khai bằng Google Kubernetes Engine (GKE), cung cấp ba node vật lý ảo hóa với cấu hình đồng nhất. GKE chịu trách nhiệm quản lý hoàn toàn control-plane, trong khi các node worker đảm nhận việc chạy các pod dịch vụ thử nghiệm. Việc lựa chọn GKE nhằm đảm bảo: (i) Khả năng tái lập môi trường thử nghiệm; (ii) Hỗ trợ auto-scaling và quản lý tài nguyên tự động; (iii) khả năng tích hợp liền mạch với bộ công cụ giám sát gốc của hệ sinh thái CNCF. Cấu hình này mô

phỏng sát thực tế hoạt động của một cụm Kubernetes trong sản xuất, nơi các sự cố liên quan đến tài nguyên, mạng, hoặc lỗi dịch vụ có thể được tái hiện chính xác.

Lớp hệ thống giám sát

Lớp quan sát và giám sát được triển khai trực tiếp trong cụm GKE thông qua namespace riêng, bao gồm bốn thành phần chính: (i) **Prometheus** – thu thập metric từ kubelet, node exporter và các ứng dụng; (ii) **Alertmanager** – xử lý và phân tích cảnh báo; cấu hình webhook gửi trực tiếp tới MAS API; (iii) **Grafana** – cung cấp dashboard trực quan hóa cho metric và log; (iv) **Loki** – thu thập log dựa trên mô hình lưu trữ tối ưu, làm nguồn dữ liệu cho MAS.

Lớp hệ thống phân tích và đề xuất khắc phục sự cố

Lớp này triển khai toàn bộ hệ thống mà đề án xây dựng, bao gồm hệ thống đa tác tử, các thành phần giao diện, API hỗ trợ cho hệ thống. Lớp hệ thống đề xuất này được triển khai độc lập để không bị ảnh hưởng bởi các sự cố xảy ra ở lớp hạ tầng.

Lớp giao tiếp hỗ trợ

Lớp giao tiếp hỗ trợ kỹ sư vận hành được triển khai bằng **Telegram Bot API**, kết nối trực tiếp với MAS API thông qua webhook. Lớp giao tiếp này đóng vai trò như **giao diện ChatOps** của hệ thống, góp phần mô phỏng thực tế cách đội ngũ SRE/DevOps nhận thông tin và xử lý sự cố.

Kiến trúc thử nghiệm được tổ chức theo mô hình nhiều lớp, từ hạ tầng Kubernetes cho đến hệ thống phân tích đa tác tử và lớp giao tiếp dựa trên Telegram. Cách thiết kế này đảm bảo khả năng mô phỏng đầy đủ các tình huống gián đoạn xảy ra trong môi trường đám mây, đồng thời duy trì tính linh hoạt, cô lập và khả năng tái lập cho các thí nghiệm của luận văn.

3.1.2. Công cụ và công nghệ triển khai

Các công cụ và framework được lựa chọn để phù hợp với mục tiêu nghiên cứu và khả năng triển khai thực tế. Cụ thể như sau:

- ***Ngôn ngữ lập trình:*** Sử dụng Python 3.10, phục vụ phát triển các Agent và dịch vụ tích hợp.

- ***Framework xây dựng Agent:*** Google Agent Developer Kit (ADK) – Hỗ trợ định nghĩa, điều phối và kết nối nhiều tác tử trong luồng xử lý.

- ***Mô hình ngôn ngữ lớn***: Gemini 1.5 Flash/Pro

- ***RAG Framework***: ChromaDB – Cơ sở dữ liệu vector dùng để lưu trữ embeddings tri thức kỹ thuật và truy xuất theo vector similarity.

- ***ChatOps***: TelegramBot API

3.1.3. Tập dữ liệu cảnh báo và tài liệu kỹ thuật mẫu

Để đánh giá một cách toàn diện khả năng phân tích và đề xuất phương án khắc phục sự cố của hệ thống, cần xây dựng một tập dữ liệu thử nghiệm bao gồm dữ liệu cảnh báo và bộ tri thức kỹ thuật. Các tập dữ liệu này được thiết kế dựa trên những tình huống sự cố phổ biến trong môi trường Kubernetes, đồng thời có tính đại diện đủ cao để phản ánh nhiều dạng bất thường có thể xảy ra trong thực tế.

Tập dữ liệu cảnh báo được thu thập từ hai nguồn chính:

1. Các cảnh báo sinh tự động trong quá trình mô phỏng sự cố, bao gồm các trường hợp như giới hạn tài nguyên bị vượt ngưỡng, container rơi vào trạng thái CrashLoopBackOff, độ trễ mạng tăng bất thường hoặc lỗi HTTP 5xx kéo dài.

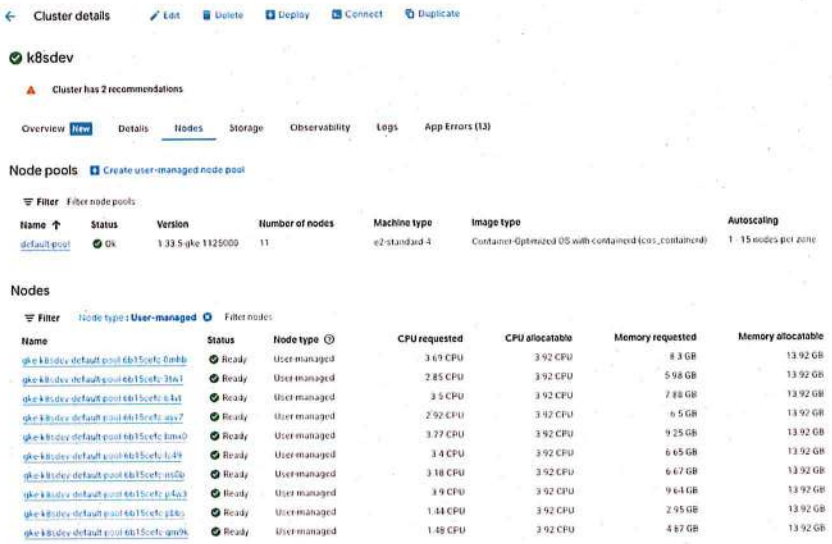
2. Tập cảnh báo mẫu được xây dựng dựa trên các sự cố thực tế. Các cảnh báo này được chuẩn hoá về định dạng JSON để phục vụ thử nghiệm hệ thống.

Bên cạnh dữ liệu cảnh báo, bộ tài liệu kỹ thuật phục vụ RAG bao gồm tài liệu vận hành, hướng dẫn cấu hình, tài liệu xử lý lỗi, cũng như các ghi chú từ thực tiễn vận hành. Các tài liệu này được chia nhỏ theo đơn vị tri thức, chuyển đổi thành embedding và lưu trữ trong cơ sở dữ liệu Vector.

3.2. Kết quả triển khai

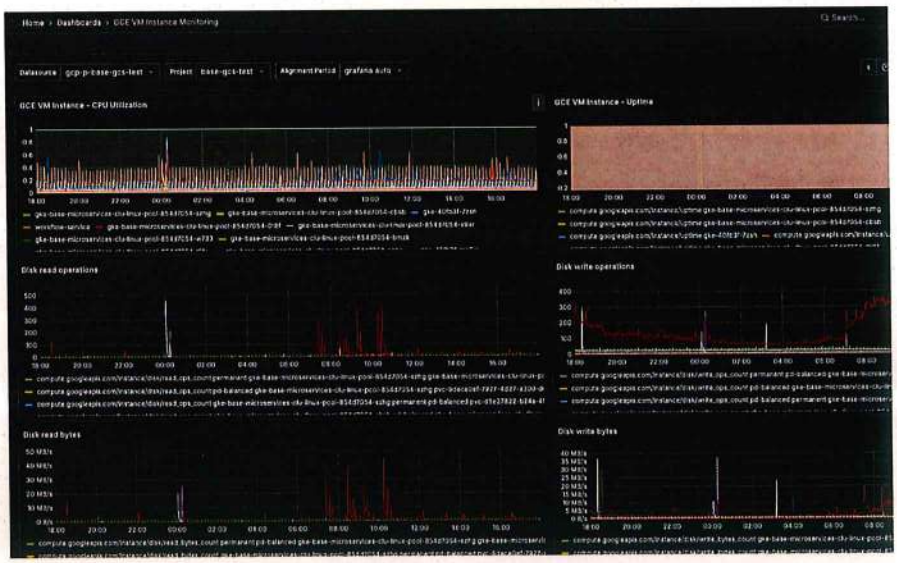
Do giới hạn của đề án tốt nghiệp, các thao tác triển khai sẽ không được trình bày chi tiết. Dưới đây là kết quả sau khi triển khai hệ thống để phục vụ cho các kịch bản thử nghiệm ở sau.

Đầu tiên, một cụm Kubernetes gồm nhiều nút được triển khai trên nền tảng Google Kubernetes Engine (GKE).



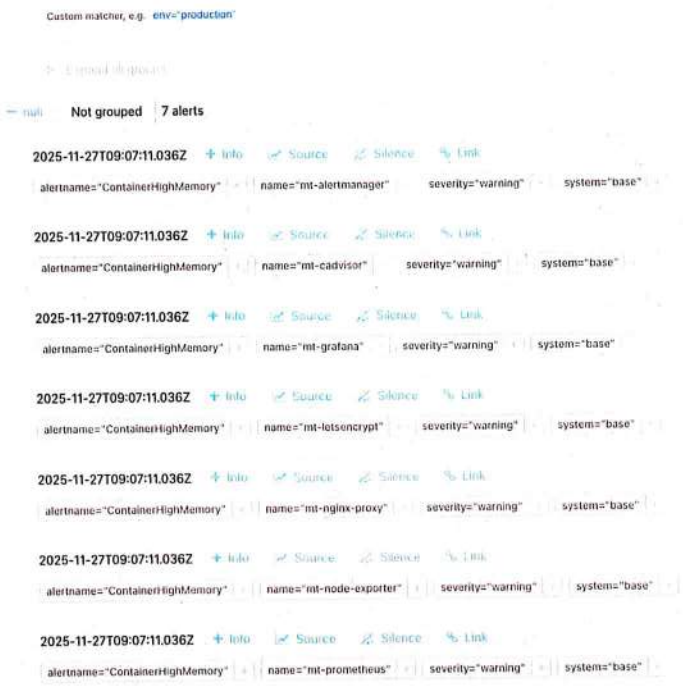
Hình 3.2 Cụm Kubernetes trên khai trên GKE

Tiếp theo, hệ thống giám sát sẽ được triển khai bên trong cụm Kubernetes giúp giám sát các thông số của cụm. Giao diện giám sát trên công cụ Grafana như sau:



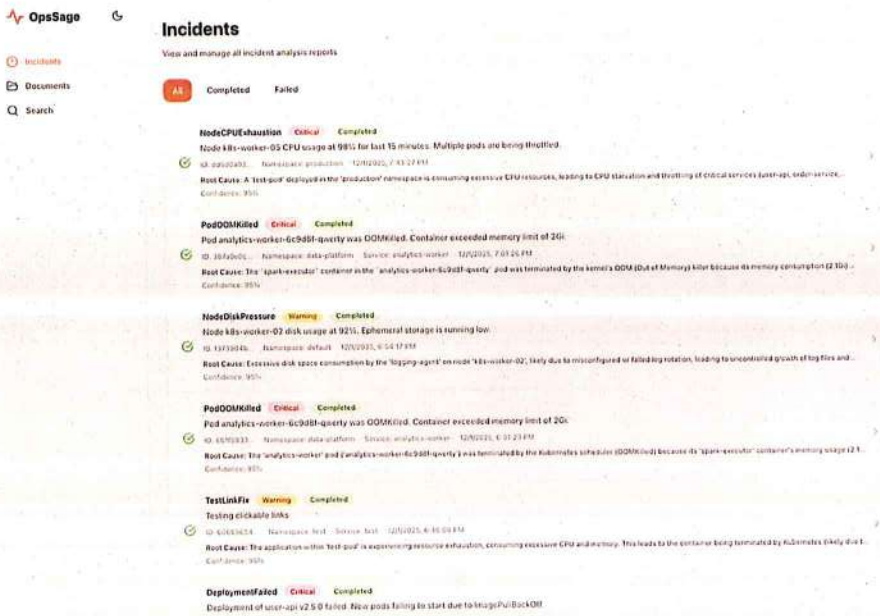
Hình 3.3 Giao diện giám sát trạng thái trên Grafana

Một số cảnh báo xuất hiện trên giao diện công cụ Alertmanager:



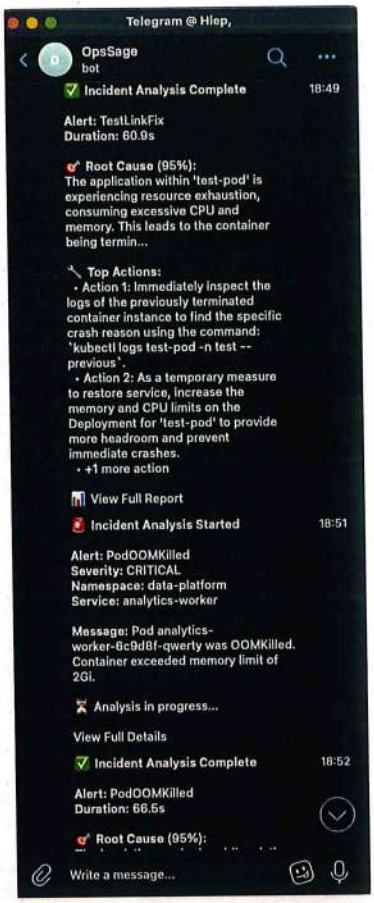
Hình 3.4 Giao diện Alertmanager với một số cảnh báo

Giao diện hệ thống hỗ trợ phân tích và xử lý sự cố:



Hình 3.5 Giao diện hệ thống phân tích và hỗ trợ xử lý sự cố

Giao diện cảnh báo hiển thị trên Telegram:



Hình 3.6 Giao diện cảnh cáo trên Telegram

Sau khi hệ thống được triển khai đầy đủ các thành phần dựa trên mô hình đã mô tả, các phần nội dung tiếp theo sẽ tiến hành mô phỏng, thử nghiệm các sự cố nhằm chứng minh khả năng giải quyết bài toán mà đề án đặt ra.

3.3. Tiêu chí đánh giá

Phần này trình bày các tiêu chí đo lường được sử dụng để kiểm chứng khả năng phân tích – suy luận – đề xuất giải pháp của hệ thống khi đối diện với các kịch bản sự cố mô phỏng.

(1) Độ chính xác phân tích nguyên nhân (RCA Accuracy)

Độ chính xác phân tích nguyên nhân giúp đo lường khả năng mà hệ thống nhận diện đúng nguyên nhân sự cố dựa trên dữ liệu cảnh báo và tri thức tĩnh từ kho tri thức. Cách đánh giá tiêu chí này là so sánh kết quả phân tích của hệ thống với

"ground truth" từ các nguồn ngư: Tài liệu lỗi mô phỏng, các kết quả đã được chuyên gia phân tích sau đó tính tỷ lệ chính xác theo (%) hoặc mức độ tương quan.

(2) Time-to-insight (TTI)

Mean Time to Recovery (MTTR) là chỉ số kinh điển trong quản lý sự cố. Tuy nhiên, vì hệ thống nguyên mẫu chỉ hỗ trợ phân tích và đề xuất, không trực tiếp tự động khôi phục. Do đó, đề án đề xuất sử dụng một chỉ số khác là Time to Insight (TTI) – Là khoảng thời gian từ khi hệ thống tiếp nhận webhook cảnh báo đến khi gửi báo cáo RCA hoàn chỉnh tới các kênh giao tiếp. TTI càng thấp thể hiện hệ thống càng hiệu quả trong việc hỗ trợ kỹ sư rút ngắn thời gian khoanh vùng sự cố. Việc chọn TTI thay vì MTTR là vì hệ thống đóng vai trò quyết định, rút ngắn giai đoạn khoanh vùng – bước tốn thời gian nhất trong vận hành thủ công.

(3) Độ phù hợp của đề xuất

Chỉ số này đánh giá mức độ hợp lý và khả năng áp dụng của các giải pháp được đề xuất. Độ phù hợp này được tính toán dựa trên mức độ chính xác về mặt kỹ thuật, tính khả thi khi triển khai thực tế, mức độ ảnh hưởng tới hệ thống. Chỉ số này sau đó sẽ được tính điểm dựa trên thang đo Likert 1-5. Cụ thể:

- 5 điểm: Đề xuất hoàn toàn chính xác, có thể thực thi ngay để sửa lỗi.
- 3 điểm: Đề xuất có liên quan nhưng cần kỹ sư điều chỉnh thông số.
- 1 điểm: Đề xuất không liên quan hoặc gây rủi ro cho hệ thống.

(4) Chi phí tính toán

Hệ thống sử dụng mô hình ngôn ngữ lớn (LLM), do đó chi phí tính toán là một yếu tố quan trọng khi đánh giá tính khả thi triển khai ở quy mô sản xuất. Chi phí được ước lượng thông qua tổng số token tiêu thụ trên mỗi tác vụ. Chỉ số này giúp đánh giá khả năng mở rộng của hệ thống. Một hệ thống có chi phí vận hành quá cao sẽ khó áp dụng trong môi trường doanh nghiệp có số lượng cảnh báo lớn.

Kết hợp các tiêu chí trên, ta có đủ các công cụ để đo lường, thẩm định hệ thống nguyên mẫu, tạo thành nền tảng để phân tích kết quả thử nghiệm ở các phần tiếp theo và để đối chiếu với mục tiêu nghiên cứu ban đầu của đề án.

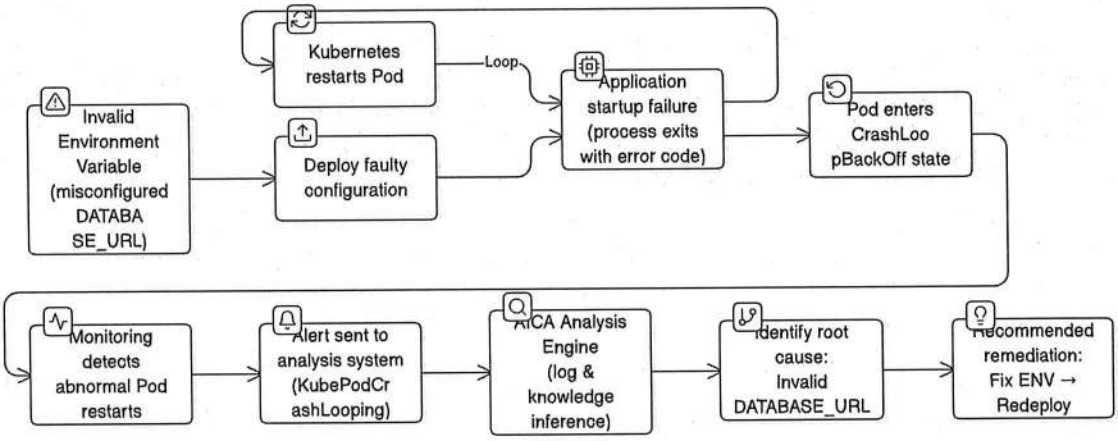
3.4. Thử nghiệm hệ thống

Trong phần nội dung này, đề án sẽ trình bày quá trình thực hiện các kịch bản kiểm thử nhằm đánh giá hiệu quả của hệ thống nguyên mẫu trong việc phân tích và hỗ trợ khắc phục các sự cố gián đoạn trong môi trường đám mây. Các kịch bản được thiết kế có chủ đích theo mức độ tăng dần về độ phức tạp, dựa trên các sự cố phổ biến thuộc nhóm sự cố gián đoạn đã đề cập ở chương đầu tiên. Độ chính xác được tính toán bằng cách đối chiếu kết luận của hệ thống với kịch bản đã biết trước được thiết lập trong quá trình giả lập. Một kết quả được coi là chính xác khi nhận diện đúng: (1) Dịch vụ gốc gây lỗi và (2) Bản chất kỹ thuật của lỗi.

3.4.1. Kịch bản 1 – Sự cố đơn giản: Pod CrashLoopBackOff

Mô tả kịch bản

Kịch bản mô phỏng một dạng sự cố gián đoạn phổ biến trong vận hành hệ thống đám mây sử dụng Kubernetes. Khi Pod liên tục khởi động thất bại và rơi vào trạng thái CrashLoopBackOff. Để tạo giả lập kịch bản này một cách có kiểm soát, đề án chủ động cấu hình sai một biến môi trường quan trọng trong cấu hình triển khai của dịch vụ. Sai lệch này khiến ứng dụng không thể khởi tạo đúng cách, dẫn đến tiến trình bên trong container ngay lập tức thoát ra với mã lỗi. Sau khi bản cấu hình chứa lỗi được triển khai, hệ thống sẽ liên tục cố gắng khởi động lại theo cơ chế tự phục hồi mặc định. Tuy nhiên, mỗi lần khởi chạy đều sẽ thất bại, khiến Pod chuyển sang trạng thái CrashLoopBackOff sau một số vòng lặp. Hệ thống giám sát ghi nhận các tín hiệu bất thường về trạng thái của Pod. Khi điều kiện cảnh báo thoả mãn ngưỡng đã định nghĩa, hệ thống giám sát gửi cảnh báo tới hệ thống phân tích. Tại đây, hệ thống sẽ phân tích nguyên nhân, tiên hành suy luận và đề xuất hướng xử lý phù hợp.



Hình 3.7 Minh hoạ kịch bản sự cố 1

Mục tiêu đánh giá

Kịch bản này nhằm kiểm tra năng lực xử lý của hệ thống trong trường hợp đơn giản, xảy ra trên phạm vi nhỏ và có dấu hiệu rõ ràng. Đây là kịch bản "baseline", dùng để xác nhận rằng pipeline phân tích hoạt động chính xác trước khi thử nghiệm các tình huống phức tạp hơn.

Kết quả kỳ vọng

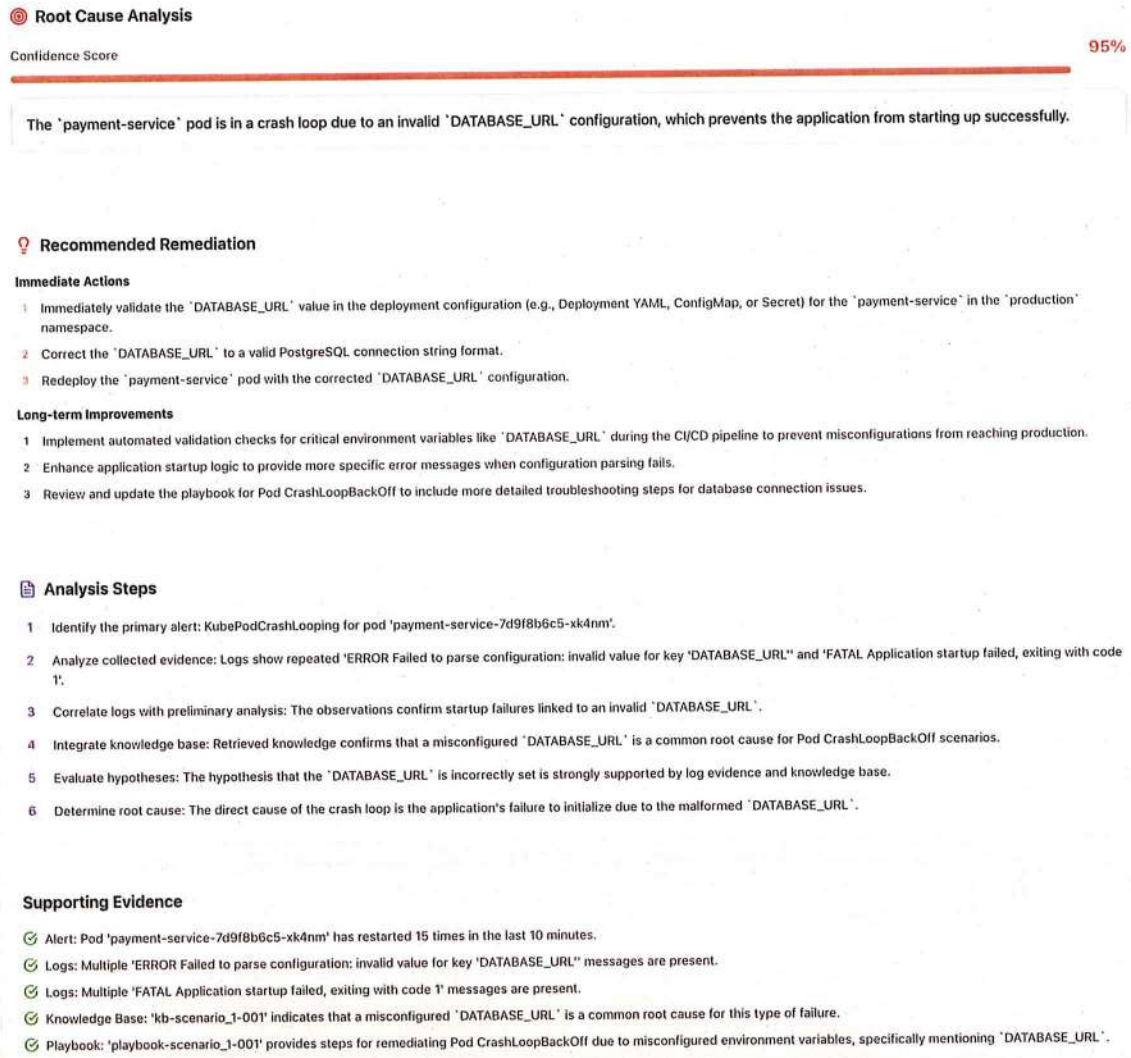
Nhận diện chính xác loại lỗi: *CrashLoopBackOff*. Khoanh vùng nguyên nhân: biến môi trường sai hoặc thiếu giá trị. Đồng thời đưa ra các đề xuất phù hợp.

Kết quả kỳ vọng được mô tả dưới dạng JSON như sau:

```
    "expected_root_cause": "Misconfigured environment variable DATABASE_URL in
the pod deployment configuration",
    "expected_suggestions": [
        "Validate the DATABASE_URL environment variable in the deployment
manifest",
        "Check the ConfigMap or Secret referenced by the deployment for correct
database connection string format",
        ≤≤÷"Redeploy the pod with corrected environment variable configuration"
    ],
```

Kết quả thử nghiệm

Kết quả thử nghiệm hiển thị trên giao diện hệ thống đề xuất:



Hệ thống xác định nguyên nhân gốc là biến môi trường DATABASE_URL được cấu hình sai, khiến ứng dụng không thể khởi động và Pod rơi vào trạng thái *CrashLoopBackOff*.

Kết luận này đạt độ chính xác cao (95%), thể hiện qua chuỗi suy luận: (i) Ở bước phân tích cảnh báo, hệ thống ghi nhận alert *KubePodCrashLooping* đối với pod payment-service; (ii) Log được trích xuất cho thấy các lỗi lặp lại: "`ERROR Failed to parse configuration: invalid value for key 'DATABASE_URL'`" và "`FATAL Application startup failed...`"; (iii) Hệ thống đối chiếu với tri thức trong kho (kb-scenario_1-001) về các trường hợp CrashLoopBackOff phổ biến do biến môi trường sai định dạng;

(iv) Khi đánh giá các giả thuyết, hệ thống loại bỏ các khả năng không phù hợp (CPU, readiness probe, thiếu tài nguyên), tập trung vào nhóm lỗi cấu hình.

Do đây là một kịch bản đơn tuyến, không có cảnh báo nhiễu và bằng chứng log rất rõ ràng, thời gian để hệ thống hội tụ kết luận được rút ngắn đáng kể. TTI thấp *(60,3 giây theo ghi nhận từ hệ thống)* phản ánh tốt vai trò của hệ thống trong việc giảm thời gian khoanh vùng nguyên nhân, hỗ trợ kỹ sư tránh việc kiểm tra thủ công từng khả năng.

Đề xuất của hệ thống cho thấy mức độ phù hợp cao, cả về hành động tức thời lẫn cải thiện dài hạn: (i) Hành động tức thời *(kiểm tra, chỉnh sửa và triển khai lại DATABASE_URL)* bám sát nguyên nhân gốc, có tính khả thi và đúng quy trình vận hành chuẩn của Kubernetes; (ii) Đề xuất dài hạn như bổ sung kiểm tra biến môi trường trong CI/CD, cải thiện thông báo lỗi startup, và cập nhật playbook thể hiện sự hiểu biết toàn diện hơn về vòng đời cấu hình, giúp giảm thiểu tái diễn lỗi.

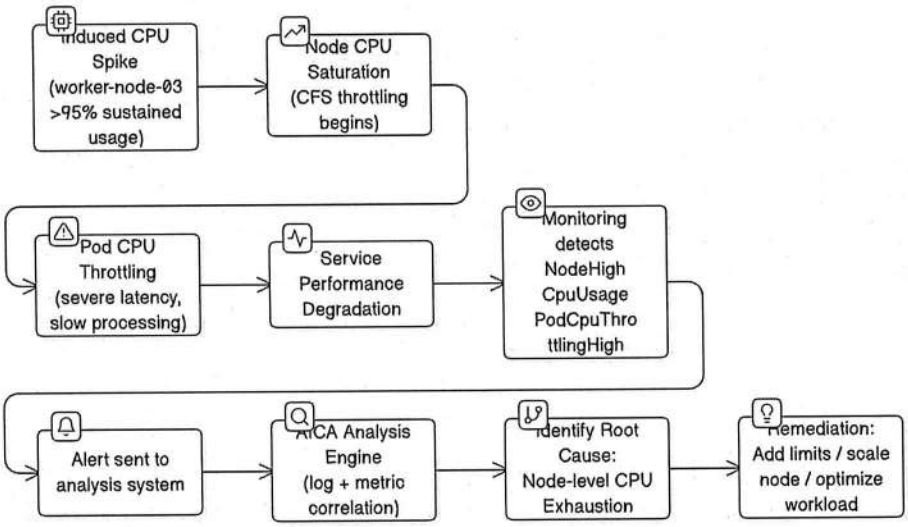
Kịch bản 1 cho thấy hệ thống đạt hiệu quả cao trên cả ba tiêu chí: (i) Độ chính xác: Kết luận khớp hoàn toàn với sự cố mô phỏng, dựa trên bằng chứng rõ ràng từ log và tri thức; (ii) TTI: Quy trình suy luận nhanh, không đi vào nhánh phân tích thừa, tận dụng tốt tín hiệu log để hội tụ; (iii) Đề xuất phù hợp: Cụ thể, khả thi, có tính phòng ngừa và nhất quán với best practices trong khắc phục Pod CrashLoopBackOff.

3.4.2. Kịch bản 2 – Sự cố trung bình: Resource Exhaustion

Mô tả kịch bản

Kịch bản này mô tả một sự cố gián đoạn tiêu biểu liên quan đến hiện tượng cạn kiệt tài nguyên CPU tại mức node trong một hệ thống đám mây vận hành bằng Kubernetes. Hiện tượng này dẫn đến việc nhiều Pod bị throttling và gây suy giảm hiệu năng tổng thể của dịch vụ. Để tạo ra tình huống này, đề án chủ động sinh tải CPU đột biến trên node *worker-node-03*, khiến mức sử dụng CPU duy trì trên 95% trong thời gian dài. Khi node bị quá tải, cơ chế điều phối CPU của Linux bắt đầu giới hạn thời gian CPU cấp cho container, khiến các Pod chạy trên node bị đình trệ và tốc độ xử lý giảm mạnh.

Hệ thống giám sát ghi nhận tín hiệu bất thường, dữ liệu log thể hiện rõ leo thang của sự cố, metrics đi kèm ghi nhận mức độ nghiêm trọng của tình trạng quá tải.



Hình 3.8 Minh hoạ kịch bản sự cố 2

Mục tiêu đánh giá

Kịch bản được thiết kế nhằm kiểm tra năng lực của hệ thống trong các tình huống sự cố đòi hỏi khả năng suy luận quan hệ nhân quả giữa lớp hạ tầng và lớp ứng dụng. Cụ thể, hệ thống cần nhận diện mối quan hệ giữa sự cố tài nguyên của node và hiện tượng suy giảm hiệu năng của ứng dụng. Loại bỏ các thông tin gây nhiễu và tập trung vào việc xác định nguyên nhân gốc rễ.

Kết quả kỳ vọng

Kết quả kỳ vọng là hệ thống nhận diện đúng nhóm lỗi: Resource Exhaustion và xác định được nguyên nhân gốc là tình trạng CPU tại node *work-node-03* bị bão hào, dẫn đến throttling pod và suy giảm hiệu năng ứng dụng. Sau đó đề xuất các hành động khắc phục hợp lý.

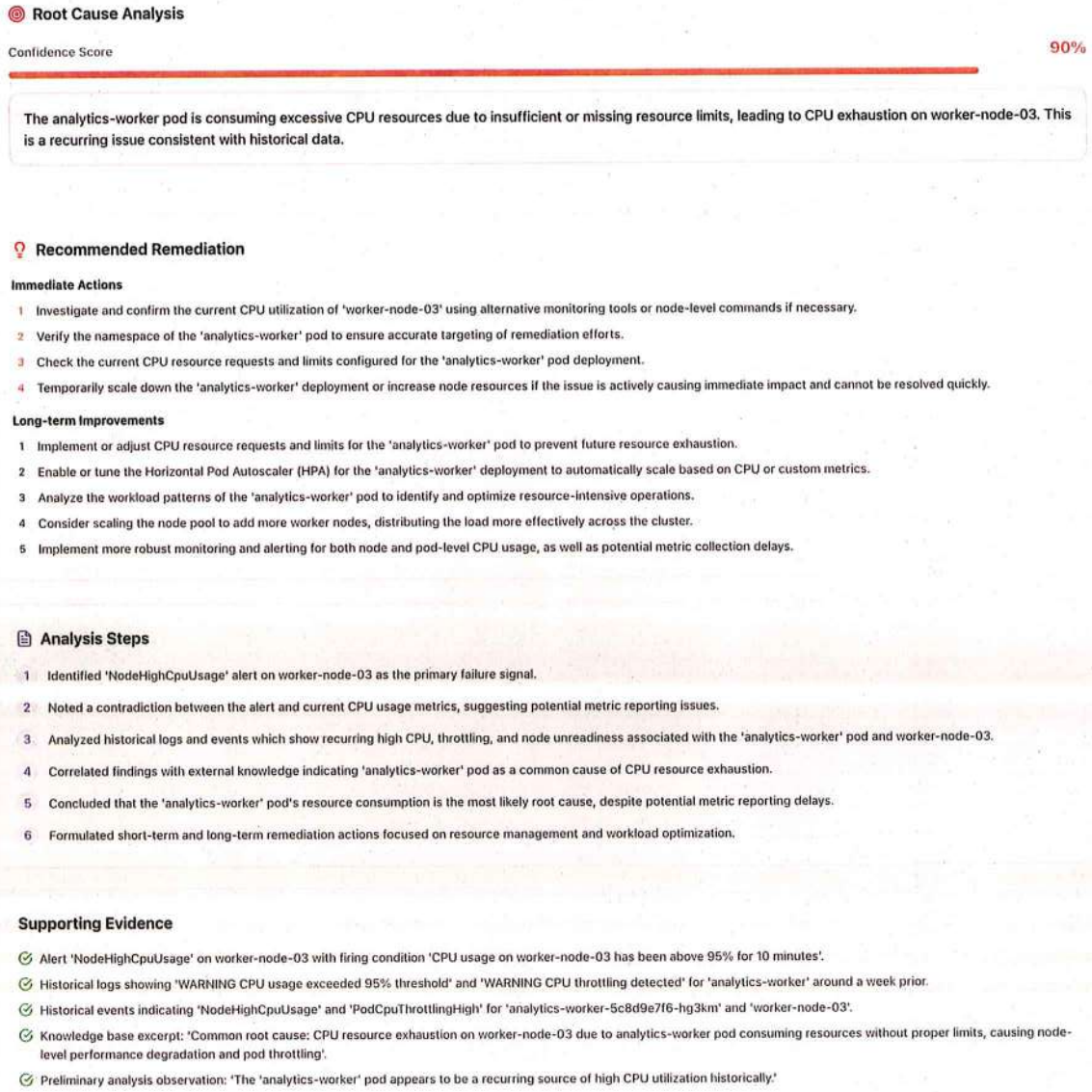
Kết quả kỳ vọng được mô tả dưới dạng JSON như sau:

```
"expected_root_cause": "CPU resource exhaustion on worker-node-03 due to analytics-worker pod consuming resources without proper limits, causing node-level performance degradation and pod throttling",
"expected_suggestions": [
    "Adjust CPU resource requests and limits for analytics-worker pod to prevent resource exhaustion",
```

```
        "Enable or tune Horizontal Pod Autoscaler (HPA) for analytics-worker
deployment to distribute load across multiple pods",
        "Scale the node pool to add additional worker nodes and redistribute pod
workloads",
        "Investigate analytics-worker workload patterns and optimize resource-
intensive operations"
    ],
```

Kết quả thử nghiệm

Kết quả thử nghiệm hiển thị trên giao diện hệ thống đề xuất:



Hệ thống xác định nguyên nhân gốc là pod analytics-worker tiêu thụ CPU vượt mức do thiếu/không có giới hạn tài nguyên, dẫn đến tình trạng CPU exhaustion tại

node *worker-node-03*. Với mức tin cậy 90%, kết luận này tương đối chính xác và phù hợp với bản chất sự cố được mô phỏng.

Quy trình suy luận cho thấy hệ thống đã: (i) Nhận diện chính xác cảnh báo quan trọng nhất: NodeHighCpuUsage, phản ánh mức CPU của node duy trì trên 95%; (ii) Phân tích các bằng chứng lịch sử: log cho thấy nhiều cảnh báo recurring về CPU throttling, node unreadiness và mức tiêu thụ cao từ chính *analytics-worker*; (iii) Đối chiếu với tri thức từ kho: knowledge base mô tả đây là nguyên nhân phổ biến tại cùng node và cùng workload; (iv) Giải thích hợp lý về sự không nhất quán giữa cảnh báo và số liệu metric hiện tại bằng giả định *delayed metrics reporting*, tránh suy luận sai lệch.

Nhìn chung, dù dữ liệu có nhiễu (metrics hiện tại chưa phản ánh tình trạng quá tải), hệ thống vẫn đưa ra kết luận đúng nhờ dựa vào chuỗi bằng chứng lịch sử + cấu trúc tri thức về tác nhân gây tải. Điều này cho thấy khả năng suy luận vượt qua hạn chế của dữ liệu tức thời.

TTI của kịch bản này ở mức trung bình *(74 giây theo ghi nhận của hệ thống)*, do hệ thống phải xử lý tình huống có dữ liệu mâu thuẫn (alert báo CPU cao, nhưng metric tại thời điểm truy vấn chưa cập nhật). Mặc dù không thấp như kịch bản 1 nhưng vẫn đảm bảo trong ngưỡng hiệu quả, đặc biệt khi hệ thống phải nhận diện quan hệ nhân–quả không trực tiếp và xử lý nhiễu dữ liệu.

Các khuyến nghị của hệ thống có mức độ phù hợp cao, bao quát cả biện pháp ngắn hạn và dài hạn, bao gồm: (i) Hành động tức thời như kiểm tra CPU node, xác minh resource limits, tạm scale down hoặc bổ sung tài nguyên đều hợp lý và đúng quy trình vận hành Kubernetes; (ii) Giải pháp dài hạn (thiết lập requests/limits chuẩn, bật HPA, tối ưu workload analytics, mở rộng node pool) thể hiện hiểu biết sâu về nguyên nhân hệ thống và hướng phát triển bền vững; (iii) Các đề xuất tập trung vào resource hygiene – một phương pháp tiếp cận chuẩn trong SRE để giảm thiểu các sự cố resource exhaustion tái diễn.

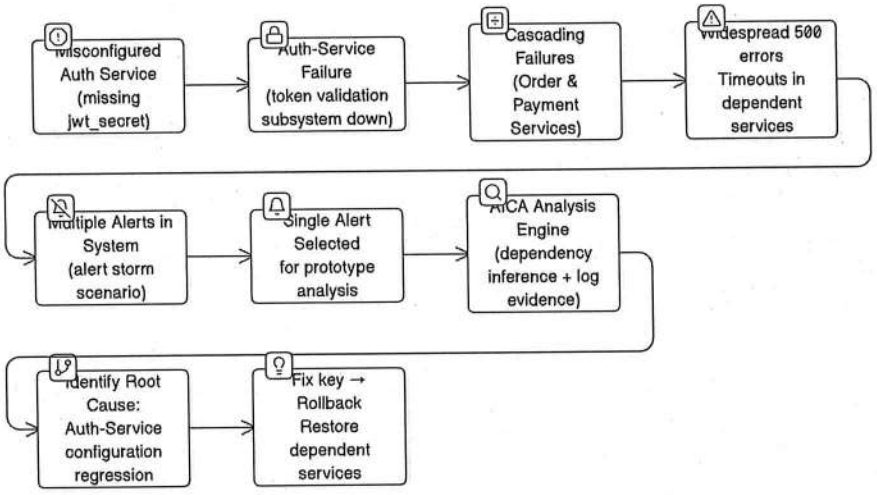
Kịch bản 2 phản ánh năng lực xử lý sự cố mức độ trung bình của hệ thống: (i) Độ chính xác cao, dù có dữ liệu nhiễu; hệ thống dựa vào bằng chứng lịch sử và tri thức để suy luận đúng nguyên nhân gốc. (ii) TTI trung bình nhưng hợp lý do phải xử lý xung đột dữ liệu; hệ thống vẫn hội tụ insight hiệu quả. (iii) Đề xuất phù hợp, bao quát cả cấp độ ứng dụng và hạ tầng, thể hiện hiểu biết toàn diện về quản lý tài nguyên trong Kubernetes.

3.4.3. Kịch bản 3 – Sự cố phức tạp: Microservice Dependency Failure

Mô tả kịch bản

Trong kịch bản này, đề án chủ động tạo lỗi bằng cách cấu hình sai Auth Service – Cụ thể là thiếu khoá *jwt_secrets* trong cấu hình. Lỗi này khiến Auth Service không thể hoạt động bình thường. Vì Auth Service đóng vai trò trung tâm trong luồng xác thực người dùng, lỗi của nó nhanh chóng lan truyền sang các dịch vụ phụ thuộc. Order Service không thể xác minh token hợp lệ, trong khi Payment Service liên tục gặp lỗi khi gọi Auth Service.

Mặc dù trong tình huống này, một "storm-alert" sẽ xuất hiện trên hệ thống giám sát, khi nhiều thành phần gặp sự cố cùng lúc. Tuy nhiên, với phạm vi hệ thống nguyên mẫu, hệ thống đề xuất sẽ chỉ tiếp nhận một trong số các cảnh báo để tiến hành phân tích. Từ cảnh báo này, hệ thống cần tái dựng bối cảnh sự cố dựa trên tri thức phụ thuộc giữa các dịch vụ, kết hợp với các dữ liệu truy vấn được để tìm được nguyên nhân gốc.



Hình 3.9 Minh hoạ kịch bản sự cố 3

Mục tiêu đánh giá

Kịch bản này được thiết kết nhằm kiểm tra năng lực suy luận của hệ thống trong bối cảnh xảy ra hiện tượng lan truyền lỗi – Dạng sự cố gián đoạn có độ phức tạp cao và thường gặp trong kiến trúc vi dịch vụ.

Kết quả kỳ vọng

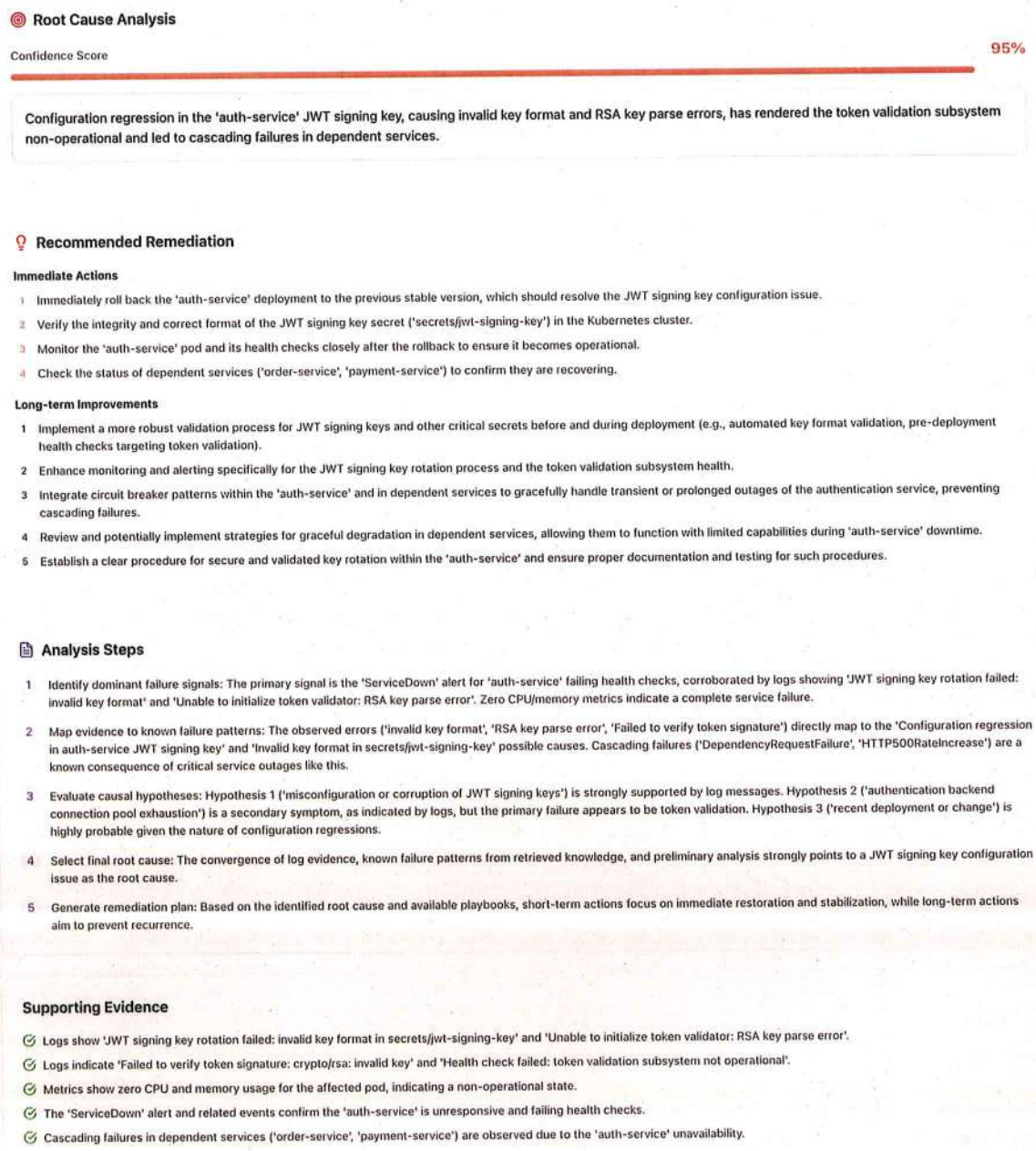
Hệ thống được kỳ vọng sẽ nhận diện được tính chất lan truyền của sự cố, thay thì coi đây là lỗi riêng lẻ của Order Service hoặc Payment Service. Đồng thời, suy luận được chính xác nguyên nhân gốc là lỗi cấu hình tại Auth Service (Thiếu *jwt_secret*), sau đó đề xuất các giải pháp khắc phục phù hợp.

Kết quả kì vọng này được mô tả dưới dạng JSON như sau:

```
    "expected_root_cause": "Configuration regression in auth-service JWT signing key after recent deployment, causing token signature validation failures and cascading authentication errors across dependent services",
    "expected_suggestions": [
        "Roll back auth-service to previous stable version (before JWT key rotation change)",
        "Apply circuit breaker pattern to prevent cascading failures when auth-service is degraded",
        "Improve release validation process with pre-deployment health checks for critical configuration changes",
        "Implement graceful degradation in dependent services to handle auth-service outages",
        "Add monitoring and alerting for JWT key rotation and validation subsystem health"
    ],
```

Kết quả thử nghiệm

Kết quả thử nghiệm hiển thị trên giao diện hệ thống đề xuất:



Root Cause Analysis

Confidence Score 95%

Configuration regression in the 'auth-service' JWT signing key, causing invalid key format and RSA key parse errors, has rendered the token validation subsystem non-operational and led to cascading failures in dependent services.

Recommended Remediation

Immediate Actions

1. Immediately roll back the 'auth-service' deployment to the previous stable version, which should resolve the JWT signing key configuration issue.
2. Verify the integrity and correct format of the JWT signing key secret ('secrets/jwt-signing-key') in the Kubernetes cluster.
3. Monitor the 'auth-service' pod and its health checks closely after the rollback to ensure it becomes operational.
4. Check the status of dependent services ('order-service', 'payment-service') to confirm they are recovering.

Long-term Improvements

1. Implement a more robust validation process for JWT signing keys and other critical secrets before and during deployment (e.g., automated key format validation, pre-deployment health checks targeting token validation).
2. Enhance monitoring and alerting specifically for the JWT signing key rotation process and the token validation subsystem health.
3. Integrate circuit breaker patterns within the 'auth-service' and in dependent services to gracefully handle transient or prolonged outages of the authentication service, preventing cascading failures.
4. Review and potentially implement strategies for graceful degradation in dependent services, allowing them to function with limited capabilities during 'auth-service' downtime.
5. Establish a clear procedure for secure and validated key rotation within the 'auth-service' and ensure proper documentation and testing for such procedures.

Analysis Steps

1. Identify dominant failure signals: The primary signal is the 'ServiceDown' alert for 'auth-service' failing health checks, corroborated by logs showing 'JWT signing key rotation failed: invalid key format' and 'Unable to initialize token validator: RSA key parse error'. Zero CPU/memory metrics indicate a complete service failure.
2. Map evidence to known failure patterns: The observed errors ('invalid key format', 'RSA key parse error', 'Failed to verify token signature') directly map to the 'Configuration regression in auth-service JWT signing key' and 'Invalid key format in secrets/jwt-signing-key' possible causes. Cascading failures ('DependencyRequestFailure', 'HTTP500RateIncrease') are a known consequence of critical service outages like this.
3. Evaluate causal hypotheses: Hypothesis 1 ('misconfiguration or corruption of JWT signing keys') is strongly supported by log messages. Hypothesis 2 ('authentication backend connection pool exhaustion') is a secondary symptom, as indicated by logs, but the primary failure appears to be token validation. Hypothesis 3 ('recent deployment or change') is highly probable given the nature of configuration regressions.
4. Select final root cause: The convergence of log evidence, known failure patterns from retrieved knowledge, and preliminary analysis strongly points to a JWT signing key configuration issue as the root cause.
5. Generate remediation plan: Based on the identified root cause and available playbooks, short-term actions focus on immediate restoration and stabilization, while long-term actions aim to prevent recurrence.

Supporting Evidence

- Logs show 'JWT signing key rotation failed: invalid key format in secrets/jwt-signing-key' and 'Unable to initialize token validator: RSA key parse error'.
- Logs indicate 'Failed to verify token signature: crypto/rsa: invalid key' and 'Health check failed: token validation subsystem not operational'.
- Metrics show zero CPU and memory usage for the affected pod, indicating a non-operational state.
- The 'ServiceDown' alert and related events confirm the 'auth-service' is unresponsive and failing health checks.
- Cascading failures in dependent services ('order-service', 'payment-service') are observed due to the 'auth-service' unavailability.

Hệ thống xác định nguyên nhân gốc là lỗi cấu hình khóa ký JWT của auth-service, dẫn đến lỗi *RSA key parse error*, khiến toàn bộ subsystem xác thực token bị vô hiệu. Đây là nhận định chính xác và phù hợp hoàn toàn với sự cố mô phỏng, thể hiện trong mức tin cậy 95%.

Quy trình suy luận bao gồm: (i) Nhận diện tín hiệu chính: ServiceDown cho *auth-service*, cho thấy dịch vụ ngừng hoạt động hoàn toàn; (ii) Phân tích log: lặp lại các lỗi *invalid key format*, *RSA key parse error*, *token validator not operational*. Những lỗi này tương ứng trực tiếp với việc thiếu hoặc sai định dạng jwt-signing-key. (iii) Kiểm chứng qua metrics: CPU/memory bằng 0, cho thấy dịch vụ không vận hành; (iv) So sánh với tri thức trong kho: hệ thống xác định đây là mẫu lỗi đặc trưng của configuration regression tại tầng xác thực; (v) Suy luận về ảnh hưởng lan truyền: các lỗi HTTP 500 tại order-service và dependency failures tại payment-service củng cố kết luận sự cố xuất phát từ auth-service chứ không phải các dịch vụ downstream.

Như vậy, hệ thống đã sử dụng bằng chứng log trực tiếp, tương quan hành vi giữa dịch vụ phụ thuộc, và tri thức đã biết để đưa ra kết luận chính xác, không bị nhiễu bởi hậu quả của sự cố lan truyền.

TTI của kịch bản này ở mức tốt, mặc dù đây là sự cố phức tạp có tính lan truyền. Hệ thống nhanh chóng hội tụ vào nguyên nhân gốc nhờ: (i) Xác định ServiceDown của auth-service là tín hiệu ưu tiên cao nhất; (ii) Phân tích log ngay lập tức cho thấy lỗi định dạng khóa ký JWT – dạng lỗi có tính phân biệt mạnh, giúp loại bỏ các giả thuyết khác như backend quá tải hay lỗi mạng nội bộ. (iii) Ghi nhận trạng thái CPU/memory bằng 0, củng cố nhận định dịch vụ không thể khởi tạo được; (iv) Xem xét tác động lan truyền và kết luận rằng order-service và payment-service chỉ là triệu chứng thứ cấp, không phải nguyên nhân; (v) Mặc dù chỉ có một cảnh báo đầu vào được hệ thống xử lý, hệ thống vẫn tái dựng đúng bối cảnh đầy đủ của sự cố—đây là dấu hiệu cho thấy năng lực suy luận dựa trên tri thức phụ thuộc (service dependency graph) hoạt động hiệu quả.

Các đề xuất mà hệ thống đưa ra được đánh giá là phù hợp và có chiều sâu, bao gồm cả việc khắc phục trước mắt và cải thiện lâu dài:

- Hành động tức thời: rollback phiên bản auth-service, kiểm tra lại secret jwt-signing-key, theo dõi health check và trạng thái các dịch vụ phụ thuộc.

- Giải pháp dài hạn: bổ sung kiểm tra định dạng khóa, tăng cường giám sát chu trình key rotation, áp dụng circuit breaker cho các dịch vụ downstream, và thiết kế cơ chế graceful degradation.

Những đề xuất này không chỉ khắc phục lỗi cấu hình mà còn giúp giảm thiểu khả năng tái diễn sự cố lan truyền trong tương lai. Điều này cho thấy hệ thống hiểu rõ bản chất sự cố: *lỗi cấu hình tại dịch vụ trung tâm có thể kéo theo sụp đổ dây chuyền nếu không có bảo vệ kiến trúc.*

Kịch bản 3 thể hiện đầy đủ khả năng của hệ thống khi đối mặt với sự cố phức tạp: (i) Độ chính xác rất cao (95%), nhờ sử dụng log làm bằng chứng cốt lõi và nhận diện đúng mối quan hệ phụ thuộc giữa các dịch vụ; (ii) TTI ở tốt, bất chấp tính chất cascaded failure, do hệ thống nhanh chóng khoanh vùng auth-service là nguồn gốc lỗi. (iii) Đề xuất hợp lý, khả thi, và có tầm nhìn dài hạn, nhấn mạnh tăng cường an toàn cấu hình và thiết kế chống lan truyền lỗi.

3.5. Đánh giá, nhận xét

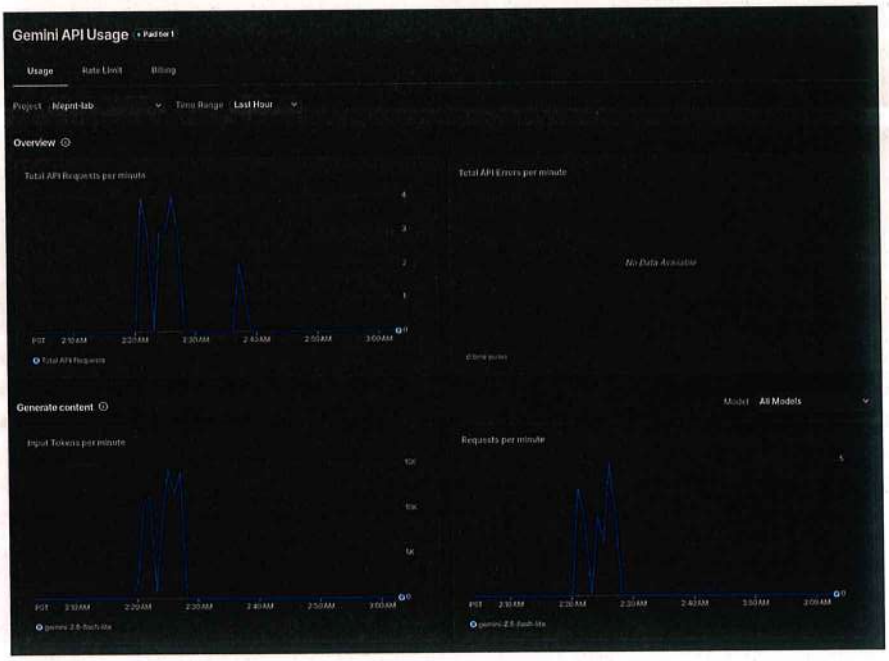
Kết quả thử nghiệm trên ba kịch bản cho thấy hệ thống hỗ trợ phân tích và xử lý sự cố dựa trên kiến trúc đa tác tử kết hợp mô hình ngôn ngữ lớn đạt hiệu quả cao trong việc nhận diện nguyên nhân gốc rễ, đánh giá bối cảnh và đề xuất biện pháp khắc phục. Điểm nổi bật quan trọng là hệ thống không chỉ tái hiện khả năng phân tích của chuyên gia mà còn mở rộng thêm năng lực suy luận đa chiều dựa trên dữ liệu logs, metrics, alerts và tri thức nền. Điều này đánh dấu sự cải thiện rõ rệt so với phương pháp truyền thống vốn chủ yếu dựa trên thao tác thủ công, dễ sai sót và đòi hỏi sự can thiệp sâu của con người.

Về độ chính xác, cả ba kịch bản đều được hệ thống xác định đúng nhóm sự cố và nguyên nhân gốc rễ, thể hiện khả năng hội tụ thông tin và suy luận tiến–lùi dựa trên bằng chứng. Ở kịch bản 1, hệ thống nhận diện chính xác lỗi cấu hình DATABASE_URL, dựa trên sự kết hợp giữa chuỗi thông báo lỗi, trạng thái CrashLoopBackOff và tri thức trong kho sự cố. Trong kịch bản 2, hệ thống xử lý được tình huống phức tạp hơn với nhiều tín hiệu cảnh báo cùng lúc, cho thấy năng lực phân tích quan hệ giữa tầng hạ tầng (node) và tầng ứng dụng (pod). Ở kịch bản 3,

hệ thống suy luận được cấu hình khoá JWT bị lỗi định dạng – một loại sự cố có tính chất sâu, khó phát hiện nếu chỉ dựa trên biểu hiện bề mặt. Độ tin cậy của các kết luận được lượng hóa thông qua điểm *Confidence Score*, thể hiện tính minh bạch và khả năng tự giải thích của hệ thống.

Time-to-insight được cải thiện rõ rệt so với quy trình truyền thống. Thay vì phải trải qua nhiều bước tổng hợp logs, kiểm tra cấu hình, đối chiếu playbook và thử nghiệm giả định, hệ thống đưa ra kết luận trong thời gian ngắn bằng cách tự động hoá toàn bộ chuỗi phân tích: nhận diện tín hiệu chính, đối chiếu với bối cảnh lịch sử, loại trừ các giả thuyết không phù hợp và đưa ra nguyên nhân có độ khả tín cao nhất. Điều này đặc biệt quan trọng trong môi trường đám mây, nơi các sự cố lan nhanh và gây ảnh hưởng dây chuyền.

Độ phù hợp của đề xuất khắc phục cũng cho thấy tính thực tiễn của hệ thống. Ở cả ba kịch bản, các đề xuất đều có cấu trúc phân tầng: hành động tức thời nhằm ổn định hệ thống, tiếp đến là biện pháp dài hạn nhằm ngăn ngừa tái diễn. Việc hệ thống tự động trích xuất hướng dẫn từ playbook và tri thức chuyên môn cho thấy khả năng hỗ trợ thao tác vận hành một cách toàn diện, thay vì chỉ dừng lại ở việc trả về cảnh báo.



Hình 3.10 Mức độ tiêu thụ token của thử nghiệm

Dựa trên biểu đồ Hình 3.10, thống kê cho thấy mức tiêu thụ tài nguyên của hệ thống rất tối ưu. Đỉnh tiêu thụ token (Khoảng vài nghìn đến 15.000 input tokens mỗi phút) chỉ xuất hiện tại các thời điểm hệ thống thực hiện suy luận phức tạp hoặc phân tích thông tin log có độ dài rất lớn. Điều này phản ánh hai đặc điểm quan trọng.

Thứ nhất, hệ thống đạt được hiệu quả phân tích cao mà không phụ thuộc vào việc tiêu thụ lượng lớn tài nguyên mô hình. Các đoạn tăng đột biến token trùng với thời điểm hệ thống thực hiện suy luận sâu (như phân tích chuỗi logs dài hoặc xây dựng quan hệ giữa các tín hiệu ở kịch bản 2), cho thấy hệ thống chỉ sử dụng token khi thực sự cần thiết. Thứ hai, khả năng duy trì mức tiêu thụ thấp đồng nghĩa với chi phí vận hành mô hình LLM được tối ưu, giúp hệ thống phù hợp cho triển khai thực tế trong môi trường production – nơi vấn đề chi phí inference là yếu tố quan trọng.

Tổng hợp toàn bộ kết quả cho thấy hệ thống phân tích và hỗ trợ khắc phục sự cố đã chứng minh được giá trị thực tiễn trong việc tăng cường năng lực giám sát và phân tích sự cố trong môi trường điện toán đám mây. Thay vì chỉ phản ứng bị động như mô hình giám sát truyền thống, hệ thống có khả năng hiểu ngữ cảnh, suy luận nguyên nhân, hỗ trợ quyết định và đề xuất biện pháp khắc phục có hệ thống. Điều này không chỉ rút ngắn thời gian xử lý sự cố mà còn nâng cao tính dự đoán và tính ổn định của hạ tầng. Ý nghĩa của đề tài vì vậy không chỉ nằm ở việc cải thiện hiệu quả vận hành của của các hệ thống đám mây, mà còn mở ra cách tiếp cận mới cho lĩnh vực quản trị, xử lý sự cố, nơi tri thức tự động và suy luận thông minh đóng vai trò trung tâm.

KẾT LUẬN VÀ KIẾN NGHỊ

Đề án đã nghiên cứu và triển khai mô hình hỗ trợ phân tích, xử lý sự cố gián đoạn trong môi trường điện toán đám mây dựa trên kiến trúc đa tác tử kết hợp mô hình ngôn ngữ lớn và cơ chế truy xuất tri thức. Kết quả cho thấy hệ thống nguyên mẫu mà đề án xây dựng có khả năng tổng hợp dữ liệu sự cố, phân tích nguyên nhân và đưa ra gợi ý khắc phục ở mức độ chính xác tương đối cao. Việc sử dụng mô hình ngôn ngữ lớn kết hợp truy xuất tri thức giúp cải thiện đáng kể năng lực suy luận ngữ nghĩa, đặc biệt trong các kịch bản liên quan đến log phức tạp hoặc lỗi cấu hình. Bên cạnh đó, kiến trúc đa tác tử cho thấy tính linh hoạt và khả năng mở rộng tốt, cho phép phân tách nhiệm vụ, giảm tải cho một tác tử duy nhất và hỗ trợ việc bổ sung chức năng trong tương lai. Các thử nghiệm với các kịch bản giả lập sự cố phổ biến trong môi trường Kubernetes cho thấy thời gian phân tích và đề xuất hướng xử lý được rút ngắn so với thao tác thủ công, đồng thời mô hình tương tác qua ChatOps giúp nâng cao hiệu quả hỗ trợ kỹ sư vận hành.

Tuy nhiên, hệ thống vẫn tồn tại những hạn chế nhất định. Hiệu quả phân tích phụ thuộc vào chất lượng dữ liệu trong kho tri thức; mô hình vẫn có khả năng tạo ra kết luận chưa chính xác trong một số tình huống đặc thù. Phạm vi thử nghiệm còn hạn chế và chưa phản ánh toàn bộ độ phức tạp của dữ liệu vận hành thực tế. Ngoài ra, hệ thống mới dừng ở mức hỗ trợ ra quyết định mà chưa triển khai cơ chế tự động hóa khép kín trong quá trình khắc phục sự cố.

Từ kết quả đạt được, đề án kiến nghị một số hướng phát triển tiếp theo. Trước hết, cần mở rộng và hoàn thiện kho tri thức để nâng cao chất lượng truy xuất và hạn chế sai lệch trong phân tích. Tiếp theo, việc triển khai thử nghiệm trong môi trường thực tế với khối lượng dữ liệu lớn hơn sẽ giúp đánh giá rõ hơn về độ tin cậy và khả năng mở rộng của hệ thống. Ngoài ra, việc bổ sung cơ chế học liên tục và tự động hóa từng phần trong quy trình khắc phục có thể giúp hệ thống tiến gần hơn tới mục tiêu vận hành tự động thông minh. Những định hướng này góp phần hoàn thiện mô hình và mở rộng phạm vi ứng dụng của hệ thống trong tương lai.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] ITIC, "ITIC 2024 Hourly Cost of Downtime Report Part 1," 2024. Accessed: [Insert Date Accessed if known, e.g., Jul. 23, 2025]. [Online]. Available: https://itic-corp.com/itic-2024-hourly-cost-of-downtime-report/

[2] U. Madubueze, "The Real Cost of Cloud Downtime: Prevention Strategies for Maximum Uptime," LinkedIn, accessed: [Insert Date Accessed if known, e.g., Jul. 23, 2025]. [Online]. Available: https://www.linkedin.com/pulse/real-cost-cloud-downtime-prevention-strategies-uptime-madubueze--esaof

[3] Forbes Technology Council, "The True Cost Of Downtime (And How To Avoid It)," Forbes, Apr. 10, 2024. Accessed: [Insert Date Accessed if known, e.g., Jul. 23, 2025]. [Online]. Available: https://www.forbes.com/councils/forbestechcouncil/2024/04/10/the-true-cost-of-downtime-and-how-to-avoid-it/

[4] R. Feiner, "OpenAI says ChatGPT has recovered after hours-long outage," CNBC, Dec. 11, 2024. Accessed: [Insert Date Accessed if known, e.g., Jul. 23, 2025]. [Online]. Available: https://www.cnbc.com/2024/12/11/openais-chatgpt-suffers-outage-and-says-its-working-on-a-fix.html

[5] I. Lomas, "OpenAI fixes the issue that caused ChatGPT outage for several hours," TechCrunch, Jun. 4, 2024. Accessed: [Insert Date Accessed if known, e.g., Jul. 23, 2025]. [Online]. Available: https://techcrunch.com/2024/06/04/chatgpt-is-down-for-several-users-openai-is-working-on-a-fix/

[6] R. Feiner, "Google suffers cloud outage, disruptions for many internet services," CNBC, Jun. 12, 2025. Accessed: [Insert Date Accessed if known, e.g., Jul. 23, 2025]. [Online]. Available: https://www.cnbc.com/2025/06/12/google-cloud-and-other-internet-services-are-reporting-outages.html

[7] J. Duckett, "A major Google Cloud outage took down swathes of the internet and even Workspace apps - here's what we know," TechRadar Pro, accessed: [Insert

Date Accessed if known, e.g., Jul. 23, 2025]. [Online]. Available: https://www.techradar.com/pro/a-major-google-cloud-outage-took-down-swathes-of-the-internet-and-even-workspace-apps-heres-what-we-know

[8] Armbrust, M. et al. (2010). A View of Cloud Computing. Communications of the ACM, 53(4), 50–58.

[9] Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. NIST Special Publication 800-145.

[10] Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2021). Distributed Systems: Concepts and Design (6th ed.). Pearson.

[11] Dean, J., & Barroso, L. A. (2013). *The Tail at Scale*. Communications of the ACM, 56(2), 74–80.

[12] Barrett, B., Burns, B., & Oppenheimer, D. (2016). *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media.

[13] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2019). *Kubernetes: Up and Running*. O'Reilly Media.

[14] Duckett, J. (2025). *A Major Google Cloud Outage Took Down Swathes of the Internet*. TechRadar Pro.

[15] Feiner, R. (2024). *OpenAI Says ChatGPT Has Recovered After Hours-long Outage*. CNBC.

[16] Gartner. (2020). *Market Guide for AIOps Platforms*. Gartner Research.

[17] ITIC. (2024). *2024 Hourly Cost of Downtime Report*. ITIC Corporation.