Bài tập nhóm này nhằm giúp sinh viên:

- Hiểu bản chất kiến trúc hệ thống phân tán trong các tình huống thực tế, đặc biệt là các hệ thống tài chính AI cần xử lý dữ liệu thời gian thực, thu thập dữ liệu quy mô lớn, và đảm bảo bảo mật khi mở rộng.
- Tư duy phản biện và phân tích kỹ thuật chuyên sâu các vấn đề thường gặp trong vận hành hệ thống ở môi trường production: mở rộng kết nối WebSocket, thu thập dữ liệu không ổn định từ nhiều nguồn, bảo vệ hệ thống khỏi tấn công bảo mật.
- Đề xuất kiến trúc phù hợp, thể hiện qua sơ đồ rõ ràng và có lý do giải thích từng thành phần (gateway, load balancer, broker, hash router, LLM parser, auth service...).
- Hiểu và so sánh các giải pháp kỹ thuật, phân tích trade-off về hiệu năng, chi phí, khả năng chịu lỗi và mở rộng.
- Vận dụng LLM như một công cụ hỗ trợ phân tích, biết cách đặt câu hỏi hiệu quả, phản biện và đánh giá kết quả thay vì sao chép máy móc.
- Phát triển kỹ năng thiết kế kiến trúc phần mềm hiện đại, áp dụng vào các hệ thống thực tế như: dịch vụ tài chính, streaming, giao dịch thời gian thực, hệ thống phân tích AI, microservices bảo mật cao.

Sinh viên cần:

- Tự nghiên cứu vấn đề kỹ thuật dựa vào tình huống.
- Thiết kế sơ đồ kiến trúc minh họa rõ luồng xử lý.
- So sánh và giải thích các giải pháp đã lựa chọn.
- Viết prompt để khai thác LLM một cách hiệu quả và phản biện kết quả nhận được.
- Rút ra bài học tổng kết về tư duy kiến trúc và kỹ năng kỹ thuật.

Dựa vào yêu cầu đồ án cuối kì để trả lời các vấn đề sau:

https://docs.google.com/document/d/178WNXKYaEVrIRbhyzhxfncOUJsNZu-peibyGSHf0cDA

TÌNH HUỐNG 1: Scale WebSocket với hơn 1000 client và nhiều cặp tiền

Bối cảnh

Bạn được giao xây dựng hệ thống hiển thị giá tài chính thời gian thực cho hơn 1000 người dùng đồng thời. Mỗi người có thể mở nhiều tab (BTCUSDT, ETHUSDT...) và mỗi tab sẽ mở một kết nối WebSocket riêng để nhận dữ liệu từ Binance.

Vấn đề gặp phải:

- Kết nối WebSocket là kết nối bền vững (long-lived), khiến việc mở rộng trở nên khó khăn.
- Load balancer chỉ điều phối kết nối mới, không điều phối được kết nối đã mở, dẫn đến mất cân bằng giữa các instance.
- Khi số lượng client tăng cao, các instance cũ bị quá tải trong khi các instance mới không nhận được kết nối.
- Nếu một server chết, kết nối bị mất và client không thể tự khôi phục dữ liệu.
- Việc dùng Pub/Sub để đồng bộ tin nhắn giữa các server tốn tài nguyên và chi phí cao.
- Không thể tự động remap kết nối khi scale in/out nếu không có cơ chế hashing hoặc custom load balancer.

Yêu cầu

1. Phân tích các vấn đề kỹ thuật liên quan đến việc mở rộng WebSocket.
2. Giải thích nguyên lý của các giải pháp như sticky session, IP hash, rendezvous hashing.
3. Đề xuất kiến trúc giải quyết vấn đề cân bằng tải khi scale WebSocket trên môi trường Kubernetes.

4. Vẽ sơ đồ kiến trúc hệ thống WebSocket bao gồm: WebSocket Gateway, Load Balancer, Message Broker, Hash Routing hoặc các giải pháp custom khác.

5. Kết luận: Giải pháp của bạn có thể hỗ trợ bao nhiêu client? Độ trễ trung bình là bao nhiêu? Có chịu lỗi hay không?

Tài liệu tham khảo:
https://medium.com/lumen-engineering-blog/how-to-implement-a-distributed-and-auto-scalable-websocket-server-architecture-on-kubernetes-4cc32e1dfa45

TÌNH HUỐNG 2: Thu thập tin tức từ nhiều website khi cấu trúc trang web thay đổi

Bối cảnh

Bạn cần thu thập tin tức tài chính từ hàng trăm website khác nhau. Một số trang sử dụng JavaScript để hiển thị nội dung (Single Page App), một số thường xuyên thay đổi cấu trúc HTML. Các công cụ crawler truyền thống như XPath hoặc CSS Selector liên tục bị lỗi, cần cập nhật lại thủ công, gây tốn kém chi phí bảo trì.

Bạn muốn hệ thống có thể:

- Tự thích ứng với thay đổi cấu trúc trang.
- Phân tích nội dung chính của bài viết (tiêu đề, ngày, nội dung) từ HTML hỗn tạp.
- Vận hành ổn định khi scale lên hàng nghìn trang cần thu thập.

Yêu cầu

1. Trình bày lý do tại sao các crawler truyền thống dễ bị vỡ khi trang web thay đổi.

2. Đề xuất một kiến trúc crawler sử dụng AI hoặc LLM để phân tích nội dung trang web.

3. So sánh giữa việc dùng rule cố định (XPath) và sử dụng AI như Firecrawl.dev, LlamaParse hoặc Google Document AI.

4. Vẽ sơ đồ kiến trúc crawler gồm: Crawl Scheduler, URL Queue, Crawler Engine, AI Parser, Data Store.

5. Kết luận: Trong trường hợp nào nên dùng AI-based parser? Trường hợp nào dùng rule-based hiệu quả hơn?

TÌNH HUỐNG 3: Sự cố bảo mật trong hệ thống khi mở rộng

Bối cảnh

Trong giai đoạn thử nghiệm mở rộng của hệ thống tài chính AI, nhóm phát triển phát hiện ra một số API bị gọi bất thường trực tiếp vào các service nội bộ như `price-prediction-service` và `portfolio-backtest-service`. Các yêu cầu đó:

- Không đi qua API Gateway.
- Đính kèm JWT hợp lệ nhưng đã cũ (được cấp cách đây 1 tuần).
- Bị gửi từ các client giả mạo.

Hệ quả:

- Một số chiến lược đầu tư bị thực thi sai do sử dụng dữ liệu giả mạo.
- Không có log kiểm tra xác thực nên không phát hiện được các yêu cầu giả.
- Attacker phát tán hàng loạt JWT giả đến các microservice backend.

Yêu cầu

1. Phân tích tại sao việc chỉ sử dụng JWT là không đủ bảo mật trong hệ thống phân tán.

2. Trình bày cách kết hợp OAuth2 và JWT để vừa đảm bảo tính không trạng thái vừa đảm bảo khả năng kiểm soát truy cập.

3. Đề xuất cách triển khai refresh token và blacklist token để giảm rủi ro bị lạm dụng.

4. Giải thích tại sao các microservice không nên tin tưởng lẫn nhau, và cách triển khai xác thực theo mô hình zero-trust.

5. Trình bày vai trò của API Gateway trong kiểm soát truy cập đầu vào. Nếu attacker gửi yêu cầu trực tiếp vào IP nội bộ của service, bạn sẽ xử lý như thế nào?

6. Vẽ sơ đồ xác thực gồm: Auth Service, API Gateway, OAuth2 Flow, JWT, Redis hoặc DB lưu phiên.

7. Kết luận: Qua tình huống này, bạn rút ra bài học gì về kiến trúc bảo mật cho hệ thống microservice?

Tài liệu sinh viên có thể tham khảo:

https://medium.com/lumen-engineering-blog/how-to-implement-a-distributed-and-auto-scalable-websocket-server-architecture-on-kubernetes-4cc32e1dfa45
Omphalos, Uber's Parallel and Language-Extensible Time Series Backtesting Tool, truy cập vào tháng 8 4, 2025, https://www.uber.com/blog/omphalos/
OAuth vs. JWT: What Is the Difference & Using Them Together - Frontegg, truy cập vào tháng 8 4, 2025, https://frontegg.com/blog/oauth-vs-jwt
JWT vs OAuth: Build a Future-Proof Authentication System - Strapi, truy cập vào tháng 8 4, 2025, https://strapi.io/blog/jwt-vs-oauth
The Growing Importance of a Separate Authorization Service in Modern Microservices Architectures | by Abhishek Maurya | Medium, truy cập vào tháng 8 4, 2025, https://medium.com/@abhishek4023/the-growing-importance-of-a-separate-authorization-service-in-modern-microservices-architectures-58a2917f866c
Best Practices for Authorization in Microservices - Oso, truy cập vào tháng 8 4, 2025, https://www.osohq.com/post/microservices-authorization-patterns
Managing User Authentication and Authorization in Microservices Architectures - Medium, truy cập vào tháng 8 4, 2025, https://medium.com/hexaworks-papers/managing-user-authentication-and-authorization-in-microservices-architectures-fae705f40627