001

Tóm tắt nội dung:

- Phần trước đã nói về cách microservices giao tiếp bất đồng bộ bằng RabbitMQ.
- Phần này sẽ tập trung vào giao tiếp bất đồng bộ bằng Apache Kafka.

So sánh RabbitMQ và Kafka:

Thiết kế:

- Kafka là nền tảng stream dữ liệu phân tán.
- RabbitMQ là message broker (trình môi giới thông điệp).
- Kafka phù hợp cho dữ liệu lớn, RabbitMQ tốt cho routing phức tạp.

Lưu trữ dữ liệu:

- Kafka lưu trên ổ đĩa → lưu lâu dài.
- RabbitMQ lưu trong bộ nhớ → thích hợp cho ứng dụng yêu cầu độ trễ thấp.

Hiệu suất:

- Kafka nhanh hơn khi xử lý khối lượng dữ liệu lớn.
- RabbitMQ hiệu quả hơn trong routing phức tạp.

Khả năng mở rộng:

- Kafka mở rộng dễ dàng bằng cách thêm broker.
- RabbitMQ bị giới hạn trong khả năng mở rộng.

Kết luận:

- Kafka phù hợp với hệ thống cần xử lý lượng dữ liệu lớn, hiệu suất cao.
- RabbitMQ phù hợp với hệ thống cần routing phức tạp, dễ bảo trì, và dữ liệu nhỏ hằng ngày.
- Việc chọn công cụ phụ thuộc vào nhu cầu cụ thể của hệ thống.

002

Giới thiệu bằng ví dụ đời sống thực:

- Giảng viên so sánh Apache Kafka với một hệ thống giải trí tại gia, nơi các thiết bị như đầu đĩa DVD, USB, antenna đóng vai trò là producers (nhà sản xuất dữ liệu).

- Bộ thu (receiver) là broker (môi giới), tiếp nhận và phân phối dữ liệu đến television/speakers (consumers).
- Ví dụ này giúp hình dung Kafka là một môi giới trung gian truyền dữ liệu từ nơi tạo ra đến nơi tiêu thụ.

So sánh Kafka với RabbitMQ:

- Kafka có khả năng xử lý lượng dữ liệu lớn theo thời gian thực, còn RabbitMQ bị giới hạn ở lượng dữ liệu nhỏ.

Khái niệm cơ bản về Apache Kafka:

- Là nền tảng mã nguồn mở, phân tán, xử lý sự kiện theo thời gian thực.
- Được dùng để xây dựng data pipelines và real-time applications.
- Hỗ trợ throughput cao, fault-tolerant, scalable.

Các thành phần chính của Kafka:

- Producers: Tạo và gửi dữ liệu (sự kiện).
- Kafka Cluster: Gồm nhiều brokers, mỗi broker là một máy chủ tiếp nhận dữ liệu và phân phối đến consumer.
- Topics: Tương tự như "exchange" trong RabbitMQ – nơi lưu trữ dữ liệu từ producers.
- Partitions: Mỗi topic có thể chia thành nhiều phân vùng để phân tán và mở rộng quy mô lưu trữ.
- Offset: Mỗi message trong partition có một ID (offset) để xác định thứ tự và vị trí, giống như sequence ID trong cơ sở dữ liệu.

Lợi ích của partition và offset:

- Giúp Kafka lưu trữ và xử lý dữ liệu lớn bằng cách phân chia công việc và lưu trữ ra nhiều broker khác nhau.
- Offset giúp consumers biết đã xử lý đến đâu trong từng partition.

003

Phía Producer (Người gửi):

Cấu hình Producer:

- Xác định URL Kafka broker, định dạng serialization, và các tuỳ chọn như nén dữ liệu, batching.

Chọn Topic:

- Producer phải chọn topic để gửi dữ liệu. Nếu topic chưa tồn tại, nó có thể được tạo tự động (nếu Kafka broker cho phép).

Gửi Message:

- Producer sử dụng API Kafka để gửi message đã được serialize đến topic và (nếu có) chỉ định partition key.

Kafka xử lý Message:

- Kafka gán message vào một partition (dựa theo partition key hoặc dùng Round Robin/hash).

Gán offset ID cho message.

- Lưu trữ message và nếu được cấu hình, tiến hành replication (đồng bộ hay bất đồng bộ).

Gửi Acknowledgment:

- Kafka gửi xác nhận lại cho producer (thành công hoặc lỗi).
- Tùy cấu hình, producer có thể đợi xác nhận từ tất cả replicas hoặc chỉ leader replica.

Phía Consumer (Người nhận):

Tham gia Consumer Group:

- Consumer phải tham gia vào một consumer group và đăng ký (subscribe) các topic.

Phân phối Partition:

- Mỗi partition được chỉ định cho một consumer duy nhất trong nhóm, giúp phân phối đều và xử lý song song.

Quản lý Offset:

- Consumer giữ track offset để biết đã xử lý đến đâu.

Gửi Fetch Request:

- Consumer yêu cầu Kafka gửi message bắt đầu từ offset hiện tại, và có thể nhận nhiều message mỗi lần.

Nhận & Xử lý Message:

- Kafka trả về message cùng offset & metadata.

- Consumer xử lý message theo business logic (chuyển đổi, tính toán, tổng hợp...).

Commit Offset:

- Sau khi xử lý, consumer commit offset lên Kafka để lưu tiến độ — giúp resume nếu có lỗi hoặc restart.

Lặp lại liên tục:

- Consumer liên tục fetch, xử lý và commit offset — tạo thành polling loop để xử lý message theo thời gian thực.

Kết luận:

- Mặc dù hệ thống có vẻ phức tạp, Spring Cloud Stream giúp đơn giản hóa toàn bộ quá trình cấu hình và tích hợp Kafka vào ứng dụng microservices.
- Các thao tác như tạo topic, partition hay offset sẽ được Spring tự động xử lý, giúp developer tập trung vào logic chính.

004

Giới thiệu Kafka:

Kafka là một nền tảng xử lý dữ liệu theo thời gian thực, được hơn 80% các công ty Fortune 100 sử dụng, đặc biệt trong các ngành như sản xuất, ngân hàng, bảo hiểm và viễn thông — nơi có lượng dữ liệu lớn cần xử lý bất đồng bộ.

Truy cập trang Kafka:

Truy cập trang chính thức kafka.apache.org để đọc giới thiệu và hướng dẫn cài đặt.

Cài đặt Kafka cục bộ:

1) Vào phần “Get started” → “Quick start”.
2) Tải xuống Kafka phù hợp với hệ điều hành (trong video là MacOS).
3) Giải nén file đã tải.
4) Di chuyển vào thư mục Kafka trong terminal.

Chạy Kafka với KRaft (thay thế Zookeeper):

1) Tạo Cluster ID.
2) Khởi tạo cluster với cluster ID đã tạo.

3) Chạy Kafka server, mặc định tại cổng 9092.

Hoàn tất cài đặt: Sau khi Kafka chạy thành công, ta đã sẵn sàng tích hợp nó với các microservices.

Tiếp theo: Trong bài học tiếp theo, sẽ cập nhật mã nguồn của các microservices (như accounts và messages) để kết nối và sử dụng Kafka cho giao tiếp bất đồng bộ và truyền sự kiện.

005

Mục tiêu chính:

Chuyển đổi microservices accounts và message từ sử dụng RabbitMQ sang Apache Kafka thông qua Spring Cloud Stream.

Các bước thực hiện:

Chuẩn bị Workspace:

- Tạo folder mới cho Section 14.
- Sao chép mã nguồn từ Section 13, đổi tên folder thành section14.
- Mở dự án trong IntelliJ và tải lại các Maven projects.

Cập nhật Dependencies:

- Trong pom.xml của cả accounts và message microservice:
- Thay spring-cloud-stream-binder-rabbit bằng spring-cloud-stream-binder-kafka.
- Đổi tag từ s13 thành s14.
- Reload lại Maven sau khi sửa.

Cấu hình application.yml:

- Xóa các cấu hình liên quan đến RabbitMQ.
- Thêm cấu hình Kafka

Khởi động các microservices:

- Bao gồm: Config server, Eureka server, accounts, message, và gateway.
- Kiểm tra kết nối Kafka qua plugin Kafkalytic trong IntelliJ:
- Xác nhận có broker, consumers và topics (communication-sent, send-communication) được tạo đúng.

Kiểm tra giao tiếp bất đồng bộ:

- Khởi động Keycloak (vì gateway bảo mật bằng OAuth2).
- Lấy access token từ Postman.
- Gửi request đến create API của accounts.
- Đặt breakpoint trong message microservice → xác nhận message được nhận qua Kafka.
- Kiểm tra dữ liệu trong H2 database để xác minh cập nhật từ message.

Kết luận:

Chỉ cần thay đổi dependencies và cấu hình kết nối, các microservices có thể chuyển từ RabbitMQ sang Kafka dễ dàng nhờ Spring Cloud Stream. Việc tích hợp trở nên linh hoạt và tối giản cho developer.

006

1. Tạo Docker Images và Đẩy Lên Docker Hub:

Tạo Docker images cho tất cả 7 ứng dụng với tag "S14".

Đẩy các image này lên Docker Hub để xác nhận, mỗi image đều có tag "s14".

2. Cập Nhật Docker Compose:

Cập nhật file Docker Compose để sử dụng các image "s14" vừa tạo.

Xóa bỏ phần cấu hình cho RabbitMQ và thay thế bằng cấu hình dịch vụ Kafka.

Lấy tham khảo cách cấu hình từ repo GitHub của Bitnami (một nguồn đáng tin cậy được VMware hỗ trợ) để thiết lập Kafka với Docker Compose, bao gồm:

Định nghĩa service Kafka với image, port mapping và cấu hình volumes (ví dụ: ánh xạ thư mục lưu trữ dữ liệu Kafka vào thư mục trên máy cục bộ).

Cập nhật các environment variable trong microservice (Accounts, Message) để thay đổi thông tin kết nối từ RabbitMQ sang Kafka (sử dụng biến môi trường spring_cloud_stream_kafka_binder_brokers với giá trị là tên service Kafka và port 9092).

3. Khởi Động Lại Các Container:

Dừng các container và server đang chạy (bao gồm cả Kafka server chạy cục bộ và container Keycloak) để đảm bảo không xung đột.

Chạy lại Docker Compose bằng lệnh docker compose up -d để khởi động tất cả container mới theo cấu hình đã cập nhật.

4. Cấu Hình và Kiểm Tra Keycloak:

Cập nhật cấu hình client trong Keycloak (tạo client mới với ID easybank-callcenter-cc, cấu hình xác thực, tắt standard flow và bật service account roles).

Lấy thông tin credential từ Keycloak để dùng trong Postman.

5. Kiểm Tra Giao Tiếp Giữa Microservices:

Dùng Postman gọi API từ Accounts microservice để trigger quá trình giao tiếp bất đồng bộ.

Xác nhận thông qua log của Message microservice (breakpoint dừng tại hàm gửi email, SMS, thông tin offset) và log của Accounts microservice (xác nhận cập nhật trạng thái giao tiếp).

Kiểm tra Kafka topics và consumer thông qua plugin Kafkalytic trong IntelliJ, cho thấy cả hai service (Accounts và Message) kết nối thành công tới Kafka.

6. Tổng Kết và Đánh Giá:

Các thay đổi chỉ bao gồm:

- Thay thế dependency của RabbitMQ bằng Kafka.
- Cập nhật thông tin kết nối trong file application.yml.

Hệ thống microservices sử dụng Apache Kafka để giao tiếp bất đồng bộ đã được triển khai thành công.

Cuối cùng, giảng viên yêu cầu học viên đánh giá khóa học trên Udemy và gửi phản hồi qua các kênh như Udemy, LinkedIn hoặc email.