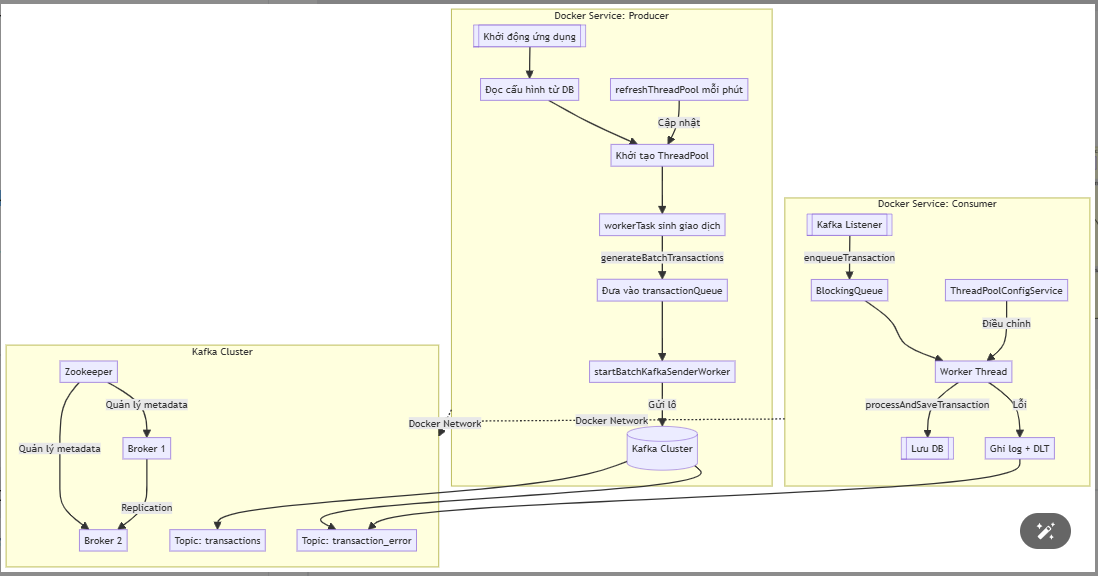
Sơ đồ flow diagram

Luồng hoạt động:

Khởi động:

Ứng dụng khởi động với ProducerApplication

MultiThreadTransactionProducer được khởi tạo (implement CommandLineRunner)

Đọc số lượng thread từ database (mặc định là 5)

Khởi tạo thread pool và các worker

Sinh giao dịch:

Mỗi worker thread chạy workerTask()

Sinh giao dịch theo batch (mỗi batch 10 giao dịch)

Mỗi thread gửi 300 giao dịch/phút

Giao dịch được đưa vào transactionQueue

Gửi giao dịch:

Thread kafka-batch-sender lấy giao dịch từ queue

Gom batch tối đa 50 giao dịch

Gửi batch đến Kafka thông qua TransactionProducer

Đợi 1 giây trước khi gửi batch tiếp theo

Tối ưu hiệu năng:

Sử dụng batch size 64KB

Linger time 5ms để gom batch

Buffer memory 32MB

ACK level 1 (chỉ cần leader xác nhận)

Queue size 10,000 để tránh tràn RAM

Batch size 50 giao dịch khi gửi

I. Tài liệu mô tả Service Producer

# 1. Giới thiệu

Service Producer được sử dụng để sinh ra các giao dịch giả lập và gửi chúng đến Kafka theo lô, sử dụng cơ chế đa luồng. Lớp này triển khai `CommandLineRunner`, cho phép chạy logic ngay khi ứng dụng khởi động.

# 2. Các thành phần chính

## 2.1. Các dependency được inject:

- `TransactionProducer`: Service gửi giao dịch tới Kafka.

- `ThreadConfigRepository`: Truy vấn số lượng luồng từ cấu hình trong CSDL.

- `ObjectMapper`: Dùng để chuyển đối tượng Java thành chuỗi JSON.

## 2.2. Biến và cấu hình:

- `senderExecutor`: Thread pool quản lý các worker sinh giao dịch.

- `transactionQueue`: Hàng đợi chứa các giao dịch được sinh ra trước khi gửi.

- `currentThreadCount`: Theo dõi số luồng hiện tại.

- `SENDER\_BATCH\_SIZE`: Kích thước mỗi lô gửi Kafka.

- `SENDER\_INTERVAL\_MS`: Thời gian nghỉ giữa các lần gửi lô.

# 3. Luồng hoạt động của class

## 3.1. Phương thức run(String... args)

- Được gọi khi ứng dụng khởi động.

- Lấy cấu hình số luồng từ DB và khởi tạo ThreadPool.

- Bắt đầu các worker sinh giao dịch (`workerTask`).

- Khởi động luồng gửi giao dịch theo lô (`startBatchKafkaSenderWorker`).

## 3.2. Phương thức workerTask()

- Chạy vô hạn trong thread pool để sinh giao dịch giả lập.

- Tạo danh sách JSON giao dịch và đưa vào hàng đợi.

- Nghỉ một khoảng thời gian để kiểm soát tốc độ sinh giao dịch.

## 3.3. Phương thức startBatchKafkaSenderWorker()

- Tạo một thread daemon liên tục đọc từ hàng đợi.

- Gom các giao dịch thành từng lô và gửi từng bản ghi đến Kafka.

- Ghi log thông tin số lượng bản ghi đã gửi.

## 3.4. Phương thức refreshThreadPool()

- Chạy định kỳ mỗi phút nhờ annotation `@Scheduled`.

- Tự động điều chỉnh số lượng luồng theo cấu hình mới từ DB.

## 3.5. Phương thức generateBatchTransactions(int count)

- Sinh ra một danh sách các giao dịch giả lập (transactionId, amount, timestamp).

- Serialize thành chuỗi JSON để gửi đi.

# 4. Tổng kết

Lớp `MultiThreadTransactionProducer` đóng vai trò quan trọng trong việc mô phỏng hệ thống xử lý giao dịch số lượng lớn bằng Kafka trong môi trường Spring Boot. Việc sử dụng thread pool và hàng đợi giúp xử lý hiệu quả, tránh mất dữ liệu và dễ dàng mở rộng khi cần thiết.

II. Tài liệu mô tả ứng dụng Kafka Consumer

# 1. Giới thiệu

Ứng dụng Kafka Consumer có nhiệm vụ nhận các giao dịch từ Kafka topic, xử lý chúng song song thông qua ThreadPool, ghi vào cơ sở dữ liệu, đồng thời xử lý các lỗi phát sinh bằng cách lưu log và gửi message lỗi vào topic DLT (Dead Letter Topic).

# 2. Kiến trúc tổng thể

Ứng dụng bao gồm các thành phần chính:  
- Kafka Listener: nhận message từ topic và đẩy vào hàng đợi.  
- Queue trung gian (BlockingQueue): đóng vai trò hàng đợi xử lý song song.  
- ThreadPoolExecutor: thực thi các task xử lý giao dịch song song.  
- TransactionProcessorService: lớp điều phối xử lý giao dịch.  
- ThreadPoolConfigService: quản lý cấu hình động số lượng thread.  
- KafkaTemplate: gửi message lỗi vào topic DLT.  
- JdbcTemplate: ghi dữ liệu vào DB.  
- errorLogRepository: ghi log lỗi vào bảng.

# 3. Kafka Listener

Lớp lắng nghe topic 'transaction', mỗi message sẽ được đẩy vào hàng đợi trung gian thông qua phương thức `enqueueTransaction()`. Offset chỉ được commit sau khi enqueue thành công để đảm bảo không mất dữ liệu.

# 4. TransactionProcessorService

- Tạo queue trung gian.  
- Khởi tạo các worker thread từ ThreadPool để xử lý queue.  
- Mỗi worker thread lấy message từ queue và gọi `processAndSaveTransaction()`.  
- Nếu xảy ra lỗi, log lỗi và gửi message sang topic 'transaction\_error'.

# 5. ThreadPoolConfigService

- Quản lý thread pool động, lấy thông tin số lượng thread từ bảng cấu hình DB.  
- Có khả năng điều chỉnh thread pool khi cấu hình thay đổi.  
- Cho phép submit task xử lý song song.

# 6. Luồng xử lý giao dịch

1. Kafka Listener nhận message.  
2. Đẩy message vào queue trung gian.  
3. Worker thread lấy message từ queue và xử lý:  
 - Parse JSON.  
 - Lưu vào bảng transactions.  
 - Nếu lỗi -> ghi log + gửi topic transaction\_error.

# 7. Ưu điểm mô hình

- Xử lý song song tối ưu với cấu hình thread pool linh hoạt.  
- Đảm bảo an toàn dữ liệu, không mất message.  
- Có cơ chế log và gửi lại lỗi để xử lý sau.  
- Dễ mở rộng và bảo trì.

# III . Triển Khai Docker

Tách biệt các dịch vụ (Producer và Consumer chạy độc lập)

1. Kiến trúc tổng thể

**Kafka Cluster**:

* Đây là nơi lưu trữ và phân phối dữ liệu giữa các Producer và Consumer. Kafka thường được triển khai trên các máy chủ riêng biệt hoặc dưới dạng một Kafka Cluster với nhiều node để đảm bảo tính sẵn sàng cao và khả năng chịu lỗi.
* Kafka Cluster: Gồm nhiều broker (máy chủ), giúp phân phối và lưu trữ dữ liệu. Các producer kết nối với Kafka Cluster để ghi dữ liệu vào topic, và consumer kết nối để đọc dữ liệu.

**Kafka Producer**:

* Chạy trên một hoặc nhiều ứng dụng hoặc dịch vụ độc lập. Các Producer gửi dữ liệu (giao dịch, thông điệp, v.v.) vào các topic của Kafka.
* Producer Service: Một dịch vụ độc lập gửi dữ liệu vào Kafka. Producer có thể sử dụng các API của Kafka (như KafkaProducer trong Java) để gửi các message hoặc giao dịch vào một topic của Kafka.

**Kafka Consumer**: Chạy trên các ứng dụng hoặc dịch vụ độc lập, tiêu thụ dữ liệu từ các topic của Kafka và xử lý dữ liệu (lưu vào cơ sở dữ liệu, tính toán, v.v.).

**Zookeeper**: Quản lý các metadata của Kafka và giúp đồng bộ hóa các node trong Kafka Cluster. Zookeeper đảm bảo Kafka có thể phân phối dữ liệu giữa các producer và consumer một cách đồng bộ các thành phần chính của hệ thống

## 2. Cách thức hoạt động

### 2.1 Kafka Producer (Gửi dữ liệu vào Kafka)

Cấu hình: Producer cần có cấu hình kết nối tới Kafka Cluster (bao gồm các Kafka broker) và các thông tin về topic mà nó sẽ ghi dữ liệu vào.

Luồng gửi dữ liệu: Producer thực hiện việc serialize dữ liệu (chuyển đối tượng Java thành định dạng JSON hoặc Avro) và gửi chúng vào các topic Kafka. Dữ liệu có thể được gửi theo từng message nhỏ hoặc theo từng batch (lô).

Đảm bảo tin cậy: Để đảm bảo dữ liệu không bị mất, các Producer thường sử dụng acknowledgement (ACK) từ Kafka để xác nhận rằng message đã được ghi vào topic thành công.

### 2.2 Kafka Consumer (Nhận và xử lý dữ liệu từ Kafka)

Cấu hình: Consumer cũng cần cấu hình kết nối với Kafka Cluster và topic mà nó sẽ tiêu thụ. Consumer sẽ đăng ký vào một topic và bắt đầu nhận dữ liệu từ topic đó.

Đọc dữ liệu: Consumer đọc dữ liệu theo từng message hoặc theo batch. Nó có thể sử dụng các kỹ thuật như commit offset để đảm bảo rằng message đã được xử lý thành công và không bị mất.

Xử lý dữ liệu: Sau khi nhận được dữ liệu từ Kafka, Consumer sẽ tiến hành xử lý (chẳng hạn như lưu vào cơ sở dữ liệu, tính toán, phân tích, v.v.). Nếu có lỗi, Consumer có thể ghi lại lỗi vào bảng log hoặc gửi lại lỗi vào một topic DLT (Dead Letter Topic).

### 2.3 Quản lý và giám sát

Quản lý luồng: Mỗi dịch vụ (Producer và Consumer) có thể được triển khai độc lập và có thể mở rộng riêng biệt. Ví dụ, nếu số lượng giao dịch tăng lên, bạn có thể mở rộng số lượng Kafka Producers mà không ảnh hưởng đến Consumer.

Giám sát Kafka: Kafka cần được giám sát để đảm bảo hiệu suất và tính sẵn sàng cao. Các công cụ như Kafka Manager hoặc Confluent Control Center có thể giúp theo dõi tình trạng Kafka Cluster, các topic, partition, và consumer lag (khoảng cách giữa producer và consumer).

Điều chỉnh số lượng thread trong Consumer: Bạn có thể điều chỉnh số lượng thread trong Kafka Consumer để tăng hiệu suất xử lý song song, giúp hệ thống xử lý nhiều giao dịch cùng lúc.

## 3. Kết nối giữa Producer và Consumer qua Kafka

Kafka đóng vai trò là hệ thống message broker, kết nối giữa các Producer và Consumer. Producer chỉ cần gửi dữ liệu vào Kafka, và Consumer sẽ nhận dữ liệu này bất kể Producer đang gửi dữ liệu từ đâu.

Partition và Replication: Kafka sử dụng cơ chế phân tán dữ liệu thông qua các partition và replication. Một topic có thể có nhiều partition, giúp phân tán dữ liệu và tăng khả năng xử lý song song. Kafka cũng sao lưu dữ liệu qua các replication để tránh mất mát dữ liệu khi có sự cố.

