Apache Kafka là một nền tảng streaming phân tán. Nói đơn giản, nó cho phép bạn publish và subscribe các luồng dữ liệu (messages), lưu trữ một cách đáng tin cậy, và xử lý dữ liệu gần như realtime. Kafka thường được dùng cho các pipeline dữ liệu tốc độ cao, có thể mở rộng và chịu lỗi tốt.
Sự khác nhau chính là cách xử lý message. Queue truyền thống như IBM MQ lưu message cho đến khi consumer xử lý, và thường xóa message sau khi tiêu thụ. Kafka thì lưu message trong khoảng thời gian có thể cấu hình, và nhiều consumer có thể đọc cùng một message độc lập. Kafka thiết kế để streaming dữ liệu tốc độ cao, phân tán, còn queue truyền thống tập trung vào giao dịch, đảm bảo message chỉ xử lý một lần.
Trong Kafka, Producer là ứng dụng hoặc service gửi message (record) vào Kafka topic. Consumer là ứng dụng hoặc service đọc message từ Kafka topic. Producer và Consumer hoàn toàn tách biệt, nên nhiều consumer có thể đọc cùng một message độc lập mà không ảnh hưởng đến producer.
Trong Kafka, topic giống như một danh mục hoặc tên luồng để gửi message. Producer gửi message vào topic, còn consumer đọc message từ topic đó. Topic có thể được chia thành nhiều partition, giúp Kafka mở rộng ngang và phân phối tải giữa nhiều consumer.
Trong Kafka, partition là một phần nhỏ của topic, giống như một log nơi lưu trữ các message theo thứ tự. Partition cho phép topic chia ra nhiều broker, giúp Kafka mở rộng ngang và xử lý lượng dữ liệu lớn. Partition còn cho phép nhiều consumer trong cùng một consumer group đọc message song song, cải thiện hiệu năng và parallelism. Mỗi message trong partition có một offset duy nhất, giúp theo dõi thứ tự message.
Trong Kafka, offset là một định danh duy nhất cho mỗi message trong một partition. Nó thể hiện vị trí của message trong log của partition. Consumer dùng offset để theo dõi những message đã đọc, nhờ đó nếu consumer khởi động lại hoặc gặp lỗi, nó có thể tiếp tục từ đúng vị trí. Offset giúp Kafka đảm bảo tiêu thụ message đáng tin cậy và có thể replay lại.
Trước phiên bản Kafka 2.8, ZooKeeper được dùng để quản lý metadata và phối hợp giữa các broker trong cluster. Nó theo dõi các broker, topic, partition và leader election. ZooKeeper giúp Kafka biết broker nào đang hoạt động, broker nào là leader của partition, và đảm bảo tính nhất quán và chịu lỗi của cluster. Nói đơn giản, ZooKeeper giống như trình quản lý cluster của Kafka.
Khi một broker trong Kafka bị lỗi, các partition mà broker đó đang làm leader sẽ được chuyển leadership tự động sang một replica đồng bộ khác. Consumer có thể tạm thời không đọc được dữ liệu từ những partition đó cho đến khi leader mới được bầu. Producer vẫn có thể gửi message vào các partition khác bình thường. Cơ chế failover này đảm bảo khả năng sẵn sàng cao và chịu lỗi, giúp cluster tiếp tục hoạt động ngay cả khi một broker gặp sự cố.
Kafka đảm bảo khả năng chịu lỗi và độ tin cậy thông qua replication, kiến trúc leader-follower và cơ chế xác nhận (ack). Mỗi partition có một leader và nhiều follower (replica). Producer có thể chọn mức ack để đảm bảo message được ghi ít nhất một replica hoặc tất cả replica. Nếu một broker bị lỗi, follower sẽ tự động trở thành leader mới, dữ liệu không bị mất. Ngoài ra, Kafka lưu message trên disk, đảm bảo lưu trữ bền vững và có thể replay message khi cần.
Consumer Group là một tập hợp các consumer cùng làm việc để tiêu thụ message từ một hoặc nhiều topic. Mỗi consumer trong nhóm đọc message từ các partition khác nhau, đảm bảo mỗi message chỉ được xử lý bởi một consumer duy nhất trong nhóm. Kafka tự động cân bằng các partition giữa các consumer. Điều này giúp xử lý song song, mở rộng và phân phối tải. Nhiều consumer group có thể tiêu thụ cùng một topic độc lập, mỗi nhóm nhận tất cả message.
Đây là các đảm bảo về cơ chế gửi message trong Kafka hoặc hệ thống messaging khác: - At most once – Message được gửi 0 hoặc 1 lần. Có thể có message bị mất, nhưng không bao giờ bị trùng. - At least once – Message được gửi ít nhất 1 lần, có thể nhiều lần. Message không bị mất, nhưng consumer có thể nhận trùng lặp. - Exactly once – Message được gửi chính xác 1 lần, không mất và không trùng lặp. Rất quan trọng trong các giao dịch tài chính. Trong Kafka, bạn có thể cấu hình producer và consumer để đạt at least once hoặc exactly once bằng cách sử dụng acknowledgment và transactional đúng cách.
Kafka đạt hiệu năng cao nhờ một số cơ chế chính: 1. Partitioning: Topic được chia thành nhiều partition, cho phép đọc và ghi song song trên nhiều broker. 2. Batching: Producer và consumer có thể gửi và nhận message theo lô (batch), giảm overhead mạng. 3. Ghi đĩa theo thứ tự (sequential write): Kafka ghi message vào log theo thứ tự, nhanh hơn nhiều so với ghi ngẫu nhiên. 4. Zero-copy: Kafka dùng cơ chế zero-copy để truyền dữ liệu từ đĩa ra mạng, giảm tải CPU. 5. Replication và parallelism: Nhiều broker và consumer group cho phép xử lý phân tán, hệ thống mở rộng ngang. Nhờ những cơ chế này, Kafka có thể xử lý hàng triệu message mỗi giây với độ trễ thấp.
Kafka Broker là một server Kafka đơn lẻ, nó lưu trữ dữ liệu và xử lý yêu cầu từ client (gửi và nhận message). Broker quản lý topic, partition và lưu trữ message. Kafka Cluster là một nhóm nhiều broker cùng hoạt động, giúp hệ thống chịu lỗi, sẵn sàng cao và có thể mở rộng. Mỗi partition của topic được phân phối trên nhiều broker với leader và follower, nên nếu một broker gặp sự cố, broker khác có thể thay thế.
Retention policy trong Kafka định nghĩa thời gian mà message được giữ trong topic trước khi bị xóa. Message được lưu trên đĩa trong một khoảng thời gian có thể cấu hình (retention.ms) hoặc cho đến khi topic đạt kích thước tối đa (retention.bytes). Điều này cho phép Kafka replay message, hỗ trợ các consumer chạy muộn và quản lý dung lượng đĩa. Chính sách retention có thể cấu hình riêng cho từng topic, nên các topic quan trọng có thể giữ dữ liệu lâu hơn topic tạm thời.
Trong Kafka, replication có nghĩa là mỗi partition của topic được sao chép trên nhiều broker. Một broker đóng vai trò leader, các broker khác là follower (replica). Producer và consumer tương tác với leader, trong khi followers luôn đồng bộ. Replication quan trọng vì nó đảm bảo khả năng chịu lỗi: nếu broker leader bị lỗi, một follower có thể trở thành leader mới, đảm bảo không mất dữ liệu. Nó cũng tăng tính sẵn sàng cao, giúp cluster tiếp tục hoạt động ngay cả khi một số broker gặp sự cố.
Nếu hai consumer thuộc cùng một consumer group, Kafka sẽ chia các partition của topic giữa các consumer. Mỗi partition chỉ được một consumer trong nhóm đọc, nên message không bị xử lý nhiều lần trong cùng nhóm. Điều này giúp xử lý song song và cân bằng tải giữa các consumer. Nếu một consumer bị lỗi, Kafka sẽ rebalance các partition, để consumer còn lại tiếp nhận công việc.
Trong Kafka, mỗi partition có một broker leader chịu trách nhiệm đọc và ghi toàn bộ dữ liệu của partition đó. Các broker khác giữ replica là follower. Cơ chế bầu leader diễn ra tự động bằng ZooKeeper (trước Kafka 2.8) hoặc KRaft controller nội bộ (từ 2.8 trở đi). Khi broker leader bị lỗi: - ZooKeeper/KRaft nhận biết broker leader đã chết. - Chọn một leader mới từ các replica đồng bộ (ISR). - Producer và consumer được thông báo leader mới và tiếp tục hoạt động. Cơ chế này đảm bảo tính sẵn sàng cao, chịu lỗi, và cluster vẫn phục vụ dữ liệu ngay cả khi một broker gặp sự cố.
Trong Kafka, ISR (In-Sync Replica) là một replica của partition mà luôn đồng bộ với leader. Điều này có nghĩa là nó đã có tất cả các message mà leader đã xác nhận. Chỉ những replica trong ISR mới đủ điều kiện trở thành leader khi leader hiện tại gặp sự cố. ISR đảm bảo độ tin cậy và khả năng chịu lỗi, vì ngay cả khi broker leader chết, một replica trong ISR vẫn giữ dữ liệu mới nhất.

Kafka đảm bảo thứ tự message trong một partition, không đảm bảo thứ tự trên toàn topic. Khi producer gửi message đến một partition, chúng được append theo thứ tự. Consumer đọc message theo thứ tự tuần tự từ log partition sử dụng offset.
Nếu cần giữ thứ tự trên nhiều partition, producer có thể dùng key, để tất cả message cùng key luôn đi vào cùng một partition. Nhờ đó, message cùng key luôn được xử lý theo thứ tự.

Trong Kafka, consumer sử dụng cơ chế pull-based để đọc message. Điều này có nghĩa là consumer chủ động yêu cầu message từ broker theo tốc độ của riêng mình, thay vì broker push message đến consumer.
Cơ chế này có một số lợi ích:
1. Consumer có thể kiểm soát tốc độ tiêu thụ message.
2. Giúp cân bằng tải giữa các consumer trong consumer group.
3. Cho phép batch message hiệu quả, giảm overhead mạng.
Nhờ đó, Kafka cung cấp cho consumer linh hoạt và khả năng mở rộng, vì consumer chủ động pull message thay vì bị push.

Trong Kafka, tình trạng backpressure từ consumer chậm được xử lý tự nhiên nhờ cơ chế pull-based. Consumer fetch message theo tốc độ của mình, nên nếu consumer chậm, nó chỉ pull message chậm hơn.
Ngoài ra:
1. Kafka giữ message trên đĩa theo retention policy, nên consumer chậm vẫn có thể catch up sau.
2. Consumer có thể commit offset thủ công, kiểm soát khi nào message được coi là đã tiêu thụ.
3. Broker không bị quá tải vì message không bị push, nên producer vẫn gửi dữ liệu với tốc độ cao.

Kafka thực hiện exactly-once semantics (EOS) trong stream processing bằng cách dùng producer idempotent và transaction.
1. Idempotent producer đảm bảo rằng ngay cả khi producer retry gửi message do lỗi, message vẫn chỉ được ghi một lần vào partition.
2. Transaction cho phép producer ghi đồng bộ vào nhiều partition hoặc topic. Consumer đọc các topic này ở chế độ read_committed chỉ thấy các transaction đã commit hoàn toàn.
Sự kết hợp này đảm bảo message không bị mất và không bị trùng lặp, ngay cả trong xử lý stream phân tán.

Kafka và các message broker truyền thống như RabbitMQ hoặc ActiveMQ đều dùng để gửi nhận message, nhưng khác nhau về kiến trúc, mục tiêu thiết kế và use case.
- Kafka là một nền tảng streaming phân tán thiết kế cho high-throughput, log phân vùng và lưu trữ bền vững. Nó dùng cơ chế pull-based, message được lưu theo retention policy, và nhiều consumer có thể đọc cùng một message độc lập. Kafka phù hợp cho stream processing, event sourcing và pipeline dữ liệu.
- RabbitMQ / ActiveMQ là message broker truyền thống thiết kế cho giao dịch và messaging đáng tin cậy. Chúng thường push-based, message bị xóa sau khi tiêu thụ (trừ khi cấu hình khác), tập trung vào exactly-once delivery, routing, và các pattern messaging linh hoạt. Chúng phù hợp cho transactional system, task queue và request-response messaging.

Để theo dõi hiệu năng Kafka, thường sử dụng JMX metrics, các công cụ như Prometheus + Grafana, hoặc nền tảng Kafka như Confluent Control Center. Những metric quan trọng gồm:
1. Broker metrics:
- Under-replicated partitions – số partition chưa đồng bộ replica.
- Active controller count – đảm bảo chỉ có 1 controller đang hoạt động.
2. Producer metrics:
- Request latency – thời gian gửi message.
- Record send rate – số message mỗi giây được producer gửi.
3. Consumer metrics:
- Lag – số message consumer đang bị lùi so với producer.
- Fetch rate and latency – tốc độ đọc message.
4. Topic & partition metrics:
- Log size – dung lượng dữ liệu đang lưu trữ.
- Bytes in/out per second – throughput của topic.
Theo dõi các metric này giúp đảm bảo tính sẵn sàng cao, độ trễ thấp, và cân bằng tải trong Kafka cluster.

Kafka Streams là một thư viện để xây dựng ứng dụng xử lý dữ liệu streaming thời gian thực trực tiếp trên Kafka. Nó cho phép bạn xử lý, biến đổi, tổng hợp và join dữ liệu từ các Kafka topic với đảm bảo exactly-once hoặc at-least-once.
Khác với ứng dụng consumer thông thường:
1. Kafka Streams cung cấp quản lý state, windowing và aggregation sẵn, bạn không cần tự cài đặt.
2. Nó tự động xử lý song song, mở rộng và chịu lỗi giữa các instance.
3. Ứng dụng consumer thông thường chỉ tiêu thụ message, xử lý và có thể produce output, nhưng không có các tính năng stream processing nâng cao.

Để thiết kế hệ thống phân tích thời gian thực bằng Kafka, tôi sẽ làm như sau:
1. Data ingestion: Tất cả sự kiện từ ứng dụng, service, hoặc database được gửi vào Kafka topic bởi producer. Topic có thể chia partition theo loại sự kiện, user ID hoặc key khác.
2. Stream processing: Dùng Kafka Streams hoặc engine như Flink/Spark Streaming để tiêu thụ message, biến đổi dữ liệu, enrich sự kiện và tổng hợp số liệu (ví dụ: count, average, rolling window).
3. State management: Duy trì stateful computation cho các aggregation hoặc join. Kafka Streams cung cấp state store để lưu trạng thái này.
4. Data storage: Lưu kết quả đã xử lý vào cơ sở dữ liệu nhanh như Elasticsearch, Redis hoặc DB để dashboard và truy vấn analytics.
5. Downstream services: Dashboard, hệ thống cảnh báo hoặc báo cáo subscribe vào processed topic hoặc query storage để hiển thị dữ liệu thời gian thực.
6. Scalability & fault tolerance: Chia partition phù hợp, dùng consumer group để xử lý song song, enable replication, và cấu hình retention policy để đảm bảo throughput cao, reliability, và exactly-once nếu cần.
Thiết kế này cho phép metrics thời gian thực, insight theo sự kiện, và phản ứng nhanh với các tình huống kinh doanh.

Trong Kafka, việc replay dữ liệu khá đơn giản vì message được lưu trên đĩa theo retention policy có thể cấu hình. Để reprocess dữ liệu:
1. Reset consumer offset: Bạn có thể đặt lại offset của consumer về một điểm cũ trong topic (ví dụ: từ đầu hoặc một offset cụ thể) để consumer đọc lại message.
2. Dùng consumer group mới: Tạo một consumer group mới cho phép reprocess độc lập mà không ảnh hưởng đến consumer hiện tại.
3. Lưu trữ topic nếu cần: Để replay lâu dài, bạn có thể giữ message cũ trong topic riêng hoặc storage và đọc lại sau.
Cách này hữu ích để sửa lỗi, tái tạo aggregate, hoặc đưa dữ liệu vào pipeline analytics mới mà không mất dữ liệu.

Để bảo mật Kafka, chúng ta tập trung vào ba khía cạnh chính: xác thực, phân quyền và mã hóa:
1. Xác thực (Authentication): Xác minh danh tính của client và broker bằng SASL hoặc SSL certificate, đảm bảo chỉ producer, consumer và broker được phép mới kết nối được.
2. Phân quyền (Authorization): Kiểm soát quyền của client bằng ACL (Access Control List). Ví dụ, cho phép một client chỉ produce vào topic nhưng không consume, hoặc ngược lại.
3. Mã hóa (Encryption): Bảo vệ dữ liệu khi truyền qua mạng bằng SSL/TLS, mã hóa message giữa producer, broker và consumer. Có thể bật encryption at rest sử dụng storage hoặc hệ thống OS.
Kết hợp các cơ chế này, cluster Kafka sẽ được bảo vệ khỏi truy cập trái phép, dữ liệu bị thay đổi và nghe trộm.

Trong Kafka, số lượng partition ảnh hưởng đến throughput, parallelism và độ phức tạp vận hành.
Nhiều partition nhỏ:
- Ưu điểm: Song song cao, cân bằng tải tốt giữa các consumer, throughput cao.
- Nhược điểm: Broker phải quản lý nhiều metadata, overhead controller cao, thời gian bầu leader lâu hơn, có thể gây áp lực đĩa và bộ nhớ.
Ít partition lớn:
- Ưu điểm: Quản lý đơn giản hơn, controller load thấp, dễ replication và failover.
- Nhược điểm: Parallelism hạn chế, throughput thấp hơn, ít consumer slot để scale.
Vì vậy, trade-off là giữa hiệu năng và khả năng mở rộng so với đơn giản trong vận hành và quản lý. Bạn phải chọn số lượng partition phù hợp với workload và số consumer.

Để mở rộng Kafka xử lý hàng triệu message mỗi giây, tôi sẽ thực hiện các chiến lược sau:
1. Tăng số partition: Nhiều partition hơn giúp song song cao hơn, cho phép nhiều consumer đọc và nhiều broker ghi cùng lúc.
2. Thêm broker: Phân phối partition trên nhiều broker để chia tải và cải thiện throughput.
3. Tối ưu producer: Bật batching, compression và gửi bất đồng bộ để giảm overhead mạng.
4. Tối ưu consumer: Dùng consumer group, multi-threaded consumer, và batch processing để theo kịp throughput cao.
5. Tuning replication: Giữ replication factor hợp lý để chịu lỗi nhưng không gây overhead quá lớn.
6. Tối ưu phần cứng và mạng: Dùng ổ SSD, băng thông mạng cao và đủ bộ nhớ để giảm bottleneck I/O.
7. Giám sát và tuning: Theo dõi liên tục các metric quan trọng (throughput, latency, under-replicated partition, consumer lag) và điều chỉnh cấu hình như num.partitions, batch.size, linger.ms, retention.
Kết hợp các chiến lược này giúp Kafka mở rộng ngang, xử lý lượng message rất lớn và vẫn đảm bảo độ tin cậy.

Trong dự án của chúng tôi, IBM MQ được sử dụng để xử lý các giao dịch ngân hàng quan trọng, đảm bảo tính reliable, ordered, và transactional giữa các microservice.
Các luồng microservice cụ thể:
1. Transaction Validation & Processing Service
Nhận các request giao dịch từ Gateway (user authentication & routing).
Thực hiện các bước kiểm tra tính hợp lệ: số dư tài khoản, giới hạn giao dịch, chống trùng lặp.
Nếu hợp lệ, gửi message persistent vào IBM MQ queue.
Message này bao gồm tất cả thông tin giao dịch cần thiết cho backend (ví dụ: account ID, amount, transaction type, timestamp).
2. Core Banking Backend Service
Tiêu thụ message từ IBM MQ queue theo thứ tự FIFO để đảm bảo tính tuần tự trong xử lý giao dịch.
Thực hiện các thao tác trên database (IBM Db2 / PostgreSQL), ví dụ cập nhật số dư, ghi lịch sử giao dịch.
Sau khi thành công, gửi acknowledgment cho MQ để xóa message khỏi queue.
Trong trường hợp xử lý thất bại, transaction được rollback, message vẫn tồn tại trên queue hoặc được chuyển tới Dead Letter Queue (DLQ).

Lý do sử dụng IBM MQ:
Đảm bảo exactly-once processing cho các giao dịch tài chính quan trọng.
Persistent messages giúp không mất dữ liệu, ngay cả khi hệ thống gặp lỗi.
FIFO queue giữ thứ tự giao dịch, quan trọng trong core banking.
Hỗ trợ transactional integrity, rollback khi có lỗi, tránh lỗi trùng lặp hoặc mất dữ liệu.

Nếu sử dụng Kafka cho luồng này:
Kafka chỉ đảm bảo thứ tự trong cùng partition, nên nếu các giao dịch liên quan đến cùng tài khoản bị chia ra nhiều partition, thứ tự có thể bị phá vỡ → số dư sai hoặc transaction bị ghi nhầm.
Kafka yêu cầu cấu hình transactional producer và consumer phức tạp để đạt exactly-once, dễ lỗi trong môi trường banking.
Message không được persist ngay lập tức nếu producer gửi không đúng cấu hình → rủi ro mất giao dịch khi broker crash.
Ví dụ cụ thể:
Nếu một khách hàng thực hiện 2 giao dịch chuyển tiền liên tiếp:
Chuyển 1.000 USD → gửi Kafka topic
Chuyển 500 USD → gửi Kafka topic
Nếu hai message này rơi vào 2 partition khác nhau, consumer đọc không theo thứ tự → số dư hiện tại có thể sai → ngân hàng ghi nhầm số dư → lỗi tài chính nghiêm trọng.

Trong khi đó, Kafka được sử dụng cho các dịch vụ dữ liệu và đồng bộ thời gian thực, nơi throughput cao và một mức độ độ trễ nhỏ có thể chấp nhận được. Kafka giúp tách biệt các luồng non-critical khỏi critical transaction flow, tránh làm tắc nghẽn hệ thống.
1. Data Aggregation Service
Theo dõi các thay đổi từ các hệ thống backend hoặc IBM MQ, ví dụ: số dư tài khoản, log giao dịch, sự kiện audit.
Chuẩn hóa dữ liệu, enrich nếu cần (thêm metadata, timestamp chuẩn hóa…).
Publish các update lên Kafka topics, phân chia theo loại dữ liệu (transaction log topic, account balance topic, audit events topic).
2. Downstream Services (Analytics, Reporting, View Update)
Subscribe Kafka topics để tiêu thụ message bất đồng bộ.
Xử lý dữ liệu, tính toán hoặc chuyển đổi nếu cần.
Update dữ liệu vào Elasticsearch để hỗ trợ tìm kiếm, dashboard và báo cáo realtime.
Gửi yêu cầu đến BroadcastService, nơi đang quản lý các kết nối realtime với UI client (WebSocket / SSE / push notifications).
Kafka cho phép parallel consumption qua các consumer group, giúp xử lý dữ liệu lớn mà không ảnh hưởng tới luồng transaction chính.
3. UI Client và ViewService
UI client nhận thông báo realtime từ BroadcastService, ví dụ: số dư cập nhật, transaction mới.
Khi cần hiển thị chi tiết, UI client gọi ViewService.
ViewService truy vấn Elasticsearch, trả dữ liệu đã chuẩn hóa cho UI client, đảm bảo hiển thị thông tin chính xác và realtime.

Lý do sử dụng Kafka:
Hỗ trợ streaming tốc độ cao, xử lý hàng ngàn đến hàng triệu message mỗi giây.
Partitioning và consumer group giúp cân bằng tải, mở rộng theo nhu cầu.
Cho phép message replay, thuận tiện khi các dịch vụ downstream cần xử lý lại dữ liệu cũ hoặc khôi phục sau lỗi.
Tách biệt các luồng critical transaction (IBM MQ) khỏi data streaming non-critical, đảm bảo độ tin cậy và hiệu năng cho cả hệ thống.

Nếu sử dụng IBM MQ cho luồng này:
MQ persistent transactional messages sẽ tạo overhead lớn nếu publish hàng triệu event mỗi giây → throughput giảm.
MQ không tối ưu cho parallel consumption với nhiều service downstream → analytics/reporting sẽ bị tắc.
MQ không có cơ chế replay message như Kafka → nếu service downstream crash, khó xử lý lại dữ liệu lịch sử.
Ví dụ cụ thể:
Khi tất cả transaction log và account update được publish tới analytics/BI mỗi giây:
Nếu dùng IBM MQ, queue sẽ chậm hoặc backlog → dashboard hiển thị dữ liệu trễ hàng phút, không realtime.
Các downstream service muốn replay dữ liệu cũ để rebuild báo cáo sẽ gặp khó khăn vì MQ không dễ replay message theo thời gian như Kafka.