1. Mục đích: xây dựng các ứng dụng bằng spring boot tương tác với nhau bất đồng bộ thông qua hệ thống kafka ở giữa chứa dữ liệu được trao đổi qua lại.
2. Event-Driven Architecture in Microservice

* EDA là 1 design pattern
* Các ứng dụng khác nhau có thể bất dồng bộ, tương tác được với nhau . 1 ứng dụng đc gọi là produce tạo ra các events/messages gửi lên 1 ứng dụng trung gian gọi event broker/messahe broker. Sau đó có nhiều ứng dụng khác gọi là consume – cái tiêu thụ event/message này gọi là consumer
  + Các ứng dụng này có thể tương tác với nhau gửi và nhận events/message 1 cách bất đồng bộ
* Ví dụ:
  + Order Service(Producer) Order Event Message Broker(Ứng dụng trung gian)

Order Event Shipment Service(Consumer)

Order Event Email Service(Consumer)

* Event-Driven có thể viết trong bất cứ ngôn ngữ nào cx đc. Các ứng đụng đc tương tác với nhau bất đồng bộ
* Tác dụng Event-Driven so với API
  + Bất đồng bộ với nhau
  + Message Broker dễ dàng phân tán, nâng cao hiệu suất cũng như độ tin tưởng và khả năng mở rộng
  + Vì là 1 ứng dụng chạy độc lập nên có nhiều opensource: Apache Kafka, RabbitMQ, ActiveMQ….
  + Phổ biến là Apache Kafka vì có thể scale tốt trong việc xử lý 1 lượng lớn events/messages đc gửi lên

1. Kafka Architecture

* Kafka Cluster là tập hợp của nhiều broker Server chạy độc lập với nhau trên môi trường khác nhau 🡪 Đảm bảo các server ko bị down cùng 1 lúc.
  + Ví dụ: Broker 1 down thì dung Broker 2,3.. thay thế (Tối thiếu phải 3 con trở lên trong env production)
* Zoo keeper là 1 phần mềm để quản lý tất cả mọi thứ của Kafka như cluster, brokers, topics, offset, events,…
* Event là bất cứ dữ liệu nào diễn ra theo 1 thời gian (Giống như 1 bảng ghi trong db)
* Producer là 1 nhóm các ứng dụng gửi các events lên những broker, thường gửi lên topic
* Consumer là những ứng dụng sẽ đọc events này về từ topics trong brokers.
* Bản chất Producer với Consumer chạy độc lập với nhau, Kafka ở giữa đóng vai trò nhận events và consumer đọc events về và xử lý 🡪 Bất đồng bộ thông qua Kafka
* Đặc điểm:
  + Khả năng chính: bang thông cao, khả năng mở rộng, khả năng lưu lại events, khả năng tồn tại cao
  + Ecosytem: Xử lý luồng đc tích hợp, kết nối với nhiều dữ liệu, nhiều thư viên, nhiều tool opensurce
  + Sự tin tường & dễ dùng: dung cho mục đích quan trong, nhiều công ty dùng làm hệ thống nhận gửi event, cộng đồng đông, tài liệu nhiều
* Use Case:
  + Traditonal messaging sytems
  + Theo dõi những hoạt động tracking của web
  + Metrics đo lường
  + Phân tích log
  + Xử lý luồng
  + Đón nhận event source như nguồn sữ liệu từ thiết bị iot, camera,….

1. Kafka Core Concept
   1. Kafka là 1 event streaming platform

* Event là bất cứ dữ liệu gì được ghi lại trong thời gian thực. (giống như bản ghi trong db)
* Even Stream là 1 luồng dữ liệu theo tuần tự thời gian từ nhiều nguồn dữ lieu khác nhau như db, iot, apps,…
* Kafka lưu lại những events trong file để có thể truy vấn đọc lại…(giống db nhưng Kafka dung file để lưu)
* Kafka sử dụng để:
  + Pusblish (ghi) và subscribe (đọc) luồng dữ liệu, events bao gồm liên tục ghi/đọctrên dữ liệu từ các hệ thống khác nhau 🡪 tradional messaging
  + Lưu lại luồng sự kiện lâu dài đến khi nào xóa đi
  + Xử lý luồng theo thời gian thực
  1. Kafka Cluster
* Kafka là 1 hệ thống phân tán cahyj như 1 cluster
* Kafka Cluster là tập hợp của nhiều broker server (>=3 thực tế) dễ dàng mở rộng brokers lên các trung tâm dữ liệu khác nhau cũng như có những server để chạy kafka connect kết nối với client 🡪 Sever liên kết với nhau tạo thành 1 Cluster để đảm bảo Kafka hoạt động ổn định
* Kafka Cluster đảm bảo mở rộng cao và khả năng chịu lỗi. Bất cứ server nào trong cluster bị sập thì những con service khác đc thay thế đảm bảo dữ liệu đc ghi/đọc liên tục và ko bi mất dữ liệu.
  1. Kafka Broker
* Broker thực chất là 1 server mang hình thức của 1 lớp lưu dữ liệu, lưu sự kiện ở trong file ở các kafka broker lâu như mong muốn.
* Applications(Producer và Consumer) sử dụng Kafka Broker để trao đổi dữ liệu(message) với nhau 1 cách bất đồng bộ. Chúng ko trực tiếp liên lạc với nhau mà sử dụng broker để gửi dữ liệu và consumer đọc về.
* Broker tổ chức events vào 1 or nhiều topic để lưu và phân phối dữ liệu dễ dàng hơn.
  1. Producer và Consumer
* Producer là ứng dụng client dùng để gửi events đến broker server
* Consumer dung để đọc events
* Cả 2 đều là nhưng ứng dụng chạy độc lập ko liên quan với nhau ví dụ code Producer cho Java, Consumer cho NodeJs và cả 2 ứng dụng này có thể truyền tải dữ liệu qua lại qua Kafka. Khi chuyển dữ liệu từ producer 🡪 Kafka ta phải có thư viện Serializer để convert đối tượng về chuỗi byte để Kafka lưu và truyền tỉa dữ liệu và Consumer phải có Deserializer để convert luồng byte về dạng đối tượng để sử dụng
* Kafka ddc thiết kế để đảm bảo truyền tải dữ liệu 1 cách tối ưu nhất
  1. Topic
* Topic giống như 1 danh mục để bố trí các events trong 1 broker (giống như bảng trong db or folder)
* Topic đc chỉ ra 1 cái tên
* 1 Broker Server chứa nhiều topic khác nhau
* 1 topic có nhiều ứng dụng produce gửi events đên và consumer đọc về
  1. Kafka Topic Partitions
* Topic chia thành nhiều phần nhỏ hơn để chứa dữ liệu gọi là partition
* Partition đc phân phối ở broker khác nhau đảm bảo dũ liệu đc phân tán, dễ dang mở rộng cũng như client đọc ghi dữ liệu ở broker khác nhau cùng 1 thời gian
* Khi Producer gửi event lên topic thì event đc găn vào những partition của topic (event gắn vào p1 rồi thì p2,3,4.. sẽ ko có event này) 🡪 Id sẽ đc genr ra và gắn vào event đấy(giống id trong db) và id đấy là cố định ko thay đổi. Consumer sẽ đọc về theo cùng 1 thứ tự như vậy
* Gửi event lên 1 key giống nhau thì event đấy luôn gắn vào 1 partition xác định từ trước. Nếu gửi event ko với key thì event sẽ đc ngẫu nhiên phân phối vào các partition trong topic
  1. Kafka Event’s Offset
* Event Offset hay là Id đc găn vào event khi nó gửi đến 1 partition trong topic, dung để xác định vị trí của event trong partition của 1 topic
* Event gắn ra rồi thì ko bao giờ thay đổi
  1. Kafka Partition’s Replication
* Replication có nghĩa là sao chép lại 1 partition trong topic thành n lần và triển khai ở các broker khác trong kafka cluster
* 1 partition là leader, partition là folowers. Followers sẽ copy dữ liệu đồng bộ với leader và producer sẽ ghi event đến partition leader
* Khi partition leader bị fail (Server broker 1 bị sập) thì nhanh chóng chọn broker 2,3…
* Cơ chế này đảm bảo dữ liệu luôn luôn online đối client application ko bị gián đoạn
* Trong production nên để số lượng broker là số lẻ

**#### 1. ACK**

- Là 1 tín hiệu từ đầu nhận -> message được nhận (đã nhận và xử lý thành công)

- ACKS là số acknowledgements mà đầu producer yêu cầu leader broker phải nhận được trước khi 1 record được xử lý thành công.

- TH1: ACKS = none (0)

    - Producers ko yêu cầu ACK từ leader ("Fire and forget")

    - Ưu: Performant tốt nhất vì ko cần đợi ACKS response từ phía broker trả về để thực hiện message tiếp

    - Nhược: 1 số TH lỗi message bị lỗi đầu producer ko gửi lại -> message mất. Khi mà gửi record vào đầu broker thì ghi record vào partition thì record mới có offset -> record ở đầu producer sẽ ko có thông tin về offset -> Durability yếu nhất

    - Use cases: Phù hợp với metrics từ các iot: thu thập độ ẩm, sáng, log ko quan trg....

    - Các records vẫn đc sao chép qua các follower 1 cách bất đồng bộ

- TH2: ACKS = One (1)

    - Producers chỉ yêu cầu 1 ACK từ phía leader broker

    - Ưu: High Performance, có offset từ đầu producer

    - Nhược: Durability yếu, messages có thể bị mất

    - Use case: Phù hợp với tracking hành động người dùng, clicks, views

    - Các records vẫn đc sao chép qua các follower 1 cách bất đồng bộ

- TH3: ACKS = ALL (-1)

    - Producer gửi gói tin vào  leader broker -> ghi record vào partition và replicate sang các follewer và chơ follower (các In-Sync Replicas - ISRs) trả về ACK -> Broker leader mới trả ACK cho Producer

    - Replication ko bất đồng bộ.

    - Ưu: Durability mạnh

    - Nhược: High Latency

    - Use case: hành động liên quan đến tài chính, order, payment

**#### 2. Retry**

- Producer gửi gói tín lỗi ko bt -> retry

- Loại lỗi:

    - Retriable error: LEADER\_NOT\_AVAILABLE, các issues liên quan đến network

    - Non-Retriable error: INVALID\_CONFIG, message quá lớn

- 2 cách xử lý:

    - Auto: retry N lần -> give up

    - bằng tay

- Mặc định tries:

    - Kafka <= 2.0: 0

    - Kafka >= 2.1: 2147583647

- Retry.backoff.ms: 100ms

- Producers retry trong 2 phút (delivery.timeout.ms=120000)

- Nên để retry 30 lần

- Retry -> có thể gây Duplicate messages và Ordering in producer side

**#### 3. Batching (Bottleneck của Kafka Disk nằm ở network)**

- Gom lại nhiều records thành 1 batch rồi mới gửi đi

- 1 cơ chế giúp cải thiện network bandwidth và thông lượng của Kafka

- Batching đc thực hiện bởi producer

- 1 write request chứa 1 batch

- 1 batch đc ghi vào 1 partition

- Batching đc kiểm soát bởi 2 cấu hình:

    - linger.ms: khoảng thời gian produvcer đợi trc khi gửi 1 batch

    - batch.size

- linger.ms:

    - Producer đọi 1 TG delay để batch trước khi gửi đi

    - Thêm đỗ trễ latency

    - Giá trị mặc đinh: 0 -> producers gửi càng nhanh càng tốt

    - nếu linger.ms = 0, batching vẫn hoạt động

    - nếu linger.ms > 0, tăng tuần suất tỉ lệ batch

- batch.size:

    - Kích cỡ tối đa của Batch

    - Giá trí mặc định = 16KB

    - 1 batch quá lớn có thể tốn nhiều memory

    - 1 message > batch size thì message ko đc batch

- Trade-off: Throughput vs latency

- Broker ghi 1 record vào đĩa nhưng ko unpack gỡ batch ra -> cải thiện disk I/

**#### 4. Compression (Bacth có thể lớn -> compress or split)**

- Cơ ché nén có tác động tích cực đến hiệu năng của hệ thống nói chung

    - Cai thiện network bandwith

    - Tiết kiệm ko gian lưu trữ

    - latency sẽ giảm đi

- Kết hợp với batching thì tỉ lệ nén có thể lớn hơn

- Nhược: Tốn CPU ở producer và consumer

- Note: Compression != Encryption

- Loại:

    - None: (Mặc định) producer ko nén

    - gzip

    - snappy

    - Lz4

    - zstd

- Recommend:

    - Lz4: normal case

    - zstd: throughput cao

- End-to-End Concern:

    - Đầu tiên producer nén batch lại -> ghi vào broker (Leader partition) -> Consumer giải nén batch và đọc record

    - Nhược: Thuật toán nén của producer phải tương thích version của consumer

**#### 5. Sticky Partioner - logic producer -> xác định 1 message đc ghi vào partiton nào**

- Đối message ko có key thì chọn 1 partiton -> tạo 1 batch tương ứng với partition đo -> ghi các message ko có key vảo batch -> đến khi batch đấy hoặc linger thì batch này đc đóng gói gửi lên partiton -> producer chọn random partition tiếp theo -> tạo ra batch mới -> ghi các record ko có key vào partiton mới...

- Giải quyết đc sô lượng batch

- Đảm bảo dữ liệu messsage trải đều partiton

**#### 6. Max In Flight Requests - giúp cho producer cải thiện throutput**

- max.in.flight.requests.per.connection: số request tối đa chưa nhận đc acks

- Giá trị mặc định = 5

- Cải thiện Throughput nhưng ko cải thiện network bandwith

**#### 7. Idempotent**

- Khi làm việc với collections thì cần quan tâm: Record duplication và Out of order -> Idempotent

- Idempotent: thao tác thực hiện nhiều lần ko thay đổi kết quả

- Cách hoạt động của idempotent producer

    - Kafka assign cho producer 1 producer ID (PID)

    - Producer duy trì tăng dần Ví dụ:

    ![alt text](dd.png)

- Cách triển khai:

    - enable.idempotent  = true (recommended) - default với kafka 3.0

    - Kết hợp với

        - acks=all

        - retries=INT\_MAX

        - max.in.flight.requests=5

- Note: Chỉ giải quyết order ở producer chứ ko ở consumer