**Báo cáo tìm hiểu Kafka**

**1. Tổng quan về Kafka**

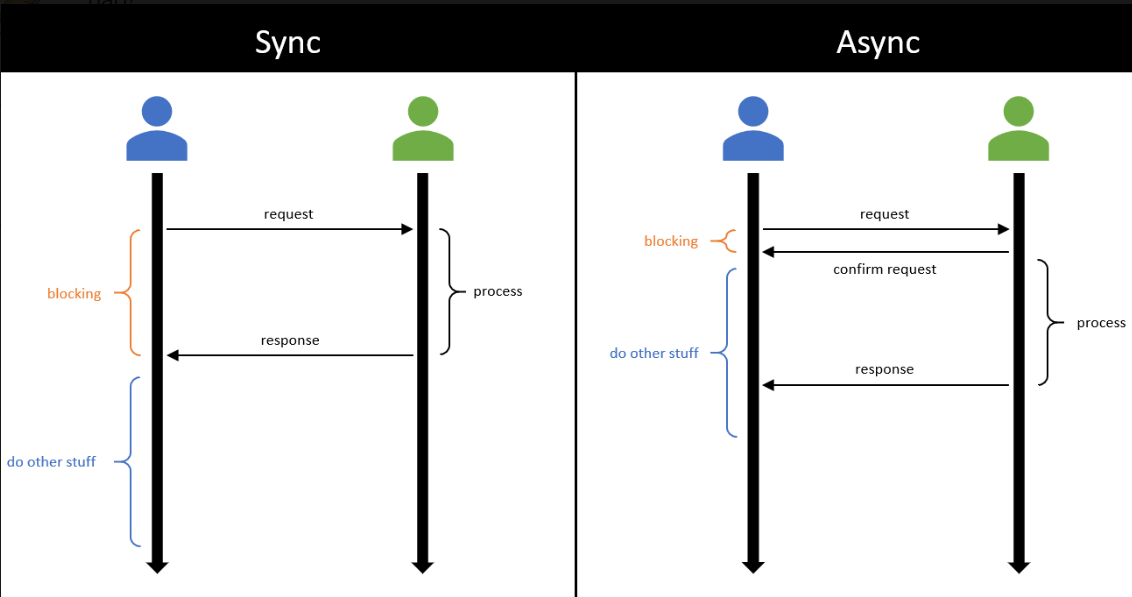
**Là message broker và hoạt động dựa trên 2 nguyên tắc chính Asynchronous programming và Message-driven programming**

* 1. **Asynchronous programming**

**Asynchronous programming: trong việc giao tiếp giữa các service khác nhau đều muốn tránh blocking khi xử lý request và cách giải quyết bài toán ở đây sẽ là Tách biệt Request và Response**

**Tách biệt hai luồng request và response, nghĩa là hoàn toàn có thể thực hiện giao tiếp trực tiếp giữa nhiều service với nhau.**

* **Service A gửi request đến Service B.**
* **Service B nhận request và phản hồi đã nhận được request. Để đó xử lý sau.**
* **Trong lúc đó, Service A hoàn toàn rảnh rang thực hiện các công việc khác.**
* **Sau khi Service B xử lý xong request sẽ gửi ngược lại cho Service A.**

****

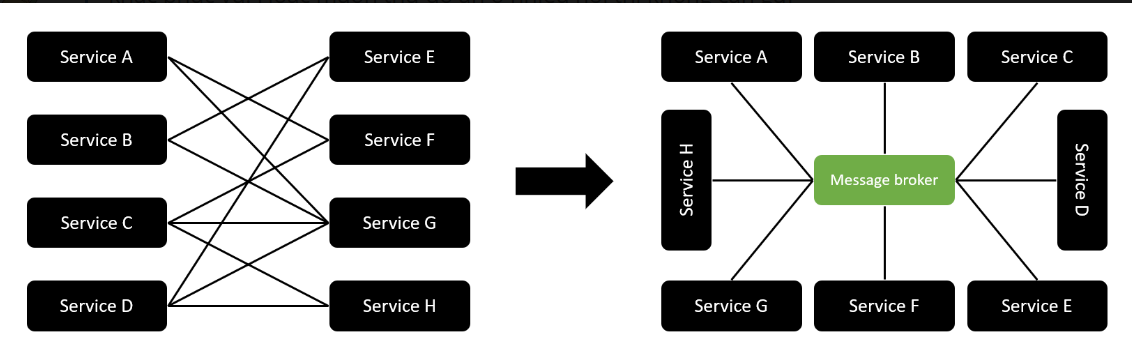
**1.2 Message-driven programming**

**Message-driven programming : Server và client không giao tiếp trực tiếp với nhau. Tất cả các request sẽ được gửi dưới dạng message cho bên thứ 3. Bên thứ 3 có nhiệm vụ điều hướng các message đến địa chỉ cụ thể với 2 mục tiêu:**

* **Đảm bảo gửi message thành công.**
* **Và gửi đến đúng địa chỉ.**

**Mục đích chính của message broker là : hướng, trung chuyển message từ người gửi đến người nhận với những lợi ích là :**

* **Giảm tải cho các server bằng việc giảm các tương tác trực tiếp.**
* **Lưu trữ request, trong trường hợp server gặp sự cố.**
* **Phân phối request đến các nhiều server trong các bài toán cụ thể.**
* **Đơn giản hóa quá trình gửi nhận message trong môi trường multi-services.**

****

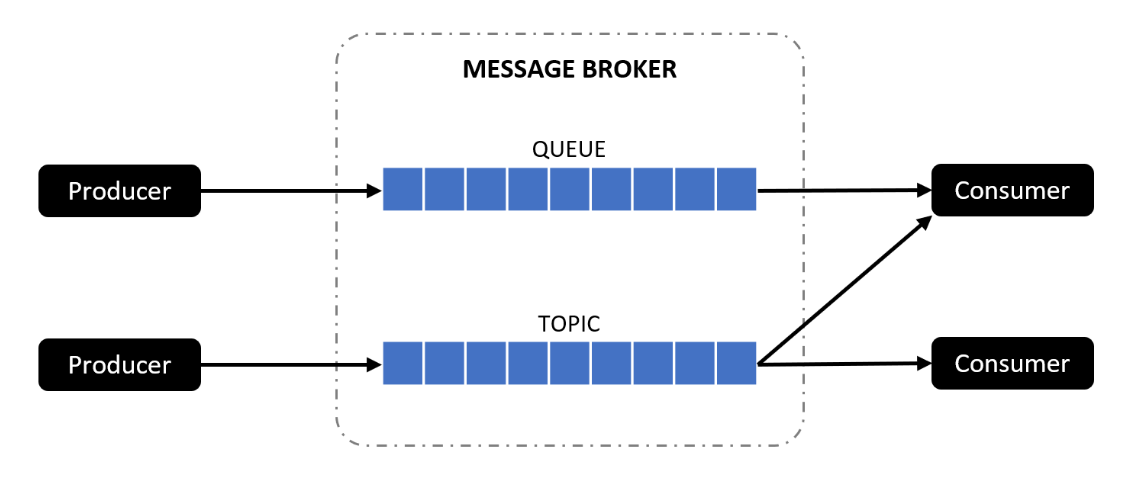
**1.3 Message Broker**

**Message Broker cung cấp 2 Patterns:**

* **Point-to-point messaging: Queue, phân phối message theo quan hệ 1:1**
* **Broadcast messaging: Topic, một message có thể được gửi tới nhiều địa chỉ khác nhau, chỉ những người subcribe nội dung đó mới nhận được message**

**Message Broker được đặt trong hệ thống sẽ bao gồm với bản chất vẫn là 1 bên gửi 1 bên nhận**

* **Producer/Publisher: nơi gửi message.**
* **Message broker: hệ thống điều hướng message.**
* **Consumer/Subcriber: nơi nhận message.**

****

**2. Kafka cơ chế lưu Message**

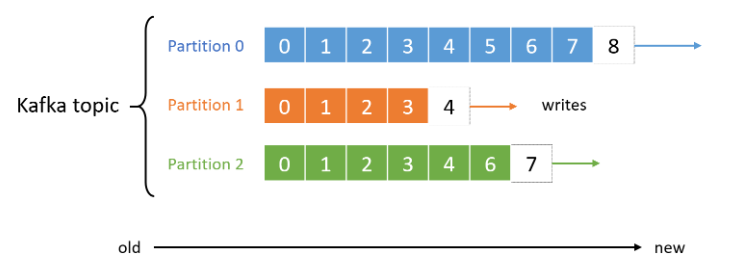
**2.1 Topic, partition và offset**

**1 message trong Kafka được xác định bởi 3 yếu tố topic name, partition, offset**

**Topic: là stream of Data, luồng lưu trữ dữ liệu của Kafka, là 1 dãy các message nối tiếp nhau**

**Partition: Topic sẽ chưa 1 hoặc nhiều partition và message được lưu trên đó**

**Offset: Các message trong partition được order theo thứ tự từ cũ đến mới được gọi là Offset**

****

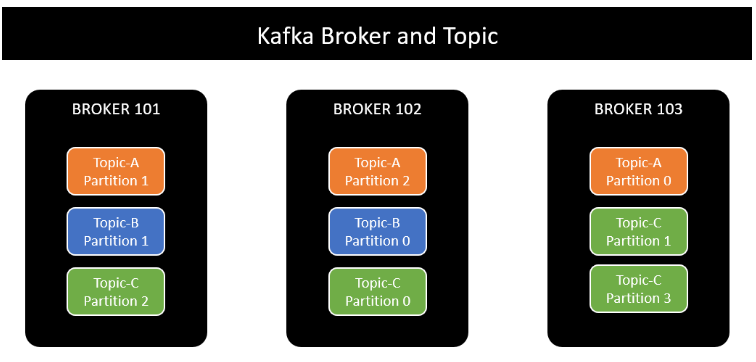
## ****2.2 Kafka broker****

**Topic được lưu trữ trên file, trên disk, và tất cả đều được lưu trữ trên server. Và server là một Kafka broker trong Kafka cluster**

**Kafka cluster là nhiều server tập trung lại thành một cụm làm việc với nhau - multi-brokers (multi-servers). Broker bao gồm 3 nhiệm vụ:**

* **Receive message từ Producer và ack.**
* **Lưu message tại log file để đảm bảo an toàn, tránh trường hợp mất message.**
* **Gửi message đến Consumer trong trường hợp được yêu cầu.**

**Để đảm bảo high reliable, Kafka tự động phân tán các partition trên tất cả broker đang có. Mỗi partition nằm trên một broker. Partition của Topic có thể nằm trên bất kì broker nào mà không phụ thuộc thứ tự.**

****

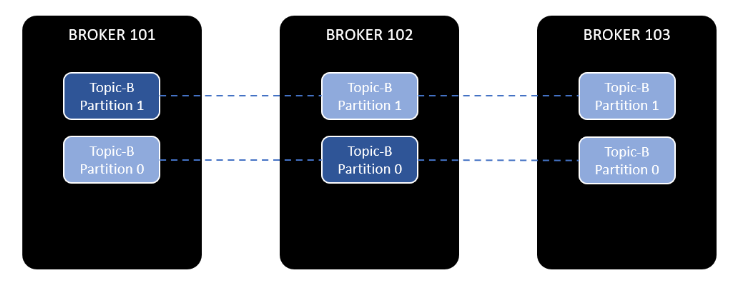
## ****2.3 Topic replication****

**Đối với cách tổ chức phân tán các partition trên các broker, mỗi partition nằm trên 1 broker thì có thể gặp phải single-point failure**

**Ví dụ: nếu 1 broker chứa 1 partition bị sập thì sẽ mất message**

**Thì cơ chế xử lý vấn đề này sẽ là tạo ra nhiều bản giao partition và lưu trên các broker khác thông qua Replication Factor**

**Trong trường hợp 1 broker bị sập thì đã có back-up từ các broker còn lại**

****

**Để đảm bảo yếu tố performance và tránh rắc rối khi đồng bộ message giữa các replication, Kafka lựa chọn implement theo hướng Leader - Follower, chỉ định một và chỉ một replication bất kì làm leader. Như vậy, tất cả các thao tác read/write message của partition đều thông qua replication leader.**

**3. Gửi nhận message trong Kafka**

### **3.1 Producer**

**Producer là người gửi message đến Message broker. Cụ thể với Kafka, producer write data vào partition của topic.**

**Producer tự động biết nên write vào broker nào và partition nào. Trong trường hợp đang write data mà broker gặp sự cố, producer sẽ có cơ chế tự động retry cho đến khi thành công.**

**Trong trường hợp gửi message đến topic mà không chỉ định partition, producer sẽ gửi message đến broker theo cơ chế round-robin.**

**Producer: sử dụng cơ chế ack – acknowledgment để biết biết message đã được write thành công ở partition**

* **acks=0: giống fire-and-forget, gửi message mà không chờ phản hồi. Do vậy có thể dẫn đến tình huống mất message.**
* **acks=1: default setting. producer chờ cho tới khi nhận được phản hồi từ replication leader. Tuy nhiên chưa ngăn chặn hoàn toàn việc mất message. Replication leader write message thành công, báo lại cho producer, tuy nhiên broker có thể gặp sự cố với disk, không thể khôi phục data.**
* **acks=all: đảm bảo không mất message. Producer sẽ nhận được phản hồi khi tất cả replication leader và IRS write data thành công.**

## ****3.2 Consumer****

## ****3.2.1 Consumer****

* **Consumer đọc message từ topic, xác định bằng topic name.**
* **Đồng thời, consumer biết nên đọc message từ broker nào. Trong trường hợp chưa read xong mà broker gặp sự cố, consumer cũng có cơ chế tự phục hồi.**
* **Việc đọc message trong một partition diễn ra tuần tự để đảm bảo message ordering. Có nghĩa là consumer không thể đọc message offset=3 khi chưa đọc message offset=2. Message ordering chỉ đảm bảo trong một partition. Việc đọc ghi message giữa nhiều partition không đảm bảo thứ tự**
* **Một consumer cũng có thể đọc message từ một hoặc nhiều hoặc tất cả partition trong một topic**

### **3.2.2 Consumer group**

**- Các consumer sẽ thuộc cùng một nhóm được gọi là consumer group**

**- Consumer group read toàn bộ data của các partition và chia vào các consumer bên trong để xử lý.**

**- Mỗi consumer thuộc consumer group sẽ đọc toàn bộ data của một hoặc nhiều partition để đảm bảo message ordering. Không tồn tại nhiều consumer cùng đọc message từ một partition**

**- Nếu số lượng consumer trong consumer group lớn hơn số lượng partition thì những consumer dư thừa có trạng thái inactive - không nhận bất kì message nào từ topic.**

**- Trong trường hợp một active consumer gặp vấn đề và không thể tiếp tục hoạt động, một trong những inactive consumer còn lại được đẩy lên thay thế và tiếp tục công việc ngay lập tức. Nếu không có inactive consumer nào, message sẽ được route tới một active consumer bất kì khác. Quá trình re-assign này được gọi là partition rebalance.**

## ****3.2.3 Consumer offset****

**- Để consume lại các message đã xử lý, hoặc tiếp tục xử lý các message chưa được xử lý trong topic, consumer cần biết offset chính xác vị trí của message cần đọc. consumer offset giống như một checkpoint cho consumer group.**

**- Mỗi consumer group khác nhau có consumer offset khác nhau.**

**- Khi consumer xử lý xong message, chúng ta nên commit giá trị offset, các giá trị này sẽ lưu tại Kafka topic có tên là \_\_consumer\_offsets. Trong trường hợp consumer gặp sự cố và khởi động lại sau đó hoặc một consumer khác thay thế, nó có thể tiếp tục từ message cuối cùng được xử lý. Tránh trường hợp đọc lại toàn bộ message từ đầu.**

**-Các tình huống có thể xảy ra**

### **At most once: nhiều nhất 1 lần consume message**

**Consumer commit offset ngay khi nhận được message.**

**Gặp vấn đề trong trường hợp đang xử lý message mà consumer gặp sự cố và transaction chưa được commit thì Không thể consume lại message để xử lý.**

### **At least once: ít nhất 1 lần consume message**

* **Consumer commit offset sau khi xử lý xong message. Như vậy, nếu consumer gặp sự cố trong khi process message cũng không có vấn đề lớn. Consumer sau khi wake up sẽ consume và xử lý lại message đó.**
* **Gặp vấn đề trong trường hợp: trong quá trình process cần gọi đến 1 bên khác để xử lý , và bên thứ 3 đã xác nhận thay đổi dữ liệu thành công, mà consumer gặp sự cố thì phải consume lại message để xử lý thì sẽ tiếp tục gọi lại bên thứ 3 một lần nữa. sẽ gặp vấn đề**
* **Như vậy message có thể đến nhiều hơn một lần, nên được gọi là at least once. Cần rất chú ý khi xử lý những case như trên. Cần đảm bảo quá trình process là idempotent (Idempotent là những xử lý có thể lặp lại nhiều lần mà không ảnh hưởng đến hệ thống)**

### **Exactly once: chỉ consume message 1 lần duy nhất**

* **Nó chỉ xuất hiện trong case message được transfer từ Kafka sang Kafka.**
* **Có thể biến at least once thành exactly one đó là sử dụng các idempotent consumer với lý do nhận nhiều message xử lý nhiều lần nhưng giống như nhận một message xử lý một lần**

**3.3 Queue và Topic**

* **Broadcast messaging - Topic.**
* **Nếu một topic có duy nhất một consumer group, nó là point-to-point messaging - queue.**
* **Ngược lại nếu muốn implement hệ thống broadcast messaging - topic, hãy tạo ra nhiều consumer group subcribe trên cùng một topic.**

**4. Thông tin liên quan**

## ****4.1 Kafka broker discovery****

## **Kafka brokers bao gồm rất nhiều broker. Mỗi topic có thể có nhiều partition, mỗi partition được lưu trên các broker khác nhau. Tuy nhiên consumer chỉ cần connect tới một broker bất kì là có thể connect với toàn bộ Kafka cluster. Điều đó giúp consumer có thể đọc được message của topic nằm trên bất kì một broker nào**

**Mỗi Kafka broker là một bootstrap server, nó có cơ chế thông báo cho client biết làm thế nào để connect tới các broker còn lại. Như vậy khi connect tới một broker là connect tới toàn bộ Kafka brokers.**

**Bootstrap servers là một danh sách các endpoint của từng broker trong cụm cluster. Một broker có toàn bộ các thông tin của các broker còn lại.**

## ****4.2 Zookeeper****

**Zookeeper  duy trì và đảm bảo hoạt động của Kafka**

* **Lưu trữ tất cả thông tin của Kafka broker, topic, partition... Hay nói cách khác là quản lý chúng.**
* **Thực hiện leader election cho các partition.**
* **Gửi thông tin đến Kafka về các event phát sinh trong hệ thống: new topic, delete topic, partition die, broker die, broker comes up...**