**Họ và tên:** Nguyễn Quốc Hùng

**BÁO CÁO EX10 – MESSAGE QUEUE**

1. Message Queue

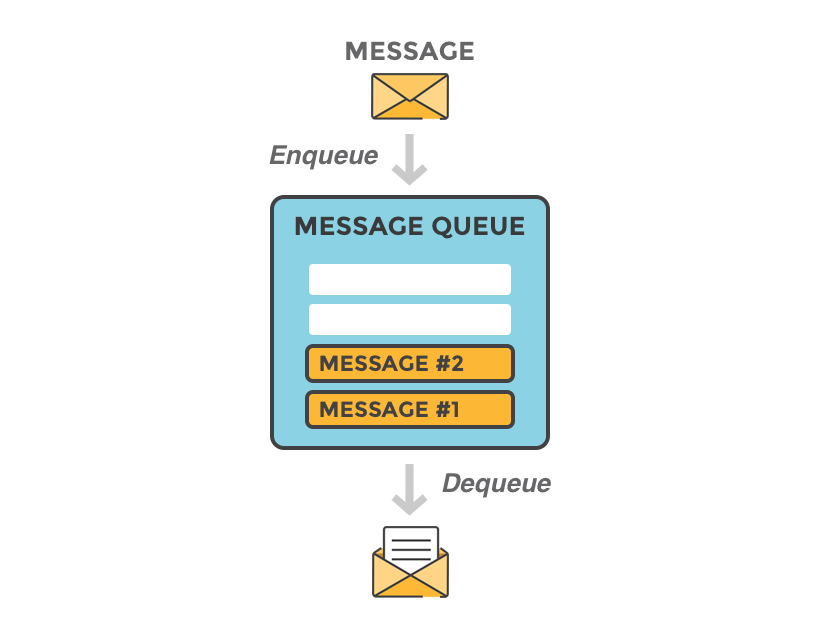
* **Message Queue là gì?**

Message Queue là một dạng giao tiếp giữa các dịch vụ không đồng bộ được sử dụng trong kiến ​​trúc serverless và microservice. Message được lưu trữ trên hàng đợi cho đến khi chúng được xử lý và xóa. Mỗi tin nhắn chỉ được xử lý một lần bởi một Consumer. Message Queue có thể được sử dụng để tách riêng quá trình xử lý nặng, xử lý đệm hoặc xử lý hàng loạt và làm trơn tru khối lượng công việc tăng vọt.

A computer screen shot of a document

Description automatically generated

* + Ý nghĩa của “queue” ở đây chính là 1 hàng đợi chứa message chờ để được xử lý tuần tự theo cơ chế vào trước thì ra trước (FIFO - First In First Out).
  + Một message là các dữ liệu cần vận chuyển giữa người gửi và người nhận.



* Message queue giống như một hòm thư email. Khi một email được gửi đi, người gửi tiếp tục xử lý những thứ khác mà không cần phản hồi ngay lập tức từ người nhận (giao tiếp không đồng bộ). Cách xử lý tin nhắn này tách người gửi khỏi người nhận để họ không cần phải tương tác với hàng đợi tin nhắn cùng một lúc.
* **Kiến trúc của Message Queue**

A line of mails and a few envelopes

Description automatically generated with medium confidence

Kiến trúc cơ bản của message queue bao gồm các thành phần:

* + **Message**: Thông tin được gửi (có thể là text, binary hoặc JSON).
  + **Producer**: Service tạo ra thông tin, đưa thông tin vào message queue.
  + **Message** **Queue**: Nơi chứa những message này, cho phép producer và consumer có thể trao đổi với nhau.
  + **Consumer**: Service nhận message từ message queue và xử lý.

Một chương trình/ service vừa có thể là producer, vừa là consumer.

* **Các loại Message Queue**

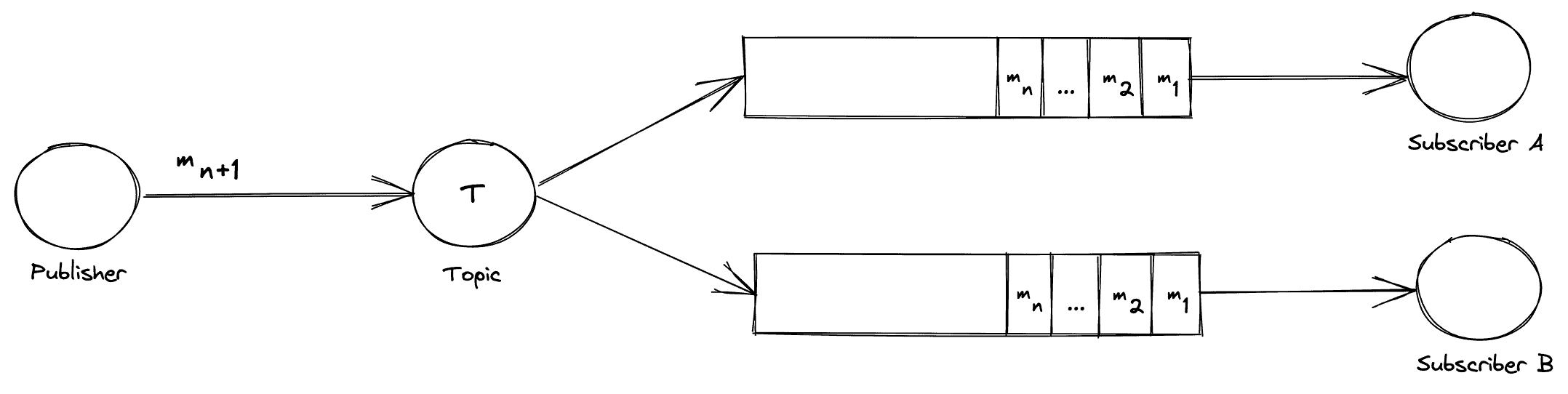
1. **Point-to-point**

A diagram of a message

Description automatically generated

Trong mô hình Point-to-point, Message được gửi từ người gửi đến chỉ một người nhận ngay cả khi có nhiều người nhận đang nghe trong cùng một Message Queue. Trong mô hình Point-to-point, thuật ngữ message sender và message receiver thường được sử dụng dụng thay vì message publisher và message consumer. Có hai loại nhắn tin Point-to-point, fire-and-forget (một chiều) và request/reply (yêu cầu-phản hồi). Sự khác biệt chúng là cách người gửi quan tâm đến trạng thái của message được gửi.

1. **Publisher-Subscriber**



Publisher (nhà sản xuất) gửi tin nhắn đến hàng đợi (trong trường hợp này được gọi là Topic) và tất cả subscriber (người đăng ký) vào cùng 1 Topic đều sẽ nhận được tin nhắn trong Topic đó.

* **Ưu điểm Message Queue**
  + **Dễ scaling hệ thống:** Vào giờ cao điểm, nhiều truy vấn, ta có thể tăng số lượng consumer lên để xử lý được nhiều messege hơn. Khi không cần ta có thể giảm lại.
  + **Phân tán hệ thống:** Giúp phân tách hệ thống thành nhiều service nhỏ hơn, mỗi service chỉ xử lý 1 chức năng nhất định theo cấu trúc microservice.
  + **Đảm bảo duration/recovery:** Do message đã được lưu trong queue, khi 1 service đang xử lý nhưng bị crash, ta không lo bị mất data vì có thể lấy message từ trong queue ra và retry. Trong 1 hệ thống có nhiều consumer, nếu vài consume crash cũng không làm crash cả hệ thống.
  + **Hỗ trợ rate limit, batch process:** Trong trường hợp khả năng xử lý của hệ thống có hạn (chỉ có thể xử lý 100 lượt release/s) mà phải xử lý 10000 lượt release. Với message queue, ta có thể lấy từng lượt release chưa xử lý trong queue ra xử lý từ từ, không sợ bị mất.
* **Điểm cần lưu ý khi sử dụng Message Queue**
  + **Làm hệ thống phức tạp hơn:** Thêm message queue sẽ tăng tính phức tạp của hệ thống. Ta cần phải biết rõ message nào gửi vào queue nào, ai gửi ai nhận. Lúc debug ở local sẽ khó khăn hơn.
  + **Phải có Monitor Queue:** Phải có các biện phát theo dõi (monitor), để đảm bảo lượng message queue không quá nhiều, làm đầy queue. Queue tốt nhất là queue luôn rỗng, hoặc số lượng message trong queue không tăng lên nhiều (message gửi vào queue đều bị consume hết trong thời gian ngắn nhất)
  + **Phải có message format:** Để gửi/nhận từ 2 phía producer và consumer thống nhất format với nhau. Nếu 1 bên thay đổi sẽ làm bên kia không đọc được dữ liệu.
  + **Khó xử lý đồng bộ:** Không phải hệ thống nào cũng cần tới message queue. Nếu như service A gọi service B, theo cơ chế đồng bộ, cần kết quả xử lý ngay, ta nên dùng Rest hoặc gRPC sẽ tốt hơn.
* **Một số Message Queue được dùng hiện nay**
  + Kafka
  + Pulsar
  + RabitMQ
  + ActiveMQ
  + SQS
  + ZeroMQ
  + MSMQ
  + IronMQ
  + Kinesis
  + RocketMQ

1. Kafka

* **Kafka là gì?**

Kafka là một nền tảng phân phối dành cho xử lý dòng dữ liệu thời gian thực. Nó được phát triển bởi Apache Software Foundation và được thiết kế để xử lý, lưu trữ và truyền tải hàng tỷ dòng dữ liệu mỗi giây từ các nguồn khác nhau.

Kafka hoạt động dựa trên mô hình dòng dữ liệu (stream) và sự kiện (event), nơi dữ liệu được chia thành các dòng nhỏ được gọi là "dòng dữ liệu". Kafka giúp các ứng dụng có thể trao đổi dữ liệu theo thời gian thực, cho phép xử lý dữ liệu liên tục và đồng thời. Điều này làm cho nó trở thành một công cụ mạnh mẽ cho việc xây dựng các ứng dụng thời gian thực như xử lý sự kiện, giám sát hệ thống, phân tích dữ liệu thời gian thực, và nhiều ứng dụng khác.

* **Các thuộc tính của Kafka**
  + **Phân phối:** Kafka cho phép phân phối dữ liệu trên nhiều máy chủ, giúp mở rộng quy mô và đảm bảo khả năng chịu lỗi cao.
  + **Bền bỉ:** Dữ liệu được lưu trữ một cách bền bỉ và an toàn, đảm bảo không bị mất dữ liệu khi xảy ra sự cố.
  + **Lưu trữ theo thời gian thực:** Kafka cho phép lưu trữ dữ liệu theo thời gian thực trong một kho dữ liệu có khả năng truy vấn tương tự cơ sở dữ liệu truyền thống.
  + **Xử lý lại dữ liệu:** Dữ liệu trong Kafka có thể được xử lý lại và xem xét để đảm bảo tính chính xác và khả năng kiểm tra.
  + **Giao thức nhẹ nhàng:** Kafka sử dụng giao thức thời gian thực nhẹ nhàng cho phép các ứng dụng kết nối và trao đổi dữ liệu một cách hiệu quả.

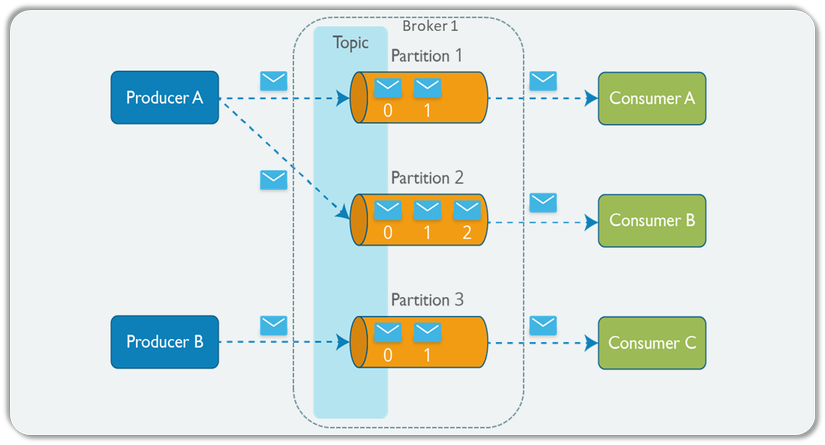
Kafka thường được sử dụng trong các hệ thống phân tán, các ứng dụng xử lý dữ liệu thời gian thực, hệ thống ghi nhật ký (log), xây dựng hệ thống theo dõi và giám sát, và nhiều tình huống khác nơi việc xử lý dữ liệu theo thời gian thực là quan trọng.

* **Các thành phần chính của Kafka**
  + **Producer:** Là thành phần tạo ra và gửi dữ liệu vào Kafka. Các producer chịu trách nhiệm gửi các dòng dữ liệu đến các topic trong Kafka.
  + **Consumer:** Là thành phần tiếp nhận và xử lý dữ liệu từ Kafka. Các consumer đọc dữ liệu từ các topic và thực hiện các xử lý theo thời gian thực hoặc lưu trữ dữ liệu cho việc xử lý sau này.
  + **Broker:** Là máy chủ Kafka chịu trách nhiệm lưu trữ dữ liệu và xử lý yêu cầu từ các producer và consumer. Nhiều broker hoạt động cùng nhau để tạo thành một cụm Kafka.
  + **Topic:** Là tên chủ đề mà dữ liệu được gửi đến và đọc từ. Mỗi topic có thể có nhiều partition để phân phối dữ liệu.
  + **Partition:** Là đơn vị nhỏ của dữ liệu trong một topic. Dữ liệu trong một partition được lưu trữ và xử lý độc lập với các partition khác.

Apache Kafka đã trở thành một công cụ quan trọng trong việc xây dựng các ứng dụng xử lý dữ liệu thời gian thực, giúp các doanh nghiệp và tổ chức tận dụng dữ liệu để đưa ra quyết định thông minh và định hình tương lai.

* **Cơ chế hoạt động của Kafka**

Cơ chế hoạt động của Apache Kafka dựa trên mô hình dòng dữ liệu (streaming) và sử dụng các thành phần như producer, broker, topic, partition và consumer để xử lý và quản lý dữ liệu thời gian thực. Dưới đây là mô tả chi tiết về cách hoạt động của Kafka:



* + **Producer và Consumer:**
    - Producer: Producer tạo và gửi dữ liệu đến Kafka. Để gửi dữ liệu, producer chỉ cần gửi dòng dữ liệu tới một topic cụ thể. Producer có thể gửi dữ liệu tới nhiều topic khác nhau. Dữ liệu sau khi được gửi đi sẽ không thể thay đổi hoặc xóa.
    - Consumer: Consumer là thành phần tiếp nhận và xử lý dữ liệu từ Kafka. Consumer đọc dữ liệu từ các topic và partition. Mỗi consumer có thể đọc dữ liệu từ một hoặc nhiều partition cùng một lúc.
  + **Topic và Partition:**
    - Topic: Mỗi dòng dữ liệu được gửi đến Kafka được gắn với một topic. Topic là cách để nhóm các dòng dữ liệu có cùng chủ đề hoặc loại dữ liệu. Các producer gửi dữ liệu tới các topic cụ thể.
    - Partition: Mỗi topic có thể được chia thành nhiều partition nhỏ. Mỗi partition là một đơn vị độc lập để lưu trữ và xử lý dữ liệu. Partition giúp mở rộng quy mô và tăng hiệu suất cho Kafka.
  + **Broker:** Kafka hoạt động trên cơ sở của một hoặc nhiều máy chủ gọi là broker. Mỗi broker là một máy chủ Kafka chịu trách nhiệm lưu trữ dữ liệu và xử lý yêu cầu từ producer và consumer. Các broker cùng nhau tạo thành một cụm Kafka. Cụm này giúp mở rộng khả năng chịu tải và đảm bảo khả năng chịu lỗi.
  + **Quá trình lưu trữ và truy xuất dữ liệu:** Khi producer gửi dữ liệu đến Kafka, dữ liệu được chia thành các dòng và gắn với topic tương ứng. Mỗi partition của topic sẽ chứa một phần dữ liệu. Dữ liệu trong mỗi partition được lưu trữ trên đĩa và có thể được lưu trữ theo thời gian thực. Consumer có thể đọc dữ liệu từ các partition của topic. Mỗi consumer theo dõi việc đọc của mình bằng việc theo dõi offset - chỉ số đánh dấu dòng dữ liệu cuối cùng mà nó đã đọc.
  + **Xử lý lại dữ liệu và tính bền bỉ:** Kafka cho phép consumer xử lý lại dữ liệu bằng cách xác định offset mà nó muốn đọc từ. Điều này giúp đảm bảo dữ liệu có thể được xử lý lại hoặc kiểm tra lại tính chính xác. Dữ liệu trong Kafka được lưu trữ một cách bền bỉ và an toàn trên đĩa của các broker, đảm bảo tính chất bền bỉ của dữ liệu ngay cả khi có sự cố.

1. RabbitMQ

* **RabbitMQ là gì?**

RabbitMQ là một AMQP message broker hay còn gọi là phần mềm quản lý hàng đợi message, là một phần mềm trung gian giúp các hệ thống, server, ứng dụng có thể giao tiếp, trao đổi dữ liệu với nhau. Nhiệm vụ của RabbitMQ được hiểu đơn giản là: nhận message từ nhiều nguồn => lưu trữ, sắp xếp sao cho hợp lý => đẩy tới đích đến.

Là một message broker mã nguồn mở, có dung lượng nhẹ, dễ dàng triển khai trên rất nhiều hệ điều hành lẫn Cloud, vì thế RabbitMQ vô cùng được ưa chuộng và trở nên phổ biến trong thời gian qua.

* **Message Broker là gì?**

Message broker là một chương trình trung gian được thiết kế để validating, transforming và routing messages. Chúng phục vụ các nhu cầu giao tiếp giữa các ứng dụng với nhau.

Với Message broker, ứng dụng nguồn (producer) gửi một message đến một server process mà nó có thể cung cấp việc sắp xếp dữ liệu, routing, message translation, persistence và delivery tất cả điểm đến thích hợp(consumer).

Có 2 hình thức giao tiếp cơ bản với một Message Broker: Publish và Subscribe (Topics), Point-to-Point (Queues).

* **Các thành phần chính trong RabbitMQ**

Trong kiến trúc của RabbitMQ, các thành phần chính bao gồm message, producer, exchange, bindings, queue và consumer:

* + **Message:**

Message (tin nhắn) là đơn vị cơ bản để trao đổi thông tin trong RabbitMQ. Nó bao gồm một body (còn gọi là payload) và các property (thuộc tính) tuỳ chọn khác như header (tiêu đề), routing key (khoá định tuyến), message ID, ...

* + **Producers:**

Producer là một ứng dụng (hoặc phiên bản ứng dụng) gửi message tới RabbitMQ. Đây là điểm khởi đầu trong workflow của RabbitMQ.

Producer thường mở một hoặc nhiều kết nối TCP tới exchange trong quá trình ứng dụng khởi tạo. Các kết nối này thường tồn tại miễn là kết nối của chúng hoặc ứng dụng đang chạy (long-lived connection).

* + **Bindings:**

Binding là các ràng buộc được thiết lập để xác định mối quan hệ giữa exchange và queue. Chúng chỉ định các quy tắc định tuyến để gửi message từ exchange tới queue thích hợp. Một queue có thể liên kết với nhiều exchange và một exchange có thể có nhiều liên kết tới các queue khác nhau.

* + **Exchanges:**

Trong RabbitMQ, exchange (sàn giao dịch) đóng vai trò như một bộ định tuyến message. Khi producer gửi message tới RabbitMQ, message sẽ được gửi tới exchange. Sau đó, exchange sẽ định tuyến message đến queue thích hợp.

* + **Queues:**

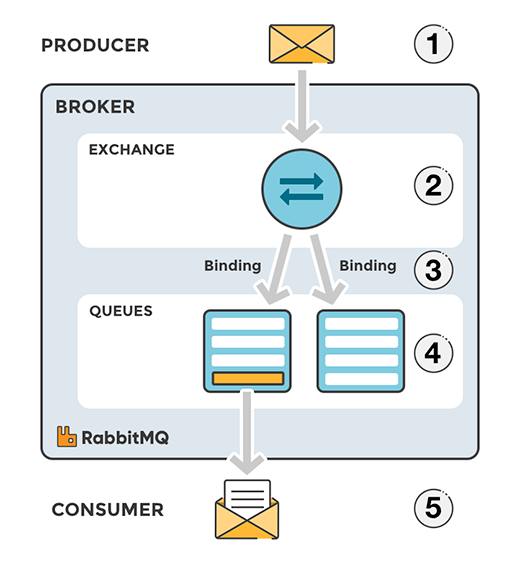
Trong RabbitMQ, queue là một danh sách các message được lưu trữ cho tới khi chúng được tiêu thụ bởi consumer.

Message được định tuyến tới queue thông qua bindings từ exchange. RabbitMQ cho phép tạo nhiều queue với các cấu hình khác nhau. Mỗi queue có một tên, ngoài ra còn có các thuộc tính tùy chọn khác như độ bền, thời gian hết hạn của message và độ dài tối đa của message.

* + **Consumers:**

Consumer là một ứng dụng (hoặc phiên bản ứng dụng) nhận message từ một hay nhiều queue. Client có thể đóng kết nối tới RabbitMQ. Khi RabbitMQ phát hiện mất kết nối, việc gửi message sẽ dừng lại.

* **Cơ chế hoạt động của RabbitMQ**



Luồng message trong RabbitMQ hoạt động như sau:

* + Producer gửi một message tới một exchange. Exchange phải được chỉ định type khi được tạo.
  + Exchange nhận được message và chịu trách nhiệm định tuyến message. Việc trao đổi sẽ tính đến các thuộc tính của message và loại exchange, chẳng hạn như routing key.
  + Giữa exchange và queue phải thiết lập bindings từ trước. Exchange định tuyến message vào queue tùy thuộc vào thuộc tính của message. Trong ví dụ trên, có 2 bindings từ exchange với 2 queue khác nhau.
  + Các message vẫn ở trong queue cho đến khi chúng được consumer xử lý.
  + Consumer xử lý message.
* **Ưu điểm của RabbitMQ**
  + **Mã nguồn mở:**

RabbitMQ dược viết bằng mã nguồn mở nên các nhà phát triển và kỹ sư trong cộng đồng RabbitMQ có thể đóng góp cải tiến và tiện ích bổ sung, đồng thời tận dụng được sự hỗ trợ tài chính của công ty Pivotal.

* + **Nhẹ:**

RabbitMQ rất nhẹ, cần ít hơn 40 MB RAM để chạy ứng dụng RabbitMQ cùng với các plugin khác như giao diện quản lý.

* + **Độ tin cậy:**

RabbitMQ cung cấp độ tin cậy và khả năng mở rộng để quản lý việc phân phối message trên mạng lưới các nodes, đảm bảo rằng message được gửi theo thứ tự mong muốn và không bị thất lạc.

* + **Ưu tiên message:**

RabbitMQ cho phép các message được chỉ định mức độ ưu tiên khác nhau. Điều này đảm bảo rằng các message có mức độ ưu tiên cao sẽ được xử lý trước các message có mức độ ưu tiên thấp hơn.

* + **Tính linh hoạt trong kiểm soát cân bằng message:**

RabbitMQ cung cấp các mức QoS khác nhau như "At Most Once" (Tối đa một lần), "At Least Once" (Tối thiểu một lần) và "Exactly Once" (Đúng một lần), cho phép bạn kiểm soát sự cân bằng giữa reliability (độ tin cậy) và performance (hiệu suất).

Các message trong RabbitMQ có thể được lưu vào vào ổ đĩa, đảm bảo message không bị mất khi khởi động lại broker. Tuy nhiên, điều này có thể làm tăng chi phí I/O, ảnh hưởng tới hiệu năng. Bạn có thể chọn đánh đổi durability (độ bền) để có throughput (thông lượng) cao hơn.

RabbitMQ cung cấp sự linh hoạt trong phân phối message và tiêu thụ message thông qua việc chọn hình thức exchange phù hợp (direct, topic, fanout, headers) và chọn routing key (khoá định tuyến) phù hợp.

* + **Plugin dành cho môi trường có độ trễ cao:**

Do các cấu trúc liên kết và kiến trúc mạng không giống nhau nên RabbitMQ cung cấp tính năng trao đổi message trong môi trường có cả độ trễ thấp và cao, chẳng hạn Internet. Điều này cho phép RabbitMQ được phân cụm trên cùng một mạng cục bộ và chia sẻ message thống nhất trên nhiều datacenter.

* + **Hỗ trợ ngôn ngữ và giao thức linh hoạt:**

RabbitMQ hỗ trợ chạy trên nhiều hệ điều hành và cả trên môi trường cloud, đồng thời cung cấp nhiều client libraries cho hầu hết các ngôn ngữ lập trình phổ biến. Ngoài ra, RabbitMQ còn hỗ trợ các giao thức nhắn tin khác nhau như STOMP, MQTT, ...

* + **Nhiều lớp bảo mật:**

Trong RabbitMQ có nhiều lớp bảo mật. Kết nối của client có thể được bảo mật bằng cách thực thi giao tiếp chỉ có SSL và xác thực chứng chỉ client. Quyền truy cập của người dùng có thể được quản lý ở máy chủ ảo, cung cấp sự cô lập mức độ cao cho message và tài nguyên. Ngoài ra, quyền truy cập cấu hình, đọc từ queues (hàng đợi) và ghi vào exchanges (sàn giao dịch) được quản lý bằng regex.

* **Nhược điểm của RabbitMQ:**
  + **Đường cong học tập:** Việc thiết lập và định cấu hình RabbitMQ có thể phức tạp, đặc biệt đối với những người mới làm quen với khái niệm message queuing. Quản lý phù hợp đòi hỏi sự hiểu biết về các khái niệm cốt lõi trong RabbitMQ như queues, exchanges, bindings (ràng buộc), ...
  + **Chi phí hiệu năng:** Mặc dù độ tin cậy của RabbitMQ là một điểm mạnh nhưng cơ chế xác nhận và lưu trữ có thể gây ra một số chi phí về hiệu năng, đặc biệt là trong các tình huống cần thông lượng cao.
  + **Hàng đợi tích luỹ:** Nếu consumer không thể tiêu thụ kịp message dẫn tới tích lũy message trong queue. Điều này có thể gây ra tiêu tốn tài nguyên và có khả năng dẫn đến các vấn đề về hiệu suất.
  + **Tuần tự hoá tin nhắn:** Việc tuần tự hóa và giải tuần tự hóa các đối tượng phức tạp có thể tốn thời gian, ảnh hưởng đến hiệu suất tổng thể.

1. So sánh rabbitmq/kafka

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Kafka** | **RabbitMQ** |
| **Structure** | | Broker-Based  Producer  Consumer  Topic  Log-Based Storage | Message Broker  Producer  Consumer  Queue |
| **Ngôn ngữ** | | Scala, Java | Erlang |
| **Protocol** | | binary protocol over TCP | AMQP |
| **Hỗ trợ Client API** | | Java, Ruby, Python, Node.js | Java, Ruby, JavaScript, Go, C, Swift, Spring, Elixir, PHP, .NET |
| **Bảo mật** | | TLS, JAAS | Quản lý qua tool dành cho admin |
| **Khả năng chịu lỗi** | | Có | Có |
| **Hiệu năng** | | triệu message/giây | ~10.000 message/giây |
| **Hệ sinh thái** | | Hỗ trợ tốt Big Data và streaming | Hạn chế, không bằng Kafka |
| **Cách quản lý Message** | **Delivery Guarantee** | - **At Least Once**: Kafka đảm bảo gửi message ít nhất một lần. Điều này đảm bảo rằng message sẽ không bị mất và có thể được gửi lại nhiều lần. - **Exactly Once**: Kafka cung cấp tính năng "exactly once" gửi message chính xác 1 lần. - **At Most Once**: Message được gửi tối đa 1 lần. Có thể xảy ra tình huống message bị mất do không có retry. | - **At Least Once**: RabbitMQ đảm bảo gửi message ít nhất một lần bằng cách gửi xác nhận (acknowledgment) từ consumer về producer. - **At Most Once**: Cũng như Kafka, RabbitMQ có thể được cấu hình để đảm bảo giao message tối đa một lần. |
| **Tiêu thụ message** | pull model | - push model (mặc định) - pull model |
| **Ưu tiên message** | - Kafka không có cơ chế tích hợp sẵn để xử lý ưu tiên của message. - Ưu tiên message có thể được tùy chỉnh thông qua cách sắp xếp Kafka topic hoặc thực hiện trong ứng dụng. | RabbitMQ hỗ trợ xử lý ưu tiên message bằng cách gán priority level (mức ưu tiên) cho các message và sắp xếp chúng dựa trên priority level. |
| **Sắp xếp message** | Sắp xếp message bằng topic | Sắp xếp theo thứ tự trong queue |
| **Xoá message** | - Kafka giữ message trong một kho dữ liệu (log) trong khoảng thời gian được cấu hình. - Message có thể được xóa sau khi đạt đến thời gian sống định sẵn. | - RabbitMQ xóa message sau khi chúng đã được consumer xác nhận (ACK). - Có khả năng cấu hình thời gian sống (TTL) cho queue hoặc cho từng message. |
| **Thời gian tồn tại của message** | Thời gian sống của một message trong Kafka có thể được cấu hình theo thời gian thực. Message có thể tồn tại trong khoảng thời gian bạn chọn. | RabbitMQ cho phép cấu hình thời gian sống (TTL) cho queue hoặc từng message. Message sẽ bị xóa khi hết thời gian sống. |
| **Hiệu suất** | | -Tốc độ xử lý cao: Kafka được thiết kế với mục tiêu đảm bảo tốc độ xử lý cao cho việc truyền tải message. Điều này làm cho nó phù hợp cho các ứng dụng cần xử lý dòng dữ liệu lớn và có yêu cầu thời gian thực.  -Xử lý đồng thời: Kafka hỗ trợ xử lý đồng thời (concurrency) và có khả năng mở rộng bằng cách thêm broker vào cụm Kafka để tăng khả năng xử lý. | -Hiệu suất tốt cho hàng đợi: RabbitMQ là một hệ thống quản lý hàng đợi message xuất sắc và thích hợp cho các ứng dụng sử dụng hàng đợi để truyền tải message. Nó có hiệu suất tốt trong việc quản lý hàng đợi và đảm bảo tính ổn định. |
| **Tính ổn định** | | -Tính ổn định cao: Kafka được sử dụng rộng rãi trong các ứng dụng quan trọng và đòi hỏi tính ổn định cao. Nó được thiết kế để chịu lỗi và đảm bảo tính toàn vẹn dữ liệu.  -Lưu trữ lâu dài: Kafka có khả năng lưu trữ message lâu dài trong kho dữ liệu (log), cho phép bạn tái xử lý message hoặc xem lịch sử. | -Tính ổn định cho message queue: RabbitMQ là một message broker truyền thống và có tính ổn định cho việc quản lý hàng đợi message. Nó đảm bảo tính toàn vẹn của message và cung cấp kiểm soát độ tin cậy thông qua xác nhận (acknowledgment). |
| **Trường hợp sử dụng** | | Kafka thường được sử dụng cho việc xử lý dòng dữ liệu lớn, ứng dụng thời gian thực và lưu trữ dữ liệu thời gian thực. Một vài trường hợp sử dụng phổ biến của Kafka bao gồm: Xử lý Big Data, Real-time Applications, Monitoring Systems, Data Storage,… | RabbitMQ thích hợp cho việc quản lý message queue (hàng đợi thông điệp), load balancing, tích hợp ứng dụng, và xử lý message priority. Một vài trường hợp sử dụng phổ biến của RabbitMQ bao gồm: Quản lý message queue, Cân bằng tải, Tích hợp với ứng dụng khác, Xử lý thông điệp ưu tiên (Message Priority),… |