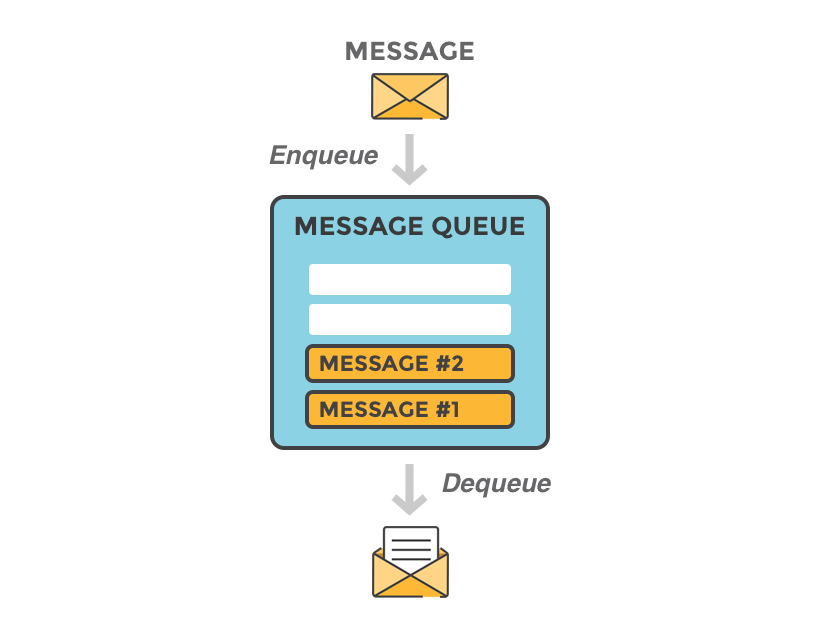
**Báo cáo DAY 14**

1. **Messege Queue là gì?**

Message Queue nôm na là Queue (hàng đợi), chứa Message (Tin nhắn) như hộp thư . Và nó cho phép các thành phần/service trong một hệ thống (hoặc nhiều hệ thống), trao đổi thông tin cho nhau.



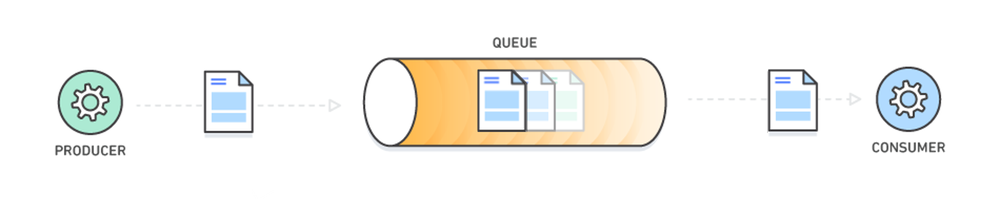
Ý nghĩa của queue (hàng đợi) là nó thực hiện việc lấy message theo cơ chế vào trước thì ra trước ( First In First Out ).

Một hệ thống Message Queue thường có những thành phần sau:

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

Một số Message queue được dùng hiện nay:

* [Kafka](https://lcdung.top/apache-kafka-la-gi/)
* [Pulsar](https://lcdung.top/apache-pulsar-incubator/)
* RabitMQ
* ActiveMQ
* SQS
* ZeroMQ
* MSMQ
* IronMQ
* Kinesis
* RocketMQ

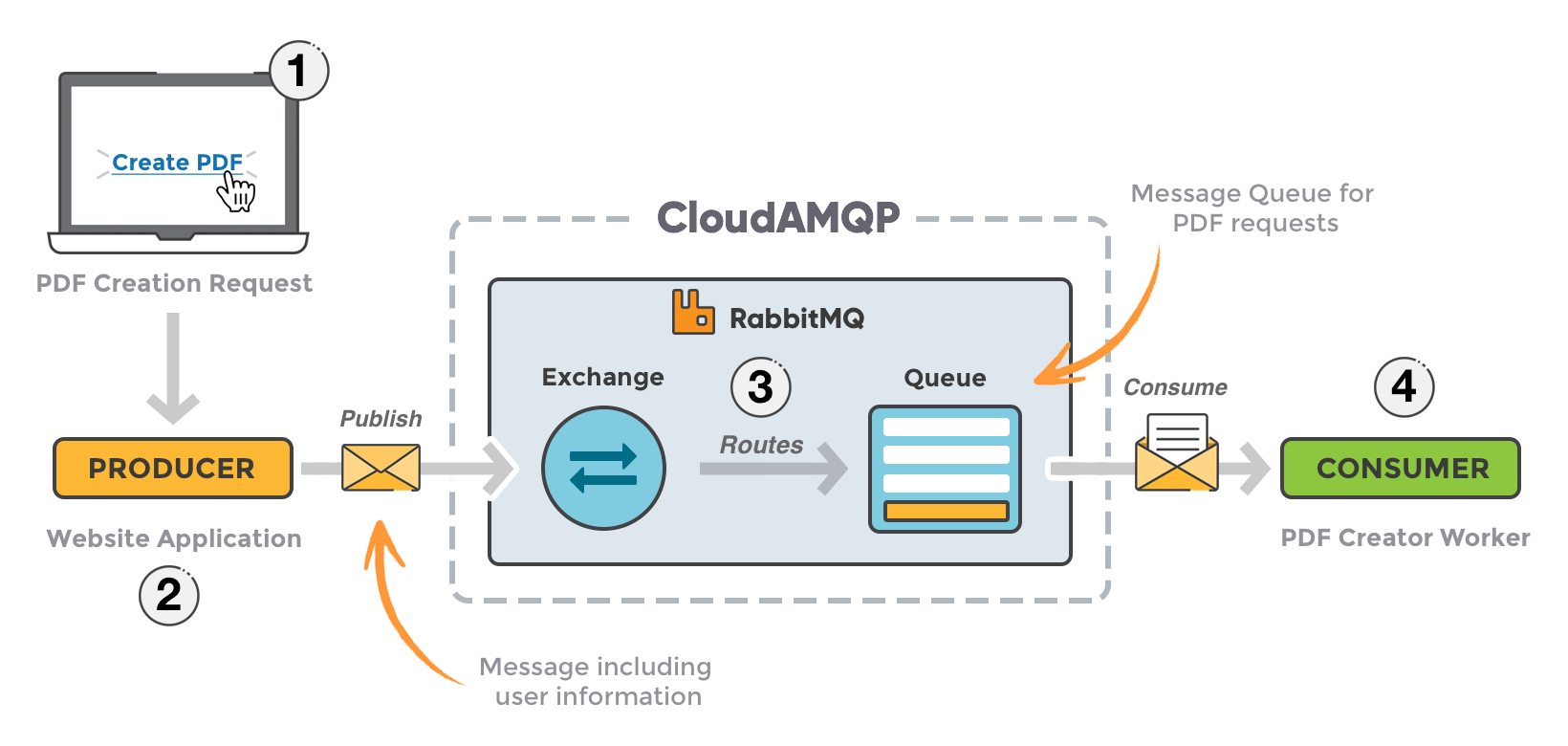


**Thực tế Message Queue được sử dụng thế nào?**

Trong các hệ thống [dùng kiến trúc microservice](https://lcdung.top/phat-trien-phan-mem-theo-kien-truc-microservice/), ta sử dụng message queue để giúp các service liên hệ với nhau **một cách bất đồng bộ**. Service X làm xong việc có thể gửi message queue để service Y kích hoạt xử lý.

Ví dụ: có một trang web cho phép người dùng tải video từ hệ thống thì nó sẽ có các thành phần sau:

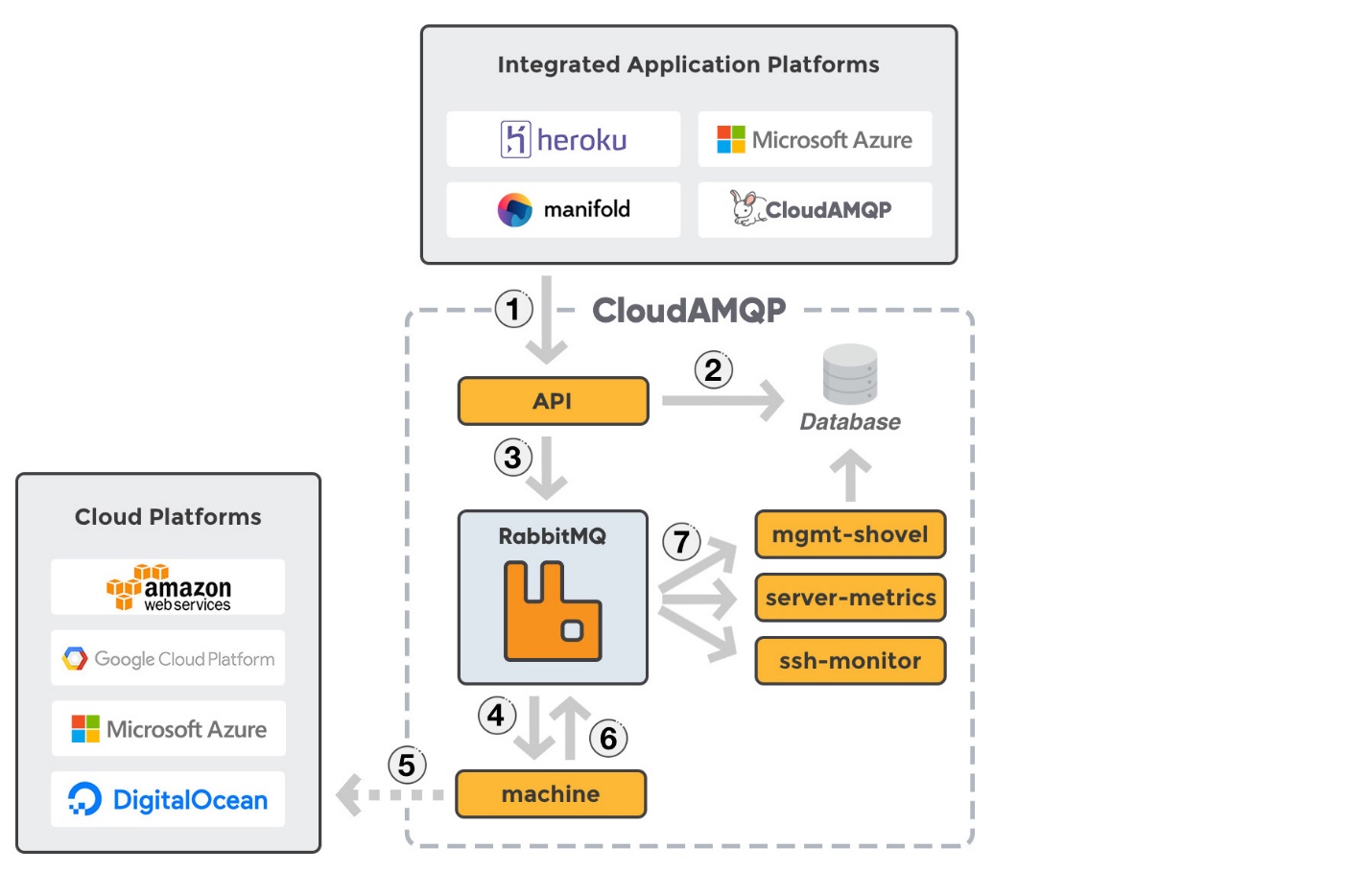
* **API service**: Là 1 producer. Nhận thông tin (URL Video) từ phía người dùng và đưa thông tin này vào message queue
* **Processing Service**: Vừa là consumer vừa là producer. Service này đọc URL Video từ message queue, bắt đầu tải file Video về và encode lại, lưu vào server. Sau khi encode xong, nó đưa URL của file đã encode vào message queue
* **Uploading Service**:  Khi nhận được message từ processing server, nó sẽ upload video đó lên Amazon S3



**Tại sao lại sử dụng Message Queue?**

**Ưu điểm về 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](https://lcdung.top/phat-trien-phan-mem-theo-kien-truc-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 về 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.

1. **Rabbit Message Queue:**
   1. RabbitMQ là gì:

**RabbitMQ** là một AMQP message broker hay còn gọi là phần mềm quản lý hàng đợi message. Nói đơn giản, đây là phần mềm định nghĩa hàng đợi một ứng dụng khác có thể kết nối đến để bỏ message vào và gửi message dựa trên nó.

* 1. **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 (Định tuyến), message translation, persistence và delivery tất cả các 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)

## **Khi nào và tại sao dùng RabbitMQ**

RabbitMQ giúp các [web server](https://topdev.vn/blog/webserver-la-gi/) gửi các reponse cho các request rất nhanh thay vì bị ép buộc chạy một procedure ngốn tài nguyên trên một hệ thống. Việc đưa message vào hàng đợi là một giải pháp hay khi ta muốn phân tán message cho nhiều người nhận giúp giảm tải cho các worker xử lý.

VD như user được phép tạo file PDF cho [mẫu cv it](https://topdev.vn/blog/cv-it/) từ phần mềm [tạo CV Online](https://topdev.vn/tao-cv-online) của TopDev, bài toán là khi hàng nghìn user cùng nhấp vào nút tạo PDF, lúc này server nhận rất nhiều request sẽ gây ra vài vấn đề như chậm, quá tải, thậm chí không tạo được file PDF do nghẽn…lúc này chúng ta cần dùng RabbitMQ để đẩy các request này vào hàng chờ. Cơ chế như sau:

Một consumer lấy message từ hàng đợi và bắt đầu xử lý PDF trong lúc với một producer đang bỏ thêm những message mới vào trong hàng đợi. Một request có thể được tạo bằng ngôn ngữ này và xử lý bằng một ngôn ngữ khác. Hai ứng dụng trao đổi với nhau qua các message. Do đó, hai ứng dụng gửi và nhận sẽ có độ ràng buộc thấp.

**Những khái niệm cơ bản trong RabbitMQ**

* **Producer**: Ứng dụng gửi message.
* **Consumer**: Ứng dụng nhận message.
* **Queue**: Lưu trữ messages.
* **Message**: Thông tin truyền từ Producer đến Consumer qua RabbitMQ.
* **Connection**: Một kết nối TCP giữa ứng dụng và RabbitMQ broker.
* **Channel**: Một kết nối ảo trong một Connection. Việc publishing hoặc consuming từ một queue đều được thực hiện trên channel.
* **Exchange**: Là nơi nhận message được publish từ Producer và đẩy chúng vào queue dựa vào quy tắc của từng loại Exchange. Để nhận được message, queue phải được nằm trong ít nhất 1 Exchange.
* **Binding**: Đảm nhận nhiệm vụ liên kết giữa Exchange và Queue.
* **Routing key**: Một key mà Exchange dựa vào đó để quyết định cách để định tuyến message đến queue. Có thể hiểu nôm na, Routing key là địa chỉ dành cho message.
* **AMQP**: Giao thức Advance Message Queuing Protocol, là giao thức truyền message trong RabbitMQ.
* **User**: Để có thể truy cập vào RabbitMQ, chúng ta phải có username và password. Trong RabbitMQ, mỗi user được chỉ định với một quyền hạn nào đó. User có thể được phân quyền đặc biệt cho một Vhost nào đó.
* **Virtual host/Vhost**: Cung cấp những cách riêng biệt để các ứng dụng dùng chung một RabbitMQ instance. Những user khác nhau có thể có các quyền khác nhau đối với vhost khác nhau. Queue và Exchange có thể được tạo, vì vậy chúng chỉ tồn tại trong một vhost.

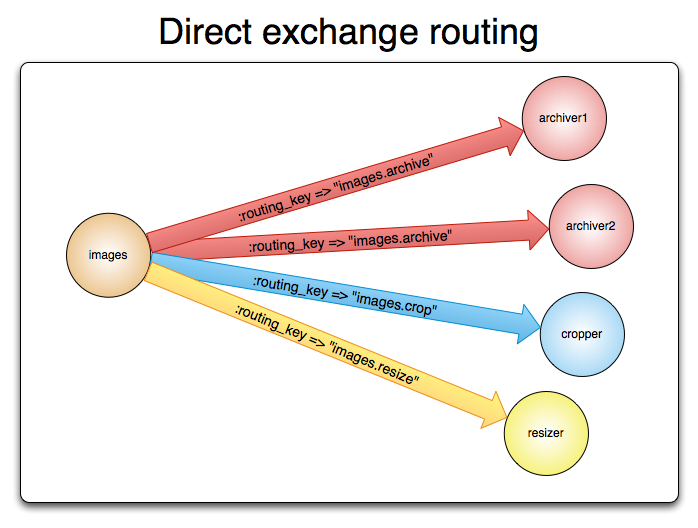
Exchanges, connections, và queue có thể được cấu hình với các tham số như *durable*, *temporary*và *auto delete* khi tạo. Durable exchanges tồn tại khi máy chủ khởi động lại và kéo dài cho đến khi chúng bị xóa . Temporary exchanges tồn tại cho đến khi RabbitMQ ngừng hoạt động.  Auto-deleted exchangesa sẽ bị xóa sau khi đối tượng bị ràng buộc cuối cùng không bị ràng buộc khỏi trao đổi.

Trong RabbitMQ, có bốn loại **Exchange** khác nhau định tuyến thông điệp theo cách khác nhau bằng cách sử dụng các thông số và thiết lập ràng buộc khác nhau. Khách hàng có thể tạo các exchanges của riêng họ hoặc sử dụng các exchanges mặc định được xác định trước khi máy chủ khởi động lần đầu tiên.

**Các loại Exchange**

**a. Direct Exchange**

      Direct Exchange vận chuyển message đến queue dựa vào routing key. Thường được sử dụng cho việc định tuyến tin nhắn unicast-đơn hướng (mặc dù nó có thể sử dụng cho định tuyến multicast-đa hướng). Các bước định tuyến message:

* Một queue được ràng buộc với một direct exchange bởi một routing key K.
* Khi có một message mới với routing key R đến direct exchange. Message sẽ được chuyển tới queue đó nếu R=K.

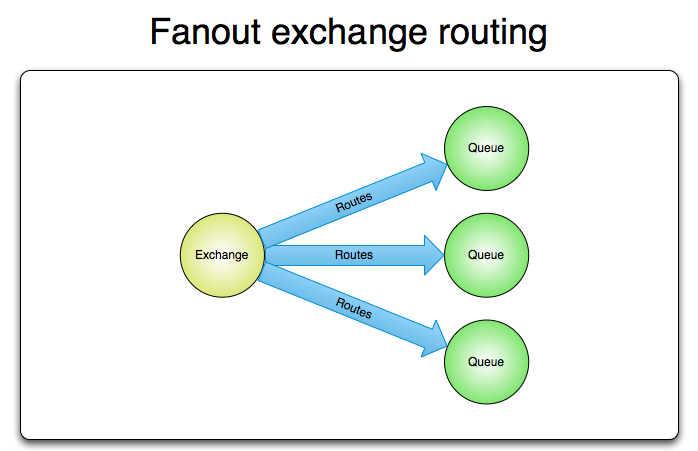
*Hình 2. Các bước vận chuyển message trong Direct exchange*

**b. Default Exchange**

      Mỗi một exchange đều được đặt một tên không trùng nhau, default exchange bản chất là một direct exchange nhưng không có tên (string rỗng). Nó có một thuộc tính đặc biệt làm cho nó rất hữu ích cho các ứng dụng đơn giản: mọi queue được tạo sẽ tự động được liên kết với nó bằng một routing key giống như tên queue.

      Ví dụ, nếu bạn tạo ra 1 queue với tên "hello-world", RabbitMQ broker sẽ tự động binding default exchange đến queue "hello-word" với routing key "hello-world".

**b. Fanout Exchange**

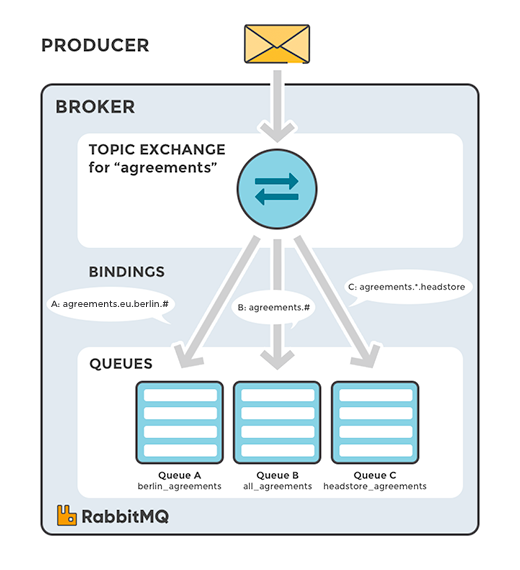
      Fanout exchange định tuyến message tới tất cả queue mà nó bao quanh, routing key bị bỏ qua. Giả sử, nếu nó N queue được bao quanh bởi một Fanout exchange, khi một message mới published, exchange sẽ vận chuyển message đó tới tất cả N queues. Fanout exchange được sử dụng cho định tuyến thông điệp broadcast - quảng bá.

*Hình 3. Các bước vận chuyển message trong Fanout exchange*

**c. Topic Exchange**

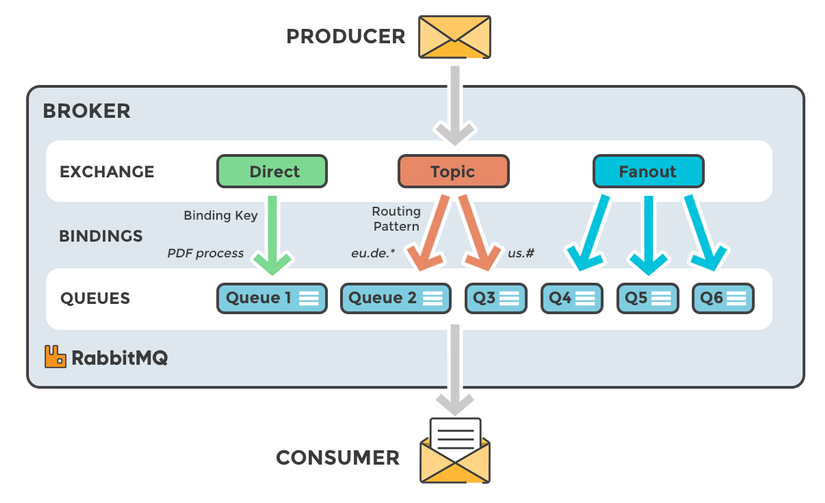
      Topic exchange định tuyến message tới một hoặc nhiều queue dựa trên sự trùng khớp giữa routing key và pattern. Topic exchange thường sử dụng để thực hiện định tuyến thông điệp multicast. Ví dụ một vài trường hợp sử dụng:

* Phân phối dữ liệu liên quan đến vị trí địa lý cụ thể.
* Xử lý tác vụ nền được thực hiện bởi nhiều workers, mỗi công việc có khả năng xử lý các nhóm tác vụ cụ thể.
* Cập nhật tin tức liên quan đến phân loại hoặc gắn thẻ (ví dụ: chỉ dành cho một môn thể thao hoặc đội cụ thể).
* Điều phối các dịch vụ của các loại khác nhau trong cloud



**d. Headers Exchange**

      Header exchange được thiết kế để định tuyến với nhiều thuộc tính, đễ dàng thực hiện dưới dạng tiêu đề của message hơn là routing key. Header exchange bỏ đi routing key mà thay vào đó định tuyến dựa trên header của message. Trường hợp này, broker cần một hoặc nhiều thông tin từ application developer, cụ thể là, nên quan tâm đến những tin nhắn với tiêu đề nào phù hợp hoặc tất cả chúng.



**Tham khảo thêm:** [**https://www.cloudamqp.com/blog/2015-09-03-part4-rabbitmq-for-beginners-exchanges-routing-keys-bindings.html#:~:text=Topic%20Exchange,routing%20key%20and%20this%20pattern**](https://www.cloudamqp.com/blog/2015-09-03-part4-rabbitmq-for-beginners-exchanges-routing-keys-bindings.html#:~:text=Topic%20Exchange,routing%20key%20and%20this%20pattern)**.**

**Tham khảo: Cách tạo Producer và Consumer với RabbitMQ và Spring Boot:**

* + **Producer:**[**https://www.javainuse.com/spring/spring-boot-rabbitmq-hello-world**](https://www.javainuse.com/spring/spring-boot-rabbitmq-hello-world)
  + **Consumer:**[**https://www.javainuse.com/spring/spring-boot-rabbitmq-consume**](https://www.javainuse.com/spring/spring-boot-rabbitmq-consume)

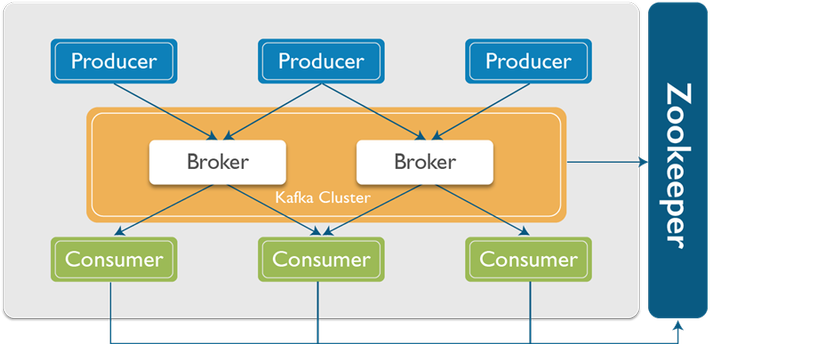
1. **Kafka message queue:**
   1. **Kafka là gì:**

**Kafka** là gì? – Đó là hệ thống message pub/sub phân tán (distributed messaging system). Bên pulbic dữ liệu được gọi là producer, bên subscribe nhận dữ liệu theo topic được gọi là consumer. Kafka có khả năng truyền một lượng lớn message theo thời gian thực, trong trường hợp bên nhận chưa nhận message vẫn được lưu trữ sao lưu trên một hàng đợi và cả trên ổ đĩa bảo đảm an toàn. Đồng thời nó cũng được replicate trong cluster giúp phòng tránh mất dữ liệu.

## **Kafka hoạt động như thế nào?**

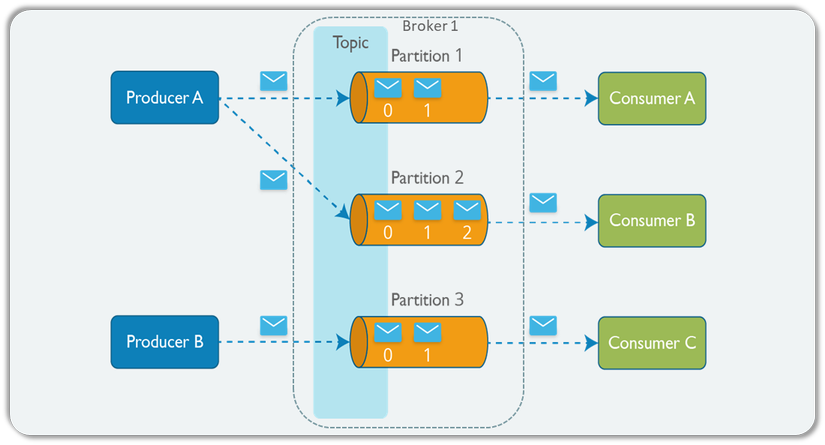
Kafka được xây dựng dựa trên mô hình publish/subcribe, tương tự như bất kỳ hệ thống message nào khác. Các ứng dụng (**đóng vai trò là producer**) gửi các messages (**records**) tới một node kafka (**broker**) và nói rằng những messages này sẽ được xử lý bởi các ứng dụng khác gọi là **consumers**. Các messages được gửi tới kafka node sẽ được lưu trữ trong một nơi gọi là **topic** và sau đó consumer có thể subcribe tới topic đó và lắng nghe những messages này. Messages có thể là bất cứ thông tin gì như giá trị cảm biến, hành động người dùng,…

Topic có thể được xem như là tên của một danh mục mà các messages sẽ được lưu trữ và được đẩy vào.



**Những khái niệm cơ bản trong Kafka**

* **PRODUCER**: Kafka lưu, phân loại message theo topic, sử dụng producer để publish message vào các topic. Dữ liệu được gửi đển partition của topic lưu trữ trên Broker.
* **CONSUMER**: Kafka sử dụng consumer để subscribe vào topic, các consumer được định danh bằng các group name. Nhiều consumer có thể cùng đọc một topic.
* **TOPIC**: Dữ liệu truyền trong Kafka theo topic, khi cần truyền dữ liệu cho các ứng dụng khác nhau thì sẽ tạo ra cá topic khác nhau.
* **PARTITION**: Đây là nơi dữ liệu cho một topic được lưu trữ. Một topic có thể có một hay nhiều partition. Trên mỗi partition thì dữ liệu lưu trữ cố định và được gán cho một ID gọi là offset. Trong một Kafka cluster thì một partition có thể replicate (sao chép) ra nhiều bản. Trong đó có một bản leader chịu trách nhiệm đọc ghi dữ liệu và các bản còn lại gọi là follower. Khi bản leader bị lỗi thì sẽ có một bản follower lên làm leader thay thế. Nếu muốn dùng nhiều consumer đọc song song dữ liệu của một topic thì topic đó cần phải có nhiều partition.
* **BROKER**: Kafka cluster là một set các server, mỗi một set này được gọi là 1 broker
* **ZOOKEEPER**: được dùng để quản lý và bố trí các broker.

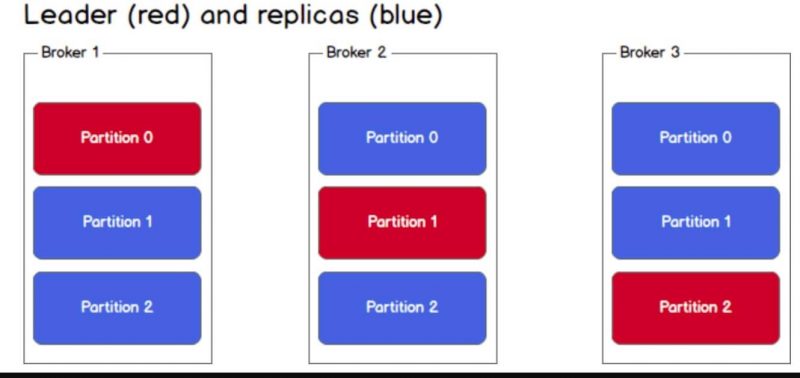


### Partition

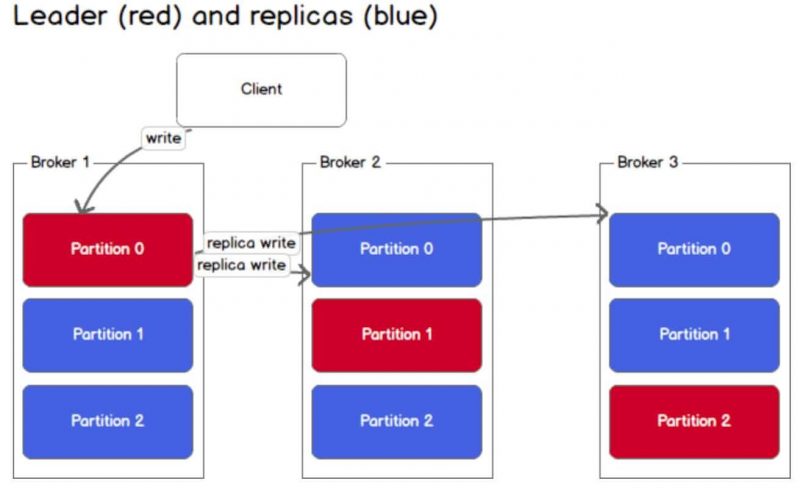
Topics trong kafka có thể có kích cỡ rất lớn, như vậy không nên lưu trữ tất cả dữ liệu của một topic trên một node, dữ liệu cần đươc phân chia ra thành nhiều partition sẽ giúp bảo toàn dữ liệu cũng như xử lý dữ liệu dễ dàng hơn. Partitions cho phép chúng ta thực hiện subcribe song song tới một topic cụ thể bằng cách phân chia dữ liệu trong một topic cụ thể ra cho nhiều broker khác nhau (**kafka node**), mỗi partition có thể được đặt trên một máy riêng biệt – cho phép nhiều consumer đọc dữ liệu từ một topic diễn ra một cách song song.

Để tăng tính khả dụng (**availability**) của partition, mỗi partition cũng có giá trị replicas của riêng nó. Để dễ hiểu hơn về kafka, mình sẽ trình bày bằng ví dụ với 3 node/broker.

Bây giờ, một topic sẽ được chia ra thành 3 partitions và mỗi broker sẽ có một bản copy của partition. Trong số những bản copy partition này, sẽ có một bản copy được bầu chọn làm leader, trong khi những bản copy khác chỉ thực hiện đồng bộ dữ liệu với partition leader.



Tất cả các hành động ghi và đọc tới một topic sẽ đều phải đi qua partition leader tương ứng và leader sẽ phối hợp để cập nhật dữ liệu mới tới các replica parition khác. Nếu leader bị hỏng, một trong các replica partition sẽ đảm nhận vai trò là một leader mới.

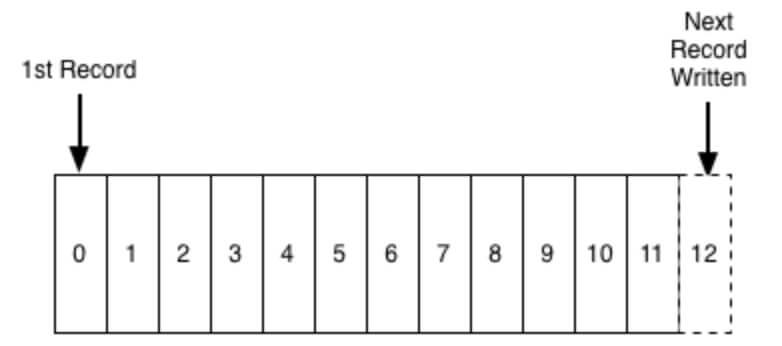


Để một producer/consumer ghi/đọc message từ một partition, chắc chắn chúng cần phải biết leader là ai phải không? Thông tin này cần phải có sẵn ở một vị trí nào đó.

Kafka lưu trữ những thông tin như vậy là metadata trong một dịch vụ gọi là Zookeeper.

## **Cấu trúc dữ liệu log trong Kafka**

Chìa khóa chính dẫn tới khả năng mở rộng và hiệu suất của kafka chính là log. Thường thì các developer khi mới tiếp cận kafka cảm thấy khá rối khi lần đầu tiên nghe đến “**log**“, bởi vì chúng ta thường hiểu “**log**” chính là thuật nghữ được sử dụng trong log ứng dụng. Tuy nhiên, những gì mình đang nói ở đây, **là cấu trúc dữ liệu log**. Log là một cấu trúc dữ liệu có thứ tự nhất quán mà chỉ hỗ trợ dạng nối thêm (append). Bạn không thể chỉnh sửa hay xóa các records từ nó. Nó được đọc từ trái sang phải và được đảm bảo thứ tự các item.



Một nguồn dữ liệu sẽ ghi message vào log và một hoặc nhiều consumer khác sẽ đọc message từ log tại thời điểm họ lựa chọn.

Mỗi entry trong log được định danh bởi một con số gọi là offset, hay nói một cách dễ hiểu hơn, offset giống như chỉ số tuần tự trong một array vậy.

Vì chuỗi/offset chỉ có thể được duy trì trên từng node/broker cụ thể và không thể được duy trì đối với toàn bộ cluster, do đó Kafka chỉ đảm bảo sắp xếp thứ tự dữ liệu cho mỗi partition.

## **Parsistence data trong Kafa**

Kafka lưu trữ tất cả message vào disk (không hề lưu trên RAM) và được sắp xếp có thứ tự trong cấu trúc log cho phép kafka tận dụng tối đa khả năng đọc và ghi lên disk một cách tuần tự.

Nó là một cách lựa chọn khá phổ biến để lưu trữ dữ liệu trên disk mà vẫn có thể sử dụng tối đa hóa hiệu năng, có một số lý do chính dưới đây:

1. Kafka có một giao thức mà nhóm các message lại với nhau. Điều này cho phép các request network nhóm các message lại với nhau, giúp giảm thiểu chi phí sử dụng tài nguyên mạng, server, gom các message lại thành một cục và consumer sẽ tìm nạp một khối message cùng một lúc – do đó sẽ giảm tải disk cho [hệ điều hành](https://vsudo.net/blog/he-dieu-hanh).
2. Kafka phụ thuộc khá nhiều vào pagecache của hệ điều hành cho việc lưu trữ dữ liệu, sử dụng RAM trên máy một cách hiệu quả.
3. Kafka lưu trữ các messages dưới định dạng nhị phân xuyên suốt quá trình (**producer > broker > consumer**), làm cho nó có thể tận dụng tối ưu hóa khả năng zero-copy. Nghĩa là khi hệ điều hành copy dữ liệu từ pagecache trực tiếp sang socket, hoàn toàn bỏ qua ứng dụng trung gian là kafka.
4. Đọc/ghi dữ liệu tuyến tính trên disk nhanh. Vấn đề làm cho disk chậm hiện nay thường là do quá trình tìm kiếm trên disk nhiều lần. Kafka đọc và ghi trên disk tuyến tính, do đó nó có thể tận dụng tối đa hóa hiệu suất trên disk.

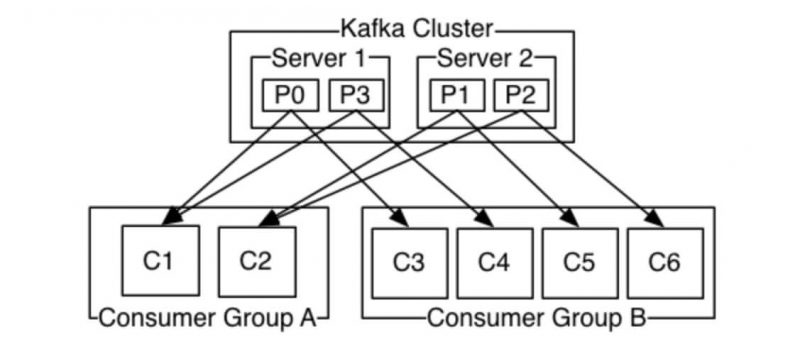
## **Consumer và Consumer Group**

Consumer đọc các messages từ bất kỳ partition nào, cho phép bạn mở rộng lượng message được sử dụng tương tự như cách các producer cung cấp message.

Consumer cũng được tổ chức thành các consumer groups cho một topic cụ thể – mỗi consumer bên trong group đọc message từ một partition duy nhất, để tránh việc có 2 consumer cùng xử lý đọc cùng một message 2 lần và toàn bộ group xử lý tất cả các message từ toàn bộ topic.

* Nếu bạn có số consumer**>** số partition, khi đó một số consumer sẽ ở chế độ rảnh rỗi bởi vì chúng không có partition nào để xử lý.
* Nếu bạn có số partition > số consumer, khi đó consumer sẽ nhận các message từ nhiều partition.
* Nếu bạn có số consumer = số partition, mỗi consumer sẽ đọc message theo thứ tự từ 1 partition.

Để dễ hiểu hơn, các bạn xem qua hình ảnh dưới đây



Trong bức ảnh ở trên, Server 1 giữ partition 0 và 3 và server 2 giữ các partition 1 và 2. Chúng ta có 2 consumer groups là A và B. Group A có 2 consumer và group B có 4 consumer. Consumer group A có 2 consumer, vậy nên mỗi consumer sẽ đọc message từ 2 partition.Trong consumer group B, số lượng consumer bằng số partition nên mỗi consumer sẽ đọc message từ 1 partition.

Kafka tuân theo các quy tắc được cung cấp bởi broker và consumer. Nghĩa là kafka không theo dõi các record được đọc bởi consumer và do đó không biết gì về hành vi của consumer. Việc giữ lại các messages trong một khoảng thời gian được cấu hình trước và nó tùy thuộc vào consumer, để điều chỉnh thời gian sao cho phù hợp. Bản thân consumer sẽ thăm dò xem Kafa có message nào mới hay không và cho Kafka biết những record nào chúng muốn đọc. Điều này cho phép chúng tăng/giảm offset mà consumer muốn, do đó nó có thể đọc lại các message đã được đọc rồi và tái xử lý các sự kiện trong trường hợp gặp sự cố.

**Ví dụ:** nếu Kafka được cấu hình để giữ các messages tồn tại trong một ngày và consumer bị down lâu hơn 1 ngày, khi đó consumer sẽ mất message. Tuy nhiên, nếu consumer chỉ bị down trong khoảng 1 giờ đồng hồ, khi đó nó hoàn toàn có thể bắt đầu đọc lại message từ offset mới nhất.

## **Vai trò của Zookeeper**

Zookeeper đóng vai trò là nơi lưu trữ dữ liệu phân tán dạng key-value. Nó được tối ưu hóa cho tác vụ đọc nhanh nhưng ghi chậm. Kafka sử dụng Zookeeper để thực hiện việc bầu chọn leader của Kafka broker và topic partition. Zookeeper cũng được thiết kế cho khả năng chịu lỗi cao, do đó Kafka phụ thuộc khá nhiều vào Zookeeper.

Nó cũng được sử dụng để lưu trữ tất cả metadata như là:

* Offset cho mỗi partition của consumer group
* ACL (Access control list) – được sử dụng cho việc giới hạn truy cập/ủy quyền
* Quota của consumer/producer – số lượng message tối đa mỗi giây
* Partition Leader và trạng thái của chúng

Producer và consumer không tương tác trực tiếp với Zookeeper để biết leader của partition hay những metadata khác, thay vào đó chúng sẽ truy vấn metadata tới Kafka broker – sau đó Kafka tương tác với Zookeeper và gửi phản hồi metadata về lại cho chúng.

## **Tại sao nên sử dụng Apache Kafka?**

Kafka là dự án opensoure, đã được đóng gói hoàn chỉnh, khả năng chịu lỗi cao, hiệu năng rất tốt và dễ dàng mở rộng mà không cần dừng hệ thống.

1. **So sánh giữa Kafka và RabbitMQ:**
   1. **Những điểm chung:**

Cả RabbitMQ và Apache Kafka đều truyền thông điệp giữa nhà sản xuất và người tiêu dùng thông qua hàng đợi (chủ đề). Tin nhắn có thể bao gồm bất kỳ loại thông tin nào, chẳng hạn như tin nhắn văn bản đơn giản kích hoạt chuỗi sự kiện trên các ứng dụng khác. Hệ thống nhắn tin lý tưởng để xây dựng nền tảng microservice, kết nối các thành phần khác nhau, chuyển công việc cho nhân viên từ xa và truyền dữ liệu theo thời gian thực.

Cả hai dịch vụ tin nhắn đều được coi là đã mature, với RabbitMQ có mặt trên thị trường từ năm 2007 và Kafka hoạt động từ năm 2011. Độ tin cậy và khả năng mở rộng của cả hai dịch vụ đều ngang bằng nhau, đó là lý do tại sao cả hai thường được so sánh với nhau. Sự khác biệt giữa hai dịch vụ này và nhu cầu ứng dụng của bạn là thứ quyết định dịch vụ nào sẽ sử dụng khi nào.

* 1. **Những điểm khác nhau chính:**

Không giống như hầu hết các hệ thống MQ, hàng đợi của Kafka là Persistent. Điều này có nghĩa là dữ liệu được gửi đến Kafka được lưu trữ cho đến khi một khoảng thời gian cụ thể trôi qua hoặc đạt đến giới hạn kích thước. Cho đến khi một trong hai điều đó xảy ra, tin nhắn vẫn còn trong hàng đợi ngay cả khi nó đã được Consume. Trong Kafka, tin nhắn có thể được phát lại hoặc sử dụng nhiều lần; một cài đặt có thể điều chỉnh hữu ích trong các tình huống khác nhau.

Mặt khác, RabbitMQ lưu trữ các tin nhắn cho đến khi ứng dụng nhận kết nối với hàng đợi và nhận nó. Máy khách có thể xác nhận thông báo tại thời điểm đó hoặc khi quá trình xử lý hoàn tất; một trong hai cách, khi tin nhắn được đánh dấu chọn, nó sẽ biến mất khỏi hàng đợi.

* + 1. **Replay and not Replay:**

Có thể phát lại messages trong Kafka, nghĩa là ứng dụng của bạn có thể đọc một messages nhiều lần. Việc tận dụng khả năng phát lại messages trong Kafka nên được thực hiện một cách cẩn thận. Ví dụ, lưu một đơn đặt hàng của khách hàng nhiều lần thường không phải là một ý kiến ​​hay. Mặt khác, giả sử người tiêu dùng của bạn có một lỗi và một phiên bản mới hơn cần được triển khai. Có thể xử lý lại một số hoặc tất cả các thư tại thời điểm đó là một lợi ích lớn.

Không thể phát lại messages trong RabbitMQ, vì chúng sẽ bị xóa sau khi được acked. Tuy nhiên, một ứng dụng Client RabbitMQ có thể lấy (xác nhận phủ định) một thông báo khi nó không xử lý được thông báo; điều này rất hữu ích trong trường hợp người tiêu dùng gặp lỗi tạm thời. messages sẽ chỉ được thêm trở lại hàng đợi.

* + 1. **Protocol:**

Sự phức tạp mà các messages  được định tuyến là một khác biệt khác giữa hai Kafka và RabbitMQ Trong khi RabbitMQ có một phương pháp định tuyến phức tạp hơn, Kafka lại rất đơn giản. Đơn giản là tốt, phải không? Không phải trong trường hợp này, một trong những lợi ích lớn nhất của RabbitMQ là tính linh hoạt trong việc định tuyến tin nhắn.

RabbitMQ có bốn tùy chọn khác nhau để định tuyến tin nhắn - direct, topic, fanout và header. Direct exchange định tuyến thông điệp đến tất cả các hàng đợi khớp chính xác với khóa định tuyến của nó(routing keys). Topics cho phép đối sánh ký tự đại diện và đối sánh chính xác thông qua khóa định tuyến(routing keys). Fanout trao đổi thông điệp phát sóng đến mọi hàng đợi được ràng buộc với Exchange. Header Exchange sử dụng thông tin được tìm thấy trong header của message và các giá trị tùy chọn, rất giống với trao đổi chủ đề nhưng không có khóa định tuyến. Xem bài viết này để biết thêm thông tin về các loại Exchange khác nhau.

Mặt khác, Kafka không hỗ trợ định tuyến và thay vào đó dựa vào các phân vùng chứa các thông báo theo một trình tự hoàn toàn không thể thay đổi được. Tạo định tuyến động của riêng bạn với sự trợ giúp của các luồng Kafka là một tùy chọn nhưng không phải là mặc định với Kafka và yêu cầu sử dụng các nhóm khách hàng và các chủ đề liên tục.

* + 1. **Sự ưu tiên:**

Trong RabbitMQ, một hàng đợi có thể được thiết lập để có nhiều mức độ ưu tiên. Tùy thuộc vào mức độ ưu tiên của tin nhắn, nó được đặt trong hàng đợi thích hợp. Trong Kafka, một tin nhắn không thể được gửi với mức độ ưu tiên, cũng như không thể gửi nó theo thứ tự ưu tiên. Tất cả các tin nhắn trong Kafka được coi là bình đẳng và được gửi theo thứ tự mà chúng được nhận cho dù người tiêu dùng bận rộn đến đâu.

* + 1. **Acking:**  
       Acknowledgment - hay acking -  là tín hiệu mà hai tiến trình cung cấp cho nhau khi nhận được thông điệp được gửi hoặc xử lý. Cả nhà sản xuất hỗ trợ RabbitMQ và Apache Kafka đều xác nhận thông báo đã đến được nhà môi giới một cách an toàn. Trong RabbitMQ, một tin nhắn có thể được coi là đã gửi khi nó được gửi đi hoặc khi nó đã được người tiêu dùng nhận được. Ngược lại, các máy khách RabbitMQ cũng có thể ghi lại (xác nhận phủ định) một thông báo nếu nó không được xử lý, đưa nó trở lại hàng đợi như thể nó mới. Kafka duy trì một khoảng bù cho mỗi thư trong một phân vùng, với vị trí đã cam kết là khoảng bù cuối cùng được lưu. Nếu quá trình không thành công và khởi động lại, phần bù đắp này sẽ phục hồi. Người tiêu dùng ở Kafka có thể cam kết bù đắp theo định kỳ hoặc thủ công. Làm thế nào điều này được theo dõi khác nhau trong các phiên bản khác nhau của Kafka.
    2. **Queuespeed:**  
       Đối với RabbitMQ, tốc độ của hàng đợi là lớn nhất khi hàng đợi trống. Kafka được thiết kế để chứa một lượng lớn tin nhắn, vì vậy độ trống không phải là một yếu tố ảnh hưởng đến tốc độ. Bật lazy queue trong RabbitMQ là một cách tuyệt vời để tạo một cluster ổn định hơn nếu bạn cho rằng Consumer của mình sẽ không theo kịp tốc độ của Producer.
    3. **Scaling:**

Cả RabbitMQ và Kafka đều có thể mở rộng theo cách riêng của nó, cho phép bạn điều chỉnh số lượng consumer, sự ảnh hưởng của broker hoặc thêm nhiều nút hơn theo yêu cầu.

* + Scaling Consumer:

Việc publish nhanh hơn mức mà Consumer của bạn có thể consume trong RabbitMQ có thể dẫn đến hàng đợi tăng lên thành hàng triệu tin nhắn và cuối cùng là hết bộ nhớ. Trong trường hợp này, mở rộng số lượng Consumer đang xử lý tin nhắn là một cách dễ dàng để giải quyết tình huống này. Mỗi hàng đợi trong RabbitMQ có thể có nhiều người tiêu dùng, tất cả những người này đều có thể cạnh tranh để tiêu thụ các Message trong hàng đợi.

Kafka yêu cầu phân vùng topic để phân phối người tiêu dùng hiệu quả hơn, trong đó mỗi consumer trong một nhóm được dành riêng cho một hoặc nhiều phân vùng. Các phân vùng có thể được đặt để gửi các tập hợp tin nhắn khác nhau tùy thuộc vào id người dùng, vị trí hoặc các yếu tố khác.

* + Scaling Broker:

Kafka có lợi thế khi mở rộng quy mô. Nó được xây dựng với tính đến khả năng mở rộng lớn. Thêm nhiều nút hơn vào cụm hoặc thêm nhiều phân vùng hơn cho các chủ đề là những cách dễ dàng để mở rộng quy mô trong Kafka. Trong RabbitMQ, mở rộng quy mô theo chiều dọc - tăng thêm sức mạnh - là cách dễ nhất để mở rộng quy mô. Vì sẽ luôn có giới hạn về độ lớn của máy bạn có thể mua, nên việc mở rộng quy mô theo chiều ngang ở Kafka là một lợi thế.

Tuy nhiên, cả hai dịch vụ tin nhắn đều có thể hỗ trợ khối lượng tin nhắn lớn mỗi giây mà không gặp bất kỳ sự cố nào, vì RabbitMQ hoặc Kafka sẽ hết dung lượng là một kịch bản hiếm gặp.

* + 1. **Log Compaction (Kafka as Database):**

Điều này không tồn tại trong RabbitMQ nhưng là một tính năng làm cho Apache Kafka nổi bật. Log compaction đảm bảo rằng giá trị đã biết cuối cùng cho mỗi khóa thông báo được giữ trong hàng đợi cho một phân vùng chủ đề duy nhất. Nói cách khác, Kafka giữ phiên bản mới nhất của tin nhắn, xóa các phiên bản cũ hơn có cùng khóa. Log compaction đảm bảo rằng thông tin mới nhất có sẵn ngay lập tức, chẳng hạn như nếu chúng tôi đang hiển thị trạng thái mới nhất của một cụm trong số hàng nghìn cụm đang chạy. Thay vì lưu trữ liệu một cụm có đang phản hồi hay không, chỉ lưu trữ trạng thái cuối cùng.

* + 1. **Complexity:**

Ý kiến ​​của các developer mà chúng tôi làm việc cùng là kiến ​​trúc của Kafka phức tạp hơn, vì nó bao gồm nhiều khái niệm hơn như topics, partitions, và message offsets. Làm quen với các nhóm người tiêu dùng và xử lý các khoản bù đắp là điều kiện tiên quyết khi làm việc với Kafka. Đặc biệt, việc xử lý lỗi là một sự phức tạp khi làm việc với Kafka, đòi hỏi nhiều thời gian hơn và phức tạp hơn so với trong RabbitMQ.

* + 1. **Use-case:**
  + RabbitMQ:

Hai trường hợp sử dụng chính nổi bật để đưa ra lựa chọn cho RabbitMQ: các tác vụ chạy lâu dài và tích hợp giữa các ứng dụng microservice. Tuy nhiên, nói chung, RabbitMQ là một message broker kiểu pub/sub đơn giản và truyền thống. Nó sẽ mở rộng quy mô hơn hầu hết các hệ thống sẽ cần và dễ sử dụng ngay sau khi cài đặt. RabbitMQ cũng là một lựa chọn tốt cho các hệ thống có yêu cầu đơn giản và những hệ thống không cần lưu giữ và phát trực tuyến.

* Long running task:  
  Về bản chất, Message queue là quá trình xử lý dữ liệu không đồng bộ - có nghĩa là chúng cho phép đặt message vào hàng đợi mà không cần xử lý nó ngay lập tức. Một ví dụ về tình huống như vậy là dịch vụ chia tỷ lệ hình ảnh cho phép người dùng tải hình ảnh lên một trang web, với mục đích chia tỷ lệ hình ảnh và gửi lại cho người dùng qua email. Các kiến ​​trúc microservice dựa trên sự kiện xử lý hàng trăm triệu người dùng là hoàn hảo cho RabbitMQ.
* Integration between application:  
  Các dịch vụ như MapQuest, hỗ trợ 23,1 triệu người dùng di động hàng tháng, dựa vào RabbitMQ để hỗ trợ các topic trải rộng trên nhiều hàng đợi. Đóng vai trò là người trung gian để giao tiếp giữa các ứng dụng, RabbitMQ cho phép các hệ thống tránh tắc nghẽn khi chuyển các thông báo qua lại.
  + Kafka:  
    Framework hoàn hảo để lưu trữ, đọc, đọc lại và phân tích dữ liệu phát trực tuyến thì Apache Kafka là lựa chọn tốt hơn. Các hệ thống được kiểm toán hoặc cần lưu trữ vĩnh viễn tin nhắn là nơi Kafka được ứng dụng tốt nhất.
* Analysis, tracking, ingestion, logging, security

Trong tất cả các tình huống này, một lượng lớn dữ liệu phải được thu thập, lưu trữ và xử lý. Các nền tảng cần cung cấp tính năng tìm kiếm, phân tích hoặc kiểm tra lượng lớn dữ liệu hoàn toàn hài lòng với hiệu suất của Kafka. Quay trở lại nguồn gốc của Kafka, ban đầu nó được dùng để theo dõi hoạt động của trang web như lượt xem trang, tải lên và các hành động khác của người dùng.

Với Kafka, các nhà sản xuất gửi dữ liệu đến một nơi duy nhất, nơi đó sau đó sẽ được phụ trách bởi một loạt các dịch vụ phụ trợ sử dụng dữ liệu theo yêu cầu. Các hệ thống phân tích dữ liệu chính và hệ thống lưu trữ / tìm kiếm tích hợp các chức năng của chúng với Kafka.

* Real-Time processing

Khả năng thông lượng cao của hệ thống Kafka đẩy các luồng dữ liệu vào các dịch vụ mục tiêu, các dịch vụ này đang kéo dữ liệu qua trong thời gian thực. Các dịch vụ xuất bản như Spotify và Rabobank xuất bản thông tin bằng các dịch vụ thời gian thực của Kafka. Thông lượng cao và thời gian thực kết hợp để tạo thành một ứng dụng mạnh mẽ cho nhật ký, số liệu và hơn thế nữa.