# **Báo cáo kiến thức tìm hiểu được**

## Message queue

Message Queue là một thành phần quan trọng thường sử dụng trong các hệ thống lớn (Ví dụ Yahoo, Tiki) hoặc phần mềm theo kiến trúc microservice. Tuy vậy, nếu không gặp các dự án có hệ thống lớn thì sẽ không biết rõ Message Queue là gì, được sử dụng với mục đích gì!

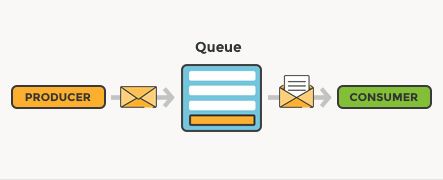
**Messege Queue là gì?**

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:

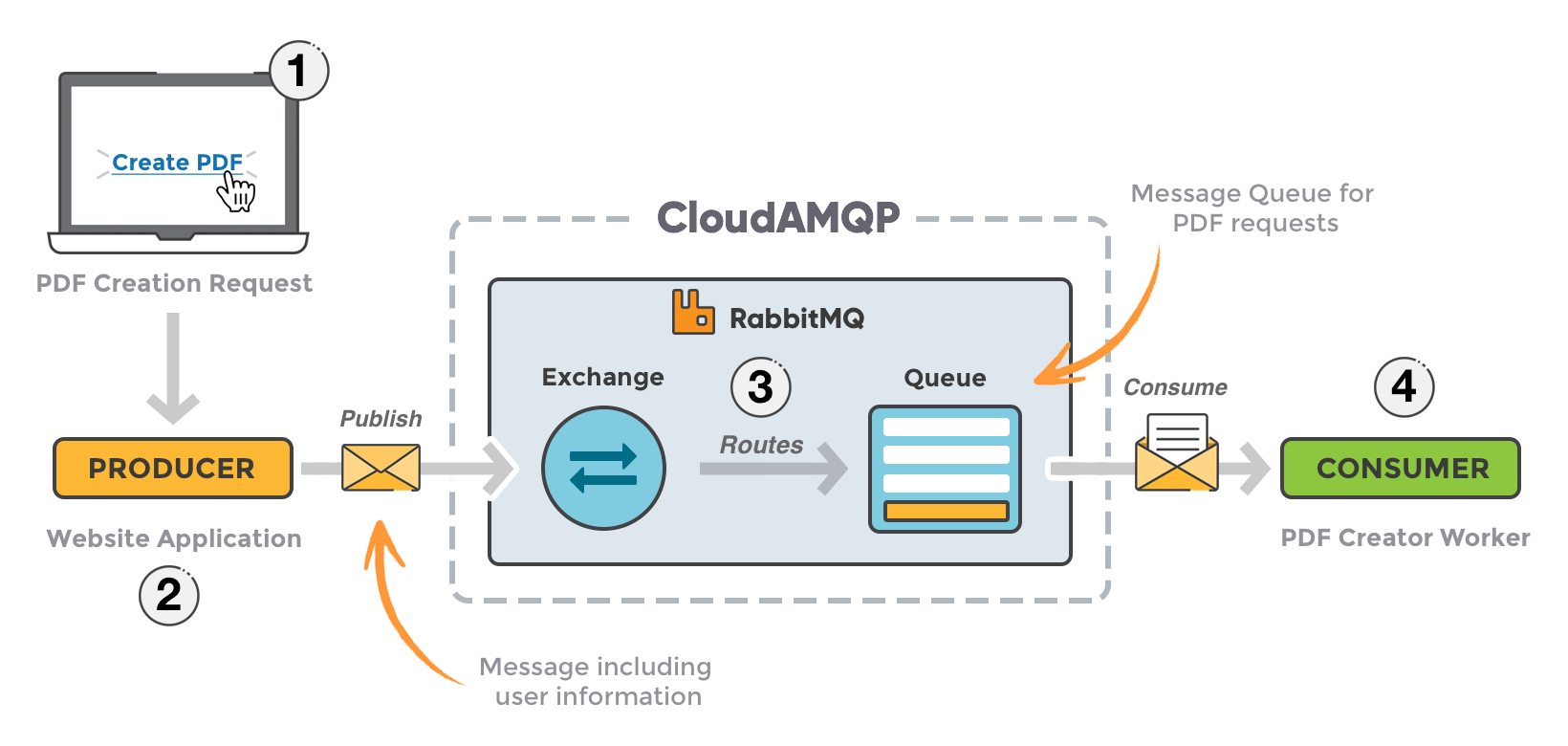
* Kafka
* Pulsar
* RabitMQ
* ActiveMQ
* SQS
* ZeroMQ
* MSMQ
* IronMQ
* Kinesis
* RocketMQ

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

Trong các hệ thống dùng kiến trúc microservice, 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:

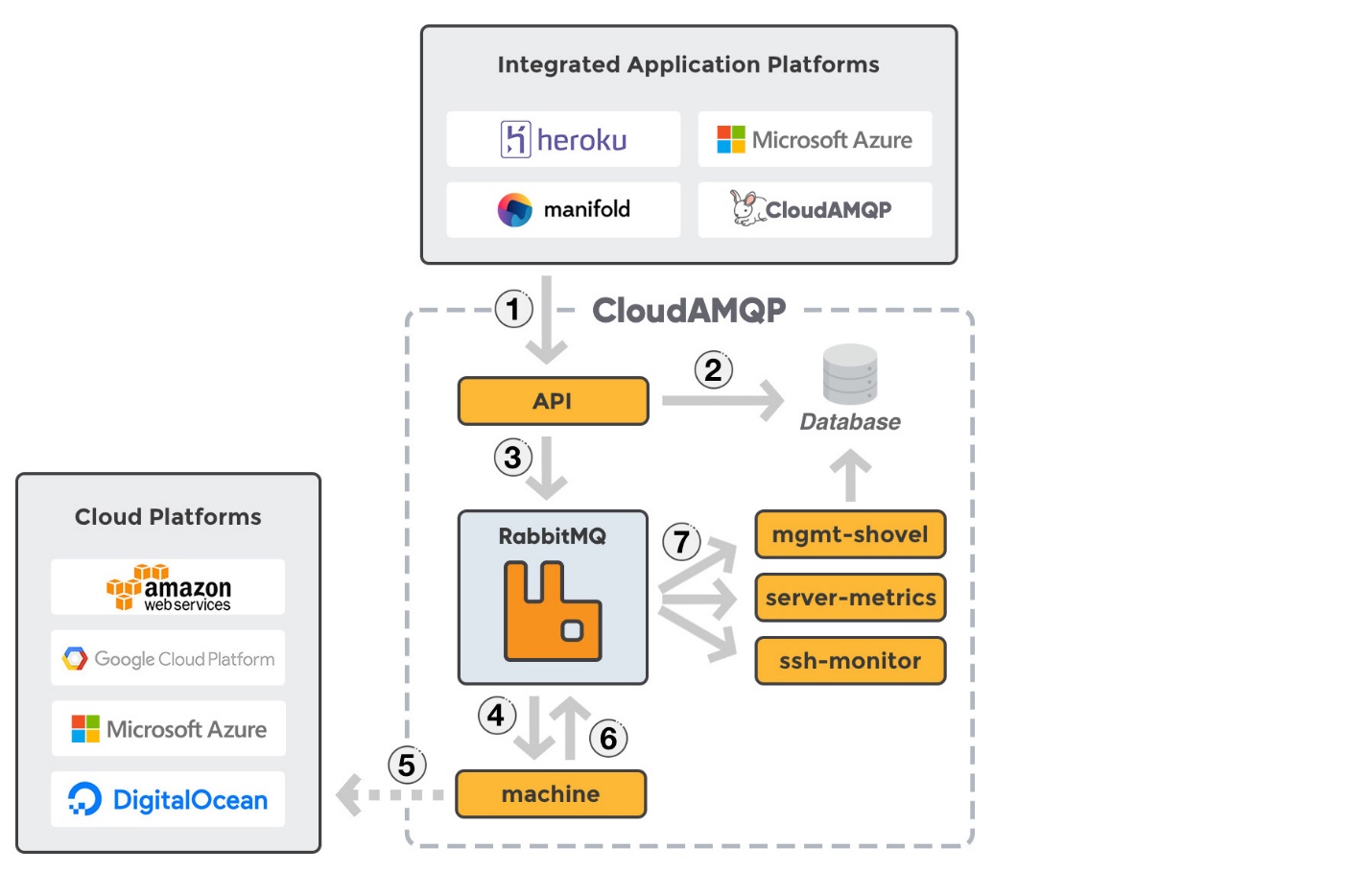
* **API service**: Là 1 producer. Nhận thông tin (URL Video) từ phía người dùng và đưa thông tin này vào message queue
* **Processing Service**: Vừa là consumer vừa là producer. Service này đọc URL Video từ message queue, bắt đầu tải file Video về và encode lại, lưu vào server. Sau khi encode xong, nó 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.
* **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.



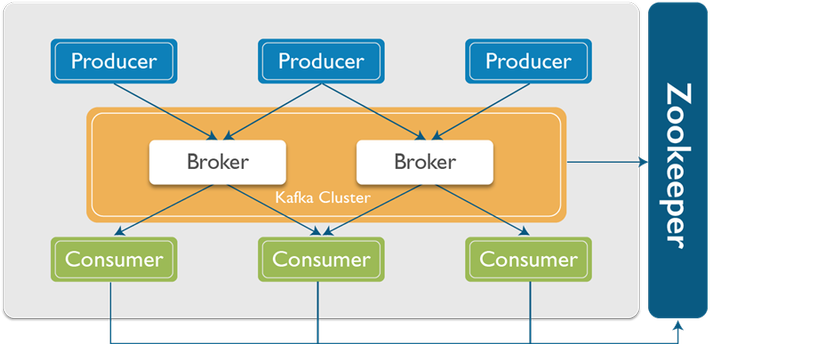
**Điểm cần lưu ý về Message Queue**

* **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.

## KAFKA

### 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.



### 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.

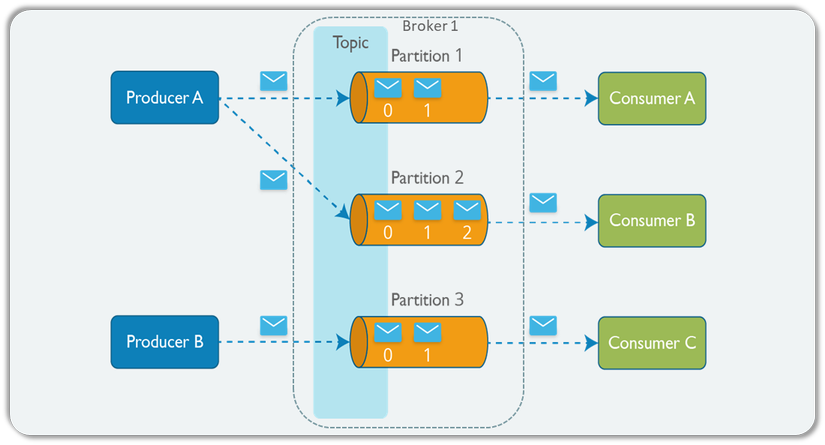
Với Message broker, ứng dụng nguồn (producer) gửi một message đến một server process mà nó có thể cung cấp việc sắp xếp dữ liệu, routing (Định tuyến), message translation, persistence và delivery tất cả các điểm đến thích hợp (consumer).

Có 2 hình thức giao tiếp cơ bản với một Message Broker:

* Publish và Subscribe (Topics)
* Point-to-Point (Queues)

### Định nghĩa trên được giải thích bằng các khái niệm sau:

* **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.



### Tại sao nên sử dụng Apache Kafka?

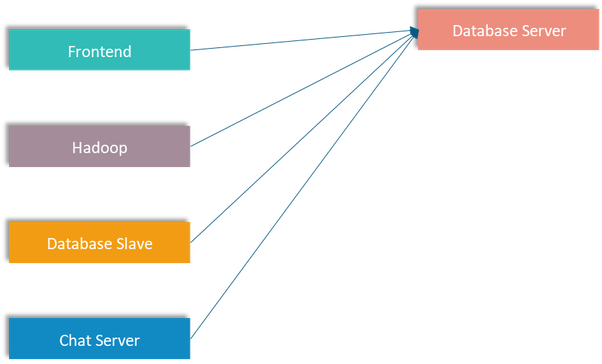
Kafka là dự án opensoure, đã được đóng gói hoàn chỉnh, khả năng chịu lỗi cao, hiệu năng rất tốt và dễ dàng mở rộng mà không cần dừng hệ thống.

**Một vài use case cho kafka:**

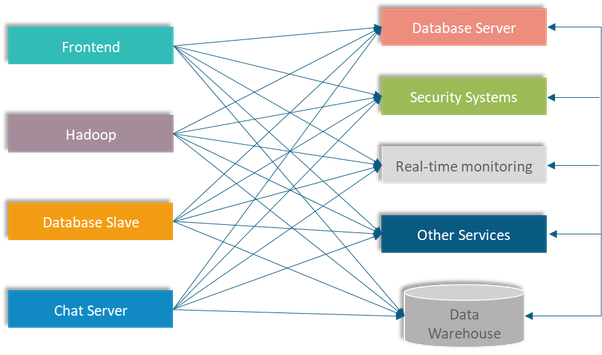
* Sử dụng như một hệ thống message queue thay thế cho ActiveMQ hay RabbitMQ
* **Website Activity Monitoring**: theo dõi hoạt động của website
* **Stream Processing**: Kafka là một hệ thống rất thích hợp cho việc xử lý dòng dữ liệu trong thời gian thực. Khi dữ liệu của một topic được thêm mới ngay lập tức được ghi vào hệ thống và truyền đến cho bên nhận. Ngoài ra Kafka còn là một hệ thống có đặc tính duribility dữ liệu có thể được lưu trữ an toàn cho đến khi bên nhận sẵn sàng nhận nó.
* **Log Aggregation**: tổng hợp log
* **Metrics Collection**: thu thập dữ liệu, tracking hành động người dùng như các thông số như page view, search action của user sẽ được publish vào một topic và sẽ được xử lý sau
* **Event-Sourcing**: Lưu lại trạng thái của hệ thống để có thể tái hiện trong trường hợp system bị down.

### Use case Kafka cho thương mại điện tử

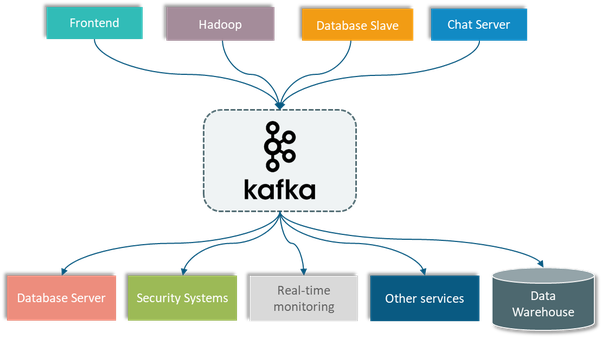
Một hệ thống thương mại điện tử có nhiều server thực hiện các tác vụ khác nhau. Tất cả các server này đều sẽ giao tiếp với database server để đọc ghi dữ liệu. Vì vậy sẽ có rất nhiều data pipeline kết nối từ rất nhiều server khác đến database server này. Cơ cấu như sau:



Nhìn đơn giản vậy thôi chứ đây là hệ thống nhỏ, đối với hệ thống lớn hơn thì nó sẽ như vầy:



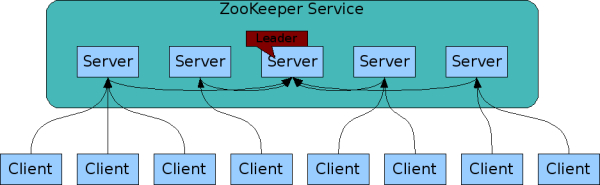
Lúc này data pipeline sẽ phức tạp khủng khiếp do gia tăng lượng hệ thống server. Lúc này nếu ta sử dụng Kafka tách rời các data pipeline giữa các hệ thống để làm cho việc giao tiếp giữa các hệ thống trở nên đơn giản hơn và dễ quản lý hơn.



### Zookeeper là gì?

Zookeeper là một dịch vụ (một server) tập trung cho việc duy trì thông tin cấu hình, đặt tên, cung cấp sự đồng bộ phân tán, cung cấp dịch vụ nhóm. Nói cách khác, Zookeeper dịch vụ phối hợp phân tán cho các ứng dụng phân tán. Zookeeper là một mã nguồn mở (open source) và được viết bằng Java.

Lợi ích và ứng dụng của Zookeeper



Client kết nối tới một ZooKeeper server (một single node). Client duy trì kết nối TCP thông qua việc gửi nhận request, response, lắng nghe sự kiện… Nếu server bị chết thì client sẽ được kết nối tới một server khác.

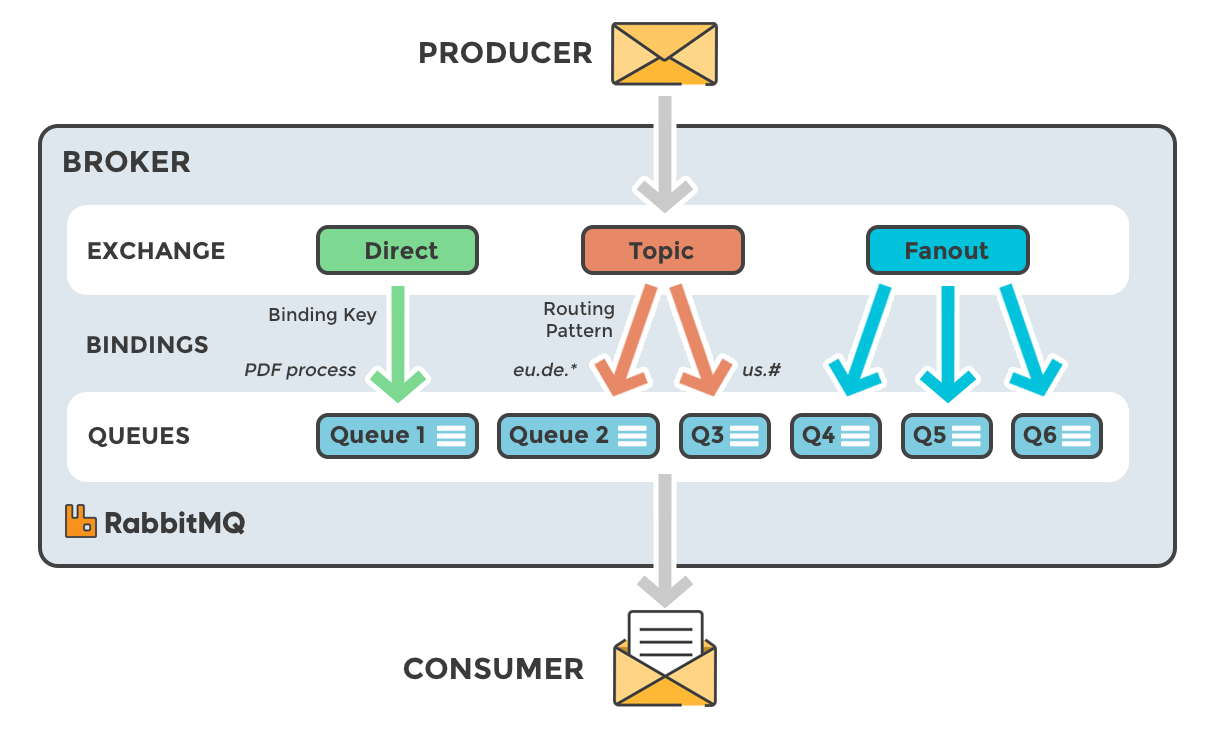
## RabbitMQ

### 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.

### RabbitMQ và trong messaging nói chung sử dụng những thuật như sau:

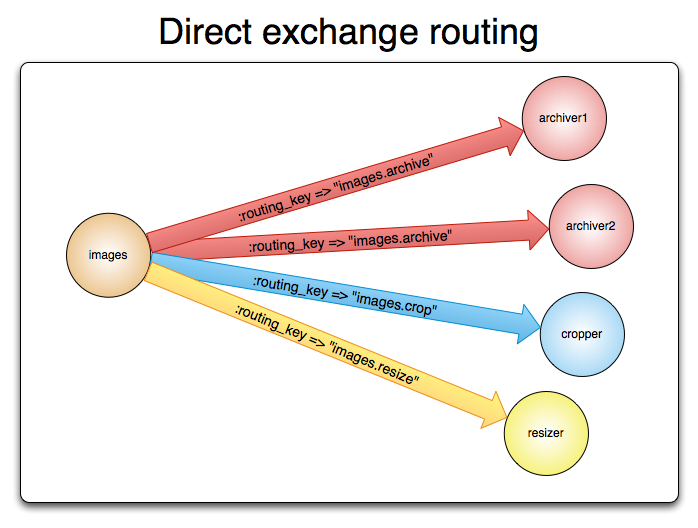
* **Producer**: Ứng dụng gửi message.
* **Consumer**: Ứng dụng nhận message.
* **Queue**: Lưu trữ messages.
* **Message**: Thông tin truyền từ Producer đến Consumer qua RabbitMQ.
* **Connection**: Một kết nối TCP giữa ứng dụng và RabbitMQ broker.
* **Channel**: Một kết nối ảo trong một Connection. Việc 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**.**



### Các loại Exchange

**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.



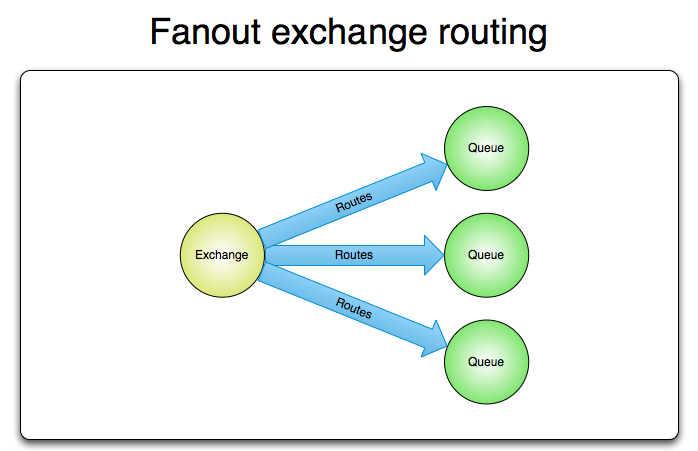
**Default Exchange**

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".

**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.



**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

**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.

## So sánh Kafka và RabbitMQ

### Phân tán và song song

Cả hai đều là giải pháp phân tán tốt, nhưng Kafka và RabbitMQ có một số điểm khác biệt.

RabbitMQ có thể mở rộng về số lượng các consumer, có nghĩa là với mỗi thể hiện của hàng đợi, bạn có thể có nhiều consumer.

### Tính sẵn sàng cao (highly available)

Cả hai giải pháp đều có tính sẵn sàng cao, nhưng có vẻ Kafka tiến nhanh hơn một bước bằng việc sử dụng Zookeeper để quản lý trạng thái của các cụm (cluster).

### Hiệu năng

**Kafka** tận dụng sức mạnh của truy xuất tuần tự I/O và yêu cầu ít phần cứng hơn. Việc này giúp đáp ứng được thông lượng cao: hàng triệu thông điệp mỗi giây với số lượng node nhỏ hơn.

**RabbitMQ** cũng có thể xử lý hàng triệu thông điệp mỗi giây nhưng yêu cầu hơn 30 nodes.

### Replication

Theo thiết kế, **Kafka** tự động nhân bản các broker. Khi broker chính (master broker) ngưng hoạt động, tất cả xử lý sẽ tự động chuyển qua broker khác là bản sao đầy đủ của broker ngưng hoạt động. Khi đó, các thông điệp được giữ nguyên vẹn.

Trong hàng đợi **RabbitMQ** không có cơ chế tự động nhân bản, ta cần phải cấu hình thủ công.

### Đa subscriber

Trong **Kafka**, thông điệp có thể được đăng ký bởi nhiều consumer.

Còn với **RabbitMQ**, thông được chỉ được đăng ký bởi 1 consumer. Và khi đã consume, thông điệp sẽ biến mất và không thể truy cập được nữa.

### Thứ tự thông điệp

**Kafka** phân vùng thông điệp nên bạn có thể lấy thứ tự thông điệp. Thông điệp được điều hướng đến các chủ đề bởi khóa thông điệp, do vậy khi chọn đúng khóa, bạn có thể lấy được chủ đề cho vất cứ khóa nào với các thông điệp đã được sắp xếp. Điều tương tự không thể làm được trong **RabbitMQ**. Ta chỉ có thể bắt trước bằng cách định nghĩa nhiều hàng đợi và gửi mỗi thông điệp đến các hàng đợi khác nhau. Khi mở rộng, việc này có thể gây ra nhiều khó khăn.

**Compaction log**: Nếu thông điệp cùng khóa trở lại nhiều lần thì Kafka chỉ lưu giá trị cuối cùng trong log và xóa các thông điệp cũ.

### Giao thức thông điệp

**RabbitMQ** chỉ hỗ trợ các giao thức hàng đợi chuẩn như AMQP, STOMP (Text based), MQTT (lightweight publish/subscribe messaging) và HTTP trong khi **Kafka** hỗ trợ thông điệp primitive (int8, int16, int32, int64, string, arrays) và thông điệp binary.

### Message lifetime

Bởi vì **Kafka** là log, message luôn ở đó, bạn có thể kiểm soát điều này bằng cách xác định chính sách lưu giữ tin nhắn. **RabbitMQ** là một hàng đợi, các tin nhắn bị xóa sau khi được sử dụng và xác nhận đến.

Trong **RabbitMQ**, bạn có thể định cấu hình các thông báo là bền vững, đánh dấu hàng đợi là lâu bền và các thông báo là bền vững, Từ tài liệu: Đảm bảo độ bền không mạnh

### Xác nhận thông điệp

Trong cả 2 giải pháp, producer nhận xác nhận rằng thông điệp đã đến hàng đợi/chủ đề và chỉ consumer gửi một xác nhận khi thông điệp đã được consume thành công. Do vậy bạn có thể đảm bảo rằng thông điệp không bị mất bằng cách nào đó.

### **Định tuyến linh hoạt đến hàng đợi/ chủ đề**

Thông điệp trong Kafka được gửi đến chủ đề bởi khóa, còn trong RabbitMQ thì có nhiều lựa chọn hơn, ví dụ bởi biểu thức chính quy, wildcard.

### **Ưu tiên thông điệp**

Trong RabbitMQ, bạn có thể định nghĩa mức ưu tiên cho thông điệp và thu nhận thông điệp có mức ưu tiên cao hơn trước. Việc này khó thực hiện trong Kafka (có thể thực thiện theo khóa thông điệp nhưng khi mở rộng, việc này có thể khó khăn hơn)