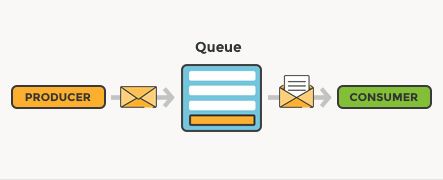
## Message queue là gì?

Vậy Message queue là gì và nó hoạt động như thế nào?

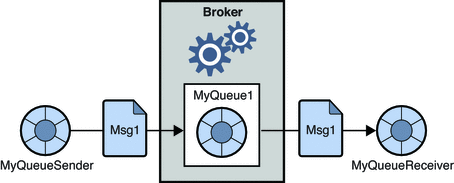
Message queue là một kiến trúc cung cấp giao tiếp không đồng bộ. Ý nghĩa của queue ở đây chính là 1 hàng đợi chứa message chờ để được xử lý tuần tự theo cơ chế vào trước thì ra trước (FIFO - First In First Out). Một message là các dữ liệu cần vận chuyển giữa người gửi và người nhận. Vậy có thể hiểu đơn giản, message queue giống như một hòm thư email của chúng ta. Email có lẽ là ví dụ tốt nhất về giao tiếp không đồng bộ. 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. 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 cơ bản của message queue rất đơn giản, bao gồm các thành phần như sau:

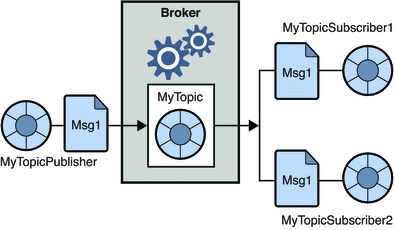
* Message: Thông tin được gửi (có thể là text, binary hoặc JSON)
* Producer: Service tạo ra thông tin, đưa thông tin vào message queue.
* Message Queue: Nơi chứa những message này, cho phép producer và consumer có thể trao đổi với nhau
* Consumer: Service nhận message từ message queue và xử lý
* Một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

## Các loại message queue

Trong các hệ thống của chúng ta, nhiều khi ta cần tới hai hoặc nhiều các service độc lập cùng xử một yêu cầu, vậy phải làm thế nào nhỉ? Đừng lo, vì message queue có những cơ chế khác nhau giúp giải quyết vấn đề này.

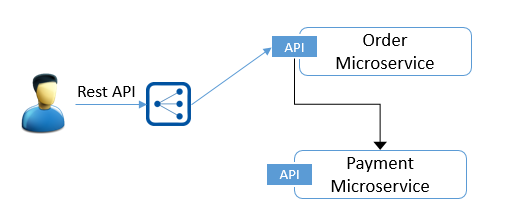
**1. Point-to-poin** Message queue có thể là kiểu point-to-point, tức là khi đó ta chỉ có một hàng đợi và một consumer duy nhất dể xử lý các tin nhắn trong hàng đợi:

**2.Publisher-Subscriber**

Ngoài ra, message queue có thể sử dụng định dạng Publisher-Subscriber, trong đó 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:

Hãy xem xét một ví dụ cụ thể sau

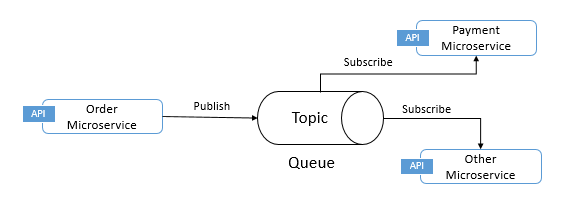
Service Order và Payment exposed các API để client hoặc các service khác gọi đến nó



Khi client call REST API request để thực hiện một Order. Order service nhận request xử lý business logic rồi gọi tới Payment Service qua API để thực hiện Payment.

Mô hình trên là loại mô hình kết nối Point-To-Point, các service kết nối trực tiếp với nhau thông qua API end-point. Tuy nhiên, mô hình này chỉ áp dụng với hệ thống có số lượng service nhỏ. Nếu số lượng service tăng, giao tiếp kiểu này sẽ trở nên rắc rối phức tạp, khó quản lý

Để giải quyết vấn đề trên, Microservice sử dụng mô hình Message Queue để giao tiếp giữa các service



Trong mô hình này, các Microservice không giao tiếp trực tiếp với nhau mà thông qua hệ thống Message Queue, giao tiếp bất đồng bộ

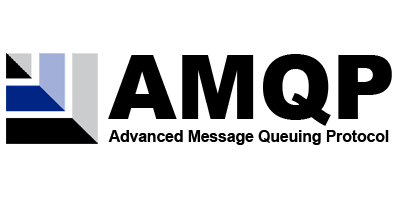
* Order Microservice **publish** một message đến Message queue theo một topic nào đó.
* Payment Microservice và các Microservice sẽ **subscribe** các message theo một topic cụ thể. Ví dụ Payment Microservice subscribe message topic “ABC” thì chỉ khi Order Microservice gửi message ABC thì nó mới nhận.

Cơ chế này tương tự như cách bạn gửi thư từ A đến B. A sẽ không đem thư đến tận nơi cho B mà gửi qua bưu điện, ghi rõ địa chỉ người nhận. Bưu điện đóng vai trò như một Message Broker để phát thư đến người nhận theo địa chỉ đã cho.

Cách này tách biệt người gửi và người nhận, và người truyền tin trung gian sẽ lưu tin nhắn đến khi người nhận có thể xử lý.

Giao tiếp giữa người gửi/ người nhận được tạo ra bởi message broker qua các tiêu chuẩn truyền tin bất đồng bộ như AMQP, MQTT.

**Advance message queue protocol (AMQP)**



AMQP một giao thức internet mở và được chuẩn hóa để truyền message tin cậy giữa các ứng dụng hoặc tổ chức. AMQP giúp các chuyên gia CNTT xây dựng một hệ sinh thái tin nhắn đồng nhất, đa dạng, kết nối các hệ thống một cách tương tác và hợp tác.



AMQP kết nối qua:

* Tổ chức - ứng dụng trong các tổ chức khác nhau
* Công nghệ - ứng dụng trên các nền tảng khác nhau
* Thời gian: Hỗ trợ giao tiếp bất đồng bộ
* Không gian - hoạt động ở khoảng cách xa hoặc trên các mạng nghèo nàn

Ngoài ra AMQP được thiết kế với các đặc điểm chính sau



**Ứng dụng của AMQP trong Microservice**

Vì những ưu thế của AMQP, nên nó đã được sử dụng trong Microservice để thực hiện communication giữa các service

## Apache Kafka là gì?

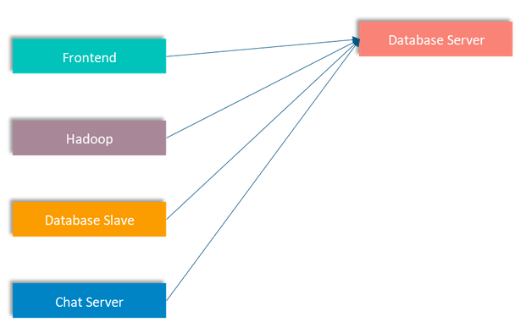
Producer gửi một message đến một topic (topic là một chủ đề bao gồm các message - hiểu đơn giản là ngôn ngữ giao tiếp giữa producer vs consumer). Các consumer theo dõi (subscribe) một topic và bắt đầu nhận message từ nó. Khi một topic được phân vùng thì mỗi vùng sẽ có một thể hiện consumer riêng. Các thể hiện trên cùng một consumer được gọi là một consumer group.

Mỗi partition là một chuỗi log, có thứ tự và không thể thay đổi (immutable).

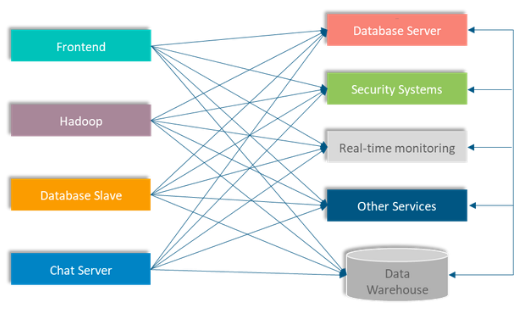
Mỗi message trong partition đều có id tăng dần, gọi là offset. Tại offset này, consumer có thể lựa chọn để đọc.

Kafka cluster sẽ lưu lại mọi message đã được published, cho dù message đó đã consume hay chưa.

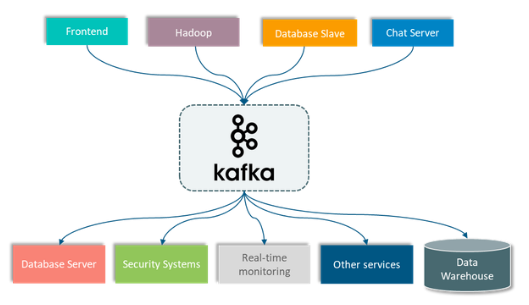
Giả sử bạn xây dựng một hệ thống thương mại điện tử như Shopee có nhiều máy chủ thực hiện các công việc khác nhau, tất cả máy chủ này đều muốn giao tiếp với database server, vì vậy chúng ta sẽ có nhiều data pipeline kết nối các server khác đến database server này, hình ảnh minh họa như sau:



Nhưng trong thực tế, hệ thống thương mại điện tử sẽ còn phải kết nối đến một vài server khác nữa như là:



Như bạn thấy ở ảnh trên, data pipeline đang trở nên phức tạp theo sự gia tăng của số lượng hệ thống. Để giải quyết vấn đề này thì kafka ra đời. Kafka tách rời các data pipeline giữa các hệ thống để làm cho việc communicate giữa các hệ thống trở nên đơn giản hơn và dễ quản lý hơn.



* Kafka là một **message broker** chạy trên hệ thống phân tán. Kafka đang được phát triển và bảo trì bởi tổ chức [Apache](https://www.apache.org/), do đó hay còn gọi là Apache Kafka. Kafka có mã nguồn mở. Tên Kafka được đặt theo tên của một nhà văn / tiểu thuyết gia nổi tiếng [Franz Kafka](https://en.wikipedia.org/wiki/Franz_Kafka).

## Giải thích message broker

* Message broker mình không biết nên dịch qua tiếng Việt như thế nào cho sát nghĩa. **Message** nghĩa là tin nhắn, còn **broker** nghĩa là môi giới. Về cơ chế, ta có thể hiểu nôm na rằng các gói dữ liệu (message) sẽ được module message broker đón nhận và tổ chức lưu trữ (trên vùng nhớ hoặc đĩa), sau đó sẽ có một hệ thống khác truy cập message broker để lấy ra các gói dữ liệu này để xử lý.

Trong cùng lĩnh vực message broker, ta còn có một "đối thủ" khác của Kafka, đó chính là [RabbitMQ](https://www.rabbitmq.com/).

## Cơ chế vận hành của Kafka

Vai trò của Kafka là kết nối giữa producer và consumer. Producer là các đơn vị có data, message cần gửi đi. Consumer là các đơn vị sẽ tiêu thụ data, message. Kafka tổ chức phân loại các message theo chủ đề (topic).

Nói vui rằng message broker (Kafka) hoạt động như một anh môi giới nhà đất chính hiệu. Anh ta cất giữ nhiều tài liệu (message) bất động sản của nhiều người rao bán nhà (producer), cứ có ai cần bán hay cho thuê mặt bằng gì đều đẩy thông tin (message) đến cho anh ta (message broker). Sau đó, những người có nhu cầu (consumer) về bất động sản sẽ liên hệ anh môi giới này để lấy thông tin (message) phù hợp với nhu cầu / chủ đề (topic) mình đang có. Những người tiêu thụ có cùng nhu cầu có thể được tổ chức phân nhóm (group). Minh vừa giải thích cách vận hành của Kafka gần theo cuộc sống.

**Kafka là một hệ thống phân tán**

Apache Kafka được phát triển trên nền tảng Apache ZooKeeper. ZooKeeper là một nền tảng mã nguồn mở quản lý hệ thống phân tán (nhiều máy tính). Do đó, Kafka có khả năng scale up chạy trên nhiều máy.

Tính chất của Kafka:

* **Fast**: cực kỳ nhanh @ gửi nhận và đọc message.
* **Scalable**: khả năng mở rộng lên nhiều máy. Trào lưu máy tính hiện tại là vận hành hệ thống lớn trên nhiều máy thay vì đầu tư phần cứng vào một máy tính cấu hình cực mạnh.
* **Reliable**: tính tin cậy, nội dung message sẽ được backup trên nhiều máy trong hệ thống phân tán.

## Ứng dụng Kafka vào đâu?

Sau khi nắm rõ cơ chế vận hành của Kafka, Minh xin gợi ý việc áp dụng Kafka ở vài vấn đề cụ thể sau:

* **Hệ thống IoT**: như mọi người đã biết, Internet of Things cho phép các thiết bị gửi các dữ liệu thu thập được (qua sensor chẳng hạn) đến hệ thống máy chủ. Ta có thể dùng Kafka để đón nhận các dữ liệu này, thay vì phải tự phát triển API. Vì dữ liệu thu thập bởi thiết bị có thể nói chúng là "Big data", tần suất gửi đi là rất lớn, yêu cầu xử lý và đáp ứng trong real-time.
* **Stream processing**: đây là một bài toán yêu cầu phân tích luồng dữ liệu (stream) với tốc độ cao. Một bài toán ví dụ đó là ta muốn phân tích hành vi xem sản phẩm của người dùng trên web. Ta sẽ cần ghi nhận lại các dữ liệu thô đơn giản như cú click chuột, đơn hàng. Số lượng người xem web lớn do đó lượng message gửi đến sẽ cực kỳ nhiều và đòi hỏi phân tích tốc độ cao ở real-time.
* **Chuyển đổi ngôn ngữ lập trình**: ta muốn chuyển ngôn ngữ lập trình cấp thấp hơn sang cấp cao hơn, ví dụ từ C sang Python. Các giải pháp về share memory để có thể tích hợp 2 ngôn ngữ lập trình thì rất khó khăn để hiện thực. Do đó, một giải pháp đơn giản thú vị hơn đó là đẩy kết quả dữ liệu của chương trình viết bằng C vào Kafka, sau đó ta dùng chương trình Python để lắng nghe "topic" cụ thể lấy ra dữ liệu (message) tương ứng để phân tích hoặc xử lý tiếp. Đây là một cách khá khả quan, chính nhờ sau khi tìm hiểu Kafka mình phần nào giải thích được làm thế nào mà một số hệ thống lại build được dùng nhiều ngôn ngữ lập trình khác nhau :D.
* **Phân tích log, sự kiện**: in log trong lúc hệ thống đang chạy khá quan trọng cho việc bảo trì hệ thống. Khi ta có nhu cầu về phân tích các log này, ta có thể đây dữ liệu log vào Kafka.

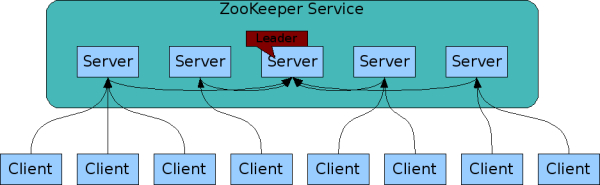
**Zookeeper là gì**

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**

[](https://stackjava.com/wp-content/uploads/2018/05/zkservice.jpg)

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.

## Thành phần trong Apache Kafka

#### **1. Broker**

Một host có thể chạy nhiều server kafka, mỗi server như vậy gọi là một broker. Các broker này cùng trỏ tới chung 1 zookeeper thì chúng là 1 cụm broker(Clusters). Broker là nơi chứa các partition. Một broker có thể chứa nhiều partition.

#### **2. Topic**

Nhìn về mặt database thông thường thì topic giống như một table trong CSDL SQL. Topic là nơi chứa message. Mỗi message giống như record trong table (Chỉ khác là chúng NoSql).

#### **3. Partition**

Nơi lưu trữ message của topic. Một topic có thể có nhiều partition. Khi khởi tạo topic cần set số partition cho topic đó. Partition càng nhiều thì khả năng làm việc song song cho đọc và ghi được thực hiện nhanh hơn. Các message trong partition được lưu theo thứ tự bất biến(offset). Một partition sẽ có tối thiểu 1 replica để đề phòng trường hợp bị lỗi. Số lượng replica luôn nhỏ hơn số lượng broker.

#### **4. Producer**

Là nơi đẩy dữ liệu từ người dùng vào các partition của topic. Tùy vào việc có chỉ định rõ ghi vào phân vùng nào không thì producer sẽ gửi đến phân vùng của topic đó hoặc phân bố đều vào các partition.

#### **5. Consumer**

Mỗi consumer phụ trách 1 vài partition của topic. Nhiều consumer có thể sử dụng chung 1 nhóm (group consumer) để hỗ trợ cho việc xử lý nhanh các dữ liệu. Có nhiều group consumer xử lý cùng 1 topic theo các điểm start offset khác nhau. Điều này có thể phục vụ cho việc bạn có 2 process riêng biệt, một phục vụ cho việc duy trì hệ thống với dữ liệu mới đẩy vào, một phục vụ cho việc re-process lại toàn bộ data.

1. **MESSAGES TRONG KAFKA LƯU Ở ĐÂU?**

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.

## 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.

**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.

# RabbitMQ

RabbitMQ là một message broker ( message-oriented middleware) hỗ trợ các protocols như MQTT, AMQP (Advanced Message Queue Protocol - giao thức phổ biến được sử dụng), và STOMP.

Flow cơ bản: Producer -> Exchange -> Binding -> Queues -> Consumer.

Producer đẩy message vào exchange. Sau khi exchange nhận message, exchange (4 loại: direct, topic, fanout, headers ) là một lớp middleware nên nó giúp điều hướng binding message (dựa trên thuộc tính của message - routing key) đến hàng đợi (queue). Các message tại queue sẽ được xử lý bới consumer, một khi được xử lý xong sẽ đánh dấu status của consumer và message được xóa khỏi queue.

## Như vậy, mình sẽ tóm lại như thế này từ bảng so sánh trên:

* Nếu hệ thống của bạn không cần lưu trữ message, ưu tiên cho việc đòi hỏi sự đảm bảo rằng các consumer đều nhận được message và duy nhất bên cạnh độ ưu tiên của từng message thì dạng messaging truyền thống như RabbitMQ sẽ thực sự hữu dụng.
* Nếu hệ thống của bạn đòi hỏi về mặt lưu trữ và tốc độ truyền tải message. Consumer muốn lựa chọn số lượng message mình cần, có thể lấy theo thứ tự hoặc muốn lấy từ lúc bắt đầu, một consumer có thể nhận đi nhận lại nhiều lần message đấy. Lúc này đây, hệ thông messaging dạng pipeline như Kafka sẽ được tin dùng. Có thể kể đến một số user cases như: Stream Processing, Event sourcing,..

## Kafka và RabbitMQ

## Các thành phần và cơ chế hoạt động

Kafka

1. Consumer / Consumer groups
2. Producer
3. Kafka source connect
4. Kafka sink connect
5. Topic and topic partition
6. Kafka stream
7. Broker
8. Zookeeper

Kafka là một hệ thống mạnh mẽ bao gồm nhiều thành phần nhưng khi đã hiểu luồng hoạt động, bạn rất dễ để quản lý và làm việc với nó.

Consumer gửi một bản ghi thông điệp đến một **topic** (topic là một chủ đề bao gồm các bản ghi được phát hành (published)), nó có thể được phân vùng để có hiệu năng tốt nhất. Các consumer theo dõi (subscribe) một topic và bắt đầu nhận thông điệp từ nó. Khi một topic được phân vùng thì mỗi vùng sẽ có một thể hiện consumer riêng. Các thể hiện trên cùng một consumer được gọi là một consumer group.

Trong Kafka, các thông điệp luôn được lưu trữ trong topic mặc dù chúng đã được thu nhận với thời gian giới hạn theo chính sách định nghĩa trong hệ thống.

Ngoài ra, Kafka sử dụng truy cập tuần tự giúp tăng hiệu suất của Kafka, khiến Kafka là lựa chọn hàng đầu trong việc thực thi hàng đợi và cũng là một lựa chọn an toàn trong trường hợp phải xử lý với dữ liệu lớn.

## RabbitMQ

1. Consumer
2. Publisher
3. Exchange
4. Route

Luồng xử lý bắt đầu từ Publisher, thông điệp được gửi đến Exchange. Exchange là một lớp middleware giúp điều hướng thông điệp đến hàng đợi (queue). RabbitMQ đẩy một thông điệp đến consumer, và một khi đã được consume và thông điệp đã đến nơi, thông điệp sẽ bị xóa khỏi hàng đợi. Mọi thành phần trong hệ thống để có thể mở rộng: Producer, Consumer, và bản thân RabbitMQ có thể cluster và có tính sẵn sàng cao.

So sánh

## 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 manghj 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.

## 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)

## Theo dõi

Với Kafka, có 3 công cụ theo dõi

**Trả phí**

* Confluent <https://www.confluent.io/product/control-center/>
* Landoop <http://www.landoop.com/>

**Miễn phí**

* Burrow <https://github.com/linkedin/Burrow>
* Kafka Tool <http://www.kafkatool.com/>

RabbitMQ cung cấp sẵn công cụ quản lý thông qua giao diện web

## Hỗ trợ giao dịch

Cả hai hỗ trợ atomic write, có nghĩa là khi ghi một loạt thông điệp vào hàng đợi và một trong số chúng thất bại, tất cả các giao dịch sẽ được hủy bỏ. Việc này được sử dụng thường xuyên khi xử lý stream trong Kafka.