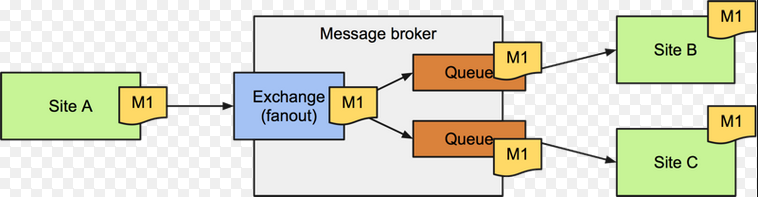
### Messege Queue

* 1. **Khái niệm MQ**
* Sở dĩ gọi nó là queue (hàng đợi) vì nó thực hiện việc lấy message theo cơ chế FIFO – First In First Out, tức đút vào trước thì rút ra trước.
* Một hệ thống sử dụng Message Queue thường có những thành phần sau đây:
  + **Message**: Thông tin được gửi đi (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**: Chương trình/service tạo ra thông tin, đưa thông tin vào message queue
  + **Consumer**: Chương trình/service nhận message từ message queue và xử lý
  + Một chương trình/service có thể **vừa là producer, vừa là consumer**
  1. **Message Queue được sử dụng ra sao trong thực tế?**
* Trong các hệ thống [dùng kiến trúc microservice](https://toidicodedao.com/2017/02/21/tong-quan-micro-service/), ta sử dụng message queue để giúp các service liên hệ với nhau **một cách bất đồng bộ**. *Service A* làm xong việc có thể gửi message queue để *service B* biết mà xử lý, **không cần phải chờ** service B làm xong.
* Giả sử, mình có một trang web cho phép người dùng tải link Youtube, mình sẽ có các bộ phận sau:
  + **Web service**: Là 1 producer. Nhận thông tin (url Youtube) từ phía người dùng, đưa thông tin này vào message queue
  + **Processing Service**: Vừa là consumer vừa là producer. Service này đọc url Youtube từ message queue, bắt đầu tải file 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 các video đó lên Google Drive v…v
  1. **Message queue giải quyết được những vấn đề gì?**
* Trong thực tế, message queue giải quyết được khá nhiều vấn đề hóc búa trong hệ thống:
  + **Đảm bảo duration/recovery**: Do message đã được lưu trong queue, khi 1 service đang xử lý nhưng bị crash hoặc lỗi, ta không lo bị mất dữ liệu; vì có thể lấy message từ trong queue ra và chạy lại. Trong 1 hệ thống có nhiều consumer, nếu 1, 2 consume bị crash cũng không làm sụp toàn hệ thống
  + **Phân tách hệ thống**: Giúp phân tách hệ thống thành nhiều service nhỏ hơn, mỗi service chỉ xử lý 1 chức năng nhất định (Ưu nhược điểm thì các bạn xem lại [bài về microservice](https://toidicodedao.com/2017/02/21/tong-quan-micro-service/) nhé)
  + **Hộ trợ rate limit, batching**: Trong nhiều trường hợp, năng lực xử lý hệ thống có hạn (chỉ có thể xử lý 300 đơn hàng/s). Với message queue, ta có thể dần dần lấy đơn hàng trong queue ra xử lý, không sợ thất lại. Hoặc thay vì mỗi lần gửi email mất thời gian lâu, ta có thể đợi message queue có yêu cầu gửi 200 email rồi gửi luôn 1 lượt.
  + **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.
  1. **Một số hạn chế**
* **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.
* **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 cũng sẽ khó khăn hơn
* **Cần đảm bảo message format**: Để gửi/nhận, 2 phía producer và consumer phải thống nhất format với nhau. Nếu không cẩn thận lỡ 1 bên thay đổi sẽ làm bên kia không đọc được dữ liệu.
* **Cần Monitoring Queue**: Cần 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 (message gửi vào queue đều bị consume hết)

1. **RabbitMQ**
   1. **Khái niệm**

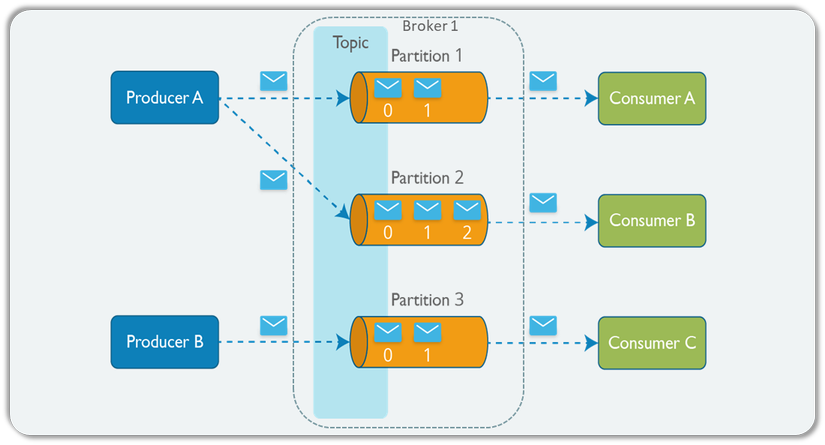
* RabbitMQ là một message broker ( message-oriented middleware) sử dụng giao thức AMQP - Advanced Message Queue Protocol (Đây là giao thức phổ biến, thực tế rabbitmq hỗ trợ nhiều giao thức). RabbitMQ được lập trình bằng ngôn ngữ Erlang. RabbitMQ cung cấp cho lập trình viên một phương tiện trung gian để giao tiếp giữa nhiều thành phần trong một hệ thống lớn ( Ví dụ openstack - Một công nghệ rất thú vị hi vọng một ngày nào đó tôi đủ sức để viết vài bài về chủ đề này ). RabbitMQ sẽ nhận message đến từ các thành phần khác nhau trong hệ thống, lưu trữ chúng an toàn trước khi đẩy đến đích.
  1. **Một số đặc tính**
* Asynchronous: Đặc tính này có thể được tận dụng để xây dựng các hệ thống lưu trữ và xử lý log.
* cluster: các bạn có thể gom nhiều rabbitmq instance vào một cluster. Một queue được đinh nghĩa trên một instance khi đó đều có thể truy xuất từ các instance còn lại. Có thể tận dụng để làm load balancing (tuy rằng có hạn chế, tôi sẽ nói sau)
* high availibilty: cho phép failover khi sử dụng mirror queue.
* reliability: có cơ chế ack để đảm bảo message được nhận bởi consumer đã được xử lý.
  1. **Mô hình**

[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/Screenshot%20from%202015-05-24%2022:07:35.png_syxcagcxq7)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/Screenshot%20from%202015-05-24%2022%3A07%3A35.png_syxcagcxq7)

* Có thể hiểu message broker gần như bưu điện. Site A theo cách gọi của rabbitmq là producer (người gửi thông điệp). Site B và Site C theo cách gọi của rabbitmq là consumer (người nhận thông điệp). Producer connect đến message broker để đẩy message. Message sẽ đi qua message broker để đến được consumer. Cấu trúc của message broker chỉ gồm hai phần exchange và queue.
* Exchange có nhiều loại. Trong hình vẽ trên exchange type là fanout. Lựa chọn các exchange type khác nhau sẽ dẫn đến khác đối xử khác nhau của message broker với thông điệp nhận được từ producer. Exchange được bind (liên kêt) đến một số queue nhất định. Với exchange type là fanout, message sẽ được broadcast đến các queue được bind với exchange. Consumer sẽ connect đến message broker để lấy message từ các queue.

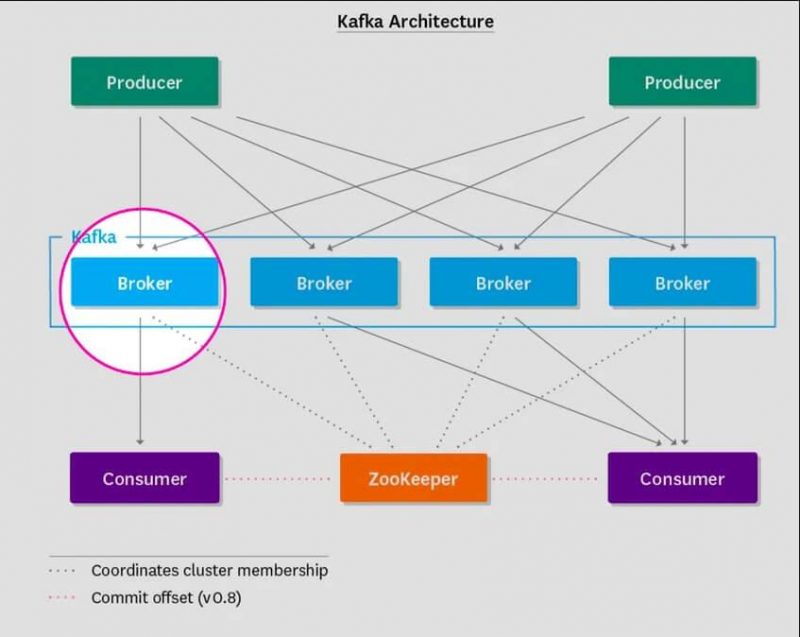
1. **Kafka**
   1. **Khái niệm**

* **Kafka** 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** có thể hiểu là một hệ thống **logging**để lưu lại các **trạng thái** của hệ thống đề phòng tránh mất thông tin.
* 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.



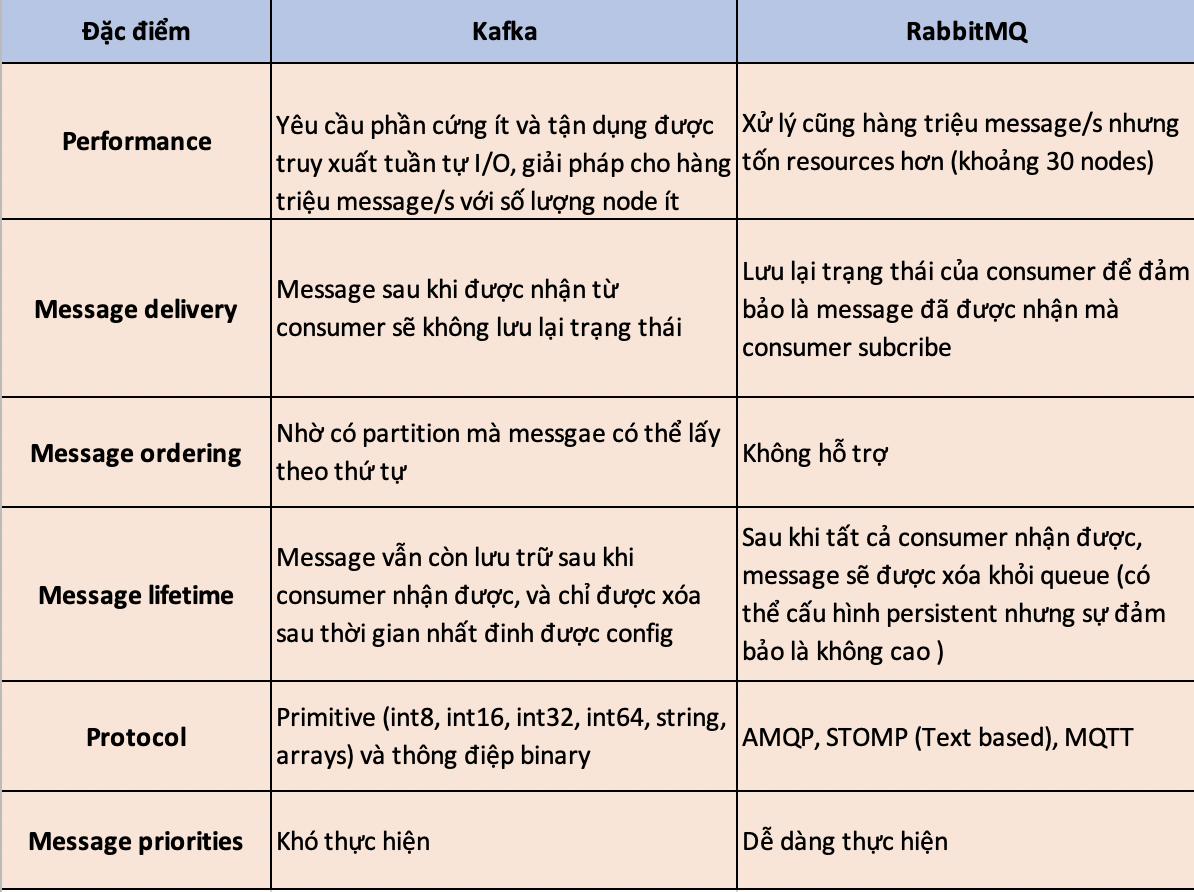
## **Kafka hoạt động như thế nào?**

* 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.
  1. **Các câu hỏi thường gặp về Apache Kafka**
* **Apache Kafka là gì?** Kafka Kafka là nền tảng streaming phân tán, có thể mở rộng và là sản phẩm mã nguồn mở.
* **Apache Kafka có khả năng gì?**Nó có thể thực hiện 3 khả năng chính như sau:
  + Cơ chế publish/subcribe tương tự như các hệ thống message queue doanh nghiệp khác.
  + Cơ chế lưu trữ stream các bản ghi trên nhiều node broker khác nhau, giúp khắc phục lỗi xảy ra.
  + Xử lý stream các bản ghi ngay khi nó vừa đến.
* **Các thành phần chính trong Kafka là gì?** Nó có 4 thành phần khác nhau như sau:
  + Kakfa Broker
  + Zookeeper
  + Producer
  + Consumer
* Mục đích cơ bản sử dụng Kafka?
  + Sử dụng để theo dõi lưu lượng truy cập, giám sát web server.
  + Xử lý sự kiện

1. **So sánh và phân biệt**

Sự khác nhau giữa kafka & rabbitMQ

Như vậy, có thể tóm lại như thế này từ bảng so sánh trên:

* Nếu hệ thống 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 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,..