# Message Broker (Message Oriented Middleware)

* Khi làm việc với kiến trúc cloud, các ứng dụng được chia nhỏ thành nhiều components cô lập nhau để dễ dàng bảo trì và chỉnh sửa. Và việc giao tiếp này có thể yêu cầu nhiều request gửi qua lại, và trở nên vô cùng khó khăn. Lúc này chúng ta nghĩ ra một bài toán mới, là có thể giao tiếp với các hệ thống một cách an toàn và dễ quản lý, mọi request được xử lý mà không mất cái nào. Message Broker được đưa ra.
* Message Broker là một middleware được thiết kế với mục đích để chuyển các gói tin (message) từ producer (người gửi) đến consumer (người nhận). Các gói tin ở đây là chuỗi ký tự byte được lưu vào vào trong hàng đợi (Queue).

# RabbitMQ

* **RabbitMQ** là một message broker (Message Oriented Middleware) sử dụng giao thức Advanced Message Queue Protocol (AMQP) (có nhiều loại giao thức AMQP). RabbitMQ được viết bằng Erlang, được sử dụng chủ yếu cho các công việc của Message Broker.
* RabbitMQ sử dụng giao thức AMQP 0-9-1
* Mô hình AMQP 0-9-1 được mô tả qua quá trình: Messages mà Pulishers gửi được đưa tới *exchanges* (giống như một hòm thư), exchanges sau đó phân bố các bản sao của message tới Queues theo một quy tắc đặt tên là *bindings*. Sau đó, message broker gửi messages tới các Consumers đăng ký Queues (push API) hoặc Consumers tự nhận messages về qua Queue (pull API). Ngoài ra, trước khi gửi messages, Publishers có thể thêm các metadata về messages, metadata này có thể được broker sử dụng để điều phối phân phát messages tới consumers thích hợp.
* Đặc biệt hơn, AMQP có thể xử lí tình trạng message không thể gửi tới Consumers do vấn đề về mạng, phương thức mà AMQP sử dụng là *message acknowledgements* **(ack)**, cụ thể là khi một message được nhận bởi/gửi tới một Consumer thì một notification sẽ được truyền tới message broker nhằm thông báo cho broker về việc gửi nhận thành công message, và chỉ trong tình huống này bản sao của message có thể được loại bỏ khỏi Queue. Trong tình huống messages không thể được phân phát tới Queue thì messages đó có thể được gửi trả lại Pulishers, hoặc bị loại bỏ, hay broker đưa messages vào queue đặc biệt gọi là Dead Letter Queue. RabbitMQ hỗ trợ Pulishers trong tình huống nêu trên.

## Bindings

## Bindings là các quy tắc mà khối Exchanges sử dụng để chuyển messages tới Queue. Bindings có thể bao gồm một *routing key* với mục đích để lựa chọn các messages tới một Queue.

## Exchanges

* Exchanges là một entity của AMQP 0-9-1, là nơi mà messages được gửi tới trước khi phân phát tới Queues thông qua quy tắc bindings và phụ thuộc vào *exchange types*. Hiện tại, AMQP 0-9-1 cung cấp 5 exchange types: Default exchange, Direct exchange, Fanout exchange, Topic exchange, Headers exchange.
* AMQP 0-9-1 còn định nghĩa exchanges thông qua các thuộc tính như: Name, Durability, Auto-delete, Arguments.

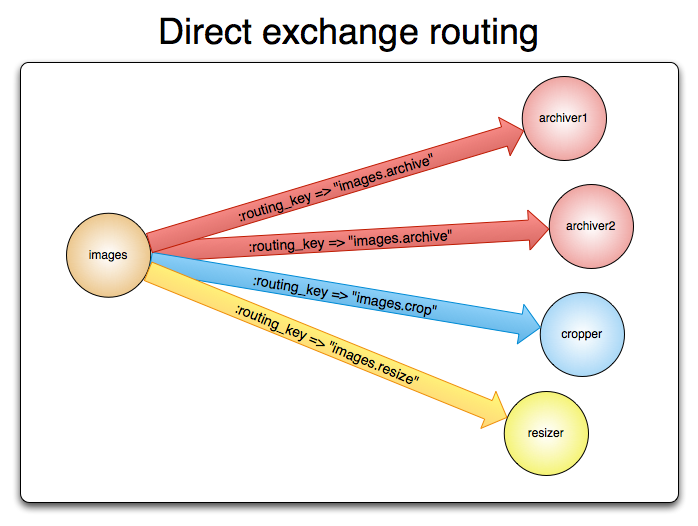
A screenshot of a cell phone

Description automatically generated

### Default exchange

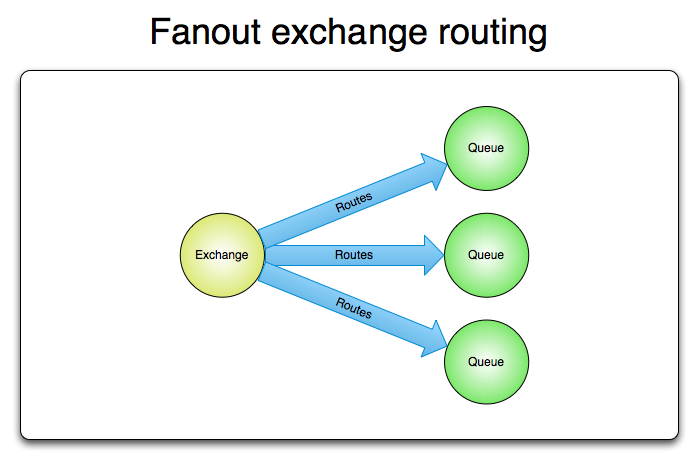
* Default exchange là một Direct Exchange không có thuộc tính Name định nghĩa sẵn bởi message broker (theo cách hiểu khác là Name sẽ được định nghĩa bởi người dùng). Và routing key sẽ được lấy đúng bằng Name do người dùng đặt sẵn, từ đó các messages với routing key trùng với Name của Default exchange sẽ được chuyển tới Queue mang tên đó.

### Direct exchange



* Direct exchange thực hiện theo các bước:
* Một Queue được bind với Exchange thông qua routing key K
* Khi một message với routing key R được đưa tới Direct exchange thì message đó được chuyển tới Queue K nếu K == R.

### Fanout exchange



* Fanout exchange thực hiện chuyển messages tới tất cả các Queues được bind tới exchange đó và không sử dụng routing key để phân phát messages. Nếu N Queues được bind tới một Fanout exchange, thì khi một message được đưa tới exchange sẽ được tạo N bản sao và gửi tới N Queues đó.

### Topic exchange

## Topic exchange thực hiện chuyển messages tới một hoặc nhiều Queues dựa trên điểm tương đồng giữa routing key của message và bindings của queue tới exchange. Ví dụ như phân phát tin tức tới các tiêu đề cụ thể (tin tức chính trị, thể thao, giải trí, ...)

### Headers exchange

* Headers exchange thực hiện chuyển messages dựa trên các thuộc tính chứa nhiều thông tin và dễ mô tả thay vì sử dụng một routing key. Các thuộc tính đó gọi là headers attribute. Một message được coi là khớp nếu giá trị của header tương đồng với giá trị định nghĩa trong binding.
* Một Queue có thể được bind tới Headers exchange với nhiều hơn một header nhằm phục vụ cho việc matching. Trong tình huống này, message broker có 2 lựa chọn: message được coi là khớp nếu chỉ một trong tất cả header của message tương đồng với một giá trị trong bindings, hoặc message được coi là khớp nếu như tất cả các header của message tương đồng với tất cả giá trị định nghĩa trong bindings.

*<Ghi chú tới team: Như thanh search trên thuvienphapluat.vn, coi nội dung search là header của message, các luật là giá trị bindings>*

## Queue

* Queue bao gồm các thuộc tính:
  + Name
  + Durable: Durable Queue không ảnh hưởng bởi việc broker restart
  + Exclusive (used by only one connection and the queue will be deleted when that connection closes)
  + Auto-delete (queue that has had at least one consumer is deleted when last consumer unsubscribes)
  + Arguments (optional; used by plugins and broker-specific features such as message TTL, queue length limit, etc)
  + AMQP 0-9-1 quy định Queue cần được định nghĩa trước khi sử dụng, nếu Queue chưa tồn tại thì việc định nghĩa sẽ khởi tạo một Queue mới và nếu ngược lại thì Queue tồn tại sẽ không bị loại bỏ mà giữ nguyên giá trị các thuộc tính vốn có. Và trường hợp Queue tồn tại có thuộc tính không như những thuộc tính đã khai báo từ trước sẽ gặp lỗi 406 (PRECONDITION\_FAILED)

### Queue Names

* + Các ứng dụng có thể tự đặt tên hoặc yêu cầu broker sinh tên.