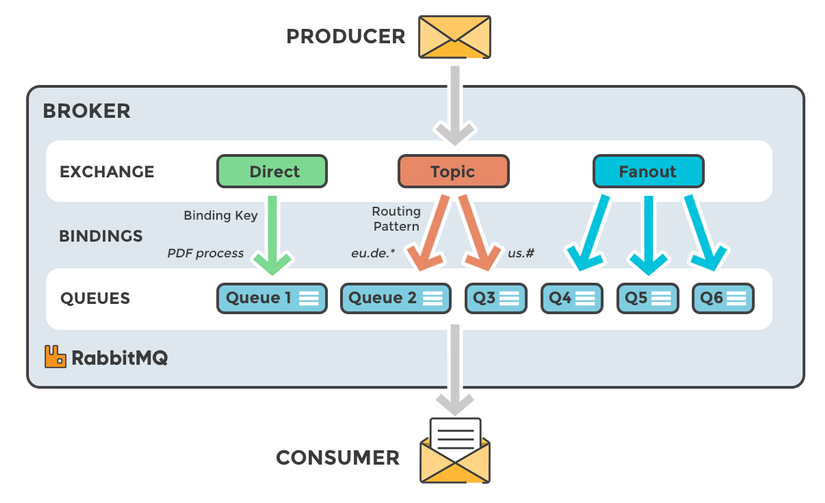
**1. Khái niệm**

      RabbitMQ là một message-queuing software có thể được biết đến như là một người vận chuyển message trung gian hoặc một người quản lí các queue. Nói một cách đơn giản, nó là một phần mềm nơi các queue được định nghĩa, phục vụ cho ứng dụng với mục đích vận chuyển một hoặc nhiều message.

      Vậy, tại sao chúng ta phải cần đến RabbitMQ? Hãy tưởng tượng, hiện bạn đang có một web service, phải nhận rất rất nhiều request mỗi giây, mà lại phải đảm bảo rằng không có bất cứ một request nào bị mất. Và web service của bạn luôn luôn sẵn sàng tiếp nhận request mới thay vì locked bởi đang xử lí request trước đó. Vậy ý tưởng ở đây là đặt chúng vào một queue giữa web service và processing service. Lúc này sẽ đảm bảo rằng 2 process sẽ hoàn toàn tách rời nhau. Ngoài ra, queue sẽ lưu trữ những request, không bị thiếu sót request nào khi số lượng của chúng trở nên vô cùng lớn.

**1.1 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.

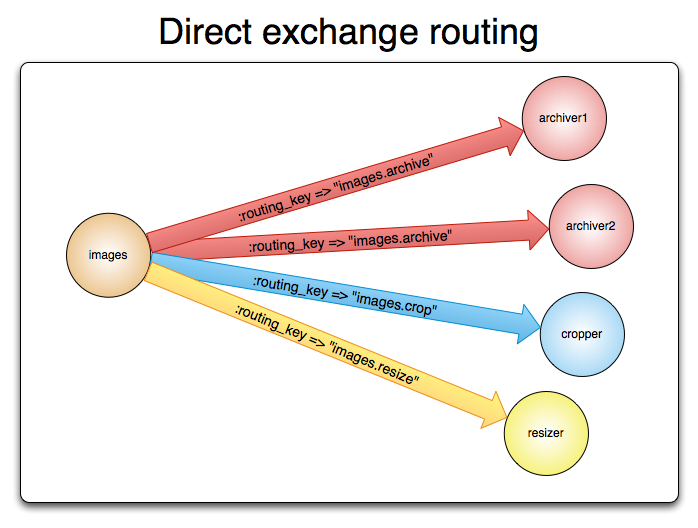


*Hình 1. Sơ đồ vận chuyển message trong RabbitMQ*

**1.2. 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. 

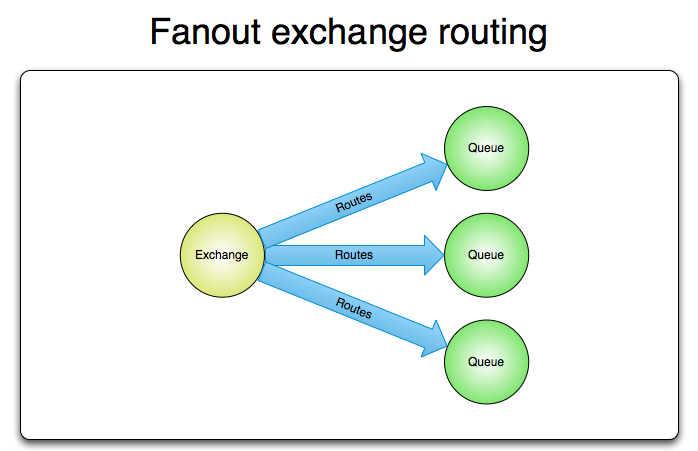
*Hình 2. Các bước vận chuyển message trong Direct exchange*

**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.

**2. Xây dựng ứng dụng với Spring Boot và RabbitMQ**

**2.1. Tạo một RabbitMQ instance**

      Nếu bạn muốn tự cài đặt và cấu hình RabbitMQ thì bạn phải cài đặt [Erlang](https://www.erlang.org/downloads) làm môi trường trước tiên. Sau đó, tải [RabbitMQ](https://www.rabbitmq.com/download.html) và cài đặt thôi. Nếu bạn không muốn mất công tìm kiếm cách cài đặt cũng như cấu hình phức tạp thì có thể vào [AMPQ Cloud](https://www.cloudamqp.com/plans.html), tự tạo cho mình một RabbitMQ instance chỉ qua vài bước đơn giản.

**2.2. Tạo ứng dụng Spring Boot và RabbitMQ**

      - Tạo một ứng dụng Spring Boot với những dependencies sau:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

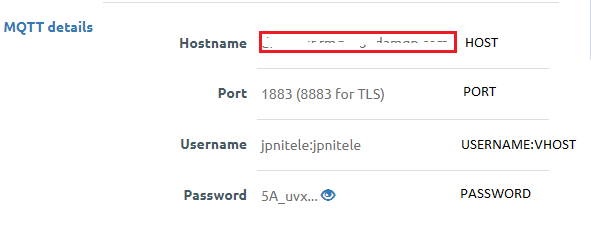
<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

      - Cấu hình RabbitMQ trong file application.properties:



spring.rabbitmq.host=${RABBITMQ\_HOST}

spring.rabbitmq.port=${RABBITMQ\_PORT}

spring.rabbitmq.username=${RABBITMQ\_USERNAME}

spring.rabbitmq.password=${RABBITMQ\_PASSWORD}

spring.rabbitmq.virtual-host=${RABBITMQ\_VHOST}

      - Tạo file RabbitMQConfig.java để cấu hình routing và listener:

@Bean

public Declarables directBingdings() {

Queue directQueue1 = new Queue(TOPIC\_QUEUE\_1\_NAME, false);

Queue directQueue2 = new Queue(TOPIC\_QUEUE\_2\_NAME, false);

DirectExchange directExchange = new DirectExchange(DIRECT\_EXCHANGE\_NAME);

return new Declarables(

directExchange,

bind(directQueue1).to(directExchange).with("direct.exchange-1"),

bind(directQueue2).to(directExchange).with("direct.exchange-2")

);

}

@RabbitListener(queues = {TOPIC\_QUEUE\_1\_NAME})

public void receiveMessageTopic1(String message) {

System.out.println(String.format("[%s] [%s] Received message: %s", TOPIC\_EXCHANGE\_NAME, TOPIC\_QUEUE\_1\_NAME, message));

}

@RabbitListener(queues = {TOPIC\_QUEUE\_2\_NAME})

public void receiveMessageTopic2(String message) {

System.out.println(String.format("[%s] [%s] Received message: %s", TOPIC\_EXCHANGE\_NAME, TOPIC\_QUEUE\_2\_NAME, message));

}

      Sử dụng rabbitTemplate để thực hiện việc gửi message:

@RestController

@RequestMapping("/api/v1")

public class MessageController {

@Autowired

private RabbitTemplate rabbitTemplate;

@PostMapping("/message")

public void sendMessage(@RequestBody final Message message) {

rabbitTemplate.convertAndSend(RabbitMQConfig.TOPIC\_EXCHANGE\_NAME, "msg.important.warn",

"topic important warn: " + message.getMessage());

rabbitTemplate.convertAndSend(RabbitMQConfig.TOPIC\_EXCHANGE\_NAME, "msg.error",

"topic important error: " + message.getMessage());

}

}