

## BÁO CÁO BÀI TẬP LỚN SỐ 2:

Nhóm: Trần Thu Phương - 17021321

Nguyễn Tuấn Quốc - 17021326

Nguyễn Minh Quân – 17021325

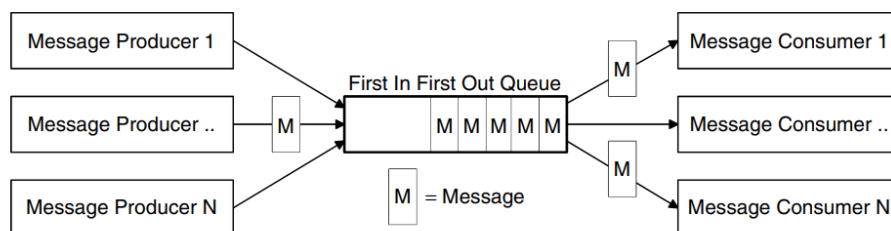
### Đề tài: Tìm hiểu về Message Oriented Middleware (MOM) và RabbitMQ?

#### I. MESSAGE ORIENTED MIDDLEWARE (MOM)

##### 1. Giới thiệu Message Oriented Middleware

Một phương thức dựa trên cơ chế gửi thông điệp không đồng bộ - asynchronous message, tức là máy khách gửi yêu cầu tới máy chủ mà không cần chờ kết quả phản hồi từ máy chủ. Trong phương thức này, các ứng dụng không tương tác trực tiếp với nhau, mà chúng tương tác gián tiếp thông qua hàng đợi. Một hàng đợi là tập hợp các thông điệp có thể được chia sẻ với nhiều máy tính. Những đoạn mã được xây dựng để kết nối được gọi là Message-oriented middleware (MOM).

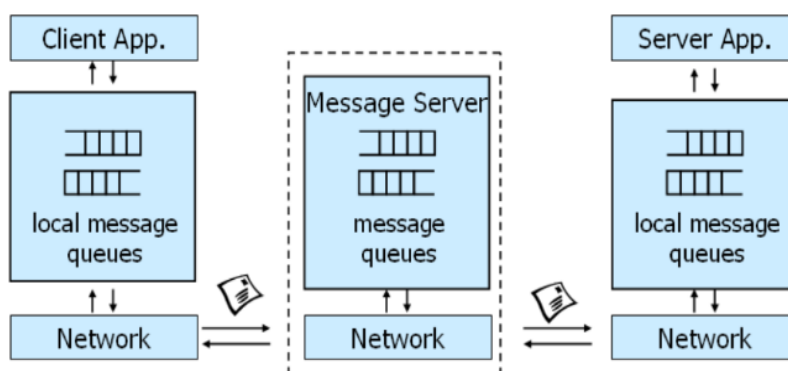
Hàng đợi là nơi để thực hiện mô hình tương tác không đồng bộ trong MOM. Một hàng đợi là một tập hợp các thông điệp có thể được gửi đến và nhận, thông thường các thông điệp trong hàng đợi được sắp xếp theo thứ tự cụ thể. Hàng đợi tiêu chuẩn được tìm thấy trong một hệ thống gửi thông điệp là First-In First-Out (FIFO); như tên cho thấy, thông điệp đầu tiên được gửi đến hàng đợi là thông điệp đầu tiên được lấy từ hàng đợi.



MOM có 2 loại là Message queueing và Message passing. Message queueing là mô hình truyền thông gián tiếp thông qua hàng đợi. Message passing là mô hình truyền thông trực tiếp giữa hai tiến trình. Tuy nhiên, hiện nay các MOM đều hỗ trợ cả 2 mô hình truyền thông này, do vậy các thuật ngữ MOM, hay Message Queueing System, Messaging – Queueing System có thể hiểu chung một nghĩa. Để cho ngắn gọn, ta sử dụng cụm từ hệ thống messaging để chỉ hệ thống này, và công nghệ messaging để chỉ công nghệ phát triển ứng dụng dựa trên các hệ thống Messaging.

Các hệ thống MOM cung cấp các truyền thông phân tán trên cơ sở mô hình tương tác không đồng bộ. Mô hình này cho phép MOM giải quyết nhiều hạn chế trong RPC. Người tham gia trong hệ thống dựa trên MOM không bắt buộc phải chặn và chờ đợi thông điệp gửi, họ được phép tiếp tục xử lý một khi thông điệp đã được gửi. Điều này cho phép gửi thông điệp khi người gửi hoặc người nhận không hoạt động hoặc có sẵn để trả lời tại thời điểm thực hiện.

MOM trái ngược với các cơ chế như RPC hay RMI. MOM hỗ trợ các thông điệp có thể mất vài phút để truyền đi còn RPC hay RMI thì chỉ mất một phần nghìn giây hoặc một giây để gửi.



*Mô hình truyền thông trong MOM*

Truyền thông trong MOM được thực hiện bởi việc sử dụng các thông điệp, các thông điệp được gửi và nhận theo cơ chế đồng bộ thông qua hàng chờ/queue. Các thông điệp này phải được gửi đến đích mong muốn nếu chưa đến đích thì MOM sẽ thực hiện gửi lại ngay. MOM có cơ chế điều phối thông điệp do đó giảm thiểu vấn đề quá tải server. Một số công nghệ phổ biến được xây dựng trên kiến trúc MOM là: IBM MQ, Java Messaging Service, MS MQ,...

## 2. Một số tính chất của MOM

- ***Coupling***

MOM cung cấp một layer giữa người gửi và người nhận, cho phép người gửi và người nhận message sử dụng lớp độc lập này làm trung gian để trao đổi thông điệp. Lợi ích chính của MOM là sự kết hợp lỏng lẻo giữa những người tham gia trong một hệ thống - khả năng liên kết các ứng dụng mà không phải điều chỉnh hệ thống nguồn và hệ thống đích với nhau, dẫn đến việc triển khai hệ thống được tách rời, gắn kết cao.

- ***Reliability***

Việc xảy ra lỗi mất thông điệp qua network hoặc hệ thống được MOM ngăn chặn bằng cách sử dụng cơ chế lưu trữ và chuyển tiếp để duy trì thông báo. Khả năng này của MOM cho thấy mức độ tin cậy cao vào cơ chế này khi các bộ phận của hệ thống không khả dụng hoặc bận. Mức độ tin cậy cụ thể thường có thể cấu hình, nhưng các hệ thống gửi thông điệp MOM có thể đảm bảo rằng một thông điệp sẽ được gửi và nó sẽ được gửi đến từng người nhận một lần theo dự định.

- ***Scalability***

Các hệ thống con của MOM có thể được tách rời một cách độc lập, ít hoặc không có ảnh hưởng đến sự gián đoạn đối với các hệ thống con khác. Các mô hình gửi message MOM chứa một số đặc điểm tự nhiên, cho phép cân bằng tải đơn giản và hiệu quả, bằng cách cho phép một hệ thống con chọn chấp nhận thông điệp khi nó sẵn sàng thực hiện thay vì nó buộc phải chấp nhận. Các nền tảng MOM cấp doanh nghiệp hiện đại đã được sử dụng làm xương sống để tạo ra các hệ thống có khả năng mở rộng quy mô với sự hỗ

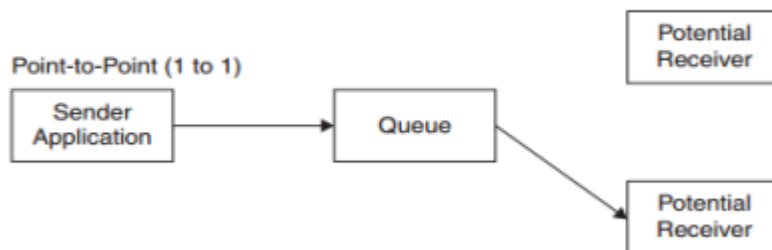
trợ xử lý 16,2 triệu truy vấn đồng thời mỗi giờ và hơn 270.000 yêu cầu đặt hàng mới mỗi giờ.

- **Availability**

MOM luôn sẵn sàng cho phép các hệ thống vận hành liên tục và xử lý các sự cố khiến hệ thống không được mượt. Sự thất bại của một trong các hệ thống con sẽ không gây ra lỗi trong toàn bộ hệ thống. MOM cũng có thể cải thiện thời gian phản hồi của hệ thống do việc “loose coupling” giữa các thành phần tham gia MOM. Điều này có thể giúp giảm thời gian hoàn thành tiến trình và cải thiện khả năng đáp ứng và tính sẵn sàng của toàn bộ hệ thống.

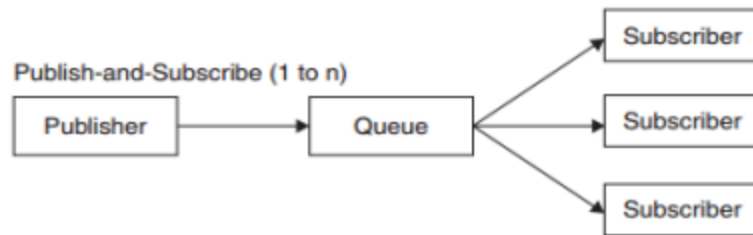
### 3. Các thành phần của MOM

- Hàng đợi/Kênh (Queues/Channels): sử dụng để truyền dữ liệu. Có hai loại hàng đợi:
  - *Point-to-point*: chỉ một điểm nhận nhận mỗi thông điệp



*Hàng đợi Point-to-point*

- *Push and Subscribe*: thông điệp được gửi tới tất cả các subscriber. Mỗi subscriber như một bản sao lưu của thông điệp và pushlisher không quan tâm tới ai lắng nghe



*Hàng đợi Push and Subscribe*

- Thông điệp (Message): đóng gói dữ liệu (function) cần trao đổi giữa client và server. Thông điệp bao gồm:
  - *Header*: định nghĩa thông điệp và các thông tin điều khiển.
  - *Body*: chứa thông tin sẽ được xử lý khi ứng dụng nhận thông điệp.
- Điểm kết thúc (EndPoints): điểm cho phép client/server kết nối được với MOM và để gửi hay nhận một thông điệp.

#### 4. Ưu điểm, nhược điểm của MOM

- **Ưu điểm:**
  - Nâng cao tính mở rộng của hệ thống tích hợp.
  - Tương tác bất đồng bộ: client/server không bị ràng buộc chặt chẽ, các thông điệp được đặt vào hàng đợi. Các tính chất này rất tốt cho việc tích hợp ứng dụng.
  - Hỗ trợ dịch vụ truyền tin cậy, MOM giữ các hàng đợi trong các kho chứa.
  - Xử lý các thông điệp bởi các server thông điệp trung gian. Có các cơ chế như lọc, chuyển đổi, lưu vết,...
- **Nhược điểm:**
  - Sử dụng nhiều MOM cùng một lúc có thể dẫn tới sự không đồng nhất như là giao thức không đồng nhất (HTTP hoặc HTTPS); định dạng thông điệp không đồng nhất.
  - Có những ứng dụng cần cả cơ chế gọi hàm đồng bộ và không đồng bộ.

## 5. Một vài MOM services phổ biến

### 5.1. Message Filtering

Message Filtering cho phép người nhận tin được chọn lọc các thông điệp mà họ nhận được từ một kênh. Filtering có thể hoạt động trên một số cấp độ khác nhau. Các bộ filters sử dụng các biểu thức logic Boolean để khai báo các thông điệp quan tâm đến Client, định dạng chính xác của biểu thức phụ thuộc vào implementation nhưng các mệnh đề WHERE của SQL-92 (hoặc một tập hợp con) thường được sử dụng làm cú pháp. Các mô hình lọc thường hoạt động trên các thuộc tính (cặp tên / giá trị) của tin nhắn; tuy nhiên, một số projects đã mở rộng tính năng lọc đối với message payloads.

Vì có một số khả năng lọc được tìm thấy trong các hệ thống gửi message, nên rất hữu ích để ghi lại khi mà các kỹ thuật lọc ngày càng tiên tiến, chúng có thể sao chép các kỹ thuật thực hiện chúng. Ví dụ: lọc dựa trên chủ đề có thể sao chép lọc dựa trên kênh, giống như lọc dựa trên nội dung có thể sao chép cả lọc theo chủ đề và lọc dựa trên kênh.

### 5.2. Transactions

Tài nguyên là một kho lưu trữ dữ liệu liên tục đang tham gia vào một transaction sẽ được cập nhật. MOM có khả năng bao gồm một thông điệp được gửi hoặc nhận trong một transaction. Khi nào kiểm tra các khía cạnh giao dịch của các hệ thống gửi thông điệp, điều quan trọng cần nhớ là các message MOM là các thực thể độc lập tự chủ. Trong message tên miền, có hai loại giao dịch phổ biến là Local Transactions và Global Transactions. Các Local Transactions diễn ra trong một trình quản lý tài nguyên như một nhà môi giới gửi message duy nhất. Global Transactions liên quan đến nhiều, có khả năng phân phối người quản lý tài nguyên không đồng nhất với người quản lý giao dịch bên ngoài điều phối các giao dịch.

### 5.3. Guaranteed Message Delivery

Để các nền tảng MOM đảm bảo việc gửi thông điệp, nền tảng phải lưu tất cả các thông điệp trong một cửa hàng không dễ biến đổi như đĩa cứng. Nền tảng sau đó gửi thông điệp đến người tiêu dùng và chờ người tiêu dùng xác nhận việc gửi. Nếu người tiêu dùng không

chấp thuận message trong một khoảng thời gian hợp lý, máy chủ sẽ gửi lại message. Khi người tiêu dùng đã nhận được một message, một báo cáo tiêu thụ (biên lai) sẽ được tạo và gửi đến người gửi message để xác nhận mức tiêu thụ của message.

#### **5.4. Message Formats**

Tùy thuộc vào việc implement MOM, một số định dạng message có thể có sẵn cho người dùng. Một số loại thông báo phổ biến hơn bao gồm: Text (including XML), Object, Stream, HashMaps, Streaming Multimedia,... Các nhà cung cấp MOM có thể cung cấp các cơ chế để chuyển đổi một định dạng message sang một định dạng khác và để chuyển đổi / thay đổi định dạng của tải trọng thư; một số MOM implementation cho phép chuyển đổi XSL được thực hiện bằng tải trọng thông báo XML.

## **II. RABBITMQ**

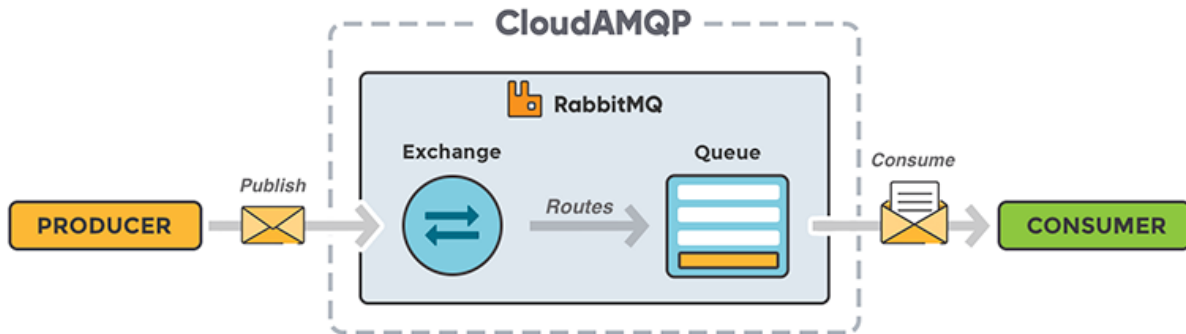


### **1. RabbitMQ là gì?**

RabbitMQ là một message broker ( Message-Oriented Middleware) sử dụng giao thức AMQP(Advanced Message Queue Protocol - Giao thức giao nhận tin nhắn sử dụng hàng đợi). Đây là chương trình đóng vai trò trung gian lưu trữ cũng như điều phối các yêu cầu (message) giữa người nhận (consumer) và người gửi (producer).

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. RabbitMQ sẽ nhận message đến từ các thành phần khác nhau trong hệ thống, lưu trữ chúng an toàn trước khi đẩy đến đích.

## 2. AMQP là gì?



AMQP (Advanced Message Queue Protocol - Giao thức giao nhận tin nhắn sử dụng hàng đợi) là một giao thức làm trung gian cho các gói tin trên lớp ứng dụng với mục đích thay thế các hệ thống truyền tin độc quyền và không tương thích. Các tính năng chính của AMQP là định hướng message, hàng đợi, định tuyến (bao gồm point-to-point và publish-subscribe) có độ tin cậy và bảo mật cao. Các hoạt động sẽ được thực hiện thông qua broker, nó cung cấp khả năng điều khiển luồng (Flow Control).

Một trong các Message Broker phổ biến là 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.

Không giống như các giao thức khác, AMQP là một giao thức có dây (wire-protocol), có khả năng diễn tả các message phù hợp với định dạng dữ liệu, có thể triển khai với rất nhiều loại ngôn ngữ lập trình.

## 3. Tại sao sử dụng RabbitMQ?

- **Vấn đề**

Đối với các hệ thống sử dụng kiến trúc microservice thì việc gọi chéo giữa các service khá nhiều, khiến cho luồng xử lý khá phức tạp

Mức độ trao đổi data giữa các thành phần tăng lên khiến cho việc lập trình trở nên khó khăn hơn (vấn đề maintain khá đau đầu)



Khi phát triển ứng dụng làm sao để các lập trình viên tập trung vào các domain, business logic thay vì các công việc trao đổi ở tầng infrastructure

Với các hệ thống phân tán, khi việc giao tiếp giữa các thành phần với nhau đòi hỏi chúng cần phải biết nhau. Nhưng điều này gây rắc rối cho việc viết code. Một thành phần phải biết quá nhiều đâm ra rất khó maintain, debug

- **Lợi ích**

Một producer không cần phải biết consumer. Nó chỉ việc gửi message đến các queue trong message broker. Consumer chỉ việc đăng ký nhận message từ các queue này.

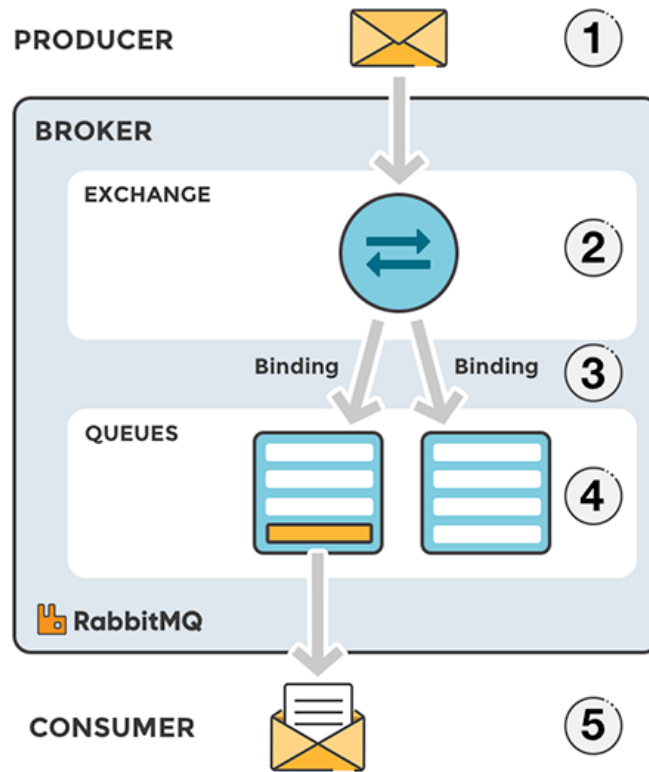
Vì producer giao tiếp với consumer trung gian qua message broker nên dù producer và consumer có khác biệt nhau về ngôn ngữ thì giao tiếp vẫn thành công. (Hiện nay rabbitmq đã hỗ trợ rất nhiều ngôn ngữ khác nhau).

Một đặc tính của rabbitmq là bất đồng bộ(asynchronous). Producer không thể biết khi nào message đến được consumer hay khi nào message được consumer xử lý xong. Đối với producer, đẩy message đến message broker là xong việc. Consumer sẽ lấy message về khi nó muốn. Đặc tính này có thể được tận dụng để xây dựng các hệ thống lưu trữ và xử lý log.

Bên cạnh các lợi ích kể trên, RabbitMQ còn có nhiều tính năng thú vị khác như:

- Cluster: các bạn có thể gom nhiều RabbitMQ instance vào một cluster. Một queue được định nghĩa trên một instance khi đó đều có thể truy xuất từ các instance còn lại. Có thể tận dụng để làm load balancing.
- High availability: 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.

#### 4. Mô hình hoạt động của RabbitMQ



Step 1: Producer đẩy message vào exchange. Khi tạo exchange, bạn phải mô tả nó thuộc loại gì. Các loại exchange sẽ được giải thích phía dưới.

Step 2: Sau khi exchange nhận message, nó chịu trách nhiệm định tuyến message. Exchange sẽ chịu trách về các thuộc tính của message, ví dụ routing key, phụ thuộc loại exchange.

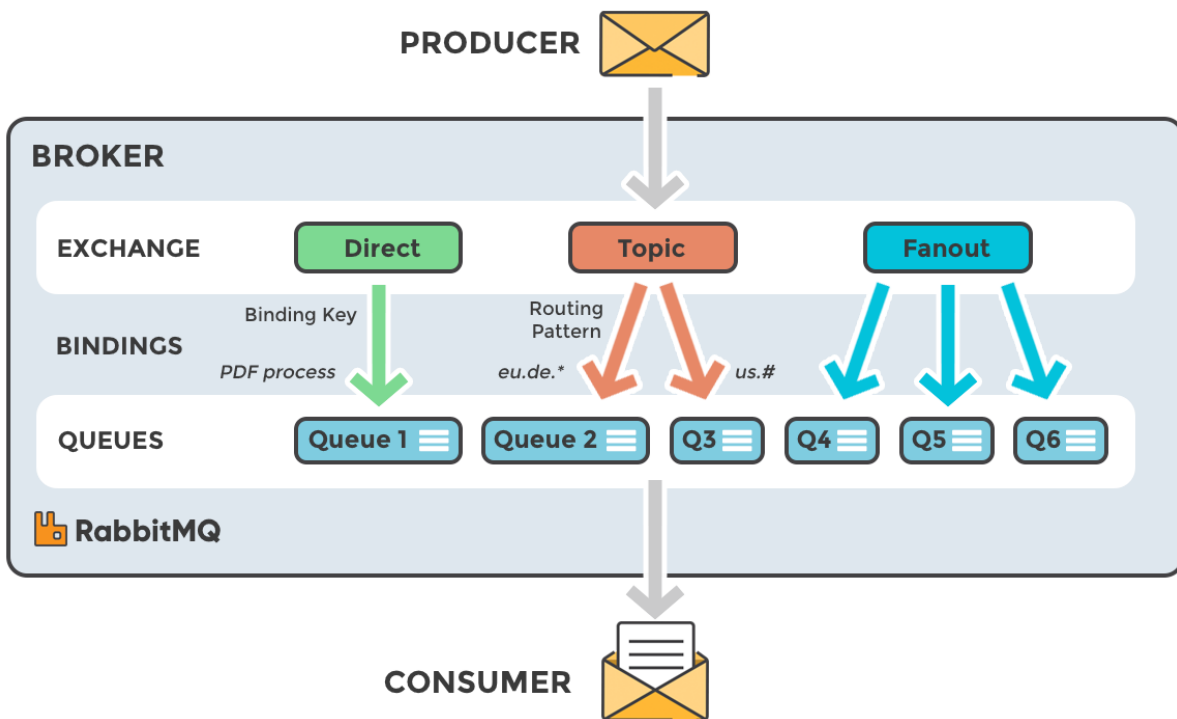
Step 3: Việc binding phải được tạo từ exchange đến hàng đợi. Trong trường hợp này, ta sẽ có hai binding đến hai hàng đợi khác nhau từ một exchange. Exchange sẽ định tuyến message vào các hàng đợi dựa trên thuộc tính của của từng message.

Step 4: Các message nằm ở hàng đợi đến khi chúng được xử lý bởi một consumer.

Step 5: Consumer xử lý message trong queue.

## 5. Exchange và các loại Exchange trong RabbitMQ

Exchange là 1 thực thể AMQP nơi mà các message được gửi. Các exchange lấy 1 message và định tuyến nó tới các hàng đợi. Thuật toán định tuyến được dùng phụ thuộc vào loại exchange và các luật được gọi là bindings.



Có 4 loại Exchange: *direct*, *topic*, *fanout*, *headers*.

### 5.1. Direct Exchange

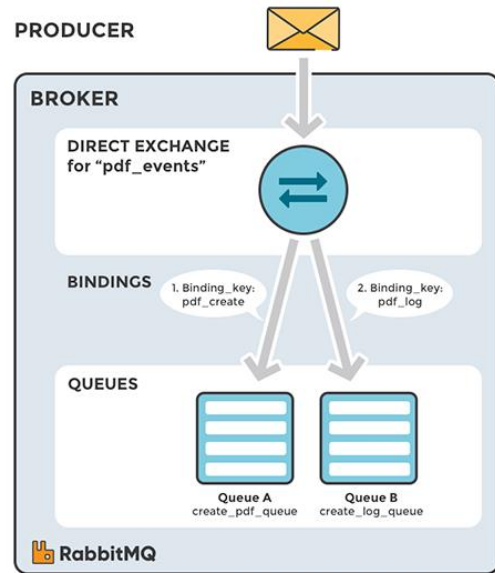
Một Direct Exchange sẽ đẩy message đến hàng đợi dựa theo khóa định tuyến - routing key. Routing key là một thuộc tính của message được thêm vào message header từ producer. Routing key có thể được xem là một địa chỉ mà Exchange sử dụng để định tuyến

các messages. A message goes to the queue(s) whose binding key exactly matches the routing key of the message.

Direct Exchange được sử dụng trong trường hợp bạn muốn phân biệt các messages published cho cùng một Exchange bằng cách sử dụng một chuỗi định danh đơn giản.

Tên khai báo mặc định AMQP brokers cung cấp cho Direct Exchange là "amq.direct".

Ví dụ:



Một message với routing key *pdf\_log* được gửi từ exchange *pdf\_events*. Message đó được định tuyến tới *pdf\_log\_queue* vì outing key (*pdf\_log*) khớp với binding key (*pdf\_log*). Nếu message routing key không khớp với binding key nào thì sẽ bị hủy bỏ.

- **Default Exchange**

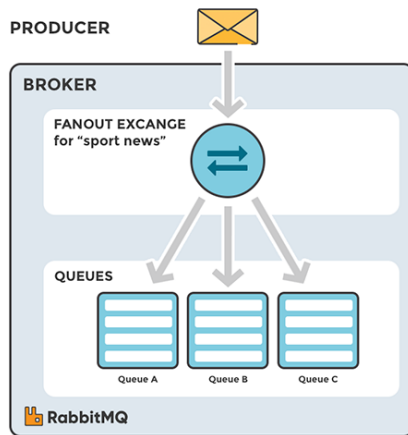
Mỗi một Exchange đều được đặt một tên không trùng nhau, Default Exchange là một Direct Exchange không có tên (thường được biểu diễn bởi một chuỗi rỗng "").

Khi sử dụng Default Exchange, message của bạn được gửi đến queue có tên chính là Routing key của message. Tất cả queue tự động được liên kết với Default Exchange kèm với một Routing key giống như tên của 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".

## 5.2. Fanout Exchange



Một Fanout Exchange sẽ copy message và đẩy message đến toàn bộ hàng đợi gắn với nó. Nó chỉ bỏ qua routing key và bất kỳ pattern nào đã được đăng ký.

Fanout Exchange được dùng trong trường hợp khi một hoặc nhiều message được gửi tới một hoặc nhiều queue với nhiều consumer có thể xử lý cùng một message theo nhiều cách khác nhau.

Tên khai báo mặc định AMQP brokers cung cấp cho Fanout Exchange là “amq.fanout”.

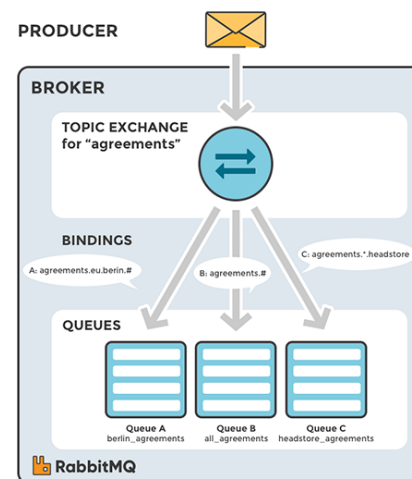
## 5.3. Topic Exchange

Topic Exchange định tuyến các messages tới queue được đối chiếu, đối sánh ký tự (wildcard matches) giữa Routing key và Routing pattern được chỉ định bởi queue binding. Message được định tuyến tới một hoặc nhiều queue được khớp giữa một Routing key của message và Routing pattern của nó.

Routing key phải là một danh sách của các từ (a list of words) phân cách bởi dấu chấm (.) ví dụ “agreements.us” và “agreements.eu.stockholm” trong trường hợp này xác định các thỏa thuận được thiết lập cho một công ty có văn phòng ở nhiều địa điểm khác nhau. Routing pattern có thể chứa dấu hoa thị (\*) để thay thế một từ ở một vị trí cụ thể của Routing key. Ký tự thăng (#) có thể thay thế không hoặc nhiều hơn một từ.

*Ví dụ:*

booking.\*.b -> Được đăng ký bởi tất cả những key bắt đầu bằng booking và kết thúc bằng b.  
booking.# -> Được đăng ký bởi tất cả những key booking hoặc bắt đầu với booking.



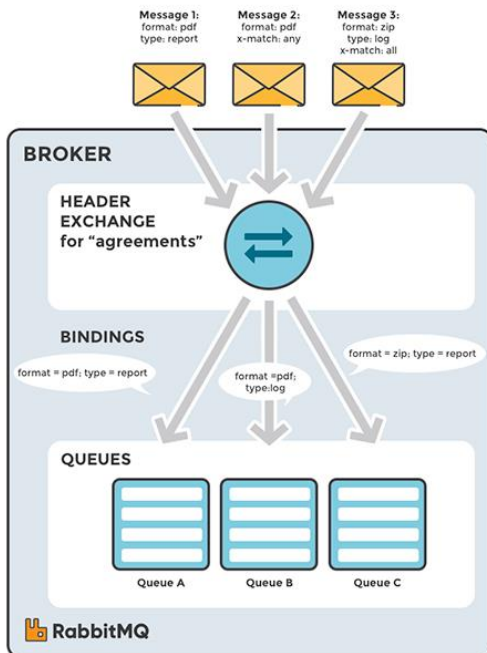
Consumers biểu diễn topic mà họ quan tâm như đăng kí để có một nguồn dữ liệu cho từng thẻ riêng biệt. Consumers tạo ra một queue và thiết lập một binding với một Routing pattern đã cho tới Exchange. Tất cả message với một Routing key khớp với Routing pattern được định tuyến đến queue và ở đó đến khi consumers đến lấy message.

Topic Exchange được sử dụng trong một vài trường hợp:

- 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.

Tên khai báo mặc định AMQP brokers cung cấp cho Topic Exchange là “amq.topic”.

#### 5.4. Headers Exchange



Header Exchange được thiết kế để định tuyến với nhiều thuộc tính, dễ 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 Developer, cụ thể là, nên quan tâm đến những message với tiêu đề nào phù hợp hoặc tất cả chúng.

Tên khai báo mặc định AMQP brokers cung cấp cho Header Exchange trong RabbitMQ là “amq.headers”.

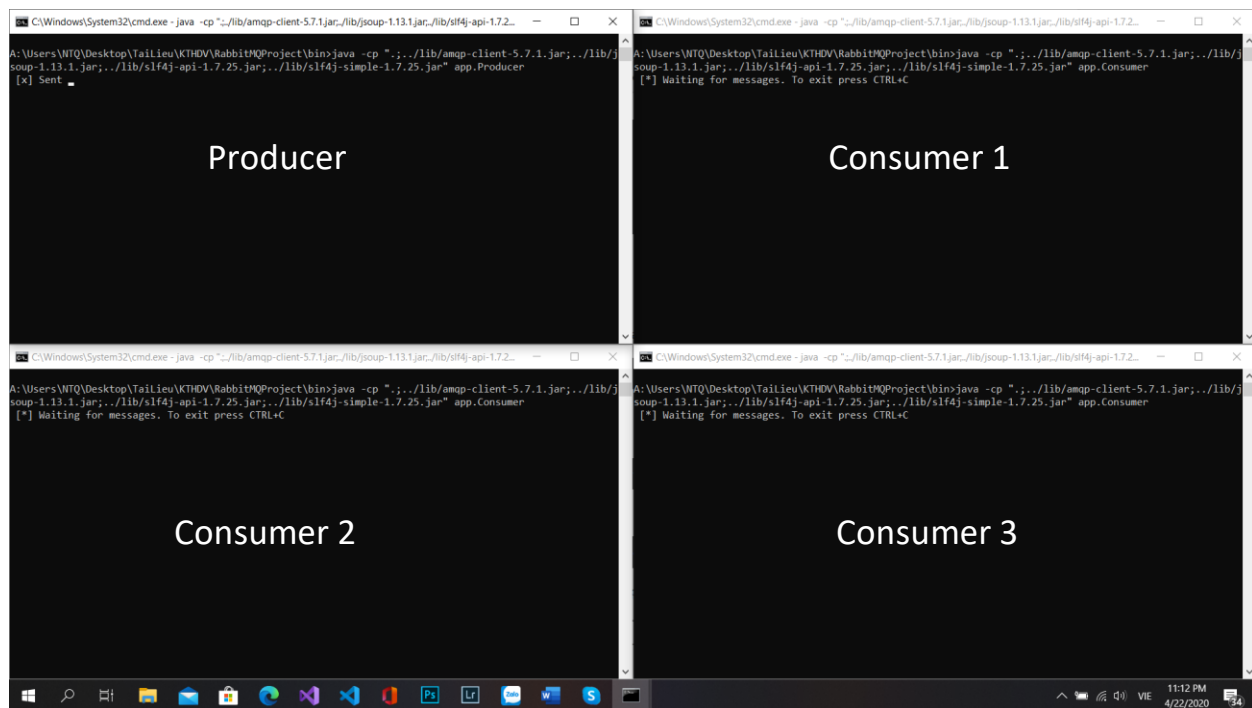
- **Dead Letter Exchange**

Nếu không tìm thấy queue phù hợp cho message, message sẽ tự động bị hủy. RabbitMQ cung cấp một tiện ích mở rộng AMQP được gọi là “Dead Letter Exchange” - Cung cấp chức năng để bắt các message không thể gửi được.

### III. ỨNG DỤNG THỰC TẾ

#### Ứng dụng Crawl Website sử dụng RabbitMQ

Thông qua RabbitMQ tạo một hệ thống xử lý phân tán.



- Producer phân phát các url cần crawl qua cho tất cả các Consumer cùng xử lý.
  - Consumer xử lý url đã nhận, nếu chưa tồn tại thì xử lý và sẽ đẩy kết quả là một list các url liên kết trả về cho Producer.
- > Lặp lại vòng lặp cho đến khi xử lý xong tất cả các url liên kết.

```
C:\Windows\System32\cmd.exe - java -cp ".\lib/amqp-client-5.7.1.jar;.\lib/soup-1.13.1.jar;.\lib/slf4j-api-1.7.2..."
----- : https://www.rabbitmq.com/devtools.html
----- : https://www.rabbitmq.com/news.html#2020-04-09T12:00:00:00
----- : https://www.rabbitmq.com/news.html#2020-03-10T12:00:00:00
----- : https://www.rabbitmq.com/news.html#2020-02-13T12:00:00:00
----- : https://www.rabbitmq.com/protocols.html
----- : https://www.rabbitmq.com/tutorials/tutorial-two-python.html
----- : https://www.rabbitmq.com/reliability.html
----- : https://www.rabbitmq.com/tutorials/tutorial-four-python.html
----- : https://www.rabbitmq.com/tutorials/amqp-concepts.html
----- : https://www.rabbitmq.com/download.html
----- : https://www.rabbitmq.com/clustering.html
----- : https://www.rabbitmq.com/federation.html
----- : https://www.rabbitmq.com/authentication.html
----- : https://www.rabbitmq.com/access-control.html
----- : https://www.rabbitmq.com/ssl.html
----- : https://www.rabbitmq.com/ldap.html
----- : https://www.rabbitmq.com/plugins.html
----- : https://www.rabbitmq.com/management.html
----- : https://www.rabbitmq.com/getstarted.html
----- : https://www.rabbitmq.com/github.html
----- : https://www.rabbitmq.com/contact.html
----- : https://www.rabbitmq.com/trademark-guidelines.html

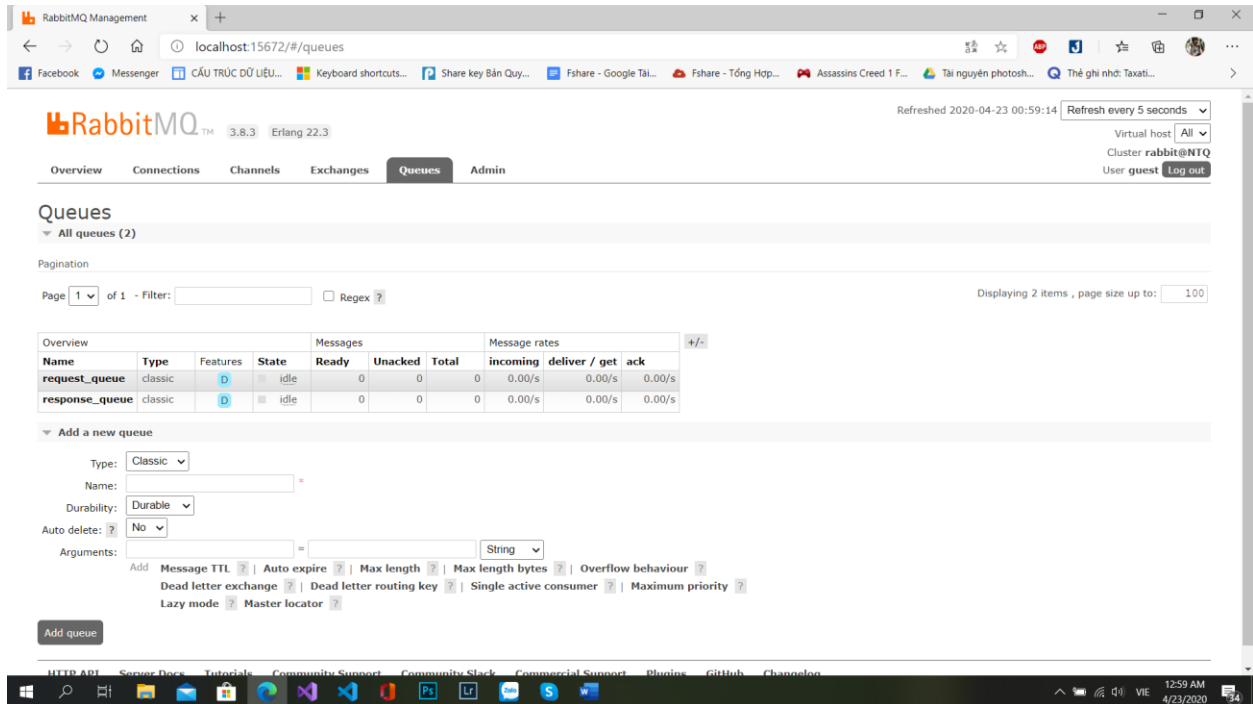
A:\Users\NTQ\Desktop\Tailieu\KTHOV\RabbitMQProject\bin>java -cp ".\lib/amqp-client-5.7.1.jar;.\lib/soup-1.13.1.jar;.\lib/slf4j-api-1.7.25.jar;.\lib/slf4j-simple-1.7.25.jar" app.Consumer
[*] Waiting for messages. To exit press CTRL+C
[x] Received 'https://www.rabbitmq.com/'
[x] Done
[x] Received 'https://www.rabbitmq.com/#getstarted'
[x] Done
[x] Received 'https://www.rabbitmq.com/documentation.html'
[x] Done
[x] Received 'https://www.rabbitmq.com/news.html#2020-04-09T12:00:00:00'
[x] Done
[x] Received 'https://www.rabbitmq.com/protocols.html'

A:\Users\NTQ\Desktop\Tailieu\KTHOV\RabbitMQProject\bin>java -cp ".\lib/amqp-client-5.7.1.jar;.\lib/soup-1.13.1.jar;.\lib/slf4j-api-1.7.25.jar;.\lib/slf4j-simple-1.7.25.jar" app.Consumer
[*] Waiting for messages. To exit press CTRL+C
[x] Received 'https://www.rabbitmq.com/'

A:\Users\NTQ\Desktop\Tailieu\KTHOV\RabbitMQProject\bin>java -cp ".\lib/amqp-client-5.7.1.jar;.\lib/soup-1.13.1.jar;.\lib/slf4j-api-1.7.25.jar;.\lib/slf4j-simple-1.7.25.jar" app.Consumer
[*] Waiting for messages. To exit press CTRL+C
[x] Received 'https://www.rabbitmq.com/#features'
```



Producer và các Consumer gửi và nhận message bằng cách follow request\_queue và response\_queue.



RabbitMQ Management 3.8.3 Erlang 22.3

Overview Connections Channels Exchanges **Queues** Admin

Refreshed 2020-04-23 00:59:14 Refresh every 5 seconds

Virtual host All Cluster rabbit@NTQ User guest Log out

### Queues

All queues (2)

Pagination: Page 1 of 1 - Filter: Regex Displaying 2 items, page size up to: 100

Overview				Messages			Message rates			
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver	get	ack
request_queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	0.00/s
response_queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	0.00/s

Add a new queue

Type: Classic

Name:

Durability: Durable

Auto delete: No

Arguments: = String

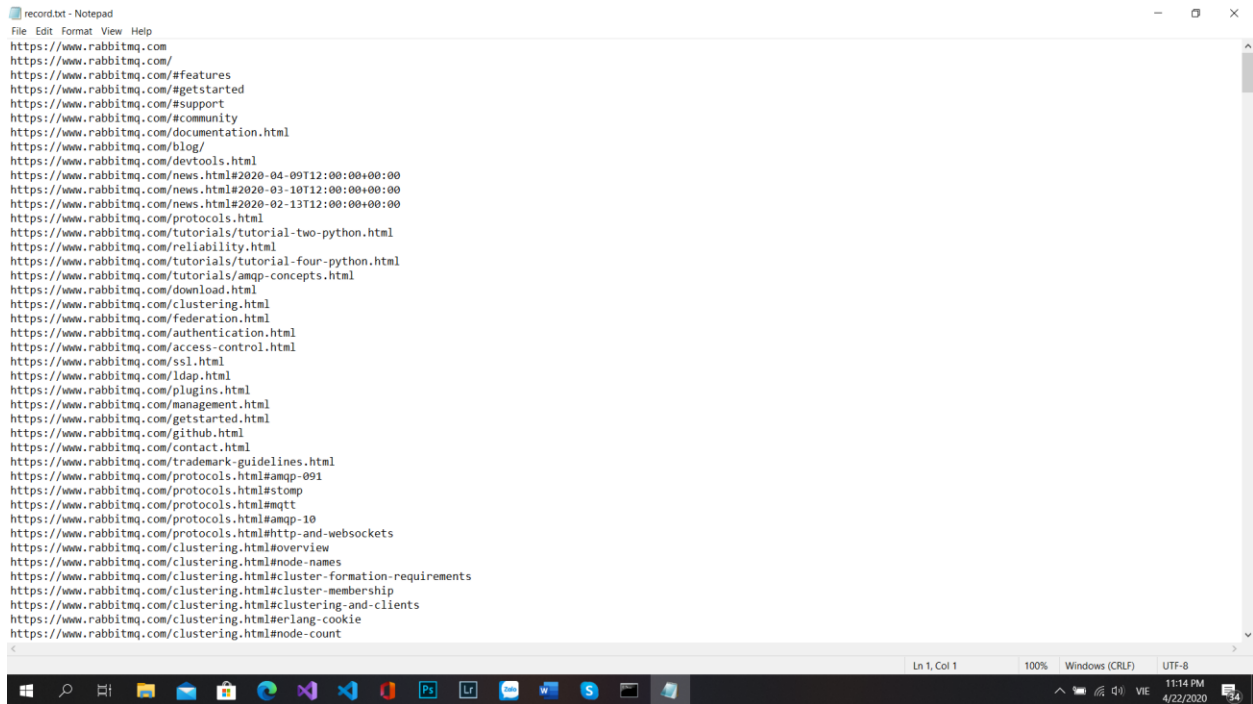
Add Message TTL Auto expire Max length Max length bytes Overflow behaviour Dead letter exchange Dead letter routing key Single active consumer Maximum priority Lazy mode Master locator

Add queue

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

12:59 AM 4/23/2020

Kết quả được xuất ra file record.txt



```
record.txt - Notepad
File Edit Format View Help
https://www.rabbitmq.com
https://www.rabbitmq.com/
https://www.rabbitmq.com/#features
https://www.rabbitmq.com/#getstarted
https://www.rabbitmq.com/#support
https://www.rabbitmq.com/#community
https://www.rabbitmq.com/documentation.html
https://www.rabbitmq.com/blog/
https://www.rabbitmq.com/devtools.html
https://www.rabbitmq.com/news.html#2020-04-09T12:00:00+00:00
https://www.rabbitmq.com/news.html#2020-03-10T12:00:00+00:00
https://www.rabbitmq.com/news.html#2020-02-13T12:00:00+00:00
https://www.rabbitmq.com/protocols.html
https://www.rabbitmq.com/tutorials/tutorial-two-python.html
https://www.rabbitmq.com/reliability.html
https://www.rabbitmq.com/tutorials/tutorial-four-python.html
https://www.rabbitmq.com/tutorials/amqp-concepts.html
https://www.rabbitmq.com/download.html
https://www.rabbitmq.com/clustering.html
https://www.rabbitmq.com/federation.html
https://www.rabbitmq.com/authentication.html
https://www.rabbitmq.com/access-control.html
https://www.rabbitmq.com/ssl.html
https://www.rabbitmq.com/ldap.html
https://www.rabbitmq.com/plugins.html
https://www.rabbitmq.com/management.html
https://www.rabbitmq.com/getstarted.html
https://www.rabbitmq.com/github.html
https://www.rabbitmq.com/contact.html
https://www.rabbitmq.com/trademark-guidelines.html
https://www.rabbitmq.com/protocols.html#amp-091
https://www.rabbitmq.com/protocols.html#stomp
https://www.rabbitmq.com/protocols.html#mqtt
https://www.rabbitmq.com/protocols.html#amp-10
https://www.rabbitmq.com/protocols.html#http-and-websockets
https://www.rabbitmq.com/clustering.html#overview
https://www.rabbitmq.com/clustering.html#node-names
https://www.rabbitmq.com/clustering.html#cluster-formation-requirements
https://www.rabbitmq.com/clustering.html#cluster-membership
https://www.rabbitmq.com/clustering.html#clustering-and-clients
https://www.rabbitmq.com/clustering.html#erlang-cookie
https://www.rabbitmq.com/clustering.html#node-count
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8 11:14 PM 4/22/2020