2020

Tìm hiểu về Message Broker và RabbitMQ

NHÓM SOA

Thông tin nhóm trong tệp Readme.md

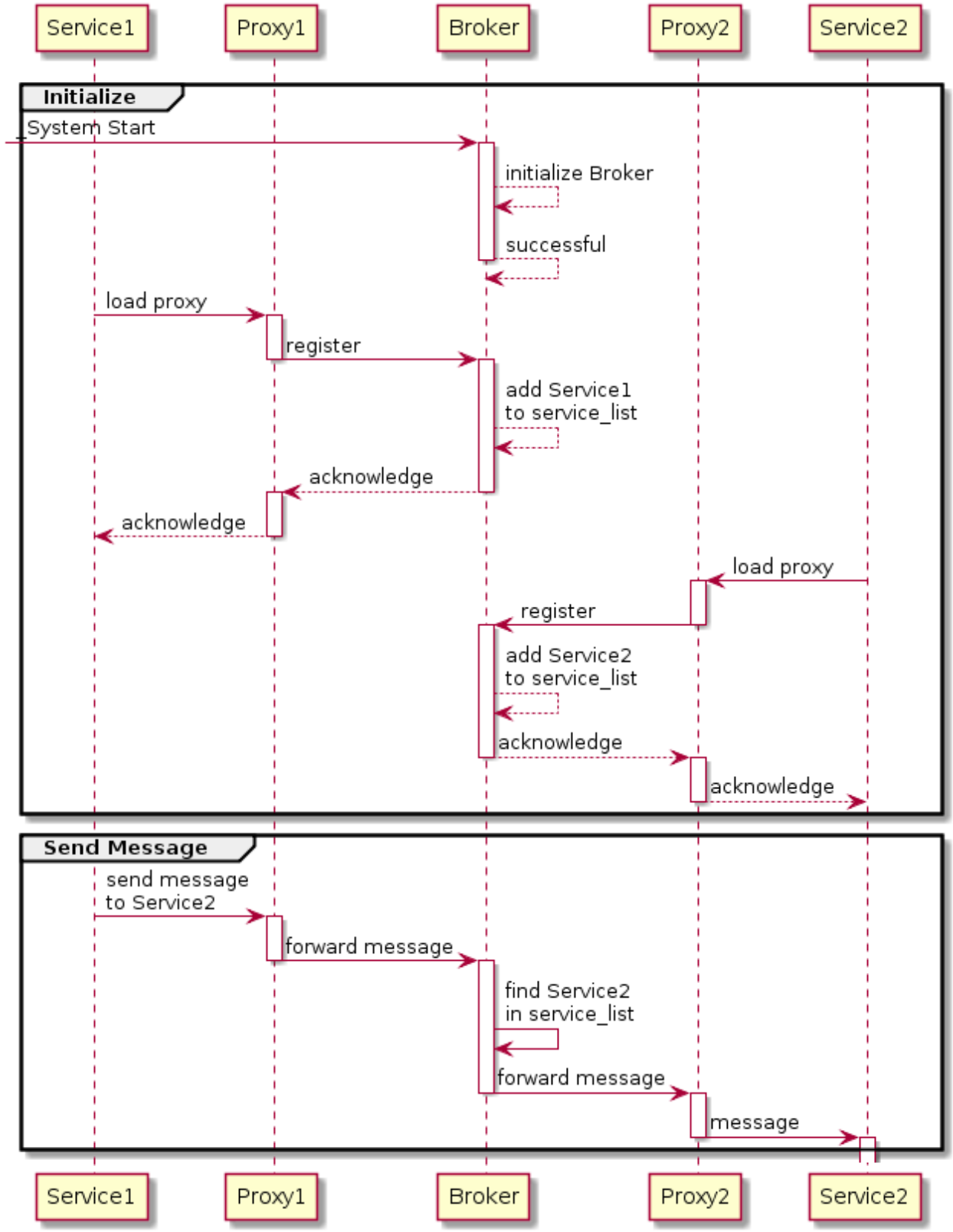
1. Message Broker (Message Oriented Middleware)

- 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.

- Là một mô-đun trung gian (middleware) làm cầu nối giữa phương thức messaging của bên gửi và bên nhận. Message Broker là một mẫu thiết kế nhằm thẩm định, vận chuyển và định tuyến messages. Nó điều tiết sự giao tiếp giữa các ứng dụng, làm giảm đi sự can thiệp trực tiếp của ứng dụng vào quá trình giao tiếp, từ đó làm giúp giải quyết bài toán về ràng buộc giữa chúng. Bởi vì bên gửi và bên nhận không trực tiếp thao tác với nhau, nó cũng dẫn đến sự bảo mật tốt hơn với cả hai bên. Mục đích chính của Message Broker là nhận messages từ các ứng dụng và làm một số thao tác với chúng. Message Broker chủ yếu được dựa trên hai mẫu thiết kế: hub-and-spoke và message bus. Ở hub-and-spoke, một server trung tâm cung cấp những dịch vụ tích hợp. Ở message bus, message broker đóng vai như xương sống cho sự giao tiếp hay là một dịch vụ phân tán hoạt động trên bus.

– Miêu tả luồng hoạt động:

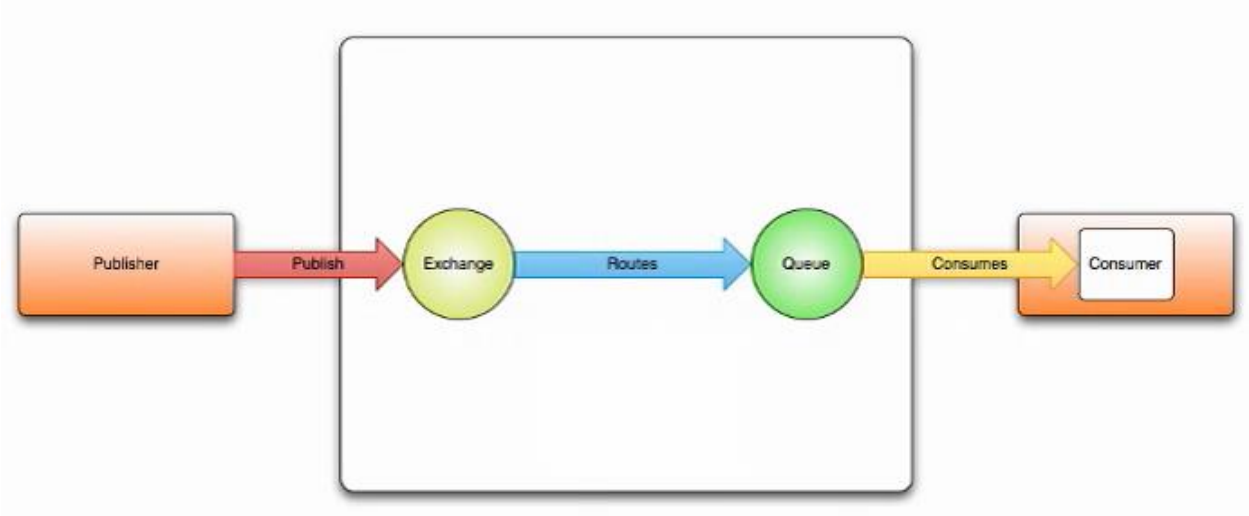


- Sau quá trình khởi tạo, Service 1 và Service 2 load proxy và register đến Broker. Từ đó, Message Broker vận chuyển các messages đến các proxy đã được register từ trước. Ta có thể thấy được một số lợi ích từ mô hình này:
    + Hai service không cần biết nhau, chúng chỉ cần gửi messages đến Broker trung gian. Từ đó, Broker sẽ tự lo việc vận chuyển các message đến nơi đã được đăng kí nhận.
    + Do service 1 và service 2 không giao tiếp trực tiếp, nhờ đó mà chúng có thể khác nhau về ngôn ngữ lập trình.
    + Với mô hình này, chúng ta có thể cài đặt cơ chế bất đồng bộ (asynchronous). Khi một service gửi message thì nó không cần quan tâm lúc nào service khác nhận được message hay khi nào service đó xử lý message xong, và service nhận message cũng có thể lấy message bất cứ khi nào nó muốn.
- Vậy khi nào thì cần Message Broker:
    + Khi cần kiểm soát lưu lượng dữ liệu ra vào.
    + Truyền dữ liệu đến một số ứng dụng khác nhau.
    + Khi cần hoàn thiện một công việc theo một trật tự nhất định (VD: hệ thống giao dịch)
- Phân loại Message broker:
    + Publish and Subscribe Messaging:
        ❖ Trong hệ thống này, ta có nhà cung cấp (publisher) và người nhận (subscriber). Một hay nhiều nhà cung cấp có thể cung cấp một hay nhiều messages thuộc cùng một topic, mà một người nhận có thể nhận được các messages từ một hay nhiều nhà cung cấp. Người nhận theo dõi (subscribe) một topic, mà mọi messages thuộc topic đó sẽ được gửi đến người nhận. Mô hình này cung cấp một framework dựa trên sự quan tâm của người nhận (interest-driven).
    + Point-to-Point Communication:
        ❖ Point-to-Point là hình thức giao tiếp đơn giản nhất giữa một nhà cung cấp và một người nhận. Với mô hình trao đổi thông tin này, một hàng đợi (queue) sẽ được sử dụng để chứa các messages đến khi chúng đến được người nhận. Nhà cung cấp gửi message đến hàng đợi, người nhận sẽ lấy message từ hàng đợi và trả lại xác nhận (ack) message đã được nhận. Nhiều nhà cung cấp và nhiều người nhận có thể chia sẻ chung một hàng đợi. Nhưng với trường hợp nhiều người nhận, mỗi người nhận thường sẽ chỉ lấy một số các messages chứ không phải toàn bộ.

2. 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.

2.1. Messages

- Messages trong AMQP 0-9-1 bao gồm nhiều thuộc tính như content type, content encoding, routing key, delivery mode (persistent hoặc impersistent), message priority, message publishing timestamp, expiration period, pulisher id. Một số thuộc tính là ngoại lệ (được gọi là headers tương tự như X-Headers trong HTTP). Các thuộc tính của messages được khởi tạo sau khi Pulisher gửi messages.
- Messages cũng có thêm một thuộc tính payload, đại diện cho dữ liệu mà messages đó chứa, AMQP brokers định nghĩa payload là một chuỗi dữ liệu không có kiểu dữ liệu cụ thể, tuy nhiên messages cũng không cần có payload mà chỉ có các thuộc tính nêu trên. Broker không có khả năng thay đổi payload của messages. Payload của messages thường được biểu diễn dưới các format

như JSON, Thrift, Protocol Buffers và MessagePack, thông tin về kiểu format sử dụng được lưu trữ trong thuộc tính content type hoặc content encoding.

- Messages có thể được gửi dưới dạng persistent, ow các messages đó được broker lưu trữ lại trong bộ nhớ. Nếu trong trường hợp server được khởi động lại, thì hệ thống luôn đảm bảo các messages dưới mode persistent không bị thất lạc. Tuy nhiên việc lưu trữ lại nhiều messages trong bộ nhớ sẽ dẫn đến ảnh hưởng tới hiệu suất của hệ thống.

2.2. 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.

2.3. 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.

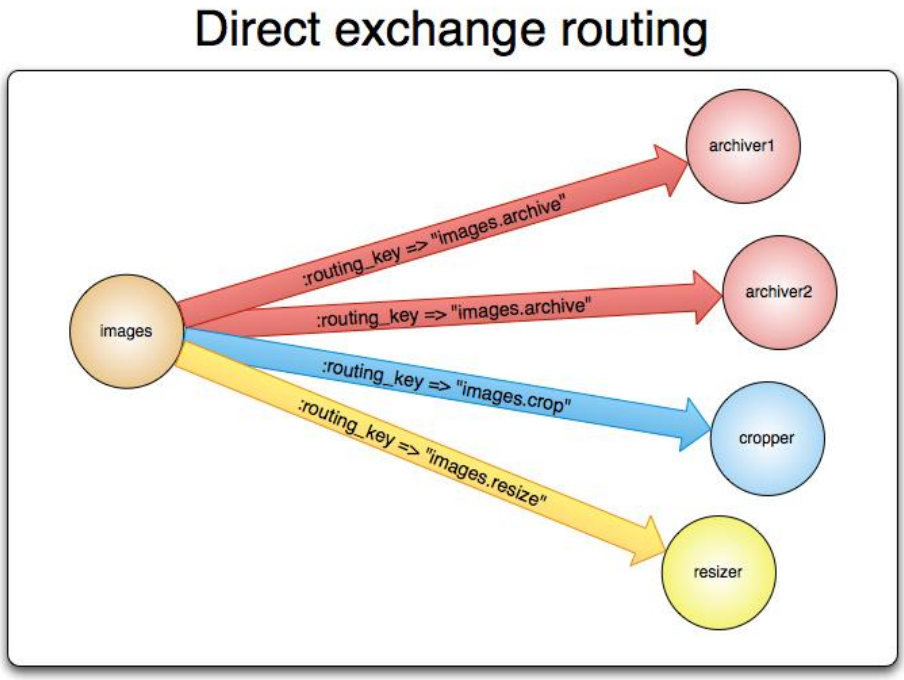
Name	△ Type	Features	Message rate in	Message rate out	+/-
(AMQP default)	direct	D	0.00/s	0.00/s	
amq.direct	direct	D			
amq.fanout	fanout	D			
amq.headers	headers	D			
amq.match	headers	D			
amq.rabbitmq.trace	topic	D I			
amq.topic	topic	D			

▸ Add a new exchange

2.3.1. Default exchange

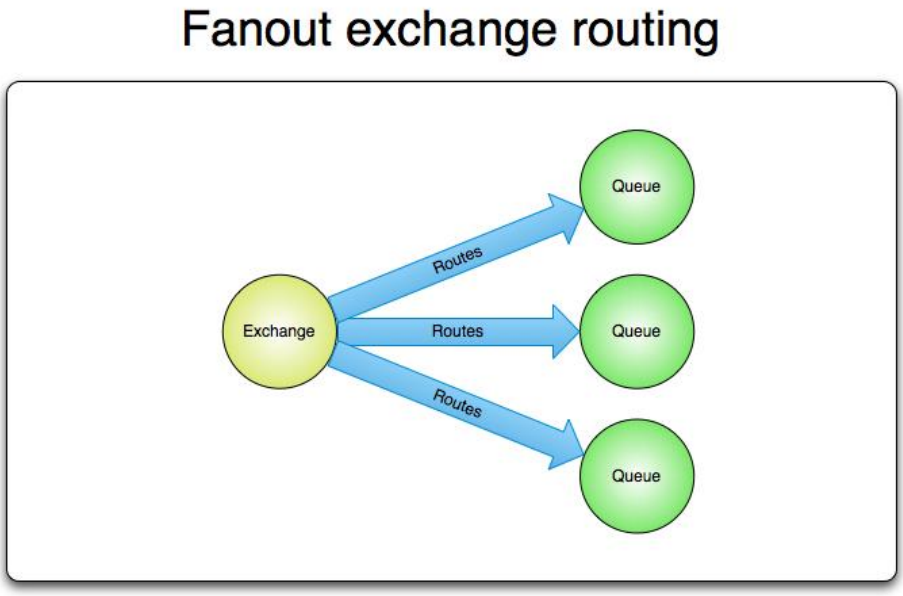
- Default exchange là một Direct Exchange không có thuộc tính Name định nghĩa sẵn bởi message broker (theo cách hiểu khác là Name sẽ được định nghĩa bởi người dùng). Và routing key sẽ được lấy đúng bằng Name do người dùng đặt sẵn, từ đó các messages với routing key trùng với Name của Default exchange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

2.3.2. 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.

2.3.3. 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.

2.3.4. 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, ...)

2.3.5. 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.

2.4. Queue

- Queue bao gồm các thuộc tính:
  + Name
  + Durable: Durable Queue không ảnh hưởng bởi việc broker restart
  + Exclusive (chỉ dùng cho một kết nối và queue sẽ bị xóa khi kết nối đóng)
  + 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)

2.4.1. Queue Names

- Các ứng dụng có thể tự đặt tên hoặc yêu cầu broker sinh tên. Tên đặt do broker sẽ bắt đầu là "amq.".

2.4.2. Queue Durability

- Durability Queue được lưu trên dĩa và vẫn sẽ sống kể cả khi broker sập/khởi động lại.
- Tuy nhiên không phải kịch bản sử dụng nào cũng sẽ yêu cầu Queue luôn sống. Và mặc dù broker khi khởi động lại, thì queue sẽ vẫn sẽ khởi tạo lại và chỉ có message là giữ lại.

2.4.3. Bindings

- Bindings là các quy tắc mà exchanges sử dụng để định tuyến các messages tới queue. Để exchange định tuyến message đến queue nào đó, thì exchange phải gán(bind) queue. Bindings có thể gồm thuộc tính là routing key được sử dụng trong một số loại exchange.
- Routing key dùng để lấy những tin nhắn cụ thể được publish từ exchange đến queue.
- **VD: Đi máy bay từ điểm sân bay A đến B. Queue là đích (điểm B), Exchange là sân bay (điểm A). Bindings là đường đi từ A đến B (Có thể có nhiều đường đi).**

2.5. Message Acknowledgement (Ack)

- Là một application của Consumer chịu trách nhiệm việc nhận và xử lí messages, tuy nhiên MA có thể gặp tình trạng không thể xử lí được messages của từng cá nhân hoặc gặp sự cố khiến MA dừng hoạt động. Một trong nhiều nguyên nhân gây ra là lỗi liên quan đến mạng. Do đó, AMQP 0-9-1 cài đặt các

acknowledgement modes nhằm giao cho Consumer quyền kiểm soát quá trình nhận messages sau khi messages broker gửi một messsage tới MA (mode 1) và sau khi MA gửi lại thông báo acknowledgement (mode 2).

- Mode 1 gọi là Automatic Acknowledgement, mode 2 gọi là Explicit Acknowledgement. Đối với mode 2, MA quyết định thời điểm gửi lại thông báo tới broker (có thể là ngay sau khi nhận được messages, hoặc sau khi lưu trữ lại messages, hay sau khi đã xử lí hoàn tất messages).
- Nếu một Consumer không còn hoạt động (không còn khả năng nhận messages) mà không gửi trả acknowledgement tới broker, thì broker sẽ gửi lại messages tới một Consumer khác hoặc nếu không có Consumer hoạt động, thì broker sẽ thực hiện chờ ít nhất một Consumer bắt đầu hoạt động trong Queue phân phát messages rồi gửi messages tới đó.
- Trong trường hợp nhiều Consumers cùng chia sẻ một Queue, phương hướng xử lý tốt là quy định số lượng consumers được nhận messages một lần gửi nhằm xử lý load balancing, nâng cao throughput trong trường hợp messages được gửi theo các batches trong thời gian ngắn.

2.6. Channel

- Một số applications cần sử dụng nhiều kết nối tới broker, tuy nhiên việc duy trì nhiều kết nối TCP sẽ gây hao tổn tài nguyên hệ thống và khó khăn trong kiểm soát bảo mật. Do đó, AMQP 0-9-1 thực hiện ghép kênh, hay tạo ra các channels với một channel như là một kết nối TCP.
- Channels của AMQP 0-9-1 khá lightweight, sử dụng số lượng tương đối ít tài nguyên bộ nhớ clients. Tuy nhiên tùy thuộc vào các thư viện clients cài đặt, channel có thể thay đổi theo, ví dụ như khi client sử dụng multithread, vì thế nên giới hạn số lượng channels trong một kết nối. Trong trường hợp số lượng channels mà clients khởi tạo tối đa vượt quá thì kết nối trên channel đó sẽ bị dừng, hệ thống sẽ thông báo lỗi tới clients.
- Tất cả các thao tác của clients tới hệ thống được thực hiện thông qua một channel. Để đảm bảo tính riêng rẽ, các giao tiếp trên một channel được tách biệt hoàn toàn với các channels khác, do đó mỗi giao thức thông qua channels đều được gán một channel ID.
- Khi một kết nối được ngắt thì các channels chứa kết nối đó cũng được ngắt theo. Nếu một application sử dụng multithreads hoặc multiprocesses thì mỗi channel được gán cho một thread hoặc một process và các channels tách biệt nhau.

2.6.1. Channel Lifecycle

- Sau khi application tạo thành công kết nối, channel sẽ được khởi tạo.
- Khi clients không cần sử dụng channels thì channels sẽ được đóng và tài nguyên của channels sẽ được hệ thống lấy lại. Các giao thức trên channels bị đóng sẽ bị hệ thống gửi trả thông báo lỗi.

2.6.2. Common Channels Errors

- Channel Leaks:
  + Channel Leaks là trường hợp một application mở liên tục nhiều channels mà không dừng các channels đó hoặc chỉ đóng một số ít các channels đó. Tình trạng này dần dần sẽ tiêu hao RAM và CPU của node (node là một cài đặt server của mô hình AMQP 0-9-1).
- High Channel Churn (HCC)
  + HCC là trường hợp tốc độ channels mới được mở và channels được đóng cao, hiện tượng này mô tả các applications sử dụng short lived channels hoặc các channels bị đóng do lỗi liên quan đến channels.
  + Để xử lý các tình trạng trên, RabbitMQ đã cung cấp cho người dùng CLI tools có khả năng thống kê tài nguyên sử dụng bởi channels.

3. Làm việc với RabbitMQ

- Project sử dụng RabbitMQ ở đây là Web Crawler lấy thông tin phim trên IMDB viết bằng Node.js.
- Yêu cầu cài đặt:
  + Erlang (không có thì sẽ không cài được RabbitMQ)
  + RabbitMQ
  + Node.js 12.16.2
  + Sau khi cài xong Node.js chúng ta
  + Khởi tạo project webCrawler và cài đặt amqplib-as-promised (một module cải tiến từ amqplib để hỗ trợ viết hàm promise dễ hơn), cheerio (để đọc nội dung web), và request (để xử lý các phương thức gọi http).

- RabbitMQ ở đây chúng ta sẽ sử dụng amqplib-as-promised dành cho Node.
- Exchange type sử dụng trong project ở đây là Default Exchange và Direct Exchange (2 exchange type này đã được miêu tả ở mục 2.3.1 và 2.3.2).

3.1. Default Exchange

3.1.1. Gửi message

- Ta có một tệp **sender.js** như sau:

```
1  const { Connection } = require('amqplib-as-promised');
2
3  const getMovieByIndex = (index) => {
4    index += 137523;
5    return `https://www.imdb.com/title/tt0${index}`;
6  }
7
8  async function sender() {
9    const connection = new Connection('amqp://localhost');
10   await connection.init();
11   const channel = await connection.createChannel();
12
13   var queue = 'filmQueue';
14   await channel.assertQueue(queue, {durable: true});
15
16   for (var i = 0; i < 20; i++)  {
17     var url= getMovieByIndex(i);
18     await channel.sendToQueue(queue, Buffer.from(url), { persistent: true });
19     console.log('Sent: ' + url);
20   }
21   await channel.close();
22   await connection.close();
23   process.exit(0);
24 }
25
26 sender();
```

- Miêu tả:
  - Chúng ta tạo biến Connection từ thư viện amqplib-as-promised đã cài.
  - Trong hàm sender:
    - Chúng ta khởi tạo kết nối tới RabbitMQ (dòng 9 & 10).
    - Sau đó ta khởi tạo Channel (dòng 11) và hàng đợi có tên là filmQueue (dòng 13 & 14). Lưu ý là hàng đợi ở đây sẽ chỉ tạo khi chưa tồn tại, durable như đã nói qua được khai báo true ở đây tức là hàng đợi sẽ không chết ngay cả RabbitMQ bị tắt.
    - Message được lấy từ getMovieByIndex, và được đẩy vào queue. Như trên thì đẩy 20 message vào queue (dòng 16 – 20). Khi đẩy vào queue thì có persistent tức là yêu cầu RabbitMQ lưu message vào bộ nhớ, tuy nhiên là không phải ngay lập tức vì sẽ có một khoảng thời gian ngắn nhất định để Rabbit lưu. Nội dung message có dạng byte.
    - Sau đó đóng channel và kết nối (dòng 21 – 23).

3.1.2. Nhận message

- Ta có tệp **receiver.js** như sau:

```
1 const { Connection } = require('amqplib-as-promised');
2 const request = require('request');
3 const cheerio = require('cheerio');
4 const fs = require('fs');
5
6 function crawler(error, response, html) {
7       if (!error) {
8           let $ = cheerio.load(html);
9           var film = { title: '', releaseDate: '', rating: '' }
10          film.title = $('.title_wrapper').children('h1').text().trim();
11          film.releaseDate = $('#titleYear').children('a').text().trim();
12          film.rating = $('.ratingValue').children('strong').children('span').attr('itemprop',
  'ratingValue').text().trim();
13          if (film.title ≠ '') {
14              var title = film.title.trim().replace(/[<>:;%\$\s]+/g, '-');
15
16              fs.writeFile(`filmList/${title}.json`, JSON.stringify(film) , function (err) {
17                if (err) {
18                    throw err;
19                }
20                console.log('Crawled %s !', film.title.trim());
21              });
22          }
23      }
24 }
25
26 async function receiver() {
27     const connection = new Connection('amqp://localhost');
28     await connection.init();
29     const channel = await connection.createChannel();
30
31     var queue = 'filmQueue';
32     await channel.assertQueue(queue, {durable: true});
33
34     channel.prefetch(1);
35     console.log(" [*] Waiting for messages in %s. To exit press CTRL+C", queue);
36     await channel.consume(queue, function(msg) {
37                     var url = msg.content.toString();
38                     console.log("Received: ", url);
39                     setTimeout(() ⇒ {
40                         channel.ack(msg);
41                         console.log('Acked!');
42                         request(url, crawler);
43                     }, 1000);
44                 }, { noAck: false });
45 }
46
47 receiver();
48
```

- Miêu tả:
  - Chúng ta tạo biến Connection giống như đã làm ở tệp sender.js và thêm các biến request (để gọi http request), cheerio (để crawl dữ liệu), fs (để viết ra file json).
  - Trong hàm receiver:
    - Chúng ta cũng vẫn khởi tạo kết nối tới RabbitMQ, Channel và hàng đợi filmQueue như ở sender (từ dòng 27 – 32). Lưu ý là hàng đợi cũng được khai báo ở đây,

vì trường hợp xảy ra nếu receiver chạy trước khi sender thì chúng ta cần chắc chắn là có thể lấy được message từ đó.

- ❖ Hàm consume lấy tất cả cả message từ hàng đợi (dòng 36). Message ở đây sẽ được lấy một cách bất đồng bộ. Và sau khi lấy được message từ queue thành công chúng ta phải lại ack để báo với RabbitMQ là đã gửi thành công. Lưu ý trước hàm consume ta có prefetch(1), tức là chỉ cho phép RabbitMQ gửi message mới một khi message trước đó đã báo gửi thành công.

+ Trong hàm crawler:
  - ❖ Để lấy nội dung mà cụ thể là phim từ website mà ghi nôi dung ra file json sử dụng tiêu đề phim là tên tệp.

3.1.3. Kết quả

– Trong Terminal, gõ node sender, ta có kết quả sau:

```
PS I:\Software Engineering\SOA\Crawler> node sender
Sent: https://www.imdb.com/title/tt0137523
Sent: https://www.imdb.com/title/tt0137524
Sent: https://www.imdb.com/title/tt0137525
Sent: https://www.imdb.com/title/tt0137526
Sent: https://www.imdb.com/title/tt0137527
Sent: https://www.imdb.com/title/tt0137528
Sent: https://www.imdb.com/title/tt0137529
Sent: https://www.imdb.com/title/tt0137530
Sent: https://www.imdb.com/title/tt0137531
Sent: https://www.imdb.com/title/tt0137532
Sent: https://www.imdb.com/title/tt0137533
Sent: https://www.imdb.com/title/tt0137534
Sent: https://www.imdb.com/title/tt0137535
Sent: https://www.imdb.com/title/tt0137536
Sent: https://www.imdb.com/title/tt0137537
Sent: https://www.imdb.com/title/tt0137538
Sent: https://www.imdb.com/title/tt0137539
Sent: https://www.imdb.com/title/tt0137540
Sent: https://www.imdb.com/title/tt0137541
Sent: https://www.imdb.com/title/tt0137542
PS I:\Software Engineering\SOA\Crawler>
```

– Sau đó, gõ node receiver, ta có kết quả sau:

```
PS I:\Software Engineering\SOA\Crawler> node receiver
 [*] Waiting for messages in filmQueue. To exit press CTRL+C
Received:  https://www.imdb.com/title/tt0137524
Acked!
Received:  https://www.imdb.com/title/tt0137526
Acked!
Received:  https://www.imdb.com/title/tt0137528
Acked!
Received:  https://www.imdb.com/title/tt0137530
Crawled The Final Verdict (1914) !
Crawled Flesh and Boner (1994) !
Acked!
Received:  https://www.imdb.com/title/tt0137532
Acked!
Received:  https://www.imdb.com/title/tt0137534
Crawled Flesh for Fantasies (1986) !
Crawled Flesh for Frankenstein (1988) !
Acked!
Received:  https://www.imdb.com/title/tt0137536
Crawled Flesh... and the Fantasies (1991) !
Acked!
Received:  https://www.imdb.com/title/tt0137538
Crawled Floor Play (1989) !
Acked!
Received:  https://www.imdb.com/title/tt0137540
Crawled Fly Now, Pay Later (1969) !
Acked!
Received:  https://www.imdb.com/title/tt0137542
Acked!
Crawled Fondle with Care (1989) !
Crawled Flying High with Tracey Adams (1987) !
```

3.2. Direct Exchange

3.2.1. Gửi message

– Ta có một tệp **senderDirectExchange.js** như sau:

```
1  const { Connection } = require('amqplib-as-promised');
2
3  const getMovieByIndex = (index) => {
4    index += 137523;
5    return `https://www.imdb.com/title/tt0${index}`;
6  }
7
8  async function sender() {
9    const connection = new Connection('amqp://localhost');
10   await connection.init();
11   const channel = await connection.createChannel();
12
13   var args = process.argv.slice(2);
14   var routing_key = (args.length > 0) ? args[0] : 'info';
15
16   var exchange = 'exchangeFilmURLs';
17   channel.assertExchange(exchange, 'direct', { durable: true
18 });
19   for (var i = 0; i < 10; i++)  {
20     var url= getMovieByIndex(i);
21     channel.publish(exchange, routing_key, Buffer.from(url));
22     console.log("Sent [%s]: '%s'", routing_key, url);
23   }
24   await channel.close();
25   await connection.close();
26   process.exit(0);
27 }
28
29 sender();
30
```

– Miêu tả:

+ Trong hàm sender:

❖ Tương tự như tệp sender.js của Default Exchange, chúng ta cũng tạo Connection, Channel tương tự. Tuy nhiên, bắt đầu từ dòng 16 trở đi, lần này chúng ta sẽ không đặt tên hàng đợi cụ thể như trước, mà thay vào đó, chúng ta tạo exchange rồi bind bằng routing key (mặc định là info) được sử dụng làm routing key). Sau đó chúng ta publish exchange (dòng 21).

❖ Sau đó đóng channel và kết nối (dòng 24 – 26).

3.2.2. Nhận message

– Ta có tệp **receiverDirectExchange.js** như sau:

```
const { Connection } = require('amqplib-as-promised');
const request = require('request');
const cheerio = require('cheerio');
const fs = require('fs');

function crawler(error, response, html) {
    if (!error && response.statusCode == 200) {
        let $ = cheerio.load(html);
        var film = { title: '', releaseDate: '', rating: '' }
        film.title = $('.title_wrapper').children('h1').text().trim();
        film.releaseDate = $('#titleYear').children('a').text().trim();
        film.rating = $('.ratingValue').children('strong').children('span').attr('itemprop',
'ratingValue').text().trim();
        // console.log(film);
        if (film.title != '') {
            var title = film.title.trim().replace(/[<>:;%\$\s]+/g, '-');
            fs.writeFile(`filmList/${title}.json`, JSON.stringify(film) , function (err) {
                if (err) {
                    throw err;
                }
                console.log('Crawled %s !', film.title.trim());
            });
        }
    }
}

async function receiver() {
    const connection = new Connection('amqp://localhost');
    await connection.init();
    const channel = await connection.createChannel();

    var exchange = 'exchangeFilmURLs';
    await channel.assertExchange(exchange, 'direct', { durable: true });

    await channel.assertQueue('', {
        exclusive: true
        }).then((q) => {
            console.log(" [*] Waiting for messages in %s. To exit press CTRL+C", q.queue);

            var args = process.argv.slice(2);
            args.forEach(function (routing_key) {
                channel.bindQueue(q.queue, exchange, routing_key);
            });
            var queue = q.queue;
            channel.consume(queue, function(msg) {
                if (msg != null) {
                    var url = msg.content.toString();
                    console.log(" Received: [%s] %s", msg.fields.routingKey, url);
                    channel.ack(msg);
                    request(url, crawler);
                }
            }, {
                noAck: false,
            });
    }).catch((err) => {
        throw err2;
    });
}

receiver();

```

- Miêu tả:
    + Trong hàm receiver:
        ❖ Gần như tương tự như tệp receiver.js, và giống như đã là với senderDirectExchange.js, chúng ta cũng tạo exchange. Tuy nhiên do hàng đợi không phải do ta đặt tên nên chúng ta phải đăng ký thông qua routing key rồi mới có thể lấy được message ra (từ dòng 34 – hết).

3.2.3. Kết quả

- Trong Terminal, gõ node senderDirectExchange (Lưu ý: mặc định routing key là info), ta có kết quả sau:

```
Sent [info]: 'https://www.imdb.com/title/tt0137532'
PS I:\Software Engineering\SOA\Crawler> node senderDirectExchange
Sent [info]: 'https://www.imdb.com/title/tt0137523'
Sent [info]: 'https://www.imdb.com/title/tt0137524'
Sent [info]: 'https://www.imdb.com/title/tt0137525'
Sent [info]: 'https://www.imdb.com/title/tt0137526'
Sent [info]: 'https://www.imdb.com/title/tt0137527'
Sent [info]: 'https://www.imdb.com/title/tt0137528'
Sent [info]: 'https://www.imdb.com/title/tt0137529'
Sent [info]: 'https://www.imdb.com/title/tt0137530'
Sent [info]: 'https://www.imdb.com/title/tt0137531'
Sent [info]: 'https://www.imdb.com/title/tt0137532'
```

– Trong Terminal, gõ node receiverDirectExchange info (Lưu ý: mặc định routing key là info), ta có kết quả sau:

```
PS I:\Software Engineering\SOA\Crawler> node receiverDirectExchange info
 [*] Waiting for messages in amq.gen-EJ4k9UxlFD1X3MjLhrzVtQ. To exit press CTRL+C
 Received: [info] https://www.imdb.com/title/tt0137523
 Received: [info] https://www.imdb.com/title/tt0137524
 Received: [info] https://www.imdb.com/title/tt0137525
 Received: [info] https://www.imdb.com/title/tt0137526
 Received: [info] https://www.imdb.com/title/tt0137527
 Received: [info] https://www.imdb.com/title/tt0137528
 Received: [info] https://www.imdb.com/title/tt0137529
 Received: [info] https://www.imdb.com/title/tt0137530
 Received: [info] https://www.imdb.com/title/tt0137531
 Received: [info] https://www.imdb.com/title/tt0137532
Crawled Flesh for Frankenstein (1988) !
Crawled Flesh and Boner (1994) !
Crawled Flesh for Fantasies (1986) !
Crawled Flesh and Ecstasy (1985) !
Crawled The Final Verdict (1914) !
Crawled Fishing for Trouble (1934) !
Crawled Fight Club (1999) !
Crawled Flesh, Sea and Symphony (1993) !
Crawled Flesh... and the Fantasies (1991) !
Crawled Flesh for Fantasy (1994) !
```

– Lưu ý:
  + Message sẽ tự mất nếu không có exchange ko binding hàng đợi nào, và receiver nếu không có thì message sẽ tự động mất luôn. Cho nên chúng ta sẽ chạy receiverDirectChange trước để đăng ký rồi senderDirectChange gửi message.