Công ty An ninh mạng Viettel

Báo cáo về Rabbitmq

Sinh viên: Nguyễn Đan Trường

*MỤC LỤC*

[Chương 1. Tổng quan về Message queue 3](#_Toc159399153)

[Chương 2. Rabbitmq 5](#_Toc159399154)

[1. Hoạt động của RabbitMQ 5](#_Toc159399155)

[2. Các thành phần chính trong RabbitMQ 6](#_Toc159399156)

[3. Consumers 13](#_Toc159399157)

[Chương 3. Cấu hình cluster RabbitMQ 14](#_Toc159399158)

[Chương 4. Lab triển khai Cluster RabbitMQ bằng docker 18](#_Toc159399159)

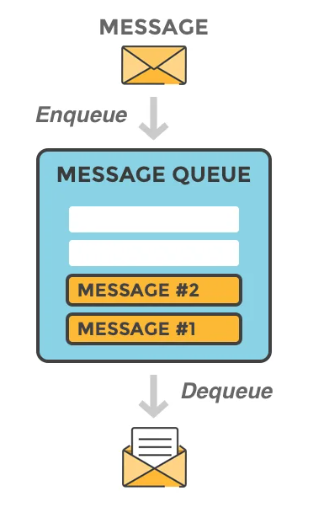
[1. Xây dựng Image 18](#_Toc159399160)

[2. Xây dựng cluster bằng docker-compose 20](#_Toc159399161)

[3. Lab một số kịch bản 24](#_Toc159399162)

# Tổng quan về Message queue

Message queue là một cơ chế trong lập trình và kiến trúc phần mềm, được sử dụng để truyền thông tin (thông điệp) giữa các thành phần của hệ thống mà không cần chúng tương tác trực tiếp với nhau.

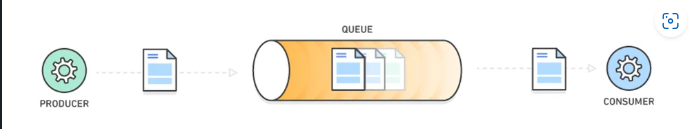


**Các thành phần chính của Message queue**

Message queue gồm các thành phần chính như sau:

* Producer: bộ phân tạo ra thông tin để tương tác với bộ phận khác. Thông tin này sẽ được truyền vào Message queue.
* Consumer: bộ phận/thành phần trong hệ thống nhận thông tin và xử lý thông tin từ Producer thông qua Message queue.
* Message: Thông tin (thông điệp) thường ở dạng text hoặc JSON, đôi khi có thể là Binary do Producer tạo ra.
* Message queue: Nơi lưu trữ tạm thời Message cho tới khi được Consumer lấy ra và xử lý.
* Broker: Xử lý Message và quản lý Message queue để đảm bảo Producer và Consumer truyền thông tin được cho nhau. Broker giúp định tuyến thông tin, quản lý tình trạng của hàng đợi, và đảm bảo rằng thông tin được chuyển giao đúng cách.
* Channel: Là cơ chế truyền thông tin giữa producer và consumer thông qua Message Queue. Channel đóng vai trò như một cầu nối để truyền thông điệp qua lại giữa các bên.

**Cách thức hoạt động của Message queue**



Bước 1: Producer tạo ra thông điệp cần truyền đi. Thông tin này có thể là dữ liệu hoặc các thông tin bổ sung. Thông tin này sẽ được truyền vào Mesage queue thông qua channel và được lưu trữ tạm thời tại đây.

Bước 2: Consumer sẽ lấy thông điệp của Producer thông qua Message queue. Thông tin thường được được lấy theo cơ chế FIFO (first in - first out), tuy nhiên vẫn có thể can thiệp vào cơ chế này bằng cách định ra các ưu tiên.

Bước 3: Sau khi Consumer lấy được thông tin sẽ tiếp tục xử lý và thực hiện các hành động tuỳ thuộc vào yêu cầu của hệ thống.

**Ưu điểm**

* Bất đồng bộ: Message Queue hỗ trợ truyền thông điệp giữa các thành phần mà không đòi hỏi chúng phải chờ đợi nhau. Điều này giúp cải thiện hiệu suất và tăng tính mở rộng của hệ thống.
* Tính độc lập: Producer và consumer không cần biết về sự tồn tại của nhau. Sự phân tách này giúp giảm sự phụ thuộc giữa các thành phần và tăng khả năng mở rộng của hệ thống.
* Xử lý lưu lượng cao: Message Queue có thể xử lý lượng thông điệp lớn và đồng thời từ nhiều nguồn mà không gây ảnh hưởng lớn đến hiệu suất của hệ thống.
* Đảm bảo giao tiếp tin cậy: Hệ thống Message Queue thường có cơ chế đảm bảo rằng việc trao đổi thông tin giữa Producer và Consumer được chính xác và xử lý đúng cách.

**Nhược điểm**

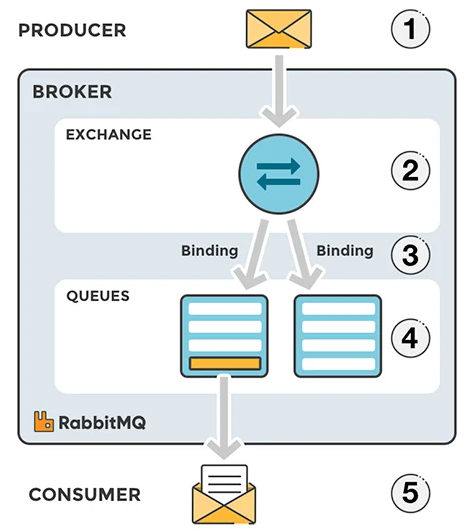
* Phức tạp hóa hệ thống: Sử dụng Message Queue có thể làm phức tạp hóa hệ thống và tốn kém. Đối với các hệ thống nhỏ, đôi khi triển khai Message queue là không cần thiết.

# Rabbitmq

RabbitMQ là một trong những message broker đầu tiên triển khai giao thức AMQP (Advanced Message Queuing Protocol).

AMQP là một giao thức mở (open protocol) truyền thông tiêu chuẩn hóa nhằm mục đích gửi message có độ tin cậy cao và hiệu quả giữa các ứng dụng và message broker. AMQP không chỉ xác định giao thức để giao tiếp với RabbitMQ mà còn xác định mô hình logic phát thảo chức năng cốt lõi của RabbitMQ.

## Hoạt động của RabbitMQ

****

1. Producer gửi một message tới một exchange. Exchange phải được chỉ định type khi được tạo.
2. Exchange nhận được message và chịu trách nhiệm định tuyến message. Việc trao đổi sẽ tính đến các thuộc tính của message và loại exchange, chẳng hạn như routing key.
3. Giữa exchange và queue phải thiết lập bindings từ trước. Exchange định tuyến message vào queue tùy thuộc vào thuộc tính của message. Trong ví dụ trên, có 2 bindings từ exchange với 2 queue khác nhau.
4. Các message vẫn ở trong queue cho đến khi chúng được consumer xử lý.
5. Consumer xử lý message.

## Các thành phần chính trong RabbitMQ

**Message**

Message là đơn vị cơ bản để trao đổi thông tin trong RabbitMQ. Nó bao gồm một body và các property (thuộc tính) tùy chọn khác như header, routing key, …

Message body chứa dữ liệu thực tế cần được gửi, có thể ở nhiều định dạng khác nhau như JSON, XML, binary hoặc bất kỳ cấu trúc nào. Nội dung của message body có thể đại diện cho các lệnh, sự kiện, thông báo hoặc bất kỳ thông tin nào cần trao đổi.

Message property (thuộc tính của tin nhắn) bao gồm các atribute và metadata khác nhau đi kèm với message.

* Routing key (Khoá định tuyến): được exchange dùng để xác định cách định tuyến message tới queue (phụ thuộc vào exchange type).
* Message ID: mã định danh duy nhất xác định message.
* Timestamp: thời gian khi message được tạo.
* Headers (Tiêu đề): Tiêu đề tuỳ chỉnh chứa thông tin bổ sung về message.
* Delivery Mode (Chế độ chuyển phát): chỉ định chế độ chuyển phát, liệu có nên lưu message vào đĩa để duy trì độ bền hay không.
* Expiration (Hạn dùng): thời gian message bị loại bỏ nếu không được sử dụng.
* Priority (Độ ưu tiên): mức độ ưu tiên của message.
* Content type (Loại nội dung): mô tả định dạng của nội dung message (ví dụ: application/json).
* Content encoding (Mã hoá nội dung): chỉ định kiêủ mã hoá được sử dụng cho nội dung message (ví dụ: utf-8).

**Producers**

Producers là một ứng dụng gửi message tới RabbitMQ. Đây là điểm khởi đầu trong workflow của rabbitmq

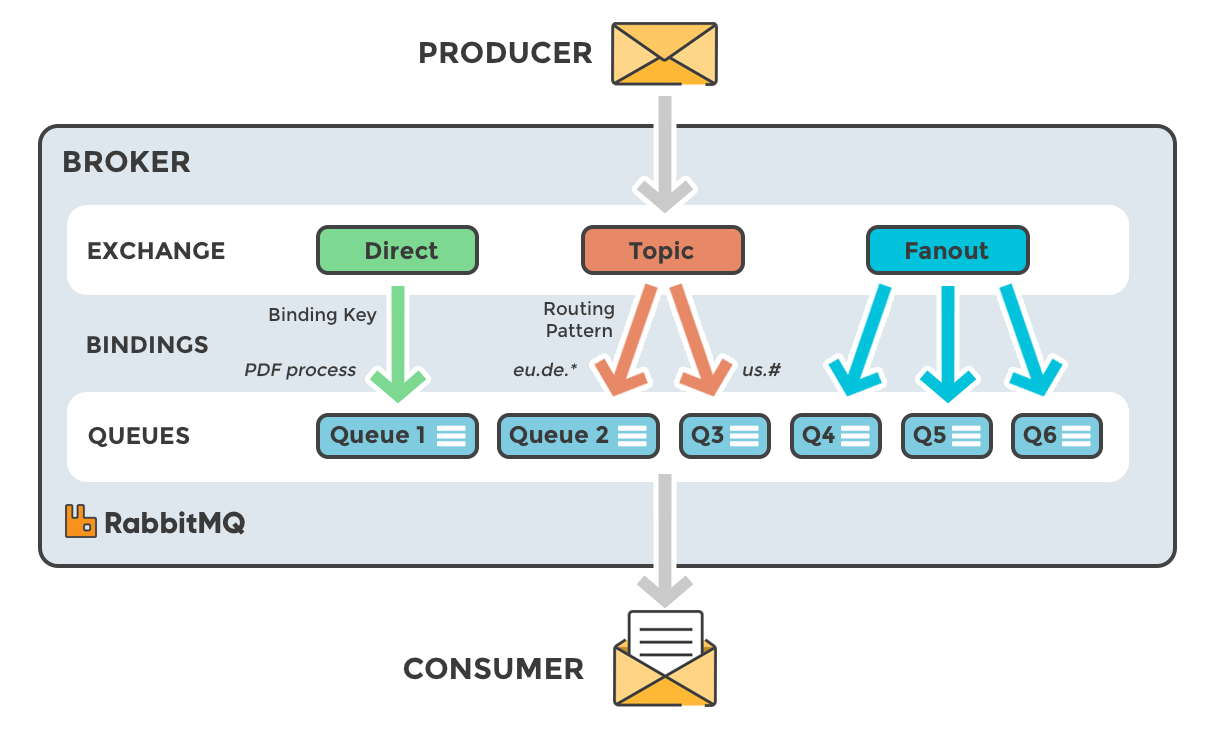
Producer thường mở một hoặc nhiều kết nối TCP tới exchange trong quá trình ứng dụng khởi tạo. Các kết nối này thường tồn tại miễn là kết nối của chúng hoặc ứng dụng đang chạy (long-lived connection).

**Bindings**

Binding là các ràng buộc được thiết lập để xác định mối quan hệ giữa exchange và queue. Chúng chỉ định các quy tắc định tuyến để gửi message từ exchange tới queue thích hợp. Một queue có thể liên kết với nhiều exchange và một exchange có thể có nhiều liên kết tới các queue khác nhau.

**Exchanges**

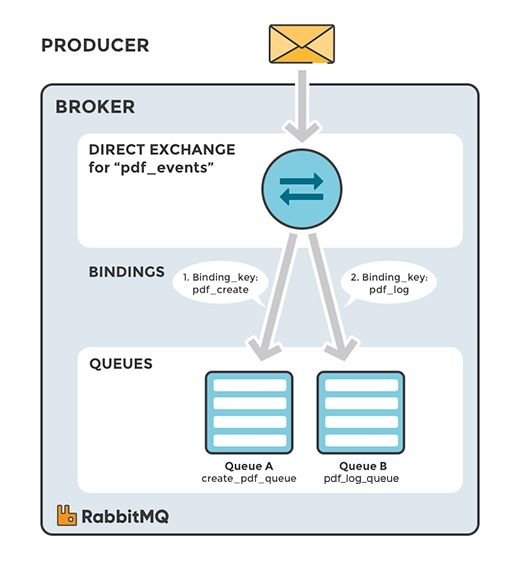
Trong RabbitMQ, exchange (sàn giao dịch) đóng vai trò như một bộ định tuyến message. Khi producer gửi message tới RabbitMQ, message sẽ được gửi tới exchange. Sau đó, exchange sẽ định tuyến message đến queue thích hợp.



RabbitMQ có 5 loại exchange khác nhau dùng để định tuyến message. Mỗi loại exchange sẽ có cách sử dụng tham số và thiết lập binding khác nhau. Client có thể tự tạo exchange hoặc sử dụng exchange mặc định khi server khởi tạo.

**Driect exchange**

Trong direct exchange, message được định tuyến đến queue dựa trên routing key. Nếu routing key của message khớp với routing key của queue, message sẽ được chuyển đến queue đó. Ngược lại, nếu routing key của message không khớp với bất kỳ routing key của queue nào, message sẽ bị loại bỏ.



Exchange: pdf\_events

Queue A: create\_pdf\_queue

Queue B: create\_log\_queue

Binding key giữa exchange và Queue A: pdf\_create

Binding key giữa exchange và Queue B: pdf\_log

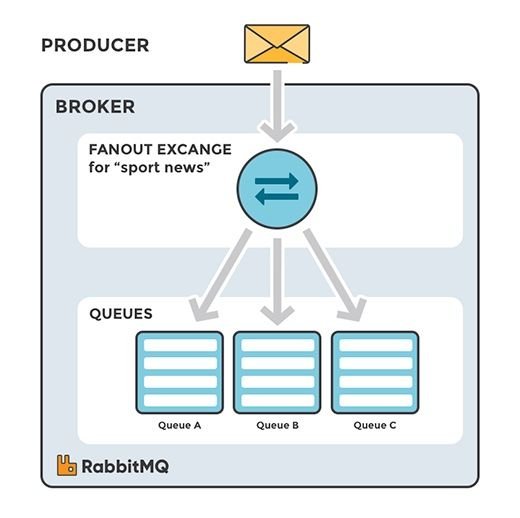
Một message có routing key là pdf\_log được gửi tới exchange pdf\_events. Message này sẽ định tuyến tới queue B pdf\_log\_queue vì routing key của message trùng với binding key pdf\_log.

**Default exchange**

Default exchange là một direct exchange mặc định, được khai báo trước mà không có tên, cho phép định tuyến message trực tiếp tới queue với tên là routing key của message. Mỗi queue đều tự động liên kết với exchange mặc định bằng một routing key giống tên của queue.

**Fanout exchange**

Trong fanout exchange, message được định tuyến đến tất cả các queue được liên kết với nó, bất kể chúng có routing key như nào.



Exchange: sport news

Queue A

Queue B

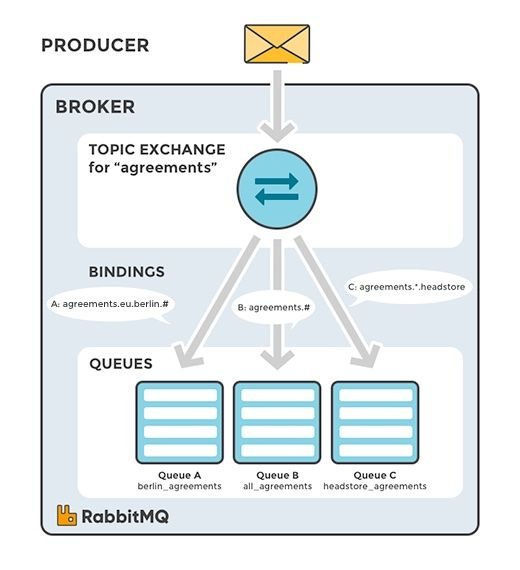
Queue C

Binding: queue A, B, C có cùng binding tới exchange

Message gửi từ exchange sport news sẽ được định tuyến tới cả 3 queue A, B, C bất kể chúng có routing key khác nhau do chúng đều được liên kết với exchange.

**Topic exchange**

Trong topic exchange, message được định tuyến đến queue dựa trên topic, cho phép sử dụng regular expression pattern trong routing key như \* để đại diện cho một ký tự và # đại diện cho không hoặc nhiều ký tự.



Exchange: aggreements

Queue A: berlin\_agreements

Queue B: ali\_agreements

Queue C: headstone\_agreements

Binding key giữa exchange và Queue A: agreements.eu.berlin.#

Binding key giữa exchange và Queue B: agreements.#

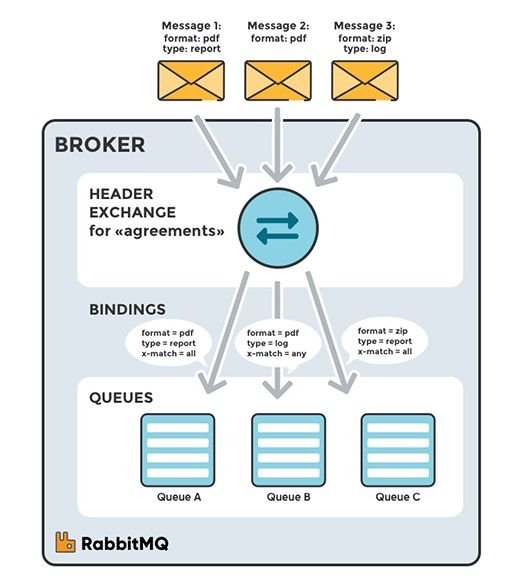
Binding key giữa exchange và Queue C: agreements.\*.headstone

Message có routing key agreements.eu.berlin được gửi tới exchange agreements. Message này sẽ được định tuyến tới queue A berlin\_agreements vì routing pattern khớp với binding key agreements.eu.berlin.#.

Đồng thời, message này cũng được gửi tới queue B vì khớp với binding key agreements.#.

**Header exchange**

Trong header exchange, message được định tuyến dựa trên thuộc tính header của message thay vì routing key.



Message 1: header có dạng key: value là format: pdf, type: report

Message 2: header có dạng key: value là format: pdf

Message 3: header có dạng key: value là format: zip, type: log

Exchange: aggreements

Queue A

Queue B

Queue C

Binding 1 giữa exchange và Queue A: có dạng key: value là format: pdf, type: report, x-match: all

Binding 2 giữa exchange và Queue B: có dạng key: value là format: pdf, type: log, x-match: any

Binding 3 giữa exchange và Queue C: có dạng key: value là format: zip, type: report, x-match: all

Tham số x-match trong binding xác định cách so sánh các cặp key/value của header trong message với header của binding. Có 2 giá trị cho tham số x-match là any và all.

* x-match: any: chỉ cần một thuộc tính trong header của message phải khớp với header của binding.
* x-match: all: tất cả các thuộc tính trong header của message phải khớp với header của binding.

Message 1 sẽ được định tuyến tới queue A vì tất cả cặp key/value trong header của message đều khớp với binding.

Message 1 sẽ được định tuyến tới queue B vì có cặp key/value format: pdf trùng và thuộc tính x-match: any yêu cầu chỉ cần một thuộc tính trong header khớp dù message 1 có type: report không khớp với type: log của binding 1.

Message 2 sẽ được định tuyến tới queue B vì khớp format: pdf.

Message 3 sẽ được định tuyến tới queue B vì type: log.

**Queues**

Trong RabbitMQ, queue là một danh sách các message được lưu trữ cho tới khi chúng được tiêu thụ bởi consumer.

Message được định tuyến tới queue thông qua bindings từ exchange. RabbitMQ cho phép tạo nhiều queue với các cấu hình khác nhau. Mỗi queue có một tên, ngoài ra còn có các thuộc tính tùy chọn khác như độ bền, thời gian hết hạn của message và độ dài tối đa của message.

## Consumers

Consumer là một ứng dụng (hoặc phiên bản ứng dụng) nhận message từ một hay nhiều queue.

Client có thể đóng kết nối tới RabbitMQ. Khi RabbitMQ phát hiện mất kết nối, việc gửi message sẽ dừng lại.

**Chế độ hoặc động**

Khi một message mới đến queue, consumer sẽ tiêu thụ message thông qua 2 cơ chế hoạt động là:

* Push: là chế độ hoạt động mặc định, RabbitMQ sẽ push message đến conumer
* Pull: consumer sẽ chủ động pull message từ queue

**Cơ chế push**

Trong cơ chế push, RabbitMQ push (đẩy) message tới consumer bất cứ khi nào có message mới trong queue. Nhờ vậy, client nhận được dữ liệu real-time hoặc gần real-time với độ trễ thấp và phản ứng với thay đổi một cách tức thì.

Do RabbitMQ là một message broker có mục đích chính là đảm bảo rằng message được gửi đến consumer nhanh chóng và hiệu quả nhất có thể, nên push trở thành chế độ hoạt động mặc định trong RabbitMQ.

Tuy nhiên, vì RabbitMQ kiểm soát tốc độ truyền dữ liệu nên cơ chế push gặp khó khăn trong việc xử lý khi các consumer có tốc độ tiêu thụ khác nhau. Ngoài ra, cơ chế push chỉ hoạt động được khi mà kết nối bidirectional giữa client và server được thiết lập sẵn. Nếu client bị ngắt kết nối khi server push response thì có thể response sẽ bị mất.

**Cơ chế pull**

Trong chế độ pull, consumer chủ động poll message từ queue.

Consumer gửi một yêu cầu Basic.Get tới RabbitMQ khi nó trong trạng thái sẵn sàng xử lý message. RabbitMQ sẽ gửi lại message từ queue nếu được.

Chế độ pull thích hợp dùng cho các trường hợp consumer muốn truy xuất message theo yêu cầu thay vì liên tục được push message.

Chế độ pull có thể kém hiệu quả hơn chế độ push đặc biệt nếu tần suất polling không được điều chỉnh tốt. Nếu consumer thường xuyên polling trong khi không có message nào trong queue thì sẽ dẫn tới lãng phí tài nguyên. Nếu consumer ít polling thì có thể thời gian nhàn rỗi của nó sẽ tăng.

**Delivery acknowledgment (chế độ xác nhận chuyển phát)**

Consumer có thể chọn giữa 2 chế độ xác nhận chuyển phát để queue biết cách xử lý các message đã gửi cho consumer:

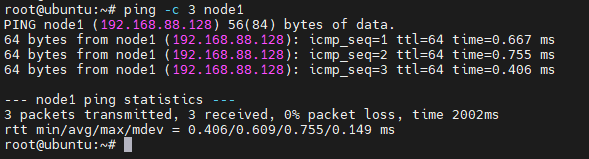
* Automatic: không yêu cầu xác nhận của consumer
* Manual: yêu cầu xác nhận của consumer

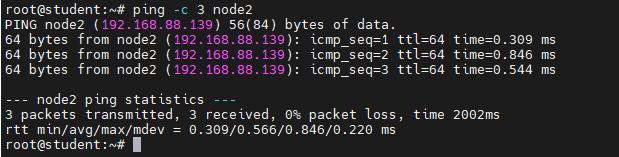
# Cấu hình cluster RabbitMQ

Sử dụng 2 máy ubuntu 18.04 cài đặt rabbitmq để join cluster

Trên cả 2 host thêm thông tin vào file /etc/hosts:





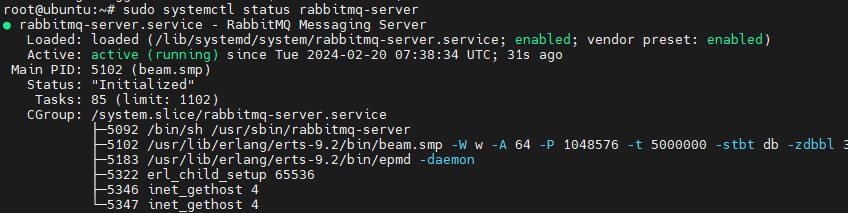


**Cài đặt RabbitMQ cho cả 2 node**

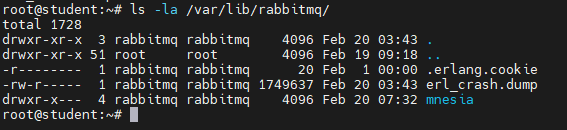
*sudo apt update*

*sudo apt install rabbitmq-server –y*





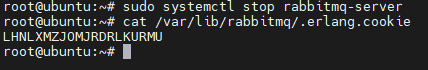
**Cài đặt RabbitMQ Cluster**



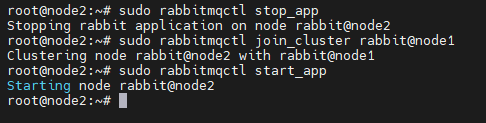
Để tạo rabbitmq cluster, bạn phải sửa file .erlang.cookie trên tất cả các node để giống nhau

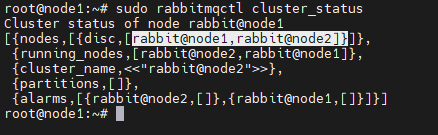


Trên node2, stop rabbitmq và chỉnh sửa file .erlang.cookie



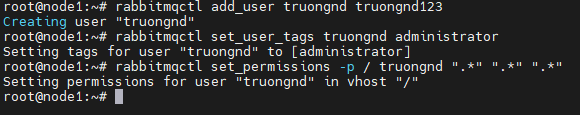
Start lại rabbitmq và sử dụng công cụ rabbitmqctl trên node2 và join vào cluster:





**Thêm Admin user cho RabbitMQ**

Sau khi cài đặt được cluster, bạn cần tạo một user admin và xóa bỏ default guest user. User admin có quyền configure, read, write lên toàn bộ tài nguyên rabbitmq cluster



Cho phép (-p /) truongnd có thể access vào all host trong cluster

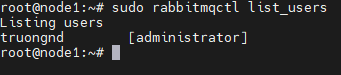
“.\*” – đầu tiên, cho configure mọi thực thể và vhosts

“.\*” – thứ 2, enables write-permission cho mọi thực thể và vhosts

“.\*” – thứ 3, enables read-permission cho mọi thực thể và vhosts

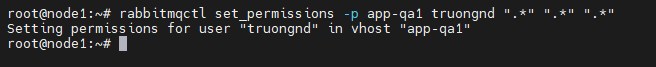
Sau đó xóa bỏ user guest



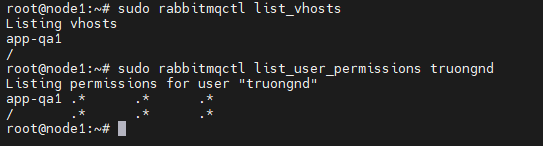


**Tạo Virtual Host và admin user**

Bạn đã thiết lập cluster và tạo admin user. Nhưng bây giờ cần tạo một vhost và user để ứng dụng có thể kết nối tới rabbitmq

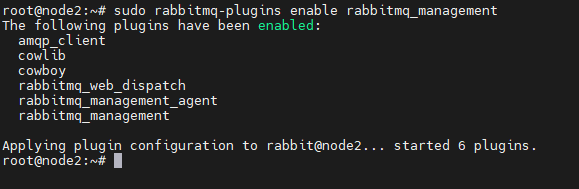


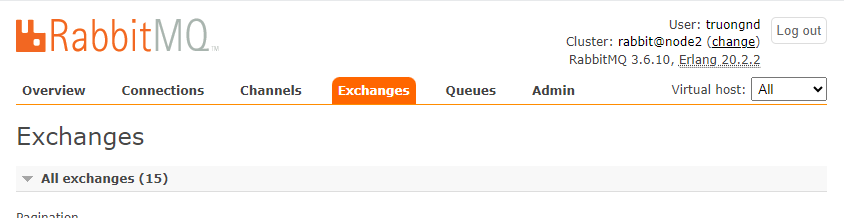




**Tạo exchange, queue, binding**

**Enable the rabbitmq management plugin**

****

****

# Lab triển khai Cluster RabbitMQ bằng docker

## Xây dựng Image

* Dockerfile

|  |
| --- |
| *FROM rabbitmq:3.10-management*  *COPY rabbitmq-qq.conf /etc/rabbitmq/rabbitmq.conf*  *COPY rabbitmq-qq-definitions.json /etc/rabbitmq/rabbitmq-definitions.json*    *#ENV RABBITMQ\_CONF\_ENV\_FILE /etc/rabbitmq/rabbitmq-env.conf*  *ENV RABBITMQ\_ERLANG\_COOKIE cookieSecret*  *#RUN apt-get update && apt-get install -y iputils-ping && apt-get install -y telnet && apt-get install -y nano* |

Lưu ý biến môi trường RABBITMQ\_ERLANG\_COOKIE rất quan trọng. Các node rabbitmq muốn giao tiếp được với nhau thì giá trị cookie này cần phải giống nhau thì mới có thể authen được. (Nếu không set giá trị này, thì giá trị erlang.cookie sẽ được sinh ngẫu nhiên, và khác nhau trên mỗi node).

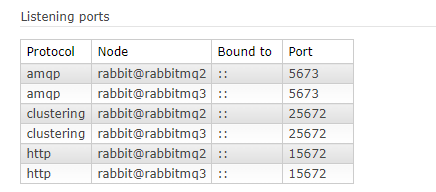
* File cấu hình rabbitmq-qq.conf

|  |
| --- |
| *loopback\_users.guest = false*  *listeners.tcp.default = 5672*  *management.listener.port = 15672*  *management.listener.ssl = false*  *vm\_memory\_high\_watermark.absolute = 1536MB*  *cluster\_name = rabbitmq-qq*  *cluster\_formation.peer\_discovery\_backend = rabbit\_peer\_discovery\_classic\_config*  *cluster\_formation.classic\_config.nodes.1 = rabbit@rabbitmq1*  *cluster\_formation.classic\_config.nodes.2 = rabbit@rabbitmq2*  *cluster\_formation.classic\_config.nodes.3 = rabbit@rabbitmq3*  *management.load\_definitions = /etc/rabbitmq/rabbitmq-definitions.json*  *# background\_gc\_enabled = true*  *# Increase the 5s default so that we are below Prometheus' scrape interval,*  *# but still refresh in time for Prometheus scrape*  *# This is linked to Prometheus scrape interval & range used with rate()*  *collect\_statistics\_interval = 10000*  *# Enable debugging*  *log.file = rabbit.log*  *log.dir = /var/log/rabbitmq*  *log.console.level = info*  *cluster\_partition\_handling = pause\_minority* |

listeners.tcp.default=5672. port để client kết nối vào broker. Mặc định port này là 5672. Có thể đổi sang port khác nếu conflict

management.listener.port = 15672. port để vào webadmin gui

Có 1 port là epmd không để trong file config. Mặc định port này là 4369. Port này rất quan trọng, các node dùng port này để discovery nhau. Bắt buộc số port phải là giống nhau trên các node



[Networking and RabbitMQ — RabbitMQ](https://www.rabbitmq.com/networking.html#ports)

rabbitmq1, rabbitmq2, rabbitmq3 lần lượt là hostname của 3 node

cluster\_partition\_handling=pause\_minority: khi có sự cố networking partition, rabbitmq cung cấp 3 policy để handler, là pause-minority, pause-if-all-down, autoheal. Nếu bạn không khai báo cấu hình này thì mặc định nó sẽ ignore, không làm gì cả. Với mode pause-minority thì bên phía các node có số lượng ít hơn, sẽ bị down luôn. Các message, queue sẽ được gửi về bên phía có số lượng node nhiều hơn.

* File rabbitmq-qq-definitions.json

|  |
| --- |
| {  "global\_parameters": [  {"name": "cluster\_name", "value": "rabbitmq-qq"}  ],  "permissions": [  {  "configure": ".\*",  "read": ".\*",  "user": "truongnd",  "vhost": "/",  "write": ".\*"  }  ],  "users": [  {  "name": "truongnd",  "password": "truong123",  "tags": "administrator"  }  ],  "vhosts": [{"name": "/"}]  } |

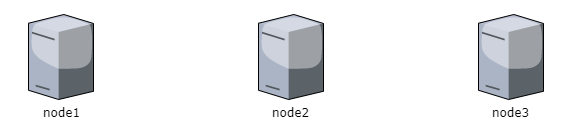
Cấu hình này hiện tại chỉ dùng đẻ define account đăng nhập

* Có thể build docker image bằng command:

*docker build -t rabbitmq\_ha\_qq -f Dockerfile .*

## Xây dựng cluster bằng docker-compose

Sơ đồ xây dựng cluster:

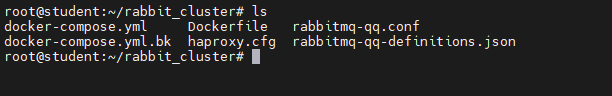


Docker-compose.yml

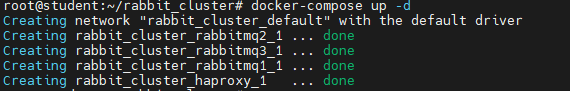
|  |
| --- |
| version: '3.7'  volumes:  rabbitmq\_volume:  rabbitmq\_log:  services:  rabbitmq1:  build: .  hostname: rabbitmq1  volumes:  - rabbitmq\_volume:/var/lib/rabbitmq  - rabbitmq\_log:/var/log/rabbitmq  rabbitmq2:  build: .  # cap\_add:  # - ALL  hostname: rabbitmq2  volumes:  - rabbitmq\_volume:/var/lib/rabbitmq  - rabbitmq\_log:/var/log/rabbitmq  rabbitmq3:  build: .  hostname: rabbitmq3  volumes:  - rabbitmq\_volume:/var/lib/rabbitmq  - rabbitmq\_log:/var/log/rabbitmq  haproxy:  image: haproxy:1.7  volumes:  - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro  depends\_on:  - rabbitmq1  - rabbitmq2  - rabbitmq3  ports:  - 15672:15672  - 5672:5672  - 1936:1936  - 5673:5673 |

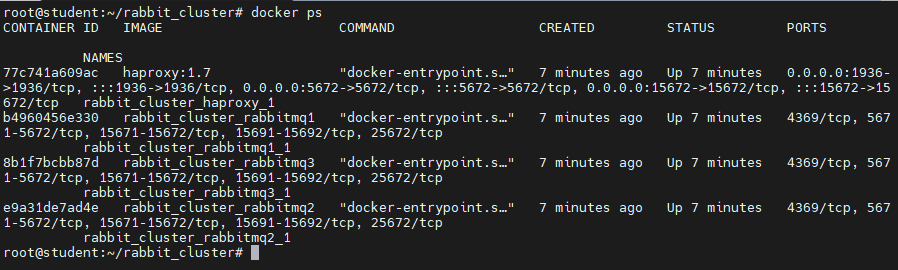
Sử dụng haproxy làm loadbalancer đứng trước 3 node rabbitmq. Cấu hình file haproxy.cfg:

|  |
| --- |
| global  log 127.0.0.1 local1  maxconn 4096  defaults  log global  mode tcp  option tcplog  retries 3  option redispatch  maxconn 2000  timeout connect 5000  timeout client 50000  timeout server 50000  listen stats  bind \*:1936  mode http  stats enable  stats hide-version  stats realm Haproxy\ Statistics  stats uri /  listen rabbitmq  bind \*:5672  mode tcp  balance roundrobin  timeout client 3h  timeout server 3h  option clitcpka  server rabbitmq1 rabbitmq1:5672 check inter 5s rise 2 fall 3  server rabbitmq2 rabbitmq2:5672 check inter 5s rise 2 fall 3  server rabbitmq3 rabbitmq3:5672 check inter 5s rise 2 fall 3  listen mgmt  bind \*:15672  mode tcp  balance roundrobin  timeout client 3h  timeout server 3h  option clitcpka  server rabbitmq1 rabbitmq1:15672 check inter 5s rise 2 fall 3  server rabbitmq2 rabbitmq2:15672 check inter 5s rise 2 fall 3  server rabbitmq3 rabbitmq3:15672 check inter 5s rise 2 fall 3 |

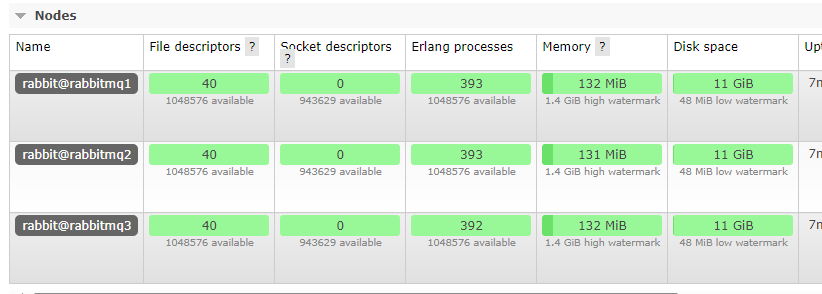
\

Run cluster:



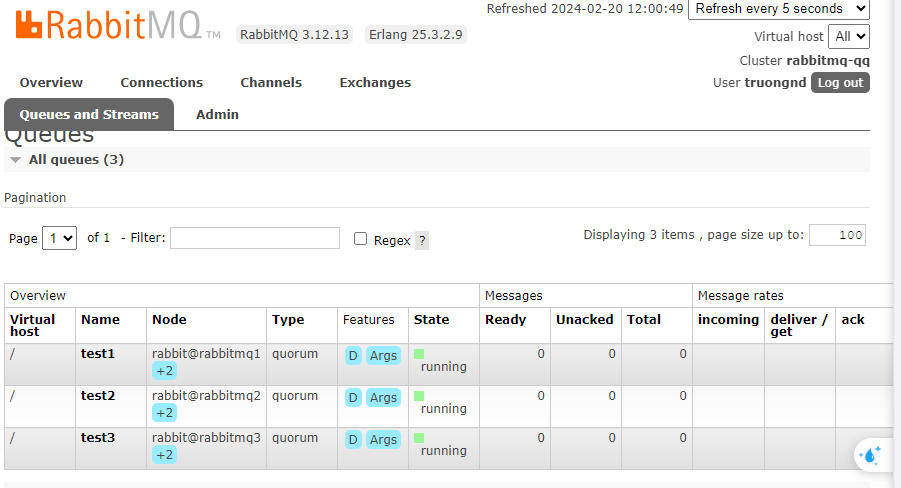


Truy cập vào rabbitmq: <http://IP:15672>, thông tin các node trong cluster được show trong tab Overview



**Tạo Quorum Queue**

Vào tab Queues để tạo queue, và trải nghiệm bật tắt các node. Để test việc Hight Avalibility của queue

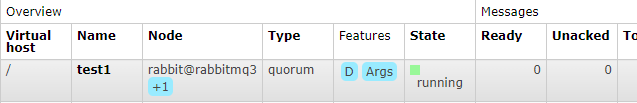


Lưu ý: chọn bất kỳ 1 node để làm leader cho queue. (không quan trọng, sau này có sự cố tự động cluster sẽ bầu lại leader mới)

## Lab một số kịch bản

**Có node bị down và reup**

Ban đầu queue test1 có node master là rabbit@rabbitmq1, sau đó stop container trên node 1 thì node master được chuyển sang node2 hoặc node3 mà message không bị mất



**Network partition**

Kịch bản này có thể xảy ra khi network giữa node3 và 2 node còn lại không thông nhau.

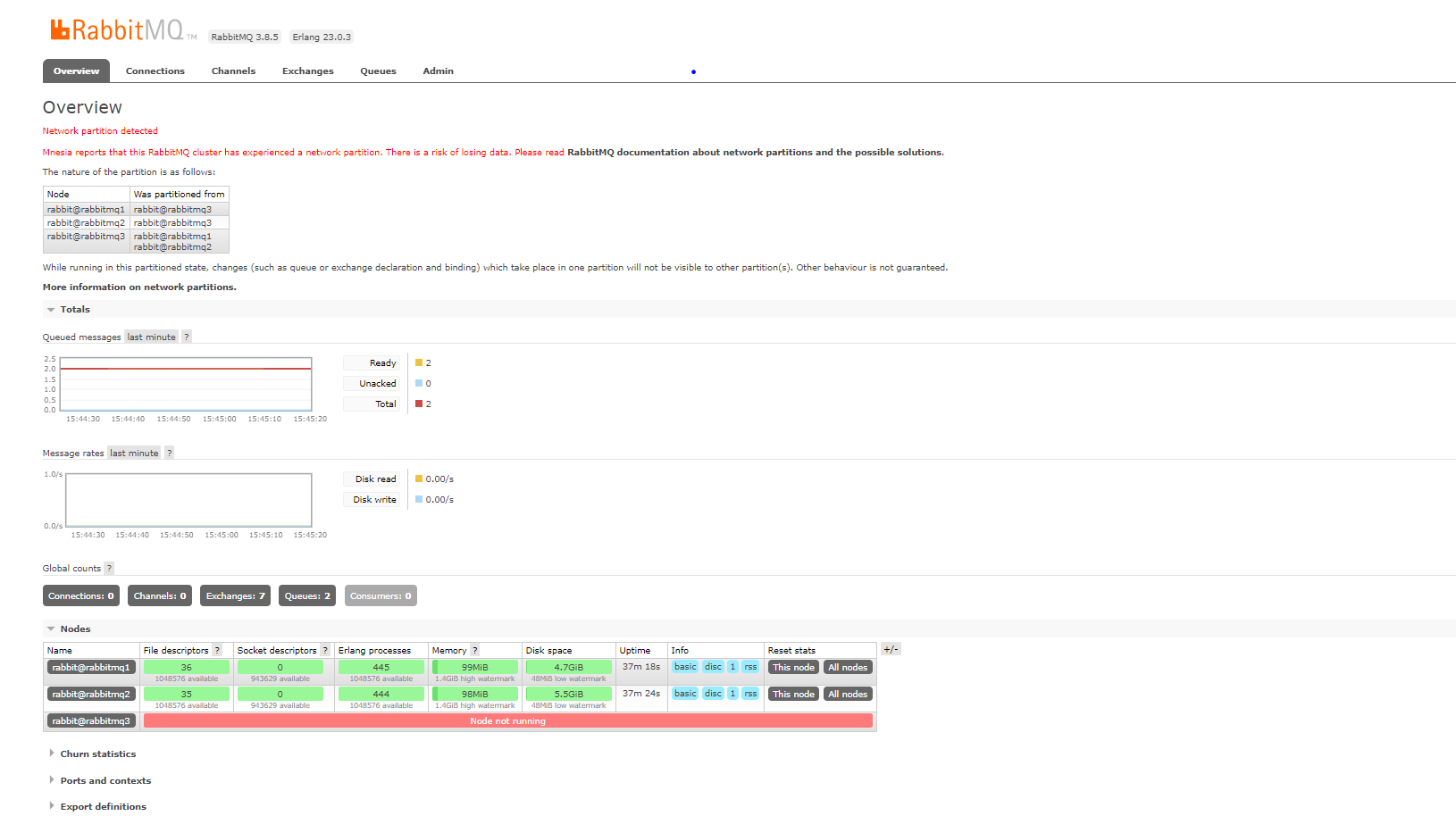
* Nếu trong file rabbitmq-qq.conf không cấu hình cluster\_partition\_handling = pause\_minority thì kịch bản sau sẽ diễn ra:

Node3 nghĩ rằng 2 node kia down và tự nó lên làm leader của cluster

Cụm node1+node2 nghĩ rằng node3 down, 2 node này tự bầu leader

* Vấn đề này gói là split-brain. Việc này dẫn tới khi clident tạo queue mới hoặc ghi message vào queue trên node3 thì sẽ không có đồng bộ data với node1+node2 và ngược lại.

Lúc này cả 2 phe node3, và node1+node2 đều nghĩ rằng bên kia down. Chứ chưa phát hiện ra sự cố network partition. Chỉ tới khi chúng ta cho thông lại network giữa 3 node với nhau. Lúc này cluster mới phát hiện được.



Hướng xử lý trong tình huống này là chọn 1 bên làm chuẩn, sau đó restart lại rabbitmq bên còn lại để rabbitmq bên còn lại rejoin lại cluster và đồng bộ lại message từ bên chuẩn sang.

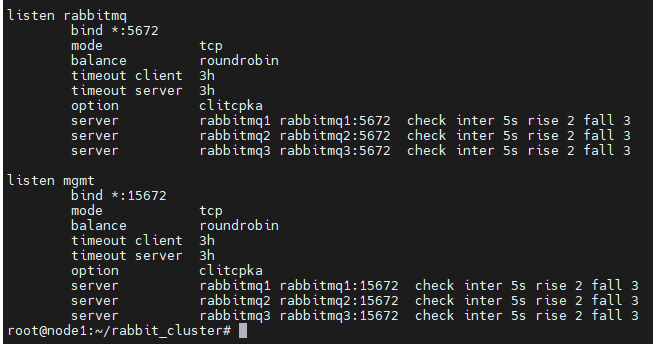
* Nếu trong file rabbitmq-qq.conf mình CÓ cấu hình cluster\_partition\_handling = pause\_minority thì kịch bản sau sẽ diễn ra:

node3 thấy nó chỉ có mình nó. Cả cluster khai báo 3 node. Vì 1 bé hơn 2. Nên rabbitmq trên node3 cho down luôn. Còn cụm node1+node2 vẫn chạy bình thường. (lưu ý là rabbitmq bị shutdown, chứ container vẫn chạy bình thường. Có thể kiểm tra bằng cách sử dụng rabbitmqctl)

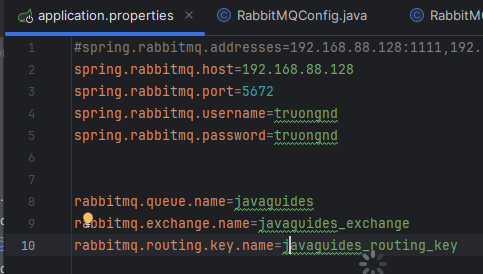
Lúc này việc route từ client sẽ được chuyển về cụm node1+node2.

## Test connection

Sử dụng Spring boot tạo kết nối tới rabbitMQ. Với cấu hình như trên chúng ta sẽ kết nối thông qua HAProxy

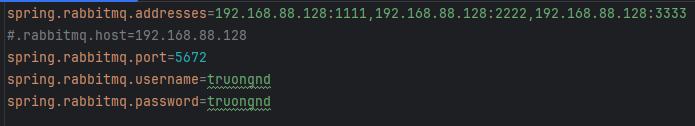


Cấu hình springboot



Thực hiện tạo connection, tại đây springboot sẽ kết nối thông qua HAProxy và việc kiểm tra sự sống của các node rabbitmq là công việc của HAProxy.

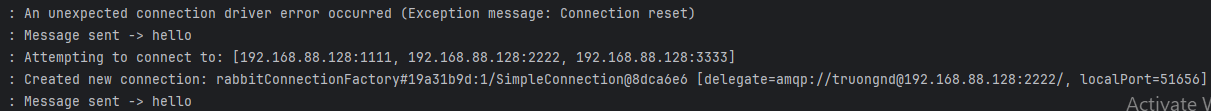
**TH2:** Springboot kết nối trực tiếp tới 3 node rabbitmq



Tại đây springboot sẽ tự động kiểm tra xem node nào còn sống và nếu node đang tạo kết nối bị đứt thì sẽ tự động chuyển sang node khác.



Hiện tại đang kết nối với node rabbitmq1. Thực hiện stop container rabbitmq1 đi và gửi lại message



Lúc này Spring boot đã tự động chuyển kết nối tới node2