|  |
| --- |
|  |
| NodeRED with RabbitMQ |
| Implementing RabbitMQ as a message broker for NodeRED |

|  |
| --- |
| Tran Nguyen Hoang Huy  8-7-2025 |

# Mục lục

[Mục lục 1](#_Toc206138106)

[1. Giới thiệu 2](#_Toc206138107)

[1.1. Mục tiêu nghiên cứu 2](#_Toc206138108)

[1.2. Phạm vi nghiên cứu 2](#_Toc206138109)

[1.3. Phương pháp nghiên cứu 2](#_Toc206138110)

[1.4. Công cụ nghiên cứu 2](#_Toc206138111)

[2. Hệ thống RabbitMQ 3](#_Toc206138112)

[2.1. Giới thiệu 3](#_Toc206138113)

[2.2. Overivew phương pháp 3](#_Toc206138114)

[2.2.1. Trường hợp 1: NodeRED offline trước khi message đến 4](#_Toc206138115)

[2.2.2. Trường hợp 2: NodeRED ngắt kết nối khi đang xử lý message 4](#_Toc206138116)

[5. Kết luận 4](#_Toc206138117)

[Reference 5](#_Toc206138118)

# 1. Giới thiệu

## 1.1. Mục tiêu nghiên cứu

Nghiên cứu này tập trung vào việc tìm hiểu cách tích hợp RabbitMQ trong hệ thống kết nối giữa Spring Boot và Node-RED. RabbitMQ đóng vai trò là message broker giúp truyền dữ liệu bất đồng bộ giữa các dịch vụ. Báo cáo sẽ mô tả tổng quan RabbitMQ, cơ chế hoạt động, và triển khai thực tế khi Spring Boot là nguồn phát (publisher) hoặc xử lý dữ liệu (consumer), còn Node-RED đóng vai trò là một thành phần giao tiếp hoặc xử lý dữ liệu trung gian.

## 1.2. Phạm vi nghiên cứu

**Phạm vi nghiên cứu bao gồm:**

* Khái niệm và cơ chế hoạt động của RabbitMQ.
* Cách Spring Boot gửi và nhận message từ RabbitMQ.
* Cách Node-RED tích hợp RabbitMQ qua node AMQP/MQTT.

Tình huống thực tế: Spring Boot gửi thông báo sự kiện, Node-RED xử lý luồng dữ liệu và gửi phản hồi.

## 1.3. Phương pháp nghiên cứu

Nghiên cứu sử dụng phương pháp triển khai thực nghiệm và phân tích. Tham khảo tài liệu chính thức của RabbitMQ, Spring AMQP, và Node-RED AMQP node. Xây dựng một mô hình thử nghiệm với:

* Một ứng dụng Spring Boot đóng vai trò publisher.
* Một flow Node-RED kết nối RabbitMQ để xử lý hoặc chuyển tiếp thông tin.

## 1.4. Công cụ nghiên cứu

* RabbitMQ Server
* Spring Boot (Spring AMQP)
* Node-RED
* Postman
* Visual Studio Code
* Docker

# 2. Hệ thống RabbitMQ

## 2.1. Giới thiệu

RabbitMQ là một message broker mã nguồn mở, hỗ trợ nhiều giao thức nhắn tin như AMQP (Advanced Message Queuing Protocol). Nhiệm vụ chính của RabbitMQ là nhận, lưu trữ tạm thời và phân phối thông điệp giữa các thành phần của hệ thống.

* **Exchange:** Nhận message từ producer và phân phối đến hàng đợi (queue) dựa vào routing key hoặc pattern.
* **Queue:** Nơi lưu trữ tạm thời message cho đến khi consumer nhận.
* **Binding:** Quy tắc kết nối giữa exchange và queue.
* **Routing key:** Khóa định tuyến xác định message sẽ được gửi đến queue nào.

RabbitMQ phù hợp cho các hệ thống cần giao tiếp bất đồng bộ, giảm tải cho API trực tiếp, và xử lý dữ liệu phân tán.

## 2.2. Kiến trúc và thành phần chi tiết

* **Producer**

Là thành phần đóng vai trò gửi message vào RabbitMQ. Producer không gửi trực tiếp message đến queue mà gửi đến Exchange kèm theo Routing key.  
Ví dụ: Một service cảnh báo trong hệ thống gửi thông tin “Alert” đến RabbitMQ để các service khác xử lý.

* **Exchange**

Nhận message từ Producer và định tuyến đến một hoặc nhiều queue dựa vào loại exchange và routing key. Các loại exchange phổ biến:

* **Direct Exchange:** Gửi message đến queue có routing key trùng khớp chính xác.
* **Topic Exchange:** Gửi message đến queue có routing key khớp với pattern (\* đại diện cho 1 từ, # đại diện cho nhiều từ).
* **Fanout Exchange:** Gửi message đến tất cả queue được bind vào exchange, bỏ qua routing key.
* **Headers Exchange:** Định tuyến dựa trên header của message thay vì routing key.
* **Queue**

Là nơi lưu trữ tạm thời message cho đến khi consumer xử lý. Một số đặc điểm cấu hình:

* **Durable**: Queue vẫn tồn tại khi RabbitMQ restart.
* **Exclusive**: Queue chỉ được dùng bởi một kết nối duy nhất.
* **Auto-delete**: Queue sẽ tự xóa khi không còn consumer nào kết nối.
* **Binding & Routing key**

**Binding** là quy tắc kết nối giữa exchange và queue.

**Routing key** là chuỗi ký tự dùng để xác định message sẽ được gửi đến queue nào.  
Ví dụ: Queue alertQueue bind với exchange alertExchange bằng routing key alertRoutingKey → chỉ message có routing key này mới vào queue.

* **Consumer**

Là thành phần nhận và xử lý message từ queue. Có thể có nhiều consumer cùng đọc từ một queue để tăng khả năng xử lý song song. Consumer sẽ gửi một **acknowledge (ack)** về RabbitMQ khi xử lý xong.

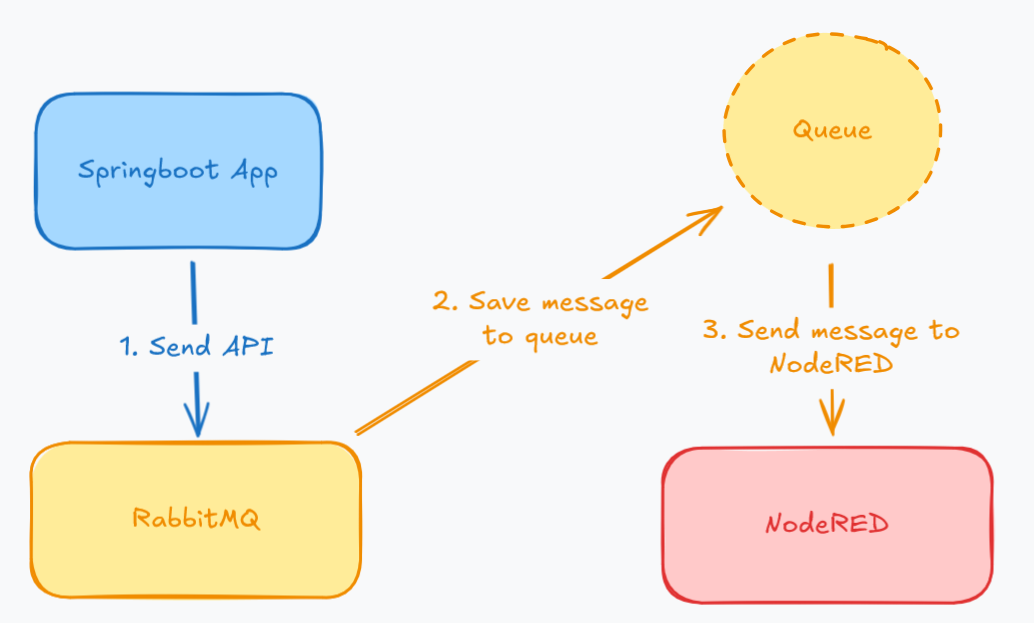
## 2.3. Cơ chế hoạt động của RabbitMQ

* **Chu trình từ Producer → Exchange → Queue → Consumer**

1. **Producer** gửi message kèm routing key đến Exchange.
2. **Exchange** dựa vào quy tắc Binding và Routing key để gửi message đến queue phù hợp.
3. **Queue** lưu message cho đến khi consumer nhận.
4. **Consumer** lấy message từ queue, xử lý và gửi ack/nack.

* **Ví dụ:** Hệ thống Alert gửi message ở dạng JSON với routing key **alertRoutingKey** → **Exchange** gửi message này đến alertQueue → **Consumer** (NodeRED) nhận message và gửi thông báo đến người dùng.
* **Cơ chế ack, nack, requeue**
  + **ack**: Consumer báo RabbitMQ là đã xử lý xong message → RabbitMQ xóa message khỏi queue.
  + **nack**: Consumer báo lỗi khi xử lý message.
    - Nếu **requeue = true** → message quay lại queue để retry.
    - Nếu **requeue = false** → message bị gửi sang DLQ (nếu cấu hình).

## 2.4. Overivew phương pháp



**Luồng xử lý:**

1. App gửi API đến RabbitMQ
2. Các API được lưu thành một message vào queue
3. Theo thứ tự, các message sẽ được gửi đến NodeRED

Ngoài ra sẽ có các trường hợp được xử lý.

### 2.4.1. Trường hợp 1: NodeRED offline trước khi message đến

* Nếu RabbitMQ không kết nối đến NodeRED được thì tất cả các **message** đã gửi đến sẽ được lưu vào queue.

### 2.4.2. Trường hợp 2: NodeRED ngắt kết nối khi đang xử lý message

* Trong quá trình NodeRED đang xử lý **message,** nếu Rabbit thấy bị mất kết nối với NodeRED thì sẽ store lại **message** được gửi về trạng thái **Ready**.
* Vì mỗi **message** sẽ cần một **acknowledgement** để biết được trạng của nó, thông tin **acknowledgement** này do NodeRED gửi về sau mỗi lần xử lý **flow** thành công.

# 3. Kết luận

RabbitMQ đóng vai trò là “trung tâm phân phối” thông điệp, đảm bảo quá trình giao tiếp giữa các thành phần hệ thống được **bất đồng bộ, đáng tin cậy và dễ mở rộng**.  
Với kiến trúc gồm **Producer – Exchange – Queue – Consumer**, RabbitMQ giúp:

* Giảm tải cho API trực tiếp bằng cách tách biệt quá trình gửi và xử lý dữ liệu.
* Đảm bảo không mất dữ liệu ngay cả khi một thành phần (như NodeRED) bị gián đoạn hoặc mất kết nối.
* Hỗ trợ nhiều kịch bản định tuyến khác nhau thông qua các loại Exchange.
* Dễ dàng mở rộng xử lý song song bằng nhiều Consumer.

Tóm lại, RabbitMQ là giải pháp phù hợp cho các hệ thống cần **độ tin cậy cao, khả năng chịu lỗi và khả năng mở rộng linh hoạt**, đặc biệt trong các bài toán xử lý thông tin theo hàng đợi như hệ thống cảnh báo, xử lý sự kiện, và tích hợp giữa nhiều dịch vụ.

# Reference

[Get RabbitMQ on Docker](https://hub.docker.com/_/rabbitmq)

[RabbitMQ Springboot config](https://www.geeksforgeeks.org/springboot/spring-boot-rabbitmq-configuration/)

[RabbitMQ with Springboot](https://spring.io/guides/gs/messaging-rabbitmq)

[RabbitMQ node for NodeRED](https://flows.nodered.org/node/@mnn-o/node-red-rabbitmq)

[Project Repo](https://github.com/shiirouK/alert-nodeRed-demo)

[RabbitMQ docs](https://www.rabbitmq.com/tutorials/tutorial-one-java)