# Mục lục

[1. Giới thiệu 1](#_Toc204180469)

[1.1. Mục tiêu nghiên cứu 1](#_Toc204180470)

[1.2. Phạm vi nghiên cứu 1](#_Toc204180471)

[1.3. Phương pháp nghiên cứu 1](#_Toc204180472)

[1.4. Công cụ nghiên cứu 1](#_Toc204180473)

[2. Hệ thống NodeRED 2](#_Toc204180474)

[2.1. Giới thiệu NodeRED 2](#_Toc204180475)

[2.2. Node trong NodeRED là gì? 2](#_Toc204180476)

[2.3. Nguyên lý hoạt động của một flow trong NodeRED 3](#_Toc204180477)

[2.4. HTTP Nodes 3](#_Toc204180478)

[2.4.1. HTTP In 3](#_Toc204180479)

[2.4.2. HTTP Response 4](#_Toc204180480)

[2.4.3. HTTP Request 4](#_Toc204180481)

[2.5. Các cách gửi email 4](#_Toc204180482)

[2.5.1. Sử dụng Email node 4](#_Toc204180483)

[2.5.2. Sử dụng API của SendGrid 4](#_Toc204180484)

[3. Ứng dụng vào Alert 5](#_Toc204180485)

[3.1. Input 6](#_Toc204180486)

[3.2. Extract 6](#_Toc204180487)

[3.3. Conditions 7](#_Toc204180488)

[3.4. Format Alert 7](#_Toc204180489)

[4. So sánh giữa Zendesk Triggers và NodeRED 8](#_Toc204180490)

[4.1. Input 8](#_Toc204180491)

[4.2. Conditions 8](#_Toc204180492)

[5. Kết luận 9](#_Toc204180493)

# 1. Giới thiệu

## 1.1. Mục tiêu nghiên cứu

Nghiên cứu này tập trung vào việc phân tích và mô tả chuyên sâu cách hệ thống NodeRED hoạt động, đặc biệt trong bối cảnh thiết kế các quy trình tự động hóa và cảnh báo. Mục tiêu chính là hiểu rõ các nguyên tắc thiết kế luồng (flow-based programming), cách truyền dữ liệu giữa các node, và cách triển khai logic xử lý để từ đó ứng dụng vào việc xây dựng một hệ thống Alert.

## 1.2. Phạm vi nghiên cứu

Phạm vi nghiên cứu bao gồm:

* Cơ chế truyền thông tin trong NodeRED: Tìm hiểu cách dữ liệu được truyền qua các node và cách xử lý tin nhắn (msg) giữa các bước.
* Cấu trúc và nguyên lý vận hành của luồng (flow): Phân tích cách xây dựng, thiết kế và triển khai các luồng tự động hóa bằng NodeRED.
* Cách sử dụng các node chức năng (function node, switch, change, delay...): Mô tả cách kết hợp các node để tạo nên một logic cảnh báo phức tạp.
* Tích hợp cơ sở dữ liệu và dịch vụ bên ngoài: Khảo sát cách NodeRED kết nối dịch vụ email hoặc HTTP để thực hiện các hành động tự động.

## 1.3. Phương pháp nghiên cứu

Nghiên cứu sử dụng phương pháp thực hành triển khai và phân tích hệ thống. Tham khảo tài liệu chính thức của NodeRED, xây dựng một mô hình thử nghiệm gồm các flow trong NodeRED, mô phỏng tình huống cảnh báo thực tế, từ đó phân tích luồng xử lý và các quyết định thiết kế.

## 1.4. Công cụ nghiên cứu

* NodeRED
* MySQL
* SendGrid
* Postman
* ChatGPT
* Google
* Visual Studio Code

# 2. Hệ thống NodeRED

## 2.1. Giới thiệu NodeRED

**NodeRED** là một nền tảng lập trình trực quan mã nguồn mở, được phát triển bởi IBM, nhằm hỗ trợ thiết kế các ứng dụng tích hợp dữ liệu và điều khiển thiết bị theo mô hình lập trình dòng chảy (flow-based programming). Với giao diện kéo-thả, NodeRED cho phép người dùng dễ dàng tạo ra các luồng xử lý dữ liệu (flows) mà không cần phải viết nhiều mã lập trình.

NodeRED thường được sử dụng trong các hệ thống IoT, tự động hóa quy trình, tích hợp API, cảnh báo, và xử lý dữ liệu thời gian thực. Một số khả năng chính của NodeRED bao gồm:

* **Thu thập dữ liệu** từ nhiều nguồn khác nhau (HTTP, MQTT, WebSocket, cơ sở dữ liệu, v.v.)
* **Xử lý dữ liệu** qua các node chức năng (function, switch, change, delay, v.v.)
* **Tự động hóa hành động** dựa trên điều kiện logic
* **Tích hợp hệ thống bên ngoài** như email, SMS, cơ sở dữ liệu, REST API
* **Triển khai nhanh chóng** qua giao diện web mà không cần IDE

## 2.2. Node trong NodeRED là gì?

Trong NodeRED, **node** là các khối chức năng có vai trò xử lý dữ liệu. Mỗi node có một mục đích cụ thể, ví dụ như:

* **Inject node**: gửi dữ liệu đầu vào khởi tạo luồng
* **Function node**: viết mã JavaScript tùy chỉnh để xử lý logic
* **Switch node**: phân nhánh logic dựa trên điều kiện
* **MySQL node**: tương tác với cơ sở dữ liệu
* **Email node**: gửi email tự động
* **HTTP nodes**: nhận/gửi dữ liệu qua các API
* …

Một **flow** là tập hợp của nhiều node được kết nối với nhau theo thứ tự để tạo thành quy trình xử lý từ đầu vào đến đầu ra.

## 2.3. Nguyên lý hoạt động của một flow trong NodeRED

NodeRED vận hành theo mô hình **lập trình theo luồng dữ liệu (flow-based programming)**, trong đó dữ liệu được truyền từ node này sang node khác qua một đối tượng JavaScript gọi là msg. Thành phần chính trong **msg** **là msg.payload**, chứa dữ liệu được xử lý xuyên suốt flow.

**Cấu trúc một flow gồm 3 giai đoạn chính:**

* **Trigger**
  + Một **node** như inject, http in hoặc mqtt in tạo ra một đối tượng msg và bắt đầu luồng xử lý.
* **Processing: msg** được truyền qua nhiều node trung gian:
  + **function**: Viết logic tùy chỉnh bằng JavaScript.
  + **switch**: Điều hướng dựa theo điều kiện.
  + **change**: Thay đổi giá trị của trường trong msg.
  + **template**: Chuyển đổi định dạng (HTML, JSON, văn bản...).
* **Action:** Luồng được kết thúc bằng một hành động cụ thể:
  + **http response**: Trả phản hồi về client.
  + **mysql**: Ghi dữ liệu vào cơ sở dữ liệu.
  + **email**: Gửi thông báo.
  + **debug**: Hiển thị thông tin để kiểm tra.

NodeRED có các vùng lưu trữ dữ liệu dùng chung như:

* **Context:** Dữ liệu chỉ được dùng chung trong một node
* **Flow:** Dữ liệu chỉ được dùng chung trong một flow
* **Global:** Dữ liệu dùng chung trên cả hệ thống

## 2.4. HTTP Nodes

NodeRED cung cấp các node HTTP có khả năng giao tiếp với các hệ thống bên ngoài thông qua giao thức HTTP. Các node này thường được sử dụng để:

* Tạo API nhận/gửi dữ liệu từ hệ thống khác
* Nhận dữ liệu từ ứng dụng bên ngoài như Spring Boot, IoT device
* Trả phản hồi về cho client hoặc gửi dữ liệu sang API khác

### 2.4.1. HTTP In

Vai trò của nó là nhận các request từ client bên ngoài qua các giao thức thông thường (POST, GET, PUT,…). Node này chỉ có input, người dùng có thể cấu hình các giao thức này cùng với endpoint của client bên ngoài. Một khi nhận được request, node sẽ bắt đầu luồng.

### 2.4.2. HTTP Response

Nếu có In thì sẽ phải có out, vai trò của node này là gửi về một response cho client, vì vậy đây là một node output bắt buộc phải có nếu luồng bắt đầu với node **HTTP In**. Dữ liệu trả về có thể được cấu hình qua payload và cũng sẽ mặc định trả về một HTTP status code tương ứng với trạng thái của request.

### 2.4.3. HTTP Request

So với 2 node trên thì node **HTTP Request** sẽ linh hoạt hơn, nó vừa có input và output cho phép người dùng cấu hình theo mong muốn. Thông thường thì node này sẽ đi kèm với các **node Function**, vừa làm input để cấu hình endpoint, gửi và nhận dữ liệu, và một **node Function** ở sau để trích xuất và xử lý dữ liệu.

## 2.5. Các cách gửi email

Với NodeRED, người dùng có 2 cách gửi email chính là:

* Sử dụng SMTP thông qua Email node
* Sử dụng HTTP Request để gọi API của các dịch vụ gửi mail như SendGrid

### 2.5.1. Sử dụng Email node

Đối với phương thức này, đặc biệt là Gmail, người dùng cần phải cung cấp thông tin đăng nhập cho Email node để nó gửi mail, với mật khẩu ở đây là một app password, do Google cung cấp ở <https://myaccount.google.com/apppasswords>. Các thuộc tính của email như subject, body hoặc attachment có thể được cấu hình qua **msg** như:

* **msg.topic:** email subject
* **msg.payload/msg.description:** email body
* **msg.attachments:** file đính kèm

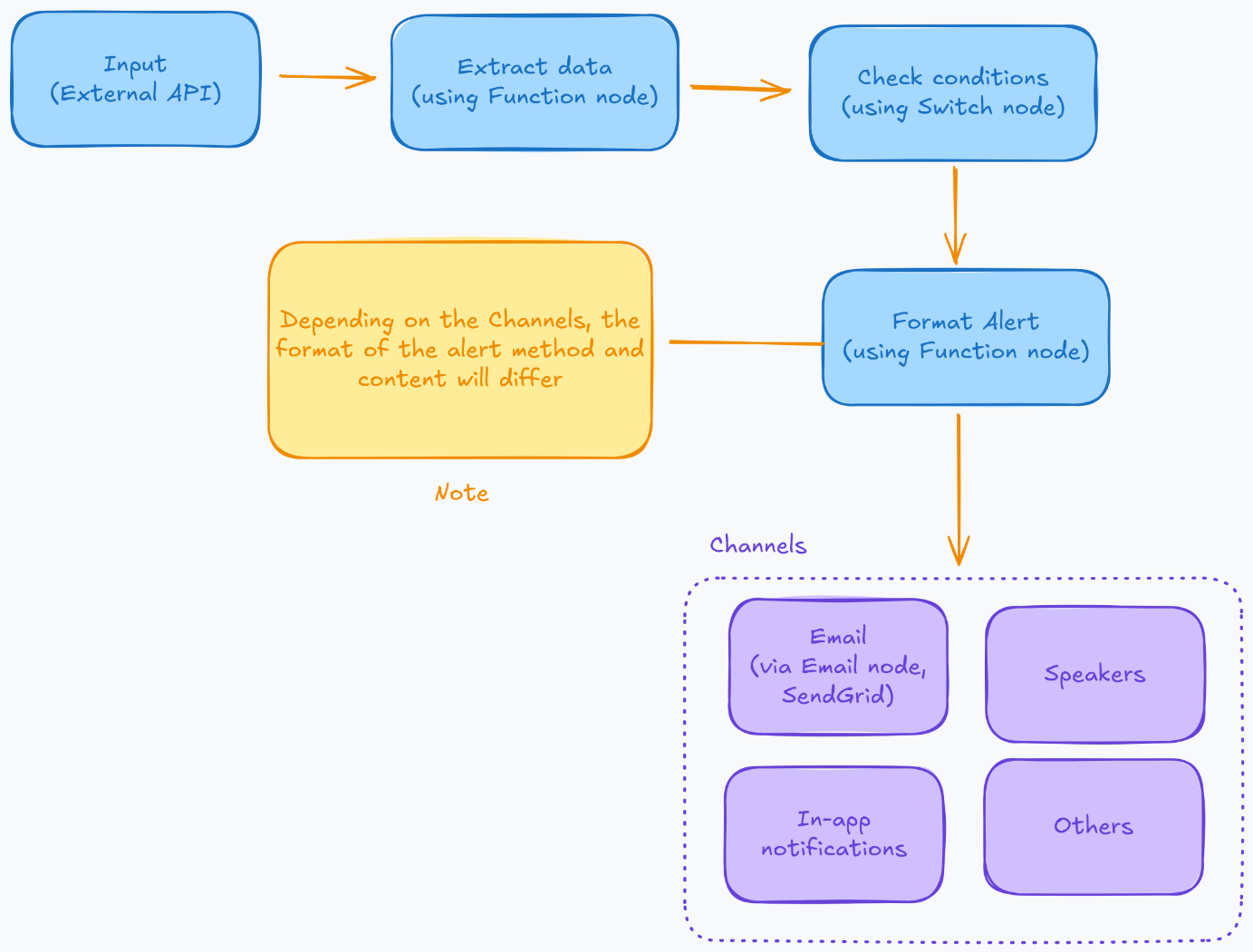
### 2.5.2. Sử dụng API của SendGrid

Để sử dụng API của SendGrid thì ta cần phải tạo một API key và xác thực email của sender, trong trường hợp này là email của tài khoản đăng ký, hoặc dùng một domain ngoài.

Vì đang sử dụng API nên ta phải sử dụng **Function node** để cấu hình thông tin của email và **HTTP Request node** để gửi API, so với việc định dạng nội dung của mail sẽ khó hơn so với **Email node** vì phải định dạng **payload** manual ở dạng JSON.

# 3. Ứng dụng vào Alert

Đối với flow xử lý sẽ có 4 bước:

* **Input:** Nhận API từ client bên ngoài
* **Extract:** Lấy ra dữ liệu của API và định dạng lại để có thế sử dụng
* **Conditions:** Tạo ra các điều kiện đế đối chiếu dữ liệu và thực thi hành đông tiếp theo
* **Format Alert:** Tạo ra các template tùy theo hình thức thông báo

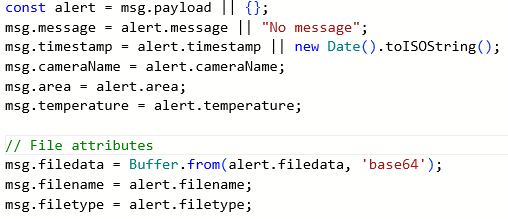
## 3.1. Input

Sử dụng **HTTP In node,** hệ thống nhận dữ liệu từ API qua một endpoint HTTP. Dữ liệu được gửi dưới dạng JSON hoặc Form data và có thể nhận file.



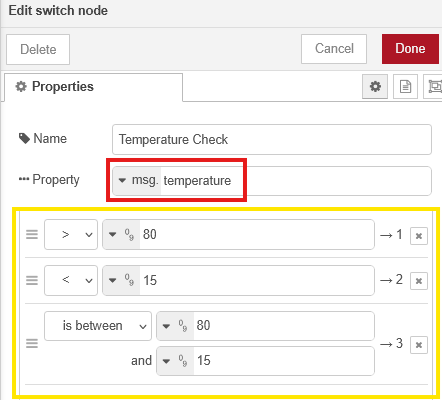
## 3.2. Extract

Như ảnh ở trên thì dữ liệu ta khi nhận ở dạng JSON sẽ có các key tương ứng, toàn bộ object này đang nằm trong biến **msg.payload**. Từ đó, mình có thể dùng **Function node** để tách ra từng **key** thành các biến cục bộ trong flow như:



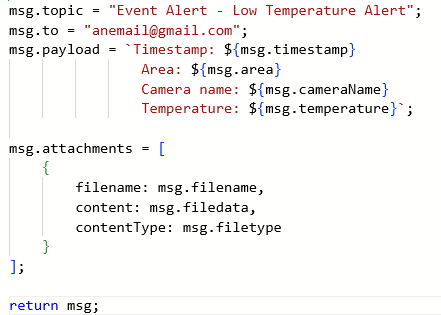
## 3.3. Conditions

Các điều kiện trong flow sẽ được cấu hình qua **Switch node**, với biến cục bộ đã được set trước như temperature, mình có thể đặt thuộc tính cần kiểm tra là **msg.temperature** rồi sau đó đặt các điều kiện tương ứng.



## 3.4. Format Alert

Tùy theo hình thức gửi alert thì sẽ có cái format khác nhau, như nếu qua loa AXIS thì sẽ dùng API của họ (sẽ tìm hiểu kĩ hơn sau), còn qua **Email node**, mình sẽ dùng **Function node** để khai báo các thuộc tính như **subject, receiver, content, attachments**.



# 4. So sánh giữa Zendesk Triggers và NodeRED

Trước tiên thì Zendesk Support và NodeRED là 2 hệ thống khác nhau hoàn toàn, với Zendesk thì họ chỉ hỗ trợ đối với các ticket, chat trong hệ thống của họ. Còn NodeRED là hệ thống thường được áp dụng vào các hệ thống IoT, có thể mở rộng tùy theo nhu cầu của người dùng. Vì vậy nếu so sánh thì chỉ có thể so được về Ease of Use, cách cấu hình, thực thi điều kiện.

## 4.1. Input

|  |  |  |
| --- | --- | --- |
|  | **Zendesk Triggers** | **NodeRED** |
| **Source** | Ticket, chat, hoặc messaging được thay đổi **bên trong Zendesk** | Bất kỳ nguồn dữ liệu nào: HTTP, MQTT, WebSocket, cảm biến IoT, Cloud service,... |
| **Trigger** | Cần phải dùng **Zendesk API hoặc Webhook để tạo ticket** đối với các source bên ngoài | Có thể kích hoạt trực tiếp từ bất kỳ sự kiện nào nhờ **Input node (HTTP In, MQTT In,…)** |
| **Scalability** | Chỉ theo dõi các thay đổi trong hệ thống Zendesk | Có thể cấu hình tùy chọn, các node chức năng mới |

Vậy thì Zendesk sẽ bị giới hạn trong hệ thống của họ, còn NodeRED cho phép người dùng cấu hình input tùy chỉnh với nhiều thiết bị, hệ thống bên ngoài.

## 4.2. Conditions

|  |  |  |
| --- | --- | --- |
|  | **Zendesk Triggers** | **NodeRED** |
| **Cấu hình điều kiện** | Sử dụng các rule logic cố định trong UI | Sử dụng các node như **Switch**, **Function** để viết điều kiện trong flow |
| **Độ phức tạp** | Tương đối cơ bản, chủ yếu dùng điều kiện ANY/ALL đơn giản | Mỗi condition trong **Switch** sẽ tương ứng với một output  Nếu muốn AND/OR thì phải viết bằng **Function** |
| **Cách thực thi** | Kiểm tra toàn bộ điều kiện | **Switch c**ho phép dừng tại điều kiện đúng đầu tiên hoặc Kiểm tra tất cả điều kiện, đúng điều kiện nào thì sẽ chạy điều kiện đó |

# 5. Kết luận

Nếu dùng Zendesk Triggers làm template để tự tạo một UI cho hệ thống Alert thì sẽ phù hợp hơn, các lựa chọn về condition và action đơn giản, dễ cho người dùng sử dụng. Còn NodeRED với kiểu lập trình kéo thả thì có thể ứng dụng về các service nằm bên dưới, nhận dữ liệu từ nhiều nguồn khác nhau, tạo logic xử lý điều kiện và tương tác với các Channels qua các node, hoặc request bằng API.