Instalar Node-red en Raspberry Pi

1. **Preparativos**:
    a. Asegúrate de que tu Raspberry Pi esté ejecutando Raspberry Pi OS Bullseye o una versión más reciente.
    b. Conéctate a tu Raspberry Pi mediante una terminal o una conexión SSH.
2. **Instalación de Node-RED**:
    a. Ejecuta el siguiente comando para descargar e instalar Node.js, npm y Node-RED:bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)

    b. Este script realizará las siguientes acciones:
        i. Eliminará cualquier versión existente de Node-RED.
        ii. Instalará Node.js 16 LTS si no está presente o si la versión actual es anterior a la 14.
        iii. Instalará la versión más reciente de Node-RED.
        iv. Opcionalmente, instalará nodos específicos para Raspberry Pi.
        v. Configurará Node-RED para que se ejecute como un servicio y proporcionará comandos para gestionarlo.
3. **Ejecutar Node-RED**:
    a. Para iniciar Node-RED manualmente, utiliza:node-red-start

    b. Para detener el servicio:node-red-stop

    c. Para reiniciar el servicio:node-red-restart

    d. Para ver los registros de Node-RED:node-red-log

4. **Configurar inicio automático**:
    a. Si deseas que Node-RED se inicie automáticamente al encender la Raspberry Pi, ejecuta:sudo systemctl enable nodered.service

    b. Para desactivar el inicio automático:sudo systemctl disable nodered.service

5. **Acceder al editor de Node-RED**:

a. Una vez que Node-RED esté en funcionamiento, abre un navegador web y accede a:
    i. Desde la Raspberry Pi: http://localhost:1880
    ii. Desde otro dispositivo en la misma red, reemplaza <dirección_ip_de_tu_pi> con la IP de tu Raspberry Pi: http://<dirección_ip_de_tu_pi>:1880
b. Para obtener la dirección IP de tu Raspberry Pi, ejecuta:hostname -I

El comando para configurar Node-RED para que se inicie automáticamente al arrancar el sistema es:

```
sudo systemctl enable nodered.service
```

Este comando habilita el servicio de Node-RED en el sistema para que se ejecute automáticamente al inicio.

Si en algún momento deseas deshabilitar el arranque automático de Node-RED, puedes usar:

sudo systemctl disable nodered.service

Mi primer flujo

Para crear tu primer flujo en Node-RED utilizando únicamente los nodos "Inject" y "Debug", sigue estos pasos:

1. **Acceder al editor de Node-RED**:
    a. Asegúrate de que Node-RED esté en funcionamiento.
    b. Abre un navegador web y dirígete a http://localhost:1880 si estás en la misma máquina donde se ejecuta Node-RED. Si estás en otra máquina, utiliza la dirección IP correspondiente, por ejemplo, http://<dirección_ip>:1880.
2. **Agregar un nodo Inject**:
    a. En la paleta de nodos (barra lateral izquierda), busca el nodo llamado "Inject".
    b. Arrastra y suelta el nodo "Inject" en el espacio de trabajo central.
    c. Este nodo permite inyectar mensajes manualmente en el flujo.
3. **Agregar un nodo Debug**:
    a. En la paleta de nodos, busca el nodo llamado "Debug".
    b. Arrastra y suelta el nodo "Debug" en el espacio de trabajo.
    c. El nodo "Debug" muestra el contenido de los mensajes en el panel de depuración (barra lateral derecha).

4. **Conectar los nodos**:
    a. Haz clic en el pequeño cuadro en el lado derecho del nodo "Inject" y, sin soltar, arrastra una línea hasta el pequeño cuadro en el lado izquierdo del nodo "Debug". Esto crea una conexión entre ambos nodos, permitiendo que los mensajes fluyan del nodo "Inject" al nodo "Debug".
5. **Desplegar el flujo**:
    a. Haz clic en el botón "Deploy" en la esquina superior derecha para implementar el flujo en el servidor de Node-RED.
6. **Probar el flujo**:
    a. Asegúrate de que el panel de depuración esté visible; si no es así, haz clic en el ícono de error (un insecto) en la barra lateral derecha.
    b. En el nodo "Inject" del espacio de trabajo, haz clic en el pequeño botón cuadrado en su lado izquierdo. Esto inyectará un mensaje en el flujo.
    c. Observa el panel de depuración para ver la salida del nodo "Debug". Por defecto, el nodo "Inject" envía una marca de tiempo (el número de milisegundos desde el 1 de enero de 1970).

Este sencillo flujo te permite inyectar manualmente un mensaje y ver su contenido en el panel de depuración, lo que es útil para probar y depurar flujos en Node-RED.

Nodos inject y debug

Los nodos **Inject** y **Debug** en Node-RED son herramientas fundamentales para el desarrollo y prueba de flujos. A continuación, se detallan sus características y funcionalidades:

Nodo Inject:

- **Propósito**: Permite activar manualmente un flujo haciendo clic en el botón del nodo dentro del editor. También puede configurarse para activar flujos automáticamente a intervalos regulares.
- **Configuración del mensaje**: El nodo Inject permite establecer las propiedades payload y topic del mensaje. El payload puede configurarse con diversos tipos de datos, como:
    - Valores de propiedades de contexto de flujo o global.
    - Cadenas de texto (String), números, booleanos, Buffers u objetos.
    - Una marca de tiempo en milisegundos desde el 1 de enero de 1970.
- **Configuración de intervalos**: El nodo puede configurarse para activar flujos en intervalos de tiempo específicos, con un máximo de 596 horas (aproximadamente 24 días). Para intervalos mayores, se recomienda utilizar nodos de programación que manejen cortes de energía y reinicios.

- **Opciones de tiempo**: Las opciones de "intervalo entre tiempos" y "a una hora específica" utilizan el sistema cron estándar. Por ejemplo, un intervalo de 20 minutos se ejecutará a los 20 y 40 minutos de cada hora, no cada 20 minutos desde el momento actual.
- **Capacidades adicionales**: Desde Node-RED 1.1.0, el nodo Inject puede establecer cualquier propiedad en el mensaje, no solo payload y topic.

Nodo Debug:

- **Propósito**: Se utiliza para mostrar mensajes en la barra lateral de depuración dentro del editor, facilitando la exploración y comprensión de los mensajes que fluyen a través del sistema.
- **Visualización estructurada**: La barra lateral proporciona una vista estructurada de los mensajes, lo que facilita la exploración de su contenido.
- **Información adicional**: Junto a cada mensaje, se muestra la hora de recepción y el nodo Debug que lo envió. Al hacer clic en el ID del nodo fuente, se resalta ese nodo en el espacio de trabajo.
- **Control de salida**: El botón en el nodo permite habilitar o deshabilitar su salida. Se recomienda desactivar o eliminar los nodos Debug que no se estén utilizando para mantener la claridad en el flujo.
- **Opciones de registro**: El nodo puede configurarse para enviar todos los mensajes al registro de tiempo de ejecución o para mostrar mensajes cortos (hasta 32 caracteres) en el estado del nodo.
- **Uso recomendado**: Es una práctica común utilizar el nodo Debug para comprender la estructura de los mensajes, especialmente al trabajar con mensajes complejos o al desarrollar nuevas funcionalidades.

Estos nodos son esenciales para el desarrollo y depuración en Node-RED, proporcionando mecanismos para inyectar datos en los flujos y monitorear su comportamiento de manera efectiva.

Mensajes o msg

En Node-RED, los **msg** son objetos que representan los mensajes que fluyen a través de los nodos en un flujo. Son esenciales en la arquitectura de Node-RED, ya que contienen la información que los nodos procesan, manipulan y transmiten.

Características de los msg

1. **Estructura flexible**:
    a. Un msg es un objeto JavaScript que puede tener múltiples propiedades.

    b. La propiedad más común y utilizada es msg.payload, pero se pueden agregar y utilizar otras propiedades según sea necesario.
2. **Transporte de datos**:
    a. Los msg actúan como contenedores de datos, permitiendo que los nodos intercambien información en un flujo.
3. **Personalización**:
    a. Los usuarios pueden definir y manipular propiedades del msg según lo que requiera el flujo.

Uso de los msg

- Los msg son el medio principal para que los nodos interactúen entre sí.
- Los nodos pueden:
    - **Leer**: Acceder a las propiedades existentes en un msg para procesar información.
    - **Modificar**: Alterar las propiedades de un msg para cambiar el contenido que se pasa a otros nodos.
    - **Crear nuevas propiedades**: Añadir datos específicos necesarios para otros nodos del flujo.

Ejemplo básico de un msg:

```
{
  payload: "Hello, World!",
  topic: "greeting",
  timestamp: 1677721600
}
```

Propiedades comunes de un msg

1. **msg.payload**:
    a. Es la propiedad principal que transporta el contenido o datos del mensaje.
    b. Puede contener cualquier tipo de dato: texto, números, objetos, arreglos, etc.
2. **msg.topic**:
    a. Una propiedad opcional que suele utilizarse para categorizar o identificar el origen del mensaje.
3. **Otras propiedades personalizadas**:
    a. Los usuarios pueden agregar propiedades como msg.temperature, msg.user, o msg.location según las necesidades del flujo.

Tipos de propiedades en un msg

Las propiedades de un msg pueden ser de cualquier tipo de dato válido en JavaScript:

1. **Primitivos**:
    a. String: Cadena de texto. Ejemplo: "Hola".
    b. Number: Números. Ejemplo: 42.
    c. Boolean: Valores verdadero/falso. Ejemplo: true.
2. **Estructurados**:
    a. Object: Contenedores de propiedades clave-valor. Ejemplo:{ temperature: 25, unit: "Celsius" }

    b. Array: Listas de valores. Ejemplo: [1, 2, 3, 4].
3. **Especiales**:
    a. Buffer: Secuencias de datos binarios, útiles para manejar datos como imágenes o archivos.
    b. Date: Fechas y horas.
4. **Otros**:
    a. null: Valor vacío o inexistente.

Ejemplo práctico del uso de un msg

Imagina un flujo donde un sensor de temperatura envía datos. El nodo puede generar un mensaje como:

```
{
  payload: 22.5,
  topic: "sensor/temperature",
  unit: "Celsius",
  timestamp: 1677721600
}
```

- **Nodo Debug**: Muestra el mensaje completo en el panel de depuración.
- **Nodo Function**: Puede modificar el mensaje para incluir más información:msg.warning = msg.payload > 30 ? "High temperature" : "Normal"; return msg;

Nodo Change

El nodo **Change** en Node-RED se utiliza para modificar las propiedades de un mensaje o establecer propiedades de contexto sin necesidad de recurrir a un nodo Function.

Características del nodo Change

- **Operaciones disponibles**:
  - **Set (Establecer)**: Permite asignar un valor a una propiedad específica. El valor puede ser de diversos tipos o derivarse de una propiedad existente del mensaje o del contexto.
  - **Change (Cambiar)**: Realiza una búsqueda y reemplazo en una propiedad del mensaje.
  - **Move (Mover)**: Permite mover o renombrar una propiedad.
  - **Delete (Eliminar)**: Elimina una propiedad específica del mensaje.
- **Uso de expresiones JSONata**: Al establecer una propiedad, el valor puede derivarse de una expresión JSONata, un lenguaje declarativo para la consulta y transformación de datos JSON.

Posibles usos del nodo Change

- **Modificación de datos en tránsito**: Ajustar el contenido de msg.payload antes de que llegue a otros nodos, como cambiar un valor numérico o formatear una cadena de texto.
- **Renombrar propiedades**: Mover msg.payload a msg.data para cumplir con los requisitos de nodos posteriores que esperan datos en msg.data.
- **Eliminación de propiedades innecesarias**: Eliminar propiedades no deseadas del mensaje para simplificar su estructura antes de su procesamiento posterior.
- **Establecer propiedades de contexto**: Asignar valores a propiedades de contexto de flujo o globales, permitiendo compartir datos entre diferentes partes del flujo sin necesidad de un nodo Function.
- **Transformaciones complejas con JSONata**: Aplicar expresiones JSONata para realizar transformaciones avanzadas en los datos del mensaje, como cálculos matemáticos, manipulaciones de cadenas o filtrado de datos.

El nodo Change es una herramienta versátil que facilita la manipulación de mensajes y propiedades de contexto en Node-RED, ofreciendo una alternativa más sencilla y directa al uso de código JavaScript en nodos Function para tareas comunes de transformación de datos.

Contexto

En Node-RED, el **contexto** es una funcionalidad que permite almacenar información que puede ser compartida entre diferentes nodos sin necesidad de pasarla explícitamente a través de los mensajes en un flujo. Esto facilita la gestión de estados y datos persistentes dentro de una aplicación.

Niveles de Contexto

Node-RED ofrece tres niveles de contexto, cada uno con un alcance específico:

1. **Contexto de Nodo**:
    a. **Alcance**: Visible únicamente para el nodo que establece el valor.
    b. **Uso**: Ideal para almacenar datos que solo son relevantes dentro de un nodo específico, como contadores locales o estados temporales.
2. **Contexto de Flujo**:
    a. **Alcance**: Accesible para todos los nodos dentro del mismo flujo o pestaña en el editor.
    b. **Uso**: Útil para compartir información entre nodos que forman parte del mismo flujo, como configuraciones compartidas o resultados intermedios.
3. **Contexto Global**:
    a. **Alcance**: Disponible para todos los nodos en todos los flujos.
    b. **Uso**: Adecuado para datos que deben ser accesibles en toda la aplicación, como configuraciones globales o estados compartidos entre múltiples flujos.

Uso del Contexto en un Flujo

Para interactuar con el contexto en un flujo, se pueden utilizar nodos como el nodo **Change**, que permite establecer, cambiar, mover o eliminar valores en el contexto deseado. Por ejemplo, para almacenar el valor de msg.payload en el contexto de flujo bajo la clave miDato, se puede configurar una regla en el nodo Change.

Además, nodos como el nodo **Inject** pueden configurarse para inyectar valores del contexto, y el nodo **Switch** puede enrutar mensajes basándose en valores almacenados en el contexto.

Almacenamiento Persistente del Contexto

Por defecto, el contexto se almacena en memoria, lo que significa que su contenido se pierde al reiniciar Node-RED. Sin embargo, es posible configurar Node-RED para que guarde los datos del contexto en el sistema de archivos, asegurando su persistencia

entre reinicios. Esto se logra modificando la propiedad contextStorage en el archivo settings.js. Por ejemplo, para habilitar el almacenamiento en el sistema de archivos:

```
contextStorage: {
  default: {
    module: "localfilesystem"
  }
}
```

Esta configuración almacena los datos del contexto en archivos bajo ~/.node-red/context/ y los escribe en el sistema de archivos cada 30 segundos.

Buenas Prácticas

- **Alcance Adecuado**: Selecciona el nivel de contexto más apropiado según la necesidad. Si un valor solo es necesario en un nodo, utiliza el contexto de nodo. Para compartir datos en un flujo, emplea el contexto de flujo, y para datos globales, el contexto global.
- **Persistencia Necesaria**: Configura el almacenamiento persistente solo para los datos que deben sobrevivir a los reinicios de Node-RED, evitando un uso innecesario de recursos.
- **Gestión de Subflujos**: Ten en cuenta que los nodos dentro de un subflujo comparten el contexto de flujo del subflujo, no del flujo principal. Desde Node-RED 0.20, es posible acceder al contexto del flujo principal desde un subflujo utilizando $parent. en la clave del contexto.
- **Eliminación de Datos Innecesarios**: Utiliza nodos como el nodo Change para eliminar propiedades del contexto que ya no sean necesarias, manteniendo el entorno limpio y eficiente.

Nodo Function

El nodo **Function** en Node-RED permite ejecutar código JavaScript personalizado sobre los mensajes que fluyen a través de él, proporcionando una flexibilidad significativa para manipular datos, implementar lógica condicional y realizar operaciones complejas que van más allá de las capacidades de los nodos predeterminados.

Estructura Básica de una Función

Dentro del nodo Function, el mensaje entrante se representa como un objeto llamado msg. Por convención, este objeto contiene una propiedad msg.payload que alberga el

contenido principal del mensaje. El código que se escribe en el nodo Function corresponde al cuerpo de una función que recibe msg como argumento.

Un ejemplo sencillo de una función que retorna el mensaje tal cual se recibió:

```
return msg;
```

Si la función retorna null, el flujo se detiene y no se envía ningún mensaje a los nodos siguientes. Es importante que la función siempre retorne un objeto msg. Retornar tipos de datos primitivos como números o cadenas resultará en un error.

Manipulación del Mensaje

El nodo Function permite crear un nuevo objeto msg o modificar el existente. Sin embargo, es fundamental tener en cuenta que al construir un nuevo objeto desde cero, se pueden perder propiedades del mensaje original que podrían ser necesarias en otros nodos del flujo. Por ejemplo, en un flujo HTTP, las propiedades msg.req y msg.res deben preservarse para garantizar una correcta respuesta.

Ejemplo de creación de un nuevo mensaje basado en el original:

```
var newMsg = { payload: msg.payload.length };
return newMsg;
```

Múltiples Salidas

El nodo Function puede configurarse para tener múltiples salidas. Para dirigir mensajes a salidas específicas, se retorna un arreglo donde cada posición corresponde a una salida. Por ejemplo, para enviar un mensaje a la segunda salida si el tema es "banana":

```
if (msg.topic === "banana") {
  return [ null, msg ];
} else {
  return [ msg, null ];
}
```

Además, es posible enviar múltiples mensajes por una misma salida retornando un arreglo de mensajes en la posición correspondiente:

```
var msg1 = { payload: "primer mensaje" };
var msg2 = { payload: "segundo mensaje" };
return [ [ msg1, msg2 ], null ];
```

Envío Asíncrono de Mensajes

Cuando se realizan operaciones asíncronas dentro de un nodo Function, no es posible retornar el mensaje de inmediato. En su lugar, se utiliza node.send() para enviar el mensaje una vez completada la operación asíncrona:

```
doSomeAsyncWork(msg, function(result) {
   msg.payload = result;
   node.send(msg);
   node.done();
});
return;
```

Es importante llamar a node.done() después de enviar el mensaje para que el tiempo de ejecución de Node-RED pueda rastrear correctamente los mensajes a través del sistema.

Ejemplos Prácticos

1. **Dividir una Cadena en Palabras**:

Este ejemplo toma una cadena en msg.payload, la divide en palabras y envía cada palabra como un mensaje separado:

```
var words = msg.payload.split(" ");
var outputMsgs = words.map(word => ({ payload: word }));
return [ outputMsgs ];
```

2. **Contador de Mensajes**:

Este ejemplo mantiene un contador de cuántos mensajes han pasado por el nodo Function:

```
var count = context.get('count') || 0;
count += 1;
context.set('count', count);
msg.count = count;
return msg;
```

Buenas Prácticas

- **Manejo de Errores**: Utiliza node.error() para registrar errores y detener el flujo cuando sea necesario.
- **Registro y Depuración**: Emplea node.warn() para mostrar advertencias y node.log() para mensajes informativos durante el desarrollo y depuración.
- **Uso de Contexto**: Aprovecha las variables de contexto (context, flow, global) para almacenar datos que deben persistir entre ejecuciones de la función o ser compartidos entre nodos.
- **Asincronía**: Asegúrate de manejar correctamente las operaciones asíncronas utilizando node.send() y node.done() para evitar bloqueos en el flujo.

Los siguientes objetos están disponibles dentro del nodo Función.

node

- node.id: el id del nodo de función - agregado en 0.19
- node.name: el nombre del nodo de función - agregado en 0.19
- node.outputCount: número de salidas establecidas para el nodo
- node.log(..): Registrar un mensaje
- node.warn(..): Registrar un mensaje de advertencia
- node.error(..): Registrar un mensaje de error
- node.debug(..): Registrar un mensaje de depuración
- node.trace(..): Registrar un mensaje de seguimiento
- node.on(..): Registrar un controlador de eventos
- node.status(..): actualizar el estado del nodo
- node.send(..): enviar un mensaje
- node.done(..): terminar con un mensaje

context

- context.get(..): obtener una propiedad de contexto con alcance de nodo
- context.set(..):establece propiedad de contexto con alcance de nodo
- context.keys(..): devuelve una lista de todas las claves de propiedades de contexto del ámbito del nodo
- context.flow:lo mismo queflow
- context.global:lo mismo queglobal

flow

- flow.get(..): obtener una propiedad de contexto con alcance de flujo
- flow.set(..):establecer una propiedad de contexto con alcance de flujo
- flow.keys(..): devuelve una lista de todas las claves de propiedades de contexto con alcance de flujo

global

- global.get(..):obtener una propiedad de contexto de alcance global
- global.set(..):establecer una propiedad de contexto de alcance global

global.keys(..): devuelve una lista de todas las claves de propiedades de contexto de ámbito global

RED

RED.util.cloneMessage(..): clona de forma segura un objeto de mensaje para que pueda reutilizarse

env

env.get(..):obtener una variable de entorno

Otros módulos y funciones

El nodo Función también pone a disposición los siguientes módulos y funciones:

Buffer- el Buffermódulo Node.js

console- el consolemódulo Node.js (node.loges el método de registro preferido)

util- el utilmódulo Node.js

setTimeout/clearTimeout- las funciones de tiempo de espera de javascript.

setInterval/clearInterval- las funciones de intervalo de javascript.

Nota: el nodo de función borra automáticamente cualquier tiempo de espera o temporizador de intervalo pendiente cada vez que se detiene o se vuelve a implementar.

Nodo Complete

Descripción y Funcionamiento del Nodo Complete en Node-RED

El nodo **Complete** se utiliza para capturar el evento de finalización de la ejecución de otro nodo dentro de un flujo. Su propósito principal es facilitar el seguimiento y la gestión de tareas complejas, especialmente en flujos donde se requiere realizar acciones después de que ciertos nodos hayan completado su ejecución.

Funcionamiento del Nodo Complete

1. **Evento de Finalización**:
    a. El nodo Complete detecta cuando un nodo específico del flujo ha terminado su tarea y ha invocado la función node.done() (para nodos que

requieren seguimiento asíncrono) o simplemente ha completado su procesamiento.

2. **Configuración**:
    a. Para usar el nodo Complete, debes configurarlo para monitorear un nodo específico en el flujo. Esto se hace seleccionando el nodo objetivo en el menú desplegable de configuración del nodo Complete.
3. **Mensaje Emitido**:
    a. Cuando el nodo objetivo finaliza, el nodo Complete emite un mensaje que contiene información adicional, como:
        i. El msg.payload enviado por el nodo que completó la tarea.
        ii. Detalles sobre el estado de la ejecución, si están disponibles.
4. **Compatibilidad con Nodos Personalizados**:
    a. Es particularmente útil con nodos que implementan operaciones asíncronas, ya que permite capturar el momento exacto en que dichas operaciones se completan.

<u>Casos de Uso del Nodo Complete</u>

1. **Monitoreo de Procesos Asíncronos**:
    a. Cuando un nodo Function o un nodo personalizado realiza una operación asíncrona (como llamadas a APIs o lecturas de bases de datos), el nodo Complete puede capturar el evento de finalización para ejecutar tareas posteriores, como limpiar recursos o iniciar otros procesos.

Ejemplo:

    b. Un nodo HTTP Request hace una llamada a una API externa. El nodo Complete se usa para registrar en el sistema de logs que la solicitud se completó correctamente.
2. **Tareas de Finalización**:
    a. Realizar tareas complementarias después de que un nodo termine su ejecución, como:
        i. Enviar notificaciones.
        ii. Actualizar estados en un sistema externo.
        iii. Guardar resultados en una base de datos.

Ejemplo:

    b. Después de procesar un archivo en un flujo, el nodo Complete se usa para mover el archivo procesado a una carpeta "completados".

Nodo catch

El nodo **Catch** permite gestionar errores que ocurren durante la ejecución de un flujo. Permite interceptar y manejar excepciones generadas por otros nodos, facilitando la implementación de estrategias de manejo de errores y asegurando que la aplicación responda de manera adecuada ante situaciones inesperadas.

Funcionamiento del Nodo Catch

- **Captura de Errores**: El nodo Catch detecta errores notificados por otros nodos en el mismo flujo. Cuando un nodo encuentra una condición de error y la reporta adecuadamente, el nodo Catch intercepta este evento.
- **Configuración**: Se puede configurar el nodo Catch para que responda a errores de:
    - **Todos los nodos** en la misma pestaña del editor.
    - **Nodos específicos** seleccionados por el usuario.
    - **Nodos dentro del mismo grupo** al que pertenece el nodo Catch.
- **Mensaje Emitido**: Al interceptar un error, el nodo Catch emite un mensaje que contiene:
    - **msg.payload**: El payload del mensaje original que causó el error.
    - **msg.error**: Un objeto con detalles del error, incluyendo:
        - **message**: Descripción del error.
        - **source**: Información sobre el nodo que generó el error, como id, type, name y count (número de veces que se ha producido este error).

Casos de Uso del Nodo Catch

1. **Manejo de Errores en Integraciones Externas**:
    a. **Descripción**: Al interactuar con APIs externas, bases de datos o servicios de red, pueden ocurrir errores debido a problemas de conectividad, tiempos de espera o respuestas inesperadas.
    b. **Implementación**: Configura el nodo Catch para interceptar estos errores y dirigirlos a un nodo Debug para su registro, o a un nodo Function que implemente lógica de reintento o notificaciones al usuario.
2. **Validación de Entradas del Usuario**:
    a. **Descripción**: En aplicaciones que procesan datos ingresados por usuarios, es fundamental validar la información para evitar errores en el flujo.
    b. **Implementación**: Utiliza el nodo Catch para capturar errores generados por datos inválidos y redirigirlos a un nodo que informe al usuario sobre la corrección necesaria.

3. **Monitoreo de Estados de Conexión**:
    a. **Descripción**: Nodos que manejan conexiones, como los nodos MQTT, pueden experimentar desconexiones inesperadas.
    b. **Implementación**: El nodo Catch puede configurarse para detectar estos eventos y activar flujos que intenten reconectar o notifiquen al administrador del sistema.
4. **Registro Centralizado de Errores**:
    a. **Descripción**: En aplicaciones complejas, es útil tener un registro centralizado de errores para análisis y depuración.
    b. **Implementación**: Dirige todos los mensajes del nodo Catch a un sistema de logging o base de datos donde se almacenen los detalles de los errores para su posterior revisión.

Nodo status

El nodo **Status** en Node-RED es una herramienta que permite monitorear y reaccionar ante los cambios de estado de otros nodos dentro de un flujo. Al capturar estas actualizaciones, es posible implementar acciones específicas basadas en el estado operativo de los nodos, mejorando la capacidad de respuesta y la robustez de las aplicaciones desarrolladas en Node-RED.

Funcionamiento del Nodo Status

- **Captura de Estados**: El nodo Status detecta y recibe las actualizaciones de estado emitidas por otros nodos en el mismo flujo. Estos estados suelen reflejar condiciones como conexiones establecidas, errores, desconexiones u otros eventos relevantes.
- **Configuración**: Al añadir un nodo Status al flujo, se puede configurar para que escuche:
    - **Todos los nodos** en la misma pestaña del editor.
    - **Nodos específicos** seleccionados por el usuario.
    - **Nodos dentro del mismo grupo** al que pertenece el nodo Status.
- **Mensaje Emitido**: Cuando un nodo monitoreado actualiza su estado, el nodo Status emite un mensaje que contiene:
    - **msg.status**: Un objeto con detalles del estado, incluyendo:
        - **text**: Descripción del estado actual.
        - **source**: Información sobre el nodo que reportó el estado, como id, type, name.

Casos de Uso del Nodo Status

1. **Monitoreo de Conexiones**:

    a. **Descripción**: En flujos que interactúan con servicios externos, como bases de datos o APIs, es crucial conocer el estado de las conexiones para garantizar una comunicación efectiva.
    b. **Implementación**: Configura el nodo Status para escuchar los nodos de conexión y, en caso de detectar una desconexión, activar alertas o procedimientos de reconexión automática.
2. **Gestión de Errores en Tiempo Real**:
    a. **Descripción**: Detectar y reaccionar ante errores operativos de manera inmediata es esencial para mantener la integridad del flujo.
    b. **Implementación**: Utiliza el nodo Status para capturar estados de error de nodos críticos y redirigir esta información a sistemas de notificación o registros de auditoría.
3. **Control de Flujo Basado en Estados**:
    a. **Descripción**: Algunos procesos pueden depender del estado de ciertos nodos para proceder con su ejecución.
    b. **Implementación**: Emplea el nodo Status para determinar cuándo un nodo ha alcanzado un estado específico (por ejemplo, "listo") y, a partir de ello, desencadenar la siguiente etapa del proceso.
4. **Visualización de Estados en Interfaces de Usuario**:
    a. **Descripción**: Proporcionar a los usuarios información en tiempo real sobre el estado de diversos componentes del sistema mejora la transparencia y la experiencia de uso.
    b. **Implementación**: Conecta el nodo Status a nodos de interfaz, como dashboards, para mostrar visualmente los estados actuales de diferentes nodos o servicios.

Consideraciones Importantes

- **Selección de Nodos**: Asegúrate de configurar el nodo Status para escuchar únicamente los nodos relevantes, evitando sobrecargar el flujo con información innecesaria.
- **Manejo de Estados**: Implementa lógica adecuada para manejar diferentes estados reportados por los nodos, garantizando que el sistema responda de manera coherente ante diversas situaciones operativas.
- **Integración con Sistemas de Notificación**: Considera conectar el nodo Status a sistemas de notificación externos, como correos electrónicos o mensajes instantáneos, para alertar a los administradores sobre eventos críticos en tiempo real.

Nodos link

Los nodos **Link** en Node-RED facilitan la conexión lógica entre diferentes partes de un flujo, ya sea dentro de la misma pestaña o entre pestañas distintas, sin necesidad de utilizar conexiones físicas directas. Esto permite organizar y estructurar los flujos de manera más clara y eficiente.

Tipos de Nodos Link

1. **Link Out**: Este nodo se utiliza para enviar mensajes desde un punto específico del flujo.
2. **Link In**: Este nodo recibe mensajes enviados por uno o más nodos Link Out.
3. **Link Call**: Introducido en versiones más recientes de Node-RED, este nodo permite invocar un subflujo definido por un nodo Link In y esperar una respuesta, facilitando la creación de flujos modulares y reutilizables.

Funcionamiento de los Nodos Link

- **Conexión Lógica**: Los nodos Link establecen conexiones lógicas representadas por líneas discontinuas en el editor de Node-RED, indicando la relación entre los nodos sin una conexión física directa.
- **Configuración**: Para establecer una conexión, se configura un nodo Link Out para que apunte a uno o más nodos Link In. Esta configuración se realiza seleccionando los nodos de destino en las propiedades del nodo Link Out.
- **Transmisión de Mensajes**: Cuando un nodo Link Out recibe un mensaje, lo transmite automáticamente a todos los nodos Link In asociados, permitiendo la comunicación entre diferentes partes del flujo.

Casos de Uso de los Nodos Link

1. **Organización de Flujos Complejos**: En flujos con múltiples nodos y conexiones, los nodos Link ayudan a reducir el desorden visual al eliminar la necesidad de largas conexiones físicas, mejorando la legibilidad y mantenimiento del flujo.
2. **Comunicación entre Pestañas**: Permiten la transferencia de mensajes entre diferentes pestañas del editor, facilitando la modularización y separación lógica de componentes del flujo.
3. **Reutilización de Subflujos**: Con el nodo Link Call, es posible definir subflujos que pueden ser invocados desde diferentes partes del flujo principal, promoviendo la reutilización de lógica común y la creación de flujos más estructurados.
4. **Implementación de Lógica Condicional**: Los nodos Link pueden utilizarse para dirigir mensajes a diferentes partes del flujo basándose en condiciones

específicas, mejorando la flexibilidad y control sobre el procesamiento de datos.

Buenas Prácticas

- **Nombres Descriptivos**: Asigna nombres claros y descriptivos a los nodos Link para facilitar la identificación de sus funciones y conexiones, especialmente en flujos complejos.
- **Documentación**: Utiliza las propiedades de los nodos para añadir descripciones que expliquen el propósito de cada nodo Link y cómo se relaciona con otros nodos en el flujo.
- **Estructuración Lógica**: Emplea nodos Link para dividir el flujo en secciones lógicas, mejorando la claridad y facilitando el mantenimiento y la escalabilidad del proyecto.

Nodo comment

El nodo **Comment** en Node-RED es esencial para documentar y mejorar la legibilidad de los flujos. Permite añadir anotaciones descriptivas que facilitan la comprensión, el mantenimiento y la colaboración en proyectos desarrollados con Node-RED.

Descripción y Funcionamiento del Nodo Comment

- **Propósito**: El nodo Comment se utiliza para insertar comentarios dentro del flujo, proporcionando contexto adicional sobre la funcionalidad, propósito o detalles específicos de una sección del flujo. Estos comentarios son visibles directamente en el editor, lo que ayuda a otros desarrolladores (o a uno mismo en el futuro) a entender rápidamente la lógica implementada.
- **Configuración**: Al añadir un nodo Comment al flujo, puedes establecer:
    - **Etiqueta (Label)**: Un título breve que aparece en el nodo dentro del flujo.
    - **Descripción**: Un texto más detallado que se muestra en la barra lateral de información cuando se selecciona el nodo. Esta descripción admite formato Markdown, lo que permite incluir listas, enlaces y otros elementos de formato para enriquecer la documentación.
- **Visualización**: El nodo Comment no afecta la ejecución del flujo, ya que no procesa mensajes ni se conecta mediante cables a otros nodos. Su función es puramente informativa y visual, sirviendo como anotación dentro del editor.

Casos de Uso del Nodo Comment

1. **Documentación de Flujos Complejos**:

    a. **Descripción**: En flujos con lógica intrincada, los nodos Comment pueden utilizarse para explicar secciones específicas, describir la funcionalidad de grupos de nodos o detallar procesos complejos, facilitando la comprensión y el mantenimiento.
2. **Instrucciones para Colaboradores**:
    a. **Descripción**: Al trabajar en equipo, es útil proporcionar instrucciones o notas para otros desarrolladores. Los nodos Comment pueden indicar áreas que requieren atención, sugerir mejoras o explicar decisiones de diseño.
3. **Marcar Tareas Pendientes**:
    a. **Descripción**: Durante el desarrollo, puedes utilizar nodos Comment para señalar partes del flujo que necesitan ser completadas o revisadas, actuando como recordatorios visuales de tareas pendientes.
4. **Explicación de Parámetros de Configuración**:
    a. **Descripción**: Si un flujo depende de ciertos parámetros o variables de configuración, los nodos Comment pueden detallar su propósito, valores esperados y cómo afectan al funcionamiento del flujo.

Buenas Prácticas al Utilizar el Nodo Comment

- **Claridad y Concisión**: Aunque es posible añadir descripciones extensas, es recomendable ser claro y conciso, proporcionando la información necesaria sin abrumar con detalles innecesarios.
- **Uso de Formato**: Aprovecha las capacidades de formato Markdown en la descripción para estructurar la información de manera efectiva, utilizando listas, encabezados y enlaces según sea apropiado.
- **Consistencia**: Mantén un estilo consistente en los comentarios a lo largo del flujo, facilitando la lectura y comprensión uniforme para cualquier persona que revise el proyecto.
- **Actualización Regular**: Asegúrate de mantener los comentarios actualizados conforme el flujo evoluciona, evitando descripciones obsoletas que puedan generar confusión.

Nodo switch

El nodo **Switch** en Node-RED permite dirigir mensajes a diferentes ramas de un flujo evaluando un conjunto de reglas definidas para cada mensaje. Su nombre proviene de la instrucción "switch" común en muchos lenguajes de programación y no se refiere a un interruptor físico.

Configuración del Nodo Switch

- **Propiedad a Evaluar**: Se debe especificar la propiedad del mensaje o del contexto que será evaluada. Por defecto, suele ser msg.payload, pero puede configurarse para cualquier otra propiedad.
- **Tipos de Reglas**: El nodo Switch permite definir varios tipos de reglas para evaluar la propiedad seleccionada:
    - **Reglas de Valor**: Comparan la propiedad con un valor específico utilizando operadores como igual a, diferente de, mayor que, menor que, entre otros.
    - **Reglas de Secuencia**: Se utilizan para evaluar secuencias de mensajes, como las generadas por el nodo Split, permitiendo operaciones como detectar el primer o último mensaje de una secuencia.
    - **Expresiones JSONata**: Permiten evaluar expresiones complejas contra el mensaje completo. Si la expresión devuelve true, la regla se considera cumplida. JSONata es un lenguaje de consulta y transformación declarativo para datos JSON.
    - **Regla "De lo Contrario" (Otherwise)**: Esta regla se cumple si ninguna de las reglas anteriores ha coincidido, actuando como una cláusula de respaldo.
- **Comportamiento de Evaluación**: El nodo puede configurarse para:
    - **Evaluar Todas las Reglas**: El mensaje se enviará a todas las salidas cuyas reglas coincidan.
    - **Detenerse en la Primera Coincidencia**: El nodo dejará de evaluar reglas después de encontrar la primera coincidencia, enviando el mensaje solo a la salida correspondiente.

Nodo Range

El nodo **Range** en Node-RED permite mapear un valor numérico de un rango de entrada a otro rango de salida, escalando linealmente los valores. Esto es útil en situaciones donde se requiere convertir datos de sensores o ajustar valores para diferentes unidades de medida.

Configuración del Nodo Range

- **Propiedad a Escalar**: Por defecto, el nodo toma msg.payload como la propiedad a escalar, pero se puede configurar para utilizar cualquier otra propiedad del mensaje.
- **Rango de Entrada**: Se define el rango numérico original del valor que se recibe.
- **Rango de Salida**: Se establece el rango numérico al cual se desea mapear el valor de entrada.
- **Opciones de Escalado**:

    - **Escalar**: Realiza una escala lineal simple del valor de entrada al rango de salida, permitiendo que los resultados excedan los límites del rango de salida si el valor de entrada está fuera del rango de entrada.
    - **Escalar y Limitar al Rango de Salida**: Escala el valor y lo restringe dentro del rango de salida, asegurando que el resultado no supere los límites establecidos.
    - **Escalar y Envolver dentro del Rango de Salida**: Escala el valor y, si está fuera del rango de salida, lo ajusta cíclicamente dentro de los límites del rango de salida.
    - **Escalar, pero Descartar si está Fuera del Rango de Entrada**: Escala el valor solo si está dentro del rango de entrada; si no, descarta el mensaje.
- **Redondear al Entero más Cercano**: Opción para redondear el resultado al número entero más cercano, útil cuando se requieren valores discretos.

Ejemplos de Uso del Nodo Range

1. **Conversión de Lecturas de Sensor a Porcentaje**:
    a. **Escenario**: Un sensor proporciona valores en el rango de 0 a 1023, y se desea convertir estos valores a un porcentaje entre 0% y 100%.
    b. **Configuración**:
        i. Rango de Entrada: 0 a 1023
        ii. Rango de Salida: 0 a 100
        iii. Opción: Escalar y Limitar al Rango de Salida
    c. **Resultado**: Una lectura de 512 se mapeará aproximadamente a un 50%.
2. **Ajuste de Valores PWM a Voltaje**:
    a. **Escenario**: Se tiene un valor PWM en el rango de 0 a 255 y se desea convertirlo a un rango de voltaje de 0V a 5V.
    b. **Configuración**:
        i. Rango de Entrada: 0 a 255
        ii. Rango de Salida: 0 a 5
        iii. Opción: Escalar
    c. **Resultado**: Un valor PWM de 128 se mapeará a aproximadamente 2.5V.
3. **Normalización de Datos para Visualización**:
    a. **Escenario**: Datos de temperatura en el rango de -40°C a 100°C necesitan ser normalizados a un rango de 0 a 1 para visualización en una gráfica.
    b. **Configuración**:
        i. Rango de Entrada: -40 a 100
        ii. Rango de Salida: 0 a 1
        iii. Opción: Escalar y Limitar al Rango de Salida
    c. **Resultado**: Una temperatura de 30°C se mapeará a aproximadamente 0.583.

Consideraciones Importantes

- **Valores Fuera del Rango de Entrada**: Si el valor de entrada está fuera del rango definido, el comportamiento dependerá de la opción de escalado seleccionada. Es importante configurar el nodo según el manejo deseado para estos casos.
- **Precisión**: Al trabajar con valores que requieren alta precisión, considera el impacto de la opción de redondeo y selecciona la configuración que mejor se adapte a tus necesidades.

Nodo Template

El nodo **Template** en Node-RED permite generar contenido dinámico insertando datos en plantillas de texto. Utiliza el lenguaje de plantillas Mustache, que permite incrustar variables y expresiones dentro de una estructura de texto predefinida.

Configuración del Nodo Template

- **Propiedad de Entrada**: Por defecto, el nodo utiliza msg.payload como la fuente de datos para la plantilla, pero se puede configurar para utilizar cualquier otra propiedad del mensaje.
- **Formato de la Plantilla**: El nodo admite diferentes formatos de plantilla, como texto plano, HTML o JSON, dependiendo de la aplicación específica.
- **Salida**: El resultado procesado por la plantilla se puede asignar a msg.payload u otra propiedad especificada. Además, el nodo puede configurarse para analizar la salida como JSON o YAML, convirtiéndola en un objeto JavaScript para su uso posterior en el flujo.

Uso del Lenguaje de Plantillas Mustache

Mustache es un lenguaje de plantillas sin lógica que permite insertar variables y expresiones en el texto. Los marcadores de posición se definen utilizando dobles llaves {{ }}. Por ejemplo:

```
<p>Hola, {{nombre}}!</p>


Si msg.nombre es "Juan", el resultado será:

<p>Hola, Juan!</p>
```

Ejemplos de Uso del Nodo Template

1. **Generación de Páginas HTML Dinámicas**:

El nodo Template se puede utilizar para crear contenido HTML dinámico insertando datos en una estructura HTML predefinida. Por ejemplo:

```
<html>
<head>
  <title>Reporte de Sensores</title>
</head>
<body>
  <h1>Datos del Sensor</h1>
  <p>Temperatura: {{payload.temperatura}} °C</p>
  <p>Humedad: {{payload.humedad}} %</p>
</body>
</html>
```

Si msg.payload contiene { "temperatura": 22, "humedad": 60 }, el nodo generará una página HTML mostrando estos valores.

2. **Creación de Configuraciones JSON**:

El nodo Template puede generar configuraciones JSON dinámicas. Por ejemplo:

```
{
  "dispositivo": "{{payload.dispositivo}}",
  "estado": "{{payload.estado}}"
}
```

Con msg.payload como { "dispositivo": "Sensor1", "estado": "activo" }, el resultado será:

```
{
  "dispositivo": "Sensor1",
  "estado": "activo"
}
```

Además, configurando el nodo para analizar la salida como JSON, se convertirá en un objeto JavaScript utilizable en el flujo.

3. **Uso de Bucles con Mustache**:

Mustache permite iterar sobre arreglos de datos. Por ejemplo, para generar una lista HTML a partir de un arreglo de elementos:

```
<ul>
{{#payload.elementos}}
  <li>{{nombre}}: {{valor}}</li>
{{/payload.elementos}}
</ul>
```

Si msg.payload.elementos es un arreglo de objetos con propiedades nombre y valor, el nodo generará una lista <ul> con cada elemento representado como un <li>.

Consideraciones y Buenas Prácticas

- **Validación de Datos**: Asegúrate de que las propiedades utilizadas en la plantilla existan en el mensaje para evitar resultados inesperados.
- **Seguridad**: Al generar contenido HTML, ten cuidado con la inserción de datos no confiables para prevenir vulnerabilidades como ataques XSS.
- **Legibilidad**: Mantén las plantillas claras y bien documentadas para facilitar su mantenimiento y comprensión.
- **Uso de Comentarios**: Puedes añadir comentarios en las plantillas Mustache utilizando la sintaxis {{! comentario }} para mejorar la documentación interna.

Nodo delay

El nodo **Delay** en Node-RED sirve para controlar el flujo de mensajes, permitiendo introducir retrasos específicos o limitar la tasa de mensajes que pasan a través de él. Esto es particularmente útil en aplicaciones donde es necesario espaciar temporalmente los mensajes o evitar sobrecargar sistemas externos con demasiadas solicitudes en un corto período.

Configuración del Nodo Delay

El nodo Delay ofrece dos modos principales de operación:

1. **Retrasar cada mensaje**: Permite introducir un retraso fijo o aleatorio para cada mensaje que pasa por el nodo.
    a. **Retraso fijo**: Se especifica un tiempo constante en milisegundos, segundos, minutos u horas.
    b. **Retraso aleatorio**: Se define un rango de tiempo, y el nodo aplicará un retraso aleatorio dentro de ese intervalo para cada mensaje.
2. **Limitar la tasa de mensajes**: Controla la frecuencia con la que los mensajes se transmiten, permitiendo establecer un intervalo mínimo entre mensajes.

a. **Aplicar a todos los mensajes**: Se establece una tasa global para todos los mensajes que pasan por el nodo.
b. **Aplicar por cada msg.topic**: Permite mantener tasas independientes para cada tema (topic), útil cuando se manejan múltiples flujos de datos simultáneamente.

Opciones Adicionales

- **Anular retraso con msg.delay**: Si se habilita, el nodo puede utilizar un valor específico proporcionado en msg.delay para determinar el retraso de cada mensaje individualmente.
- **Restablecer (msg.reset)**: Al recibir un mensaje con la propiedad msg.reset establecida, el nodo limpiará todos los mensajes pendientes en la cola sin enviarlos.
- **Liberar mensajes (msg.flush)**: Un mensaje con la propiedad msg.flush puede desencadenar la liberación inmediata de mensajes en la cola. Si msg.flush es un número, se liberará esa cantidad de mensajes; si es de otro tipo, se liberarán todos los mensajes pendientes.
- **Prioridad de mensajes (msg.toFront)**: Si se establece en true, el mensaje actual se coloca al frente de la cola y será el próximo en ser enviado.

Ejemplos de Uso del Nodo Delay

1. **Introducir un Retraso Fijo entre Mensajes**:
    a. **Escenario**: Se desea retrasar cada mensaje en 5 segundos antes de que continúe en el flujo.
    b. **Configuración**:
        i. Modo: Retrasar cada mensaje.
        ii. Retraso fijo: 5 segundos.
    c. **Resultado**: Cada mensaje que pasa por el nodo se retiene durante 5 segundos antes de ser enviado al siguiente nodo.
2. **Limitar la Tasa de Mensajes a Uno por Segundo**:
    a. **Escenario**: Se reciben múltiples mensajes en ráfaga, pero se necesita procesar solo uno por segundo para no sobrecargar un servicio externo.
    b. **Configuración**:
        i. Modo: Limitar la tasa de mensajes.
        ii. Tasa: 1 mensaje por segundo.
    c. **Resultado**: El nodo permite pasar un solo mensaje cada segundo, descartando los mensajes intermedios si llegan más rápido de lo permitido.
3. **Retrasar Mensajes con un Valor Personalizado**:

a. **Escenario**: Cada mensaje contiene una propiedad msg.delay que especifica el tiempo de retraso deseado.
b. **Configuración**:
    i. Modo: Retrasar cada mensaje.
    ii. Habilitar la opción para anular retraso con msg.delay.
c. **Resultado**: El nodo aplica un retraso específico para cada mensaje basado en el valor de msg.delay.

Consideraciones y Buenas Prácticas

- **Manejo de Colas**: Al utilizar el nodo Delay en modo de limitación de tasa, los mensajes que exceden la tasa permitida pueden acumularse en la cola, lo que podría llevar al consumo de memoria si no se gestionan adecuadamente.
- **Uso de msg.reset y msg.flush**: Estas propiedades son útiles para controlar dinámicamente la cola de mensajes, permitiendo restablecer o liberar mensajes según las necesidades del flujo.
- **Aplicaciones en Domótica**: El nodo Delay es especialmente útil en sistemas de automatización del hogar para gestionar temporizaciones, como apagar luces después de un período sin detección de movimiento.

Nodo trigger

El nodo **Trigger** en Node-RED permite controlar la emisión de mensajes en función de eventos específicos, introduciendo retrasos, repeticiones y condiciones de reinicio. Es especialmente útil para gestionar temporizadores, manejar eventos de tiempo y crear automatizaciones que requieren acciones diferidas o repetitivas.

Configuración del Nodo Trigger

El nodo Trigger ofrece diversas opciones de configuración para adaptarse a múltiples escenarios:

1. **Enviar un Mensaje Inicial**:
    a. Al recibir un mensaje, el nodo puede enviar inmediatamente un mensaje con un contenido específico, que puede ser el msg.payload actual, un valor fijo o ningún mensaje.
2. **Esperar y Enviar un Segundo Mensaje**:
    a. Después del mensaje inicial, el nodo puede esperar un período definido y luego enviar un segundo mensaje. Este segundo mensaje puede configurarse con un contenido específico o no enviar nada.
3. **Repetir Mensajes a Intervalos Regulares**:

    a. El nodo puede configurarse para reenviar el mensaje inicial a intervalos regulares hasta que se reciba una señal de reinicio (msg.reset).
4. **Extender el Retraso con Nuevos Mensajes**:
    a. Si se habilita, la llegada de nuevos mensajes reinicia el temporizador de espera, útil para implementar temporizadores de inactividad.
5. **Manejo de Reinicios**:
    a. El nodo puede reiniciarse al recibir un mensaje con la propiedad msg.reset o un msg.payload específico, deteniendo cualquier acción pendiente.
6. **Anulación de Retrasos con msg.delay**:
    a. Permite definir dinámicamente el tiempo de espera utilizando la propiedad msg.delay en milisegundos.

Ejemplos de Uso del Nodo Trigger

1. **Automatización de Luces con Sensores de Presencia**:
    a. **Escenario**: Encender una luz al detectar movimiento y apagarla después de un período sin actividad.
    b. **Configuración**:
        i. Al detectar movimiento, el nodo envía un mensaje para encender la luz.
        ii. Luego, espera un tiempo definido (por ejemplo, 5 minutos) y envía un mensaje para apagar la luz.
        iii. Si se detecta movimiento adicional durante el período de espera, el temporizador se reinicia, asegurando que la luz permanezca encendida mientras haya actividad.
    c. **Referencia**:
2. **Manejo de Timeouts en Flujos**:
    a. **Escenario**: Detectar la ausencia de mensajes en un período específico y desencadenar una acción.
    b. **Configuración**:
        i. El nodo se configura para no enviar un mensaje inicialmente.
        ii. Espera un tiempo definido (por ejemplo, 10 segundos) y, si no recibe nuevos mensajes durante ese período, envía un mensaje de "timeout".
        iii. Si llega un nuevo mensaje antes de que expire el tiempo, el temporizador se reinicia, evitando el envío del mensaje de "timeout".
    c. **Referencia**:
3. **Generación de Pulsos para Dispositivos**:
    a. **Escenario**: Enviar un pulso de activación a un dispositivo y luego desactivarlo después de un breve período.

b. **Configuración**:
    i. Al recibir un mensaje, el nodo envía un mensaje para activar el dispositivo.
    ii. Espera un corto período (por ejemplo, 500 ms) y luego envía un mensaje para desactivar el dispositivo, creando un pulso controlado.

Consideraciones y Buenas Prácticas

- **Gestión de Temporizadores**: El nodo Trigger es ideal para implementar temporizadores y manejar eventos basados en tiempo dentro de los flujos de Node-RED.
- **Prevención de Condiciones de Carrera**: Al manejar eventos asincrónicos, asegúrate de que los temporizadores y reinicios estén configurados correctamente para evitar comportamientos inesperados.
- **Optimización de Recursos**: Utiliza las opciones de reinicio y extensión de temporizadores para evitar la sobrecarga de mensajes y garantizar un flujo eficiente.

Nodo exec

El nodo **Exec** en Node-RED permite ejecutar comandos del sistema operativo directamente desde un flujo, integrando los resultados en el procesamiento de datos. Esto es útil para interactuar con programas externos, scripts personalizados o utilidades del sistema.

Configuración del Nodo Exec

El nodo Exec tiene una entrada y tres salidas:

- **Entrada**: Recibe mensajes que pueden activar la ejecución del comando configurado.
- **Salidas**:
    - **Salida estándar (stdout)**: Devuelve la salida estándar del comando ejecutado.
    - **Error estándar (stderr)**: Proporciona cualquier mensaje de error generado durante la ejecución.
    - **Código de retorno**: Entrega un objeto con el código de retorno del comando, indicando su estado de finalización.

En la configuración del nodo, se puede especificar el comando a ejecutar. Opcionalmente, se puede permitir que el comando sea definido dinámicamente a través de msg.payload. Además, es posible elegir entre dos modos de ejecución:

- **Exec**: Espera a que el comando finalice antes de devolver la salida completa.
- **Spawn**: Devuelve la salida en tiempo real, línea por línea, útil para comandos que generan salidas continuas.

Ejemplos de Uso del Nodo Exec

1. **Ejecutar un Comando Simple**:
    a. **Escenario**: Obtener la fecha y hora actuales del sistema.
    b. **Configuración**:
        i. Comando: date
    c. **Resultado**: La salida estándar proporcionará la fecha y hora actuales.
2. **Pingar una Dirección IP**:
    a. **Escenario**: Verificar la conectividad con google.com.
    b. **Configuración**:
        i. Comando: ping -c 4 google.com
    c. **Resultado**: La salida estándar mostrará los resultados del ping.
3. **Ejecutar un Script Personalizado**:
    a. **Escenario**: Ejecutar un script Python llamado myscript.py.
    b. **Configuración**:
        i. Comando: python3 /ruta/a/myscript.py
    c. **Resultado**: La salida estándar contendrá cualquier resultado impreso por el script.

Consideraciones y Buenas Prácticas

- **Seguridad**: Asegúrate de que los comandos ejecutados no comprometan la seguridad del sistema. Evita ejecutar comandos con privilegios elevados a menos que sea absolutamente necesario.
- **Manejo de Errores**: Utiliza la segunda salida para capturar errores y manejar situaciones en las que el comando no se ejecute correctamente.
- **Ejecución Asíncrona**: Si el comando puede tardar en ejecutarse, considera utilizar el modo "spawn" para procesar la salida en tiempo real y evitar bloquear el flujo.
- **Permisos**: Verifica que el usuario bajo el cual se ejecuta Node-RED tenga los permisos necesarios para ejecutar los comandos o scripts deseados.
- **Variables Dinámicas**: Puedes construir comandos dinámicamente utilizando propiedades del mensaje, lo que permite una mayor flexibilidad en la ejecución de diferentes comandos basados en la lógica del flujo.

Nodo filter

El nodo **Filter** en Node-RED, anteriormente conocido como "Report by Exception" (RBE), es una herramienta esencial para controlar el flujo de mensajes basándose en cambios específicos en los datos. Permite optimizar la transmisión de datos al asegurar que solo se envíen mensajes cuando ocurren variaciones significativas, reduciendo así el tráfico innecesario y mejorando la eficiencia del sistema.

Modos de Operación del Nodo Filter

El nodo Filter ofrece dos modos principales de operación:

1. **Modo Deadband**:
    a. **Descripción**: En este modo, el nodo transmite datos únicamente cuando el valor medido cambia más allá de un umbral específico, conocido como "deadband". Este umbral puede definirse como un valor absoluto o como un porcentaje del rango de medición.
    b. **Aplicación**: Ideal para evitar transmisiones frecuentes cuando los valores fluctúan mínimamente alrededor de un punto fijo. Por ejemplo, en un sensor de temperatura con un rango de 0-100°C y un deadband establecido en 2°C, el sistema solo reportará cambios de temperatura superiores a 2°C.
2. **Modo Narrowband**:
    a. **Descripción**: En este modo, el nodo transmite datos únicamente cuando el valor medido se encuentra fuera de un rango específico, denominado "narrowband" o histéresis. Este rango puede definirse en valores absolutos o porcentuales.
    b. **Aplicación**: Útil para evitar transmisiones innecesarias cuando los valores permanecen dentro de un rango aceptable. Por ejemplo, si el narrowband se establece en ±5°C, el sistema solo reportará cambios de temperatura que excedan este rango en relación con el último valor reportado.

Configuración del Nodo Filter

- **Propiedad a Monitorear**: Por defecto, el nodo evalúa msg.payload, pero puede configurarse para supervisar cualquier otra propiedad del mensaje.
- **Umbral de Cambio**: Se define el valor absoluto o porcentual que determinará cuándo se debe transmitir un nuevo mensaje.
- **Comparación con Valor Anterior**: El nodo puede configurarse para comparar el valor actual con el último valor válido enviado o con el último valor recibido, permitiendo adaptarse a diferentes escenarios de monitoreo.

- **Aplicación por Tópico (msg.topic)**: Si se especifica, el nodo opera de manera independiente para cada tópico, lo que es útil al manejar múltiples flujos de datos simultáneamente.
- **Reinicio (msg.reset)**: Al recibir un mensaje con la propiedad msg.reset, el nodo restablece el valor almacenado para el tópico especificado o para todos los tópicos si no se especifica ninguno.

Ejemplos de Uso del Nodo Filter

1. **Reporte de Cambios Significativos en un Sensor de Temperatura**:
    a. **Escenario**: Un sensor de temperatura envía lecturas constantes, pero solo se desean reportar cambios superiores al 5% respecto al último valor reportado.
    b. **Configuración**:
        i. Modo: Deadband.
        ii. Umbral: 5% en comparación con el último valor válido enviado.
    c. **Resultado**: El nodo solo transmitirá mensajes cuando la temperatura cambie en más del 5% respecto al último valor reportado, reduciendo así el tráfico de datos.
2. **Monitoreo de Fluctuaciones en un Sensor de Humedad**:
    a. **Escenario**: Se desea recibir alertas únicamente cuando la humedad relativa salga de un rango aceptable de ±3% respecto al último valor medido.
    b. **Configuración**:
        i. Modo: Narrowband.
        ii. Umbral: ±3% en comparación con el último valor recibido.
    c. **Resultado**: El nodo filtrará las lecturas de humedad, permitiendo el paso solo de aquellas que excedan el rango de ±3%, lo que ayuda a identificar cambios ambientales significativos.
3. **Filtrado de Datos de Múltiples Sensores por Tópico**:
    a. **Escenario**: Se reciben datos de varios sensores, cada uno con un msg.topic diferente, y se desea filtrar los cambios significativos para cada sensor de manera independiente.
    b. **Configuración**:
        i. Modo: Deadband o Narrowband, según la necesidad.
        ii. Aplicar filtrado por msg.topic.
    c. **Resultado**: El nodo gestionará y filtrará los datos de cada sensor de forma independiente, asegurando que solo se transmitan los cambios relevantes para cada uno.

Consideraciones y Buenas Prácticas

- **Optimización de Tráfico**: El uso adecuado del nodo Filter puede reducir significativamente el tráfico de datos, especialmente en redes con recursos limitados o costos asociados al volumen de datos transmitidos.
- **Ajuste de Umbrales**: Es importante configurar los umbrales de deadband o narrowband de acuerdo con la sensibilidad requerida para la aplicación específica, evitando tanto la sobrecarga de datos como la omisión de cambios importantes.
- **Monitoreo de Múltiples Tópicos**: Al manejar múltiples fuentes de datos, utilizar la funcionalidad de msg.topic permite un filtrado más preciso y adaptado a cada flujo de información.
- **Reinicio de Valores**: La capacidad de reiniciar los valores almacenados mediante msg.reset es útil para recalibrar el sistema o adaptarse a cambios en las condiciones de operación.

Nodo Random

El nodo **Random** en Node-RED genera números aleatorios dentro de un rango específico cada vez que se activa. Esto es útil para simular datos, realizar pruebas o introducir variabilidad en los flujos de trabajo.

Instalación del Nodo Random

Para utilizar el nodo Random, es necesario instalar el paquete correspondiente. Puedes hacerlo mediante la opción "Manage Palette" en el menú del editor de Node-RED o ejecutando el siguiente comando en el directorio de usuario de Node-RED, típicamente ~/.node-red:

npm install node-red-node-random

Configuración del Nodo Random

Una vez instalado, puedes configurar el nodo Random según tus necesidades:

- **Rango de Valores**: Define los valores mínimo y máximo entre los cuales se generará el número aleatorio.
- **Tipo de Número**:
    - **Entero (Integer)**: Genera un número entero dentro del rango especificado, incluyendo los valores límite. Por ejemplo, un rango de 1 a 6 puede producir 1, 2, 3, 4, 5 o 6.
    - **Punto Flotante (Float)**: Genera un número decimal desde el valor mínimo hasta, pero sin incluir, el valor máximo. Por ejemplo, un rango de 1 a 6 puede producir valores como 1.5, 3.8, etc.

- **Valores Dinámicos**: Es posible proporcionar dinámicamente los valores "Desde" y "Hasta" utilizando msg.from y msg.to en el mensaje de entrada. Sin embargo, los valores configurados directamente en el nodo tienen prioridad sobre estos.

Ejemplos de Uso del Nodo Random

1. **Simulación de Lanzamiento de un Dado**:
    a. **Configuración**:
        i. Rango: 1 a 6
        ii. Tipo: Entero
    b. **Resultado**: Cada activación del nodo generará un número entero entre 1 y 6, simulando el lanzamiento de un dado.
2. **Generación de Temperaturas Aleatorias para Pruebas**:
    a. **Configuración**:
        i. Rango: 15.0 a 25.0
        ii. Tipo: Punto Flotante
    b. **Resultado**: Cada activación producirá un número decimal entre 15.0 y 25.0, útil para simular lecturas de temperatura en pruebas de sistemas HVAC.
3. **Valores Dinámicos Basados en la Entrada**:
    a. **Configuración**:
        i. Rango: Determinado por msg.from y msg.to
    b. **Resultado**: El nodo generará un número aleatorio dentro del rango especificado por las propiedades del mensaje de entrada, permitiendo flexibilidad en diferentes contextos.

Consideraciones y Buenas Prácticas

- **Validación de Rango**: Asegúrate de que los valores mínimo y máximo estén correctamente definidos para evitar resultados inesperados.
- **Uso en Pruebas**: El nodo Random es especialmente útil para generar datos de prueba en flujos de Node-RED, facilitando la simulación de diversas condiciones sin necesidad de fuentes de datos externas.

Nodo smooth

El nodo **Smooth** en Node-RED sirve para aplicar algoritmos de suavizado a datos numéricos entrantes, permitiendo obtener valores como el mínimo, máximo, media, desviación estándar y aplicar filtros de paso alto y bajo. Esto es especialmente útil para procesar señales de sensores, eliminar ruido y obtener tendencias más claras en los datos.

Instalación del Nodo Smooth

Para utilizar el nodo Smooth, es necesario instalar el paquete correspondiente. Puedes hacerlo mediante la opción "Manage Palette" en el menú del editor de Node-RED o ejecutando el siguiente comando en el directorio de usuario de Node-RED, típicamente ~/.node-red:

npm install node-red-node-smooth

Configuración del Nodo Smooth

Una vez instalado, puedes configurar el nodo Smooth según tus necesidades:

- **Propiedad a Procesar**: Por defecto, el nodo opera sobre msg.payload, pero puedes especificar otra propiedad del mensaje si es necesario.
- **Acción**: Selecciona la operación que deseas realizar:
    - **Min**: Calcula el valor mínimo de una ventana de datos.
    - **Max**: Calcula el valor máximo de una ventana de datos.
    - **Mean**: Calcula la media (promedio) de una ventana de datos.
    - **Standard Deviation**: Calcula la desviación estándar de una ventana de datos.
    - **Low Pass**: Aplica un filtro de paso bajo para suavizar los datos, permitiendo el paso de señales de baja frecuencia y atenuando las de alta frecuencia.
    - **High Pass**: Aplica un filtro de paso alto, permitiendo el paso de señales de alta frecuencia y atenuando las de baja frecuencia.
- **Número de Muestras**: Especifica la cantidad de valores anteriores que el nodo debe considerar para las operaciones de Min, Max, Mean y Standard Deviation. Estas operaciones se realizan sobre una ventana móvil basada en el número de muestras especificado.
- **Factor de Suavizado**: Para los filtros de paso alto y bajo, define un factor de suavizado. Un valor más alto implica una mayor suavización. Por ejemplo, un valor de 10 es similar a un α de 0.1. Este factor es análogo a una constante de tiempo RC, pero en este caso se basa en eventos que llegan, no en el tiempo real.
- **Reinicio de Valores**: Si el nodo recibe un mensaje con la propiedad msg.reset, restablecerá todos los contadores y valores intermedios a su estado inicial, comenzando de nuevo el proceso de acumulación de datos.

Ejemplos de Uso del Nodo Smooth

1. **Cálculo de la Media Móvil de Temperaturas**:

    a. **Escenario**: Tienes un sensor que envía lecturas de temperatura cada segundo y deseas calcular la media de las últimas 10 lecturas para suavizar las fluctuaciones rápidas.
    b. **Configuración**:
        i. Acción: Mean
        ii. Número de Muestras: 10
    c. **Resultado**: El nodo emitirá la media de las últimas 10 lecturas de temperatura, proporcionando una visión más estable de las tendencias de temperatura.
2. **Aplicación de un Filtro de Paso Bajo a Señales de Ruido**:
    a. **Escenario**: Recibes datos de un sensor con mucho ruido y deseas suavizar la señal para obtener una lectura más clara.
    b. **Configuración**:
        i. Acción: Low Pass
        ii. Factor de Suavizado: 10
    c. **Resultado**: El nodo aplicará un filtro de paso bajo, atenuando las variaciones rápidas y permitiendo una señal más suave y clara.
3. **Detección de Picos en Datos de Humedad**:
    a. **Escenario**: Deseas identificar picos en las lecturas de humedad para detectar eventos inusuales.
    b. **Configuración**:
        i. Acción: Max
        ii. Número de Muestras: 5
    c. **Resultado**: El nodo emitirá el valor máximo de las últimas 5 lecturas, ayudando a identificar picos recientes en los datos de humedad.

Consideraciones y Buenas Prácticas

- **Datos Numéricos**: El nodo Smooth opera únicamente con datos numéricos. Si recibe otro tipo de datos, intentará convertirlos a números y los rechazará si la conversión falla.
- **Selección de la Acción Adecuada**: Elige la acción que mejor se adapte a tus necesidades de procesamiento de datos. Por ejemplo, utiliza "Mean" para obtener un promedio, "Low Pass" para suavizar señales ruidosas o "Max" para detectar picos.
- **Ajuste del Número de Muestras y Factor de Suavizado**: Configura estos parámetros según la naturaleza de tus datos y la sensibilidad deseada. Un mayor número de muestras o un factor de suavizado más alto resultará en una respuesta más lenta pero más suave.
- **Reinicio de Valores**: Utiliza msg.reset para restablecer el estado del nodo cuando sea necesario, especialmente si los datos entrantes cambian de contexto o si deseas reiniciar el cálculo de las estadísticas.

Nodo Split

El nodo **Split** en Node-RED es una herramienta fundamental que permite dividir un mensaje entrante en múltiples mensajes basados en reglas definidas. Esto facilita el procesamiento individual de elementos dentro de estructuras de datos complejas, como cadenas de texto, matrices u objetos, permitiendo una manipulación más granular y eficiente de la información en los flujos.

Funcionamiento del Nodo Split

El comportamiento del nodo Split varía según el tipo de dato presente en msg.payload:

- **Cadenas de texto (strings)**: El nodo puede dividir la cadena utilizando un delimitador específico (por defecto, el carácter de nueva línea \n) o en longitudes fijas, generando un mensaje separado por cada segmento resultante.
- **Matrices (arrays)**: Cada elemento de la matriz se convierte en un mensaje individual. Es posible configurar el nodo para que divida la matriz en subgrupos de longitud fija, emitiendo un mensaje por cada subgrupo.
- **Objetos**: El nodo puede descomponer el objeto en pares clave-valor, emitiendo un mensaje por cada par, lo que facilita el procesamiento de cada propiedad por separado.

Cada mensaje generado por el nodo Split incluye una propiedad msg.parts que contiene información sobre cómo se dividió el mensaje original, lo cual es esencial para operaciones posteriores, como la recombinación de mensajes utilizando el nodo Join.

Ejemplos de Uso del Nodo Split

1. **División de una Cadena de Texto en Líneas Individuales**:
    a. **Escenario**: Se dispone de un bloque de texto con múltiples líneas, y se requiere procesar cada línea por separado.
    b. **Configuración**:
        i. Tipo de dato: Cadena de texto.
        ii. Delimitador: \n (nueva línea).
    c. **Resultado**: El nodo Split generará un mensaje por cada línea de texto, permitiendo su procesamiento individual.
2. **Procesamiento de Elementos de una Matriz**:
    a. **Escenario**: Se recibe una matriz de datos y se necesita aplicar una operación específica a cada elemento.
    b. **Configuración**:
        i. Tipo de dato: Matriz.

        ii. Modo de división: Elementos individuales.
    c. **Resultado**: El nodo emitirá un mensaje por cada elemento de la matriz, facilitando su procesamiento individual en el flujo.
3. **Descomposición de un Objeto en Pares Clave-Valor**:
    a. **Escenario**: Se dispone de un objeto con múltiples propiedades y se requiere tratar cada par clave-valor por separado.
    b. **Configuración**:
        i. Tipo de dato: Objeto.
    c. **Resultado**: El nodo Split generará un mensaje por cada par clave-valor, permitiendo operaciones específicas en cada propiedad del objeto.

Consideraciones y Buenas Prácticas

- **Uso del Nodo Join**: Después de dividir mensajes con el nodo Split, es común recombinarlos utilizando el nodo Join. Asegúrate de configurar correctamente el nodo Join para que coincida con la estructura de los mensajes divididos, permitiendo una recombinación adecuada.
- **Manejo de la Propiedad msg.parts**: Esta propiedad es crucial para rastrear la relación entre los mensajes divididos y el mensaje original. No la modifiques manualmente, ya que es utilizada por el nodo Join para recombinar los mensajes correctamente.
- **Configuración Adecuada del Delimitador**: Al dividir cadenas de texto, selecciona el delimitador que corresponda con la estructura de tus datos (por ejemplo, comas, punto y coma, espacios) para asegurar una división precisa.
- **Procesamiento de Grandes Volúmenes de Datos**: Al dividir grandes matrices u objetos, considera el impacto en el rendimiento y la memoria. Asegúrate de que el sistema pueda manejar el número de mensajes generados sin afectar la estabilidad del flujo.

Nodo join

El nodo **Join** en Node-RED permite combinar múltiples mensajes entrantes en un único mensaje, facilitando la integración y el procesamiento conjunto de datos provenientes de diversas fuentes. Su uso es particularmente útil cuando se requiere consolidar información dispersa para análisis, almacenamiento o visualización.

Modos de Operación del Nodo Join

El nodo Join ofrece varios modos de operación que permiten adaptarse a diferentes necesidades:

1. **Modo Automático**:

    a. **Descripción**: Este modo se utiliza en conjunto con el nodo Split para recombinar mensajes que previamente fueron divididos. El nodo Join reconoce automáticamente las propiedades de los mensajes para reconstruir la estructura original.
2. **Modo Manual**:
    a. **Descripción**: Permite configurar manualmente cómo se deben combinar los mensajes entrantes. Ofrece opciones para crear:
        i. **Cadena o Buffer**: Concatenando propiedades específicas de cada mensaje con un delimitador definido.
        ii. **Array**: Agregando propiedades seleccionadas o mensajes completos a una lista.
        iii. **Objeto Clave/Valor**: Utilizando una propiedad del mensaje (como msg.topic) como clave y otra como valor.
        iv. **Objeto Combinado**: Fusionando propiedades de múltiples mensajes en un solo objeto.
3. **Modo Reducir Secuencia**:
    a. **Descripción**: Aplica una expresión JSONata a una secuencia de mensajes para reducirlos a un único resultado acumulado. Es útil para operaciones como sumar valores o calcular promedios.

Configuración del Nodo Join

Al configurar el nodo Join, es importante considerar:

- **Número de Mensajes a Combinar**: Especifica cuántos mensajes deben recibirse antes de generar el mensaje combinado.
- **Tiempo de Espera**: Define un tiempo límite para recibir los mensajes necesarios. Si se excede este tiempo, el nodo enviará el mensaje combinado con los datos disponibles hasta ese momento.
- **Propiedad msg.complete**: Al recibir un mensaje con esta propiedad establecida, el nodo finalizará la combinación y enviará el mensaje resultante, independientemente de si se han recibido todos los mensajes esperados.
- **Propiedad msg.reset**: Permite restablecer el estado interno del nodo, descartando cualquier mensaje acumulado y reiniciando el proceso de combinación.

Ejemplos de Uso del Nodo Join

1. **Combinación de Lecturas de Sensores**:
    a. **Escenario**: Tres sensores (temperatura, humedad y presión) envían datos por separado, y se desea combinarlos en un único mensaje para su almacenamiento en una base de datos.
    b. **Configuración**:

        i. Modo: Manual.
        ii. Crear: Objeto Clave/Valor.
        iii. Clave: msg.topic.
        iv. Valor: msg.payload.
        v. Enviar el mensaje después de recibir 3 partes.
    c. **Resultado**: El nodo Join emitirá un mensaje con una estructura como:{
       "temperature": 22.5,
       "humidity": 60,
       "pressure": 1013
      }

2. **Reconstrucción de Datos Divididos**:
    a. **Escenario**: Un mensaje JSON grande se ha dividido en partes más pequeñas utilizando el nodo Split, y se requiere recomponerlo.
    b. **Configuración**:
        i. Modo: Automático.
    c. **Resultado**: El nodo Join reconstruirá el mensaje original a partir de las partes recibidas, siempre que contengan las propiedades adecuadas generadas por el nodo Split.
3. **Cálculo de Promedio de Valores**:
    a. **Escenario**: Se reciben múltiples lecturas de un sensor, y se desea calcular el promedio de los valores.
    b. **Configuración**:
        i. Modo: Reducir Secuencia.
        ii. Expresión de Reducción: $A + payload.
        iii. Valor Inicial: 0.
        iv. Expresión de Ajuste: $A / $N.
    c. **Resultado**: El nodo Join emitirá un mensaje con el valor promedio de las lecturas recibidas.

<u>Consideraciones y Buenas Prácticas</u>

- **Sincronización de Mensajes**: Asegúrate de que los mensajes que deseas combinar lleguen dentro del tiempo esperado y contengan las propiedades necesarias para la combinación.
- **Uso de msg.topic**: Asignar valores únicos a msg.topic en diferentes flujos facilita la identificación y combinación adecuada de mensajes en el nodo Join.
- **Manejo de Errores**: Implementa lógica adicional para manejar situaciones donde no se reciban todos los mensajes esperados, evitando resultados incompletos o incorrectos.

Nodo sort

El nodo **Sort** en Node-RED es una herramienta esencial para organizar datos de manera eficiente dentro de tus flujos. Permite ordenar arrays contenidos en propiedades de mensajes o secuencias de mensajes en función de criterios específicos, facilitando el procesamiento y análisis de la información.

Modos de Operación del Nodo Sort

1. **Ordenar una Propiedad de Mensaje que Contiene un Array**:
    a. **Descripción**: Este modo permite ordenar un array presente en una propiedad específica del mensaje, como msg.payload.
    b. **Configuración**:
        i. **Propiedad a Ordenar**: Especifica la propiedad del mensaje que contiene el array a ordenar.
        ii. **Clave de Ordenación**:
            1. **Valor del Elemento**: Ordena los elementos del array según su valor directo.
            2. **Expresión JSONata**: Permite definir una expresión para determinar el valor por el cual se ordenará cada elemento.
        iii. **Orden**:
            1. **Ascendente**: Ordena de menor a mayor.
            2. **Descendente**: Ordena de mayor a menor.
        iv. **Tratar como Números**: Si se activa, los valores se interpretan como números para la ordenación; de lo contrario, se consideran cadenas de texto.
2. **Ordenar una Secuencia de Mensajes**:
    a. **Descripción**: Este modo permite ordenar una serie de mensajes individuales basándose en una propiedad específica de cada mensaje.
    b. **Configuración**:
        i. **Clave de Ordenación**:
            1. **Propiedad del Mensaje**: Especifica la propiedad del mensaje utilizada para la ordenación.
            2. **Expresión JSONata**: Define una expresión para determinar el valor de ordenación.
        ii. **Orden**:
            1. **Ascendente**: Ordena de menor a mayor.
            2. **Descendente**: Ordena de mayor a menor.
        iii. **Tratar como Números**: Si se activa, los valores se interpretan como números para la ordenación.

Ejemplos de Uso del Nodo Sort

1. **Ordenar una Lista de Números en Orden Ascendente**:
    a. **Escenario**: Tienes un array de números en msg.payload y deseas ordenarlos de menor a mayor.
    b. **Configuración**:
        i. **Propiedad a Ordenar**: msg.payload
        ii. **Clave de Ordenación**: Valor del Elemento
        iii. **Orden**: Ascendente
        iv. **Tratar como Números**: Activado
    c. **Resultado**: El array en msg.payload se ordenará en orden ascendente numérico.
2. **Ordenar una Lista de Objetos por una Propiedad Específica**:
    a. **Escenario**: Tienes un array de objetos en msg.payload, cada uno con una propiedad edad, y deseas ordenarlos de mayor a menor edad.
    b. **Configuración**:
        i. **Propiedad a Ordenar**: msg.payload
        ii. **Clave de Ordenación**: Expresión JSONata: edad
        iii. **Orden**: Descendente
        iv. **Tratar como Números**: Activado
    c. **Resultado**: El array de objetos se ordenará de mayor a menor según la propiedad edad.
3. **Ordenar una Secuencia de Mensajes por una Propiedad**:
    a. **Escenario**: Recibes una secuencia de mensajes, cada uno con una propiedad timestamp, y deseas procesarlos en orden cronológico.
    b. **Configuración**:
        i. **Clave de Ordenación**: Propiedad del Mensaje: timestamp
        ii. **Orden**: Ascendente
        iii. **Tratar como Números**: Dependiendo del formato del timestamp (activar si es numérico).
    c. **Resultado**: Los mensajes se ordenarán cronológicamente de acuerdo con la propiedad timestamp.

<u>Consideraciones y Buenas Prácticas</u>

- **Consistencia en los Tipos de Datos**: Asegúrate de que los datos en la propiedad que estás ordenando sean consistentes en tipo (todos números o todas cadenas) para evitar resultados inesperados.
- **Uso de Expresiones JSONata**: Las expresiones JSONata proporcionan flexibilidad para definir claves de ordenación basadas en propiedades dinámicas o cálculos complejos.
- **Manejo de Secuencias de Mensajes**: Al ordenar secuencias de mensajes, es importante que cada mensaje contenga la propiedad msg.parts adecuada, especialmente si se han dividido previamente con el nodo Split.

- **Rendimiento**: Considera el tamaño de los arrays o la cantidad de mensajes a ordenar, ya que ordenar grandes conjuntos de datos puede afectar el rendimiento del flujo.

Nodo Join

El nodo **Batch** en Node-RED permite agrupar secuencias de mensajes en lotes según criterios específicos, facilitando el procesamiento conjunto de múltiples mensajes.

Modos de Operación del Nodo Batch

1. **Agrupar por Número de Mensajes**:
    a. **Descripción**: El nodo agrupa mensajes en lotes basados en una cantidad específica. Por ejemplo, al configurar un tamaño de lote de 5, los primeros 5 mensajes se agruparán y se enviarán juntos, luego los siguientes 5, y así sucesivamente.
    b. **Configuración**:
        i. **Tamaño del Lote**: Número de mensajes por lote.
        ii. **Superposición**: Número de mensajes que se repetirán al inicio del siguiente lote.
2. **Agrupar por Intervalo de Tiempo**:
    a. **Descripción**: El nodo agrupa mensajes que llegan dentro de un intervalo de tiempo específico. Por ejemplo, al establecer un intervalo de 2 segundos, todos los mensajes recibidos en ese período se agruparán y enviarán juntos.
    b. **Configuración**:
        i. **Intervalo de Tiempo**: Duración en segundos para agrupar mensajes.
        ii. **Permitir Secuencias Vacías**: Determina si se debe enviar un mensaje vacío si no se reciben mensajes durante el intervalo.
3. **Concatenar Secuencias**:
    a. **Descripción**: El nodo concatena secuencias de mensajes entrantes en un solo mensaje. Cada mensaje debe tener una propiedad msg.topic y msg.parts que identifique su secuencia.
    b. **Configuración**:
        i. **Lista de Temas**: Especifica los valores de msg.topic para identificar el orden en que se concatenarán las secuencias.

Ejemplos de Uso del Nodo Batch

1. **Agrupación de Lecturas de Sensores para Inserción en InfluxDB**:

    a. **Escenario**: Se reciben lecturas de múltiples sensores y se desea agruparlas en lotes para inserciones eficientes en una base de datos InfluxDB.
    b. **Configuración**:
        i. **Modo**: Agrupar por Intervalo de Tiempo.
        ii. **Intervalo de Tiempo**: 10 segundos.
    c. **Resultado**: El nodo Batch recopila las lecturas durante 10 segundos y las envía como un solo mensaje al nodo de InfluxDB para una inserción por lotes.
2. **Procesamiento de Secuencias de Mensajes con el Nodo Join**:
    a. **Escenario**: Se requiere combinar múltiples mensajes en una secuencia para su procesamiento conjunto.
    b. **Configuración**:
        i. **Modo**: Concatenar Secuencias.
        ii. **Lista de Temas**: Especificar los temas relevantes para la concatenación.
    c. **Resultado**: El nodo Batch concatena las secuencias de mensajes según los temas especificados, y luego el nodo Join puede procesar la secuencia combinada.

Consideraciones y Buenas Prácticas

- **Buffering de Mensajes**: El nodo Batch almacena mensajes internamente para formar los lotes. Es importante considerar la configuración de nodeMessageBufferMaxLength para limitar la cantidad de mensajes que los nodos almacenarán en buffer, evitando un uso excesivo de memoria.
- **Reinicio de Secuencias**: Si se recibe un mensaje con la propiedad msg.reset establecida, el nodo Batch eliminará los mensajes almacenados en buffer y no los enviará, reiniciando el proceso de agrupación.
- **Sincronización con el Nodo Join**: El nodo Batch se utiliza comúnmente junto con el nodo Join para combinar mensajes en secuencias. Cada mensaje en el lote recibe una propiedad msg.parts con un índice y un número de cuenta, que el nodo Join utiliza para crear un objeto fusionado, cadena, array, etc.

Nodo batch

El nodo **Batch** en Node-RED permite agrupar secuencias de mensajes en lotes según criterios específicos, facilitando el procesamiento conjunto de múltiples mensajes.

Modos de Operación del Nodo Batch

1. **Agrupar por Número de Mensajes**:

    a. **Descripción**: El nodo agrupa mensajes en lotes basados en una cantidad específica. Por ejemplo, al configurar un tamaño de lote de 5, los primeros 5 mensajes se agruparán y se enviarán juntos, luego los siguientes 5, y así sucesivamente.
    b. **Configuración**:
        i. **Tamaño del Lote**: Número de mensajes por lote.
        ii. **Superposición**: Número de mensajes que se repetirán al inicio del siguiente lote.
2. **Agrupar por Intervalo de Tiempo**:
    a. **Descripción**: El nodo agrupa mensajes que llegan dentro de un intervalo de tiempo específico. Por ejemplo, al establecer un intervalo de 2 segundos, todos los mensajes recibidos en ese período se agruparán y enviarán juntos.
    b. **Configuración**:
        i. **Intervalo de Tiempo**: Duración en segundos para agrupar mensajes.
        ii. **Permitir Secuencias Vacías**: Determina si se debe enviar un mensaje vacío si no se reciben mensajes durante el intervalo.
3. **Concatenar Secuencias**:
    a. **Descripción**: El nodo concatena secuencias de mensajes entrantes en un solo mensaje. Cada mensaje debe tener una propiedad msg.topic y msg.parts que identifique su secuencia.
    b. **Configuración**:
        i. **Lista de Temas**: Especifica los valores de msg.topic para identificar el orden en que se concatenarán las secuencias.

<u>Ejemplos de Uso del Nodo Batch</u>

1. **Agrupación de Lecturas de Sensores para Inserción en InfluxDB**:
    a. **Escenario**: Se reciben lecturas de múltiples sensores y se desea agruparlas en lotes para inserciones eficientes en una base de datos InfluxDB.
    b. **Configuración**:
        i. **Modo**: Agrupar por Intervalo de Tiempo.
        ii. **Intervalo de Tiempo**: 10 segundos.
    c. **Resultado**: El nodo Batch recopila las lecturas durante 10 segundos y las envía como un solo mensaje al nodo de InfluxDB para una inserción por lotes.
2. **Procesamiento de Secuencias de Mensajes con el Nodo Join**:
    a. **Escenario**: Se requiere combinar múltiples mensajes en una secuencia para su procesamiento conjunto.
    b. **Configuración**:

i. **Modo**: Concatenar Secuencias.
ii. **Lista de Temas**: Especificar los temas relevantes para la concatenación.

c. **Resultado**: El nodo Batch concatena las secuencias de mensajes según los temas especificados, y luego el nodo Join puede procesar la secuencia combinada.

Consideraciones y Buenas Prácticas

- **Buffering de Mensajes**: El nodo Batch almacena mensajes internamente para formar los lotes. Es importante considerar la configuración de nodeMessageBufferMaxLength para limitar la cantidad de mensajes que los nodos almacenarán en buffer, evitando un uso excesivo de memoria.
- **Reinicio de Secuencias**: Si se recibe un mensaje con la propiedad msg.reset establecida, el nodo Batch eliminará los mensajes almacenados en buffer y no los enviará, reiniciando el proceso de agrupación.
- **Sincronización con el Nodo Join**: El nodo Batch se utiliza comúnmente junto con el nodo Join para combinar mensajes en secuencias. Cada mensaje en el lote recibe una propiedad msg.parts con un índice y un número de cuenta, que el nodo Join utiliza para crear un objeto fusionado, cadena, array, etc.
- **Manejo de Tiempos en Node-RED**: Dado que Node-RED utiliza Node.js, que es mayormente monohilo, pueden ocurrir retrasos en la respuesta debido a operaciones como la recolección de basura de memoria. Es importante tener esto en cuenta al configurar intervalos de tiempo en el nodo Batch.

Importar y exportar flujos

Exportar e importar flujos en Node-RED es una funcionalidad clave que permite compartir, respaldar y reutilizar configuraciones de flujos de manera sencilla. A continuación, te explico el proceso:

Exportar Flujos en Node-RED

1. **Seleccionar el Flujo**:
   a. Abre el editor de Node-RED.
   b. Selecciona el flujo que deseas exportar.
   c. Puedes seleccionar nodos específicos, una sección del flujo o toda la pestaña.
2. **Acceder al Menú de Exportación**:
   a. Haz clic en el menú desplegable en la esquina superior derecha del editor (icono de las tres líneas horizontales).
   b. Ve a **"Export"**.
3. **Seleccionar el Alcance de Exportación**:

    a. **Selected nodes**: Exporta solo los nodos seleccionados.
    b. **Current flow**: Exporta todos los nodos de la pestaña actual.
    c. **All flows**: Exporta todos los flujos del proyecto.
4. **Obtener el Código JSON**:
    a. Node-RED generará un bloque de código JSON que representa el flujo seleccionado.
    b. Puedes copiar este código al portapapeles.
5. **Guardar el Código Exportado**:
    a. Puedes pegar el JSON en un archivo de texto o directamente en un sistema de gestión de código fuente (por ejemplo, Git) para respaldarlo.

Importar Flujos en Node-RED

1. **Acceder al Menú de Importación**:
    a. Haz clic en el menú desplegable en la esquina superior derecha del editor.
    b. Ve a **"Import"**.
2. **Insertar el Código JSON**:
    a. Pega el código JSON del flujo que deseas importar en el cuadro de texto.
3. **Seleccionar Ubicación para Importar**:
    a. Decide dónde quieres que se coloque el flujo importado:
        i. **Current Flow**: Agrega los nodos importados a la pestaña activa.
        ii. **New Flow**: Crea una nueva pestaña para el flujo importado.
4. **Finalizar la Importación**:
    a. Haz clic en **"Import"**.
    b. Los nodos aparecerán en el editor y estarán listos para ser usados.

Consejos y Buenas Prácticas

1. **Respaldo Regular**:
    a. Exporta regularmente tus flujos y guárdalos en un sistema de control de versiones como Git para evitar pérdidas de datos.
2. **Cuidado con las Credenciales**:
    a. Al exportar un flujo, las credenciales sensibles (como contraseñas y claves API) no se incluyen en el JSON por razones de seguridad. Si es necesario, debes reconfigurarlas después de la importación.
3. **Verificar la Compatibilidad de Nodos**:
    a. Antes de importar un flujo en un nuevo entorno, asegúrate de que los nodos utilizados estén instalados en ese entorno. De lo contrario, Node-RED indicará que hay nodos desconocidos.

4. **Organización de Flujos**:
    a. Usa nombres descriptivos para tus pestañas y flujos para facilitar la gestión, especialmente cuando trabajas con múltiples flujos o compartes con equipos.
5. **Uso de Subflujos**:
    a. Si utilizas patrones repetitivos, considera convertirlos en subflujos antes de exportarlos. Esto facilita su reutilización y reduce la complejidad.

Grupos y subflows

En Node-RED, **grupos** y **subflujos** son herramientas que ayudan a organizar y estructurar flujos de manera más eficiente, especialmente en proyectos grandes o complejos. Ambos tienen propósitos específicos y ofrecen diferentes beneficios.

Grupos

Un **grupo** en Node-RED es una forma visual de agrupar nodos relacionados dentro de una pestaña del editor. Los grupos ayudan a organizar el diseño y a identificar secciones lógicas de un flujo.

Características de los Grupos

1. **Propósito Visual y Organizativo**:
    a. Los grupos no cambian cómo funcionan los nodos, pero ayudan a mantener los flujos organizados.
    b. Facilitan la comprensión al mostrar relaciones entre nodos que trabajan juntos.
2. **Configuración Compartida**:
    a. Puedes asignar propiedades comunes al grupo, como colores de fondo y bordes, para diferenciarlos visualmente.
    b. Se pueden definir configuraciones compartidas como un valor predeterminado para todos los nodos dentro del grupo.
3. **Colapso y Expansión**:
    a. Los grupos pueden colapsarse para ahorrar espacio visual en flujos grandes, mostrando solo un indicador compacto.
4. **Herencia de Contexto**:
    a. Los nodos dentro de un grupo pueden compartir un contexto local, lo que permite almacenar y acceder a datos compartidos entre nodos del mismo grupo.

Cómo Crear y Usar Grupos

1. Selecciona varios nodos en el editor.
2. Haz clic derecho y elige **"Group"** del menú contextual.
3. Configura el grupo: puedes cambiar su nombre, color y propiedades en la barra lateral de propiedades.

Subflujos

Un **subflujo** es una forma de encapsular una sección del flujo en una unidad reutilizable. Es como una función modular que puedes usar en múltiples lugares dentro de tu proyecto.

Características de los Subflujos

1. **Reutilización**:
    a. Un subflujo puede definirse una vez y utilizarse en diferentes partes del proyecto, mejorando la consistencia y ahorrando tiempo.
2. **Encapsulación**:
    a. Los subflujos encapsulan nodos, lo que reduce el desorden visual y protege los flujos principales de cambios innecesarios.
3. **Parámetros Personalizables**:
    a. Puedes definir entradas y salidas para el subflujo, permitiendo que procese datos específicos y devuelva resultados.
    b. Los subflujos también pueden tener propiedades configurables, como valores predeterminados o parámetros personalizables que puedes modificar según dónde se usen.
4. **Herencia de Contexto**:
    a. Los subflujos tienen su propio contexto local, pero también pueden acceder al contexto global o del flujo principal.

Cómo Crear y Usar Subflujos

1. **Crear un Subflujo**:
    a. Selecciona los nodos que deseas encapsular.
    b. Haz clic derecho y selecciona **"Create Subflow"**.
    c. Asigna un nombre al subflujo y configúralo en la barra lateral.
2. **Editar un Subflujo**:
    a. Ve a la pestaña "Subflows" en la barra lateral.
    b. Haz clic en el subflujo para editar su contenido y configuración.
3. **Usar un Subflujo**:
    a. Arrastra el subflujo desde la barra lateral al área de trabajo.
    b. Configura sus propiedades y conecta entradas y salidas según sea necesario.

Dashboard

1. Instalación de los Nodos del Dashboard

Antes de empezar, asegúrate de que Node-RED esté instalado en tu sistema.

1. **Instalar el Módulo Dashboard**:
    a. Abre el terminal y ejecuta el siguiente comando en el directorio de trabajo de Node-RED:npm install node-red-dashboard

    b. También puedes instalarlo desde el editor de Node-RED:
        i. Haz clic en el menú **Manage Palette**.
        ii. Ve a la pestaña **Install** y busca node-red-dashboard.
        iii. Haz clic en **Install**.

2. Configuración Inicial del Dashboard

1. **Accede al Dashboard**:
    a. El dashboard estará disponible en la URL:http://<IP_DEL_SERVIDOR>:1880/ui

    b. Reemplaza <IP_DEL_SERVIDOR> con la dirección de tu servidor.
2. **Configura el Tab Principal**:
    a. Abre el editor de Node-RED.
    b. Agrega un nodo de dashboard (por ejemplo, un nodo ui_text o ui_chart) al flujo.
    c. En las propiedades del nodo, haz clic en el lápiz junto al campo **Tab** para crear un nuevo tab (pestaña).
3. **Crea un Grupo Dentro del Tab**:
    a. Al configurar el nodo, haz clic en el lápiz junto al campo **Group** para crear un grupo dentro del tab.
    b. Asigna un nombre descriptivo al grupo, como "Sensores" o "Controles".

3. Agregar Elementos Gráficos al Dashboard

Ahora que tienes un tab y un grupo configurados, es hora de añadir elementos visuales al dashboard.

3.1 Mostrar Datos con un Texto Simple

1. Arrastra un nodo **ui_text** desde la paleta al área de trabajo.

2. Configura el nodo:
    a. **Tab/Group**: Selecciona el tab y grupo que creaste.
    b. **Label**: Escribe una etiqueta descriptiva, como "Temperatura".
    c. **Value Format**: Usa {{msg.payload}} para mostrar el valor recibido en msg.payload.
3. Conecta el nodo a una fuente de datos, como un nodo inject o function.

3.2 Gráficas en Tiempo Real

1. Arrastra un nodo **ui_chart** al área de trabajo.
2. Configura el nodo:
    a. **Tab/Group**: Selecciona el tab y grupo correspondiente.
    b. **Chart Type**: Selecciona el tipo de gráfica (por ejemplo, "Line chart").
    c. **X-Axis**: Define el número de puntos visibles.
3. Conecta el nodo a un flujo que envíe datos periódicamente (por ejemplo, desde un nodo inject o un sensor).

3.3 Controles de Usuario

1. **Interruptor (Switch)**:
    a. Usa el nodo **ui_switch** para activar o desactivar algo.
    b. Configura:
        i. **Tab/Group**: Selecciona el tab y grupo.
        ii. **Label**: Escribe "Encender/Apagar".
        iii. **Output**: Define los valores de salida (por ejemplo, true para encendido y false para apagado).
    c. Conecta el nodo a un flujo que controle un dispositivo o función.
2. **Deslizadores (Slider)**:
    a. Usa el nodo **ui_slider** para seleccionar valores dentro de un rango.
    b. Configura:
        i. **Tab/Group**: Selecciona el tab y grupo.
        ii. **Range**: Define el rango mínimo y máximo (por ejemplo, de 0 a 100).
        iii. **Step**: Especifica los pasos (por ejemplo, incrementos de 1).
3. **Botones**:
    a. Usa el nodo **ui_button** para acciones específicas.
    b. Configura:
        i. **Label**: Define el texto del botón (por ejemplo, "Enviar").
        ii. **Payload**: Especifica el valor que enviará el botón (por ejemplo, "start").

3.4 Tablas para Visualizar Datos

1. Arrastra un nodo **ui_table** al área de trabajo.
2. Configura:

a. **Tab/Group**: Selecciona el tab y grupo.
b. **Data Source**: Proporciona datos como un array de objetos. Por ejemplo:[
{"Sensor": "Temperatura", "Valor": 22.5},
{"Sensor": "Humedad", "Valor": 60}
]

4. Personalización del Dashboard

1. **Organización Visual**:
    a. Usa las opciones de diseño del dashboard para cambiar el tamaño y posición de los widgets.
    b. Ajusta las propiedades del grupo para controlar el diseño.
2. **Temas y Apariencia**:
    a. Ve a **Dashboard > Site** en la barra lateral del editor.
    b. Personaliza:
        i. **Tema**: Selecciona un tema claro u oscuro.
        ii. **Fuente**: Cambia el tamaño y tipo de fuente.
        iii. **Color**: Define colores para tabs, widgets y fondo.