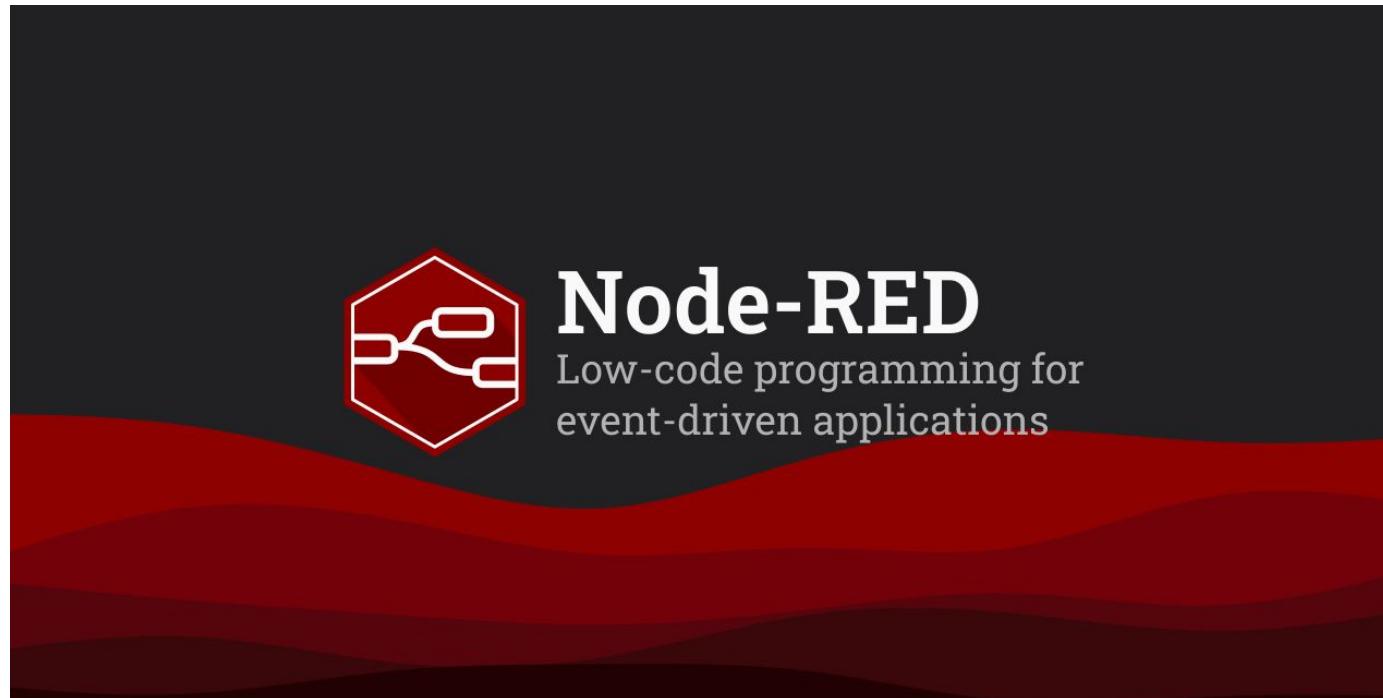


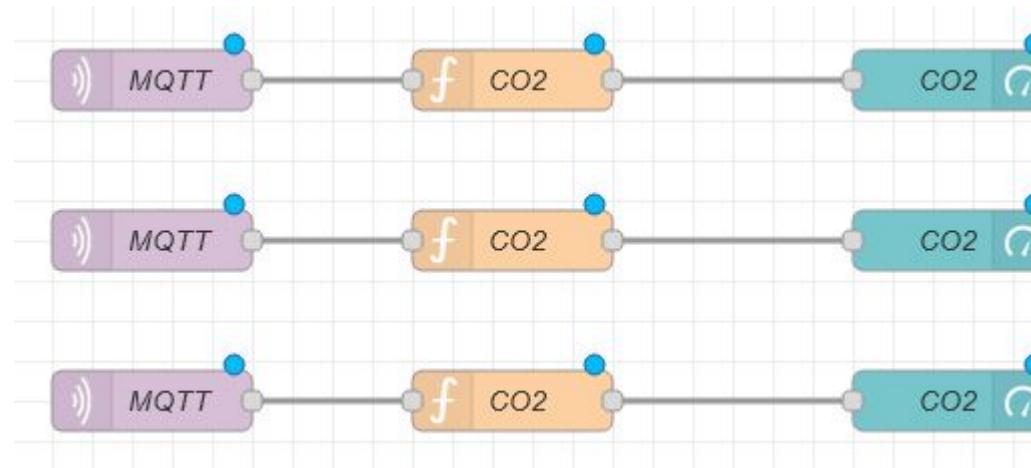
## NODE RED

Node-RED es una herramienta de desarrollo basada en flujo, diseñada para conectar dispositivos de hardware, APIs y servicios en línea de una manera visual e intuitiva. Fue desarrollada originalmente por IBM y es especialmente popular en entornos de Internet de las Cosas (IoT) debido a su facilidad para integrar dispositivos y sistemas diversos.



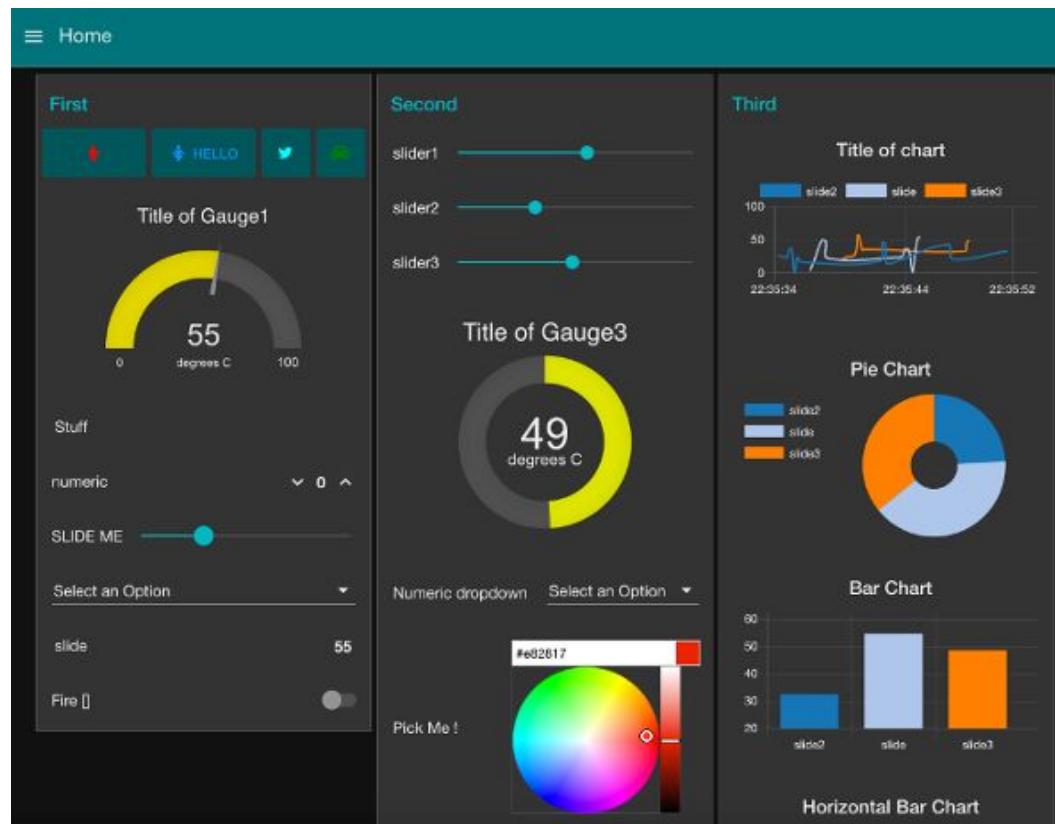
# NODE RED

Node-RED es una herramienta de desarrollo basada en flujo, diseñada para conectar dispositivos de hardware, APIs y servicios en línea de una manera visual e intuitiva. Fue desarrollada originalmente por IBM y es especialmente popular en entornos de Internet de las Cosas (IoT) debido a su facilidad para integrar dispositivos y sistemas diversos.



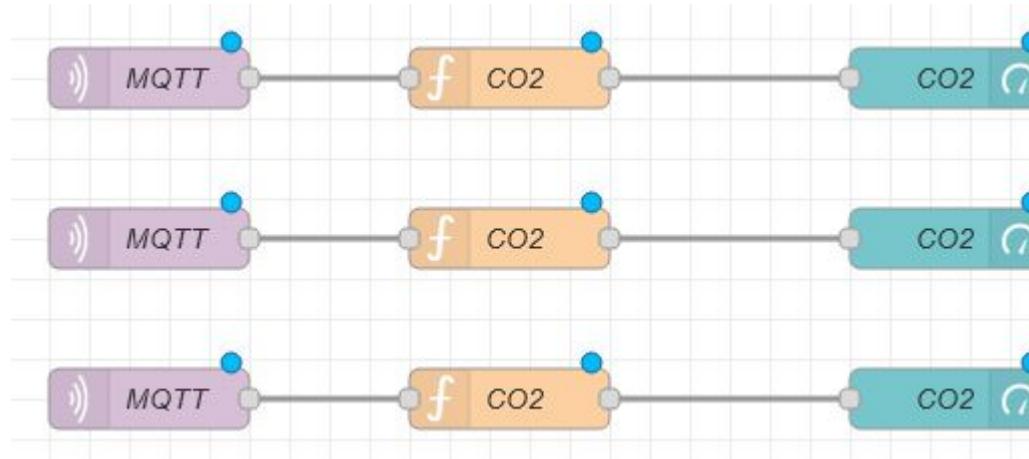
# NODE RED

Node-RED es una herramienta de desarrollo basada en flujo, diseñada para conectar dispositivos de hardware, APIs y servicios en línea de una manera visual e intuitiva. Fue desarrollada originalmente por IBM y es especialmente popular en entornos de Internet de las Cosas (IoT) debido a su facilidad para integrar dispositivos y sistemas diversos.



# CARACTERÍSTICAS

1. Interfaz visual de programación
2. Basado en JavaScript
3. Gran biblioteca de nodos
4. Desarrollo y despliegue rápidos



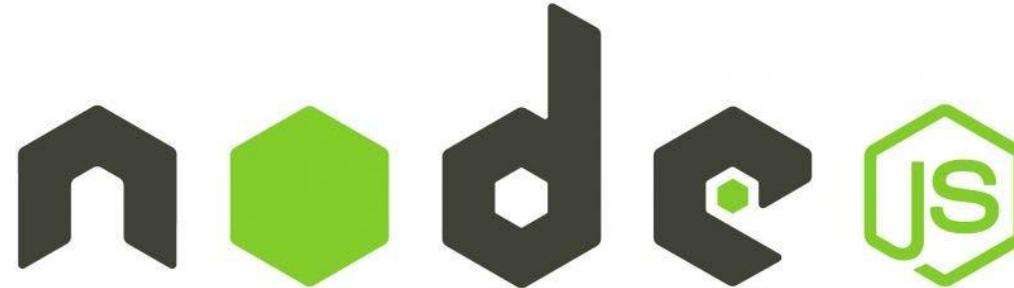
# APLICACIONES

- Automatización de hogares
- Monitorización de sistemas IoT
- Integración de APIs
- Procesamiento de datos en tiempo real



## NODE JS

Node.js es un entorno de ejecución para JavaScript en el lado del servidor, que permite crear aplicaciones de alto rendimiento y escalables. Se basa en un modelo de ejecución asíncrono y orientado a eventos, ideal para manejar múltiples conexiones simultáneas sin bloquear recursos.



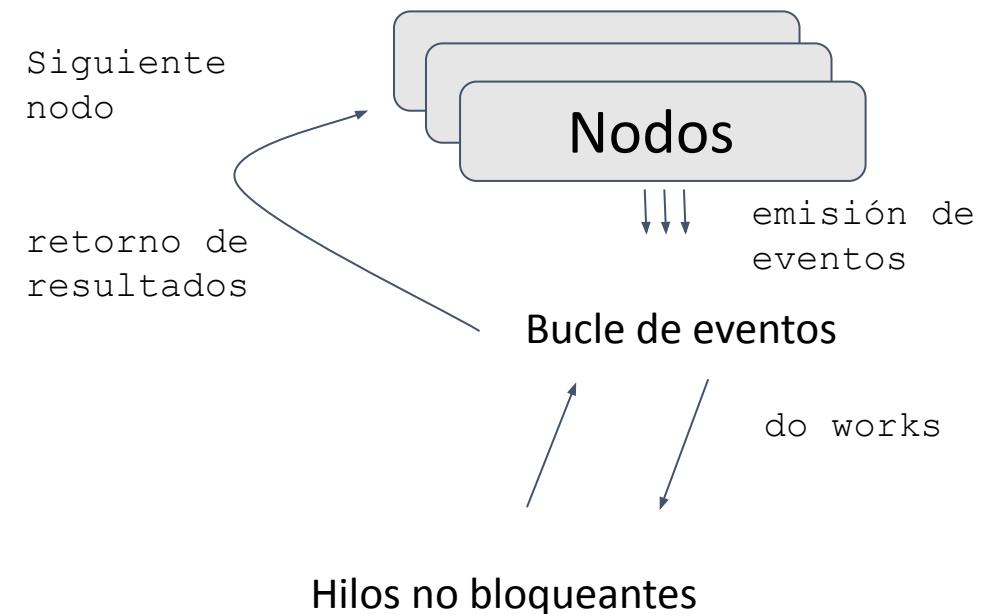
# ARQUITECTURA DE NODE RED

## Siguiente nodo:

Cuando se ejecuta un flujo en Node-RED, cada nodo verifica cuál es el siguiente nodo en el flujo para procesar la información. Este paso es importante porque permite que los nodos sigan una secuencia lógica de ejecución en función del diseño del flujo.

## Emisión de eventos:

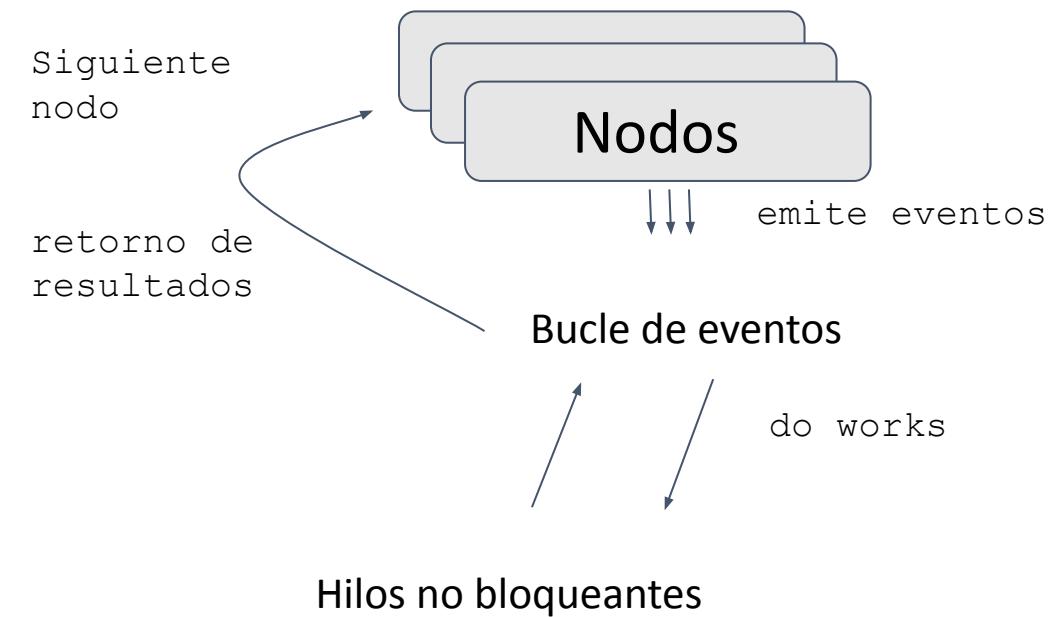
Después de identificar el siguiente nodo en el flujo, el nodo actual emite un evento para notificar al sistema de que la información o tarea está lista para pasar al siguiente nodo. Node-RED utiliza un modelo de eventos basado en el `EventEmitter` de Node.js, lo que permite que los nodos se comuniquen de forma asíncrona y eficiente sin bloquear el sistema.



# ARQUITECTURA DE NODE RED

## Realización de trabajo ("do work"):

Aquí es donde se realiza el procesamiento principal en el nodo. Si el trabajo es sencillo y rápido, se ejecuta directamente en el hilo principal (el bucle de eventos). Sin embargo, si se trata de una operación que toma mucho tiempo, como una operación de entrada/salida (I/O) o el procesamiento de datos intensivo, Node-RED delega esta tarea a **hilos de trabajo no bloqueantes**. Estos hilos permiten que el flujo continúe sin que el hilo principal quede "congelado" esperando que termine la operación.



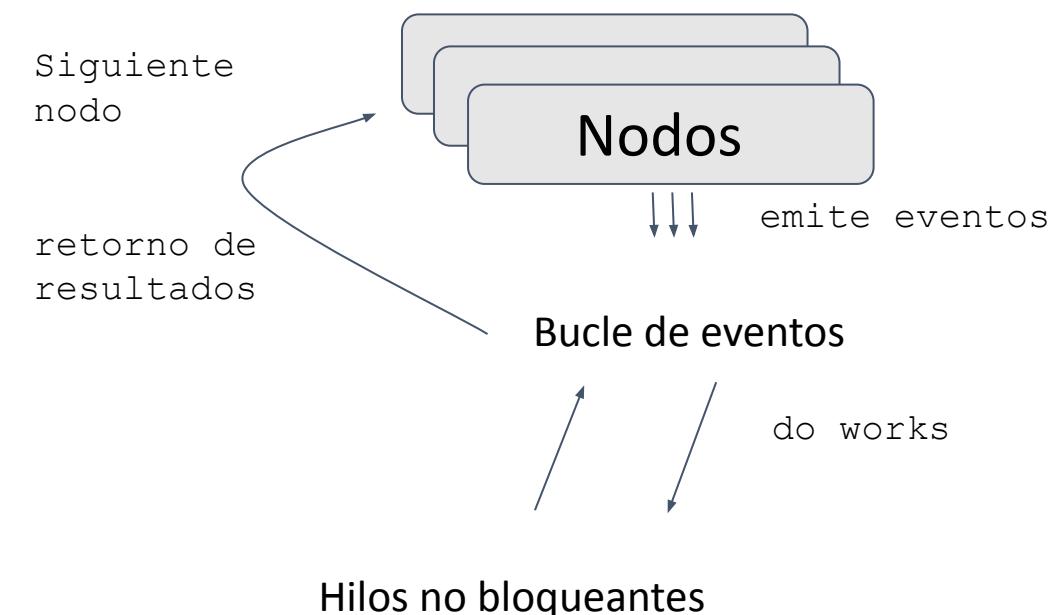
# ARQUITECTURA DE NODE RED

## Retorno de resultados desde los hilos de trabajo ("return results"):

Una vez que los hilos de trabajo completan una operación, devuelven los resultados al hilo principal (el bucle de eventos). Esto permite que el flujo de trabajo reciba la información procesada sin que el sistema principal se haya detenido, manteniendo la eficiencia y respondiendo rápidamente a otros eventos que pueden ocurrir en paralelo.

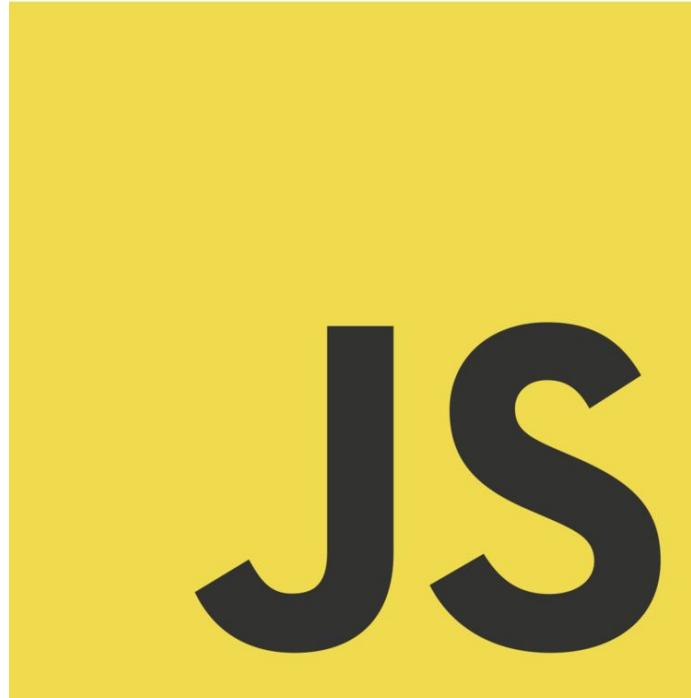
## Retorno de resultados al nodo principal ("return results"):

Finalmente, el resultado procesado regresa al nodo principal en el flujo de eventos, que luego lo pasa al siguiente nodo en la cadena. Esto completa el ciclo de procesamiento de ese nodo específico y permite que el flujo continúe avanzando al siguiente paso.



# JAVASCRIPT

JavaScript es un lenguaje de programación que se utiliza principalmente en el desarrollo web para crear interactividad en sitios web. A diferencia de lenguajes de programación como Python o Java, JavaScript se ejecuta en el navegador del usuario, lo que permite que las páginas web respondan a las acciones del usuario sin necesidad de recargar la página.

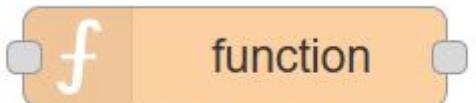
A large, bold, black "JS" logo centered on a yellow background. The letters are stylized with thick, rounded strokes.

JS

# USO DE JS EN EL NODO FUNCTION

El nodo **function** de Node-RED permite ejecutar código JavaScript personalizado para procesar mensajes que pasan a través del flujo. Este nodo es muy útil cuando se necesitan transformaciones de datos específicas o lógica avanzada que no se puede lograr fácilmente con otros nodos.

Aquí tienes un resumen de su funcionamiento:



1. **Entrada:** recibe un mensaje (`msg`) que contiene propiedades como `msg.payload` (donde suele estar el contenido principal) y otros datos.
2. **Código:** dentro del nodo, puedes escribir código JavaScript para modificar el mensaje, crear nuevos datos, realizar cálculos, entre otros.
3. **Salida:** el código puede devolver uno o más mensajes, modificados o nuevos, que serán enviados a los siguientes nodos en el flujo.

## TIPOS DE VARIABLES

```
let x = 12;
```

No admite redeclaración

```
var edad = 28
```

Admite redeclaración

```
const PI = 3.14159;
```

Variable de valor constante

## TIPOS DE DATOS

### Primitivos

String , Number, BigInt, Boolean, undefined, null, Symbol

### Compuestos

function, Object, arrays

# FUNCIONES

En JavaScript, un **objeto** es una colección de propiedades, donde cada propiedad es una clave (o nombre) asociada a un valor. Los objetos permiten agrupar datos y funciones relacionadas en una estructura organizada.

```
const obj = {};
```

Objeto vacío

```
const persona = {  
    nombre: "Juan",  
    edad: 25,  
    pais: "México"  
};
```

Objeto con  
propiedades

```
const persona = {  
    nombre: "Juan",  
    edad: 25,  
    saludar: function() {  
        return `Hola, me llamo ${this.nombre}`;  
    }  
};
```

Objeto con propiedades y métodos

## Acceso a propiedades

```
persona.nombre  
persona["edad"]  
persona.saludar()
```

# OBJETOS

En JavaScript, un **objeto** es una colección de propiedades, donde cada propiedad es una clave (o nombre) asociada a un valor. Los objetos permiten agrupar datos y funciones relacionadas en una estructura organizada.

```
const obj = {};
```

Objeto vacío

```
const persona = {  
    nombre: "Juan",  
    edad: 25,  
    pais: "México"  
};
```

Objeto con  
propiedades

```
const persona = {  
    nombre: "Juan",  
    edad: 25,  
    saludar: function() {  
        return `Hola, me llamo ${this.nombre}`;  
    }  
};
```

Objeto con propiedades y métodos

## Acceso a propiedades

```
persona.nombre  
persona["edad"]  
persona.saludar()
```

# CLASES

Al imprimir el objeto desde el nodo `Function`, debes convertir el objeto a una cadena JSON usando `JSON.stringify()`. Esto convierte el objeto en una representación de texto que puedes ver en el `debug` o en cualquier otro flujo de trabajo.

```
class Persona {  
    constructor(nombre, edad) {  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
}  
  
const persona1 = new Persona('Juan', 30);  
  
// Convertir el objeto a una cadena JSON legible  
msg.payload = JSON.stringify(persona1); // Esto lo convierte en un string legible  
return msg;
```

# CLASES

Se puede acceder a cualquier miembro con la notación .

```
class Persona {  
    constructor(nombre, edad) {  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
  
    saludar() {  
        return this.nombre;  
    }  
}  
  
// Crear una nueva instancia del objeto  
const juan = new Persona("Juan", 25);  
msg.payload = juan.saludar(); // Resultado: "Hola, me llamo Juan"  
return msg;
```

# HERENCIA

```
class Persona {
    constructor(nombre, edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    saludar() {
        node.log(`¡Hola, mi nombre es ${this.nombre} y tengo ${this.edad} años!`);
    }
}

class Estudiante extends Persona {
    constructor(nombre, edad, materia) {
        super(nombre, edad); // Llamamos al constructor de la clase padre
        this.materia = materia;
    }

    estudiar() {
        node.log(`${this.nombre} está estudiando ${this.materia}`);
    }
}

const estudiante1 = new Estudiante('Carlos', 20, 'Matemáticas');
estudiante1.saludar(); // Imprime: ¡Hola, mi nombre es Carlos y tengo 20 años!
estudiante1.estudiar(); // Imprime: Carlos está estudiando Matemáticas.
```

## Bucles while en function

```
// Bucle while ejemplo
let i = 0;
let resultados = [];

while (i < 5) {
    resultados.push(i * 2); // Ejemplo de operación
    i++;
}

msg.payload = resultados;
return msg;
```

## Bucles for en function

```
// Ejemplo con un array
let numeros = [1, 2, 3, 4, 5];
let resultados = [];

// Usando un bucle for para iterar sobre el array
for (let i = 0; i < numeros.length; i++) {
    resultados.push(numeros[i] * 2); // Ejemplo de operación dentro del bucle
}

// Asignar el resultado al payload para ver el resultado
msg.payload = resultados;
return msg;
```

# MANEJO DE TIEMPOS Y FECHAS

En programación, **fechas y tiempos** son fundamentales para registrar eventos, calcular diferencias entre sucesos y programar tareas automatizadas. En JavaScript (y en Node-RED, que usa JavaScript en sus nodos Function), el tiempo se maneja mediante el objeto `Date`, que proporciona la fecha y hora actuales, y permite realizar varias operaciones sobre ellas.

En Node-RED, manejar el tiempo es especialmente útil para:

- **Registrar marcas de tiempo** en los mensajes que pasan por un flujo.
- **Programar acciones** para ejecutarse en momentos específicos o después de ciertos intervalos.
- **Calcular duraciones o intervalos** entre eventos (por ejemplo, cuánto tiempo ha pasado desde que un sensor envió su última actualización).

# MANEJO DE TIEMPOS Y FECHAS

```
let fechaActual = new Date();
msg.payload = fechaActual;
return msg;
```

```
let fechaActual = new Date();
let formato = fechaActual.getFullYear() + "-" + (fechaActual.getMonth() + 1) + "-" + fechaActual.getDate();
let hora = fechaActual.getHours() + ":" + fechaActual.getMinutes() + ":" + fechaActual.getSeconds();
msg.payload = "Fecha: " + formato + " Hora: " + hora;
return msg;
```

# ESPERAS Y RETRASOS

En Node-RED, se pueden introducir retrasos y esperar de diversas maneras utilizando nodos específicos como **delay** y funciones dentro de un **nodo Function** para manejar tiempos de espera.

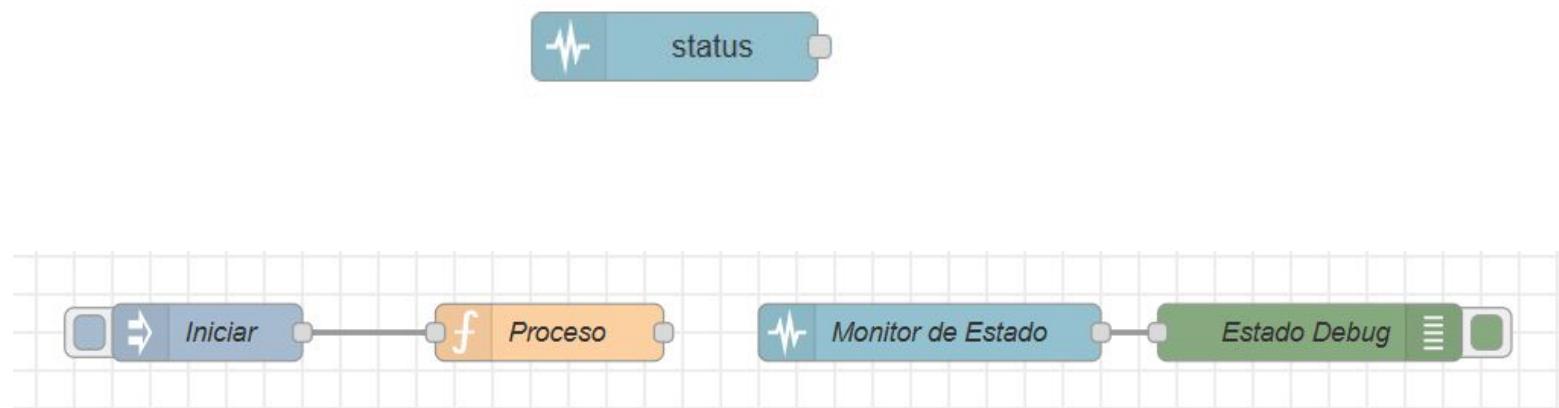
```
// Establecemos un retraso de 3 segundos (3000 ms)
setTimeout(() => {
    // Modificamos el mensaje después del retraso
    msg.payload = "Proceso completado";

    // Enviamos el mensaje al siguiente nodo
    node.send(msg);
}, 3000);

// Devolvemos null inmediatamente para no detener el flujo
return null;
```

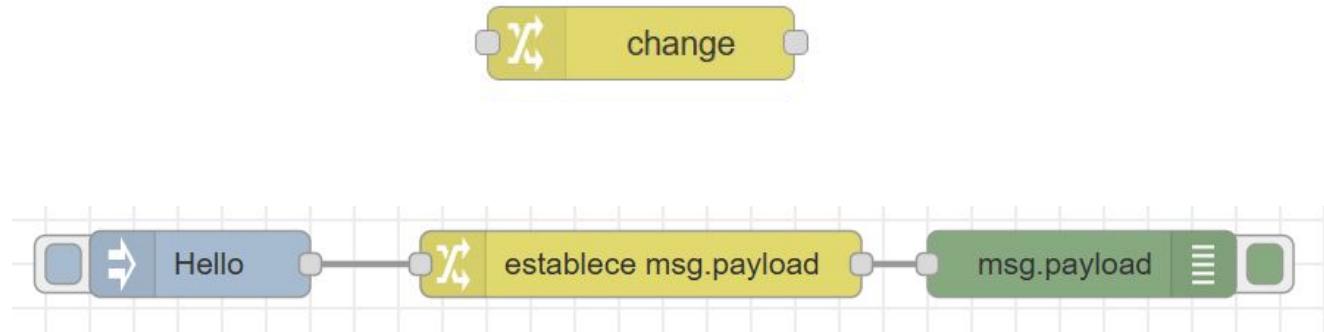
# NODO STATUS

El nodo **Status** en Node-RED permite monitorear el estado de otros nodos en tiempo real. Esto puede ser útil, por ejemplo, para saber si un nodo de conexión a una base de datos está conectado, si un nodo MQTT está en línea, o si hay algún problema en la comunicación. El nodo Status generará un mensaje cada vez que el estado del nodo que está monitoreando cambie, lo cual se puede usar para activar otros flujos o para registrar eventos en tiempo real.



## NODO CHANGE

El nodo `change` en Node-RED permite modificar mensajes sin necesidad de escribir JavaScript, lo que lo hace útil para cambios simples. Puedes usarse para establecer, cambiar, o eliminar propiedades de `msg` de una manera sencilla.



## NODO RANGE

El nodo `range` en Node-RED se utiliza para cambiar el rango de un valor numérico, permitiendo mapearlo de un rango de entrada a otro rango de salida. Esto es útil, por ejemplo, cuando necesitas adaptar datos de sensores a diferentes escalas.

