GRUPO BLR

INTRODUCCIÓN A LA PROGRAMACIÓN VISUAL **CON NODE-RED EN IOT**

ISBN: 978-9907-802-67-2

Dixys Leonardo Hernández Rojas

2026

INTRODUCCIÓN A LA PROGRAMACIÓN VISUAL CON NODE-RED EN IOT

Dixys Leonardo Hernández Rojas

AUTOR



UTMACH

Este libro ha sido debidamente examinado y valorado en la modalidad doble par ciego con fin de garantizar la calidad científica.

©Grupo Editorial BLR
Universidad Técnica de Machala
Riobamba – Ecuador
Correo: publicaciones@grupoblr.com
https://grupoblr.com/libros-investigo
REPOSITORIO



Hernández, D. (2026) Introducción a la programación visual con Node-RED en IoT. Grupo Editorial BLR.

© Dixys Leonardo Hernández Rojas

Introducción a la programación visual con Node-RED en IoT

ISBN: 978-9907-802-67-2

El copyright promueve la libertad de expresión, protege la diversidad de ideas y conocimiento, además apoya la libre expresión. Se prohíbe de manera rigurosa la producción o el almacenamiento de esta publicación, ya sea en su totalidad o en parte, está estrictamente prohibido por ley, incluyendo el diseño de la portada, así como su difusión a través de cualquiera de sus medios, ya sean electrónicos, mecánicos, ópticos, de grabación o incluso de fotocopia, sin permiso de los propietarios de los derechos de autor.

FILIACIÓN DEL AUTOR

Dixys Leonardo Hernández Rojas

Universidad Técnica de Machala

Correo Electrónico: dhernandez@utmachala.edu.ec

ORCID: https://orcid.org/0000-0002-2116-6531

INTRODUCCIÓN

Actualmente estamos viviendo el auge de la inteligencia artificial y la conectividad, donde ya es común disponer de los mismos objetos del ayer con capacidad de recibir comandos remotos e incluso reaccionar de forma autónoma según su entorno. El Internet de las Cosas (IoT, por sus siglas en inglés), es sin duda uno de los promotores y preecursores de toda esta transformación digital de nuestro mundo. Permitiendo interactuar con nuestro entorno, desde pequeña escala en nuestros hogares hasta llegar a contar con ciudades inteligentes y sectores industriales como la agricultura y la salud digital. En el ámbito del Io millones de dispositivos recolectan datos del mundo físico, escalados dentro de una arquitectura distribuida a través de protocolos y estándares, que permiten su procesamiento en diferentes niveles dentro de la arquitectura de referencia seleccionada, lo cual permite tomar decisionesoportunas, ágiles y seguras.

Todas estas bondades traen consigo enormes desafíos técnicos para poder hacerlos realidad, por ejemplo: la heterogeneidad de dispositivos (tipos y fabricantes), múltiples protocolos de comunicación y serialización de los datos y en varios casos lanecesidad de procesar dichos datos en tiempo real. Elevando así su curva de aprendizaje en términos de programación e integración de soluciones. Un aspecto interesante a considerar, es la familia de desarrolladores, aquí aparecen tanto los ingenieros, estudiantes como los “makers”, activos y entusiastas autodidactas que han elevado el nivel de acceso a estas tecnologías, permitiendo hacer prototipos rápidos sin dominar

completamente toda la teoría asociada. Aquí es donde la programación visual por flujos disminuye dicha curva de aprendizaje, con resultados rápidos y funcionales, listos para su uso tanto en la docencia como en la industria.

Node-RED fue desarrollado por IBM con IoT en mente y que fuera una herramienta útil para todo el universo de desarrolladores antes mencionado. Por tanto, esta herramienta es gratis, abierta y permite escalar los proyectos incluso de forma profesional. Node-RED, como su nombre lo indica, permite crear flujos arrastrando y uniendo nodos funcionales de forma gráfica, lo cual da una sensación de no estar escribiendo códigos de algún lenguaje de promación tradicional. Pero, Node-RED está desarrollado encima del ecosistema Node.js, siendo un servidor web en si mismo, requiriendo solamente de un navegador para comenzar a desarrollar. Cada uno de estos bloques, llamados “nodos”, representa acciones como capturar datos, transformarlos, decidir condiciones o enviarlos a otros sistemas o dispositivos.

La gran ventaja de este enfoque visual es que reduce drásticamente la barrera de entrada para desarrollar soluciones IoT. Profesionales que no provienen de la ingeniería de software, como técnicos electrónicos, ingenieros industriales, profesores o makers, pueden construir sistemas funcionales con mínima o ninguna experiencia en programación tradicional. Además, esta forma de construir flujos facilita la lectura colaborativa del código, su mantenimiento, su documentación y su comprensión pedagógica, lo cual es vital para procesos de formación académica y transferencia tecnológica.

Node-RED se posiciona, así como una plataforma de orquestación de procesos IoT, donde convergen sensores, protocolos de comunicación como MQTT y HTTP, sistemas de almacenamiento locales o en la nube, y mecanismos de visualización y automatización. La genialidad de IBM al regalarnos una plataforma web super liviana de desarrollo, permite su ejecución a todo nivel, desde un contenedor corriendo en una simple PC, un servidor robusto o simplemente en una Raspberry Pi.

Node-RED dentro del ámbito del IoT permite resolver algunos de los siguientes problemas:

- La integración de dispositivos y plataformas heterogéneas usadas en IoT, ya sea integrar dentro de una misma plataforma IoT a varios dispositivos de diferentes fabricantes o estos mismo dispositivos integrados a plataformas de diferentes empresas, incluso de forma simultánea.

- Automatizaciones robustas a través de reglas y lógicas complejas. Dentro de Node-RED nodos que veremos más adelante como Switch, Change, Join y Function facilitan la integración.

- Necesidad de construir prototipos y POC (pruebas de concepto) funcionales y económicos, que permitan luego escalar a soluciones completas de forma rápida o inmediata.

- Intercomunicación entre capas de todos los elementos presentes en una arquitectura IoT, desde el diseño a su implementación, dentro de un mismo entorno de desarrollo. Por ejemplo: sensores

de la capa de precepción, gateway en la capa de red y bases de datos o dashboards en la capa de aplicaciones.

Disponer de un solo entorno didáctico, de herramientas para la enseñanza de varias disciplinas como la electrónica, microprocesadores y programación de alto nivel, de fácil uso y rápida adopción de todos los involucrados en un proceso educativo tanto de pregrado como postgrado. El propósito de este libro es proporcionar una guía integral, didáctica, que permita un aprendizaje profundo y progresivo del uso de Node-RED en el desarrollo y despliegue de sistemas IoT. Sus capítulos, explican la teoría a través de ejemplos prácticos, flujos (programas) de uso inmediato en situaciones reales, integración con el hardware en todas las capas de una arquitectura IoT, visualización de datos a través de dashboard interactivos con salida profesional. Además, con un enfoque pedagógico— basado en con pequeñas prácticas, añadiendo tips técnicos, validaciones, esquemas y preguntas de revisión en cada tema tratado, facilitando al lector la consolidación de conocimientos, y dotándolo de competencias reales.

Si bien este libro ha sido concebido como un instrumento pedagógico para docentes y estudiantes universitarios vinculados al área de la electrónica, sistemas embebidos, telecomunicaciones, automatización, informática industrial o ingeniería en general. Su enfoque visual, lenguaje claro y estructura modular lo convierten en una excelente herramienta para la comunidad “maker” que deseen empezar a crear soluciones en el mundo del IoT sin chocar con la complejidad de la programación tradicional.

Hoy en día, donde convivimos con objetos conectados, donde la eficiencia energética, la automatización y la toma de decisiones en tiempo real se convierten en prioridad, aprender Node-RED se convierte en una necesidad estratégica. La capacidad de transformar datos en acciones, y flujos de información en soluciones automatizadas, será una habilidad clave en el futuro tecnológico inmediato. Este libro te acompañará en ese camino.

Objetivo general

Guiar a través de este libro, de forma progresiva y aplicadamente el uso de Node-RED en entornos de Internet de las Cosas, combinando fundamentos teóricos, flujos de automatización y escenarios reales, con un enfoque pedagógico orientado a docentes, estudiantes y profesionales del área tecnológica.

Estructura del libro

El contenido del libro está dividido en **cuatro capítulos**, organizados de manera progresiva, donde cada uno consolida las bases del anterior y prepara al lector para abordar soluciones cada vez más complejas y profesionales:

📗 Capítulo 1: Introducción a Node-RED

Presenta los fundamentos de Node-RED, su arquitectura sobre Node.js, las diferentes formas de instalación en diversos sistemas operativos (Windows, Linux, Raspberry Pi, Android), la exploración del entorno

visual y la incorporación de nuevos nodos y configuraciones de seguridad básica.

📗 Capítulo 2: Programación lógica en Node-RED

Desarrolla las estructuras de control típicas en cualquier lenguaje de progaramción tradicional como las condicionales, bucles o lazos, temporizadores y transformaciones de datos. Adicional a la enseñanza de algunosnodos básicos comoinject, switch, change, template, function, se introducen las buenas prácticas necesarias para un diseño modular mediante flujos.

📗 Capítulo 3: Flujos avanzados

Luego de aprender lo básico y necesario para desarrollar un prototipo funcional real con los capítulos anteriores, pasamos a desarrollar flujos complejos y una estructura de desarrollo cooperativa profesional, por ejemploel manejo de variables de contexto, estructuras anidadas, proyectos versionables con Git, integración de librerías externas y creación de nodos personalizados. Se introduce también s la seguridad en el diseño y en la lógica de JavaScript.

📗 Capítulo 4: IoT en acción

Aquí veremos el uso deNode-RED en contextos reales de IoT: para ello se integran protocolos típicos de IoTcomo MQTT, HTTP, UART, GPIO, I²C y SPI; control de hardware (Raspberry Pi); gestión de bases de datos locales y en la nube; diseño y elaboración de dashboards interactivos; y

conexión con plataformas IoT domóticas como Home Assistant y ESPHome.

Estructura pedagógica

A lo largo de los capítulos, se incorporan elementos (teórico – prácticos) que favorecen el aprendizaje autónomo y la comprensión progresiva:

- **Apertura de cada capítulo** con objetivos claros, contenidos a cubrir y lo que el lector será capaz de hacer al finalizarlo.
- **Desarrollo progresivo de secciones**, con ejemplos visuales, capturas de pantalla, diagramas explicativos y estructuras bien definidas.
- **Recursos pedagógicos internos**, como:
  - ⚠️ Precauciones técnicas
  - ☑️ Buenas prácticas
  - 💡 Tips técnicos
  - 🛠️ Instrucciones paso a paso
  - 🧪 Mini prácticas
- **Preguntas de revisión** al final de cada capítulo, con íconos que estimulan la autoevaluación (❓ ✍️ ☑️) y una **clave de respuestas** estructurada.

- **Bibliografía técnica actualizada** en cada capítulo para fomentar la profundización y el aprendizaje autónomo continuo.

Estas características convierten al libro en un recurso ideal tanto para el uso en aula, como en programas de formación técnica, talleres profesionales o autoformación guiada.

Preguntas de revisión y autoevaluación: Al final de cada capítulo, se incluyen preguntas de opción múltiple, verdadero/falso y respuesta corta para reforzar el aprendizaje.

ÍNDICE

INTRODUCCIÓN ..... **i**

ÍNDICE ..... **ix**

ÍNDICE DE TABLAS ..... **xv**

ÍNDICE DE FIGURAS ..... **xvi**

CAPÍTULO I **23**

1 INTRODUCCIÓN A NODE-RED **23**

1.1 ¿Qué es Node-RED y por qué se convirtió en la herramienta mas usada en IoT? 25

1.2 Línea de Tiempo de las Versiones Principales de Node-RED 27

1.2.1 Versión 1.x 27

1.3 ¿Por qué Node-RED es útil en IoT? 30

1.4 Instalando Node-RED 31

1.4.1 Instalación en plataformas de hardware más populares: PC (Windows), Raspberry Pi y Docker 33

1.4.2 Instalación de Node.js en Windows 38

1.4.3 Recomendaciones para diferentes sistemas operativos 49

1.5 Crear diferentes instancias de Node-RED por sistema operativo 52

1.6 Conociendo el Front-end de Node-RED 57

Editor visual: interfaz, áreas y menús 60

1.6.1 Flujos y nodos: estructura y conexión 64
1.7 Añadir e instalar nuevos nodos 65
1.7.1 Instalación desde la interfaz gráfica (Palette Manager) 66
1.7.2 Instalación de nuevos nodos mediante línea de comandos (npm) sobre Node.js 67
1.8 Configuración general y seguridad básicos 69
1.8.1 El archivo settings.js: núcleo de configuración de Node-RED 70
1.8.2 Control de acceso con usuarios y contraseñas 79
1.8.3 Seguridad en despliegues remotos 81
1.9 Copias de seguridad periódicas del entorno Node-RED 86
CAPÍTULO II 96
2 PROGRAMACIÓN LÓGICA EN NODE-RED 96
2.1 Nodos básicos (Core nodes) 98
Nodos: inject, debug, change, function, template 98
2.2 Hola mundo 102
2.3 Estructuras de control 106
Condicionales y decisiones 106
Loops y repeticiones 108
2.3.1 Timeout, delays e intervals 142
2.4 Intervalos en Node-RED (Repeticiones temporizadas) 143
2.5 Manipulación de datos 148

2.5.1 Escalado de datos (range, function) 148
2.5.2 Fusión de datos (merging) 151
2.5.3 Separación de datos (splitting) 153
2.5.4 Reestructuración o transformación de mensajes (change, function) 154
2.5.5 Conversión de tipos y formatos 156
2.6 Buenas prácticas de diseño de flujos 163
2.6.1 Usa comentarios y etiquetas descriptivas 163
2.6.2 Agrupa por bloques funcionales 164
2.6.3 Usa subflujos para modularizar 165
2.6.4 Limita el uso de nodos function innecesarios 166
2.6.5 Usa colores y nombres personalizados 166
2.6.6 Controla la cantidad de conexiones (wires) 166
2.6.7 Despliega frecuentemente con control 167
CAPÍTULO III 171
3 FLUJOS AVANZADOS 171
3.1 Gestión de datos y contexto 173
3.1.1 Variables locales, de flujo y globales 173
3.1.2 Mensajes y estructuras anidadas 179
3.2 Trabajo con proyectos en Node-RED 184
3.2.1 Activación del sistema de proyectos 184
3.2.2 Crear un nuevo proyecto 186

3.2.3 Versionamiento y backups 186
3.3 Programación con funciones 192
3.3.1 Uso del nodo function en JavaScript 192
3.3.2 Validación y manejo de errores 195
3.3.3 Importación de librerías externas 197
3.4 Nodos personalizados 201
3.4.1 Estructura básica de un nodo personalizado 202
3.4.2 Crear un nodo personalizado paso a paso 203
3.4.3 Publicar el nodo en NPM 208
3.4.4 Registro en el flujo de trabajo 209
3.5 Seguridad avanzada 214
3.5.1 Gestión de usuarios y contraseñas 214
3.5.2 Protección de nodos HTTP (http in, http response) 216
3.5.3 Permisos en entornos colaborativos 216
3.5.4 Seguridad en nodos function y uso de require() 217
3.5.5 Aislamiento por red 218
3.5.6 Supervisión y logs 218
Capítulo IV 223
4 IoT en Acción 223
4.1 Protocolos de comunicación 225
4.1.1 MQTT: suscripción y publicación 225
4.1.2 HTTP: clientes y servidores REST 226

4.1.3 UART, I2C y SPI en Raspberry Pi 227
4.2 Acceso al hardware desde Node-RED 234
4.2.1 Control de GPIO en Raspberry Pi 234
4.2.2 Lectura de entradas digitales 236
4.2.3 Escritura de salidas digitales 237
4.2.4 Sensores y actuadores mediante I²C y SPI 239
4.3 Almacenamiento y persistencia 245
4.3.1 Almacenamiento en bases de datos locales 248
4.3.2 Bases de datos en la nube: Firebase, MongoDB Atlas 249
4.4 Integraciones domóticas 251
4.4.1 Integración con Home Assistant 252
4.4.2 Integración con ESPHome y dispositivos IoT 254
4.5 Visualización e interfaces de usuario 256
4.5.1 El nodo node-red-dashboard 257
4.5.2 Personalización con CSS 260
4.6 Manejo de formatos de datos 262
4.6.1 JSON (JavaScript Object Notation) 262
4.6.2 CSV (Comma-Separated Values) 263
4.6.3 Texto plano (string) 265
4.6.4 YAML (Yet Another Markup Language) 266
4.6.5 HTML (HyperText Markup Language) 266
4.6.6 XML (Extensible Markup Language) 267

4.6.7 Estructuras mixtas 268

4.7 Validación de tipo de dato en Node-RED 274

4.7.1 Validación de null e undefined en Node-RED 278

4.8 Validaciones con nodos visuales 283

BIBLIOGRAFÍA 290

GLOSARIO 292

ÍNDICE DE TABLAS

Tabla 1. Validación de valores en Node-RED (function). 281

ÍNDICE DE FIGURAS

Figura 1. Aplicación web Node-red. 26

Figura 2. Node-red presente en todas las capas de una arquitectura de IoT. 31

Figura 3. Página inicial oficial de Node-red. 33

Figura 4. Muestra un resumen visual de los pasos a seguir para instalar node.js en Windows. 34

Figura 5. Pasos para instalar Node.js en Windows. 37

Figura 6. Consideraciones básicas según el sistema operativo. 52

Figura 7. Consola de la instancia de node-red corriendo sobre Windows. 58

Figura 8. Logs en la consola de Node-red. 60

Figura 9. Editor de Node-red. 61

Figura 10. Estructura básica de un flujo en node-red. 65

Figura 11. Instalación de nuevos nodos a través del Manage palette. 67

Figura 12. Ubicación del archivo de configuración sttings.js en Windows. 70

Figura 13. Configuración del nodo Inject. 103

Figura 14. Modo completo del nodo Debug. 104

Figura 15. Ejemplo de una salida en la zona Debug. Elaboración.propia. 105

Figura 16. Ejemplo: Hola mundo con debug completo. 105

Figura 17. Ejemplo de uso del nodo Switch. 107

Figura 18. Configuración del nodo Switch. 107

Figura 19. Uso del nodo Change para definir un valor específico del payload. 108

Figura 20. Lazo repetitivo usando el nodo Inject. 110

Figura 21. Configuracion del nodo Inject para conseguir lazos repetitivos. 111

Figura 22. Lazo creado en el mismo flujo a través del nodo Delay. 113

Figura 23. Configuracion del nodo Inject incializando una variable. 113

Figura 24. Configuración del nodo function para crear bucles. . 114

Figura 25. Configuración del nodo delay. 116

Figura 26. Gestión de arreglos usando la combinación de nodos Split - Join. 117

Figura 27. Arreglo incial en forma de Json. 117

Figura 28. Configuración del nodo Split. 118

Figura 29. Configuración del nodo Function para gestionar cada elemento del arreglo. 119

Figura 30. Configuración automática del nodo Join. 119

Figura 31. Flujo del ejemplo de Counter-loop. Puede encontrar este flujo en los ejemplos que se instalaron junto a la librería node-red-contrib-loop-processing. 120

Figura 32. Resultado de cada iteración del Counter-loop. 122

Figura 33. Configuración del nodo Inject para inicializa rel loop. 123

Figura 34. Configuración del nodo Counter-loop con 5 iteraciones. 124

Figura 35. Configuración del Change del flujo Counter-loop, el procesamiento es un simple Join. 125

Figura 36. Configuración del JsonNata del Join. 126

Figura 37. Flujo de ejemplo de suso del nodo: array-loop. 127

Figura 38. Array inicializado desde el nodo Inject. 128

Figura 39. Configuración del nodo Change con la función Split dentro de un JSONata 129

Figura 40. Definición del array dentro del loop. 130

Figura 41. Nodo function dentro del array-loop permite hace alguna acción dentro de cada iteración. 130

Figura 42. Resultados de cada iteración del array-loop. 131

Figura 43. Flujo de ejemplo del Nodo: While-loop. 132

Figura 44. Se inicializa el lazo con el valor 9, para probar desde 9 hasta el 0 cuales números serán divisibles por 5. 133

Figura 45. Nodo centrol While-loop, permite hacer iteraciones mientras se cumple una condición. 134

Figura 46. Configuración del node Change para decrementar un valor numérico. 135

Figura 47. Resultado de cada iteración y salida del While, comenzando con 5. 136

Figura 48. Subflujo Iterate que simula un lazo iterativo. 139

Figura 49. Flujo de prueba de la solución "Iterate" basada en un subflujo. 140

Figura 50. Resultados de cada iteración y resultado final del subflujo Iterate. 140

Figura 51. Opciones de repetición disponibles en el nodo Inject. 145

Figura 52. Opción disponible dentro del nodo inject para ejecutar dicho nodo a un tiempo específico. 147

Figura 53. Ejemplo de uso del nodo range. 149

Figura 54. Configuración y Salida del nodo range. 150

Figura 55. Ejemplo de uso del nodo Join para crear un Json con los valores de 2 sensores. 152

Figura 56. Ejemplo del uso del nodo Split y su resultado. 154

Figura 57. Configuraciones del nodo Change. 156

Figura 58. Ejemplo de uso del nodo change con JSONata para hacer una conversión de texto a número. 157

Figura 59. Opciones de conversión bidireccional del nodo JSON. 160

Figura 60. Ejemplo de uso del nodo Change para obtener de forma legible un timestamp. 161

Figura 61. Ejemplo del uso del nodo Comments para hacer más descriptivos los flujos. 164

Figura 62. Ejemplo de uso de bloques y colores para hacer más descriptivo los flujos. 165

Figura 63. Opciones para hacer un deploy (instanciar) 167

Figura 64. Ejemplo de uso de persistencia en la memoria RAM a través de las variables flow, context y global. El valor de las variables var1, var2 y var3, aparecen en el extremo derecho de la figura. 177

Figura 65. Ejemplo de persistencia de las variables var1, var 2 y var3, luego de un incremento por medio de la ejecución del flujo. 178

Figura 66. Uso del nodo change para acceder y modificar estructuras anidada en un Json. 182

Figura 67. Habilitación del uso de proyectos dentro de Node-red en el archivo settings.js. 185

Figura 68. Pasos a seguir para crear un nuevo proyecto desde la interfaz gráfica de Node-red. 189

Figura 69. Primeros pasos para crear un nodo propio, modificando el archivo package.json. 204

Figura 70. Creacion de archivos.js y .html e instalacion del nodo propio dentro del ecosistema de node-red. 207

Figura 71. Nodo propio creado: mi-nodo-libro, ya aparece en node-red y es usado en un flujo sencillo, mostrando resultados en el debug. 208

Figura 72. Nodos UART instalados desde la interfaz gráfica de Node-red. 229

Figura 73. Nodos I2C disponibles a ser instalados, uno especifico (BME280) y otro genérico. 231

Figura 74. Nodos SPI disponibles a ser instalados desde la interfaz gráfica de Node-red. 233

Figura 75. Nodos para gestionar GPIO como entradas digitales en una Rpi. Elaboracinón propia. 237

Figura 76. Nodos para gestionar GPIO como salidas digitales en una Rpi. 238

Figura 77. Gestión de datos a traves de archivos .csv. 248

Figura 78. Nodos disponibles para las bases de datos: firebase, mongodb,dynamodb y cosmosdb. 250

Figura 79. Integración de Node-red dentro de la plataforma domótica Home Assistant. Tomado de (Home Assistant Community Add-on: Node-RED, 2018). 252

Figura 80. Opciones de instalación del paquete de Dashboard en Node-red. 258

Figura 81. Front-end crado con el dashboard2.0 de Flowfuse. A la derecha se muestrasn los nodos disponibles. Tomado de (@Flowfuse/Node-red-dashboard, 2026). 259

CAPÍTULO I

1 INTRODUCCIÓN A NODE-RED

🎯 Objetivos de aprendizaje:

- Comprender la arquitectura de Node-RED y su relación con Node.js.
- Instalar Node-RED en diferentes sistemas operativos (Windows, Linux, Raspberry Pi, Android).
- Explorar el editor visual y su interfaz gráfica.
- Añadir e instalar nodos personalizados.
- Configuración de respaldos y elementos básicos de seguridad

🛠️ Competencias que serán obtenidas:

- Instala y configura Node-RED según plataforma seleccionada.
- Identifica los componentes necesarios para el desarrollo gráfico y visual.
- Uso de nodos (elementos gráficos) para construir un flujo funcional.
- Uso de buenas prácticas de programación, respaldom configuración y seguridad en un entorno visual.

❓ Preguntas decomprobación:

- ¿En un entorno de desarrollo de soluciones IoT, qué papel podría cumplir Node-RED?
- ¿Cuál es la estructura interna de Node-RED?
- Indique los pasos a seguir para instalar y configurar Node-RED en su plataforma de trabajo.
- ¿Cómo se respalda tu trabajo en Node-RED?

En este capítulo queremos que el lector comprenda qué es Node-RED y por qué se ha convertido en la herramienta mas usada por toda la comunidad que desarrolla soluciones para IoT. El lector comenzará su propio camino de desarrollo con una nueva visión de programación, en este caso programación visual o gráfica, pasando de simples prototipos a soluciones profesionales dentro del mismo entorno. Asimismo, se presentarán los entornos recomendados para su instalación, las interfaces principales del editor, la manera de incorporar nuevos nodos y los aspectos básicos de configuración y seguridad necesarios para su correcto funcionamiento.

Al finalizar este capítulo, el lector estará en capacidad de instalar y ejecutar Node-RED en diferentes plataformas, explorar su entorno gráfico, comprender la estructura básica de un flujo, añadir nuevos nodos desde el administrador de paletas, y aplicar configuraciones iniciales que garanticen un entorno de trabajo seguro y funcional. Una vez adquiridos los elementos básicos inciales, el lector podrá incluir elementos avanzados en los siguientes capítulos.

1.1 ¿Qué es Node-RED y por qué se convirtió en la herramienta más usada en IoT?

Node-RED es un servidor web y a su vez una plataforma de desarrollo. A diferencia de los entornos tradicionales está basada en flujos creados mediante la programación visual (gráfica). Como habíamos mencionado en la introducción fue creado originalmente por IBM, pero en la actualidad su crecimiento exponencial se debe en gran medida a los aportes de toda la comuniad, ya que fue concebida open-source. Gracias a una interfaz gráfica intuitiva, sencilla y fácil de usar y su arquitectura flexible, ver **¡Error! No se encuentra el origen de la referencia.**, basada en JavaScript (Node.js), Node-RED ha permitido una amplia adopción no solo en el ámbito del Internet de las Cosas (IoT), sino también en entornos industriales, académicos, incluso a nivel doméstico.

Los flujos (programas) están formado por la unión de varios nodos funcionales, los cuales tienen entradas y salidas, permitiendo ingesta, procesamiento y salida personalizada de los datos. Los usuarios (todos se conveirten en programadores) pueden diseñar, probar y desplegar soluciones completas integrando sensores, actuadores, de forma local o en la nube y dashboards atractivos e interactivos. Además, al estar desarrollado sobre Node.js, permite crecer su funcionalidad con funciones propias inclusive, personalizadas y escritas en JavaScript directamente, esto amplia aún mas la versatilidad y flexibilidad del desarrollo, combinando programas escritos textuales y programas (flujos) gráficos.

Podemos indicar que un entorno IoT, donde confluyen múltiples tecnologías, protocolos y dispositivos heterogéneos, Hernández Rojas, D. L. (2025), Node-RED actúa como un orquestador simplificando la integración de todos sus componentes. Ya sea para controlar una red de sensores en un invernadero, automatizar dispositivos en una casa inteligente o monitorear variables industriales en tiempo real, Node-RED se posiciona como una herramienta versátil y potente, ideal para quienes buscan crear soluciones eficientes sin tener que desarrollar desde cero infraestructuras complejas.

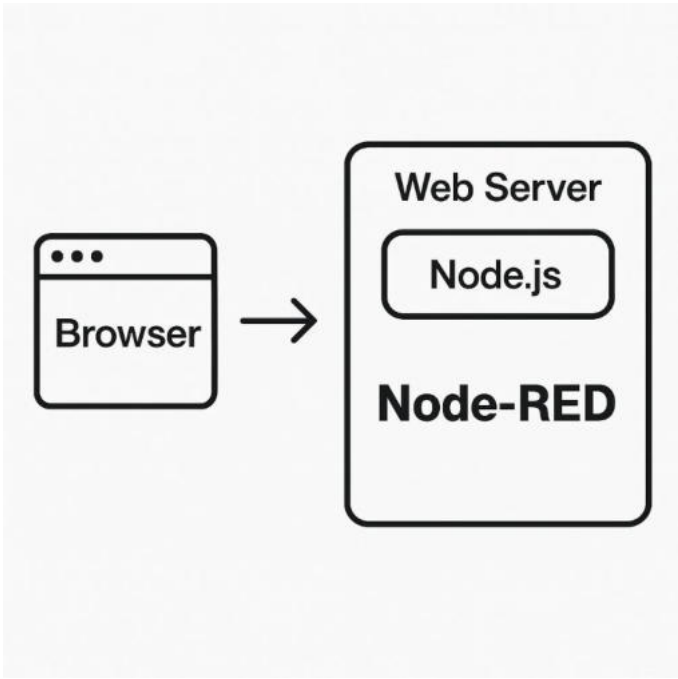


Figura 1. Aplicación web Node-red.

Node-red fue creado por **Nick O'Leary** con la intención de crear una plataforma fácil de usar e intuitiva basada en una programación visual para el Internet de las Cosas (IoT), desde IBM, espcíficamente en el

grupo de Tecnología Emergente, junto a **Dave Conway-Jones**, ambos crearon una herramienta que permitiera construir flujos de integración entre dispositivos, APIs y servicios prácticamente sin saber programar o sin una codificación compleja.

NOde-RED fue lanzado en **2013**, desde este entonces, Nick lidera el camino de Node-RED, el cuál comenzó desde un proyecto interno de laboratorio hasta convertirse en la herramienta open-source por excelencia y adoptada tanto en la docencia pregrado, postgrado como en la industria. Nick es un elemento activo dentro del movimiento *low-code* y ha dictado varias conferencias internacionales, promoviendo a Node-RED como la pieza clave que une la electrónica y servicios digitales desde lo simple a lo complejo.

Actualmente, sigue activo dentro de la comunidad desde GitHub, donde se encarga de mantener el repositorio incial y oficial y se mantiene vigilante de la evolución del proyecto, consolidando de esta forma a Node-REDdentro del IoT moderno. En el próximo apartado, se resumen las principales versiones que han sido liberadas desde su lanzamiento hasta la fecha que fue escrito este libro, obviamente seguirá evolucionando como nos tienen acostumbrado.

1.2 Línea de Tiempo de las Versiones Principales de Node-RED

1.2.1 Versión 1.x

📅 *Fecha de lanzamiento:* 30 de septiembre de 2019

🔧 *Principales características:*

- Introducción del uso de mensajes asincrónicos.
- Interfaz de usuario mejorada, se añade un editor JSON visual y la barra lateral de contexto.
- Paleta de nodos reestructurada y mas opciones para la importación/exportación de flujos.
- Se da soporte a la creación de subflujos con propiedades que pueden ser configuradas.
- Versión estable.

1.2.1.1 🟡 Versión 2.x

🗓️ *Fecha de lanzamiento:* 22 de julio de 2021

🔧 *Principales características:*

- Necesidad de uso de Node.js 12 o superior a partir de esta versión.
- Mas estabilidad y aumenta el rendimiento.
- Editor con nuevas funciones y nodos básicos actualizados.
- Mas soporte personalizado para el editor.
- Mejoras adicionales al subflujos y a los nodos de configuración.

1.2.1.2 Versión 3.x

Fecha de lanzamiento: 14 de julio de 2022

Principales características:

- Se añade Multiplayer Mode (colaboración en tiempo real) (.
- Se incluye autocompletado en la interfaz de usuario y se mejora la vista de diferencias.
- Se añaden nuevos nodos y se actualizan los existentes.
- Se mejora la gestión de proyectos y flujos y subflujos.

1.2.1.3 Versión 4.x

Fecha de lanzamiento: 30 de abril de 2024

Principales características:

- Requiere Node.js 18 o superior.
- Mejoras en el rendimiento del despliegue de flujos grandes.
- Nuevas opciones de formato de timestamp en el nodo Inject.
- Posibilidad de personalizar nodos de configuración en subflujos.
- Mejoras en la interfaz de usuario y experiencia del editor.

1.2.1.4 🔴 Versión 5.x

🗓️ *Principales características:* Enero 2026, actualizada el 30 de julio de 2026

🔧 *Principales características:*

- Se requiere Node.js 22 o superior, a partir de esta versión.
- Interfaz node-RED UX moderna.
- Nodos muestran una nueva apariencia
- Aumento considerable de nuevas funcionalidades
- Actualización a la infraestructura de proyectos.

1.3 ¿Por qué Node-RED es útil en IoT?

Una de las razones fundamentales por las que **Node-RED se ha consolidado como una herramienta estratégica en el desarrollo de soluciones IoT** es su extraordinaria versatilidad para ejecutarse en todas las capas de una arquitectura típica del Internet de las Cosas, como se puede apreciar en la **¡Error! No se encuentra el origen de la referencia.**. En la **capa de dispositivos**, puede correr de forma eficiente sobre plataformas embebidas como una Raspberry Pi, permitiendo la adquisición directa de datos desde sensores y el control de actuadores locales. En cambio en la capa **de comunicaciones**, Node-RED puede correr sobre plataformas diversas desde una Rpi como servidores dedicados para actuar como intermediario entre los dispositivos de

borde (de la capa anterior) y sistemas superiores, a través de protocolos como MQTT, HTTP o serial (UART/I2C). Por último, en la **capa de aplicación**, Node-RED puede desplegarse tanto en servidores locales, escritorios, como en servicios o servidores en la nube e incluso dispositivos móviles, facilitando la orquestación de todos los datos, ya sea integrando bases de datos, como garantizando el monitoreo de variables del entorno IoT mediante dashboards, además de permitir la interoperabilidad con otras plataformas externas. Esta **facilidad de trabajaren cualquier capa** lo convierte en la solución ideal para entornos educativos, industriales y de investigación.

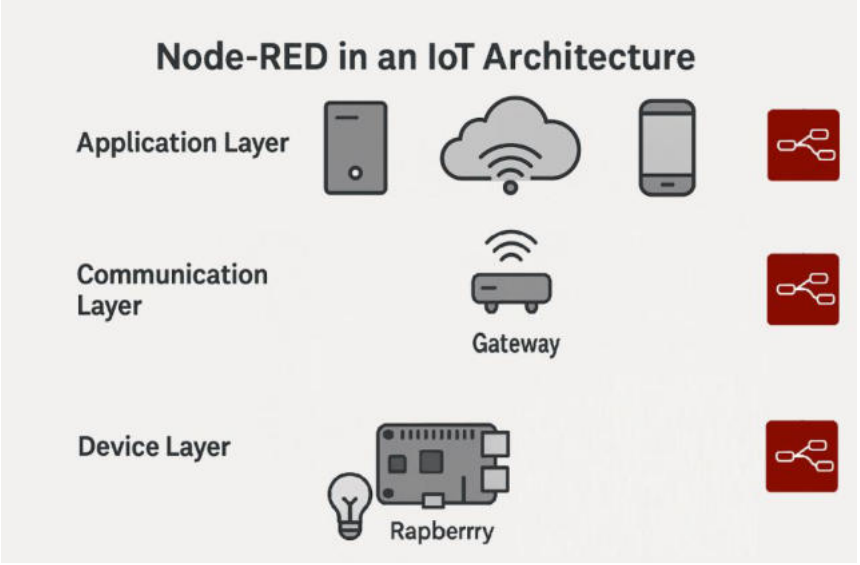


Figura 2. Node-red presente en todas las capas de una arquitectura de IoT.

1.4 Instalando Node-RED

Como hemos indicado a lo largo de este libro, una de las ventajas de adoptar a Node-RED como herramienta base, es la posibilidad de uso sobre múltiples pltaformas de hardware. Node-RED puede correr sobre

sistemas operativos de escritorio como en plataformas con menos recursos como la Raspberry Pi (Raspberry Pi. 2026), obviamente en plataformas con más recursos como servidores locales o plataformas en la nube, también es posible En este epígrafe el lector podrá saltar directamente a la plataforma de su elección y revisar los pasos a seguir. Como Node.RED corre encima de Node-js, tambienj se incluye su respectiva guía de instalación. Esta flexibilidad convierte a Node-RED en una herramienta multiplataforma y escalable, facilitando su adopción en diversos contextos académicos, domésticos e industriales.

Si observamos la página oficial de Node-red, mostrada en la **¡Error! No se encuentra el origen de la referencia.**, tenemos a nuestra disposición la documentación requerida para conseguir instalar nuestro espacio de desarrollo prácticamente en cualquier plataforma de hardware, incluso en la nube. No obstante, en este epígrafe, hacemos la introducción de las plataformas más populares disponibles para nuestra audiencia, sobre todo los principiantes.

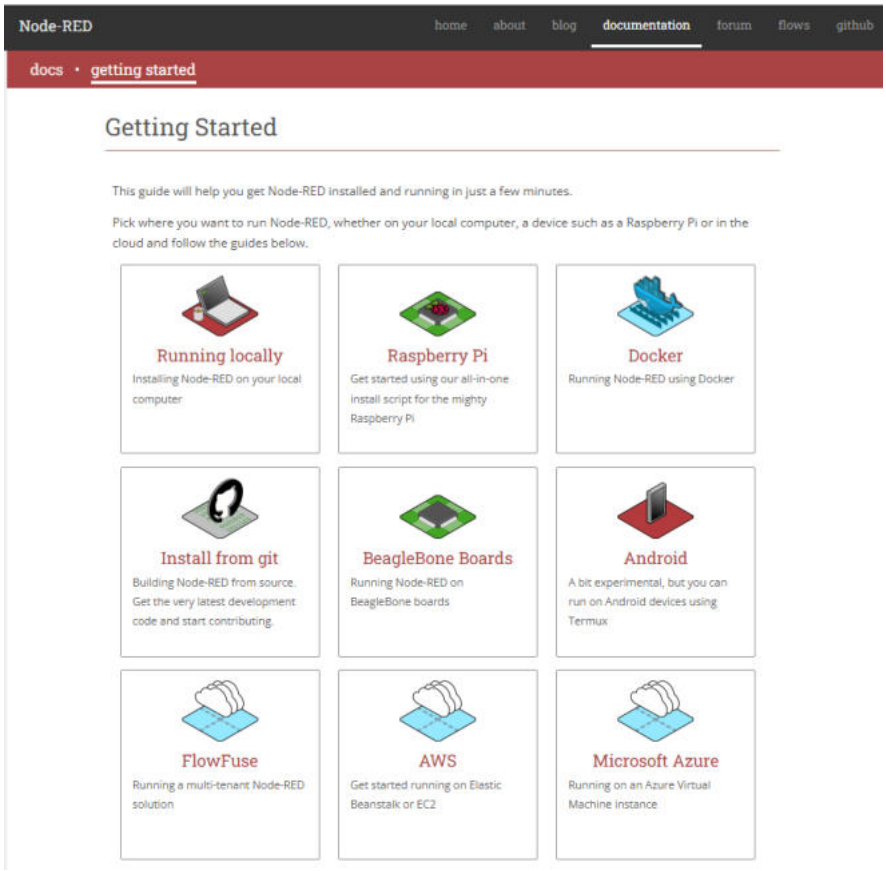


Figura 3. Página inicial oficial de Node-red.

Tomado de (Getting Started: Node-RED, 2026).

1.4.1 Instalación en plataformas de hardware más populares: PC (Windows), Raspberry Pi y Docker

💻 A. Instalación en PC con Windows (usando Node.js)

Requisitos previos:

- Conexión a Internet.

- Tener privilegios de administrador en el sistema.

Pasos detallados:

1. **Instalar Node.js (requisito fundamental):**
   Node-RED se ejecuta sobre Node.js (Node.js — Run JavaScript everywhere, 2026), por lo que este debe instalarse primero.

   - Ingresar a la página oficial: https://nodejs.org
   - Descargar la versión LTS recomendada (por ejemplo, Node.js 18.x.x LTS).
   - Ejecutar el instalador .msi descargado. Asegurarse de seleccionar la opción que agrega node y npm al PATH del sistema.
   - Al finalizar, abrir la terminal (buscar "Símbolo del sistema" o "cmd") y verificar la instalación con:

```
bash

CopiarEditar

node -v

npm -v
```

Figura 4. Muestra un resumen visual de los pasos a seguir para instalar node.js en Windows.

2. **Instalar Node-RED desde la terminal:**

- Una vez que estemos dentro de la terminal de Windows, podemos escribir los siguientes comandos:

```
bash

CopiarEditar

npm install -g --unsafe-perm node-red
```

- Esto descargará e instalará Node-RED de forma global (-g) en tu equipo. Puede tardar unos minutos.

3. **Iniciar Node-RED:**

- En la terminal, ejecutar:

```
bash

CopiarEditar

node-red
```

- Una vez que Node-RED está instalado correctamente, cuando es invocado con el comando anterior, se abre una instancia del servidor web en su plataforma de hardware, por ejemplo, se puede acceder a dicho servidor usando el siguiente comando:

```
cpp

CopiarEditar

http://127.0.0.1:1880/
```

4. **Abrir el editor:**

- Abrir un navegador web e ingresar la dirección indicada (por defecto es http://localhost:1880).
- Se mostrará la interfaz gráfica de Node-RED lista para usarse.



Figura 5. Pasos para instalar Node.js en Windows.

1.4.2 Instalación de Node.js en Windows

Node.js es un entorno de ejecución necesario para que Node-RED funcione. A continuación, se describen los pasos para instalarlo correctamente en Windows.

Paso 1: Descargar el instalador desde la página oficial

1. Abrir el navegador web (Chrome, Edge, Firefox, etc.).
2. Acceder al sitio web oficial de Node.js: https://nodejs.org
3. Se mostrarán dos versiones:

- LTS (Long Term Support) → *Recomendada para la mayoría de los usuarios.*
- Current → *Versión más reciente, pero menos estable para entornos de producción.*

4. Hacer clic en el botón verde de la versión LTS (archivo .msi para Windows).

Paso 2: Ejecutar el instalador

1. Una vez descargado, hacer doble clic en el archivo node-vXX.X.X-x64.msi para iniciar la instalación.
2. Se abrirá el asistente de instalación de Node.js. Presionar Next.

Paso 3: Aceptar los términos y condiciones

1. Leer el acuerdo de licencia y seleccionar la casilla “I accept the terms in the License Agreement”.

2. Presionar: Next.

Paso 4: Seleccionar carpeta de instalación

1. Por defecto, Node.js se instalará en:
   C:\Program Files\nodejs\

2. Se recomienda dejar esta ubicación. Presionar: Next.

Paso 5: Seleccionar componentes a instalar

1. Asegúrate de que estén marcadas las siguientes casillas:

- Node.js runtime
- npm package manager
- Add to PATH (MUY IMPORTANTE)
- Install tools for native modules *(opcional, no necesario para Node-RED)*

2. Presionar: Next y luego Install.

Paso 6: Finalizar la instalación

1. Esperar a que la barra de progreso termine.

2. Presionar: Finish.

Paso 7: Verificar que Node.js se haya instalado correctamente

1. Abrir el Símbolo del sistema (buscar “cmd” en el menú inicio).

2. Escribir los siguientes comandos uno por uno y presionar Enter:

```
bash

CopiarEditar

node -v
```

(Debería mostrar algo como v18.19.1)

```
bash

CopiarEditar

npm -v
```

(Debería mostrar algo como 9.2.0)

Con esto, Node.js y npm estarán instalados correctamente en tu sistema, listos para continuar con la instalación de Node-RED.

🍓 *B. Instalación en Raspberry Pi (recomendada por la comunidad)*

Requisitos previos para esta plataforma:

- Preferiblemente tener instalado y actualizado el sistema operativo Raspberry Pi OS.
- Tener acceso a Internet de forma estable.

- Poder acceder a través de una terminal (local o por SSH).

Pasos a seguir:

1. **Actualizar el sistema operativo:**

   - En la terminal ejecutar:

```
bash

CopiarEditar

sudo apt update && sudo apt upgrade -y
```

2. **Ejecutar el script oficial de instalación de Node-RED:**

   - Desde el terminal usar el siguiente comando:

```
bash

CopiarEditar

bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-
    installers/master/deb/update-nodejs-and-nodered)
```

   - La ventaja de es script es que permite instalar de forma conjunta Node.js y Node-RED.

   - Probablemente debe estar pendiente a confirmar, presionando la tecla “Y” (yes) durante la instalación.

3. **Opcionalmente podemos agregar Node-RED al arranque automático:**

- El siguiente comando permitirá que Node-RED inicie automáticamente:

```
bash

CopiarEditar

sudo systemctl enable nodered.service
```

4. **Iniciar Node-RED manualmente:**

```
bash

CopiarEditar

node-red-start
```

5. **Para acceder al editor:**

- En el navegador del Pi o desde otro equipo en la red local, ingresar la IP del Pi seguido de :1880, por ejemplo:

```
cpp

CopiarEditar

http://192.168.1.100:1880
```

🐳 *C. Instalación en un contenedor Docker*

Requisitos previos:

- Docker instalado en el sistema.
- Familiaridad básica con la terminal.

Pasos a seguir:

1. **Se debe instalar Docker (si no se tiene):**
   - Ir a https://www.docker.com/products/docker-desktop/
   - Desde la página oficial anterior se debe descargar e instalar Docker Desktop según tu sistema operativo.
   - Luego, puedes verificar tu instalación con el siguiente comando:

```
bash

CopiarEditar

docker -v
```

2. **Opcionalmente se puede crear un volumen persistente (recomendado):**

```
bash

CopiarEditar

docker volume create node_red_data
```

3. **Node-RED desde un contenedor:**

- Usar el siguiente comando:

```
bash

CopiarEditar

docker run -it -p 1880:1880 -v node_red_data:/data --name mynodered
    nodered/node-red
```

- Desde la consola, al usar el comando anterior, le permitirá descargar la imagen oficial. Además, nodered/node-red se abre desde Docker Hub, en el puerto 1880 y montando un volumen para la persistencia de datos.

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

- Abrir el navegador y escribir:

```
arduino

CopiarEditar

http://localhost:1880
```

5. **Detener y reiniciar el contenedor:**

- Si se desea detener, debe usar el siguiente comando:

```
bash
```

```
CopiarEditar

docker stop mynodered
```

- En cambio, si desea reiniciar, debe usar el siguiente comando:

```
bash

CopiarEditar

docker start mynodered
```

D. Instalación de Node-RED en Android usando Termux

En el mundo del IoT los dispositivos inteligentes móviles tienen una participación importante, en todas las capas de una arquitectura IoT. Pensando en ello el equipo de desarrollo de Node-red ha creado una guía de instalación sobre dispositivos con sistema operativo Android. Hasta la fecha de esta publicación no se dispone de una instalación oficial estable sobre iOS. A continuación, comportimos los pasos a seguir para dicha instalación:

Requisitos previos recomendados:

- Un dispositivo Android moderno (con al menos 2 GB de RAM).
- Acceso a Google Play o F-Droid. En versiones actuales de Android no está disponible de manera oficial la aplicación de Termux (emulador de Linux), por tanto, debemos recurrir a la página de F-Droid para encontrar a Termux.

- Conexión a Internet.

Paso 1: Antes de instalar Node-RED, debemos instalar Termux

Opción 1: Se recomienda usar el repositorio de F-Droid para descargar el apk correspondiente

1. Ir a https://f-droid.org desde el navegador del teléfono.
2. Descargar e instalar el archivo .apk de Termux.
3. Abrir Termux desde el menú de aplicaciones.

Opción 2: Desde Google Play

- No es la versión oficial mantenida, pero puede servir en algunos casos.

Paso 2: Actualizar paquetes base de Termux

Dentro de la app Termux, escribe los siguientes comandos uno por uno:

```
bash

CopiarEditar

pkg update -y

pkg upgrade -y
```

Paso 3: Luego, debemos instalar Node.js y sus dependencias

```
bash

CopiarEditar

pkg install nodejs -y

pkg install git -y

pkg install nano -y  # (opcional, editor de texto)
```

Si queremos comprobar la versión que está instalada, debemos usar el siguiente comando:

```
bash

CopiarEditar

node -v

npm -v
```

Paso 4: Finalmente, podemos instalar Node-RED globalmente

Ahora que ya tienes Node.js funcionando, puedes instalar Node-RED con el siguiente comando:

```
bash

CopiarEditar

npm install -g --unsafe-perm node-red
```

Este paso puede tardar varios minutos dependiendo de tu red y del hardware del teléfono.

Paso 5: Para iniciar Node-RED

Como el siguiente comando podemos ejecutar una instancia de Node-RED:

```
bash

CopiarEditar

node-red
```

Después de varios segundos podemos verificar que Node-RD está corriendo y aparecerá algo similar a esto:

```
CopiarEditar

Server now running at http://127.0.0.1:1880/
```

Paso 6: Acceder desde el navegador del teléfono

1. Abre el navegador web del teléfono (Chrome, Firefox, etc.).

2. Escribe en la barra de direcciones:

```
CopiarEditar

http://localhost:1880
```

3. Verás la interfaz gráfica de Node-RED funcionando directamente desde tu Android.

Paso 7: Opcinalmente podemos ejecutar Node-RED para que corra en segundo plano

Una vez, que Termux no permite correr procesos en segundo plano por defecto, entonces podemos usar el siguiente truco:

```
bash

CopiarEditar

alias nred='node-red > /dev/null 2>&1 &'
```

Notas adicionales:

- Es posible experimentar bajos rendimientos en dispositivos antiguos.
- Conexiones al teléfono mediante Bluetooth, USB OTG o redes WiFi están permitidas.
- El uso de Node-RED desde Termux es adecuado para fines educativos, pruebas rápidas o demostraciones, no se recomienda para fiens comerciales o aplicaciones que requieran estabilidad.

1.4.3 Recomendaciones para diferentes sistemas operativos

NOde-RED , en cuanto a su instalación y operación puede variar ligeramente en dependencia del sistema operativo utilizado. Si bien la

base es la misma, ya que Node-RED corre sobre Node.js, existen algunas diferencias , por ejemplo: rutas de archivos, privilegios administrativos, comandos de terminal y herramientas de supervisión. La **¡Error! No se encuentra el origen de la referencia.** muestra las principales consideraciones y buenas prácticas a tener en cuenta al instalar Node-RED en sistemas operativos como: **Windows, Linux y macOS**.

1.4.3.1 Windows

- **Ruta de instalación**: Node.js añade automáticamente node y npm al PATH del sistema, lo cual permite llamar a Node-RED desde la terminal (cmd) sin añadir configuraciones adicionales.

- **Permisos**: Se recomienda ejecutar la terminal como administrador para evitar problemas de permisos a la hora de instalar paquetes globales con npm.

- **Firewall**: Debe permitir conexiones entrantes en el puerto 1880 cuando se necesite acceder a Node-RED desde otros dispositivos, pero en la red local.

- **Inicio automático**: No hay un gestor de este servicio por defecto. Para ello se debe crear un acceso directo o usar herramientas como *nssm* para correr Node-RED como un servicio.

1.4.3.2 Linux: Ubuntu, Debian, Raspberry Pi OS, etc.

- **Terminal nativa**: El uso de la consola es ideal para la instalación y administración de Node-RED.

- **Privilegios root**: Debe usar sudo para poder instalar paquetes globales y ejecutar scripts del sistema si fuese necesario.

- **Sistema de servicios**: Aquí, Node-RED si puede ejecutarse como un servicio mediante *systemd*, habilitando su arranque de forma automática y una gestión robusta del mismo.

- **Acceso remoto**: Se puede instalar Node-RED por SSH desde un simple navegador desde otro dispositivo conectado a la misma red.

1.4.3.3 macOS

- **Instalación de Node.js**: Existen dos formas de hacerlo, mediante el instalador .pkg disponible en su web oficial o usando el instalador Homebrew (brew install node).

- **Terminal y permisos**: Se recomienda el uso de la terminal usando privilegios de administrador (sudo) cuando sea necesario.

- **Gestión de servicios**: Aquí no usa systemd, pero se pueden crear scripts de arranque o usar herramientas como launchctl para automatizar el inicio.

- **Acceso local y remoto**: Se puede acceder desde un browser apuntando a la dirección http://localhost:1880 o mediante la IP del equipo, desde un equipo remoto dentro de la misma red.



Figura 6. Consideraciones básicas según el sistema operativo.

1.5 Crear diferentes instancias de Node-RED por sistema operativo

Como hemos comentado anteriormente, Node-red es una aplicación web y por tanto cuando, queremos tener corriendo más de una aplicación hecha con Node-red, requerimos de crear diferentes instancias dentro de nuestro servidor web instalado. A continuación, algunas pautas para conseguirlo en función del sistema operativo disponible.

🖥️ A. En el Sistema Operativo Windows: Mediante terminal CMD o PowerShell

Requisitos iniciales:

- Tener instalado Node-RED de forma global (npm install -g node-red)

- Tener diferentes carpetas para las instancias

Pasos:

1. **Crear una carpeta para cada instancia:**

```
bash

CopiarEditar

mkdir C:\Users\TuUsuario\nodered-instancia1

mkdir C:\Users\TuUsuario\nodered-instancia2
```

2. **Especificar la carpeta de usuario y puerto:**

```
bash

CopiarEditar

node-red -u C:\Users\TuUsuario\nodered-instancia1 --port 1881

node-red -u C:\Users\TuUsuario\nodered-instancia2 --port 1882
```

3. **Abrir cada instancia desde un navegador, indicando IP y puerto correspondiente:**

```
Instancia 1: http://localhost:1881

Instancia 2: http://localhost:1882
```

Nota: Para automatizar el arranque se puede crear un archivo .bat.

B. En el Sistema Operativo Linux: Ubuntu, Debian, Raspberry Pi OS, etc.

Requisitos iniciales:

- Se debe tener Node-RED ya instalado.
- No usar root, solamente con permisos de usuario normal.

Pasos:

1. **Se deben crear las carpetas de configuración para cada instancia:**

```
bash

CopiarEditar

mkdir ~/nodered-proyectoA

mkdir ~/nodered-proyectoB
```

2. **En cada instancia se llama a Node-RED con su propia carpeta y puerto:**

```
bash

CopiarEditar

node-red -u ~/nodered-proyectoA --port 1881

node-red -u ~/nodered-proyectoB --port 1882
```

3. **Opcionalmente se puede crear un servicio con systemd:**

- Se debe clonar el servicio original de Node-RED y además modificar la línea ExecStart para cada uno de los usuarios, con su respectiva carpeta y puerto.

🍎 C. En macOS

Se utilizan pasos similares al Sistema Operativo Linux:

1. **Se deben crear carpetas separadas por instancia:**

```
bash

CopiarEditar

mkdir ~/nodered-lab1

mkdir ~/nodered-lab2
```

2. **Accedemos a Node-RED desde un terminal:**

```
bash

CopiarEditar

node-red -u ~/nodered-lab1 --port 1883

node-red -u ~/nodered-lab2 --port 1884
```

3. **Se puede automatizar el arranque usando scripts de inicio (opcional):**
   Los archivos .sh se usan para dichos scripts o se puede optar por el uso de launchd para gestionar los procesos.

🐳 D. Uso de Docker para gestionar múltiples contenedores

Pasos a seguir:

1. **Se deben crear diferentes volúmenes para la persistencia de cada instancia:**

```
bash

CopiarEditar

docker volume create nodered_data1

docker volume create nodered_data2
```

2. **Para ejecutar múltiples contenedores en diferentes puertos:**

```
bash

CopiarEditar

docker run -d -p 1885:1880 -v nodered_data1:/data --name nodered1
    nodered/node-red

docker run -d -p 1886:1880 -v nodered_data2:/data --name nodered2
    nodered/node-red
```

3. **Se accede desde su navegador:**

Instancia 1: http://localhost:1885

Instancia 2: http://localhost:1886

1.6 Conociendo el Front-end de Node-RED

Si hemos llegado hasta aquí, ya tenemos a Node-RED correctamente instalado e iniciado, por tanto, ahora vamos a recorrer su interfaz gráfica web, conocida también como front-end. Recordemos que Node-RED es gráfico, por lo que desde su entorno visual vamos a diseñar, editar y ejecutar flujos de datos (programas) realizando simples acciones como el arrastre y conexión de nodos, sin tener que escribir ningún código, como es habitual en otros entornos de programación tradicional basados en texto. A través de esta sección vamos a familiarizar al lector con cada una de las áreas funcionales de su editor visual, facilitando su comprensión de forma rápida desde el primer contacto con dicha plataforma.

El primer paso que debemos hacer, es invocar la instancia de node-red desea. En nuestro caso haremos la prueba desde Windows, con una instancia creada en la carpeta llamada: Node-red 1880. El nombre debe ser según su elección, aunque se recomienda que incluya el puerto asociado a la instancia correspondiente. En nuestro ejemplo la instancia correrá en el puerto 1880, como podemos apreciar en la **¡Error! No se encuentra el origen de la referencia.**.

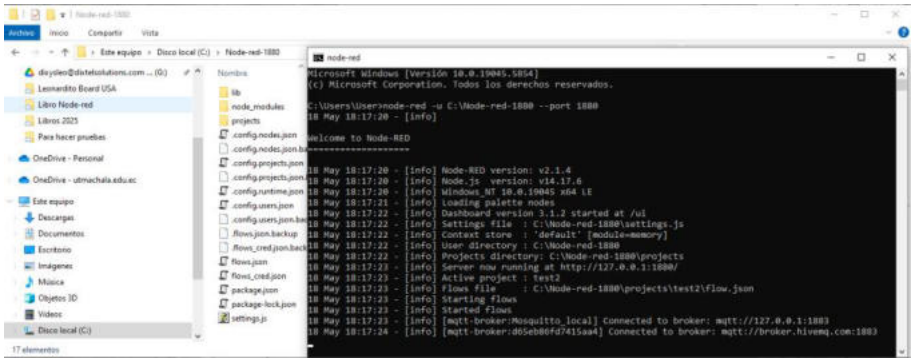
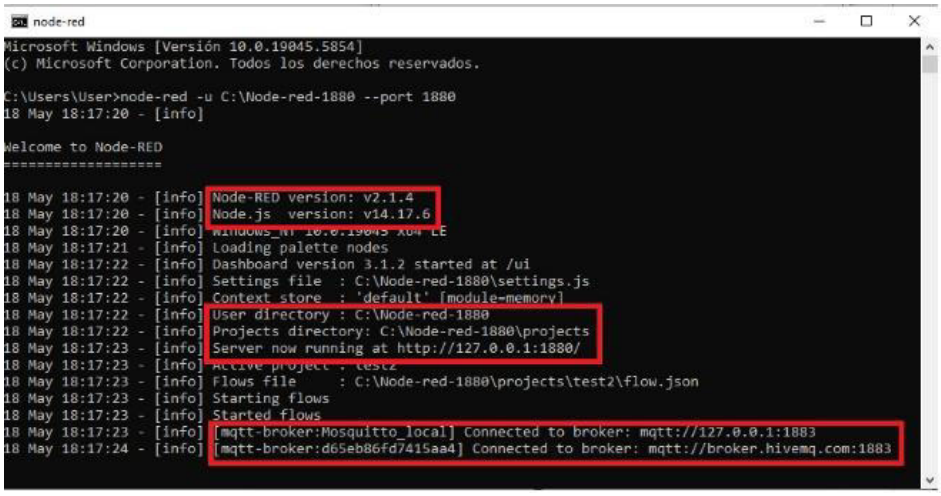


Figura 7. Consola de la instancia de node-red corriendo sobre Windows.



, la cual actúa como un visor de la bitácora de Node-red, es decir, todo lo que suceda en dicha instancia aparecerá en esta consola. Por tanto, no podemos cerrar nunca dicha consola, a no ser que precisamente sea eso lo que se desee. Resumiendo, si queremos "matar" dicha instancia solo debemos cerrar la consola. En dicha figura hemos resaltado en recuadros rojos algunos aspectos como son:

- Primer recuadro: versión de Node-red y versión de Noder-red (dependen de las versiones y actualizaciones que usted tenga en su instalación)
- Segundo recuadro: Nos indica donde se encuentra el sistema de archivos de nuestro servidor web (instancia de Node-red) y además como podemos acceder a Node-red desde un browser de internet (http://127.0.0.1:1880). Es importante indicar que a dicho servidor podrán acceder todos aquellos dispositivos que se encuentren autorizados en la red del PC o servidor donde está corriendo dicha instancia.
- Tercer recuadro: Muestra los logs de node-red, por ejemplo: que los flujos activos están corriendo y en mi caso, uno de esos flujos se ha conectado al broker MQTT Mosquitto, que está corriendo localmente en el puerto 1883 y otro flujo se ha conectado a un broker MQTT disponible en la nube, llamado HiveMQ. Estas conexiones a MQTT forman parte del argot técnico de IoT y serán

tratados con mayor detenimiento en el último capítulo de este libro.

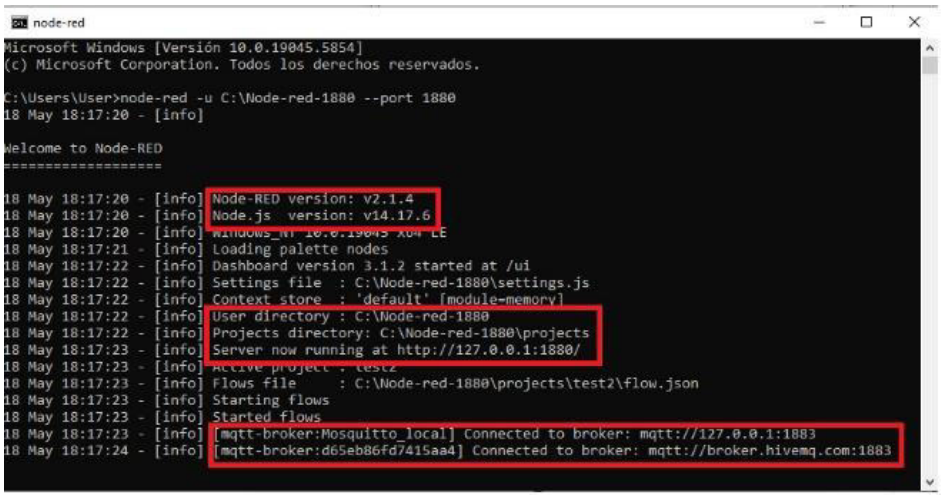


Figura 8. Logs en la consola de Node-red.

Editor visual: interfaz, áreas y menús

Una vez, que usemos un browser de internet para llegar a (http://127.0.0.1:1880), aparecerá el editor de Node-red, mostrado en la **¡Error! No se encuentra el origen de la referencia.**. Dicho editor está dividido en tres secciones principales: la paleta de nodos (a la izquierda), el área de trabajo o lienzo (en el centro), y el panel lateral (a la derecha). Cada una de estas zonas cumple una función específica en el proceso de diseño y depuración de flujos. Desde la selección de nodos, el enlace de conexiones y la configuración de propiedades, hasta el monitoreo en tiempo real, el entorno visual de Node-RED ofrece una experiencia de desarrollo intuitiva, eficiente y altamente accesible para usuarios de todos los niveles.

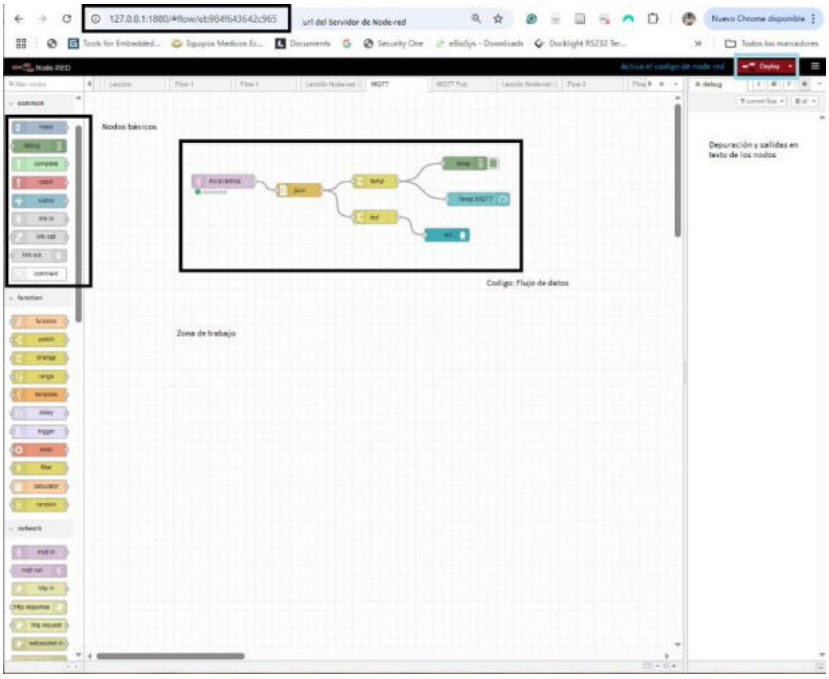


Figura 9. Editor de Node-red.

Paleta de Nodos

La paltea ubicada en el lado izquierdo de la interfaz, es un simple cajón de herramientas de Node-RED. Contiene todos los nodos disponibles organizados por categorías funcionales, como:

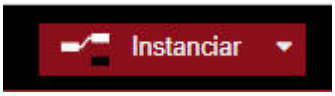
- Input (Entrada): ejemplos: inject, mqtt in, http in, etc.
- Output (Salida): ejemplos: debug, mqtt out, http response.
- Function (Procesamiento): ejemplos: function, template, change, switch, etc.
- Network, Storage y Social: según los nodos adicionales instalados.

Cada nodo representa una unidad de procesamiento que se puede arrastrar y soltar sobre el área de trabajo. Al hacer clic sobre un nodo en la paleta, aparece una breve descripción emergente que explica su función. La paleta puede personalizarse desde el "Manage Palette", permitiendo instalar nuevos nodos, deshabilitar categorías o reorganizarlas.

El Área de Trabajo (Lienzo de flujos)

Situada en el centro de la pantalla, el área de trabajo es el espacio donde el usuario diseña visualmente sus flujos de datos. Aquí se colocan los nodos, se los conecta mediante enlaces (wires) y se estructuran las distintas etapas del procesamiento de datos. Algunas características clave incluyen:

- Cada flujo es una pestaña (tab) en el lienzo, y pueden crearse múltiples flujos independientes.
- Los nodos se conectan arrastrando desde el punto de salida de uno hasta el punto de entrada de otro.
- Se pueden usar colores, comentarios y subflujos para organizar el diseño.
- La interfaz permite hacer zoom y desplazar el área de trabajo para trabajar en flujos grandes.

Además, el lienzo posee un botón de “Deploy” (Instanciar) 

en la parte superior derecha, que permite guardar y activar los cambios realizados en tiempo real.

Panel Lateral Derecho

Este panel lateral dispone de las herramientas contextuales que varían según la selección del usuario. Se divide en varias pestañas, algunas de ellas son:

- Info: Es como una ayuda puntual del nodo, ya que muestra información detallada del nodo incluyendo entradas, salidas y opciones.
- Debug: Una especie log funcional, en forma de texto podemos observar un objeto completo, muy útil para depurar errores o descubrir formatos de tramas desconocidas.
- Context Data: Es el visor de las variables de contexto :flow, global y node.
- Configuration Nodes: Lista todos los nodos de configuración usados en el flujo, por ejemplo: conexiones MQTT, HTTP o bases de datos.
- Subflow y Deploy history: En dependencia de la versión, se incluyen herramientas adicionales para flujos personalizados y control de cambios.

Como puede apreciar, este panel es fundamental tanto para la depuración, documentación, como el análisis del comportamiento de todo el sistema durante la fase de desarrollo.

1.6.1 Flujos y nodos: estructura y conexión

En Node-RED, los programas se denominan flujos, los cuales son un conjunto de nodos interconectados que dan una secuencia lógica del procesamiento de los datos. Cada nodo es una unidad funcional que realiza una operación específica: puede generar datos, procesarlos, tomar decisiones, o enviarlos a un destino, como se puede apreciar en **¡Error! No se encuentra el origen de la referencia.**. La estructura de un flujo se basa en el principio de programación reactiva, donde los mensajes viajan entre nodos a través de conexiones, y cada nodo responde a esos mensajes de acuerdo con su configuración.

Los flujos se construyen arrastrando nodos desde la paleta hacia el área de trabajo, y conectándolos mediante líneas que indican el camino que sigue un mensaje (msg) desde un nodo de salida hacia uno de entrada. La lógica que se desea programar o el camino que debe seguir un dato es controlado a través de las ramificaciones que puede tener un flujo, por tanto, los nodos disponen de conectores de entrada, incluso pueden tener múltiples salidas para procesos en paralelo.

Dentro de una pestaña, puede existir incluso más de un flujo, aunque se recomienda por limpieza y legibilidad que cada flujo resida en su propia pestaña. A su vez, un proyecto puede estar formado por varias pestañas, dando así una gran flexibilidad y escalabilidad al proyecto. Además, es posible utilizar subflujos para encapsular fragmentos de lógica

reutilizable, mejorando la modularidad y limpieza del diseño. Esta forma de estructurar la lógica convierte a Node-RED en una herramienta poderosa para representar y ejecutar procesos complejos de forma intuitiva, visual y mantenible.

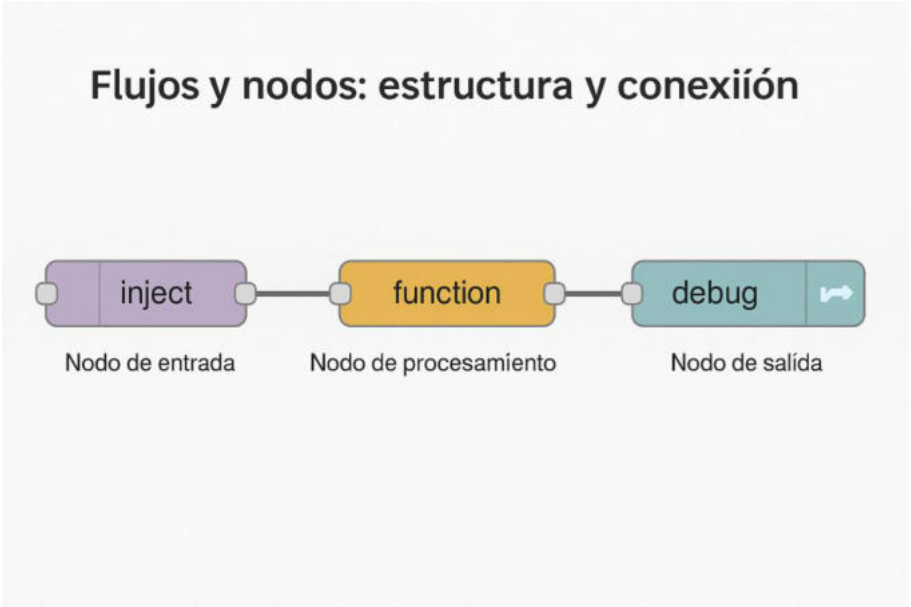


Figura 10. Estructura básica de un flujo en node-red.

1.7 Añadir e instalar nuevos nodos

Una de las grandes fortalezas de Node-RED es su **arquitectura extensible**, que permite ampliar fácilmente sus capacidades mediante la instalación de **nuevos nodos**, desarrollados por la comunidad o diseñados a medida para casos específicos, los cuales pueden ir desde integrar nuevas APIs, gestionar nuevos dispositivos, incluir protocolos nuevos o modificados, conectarse a servicios en la nube, crear sus propios dashboards, conectarse a diferentes bases de datos, incluso el uso de herramientas de inteligencia artificial modernas.

Los nodos que no vienen en la instalación inicial de Node-RED deben ser instalados desde el **administrador de paleta gráfico (Palette Manager)** o mediante la **línea de comandos con** npm directamente sobre Node-js..

1.7.1 Instalación desde la interfaz gráfica (Palette Manager)

1. Desde un navegador ingres IPy puerto, para el caso de su propia PC puede usar http://localhost:1880.

2. Hacer clic en el botón de menú (≡) en la esquina superior derecha, ver **¡Error! No se encuentra el origen de la referencia.**.

3. Seleccionar la opción **"Manage palette"** (Administrar paleta).

4. Ir a la pestaña **Install**.

5. Desde ahí, se puede buscar el nodo deseado, ya sea: por nombre, palabra clave o función- Por ejemplo: node-red-dashboard, node-red-contrib-mqtt-broker, etc..

6. Solo resta dar click sobre el nodo deseado y este se instalará.

7. Una vez instalado correctamente, el nuevo nodo aparecerá automáticamente en la paleta del lado izquierdo.

Este método es el que recomendamos, porque evitamos tener en cuenta todas las dependencias necesarias cuando se instala sobre node.js directamente. Node-RED garantiza que el nodo quedará funcional independientemente de la versión de node.js instalada.

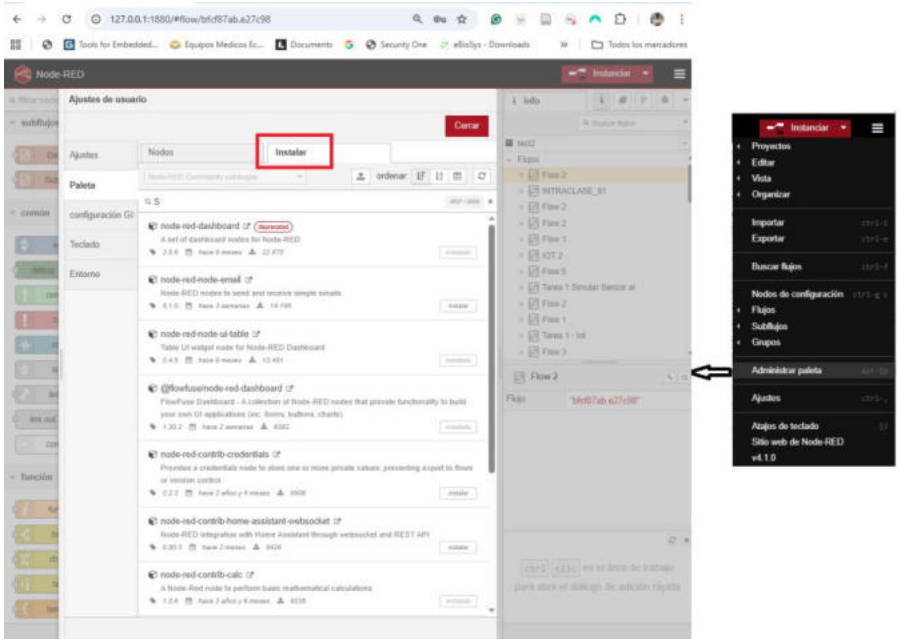


Figura 11. Instalación de nuevos nodos a través del Manage palette.

1.7.2 Instalación de nuevos nodos mediante línea de comandos (npm) sobre Node.js

A pesar de la complejidad del caso, este método resulta ser el más flexible, permitiendo mayor control, este es el método ideal para los usuarios avanzados o cuando se desea instalar versiones específicas.

1. Debemos usar un terminal, ya seacmd, PowerShell o terminal de Linux/macOS.

2. Desde el directorio de usuario de Node-RED. Use el siguiente comando:

```
bash
```

```
CopiarEditar

cd ~/.node-red
```

3. Luego para instalar un nodo debemos usar:

```
bash

CopiarEditar

npm install node-red-contrib-<nombre-del-nodo>
```

Por ejemplo:

```
bash

CopiarEditar

npm install node-red-contrib-blynk-ws
```

4. Para que la instalación tenga efecto debemos reiniciar Node-RED:

```
bash

CopiarEditar

node-red-stop

node-red-start
```

Después del reinicio un nuevo nodo aparecerá en la paleta izquierda de Node-RED. Este método también admite la instalación desde archivos .tgz o repositorios GitHub.

1.7.2.1 ☑ Recomendaciones de buenas prácticas

- Verificar siempre la compatibilidad de versiones entre el nodo y la versión de Node-RED y Node.js.
- Verfificar la reputación y que el paquete instalado tenga aún soporte en https://flows.nodered.org.
- Se recomienda encarecidamente desinstalar todos aquellos nodos que no se utilicen y así mantener el entorno limpio y ligero.
- En producción no se debe instalar ningún nodo para pruebas.

1.8 Configuración general y seguridad básicos

Ya con Node-RED instalado, podemos pasar a personalizar nuestra configuración y aplicar algunos mecanismos de **seguridad básicos**, sobre todo si lo vamos a utilizar en ambientes compartidos o públicos. Recordemos que Node-REDno incluye ni autenticación ni cifrado, por tanto, no es recomendable para accesos remotos o públicos, salvo en modo de pruebas.

En el archivo sttings.js encontramos toda la configuración de Node-RED, y este se encuentra en el directorio del usuario de Node-RED (por defecto, ~/.node-red/settings.js). Desde este archivo es posible modificar puertos, rutas de acceso, comportamiento del editor,

autenticación de usuarios, contexto de ejecución, uso de HTTPS, y otras opciones avanzadas.

1.8.1 El archivo settings.js: núcleo de configuración de Node-RED

El archivo settings.js es el **archivo de configuración principal** de una instancia de Node-RED. Lo podemos encontrar dentro del directorio de usuario (~/.node-red/settings.js en sistemas Linux/macOS o C:\Users\<usuario>\.node-red\settings.js en Windows), ver **¡Error! No se encuentra el origen de la referencia.**. Este archivo está escrito en **JavaScript (formato de objeto JSON extendido)** y permite personalizar el comportamiento del entorno de ejecución, la interfaz de usuario, la seguridad, el almacenamiento, la ejecución de proyectos, y más.

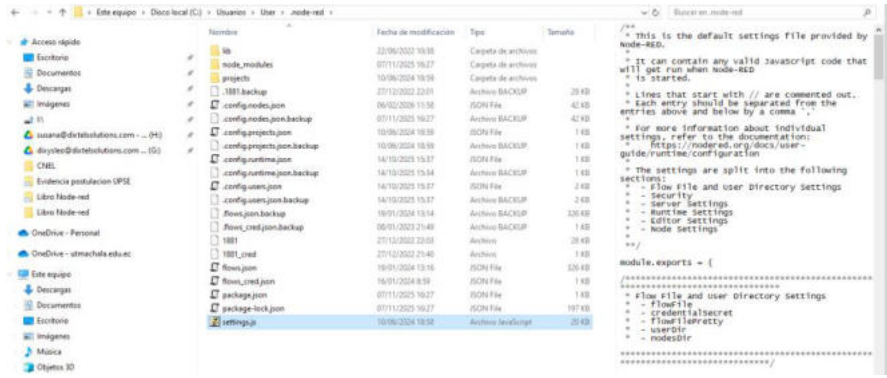


Figura 12. Ubicación del archivo de configuración sttings.js en Windows.

🔍 Estructura del **del archivo** settings.js

1. Puerto del servidor (interfaz web)

```
js

CopiarEditar

uiPort: process.env.PORT || 1880,
```

En esta parte se define el puerto del servidor cuando es accedido desde un navegador, resulta útil su uso cuando se trabaja con múltiples instancias, evitando conflictos con otros servicios.

El puerto 1880 es el puerto asignado a Node-RED, no obstante, como ya hemos indicado anteriormente podemos modificarlo para correr múltiples instancias. Abrir el archivo settings.js:

```
bash

CopiarEditar

nano ~/.node-red/settings.js
```

1. Buscar la línea:

```
js

CopiarEditar

//uiPort: process.env.PORT || 1880,
```

2. Quitar los // y modificar el puerto deseado. Aquí un ejemplo:

```
js
```

```
CopiarEditar

uiPort: process.env.PORT || 3001,
```

- **Contexto de ejecución (variables flow y global)**

```
js

CopiarEditar

contextStorage: {

   default: "memoryOnly",

   memoryOnly: { module: 'memory' },

   file: { module: 'localfilesystem' }

},
```

Permite definir el tipo de almacenamiento para las variables flow y global. Opciones disponibles:

- memory: para variables volátiles, pero se borran al reiniciar.
- localfilesystem: garantiza la persistencia entre reinicios y son almacenadas como JSON.

- **Seguridad de acceso al editor (adminAuth)**

```
js

CopiarEditar

adminAuth: {

   type: "credentials",

   users: [{

      username: "admin",

      password: "hash_de_contraseña",

      permissions: "*"

   }]

}
```

Aquí podemo definir credenciales de los usuarios y sus respectivos permisos. El formato de las contraseñas debe ser **bcrypt hash (Bcryptjs, 2026)**, generado con node-red-admin hash-pw.

- **Añadiendo seguridad para el endpoints HTTP (httpNodeAuth)**

```
js

CopiarEditar

httpNodeAuth: { user:"usuario", pass:"hash" }
```

Esto permite proteger a los nodos: tipo http in, mediante una autenticación básica.

- **Habilitando la conexión segura mediante HTTPS**

```
js

CopiarEditar

https: {

   key: require("fs").readFileSync("/ruta/privkey.pem"),

   cert: require("fs").readFileSync("/ruta/cert.pem")

},
```

Esto permite el uso de Node-RED de forma segura, actuando como un servidor HTTPS. Obviamente, se requiere que los certificados estén válidos.

- **Ruta base del editor web (httpAdminRoot)**

```
js

CopiarEditar

httpAdminRoot: "/editor",
```

Modifica la URL base para acceder al editor (por ejemplo, http://localhost:1880/editor), útil para ocultar la interfaz en entornos públicos.

- **Ruta base para APIs HTTP (httpNodeRoot)**

```
js

CopiarEditar

httpNodeRoot: "/api",
```

Permite modificar la URL de entrada para los nodos http in, por ejemplo: http://localhost:1880/api/temperatura.

- **Activar modo lectura (read-only)**

```
js

CopiarEditar

readOnly: true,
```

Convierte el editor en un visor: permite visualizar flujos pero no modificarlos ni desplegarlos.

- **Definir el editor de código embebido (code editor)**

```
js

CopiarEditar

editorTheme: {

   codeEditor: {
```

```
    lib: "monaco",
    options: {
      fontSize: 14,
      theme: "vs-dark"
    }
  }
}
```

Permite personalizar la apariencia del editor de funciones (JavaScript), incluyendo el tamaño de letra, colores, etc.

- **Personalizar el tema visual del editor**

```
js
CopiarEditar
editorTheme: {
  page: {
    title: "Mi Plataforma IoT",
    favicon: "/ruta/favicon.ico",
    css: "/ruta/estilo-personalizado.css"
```

```
  },
  header: {
    title: "Dashboard UTMACH",
    image: "/ruta/logo.png",
    url: "https://utm.edu.ec"
  }
}
```

Permite añadir logos, estilos y modificar el título del editor para entornos institucionales o personalizados.

- **Configuración de Proyectos (Git)**

```
js
CopiarEditar
projects: {
  enabled: true,
  workflow: {
    mode: "manual"
  }
```

```
}
```

Habilita la funcionalidad de **proyectos con control de versiones Git** desde el editor.

- **Configurar la consola de log**

```
js

CopiarEditar

logging: {

   console: {

      level: "info",

      metrics: false,

      audit: false

   }

}
```

Controla la verbosidad de los mensajes que se muestran en la terminal (fatal, error, warn, info, debug, trace).

- **Cargar nodos personalizados desde otra ruta**

```
js

CopiarEditar
```

```
nodesDir: "/home/user/nodos-especiales"
```

Permite cargar nodos desarrollados a medida o descargados desde otros repositorios externos.

☑ Recomendaciones de uso

- **Siempre hacer una copia de seguridad** del archivo antes de editarlo.
- Modificar usando un editor compatible con sintaxis JavaScript (como VS Code o nano).
- **Reiniciar Node-RED** después de hacer cambios para que surtan efecto.
- Validar que no haya errores de sintaxis al guardar el archivo.

1.8.2 Control de acceso con usuarios y contraseñas

Node-RED permite agregar un sistema de autenticación básico para acceder a la interfaz web del editor.

1. En settings.js, ubicar la sección:

```
js

CopiarEditar

adminAuth: {

   type: "credentials",
```

```
  users: [ /* usuarios aquí */ ]

}
```

2. Para crear una contraseña encriptada, ejecutar en terminal:

```
bash

CopiarEditar

node-red-admin hash-pw
```

3. Copiar la contraseña hash generada y crear un usuario así:

```
js

CopiarEditar

adminAuth: {

  type: "credentials",

  users: [{

    username: "admin",

    password: "$2b$08$Ejemplod3Cl4v3H4sh3adaG3n3r4d4",

    permissions: "*"

  }]

}
```

4. Guardar cambios y reiniciar Node-RED. Al acceder al editor, se pedirá el usuario y la contraseña.

1.8.3 Seguridad en despliegues remotos

Cuando Node-RED se ejecuta fuera de un entorno local, en una Raspberry Pi o plataforma similar, pero accesible desde la red, o en un servidor en la nube, es importante implementar medidas de seguridad robustas que protejan tanto el editor como los flujos, especialmente cuando se manejan datos sensibles, dispositivos conectados o servicios productivos.

A diferencia de entornos de desarrollo locales, un despliegue remoto expone públicamente la interfaz web de Node-RED (por defecto en el puerto 1880), lo que puede ser detectado fácilmente por escáneres automatizados y bots maliciosos. A continuación, se detallan las mejores prácticas para asegurar un despliegue remoto de Node-RED:

🔐 **Autenticación obligatoria para el editor**

El acceso al editor debe estar **protegido con nombre de usuario y contraseña** utilizando adminAuth en el archivo settings.js, empleando hashes bcrypt, como se explicó en el epígrafe anterior. Nunca debe dejarse abierto sin autenticación.

🔒 **Uso de HTTPS con certificado válido**

La interfaz web de Node-RED debe ser accesible solo a través de https://, utilizando un **certificado TLS/SSL:**

- En servidores públicos se puede usar **Let's Encrypt** con certbot.
- En Raspberry Pi o entornos privados, se pueden generar certificados autofirmados (self-signed) con openssl.

Esto asegura la **confidencialidad** de las credenciales y los datos transmitidos.

🔐 1. Activar autenticación para flujos HTTP

Si se usan endpoints HTTP (http in, http out), se recomienda protegerlos con autenticación.

En settings.js, habilitar la autenticación básica HTTP en esta sección:

```
js

CopiarEditar

httpNodeAuth: {user:"usuario", pass:"$2b$08$Clav3Hash..."},
```

🔐 2. Configurar HTTPS (opcional)

Para permitir una conexión segura (https://) es necesario tener un certificado SSL válido.

1. En settings.js, descomentar y completar:

```
js

CopiarEditar
```

```
https: {
  key: require("fs").readFileSync('/ruta/a/clave-privada.key'),
  cert: require("fs").readFileSync('/ruta/a/certificado.crt')
},
```

2. Node-RED ahora se ejecutará sobre HTTPS. Ideal para entornos productivos.

🛡️ 3. Acceso restringido por firewall o VPN

- Usar **firewalls (iptables, UFW, firewalld)** para permitir únicamente las IPs de confianza.
- Para mayor seguridad, desplegar Node-RED **detrás de una VPN** (como WireGuard o Tailscale).
- Alternativamente, limitar accesos con reglas de red en la nube (AWS, Azure, GCP).

🔁 4. Proxy inverso con Nginx o Caddy

Implementar un **reverse proxy** permite:

- Redirigir peticiones a https://midominio.com/red.
- Proteger con certificados TLS automáticamente.
- Limitar peticiones por IP (rate limiting).

- Integrar autenticación adicional con basic_auth, OAuth o JWT.

Ejemplo de proxy con Nginx:

```
nginx

CopiarEditar

location /red/ {

    proxy_pass http://localhost:1880/;

    proxy_set_header Host $host;

}
```

👥 5. Control de permisos por usuario

Node-RED permite definir múltiples usuarios con distintos niveles de permisos:

- read: solo visualizar flujos.
- write: editar y desplegar flujos.
- *: permisos completos (admin).

Esto es especialmente útil cuando se trabaja en equipo o en entornos académicos.

📦 6. Actualizaciones periódicas

Mantener actualizado Node-RED y sus nodos es clave para corregir vulnerabilidades conocidas:

```
bash

CopiarEditar

npm install -g --unsafe-perm node-red
```

También debe actualizarse Node.js y los paquetes instalados en el entorno.

☁️ **7. Evitar exposiciones innecesarias**

- **Nunca abrir el puerto 1880 al público sin protección.**
- Usar herramientas como **ngrok, tailscale** o túneles SSH para acceder de forma segura.
- Realizar auditorías periódicas del entorno de ejecución y los nodos instalados.

La seguridad en despliegues remotos no debe tratarse como una opción más, sino como un **componente fundamental** del diseño de cualquier solución basada en Node-RED. La combinación de autenticación, cifrado, aislamiento de red y control de versiones contribuye a la **construcción de soluciones IoT robustas, confiables y seguras**.

☑ Recomendaciones generales de seguridad:

- Cambiar el puerto por defecto en entornos públicos.

- No exponer Node-RED a Internet sin autenticación ni cifrado.
- Utilizar VPN o túneles seguros (como Ngrok, Tailscale) si se requiere acceso remoto.
- Crear copias de seguridad periódicas del directorio .node-red.

1.9 Copias de seguridad periódicas del entorno Node-RED

Realizar copias de seguridad periódicas del entorno de Node-RED es una práctica esencial para preservar la integridad de los flujos, configuraciones, credenciales y nodos instalados. Dado que toda la lógica del sistema se almacena en el directorio .node-red, este se convierte en el núcleo de la plataforma, por tanto, en el principal objetivo a resguardar ante errores del sistema, actualizaciones fallidas o pérdidas accidentales de datos. En este epígrafe se presentan distintos métodos para automatizar y gestionar respaldos según el sistema operativo utilizado, ya sea en Windows, Linux, Android (vía Termux) o entornos basados en contenedores Docker. Una implementación adecuada de estas estrategias de respaldo permite no solo garantizar la recuperación rápida del servicio ante fallos, sino también facilitar la migración y el versionado controlado en proyectos de IoT.

El directorio. node-red contiene:

- Todos los **flujos** (flows.json, flows_cred.json).
- Las **credenciales encriptadas**.
- El archivo de configuración settings.js.

- Las **dependencias** instaladas vía npm.
- Archivos de contexto persistente si están habilitados (contextStorage con filesystem).

🖥️ A. En Windows

Ruta típica:

C:\Users\<TuUsuario>\.node-red

🔁 Automatización con archivo .bat (ejecutar manual o con el programador de tareas):

1. Crear un archivo llamado backup-nodered.bat con el siguiente contenido:

```
bat

CopiarEditar

@echo off

set fecha=%date:~6,4%-%date:~3,2%-%date:~0,2%

set destino=C:\Backups\nodered-%fecha%

xcopy /E /I /Y "%USERPROFILE%\.node-red" "%destino%"

echo Backup creado en %destino%
```

2. Ejecutar el archivo manualmente o programarlo con el **Task Scheduler** de Windows para que se ejecute cada día/semana.

🐧 B. En Linux (Ubuntu, Debian, Raspberry Pi OS)

Ruta típica:

/home/<usuario>/.node-red

🔁 Script bash para copia y cron:

1. Crear el script backup-nodered.sh:

```
bash

CopiarEditar

#!/bin/bash

FECHA=$(date +%F)

DESTINO="$HOME/backups/nodered-$FECHA"

mkdir -p "$DESTINO"

cp -r ~/.node-red "$DESTINO"

echo "Backup creado en $DESTINO"
```

2. Dar permisos:

```
bash
```

```
CopiarEditar

chmod +x backup-nodered.sh
```

3. Programar con cron (ejemplo: diario a las 2:00 am):

```
bash

CopiarEditar

crontab -e
```

Y añadir:

```
bash

CopiarEditar

0 2 * * * /home/usuario/backup-nodered.sh
```

C. En Android (con Termux)

Ruta típica:

/data/data/com.termux/files/home/.node-red

Script bash y cron:

1. Crear el script backup.sh en Termux:

```
bash

CopiarEditar
```

```
#!/data/data/com.termux/files/usr/bin/bash

FECHA=$(date +%F)

mkdir -p ~/backups

cp -r ~/.node-red ~/backups/nodered-$FECHA
```

2. Hacerlo ejecutable:

```
bash

CopiarEditar

chmod +x backup.sh
```

3. Instalar cron si no está:

```
bash

CopiarEditar

pkg install cronie

crontab -e
```

4. Añadir línea para ejecutar diario:

```
bash

CopiarEditar

0 3 * * * ~/backup.sh
```

5. Iniciar cron:

```
bash

CopiarEditar

crond
```

Alternativa: subir el backup a Google Drive, Dropbox o vía scp a un servidor externo.

🐬 D. En Docker

Importante: Node-RED debe estar usando un **volumen persistente**, como se recomienda en este libro.

🔁 Copia del volumen con docker cp:

1. Obtener el ID del contenedor:

```
bash

CopiarEditar

docker ps
```

2. Copiar el directorio /data del contenedor:

```
bash

CopiarEditar
```

```
docker cp mynodered:/data ~/backups/nodered-$(date +%F)
```

🧠 Alternativa: montar backup en docker-compose o usar bind mount:

Puedes usar un volumen externo en el docker-compose.yml que sea más fácil de respaldar:

```
yaml

CopiarEditar

volumes:

  - ./nodered-data:/data
```

Entonces, solo necesitas respaldar la carpeta local ./nodered-data.

☑ Recomendaciones adicionales

- Comprimir los respaldos con zip o tar.gz para ahorrar espacio.
- Realizar respaldos automáticos **antes de instalar nuevos nodos** o modificar settings.js.
- Subir periódicamente los backups a la nube o a un disco externo.
- Restaurar el backup es tan simple como **reemplazar el directorio** .node-red y reiniciar Node-RED.

Elemento de cierre

Resumen del capítulo:

En este capítulo aprendiste qué es Node-RED, cómo se estructura su arquitectura, cómo instalarlo en distintos sistemas operativos, y cómo comenzar a trabajar con su editor visual. También configuraste medidas básicas de seguridad y aprendiste a instalar nuevos nodos para extender sus funcionalidades.

Preguntas de revisión

❓ **Pregunta 1:** ¿Qué tecnología debe estar instalada para que funcione Node-RED?

✍️ a) Python

✍️ b) Java

✍️ c) Node.js

✍️ d) Apache

☑️ **Respuesta correcta:** c) Node.js

❓ **Pregunta 2:** ¿Cómo se accede al editor de Node-RED?

✍️ a) Desde una app nativa

✍️ b) Vía navegador en localhost:1880

✍️ c) Por consola directamente

✍️ d) Desde el explorador de archivos

☑ **Respuesta correcta:** b) Vía navegador en localhost:1880

❓ **Pregunta 3:** ¿Qué archivo contiene las configuraciones principales del entorno?

✍ a) flows.json

✍ b) package.json

✍ c) settings.js

✍ d) config.env

☑ **Respuesta correcta:** c) settings.js

❓ **Pregunta 4:** ¿Cuál de estos métodos permite automatizar copias de seguridad?

✍ a) node-red-contrib-backup

✍ b) system

✍ c) cron

✍ d) npx

☑ **Respuesta correcta:** c) cron

Clave de respuestas

Pregunta	Respuesta correcta
1	c) Node.js
2	b) Navegador
3	c) settings.js
4	c) cron

CAPÍTULO II

2 PROGRAMACIÓN LÓGICA EN NODE-RED

🎯 Objetivos de aprendizaje:

- Comprender la lógica de los nodos básicos de flujo (inject, debug, switch, change, template, function).
- Construir estructuras condicionales, bucles, temporizadores y transformaciones de datos.
- Manipular diferentes tipos y formatos de datos (string, number, JSON).
- Aplicar buenas prácticas en la organización visual y lógica de los flujos.

🛠️ Competencias:

- Diseña flujos lógicos y funcionales usando programación visual.
- Implementa condicionales, ciclos y retrasos en el procesamiento de datos.
- Estructura flujos ordenados, claros y reutilizables.
- Aplica técnicas de depuración y documentación visual en el desarrollo.

❓ Preguntas de enfoque:

- ¿Qué nodos permiten definir lógica condicional en Node-RED?
- ¿Cómo puedo transformar los datos que fluyen por el sistema?
- ¿Qué diferencia hay entre programar con nodos change y con function?
- ¿Cómo estructuro un flujo para que sea fácil de entender y mantener?

En este capítulo vamos a profundizar en los principios de lógica de programación visual aplicados dentro del entorno de Node-RED, permitiendo al lector desarrollar flujos cada vez más funcionales, eficientes y estructurados. A través del análisis y uso de los nodos básicos (core nodes) y herramientas de control como condicionales, bucles, temporizadores y funciones personalizadas, además, se abordará la manera en que se construyen algoritmos y estructuras de decisión sin necesidad de escribir código extenso, pero manteniendo la lógica y flexibilidad de un sistema completamente programable.

Luego, el lector será capaz de utilizar nodos fundamentales para manipular mensajes, construir condiciones lógicas, controlar el tiempo de ejecución mediante delays, loops y temporizadores, y organizar flujos de forma modular y mantenible. Además, se introducirá el uso de subflujos, antes mencionado, comentarios, y técnicas para escalar y combinar datos, para de esta forma desarrollar soluciones IoT reales con una lógica robusta y adaptable. En esta fase pasamos de amateur a profesional y se comienza la transición desde el uso elemental del editor

hacia una programación visual consciente, estructurada y orientada a buenas prácticas.

2.1 Nodos básicos (Core nodes)

Los **nodos básicos**, también conocidos como *core nodes (The Core Nodes : Node-RED, 2026)*, constituyen el conjunto fundamental de herramientas en Node-RED. Estos son los nodos que vienen incluidos por defecto en cualquier instalación y permiten construir flujos funcionales sin necesidad de instalar extensiones adicionales. Comprender su comportamiento, propósito y formas de conexión es vital para dominar la lógica visual en Node-RED y sentar una base sólida para el desarrollo de proyectos IoT.

Nodos: inject, debug, change, function, template

Cada nodo básico tiene una función específica en el procesamiento de mensajes (msg) que fluyen a través del sistema. Algunos de estos nodos generan datos (como inject), otros los modifican (change, function, template) y algunos permiten visualizar el resultado (debug). A continuación, se describen los más representativos:

- **Inject**

- **Imagen:** 

- **Función:** Genera un mensaje msg de forma manual o periódica.

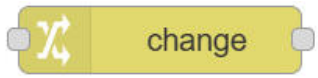
- **Uso común:** Simular datos, iniciar flujos, programar envíos por tiempo.
- **Configuración destacada:** permite definir payload, topic y el intervalo de ejecución.

◆ **Debug**

- **Imagen:** 

- **Función:** Muestra en el panel lateral los datos que viajan por el flujo.
- **Uso común:** Verificar estructuras msg.payload, msg.topic o variables internas.
- **Configuración destacada:** permite elegir qué parte del mensaje mostrar (entero, solo payload, etc.).

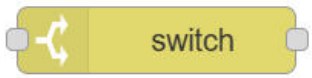
◆ **Change**

- **Imagen:** 

- **Función:** Modifica propiedades del mensaje sin necesidad de código.
- **Uso común:** Reasignar valores a msg.payload, eliminar propiedades, copiar datos entre campos.

- **Configuración destacada:** ofrece operaciones tipo *set*, *change*, *delete*, *move*.

🔹 **Switch**

- **Imagen:** 

- **Función:** Crea una bifurcación condicional en el flujo.
- **Uso común:** Enrutar mensajes según su contenido, aplicar lógica if.
- **Configuración destacada:** múltiples salidas, condiciones por valor, tipo de dato o expresión.

🛠️ **¿CÓMO SE CONFIGURA? – Nodo** switch **para condiciones múltiples**

1. Añade un nodo switch.
2. En “property”, coloca msg.payload.
3. Agrega reglas como:
   - == 0 → salida 1
   - > 0 && <= 25 → salida 2
   - > 25 → salida 3
4. Conecta cada salida a distintas acciones o rutas del flujo.

- **Template**

- **Imagen:** 

- **Función:** Genera contenido en formato HTML, texto plano o JSON con campos dinámicos.

- **Uso común:** Crear respuestas personalizadas, construir estructuras complejas.

- **Configuración destacada:** permite insertar valores con {{msg.payload}}.

- **Function**

- **Imagen:** 

- **Función:** Permite escribir lógica personalizada en JavaScript (Writing Functions : Node-RED, 2026) y (W3Schools.com, 2026).

- **Uso común:** Procesamiento avanzado, condicionales complejos, estructuras dinámicas.

- **Configuración destacada:** acceso completo al objeto msg, uso de context, flow y global.

- **Comment**

- **Imagen:** 

- **Función:** Añade anotaciones visuales al flujo.
- **Uso común:** Documentar procesos, organizar áreas del diagrama.
- **No afecta la lógica** del flujo.

El dominio de estos nodos permite resolver una gran variedad de problemas prácticos en IoT, desde la lectura de sensores, la transformación de datos, el control de actuadores o la generación de notificaciones, es decir, todo el pipeline típico de un dato en IoT. En los siguientes apartados, se estudiarán estructuras de control más avanzadas para expandir estas capacidades.

2.2 Hola mundo

Es hora de ver node-red en acción y lo haremos al estilo del clásico “hola mundo”, a continuación, explicaremos paso a paso como conseguirlo:

- Arrastramos el nodo Inject hacia la zona central y lo configuramos según la **¡Error! No se encuentra el origen de la referencia.**13.

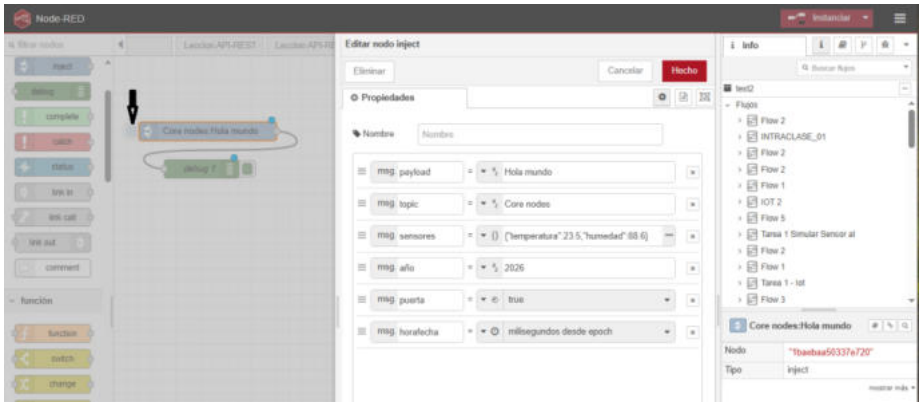


Figura 13. Configuración del nodo Inject.

Este nodo, es sumamente versátil, puede ser usado para pruebas de concepto, como para crear automatizaciones temporales, es decir, asociadas a tiempo, creando o invocando la acción de todo el flujo que se encuentra unido a él. En este ejemplo, hemos creado diferentes propiedades de un mensaje para que se pueda apreciar algunos de los diferentes tipos de datos disponibles, de arriba hacia abajo aparecen:

- msg.payload: (tipo texto) y lo inicializamos con el texto: “Hola mundo”
- msg.topic: (tipo texto) y lo inicializamos con: Core nodes
- msg.sensores: (tipo json) y lo inicializamos con valores de temperatura y humedad.
- msg.Año: (tipo numerico) y lo inicializamos con: 2026
- msg.Puerta: (tipo boolean) y lo inicializamos con: true
- msg.Horafecha: (tipo timestamp), devuelve el timestamp cuando el nodo es ejecutado.

Es importante indicar que para que este nodo se ejecute debemos presionar el botón que se encuentra en el extremo izquierdo del nodo.

- Luego, arrastramos el nodo Debug y lo configuramos según **¡Error! No se encuentra el origen de la referencia.**14.

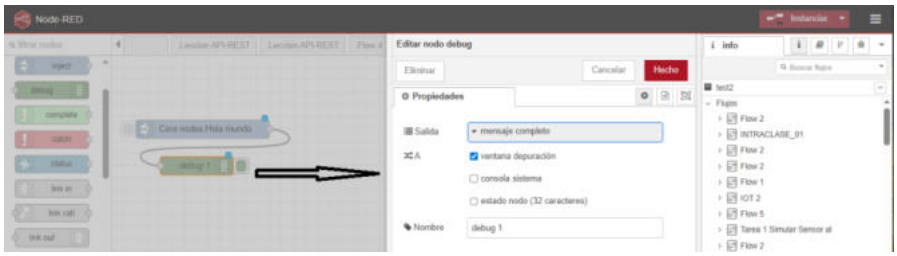
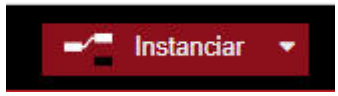


Figura 14. Modo completo del nodo Debug.

Para que se pueda ver todo el objeto, es decir, todas las propiedades del objeto en la ventana de debug, debemos activar el modo Completo, caso contrario solo mostrará la propiedad Payload. Pruébalo …

- Unimos ambos nodos
- Hacemos un Deploy 

- Observamos el resultado en la zona de debug, ver **¡Error! No se encuentra el origen de la referencia.**15. Para ello debemos presionar el icono  en el panel lateral derecho.

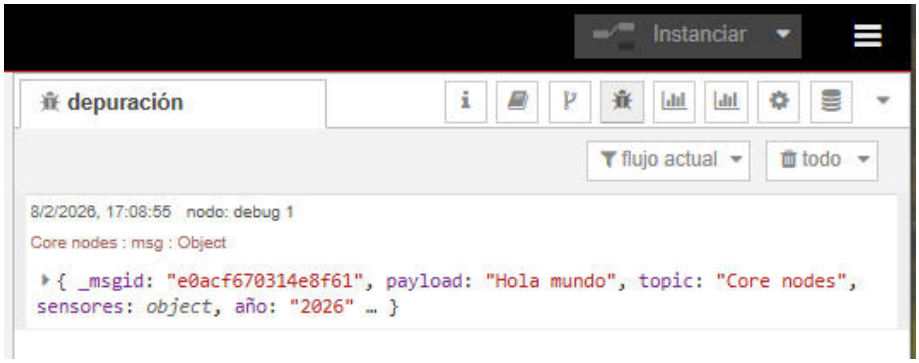


Figura 15. Ejemplo de una salida en la zona Debug. Elaboración.propia.

- El resultado debe quedar como la **¡Error! No se encuentra el origen de la referencia.**16

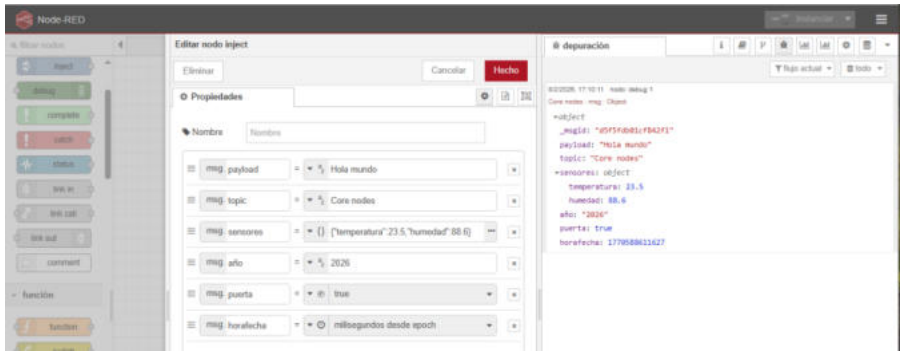


Figura 16. Ejemplo: Hola mundo con debug completo.

☑ **ATENCIÓN – Usa variables de contexto con precaución**

Almacenar datos en flow o global puede ser útil para compartir información entre nodos, pero **no olvides limpiar las variables cuando ya no se usen** o si pueden crecer indefinidamente.

☑ **ATENCIÓN – Diferencia entre** change **y** function

El nodo change es ideal para reemplazos simples, reasignaciones o transformaciones sin código. Solo debes usar un nodo function si necesitas lógica condicional compleja, cálculos o validaciones múltiples.

2.3 Estructuras de control

En programación, las estructuras de control son los mecanismos que permiten tomar decisiones, repetir procesos y controlar el flujo de ejecución de manera lógica. Aunque Node-RED se basa en un paradigma de programación visual, proporciona herramientas potentes para implementar estas estructuras mediante nodos configurables que responden a condiciones, eventos o tiempos definidos (Node-RED Cookbook : Node-RED, 2026).

Aquí, se aborda el uso práctico de las principales estructuras de control en Node-RED: condicionales, bucles, temporizadores, retardos y escalado de datos. Estas estructuras permiten implementar inclusive algoritmos reactivos complejos, automatizar tareas en función de entradas dinámicas, y construir procesos que se adaptan al tiempo o al contenido de los mensajes que fluyen.

Condicionales y decisiones

Los nodos switch permiten bifurcar el flujo de datos según condiciones del mensaje de entrada. Cada condición puede generar una salida diferente, replicando el comportamiento de estructuras if...else.

Ejemplo práctico:

Redirigir lecturas de temperatura:

- Si msg.payload < 20 → activar calefacción
- Si msg.payload >= 30 → activar ventilación

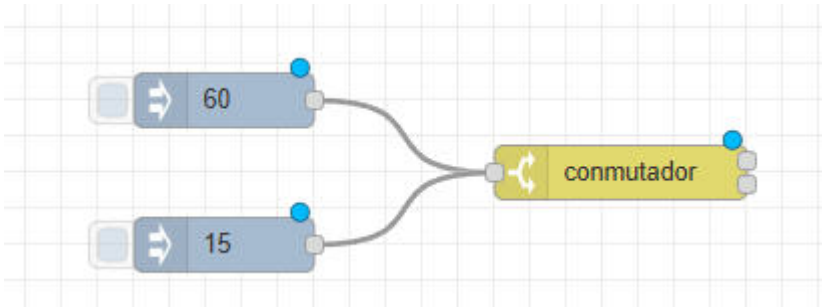


Figura 17. Ejemplo de uso del nodo Switch.

En la **¡Error! No se encuentra el origen de la referencia.**17 se utilizaron dos nodos inject para facilitar el ejemplo y solo presionar cada botón y enviar el valor deseado, ya sea 60 para invocar la activación de la ventilación o 15 para activar la calefacción. El nodo switch fue configurado como se muestra en la **¡Error! No se encuentra el origen de la referencia.**18.

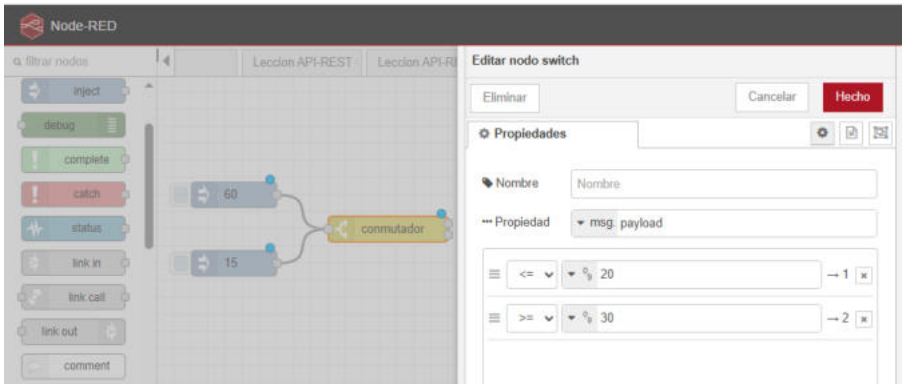


Figura 18. Configuración del nodo Switch.

Las acciones deseadas, en este caso, estamos simulando dichas acciones con textos de activación de calefacción o ventilación, fueron generados con el nodo Change, en este caso para forzar el valor del payload en esta parte del flujo, como se aprecia en la **¡Error! No se encuentra el origen de la referencia.**19.

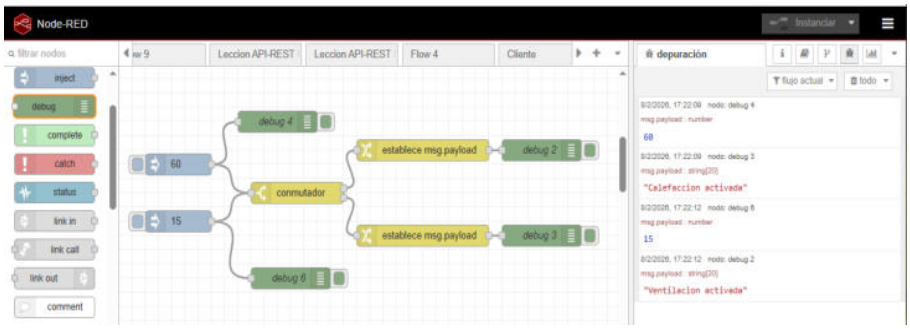


Figura 19. Uso del nodo Change para definir un valor específico del payload.

El nodo function también puede usarse para condiciones más complejas con expresiones JavaScript:

```
javascript
```

```
CopiarEditar

if (msg.payload.humedad > 60) {

   return [msg, null]; // salida 1

} else {

   return [null, msg]; // salida 2

}
```

Loops y repeticiones

Node-RED no incluye bucles clásicos (for, while) como los lenguajes tradicionales, ya que está orientado a flujos de eventos. Sin embargo, los bucles pueden simularse mediante las siguientes posibles soluciones:

☑ Solución 1. **Usar el nodo** inject **con repetición automática**

Permite generar un mensaje cada cierto tiempo.

- **Configura un nodo** inject con:
  - Tipo de dato: timestamp
  - Intervalo: cada X segundos
- Se usa para crear **loops temporizados** (polling, monitoreo, animaciones).

Ejemplo:

Haremos dos lazos, el primero lo usaremos para simular el envio de datos de un sensor de temperatura cada 1minuto, dicho valor será guardado en una variable global y el segundo lazo se encarga de monitorear el valor de temperatura cada 30 segundos, leyendo de la variable global y hacer la acción correspondiente. Para esto reutilizaremos parte del flujo que usamos en el ejemplo del uso del nodo switch.

En la **¡Error! No se encuentra el origen de la referencia.**20, el flujo superior corresponde al lazo 1 y el flujo inferior al lazo 2. En la zona del debug (panel lateral derecho), se demuestra que el lazo 1 se repite exactamente cada 1min y el lazo 2 cada 30 segundos y cuando se cumplen las condiciones definidas en la condicional del nodo switch, ya explicado anteriormente se envia el mensaje correspondiente.

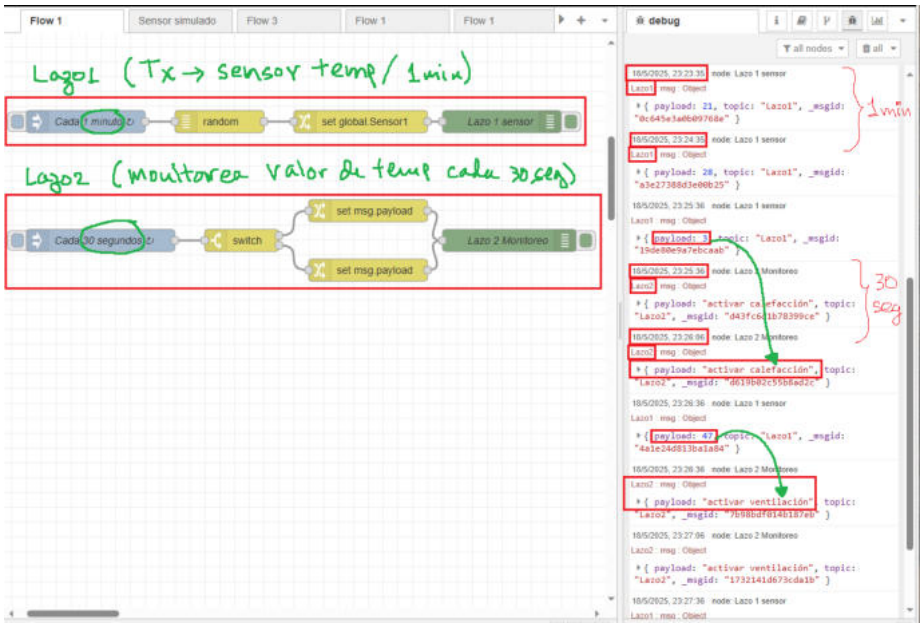


Figura 20. Lazo repetitivo usando el nodo Inject.

Los nodos Injects del lazo 1 y 2 son configurados como se muestra en la **¡Error! No se encuentra el origen de la referencia.**1.

Nota: El nodo: random 

debe ser instalado desde Manage Palette.

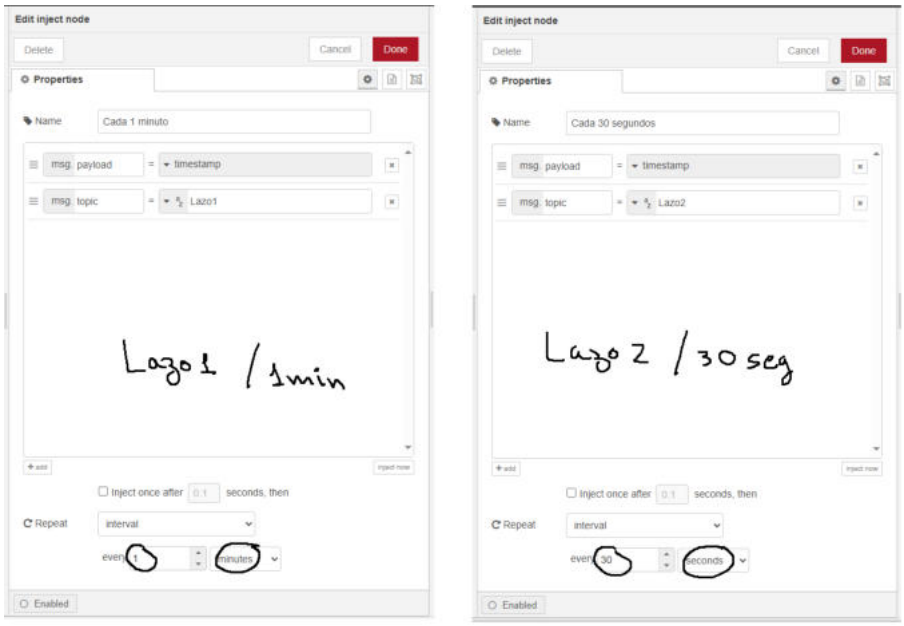


Figura 21. Configuracion del nodo Inject para conseguir lazos repetitivos.

☑ Solución 2: Nodo function conectado a sí mismo

Crear un flujo para que el nodo function **reemita el mensaje al mismo nodo** function, creando así un ciclo controlado.

🔁 Lógica del bucle tipo for

- El nodo inject inicializa la variable count = 0.

- El nodo function incrementa count y se reenvía a sí mismo hasta llegar al límite.
- Se introduce un nodo delay para simular un intervalo entre iteraciones.

🧠 Código del nodo function:

```
javascript

CopiarEditar

let count = msg.count || 0;

let limit = 5;



if (count < limit) {

    msg.payload = "Iteración " + count;

    msg.count = count + 1;

    return [msg, null];  // Continuar el bucle

} else {

    return [null, { payload: "Bucle terminado" }];

}
```

Este nodo tiene **2 salidas**:

1. Se conecta de nuevo al mismo function (pasando por un delay)

2. Sale al debug cuando se termina el bucle

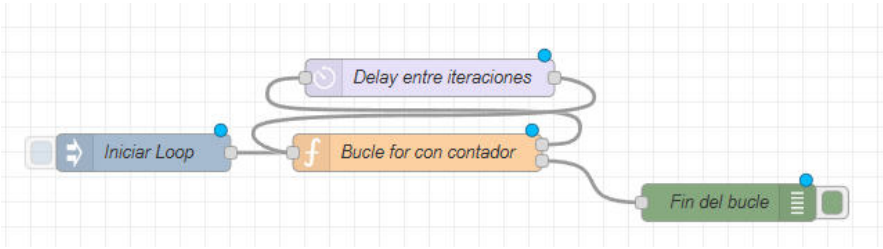


Figura 22. Lazo creado en el mismo flujo a través del nodo Delay.

Como se puede apreciar en la **¡Error! No se encuentra el origen de la referencia.**2 se ha conseguido un lazo con 5 iteraciones. A continuación, explicaremos como se crea el lazo o bucle.

- El mensaje inicial (msg.count = 0) llega desde el nodo inject, ver en la **¡Error! No se encuentra el origen de la referencia.**3.

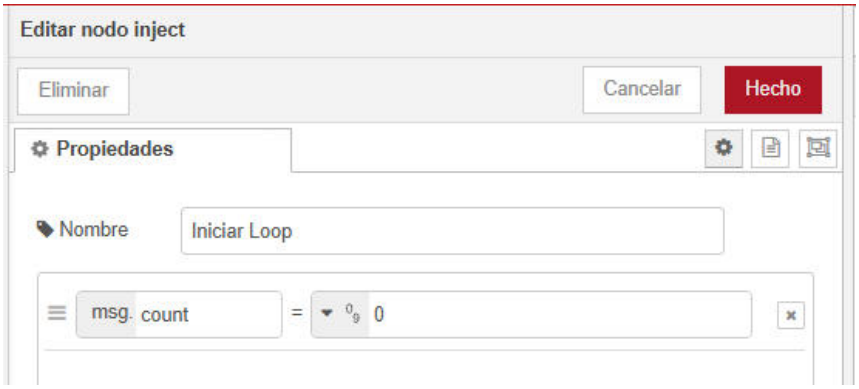


Figura 23. Configuracion del nodo Inject incializando una variable.

- El nodo function evalúa y devuelve el mensaje incrementado.

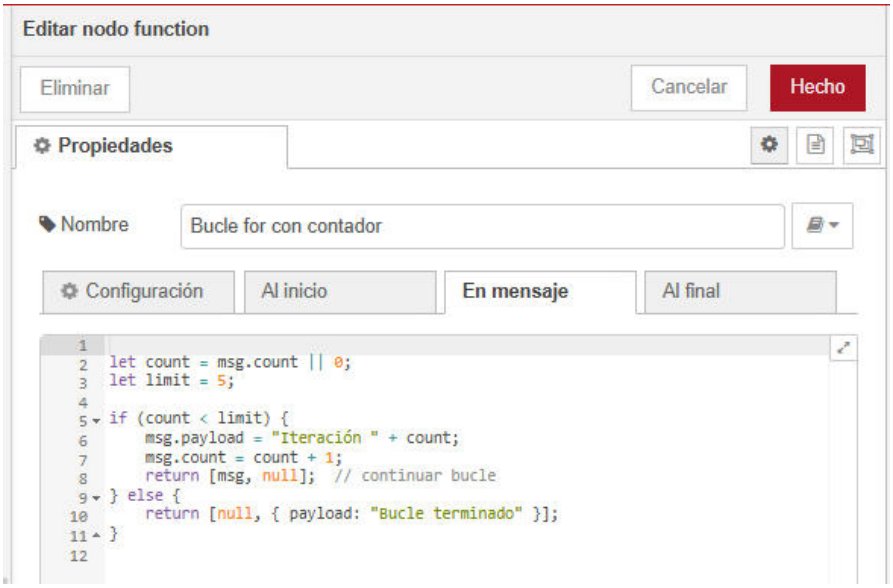


Figura 24. Configuración del nodo function para crear bucles.

A continuación, una explicación detallada de la **¡Error! No se encuentra el origen de la referencia.**:

1 let count = msg.count || 0;

- Intenta obtener el valor de count desde el mensaje entrante (msg.count).
- Si no existe (por ejemplo en la primera ejecución), lo inicializa con 0.

2 let limit = 5;

- Define un **límite superior** para las repeticiones del bucle.
- En este caso, se repetirán **5 veces** (desde 0 hasta 4).

3 if (count < limit) { ... }

- Verifica si el contador aún no ha alcanzado el límite.
- Si **sí**, se ejecuta el cuerpo del bucle.
- Si **no**, se termina el bucle y se emite el mensaje final.

4 msg.payload = "Iteración " + count;

- Genera un mensaje de salida con el texto: Iteración 0, Iteración 1, etc.
- Este será visible en el panel debug.

5 msg.count = count + 1;

- Incrementa el contador y lo **guarda en el propio mensaje** para que la siguiente iteración lo reciba.

6 return [msg, null];

- **Devuelve el mensaje por la primera salida**, que está conectada a un nodo delay, el cual reinyecta el mensaje nuevamente al nodo function, simulando el ciclo.
- El segundo valor (null) indica que **no se emite nada por la segunda salida** en esta iteración.

7 else { return [null, { payload: "Bucle terminado" }]; }

Cuando el contador llega al límite (por ejemplo count == 5):

- No se continúa el ciclo.
- Se emite un mensaje de texto "Bucle terminado" por la **segunda salida**, que está conectada al nodo debug.

- El nodo delay pausa 1 segundo y lo reenvía al mismo nodo function**,** ver ¡Error! No se encuentra el origen de la referencia.**5**.

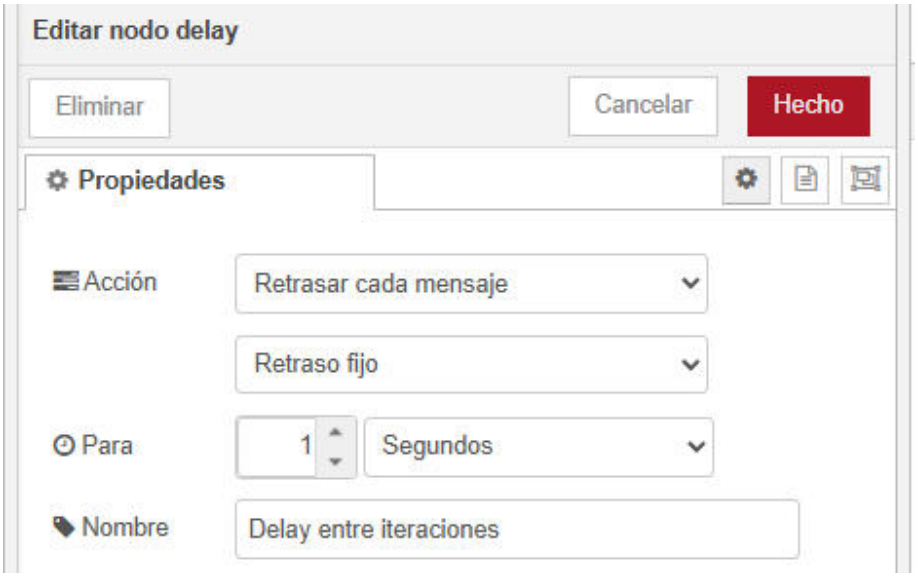


Figura 25. Configuración del nodo delay.

- Esto se repite hasta que count == 5.
- Se emite el mensaje final y el ciclo termina.

☑ Solución 3: crear un bucle sobre un array

Este es un caso típico de gestión de un arreglo de valores, que permite la gestión de cada elemento del arreglo por separado simulando un bucle “for”.

Por ejemplo, queremos convertir los valores de temperatura de 4 sensores de Celcius a Fahrenheit. Los valores de temperatura se encuentran en un array. [21.5, 22.0, 23.1, 24.0]

Para dar solución a este problema usaremos en conjunto un par de nodos: Split – Join. El primero separa los elementos y Join los une nuevamente, como se aprecia en la **¡Error! No se encuentra el origen de la referencia.**6, el arreglo incial (Grados celcius) y el arreglo final (Grados Fahrenheit).

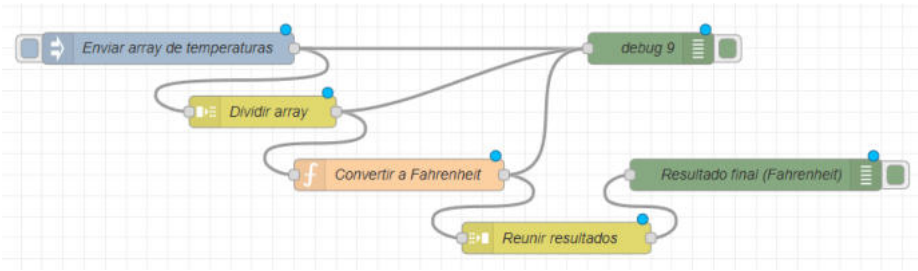


Figura 26. Gestión de arreglos usando la combinación de nodos Split - Join. .

El arreglo incial, es pasado a través del nodo Inject como un Json, como se apecia en la **¡Error! No se encuentra el origen de la referencia.**7.

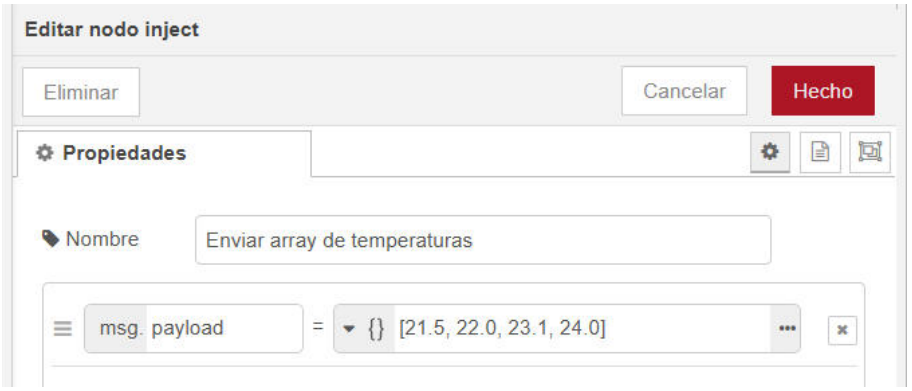


Figura 27. Arreglo incial en forma de Json.

Luego, el nodo Split divide cada elemento del arreglo con la configuración mostrada en la **¡Error! No se encuentra el origen de la referencia.**8.

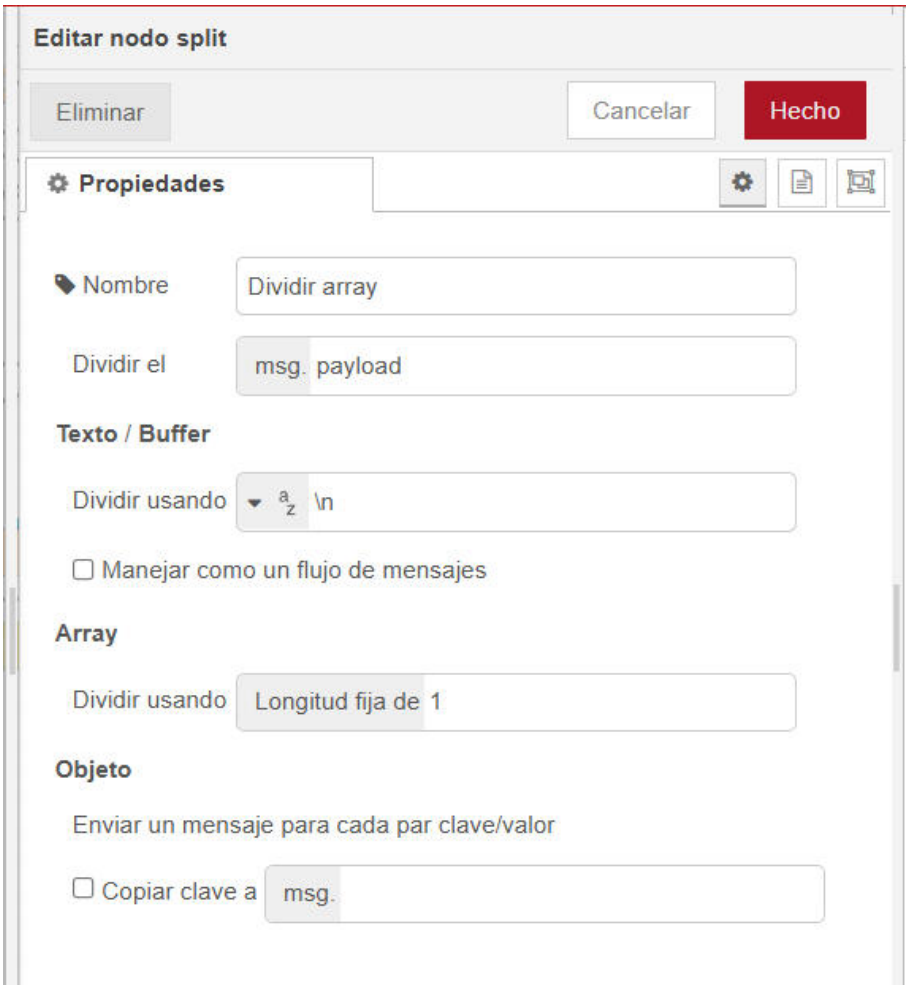


Figura 28. Configuración del nodo Split.

Luego que los elementos han sido divididos, estos pueden ser procesados individualmente cada uno de ellos con los nodos que sigan en el flujo, en nuestro caso, hemos usado el nodo function para visualizar

la conversión de grados, como gestión de cada elemento, ver **¡Error! No se encuentra el origen de la referencia.**9.

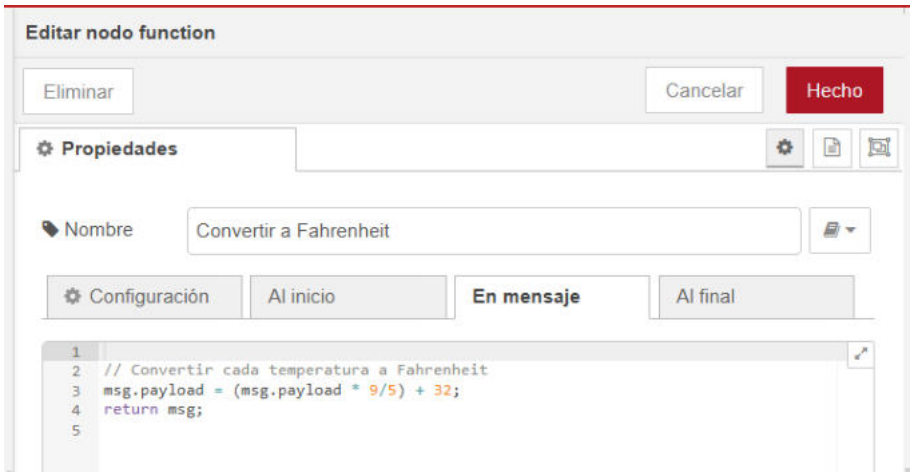


Figura 29. Configuración del nodo Function para gestionar cada elemento del arreglo.

Finalmente, esta solución requiere del nodo Join en conjunto con el nodo Split. En la figura **¡Error! No se encuentra el origen de la referencia.**30, se muestra la recuperación de elementos de forma automática.

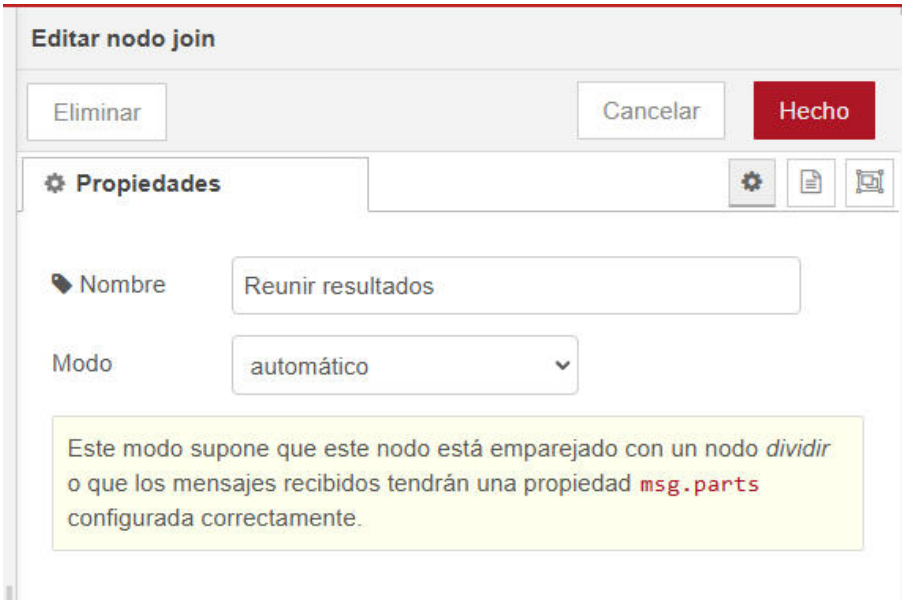
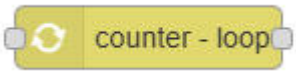


Figura 30. Configuración automática del nodo Join.

☑ Solución 4: Uso de la librería **node-red-contrib-loop-processing**

Esta librería es una solución de "terceros", es decir, es mantenida por la gran comunidad "Open" que enriquece a Node-red constantemente. Esta librería debe ser instalada desde el Manage Palette y fue tomada de [node-red-contrib-loop-processing (node) - Node-RED].

Una vez instalada dispondremos de 3 nuevos nodos:

- Counter-loop 

, simula un lazo For
- Array-loop 

, lazo para gestionar un Array

- While-loop 

, simula un lazo While

Esta biblioteca instala además de los nodos mostrados unos flujos de ejemplo, los cuales explicaremos a continuación:

Flujo de ejemplo: Counter-loop

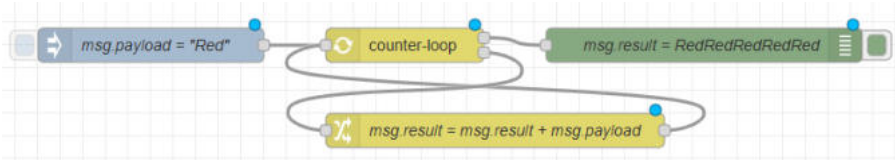


Figura 31. Flujo del ejemplo de Counter-loop. Puede encontrar este flujo en los ejemplos que se instalaron junto a la librería node-red-contrib-loop-processing.

El flujo de la **¡Error! No se encuentra el origen de la referencia.**31 simula un lazo For, con 5 iteraciones (For i=0 to 4) y como acción a realizar en el lazo será concatenar cada uno de los elementos de entrada a la salida. Por tanto, el resultado de la concatenación irá creciendo en cada iteración y al final tendremos 5 elementos concatenados. En este ejemplo, el elemento de entrada será un string (la palabra Red) y en la **¡Error! No se encuentra el origen de la referencia.**32, podemos ver cómo va cambiando el resultado con cada iteración.

En el flujo podemos identificar perfectamente, un nodo de entrada al lazo (Inject), un nodo de salida (Debug), dos nodos involucrados en el lazo, el nodo conuter-loop que contiene propiamente la lógica de las iteraciones y la acción a realizar en cada iteración se encuentra en el nodo Change.

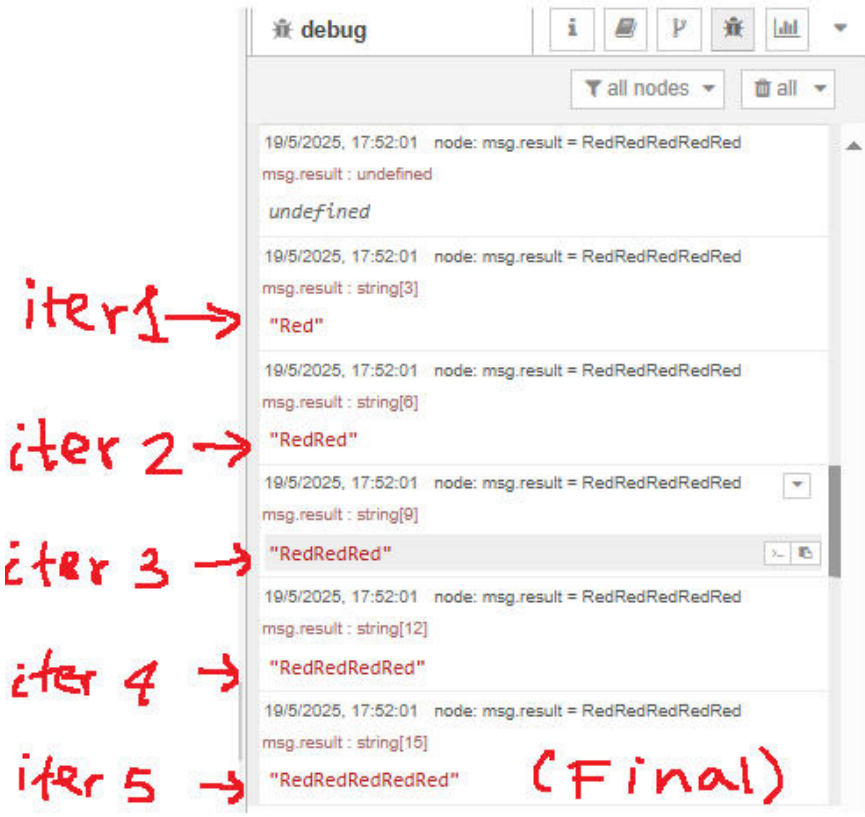


Figura 32. Resultado de cada iteración del Counter-loop.

La **¡Error! No se encuentra el origen de la referencia.**33, muestra la configuración del nodo inject, donde se inicializa el lopp con un string: Red en el payload.

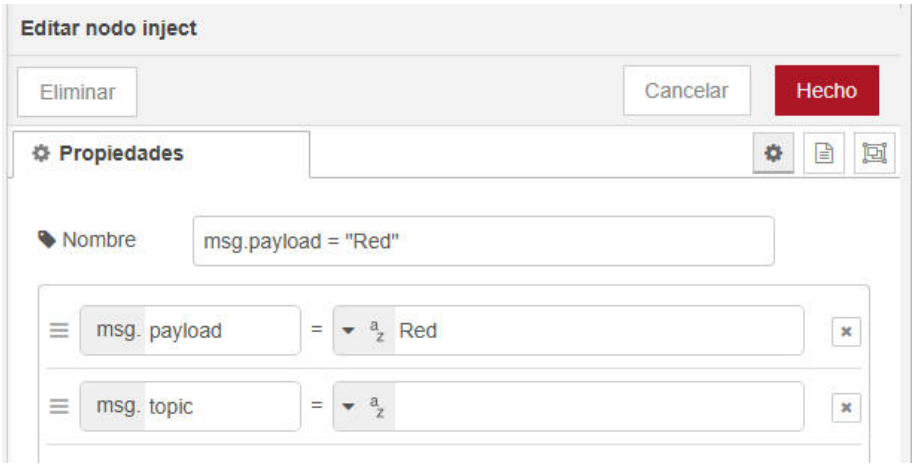


Figura 33. Configuración del nodo Inject para inicializa rel loop.

Luego, la **¡Error! No se encuentra el origen de la referencia.**34, muestra la configuración del nodo principal del loop, en este caso con 5 iteraciones. Es importante indicar, que este nodo dispone de 2 salidas, la salida superior (Flase) cuando ya no se cumpla condición del lazo y la salida inferior (true) durante cada iteración. En este ejemplo hemos conectado dicha salida tambien al nodo debug para poder visualizar la salida en cada iteración.

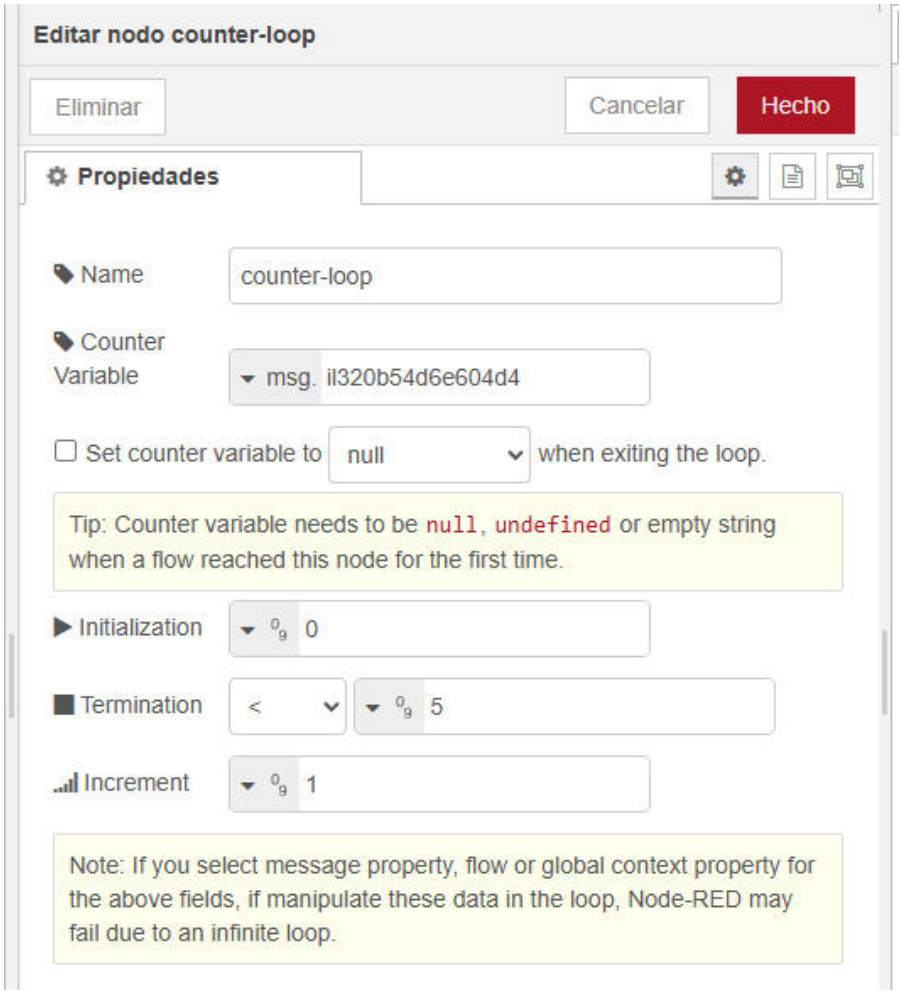


Figura 34. Configuración del nodo Counter-loop con 5 iteraciones.

Como habíamos comentado, en cada iteración la acción de este ejemplo del lazo es concatenar, a través de un Join, las salidas de cada iteración. La **¡Error! No se encuentra el origen de la referencia.**35 y **¡Error! No se encuentra el origen de la referencia.**36, muestran las acciones (Join) que en este ejemplo se realizan dentro del lazo.

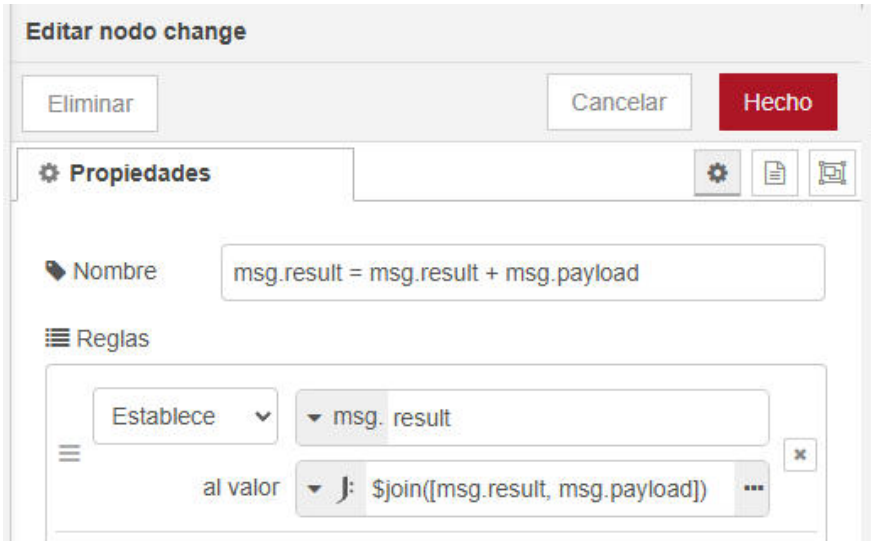


Figura 35. Configuración del Change del flujo Counter-loop, el procesamiento es un simple Join.

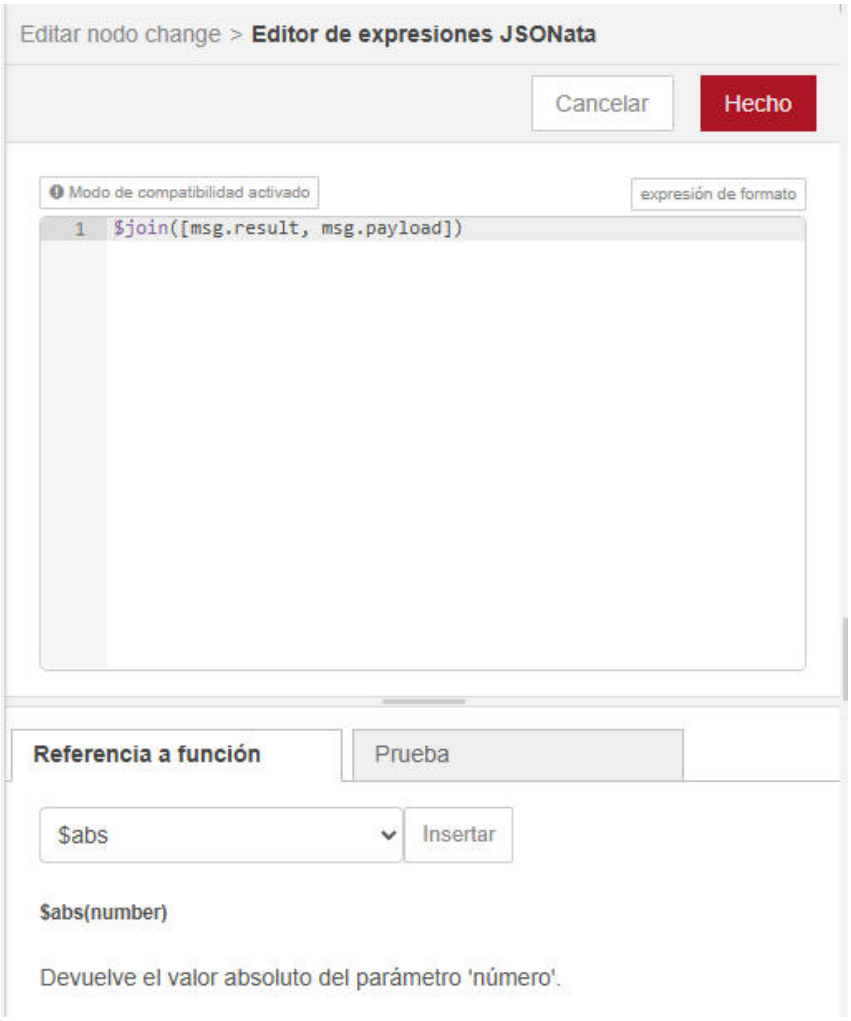


Figura 36. Configuración del JsonNata del Join.

Flujo de ejemplo: Array-loop

Este ejemplo hace uso del nodo: array-loop, el cual permite recorrer un array de varios elementos, separarlos, modificarlos si es necesario y entregar el resultado en la salida.

Específicamente, el ejemplo de la **¡Error! No se encuentra el origen de la referencia.**37 recorre un array formado por las letras desde la A hasta la E, separados por comas, para luego despachar 1 letra a la vez cada 1 segundo. Revisemos dicho flujo:

El flujo comienza con el nodo Inject, enviado por el payload el arrary [A,B,C,D,E], como se aprecia en la **¡Error! No se encuentra el origen de la referencia.** 38, el cual es dividido con la función Split, dentro del JsonNata del Change de la **¡Error! No se encuentra el origen de la referencia.**39, antes de ser enviado al nodo centroal: array-loop.

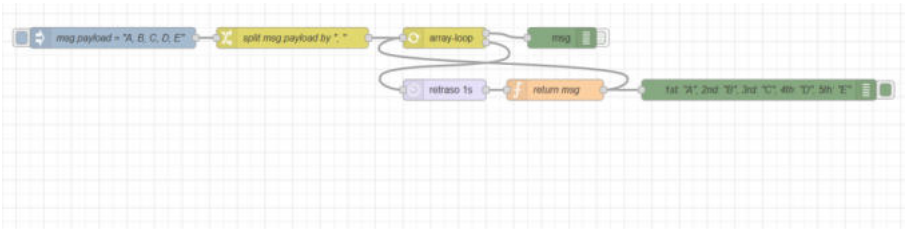


Figura 37. Flujo de ejemplo de suso del nodo: array-loop.

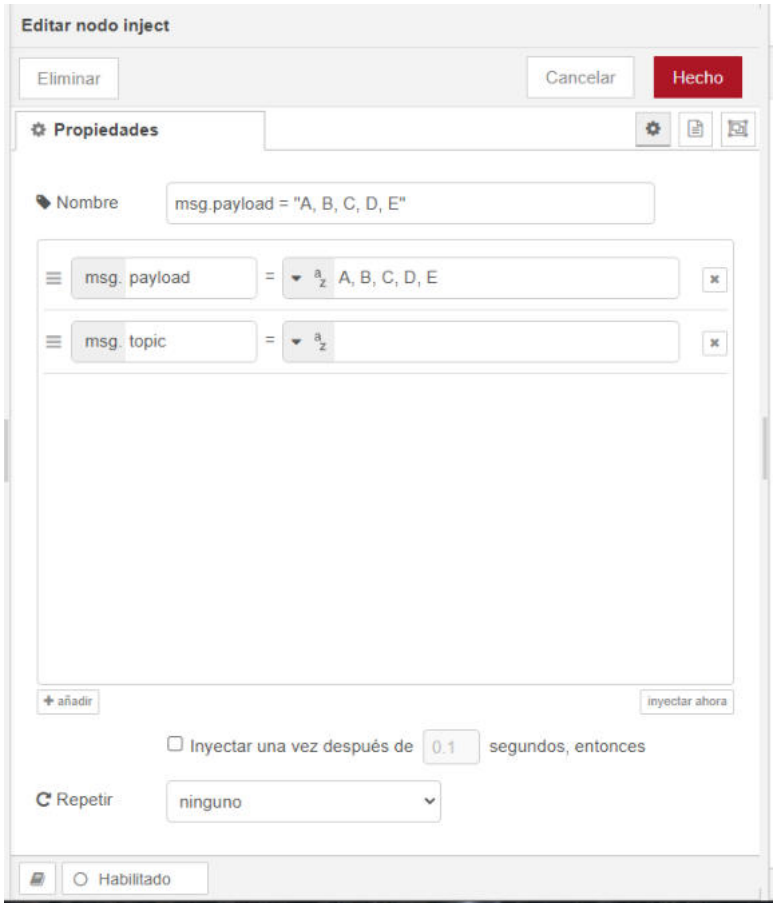


Figura 38. Array inicializado desde el nodo Inject.

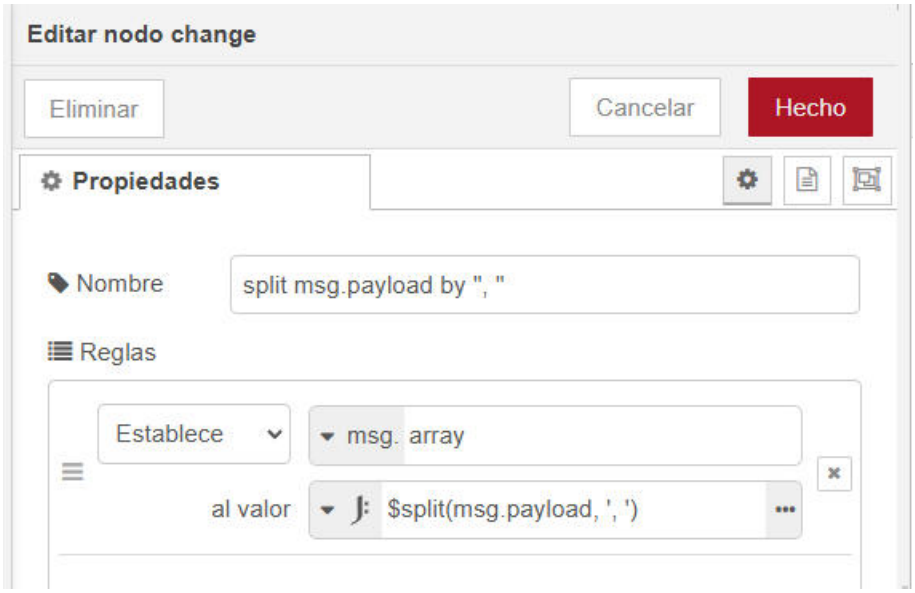


Figura 39. Configuración del nodo Change con la función Split dentro de un JSONata

En el nodo array-loop prácticamente solo debemos definir el nombre del array que vamos a recorrer en el lazo, como se aprecia en la **¡Error! No se encuentra el origen de la referencia.**40. El procesamiento dentro del lazo, en esta ocasión se encuentra dentro del nodo function y solamente devuelve el payload recibido, como se puede apreciar en la **¡Error! No se encuentra el origen de la referencia.**41.

.

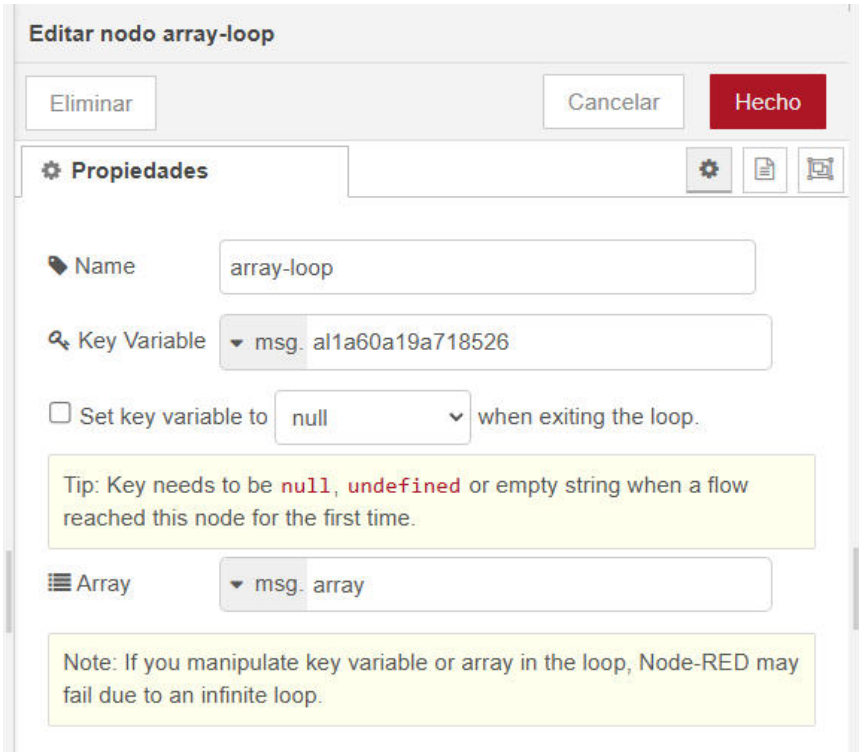


Figura 40. Definición del array dentro del loop.

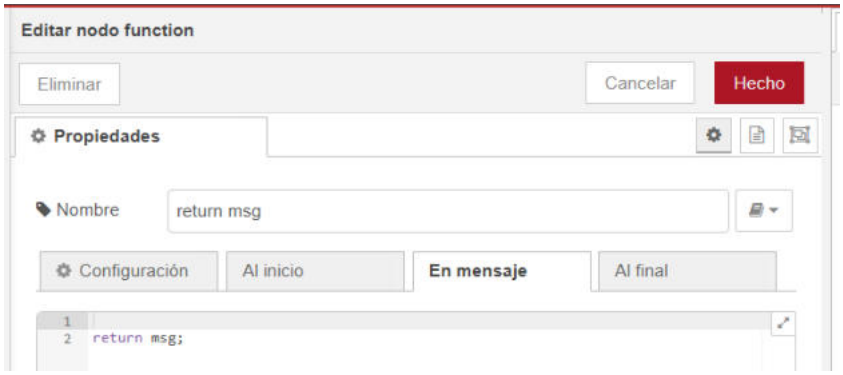


Figura 41. Nodo function dentro del array-loop permite hace alguna acción dentro de cada iteración.

Finalmente, en la **¡Error! No se encuentra el origen de la referencia.**42, podemos ver el resultado del lazo en cada iteración. Se observa todo el array y cada una de las letras enviadas en cada iteración.



Figura 42. Resultados de cada iteración del array-loop.

Flujo de ejemplo: While-loop

El nodo While-loop, permite crear iteraciones repetidas mientras se cumpla cierta condición, lo cual típico de un lazo While en la mayoría de los lenguajes de programación convencionales. En el ejemplo mostrado en la **¡Error! No se encuentra el origen de la referencia.**43, permite conocer si un número es divisible por 5. A continuación explicaremos dicho flujo:

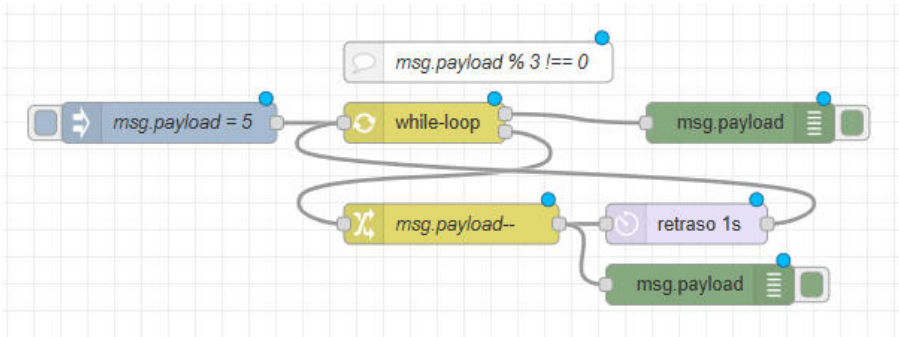


Figura 43. Flujo de ejemplo del Nodo: While-loop.

En cada iteración comprobamos si el número es divisible por 5, si se cumple entonces el ciclo se detiene, caso contrario, se decrementa en 1 el valor inicial y se repite el ciclo. Para ello, debemos inicializar el ciclo con un valor que nos permita algunas iteraciones a modo de ejemplo, el 9 (ver **¡Error! No se encuentra el origen de la referencia.**44), permitiría las iteraciones del 9, 8, 7, 6 y finalmente llegamos al 5 y se detendrá el ciclo, mostrando como resultado final dicho número en el nodo debug: Salida del While, demostrando que por ahí fue el flujo. Durante cada iteración veremos el resultado del flujo en el nodo debig: Cada iteración. Es importante, poner nombres a cada nodo en Node-red para poder identificarlos y saber el flujop real de los datos.

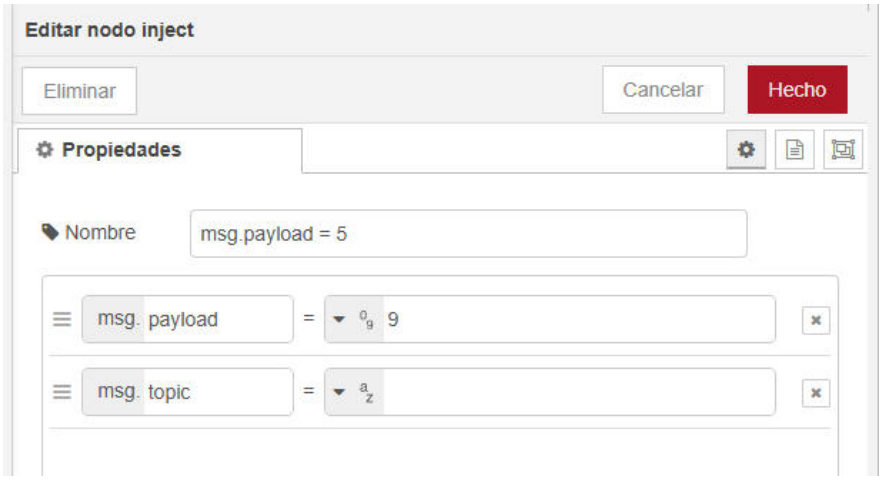


Figura 44. Se inicializa el lazo con el valor 9, para probar desde 9 hasta el 0 cuales números serán divisibles por 5.

La **¡Error! No se encuentra el origen de la referencia.**45 muestra la condición usada en el While, en este caso, se desea verificar que el número que viene en el payload sea divisible por 5.

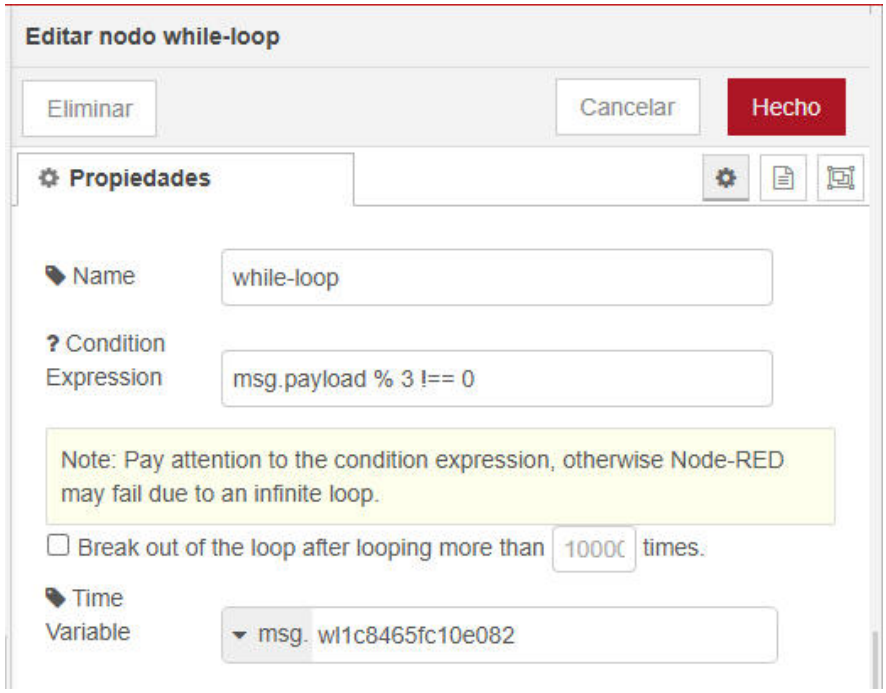


Figura 45. Nodo centrol While-loop, permite hacer iteraciones mientras se cumple una condición.

Si la condición no se cumple se pasa la siguiente iteración decrementando el número inicial, como se observa en la **¡Error! No se encuentra el origen de la referencia.**46.

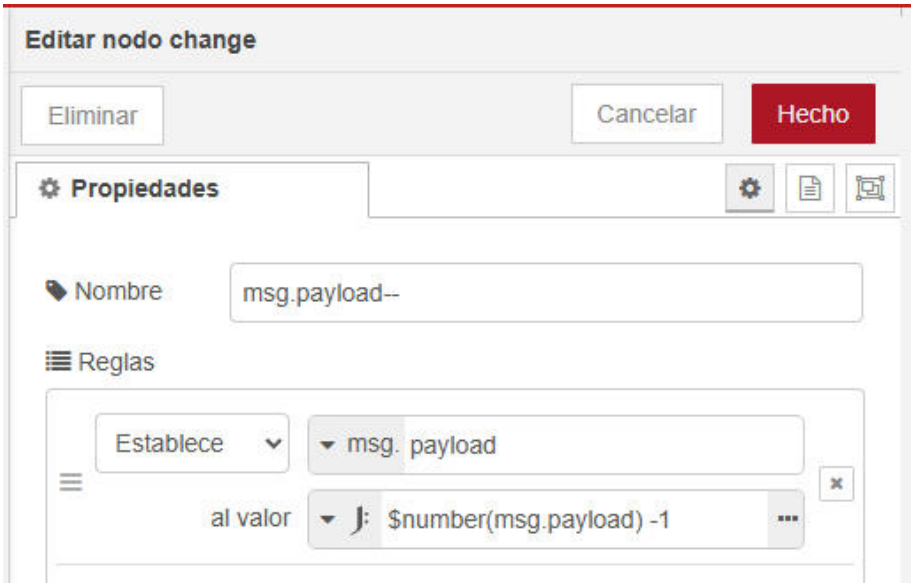


Figura 46. Configuración del node Change para decrementar un valor numérico.

Los resultados de cada iteración y salida del While se muestran en la **¡Error! No se encuentra el origen de la referencia.**47.

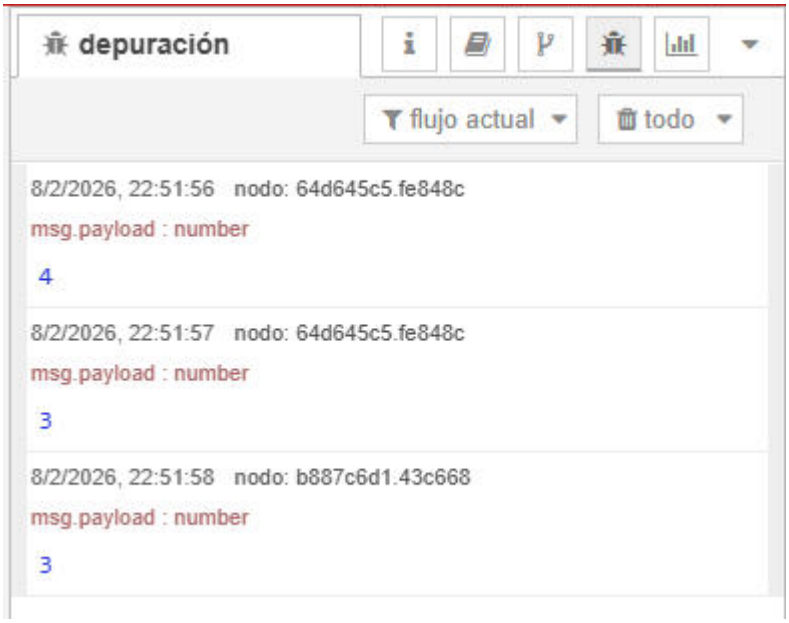


Figura 47. Resultado de cada iteración y salida del While, comenzando con 5.

☑ Solución 5: Uso del subflujo forEach (array iteration & complention)

Esta es una solución también de la comunidad, por tanto, debes probarla y ver si cumple con las necesidades de tu proyecto. La idea de esta solución fue encapsular en un subflujo (un pedazo de flujo) un código de javascript que permite hacer iteraciones. Lo interesante de esta solución, es que no tienes que instalar una librería, aunque una vez que copies dicho código, te aparecerá en lado izquierdo junto con el resto de nodos disponibles, clasificado como subflujos y se verá de la siguiente forma: 

.

Si damos doble click sobre el “iterate”, entonces aparece el subflujo de la **¡Error! No se encuentra el origen de la referencia.**48, y si damos doble click sobre el nodo function, entonces podemos ver el siguiente código:

```
//Node has 2 outputs - 1 for itteration and 1 for completion

var nextObj, out;

var itt = msg.iterationInfo;




//If the iterating has not yet begun set up the iteration metadata in the msg

if (typeof itt === 'undefined') {

   //Make sure payload is an array

   if( Object.prototype.toString.call(msg.payload) !== '[object Array]' ) {

      msg.payload = [msg.payload];

   }



   msg.iterationInfo = itt = {};
```

```
    itt.index = -1;

    itt.inArray = msg.payload;

    itt.outArray = [];



//Otherwise just push the input to the output array

} else {

    itt.outArray.push(msg.payload)

}

 //Goto next object

itt.index ++;

 //If there are stil objects left to iterate goto the next one in the original
array

if (itt.index < itt.inArray.length) {

    nextObj = msg;

    msg.payload = itt.inArray[itt.index];

 //otherwise pass the out array as the payload

} else {

    out = msg;
```

```
    msg.payload = itt.outArray;

    delete msg.iterationInfo;

}

return [nextObj, out];
```

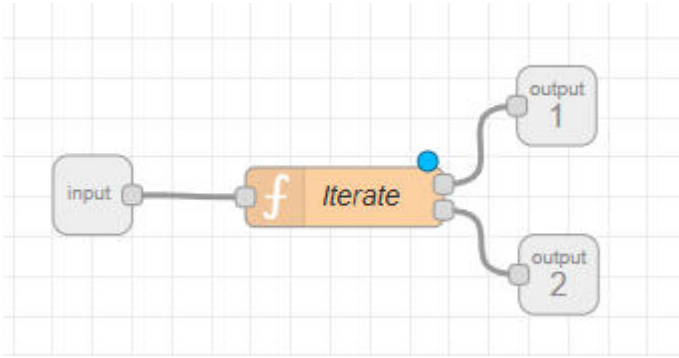


Figura 48. Subflujo Iterate que simula un lazo iterativo.

Para probar esta solución, creamos el flujo de la **¡Error! No se encuentra el origen de la referencia.**49, en la que hemos reutilizado parte de códigos de las soluciones anteriores, por ejemplo, el nodo inject crea un arreglo de temperaturas en grados celcius (¿Ya te acordaste de este ejemplo?), las cuales serán convertidas a grados Farhengeit dentro del nodo function, el cual a su vez forma parte del lazo iterativo. Los resultados de cada iteración se ven en el nod debug: Cada iteración y cuando se termine de recorrer todo el arreglo de temperaturas de entrada, entonces el flujo entrega un arreglo con las temperaturas convertidas, como se puede apreciar en la **¡Error! No se encuentra el origen de la referencia.**50.

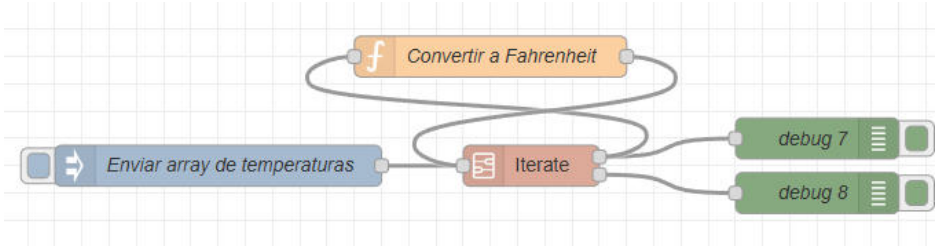


Figura 49. Flujo de prueba de la solución "Iterate" basada en un subflujo.



Figura 50. Resultados de cada iteración y resultado final del subflujo Iterate.

El nodo de ejemplo original se puede descargar de forEach (array iteration & completion) (flow) - Node-RED. Para mayor facilidad puedes copiarlo del siguiente recuadro.


```
[{"id":"b7765500.4889a8","type":"subflow","name":"Iterate","in":[{"x":220,"y":219,"wires":[{"id":"a1ad0c48.5e52f"}]}],"out":[{"x":454,"y":174,"wires":[{"id":"a1ad0c48.5e52f","port":0}]},{"x":455,"y":259,"wires":[{"id":"a1ad0c48.5e52f","port":1}]}]},{"id":"a1ad0c48.5e52f","type":"function","name":"Iterate","func":"//Node has 2 outputs - 1 for itteration and 1 for completion\nvar nextObj, out;\nvar itt = msg.iterationInfo;\n\n\n//If the iterating has not yet begun set up the iteration metadata in the msg\nif (typeof itt === 'undefined') {\n //Make sure payload is an array\n if( Object.prototype.toString.call(msg.payload) !== '[object Array]' ) {\n msg.payload = [msg.payload];\n }\n\n msg.iterationInfo = itt = {};\n itt.index = -1;\n itt.inArray = msg.payload;\n itt.outArray = [];\n\n//Otherwise just push the input to the output array\n} else {\n itt.outArray.push(msg.payload)\n}\n\n//Goto next object\nitt.index ++;\n\n//If there are stil objects left to iterate goto the next one in the original array\nif (itt.index < itt.inArray.length) {\n nextObj = msg;\n msg.payload = itt.inArray[itt.index];\n\n//otherwise pass the out array as the payload\n} else {\n out = msg;\n msg.payload = itt.outArray;\n delete msg.iterationInfo;\n}\n\nreturn [nextObj, out];","outputs":"2","noerr":0,"x":347,"y":220,"z":"b7765500.4889a8","wires":[[],[]]},{"id":"d031fa3c.2fce08","type":"subflow:b7765500.4889a8","name":"Iterate","x":568,"y":322,"z":"ce44b4c2.31bb48","wires":[["a5b7e659.5a4818"],["40b4c609.bf4b38"]]}]
```

2.3.1 Timeout, delays e intervals

Delay

- Imagen**:** 

- Nodo delay: introduce una pausa (en ms, segundos, minutos) antes de pasar el mensaje al siguiente nodo.
- Nodo inject: puede programarse para enviar mensajes periódicamente o a una hora específica.

Ejemplo: enviar un mensaje cada 10 minutos, o retardar el apagado de una luz por 5 segundos.

⚠️ **PRECAUCIÓN – El nodo** delay **puede bloquear flujos**

Si usas el nodo delay en modo "rate limit", podrías estar impidiendo que nuevos mensajes pasen durante cierto intervalo. Úsalo con cuidado cuando trabajes con eventos críticos.

Timeout

Para programaciones más avanzadas, el nodo function permite usar temporizadores de JavaScript:

```
javascript

CopiarEditar

setTimeout(function() {
```

```
    node.send({ payload: "fin de espera" });

}, 5000);
```

⚠️ Importante: este tipo de uso requiere cuidado, ya que un mal manejo puede generar múltiples ejecuciones en paralelo no deseadas.

Estas estructuras de control permiten construir flujos que reaccionan inteligentemente al entorno y los datos, lo cual es esencial para cualquier aplicación IoT automatizada o interactiva. En el siguiente apartado se verá cómo combinar estas estructuras y organizarlas eficientemente dentro de flujos visuales mantenibles.

2.4 Intervalos en Node-RED (Repeticiones temporizadas)

Un **intervalo** es una acción programada que se repite de forma periódica, por ejemplo: enviar un mensaje cada 10 segundos, consultar un sensor cada minuto, o generar una notificación diaria. Node-RED permite lograr esto de forma muy sencilla mediante el nodo inject, y también con lógica avanzada dentro de un function.

☑ **1. Intervalos simples con nodo** inject

El nodo inject permite **enviar mensajes periódicamente** sin necesidad de código.

Pasos:

1. Arrastrar un nodo inject al flujo.

2. Hacer doble clic para configurarlo.

3. En “Payload”, definir el valor a enviar (por ejemplo, timestamp o un texto).

4. En la sección **“Repeat”**, elegir la opción **“interval”** y establecer el tiempo (en segundos o minutos).

5. Conectar a un nodo debug o cualquier otro nodo.

Ejemplo de uso: la **¡Error! No se encuentra el origen de la referencia.**51, muestra como enviar la hora actual cada 10 segundos para registrar eventos.

🔧 **2. Intervalos personalizados con** setInterval **en un nodo** function

Si se necesita más control (por ejemplo, iniciar/detener un intervalo bajo ciertas condiciones), se puede usar JavaScript en un nodo function.

Ejemplo básico:

```
javascript

CopiarEditar

let id = setInterval(function() {

    node.send({ payload: "Ejecutando cada 5s" });

}, 5000);
```

// Guardar el ID del intervalo en el contexto para cancelarlo luego

```
        context.set("intervalId", id);
```

// Importante: el nodo no envía inmediatamente, sino que el `setInterval` lo hará después

return null;

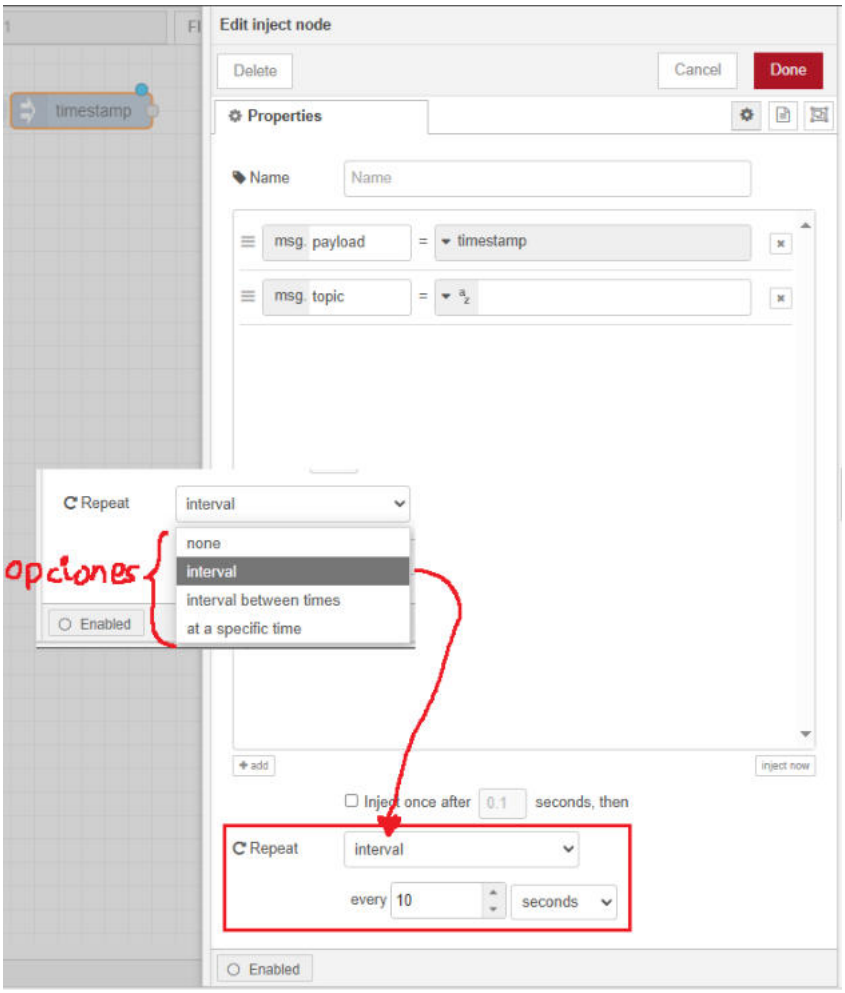


Figura 51. Opciones de repetición disponibles en el nodo Inject.

Para detener el intervalo:

```
javascript

CopiarEditar

let id = context.get("intervalId");

clearInterval(id);

context.set("intervalId", null);

return { payload: "Intervalo detenido" };
```

⚠️ **Precaución:**

- Usar setInterval directamente puede causar múltiples ejecuciones si no se controla correctamente.
- Es preferible usar esta técnica solo si necesitas condiciones dinámicas de inicio/parada.

🗓️ **3. Intervalos programados con** inject **+ configuración de hora**

También puedes usar el nodo inject para:

- Ejecutar algo a una **hora específica del día.**
- Repetir cada día, semana o a horas determinadas.

Ejemplo: enviar un mensaje todos los días a las 6:00 AM, ver **¡Error! No se encuentra el origen de la referencia.**52.

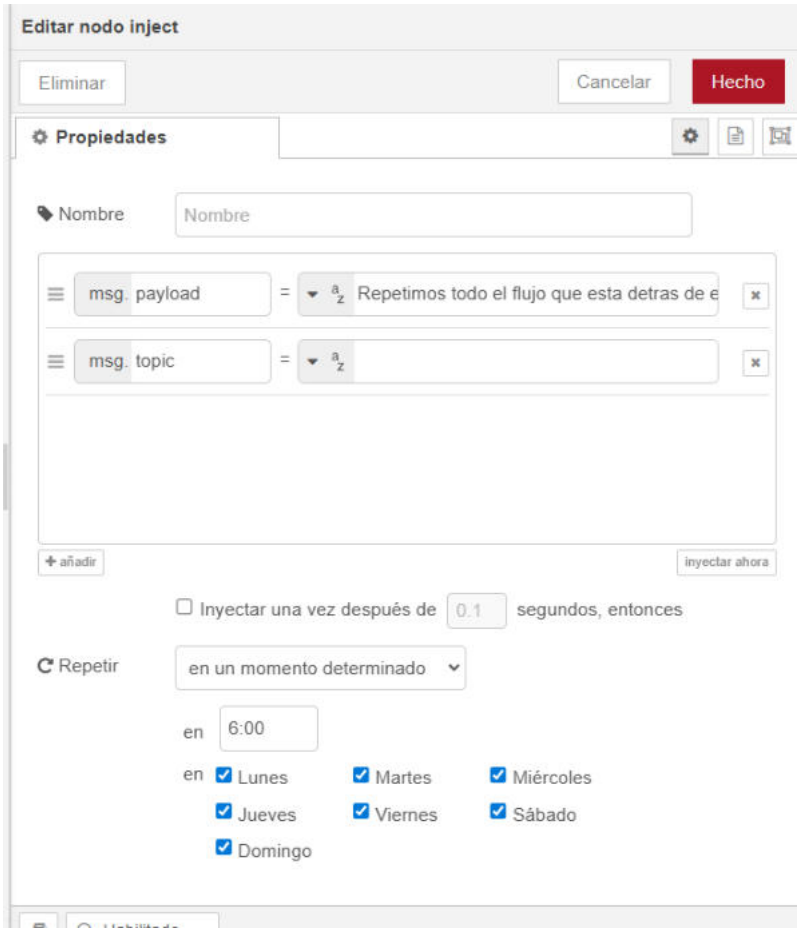


Figura 52. Opción disponible dentro del nodo inject para ejecutar dicho nodo a un tiempo específico. .

🧠 Buenas prácticas

- Siempre usa inject si el intervalo es fijo y simple.
- Si necesitas que el intervalo **dependa de un evento**, considera usar setInterval con contexto.
- No usar múltiples setInterval sin control, ya que pueden sobrecargar el sistema o producir errores lógicos.

2.5 Manipulación de datos

En el ámbito de IoT, típicamente, los datos deben ser **interpretados, modificados y transformados** para que puedan ser usados correctamente en la toma de decisiones, el almacenamiento o la visualización. Node-RED proporciona nodos específicos y técnicas visuales para realizar este procesamiento sin necesidad de codificar grandes estructuras lógicas. Este epígrafe aborda las técnicas fundamentales de manipulación de datos, incluyendo **escalado, fusión, separación y transformación de estructuras.**

En Node-RED, los datos se encapsulan en objetos msg (mensajes), cuyo campo o propiedad principal suele ser msg.payload. Sin embargo, un mensaje puede tener muchos otros campos (como msg.topic, msg.timestamp, msg.deviceID, etc.). Aprender a modificar, combinar o estructurar estos campos correctamente es esencial para un desarrollo IoT eficiente y mantenible.

2.5.1 Escalado de datos (range, function)

El **escalado** es útil cuando se requiere adaptar un valor numérico de un rango a otro. Por ejemplo, convertir una lectura de 0–1023 (ADC) a un porcentaje (0–100%), ver **¡Error! No se encuentra el origen de la referencia.**53.

Opción A: con nodo range **(si está instalado)**

- Imagen: 

- Ingresar el rango de entrada y de salida.
- El nodo realiza la interpolación automáticamente.

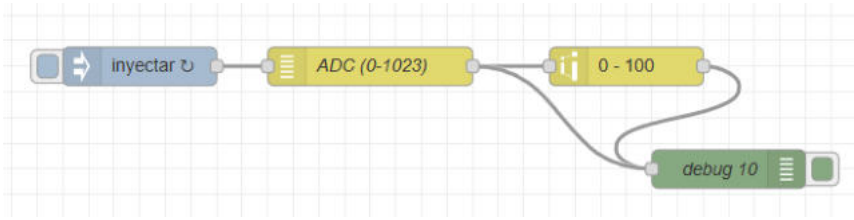


Figura 53. Ejemplo de uso del nodo range.

En la **¡Error! No se encuentra el origen de la referencia.**54, del lado izquierdo se muestra cómo se puede configurar el nodo range, según el ejemplo o caso de uso, de medición de un sensor de temperatura con una precisión de 10 bits. Es decir, la salida de un convertidor análogo-digital (ADC) de 10 bits, brinda 1023 posibles salidas digitales, por tanto la humedad es medida con esa precisión. Luego esos valores deben ser convertidos a unidades de ingeniería, por ejemplo, ene este caso de Humedad, la variable tiene un rango de 0 a 100% de humedad.

Luego, en el lado derecho de dicha figura, se muestran los resultados de varias pruebas de generar un dato aleatorio del convertidor ADC y como este es correctamente escalado con las unidades de ingeniería definidas.

Nota: Recuerde que el nodo Random, debe ser instalado desde el manage pallete.

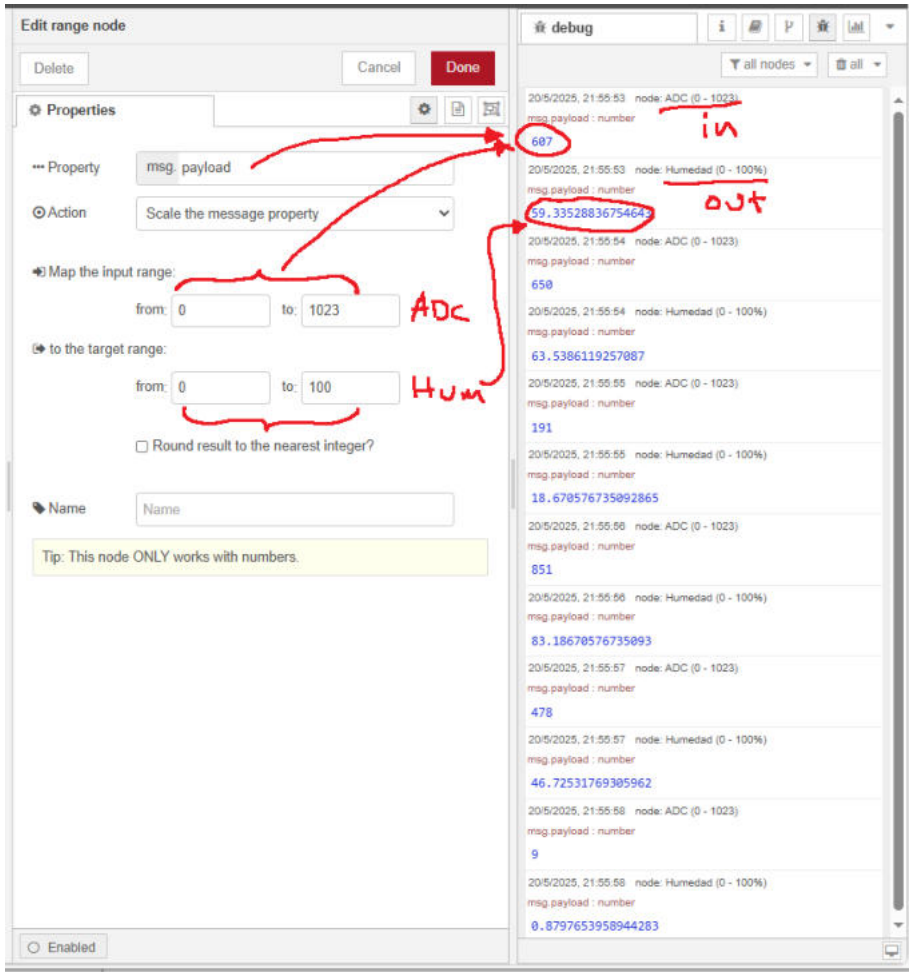


Figura 54. Configuración y Salida del nodo range.

Opción B: con nodo function**:**

```
javascript

CopiarEditar

let x = msg.payload; // entrada 0–1023

msg.payload = (x - 0) * (100 - 0) / (1023 - 0);
```

```
return msg;
```

2.5.2 Fusión de datos (merging)

A veces se requiere **combinar datos provenientes de diferentes fuentes** (por ejemplo, sensor de temperatura + humedad).

Métodos comunes:

- Usar join para unir varios mensajes en un solo objeto JSON.
- Usar function para construir estructuras más personalizadas.

Ejemplo con join**:**

1. Imagen: 

2. Conectar múltiples nodos inject o sensores a un join.
3. Configurar como *manual* y unir por número de mensajes o campos topic.

Resultado:

```
json

CopiarEditar

{

 "temperatura": 22.5,
```

```
  "humedad": 56

}
```

En la **¡Error! No se encuentra el origen de la referencia.**55 podemos observar un ejemplo clásico del uso del nodo Join, donde hemos configurado dicho nodo de forma manual, para que acepte 2 conjuntos de key/value, provenientes de dos sensores, el de humedad y temperatura. Como se aprecia, el msg.topic ayuda para definir el "key" y el payload trae el "value"

Nota: ¿Recuerdas el uso del nodo Join, en ejemplos anteriores? Búscalo …

⚠ PRECAUCIÓN – El nodo inject puede programarse mal

Asegúrate de configurar correctamente la frecuencia de inyección si usas modo repetitivo. Un intervalo muy corto puede saturar el flujo o sobrecargar el CPU en Raspberry Pi.

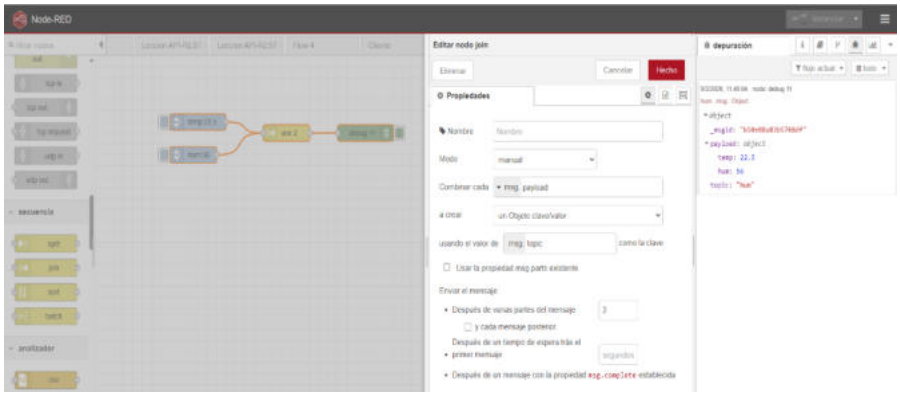


Figura 55. Ejemplo de uso del nodo Join para crear un Json con los valores de 2 sensores.

2.5.3 Separación de datos (splitting)

Cuando un mensaje contiene una lista o array, puede ser necesario **dividirlo en varios mensajes individuales.**

Nodo split:

- Imagen: 

- Divide un array de msg.payload en múltiples mensajes secuenciales.
- Útil para enviar datos de uno en uno a través del flujo.

Ejemplo:

Entrada:

```
json

CopiarEditar

{ "payload": [10, 20, 30] }
```

Salida: tres mensajes separados con payloads 10, 20 y 30. En la **¡Error! No se encuentra el origen de la referencia.**56, del lado izquierdo tenemos un flujo de ejemplo, que “inyecta” un array de temperaturas, que el nodo Split las separa y las saca por separado en el payload, como se aprecia en el lado derecho de lso resultados.

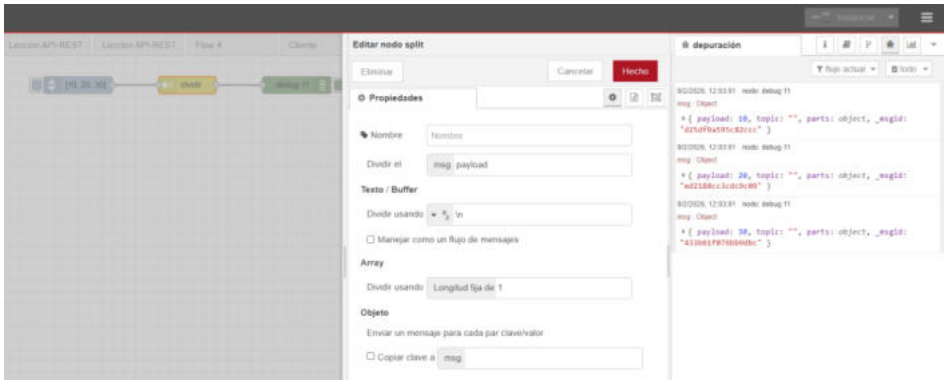


Figura 56. Ejemplo del uso del nodo Split y su resultado.

2.5.4 Reestructuración o transformación de mensajes (change, function)

Cambiar el formato o contenido de los mensajes es una tarea recurrente.

Nodo change**:** 

- Permite asignar, copiar, mover o eliminar campos del mensaje de forma visual, como se aprecia en la **¡Error! No se encuentra el origen de la referencia.**57.

- Este nodo, ha sido y será usado extensamente en los ejemplos de este libro. Por tanto, no abundaremos en este capítulo mucho al respecto y dejaremos que la magia de node-red aparezca en cada ejemplo, para sorprendernos con la facilidad de solución a problemas complejos en otros lenguajes de programación.

- En próximos epígrafes lo usaremos para extraer información de un json en diferentes protocolos de comunicación de una manera muy fácil.

Nodo function**:** 

Ideal para estructuras más complejas:

```
javascript

CopiarEditar

msg.payload = {

  valor: msg.payload,

  unidad: "°C",

  timestamp: Date.now()

};

return msg;
```

También este nodo ha sido usado con mucha frecuencia en este libro, por tanto, no necesita presentación. Lo veremos en acción siempre, búscalo en cada epígrafe …

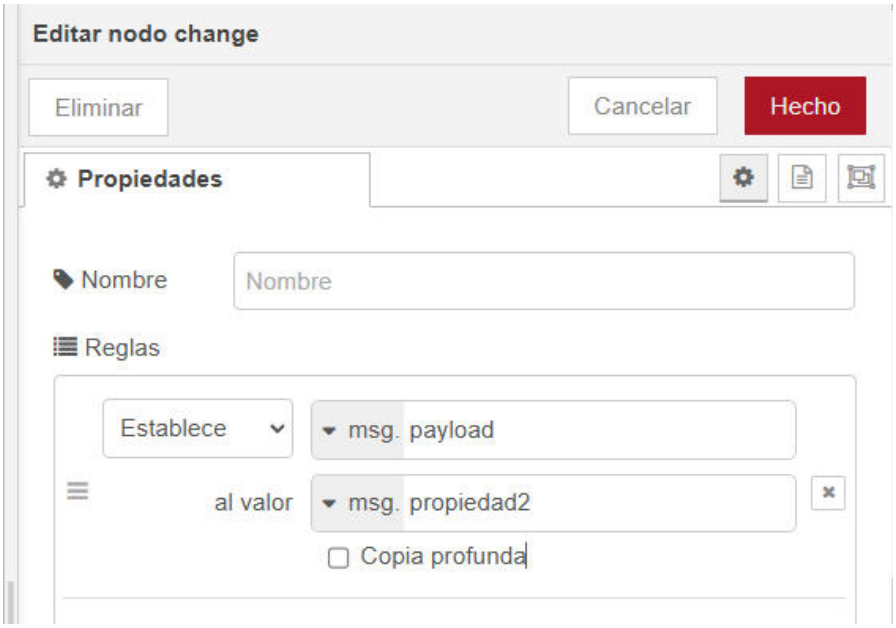


Figura 57. Configuraciones del nodo Change.

2.5.5 Conversión de tipos y formatos

En flujos reales es común convertir:

- De **string a número** (Number(msg.payload)).
- De **JSON a objeto** (usando JSON.parse si llega como texto).
- De **timestamp a formato legible** (con librerías como moment.js o funciones manuales).

🔁 Ejemplos comunes de conversión de tipos y formatos

☑ 1. De texto (string) a número

Problema:

Un sensor MQTT envía datos como texto: "25.3"

Pero se requiere usarlo como número para comparaciones o cálculos.

Solución: nodo function

```
javascript

CopiarEditar

msg.payload = Number(msg.payload);

return msg;
```

Ahora msg.payload será un número: 25.3

Una forma “elegante” de usar código de javascript sin el nodo Function, es usar expresiones JSONata dentro de un nodo change, como se aprecia en la **¡Error! No se encuentra el origen de la referencia.**58.

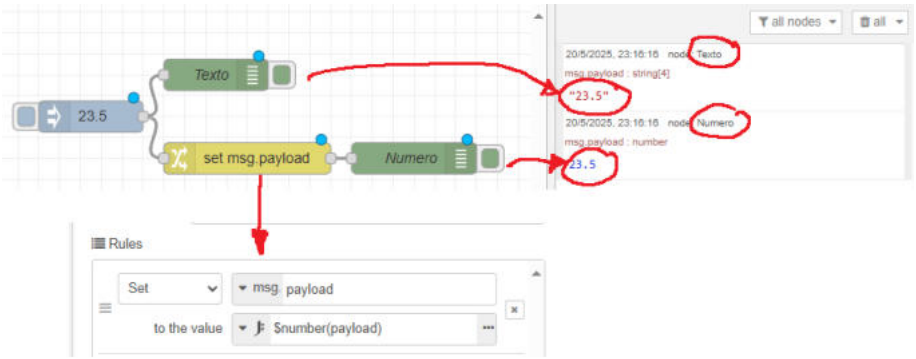


Figura 58. Ejemplo de uso del nodo change con JSONata para hacer una conversión de texto a número.

☑ 2. De número a string

Caso típico: quieres concatenar datos para formar una cadena legible.

```
javascript

CopiarEditar

msg.payload = msg.payload.toString();

return msg;
```

☑ 3. De texto JSON a objeto

Entrada:

```
json

CopiarEditar

{ "payload": "{\"temp\":25,\"hum\":60}" }
```

Función para convertir:

```
javascript

CopiarEditar

msg.payload = JSON.parse(msg.payload);

return msg;
```

Salida como objeto usable:

```
json

CopiarEditar

{ "temp": 25, "hum": 60 }
```

☑ 4. De objeto a texto JSON

Útil para enviar datos como string a otros servicios (HTTP, MQTT, archivos).

```
javascript

CopiarEditar

msg.payload = JSON.stringify(msg.payload);

return msg;
```

Existe además un nodo llamado Json, que permite hacer dichas conversiones an las dos direccions, es decir de Objeto a Json y viceversa, como se puede apreciar en la **¡Error! No se encuentra el origen de la referencia.**.

Imagen: 

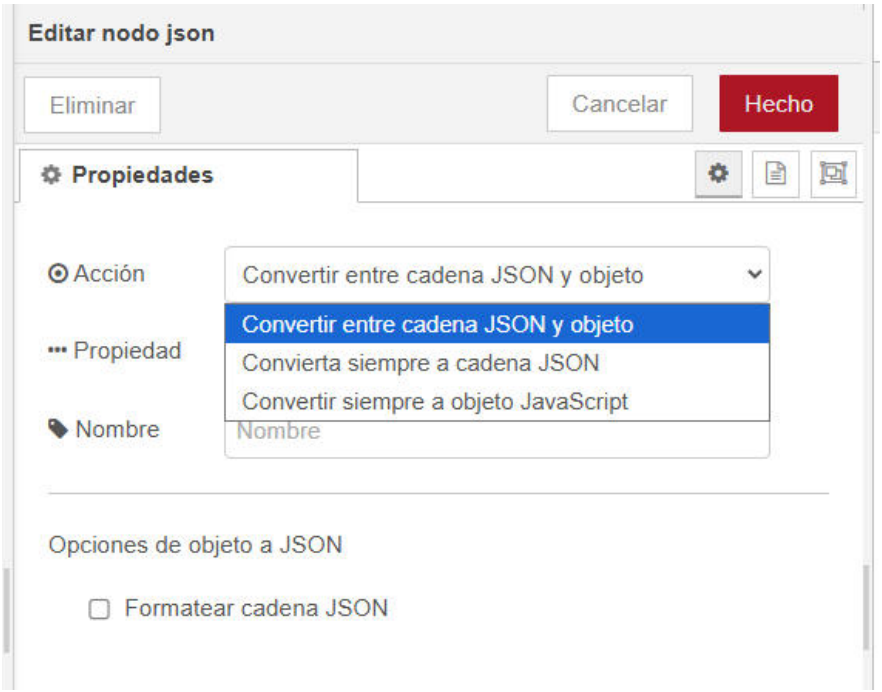


Figura 59. Opciones de conversión bidireccional del nodo JSON.

☑ 5. De timestamp a fecha legible

Entrada: un valor numérico UNIX timestamp:

```
json

CopiarEditar

{ "payload": 1684161345789 }
```

Función:

```
javascript

CopiarEditar
```

```
let fecha = new Date(msg.payload);

msg.payload = fecha.toLocaleString();  // o toISOString(),
toDateString(), etc.

return msg;
```

Salida: "14/5/2025, 14:35:45" (dependiendo del sistema y zona horaria)

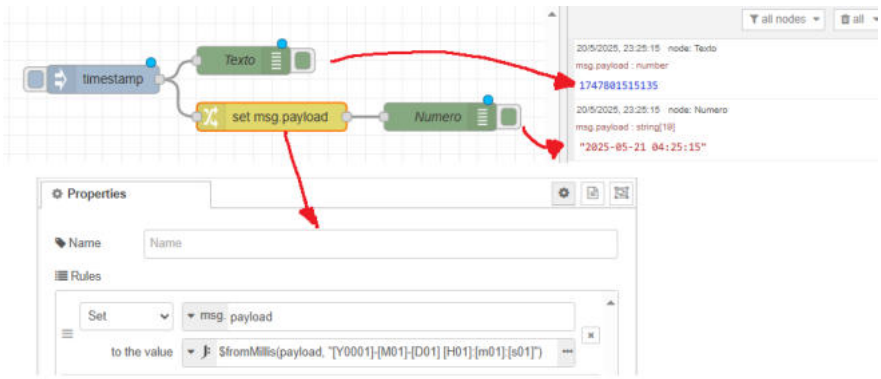


Figura 60. Ejemplo de uso del nodo Change para obtener de forma legible un timestamp.

La **¡Error! No se encuentra el origen de la referencia.**60 muestra cómo obtener de forma visual una conversión tipo timestamp.

☑ 6. De string a booleano

Entrada: "true" o "false" como texto.

```
javascript

CopiarEditar

msg.payload = (msg.payload === "true");
```

```
return msg;
```

Resultado: true o false como tipo booleano.

☑ 7. De grados Celsius a Fahrenheit

Conversión de unidades dentro de un flujo

```
javascript

CopiarEditar

let celsius = msg.payload;

msg.payload = (celsius * 9/5) + 32;

return msg;
```

Entrada: 25 → Salida: 77

🔍 Consejos adicionales

- Node-RED trata muchos valores automáticamente según el contexto, pero **siempre es mejor convertir explícitamente** para evitar errores en flujos complejos.
- Puedes usar nodos change para conversiones simples, pero para lógica más flexible se recomienda el nodo function.

☑ Conclusión

La manipulación eficiente de datos en Node-RED es clave para el desarrollo de soluciones IoT dinámicas, robustas y adaptadas a distintas fuentes. Escalar valores, combinar datos, dividir arrays o transformar estructuras son operaciones fundamentales que deben ser dominadas antes de avanzar hacia integraciones con bases de datos, APIs o dashboards.

2.6 Buenas prácticas de diseño de flujos

A medida que los flujos en Node-RED crecen en complejidad, se vuelve fundamental aplicar **buenas prácticas de diseño visual, estructural y funcional** que permitan mantener el control del sistema, facilitar el trabajo colaborativo y asegurar la reutilización de componentes. Aunque Node-RED permite prototipar rápidamente, un enfoque descuidado puede derivar en flujos desorganizados, difíciles de depurar o escalar.

Este apartado presenta principios clave y recomendaciones prácticas para diseñar flujos eficientes y sostenibles, abordando aspectos visuales, lógicos y organizativos.

2.6.1 Usa comentarios y etiquetas descriptivas

- Añade nodos comment 

para documentar bloques de lógica o decisiones.

- Utiliza nombres significativos en nodos function, switch, template, etc.

- Evita etiquetas genéricas como “function 1” o “debug 2”; en su lugar, usa nombres como “Validar humedad” o “Enviar alerta por correo”. Vea los ejemplos de la **¡Error! No se encuentra el origen de la referencia.**61.



Figura 61. Ejemplo del uso del nodo Comments para hacer más descriptivos los flujos.

2.6.2 Agrupa por bloques funcionales

Organiza el lienzo en **secciones visuales** que representen tareas específicas, ver **¡Error! No se encuentra el origen de la referencia.**62:

- Entrada de datos
- Procesamiento
- Decisiones

- Salidas o acciones

Puedes usar nodos comment o nodos status como separadores visuales. También puedes usar subflujos para encapsular lógicas repetitivas.

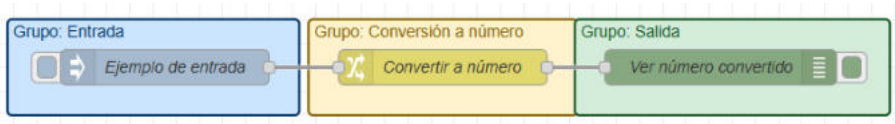


Figura 62. Ejemplo de uso de bloques y colores para hacer más descriptivo los flujos.

2.6.3 Usa subflujos para modularizar

Si repites estructuras similares en diferentes partes del flujo, crea un **subflujo reutilizable**.

Esto:

- Reduce el desorden visual
- Aumenta la mantenibilidad
- Facilita la edición centralizada

Ejemplo: un subflujo llamado "Escalar valor ADC" puede ser usado en múltiples sensores.

🛠️ ¿CÓMO SE CONFIGURA? – Crear un subflujo reutilizable

1. Selecciona un conjunto de nodos.
2. Clic derecho → *Create subflow*.
3. Aparecerá como una nueva pestaña.

4. Desde el panel izquierdo, podrás arrastrarlo como un nuevo "nodo personalizado" en cualquier otro flujo.

2.6.4 Limita el uso de nodos function innecesarios

Aunque los nodos function ofrecen flexibilidad con JavaScript, abusa de ellos puede hacer que el flujo pierda legibilidad. Siempre que sea posible:

- Usa nodos change, switch, template o join.
- Reserva function para lógica que no pueda expresarse con nodos visuales.

2.6.5 Usa colores y nombres personalizados

En la pestaña de configuración de cada nodo, puedes:

- Cambiar el **color** del nodo para distinguir funciones visualmente.
- Asignar un **nombre corto y claro** que se muestre dentro del nodo.

Esto ayuda a entender flujos complejos sin necesidad de abrir cada nodo.

2.6.6 Controla la cantidad de conexiones (wires)

Evita conexiones cruzadas o múltiples líneas desde un solo nodo que se crucen visualmente. En su lugar:

- Usa nodos link in y link out para conectar lógicamente flujos sin desorden físico.

- Divide procesos en varias pestañas si es necesario.

2.6.7 Despliega frecuentemente con control

Evita despliegues innecesarios. Usa la opción:

- **"Modified Flows"** para actualizar solo lo que cambió.
- **"Full"** solo si es necesario.
- En la **¡Error! No se encuentra el origen de la referencia.**63, se observan todas las opciones posibles a la hora de hacer un deploy.

Esto evita reinicios innecesarios y mantiene el sistema más estable.

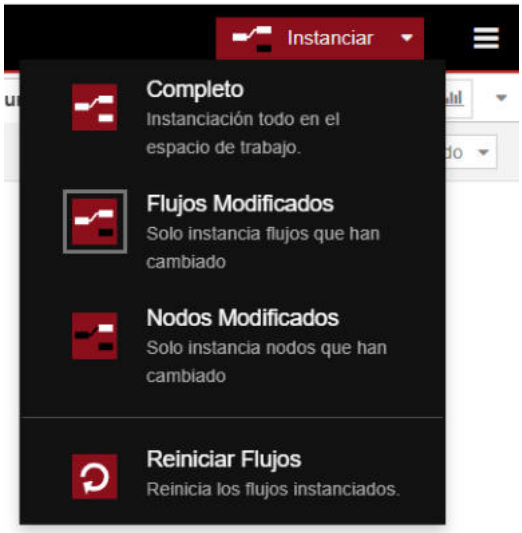


Figura 63. Opciones para hacer un deploy (instanciar) .

🧠 Conclusión

Aplicar buenas prácticas en el diseño de flujos no solo mejora la presentación visual del proyecto, sino que también tiene un impacto directo en la **facilidad de mantenimiento, depuración, escalabilidad y colaboración.** Un flujo bien organizado refleja profesionalismo y facilita el crecimiento del sistema sin perder el control sobre su estructura interna.

Elemento de cierre

Resumen del capítulo:

En este capítulo aprendiste a construir flujos lógicos en Node-RED utilizando nodos visuales para implementar decisiones, repeticiones y manipulaciones de datos. Comprendiste cómo diseñar flujos limpios y estructurados mediante buenas prácticas y técnicas de modularización como los subflujos.

Preguntas de revisión

❓ **Pregunta 1:** ¿Qué nodo te permite evaluar condiciones y tomar decisiones basadas en msg.payload?

✍️ a) switch

✍️ b) template

✍️ c) inject

✍️ d) ui_button

☑️ **Respuesta correcta:** a) switch

❓ **Pregunta 2:** ¿Qué nodo debes usar para ejecutar una acción cada 10 segundos?

✍️ a) delay

✍️ b) inject (en modo repetición)

✍️ c) debug

✍️ d) status

☑️ **Respuesta correcta:** b) inject (modo repetición)

❓ **Pregunta 3:** ¿Cuál de estos nodos permite modificar msg.payload sin código?

✍️ a) function

✍️ b) change

✍️ c) debug

✍️ d) json

☑️ **Respuesta correcta:** b) change

❓ **Pregunta 4:** ¿Qué estructura permite encapsular un grupo de nodos como unidad reutilizable?

✍️ a) Subflujo

✍️ b) JSON

✍️ c) Tab

✍️ d) Payload

☑️ **Respuesta correcta:** a) Subflujo

Clave de respuestas

Pregunta	Respuesta correcta
1	a) switch
2	b) inject
3	b) change
4	a) Subflujo

CAPÍTULO III

3 FLUJOS AVANZADOS

🎯 Objetivos de aprendizaje:

- Comprender y usar variables de contexto (flow, global) para almacenamiento temporal o persistente.
- Trabajar con estructuras de datos complejas y mensajes anidados.
- Utilizar proyectos versionables con Git.
- Programar lógica personalizada con JavaScript y librerías externas.
- Crear nodos personalizados reutilizables.

🛠️ Competencias:

- Integra flujos complejos que gestionan múltiples niveles de información.
- Controla versiones de tus proyectos mediante sistemas como Git.
- Desarrolla nodos personalizados y módulos extendidos en Node.js.
- Aplica mecanismos de seguridad avanzada en el diseño de flujos.

❓ Preguntas de enfoque:

- ¿Cuándo conviene usar variables flow o global?
- ¿Cómo puedo estructurar mensajes complejos dentro de Node-RED?
- ¿Qué ventajas ofrece usar proyectos y control de versiones?
- ¿Cómo se construye un nodo personalizado para Node-RED?

Este capítulo está orientado a implementar conceptos avanzados en el desarrollo de flujos más sofisticados, dinámicos y modulares dentro de Node-RED. Se abordarán técnicas de programación personalizada con JavaScript, manipulación estructurada de variables de contexto como flow yglobal, reutilización de códigos mediante subflujos y gestión de proyectos complejos con control de versiones. Además, de ampliar las capacidades del sistema mediante el uso de librerías externas, manejo de errores y en casos avanzados, la creación de nodos personalizados.

Al finalizar este capítulo, el lector será capaz de implementar soluciones que requieran persistencia de datos, encapsulamiento lógico, automatización avanzada y control granular del comportamiento de los flujos. Podrá adquirir las habilidades necesarias para depurar errores de forma estructurada, optimizar el rendimiento del sistema, y crear entornos de desarrollo organizados, seguros y adaptados a las necesidades específicas de cada proyecto. Este nivel de dominio permitirá construir soluciones IoT escalables, mantenibles y listas para su despliegue en entornos reales.

3.1 Gestión de datos y contexto

En flujos simples de Node-RED, los datos se transportan entre nodos dentro del objeto msg, el cual contiene propiedades como msg.payload, msg.topic y otras definidas por el usuario. Sin embargo, cuando se requiere almacenar información entre ejecuciones, compartir datos entre flujos o conservar variables durante largos periodos de tiempo, se hace necesario utilizar el sistema de contexto de Node-RED.

3.1.1 Variables locales, de flujo y globales

Node-RED ofrece tres niveles de contexto para guardar y acceder a datos temporales o persistentes:

- Contexto local de nodo (context): solo accesible desde el nodo actual.
- Contexto de flujo (flow): accesible desde todos los nodos dentro del mismo flujo (misma pestaña).
- Contexto global (global): accesible desde cualquier nodo en cualquier flujo.

Estas variables se almacenan en memoria por defecto, pero también pueden configurarse para almacenamiento persistente en disco, permitiendo retener datos incluso después de reiniciar Node-RED. Este mecanismo es fundamental en aplicaciones que requieren recordar estados, acumular valores, manejar sesiones o comunicar subprocesos.

- **Uso básico de variables de contexto en un nodo** function

Guardar una variable:

```
javascript

CopiarEditar

context.set("contador", 5);

flow.set("estado", "activo");

global.set("temperatura", 22.3);
```

Leer una variable:

```
javascript

CopiarEditar

let contador = context.get("contador");

let estado = flow.get("estado");

let temp = global.get("temperatura");
```

- **Persistencia del contexto**

Para habilitar el almacenamiento persistente:

1. Editar el archivo settings.js.
2. Descomentar y configurar:

```
CopiarEditar
```

```
contextStorage: {
    default: "file",
    memoryOnly: { module: "memory" },
    file: { module: "localfilesystem" }
}
```

Esto permite que todas las variables almacenadas en el contexto "file" se conserven entre reinicios. Es útil para contadores, configuraciones del usuario o estados de dispositivos.

🔸 **Visualización del contexto**

Desde la interfaz de Node-RED, se puede **inspeccionar el contenido de las variables de contexto:**

- Abrir el panel lateral derecho.
- Ir a la pestaña **Context Data**.
- Visualizar y editar las variables en tiempo real.

☑ Aplicaciones prácticas

- Recordar el **estado anterior de un sensor** o actuador.
- Implementar **temporizadores, contadores o acumuladores**.

- Almacenar datos entre ciclos de flujo sin necesidad de nodos externos.
- Coordinar flujos paralelos mediante variables compartidas.

Dominar el sistema de contexto es una habilidad esencial para construir flujos avanzados en Node-RED. Permite diseñar soluciones más eficientes, limpias y estructuradas, minimizando la dependencia de bases de datos externas cuando no son necesarias.

Para ejemplificar el uso de estas variables, podemos apoyarnos en los flujos mostrados en las siguientes figuras, donde el primer flujo lo usaremos para escribir en memoria. En el extremo derecho, podemos apreciar el estado de la información de contexto en Node-red, como se puede apreciar en la **¡Error! No se encuentra el origen de la referencia.**64.

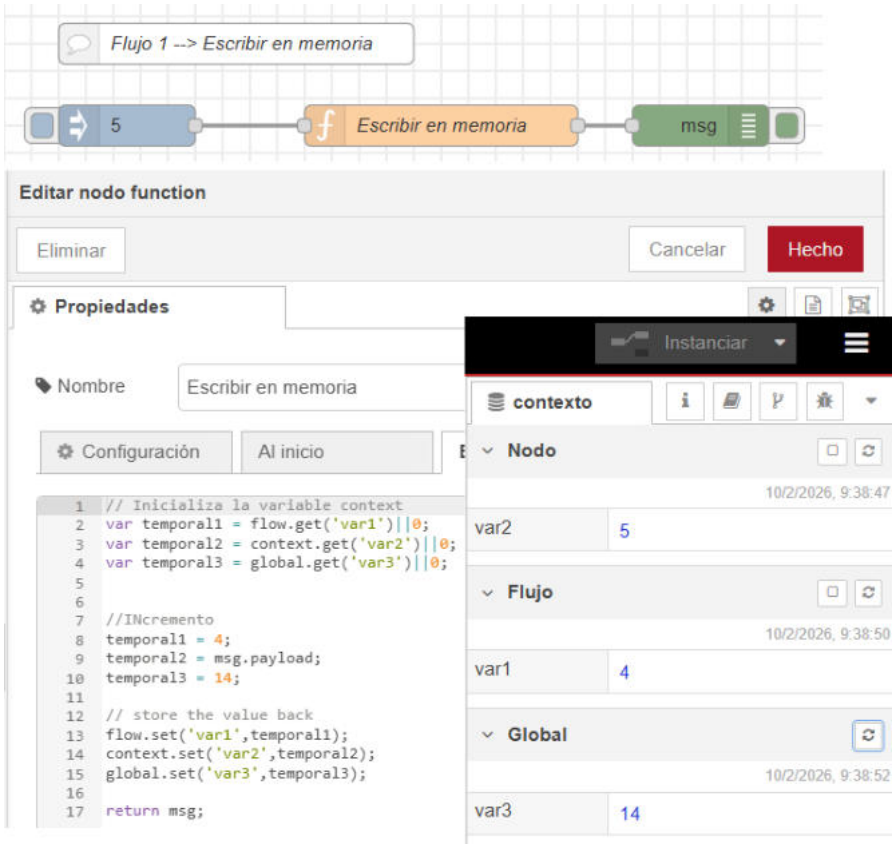


Figura 64. Ejemplo de uso de persistencia en la memoria RAM a través de las variables flow, context y global. El valor de las variables var1, var2 y var3, aparecen en el extremo derecho de la figura.

Luego, ejecutamos el segundo flujo, que incrementa en 1 a cada una de las variables en memoria, como se puede apreciar en **¡Error! No se encuentra el origen de la referencia.**.

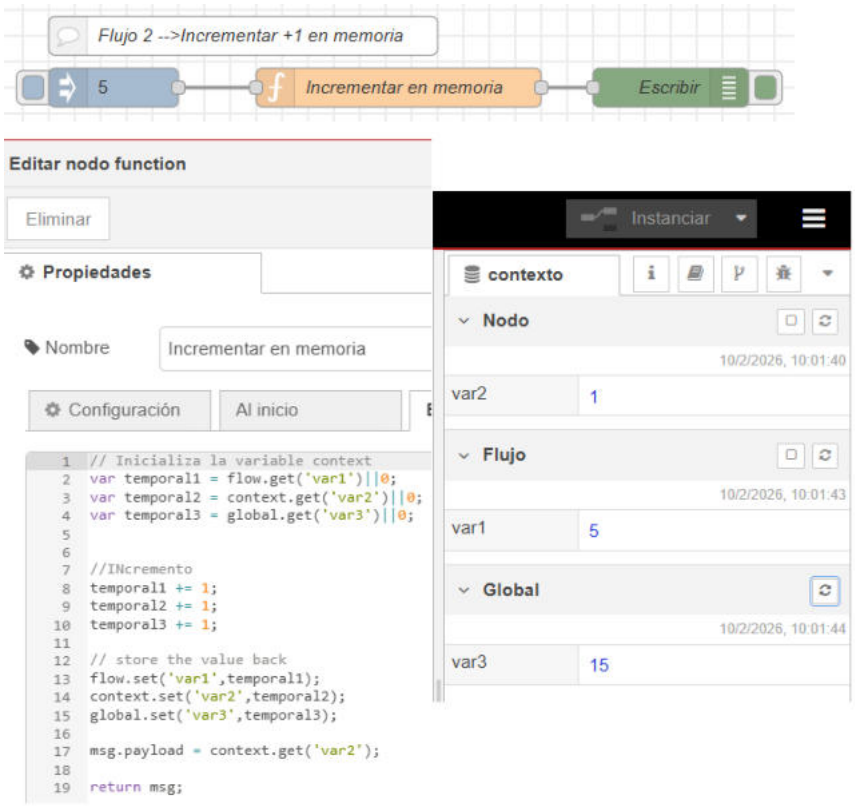


Figura 65. Ejemplo de persistencia de las variables var1, var 2 y var3, luego de un incremento por medio de la ejecución del flujo.

Puede comprobar que, si detiene el flujo y lo vuelve a instanciar, las variables var 1 y var 3 mantienen su valor en memoria.

⚠️ PRECAUCIÓN – Objetos msg complejos pueden dificultar el debug

Cuando los mensajes tienen muchos niveles anidados, es mejor usar debug configurado en modo “complete msg object” para visualizar todo el contenido y no solo el payload.

3.1.2 Mensajes y estructuras anidadas

En Node-RED, la unidad fundamental de información que viaja a través de los nodos es el **mensaje**, representado como un objeto JavaScript llamado msg. Este objeto es altamente flexible y puede contener múltiples propiedades, no solo msg.payload. Comprender su estructura y la forma en que pueden definirse **objetos anidados** es clave para desarrollar flujos avanzados, eficientes y ordenados.

Un mensaje básico puede verse así:

```
json

CopiarEditar

{

 "payload": 25.4

}
```

Sin embargo, en flujos reales —especialmente en proyectos IoT— es habitual trabajar con **mensajes más complejos**, por ejemplo:

```
json

CopiarEditar

{

 "payload": {
```

```
  "temperatura": 25.4,
  "humedad": 60,
  "unidad": "°C"
 },
 "topic": "sensor/clima",
 "deviceID": "esp32_01",
 "timestamp": 1684158900000
}
```

Estas estructuras permiten encapsular múltiples variables, metadatos y contexto en un solo objeto, lo que **facilita la integración, el filtrado y el procesamiento condicional** sin necesidad de múltiples flujos paralelos.

- **Acceso a propiedades anidadas**

Desde cualquier nodo (por ejemplo, function, switch, change), se puede acceder o modificar las propiedades anidadas mediante la notación de punto (.) o la notación de corchetes ([]):

```
javascript

CopiarEditar

let temp = msg.payload.temperatura;
```

```
msg.payload.humedad = 65;
```

O también:

```
javascript

CopiarEditar

let unidad = msg["payload"]["unidad"];
```

- **Creación de estructuras anidadas en** function

Puedes construir objetos personalizados dentro de un nodo function para adaptar el mensaje a servicios externos, APIs o dashboards:

```
javascript

CopiarEditar

msg.payload = {
  lectura: {
    temperatura: 23.7,
    humedad: 58
  },
  dispositivo: "node1",
  fecha: new Date().toISOString()
```

```
};

return msg;
```

- **Uso del nodo** change **para manipular estructuras anidadas**

El nodo change permite acceder y modificar propiedades profundas utilizando la misma notación:

- **Set** msg.payload.estado to "OK"
- **Move** msg.payload.temp to msg.datos.ambiente.temperatura

Esto evita el uso de nodos function para operaciones simples.

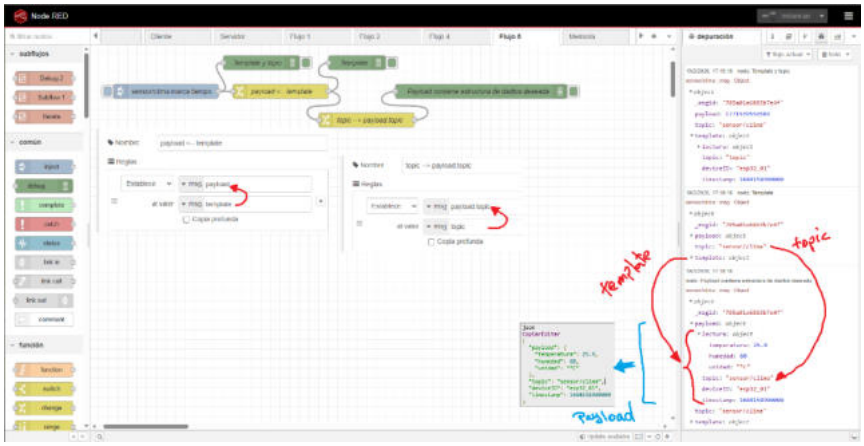


Figura 66. Uso del nodo change para acceder y modificar estructuras anidada en un Json.

El flujo de la **¡Error! No se encuentra el origen de la referencia.**66, hace uso del nodo Change, para crear el típico mensaje IoT complejo, mencionado en un ejemplo anteriormente, donde se tiene las lecturas de temperatura (25.4) y humedad (60), unidad (ºC), tópico (sensor/clima), deviceid (esp32_01) y el timestamp. Para ello, se parte de un esqueleto del mensaje (tempate) al cual se le actualizan ciertos campos en tiempo real. El mensaje completo queda listo en el payload. Utilizando esta metodología puede crear mensajes con multiples anidaciones en función de la arquitectura y aplicación que tenga que implementar en el futuro.

🔸 **Ventajas de usar estructuras anidadas**

- Mayor organización del mensaje.
- Facilita el uso de switch, template y dashboards personalizados.
- Mejora la interoperabilidad con APIs REST, bases de datos y servicios externos.
- Permite representar jerarquías lógicas del mundo real (sensor → ubicación → tipo de dato).

🧠 Conclusión

Aprender a trabajar con mensajes estructurados y propiedades anidadas permite desarrollar flujos **más limpios, escalables y profesionales**. Esta habilidad es fundamental para la integración con sistemas externos, la visualización avanzada y la manipulación de datos complejos en entornos IoT.

3.2 Trabajo con proyectos en Node-RED

Para entornos de desarrollo profesional o educativos o colaborativos, es común manejar múltiples flujos de Node-RED en paralelo, realizar pruebas de versiones separadas o mantener el control de cambios sobre lo que se implementa. Para estos casos, Node-RED ofrece una funcionalidad avanzada denominada Proyectos, la cual permite gestionar el desarrollo de flujos como si se tratara de repositorios Git, incluyendo control de versiones, historial de cambios, ramas y comentarios.

Esta característica convierte a Node-RED en una plataforma robusta y profesional, ofertando a los desarrolladores un entorno organizado para trabajar de manera controlada, incluso en equipos. Los proyectos son especialmente útiles cuando se desarrollan soluciones IoT complejas o cuando se requiere mantener un entorno de producción separado del de pruebas.

3.2.1 Activación del sistema de proyectos

Por defecto, el modo proyectos viene deshabilitado. Para activarlo:

1. Ir al archivo settings.js.

2. Buscar la siguiente sección y modificarla:

```
js

CopiarEditar

projects: {
```

```
    enabled: true,
    workflow: {
        mode: "manual"
    }
}
```

3. Guardar y reiniciar Node-RED.

Una vez activado, al abrir el editor se solicitará crear o seleccionar un proyecto antes de acceder a los flujos.

En **¡Error! No se encuentra el origen de la referencia.**67 se puede apreciar el archivo settings.js

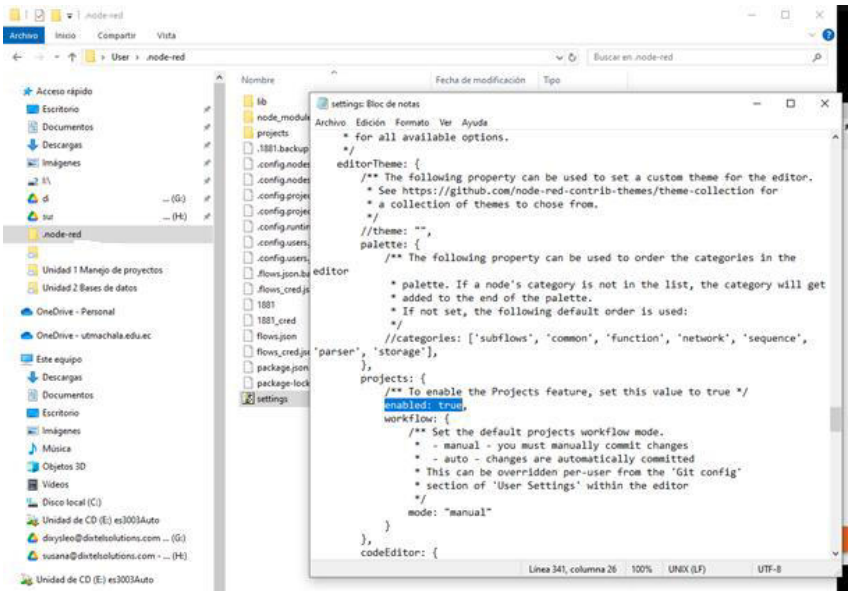


Figura 67. Habilitación del uso de proyectos dentro de Node-red en el archivo settings.js.

3.2.2 Crear un nuevo proyecto

Desde el editor:

1. Hacer clic en el menú (☰) → *Projects* → *Create New Project*.

2. Asignar un nombre, descripción y credenciales (opcional).

3. Node-RED creará una carpeta con archivos de configuración:

- flows.json
- flows_cred.json
- README.md
- .gitignore
- Archivos Git (.git)

Esto permite mantener **un repositorio de flujo versionado** de forma automática.

3.2.3 Versionamiento y backups

Una de las ventajas distintivas del modo Proyectos en Node-RED es su integración con Git, una herramienta de control de versiones que permite llevar un registro completo de los cambios realizados sobre los flujos, configuraciones y archivos asociados. Esto no solo facilita el seguimiento y la recuperación de versiones anteriores, sino que también

mejora la colaboración entre varios desarrolladores y reduce el riesgo de pérdida de información.

Además del versionamiento, Node-RED permite realizar copias de seguridad (backups) completas de todo el entorno de trabajo, esencial para proteger los flujos ante fallos del sistema, errores humanos o actualizaciones inesperadas. Ambas estrategias de versionamiento y respaldo deben considerarse como prácticas complementarias, pero útiles para mantener un desarrollo IoT robusto y sostenible.

Una vez que se activa el modo proyecto, Node-RED inicia automáticamente un repositorio Git en la carpeta del proyecto. Desde la interfaz gráfica es posible hacer lo siguiente:

- **Hacer commits:** guardar cambios con una descripción.
- **Ver el historial de versiones:** acceder a cambios anteriores por fecha, usuario y descripción.
- **Revertir versiones:** volver a un estado anterior en caso de errores o fallos.
- **Crear ramas:** separar versiones para pruebas sin afectar el flujo principal.
- **Conectar con repositorios remotos (GitHub (GitHub.com Documentación De Ayuda, 2026), GitLab, etc.):** útil para sincronización en la nube, trabajo en equipo o restauración desde otros equipos.

Ejemplo de flujo de trabajo:

1. Modificar un flujo.
2. Hacer clic en *Commit Changes* y escribir una descripción.
3. (Opcional) Hacer *Push* si está conectado a un repositorio remoto.

💾 Backups del entorno Node-RED

Aunque el control de versiones con Git es poderoso, se recomienda complementar con **backups periódicos del entorno completo** (node-red o carpeta del proyecto) para cubrir situaciones como:

- Corrupción del archivo flows.json.
- Incompatibilidad de nodos al actualizar.
- Reemplazo de hardware o migración de sistema.
- Restauración rápida sin depender de Git.

En **¡Error! No se encuentra el origen de la referencia.**68 se han enumerado 4 pasos que de forma resumida debes llevar a cabo para teern un proyecto de node-red actualizado en un repositorio Git. Para profundizar en este tema puede revisar la página oficial de node-red (https://nodered.org/docs/user-guide/projects/).

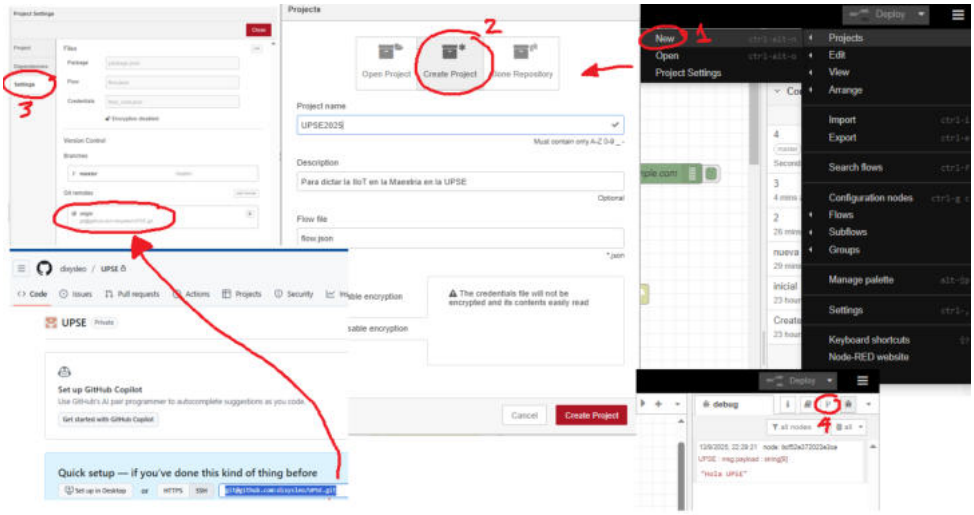


Figura 68. Pasos a seguir para crear un nuevo proyecto desde la interfaz gráfica de Node-red.

Métodos de respaldo sugeridos:

- Copiar la carpeta del proyecto manual o automáticamente (ver epígrafe 1.5.x).
- Comprimir la carpeta en .zip o .tar.gz.
- Automatizar con cron (Linux) o Task Scheduler (Windows).
- Almacenar en disco externo o nube (Google Drive, Dropbox, Nextcloud, etc.).

Ejemplo de estructura de backup:

pgsql

CopiarEditar

/backups/nodered-2025-05-15/

├── flows.json

├── flows_cred.json

├── settings.js

├── package.json

├── node_modules/

└── .git/

☑ Buenas prácticas

- Realiza un **commit** cada vez que completes una tarea o hagas cambios importantes.
- Añade descripciones claras y útiles en los mensajes de commit.
- Usa **ramas separadas** para desarrollo y producción.
- Programa **backups automáticos diarios o semanales**.
- Antes de actualizar Node-RED o instalar nuevos nodos**, haz un backup completo.**
- Si usas repositorios remotos, **asegura autenticación con clave SSH o token personal.**

Ventajas del uso de proyectos

    - **Control de versiones** con historial completo.
- **Separación de ambientes**: desarrollo, pruebas y producción.
- **Colaboración** entre múltiples usuarios con herramientas estándar (Git).
- **Organización estructurada** de archivos y flujos.
- **Seguridad**: permite tener copias confiables y regresar a estados estables ante errores.

⚠️ **Recomendaciones**

- Evitar proyectos si el usuario no está familiarizado con Git (al menos a nivel básico).
- Usar nombres descriptivos y commits claros.
- No incluir archivos sensibles (como certificados o claves) dentro del proyecto.

🧠 Conclusión

Usar el modo proyectos eleva a Node-RED a un nivel profesional de desarrollo, al integrar capacidades típicas de otros entornos de software colaborativo. Esta funcionalidad es especialmente útil en aplicaciones IoT avanzadas, en contextos educativos con prácticas estructuradas, y en entornos industriales donde la trazabilidad del código y la gestión de versiones son críticas.

El uso combinado de Git para versionamiento y backups periódicos del entorno garantiza la **trazabilidad, seguridad y portabilidad** de los proyectos desarrollados en Node-RED. Esto no solo protege el trabajo del desarrollador, sino que también sienta las bases de buenas prácticas profesionales de desarrollo en entornos IoT, académicos e industriales.

3.3 Programación con funciones

Aunque Node-RED está diseñado para simplificar el desarrollo mediante programación visual, existen numerosos escenarios en los que es necesario aplicar **lógica personalizada**, manipular estructuras de datos complejas o responder a condiciones no cubiertas por nodos básicos. Para estos casos, Node-RED proporciona el **nodo** function, que permite escribir código JavaScript (JavaScript Reference - JavaScript | MDN, 2025) directamente dentro del flujo.

Este nodo actúa como un puente entre la tradicional programación textual y la eficiencia del entorno visual, ofreciendo acceso completo al objeto msg, así como a las variables de contexto locales: context, flow y global. Dominar la programación con funciones permitirá construir flujos avanzados, dinámicos y adaptables a entornos IoT exigentes.

3.3.1 Uso del nodo function en JavaScript

El nodo function contiene un editor de texto donde se escribe el código en JavaScript. La sintaxis básica es:

```
javascript

CopiarEditar
```

```
msg.payload = "Hola desde función";

return msg;
```

- **Entrada:** recibe un objeto msg.
- **Procesamiento:** se aplica lógica sobre msg y/o contexto.
- **Salida:** debe retornar msg (o un array de mensajes).

Si no se retorna nada, el flujo se detiene.

Manipulación del objeto msg

El objeto msg puede contener múltiples propiedades, que puedes crear, leer o modificar:

```
javascript

CopiarEditar

msg.temperatura = 22.5;

msg.payload = msg.payload + 10;

msg.topic = "dispositivo/alerta";

return msg;
```

También puedes eliminar propiedades:

```
javascript
```

```
CopiarEditar

delete msg.payload;
```

Salidas múltiples

Un nodo function puede tener más de una salida. Puedes controlar qué mensaje sale por cada una:

```
javascript

CopiarEditar

if (msg.payload > 30) {

    return [msg, null];  // salida 1

} else {

    return [null, msg];  // salida 2

}
```

Para enviar múltiples mensajes desde una salida:

```
javascript

CopiarEditar

return [[msg1, msg2, msg3], null];
```

Uso de contexto en funciones

Puedes almacenar o recuperar variables persistentes desde dentro de la función:

```
javascript

CopiarEditar

let contador = context.get("cont") || 0;

contador++;

context.set("cont", contador);

msg.payload = contador;

return msg;
```

Esto es útil para contadores, sesiones o datos que se deben conservar entre ejecuciones.

⚠️ **PRECAUCIÓN – Uso excesivo de nodos** function

Aunque es tentador resolver todo con código, **Node-RED promueve la programación visual.** Si puedes lograrlo con switch, change, template o json, es preferible por legibilidad y mantenibilidad.

3.3.2 Validación y manejo de errores

Las funciones pueden implementar estructuras if, try/catch, y lógica compleja:

```
javascript
```

```
CopiarEditar

try {
    if (typeof msg.payload !== "number") {
        throw "El payload debe ser un número";
    }
    msg.payload = msg.payload * 2;
    return msg;
} catch (err) {
    node.error(err, msg);
    return null;
}
```

El uso de node.error() permite visualizar errores en el panel de depuración.

☑ Aplicaciones comunes

- Calcular valores derivados (ej. conversión de unidades).
- Verificar estructuras JSON y validar datos.
- Controlar múltiples salidas dinámicamente.

- Generar respuestas personalizadas a APIs o sensores.
- Crear flujos condicionales complejos que no pueden expresarse con nodos básicos.

🧠 Conclusión

El nodo function es una herramienta poderosa para extender las capacidades de Node-RED más allá del paradigma visual. Al combinarlo con las estructuras de datos, variables de contexto y flujos condicionales, permite desarrollar **soluciones altamente personalizadas y profesionales** en proyectos IoT complejos.

💡 **TIP TÉCNICO – Cómo saber si una variable está en memoria o en disco**

Cuando activas la persistencia en settings.js, puedes ver en el panel derecho (pestaña *Context Data*) si una variable está en memory o en file. Esto te ayuda a saber qué se conservará tras reiniciar Node-RED.

💡 **TIP TÉCNICO – Validar tipo de dato sin** function

Puedes usar el nodo switch para validar si un valor es numérico (is number) o igual a cierto texto. Esto permite hacer validaciones sin escribir código.

3.3.3 Importación de librerías externas

Node-RED, al estar construido sobre Node.js, permite extender sus capacidades más allá de los nodos básicos mediante la **importación de**

librerías externas. Estas permiten realizar tareas avanzadas tales como manipulación de fechas, criptografía, operaciones matemáticas complejas, acceso a APIs, procesamiento de archivos, entre otras. Su uso es especialmente valioso dentro del **nodo** function, donde se invocan funciones de dichas librerías como si se tratara de un entorno JavaScript tradicional.

🔧 ¿Cómo instalar una librería externa?

1. Accede al directorio del usuario de Node-RED:

```
bash

CopiarEditar

cd ~/.node-red
```

2. Instala la librería usando npm. Ejemplos:

- moment para fechas:

```
bash

CopiarEditar

npm install moment
```

- axios para hacer peticiones HTTP:

```
bash

CopiarEditar
```

```
npm install axios
```

- mathjs para operaciones matemáticas:

```
bash

CopiarEditar

npm install mathjs
```

3. Reinicia Node-RED para que reconozca la nueva librería.

🧠 **¿Cómo usar una librería dentro de un nodo** function**?**

Desde la versión 1.2 de Node-RED, es posible habilitar el uso de require() dentro del nodo function. Para ello:

1. Abre el archivo settings.js.

2. Busca y activa la siguiente opción:

```
js

CopiarEditar

functionExternalModules: true
```

3. Reinicia Node-RED.

Ahora puedes importar una librería en un nodo function así:

```
javascript
```

```
CopiarEditar

const moment = require("moment");

msg.payload = "Fecha actual: " + moment().format("YYYY-MM-DD
        HH:mm:ss");

return msg;
```

📌 **Ejemplo práctico con** mathjs**:**

1. Instala la librería:

```
bash

CopiarEditar

npm install mathjs
```

2. En un nodo function:

```
javascript

CopiarEditar

const math = require("mathjs");

let resultado = math.evaluate("sqrt(49) + 7 * 3");

msg.payload = "Resultado: " + resultado;

return msg;
```

☑ Salida: "Resultado: 28"

⚠️ **Recomendaciones de seguridad**

- Habilitar require() expone Node-RED a la ejecución de módulos externos; solo debe hacerse en entornos controlados.
- Revisa la reputación de cada paquete en https://www.npmjs.com.
- Evita usar librerías muy pesadas en dispositivos embebidos como Raspberry Pi si afectan el rendimiento.

🧠 Conclusión

La posibilidad de importar librerías externas dentro de funciones convierte a Node-RED en una **plataforma híbrida entre programación visual y textual**, resolviendo además problemas complejos, integrando servicios específicos y realizando cálculos avanzados sin necesidad de nodos adicionales. Todas estas bondades, permiten desarrolar aplicaciones profesionales de IoT, con flujos de precisión, optimización y adaptabilidad.

3.4 Nodos personalizados

Node-RED permite **crear nodos personalizados**, adaptados a necesidades específicas de los desarrolladores en sus proyectos. Esta lo convierte en una plataforma verdaderamente extensible, donde se pueden encapsular lógicas complejas, conectar hardware propietario o implementar protocolos no soportados de forma nativa.

Un **nodo personalizado** es esencialmente un **módulo Node.js con una definición JSON de su interfaz visual y un archivo JavaScript** que contiene la lógica de ejecución. Estos nodos pueden ser empaquetados como módulos reutilizables, compartidos con otros desarrolladores o publicados en el catálogo oficial de Node-RED (https://flows.nodered.org).

3.4.1 Estructura básica de un nodo personalizado

Un nodo personalizado se compone de:

css

CopiarEditar

mi-nodo/

├── mi-nodo.js → lógica del nodo (servidor)

├── mi-nodo.html → definición visual del nodo (cliente)

├── package.json → metadatos del paquete

└── icon.png → (opcional) ícono personalizado

Cada archivo tiene una función clara:

- mi-nodo.js: contiene el código que se ejecuta cuando el nodo recibe un mensaje.
- mi-nodo.html: define cómo se configura visualmente el nodo en el editor.

- package.json: especifica nombre, versión, dependencias, y registra el nodo con Node-RED.

💡 **TIP TÉCNICO – Un nodo personalizado puede ser una unidad reutilizable**

Si hay una lógica que repites en varios flujos, empaquétala como nodo propio. Puedes compartirla con otros o mantenerla en tu organización como "bloque de construcción" estándar.

3.4.2 Crear un nodo personalizado paso a paso

1. **Crear la estructura del módulo:**

```
bash

CopiarEditar

mkdir node-red-contrib-mi-nodo-libro

cd node-red-contrib-mi-nodo-libro

npm init -y
```

2. **Modificar el** package.json**:**

```
json

CopiarEditar

{
```

```
  "name": "node-red-contrib-mi-nodo-libro",
  "version": "1.0.0",
  "description": "Nodo personalizado de ejemplo",
  "dependencies": {},
  "node-red": {
    "nodes": {
      "mi-nodo": "mi-nodo-libro.js"
    }
  }
}
```

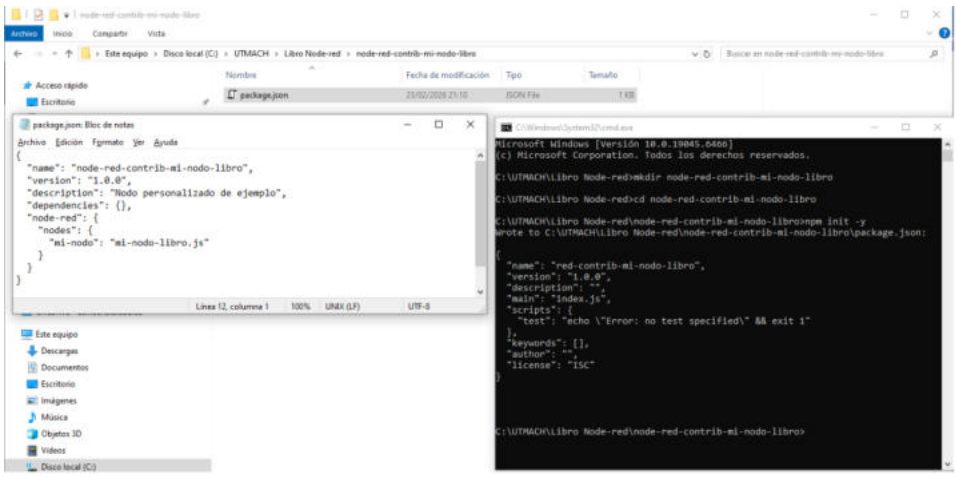


Figura 69. Primeros pasos para crear un nodo propio, modificando el archivo package.json.

La **¡Error! No se encuentra el origen de la referencia.**69 muestra el resultado de los 2 pasos anteriores.

3. **Crear el archivo** mi-nodo-libro.js**:**

```
javascript

CopiarEditar

module.exports = function(RED) {

  function MiNodo(config) {

    RED.nodes.createNode(this, config);

    var node = this;

    node.on('input', function(msg) {

      msg.payload = "Mensaje recibido: " + msg.payload;

      node.send(msg);

    });

  }

  RED.nodes.registerType("mi-nodo-libro", MiNodo);

}
```

4. **Crear el archivo** mi-nodo-libro.html**:**

```
html

CopiarEditar

<script type="text/javascript">

    RED.nodes.registerType('mi-nodo-libro', {

        category: 'function',

        color: '#a6bbcf',

        defaults: {

            name: {value:""}

        },

        inputs:1,

        outputs:1,

        icon: "icon.png",

        label: function() {

            return this.name || "mi-nodo-libro";

        }

    });

</script>
```

```
<div class="form-row">

    <label for="node-input-name"><i class="fa fa-tag"></i>
        Nombre</label>

    <input type="text" id="node-input-name">

</div>
```

La ¡Error! No se encuentra el origen de la referencia. muestra el resultado de los pasos 3, 4 y 5.

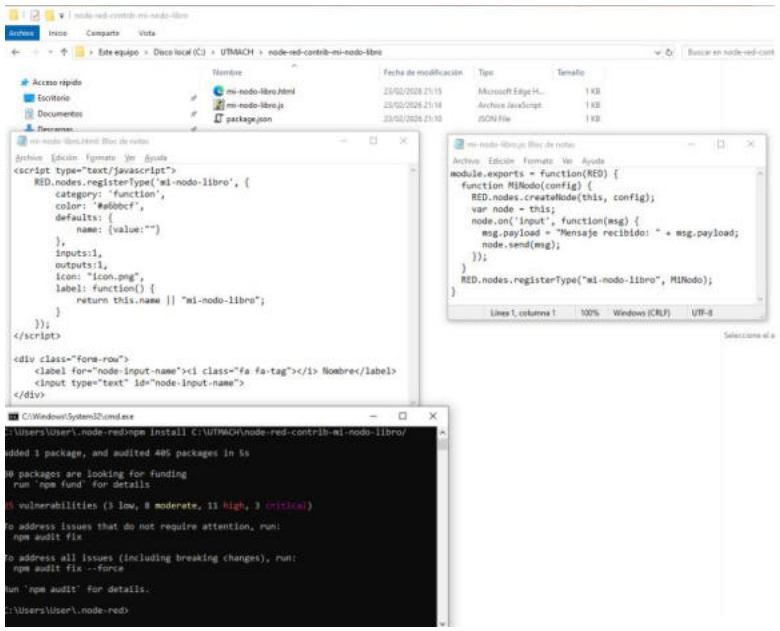


Figura 70. Creacion de archivos.js y .html e instalacion del nodo propio dentro del ecosistema de node-red.

5. **Instalar el nodo localmente para probar:**

```
bash

CopiarEditar

C:\Users\User\.node-red

npm install C:\UTMACH\node-red-contrib-mi-nodo-libro/
```

6. **Reiniciar Node-RED**. El nodo aparecerá en la categoría "function" del editor, como se puede apreciar en **¡Error! No se encuentra el origen de la referencia.**71. Del lado izquierdo esta nuestro nuevo nodo propio creado y a la derecha es usado en un nuevo flujo. En aras de simplificar el ejemplo, este nodo solo traslada la entrada a la salida.

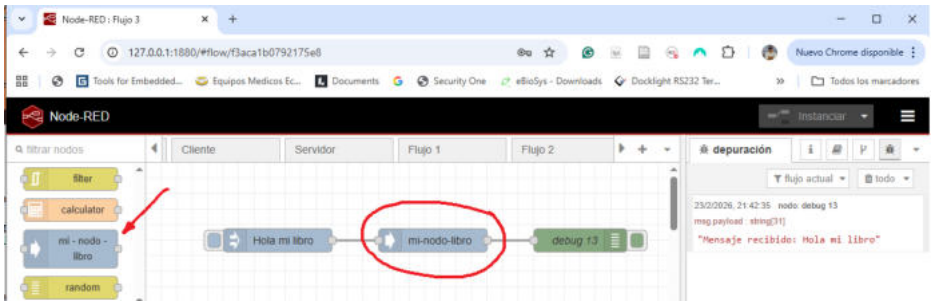


Figura 71. Nodo propio creado: mi-nodo-libro, ya aparece en node-red y es usado en un flujo sencillo, mostrando resultados en el debug.

3.4.3 Publicar el nodo en NPM

Una vez probado, puedes publicar tu nodo para uso público:

```
bash

CopiarEditar
```

npm login

npm publish

Y aparecerá en https://flows.nodered.org si cumple los requisitos del ecosistema.

☑ Casos comunes de uso

- Conectar hardware específico (sensores, actuadores, APIs industriales).
- Encapsular una función usada frecuentemente en varios flujos.
- Crear interfaces simplificadas para usuarios sin conocimientos técnicos.
- Controlar flujos complejos con múltiples parámetros desde un solo nodo visual.

🧠 Conclusión

Los nodos personalizados extienden las capacidades de Node-al permitir a los desarrolladores **adaptar la plataforma a cualquier entorno de trabajo o solución IoT específica**. Esta práctica combina lo mejor del desarrollo visual con la potencia de Node.js, y representa el nivel más avanzado de personalización dentro del ecosistema Node-RED (Creating Your First Node : Node-RED, 2026).

3.4.4 Registro en el flujo de trabajo

Una vez creado un nodo personalizado, el paso fundamental para que pueda ser utilizado dentro de Node-RED es su **registro en el entorno de ejecución y su integración en el editor visual**. Este proceso permite que el nodo aparezca junto a los demás en la paleta, pueda ser arrastrado al lienzo, configurado por el usuario, y conectado dentro de cualquier flujo como si se tratara de un nodo nativo.

Este "registro en el flujo de trabajo" implica dos niveles:

- El **registro lógico** en el runtime (servidor de Node-RED), donde se define la funcionalidad.
- El **registro visual** en el editor (cliente del navegador), donde se define cómo el usuario interactúa con él.

🔧 **Registro lógico (archivo** .js**)**

En el archivo JavaScript principal del nodo personalizado (por ejemplo, mi-nodo-libro.js), se realiza el registro usando:

```
javascript

CopiarEditar

RED.nodes.registerType("mi-nodo-libro", MiNodo);
```

Esto indica al sistema que se está definiendo un nodo con el identificador "mi-nodo", cuya funcionalidad estará contenida en la función

MiNodo(config). Aquí también se gestiona cómo responde el nodo a eventos como input, close, o error.

Cada instancia del nodo será creada mediante:

```
javascript

CopiarEditar

RED.nodes.createNode(this, config);
```

Esta línea vincula la instancia al editor visual y permite acceder a sus propiedades (this.name, this.status, etc.).

🎨 **Registro visual (archivo** .html**)**

En el archivo HTML correspondiente (por ejemplo, mi-nodo.html), se registra el nodo en el **editor visual** mediante:

```
html

CopiarEditar

RED.nodes.registerType('mi-nodo-libro', {

    category: 'function',

    color: '#a6bbcf',

    defaults: {

        name: {value:""}
```

```
    },
    inputs: 1,
    outputs: 1,
    icon: "icon.png",
    label: function() {
        return this.name || "mi-nodo";
    }
});
```

Este fragmento define:

- En qué **categoría** aparecerá el nodo (como "function", "input", "output", etc.).
- ¿Cuántas **entradas y salidas** tienen?
- ¿Qué **parámetros configurables** se mostrarán en el editor?
- ¿Qué **etiqueta** se mostrará sobre el nodo?
- ¿Qué **color e ícono** tendrá?

📥 Inclusión automática al instalar

Cuando un usuario instala el nodo personalizado en su entorno (vía npm install), Node-RED:

1. Busca en el package.json la propiedad node-red.nodes.
2. Carga los archivos .js y .html registrados.
3. Agrega el nodo a la paleta después del reinicio.

Esto permite que el nodo se use inmediatamente dentro de cualquier flujo de trabajo.

☑ Verificación del registro

Una vez instalado el nodo:

- Debe aparecer en la paleta bajo la categoría definida.
- Al arrastrarlo al lienzo, debe mostrar su interfaz de configuración.
- Al desplegar el flujo, el nodo ejecutará su función definida en el servidor.

Puedes verificar que el nodo está activo en la terminal al iniciar Node-RED o inspeccionando la pestaña **Manage palette** en el menú del editor.

🧠 Conclusión

El registro correcto de un nodo personalizado es el paso final que lo convierte en una herramienta de desarrollo visual integrada al entorno de Node-RED. Esta integración bidireccional, entre la lógica ejecutable y la interfaz gráfica, permite que el nodo sea usado, documentado, compartido y desplegado como cualquier otro nodo oficial. Comprender

esta arquitectura modular es esencial para extender la plataforma de manera profesional y segura.

3.5 Seguridad avanzada

En despliegues de proyectos reales, no educativos, la seguridad no es una opción, sino una necesidad crítica. Node-RED, al ser una plataforma accesible vía navegador y extensible mediante JavaScript, debe protegerse adecuadamente para evitar accesos no autorizados, manipulaciones maliciosas, exposición de datos o interferencias en la operación de dispositivos conectados.

Este epígrafe presenta técnicas y configuraciones que fortalecen la seguridad en el editor, en los endpoints expuestos (HTTP/MQTT), y en la comunicación de datos, abordando temas como autenticación robusta, uso de HTTPS, aislamiento de red, control de permisos, y seguridad en nodos function.

3.5.1 Gestión de usuarios y contraseñas

Además del mecanismo básico de usuario/contraseña definida en el archivo: *settings.js*, se pueden implementar configuraciones más robustas:

- **Multiples usuarios** con diferentes niveles de permiso:

```
js

CopiarEditar

adminAuth: {
```

```
 type: "credentials",
 users: [
  {
   username: "admin",
   password: "hash_bcrypt",
   permissions: "*"
  },
  {
   username: "usuario1",
   password: "otro_hash",
   permissions: "read"
  }
 ]
}
```

Esto permite restringir acciones como edición, despliegue y administración.

- **Integración con OAuth2 / LDAP / SSO**: con configuración adicional y módulos externos, es posible integrar el acceso con sistemas corporativos de identidad.

3.5.2 Protección de nodos HTTP (http in, http response)

Los endpoints expuestos por nodos http in deben protegerse mediante:

- **Autenticación básica**, configurando en settings.js:

```
js

CopiarEditar

httpNodeAuth: { user: "apiuser", pass: "bcrypt_hash" }
```

- **Verificación manual en el flujo**, usando un nodo function para validar tokens, IPs, headers o claves personalizadas.
- **Uso de HTTPS**, como se verá en el siguiente punto.

3.5.3 Permisos en entornos colaborativos

Para cifrar la comunicación entre el navegador y el servidor Node-RED:

1. Generar o adquirir un certificado SSL:
   - Local: con openssl.
   - Público: con **Let's Encrypt** (recomendado en servidores reales).
2. Modificar settings.js:

```
js

CopiarEditar

https: {

  key: require("fs").readFileSync("/ruta/privkey.pem"),

  cert: require("fs").readFileSync("/ruta/fullchain.pem")

},



requireHttps: true
```

3. Reiniciar Node-RED y acceder vía https://.

Esto evita que contraseñas o tokens se transmitan sin cifrado.

3.5.4 Seguridad en nodos function y uso de require()

Si se permite require() en nodos function, es importante:

- Activarlo solo si se requiere en settings.js:

```
js

CopiarEditar

functionExternalModules: true
```

- Validar las librerías instaladas (evitar código malicioso).

- Usar node.error() y try/catch para evitar que excepciones interrumpan flujos.
- **Nunca** ejecutar comandos del sistema (como child_process) sin validación estricta.

3.5.5 Aislamiento por red

- Ejecutar Node-RED **detrás de un reverse proxy** (como Nginx, Caddy o Apache).
- Usar firewalls (UFW, iptables, reglas de cloud) para **permitir solo IPs confiables**.
- En soluciones distribuidas: conectar dispositivos mediante **VPN (WireGuard, Tailscale)** para mantener el tráfico en una red segura.

3.5.6 Supervisión y logs

- Usar la opción de audit en settings.js para registrar eventos sensibles:

```
js

CopiarEditar

logging: {

  console: {

    level: "info",
```

```
    audit: true
  }
}
```

- Habilitar registros de cambios de flujo, accesos y errores.

☑ Recomendaciones finales

- Nunca dejes Node-RED expuesto sin autenticación ni cifrado.
- Realiza auditorías regulares de nodos instalados y configuraciones activas.
- Restringe la capacidad de editar flujos a usuarios autorizados.
- Usa backups cifrados en entornos sensibles.

🧠 Conclusión

La seguridad avanzada en Node-RED debe ser aplicada para proteger tanto los datos como losdispositivos y usuarios en entornos productivos. Una configuración robusta basada en autenticación, cifrado, control de red y supervisión activa, convierte a Node-RED en una plataforma robusta y fiable para la implementación de soluciones IoT profesionales a niel industrial.

Elemento de cierre

Resumen del capítulo:

En este capítulo profundizaste en el diseño avanzado de flujos con

Node-RED, aprendiendo a trabajar con variables de contexto, estructuras anidadas, lógica programada con funciones, integración de librerías externas y creación de nodos personalizados. También conociste el sistema de proyectos con control de versiones, y aplicaste buenas prácticas de seguridad para entornos reales.

Preguntas de revisión

❓ **Pregunta 1:** ¿Qué tipo de variable permite compartir datos entre diferentes flujos?

✍️ a) context

✍️ b) flow

✍️ c) global

✍️ d) payload

☑️ **Respuesta correcta:** c) global

❓ **Pregunta 2:** ¿Qué nodo permite definir lógica personalizada en JavaScript?

✍️ a) switch

✍️ b) function

✍️ c) template

✍️ d) change

☑️ **Respuesta correcta:** b) function

❓ **Pregunta 3:** ¿Cuál es el propósito del archivo settings.js en Node-RED?

✍️ a) Guardar variables globales

✍️ b) Crear flujos de automatización

✍️ c) Configurar opciones del sistema, seguridad y módulos

✍️ d) Ejecutar scripts externos

☑️ **Respuesta correcta:** c) Configurar opciones del sistema, seguridad y módulos

❓ **Pregunta 4:** ¿Qué herramienta permite gestionar cambios y versiones en proyectos Node-RED?

✍️ a) Dashboard

✍️ b) SQLite

✍️ c) Git

✍️ d) Context Storage

☑️ **Respuesta correcta:** c) Git

Clave de respuestas

Pregunta	Respuesta correcta
1	c) global

Pregunta	Respuesta correcta
2	b) function
3	c) settings.js
4	c) Git

CAPÍTULO IV

4 IOT EN ACCIÓN

🎯 Objetivos de aprendizaje:

- Integrar protocolos de comunicación como MQTT, HTTP y UART en flujos IoT.
- Conectar Node-RED con sensores y actuadores a través de GPIO, I²C y SPI.
- Almacenar datos en archivos, bases de datos locales o plataformas cloud.
- Diseñar dashboards visuales para interacción en tiempo real.
- Integrar Node-RED con plataformas domóticas como Home Assistant y ESPHome.

🛠️ Competencias:

- Desarrolla soluciones IoT completas usando Node-RED como orquestador.
- Implementa comunicaciones entre dispositivos físicos y servicios en red.
- Diseña interfaces visuales funcionales para monitoreo y control.
- Gestiona almacenamiento de datos en distintos formatos y ubicaciones.

❓ Preguntas de enfoque:

- ¿Qué protocolos son más adecuados para comunicar mis dispositivos?
- ¿Cómo puedo controlar un sensor o actuador físico desde Node-RED?
- ¿Qué opciones tengo para almacenar datos en tiempo real?
- ¿Cómo puedo visualizar e interactuar con mis flujos desde una interfaz web?

Este capítulo tiene como objetivo aplicar todo lo aprendido de Node-RED en el ámbito del Internet de las Cosas (IoT), integrando los dispositivos físicos con protocolos de comunicación, servicios en la nube y plataformas de automatización. Con ejemplos prácticosde conectividad con hardware y software específico, el lector podrá consolidar sus conocimientos y llevar sus flujos desde la simulación hast convertirlas en aplicaciones funcionales para sistemas reales.

denle este capítulo se explorarán las formas más comunes de integración en IoT, por ejemplo a través de lectura de sensores con interfaces GPIO, UART, I²C o SPI; envío y recepción de datos mediante protocolos MQTT y HTTP; almacenamiento local y en la nube, conexión con plataformas como Home Assistant y ESPHome y la creación de dashboards funcionales con estilos personalizados. Al finalizar, el lector será capaz de diseñar soluciones completas y operativas de IoT con

Node-RED, incluyendo el monitoreo, automatización y visualización de datos en tiempo real.

4.1 Protocolos de comunicación

En el ecosistema IoT, los protocolos de comunicación son fundamentales para permitir el intercambio de información entre dispositivos, plataformas, servicios y usuarios. Node-RED, como herramienta de orquestación visual, soporta múltiples protocolos que permiten tanto la recolección de datos desde sensores como el control de actuadores, así como la interacción con APIs web, brokers de mensajería y hardware embebido.

Este epígrafe presenta tres de los protocolos más utilizados en proyectos IoT: MQTT, ideal para comunicaciones ligeras y asincrónicas entre dispositivos distribuidos; HTTP, utilizado para integrar servicios RESTful y acceder a recursos web; y UART, empleado para la comunicación directa entre el sistema anfitrión y dispositivos físicos conectados mediante puertos serie. Cada protocolo tiene ventajas particulares según el escenario, y puede ser implementado fácilmente en Node-RED mediante nodos nativos o contribuidos.

4.1.1 MQTT: suscripción y publicación

- Modelo: Publicador/Suscriptor con intermediario (broker).
- Ventajas: Ligero, ideal para redes de bajo ancho de banda, confiable y escalable.
- Uso en Node-RED:

- Nodos mqtt in y mqtt out.
- Configuración de cliente y tópicos.
- Integración con brokers como Mosquitto o EMQX.

Ejemplo:
Un ESP32 publica la temperatura en el tópico casa/sala/temp, y Node-RED lo recibe para visualizarlo o generar alertas.

☑ **ATENCIÓN – ¿Por qué usar MQTT en lugar de HTTP?**

MQTT está optimizado para comunicaciones IoT: es ligero, trabaja por eventos (pub/sub), y consume menos ancho de banda que HTTP. Además, permite una arquitectura descentralizada, ideal para sensores distribuidos o escenarios con conexiones inestables (MQTT Essentials: Your 2025 Learning Hub for IoT & IIoT Data Streaming, 2026).

4.1.2 HTTP: clientes y servidores REST

- Modelo: Petición/respuesta (cliente-servidor).
- Ventajas: Estándar global, soporte para REST APIs, amplio ecosistema.
- Uso en Node-RED:
  - Nodos http request (cliente) y http in/http response (servidor).
  - Consumo de APIs externas, creación de servicios propios.

Ejemplo:
Node-RED envía una solicitud GET a una API meteorológica y muestra la temperatura actual en un dashboard.

🧠 Conclusión

Dominar estos protocolos permite que Node-RED funcione como un **centro de integración**, capaz de comunicarse con sensores, actuadores, APIs web y plataformas de automatización en tiempo real. La elección del protocolo dependerá del tipo de dispositivo, la arquitectura del sistema y las necesidades del proyecto, pero todos pueden coexistir en un flujo bien diseñado.

4.1.3 UART, I2C y SPI en Raspberry Pi

Una de las ventajas de usar Node-RED en una Raspberry Pi (Raspberry Pi Documentation, 2026) es que esta plataforma permite interactuar directamente con sensores, actuadores y periféricos físicos a través de interfaces de comunicación de bajo nivel como UART, I²C y SPI. Estas interfaces son fundamentales en proyectos de IoT, ya que permiten leer datos desde sensores analógicos/digitales o controlar dispositivos electrónicos con gran eficiencia.

En este apartado se describe cómo habilitar y utilizar estas interfaces en una Raspberry Pi, y cómo integrarlas en Node-RED para construir flujos de automatización con comunicación física directa.

- **UART (Serial)**

- **Descripción:** Comunicación punto a punto, asíncrona, bidireccional.
- **Uso típico:** Comunicación con microcontroladores, módulos GPS, lectores RFID.

Pasos para usar UART en Node-RED:

1. **Habilitar UART en Raspberry Pi:**
   - Ejecutar sudo raspi-config
   - Ir a *Interfacing Options → Serial*.
   - Desactivar la consola serial (para liberar el puerto).
   - Activar la interfaz UART.
2. **Instalar nodos serial en Node-RED:**

```
bash

CopiarEditar

cd ~/.node-red

npm install node-red-node-serialport
```

3. **Usar nodos** serial in **y** serial out**:**
   - Configurar el puerto /dev/serial0 o /dev/ttyAMA0.
   - Establecer velocidad (ej. 9600 baudios) y delimitadores (\n, \r).

Ejemplo: recibir datos de un sensor de CO_2 conectado vía UART y graficarlo en un dashboard.

⚠️ PRECAUCIÓN – UART en Raspberry Pi puede estar ocupado

Antes de usar el puerto UART (/dev/serial0) en Raspberry Pi, asegúrate de desactivar la consola serial desde raspi-config. De lo contrario, el sistema podría estar utilizando ese puerto, impidiendo que Node-RED acceda correctamente.

Desde la interfaz gráfica (Manage Palette) también podemos instalar nodos UART, como se observa en **¡Error! No se encuentra el origen de la referencia.**72.

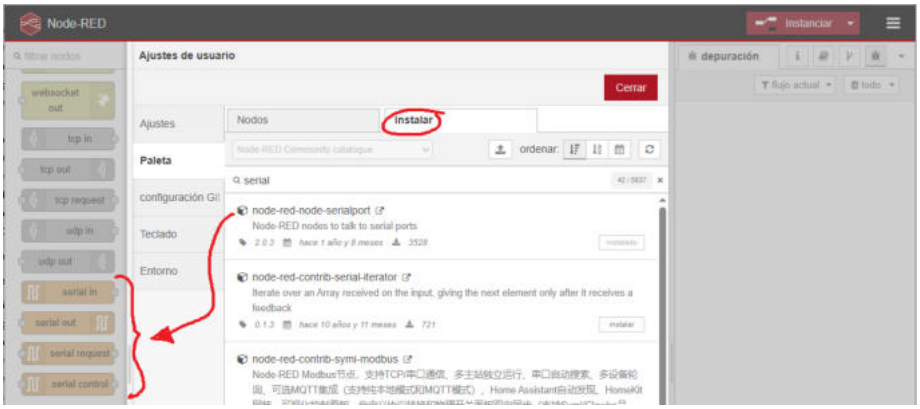


Figura 72. Nodos UART instalados desde la interfaz gráfica de Node-red.

◆ I²C (Inter-Integrated Circuit)

- **Descripción:** Bus serial síncrono, maestro-esclavo, usa dos líneas: SDA y SCL.

- **Uso típico:** Sensores de temperatura, humedad, presión, pantallas OLED.

Pasos para usar I²C en Node-RED:

1. **Habilitar I²C en Raspberry Pi:**
   - Ejecutar sudo raspi-config → *Interfacing Options* → *I2C* → Habilitar.
   - Instalar herramientas:

```
bash

CopiarEditar

sudo apt install i2c-tools

i2cdetect -y 1
```

2. **Instalar librerías según el sensor (ej. BME280):**

```
bash

CopiarEditar

npm install node-red-contrib-bme280
```

3. **Usar el nodo I²C específico:**
   - El nodo se encarga de acceder al dispositivo y publicar lecturas periódicas en el flujo.

Alternativa: usar el nodo function con una librería personalizada (requiere functionExternalModules: true).

En **¡Error! No se encuentra el origen de la referencia.**73, podemos ver cómo podemos instalar tanto la librería especifica I2C del dispositivo BME280 como un I2C genérico.

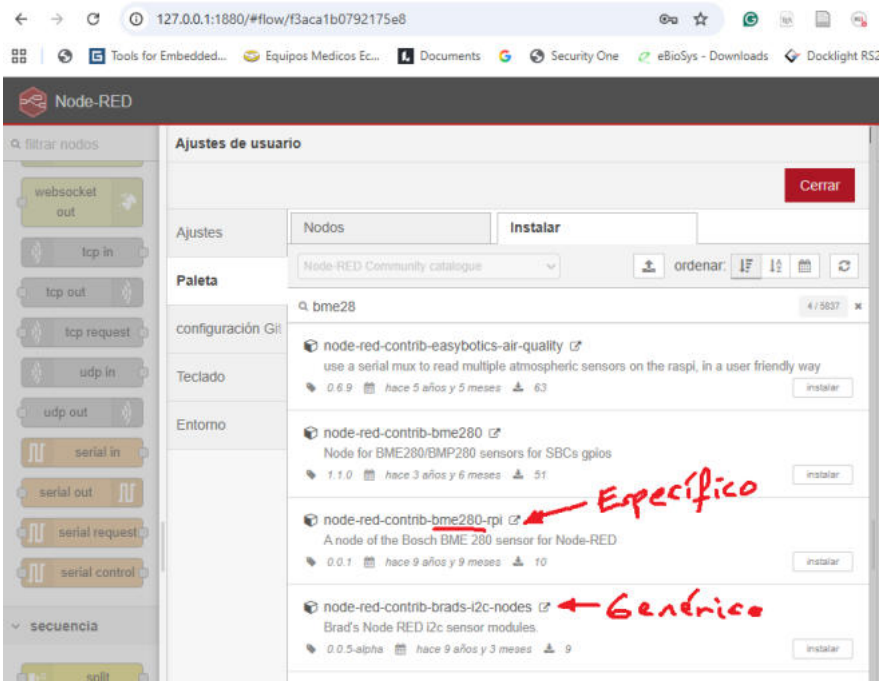


Figura 73. Nodos I2C disponibles a ser instalados, uno especifico (BME280) y otro genérico.

🛠️ ¿CÓMO SE CONFIGURA? – Crear un flujo de lectura I²C (ej. BME280)

1. Instala el nodo: npm install node-red-contrib-bme280

2. Conecta el sensor físico a SDA/SCL de la Raspberry Pi.

3. Abre Node-RED y añade el nodo BME280.

4. Configura la dirección I²C (normalmente 0x76).

5. Conecta a un nodo debug o ui_gauge para visualizar los datos.

🔹 SPI (Serial Peripheral Interface)

- **Descripción:** Bus síncrono, más rápido que I²C, ideal para transferencias de datos grandes. Utiliza 4 líneas: MISO, MOSI, SCLK y CS.
- **Uso típico:** Displays TFT, memorias flash, ADCs externos.

Pasos para usar SPI en Node-RED:

1. **Habilitar SPI:**
   - En raspi-config → *Interfacing Options* → *SPI* → Habilitar.
2. **Instalación y acceso:**
   - No existen nodos genéricos SPI en Node-RED, pero puedes usar nodos function junto con librerías Node.js como spi-device.

Ejemplo:

```
bash

CopiarEditar

npm install spi-device
```

3. Código básico en función:

```
javascript

CopiarEditar

const spi = require('spi-device');

// Crear dispositivo y realizar transferencia
```

Nota: esta técnica requiere activar el uso de módulos externos y escribir código personalizado.

Un ejemplo de instalación de nodos de forma gráfica de un dispositivo específico via SPI aparece en **¡Error! No se encuentra el origen de la referencia.**74.

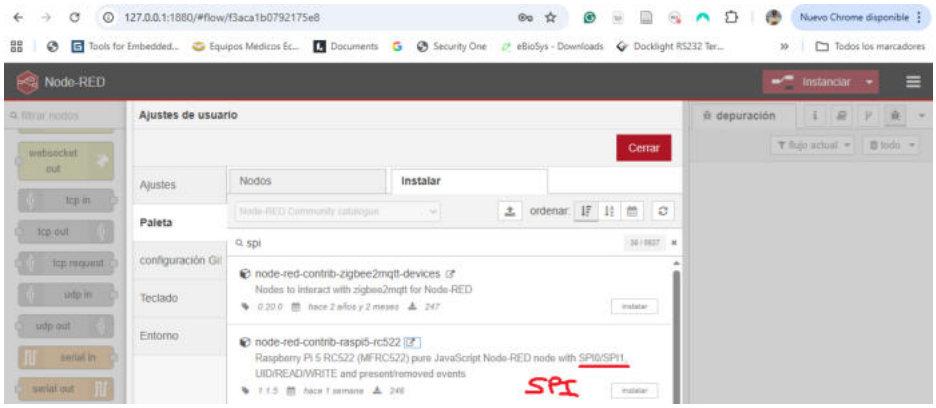


Figura 74. Nodos SPI disponibles a ser instalados desde la interfaz gráfica de Node-red.

🧠 Conclusión

Las interfaces UART, I²C y SPI permiten que Node-RED corriendo en una plataforma tipo SBC como la Raspberry Pi pueda interactuar **directamente con el mundo físico real**, sin necesidad de otros microcontroladores intermedios. Dominar estas interfaces permite construir soluciones IoT compactas, de bajo costo y eficientes, donde los datos de sensores y actuadores se integran perfectamente en flujos de automatización y visualización.

4.2 Acceso al hardware desde Node-RED

Cuando Node-RED se ejecuta en una Raspberry Pi entonces se puede interactuar directamente con el hardware físico: leer sensores, activar relés, encender LEDs o controlar motores. Esto permite cerrar el ciclo completo de un sistema IoT, desde la captura de datos hasta la acción, todo desde una interfaz visual y sin necesidad de escribir firmware para microcontroladores.

Este epígrafe presenta las principales formas de acceso al hardware desde Node-RED, principalmente el uso de la interfaz GPIO, así como el acceso a sensores digitales y analógicos mediante protocolos como I²C, SPI o UART, tal como se indicó previamente. También se incluyen las consideraciones de permisos, instalación de nodos y seguridad al trabajar con componentes físicos.

4.2.1 Control de GPIO en Raspberry Pi

Los pines GPIO de la Raspberry Pi permiten leer y escribir señales digitales, lo que permite interactuar directamente con sensores, pulsadores, relés, zumbadores, entre otros.

☑ **Pasos para usar GPIO en Node-RED:**

1. **Habilitar los pines GPIO:**
   - En Raspberry Pi OS ya vienen habilitados por defecto.
   - Asegúrate de no usar los pines reservados (alimentación, UART, etc.).
2. **Instalar el paquete de nodos GPIO:**

```
bash

CopiarEditar

cd ~/.node-red

npm install node-red-node-pi-gpio
```

3. **Reiniciar Node-RED:**

```
bash

CopiarEditar

node-red-stop

node-red-start
```

4. **Nodos disponibles:**
   - rpi-gpio in: para leer pines de entrada.

- rpi-gpio out: para escribir en pines de salida.
- rpi-mouse, rpi-keyboard: para interfaces HID (opcionales).

⚙️ **Configuración típica de un nodo** rpi-gpio out**:**

- Pin: seleccionar el número físico o BCM.
- Estado inicial: 0 (apagado) o 1 (encendido).
- El nodo recibe valores tipo 0/1 o "true"/"false" para cambiar el estado.

Ejemplo: activar un relé al recibir un valor desde MQTT.

4.2.2 Lectura de entradas digitales

Con el nodo rpi-gpio in, puedes recibir valores 0 o 1 según el estado de un botón o sensor PIR (presencia).

- Soporta detección por flanco (rising, falling, both).
- Ideal para activar flujos según eventos físicos.

Cuando necesitamos leer entradas digitales, podemos usar el nodo: rpi-gpio-in, como se aprecia en **¡Error! No se encuentra el origen de la referencia.**75, podemos incluso activar los resistores internos en el rpi, ya sea pullup o pulldown en función de la necesidad de la aplicación.

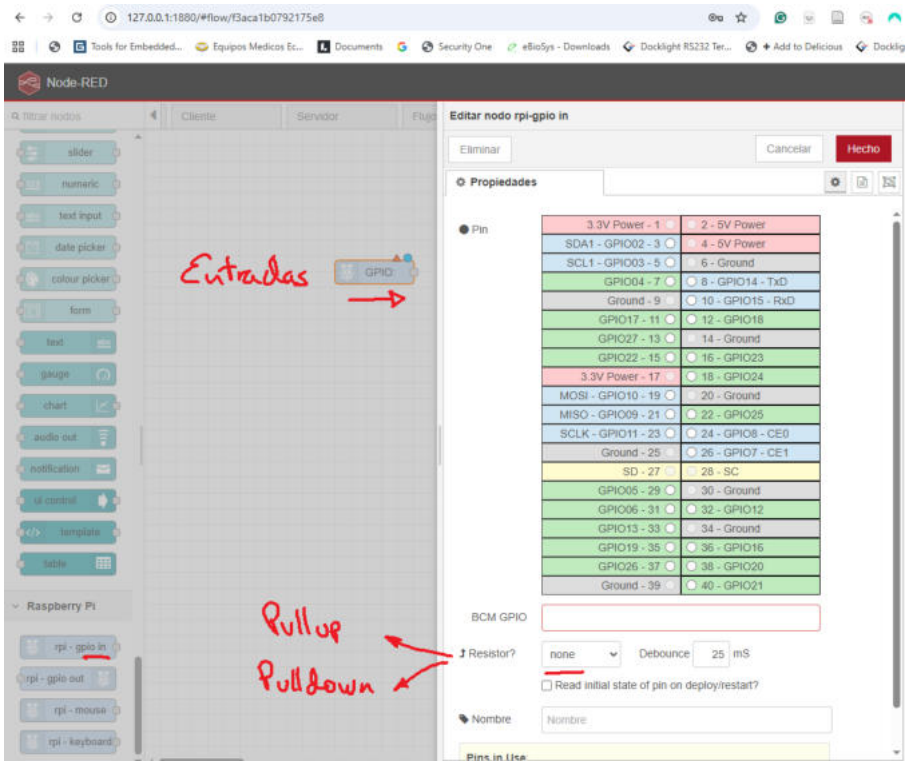


Figura 75. Nodos para gestionar GPIO como entradas digitales en una Rpi. Elaboracinón propia.

4.2.3 Escritura de salidas digitales

El nodo rpi-gpio out permite controlar actuadores simples:

- LEDs, relés, motores ON/OFF, zumbadores, etc.
- Puedes controlar el encendido con temporizadores, lógica condicional o reglas automáticas desde sensores.

En la **¡Error! No se encuentra el origen de la referencia.**76 se muestra cómo podemos seleccionar y configurar los pines de salida de un Rpi

que queremos usar en nuestra aplicación, incluso podemos seleccionar si es una salida digital (1/0) o tipo salida PWM.

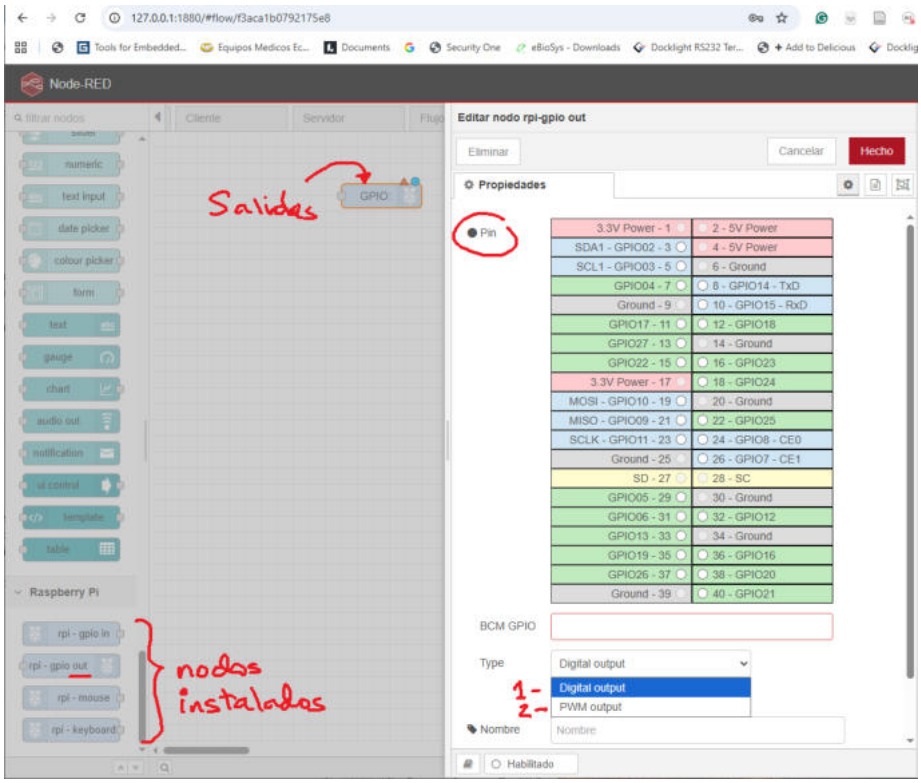


Figura 76. Nodos para gestionar GPIO como salidas digitales en una Rpi.

Consideraciones de hardware y seguridad

- **Usar resistencias de pull-up o pull-down** si el circuito lo requiere.
- **Nunca conectar dispositivos de más de 3.3V directamente** a un GPIO sin protección.
- **Evitar el uso de GPIOs sin conocer su función secundaria** (ej. SPI, I2C, UART).

- Para cargas elevadas, usar **transistores o relés optoacoplados**.

💡 **Aplicaciones típicas**

- Encender luces desde un dashboard Node-RED.
- Activar un ventilador si un sensor I²C supera un umbral.
- Registrar pulsaciones de un botón físico en un flujo MQTT.

🧠 Conclusión

Acceder al hardware directamente desde Node-RED en una Raspberry Pi permite desarrollar **sistemas embebidos completos sin programar firmware para microcontroladores**, usando flujos visuales que integran sensores, lógica de control y visualización. Esta capacidad transforma a Node-RED en una plataforma ideal para prototipado rápido, POC (pruebas de concepto), formación técnica y despliegue de soluciones IoT robustas en escenarios reales.

4.2.4 Sensores y actuadores mediante I²C y SPI

En proyectos IoT más sofisticados, donde se requiere interactuar **con múltiples sensores digitales, pantallas, conversores o actuadores inteligentes**, las interfaces **I²C (Inter-Integrated Circuit) y SPI (Serial Peripheral Interface)** son al día de hoy prácticamente obligatorias. Estas interfaces permiten comunicar de forma eficiente y a alta velocidad con dispositivos digitales externos, manteniendo una topología limpia y extendible.

Desde Node-REDpodemos acceder a estas interfaces mediante nodos específicos o librerías JavaScript en nodos function. Esto permite leer sensores de temperatura, presión, luminosidad, o incluso controlar pantallas OLED, ADCs, DACs y drivers de motores, integrándolos directamente a flujos visuales de automatización y monitoreo.

🔸 **Uso de sensores mediante I²C**

- **Ventajas del I²C:** comunicación bidireccional con múltiples dispositivos usando solo 2 líneas (SDA y SCL).
- **Ejemplos comunes:** sensores BME280, BH1750, ADS1115, pantallas SSD1306.

☑ **Pasos para usar un sensor I²C en Node-RED:**

1. **Habilitar la interfaz I²C:**

```
bash

CopiarEditar

sudo raspi-config → Interfacing Options → I2C → Enable
```

2. **Detectar dispositivos conectados:**

```
bash

CopiarEditar

sudo apt install i2c-tools
```

```
i2cdetect -y 1
```

3. **Instalar un nodo específico para el sensor:**

   - Ejemplo: sensor BME280

```
bash

CopiarEditar

cd ~/.node-red

npm install node-red-contrib-bme280
```

4. **Configurar el nodo en el editor:**

   - Dirección I²C del sensor (ej. 0x76).
   - Intervalo de lectura (ej. cada 10 segundos).

5. **Visualizar datos con nodos** debug **o** ui_chart**.**

- **Control de actuadores por I²C**

Algunos actuadores como **DACs, servocontroladores o expansores de pines (PCF8574, MCP23017)** se controlan también por I²C. Pueden utilizarse para extender la cantidad de salidas o entradas digitales de la Raspberry Pi, controladas por flujos Node-RED.

- **Uso de sensores y actuadores mediante SPI**

- **Ventajas del SPI:** mayor velocidad de transferencia que I²C, ideal para dispositivos que requieren alta frecuencia o gran volumen de datos.
- **Ejemplos comunes:** sensores de presión (MCP3008), displays TFT, memorias flash, controladores LED WS2801.

☑ **Pasos para usar SPI en Node-RED:**

1. **Habilitar la interfaz SPI:**

```
bash

CopiarEditar

sudo raspi-config → Interfacing Options → SPI → Enable
```

2. **Instalar una librería Node.js compatible con tu sensor o actuador:**

```
bash

CopiarEditar

cd ~/.node-red

npm install spi-device
```

3. **Activar** functionExternalModules **en** settings.js**:**

```
js
```

```
CopiarEditar

functionExternalModules: true
```

4. **Usar un nodo** function **con código personalizado:**

```
javascript

CopiarEditar

const spi = require('spi-device');



const device = spi.openSync(0, 0); // bus 0, chip select 0



const message = [{

  sendBuffer: Buffer.from([0x01, 0x80, 0x00]),

  receiveBuffer: Buffer.alloc(3),

  byteLength: 3,

  speedHz: 20000

}];



device.transfer(message, (err, message) => {
```

```
 if (!err) {

   let rawValue = ((message[0].receiveBuffer[1] & 3) << 8) +
       message[0].receiveBuffer[2];

   msg.payload = rawValue;

   node.send(msg);

 }

});



return null;
```

5. **Conectar el nodo** function **a** debug **o** chart**.**

🧠 **Consejos prácticos**

- Verifica siempre la dirección I²C y las conexiones físicas (SDA/SCL o MOSI/MISO).
- Utiliza resistencias pull-up para SDA y SCL si el sensor no las integra.
- SPI puede tener múltiples dispositivos mediante pines de selección (CS/CE).
- Si no existe un nodo contribuyente para tu sensor, puedes usar un function con librerías Node.js del fabricante.

☑ Conclusión

Cuando tenemos la opción de correr Node-RED sobre un SBC, entonces podemos integrar sensores y actuadores mediante I²C y SPI en nuestro flujos., lo cual amplía enormemente el alcance de los sistemas IoT desarrollados. Estas interfaces permiten una comunicación eficiente y precisa con componentes electrónicos avanzados, desde sensores ambientales hasta controladores industriales, con la facilidad de gestionarlo todo desde una interfaz visual y modular.

🧪 **MINI PRÁCTICA – Automatización combinada**

Reto: Crea un flujo que cumpla las siguientes condiciones:

- Un sensor PIR detecta movimiento (msg.payload === 1)
- La humedad es mayor a 60%
- Y la hora es entre las 18:00 y 22:00

Acción: Enciende un ventilador (por GPIO o MQTT).

Pistas:

- Usa nodos switch, current state, time range, rbe
- Agrupa las condiciones con nodos join o function

4.3 Almacenamiento y persistencia

En un sistema IoT funcional, luego de capturar datos y procesarlos en tiempo real, se requiere su almacenamiento de forma organizada y

persistente para análisis histórico, trazabilidad, visualización a largo plazo o integración con otros sistemas. Node-RED proporciona diversas formas de almacenamiento que van desde el uso de variables persistentes locales hasta bases de datos completas, tanto locales como en la nube, permitiendo adaptar la solución a necesidades simples o complejas a niel empresarial.

Este epígrafe presenta las opciones más utilizadas para almacenar datos en proyectos IoT con Node-RED: variables de contexto persistentes, archivos locales (CSV, JSON, TXT) y bases de datos como SQLite, InfluxDB (InfluxDB OSS V2 Documentation, 2026) y MySQL. Además, se explican estrategias de escritura eficiente, control de crecimiento de datos y recomendaciones sobre cuándo usar cada tipo de almacenamiento.

Variables de contexto persistente

- Almacenan datos entre ejecuciones.
- Se configuran en settings.js:

```
js

CopiarEditar

contextStorage: {

 default: "file",

 memoryOnly: { module: "memory" },
```

```
  file: { module: "localfilesystem" }

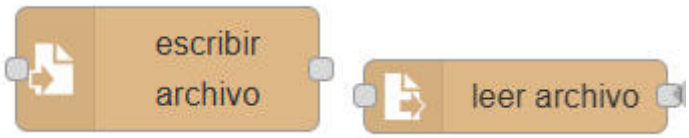
}
```

- Se accede desde nodos function con flow.set(), flow.get(), etc.
- Útiles para pequeños volúmenes: contadores, últimos valores, flags de estado.
- Recuerda, ya trabajamos antes con esto, revisa nuevamente dicho epigrafe.

Almacenamiento en archivos locales

Node-RED puede escribir en archivos locales para guardar información en formato:

- .txt → texto plano (registros simples).
- .csv → hojas de cálculo, datos tabulares.
- .json → estructuras de datos.

Nodo útil: file 

Ejemplo: guardar una lectura en datos.csv**:**

```
json

CopiarEditar
```

```
"25.3, 60, 2025-05-15 10:42:12"
```

- Permite abrir el archivo en Excel, Google Sheets, etc.
- Ideal para soluciones autónomas sin base de datos.

Un ejemplo para guardar lecturas de un sensor en un archivo.csv lo podemos encontrar en **¡Error! No se encuentra el origen de la referencia.**77.

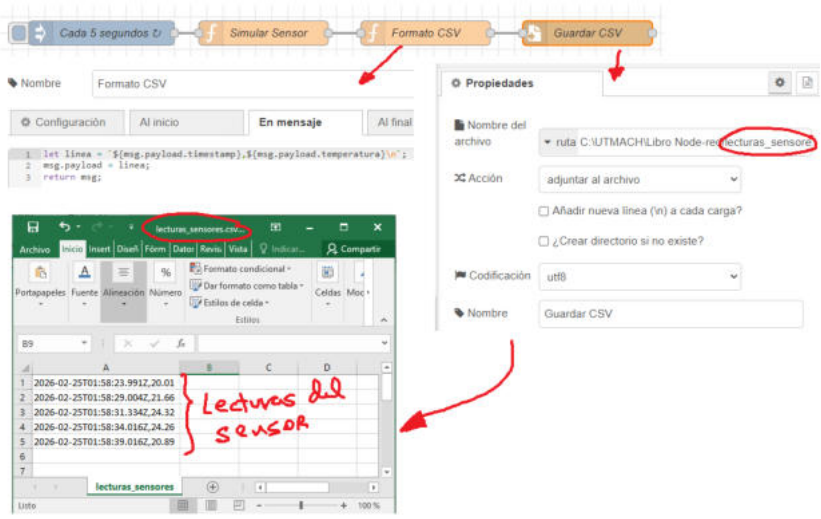


Figura 77. Gestión de datos a traves de archivos .csv.

4.3.1 Almacenamiento en bases de datos locales

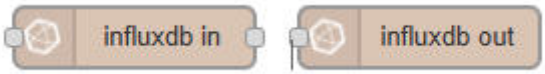
☑ **SQLite**



- Base de datos embebida, ideal para Raspberry Pi.
- No requiere servidor.

- Soporte de nodos: node-red-node-sqlite.
- Este nodo no existe, debe ser instalado a taves del Manage Palette.

Aplicación típica: registrar datos de sensores cada minuto con INSERT INTO.

☑ **InfluxDB** 

- Base de datos orientada a series temporales.
- Excelente para datos con timestamp, como temperatura, humedad, energía.
- Soporte de nodos: node-red-contrib-influxdb.
- Estos nodos no existen, deben ser instalado a taves del Manage Palette.

Ventajas:

- Muy rápida en escritura y lectura.
- Compatible con herramientas de visualización como **Grafana**.

4.3.2 Bases de datos en la nube: Firebase, MongoDB Atlas

Para proyectos distribuidos o empresariales, es útil almacenar los datos en la nube, por ejemplo:

- **Firebase** (Google): node-red-contrib-firebase
- **Google Sheets API**: almacenamiento en hojas de cálculo online.
- **MySQL en servidor cloud**: acceso remoto desde múltiples ubicaciones.
- **Amazon DynamoDB, MongoDB Atlas, Azure CosmosDB**, etc.

Ventaja: acceso remoto, integración con dashboards en web o móviles.

La **¡Error! No se encuentra el origen de la referencia.**78, muestra el conjunto de nodos que deben ser instalados para poder usar en node-red, las bases de datos de firebase, mongoDB, dynamoDb y cosmosDB respectivamente.



Figura 78. Nodos disponibles para las bases de datos: firebase, mongodb,dynamodb y cosmosdb.

🔒 Consideraciones de seguridad y rendimiento

- Validar la escritura para evitar corrupción por cortes de energía.
- Controlar el crecimiento del archivo o base de datos (rotación de archivos, borrado automático).

- Usar backups automáticos si los datos son críticos.
- Evitar escrituras excesivas en tarjetas SD: usar buffers o intervalos de escritura.

☑ Conclusión

El almacenamiento y la persistencia son pilares de cualquier arquitectura IoT confiable. Node-RED ofrece opciones simples y potentes para guardar datos localmente o en la nube, permitiendo que los sistemas no solo reaccionen en tiempo real, sino que también aprendan, se auditen y evolucionen con base en información histórica bien gestionada.

⚠ **PRECAUCIÓN – Escritura intensiva en SD puede dañarla**

Evita escribir datos en disco cada segundo. Usa un buffer (por ejemplo, escribir cada 1 minuto con delay) o considera bases de datos orientadas a series temporales como InfluxDB.

4.4 Integraciones domóticas

Node-RED es la herramienta ideal para integrar, extender y personalizar plataformas domóticas gracias a su enfoque visual, modular y orientado a flujos de datos. En el ámbito de la automatización del hogar (domótica), las integraciones más comunes se realizan con plataformas como **Home Assistant (Assistant, 2026)**, un sistema de automatización ampliamente utilizado, y **ESPHome (ESPHome, 2026)**, una solución que permite configurar microcontroladores ESP desde una página web integrada o usando configuraciones tipo yaml.

Este epígrafe muestra cómo establecer integraciones bidireccionales entre Node-RED y dichas plataformas con el objetivo de **monitorear sensores, activar actuadores, crear automatizaciones complejas y visualizar estados**, permitiendo un control centralizado y personalizado de sistemas inteligentes residenciales o institucionales.

4.4.1 Integración con Home Assistant

Home Assistant (HA) es una plataforma domótica de código abierto que permite gestionar dispositivos, sensores y automatizaciones en una única interfaz. Node-RED puede integrarse con HA de forma nativa mediante el complemento oficial o vía MQTT y APIs REST.

La **¡Error! No se encuentra el origen de la referencia.**79, muestra a node-red corriendo dentro de homeassistant. A través de node-red se puede añadir una capa de procesamiento extra, asi como acceso a los eventos generados dentro de la plataforma homeassistant, permitiendo posibles automatizaciones posteriores.

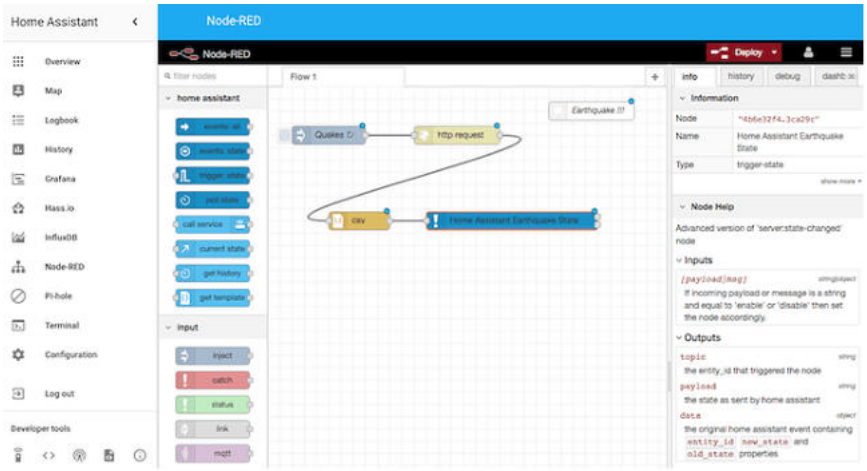


Figura 79. Integración de Node-red dentro de la plataforma domótica Home Assistant. Tomado de (Home Assistant Community Add-on: Node-RED, 2018).

☑ A. Integración directa mediante complemento (Hass.io)

1. En Home Assistant, instalar el complemento oficial:

- *Supervisor → Add-on store → Node-RED → Install*
- Iniciar el complemento y acceder vía http://<HA-IP>:1880.

2. El sistema ya incluirá nodos como:

- events: state → escucha cambios en entidades HA.
- call service → ejecuta acciones (encender luz, abrir puerta).
- current state → consulta el estado de una entidad.
- poll state, api, render template, etc.

3. Requiere autenticación mediante tokens de acceso largo (long-lived access tokens).

Ejemplo:
Detectar movimiento (binary_sensor.pir_sala) y encender una luz (light.sala) con un nodo events: state → call service.

☑ **B. Integración vía MQTT (si HA no tiene Node-RED embebido)**

1. Configurar Mosquitto como broker MQTT en Home Assistant.
2. En Node-RED:
   - Usar mqtt in y mqtt out para recibir y enviar estados/comandos.
3. En HA:
   - Crear entidades MQTT manuales o usar auto-discovery.

Ejemplo:
ESP32 publica en sala/temperatura, Node-RED recibe el dato y lo almacena o lo muestra en un dashboard.

4.4.2 Integración con ESPHome y dispositivos IoT

ESPHome permite crear firmware para dispositivos ESP32/ESP8266 sin necesidad de programar en C o C++.

☑ **Integración mediante MQTT:**

1. En esphome.yaml:

```
yaml

CopiarEditar

mqtt:

  broker: <ip-del-broker>

  topic_prefix: oficina
```

2. En Node-RED:

   - mqtt in: recibe telemetría (oficina/temp, oficina/presencia, etc.)
   - mqtt out: envía comandos (oficina/luz/set)

3. Los mensajes deben estar estructurados según el formato esperado (por ejemplo, {"state":"ON"} o 25.6 como número simple).

☑ Integración vía Home Assistant + ESPHome

Si ESPHome está conectado a HA, entonces Node-RED puede interactuar con los dispositivos ESP directamente a través de los nodos de Home Assistant (entity_id → switch.rele_01, sensor.temp_bano, etc.).

🔁 Ventajas de usar Node-RED como orquestador domótico

- Automatizaciones más complejas que las YAML de HA.
- Visualización de procesos y lógica condicional más clara.

- Integración con bases de datos, APIs web, dashboards, sistemas externos.
- Facilidad para crear lógica distribuida, por ejemplo: *si hay presencia + humedad alta + fuera del horario laboral → activar ventilación*.

⚠️ **PRECAUCIÓN – Seguridad en integraciones con Home Assistant**

Si usas Node-RED externo a Home Assistant, **nunca dejes la API de HA abierta sin autenticación**. Usa long-lived access tokens y restringe accesos con VPN o firewalls.

🧠 Conclusión

La integración conseguida entreNode-RED, Home Assistant y ESPHome permiten a estos sistemas domóticos coordinar sensores, automatizaciones, datos históricos y flujos externos con gran flexibilidad. Estas integraciones permiten diseñar soluciones escalables tanto para hogares como para oficinas, instituciones educativas o infraestructuras industriales.

4.5 Visualización e interfaces de usuario

Visualizar datos en tiempo real y controlar dispositivos desde una interfaz amigable es básico y necesario en cualquier solución IoT. Node-RED permite construir **interfaces de usuario interactivas** directamente desde un navegador, facilitando la creación de tableros de monitoreo (dashboards), paneles de control y formularios de entrada sin

necesidad de programar aplicaciones web desde cero. Es decir, construir el front-end de una aplicación web, para gestionar datos en tiempo real.

En este epígrafe exploraremos las opciones que ofrece Node-RED para crear dashboards integrados, incluyendo el uso del complemento node-red-dashboard, la personalización mediante CSS y otras opciones avanzadas para crear soluciones visuales profesionalmente. Al finalizar, el lector será capaz de desplegar **dashboards funcionales** que visualizan datos en tiempo real, reciben entradas del usuario y se adaptan a contextos móviles o de escritorio.

4.5.1 El nodo node-red-dashboard

El paquete node-red-dashboard es el método más utilizado para crear interfaces gráficas integradas en Node-RED.

☑ Instalación:

```
bash

CopiarEditar

cd ~/.node-red

npm install node-red-dashboard
```

Luego reiniciar Node-RED. El dashboard estará disponible en:

Estos paquetes no vienen incluidos en la instalación inicial, por tanto, debemos instalarlo. Aquí, debemos tener cuidado en que paquete vamos a instalar, dado que, desde el 27 de junio de 2024, el paquete oficial

node-red-dashboard tiene el estado de *deprecated* y no debe ser instalado. En su defecto, se sugiere instalar el paquete *FlowFuse Dashboard, ver* ***¡Error! No se encuentra el origen de la referencia.***80.

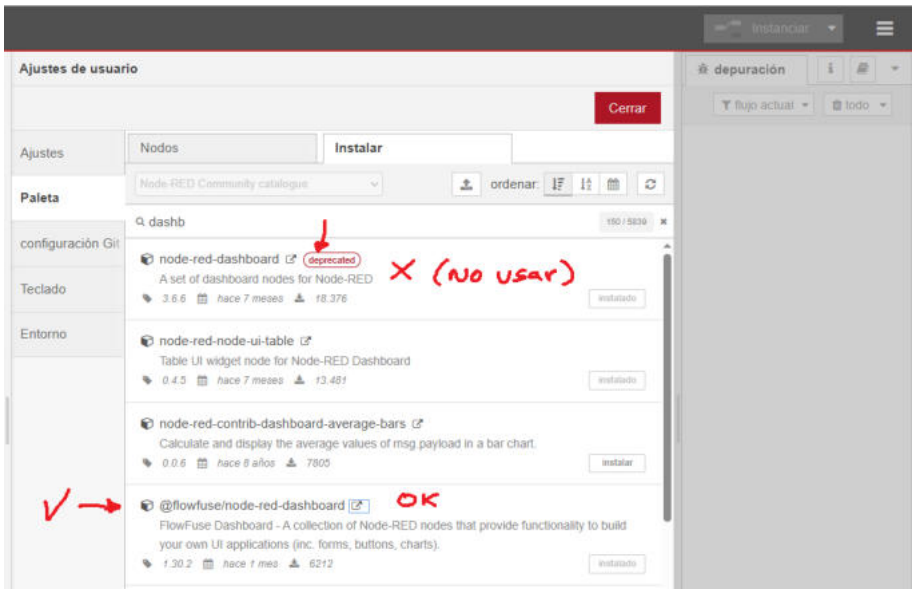


Figura 80. Opciones de instalación del paquete de Dashboard en Node-red.

☑ Componentes disponibles:

- **Visualización de datos:** ui_chart, ui_gauge, ui_text, ui_template, etc.
- **Control:** ui_button, ui_switch, ui_slider, ui_numeric, etc.
- **Organización:** ui_group, ui_theme, ui_spacer.

Cada widget se asigna a una **pestaña (tab)** y un **grupo**, lo que permite organizar visualmente el contenido. Un ejemplo de distribución de

widgets lo vemos en xxx, donde, además, aparecen los nodos que estarían disponibles con su instalación.

En la parte superior tenemos a la izquierda un chart con dos señales, en aras del ejemplo se toman los datos de dos controles: slider 1 y slider 2. Al lado derecho del chart tenemos un típico grafico de barras. Mas abajo, tenemos un gran botón denominado: Clear Chart, que permite reiniciar los valores de las sliders. A continuación, tenemos 3 widgets muy usados como son: un gauge, indicador de nivel e indicador de estado (texto y color) y por ultimo otra slider que para simular datos a mostrar.

Un buen punto de partido para trabajar con los nodos del dashboard es replicar y hacer funcionar el dashboard mostrado en **¡Error! No se encuentra el origen de la referencia.**81.



Figura 81. Front-end crado con el dashboard2.0 de Flowfuse. A la derecha se muestrasn los nodos disponibles. Tomado de (@Flowfuse/Node-red-dashboard, 2026).

Ejemplos de uso

- **Sensor de temperatura:** se recibe por MQTT → ui_gauge.
- **Control de relé:** botón ui_button envía "ON" o "OFF" por MQTT.
- **Gráfica de humedad:** ui_chart conectado a lectura periódica de un sensor.
- **Estado de red:** ui_text muestra si un dispositivo está "Conectado" o "Desconectado".

💡 TIP TÉCNICO – Visualización más limpia en dashboards

Agrupa widgets por habitación, área o tipo de dato. Usa ui_group y ui_tab para mantener tu interfaz ordenada y comprensible. Usa ui_spacer y colores personalizados para separar bloques visuales.

4.5.2 Personalización con CSS

- Usar ui_template para insertar código HTML y CSS personalizado.
- Permite aplicar estilos únicos, cambiar colores, tamaños o animaciones.

Ejemplo de botón con estilo propio:

```
html

CopiarEditar

<md-button style="background-color: #1abc9c; color: white; width: 100%;">

  Activar Ventilador

</md-button>
```

Buenas prácticas para dashboards

- Agrupar sensores y controles por zona o tipo de dispositivo.
- Usar colores para indicar estado: verde (OK), rojo (alerta), gris (desactivado).

- Adaptar la visualización a móviles usando layout responsivo.
- Evitar sobrecargar con muchos gráficos si se trabaja en Raspberry Pi.

🧠 Alternativas al dashboard oficial

Si se requieren interfaces más modernas o interactivas, se puede optar por:

- **UIBuilder:** construcción avanzada con Vue.js o Bootstrap.
- **Grafana:** dashboards profesionales para bases de datos como InfluxDB.
- **Home Assistant (Lovelace UI):** si se integra como interfaz principal.

☑ Conclusión

Node-RED ofrece una forma simple pero poderosa de crear dashboards para monitorear y controlar sistemas automatizados con arquitecturas de referencia IoT. La visualización de datos en tiempo real, combinada con controles interactivos y personalización mediante HTML/CSS, permite desarrollar soluciones domóticas e industriales funcionales, estéticamente agradables y accesibles desde cualquier dispositivo conectado a la red.

4.6 Manejo de formatos de datos

En un entorno IoT real, los datos no siempre vienen en un formato único o uniforme. Los dispositivos pueden enviar información en texto plano, estructuras JSON, archivos CSV o incluso HTML y YAML, dependiendo de su diseño o de la API utilizada. Por ello, una habilidad esencial para trabajar con Node-RED es saber **identificar, interpretar, transformar y generar datos en diversos formatos**, adaptándolos al flujo que requiera la solución.

En este epígrafe veremos los formatos más utilizados en IoT y cómo gestionarlos eficientemente dentro de Node-RED, utilizando nodos básicos como *change*, *template*, *function* y nodos externos especializados. Además, se explicarán las técnicas para la conversión de tipos, estructuración de mensajes, y manipulación de contenido en flujos dinámicos.

4.6.1 JSON (JavaScript Object Notation)

El formato más común en IoT.

Se usa en APIs, MQTT, dashboards, y almacenamiento.

- En Node-RED, msg.payload suele contener un objeto JSON.
- Para convertir un string JSON a objeto:

```
javascript

CopiarEditar
```

```
msg.payload = JSON.parse(msg.payload);

return msg;
```

- Para convertir objeto a string JSON:

```
javascript

CopiarEditar

msg.payload = JSON.stringify(msg.payload);

return msg;
```

Uso práctico: recibir { "temp": 24.6, "hum": 60 } y acceder a msg.payload.temp.

4.6.2 CSV (Comma-Separated Values)

Formato tabular muy usado en registros históricos o exportación a Excel.

- Nodo recomendado: csv
  - Convierte una fila CSV a objeto JSON y viceversa.

Ejemplo:
Entrada:

CopiarEditar

25.4,60

→

```
json

CopiarEditar

{ "column1": 25.4, "column2": 60 }
```

Uso típico: almacenamiento local o carga masiva de datos en dashboards.

🛠️ **¿CÓMO SE CONFIGURA? – Guardar datos en archivo CSV**

1. Agrega un nodo csv (modo *from JSON to CSV*).
2. Conecta un nodo file y configura la ruta (/home/pi/lecturas.csv).
3. Activa “Append to file”.
4. Asegúrate de que el JSON tenga campos simples (sin anidar).

4.6.3 Texto plano (string)

Usado en sensores simples o comandos seriales (UART, HTTP, etc.).

- Nodo function o template para formatear o extraer datos.
- Uso de expresiones regulares (regex) para separar contenido.

Ejemplo:

```
arduino

CopiarEditar

"Temp=25.4;Hum=60"
```

Se puede dividir con:

```
javascript

CopiarEditar

let partes = msg.payload.split(";");

let temp = partes[0].split("=")[1];

msg.payload = { temp: parseFloat(temp) };

return msg;
```

4.6.4 YAML (Yet Another Markup Language)

- Formato legible por humanos, usado en configuraciones (ej. ESPHome, Home Assistant). (Npm: Js-yaml, 2026)
- Para manipular YAML en Node-RED se puede usar:

```
bash

CopiarEditar

npm install js-yaml
```

Y en un nodo function:

```
javascript

CopiarEditar

const yaml = require("js-yaml");

msg.payload = yaml.load(msg.payload); // YAML a objeto

return msg;
```

4.6.5 HTML (HyperText Markup Language)

- Se usa para generar reportes, dashboards personalizados o enviar correos con estilo.
- Nodo template permite insertar HTML en el msg.payload.

Ejemplo:

```
html

CopiarEditar

<b>Temperatura:</b> {{payload.temp}} °C
```

- Útil al combinar con nodos email, ui_template, etc.

4.6.6 XML (Extensible Markup Language)

- Menos común en nuevos proyectos IoT, pero aún utilizado en sistemas industriales o heredados.

- Usar nodo xml para convertir entre XML ↔ JSON.

Ejemplo:

```
xml

CopiarEditar

<sensor><temp>25.6</temp></sensor>

→

json

CopiarEditar

{ "sensor": { "temp": 25.6 } }
```

☑ Buenas prácticas

- Siempre validar el tipo de dato recibido antes de operar sobre él.
- Convertir a objetos JSON lo antes posible para trabajar más cómodamente en Node-RED.
- Mantener formatos homogéneos dentro del flujo para facilitar trazabilidad y depuración.
- Documentar la estructura esperada del mensaje con nodos comment.

🧠 Conclusión

El manejo de formatos de datos es vital para conseguir **interoperabilidad, visualización y persistencia** de la información en sistemas IoT. Node-RED, , permite convertir, analizar y estructurar múltiples formatos sin complejidad, facilitando la integración con sensores, servicios, APIs y bases de datos.

4.6.7 Estructuras mixtas

En muchas aplicaciones IoT, los datos no siempre llegan en un formato limpio y plano. Es común encontrar **estructuras mixtas** que combinan texto, JSON anidado, arreglos, listas CSV dentro de campos o incluso HTML embebido en propiedades. Estas estructuras complejas requieren técnicas más refinadas de análisis, transformación y validación para que puedan ser procesadas adecuadamente dentro de Node-RED.

En este subepígrafe aprenderemos a trabajar con mensajes que contienen **múltiples tipos de datos al mismo tiempo**, a extraer información de niveles anidados, y cómo estructurar mensajes híbridos de forma controlada para luego poder almacenarlos, visualizarlos o enviarlos a otras plataformas.

🧩 **Ejemplo 1: JSON con campos en formato CSV**

Entrada:

```
json

CopiarEditar
```

```
{
  "device": "esp32_01",
  "data": "25.4,60,1012",
  "timestamp": "2025-05-15T10:30:00Z"
}
```

Conversión con function**:**

```
javascript
CopiarEditar
let valores = msg.payload.data.split(",");
msg.payload = {
  device: msg.payload.device,
  temp: parseFloat(valores[0]),
  hum: parseFloat(valores[1]),
  presion: parseFloat(valores[2]),
  timestamp: msg.payload.timestamp
};
return msg;
```

🧩 Ejemplo 2: JSON anidado con arreglo interno

Entrada:

```
json

CopiarEditar

{

 "room": "lab1",

 "sensors": [

   { "type": "temp", "value": 24.3 },

   { "type": "hum", "value": 58 },

   { "type": "co2", "value": 620 }

 ]

}
```

Extraer un valor específico:

```
javascript

CopiarEditar

let sensores = msg.payload.sensors;

let temp = sensores.find(s => s.type === "temp");
```

```
msg.payload = temp.value;

return msg;
```

🧩 Ejemplo 3: HTML embebido en campo JSON

Entrada:

```
json

CopiarEditar

{

 "device": "esp01",

 "status_html": "<div style='color:red'>ALERTA</div>"

}
```

Uso práctico:

- Renderizar directamente en un ui_template.
- Evitar procesamiento adicional.

🧩 Ejemplo 4: JSON dentro de un string

Entrada:

```
json

CopiarEditar
```

```
{
  "payload": "{\"temp\":22.4,\"hum\":55}"
}
```

Conversión:

```
javascript
CopiarEditar
msg.payload = JSON.parse(msg.payload);
return msg;
```

☑ Buenas prácticas al trabajar con estructuras mixtas

- **Validar siempre el tipo de dato** con typeof, Array.isArray(), o try/catch.
- **Documentar la estructura esperada** (ej. con comentarios o ejemplos).
- Usar debug para inspeccionar niveles anidados.
- Si el flujo lo permite, **normalizar los datos** lo antes posible para trabajar con estructuras limpias.

🧠 Conclusión

Las estructuras mixtas son frecuentes en proyectos IoT reales, especialmente cuando se integran múltiples dispositivos, APIs y fuentes de datos heterogéneas. Node-RED las herramientas necesarias para descomponer, transformar y reorganizar estos mensajes complejos, permitiendo a su vez, mantener la coherencia en el procesamiento y asegurando que los datos fluyan correctamente hacia bases de datos, dashboards o lógicas de control.

💡 **TIP TÉCNICO – Convertir estructuras mixtas a JSON plano**

Usa un nodo *function* para extraer y transformar campos complejos, como arrays o texto separado por;. Luego conviértelo en un objeto limpio para almacenar o enviar.

4.7 Validación de tipo de dato en Node-RED

☑ **1. Verificar si** msg.payload **es un número**

```
javascript

CopiarEditar

if (typeof msg.payload === "number") {

    node.status({fill:"green",shape:"dot",text:"Número válido"});

    return msg;

} else {

    node.error("El payload no es un número", msg);
```

```
    return null;

}
```

☑ 2. Validar si es una cadena de texto (string)

```
javascript

CopiarEditar

if (typeof msg.payload === "string") {

    msg.payload = msg.payload.trim();

    return msg;

} else {

    node.error("Se esperaba un string");

    return null;

}
```

☑ 3. Validar si es un arreglo (array)

```
javascript

CopiarEditar

if (Array.isArray(msg.payload)) {

    msg.payload = msg.payload.length;  // Ejemplo: contar elementos
```

```
    return msg;

} else {

    node.error("El payload no es un array");

    return null;

}
```

☑ **4. Verificar si es un objeto JSON válido**

```
javascript

CopiarEditar

if (typeof msg.payload === "object" && !Array.isArray(msg.payload)
&& msg.payload !== null) {

    return msg;

} else {

    node.error("El payload no es un objeto JSON");

    return null;

}
```

☑ **5. Verificar si un campo específico existe y tiene el tipo correcto**

```
javascript

CopiarEditar

if (msg.payload && typeof msg.payload.temp === "number") {

    msg.payload.estado = (msg.payload.temp > 30) ? "Alto" : "Normal";

    return msg;

} else {

    node.error("Campo temp ausente o no numérico");

    return null;

}
```

☑ 6. Detectar si un string puede convertirse a número

```
javascript

CopiarEditar

let valor = Number(msg.payload);

if (!isNaN(valor)) {

    msg.payload = valor;

    return msg;

} else {
```

```
    node.error("El valor no es numérico");

    return null;

}
```

🧠 Buenas prácticas

- Siempre validar antes de operar sobre un dato.
- Combinar typeof, Array.isArray() y validaciones de nulos (!== null).
- Mostrar errores en el panel lateral con node.error().
- Usar try/catch si el flujo puede lanzar excepciones (ej. en JSON.parse()).

4.7.1 Validación de null e undefined en Node-RED

☑ **1. Validar si** msg.payload **es distinto de** null **y** undefined

```
javascript

CopiarEditar

if (msg.payload !== null && msg.payload !== undefined) {

    return msg;

} else {

    node.error("msg.payload es null o undefined");
```

```
    return null;

}
```

Esto es la forma más segura para asegurar que el valor existe.

☑ 2. Validación compacta con doble negación (!!)

```
javascript

CopiarEditar

if (!!msg.payload) {

    // Continúa el flujo

    return msg;

} else {

    node.warn("Payload vacío o no definido");

    return null;

}
```

Este método excluye null, undefined, false, 0, "" y NaN.

☑ 3. Validar si una propiedad específica es null

```
javascript

CopiarEditar
```

```
if (msg.payload && msg.payload.temperatura !== null) {
    return msg;
} else {
    node.error("Temperatura no presente o nula");
    return null;
}
```

☑ **4. Uso de operador** typeof **para evitar errores de acceso**

```
javascript
CopiarEditar
if (typeof msg.payload === "object" && msg.payload !== null) {
    // El payload es un objeto válido
    return msg;
} else {
    node.error("El payload no es un objeto");
    return null;
}
```

☑ **5. Validar si un campo está definido antes de usarlo**

```
javascript

CopiarEditar

if (msg.payload && typeof msg.payload.valor !== "undefined") {

    return msg;

} else {

    node.error("Campo 'valor' no definido");

    return null;

}
```

🧠 Recomendaciones

- Siempre verifica existencia antes de acceder a propiedades profundas: if (msg && msg.payload && msg.payload.valor)
- Usa !== null && !== undefined para validaciones estrictas.
- Emplea try/catch si accedes a campos que pueden no existir.
- En la Tabla 1 hay un resumen de las validaciones más usadas.

Tabla 1. Validación de valores en Node-RED (function).

Tipo de valor	Descripción	Ejemplo de validación	Evalúa como true	Notas importantes

null	Valor nulo explícito	msg.payload !== null	No	typeof null === "object" por compatibilidad histórica
undefined	Variable no inicializada o campo ausente	typeof msg.payload !== "undefined"	No	No lances error directo al acceder si puede ser undefined
NaN	Resultado inválido de una operación numérica	!isNaN(msg.payload)	No	typeof NaN === "number"
Cadena vacía ("")	Texto sin contenido	msg.payload !== ""	No	!!"" es false
Cero (0)	Valor numérico válido	typeof msg.payload === "number"	No con !!	Úsalo con cuidado si 0 es un valor válido
Boolean o false	Valor booleano válido	msg.payload === false	No con !!	Diferenciar entre booleano falso y ausencia de valor
Objeto vacío {}	Objeto sin propiedades	Object.keys(msg.payload).length > 0	Sí con !!, pero puede estar vacío	
Array vacío []	Lista sin elementos	Array.isArray(msg.payload) && msg.payload.length > 0	Sí con !!, pero debe validarse contenido	
String JSON (como texto)	Texto que representa JSON válido	try { JSON.parse(msg.payload); } catch (e) { ... }	Sí con !! si no está vacío, requiere parsing	

☑ Recomendación general para validar datos en Node-RED:

```
javascript

CopiarEditar

if (

  msg.payload !== null &&

  msg.payload !== undefined &&

  msg.payload !== "" &&

  !Number.isNaN(msg.payload)

) {

  // El dato es válido

  return msg;

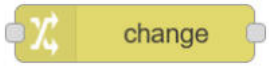
} else {

  node.error("Valor no válido en payload", msg);

  return null;

}
```

4.8 Validaciones con nodos visuales

1. Nodo change 

- Descripción: Permite modificar o verificar el valor de los mensajes. Puedes usarlo para comprobar si un valor existe, si es igual a un valor específico, o incluso para hacer asignaciones y validaciones condicionales.

- Ejemplo de uso:

  - Operación de comparación: Puedes utilizar el nodo change para verificar si el valor de un mensaje es igual a un determinado valor o no.

  - Validación de null o undefined: Puedes usar el nodo change para asegurarte de que un mensaje tiene un valor definido antes de proceder a la siguiente acción.

- Configuración de ejemplo:

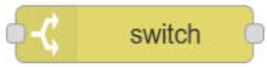
  - Si el payload no es igual a null, puedes cambiar el valor para permitir que el flujo continúe.

  - O bien, puedes bloquear un mensaje si no cumple con los criterios definidos.

2. Nodo rbe (Report by Exception) o filter 

- Descripción: Este nodo es útil para validar cambios de datos. rbe solo pasa los mensajes cuando el valor cambia, lo cual es ideal para evitar la sobrecarga de datos innecesarios y asegura que los valores sean validados antes de ser transmitidos.

- Ejemplo de uso: Si un sensor está generando lecturas continuas, pero solo te interesa si los valores cambian, este nodo ayudará a filtrar datos repetidos y enviar solo los mensajes que representan una actualización válida.

3. Nodo switch 

- Descripción: El nodo switch permite realizar comparaciones más avanzadas sobre el valor de un mensaje. Puedes configurar diferentes condiciones lógicas, como "es igual a", "es mayor que", "es menor que", entre otras, y redirigir el flujo según si se cumple o no la condición.

- Ejemplo de uso: Validar si el valor de un mensaje es un número mayor que cero y pasar el mensaje solo si la condición se cumple.

- Configuración de ejemplo:

  - Si el valor de msg.payload es mayor que 0, pasa el flujo; si no, redirige el flujo a una ruta de error.

4. Nodo json 

- Descripción: Este nodo permite validar que un mensaje esté en formato JSON. Si el contenido no es un JSON válido, el flujo no pasará el mensaje, lo que actúa como una validación de datos.

- Ejemplo de uso: Si recibes datos de una API externa en formato JSON y necesitas verificar si el JSON es válido antes de procesarlo, el nodo json se asegura de que el formato sea correcto.

5. Nodo template 

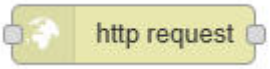
- Descripción: El nodo template puede ser utilizado para validar y modificar cadenas de texto. Es útil cuando trabajas con mensajes que contienen texto o datos estructurados que deben seguir un formato específico.

- Ejemplo de uso: Si necesitas que un mensaje contenga un valor específico (como una fecha con formato específico o una estructura JSON determinada), puedes usar este nodo para validar y cambiar su estructura.

6. Nodo delay 

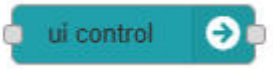
- Descripción: Aunque su función principal es introducir una demora, el nodo delay puede ser útil para controlar la cantidad de mensajes que pasan por un flujo, lo cual indirectamente actúa como una validación de tiempo.

- Ejemplo de uso: Si se necesitan validar solo las entradas dentro de un intervalo de tiempo, puedes usar este nodo para filtrar las entradas que no cumplan con el tiempo esperado.

7. Nodo http request 

- Descripción: Si deseas validar la respuesta de una solicitud HTTP (por ejemplo, de una API externa), puedes usar el nodo http request para realizar la solicitud y luego usar un nodo switch o function para validar el estado de la respuesta.

- Ejemplo de uso: Validar que el código de estado de la respuesta HTTP sea 200 (OK) antes de continuar con el flujo.

8. Nodo ui_control (para dashboards) 

- Descripción: Si estás trabajando con un dashboard, este nodo permite enviar mensajes para modificar el estado o valor de un control en la interfaz de usuario. Se puede usar también para validar interacciones del usuario en el dashboard.

- Ejemplo de uso: Validar si un botón en un dashboard se presiona y ejecutar acciones en base a esta validación.

¿Cuándo usar cada uno?

- change y switch: son útiles para validar de manera sencilla si los datos cumplen con ciertos criterios sin tener que escribir mucho código.

- Rbe: es ideal para datos que solo necesitas si han cambiado.

- Json: es perfecto para validar el formato de datos que vienen en JSON.
- Template: es útil cuando necesitas manipular cadenas y asegurar que los datos sigan un patrón o formato.
- http request: es esencial cuando trabajas con APIs y necesitas validar respuestas.

Elementos de cierre

Resumen del capítulo:

En este capítulo aprendiste a interactuar con el mundo físico usando Node-RED. Desde el uso de protocolos como MQTT o HTTP hasta la conexión con sensores mediante GPIO, I²C o SPI, has explorado cómo construir flujos funcionales que recolectan datos, los procesan y los muestran en dashboards. Además, integraste plataformas de automatización domótica y aprendiste a trabajar con distintos formatos de datos, incluyendo estructuras complejas y mixtas.

Preguntas de revisión

❓ **Pregunta 1:** ¿Qué protocolo es más adecuado para transmitir datos desde sensores distribuidos a un servidor central en tiempo real?

✍️ a) HTTP

✍️ b) MQTT

✍️ c) UART

✍️ d) SPI

☑ **Respuesta correcta:** b) MQTT

❓ **Pregunta 2:** ¿Cuál de las siguientes opciones corresponde a una interfaz síncrona y de alta velocidad?

✍ a) UART

✍ b) I²C

✍ c) SPI

✍ d) GPIO

☑ **Respuesta correcta:** c) SPI

❓ **Pregunta 3:** ¿Qué nodo de Node-RED permite visualizar una gráfica en tiempo real en un dashboard?

✍ a) debug

✍ b) ui_chart

✍ c) ui_switch

✍ d) template

☑ **Respuesta correcta:** b) ui_chart

❓ **Pregunta 4:** ¿Cuál es el nodo recomendado para convertir un string CSV en un objeto JSON?

✍ a) function

✍ b) json

✍ c) csv

✍ d) change

☑ **Respuesta correcta:** c) csv

Clave de respuestas (resumen)

Pregunta	Respuesta correcta
1	b) MQTT
2	c) SPI
3	b) ui_chart
4	c) csv

BIBLIOGRAFÍA

CAPÍTULO 1

Hernández Rojas, D. L. (2025). Fundamentos y aplicaciones del Internet de las cosas: arquitectura, comunicación e implementación práctica. Editorial Grupo Compás. http://142.93.18.15:8080/jspui/handle/123456789/1281

Getting started : Node-RED. (2026). https://nodered.org/docs/getting-started/

Node.js — Run JavaScript everywhere. (2026). https://nodejs.org/

Bcryptjs. (2026). https://www.npmjs.com/package/bcryptjs

Raspberry Pi. (2026). Raspberry Pi. https://www.raspberrypi.com/

CAPÍTULO 2

The core nodes : Node-RED. (2026). https://nodered.org/docs/user-guide/nodes

Node-RED Cookbook : Node-RED. (2026). https://cookbook.nodered.org/

Writing Functions : Node-RED. (2026). https://nodered.org/docs/user-guide/writing-functions

W3Schools.com. (2026). https://www.w3schools.com/jsref/

CAPÍTULO 3

GitHub.com Documentación de ayuda. (2026). GitHub Docs. https://docs.github.com/es

JavaScript reference - JavaScript | MDN. (2025, July 29). https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference

Creating your first node : Node-RED. (2026). https://nodered.org/docs/creating-nodes/first-node

CAPÍTULO 4

npm: js-yaml. (2026). Npm. https://www.npmjs.com/package/js-yaml

@flowfuse/node-red-dashboard. (2026). https://flows.nodered.org/node/@flowfuse/node-red-dashboard

MQTT Essentials: your 2025 learning hub for IoT & IIoT data Streaming. (2026). https://www.hivemq.com/mqtt/

ESPHome. (2026). ESPHome - Smart Home Made Simple. https://esphome.io/

Assistant, H. (2026). Documentation. Home Assistant. https://www.home-assistant.io/docs/

InfluxDB OSS v2 Documentation. (2026). InfluxData Inc. https://docs.influxdata.com/influxdb/v2/

Raspberry Pi Documentation. (2026). https://www.raspberrypi.com/documentation/

GLOSARIO

Término	Definición	Capítulo
API (Interfaz de programación de aplicaciones)	Conjunto de funciones que permite la comunicación entre sistemas de software, frecuentemente usadas para consumir datos desde servicios web.	4
Broker MQTT	Servidor que gestiona la mensajería entre dispositivos usando el protocolo MQTT bajo el modelo publicador/suscriptor.	4.1
Contexto (flow/global)	Espacios de almacenamiento de datos en Node-RED que permiten compartir información entre nodos o flujos.	3.1
CSV	Formato de archivo de texto plano usado para representar datos tabulados, separado por comas o punto y coma.	4.6
Dashboard	Interfaz gráfica web para visualización y control de flujos, sensores y automatizaciones en tiempo real.	4.5
ESPHome	Plataforma que permite configurar microcontroladores ESP8266/ESP32 sin programar, integrándolos fácilmente con Home Assistant y Node-RED.	4.4
Flujo (Flow)	Conjunto de nodos conectados entre sí que ejecutan una lógica o tarea específica dentro del entorno visual de Node-RED.	1.3, 2
Function	Nodo que permite escribir código JavaScript para realizar operaciones lógicas complejas dentro de un flujo.	2.2, 3.3
GPIO (General Purpose Input/Output)	Puntos de entrada/salida digital programables en dispositivos como Raspberry Pi para controlar hardware físico.	4.2
HTTP	Protocolo de comunicación basado en solicitudes/respuestas, usado comúnmente para integrar Node-RED con APIs REST.	4.1
I²C (Inter-Integrated Circuit)	Protocolo de comunicación síncrono y en bus usado para conectar sensores y dispositivos digitales con dos líneas (SDA y SCL).	4.1.3, 4.2
Inject	Nodo que genera mensajes manuales o programados para iniciar flujos o simular entradas.	2.1
JSON (JavaScript Object Notation)	Formato de intercambio de datos liviano, basado en estructuras clave-valor, ampliamente usado en flujos de IoT.	2.3, 4.6

Término	Definición	Capítulo
Librería externa	Módulo adicional de Node.js que puede ser importado en nodos function para extender funcionalidades (ej. moment, mathjs).	3.3.3
MQTT	Protocolo ligero de mensajería basado en publicador/suscriptor, ideal para dispositivos IoT y redes de bajo consumo.	4.1
Node.js	Entorno de ejecución JavaScript sobre el cual se ejecuta Node-RED. Permite acceso a librerías, servidores y archivos del sistema.	1.1
Node (Nodo)	Unidad funcional dentro de un flujo de Node-RED. Puede representar una entrada, procesamiento, condición o salida.	1.3, 2.1
Paleta (Palette)	Conjunto de nodos disponibles en el editor visual de Node-RED, los cuales pueden ser instalados, eliminados o organizados.	1.4
Payload	Principal campo de datos de un mensaje en Node-RED (msg.payload). Generalmente contiene el valor a procesar o enviar.	2.3
Plantilla (Template)	Nodo que permite generar contenido con HTML o texto personalizado usando valores dinámicos (ej. {{payload}}).	2.1, 4.6
Proyecto	Colección versionada de flujos, configuraciones y dependencias de Node-RED gestionada mediante Git.	3.2
Raspberry Pi	Microcomputadora de bajo costo y alto uso en IoT, ideal para ejecutar Node-RED en proyectos embebidos.	1.2, 4.2
Security (Seguridad)	Conjunto de prácticas y configuraciones en Node-RED que protegen el acceso a flujos, APIs, editores y datos sensibles.	1.5, 3.5
Serial/UART	Protocolo de comunicación por puerto serie entre dispositivos electrónicos. Utilizado para conectar sensores o microcontroladores.	4.1.3
SPI (Serial Peripheral Interface)	Protocolo de comunicación síncrono y de alta velocidad, ideal para dispositivos digitales que requieren transferencia de datos rápida.	4.1.3, 4.2
Subflujo	Flujo encapsulado y reutilizable que actúa como un nodo personalizado dentro de otros flujos.	2.4
Switch	Nodo que evalúa condiciones y dirige los mensajes a diferentes rutas según valores de entrada (msg.payload).	2.2

Término	Definición	Capítulo
Template HTML	Uso del nodo template para crear estructuras visuales o mensajes HTML personalizados para dashboards, correos, etc.	4.5, 4.6
Topic	Campo opcional en msg usado comúnmente en MQTT para categorizar mensajes y suscribirse a canales específicos.	2.3, 4.1
UI (Interfaz de usuario)	Conjunto de widgets y visualizaciones gráficas creadas con node-red-dashboard para mostrar datos y permitir interacción del usuario.	4.5
Variable de contexto	Variables que se almacenan a nivel de nodo (context), flujo (flow) o global (global) para compartir o guardar datos temporalmente o de forma persistente.	3.1
YAML	Formato de datos estructurado, usado frecuentemente en configuraciones de ESPHome y Home Assistant.	4.6

Introducción a la programación visual con Node-RED en IoT, se publicó en el mes de noviembre de 2026.

ISBN: 978-9907-802-67-2

Grupo Editorial BLR
Ecuador
Cel: +593 98 320 4362
https://grupoblr.com/
publicaciones@grupoblr.com

BIOGRAFÍA DEL AUTOR

Dixys Leonardo Hernández Rojas:
Ingeniero electrónico y Máster en Electrónica por la Universidad Central de las Villas - Cuba. Obtuvo su PhD en Tecnología de la Información de las Comunicaciones Radio Móviles (TIC-RM) interuniversitario, otorgado por 6 universidades de España, Universidad de la Coruña, Zaragoza, País vasco, entre otras. Con 33 años de experiencia laboral, ha trabajado en la docencia universitaria de pregrado y postgrado y en la industria en varios países como Cuba, México, USA, Colombia, Chile y Ecuador, desarrollando proyectos y productos con sistemas embebidos full stack, telecomunicaciones y automatización industrial. Su línea de investigación actual es el Internet de las Cosas, Redes de sensores inteligentes, Redes inalámbricas con BLE, Zigbee, LoRA, Domótica, Realidad aumentada, Blockchain y Seguridad IoT. Ha dictado conferencias en numerosos congresos nacionales e internacionales en varios países. Ha publicado varios libros, ponencias y artículos científicos, muchos de ellos en revistas de alto impacto. Ha dirigido varios proyectos de investigación, fundó el Grupo IotMach y actualmente dirige el grupo de investigación AutoMathTIC y el semillero estudiantil Redes de Sensores inalámbricos de la UTMACH, dictando además la materia de Internet de las cosas en la carrera de pregrado de Tecnología de la Información en la UTMACH desde el 2021 y a nivel de postgrado de Maestrias desde el 2025.

INTRODUCCIÓN A LA PROGRAMACIÓN VISUAL CON NODE-RED EN IOT

Estimado lector, esta obra técnico-pedagógica aborda los retos de interoperabilidad, integración y despliegue del Internet de las Cosas (IoT) en el marco de la actual convergencia entre conectividad global e inteligencia artificial. El texto examina cómo la heterogeneidad de fabricantes, la dispersión de estándares de comunicación y la exigencia del procesamiento de datos en tiempo real elevan la complejidad del desarrollo tradicional de software, fundamentando en la programación visual basada en flujos una alternativa estratégica para reducir la curva de aprendizaje sin perder robustez funcional ni capacidad de escalamiento industrial.

Centrado en la plataforma de código abierto Node-RED, el volumen desglosa su arquitectura sobre el entorno Node.js y su versatilidad para operar como motor de orquestación en hardware diverso, desde microcomputadores como Raspberry Pi hasta servidores y entornos virtualizados en la nube. A través de una metodología didáctica orientada a la resolución de problemas, la obra sistematiza la interconexión entre las distintas capas del IoT: captura de variables mediante sensores (capa de percepción), intermediación por protocolos como MQTT y HTTP (capa de red/transporte), y procesamiento lógico, persistencia en bases de datos y despliegue de tableros de control interactivos (capa de aplicación). Asimismo, profundiza en el uso de nodos de enrutamiento y transformación de datos —tales como Switch, Change, Join y Function— para implementar automatizaciones complejas y pruebas de concepto rápidas.

En su conjunto, el libro se consolida como una guía metodológica y formativa indispensable tanto para docentes y estudiantes de ingeniería electrónica, telecomunicaciones, sistemas embebidos y automatización, como para desarrolladores y entusiastas de la comunidad maker. Su enfoque modular y aplicado dota al lector de competencias directas para diseñar, prototipar y poner en marcha soluciones IoT eficientes y seguras frente a las demandas de la industria contemporánea.

GRUPO BLR

Grupo Editorial BLR
Ecuador
Cel: +593 98 320 4362
https://grupoblr.com/
publicaciones@grupoblr.com

ISBN: 978-9907-802-67-2

UNIVERSIDAD TÉCNICA DE MACHALA 1969