

ACTIVIDAD 4. Recepción de datos

Inicio

El objetivo de esta actividad es configurar una plataforma para que reciba los datos enviados por el objeto IoT.

Trabajaremos con Node-RED que recibirá los datos a través del protocolo MQTT

Node-red

Tomado de: <https://nodered.org/>

Node-RED es una herramienta de programación visual para la creación de aplicaciones de Internet de las cosas (IoT), flujos de trabajo y automatización en la nube. Se basa en una interfaz de usuario gráfica que permite a los usuarios crear flujos de trabajo mediante la conexión de nodos predefinidos que representan funciones o servicios específicos.

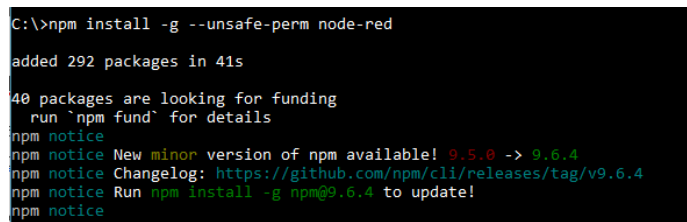
Los usuarios pueden seleccionar los nodos que necesitan y conectarlos arrastrando y soltando líneas entre ellos en una paleta de nodos en la interfaz gráfica de usuario. Los nodos pueden incluir una amplia variedad de servicios como bases de datos, servicios de la nube, redes sociales, dispositivos IoT, entre otros.

Node-RED se ejecuta en Node.js y utiliza el motor de JavaScript V8 de Google Chrome. Es una plataforma de código abierto, por lo que puede ser personalizada y ampliada por los usuarios a través de la creación de nuevos nodos o la integración de servicios externos.

Node-RED es popular en la creación de soluciones de automatización doméstica, como sistemas de seguridad y monitoreo, y en la automatización de procesos de negocio en empresas. También es compatible con múltiples plataformas y protocolos de IoT, como MQTT, HTTP, TCP y UDP.

Para instalar Node-RED debemos seguir los siguientes pasos:

1. Instalar la última versión de Node.js. Podemos descargarlo desde su sitio web oficial: <https://nodejs.org/en/download>.
2. Desde la línea de comandos ejecuta el siguiente comando para instalar Node-RED globalmente: **npm install -g --unsafe-perm node-red**



```
C:\>npm install -g --unsafe-perm node-red
added 292 packages in 41s
40 packages are looking for funding
  run 'npm fund' for details
npm notice
npm notice New minor version of npm available! 9.5.0 -> 9.6.4
npm notice Changelog: https://github.com/npm/cli/releases/tag/v9.6.4
npm notice Run npm install -g npm@9.6.4 to update!
npm notice
```

3. Después de esto simplemente se ejecuta el comando **node-red** desde el cmd.

```

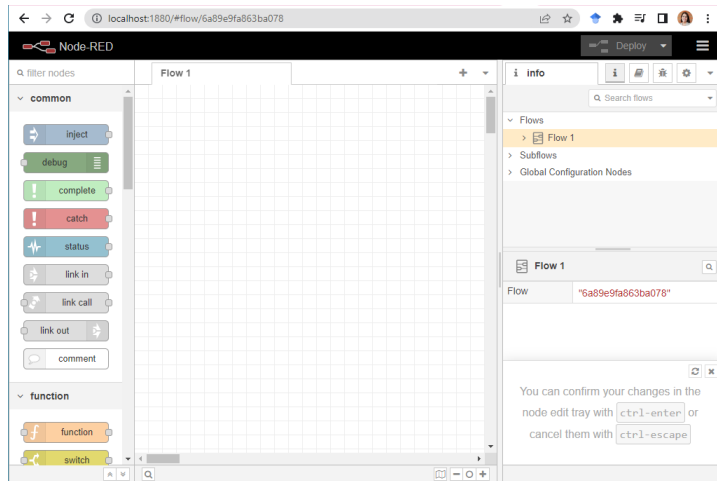
C:\>node-red
11 Apr 14:44:13 - [info]

Welcome to Node-RED
=====

11 Apr 14:44:13 - [info] Node-RED version: v3.0.2
11 Apr 14:44:13 - [info] Node.js version: v18.15.0
11 Apr 14:44:13 - [info] Windows_NT 10.0.15063 x64 LE

```

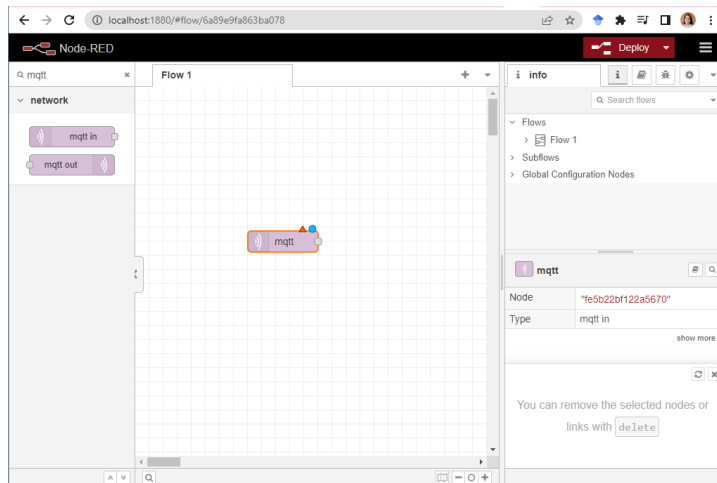
4. Por último, se accede al entorno de node-RED desde el navegador con la URL <http://localhost:1880>



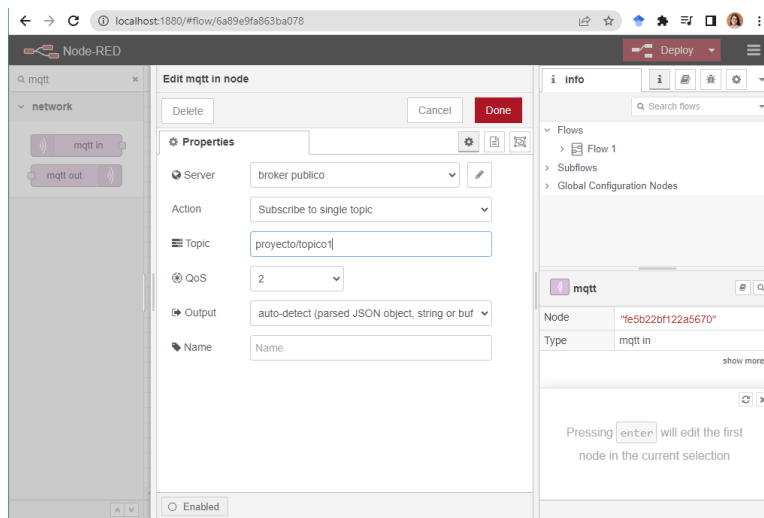
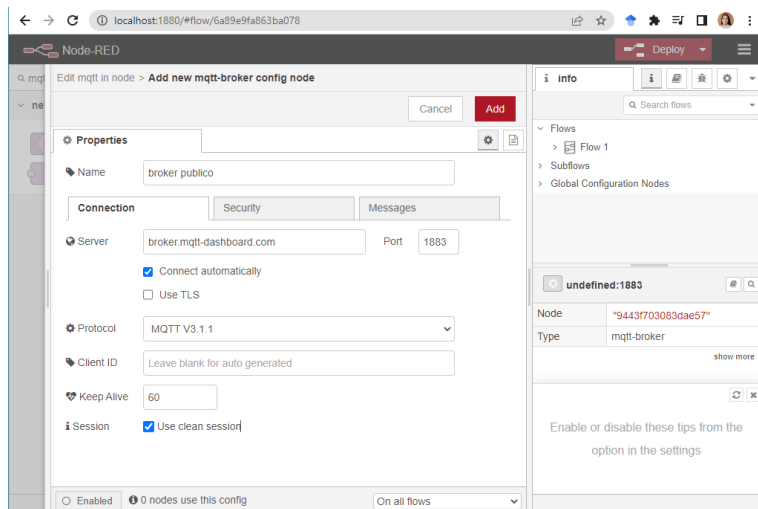
Configuración del protocolo MQTT en node-RED

Los pasos para configurar MQTT son los siguientes:

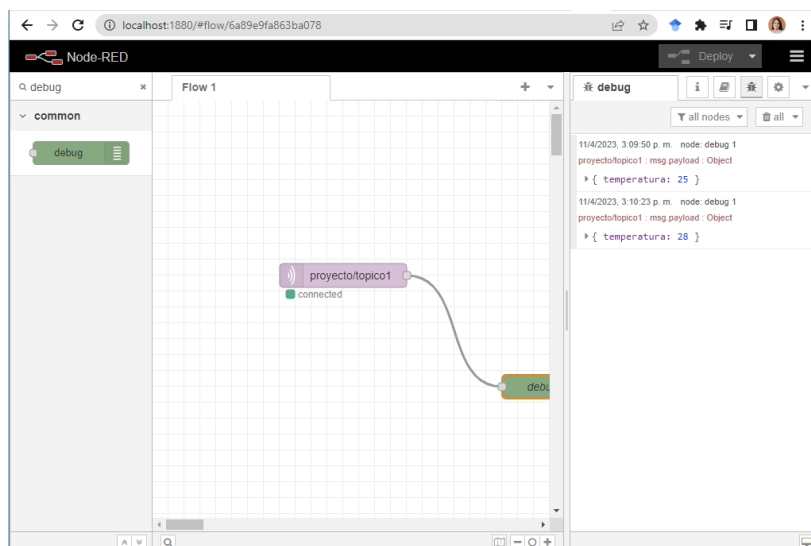
1. Se busca el nodo MQTT y se arrastra a la pantalla.



2. Se da doble clic sobre el nodo para configurarlo.



3. Se crea un debug para poder ver los mensajes en la console.



Aquí estoy haciendo una prueba enviando desde mqtt Explorer.

Configuración de la conexión al protocolo MQTT en el ESP32 (taller anterior)

A continuación, se muestran un sketch de ejemplo para publicar datos en el bróker en un tópico denominado **proyecto/topico1**

```
#include <ArduinoJson.h>
#include <PubSubClient.h>
#include <WiFi.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>

#define mqttUser ""
#define mqttPass ""
#define mqttPort 1883
#define Led 2

Adafruit_MPU6050 mpu;

const char* ssid = "xxxxx"; //ssid de la red inalambrica
const char* password = "xxxxxxx"; //password para conectarse a la red

char mqttBroker[] = "broker.mqtt-dashboard.com"; //ip del servidor
char mqttClientId[] = "device1"; //cualquier nombre
char inTopic1[] = "proyecto/topico1";
char inTopic2[] = "proyecto/topico2";

void callback(char* topic, byte* payload, unsigned int length) {

    String json=String((char *)payload);
    Serial.println();
    StaticJsonDocument<300> doc;
    DeserializationError error = deserializeJson(doc, json);
    if (error) { return; }
    int estado = doc["estado"];

    Serial.println(estado);
    if (estado == 1) {
        digitalWrite(Led, HIGH);
    }
    else {
        digitalWrite(Led, LOW);
    }
}

WiFiClient BClient;
```

```

PubSubClient client(BClient);

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect(mqttClientId, mqttUser, mqttPass)) {
      Serial.println("connected");
      client.subscribe(inTopic2);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup()
{
  pinMode(Led, OUTPUT);
  Serial.begin(115200); //Serial connection
  setup_wifi(); //WiFi connection
  client.setServer( mqttBroker, mqttPort );
  client.setCallback( callback );
  Serial.println("Setup done");
  delay(1500);

  Serial.println("Adafruit MPU6050 test!");
  // Try to initialize!
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");

  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  Serial.print("Accelerometer range set to: ");
  switch (mpu.getAccelerometerRange()) {
    case MPU6050_RANGE_2_G:
      Serial.println("+2G");
      break;
    case MPU6050_RANGE_4_G:
      Serial.println("+4G");
      break;
  }
}

```

```

case MPU6050_RANGE_8_G:
    Serial.println("+8G");
    break;
case MPU6050_RANGE_16_G:
    Serial.println("+16G");
    break;
}
mpu.setGyroRange(MPU6050_RANGE_500_DEG);
Serial.print("Gyro range set to: ");
switch (mpu.getGyroRange()) {
case MPU6050_RANGE_250_DEG:
    Serial.println("+ 250 deg/s");
    break;
case MPU6050_RANGE_500_DEG:
    Serial.println("+ 500 deg/s");
    break;
case MPU6050_RANGE_1000_DEG:
    Serial.println("+ 1000 deg/s");
    break;
case MPU6050_RANGE_2000_DEG:
    Serial.println("+ 2000 deg/s");
    break;
}

mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
Serial.print("Filter bandwidth set to: ");
switch (mpu.getFilterBandwidth()) {
case MPU6050_BAND_260_HZ:
    Serial.println("260 Hz");
    break;
case MPU6050_BAND_184_HZ:
    Serial.println("184 Hz");
    break;
case MPU6050_BAND_94_HZ:
    Serial.println("94 Hz");
    break;
case MPU6050_BAND_44_HZ:
    Serial.println("44 Hz");
    break;
case MPU6050_BAND_21_HZ:
    Serial.println("21 Hz");
    break;
case MPU6050_BAND_10_HZ:
    Serial.println("10 Hz");
    break;
case MPU6050_BAND_5_HZ:
    Serial.println("5 Hz");
    break;
}

```

```

    Serial.println("");
    delay(100);
}

void setup_wifi() {

    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}

void loop()
{
    if (!client.connected()) {
        reconnect();
    }
    client.loop();

    //aquí va el código asociado al sensor y generación del json
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);

    String json;
    StaticJsonDocument<300> doc;

    doc["accx"] = a.acceleration.x;
    doc["accy"] = a.acceleration.y;
    doc["accz"] = a.acceleration.z;
    doc["rotx"] = g.gyro.x;
    doc["roty"] = g.gyro.y;
    doc["rotz"] = g.gyro.z;
    doc["temp"] = temp.temperature;

    serializeJson(doc, json);
}

```

```

Serial.println(json);
int lon = json.length()+1;
char datojson[lon];
json.toCharArray(datojson, lon);
client.publish(inTopic1,datojson);
delay (5000);
}

```

Los mensajes deberán aparecer en node-RED.

The screenshot shows the Node-RED web interface running on localhost:1880. The main workspace displays a flow named 'Flow 1' with a single node labeled 'proyecto/topico1' (a debug node) connected to a 'debug' node. The left sidebar shows the 'common' and 'function' node categories. The right sidebar shows the 'debug' console with a list of received messages. The messages are JSON objects containing accelerometer and gyroscope data for a topic named 'proyecto/topico1'.

Debug Console Output:

```

proyecto/topico1 : msg.payload : Object
  { accx: -0.225054964, accy: 0.071826048, accz: 10.76433086, rotx: -0.095926493, roty: -0.013589587 ... }

12/4/2023, 9:18:33 a. m. node: debug 1
proyecto/topico1 : msg.payload : Object
  { accx: -0.213083953, accy: 0.057460841, accz: 10.81460953, rotx: -0.095127106, roty: -0.012523737 ... }

12/4/2023, 9:18:38 a. m. node: debug 1
proyecto/topico1 : msg.payload : Object
  { accx: -0.220266551, accy: 0.10295067, accz: 10.79306126, rotx: -0.095660031, roty: -0.01385605 ... }

12/4/2023, 9:18:44 a. m. node: debug 1
proyecto/topico1 : msg.payload : Object
  { accx: -0.227449164, accy: 0.088585466, accz: 10.79306126, rotx: -0.094594181, roty: -0.011990812 ... }

12/4/2023, 9:18:48 a. m. node: debug 1
proyecto/topico1 : msg.payload : Object
  { accx: -0.213083953, accy: 0.105344877, accz: 10.78587818, rotx: -0.095393561, roty: -0.014122511 ... }

```