

2 ème Année
Cycle supérieur-SIQ

Systèmes Communicants et Intelligents
Rapport

Système IoT complet avec utilisation de MQTT et NodeRed

Réalisé par :

- BOUROUINA Anfal
- SIQ-3-Binome-05

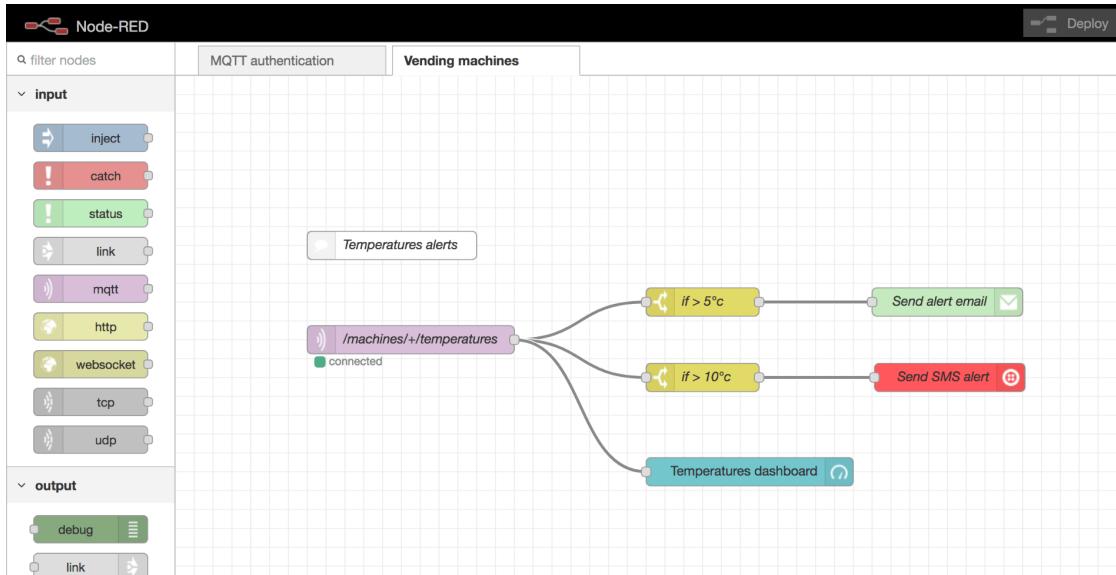
Encadré par :

- Mr. Abdennour SEHAD

2021-2022

1. Partie théorique

1. Introduction à la plateforme Node Red



Node-RED est un outil de développement basé sur les flux pour la programmation visuelle développé à l'origine par IBM pour connecter des périphériques matériels, des API et des services en ligne dans le cadre de l'Internet des objets.

Node-RED fournit un éditeur de flux au sein du navigateur Web, et est articulé autour de noeuds (nodes). L'environnement est construit sur Node.js, les flux créés sont stockés à l'aide de fichiers JSON, les fonctions sont en JavaScript et les éléments des applications peuvent être enregistrés ou partagés pour être réutilisés. Node-RED est nativement en mesure de souscrire et publier en MQTT, ce qui en fait un compagnon idéal pour la domotique.

Les noeuds disponibles sont nombreux et classés en catégories, les communs (avec l'injection de données, l'ajout de logs debug), les composants réseaux (pour faire des requêtes tcp, http ou mqtt), des fonctions (soit à définir soi-même avec du code JavaScript, soit avec des fonctions logiques telles que de l'analyse de cas), des éléments permettant de formater, trier, séparer ou joindre des données entre elles, du stockage de fichiers, ... En plus des noeuds déjà existants, il est tout à fait possible d'ajouter des noeuds développés et proposés par la communauté.

2. Etude théorique sur le protocole MQTT¹

MQTT est l'abréviation de Message Queuing Telemetry Transport. MQTT est un protocole de messagerie simple, conçu pour les appareils contraints ayant une faible bande passante. C'est donc la solution idéale pour échanger des données entre plusieurs appareils IoT.

¹ [Install Mosquitto Broker Raspberry Pi | Random Nerd Tutorials](#)

La communication MQTT fonctionne comme un système de publication et d'abonnement. Les appareils publient des messages sur un sujet spécifique (Topic) . Tous les appareils qui sont abonnés à ce sujet reçoivent le message.

Le courtier MQTT est chargé de recevoir tous les messages, de les filtrer, de décider qui s'y intéresse, puis de publier le message à tous les clients abonnés.

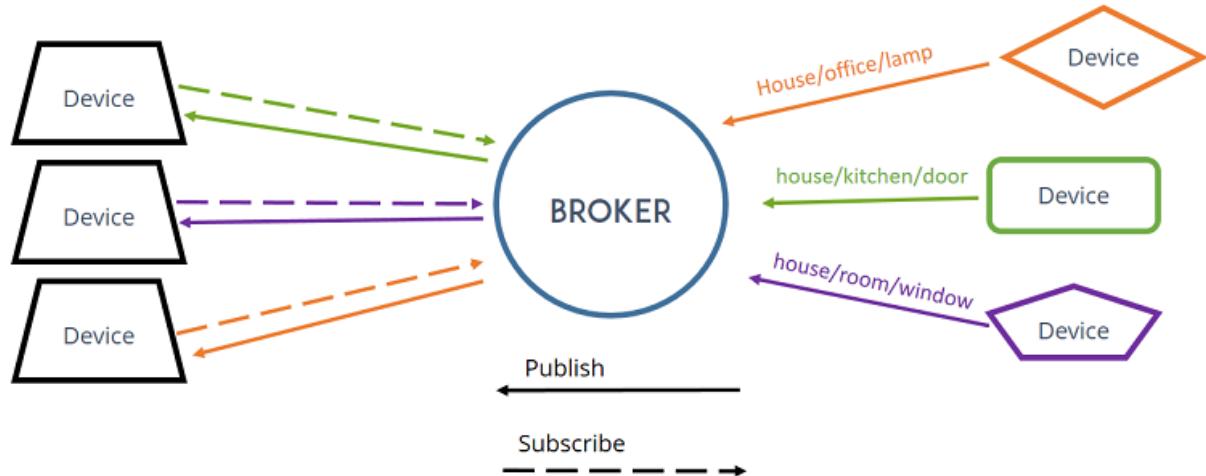


Figure 1. Architecture de communication avec le protocole MQTT.

Il existe plusieurs brokers qu'on peut utiliser. Dans les projets domotiques, nous utilisons souvent le broker Mosquitto installé sur un Raspberry Pi.

Mosquitto Broker²



Eclipse Mosquitto est un courtier (broker) de messages open source (sous licence EPL/EDL) qui met en œuvre le protocole MQTT dans ses versions 5.0, 3.1.1 et 3.1. Mosquitto est léger et peut être utilisé sur tous les appareils, des ordinateurs monocartes à faible puissance aux serveurs complets. Le projet Mosquitto fournit également une bibliothèque C pour la mise en œuvre de clients MQTT, ainsi que les très populaires clients MQTT en ligne de commande `mosquitto_pub` et `mosquitto_sub`.

Figure 2. Logo mosquitto.

² [Eclipse Mosquitto](#)

2. Partie pratique

1. Capter des données à travers Raspberry pi :

En premier lieu, On a pensé à utiliser le broker mosquito pour envoyer et recevoir les données depuis raspberry pi, donc on a utilisé raspberry pi comme un subscriber et le pc comme un publisher, dont le serveur central est le serveur de test de mosquito "test.mosquitto.org". En utilisant mqtt , on allume et on éteint la led connectée à raspberry pi depuis le pc , suivant le schéma :

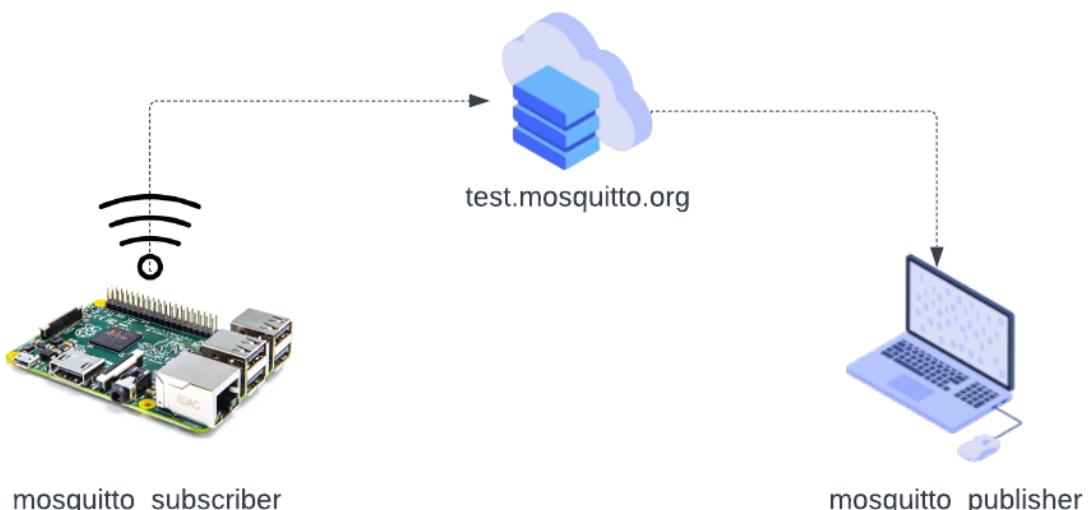


Figure 3 : Schéma de communication mqtt à distance.

Configuration PC (os: Windows 10):

1. Télécharger et Installer Mosquitto depuis le site officiel.³
2. Lancer les commandes Mosquitto à partir du C:\Program Files\mosquitto
3. Utiliser la commande "**mosquitto_pub**" pour publier dans un topic et "**mosquitto_sub**" pour s'abonner à un topic.

Le pc va agir comme un publisher qui envoie des messages à raspberry pi à travers les topic "**CoreElectronics/test**" et "**CoreElectronics/topic**" définit sur le serveur de test de mosquito.

On a testé deux commandes simples :

- La commande "Hello" pour allumer la led.
- La commande "World!" pour l'éteindre.

```
C:\Program Files\mosquitto>mosquitto_pub -h test.mosquitto.org -t CoreElectronics/topic -m World!
C:\Program Files\mosquitto>mosquitto_pub -h test.mosquitto.org -t CoreElectronics/topic -m "World!"
C:\Program Files\mosquitto>mosquitto_pub -h test.mosquitto.org -t CoreElectronics/test -m "World!"""
C:\Program Files\mosquitto>mosquitto_pub -h test.mosquitto.org -t CoreElectronics/test -m Hello
C:\Program Files\mosquitto>mosquitto_pub -h test.mosquitto.org -t CoreElectronics/topic -m World!
```

³ [Download | Eclipse Mosquitto](#)

Configuration Raspberry pi :

Sur Raspberry pi on a utilisé la librairie python “`paho.mqtt`”, pour construire des programmes qui utilisent le protocole mqtt pour l'échanges de messages. Nous avons créé deux programmes , un pour publier des messages sur les topics “`CoreElectronics/test`” et “`CoreElectronics/topic`” et l'autre pour s'abonner à ces deux topics.

+ Code de subscriber : `mqtt_client_demo.py`

```
# MQTT Client demo
# Continuously monitor two different MQTT topics for data,
# check if the received data matches two predefined 'commands'

import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO #Importe la bibliothèque pour contrôler les GPIOs

GPIO.setmode(GPIO.BOARD) #Définit le mode de numérotation (Board)
GPIO.setwarnings(False) #On désactive les messages d'alerte

LED = 26 #(7) Définit le numéro du port GPIO qui alimente la led
GPIO.setup(LED, GPIO.OUT) #Active le contrôle du GPIO
state = GPIO.input(LED) #Lit l'état actuel du GPIO, vrai si allumé, faux si éteint

# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

    # Subscribing in on_connect() - if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe("CoreElectronics/test")
    client.subscribe("CoreElectronics/topic")

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

    if msg.payload == b'Hello':
        print("Received message #1, do something")
        # Do something
        GPIO.output(26,GPIO.HIGH)

    if msg.payload == b'World!':
        print("Received message #2, do something else")
        # Do something else
        GPIO.output(26,GPIO.LOW)

# Create an MQTT client and attach our routines to it.
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("test.mosquitto.org", 1883, 60)

# Process network traffic and dispatch callbacks. This will also handle
# reconnecting. Check the documentation at
# https://github.com/eclipse/paho.mqtt.python
# for information on how to use other loop*() functions
client.loop_forever()
```

+ Code de publisher : [mqtt_publish_demo.py](#)

```
# MQTT Publish Demo
# Publish two messages, to two different topics

import paho.mqtt.publish as publish

publish.single("CoreElectronics/test", "Hello", hostname="test.mosquitto.org")
publish.single("CoreElectronics/topic", "World!", hostname="test.mosquitto.org")
print("Done")
```

+ Branchement:

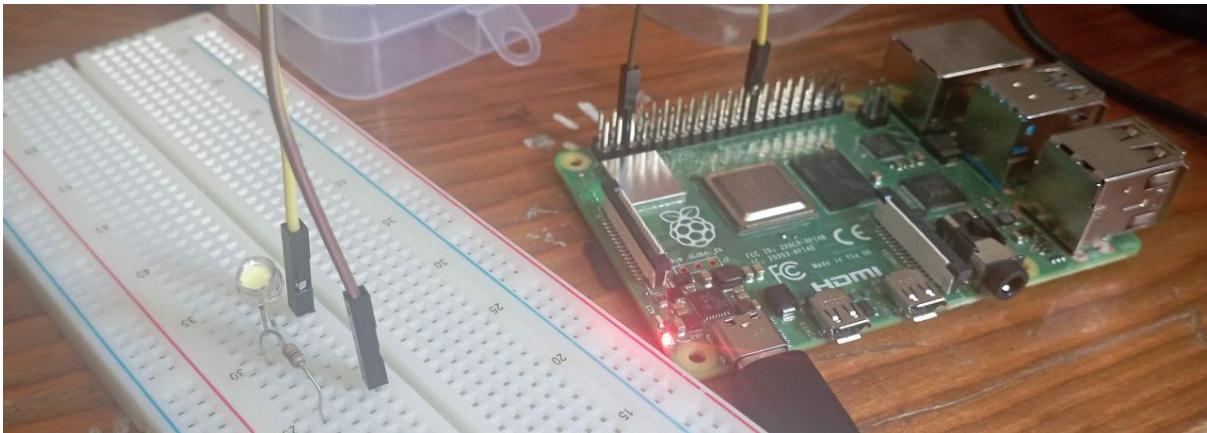


Figure 4 : Branchement LED avec Raspberry Pi.

+ Test

On exécutant le 1er programme on peut recevoir les messages à partir du pc et les interpréter pour contrôler la led.

```
20
21
22
23
24 # reconnect then subscriptions will be renewed
25 client = mqtt.Client()
26 client.connect("test.mosquitto.org")
27 client.loop_forever()
28
29 def on_message(client, userdata, msg):
30     print(msg.topic + " " + str(msg.payload))
31
32     if msg.topic == "CoreElectronics/test":
33         print("Received message #1, do something")
34     elif msg.topic == "CoreElectronics/topic":
35         print("Received message #2, do something else")
36
37
38 if __name__ == "__main__":
39     client.on_message = on_message
40     client.publish("CoreElectronics/test", "Hello")
41     client.publish("CoreElectronics/topic", "World!")
42
43     client.loop_forever()
```

Figure 5 : Réception des messages mqtt à partir du PC.

2. Utilisation de Node Red avec MQTT

Dans cette deuxième partie, nous avons utilisé un set-up local , ou le raspberry pi agit comme un serveur de broker mosquitto , un arduino comme publisher en envoyant les valeurs d'un capteur "ultrasonique de distance" à la plateforme node-red sur le topic "ultraSonic" et un arduino comme subscriber , recevant les données à partir du node-red sur le topic "ledRadar" et actionne (allume en led ou un buzzer selon la valeur envoyée), et un pc ou on utilise node-red pour contrôler le flux mqtt.

le set up est comme suit :

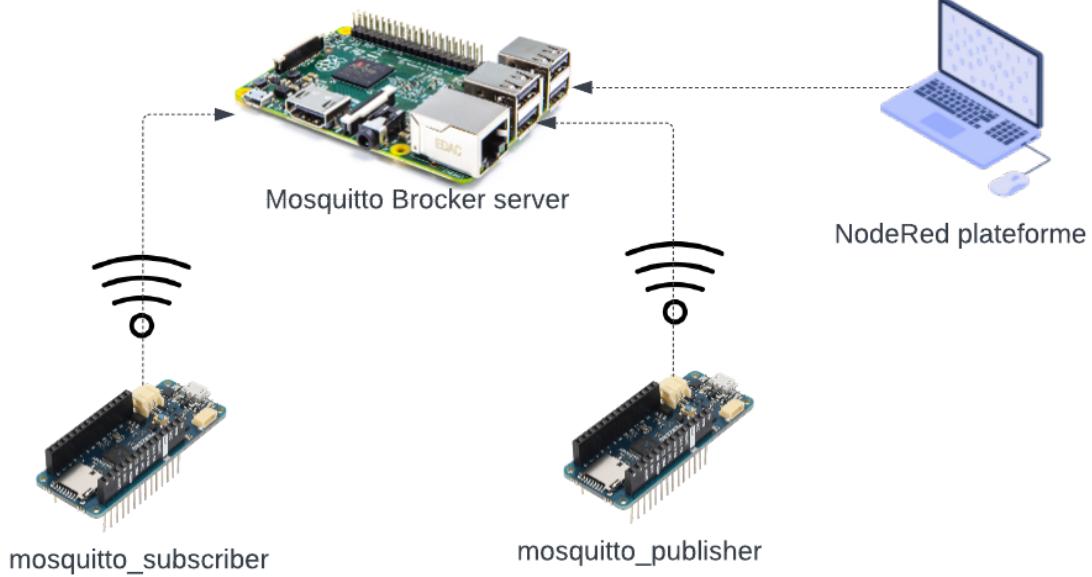


Figure 6 : Schéma de communication mqtt local.

Branchement

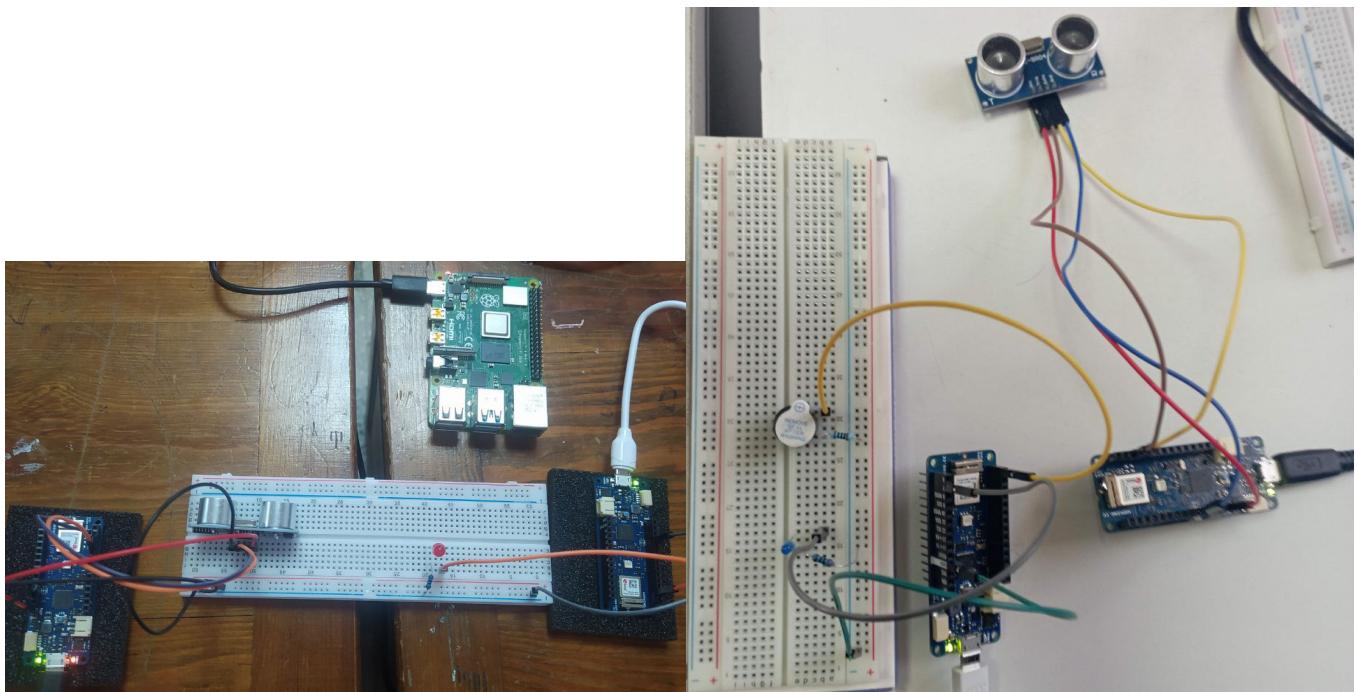


Figure 7 : Branchement des composants IoT.

Configuration raspberry pi

1. Installer mosquitto

```
sudo apt install -y mosquitto mosquitto-clients
```

2. Vérifier que le processus mosquitto est activé

```
sudo systemctl status mosquitto
```

```
● mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2020-04-09 13:31:13 CEST; 1h 9min ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
 Main PID: 1574 (mosquitto)
    Tasks: 1 (limit: 2077)
   Memory: 788.0K
      CGroupl: /system.slice/mosquitto.service
                 └─1574 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

avr 09 13:31:13 raspberrypi systemd[1]: Starting Mosquitto MQTT v3.1/v3.1.1 Broker...
avr 09 13:31:13 raspberrypi systemd[1]: Started Mosquitto MQTT v3.1/v3.1.1 Broker.
```

Code Arduino

Arduino 1 (publisher):

cet arduino va envoyer les données reçues à partir du capteur de distance au Node-red connecté au broker mosquitto sur le topic “ultraSonic”.

L'arduino est connecté au réseau de serveur local mqtt à travers le wifi.

Les librairies ArduinoMqtt et WIFININA doivent être installées.

```
#define echoPin 2 // attach pin D2 Arduino to pin Echo of HC-SR04
#define trigPin 3 //attach pin D3 Arduino to pin Trig of HC-SR04
#include <ArduinoMqttClient.h>
#if defined(ARDUINO_SAMD_MKRWIFI1010) || defined(ARDUINO_SAMD_NANO_33_IOT) ||
defined(ARDUINO_AVR_UNO_WIFI_REV2)
  #include <WIFININA.h>
#elif defined(ARDUINO_SAMD_MKR1000)
  #include <WiFi101.h>
#elif defined(ARDUINO_ESP8266_ESP12)
  #include <ESP8266WiFi.h>
#endif

#include "arduino_secrets.h"
//please enter your sensitive data in the Secret tab/arduino_secrets.h
char ssid[] = "SSID";          // your network SSID (name)
char pass[] = "PASSWORD";      // your network password (use for WPA, or use as key
for WEP)
int distance;
// To connect with SSL/TLS:
// 1) Change WiFiClient to WiFiSSLClient.
// 2) Change port value from 1883 to 8883.
// 3) Change broker value to a server with a known SSL/TLS root certificate
//     flashed in the WiFi module.

WiFiClient wifiClient;
MqttClient mqttClient(wifiClient);

const char broker[] = "192.168.73.169";
int      port      = 1883;
const char topic[] = "ultraSonic";

const long interval = 1000;
unsigned long previousMillis = 0;

int count = 0;
```

```

void setup() {
    //Initialize serial and wait for port to open:
    Serial.begin(9600);
    while (!Serial) {
        ; // wait for serial port to connect. Needed for native USB port only
    }

    // attempt to connect to Wifi network:
    Serial.print("Attempting to connect to WPA SSID: ");
    Serial.println(ssid);
    while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
        // failed, retry
        Serial.print(".");
        delay(5000);
    }

    Serial.println("You're connected to the network");
    Serial.println();

    // You can provide a unique client ID, if not set the library uses
    Arduino-millis()
    // Each client must have a unique client ID
    // mqttClient.setId("clientId");

    // You can provide a username and password for authentication
    // mqttClient.setUserNamePassword("username", "password");

    Serial.print("Attempting to connect to the MQTT broker: ");
    Serial.println(broker);

    if (!mqttClient.connect(broker, port)) {
        Serial.print("MQTT connection failed! Error code = ");
        Serial.println(mqttClient.connectError());

        while (1);
    }

    Serial.println("You're connected to the MQTT broker!");
    Serial.println();
    pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
    pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT
}

void loop() {
    // call poll() regularly to allow the library to send MQTT keep alives which
    // avoids being disconnected by the broker
    mqttClient.poll();

    // avoid having delays in loop, we'll use the strategy from BlinkWithoutDelay
    // see: File -> Examples -> 02.Digital -> BlinkWithoutDelay for more info
    unsigned long currentMillis = millis();

    if (currentMillis - previousMillis >= interval) {
        // save the last time a message was sent
        previousMillis = currentMillis;

        Serial.print("Sending distance to topic: ");
        Serial.println(topic);
        Serial.print("distance : ");
        Serial.println(distance);
        Serial.print(" cm.");
        // send message, the Print interface can be used to set the message contents
        distance = captureDistance();
        mqttClient.beginMessage(topic);
        mqttClient.print(distance);
        mqttClient.endMessage();
    }
}

```

```

        Serial.println();

        count++;
    }

}

int captureDistance()
{
    long duration; // variable for the duration of sound wave travel
    int distanceTmp;
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    // Reads the echoPin, returns the sound wave travel time in microseconds
    duration = pulseIn(echoPin, HIGH);
    // Calculating the distance
    distanceTmp = duration * 0.034 / 2;
    return distanceTmp;
}

```

Arduino 2 (subscriber):

Cet arduino va s'abonner au topic “ledRadar” et recevoir les données envoyées depuis Node-Red, les interpréter pour faire une action. dans ce cas, le node-red lui envoie une valeur binaire 1 ou 0 pour allumer/ éteindre une led attachée à arduino.

L'arduino est connecté au réseau de serveur local mqtt à travers le wifi.

Les librairies ArduinoMqtt et WiFiNINA doivent être installées.

```

#include <ArduinoMqttClient.h>
#if defined(ARDUINO_SAMD_MKRWIFI1010) || defined(ARDUINO_SAMD_NANO_33_IOT) ||
defined(ARDUINO_AVR_UNO_WIFI_REV2)
    #include <WiFiNINA.h>
#elif defined(ARDUINO_SAMD_MKR1000)
    #include <WiFi101.h>
#elif defined(ARDUINO_ESP8266_ESP12)
    #include <ESP8266WiFi.h>
#endif

#include "arduino_secrets.h"
/////////please enter your sensitive data in the Secret tab/arduino_secrets.h
char ssid[] = SECRET_SSID;          // your network SSID (name)
char pass[] = SECRET_PASS;         // your network password (use for WPA, or use as
key for WEP)

// To connect with SSL/TLS:
// 1) Change WiFiClient to WiFiSSLClient.
// 2) Change port value from 1883 to 8883.
// 3) Change broker value to a server with a known SSL/TLS root certificate
//     flashed in the WiFi module.

WiFiClient wifiClient;
MqttClient mqttClient(wifiClient);

const char broker[] = "192.168.73.169";
int      port      = 1883;
const char topic[] = "ledRadar";
int ledPin = 7;

```

```

void setup() {
    //Initialize serial and wait for port to open:
    Serial.begin(9600);
    while (!Serial) {
        ; // wait for serial port to connect. Needed for native USB port only
    }

    // attempt to connect to Wifi network:
    Serial.print("Attempting to connect to WPA SSID: ");
    Serial.println(ssid);
    while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
        // failed, retry
        Serial.print(".");
        delay(5000);
    }

    Serial.println("You're connected to the network");
    Serial.println();

    // You can provide a unique client ID, if not set the library uses
    Arduino-millis()
    // Each client must have a unique client ID
    // mqttClient.setId("clientId");

    // You can provide a username and password for authentication
    // mqttClient.setUserNamePassword("username", "password");

    Serial.print("Attempting to connect to the MQTT broker: ");
    Serial.println(broker);

    if (!mqttClient.connect(broker, port)) {
        Serial.print("MQTT connection failed! Error code = ");
        Serial.println(mqttClient.connectError());

        while (1);
    }

    Serial.println("You're connected to the MQTT broker!");
    Serial.println();

    Serial.print("Subscribing to topic: ");
    Serial.println(topic);
    Serial.println();

    // subscribe to a topic
    mqttClient.subscribe(topic);

    Serial.print("Waiting for messages on topic: ");
    Serial.println(topic);
    Serial.println();
    pinMode(ledPin, OUTPUT);
}

void loop() {
    int messageSize = mqttClient.parseMessage();
    if (messageSize) {
        // we received a message, print out the topic and contents
        Serial.print("Received a message with topic '");
        Serial.print(mqttClient.messageTopic());
        Serial.print("'", length );
        Serial.print(messageSize);
        Serial.println(" bytes:");

        // use the Stream interface to print the contents
        while (mqttClient.available()) {
            Serial.print((char)mqttClient.read());
            char msg = (char)mqttClient.read();
    }
}

```

```

        }
        // allumer / eteindre la led selon la valeur reçu
        if (msg == '1') {
            digitalWrite(ledPin, HIGH);
        }else if (msg == '2') {
            digitalWrite(ledPin, LOW);
        }
        Serial.println();
        Serial.println();
    }
}

```

La plateforme Node-Red:

Nous avons installé et configuré node-red sur un pc dans le même réseau que le serveur du broker mqtt sur raspberry pi.

Installation Node-Red⁴:

1. Télécharger et installer NodeJS depuis le site officiel⁵.
2. Installer node-red à travers la gestionnaire des packages “npm” en utilisant la commande: `npm install -g --unsafe-perm node-red`.
3. Lancer node-red en utilisant la commande node-red:

```
C:\Users\DELL>node-red
7 Jun 21:12:13 - [info]

Welcome to Node-RED
=====
7 Jun 21:12:13 - [info] Node-RED version: v2.2.2
7 Jun 21:12:13 - [info] Node.js  version: v14.16.1
7 Jun 21:12:13 - [info] Windows_NT 10.0.19043 x64 LE
7 Jun 21:12:42 - [info] Loading palette nodes
7 Jun 21:13:19 - [info] Dashboard version 3.1.7 started at /ui
7 Jun 21:13:20 - [info] Settings file  : C:\Users\DELL\.node-red\settings.json
7 Jun 21:13:20 - [info] Context store  : 'default' [module=memory]
7 Jun 21:13:20 - [info] User directory : \Users\DELL\.node-red
7 Jun 21:13:20 - [warn] Projects disabled : editorTheme.projects.enabled=false
7 Jun 21:13:20 - [info] Flows file    : \Users\DELL\.node-red\flows.json
7 Jun 21:13:20 - [info] Creating new flow file
7 Jun 21:13:20 - [warn]
```

4. Accéder à la plateforme sur : <http://127.0.0.1:1880/>.

Configuration de Node-Red :

Pour recevoir les données mqtt et les traiter sur Node-Red nous avons besoin de le configurer en utilisant `mqttIn` et `mqttOut` pour les cartes arduino publisher et subscriber respectivement.

⁴ [Running on Windows : Node-RED \(nodered.org\)](#)

⁵ [Download | Node.js \(nodejs.org\)](#)

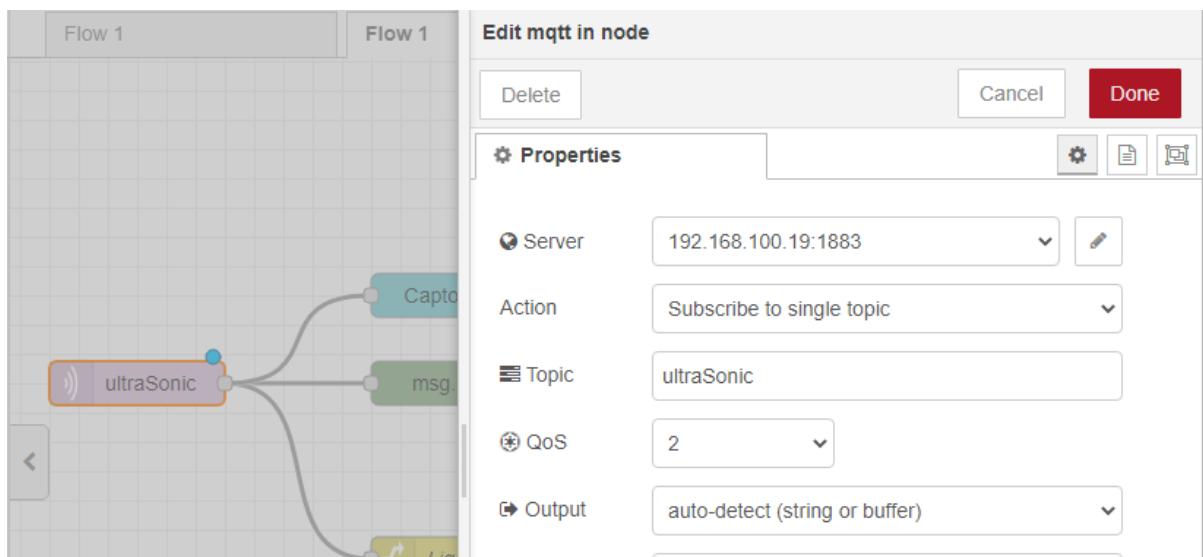


Figure 8: Configuration de mqttIn (topic “ultraSonic”).

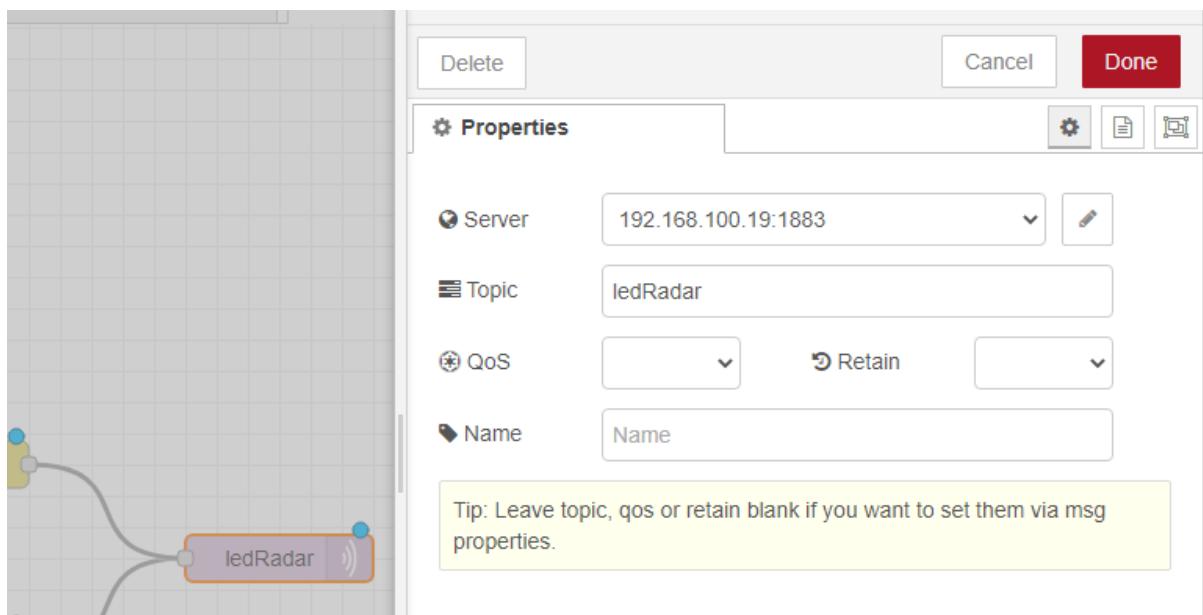


Figure 9: Configuration de mqttOut (topic “ledRadar”).

Voici la configuration globale du Node Red :

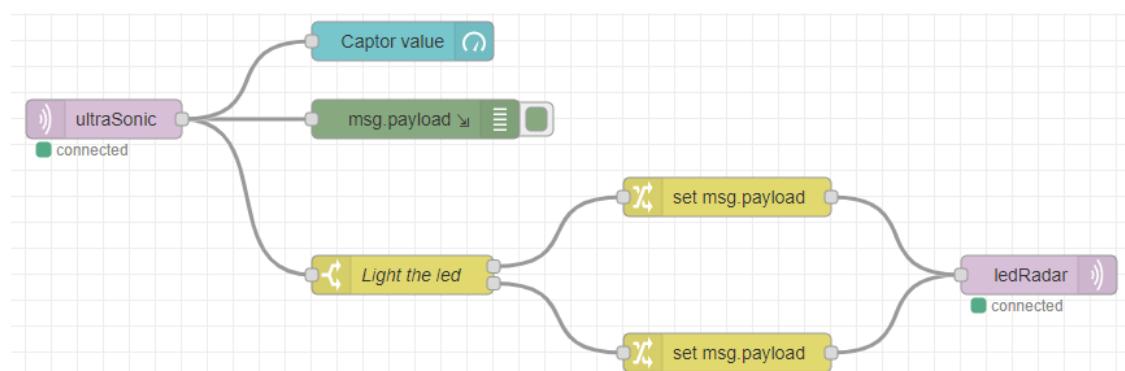


Figure 10: Configuration globale de NodeRed.

Le capteur de distance est représenté par une gauge “Captor value” envoyant ses données vers le topic “ultraSonic”.

Les messages reçus peuvent être affichés à travers msg payload.

Pour contrôler la led attaché à l'arduino subscriber, nous avons introduit un switch “Light the led” qui contient une logique binaire. En se basant sur la valeur reçue du capteur de distance, on la compare avec un seuil (10 cm), si cette valeur est inférieure à 10 cm on allume la led, sinon on l'éteint.

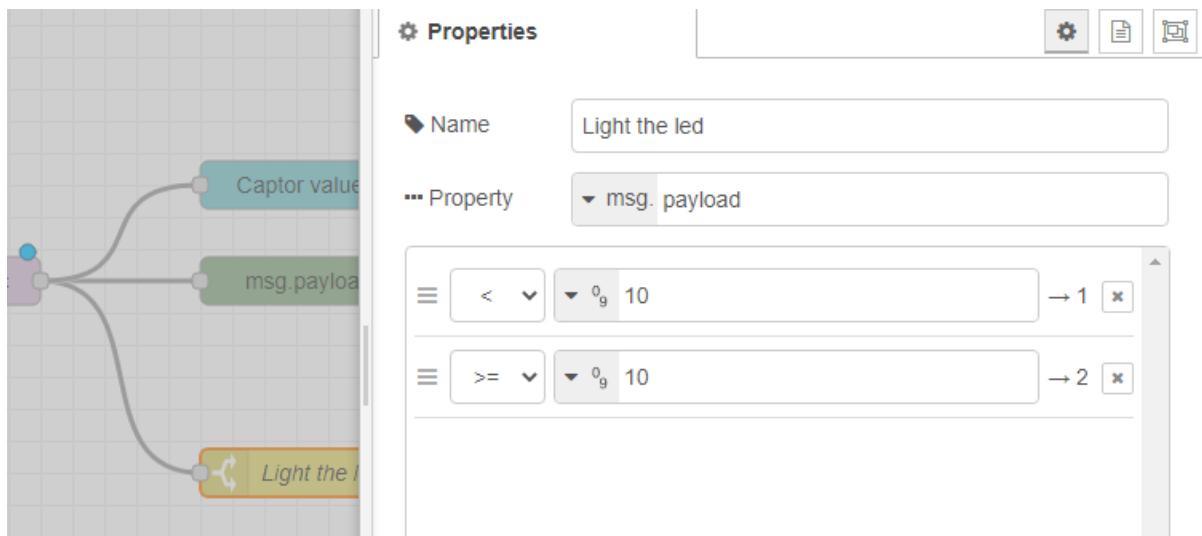


Figure 11 : Configuration de switch “Light the led”.

Le fichier de configuration est attaché sous le nom “**flows.json**”.