UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Desarrollo de una aplicación IoT para una red de sensores doméstica mediante el protocolo MQTT-SN

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Dias Toapanta, Jonathan Xavier

Tutor/a: León Fernández, Antonio

CURSO ACADÉMICO: 2022/2023

Resumen

MQTT-SN (MQTT for Sensor Networks) es una variación poco conocida del popular protocolo MQTT para Internet de las Cosas (IoT).

Basado en UDP está diseñado para funcionar en redes Wireless con sensores de bajas prestaciones.

El objetivo del trabajo es diseñar una aplicación IoT basada en MQTT-SN que gestione una red doméstica elaborada con distintos tipos de sensores con la finalidad de poder controlar dichos sensores remotamente a través de dispositivos móviles y/o desde Internet.

Resum

MQTT-SN (MQTT for Sensor Networks) és una variació poc coneguda del popular protocol MQTT per a Internet de les Coses (IoT).

Basat en UDP està dissenyat per a funcionar en xarxes Wireless amb sensors de baixes prestacions.

L'objectiu del treball és dissenyar una aplicació IoT basada en MQTT-SN que gestione una xarxa domèstica elaborada amb diferents tipus de sensors amb la finalitat de poder controlar aquests sensors remotament a través de dispositius mòbils i/o des d'Internet.

Abstract

MQTT-SN (MQTT for Sensor Networks) is a little-known variation of the popular MQTT protocol for the Internet of Things (IoT).

Based on UDP, it is designed to work in wireless networks with low-performance sensors.

The work aims to design an IoT application based on MQTT-SN that manages a home network built with different types of sensors in order to be able to control these sensors remotely through mobile devices or from the Internet.

Índice

Lista de Figuras

Capítulo 1. Introducción

1.1 Motivación

En la actualidad la tecnología del Internet de las Cosas (IoT) está presente en cualquier ámbito de la vida de las personas. Estamos en la etapa de mayor interconexión tanto entre las personas como entre los dispositivos, ya sea, en el ámbito empresarial, público o doméstico.

Por esta razón continúa en aumento el desarrollo de servicios y aplicaciones IoT bajo distintos protocolos de comunicación en función de las necesidades de la aplicación y el tipo de red donde trabajan.

MQTT, CoAP y HTTP son los protocolos de comunicación más comunes que se utilizan en dispositivos o aplicaciones IoT. Aunque existen otros, hay un protocolo llamado MQTT-SN que llama la atención porque funciona de forma similar al popular protocolo MQTT, pero que cuenta con una especificación propia y está diseñado para redes inalámbricas de sensores.

El protocolo MQTT-SN no es tan popular como los mencionados anteriormente y a día de hoy tampoco está estandarizado, por lo que supone un reto y motivación utilizarlo en un proyecto propio.

En este contexto surge el interés de desarrollar este trabajo. Diseñar e implementar una aplicación IoT que interactúe con una red inalámbrica local que este compuesta de dispositivos sensores y su comunicación se base en el protocolo MQTT-SN.

Este trabajo es uno de los primeros que persiguen mostrar las características del protocolo MQTT-SN en una red inalámbrica doméstica.

1.2 Objetivos

El objetivo general del trabajo es diseñar e implementar una aplicación IoT que monitoree cada una de las variables medidas por los dispositivos sensores (temperatura, humedad, entre otros) que se encuentran desplegados en una red inalámbrica local donde la comunicación entre los dispositivos sensores sea mediante el protocolo MQTT-SN.

Al mismo tiempo, el trabajo pretende mostrar algunas de las características del protocolo MQTT-SN de una forma asequible y económica.

Este objetivo general se puede dividir en los siguientes subobjetivos:

- Realizar el diseño de una red inalámbrica local basada en arquitectura MQTT-SN.
- Realizar el diseño de la aplicación para que sea sencilla de utilizar y fácil de ampliar. Para este subobjetivo se pretende que el usuario pueda:
  - monitorear las variables de medida de los sensores de la vivienda.
  - tomar acciones de control sobre los sensores de la vivienda en tiempo real.
  - Añadir nuevos dispositivos sensores en la arquitectura de la red y en la interfaz de la aplicación.

1.3 Estructura de la memoria

Esta memoria está compuesta por seis capítulos y son los siguientes:

Primer capítulo: introducción. Este apartado está formado por la motivación, los objetivos, la estructura de la memoria y la metodología para el diseño y desarrollo de la aplicación.

Segundo capítulo: tecnologías. En este apartado se expone el protocolo MQTT-SN, los dispositivos hardware a utilizar y las herramientas de programación a utilizar.

Tercer capítulo: diseño y desarrollo. En este apartado se describen los agentes elegidos de la arquitectura de red, el diseño de la aplicación y las pruebas de testeo con la aplicación.

Cuarto capítulo: conclusiones. En este apartado se exponen las conclusiones del trabajo y posibles mejoras de futuro.

Quinto capítulo: bibliografía. En este apartado se indican las fuentes documentales utilizadas en el desarrollo del trabajo.

Sexto capítulo: anexos. En este último apartado se exponen los pasos de instalación y configuración de los distintos componentes utilizados en la aplicación, así como el código funcional de un dispositivo sensor.

1.4 Metodología

Para alcanzar los objetivos del trabajo, la metodología a seguir se llevará a cabo en cuatro etapas y son las siguientes:

- **Fase de investigación:** en esta etapa se ha hecho una investigación del rp, sin olvidar los requisitos técnicos del desarrollo. Las tareas a realizar son:
  - Estudio del concepto de Internet de las Cosas, Redes de Sensores y su relación.
  - Estudio del protocolo MQTT e investigación de aplicaciones desarrolladas.
  - Estudio del protocolo MQTT-SN y sus diferencias con el protocolo MQTT.
  - Estudio de los dispositivos Raspberry Pi, NodeMCU (ESP-12E) y Sistemas en Chip (ESP8266, ESP32).
  - Estudio de las características de los sensores del mercado (DHT11, DHT22, HC-SR04) y de su conexionado con los Sistemas en Chip.
  - Elección de la ubicación de los dispositivos sensores en la vivienda.
- **Fase de diseño y desarrollo:** teniendo en cuenta la información recogida en la fase anterior se crea el diseño de la arquitectura de la red a utilizar y el diseño de la interfaz gráfica de la aplicación. Las tareas a realizar son:
  - Estudio y elección de los diferentes tipos de agentes en una arquitectura MQTT-SN (clientes, puertas de enlace, broker).
  - Familiarización con el IDE de Arduino.
  - Elección del tipo de aplicación IoT a desarrollar según los subobjetivos marcados.
  - Familiarización con la herramienta de programación Node-Red.
- **Fase de implementación:** a partir del diseño creado se trata de implementar soluciones, que luego se sometan a pruebas para encontrar aspectos a mejorar.

  Las tareas a realizar son:
  - Montaje de la arquitectura MQTT-SN.

  - Programación del código funcional de los Sistemas en Chip (NodeMCU, ESP8266, ESP32).
  - Programación de la aplicación web IoT.
  - Enlace de la aplicación desarrollada con la arquitectura MQTT-SN desplegada.

- **Fase de evaluación:** se realizan pruebas con la aplicación desarrollada y se hacen los cambios pertinentes. Las tareas a realizar son:
  - Pruebas con la aplicación y resolución de errores.
  - Análisis de resultados y conclusiones.

Capítulo 2. Tecnologías

Este capítulo abordará el protocolo MQTT-SN para la comunicación de los dispositivos dentro de la red doméstica, los dispositivos hardware a utilizar y las herramientas software a utilizar en el desarrollo de la aplicación IoT.

2.1 Protocolo de comunicación MQTT-SN

El aumento del interés por las Redes de Sensores Inalámbricas, tanto desde el punto de vista comercial como técnico, originó que se quisiera extender el sistema de publicación/suscripción de la empresa a las Redes de Sensores Inalámbricas con el objetivo de que los datos de campo recopilados por los sensores y actuadores estuvieran disponibles para todas las aplicaciones como cualquier otra información empresarial y permitir el control de los sensores y actuadores desde cualquier lugar.

En las Redes de Sensores Inalámbricas, generalmente los dispositivos con Sistemas en Chip (SoC) utilizados son de bajo costo, con recursos de procesamiento y almacenamiento limitados, que funcionan con baterías que operan en redes que nos son TCP/IP, como Zigbee, UDP, entre otras. Esto los limita a poder utilizar el protocolo de comunicación MQTT, ya que este protocolo se aplica a la pila TCP/IP y su correcto funcionamiento se basa en las garantías subyacentes que ofrece TCP.

Por esta razón se diseñó una variación del protocolo MQTT para estas Redes de Sensores Inalámbricas, denominado protocolo MQTT-SN (MQTT for Sensor Networks) que está optimizado para implementarse en los dispositivos antes mencionados con el objetivo de extender el protocolo MQTT más allá del alcance de la infraestructura TCP/IP para soluciones de sensores y actuadores.

MQTT-SN puede funcionar junto con MQTT, pero en cuanto se quiere tener conexión con la red de Internet es necesario usar el protocolo MQTT completo, diseñado para la conectividad.

MQTT-SN cuenta con su propia especificación, que está disponible públicamente, pero no está estandarizado por un organismo de estándares como puede ser OASIS.

2.1.1 Características principales del protocolo MQTT-SN

El diseño de MQTT-SN es estar lo más cerca posible de MQTT. Por lo tanto, toda la semántica del protocolo permanece, en la medida de lo posible, igual a la definida por MQTT.

A continuación, se exponen aquellos puntos que son nuevos o se desvían de MQTT.

1. Uso de UDP como protocolo de transporte.
2. Reducción en el tamaño de carga del mensaje.
3. Mensaje CONNECT dividido en tres mensajes. Los dos mensajes adicionales son opcionales y se utilizan para transferir el tema Will y el mensaje Will a la puerta de enlace.
4. Registro de los topics con nombres largos y obtención de identificadores asociados para posteriormente publicar o suscribirse.
5. Uso de topics cortos de dos bytes de tamaño fijo que se transportan junto con los datos dentro sin requerir registro.
6. Uso de identificadores “predefinidos” de topics sin necesidad de registro. Su asignación se define en la puerta de enlace.
7. Procedimiento de “descubrimiento”, ayuda a localizar la dirección de red de una puerta de enlace operativa.

8. Posibilidad de persistencia en el topic Will y mensaje Will. Un cliente puede modificar su tema Will y su mensaje Will durante una sesión.
9. Procedimiento de "cliente dormido". Los dispositivos pueden entrar en suspensión y los mensajes destinados a ellos se almacenan en un búfer en la puerta de enlace y son entregados más tarde cuando los dispositivos se despierten.

2.1.2 Arquitectura de red y agentes

En la arquitectura del protocolo MQTT-SN hay 3 tipos de componentes. Los clientes, la puerta de enlace (integrada o no en un broker) y el reenviador.

Los clientes se conectan a una puerta de enlace en su red, utilizando el protocolo MQTT-SN. La puerta de enlace se conecta a un broker MQTT para que los datos puedan salir de la red local.

Si un cliente no tiene acceso a una puerta de enlace conectada directamente a su red, puede utilizar un reenviador para acceder a una puerta de enlace en otra red. La función de este reenviador es encapsular las tramas MQTT-SN que recibe de la red inalámbrica y las reenvía sin alterarlas a la puerta de enlace de la otra red. En sentido contrario, el proceso es el mismo, des encapsula las tramas que recibe de la puerta de enlace y sin alterarlas las reenvía a los clientes.

Podemos disponer de puertas de enlace ya integradas en un broker MQTT, por ejemplo, el broker RSMB y el broker EMQX.

En el caso de que la puerta de enlace no esté integrada en un broker, es necesario un broker MQTT externo para que la comunicación funcione. El protocolo de comunicación entre la puerta de enlace y el broker MQTT es MQTT y la función principal de la puerta de enlace es la traducción entre el protocolo MQTT-SN y MQTT. El protocolo MQTT-SN está diseñado para poder convivir y funcionar con el protocolo MQTT, en una red podemos tener tanto clientes MQTT-SN como clientes MQTT y podrán comunicarse sin problemas pese a utilizar protocolos diferentes.

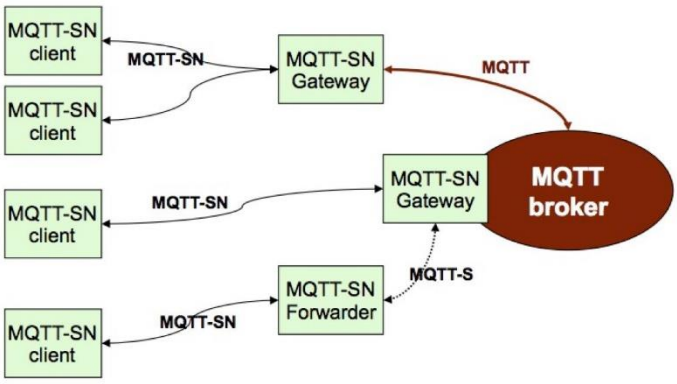

Figura 1: Arquitectura MQTT-SN

2.1.3 Formato de Mensaje

En el protocolo MQTT-SN, los mensajes que se envían tienen un tamaño de carga útil reducido. Los mensajes enviados están compuestos de dos partes:

- Cabecera: su tamaño varía entre 2 o 4 bytes. Siempre está presente y contiene los mismos campos.
  - *Length*: especifica el número total de octetos contenidos en el mensaje (incluido el propio campo Length).
  - *MsgType*: especifica el tipo de mensaje.

- Parte variable: su tamaño puede ser de "n" bytes. Su presencia y contenido depende del tipo de mensaje. Los campos definidos en esta parte variable son:
  - *ClientId, Data, Duration, Flags, GwId, MsgId, ProtocolId, Radius, ReturnCode, TopicId, TopicName, WillMsg, WillTopic.*

Figura 2: Formato mensaje MQTT-SN

2.1.4 Tipos de Mensaje

El protocolo MQTT-SN dispone de 27 tipos de mensaje, mientras que el protocolo MQTT dispone de 14 tipos de mensajes.

La siguiente tabla muestra los tipos de mensajes nuevos del protocolo MQTT-SN.

Tipo Mensaje	MQTT-SN	MQTT
CONNECT	✓	✓
CONNACK	✓	✓
PUBLISH	✓	✓
PUBACK	✓	✓
PUBREC	✓	✓
PUBREL	✓	✓
PUBCOMP	✓	✓
SUBSCRIBE	✓	✓
SUBACK	✓	✓
UNSUBSCRIBE	✓	✓
UNSUBACK	✓	✓
PINGREQ	✓	✓
PINGRESP	✓	✓
DISCONNECT	✓	✓
ADVERTISE	✓	No
SEARCHGW	✓	No
GWINFO	✓	No
WILLTOPICREQ	✓	No
WILLTOPIC	✓	No
WILLMSGREQ	✓	No
WILLMSG	✓	No
REGISTER	✓	No
REGACK	✓	No
WILLTOPICUPD	✓	No
WILLMSGUPD	✓	No
WILLTOPICRESP	✓	No
WILLMSGRESP	✓	No

Figura 3. Tipos de mensaje MQTT-SN

2.1.5 Calidad de Servicio (QoS)

MQTT-SN dispone de cuatro niveles de calidad de servicio:

- QoS 0: los mensajes se envían sin garantía de entrega, no hay confirmación de recepción. Es el nivel por defecto.
- QoS 1: garantiza que un mensaje se entregue al menos una vez al receptor. El remitente almacena el mensaje hasta que recibe un mensaje PUBACK del receptor que acusa haber recibido el mensaje. Es posible que un mensaje se envíe o entregue varias veces.
- QoS 2: garantiza que cada mensaje sea recibido solo una vez por los destinatarios previstos. Se utilizan los mensajes PUBREC, PUBREL, PUBCOMP para dicha garantía. Es el nivel más seguro y lento.
- QoS 3: también denominado QoS -1. Este modo de publicación de mensajes no requiere de una conexión inicial (CONNECT) y requiere el uso de nombres de tema cortos o ID de tema predefinidos. Es ideal para sensores simples.

2.1.6 Proceso de conexión

Un cliente MQTT-SN necesita configurar una conexión a una puerta de enlace antes de poder intercambiar información. De modo que el cliente envía un mensaje CONNECT a la puerta de enlace. La conexión que se establece es UDP y el puerto utilizado se puede configurar.

Si la bandera Will del mensaje CONNECT está activa, la puerta de enlace solicita al cliente la transferencia del topic Will. El cliente contesta con dicha información y a continuación la puerta de enlace solicita el mensaje Will al cliente, entonces el cliente contesta con dicha información y el procedimiento finaliza con el mensaje CONNACK enviado por la puerta de enlace.

Si la bandera Will no está activa, la puerta de enlace responde directamente con un mensaje CONNACK.

Si la puerta de enlace no puede aceptar la solicitud de conexión, por la congestión o no admite alguna función indicada en el mensaje CONNECT, devuelve un mensaje CONNACK con el motivo del rechazo.

De este modo queda establecida la conexión entre un cliente y la puerta de enlace hasta que el cliente envié un mensaje de desconexión (DISCONNECT) o se desconecte de forma repentina. Mientras exista la conexión entre ambos, el cliente puede publicar mensajes, suscribirse o cancelar la suscripción a un topic.

Figura 4. Proceso de conexión MQTT-SN

2.2 Dispositivos hardware

En este apartado se especifican los dispositivos hardware utilizados en el desarrollo del proyecto junto con sus características.

2.2.1 Raspberry Pi

Raspberry Pi es un ordenador de placa simple (Single Board Computer) de bajo costo que dispone de un procesador Broadcom, memoria RAM, GPU, puertos USB, HDMI, Ethernet, un conector para cámara y 40 pines GPIO. Tiene una ranura para tarjeta MicroSD que sirve como unidad de almacenamiento y es donde se instala el sistema operativo.

La disposición de pines GPIO hace que la Raspberry Pi hace que sea compatible con distintos tipos de sensores. Se puede utilizar protocolos de comunicación como el MQTT y debido a su bajo coste se ha popularizado en el ámbito del Internet de las Cosas.

Figura 5. Raspberry Pi 3B+

En el proyecto se utiliza el modelo "Raspberry Pi 3B+". En la siguiente tabla se especifican sus características:

Sistema en Chip	Broadcom BCM2837B0
CPU	1.4GHz 64-bit quad-core ARMv8
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, MPEG-2 y VC-1, 1080p30 H.264/MPEG-4 AVC
Memoria	1GB LPDDR2 SDRAM
Conectividad de Red	Gigabit Ethernet sobre USB 2.0 (300 Mbps)
Conectividad Inalámbrica	Wi-Fi + Bluetooth: 2.4GHz y 5GHz IEEE 802.11.b/g/n/ac, Bluetooth 4.2, BLE
Puertos	GPIO de 40 pines HDMI 4 puertos USB 2.0 Puerto CSI para conectar una cámara. Puerto DSI para conectar una pantalla táctil Salida de audio estéreo y vídeo compuesto Alimentación Micro USB o puerto GPIO Power-over-Ethernet (PoE) Almacenamiento Micro-SD
Fecha lanzamiento	Marzo 2018
Precio	44,98 €

Figura 6. Características Raspberry Pi

2.2.2 NodeMCU ESP8266

El NodeMCU v3 es una placa de desarrollo de código abierto basada en el chip ESP8266, que combina un microcontrolador y un módulo Wi-Fi en un solo dispositivo. Se utiliza para proyectos de Internet de las cosas (IoT) debido a su bajo costo, facilidad de uso y capacidad para conectarse a redes inalámbricas.

La placa tiene pines de entrada/salida que permiten conectar sensores, actuadores y otros dispositivos electrónicos. Además, dispone de una interfaz USB para la alimentación y la programación, lo que hace que sea muy conveniente para cargar y ejecutar el código en la placa.

Para programar la placa se puede utilizar el lenguaje de programación Arduino o el lenguaje de script Lua.

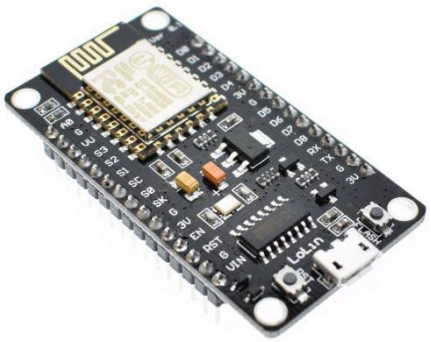

Figura 7: Microcontrolador NodeMCU ESP8266

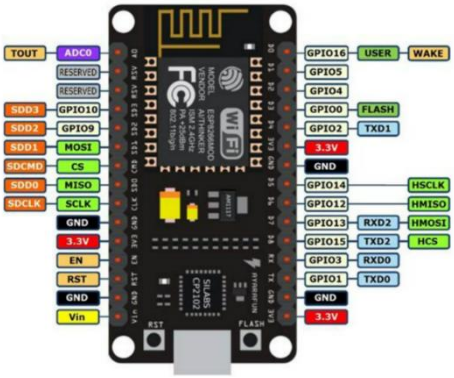

Figura 8. Distribución de pines NodeMCU ESP8266

2.2.3 ESP32-WROOM-32

El chip ESP32-WROOM-32 es un microcontrolador de bajo costo y alta versatilidad, diseñado especialmente para proyectos de Internet de las cosas (IoT). Se usa mucho por su potencia de procesamiento, capacidad de conexión y bajo consumo de energía.

Cuenta con un microprocesador de doble núcleo y una velocidad de reloj de hasta 240 MHz, lo que le brinda una gran capacidad de procesamiento y rendimiento. Además, posee una memoria RAM integrada para almacenar y ejecutar programas de manera eficiente.

Tiene soporte para Wi-Fi que permite que se conecte a redes inalámbricas y pueda enviar y recibir datos a través de Internet. También tiene soporte para Bluetooth, lo que facilita la comunicación con otros dispositivos, como teléfonos móviles o sensores externos.

El chip tiene una gran variedad de pines de entrada/salida, por lo que puedes conectar sensores, actuadores y otros dispositivos electrónicos para realizar tareas. Además, dispone de interfaces adicionales, como SPI, I2C y UART, que permiten la comunicación con otros dispositivos externos.

La programación del ESP32-WROOM-32 se realiza utilizando el entorno de desarrollo de Arduino o el lenguaje de programación C/C++.

Figura 9. Microcontrolador ESP32

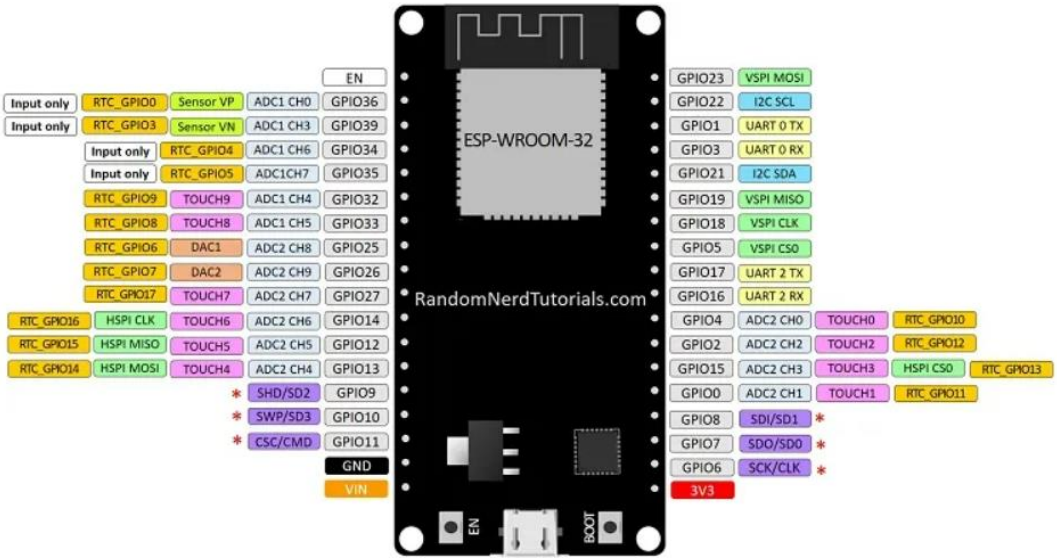

Figura 10. Distribución de pines ESP32

2.2.4 Sensor DHT11

El sensor DHT11 es un sensor de medida de temperatura y humedad muy simple y de bajo costo. Envía datos en formato digital que son leídos por un microcontrolador, como el NodeMCU o el ESP32. Solo necesita tres conexiones para funcionar:

- Alimentación (VCC).
- Tierra (GND).
- Comunicación de datos (DATA).

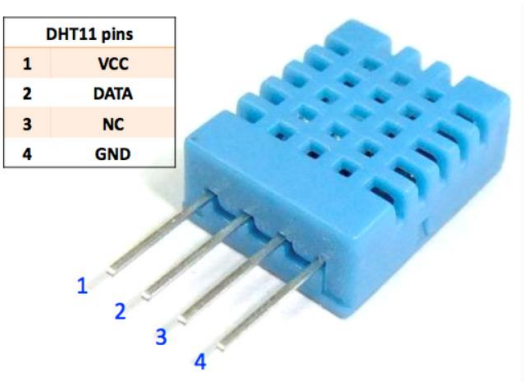

Figura 11: Sensor DHT11

2.2.5 Sensor DHT22

El sensor DHT22 es una versión mejorada del sensor DHT11. También es un sensor digital y utiliza una única conexión de datos para comunicarse con un microcontrolador.

El rango de medición es más amplio en comparación con el DHT11, puede medir temperaturas en un rango de -40 a 125 grados Celsius, y la humedad relativa en un rango de 0% a 100%.

Pin	Función
1	Vcc
2	Datos
3	Sin conexión
4	GND

Figura 12: Sensor DHT22

2.2.6 Sensor HC-SR04

El sensor de ultrasonidos HC-SR04 es un dispositivo muy utilizado para medir distancias usando ondas de sonido. Es económico, fácil de usar y ofrece buena precisión.

El HC-SR04 funciona emitiendo un pulso de ultrasonidos y midiendo el tiempo que tarda en recibir el eco de vuelta. El sensor tiene dos componentes principales: un transmisor y un receptor de ultrasonidos.

Con base en el tiempo que tarda en recibir el eco, se puede calcular la distancia entre el sensor y el objeto. Esto se logra con una fórmula simple que considera la velocidad del sonido en el aire y el tiempo transcurrido.

Para utilizar el HC-SR04, se necesitan al menos dos pines en un microcontrolador, uno para enviar la señal de activación y otro para recibir el eco.

Figura 13: Sensor HC-SR04

2.2.7 Pantalla LCD

Una pantalla LCD 16x2 es un tipo de dispositivo de visualización que consta de 16 caracteres en cada una de las dos filas. Utiliza tecnología de cristal líquido (LCD) para mostrar caracteres alfanuméricos y otros símbolos.

Para controlar la pantalla se utilizan pines de control que permiten enviar comandos y datos. Algunos de los pines comunes son:

- VCC y GND: Estos pines se utilizan para suministrar energía a la pantalla, donde VCC es el pin de voltaje positivo y GND es el pin de tierra.
- RS (Registro de selección): Este pin se utiliza para seleccionar entre enviar datos o comandos a la pantalla. Se establece en alto (1) para datos y en bajo (0) para comandos.
- E (Enable): Este pin se utiliza para habilitar la lectura o escritura de datos en la pantalla.
- D0-D7 (Datos): Estos pines se utilizan para enviar datos a la pantalla LCD. Dependiendo del modo de operación seleccionado (4 bits o 8 bits), se utilizan diferentes combinaciones de estos pines.

Además, la pantalla también tiene un potenciómetro ajustable para controlar el contraste de la pantalla y una luz de fondo que mejora la visibilidad en condiciones de poca luz.

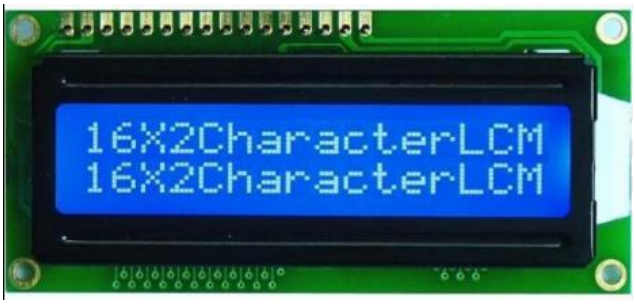

Figura 14: Pantalla LCD 16x2

2.2.8 Micro servomotor

El micro servo SG90 es un pequeño dispositivo electromecánico. Funciona como un actuador que convierte señales eléctricas en movimiento físico controlado. Es conocido por su tamaño compacto, bajo consumo de energía y precio económico. Tiene un rango de operación típico de 0 a 180 grados, lo que le permite realizar movimientos angulares precisos.

En cuanto a los pines del micro servo SG90 tiene tres cables:

- Cable de alimentación (VCC): Se conecta a la fuente de alimentación de 5V para suministrar energía al servo.
- Cable de tierra (GND): Se conecta al terminal de tierra para cerrar el circuito eléctrico.
- Cable de control (señal): Este cable se conecta a una salida PWM (Modulación por Ancho de Pulso) de un microcontrolador o una tarjeta de desarrollo. La señal PWM controla la posición angular del servo.

Figura 15: Micro servomotor SG90

2.3 Herramientas de programación

2.3.1 IDE Arduino

El conocido IDE (entorno de desarrollo integrado) de Arduino tiene una interfaz gráfica de usuario sencilla con herramientas de depuración, verificación y carga de código para la mayoría de placas del mercado. De modo que se usará para programar el código funcional de los microcontroladores esp8266 y esp32 mediante conexión USB.

2.3.1.1 Secciones del IDE de Arduino

Para que se posible utilizar el IDE de Arduino con los microcontroladores ESP8266 y ESP32 es necesario incluir dichas tarjetas al gestor del IDE.

Para ello, es necesario incluir las siguientes URL en la configuración del IDE:

- http://arduino.esp8266.com/stable/package_esp8266com_index.json
- https://dl.espressif.com/dl/package_esp32_index.json

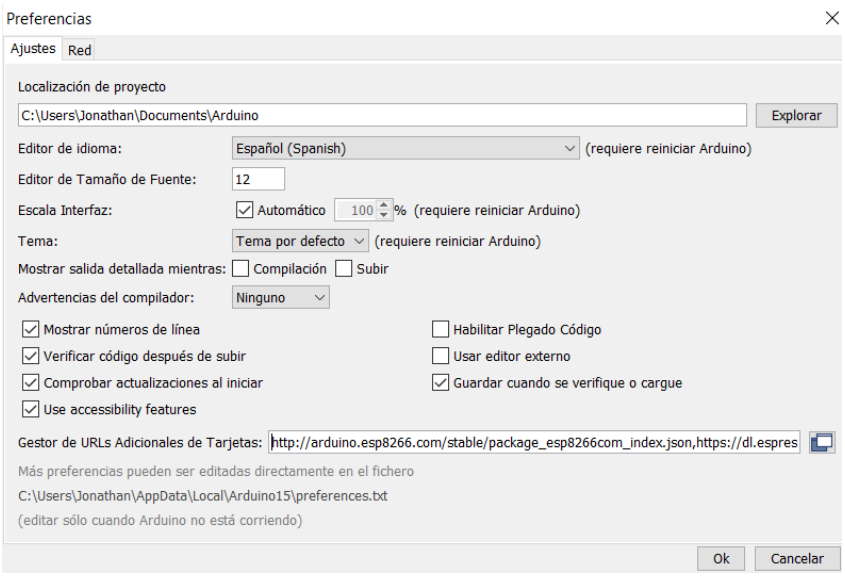

Figura 16: Gestor de URL del IDE de Arduino

En el IDE de Arduino la mayoría de las librerías están preinstaladas, sin embargo, hay la opción de incluir librerías de fuentes externas. El apartado "Programa" se permite incluirlas o también hacerlo de forma manual.

En el proyecto se ha utilizado la librería "arduino-mqtt-sn-client" y los pasos a seguir para su inclusión son los siguientes:

- Descargar la librería.
- Extraer el archivo, cambiar su nombre y copiarlo en el directorio de librerías de Arduino.
- Reiniciar el IDE Arduino.
- Seleccionar la placa ESP8266 en el IDE de Arduino.
- Abrir el ejemplo → arduin-mqtt-sn-client → esp8266 →WiFiUdpMqttSnClient.
- Adaptar el "ssid" y "password" de la red Wi-Fi utilizada.
- Cambiar "gatewayIPAddress" y "localUdpPort" por la dirección IP y el puerto UDP de la puerta de enlace MQTT-SN utilizada.
- Cargar el programa.

2.3.1.2 Programación en el IDE de Arduino

Con el IDE Arduino se pueden escribir programas en lenguaje C/C++ utilizando una estructura básica de setup(), que se ejecuta sólo una vez, y loop(), que se ejecuta todo el tiempo.

En la siguiente figura se muestra un ejemplo de ambas funciones.

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}
```

Figura 17: Estructura setup() y loop()

2.3.2 Node-Red

Node-RED es una herramienta de programación de código abierto que simplifica la programación, ya que permite conectar lo que la herramienta denomina bloques de código (nodos) entre sí para realizar tareas específicas.

Node-Red permite hacer lo siguiente:

- Acceder a las GPIO de diversas placas: arduino, esp8266, esp32, Raspberry Pi, entre otros.
- Establecer conexiones MQTT con otras placas.
- Crear interfaces gráficas para proyectos IoT de forma sencilla.
- Crear eventos que se activen por tiempo.
- Comunicación con servicios de terceros (Adafruit.io, Thing Speak, IFTTT.com, entre otros).
- Capturar datos de la web (correos electrónicos, pronóstico del tiempo, precios de acciones, entre otros).
- Almacenar y recuperar datos de una base de datos.

2.3.2.1 Secciones de Node-Red

En el lado izquierdo se encuentra todos los tipos de bloques (nodos) de la herramienta separados por su funcionalidad. Se puede seleccionar un nodo y en la pestaña de información podemos ver cómo funciona. La herramienta permite instalar nuevos nodos que no estén instalados por defecto e incluso crear nuevos nodos programando.

En el centro de la pantalla se encuentra lo que se conoce como flujo y es donde se colocan los nodos.

En la parte de la derecha tenemos varias pestañas muy útiles. Una de ella es el botón de “Deploy” que debemos pulsar con cada cambio realizado en el flujo de nodos. La pestaña de “Debug” nos permite observar los mensajes que llegan o salen del flujo de nodos.

En la siguiente figura se muestran las partes de la herramienta.

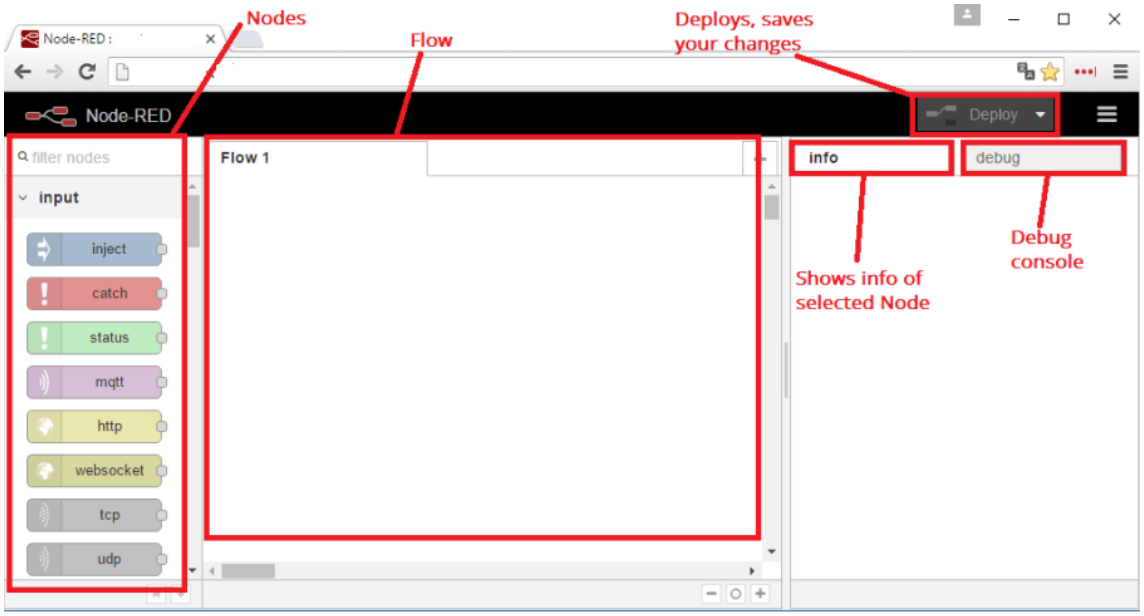

Figura 18: Secciones principales de Node-Red

Como se ha mencionado anteriormente, Node-Red dispone de la posibilidad de crear interfaces gráficas de usuario. Para ello hay que utilizar los nodos de tipo "Dashboard" los cuales proporcionan widgets que se muestran en la interfaz de usuario (UI) de la aplicación.

Esta interfaz de usuario se organiza en pestañas y dentro de cada pestaña existen grupos que dividen las pestañas en diferentes secciones y así poder organizar los widgets. Además, tenemos opciones que nos permiten modificar el formato y estilo de la aplicación web.

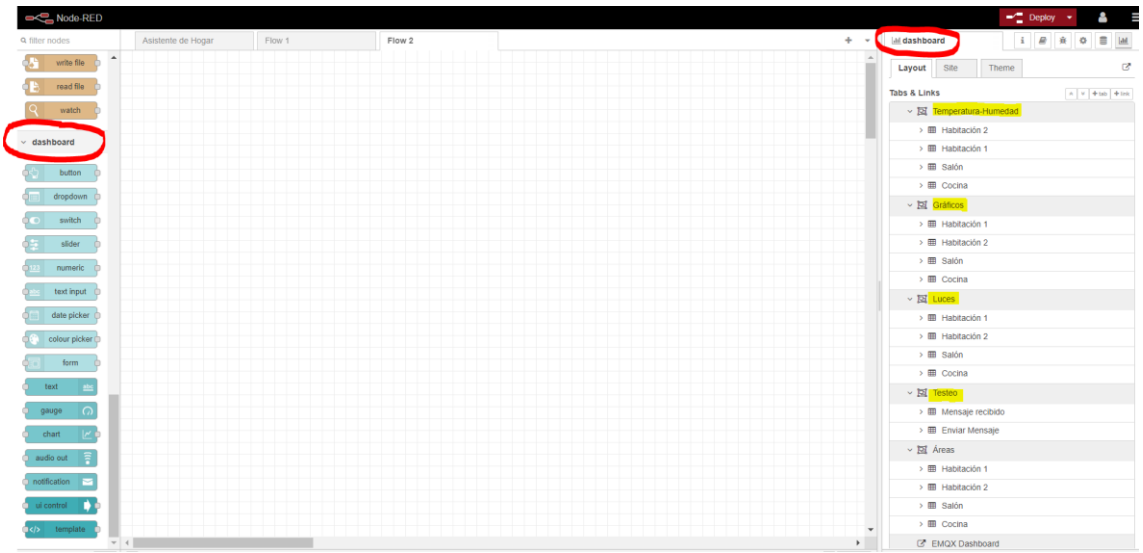

Figura 19: Pantalla principal Node-Red

Capítulo 3. Desarrollo

3.1 Arquitectura de la red desplegada

La arquitectura del proyecto es la siguiente:

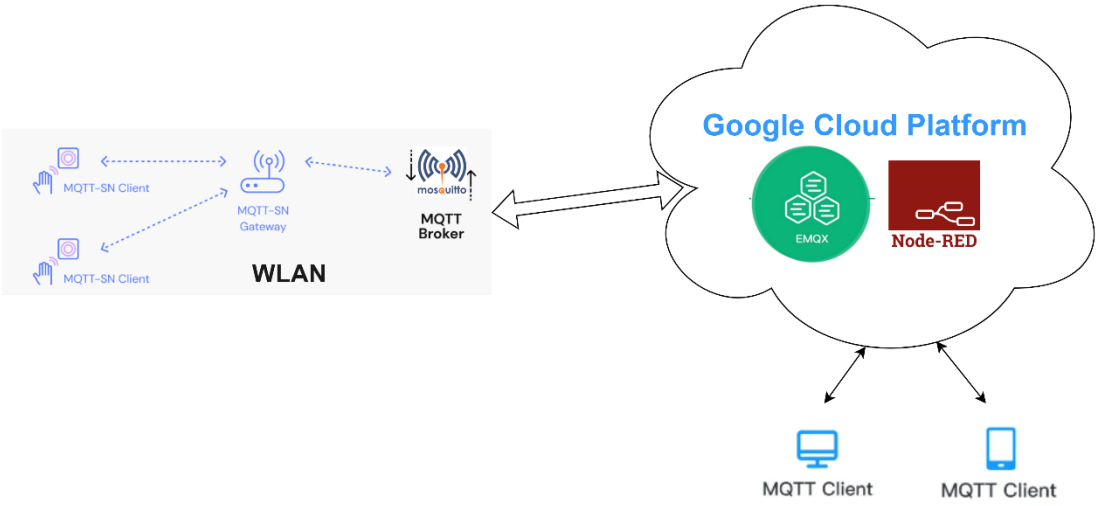

Figura 20: Arquitectura de red desplegada

Los dispositivos sensores (microcontrolador + sensor) son los clientes MQTT-SN de la arquitectura desplegada. Están programados mediante el IDE de Arduino para publicar y suscribirse a diferentes topics predefinidos.

En un mismo dispositivo Raspberry Pi se instala tanto la puerta de enlace transparente (Paho Gateway) como el broker MQTT mosquitto para que el protocolo MQTT-SN opere.

Los clientes se comunican con la puerta de enlace y la puerta de enalce tras hacer la traducción de protocolos se comunica con el broker mosquitto, estableciéndose así una comunicación completa a nivel local basada en el protocolo MQTT-SN.

Debido a que el broker mosquitto no tiene gran capacidad para trabajar en la nube y tampoco dispone de una herramienta gráfica para gestionar los dispositivos sensores, aparece en escena el uso de un broker remoto, EMQX instalado en la nube de Google Cloud Platfomr, que ofrece ambas cosas. Luego, el broker mosquitto actúa como un dispositivo puente que se encarga de reenvíar los datos desde la red local hacia el broker remoto y este hacia la aplicación IoT a desarrollar.

Como la aplicación IoT a desarrollar debe interactuar con los dispositivos sensores desde el exterior, también se instala Node-Red en la nube de Google Cloud Platform, permitiendo de esta forma usar la aplicación IoT desde cualquier dispositivo con conexión a Internet independientemente del tipo protocolo, ya sea MQTT o MQTT-SN.

3.2 Puerta de enlace

La puerta de enlace elegida es la correspondiente a la librería “paho.mqtt-sn.embedded-c”. Es una puerta pura que necesita de un broker externo para funcionar. Se usará en modo transparente lo que significa que configura y mantiene una conexión MQTT con el broker externo para cada

cliente MQTT-SN, es decir, habrá tantas conexiones MQTT entre la puerta de enlace y el broker externo como clientes MQTT-SN haya conectados a la puerta de enlace.

La puerta de enlace actúa como intermediario entre los dispositivos MQTT-SN y el broker MQTT. Recibe los mensajes enviados por los dispositivos MQTT-SN y los traduce al formato compatible con MQTT convencional, ya que este es el estándar ampliamente utilizado para la comunicación en Internet de las Cosas (IoT).

De manera similar, cuando el servidor MQTT envía mensajes a través de la puerta de enlace transparente, este los convierte al formato MQTT-SN y los transmite a los dispositivos MQTT-SN de destino.

El dispositivo en el que se instala tanto la puerta de enlace como el broker mosquitto es la raspberry pi 3B+. Debido a que su sistema operativo es una distribución de Linux basada en Debian se puede hacer uso de comandos Linux para instalar y administrar paquetes de software, navegar por el sistema de archivos, copiar, mover y eliminar archivos, configurar redes, administrar usuarios y permisos, entre otras tareas.

3.3 Clientes MQTT-SN

En este apartado se muestran los clientes MQTT-SN utilizados en el proyecto. Su ubicación dentro del hogar y los topics a los que envían o reciben los mensajes.

3.3.1 NodeMCU ESP8266 (salón)

Este microcontrolador está ubicado en el salón, tiene conectado a sus pines un sensor DHT22 del cual leerá los valores de temperatura y humedad y los publicará en los topics predefinidos:

- 2 → correspondiente a *salon/sensor/temperatura*.
- 3 → correspondiente a *salon/sensor/humedad.*

También tiene conectado a sus pines un led que podremos encender/apagar, ya que se suscribe al topic predefinido:

- 1 → correspondiente a *salon/control/luz.*

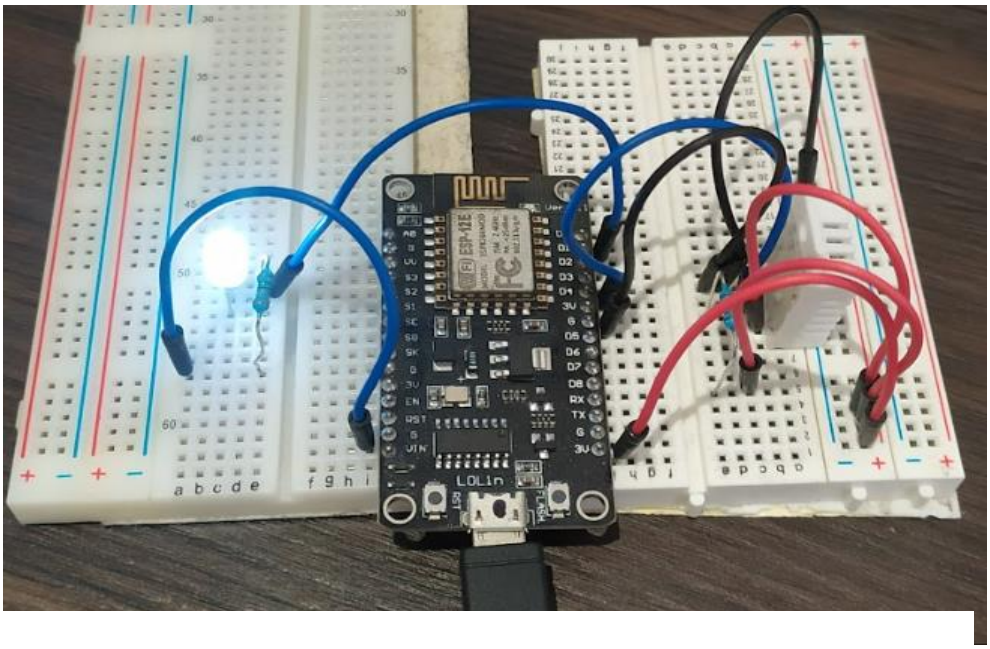

Figura 21: Conexionado NodeMCU ESP8266

3.3.2 ESP8266 (cocina)

Este segundo microcontrolador ubicado en la cocina tiene conectado a sus pines un sensor DHT11 del cual leerá los valores de temperatura y humedad y los publicará en los topics predefinidos:

- 5 → correspondiente a *cocina/sensor/temperatura*.
- 6 → correspondiente a *cocina/sensor/humedad*.

También tiene conectado a sus pines una pantalla LCD a la que enviaremos mensajes. Para ello está suscrito al topic predefinido:

- 4 → correspondiente a *cocina/control/mensaje*.

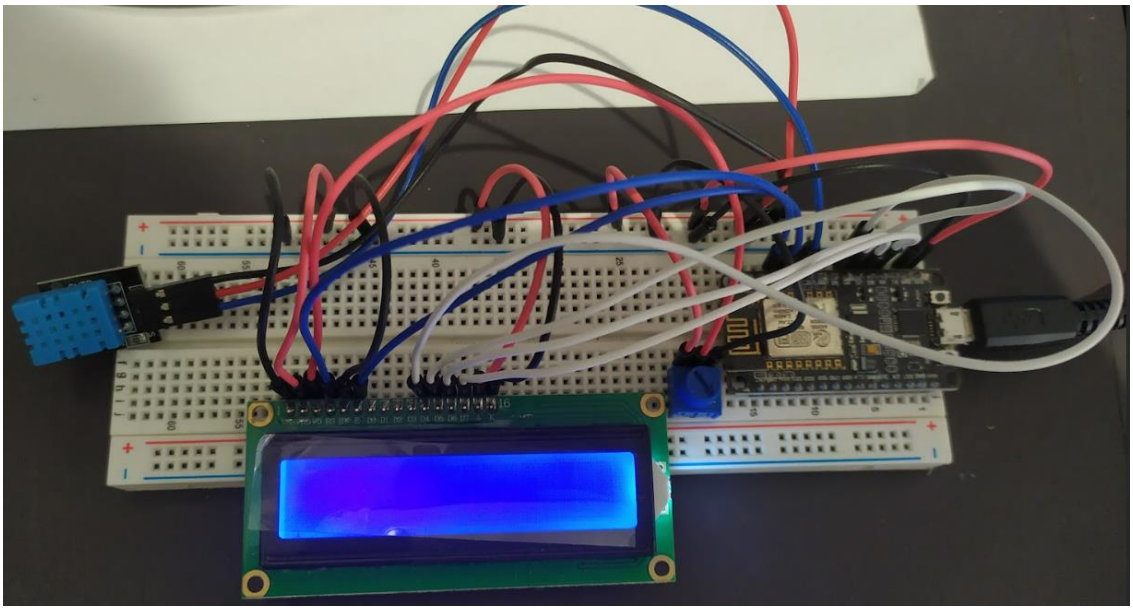

Figura 22: Conexionado ESP8266

3.3.3 ESP32_1 (habitación 1)

Este primer microcontrolador ESP32 ubicado en una habitación, tiene conectado a sus pines un sensor de distancia por ultrasonidos HC-SR04 del cual leerá valores de distancia del objeto que tiene delante y encenderá un led cuando esa distancia sea menor a un valor que programamos (en este caso 15 cm). Está información de distancia la publicará en el topic predefinido:

- 12 → correspondiente a *habitacion1/sensor/distancia*.

También de forma hard-code (datos directamente incrustados en el código fuente del programa) publicará valores aleatorios al topic predefinido:

- 11 → correspondiente a *habitacion1/sensor/temperatura.*

Por último, estará suscrito al topic predefinido:

- 10 → correspondiente a *habitacion1/control/luz.*

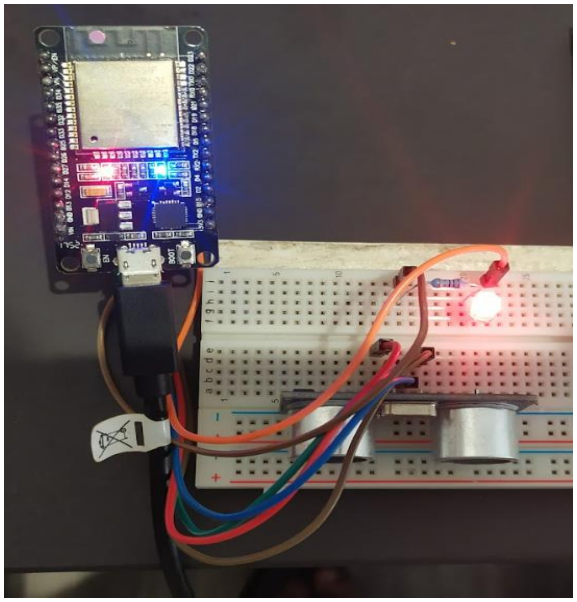

Figura 23. Conexionado ESP32_1

3.3.4 ESP32_2 (habitación 2)

Este segundo microcontrolador ESP32 ubicado en otra habitación, publicará de forma hard-code valores aleatorios en el topic predefinido:

- 14 → correspondiente a *habitacion2/sensor/temperatura*.

Tiene conectado en sus pines un micro servomotor SG90 que podrá ser activado al estar suscrito al topic predefinido:

- 13 → correspondiente a *habitacion2/control/luz*.

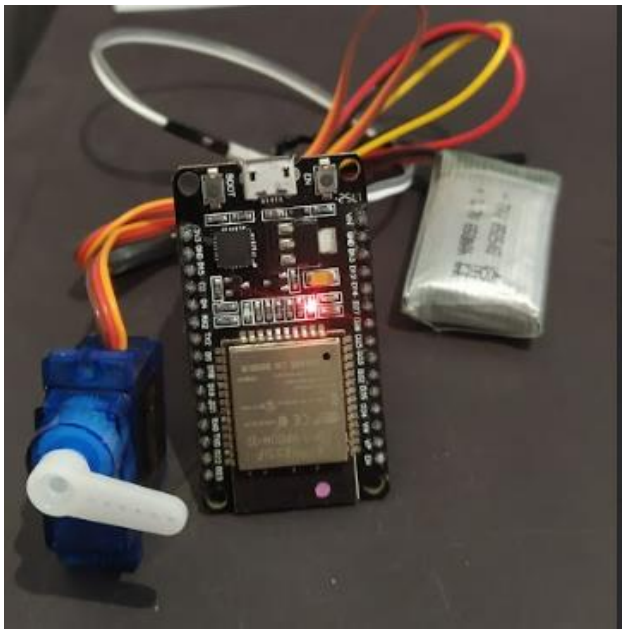

Figura 24. Conexionado ESP32_2

3.3.5 Raspberry Pi

Debido a las características de mini ordenador que tiene este dispositivo podemos ejecutar en él programas escritos en Python o incluso utilizar la consola de comandos como clientes MQTT-SN.

De modo que se ha instalado en este dispositivo una herramienta para el terminal de línea de comandos (CLI). Esta herramienta permite hacer publicaciones y suscripciones mediante protocolo MQTT-SN de una forma rápida y fácil y así poder testear el protocolo sin necesidad de microcontroladores y sensores.

En las siguientes figuras se muestran los parámetros necesarios para los comandos de publicación y suscripción.

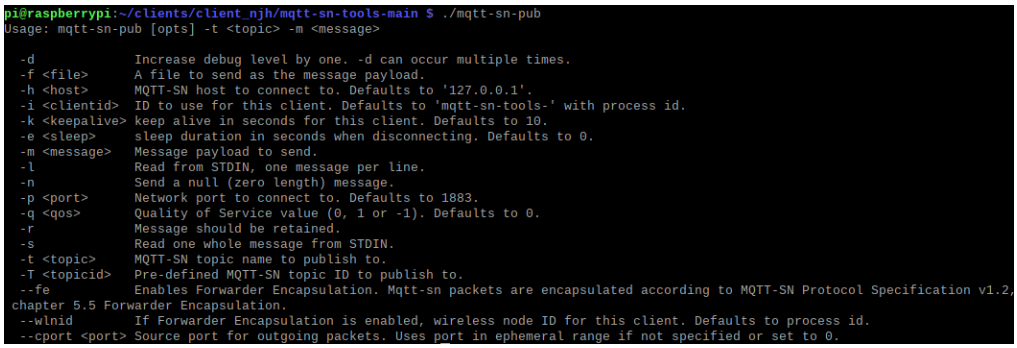

Figura 25: Parámetros publicación mensaje MQTT-SN

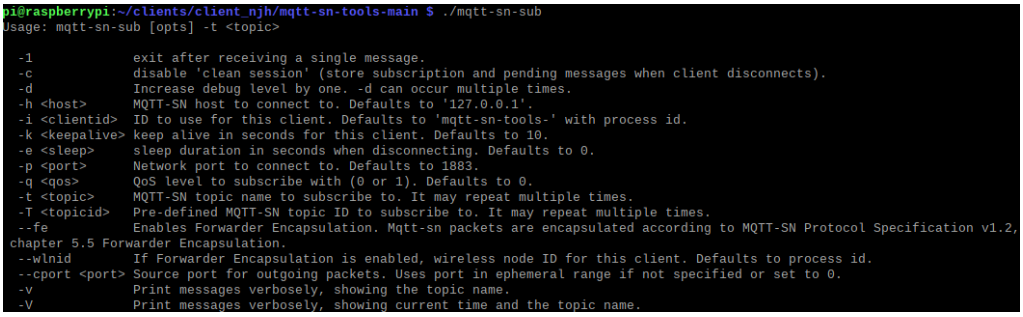

Figura 26: Parámetros suscripción a mensaje MQTT-SN

Los resultados de las pruebas con esta herramienta se muestran en el apartado de “Pruebas de desarrollo”.

3.3.6 MQTT Dash

Este es un cliente basado en el protocolo MQTT. Se puede conseguir descargándolo en un dispositivo móvil, ya que es una aplicación móvil gratuita para Android.

Se utiliza en el proyecto para mostrar como la comunicación entre dispositivos de distinto protocolo pueden trabajar juntos en una red MQTT-SN.

Los resultados de las pruebas con esta aplicación móvil se muestran en el apartado de “Pruebas de desarrollo”.

3.4 Broker Mosquitto

Es el punto central de la arquitectura de red local desplegada, ya que proporciona la interoperabilidad y la infraestructura necesaria para la comunicación MQTT entre los clientes MQTT-SN y otros clientes MQTT. La opción de configurar el broker mosquitto como un puente, permite la comunicación con un broker remoto desplegado en la nube.

Los pasos para su instalación y configuración se encuentran en el apartado de anexos de este proyecto.

3.5 Google Cloud Platform

Google Cloud Platform (GCP) es una plataforma de servicios alojada en la nube ofrecida por Google.

Para este proyecto se utiliza la herramienta Google Compute Engine que sirve para crear y administrar máquinas virtuales (VM) en la infraestructura de Google. De esta forma se puede desplegar en ella el broker remoto EMQX y la herramienta de programación Node-Red y hacer accesible la aplicación IoT a desarrollar desde cualquier red y dispositivo.

Para hacer uso de esta herramienta es necesario tener una cuenta de correo de Google y registrarse en la plataforma. Se dispone de 90 días y de un crédito de 300 dólares para acceder a los distintos servicios de forma gratuita.

Los pasos para su instalación y configuración se encuentran en el apartado de anexos de este proyecto.

3.6 Broker EMQX

Es el broker remoto elegido para el proyecto por su alto rendimiento en el procesamiento de mensajes en tiempo real y una garantía de entrega eficiente y segura de los datos. Su función es la de intermediario entre los dispositivos IoT.

Además, dispone de un panel de control que facilita la gestión de los dispositivos y el seguimiento de distintos indicadores que están relacionados con la comunicación MQTT y MQTT-SN.

Los pasos para su instalación y configuración se encuentran en el apartado de anexos de este proyecto.

3.6.1 Acceso al panel de control de EMQX

Una vez el broker EMQX está arrancado, se puede acceder a la herramienta Dashboard escribiendo en el navegador web la ruta IP de la máquina virtual y el puerto destinado al Dashboard.

- http://34.116.209.205:18083.

El nombre de usuario predeterminado es “admin” y la contraseña es “public”, la cual se cambia posteriormente para mayor seguridad.

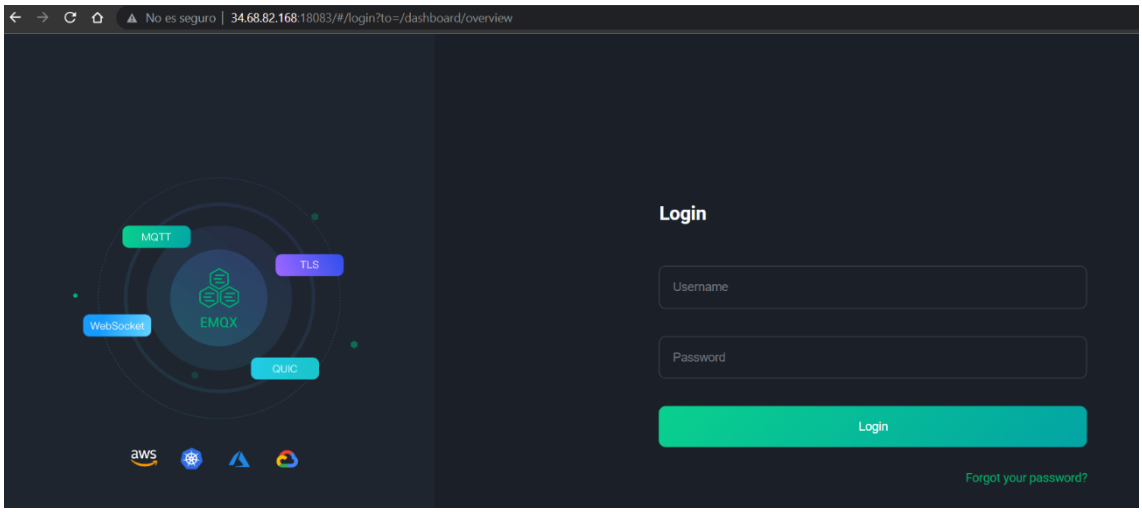

Figura 27. Acceso a panel de control de EMQX

Una vez hecho el login, la siguiente figura muestra la vista general del panel de control.

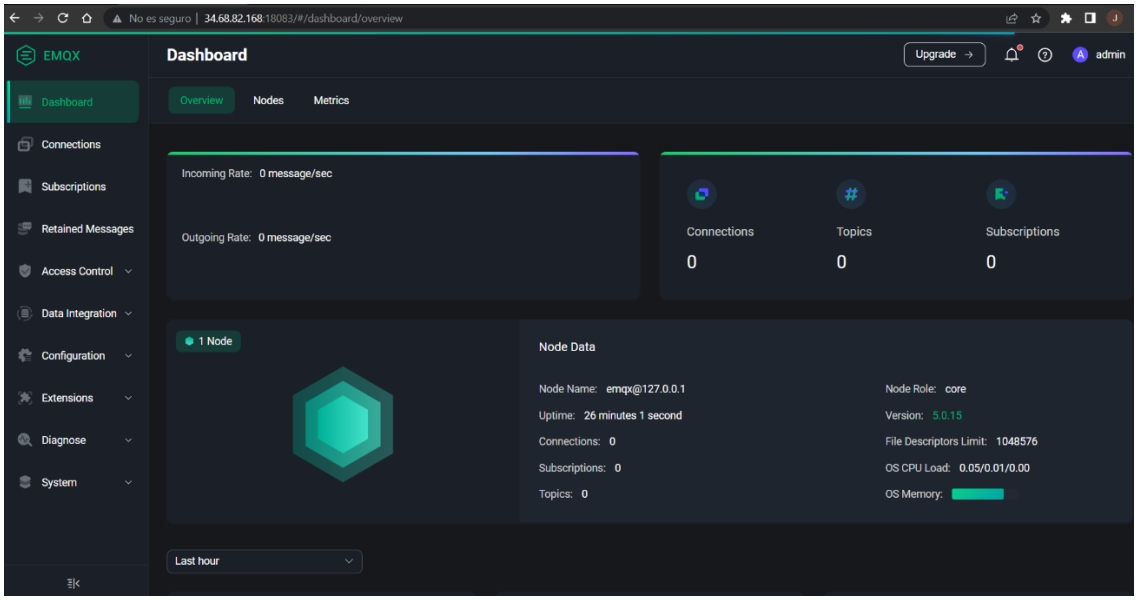

Figura 28. Vista general del panel de control EMQX

Como se observa, el panel tiene muchas opciones que ofrecer. A través de este podemos ver y utilizar:

- Información básica del servidor.
- Información de la carga y datos estadísticos.
- El estado de conexión de un cliente e incluso desconectarlo.
- Cargar y descargar complementos específicos de forma dinámica.

- Operación visual del motor de reglas.
- Herramienta de un cliente MQTT simple para pruebas de usuario.

3.7 Node-Red

Es la herramienta utilizada para programar la aplicación web IoT mediante programación por bloques de nodos.
Los pasos para su instalación y configuración se encuentran en el apartado de anexos de este proyecto.

3.7.1 Acceso a Node-Red

Se escribe en el navegador web la ruta de la máquina virtual utilizada y el puerto donde está el servicio Node-Red.

Para acceder a la parte de interfaz gráfica de usuario, se puede acceder desde la parte de gestión directamente o escribiendo la ruta en el navegador web. Para ambas partes se necesitan las credenciales establecidas en el paso de configuración

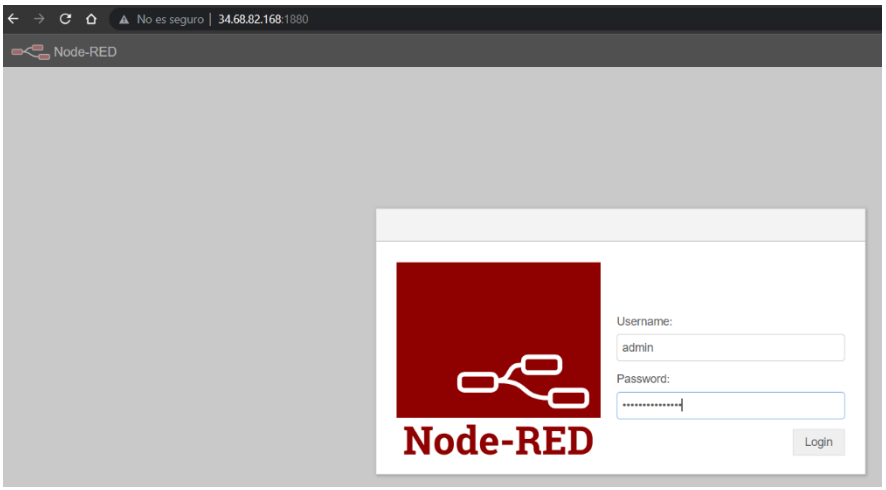

Figura 29. Acceso a la programación de nodos de Node-Red

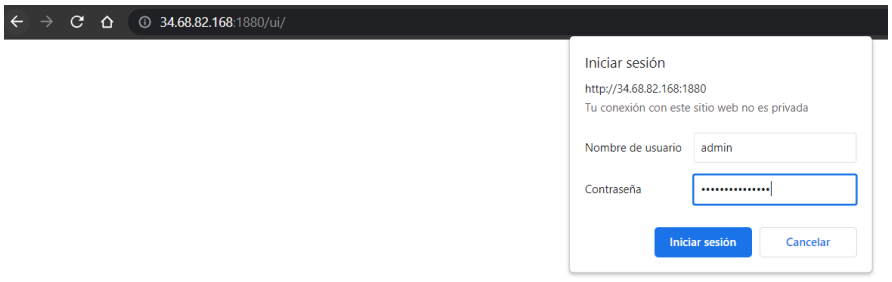

Figura 30. Acceso a la interfaz gráfica de Node-Red

En la siguiente figura se muestra la parte de gestión de la herramienta de programación basada en bloques de nodos.

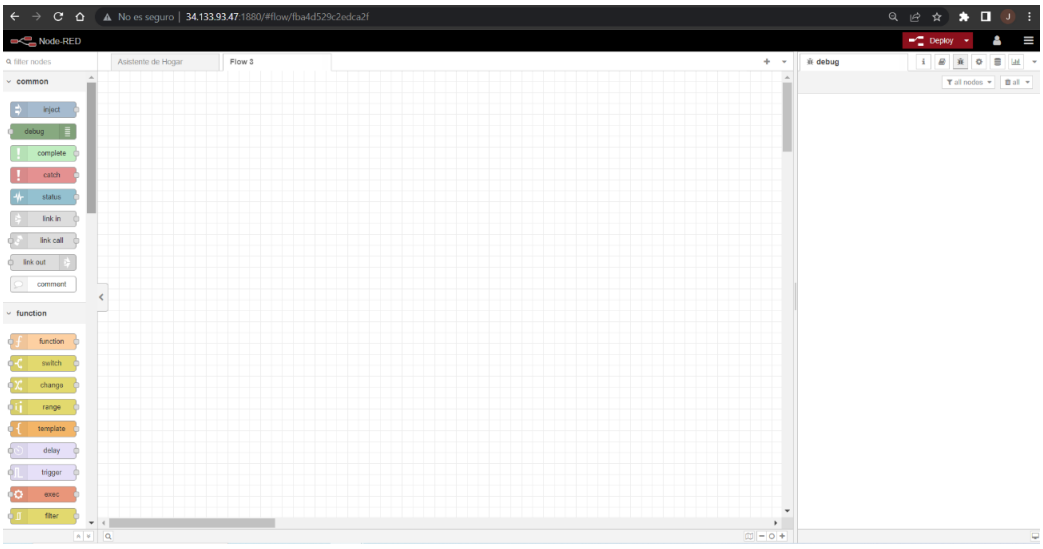

Figura 31. Pantalla principal de Node-Red

3.7.2 Programación de bloques de nodos

Para la programación de la aplicación web por nodos se han utilizado los nodos de tipo “Network” que admiten la funcionalidad MQTT, los nodos de tipo “Common” para poder ver los datos que llegan y salen en la consola de debug y los nodos de tipo “Dashboard” para crear los widgets de la interfaz gráfica de usuario.

Para que la interfaz gráfica este dividida en pestañas se deben agrupar los distintos widgets por grupos y secciones. Esto se hace desde la pestaña “Dashboard” en la parte derecha de la herramienta.

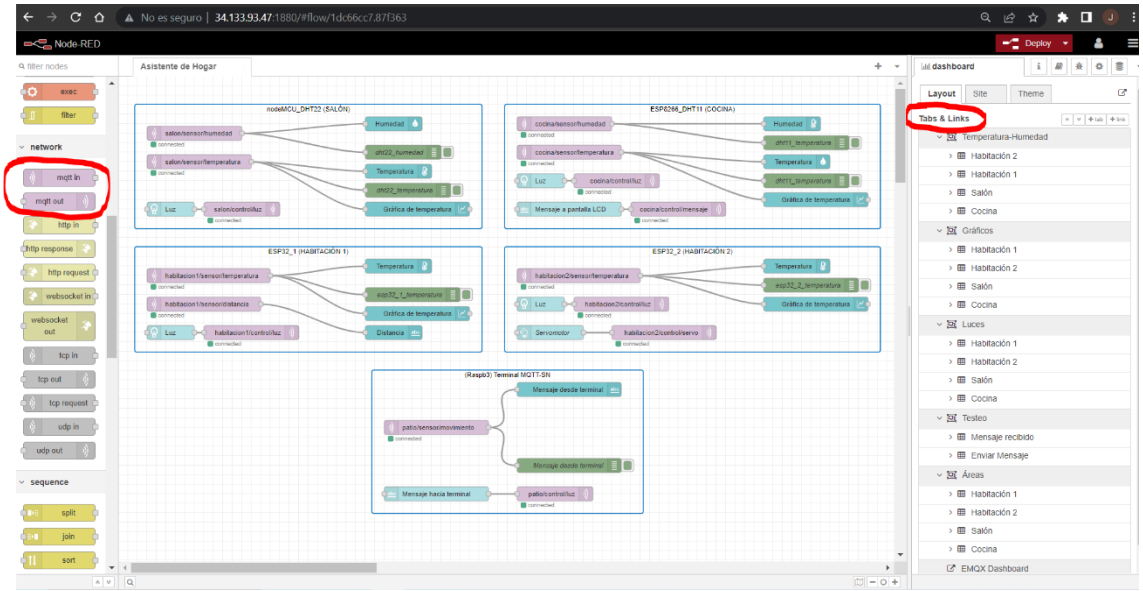

Figura 32. Tipos de nodos de Node-Red

A continuación, se muestran los detalles de un grupo de nodos solamente, ya que los demás nodos de la aplicación siguen el mismo patrón de configuración y programación.

La primera parte del bloque de conexión de nodos se encarga de recibir los datos que llegan del microcontrolador NodeMCU esp8266 de la lectura de su sensor DHT22 al topic *salon/sensor/temperatura* y al topic *salon/sensor/humedad.*

La segunda parte del bloque se encarga de enviar la orden de encender o apagar el led conectado al mismo microcontrolador publicando en el topic *salon/control/luz*.

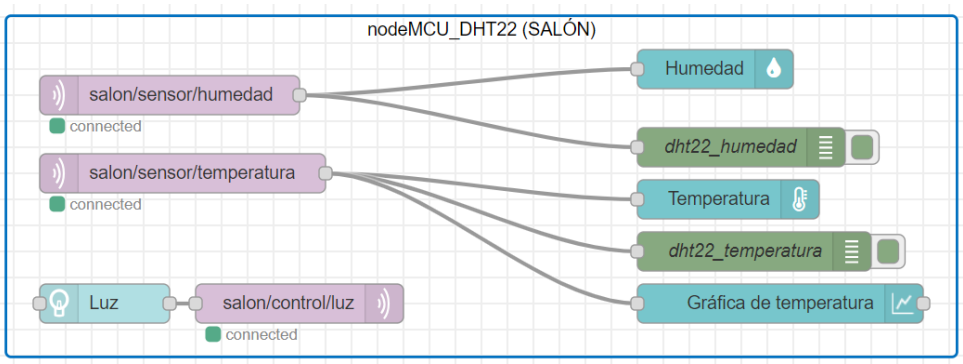

Figura 33. Bloque de nodos para el microcontrolador NodeMCU

En cuanto a la configuración del nodo MQTT para poder publicar y suscribirse, los parámetros necesarios son siguientes:

- Dirección IP del servidor broker EMQX.
- Puerto del servidor broker EMQX.
- Versión del protocolo MQTT.
- El keep alive de la conexión
- Sesión limpia.
- Acción de suscripción o publicación.
- Topic.
- Calidad de servicio (QoS).

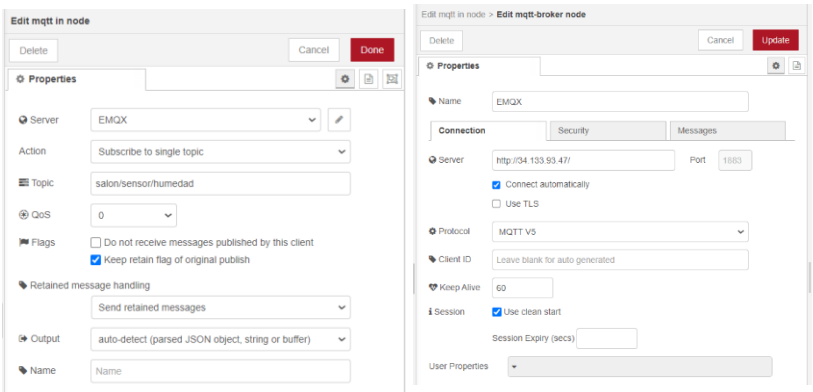

Figura 34. Configuración nodo network

Con los nodos de tipo “Common” se pueden visualizar los datos en la pestaña debug y su configuración es la siguiente:

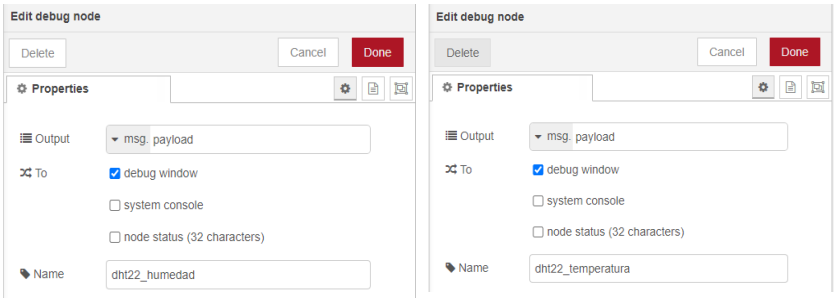

Figura 35. Configuración nodo common

Con los nodos de tipo "Gauge" se puede visualizar la temperatura y humedad en formato indicador y su configuración es la siguiente

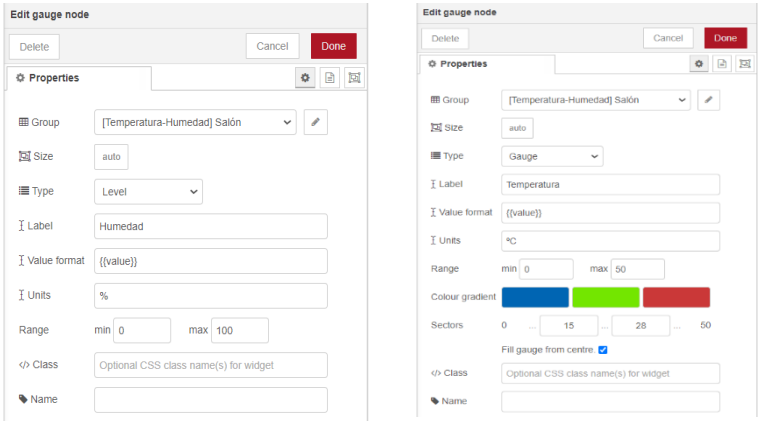

Figura 36. Configuración nodo gauge

Con el nodo de tipo "chart" se pueden visualizar los datos en formato gráfica y su configuración es la siguiente:

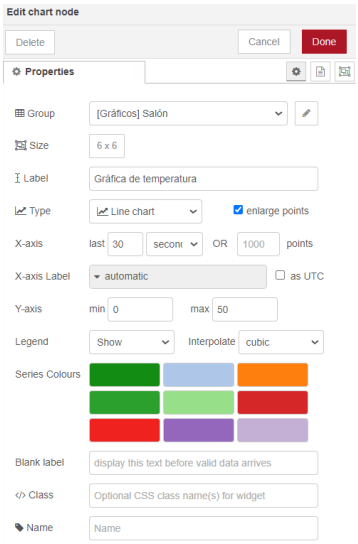

Figura 37. Configuración nodo chart

Con el nodo de tipo “Switch” se puede encender o apagar los leds simplemente pulsando un switch y su configuración es la siguiente:

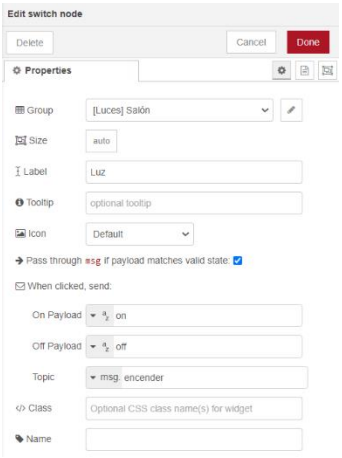

Figura 38. Configuración nodo switch

Por último, los nodos se deben ordenar en grupos y secciones para su correcta distribución en la interfaz gráfica de usuario. La configuración es la siguiente:

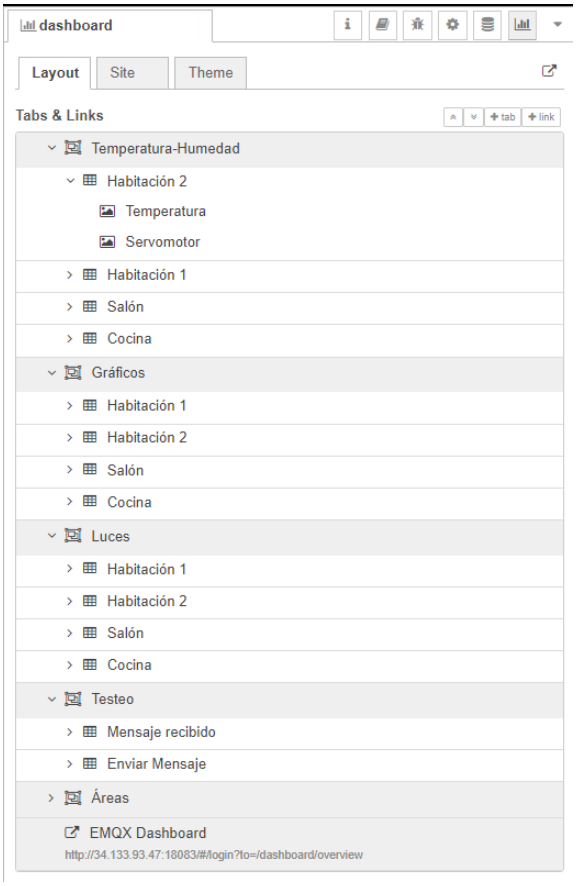

Figura 39. Configuración panel de distribución

Finalizada la configuración de cada nodo widget, se accede a la interfaz de usuario de la aplicación y se observa el resultado final. Un panel amplio, dividido en secciones y cada sección proporciona información con distintos tipos de widgets.

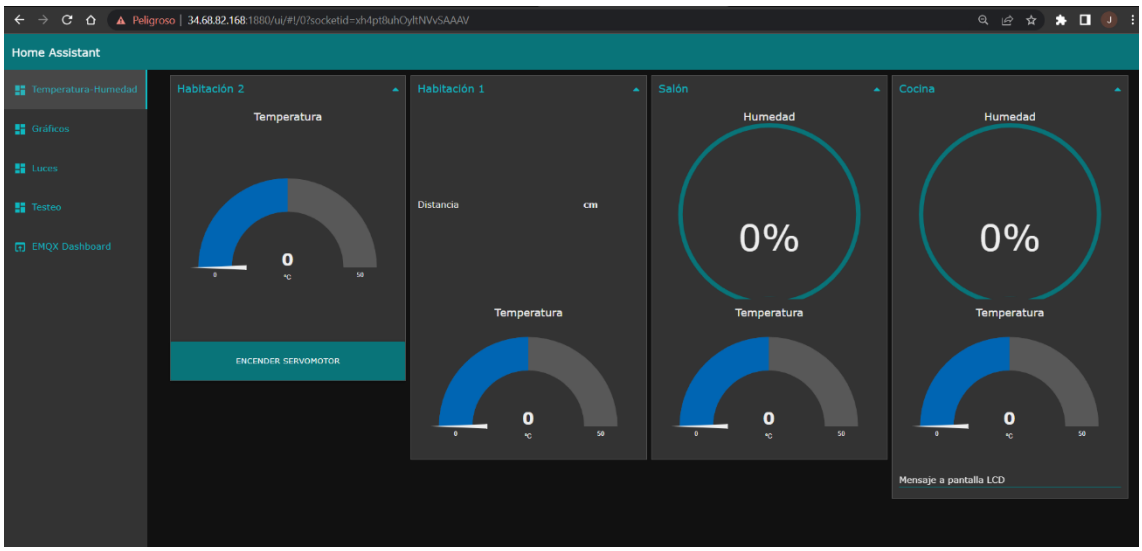

Figura 40. Pantalla principal de la aplicación IoT

En el apartado "Gráficos" se observan las temperaturas en formato gráfica. Las dos primeras corresponden a los datos de los microcontroladores ESP32 y las siguientes a los sensores DHTT22 y DHT11, respectivamente.

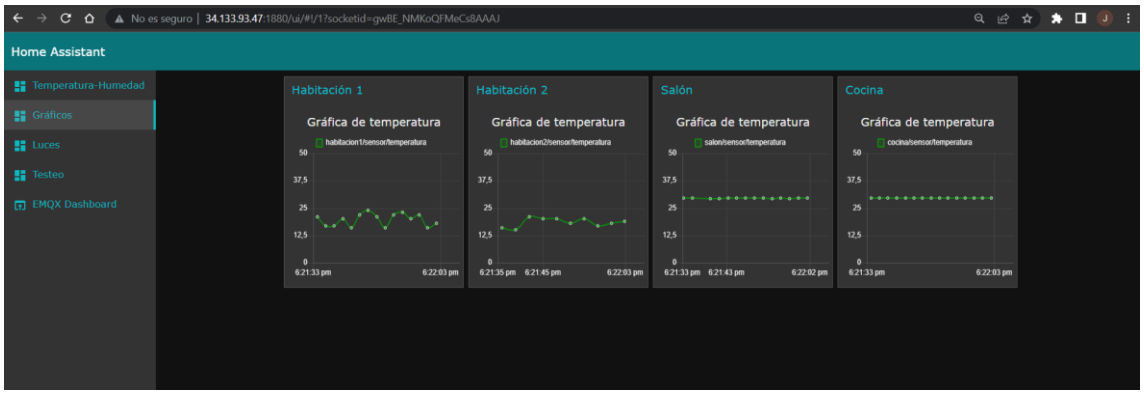

Figura 41. Pantalla gráficos de la aplicación IoT

En el apartado “Luces” se pueden ver los switches para encender/apagar los leds que hacen de bombilla.

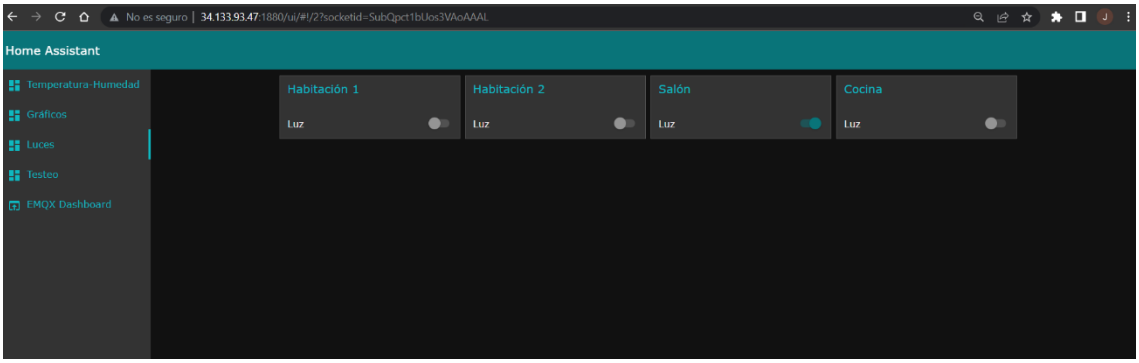

Figura 42. Pantalla luces de la aplicación IoT

En el apartado de “Testeo” se pueden enviar o recibir mensajes desde la aplicación hacia diferentes tipos de clientes, tanto MQTT-SN como MQTT.

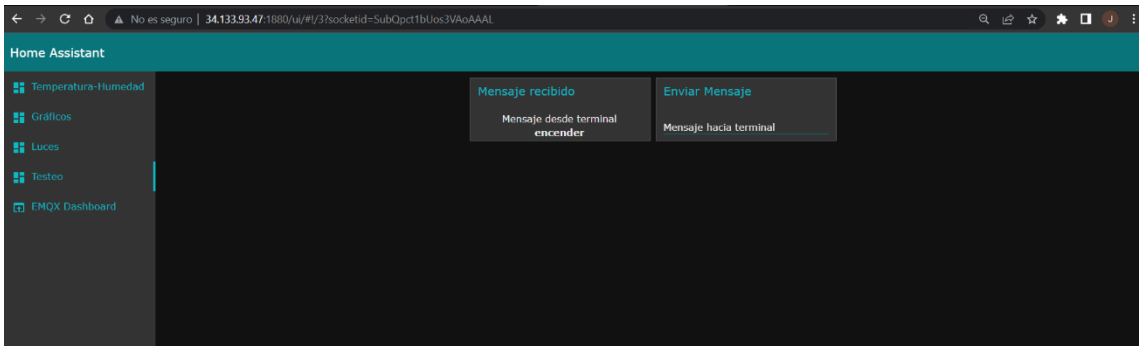

Figura 43. Pantalla testeo de la aplicación IoT

Por último, en el apartado “EMQX Dashboard” tenemos un enlace de navegación hacia el Dashboard de control de EMQX. Este panel es muy interesante porque aporta información más detallada de toda la comunicación.

3.8 Pruebas de desarrollo

En este apartado se muestran las pruebas y comprobaciones del funcionamiento de la arquitectura de red desplegada y de la aplicación web desarrollada.

3.8.1 Testeo cliente MQTT-SN Python

3.8.1.1 Búsqueda de puerta de enlace

En un segundo dispositivo Raspberry Pi se ha instalado una librería Python que funciona como cliente MQTT-SN. Con este cliente se muestra la característica de búsqueda de puerta de enlace que define la especificación del protocolo MQTT-SN.

El cliente MQTT-SN no conoce la dirección IP y puerto de la puerta de enlace, por tanto, envía un mensaje “SEARCHGW” en la red local, a la espera de que la puerta de enlace que esté activa le responda y así poder establecer una conexión con ella.

La puerta de enlace contesta con un mensaje “GWINFO” con la información necesaria para establecer la conexión.

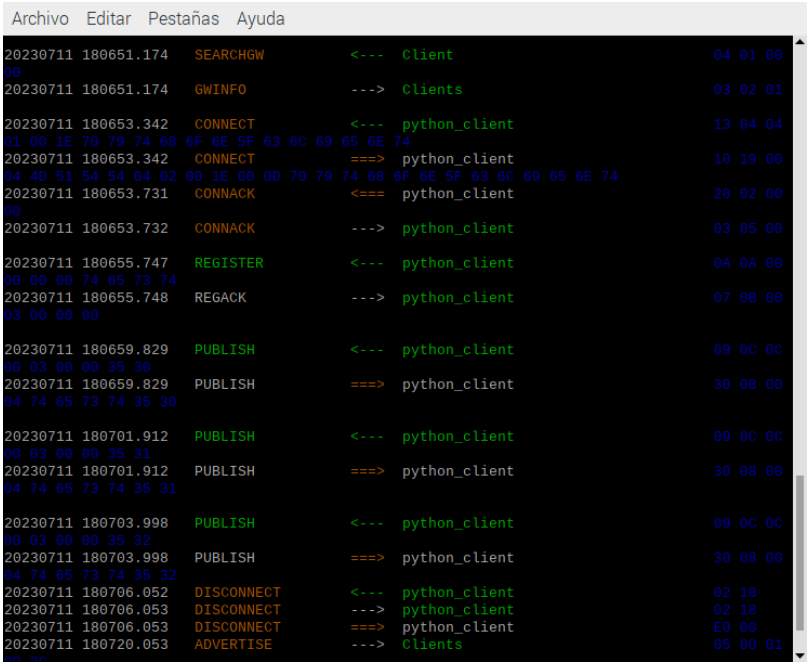

Figura 44. Búsqueda de puerta de enlace

En la variable “gateways” se muestra la información que identifica a la puerta de enlace activa. Con esta información el cliente Python se conecta a esta puerta de enlace y puede comenzar a publicar y suscribirse.

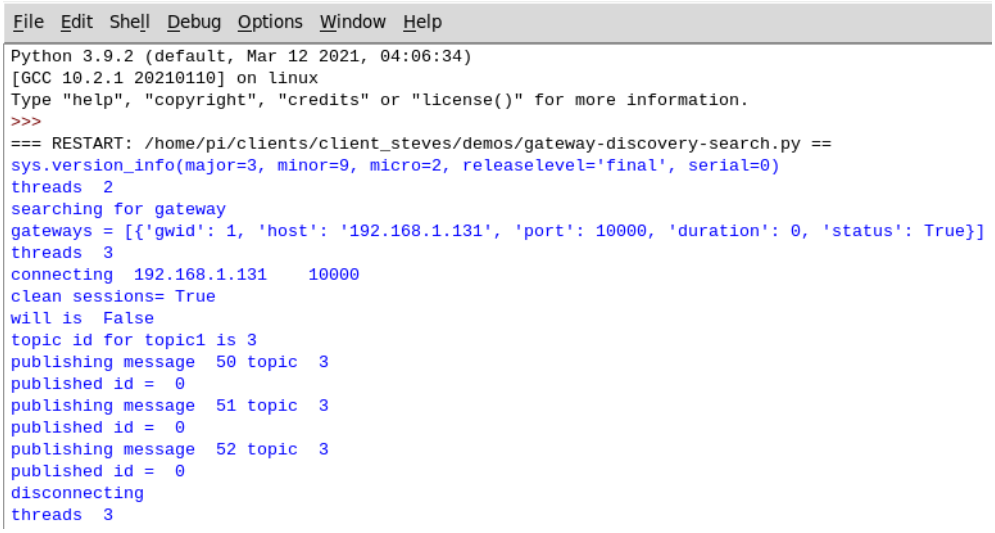

Figura 45. Cliente Python MQTT-SN

3.8.1.2 Registro de topic largo

El protocolo MQTT-SN permite publicar o suscribirse a topics largos, pero dado que el protocolo se identifica por la capacidad de enviar menor carga en los mensajes, asigna un identificador de topic a ese topic largo y usa este identificador en lugar del topic largo.

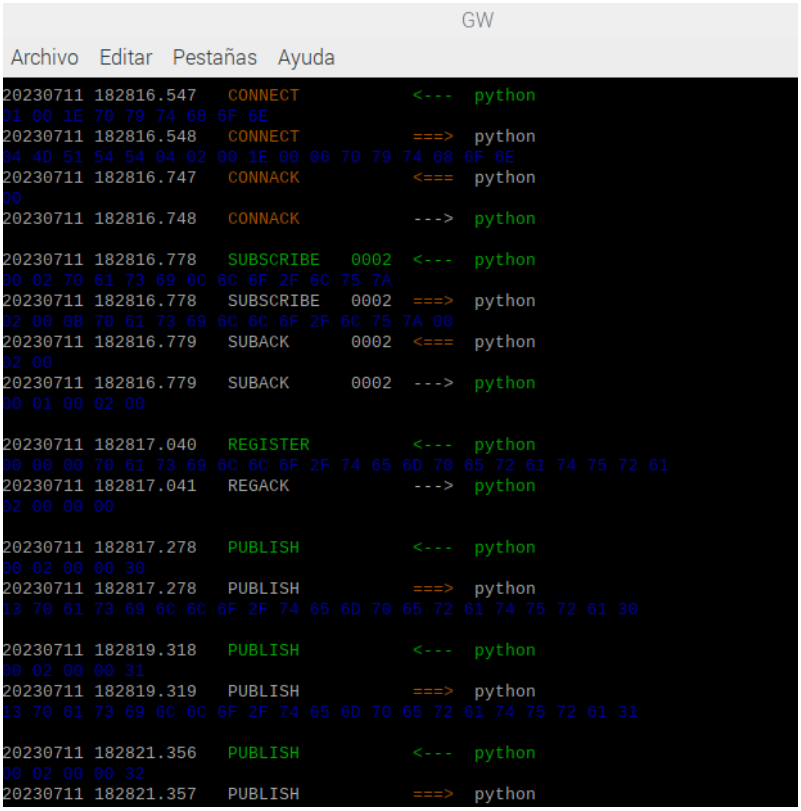

Figura 46. Registro de topic largo

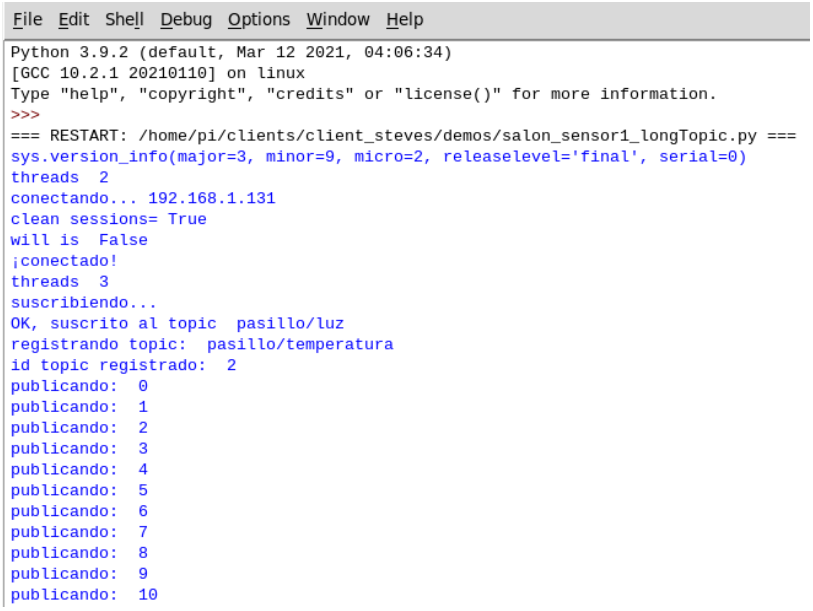

Figura 47. Registro topic largo cliente Python

3.8.2 Testeo con CLI

En el dispositivo Raspberry Pi se hace uso de una herramienta de publicación/suscripción mediante línea de comandos. De esta forma se puede testear de forma rápida y sencilla el funcionamiento de la red.

En la siguiente figura se muestra un ejemplo de publicación y suscripción al topic predefinido:

- 4 → correspondiente a *cocina/control/mensaje*

Y se puede observar el intercambio de mensajes entre clientes MQTT-SN, puerta de enlace y broker mosquitto

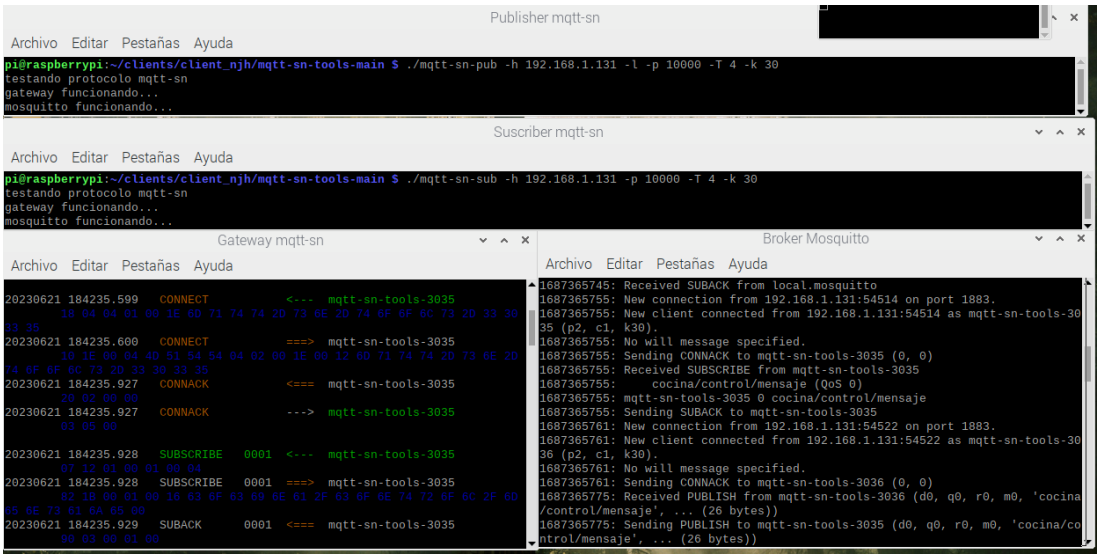

Figura 48: Funcionamiento del protocolo MQTT-SN

3.8.3 Testeo con microcontroladores

En este apartado se muestran los resultados del testeo de la interacción de la aplicación IoT desarrollada con los dispositivos sensores.

Para la alimentación de corriente de los microcontroladores se han usado baterías de polímero de litio y alimentación vía USB.

Una vez encendidos comienzan a publicar sus datos y a suscribirse a los topics predefinidos correspondientes.

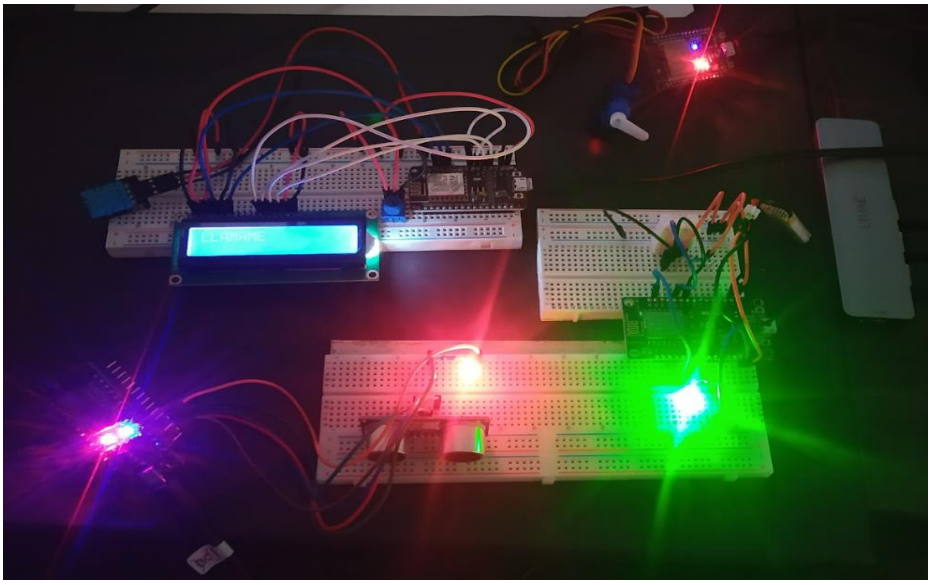

Figura 49. Dispositivos sensores operando

En la siguientes figuras, para mayor claridad, se muestran los resultados del intercambio de mensajes del microcontrolador NodeMCU esp8266 con la aplicación IoT desarrollada

La siguiente figura muestra el proceso de conexión, suscripción y publicación del microcontrolador NodeMCU esp8266.

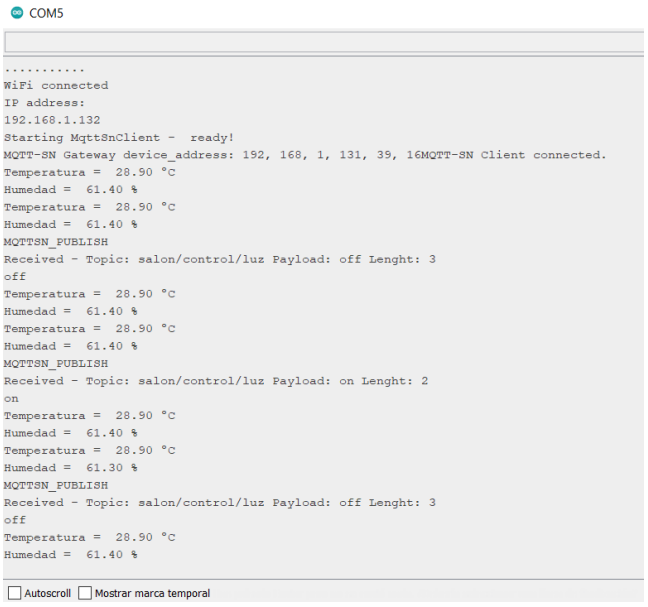

Figura 50. Proceso de conexión, suscripción y publicación

En la siguiente figura, se puede ver como la puerta de enlace PahoGateway está constantemente enviando mensajes de tipo “ADVERTISE”, anunciado que está activa. Luego, llega el mensaje “CONNECT” desde el microcontrolador NodeMCU 8266 llamado SN_22. La puerta de enlace traduce el mensaje MQTT-SN y lo reenvía al broker mosquitto. El broker mosquitto lo recibe y

contesta con un mensaje “CONNACK” hacia la puerta de enlace y la puerta de enlace a su vez con un mensaje “CONNACK” hacia el microcontrolador.

Después de establecerse la conexión comienza el intercambio de mensajes, por un lado, “SUBSCRIBE” y sus respectivos “SUBACK” y luego los mensajes “PUBLISH”.

Figura 51. Intercambio de mensajes del protocolo MQTT-SN

En cuanto al broker mosquitto, como está configurado en modo de puente, los mensajes “PUBLISH” enviados hacia *local.moquitto* hacen referencia a la configuración puente, es decir, hacia el broker remoto EMQX.

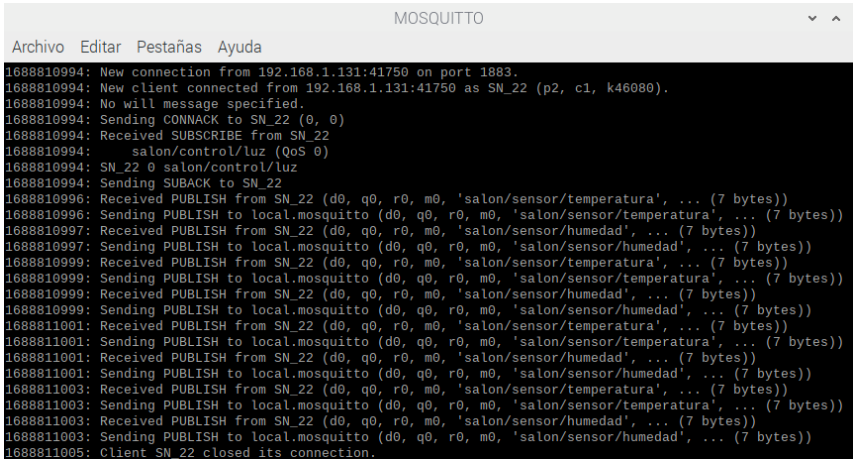

Figura 52. Broker mosquitto operando como puente

Haciendo uso de la herramienta “Log Trace” del dashboard de EMQX, se puede crear una traza y visualizar los mensajes que llegan al broker EMQX.

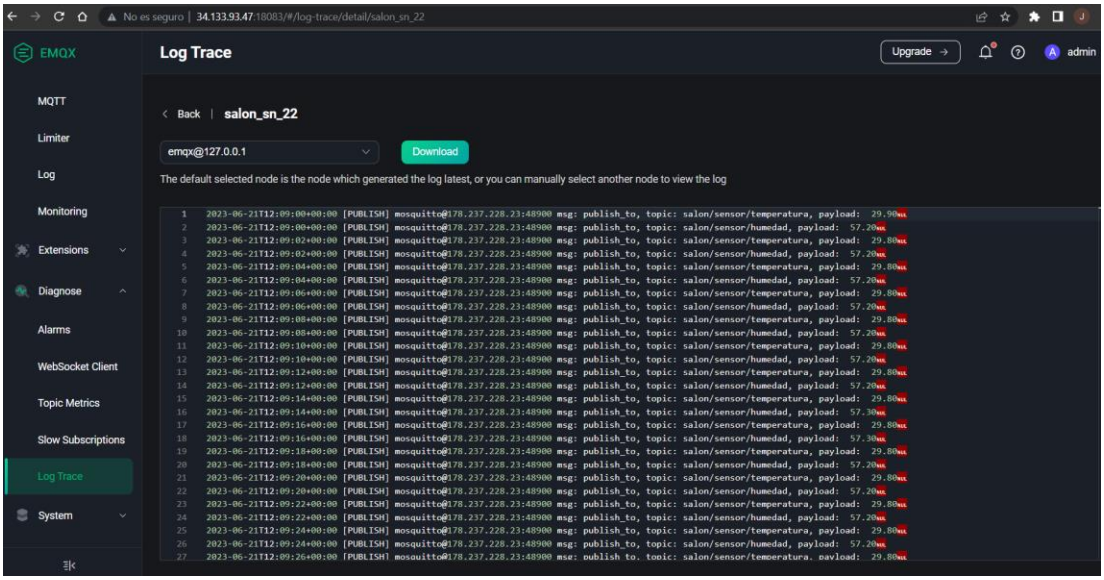

Figura 53. Traza de datos que llegan al broker EMQX

Accediendo a la aplicación IoT desarrollada con bloques de nodos, se puede ver en la parte de “Debug” como llegan los datos de todos los dispositivos sensores.

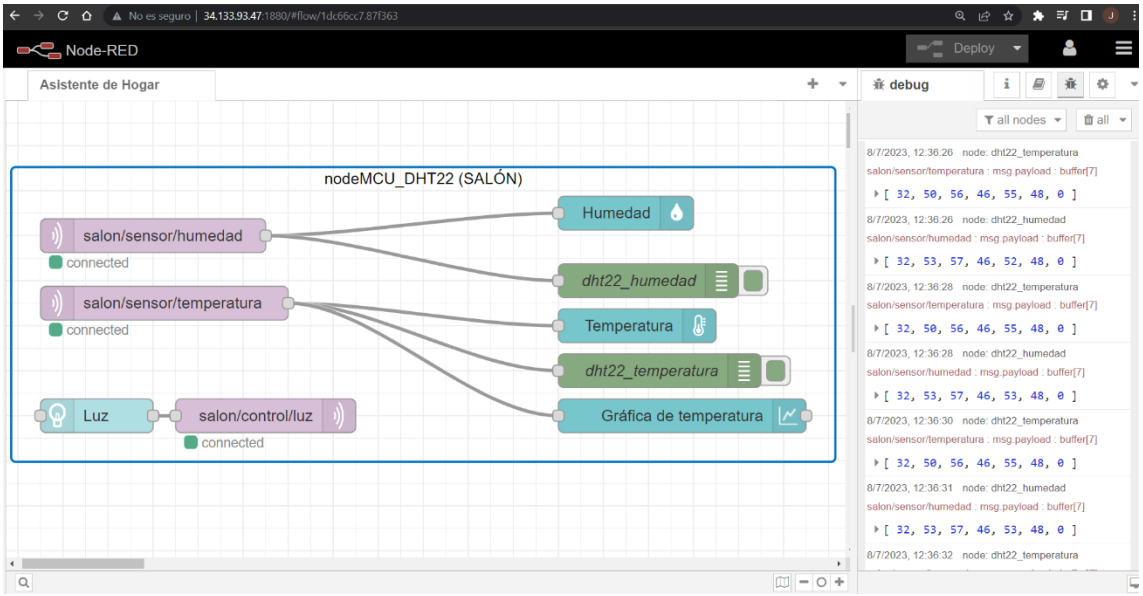

Figura 54. Datos que llegan del microcontrolador NodeMCU

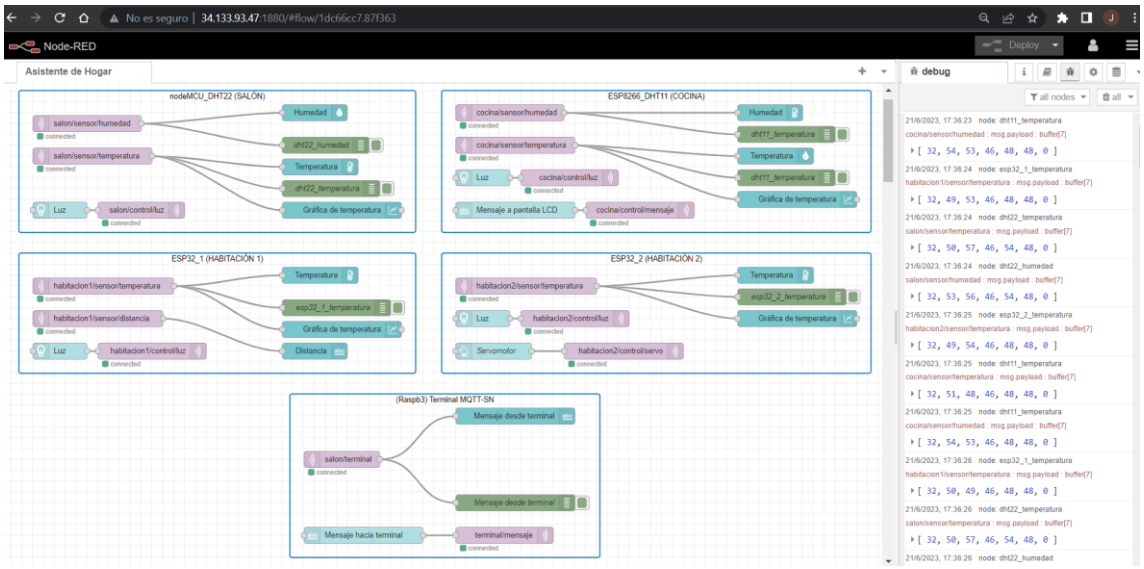

Figura 55. Datos que llegan de todos los dispositivos sensores

Accediendo a la interfaz de usuario de aplicación IoT, se pueden ver los datos de temperatura, humedad, distancia que están llegando e interactuar con los microcontroladores, encendiendo leds, el servomotor o enviando un mensaje a la pantalla LCD.

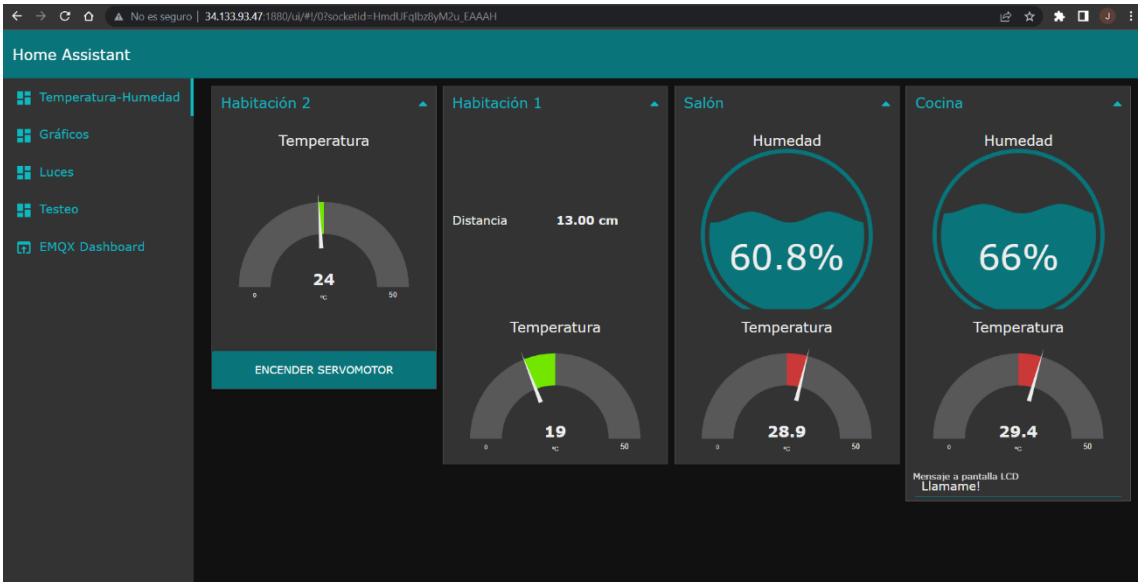

Figura 56. Visualización de los datos en la aplicación IoT

3.8.4 Testeo con clientes MQTT

En apartado se muestra como clientes MQTT-SN y clientes MQTT pueden comunicarse en la misma arquitectura de red desplegada.

3.8.4.1 Configuración MQTT Dash

La aplicación es sencilla de utilizar y de configurar. Los parámetros son los siguientes:

- Nombre para la conexión.
- Dirección del broker al cual conectarse.
- Puerto del broker al cual conectarse.
- Identificador de cliente (por defecto).

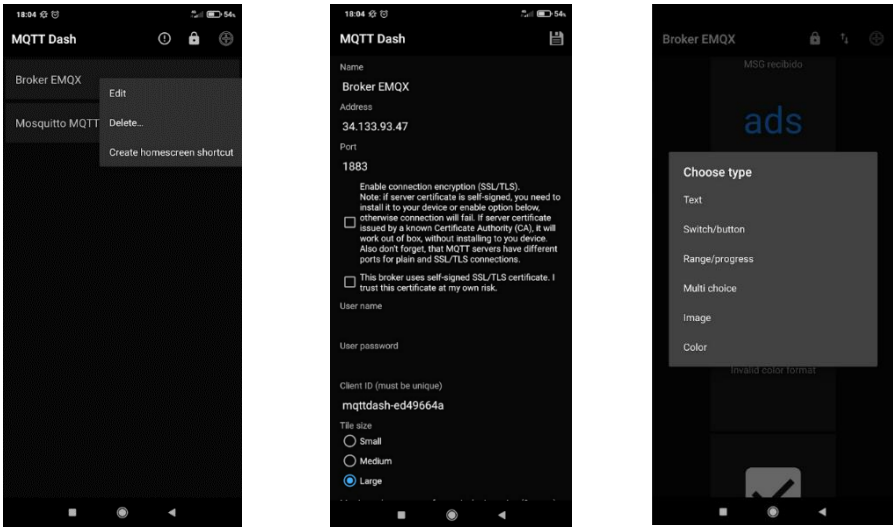

Figura 57. Configuración aplicación MQTT-Dash

Después se eligen los widgets que se quieren utilizar y se configuran.

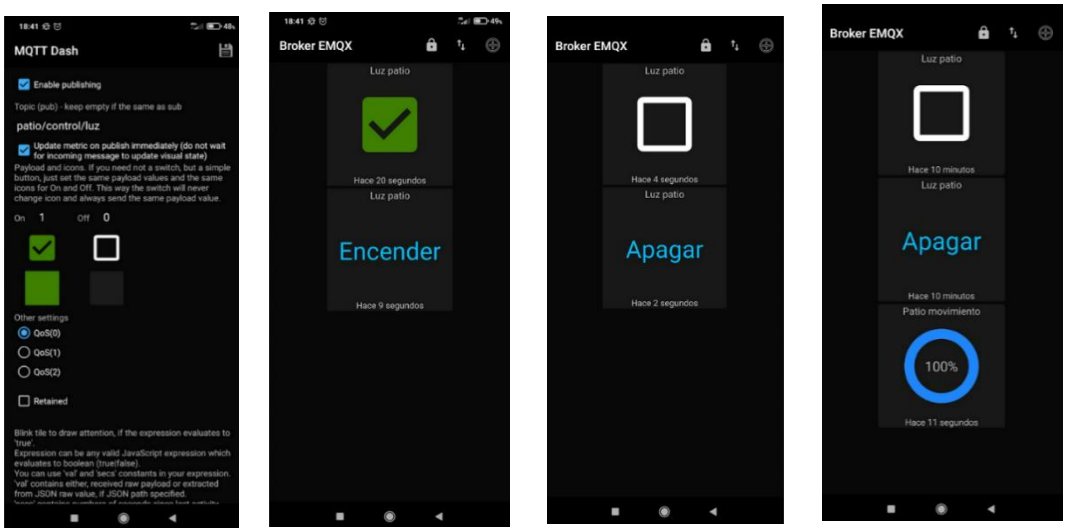

Figura 58. Configuración widgets aplicación MQTT-Dash

Para este testeo, la transferencia de mensajes es desde la aplicación móvil hacia la línea de comando de la Raspberry Pi.

La aplicación móvil publica datos en el tema largo *patio/control/luz* y desde la línea de comandos nos suscribimos a este mismo tema largo y también a su correspondiente tema predefinido (7).

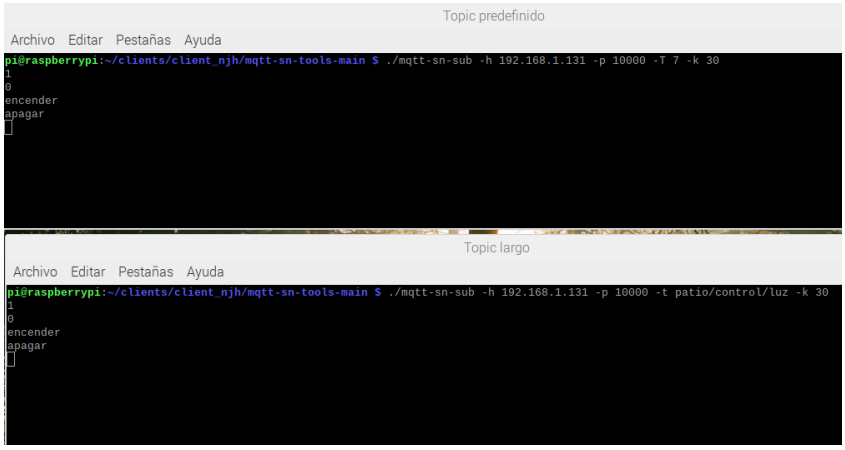

Figura 59. Suscripción cliente MQTT-SN por línea de comandos

En ambos casos, los mensajes llegan al cliente MQTT-SN sin inconvenientes. Estos mensajes siguen el flujo de la red desplegada. De la aplicación MQTT-Dash van al broker EMQX, desde EMQX van al broker local mosquitto, desde mosquitto a la puerta de enlace Paho y desde ella al cliente MQTT-SN.

Capítulo 4. Conclusiones

Este proyecto tenía como propósito desarrollar una aplicación IoT para interactuar con los dispositivos sensores en una red local basada en el protocolo de comunicación MQTT-SN.

Los desafíos más críticos durante el desarrollo de este proyecto han sido la escasez de librerías MQTT-SN estables, tanto para los clientes MQTT-SN como para la puerta de enlace y ningún soporte o arreglo de bugs para las librerías existentes en la actualidad por lo que problemas con la versione de las librerías y el arreglo de bugs en ellas han sido un problema recurrente.

Otros desafíos han sido el aprendizaje de las herramientas de programación utilizadas, el IDE de Arduino y Node-Red, así como el aprendizaje en la utilización de comandos Linux.

Una vez superados los desafíos, se dispone de una red MQTT-SN estable que permite tener un proyecto final con las siguientes características:

- Una red MQTT-SN de coste mínimo con cargas de mensaje mínimas debido al uso de los topics predefinidos, fácilmente escalable en el número de dispositivos sensores.
- Una aplicación web IoT accesible desde cualquier dispositivo con conexión a Internet.
- Una aplicación web IoT fácilmente escalable simplemente ampliado el número de nodos y reutilizando el mismo código de programación tanto para los microcontroladores como para los bloques de nodos.
- Un panel de control que gestiona e informa de toda la comunicación MQTT y MQTT-SN.

El hecho de haber desarrollado una aplicación IoT completa ha significado un mayor aprendizaje en cuanto a las fases de las que consta un proyecto.

Por otra parte, ha hecho que aprendiera y afianzará conocimientos en tecnologías como las placas de desarrollo, el dispositivo Raspberry Pi, el protocolo MQTT y MQTT-SN y los distintos servicios en la nube de Google.

4.1 Posibles mejoras de futuro

El presente proyecto deja la posibilidad para incorporar nuevas funcionalidades y tecnologías que ayuden a complementar la propuesta inicial del proyecto.

Una de las mejoras a corto plazo sería la incorporación de más dispositivos sensores que doten de más y mejor funcionalidad a la aplicación.

Como mejoras a más largo plazo, la implementación de más características del protocolo MQTT-SN definidas en su especificación, como puede ser la utilización del nivel 3 de QoS y la posibilidad de que los clientes entren en un estado de stand-by (cliente dormido). Para ello, tienes que seguir desarrollándote como programador de sistemas en chip.

Otra mejora podría ser añadir un sistema de almacenamiento en bases de datos que permita tener un histórico de los datos enviados por los dispositivos sensores.

Finalmente, para aumentar la robustez del sistema, incorporar una capa de seguridad en el envío de los datos por MQTT-SN.

Capítulo 5. Bibliografía

[1] Similitudes y diferencias entre Redes de Sensores Inalámbricas e Internet de las Cosas. Dialnet. https://dialnet.unirioja.es/servlet/articulo?codigo=6720876 (accedida el 02 de marzo de 2023).

[2] MQTT Specifications. MQTT. https://mqtt.org/mqtt-specification/ (accedida el 06 de marzo de 2023).

[3] MQTT Version 3.1.1. Oasis. https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (accedida el 06 de marzo de 2023).

[4] MQTT Version 5.0. Oasis. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (accedida el 06 de marzo de 2023).

[5] Introduction to MQTT-SN (MQTT for Sensor Networks). Web de Steve's Internet Guide. http://www.steves-internet-guide.com/mqtt-sn/ (accedida el 06 de marzo de 2023).

[6] Easier than MQTT? MQTT / UDP. Sudonull. https://sudonull.com/post/8428-Easier-than-MQTT-MQTT-UDP (accedida el 07 de marzo de 2023).

[7] Client-Server Communication over MQTT-SN Protocol : IOT Part 41. Engineers Garage. https://www.engineersgarage.com/client-server-communication-over-mqtt-sn-protocol-iot-part-41/ (accedida el 07 de marzo de 2023).

[8] Using MQTT-SN over BLE with the BBC micro:bit. IoT - AI - Open Source & random tech news. https://blog.benjamin-cabe.com/2017/01/16/using-mqtt-sn-over-ble-with-the-bbc-microbit (accedida el 07 de marzo de 2023).

[9] Configurar la Raspberry Pi. Tutorial de YouTube. https://youtu.be/X0CHQqrVl2o (accedida el 07 de marzo de 2023).

[10] Que es BOARD y BCM en Raspberry Pi. MNP. https://www.mnp.cl/es/post/que-es-board-bcm-raspberry-pi (accedida el 07 de marzo de 2023).

[11] paho.mqtt-sn.embedded-c. Github. https://github.com/eclipse/paho.mqtt-sn.embedded-c (accedida el 10 de julio 2023).

[12] Using The Python MQTT-SN Client. Web de Steve's Internet Guide. http://www.steves-internet-guide.com/python-mqttsn-client/ (accedida el 10 de julio 2023).

[13] arduino-mqtt-sn-client. Github. https://github.com/S3ler/arduino-mqtt-sn-client (accedida el 10 de julio 2023).

[14] Error: Multiple libraries were found for “WiFi.h”. Random Nerd Tutorials Lab. https://rntlab.com/question/error-multiple-libraries-were-found-for-wifi-h/ (accedida el 04 de junio 2023).

[15] Web de Google Cloud Platform. https://cloud.google.com/?hl=es (accedida el 10 de julio 2023).

[16] How to create a virtual machine in Google Cloud Platform. TechRepublic. https://www.techrepublic.com/article/how-to-create-a-virtual-machine-in-google-cloud-platform/ (accedida el 20 abril de 2023).

[17] Introduction. Web de EMQX. https://www.emqx.io/docs/en/v5.0/

[18] How to Deploy EMQX Enterprise on Google Cloud. EMQ. https://www.emqx.com/en/blog/how-to-deploy-emqx-enterprise-on-google-cloud (accedida el 20 abril de 2023).

[19] EMQX-SN. Github. https://github.com/emqx/emqx-sn (accedida el 20 abril de 2023).

[20] How to deploy a node-red environment on a gcp instance. b105 lab. https://elb105.com/how-to-deploy-a-node-red-environment-on-a-gcp-instance/ (accedida el 20 abril de 2023).

[21] ESP8266 con DHT22 mediante WiFi, Broker MQTT en Raspberry Pi y visualización en Node-RED. Tutorial de YouTube. https://youtu.be/8XN8nIKINBA (accedida el 04 de mayo de 2023).

[22] Servidor Web mediante ESP8266 y DHT22 en red WiFi, código en Micropython, base para futuro MQTT, IoT. Tutorial de YouTube. https://youtu.be/PVOwfp_EjR8 (accedida el 04 de mayo de 2023).

[23] MOOC Tecnología DIY. Comunicación por mqtt entre un esp8266 y la Raspberry Pi. Tutorial de YouTube. https://youtu.be/ibwxqt4TGAU (accedida el 04 de mayo de 2023).

[24] ESP32 with HC-SR04 Ultrasonic Sensor with Arduino IDE. Random Nerd Tutorials. https://randomnerdtutorials.com/esp32-hc-sr04-ultrasonic-arduino/ (accedida el 04 de mayo de 2023).

[25] Steve’s Node-Red Guide. Web de Steve’s Node-Red Guide. https://stevesnoderedguide.com/ (accedida el 15 de junio de 2023).

[26] How to style a custom Node Red Dashboard. Tutorial de YouTube. https://youtu.be/l80VSYVQts0 (accedida el 15 de junio de 2023).

Capítulo 6. Anexos

6.1 Instalación Puerta de Enlace

Para construir la puerta de enlace se deben realizar los siguientes pasos:

1. Descargar librería “paho.mqtt-sn.embedded-c”.

Y ejecutar los siguientes comandos:

2. cd paho.mqtt-sn.embedded-c/MQTTSNGateway.
3. ./build.sh [udp].

Para arrancar la puerta de enlace ejecutar comando:

4. cd bin.
5. ./MQTT-SNGateway.

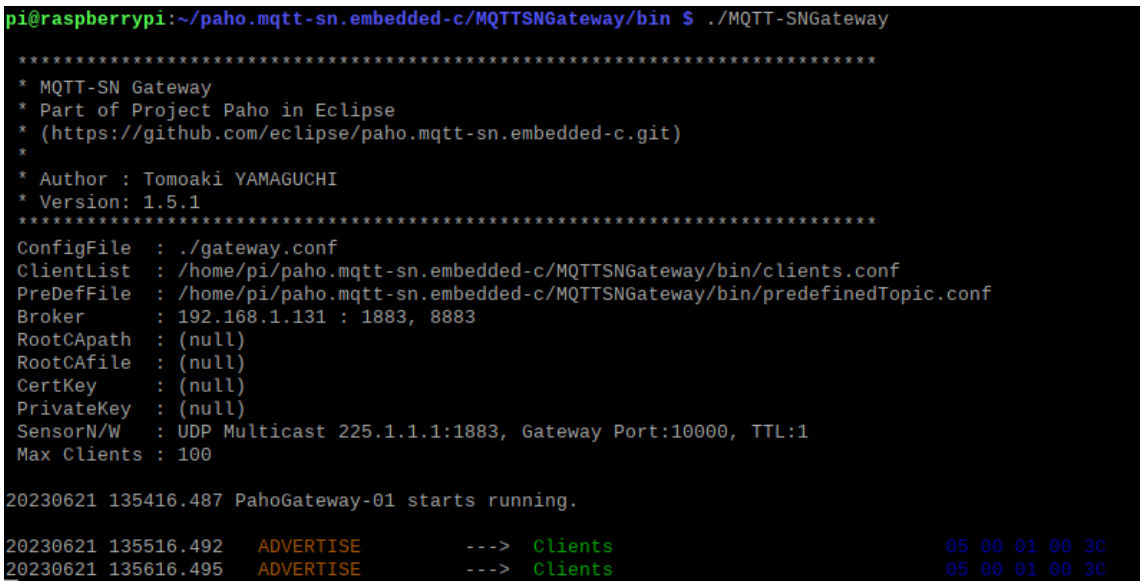

Figura 60. Arranque de puerta de enlace

6.1.1 Archivo de configuración

A continuación, debemos modificar el archivo de configuración de la puerta de enlace según nuestras necesidades. Para ello, en el mismo directorio donde hemos instalado la puerta de enlace ejecutamos el comando:

- sudo nano gateway.config

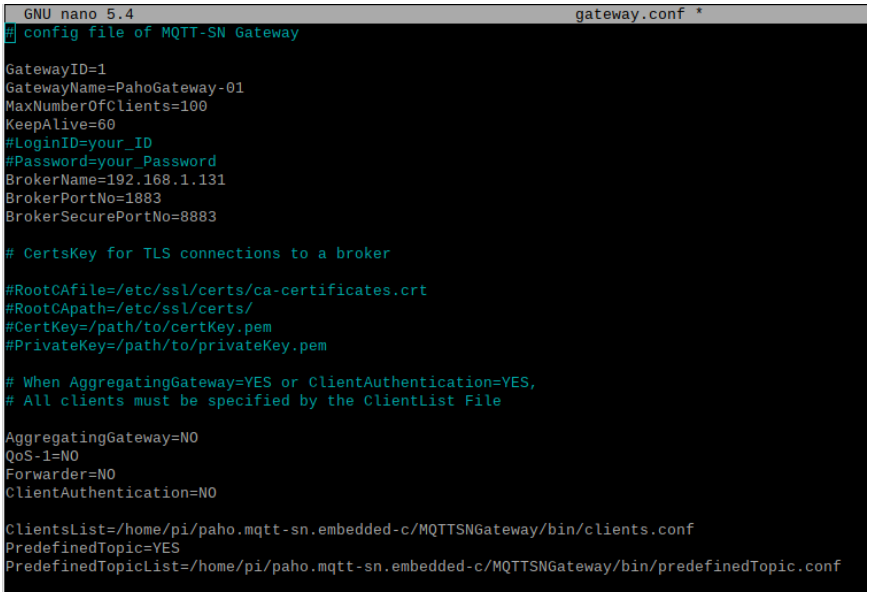

Figura 61. Archivo de configuración de puerta de enlace

Los datos que modificamos son los siguientes:

BrokerName: nombre de dominio o la dirección IP del broker destino.

BrokerPortNo: número de puerto del broker destino.

AggregatingGateway: tipo de puerta de enlace (transparente o de agregación).

PredefinedTopic: habilitar el uso de topics predefinidos.

PredefinedTopicList: ruta del archivo donde se encuentran los topics predefinidos.

6.1.2 Archivo de configuración de topics predefinidos

Para modificar el archivo de los topics predefinidos ejecutamos el comando:

- sudo nano predefinedTopic.config

Como se observa en la siguiente figura podemos construir los id de topic predefinidos en función de los topics largos que queremos utilizar. De esta forma, al publicar o suscribirse utilizaremos la característica de topics predefinidos y reducimos el payload de los mensajes intercambiados.

```
  GNU nano 5.4                                          predefinedTopic.conf
#  pre-defined-topics are defined by this file.
#  A format of this file is in CSV as follows:
#
#     ClientID, TopicName, TopicID
#
#     Topics is common to all clients, ClientID should be *.

#
#  pre-defined-topics for Clients
#
#NodeMCU DHT22
*,salon/control/luz, 1
*,salon/sensor/temperatura, 2
*,salon/sensor/humedad, 3
#esp8266 DHT11 + LCD
*,cocina/control/mensaje, 4
*,cocina/sensor/temperatura, 5
*,cocina/sensor/humedad, 6
#raspberry 4+
*,patio/control/luz, 7
*,patio/sensor/movimiento, 8
*,patio/sensor/camara, 9
#esp32_1 datos en código
*,habitacion1/control/luz, 10
*,habitacion1/sensor/temperatura, 11
*,habitacion1/sensor/distancia, 12
#esp32_2 datos en código
*,habitacion2/control/luz, 13
*,habitacion2/sensor/temperatura, 14

#
```

Figura 62. Topics predefinidos en archivo configuración

Una vez realizada la configuración de ambos archivos, arrancamos la puerta de enlace con el comando:

- ./MQTT-SNGateway

En la siguiente figura se observa cómo es la comunicación entre los clientes MQTT-SN, la puerta de enlace y el broker mosquitto.

```
20230621 135416.487 PahoGateway-01 starts running.

20230621 135516.492   ADVERTISE         --->  Clients                 05 00 01 00 3C
20230621 135616.495   ADVERTISE         --->  Clients                 05 00 01 00 3C
20230621 135716.499   ADVERTISE         --->  Clients                 05 00 01 00 3C
20230621 135816.502   ADVERTISE         --->  Clients                 05 00 01 00 3C
20230621 135916.506   ADVERTISE         --->  Clients                 05 00 01 00 3C

20230621 135926.779   CONNECT           <---  SN_22                   0C 04 04 01 B4 00 53 4E 5F 32 32 00
20230621 135926.780   CONNECT           ===>  SN_22                   10 11 00 04 4D 51 54 54 04 02 B4 00 00 00 05 53 4E 5F 32 32
20230621 135927.059   CONNACK           <===  SN_22                   20 02 00 00
20230621 135927.059   CONNACK           --->  SN_22                   03 05 00

20230621 135927.066   SUBSCRIBE   0200  <---  SN_22                   17 12 00 02 00 73 61 6C 6F 6E 2F 63 6F 6E 74 72 6F 6C 2F 6C 75 7A 00
20230621 135927.066   SUBSCRIBE   0200  ===>  SN_22                   82 16 02 00 00 11 73 61 6C 6F 6E 2F 63 6F 6E 74 72 6F 6C 2F 6C 75 7A 00
20230621 135927.066   SUBACK      0200  <===  SN_22                   90 03 02 00 00
20230621 135927.066   SUBACK      0200  --->  SN_22                   08 13 00 00 01 02 00 00

20230621 135929.077   PUBLISH           <---  SN_22                   0E 0C 01 00 02 00 00 20 33 30 2E 30 30 00
20230621 135929.078   PUBLISH           ===>  SN_22                   30 21 00 10 73 61 6C 6F 6E 2F 73 65 6E 73 6F 72 2F 74 65 6D 70 65 72 61 74
75 72 61 20 33 30 2E 30 30 00

20230621 135929.078   PUBLISH           <---  SN_22                   0E 0C 01 00 03 00 00 20 35 37 2E 30 30 00
20230621 135929.078   PUBLISH           ===>  SN_22                   30 1D 00 14 73 61 6C 6F 6E 2F 73 65 6E 73 6F 72 2F 68 75 6D 65 64 61 64 20
35 37 2E 30 30 00

20230621 135931.085   PUBLISH           <---  SN_22                   0E 0C 01 00 02 00 00 20 32 39 2E 39 30 00

20230621 135931.085   PUBLISH           <---  SN_22                   0E 0C 01 00 03 00 00 20 35 37 2E 30 30 00
20230621 135931.085   PUBLISH           ===>  SN_22                   30 21 00 10 73 61 6C 6F 6E 2F 73 65 6E 73 6F 72 2F 74 65 6D 70 65 72 61 74
75 72 61 20 32 39 2E 39 30 00
20230621 135931.086   PUBLISH           ===>  SN_22                   30 1D 00 14 73 61 6C 6F 6E 2F 73 65 6E 73 6F 72 2F 68 75 6D 65 64 61 64 20
```

Figura 63. Puerta de enlace arrancada

6.2 Instalación broker Mosquitto

Para su instalación en el dispositivo Raspberry Pi se siguen los siguientes pasos:

1. Importar la clave de firma del paquete del repositorio:

- wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
- sudo apt-key add mosquitto-repo.gpg.key

2. Poner el repositorio a disposición de apt:
   - cd /etc/apt/sources.list.d/
3. Dependiendo de la versión de Debian que se está utilizando ejecutar:
   - sudo wget http://repo.mosquitto.org/debian/mosquitto-jessie.list
   - sudo wget http://repo.mosquitto.org/debian/mosquitto-stretch.list
   - sudo wget http://repo.mosquitto.org/debian/mosquitto-buster.list
4. Actualizar la información de apt:
   - apt-get update
5. Finalmente instalar:
   - apt-get install mosquitto

6.2.1 Modificación archivo de configuración

Es necesario modificar el archivo de configuración que se encuentra en la ruta /etc/mosquitto/. Para ello se ejecuta el comando:

- sudo nano mosquitto.config

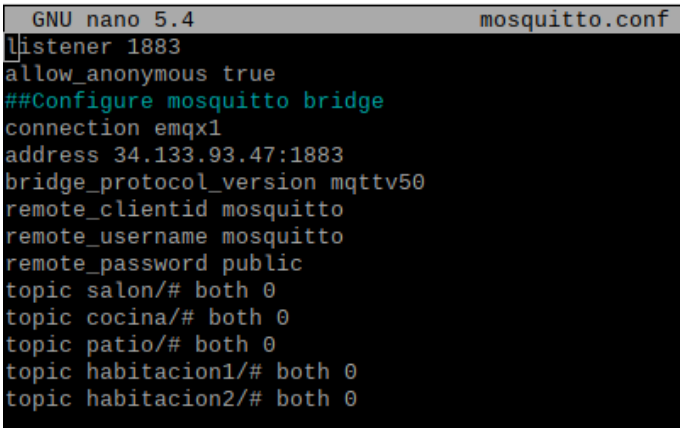

Figura 64. Archivo de configuración broker mosquitto

Las modificaciones son las siguientes:

- listener: escucha la conexión de red entrante en el puerto especificado.
- allow_anonymous: permite o no que los clientes que se conectan sin proporcionar un nombre de usuario pueden conectarse. Si se establece a false, entonces se debe crear otro medio de conexión para controlar el acceso de clientes autenticados.

El tercer parámetro configurado corresponde al modo puente en el que puede trabajar el broker mosquitto, de esta forma el broker mosquitto reenviará los mensajes de la red local hacia el broker remoto y viceversa. Este parámetro se explica con más detalle en el apartado de la instalación del broker EMQX.

Por último, para que la configuración surta efecto, es necesario reiniciar el broker mosquitto con el comando:

- sudo service mosquitto restart.

6.3 Instalación broker EMQX

Para poder instalar el broker EMQX es necesario acceder al terminal ssh de la máquina virtual. Para ello se hace clic en la opción “ssh"

Una vez dentro del terminal ssh se ejecuta el comando:

- sudo su (cambia la cuenta de usuario a la cuenta de administrador del sistema)
- cd .. (x2 accedemos al directorio raíz)

En esta ruta se descarga EMQX con el comando

- wget <ruta_de_descarga>
- ls (para comprobar que el archivo de descargó).

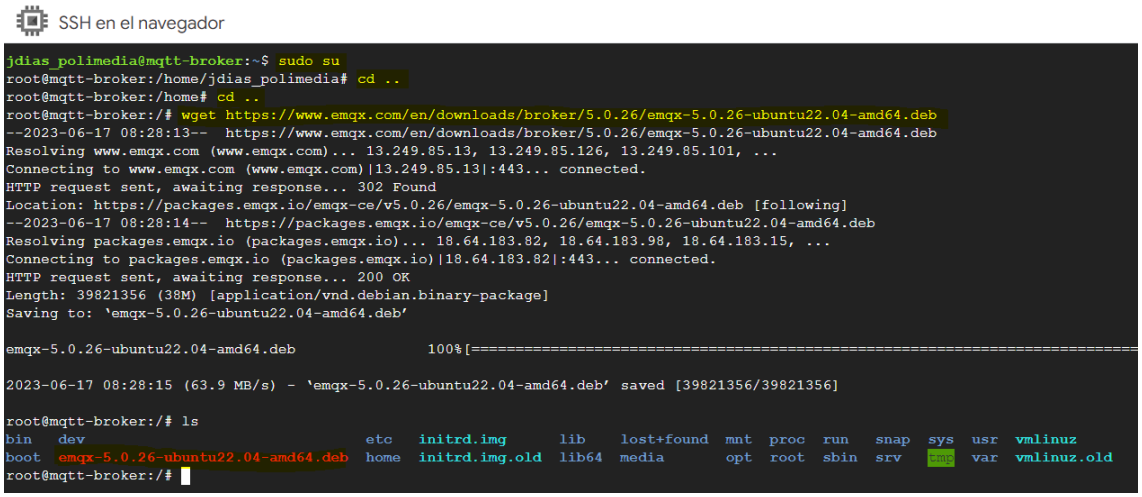

Figura 65. Terminal ssh de la máquina virtual

Se instala EMQX ejecutando el comando:

- sudo apt install ./emqx-5.0.26-ubuntu22.04-amd64.deb.

Al finalizar la instalación se tiene un directorio con el nombre “emqx” donde se encuentra todo el proyecto EMQX, los ficheros de configuración y los módulos.

Para arrancar el broker EMQX se ejecuta el comando: sudo systemctl start emqx.

6.3.1 Comunicación broker Mosquitto hacia broker EMQX

Una vez el broker EMQX está instalado y configurado, se dispone de los datos necesarios para establecer la conexión puente entre el broker mosquitto y el broker EMQX.

Los datos necesarios son los siguientes:

	Dirección	**Puerto de Escucha**
emqx1	34.133.93.47	1883
mosquitto	127.0.0.1	1883

Figura 66. Parámetros de configuración

Los pasos a seguir para configurar el modo puente en el broker mosquitto son los siguientes:

1) Añadir un nombre a la conexión.
2) Añadir dirección y puerto del nodo remoto punteado.
3) Añadir la versión del protocolo MQTT a utilizar (por defecto la 3.1.1 pero EMQX soporta MQTT 5.0).
4) Añadir el ID del cliente para el nodo remoto.
5) Añadir el nombre de usuario para el nodo remoto.
6) Añadir la contraseña para el nodo remoto.
7) Especificar los topics MQTT que se van a puntear.

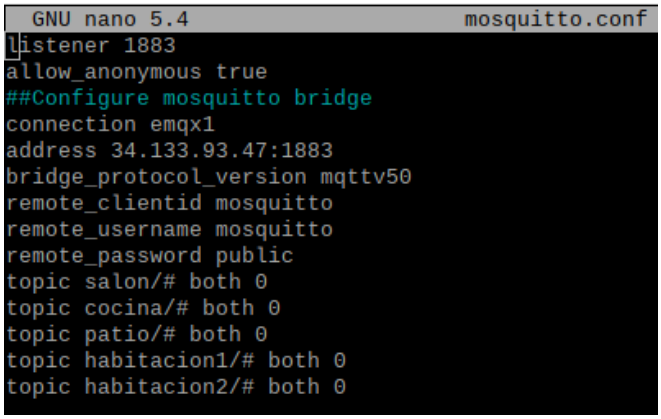

Figura 67. Configuración puente broker mosquitto

Respecto al broker EMQX, no es necesario configurar ningún parámetro para que esta comunicación puente surta efecto.

En la figura se observa como al arrancar mosquitto mediante el archivo de configuración *mosquitto.config* la conexión puente se establece sin problemas

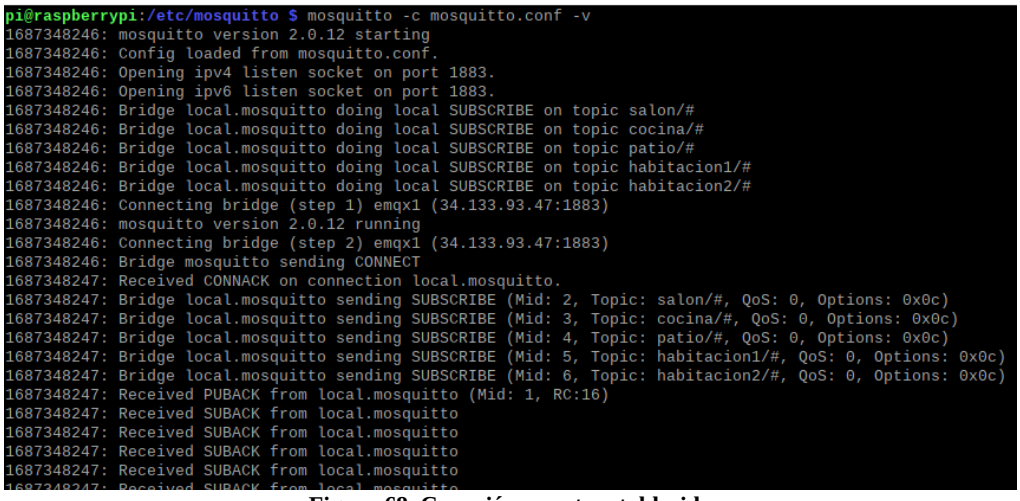

Figura 68. Conexión puente establecida

6.4 Creación de máquina virtual en Google Cloud Platform

Desde el menú de navegación de Google Cloud, se accede a la opción Compute Engine, se selecciona la opción Instancias de VM y se elige crear una nueva instancia.

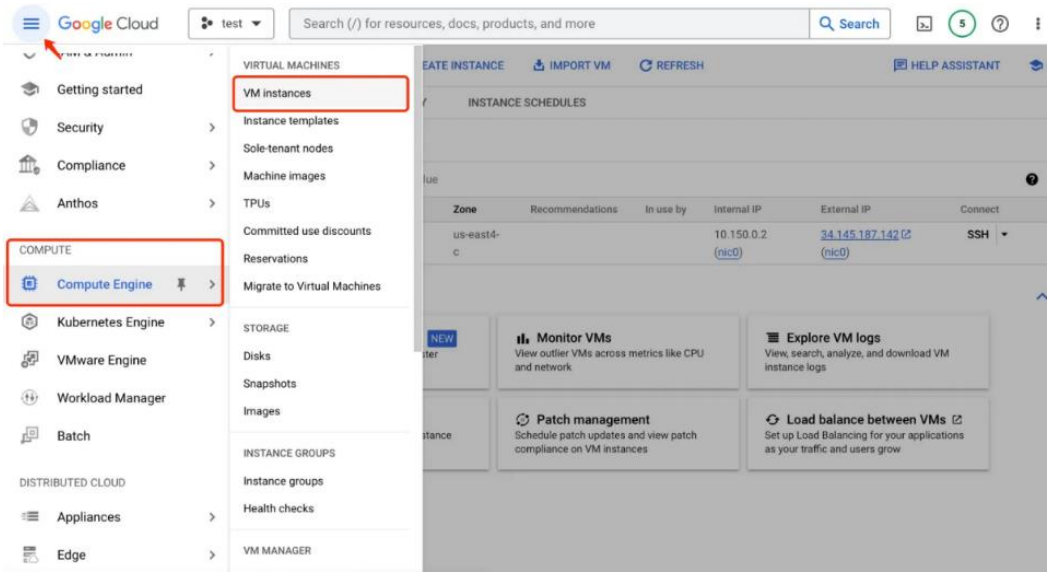

Figura 69. Creación máquina virtual en GCP

Ahora se definen las características de la instancia:

- Nombre de la instancia: mqtt-broker.
- Región en donde crear la instancia: us-central1-c
- Configuración de la máquina: dependiendo del tipo de máquina variará su precio. Por ejemplo, fi-micro [1 vCPU, 614 MB de memoria] con un precio mensual de 5,39 $.

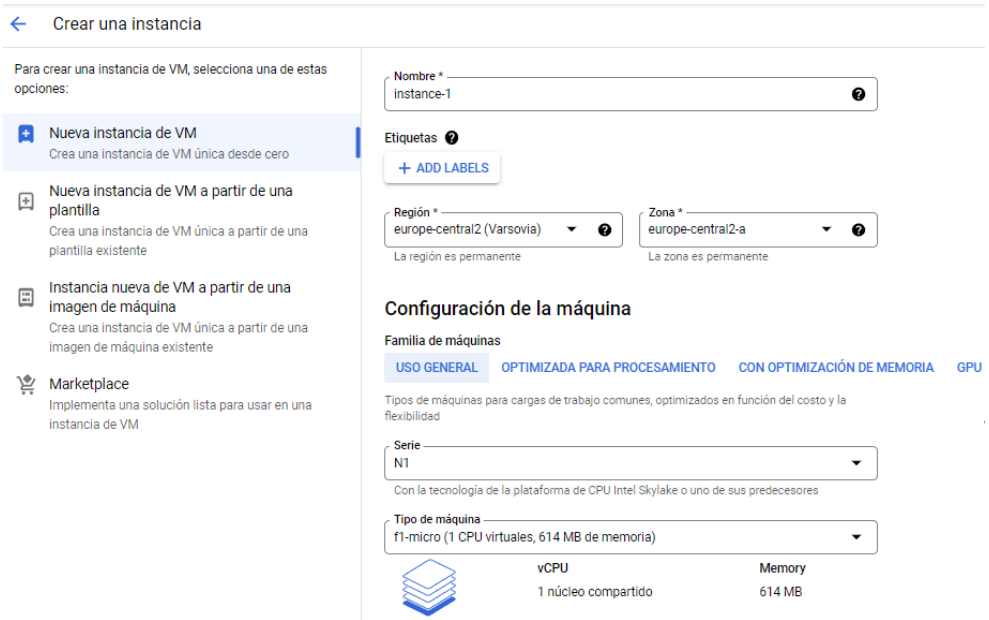

Figura 70. Parámetros de la instancia de máquina virtual

En las opciones de disco de arranque:

- Disco de arranque: Ubuntu 18.04 LTS por familiaridad.
- Tipo de disco de arranque: disco SSD persistente de 10GB.

En las opciones de identidad y acceso a la API:

- Cuentas de servicio: Compute Engine default service account.
- Permisos de acceso: permitir el acceso total a todas las API de Cloud.

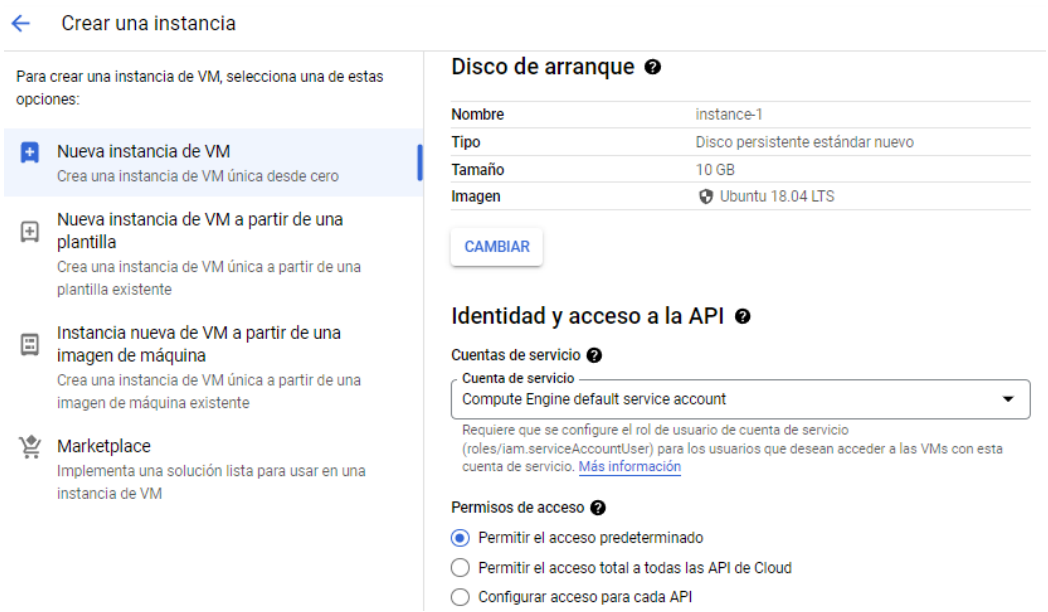

Figura 71. Más parámetros de instancia

En las opciones de Firewall:

- Permitir tráfico HTTP: habilitar
- Permitir tráfico HTTPS: habilitar

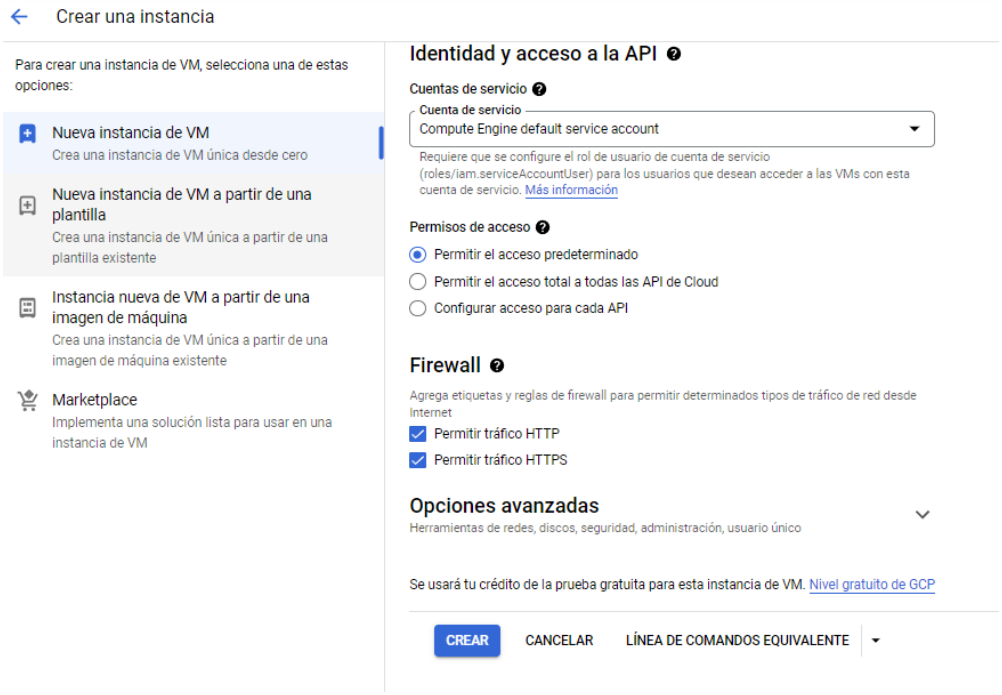

Figura 72. Parámetros de firewall

Por último, se pulsa “Crear” y aparecerá un mensaje indicando que se usará el crédito disponible para crear la instancia.

6.4.1 Definición dirección IP estática

Es necesario definir una dirección IP estática externa para la máquina virtual creada.

Para ello, desde el menú de navegación de Google Cloud, buscamos el apartado “Herramientas de redes” y pulsamos en “Red de VPC”. En el apartado “Direcciones IP externas” localizamos la máquina virtual creada y a su derecha vemos la opción “Reservar”. Al pulsar nos pide un nombre identificador, lo ponemos y aceptamos.

De esta forma la dirección IP externa pasará de tipo “efímera” a “estática”.

Figura 73. IP estática para máquina virtual

6.4.2 Configuración firewall

Google Cloud Platform dispone de un firewall externo que es independiente de la máquina virtual que se ha creado. Este firewall se encarga de bloquear todas las conexiones externas que se realicen. Por lo tanto, es necesario configurarlo abriendo puertos que permitan la comunicación.

Para ello, es necesario crear nuevas reglas en el firewall de la plataforma.

Desde el menú de navegación de Google Cloud se accede a "Red de VPC", en el apartado "Firewall", se selecciona "Crear regla de firewall".

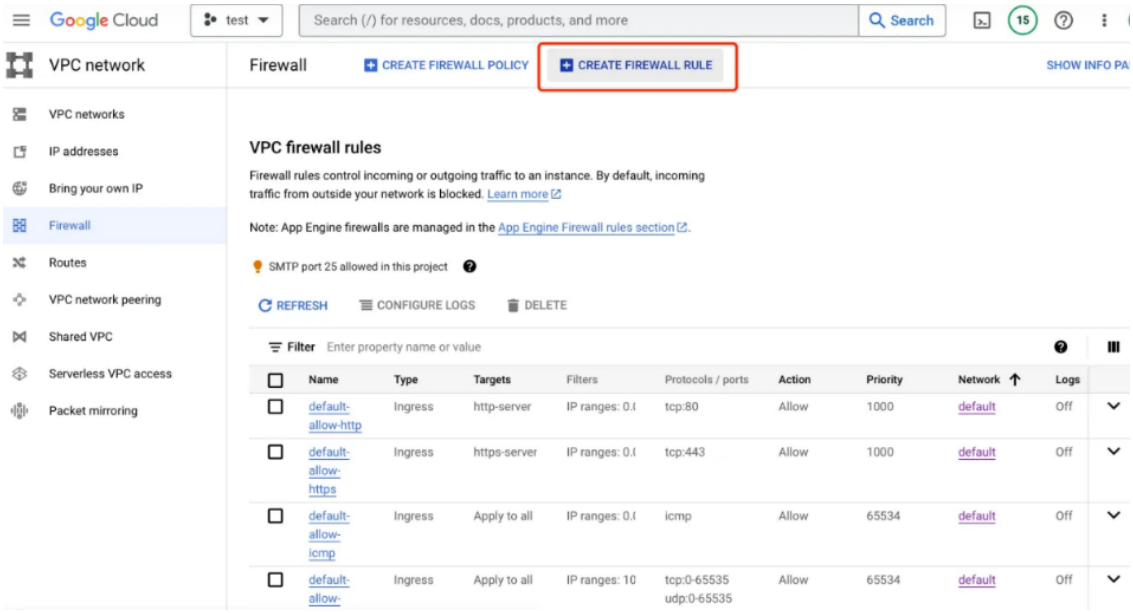

Figura 74. Crear nueva regla de firewall

Se debe dar un nombre a la nueva regla y rellenar sus campos:

- Registros: desactivado.
- Red: default.
- Prioridad con los valores por defecto: 1000.
- Dirección del tráfico: entrada.
- Acción en caso de coincidencia: permitir la acción a realizar.
- Destinos: todas las instancias de la red (en caso de tener otro VPC va a funcionar con las mismas reglas si está en la misma red).
- Filtro de origen: rangos de IPv4.
- Intervalos de IP de origen: 0.0.0.0/0 (se permite que se conecten desde cualquier dirección IP a este puerto en particular).
- Segundo filtro de origen: opción por defecto.

- Protocolos y puertos: habilitar la opción TCP y definir los puertos desde los que se va a permitir conexiones TCP. Se habilita el puerto "1883" para conexiones TCP, el puerto "18083" para acceder al Dashboard de EMQ X y el puerto "8083" para el WebSocket,
- Protocolos y puertos: habilitar la opción UDP y definir los puertos desde los que se va a permitir conexiones UDP. Se habilita el puerto 1884 para conexiones UDP.

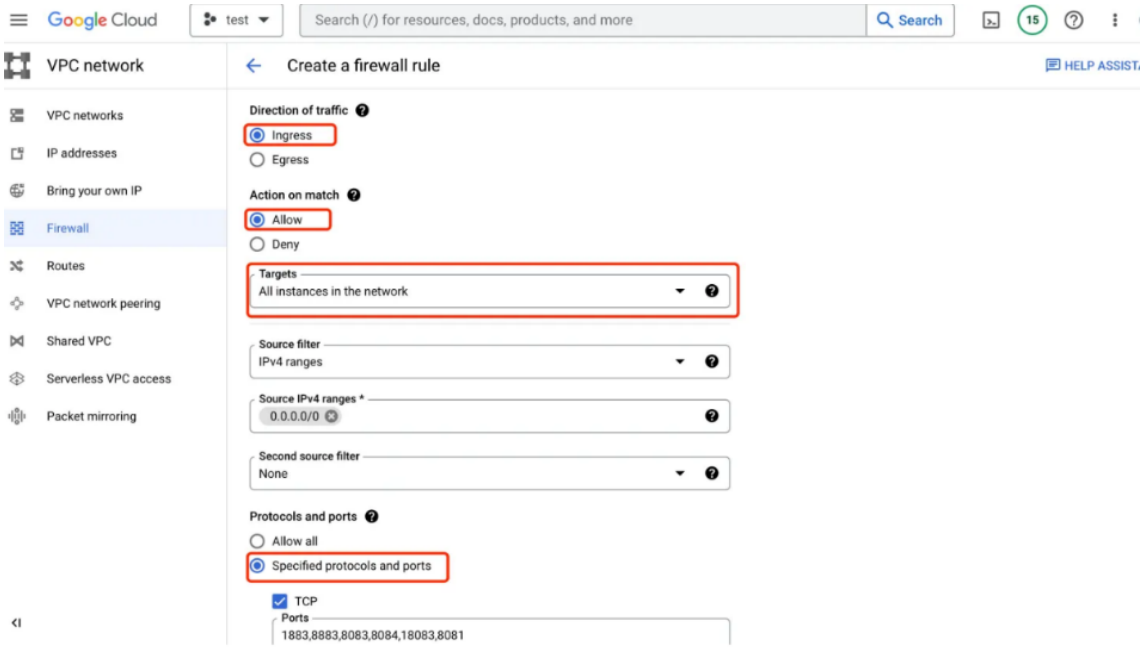

Figura 75. Parámetros nueva regla firewall

Por último, se pulsa en "Crear" y se observa que la regla se ha creado.

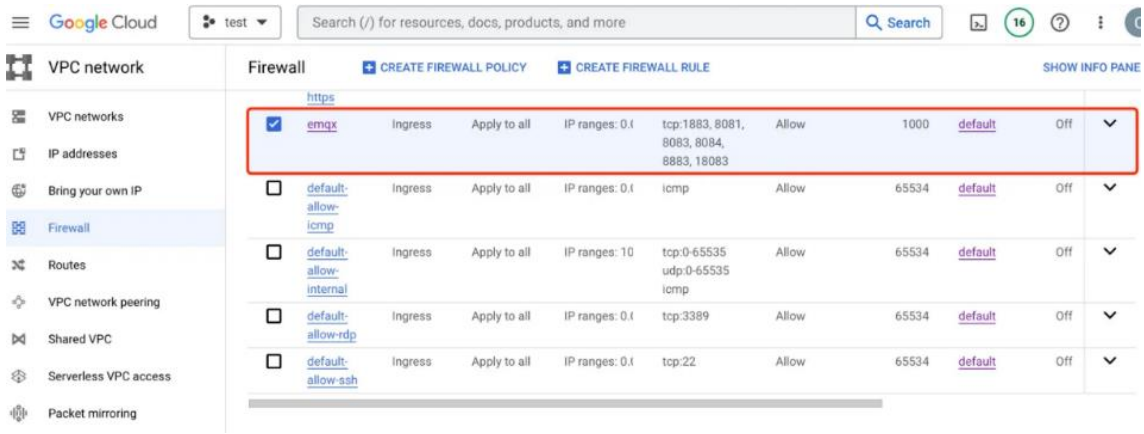

Figura 76. Nueva regla firewall creada

6.5 Instalación Node-Red

En este paso se realizará la instalación de la herramienta de programación Node-Red en la máquina virtual. Se accede al terminal ssh de nuestra máquina virtual y ejecutamos los siguientes comandos:

1. sudo apt-get install build-essential
2. curl -sL https://deb.nodesource.com/setup_16.x | sudo -E bash -
3. sudo apt-get install -y nodejs
4. sudo npm install -g --unsafe-perm node-red
5. sudo npm install -g node-red-admin

Para comprobar que está correctamente instalado ejecutamos el comando "node-red". Cuando termina de ejecutar, devuelve la ruta que debemos escribir en el navegador web.

6.5.1 Configuración Node-Red

Ahora vamos a ejecutar una serie de comandos para dar mayor seguridad al servidor Node-Red.

Se detiene la ejecución y ejecutamos el siguiente comando:

- node-red-admin hash-pw.

Escribimos una contraseña y nos devuelve un hash con la contraseña encriptada. Copiamos el hash para usarlo en el archivo de configuración. Para abrir dicho archivo ejecutamos el comando:

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

Dentro del archivo buscamos el siguiente texto:

```
adminAuth: {
type: "credentials",
users: [{
username: "USUARIO",
password: "HASH_GENERADO",
permissions: "*"
}]
},

// To password protect the node-defined HTTP endpoints (httpNodeRoot), or
// the static content (httpStatic), the following properties can be used.
// The pass field is a bcrypt hash of the password.
// See http://nodered.org/docs/security.html#generating-the-password-hash
httpNodeAuth: {user:"USUARIO",pass:"HASH_GENERADO"},
httpStaticAuth: {user:"USUARIO",pass:"HASH_GENERADO"},
```

Figura 77. Archivo configuración de credenciales

En los textos en rojo escribimos por un lado el usuario que se quiera tener y en *password* y *pass* se pega el hash generado anteriormente.

La primera parte protegerá la parte de gestión (donde colocamos nodos) y la segunda parte protegerá la visualización (la interfaz gráfica).

Por último, se quiere que el servidor Node-Red arranque automáticamente cuando se reinicie. Para ello, se escribe el comando:

- whoami (nos indica nuestro nombre de usuario)

y el comando:

- sudo nano /etc/systemd/system/node-red.service"

Se abre un archivo y copiamos el siguiente texto:

```
[Unit]
Description=Node-RED
After=syslog.target network.target

[Service]
ExecStart=/usr/bin/node-red
Restart=on-failure
KillSignal=SIGINT

# log output to syslog as 'node-red'
SyslogIdentifier=node-red
StandardOutput=syslog

# non-root user to run as
WorkingDirectory=/home/TUUSUARIO/
User=TUUSUARIO
Group=TUUSUARIO

[Install]
WantedBy=multi-user.target
```

Figura 78. Archivo configuración inicio automático

Se modifican los campos en rojo con el nombre de usuario.

Para activar el servicio creado, se ejecuta el comando:

- sudo systemctl enable node-red.
- sudo reboot.

6.6 Código del microcontrolador NodeMCU ESP8266

El código del programa NodeMCU_SN_22.ino se compone principalmente de tres partes. La primera parte es la función principal.

La segunda parte está compuesta por las funciones de conectar, suscribirse y publicar.

La una última parte, formada por las funciones de convertir los datos de red a un formato acorde con la puerta de enlace.

En la siguiente figura se indican todos los parámetros necesarios para el funcionamiento del microcontrolador junto con los sensores conectados. Se muestran los parámetros de conexión a los GPIO del microcontrolador, los parámetros de la red Wi-Fi del hogar, los parámetros de conexión hacia la puerta de enlace y los topic en los que va a publicar y suscribirse el microcontrolador.

NodeMCU_SN_22 §

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
#include "WiFiUdpSocket.h"
#include "MqttSnClient.h"
#include "DHT.h"
#define pinLed 5                                        //GPIO5 = D1 en NodeMCU esp8266 Lolin
#define pinDatos 2                                      //GPIO2 = D4 en NodeMCU esp8266 Lolin
#define buffer_length 10

char buffer[buffer_length + 1];
uint16_t buffer_pos = 0;

const char* ssid     = "MIWIFI_2G_tGjN";
const char* password = "q4sdcd3sfxbq";

IPAddress gatewayIPAddress(192, 168, 1, 131);           //.131 raspb3 Eth0    EMQX 34.116.204.131
uint16_t localUdpPort = 10000;
WiFiUDP udp;
WiFiUdpSocket wiFiUdpSocket(udp, localUdpPort);
MqttSnClient<WiFiUdpSocket> mqttSnClient(wiFiUdpSocket);

DHT sensorTH (pinDatos, DHT22);                         //se crea objeto sensorTH

const char* clientId = "SN_22";
char* subscribe_light = "salon/control/luz";            //salon/control/luz
uint16_t publish_temperature = 2;                       //salon/sensor/temperatura
uint16_t publish_humidity = 3;                          //salon/sensor/humedad
int8_t qos = 0;
```

Figura 79. Parámetros de conexión

En la siguiente figura se muestra la función principal en donde se realizan los intentos de conexión a la red Wi-Fi.

NodeMCU_SN_22

```
int8_t qos = 0;                                         //no se pu

  /*CONECT */
void setup() {
  Serial.begin(115200);
  pinMode(pinLed,OUTPUT);
  sensorTH.begin ();                                    //iniciali
  delay(10);
  Serial.println();
  Serial.print("Conectando a ");
  Serial.println(ssid);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("Conectado al WiFi ");
  Serial.println("Dirección IP: ");
  Serial.println(WiFi.localIP());

  Serial.print("Iniciando cliente MQTT-SN- ");
  mqttSnClient.setCallback(mqttsn_callback);
  if  (!mqttSnClient.begin()) {
    Serial.print("No se ha podido inicializar cliente MQTT-SN ");
    while (true) {
      Serial.println(".");
      delay(1000);
    }
  }
  Serial.println(" ready!");
}
```

Figura 80. Función de conexión a la red Wi-Fi

En la siguiente figura se muestra la función encargada de los intentos de conexión con la puerta de enlace, previamente formateando los datos de dirección IP y puerto de la puerta de enlace para que sean compatibles con la puerta de enlace utilizada.

NodeMCU_SN_22 §

```
68
69 void loop() {
70   if (!mqttSnClient.is_mqttsn_connected()) {
71 #if defined(gatewayHostAddress)
72     IPAddress gatewayIPAddress;
73     if (!WiFi.hostByName(gatewayHostAddress, gatewayIPAddress, 20000)) {
74       Serial.println("No se encuentra Gateway MQTT-SN.");
75       return;
76     }
77 #endif
78     device_address gateway_device_address;
79     convertIPAddressAndPortToDeviceAddress(gatewayIPAddress, localUdpPort, gateway_device_address);
80     Serial.print("Dirección IP Gateway: ");
81     printDeviceAddress(&gateway_device_address);
82
83     if (!mqttSnClient.connect(&gateway_device_address, clientId, 180) ) {
84       Serial.println("No se ha podido conectar el cliente MQTT-SN.");
85       delay(1000);
86       return;
87     }
88     Serial.println("Cliente MQTT-SN conectado.");
89     mqttSnClient.subscribe(subscribe_light, qos);
90   }
91   delay (2000);
92
93   //Lectura valores de temperatura y humedad de los sensores
94   float humidity = sensorTH.readHumidity();
95   char humedad [8];
96   dtostrf(humidity,6,2,humedad);
97   float temperature = sensorTH.readTemperature();
98   char temperatura [8];
99   dtostrf(temperature,6,2,temperatura);
```

Figura 81. Función conexión con puerta de enlace

En la siguiente figura se muestra como el microcontrolador publica los datos de temperatura y humedad, previamente formateados al topic predefinido correspondiente.

NodeMCU_SN_22 §

```
100
101   /*PUBLISH*/
102   mqttSnClient.publish(temperatura, publish_temperature , qos);  //separamos los datos del sensor en topic diferentes
103   mqttSnClient.publish(humedad, publish_humidity , qos);
104   Serial.print ("Temperatura = ");
105   Serial.print (temperatura);
106   Serial.println (" °C");
107   Serial.print ("Humedad = ");
108   Serial.print (humedad);
109   Serial.println (" %");
110
111   mqttSnClient.loop();
112 }
```

Figura 82. Función de publicación de datos

En la siguiente figura se muestra la suscripción del microcontrolador al topic predefinido correspondiente utilizando una función callback.

NodeMCU_SN_22 §

```
113
114   /*SUSCRIBE*/
115 void mqttsn_callback(char *topic, uint8_t *payload, uint16_t length, bool retain) {
116   Serial.print("Received - Topic: ");
117   Serial.print(topic);
118   Serial.print(" Payload: ");
119   String message;
120   for (uint16_t i = 0; i < length; i++) {
121     char c =  (char) * (payload + i);
122     message += (char)payload[i];
123     Serial.print(c);
124   }
125   Serial.print(" Lenght: ");
126   Serial.println(length);
127   if (String(topic) == "salon/control/luz") {
128     if(message == "on"){
129       Serial.println("on");
130       digitalWrite(pinLed, HIGH);
131     }
132     else if(message == "off"){
133       Serial.println("off");
134       digitalWrite(pinLed, LOW);
135     }
136   }
137 }
```

Figura 83. Función de suscripción de datos

En la siguiente figura se muestra la función encargada de la conversión de formato de la IP y puerto para que sea compatible con la puerta de enlace Paho Gateway.

```
139 void convertIPAddressAndPortToDeviceAddress(IPAddress& source, uint16_t port, device_address& target) {
140   // IPAdress 0 - 3 bytes
141   target.bytes[0] = source[0];
142   target.bytes[1] = source[1];
143   target.bytes[2] = source[2];
144   target.bytes[3] = source[3];
145   // Port 4 - 5 bytes
146   target.bytes[4] = port >> 8;
147   target.bytes[5] = (uint8_t) port ;
148 }
```

Figura 84. Función de formateo IP y puerto