UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Desarrollo de una aplicación IoT para la gestión de un hogar inteligente mediante el protocolo MQTT y Sistemas en chip (SoC) ESP32

Raynel Moreno Hernández

Tutor: Antonio León Fernández

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-2020

Valencia, 9 de septiembre de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV- 000030

Resumen

El proyecto tiene como objetivo el diseño de una aplicación IoT para la gestión de un hogar inteligente, procurando ser una propuesta económica y de fácil implementación que nos ayude en el quehacer diario en nuestras casas.

La idea es que, utilizando varios sensores conectados a internet mediante la red wifi-doméstica, podamos realizar tareas desde nuestros terminales móviles como: detección de fugas de gas, alertar de incendios, automatización del riego de plantas ornamentales o tener datos de temperatura y humedad del interior de nuestras casas. Todo esto controlado por un chip muy económico, pero con muchísimas prestaciones como es el ESP32.

Como protocolo de comunicación entre los dispositivos y el servidor utilizaremos el MQTT que ha ganado gran popularidad por su simpleza y mínimo coste computacional. El cerebro de nuestra aplicación estará diseñado en Node Red el cual permite, mediante una interfaz gráfica muy sencilla, programar toda la lógica de nuestro proyecto.

Resum

El projecte té com a objectiu el disseny d'una aplicació *IoT per a la gestió d'una llar intel·ligent, procurant ser una proposta econòmica i de fàcil implementació que ens ajude en el quefer diari a les nostres cases.

La idea és que, utilitzant diversos sensors connectats a internet mitjançant la xarxa wifi-domèstica, puguem fer tasques des dels nostres terminals mòbils com: detecció de fugues de gas, alertar d'incendis, automatització del reg de plantes ornamentals o tindre dades de temperatura i humitat de l'interior de les nostres cases. Tot això controlat per un xip molt econòmic, però amb moltíssimes prestacions com és l'ESP32.

Com a protocol de comunicació entre els dispositius i el servidor utilitzarem el *MQTT que ha guanyat gran popularitat per la seua ximplesa i mínim cost computacional. El cervell de la nostra aplicació estarà dissenyat en *Node Xarxa el qual permet, mitjançant una interfície gràfica molt senzilla, programar tota la lògica del nostre projecte.

Abstract

The project aims to design an IoT application for the management of a smart home, trying to be an economical and easy-to-implement proposal that helps us in the daily routine of our homes.

The idea is to use, using various sensors connected to the internet through the Wi-Fi network, we can perform tasks from our mobile terminals such as: detection of gas leaks, alerting of fires, automation of irrigation of ornamental plants or having temperature and humidity data of the interior of our houses. All this controlled by a very economical chip, but with many benefits which the ESP32.

As a communication protocol between the devices and the server, we will use the MQTT, which has gained great popularity for its simplicity and minimal computational cost. The brain of our application will be designed in Node Red, which allows, through a very simple graphical interface, to program all the logic of our project.

Índice

Capítulo 1. Introducción

1.1 Motivación.

El desarrollo de este proyecto surge con el afán de llevar a la práctica los conocimientos adquiridos durante la carrera universitaria y de materializarlo en una implementación útil que ayudara a mejorar la vida de las personas. Después de un extenso estudio, la tecnología del IoT (**Internet of Things**), fue sin dudas el candidato número uno que motivó a la realización de este trabajo final de carrera.

La infinidad de implementaciones que está teniendo esta tecnología, demuestran que el **IoT** ya no es cosa del futuro, sino cada vez más del presente. Utilizarlo en todo tipo de productos y servicios es, por tanto, una forma de ser más eficientes, mejorar las comunicaciones y controlar cada fase y momento de la producción. Permite generar datos muy interesantes para determinar las evoluciones de los productos y servicios y poder, de este modo, mejorarlos o adaptarlos en el futuro. Sin dudas, el desarrollo de esta nueva tecnología ha revolucionado el sector industrial, trayendo consigo el surgimiento de la llamada Industria 4.0.

Pero estas soluciones no sólo las vemos en los grandes sectores industriales, sino que cada día es más frecuente su utilización en entornos domésticos. El flujo de información de electrodomésticos y dispositivos inteligentes, diseñados para facilitarnos la vida en nuestros hogares, con la capacidad añadida de intercambiar información a Internet es lo que conocemos popularmente como domótica.

Precisamente, basados en la idea de la domótica, se desarrolla el contexto de este trabajo. Se pretende el desarrollo de una aplicación IoT que ofrezca una solución asequible y sencilla, que permita obtener información en tiempo real de diversos sensores distribuidos por el hogar.

1.2 Objetivos del TFG

Con el presente trabajo se pretende desarrollar una aplicación IoT completa para el control domótico de una vivienda. El objetivo principal será el diseño de una aplicación con Node-RED, que permita controlar varios sensores mediante el estandarizado protocolo de comunicación MQTT. Como es característico en este protocolo, será necesario la instalación de un broker MQTT, el cual será el encargado de recibir todos los mensajes, filtrarlos según los tópicos y luego publicar los mensajes en los correspondientes clientes suscritos. En este caso, la instalación del broker se llevará a cabo en una instancia de una máquina virtual en Google Cloud, aprovechando las ventajas tecnológicas y económicas que esta ofrece.

Para lograr una mejor descripción de todos los puntos a tratar y a su vez, lograr una mejor comprensión del desarrollo del proyecto, dicho objetivo principal se divide a su vez en otros subobjetivos.

Los subobjetivos quedan dispuestos de la siguiente forma:

- Realizar un previo estudio sobre las soluciones actuales del IoT en el mundo de la domótica, con la idea de que esta información sirva de base para el desarrollo del presente proyecto.
- Conocer a fondo el funcionamiento el protocolo MQTT para lograr una correcta implementación en el mecanismo de comunicación entre los dispositivos.
- Adquirir los conocimientos de programación necesarios para poder utilizar el microprocesador ESP32, encargado de controlar las funciones de los sensores.
- Familiarización con el lenguaje de programación Phyton, el cual será utilizado para programar el intercambio de información entre los dispositivos.

- Investigar sobre los sensores más utilizados en el mercado de la dómotica, sus características y funcionalidades, con el objetivo de que los utilizados cumplan con los requisitos del proyecto.
- Relacionarse con las características de funcionamiento y pinout de los distintos sensores escogidos: **DTH11**, **MQ-135**, **ESP32-CAM**, **KY-026**, **Capacitive Soil Moisture Sensor**, **Módulo Wifi Relé de 5V** y el **FT232RL FTDI**, convertidor de RS232 a USB necesario para programar el ESP32-CAM.
- Aprender a manejar la herramienta Node-RED, programador visual que permitirá la creación de la interfaz gráfica del trabajo.

1.3 Estructura de la memoria

La memoria de este trabajo está estructurada en 5 capítulos, en los cuales se pretende abordar de manera detallada el desarrollo de este proyecto y los anexos finales que muestra el funcionamiento de la aplicación.

Se comienza haciendo una breve explicación de la repercusión que tiene hoy en día la tecnología del IoT, pasando luego a analizar algunos aspectos teóricos relacionados con el proyecto y se termina, con las conclusiones y algunas propuestas para implementaciones futuras que darían continuidad a este trabajo de investigación.

En el **Capítulo 1** se explica las motivaciones que impulsaron al desarrollo de este trabajo fin de carrera. Luego se detallan los objetivos a tratar y seguidamente se define la estructura de la memoria. Se cierra el primer capítulo explicando la metodología utilizada en la realización del trabajo.

En el **Capítulo 2,** a modo de resumen, se detallan las características de las tecnologías a utilizar. Se comienza con el análisis del protocolo MQTT, luego se hace un breve comentario de las especificaciones técnicas del chip ESP32 y se termina explicando las características de los sensores utilizados en la aplicación domótica.

Se continúa con el **Capítulo 3**, capítulo de vital importancia, ya que se expone toda la parte del desarrollo de la aplicación. Se explicará mediante el plano de un apartamento convencional, cómo quedarían dispuestos los diferentes sensores, teniendo en cuenta su funcionalidad y los datos en los que se está interesado en monitorizar. Además, se explicará el proceso seguido para la instalación del Broker MQTT en la plataforma Google Cloud, las ventajas que nos posibilita y sus limitaciones. Como última parte de este capítulo, se hará una descripción de la aplicación creada en Node-RED, se explicarán los flujos elegidos para diseñar el programa y la lógica que posibilita su funcionamiento.

Seguidamente, en el **Capítulo 4** analizaremos los costes asociados a la realización de este proyecto, las ideas de trabajo futuro para continuar con su desarrollo y las conclusiones.

En el **Capítulo 5** se detallarán las fuentes bibliográficas. Se especificarán los enlaces url, indicando el día y la hora de las consultas realizadas a cada una de ellas.

Cómo última parte de esta memoria están los **Anexos**. Aquí se pretende mostrar mediante capturas de pantalla, algunos de los procesos llevados a cabo para la configuración de la aplicación, explicar parte del código principal del programa y el significado de muchas de las siglas utilizadas en el proyecto.

1.4 Metodología

La metodología para realizar el trabajo parte de una ardua tarea de investigación, seguida de la debida selección de la información recopilada, su análisis, asimilación de los conocimientos necesarios para llevar a cabo el proyecto, implementación y, por último, el resumen de todo lo aprendido durante el proceso. Por lo tanto, es necesario definir una serie de tareas que nos permita dividir el trabajo para ir desarrollando progresivamente cada uno de los objetivos que se persiguen en este TFG.

Las tareas propuestas son las siguientes:

- **Tarea 1:** Recopilación de información sobre el mundo del IoT enfocado principalmente al sector de la domótica, con el objetivo que sirva de guía y motivación en el desarrollo del trabajo.
- **Tarea 2:** Investigar sobre el funcionamiento del protocolo MQTT. Encontrar entre las múltiples soluciones actuales la forma de crear un broker MQTT propio, que cumpla tanto, con las exigencias técnicas como financieras del proyecto.
- **Tarea 3:** Explorar las diferentes opciones que ofrece la plataforma Google Cloud, con el objetivo de aprovechar estas ventajas para crear el servidor virtual en el cual se instalará el broker MQTT.
- **Tarea 4:** Estudiar las especificaciones técnicas del chip ESP32. Buscar ejemplos de proyectos IoT en los que se emplee este chip y el protocolo MQTT, que sirvan de base para la realización del TFG.
- **Tarea 5:** Realizar una búsqueda minuciosa de sensores que cumplan con las exigencias de las funciones que se desean implementar, para después realizar su posterior compra en Internet.
- **Tarea 6:** Una vez definidos los elementos que formarán parte de la aplicación, aprender a configurarlos correctamente y hacer que puedan interactuar entre ellos.
- **Tarea 7:** Aprender a manejar la herramienta Node-RED, en la cual se creará el backend del proyecto. En esta aplicación, se coordinarán las diferentes acciones entre el servidor MQTT y los microcontroladores conectados a los respectivos sensores.
- **Tarea 8:** Definir el esquema del flujo de la aplicación y una vez plasmado, comenzar con el desarrollo de la aplicación. Realizar las comprobaciones y validaciones pertinentes.
- **Tarea 9:** Definir las conclusiones del proyecto y tareas futuras.

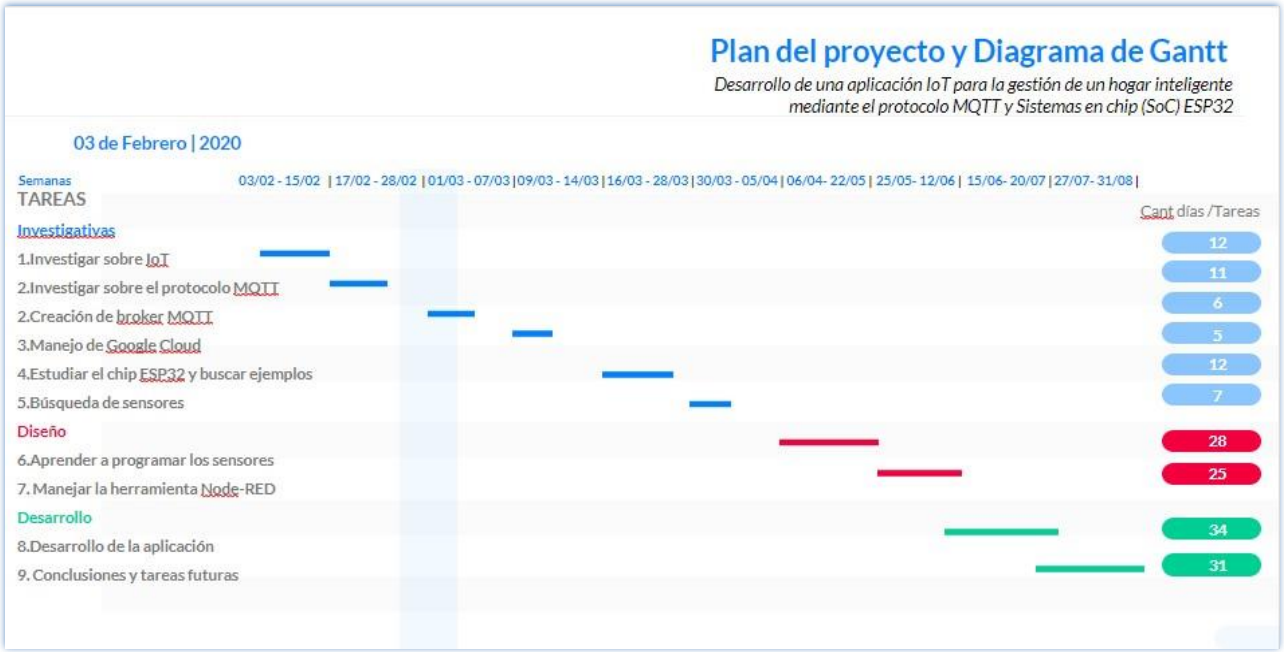


Figura 1. Plan de proyecto y Diagrama de Gantt.

Capítulo 2. Tecnologías

2.1 Protocolo MQTT

2.1.1 Introducción

El protocolo MQTT [1] antes conocido como MQ Telemetry Transport, es un protocolo ligero y simple de transporte de mensajes con modelo publicador/ suscriptor. Por estas características, ha ganado gran popularidad en aplicaciones como telemetría M2M (máquina a máquina) e Internet de las cosas (IoT) donde se requiere una pequeña huella de código.

2.1.2 Historia

Fue creado en 1999 por el Dr. Andy Stanford-Clark de IBM y Arlen Nipper de Eurotech, como parte de un proyecto para conectar sistemas de telemetría en oleoductos de petróleo por satélites a través del desierto. El objetivo principal de esta solución era tener un protocolo liviano que garantizase que el consumo de ancho de banda y energía de los dispositivos fuese mínimo.

Los dos inventores plantearon varios requisitos [2] como bases del protocolo:

- Ligero y ancho de banda eficiente
- Entrega de datos con calidad de servicio
- Implementación simple
- Conciencia continua de la sesión
- Agnóstico de datos

Aunque comenzó como un protocolo propietario de IBM, se lanzó Royalty free en 2010 con la versión **MQTT 3.1**. Cuatro años más tarde se convirtió en un estándar OASIS en 2014 con la nueva versión **MQTT 3.1.1** aunque con pocos cambios.

2.1.3 Conceptos básicos de MQTT

MQTT se basa en la arquitectura **publicador/suscriptor** (Figura 2), la cual proporciona una alternativa a la tradicional de **cliente-servidor**. En el modelo cliente-servidor, un cliente se comunica directamente con un host final. Sin embargo, con el nuevo modelo **pub/sub** se elimina la comunicación directa entre publicador (remitente) y el suscriptor (destinatario) existiendo un desacoplamiento entre ambos [3]. La actividad de filtrado del broker hace posible controlar qué cliente / suscriptor recibe qué mensaje.

Características principales de la arquitectura Pub/Sub:

- El desacoplamiento tiene tres dimensiones: espacio, tiempo y sincronización
- Mayor escalabilidad
- Filtrado de mensajes (por temas, contenido y clases)

La topología en estrella es una de las principales características de la arquitectura de MQTT, formado por dos entidades: ***broker*** y ***cliente***. Esta topología junto con las funciones del broker, permiten al protocolo mantener comunicaciones unicast y multicast.

Broker MQTT: Es el encargado de recibir, filtrar, determinar quién está suscrito a cada uno de los mensajes y enviarlos a estos clientes suscritos. Contiene los datos de sesión de todos los clientes que tienen sesiones persistentes, incluidas las suscripciones y los mensajes perdidos. Otra responsabilidad del broker es la autenticación y autorización de los clientes.

Cliente MQTT: se trata de cualquier dispositivo que ejecuta una biblioteca MQTT y se conecta con un broker MQTT a través de una red. En términos generales, el cliente que envía el mensaje se llama publicador y quien recibe el mensaje se llama suscriptor, aunque un cliente puede hacer la función de suscriptor y editor a la vez.

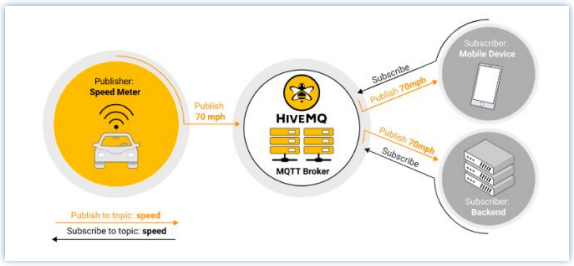


Figura 2. Arquitectura de MQTT "publicador/suscriptor"

2.1.4 Establecimiento de la conexión MQTT

El protocolo se ejecuta principalmente sobre TCP/IP (Figura 3), sin embargo, cualquier protocolo de red que proporcione conexiones bidireccionales ordenadas y sin pérdidas puede admitir MQTT.

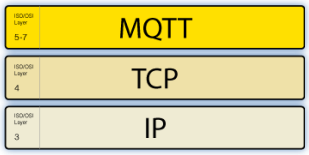


Figura 3 Capas en el protocolo MQTT.

2.1.4.1 Mensaje CONNECT

La conexión MQTT es siempre entre un cliente y el broker. Esto da independencia a los editores y suscriptores porque no necesitan saber la existencia del otro para comunicarse (no se intercambian la dirección IP ni el puerto).

Como primer paso [3], el cliente envía una petición de conexión al broker con el mensaje CONNECT (Figura 4). Si no hay problemas en la conexión, el broker responderá con un mensaje CONNACK. Así quedará establecida la conexión entre el cliente y el broker hasta que el cliente envíe un mensaje de desconexión.

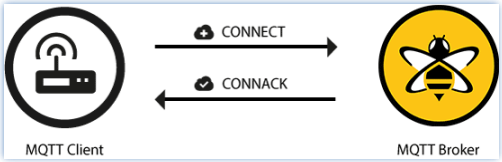


Figura 4 Establecimiento de la conexión MQTT.

Información contenida en el mensaje CONNECT:

- ***ClientId:*** identifica a los clientes MQTT y su estado actual.
- ***CleanSession:*** indica al broker si el cliente desea establecer una sesión persistente o no. (CleanSession = false) es una sesión persistente, donde el broker almacena información de las conexiones del cliente con QoS 1 o QoS 2. (CleanSession = true) sesión no persistente, el intermediario no almacena información del cliente.
- ***Username/Password:*** nombre de usuario y contraseña para la autenticación y autorización del cliente.
- ***Will Message:*** si el cliente se desconecta repentinamente, el intermediario envía el mensaje LWT (última voluntad y testamento) para notificar a los otros clientes. Este indica el tema, indicador de mensaje retenido, QoS y carga útil.

- ***KeepAlive:*** garantiza que la conexión entre el broker y el cliente todavía está abierta y que son conscientes de estar conectados. Es el período de tiempo más largo que el broker y el cliente pueden soportar sin enviar un mensaje. PINGREQ es enviado por el cliente e indica al broker que el cliente aún está activo. Responde con un paquete PINGRESP para mostrarle al cliente que todavía está disponible.

2.1.4.2 Mensaje CONNACK

El servidor responderá con un mensaje CONNACK al cliente, después de recibir el mensaje de petición de conexión. Si el cliente no recibe un paquete CONNACK por parte del broker en cierto período de tiempo, el cliente cerrará la sesión e intentará nuevamente establecer la conexión.

El mensaje CONNACK contiene dos entradas de datos:

- ***sessionPresent:*** (indicador de sesión actual) le dice al cliente si el broker ya tiene una sesión persistente disponible de interacciones anteriores.
- ***returnCode:*** código de retorno que le dice al cliente si el intento de conexión fue exitoso.

Otro caso en el que se deberá restablecer la conexión con el broker, es si CONNACK contiene un valor distinto de cero. Esto significa que el servidor no puede procesar el paquete de conexión por alguna razón. Entonces el servidor debe cerrar la conexión de red.

2.1.5 Publicación, suscripción y cancelación de suscripción de MQTT

Un cliente MQTT puede publicar mensajes una vez establecida la conexión con el broker [4]. MQTT filtra los mensajes en el broker basado en los temas de cada mensaje, los cuales usa para reenviar a los clientes suscritos. El intercambio de mensajes para la publicación se detalla en la Figura 5

2.1.5.1 Mensaje PUBLISH

Atributos del mensaje PUBLISH:

- ***topicName:*** Topic o tema de los mensajes**.** Tiene estructura jerárquica o ramificada separada con el signo "/". Una suscripción puede ser a un tema explícito o puede incluir comodines. Hay dos comodines disponibles: **+ (**para un solo nivel de jerarquía**)** o **# (**para todos los niveles restantes de jerarquía**)**. Los temas que comienzan con un símbolo $ están reservados para las estadísticas internas del broker MQTT. A modo de seguridad, los tópicos deben de ser lo más personalizados posible de manera que no se produzcan conflictos.
- ***retainFlag:*** Este indicador define si el intermediario guarda el mensaje como el último valor válido conocido para un tema específico.
- ***Payload:*** contenido real del mensaje.
- ***packetId:*** identifica de forma exclusiva un mensaje a medida que fluye entre el cliente y el intermediario.
- ***dupFlag:*** indica si el mensaje es un duplicado y se reenvió porque el destinatario (cliente o broker) no reconoció el mensaje original.
- ***QoS:*** MQTT define tres niveles de calidad de servicio (QoS): 0,1 y 2. La calidad del servicio [5] define la robustez con la que el broker intentará garantizar que se reciba un mensaje. Si el cliente suscriptor define una QoS más baja que el cliente de publicación, el intermediario transmite el mensaje con la menor calidad de servicio. Los niveles más altos de QoS son más confiables, pero implican una latencia más alta y tienen mayores requisitos de ancho de banda.

Niveles de calidad de servicios:

- ***QoS 0:*** Se entregará el mensaje una vez, sin confirmación. Este nivel es el usado por defecto. No hay garantía de entrega

Figura 5. Publish QoS 0

- ***QoS 1:*** El cliente asegura que entregará el mensaje al menos una vez, hasta que el broker confirme el envío requerido a la red con el mensaje PUBACK (Figura 6).

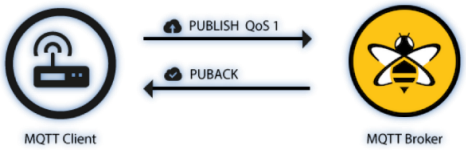


Figura 6. Publish QoS 1

- ***QoS 2:*** El cliente garantiza que entregará el mensaje exactamente una vez y utiliza como mecanismo de confirmación de recepción un handshake de cuatro pasos (Figura 7).

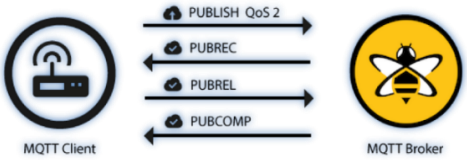


Figura 7. Publish QoS 2

Cuando un receptor recibe el mensaje PUBLISH de un remitente con QoS 2, procesa el mensaje de publicación en consecuencia y responde al remitente con un paquete PUBREC que reconoce el paquete PUBLISH. Una vez el remitente recibe y almacena el paquete PUBREC del receptor, responde con un paquete PUBREL. Recibido el paquete PUBREL, el receptor puede descartar todos los estados almacenados y responde con un paquete PUBCOMP (lo mismo sucede con el remitente).

Hasta que el receptor complete el procesamiento y envíe el paquete PUBCOMP al remitente, el receptor almacena una referencia al identificador del paquete PUBLISH original. Este paso es importante para evitar procesar el mensaje por segunda vez.

2.1.5.2 Mensaje SUBSCRIBE

Para recibir mensajes sobre temas de interés, el cliente envía un mensaje SUBSCRIBE (Figura 8) al agente MQTT. Este mensaje de suscripción es muy simple y cuenta con dos atributos:

- ***packetId:*** que identifica a dicho paquete.
- ***listSubscriptions:*** Un mensaje SUBSCRIBE puede contener múltiples suscripciones para un cliente. Cada suscripción se compone de un tema y un nivel de QoS.

2.1.5.3 Mensaje SUBACK

Para confirmar cada suscripción, el agente envía un mensaje de confirmación SUBACK al cliente.

- ***packetId:*** identifica el mensaje SUBACK. Este ID coincide con el del mensaje SUBSCRIBE.
- ***returnCode #:*** El intermediario envía un código de retorno para cada par tema y QoS que recibe en el mensaje SUBSCRIBE.

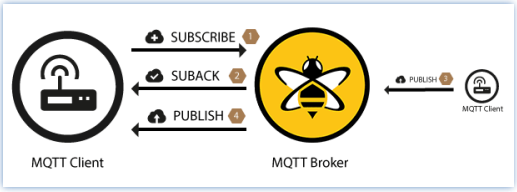


Figura 8 Suscripción de un cliente a un tema en el broker.

2.1.5.4 Mensaje UNSUBSCRIBE.

El mensaje UNSUBSCRIBE elimina las suscripciones existentes de un cliente en el broker. Este comando es similar al mensaje SUBSCRIBE. Contiene un identificador de paquete (***packetId)*** y una lista de temas (***topic #***) de los cuales quiere darse de baja independientemente de la QoS.

2.1.5.5 Mensaje UNSUBACK.

Para confirmar la cancelación de la suscripción, el broker envía un mensaje de confirmación UNSUBACK al cliente. Este mensaje contiene solo el identificador de paquete (***packetId***) que coincide con el del mensaje UNSUBCRIBE.

2.1.5.6 Mensaje DISCONNECT

El paquete de DESCONEXIÓN es el paquete de control final enviado desde el cliente al servidor. Indica que el cliente se está desconectando limpiamente. Las características principales de este paquete DISCONNECT es que no tiene payload y no tiene encabezado variable.

Después que el broker envía un paquete DISCONNECT al cliente:

- Debe cerrar la conexión de red.
- No debe enviar más paquetes de control en esa conexión de red.

Cuando el broker recibe un paquete DISCONNECT del cliente:

- Debe descartar cualquier paquete Will Message asociado con la conexión actual.
- Debe cerrar la conexión de red si el cliente aún no lo ha hecho.

2.1.6 Retención de mensajes

Un mensaje retenido [6] es un mensaje MQTT normal con el indicador retenido establecido en verdadero. El broker almacena el último mensaje retenido y la QoS correspondiente para ese tema. Esto significa que el broker mantendrá el mensaje incluso después de enviarlo a todos los suscriptores actuales. Si se realiza una nueva suscripción que coincide con el topic del mensaje retenido, el mensaje se enviará al cliente. Esto es útil si un tema solo se actualiza con poca frecuencia, porque permite que un cliente recién suscrito no tenga que esperar mucho tiempo para recibir una actualización.

2.1.7 Seguridad en el protocolo MQTT

La seguridad en MQTT se divide en diferentes capas, el nivel de red, el nivel de transporte y el nivel de aplicación [7]

Nivel de red

Una forma de proporcionar una conexión segura y confiable es usar una red físicamente segura o VPN para toda comunicación entre clientes y servidores. Esta solución es adecuada para aplicaciones de puerta de enlace donde está conectada a dispositivos por un lado y con el agente a través de VPN por el otro.

Nivel de transporte

Transport Layer Security (TLS) y Secure Sockets Layer (SSL) proporcionan un canal de comunicación seguro entre un cliente y un servidor. Este método es una forma segura y comprobada de garantizar que los datos no se puedan leer durante la transmisión. Con Transport Layer Security (TLS), la validación exitosa de un certificado de cliente se utiliza para autenticar al cliente en el servidor. El puerto 8883 está reservado exclusivamente para MQTT sobre TLS, para cifrar toda la comunicación.

Nivel de aplicación

En el nivel de transporte, la comunicación se cifra y las identidades se autentican. El protocolo MQTT proporciona un identificador de cliente y credenciales de nombre de usuario / contraseña para autenticar dispositivos en el nivel de la aplicación. Estas propiedades son proporcionadas por el propio protocolo. La autorización o el control de lo que cada dispositivo puede hacer está definido por la implementación específica del broker. Además, es posible utilizar el cifrado de carga útil (payload) en el nivel de la aplicación. Esto asegura que la información transmitida vaya cifrada sin la necesidad de un cifrado de transporte completo.

2.2 Chip Az-Delivery ESP32-WROOM-32.

2.2.1 Introducción.

Este chip (Figura 9) va a ser clave para la solución de este proyecto como herramienta hardware al ser un requisito principal, y el exigido por el tutor. Es realmente llamativo por su gran variedad de aplicaciones con un tamaño tan pequeño a un bajo costo.

ESP32-WROOM-32 es un potente y genérico módulo Wi-Fi + BT + BLE MCU sucesor del muy popular ESP8266. El cambio más notable de este chip es la conectividad Bluetooth 4.2 BLE agregada. Este chip [8] tiene una amplia variedad de aplicaciones en el mundo de Internet de las cosas (Internet of Things), que van desde redes de sensores de baja potencia hasta las tareas más exigentes, como codificación de voz, transmisión de música y decodificación de MP3.

A todas estas ventajas se suma, que integra un rico conjunto de periféricos que van desde sensores táctiles capacitivos hasta sensores Hall, interfaz de tarjeta SD, Ethernet, SPI de alta velocidad, UART, I²S y I²C.

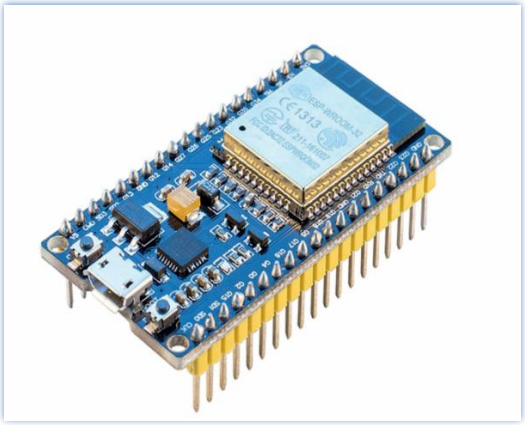


Figura 9. ESP32-WROOM-32

Actualmente hay muchas maneras de programar el chip como: MicroPython (que implementa Python), ESPlorer (que ofrece la posibilidad de la utilización de comandos LUA, Microphyton y comandos AT) y Arduino entre otros.

Finalmente se ha apostado por el uso del software Arduino IDE por las ventajas que ofrece, descritas en el apartado 2.5.

2.2.2 Características

En el núcleo de este módulo se encuentra el chip ESP32-D0WDQ6. El chip integrado está diseñado para ser escalable. Hay dos núcleos de CPU que pueden controlarse individualmente, y la frecuencia de reloj de la CPU es ajustable de 80 MHz a 240 MHz. El usuario también puede apagar la CPU y utilizar el coprocesador de baja potencia, para supervisar constantemente los periféricos y detectar cambios o cruzar umbrales.

La integración de Bluetooth, Bluetooth LE y Wi-Fi garantiza que se pueda apuntar a una amplia gama de aplicaciones. El uso de Wi-Fi permite un amplio rango físico y una conexión directa a Internet a través del router, mientras que el uso de Bluetooth permite al usuario conectarse convenientemente al teléfono o transmitir señales de baja potencia para su detección.

La corriente de reposo del chip ESP32 es inferior a 5 A, lo que lo hace adecuado para aplicaciones electrónicas con baterías y dispositivos portátiles. El módulo admite una velocidad de datos de hasta 150 Mbps y 20 dBm de potencia de salida en la antena, lo que permite garantizar un rango físico más amplio. Como tal, el módulo ofrece especificaciones líderes en la industria y el mejor rendimiento para integración electrónica, rango, consumo de energía y conectividad.

El sistema operativo elegido para ESP32 es freeRTOS con LwIP; TLS 1.2 con aceleración de hardware integrado. La actualización segura (encriptada) por aire (OTA) también es compatible, para que los usuarios puedan actualizar sus productos incluso después de su lanzamiento, a un costo y esfuerzo mínimos.

Las especificaciones principales de este dispositivo se aprecian en la Tabla 1.

Tabla 1. Especificaciones de ESP32-WROOM-32

Categorías	Elementos	Especificaciones
Certificación	Certificación RF	FCC/CE-RED/IC/TELEC/KCC/SRRC/NCC
	Certificación Wi-Fi	Wi-Fi Alliance
	Certificación Bluetooth	BQB
	Certificación ambiental	RoHS/REACH
Test	Fiabilidad	HTOL/HTSL/uHAST/TCT/ESD
Wi-Fi	Protocolos	802.11 b/g/n (802.11n hasta 150 Mbps)
		Agregación A-MPDU y A-MSDU e intervalo de guarda de apoyo de 0.4 s
	Rango de frecuencias	2.4 GHz ~ 2.5 GHz
Bluetooth	Protocolos	Bluetooth v4.2 BR / EDR y especificación BLE
	Radio	Receptor NZIF con sensibilidad de –97 dBm

		Transmisor de clase 1, clase 2 y clase 3
		AFH
	Audio	CVSD y SBC
Hardware	Interfaces de módulos	Tarjeta SD, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2S, IR, contador de pulsos, GPIO, sensor táctil capacitivo, ADC, DAC
	Sensor en chip	Sensor Hall
	Cristal integrado	Cristal de 40 MHz
	Flash SPI integrado	4 MB
	Tensión de funcionamiento / fuente de alimentación	3.0 V ~ 3.6 V
	Corriente de funcionamiento	Promedio: 80 mA
	Corriente mínima entregada por fuente de alimentación	500 mA
	Rango de temperaturas de funcionamiento recomendadas	–40 °C ~ +85 °C
	Tamaño del embalaje	(18.00±0.10) mm × (25.50±0.10) mm × (3.10±0.10) mm
	Nivel de sensibilidad a la humedad (MSL)	Nivel 3

2.2.3 Pinout

El chip ESP32-WROOM-32 tiene un total de 38 pines ,tal y como se muestra en la figura 10, 4 de los cuales están relacionados con las entradas de alimentación (3 pines con conexión a tierra y 1 pin para la alimentación del dispositivo a 3V), 5 pines de entradas, 28 pines con función de entrada y salida y el pin NC sin una función especificada como se detalla en la Figura 9. Hay que señalar que el flash SPI tiene integrado en el módulo los pines: SCK / CLK, SDO / SD0, SDI / SD1, SHD / SD2, SWP / SD3 y SCS / CMD denotados en la placa de la GPIO6 a la GPIO11 y los cuales no se recomiendan para otros usos.

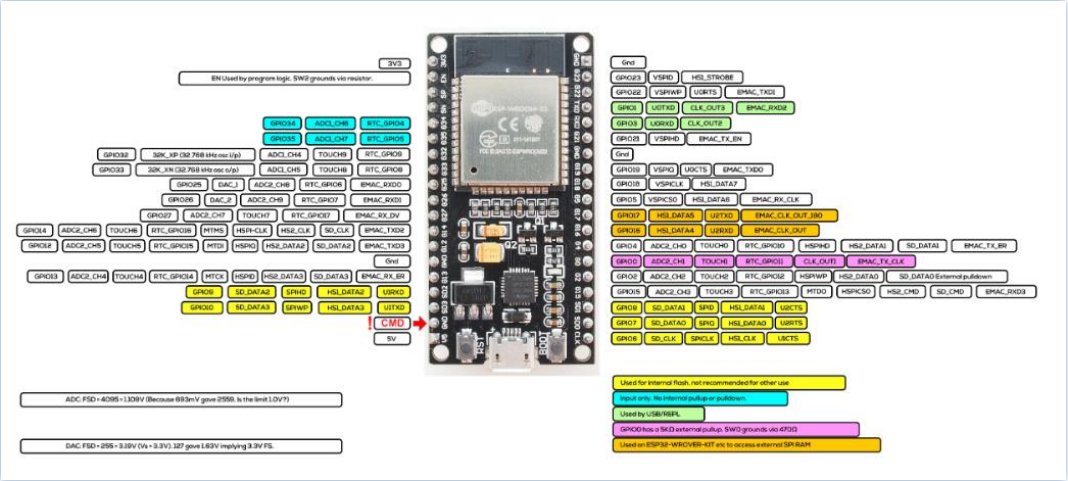


Figura 10. Pinout del chip ESP32-WROOM-32.

Cuenta además con cinco pines de flejado: MTDI, GPIO0, GPIO2, MTDO, GPIO5. Estos pines de flejado muestran el nivel de voltaje como bits de flejado de "0" o "1", y sostienen estos bits hasta que el chip está apagado. Los bits de flejado son utilizados para configurar el modo de arranque del dispositivo, el voltaje operativo de VDD_SDIO y otras configuraciones iniciales del sistema. El software para esta función lee los valores de cinco bits en el registro "GPIO_STRAPPING". Después de liberar el restablecimiento, los pines de flejado funcionan como pines de función normal.

2.2.4 CPU y Memoria Interna

Contiene dos microprocesadores Xtensa® LX6 de 32 bits de baja potencia. La memoria interna incluye:

• 448 KB de ROM para arranque y funciones principales.

• 520 KB de SRAM en chip para datos e instrucciones.

• 8 KB de SRAM en RTC, que se llama RTC FAST Memory y se puede usar para el almacenamiento de datos; se accede por la CPU principal durante el arranque RTC desde el modo de reposo profundo.

• 8 KB de SRAM en RTC, que se denomina memoria RTC SLOW y a la que puede acceder el coprocesador durante la función en modo de sueño profundo.

• 1 Kbit de eFuse: se utilizan 256 bits para el sistema (dirección MAC y configuración de chip)

2.2.5 Flash Externa y SRAM

ESP32 es compatible con múltiples flashes externas QSPI y chips SRAM a las cuales se pueden acceder a través de cachés de alta velocidad. Admite cifrado/ descifrado de hardware basado en AES para proteger programas y datos de desarrolladores en el flash.

El flash externo se puede asignar al espacio de memoria de instrucciones de la CPU (hasta 11 MB + 248 KB mapeados a la vez) y al espacio de memoria (hasta 4 MB en una hora) de solo lectura simultáneamente. Se admiten lecturas de 8 bits, 16 bits y 32 bits.

Como se mencionó antes, el ESP32-WROOM-32 integra un flash SPI de 4 MB, que está conectado a los pines GPIO6, GPIO7, GPIO8, GPIO9, GPIO10 y GPIO11. Estos no se pueden usar como GPIO normales.

2.3 Sensores utilizados

2.3.1 ESP32-CAM

El ESP32-CAM (figura 11) es un módulo de cámara muy pequeño y económico que cuenta con el chip ESP32-S. Además de la cámara OV2640 y varios GPIO para conectar periféricos, también cuenta con una ranura para tarjeta microSD que puede ser útil para almacenar imágenes tomadas con la cámara o para almacenar archivos para servir a los clientes.

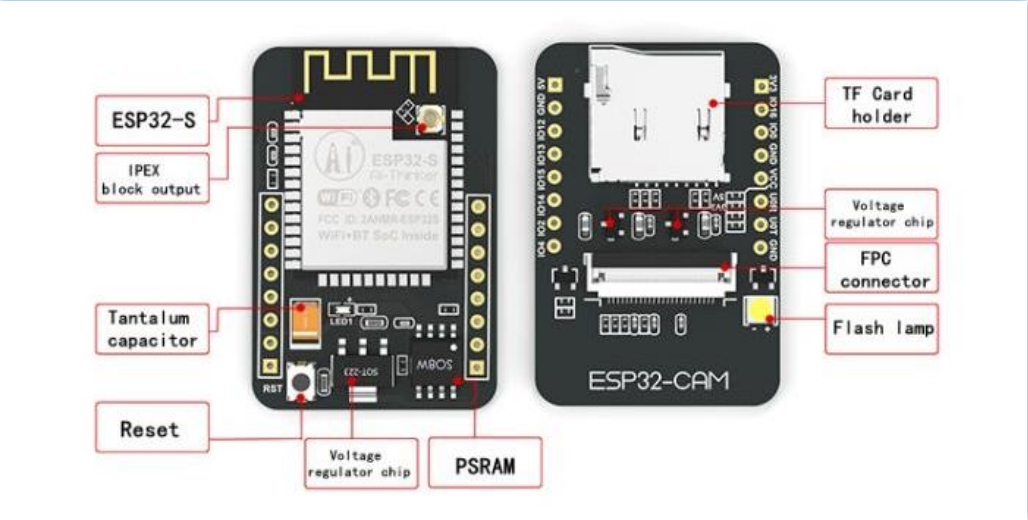


Figura 11.Estructura del ESP 32 CAM.

Principales características de ESP32-CAM

- El módulo de SoC Wi-Fi BT 802.11b / g / n más pequeño
- CPU de 32 bits de baja potencia, también puede servir al procesador de aplicaciones
- Velocidad de reloj de hasta 160MHz, potencia de cálculo resumida de hasta 600 DMIPS
- Incorporado 520 KB SRAM, externo 4MPSRAM
- Admite UART / SPI / I2C / PWM / ADC / DAC
- Admite cámaras OV2640 y OV7670, lámpara de flash incorporada
- Imagen de soporte de carga de WiFI
- Soporte de tarjeta TF
- Admite múltiples modos de suspensión
- Embebido Lwip y FreeRTOS
- Admite el modo de operación STA / AP / STA + AP
- Admite la tecnología Smart Config / AirKiss
- Soporte para actualizaciones de firmware local y remoto de puerto serie (FOTA)

Pinout ESP32-CAM

Los GPIO 1 y GPIO 3 son los pines en serie, necesarios para cargar el código en la placa. El chip también cuenta con tres pines GND y dos pines de alimentación de 3.3V o 5V, lo que nos permite elegir la salida que queramos dar. Al alimentar el ESP32-CAM con 5V se obtiene una señal Wi-Fi más estable. El GPIO 0 también juega un papel importante, ya que al conectarse a GND, el ESP32 entra modo programable.

Los siguientes pines están conectados internamente al lector de tarjetas microSD (figura 12):

- GPIO 14: CLK
- GPIO 15: CMD
- GPIO 2: Fecha 0
- GPIO 4: Datos 1 (también conectado al LED incorporado)
- GPIO 12: Datos 2
- GPIO 13: Datos 3

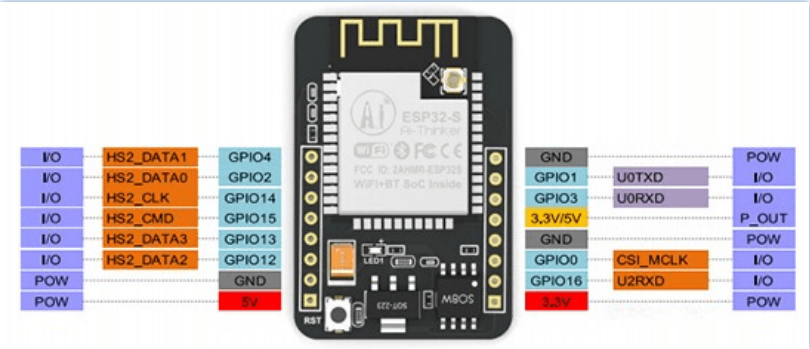


Figura 12.Pinout ESP32-CAM.

2.3.1.1 FT232RL

El ESP32-CAM no viene con un conector USB, por lo que necesita un programador FTDI para cargar el código a través de **U0R** y **U0T** pines (pines de serie). Para que se pueda subir correctamente el código a la ESP32-CAM es necesario que el GPIO 0 esté conectado a GND.

Tabla 2.Conexión entre los pines del ESP32-CAM y el FT232RL.

ESP32-CAM	FT232RL
GND	GND
5V	VCC (5V)
U0R	TX
U0T	RX
GPIO 0	GND

Diagrama esquemático

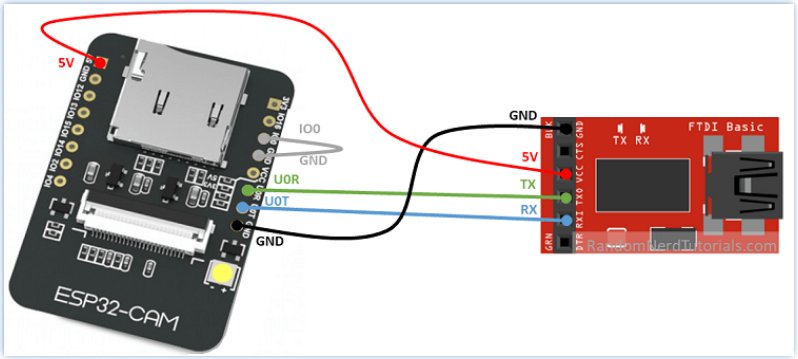


Figura 13.Conexión entre ESP32-CAM y el FT232RL.

2.3.2 DHT11

Para la lectura de la humedad y la temperatura utilizando el chip ESP32 se decidió utilizar el sensor DHT11 (***figura 14***) de la familia de los sensores DHT.

Figura 14. Sensor de humedad y temperatura Az-Delivery DHT11.

Estos sensores tienen como característica que realizan una lectura analógica de los datos, en este caso de temperatura y humedad. Esto los hace muy fáciles de usar con cualquier microcontrolador.

Principales características del DHT11:

- Rango de temperaturas: de 0 a 50 ºC +/- 2 ºC
- Rango de humedad: de 20 a 90% +/- 5%
- Resolución: Humedad: 1%, Temperatura: 1ºC
- Tensión de funcionamiento: de 3 a 5.5 V
- Suministro de corriente: de 0.5 a 2.5 mA
- Periodo de muestreo: 1 segundo
- Precio: entre $ 1 a $ 5

A la hora de realizar el proyecto se encontró otra solución utilizando el sensor DHT22, que ofrece una mejor resolución y un rango de medición de temperatura y humedad más amplio. Como inconveniente, el precio es más elevado que el sensor escogido y solo puede solicitar lecturas con un intervalo de 2 segundos. El DHT11, pese que tiene un rango más pequeño y es menos preciso, puede solicitar lecturas del sensor cada segundo y su precio es más económico.

A pesar de sus diferencias, ambos sensores funcionan de manera similar y pueden usar el mismo código para leer la temperatura y la humedad. Solo necesita seleccionar en el código el tipo de sensor que está utilizando.

Pinout del DHT

Los sensores DHT tienen cuatro pines como se muestra en la ***figura 15***, sin embargo, en el adquirido venía incorporado en una placa de conexión. Este viene solo con tres pines y con una resistencia pull-up interna en el pin 2.

Figura 15. Pines de los sensores DHT.

La siguiente tabla muestra el pinout DHT22 / DHT11. Cuando el sensor está mirando hacia usted, la numeración del pin comienza en 1 de izquierda a derecha.

Tabla 3. Pinout del sensor MQ-135

Pines	Descripción
1	3.3V
2	Cualquier GPIO digital; también conecte una resistencia pull-up de 10k Ohm
3	No conectar
4	GND

Diagrama esquemático

Conectamos el sensor DHT11 a la placa de desarrollo ESP32 como se muestra en la siguiente figura (figura 16)

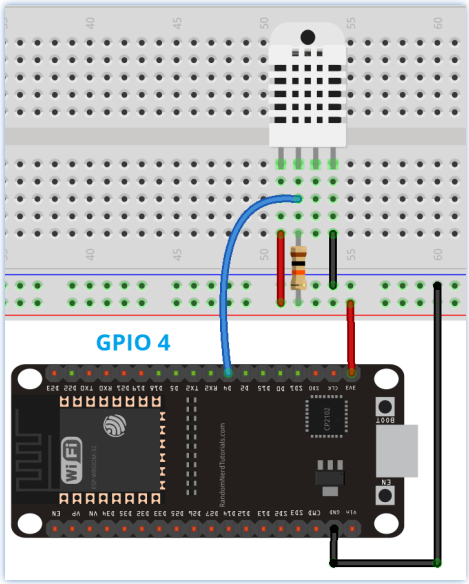


Figura 16. Conexión entre ESP32 y el sensor DHT11.

2.3.3 MQ-135

La calidad del aire se deteriora día a día y la contaminación por gases nocivos se ha convertido en un fenómeno común en todas partes. Esto se ve afectado especialmente en las zonas urbanas por el aumento inevitable de las industrias, la urbanización y el tráfico.

Algunos sensores como el MQ-*135* (**figura 17**), son una solución económica y fiable dentro del mundo del IoT. Permiten monitorizar la calidad del aire y son adecuados para detectar gases como CO2, humo, NH3, NOx, alcohol, benceno, etc.

Figura 17. Sensor MQ-135.

Su objetivo principal es desarrollar un dispositivo que pueda monitorear los parámetros contaminantes en el aire. Especialmente, la detección en tiempo real de grandes concentraciones de CO2 usando el sensor de gas MQ135 con el módulo WiFi ESP32 Posteriormente, la información se enviaría a la plataforma Node-RED.

Niveles de indicadores tomados para leer los valores de CO2 PPM:

- **Menos de 400 PPM:** concentración normal de fondo en aire ambiente exterior.
- **De 400 a 1000 PPM:** Típico en espacios interiores ocupados con buen intercambio de aire.
- **Más de 1000 PPM:** aire pobre.

Principales características del MQ-135

- El voltaje de funcionamiento es + 5V.
- Voltaje de salida: Analógico 0V a 5V o Digital 5V TTL Logic.
- Respuesta rápida y alta sensibilidad.
- Podemos usar el pin digital o el pin analógico para obtener una indicación de los PPM (partes por millón) de gas específico en el aire. La salida digital (5V TTL Logic) solo proporciona un estado bueno o malo al establecer un valor umbral utilizando el potenciómetro incorporado, mientras que la salida analógica proporciona las mediciones de datos sin procesar.

¿Cómo funciona el sensor?

Es importante destacar que para que el sensor haga las lecturas correctamente, es necesario precalentar durante 20 segundos aproximadamente.

La resistencia del sensor es diferente según el tipo de gas. El sensor de humo tiene un potenciómetro incorporado que le permite ajustar el umbral de salida digital del sensor (D0). Este umbral establece el valor por encima del cual el pin digital emitirá una señal ALTA.

El voltaje que genera el sensor es proporcional a la concentración de humo / gas que existe en la atmósfera. En otras palabras, cuanto mayor es la concentración de gas, mayor es el voltaje de salida.

Pinout del MQ-135

El sensor MQ-2 tiene 4 pines, como se muestra en la figura 18

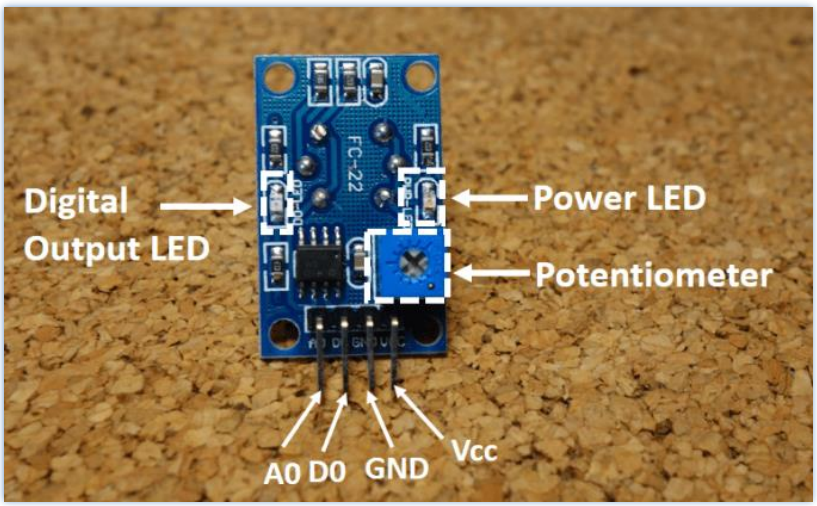


Figura 18.Entradas y salidas del MQ-135

Los pines de entrada y salida del sensor MQ-135 se detallan en la siguiente tabla:

Tabla 4.Pinout del sensor MQ-135

Pines	Descripción
A0	Analog Output
D0	Digital Output
GND	Ground
VCC	Alimentación de 3.3V a 12V

La salida puede ser tanto analógica (pin A0) como digital (pin D0), leyéndose estos valores por las respectivas entradas del chip ESP32.

2.3.4 KY-026

El sensor de llama óptico KY-026 (figura 19) es un dispositivo que permite detectar la existencia de combustión por la luz emitida por la misma. Esta luz es detectada por un sensor óptico y es capturado por las entradas digitales y analógicas del chip ESP32.

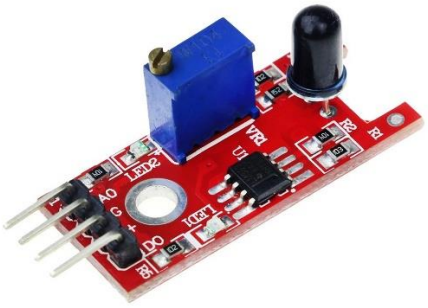


Figura 19. Sensor KY-026.

El espectro de emisión de llama depende de los elementos que intervienen en la reacción. En el mundo del IoT podemos encontrar sensores de llama baratos. Frecuentemente estos sensores constan únicamente de un sensor infrarrojo ajustado a 760-1100 nm. El ángulo de detección es de 60º, y la distancia de detección entre 0.40 m a 0.80.

Este tipo de sensores de llama infrarrojos suelen incorporar una placa de medición estándar con el comparador LM393, que permite obtener la lectura tanto como un valor analógico como de forma digital cuando se supera un cierto umbral, regulándose a través de un potenciómetro ubicado en la placa (figura 20)

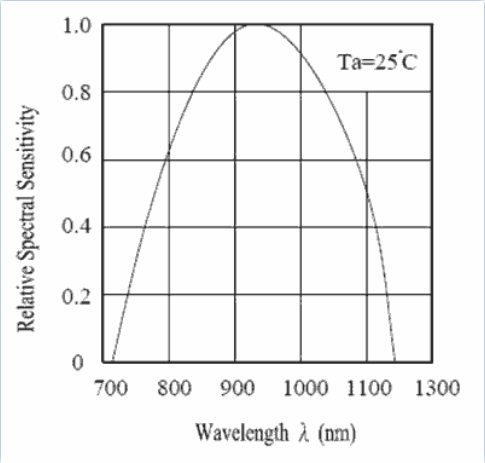


Figura 20. Sensibilidad espectral relativa vs Longitud de onda del sensor KY-026.

El espectro de emisión de llama depende de los elementos que intervienen en la reacción. En el sector industrial, la combustión de productos con carbón en presencia del oxígeno tenemos dos picos característicos: en ultravioleta las longitudes de onda oscilan entre 185nm-260nm y en infrarrojo en longitudes de onda 4400-4600nm (figura 21)

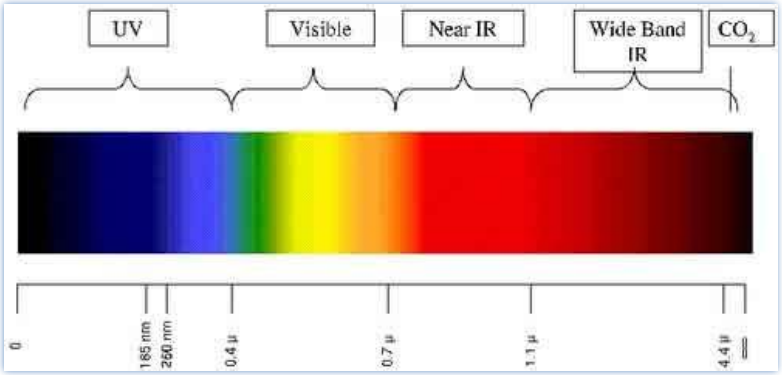


Figura 21. Espectro electromagnético.

Como vemos, las longitudes de onda de estos sensores baratos poco tienen que ver con las emisiones características de las llamas y de los sensores industriales. De hecho, son afectados incluso por la iluminación interior, dando lugar a numerosos falsos positivos.

Por tanto, la sensibilidad y fiabilidad de estos detectores baratos no son suficientes para considerarlos un auténtico dispositivo de seguridad, aunque pueden ser interesantes en pequeños proyectos de domótica como este.

Principales características del KY-026

- Espectro: 760nm ~ 1100nm
- Ángulo de detección: 0 – 60 grados
- Alimentación: 3.3V ~ 5.3V
- Temperatura: -25℃ ~ 85℃
- Dimensión: 27.3mm * 15.4mm
- Salida analógica.
- Salida digital.

Pinout del KY-026

El pinout de este sensor es muy simple. Cuenta con cuatro terminales, los cuales están serigrafiados en la placa, indicados en la siguiente tabla.

Tabla 5. Pinout del sensor KY-026.

Pines	Descripción
A0	Analog Output
D0	Digital Output
GND	Ground
VCC	Alimentación de 3.3V a 12V

2.3.5 Sensor capacitivo de humedad del suelo V1.2

Este sensor (figura 22), mide los niveles de humedad del suelo mediante detección capacitiva en lugar de detección resistiva como otros sensores en el mercado. No es tan preciso como el sensor resistivo, pero es significativamente menos costoso y no usa ningún puerto GPIO. Como no está exponiendo los electrodos a la humedad del suelo, la corrosión no es un problema y le da una excelente vida útil. Este módulo incluye un regulador de voltaje a bordo que le da un rango de voltaje operativo de 3.3 ~ 5.5V y es perfecto para MCU de bajo voltaje.

Figura 22. Sensor capacitivo de humedad.

Principales características del sensor de humedad del suelo:

- Voltaje de funcionamiento: 3.3 ~ 5.5 VDC
- Voltaje de salida: 0 ~ 3.0VDC
- Corriente de funcionamiento: 5 mA
- Interfaz: PH2.0-3P
- Dimensiones: 99x16mm / 3.9x0. 63 ″
- Peso: 15g
- Admite interfaz de sensor de 3 pines
- Salida analógica

¿Cómo funciona un sensor capacitivo de humedad?

Un sensor de humedad capacitivo funciona midiendo los cambios en la capacitancia causados por los cambios en el dieléctrico. No mide la humedad directamente (el agua pura no conduce bien la electricidad), mide los iones que se disuelven en la humedad. Estos iones y su concentración pueden verse afectados por una serie de factores. Por ejemplo, agregar fertilizante disminuirá resistencia del suelo. La medición capacitiva básicamente mide el dieléctrico formado por el suelo y el agua es el factor más importante que afecta al dieléctrico.

La medición capacitiva tiene algunas ventajas. No solo evita la corrosión de la sonda, sino que también ofrece una mejor lectura del contenido de humedad del suelo en lugar de utilizar un sensor de humedad resistivo. Dado que los contactos (la placa positiva y la placa negativa del condensador) no están expuestos al suelo, no hay corrosión del sensor en sí.

La capacitancia del sensor se mide por medio de un oscilador de frecuencia fija construido con un **555 Timer IC**, que produce un voltaje proporcional al capacitor insertado en el suelo.

La medición de este voltaje es mediante el uso de un convertidor analógico a digital que produce un número que luego podemos interpretar como humedad del suelo.

Rango de calibración del sensor:

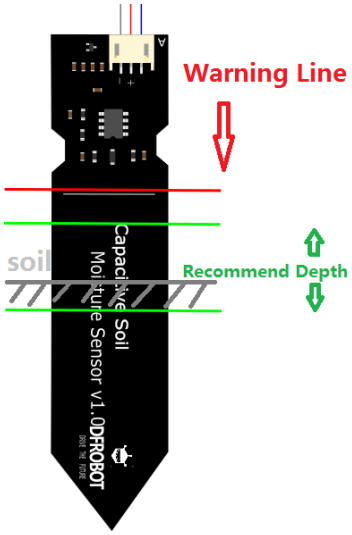


Figura 23. Sensor de humedad del suelo.

Para calibrar el sensor es necesario insertarlo correctamente, como se indica en la figura 23, y realizar dos medidas previas. La primera es el valor del sensor cuando la sonda esté expuesta al aire como "Valor 1 (es el valor límite del suelo seco "Humedad: 0% HR") y la segunda medida la tomamos introduciendo la sonda en una taza de agua no más allá de la línea roja en el diagrama. De esta forma registramos el valor del sensor cuando la sonda esté expuesta al agua como "Valor 2"(es el valor límite del suelo húmedo "Humedad: 100% HR")

El valor de salida final se ve afectado por la profundidad de inserción de la sonda y de la humedad de la tierra. Consideramos "value_1" como tierra seca y "value_2" como tierra empapada. Este es el rango de detección del sensor. Por ejemplo: Valor_1 = 520; Valor_2 = 260. El rango se dividirá en tres secciones: seco, húmedo, agua. Sus valores relacionados son:

- Seco: (520 - 430]
- Húmedo: (430 - 350]
- Agua: (350 - 260]

Pinout del sensor

Tabla 6. Pinout del sensor capacitivo de humedad del suelo.

Pines	Descripción
A0	Analog Output
GND	Ground
VCC	Alimentación de 3.3V a 12V

2.3.6 Módulo wifi relay

Un relé es un interruptor operado eléctricamente y, como cualquier otro interruptor, puede encenderse o apagarse, dejando pasar la corriente o no. Hay diferentes módulos de relé con un número diferente de canales como se muestra en la figura 24.

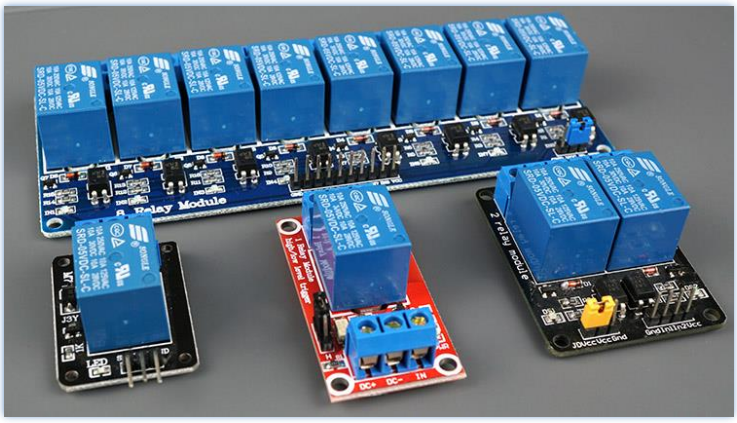


Figura 24. Módulos relay de diferentes canales.

Principales características del módulo wifi relay:

- Voltaje de funcionamiento: 3.3 ~ 5.5 VDC
- Permite controlar altos voltajes como: 12V, 24V o voltajes de red (230V en Europa y 120V en los Estados Unidos)
- Pueden presentar diferentes números de canales: uno, dos, cuatro, ocho e incluso dieciséis. El número de canales determina el número de salidas que podremos controlar.
- Pueden tener optoacoplador incorporado: este dispositivo agrega una "capa" adicional de protección, aislando ópticamente el ESP32 del circuito de relé.

Pinout del relé

Para fines explicativos analizaremos el pinout de un módulo de relé de 2 canales. Si se usa un módulo de relé con un número diferente de canales el pinout es similar.

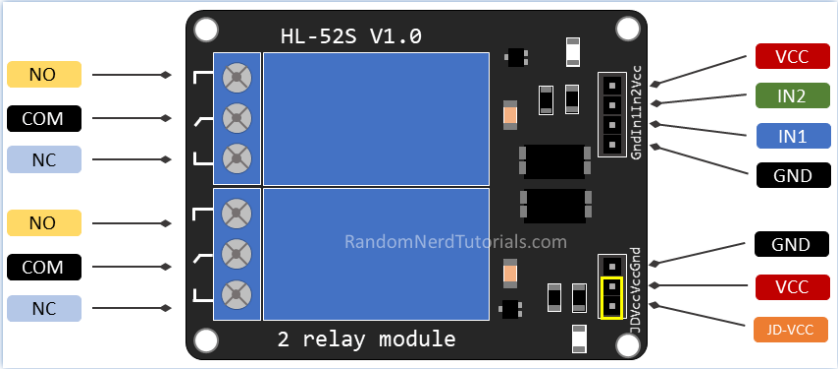


Figura 25. Pinout de Módulo Relay de 2 canales.

Como se ve en la figura 25, en el lado izquierdo, hay dos conjuntos de tres enchufes para conectar altos voltajes, y los pines en el lado derecho (bajo voltaje) se conectan a los GPIO ESP32.

El módulo de relé que se muestra en la foto anterior tiene dos conectores, cada uno con tres enchufes: común (**COM**), Normalmente cerrado (**NC**), y normalmente abierto (**NO**).

- **COM:** conecte la corriente que desea controlar (tensión de red).
- **NC (normalmente cerrado):** la configuración normalmente cerrada se utiliza cuando desea que el relé se cierre por defecto. Los NC son los pines COM conectados, lo que significa que la corriente fluye a menos que envíe una señal desde el ESP32 al módulo de relé para abrir el circuito y detener el flujo de corriente.
- **NO (normalmente abierto):** la configuración normalmente abierta funciona al revés: no hay conexión entre los pines **NO** y **COM**, por lo que el circuito está roto a menos que envíe una señal desde el ESP32 para cerrar el circuito.

El lado de bajo voltaje del relay donde se conecta el ESP32 tiene dos conjuntos de pines, uno de cuatro y otro de tres. El primer conjunto consta de **VCC** y **GND** para encender el módulo y la entrada 1 (**EN 1**) y entrada 2 (**EN 2**) para controlar los relés inferior y superior, respectivamente.

Si el módulo de relé solo cuenta con un canal, solo tendrá un pin **IN**. Si tiene cuatro canales, tendrá cuatro pines **IN**, y así sucesivamente. La señal que envía a los pines **IN** determina si el relé está activo o no. El relé se activa cuando la entrada desciende por debajo de aproximadamente 2V.

Esto significa que tendrá los siguientes escenarios:

Configuración normalmente cerrada (NC):

- Señal ALTA: la corriente fluye
- Señal BAJA: la corriente no fluye

Configuración normalmente abierta (NO):

- Señal ALTA: la corriente no fluye
- Señal BAJA: corriente en flujo

La configuración normalmente cerrada **NC**, se debe usar cuando la corriente fluye y solo deseamos detenerla ocasionalmente. Por otra parte, se usa una configuración normalmente abierta **NO** cuando solo se desea poner la corriente en funcionamiento en determinadas ocasiones.

El segundo conjunto de pines consiste en: **GND**, **VCC** y **JD-VCC**.

Los pines **JD-VCC** alimentan el electroimán del relé. Observe que el módulo tiene una tapa de puente que conecta los pines **VCC** y **JD-VCC**; en la imagen anterior (figura 25) el que se muestra es amarillo, aunque puede variar en dependencia del fabricante.

- Con la tapa del puente puesta, el **VCC** y los pines **JD-VCC** están conectados. Eso significa que el electroimán de relé se alimenta directamente del pin de alimentación ESP32, por lo que el módulo de relé y los circuitos ESP32 no están físicamente aislados entre sí.
- Sin la tapa del puente, debe proporcionar una fuente de alimentación independiente para encender el electroimán del relé a través del pin **JD-VCC**. Esa configuración aísla físicamente los relés del ESP32 con el optoacoplador incorporado del módulo, lo que evita daños al ESP32 en caso de picos eléctricos.

2.4 Herramienta Node-RED

Node-RED es una poderosa herramienta de código abierto desarrollado por IBM para crear aplicaciones de Internet de las cosas (IoT), con el objetivo de simplificar el componente de programación. Utiliza una programación visual que le permite conectar bloques de código conocidos como nodos, para realizar una tarea. Los nodos cuando están conectados entre sí se denominan flujos.

Ventajas de utilizar Node-RED:

- Permite acceder a las GPIO de diversas placas como: Raspberry Pi, ESP8266, ESP32, Arduino.
- Establece conexiones MQTT con otras placas (Arduino, ESP8266, etc.)
- Crea una interfaz gráfica para los proyectos IoT de una manera muy sencilla.
- Comunica con servicios de terceros (IFTTT.com, Adafruit.io, Thing Speak, etc.)
- Captura datos de webs (pronóstico del tiempo, precios de acciones, correos electrónicos, etc.)
- Crear eventos activados por tiempo.
- Almacena y recupera datos de una base de datos.

Secciones principales del programa:

Como se ve en la figura 27, en el lado izquierdo, hay una amplia lista de bloques llamados nodos que están separados por su funcionalidad. Si selecciona un nodo, se puede saber cómo funciona en la pestaña de información. En el centro se encuentra el flujo que es donde se colocan los nodos.

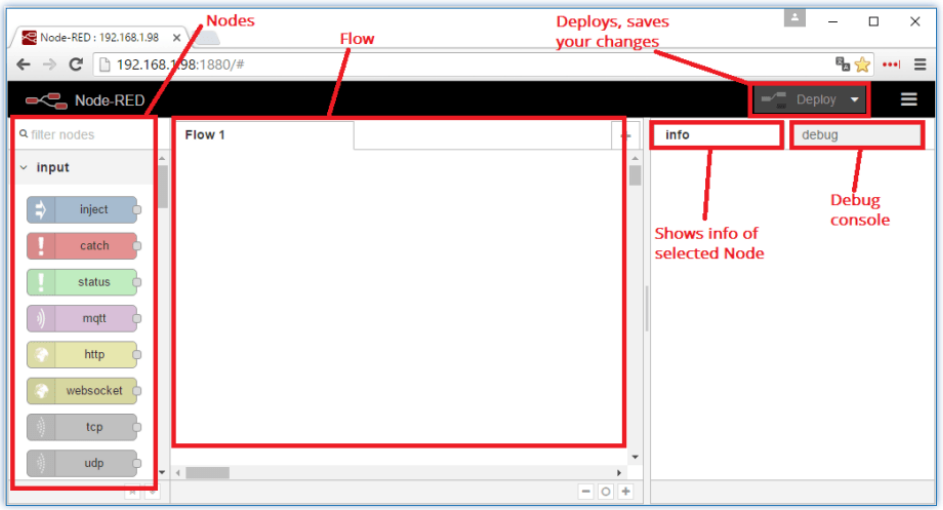


Figura 26. Principales secciones de Node-RED.

Para crear una interfaz de usuario en Node-RED se utilizan los nodos de tipo **Dashboard** que proporcionan widgets mostrándose en la interfaz de usuario IU de la aplicación. La interfaz de usuario está organizada en pestañas y grupos. Dentro de cada pestaña existen grupos que dividen las pestañas en diferentes secciones, para poder organizar los widgets.

Node-RED ofrece una amplia gama de opciones que nos permiten modificar el estilo y el formato de nuestra aplicación web, posibilitando ajustar la herramienta a nuestra medida.

2.5 IDE Arduino

Para la programación del chip ESP32 se decidió utilizar el conocido IDE (***entorno de desarrollo integrado***) de Arduino. Esta es una plataforma de código abierto que ha ganado gran popularidad por las facilidades de uso que brinda a los usuarios.

Arduino surgió en el Ivrea Interaction Design Institute como una herramienta fácil para la creación rápida de prototipos. Está basado en el lenguaje de programación de Java y se caracteriza por ser una aplicación multiplataforma, posibilitando su instalación tanto en Windows, como en macOS o Linux. El código fuente para el IDE de Arduino se publica bajo la Licencia Pública General de GNU. Admite los lenguajes C y C ++ utilizando reglas especiales de estructuración de códigos y suministra una biblioteca de software del proyecto Wiring.

Secciones del IDE de Arduino

El entorno de desarrollo integrado de Arduino contiene un editor de texto para escribir código, un área de mensajes, una consola de texto, una barra de herramientas con botones para funciones comunes y una serie de menús como se detalla en la ***Figura 28***. Se conecta al hardware Arduino y Genuino para cargar programas y comunicarse con ellos.

Figura 27. Secciones principales del IDE

Los programas escritos con el software Arduino (IDE) se denominan bocetos. Estos bocetos se escriben en el editor de texto y se guardan con la extensión de archivo ***.ino*** **(**generará en última instancia un archivo hexadecimal que luego se transfiere y se carga en el controlador de la placa**)**. El editor tiene funciones para cortar / pegar y para buscar / reemplazar texto. El área de mensajes proporciona información al guardar y exportar y también muestra errores. La consola (***Figura 48)*** muestra la salida de texto del software Arduino (IDE), incluidos los mensajes de error completos y otra información .

La esquina inferior derecha de la ventana muestra la placa configurada y el puerto serie. Los botones de la barra de herramientas le permiten verificar y cargar programas, crear, abrir y guardar bocetos y abrir el monitor en serie (figura 29)

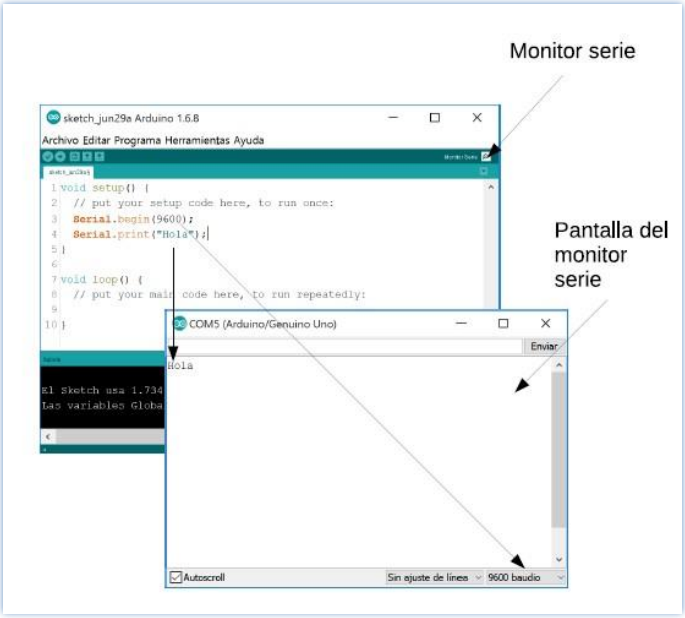


Figura 28. Pantalla del monitor serie.

La opción de “Herramientas” (figura 30) se utiliza principalmente para configurar proyectos de prueba. La sección del “Programador” de este panel se utiliza para grabar un cargador de arranque en el nuevo microcontrolador. Es de suma importancia seleccionar correctamente el modelo de la placa y el puerto COM serie en el IDE. Se puede buscar el dispositivo serie USB en la sección de puertos del Administrador de dispositivos de Windows.

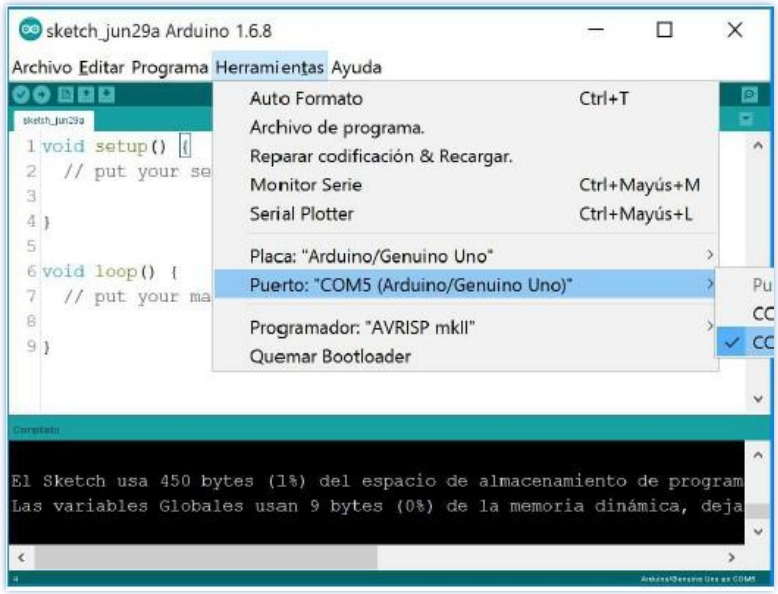


Figura 29. Opción herramientas.

Las bibliotecas son muy útiles para agregar la funcionalidad adicional al módulo Arduino. Hay una lista de bibliotecas que se pueden agregar haciendo clic en el botón Sketch en la barra de menú y yendo a “Incluir biblioteca”(Figura 31). Al hacer clic en” Incluir biblioteca “y agregar la biblioteca respectiva, aparecerá en la parte superior del boceto con un signo ***#incluir***. La mayoría de las bibliotecas están preinstaladas y vienen con el software Arduino. Sin embargo, también se pueden descargarlos de fuentes externas.

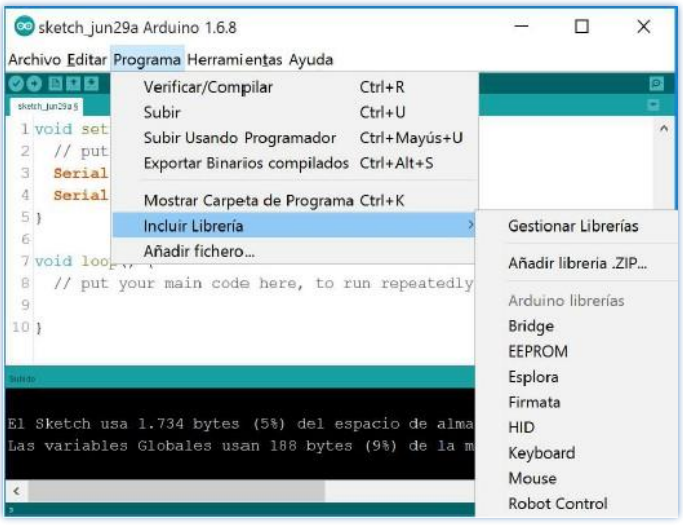


Figura 30. Opción para incluir librerías.

Programación en el IDE de Arduino

La estructura del software consta de dos funciones principales Figura 32:

Función de configuración (): La función ***setup ()*** se llama cuando se inicia un boceto. Se usa para inicializar las variables, modos de pin, comenzar a usar bibliotecas, etc. La función de configuración solo se ejecutará una vez, después de cada encendido o reinicio de la placa Arduino. Aquí, se puede definir la velocidad de salida en el ***Serial.begin*** () (declaración que abre el puerto serie para permitir que la placa envíe salida para que la muestre el monitor serie).

Función loop (): Después de llamar a la función setup () , que inicializa y establece los valores iniciales, la función loop () hace precisamente lo que sugiere su nombre y se repite consecutivamente, lo que permite que su programa cambie y responda. Se utiliza para controlar activamente la placa Arduino.

```
// la rutina de configuración se ejecuta una vez cuando presiona reiniciar:
void  setup ()
{
  // inicializar la comunicación serial a 9600 bits por segundo:
Serial.begin ( 9600 );
}

// la rutina de bucle se repite una y otra vez para siempre:
void  loop ()
{
  // lee la entrada en el pin analógico 0:
  int sensorValue = analogRead (A0);
  // imprime el valor leído:
  Serial.println (sensorValue);
  retraso (1);   // retraso entre lecturas para estabilidad
}
```

Figura 31. Estructura del código en Arduino

Capítulo 3. Desarrollo

3.1 Google Cloud Platform

Google Cloud surge como una nueva solución de Google en la cual reúne en una sola plataforma todas las aplicaciones de desarrollo web que anteriormente ofrecía de manera separada.

Una de las ventajas más destacables de esta tecnología es que no hay que preocuparse por tener una infraestructura propia, sino que nos brinda todas las condiciones para que nos centremos en la creación de nuestro producto final. La rapidez, seguridad y la escalabilidad de su infraestructura constituye una alternativa a tareas de acceso, almacenamiento y gestión de datos que antes requerían de la utilización de hardware o software.

Productos de Google Cloud.

Los productos de Google Cloud Platform [9] están categorizados según sus diferentes aplicaciones.

Entre ellas se encuentran (figura 33):

- **Computación:** Esta solución destaca por el potencial para realizar cálculos matemáticos a gran velocidad, por ejecutar grandes bases de datos en memoria y por crear aplicaciones nativas de la nube.
- **Cloud Storage:** ofrece soluciones sencillas, seguras y fiables para tratar los datos multimedia, analíticos y de aplicaciones. Permite realizar funciones avanzadas integradas como redundancia geográfica, almacenamiento en caché perimetral, controlar la recuperación tras fallos y funciones de inteligencia artificial que permiten sacar el máximo partido al almacenamiento de los datos.
- **Bases de datos:** pone a nuestra disposición bases de datos seguras, fiables y de alta disponibilidad para manejar los datos. Se caracterizan por ser escalables y por tener amplia compatibilidad con código abierto.
- **Red:** pone a disposición de los usuarios una amplia red de fibra con alcance internacional caracterizada por utilizar una seguridad de red de defensa reforzada y por contar con soluciones basadas en acuerdos de nivel de servicios líderes en el sector.
- **Operaciones:** permite monitorizar y mejorar el rendimiento de las aplicaciones en el entorno de Google Cloud y solucionar el surgimiento de futuros problemas. Esto era lo que antes se conocía como Stackdriver.
- **Herramientas de desarrollo:** permite crear software y aplicaciones nativas de la nube en múltiples lenguajes de manera sencilla y rápida. Además, estandariza la integración, crea flujos de procesamiento y automatiza los despliegues, gracias a las herramientas y a las bibliotecas pensadas para mejorar la productividad de los desarrolladores.
- **Analíticas de datos:** ofrece información valiosa en tiempo real que ayuda a los clientes a tomar decisiones más acertadas que contribuyan a fomentar la innovación de sus proyectos.
- **Inteligencia artificial y aprendizaje automático:** brinda productos y servicios de aprendizaje automático innovadores.
- **Gestión de APIs:** ofrece una plataforma de gestión, desarrollo y seguridad de APIs.
- **Entornos híbridos y multinube:** ***"Anthos"*** plataforma que permite modernizar las aplicaciones ejecutadas en máquinas virtuales y desplegar aplicaciones nativas de la nube en contenedores. Esto ofrece una experiencia de desarrollo y operaciones uniforme en todos tus despliegues.
- **Migración de centros de datos:** Migra aplicaciones y cargas de trabajo sin tener que reescribirlas ni modificarlas.

- **Seguridad e identidad:** Permite proteger los datos, identidades, aplicaciones y dispositivos con la infraestructura segura desde el diseño, la protección integrada y la red mundial usada por Google.
- **Computación sin servidor:** la plataforma permite escribir código sin tener que preocuparse por la infraestructura subyacente, es decir, se puede crear aplicaciones completas sin servidor gracias al almacenamiento, las bases de datos, el aprendizaje automático y otros recursos de Google Cloud.
- **Contenedores:** La creación de contenedores permite a nuestros equipos de desarrollo moverse con rapidez, desplegar software con eficacia y funcionar a una escala sin precedentes.
- **IoT:** Servicio de conexión, integración y gestión de dispositivos del Internet de las cosas.
- **Herramientas de gestión:** esta herramienta permite desarrollar, desplegar, gestionar las aplicaciones en la nube y optimizar la gestión de estas.
- **Sanidad y ciencias biológicas:** Solución para vincular sistemas y aplicaciones de atención sanitaria, biomédicos y servicios digitales.

Figura 32. Productos de Google Cloud Platform.

Para la realización de este trabajo se utilizó el servicio de ***Compute Engine*** para la creación de una máquina virtual en la cual se instalará el broker MQTT y la aplicación Node-RED. Esta solución ofrece una infraestructura de computación con distintos tamaños de máquinas predefinidos o personalizados para agilizar la transformación de la nube.

3.2 Creación y configuración de máquina virtual en Google Cloud Platform.

Para poder trabajar en la plataforma de Google Cloud es necesario disponer de una cuenta de correo de Google con la cual registrarse. Una vez se accede, da la opción de poder trabajar con una prueba gratuita en la que se dispone de un crédito de 300 dólares para explorar todos los servicios de la herramienta.

Haciendo uso de esta ventaja se procederá a explicar los pasos seguidos para crear la instancia de la máquina virtual utilizada [10]. :

1-Se accede desde el menú principal a la opción **Compute Engine**, donde se selecciona la opción de **Instancias de VM** y se crea una nueva instancia.

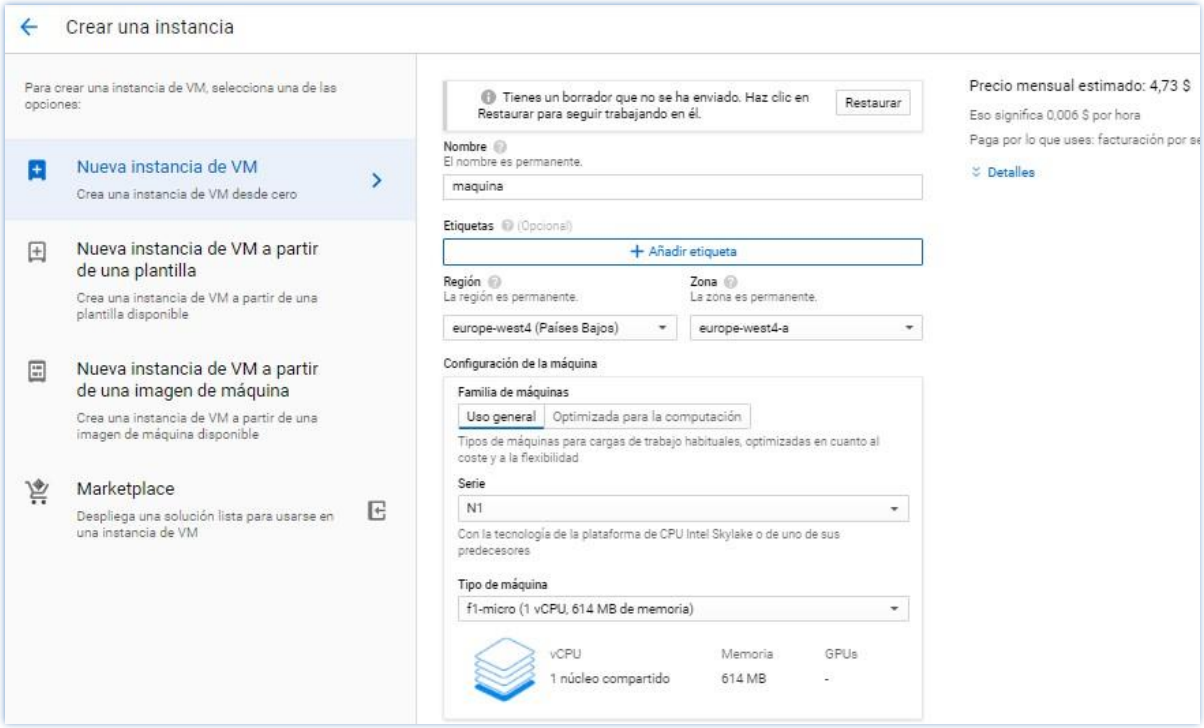


Figura 33. Creación de una instancia de una VM.

Para que se cree correctamente la instancia (figura 34) será necesario definir las características de estas como: el nombre de la instancia (***ej: mqqtbroker***), seleccionar correctamente la región donde se desea crear la instancia (dependerá del proyecto en cuestión, pero se sugiere que se cree lo más cercano al lugar donde se vaya a utilizar el broker, para evitar grandes latencias ***ej: europe-west4 (países Bajos)***).

En la **"Configuración de la máquina"** se elegirá el tipo de máquina a utilizar, variando el precio según la opción elegida. Con la idea de sacarle el mayor partido al crédito promocional, se eligió una máquina que cumpliera con los requisitos del trabajo y que a su vez, no tuviera un alto coste (***ej: fi-micro [1 vCPU, 614 MB de memoria***] con un precio mensual estimado de 4,73 $.

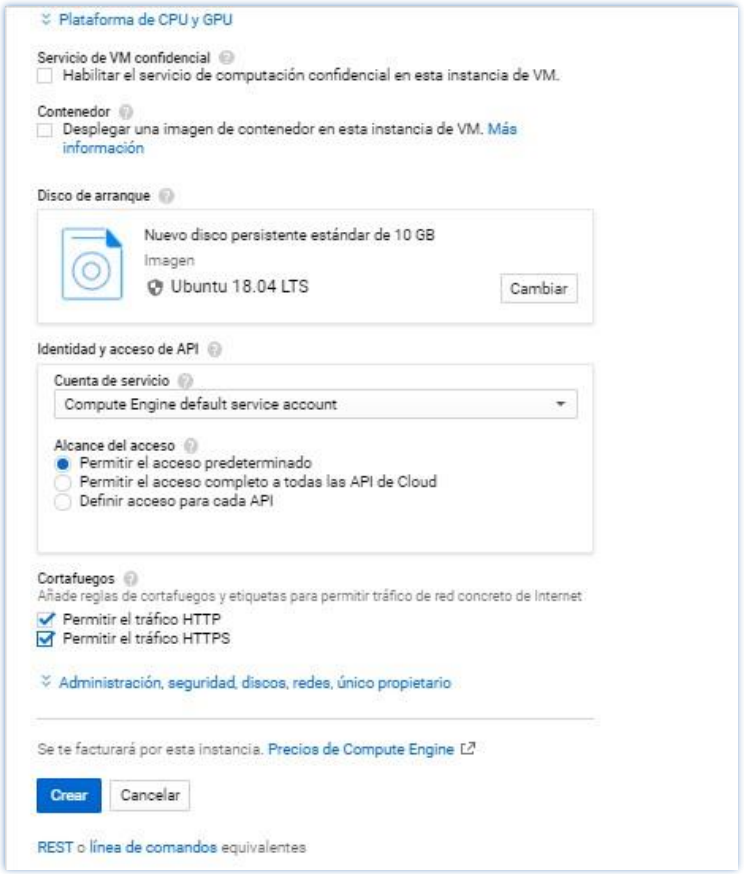


Figura 34. Continuación de la creación de la instancia de VM.

A la hora de seleccionar el disco de arranque se seleccionó el Ubuntu 18.04 LTS que ofrece una gran cantidad de herramientas (figura 35). En el tipo de disco de arranque se dejó la opción de "***Disco persistente estándar"*** con un tamaño de 10GB.

Las opciones de *"**Identidad y acceso de API"*** y de ***"Alcance del acceso"*** se dejaron con sus opciones por defecto: *"**Compute Engine default service account"*** y *"**Permitir el acceso predeterminado"*** respectivamente. En "***Firewall"*** permitirá el tráfico HTTP y HTTPS. Al hacer clic en la opción de ***Crear***, se muestra un mensaje avisando que se usará el crédito de la prueba gratuita para crear la instancia.

2- Una vez creada la máquina virtual, es necesario definir una dirección IP externa fija desde la cual esté accesible (figura 36). Para esto, accediendo al menú de ***"Red de VPC"*** en la opción "***Direcciones IP externas",*** debe cambiarse el tipo de ***"Efímera"*** a ***"Estática"*** y se debe dar un nombre (***ej: mqttbroker***).

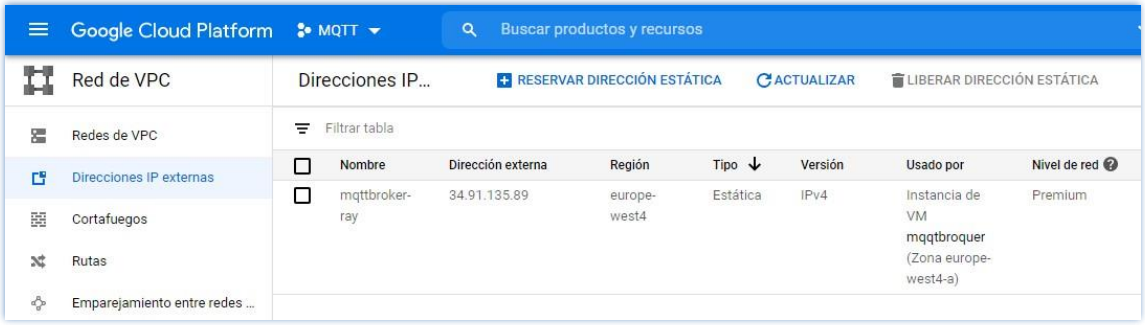


Figura 35.Creación de la IP externa fija para la máquina virtual.

Google Cloud cuenta con un firewall externo independiente a la máquina virtual creada, encargado de bloquear todas las conexiones externas que quieran ocurrir. Para evitar este bloqueo, se debe realizar unas configuraciones para permitir la comunicación, explicado en el siguiente punto.

3-Para permitir las conexiones externas va a ser necesario abrir algunos puertos en este firewall externo.

Accediendo desde el último menú ***"Red de VPC",*** en la opción "***Reglas de firewall"*** será necesario crear nuevas reglas para poder conectar. La primera regla para crear va a ser la llamada ***"dashboard"***(figura 37) . Se dejan las opciones de ***"Registros", "Red"*** y ***"Prioridad"*** con los valores por defecto ***"Desactivado"***, ***"default"*** y ***"1000"*** como valor de prioridad respectivamente.

Es importante destacar que en las opciones de ***"Dirección del tráfico"*** debe tener el valor ***"Entrada"*** y en ***"Acción en caso de coincidencia"*** permitir la acción a realizar. En la opción de ***"Destinos"*** se debe elegir la opción ***"Todas las instancias de la red"***, teniendo presente que en caso de tener otro **VPC** va a funcionar con las mismas reglas si está en la misma red. En la opción ***"Filtro de fuente"*** se debe elegir el valor ***"Rangos de IP"***, completándose con la opción ***"Intervalos de IP de origen"*** a la cual se le dará el valor ***"0.0.0.0/0"*** para que no filtre por una IP en particular, si no, que vamos a permitir que se conecten desde cualquier IP a este puerto en particular.

La opción ***"Segundo filtro de fuente"*** se deja con la opción por defecto ***"No hay recursos para mostrar"*** y en la opción ***"Protocolos y puertos"*** se debe restringir el acceso sólo al puerto deseado. Hay que habilitar la opción ***"Puertos y protocolos especificados"*** y marcar el check de la opción ***"tcp"*** y definir los puertos desde los cuales sen va a permitir conexiones TCP. Se habilitarán los puertos ***"18083"*** para acceder al Dashboard EMQX, el puerto ***"8083"*** para habilitar el websocket y el puerto ***"8083"*** para habilitar conexiones TCP. El resto de las opciones en este apartado no se seleccionan. Finalmente se da clic en la opción ***"Crear"*** para establecer la nueva regla.

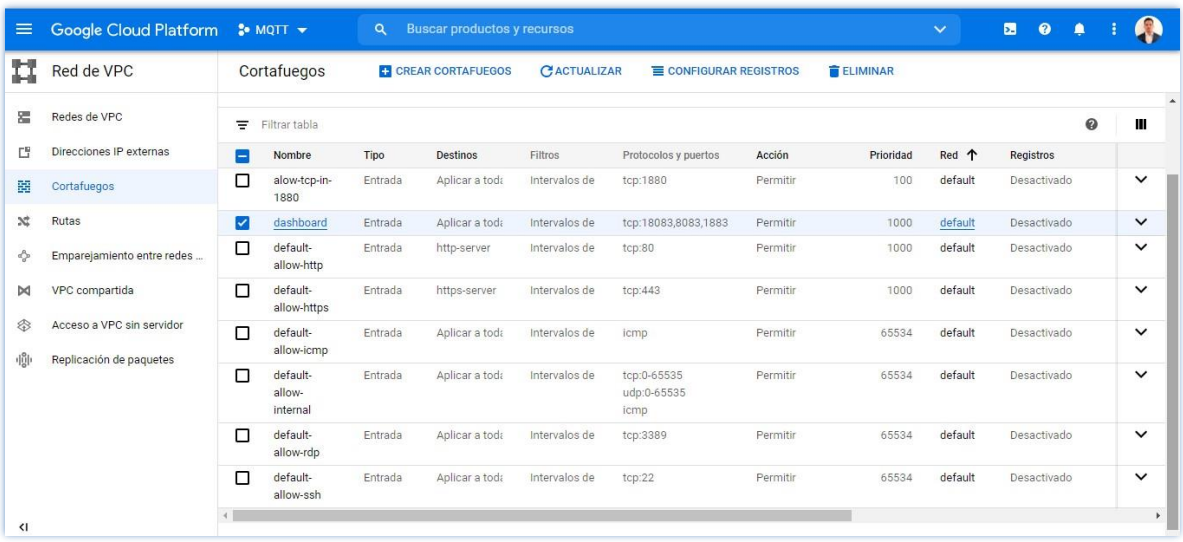


Figura 36. Creación de la regla "dashboard" para permitir el acceso al panel de control de EMQ.

Una gran ventaja con la que cuentan estás instancias de Google Cloud, es que no necesitan de un programa externo (ej: **PuTTY**) para acceder al terminal del servidor creado. Cuenta con la opción **SSH,** que permite con un simple clic acceder al terminal de la máquina virtual.

4- Para instalar el Dashboard EMQ en la máquina virtual creada, es necesario acceder a la consola SSH de la máquina virtual. (figura 38)

Una vez en la consola, se accede con los permisos de administrador con el comando **"sudo su"** para poder hacer las instalaciones pertinentes en el servidor. El siguiente paso es ir hasta la carpeta raíz del servidor (`root@mqqtbroquer:/`) con el comando **"cd .."**. Una vez en la ubicación deseada, se descargará en el servidor un zip que contendrá el instalador de EMQ. Poniendo el comando **"wget"** seguido del enlace de descarga de la versión deseada (ej: `https://www.emqx.io/downloads/broker/v3.0.1/emqx-ubuntu18.04-v3.0.1.zip`) tendremos el archivo deseado.

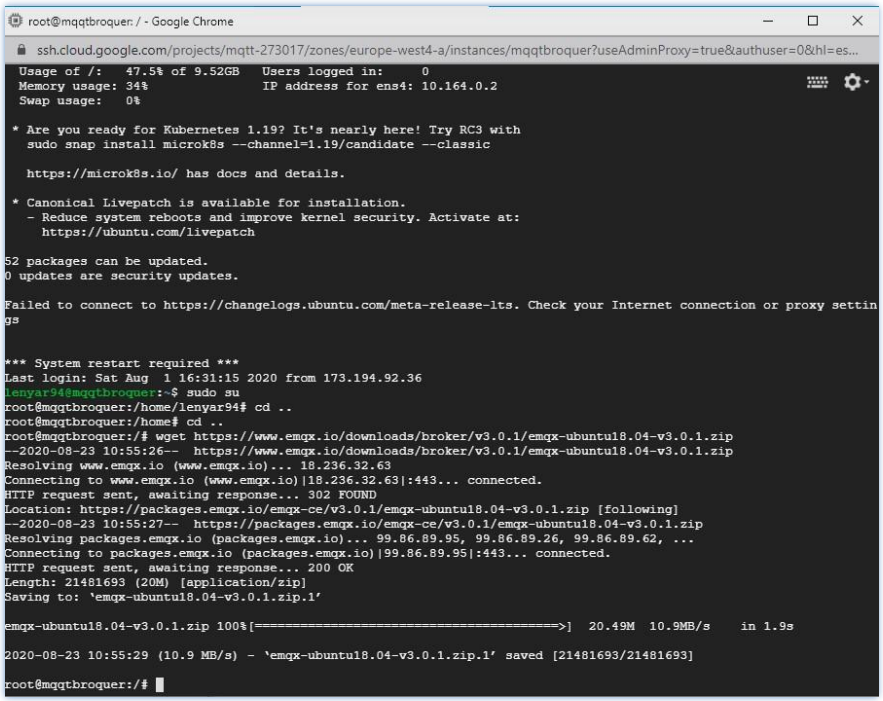


Figura 37. Terminal SSH de la máquina virtual.

Si listamos los archivos que tenemos en el servidor con el comando **"ls"**, se verá que está la carpeta zip con el instalador. Será necesario descomprimir la carpeta para poder ejecutar el instalador. Para esto se debe instalar primero el comando **unzip** con el comando **"apt install unzip"**. Una vez instalado unzip, con el comando unzip seguido de la carpeta a descomprimir se podrá extraer los archivos e instalarlo (ej: "**"unzip emqx-ubuntu18.04-v3.0.1.zip"**).

Al terminar todo el proceso anterior, se crea una carpeta emqx donde estará el proyecto. Si se accede a ella con el comando **"cd emqx"** y se lista los archivos, se verá que hay algunas carpetas propias del programa.

5-En este punto, antes de poner en marcha el dashboard, será necesario abrir los puertos en el firewall interno que presenta la máquina en Ubuntu [11]. Esta cuenta con una doble capa de protección, un firewall externo y uno interno, que protege a la máquina virtual de ataques de hacker.

Para abrir el puerto, con el comando **"ufw allow from any to any port 18083 proto tcp"** se permite, a través del puerto 18083,acceder al dashboard. Análogamente se debe permitir el acceso a los puertos 1883 y al 8083.

Por último, se encendará el broker con los comandos **"./bin/emqx start"** y se detendrá con el comando **"./bin/emqx stop"** (figura 39)

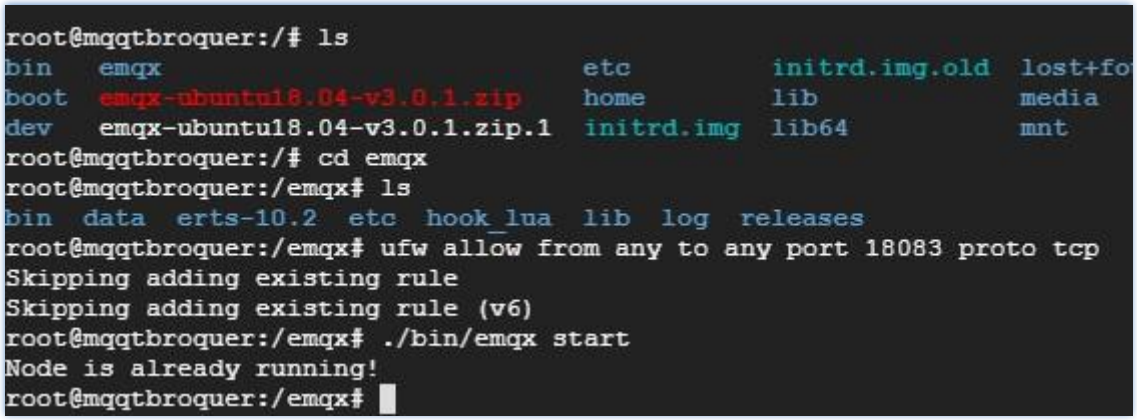


Figura 38. Configuración en el firewall interno de la VM de Ubuntu y arranque del broker.

3.3 EMQX Dashboard.

EMQ X (*Erlang / Enterprise / Elastic MQTT Broker*) es un broker de mensajes MQTT de código abierto basado en la plataforma Erlang/OTP. Erlang/OTP es una excelente plataforma de desarrollo distribuida, de baja latencia y en tiempo real.

EMQ X está diseñado para el acceso masivo de clientes y realiza un enrutamiento de mensajes rápido y de baja latencia entre dispositivos de red físicos masivos:

- Estable para alojar conexiones de cliente MQTT a gran escala y un nodo de servidor único que admite millones de conexiones.
- Clúster distribuido, enrutamiento de mensajes rápido y de baja latencia. Un solo clúster admite decenas de miles de rutas.
- Extensible, admite complementos personalizados, como autenticación y otras funciones.
- Compatibilidad completa con el protocolo IoT, incluidos MQTT, MQTT-SN, CoAP, LwM2M y otros protocolos patentados basados en TCP / UDP.

EMQ X Dashboard [12] es una aplicación web a la que se puede acceder directamente a través del navegador sin instalar ningún otro software. EMQ X Broker proporciona un panel para facilitar a los usuarios la gestión de equipos y el seguimiento de los indicadores relacionados. A través de Dashboard, se puede ver la información básica del servidor, cargar y ver datos estadísticos, ver el estado de conexión de un cliente e incluso desconectarlo. Además, proporciona una interfaz de operación visual del motor de reglas y también integra una herramienta de cliente MQTT simple para pruebas de usuario. EMQ X Dashboard admite oyentes HTTP y HTTPS, pero solo los oyentes HTTP con un puerto de escucha de 18083 están habilitados de forma predeterminada.

La página de visualización predeterminada de Dashboard **Overview** (figura 40) proporciona información detallada de los nodos actuales de EMQ X Broker e información clave de otros nodos en el clúster para ayudar a los usuarios a obtener rápidamente el estado de cada nodo.

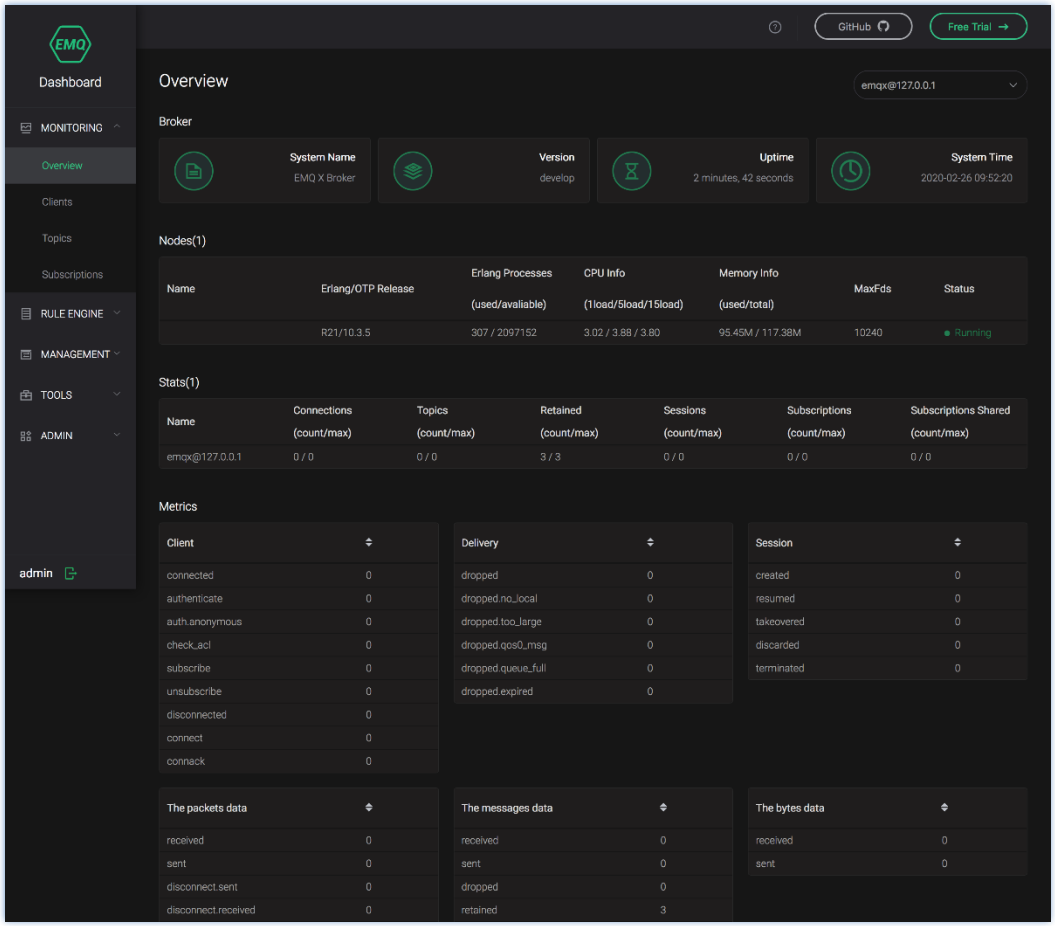


Figura 39. Vista Overview del Dashboard de EMQX.

La página **Clients** proporciona una lista de clientes conectados al nodo especificado y también admite la búsqueda directa de clientes a través de Client ID. Además de ver la información básica del cliente, también al hacer clic en el Kick Outbotón en el lado derecho de cada registro, se puede expulsar al cliente. Esta operación desconectará al cliente y finalizará su sesión.

En la pestaña **Subscriptions ,**en la parte de los detalles del cliente, se puede ver la información de suscripción del cliente actual y crear o cancelar suscripciones. El motor de reglas de EMQ X Broker puede procesar mensajes y eventos de manera flexible, como convertir el mensaje a un formato específico y almacenarlo en una tabla de base de datos o reenviarlo a la cola de mensajes.

La página **Plugins** enumera todos los complementos que EMQ X Broker puede encontrar, incluidos los complementos oficiales de EMQ X y los complementos propios que se hayan desarrollado de acuerdo con los estándares oficiales de EMQ X.

La página **Applications** enumera las aplicaciones creadas actualmente. Se pueden realizar operaciones como crear aplicaciones, deshabilitar temporalmente o iniciar permisos de acceso para una aplicación determinada.

La página **Websocket** proporciona una herramienta de cliente WebSocket (figura 41) sencilla pero eficaz, que incluye funciones de conexión, suscripción y publicación. Al mismo tiempo, se pueden ver los datos del mensaje que envía y recibe.

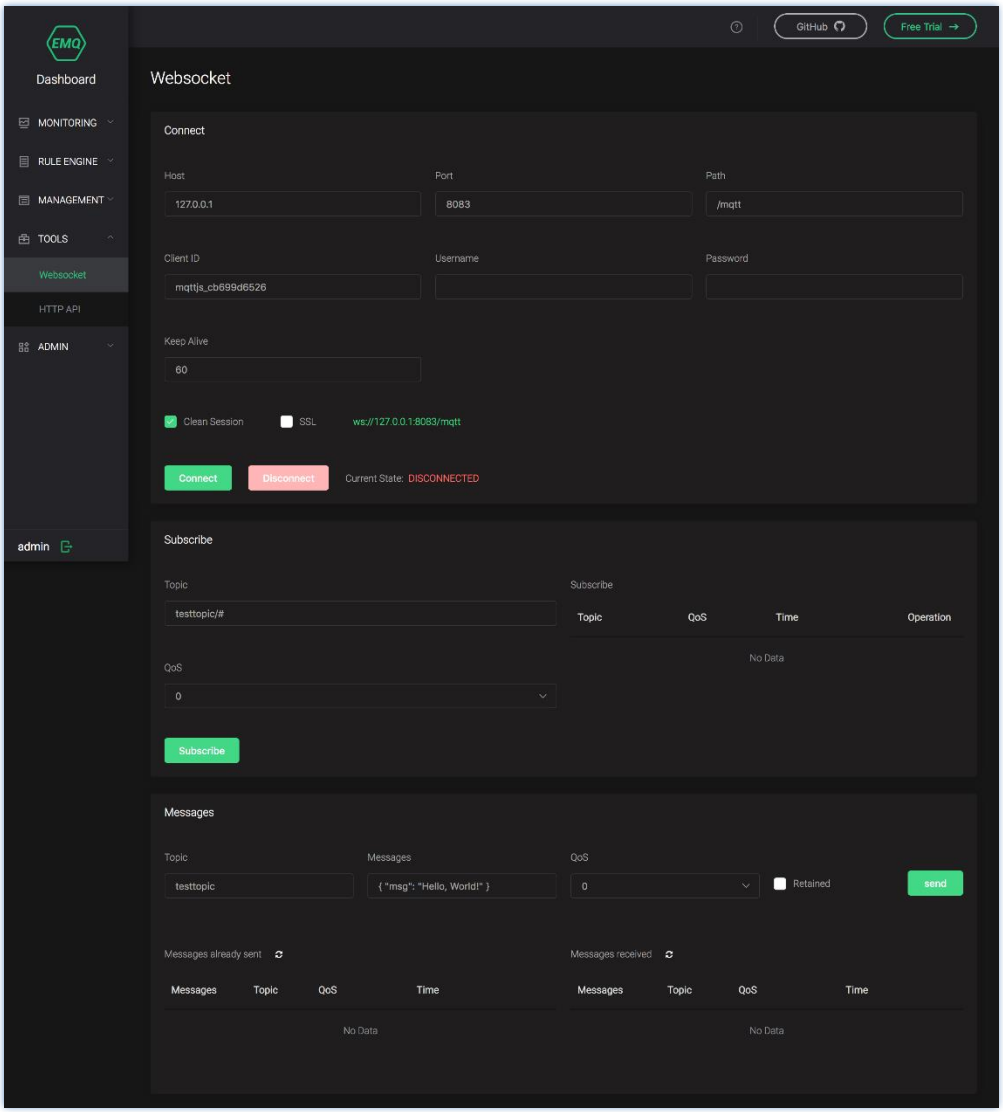


Figura 40. Página Websocket.

Adicionalmente, este Dashboard incluye las siguientes páginas **API HTTP** donde se muestran todas las API HTTP admitidas actualmente por EMQ X Broker y sus descripciones. La página **Users** en la cual se pueden ver y administrar los usuarios que pueden acceder y operar en el Dashboard. En la página **Settings** se puede configurar el idioma y el tema del panel y por último, la página **Help** donde se muestran los enlaces a fórums y preguntas más frecuentes para ayudar a los usuarios en caso de problemas.

3.4 Estructura general de la aplicación en Node- RED

En esta sección de la memoria se pretende explicar principalmente, la lógica utilizada para unir los nodos que componen el backend de la aplicación y como queda reflejada esta composición, en la interfaz gráfica del programa.

La aplicación agrupa en dos opciones las acciones que controlan las funciones de los sensores. Las opciones creadas son las siguientes:

- **Control:** esta opción tiene como objetivo convertir la vivienda en un lugar más confortable a través de la gestión de dispositivos y actividades domésticas. Entre las funciones que se pueden gestionar están: ***control de luces***, ***obtener datos de la climatización*** y ***controlar el riego de plantas ornamentales*** (figura 42)
- **Seguridad:** aporta seguridad mediante la opción de ***cámara de vigilancia*** para controlar personas, animales y bienes. Además, ofrece la posibilidad de activar alarmas técnicas como ***detector de incendios*** y ***fugas de gas,*** para ayudar a prevenir accidentes (figura 43)

Dashboard de la aplicación en Node-RED

Figura 41. Interconexión de los nodos en Node–RED de la opción "Control".

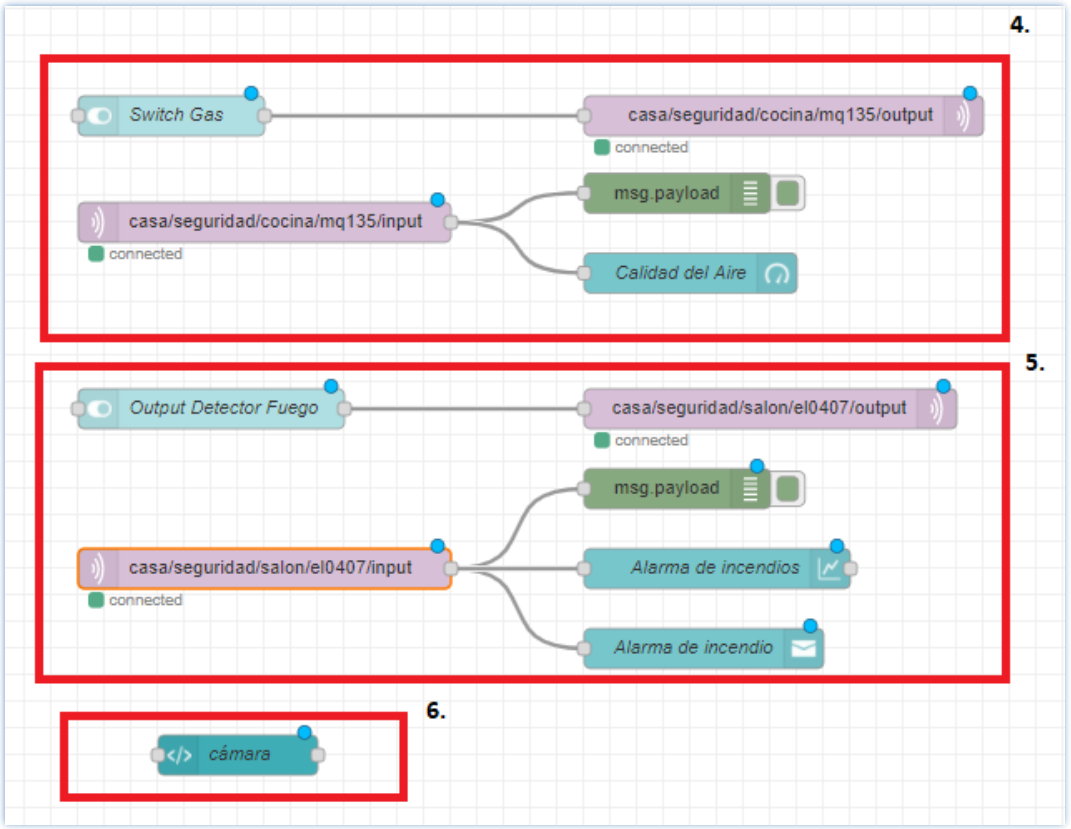


Figura 42. Interconexión de los nodos en Node–RED de la opción "Seguridad".

En las ***Figura 42*** y ***Figura 43*** se pueden apreciar la estructura de los nodos que componen las opciones **"Control" y "Seguridad"**. Para ayudar en la comprensión del diagrama, se delimitaron numéricamente las funciones de la siguiente manera:

1. El Tiempo
2. Control de luces
3. Control del riego
4. Fugas de gas
5. Detector de incendios
6. Cámara de vigilancia

Debido a que la estructura de los nodos es muy similar en algunas de las funciones, se pasará a explicar la estructura elegida para las funciones del ***control de la climatización*** y el de la ***cámara de vigilancia***. Con esta explicación, se deberían comprender las otras funciones que no se explicaron como: ***el control de luces***, ***el riego de plantas ornamentales***, ***el control de fugas de gas*** y ***el detector de incendios***, con excepción de que cada uno de los nodos de salida y de entrada MQTT en estas funciones utilizan un tópico específico para publicar o recibir los mensajes.

Configuración de la función "Climatización"

Para controlar la función de la temperatura se utiliza un interruptor que publica el string ***on/off*** para controlar la salida del ESP32. Unido al switch, hay un nodo de salida MQTT que es el encargado de publicar bajo el topic "**casa/control/salon/dth11/output"** un mensaje en el ESP32 de acuerdo con el estado del interruptor (figura 44)

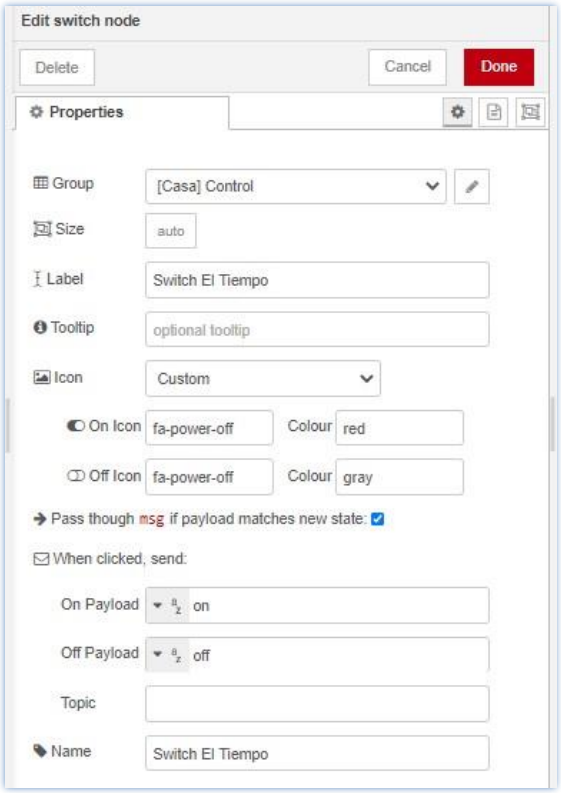


Figura 43. Configuración del switch de control en la función de Climatización.

Se han insertado también dos nodos de entrada MQTT suscritos a los temas: (***casa/control/salon/dth11/input/temperatura***) y (***casa/control/salon/dth11/input/humedad***) para recibir los datos procedentes del sensor DTH11 que se encuentra conectado al ESP32. Es necesario configurar también en estos nodos de entrada y salida MQTT, la dirección IP del servidor en el que está instalado el bróker (en este caso, la dirección IP (***34.91.135.89)*** de la máquina virtual creada en Google Cloud Platform). Todas las demás configuraciones en los nodos están configuradas correctamente de forma predeterminada. Un ejemplo de configuración de estos nodos se muestra en la Figura 45.

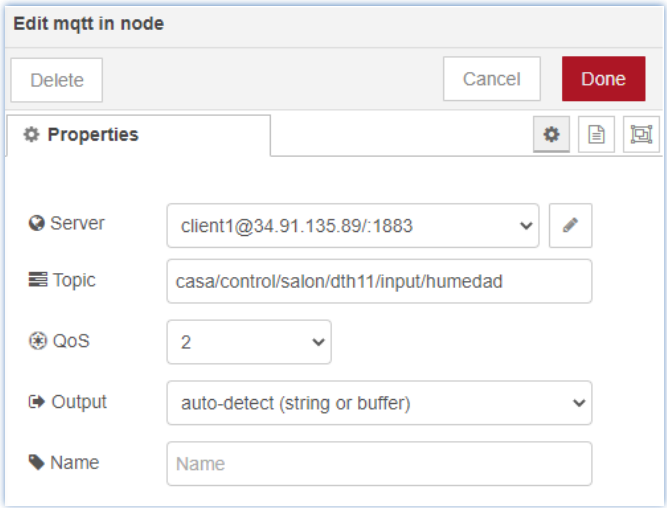


Figura 44. Configuración del nodo MQTT de entrada que recibe datos de humedad.

Por último, se inserta un gráfico para mostrar las lecturas del sensor de temperatura y un medidor de calibre para mostrar los porcentajes de humedad que registra el sensor, como se indica en las figuras 46 y 47.

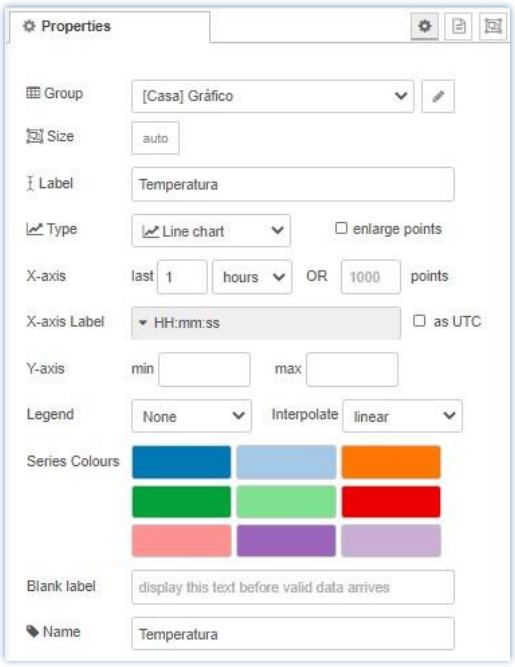


Figura 45.Configuración del gráfico de temperatura.

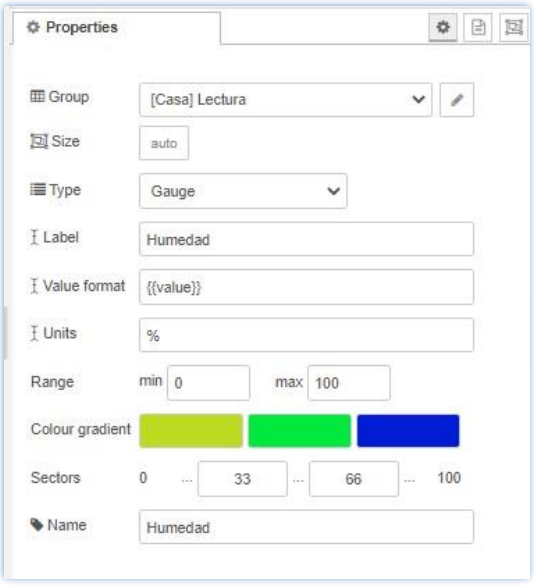


Figura 46. Configuración del gráfico de humedad.

Configuración de la función "Cámara de vigilancia"

Para agregar el servidor web de transmisión de video en Node- RED se utiliza un nodo *Template*. Dentro de su configuración se define el tamaño del video y la IP desde la cual está transmitiendo la ESP32 CAM (figura 48)

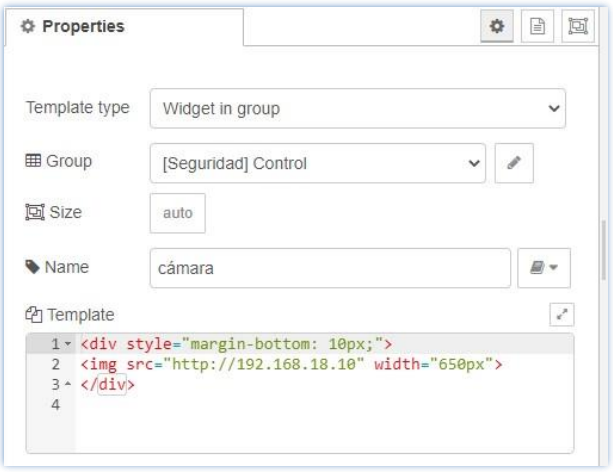


Figura 47. Configuración del widget Template de la cámara de vigilancia.

Interfaz gráfica de la aplicación

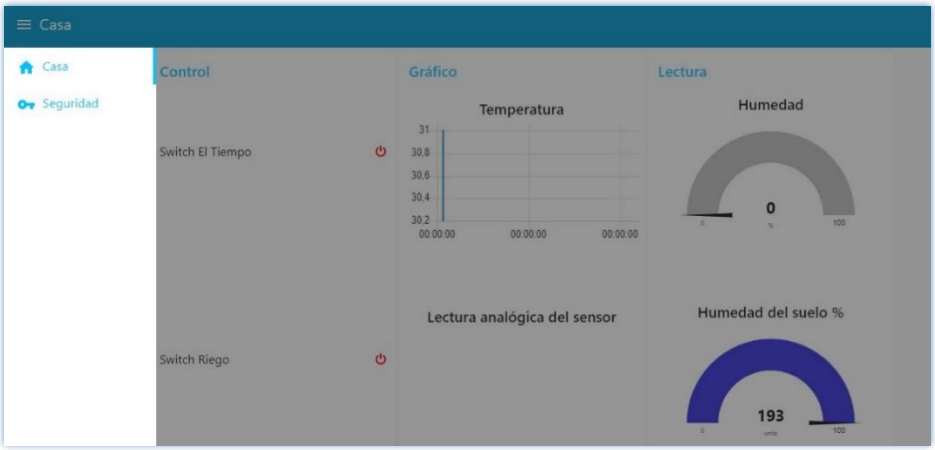


Figura 48. Vista del Menú principal de la interfaz gráfica.

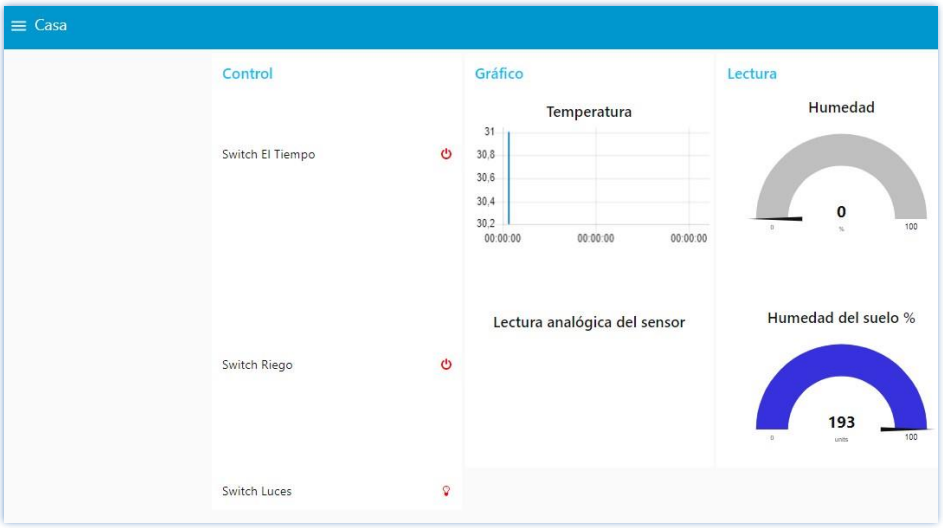


Figura 49. Vista de la opción "Casa".

Figura 50. Vista de la opción "Seguridad".

3.5 Programación de los sensores en el IDE de Arduino

A continuación, se va a explicar la programación realizada en el ESP32 para controlar los diferentes sensores que componen la aplicación desarrollada. En aras de explicar, con la mayor brevedad posible, el desarrollo del código se indicará con detalle la programación del sensor DTH11 ya que hay partes del código comunes para todos los sensores. Luego se explicarán las partes del código que son específicas para cada uno de los sensores.

Configuraciones en el IDE de Arduino

Como requisito previo a este paso, es necesario cerciorarse que se está trabajando con la última versión del IDE de Arduino, la cual se descarga desde su página web: https://www.arduino.cc/en/main/software.

Para poder programar el ESP32 usando el IDE de Arduino, es necesario instalar previamente un complemento que permita trabajar con la placa ESP32. Los pasos seguidos son los siguientes:

1. En el IDE de Arduino, se selecciona la opción **Archivo > Preferencias.** Una vez dentro de la configuración de la opción **Preferencias**, se ingresa en el campo de "URL adicionales del administrador de tablero" la siguiente url: https://dl.espressif.com/dl/package_esp32_index.json y damos "Aceptar"(figura 52)
2.

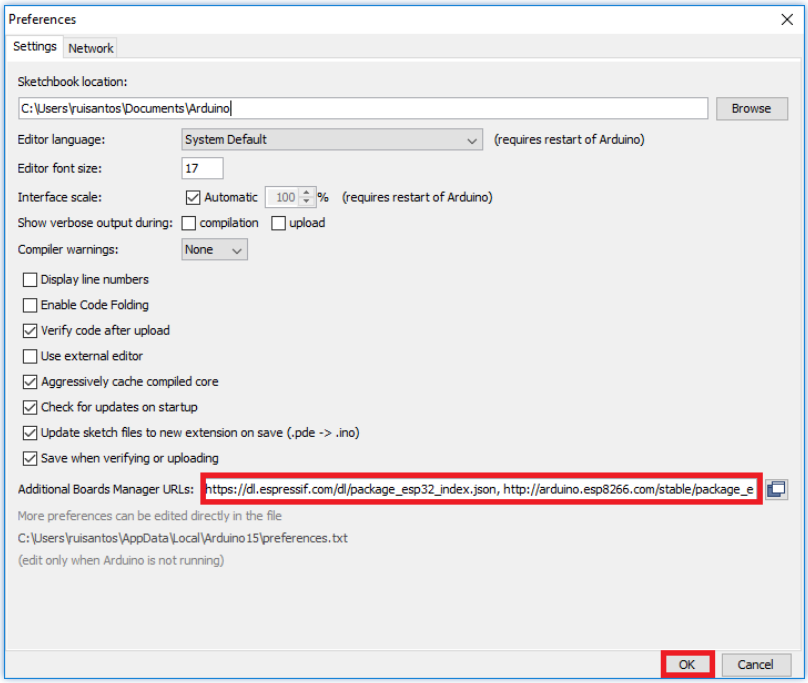


Figura 51. Configuración de la opción "Preferencias".

3. Posteriormente, se va a la opción **Herramientas > Placas > Gestor de tarjetas** para adicionar la placa deseada

4. Buscamos la opción **ESP32** y presionamos en el botón de instalación para **"ESP32 by Espressif Systems"**.

Una vez instalada la librería de ESP32, se selecciona la placa **"ESP32 Dev Module"** (figura 55) en la que se sube el programa una vez compilado. También es necesario seleccionar adecuadamente el puerto COM serie *(ej. COM3)* y la velocidad de subida "Upload Speed" *(ej. 115200 baudios).*

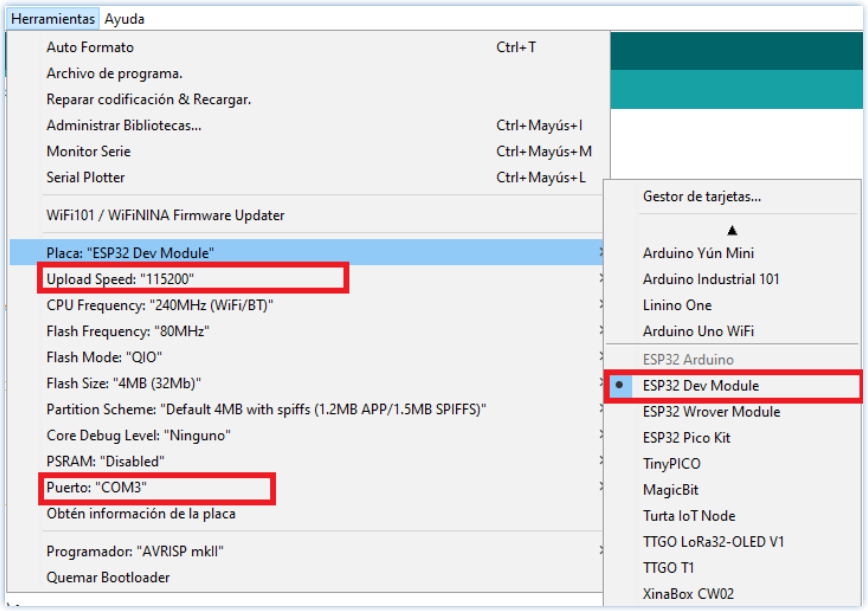


Figura 52. Placa ESP32 Dev Module.

Es importante conocerque, una vez se le da al botón **"Subir"** para cargar el código en la placa y se muestre el mensaje ***"Connecting…"***, es necesario dejar presionado el botón **BOOT** del ESP32 para que entre en modo programación. Si no se hace así, dará el mensaje de error mensaje de error ***" Se produjo un error fatal: No se pudo conectar a ESP32: Se agotó el tiempo de espera ... Conectando ... "***

Para instalar una librería en el IDE de Arduino, se debe ir a **Herramientas**> **Administrar bibliotecas**. En el "**Gestor de librerías**" se puede hacer una búsqueda de las librerías que se necesita utilizar (figura 56)

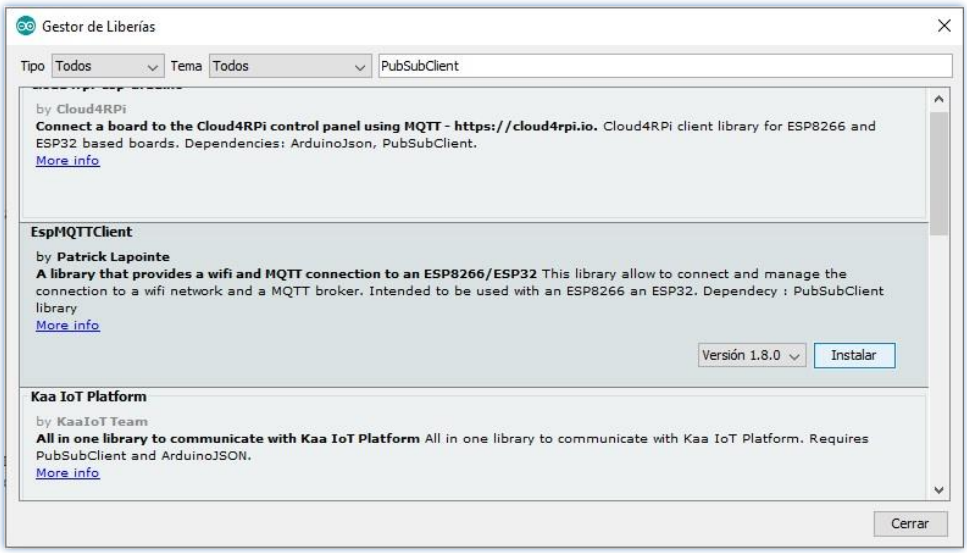


Figura 53.Instalación de librerías en IDE de Arduino.

Otra opción para añadir una librería es añadirla directamente en la carpeta donde el Arduino suele almacenarlas, ubicado en ***C:\Users\UserFolder\Documents\Arduino\libraries***. Una vez instaladas, se debe cerrar y volver abrir el IDE. Así la librería queda lista para ser implementada.

Descripción del proyecto

La estructura general del trabajo consta de varias partes. Hay una aplicación Node-RED que controla las salidas del ESP32 y recibe las lecturas del sensor conectado al ESP32, usando el protocolo de comunicación MQTT. Esta aplicación Node-RED se está ejecutando en la máquina virtual de Google Cloud creada.

Por otra parte, se utiliza EMQ X, broker MQTT responsable de recibir todos los mensajes, filtrarlos, decidir quién está interesado en ellos y publicarlos luego a todos los clientes suscritos.

La siguiente Figura 57 muestra una descripción general de la arquitectura planteada:

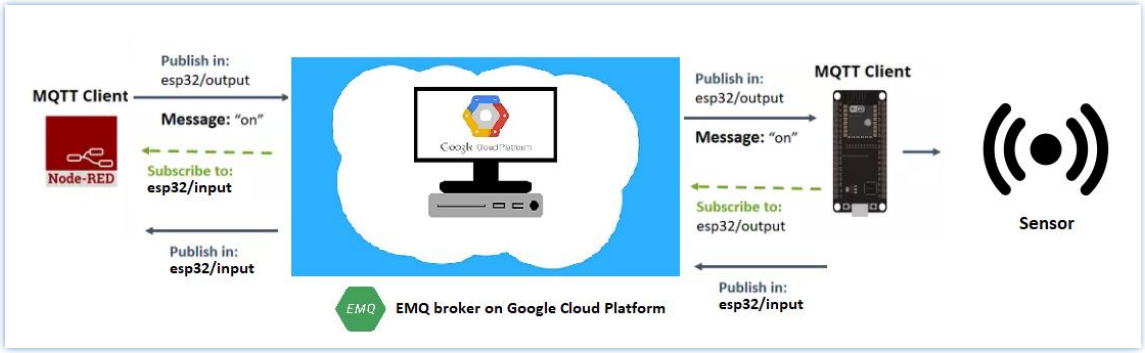


Figura 54. Arquitectura general del proyecto.

3.5.1 Programación del sensor DTH11

De manera muy breve, la función principal de este programa es recibir los mensajes ("**on**" o "**off**") que se publican al variar el estado del switch de control desde la aplicación Node-RED. Estos se envían en el topic "**casa/control/salon/dth11/output**" que coincide con el tema al que está suscrito el ESP32. En dependencia del mensaje recibido, se enciende o apaga el LED de control que indica si está en funcionamiento el sensor.

Por otra parte, el ESP32 publica los temas: "**casa/control/salon/dth11/input/temperatura**" y "**casa/control/salon/dth11/input/humedad**" para enviar los datos leídos. La aplicación Node-RED está suscrita a esos temas, por lo tanto, recibe las lecturas de humedad y temperatura que son las que se muestran en el gráficos o indicadores seleccionados.

Diagrama esquemático

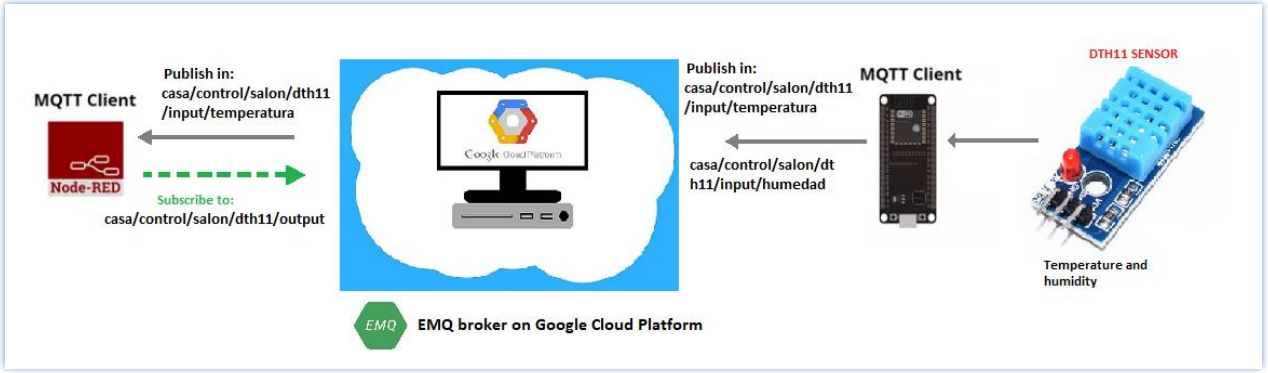


Figura 55. Diagrama esquemático con el sensor DTH11.

Montaje del circuito

A la salida del sensor DTH11 es necesario conectar una resistencia de 10 10 KΩ para poder leer desde el ESP32.

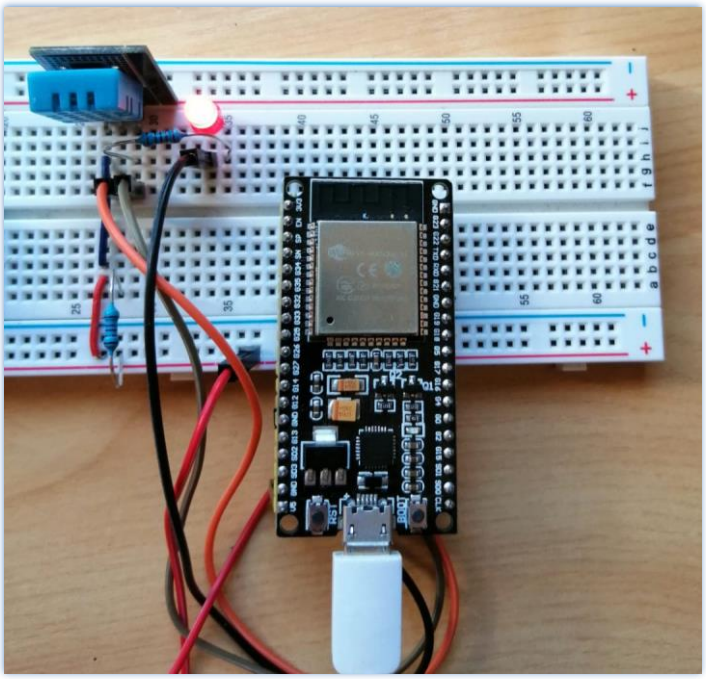


Figura 56.Montaje del circuito con el sensor DTH11.

Código del programa

Como primera parte del código, se importan las librerías necesarias que luego permitirán utilizar determinadas funciones en el programa.

Estas librerías son:

- **Librería Wifi:** con esta librería se puede instanciar servidores, clientes o enviar y recibir paquetes UDP a través de la wifi. Además, permite conectar con redes abiertas o encriptadas WEP o WPA. La dirección IP puede ser asignada de manera estática o por DHCP. La librería puede administrar también DNS.
- **Librería PubSubClient:** Esta biblioteca básicamente permite que el ESP32 se pueda comunicar con Node-RED. Proporciona un cliente para realizar mensajes simples de **publicación/suscripción** con un servidor MQTT.
- **Librerías de Adrafuit:** Para leer desde el sensor DHT, es necesario instalar las librerías **DHT sensor** y **Adafruit Unified Sensor**.
-

Figura 57. Importación de las librerías.

Ahora se declararán los parámetros de la wifi a la que se desea conectar, SSID (nombre de la wifi) y la contraseña. También es necesario definir el dominio o la dirección IP del bróker MQTT al que deseamos conectar, en este caso, la dirección IP de la máquina virtual en la que está instalado el bróker ***34.91.135.89.*** Se crea el objeto **espClient** de la clase **WifiClient,** (**WifiClient espClient;**) y un objeto **client** de la clase **PubSubClient** que hereda de la clase **WifiCient** (**PubSubClient client (espClient);**) para poder conectar a Internet y al servidor MQTT (figura 61)

```
//*********************************//
//Configuración de la conexión wifi
//*********************************//
const char* ssid = "Wifi Movil";
const char* password = "WifiTFG";

const char* mqtt_server = "34.91.135.89"; //Conexión con el broker MQTT


WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;
```

Figura 58. Configuración de los parámetros de la wifi y del bróker MQTT.

En el siguiente punto del código se definen los pines y la configuración del sensor DTH. El pin 13 es al cual se conecta el LED para controlar el encendido y apagado del sensor. También se define el pin 4, como pin por donde se leerán los datos de humedad y temperatura del sensor DTH. Es necesario definir también el tipo de sensor DTH que se está utilizando (**#define DHTTYPE DHT11**). En este caso se especifica que se está utilizando el sensor DTH11, esto es importante ya que la biblioteca admite los sensores: DTH11, DTH21 y DTH22.

Donde pone "**DHT dht (DHTPIN, DHTTYPE);**" se crea un objeto **dht** de la clase **DHT** en el pin y con el tipo de sensor que se ha especificado anteriormente.

Por último, en esta parte del código inicializan las variables de tipo float: ***temperature*** y ***humidity*** donde se almacenan los valores de humedad y temperatura leídos por el sensor.

```
//**********************************//
//Configuración del sensor DHT11
//**********************************//
#define BUILTIN_LED 13 //definimos el LED para controlar el ON/OFF del DHT11
#define DHTPIN 4 // pin al que se conecta la salida del sensor DTH11 a la entrada del ESP32
#define DHTTYPE DHT11   // DHT 11
DHT dht(DHTPIN, DHTTYPE);

float temperature = 0;
float  humidity = 0;
```

Figura 59. Definición de los pines y configuración del sensor DTH11.

En la función **setup ()** se configura el sketch de todo el programa. Primeramente, se inicializa la depuración serial a una velocidad de 115200 baudios "**Serial.begin(115200);**" y se inicializa el sensor DHT "**dht.begin();**".

```
  void setup() {
  Serial.begin(115200);
  dht.begin();

  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  client.connected();

  pinMode(BUILTIN_LED, OUTPUT);
}
```

Figura 60. Función setup().

Seguidamente en la función ***setup_wifi*** se estalece la conexión con la wifi de acuerdo con los valores de SSID y password proporcionados al inicio. Como primera condición de la función, si se establece la conexión, muestra el mensaje por consola "**Wifi connected**" y el mensaje "**IP address**" con la información de la dirección IP privada asignada. En caso de que no se establezca la conexión, hace un delay de 5 segundos e imprime puntos suspensivos (...) en espera a que se realice la conexión.

```
//*********************************//
//Conexión a la wifi
//*********************************//
void setup_wifi() {

  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

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

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

Figura 61. Función setup_wifi()

Para establecer los parámetros de la conexión con el servidor MQTT se utiliza la función **client.setServer(mqtt_server, 1883)**. En esta se define la dirección IP del servidor y el puerto al que se desea conectar (puerto1883 específico para realizar la comunicación MQTT). La función **client.setCallback(callback);** ejecuta la función callback con la cual, el ESP32 recibe los mensajes MQTT de los temas suscritos. En dependencia del tema y el mensaje MQTT, se enciende o apaga el led. Como última parte del **setup()** del programa, se llama a la función **client.connected()** para establecer la conexión con el servidor y se define el pin **BUILTIN_LED** como pin de salida (**pinMode(BUILTIN_LED, OUTPUT);**) .

```
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: " );
  String messageTemp;

  for (int i = 0; i < length; i++) {
  Serial.print((char)payload[i]);
  messageTemp += (char)payload[i];
 }
  Serial.println();

  // If a message is received on the topic esp32/output, you check if the message is either "on" or "off".
  // Changes the output state according to the message
   if (String(topic) == "casa/control/salon/dth11/output") {
     Serial.print("Changing output to ");
     if(messageTemp == "on"){
     Serial.println("on");
     digitalWrite(BUILTIN_LED, HIGH);
     delay(1000);
     }
     else if(messageTemp == "off"){
      Serial.println("off");
      digitalWrite(BUILTIN_LED, LOW);
      delay(1000);
    }
  }
}
```

Figura 62. Función callback ().

Después de crear una función **setup()**, que inicializa y establece los valores iniciales, la función **loop()** hace precisamente lo que sugiere su nombre y se repite consecutivamente. Permite que el programa cambie y responda. Se usa para controlar activamente la placa Arduino.

La primera condición de la función **loop()** valida que se haya establecido la conexión a la wifi, en caso que no se haya conectado ejecuta la función de reconexión (**reconnect()**).

```
void loop() {

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

Figura 63. Función loop().

Precisamente la función **reconnect()** lo primero que hace es validar si estamos conectados o no a la red (**while (!client.connected())**) . En caso no estar conectados, inicia el proceso de conexión. Primero muestra un mensaje por consola indicando que se está estableciendo la conexión con el bróker MQTT. Para establecer la conexión es necesario un **ID** que identifique al ESP32. Para esto se utiliza una cadena de caracteres formada por el string **"ESP32D0WDQ6-"** con otro string, resultado de hacer una función random de un valor hexadecimal (**clientId += String(random(0xffff), HEX);).** Con este **ID** cada conexión tendrá un valor único para evitar conflictos en el servidor. Si se establece la conexión, muestra un mensaje de confirmación y se suscribe al topic "**casa/control/salon/dth11/output**".

En caso de que no se pudiese establecer la conexión, muestra en mensaje de error y el código del error. Luego de 5 segundos intentará establecer de nuevo la conexión y mostrará un mensaje de notificación al usuario.

```
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    String clientId = "ESP32D0WDQ6-";
    clientId += String(random(0xffff), HEX);

    //Attemt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");

      //Suscripción
      client.subscribe("casa/control/salon/dth11/output");

    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

Figura 64. Función reconnect.

Continuado con el resto del código de la función Loop(), como se muestra en la figura 69,el ESP32 hace la lectura de la temperatura (**temperature = dht.readTemperature();**) en grados Celsius. Para enviar los datos es necesario convertir la variable de tipo float de la temperatura en una matriz de caracteres (**dtostrf(temperature, 1, 2, tempString);**) de modo que se pueda publicar la lectura de temperatura en el tema "**casa/control/salon/dth11/input/temperatura**". Lo anteriormente descrito también se cumple para el caso de la lectura de la humedad, con excepción del topic que es "**casa/control/salon/dth11/input/humedad**".

Esta información se envía cada 5 segundos con la siguiente función:

```
long now = millis();
if (now - lastMsg > 5000) {
lastMsg = now;
```

Figura 65. Envío de mensajes al servidor MQTT.

```
// Temperature in Celsius
temperature = dht.readTemperature();


// Convert the value to a char array
char tempString[8];
dtostrf(temperature, 1, 2, tempString);
Serial.print("Temperature: ");
Serial.println(tempString);
client.publish("casa/control/salon/dthll/input/temperatura", tempString);

humidity = dht.readHumidity();

// Convert the value to a char array
char humString[8];
dtostrf(humidity, 1, 2, humString);
Serial.print("Humidity: ");
Serial.println(humString);
client.publish("casa/control/salon/dthll/input/humedad", humString);
}
 if (isnan(temperature) || isnan(humidity)) {
 Serial.println(F("Failed to read from DHT sensor!"));
 delay(2000);

 }
}
```

Figura 66. Lectura de los valores de humedad y temperatura.

En caso de que no se puedan leer estos valores se muestra en mensaje de error con un delay de 2 segundos.

3.5.2 Programación del sensor Wifi Relay

El código para controlar un relé con el chip ESP32 es tan simple como controlar un LED o cualquier otra salida. En este ejemplo, como estamos usando una configuración normalmente abierta, necesitamos enviar una señal BAJA para dejar que fluya la corriente y una señal ALTA para detener el flujo de corriente.

Diagrama esquemático

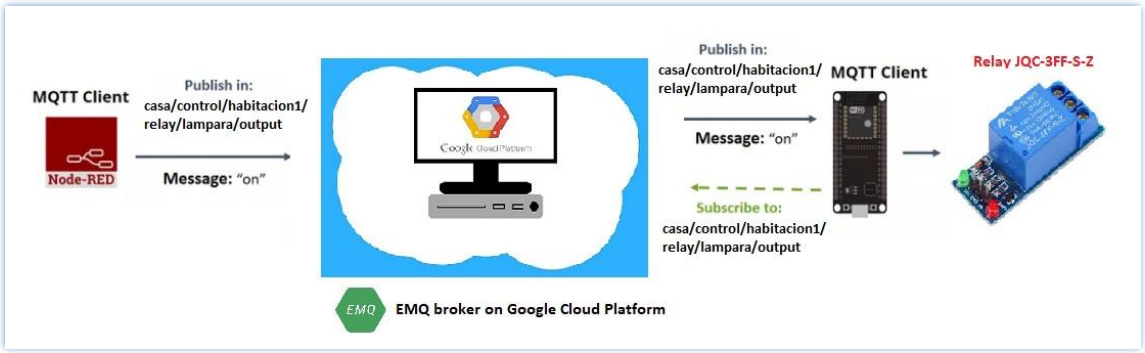


Figura 67. Diagrama esquemático con el Relay JQC-3FF-S-Z.

Montaje del circuito

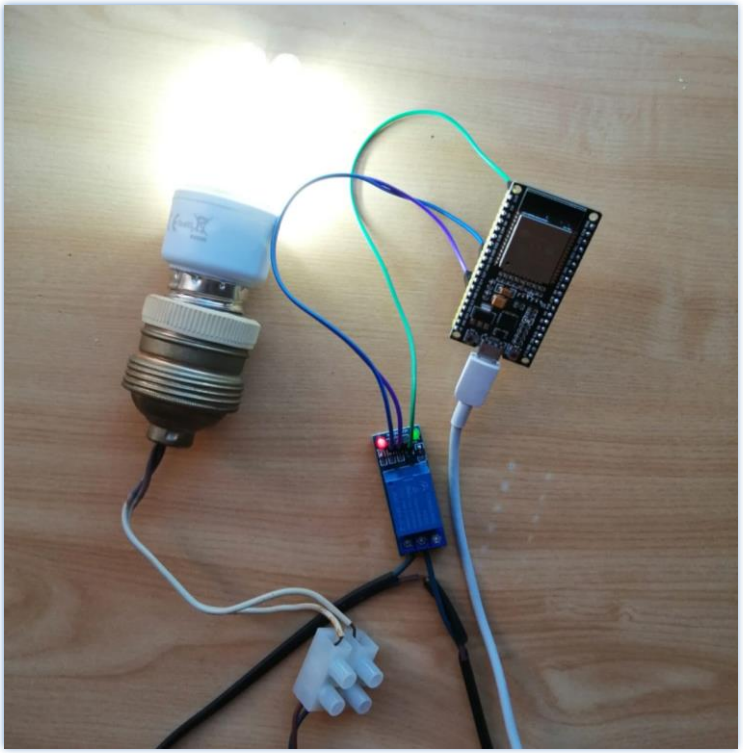


Figura 68.Montaje del circuito con el relay JQC-3FF-S-Z y el ESP32.

Código del programa

Debido a que la estructura del código es muy similar al anterior, se matizarán solo los pequeños detalles que son específicos para este dispositivo.

Para el desarrollo de este programa solo es necesario importar las librerías **Wifi** y **PubSubClient**. Se define el pin 13 de salida al cual se conecta el led y el pin 26 al que se conectará la entrada del relay.

```
//**********************************//
//Configuración del Relay
//**********************************//
#define BUILTIN_LED 13 //definimos el LED para controlar el ON/OFF del relay
#define relay 26 // Digital pin connected to the DHT sensor
```

Figura 69. Configuración de los pines.

En la función **loop()** se define el pin del relay como salida (**pinMode(relay, OUTPUT);**).Tanto en la función **reconnect** como en la de **callback** el topic a utilizar es "**casa/control/habitacion1/relay/lampara/output**". Específicamente en la función **callback,** al recibir un mensaje "**on**" para que circule la corriente se debe enviar una señal digital de valor 1 y para abrir el circuito un 0.

3.5.3 Programación del sensor Capacitive Soil Moisture

Este sensor se va a utilizar para medir la humedad relativa del suelo en los jardines y en dependencia de ese valor, activar el riego de las plantas de forma remota. La particularidad de este sensor es que el valor de salida final se ve afectado por la profundidad de inserción de la sonda y la compactación del suelo a su alrededor. Se considera "value_1" como suelo seco y "value_2" como suelo empapado. Este es el rango de detección del sensor. Por ejemplo: Value_1 = 520; Value_2 = 260. El rango se dividirá en tres secciones: seco, húmedo y agua.

Diagrama esquemático

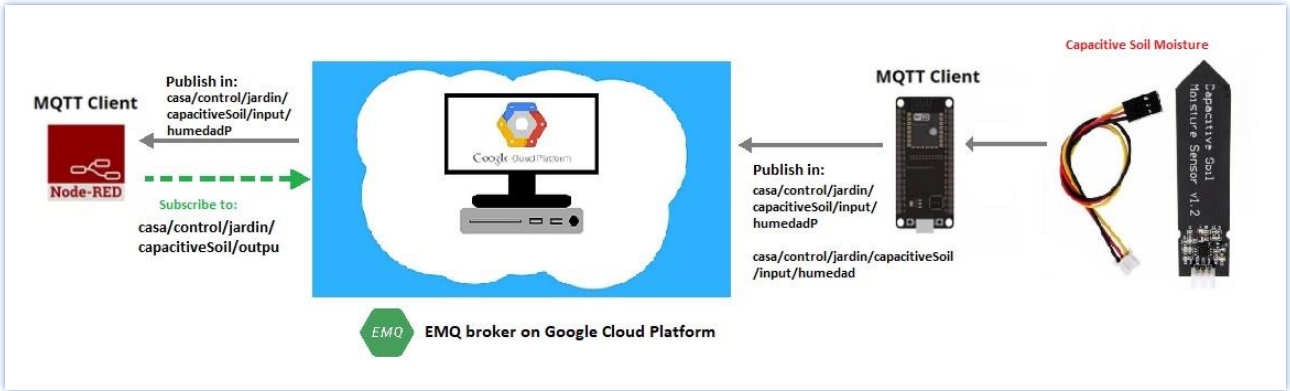


Figura 70. Diagrama esquemático con el sensor Capacitive Soil Moisture.

Montaje del circuito

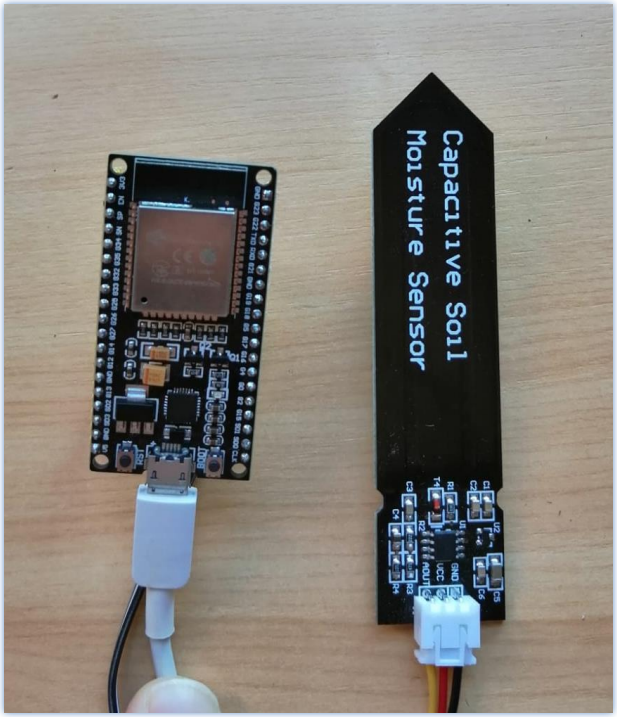


Figura 71. Figura 72. Montaje del circuito con sensor Capacitive Soil Moisture.

Código del programa

Se utilizan las mismas librerías que en el programa del relay, pero una de las diferencias es que se utilizan dos tópicos: " **casa/control/jardin/capacitiveSoil/input/humedad**" y " **casa/control/jardin/capacitiveSoil/input/humedadP**". Una para enviar la lectura analógica de la humedad relativa del suelo y la otra para enviar el valor de la humedad convertido a porciento.

Se define el pin A0 como pin de salida de datos procedente del sensor y el pin digital 26 al que se conectará el LED. Es necesario definir tres variables para lograr una correcta calibración del sensor. La primera variable **AirValue,** tiene el valor límite del suelo seco "Humedad: 0% RH"(valor del sensor cuando la sonda se expone al aire). La segunda variable **WaterValue,** el valor límite del suelo húmedo "Humedad: 100% HR" (valor del sensor cuando la sonda se expone al agua). Por último, la variable **intervals** para definir el rango de las tres secciones.

```
//*********************************//
//Configuración del Capacitive Soil Sensor
//*********************************//
#define SENSOR A0   // pin de salida del sensor
#define RELAY 26 // Digital pin connected to the DHT sensor


const int AirValue = 3365;   //valor de referecia cuando el sensor está completamente seco
const int WaterValue = 1500;  //valor de referecia cuando el sensor está completamente sumergido en agua
int intervals = (AirValue - WaterValue)/3;
int soilMoistureValue = 0;
int soilMoistureValueP = 0;
```

Figura 73. Definición de pines y variables.

En la función **reconnect()** se utiliza el topic "**casa/control/jardin/capacitiveSoil/output**" para suscribirse a los mensajes procedentes de Node-RED. En la función **callback()** en dependencia del mensaje recibido "**on/off**" se activará el relay que activa el riego de las plantas. Si el mensaje recibido es "**on**" se envía un 1 para dejar pasar la corriente. En caso contrario, si se recibe un "**off**" se envía un 0 y se desactiva el riego.

En la función **setup()** se define **pin SENSOR** como entrada y **pin RELAY** como salida. En la función **loop()** a diferencia de los sensores anteriores, se realiza la lectura analógica del sensor **int soilMoistureValue = analogRead(SENSOR);** y se definen las condiciones que determinarán las tres secciones: seco, húmedo y agua.

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

  int soilMoistureValue = analogRead(SENSOR);


if(soilMoistureValue > WaterValue && soilMoistureValue < (WaterValue + intervals))
{
  Serial.print("Muy húmedo- ");
}
else if(soilMoistureValue > (WaterValue + intervals) && soilMoistureValue < (AirValue - intervals))
{
  Serial.print("Húmedo- ");
}
else if(soilMoistureValue < AirValue && soilMoistureValue > (AirValue - intervals))
{
  Serial.print("Seco- ");
}
```

Figura 74.Lectura analógica del sensor y definición del rango de lecturas.

Para dar más información a los usuarios se muestra también las lecturas en porcentajes. El valor en porciento se guarda en la variable soilMoistureValueP y la función usada es soilMoistureValueP = map(soilMoistureValue,3325,1610,0,100);. Es necesario convertir la variable de tipo int de la humedad en una matriz de caracteres (snprintf (mqtt_payload1, 30, "%ld", soilMoistureValueP);), de modo que se pueda publicar la lectura de la humedad en el topic "casa/control/jardin/capacitiveSoil/input/humedadP". Igualmente sería necesario convertir en una matriz de caracteres, el valor analógico que se lee directamente del sensor, lo que este se enviaría bajo el topic "casa/control/jardin/capacitiveSoil/input/humedad". Los mensajes en este sensor se ha programado para que se envíen cada 1 segundo.

3.5.4 Programación del sensor EL0407

En este proyecto vamos a crear un detector de llama o fuego usando el ESP32 y el sensor EL0407. Este sensor de llama óptico permite detectar la existencia de combustión por la luz emitida por fuego. Con esta solución se intenta crear un dispositivo económico que ayude a prevenir incendios o su detección temprana para intentar salvar vidas y bienes económicos, como estructuras edilicias, mercadería, documentos, etc.

Diagrama esquemático

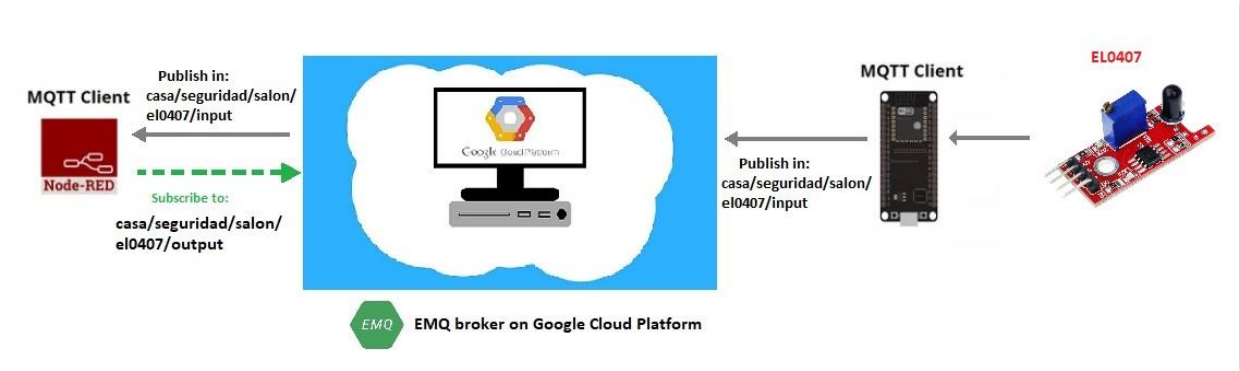


Figura 75. Diagrama esquemático con el sensor EL0407.

Montaje del circuito

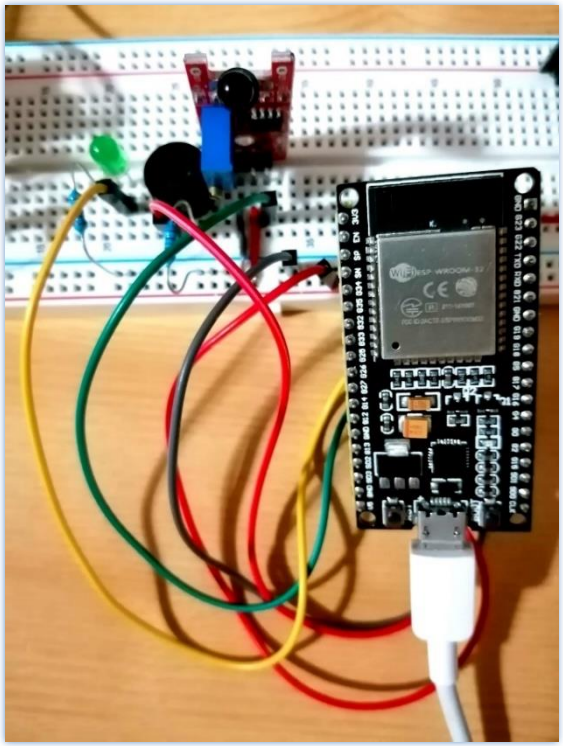


Figura 76. Montaje del circuito con sensor EL0407.

Código del programa

Nuevamente las librerías a utilizar son Wifi y PubSubClient. Se declaran el pin 2 de entrada para recibir las lecturas digitales del sensor. Por el pin 14 de salida se conecta un led que se activará cuando se detecte una llama y por el pin de salida 12 otro led para controlar el encendido del sensor.

En el pin 13 de salida se conecta un zumbador pasivo (buzzer) para generar el sonido de la alarma. Se declaran dos variables pirState y valSensor, la primera con valor LOW, para inicializar el sensor como que no se detecta un incendio. La segunda, con valor 0 ya que almacenará los estados de los pines para controlar el cambio de estado de la detección.

```
int ledPin = 14;                // cuando se detecta una llama se enciende
int inputPin = 2;               // para leer las lecturas del EL0407
int pirState = LOW;             // incializamos el estado del sensor como que no se detectan llamas
int valSensor = 0;                    // variable para la lectura de los estados de los pines
int pinSpeaker = 13;           //salida al buzzer para generar el sonido de alarma
#define GREENLED 14 // está activo siempre que nos se detecten incendios
```

Figura 77. Declaración de pines y variables del sensor EL0407.

Tanto en la función **reconnect()** como en la de **callback(),** el topic al cual se va a suscribir el sensor y del cual se van a revisar los mensajes de control "**on/off**" va a ser el topic **"casa/seguridad/salon/el0407/output"**. En la función **setup()** se realiza la correspondiente configuración de los pines de entrada y salida, la velocidad de subida se establece a 115220 baudios y se establece la conexión.

En la función **loop()**, se realiza una lectura digital de los datos del sensor EL0407, después se debe verificar el valor de la lectura para activar o no la alarma de incendios. Si la lectura del sensor vale 1 ("**if (valSensor == HIGH)")**, se enciende el led rojo y se activa la función **playTone** con una duración y frecuencia determinada para generar la alarma. Como se ha activado la alarma el valor de **pirState** pasa de valor LOW a HIGH. En caso contrario, si el valor de **valSensor** es LOW quiere decir que no se ha detectado ningún incendio. El led rojo se apaga y no se genera ningún sonido; **pirState** deberá tener el valor LOW.

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

  valSensor = digitalRead(inputPin);  // read input value
  if (valSensor == HIGH) {            // check if the input is HIGH
    digitalWrite(ledPin, HIGH);  // turn LED ON
    playTone(300, 160);
    delay(150);


    if (pirState == LOW) {
      Serial.println("Fuego detectado");
      pirState = HIGH;
    }
  } else {
      digitalWrite(ledPin, LOW); // turn LED OFF
      playTone(0, 0);
      delay(300);
      if (pirState == HIGH){
      // we have just turned of
      Serial.println("Fuego extinguido");
      pirState = LOW;
    }
  }
```

Figura 78. Función loop() del sensor EL0407.

Es necesario convertir variable **int valSensor** en una matriz de caracteres de modo que se pueda publicar la lectura del sensor en el topic "**casa/seguridad/salon/el0407/input**" cada 5 segundos.

```
  long now = millis();
  if (now - lastMsg > 5000) {
  lastMsg = now;
  char valSensorString [8];
  dtostrf(valSensor, 1, 2, valSensorString);
  Serial.print(" Lectura del sensor :");
  Serial.println(valSensor);
  client.publish("casa/seguridad/salon/el0407/input", valSensorString );
}
  return;
}
```

Figura 79. Envío de la lectura del sensor EL0407.

3.5.5 Programación del sensor MQ135.

El sensor de gas MQ135 da una idea cualitativa de los gases con compuestos orgánicos volátiles en el aire circundante. Por lo tanto, se puede obtener tendencias, comparar resultados y ver si la calidad del aire aumenta o disminuye. Para obtener mediciones precisas, se debe calibrar el sensor con fuentes conocidas y construir una curva de calibración.

Diagrama esquemático

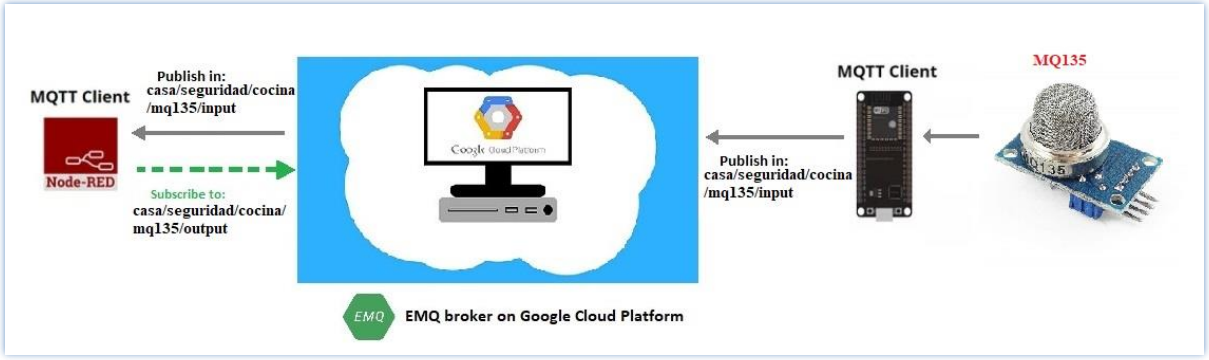


Figura 80. Diagrama esquemático con el sensor EL0407.

Montaje del circuito

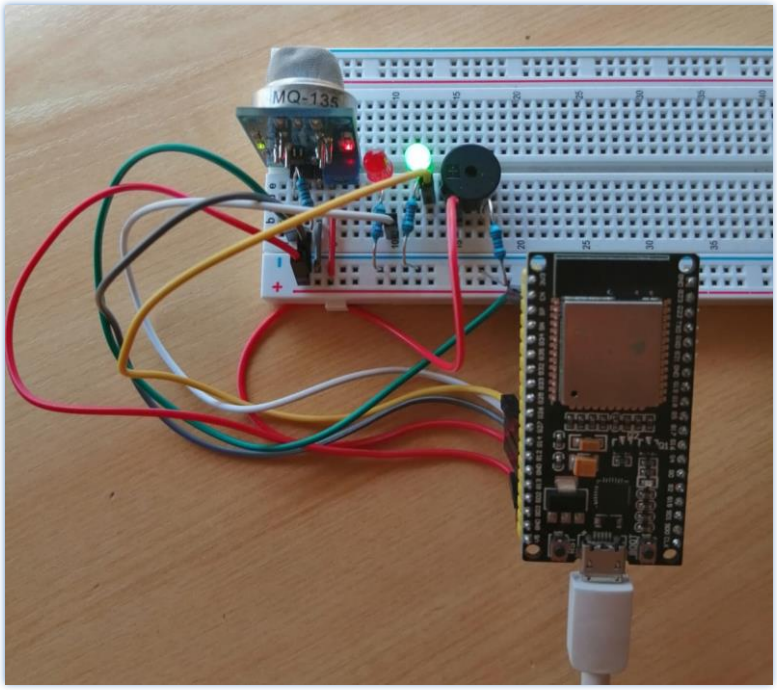


Figura 81. Montaje del circuito con sensor MQ135.

Código del programa

Para obtener las lecturas del sensor MQ135 se utilizan las librerías **Wifi** y **PubSubClient.** Se define el pin 12 para encender un led rojo que indica que se detectan gases peligrosos. El pin 14 para encender un led verde que informa de la puesta en marcha del sensor remotamente desde la aplicación y para indicar que la calidad del aire es buena. Por el pin 13 se activa el zumbador que provoca el sonido de la alarma. Por otra parte, declaramos una variable **sensorThres** que determina el valor a partir del cual la lectura del sensor comienza a considerarse peligrosa.

```
#define REDLED 12 //cuando se activa nos indica que hay gases peligrosos
#define GREENLED 14 // está activo siempre que nos se detecten gases peligrosos

//**********************************//
//Configuración del sensor MQ-135
//**********************************//
//#define SENSOR 2
int sensorThres = 2000;
const int buzzer = 13;
```

Figura 82. Definición de los pines y variables para la lectura en el sensor MQ135.

En la función reconnect() y en la función callback() el topic que se utiliza es "**casa/seguridad/cocina/mq135/output**" para enviar los mensajes de control desde la aplicación Node-RED.

En la función **setup()** se declaran los pines de entrada y salida del sensor, además la velocidad de subida se define a 115200 baudios. En la función **loop()** del programa se realiza la lectura analógica del sensor y se guarda en el variable **lectura**. Luego de hacer la lectura es necesario hacer una comparación del valor leído con el valor de referencia que se define en la variable **sensorThres**. Si el valor leído es mayor que **sensorThres** cada 1 segundo se genera una alarma intermitente que consiste en: apagar el led verde y encender el led rojo junto al zumbador.

```
int lectura = analogRead(A0);
Serial.print(" Calidad del aire: ");
Serial.println(lectura);

if (lectura > sensorThres)
{
   for(int i = 250; i > 0 ; --i)
{
  digitalWrite(buzzer, HIGH);
  digitalWrite(REDLED, HIGH);   // turn the LED on (HIGH is the voltage level)
  digitalWrite(GREENLED, LOW);
  delay(1);
  digitalWrite(buzzer, LOW);
  digitalWrite(REDLED, LOW);
  digitalWrite(GREENLED, LOW);
  delay(1);
}
}
else
{
  digitalWrite(REDLED, LOW);
  digitalWrite(GREENLED, HIGH);
}
delay(1000);
client.loop();
```

Figura 83. Lectura de datos y generador de la alarma en el sensor MQ135.

Tal y como se venía realizando en los programas anteriores, es necesario convertir variable **int lectura** en una matriz de caracteres de modo que se pueda publicar la lectura del sensor en el topic "**casa/seguridad/cocina/mq135/input**" cada 5 segundos.

```
long now = millis();
if (now - lastMsg > 5000) {
lastMsg = now;

char lecturaString [8];
dtostrf(lectura, 1, 2, lecturaString);
Serial.print(" Valor convertido a string: ");
Serial.println(lecturaString);
client.publish("casa/seguridad/cocina/mq135/input", lecturaString );
// Checks if it has reached the threshold value
```

Figura 84. Envío de datos del sensor MQ135.

3.5.6 Programación de la cámara ES32-CAM.

En este proyecto se va a conectar la placa ESP32-CAM para crear una cámara de vigilancia IP. La cámara ESP32 alojará un servidor web de transmisión de video al que puede acceder a través Node-RED.

Diagrama esquemático

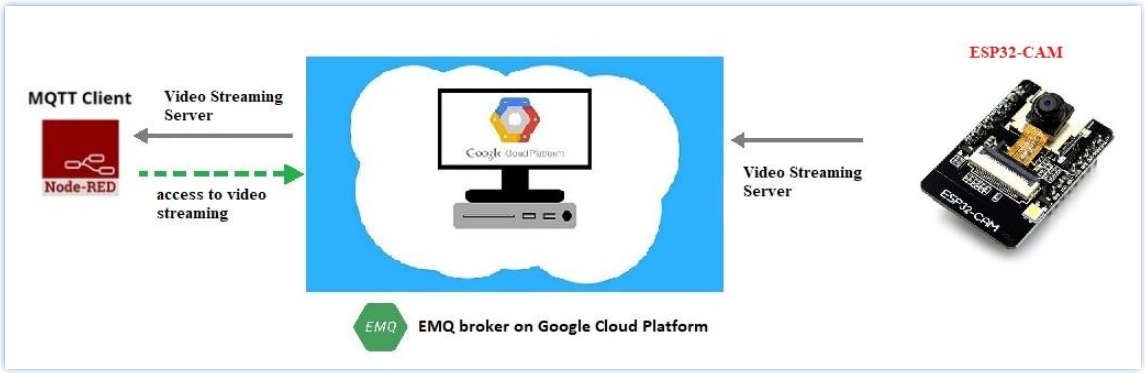


Figura 85. Diagrama esquemático con la placa ESP32-CAM.

Montaje del circuito

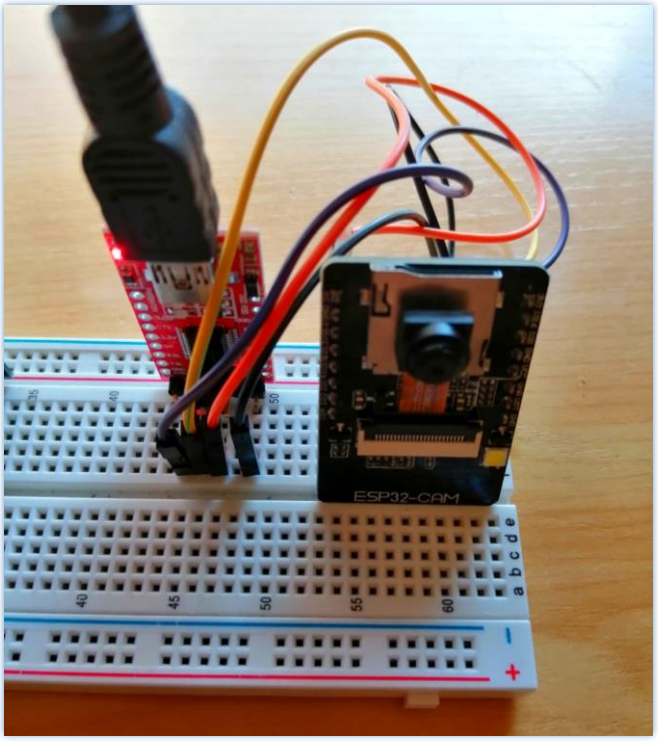


Figura 86. Montaje del circuito con placa ESP32-CAM.

Código del programa

El primer paso para poder transmitir el streaming de video utilizando el ESP32-CAM va a ser importar las librerías necesarias para trabajar con este dispositivo.

```
//*********************************//
//Definición de librerías
//*********************************//
#include "esp_camera.h"
#include <WiFi.h>
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "fb_gfx.h"
#include "soc/soc.h" //disable brownout problems
#include "soc/rtc_cntl_reg.h"  //disable brownout problems
#include "esp_http_server.h"
```

Figura 87. Importación de las librerías del ESP32-CAM.

Similar a lo realizado en los anteriores dispositivos, es necesario definir los parámetros de la red wifi a la que se desea conectar. También es necesario definir el modelo de la ESP32-CAM que se está usando, ya que mediante condicionantes "if" se van a definir los pines de la cámara según el modelo utilizado.

```
const char* ssid = "HUAWEI-2.4G-6uPB";
const char* password = "5cHN7UFM";

//*********************************************//
//Configuración de la cámara según el modelo
//*********************************************//

#define PART_BOUNDARY "123456789000000000000987654321"
#define CAMERA_MODEL_AI_THINKER

#if defined(CAMERA_MODEL_WROVER_KIT)
  #define PWDN_GPIO_NUM    -1
  #define RESET_GPIO_NUM   -1
  #define XCLK_GPIO_NUM    21
  #define SIOD_GPIO_NUM    26
  #define SIOC_GPIO_NUM    27

  #define Y9_GPIO_NUM      35
  #define Y8_GPIO_NUM      34
  #define Y7_GPIO_NUM      39
  #define Y6_GPIO_NUM      36
  #define Y5_GPIO_NUM      19
  #define Y4_GPIO_NUM      18
  #define Y3_GPIO_NUM       5
  #define Y2_GPIO_NUM       4
  #define VSYNC_GPIO_NUM   25
  #define HREF_GPIO_NUM    23
  #define PCLK_GPIO_NUM    22
```

Figura 88. Pines de la ESP32-CAM según el modelo y configuración de la wifi.

Para que la transmisión del vídeo en streaming se haga correctamente es necesario definir ciertos parámetros propios de la transmisión de contenido de varias partes (The Multipart Content-Type), tal y como se hace en la siguiente imagen de códigos:

```
static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary=" PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n";

httpd_handle_t stream_httpd = NULL;

static esp_err_t stream_handler(httpd_req_t *req){
  camera_fb_t * fb = NULL;
  esp_err_t res = ESP_OK;
  size_t _jpg_buf_len = 0;
  uint8_t * _jpg_buf = NULL;
  char * part_buf[64];

  res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
  if(res != ESP_OK){
    return res;
  }

  while(true){
    fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Camera capture failed");
      res = ESP_FAIL;
    } else {
      if(fb->width > 400){
        if(fb->format != PIXFORMAT_JPEG){
```

Figura 89. Configuración del streaming de video.

Con la función **startCameraServer()** se logra acceder al servidor web de streaming de video usando el protocolo HTTP.

```
void startCameraServer(){
  httpd_config_t config = HTTPD_DEFAULT_CONFIG();
  config.server_port = 80;

  httpd_uri_t index_uri = {
    .uri       = "/",
    .method    = HTTP_GET,
    .handler   = stream_handler,
    .user_ctx  = NULL
  };

  //Serial.printf("Starting web server on port: '%d'\n", config.server_port);
  if (httpd_start(&stream_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(stream_httpd, &index_uri);
  }
}
```

Figura 90. Función startCameraServer.

En la función **setup()** se configura la velocidad de baja a 115200 baudios, se importan las configuraciones de los pines según el modelo de la ESP32-CAM utilizada y la calidad de la transmisión. Luego en la misma función **setup()** se inicializa la cámara y se ejecuta la función para conectar a la wifi.. Por último en la función **loop()** se hace un delay de 1 segundo.

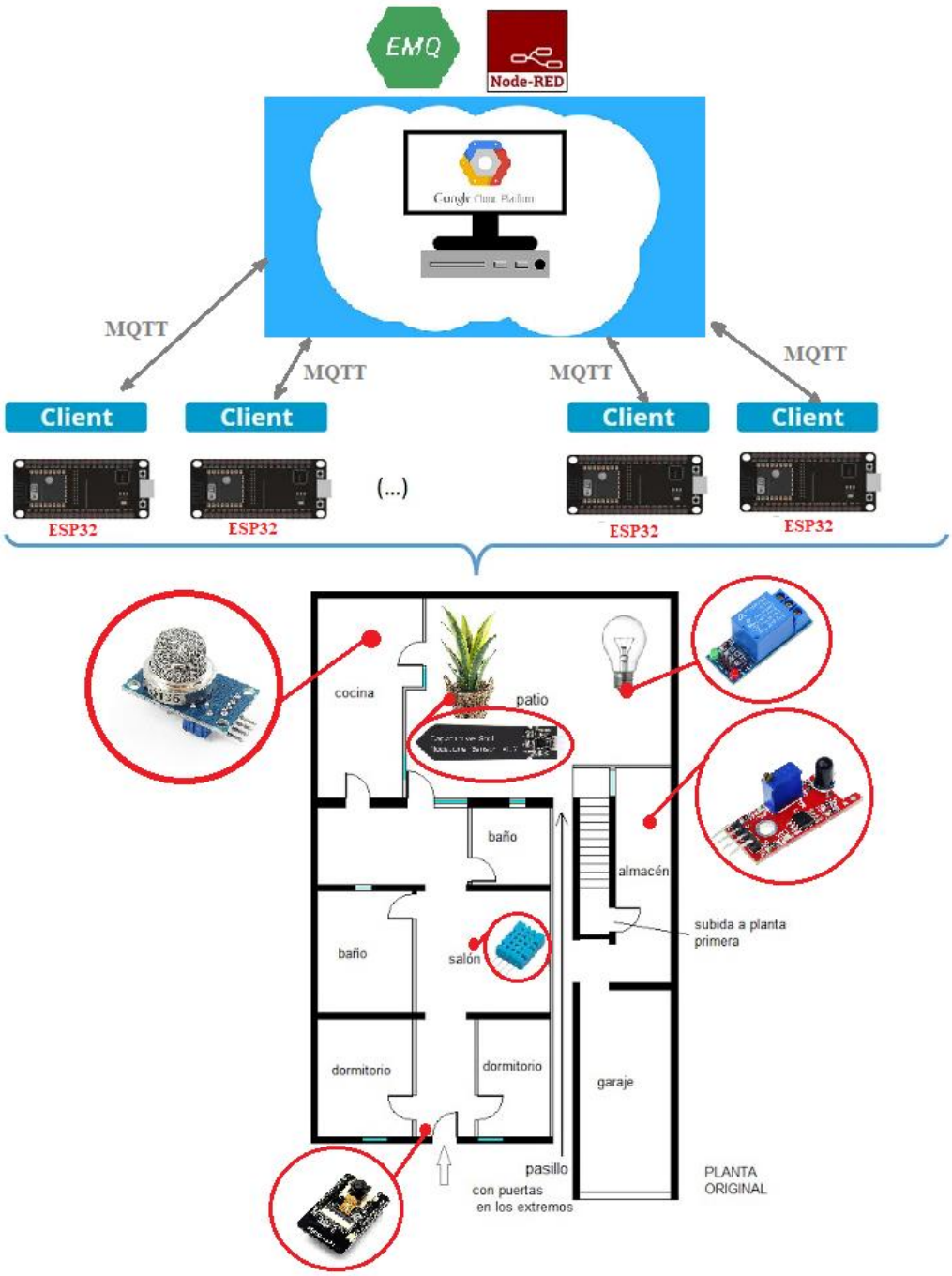
```
 //********************************//
//Inicialización de la cámara
//********************************//
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

//********************************//
// Función de conexión a la wifi
//********************************//
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  Serial.print("Camera Stream Ready! Go to: http://");
  Serial.print(WiFi.localIP());

  // Start streaming web server
  startCameraServer();
}
```

Figura 91. Inicialización de la cámara y función de conexión a la wifi.

3.6 Disposición de los sensores dentro de la casa

En el siguiente diagrama se intenta ejemplificar una de las posibles ubicaciones que se le podrían dar a los sensores en las viviendas. Estos están dispuestos con la idea que nos aporten información relevante de puntos clave en el hogar o que hagan funciones que mejoren la estancia en los mismos.

Este proyecto abre las puertas para la incorporación de muchas más opciones que ayuden a complementar las ya implementadas. En el próximo capítulo se detallarán los futuros proyectos, en aras de mejorar este prototipo inicial.



Capítulo 4. Conclusiones

Al comienzo de este proyecto se proponía el desarrollo de un sistema capaz de monitorizar diferentes parámetros físicos y de realizar acciones relevantes en el hogar. Tras la finalización del trabajo se ha conseguido disponer de las siguientes características:

- El objetivo principal planteado en el **Capítulo 1** consistía en conseguir un sistema barato, escalable y de fácil implementación. Esto se ha conseguido puesto que el presupuesto dispuesto para la compra del controlador ESP32, los 4 sensores con sus complementos y el kit electrónico para hacer las pruebas no supera los 65€.
- Se puede acceder de manera muy sencilla a la aplicación web en Node-RED desde cualquier dispositivo con conexión a internet.
- Es fácilmente escalable ya que se puede ampliar el número de nodos sensores con el único límite de las direcciones IPs disponibles en la red, puesto que se puede reutilizar el mismo código para programar todos los controladores.
- Después de haber testado los servicios de Google Cloud Platform, se ha comprobado que se puede instalar todo el sistema con relativa facilidad en un servidor externo propio siempre que preste alta fiabilidad y disponibilidad.

Se puede afirmar que el sistema ofrece una gran versatilidad, ya que su funcionamiento es fácilmente extrapolable a otros ecosistemas como oficinas, escuelas, hospitales o cualquier otro centro en el cual se esté interesado instalar un sistema de monitorización.

4.1 Posibles mejoras y objetivos futuros

El presente trabajo deja riendas sueltas al ingenio y a la creatividad de todo aquel interesado en el mundo del IoT, dejando la posibilidad para incorporar nuevas funcionalidades y tecnologías que ayuden a complementar la propuesta inicial del proyecto.

Una de las mejoras a incorporar es la implementación de un sistema de almacenamiento en bases de datos, de manera que podamos tener un histórico de las lecturas realizadas por los sensores. La implementación de un sistema de alertas vía correo o SMS para notificar a los usuarios de ciertos valores que no entran dentro de unos parámetros seguros, daría capacidad de gestión frente a ciertas circunstancias de riesgo. La incorporación de baterías de respaldo permitiría al sistema estar protegido contra posibles cortes de luz.

Otro de los objetivos marcados a corto plazo es la instalación y configuración de diferentes sensores para añadir más funcionalidad a la aplicación. Se pretende incorporar un sensor de proximidad, sensores para controlar la apertura de ventanas y sensores de movimiento. Además, se plantean ideas más ambiciosas como la construcción de un sistema de reconocimiento de matrículas y modelos de automóviles para automatizar la apertura de las puertas de garajes, integrar la aplicación con un asistente por voz e incorporar un sistema de alimentación a los ESP32 mediante paneles solares. Finalmente, para aumentar la robustez y eficacia del sistema, se incorpora una capa de seguridad en el envío de los datos por MQTT.

Capítulo 5. Bibliografía

[1] Web de Steve's Internet Guide. Disponible en http://www.steves-internet-guide.com/mqtt/ (accedida el 04 de mayo de 2020).

[2] Web de HiveMQ. Disponible en https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/ (accedida el 04 de mayo de 2020).

[3] Web de HiveMQ. Disponible en https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/ (accedida el 05 de mayo de 2020).

[4] Web de HiveMQ. Disponible en https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/ (accedida el 05 de mayo de 2020).

[5] Web de HiveMQ. Disponible en https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/ (accedida el 06 de mayo de 2020).

[6] Web de HiveMQ. Disponible en https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages/ (accedida el 06 de mayo de 2020).

[7] Web de HiveMQ. Disponible en https://www.hivemq.com/blog/introducing-the-mqtt-security-fundamentals/ (accedida el 8 de mayo de 2020).

[8] Web de AllDataSheet.com. Disponible en https://www.az-delivery.de/products/esp32-developmentboard?_pos=2&_sid=0f61b84dc&_ss=r (accedida el 9 de mayo de 2020).

[9] Web de cloud.google.com . Disponible en https://cloud.google.com/products?hl=es (accedida el 23 de agosto de 2020).

[10] Tutorial de Ióticos en youtube.com . Disponible en https://youtu.be/42ksFrj1CyQ (accedida el 13 de marzo de 2020).

[11] Tutorial de Ióticos en youtube.com . Disponible en https://youtu.be/e8uU3LyI_n8 (accedida el 13 de marzo de 2020).

[12] Web de EMQX. Disponible en https://docs.emqx.io/broker/latest/en/getting-started/dashboard.html (accedida el 23 de agosto de 2020).

Capítulo 6. Anexos

En este capítulo se pretende mostrar cómo se vería la interfaz gráfica con los sensores en funcionamiento y como guardan relación con los valores mostrados por la consola del IDE de Arduino.

Interfaz de la opción "Control"

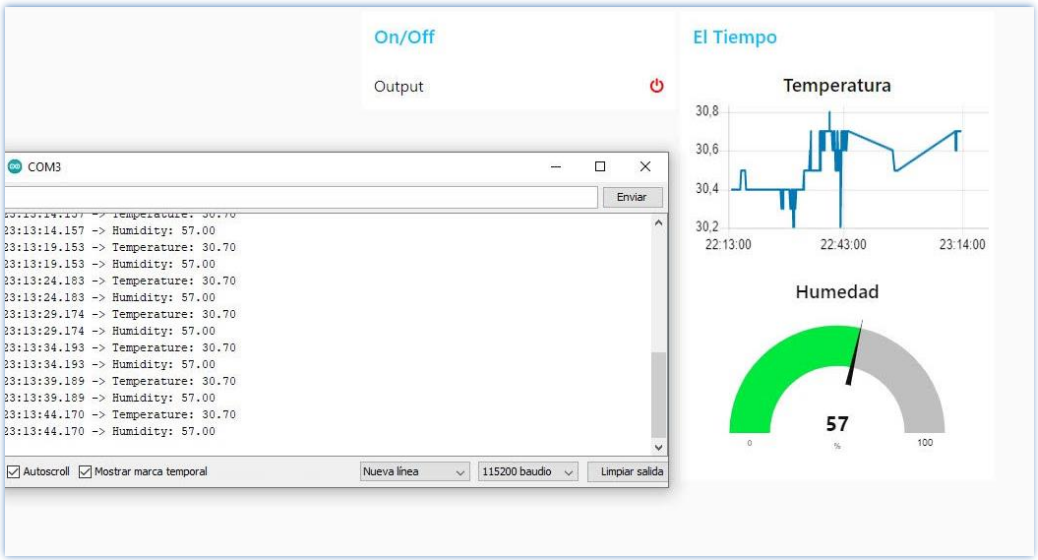


Figura 92. Lectura de valores de humedad y temperatura sensor DTH11

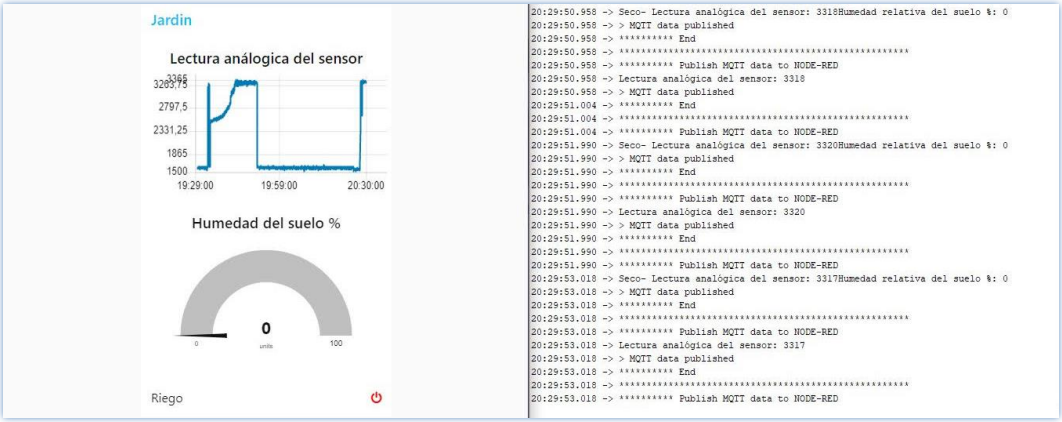


Figura 93. Lectura de la humedad relativa del suelo del sensor Capacitive Soil.

Interfaz de la opción “Seguridad”

Figura 94. Interfaz de la opción “Seguridad” con la cámara de vigilancia y el sensor MQ135 activos.