Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

DESARROLLO DE PRÁCTICAS CON ENTORNOS VIRTUALES EN EL ÁMBITO DE LAS TELECOMUNICACIONES

Alumno: **Zenobia Milla Herrero**

Tutor: Prof. D. Sebastián García Galán
Prof. D. José Enrique Muñoz Expósito
Depto.: Ingeniería de Telecomunicación

Noviembre, 2023

Universidad de Jaén

Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

Trabajo Fin de Grado

DESARROLLO DE PRÁCTICAS CON ENTORNOS VIRTUALES EN EL ÁMBITO DE LAS TELECOMUNICACIONES

Alumno: Zenobia Milla Herrero

Tutores: Dr. Sebastián García Galán

José Enrique Muñoz Expósito

Depto.: Ingeniería de Telecomunicación.

Firma Alumno:

ZENOBIA MILLA HERRERO 77336245X

Firma Tutor:

Firma Tutor:

MUÑOZ EXPOSITO JOSE ENRIQUE - 27386513E

Firmado digitalmente por MUÑOZ EXPOSITO JOSE ENRIQUE - 27386513E
Fecha: 2023.11.03 21:40:40 +01'00'

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. RESUMEN

1.1 Resumen

En el presente Trabajo Fin de Grado (TFG) se han mostrado las posibilidades de desarrollo de aplicaciones que ofrecen los ordenadores con placa reducida y de bajo coste, realizando una serie de experimentos que pueden ser interesantes para prácticas en el ámbito de la Ingeniería de Telecomunicaciones. Se ha trabajado en entornos virtuales con los siguientes dispositivos: ESP32, Arduino UNO y Raspberry PI.

Para la virtualización de los servicios ofrecidos se han utilizado contenedores Docker. Se han creado tantos contenedores, como servicios se han ofrecido. Los servicios son los siguiente: un gestor de bases de datos (MySQL), un administrador de base de datos (phpMyAdmin), una herramienta para desarrollar aplicaciones basada en programación de flujo (Node-RED), un editor de programación visual basado en bloques (BlocklyDuino) y un servidor utilizado como proxy inverso (Nginx). Se dispone de un servicio adicional, una pasarela Zigbee2MQTT, que se ha utilizado en una de las actividades prácticas que se han presentado en este TFG, para el caso de tener el despliegue de contenderos Docker en un PC o Raspberry PI físico del laboratorio.

Uno de los servicios desplegados es una extensión de un proyecto llamado BlocklyDuino, que utiliza el concepto de programación basada en bloques para crear aplicaciones que se ejecutan en microcontroladores Arduino y ESP32. Algunas de estas funcionalidades son: nuevos bloques, un monitor que permita ver la información enviada al puerto serie y una pestaña de registro que controle y muestre el proceso de carga y compilación del proyecto en un dispositivo físico. También se ha modificado el Arduino Create Agent, necesario para la carga del programa en el dispositivo físico.

En el capítulo 5 (Resultados y discusión), se han detallado nueve actividades prácticas, que ilustran como estas herramientas y servicios, facilitan el aprendizaje de conocimientos relacionados con las Telecomunicaciones.

1.2 Abstract

The possibilities of developing applications offered by low-cost, small-board computers have been shown, focusing on a series of experiments that may be interesting in the field of Telecommunications Engineering. It has been worked in virtual environments with the following devices: ESP32, Arduino UNO and Raspberry PI.

Docker containers have been used to virtualize the services offered. As many containers have been created as services have been offered. The services are the following: a database engine (MySQL), a database administrator (phpMyAdmin), a tool for developing applications based on flow programming (Node-RED), a block-based visual programming editor (BlocklyDuino) and a server used as a reverse proxy (Nginx). An additional service is available, a Zigbee2MQTT gateway, which has been used in one of the practical activities that have been presented in this Final Degree Project (TFG).

One of the services deployed is an extension of a project called BlocklyDuino, which uses the concept of block-based programming to create applications that run on Arduino and ESP32 microcontrollers. Some of these functionalities are: new blocks, a monitor that allows you to see the information sent to the serial port and a registration tab that controls and shows the project loading and compilation process on a physical device. The Arduino Create Agent has also been modified, necessary for loading the program on the physical device.

In chapter 5 (Results and discussion), nine practical activities have been detailed, which illustrate how these tools and services facilitate the learning of knowledge related to Telecommunications.

2. INTRODUCCIÓN

En este apartado se realiza una introducción al proyecto realizado, presentando la motivación que ha originado su realización, antecedentes y estructura del documento donde se hace un breve resumen de los distintos capítulos de la memoria.

2.1 Justificación y contexto

Dentro de los sectores existentes en la sociedad, el de las telecomunicaciones es uno de los más importantes y contribuye al desarrollo económico, social y en la mejora de la calidad de vida de la población [1]. La formación de profesionales en esta área debe tener una especial atención, así como, las acciones que la faciliten deben ser tenidas en cuenta.

El desarrollo de prácticas con entornos virtuales en el ámbito de las telecomunicaciones es una estrategia que facilita las tareas de formación y también de divulgación de conocimientos relacionados con las telecomunicaciones [2]. Los entornos virtuales permiten simular situaciones y escenarios reales de manera segura y controlada, facilitando el aprendizaje y experimentación sin que se vean afectados sistemas reales.
Algunos ámbitos donde se desarrollan prácticas con entornos virtuales son:

- Simulación de redes de telecomunicaciones. El uso de software de simulación como son GNS3 [3] y Packet Tracer [4] permiten recrear topologías de redes, incluyendo encaminadores, conmutadores, servidores y dispositivos de telecomunicaciones. Se pueden realizar pruebas de rendimiento, midiendo el ancho de banda, latencia y pérdida de paquetes (entre otros).
- Seguridad en redes. Se pueden crear escenarios en los que identificar vulnerabilidades, implementado medidas de seguridad y protección de redes, todo ello para evitar ataques potenciales. Un ejemplo es Cisco DevNet Sandboxes [5]: consiste en un programa de difusión gratuita de la tecnología de Cisco, donde se pone a disposición de los desarrolladores e ingenieros un conjunto de laboratorios virtuales llamados *Sandboxes*. Existen laboratorios orientados a seguridad en red que pueden utilizarse para desarrollo, aprendizaje de configuración, capacitación y encuentros de expertos en TIC para debatir nuevas vulnerabilidades e implementar medidas de seguridad de forma colaborativa (*Hackathon*).
- VoIP (*Voice Over Internet* – Voz sobre el protocolo de internet). A través de herramientas de simulación se realizan prácticas para entender los protocolos, flujos de datos y la calidad de servicio en comunicaciones de voz. Un ejemplo son

los *Sandboxes* ofrecidos por Cisco DevNet, donde se puede realizar experiencias con el protocolo SIP (Session Initiation Protocol – Protocolo de iniciación de sesión) [6].

- Redes definidas por software (*software defined networking* - SDN). Utilizando entornos de simulación como Mininet [7] se pueden programar y automatizar tareas de gestión de red. Mininet es una herramienta para redes definidas por software. Es un simulador de red con capacidad de visualizar los conmutadores y aplicaciones SDN en un entorno virtualizado. También se utiliza para probar dispositivos SDN que utilizan protocolos OpenFlow.
- Virtualización de funciones de red (NFV – *network function virtualization*). Se pueden realizar configuraciones y despliegue de funciones de red, tales como firewalls, balanceadores de carga o encaminadores. Un ejemplo es Open Source MANagement and Orchestration (OSM MANO) [8], comunidad de código abierto alojada en ETSI (*European Telecommunications Standards Institute* - Instituto Europeo de Normas de Telecomunicaciones).
- Desarrollo de aplicaciones para dispositivos móviles. Con el uso de emuladores de dispositivos móviles se pueden probar distintas aplicaciones. Por ejemplo, para aplicaciones creadas en Java o Kotlin se puede utilizar Android Studio [9] y para aplicaciones híbridas nativas, entornos como Expo React [10].
- Internet de las cosas (*Internet of Things* - IoT). Uno de los ámbitos donde las telecomunicaciones tienen gran aplicación es el de 'Internet de las cosas' o 'Internet de los objetos', concepto asociado a la conectividad entre el usuario y objetos cotidianos o entre distintos dispositivos sin la intervención directa de un usuario. La comunicación se realiza entre los distintos dispositivos y los datos pueden ser recogidos y almacenados para realizar extracción de conocimiento y toma de decisiones. Gracias a este conocimiento, se pueden optimizar procedimientos industriales, controlando puntos críticos del sistema de fabricación, aumentando el ahorro en consumo energético, tiempo y en general mejorando la gestión empresarial. Muchos de estos dispositivos son los conocidos como microcontroladores de bajo coste, que, a través de distintas tecnologías como Wifi, Bluetooth o Zigbee permiten el proceso de comunicación.
- Simulación de microcontroladores y ordenadores de placa reducida. Aplicaciones como Wokwi [11] permiten crear y comprobar dispositivos que incorporan microcontroladores de la familia ESP32, Arduino y dispositivos Raspberry Pi Pico.

Con VirtualBox [12] se puede ejecutar una máquina virtual que simula una Raspberry Pi.

El uso de entornos virtuales en el aprendizaje de telecomunicaciones tiene asociado un conjunto de ventajas, siendo algunas de ellas:

- Permiten al estudiantado cometer errores sin afectar sistemas en producción o configuración preexistente en los equipos: esto fomenta la experimentación y el aprendizaje activo, eliminando el miedo al error.
- Flexibilidad a la hora del acceso. Los estudiantes pueden acceder a los entornos virtuales desde cualquier lugar y en cualquier momento, facilitando el autoaprendizaje, ya que se pueden imponer su propio ritmo.
- Económicas. Se reduce la necesidad de equipos físicos costosos y espacios dedicados para laboratorios: esto permite un uso más eficiente de los recursos.
- Relacionado con la anterior, se facilita la utilización de tecnologías novedosas y recientes, al poder acceder a ellas a través de dispositivos virtualizados, sin necesidad de realizar un gran desembolso económico.
- Fomenta el trabajo en equipo. Las plataformas virtualizadas presentan facilidades donde se puede trabajar en proyectos abiertos y colaborativos, facilitando la participación en equipo.

El objetivo de este TFG ha sido mostrar las posibilidades de desarrollo de aplicaciones que ofrecen los ordenadores con placa reducida y de bajo coste, orientándose a una serie de experimentos en el ámbito de la Ingeniería de Telecomunicación. Por ello, se considerado importante utilizar dispositivos como microcontroladores ESP32, Arduino UNO WiFi Rev2 y Raspberry Pi, así como, herramientas que poseen elementos de programación visual como son la programación basada en bloques y programación basada en flujos, que permitan extender esta difusión y conocimiento a estudios o niveles educativos extra universitarios. Además, se ha hecho uso de un simulador en línea y gratuito como es Wokwi, ya que se ha considerado una herramienta muy útil por las siguientes razones:

- Económicas. El estudiante puede realizar sus actividades sin necesidad de un desembolso económico.
- Accesibilidad. Solo se necesita un navegador para poder acceder al simulador, no es necesario instalar una aplicación: facilita su acceso desde cualquier ordenador.
- Existencia de una comunidad activa. Existen múltiples proyectos de acceso público creados, que pueden servir de guía en el proceso de aprendizaje.

- Facilidad de revisión. El docente o tutor de la actividad puede acceder desde cualquier lugar y supervisar el proyecto.
- Rapidez en el desarrollo. Es más rápido crear un proyecto en el simulador y una vez comprobado su correcto funcionamiento, exportarlo al dispositivo físico.

Las distintas actividades presentadas en el capítulo 5, hacen uso de distintas plataformas y soluciones de virtualización.

2.2 Antecedentes y estado del arte

En nuestros días, se han implementado múltiples aplicaciones y entornos de desarrollo que permiten crear y depurar aplicaciones en dispositivos basados en microcontroladores.

Para el ámbito de placas compatibles con Arduino, existe un entorno propio de desarrollo integrado (*Integrated Development Enviroment* - IDE), conocido como Arduino IDE [13]. Proporciona una plataforma de desarrollo para escribir y cargar fácilmente los programas en placas Arduino y otros microcontroladores compatibles con el ecosistema Arduino. Gracias a esta posibilidad, se ha utilizado en microcontroladores de la familia ESP32. Para simplificar el desarrollo de proyectos basados en estas placas, una opción muy interesante es Arduino Cloud [14] ya que está diseñado para ser fácil de usar. Arduino Cloud es una plataforma que ofrece un servicio online con almacenamiento en la nube, facilitando el acceso a los proyectos desde cualquier ubicación. Los proyectos se crean y se editan con su editor web (Arduino Web Editor) [15] y únicamente se necesita instalar el agente Arduino Create Agent [16] en nuestro ordenador, para poder cargar el programa ejecutable en el dispositivo físico. Arduino Cloud, también, dispone del servicio IoT Cloud [17] que permite la conexión entre dispositivos y a internet, de manera sencilla y segura.

Existen programas que permiten la programación gráfica de dispositivos Arduino. Una aplicación muy extendida es ArduinoBlocks [18]. Se trata de una plataforma en línea que permite realizar proyectos basados en programación de bloques. Gracias a una extensión llamada ‘ArduinoBlocks-connector’, se transfiere el código que se ha generado con los bloques a la placa Arduino conectada al ordenador (para su carga y ejecución).

Otra aplicación gráfica es BlocklyDuino [19] que se basa en el proyecto Blockly [20] para la programación de microcontroladores. Blockly es una biblioteca de funciones

creadas en lenguaje JavaScript que permite a los usuarios crear código utilizando bloques de programación en lugar de escribir código de forma tradicional. BlocklyDuino se ejecuta dentro de un navegador web, lo que significa, que no es necesario descargar ni instalar nada en nuestro ordenador para utilizarlo. En su interfaz gráfica permite al usuario hacer uso de bloques para programar los microcontroladores. Además, permite extender los bloques, añadiendo nuevas funcionalidades y posibilidades. También dispone de un área donde aparece el código generado a partir de los bloques que se hayan utilizado en la interfaz gráfica.

En las distintas actividades prácticas que se presentan en el capítulo 5 ha sido necesario utilizar un gestor de base de datos, un panel que permita administrarlas y herramientas de programación visual, ya sean de bloques (para generar aplicaciones Arduino) o bien de flujo basadas en Node-RED [21] (para crear aplicaciones de servidor basadas en lenguaje JavaScript), también se ha utilizado el simulador Wokwi, por todas sus ventajas comentadas en el apartado anterior.

Para facilitar el despliegue de todas ellas en un sistema multiplataforma, se ha decidido utilizar la virtualización, concretamente la virtualización software proporcionada por el sistema de contenedores Docker [22]. Los contenedores se despliegan en un único nodo, que puede ser una máquina virtual, un ordenador o una Raspberry Pi. Los motivos de utilizar Docker son: permiten la virtualización de los servicios utilizados en este TFG y es más sencillo si se compara con otras plataformas, por ejemplo, Kubernetes [23].

2.3 Estructura del documento

Este documento está estructurado en los siguientes capítulos:

- **Capítulo 1.** RESUMEN: Breve resumen recapitulando las características principales del proyecto.
- **Capítulo 2.** INTRODUCCIÓN: Motivación y contexto del TFG.
- **Capítulo 3.** OBJETIVOS: Se detallan los objetivos del TFG.
- **Capítulo 4.** MATERIALES Y MÉTODOS: Explicación de las tecnologías utilizadas y desarrollo de las soluciones adoptadas en este TFG.
- **Capítulo 5.** RESULTADOS Y DISCUSIÓN: Se proponen distintas prácticas y presentan las soluciones comentadas.
- **Capítulo 6.** CONCLUSIONES: Se enumeran las conclusiones y resultados obtenidos.
- **Capítulo 7.** LÍNEAS FUTURAS: Se presentan posibles ampliaciones al TFG realizado y cómo se mejoraría.
- **Capítulo 8.** BREVE ESTUDIO ECONÓMICO: Un estudio simplificado del coste de realización del TFG realizado.
- **Capítulo 9.** ANEXOS DE LA MEMORIA: Documentos adicionales que complementan la información presentada en el capítulo de materiales y métodos.
- **Capítulo 10.** BIBLIOGRAFÍA: Referencias a los documentos que se han utilizado en el desarrollo del TFG.

3. OBJETIVOS

El objetivo de este TFG es mostrar las posibilidades de desarrollo de aplicaciones que ofrecen los ordenadores con placa reducida y de bajo coste, orientándose a una serie de experimentos que pueden ser interesantes para prácticas en el ámbito de la Ingeniería de Telecomunicaciones. Se ha trabajado con los siguientes dispositivos: ESP32, Arduino UNO WiFi Rev2 y Raspberry PI.

En este documento, en el capítulo 5, se han incluido diversas propuestas de actividades prácticas y sus soluciones comentadas. Los elementos físicos y de simulación empleados, serán citados y justificados.

4. MATERIALES Y MÉTODOS

A continuación, se realiza una descripción del trabajo realizado, profundizando en los detalles del mismo y presentando las tecnologías aplicadas.

En el apartado 4.1 se muestra un esquema del conjunto de servicios y aplicaciones utilizadas y cómo interactúan entre ellas.

En el apartado 4.2 se realiza un breve resumen de las distintas tecnologías aplicadas en el desarrollo de este TFG.

Para finalizar, en el apartado 4.3 se presenta la implementación de las aplicaciones realizadas.

4.1 Esquema general del sistema

Para demostrar las capacidades de desarrollo de aplicaciones en ordenadores con placa reducida y crear una colección de experimentos, se hace uso de un conjunto de servicios y aplicaciones que han de ser desplegados y configurados para lograr comunicación e interacción entre ellos. En la Figura 4.1 muestra un esquema general del sistema donde se ha marcado con una etiqueta el tipo de servicio o aplicación:

- S: Servicio desplegado en un contenedor de software (Docker).
- C: Aplicación utilizada como interfaz de usuario.
- A: Aplicación que se comporta como cliente y servidor.

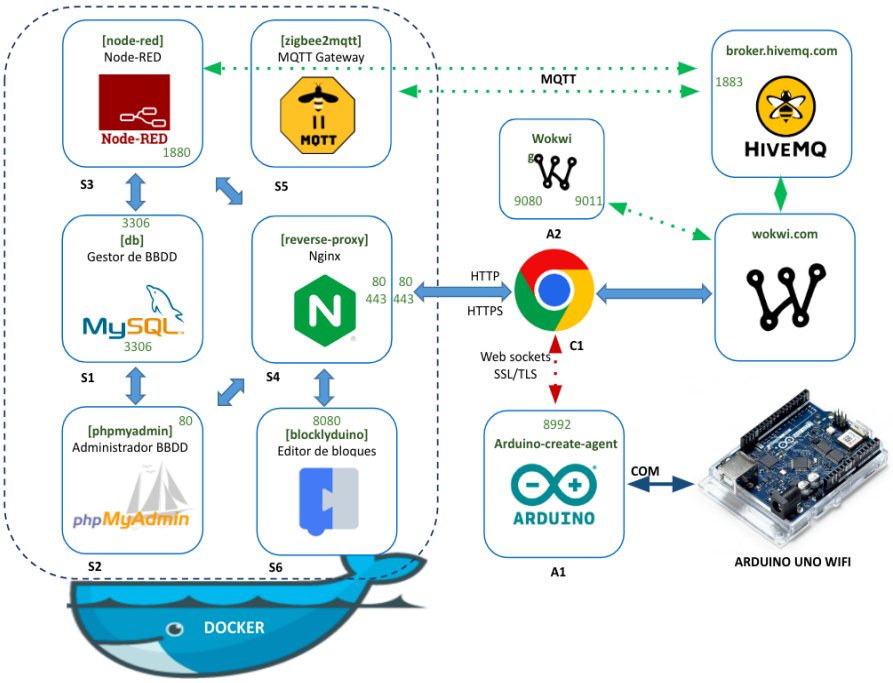


Figura 4.1. Esquema del sistema.

En la figura anterior aparecen los siguientes elementos:

- Un sistema de gestión de bases de datos relacional MySQL (**S1**). Permite crear, actualizar, administrar e interactuar con una base de datos relacionales. Utiliza lenguaje SQL para llevar a cabo las tareas de consulta o manipulación de los datos, para describir una base de datos, para definir su estructura y para crear objetos de la tabla.
- Una herramienta de administración de gestores de bases de datos MySQL. Se trata de phpMyAdmin [24], se ejecuta como un servicio Docker (**S2**). Ofrece una interfaz de usuario para la creación y administración de bases de datos.
- Una herramienta visual de desarrollo de aplicaciones basada en programación de flujo, Node-RED (**S3**). Con él, se crean aplicaciones para manipular información en bases de datos y publicar mensajes en un tema (*topic*) del bróker MQTT HiveMQ [25]. Va a relacionarse con el gestor de bases de datos MySQL y con el bróker MQTT.
- Un servidor Nginx (**S4**) utilizado como proxy inverso [26] para acceder a distintos servicios. Recibe peticiones del cliente y redirecciona a Node-RED, phpMyAdmin y BlocklyDuino.
- Una pasarela software Zigbee2MQTT [27] (**S5**), para la conexión de dispositivos que utilizan el protocolo Zigbee con otros que usan el protocolo MQTT.
- Un editor de programación visual basado en bloques para Arduino y dispositivos de la familia ESP32, desplegado como un servicio en un contenedor (**S6**). Se trata de una extensión realizada en este TFG a partir del proyecto BlocklyDuino (que a su vez se basa en Blockly). Interactúa con las placas de desarrollo y el simulador Wokwi.
- Arduino Create Agent (**A1**). Es una aplicación que se ejecuta en el equipo local donde se conecta a la placa de desarrollo Arduino y permite cargar programas a través del cable USB (puerto COM) desde una aplicación que se ejecuta en un navegador web. En este TFG se ha utilizado una versión extendida.
- Simulador online Wokwi, para dispositivos Arduino, basados en ESP32 y Raspberry Pi Pico. Interactúa con BlocklyDuino.
- Una aplicación de pasarela a la red privada IoT Wokwi (**A2**) [28]. Permite extender las posibilidades de las aplicaciones que se ejecutan en el simulador Wokwi, como recibir peticiones desde el exterior de la red Wokwi de los servicios web simulados.
- Un bróker MQTT público del fabricante HiveMQ. Interactúa con: Node-RED, Zigbee2MQTT y el simulador Wokwi.

- Un navegador web (**C1**) instalado en el equipo local y que sirve como interfaz de usuario a la hora de acceder a distintos servicios. Desde él se accede a los servicios phpMyAdmin, Node-RED y al simulador Wokwi. También se utiliza para acceder a la interfaz web de usuario que proporciona el agente de Arduino.

Aunque existen muchos elementos, su despliegue es muy sencillo:

- Los servicios virtualizados se despliegan utilizando el comando:

```
docker-compose up --build -d
```

- Se conecta el adaptador Texas Instruments CC2531 (para convertir Zigbee a MQTT) a un puerto del dispositivo donde se han desplegado los servicios virtualizados.
- En el ordenador local, únicamente hay que copiar dos ficheros y ejecutarlos:
    - arduino-create-agent.
    - wokwigw.
- Se conecta la placa de desarrollo Arduino al puerto COM del ordenador.

Los distintos servicios y aplicaciones pueden ejecutarse distribuidos entre distintos ordenadores. El criterio elegido para realizar su distribución dependerá de los recursos disponibles en cada aula educativa o laboratorio docente. Algunas situaciones que se pueden dar son:

- Una placa de desarrollo Arduino por alumno, una por grupo de N alumnos o una única para toda el aula.
- La posibilidad o no de disponer de un ordenador adicional para desplegar los servicios virtualizados.
- Disponibilidad de presupuesto para desplegar los servicios en una máquina virtual utilizando un proveedor de infraestructura Cloud (como pueden ser AWS, Azure, GCP o Clouding.io).

Estas situaciones tan distintas, han condicionado que se contemplen los siguientes escenarios para garantizar un correcto funcionamiento según los recursos disponibles:

1. Los servicios virtualizados se despliegan en una Raspberry Pi, un PC o una máquina virtual conectados a la red local del laboratorio (Figura 4.2). En este caso, existen múltiples placas de desarrollo Arduino y cada una de ellas conectada a un ordenador del laboratorio (Equipo local). En cada uno de estos ordenadores se instala un agente Arduino (*Arduino-Create-Agent*). En este escenario, es posible

utilizar la pasarela Zigbee2MQTT, ya que físicamente es posible conectar un adaptador USB dongle Texas Instruments CC2531 y hacer uso de dispositivo Zigbee en el laboratorio.

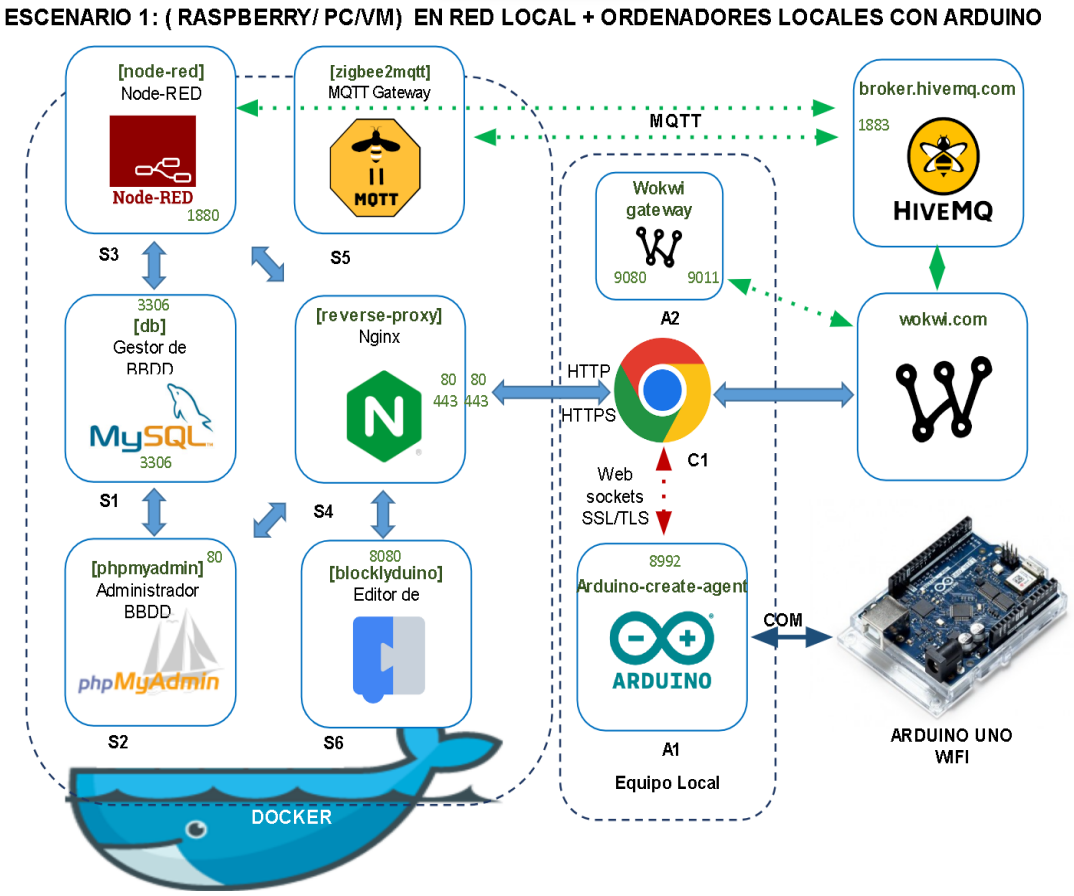


Figura 4.2. Escenario 1.

2. Los servicios virtualizados se despliegan en una máquina virtual en la infraestructura de un proveedor de servicios Cloud (Figura 4.3). Al igual que el escenario 1, existen múltiples placas de desarrollo Arduino y cada una de ellas conectada a un ordenador del laboratorio (Equipo local). En este caso no es posible utilizar el servicio Zigbee2MQTT. Pero este escenario añade una ventaja: permite que los alumnos tengan acceso remoto a los servicios educativos y si disponen de una placa de desarrollo Arduino, instalando el agente Arduino en sus ordenadores domésticos, pueden realizar prácticas o ejercicios propuestos por los docentes en cualquier momento, fomentando el aprendizaje autónomo.

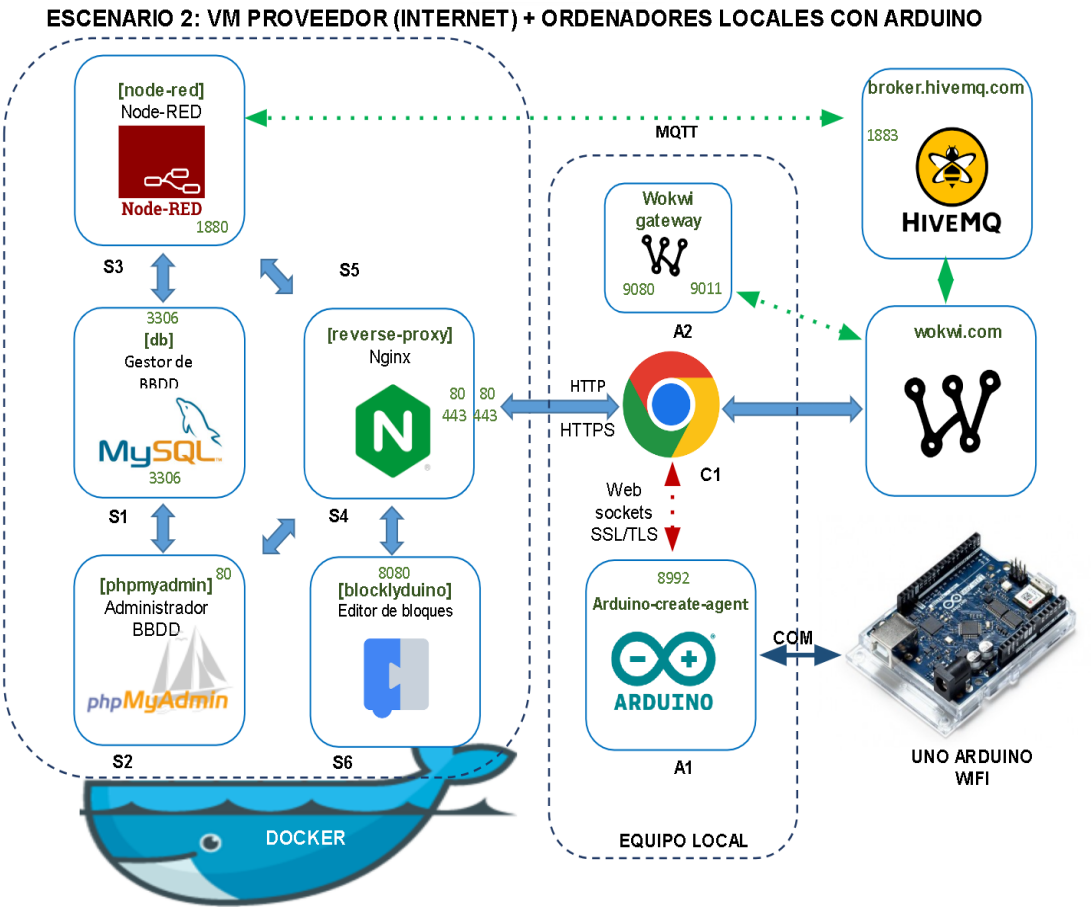


Figura 4.3. Escenario 2.

3. Los servicios se despliegan en una Raspberry o PC o máquina virtual conectados a la red local del laboratorio (Figura 4.4). Existe un número limitado de placas Arduino conectadas a algunos ordenadores de la red local. En esos ordenadores que disponen de placa Arduino (equipos con Arduino) se instalan los agentes Arduino-Create-Agent.

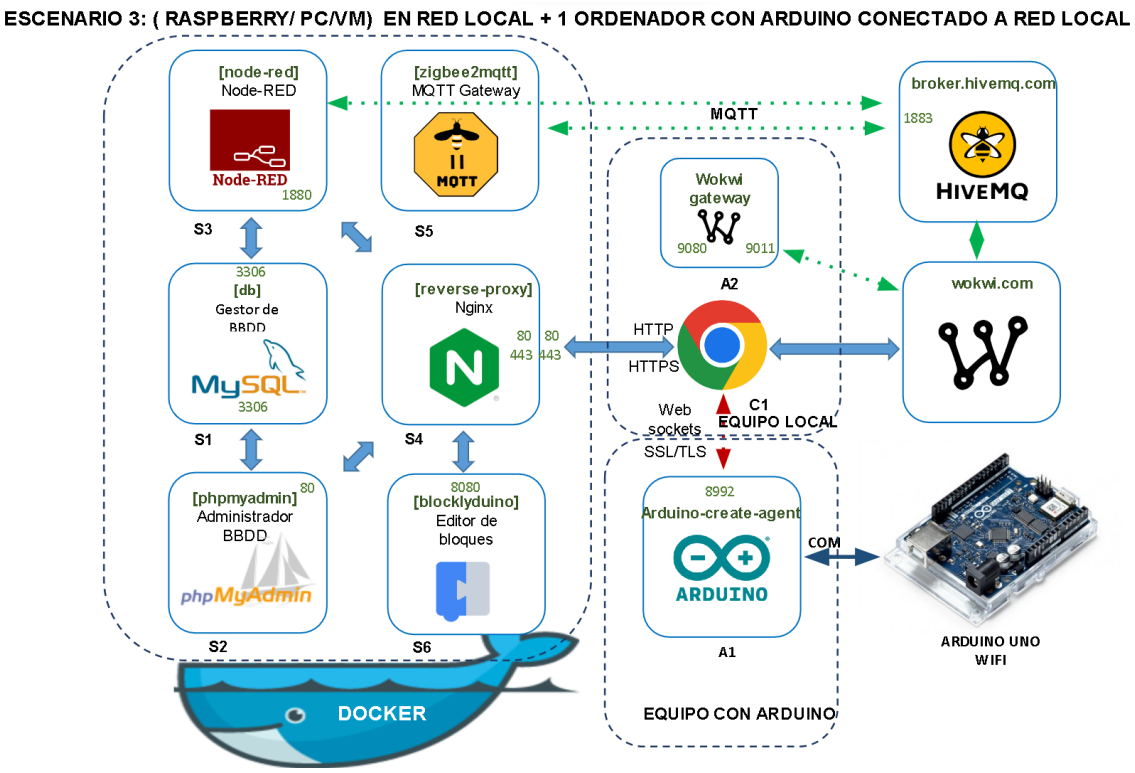


Figura 4.4. Escenario 3.

4. Los servicios se despliegan en una máquina virtual de un proveedor y existe un número limitado de placas Arduino conectadas a algunos ordenadores de la red local. Al igual que el escenario 2, no es posible usar el servicio Zigbee2MQTT. Permite que los alumnos puedan usar los servicios remotamente desde su casa y realizar prácticas propuestas por los docentes.

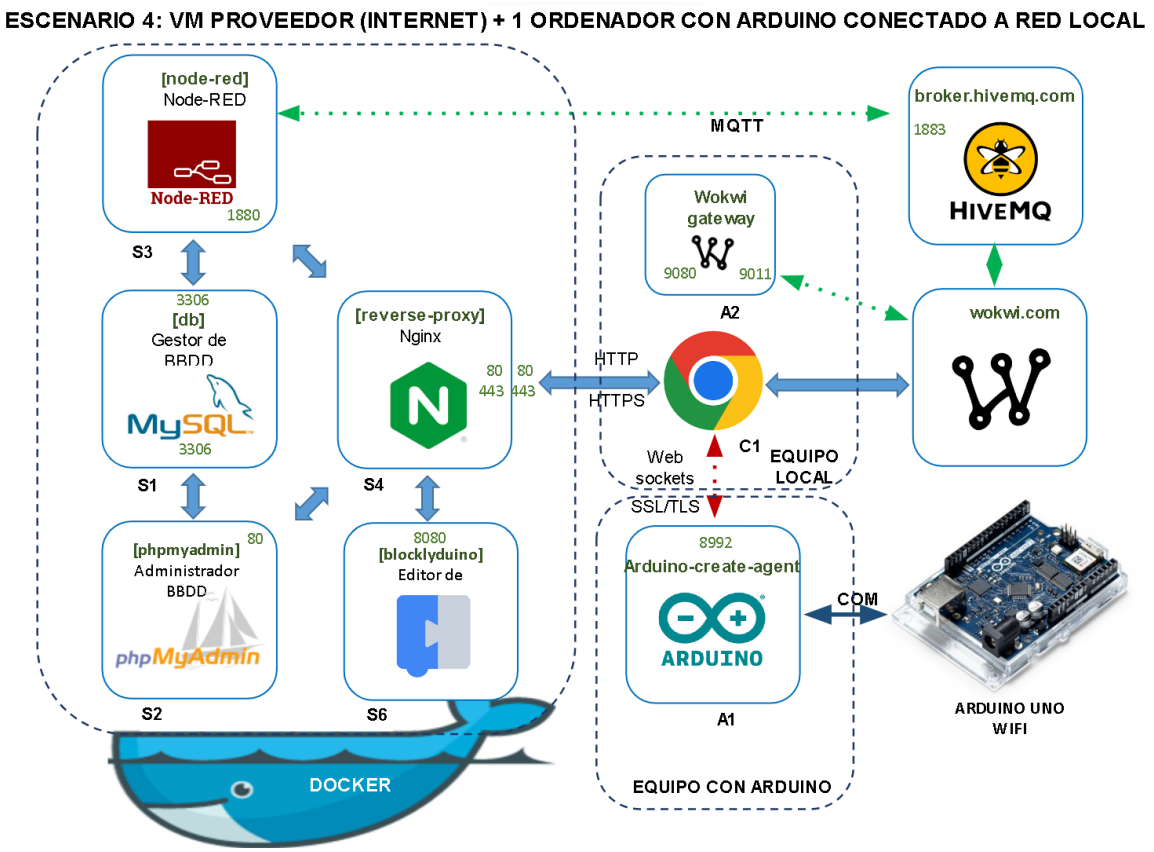


Figura 4.5. Escenario 4.

4.2 Tecnologías, aplicaciones y lenguajes de programación utilizados

En el desarrollo del trabajo se han utilizado distintas aplicaciones, lenguajes de programación y tecnologías. En este apartado, se desarrolla la tecnología de virtualización de contenedores Docker para desplegar servicios. En estos contenedores se ejecuta un servidor utilizado como proxy inverso, un sistema de gestión de bases de datos, una herramienta para administrar las bases de datos, una pasarela Zigbee a MQTT programada en lenguaje JavaScript, la herramienta Node-RED para desarrollar aplicaciones basada en programación de flujo y un editor de programación visual basado en bloques para Arduino desarrollado en lenguaje Python y JavaScript.

4.2.1 Docker y Docker-compose

Los contenedores Docker empaquetan aplicaciones software junto con todos los componentes y dependencias, como bibliotecas, herramientas del sistema y archivos de configuración, en una sola unidad.

Un contenedor Docker es una implementación de un contenedor software, es un entorno aislado de otros, que ejecuta una aplicación y todas sus dependencias. Al encontrarse aislados del ordenador del anfitrión, pueden ejecutarse en distintos entornos, independientemente de la plataforma: sólo es necesario tener instalado un sistema de ejecución de contenedores, por ejemplo, Docker. Se diferencia de las máquinas virtuales en que, las segundas, virtualizan los recursos hardware de un ordenador (CPU, memoria RAM, disco duro, interfaces de red, puertos USB) y un sistema operativo instalado en él. En el caso de los contenedores software comparten el núcleo del sistema anfitrión (recursos físicos) y sólo empaquetan las dependencias y los binarios de la aplicación: esto los hace más eficientes, ya que no hay recursos ociosos sin utilizar.

Docker Compose [29] es una herramienta que se utilizará para ejecutar aplicaciones Docker de varios contenedores. Permite crear un archivo de texto (denominados archivos YAML) donde se especifican los servicios (contenedores), las redes y los volúmenes necesarios. Utilizando comandos se puede iniciar el despliegue de los contenedores y detenerlos.

4.2.2 Servidor Nginx como proxy inverso

Nginx es un servidor web de código abierto muy utilizado como proxy inverso, balanceador de carga HTTP y proxy de correo electrónico para protocolos IMAP, POP3 y SMTP. El modo de funcionamiento de Nginx es asíncrono y controlado por eventos, esto permite el procesamiento de muchas solicitudes al mismo tiempo.

Un proxy inverso es un servidor que se sitúa delante de otros servidores y reenvía las solicitudes del cliente, cuando los clientes envían solicitudes a un servidor destino, el servidor de proxy inverso intercepta esas solicitudes en el perímetro de la red enviando las solicitudes al servidor destino, asegurando que ningún cliente se comunique directamente con un servidor dentro del perímetro. Con un proxy inverso instalado, un servicio web no necesita revelar nunca su dirección IP, esto dificulta que los atacantes usen esa información para atacar utilizando técnicas como DDoS (Distributed Denial of Service - ataque distribuido de denegación de servicio).

Si se ejecuta como servicio en un contenedor Docker, el archivo de configuración principal es: /etc/nginx/nginx.conf. En este archivo se realiza la configuración para que funcione como un proxy inverso.

4.2.3 Blockly y BlocklyDuino

Blockly es un proyecto de Google que muestra fragmentos de código a partir de bloques visuales. El código de los distintos bloques se va agregando creando un código resultante complejo, todo ello realizado de una forma muy sencilla. Está programado en JavaScript y permite producir código fuente para distintos lenguajes de programación (JavaScript, Python, PHP, Lua, Dar, XML, JSON). El editor de bloques se visualiza en un navegador web.

Blockly es muy utilizado en entornos pedagógicos, donde el alumnado se enfrenta a la tarea de programar por primera vez. La representación visual de los fragmentos de código facilita la comprensión del programa que están realizando: a modo de piezas de un rompecabezas los bloques van encajando entre ellos y el resultado se convierte en código.

Para trabajar con Blockly, se abre una categoría y se arrastra el bloque de código deseado al espacio de trabajo, vinculando con otros (Figura 4.6). Para ver el código resultante, se elige la pestaña de lenguaje deseado (Figura 4.7).

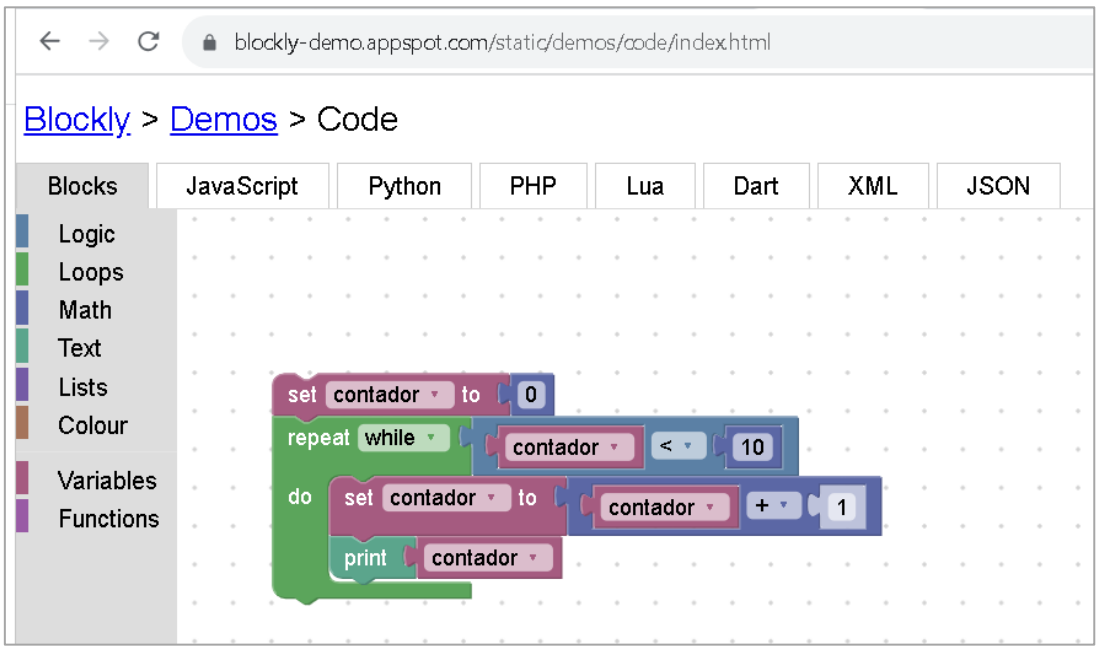


Figura 4.6. Ejemplo de programa con Blockly.

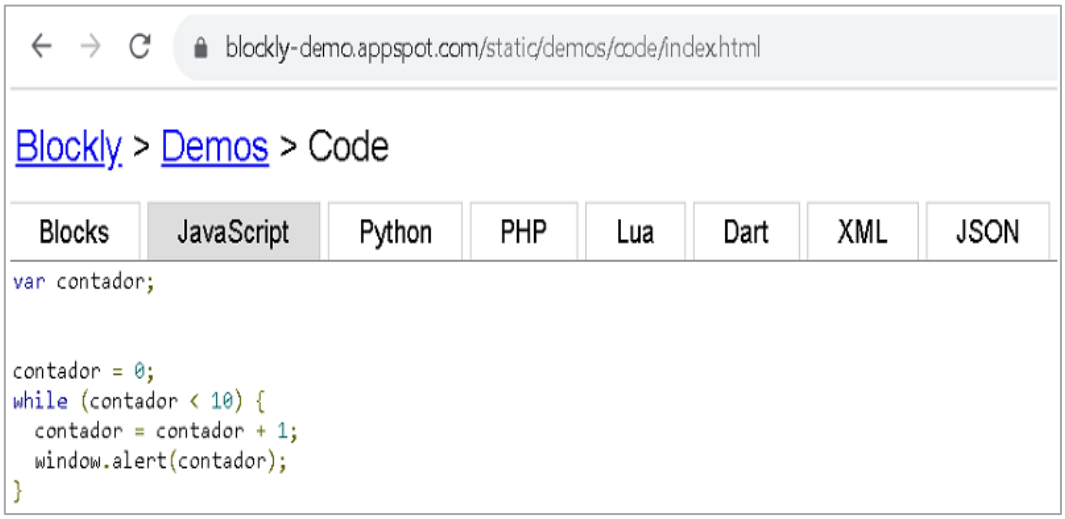


Figura 4.7. Código JavaScript generado.

BlocklyDuino es un editor de programación visual basado en web para Arduino, la interfaz de edición se muestra en la Figura 4.8. Se han extendido las bibliotecas de Blockly para utilizar bloques adaptados a Arduino y generar código que pueda ser ejecutado por Arduino. Esta capacidad de extender las bibliotecas resulta muy atractiva, ya que nos va a permitir incorporar nuevos bloques para conectar el dispositivo Arduino a la red local, para utilizar el protocolo MQTT (disponiendo de funcionalidades muy útiles en el ámbito de IoT), para crear servicios web y otras que serán descritas en el apartado 4.3.

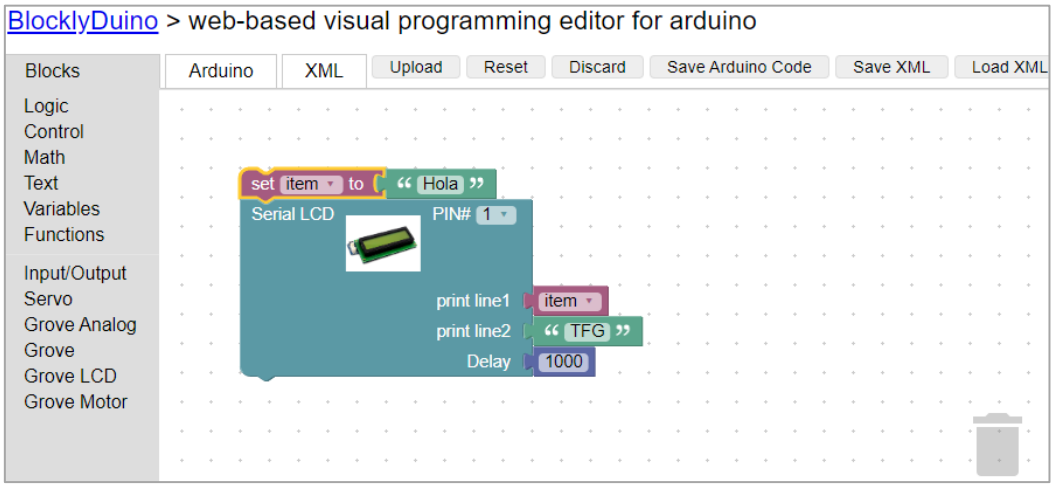


Figura 4.8. Editor BlocklyDuino.

4.2.4 Lenguaje de programación Python

Python es un lenguaje de programación interpretado (no es necesario compilarlo para ejecutar las aplicaciones) que posee estructuras de datos de alto nivel [30]. Permite la programación orientada a objetos y es un lenguaje ideal para secuencias de comandos y desarrollo rápido de aplicaciones.

Utilizar Python tiene asociados los siguientes beneficios:

- Los desarrolladores de aplicaciones entienden fácilmente los programas gracias a su sintaxis.
- Permite una alta productividad, ya que son necesarias menos líneas de código, si lo comparamos con otros lenguajes.
- Cuenta con una gran cantidad de bibliotecas, aplicadas a diferentes áreas de conocimiento.
- Es fácil integrar, a través de llamadas externas, otros programas desarrollados en otros lenguajes de programación (Java, C y C++).
- Es un lenguaje de programación de código abierto, disponible para múltiples plataformas y sistemas operativos.
- Facilita trabajar con aplicaciones orientadas a la inteligencia artificial, *big data*, *machine learning* y *data science*, entre otros.
- Posee una comunidad activa de millones de desarrolladores.
- Existe una gran cantidad de recursos disponibles en Internet: tutoriales, documentación y guías para desarrolladores.

4.2.5 Python Flask

Es un entorno de trabajo para crear aplicaciones web con Python [31] utilizando el patrón Modelo-Vista-Controlador (MVC). El patrón MVC es una manera de crear aplicaciones web donde se separan los datos de la aplicación (Modelo), las interfaces de usuario que se presentan (Vista) y la aplicación que gestiona las peticiones a la aplicación web (Controlador). Proporciona una forma sencilla de crear servicios web y gestionar solicitudes HTTP.

A la hora de instalarlo, es tan sencillo como ejecutar el siguiente comando en la consola (si se desea la versión 2.2.5, que es la utilizada en este TFG):

```
pip install Flask==2.2.5.
```

4.2.6 Lenguaje de programación JavaScript

JavaScript es un lenguaje de programación interpretado (o compilado) [32] [33]. Se utiliza para crear aplicaciones cliente (*frontend)* que se ejecutan en navegadores web y en aplicaciones de servidor (*backend*), utilizando Node.js [34]. El estándar para JavaScript es ECMAScript (ECMA-262) y la especificación de la API (*Application Programming Interfaces* - Interfaz de programación de aplicaciones) para la Internacionalización de ECMAScript (ECMA-402).

Proporciona múltiples funcionalidades, algunas son:

- Manipulación del DOM (Document Object Model - Modelo de objetos del documento), interactuando con los elementos existentes en una página web (documento HTML).
- Manejo de eventos. Permite la ejecución de un código cuando se genera un evento.
- Comunicación con servidores utilizando peticiones AJAX (JavaScript Asíncrono + XML).
- Validación de formularios.
- Generación de animaciones.

4.2.7 Lenguaje de programación Arduino

El lenguaje de programación más utilizado en Arduino está basado en el lenguaje C y C++ [35]. Es una adaptación proveniente de avr-libc que y proporciona de una librería en C para utilizarla con GCC (*GNU Compiler Collection*) en los microcontroladores AVR de Atmel. En este trabajo, se ha utilizado la opción de programación basada en lenguaje C.

Otro lenguaje utilizado es MicroPython [36], una implementación del lenguaje de programación Python diseñado para ejecutarse en microcontroladores y sistemas embebidos. Una ventaja de usar MicroPython es su facilidad de aprendizaje y que tiene una excelente documentación. En este momento, solo algunas placas son compatibles con este lenguaje.

4.2.8 phpMyAdmin

Es una herramienta que facilita la gestión de sus bases de datos sin necesidad de utilizar herramientas en línea de comandos [37]. Es una aplicación web (escrita en lenguaje

PHP) que proporciona una interfaz gráfica de usuario para gestionar y administrar bases de datos MySQL y MariaDB. Permite a los usuarios realizar las siguientes tareas:

- Creación, modificación y eliminación de bases de datos, tablas y atributos.
- Gestión de cuentas de usuario.
- Importación y exportación de datos.
- Ejecución de consultas SQL entre otras.

4.2.9 Sistema de gestión de bases de datos relacional MySQL

Es un sistema de gestión de bases de datos relacionales, utilizado por empresas y desarrolladores. Ayuda a organizar, almacenar, recuperar y gestionar datos. Está diseñado específicamente para gestionar datos en bases de datos MySQL. Permite realizar las siguientes tareas:

- Creación, modificación y eliminación de bases de datos, tablas y atributos.
- Inserción, modificación y eliminación de datos.
- Consultas y búsquedas para encontrar información
- Configuración y gestión de cuentas de usuario con diferentes niveles de acceso a los datos (roles).
- Importación y exportación de datos.
- Facilita el almacenamiento y el acceso a grandes cantidades de datos de forma estructurada.

4.2.10 Node-RED

Es una herramienta visual de desarrollo de aplicaciones basada en programación de flujo. Es muy popular para crear aplicaciones de gestión y manipulación de datos en tiempo real para soluciones IoT. Permite la conexión gráfica de bloques predefinidos, conocidos con el nombre de ‘nodos’. Dispone de más de 225 000 módulos en el repositorio de paquetes, facilitando el uso de nuevos nodos para agregar nuevas capacidades. Los nodos se conectan entre sí usando un editor en el navegador web como interfaz de usuario. Al conjunto de nodos interconectados se le denomina *Flow* o flujo de nodos. Algunos de los nodos disponibles son:

- Nodos que usan protocolos estándar como MQTT.
- Servicios REST.
- Clientes HTTP.
- Integraciones con APIs de terceros.

Los nodos pueden ser configurados y personalizados según las necesidades del usuario. Se pueden incrustar funciones de JavaScript para realizar tareas específicas en los nodos de función, como almacenar los flujos creados en archivos JSON para hacer copias de seguridad o para compartir los flujos con otros usuarios.

Node-RED está creado a partir de Node.js, heredando su potencia, escalabilidad y fiabilidad, con unos requerimientos de computación muy bajos. Dicha característica permite que se pueda ejecutar tanto en servidor Cloud, domésticos o en dispositivos embebidos de placa reducida.

4.2.11 Zigbee2MQTT

Zigbee2MQTT es un software que permite conectar dispositivos Zigbee a un servidor MQTT en una red local, facilitando la integración de estos dispositivos con otros que utilizan MQTT. Se pueden crear aplicaciones para controlar y monitorizar dispositivos Zigbee. También ofrece la capacidad de crear automatizaciones y reglas basadas en la información recibida de los dispositivos, además, de combinar dispositivos de distintas marcas que sean compatibles con el estándar Zigbee y que pueden ser ejecutados e instalados en diferentes plataformas.

Para habilitar la conectividad Zigbee en el ordenador y los demás dispositivos, se ha hecho uso del adaptador Texas Instruments CC2531, que posee una interfaz inalámbrica Zigbee e IEEE 802.15.4 (Figura 4.9).

Figura 4.9. USB dongle Texas Instruments CC2531.

4.2.12 Arduino-Create-Agent

Arduino Create Agent es una aplicación que permite a los usuarios interactuar con sus placas Arduino. Permite cargar programas de Arduino a través del cable USB desde una aplicación que se ejecuta en un navegador web.

En el presente trabajo se utiliza para cargar programas con las siguientes aplicaciones:

- Editor de programación visual desarrollado en el TFG (BlocklyDuino).
- Arduino Cloud Web Editor (un ejemplo se muestra en Anexos).
- Arduino IoT Cloud (un ejemplo en el apartado Anexos).

4.2.13 WebSockets

WebSockets es un protocolo que posibilita abrir una sesión de comunicación interactiva entre el navegador y un servidor [38]. A través de una API se pueden enviar mensajes a un servidor y recibir respuestas, todo ello controlado por eventos. Con WebSockets basta que un que el cliente establezca una conexión con el servidor, a partir de ese momento, el cliente envía al servidor los datos de identificación necesarios para el intercambio de información. El canal de comunicación queda abierto y el servidor también puede enviar información al cliente sin que éste tenga que solicitarlo cada vez, tal y como ocurre con el funcionamiento normal del protocolo HTTP, que se basa en un modo de operación petición y respuesta.

WebSockets son muy utilizados en aplicaciones web en tiempo real, posibilitando el intercambio de datos en ambos sentidos:

- Juegos en red.
- Portales de venta de productos.
- Chats de asistencia al usuario en línea.
- Actualizaciones en tiempo real de mensajes en redes sociales.

Para crear WebSockets, una herramienta muy usada es Socket.IO [39]. Se trata de una biblioteca que permite la comunicación de baja latencia, bidireccional y basada en eventos entre un cliente (como navegador web) y un servidor. Existen implementaciones para servidor y cliente utilizando distintos lenguajes, algunos de ellos son:

- Aplicaciones en el servidor: Node.js, Java, Python, Go.
- Aplicaciones en el cliente: JavaScript, Java, Python, C++, Net, Kotlin.

4.2.14 Lenguaje de programación Go

El lenguaje de programación Go [40] es un proyecto de código abierto desarrollado por Google, que busca aumentar la productividad del programador. Se centra en la simplicidad, la eficiencia y la facilidad de uso, donde sus mecanismos de concurrencia facilitan la escritura de programas que aprovechan al máximo las máquinas con múltiples núcleos y conectadas en red. Es un lenguaje compilado y concurrente.

Está orientado a la construcción de programas modulares, siendo muy rápido el tiempo de compilación a código máquina. El ejecutable resultante está muy optimizado para conseguir programas rápidos en su ejecución. Se utiliza para desarrollar una amplia variedad de aplicaciones: desde programas de línea de comandos hasta aplicaciones distribuidas y muy escalables.

4.2.15 Protocolo Zigbee

Zigbee es una especificación para la definición de un protocolo de comunicación entre dispositivos inalámbricos, utilizada en IoT [41]. Su consumo energético es pequeño, por lo que puede ser utilizado en dispositivos con batería. ZigBee aporta el nivel de red y aplicación al estándar IEEE 802.15.4.

Considera tres tipos de nodos: coordinador, enrutadores y dispositivos finales. Cada red Zigbee requiere un coordinador (responsable de formar la red). Tras formar la red, el coordinador adopta el rol de un enrutador para retransmitir datos de otros dispositivos. Los dispositivos finales sólo pueden comunicarse con el coordinador o con los enrutadores, además, pueden entrar en modo de suspensión para ahorrar energía.

Algunas características de este protocolo son:

- Baja latencia: proporciona tiempos de respuesta rápidos.
- Ancho de banda de rango medio.
- Uso de bandas de radiofrecuencia ISM (industriales, científicas y médicas) sin licencia (bandas ISM), Utiliza la banda de 2.4Ghz y 868 MHz.
- Retransmisión de datos entre los dispositivos para transportar datos a largas distancias.

4.2.16 Protocolo MQTT

MQTT (*MQ Telemetry Transport)* es un protocolo utilizado en la comunicación entre dispositivos en el contexto del IoT. Utiliza un modelo de publicación y suscripción, donde los dispositivos pueden publicar mensajes en temas específicos y otros dispositivos pueden suscribirse a estos temas para recibirlos.

Está diseñado para ser ligero y fácil de usar, con una sobrecarga mínima y admite tres niveles de QoS (*quality of service* - calidad de servicio) para garantizar una entrega fiable de los mensajes.

4.3 Implementación de las aplicaciones

Este apartado está formado por:

- Apartado 4.3.1 Extensión BlocklyDuino: En este apartado se detalla paso a paso la creación de nuevos bloques y que bloques han sido diseñados para este TFG.
- Apartado 4.3.2 Arduino Create Agent: En este apartado se justifican las decisiones realizadas para extender el agente de Arduino. El nuevo agente será el encargado de compilar el código generado por cada aplicación realizada en BlocklyDuino, además, será capaz de trasladar este código al proyecto Wokwi para ser simulado. Se definen los nuevos comandos para realizar las nuevas funciones.
- Apartado 4.3.3. Despliegue de servicios: En este apartado se muestra la estructura y configuración específica para cada uno de los contenedores. Se va a disponer de seis contenedores, conectados a una red para que puedan comunicarse entre ellos. Cada uno de estos contenedores se van a organizar utilizando imágenes y configuraciones específicas que se explican en cada subapartado.

4.3.1 Extensión BlocklyDuino

Cuando se solicita el servicio de BlocklyDuino, se accede a la interfaz de usuario personalizada que se ha creado para realizar el TFG. Como se puede observar en la Figura 4.10, en la parte superior derecha, se dispone de cinco botones: Abrir, Guardar, Descartar, Guardar código y Cargar. En la parte inferior derecha se encuentra el icono de la papelera.

Figura 4.10. Interfaz de usuario del editor visual de bloques.

La función de cada uno de los elementos indicados es la siguiente:

1. En la pestaña **Bloques**, se muestran todos los bloques de programación disponibles agrupados por categorías. En el apartado 4.3.1.1 se exponen las distintas categorías y bloques creados para este TFG.
2. En la pestaña **Arduino**, se presenta el código generado por los distintos bloques utilizados. Este código generado se muestra en un lenguaje de programación compatible con el microcontrolador como es C/C++ y será el que se cargue en el dispositivo final, que puede ser un dispositivo simulado o físico.
3. En la pestaña **Monitor**, se muestra la información enviada al puerto serie del dispositivo. Es muy útil para realizar tareas de depuración del programa. Esta funcionalidad ha sido creada para este TFG, ya que no existía en el proyecto original BlocklyDuino y se detalla en el apartado 4.3.2.
4. En la pestaña **Registro,** muestra la información y el estado del proceso de compilación y carga de un programa en un dispositivo físico. Esta funcionalidad ha sido creada en este TFG, ya que no existía en el proyecto original BlocklyDuino y se detalla en el apartado 4.3.2.
5. Botón **Abrir,** permite cargar un proyecto previamente creado en la interfaz de programación visual.
6. Botón **Guardar**, permite almacenar un proyecto para su uso futuro o para ser compartido.

7. Botón **Descartar**, borra todos los bloques del área de trabajo, sin guardar previamente el proyecto creado.
8. Botón **Guardar código**, permite guardar el código generado en un archivo con extensión 'ino'. Este código se puede utilizar, por ejemplo, en el IDE de Arduino.
9. Botón **Cargar** (resaltado de color verde), permite cargar el código generado en un dispositivo físico o un simulador. Se han contemplado cuatro escenarios diferentes:
    a. Cargar el código en un proyecto Wokwi ya existente.
    b. En un nuevo proyecto Wokwi Arduino UNO.
    c. Cargar el código en un nuevo proyecto Wokwi ESP32.
    d. Realizar la compilación y carga del código en un dispositivo físico Arduino UNO WiFi Rev2, conectado al puerto COM del ordenador. Esta funcionalidad ha sido creada en este TFG y se detalla en el apartado 4.3.2.
10. Icono **papelera**, borra el último bloque incluido en el área de trabajo.

4.3.1.1 Nuevas categorías y bloques

En el proyecto original de BlocklyDuino, las categorías disponibles en la pestaña de bloques son las siguientes: Logic, Control, Math, Text, Variables, Functions, Input/Output, Servo, Grove, Analog, Grove, Grove LCD y Grove Motor. A estas categorías se les ha añadido las siguientes:

- Categoría Project: Contiene un bloque **Project** para crear un nuevo proyecto. Es el bloque inicial del proyecto. Todos los programas empezaran con este bloque, ya que indica el tipo de dispositivo para el que se generará el código. Este bloque tiene tres parámetros:
    - Device. Indica el tipo de dispositivo para el que se generará el código (traducción de bloques a código). Puede tomar cuatro valores: 'Open existing Wokwi', 'Wokwi Arduino UNO', 'Wokwi ESP32' y 'Arduino wifi'. Los tres primeros generan código adaptado a la plataforma de simulación Wokwi, incorporando el código a un proyecto ya existente, o bien generando código en un nuevo proyecto Wokwi para Arduino UNO o ESP32. En el caso de elegir la opción 'Arduino uno wifi', genera código para un dispositivo físico Arduino UNO WiFi Rev2.
    - Wokwi Project. En este campo se indica la URL del proyecto, si se ha elegido que se genere código para incorporarlo en un proyecto Wokwi ya existente. Es un valor de tipo *String*.

- Agent address. Se reserva para indicar la dirección del agente Arduino, en caso de existir, que permitirá la carga del código programado, en el dispositivo físico: Arduino UNO WiFi.

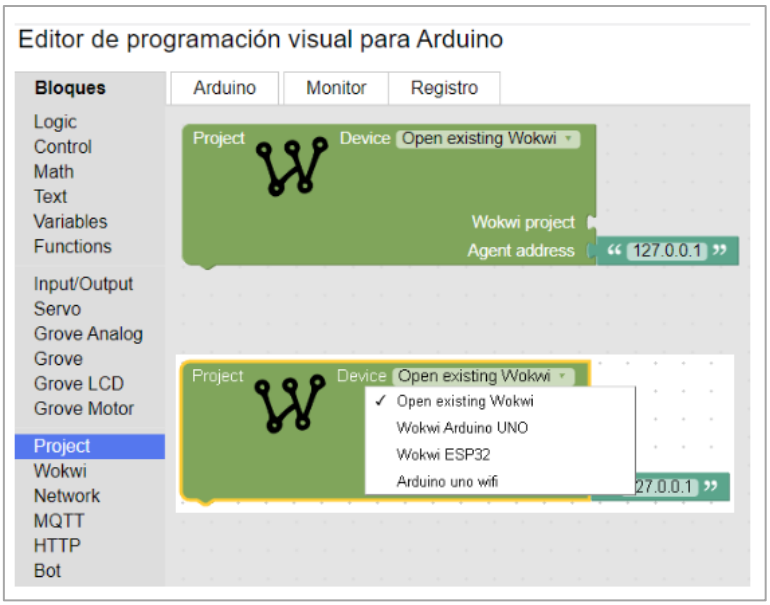


Figura 4.11. Bloque de la categoría Project.

- Categoría Wokwi. Compuesta por cuatro bloques creados para generar código para una serie de dispositivos, utilizados en las actividades desarrolladas en el capítulo 5, para que sean compatibles con Wokwi, Arduino y ESP32. Los bloques son los siguientes:
  - **Ultrasonic Ranger**. Este sensor dispara un ultrasonido y recibe el eco. Tiene los siguientes pines de conexión: VCC, Trig (disparo del ultrasonido), Echo (recepción del ultrasonido) y GND. Hay que indicar el pin del microcontrolador donde está conectada la señal Echo y se asume que en el número de ese pin+1 se encuentra conectado el Trig.
  - **DHT22 Sensor**. Este sensor posee tres pines: VCC, GND y SDA. Esta última es la línea de datos. Hay que indicar el pin del microcontrolador al cual está conectada la línea SDA.
  - **I2C LCD**. Pantalla LED de 16x2 caracteres con interfaz I2C. El bloque posee un parámetro 'Address' para indicar la dirección I2C por defecto, que puede ser 0x27 (si se usa Wokwi) o 0x3F (LCD utilizado en el TFG). Incorpora los parámetros 'print line1' y 'print line2' para escribir caracteres en la fila 1 o fila

2 y un parámetro ‘delay’, para incorporar un retardo a la escritura de datos en el LCD.

- **Piezzo buzzer**. Bloque para que un zumbador piezoeléctrico genere un tono. Hay que indicar el pin del microcontrolador donde se conecta la señal VCC del zumbador.

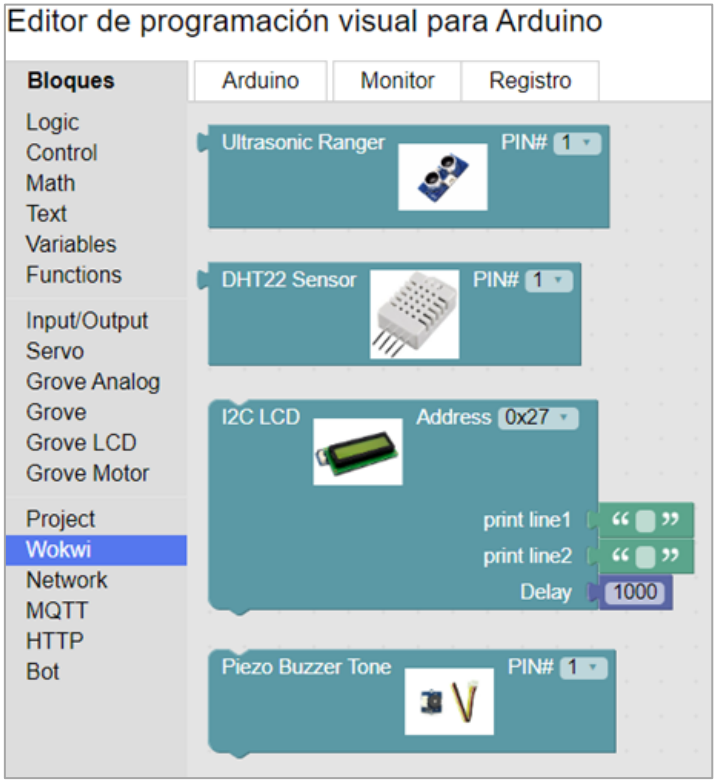


Figura 4.12. Bloques de la categoría Wokwi.

- Categoría Network. Consta de dos bloques, una para realizar una conexión WiFi a través de un punto de acceso y otra para enviar datos a través de Bluetooth. Los bloques creados son:
  - **WiFi**. Permite que el microcontrolador se conecte a un punto de acceso usando la interfaz Wifi. Hay que proporcionar dos parámetros:
    - ssid: identificador de la red.
    - passwd: contraseña.
  - **Bluetooth Send**. Genera el Código necesario para enviar un mensaje a través de un dispositivo Bluetooth HC-05, que dispone de los siguientes pines: VCC, GND, RX y TX. Hay que indicar el pin del microcontrolador donde está conectado RX. En ‘message’ se indica el mensaje que se enviará.

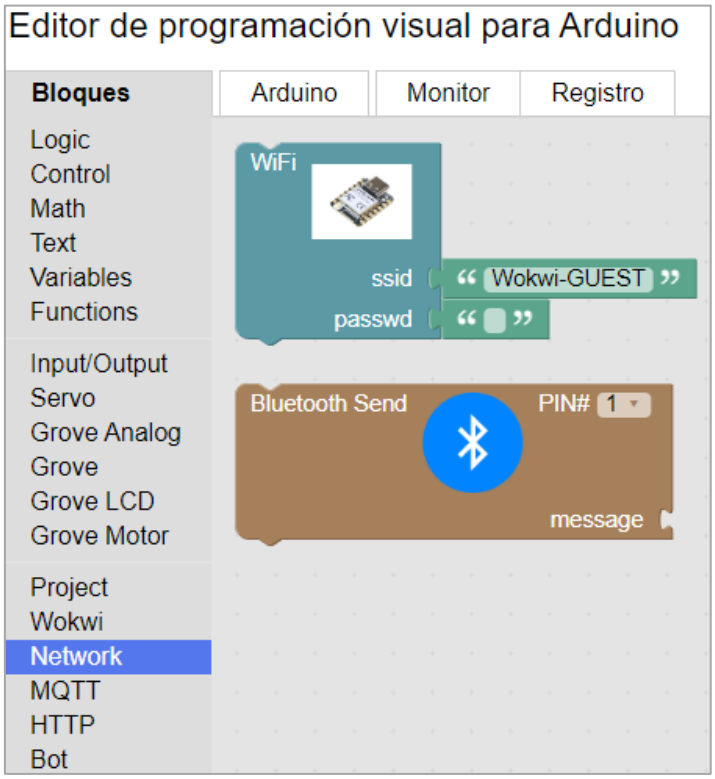


Figura 4.13. Bloques de la categoría Network.

- Categoría MQTT. Contiene tres bloques:
  - **MQTT Client.** Este bloque crea un cliente MQTT, estableciendo una conexión con el bróker MQTT. Se proporciona un parámetro: bróker, que es la dirección del bróker.
  - **MQTT Subscriber.** Permite suscribirse a un tema, indicado en el parámetro de entrada tema.
  - **MQTT Publisher.** Publica un mensaje en el tema indicado en el parámetro tema.

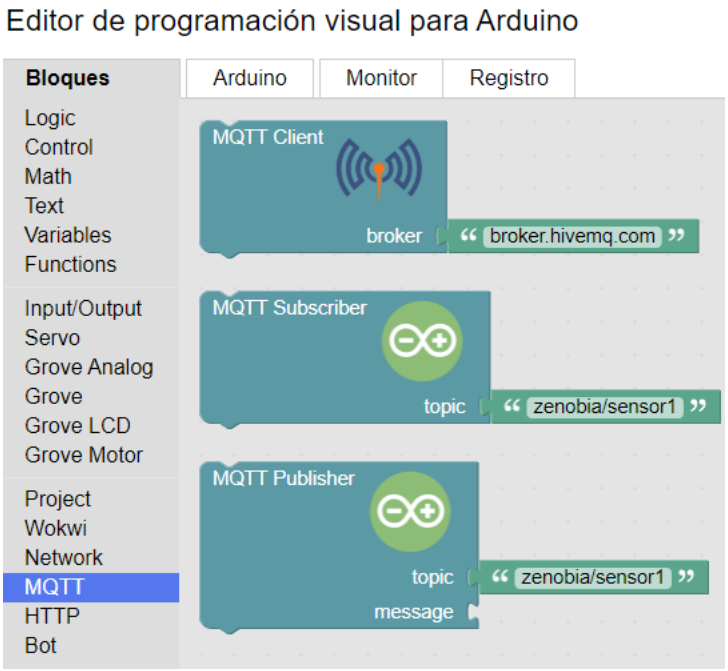


Figura 4.14. Bloques de la categoría MQTT.

- Categoría HTTP. Compuesta por dos bloques, uno para realizar peticiones HTTP (cliente) y otra para crear servicios HTTP (servidor). Estos bloques con funcionalidad de cliente y servidor HTTP son:
  - **Web Server.** Crea un servicio web, indicando la URL del servicio (‘on uri’), el mensaje que devuelve y si es necesario, se puede llamar a una función indicando su nombre en el parámetro ‘callback’.
  - **HTTP Request.** Construye un mensaje de petición HTTP usando el método GET o POST, según se elija en el parámetro ‘Method’. Dispone de otros dos parámetros llamados ‘url’ y ‘message’ para indicar la URL del servicio que se solicita y el mensaje que se envía en el mensaje de petición.

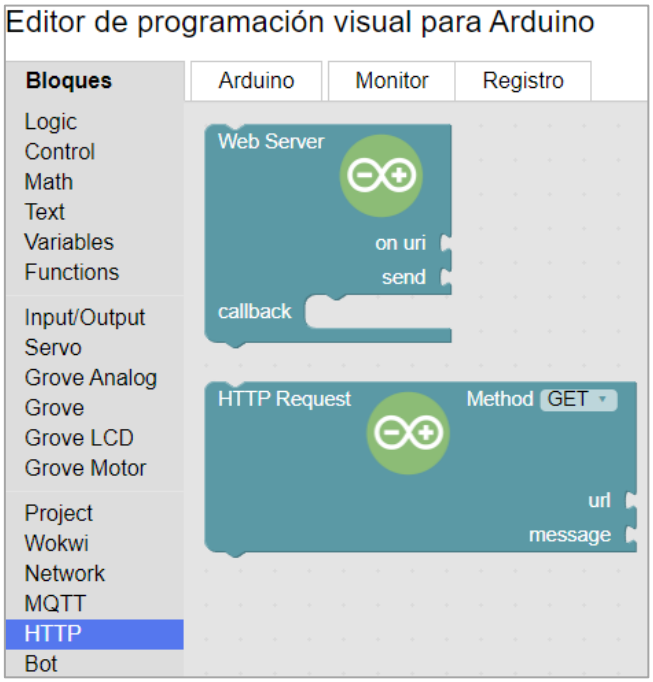


Figura 4.15. Bloques de la categoría HTTP.

- Categoría Bot. Dispone de dos bloques, uno para crear un Bot de Telegram y otro para atender los comandos que se envíen por el chat del Bot creado. Los bloques de esta categoría son:
  - **Telegram Bot**. Encargado de generar el código para construir un Bot de Telegram. Posee un parámetro 'token', generado por Telegram para poder comunicarnos con el Bot, se explicará el proceso en el desarrollo de las actividades del capítulo 5. En 'handle new messages' se van incluyendo los bloques de tipo 'Bot Message' para atender a los distintos comandos.
  - **Bot Message**. Atiende un determinado comando recibido por el chat. El parámetro 'on message' se usa para indicar el comando que será atendido por el chat y el parámetro 'send' para escribir el mensaje de respuesta. Si se desea ejecutar una función antes del envío de la respuesta, en 'callback' se indica el nombre de la función.

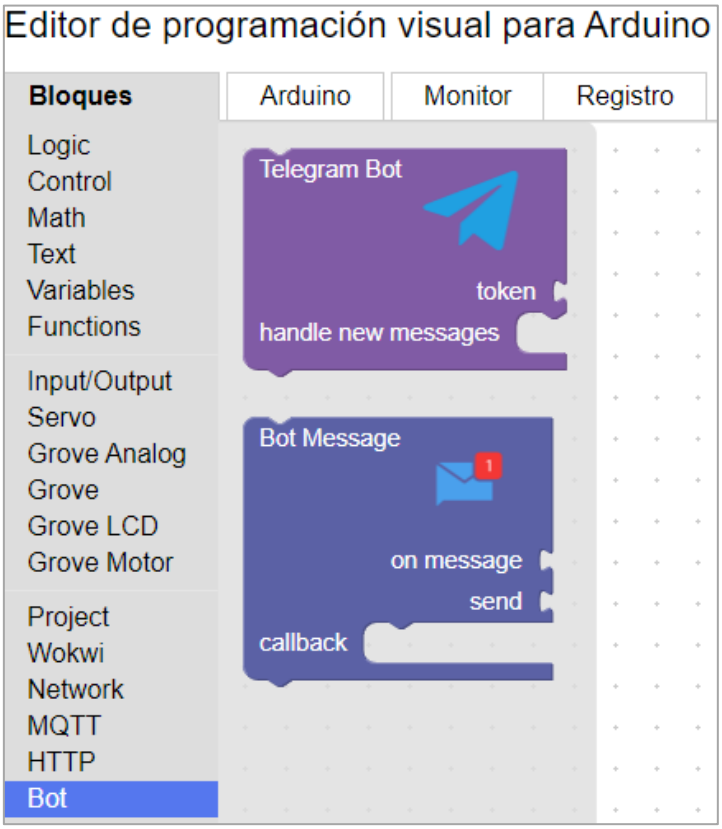


Figura 4.16. Bloques de la categoría BOT.

4.3.1.2 Creación de bloques

Para crear un nuevo bloque hay que seguir los siguientes pasos:

1. Definir un nuevo Bloque. Se indica el nombre del mismo, las entradas y campos existentes y los tipos de datos asociados a estos campos. Así mismo, se puede controlar el aspecto que tendrá el bloque, indicando la posición de los campos y el color del bloque. Toda esta información se incorpora en un nuevo archivo JavaScript en el directorio /blockly/blocks/. La generación de este nuevo archivo puede ser realizada manualmente, usando como modelo algunos de archivos ya existentes o se puede utilizar Blockly Developer Tools [42]. Se trata de una herramienta visual online que permite crear nuevos bloques. La interfaz de usuario tiene distintas áreas, en la parte izquierda, como se muestra en la Figura 4.17, tiene un menú que permite construir el nuevo bloque y en la parte derecha, como se muestra en la Figura 4.18, se presenta una previsualización y el código que hay que incluir en el archivo JavaScript.

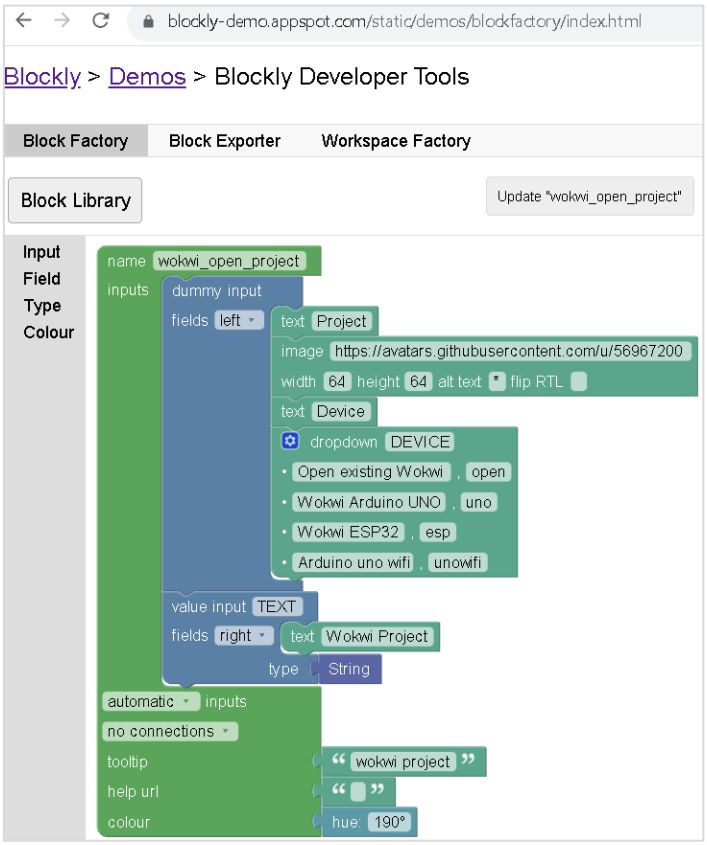


Figura 4.17. Blockly Developer Tools.

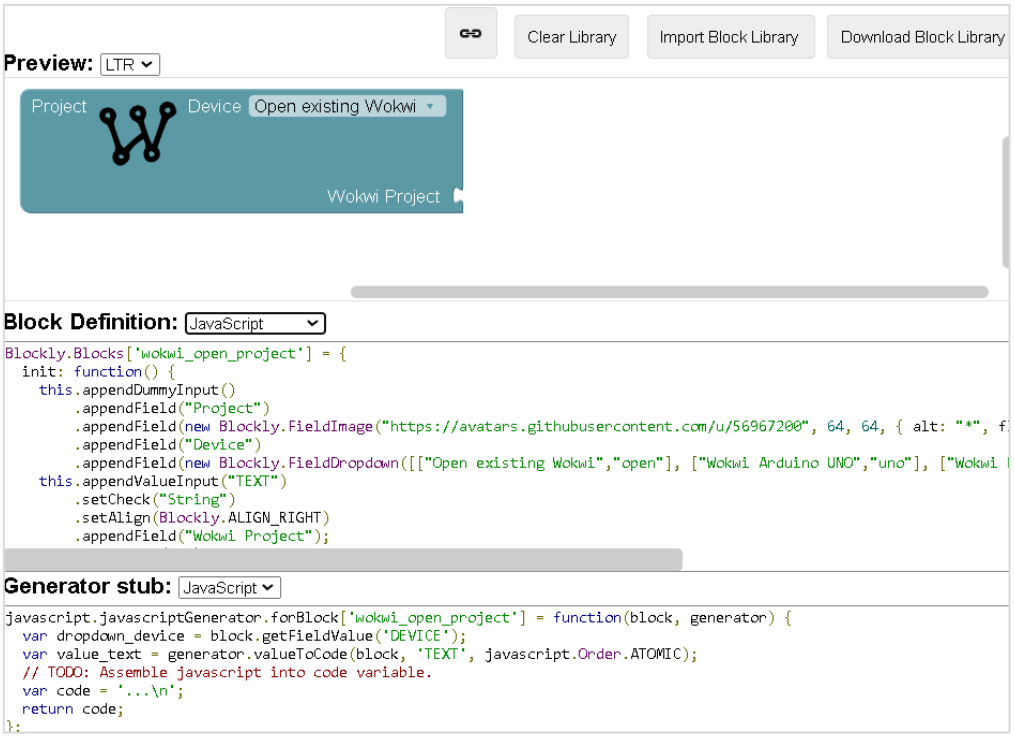


Figura 4.18. Previsualización y código del nuevo bloque.

2. Definir Generadores de Código. Se define el código que genera ese bloque. Hay que adaptarlo al dispositivo final donde se va a cargar, en nuestro caso, va a ser para dispositivos Arduino o ESP32. Para ello, se utiliza como modelo otros archivos existentes y se incorpora un nuevo archivo JavaScript en el directorio blockly/generators/Arduino con la programación necesaria para que genere el código asociado al nuevo bloque.

3. Agregarlo a una categoría: esto se realiza en el archivo **index.html**. Por ejemplo, uno de los bloques creados para este TFG llamado 'wokwi_ultrasonic_ranger' se incluye en la categoría 'Wokwi' con el siguiente código:

```
<category name="Wokwi">
<block type="wokwi_ultrasonic_ranger"></block>
</category>
```

4. El código fuente contenido en el archivo donde se ha definido el nuevo bloque hay que minimizarlo y añadirlo al archivo blocks_compressed.js. Este archivo contiene la definición de los bloques disponibles. La minimización de código consiste en optimizar el código para ahorrar espacio, reducir los tiempos de carga sin alterar la funcionalidad. Se ha realizado con la herramienta online Toptal JavaScript Minifier [43].

5. El código fuente contenido en el archivo donde se ha definido el código que genera el bloque, se minimiza y se añade al archivo arduino_compressed.js, que contiene cómo se genera el código Arduino de todos los bloques.

4.3.2 Arduino Create Agent

Durante el desarrollo del TFG se tomaron las siguientes decisiones:

- Sustituir el servidor HTTP de BlocklyDuino encargado de ofrecer la interfaz por uno más simplificado, donde se eliminan la compilación y carga.
- Realizar las tareas de compilación y carga utilizando una extensión de la aplicación Arduino Create Agent.
- Mantener la funcionalidad original del agente de Arduino, para que fuera compatible con el desarrollo de aplicaciones en un entorno Arduino IoT Cloud.
- El código generado por los distintos bloques se enviaría a los dispositivos simulados, copiando el código desde BlocklyDuino y pegando el código en un proyecto Wokwi a través del navegador. En el caso de dispositivos físicos,

enviándolo al agente Arduino para una compilación y carga. Estas tareas serían asumidas por el agente.

- Para establecer la comunicación entre navegador y agente se utilizarían WebSockets de la biblioteca socket.io. Esto suponía realizar modificaciones en el archivo index.html del proyecto BlocklyDuino, encargado de ofrecer la interfaz de usuario. El uso de WebSockets es posible ya que el agente acepta este tipo de comunicación, al haber sido diseñado para comunicarse con las aplicaciones cliente de Arduino IoT Cloud que se ejecutan en el navegador.
- El bloque de proyecto de la categoría Project debía elegir el agente al que enviar el código: para ello utilizaría la dirección IP de la máquina donde se ejecuta el agente.

Arduino Create Agent es un proyecto creado en lenguaje de programación Go. El agente original permite uso de WebSockets y peticiones HTTP o HTTPS. Es posible conectarse a él utilizando un navegador y observar la lista de comandos (Figura 4.19). Un resumen de estos comandos es:

- **list**, Lista los dispositivos conectados en los puertos serie.
- **open <portName> <baud>.** Abre un puerto llamado <portName> a una velocidad <baud>
- **send <porName> <cmd>.** Envía un comando a un dispositivo a través de un puerto <porName>. Cuando dispositivo recibe un comando, puede responder con un mensaje. El agente puede recibir mensajes enviados desde el dispositivo de modo asíncrono.
- **close <portName**>. Cierra un puerto.
- **restart.** Reinicia el agente.
- **exit**. Finaliza la ejecución del agente.
- **killupload**. Finaliza una carga de programa en el dispositivo físico.
- **download <tool>.** Descarga una herramienta desde el portal de descargas de Arduino.
- **gc.** Fuerza el inicio de la liberación de objetos de la memoria que ya no están en uso.
- **hostname**. Muestra el nombre del equipo donde se ejecuta el agente.
- **version**. Muestra la versión del agente.

Antes de ejecutar el agente es necesario generar unos certificados para utilizar HTTPS. Para ello se ejecuta la instrucción:

arduino-create-agent.exe -generateCert.

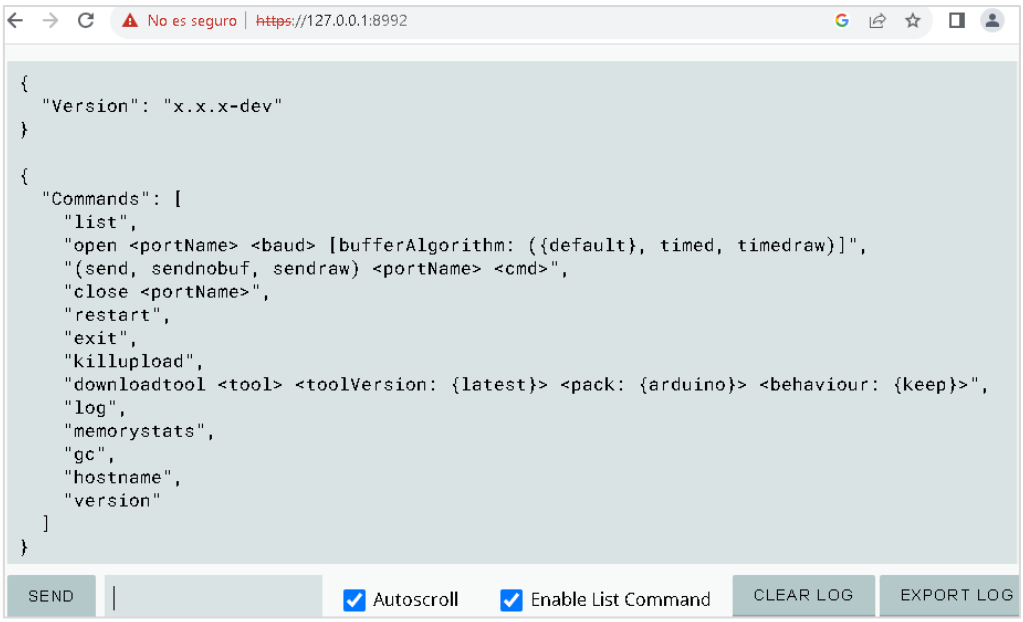


Figura 4.19. Interfaz del agente Arduino original.

La extensión del agente se ha realizado para cumplir con los siguientes objetivos:

- El primero es que el agente realice las tareas de compilación y carga de un programa realizado con el editor de bloques en un dispositivo físico.
- En caso de dispositivos simulados, el agente tendrá la capacidad de pegar código existente en el portapapeles en un proyecto Wokwi.
- No se alterará la funcionalidad original del agente para que continúen funcionando las aplicaciones desarrolladas en el entorno Arduino IoT Cloud.

Para cumplir con los objetivos, el agente extendido incorpora estos nuevos comandos:

- **downloadtool arduino-cli.** Cuando el agente recibe este comando, se instala en el equipo la aplicación ‘arduino-cli’, utilizada para compilar el código y cargar el programa resultante en el dispositivo físico. La instalación se realiza en el directorio ./arduino-create/arduino, dentro del directorio de usuario. El comando se puede enviar al agente utilizando la interfaz de usuario ofrecida en la dirección

https://127.0.0.1:8992 (Figura 4.20) o bien se puede enviar desde una instrucción Javascript incluida en index.html:

```
socket.on('connect', function () {
  socket.emit("command", "downloadtool arduino-cli");
};
```

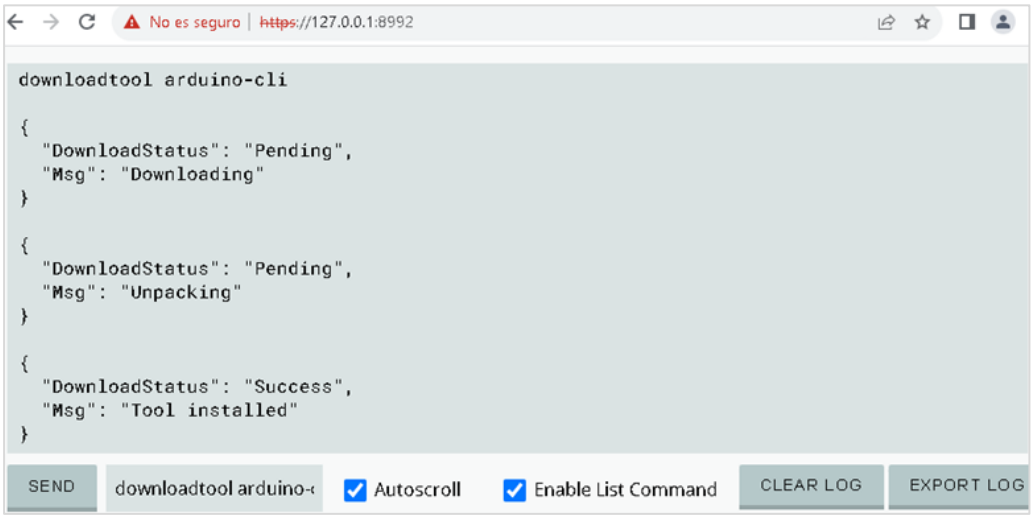


Figura 4.20. Instalación de arduino-cli desde la interfaz de usuario del agente.

Antes de instalar la aplicación, se comprueba si existe en el sistema y de ser así se devuelve un mensaje indicando que ya ha sido realizada (y no se instala), tal y como se muestra en la siguiente figura:

Figura 4.21. Aviso de la existencia de arduino-cli en el sistema.

- **wokwi.** Al recibir este comando, el agente pega en un proyecto Wokwi el código guardado en el portapapeles. El proceso es el siguiente: en el caso de dispositivos

simulados en Wokwi, al iniciar el proceso de carga en dispositivo, lo primero que hace el agente es copiar en el portapapeles el código generado por los bloques. Esto se realiza a través de un código JavaScript existente en index.html. A continuación, ese mismo código abre una nueva pestaña con un proyecto Wokwi y envía el comando 'wokwi' al agente Arduino. Cuando el agente recibe el comando, espera unos segundos y pega sobre la pestaña activa del proyecto Wokwi el contenido del portapapeles. Se genera un efecto de carga del código en el nuevo proyecto del simulador.

- **lib install <nombre_de_biblioteca>**. Se instala una biblioteca en el equipo para que el compilador la utilice. La instalación se puede realizar desde la interfaz del agente (Figura 4.22) o desde una instrucción JavaScript incluida en index.html, por ejemplo:

```
socket.on('connect', function () {
   socket.emit("command", "lib install LiquidCrystal+I2C");
 };
```

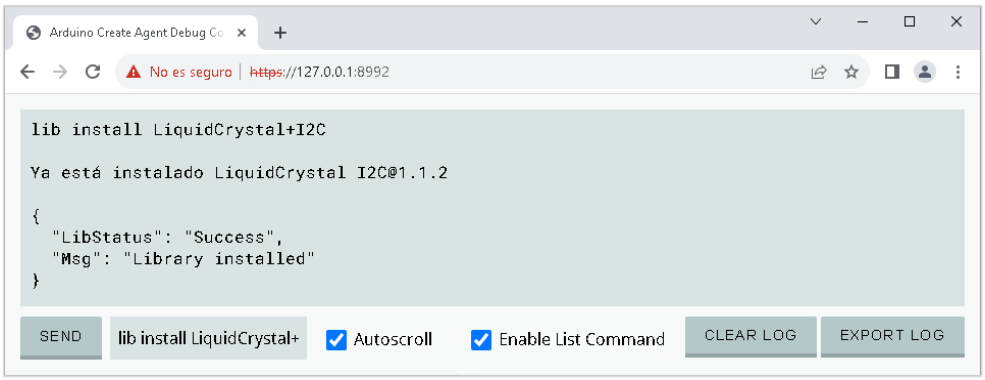


Figura 4.22. Instalación de una biblioteca de Arduino.

- **upload <portName> <FQBN> <code_base64_encoded>**. El código recibido se compila y carga en el dispositivo físico. <portName> es el puerto donde está conectado el dispositivo, <FQBN> es el nombre de la placa y <code_base64_encoded> es el código de programa codificado en base64. Cuando recibe el comando el agente, realiza dos llamadas al sistema operativo para que ejecute dos procesos orientados a la compilación del código y carga del programa resultante en el dispositivo físico. El primer proceso ejecuta la siguiente instrucción encargada de compilar el código:

arduino-cli compile -p <portName> -b <FQBN> <sketch_dir>

donde <sketch_dir> es el directorio temporal donde se ha guardado un archivo (extensión.ino) con el código de programa recibido desde el navegador. Hay que indicar que antes de guardar el código se debe decodificar en base64.

El segundo proceso ejecuta la instrucción que realiza la carga en el dispositivo:

arduino-cli upload -p <portName> -b <FQBN> <sketch_dir>.

Las nuevas funcionalidades han sido implementadas modificando los siguientes archivos:

- **main.go**. El agente original, solo permite que le envíen comandos desde aplicaciones del navegador que se han cargado desde los orígenes create.arduino.cc y cloud.arduino.cc. En el caso del agente extendido, va a permitir la recepción de comandos desde aplicaciones del navegador que se hayan cargado desde cualquier origen, incluyendo en el archivo los valores 'https://*' y 'https://*:8080' en la variable 'extraOrigins'. Esto da la flexibilidad de ubicar el editor de bloques en cualquier máquina o servidor, pero introduce una gran vulnerabilidad en el agente, ya que podría recibir comandos de otras aplicaciones que se ejecuten en el navegador y que no tengan nada que ver con el editor. Por tanto, es recomendable finalizar la ejecución del agente Arduino extendido cuando no se utilice.

- **hub.go**. Incorpora el código necesario de los nuevos comandos. Existe una función llamada checkCmd que recibe los mensajes que llegan al servidor de WebSockets, tal y como se muestra en la Figura 4.23. Los mensajes se identifican en esta función y se ejecutan las acciones asociadas a cada uno de ellos. Para crear un ejecutable del nuevo agente extendido, se ejecuta el comando 'go build'.

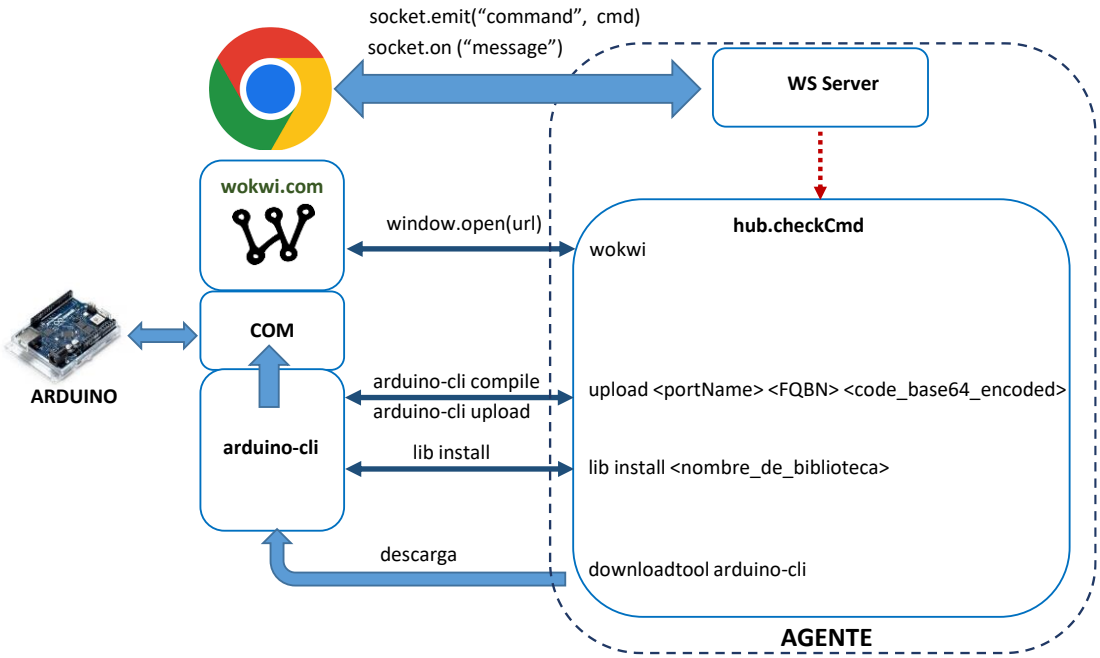


Figura 4.23. Extensión del agente Arduino.

La conexión entre el navegador y el agente Arduino, se establece utilizando WebSockets. Una vez establecida la conexión, se irán emitiendo una serie de comandos al agente y al mismo tiempo, de forma asíncrona, se irán recibiendo mensajes en el navegador originados en el agente (Figura 4.24).

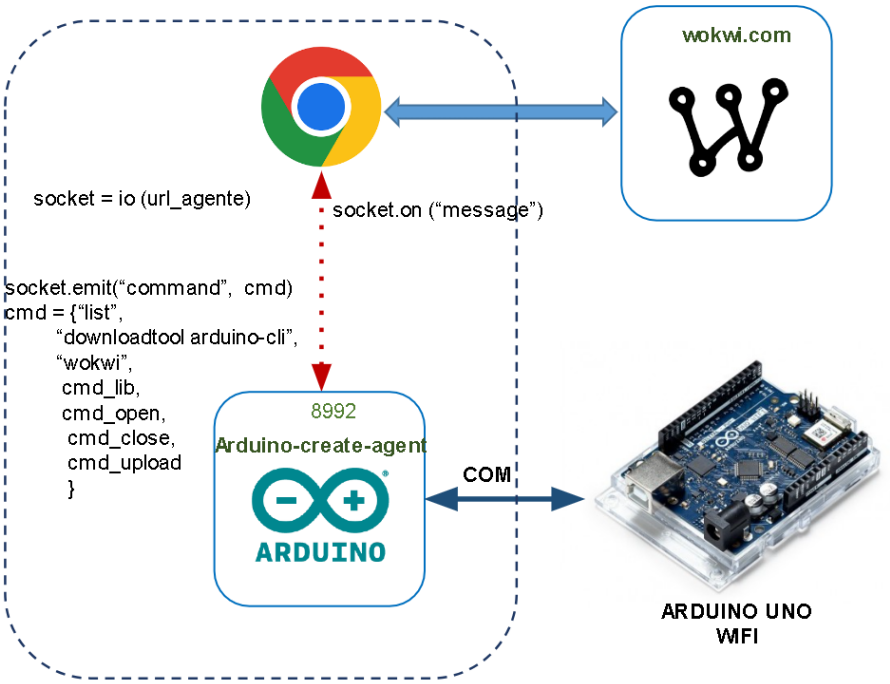


Figura 4.24. Comunicación entre el navegador y agente Arduino.

A continuación, se enumeran los mensajes que el navegador puede enviar al agente:

- **‘list’.** El navegador consulta la lista de puertos con dispositivos conectados.

Figura 4.25. Dispositivos conectados a los puertos serie.

- **‘downloadtool arduino-cli’.** El navegador solicita que se instale en el equipo del agente la aplicación ‘arduino-cli’, para compilar el código y cargar el programa en el dispositivo físico.
- **‘wokwi’.** Indica que el código debe copiarse y pegar en un proyecto Wokwi, para así simular un dispositivo. El proyecto puede ser nuevo o uno existente.
- **‘cmd_lib’**. Se hace referencia al comando compuesto por la cadena: ‘lib install’ + <nombre_de_biblioteca>. Se instala una biblioteca en el equipo agente para que sea utilizada por el compilador.
- **‘cmd_open’**. Consiste en el comando: ‘open ‘ + port + ’9600’, donde port es el puerto donde está conectado al dispositivo (por ejemplo, COM3).
- **‘cmd_compile’**. El navegador envía la cadena: ‘upload’ + port + ‘ ‘ + FQBN + ‘ ‘ + codigo_encoded_base64, donde ‘port’ es el puerto donde está conectado el dispositivo FQBN (*Fully Qualified Board Name*) es el nombre de la placa (por ejemplo, arduino:megaavr:uno2018) y ‘código_encoded_base64’ es el código de programa codificado en base64. Cuando el agente recibe este comando, compila y carga el resultado en el dispositivo.

4.3.3 Despliegue de los servicios

Los servicios se despliegan utilizando contenedores Docker. Existen imágenes que contienen comprimidos en un paquete los códigos, bibliotecas y cualquier dependencia que necesite el código. De forma sencilla y en un solo archivo de configuración Docker Compose: **docker-compose.yml**, se han definido la estructura y configuración los distintos servicios y cómo se han de ejecutar. Este archivo de configuración se ha creado manualmente utilizando un editor de texto y haciendo uso de comandos.

COMANDO	DESCRIPCIÓN
version	Define la versión del archivo Docker-compose.
services	Define la lista de servicios que se ejecutarán como contenedores.
build	Ruta al archivo Dockerfile para construir una imagen.
image	Define la imagen de Docker que se usará para un servicio.
Container-name	Nombre del contenedor que se creará para un servicio.
restart	Política de reinicio para un servicio.
volumes	Definir los volúmenes que se montarán para un servicio.
ports	Define las asignaciones de puertos para un servicio.
networks	Permite configurar redes que se pueden reutilizar en múltiples servicios.
environment	Variables de entorno que se establecerán para un servicio.

Tabla 4.1. Comandos utilizados en el documento docker-compose.yml.

En la Figura 4.26, se muestra los diferentes servicios definidos utilizando contenedores Docker:

S1: sistema de gestión de bases de datos relacional MySQL (apartado 4.3.1.1).

S2: phpMyAdmin, herramienta de administración de gestores de bases de datos MySQL (apartado 4.3.1.2).

S3: Node-Red, herramienta visual de desarrollo de aplicaciones basada en programación de flujo (apartado 4.3.1.3).

S4: proxy inverso Nginx (apartado 4.3.1.4).

S5: Zigbee2MQTT, pasarela software para la conexión de dispositivos que utilizan el protocolo Zigbee con otros que usan el protocolo MQTT (apartado 4.3.1.5).

S6: editor de programación visual basado en bloques para Arduino y dispositivos de la familia ESP32 (apartado 4.3.1.6).

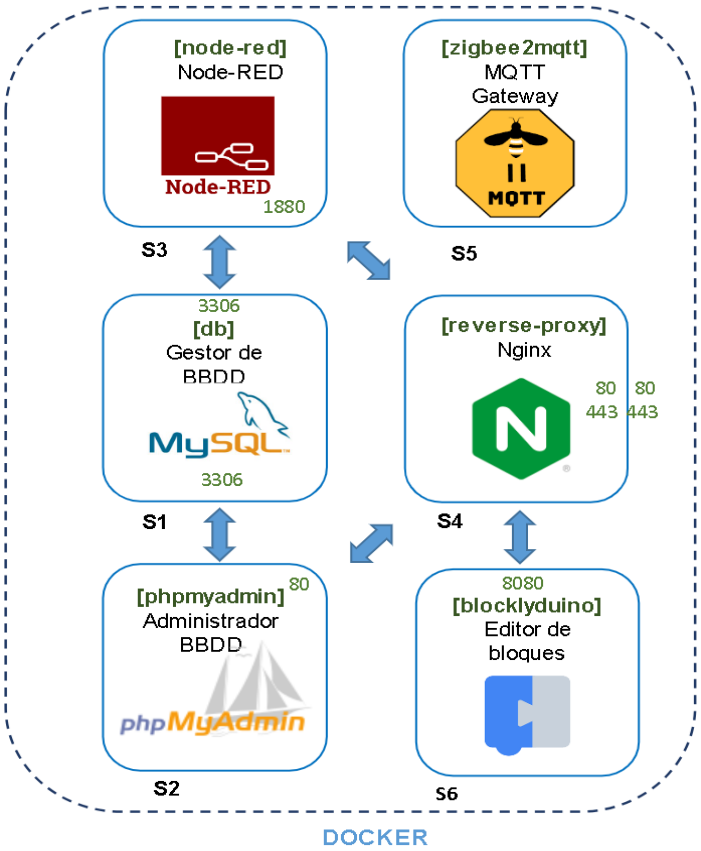


Figura 4.26. Servicios desplegados en contenedores Docker.

En el archivo 'docker-compose.yml' se crea una red virtual personalizada. Su nombre es 'telemetry' y tiene como dirección de red 172.6.238.0/24. Los comandos de configuración de esta red, son los siguiente:

```
network:
  telemetry:
    ipam:
      driver: default
          config:
                -subnet:172.16.238.0/24
```

A cada contenedor se le asigna una dirección perteneciente a 172.238.0/24 y un nombre ('container_name'), para que puedan ser referenciados.

Container_name	IP
node-red	172.16.238.2
db	172.16.238.3
Node-red1	172.16.238.4
reverse_proxy	172.16.238.5
phpmyadmin	172.16.238.6
blocklyduino	172.16.238.7
Zigbee2mqtt	172.16.238.8

Tabla 4.2. Asignación de IP a cada servicio.

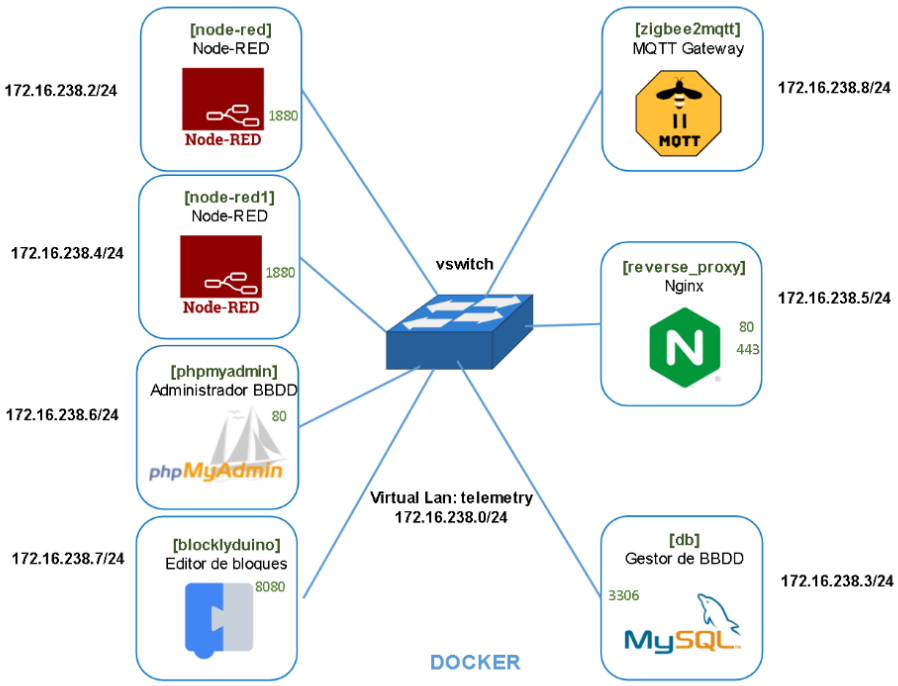


Figura 4.27. Red virtual 'telemetry'.

4.3.3.1 Servicio de gestión de bases de datos relacional MySQL

Configurar un servicio de gestión de bases de datos MySQL utilizando Docker es una forma de trabajar con bases de datos en entornos de desarrollo o producción. Para realizar el despliegue en un contenedor Docker, se han incluido los siguientes comandos en el archivo docker-compose.yml:

```
db:
  image: yobasystems/alpine-mariadb
  container_name: db
  volumes:
        - ./mysql-data/:/var/lib/mysql
  restart: always
  environment:
        - MYSQL_ROOT_PASSWORD= zenobia
        - MYSQL_DATABASE=medidas
   networks:
        - telemetry
```

A continuación, se proporciona paso a paso la configuración de servicio:

1. La imagen que se ejecuta en el contenedor es 'yobasystems/alpine-mariadeb', es una imagen para arm32v7, que es la arquitectura de la Raspberry Pi utilizada. El nombre asignado al contenedor es: db. El sistema gestor de bases de datos que se ejecuta en el contenedor acepta conexiones en el puerto 3306. No se establece mapeo de puertos, ya que no se posibilita la conexión desde fuera de la red virtual 172.16.238.0/24.
2. MYSQL_ROOT_PASSWORD=zenobia, establece la contraseña del usuario root y MySQL y MYSQL_ROOT_DATABASE=medidas, crea una base de datos inicial llamada 'medidas'.
3. Con el comando 'volumes', se monta un volumen Docker, permitiendo que el contenedor conserve los datos (sean persistentes en el tiempo) mediante el montaje de un directorio desde el sistema de archivos del host anfitrión (Raspberry Pi).
4. El comando 'restart' se incluye para que se reinicie el contenedor automáticamente cada vez que se conecte la Raspberry Pi.
5. Finalmente, con 'networks: telemetry'**,** se conecta el contenedor a la red virtual 172.16.238.0/24.

4.3.3.2 Servicio phpMyAdmin

Este es el servicio utilizado para administrar las bases de datos relacionales MySQL. Para realizar el despliegue en un contenedor Docker, se han incluido los siguientes comandos en el archivo docker-compose.yml:

```
phpmyadmin:
  image: arm32v7/phpmyadmin
  container_name: phpmyadmin
  restart: always
  environment:
      - PMA_HOST=db
      - MYSQL_ROOT_PASSWORD=root
  networks:
      - telemetry
```

A continuación, se proporciona paso a paso la configuración de servicio:

1. La imagen que ejecuta el contenedor es 'armv32/phpmyadmin'. El contenedor está conectado a la red telemetry y acepta conexiones en el puerto 80. No se establece mapeo de puertos, para impedir la conexión desde fuera de la red virtual 172.16.238.0/24.
2. Con PMA_HOST=db, se indica la dirección del sistema de gestión de bases de datos con el nombre del contenedor: db. Ese nombre tiene una traducción a 172.16.238.3 dentro del sistema de virtualización Docker.
3. MYSQL_ROOT_PASSWORD=root, indica que la contraseña para la conexión con el sistema gestor de bases de datos es la del usuario root.
4. En la Figura 4.28, se muestra la interfaz de inicio donde se introduce el usuario y contraseña para acceder a phpMyAdmin y en la Figura 4.29, se muestra la pantalla de la interfaz gráfica para realizar tareas de administración, cómo crear, eliminar y modificar bases de datos.

Figura 4.28. Interfaz de inicio de phpMyAdmin.

Figura 4.29. Interfaz de usuario para la administración de bases de datos.

4.3.3.3 Servicio Node-Red

En el caso del servicio Node-RED, se han desplegado dos contenedores: 'node-red' y 'node-red1'. Los comandos utilizados para desplegar los son:

```
node-red:
  image: nodered/node-red
  container_name: node-red
  restart: always
  volumes:
     - ./node_red_data:/data
  networks:
     - telemetry
```

```
node-red1:
  image: nodered/node-red
  container_name: node-red1
  restart: always
  volumes
     - ./node_red_data1:/data
  networks:
     - telemetry
```

A continuación, se proporciona paso a paso la configuración de servicio:

1. La imagen que ejecuta es 'nodered/node-red'. El contenedor está conectado a la red telemetry y acepta conexiones en el puerto 1880. No se establece mapeo de puertos, y se crea un volumen en el directorio ./node_red para la persistencia de datos.
2. Se han desplegado dos contenedores para ilustrar que pueden ejecutarse más de un contenedor en el caso de existir varios usuarios que van a realizar despliegue de aplicaciones. Para este caso de múltiples contenedores, es interesante realizar una modificación en el archivo settings.js de Node-RED: se modifica el parámetro 'httpAdminRoot' para cambiar la URL de acceso al servicio y que cada contenedor

tenga una distinta. Por defecto, esta ruta está configurada como /admin. Sin embargo, para evitar conflictos, se le asigna una ruta diferente. En nuestro caso se ha asignado alu0, como aparece en la Figura 4.30. Con esta modificación, la URL para acceder al servicio es: http://172.16.238.2:1880/alu0.

```
/** By default, the Node-RED UI is available at http://localhost:1880/
 * The following property can be used to specify a different root path.
 * If set to false, this is disabled.
 */
httpAdminRoot: '/alu0',
```

Figura 4.30. Modificación en el archivo setting.js.

4.3.3.4 Servicio reverse-proxy

Los comandos que se incluyen en el archivo docker-compose.yml, para crear el servicio que actúa como Proxy inverso (reverse-proxy), son los siguientes:

```
reverse-proxy:
    build: ./nginx
    container_name: reverse_proxy
    restart: always
    ports:
        - “80:80”
        - “443:443”
    networks:
        - telemetry
```

Para desplegar este servicio, se construye una imagen a partir de las instrucciones que se encuentran en un archivo Dockerfile. Además de él, son necesarios un archivo de clave, otro de certificado y uno de configuración para Nginx, como muestra la Figura 4.31.

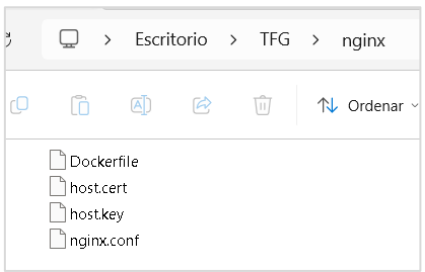


Figura 4.31. Ficheros para desplegar el servicio reverse_proxy.

Los archivos que se muestran en la imagen son:

- **host.key** y **host.cert**. Son los archivos de clave y certificado generados para utilizar SSL. Son referenciados en el archivo de configuración Nginx. Se han generado con la aplicación openssl, escribiendo los siguientes comandos en la consola:

```
openssl genrsa 2048 > host.key
openssl req -new -x509 -nodes -sha256 -days 365 -key host.key -out
```

- **nginx.conf**. Contiene la configuración para que Nginx funcione como un proxy inverso, realizando las redirecciones mostradas en la siguiente figura:

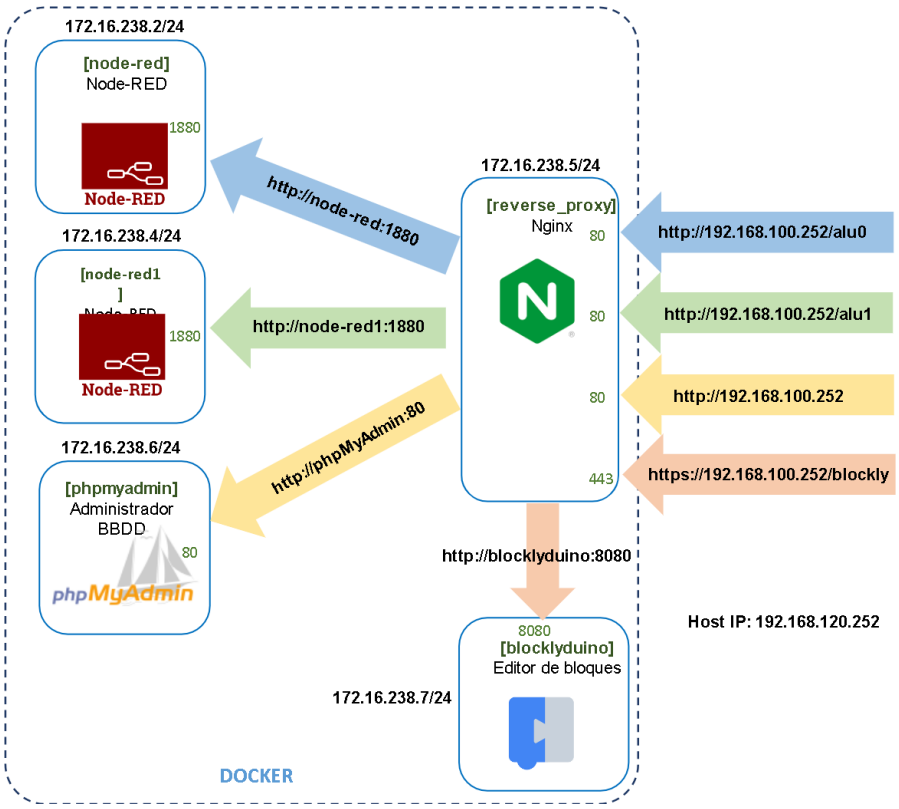


Figura 4.32. Redirecciones del servicio reverse-proxy.

Según el recurso solicitado, su funcionamiento es el siguiente:

Cuando se solicita una petición al recurso **/alu0**, se redirecciona a un contenedor llamado node-red (http://node-red:1880/alu0), si la petición es al recurso **/alu1** se redirecciona a un contenedor llamado node-red1 (http://node-red1:1880/alu1), Si recibe una petición al recurso por defecto, se reenvía al contenedor phpmyadmin (http://phpmyadmin:80) y si solicita el recurso **/blockly**, se reenvía al contenedor blocklyduino (http://blocklyduino:8080).

- **Dockerfile**. Describe como se construye la imagen: se incluye la imagen ‘nginx:latest’ y a continuación se copian los archivos nginx.conf, host.key y host.cert en el directorio ‘/etc/nginx’ de la nueva imagen. Con esto se consigue que estén disponibles cuando se inicie la ejecución del servidor.

```
FROM nginx:latest
MAINTAINER Zenobia Milla Herrero zmh00001@red.ujaen.es>

COPY ./nginx.conf /etc/nginx/nginx.conf
COPY ./host.cert /etc/nginx/host.cert
COPY ./host.key /etc/nginx/host.key
```

Figura 4.33. Contenido del archivo Dockerfile.

4.3.3.5 Servicio Zigbee2MQTT

Se ha instalado el firmware en el adaptador Texas Instruments CC2531 que permite la adaptación entre Zigbee y MQTT utilizando un depurador y programador de Texas Instruments.

Los pasos a seguir para el correcto funcionamiento son los siguientes:

1. Instalación del software para programar el adaptador (SmartRF).
2. Instalación del controlador para poder acceder al programador.
3. Conexión entre el programador y dispositivo CC2531 se realiza mediante conexión USB, como se muestra en la Figura 4.34.
4. Descarga del firmware necesario.
5. Inicio del programa SmartRF Flash Programmer, instalado en el paso 1 y configurado como se muestra en la Figura 4.35.
6. A continuación, se carga el archivo hex descargado en el paso 4 y se presiona el botón ‘Perform actions’.

Figura 4.34. Conexión entre el programador y dispositivo CC2531.

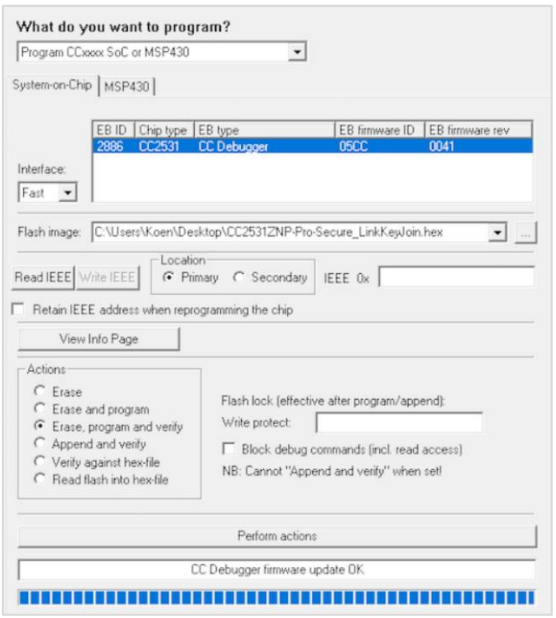


Figura 4.35. Inicio del programa SmartRF Flash Programmer**.**

7. Se ejecuta la aplicación zigbee2mqtt, que en nuestro caso es un servicio desplegado en un contenedor Docker. Se realiza la configuración donde se indica la dirección del bróker, el tema de publicación y el puerto donde está conectado el adaptador. Para esto, se edita el archivo: data\configuration.yaml. En nuestro caso el bróker utilizado ha sido el bróker público de HiveMQ y el tema elegido ‘zigbee2mqtt’. También se indica que el dispositivo CC2531 se comporta como un coordinador Zigbee al cual se unen los distintos dispositivos finales. Para desplegar el servicio, los comando que se añade al archivo docker-compose.yml son:

```
  zigbee2mqtt:
    container_name: zigbee2mqtt
    image: koenkk/zigbee2mqtt
    restart: unless-stopped
    volumes:
      - ./data:/app/data
      - /run/udev:/run/udev:ro
    ports:
      - 8081:8080
    environment:
      - TZ=Europe/Berlin
    devces:
      -/dev/serial/by-id/usb-Texas_Instruments_TI_CC2531_USB_CDC
___0X00124B001CD4E045-if00:/dev/ttyACM0
    networks:
      - telemetry
```

El archivo de configuración data/configuration.yaml contiene la siguiente información:

```
# allow new devices to join
permit_join: true

# MQTT settings
mqtt:
  # MQTT base topic for zigbee2mqtt MQTT messages
  base_topic: zigbee2mqtt
  # MQTT server URL
  server: mqtt://broker.hivemq.com

# Serial settings
serial:
  # Location of CC2531 USB sniffer
  port: /dev/ttyACM0

# Captured devices
devices:
  '0x00158d000215861b':
    friendly_name: '0x00158d000215861b'
  '0x00158d0002391187':
    friendly_name: 'pulsador'
```

Figura 4.36. Configuración de configuration.yaml.

El archivo de configuración no es estático, cuando se detecta un nuevo dispositivo Zigbee, se añade al final del archivo, en la zona de '#Captured device'. El nuevo dispositivo incorporado tiene un identificador y un parámetro asociado llamado 'friendly_name': se puede cambiar este nombre para que sea más identificable por el administrador. Cuando se ejecuta el contenedor se genera un archivo de registro de actividad (log) como al

mostrado en la Figura 4.37. Se puede observar cómo aparece la actividad de los distintos dispositivos: un dispositivo pulsador y como se publica un mensaje en el tema zigbee2mqtt/pulsador.

```
Starting Zigbee2MQTT version 1.31.0 (commit #f440c86)
Starting zigbee-herdsman (0.14.117)
zigbee-herdsman started (resumed)
Coordinator firmware version:
'{"meta":{"maintrel":3,"majorrel":2,"minorrel":6,"product":0,"revision":
20201127,"transportrev":2},"type":"zStack12"}'
Currently 2 devices are joined:
0x00158d000215861b (0x00158d000215861b): RTCGQ01LM - Xiaomi MiJia human
body movement sensor (EndDevice)
pulsador (0x00158d0002391187): WXKG01LM - Xiaomi MiJia wireless switch
(EndDevice)
`permit_join` set to `true` in configuration.yaml.
Allowing new devices to join.
Set `permit_join` to `false` once you joined all devices.
Zigbee: allowing new devices to join.
Connecting to MQTT server at mqtt://broker.hivemq.com

MQTT      publish:      topic      'zigbee2mqtt/pulsador',      payload
'{"action":"single","battery":100,"linkquality":99,"power_outage_count":
16,"voltage":3022}'
```

Figura 4.37. Registro de actividad de la aplicación Zigbee2MQTT**.**

4.3.3.6 Servicio BlocklyDuino.

Es un servicio que proporciona un editor de programación basado en bloques para Arduino y dispositivos de la familia ESP32. Es una extensión del proyecto BlocklyDuino que a su vez está basado en Blockly. Genera código que puede ser cargado en las placas de desarrollo o en el simulador Wokwi. Para desplegar el servicio se incluyen los siguientes comandos en el archivo docker-compose.yml:

```
blocklyduino:
  build: ./blocklyduino
  container_name: blocklyduino
  restart: always
  ports:
      - "8080"
  networks:
      - telemetry
```

Para desplegar este servicio, se construye una nueva imagen a partir de las instrucciones que se encuentran en el archivo: Dockerfile. Junto a ese archivo, dentro del directorio se localizan los archivos y carpetas mostrados en la siguiente figura:

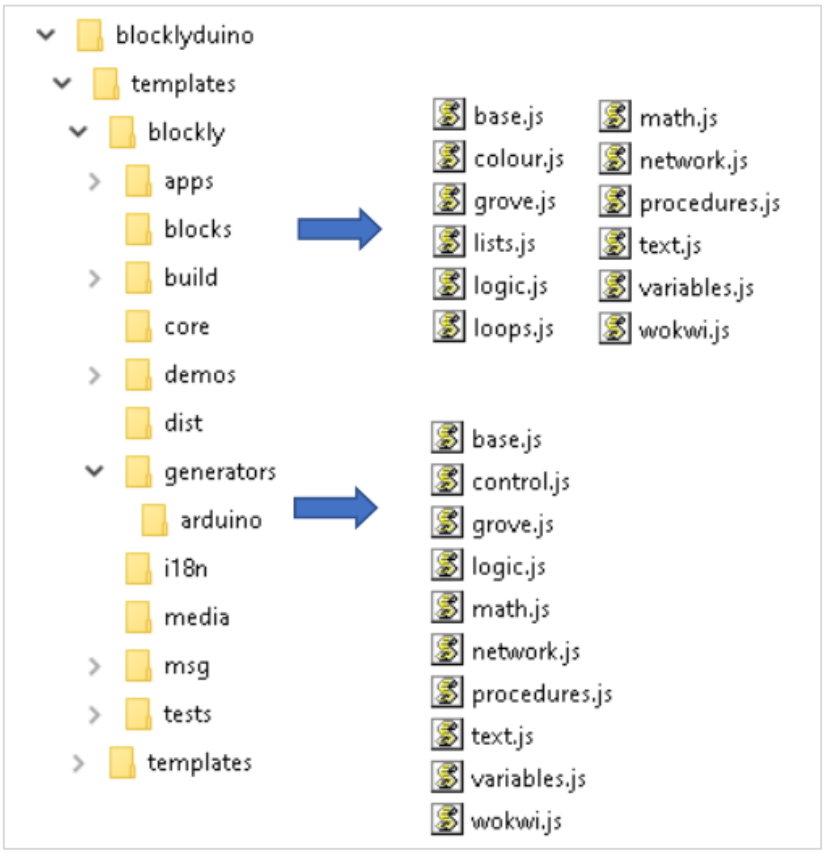


Figura 4.38. Archivos para desplegar el servicio blocklyduino.

El contenido de estos elementos es el siguiente:

- **templates**: Es una carpeta contiene los archivos del proyecto BlocklyDuino, mostrados en la siguiente imagen:

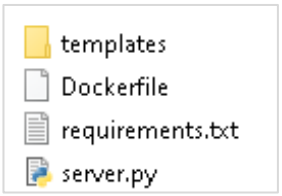


Figura 4.39. Archivos del proyecto BlocklyDuino.

- **Dockerfile**: Es un archivo que describe como se construye la nueva imagen. Los pasos para construir la nueva imagen son:
    1. Se copia el archivo requirements.txt (COPY) que contiene las bibliotecas utilizadas en el programa principal.
    2. Se instalan (RUN pip install).

3. Una vez instaladas las dependencias, se copia el programa principal (server.py) y el directorio /templates.

4. Utilizando el comando CMD python en la consola, se indica el nombre del programa principal que se ejecutará cuando se instancie un contenedor con la nueva imagen.

```
FROM python:3.7
MAINTAINER Zenobia Milla Herrero <zmh0009@red.ujaen.es>

COPY requirements.txt /home/servidor/requiremets.txt
RUN pip install -r /home/servidor/requiremets.txt

WORKDIR /home/servidor/
COPY ./server.py /home/servidor/server.py
COPY ./templates /home/servidor/templates
CMD python /home/servidor/server.py
```

Figura 4.40. Contenido del archivo Dockerfile.

- **requirements.tx:** Contiene las bibliotecas utilizadas en el programa server.py.
- **server.py:** Nombre del programa principal. Se trata de un servidor HTTP, implementado utilizando Python Flask, que recibe peticiones en el puerto 8080. Si recibe una petición solicitando el recurso /blockly, interpreta que debe enviar la interfaz de usuario del editor de programación basado en bloques y realiza una redirección interna para mostrar el contenido del archivo:

  templates/blockly/apps/blocklyduino/index.html.

5. RESULTADOS Y DISCUSIÓN

En este apartado se presentan distintas actividades prácticas y sus soluciones, para demostrar el correcto funcionamiento de los servicios desplegados, cómo la nueva extensión BlocklyDuino desarrollada.

Las actividades están relacionadas con conocimientos del ámbito de la Ingeniería de Telecomunicación, pero como se comentó en la introducción, la plataforma de servicios y los recursos presentados pueden adaptarse a otros estudios y niveles educativos.

5.1 Actividad control de la rotación de un servomotor

El objetivo de esta actividad es realizar una aplicación basada en la programación con bloques que simula el control de rotación de un servomotor utilizando un potenciómetro. Con Wokwi se simulará el comportamiento.

Tarea 1: Aplicación basada en bloques.

La aplicación creada con la extensión de BlocklyDuino se muestra en la Figura 5.1. Existe un bloque inicial Project, donde se ha seleccionado como dispositivo objetivo (Device) la opción Wokwi Arduino UNO. A continuación, se ha insertado el bloque Servo para controlar el servomotor y se ha seleccionado el pin A4 como señal de control del mismo. Se desea que el ángulo de rotación esté entre 0 y 180º, por tanto, el valor de salida del pin A4 toma valores entre [0, 180]. Para ajustar los valores de tensión medidos por el potenciómetro al rango de ángulos [0,180], se añade el bloque Map y para adquirir el valor de tensión del potenciómetro se usa el bloque Rotary Angle, seleccionando el pin A0.

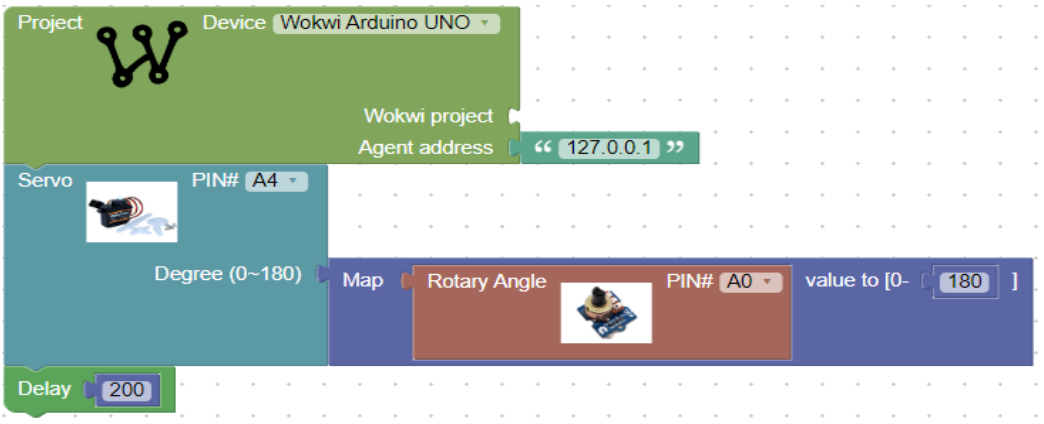


Figura 5.1. Programa con bloques para el control de la rotación de un servomotor.

Tarea 2: Simulación de la aplicación.

En el proceso de carga del código generado por los bloques, se abre una pestaña en el navegador con un nuevo proyecto en el simulador Wokwi. En él se conecta un potenciómetro con el terminal central unido al pin A0 y un servomotor unido al pin A4. Al realizar la simulación, se observa cómo al girar el potenciómetro va girando el servomotor fruto del control realizado a través del potenciómetro, como se puede observan en la Figura 5.2. El proyecto y el código fuente está disponible en: https://wokwi.com/projects/367317058521193473.

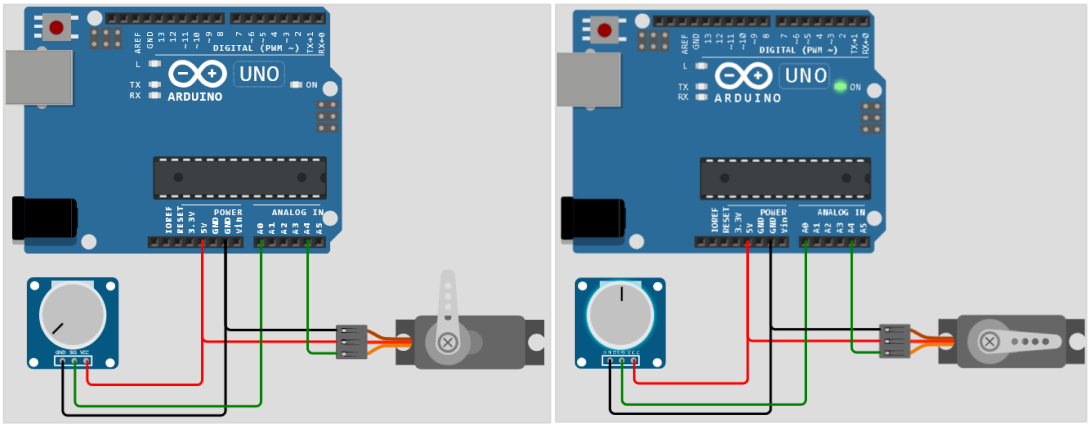


Figura 5.2. Rotación del servomotor controlado por un potenciómetro.

5.2 Actividad sensor de ultrasonidos

El objetivo de esta actividad es realizar una aplicación que sea capaz de medir la distancia a un objeto y mostrarla en una pantalla LCD. Si esta distancia es menor de 100 cm, debe activarse una señal acústica que alerte de la presencia cercana del objeto. En esta actividad, se hará uso de un proyecto Wokwi ya creado, introduciendo la posibilidad de utilizar diseños previamente realizados.

Tarea 1: Aplicación utilizando programación con bloques.

La aplicación creada se muestra en la Figura 5.3 Contiene un bloque Project donde se ha seleccionado la opción ‘Open existing Wokwi’ y se le indica la dirección donde se encuentra el proyecto Wokwi ya creado: https://wokwi.com/projects/367389353840070657.

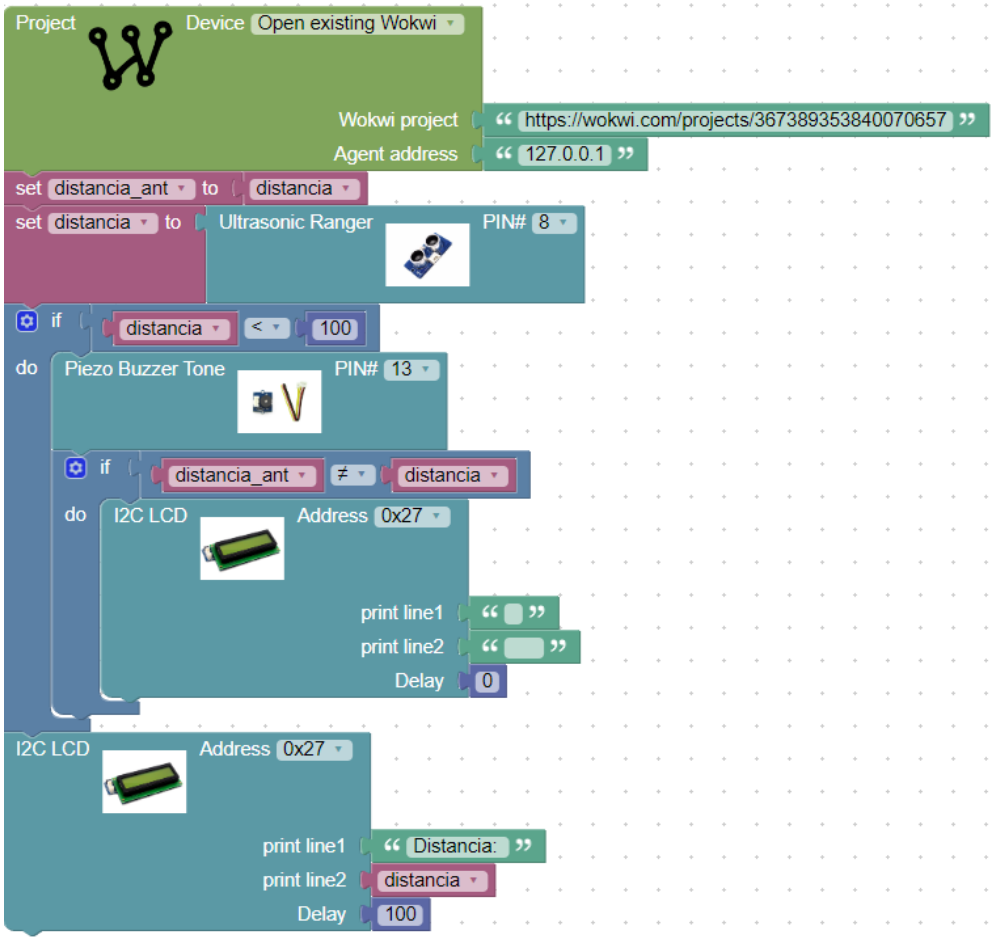


Figura 5.3. Bloques del programa.

El funcionamiento es el siguiente:

1. En la variable ‘distancia’ se almacena la lectura del sensor de ultrasonido, lectura que se obtiene utilizando el bloque Ultrasonic Ranger.
2. El sensor dispara un ultrasonido y recibe el eco. Calculando el tiempo entre estos eventos y conocida la velocidad del sonido, calcula la distancia mediante la siguiente ecuación:

   Distancia = (Tiempo entre TRIG y el ECHO) * (340 m/s)/2

   Cuando se usa el bloque Ultrasonic Ranger, hay que indicar el pin donde se conecta ECHO (pin 8) y asume que en que ese pin+1(pin 9) se encuentra conectado el TRIG.
3. Cuando la distancia es menor de 100 cm, se activa el zumbador (señal acústica) que está conectado en el pin 13 usando el bloque Piezzo Buzzer Tone. La primera vez que se cumple esta condición, se borra el contenido de la pantalla. Para realizar

el borrado, se ha utilizado el bloque I2C LCD y se ha colocado una cadena vacía en ‘print line1’ y ‘print line2’.

4. Cada vez que se adquiere una distancia, se muestra en la pantalla mediante otro bloque I2C LCD.

Tarea 2: Simulación de la aplicación.

Tras la carga del código generado con BlocklyDuino en Wokwi, hay que incluir la biblioteca: **LiquidCrystal I2C**, es biblioteca que permite controlar pantallas LCD I2C.

En la Figura 5.4 se muestra cómo se conectan los distintos elementos al Arduino UNO en el proyecto Wokwi. Un LCD I2C se conecta a los pines SCL y SDA (parte superior izquierda), el pin 13 a un zumbador, el pin 8 a la señal ECHO y pin 9 a la señal TRIG.
Al ejecutar la aplicación en el simulador, se obtiene un resultado como el de la Figura 5.4. En este caso, durante la simulación se ha manipulado la distancia del sensor al objeto, para que sea menor de 100cm, por lo que se ha activado una señal acústica para alertar de la situación.

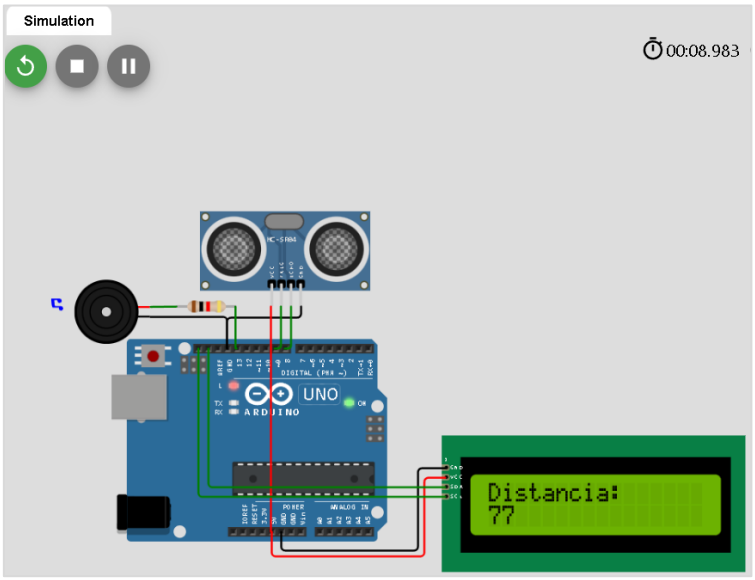


Figura 5.4. Emisión de señal acústica cuando el sensor detecta una distancia de 77cm.

5.3 Actividad *Publisher* MQTT para Arduino UNO WiFi

El objetivo de esta actividad es desarrollar una aplicación para Arduino UNO WiFi utilizando el editor de bloques. Esta aplicación, utilizando el protocolo MQTT, lee y publica el valor de distancia proporcionado por un sensor de ultrasonido en un tema. Estos datos serán leídos por un suscriptor creado con Node-RED y almacenados en una base de datos. Además, estos datos se enviarán a la plataforma Ubidots para su almacenamiento y representación gráfica.

Tarea 1: Aplicación de bloques para generar el código.

En la Figura 5.5 se muestra la aplicación donde se lee el valor de la distancia proporcionada por un sensor de ultrasonido y lo publica en un tema 'zenobia/sensor1'. Se indica el pin 8 como pin Echo en el sensor y se añade la variable 'distancia' para guardar la distancia recibida. Este valor será el publicado en el tema.

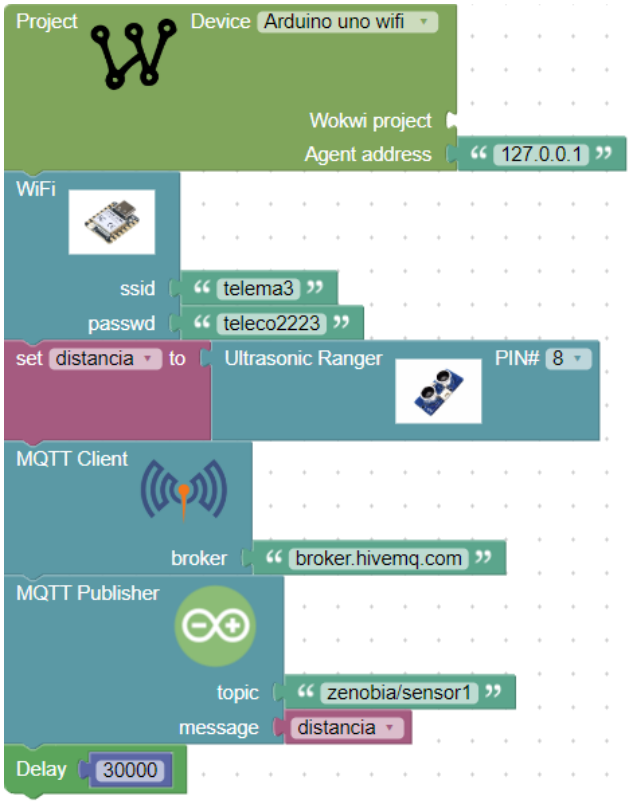


Figura 5.5. Bloques del programa 'publisher' para Arduino UNO WiFi.

Tarea 2: Ejecución de la aplicación.

Para realizar las tareas de compilación, carga y ejecución en el dispositivo físico, se pulsa el botón ‘Cargar’. El registro generado durante esas tareas se muestra en la siguiente figura:

Figura 5.6. Registro de tareas de compilación del código y carga en Arduino UNO WiFi.

Para comprobar que los mensajes son publicados en el tema. Se ha creado en Node-RED una aplicación como la ilustrada en la Figura 5.7. Está compuesta por un nodo ‘MQTT in’ que recibe los mensajes publicados en el tema ‘zenobia/sensor1’. En la Figura 5.8 se muestra la ventana de depuración con los mensajes recibidos cada 30 segundos.

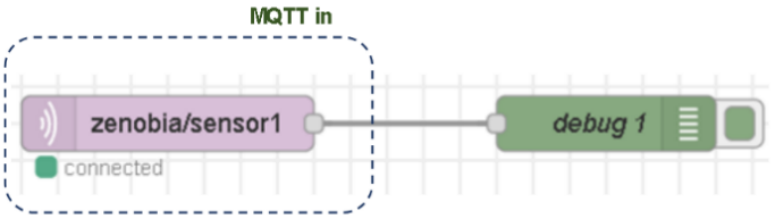


Figura 5.7. Aplicación Subscriber creada con Node-RED.

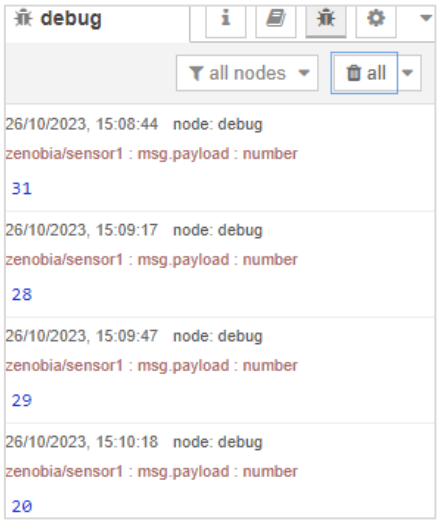


Figura 5.8. Mensajes recibidos en la ventana de depuración.

5.4 Almacenamiento de datos capturados por un sensor de temperatura y humedad

El objetivo de la actividad es desplegar una aplicación en un microcontrolador ESP32 simulado y una aplicación de servidor Node-Red con las siguientes funcionalidades:

- La aplicación ESP32 realiza una adquisición de datos utilizando un sensor DHT22 cada 30 segundos. Los valores de temperatura y humedad obtenidos se publican en formato JSON en un tema utilizando el protocolo MQTT.
- La aplicación Node-RED implementa un suscriptor MQTT que recibe los datos enviados al tema, y los almacena en una base de datos.
- Los datos recibidos por el suscriptor, se envían a la plataforma Ubidots, especializada en el desarrollo de soluciones IoT.

Tarea 1: Adquisición y publicación de los datos en el tema 'zenobia/sensor1'.

El diagrama de bloques de la aplicación encargada de la adquisición de datos cada 30 segundos y publicación en el tema MQTT se muestra en la Figura 5.9. El dispositivo elegido es un ESP32 simulado. Se realiza la conexión al punto de acceso WiFi usando el bloque WiFi y a continuación se establece una conexión con el bróker MQTT. Cada 30 segundos se adquieren los datos y publican en el tema. Se ha indicado que el pin de datos del sensor de temperatura es el número 15.

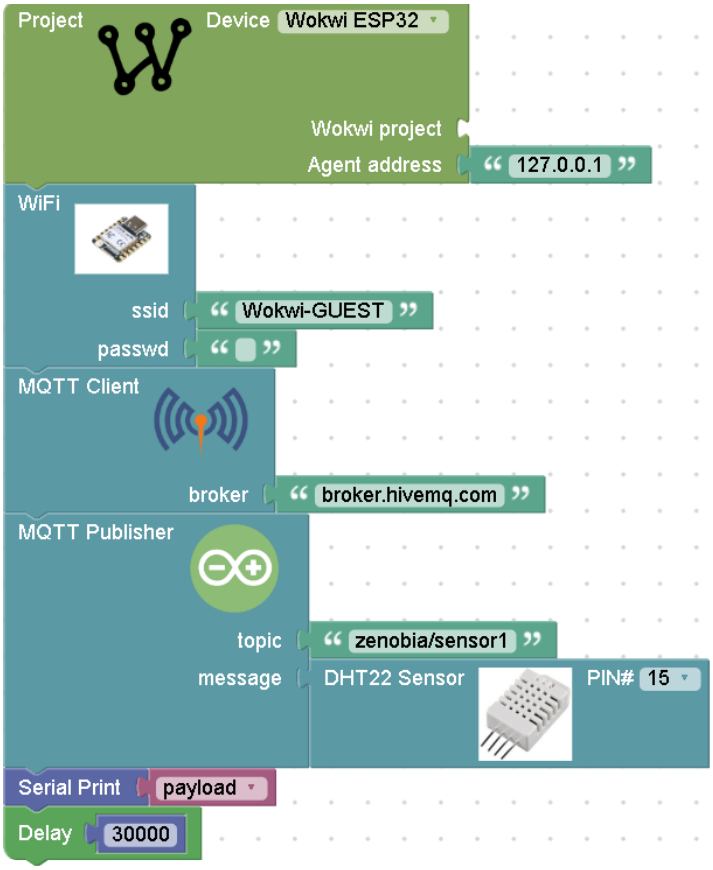


Figura 5.9. Bloques del programa para ESP32.

Tarea 2: Simulación de la aplicación encargada de la adquisición y publicación de los datos.

En el proceso de carga del código generado por los bloques, se abre una pestaña en el navegador con un nuevo proyecto ESP32 en el simulador Wokwi. En él se incluyen los elementos mostrados en la Figura 5.10. Al microcontrolador ESP32 se conecta un sensor DHT22 (para adquirir datos de temperatura y humedad). Con la idea de analizar las señales del sensor, se conecta un analizador lógico. El proyecto completo está disponible en la dirección: https://wokwi.com/projects/367316171157381121.

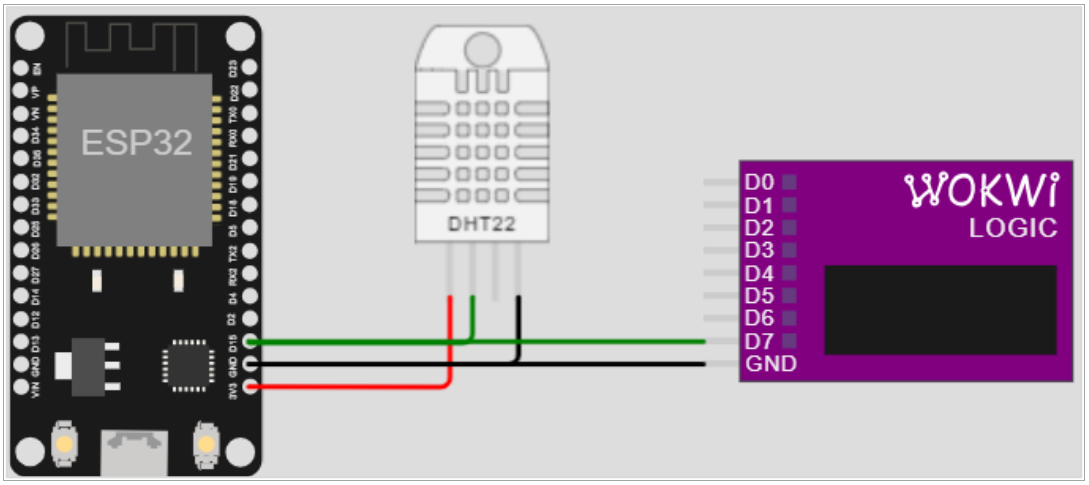


Figura 5.10. ESP32 conectado a un sensor y a un analizador lógico.

Las conexiones entre los elementos son las siguientes:

- Se conecta el pin de GND del EPS32 al pin GND del sensor y al pin GND del analizador.
- El pin D15 del ESP32 se conecta al pin SDA del sensor DHT22 y al pin D7 del analizador lógico.
- El pin VCC del sensor se conecta al pin 3,3V del ESP32.

El analizador tiene 8 canales digitales y cada uno de ellos tiene un LED de actividad, para comprobar que las señales están conectadas correctamente. Tiene una frecuencia de muestreo de 1 GHz. Cuando finaliza la simulación, se almacenan las muestras capturadas en un archivo en formato VCD (Value Change Dump), que es un formato basado en ASCII para archivos generados por herramientas de simulación lógica.

Para realizar la simulación hay que incluir las siguientes bibliotecas en la pestaña Library Manager:

- **ArduinoJson**. Permite convertir objetos a formato JSON, así como la serialización de documentos JSON a documentos sin espacios ni saltos de línea entre valores (JSON minimizado). También permite la deserialización.
- **PubSubClient**. Necesaria crear un cliente MQTT, que se conecta a un bróker y publica los datos del sensor en un tema.
- **DHT sensor library for ESPx**. Es una biblioteca para sensores de humedad y temperatura DHT11 y DHT22.

Cuando una simulación está en curso, se puede pulsar sobre el sensor DHT22 y modificar los valores de temperatura y humedad (Figura 5.11). Así mismo, si se pulsa sobre el icono de conexión WiFi (Figura 5.12) y descargar un archivo en formato PCAP que contiene el tráfico capturado en esa interfaz.

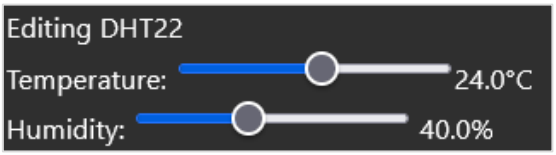


Figura 5.11. Controles para modificar los valores de temperatura y humedad.

Figura 5.12. Descarga del archivo con la captura de tráfico en la interfaz WiFi.

Si accede al archivo descargado con formato PCAP, se puede realizar un filtrado por protocolo MQTT (Figura 5.13) y visualizar los mensajes. Se comprueba como existe un mensaje ‘Connect Command’ enviado al bróker (para solicitar la conexión) y cómo el bróker le ha contestado con un ‘Connect Ack’. A continuación, el microcontrolador envía un ‘Publish Message’ para publicar contenido en el tema ‘zenobia/sensor1’.

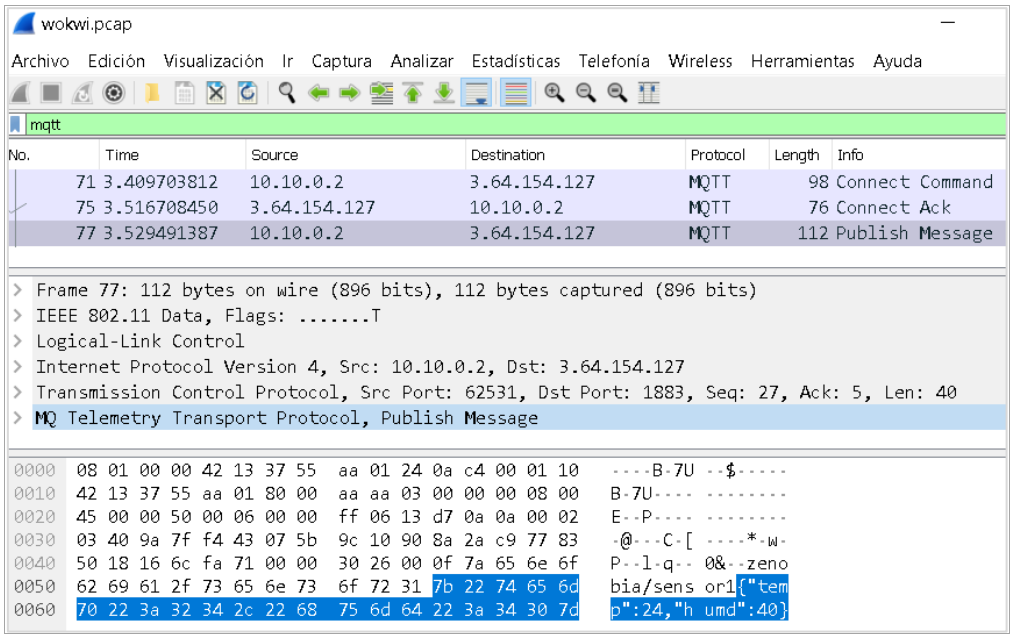


Figura 5.13. Mensajes pertenecientes al protocolo MQTT.

Cuando se detiene la simulación, el archivo wokwi-logic.vcd contiene las muestras capturadas. Para visualizar los datos de ese archivo, se utiliza la aplicación PulseView. La señal grabada en el archivo se importa haciendo uso de la opción 'Import Value Change Dump Data…' dentro del menú 'Open' como muestra la figura siguiente:

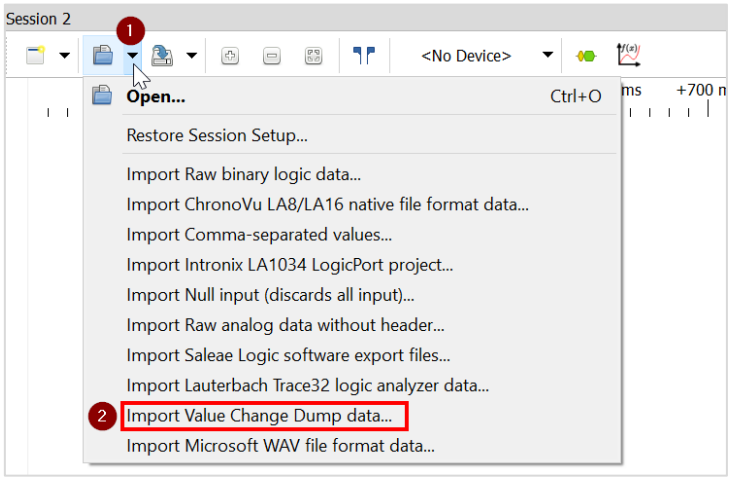


Figura 5.14. Importado de datos VCD a la aplicación PulseView.

A continuación, aparece un cuadro de diálogo como el mostrado en la Figura 5.15. Las opciones predeterminadas generalmente hacen que PulseView consuma mucha RAM y se vuelva lento. Se puede reducir el uso de la memoria configurando el factor de reducción de muestreo. Un valor de 50 debería ser suficiente para la mayoría de los casos de uso.

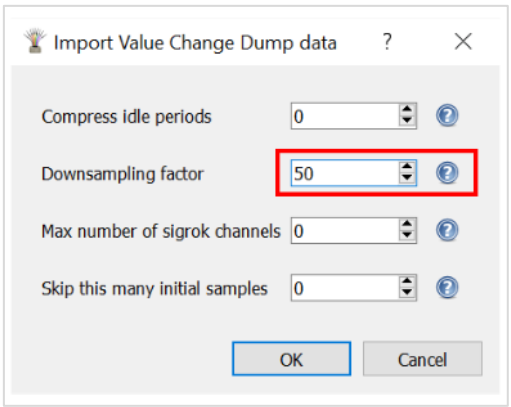


Figura 5.15. Factor de reducción de muestreo.

Después de confirmar los valores de la Figura 5.15, se muestra la señal en la pantalla. Además, se ha incorporado un decodificador de protocolo para señales de

sensores basadas en AM230x/DHTxx. De esta forma, se puede comprobar fácilmente cómo se realiza la transmisión de un dato de temperatura codificado con 2 bytes y otro de humedad, también codificado con 2 bytes. En el ejemplo de la Figura 5.16, los datos son 40% de humedad y 24ºC de temperatura.

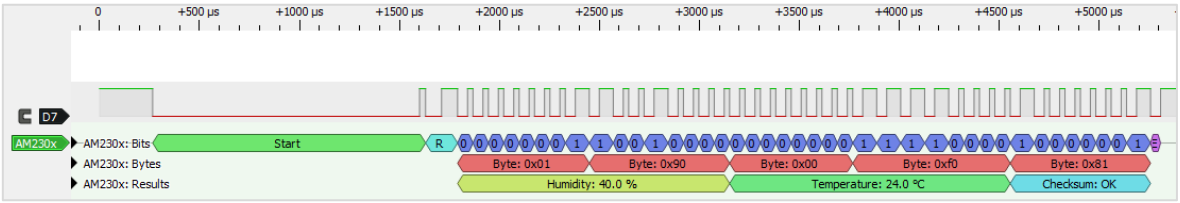


Figura 5.16. Lectura de los mensajes recibidos en el tema ‘zenobia/sensor1’.

Tarea 3: Almacenar los datos en un base de datos.

Se realiza una aplicación en Node-RED con los nodos necesarios para suscribirse al tema ‘zenobia/sensor1’ y almacenar los datos recibidos en una base de datos, tal y como se muestra en la Figura 5.17.

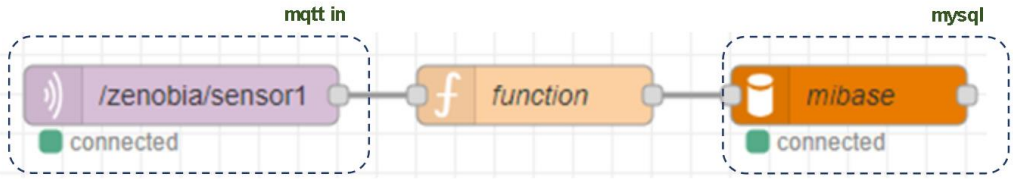


Figura 5.17. Almacenamiento de datos recibidos en una base de datos.

Para realizar la suscripción, se utiliza el nodo ‘mqtt in’ cuya función es recibir los mensajes publicados en un tema. Se pulsa dos veces sobre él y se accede a sus propiedades. Se añade la información necesaria para la conexión con el bróker MQTT (Figura 5.18), indicando la dirección IP o nombre donde está instalado el bróker MQTT y el puerto que utiliza. Una vez dada de alta la información del bróker, introducimos el valor del campo Topic, que en este caso es ‘zenobia/sensor1’.

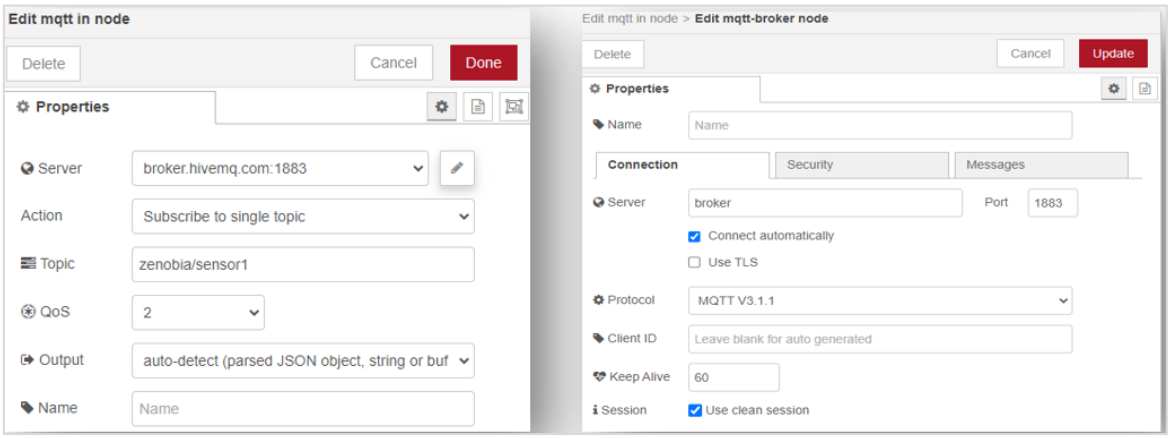


Figura 5.18. Configuración del nodo mqtt in.

Para utilizar un nodo que almacene los datos en la base de datos, hay que instalar la librería ‘node-red-node-mysql’. Para ello, se pulsa en el icono con tres líneas horizontales situado en la esquina superior derecha, se selecciona la opción ‘Manege palette’ y se accede a la pestaña ‘Install’ donde se busca la librería y se instala.

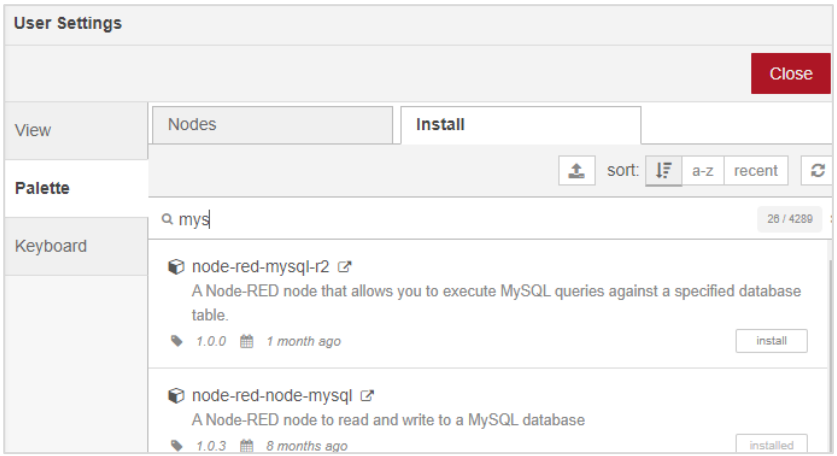


Figura 5.19. Instalacion de la librería ‘node-red-node-mysql’.

A Continuación, se accede al administrador de bases de datos y se selecciona la base de datos ‘medidas’. Se crea una tabla llamada ‘sensor1’ con los siguientes atributos: id, temp, humd, time. Seleccionamos el atributo *id* como clave primaria:

#	Nombre	Tipo	Nulo	Predeterminado	Extra
1	**id**	int(11)	No	*Ninguna*	AUTO_INCREMENT
2	**temp**	float(5,2)	No	*Ninguna*	
3	**humd**	float(5,2)	No	*Ninguna*	
4	**time**	timestamp	No	CURRENT_TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP

Figura 5.20. Atributos de la tabla sensor1.

Una vez que se tiene la tabla creada en la base de datos, se edita el nodo 'mysql' para realizar operaciones en la base de datos. En propiedades (Figuras 5.21 y 5.22) se añade la información necesaria para la conexión con el gestor de bases de datos MySQL.

Estos datos de conexión son:

Host	Dirección del gestor de base de datos (db)
Port	Puerto (3306)
User	Nombre de usuario
Password	Contraseña
Database	Nombre de la base de datos (medidas)

Tabla 5.1. Datos para la conexión con el gestor de datos MySQL.

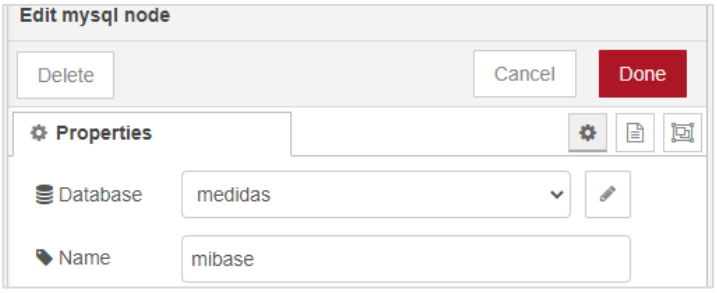


Figura 5.21. Configuracion del nodo myql.

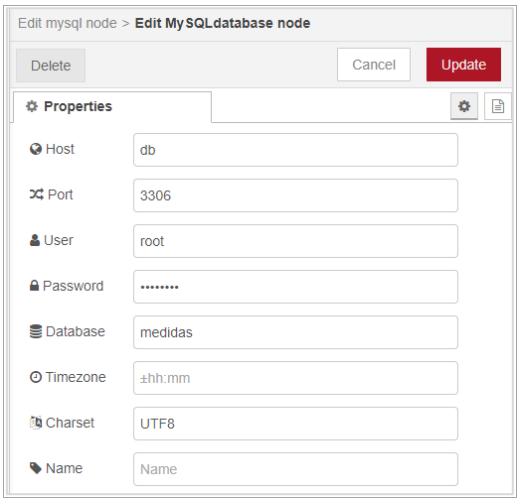


Figura 5.22. Parametros de conexión.

Para insertar la información en la base de datos hay que enviar una sentencia SQL al gestor de bases de datos. Esto lo realiza el nodo intermedio ‘function’, que se ha insertado entre ‘mqtt in’ y ‘mysql’. Este nodo se edita y se incluye el código para crear la sentencia INSERT, asignándola como valor de la propiedad ‘msg.topic’ (Figura 5.23).

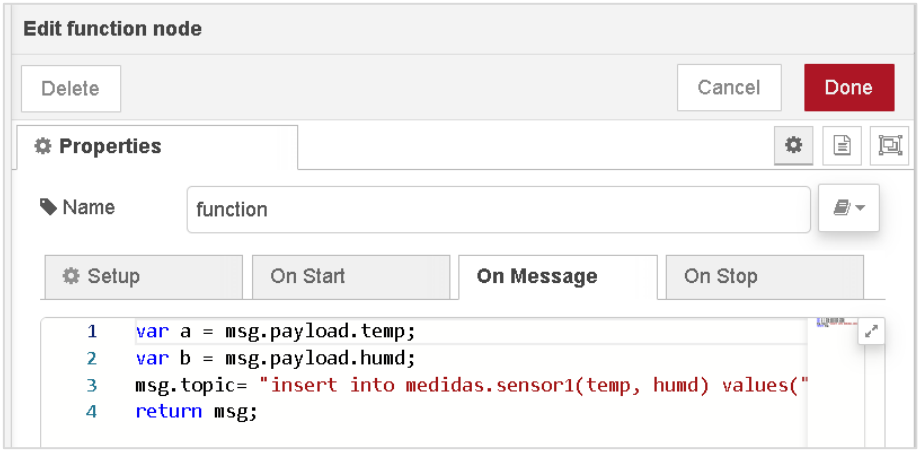


Figura 5.23. Función que construye la secuencia SQL.

Cuando el suscriptor recibe un nuevo dato, la función extrae los valores, se construye la sentencia SQL y el nodo ‘mysql’ inserta la información en la tabla sensor1 de la base de datos medidas.

id	temp	humd	time
11	24.90	40.00	2023-07-19 13:42:11
12	25.40	40.00	2023-07-19 13:42:44
13	25.30	40.00	2023-07-19 13:43:18
14	25.50	40.00	2023-07-19 13:43:48
15	25.70	40.00	2023-07-19 13:44:18

Figura 5.24. Contenido de la tabla sensor1.

Tarea 4: Envío de los datos del sensor a la plataforma Ubidots.

Para enviar datos a Ubidots, en Node-RED se instala el paquete 'ubidots-nodered' y se añade el nodo 'Ubidots out' al flujo creado en la tarea 3, como se muestra en la Figura 5.25. Se edita este nuevo nodo, para incluir el valor de 'Token' y 'Device Label' que proporciona la plataforma Ubidots (Figura 5.26).

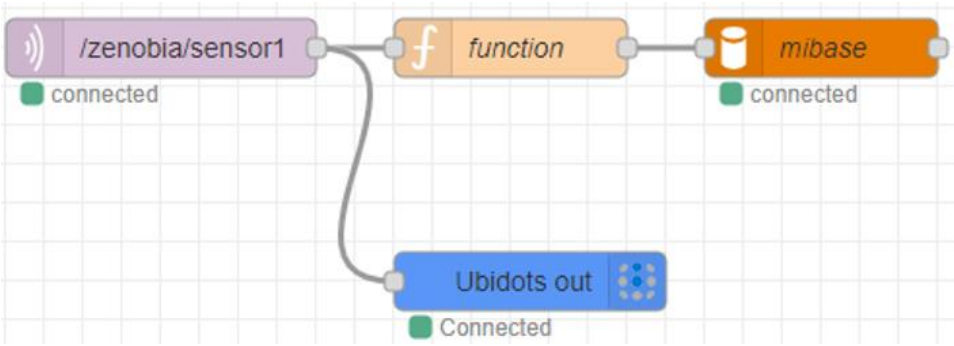


Figura 5.25. Envío de los datos recibidos a la plataforma Ubidots.

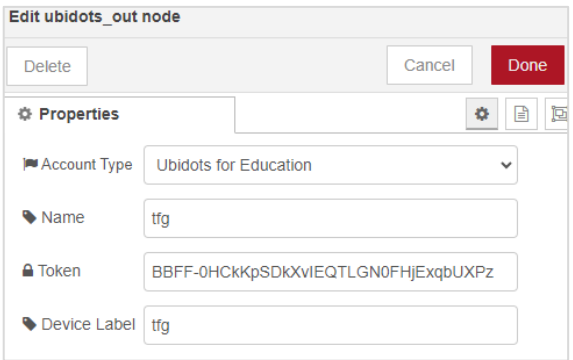


Figura 5.26. Configuracion nodo Ubidots out.

Para poder hacer uso de la plataforma Ubidots, se accede al sitio web de Ubidots en el siguiente enlace: https://stem.ubidots.com/accounts/signin/. Después de iniciar sesión, se añade un dispositivo pulsando en 'Devices' (Figura 5.27) y a continuación se pulsa en 'add new Device' para añadir un nuevo dispositivo. De los distintos tipos de dispositivos que se muestran, se elige 'Blank Device' y se asigna el nombre 'tfg' (Figura 5.28).

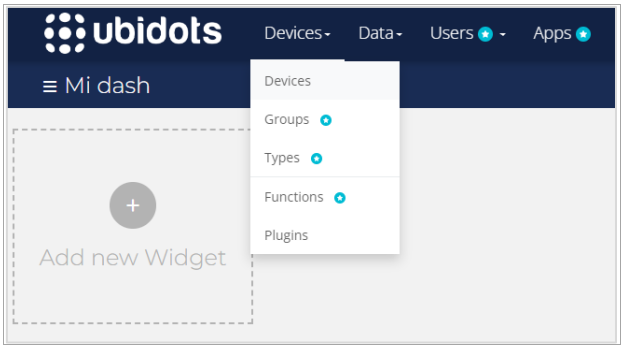


Figura 5.27. Menú Dispositivos.

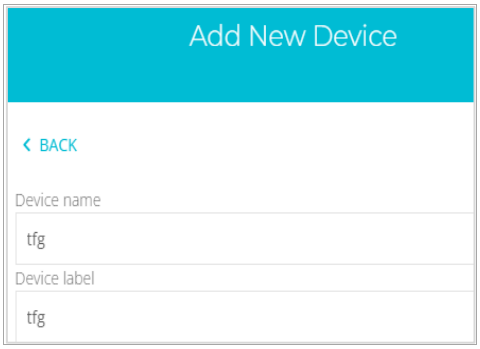


Figura 5.28. Nombre y etiqueta del nuevo dispositivo.

Una vez añadido, aparece el dispositivo en la lista 'Devices' (Figura 5.29). Al pulsar sobre él, se muestran sus propiedades (Figura 5.30), donde aparece los datos 'Token' y 'Api Label', necesarios para la configuración del nodo 'ubidots_out' que se mostró en la Figura 5.26.

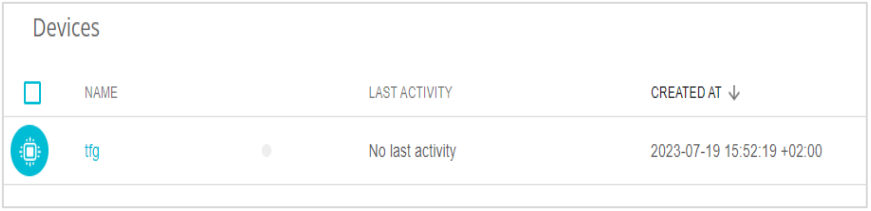


Figura 5.29. Lista de dispositivos.

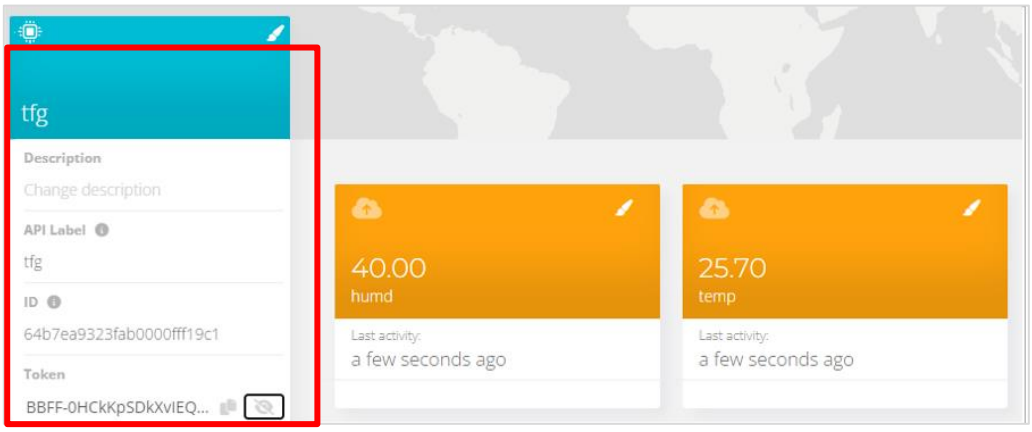


Figura 5.30. Propiedades del dispositivo ‘tfg’.

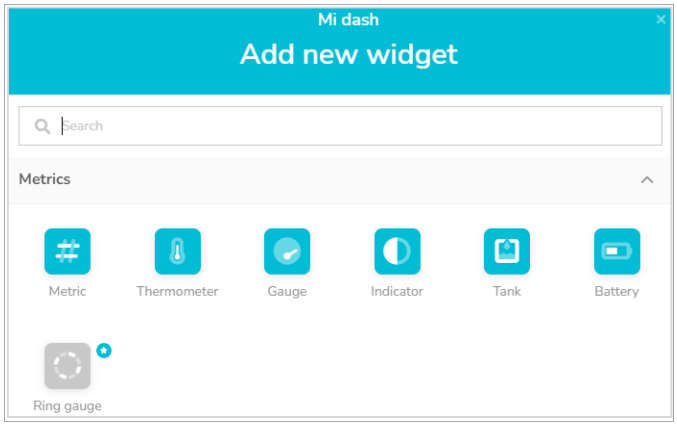


Figura 5.31. Lista de widgets.

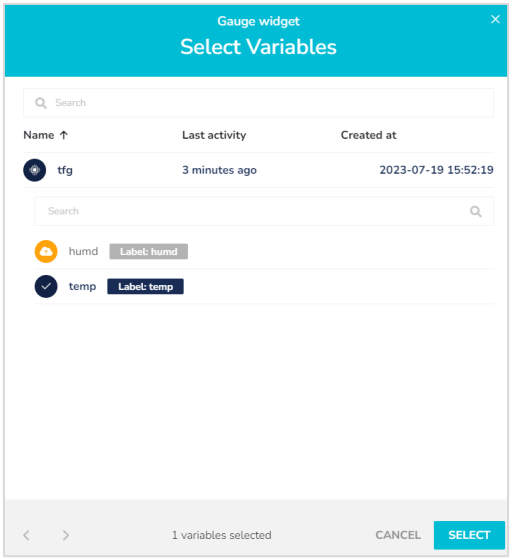


Figura 5.32. Selección de la variable que se quiere visualizar.

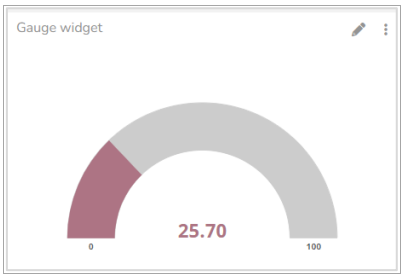


Figura 5.33. Visualización del valor recibido.

5.5 Envío de los datos de un sensor de temperatura y humedad a un servicio web

El objetivo de la actividad es desplegar una aplicación en el microcontrolador que se encarga de adquirir datos de un sensor y enviarlos a un servicio web remoto desarrollado con Node-RED.

Tarea 1: Adquisición y envío de los datos adquiridos al servicio web.

El diagrama de bloques de la aplicación encargada de la adquisición de datos cada 30 segundos y enviarlos al servicio web se muestra en la Figura 5.34. El dispositivo elegido es un ESP32 simulado. En el bloque HTTP Request hay que indicar que el método HTTP utilizado es POST, la URL del servicio que recibe los datos y el mensaje que enviará en el campo de datos del mensaje HTTP. En este caso, los datos enviados son los proporcionados por un sensor DHT22 cuya señal de datos se encuentra conectada al pin 15 del microcontrolador.

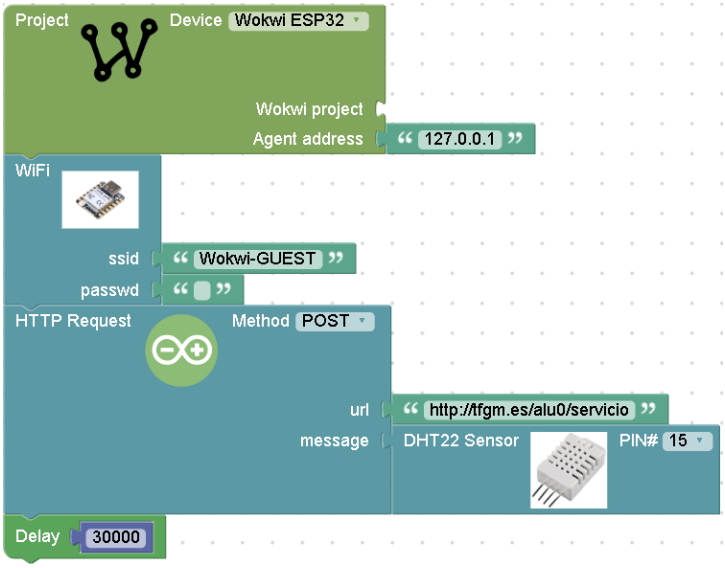


Figura 5.34. Bloques del programa para ESP32.

Tarea 2: Servicio web.

En Node-RED se crea el servicio web utilizando los nodos ‘http in’ y ‘http response’ (Figura 5.35). El nodo ‘http in’ crea un punto final HTTP para crear servicios web. Se debe seleccionar el método (GET/POST) y la URL (Figura 5.36). El ‘nodo http’ response envía respuestas a las solicitudes recibidas por un nodo ‘http in’.

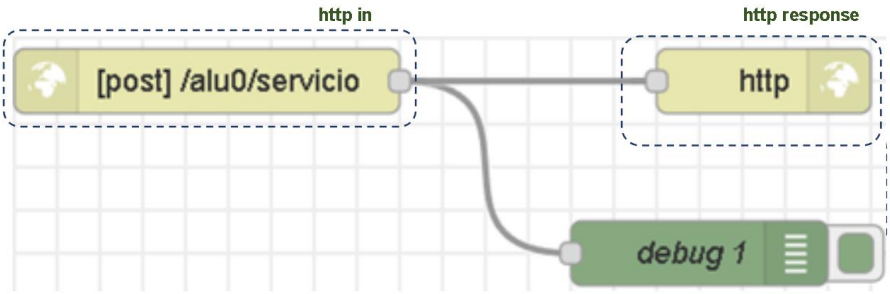


Figura 5.35. Servicio web con Node-RED.

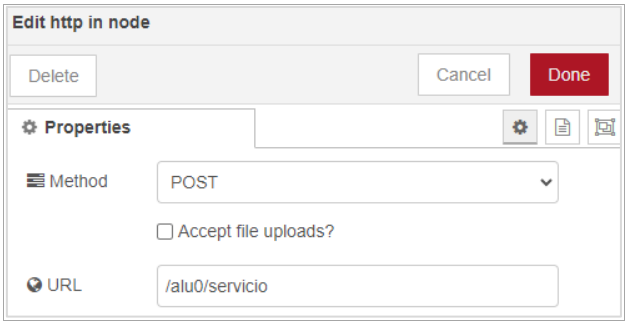


Figura 5.36. Configuración nodo http in.

Tarea 3: Simulación de la aplicación encargada de la adquisición y envío de los datos.

En el proceso de carga del código generado por los bloques, se abre una pestaña en el navegador con un nuevo proyecto ESP32 en el simulador Wokwi y en él se incluyen los elementos mostrados en la Figura 5.37. El proyecto completo, con el código, está disponible en la dirección: https://wokwi.com/projects/367316153947100161.

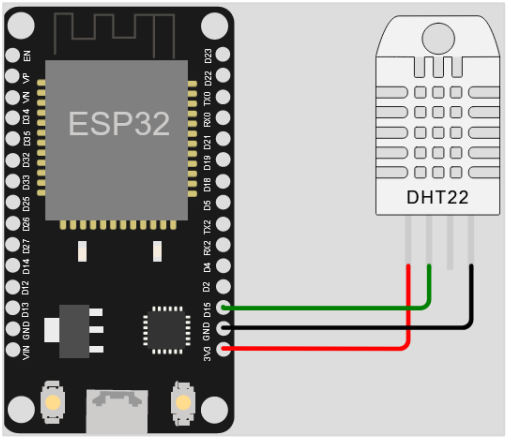


Figura 5.37. Conexión de un sensor DTH22 al microcontrolador ESP32.

Hay que incluir las siguientes bibliotecas en la pestaña Library:

- **ArduinoJson**
- **DHT sensor library for ESPx**

Para probar el funcionamiento del servicio, en Wokwi se inicia la simulación. En el área de monitor serie (Figura 5.38) se muestra el mensaje de conexión a WiFi, los datos leídos por el sensor y el código de respuesta HTTP que devuelve el servicio. En este caso es un valor igual a 200, ya que el envío de datos usando HTTP POST ha tenido éxito.

```
Conectando a Wifi....¡¡¡Conectado!!!. IP asignada:10.10.0.2
{"temp":24,"humd":40}
200
```

Figura 5.38. Estado del envío de datos utilizando un mensaje HTTP POST.

Si se filtra el tráfico capturado en la interfaz WiFi, como se muestra en la Figura 5.39, se pueden identificar los mensajes de petición HTTP enviados por el microcontrolador y la respuesta que devuelve el servicio web. En el campo de datos del mensaje de petición HTTP POST se encuentran los datos codificados en formato JSON.

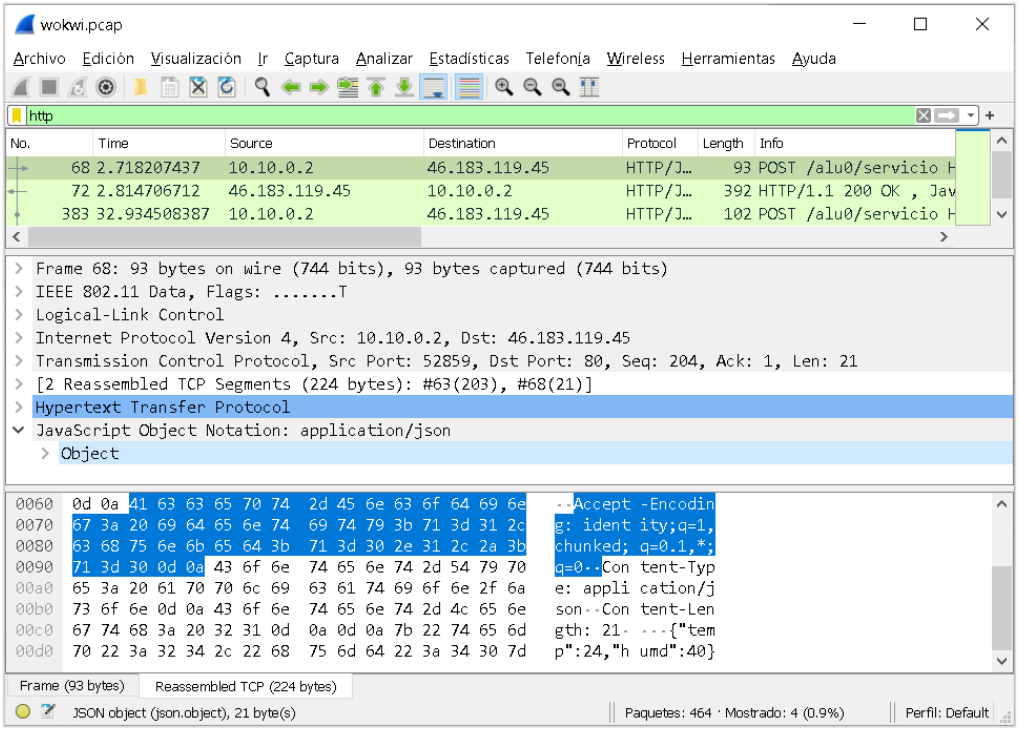


Figura 5.39. Tráfico HTTP capturado en la interfaz WiFi.

5.6 Pasarela Zigbee a MQTT

El objetivo de esta actividad es ejecutar un suscriptor MQTT en el microcontrolador cuyo objetivo es recoger datos que provienen de una pasarela zigbee2MQTT. En función del dato recibido, muestra uno de los siguientes mensajes de estado en una pantalla LCD:

- ‘Luces encendidas’.
- ‘Luces apagadas’.
- ‘Música activada’.
- ‘Música apagada’.

Tarea 1: Crear la aplicación de bloques.

Los bloques de la aplicación suscriptor se muestran en la Figura 5.40. Se trata de un suscriptor ya realizado en anteriores actividades, al que se añadirá un código adicional para poder analizar el mensaje recibido y en función de su contenido, mostrar el mensaje de estado en el LCD. En tema donde recibe los datos es zigbee2mqtt/pulsador.

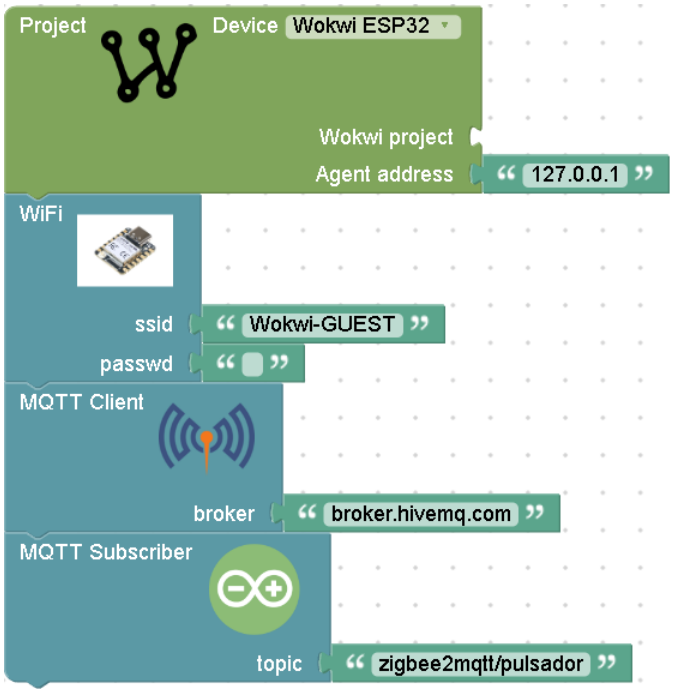


Figura 5.40. Bloques de la aplicación.

Tarea 2: Análisis del mensaje recibido y simulación.

En el proceso de carga del código generado por los bloques, se abre una pestaña en el navegador con un nuevo proyecto ESP32 en el simulador Wokwi. Se incorpora un LCD I2C de 16X2 como el mostrado en Figura 5.41. El proyecto completo está disponible en la dirección: https://wokwi.com/projects/379813610179410945.

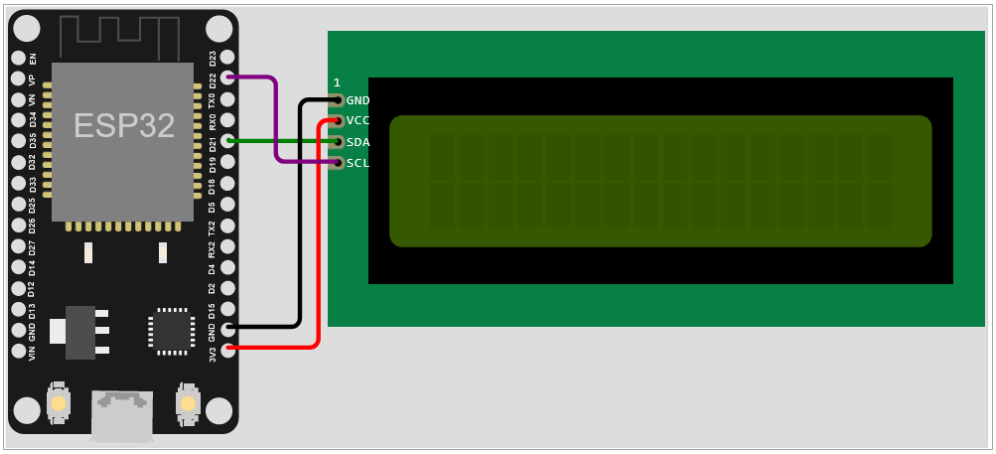


Figura 5.41. Conexión del microcontrolador ESP32 con el LCD I2C.

Para realizar la prueba de funcionamiento, se ha utilizado la aplicación zigbee2mqtt desplegada en un contenedor Docker. En esta actividad, se ha utilizado un pulsador Zigbee Xiaomi que puede reconocer una pulsación (single), dos seguidas (double), tres seguidas (triple) y una pulsación larga (long). La información del tipo de pulsación realizada se inserta en los mensajes Zigbee que son traducidos a los siguientes mensajes MQTT y publicados en el tema 'zigbee2mqtt/pulsador' (Figura 5.42).

Figura 5.42. Cuatro posibles mensajes enviados por el pulsador.

Cuando la pasarela zigbee2mqtt publica un nuevo mensaje en el tema 'zigbee2mqtt/pulsador', este es recibido por el suscriptor del ESP32. Los mensajes recibidos tienen un formato JSON y contienen un parámetro llamado 'click', con uno de los siguientes valores:

- **single**. Si se recibe este valor, se mostrará en la pantalla el mensaje 'Luces encendidas'.
- **double**. En este caso se muestra por pantalla el mensaje 'Luces apagadas'.

- **triple**. El mensaje mostrado será ‘Música activada’.
- **long**. Se mostrará el mensaje ‘Música pagada’.

En el editor de Wokwi, hay que añadir un código para analizar estos mensajes al generado por los bloques:

1. Al inicio del programa se incluyen las librerías necesarias:

```
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <LiquidCrystal_I2C.h>
```

2. Añadimos la linea para iniciar el display:

```
LiquidCrystal_I2C LCD = LiquidCrystal_I2C(0x27, 16, 2);
```

3. En la función **onMensaje** se incluye el siguiente código:

```
DynamicJsonDocument json(100);
deserializeJson(json, mensaje);
JsonObject obj = json.as<JsonObject>();
String valor = obj["click"].as<String>();
if(valor.equals("single")){
    LCD.clear();
    LCD.print("Luces encendidas");
}
if(valor.equals("double")){
    LCD.clear();
    LCD.print("Luces apagadas");
}
if(valor.equals("triple")){
    LCD.clear();
    LCD.print("Musica activada");
}
if(valor.equals("long")){
    LCD.clear();
    LCD.print("Musica apagada");
}
```

I2C es un protocolo síncrono donde se usan dos cables: uno para el reloj (SCL) y otro para los datos (SDA). Tanto el maestro como el esclavo envían datos por el mismo cable, el cual es controlado por el maestro (que crea la señal de reloj). I2C no utiliza selección de esclavo, sino direccionamiento. La dirección I2C por defecto del módulo puede ser 0x3F o en otros casos 0x27 (esta es la dirección utilizada en Wokwi). Para conectar con el microcontrolador se utilizan los pines I2C (SDA y SCL) y alimentación (GND y 5V). Los pines I2C pueden variar y en Wokwi son: SDA pin D21 y SCL pin D22.

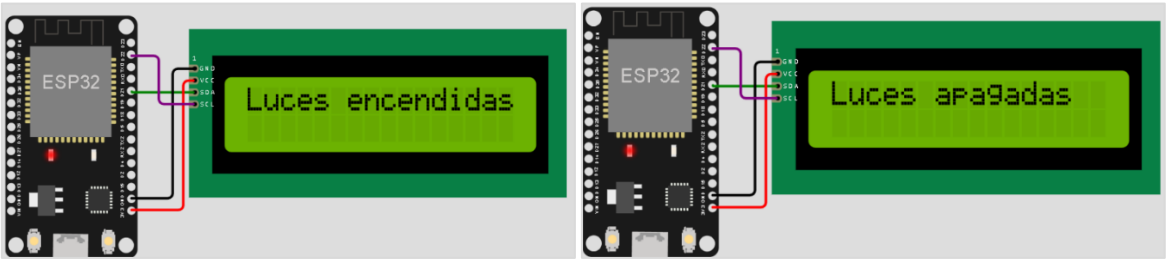


Figura 5.43. Luces encendidas/ apagadas.

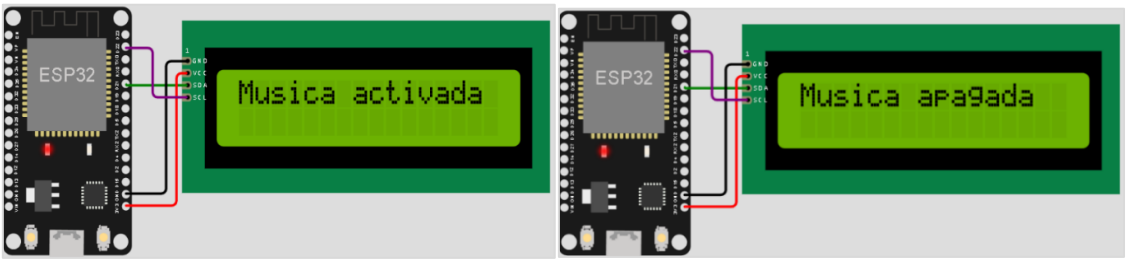


Figura 5.44. Música activada/apagada.

5.7 Consulta de datos medidos por un sensor utilizando un bot de Telegram

El objetivo de esta actividad es crear un bot de Telegram que permite consultar datos adquiridos por un sensor de temperatura y humedad DHT22.

Tarea 1: Nuevo bot de Telegram.

Para crear un bot de Telegram, se accede a un bot especial de Telegram conocido como ‘BotFather’. Se inicia una conversación con el comando /start, y BotFather nos indica los comandos disponibles. Con el comando /newbot se crea un nuevo bot llamado ‘lab4u’. Al final del proceso de creación, consultamos el API token de acceso necesaria para enviar y recibir mensajes en el nuevo bot de Telegram.

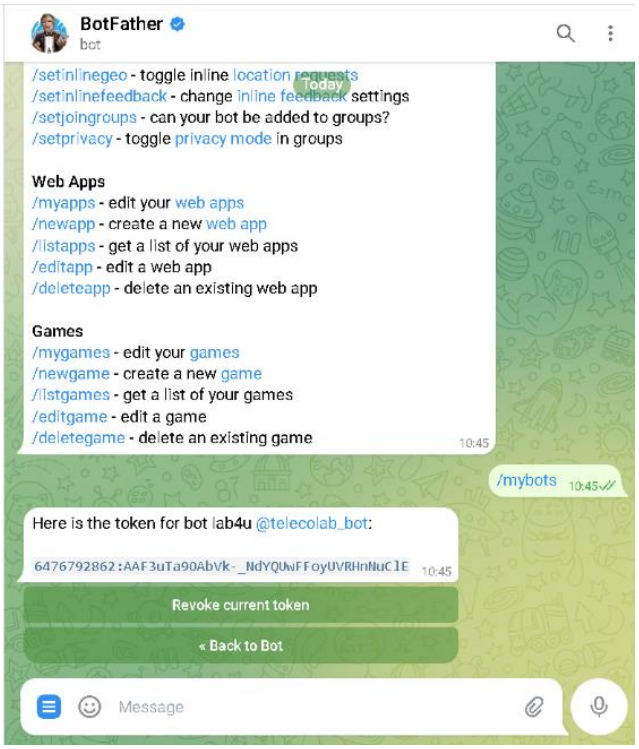


Figura 5.45. API token de acceso del bot lab4u.

Tarea 2: Aplicación de bloques.

Los bloques de la aplicación bot se muestran en la Figura 5.46. En el bloque 'Telegram Bot' se indica el token del bot y en él se incluyen todos los bloques 'Bot Message' encargados de atender a los comandos recibidos por el bot. En este ejemplo, cuando se recibe el comando '/start' se devuelve el mensaje 'Bienvenido al bot' y en caso de recibir el comando 'info', se envían los valores actuales de temperatura y humedad medidos por el sensor.

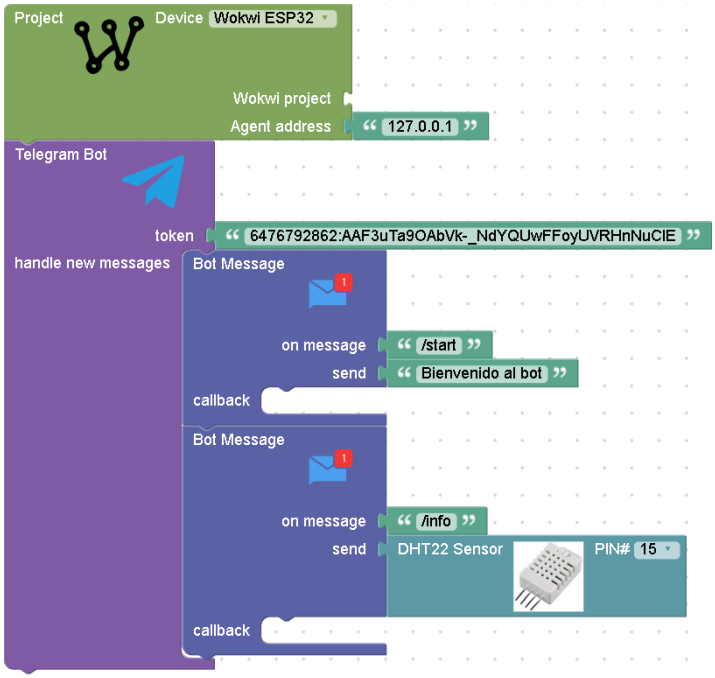


Figura 5.46. Bloques de la aplicación bot.

Tarea 3: Simulación de la aplicación.

En el proceso de carga del código generado por los bloques, se abre una pestaña en el navegador con un nuevo proyecto ESP32 en el simulador Wokwi y se incluye el sensor DHT22 (Figura 5.47). Además, en el código hay que incluir las siguientes bibliotecas: **ArduinoJson**, **DHT sensor library for ESPx** y **UniversalTelegramBot**, utilizada para para crear bots de Telegram. El proyecto completo con el código está disponible en la dirección: https://wokwi.com/projects/367316136602603521.

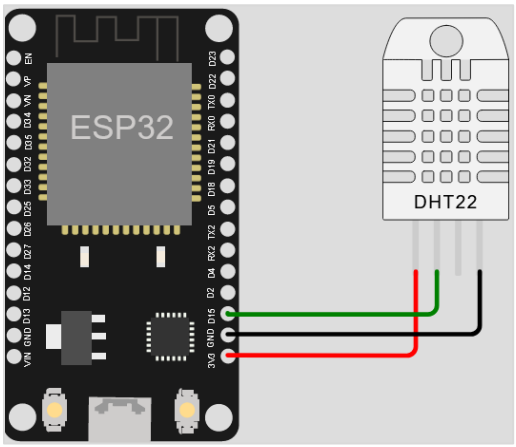


Figura 5.47. Conexión de un sensor DTH22 al microcontrolador ESP32.

Una vez creada la aplicación, se inicia la simulación y se accede al bot creado. El primer comando que se envía es /start, y se recibe el mensaje de bienvenida. A continuación, se envía el comando /info, el bot lee los datos de temperatura y humedad y los devuelve al usuario.

Figura 5.48. Ejemplo de conversación con el bot (aplicación ESP32).

5.8 Despliegue de un servicio web para consultar los datos medidos por un sensor

El objetivo de esta actividad es desplegar dos servicios web en un microcontrolador ESP32 con las siguientes funcionalidades:

- Servicio **/info**. Devuelve los valores actuales de temperatura y humedad medidos por un sensor. La ubicación del servicio es http://localhost:9080/info.
- Servicio **/cambia**. Modifica el estado de un led conectado al microcontrolador, conmutando entre apagado y encendido. La URL del servicio es http://localhost:9080/cambia.

Tarea 1: Bloques de la aplicación.

Los bloques que implementan los servicios se muestran en la Figura 5.49. Para implementar cada servicio se utiliza el bloque 'Web Server'. En ellos se indica el nombre

del recurso que se solicita y la respuesta que devuelve. En el caso del servicio /cambia, el mensaje de respuesta es ‘200 ok’ y además ejecuta una función llamada ‘cambia’. Esta función se encarga de ir cambiando el estado de la variable ledstate1, y según su valor, se pone a 0 o a 1 el led conectado en el pin 12.

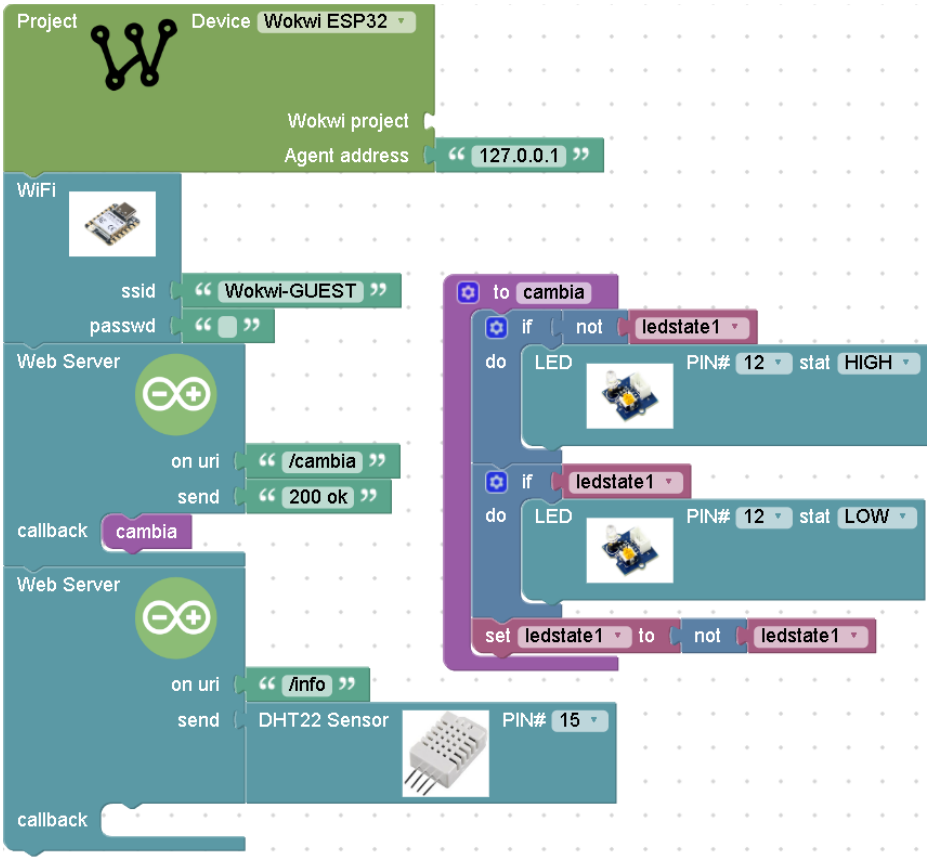


Figura 5.49. Bloques de los servicios.

Tarea 2: Simulación de los servicios.

En el proyecto Wokwi se incluyen los elementos de la Figura 5.50, un sensor de temperatura DHT22 conectado en el pin 15 y un led conectado al pin 12. Además, en el código hay que incluir las siguientes bibliotecas: **ArduinoJson** y **DHT sensor library for ESPx**. El proyecto completo está disponible en la dirección: https://wokwi.com/projects/371819597067016193.

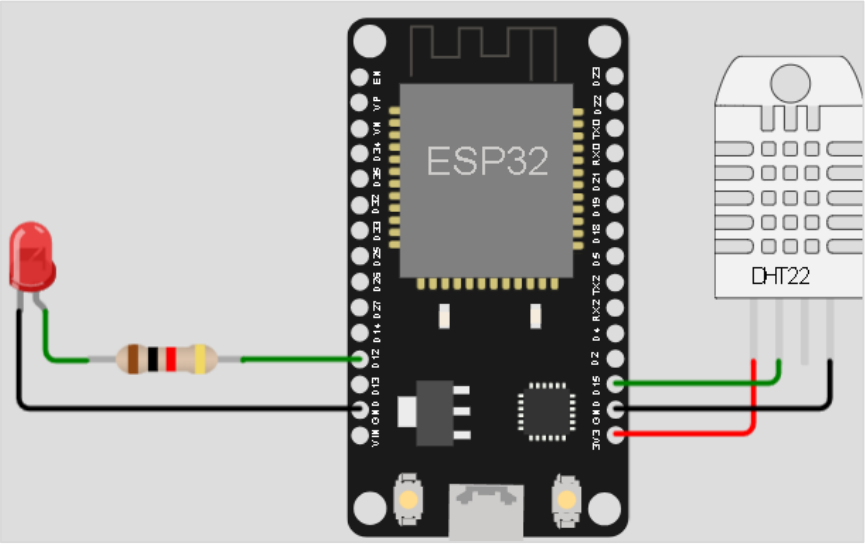


Figura 5.50. Conexión de un sensor DTH22 al microcontrolador ESP32.

Para poder acceder a los servicios desde el exterior, hay que utilizar la pasarela privada de Wokwi. En primer lugar, se descarga e instala en el ordenador la aplicación Wokwi Iot Gateway. A continuación, se ejecuta mostrando una pantalla como la de la Figura 5.51. Después de ejecutarlo, en el editor del proyecto Wokwi se presiona la tecla F1 y se selecciona 'Enable Private Wokwi IoT Gateway' como aparece en la Figura 5.52. Pregunta si desea habilitar la puerta de enlace y se responde: 'Aceptar' para habilitar la puerta de enlace privada o 'Cancelar' para deshabilitarla y volver a la puerta de enlace pública. A continuación, se ejecuta el proyecto y se conecta.

Figura 5.51. Wokwi Iot Gateway.

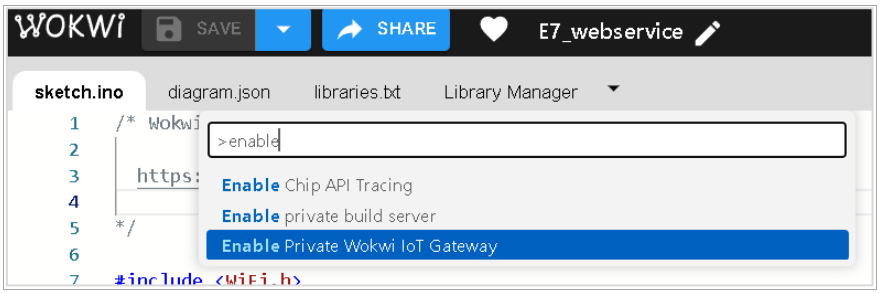


Figura 5.52. Habilita la pasarela privada.

Tarea 3: Acceso a los servicios.

Para acceder a los valores de temperatura y humedad del sensor, se llama al servicio /info. Se utiliza la aplicación curl.exe para generar un mensaje HTTP GET:

curl http://localhost:9080/info.

En la Figura 5.53, se puede observar el tráfico generado en la interfaz WiFi cómo existe un mensaje de petición HTTP GET invocando al servicio /info y cómo responde con los datos en formato JSON. La respuesta del servicio es: {"temp":24,"humd":40}.

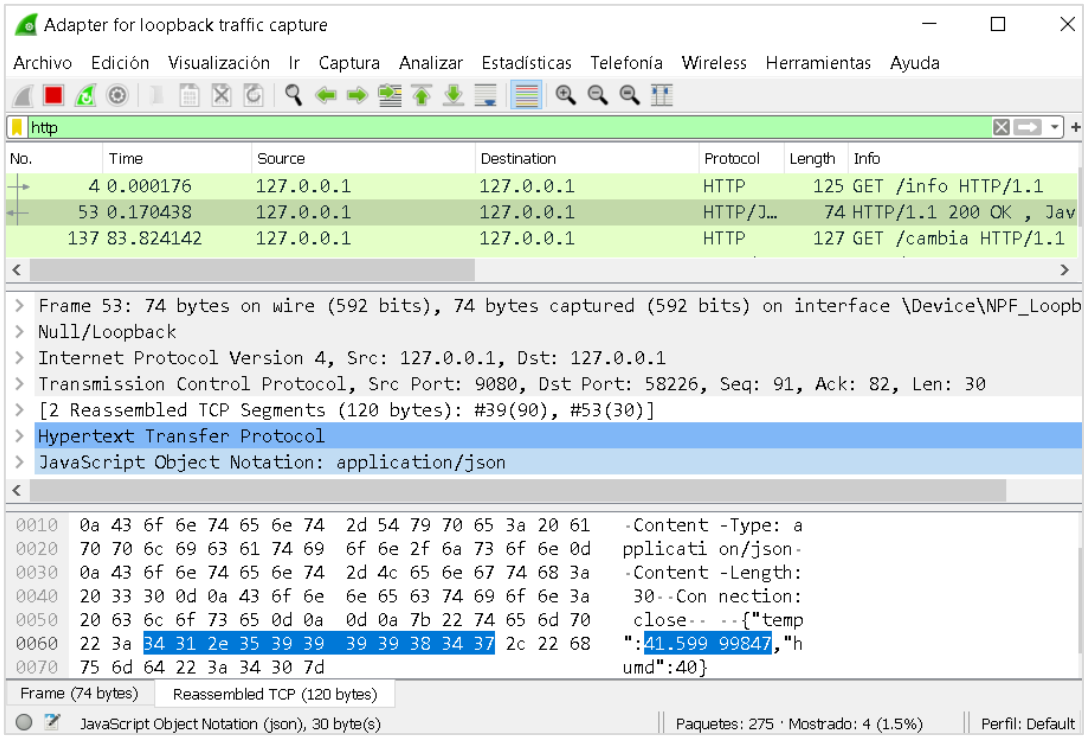


Figura 5.53. Mensajes de petición/respuesta generados al llamar al servicio /info.

Para cambiar el estado del led conectado, se llama al servicio con el siguiente comando: curl http://localhost:9080/cambia. Se observa como el led pasa de apagado a encendido.

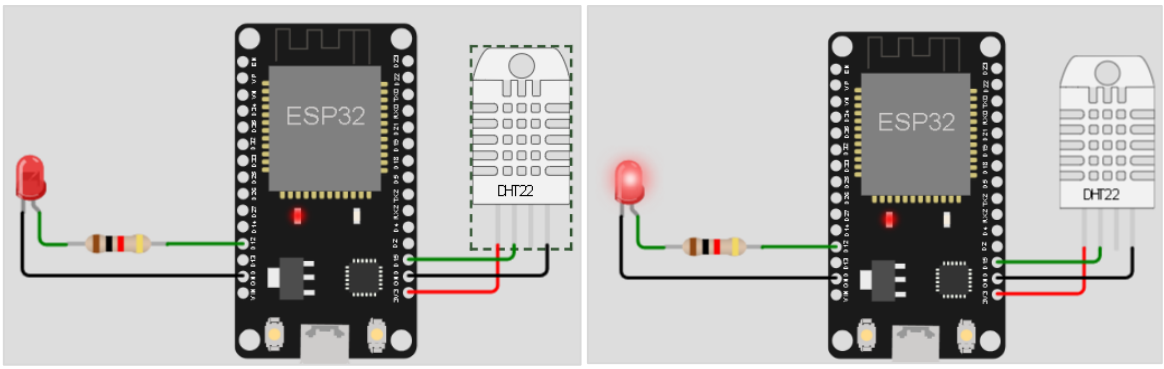


Figura 5.54. Cambio de estado del led.

5.9 Máquina virtual de un ordenador de placa reducida

Si no se dispone de una placa Raspberry Pi (como la mostrada en la Figura 5.55) con distintos sensores conectados a sus puertos para realizar actividades, una alternativa es crear una máquina virtual utilizando la aplicación Oracle VM VirtualBox. Con ella se simula un ordenador de placa reducida de bajo coste con un sistema operativo basado en Debian (llamado Raspbian), un procesador, una memoria RAM, puertos USB y conexión a red.

En esta actividad se persiguen tres objetivos:

- Mostrar cómo se puede utilizar una máquina virtual de una Raspberry Pi para realizar prácticas, útil para escenarios donde se disponga de una placa física.
- Realizar una actividad de adquisición y envío de datos utilizando MQTT, para su posterior recepción y almacenamiento.
- Visualización de los datos almacenados.

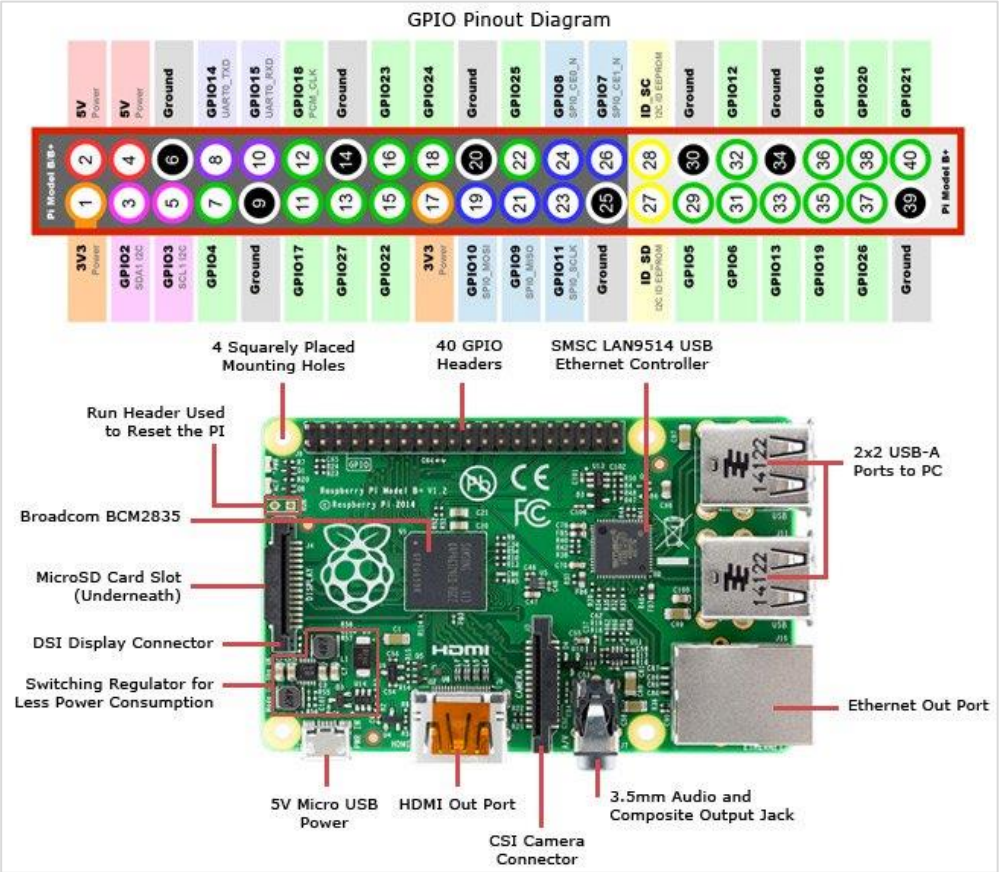


Figura 5.55. Raspberry Pi 3.

Fuente:https://www.jameco.com/Jameco/workshop/CircuitNotes/raspberry-pi-circuit-note.html

TAREA 1. Creación de una máquina virtual de una Raspberry PI

Los pasos a realizar son:

1. Se descarga la imagen ‘Raspberry Pi Desktop’ de la página oficial de Raspberry.
2. Se añade una nueva máquina utilizando la interfaz gráfica de VirtualBox. Se indica un nombre, el tipo de sistema operativo y la versión. En nuestro caso escribimos Debian y versión de 32 bits, ya que es la versión que nos ofrece Raspberry en su página oficial.
3. A continuación, se asigna el tamaño de la memoria y número de procesadores. Para esta actividad, elegimos 4096 MB y 2 procesadores.
4. Se añade el disco duro virtual, indicando el tamaño que se reserva (por ejemplo, 25GB).
5. Dentro de la opción ‘Configuración’, se selecciona la opción ‘Almacenamiento’ y se

añade una nueva unidad con la imagen del sistema descargada.

6. Se inicia la máquina virtual y se procede a la instalación del sistema contenida en la imagen. Se van siguiendo los pasos indicados durante el proceso de instalación.

TAREA 2. Creación de un publisher y subscriber MQTT

La versión de Debian utilizada tiene instalado un intérprete de Python y el entorno de desarrollo de aplicaciones Python llamado 'Thonny Python IDE' (Figura 5.56). Con ellos se crearán las aplicaciones para la captura de datos de sensores y su envío a un bróker MQTT para su posterior recepción y almacenamiento. Además, trae instalado un emulador Sense HAT, que simula el hardware Sense HAT conectado a la Raspberry (Figura 5.57). Con él, se pueden leer desde los sensores de temperatura, presión, humedad y orientación (para recopilar datos de movimiento). Posee una matriz de LED y un Joystick con 5 botones. Los tres ejes que se usan para describir el movimiento son (Figura 5.58): lateral (Pitch), longitudinal, (Roll), vertical (Yaw).

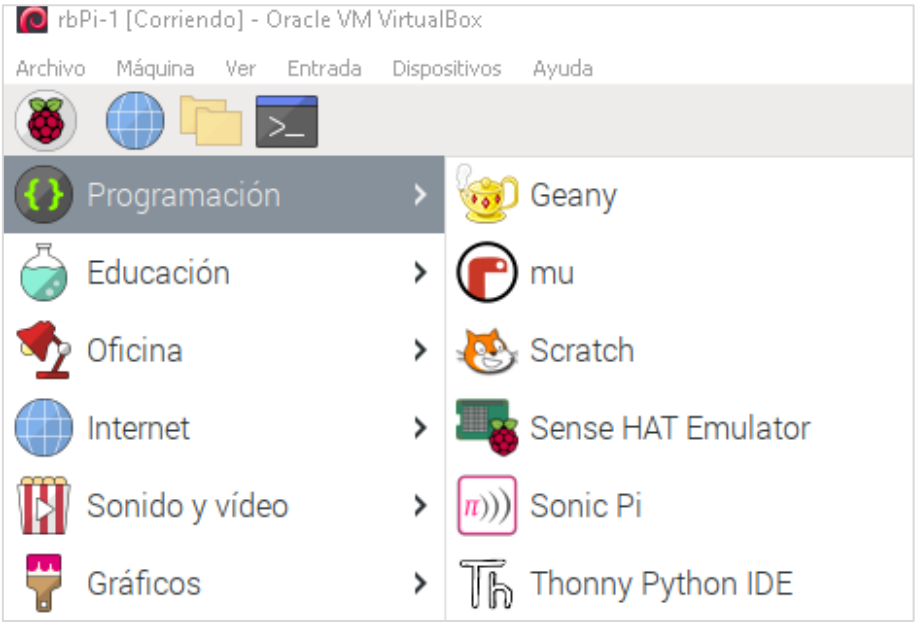


Figura 5.56. Aplicaciones del menú Programación.

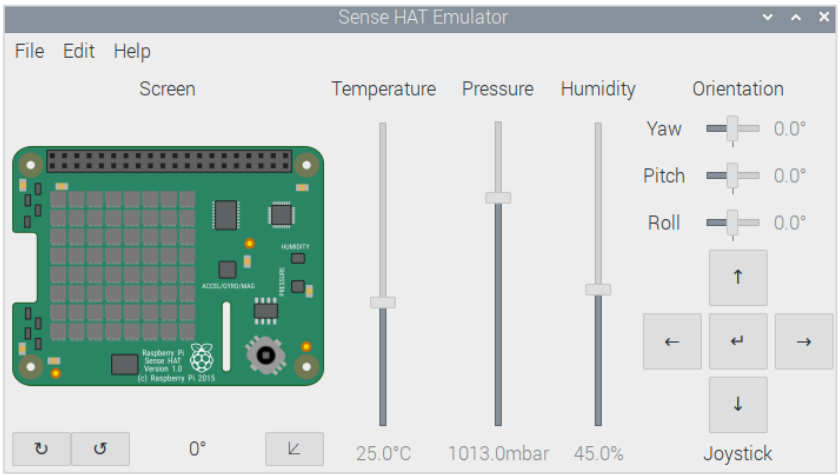


Figura 5.57. Emulador Sense HAT.

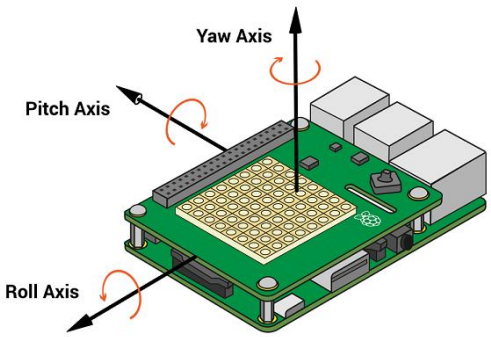


Figura 5.58. Ejes que describen el movimiento.

Además del interprete y aplicaciones preinstaladas, se necesita la instalación de Docker y desplegar los servicios mostrados en la Figura 5.59:

- Un gestor de bases de datos MySQL (S1).
- Un administrador de bases de datos phpMyAdmin (S2).
- Un servicio Grafana [44] el análisis y visualización de datos (S3).

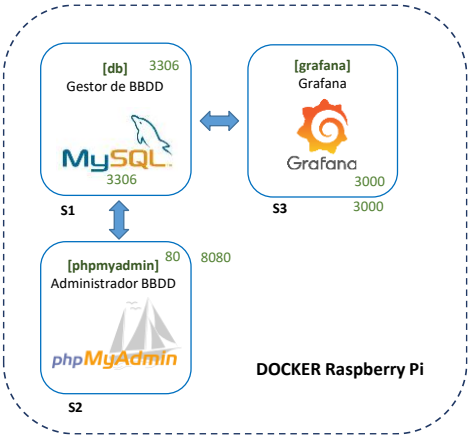


Figura 5.59. Servicios desplegados.

En Thonny se programa un *Publisher* y *Subscriber*, se ejecutan, comprobando cómo el suscriptor recibe los datos del sensor de temperatura y humedad existente en Sense HAT (Figura 5.60). Estos datos se almacenan en la base de datos creada (Figura 5.61).

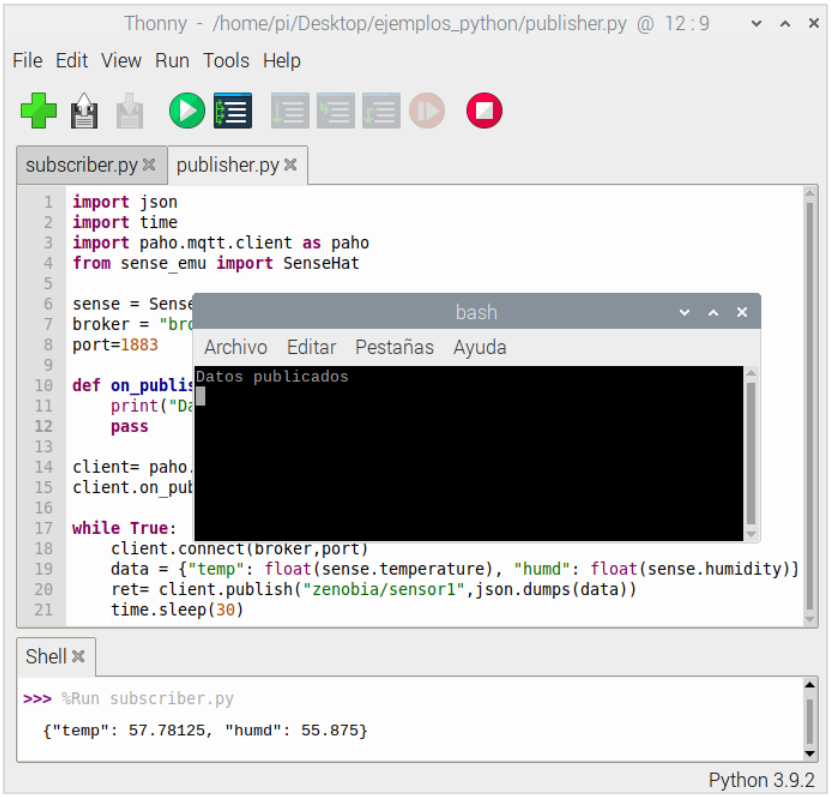


Figura 5.60. Aplicaciones Publisher y Subscriber.

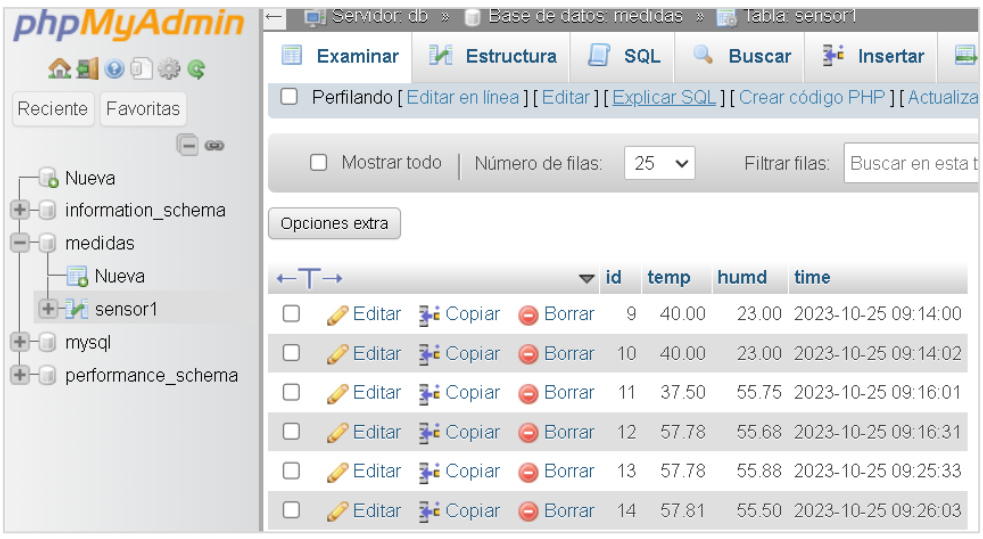


Figura 5.61. Contenido de la base de datos ‘medidas’.

TAREA 3. Visualización de los datos almacenados

Para visualizar la evolución temporal de un objeto se ha utilizado la aplicación Grafana. Es una herramienta de código abierto para el análisis y visualización de métricas. Se utiliza frecuentemente para visualizar de una forma elegante series de datos en el análisis de infraestructuras y aplicaciones. Permite el uso de gráficos totalmente interactivos y editables, así como un sistema de alertas fácilmente configurables desde la interfaz gráfica de usuario. La interfaz de usuario de Grafana es un navegador web (Figura 5.62).

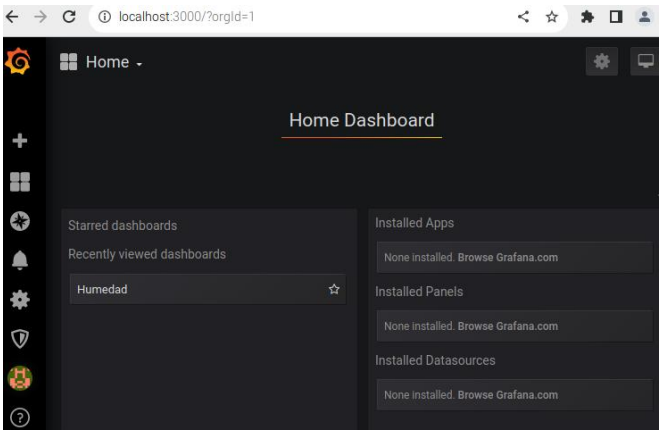


Figura 5.62. Interfaz de usuario de Grafana.

Para visualizar datos en una gráfica, previamente hay que definir los orígenes de datos (Data Sources), es decir, dónde se encuentran los datos guardados. Añadimos una nueva fuente (Figura 5.63). Cuando introduzcamos la información, pulsamos el botón 'Save and test' y si todo es correcto, nos aparece un mensaje 'Data source is working'.

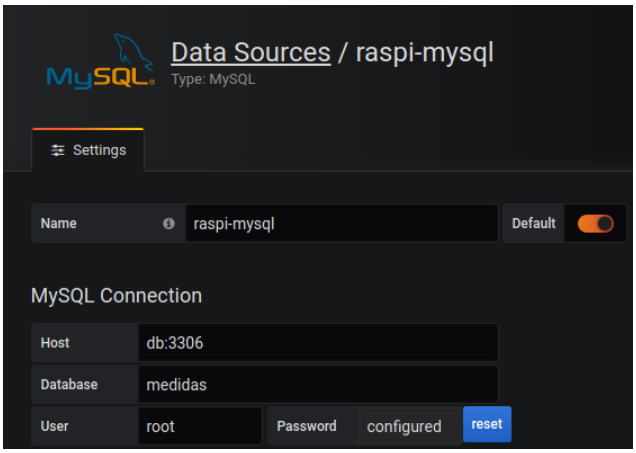


Figura 5.63. Menú Data Sources.

Para representar gráficamente los datos, hay que crear un ‘Dashboard’ (Figura 5.64). Un dashboard está formado por distintos Panels. Cada Panel tiene un origen de datos y una visualización (forma de presentar estos datos) asociada. En la figura se muestra un panel donde se visualiza el atributo humedad de la tabla ‘sensor1’.

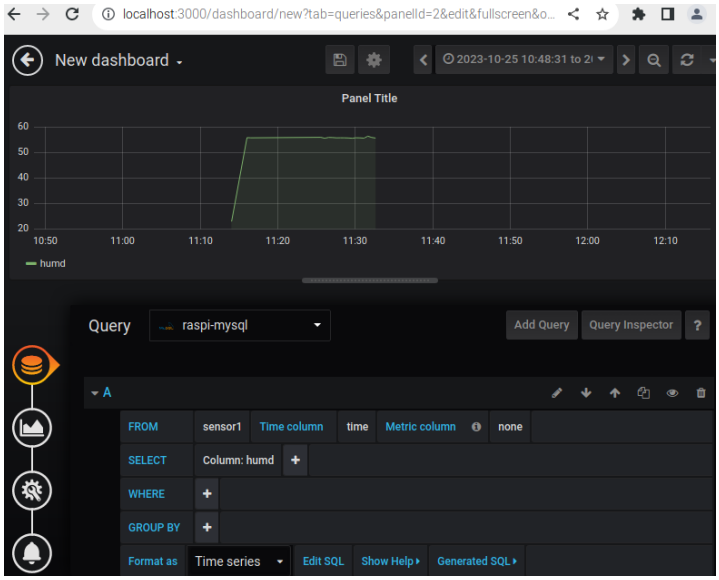


Figura 5.64. Nuevo dashboard.

6. CONCLUSIONES

En conclusión, los objetivos propuestos en este TFG, donde se pretendía mostrar las posibilidades de desarrollo que ofrecen los ordenadores con placa reducida y de bajo coste, se han logrado con éxito cómo se demuestra a través de las actividades expuestas en el capítulo 5. Trabajar con estos equipos y hacer uso de los entornos virtuales en el aprendizaje de conocimientos relacionados con las Telecomunicaciones ha ofrecido soluciones tecnológicas viables.

En el caso de la utilización de entornos virtuales, ha ofrecido la flexibilidad de adaptarse a la demanda o escenarios que se tenga en el aula de trabajo. Además, permite realizar en casa actividades propuestas en el aula.

El trabajar con hardware de placa reducida, nos ha llevado a mejorar el desarrollo de proyectos de IoT. Se ha explorado como almacenar, procesar y analizar datos por dispositivos conectados utilizando bases de datos y representación de datos en tiempo real. Utilizar herramientas de simulación como Wokwi, ha facilitado comprobar el correcto funcionamiento de los proyectos creados con programación de bloques.

Todos estos experimentos se han desarrollado con poco desembolso económico y se consideran entornos muy útiles para proyectos educativos.

7. LÍNEAS FUTURAS

Este TFG permite seguir desarrollando e impulsando el campo de la tecnología y la programación. Las posibilidades futuras son variadas.

Una de ellas es, añadir nuevas placas en el editor de bloques BlocklyDuino, como podrían ser ESP32, ESP32-S3 y familia Arduino MKR. Sería muy interesante realizar un análisis de qué dispositivos soportan OTA (Over The Air- sobre el aire), para realizar este tipo de programación y cargar código en la placa sin cables, el único requisito es que el dispositivo físico a programar y el ordenador desde el que se accede a BlocklyDuino, deben estar conectados a la misma red local. También permite de forma sencilla, añadir seguridad para evitar que otro usuario pueda cambiar el programa. Esto no solo eliminaría la necesidad de disponer de cables USB adicionales, sino que los puertos físicos quedarían libres y aumentaría la comodidad del usuario.

El desarrollo de nuevos bloques en BlocklyDuino, ha sido fundamental para adaptar la plataforma a los experimentos prácticos que se han elaborado en este TFG, que deja las puertas abiertas al desarrollo de nuevos bloques para otros protocolos IoT, como podría ser COAP, que es un protocolo que tiene mucha similitud con HTTP ya que se basa en cliente/servidor, pero sus mensajes son más pequeños y tiene restringido el número de dispositivos compatibles. Se propone este protocolo, ya que es uno de los protocolos más comunes a la hora de comunicar dispositivos IoT con internet junto a MQTT.

También se contempla el desarrollo de bloques que permitan definir nuevos tipos de variable y análisis de cadenas de texto, además de cadenas JSON. Esto sería la base para crear bloques de suscripción MQTT más complejos, que permitiesen procesar los mensajes recibidos y realizar acciones en función de esos mensajes.

Y para finalizar, una línea futura muy prometedora seria, si se quiere impulsar la plataforma hacia la vanguardia de la innovación tecnológica y la educación en la programación, incorporar nuevos bloques relacionados con la inteligencia artificial (IA), como por ejemplo el uso de redes neurales, controladores Fuzzy o clientes de ChatGPT – OpenAI.

8.ESTUDIO ECONÓMICO

El análisis económico se ha dividido en tres partes:

- **Gastos en material y equipamiento para un puesto del laboratorio**: Una Raspberry Pi donde se instala la pasarela Zigbee2MQTT, una placa de Arduino UNO WiFi, un sensor de temperatura y un ordenador.
- **Gastos en Software**: Una máquina virtual donde se tiene desplegados los servicios y suscripciones a las plataformas utilizadas.
- **Gastos en recursos humamos**: Tiempo que el ingeniero ha tardado en realizar este TFG.

MATERIAL Y EQUIPAMIENTO

Unidades	Descripción	Precio	Subtotal
1	Ordenador de sobremesa DELL 3020 i5–4430 3,00 GHz| 16GB RAM + TFT 24″	290€	290€
1	RASPBERRY PI 4 - 4GB	57.81€	57.81€
1	Arduino Uno WiFi Rev2	49.55€	49.55€
1	DHT22 AM2302 Sensor de Temperatura y Humedad y Cable compatible con Arduino.	5.45€	5.45€
		TOTAL:	402,81€

Tabla 8.1. Gastos de material y equipamiento.

GASTOS SOFTWARE

Unidades	Descripción	Precio	Subtotal
1	Suscripción Wokwi Club: Maker	8.47€/mes	8.47€/mes
1	Máquina Virtual: 4 GB RAM, 2 vCore y 60 GB SSD	14€/mes	14€/mes
1	Node-Red	-	-
1	Telegram	-	-
1	broker HiveMQ	-	-
1	Plataforma Ubidots STEM	-	-
		TOTAL:	22.47€

Tabla 8.2. Gastos software.

RECURSOS HUMANOS

Unidades	Descripción	Precio	Subtotal
1	300 horas	25€/hora	7500€
		TOTAL:	7500€

Tabla 8.3. Gastos en recursos Humanos.

Resultado del estudio económico:

SUBTOTAL	IMPUESTO I.V.A. (21%)	TOTAL
7925.28€	1664.30€	9589.58€

Tabla 8.4. Resultado del estudio económico.

9. ANEXOS

En el apartado 9.1 de este capítulo se presentan las tareas para desplegar los servicios en un nuevo equipo. En el apartado 9.2 se muestra como ejecutar el agente extendido. En el apartado 9.3 la instalación de la pasarela privada de Wokwi y finalmente se incluyen dos actividades adicionales que ilustran la utilidad de los servicios Arduino Web Editor y Arduino IoT Cloud.

9.1 Despliegue y configuración de los servicios

Para realizar el despliegue y configuración de los servicios en un nuevo equipo hay que realizar las siguientes tareas:

1. Se descarga y descomprime el archivo servicios.zip. Se comprueba que existen los elementos mostrados en la Figura 9.1.

Figura 9.1. Contenido del archivo servicios.zip.

2. Si no se va a desplegar el servicio zigbee2mqtt, se deben comentar las líneas relacionadas con él en el archivo docker-compose.yml.
3. Si se desean lanzar más de un contenedor node-red, se debe copiar el directorio /node-red y copiar el servicio en el archivo docker-compose.yml tantas veces como sea necesario. Por ejemplo, si se desea tener dos contenedores node-red, los directorios y contenido de docker-compose.yml pueden ser:

```
node-red:
  image: nodered/node-red
  container_name: node-red
  restart: always
  volumes:
    - ./node_red_data:/data
  networks:
    - telemetry
```

```
node-red1:
  image: nodered/node-red
  container_name: node-red1
  restart: always
  volumes
    - ./node_red_data1:/data
  networks:
    - telemetry
```

En el archivo nginx/nginx.conf, deben existir tantos path virtuales cómo contenedores node-red. Siguiendo con el ejemplo de dos contenedores node-red, la configuración sería:

```
location /alu0 {
    proxy_pass   http://node-red:1880/alu0;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
location /alu1 {
    proxy_pass   http://node-red1:1880/alu1;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
```

4. En los archivos de configuración de node-red (settings.js) hay que asignar a la variable 'httpAdminRoot' el valor de path virtual elegido. Por ejemplo, para el contenedor node-red se ha elegido /alu0, y se deberá incluir: httpAdminRoot: '/alu0'. En el caso de del contenedor node-red1, se ha elegido /alu1, y en su archivo de configuración debe aparecer: httpAdminRoot: '/alu1'.
5. Como última tarea, se despliegan los servicios con el comando:

```
docker-compose up --build -d.
```

9.2 Instalación del Argente Arduino extendido

Antes de ejecutar el agente hay que crear las claves y certificados para que pueda funcionar con conexiones seguras. Las tareas a realizar son:

1. Ejecutar el comando: arduino-create-agent.exe -generateCert.
2. Una vez generadas las claves y certificados, ejecutar la aplicación **arduino-create-agent.exe.**

9.3 Instalación de wokwigw

Para instalar la pasarela privada de Wokwi, hay que seguir los siguientes pasos:

1. Descargar la versión para Windows (existen versiones para macOS y Linux), disponible en la siguiente URL: https://github.com/wokwi/wokwigw/releases/tag/v1.1.2.
2. Extraer el archivo ZIP y ejecutar el archivo: wokwigw.exe.

3. Cuando en consola aparezca un logotipo, su versión y ‘Escuchando en el puerto TCP 9011’. Se ha completado la configuración.

9.4 Publisher MQTT con Arduino Web Editor

En esta actividad se realiza la aplicación Publisher, utilizando la herramienta Arduino Web Editor. Para acceder a este editor se utiliza el enlace de la plataforma Arduino Cloud: https://cloud.arduino.cc/ .

Tarea 1: Publicación de un mensaje con la plataforma Arduino Cloud.

Para crear el nuevo archivo se pulsa sobre ‘NEW SKETCH’ y en la parte superior derecha, se selecciona la placa de dispositivo y puerto donde está conectado (Arduino Uno WiFi Rev2 conectado al puerto Com3). Es necesario que la aplicación Arduino-Create-Agent este ejecutándose en el ordenador, para que la aplicación que se ejecuta en el navegador web se pueda comunicar con la placa conectada a un puerto físico.

Para publicar un mensaje se ha utilizado el código generado por aplicación de la actividad 5.3, como se muestra en la Figura 9.2. Si se ejecuta el código, en la pestaña Monitor, se observa la dirección IP asignada y que se ha conectado al bróker MQTT (Figura 9.3).

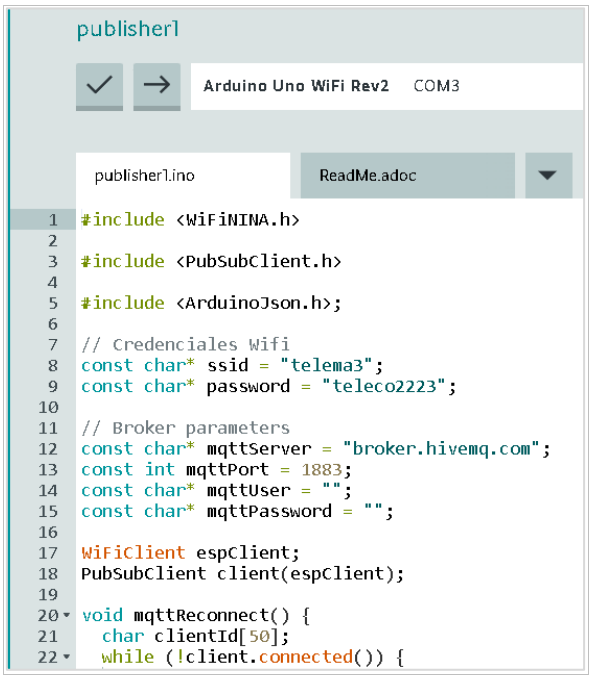


Figura 9.2. Código del programa ‘publisher’.

Figura 9.3. Actividad del Monitor.

Para probar su funcionamiento, se utiliza la aplicación Node-RED creada en la actividad 5.3. Los mensajes recibidos se muestran en la Figura 9.4, demostrando que funciona correctamente.

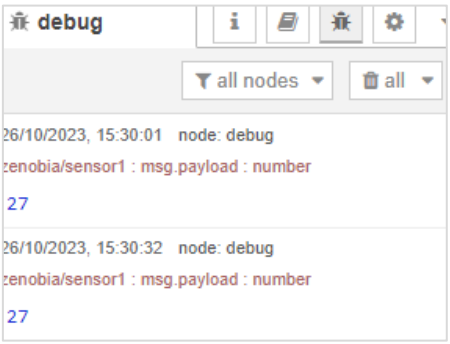


Figura 9.4. Mensajes recibidos subscriber.

9.5 ARDUINO CLOUD IoT

En esta actividad se realiza un ejemplo de uso de la plataforma Arduino IoT Cloud. Se accede usando el navegador a la plataforma Arduino Cloud: https://cloud.arduino.cc/.

Tarea 1: Visualizar los valores de una variable Cloud.

Cuando se accede a la plataforma, se elige la herramienta 'IoT Cloud'. En el apartado Things se crea un nuevo objeto llamado 'temperatura-TFG' (Figuras 9.65 y 9.6). El objeto debe tener un dispositivo físico asociado, por lo que se pulsa el botón Select Device y aparece un asistente para la configuración.

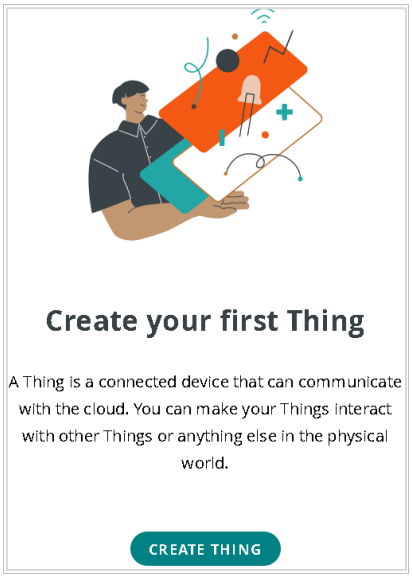


Figura 9.5. Creación de un nuevo objeto.

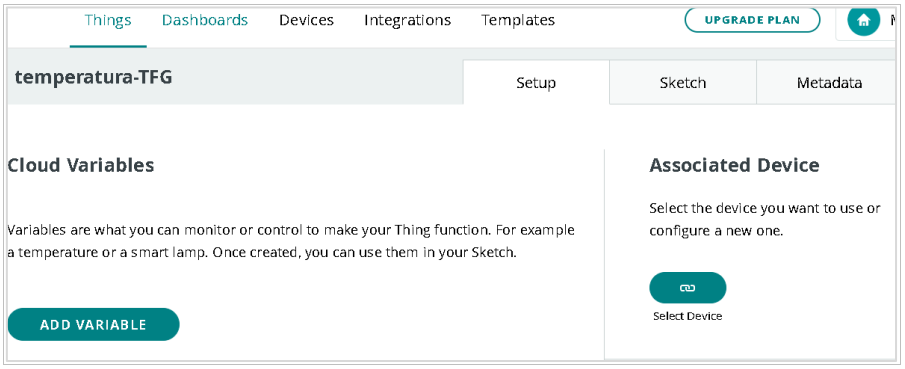


Figura 9.6. Objeto temperatura-TFG.

Como muestra la Figura 9.7, en sucesivas pantallas se elige un dispositivo y se le asigna un nombre: ‘DeviceTFG’.

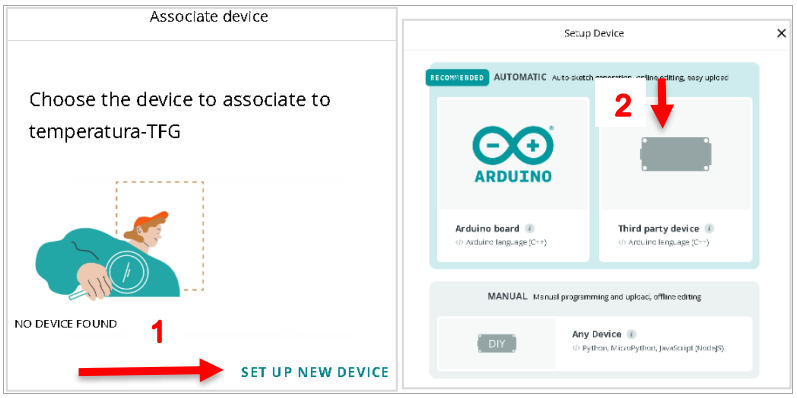


Figura 9.7. Configuración del dispositivo asociado.

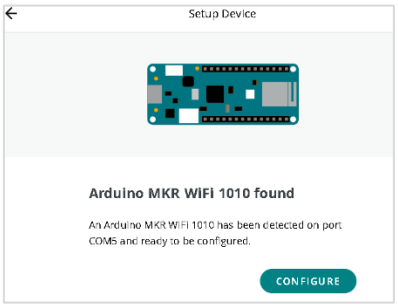


Figura 9.8. Elección del dispositivo asociado.

Si pulsamos sobre la pestaña Device en el menú superior, se puede ver el estado del dispositivo asociado (Figura 9.9), que es *offline* (sin conexión).

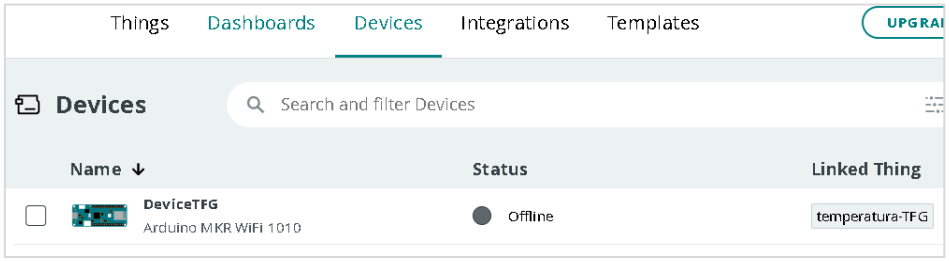
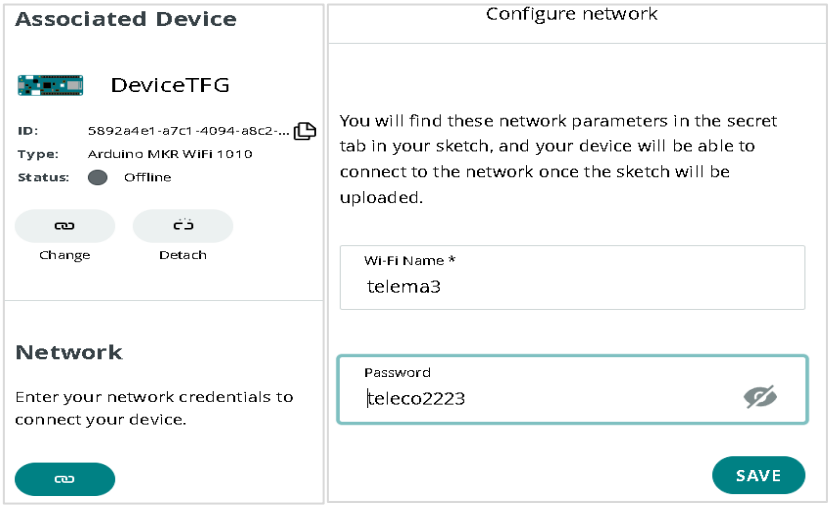


Figura 9.9. Estado del dispositivo DeviceTFG.

Se pulsa sobre el botón temperatura-TFG, y se muestra la pantalla de Figura 9.10 A. Se configura el punto de acceso (configure network), rellenando los campos que aparecen el formulario de la Figura 9.10 B (nombre de la WiFi y contraseña).



A B

Figura 9.10. Credenciales para la conexión del dispositivo.

El siguiente paso, es crear una variable Cloud (Figura 9.11). que sea accesible por todos los objetos de Cloud. En el menú de variables, se añade una nueva variable (ADD VARIABLE). La nueva variable será de tipo *float*, solo de lectura y se actualizará periódicamente cada 5 segundos (Figura 9.12).

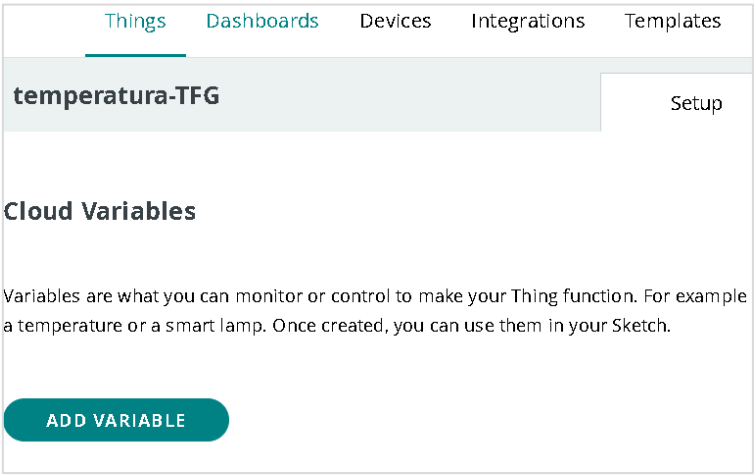


Figura 9.11. Añadiendo una variable.

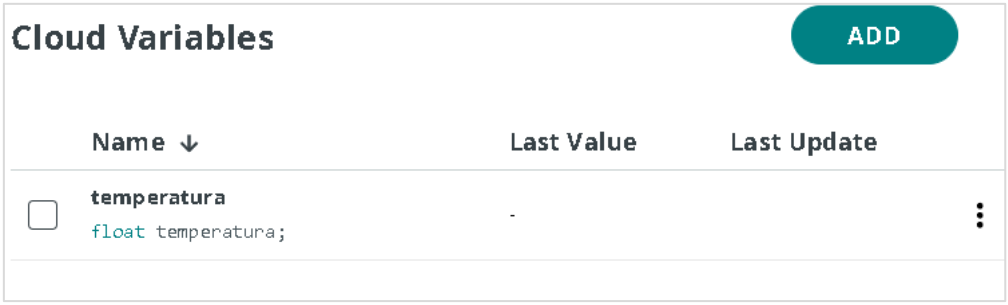


Figura 9.12. Variable Cloud 'temperatura'.

Si se selecciona la pestaña Sketch, situada en la parte superior derecha, aparece el código generado para el dispositivo asociado, como muestra la Figura 9.13. En él se ha añadido un código donde se genera un valor de temperatura aleatorio cada 5 segundos.

Figura 9.13. Código del dispositivo asociado.

Si se comprueba el estado del dispositivo, se observa que ya aparece *online* (conectado).

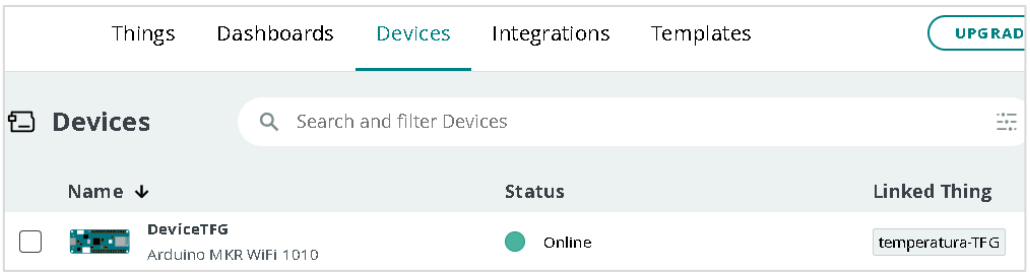


Figura 9.14. Estado del dispositivo DeviceTFG: online.

Si se quiere visualizar el valor de la variable Cloud 'temperatura', se pulsa en la pestaña Dashboards, situada en el menú superior, y a continuación, el botón BUILD DASHBOARD.

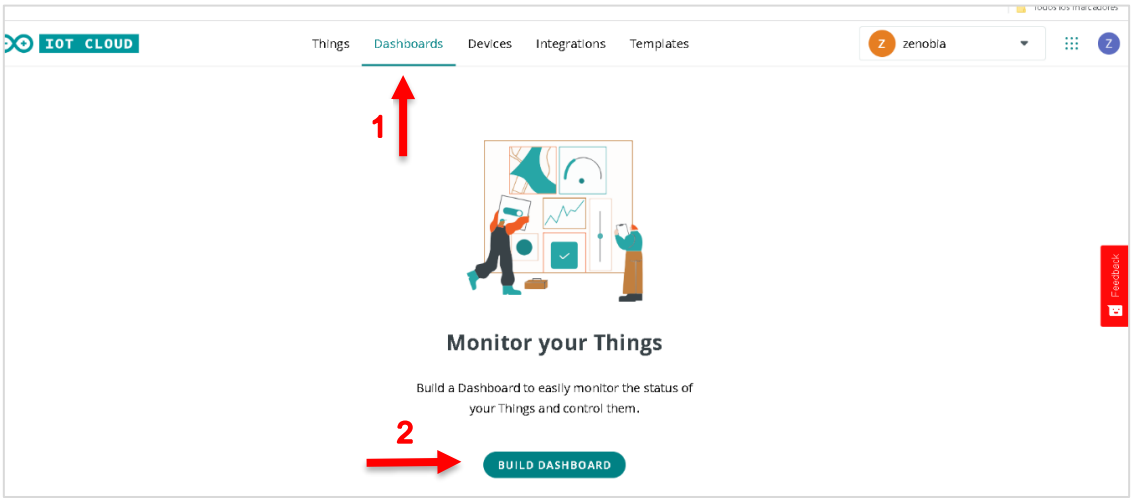
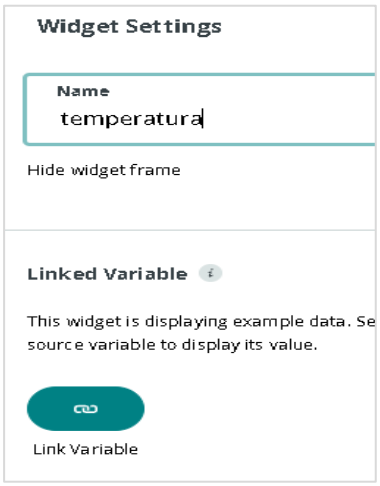


Figura 9.15. Nuevo dashboard.



Se proporciona un nombre: 'temperatura' y se enlaza con una variable, pulsando el botón Link Variable. Además, hay que proporcionar el tipo de visualización que se desea. Se ha elegido como tipo de visualización la opción Chart, el resultado es el que aparece en la siguiente Figura 9.17.

Figura 9.16. Nombre del dashboard.

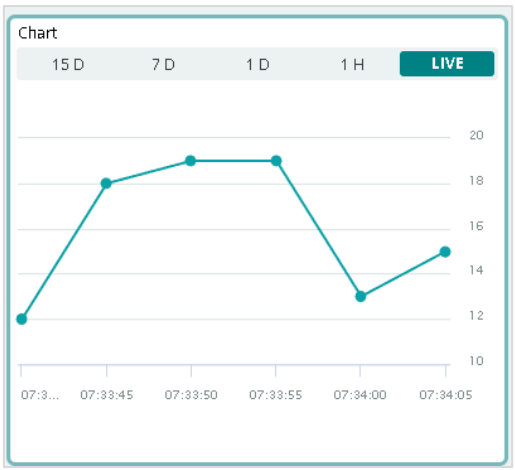


Figura 9.17. Visualización de los valores de la variable temperatura.

Tarea 2: Arduino Cloud IoT y Node-RED

Desde Node-RED se puede acceder al valor de una variable Cloud. Para ello se accede a la pestaña Integrations y aparece una pantalla donde se crea una API KEY:

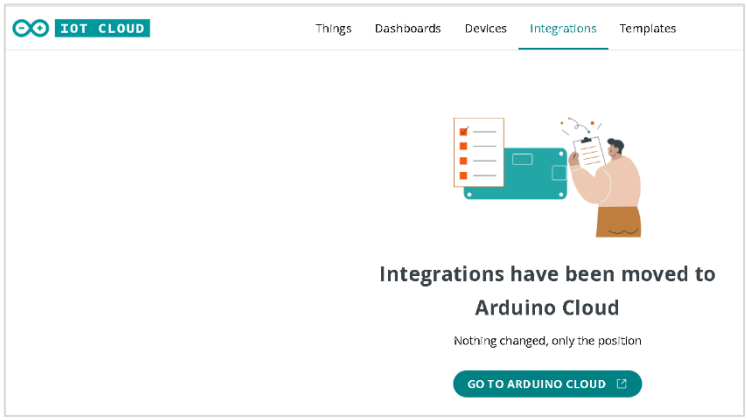


Figura 9.18. Elección de Integrations.

A la hora de crearla, se le proporciona un nombre y se generan un identificador y clave secreta.

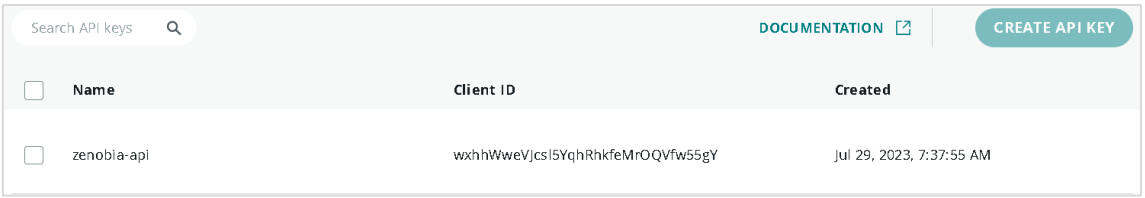


Figura 9.19. API Key creada.

Para crear la aplicación Node-RED y se instala la biblioteca ‘node-red-contrib-arduino-iot-cloud’. Se crea la aplicación mostrada en la Figura 9.20, formada por un nodo ‘property’, encargado de leer el valor de la variable Cloud y un nodo ‘debug’, para mostrar dicho valor.

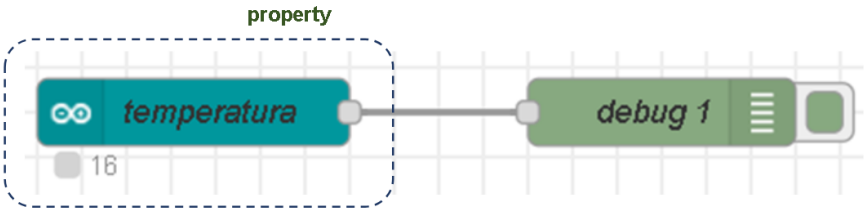


Figura 9.20. Aplicación para leer los valores de la variable Cloud ‘temperatura’.

Al realizar la configuración del nodo ‘property’ se está configurando un nodo ‘arduino-coonection’ donde se incluyen el identificador y clave secreta generados al crear el API KEY. Una vez finalizada, seleccionamos el objeto (Thing: temperatura-TFG) y la variable Cloud (Property: temperatura), tal y como se muestra en la siguiente figura:

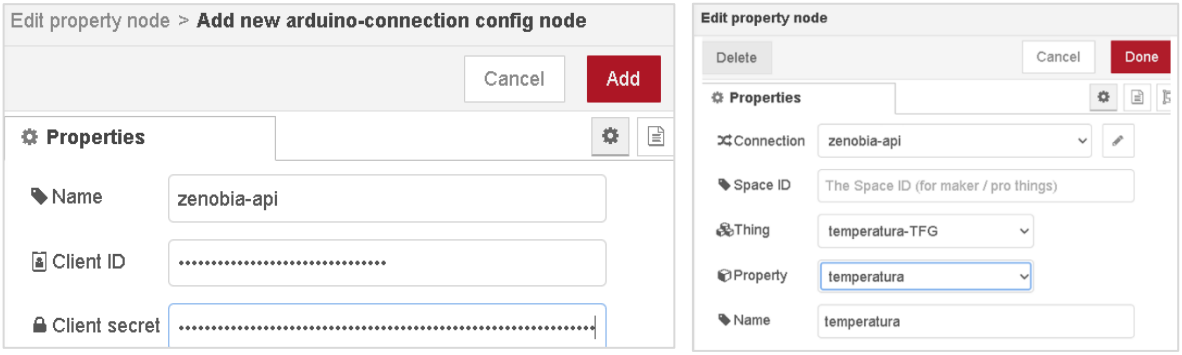


Figura 9.21. Configuración del nodo ‘property’.

Una vez desplegada la aplicación Node-RED aparecen los valores de la variable Cloud ‘temperatura’ en la ventana de depuración:

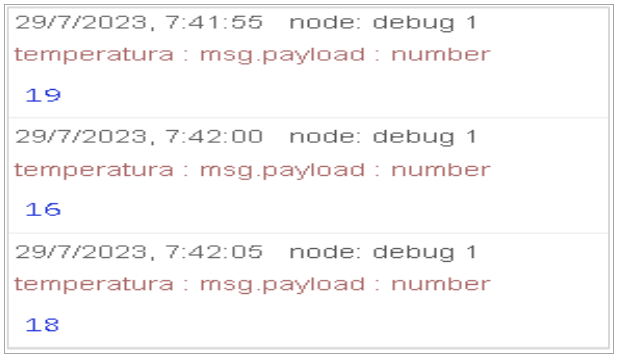


Figura 9.22. Valores de la variable Cloud recibidos.

10. BIBLIOGRAFÍA

[1] Informe económico sectorial de las telecomunicaciones y el audiovisual 2022. CNMC 2023. Disponible en la URL: https://www.cnmc.es/sites/default/files/4807231.pdf

[2] Montse Guitert, Teresa Romeu y María Pérez-Mateo. Competencias TIC y trabajo en equipo en entornos virtuales. Revista de Universidad y Sociedad del Conocimiento, UOC, 2007. Disponible en la URL: https://rusc.uoc.edu/rusc/es/index.php/rusc/article/download/v4n1-guitart-romeu-perez-mateo/289-1206-2-PB.pdf

[3] Documentación del simulador GNS3. Documento en línea. Disponible en la URL: https://docs.gns3.com/docs/

[4] Página oficial del curso **Cisco Packet Tracer de Cisco**. Disponible en la URL: https://www.netacad.com/e/courses/packet-tracer

[5] Laboratorios virtuales (sandboxes) de Cisco para seguridad. Recurso en línea disponible en la dirección la URL: https://developer.cisco.com/docs/sandbox/#!security/overview

[6] Laboratorios virtuales (sandboxes) de Cisco para VoIP. Recurso en línea. Disponible en la URL: https://developer.cisco.com/docs/ios-xe-voip/#!getting-started

[7] Página oficial del proyecto “Mininet”. Disponible en la URL: http://mininet.org/

[8] Página oficial del proyecto “Open Source MANO” de la ETSI. Instalación. Disponible en la URL: https://osm.etsi.org/docs/user-guide/latest/01-quickstart.html#installing-osm

[9] Android Studio. Página para desarrolladores. Disponible en la URL: https://developer.android.com/studio

[10] React Native. Página para desarrolladores. Disponible en la URL: https://reactnative.dev/

[11] Página oficial del simulador online Wokwi. Disponible en la URL: https://wokwi.com/

[12] Oracle Virtualbox. Recurso en línea, disponible en la URL: https://www.virtualbox.org/

[13] Documentación Arduino. Disponible en la URL: https://docs.arduino.cc/

[14] Página oficial de Arduino Cloud. Disponible en la URL: https://cloud.arduino.cc/how-it-works/

[15] Tutorial comenzando con Editor web Arduino. Disponible en la URL: https://docs.arduino.cc/arduino-cloud/getting-started/getting-started-web-editor

[16] Página del proyecto Arduino Create Agent. Recurso en línea. Disponible en la URL: https://github.com/arduino/arduino-create-agent

[17] Getting Started with the Arduino IoT Cloud. Recurso en línea. Disponible en la URL: https://docs.arduino.cc/arduino-cloud/getting-started/iot-cloud-getting-started

[18] FreeBook. Documentación en línea de ArduinoBlocks. Disponible en la URL: http://www.arduinoblocks.com/web/site/doc

[19] Learning Arduino through BlocklyDuino. Wiki del Proyecto BlocklyDuino. Recurso en línea. Disponible en la URL: https://github.com/BlocklyDuino/BlocklyDuino/wiki

[20] Página del proyecto Blockly. Recurso en línea. Disponible en la URL: https://github.com/google/blockly

[21] Node-RED. Documentación, guía de usuario. Recurso disponible en la URL: https://nodered.org/docs/user-guide/

[22] Docker Docs. Docker Docs Get Started. Recurso en línea. Disponible en la URL: https://docs.docker.com/get-started/

[23] ¿Qué es Kubernetes? Documentación oficial de Kubernetes. Recurso en línea disponible en la URL: https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/

[24] Bringing MySQL to the web. Documentación oficial de phpMyAdmin en línea. Disponible en la URL: https://www.phpmyadmin.net/docs/

[25] The Free Public MQTT Broker. Documentación y enlace a dashboard disponibles en la URL: https://www.hivemq.com/public-mqtt-broker/

[26] NGINX Reverse Proxy. Documentación en línea. Disponible en la URL: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/

[27] Zigbee2MQTT Getting started. Página oficial del proyecto. Disponible en la URL: https://www.zigbee2mqtt.io/guide/getting-started/

[28] Wokwi The Private Gateway. Uso e instalación de la pasarela a la red privada IoT Wokwi. Recurso en línea. Disponible en la URL: https://docs.wokwi.com/guides/esp32-wifi#the-private-gateway

[29] Docker Compose. Docker compose overview. Recurso en línea. Disponible en la URL: https://docs.docker.com/compose/

[30] The Python Tutorial. Documentación oficial Python. Recurso en línea. Disponible en la URL: https://docs.python.org/3/tutorial/index.html

[31]. Python Flask User's Guide. Documentación de la página oficial Python Flask. Recurso en línea. Disponible en la URL: https://flask.palletsprojects.com/en/2.3.x/

[32] JavaScript. Resources for Developers by Developers. Mozilla Developer Network Web Docs. Recurso en línea. Disponible en la URL: https://developer.mozilla.org/es/docs/Web/JavaScript

[33] Coppola, María. Qué es JavaScript, para qué sirve y cómo funciona. Recurso en línea. Disponible en la URL: https://blog.hubspot.es/website/que-es-javascript

[34] Node.js. Recurso en línea. Disponible en la URL: https://nodejs.org/es/about

[35] Guía de Referencia del lenguaje de programación Arduino. Web oficial de Arduino. Recurso en línea. Disponible en la URL: https://www.arduino.cc/reference/es

[36] MicroPython documentation. Página web del proyecto oficial. Disponible en la URL: https://micropython.org/

[37] Álvarez, Miguel Ángel. Qué te ofrece PhpMyAdmin y cómo podrás instalarlo. Recurso en línea. Disponible en la URL: https://desarrolloweb.com/articulos/844.php

[38] WebSockets. Resources for Developers by Developers. Mozilla Developer Network Web Docs. Recurso en línea. Disponible en la URL: https://developer.mozilla.org/es/docs/Web/API/WebSockets_API

[39] Socket.IO Introduction. What Socket.IO is. Recurso en línea. Disponible en la URL: https://socket.io/docs/v4/

[40] Tutorial: Get started with Go. Documentación de la página oficial del lenguaje Go. Recurso en línea. Disponible en la URL: https://go.dev/doc/tutorial/getting-started

[41] Zigbee Specification. CSA, 2015. Recurso en línea. Disponible en la URL: https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf

[42] Blockly Developer Tools. Disponible en la URL: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html

[43] Toptal JavaScript Minifier. Disponible en la URL: https://www.toptal.com/developers/javascript-minifier.

[44] Documentation: Technical documentation for Grafana Labs products and services. Recurso en línea. Disponible en la URL: https://grafana.com/docs/