Escuela Politécnica Superior de Linares

Grado en Telemática

Universidad de Jaén

Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

ENTORNO DE DESARROLLO PARA DESPLIEGUE DE APLICACIONES EN PLATAFORMAS ARDUINO

Alumno: José Antonio Barrios García

Tutor I: José Manuel Pérez Lorenzo
Depto.: Ingeniería de Telecomunicación

Tutor II: José Enrique Muñoz Expósito
Depto.: Ingeniería de Telecomunicación

Noviembre, 2024

Resumen

Este Trabajo Fin de Grado (TFG) se centra en demostrar que es posible crear código para Arduino mediante el uso de diferentes bloques prediseñados, de modo que cualquier persona pueda desarrollar sus propios programas sin necesidad de tener conocimientos avanzados de programación.

Para la virtualización de los servicios ofrecidos se han utilizado contenedores Docker. Creando tantos contenedores como servicios ofrecidos. Los servicios son los siguientes: un editor de programación visual basado en bloques (BlocklyDuino), un servidor utilizado como proxy inverso (Nginx), un servicio web para la visualización de datos (React), dos servicios para la extracción de datos (Flask/Python) y un bróker MQTT (HiveMQ).

Uno de los servicios consiste en una extensión de un proyecto desarrollado por Google llamado Blockyduino, que emplea el concepto de programación basado en bloques para crear aplicaciones ejecutables en placas Arduino. Se han añadido funcionalidades como nuevos bloques basados en Cloud, un monitor que permita ver la información enviada al puerto serie, un cliente mqtt embebido y un generador de plantillas vinculado a ciertos bloques.

La carga de datos en Arduino se realiza a través del entorno de desarrollo PlataformIO.

En el punto 4 (Diseño e implementación), se han descrito varias actividades prácticas, que ilustran el funcionamiento de estas herramientas y servicios.

Abstract

This Final Degree Project (TFG) focuses on demonstrating that it is possible to create Arduino code using various pre-designed blocks, enabling anyone to develop their own programs without needing advanced programming knowledge.

For the virtualization of the offered services, Docker containers were used, creating as many containers as services provided. The services include the following: a visual programming editor based on blocks (BlocklyDuino), a server used as a reverse proxy (Nginx), a web service for data visualization (React), two services for data extraction (Flask/Python), and an MQTT broker (HiveMQ).

One of the services is an extension of a project developed by Google called BlocklyDuino, which uses block-based programming to create executable applications for Arduino boards. New functionalities have been added, such as cloud-based blocks, a monitor for viewing information sent to the serial port, an embedded MQTT client, and a template generator linked to certain blocks.

Data upload to Arduino is done through the PlatformIO development environment.

In Section 4 (Design and Implementation), several practical activities are described that illustrate the operation of these tools and services.

Índice

Índice de figuras

Índice de tablas

1. Introducción

En este Trabajo de Fin de Grado (TFG) se plantea la exploración de diversas soluciones a problemas frecuentes en el ámbito de las telecomunicaciones, utilizando Arduino como herramienta principal. Arduino es uno de los sistemas más utilizados en la actualidad para la gestión y control de datos mediante sensores. Su bajo costo lo hace accesible a una amplia variedad de usuarios. Sin embargo, su potencial se ve limitado por la falta de conocimiento en programación, además que mucha gente tampoco tiene el tiempo necesario para poder aprender a programarlo correctamente, todo esto dificulta a muchos usuarios aprovechar plenamente sus capacidades. Este proyecto tiene como objetivo facilitar el uso optimizado de microcontroladores Arduino para usuarios con diferentes niveles de experiencia en programación.

Para lograrlo, se ha propuesto utilizar una plataforma web con una interfaz basada en bloques modulares específicos para Arduino Esta interfaz permite a los usuarios conectar diferentes bloques de acuerdo con sus necesidades, lo que posibilita la creación de programas complejos sin necesidad de conocimientos avanzados en programación. Con esta herramienta, la curva de aprendizaje se vuelve mucho más accesible, y los usuarios pueden alcanzar sus objetivos de manera intuitiva y eficaz.

Además, uno de los usos más comunes de Arduino es la transmisión de datos hacia bases de datos, tablas o programas externos. Para realizar esta transmisión de manera eficiente, es fundamental seleccionar un protocolo que sea ágil y consuma la menor cantidad de recursos posible. En este sentido, el protocolo MQTT se presenta como una solución ideal, ya que permite publicar y suscribirse a diferentes temas, facilitando que diferentes usuarios accedan simultáneamente a la misma información o a datos específicos según sus necesidades, para implementar esta funcionalidad necesitaremos un bróker MQTT el cual se encargará de recibir los mensajes que se publiquen en un determinado tema y distribuirlos a los usuarios subscritos a dicho tema, se creará un bloque que permita publicar datos en un determinado tema, para completarlo todo se creará un cliente embebido que permita obtener datos en tiempo real de un determinado tema, pudiendo también comprobar que se realiza correctamente tanto la publicación como la suscripción al tema.

También abordamos la necesidad creciente de las empresas de almacenar datos en sistemas no locales. La dependencia de un único equipo físico para el almacenamiento de

información puede representar un riesgo significativo, ya que, en caso de fallo, se podría perder la información si no se dispone de una copia de seguridad reciente. Para mitigar este riesgo, proponemos la integración de servicios en la nube, una solución ampliamente utilizada en la actualidad. A tal efecto, hemos diseñado módulos que permiten la conexión, inserción y consulta de datos en bases de datos en la nube, especialmente en plataformas como Google Cloud y Firebase. Además, añadimos un bloque genérico para la conexión con otros servicios de bases de datos SQL en la nube, lo que facilita el almacenamiento y gestión de datos en plataformas externas, reduce la dependencia de recursos locales y aumenta la seguridad y accesibilidad de la información.

Otro de los aspectos que abordamos en este trabajo es la visualización de los datos. Muchos usuarios pueden tener dificultades para interpretar grandes volúmenes de datos, especialmente los que se obtienen de múltiples sensores. Para abordar este problema, proponemos una interfaz interactiva que permite al usuario seleccionar los datos que desea visualizar. Los datos se presentan en una gráfica lineal con ejes configurables según las preferencias del usuario, lo cual facilita la interpretación de los resultados obtenidos.

2. Objetivos

El objetivo del actual TFG surge a partir de un TFG previo en el cual se creó un entorno de programación visual basado en bloques para Arduino [23]. En concreto, los objetivos consisten en añadir las siguientes funcionalidades:

1) Interacción con los siguientes elementos de la plataforma Google Cloud:
   1.1) Pub/Sub
   1.2) Cloud Function
   1.3) Cloud SQL
2) Interacción con la API de ChatGPT.
3) Creación de una interfaz web para monitorear datos en tiempo real.
4) Inserción, visualización de datos y conexión con bases de datos SQL y Firebase.
5) Creación de un Suscriptor MQTT embebido en una web.
6) Contenerización con tecnología Docker.

Cada bloque generará un código interpretable por Arduino, el usuario podrá cambiar ciertos parámetros del bloque con la configuración deseada, y la combinación de varios bloques crearan el código final que ejecutará nuestro Arduino.

3. Material y Métodos

En este punto vamos a realizar una descripción del trabajo realizado, profundizando en cada uno de los puntos presentando el planteamiento, las tecnologías aplicadas y la metodología necesaria para cumplir nuestro objetivo.

3.1 Arquitectura ideal

Para el correcto funcionamiento de nuestro ecosistema se necesita desplegar una serie de servicios que deben ser configurados correctamente para que puedan interaccionar entre ellos, en la **Figura 1** se representa el esquema planteado, el cual termina en un dispositivo Arduino ESP32 ya que es un microprocesador que contiene un módulo WIFI.
Los bloques serán:

- **Local**: Dispositivos que deben de estar configurados e instalados de forma local.
- **Nube**: Para este trabajo los módulos que vamos a crear se conectan a servicios que se encuentran en nube.
- **Contenedor**: Todos los módulos que son encapsulados en nuestro Docker de forma que se pueda desplegar en cualquier equipo.

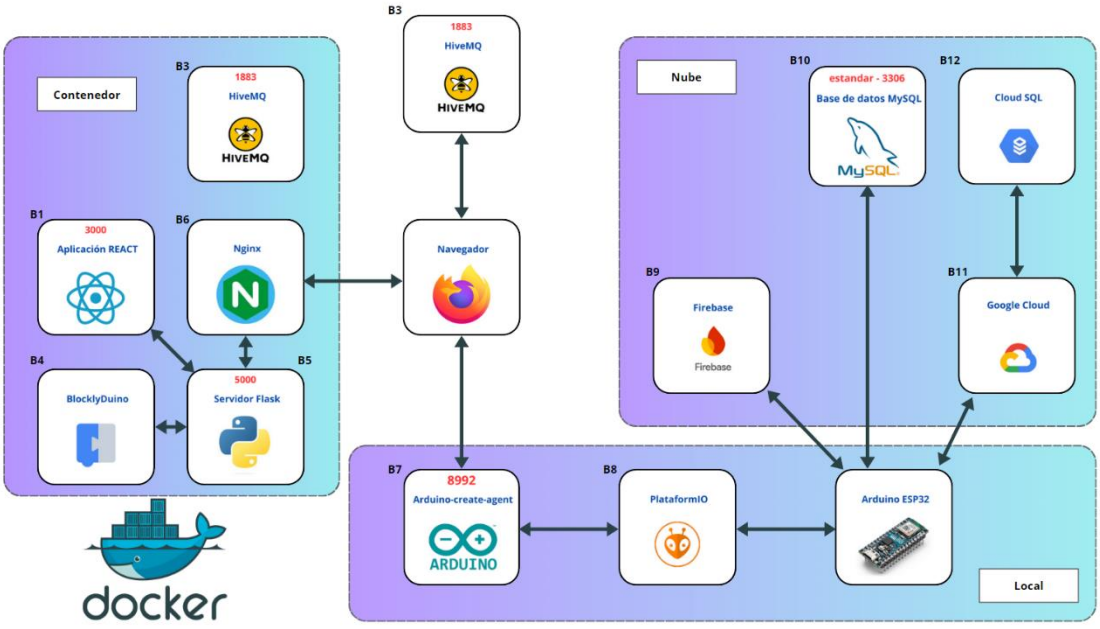


Figura 1. Esquema ideal del sistema.

En el siguiente esquema podemos diferencias los siguientes elementos:

-. **Servidor Flask** (B6) Es el framework para aplicaciones web en Python, realiza la comunicación con el proxy reservo, se comunica con el resto de los elementos a través de los distintos endpoints.

-. **Aplicación React** (B1) conectada directamente al servidor Flask, mediante los endpoint obtendrá la información necesaria para cada una de las tareas, como podría ser obtener las credenciales de acceso a una base de datos previamente especificada.

-. **HiveMQ** (B3) utilizado como bróker, desplegado en Docker, pero también puede encontrarse en otra red, puede usarse tanto de cliente como servidor, utilizando el protocolo mqtt.

-. **BlocklyDuino** (B4) Conectado al servidor Flask a través de otro de los endpoint y mostrado en el navegador, contiene las herramientas de bloques para poder configurar nuestro Arduino.

-. **Arduino-create-agent** (B7) Se comunica con el navegador y es utilizado para cargar el código generado por los bloques en nuestro dispositivo Arduino ESP32, además en el puerto 8992 contiene una interfaz gráfica por si queremos utilizar alguno de los comandos.

-. **PlataformIO** (B8) IDE que se utilizará para cargar el código en nuestro Arduino, funciona de manera similar al tradicional Arduino IDE

-. **Servicios SQL** (B9 y B10) realizará la comunicación con las bases de datos, tanto en FireBase como cualquier base de datos MySQL

-. **Google Cloud** (B11) Utilizará el sistema Pub/Sub y realizará la conexión a Google Cloud, además también podemos usar disparadores para una vez publicada la información en un topic cargar ese dato simultáneamente en una base de datos Google SQL.

3.1.1. Arquitectura Real

La arquitectura se ha modificado para utilizar un certificado SSL autofirmado. En este contexto, el uso de un certificado autofirmado presenta ciertas limitaciones para la comunicación segura entre los componentes del sistema. Por ejemplo, la aplicación cliente en React rechaza las conexiones con certificados autofirmados. Sin el uso de un certificado válido, React no puede establecer una conexión con el servidor Flask de manera segura.

Para solucionar estos problemas, se ha incorporado un segundo servidor que actúa como intermediario seguro. Este segundo servidor acepta el certificado SSL autofirmado y facilita la conexión con el servidor Flask. Al funcionar como intermediario, permite que el cliente en React se comunique de manera eficiente y segura con los servicios de Flask, evitando el rechazo de conexiones y habilitando la comunicación entre los componentes.

Por lo tanto, la nueva arquitectura para cumplir los requisitos se muestra a continuación.

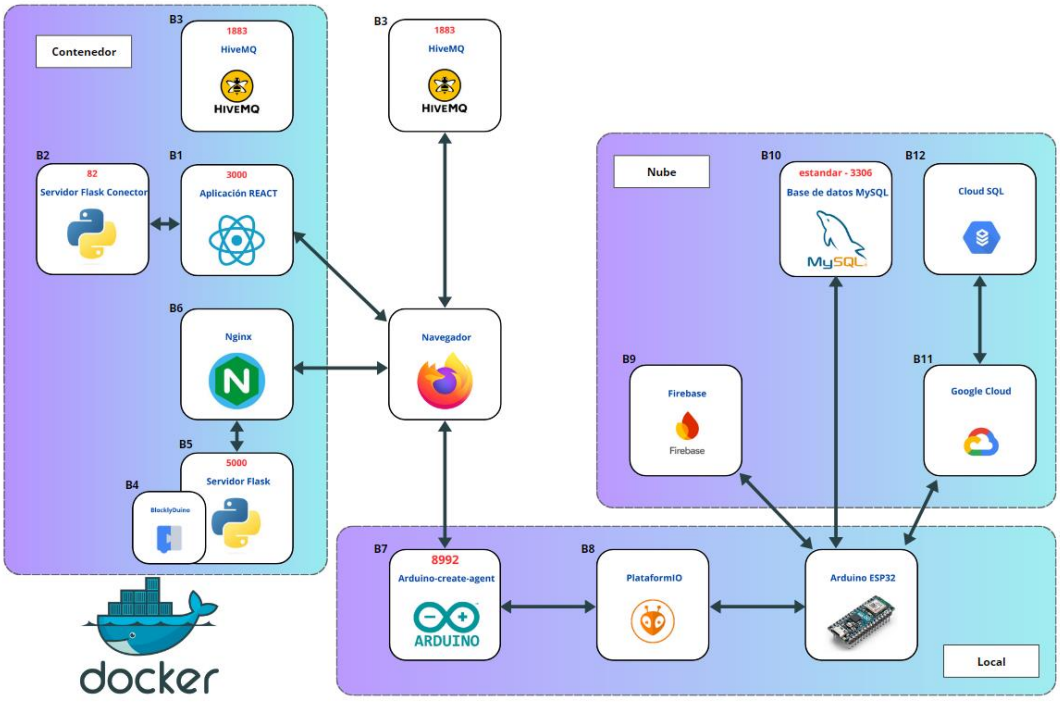


Figura 2. Esquema real del sistema

3.2. Herramientas y Tecnologías utilizadas

Como se ha explicado en el apartado anterior para este trabajo hemos utilizado diferentes tecnologías y herramientas las cuales vamos a detallar en los siguientes puntos.

3.2.1. Docker

Docker se puede definir como “una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.
La ejecución de Docker en AWS les ofrece a desarrolladores y administradores una manera muy confiable y económica de crear, enviar y ejecutar aplicaciones distribuidas en cualquier escala.”. [1]

Cada contenedor funciona de manera similar a una máquina virtual de forma que cada una de las aplicaciones se pueden ejecutar independientemente, pero se pueden conectar entre ellas, con la ventaja que no necesita tener una imagen completa del sistema operativo, cada contenedor utilizar una imagen, la cual contiene el sistema de archivos y el código necesario para poder ejecutar una determinada aplicación.

Algunas de las ventajas que nos presenta Docker son:

- **Portabilidad:** Ya que los contenedores se pueden desplegar en cualquier equipo y las dependencias se ejecutan dentro del contenedor sin necesidad de tener ninguna dependencia en el equipo local.

- **Múltiples versiones:** Cada contenedor puede ser lanzado con una versión especifica o podemos elegir la última versión del programa.

- **Escalable:** Es mucho más fácil escalar un ecosistema Docker como vamos a ver más adelante utilizando Docker compose.

Para poder lanzar un Docker tenemos que crear un archivo con una estructura similar a la siguiente:

```
#Imagen que queremos utilizar en esta caso python
FROM python:3.7

#Copiamos las dependencias que queremos utilizar en el contenedor
#Utiliza un formato COPY <Ruta local> <Ruta en el contenedor>
COPY requirements.txt /home/servidor/requiremets.txt

#Instalamos las dependencias dentro del contenedor
RUN pip install -r /home/servidor/requiremets.txt

#Establecemos el directorio de trabajo
WORKDIR /home/servidor/

#Copiamos los archivos dentro del contenedor en este caso todos los
#de la carpeta en los cuales se encuentra el fichero Docker
COPY . /home/servidor/.

#Comando que utilizamos al desplegar el contenedor
CMD python /home/servidor/server.py
```

Figura 3. Ejemplo estructura de un fichero Docker

3.2.1.1 Docker compose

“Docker Compose es una herramienta versátil que te permite definir y gestionar aplicaciones multi-contenedor de forma sencilla. Con Docker Compose, puedes describir la configuración de tu entorno de desarrollo en un archivo YAML, especificando los servicios, volúmenes y redes necesarios para tu aplicación. Luego, con un solo comando, puedes crear y ejecutar todos los contenedores definidos en tu archivo de configuración.” [2]

Con Docker Compose, es posible interconectar un grupo de contenedores. Además, solo necesitamos ejecutar Docker Compose una vez para lanzar todos los contenedores especificados. En el archivo de configuración de Docker Compose, el campo "ports" es fundamental. Este campo permite vincular un puerto interno del contenedor con un puerto local en nuestra máquina. Esto resulta muy útil, ya que nos permite desplegar varios contenedores que usen el mismo puerto interno. Por ejemplo, dos páginas web pueden

utilizar el puerto 80 en el contenedor y estar vinculadas a diferentes puertos locales. La estructura de un archivo Docker Compose podría ser la siguiente:

Figura 4. Ejemplo estructura fichero Docker Compose

En la primera línea de Docker Compose, usamos “services”. Después de “services”, listamos todos los contenedores que queremos desplegar. Con “build” indicamos la carpeta donde se encuentra el archivo Dockerfile.

El campo “container_name” define el nombre del contenedor dentro de la aplicación Docker. El campo “volumes” es importante, ya que permite compartir volúmenes de datos. Por ejemplo, se puede copiar un archivo de configuración como nginx.conf. La estructura es: <ruta local>:<ruta en el contenedor>.

Finalmente, en “ports” especificamos los puertos que usará el contenedor. La estructura es <puerto en la máquina local>:<puerto en el contenedor>

3.2.2. BlocklyDuino y Blockly

Una de las mejores formas de aprender a programar desde muy temprana edad se basa en la programación por bloques, para eso vamos a utilizar Blockly [6], una biblioteca de código abierto desarrollado por Google la cual integra un editor visual de bloques suele usarse en entornos educativos permitiendo crear aplicaciones conectando diferentes bloques, evitando tener que escribir directamente todo el código.
Blockly presenta claras ventajas como pueden ser:

- **Visual e intuitivo**
- **Personalizable**
- **Uso educativo**

Conociendo Blockly podemos decir que **BlocklyDuino** es una extensión de este, pero específicamente diseñada para trabajar con Arduino, contiene muchos bloques básicos para poder empezar a crear nuestras primeras aplicaciones con Arduino, al igual que Blockly una vez tenemos todo el código se nos mostrará y directamente copiarlo en nuestro IDE para hacerlo funcionar, pero en este trabajo hemos ido más allá automatizando esta tarea como se verá en posteriores puntos.
Al ser un programa de código abierto se nos permite fácilmente añadir nuevos bloques con las funcionalidades que deseemos de forma que tiene unas posibilidades de expansión bastante altas.

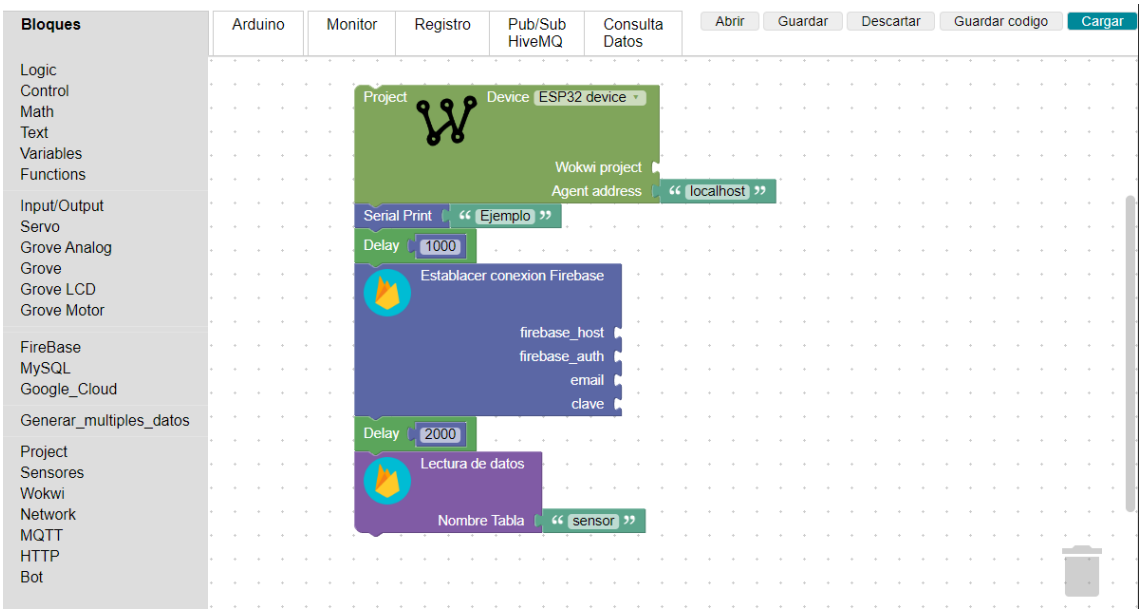


Figura 5. Interfaz BlockyDuino

3.2.3. Flask

Flask [7] es un framework web para Python. Se utiliza para construir aplicaciones web, permitiendo a los desarrolladores crear la parte del backend en una aplicación, se encarga de manejar las solicitudes y devolver las respuestas.

Flask no impone una estructura rígida para el desarrollo por lo cual puede usar el patrón Modelo-Vista-Controlador (MVC) para estructurar la aplicación, también es compatible con otros enfoques, como el Modelo-Vista-Template (MVT).
En una aplicación Flask, el backend (la parte del servidor) se encarga de gestionar las solicitudes que hacen otros servicios o aplicaciones. Los usuarios o sistemas pueden solicitar información o enviar datos a través de varios "endpoints" o puntos de acceso definidos en Flask. La información que se maneja puede provenir de distintas fuentes, como bases de datos o archivos, y se devuelve a la aplicación que hizo la solicitud.

3.2.4. Nginx

“Es un famoso software de servidor web de código abierto. En su versión inicial, funcionaba en servidores web HTTP. Sin embargo, hoy en día también sirve como proxy inverso, balanceador de carga HTTP y proxy de correo electrónico para IMAP, POP3 y SMTP. Ofrece una arquitectura asíncrona y controlada por eventos, característica que hace de NGINX uno de los servidores más confiables para la velocidad y la escalabilidad. [8]
Utilizar este proxy inverso es un plus añadido a la ciberseguridad de nuestro sistema ya que este intercepta todas las peticiones y se comunica con el servidor de forma que el cliente no mantiene una conexión directa, sino mediante el proxy reverso, esto evita que pueda intentar acceder a determinados servicios, además de exponer ciertos puertos que no nos pueda llegar a interesar tener expuesto. Se puede configurar como mejor nos convenga ya que tiene un fichero llamado “nginx.conf” dedicado a ello.

3.2.5. React

React [9] es una biblioteca la cual utiliza JavaScript para construir interfaces de usuario, se suele utilizar en la parte del frontend, últimamente está muy popularizada ya que nos permite crear interfaces interactivas modulares.
React utiliza componentes, los cuales son bloques reutilizables que representan una determinada parte de la aplicación, además presenta una clara ventaja la cual es que puedes ver los cambios realizado en tiempo real ya que utiliza Document Object Model Virtual (Virtual DOM).

Sigue un flujo de datos de manera unidireccional de forma que la información fluye desde los componentes padres a todos los componentes hijo, lo cual facilita el entendimiento del código, algunas de las ventajas de usar React pueden ser:

- **Rendimiento:** Como se ha comentado antes el uso de Virtual DOM ayuda a optimizar mucho el rendimiento de nuestro servicio.

- **Modularidad:** Al tener un funcionamiento basado en bloques permite más fácilmente crear interfaces modulares.

- **Comunidad y soporte:** Tiene detrás una gran comunidad de forma que tenemos disponibles muchas bibliotecas para utilizar.

3.2.6. HiveMQ

Es una plataforma bróker MQTT como también lo es mosquitto, permite la comunicación de mensajes entre dispositivos utilizando el protocolo MQTT, ampliamente utilizado en aplicaciones basadas en Internet de las cosas (IoT). Suele utilizarse en entornos de alta disponibilidad, escalabilidad y fiabilidad.
Presenta una interfaz de administración basada en web que facilita la configuración y la extracción de los datos.

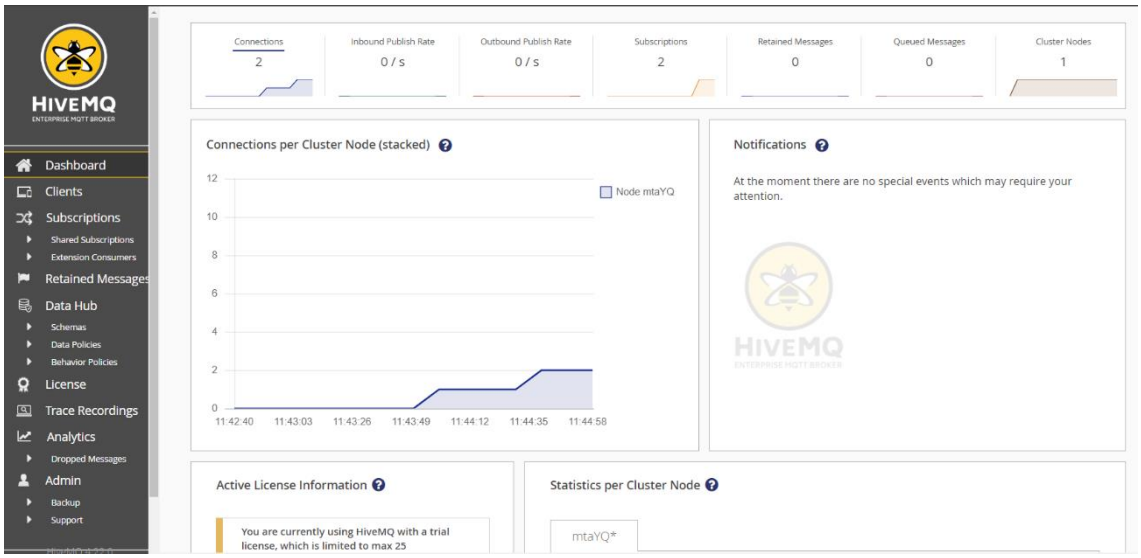


Figura 6. Interfaz HiveMQ

3.2.7. Arduino-create-agent

Arduino créate agent, una herramienta que facilita la conexión desde el ordenador con el entorno Arduino que configuremos, permite a los usuarios interactuar de una manera más eficiente con las placas Arduino a través de una interfaz web sin necesidad de tener que instalar ningún otro software adicional. Esto es muy útil ya que podemos sincronizarlo con blockyduino y cargar directamente el código generado por los bloques en nuestro Arduino.

3.2.8. PlataformIO

PlataformIO [10] es un ecosistema de desarrollo integrado para la programación de sistemas embebidos y proyectos IoT, PlatformIO proporciona un entorno unificado que admite múltiples plataformas y ecosistemas. Puede sustituir al IDE Arduino tradicional y presenta algunas características claves como pueden ser:

- **Multiplataforma**: Presenta compatibilidad con una amplia gama de microprocesadores como puede llegar a ser Arduino uno y Arduino EPS32.

- **Entorno de Desarrollo:** Puede integrarse en diferentes editores de código como puede ser Visual Studio Code.

- **Gestión de librerías:** Al igual que el IDE de Arduino tradicional, nos permite gestionar dependencias y librerías mediante un gestor de paquetes integrado.

3.2.9. Arduino ESP32

Arduino ESP32 es un controlador de bajo coste y alto rendimiento desarrollado por la empresa Espressif Systems. Es muy utilizado en el mundo de IoT debido a su tamaño y capacidad de obtener información de los elementos conectados. A diferencia del Arduino Uno esta plata contiene un módulo Wi-Fi incorporado, dispone de numerosos pones GPIOs que pueden ser usados para conectar diferentes sensores.
Tiene dos pines de 5 Voltios y otro de 3,3V además de tener varias conexiones a tierra (GND).

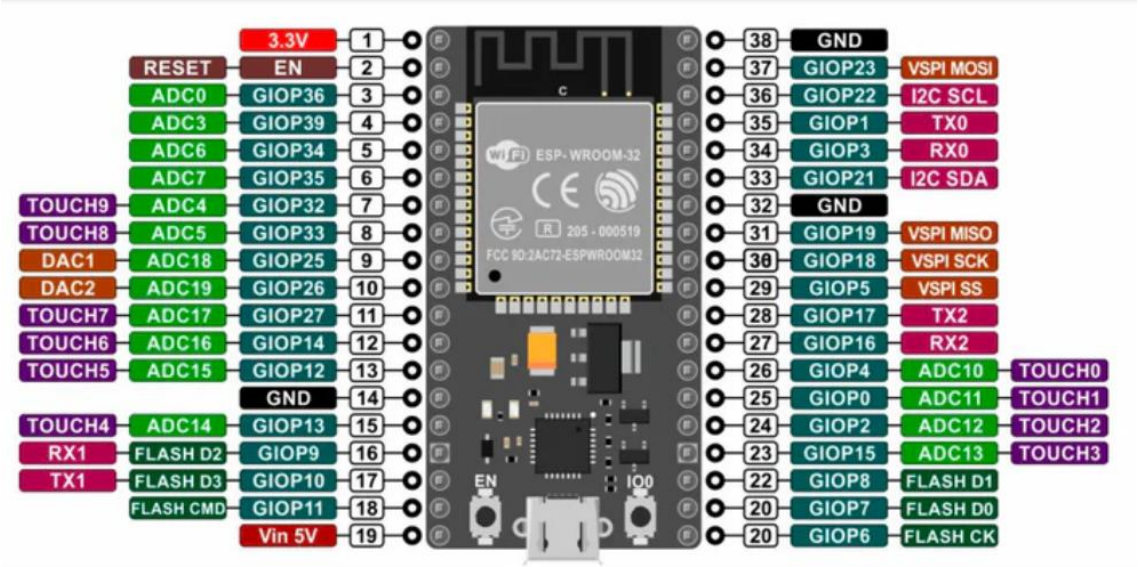


Figura 7. Esquema Conexiones Arduino EPS32

3.2.10. FireBase

Firebase de Google es una plataforma en la nube para el desarrollo de aplicaciones web y móvil. Está disponible para distintas plataformas (iOS, Android y web), con lo que es más rápido trabajar en el desarrollo.
Sus herramientas son variadas y de fácil uso, considerando que su agrupación simplifica las tareas de gestión a una misma plataforma. Las finalidades de estas se pueden dividir en cuatro grupos: desarrollo, crecimiento, monetización y análisis. Es especialmente interesante para que los desarrolladores no necesiten dedicarle tanto tiempo al backend, tanto en cuestiones de desarrollo como de mantenimiento.

Entre las herramientas que proporciona FireBase podemos destacar las siguientes:

- **Firebase Realtime Database:** Es una base de datos en tiempo real la cual permite almacenar y sincronizar datos en tiempo real, tiene amplias ventajas a la hora de crear aplicaciones colaborativas ya que los datos se sincronizan en todos los dispositivos conectados. Los datos almacenados son tipo JSON.

- **Cloud Firestore:** Es una base de datos NoSQL que nos permite almacenar y sincronizar datos en tiempo real, usando una estructura de documentos y colecciones, es parecida a Realtime Database con la diferencia de que permite consultas mucho más potentes y soporte para aplicaciones complejas.

- **Firebase Analytics:** Ofrece herramientas para analizar el comportamiento que tienen los usuarios dentro de la aplicación. Estas métricas son importantes para poder analizar como interactúa el usuario con la aplicación.

3.2.11. MySQL

MySQL es un sistema de gestión de bases de datos relacionales de código abierto, que se utiliza para almacenar, organizar y gestionar gran cantidad de volúmenes de datos, es uno de los sistemas de bases de datos más populares y ampliamente utilizados en el mundo. Algunas de las características que tiene son:

- Base de datos relacional
- Código abierto
- Compatibilidad con SQL
- Escalabilidad

3.2.12. Google Cloud

Google Cloud es una plataforma de servicios en la nube de Google, la cual ofrece una amplia gama de herramientas y servicio para almacenamiento, análisis de datos, machine Learning, inteligencia artificial entre otros.
Nos permite tener una prueba gratuita con un total de 300 créditos para poder probar este servicio. Entre la gran cantidad de servicios que nos ofrece Google Cloud vamos a destacar los utilizados en este trabajo.

3.2.12.1 Cloud Function

Es un servicio sin servidor, diseñado para ejecutar funciones individuales en respuesta a determinados eventos (por ejemplo, cambios en una base de datos, peticiones HTTP). Se utiliza para ejecutar pequeños fragmentos de código, sin necesidad de gestionar una infraestructura para ello.

3.2.12.2 Google SQL

Google SQL es un servicio de bases de datos administrado en la nube que te permite configurar, gestionar y escalar bases de datos relacionales como MySQL, PostgreSQL, y SQLServer sin preocuparte de tenerlo desplegado en local y pudiendo tener una disponibilidad constante.

3.2.12.3 Pub/Sub:

“Pub/Sub es un servicio de mensajería escalable y asíncrono que separa los servicios que producen mensajes de los que los procesan.
Pub/Sub te permite crear sistemas de productores y consumidores de eventos, llamados publicadores y suscriptores. Los publicadores se comunican con los suscriptores de forma asíncrona mediante la transmisión de eventos, en lugar de llamadas de procedimiento remoto (RPC) síncronas.
Los publicadores envían eventos al servicio de Pub/Sub sin tener en cuenta cómo o cuándo se deben procesar. Luego, Pub/Sub entrega eventos a todos los servicios que reaccionan a ellos. En los sistemas que se comunican a través de RPC, los publicadores deben esperar a que los suscriptores reciban los datos.” [3]

En este proyecto vamos a utilizar pub/sub de manera similar a la que usamos HiveMQ pero en un entorno Cloud. Con la ventaja que Google Cloud te permite crear un tema, el cual recibe mensajes y los manda a todos los usuarios subscritos a ese tema, además almacena la información de la publicación.

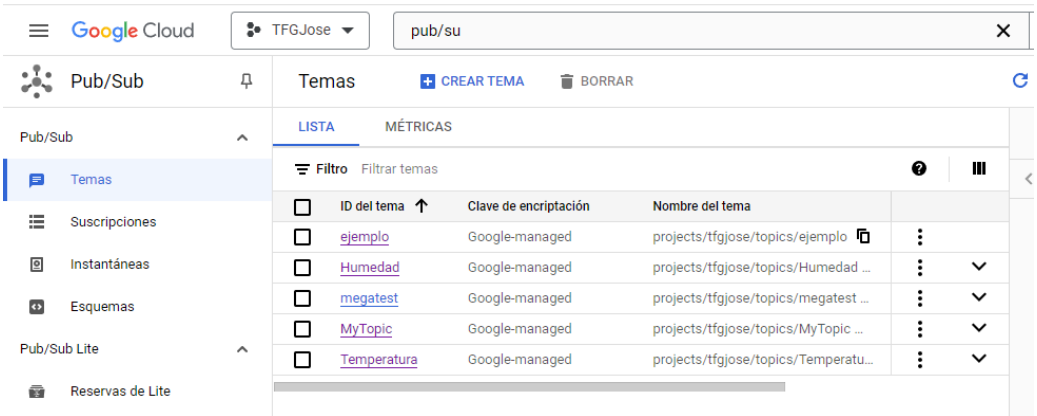


Figura 8. Ejemplo interfaz Google Cloud

3.2.12.4 Redes de VPC

Este servicio ofrece una red virtual privada que permite conectar diferentes recursos en la nube de una manera segura. Esta infraestructura es necesaria para crear y gestionar redes dentro de Google Cloud, se puede asimilar a una red tradicional, pero estando en un nivel virtual y en la nube.

3.2.12.5 Acceso a VPC sin servidores

El acceso a VPC sin servidores se refiere a la capacidad de los servicios sin servidor (serverless), como puede llegar a ser Cloud Run, Cloud Functions, para conectarse de forma segura a una red VPC privada.
Google Cloud permite que estos servicios sin servidor accedan a los recursos internos que están dentro de una red VPC, como bases de datos alojadas en instancias de VM.

3.2.13. Wokwi

Wokwi [17] es una plataforma de simulación en línea para proyectos de electrónica y microcontroladores, como Arduino, ESP32 y Raspberry Pi Pico. Permite a los usuarios diseñar, programar y probar circuitos sin necesidad de hardware físico. Es útil para aprender y prototipar proyectos de manera rápida y accesible. Ideal para educación y desarrollo en electrónica.

3.3. Lenguajes y Protocolos

En esta sección, comentaremos los lenguajes de programación y protocolos fundamentales utilizados en este trabajo. Estos elementos son clave para la comunicación y el desarrollo de las distintas partes de la aplicación, permitiendo la integración y el funcionamiento eficaz del proyecto.

3.3.1. MQTT

MQTT [11] es un protocolo de mensajería ligero y de alta eficiencia diseñado para la comunicación maquina a máquina y el IoT, se utiliza para transmitir datos entre dispositivos, consume muy poco ancho de banda.

Funciona como un modelo Publicador / Subscriptor, en lugar de una comunicación tradicional emisor-receptor utiliza un modelo en el que un dispositivo publica en un determinado tema y todos los subscriptores a ese tema reciben la información en tiempo real, en nuestro proyecto utilizamos un suscriptor embebido en la web para poder recibir en tiempo real información de manera que podamos testear si nuestro dispositivo publicador está mandado correctamente la información.

3.3.2. Python

Python [12] es un lenguaje de programación de alto nivel, es ampliamente utilizado por la cantidad de popularidad y librerías disponibles para todos los campos de la informática como puede ser desarrollo web, ciencia de datos, automatización, inteligencia artificial, entre otros.
Se puede utilizar cada en cualquier plataforma y sistema operativo.

En este proyecto se utilizará para configurar nuestro backend Flask el cual se comunica con nuestra aplicación React y con el navegador web ya que almacena el índex de Blockly.

3.3.3. JavaScript

JavaScript [13] es un lenguaje de programación de alto nivel, utilizado principalmente en sitios web dinámicos e interactivos, se creó para ejecutarse en navegadores web.

Para poder configurar los bloques en Blockly es necesario utilizar este lenguaje, tanto para mostrar la parte visual del bloque como para mostrar el código correspondiente a la unión de esos bloques, también tendrá importancia a la hora de crear la interfaz para nuestro cliente HiveMQ.

3.3.4. C y C++

Son los lenguajes de programación principales de Arduino.

Se va a utilizar este lenguaje principalmente ya que cada vez que el usuario ponga un bloque se va a generar un código Arduino correspondiente al bloque, ya que es el lenguaje que posteriormente ejecutará Arduino.

3.3.5. Http / Https

Hypertext Transfer Protocol (HTTP) [14] y Hypertext Transfer Protocol Secure (HTTPS) [15] son protocolos se utilizarán para la comunicación entre el navegador web y el servidor web, aunque ambos se utilizan para transferir datos a la web existe una diferencia significativa de seguridad.
HTTPS es una versión más segura de HTTP ya que utiliza un cifrado para proteger la integridad y la privacidad de los datos transmitidos entre el navegador y el servidor.

Para la comunicación con el agente de Arduino, el servidor que contiene el servicio Blocklyduino y Nginx vamos a utilizar un protocolo HTTPS por lo cual tenemos que importar los certificados necesarios para ello, y sin embargo para que React extraiga correctamente los datos vamos a utilizar una comunicación HTTP.

3.3.6. SQL

SQL [16] es un lenguaje diseñado para gestionar y manipular bases de datos relacionales, entre las funciones que puede realizar está:

- Insertar
- Seleccionar (Mostar una seria de datos)
- Borrar
- Modificar

Este protocolo se utiliza tanto para la comunicación con FireBase, como MySQL y Google SQL, además de poder obtener los datos y mostrarlos en nuestro cliente React.

3.3.7. Go

También conocido como Golang es un lenguaje creado por Google, tiene una compilación muy simple y rápida, además es utilizado por muchos servicios de Google entre ellos Arduino-create-agent por lo tanto para cualquier modificación de este protocolo se deberá modificar utilizando este lenguaje.

4. Diseño e implementación

Para la implementación del desarrollo vamos a dividir este punto en diferentes bloques significativos, incluyendo despliegue de bloques, carga en el microcontrolador, cliente mqtt, integración en React, generación de plantilla HTML, además del correspondiente a la encapsulación en Docker.

4.1. Bloque 1 – Despliegue de Bloques

En esta parte vamos a describir toda la información relacionada con la programación necesaria para poder generar los bloques en nuestra aplicación BlocklyDuino además de configurar los siguientes bloques: Firebase, MySQL y Google Cloud.
Vamos a utilizar la interfaz gráfica que ya mantiene incorporada BlocklyDuino, con ciertas modificaciones.

4.1.1. Esquema de funcionamiento

Para este bloque vamos a utilizar la siguiente estructura.

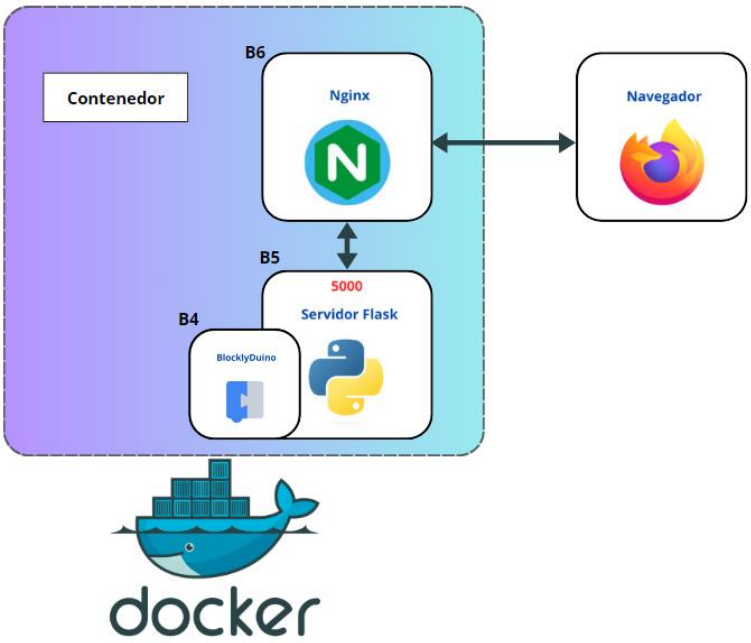


Figura 9. Esquema bloque 1

4.1.2. Servidor Flask

Toda nuestra arquitectura se mantiene sobre el servidor Flask puesto que es el que contiene todos los endpoint tanto para mostrar la información en el navegador como para poder cargar posteriormente los datos en el React.
Para la totalidad del proyecto vamos a necesitar utilizar varios endpoint los cuales son:

- (/): Contiene una Interfaz sencilla la cual nos puede redirigir al servicio React o al índex del servicio BlocklyDuino .
- (/blockly): Se redirige al fichero índex Html de BlocklyDuino.
- (/conexión_select): Como parámetros de entrada necesita la siguiente información: host, user, pass, db, tabla y datos. Estos parámetros se usan para establecer una conexión SQL. Si la base de datos existe, se devolverá como resultado todos los campos especificados en “datos” para la tabla proporcionada.
- (/conexión_insert): Como parámetros de entrada necesita un fichero JSON con la siguiente información: host, user, pass, db, tabla, campos, valores. Estos parámetros se utilizan para mediante una sentencia SQL realizar un Insert del array “valores” en el campo de la tabla. Devuelve un error o un mensaje de éxito si se ha realizado la inserción correctamente.

4.1.3. Nginx

El despliegue se realizará en Docker y no requerirá configuración adicional. El puerto que estará en escucha será el 80, y se han añadido los certificados para permitir la comunicación mediante HTTPS. Una de sus funciones es la redirección de puertos, en este caso, se redirigirá el puerto 8080, aunque se puede redirigir otro puerto modificando la siguiente línea:

```
proxy_pass http://blocklyduino:8080;
```

Esta línea redirige las solicitudes al servidor interno Flask, que se encuentra en el puerto 8080. Para más detalles, consulta el punto 4.6.2.1.

4.1.4. Blockyduino

Como hemos comentado anteriormente Blockyduino es un complemento del servicio Blocky pero enfocado en Arduino, la interfaz BlockyDuino es la siguiente:

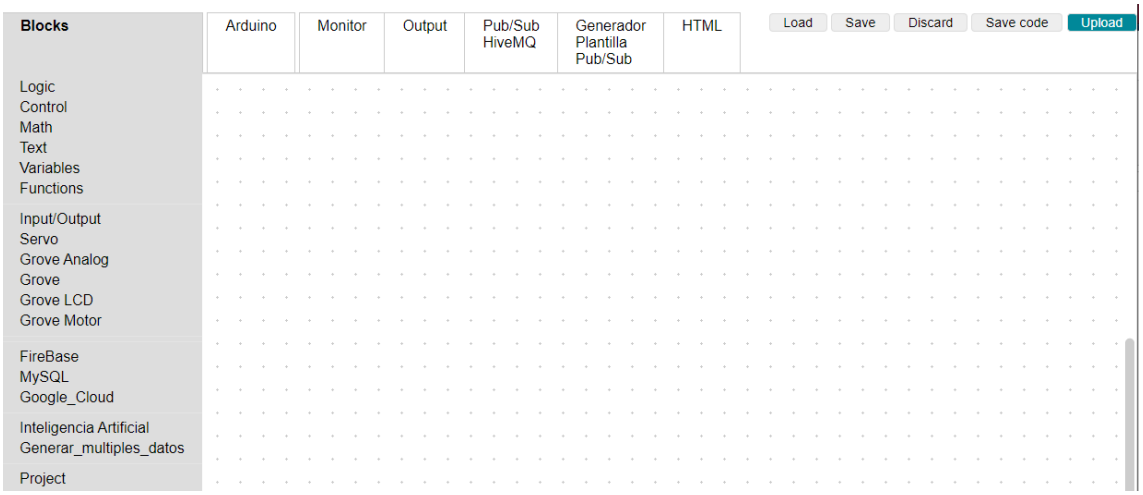


Figura 10. Interfaz de usuario BlockyDuino

1. En la pestaña **Bloques** tendremos todas las categorías, cada una con sus módulos correspondientes, las categorías añadidas son FireBase, MySQL, Google_Cloud, Inteligencia Artificial. Cada categoría añade los bloques implementados en el TFG, cada bloque llama a un código independiente como se explica a partir del punto 4.1.4.1.
2. Para la pestaña **Arduino**, podemos ver el código generado cuando conectamos diferentes bloques, es el código que se carga directamente en nuestra placa Arduino ESP32 y es código compatible con Arduino.
3. **Monitor**, muestra la información enviada al puerto serie de nuestro Arduino de forma que cuando carguemos un código podemos visualizar el Serial.print que se mostraría en nuestro IDE de Arduino.
4. Para **Output,** podemos ver la información que devuelve nuestro Arduino-create-agent, igual que podemos observar los errores de compilación.
5. En **Pub/Sub HiveMQ,** Se tratará más en detalle en el bloque 3, pero muestra un cliente MQTT que podemos configurar y conecta a un bróker, además de elegir los temas a los cuales nos queremos conectar.
6. **HTML,** contiene todo el código generado para crear un subscriptor mqtt en base a los bloques de la categoría MQTT.

4.1.4.1 Categoría FireBase

El objetivo principal de este bloque consiste en el establecimiento de una conexión con una base de datos en tiempo real y poder insertar o mostrar un dato.
Como podemos observar [Figura 11] en la categoría FireBase podemos apreciar tres bloques diferentes, los cuales vamos a detallar a continuación.

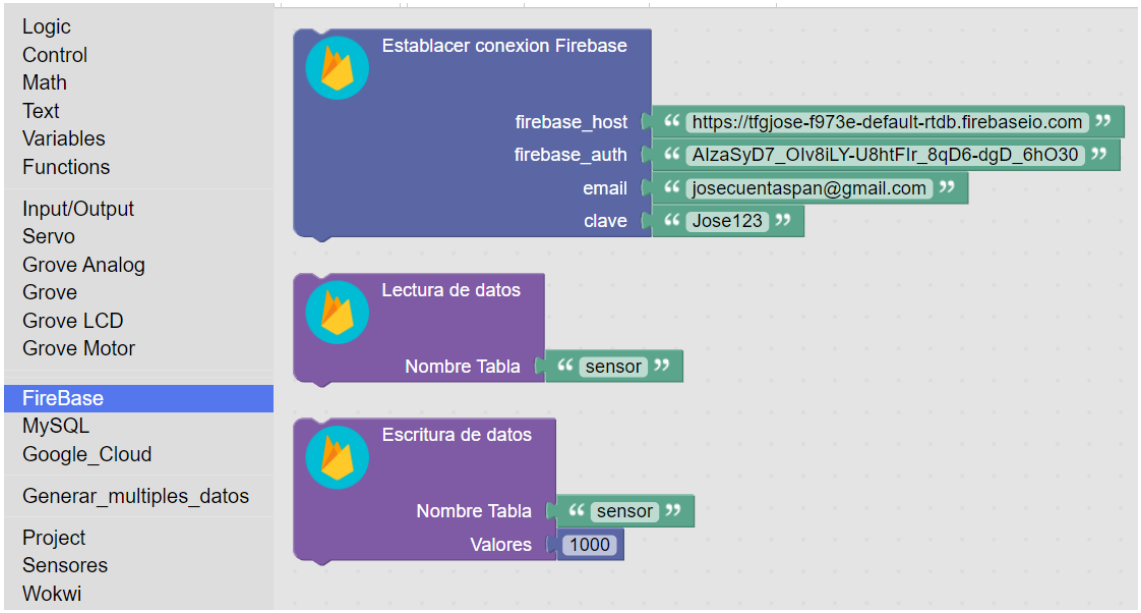


Figura 11. Bloques categoría Firebase

- **Establecer conexión Firebase:** Previamente necesitamos tener creada una base de datos “Firebase Realtime Database” y un usuario valido para poder establecer la conexión.
  Para este módulo necesitamos:

  - firebase_host : Url de la base de datos, disponible dentro de los archivos de configuración en firebase.
  - firebase_auth : Apikey de la base de datos, al igual que el punto anterior también se encuentro en los archivos de configuración.
  - email : Usuario creado dentro de la parte de autenticación
  - clave : Contraseña creada dentro de la parte de autenticación.

  Para ver donde podemos extraer esta información se puede consultar el Anexo II.

Este módulo solamente establece la conexión.

- **Lectura de datos:** Este módulo se puede conectar justo debajo del módulo anterior, puesto que se debe establecer la conexión previamente, solo tenemos un campo el cual es:

  - Nombre_tabla : Nombre de la tabla a consultar.

- **Escritura de datos:** Igual que el módulo anterior previamente debemos tener establecida la conexión y los campos son los siguientes:

  - Nombre_tabla : Nombre de la tabla a insertar los valores.
  - Valores : Valor del campo a insertar.

4.1.4.2 Categoría MySQL

Con el siguiente bloque, buscamos que nuestro Arduino pueda conectarse a cualquier base de datos SQL. Actualmente, no es posible establecer esta conexión directamente desde Arduino, ya que la única biblioteca disponible para realizar la conexión está obsoleta.

Como solución, aprovecharemos que tenemos un servidor Flask desplegado. Utilizaremos los endpoints previamente detallados (conexión_insert y conexión_select) para interactuar con la base de datos.

De este modo, enviaremos los datos necesarios al endpoint desde Arduino y recibiremos la respuesta, la cual será mostrada a través del monitor serie.

La siguiente imagen muestra los bloques MySQL:

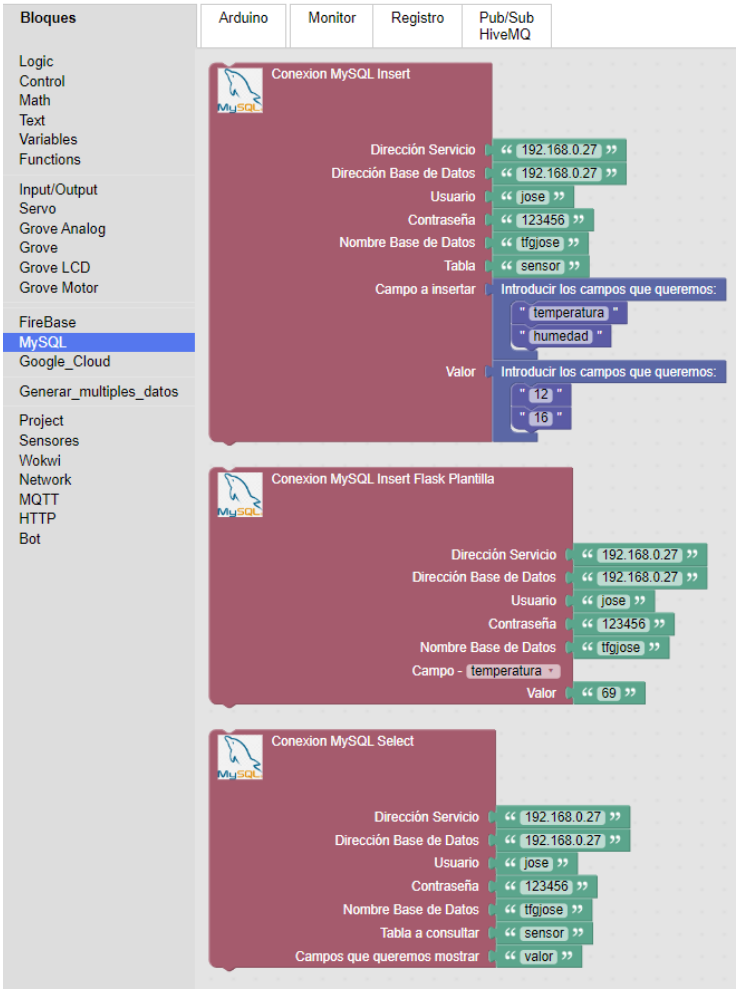


Figura 12. Bloques categoría MySql

- **Conexión MySQL Insert:** Nos permite establecer la conexión con una base de datos e insertar la cantidad de campos que deseemos, debe de tener de forma par los valores campo-valor, los campos que nos proporciona son:

  - **Dirección Servicio**: Ip en la cual se encuentra desplegado el contenedor con todos los servicios.
  - **Dirección Base de Datos**: Ip en la cual se encuentre la base de datos en la cual queremos establecer una conexión.
  - **Usuario**: usuario para la conexión a la base de datos (debe estar creado previamente y tener permisos necesarios de escritura)
  - **Contraseña**: contraseña para el usuario anterior.

  El resto de los campos son autodescriptivos.

Para poder generar un array si queremos insertar varios campos deberemos ir a la categoría “Generar_multiples_datos” el cual contiene los módulos necesarios para crear un array.

- **Conexión MySQL Insert Flask Plantilla:** Bastante similar al anterior, pero este módulo lo vamos a utilizar mayormente para establecer una conexión con el React puesto que mantiene una estructura en la que el campo esta previamente definido (Temperatura, humedad y viento), los demás campos mantienen la misma estructura que el bloque anterior.

- **Conexión MySQL Select**: Es prácticamente igual al bloque anterior, con la diferencia de que se conecta al endpoint "select". En lugar de realizar un insert, ejecutamos un select, por lo que no necesitamos enviar un valor. Deberemos especificar en el campo "campo que queremos mostrar" el nombre del campo que deseamos seleccionar.

4.1.4.3 Categoría Google Cloud

En esta categoría vamos a conseguir a través del método Pub/Sub de Google Cloud realizar con nuestra placa Arduino un Pub de ciertos datos que queramos a un determinado tema. Para poder conseguir esto debemos tener configurada una cuenta en Google Cloud como se define en el Anexo II.

Los bloques de esta categoría son los siguientes:

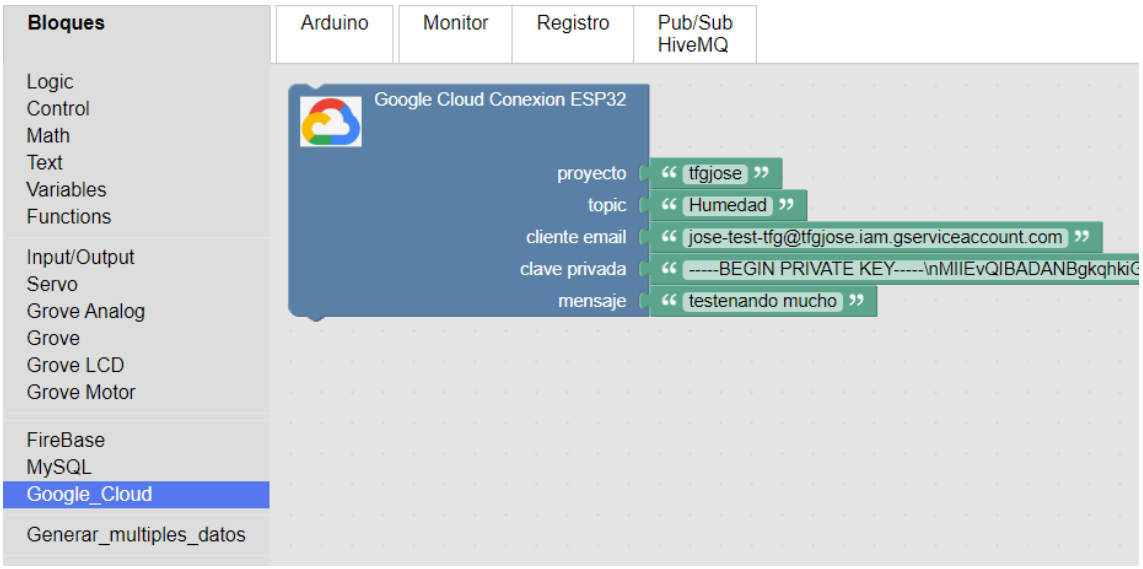


Figura 13. Bloques categoría Google Cloud

- **Google Cloud Conexión ESP32:** Nos permite publicar a través del servicio Pub/Sub de Google Cloud un determinado mensaje, los campos que necesitaremos son los siguientes:

    - **Proyecto**: Nombre del proyecto creado en Google Cloud
    - **Topic**: Tema creado en el servicio Pub/Sub
    - **Cliente email**: Cuenta de servicio la cual contenga los permisos mínimos para poder publicar un determinado mensaje
    - **Clave privada**: Clave generada para poder autenticar que el usuario que publica es el correcto
    - **Mensaje**: Mensaje el cual queremos publicar, puede ser el valor de un determinado sensor

4.1.4.4 Categoría Inteligencia Artificial

Vista la creciente demanda en el campo de la Inteligencia Artificial y aprovechando la API de ChatGPT se ha creado un bloque para establecer una conexión, poder realizar una pregunta y obtener una respuesta por Serial.

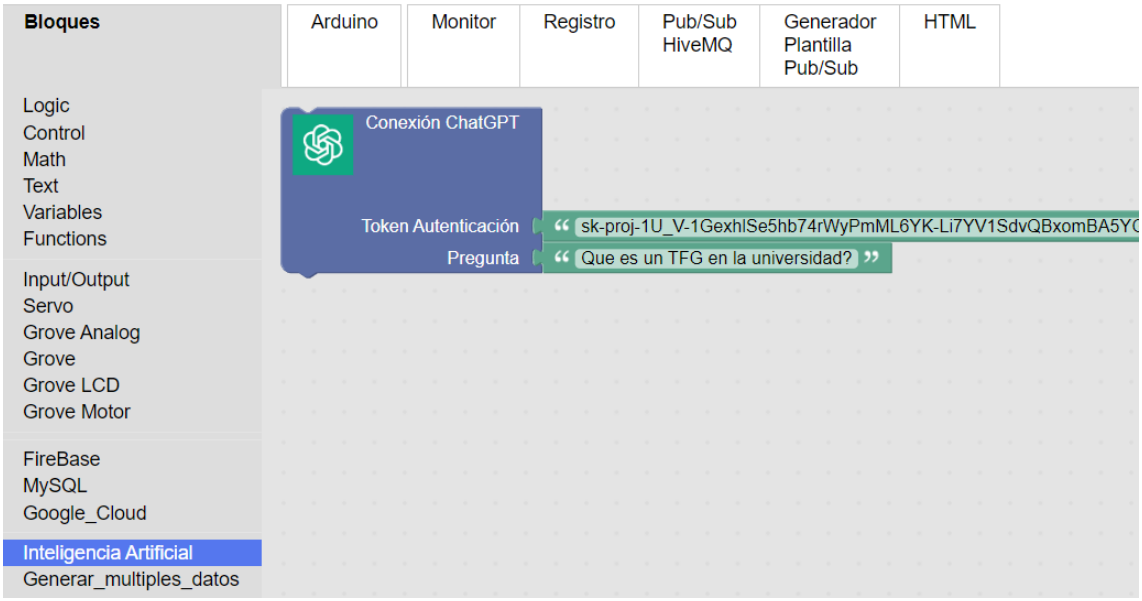


Figura 14. Bloques categoría Inteligencia Artificial

- **Conexión ChatGPT:** Actualmente este módulo mantiene todas las opciones de configuración estaticas, como puede ser el modelo, que directamente está definido como “gpt-4o-mini", nos solicita:

    - Token Autenticación: Bearer token que permite identificar al usuario que realizar la consulta.
    - Pregunta: Consulta que lanzaremos hacia la API

4.1.4.5 Otros

Ciertos módulos han sido creados con anterioridad a este TFG, se van a categorizar como “otros” a los módulos que no pertenecen a ningún bloque particularmente, pero sí que se han modificado y se pueden utilizar de forma complementaria en el resto de los módulos.

4.1.4.5.1 Modulo Bluetooth

Como modulo existente anterior se ha realizado una modificación y se ha creado un módulo bluetooth adicional para la placa Arduino Eps32, puesto que esta placa tiene incorporado un modo bluetooth que podemos utilizar, el módulo es el siguiente:

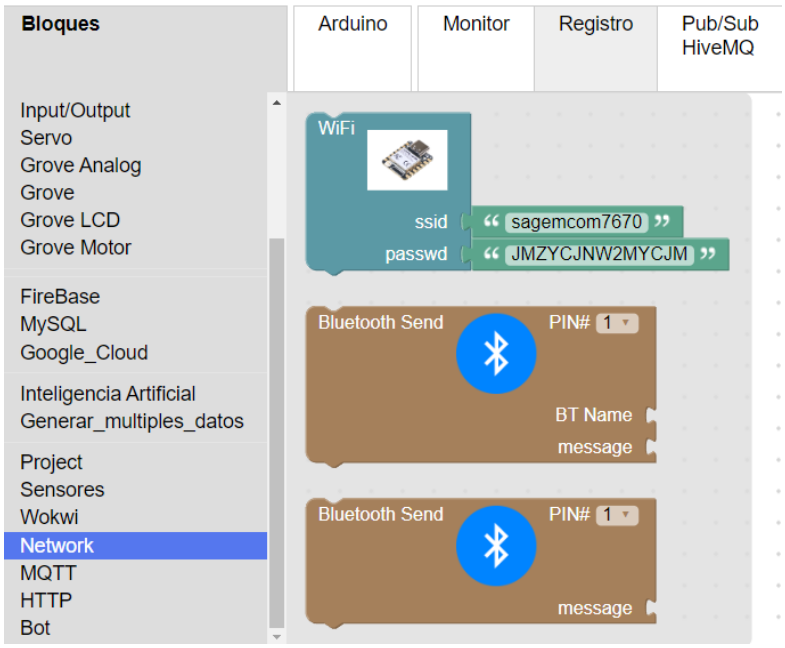


Figura 15. Bloque categoria Bluetooth

El cual solamente tienes dos campos:

- BT Name: Nombre del dispositivo el cual queramos que aparezca en la red Bluetooth
- Menssage: Mensaje que queremos enviar al conectarnos con este dispositivo

4.1.4.5.2 Módulo Array

En si no es un módulo, sino un conjunto de dos bloques el cual nos permite crear un array de campos. Se puede utilizar en módulos para insertar más de un valor en uno de los espacios, como podría ser en el caso de insertar o seleccionar campos de una base de datos, se implementó como solución a no tener que añadir un numero determinados de campos en el bloque.

Con esta forma teniendo un único conector en el bloque podemos añadir tantos valores como deseemos, tiene la siguiente forma.

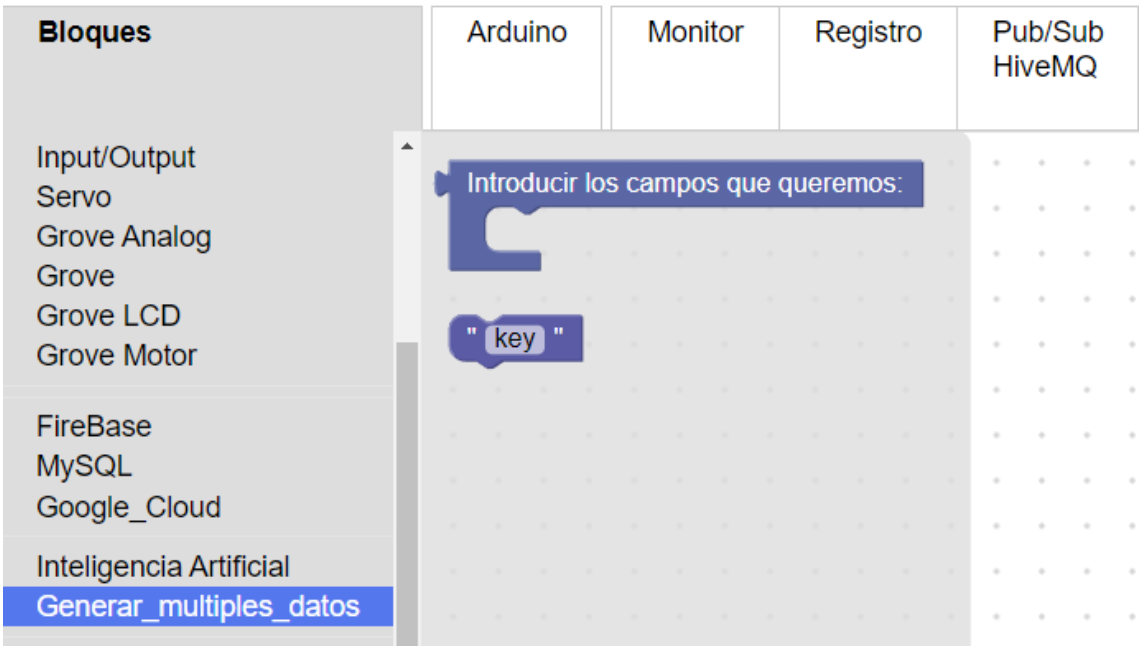


Figura 16. Bloque categoría generar datos

El primer bloque funciona como unas llaves de array, por otro lado, key es cada uno de los campos que queramos añadir, en código tiene una apariencia como la siguiente:

{key1, key2, ...}

4.2. Bloque 2 – Carga en microcontrolador a través del agente

El siguiente bloque es muy importante puesto que se usará Arduino Create Agent, una aplicación que se ejecuta en segundo plano permitiendo la comunicación entre el entorno de desarrollo y el hardware conectado, como una placa Arduino.
La función principal se basa en la carga de código en una placa directamente desde el navegador, ya que el agente actúa como intermediario entre el editor web y el puerto serie del ordenador donde está conectado el dispositivo.

4.2.1. Esquema de funcionamiento

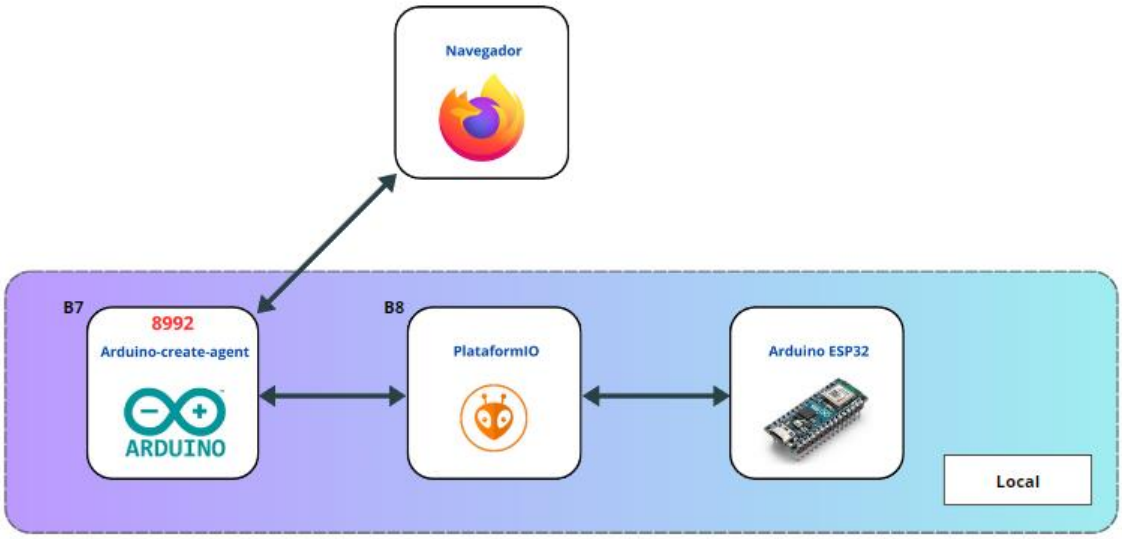


Figura 17. Esquema bloque 2

4.2.2. Arduino-create-agent

Dentro de Arduino-create-agent podremos encontrar 3 archivos principales de configuración los cuales son:

- **Main.go** : Contiene la función principal que ejecuta el programa, como la configuración de las dependencias, inicialización de hub ...

El agente original de Arduino mantiene una restricción de seguridad que solo permite recibir comandos de aplicaciones web que provengan de dos sitios web específicos: create.arduino.cc y cloud.arduino.cc. Esto significa que, si alguien intenta enviar comandos al agente desde una aplicación web que esté alojada en otro sitio (por ejemplo, desde un servidor local o cualquier otro dominio que no sea uno de esos dos), el agente no aceptará esos comandos.

En el caso del agente extendido, va a permitir la recepción de comandos 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.

- **Serial.go** : Contiene la lógica relacionada con la comunicación por puerto serie, en la interacción con los dispositivos a través de la interfaz serie se encargaría de la lectura y escritura de datos.

- **Hub.go** : Maneja de forma principal la lógica de mensajes, contiene todos los comandos necesarios para ejecutar dentro de nuestro servicio.

Todo el conjunto de ficheros se ejecuta dentro de "localhost:8992", el cual es accesible de forma individual para realizar cualquier de los comandos previamente programados, entre los comandos disponibles se han añadido los siguientes:

- **List** : Muestra información del dispositivo conectado, nos servirá para poder detectar el dispositivo para poder cargar el código, entre los valores que devuelve tenemos el nombre del puerto serie, el ID del vendedor y del producto.
- **Lib** : Permite instalar o desinstalar cierta librería en nuestro PlataformIO de manera que desde la misma interfaz se podrá configurar la instalación de librerías.
- **Donwload plataformio**: Permite la descargar del Software PlataformIO para poder gestionar el código que se cargará previamente.
- **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.

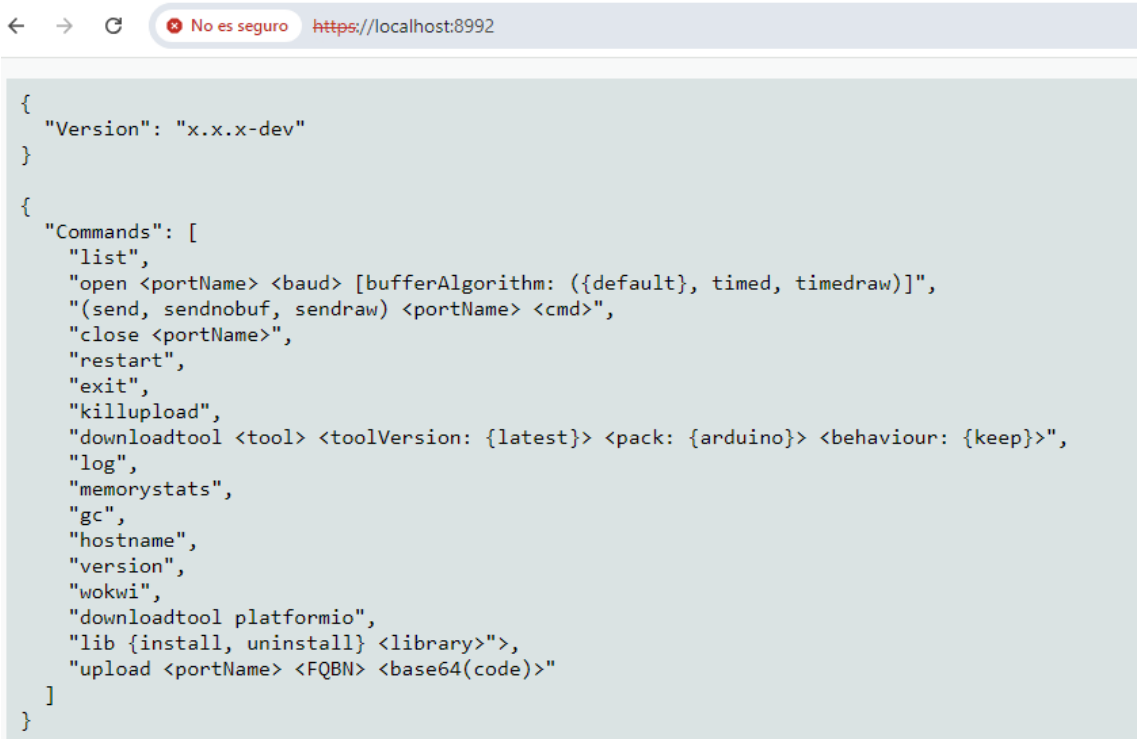


Figura 18. Comandos Arduino-create-agent

4.2.3. Interfaz

Dentro de nuestra interfaz, se añaden tres pestañas las cuales son:

- **Arduino**: Nos permite visualizar el código generado por los bloques conectados, de forma que nos permita una visualización o una importación de ese código a cualquier IDE.

  Esto es posible principalmente a que en la siguiente ruta: “generator/arduino” tenemos el código correspondiente a cada bloque, de manera que en el siguiente apartado se podrá visualizar el contenido de los múltiples bloques seleccionados, el código se estructura de igual manera que se distribuye en el generador, el cual es el siguiente:

  1. Dependencias
  2. Variables
  3. void setup ()
  4. void loop ()

El código se actualiza en tiempo en real de manera que conforme se añadan o eliminen los bloques se actualizará con el código correspondiente a los bloques actuales.

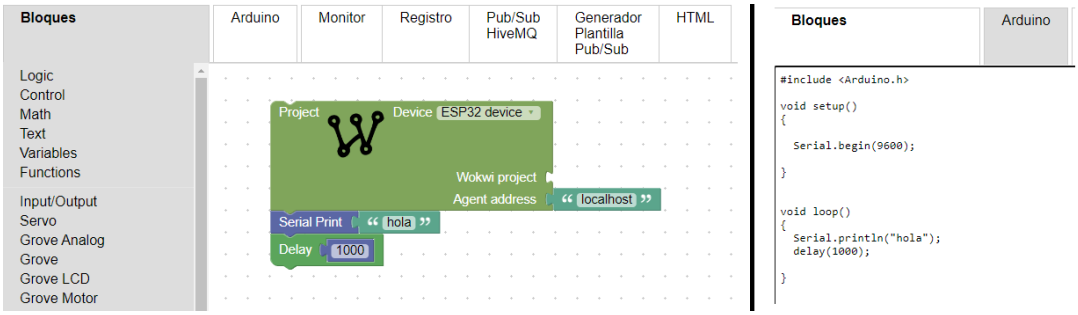


Figura 19. Interfaz Arduino

- **Monitor**: Sincronización a través de la dirección del agente con arduino-create-agent de forma que se mostrará toda la información proporcionada por el serial.
  Esto es posible ya que cuando se carga el código, nuestro agente nos devuelve por consola dos variables después de la carga de código, la cual corresponde a la letra “P” - para indicar el puerto el cual envía la información del serial, y la letra “D” - Envía la información del serial.
  Aprovechando esto solamente se ha cargado la información devuelta por el agente, filtrada, seleccionando solamente la información acompañada de la letra “D” pudiendo simular en nuestra página una interfaz Arduino.

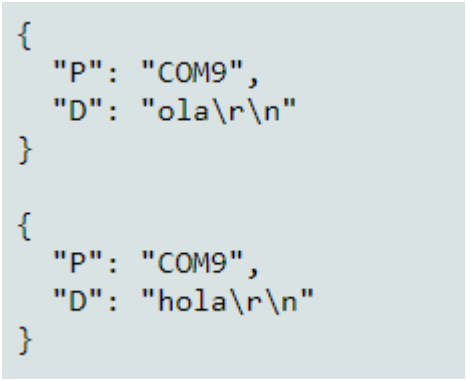


Figura 20. Información agente

Figura 21. Interfaz monitor

- **Registro**: Una vez generado el código de los bloques, se copiará ese código dentro de la dirección de nuestro arduino-create-agent y podremos ver en la pestaña "Registro" todo el proceso de carga del código generado.

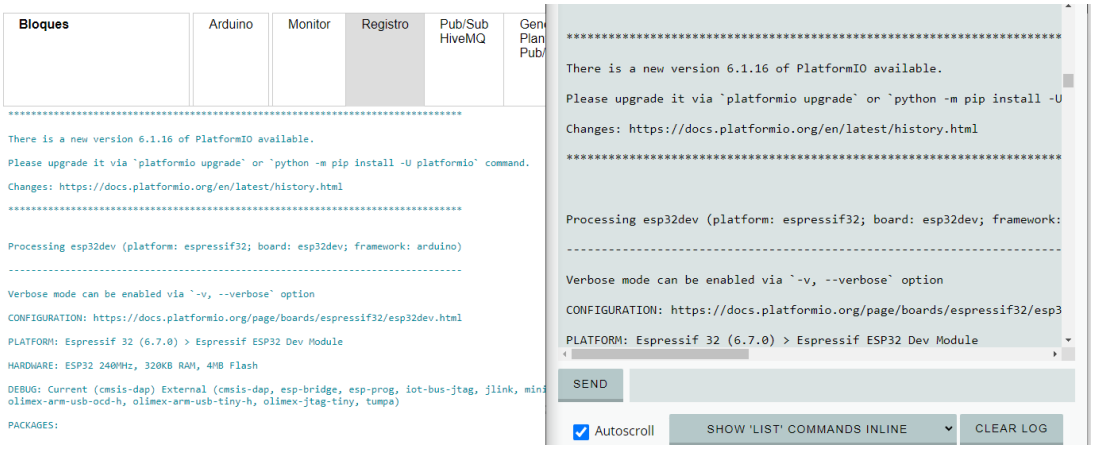


Figura 22. Interfaz registro

4.2.4 Funcionamiento Carga de código

El funcionamiento del botón de carga es bastante simple, el cual ejecuta el siguiente código al clicarlo, vamos a dividirlo en diferentes bloques y explicando la funcionalidad que tendrá.

```
var from_tab = selected;
tabClick('arduino')
area = document.getElementById('content_arduino');
area.focus();
area.select();
```

```
try {
  document.execCommand('copy');
} catch (err) {
  console.error('Unable to copy to clipboard', err);
}
tabClick(from_tab);
```

Seleccionamos la pestaña “Arduino” y copiamos al portapapeles toda la información que este contiene, en caso de que fallase se mostraría un error por consola.

```
var type = Blockly.Arduino.wokwi_device;
var open_url = '';
var wokwi_url = Blockly.Arduino.wokwi_project;
wokwi_url = wokwi_url.substring(1, wokwi_url.length-1);
var agent_url = "";

if (agent_connected == 0){
  agent_url = "https://" + eval(Blockly.Arduino.wokwi_agent) + ":8992/";
  socket = io(agent_url);
  agent(type);
  agent_connected = 1;
}
else {
  socket.disconnect();
  agent_url = "https://" + eval(Blockly.Arduino.wokwi_agent) + ":8992/";
  socket = io(agent_url);
  socket.connect();
}
```

Obtiene el tipo del dispositivo Arduino en base a los ID que nos proporciona el comando list, si no estamos conectados al agente se establece la conexión a un servidor WebSocket introduciendo la dirección + puerto e indicamos que establecimos la conexión cambiando la variable “agent_connected” a 1.

```
switch (type){
    case "open": {
      open_url = wokwi_url;
      break;
    }
    case "ino": {
      open_url = "https://wokwi.com/projects/new/arduino-uno";
      break;
     }
    case "esp": {
      open_url = "https://wokwi.com/projects/new/esp32";
      break;
    }
    case "unowifi": {
      open_url = null;
      break;
    }
    case "esp32dev": {
      open_url = null;
      break;
    }
  }
```

Viendo el tipo de dispositivo se establecerá la opción a seguir, en el caso de esp o ino se establece la url para la creación de un proyecto nuevo y el resto de los tipos de dispositivos se pondrán como nulos.

```
  if (open_url != null) {
    if (nwin!=null) nwin.close();
    nwin = open(open_url, "wokwi_tab");
    socket.emit("command", "wokwi");
  }
  else {
    encoded = btoa(Blockly.Arduino.workspaceToCode()); // base64(code)
    socket.emit("command", "list");
    cmd_on = cmd_list;
  }
```

Si la variable no es nula abrimos una nueva ventana con la url correspondiente a open_url asociada al nombre wokwi_tab y mandamos a través de WebSocket el comando wokwi, si fuera nula la variable convertimos el código a base 64 y mandamos el comando list para visualizar los datos del dispositivo conectado.

4.3. Bloque 3 – Cliente MQTT

Para esta parte vamos a detallar como se ha creado la interfaz necesaria para implementar un cliente MQTT en nuestra interfaz de manera que podamos usarlo para comprobar que datos se están cargando en el bróker HiveMQ.

4.3.1. Esquema de funcionamiento

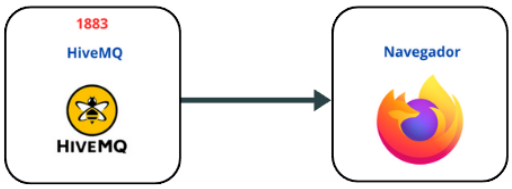


Figura 23. Esquema bloque 3

4.3.2. HiveMQ

Para poder realizar tanto suscripciones como publicaciones vamos a utilizar el Software HiveMQ junto con MQTTExplorer para poder simular ciertas publicaciones, nuestro navegador solo funcionará como suscriptor por lo cual MQTTExplorer lo vamos a utilizar como publicador (como caso ejemplo), como se comentaba en los puntos anteriores HiveMQ ya se encuentra desplegado en nuestro contenedor, aunque también podemos encontrarlo en la Nube o en Local.

4.3.3. Publicador MQTTExplorer

El siguiente Software nos permite tanto suscribirnos como publicar ciertos datos al tema que queramos dentro del bróker en el que establezcamos la conexión.

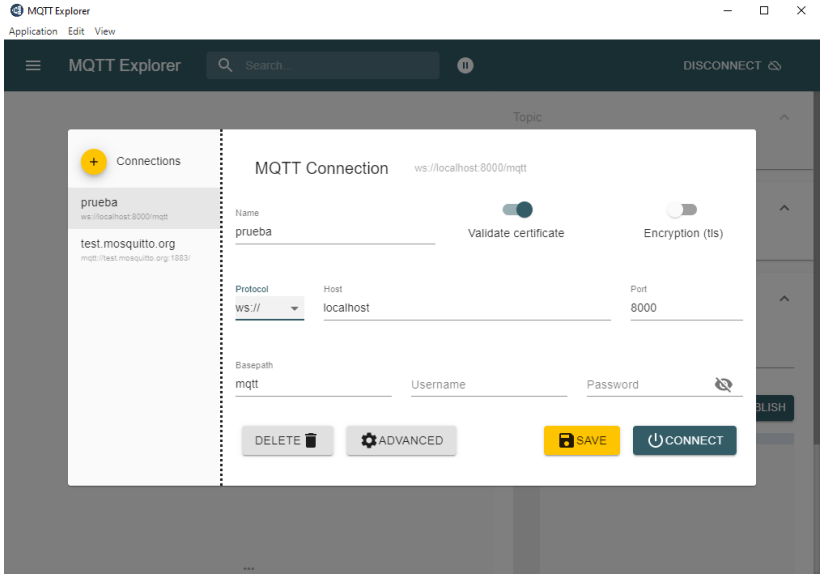


Figura 24. Interfaz MQTT Explorer

La interfaz es bastante sencilla la cual te permite según el protocolo WebSocket o Mqtt conectarnos a un host y un puerto (en el cual tengamos alojado nuestro Broker) y poder empezar a publicar mensajes.
La interfaz para publicar mensajes también es bastante sencilla en la cual solamente tenemos que escribir nuestro tema y el valor que queremos asignar a dicho tema.

4.3.4. Interfaz Cliente MQTT

Para que todo esto tenga una coherencia necesitamos un suscriptor ya que, aunque el publicador si publique los datos, al no tener ningún suscriptor nadie recibirá esos datos. Para esto hemos desarrollado la siguiente interfaz la cual se incorpora en una pestaña adicional de nuestro Index principal llamada “Pub/Sub HiveMQ”.

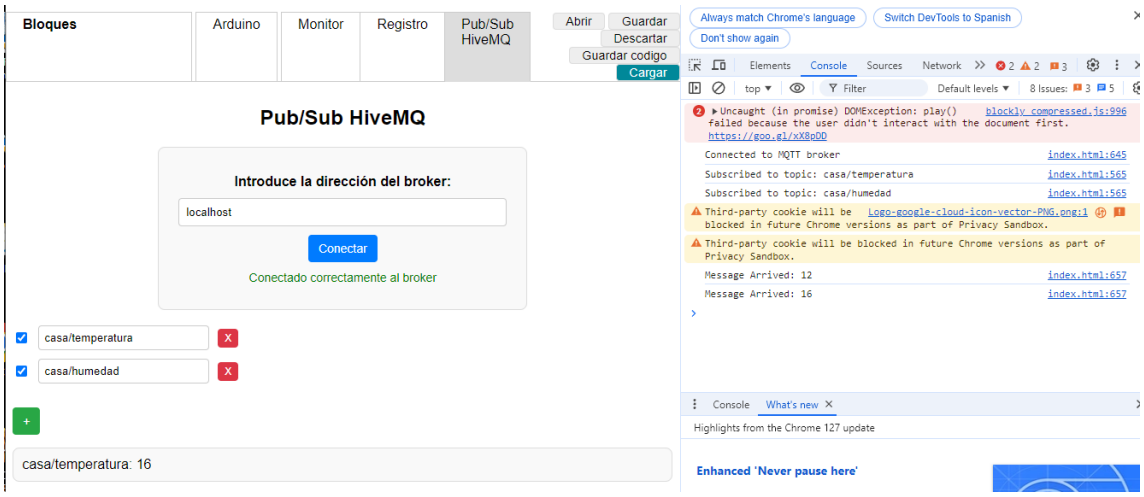


Figura 25. Interfaz Suscriptor HiveMQ

La cual mantiene un diseño bastante sencillo, el funcionamiento se basa principalmente en utilizar el script disponible para un cliente mqtt.

<script src="https://unpkg.com/mqtt/dist/mqtt.min.js"></script>

Gracias a esto, podemos conectarnos al bróker que se introduzca en el formulario, una vez conectado se puede ir escribiendo los temas de los cuales se espera recibir la información que se publique, contiene una casilla para suscribir y desuscribirse del tema, aunque también puedes borrar directamente la suscripción con el botón (x).
Una vez subscrito todos los mensajes emitidos por el publicador se recibirán en tiempo real para poder consultarlos fácilmente.

4.4. Bloque 4 – Integración de React

Esta parte es complementaria puesto que no es obligatorio su uso, es bastante útil para poder visualizar nuestros datos, se enfoca principalmente en la visualización de datos correspondientes a sensores, pero en una línea futura se podrá modificar para poder contener cualquier tipo de dato. Para funcionar es necesario utilizar un frontal que en este caso será React y un backend el cual será un servicio Flask con varios endpoints.

4.4.1. Esquema de funcionamiento

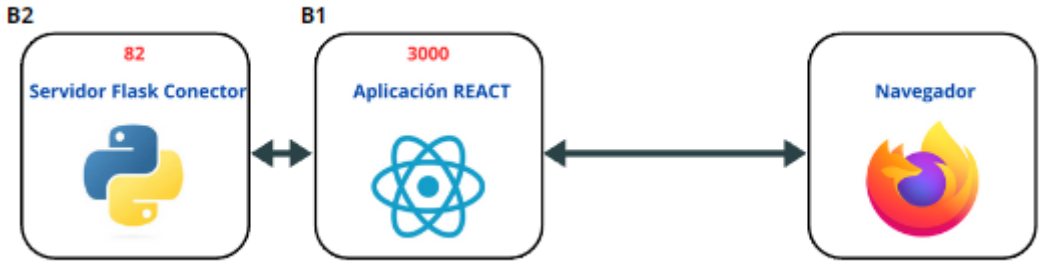


Figura 26. Esquema bloque 4

4.4.2. Funcionamiento Flask 2

Para el correcto funcionamiento del despliegue web con React es necesario apoyarse en un backend como puede ser un servicio Flask, el cual se encargará de las peticiones a la base de datos, extracción de información y envío al backend, para ello se han creado diferentes endpoint los cuales serán consultados para extraer los datos necesarios, a continuación, se van a detallar cada uno de los endpoint y el uso que pueden tener.

- /datos: Se le deberá proporcionar unos datos de conexión que deberán ser el host de la base de datos, usuario, contraseña, nombre de la base de datos y una o varias tablas, el endpoint se encargará de realizar la conexión a la base de datos y devolver todos los datos que contenga la tabla.
- /tablas: Solo se deberá entregar los datos de conexión (host de la base de datos, usuario, contraseña, nombre de la base de datos) y nos devolverá la totalidad de tablas que contiene dicha base de datos.

- /columnas: Igual que el endpoint anterior se deberá proporcionar el host de la base de datos, usuario, contraseña, nombre de la base de datos y una tabla, este endpoint devolverá el nombre de todas las columnas que contenga esa tabla.
- /datos_seleccionados: permite obtener datos específicos de varias tablas en una base de datos MySQL, pero con la particularidad de que solo se seleccionan columnas específicas de cada tabla, se deberá enviar los datos de conexión (host, usuario, contraseña, nombre de la base de datos), una lista de tablas a consultar y un diccionario que contenga las columnas seleccionadas para cada tabla, devolverá los datos seleccionados de las tablas seleccionadas.

4.4.2. Funcionamiento React

Una vez tenemos los endpoints a consultar, se detallará el funcionamiento del servicio React y la información que mostrará, la conexión se podrá realizar tanto como una base de datos local como una base de datos en nube (por ejemplo: SQL Cloud).
La interfaz se muestra a continuación:

Figura 27. Interfaz React

La primera interfaz es bastante simple, puesto que solamente pedirá la información de conexión para satisfacer los datos necesarios en el endpoint (tablas) puesto que al introducir los valores correctamente nos devuelve una lista completa de todas las tablas de la base de datos a la que nos hemos conectado, como se muestra a continuación:

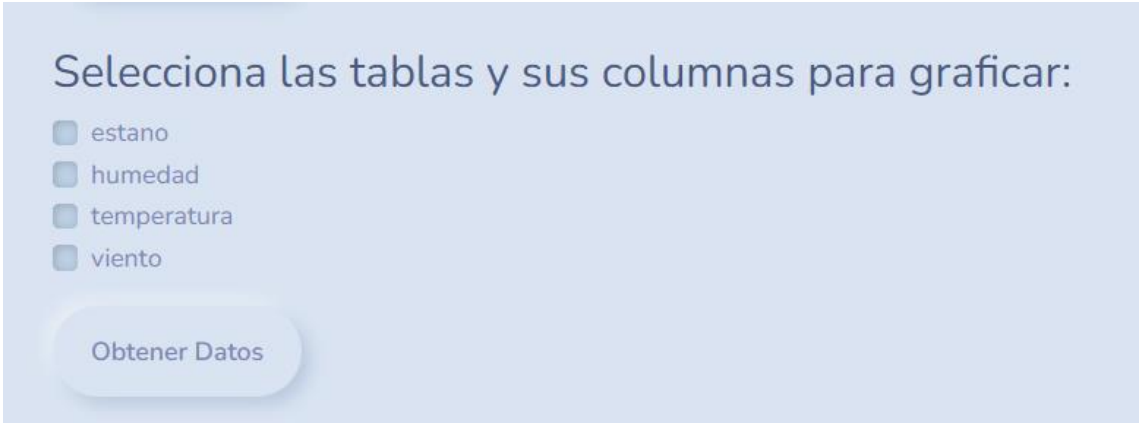


Figura 28. Interfaz tablas

Una vez que nos devuelve todas las tablas, tendremos que seleccionar que tabla o tablas queremos representar, al clickear en ellas se abrirá un menú como el que se muestra a continuación con todos los campos de dicha tabla.

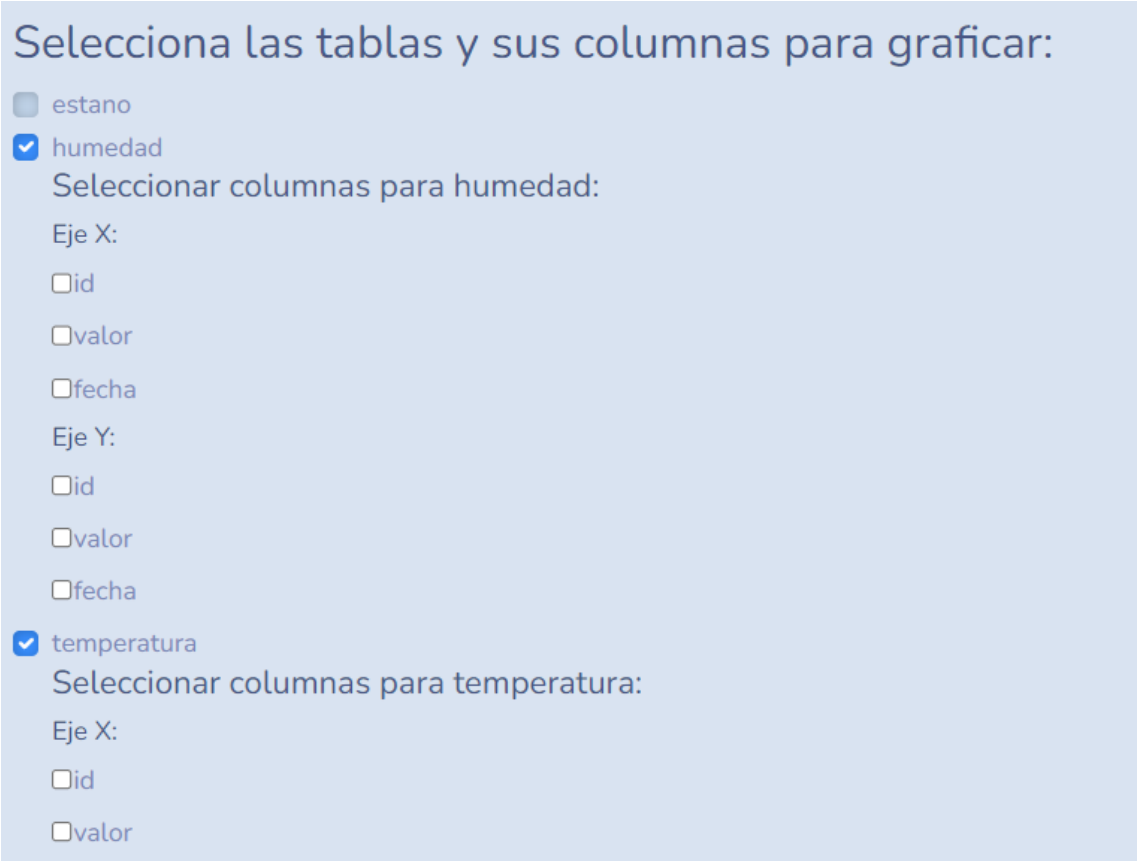


Figura 29. Campos dentro de cada tabla

Para representar cada grafica solamente tendremos que seleccionar los campos que queremos mostrar y a continuación pulsar el botón, cada grafica tiene unos colores diferentes hasta un máximo de 6 colores, un ejemplo se muestra a continuación con la dupla de valores id-valor y valor-fecha de las tablas humedad y temperatura respectivamente.

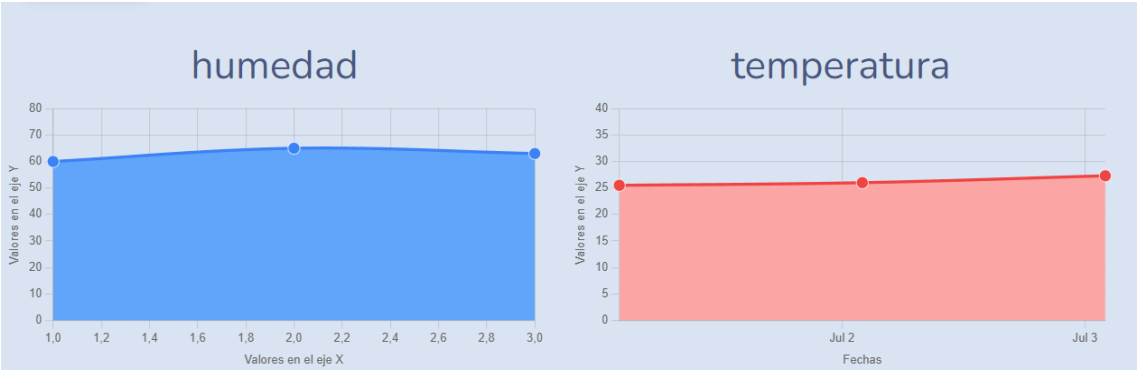


Figura 30. Gráficas de los datos obtenidos

Los menús se despliegan de forma modular de forma que conforme completes datos te muestra la siguiente información, a continuación, se muestra la página completa.

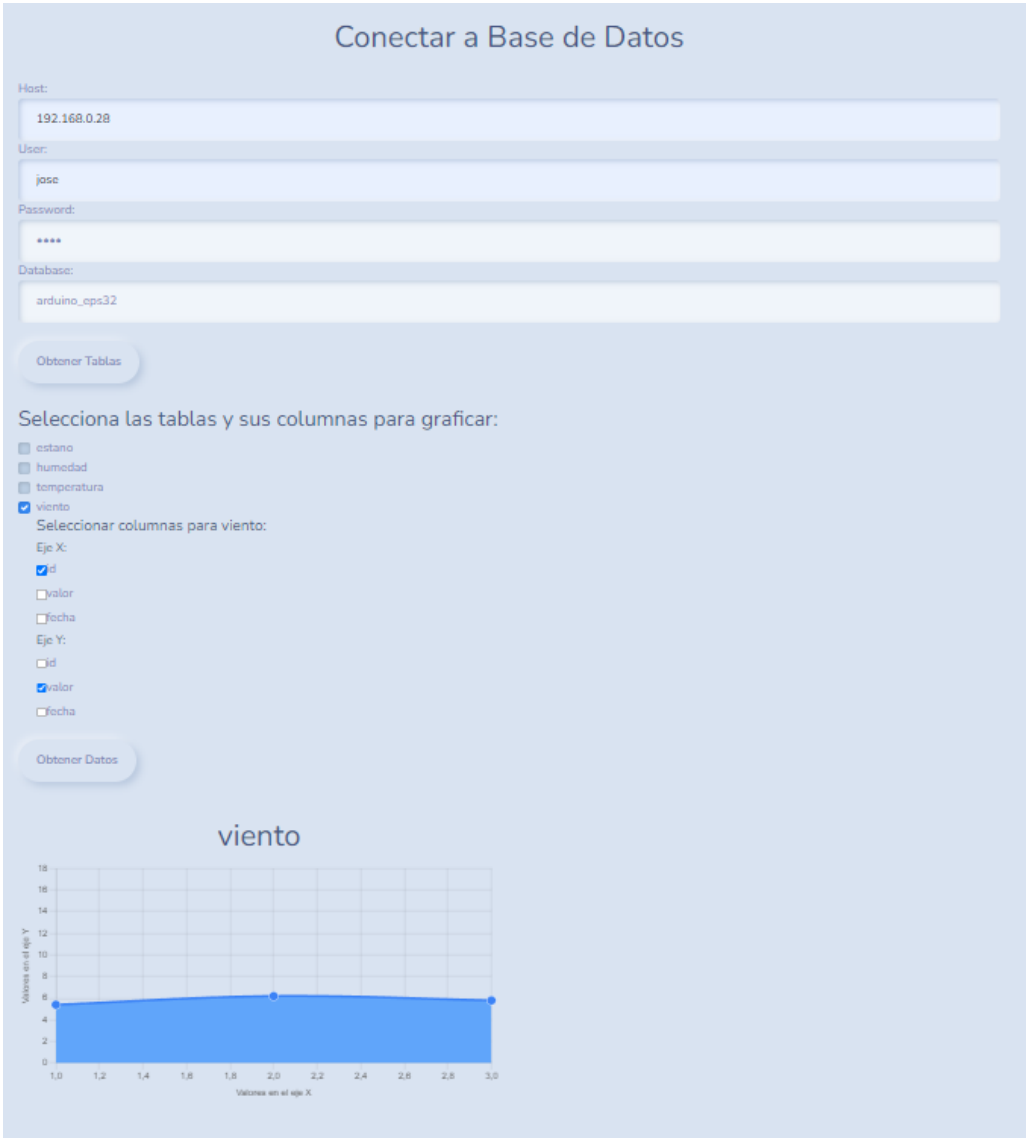


Figura 31. Ejemplo completo

4.5. Bloque 5 - Generar plantilla HTML vinculado al publicador MQTT

Otra utilidad incorporada al proyecto consiste en la creación de un código HTML el cual interactúa con los módulos de la categoría “MQTT” de forma que pueda extraer cierta información para realizar una conexión mqtt y visualizar los valores.

4.5.1. Funcionamiento

Como en el punto anterior se añadirá una pestaña adicional en nuestro fichero Index principal la cual podrá encontrar como “HTML”.
Para el correcto funcionamiento se deberá tener conectado el módulo MQTT Client y MQTT publisher puesto que se necesitará la variable host y topic de dichos módulos. La interfaz que se presenta es la siguiente:

Figura 32. Interfaz pestaña HTML

4.5.1.2 Código

El código correspondiente para el funcionamiento se puede establecer en dos bloques, el primero correspondiente a la extracción de datos de los módulos mencionados y el segundo en la sustitución de dichos datos.
Para la primera parte se puede apreciar el siguiente código:

```
}else if (content.id == 'content_html') {
  var xml = Blockly.Xml.workspaceToDom(Blockly.mainWorkspace);
  // Idenfifica los bloques de tipo network_mqtt_broker
  var element = xml.getElementsByTagName('block');
  var block_name = 'network_mqtt_broker';
  var eval_string = "node.getElementsByTagName('value')[0].getElementsByTagName('block')[0].getElementsByTagName('field')[0].textContent";
  var broker = getValue(element, block_name, eval_string);
  // Idenfifica los bloques de tipo network_mqtt_publisher
  block_name = 'network_mqtt_publisher';
  eval_string = "node.getElementsByTagName('value')[0].getElementsByTagName('block')[0].getElementsByTagName('field')[0].textContent";
  topic = getValue(element, block_name, eval_string);
  document.getElementById("content_html").value = html_code(broker, topic);
}
```

Figura 33. Extracción valores

En el cual se buscará el módulo “network_mqtt_broker” y se obtendrá el primer elemento, al igual que se obtendrá del módulo “network_mqtt_publisher”.
Una vez obtenidos estos datos se deberán sustituir dentro del fichero template.html, lo cual se realizará como se detalla en el siguiente fragmento:

```
function html_code(broker, topic){
var code = "";
 if (broker != undefined) {
    var client = new XMLHttpRequest();
    client.open('GET', 'template.html', false);
    client.onreadystatechange = function() {
      code = client.responseText;
      code = code.replace("<%= host %>", broker);
      code = code.replace("<%= topic %>", topic);
    }
 client.send();
 } else code = "No MQTT Client block found";
 return code;
}
```

Figura 34. Sustitución de valores

El fichero template.html ya se encuentra incorporado en el proyecto, pero con el fin de proporcionar herramientas para la experimentación o desarrollo se ha incorporado la pestaña “Generador de plantilla Pub/Sub” la cual contiene una réplica del fichero

template.html por si ocurriera un borrado o modificación de dicho fichero simplemente se deberá copiar ese código y crear otro fichero con el mismo nombre.

Como último, el código de la pestaña “HTML” se deberá pegar en cualquier interpretador de código Html y javascript para establecer la conexión con el broker Mqtt y la visualización del topic seleccionado, la cual se visualizará como el siguiente ejemplo:

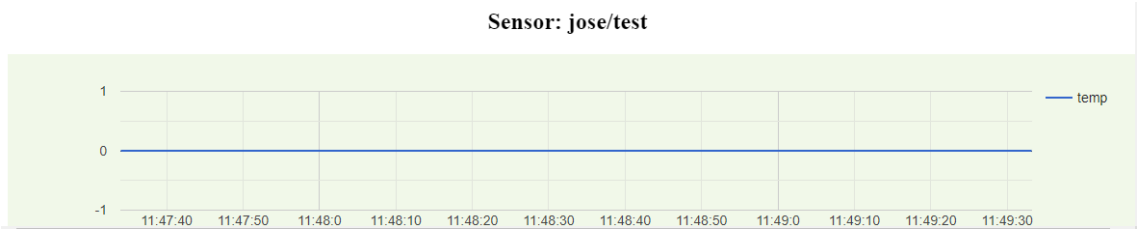


Figura 35. Ejemplo visual de datos

4.6. Despliegue en Docker

Para mejorar la portabilidad de este trabajo se encapsulará los bloques principales en un Docker para poder desplegarlos en conjunto en cualquier ordenador, de manera que también ahorraremos la instalación de dependencias, librerías…
Docker como hemos comentado anteriormente es muy útil ya que tiene un funcionamiento parecido a una máquina virtual, pero ahorrando muchísimos recursos al no tener que instalar la imagen completa del sistema operativo.
Vamos a detallar la configuración de cada uno de los contenedores.

4.6.1. Esquema Docker

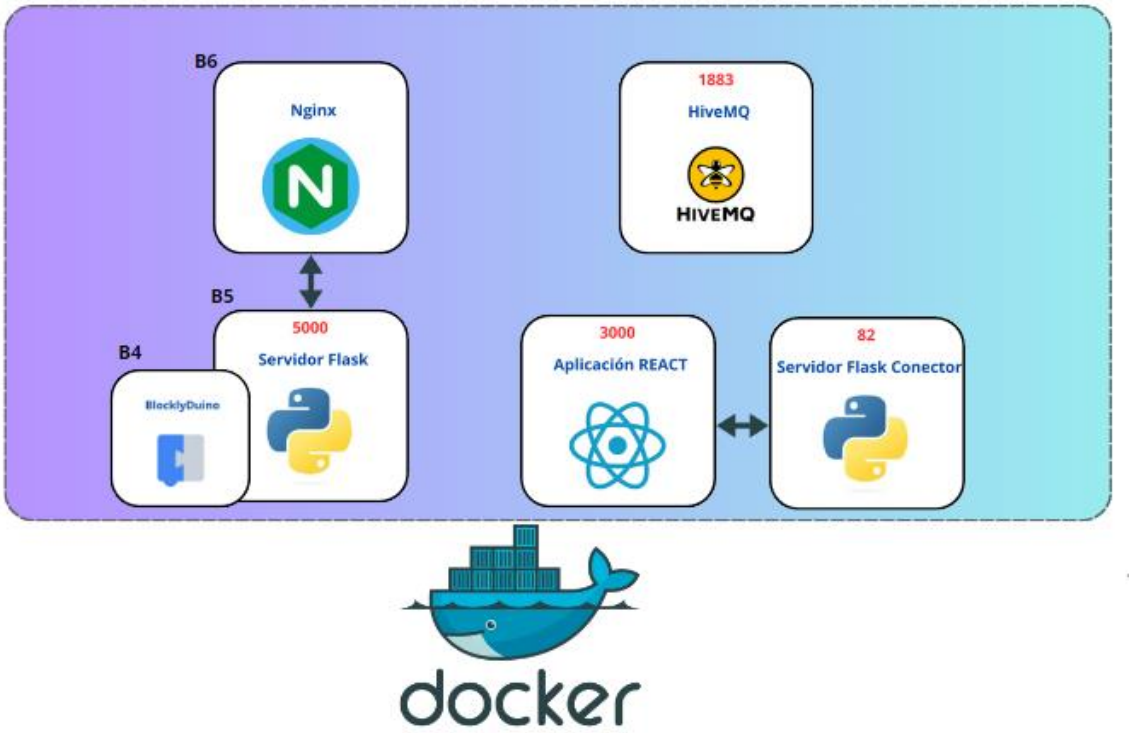


Figura 36. Esquema Despliegue contenedores

4.6.2. Configuración de los contenedores

Para poder desplegar todos los contenedores vamos a generar un único fichero Docker-compose que llame a cada uno de los Docker individuales para generar las imágenes, vamos a presentar a continuación la configuración de los contenedores, pero vamos a

exponer cada uno de los comandos con su uso. En la tabla 1 se muestran los comandos para configurar Docker-compose y en la tabla 2 los de Docker.

version	**Versión del archivo Docker-compose**
services	Lista de contenedores que se despliegan
build	Ruta de la imagen de dicho contenedor
container-name	Nombre del contenedor en Docker
restart	Reinicio del contenedor al iniciar
port	Relación de los puertos local-contenedor
networks	Configuración de la red, para la conectividad entre contenedores
volumes	Ruta para compartir archivos entre contenedor - local

Tabla 1. Comandos Docker-Compose

from	**Indica la imagen base para construir la imagen docker**
copy	Copiar los archivos del host al contenedor
run	Ejecutar comandos en el contenedor durante la construcción, instalación de paquetes.
workdir	Establecemos el directorio de trabajo
cmd	Comandos por ejecutar cuando se inicie el contenedor

Tabla 2. Comandos Docker

Para desplegar los servicios como necesitamos conectarlos todos dentro de la misma red vamos a crear una red común para todos, la cual va a llamarse telemetry, y va a tener la subred 192.168.238.0/24. No se establecerá la dirección para cada servicio puesto que usaremos los puertos de cada contenedor.

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

Figura 37. Configuración de red Docker-Compose

4.6.2.1 Contenedor Nginx

Figura 38. Configuración Nginx en Docker-Compose

Tiene dos puertos los cuales corresponden,

- **Puerto 80 (HTTP):** Este es el puerto estándar para el tráfico web no seguro (HTTP). Cuando accedes a un sitio web sin HTTPS, el tráfico pasa a través de este puerto.
- **Puerto 443 (HTTPS)**: Este puerto se utiliza para tráfico web seguro, es decir, tráfico que utiliza HTTPS. HTTPS cifra la información, protegiendo la privacidad y la integridad de los datos entre el servidor y el cliente.

Utiliza la red telemetry la cual comparten todos los contenedores, la configuración del fichero docker es muy simple puesto que ngix solo contiene tres ficheros, los cuales son:

- **Host.key**: Contiene la clave privada utilizada para establecer conexiones seguras (SSL/TLS) en Nginx. Es fundamental para la autenticación y el cifrado de datos.
- **Host.cert:** Es el certificado público que se entrega a los clientes para establecer una conexión segura, junto con la clave privada, garantiza la encriptación y autenticad de las comunicaciones.
- **Nginx.conf:** Archivo principal de configuración de Nginx que define cómo se manejan las conexiones, servidores, puertos, rutas, y parámetros globales de rendimiento y seguridad (como SSL y proxies).

La configuración de Nginx utiliza 1 proceso de trabajo y permite hasta 1024 conexiones por proceso. Activa el uso eficiente de archivos (sendfile, tcp_nopush, tcp_nodelay) y configura un timeout de 65 segundos para mantener conexiones vivas. Define dos servidores: uno escucha en el puerto 80 para HTTP con una página de error personalizada, y otro en el

puerto 443 con SSL, que actúa como proxy inverso hacia un servidor backend (blocklyduino:8080). También gestiona las solicitudes CORS añadiendo los encabezados necesarios para preflight y respuestas.

4.6.2.2 Contenedor Blockyduino

```
16  blocklyduino:
17    build: ./blocklyduino
18    container_name: blocklyduino
19    restart: always
20    ports:
21      - "5000:5000"
22    networks:
23    - telemetry
```

Figura 39. Configuración Blockyduino en Docker-compose

En el cual se establece tanto como puerto el 5000 para la maquina local como el contenedor. Se la asigna el nombre blocklyduino y para poder comunicarlo con el resto de los contenedores añadimos la red previamente creada “telemetry”.
La parte más interesante de este contenedor se encuentra en el fichero Docker dentro de la ruta /blocklyduino el cual es el siguiente:

```
blocklyduino > Dockerfile > ...
1   FROM python:3.7
2
3   COPY requirements.txt /home/servidor/requiremets.txt
4   RUN pip install -r /home/servidor/requiremets.txt
5
6   WORKDIR /home/servidor/
7   COPY . /home/servidor/.
8   CMD python /home/servidor/server.py
```

Figura 40. Configuración Blocklyduino en Docker

Aquí realizamos la instalación de una imagen Python, como se ha comentado anteriormente con el comando CMD vamos a lanzar el primer Flask utilizando Python y llamando al fichero server.py, el cual en la maquina local se encuentra en la carpeta blocklyduino, pero dentro del contenedor lo hemos copiado en la ruta /home/servidor.

Para hacer funcionar correctamente al servidor necesitamos tener instaladas todas las dependencias utilizadas en el fichero, para ellos vamos a crear un fichero “requirements” con todas estas dependencias y se deberán instalar para evitar posibles errores al ejecutar el servidor, por último, necesitamos también todo el contenido del blocklyduino por lo que se ha copiado la carpeta entera dentro de /home/servidor.
La estructura de carpetas para Blockly es la siguiente:

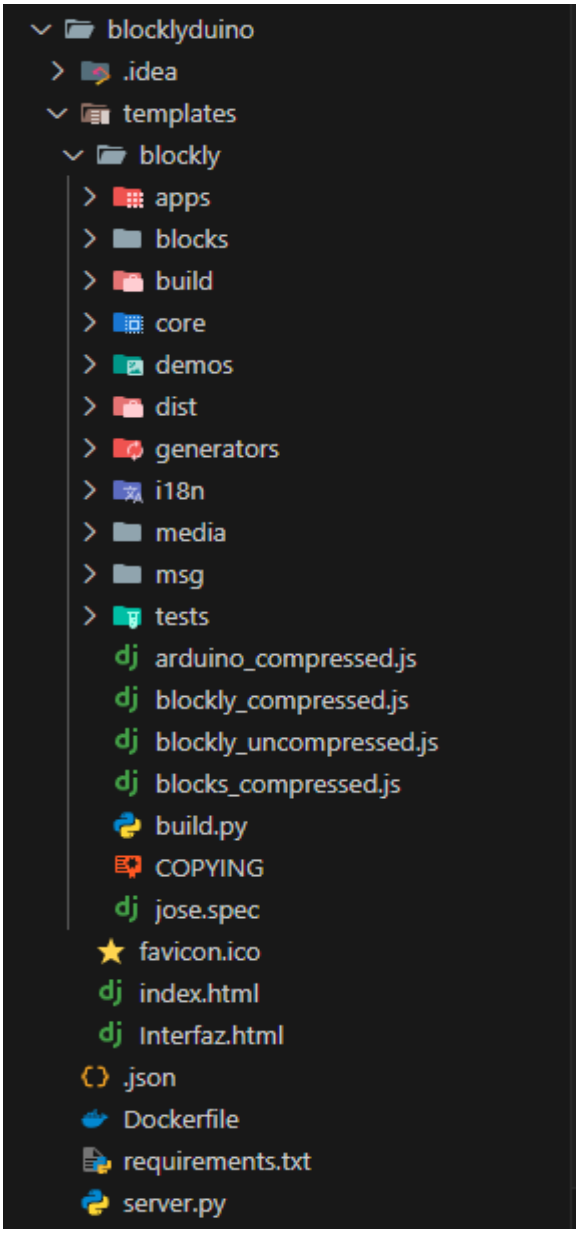


Figura 41. Estructura de carpetas Blockyduino

Donde las carpetas más importantes son las correspondientes a las rutas:

- /Blockyduino/templates/blockly/blocks – Contiene los códigos de cada uno de los bloques creados.
- /Blockyduino/templates/blockly/generators/Arduino – Contiene la funcionalidad de cada uno de los bloques definidos en blocks.
- /Blockyduino/templates/blockly/apps/Blocklyduino – Contiene el .html mediante el cual se permite al usuario interactuar con los bloques.

El fichero server.py contiene toda la información para que funcionen los endpoints mencionados en el punto 4.1.2.

4.6.2.3 Contenedor HiveMQ

```
25    hivemq:
26      image: hivemq/hivemq4
27      ports:
28        - "4500:8080"
29        - "1883:1883"
30        - "8000:8000"
31      networks:
32        - telemetry
33      container_name: Hive_MQTT2
34
```

Figura 42. Configuración Hivemq en Docker-compose

Simplemente se añade el servicio Hivemq para evitar al usuario instalar este servicio en la maquina local y se puede conectar al bróker desde Docker, hemos cambiamos el puerto 8080 el cual nos muestra la interfaz gráfica de HiveMQ al puerto local 4500 para evitar posibles conflictos con otro servicio web que tenga desplegado el usuario, al igual que los demás contenedores añadimos la misma configuración de red, este contenedor es bastante simple.

4.6.2.4. Contenedor Flask2

```
35  flask_peticion:
36    build: ./flask2
37    container_name: Flask_peticion
38    restart: always
39    ports:
40      - "82:80"
41    networks:
42    - telemetry
43
```

Figura 43. Configuración flask2 en Docker-compose

Este contenedor tiene el servidor Flask que interactúa únicamente con el frontal React desplegado a continuación, como los demás contenedores añadimos la red para que puedan interactuar entre los diferentes contenedores, y como puertos para evitar conflictos con otros servicios cambiamos el puerto 80 contenedor por el 82 en la maquina local. Para el fichero Docker tendremos la siguiente configuración.

```
1   #Imagen que queremos utilizar en esta caso python
2   FROM python:3.7
3
4   #Copiamos las dependencias que queremos utilizar en el contenedor
5   #Utiliza un formato COPY <Ruta local> <Ruta en el contenedor>
6   COPY requirements.txt /home/servidor/requiremets.txt
7
8   #Instalamos las dependencias dentro del contenedor
9   RUN pip install -r /home/servidor/requiremets.txt
10
11  #Establecemos el directorio de trabajo
12  WORKDIR /home/servidor/
13
14  #Copiamos los archivos dentro del contenedor en este caso todos los
15  #de la carpeta en los cuales se encuentra el fichero Docker
16  COPY . /home/servidor/.
17
18  #Comando que utilizamos al desplegar el contenedor
19  CMD python /home/servidor/server.py
20
21
```

Figura 44. Configuración flask2 en Docker

Bastante similar al otro contenedor en el cual desplegamos otro servicio Flask, se difieren puesto que este contenedor solo contiene el servidor Flask mientras que el otro además contiene los ficheros de configuración necesarios para desplegar BlockyDuino.

4.6.2.5 Contenedor black

```yaml
react_front:
  build:
    context: ./black
    dockerfile: Dockerfile
  ports:
    - "3000:3000"
  networks:
    - telemetry
  volumes:
    - ./black:/black
  command: npm run start
  container_name: react_frontal
```

Figura 45. Configuración frontal en Docker-compose

Contiene todos los ficheros necesarios para poder utilizar la parte frontal del proyecto, detallada en el punto 4.4, se puede apreciar la utilización del puerto 3000 propio de React además de compartir red con el resto de los contenedores, la configuración es muy simple puesto que se copiarán todos los ficheros generados al crear un proyecto React y de iniciará con el comando “npm start”, a continuación, se puede observar la configuración del Dock perteneciente a este bloque.

serial.go M | hub.go M | index.html | Dockerfile U

black > Dockerfile > ...

```dockerfile
# Use an official Node.js runtime as a parent image
FROM node:20.9
# Set the working directory to /app
WORKDIR /app
# Copy the package.json and package-lock.json files to the container
COPY package*.json ./
# Install dependencies
RUN npm install
# Copy the rest of the application code to the container
COPY . .
# Build the application
RUN npm run build
# Expose port 3000 for the application
EXPOSE 3000
# Start the application
CMD ["npm", "start"]
```

Figura 46. Configuración frontal en Docker

5. Resultados y discusiones

En este apartado vamos a tratar diferentes actividades prácticas y sus soluciones, utilizando las herramientas y bloques descritos en los puntos anteriores, las actividades se pueden complementar entre ellas. Dentro del Anexo II se encuentra detalladamente como utilizar y configurar los bloques principales

5.1 Actividad 1. Obtener respuesta Inteligencia Artificial

El objetivo de esta actividad consiste en la realización de una pregunta con nuestro Arduino y la obtención de una respuesta proporcionada por una Inteligencia Artificial.

5.1.1. Montaje de la actividad

El montaje se puede observar en la figura 47, consta de cuatro bloques, el bloque inicial Project, selecciona el tipo de servicio conectado, en nuestro caso seleccionado esp32 porque vamos a trabajar con este tipo de dispositivo, el siguiente bloque contiene la configuración de la wifi local, en este caso y el resto de módulos tendremos las mismas credenciales puesto que se realizan las actividades en la misma red, el tercer bloque contiene la pregunta a realizar y el token de autenticación necesario para poder usar la API de ChatGPT, y para finalizar añadimos un temporizador para que no realice la misma pregunta de manera recurrente.

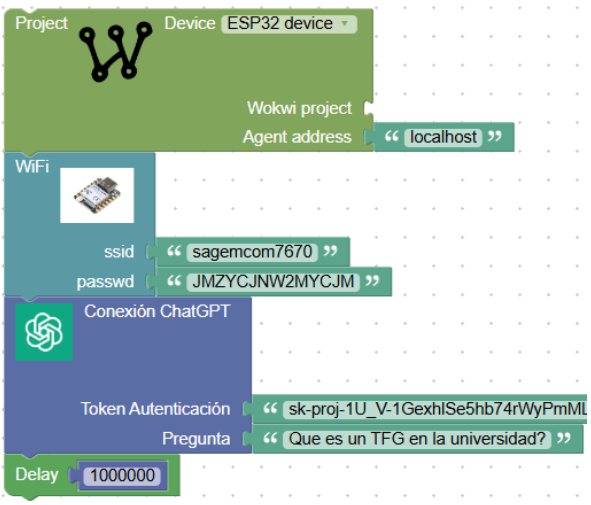


Figura 47. Consulta Inteligencia Artificial

5.1.2. Resultados

Para poder utilizar la carga automática se recomienda consultar en el Anexo I el punto I.III configuración del agente.
Simplemente una vez conectado nuestro dispositivo Arduino al ordenador se deberá pulsar el botón “Cargar”, el propio agente capturará el puerto del dispositivo y cargará el código, mostrando en la pestaña “Registro” el siguiente código:

Figura 48. Registro en actividad 1

Para poder observar la respuesta proporcionada por la inteligencia artificial solamente se deberá observar la pestaña “Monitor”, es este caso la respuesta ha sido la siguiente:

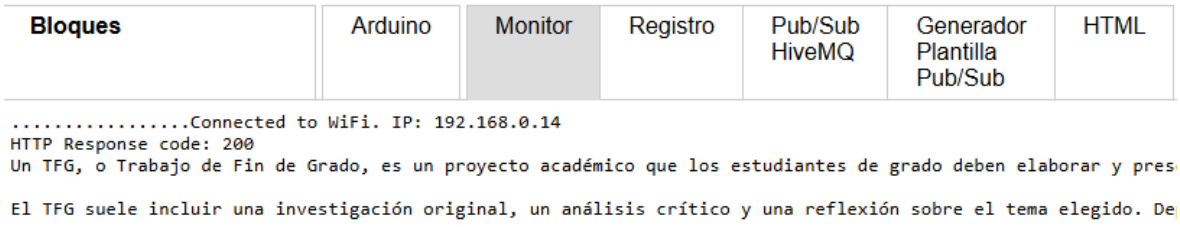


Figura 49. Respuesta actividad 1

5.2. Actividad 2. Carga de datos en Base de datos y visualización

El objetivo de esta actividad se basa en la carga de ciertos valores en una base de datos (dentro del ámbito local, pero no en la dirección local para disminuir la complejidad de contratar un servicio de base de datos SQL Cloud) y posteriormente su visualización gráfica en un servicio React.

5.2.1. Montaje de la actividad

El montaje se puede observar en la figura 50, esta actividad consta de tres bloques, el bloque inicial Project, selecciona el tipo de servicio conectado, en nuestro caso seleccionado esp32 porque vamos a trabajar con este tipo de dispositivo, el siguiente bloque contiene la configuración de la wifi local, el tercer bloque contiene la información necesaria explicada previamente en el punto 4.1.2

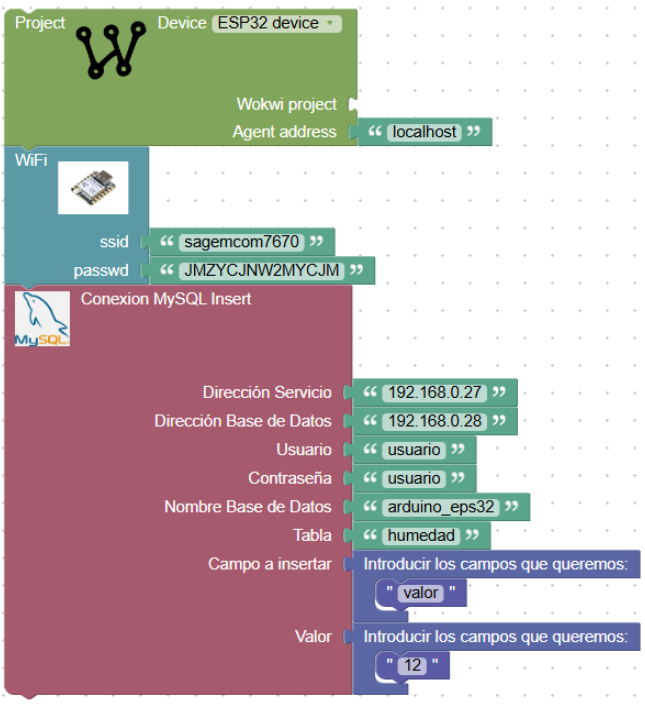


Figura 50. Carga de datos

5.2.2. Resultados

Como en la actividad anterior para la carga del código se realizará mediante el botón cargar, en este caso la información proporcionada por el serial de Arduino es la siguiente

Figura 51. Respuesta actividad 2

Para la visualización de estos datos se realizará a través de la dirección en la cual se encuentra desplegado el Docker, en este caso localhost, y el puesto 300, para la consulta se usarán los mismos datos que para el bloque anterior, y dentro de las tablas devueltas se consultará la tabla en la cual se han introducido los datos (mostrando claramente una gráfica constante de los tres valores introducidos previamente).

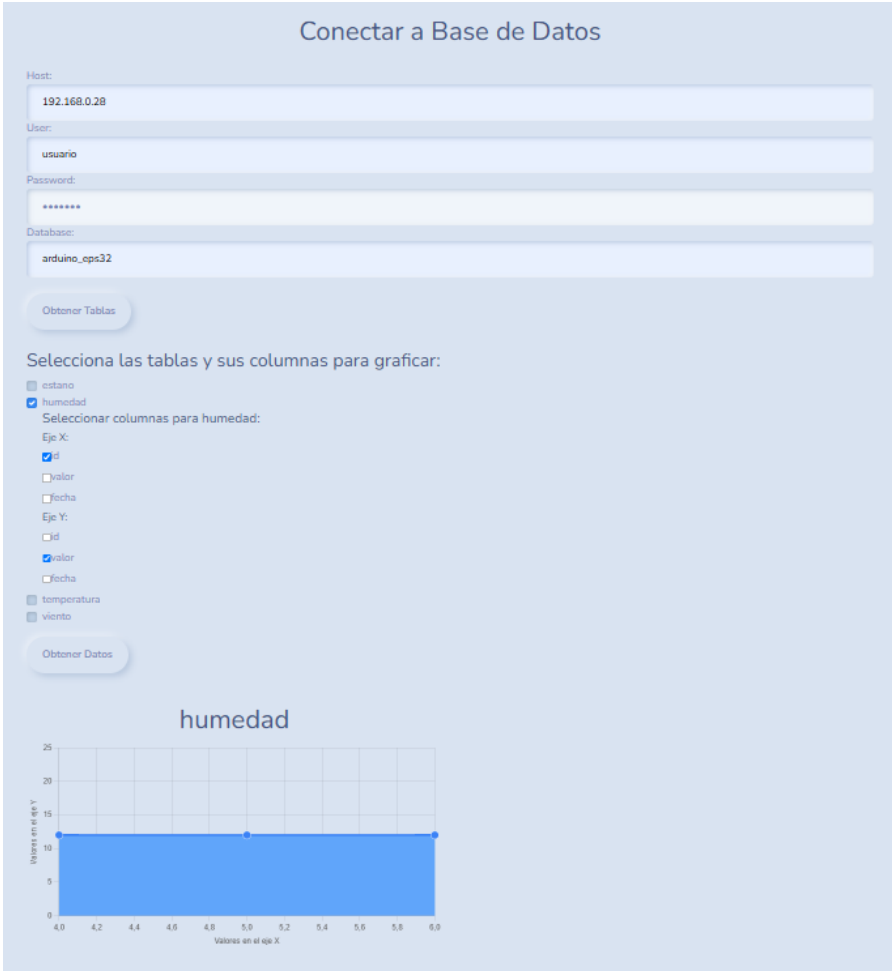


Figura 52. Visualización gráfica

5.3. Actividad 3. Visualizar datos de un publicador

El objetivo de esta actividad es visualizar en tiempo real los datos de humedad y temperatura proporcionados por una placa Arduino ubicada en una habitación. Aunque el dispositivo registra también datos de luz, solo se requiere la visualización de la humedad y la temperatura. El dispositivo Arduino publica dentro del del tema “habitacion1/” seguido del dato que manda.

5.3.1. Montaje de la actividad

Esta actividad no requiere montaje previo puesto que vamos a utilizar el bloque explicado en el punto 4.3. Pero si queremos simular esta actividad y no tenemos un dispositivo funcionando, se recomienda el uso de un publicador (MQTT Explorer) como se muestra a continuación.

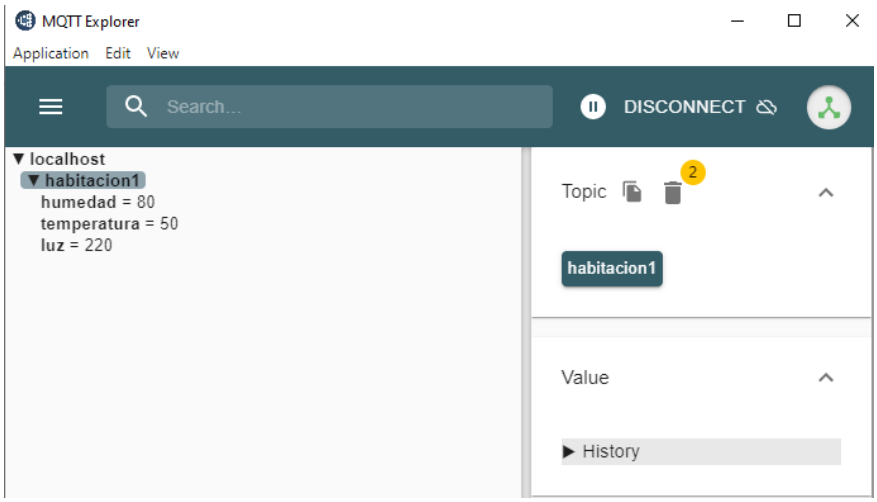


Figura 53. Simulación publicador MQTT

Para realizar esta actividad, es necesario acceder a la pestaña 'Pub/Sub HiveMQ'. En esta sección, solo debemos conectarnos al bróker MQTT donde se están publicando los mensajes de nuestro Arduino ubicado en la habitación. Como el bróker se encuentra en el mismo dispositivo, usaremos localhost como dirección. Luego, nos suscribiremos a los temas que queremos visualizar, en este caso, humedad y temperatura, tal como se muestra en la siguiente imagen.

Figura 54. Configuración Subscriptor

5.3.2. Resultado

El resultado de la actividad nos muestra en tiempo real los datos proporcionados por el publicador, en este caso en el momento de la captura se mandaron los mismos que se puede visualizar en la Figura 53, dando como resultado la siguiente información.

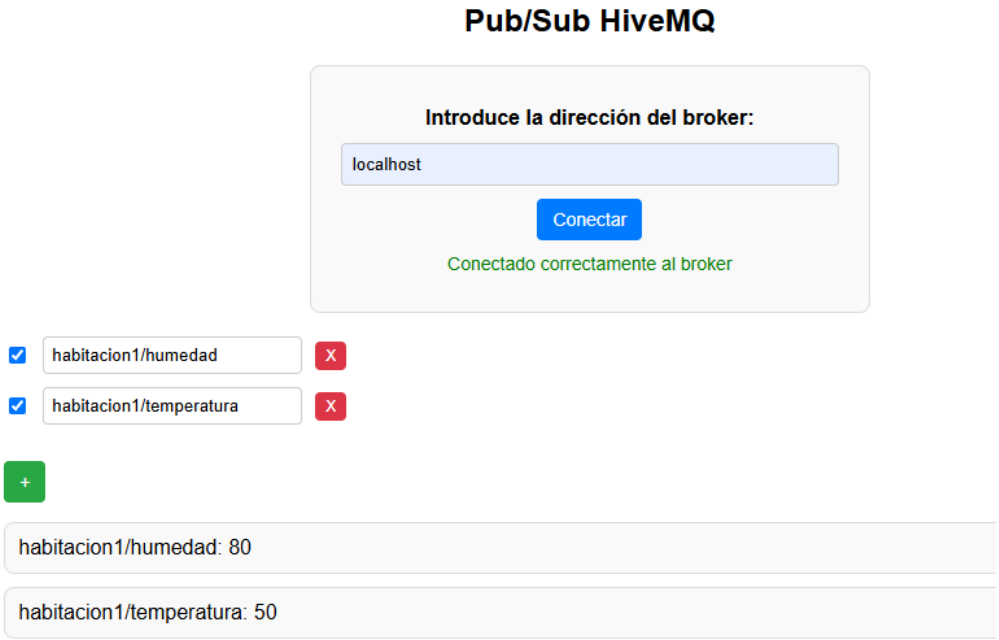


Figura 55. Resultado actividad 3

5.4. Actividad 4. Carga de datos en Firebase

El objetivo de dicha actividad consiste en el uso de Firebase, vamos a realizar la conexión, escritura y lectura de un determinado valor proporcionado por un determinado sensor, la lectura se realizará para comprobar si se realizó correctamente la inserción. La configuración y paso a paso del uso del siguiente bloque se encuentra detallado dentro del Anexo II, punto II.III.

5.4.1. Montaje de la actividad

El montaje se puede observar en la figura 56, consta de cinco bloques, el bloque inicial Project, selecciona el tipo de servicio conectado, en nuestro caso seleccionado esp32 porque vamos a trabajar con este tipo de dispositivo, el siguiente bloque contiene la configuración de la wifi local, el tercer bloque contiene la información necesaria para el establecimiento de la conexión, detallado previamente en el punto 4.1.4.1, el cuarto bloque se va a utilizar para insertar los datos del sensor, usaremos como tabla el dato sensor y el valor 1000, el quinto bloque solamente va a realizar la lectura del último dato insertado.

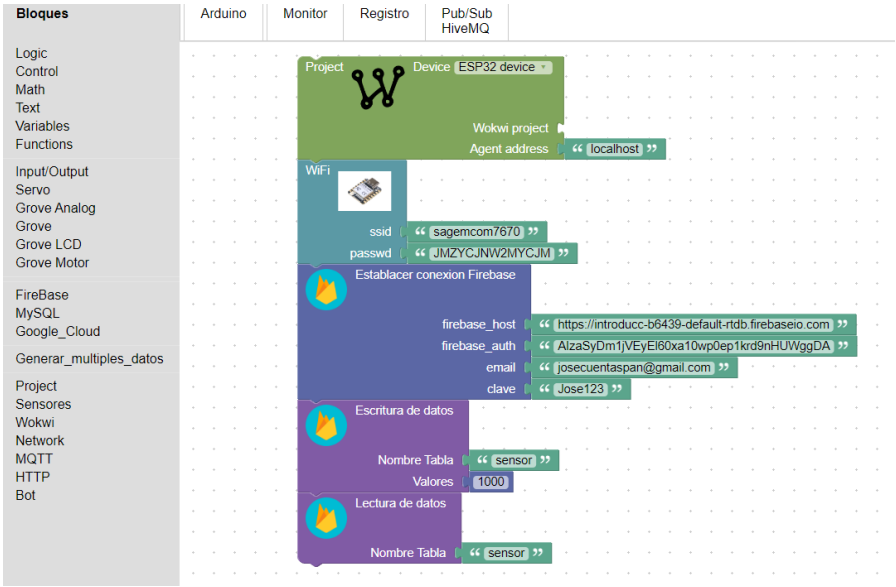


Figura 56. Carga Firebase

5.2.1. Resultado

Al cargar los bloques mediante el botón, podemos observar mediante la pestaña “Monitor” la siguiente información (Figura 57), demostrando que se insertó y se recibió el mismo

valor, y a su vez dentro de Firebase también podemos ver el ultimo valor cargado (Figura 58).

```
C◆digo de error HTTP: -11
........Connected to WiFi. IP: 192.168.0.12
-----------------------------------
Token info: type = id token (GITKit token), status = on request
Token info: type = id token (GITKit token), status = ready
Data sent successfully: 1000.00
Data received: 1000.00
```

Figura 57. Monitor actividad 4

Figura 58. Datos Firebase

5.5. Otros ejemplos

Dentro del Anexo II se encuentran más ejemplos de uso de los diferentes módulos:

apartado II.II encontramos un ejemplo y configuración avanzado de SQL,
apartado II.III un ejemplo y configuración de Firebase,
apartado II.IV un ejemplo y configuración de Google Cloud para el método Pub/Sub
apartado II.V que contiene una actividad de inserción SQL Cloud a través de evento Pub en determinado topic.

6. Conclusiones y propuesta de mejora

A lo largo de este trabajo, se ha demostrado la flexibilidad y escalabilidad que ofrece BlocklyDuino, así como la facilidad para desarrollar nuevos bloques en distintos campos relacionados con Arduino.

La capacidad de conectarse a diferentes bases de datos en la nube y la posibilidad de publicar mensajes en un tema determinado brindan una gran flexibilidad al usuario, permitiéndole monitorizar datos extraídos de sensores sin requerir conocimientos avanzados de programación.

Además, la combinación de servicios disponibles, como la suscripción a MQTT, proporciona una gran versatilidad, permitiendo que los usuarios experimenten y comprendan el comportamiento del sistema de forma práctica. Esta herramienta resulta ser muy potente tanto para el aprendizaje como para el desarrollo de proyectos, facilitando la creación de soluciones sin necesidad de poseer un conocimiento técnico profundo.

Otro aspecto destacado es la visualización de datos, que se apoya en herramientas como la interfaz desarrollada en React, la cual permite consultar datos tanto de bases locales como en la nube. Dicha interfaz ofrece modularidad para la representación de los datos, y también se dispone de múltiples herramientas, como suscriptores y visualización de temas suscritos.

Finalmente, la integración de este ecosistema en un contenedor aporta un valor añadido al permitir su despliegue en cualquier ordenador, lo que incrementa la accesibilidad y portabilidad de la solución.

Después de la realización de este TFG, se plantean las siguientes áreas de mejora o implementación:

- Implementación de nuevos bloques basados en predicción de datos, inteligencia artificial o incluso otros servicios Cloud.

- Mejora del servicio React, añadiendo diferentes tipos de gráficas, como graficas de barras, graficas circulares ...

- Creación de plantillas predefinidas que carguen proyectos completos. Por ejemplo, se podría implementar una categoría de bloques llamada Proyectos, en la que, al seleccionar uno, se cargue toda la estructura necesaria. Un ejemplo podría ser un proyecto de riego automático, que configure de manera automática los datos y bloques necesarios para el funcionamiento, incluyendo motores y conexión con bases de datos.

- Crear conexiones entre los diferentes sensores y los bloques Cloud, por ejemplo, insertar en FireBase los datos capturados por un sensor real de temperatura.

7. Estudio Económico

Para el estudio económico vamos a realizar dos presupuestos, un presupuesto de prueba basado en la utilización de las versiones de prueba y otro presupuesto añadiendo el coste que tendrá la utilización de este proyecto en un tiempo prolongado.

7.1. Versión trial

Unidades	Descripción	Precio
1	DELL 3020 Sobremesa i5–4430 3,00 GHz, 16GB RAM	134
1	Arduino Eps32	8
1	Kit de sensores Arduino	10
1	Cuenta Google Cloud	0
1	Cuenta FireBase	0

Tabla 3. Presupuesto versión trial

Teniendo en cuenta que cualquier persona puede tener un ordenador con los requisitos necesarios para poder desplegar Docker, tanto FireBase como GoogleCloud presentan cuentas de prueba.

7.2. Uso Continuado

Unidades	Descripción	Precio
1	DELL 3020 Sobremesa i5–4430 3,00 GHz, 16GB RAM	134
1	Arduino Eps32	8
1	Kit de sensores Arduino	10
1	Cuenta Google Cloud	0
x	Servicio SQLCloud, Pub/Sub, disparadores ...	12
1	Cuenta FireBase	0

Tabla 4. Presupuesto uso continuo

7.3. Recursos Humanos

Unidades	Descripción	Precio	Total
1	230 horas de trabajo	22€/hora	5060

Tabla 5. Resultado recursos humanos.

7.3. Resultado estudio económico

Tipo	**Subtotal**	**Impuesto I.V.A 21%**	**Total**
Prueba	5212	1094.52	6306.52
Continuo	5224	1097.04	6321.04

Tabla 6. Resultado estudio económico.

8. Referencias

[1] Contenedores de Docker | ¿Qué es Docker? | AWS. (s. f.). Amazon Web Services, Inc. https://aws.amazon.com/es/docker/

[2] De Imagina, E. (2024, 14 agosto). Qué es Docker Compose y Cómo Usarlo | Tutorial Completo. Imagina Formación. https://imaginaformacion.com/tutoriales/que-es-docker-compose

[3] Mora, S. L. (2022, 4 octubre). Firebase: qué es, para qué sirve, funcionalidades y ventajas. DIGITAL55. https://digital55.com/blog/que-es-firebase-funcionalidades-ventajas-conclusiones/

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

[5] ¿Qué es Pub/Sub? (s. f.). Google Cloud. https://cloud.google.com/pubsub/docs/overview?hl=es-419

[6] Blockly para desarrolladores disponible en la dirección https://developers.google.com/blockly?hl=es-419

[7] Flask para desarrolladores, disponible en la dirección https://flask.palletsprojects.com/en/stable/installation/#install-flask

[8] B, G., & B, G. (2024, 25 junio). ¿Qué es NGINX y cómo funciona? Tutoriales Hostinger. https://www.hostinger.es/tutoriales/que-es-nginx

[9] Documentación y ejemplos sobre React en la siguiente dirección: https://es.react.dev/

[10] Información y descargar plataformIO https://platformio.org/

[11] Pagina con información sobre el protocolo mqtt https://mqtt.org/

[12] Python para programadores https://es.python.org/aprende-python/

[13] JavaScript para desarrolladores https://developer.mozilla.org/es/docs/Web/JavaScript

[14] HTTP | MDN. (2023, 24 julio). MDN Web Docs. https://developer.mozilla.org/es/docs/Web/HTTP

[15] Arsys. (2024, 6 julio). La importancia del protocolo HTTPS. https://www.arsys.es/blog/importancia-protocolohttps

[16] Access SQL: conceptos básicos, vocabulario y sintaxis - Soporte técnico de Microsoft. (s. f.). https://support.microsoft.com/es-es/topic/access-sql-conceptos-b%C3%A1sicos-vocabulario-y-sintaxis-444d0303-cde1-424e-9a74-e8dc3e460671

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

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

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

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

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

[22] Desarrollo de prácticas con entornos virtuales en el ámbito de las telecomunicaciones. https://crea.ujaen.es/handle/10953.1/21103

[23] García-Galán, S., Muñoz-Expósito, J. E., & De Jaén Ingeniería de Telecomunicación, U. (2023, 19 diciembre). Desarrollo de prácticas con entornos virtuales en el ámbito de las telecomunicaciones. https://crea.ujaen.es/handle/10953.1/21103

9. ANEXOS

9.1 ANEXO I: PREPARACIÓN Y REQUISITOS PREVIOS

I. Preparación del entorno

En este anexo se ofrecerán instrucciones para la instalación del proyecto, y la configuración necesaria para poder utilizar todos los módulos, detallados en el Anexo II.

I.I. Requisitos previos

Para poder utilizar la totalidad del proyecto y poder seguir con el resto de la guía se deberá instalar o tener instalado los siguientes programas:

- **Docker** : https://www.docker.com/
- **PlataformIO** : https://platformio.org/
- **VisualStudio**: https://code.visualstudio.com/
- **Navegador web:** https://www.google.com/intl/es_es/chrome/
- **Python:** https://www.python.org/
- **Cuenta en GoogleCloud**: https://cloud.google.com/?hl=es
- **Cuenta en Firebase**: https://firebase.google.com/?hl=es-419

Guía de instalación PlataformIO:
https://programarfacil.com/blog/arduino-blog/platformio/

I.II. Inicialización del proyecto

El primer paso se basa en la clonación del proyecto en el equipo local, esto lo vamos a realizar desde Visual Studio Code, con la terminal con el comando:

Git clone https://github.com/jose020997/Blockyduino.git

```
PS C:\Users\ja.barrios\OneDrive - TeamSystem S.p.A\Escritorio> git clone https://github.com/jose020997/Blockyduino.git
Cloning into 'Blockyduino'...
remote: Enumerating objects: 57, done.
remote: Counting objects: 100% (57/57), done.
remote: Compressing objects: 100% (53/53), done.
remote: Total 57 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (57/57), 154.51 KiB | 4.41 MiB/s, done.
PS C:\Users\ja.barrios\OneDrive - TeamSystem S.p.A\Escritorio>
```

Cuando ya tengamos el proyecto descargado en local, simplemente tendremos que cargar la estructura de contenedores, la cual se ubica en el dichero docker-compose.yml para realizar esto solamente deberemos tener abierto Docker y ejecutar el siguiente comando:

docker-compose up --build -d

Con conseguirá desplegar la estructura de contenedores, para poder conectarnos directamente tendremos que abrir nuestro navegador y acceder a la siguiente dirección:

https://localhost

Si queremos acceder desde otro dispositivo solamente tendremos que cambiar localhost por la dirección IP en la cual tenemos desplegado el Docker.

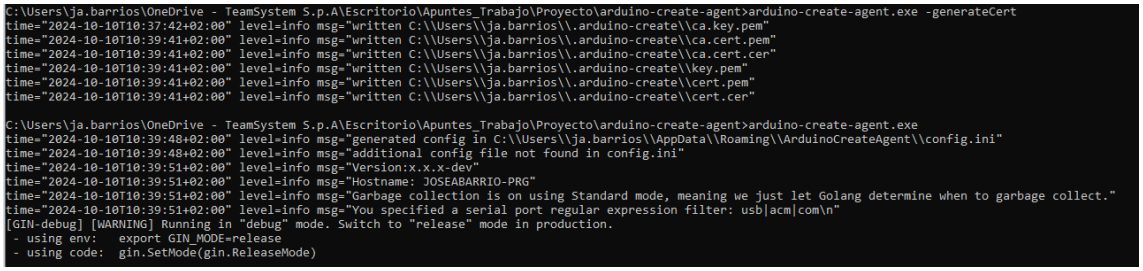
I.III. Despliegue del agente Arduino

Para poder cargar automáticamente el código en la placa Arduino, deberemos tener la placa conectada a nuestro ordenador, en cada ordenador vamos a desplegar el agente de Arduino que utilizaremos para conectarnos con PlataformIO, se hará accediendo a la raíz del proyecto, dentro de la carpeta “Arduino-create-agent”, accediendo a través de CMD solamente tendremos que ejecutar los siguientes comandos:

arduino-create-agent.exe -generateCert

arduino-create-agent.exe

Con el primero de ellos cargaremos los certificados necesarios y con el segundo dejaremos ejecutando el agente (Recomendación si tarda mucho en cargar los certificados, pulsar [Enter] ya que muchas veces se puede quedar colapsado).



A continuación, para finalizar nos aparecerá un mensaje de error indicando unknown certificate indicando que no tenemos un certificado conocido, para solucionar esto tendremos que indicar que un despliegue seguro, accediendo a la siguiente dirección y aceptando que es un sitio seguro.

https://localhost:8992/

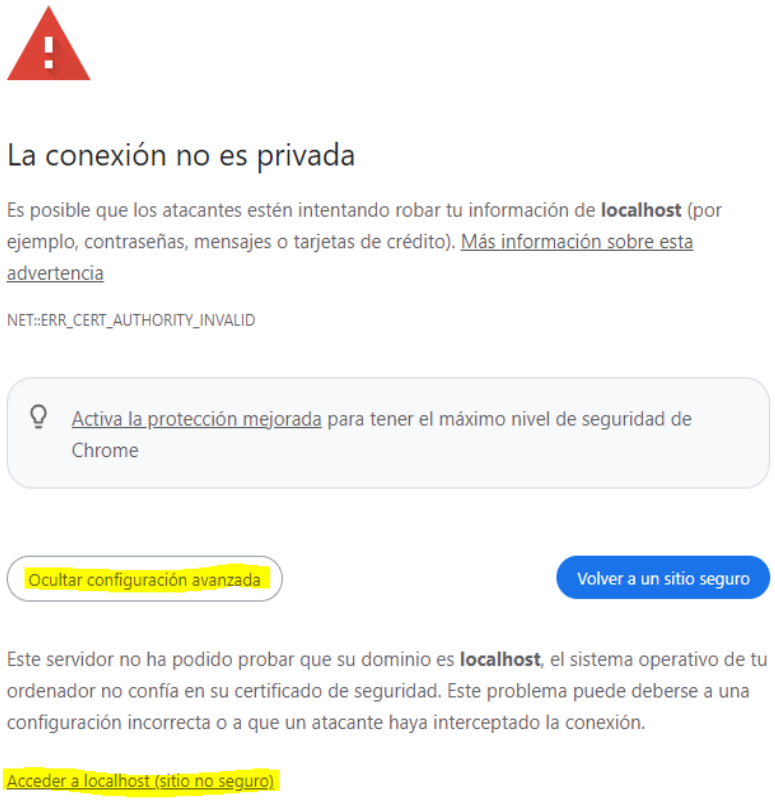


Figura 59. Anexo I, aceptación certificado SSL

Una vez finalizado tenemos desplegado nuestro agente, el cual usará Blockyduino para cargar los códigos generados por los bloques.

Podemos ver la información de nuestra placa Arduino con el comando “list” además también aparecerán el resto de los comandos disponibles dentro del array commands

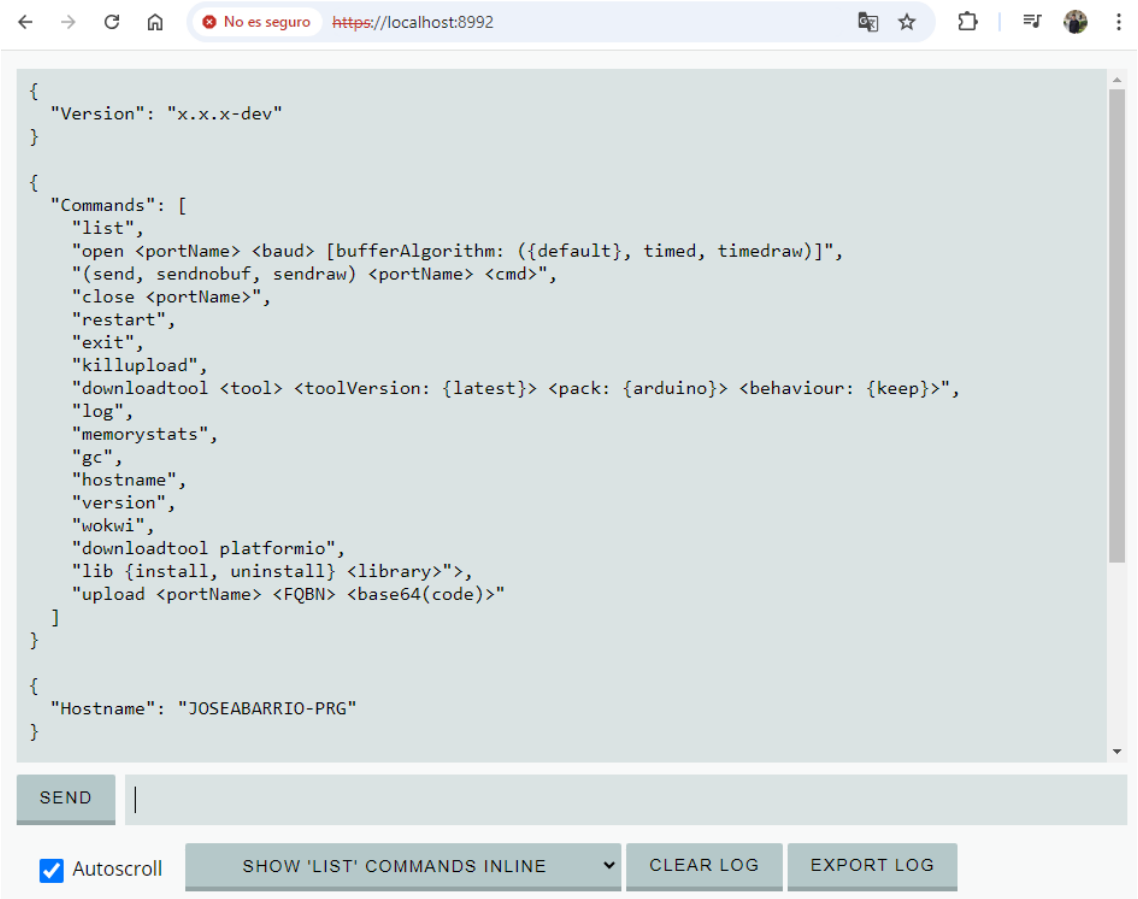


Figura 60. Anexo I, Interfaz agente

I.IV. Carga de las librerías

Como último paso para poder utilizar los bloques GoogleCloud y Firebase necesitaremos tener ciertas librerías en nuestro PlataformIO, esto lo podemos hacer de diferentes formas, la principal será instalar pio para poder utilizar los comandos para pio, lo haremos de la siguiente forma:

pip install platformio
Una vez tengamos esto, deberá instalar ambas librerías con los siguientes comandos:

pio lib --global install FirebaseESP32

pio lib --global install ESP Signer

Sino también podremos instalar estas librerías de forma local accediendo a la interfaz de Visual Code:

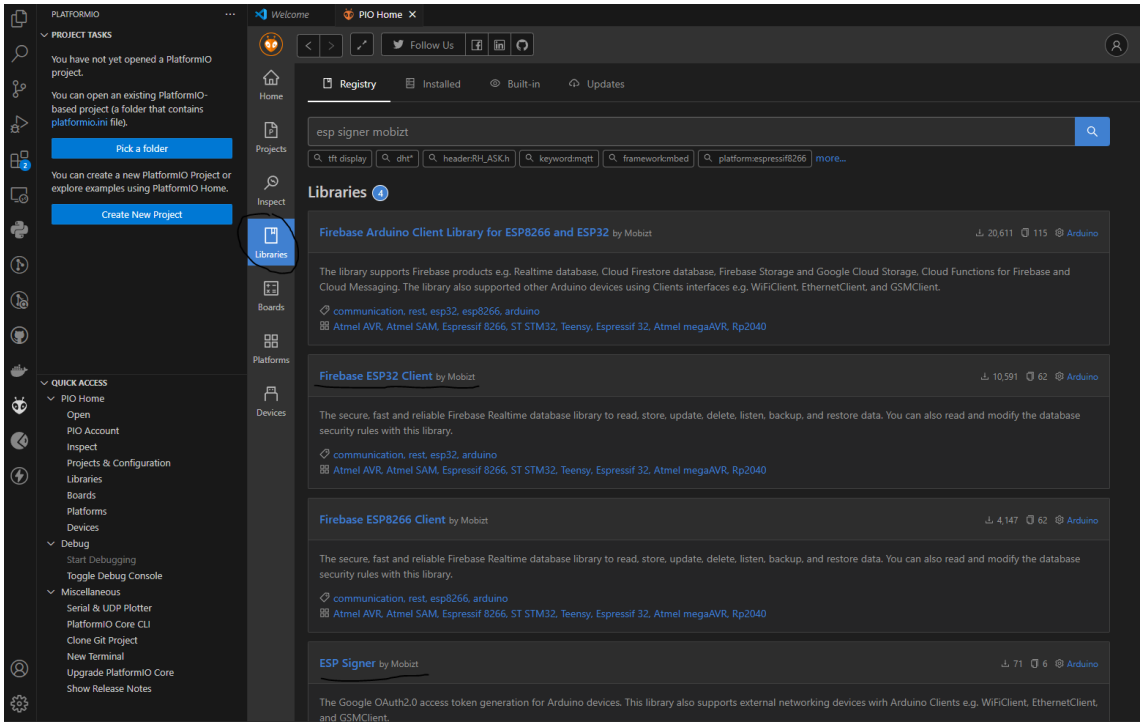


Figura 61. Anexo I, Instalación librerías

9.2 ANEXO II: USO MODULOS CLOUD

II.I. Crear un bloque en BlocklyDuino

Una de las ventajas de BlocklyDuino es su modularidad, lo que permite a cualquier persona desarrollar un bloque con las funcionalidades que desee. El objetivo de este anexo es explicar cómo generar un bloque de la forma más sencilla posible.

El primer paso es definir la estructura de nuestro bloque, lo cual lo podemos definir en la siguiente página:

https://blockly-demo.appspot.com/static/demos/blockfactory/index.html

En la parte izquierda, podemos observar cuatro categorías de bloques, de las cuales mencionaremos las más importantes:

- Input: Dentro de esta categoría tenemos 3 bloques importantes
    - Value Input: El cual nos permite conectar otros bloques, se puede añadir el tipo de dato que acepta el bloque, un texto, imagen …
    - Statement input: se utiliza para establecer múltiples opciones en el bloque
    - Dummy input: utilizado para escribir texto, principalmente para dar información al usuario.

Los demás campos son bastante descriptivos. A continuación, se adjunta una imagen de un bloque de ejemplo.

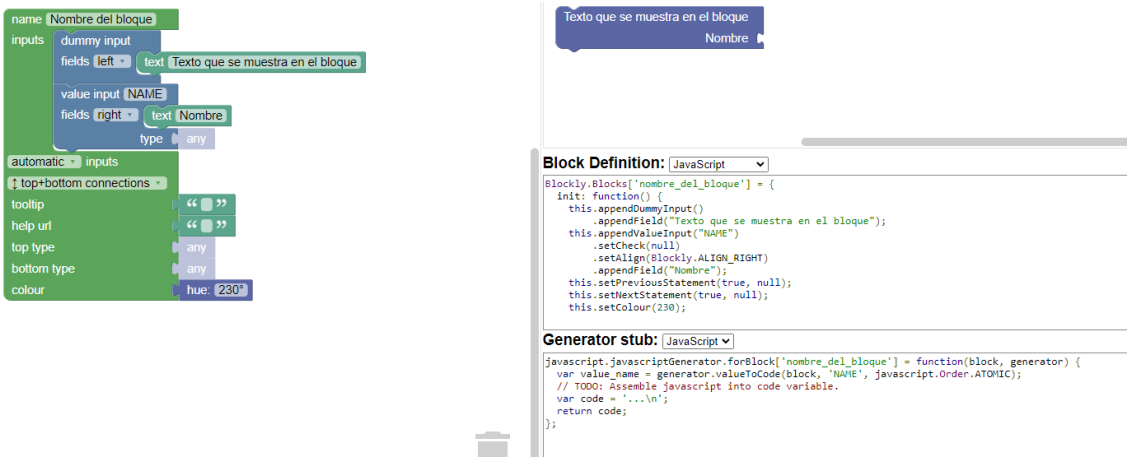


Figura 62.Anexo II, Ejemplo creación bloque

En la parte izquierda de la herramienta, definimos lo que queremos que contenga nuestro bloque. Es importante asignar un nombre y especificar qué tipos de conexiones aceptará (permitiendo la conexión de otros bloques arriba y abajo).

En la parte superior derecha, podemos visualizar un ejemplo del bloque que estamos creando, y en la sección "Block Definition" se genera el código necesario para el siguiente paso.

En la ruta /Blocklyduino/templates/blockly/blocks debemos crear un nuevo archivo y pegar el código generado en la sección "Block Definition".

Otro archivo que debemos generar se encuentra en la ruta /Blocklyduino/templates/blockly/generators/Arduino. Este archivo debe tener el mismo nombre que el anterior, y en él definiremos la funcionalidad de nuestro bloque.

Una vez que tengamos estos archivos, debemos dirigirnos a la ruta /Blocklyduino/templates/blockly/apps/Blocklyduino y editar el archivo index.html, que contiene toda la información de Blockly. Antes de continuar, importamos los dos archivos generados en el paso anterior, de la siguiente forma:

<script type="text/javascript" src="../../blocks/wokwi.js"></script>

```
<!-- Firebase-->
<category name="FireBase">
  <block type="firebase">
    <value name="host">
      <block type="text">
        <field name="TEXT">https://tfgjose-f973e-default-rtdb.firebaseio.com</field>
      </block>
    </value>
    <value name="auth">
      <block type="text">
        <field name="TEXT">AIzaSyD7_OIv8iLY-U8htFIr_8qD6-dgD_6hO30</field>
      </block>
    </value>
    <value name="email">
      <block type="text">
        <field name="TEXT">josecuentaspan@gmail.com</field>
      </block>
    </value>
    <value name="pass">
      <block type="text">
        <field name="TEXT">Jose123</field>
      </block>
    </value>
  </block>
```

Figura 63. Anexo II, Ejemplo bloque en el fichero índex

A continuación, tenemos la posibilidad de crear una categoría nueva mediante: <category name=” Nombre que queramos”>

Luego añadimos el bloque que hemos importado anteriormente poniendo el nombre del bloque, y otra de las opciones que tenemos es añadir bloques de unión al bloque principal con ciertas variables, por ejemplo, podemos añadir un bloque de texto en el apartado host con un determinado texto, a continuación, se ve como queda generado el bloque anterior.

Figura 64. Anexo II, ejemplo bloque visual final

II.II. Configuración modulo MySQL

Previamente al uso de cualquier modulo se deberá tener instalados los requisitos detallados en el Anexo I.

II.II.I. Instalación Base de datos SQL

Para poder utilizar el módulo primero vamos a configurar un entorno de base de datos, en este ejemplo vamos a instalar Xampp como ejemplo sencillo a través del siguiente enlace: https://www.apachefriends.org/es/download.html

Elegimos la versión más actualizada, y el sistema operativo que usemos, en nuestro caso Windows.
Ejecutamos el fichero .exe y vamos completando todos los pasos hasta llegar al siguiente:

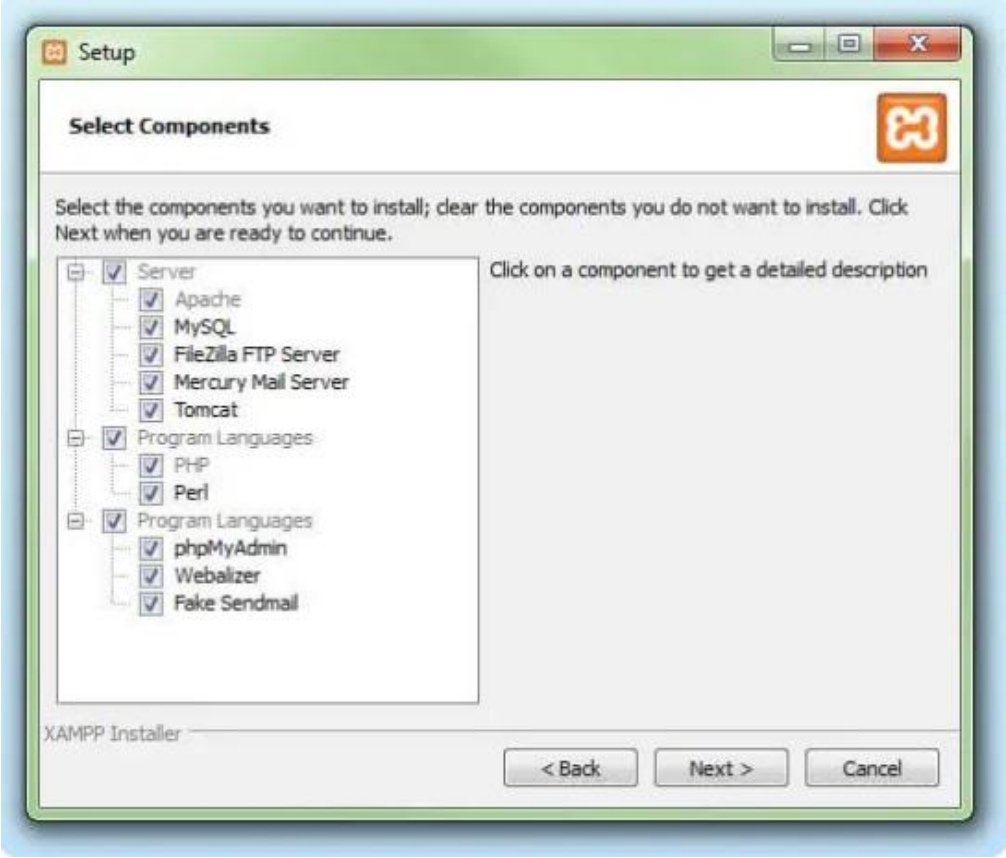


Figura 65. Anexo II, instalación SQL

Donde solamente es necesario marcar MySQL, PHP y phpMyAdmin.

Una vez completados todos los pasos aparecerá un panel similar al siguiente:

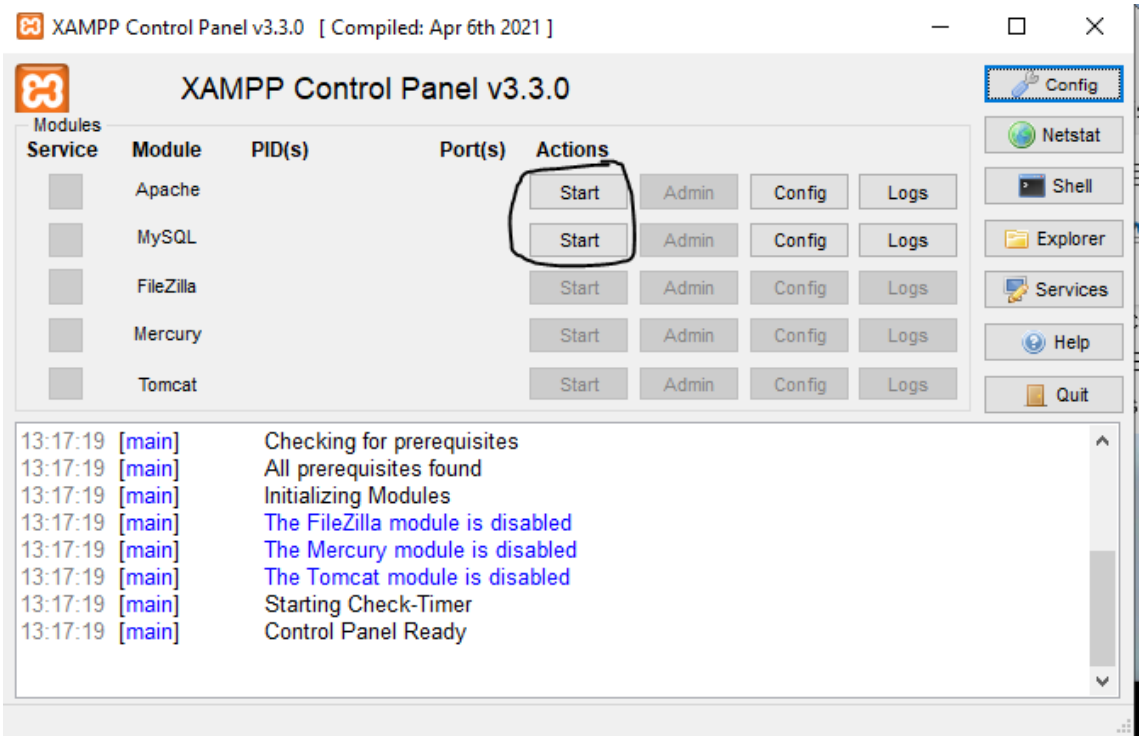


Figura 66. Anexo II, Interfaz xampp

Marcamos las dos opciones de “Start” para iniciar tanto el servicio Apache que es un servicio Web el cual nos permite acceder al panel de configuración.

Ahora tenemos que clickear en el botón “Admin” del módulo MySQL.

II.II.II Para generar una cuenta de usuario nueva

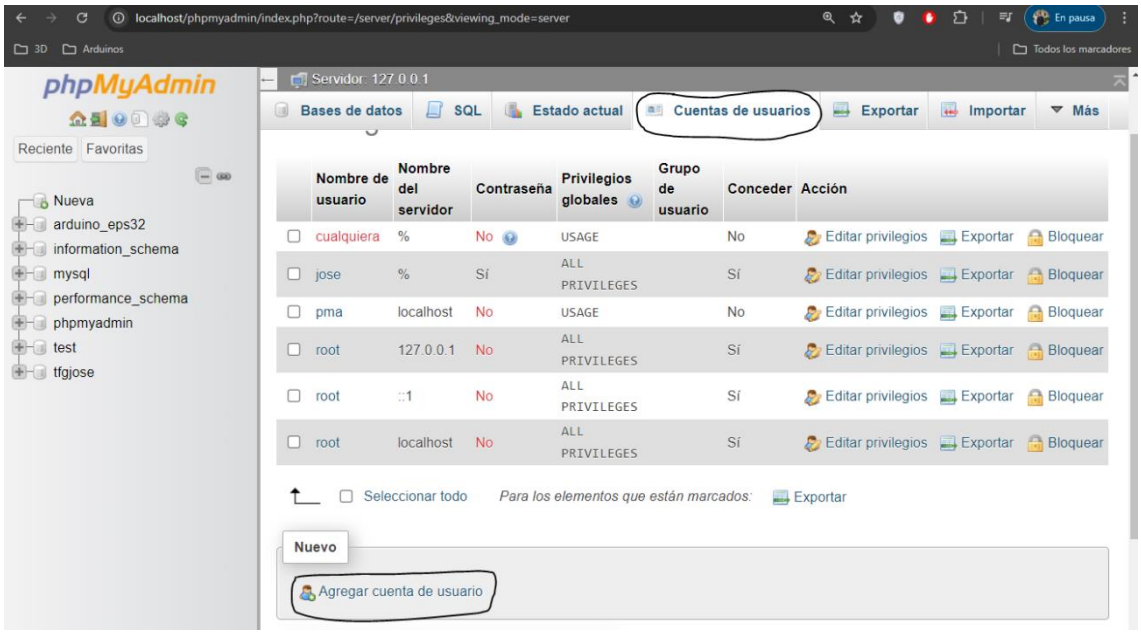


Figura 67. Anexo II, interfaz usuarios SQL

Dentro de la pestaña “Cuentas de usuarios” debemos ir al apartado “Agregar cuenta de usuario” Asignamos un nombre, contraseña y asignamos todos los permisos que queramos para este nuevo usuario.

II.II.III Creación de una base de datos

Figura 68. Anexo II, Interfaz base de datos SQL

Simplemente en la parte izquierda nos vamos a “Nueva” seguido de esto escribimos el nombre de nuestra base de datos y pulsamos crear.

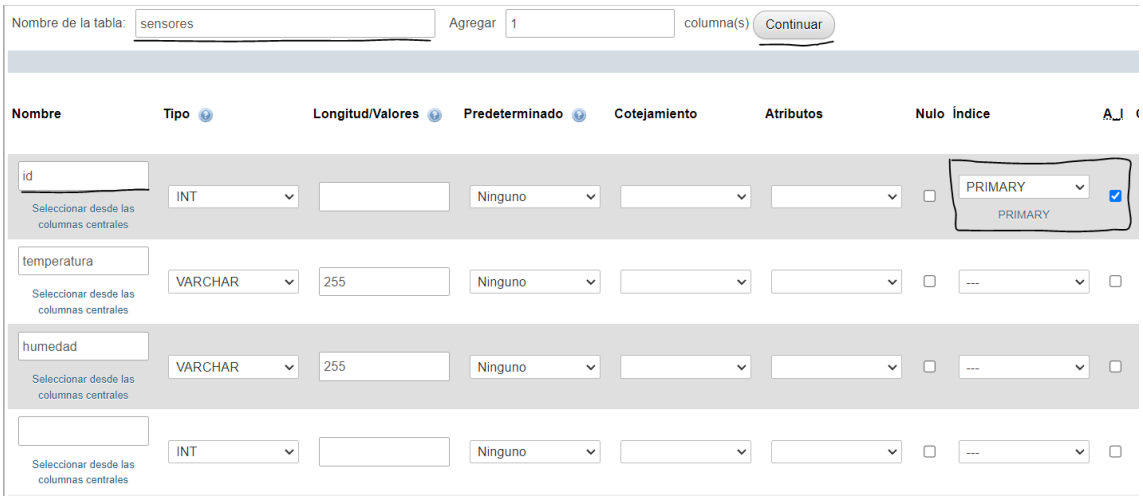


Figura 69. Anexo II, ejemplo creación base de datos

Establecemos el nombre de nuestra tabla en este caso “sensores” y asignamos los campos que queramos en este caso “ID” como un entero, clave primario y autoincrementado, todas las tablas deben tener un elemento de clave primaria y al poner auto incrementable no tenemos que introducir este valor en la consulta.
Para terminar, abajo debemos pulsar el botón “Guardar” y tenemos configurado nuestro entorno de Base de Datos.

II.II.IV. Configuración de nuestro modulo en BlocklyDuino

Dentro de esta sección tenemos los módulos para insertar y Obtener datos.

Para poder usar nuestro modulo previamente tenemos que añadir el bloque de nuestra placa en este caso Arduino ESP32 y el modulo wifi el usuario y la clave de nuestra red local, una vez configurado podemos configurar los módulos, para empezar, vamos a usar el modulo de Insert, la estructura deberá quedar como en la imagen.

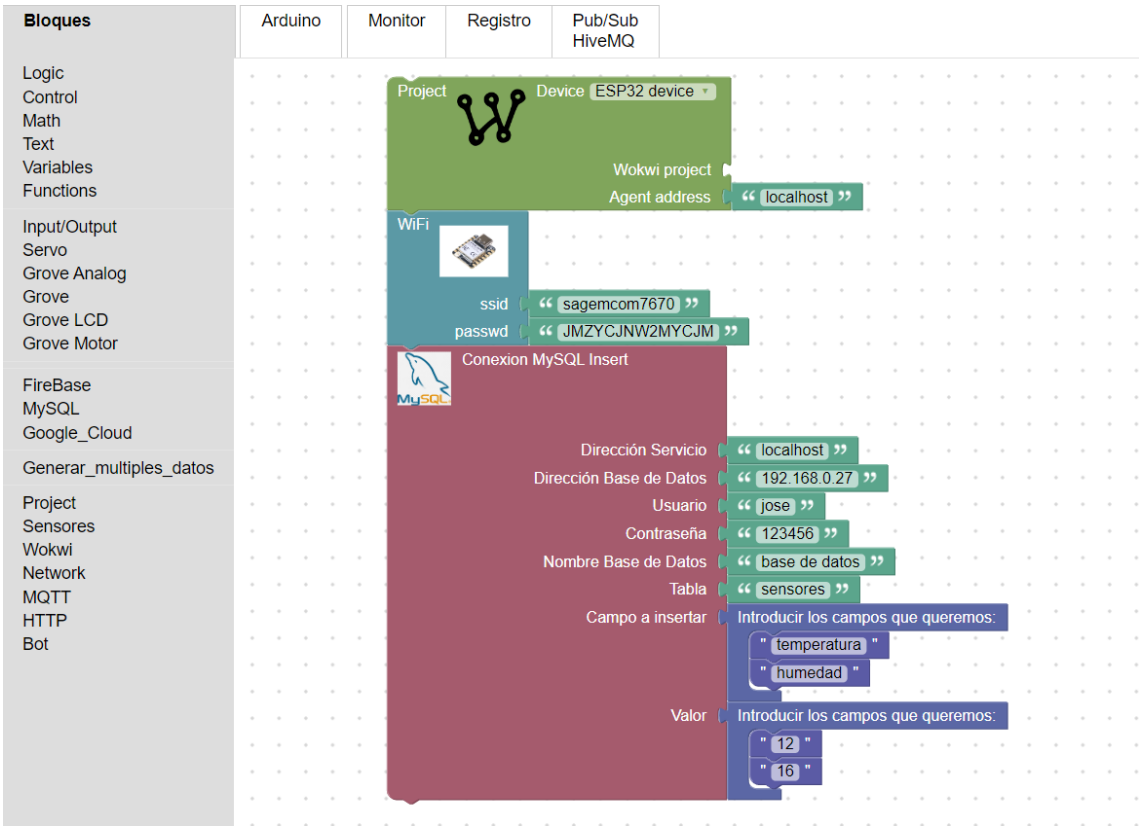


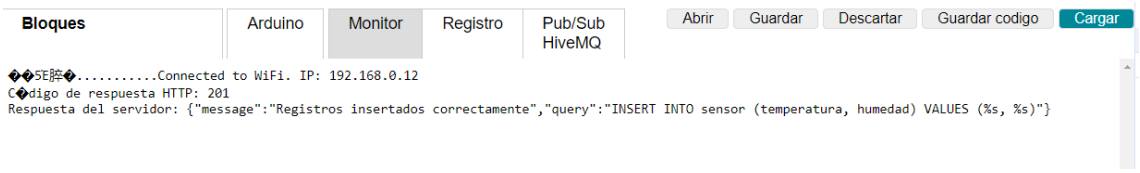
Figura 70. Anexo II, bloque completo SQL

Dentro del módulo **"Conexión MySQL Insert"** debemos introducir los siguientes datos:

- Dirección del servicio donde esta desplegado el servicio BlocklyDuino
- Dirección Base de Datos

Dentro de la parte de campos a insertar si queremos introducir mas de un valor necesitamos usar los bloques de "Generar_multiples_datos" para introducir un array, cuando tengamos la siguiente estructura es recomendable utilizar un módulo "Delay" dentro de los bloques "Control"

Dentro de la pestaña "Arduino" podemos ver el código generado y para ponerlo a funcionar solamente tenemos que conectar nuestro Arduino ESP32 e ir a la pestaña "Cargar".
Para ver si todo se esta cargando correctamente podemos ir a la pestaña "Registros" y para terminar si se ha insertado correctamente podemos comprobarlo desde el panel de MySQL o desde la pestaña "Terminal"



Podemos comprobar que se ha insertado correctamente, pero la segunda comprobación es mirar en MySQL

Figura 71. Anexo II, ejemplo inserción SQL

Ahora vamos a configurar el módulo "**Conexión MySQL Select**" debe de tener la siguiente estructura al igual que el módulo anterior.

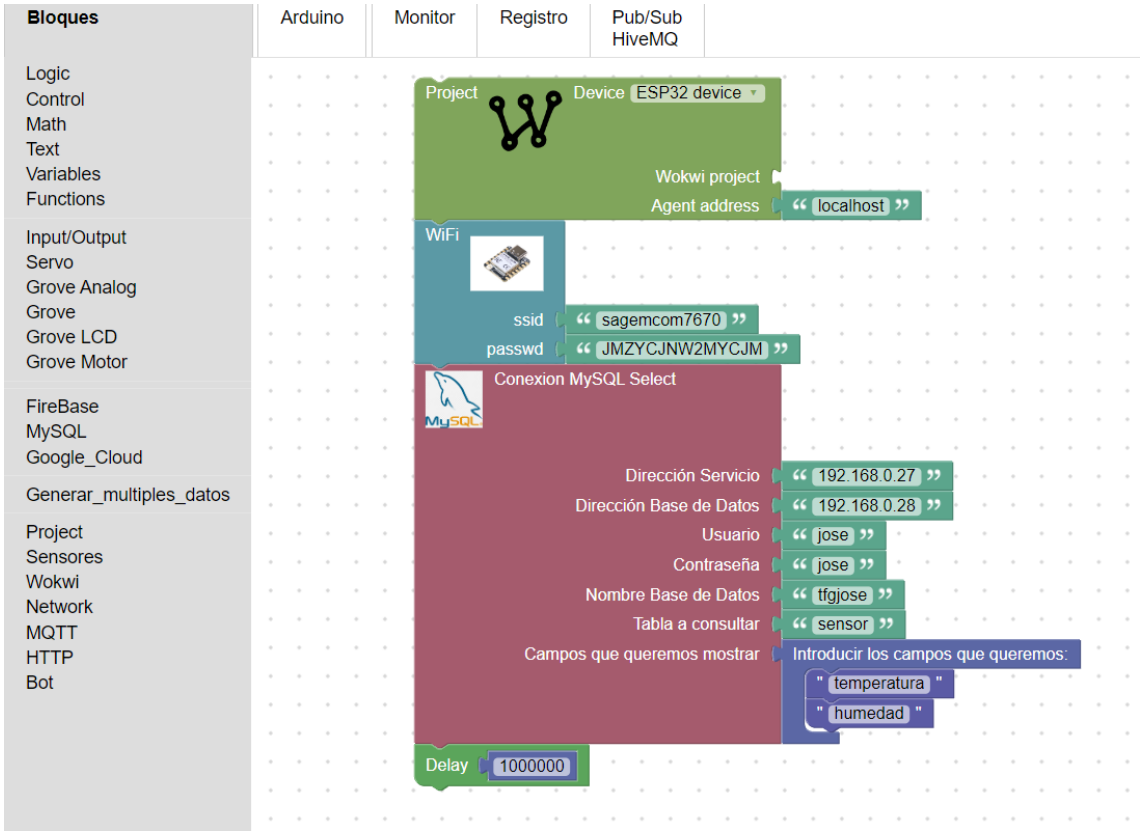


Figura 72. Anexo II, Ejemplo bloque selección SQL

Simplemente para cargar hacemos exactamente lo mismo que el módulo anterior.

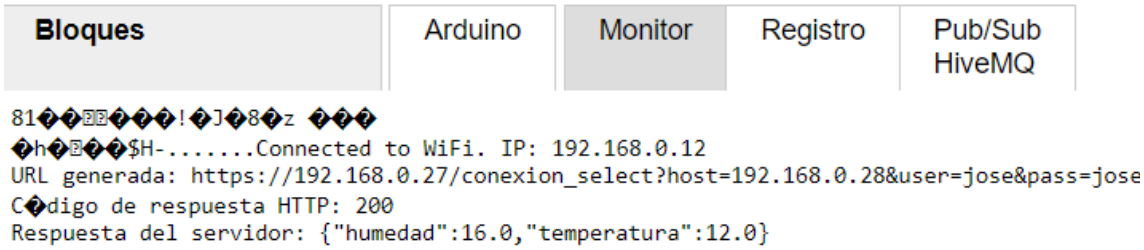


Figura 73. Anexo II. Respuesta del monitor serie

Y podemos comprobar que se devuelve el último dato introducido.

II.III. Configuración módulo FireBase

Previo a la configuración del módulo tenemos que configurar nuestra cuenta en Firebase, para empezar, tendremos que acceder a la siguiente URL:

https://firebase.google.com/?hl=es

Accedemos a la consola y creamos un nuevo proyecto:

Figura 74. Anexo II, Interfaz inicio Firebase

Añadimos el nombre de nuestro proyecto (El que desee) y posteriormente no tenemos que habilitar Analytis.

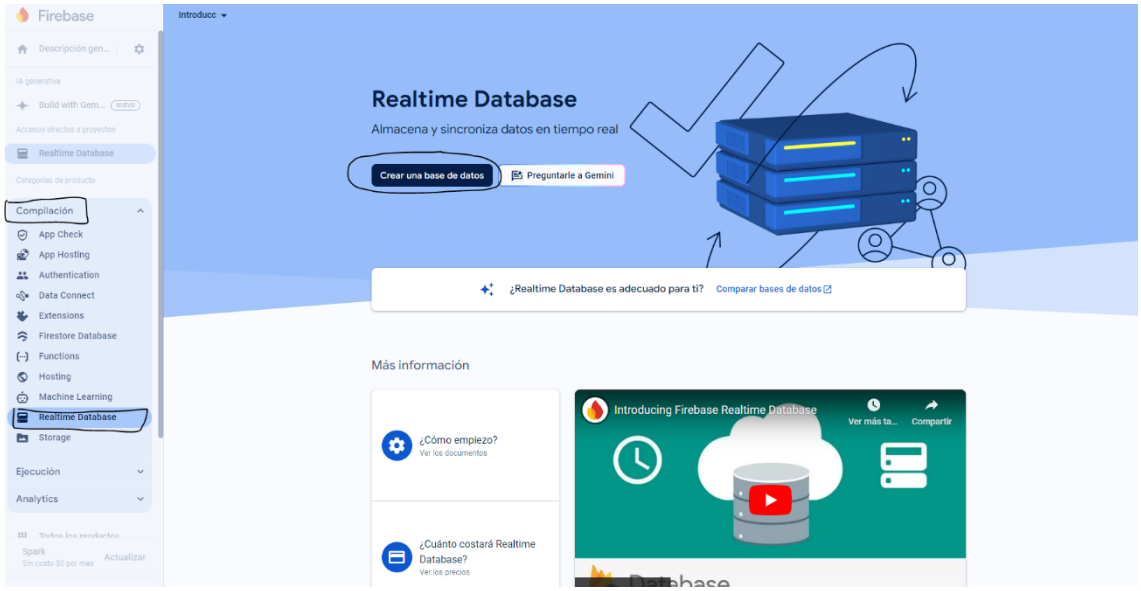


Figura 75. Anexo II, Interfaz firebase principal

Dentro del panel Izquierdo tendremos que seleccionar Compilacion – Realtime Database y creamos una base de datos, elegimos el servidor que prefiramos y después seleccionamos modo bloqueo, luego tendremos que modificar las siguientes líneas y ponerlas en true.

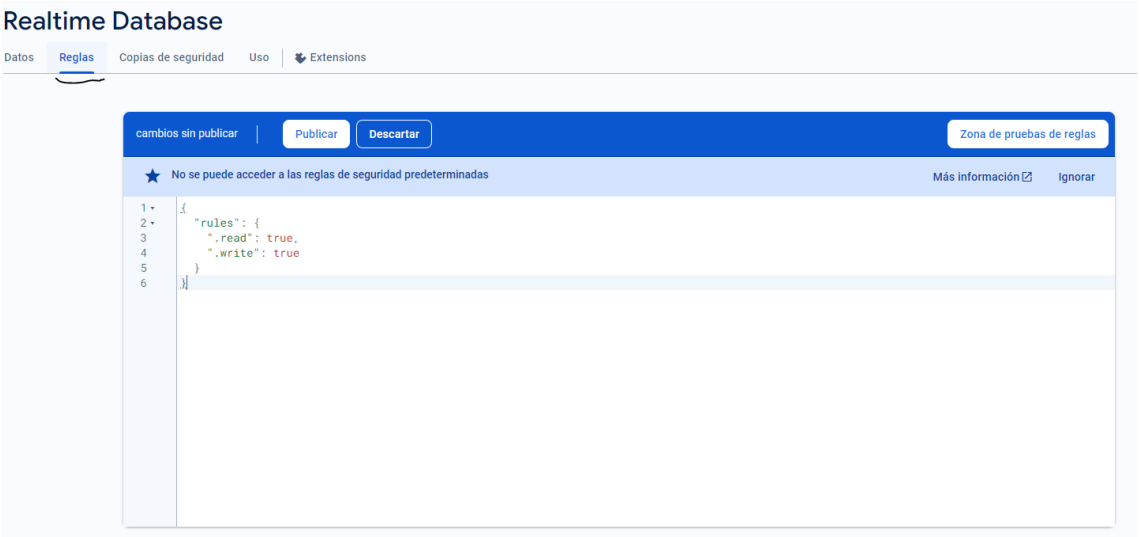


Figura 76. Anexo II, Configuración Realtime Database

Vamos a reglas cambiamos el valor a true y publicamos.

Ahora vamos a crear nuestro usuario para poder conectarnos, vamos a ir a la parte de authenticacion.

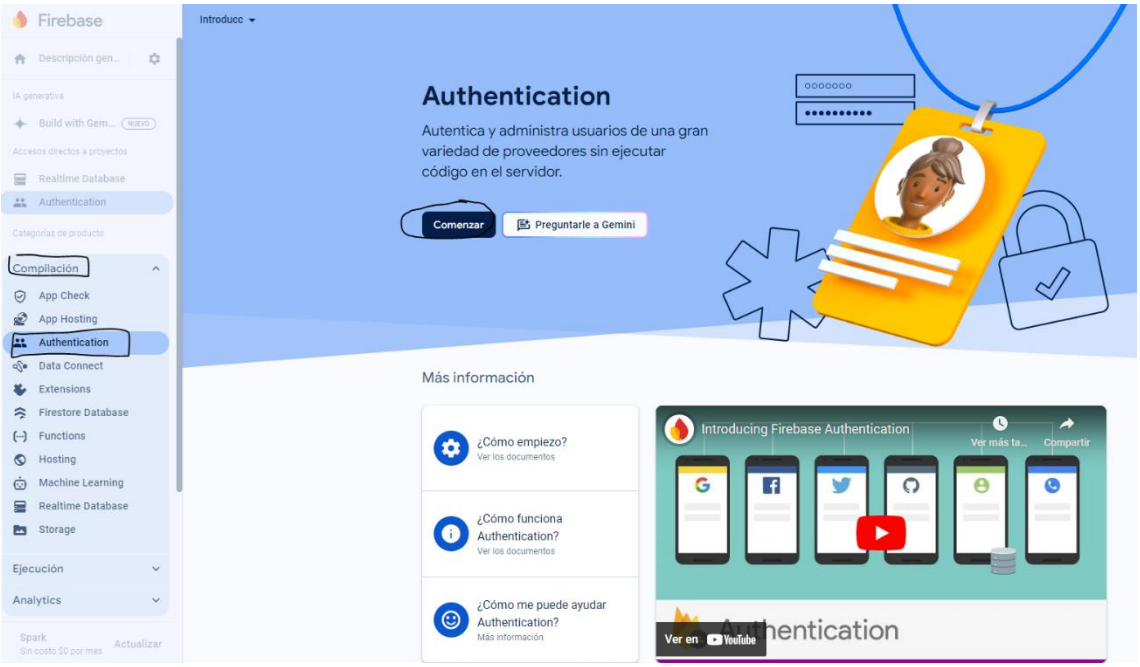


Figura 77. Anexo II, Interfaz autenticación

Seleccionamos comenzar

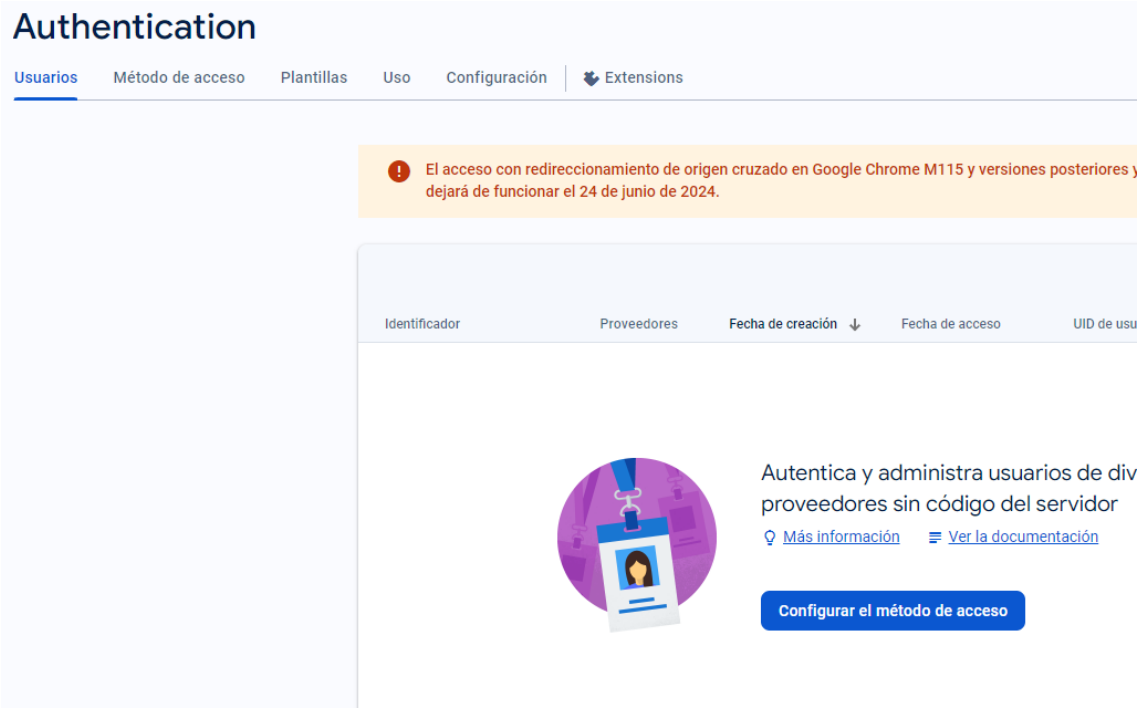


Figura 78. Anexo II, Creación usuario autorizado

Seguidamente vamos a la pestaña usuario y configuramos el método de acceso, seleccionamos correo electrónico, lo habilitamos, volvemos a la pestaña de usuario y creamos un usuario.

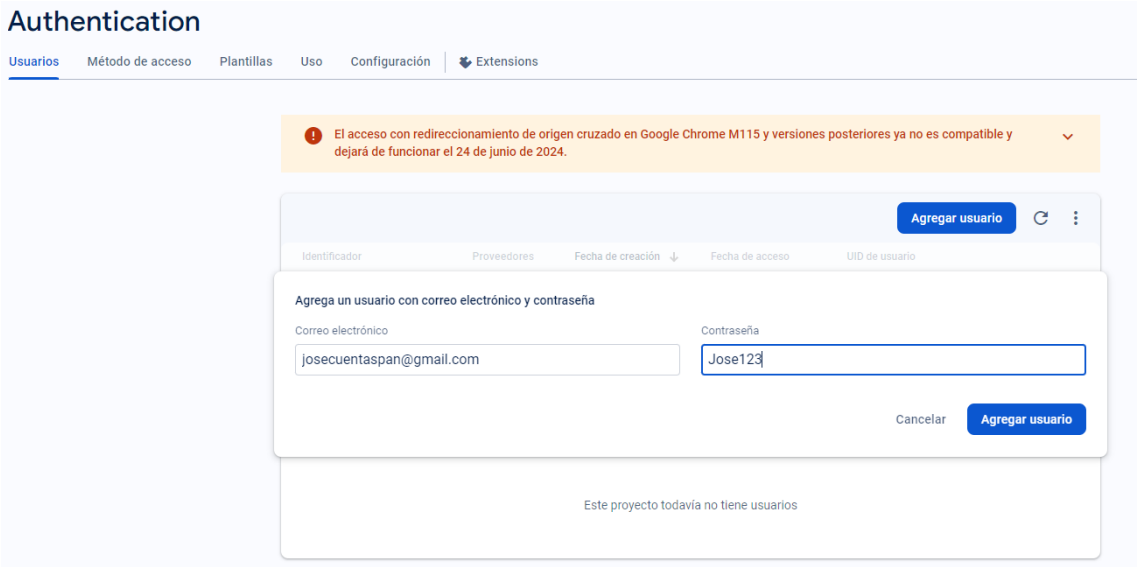


Figura 79. Anexo II, Cuenta usuario

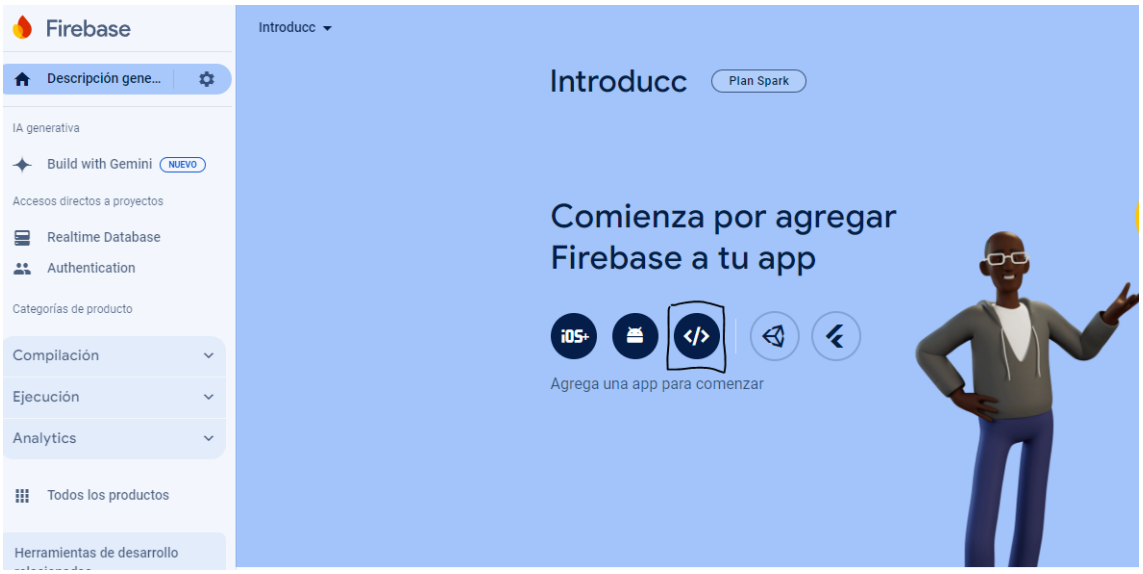


Figura 80. Anexo II, extracciones credenciales para la conexión

Ahora en la parte de descripción general seleccionamos el icono marcado en la imagen y configuramos con el nombre que deseemos como vemos a continuación.

Figura 81. Anexo II, ejemplo datos de conexión

Ahora de esta pantalla necesitamos copiar el apiKey y el databaseURL. Y ya tendríamos todo lo necesario.

II.III.I. Configuración de nuestro modulo en BlocklyDuino

Ahora tendremos que ir a nuestro Blockly y colocar los bloques que deseemos de Firebase en este ejemplo lo dejaremos así:

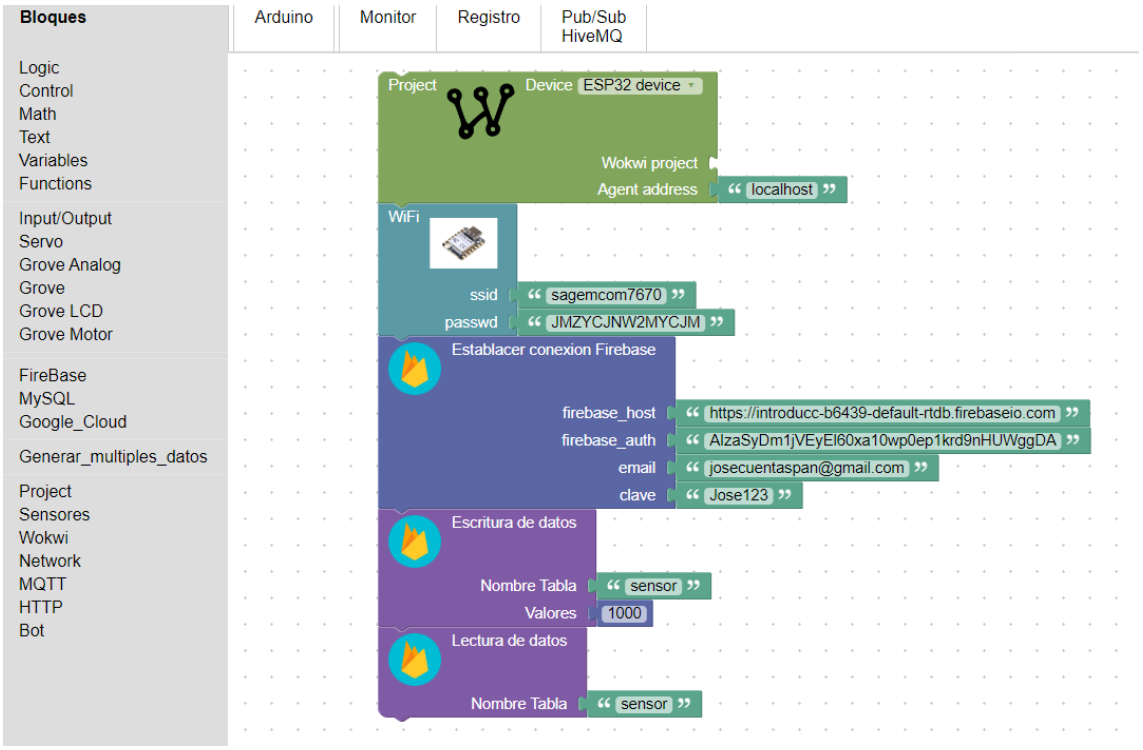


Figura 82. Anexo II, Ejemplo bloque completo Firebase

Se nos va a pedir los siguientes datos:

- Firebase_host: Corresponde al database_url
- Firebase_auth: Corresponde a apiKey
- Nombre Tabla, el nombre que queramos asignar

Si todo lo hicimos correctamente al darle al botón de cargar tendremos funcionando nuestro sistema de carga.

```
C◆digo de error HTTP: -11
........Connected to WiFi. IP: 192.168.0.12
-----------------------------------
Token info: type = id token (GITKit token), status = on request
Token info: type = id token (GITKit token), status = ready
Data sent successfully: 1000.00
Data received: 1000.00
```

Figura 83. Anexo II, respuesta monitor del bloque firebase

Y dentro de la plataforma FireBase podemos ver como se ha generado ese dato.

Figura 84. Anexo II, Dato introducido en Realtime Database

II.IV. Configuración modulo Google Cloud

Para configurar previamente deberemos tener o crear una cuenta en Google Cloud y crear un proyecto, luego en el buscador tendremos que poner Pub/Sub de la siguiente forma:

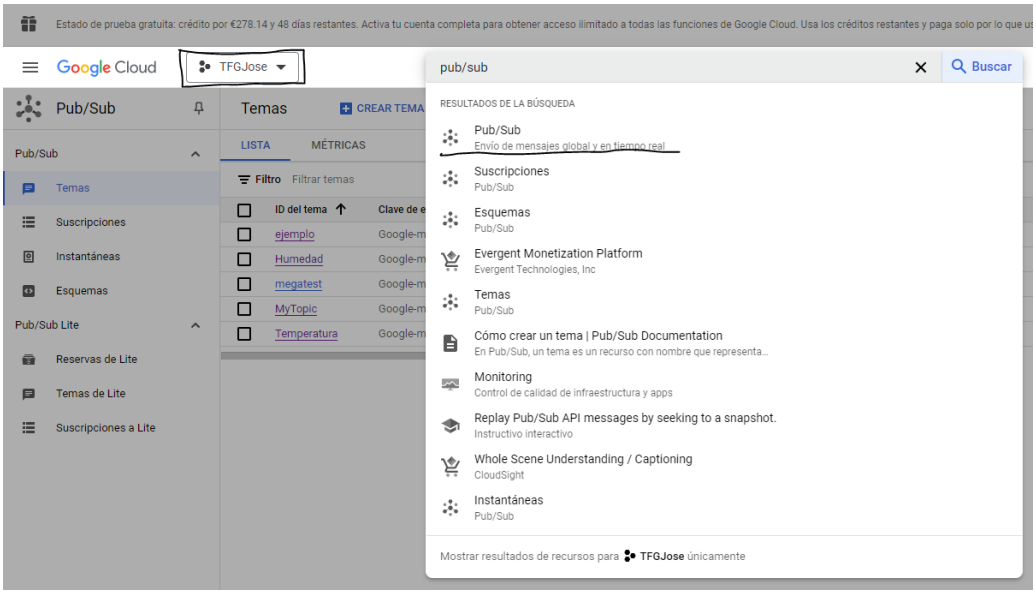


Figura 85. Anexo II, GoogleCloud menú

Ahora tendremos que darle a “Crear Tema”

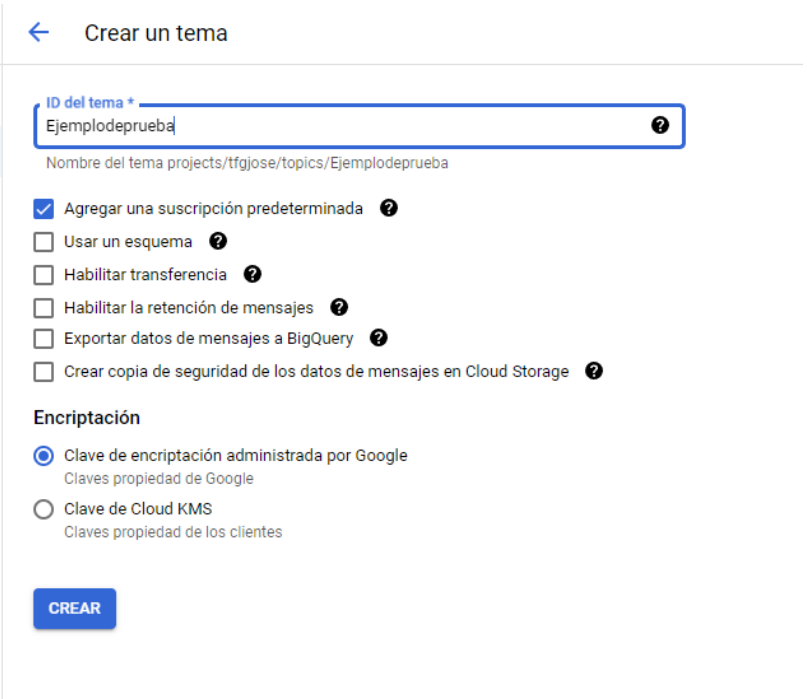


Figura 86. Anexo II, GoogleCloud creación tema

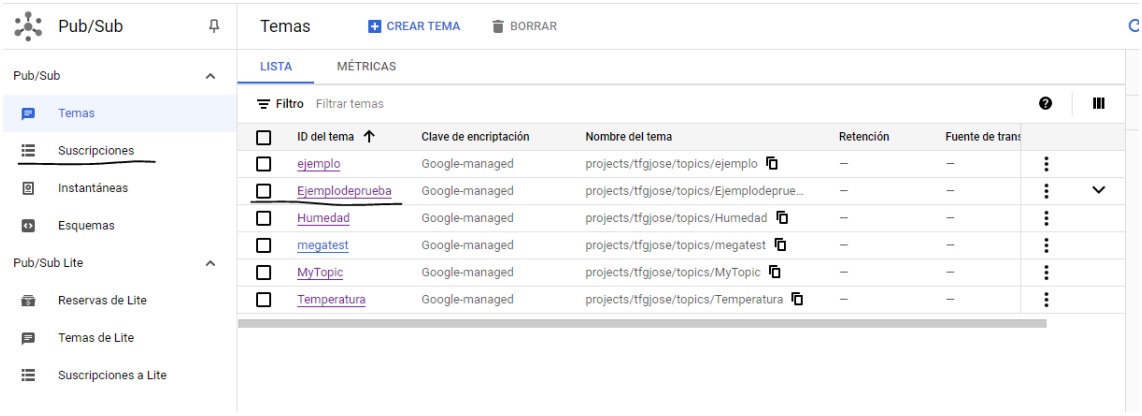


Figura 87. Anexo II, Ejemplo tema Google Cloud

Ahora tendremos generado un tema y una suscripción que se genera de forma automática, ahora tendremos que crear un usuario valido para poder subir un mensaje a un tema. En el buscador podemos poner “Cuentas de servicio”.

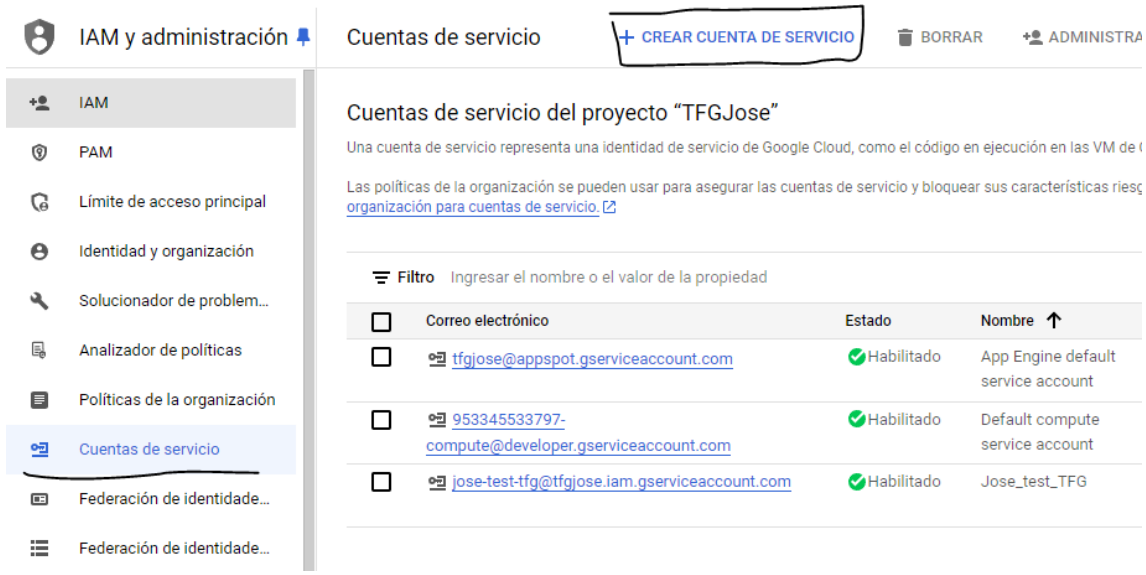


Figura 88. Anexo II, Cuentas de servicio Google Cloud

Creamos una cuenta de servicio

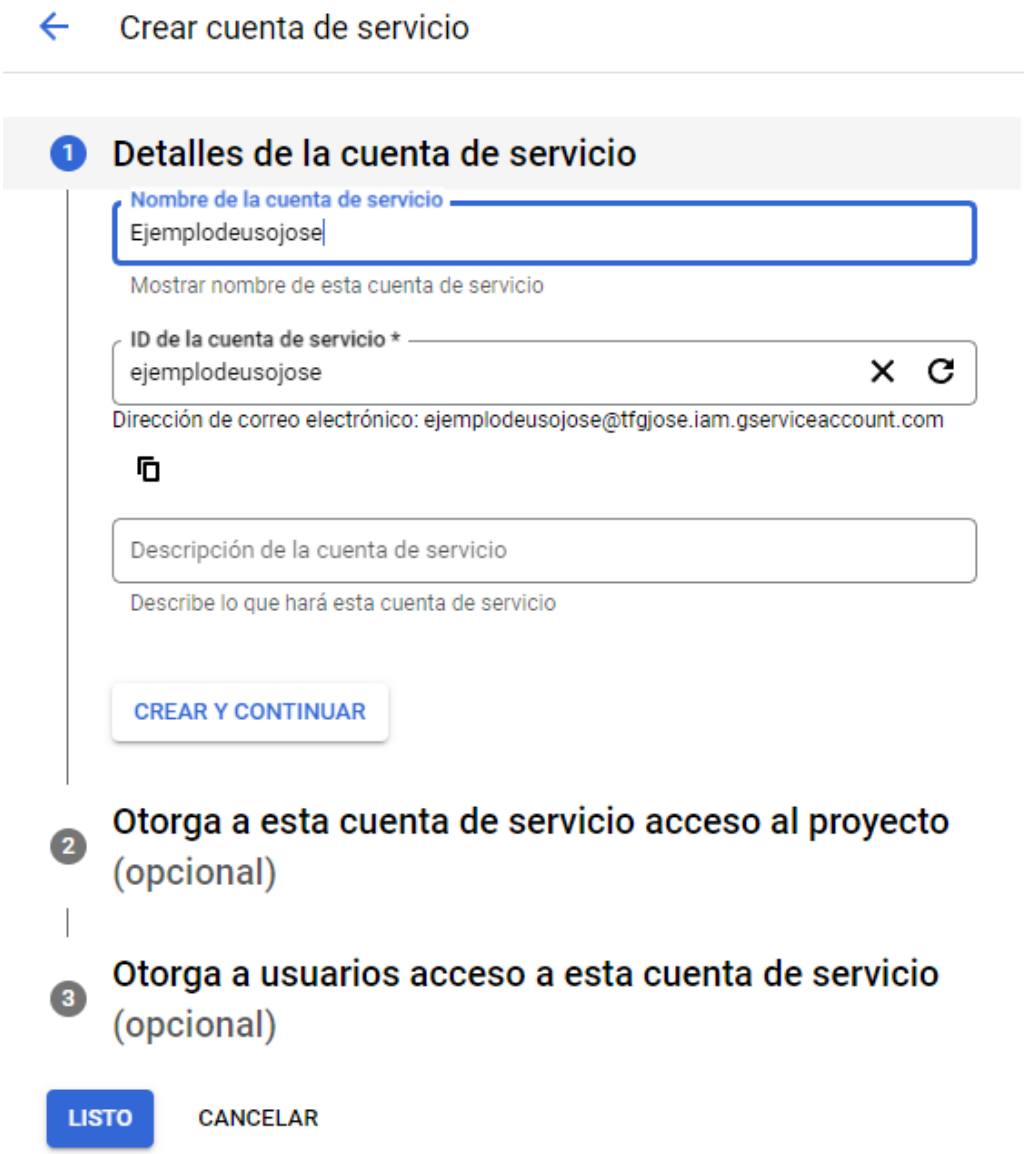


Figura 89. Anexo II, Interfaz creación cuenta de servicio

Dentro de Otorgar a esta cuenta de servicio aplicamos los privilegios de propietario para poder manejar todos los servicios.

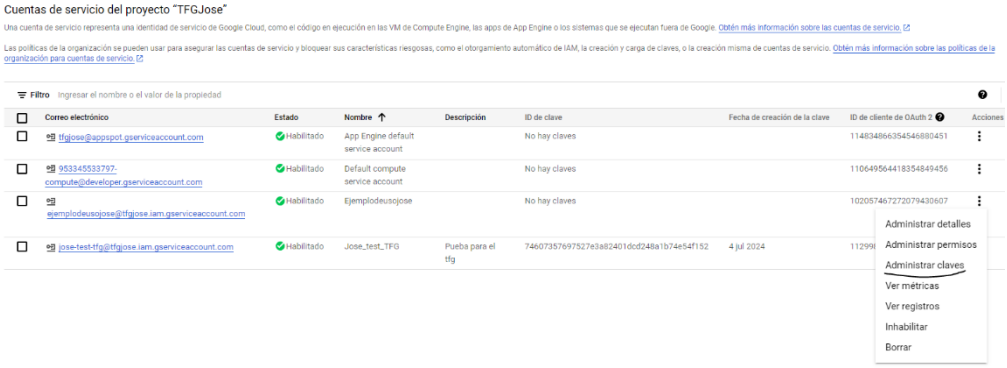


Figura 90. Anexo II, Cuentas de usuario

Creamos unas claves y se nos descargará un archivo con estas claves.

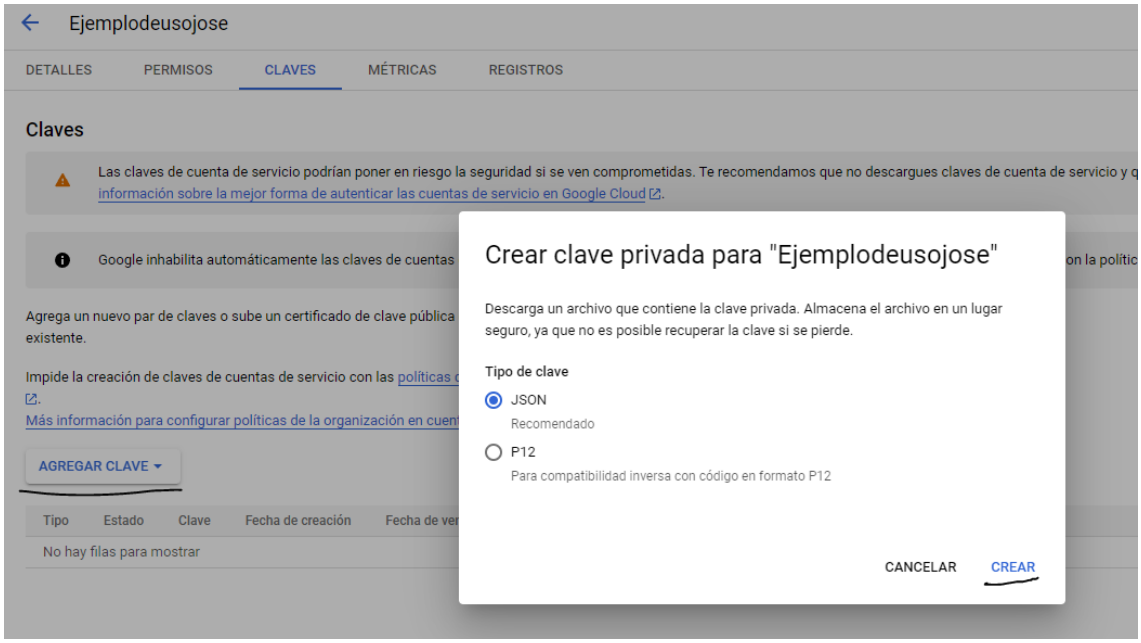


Figura 91. Anexo II, Creación de claves

Del archivo descargado necesitamos copiar la clave que es la siguiente:

Figura 92. Anexo II, Fichero credenciales conexión a Google Cloud

II.IV.I. Configuración de nuestro modulo en BlocklyDuino

Para el bloque tenemos la siguiente estructura:

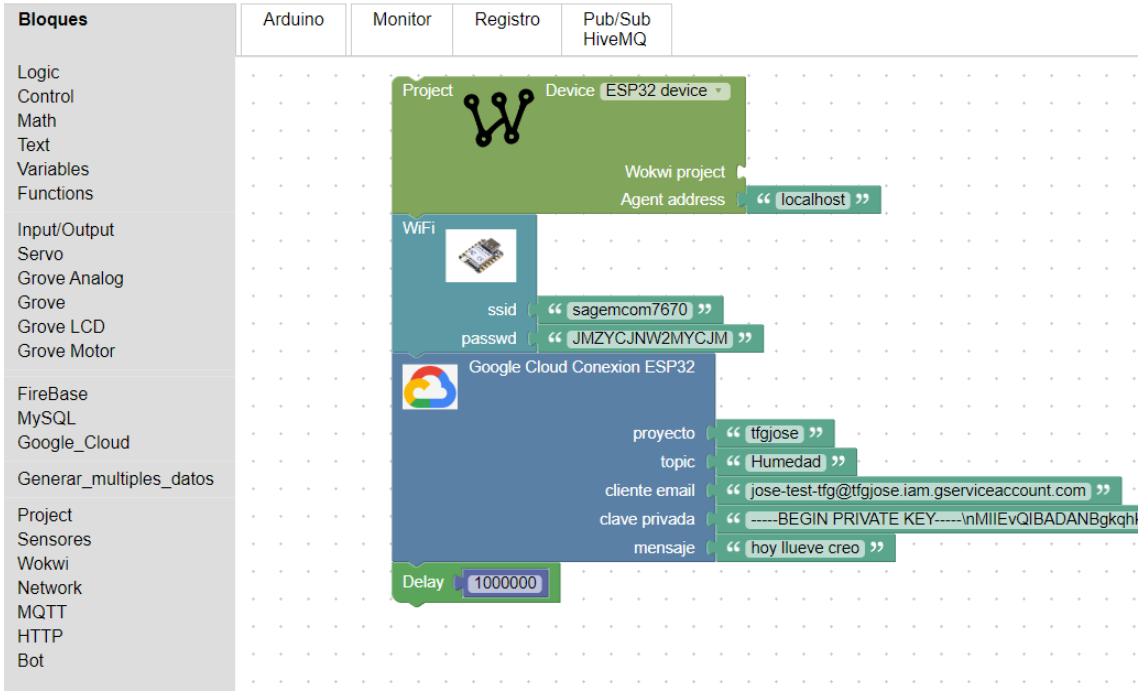


Figura 93. Anexo II, Ejemplo bloque completo Google Cloud

Donde tenemos los siguientes campos:

- Proyecto: Nombre de nuestro proyecto
- Topic: Nombre del tema creado anteriormente
- Cliente email: La nueva cuenta que hemos generado con los permisos
- Clave privada: La obtenida en el fichero. Json

Si lo ejecutamos tendremos lo siguiente:

```
.............Connected to WiFi. IP: 192.168.0.12
HTTP Response code: 200
{
  "messageIds": [
    "12021269059092992"
  ]
}
```

Figura 94. Anexo II, Ejemplo respuesta Serial Google Cloud

Nos indica que todo se ha generado correctamente, para poder localizarlo solamente tendríamos que ir a Pub/Sub elegir el suscriptor que se vincule con el tema al que enviamos el mensaje y extraemos el mensaje de la siguiente manera.

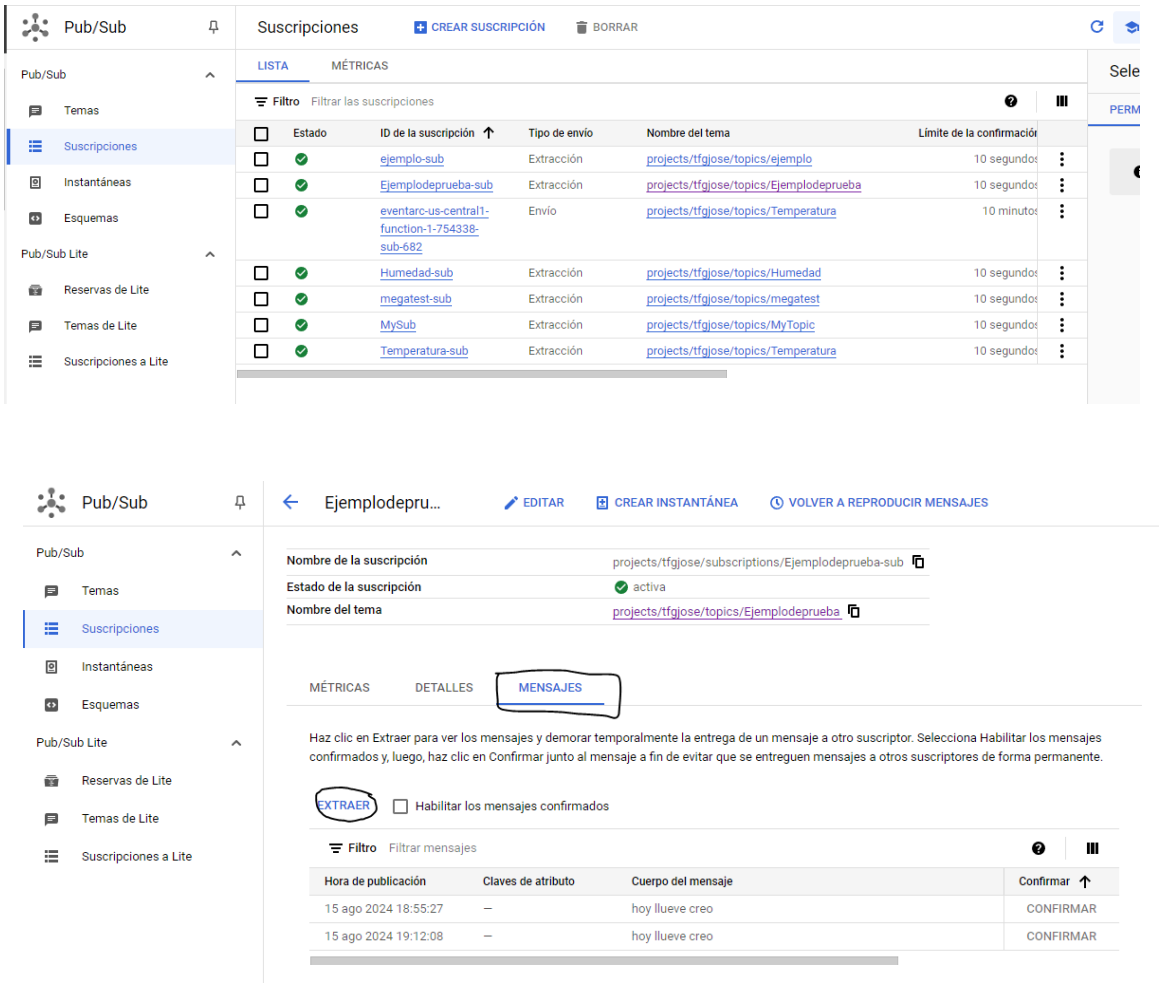


Figura 95. Anexo II, Ejemplo dato insertado mediante el método Pub

Y ahí podemos extraer todos los mensajes que tenemos publicados en dicho tema.

II.V. Insert en SQL Cloud a través de evento Pub en determinado topic

Se va a detallar la configuración necesaria para poder realizar una inserción en una base de datos SQL del valor recibido a través de Pub/Sub en un topic a través de Cloud Function.

Deberá de tener un proyecto en Google Cloud y cumplir los prerrequisitos.

Se va a tomar de forma independiente al resto de Anexos puesto que anteriormente vienen detallados ciertos pasos, para comenzar se va a crear un servicio Pub/Sub, accediendo en el buscador a Pub/Sub creamos un tema

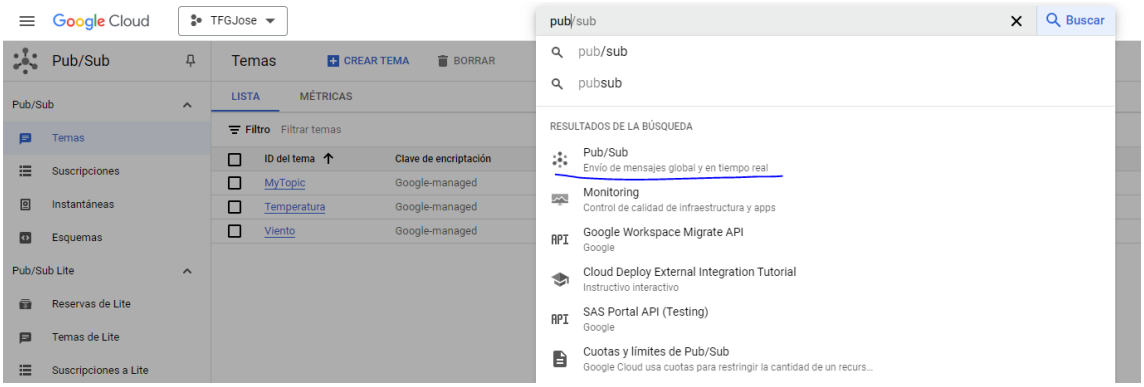


Figura 96. Anexo II, Creación Pub en Google Cloud

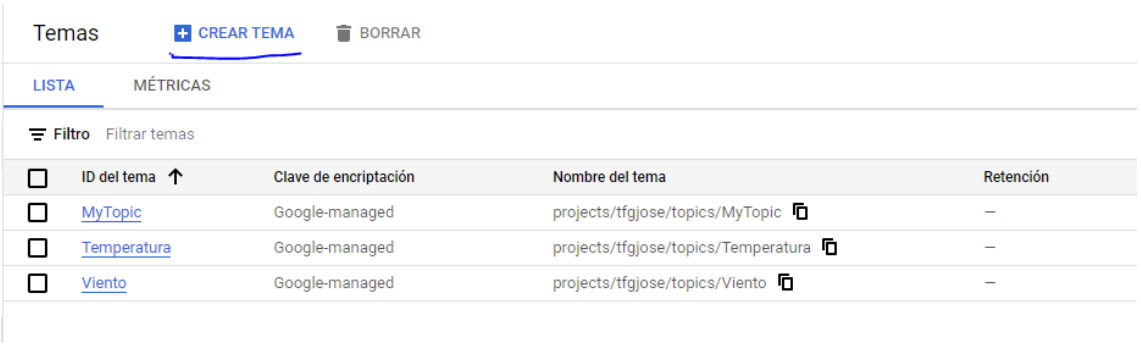


Figura 97. Anexo II, Creación nuevo tema Google Cloud

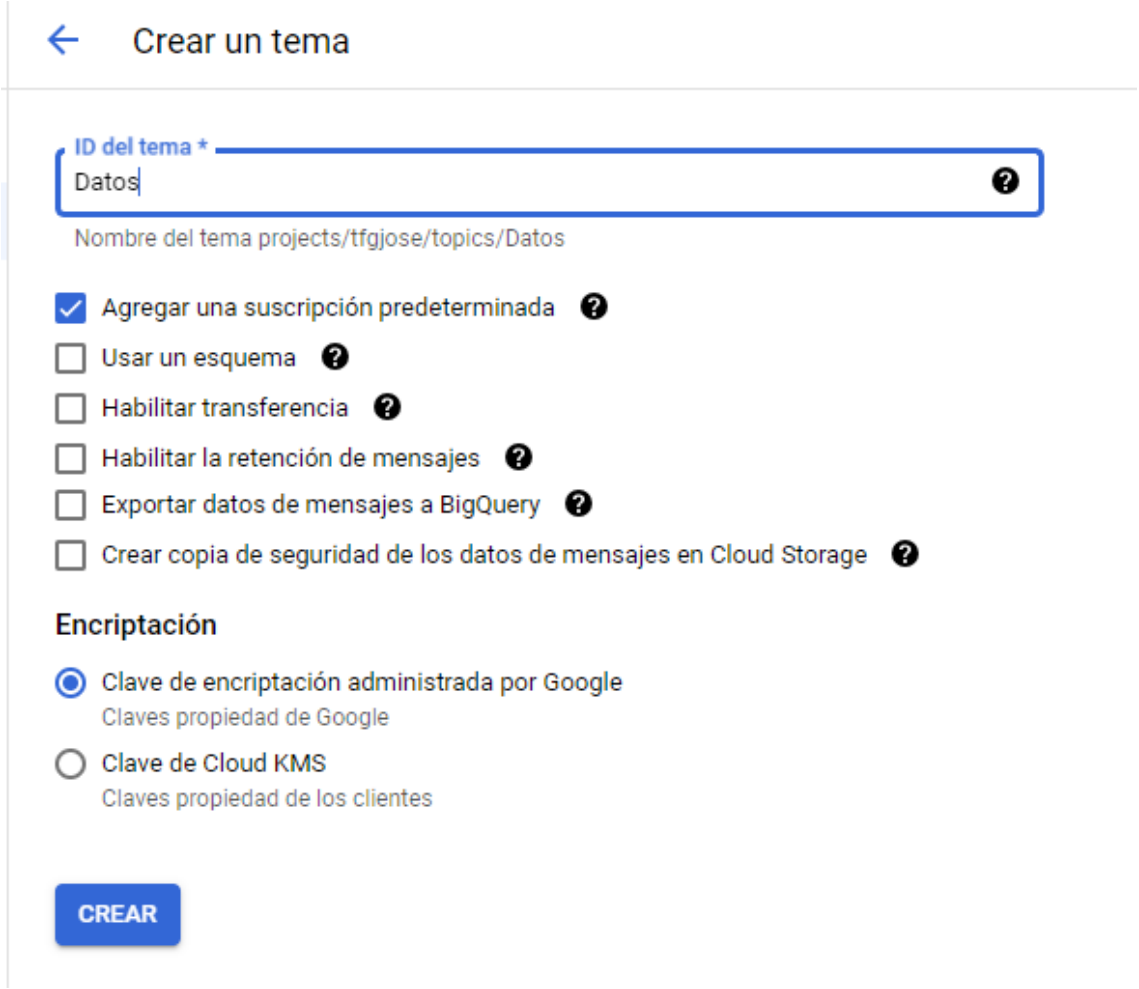


Figura 98. Anexo II, Ejemplo tema datos en Google Cloud

Al pulsar en crear directamente aparecerá el suscriptor creado, ahora crearemos una red privada virtual (VPC) mediante la cual se establecerá la conexión con la base de datos, para la creación de la VPC al igual que anteriormente en el buscador vamos a escribir “Redes de VPC” para crear una red VPC.

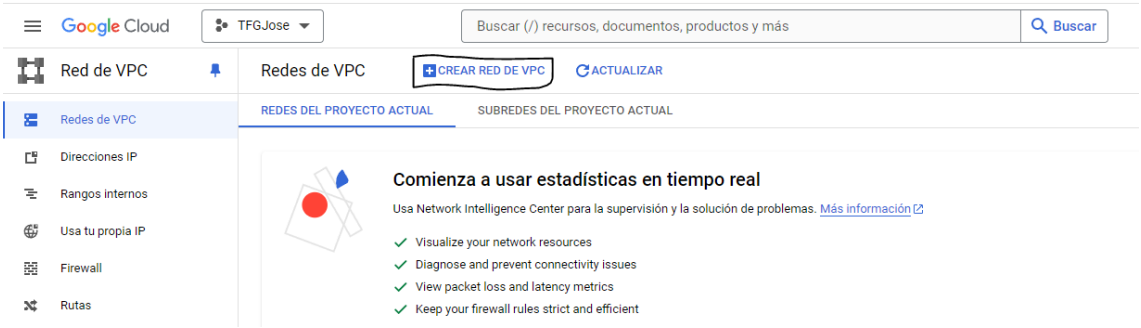


Figura 99. Anexo II, Creación red VPC

Para la configuración es importante asignar un nombre (cualquiera) una región y un rango, en la foto a continuación se ve una configuración ejemplo.

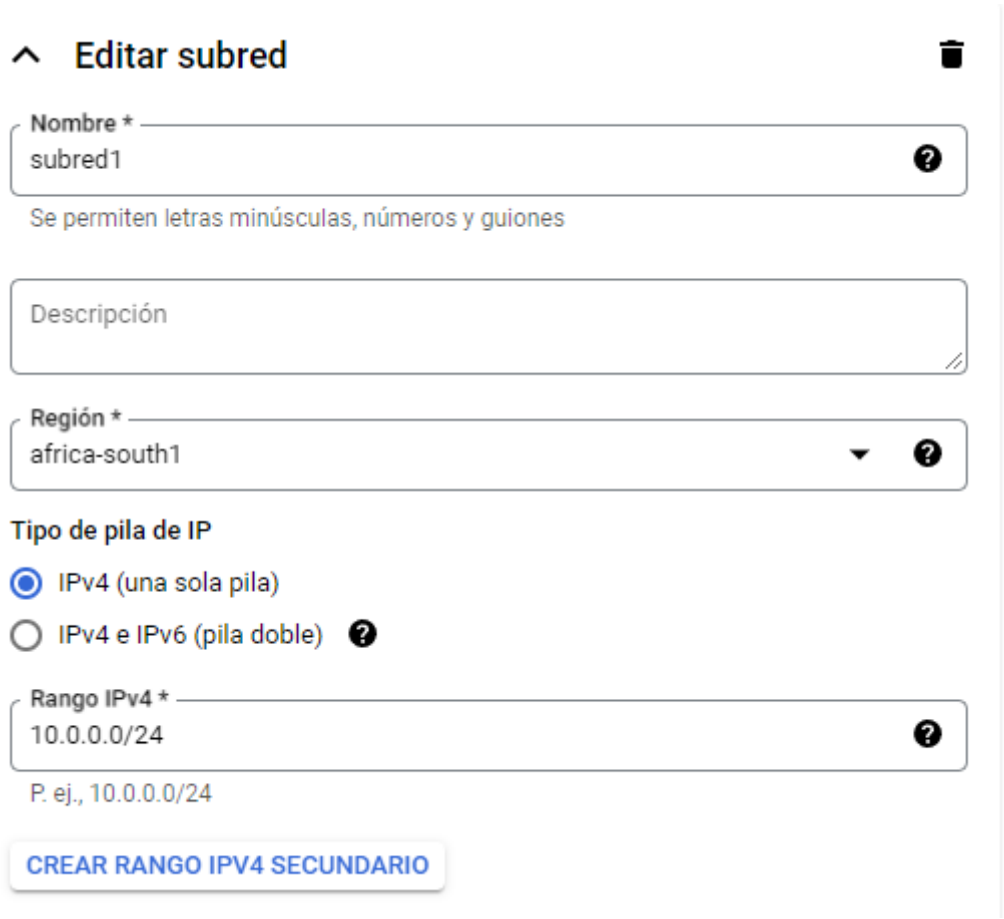


Figura 100. Anexo II, Ejemplo configuración subred

Teniendo creada la VPC se va a crear “Acceso a VPC sin servidores” para asignar un rango de direcciones que será posteriormente aceptada en la base de datos, para hacer esto creamos un conectar utilizando la VPC creada anteriormente y se le asignará un nombre, en este caso se pondrá “ejemplo” asignada a la VPC “subred1”.

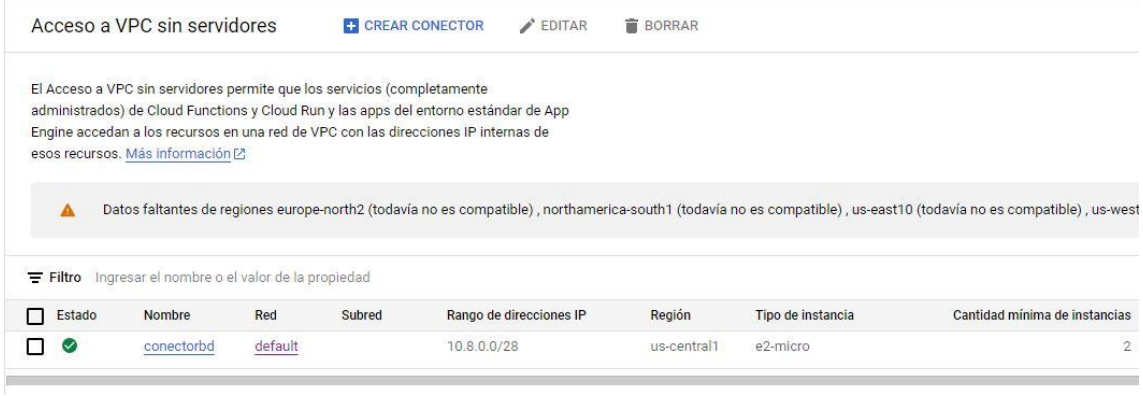


Figura 101. Anexo II, Conector VPC sin servidores

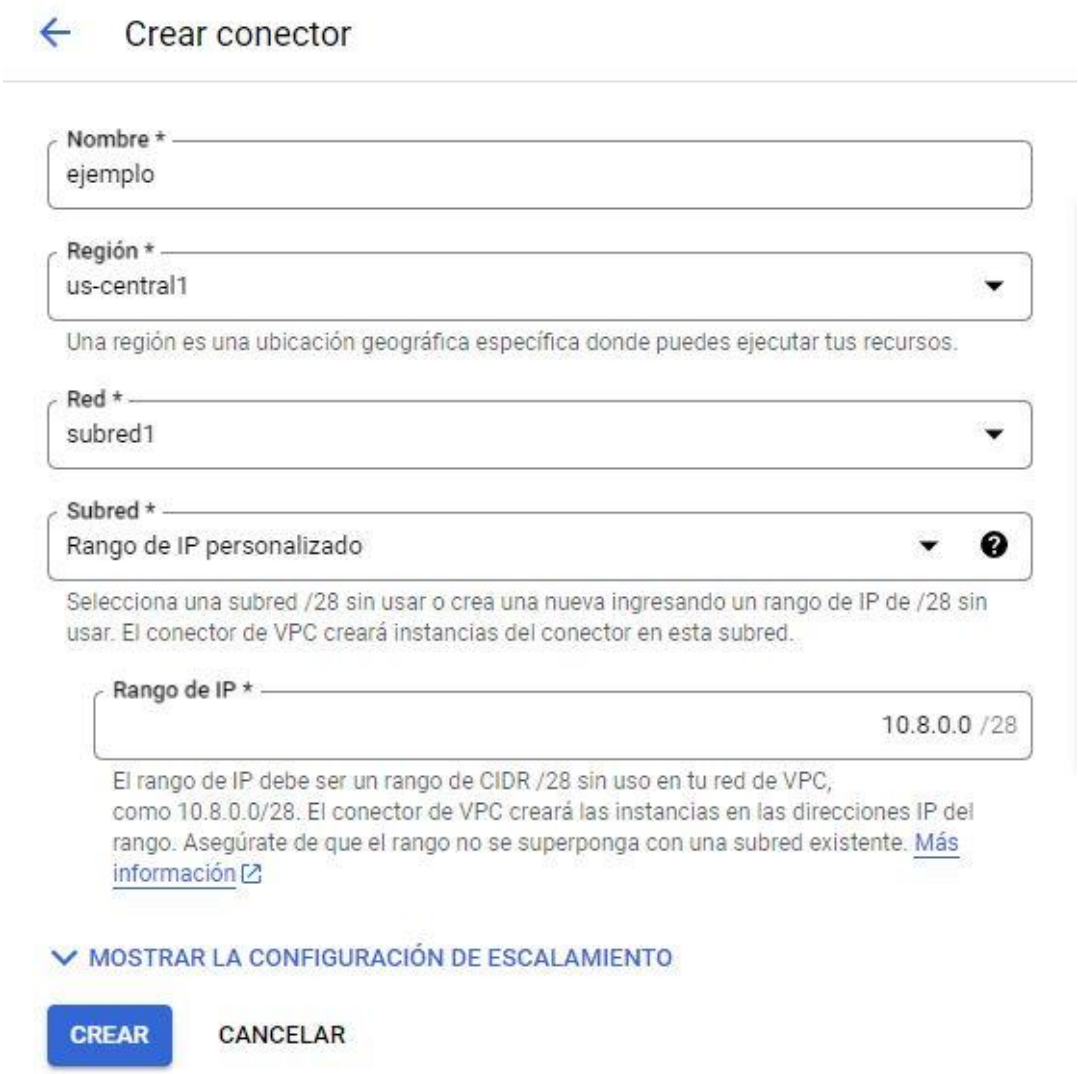


Figura 102. Anexo II, Conector Google Cloud

teniendo esto podremos pasar a la creación de la base de datos, se hará a través de SQL (Accediendo igualmente a través del buscador), se creará una instancia, como solamente tendrá un uso destinado a pruebas se creará utilizando el mínimo de recursos para optimizar el coste.

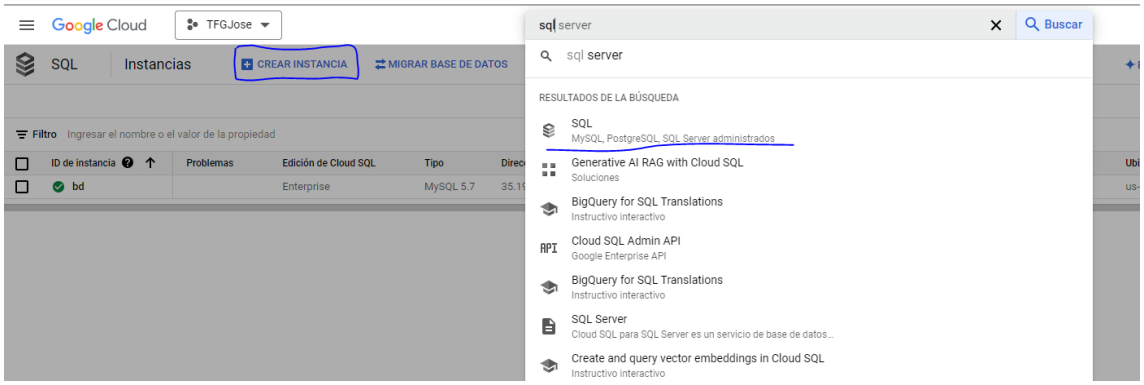


Figura 103. Anexo II, Creación instancia SQL en Google Cloud

Como motor se usará MySQL, y los parámetros se utilizará según el uso que espere de la base de datos.

Un ejemplo de configuración podría ser el siguiente:

Resumen

Edición de Cloud SQL	Enterprise
Región	us-central1 (Iowa)
Versión de la base de datos	MySQL 8.0
CPU virtuales	1 CPU virtual(es)
RAM	3.75 GB
Caché de datos	Inhabilitada
Almacenamiento	10 GB
Conexiones	IP pública
Copia de seguridad	Automatizada
Disponibilidad	Zona única
Recuperación de un momento determinado	Habilitada
Capacidad de procesamiento de la red (MB/s)	250 de 250
Capacidad de procesamiento del disco (MB/s)	Lectura: 4.8 de 200.0 Escritura: 4.8 de 200.0
IOPS	Lectura: 300 de 12,000 Escritura: 300 de 10,000

Figura 104. Anexo II, Ejemplo configuración funcional de una base de datos

Tendremos que esperar a cargar la instancia, una vez terminado se configurará las redes permitidas para establecer la conexión y la creacion de la base de datos, se hara como se indica a continuación:

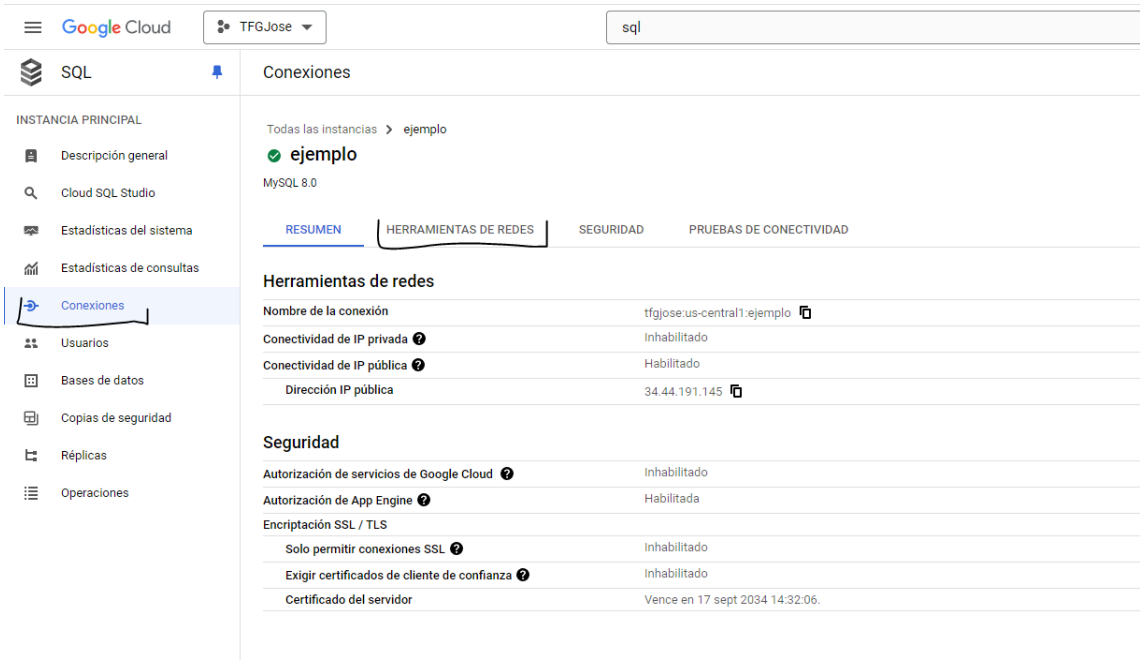
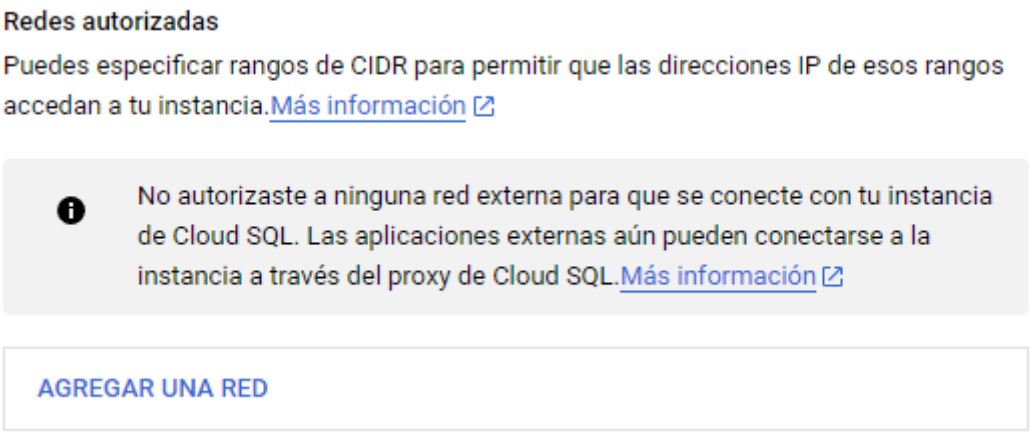


Figura 105. Anexo II, Resumen conexiones en base de datos

Es importante tener marcada la opción IP pública, además se buscará la opción “Agregar una red” y se escribirá la red configurada anteriormente en el conector, además se podrá añadir la red local si posteriormente se quiera utilizar el servicio React. Si hemos seguido el ejemplo no hará falta añadir la dirección puesto que las direcciones internas vienen previamente añadidas, para que funcione correctamente se deberá **añadir la red 0.0.0.0/0** para permitir todo el tráfico.



Para la configuración de la base de datos y como manera opcional se recomienda la creación de un usuario y una base de datos.

Figura 106. Anexo II, Agregación cuenta de usuario en la instancia SQL

Para la base de datos simplemente se accederá a su pestaña correspondiente y se le pondrá el nombre deseado, una vez creada se creará las tablas asociadas a esa base de datos, en este ejemplo de realizará a través de “Cloud SQL Studio” añadiendo la base de datos y el usuario creado previamente.

Figura 107. Anexo II, Ejemplo datos de acceso a Cloud SQL

Para finalizar se propone el siguiente comando para la creación de una tabla con diversos datos los cuales se podrá visualizar posteriormente.

```
CREATE TABLE sensores (
    id INT AUTO_INCREMENT PRIMARY KEY,
    valor INT,
    fecha DATE
);
```

Con el comando **Describe <tabla>** podremos ver el tipo de los campos creados.

Creada ya nuestra base de datos solo faltará el disparador que se activará al recibir un Pub en uno de los topic, para eso se configurará una Cloud fuction como se muestra a continuación:

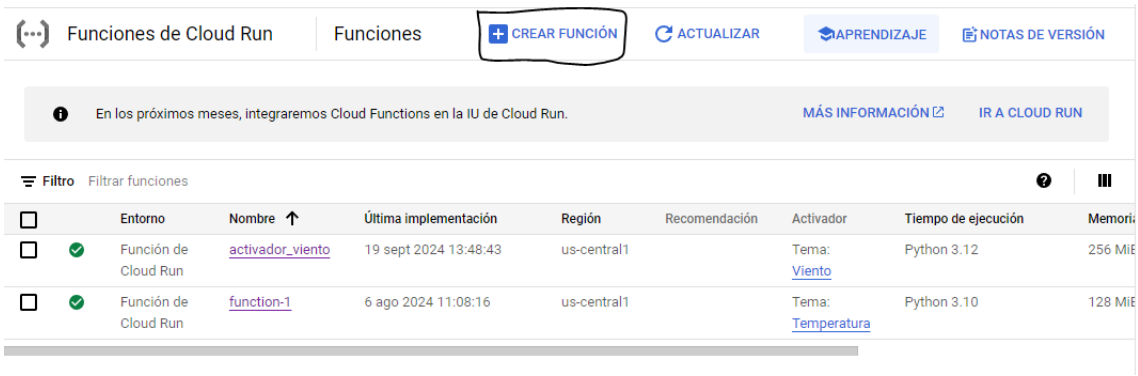


Figura 108. Anexo II, Configuración del activador

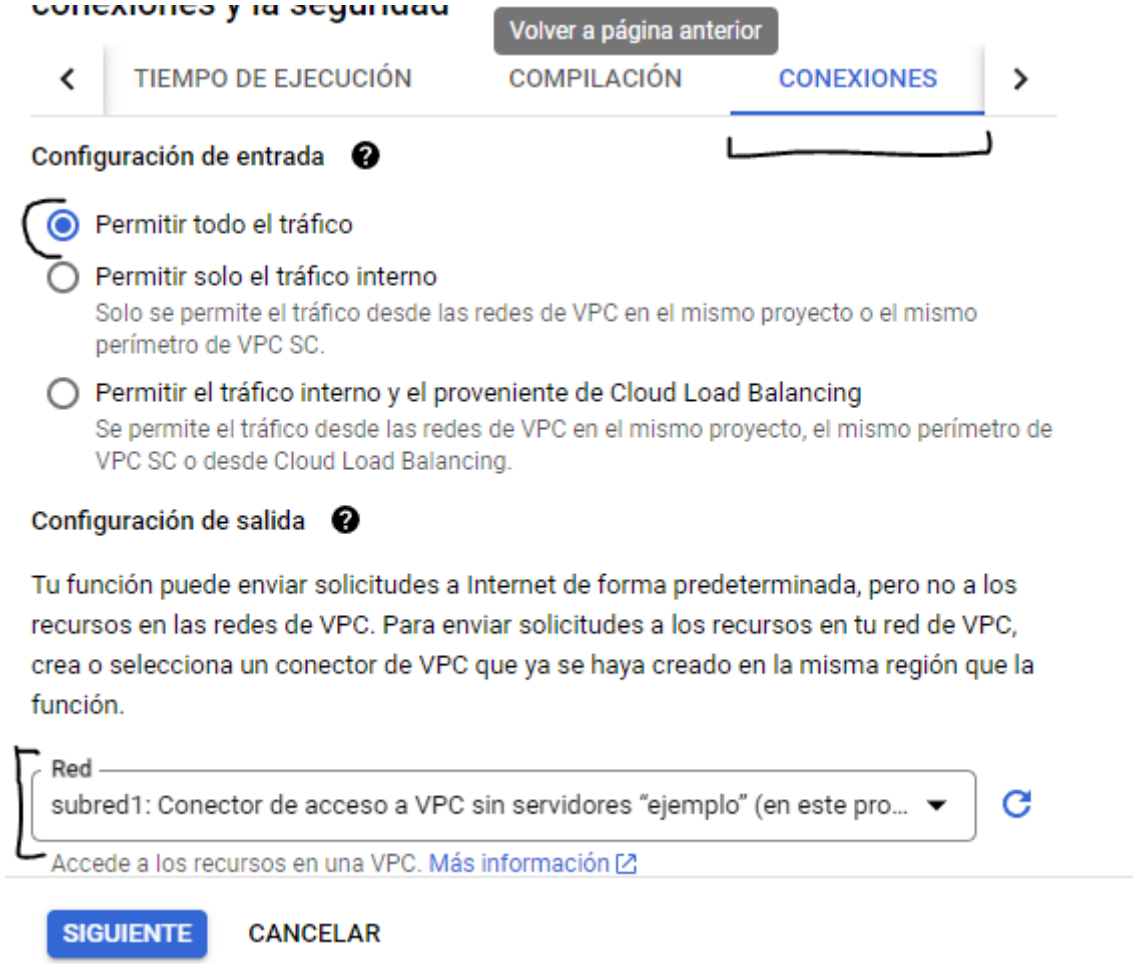


Figura 109. Anexo II, Configuración de las conexiones de Google Cloud

Importante el activador que se configurará Pub/Sub, además de las opciones de configuración de entorno se añadirá la configuración de red previamente creada en el VPC. Se pedirá el código a ejecutar cuando se cumpla el Pub, se propone en este ejemplo la conexión a la BD y el Insert de dichos datos.
Para el fichero requeriments.txt se propone:

```
functions-framework==3.*
pymysql
cloud-sql-python-connector[pymysql]
```

Para el fichero main.py se propone:

```python
import base64
import functions_framework
import pymysql

# Triggered from a message on a Cloud Pub/Sub topic.
@functions_framework.cloud_event
def hello_pubsub(cloud_event):
    # Decodificar el mensaje de Pub/Sub
    message_data = base64.b64decode(cloud_event.data["message"]["data"]).decode("utf-8")
    print(f"Mensaje recibido: {message_data}")

    # Conectar a la base de datos
    connection = pymysql.connect(
        host='34.44.191.145',
        user='jose',
        password='jose',
        database='datos'
    )

    try:
        # Usar un cursor para ejecutar la consulta
        with connection.cursor() as cursor:
            insert_query = "INSERT INTO sensores (valor, fecha) VALUES (%s, NOW())"
            cursor.execute(insert_query, (message_data,))
            connection.commit()
```

```
            print("Mensaje insertado en la base de datos.")

    finally:
        connection.close()  # Asegurarse de cerrar la conexión al final
```

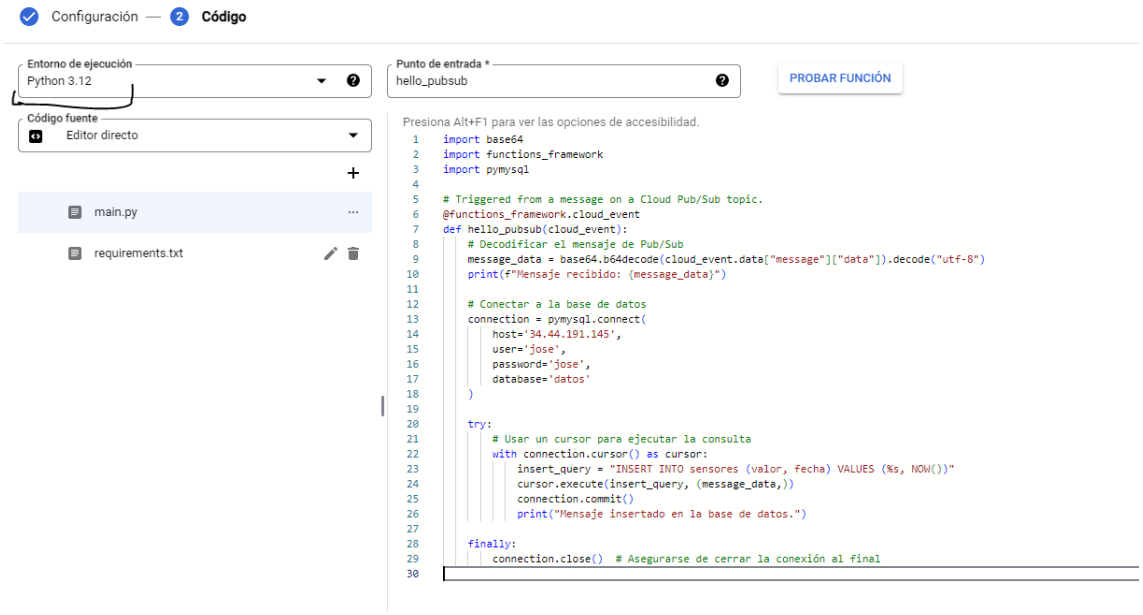


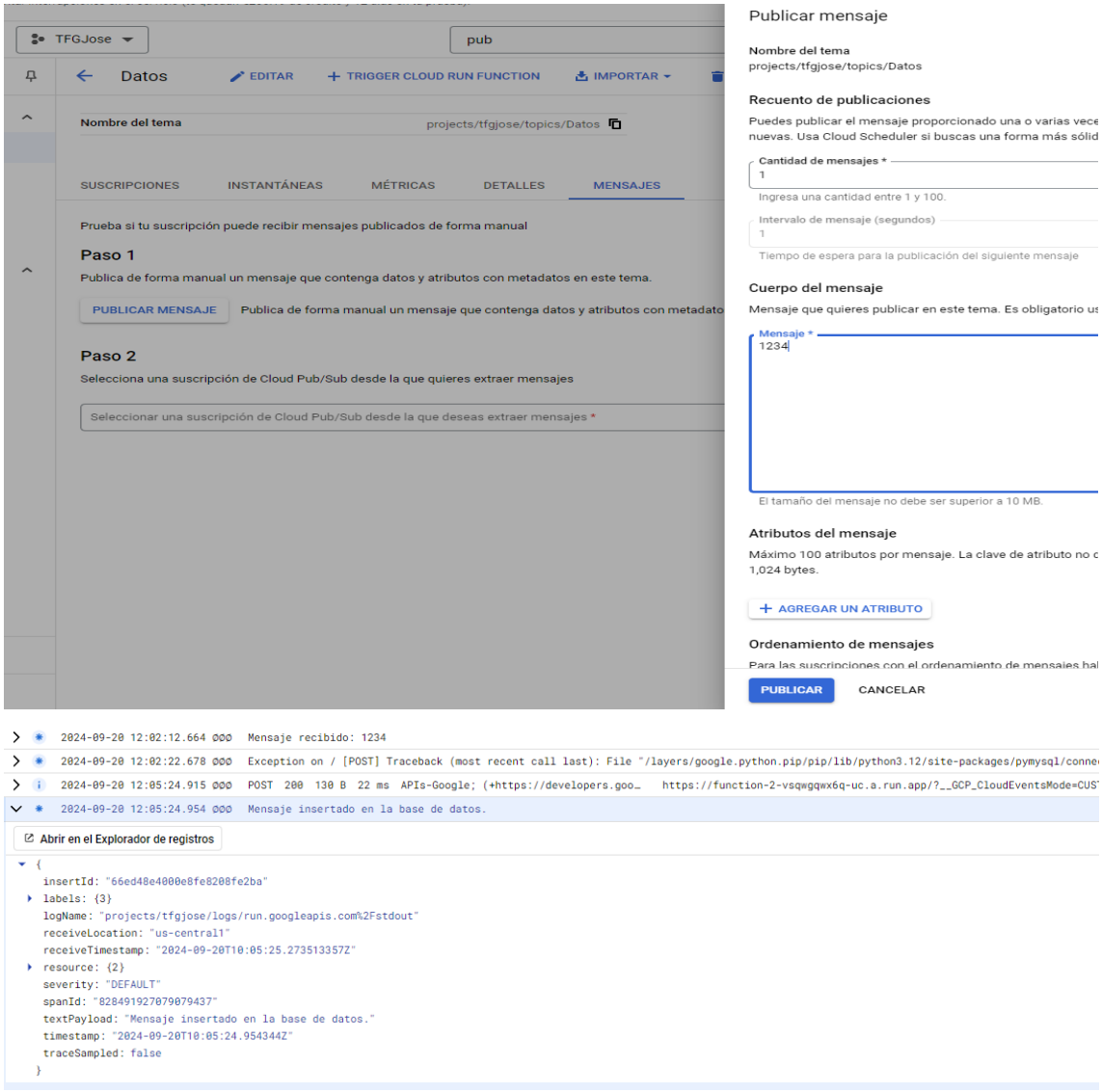
Figura 110. Anexo II, Selección del disparador

Se establecerá como lenguaje Python, luego se usará un conector propio, el cual se importará en requeriments, la configuración de este código es muy simple teniendo como, Host : dirección Ip de la base de datos, la cual se podrá ver desde SQL

Figura 111. Anexo II, Ip pública de la instancia

Usuario y contraseña que se creó anteriormente para acceder a la base de datos, y la base de datos creada, como comando se hace un insert a los campos de la misma base de datos. Para ver todos los procesos desde la pestaña Registros podremos verlo.

Una vez configurado el entorno se mostrará un ejemplo del funcionamiento. Publicando manualmente un Pub y observando la carga en la base de datos,



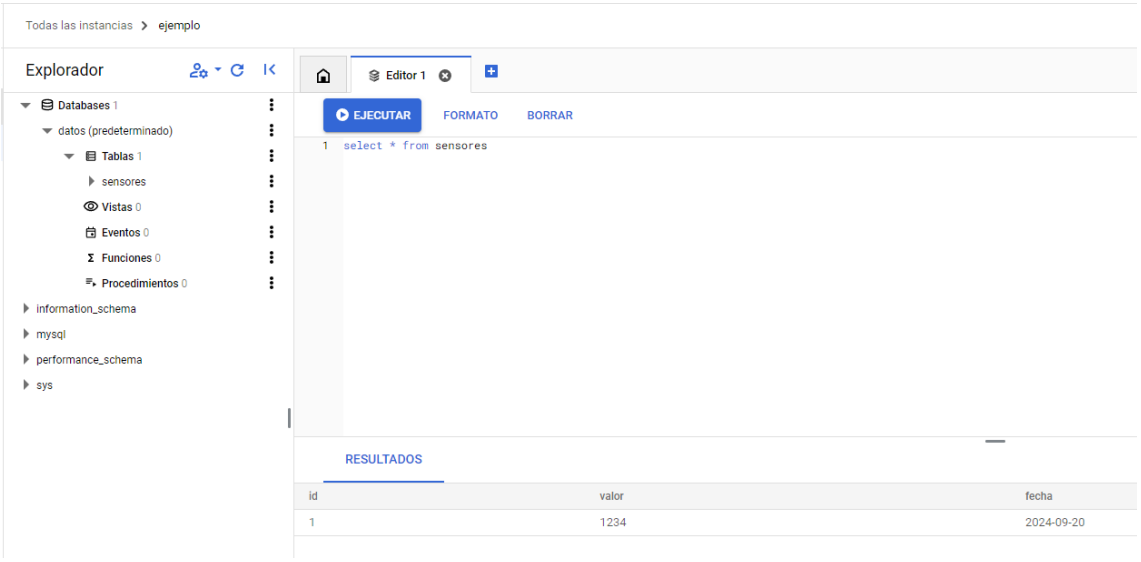


Figura 112. Anexo II, Ejemplo funcionamiento del uso