

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

ANÁLISIS DE SISTEMA DOMÓTICO PARA LA
AGRICULTURA

GRADO EN INGENIERÍA DE
SISTEMAS ELECTRÓNICOS

José Manuel Capella Rodríguez
MÁLAGA, 2022

Análisis de sistema domótico para la agricultura

Autor: José Manuel Capella Rodríguez

Tutores: Eva Gonzalez Parada y José Manuel Cano García

Departamento: Departamento de Tecnología Electrónica

Titulación: Grado en Ingeniería de Sistemas Electrónicos

Palabras clave: Agricultura, domótica, monitorización, Wi-Fi, MQTT, base de datos, IoT, microcontrolador.

Resumen

En este trabajo de fin de grado se mostrará el desarrollo de la integración de diferentes tecnologías con el fin de monitorizar distintos factores que influyen en una plantación. Para ello se hará uso de una red local en la que los dispositivos conectados se comunicarán con un servidor MQTT, el cual se alojará en el mismo dispositivo que la base de datos y la plataforma de monitorización.

La monitorización se realizará a través de un servidor web disponible tanto en ordenadores como en dispositivos móviles en el que el usuario podrá personalizar la forma de ver los datos, así como definir diferentes alarmas en función de estos.

Analysis of a domotic system for agriculture

Author: José Manuel Capella Rodríguez

Supervisors: Eva Gonzalez Parada y José Manuel Cano García

Department: Departamento de Tecnología Electrónica

Degree: Grado en Ingeniería de Sistemas Electrónicos

Keywords: Agriculture, domotics, monitoring, Wi-Fi, MQTT, database, IoT, microcontroller

Abstract

This document discusses the integration of different technologies to develop a monitoring system that allows the measurement and recording of different environmental parameters in crop areas. For this purpose, several devices will be deployed and connected to a local network, and they will exchange data by using a MQTT server, which will be hosted in the very same device as the database and the remote dashboard platform.

Data visualization can be conducted through a web interface available both on computers and mobile devices. Through this interface users are able to customize how the recorded data are represented and displayed, and it is also possible to define alarms depending on the read data values.

Contenido

Capítulo 1. Introducción	1
1.1. Contexto tecnológico	1
1.1.1. Entorno socioeconómico.....	1
1.1.2. Domótica en la agricultura.....	2
1.1.3. Empresas del sector	4
1.2. Motivación académica.....	5
1.3. Estructura de la memoria.....	5
Capítulo 2. Especificaciones del sistema	7
2.1. Modelo de contexto y casos de uso	7
2.2. Requisitos	8
Capítulo 3. Desarrollo del sistema	11
3.1. Elección de componentes	12
3.2. Diseño hardware del sistema.....	13
3.2.1. ESP32	13
3.2.2. Raspberry Pi 4	15
3.2.3. Sensor BMP280.....	16
Calibración	16
3.2.4. Sensor AHT20	17
Calibración	17
3.2.5. Sensor TEMT6000	17
Calibración	18
3.2.6. Sensor de PH	18
Calibración	19
3.2.7. Esquemático y conexiones	22
3.3. Diseño Software del prototipo.....	23

3.3.1. Programación del ESP32.....	24
3.3.2. Configuración del router	28
3.3.3. Integración de tecnologías	28
Mosquitto.....	29
Telegraf e influxDB	29
Grafana	30
Capítulo 4. Verificación y pruebas.....	31
4.1. Sistema de pruebas	31
4.2. Pruebas realizadas	31
Capítulo 5. Conclusiones y trabajo futuro.....	39
Referencias.....	41
Apéndice A. Presupuesto y horas de desarrollo	45
Apéndice B. Manual de instalación	47
Apéndice C. Manual de uso de Grafana	69

Capítulo 1. Introducción

Hoy en día, al preguntar a cualquier persona con pocos conocimientos técnicos por domótica ésta responderá haciendo alusiones a hogares inteligentes. En este Trabajo de Fin de Grado se quiere dar a saber que la domótica está presente en infinidad de sectores en la vida cotidiana. Tal es, que también lo está en la agricultura.

A lo largo de los años, la recolección de alimentos ha ido evolucionando de la mano de la tecnología y los procesos tecnológicos que hemos ido descubriendo. La agricultura ha ido evolucionando desde el uso de animales para una mayor eficiencia y rapidez de recolección hasta el uso de biotecnología para la obtención más rápida y en mayores cantidades de cultivos.

Además de procesos genéticos y de eficientes fertilizantes, la automatización de este sector es una realidad que hoy en día se sigue implementando para facilitar la monitorización de los parámetros de las plantaciones y así mejorar el control en producciones agrícolas.

1.1. Contexto tecnológico

1.1.1. Entorno socioeconómico.

Hoy en día, un 42% de la población se sustenta de la agricultura, además de ser un sector que impulsa la economía de la mayoría de los países desarrollados. Debido al significativo impacto global que las actividades humanas han tenido sobre los ecosistemas terrestres, la expansión e intensificación de la agricultura tecnológica e industrial ha incrementado la producción. La agricultura industrial

genera empleo, mejora el crecimiento económico e impulsa el sector servicios en regiones industriales[1].

Sin embargo, la intensificación de la agricultura se ha logrado a raíz de utilizar muchos fertilizantes y herbicidas que contaminan el medio ambiente y llegan a perjudicar la salud de las personas, así como impactar en la fertilidad y erosión de los suelos.

Esta situación ha hecho que empresas apuesten por métodos más sostenibles como son las granjas de interior. Estas granjas realizan un reciclaje de herbicidas y fertilizantes de forma que los desperdicios sean más leves para el medio ambiente. Además, con afán de automatizar el proceso, el cultivo se efectúa en espacios controlados, donde el crecimiento de los cultivos sea siempre igual y metódico. De esta forma, se puede estudiar el ahorro tanto de agua como de diferentes tipos de abono para hacer esta tecnología aún más sostenible, además de permitir su instalación en cualquier parte del planeta, por árida que sea. Estas granjas podrían ubicarse en cada ciudad, reduciendo así la huella de carbono producida por la continua necesidad de transportar diferentes tipos de vegetales de la zona de recogida a la zona de consumo.

La agricultura de interior se está popularizando cada vez más en todo el mundo. Los países líderes en este mercado son países asiáticos como Japón, China o Corea del Sur. Estados Unidos también apuesta por este tipo de agricultura, así como algunos países en Europa como Francia, Reino Unido o Alemania[2].

1.1.2. Domótica en la agricultura.

A raíz de la agricultura de interior empiezan a aparecer sistemas domóticos encargados del control y monitorización de los cultivos y su crecimiento. Básicamente, la domótica es la integración de múltiples dispositivos que se comunican entre sí, miden e interaccionan con sus estados externos a través de tecnología embebida que contienen[3].

En la figura 1.1 se pueden apreciar los cuatro principales componentes en las granjas inteligentes aplicando la domótica, o como se nombra en la figura, IoT.

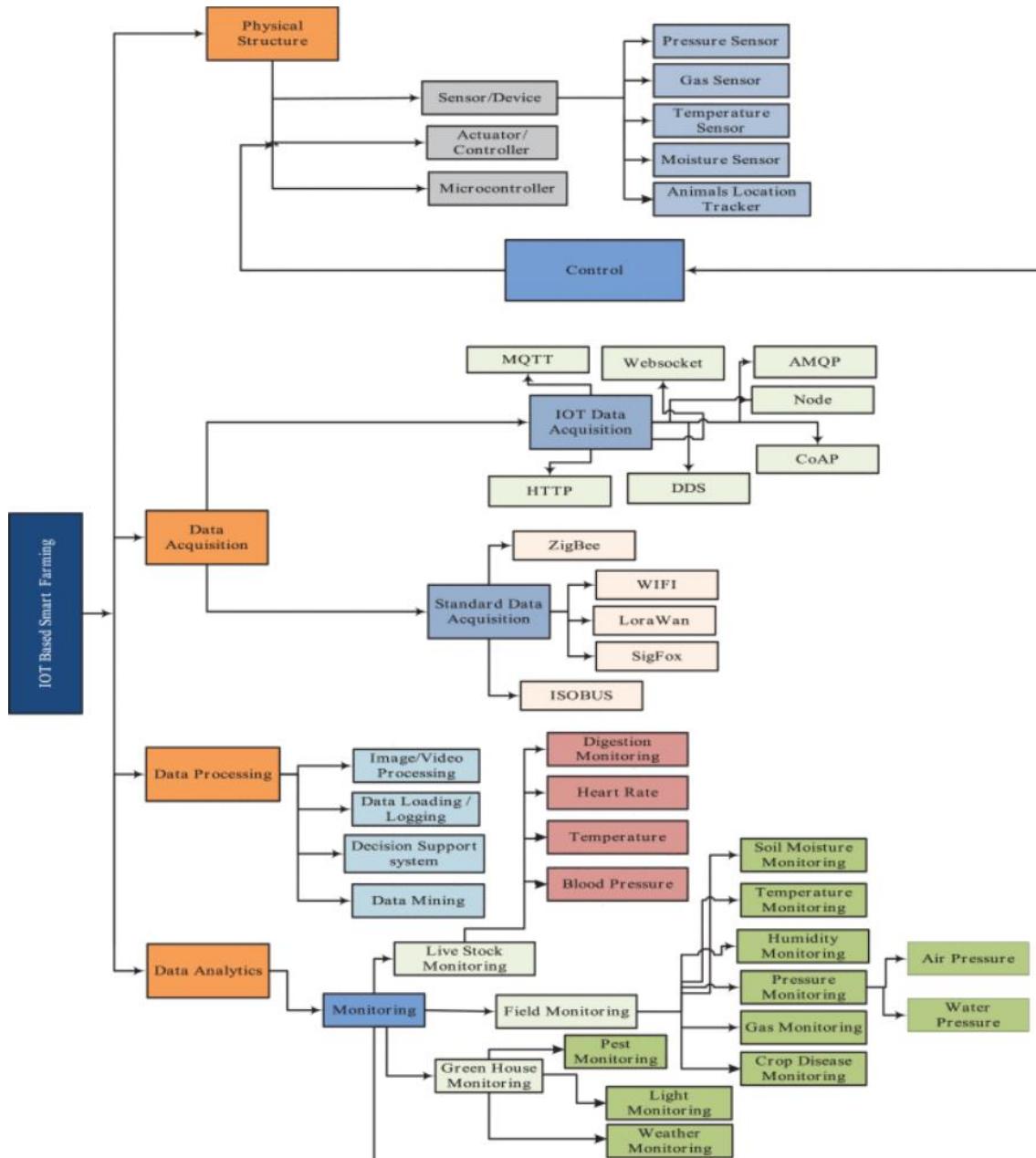


Figura 1.1. Principales componentes de la domótica en la agricultura[4]

Estos cuatro componentes se descomponen en **estructura física, recopilación de datos, procesamiento de datos y análisis de datos**.

La **estructura física** es en la que se incluyen los sensores, microcontroladores y actuadores. Este componente es el factor más importante para la precisión del cultivo. Los sensores llevan a cabo diferentes tareas como medir temperatura, humedad, agua, luminosidad, etc. Todas estas medidas se llevan a cabo por sensores controlados mediante un microcontrolador.

La **recopilación de datos** hace referencia las tecnologías inalámbricas y protocolos de comunicación disponibles en *IoT*. En función de los requerimientos y las condiciones diferentes tecnologías y protocolos podrían ser usados.

El **procesamiento de los datos** incluye diferentes funcionalidades que podrían implementarse en paralelo para integrar otros servicios.

El **análisis de datos** consiste en monitorización y control. La monitorización consiste en la visualización de diferentes parámetros que han sido medidos en la estructura física. El control hace referencia a la automatización de actuadores en función a los parámetros medidos eliminando, en alguna medida, la intervención manual en el cultivo[4].

1.1.3. Empresas del sector

Desde que esta tecnología se ha empezado a instaurar, diferentes empresas se han sumado tanto al uso de la domótica en diferentes granjas como a la creación de sistemas domóticos para la monitorización y el control de los cultivos. En cuanto a las empresas que se dedican específicamente a la creación de sistemas *IoT* para la agricultura, la mayoría tienen sede en Estados Unidos y van desde sistemas para grandes cultivos en el exterior hasta pequeños cultivos en el interior tanto en edificios como en naves industriales.

Arable[5] proporciona sistemas de recolección de datos en los que se incluyen las mediciones y monitorización a través de equipos que se pueden instalar a lo largo y ancho de las hectáreas que el cliente tenga disponibles con cultivos utilizando la tecnología *LTE* para la comunicación entre dispositivos. Esta empresa ofrece además un servicio en la nube en el que monitorizar cultivos desde cualquier parte del globo teniendo conexión a internet.

Por otro lado, *Niwa*[6] ofrece a sus clientes módulos del tamaño de una nevera. Esta empresa proporciona un módulo compacto posibilitando a cualquier persona cultivar en casa. Haciendo uso del internet de las cosas, posibilitan el control del cultivo mediante una aplicación en el teléfono móvil.

En España existe el proyecto *RIEGA-T* a manos de la empresa Extremadura *Miajadas Telecom*[7]. En este caso, este proyecto desarrolla un sistema que permite lecturas de distintos valores analógicos medioambientales, programaciones de riego mediante actuadores y sistemas de alarmas entre

otros. Utilizando tecnologías como *Wi-Fi*, *LTE* y *LoRaWAN* para la recopilación de datos y aplicaciones móviles para el posterior análisis de dichos datos.

1.2. Motivación académica

La motivación de este proyecto es comprobar que es posible la integración de diferentes tecnologías de software libre como servidores y bases de datos, y de dispositivos de desarrollo *low cost* como sensores, microcontroladores y microprocesadores para monitorizar una plantación a gran escala y en la que los parámetros de la plantación y la lectura de los datos se realice de forma sencilla para los usuarios que utilicen la herramienta.

Se quiere realizar un pequeño módulo escalable en el que se realizarán diferentes medidas del entorno para la posterior recopilación y análisis de los datos recogidos. El objetivo es la monitorización en el que un usuario podría consultar en todo momento como se encuentra el cultivo.

1.3. Estructura de la memoria

En las posteriores secciones de esta memoria se desarrollarán los aspectos técnicos relativos a este Trabajo Fin de Grado. La memoria se divide en capítulos, y estos a su vez en apartados, descritos a continuación: En el capítulo 1 se ha introducido la domótica en la agricultura y el estado del arte, así como la motivación académica. En el capítulo 2 se verán las especificaciones del sistema junto a los requisitos y pruebas que deberá pasar. El desarrollo para que se cumplan dichos requisitos se verá en el capítulo 3, donde se introducen los dispositivos que se van a utilizar. En el capítulo 4 se muestran los resultados de las pruebas realizadas para verificar el correcto funcionamiento del sistema. En el capítulo 5 se desarrollan las conclusiones personales y posibles mejoras futuras. Al final del documento se añaden 3 apéndices: apéndice A, apéndice B y apéndice C. En el primer apéndice se desglosará el presupuesto y las horas de desarrollo. Le seguirán los apéndices B y C los cuales contienen manuales de instalación y uso respectivamente.

Capítulo 2. Especificaciones del sistema

En la agricultura se pueden encontrar diferentes factores externos que se pueden monitorizar gracias a la domótica. Algunos de los más importantes son: la temperatura ambiente, la humedad relativa del entorno, el pH del agua de riego y la luminosidad. Para poder gestionar una monitorización sencilla se puede hacer uso de la integración de diferentes tecnologías como son bases de datos, protocolos de comunicación y software que permita la visualización de series de tiempo.

Este trabajo consistirá en el estudio de la integración de diferentes tecnologías para realizar un sistema que realice la monitorización de diferentes parámetros a un bajo coste. Junto al sistema se proporcionarán dos manuales: Manual de instalación del sistema y manual de uso para el cliente.

2.1. Modelo de contexto y casos de uso

El número de actores y factores externos que el sistema tendrá en cuenta, se pueden apreciar en el modelo de contexto en la figura 2.1.

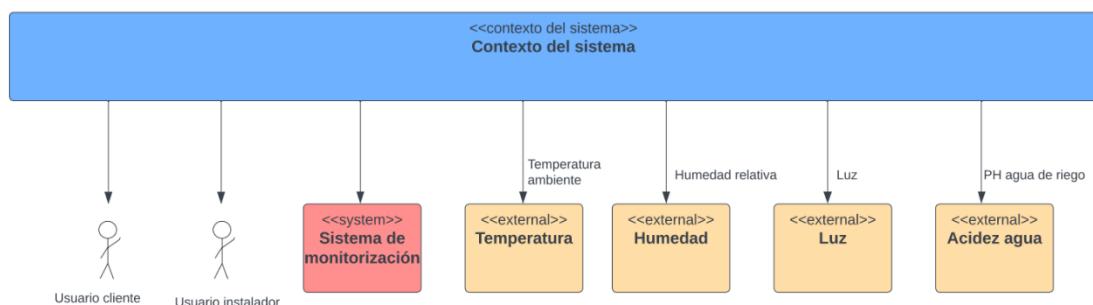


Figura 2.1. Modelo de contexto

Los factores externos que se tendrán en cuenta son los rasgos climáticos condicionados por el ambiente en el que se instale además de dos actores: usuario cliente y usuario administrador. Estos dos usuarios tendrán diferentes roles que se pueden ver en más detalle en el diagrama de casos de uso mostrado en la figura 2.2.

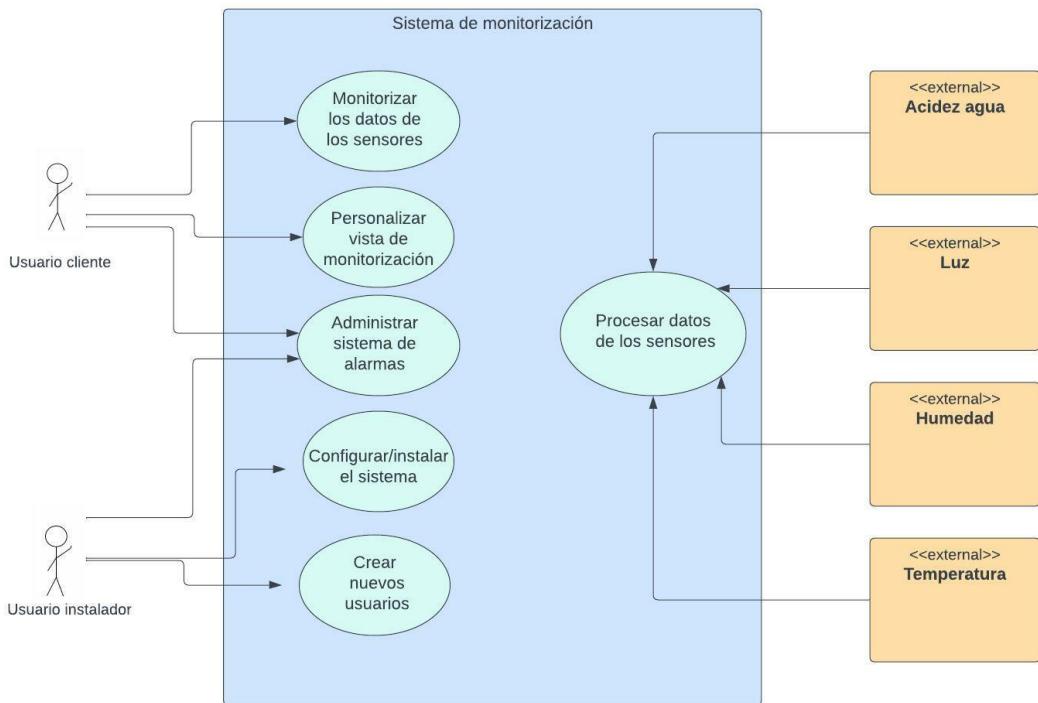


Figura 2.2. Diagrama de casos de uso

En la figura 2.2 el usuario cliente puede monitorizar todos los datos de los sensores procesados por el sistema. Además de poder añadir alarmas y personalizar la forma de ver los datos. En cuanto al instalador, además de configurar el sistema y crear usuarios, también puede administrar el sistema de alarmas.

2.2. Requisitos

Se desea hacer una integración de diversas tecnologías para monitorizar una pequeña superficie de un cultivo en interior. El sistema se hará de forma escalable, pudiendo añadir más motas en caso de querer expandirlo. Además, se diseñará con una estructura centralizada en la que un mismo servidor hospedará una base de datos que permita la posterior visualización de los datos

en cualquier lugar. En el prototipo desarrollado en este trabajo se monitorizarán temperatura, humedad y luz a los que está expuesto en el cultivo, así como el nivel de PH del agua que regará continuamente dichos cultivos.

La monitorización se hará desde un ordenador o dispositivo móvil con conexión a internet. De igual manera se dispondrá de alarmas que avisen a un teléfono móvil en caso de que algún parámetro resulte anómalo.

La definición de los requisitos a cumplir se puede observar en la tabla 2.1.

Id.	Nombre	Descripción	Tipo	Prioridad
R1	Lectura temperatura	de El sistema debe leer la temperatura presente en el cultivo.	Funcional	1
R2	Lectura de humedad	El sistema debe leer la humedad presente en el cultivo.	Funcional	1
R3	Lecturas iluminación	de El sistema debe leer la iluminación presente en el cultivo.	Funcional	1
R4	Lectura de PH	El sistema debe leer el nivel de PH en el tanque de riego.	Funcional	1
R5	Conexión a red local	El sistema debe conectarse a una red local predefinida.	Funcional	1
R6	Envío de lecturas sobre MQTT	El sistema debe publicar las lecturas a un servidor MQTT.	Funcional	1
R7	Empaquetado	El sistema debe publicar las lecturas comprimidas en un formato compacto.	Funcional	1
R8	Base de datos	El sistema debe guardar las lecturas en una base de datos.	Funcional	1
R9	Monitorización	El sistema debe mostrar los datos guardados en la base de datos.	Funcional	1
R10	Alarmas	El sistema debe alertar al usuario en caso de leer datos anómalos.	Funcional	1
R11	Manuales	El sistema debe entregarse junto a un manual de instalación y un manual de uso.	No Funcional	1
R12	Escalable	El sistema debe ser escalable. Estar centralizado.	Funcional	2
R13	Presupuesto	El sistema debe usar software de uso libre además de componerse por sensores de bajo coste.	No funcional.	2
R14	Acceso por internet	El sistema debe permitir el acceso al servicio web de monitorización a cualquier dispositivo con internet.	Funcional	1

Tabla 2.1. Tabla de descripción de requisitos

De acuerdo con los requisitos citados y la funcionalidad que se le quiere dar al sistema, se puede ver en la figura 2.3 la representación del diagrama de bloques general del sistema.

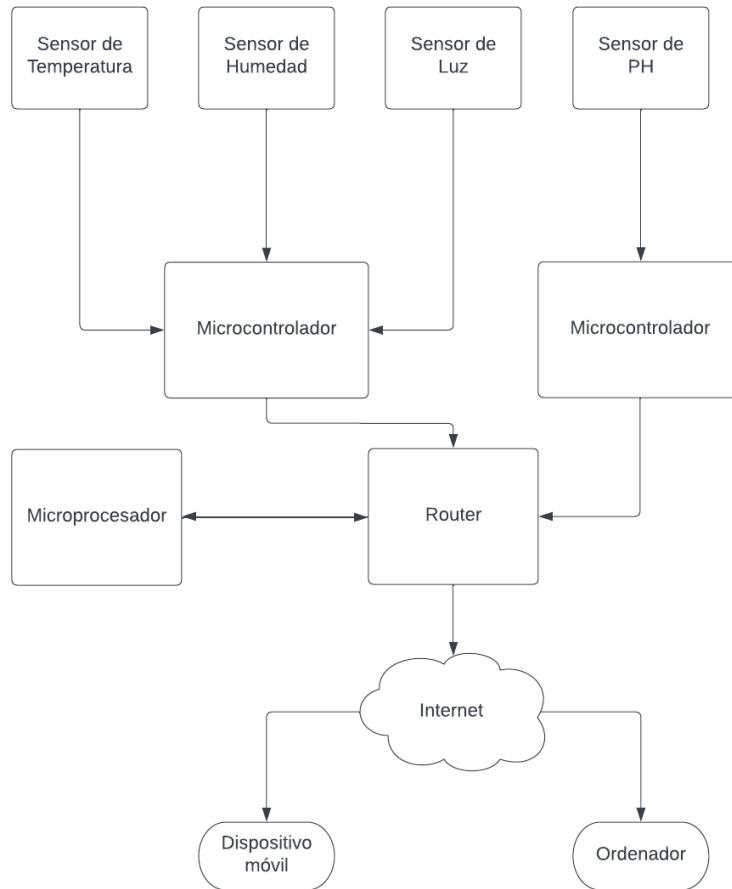


Figura 2.3. Diagrama de bloques general del sistema

Capítulo 3. Desarrollo del sistema

En el diagrama mostrado en la figura 3.1 se representa la implementación del prototipo que da respuesta al sistema de bloques general mostrado en la figura 2.3 del capítulo 2. En este se pueden ver los sensores utilizados, así como las distintas tecnologías integradas.

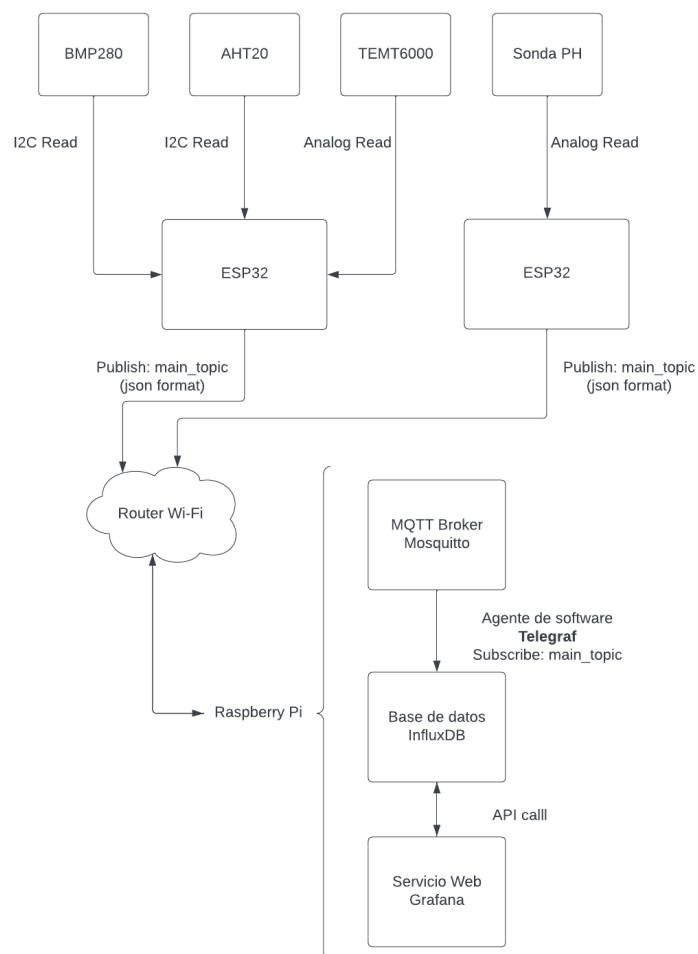


Figura 3.1. Diagrama de bloques

3.1. Elección de componentes

Los bloques mostrados en la figura 3.1 hacen referencia a los distintos dispositivos del sistema, listados en las tablas 3.1 y 3.2:

Dispositivo	Descripción
ESP32	Microcontrolador al que se conectan los diferentes sensores. Se conecta a la red local y envía los datos a través de MQTT.
Raspberry Pi 4	Microprocesador donde se instalan los diferentes servicios del sistema. Recibe datos por MQTT, los guarda en la base de datos y los muestra a través del servidor web.
BMP280	Sensor de temperatura y presión atmosférica con interfaz I2C.
AHT20	Sensor de temperatura y humedad con interfaz I2C.
TEMT6000	Sensor de luz ambiental con salida analógica. Salida GPIO con interfaz ESP32.
Sensor de PH	Sensor de PH para líquidos con salida analógica. Salida GPIO con interfaz ESP32.

Tabla 3.1. Hardware usado en el sistema

Componente	Descripción
Mosquitto	Bróker MQTT para la transferencia de datos.
Eclipse	
Telegraf	Agente software que gestiona la entrada y salida de información a la base de datos.
InfluxDB	Gestor de bases de datos.
Grafana	Plataforma donde visualizar los datos recogidos por la base de datos.

Tabla 3.2. Componentes software usados en el sistema

Los sensores, motas y la Raspberry Pi se enviarán entre sí información relevante correspondiente a las lecturas de los parámetros del sistema: temperatura,

humedad, ph y nivel de luz. Esta información será enviada desde las motas a la Raspberry Pi a través del protocolo MQTT. Para ello, se utilizará la conexión Wi-Fi tanto de las motas como de la Raspberry Pi. Después, el bróker instalado en esta última procesará los datos y serán guardados en la base de datos administrada con *InfluxDB*. Finalmente, a través de *Grafana*, habilitaremos un servidor web en el que monitorizar la información de la base de datos.

3.2. Diseño hardware del sistema

Los componentes hardware del sistema listados en la tabla 3.1 de este capítulo se definen en este apartado describiendo su funcionalidad y la conexión con los distintos dispositivos.

3.2.1. ESP32

Para la conexión de los diferentes sensores usados se ha usado la placa de desarrollo ESP32-DevKitC. Se ha elegido este *System On a Chip* por las prestaciones que presenta y su bajo coste. Este microcontrolador tiene, entre otras, las siguientes características[8]:

- Procesador de dos núcleos de 32 bits *LX6* con frecuencia de reloj máxima de 240 MHz.
- 520 KB de memoria *SRAM* y 448 KB de memoria *ROM*.
- Antena Wi-Fi integrada: 802.11 b/g/n con conexión de hasta 150 Mbps.
- Hasta 34 pines *GPIO* programables.
- Conectividad serie mediante interfaces *UART*, *SPI* e *I2C* entre otros.

Los pines de la placa de desarrollo se pueden ver en detalle en la figura 3.3.

ESP32-DevKitC

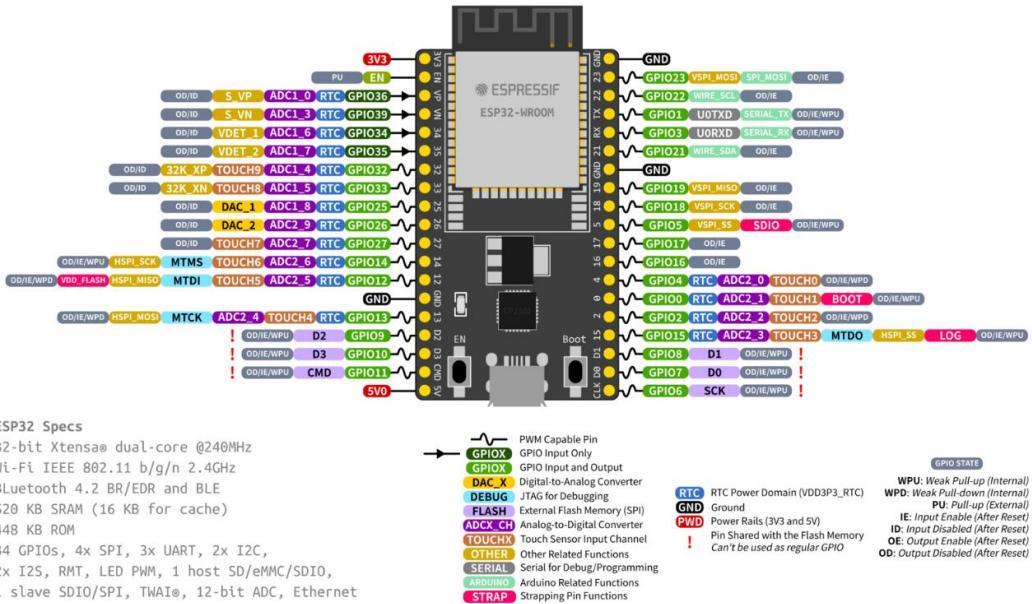


Figura 3.2. Pines de la placa de desarrollo ESP32[9]

En esta mota, además del conexionado de alimentación de los sensores, se hará uso de los pines GPIO33, GPIO21 y GPIO22:

GPIO 33. Conversor analógico-digital (ADC). Se ha elegido este pin ya que el ESP32 tendrá habilitado la antena Wi-Fi, lo cual -por conexiones internas- genera ruido que invalida al resto de entradas analógicas. Esta restricción es debida a que el controlador Wi-Fi del dispositivo utiliza este conversor.

Este conversor junto al *framework* de Arduino puede medir voltajes desde 0 a 3.3 voltios. Este voltaje tiene una resolución de 12 bits, o lo que es lo mismo, cada valor de tensión leído se mapea con valores de hasta 4095. Es decir, teóricamente 0 voltios leídos equivaldrían a 0 y 3.3 voltios equivaldrían a 4095. Sin embargo, esto no es del todo cierto ya que el comportamiento de este conversor no es del todo lineal debido a impedancias internas y demás tolerancias en los componentes. En la figura 3.4 se puede ver el comportamiento entre el voltaje real y los datos leídos por el microcontrolador.

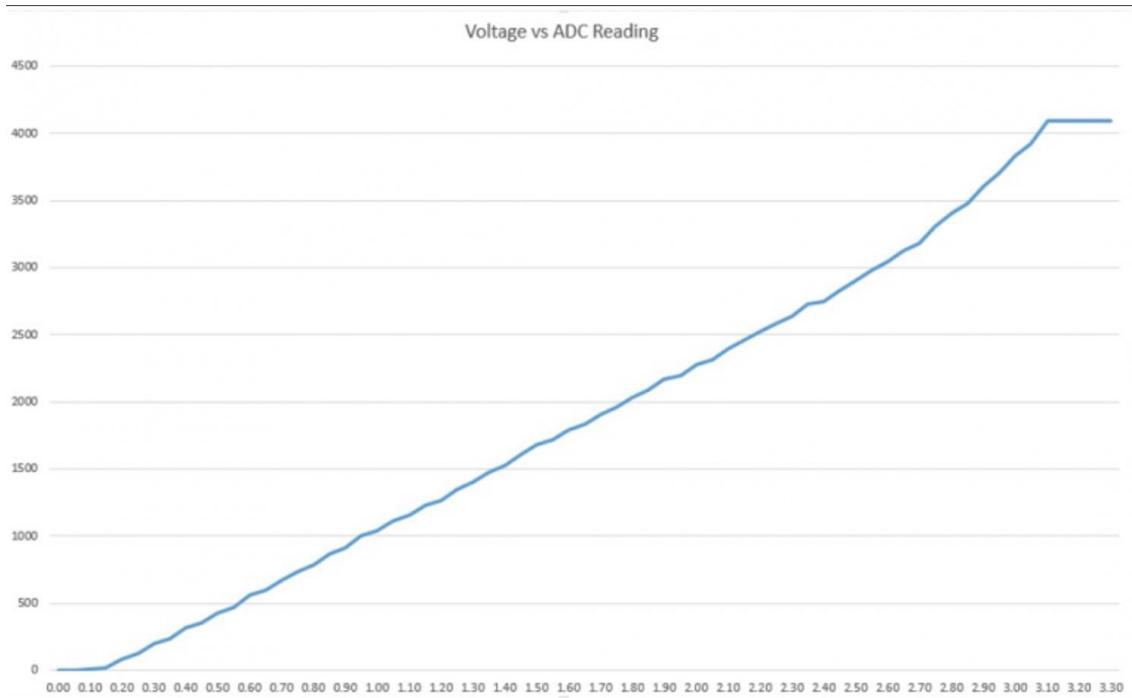


Figura 3.3. Lectura de voltaje (eje X) frente a lectura del conversor analógico-digital (eje Y)[10]

Como se puede apreciar, la realidad es que nuestro sensor medirá una tensión de entre 0.1 a 3.2 voltios. Esto se deberá tener en cuenta a la hora de calibrar el sensor o sensores que se conecten al conversor analógico-digital.

GPIO 21 y 22. Protocolo de comunicación I2C. A través de estos pines se conectarán en paralelo distintos sensores. En este puerto se pueden interconectar diferentes sensores ya que la moto puede diferenciar entre los datos recibidos de uno u otro gracias al protocolo de comunicación utilizado.

3.2.2. Raspberry Pi 4

Para satisfacer el requisito de monitorizar los parámetros recogidos por la red de sensores se ha elegido utilizar una Raspberry Pi 4.

Este dispositivo cuenta con una tarjeta de red integrada que nos permitirá conectarla a la red wifi del entorno donde vamos a trabajar. Incorpora una tarjeta SD en la que se puede cargar el sistema operativo y el resto de los servicios, además de poder guardar todos los datos leídos.



Figura 3.4. Raspberry Pi 4[11]

Las dos motas utilizadas son dos ESP32 las cuales tienen integradas antenas wifi, con esto podremos conectarlas a la red local junto a la Raspberry Pi 4.

3.2.3. Sensor BMP280

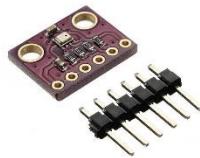


Figura 3.5. Sensor BMP280[12]

El *BMP280* es un sensor capaz de medir la temperatura y la presión atmosférica del lugar donde se encuentre. Una ventaja de este sensor sobre otros es que devuelve la temperatura en grados Celsius, que será la unidad que utilizaremos para medir y mostrar la temperatura. Este sensor de precisión está fabricado por Bosch[13].

Para el uso de este sensor utilizaremos la librería de código abierto “*BMP280_DEV*”[14].

Calibración

Este dispositivo se comunicará a través de *I2C* con nuestra mota devolviendo siempre un valor ya en grados Celsius gracias a su librería. De esta manera solo se podrá calibrar mediante software sumándole o restándole el valor decimal necesario para obtener el valor real de temperatura. La calibración se ha

realizado utilizando como patrón un termómetro ambiental digital de mayor precisión.

3.2.4. Sensor AHT20

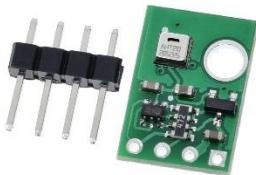


Figura 3.6. Sensor AHT20[15]

El *AHT20* es un sensor de temperatura y humedad. Este sensor será solo utilizado como sensor de humedad eliminando la funcionalidad de temperatura ya que no tiene tanta precisión como el *BMP20* anteriormente citado[16].

Para el uso de este sensor utilizaremos la librería de código abierto “*Seeed_Arduino_AHT20*”, creada por *Seed studio*[17].

Calibración

Este dispositivo se comunicará a través de *I2C* con nuestra moto devolviendo siempre un valor entre 0.00 y 1.00. Multiplicando este valor por 100 ya tendríamos el valor en porcentaje de humedad relativa del ambiente. Para la comprobación de su correcto funcionamiento se ha situado cerca un vaporizador se ha visto como satura en su valor máximo. Retirando el vaporizador y con un secador hacia el sensor se ha comprobado como alcanza valores mínimos.

3.2.5. Sensor TEMT6000



Figura 3.7. Sensor TEMT6000[18]

El TEMT6000 es un sensor de luz ambiente. Este sensor funciona como un fototransistor, a mayor luz incidente, mayor tensión en el pin de salida. Para el uso de este sensor no utilizaremos ningún tipo de librería extra, se utilizará el conversor analógico-digital del propio ESP32.

Calibración

Con un multímetro, para tener más precisión, podemos encontrar diferentes datos de salida. Se han hecho diferentes medidas: En total oscuridad y con un led regulable a una distancia fija del sensor. Los resultados han sido los siguientes:

-Valor de tensión a la salida con total oscuridad: 0 V (un valor de 0 en el conversor analógico-digital de la mota)

-Valor de tensión a la salida con luz ambiente en interior: 0.2 V(un valor de 110 en el conversor analógico-digital de la mota)

-Valor de tensión a la salida con led a máxima potencia: 1.0 V(un valor de 1100 en el conversor analógico-digital de la mota)

El conversor analógico-digital del ESP32 no distinguirá de una entrada analógica de entre 0.0 V y 0.1 V, este caso no perjudicará al diseño del sistema ya se trata de un margen de error bastante leve en cuanto a los resultados que se quieren obtener.

3.2.6. Sensor de PH



Figura 3.8. Sonda-electrodo y circuito integrado[19]

Este sensor se compone de dos dispositivos: La sonda-electrodo y el circuito integrado. Ambos se conectan a través de un conector BNC. Dependiendo de la lectura de la sonda, se generará una señal analógica mayor o menor. Debido a su bajo coste, el vendedor no proporciona hoja de datos ni fabricante del dispositivo. Debido a esto, se ha tenido que buscar información de los chips en el circuito integrado tanto para identificar el significado de dichos pines como para la calibración de este.

El valor del PH va desde 0 (acídico) a 14 (base). En nuestro caso vamos a calibrar nuestro sensor para que mida con mayor precisión valores desde 6.0 a 8.0, que son los valores en los que debe moverse el agua de riego.

El PH es una medida del grado de acidez o alcalinidad de una solución. Sin entrar en detalles, la sonda generará una señal analógica de salida en función del nivel medido en un líquido. El circuito integrado que se muestra en la figura 3.10 acondiciona dicha señal para poder usarla en un microcontrolador.

Calibración

Se ha comprobado con un multímetro que el circuito integrado de la figura 3.10 genera una señal de 2.5 Voltios con una lectura de PH 7. Dependiendo de la lectura realizada, obtendremos hasta 5 Voltios de salida con un PH nulo (0) y una salida de 0 Voltios con una lectura de PH máximo (14)[20].

Para la calibración hemos alimentado el circuito integrado con el ESP32. Después se ha procedido a hacer diferentes lecturas con líquidos encontrados en casa, todos ellos rondando los valores teóricos de lectura de entre 6 y 8 de PH. Para tener un patrón para saber qué valores de PH tienen ciertos líquidos se han utilizado valores provenientes de la etiqueta de cada producto y, en otros casos, se ha usado un kit de análisis de PH para piscinas.

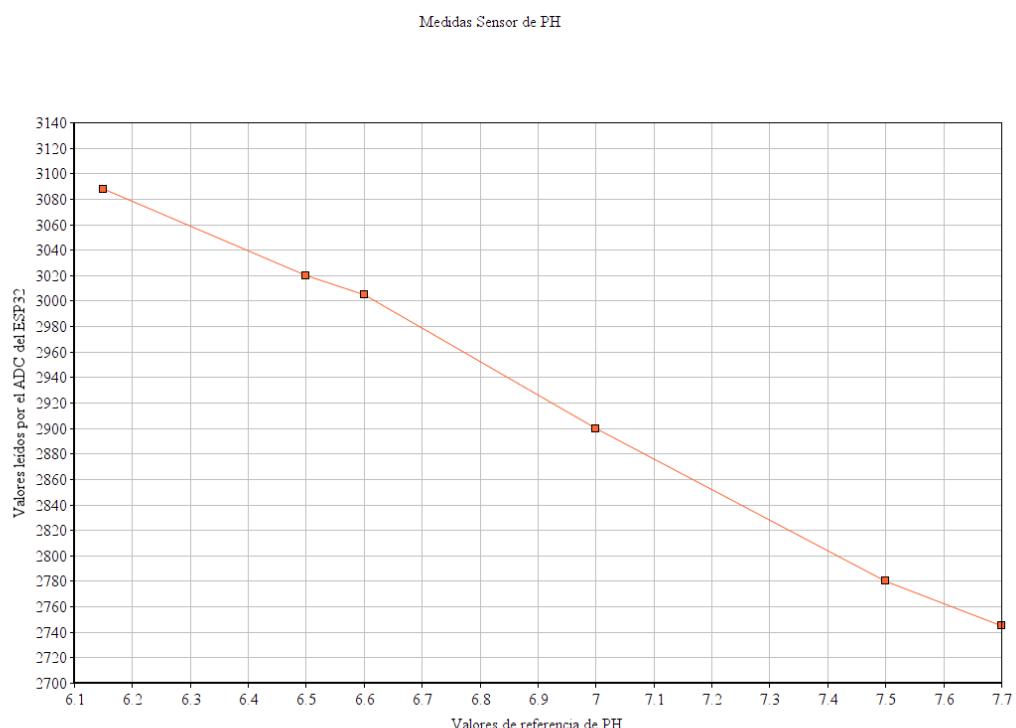
Las diferentes medidas se pueden apreciar en la tabla 3.1:

Medida	Resultado multímetro (Voltios)	Resultado del ADC del ESP32	Valor de referencia de PH
Agua Piscina 1	2.592	3088	6.15
Agua Piscina 2	2.554	3020	6.5

Leche	2.573	3005	6.6
Cortocircuito	2.499	2900	7
Agua de aire acondicionado	2.411	2780	7.5
Agua embotellada	2.385	2745	7.7

Tabla 3.3. Valores medidas del sensor de PH

Los resultados obtenidos se pueden mostrar en la siguiente gráfica para su mejor comprensión:

**Figura 3.9. Medidas tomadas con el Sensor de PH**

Como se puede apreciar, los datos leídos forman aproximadamente una recta, la cual nos puede servir para saber cómo se comportarán (grosso modo) las lecturas desde valores desde 6 a 8 de PH. Para ello se ha calculado la pendiente de la recta en ambos tramos con la fórmula del punto pendiente:

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

Ecuación 3.1. Ecuación del punto pendiente

Con los resultados obtenidos hemos podido obtener dos puntos teóricos en un valor de PH 6 y en un valor de PH 8 utilizando esa misma ecuación.

Se aproximado las medidas de forma lineal resultando los siguientes valores:

- 3125 en la lectura del ADC de la mota para un valor de 6 de PH.
- 2670 en la lectura del ADC de la mota para un valor de 8 de PH.

Este circuito integrado tiene los pines To, Do, Po, G y V+ además de dos potenciómetros como se puede ver en la siguiente figura.



Figura 3.10. Circuito integrado del sensor de pH[21]

Al que nos referiremos como potenciómetro 1. Es el que está a la izquierda en la figura.

Al que nos referiremos como potenciómetro 2. El que está a la derecha en la figura, más cerca del conector BNC.

To. Señal para la temperatura medida. Quedará al aire.

Do. Se puede ajustar el potenciómetro 1 para que al alcanzar un valor ajustable de pH se genere una señal de nivel alto o bajo en este pin.

Po. Salida analógica en función del pH medido por la sonda-electrodo. El potenciómetro 2 se utiliza para calibrar la señal de salida de este pin.

G. Tierra. Existen dos tierras en caso de que el líquido a medir tenga una toma a tierra distinta que nuestra mota. En este diseño, ambas irán conectadas entre sí.

V+. Alimentación (5 V). Este pin irá conectado a la alimentación de 5 V del ESP32.

En el diseño solo se utilizan las tierras G, la alimentación V+, la salida analógica Po y el potenciómetro 2.

3.2.7. Esquemático y conexiones

El conexionado de las motas con los diferentes sensores que componen el sistema se representa en las figuras 3.11 y 3.12 mostradas a continuación:

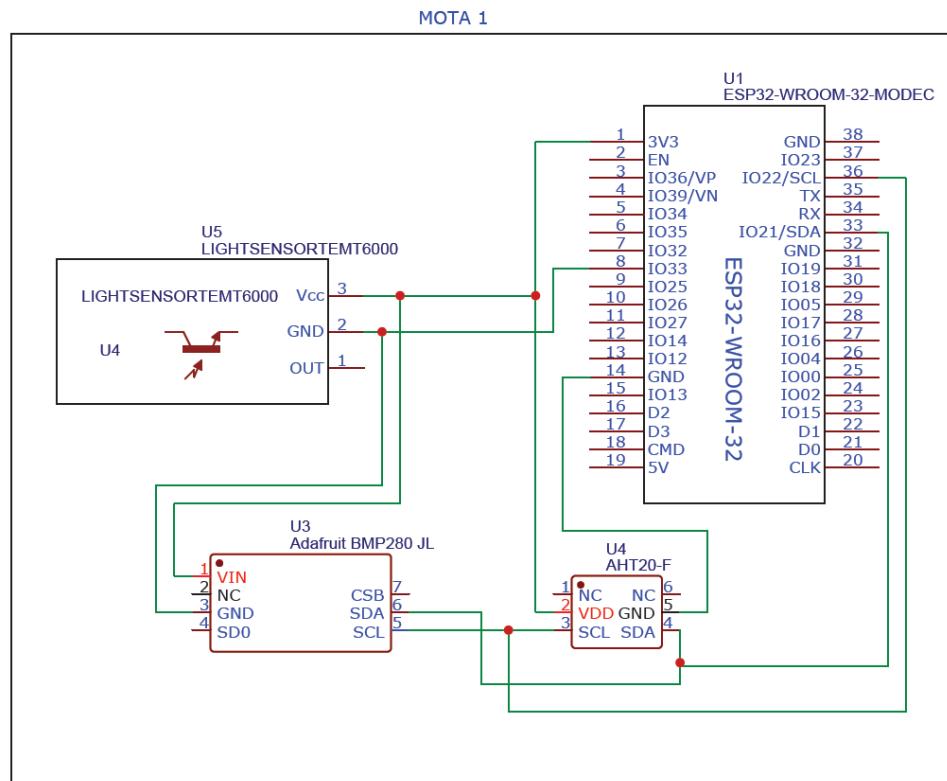


Figura 3.11. Esquemático de conexionado de mota 1

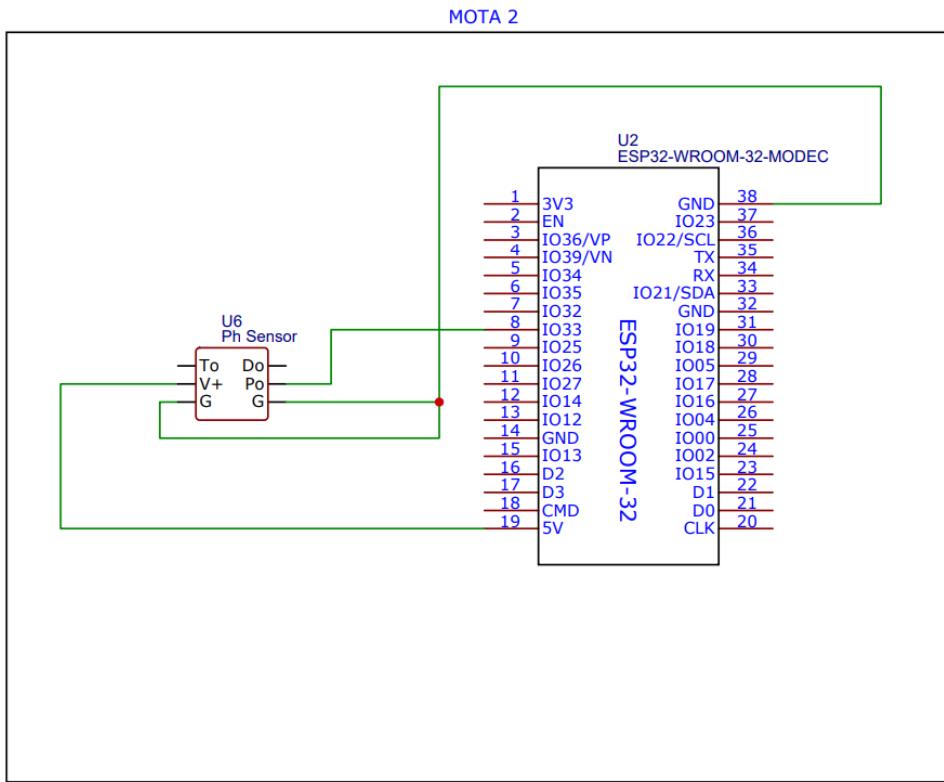


Figura 3.12. Esquemático de conexionado de mota 2

Como se puede ver en las figuras 3.11 y 3.12, los sensores conectados a la Mota 1 están alimentados a través del pin 1, el cual tiene voltaje suficiente para ello (3.3 V). En cuanto a la alimentación del microcontrolador de la sonda de PH, éste se alimenta a través del pin 19 del ESP32, que tiene una salida de 5 V. En el diseño se han elegido dos motas. Esto es porque una estará localizada en el tanque de riego para medir únicamente su PH y la otra en el cultivo midiendo el resto de los parámetros.

3.3. Diseño Software del prototipo

Por agilidad en la programación, así como para futura escalabilidad y diferentes cambios en el sistema, se ha hecho uso exclusivo de C++ y *framework* de Arduino y sus librerías para la programar el ESP32.

Para este proyecto se ha usado *Visual Studio Code*. Su instalación y configuración está detallada en el manual adjunto en el anexo de este documento.

El código y los archivos de configuración de los diferentes sistemas usados, está accesible en el siguiente repositorio de *Github*:
https://github.com/JoCapella/Domotica_Agricultura.

3.3.1. Programación del ESP32

El ESP32 se comunica con el servidor alojado en la *Raspberry Pi 4* a través de MQTT[22]. Se ha utilizado el bróker *mosquitto*[23] mandando paquetes en formato json[24] para su mejor compresión.

El diagrama de flujo del código instalado en la mota se puede ver en la figura 3.13 mostrada a continuación.

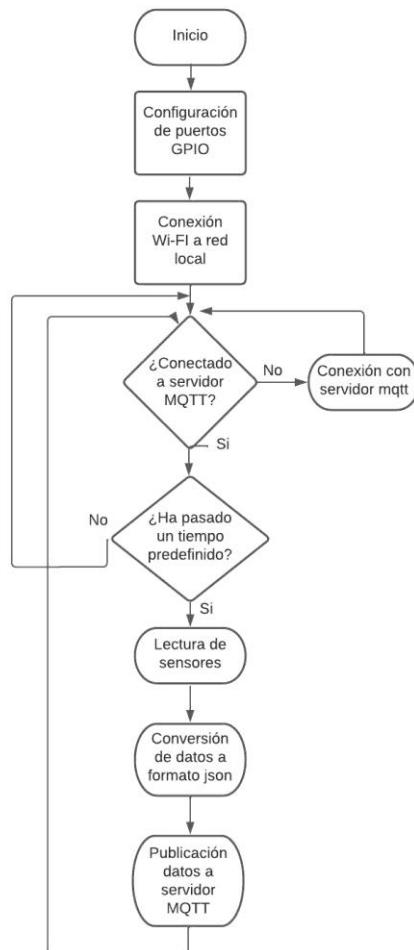


Figura 3.13. Flujograma del proceso corriendo en la mota

La función *setup_wifi()* que se muestra en la figura 3.14 permite conectar el *ESP32* a la red Wi-Fi.

```

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

    WiFi.begin(ssid, password);

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

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

```

Figura 3.14. Función setup_wifi()

Una vez conectado a la red Wi-Fi, se conecta (y reconecta en caso de fallo) al bróker MQTT con la función *mqttConection()* definida como se muestra en la figura 3.15.

```

void mqttConection()
{
    while (!client.connected())
    {
        Serial.print("Intento de conexion MQTT...");
        // Attempt to connect
        if (client.connect(clientID, mqtt_username, mqtt_password))
        {
            Serial.println("connected");
        }
        else
        {
            //Error por el cual no conecta
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" Probar de nuevo en 5 segundos");

            delay(5000);
        }
    }
}

```

Figura 3.15. Función mqttConection()

Para la lectura del sensor de temperatura *BMP280* se ha usado la función *print_Temp()* como se muestra en la figura 3.16. Este sensor también puede hacer lecturas de valores de presión y altitud, aunque no se han utilizado. La función *getCurrentTemperature()* está definida en la librería del sensor.

```
void print_Temp()
{
    bmp.getCurrentTemperature(temperature);
}
```

Figura 3.16. Función print_Temp()

Para la lectura del sensor de humedad y temperatura *AHT20* se ha utilizado la función *print_Hum()* como se puede ver en la figura 3.17. El método *getHumidity()* está definido en la librería del sensor.

```
void print_Hum()
{
    ATH.getHumidity(&humid);
    humi *= 100;
}
```

Figura 3.17. Función print_Hum()

En la figura 3.18 se puede ver la implementación de la función *printLight()* para la lectura del sensor *TEMT6000*.

```
void printLight()
{
    LightValue = analogRead(PinLight);
    percentLight = map(LightValue, valueMin, valueMax, 0, 100);
}
```

Figura 3.18. Función printLight()

Para la lectura de la sonda de PH se ha implementado la función *printpH()* que hace uso de la función *fmap()*, modificación de *map()* de la librería *WMath.cpp*[25] para que acepte valores con decimales en la entrada. El código usado se muestra en la figura 3.19

```
//-----FUNCTIONS-----
float fmap(float x, float in_min, float in_max, float out_min, float out_max)
{
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

void printPh()
{
    PinPhValue = analogRead(PinPh);
    Serial.println(PinPhValue);
    Po = fmap(PinPhValue, sensorMax, sensorMin, 6.00, 8.00);
    Serial.print("\tPH: ");
    Serial.print(Po);
    Serial.println("");
}
```

Figura 3.19. Funciones printPh() y fmap()

Para concluir, se han añadido estas funciones en un bucle y, haciendo uso de la función *millis()*, cada 5 segundos se publicarán los valores tomados en formato *json* en el bróker *MQTT*. El primer sensor enviará temperatura, humedad, y nivel de luz. Este bucle se muestra en la figura 3.20.

```
void loop()
{
    //Whenever the mqtt client/broker disconnects, we attempt to connect again
    if (!client.connected())
    {
        mqttConection();
    }
    client.loop();

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

        print_Temp();
        print_Hum();
        printLight();

        DynamicJsonDocument doc(1024);

        doc["Temperature"]      = temperature;
        doc["Humidity"]         = humi;
        doc["Light"]             = LightValue;

        serializeJson(doc, json_msg);

        client.publish("main_and_only_topic", json_msg);

        Serial.println();
    }
}
```

Figura 3.20. Función loop() de mota 1

De igual manera, como se muestra en la figura 3.21, en el segundo ESP32 se enviará el nivel de pH.

```
void loop()
{
    //Whenever the mqtt client/broker disconnects, we attempt to connect again
    if (!client.connected())
    {
        mqttConection();
    }
    client.loop();

    long now = millis();
    if (now - lastMsg > 3500)
    {
        lastMsg = now;

        printPh();

        DynamicJsonDocument doc(1024);

        doc["Ph"] = Po;
        serializeJson(doc, json_msg);

        client.publish("main_and_only_topic", json_msg);

        Serial.println();
    }
}
```

Figura 3.21. Función loop() de mota 2

3.3.2. Configuración del router

Para poder realizar la monitorización desde cualquier lugar con conexión a internet se debe configurar el router que administra nuestra red local de forma que abra el puerto utilizado por el servicio web al público utilizando su IP privada. De esta forma cualquier persona con acceso a internet, y que tenga un dispositivo con interfaz gráfica de usuario y un explorador web, podrá acceder al servidor web alojado en la Raspberry Pi.

Configurar el router puede depender del sistema operativo de éste y de la compañía que proporcione el equipo. Normalmente, estos dispositivos tienen un servidor web accesible desde la red local en el cual se puede modificar, entre otras cosas, la configuración de los puertos.

3.3.3. Integración de tecnologías

En este apartado se integran las diferentes tecnologías presentes. En la figura 3.22 se pueden observar las diferentes aplicaciones instaladas en la Raspberry Pi para la monitorización de los datos, definidas anteriormente en la tabla 3.2 de este capítulo.



Figura 3.22. Tecnologías que conforman el sistema[26]

Toda la configuración de estas aplicaciones ha sido detallada en el manual de instalación, adjunto en el Apéndice B de este documento.

Mosquitto

Mosquitto es un bróker del protocolo *MQTT*[22], un protocolo de comunicación *M2M* (máquina hacia máquina) usado en multitud de dispositivos *IoT*. Este bróker proporciona un método de comunicaciones mediante publicaciones y suscripciones que lo hace liviano a la hora de transmitir datos. *Mosquitto* crea un *topic* por el cual las motas (*ESP32*) se conectarán a este servidor y publicarán datos. *Telegraf*, detallado a continuación, se conectará al servidor de forma interna y se suscribirá al *topic* para ver y guardar los datos en la base de datos[23].

Telegraf e influxDB

Telegraf es un Agente de software específico para *InfluxDB* que, en nuestro sistema, gestionará todos los datos publicados en el *topic* del bróker *mqtt* insertándolos en una base de datos administrada por *InfluxDB*[27]. *InfluxDB* es un gestor de bases de datos de series de tiempo. En las bases de datos creadas por esta aplicación se guardan valores con una marca de tiempo asociada, la

cual hace que la monitorización con gráficos históricos de tiempo sea más sencilla[28].

Grafana

Grafana es una plataforma que permite la visualización de datos a través de un servidor web. Esta aplicación permite además la creación y personalización de gráficos a partir de múltiples fuentes, entre las que se incluye *InfluxDB*. *Grafana* permite además la creación de alarmas hacia diferentes canales de comunicación. De esta forma, el cliente puede estar informado en caso de que algún valor adverso sea leído por la plataforma[29].

Capítulo 4. Verificación y pruebas

En este capítulo se muestra la descripción y justificación de las pruebas realizadas.

4.1. Sistema de pruebas

Para la realización de las pruebas se va a hacer uso del puerto de depuración del ESP32 (puerto serie), así como de códigos software de cada funcionalidad del sistema. Una vez que el dispositivo procese datos individualmente de cada sensor, se pasará a probar que recibe datos de todos a la vez. De esta manera podremos depurar posibles fallos más fácilmente.

Para las pruebas se ha hecho uso de Visual Studio Code[30] para el código y de la herramienta *PuTTY*[31] para depurar a través del puerto serie del dispositivo. El mismo cable USB usado para alimentar y programar el *ESP32* servirá para su depuración.

4.2. Pruebas realizadas

Nº	Nombre o ID del requisito	Descripción de la prueba	Resultado
1	Lectura de temperatura	Con el sistema parcialmente encendido (solo las motas y sensores) comprobar a través de funciones de depuración y el puerto serie que se están recogiendo lecturas correctas de Temperatura del BMP280 a través de I2C.	Positivo
2	Lectura de humedad	Con el sistema parcialmente encendido (solo las motas y sensores) comprobar a través de funciones de depuración y el puerto serie que se están recogiendo lecturas correctas de Humedad del AHT20 a través de I2C.	Positivo

3	Lecturas de iluminación	Con el sistema parcialmente encendido (solo las motas y sensores) comprobar a través de funciones de depuración y el puerto serie que se están recogiendo lecturas correctas analógicas de iluminación a través del TEMT6000.	Positivo
4	Lectura de PH	Con el sistema parcialmente encendido (solo las motas y sensores) comprobar a través de funciones de depuración y el puerto serie que se están recogiendo lecturas correctas de PH a través de la sonda.	Positivo
5	Lectura general	Con el sistema parcialmente encendido (solo las motas y sensores) comprobar a través de funciones de depuración y el puerto serie que se están recogiendo lecturas de todos los sensores de la primera mota (Pruebas 1, 2 y 3 en el mismo código).	Positivo
6	Envío de lecturas sobre MQTT	Con el sistema totalmente encendido, comprobar que las motas se conectan a la red Wi-Fi y se suscriben y publican a través del canal MQTT.	Positivo
7	Empaquejson sobre MQTT	Con el sistema totalmente encendido, comprobar la prueba anterior con el empaquetado de los datos en MQTT.	Positivo
8	Recepción de datos en base de datos	Con el sistema totalmente encendido, comprobar que los datos publicados en el servidor MQTT se están guardando adecuadamente en la base de datos.	Positivo
9	Monitorización en Grafana	Con el sistema totalmente encendido, comprobar que los datos guardados en la base de datos se muestran en el servidor web a través de Grafana.	Positivo
10	Correcto funcionamiento de Alarmas	Con el sistema totalmente encendido, comprobar que las lecturas al llegar a cierto límite hacen que Grafana notifique a un bot de Telegram.	Positivo
11	Conexión remota desde cualquier dispositivo	Con el sistema totalmente encendido y configurado, realizar una conexión al servidor web con un dispositivo conectado a internet fuera de la red local.	Positivo

Tabla 4.1. Tabla de las pruebas realizadas

Los sensores de temperatura, humedad iluminación y PH registran valores correctos como se puede apreciar en las figuras 4.1, 4.2, 4.3 y 4.4 respectivamente.

```

float temperature; //, humidity, pressure, altitude;
unsigned long delayTime;

void printValues() {
    bmp.getCurrentTemperature(&temperature/*, p*/);
    Serial.print("Temperatura = ");
    Serial.print(temperature);
    Serial.println(" *C");
    Serial.println();
}

void setup() {
    Serial.begin(115200);
    Serial.println(F("BMP280 test"));

    bmp.begin(0x76);
    bmp.setTimeStandby(TIME_STANDBY_05MS);
    bmp.startNormalConversion();

    Serial.println("-- Default Test --");
    Serial.println();
}

void loop() {
    printValues();
    delay(1500);
}

```

Temperatura = 28.11 *C
 Temperatura = 27.53 *C
 Temperatura = 26.82 *C
 Temperatura = 26.60 *C
 Temperatura = 25.62 *C
 Temperatura = 24.97 *C
 Temperatura = 24.37 *C
 Temperatura = 23.75 *C
 Temperatura = 23.27 *C
 Temperatura = 22.83 *C
 Temperatura = 22.39 *C
 Temperatura = 22.05 *C

Figura 4.1. Prueba 1

```

#include <Wire.h>
#include "ATH20.h"

ATH20 ATH;

void setup()
{
    Serial.begin(115200);
    ATH.begin();
}

void loop()
{
    float humi;

    int ret = ATH.getHumidity(&humi);

    if (ret) // GET DATA OK
    {
        Serial.print("Humidity: ");
        Serial.println(humi * 100);
        Serial.println("%");
    }
    else // GET DATA FAIL
    {
        Serial.println("GET DATA FROM ATH20 FAIL");
    }

    delay(1500);
}

```

Humidity: 67.28%
 Humidity: 66.91%
 Humidity: 66.73%
 Humidity: 66.46%
 Humidity: 66.23%
 Humidity: 66.22%
 Humidity: 66.14%
 Humidity: 66.01%
 Humidity: 65.91%
 Humidity: 65.94%
 Humidity: 66.10%
 Humidity: 66.00%
 Humidity: 65.77%
 Humidity: 65.58%
 Humidity: 65.32%
 Humidity: 65.38%
 Humidity: 65.31%
 Humidity: 65.28%
 Humidity: 65.14%
 Humidity: 64.95%
 Humidity: 64.89%
 Humidity: 64.97%
 Humidity: 64.92%

Figura 4.2. Prueba 2

```

1 int sense01 = 33;//Pin analógico
2 int val01 = 0;//Valor del sensor de luz
3 int valueMax = 1100;
4 int valueMin = 0;
5 int percentLight = 50;
6 void setup()
7 {
8     Serial.begin(115200);//Inicializa el puerto serie
9     pinMode(sense01, INPUT);
10 }
11
12 void loop()
13 {
14     val01 = analogRead(sense01);
15     Serial.print(val01);
16     percentLight = map(val01, valueMin, valueMax, 0, 100);
17     Serial.print("\t");
18     Serial.print("Porcentaje de luz: ");
19     Serial.print(percentLight);
20     Serial.println("%");
21     delay(1500);
22 }
23
24 }
25

```

	Porcentaje de luz:
365	33%
362	33%
367	33%
113	10%
113	10%
98	9%
91	8%
87	8%
368	33%
368	33%
497	45%
401	36%
735	67%
1631	100%
1920	100%
685	62%
623	57%
1136	100%
1061	96%
1065	97%
2132	100%

Figura 4.3. Prueba 3

```

1 int PinPh = 33;
2 int PinPhValue = 0;
3 double Po = 0;
4 float sensorMax = 3125.00;
5 float sensorMin = 2670.00;
6
7 float fmap(float x, float in_min, float in_max, float out_min, float out_max)
8 {
9     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
10 }
11 void setup()
12 {
13     Serial.begin(115200);//Inicializa el puerto serie
14     pinMode(PinPh, INPUT);
15 }
16 void loop()
17 {
18     PinPhValue = analogRead(PinPh);
19     Serial.println(PinPhValue);
20
21     Po = fmap(PinPhValue, sensorMax, sensorMin, 6.00, 8.00);
22     Serial.print("\tPH: ");
23     Serial.print(Po);
24
25     Serial.println("");
26     delay(2000);
27 }
28
29

```

	PH:
2992	6.60
2992	6.58
2987	6.61
2987	6.61
2992	6.58
2992	6.58
2981	6.63
2973	6.67
2983	6.62
2983	6.62
3020	6.46

Figura 4.4. Prueba 4

La mota está recogiendo lecturas de todos los sensores conectados como se puede ver en la figura 4.5.

```

void loop()
{
    printLight();
    print_Temp();
    printHum();

    delay(1500);

}

void printLight()
{
    LightValue = analogRead(PinLight);
    percentLight = map(LightValue, valueMin, valueMax, 0, 100);
    Serial.print("Porcentaje de luz: ");
    Serial.print(percentLight);
    Serial.println("%");
}

void print_Temp()
{
    bmp.getCurrentTemperature(temperature);

    Serial.print("Temperatura = ");
    Serial.print(temperature);
    Serial.println(" °C; ");
}

void printHum()
{
    ATH.getHumidity(&humi);
    Serial.print("Humidity: ");
    Serial.print(humi * 100);
    Serial.println("%");
}

```

COM3 - PuTTY

Temperatura = 28.02 °C;
Humidity: 63.86%
Porcentaje de luz: 67%
Temperatura = 28.02 °C;
Humidity: 64.12%
Porcentaje de luz: 77%
Temperatura = 28.02 °C;
Humidity: 64.25%
Porcentaje de luz: 78%
Temperatura = 28.02 °C;
Humidity: 64.14%
Porcentaje de luz: 64%
Temperatura = 28.01 °C;
Humidity: 64.03%
Porcentaje de luz: 100%
Temperatura = 28.01 °C;
Humidity: 63.99%
Porcentaje de luz: 15%
Temperatura = 28.01 °C;
Humidity: 63.90%
Porcentaje de luz: 9%
Temperatura = 28.01 °C;
Humidity: 63.97%

Figura 4.5. Prueba 5

La mota se conecta al broker *mqtt* y publica los valores leídos por los sensores en formato *json* como se aprecia en la figura 4.6.

```

Símbolo del sistema - mosquitto_sub -h 192.168.0.100 -u -P -t main_and_only_topic
M
(
{"Ph":7.287912}
{"Temperature":27.88,"Humidity":67,"Light":8}
{"Ph":7.287912}
{"Ph":7.261539}
 {"Temperature":27.9,"Humidity":67,"Light":7}
 {"Ph":7.274725}
 {"Temperature":27.89,"Humidity":67,"Light":7}
 {"Ph":7.331868}
 {"Ph":7.318681}
 {"Temperature":27.9,"Humidity":67,"Light":7}

```

Figura 4.6. Pruebas 6 y 7

En la figura 4.7 se puede comprobar como la base de datos recopila los datos recibidos por *mqtt*.

```
> select * from mqtt_consumer
name: mqtt_consumer
time          Humidity Light Ph      Temperature host   topic
----          -----  ---  --      -----  ----  -----
1658334546533608409 67      11    27.85      ubuntu main_and_only_topic
1658334548420151052           7.349451      ubuntu main_and_only_topic
1658334551531943715 67      11    27.83      ubuntu main_and_only_topic
1658334551925679834           7.336264      ubuntu main_and_only_topic
1658334555421994456           7.336264      ubuntu main_and_only_topic
1658334556534190705 67      11    27.84      ubuntu main_and_only_topic
1658334559191958335           7.415385      ubuntu main_and_only_topic
1658334561535401887 67      6     27.84      ubuntu main_and_only_topic
1658334562424227981           7.301099      ubuntu main_and_only_topic
1658334565925036782           7.296703      ubuntu main_and_only_topic
```

Figura 4.7. Prueba 8

En las figura 4.8 y 4.9 se muestran paneles de monitorización de *Grafana*.



Figura 4.8. Prueba 9



Figura 4.9. Prueba 9

El sistema de alarmas de *Grafana* funciona correctamente como se puede ver en la figura 4.10.

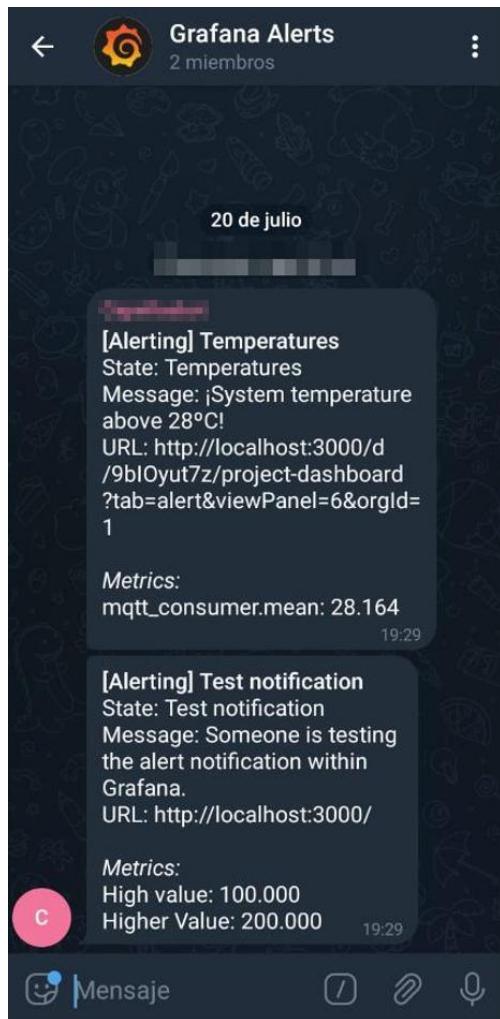


Figura 4.10. Prueba 10

El acceso al interfaz de monitorización se puede realizar desde cualquier terminal con conexión a internet como se muestra en la figura 4.11.



Figura 4.11. Prueba 11

Capítulo 5. Conclusiones y trabajo futuro

En este trabajo de fin de grado se ha conseguido el objetivo de desarrollar un prototipo para la monitorización de un cultivo de forma remota. Se ha logrado implementar con un bajo presupuesto cumpliendo con los requisitos establecidos, así como con requisitos añadidos posteriormente al ver las posibles mejoras que aportaban al sistema.

Gracias al desarrollo de este trabajo, he tenido que enfrentarme a conceptos que no he abordado en el grado como son el uso de comandos en una máquina Linux, la gestión de bases de datos y la administración de sistemas entre otros.

El factor más importante para que el sistema funcione ha sido la integración de diferentes tecnologías en una Raspberry Pi. En el desarrollo del prototipo ha sido importante saber cómo funciona cada uno de los diferentes servicios para que funcionen correctamente comunicándose entre sí.

Para que el sistema funcione correctamente se han tenido que configurar un protocolo de comunicación como lo es el MQTT sobre una red Wi-Fi. Como línea de trabajo futura se podría configurar un protocolo de red de malla con *PainlessMesh*[32], de forma que distribuir diferentes motas a lo largo de la zona de operación no suponga un problema de conexión, ya que el Wi-Fi tiene un rango limitado.

Durante el desarrollo del proyecto se ha percibido que el gestor de bases de datos utilizado no tiene la capacidad liberar espacio borrando registros antiguos. Para la cantidad de datos registrados no debería suponer un problema gracias a la gran capacidad de memoria disponible en el microprocesador. Sin embargo, crear algún *script* para poder eliminar elementos antiguos supondría una mejora. Cabe añadir, que el dispositivo cuenta con una capacidad de memoria (después

de toda la configuración) de 24 GB para una tarjeta microSD de 32 GB y que cada trama ocupa un espacio aproximado de 60 Bytes. Si se realizan medidas cada 5 segundos, tardaría en llenarse más de 60 años.

La programación de las motas se ha realizado utilizando el *framework* de *Arduino*. Para una mejor eficiencia en el consumo, se podría utilizar un sistema operativo como *FreeRTOS*, que hace uso de diferentes hilos y tareas manteniendo la mota en estado de espera en bajo consumo cuando no transmite datos.

Todas las comunicaciones del sistema se realizan en claro (sin encriptar). Una buena práctica es configurar las comunicaciones tanto de las motas con el microprocesador como las internas de la Raspberry Pi de forma que utilicen protocolos seguros como son TLS y SMQTT.

Para concluir, este trabajo de fin de grado se ha realizado gracias a los conocimientos adquiridos durante el Grado de Ingeniería de Sistemas Electrónicos. Diferentes asignaturas han sido primordiales para su correcto desarrollo ya que se han podido solventar muchos de los problemas obtenidos en la realización de este proyecto.

Referencias

- [1] J. A. Aznar-Sánchez, M. Piquer-Rodríguez, J. F. Velasco-Muñoz, y F. Manzano-Agugliaro, «Worldwide research trends on sustainable land use in agriculture», *Land use policy*, vol. 87, p. 104069, sep. 2019, doi: 10.1016/J.LANDUSEPOL.2019.104069.
- [2] «Countries Using Vertical Farming – Vertical Farming Planet». <https://verticalfarmingplanet.com/countries-using-vertical-farming/> (accedido ago. 29, 2022).
- [3] I. Lee y K. Lee, «The Internet of Things (IoT): Applications, investments, and challenges for enterprises», *Bus Horiz*, vol. 58, n.^o 4, pp. 431-440, jul. 2015, doi: 10.1016/J.BUSHOR.2015.03.008.
- [4] M. S. Farooq, S. Riaz, A. Abid, K. Abid, y M. A. Naeem, «A Survey on the Role of IoT in Agriculture for the Implementation of Smart Farming», *IEEE Access*, vol. 7, pp. 156237-156271, 2019, doi: 10.1109/ACCESS.2019.2949703.
- [5] «Arable - Decision Agriculture». <https://www.arable.com/> (accedido ago. 29, 2022).
- [6] «Automated Smart Gardening & Monitoring System With Grow Room Sensors». <https://www.getniwa.com/> (accedido ago. 29, 2022).
- [7] J. Antonio Fuentes Vicente, «AGRICULTURA DE PRECISIÓN. SMART FARMING: HACIA UNA AGRICULTURA AUTOMATIZADA Y CONECTADA RIEGA-t».

- [8] «Introduction to ESP32 | Specifications, ESP32 DevKit Board, Layout»,. <https://www.electronicshub.org/getting-started-with-esp32/> (accedido sep. 14, 2022).
- [9] «ESP32-DevKitC V4 Getting Started Guide - ESP32 - — ESP-IDF Programming Guide latest documentation». <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html> (accedido ago. 29, 2022).
- [10] «ESP32 Analog Input with Arduino IDE | Random Nerd Tutorials». <https://randomnerdtutorials.com/esp32-adc-analog-read-arduino-ide/> (accedido ago. 29, 2022).
- [11] «File:Raspberry Pi 4 Model B - Side.jpg - Wikimedia Commons». https://commons.wikimedia.org/wiki/File:Raspberry_Pi_4_Model_B_-_Side.jpg (accedido ago. 29, 2022).
- [12] «BMP280 modulo sensor presion barometrica digital remplaza al BMP180 | Leantec.ES». <https://leantec.es/tienda/bmp280-modulo-sensor-presion-barometrica-digital-reemplaza-al-bmp180/> (accedido ago. 29, 2022).
- [13] «Adafruit BMP280 I2C or SPI Barometric Pressure & Altitude Sensor [STEMMA QT]: ID 2651 : \$9.95 : Adafruit Industries, Unique & fun DIY electronics and kits». <https://www.adafruit.com/product/2651> (accedido ago. 29, 2022).
- [14] «GitHub - MartinL1/BMP280_DEV: An Arduino compatible, non-blocking, I2C/SPI library for the Bosch BMP280 barometer.» https://github.com/MartinL1/BMP280_DEV (accedido ago. 29, 2022).
- [15] «AHT20 I2C temperature and humidity sensor module high-precision humidity sensor probe DHT11 AHT10 upgraded version for arduino - ASK Electronics». <https://askelectronics.co.ke/product/aht20-i2c-temperature-and-humidity-sensor-module-high-precision-humidity-sensor-probe-dht11-aht10-upgraded-version-for-arduino/> (accedido ago. 29, 2022).

- [16] «Adafruit AHT20 - Temperature & Humidity Sensor Breakout Board [STEMMA QT / Qwiic] : ID 4566 : \$4.50 : Adafruit Industries, Unique & fun DIY electronics and kits». <https://www.adafruit.com/product/4566> (accedido ago. 29, 2022).
- [17] «GitHub - Seeed-Studio/Seeed_Arduino_AHT20: This library provides an example code to get the temperature and humidity from the AHT20 sensor.» https://github.com/Seeed-Studio/Seeed_Arduino_AHT20 (accedido ago. 29, 2022).
- [18] «Luz: Sensor de luz ambiente TEMT6000». <https://www.didacticaselectronicas.com/index.php/sensores/luz/sensor-de-luz-ambiente-temt6000-temt6000-sensores-de-luz-ambiental-ambiente-temt6000-detail> (accedido ago. 29, 2022).
- [19] «Sensor pH 4502C (ref: 1041) – electronperdido.com». <https://electronperdido.com/shop/sensores/quimicos/sensor-ph-4502c/> (accedido ago. 29, 2022).
- [20] «Help with Ph sensor pin abbreviations - Using Arduino / Sensors - Arduino Forum». <https://forum.arduino.cc/t/help-with-ph-sensor-pin-abbreviations/323936/40> (accedido sep. 02, 2022).
- [21] «Sensor de detección de PH 0 14, módulo de monitoreo de prueba de valor de PH, sonda de electrodo de prueba de valor de PH para medir el agua del grifo del suelo|Medidores de PH| - AliExpress». <https://es.aliexpress.com/item/1005002780985157.html> (accedido ago. 29, 2022).
- [22] «MQTT - The Standard for IoT Messaging». <https://mqtt.org/> (accedido ago. 29, 2022).
- [23] «Eclipse Mosquitto». <https://mosquitto.org/> (accedido ago. 29, 2022).
- [24] «JSON». <https://www.json.org/json-es.html> (accedido ago. 29, 2022).
- [25] «wmath - npm». <https://www.npmjs.com/package/wmath> (accedido ago. 29, 2022).

- [26] «Raspberry Pi IoT: Sensors, InfluxDB, MQTT, and Grafana - DZone IoT». <https://dzone.com/articles/raspberry-pi-iot-sensors-influxdb-mqtt-and-grafana> (accedido ago. 29, 2022).
- [27] «Telegraf Open Source Server Agent | InfluxDB». <https://www.influxdata.com/time-series-platform/telegraf/> (accedido ago. 29, 2022).
- [28] «InfluxDB: Open Source Time Series Database | InfluxData». <https://www.influxdata.com/> (accedido ago. 29, 2022).
- [29] «Grafana: The open observability platform | Grafana Labs». <https://grafana.com/> (accedido ago. 29, 2022).
- [30] «Visual Studio Code - Code Editing. Redefined». <https://code.visualstudio.com/> (accedido ago. 29, 2022).
- [31] «Download PuTTY - a free SSH and telnet client for Windows». <https://www.putty.org/> (accedido ago. 29, 2022).
- [32] «painlessMesh / painlessMesh · GitLab». <https://gitlab.com/painlessMesh/painlessMesh> (accedido sep. 07, 2022).
- [33] «Install Ubuntu on a Raspberry Pi | Ubuntu». <https://ubuntu.com/download/raspberry-pi> (accedido ago. 29, 2022).
- [34] «balenaEtcher - Flash OS images to SD cards & USB drives». <https://www.balena.io/etcher/> (accedido ago. 29, 2022).

Apéndice A. Presupuesto y horas de desarrollo

Componente	Coste unitario (€)	Unidades empleadas	Coste total (€)
Kit Raspberry Pi 4 4GB	104.64	1	104.64
ESP32	4	2	8
BMP280	0.74	1	0.74
Sonda electrodo (pH)	6.86	1	6.86
Módulo de monitoreo (pH)	12.45	1	12.45
TEMT6000	0.74	1	0.74
AHT20	1.36	1	1.36
Cables de conexión	0.1715	40	6.86
Placa Prototipo <i>Protoboard</i>	9.59	1	9.59
Cable micro USB	3.90	2	7.80
Estaño (50gr)	3.80	1	3.80
Total			162.84

Tabla A.1. Coste de componentes y material fungible

Equipo/software	Coste unitario (€)	Periodo de amortización	Periodo de uso	Coste total (€)
Ordenador	800	36 meses	6 meses	134
Monitor	160	36 meses	6 meses	27
Soldador	15	12 meses	0.2 meses	0.26
Teclado	4.95	12 meses	6 meses	2.48
Ratón	25	12 meses	6 meses	12.50
Total				176.24

Tabla A.2. Amortización de equipos y licencias de software

Actividad	Horas
Coordinación del TFG	7
Análisis de requisitos	6
Desarrollo hardware/software del prototipo	200
Realización de las pruebas	15
Redacción de la memoria	100
Preparación de la presentación	12
Horas totales	340

Tabla A.3. Horas invertidas en la realización del proyecto

Concepto	Coste (€)
Coste de componentes y material fungible	162.84
Amortización de equipos y licencias de software	176.24
Coste total	339.08

Tabla A.4. Presupuesto total

Apéndice B. Manual de instalación

Este anexo contiene el manual de instalación del sistema.

1. Requisitos de instalación

Se va a instalar Ubuntu Server. Este sistema operativo está certificado para funcionar en diferentes versiones de Raspberry Pi. Serán necesarios los siguientes componentes:

- Raspberry Pi 2, 3 ó 4. En la versión 2 sólo funcionará el sistema operativo de 32 bits.
- Tarjeta microSD. Recomendada de 8 GB. Por flexibilidad y prevenir problemas en cuanto a espacio disponible se refiere, se recomienda una tarjeta de 16 GB o más.
- Un *router* WiFi con conexión a internet con el que poder crear una red local. Para la primera conexión será necesario un cable RJ-45 para conectar la Raspberry Pi a la red local.

2. Configuración de microSD para instalación de sistema operativo

El sistema operativo de la Raspberry Pi se encuentra en la tarjeta SD externa. En este manual se va a instalar el sistema Ubuntu Server, imagen de este se encuentra disponible en el repositorio oficial de Ubuntu[33].

Para preparar la tarjeta microSD será necesario que descargue un programa para poder cargarle la imagen de Ubuntu Server. En este manual se utilizará Balena Etcher. Este programa se puede usar de forma portable y no será necesaria su instalación[34]. Iniciando el programa, cargará el siguiente interfaz:



Figura B.1. Interfaz de balena etcher

Como se muestra en la figura B.1, este proceso queda resumido en 3 simples pasos:

- Seleccione la imagen de Ubuntu Server con “Flash from file”.
- Elija en qué disco quieras cargar dicha imagen con “Select Target”.
- Pulse “Flash” para empezar a cargar la imagen.

Después de una corta espera en pantalla se mostrará como sigue en la figura B.2.



Figura B.2. Carga completada

Es entonces cuando puede insertar la microSD en la Raspberry Pi y posteriormente arrancarla.

3. Configuraciones previas del sistema operativo.

Una vez la Raspberry Pi esté cargada, conéctela a la red local a través de un cable ethernet. Tras conectarla deberá encontrar su dirección IP. Para ello haga un escaneo de conexiones con la herramienta Zenmap (el equivalente a nmap en Windows). Esta herramienta le mostrará todos los dispositivos conectados a su router junto a su IP como se muestra en la figura B.3.

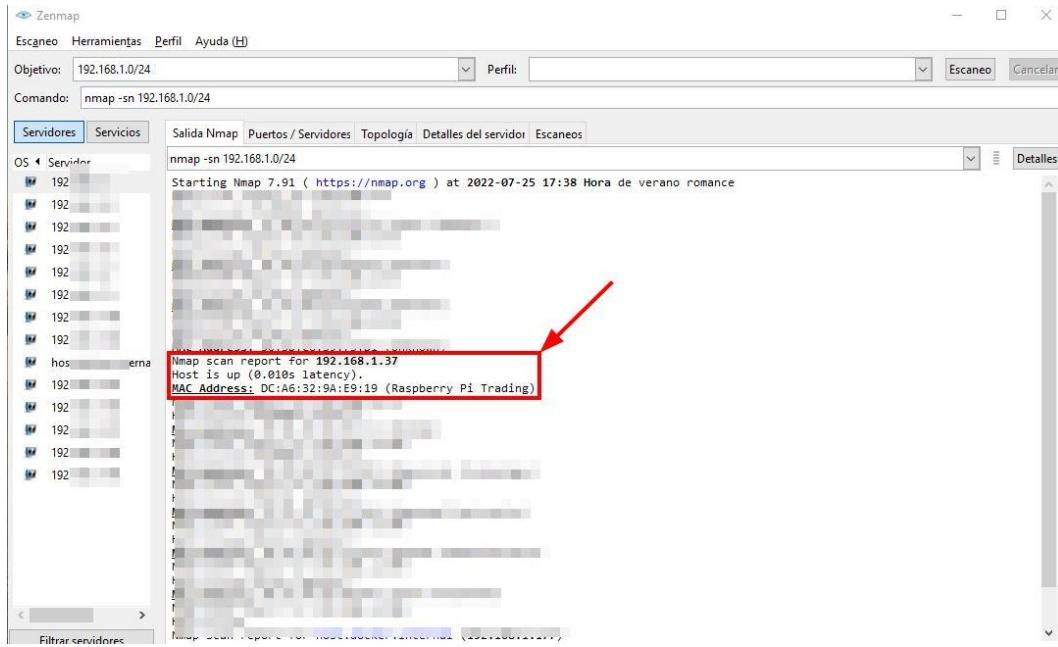


Figura B.3. Resultado de escaneo de puertos

Una vez conocemos la IP, entre al sistema operativo de la Raspberry PI a través de SSH (protocolo que estará habilitado por defecto en el sistema operativo). Para conectar a través de SSH abra un nuevo terminal y ejecute el siguiente comando

```
ssh <nombre de usuario@< IP del dispositivo>
```

y siga los pasos que nos dicte el terminal.

Una vez se conecte, las credenciales por defecto de Ubuntu Server son:

Usuario: **ubuntu**

Contraseña: **ubuntu**

Al entrar por primera vez deberá cambiar la contraseña y realizar la configuración inicial del sistema operativo. En la figura B.4 se muestra esta configuración.

```
>ssh ubuntu@192.168.1.37
The authenticity of host '192.168.1.37 (192.168.1.37)' can't be established.
ECDSA key fingerprint is SHA256:1 pNpU.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.1.37' (ECDSA) to the list of known hosts.
ubuntu@192.168.1.37's password:
You are required to change your password immediately (administrator enforced)
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-1042-raspi aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 System information as of Wed Jul 21 19:01:45 UTC 2021

 System load:  1.49           Temperature:      40.4 °C
 Usage of /:   6.4% of 28.96GB Processes:          136
 Memory usage: 6%            Users logged in:    0
 Swap usage:  0%            IPv4 address for eth0: 192.168.1.37

0 updates can be applied immediately.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

WARNING: Your password has expired.
You must change your password now and login again!
Changing password for ubuntu.
Current password:
```

Figura B.4. Configuración inicial de UbuntuServer

Una vez dentro del sistema operativo, será necesario modificar algunos archivos de configuración para poder conectar la Raspberry Pi por Wi-Fi a la red local. Esto servirá para poder mover el servidor principal de forma física al lugar donde mejor convenga en la instalación. Además, se le proporcionará una IP estática para que las motas puedan conectarse al servidor de manera que no tengan que buscar dónde está alojado el servidor cada vez que arranquen.

Para ambos casos, modifique el mismo archivo, “**/etc/netplan/50-cloud-init.yaml**”. Por defecto, los contenidos de este fichero se pueden ver en la figura B.5.

```
GNU nano 4.8 50-cloud-init.yaml
# This file is generated from information provided by the datasource. Changes
# to it will not persist across an instance reboot. To disable cloud-init's
# network configuration capabilities, write a file
# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:
# network: {config: disabled}
network:
  ethernets:
    eth0:
      dhcp4: true
      optional: true
version: 2
```

Figura B.5. Archivo de configuración de red

Se le añadirá la opción de conectarse mediante Wi-Fi además de añadir las credenciales de la red, así como la IP estática que queremos que tenga (en este caso se ha elegido la IP 192.168.1.5). Además, para poder tener acceso a internet desde la Raspberry Pi, hemos añadido dos servidores públicos DNS de Google para que pueda resolver diferentes URLs (Necesario para agilizar la posterior instalación de diferentes servicios). El archivo anteriormente mencionado deberá quedar como se muestra en la figura B.6.

```
GNU nano 4.8 50-cloud-init.yaml
# This file is generated from information provided by the datasource. Changes
# to it will not persist across an instance reboot. To disable cloud-init's
# network configuration capabilities, write a file
# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:
# network: {config: disabled}
network:
  ethernets:
    eth0:
      dhcp4: true
      optional: true
  version: 2
  wifis:
    wlan0:
      optional: true
      access-points:
        "██████████": {
          password: "██████████"
        }
      dhcp4: no
      addresses: [192.168.1.5/24]
      gateway4: 192.168.1.1
      nameservers:
        addresses: [8.8.4.4,8.8.8.8]
```

Figura B.6. Archivo de configuración de red editado

Este archivo es sensible y tener los espacios y gramática como se muestra en imagen es importante para que se apliquen los cambios satisfactoriamente. Pulsar **Ctrl+X** para salir del editor, **Y** para guardar los cambios y **Enter** para confirmar el nombre del archivo. Una vez modificado, debemos aplicar los cambios con el siguiente comando

```
sudo netplan apply
```

Para verificar que los cambios se han guardado satisfactoriamente se ha ejecutado

```
ip a
```

y obteniendo la salida mostrada en la figura B.7, comprobando que se conecta a la red Wi-Fi y obtiene la IP estática definida:

```
ubuntu@ubuntu:/etc/netplan$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group default qlen 1000
    link/ether e4:5f:01:24:89:67 brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether e4:5f:01:24:89:68 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.5/24 brd 192.168.1.255 scope global wlan0
        valid_lft forever preferred_lft forever
        inet6 fe80::e65f:1ff:fe24:8968/64 scope link
            valid_lft forever preferred_lft forever
ubuntu@ubuntu:/etc/netplan$
```

Figura B.7. Dirección IP del dispositivo

Además, se puede comprobar la conexión a internet haciendo ping al servidor de Google ejecutando

```
ping 8.8.8.8
```

y obteniendo la siguiente salida, mostrada en la figura B.8, donde el servidor responde al ping:

```
ubuntu@ubuntu:/etc/netplan$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=13.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=19.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=15.1 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 13.474/15.983/19.342/2.469 ms
ubuntu@ubuntu:/etc/netplan$
```

Figura B.8. Comprobación de conectividad

A partir de ahora se dejará de usar el cable ethernet de la Raspberry Pi ya que se conectarán automáticamente al router a través de una conexión Wi-Fi. La nueva IP estática servirá para establecer la conexión por SSH.

4. Programación del ESP32

4.1. Drivers

Se ha utilizado el sistema operativo Windows 10. Para que nuestro sistema operativo pueda detectar el puerto de comunicaciones de las motas, es necesario tener instalado el controlador pertinente. En caso de no estar instalado, se debe descargar e instalar el driver de la empresa *Silicon labs*: *CP210X USB to UART Bridge Virtual COM Port*. Este driver se puede encontrar en el repositorio oficial de la empresa, disponible en silabs.com/developers/usb-to-uart-bridge-vcp-drivers.

Una vez instalado, al conectar la placa de desarrollo ESP32 podrá comprobar en el administrador de dispositivos que nuestro sistema operativo reconoce el dispositivo asignándole un puerto de comunicación como se aprecia en la figura B.9.

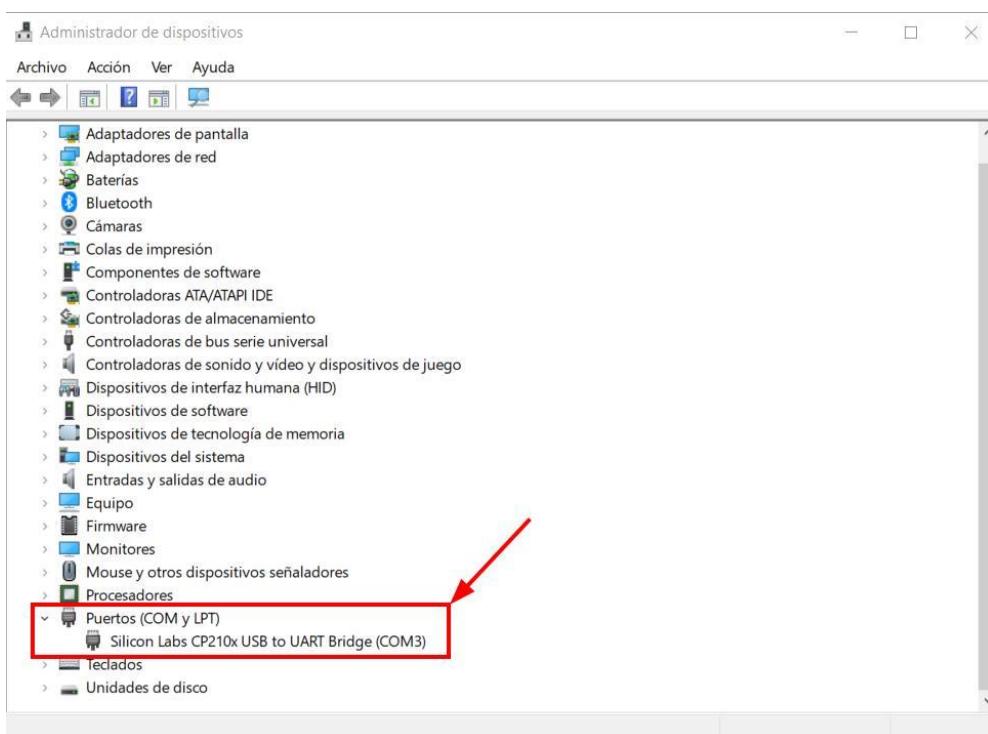


Figura B.9. Puertos de comunicación disponibles

4.2. Visual Studio Code

Una vez que el controlador esté instalado, para poder programar y cargar código en la mota es necesario Visual Studio Code. Su descarga se puede hacer desde el repositorio oficial, disponible en code.visualstudio.com/download. Basta con ejecutar el archivo descargado y seguir la ventana de instalación.

4.3. Python

Para poder cargar código en el ESP32 desde Visual Studio Code es necesario tener instalada una versión de Python 3.5 o mayor en el ordenador. En este manual hemos utilizado Python 3.8.5, disponible en el repositorio oficial de Python en python.org/downloads/. Es importante marcar la casilla de “add Python to PATH” en la ventana de instalación.

4.4. PlatformIO IDE Extension

Para instalar esta extensión en Visual Studio Code seleccione el símbolo “extensiones” para abrir la ventana de extensiones. Esta ventana se localiza como se muestra en la figura B.10.

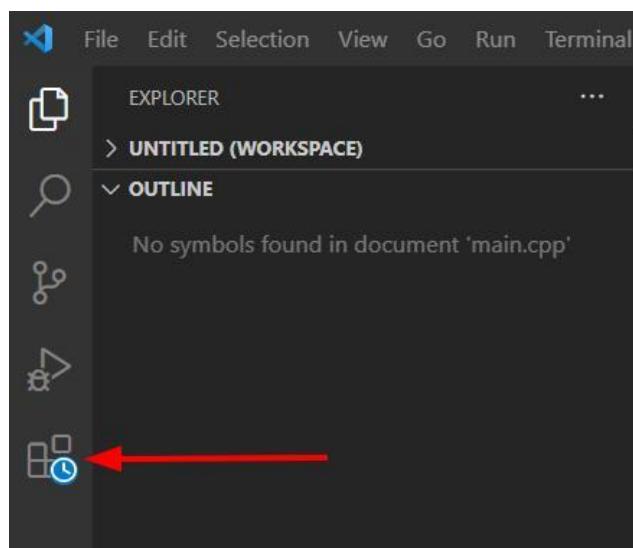


Figura B.10. Extensiones de Visual Studio Code

Una vez abierto, busque PlatformIO IDE, selecciónelo y pulse instalar. En la siguiente imagen se puede ver dónde aparecía el botón para instalar. En el caso mostrado en la figura B.11 ya se encontraba previamente instalado.

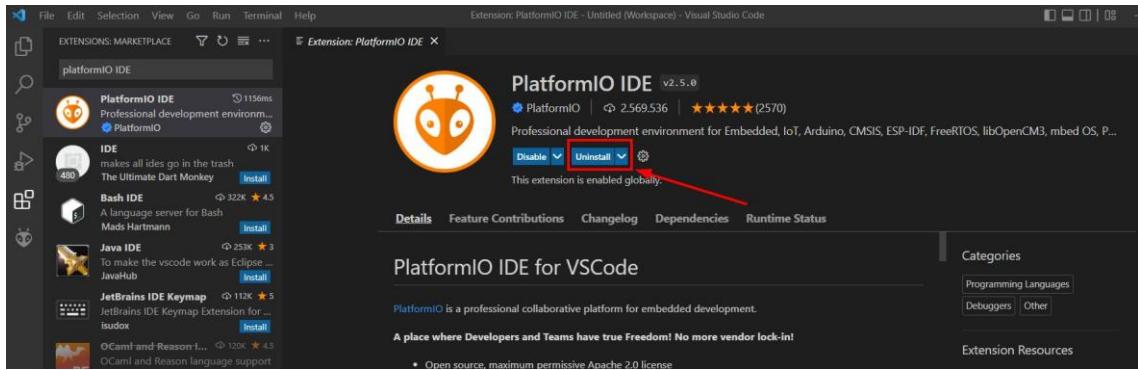


Figura B.11. Instalación de PlatformIO

En caso de no ver el icono de la extensión debajo del icono de extensiones, reinicie el programa para que los cambios surjan efecto.

Puede entonces cargar el proyecto que se encuentra en el repositorio de github. Para ello debe navegar a la casilla de inicio de PlatformIO y pulsar en Import como se muestra en la figura B.12.

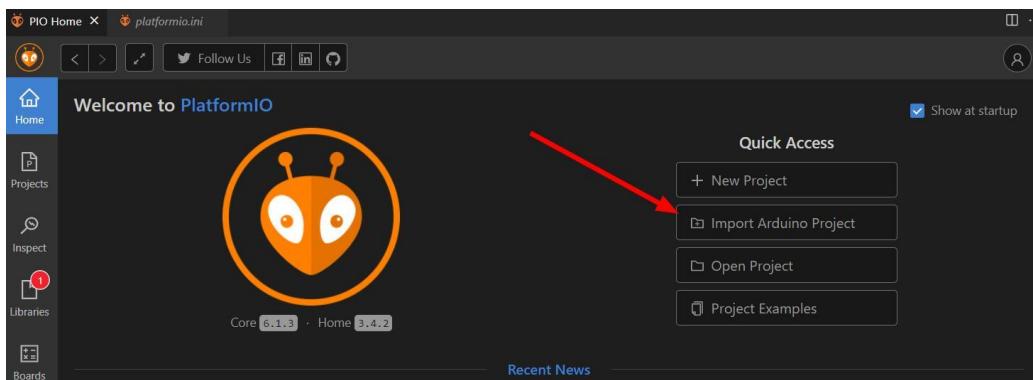
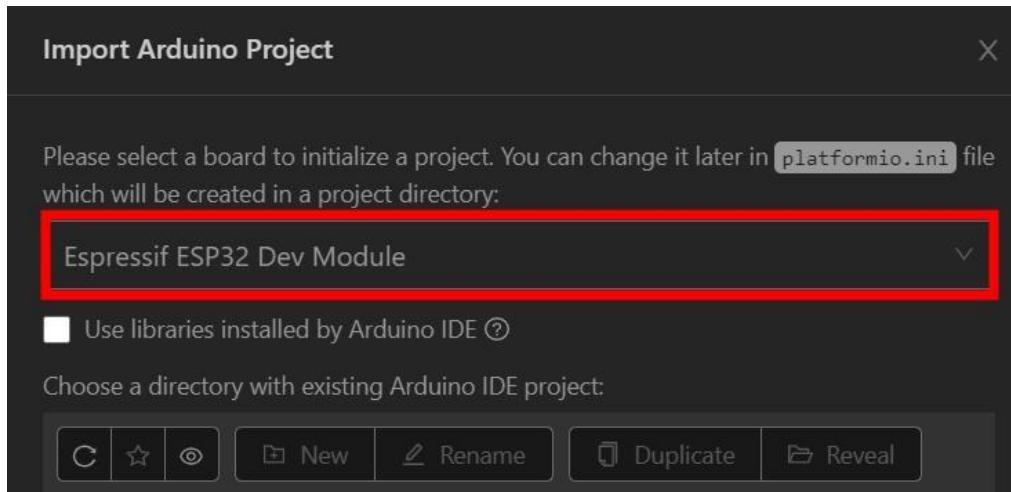


Figura B.12. Importación de proyecto

Posteriormente es necesario elegir la placa de desarrollo en la que vamos a trabajar. En este caso seleccionar *Espressif ESP32 Dev Module* como se muestra en la figura B.13.

**Figura B.13. Elección de placa de desarrollo**

Y elegir el proyecto que quiera cargarse. En este caso, los proyectos *ESP32_TFG* y *ESP32_2_TFG*.

4.5. Cargar código fuente a motas

Antes de compilar el código fuente, debe definir los siguientes parámetros:

- 1) Credenciales de la red local a la que se va a conectar la mota, SSID y contraseña. Estas credenciales son las mismas que se han configurado anteriormente en la Raspberry Pi.
- 2) IP del servidor. Es decir, IP previamente configurada en la Raspberry Pi.
- 3) Credenciales del servidor MQTT y *topic* al que publicar. Estas credenciales deben ser las mismas que se configuran en el bróker MQTT *mosquitto*.

Para definir dichos parámetros, solo hay que añadirlos en el trozo de código mostrado en la figura B.14.

```

8
9 //-----WiFi Credentials -----
10
11 const char* ssid      = "<SSID_Wi-Fi>";
12 const char* password = "<Contraseña>";
13 const char* mqtt_server = "<IP_RaspberryPi>";
14 const char* mqtt_username = "<usuario_mqtt>";
15 const char* mqtt_password = "<contraseña_mqtt>";
16 const char* clientID = "<ESP32client1>";
17 const char* main_and_only_topic = "<global/measurements>";
18

```

Figura B.14. Credenciales en código fuente

Para compilar y cargar el código fuente a las motas hay que elegir el proyecto que queremos utilizar. Para ello pulse en la ventana mostrada en la figura B.15.

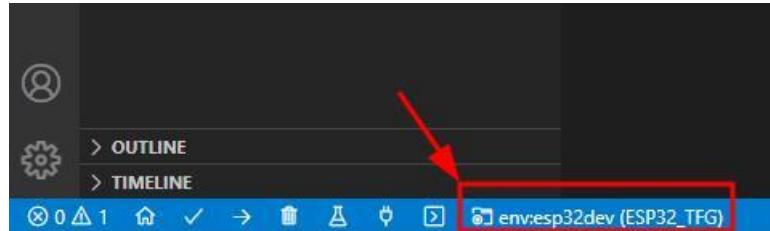


Figura B.15. Selección de proyecto

y elija el proyecto como se muestra en la figura B.16.

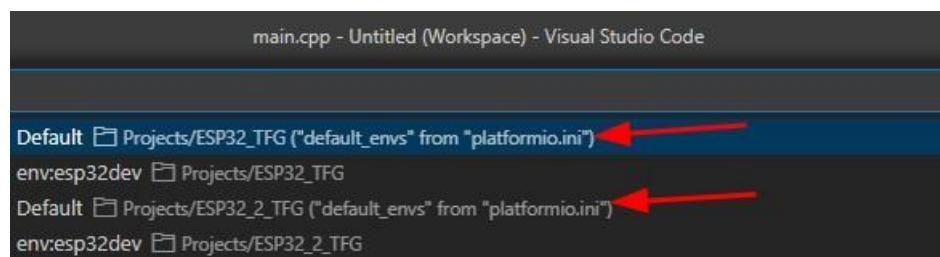


Figura B.16. Elección de proyecto

Después de cargar la mota en cuestión pulse *upload*, como se señala en la figura B.17, para compilar y cargar el código fuente.

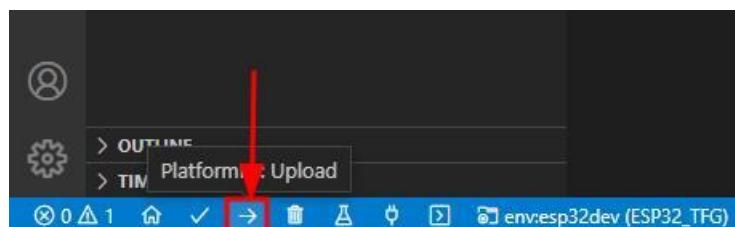


Figura B.17. Carga de proyecto a placa de desarrollo

5. Configuración de la Raspberry Pi

5.1. Instalación del bróker mosquitto

El siguiente comando sirve para instalar el bróker o servidor *MQTT* en la *Raspberry Pi*

```
sudo apt install mosquitto
```

En caso de que los paquetes de instalación no estén disponibles, ejecute el siguiente comando para actualizar los repositorios de Ubuntu.

```
sudo apt update
```

Después de instalar, modifique la configuración de este servicio mediante el fichero “**/etc/mosquitto/mosquitto.conf**”

Ejecute el comando

```
sudo nano /etc/mosquitto/mosquitto.conf
```

y añada las siguientes líneas

```
allow_anonymous false  
password file /etc/mosquitto/pwd  
listener 1883
```

Hay que crear el archivo que se ha incluido en la configuración, ‘pwd’. Para ello ejecute

```
sudo mosquitto_passwd -c /etc/mosquitto/pwd <nombre_usuario>
```

añadiendo el nombre de usuario y la contraseña posteriormente. Deben ser las mismas credenciales que hemos definido en el código en el punto 2.5-3 de este anexo.

Para ver el estado del bróker utilice el comando

```
sudo systemctl status mosquitto
```

Debería ver algo parecido a lo que se muestra en la figura B.18.

```
ubuntu@ubuntu:~$ sudo systemctl status mosquitto.service  
● mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker  
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)  
   Active: active (running) since Tue 2022-07-26 16:34:56 UTC; 10min ago  
     Docs: man:mosquitto.conf(5)  
           man:mosquitto(8)  
 Main PID: 2764 (mosquitto)  
    Tasks: 3 (limit: 4435)  
   CGroup: /system.slice/mosquitto.service  
          └─2764 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf  
  
Jul 26 16:34:56 ubuntu systemd[1]: Starting Mosquitto MQTT v3.1/v3.1.1 Broker...  
Jul 26 16:34:56 ubuntu mosquitto[2764]: [ 116.779354]~DLT~ 2764~INFO ~FIFO /tmp/dlt cannot be opened. Retrying later...  
Jul 26 16:34:56 ubuntu systemd[1]: Started Mosquitto MQTT v3.1/v3.1.1 Broker.  
ubuntu@ubuntu:~$
```

Figura B.18. Servicio *Mosquitto* corriendo en segundo plano

Para controlar el bróker, use los siguientes comandos:

Iniciar el bróker.

```
sudo systemctl start mosquitto
```

Parar el bróker:

```
sudo systemctl stop mosquitto
```

Reiniciar el bróker:

```
sudo systemctl restart mosquitto
```

Iniciar el bróker en el arranque:

```
sudo systemctl enable mosquitto
```

5.2. Instalación del gestor de bases de datos *InfluxDB*

El siguiente comando sirve para instalar el gestor de bases de datos *InfluxDB* en la *Raspberry Pi*

```
sudo apt install influxdb
```

Y el siguiente para instalar el intérprete de comandos para influxDB

```
sudo apt install influxdb-client
```

En caso de que los paquetes de instalación no estén disponibles, ejecutar el siguiente comando para actualizar los repositorios de Ubuntu.

```
sudo apt update
```

Después de instalar, modificar la configuración de este servicio mediante el fichero “/etc/influxdb/influxdb.conf”

Ejecuta el comando

```
sudo nano /etc/influxdb/influxdb.conf
```

En este archivo de configuración, en el apartado [http] debemos eliminar la almohadilla antes de la línea **enabled = true** como se muestra en la figura B.19.

```
GNU nano 4.8                                     /etc/influxdb/influxdb.conf
#####
##### Controls how the HTTP endpoints are configured. These are the primary
##### mechanism for getting data into and out of InfluxDB.
#####

[http]
# Determines whether HTTP endpoint is enabled.
enabled = true

# The bind address used by the HTTP service.
# bind-address = ":8086"

# Determines whether user authentication is enabled over HTTP/HTTPS.
# auth-enabled = false
```

Figura B.19. Archivo de configuración de *InfluxDB*

Para ver el estado del bróker utilice el comando

```
sudo systemctl status influxd
```

Debería ver algo parecido a lo que se muestra en la figura B.20.

```
ubuntu@ubuntu:~$ sudo systemctl status influxd
● influxdb.service - InfluxDB is an open-source, distributed, time series database
  Loaded: loaded (/lib/systemd/system/influxdb.service; enabled; vendor preset: enabled)
  Active: active (running) since Thu 2022-07-28 18:08:51 UTC; 10min ago
    Docs: man:influxd(1)
   Main PID: 1696 (influxd)
      Tasks: 13 (limit: 4435)
     CGroup: /system.slice/influxdb.service
             └─1696 /usr/bin/influxd -config /etc/influxdb/influxdb.conf

Jul 28 18:08:53 ubuntu influxd[1696]: ts=2022-07-28T18:08:53.602454Z lvl=info msg="Registered diagno...
Jul 28 18:08:53 ubuntu influxd[1696]: ts=2022-07-28T18:08:53.606133Z lvl=info msg="Starting precreat...
Jul 28 18:08:53 ubuntu influxd[1696]: ts=2022-07-28T18:08:53.6066720Z lvl=info msg="Storing statistic...
Jul 28 18:08:53 ubuntu influxd[1696]: ts=2022-07-28T18:08:53.607191Z lvl=info msg="Starting snapshot...
Jul 28 18:08:53 ubuntu influxd[1696]: ts=2022-07-28T18:08:53.607258Z lvl=info msg="Starting continuou...
Jul 28 18:08:53 ubuntu influxd[1696]: ts=2022-07-28T18:08:53.607284Z lvl=info msg="Starting HTTP serv...
Jul 28 18:08:53 ubuntu influxd[1696]: ts=2022-07-28T18:08:53.607304Z lvl=info msg="opened HTTP access...
Jul 28 18:08:53 ubuntu influxd[1696]: ts=2022-07-28T18:08:53.607610Z lvl=info msg="Listening on HTTP...
Jul 28 18:08:53 ubuntu influxd[1696]: ts=2022-07-28T18:08:53.607665Z lvl=info msg="Starting retention...
Jul 28 18:08:53 ubuntu influxd[1696]: ts=2022-07-28T18:08:53.607812Z lvl=info msg="Listening for sign...
ubuntu@ubuntu:~$
```

Figura B.20. Servicio *InfluxDB* corriendo en segundo plano

Para controlar el servicio, use los siguientes comandos:

Iniciar el bróker:

```
sudo systemctl start influxd
```

Parar el bróker:

```
sudo systemctl stop influxd
```

Reiniciar el bróker:

```
sudo systemctl restart influxd
```

Iniciar el bróker en el arranque:

```
sudo systemctl enable influxd
```

Para acceder al cliente de *InfluxDB* ejecute el intérprete de comandos usando el comando **influx**. Esto servirá para la depuración de la instalación y archivos de configuración, observando los datos que se van guardando en la propia base de datos. En la figura B.18. se muestra el contenido de la base de datos.

```
ubuntu@ubuntu:~$ influx
Connected to http://localhost:8086 version 1.6.4
InfluxDB shell version: 1.6.4
> use Project
Using database Project
> select * from mqtt_consumer
name: mqtt_consumer
time          Humidity Light Ph      Temperature host    topic
----          -----  --  --      -----  ----  -----
1658334546533608409 67       11     27.85      ubuntu main_and_only_topic
1658334548420151052           7.349451
1658334551531943715 67       11     27.83      ubuntu main_and_only_topic
1658334551925679834           7.336264
1658334555421994456           7.336264
1658334556534190705 67       11     27.84      ubuntu main_and_only_topic
1658334559191958335           7.415385
```

Figura B.21. Cliente *influx* administrando base de datos

Se incluyen los siguientes comandos para poder administrar las bases de datos disponibles:

Para crear una nueva base de datos:

```
create database <nombre de base de datos>
```

Para que el intérprete seleccione una base de datos:

```
use <nombre de base de datos>
```

Para mostrar valores guardados en base de datos:

```
select * from <nombre de medida>
```

El asterisco selecciona todos los valores (Temperatura, Ph, luminosidad, humedad, *topic*, host, huella de tiempo) y el nombre de medida será *mqtt_consumer*, definido en los archivos de configuración de *Telegraf*, explicados a continuación.

5.3. Instalación del agente Telegraf

El siguiente comando sirve para instalar agente de software *Telegraf* en la *Raspberry Pi*

```
sudo apt install telegraf
```

En caso de que los paquetes de instalación no estén disponibles, ejecute el siguiente comando para actualizar los repositorios de Ubuntu.

```
sudo apt update
```

Después de instalar, modifique la configuración de este servicio mediante el fichero “**/etc/telegraf/telegraf.conf**”

Ejecuta el comando

```
sudo nano /etc/telegraf/telegraf.conf
```

y modifique el fichero para añadir lo siguiente:

- Para conectar al agente como cliente al servidor *mqtt* añada en el apartado “[*inputs.mqtt_consumer*]” dónde se aloja dicho servidor, así como a qué *topics* debe suscribirse y en qué formato leer los datos publicados y las credenciales necesarias para poder conectar. Si dicho apartado no existe, créalo. Debe añadirlo como se muestra en la figura B.22.

```
[["inputs.mqtt_consumer"]]
  ## Broker URLs for the MQTT server or cluster. To connect to multiple
  ## clusters or standalone servers, use a separate plugin instance.
  ##   example: servers = ["tcp://localhost:1883"]
  ##           servers = ["ssl://localhost:1883"]
  ##           servers = ["ws://localhost:1883"]
  servers = ["tcp://localhost:1883"]

  ## Topics that will be subscribed to.
  topics = [
    "global/measurements",
    "#",
  ]

  ## Username and password to connect MQTT server.
  ##   by this value. If set
  ##   username = "admin"
  ##   password = "Password1"

  ## Data format to consume.
  ## Each data format has its own unique set of configuration options, read
  ## more about them here:
  ## https://github.com/influxdata/telegraf/blob/master/docs/DATA_FORMATS_INPUT.md
  data_format = "json"
```

Figura B.22. Archivo de configuración de *telegraf*

Las credenciales *admin* / *Password1* se han puesto a modo de ejemplo.

- Para guardar en la base de datos y que no se sobrescriba, añada las siguientes líneas como se muestra a continuación. Además, se añaden las credenciales de acceso a la base de datos como se muestra en la figura B.23.

```
[[outputs.influxdb]]
## The URLs of the InfluxDB cluster nodes.
##
## Multiple URLs can be specified for a single cluster, only ONE of the
## urls will be written to each interval.
## urls exp: http://127.0.0.1:8086
urls = ["http://127.0.0.1:8086"]

database = "Project"

skip_database_creation = true

username = "admin"
password = "Password1"
```

Figura B.23. Archivo de configuración de telegraf

Las credenciales *admin* / *Password1* se han puesto a modo de ejemplo.

Para ver el estado del servicio utilice el comando

```
sudo systemctl status telegraf
```

Debería ver algo parecido a lo que se muestra en la figura B.24.

```
ubuntu@ubuntu:~$ sudo systemctl status telegraf
● telegraf.service - The plugin-driven server agent for reporting metrics into InfluxDB
  Loaded: loaded (/lib/systemd/system/telegraf.service; enabled; vendor preset: enabled)
  Active: active (running) since Thu 2022-07-28 19:00:40 UTC; 1 weeks 3 days ago
    Docs: https://github.com/influxdata/telegraf
   Main PID: 1720 (telegraf)
      Tasks: 11 (limit: 4435)
     CGroup: /system.slice/telegraf.service
             └─1720 /usr/bin/telegraf -config /etc/telegraf/telegraf.conf -config-directory /etc/telegraf/telegraf.d

Jul 28 19:00:40 ubuntu systemd[1]: Started The plugin-driven server agent for reporting metrics into InfluxDB.
Jul 28 19:00:47 ubuntu telegraf[1720]: 2022-07-28T19:00:47Z I! Starting Telegraf 1.20.4
Jul 28 19:00:47 ubuntu telegraf[1720]: 2022-07-28T19:00:47Z I! Loaded inputs: cpu disk diskio mem mqtt_consumer net process
Jul 28 19:00:47 ubuntu telegraf[1720]: 2022-07-28T19:00:47Z I! Loaded aggregators:
Jul 28 19:00:47 ubuntu telegraf[1720]: 2022-07-28T19:00:47Z I! Loaded processors:
Jul 28 19:00:47 ubuntu telegraf[1720]: 2022-07-28T19:00:47Z I! Loaded outputs: influxdb
Jul 28 19:00:47 ubuntu telegraf[1720]: 2022-07-28T19:00:47Z I! Tags enabled: host=ubuntu
Jul 28 19:00:47 ubuntu telegraf[1720]: 2022-07-28T19:00:47Z I! [agent] Config: Interval:10s, Quiet:false, Hostname:"ubuntu"
Jul 28 19:00:47 ubuntu telegraf[1720]: 2022-07-28T19:00:47Z I! [inputs.mqtt_consumer] Connected [tcp://localhost:1883]
```

Figura B.24. Servicio telegraf corriendo en segundo plano

Para controlar el servicio, use los siguientes comandos:

Iniciar el bróker:

```
sudo systemctl start telegraf
```

Parar el bróker:

```
sudo systemctl stop telegraf
```

Reiniciar el bróker:

```
sudo systemctl restart telegraf
```

Iniciar el bróker en el arranque:

```
sudo systemctl enable telegraf
```

5.4. Instalación de Grafana

La instalación de *Grafana* debe hacerse desde el repositorio oficial de la compañía. Para ello ejecute en orden los siguientes comandos (se incluyen dependencias del servicio):

1. sudo apt-get install -y apt-transport-https
2. sudo apt-get install -y software-properties-common wget
3. sudo wget -q -O /usr/share/keyrings/grafana.key
<https://packages.grafana.com/gpg.key>
4. echo "deb [signed-by=/usr/share/keyrings/grafana.key]
<https://packages.grafana.com/enterprise/deb> stable main" |
sudo tee -a /etc/apt/sources.list.d/grafana.list Bashsudo
apt-get install grafana-enterprise
5. sudo apt-get update
6. sudo apt-get install grafana-enterprise

Para ver el estado del servicio ejecute el siguiente comando:

```
sudo systemctl status grafana
```

Debería ver algo parecido a lo que se muestra en la figura B.25.

```
ubuntu@ubuntu:~$ sudo systemctl status grafana-server
● grafana-server.service - Grafana instance
  Loaded: loaded (/lib/systemd/system/grafana-server.service; enabled; vendor preset: enabled)
  Active: active (running) since Mon 2022-08-08 16:24:52 UTC; 32min ago
    Docs: http://docs.grafana.org
   Main PID: 1696 (grafana-server)
      Tasks: 15 (limit: 4435)
     CGroup: /system.slice/grafana-server.service
             └─1696 /usr/sbin/grafana-server --config=/etc/grafana/grafana.ini --pidfile=/run/grafana/grafana-server.pid --packaging=deb cfg:defa>

Aug 08 16:53:37 ubuntu grafana-server[1696]: t=2022-08-08T16:53:37+0000 lvl=info msg="Starting DB migrations" logger=migrator
Aug 08 16:53:37 ubuntu grafana-server[1696]: t=2022-08-08T16:53:37+0000 lvl=info msg="migrations completed" logger=migrator performed=0 skipped=4>
Aug 08 16:53:37 ubuntu grafana-server[1696]: t=2022-08-08T16:53:37+0000 lvl=info msg="Validated license token" logger=licensing appURL=http://loc>
Aug 08 16:53:37 ubuntu grafana-server[1696]: t=2022-08-08T16:53:37+0000 lvl=info msg="Starting plugin search" logger=plugins
Aug 08 16:53:38 ubuntu grafana-server[1696]: t=2022-08-08T16:53:38+0000 lvl=info msg="Registering plugin" logger=plugins id=input
Aug 08 16:53:38 ubuntu grafana-server[1696]: t=2022-08-08T16:53:38+0000 lvl=info msg="Registering plugin" logger=plugins id=briangann-gauge-panel
Aug 08 16:53:38 ubuntu grafana-server[1696]: t=2022-08-08T16:53:38+0000 lvl=info msg="Live Push Gateway initialization" logger=live.push_http
Aug 08 16:53:38 ubuntu grafana-server[1696]: t=2022-08-08T16:53:38+0000 lvl=info msg="Writing PID file" logger=server path=/run/grafana/grafana-s>
Aug 08 16:53:38 ubuntu grafana-server[1696]: t=2022-08-08T16:53:38+0000 lvl=warn msg="Scheduling and sending of reports disabled, SMTP is not con>
Aug 08 16:53:38 ubuntu grafana-server[1696]: t=2022-08-08T16:53:38+0000 lvl=info msg="HTTP Server Listen" logger=http.server address=[::]:3000 pr>
ubuntu@ubuntu:~$
```

Figura B.25. Servicio Grafana corriendo en segundo plano

Una vez que tengamos el servicio corriendo, por defecto tendrá habilitado un servidor web alojado en el puerto 3000 con la interfaz gráfica de usuario. Para acceder, diríjase a la URL 192.168.1.5:3000.

Una vez en la aplicación web siga las instrucciones para definir las credenciales de esta. Al acceder encontrará una ventana como se muestra en la figura B.26

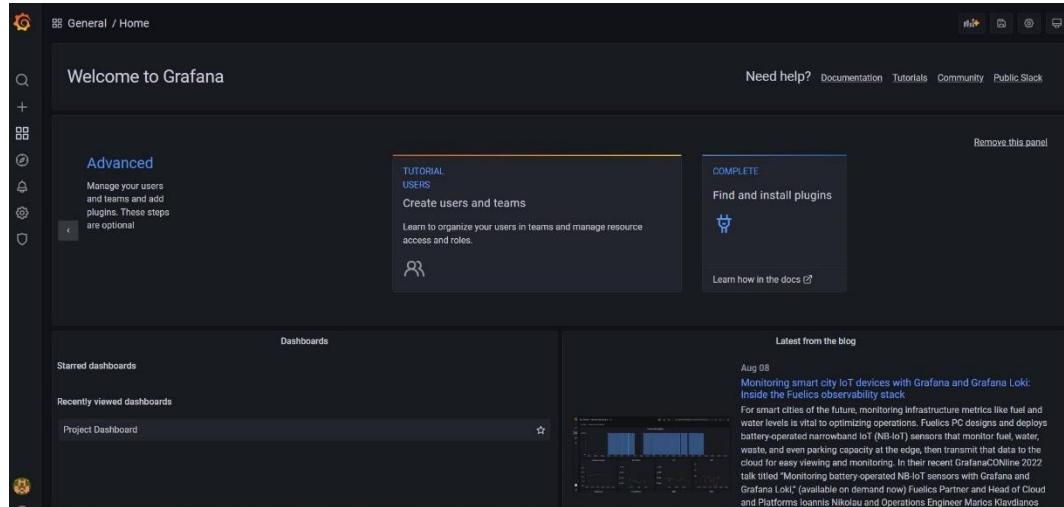


Figura B.26. Venta principal de *Grafana*

Diríjase a la ventana de configuración para añadir una nueva base de datos. Esta ventana se localiza como se señala en la figura B.27 a continuación.

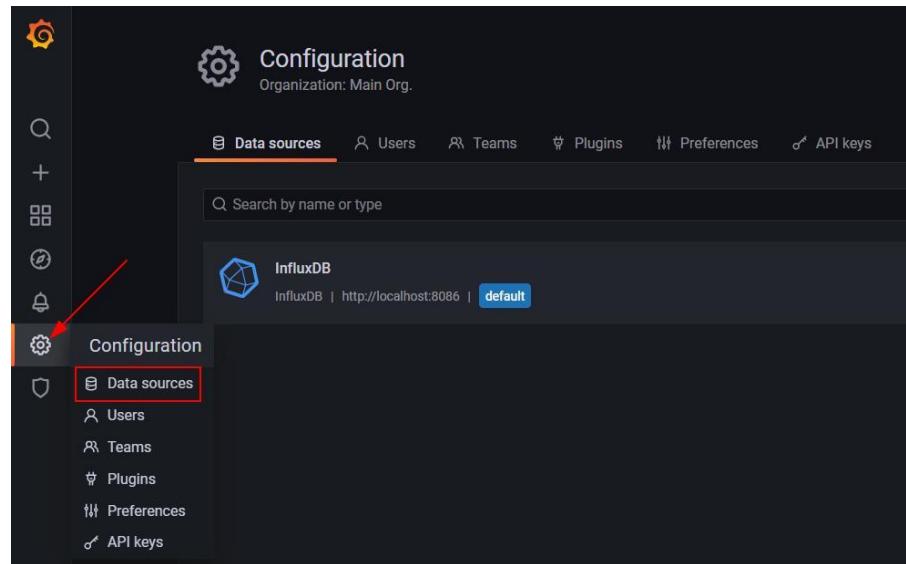


Figura B.27. Configuración de fuentes de datos en *Grafana*

Pinche en “Add data source” y seleccione la opción *InfluxDB* como se muestra en la figura B.28.

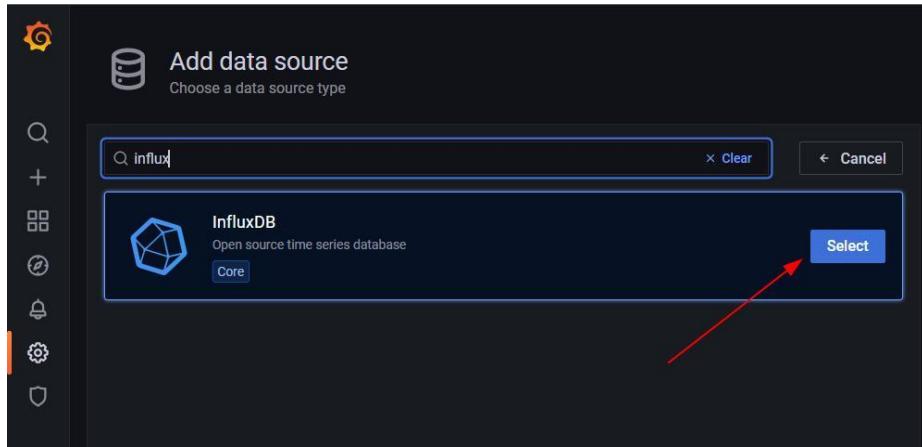


Figura B.28. Selección de fuente de datos en *Grafana*

Seleccione después la información necesaria de la base de datos. En este caso, la base de datos y *Grafana* coexisten en la misma máquina con la misma IP, de esta manera en la IP se puede colocar “*localhost*” como se muestra en la figura B.29. El puerto de *InfluxDB* es 8086 por defecto.

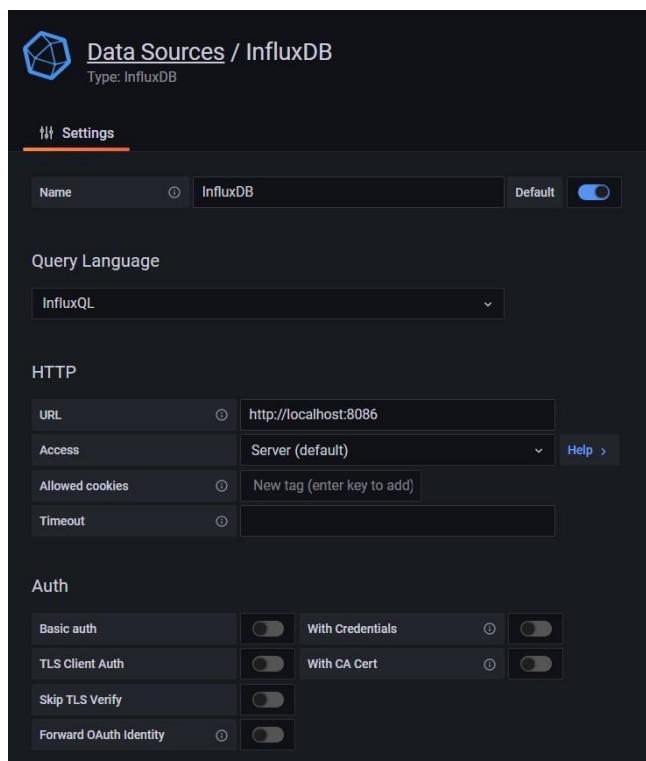


Figura B.29. Configuración de *InfluxDB* en *Grafana*

Introduzca el nombre de la base de datos definida en el archivo de configuración de *Telegraf* así como las credenciales de la misma. Además, añada “*GET*” en el apartado *HTTP Method*. Estos parámetros van como se muestra en la figura B.30.

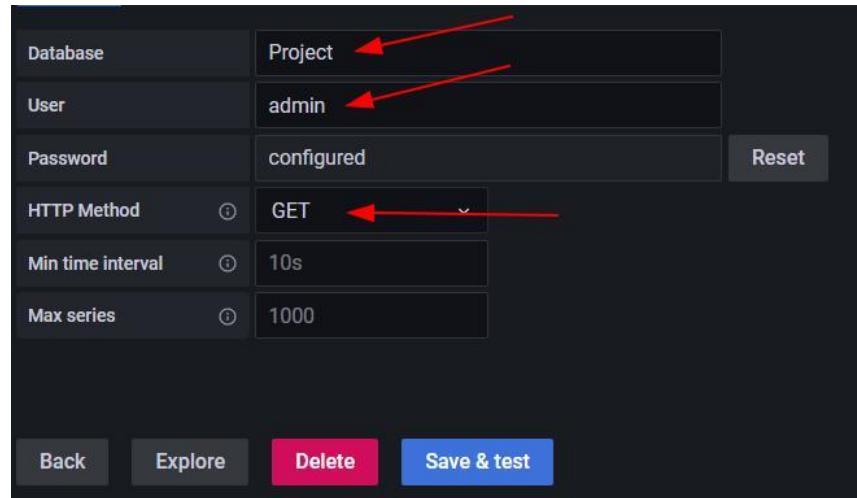


Figura B.30. Configuración de base de datos en *Grafana*

Ahora ya puede utilizar parámetros de la base de datos para poder crear cuadros de mando con históricos con los datos incluidos en dicha base de datos.

Para información adicional sobre funcionalidades de *Grafana*, en caso de que el cliente necesite ayuda con algún tipo de configuración, se adjunta a continuación el manual de uso de *Grafana*. En este manual se hace referencia a la creación de cuadros de mando, así como la configuración de alarmas entre otros.

Apéndice C. Manual de uso de Grafana

Este anexo contiene el manual de uso de *Grafana* para el cliente final y/o el instalador.

1. Inicialización del sistema

Una vez tenga los sensores en la zona deseada y las motas y la *Raspberry Pi* encendidas inicialice su navegador de confianza y diríjase a la siguiente *URL*: **192.168.1.5:3000**. Una vez acceda se encontrará con el interfaz gráfico mostrado en la figura C.1.

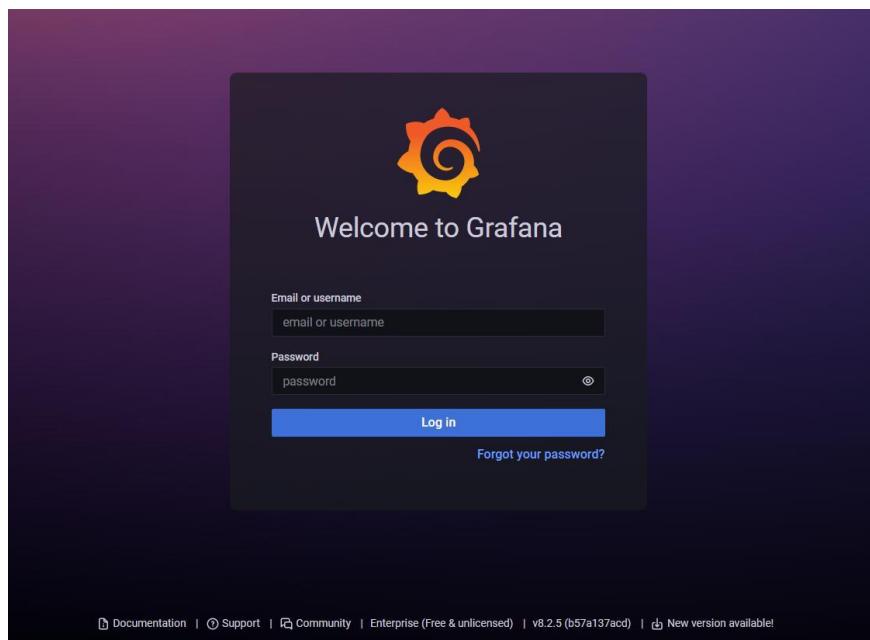


Figura C.1. Servicio web de *Grafana*

Tras introducir las credenciales que le proporcionará el instalador será dirigido a la ventana principal de *Grafana*, visible en la figura C.2.

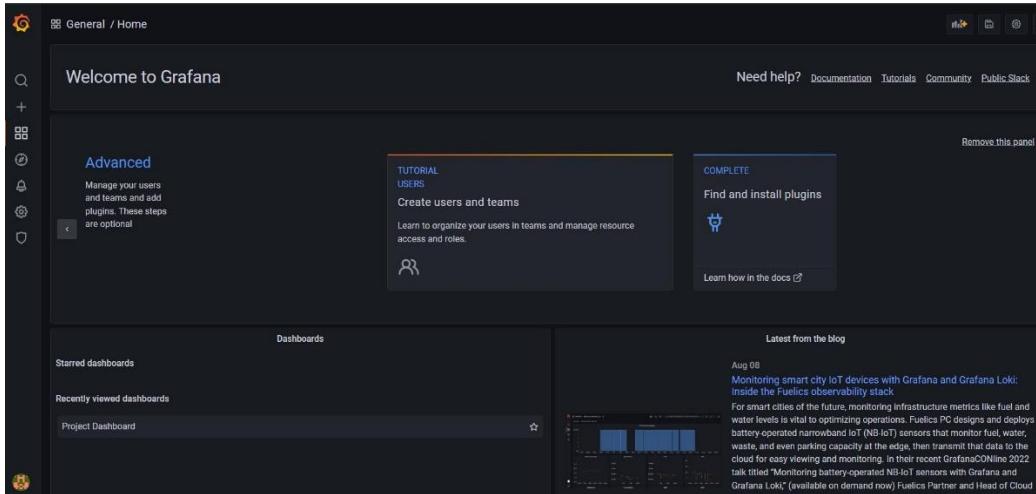


Figura C.2. Ventana principal de *Grafana*

2. Creación de cuadros de mando o *dashboards*

En la ventana nombrada anteriormente diríjase a “*Create*” y seleccione “*Dashboard*” como se muestra en la figura C.3.

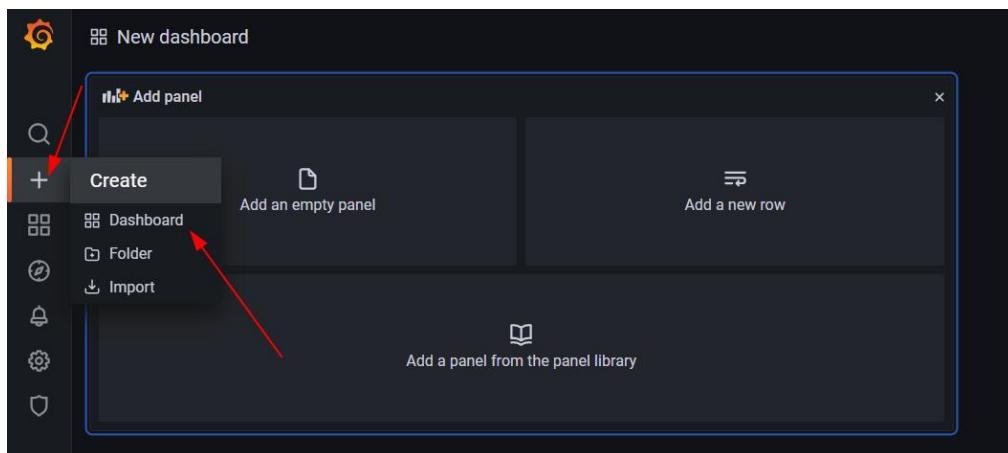


Figura C.3. Creacion de paneles en *Grafana*

Seleccione “*Add an empty panel*” y se mostrará el menú de configuración del panel. En esta nueva ventana, visible en la figura C.4, se observan tres partes destacables: la vista previa del panel, la fuente de datos a mostrar y la personalización gráfica del panel:

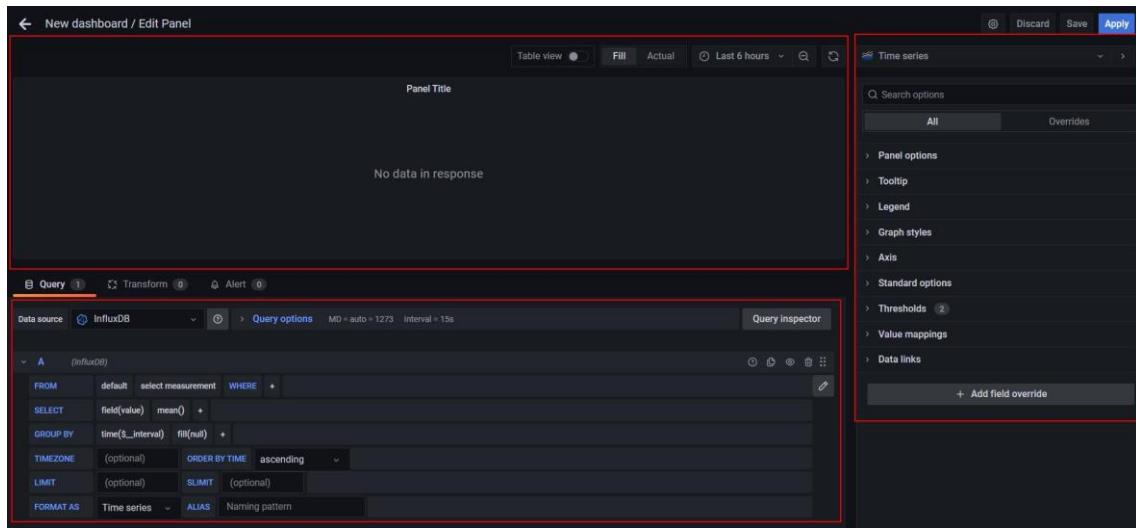


Figura C.4. Configuración general del panel

2.1. Selección de parámetros

Para seleccionar los valores a mostrar en la tabla diríjase al menú inferior. Seleccione en la fila “*FROM*” el parámetro “*select measurements*” y elija la opción “*mqtt_consumer*” como se muestra en la figura C.5.

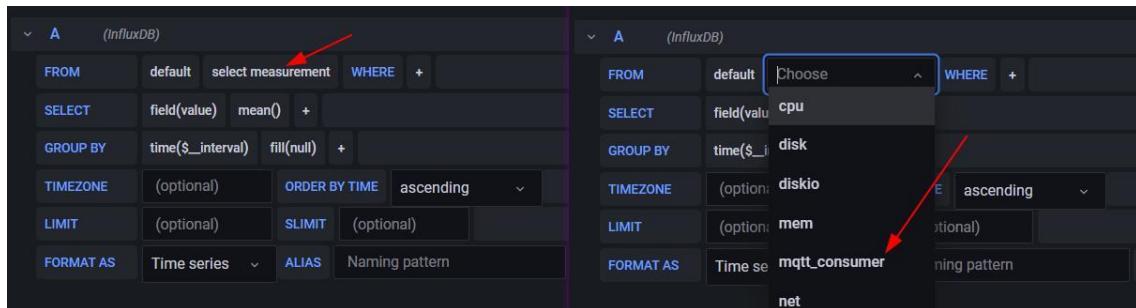


Figura C.5. Selección de medidas de base de datos

En la fila “*SELECT*” pulse en “*field(value)*” y elija uno de los cuatro parámetros guardados en la base de datos como se muestra en la figura C.6.

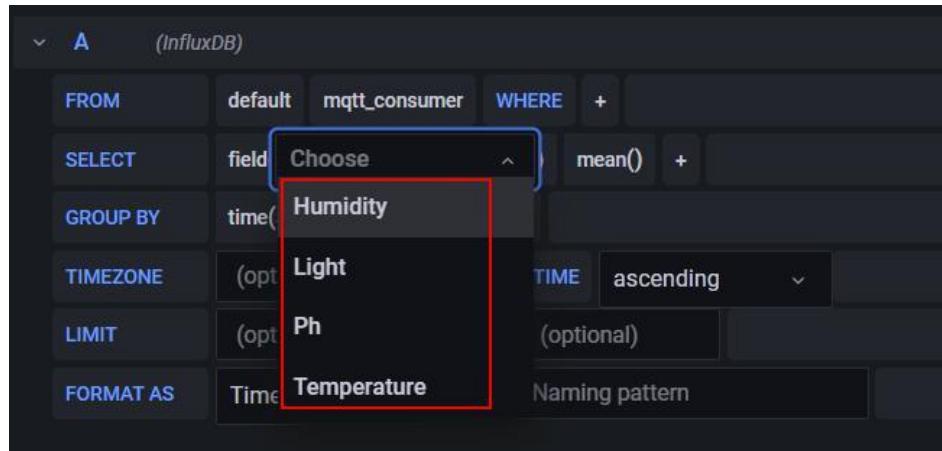


Figura C.6. Selección de campos de base de datos

Una vez elegido verá que en la vista previa se observa una gráfica con los últimos valores guardados en la base de datos. Para ver un rango de tiempo específico, seleccione la barra con el símbolo del reloj encima de la previsualización. Esta barra se localiza donde se muestra en la figura C.7.



Figura C.7. Previsualización de datos en panel

2.2. Personalización del panel

Para la personalización del panel diríjase al menú que puede observar a la derecha de la ventana general como se muestra en la figura C.8.

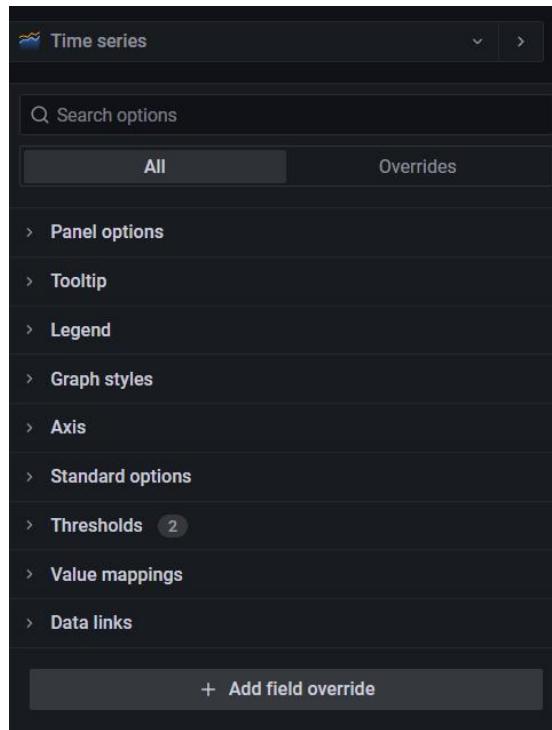


Figura C.8. Personalización de vista de datos

En este menú puede editar la visualización de la gráfica. *Grafana* proporciona además diferentes extensiones para añadir funcionalidad o diferente visualización de los datos. A continuación, se va a detallar cómo mostrar un panel en forma de medidor circular. Para ello, diríjase a la página principal de *Grafana*, seleccione el botón de configuración y pulse “plugins” como se indica en la figura C.9.

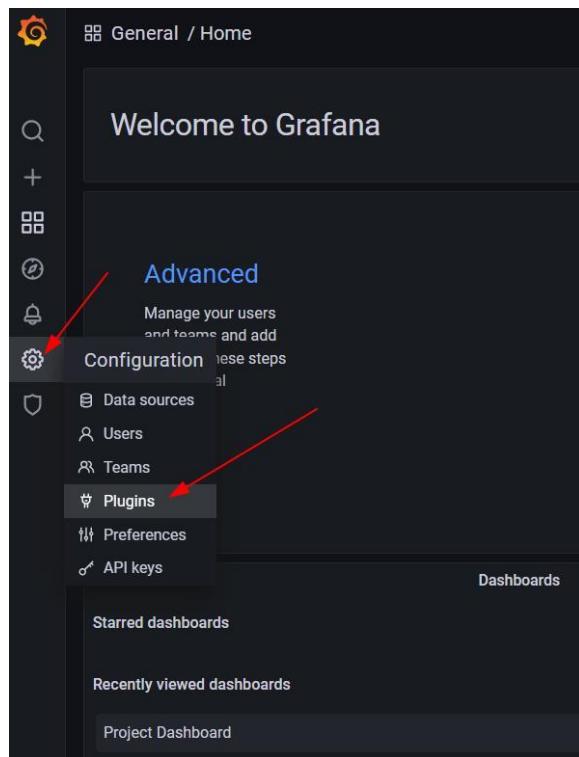


Figura C.9. Configuracion de plugins de *Grafana*

En el buscador escriba “gauge” y elija e instale “D3 Gauge” como se indica en la figura C.10.

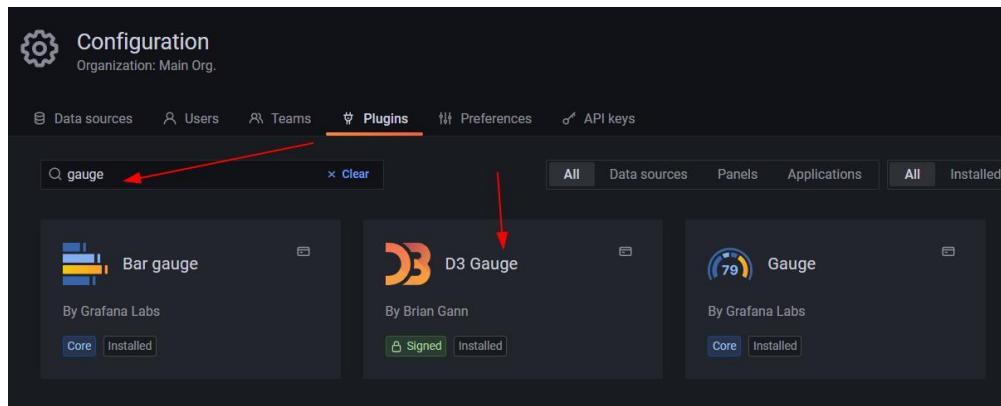


Figura C.10. Búsqueda de plugins en *Grafana*

Vuelva al panel creado anteriormente y en el menú de personalización cambie la visualización de “Time series” a “D3 Gauge” como se indica en la figura C.11.

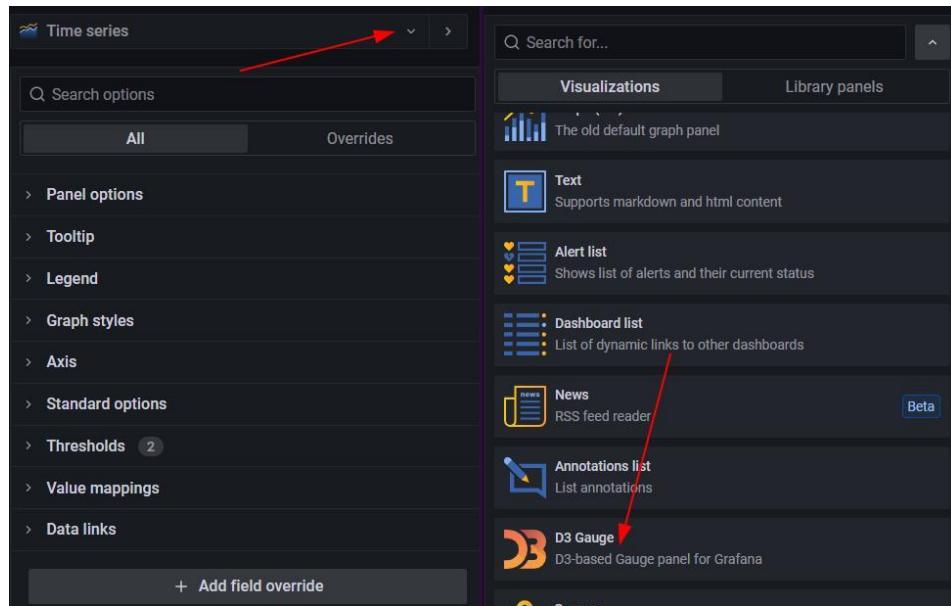


Figura C.11. Selección de plugins en **Grafana**

Por defecto este medidor muestra la media del parámetro que hayamos seleccionado. Para mostrar el último valor leído (y por tanto el más actual) cambiarlo vuelve a dirigirse al menú de parámetros y en la fila “SELECT” elimine el parámetro “mean()” y seleccione “last()” como se indica en la figura C.12.

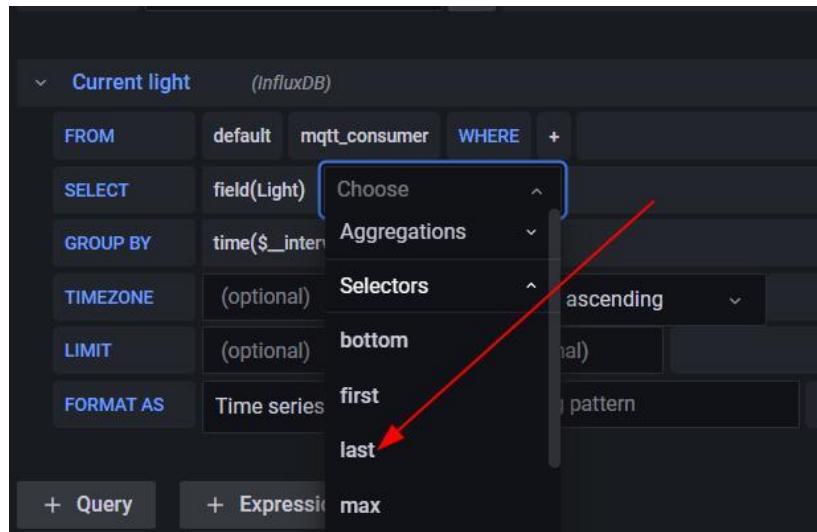


Figura C.12. Personalización de valores mostrados

En el menú de personalización puede elegir colores, rango de muestras, nombre del panel, tamaño de gráficas u objetos de medida y mucho más.

3. Creación de alarmas

El primer paso para crear alarmas es habilitar un canal de notificación. Para ello diríjase, en el menú principal, al símbolo de la campana y seleccione “*Notification channels*” como se muestra en la figura C.13.

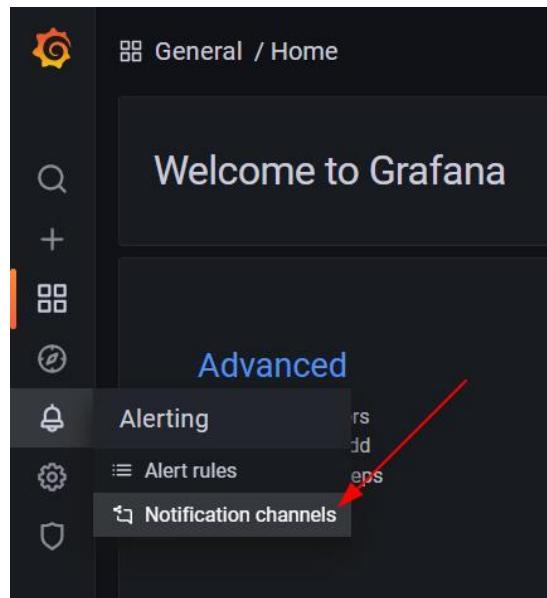


Figura C.13. Configuración de canales de comunicación

Seleccione “*new channel*” y seleccione el tipo de alerta que necesita. *Grafana* permite diferentes notificaciones a diferentes plataformas, como alertas por e-mail, *Telegram* y *Microsoft Teams* entre otros. En la figura C.14 se puede ver el menú desplegable.

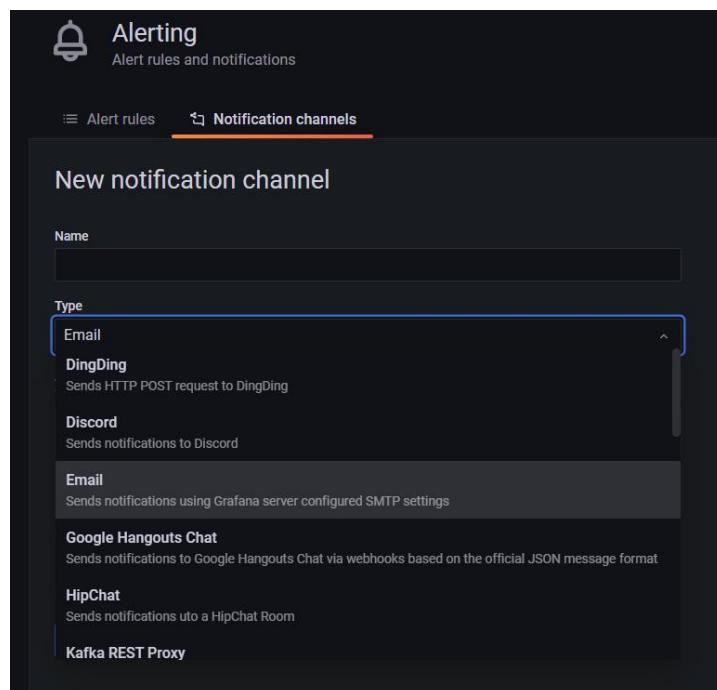


Figura C.14. Adición de nuevos canales de comunicación

Cuando el canal de comunicaciones se haya configurado diríjase, dentro de la edición del panel al que quiere añadirle la alarma, al menú inferior de configuración de datos y seleccionar la pestaña “Alerts” como se indica en la figura C.15.

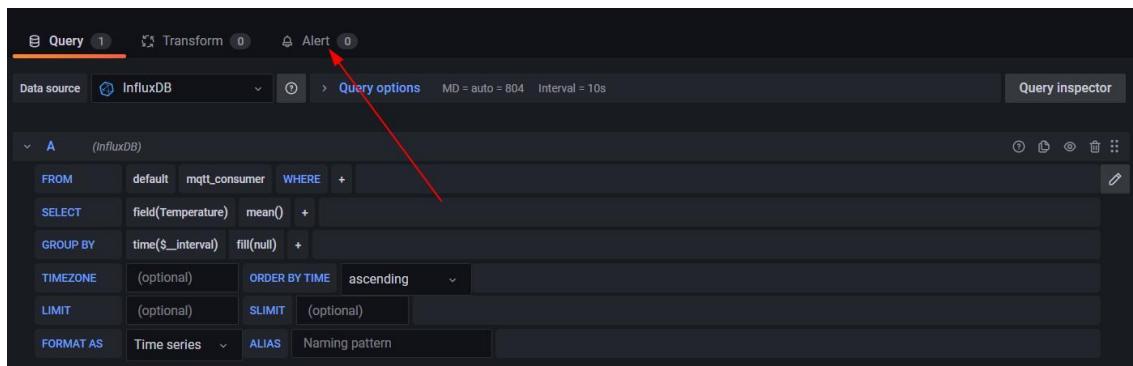


Figura C.15. Menú de alertas de un panel

Tras pinchar en “Create Alert”, seleccione las reglas para que se provoque la alerta como se muestra en la figura C.16. Para generar una alarma en función de un valor leído, seleccione “last()” en la fila “WHEN” dentro del apartado de condiciones. Seleccione después cómo desea configurarla: Por rango, por valor inferior, valor superior ó si no se ha leído ningún valor.

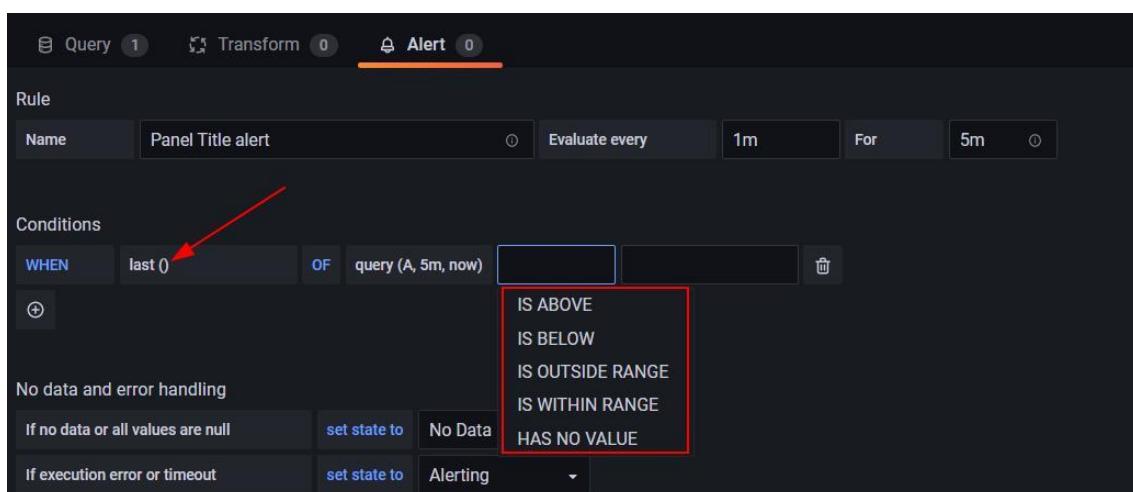


Figura C.16. Configuración de alertas

En el apartado “Notifications”, mostrado en la figura C.17, seleccione el canal previamente creado y el mensaje que quiere enviar junto a la alarma.

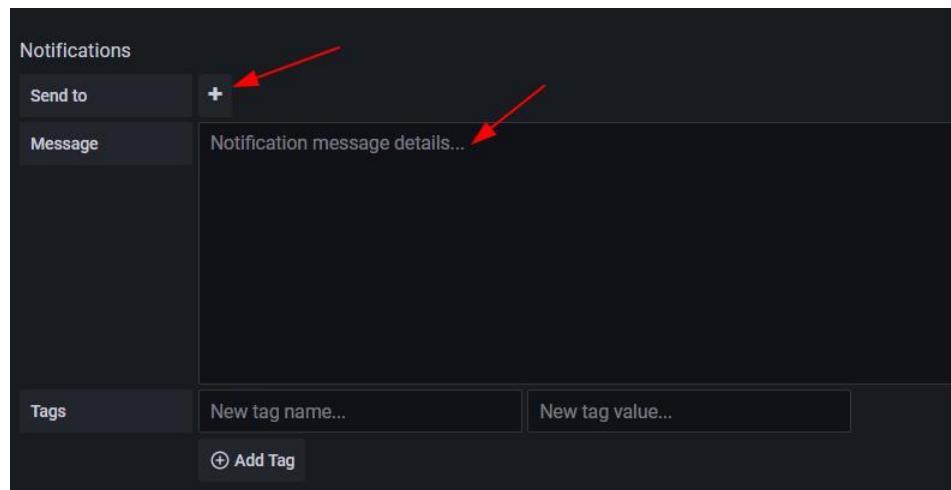


Figura C.17. Personalización de mensajes en alertas

Para finalizar, pulse Test rule para comprobar el correcto funcionamiento de la regla creada. Deberá recibir un mensaje al canal de comunicación seleccionado.