Universidad de Valladolid

Escuela Técnica Superior de Ingenieros de Telecomunicación

Estación meteorológica: recolección de datos

Autor: Andrés Espinel López

Tutores: Jesús Hernández Mangas, Iván Santos Tejido

Julio 2025

Resumen

En este trabajo se ha desarrollado un sistema para la recepción, almacenamiento y visualización de datos meteorológicos en tiempo real. La información, procedente de una estación externa, se procesa automáticamente y queda registrada con marcas temporales, permitiendo su consulta desde un panel gráfico accesible a través de la red local. Además, el sistema puede ser accedido de forma remota, lo que facilita su consulta desde ubicaciones externas. Está diseñado para ser robusto, seguro y fácilmente replicable en otros entornos.

Abstract

This work presents the development of a system for receiving, storing, and visualizing data from a meteorological station in real time. The information, collected from an external source, is automatically processed and stored with timestamps, making it available through a graphical dashboard. The system is accessible from the local network and can also be reached externally through public IP configuration or port forwarding. It is designed to be robust, secure, and easily replicable in other environments.

Keywords: meteorological station, real-time monitoring, MQTT, Raspberry Pi, InfluxDB, Grafana, time-series data, Docker.

Agradecimientos

Quiero expresar mi agradecimiento a mis tutores, Jesús Hernández Mangas e Iván Santos Tejido, por su dedicación, orientación y disponibilidad a lo largo de todo el desarrollo del proyecto.

También quiero dedicar unas palabras a quienes me han acompañado personalmente durante este camino.

A mi madre, por haberme dado siempre la oportunidad de estudiar, por su esfuerzo constante y por enseñarme, con su ejemplo, el valor del trabajo y la perseverancia.

A mi pareja, Adriana, por acompañarme siempre, por su cariño incondicional, su apoyo constante y por no soltarse nunca de mi mano.

A mi mejor amigo, Adrián, por compartir conmigo una amistad eterna y estar siempre cuando más lo he necesitado.

Y a mí, por hacer de estos años algo de lo que estar orgulloso.

Índice

1. Introducción

El monitoreo de variables meteorológicas se ha convertido en una herramienta fundamental para numerosos sectores. La aparición de tecnologías orientadas al Internet de las Cosas (IoT) ha facilitado el desarrollo de plataformas que permiten recopilar, almacenar y visualizar este tipo de datos en tiempo real de forma eficiente.

En este contexto, el presente trabajo tiene como finalidad diseñar una solución ligera y modular para la recepción y gestión de datos meteorológicos, haciendo uso de herramientas modernas como MQTT, InfluxDB, Grafana y contenedores Docker, todo desplegado sobre una Raspberry Pi.

Cabe destacar que la estación meteorológica utilizada como fuente de datos en este proyecto fue desarrollada en paralelo como parte de otro Trabajo de Fin de Grado independiente, realizado por un estudiante del mismo centro. Ambos proyectos fueron coordinados para garantizar la compatibilidad entre el sistema de adquisición y la plataforma de gestión y visualización.

Nota sobre el uso de herramientas de asistencia en la redacción:
Durante la elaboración de este Trabajo de Fin de Grado, se utilizó la herramienta ChatGPT (modelo GPT-4 de OpenAI) como apoyo puntual para mejorar la redacción formal de algunos apartados. El contenido técnico, las conclusiones y las ideas presentadas han sido desarrolladas por el autor a partir de la consulta de documentación especializada, páginas oficiales de las tecnologías empleadas y otros recursos relevantes. Posteriormente, dichas ideas se trasladaban a la herramienta mediante instrucciones verbales o escritas para recibir propuestas de redacción académica, que fueron siempre revisadas, editadas y adaptadas por el propio autor antes de su incorporación al documento final.

1.1. Objetivo del proyecto

El objetivo principal de este proyecto es desarrollar una plataforma funcional para la gestión de datos meteorológicos en tiempo real. Dicha plataforma debe ser capaz de recibir la información generada por una estación meteorológica externa, almacenarla con marcas temporales y ofrecer una visualización clara y accesible desde distintos dispositivos a través de una interfaz web.

1.1.1. Objetivos específicos

- Integrar un broker MQTT para recibir los datos emitidos por la estación meteorológica.
- Diseñar un script Python capaz de interpretar los mensajes y almacenarlos en una base de datos.

- Configurar una base de datos orientada a series temporales (InfluxDB) para guardar los registros.
- Implementar un sistema de visualización mediante Grafana.
- Orquestar todos los servicios mediante Docker y Docker Compose.
- Asegurar el acceso remoto seguro.
- Facilitar la replicabilidad y restauración del sistema mediante una copia comprimida del entorno completo.

2. Estado del arte

En esta sección se describen las tecnologías principales sobre las que se fundamenta el desarrollo del sistema propuesto. Se analizan las soluciones actuales para la monitorización meteorológica, así como las herramientas específicas empleadas en este proyecto.

2.1. Sistemas meteorológicos existentes

Los sistemas de monitorización meteorológica están ampliamente implantados en sectores como la agricultura, la investigación climática, la gestión medioambiental o la domótica. Muchos de estos sistemas emplean sensores conectados a plataformas centralizadas, que permiten registrar y consultar parámetros como temperatura, humedad, presión atmosférica, velocidad del viento, precipitación o radiación solar.

Existen soluciones comerciales como *Davis Instruments* [1] o *Netatmo* [20], así como redes colaborativas como *Meteoclimatic* [18] o *Weather Underground* [25], que permiten integrar estaciones domésticas o profesionales en una red global. Estas plataformas ofrecen interfaces web para consultar históricos, comparar regiones o exportar datos, aunque generalmente son cerradas o de difícil personalización.

2.2. Protocolos de transmisión de datos en entornos IoT

Existen múltiples protocolos diseñados para transmitir datos de forma eficiente entre dispositivos. A continuación, se describen brevemente algunos de los más relevantes para sistemas de monitorización como el desarrollado en este proyecto.

- **MQTT**: protocolo ligero basado en el modelo *publish/subscribe*. Es ideal para redes inestables, dispositivos con recursos limitados y transmisión en tiempo real [21].
- **CoAP**: protocolo basado en el modelo cliente-servidor, muy optimizado para dispositivos embebidos y entornos con restricciones severas de energía y conectividad [27].
- **HTTP/HTTPS**: ampliamente adoptado, fácil de implementar, pero menos eficiente para transmisión continua de datos. Su modelo solicitud-respuesta no se adapta tan bien a flujos en tiempo real [26].
- **WebSocket**: útil cuando se necesita una conexión bidireccional persistente, como en interfaces web interactivas. Más pesado que MQTT pero válido para algunos escenarios [13].

Aunque la elección de MQTT formaba parte de los requisitos iniciales del proyecto, se valoraron otras alternativas durante la fase de diseño, como CoAP o WebSocket. En particular, **CoAP** podría haber sido una opción válida por su bajo consumo y su idoneidad en dispositivos embebidos. Sin embargo, se descartó en favor de **MQTT** debido a la mayor

disponibilidad de bibliotecas, documentación y herramientas integradas en el ecosistema de desarrollo (como Python y Docker), así como su amplia adopción en plataformas de monitorización. Además, el modelo de comunicación *publish/subscribe* de MQTT encajaba de forma más natural con la arquitectura modular y desacoplada que se pretendía implementar en este proyecto.

2.3. Bases de datos temporales

En sistemas donde se recopilan mediciones de forma continua, como ocurre en una estación meteorológica, es fundamental utilizar una base de datos capaz de gestionar series temporales de forma eficiente. Existen diversas alternativas, cada una con características que las hacen más o menos adecuadas para este tipo de aplicaciones.

- **MySQL**: Es una base de datos relacional tradicional ampliamente conocida. Aunque puede utilizarse para almacenar datos con marcas temporales, no está optimizada para grandes volúmenes de inserciones rápidas ni consultas agregadas por ventanas de tiempo. Su uso puede complicar el diseño si se requiere rendimiento en tiempo real [22].
- **MongoDB**: Base de datos NoSQL orientada a documentos, conocida por su flexibilidad y escalabilidad. A partir de versiones recientes, incluye colecciones optimizadas para datos temporales, lo que permite almacenar mediciones con marcas de tiempo de forma más eficiente que en versiones anteriores [19]. No obstante, su arquitectura no está específicamente diseñada para trabajar con grandes volúmenes de datos cronológicos de forma intensiva, por lo que puede requerir configuraciones adicionales para alcanzar un rendimiento óptimo en escenarios de monitorización continua.
- **InfluxDB**: Diseñada específicamente para almacenar y consultar datos temporales. Ofrece una sintaxis de consulta orientada a series de tiempo, un motor optimizado para inserciones rápidas y funciones estadísticas integradas. Se integra fácilmente con herramientas como Grafana y puede desplegarse en contenedores, lo que la hace ideal para arquitecturas modulares como la de este proyecto [14].

Aunque no se estableció una base de datos concreta como requisito inicial del TFG, se valoraron distintas opciones en función de su rendimiento, facilidad de integración y orientación a datos temporales. Finalmente, se eligió **InfluxDB** por su especialización, rendimiento en sistemas de monitorización y su perfecta integración con el resto de herramientas utilizadas en el proyecto.

2.4. Visualización de datos

La representación visual de los datos es una parte esencial en cualquier sistema de monitorización. Existen diversas herramientas que permiten construir paneles interactivos a partir de diferentes fuentes de datos, facilitando la interpretación y el análisis en tiempo real.

- **Tableau**: Ampliamente utilizada en entornos empresariales por su potencia y facilidad de uso, aunque requiere licencia y está más orientada a entornos corporativos que a proyectos ligeros o autoalojados [24].
- **Kibana**: Diseñada principalmente para trabajar con Elasticsearch, ofrece dashboards avanzados con foco en el análisis de logs y métricas. No está específicamente pensada para bases de datos de series temporales como InfluxDB [5].
- **Grafana**: Plataforma open source especializada en visualización de datos en tiempo real. Soporta múltiples fuentes, como InfluxDB, Prometheus o PostgreSQL, y permite crear dashboards dinámicos, con alertas, filtros y opciones interactivas [12]. Su interfaz es intuitiva y su despliegue en contenedores facilita la integración en arquitecturas modulares.

En este proyecto se ha elegido **Grafana** por su orientación clara a series temporales, su compatibilidad nativa con InfluxDB y su enfoque open source, lo que permite un despliegue ligero y personalizable.

2.5. Contenedores y despliegue: Docker y Docker Compose

En sistemas compuestos por múltiples servicios, como los que se encuentran en entornos IoT, resulta fundamental contar con una solución que facilite el despliegue y la gestión del software de forma modular, escalable y reproducible. El uso de contenedores permite encapsular cada componente junto con sus dependencias, asegurando su correcto funcionamiento independientemente del sistema operativo anfitrión.

- **Docker y Docker Compose**: Permiten definir, construir y ejecutar contenedores de manera sencilla y eficiente. Docker facilita el empaquetado de servicios como bases de datos, brokers de mensajería o scripts, mientras que Docker Compose permite orquestar a todos, simplificando el despliegue del sistema completo [3, 2].
- **Podman**: Es una alternativa moderna a Docker que ofrece compatibilidad con su sintaxis y funcionamiento, pero sin necesidad de ejecutar un servicio en segundo plano (*daemon*). Esto lo hace especialmente útil en entornos donde se requiere mayor seguridad o donde no se dispone de permisos elevados. Aunque presenta un enfoque interesante, su adopción y soporte aún son menores en comparación con Docker [23].

Aunque el uso de Docker se planteó desde el inicio como un requisito del Trabajo de Fin de Grado, se valoró brevemente el uso de alternativas como Podman para comprobar si podían aportar ventajas en este entorno. Finalmente, se mantuvo la elección original al confirmar que **Docker** ofrecía una solución madura, ampliamente documentada y perfectamente compatible con las tecnologías utilizadas en el proyecto, como InfluxDB, Grafana o Mosquitto.

3. Arquitectura del sistema

3.1. Visión general

El sistema desarrollado se basa en una arquitectura modular desplegada sobre una Raspberry Pi 4B. Está compuesto por varios servicios independientes (broker MQTT, base de datos, visualización y script de procesamiento), cada uno ejecutado como contenedor Docker, lo que proporciona una solución flexible, portable y fácilmente replicable.

La Figura 1 muestra una visión general de los componentes principales del sistema y la relación entre ellos, representando el flujo de datos desde la recepción hasta la visualización.

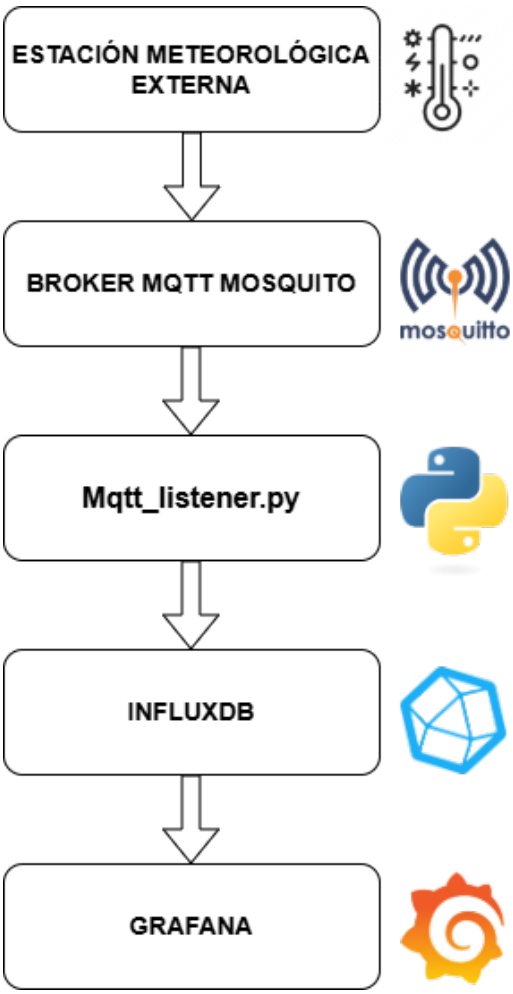

Figura 1: Diagrama de bloques con las tecnologías utilizadas en la arquitectura del sistema.

3.2. Componentes del sistema

3.2.1. Raspberry Pi 4B

La **Raspberry Pi 4B** es un microordenador de bajo consumo y tamaño reducido, ampliamente utilizado en entornos educativos, domésticos e industriales para proyectos de computación embebida e Internet de las Cosas (IoT). Dispone de un procesador de cuatro núcleos Cortex-A72, 4 GB de memoria RAM y múltiples interfaces de entrada/salida [11].

En este proyecto, actúa como el nodo central del sistema. Ejecuta *Ubuntu Server 24.04* y aloja todos los contenedores necesarios para la recolección, almacenamiento y visualización de los datos meteorológicos. Para ello, se instaló una tarjeta microSD de **32 GB**, utilizada como unidad principal para el sistema operativo y la estructura del proyecto.

Figura 2: Raspberry Pi 4B utilizada como núcleo del sistema

3.2.2. Broker MQTT – Mosquitto

El sistema emplea *Mosquitto* [6] como broker MQTT para recibir los mensajes enviados por la estación meteorológica. En el contexto de este protocolo, el broker actúa como el servidor encargado de gestionar la comunicación entre los dispositivos publicadores (sensores) y los suscriptores (clientes que procesan los datos). Este servicio escucha los tópicos asignados a los sensores y reenvía la información al script Python encargado del procesamiento.

MQTT (Message Queuing Telemetry Transport) es un protocolo de mensajería ligero, especialmente adecuado para entornos con recursos limitados, como sistemas embebidos. Su arquitectura se basa en un modelo de comunicación *publish/subscribe*, en el que los dispositivos emisores (publicadores) envían mensajes a determinados tópicos, y los receptores (suscriptores) se registran en esos tópicos para recibirlos. El componente central de esta arquitectura es el *broker*, que actúa como intermediario entre ambos, garantizando una comunicación desacoplada y eficiente.

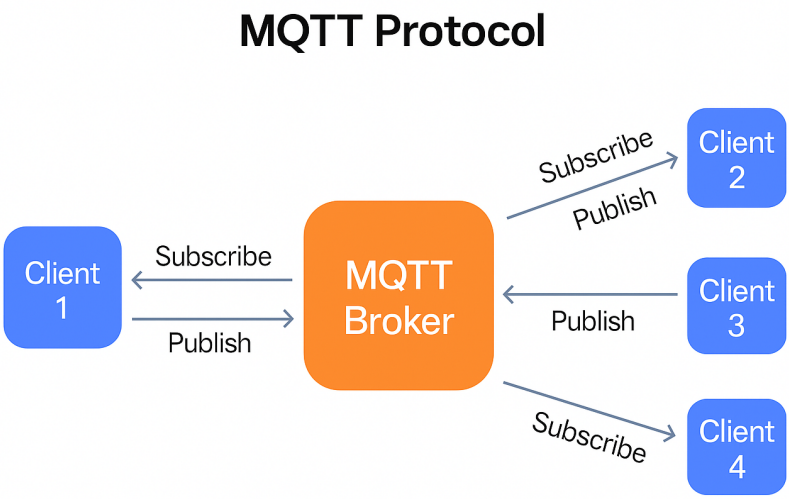

Figura 3: Esquema de comunicación del protocolo MQTT.

Mosquitto se ejecuta como un contenedor Docker independiente, lo que facilita su despliegue, mantenimiento y aislamiento con respecto al resto de componentes.

3.2.3. Script Python de escucha y procesamiento

El sistema incorpora un script desarrollado en Python (anexo B) que se encarga de recibir, interpretar y almacenar los datos meteorológicos transmitidos mediante MQTT. Este componente actúa como cliente suscriptor: se conecta al broker Mosquitto, escucha los mensajes enviados por la estación externa y decodifica el contenido.

Tras interpretar cada mensaje, el script extrae los valores de los sensores, los mapea a nombres más representativos y los envía a la base de datos InfluxDB. Su ejecución también está automatizada mediante Docker, lo que permite integrarlo fácilmente en el sistema junto al resto de servicios.

3.2.4. Visualización con Grafana

Grafana, ya introducida en el estado del arte, se integra aquí como un componente clave dentro de la arquitectura modular del sistema. Se ejecuta como un contenedor independiente y accede a los datos almacenados en InfluxDB para ofrecer una visualización clara y en tiempo real.

En este sistema, se han diseñado varios paneles que muestran tanto los valores actuales de los sensores como su evolución histórica, además de una tabla resumen con los últimos

datos recibidos. También se han implementado gráficos de series temporales para cada variable meteorológica, como temperatura, humedad, presión o radiación solar, y filtros interactivos que permiten ajustar el intervalo de visualización. Esta estructura facilita el análisis de tendencias y el diagnóstico rápido de las condiciones ambientales.

Grafana se accede desde un navegador web mediante la dirección IP del dispositivo, y el acceso se realiza a través del protocolo HTTPS para garantizar una conexión segura, tanto en red local como desde el exterior mediante el túnel configurado.

3.3. Contenedores Docker y orquestación

Todos los servicios del sistema, incluidos Mosquitto, InfluxDB, Grafana y el script Python, están desplegados como contenedores independientes mediante Docker, lo que garantiza el aislamiento entre componentes, facilita su mantenimiento y permite una alta portabilidad. La orquestación se realiza con Docker Compose, que permite definir y lanzar todos los contenedores de forma conjunta mediante un único archivo de configuración `docker-compose.yml`(anexo A), simplificando así el despliegue y asegurando la replicabilidad del sistema en cualquier entorno compatible.

3.3.1. Estructura de carpetas del proyecto

La arquitectura del sistema se organiza dentro de un directorio principal llamado `estacion-meteorologica/`, el cual contiene la definición de los servicios, sus configuraciones correspondientes y el código fuente del script encargado de recibir y procesar los datos. Esta estructura permite una separación clara entre los distintos componentes y facilita el mantenimiento y despliegue del sistema. La organización de carpetas es la siguiente:

- `estacion-meteorologica/`
  - `docker/`
    - `grafana/`
      - `data/`
      - `certs/`
    - `influx/`
      - `data/`
      - `config/`
    - `mosquitto/`
      - `data/`
      - `config/`
  - `scripts/`
    - `mqtt_listener.py`
  - `docker-compose.yml`

3.3.2. Fichero docker-compose.yml

El archivo `docker-compose.yml` define y orquesta todos los servicios que conforman el sistema: Mosquitto, InfluxDB, Grafana y el script Python. Cada servicio se configura de manera independiente, especificando su imagen, red compartida y volúmenes asociados para garantizar la persistencia de datos. Esta configuración centralizada permite desplegar todo el entorno mediante un único comando, lo que facilita su replicación y mantenimiento.

3.4. Flujo de datos en el sistema

El sistema sigue un flujo de datos estructurado y optimizado para la adquisición, el procesamiento y la visualización en tiempo real de variables meteorológicas. A diferencia del resumen visual presentado anteriormente, en este apartado se describen con más detalle los mecanismos técnicos implicados en cada etapa del flujo:

1. La estación meteorológica externa transmite los datos en formato `JSON` a través del protocolo MQTT, publicando en tópicos previamente establecidos. Estos tópicos fueron definidos de forma conjunta con el estudiante responsable del desarrollo de la estación meteorológica, garantizando la compatibilidad entre ambos sistemas. Los mensajes contienen abreviaturas de cada parámetro medido (por ejemplo, `"t": 22.3` para temperatura).

2. El broker Mosquitto, funcionando como servicio centralizado, actúa como servidor MQTT y recibe los mensajes publicados. Su función es distribuirlos a todos los clientes suscritos sin almacenar los datos, implementando un modelo desacoplado *publisher/subscriber*.

3. Un script Python suscrito a esos tópicos interpreta cada mensaje entrante, lo valida y lo transforma en un conjunto de campos etiquetados. Estos se almacenan en InfluxDB como registros individuales (tipo *Point*) dentro del bucket `estacion x, con su respectiva marca temporal en UTC.`

4. Grafana accede a InfluxDB mediante consultas de tipo Flux, configuradas previamente en los dashboards. Cada panel visual extrae métricas específicas, permite aplicar filtros por intervalo temporal y muestra los datos agregados o en tiempo real. Esta visualización puede realizarse desde un navegador web tanto en red local como de forma remota.

Este flujo modular y desacoplado permite que cada componente pueda ser modificado o sustituido sin afectar al resto del sistema, lo que favorece la escalabilidad y el mantenimiento.

4. Desarrollo del sistema

Este capítulo detalla el proceso de implementación del sistema completo, abarcando desde la configuración inicial del entorno hasta el despliegue de los distintos servicios y el desarrollo del script de procesamiento de datos meteorológicos. Además, se incluyen aspectos fundamentales como la conexión con la base de datos, la creación del panel de visualización, la gestión de usuarios, la configuración del acceso remoto y el sistema de copias de seguridad. Todo ello ha sido diseñado con el objetivo de ofrecer una solución modular, segura y fácilmente replicable en otros entornos.

4.1. Configuración del entorno

Para instalar el sistema en la Raspberry Pi 4B, se eligió el sistema operativo Ubuntu Server 24.04 LTS, disponible en el sitio oficial de Ubuntu para dispositivos Raspberry Pi [17]. La instalación se llevó a cabo utilizando la aplicación Raspberry Pi Imager, que permite grabar fácilmente la imagen del sistema en una tarjeta microSD. Durante este proceso, es necesario seleccionar el modelo de Raspberry Pi, el sistema operativo deseado y la unidad de almacenamiento (en este caso, una tarjeta SD de 32Gb).

Figura 4: Proceso de escritura del sistema operativo Ubuntu Server 24.04 en la tarjeta SD

Una de las funcionalidades destacadas de esta herramienta es la posibilidad de realizar una configuración avanzada antes de grabar la imagen. Entre los parámetros que se pueden definir se encuentran el nombre del dispositivo, las credenciales de acceso (usuario y con-

traseña) y los datos de red Wi-Fi (SSID y clave). Gracias a esto, al iniciar la Raspberry Pi por primera vez, ya se encuentra conectada a la red local y puede ser gestionada de forma remota sin necesidad de monitor, teclado ni ratón.

Figura 5: Configuración avanzada en Raspberry Pi Imager

Para establecer la conexión remota, se utilizó el cliente SSH PuTTY, que permite acceder al sistema introduciendo la dirección IP local del dispositivo.

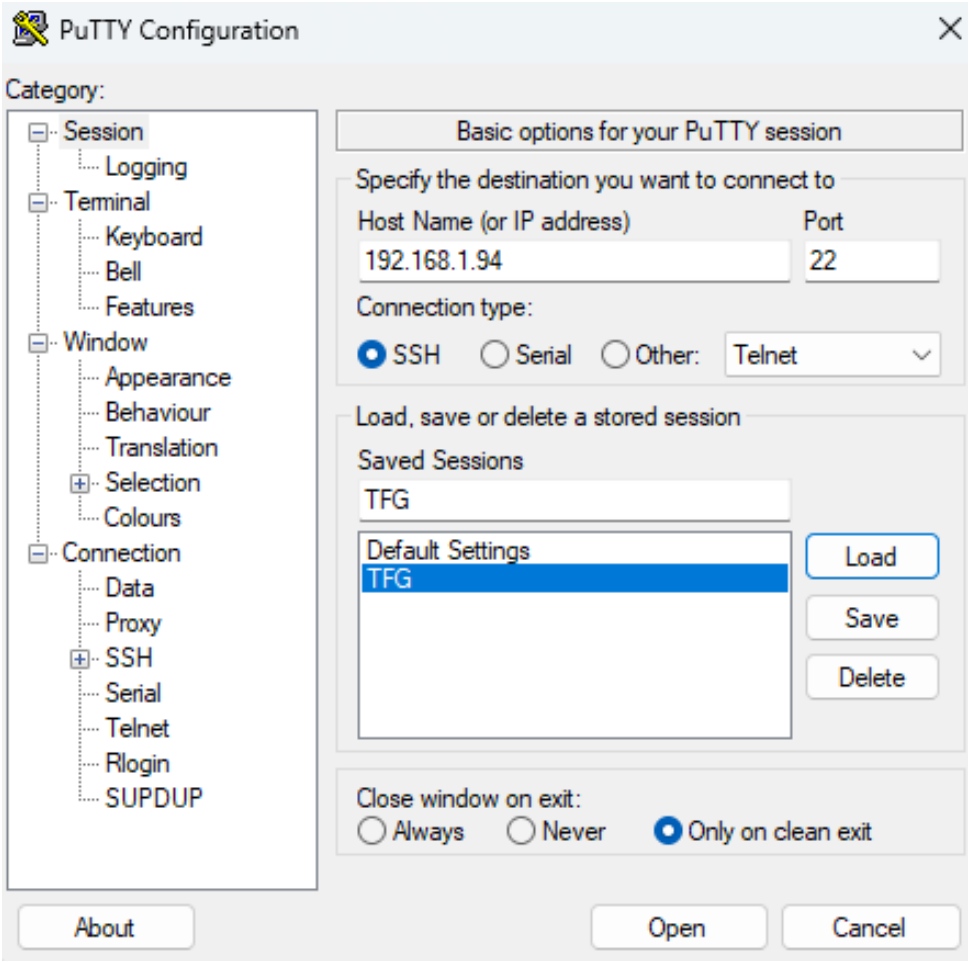

Figura 6: Conexión SSH a la Raspberry Pi desde PuTTY usando su dirección IP local

Una vez iniciada la sesión, se procedió a actualizar el sistema operativo e instalar las herramientas necesarias para la gestión de contenedores: **Docker y Docker Compose**. Finalmente, se organizó la estructura de carpetas del proyecto de forma modular, separando la configuración de los servicios, el script de procesamiento y los volúmenes de datos. Esta organización facilita un desarrollo más limpio, profesional y mantenible (véase la estructura detallada en la Sección 3.3.1).

4.2. Despliegue de servicios con Docker Compose

Una vez configurado el entorno base y organizada la estructura de carpetas del proyecto, se procede al despliegue de los distintos servicios mediante Docker Compose. Esta herramienta permite definir todos los contenedores necesarios desde un único archivo, `docker-compose.yml` (anexo A), lo que facilita una puesta en marcha coordinada y automatizada del sistema.

El fichero incluye la definición de los servicios de Mosquitto, InfluxDB, Grafana y el script Python (cuya integración automatizada se detalla más adelante en la Sección 4.3). Para cada uno de ellos se especifica la imagen Docker correspondiente, los volúmenes persistentes, los puertos de red y las opciones de configuración particulares. El despliegue completo puede iniciarse desde el directorio raíz del proyecto ejecutando el siguiente comando:

Listing 1: Ejecución de Docker Compose

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

Este comando lanza los contenedores en segundo plano, dejándolos activos y listos para ser utilizados de inmediato. El estado de cada uno puede comprobarse mediante:

Listing 2: Verificación del estado de los contenedores

```
docker ps
```

Con ello, el sistema queda desplegado de forma modular, consistente y reproducible, listo para comenzar a recibir, procesar y almacenar los datos provenientes de la estación meteorológica.

4.3. Implementación del script Python

El sistema cuenta con un script escrito en Python, llamado `mqtt_listener.py` Anexo B, que se encarga de recibir los datos enviados por la estación meteorológica a través del protocolo MQTT, procesarlos y almacenarlos en la base de datos.

En los siguientes apartados se explica con detalle cómo está estructurado este script y qué funciones desempeña cada parte.

4.3.1. Configuración del cliente y parámetros iniciales

El script comienza cargando las librerías necesarias para su funcionamiento. En algunos casos se usan alias para facilitar su uso posterior en el código, como es el caso de `import paho.mqtt.client as mqtt`. A continuación, se resumen brevemente las funciones de cada módulo:

- `paho.mqtt.client`: librería oficial de Python para implementar clientes MQTT. Permite conectarse al broker, suscribirse a tópicos y gestionar la recepción de mensajes [7].
- `influxdb_client`: conjunto de módulos para interactuar con InfluxDB 2.x. Se emplean `InfluxDBClient` para la conexión y `Point` para crear registros con campos, etiquetas y marcas de tiempo [15].
- `influxdb_client.client.write_api.SYNCHRONOUS`: constante que configura la escritura en modo síncrono, asegurando que cada inserción se complete antes de seguir. Esto garantiza que no se pierdan datos durante el proceso [15].
- `datetime`: módulo estándar de Python para trabajar con fechas y horas, necesario para asignar marcas temporales en formato UTC [8].
- `json`: módulo nativo para decodificar cadenas JSON. Se usa `json.loads()` para convertir los datos recibidos en estructuras manejables dentro del script [9].
- `time`: módulo estándar que permite pausar la ejecución durante un tiempo determinado. En este caso se utiliza para implementar un bucle de reintento en la conexión al broker MQTT, permitiendo que el script espere antes de volver a intentarlo si falla la conexión inicial [10].

Después de importar las librerías, se configuran los parámetros de conexión al broker MQTT. Dado que el servicio Mosquitto se encuentra en la misma Raspberry Pi, se utiliza la dirección `"localhost"` y el puerto estándar `1883`, destinado a conexiones sin cifrado.

El cliente se suscribe al tópico base `/teleco/#`, lo que permite recibir mensajes publicados bajo cualquier subtema dentro de esa ruta. Este enfoque hace que el sistema sea fácilmente ampliable: actualmente solo se usa `/teleco/estacion1`, pero es posible añadir otras estaciones, como `/educacion/estacion1` o `/ciencias/estacion_norte`, simplemente incorporándolas a la lista de tópicos válidos.

También se establece una autenticación básica mediante usuario y contraseña. Aunque en el código real aparecen en texto plano, en esta memoria se ocultan por motivos de seguridad.

Listing 3: Configuración MQTT

```
broker = "localhost"
port = 1883
topic_base = "/teleco/#"

TOPICS_VALIDOS = [
    "/teleco/estacion1",
    "/teleco/estacion2"
]

username = "******"
password = "******"
```

La conexión con InfluxDB se realiza a través de cuatro elementos clave: la URL del servidor, un token de autenticación, el nombre de la organización y el bucket de destino. En este caso, se emplea la dirección local ya que la base de datos también se ejecuta en la Raspberry Pi.

El token se obtiene desde la interfaz de InfluxDB y actúa como credencial segura para autorizar operaciones sobre la base. El campo `org` define la organización en la que se agrupan usuarios y buckets, y el bucket es el contenedor final de los datos meteorológicos. Para este proyecto, se ha creado un bucket llamado `estacion1`.

Listing 4: Configuración InfluxDB

```
influx_url = "http://localhost:8086"
influx_token = "******"
influx_org = "teleco"
influx_bucket = "estacion1"
```

Los datos que llegan desde la estación están en formato JSON y utilizan abreviaturas para cada parámetro medido. Para facilitar su interpretación tanto en la base de datos como en los paneles de visualización, se define un diccionario que traduce estas claves a nombres más descriptivos.

El mapeo está preparado para crecer en caso de incorporar nuevos sensores en el futuro, manteniendo la estructura clara y coherente.

Listing 5: Diccionario de mapeo de claves

```
MAPEO = {
    "t": "temperatura",
    "h": "humedad",
    "pn": "presion",
    "v": "veleta",
    "a": "anemometro",
    "pl": "pluviometro",
    "r": "radiacion"
}
```

4.3.2. Creación del cliente InfluxDB

Tras definir los parámetros de conexión, el siguiente paso consiste en crear una instancia del cliente que permitirá establecer comunicación con InfluxDB. Para ello, se utiliza la clase `InfluxDBClient`, a la que se le proporcionan la URL del servidor, el token de autenticación y el nombre de la organización.

A continuación, se configura el cliente de escritura mediante el método `write_api()`, que se inicializa con la opción `SYNCHRONOUS`. Esta configuración hace que cada escritura se realice de forma inmediata y bloqueante: el script no avanza hasta que se ha confirmado que los datos han sido almacenados con éxito. Esta estrategia resulta adecuada para este proyecto, ya que el volumen de datos es reducido y se prioriza la fiabilidad sobre el rendimiento.

Listing 6: Crear cliente InfluxDB

```
influx_client = InfluxDBClient(url=influx_url, token=influx_token, org=
    influx_org)
write_api = influx_client.write_api(write_options=SYNCHRONOUS)
```

Con el cliente ya inicializado, el script queda listo para construir los objetos de medición y escribir los datos en la base de datos a medida que se reciben los mensajes MQTT, como se detalla en el siguiente apartado.

4.3.3. Recepción de mensajes MQTT

El componente central del script es la función `on_message`, definida como callback y ejecutada automáticamente cada vez que se recibe un mensaje a través del broker MQTT. Esta función se registra en el cliente mediante la instrucción `client.on_message = on_message`, actuando como manejador de eventos.

Nada más activarse, la función extrae el tópico y el contenido del mensaje recibido, decodificando este último desde texto plano. A continuación, se comprueba si el tópico pertenece a la lista de `TOPICS_VALIDOS`; si no es así, el mensaje se descarta directamente. Este mecanismo permite filtrar los mensajes y asegurar que solo se procesen aquellos provenientes de fuentes conocidas o autorizadas.

Cuando se recibe un mensaje válido, se intenta decodificar el contenido desde JSON a un diccionario de Python usando `json.loads()`. Esta transformación permite acceder a cada parámetro transmitido como una variable independiente, facilitando su tratamiento posterior.

El bloque de código siguiente resume todo el proceso descrito:

Listing 7: Recepción de mensajes y decodificación del JSON

```
def on_message(client, userdata, message):
    topic = message.topic
    payload = message.payload.decode()

    if topic in TOPICS_VALIDOS:
        print(f"[MQTT] {topic}: {payload}")

        try:
            datos = json.loads(payload.strip())

            campos = {}
            for clave, valor in datos.items():
                clave_larga = MAPEO.get(clave, clave)
                try:
                    campos[clave_larga] = float(valor)
                except ValueError:
                    print(f"Valor invalido para '{clave_larga}': {valor
                        }")
```

Durante este procedimiento se aplica una gestión básica de errores: si el contenido del mensaje no es un JSON válido, se muestra una advertencia en consola sin detener la ejecución. De igual forma, si algún valor no puede convertirse correctamente a número, se omite dicho dato concreto y se registra el error correspondiente. Gracias a esta lógica, el sistema puede seguir funcionando de forma estable incluso ante entradas incompletas o mal formateadas.

Una vez construida correctamente la estructura `campos`, el siguiente paso consiste en crear el objeto de medición compatible con InfluxDB y almacenarlo.

4.3.4. Inserción de datos en InfluxDB

Una vez transformados los datos recibidos y construida la estructura `campos`, se crea un objeto de tipo `Point`, que representa una medición en InfluxDB. A este objeto se le asigna el nombre `"mediciones"` y se añade una etiqueta (`tag`) que identifica la estación emisora a partir del tópico del mensaje.

También se establece una marca temporal utilizando `datetime.datetime.now(datetime.timezone.utc)`, lo que garantiza que todas las entradas se almacenen en formato UTC. Esto facilita la comparación y análisis posterior de los datos, independientemente del origen geográfico o zona horaria del dispositivo emisor.

A continuación, se recorren los pares clave-valor del diccionario `campos`, añadiendo cada uno como campo dentro del punto. Finalmente, se utiliza el método `write()` del cliente para registrar la medición en el bucket correspondiente de InfluxDB.

Listing 8: Crear punto InfluxDB con zona horaria UTC

```
point = Point("mediciones").tag("estacion", topic).time(datetime.
    datetime.now(datetime.timezone.utc))
for k, v in campos.items():
    point = point.field(k, v)

write_api.write(bucket=influx_bucket, org=influx_org, record=point)
print("Datos guardados en InfluxDB")
```

Con esta operación, el sistema completa todo el ciclo de procesamiento de datos: desde la recepción y decodificación del mensaje hasta su almacenamiento en la base de datos. En el siguiente apartado se describe el funcionamiento global del sistema y las herramientas utilizadas para visualizar los datos.

4.3.5. Configuración del cliente MQTT

Para que el script pueda recibir mensajes en tiempo real, es necesario inicializar y configurar el cliente MQTT. Para ello, se crea una instancia mediante `mqtt.Client()` y se establecen las credenciales de acceso con `username_pw_set()`. A continuación, se registra la función `on_message` como manejador de los mensajes entrantes.

Dado que el script se ejecuta automáticamente mediante `systemd` al arrancar la Raspberry Pi (como se describe en el siguiente subapartado), y que puede iniciarse antes de que el servicio Mosquitto esté disponible, se implementó un bucle de reconexión indefinida. Este mecanismo permite al script intentar conectarse al broker de forma continua cada 5 segundos, reiniciando también la suscripción y el ciclo de escucha en caso de fallo o desconexión. De este modo, se garantiza que el sistema se mantenga operativo sin intervención manual, incluso ante caídas temporales del servicio MQTT.

A continuación se muestra el fragmento del código que implementa esta lógica robusta:

Listing 9: Inicialización del cliente MQTT con reconexión automática

```
# Configurar cliente MQTT
while True:
    try:
        client = mqtt.Client()
        client.username_pw_set(username, password)
        client.on_message = on_message

        client.connect(broker, port)
        client.subscribe(topic_base)
        print(f"Escuchando en {topic_base}...")
        client.loop_forever()  # Se bloquea hasta que se pierde la
            conexion

    except Exception as e:
        print(f"Error durante conexion o ejecucion del loop MQTT: {e}")
        print("Reintentando en 5 segundos...")
        time.sleep(5)
```

Con esta configuración final, el script mqtt_listener.py queda completamente preparado para funcionar de forma autónoma, resiliente y tolerante a las condiciones variables del entorno, asegurando la recepción continua de datos meteorológicos desde el inicio del sistema.

4.3.6. Automatización del script con systemd

Dado que la integración del script mqtt_listener.py mediante Docker presentó ciertos inconvenientes de ejecución, se optó por una solución más sencilla y robusta basada en `systemd`. Este método permite ejecutar el script automáticamente al arrancar la Raspberry Pi, sin depender del entorno de contenedores.

Para ello, se creó un servicio personalizado llamado `mqtt_listener.service`, ubicado en `/etc/systemd/system/`, cuyo contenido se muestra a continuación:

Listing 10: Servicio systemd para lanzar el script Python

```
[Unit]
Description=Script MQTT Listener para estacion meteorologica
After=network-online.target
Requires=docker.service
Wants=network-online.target

[Service]
ExecStartPre=/bin/sleep 30
ExecStart=/usr/bin/python3 /home/teleco/estacion-meteorologica/scripts/
   mqtt_listener.py
WorkingDirectory=/home/teleco/estacion-meteorologica/scripts
Restart=always
User=teleco
```

```
StandardOutput=journal
StandardError=journal
Environment=PYTHONUNBUFFERED=1

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

Con esta configuración, el script queda totalmente automatizado. El servicio se inicia al arrancar la Raspberry Pi, permanece activo en segundo plano y se reinicia automáticamente en caso de fallo. Esta solución asegura que el sistema comience a recibir y almacenar datos meteorológicos sin intervención manual, incluso tras un corte de energía.

La Figura 7 muestra cómo se verifica que el servicio se encuentra activo tras el arranque.

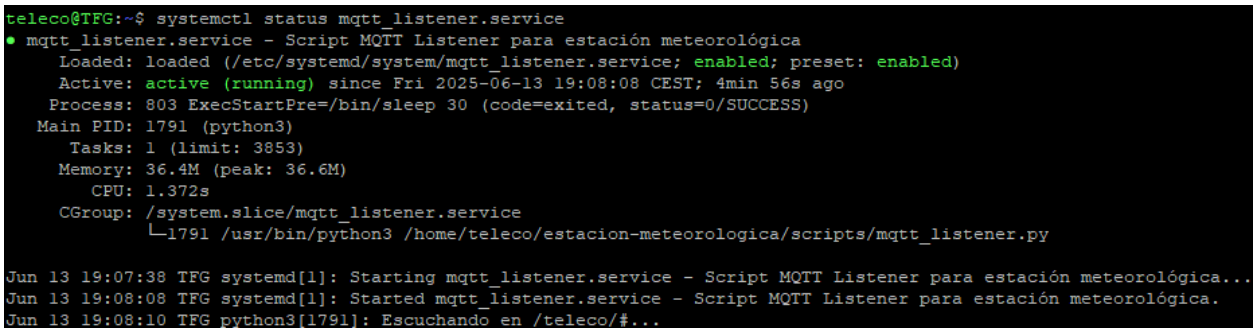

Figura 7: Verificación del servicio activo

4.4. Conexión con InfluxDB

La base de datos InfluxDB se ha desplegado como un servicio independiente dentro del entorno Docker Compose. Para ello, se ha definido un bloque de configuración en el fichero `docker-compose.yml` que especifica la imagen a utilizar, los puertos de acceso, los volúmenes persistentes para los datos y la configuración, así como las variables de entorno necesarias para su inicialización automática.

Gracias a estas variables de entorno, al levantar el contenedor por primera vez, InfluxDB ejecuta un proceso de configuración inicial en el que se crean automáticamente:

- Usuario administrador,
- Organización,
- Bucket principal para almacenar los datos meteorológicos, configurado con retención indefinida (sin límite temporal), a diferencia de los 30 días establecidos por defecto en InfluxDB.
- Token de autenticación con permisos de administración.

Una vez desplegado el servicio, se puede acceder a la interfaz web de InfluxDB desde un navegador, utilizando la dirección `http://localhost:8086` o, si se accede desde otro dispositivo en la misma red, la IP local de la Raspberry Pi seguida del puerto `8086`. Desde esta interfaz se puede iniciar sesión con las credenciales configuradas previamente y acceder al panel de gestión de la base de datos.

En este caso, no fue necesario realizar configuraciones adicionales desde la interfaz web de InfluxDB, ya que toda la estructura base del sistema quedó correctamente establecida de forma automática al levantar el contenedor mediante Docker. Además, todos los valores que aparecen en el panel de consulta de InfluxDB, como las mediciones de sensores o los tags asociados a cada estación, se generan dinámicamente a medida que el script Python envía datos al bucket. InfluxDB detecta de forma automática los campos y etiquetas utilizados en los registros y los incorpora al modelo de datos sin necesidad de intervención manual.

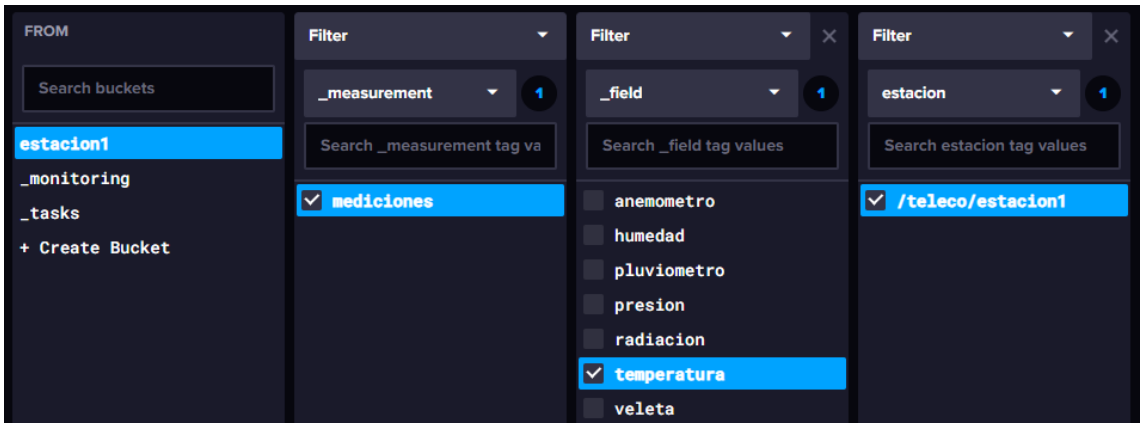

Figura 8: Panel de consulta de InfluxDB con los datos recibidos desde el script Python.

4.5. Dashboard en Grafana

Tras desplegar Grafana como un servicio independiente dentro de la infraestructura Docker del sistema, y confirmar el correcto funcionamiento de InfluxDB mediante las inserciones realizadas por el script `mqtt_listener.py` (véase la Sección 4.3.4), se procedió a configurar la visualización de los datos a través de la interfaz web de Grafana.

4.5.1. Conexión de InfluxDB en Grafana

Antes de importar cualquier dashboard, fue necesario configurar InfluxDB como fuente de datos en Grafana. Esta acción permite que los paneles accedan directamente a las mediciones almacenadas por el sistema. La configuración se realizó desde el menú lateral de Grafana, accediendo a la sección *Connections* y seleccionando una nueva fuente de tipo `InfluxDB` (Figura **??**).

En el formulario correspondiente se introdujeron los datos de conexión: la URL del servicio (dentro de la red Docker), el token de acceso generado previamente, el nombre de la organización y el bucket que almacena los registros meteorológicos (previamente definido en `docker-compose.yml`). Una vez completados los campos, se utilizó el botón *Save & Test* para validar la conexión. Como se observa en la Figura 11, la herramienta confirmó que el origen de datos era accesible y reconoció correctamente el bucket.

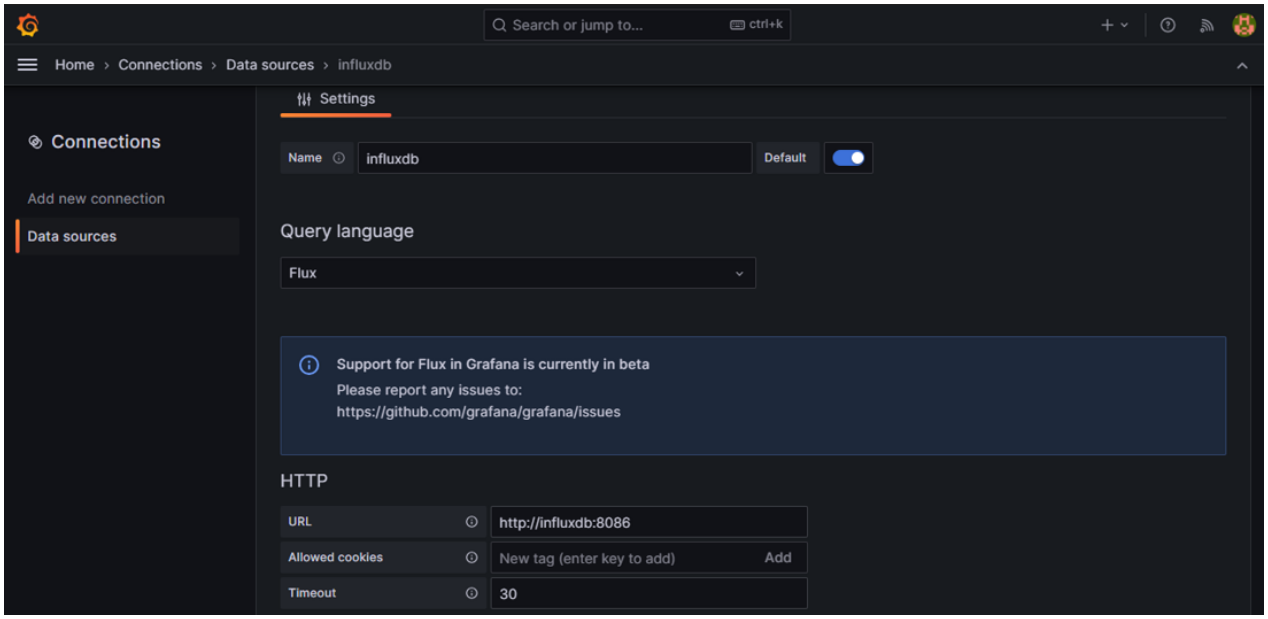

Figura 9: Configuración de InfluxDB en Grafana (parte 1: conexión HTTP y parámetros de acceso)

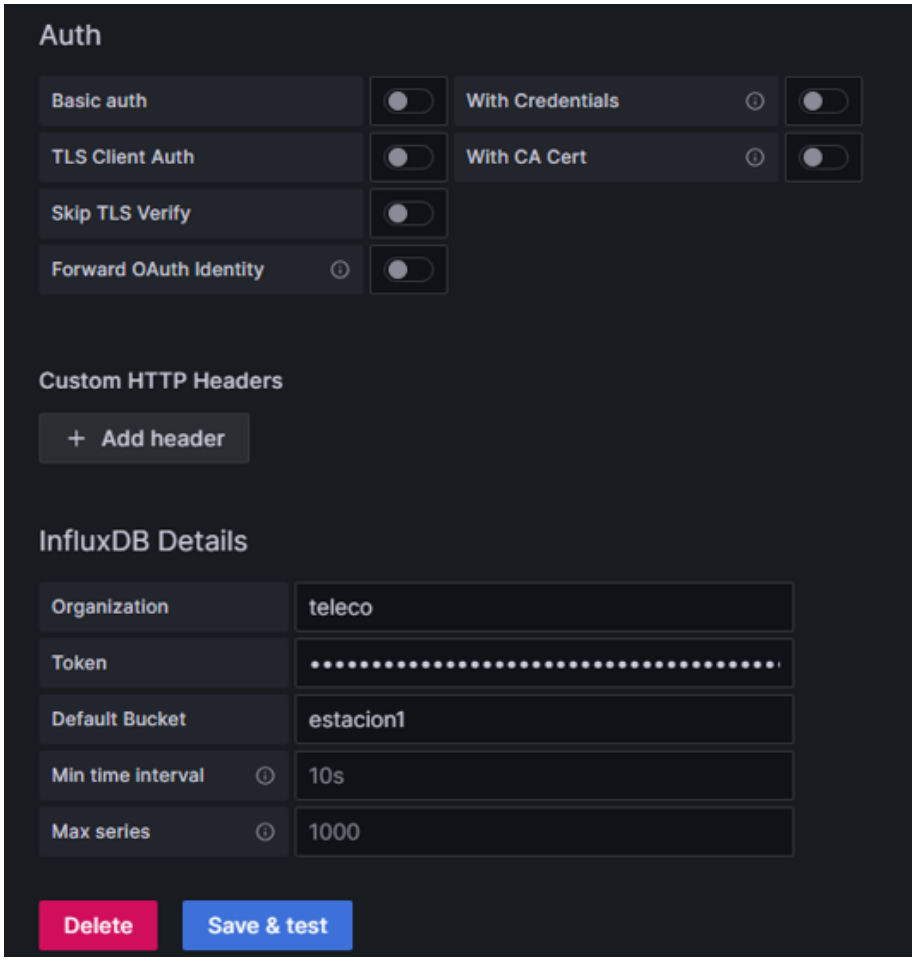

Figura 10: Configuración de InfluxDB en Grafana (parte 2: autenticación y validación de conexión)

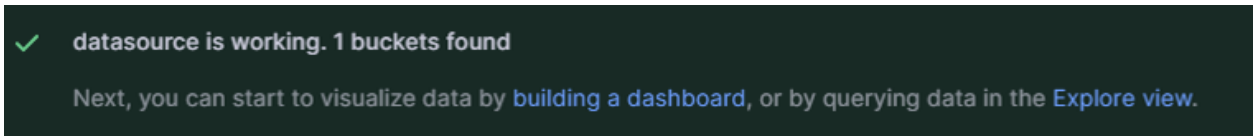

Figura 11: Conexión verificada entre Grafana e InfluxDB

4.5.2. Exportación del dashboard a JSON

El dashboard utilizado en este proyecto fue creado manualmente desde la interfaz gráfica de Grafana, partiendo de una plantilla vacía y añadiendo paneles configurados para mostrar en tiempo real las distintas variables meteorológicas almacenadas en InfluxDB.

Una vez completado su diseño, se exportó a formato JSON utilizando la funcionalidad nativa de Grafana. Esta opción permite conservar toda la estructura del panel en un archivo reutilizable, lo que facilita su restauración o despliegue en otros entornos. El archivo resultante se incluye como referencia técnica en el Anexo C de esta memoria.

4.5.3. Corrección del panel de últimos valores

En el diseño inicial del dashboard, el panel destinado a mostrar los valores actuales de los sensores estaba configurado para mostrar únicamente un sensor a la vez, obligando al usuario a seleccionar manualmente cuál visualizar. El objetivo, sin embargo, era disponer de una tabla donde se mostraran simultáneamente los valores más recientes de todas las variables meteorológicas. En su estado original, el panel no cumplía este propósito, apareciendo incluso vacío en algunos casos (Figura 12).

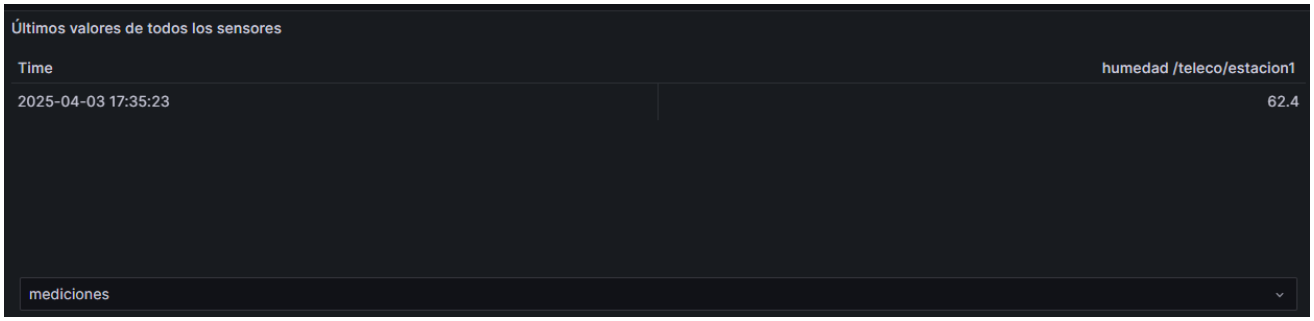

Figura 12: Visualización inicial del panel: sin datos o mostrando solo un sensor

Para solucionarlo, se reconstruyó el panel desde cero, empleando las herramientas de transformación que ofrece Grafana. En primer lugar, se modificó el tipo de visualización a *Table* (Figura 13) y se accedió a la pestaña *Transform data*, donde se seleccionó la opción *Add transformation* (Figura 14).

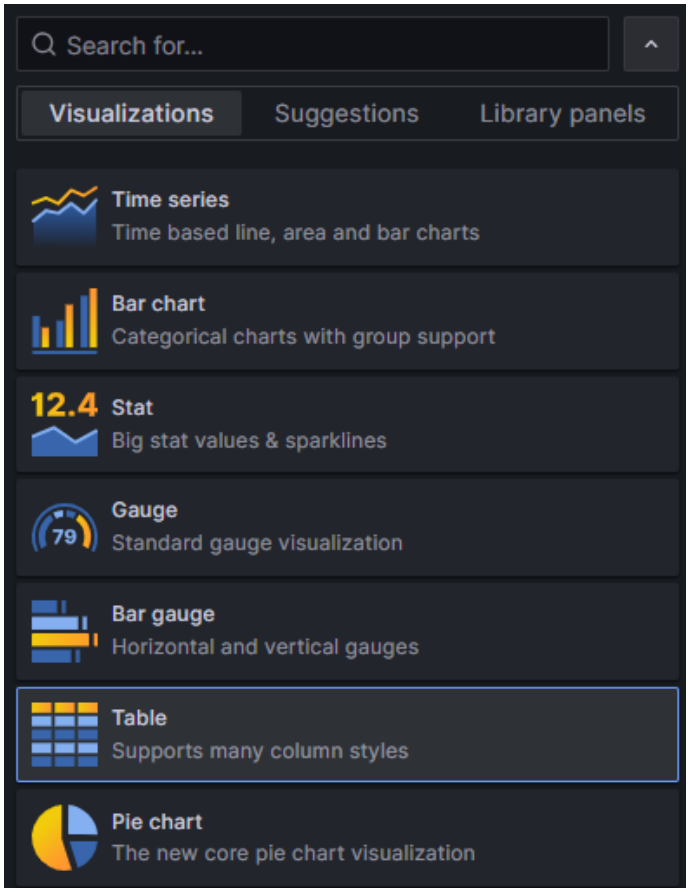

Figura 13: Cambio del tipo de panel a visualización tipo tabla

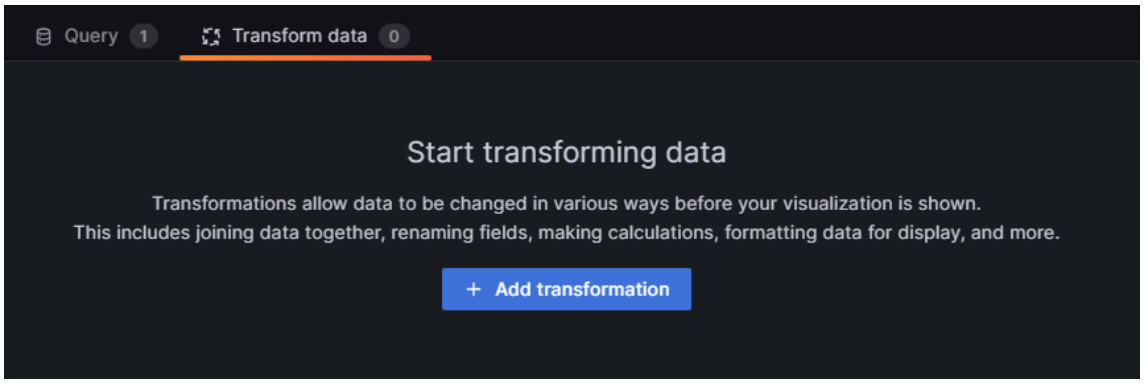

Figura 14: Acceso al menú de transformaciones en Grafana

La primera transformación aplicada fue *Join by field* (Figura 15), utilizando el campo `time` como clave de combinación. Esto permitió agrupar en una misma tabla los últimos valores de cada sensor, alineados por su marca temporal.

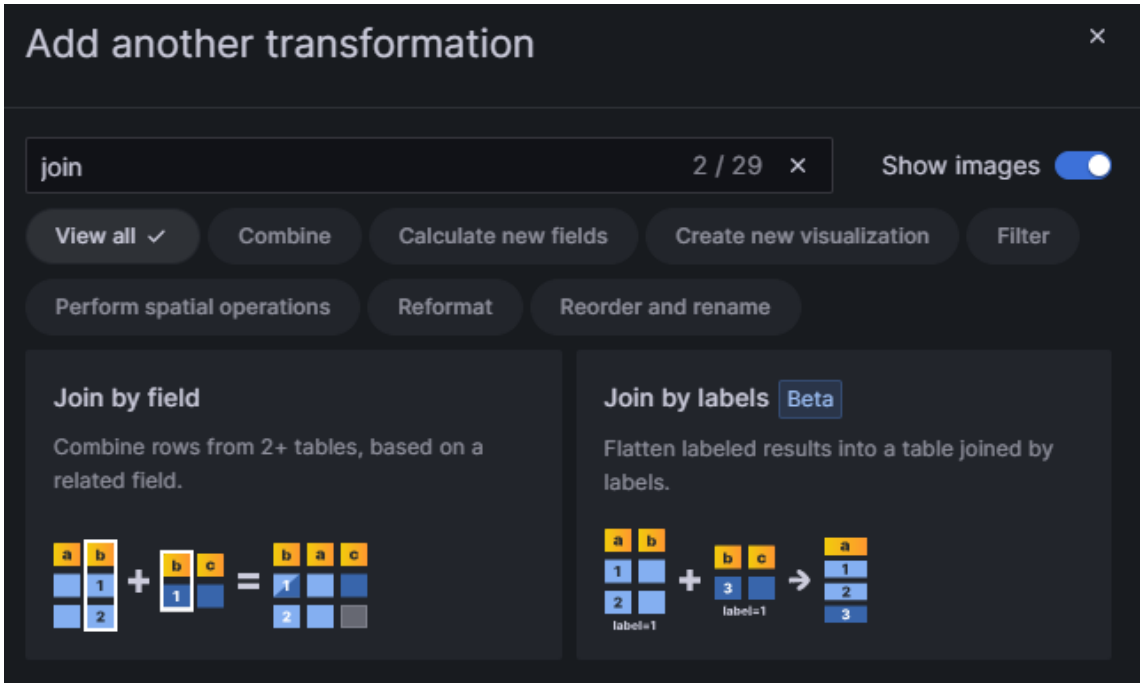

Figura 15: Selección de la transformación Join by field

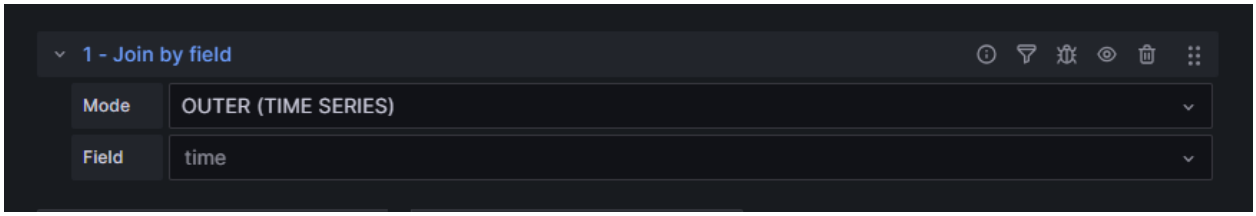

Figura 16: Configuración del Join by field utilizando el campo `time`

Una vez combinadas las métricas, se aplicó una segunda transformación: *Organize fields* (Figura 17). Esta herramienta permitió reorganizar las columnas, ocultar los campos innecesarios y asignar nombres claros a cada variable, mejorando así la legibilidad de la tabla resultante. En la Figura 18 se muestra la configuración final de esta transformación, donde se seleccionaron las columnas a mostrar y se les asignaron nombres descriptivos.

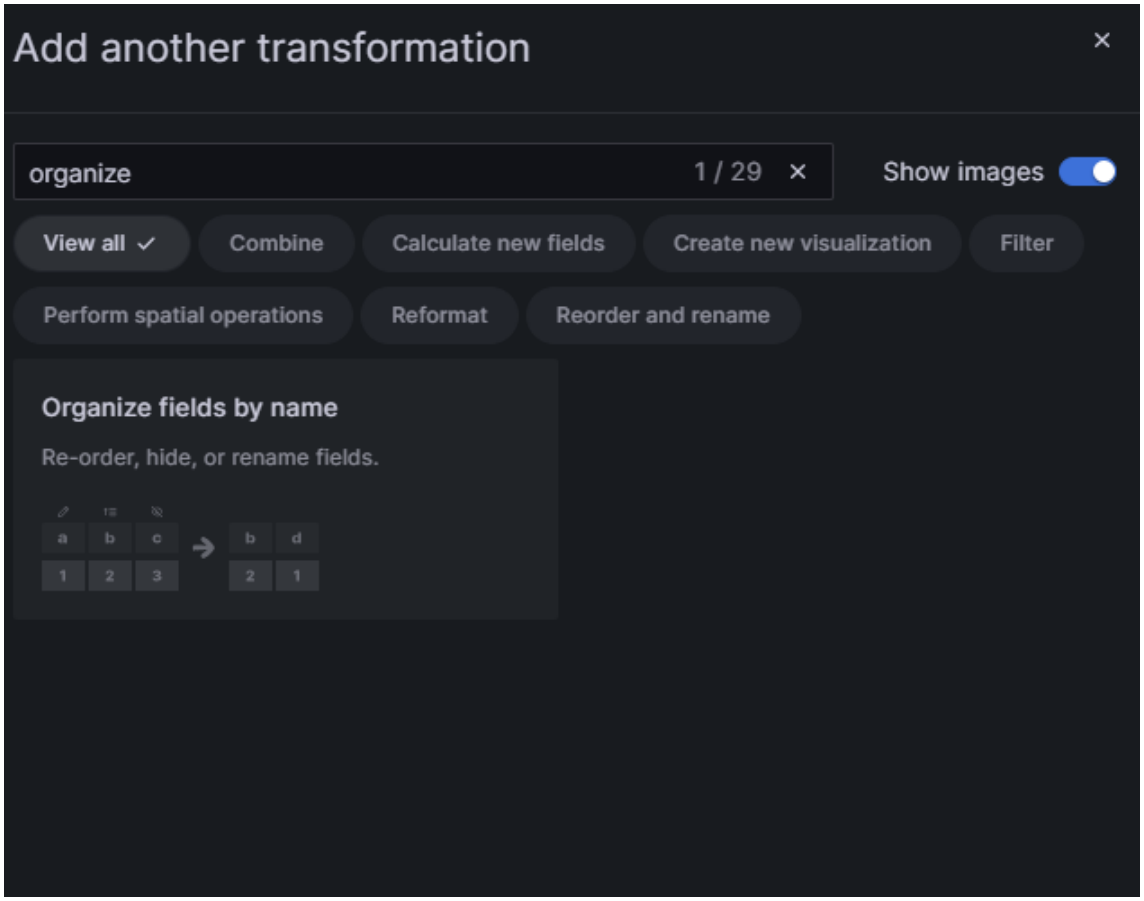

Figura 17: Selección de la transformación Organize fields

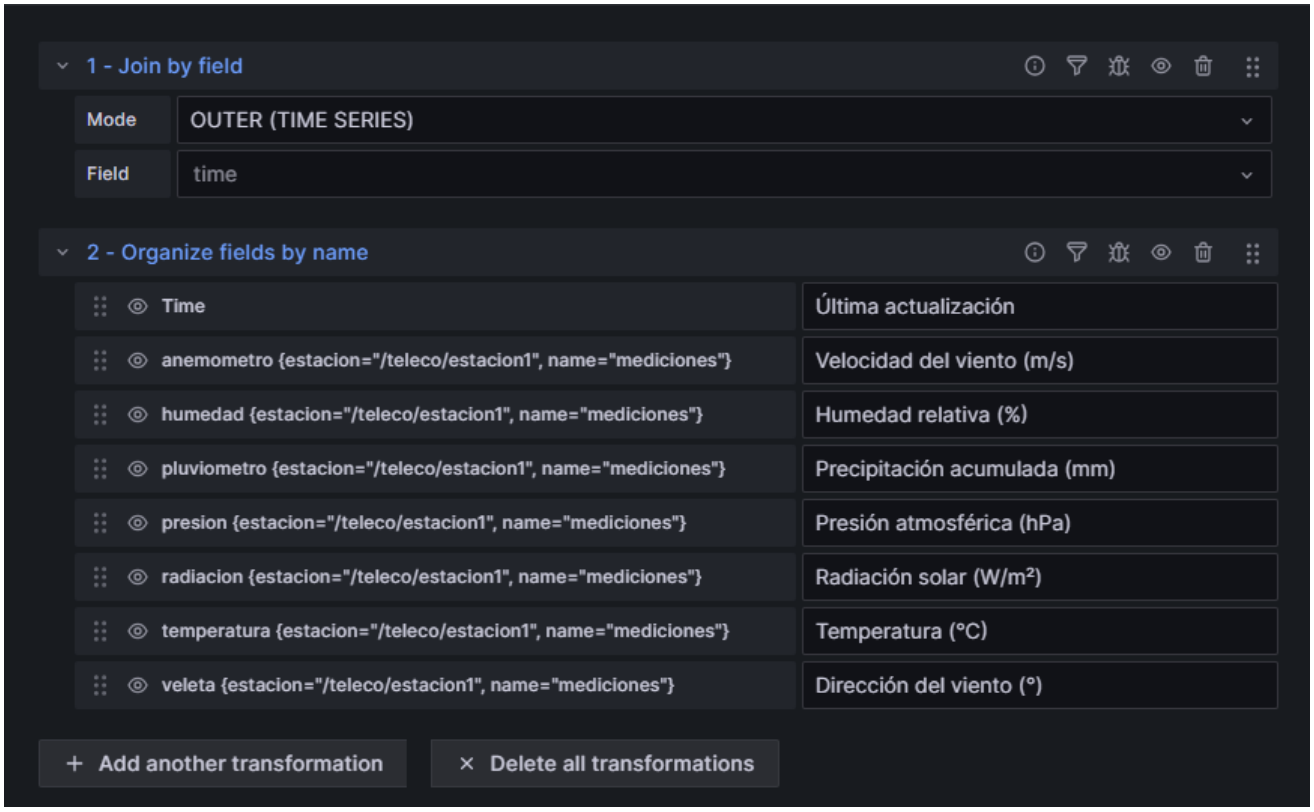

Figura 18: Organización y renombrado de columnas en el panel conjunto

Gracias a esta reconfiguración, el panel comenzó a mostrar correctamente los últimos valores registrados de todas las variables meteorológicas en una única tabla (Figura 19). Esta solución facilita una lectura rápida del estado actual de todos los sensores sin necesidad de realizar selecciones manuales.

Figura 19: Visualización final del panel conjunto con los últimos valores de cada sensor

4.5.4. Gestión de usuario de visualización

Con el objetivo de facilitar el acceso al dashboard sin comprometer las funcionalidades administrativas del sistema, se creó un usuario específico con permisos únicamente de visualización. Esta configuración se realizó desde el propio panel de administración de Grafana, utilizando la interfaz web que permite gestionar cuentas y roles de forma sencilla.

El nuevo usuario fue asignado al rol `Viewer`, lo que restringe su acceso a tareas de solo lectura sobre los dashboards definidos, sin posibilidad de modificar paneles ni alterar la configuración de las fuentes de datos. El nombre de usuario y contraseña definidos fueron escogidos de forma sencilla para facilitar el acceso durante las pruebas, aunque se recomienda modificarlos para entornos en producción.

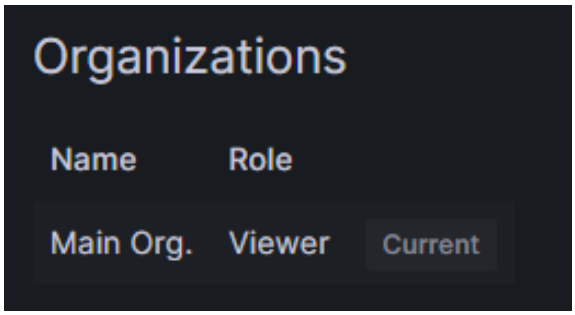

Figura 20: Rol de usuario

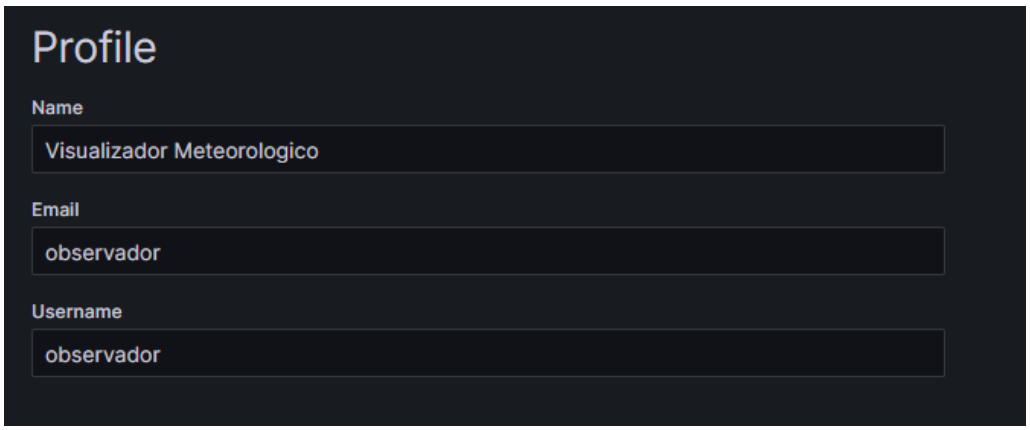

Figura 21: Perfil de usuario

Además de la reconfiguración visual del panel, se ajustó también la consulta al origen de datos para mostrar un histórico representativo del último día completo. Dado que las variables meteorológicas se registran a intervalos de cinco minutos, se optó por mostrar los últimos 288 valores, equivalentes a 24 horas de datos (12 valores por hora $\times$ 24 horas). Para ello, se sustituyó la función `last()` por una secuencia que incluye `sort()` y `limit(n:288)`, lo que permite listar los registros más recientes ordenados cronológicamente. Esta modificación asegura que el panel muestre no solo el valor actual, sino también la evolución reciente de cada variable, lo cual resulta útil para análisis comparativos y detección de patrones

diarios.

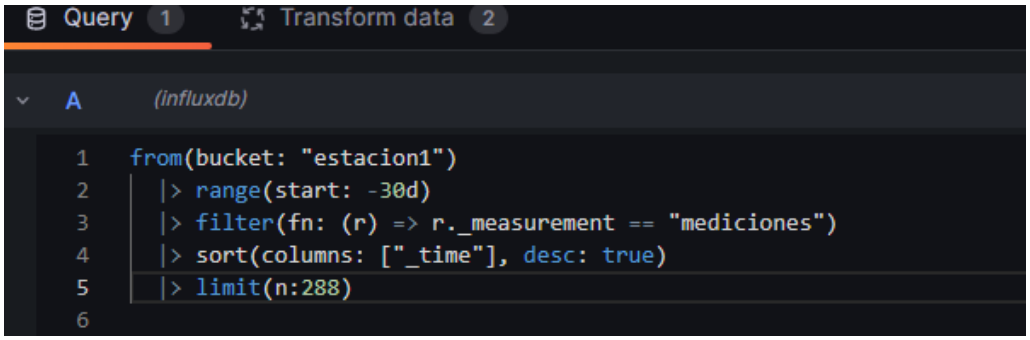

Figura 22: Últimos 288 valores (24 h) de cada variable, con lecturas cada 5 minutos.

4.6. Copias de seguridad automáticas del sistema

Dado que los datos meteorológicos almacenados en InfluxDB constituyen el núcleo del sistema, se ha considerado fundamental implementar un mecanismo de copia de seguridad automática. Esta medida tiene como objetivo proteger la información frente a posibles fallos del sistema, interrupciones de energía, errores de escritura o corrupciones en la base de datos que puedan comprometer la integridad de los datos recogidos por la estación. La estrategia adoptada consiste en generar una copia completa y periódica del bucket principal (`estacion1`) de InfluxDB y almacenarla en una carpeta local de la Raspberry Pi. Para evitar acumulación innecesaria de espacio, el sistema está configurado para mantener únicamente la copia más reciente: antes de realizar un nuevo backup, se elimina automáticamente el anterior.

El proceso se gestiona mediante un script llamado `backup_influx.sh`, ubicado en el directorio de scripts del proyecto. Su funcionamiento es simple: elimina la carpeta de respaldo anterior (si existe), crea una nueva y ejecuta el comando de copia.. La ejecución periódica del script se ha automatizado utilizando `cron`, una herramienta del sistema que permite programar tareas recurrentes. En este caso, el script se lanza automáticamente todos los días a las 2:00 de la madrugada mediante una entrada en `crontab`, asegurando que siempre exista una copia actualizada de los datos sin necesidad de intervención manual.

Listing 11: Script backup_influx.sh

```
#!/bin/bash

BACKUP_DIR="/home/teleco/backups/influxdb"

rm -rf "$BACKUP_DIR"
mkdir -p "$BACKUP_DIR"

/usr/bin/influx backup -t TU_TOKEN -b estacion1 "$BACKUP_DIR"
```

Para que la ejecución del script se repita de forma automática cada día, se ha añadido la siguiente línea al fichero `crontab` del usuario:

Listing 12: Entrada en `crontab` para el backup diario

```
0 2 * * * /home/teleco/estacion-meteorologica/scripts/backup_influx.sh
```

Gracias a esta configuración, el sistema mantiene una copia actualizada de los datos meteorológicos de forma autónoma, sin necesidad de intervención manual. Esta solución aporta una capa adicional de fiabilidad al proyecto, asegurando que la información recopilada no se pierda incluso en caso de fallo crítico.

4.7. DNS para acceso remoto

Con el objetivo de facilitar la identificación y el acceso lógico al sistema, se configuró un subdominio dinámico utilizando el servicio gratuito Duck DNS [4]. Este servicio permite asociar un nombre de dominio personalizado a la dirección IP visible desde fuera de la red donde se encuentra desplegado el sistema, actualizándola de forma automática ante cualquier cambio.

Durante la puesta en marcha, se creó una cuenta de correo específica para el proyecto y se registró el subdominio `etsit-uva-mqtt-server.duckdns.org`. Este dominio se actualiza con la dirección IP asignada por la red del laboratorio en el que se encuentra instalada la Raspberry Pi.

Para mantener el dominio actualizado, se desarrolló un script que se ejecuta automáticamente al iniciar la Raspberry Pi, mediante un servicio personalizado de systemd configurado específicamente para tal fin. Este script lanza una petición a la API de Duck DNS pasando la IP actual detectada por el dispositivo, junto con el token de autenticación generado previamente. Si la IP no ha cambiado respecto a la registrada anteriormente, el sistema de Duck DNS simplemente ignora la actualización. En caso contrario, actualiza el registro del subdominio con la nueva IP. De este modo, se garantiza que el dominio permanezca sincronizado con la dirección asignada, sin necesidad de almacenar ni comparar manualmente valores anteriores en la Raspberry.

Listing 13: Script de actualización del dominio Duck DNS

```
#!/bin/bash

curl "https://www.duckdns.org/update?domains=etsit-uva-mqtt-server&
    token=****&ip="
```

El comportamiento general del sistema es el siguiente:

- Al arrancar el sistema, la Raspberry Pi ejecuta un script automático.
- Este script lanza una petición HTTP a la API de Duck DNS, incluyendo el nombre del subdominio y el token de autenticación.
- Duck DNS recibe la IP desde la que se realiza la petición y decide si es necesario actualizar el registro.
- Si la IP ha cambiado, el servicio actualiza el registro del dominio; si no, lo deja sin modificaciones.

Actualmente, debido a las posibles políticas de red y seguridad del entorno universitario —como el uso de NAT, cortafuegos o restricciones de acceso desde el exterior— no es posible acceder directamente al dispositivo desde redes externas mediante dicho dominio (por ejemplo, vía SSH o accediendo al panel de Grafana por el puerto 3000). Sin embargo, el dominio funciona como punto de acceso lógico dentro de la red del sistema.

Además, el uso de un nombre de dominio ofrece una ventaja importante de cara a la gestión del despliegue: si en el futuro la Raspberry Pi se traslada a otro laboratorio, o si cambia su dirección IP dentro de la red, las estaciones meteorológicas u otros servicios conectados podrán seguir utilizando el dominio sin necesidad de reconfiguraciones manuales. Esto facilita la escalabilidad y portabilidad del sistema.

5. Pruebas y validación

Con el objetivo de comprobar el correcto funcionamiento del sistema propuesto, se han diseñado y ejecutado diferentes pruebas de validación. Estas pruebas permiten verificar tanto la robustez del sistema ante escenarios de alta carga como la correcta visualización de datos en el panel de Grafana. A través de simulaciones controladas, se evalúa la respuesta del sistema en condiciones similares a su uso real, incluyendo tanto el procesamiento de múltiples mensajes como la representación gráfica de las variables meteorológicas. A continuación, se describen en detalle las pruebas realizadas.

5.1. Prueba 1: Comunicación básica mediante MQTT

Esta prueba se realizó directamente desde la Raspberry Pi. Para facilitar el trabajo en múltiples terminales, se empleó la herramienta `tmux`, que permitió ejecutar de forma simultánea el script receptor y el comando de publicación.

En este punto del proyecto, el script `mqtt_listener.py` aún no estaba configurado para ejecutarse automáticamente al inicio del sistema. Por tanto, fue lanzado manualmente desde una de las sesiones de terminal para que quedara a la escucha del broker MQTT.

Desde otra sesión dentro de la misma Raspberry Pi, se envió un mensaje de prueba utilizando el comando:

Listing 14: Envío de mensaje MQTT desde terminal

```
mosquitto_pub -h localhost -p 1883 -u admin -P admin -t "/teleco/
   estacion1" -m '{"t":22.5,"h":56.7,"pn":1012,"v":90,"a":4.1,"pl
   ":0.0,"r":510}'
```

A continuación se describen las variables incluidas en el mensaje JSON mostrado, junto con sus unidades:

- `t`: temperatura, en grados Celsius (°C)
- `h`: humedad relativa, en porcentaje (%)
- `pn`: presión atmosférica, en hectopascales (hPa)
- `v`: dirección del viento (veleta), en grados (°)
- `a`: velocidad del viento (anemómetro), en kilómetros por hora (km/h)
- `pl`: pluviometría acumulada, en litros por metro cuadrado (L/m^2)
- `r`: índice UV, en vatios por metro cuadrado (W/m^2); también podría interpretarse como índice UV en ciertos contextos

La recepción del mensaje fue inmediata y se reflejó en la consola donde estaba corriendo el script, lo que confirmó que la comunicación MQTT era funcional. La Figura 23 muestra la salida por terminal correspondiente.

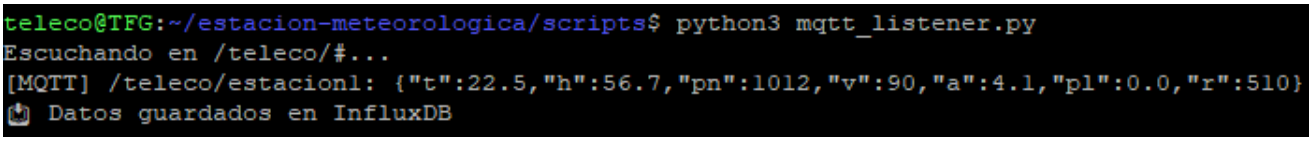

Figura 23: Captura de consola mostrando la recepción del mensaje MQTT en el script `mqtt_listener.py`

Esta prueba sirvió como paso previo fundamental para validar la conectividad entre componentes antes de avanzar con simulaciones más completas.

5.2. Prueba 2: Visualización de datos simulados en el panel de Grafana

Con el objetivo de validar la correcta representación gráfica de los datos meteorológicos en tiempo real, se desarrolló una prueba centrada exclusivamente en el comportamiento visual del sistema. Esta prueba busca comprobar que el panel de Grafana es capaz de mostrar variaciones coherentes y diferenciadas para cada parámetro sensorial.

Para ello, se diseñó un script de simulación que emite valores sintéticos generados mediante funciones matemáticas conocidas (como senos y cosenos), lo que permite verificar fácilmente que las curvas presentadas en el dashboard corresponden a los patrones esperados.

5.2.1. Simulación de valores dinámicos

El script fue programado para publicar un mensaje MQTT cada segundo durante un total de cinco minutos. Cada mensaje contiene las variables meteorológicas estándar, generadas a partir de funciones matemáticas que producen formas características en las gráficas. Este enfoque facilita la validación visual del sistema, permitiendo verificar que los datos enviados son correctamente recibidos, almacenados y representados. El código fuente completo puede consultarse en el Anexo D.

- **Temperatura**: onda senoidal con dos ciclos completos en cinco minutos, representando una oscilación suave en grados Celsius (°C).
- **Humedad relativa**: función coseno con la misma frecuencia, generando un desfase respecto a la temperatura. Se expresa en porcentaje (%).
- **Presión atmosférica**: variación más lenta y menos pronunciada, en torno a los 1010 hPa.
- **Velocidad del viento (anemómetro)**: seno de frecuencia más alta, simulando variaciones rápidas. Expresado en m/s.
- **Precipitación**: valores aleatorios pequeños (0–0.4 mm) intercalados con ceros, simulando lluvias breves. Equivale a L/m^2.
- **Índice UV**: curva basada en $\sin^2$, simulando la variación de la radiación solar a lo largo del día. En W/m^2.
- **Dirección del viento (veleta)**: valores aleatorios en grados (0–360), reflejando un comportamiento caótico natural.

Aunque no se pretende replicar condiciones meteorológicas reales, el uso de estas funciones permite evaluar con claridad el comportamiento del sistema ante distintos tipos de señal.

5.2.2. Representación en el panel de Grafana

A continuación se muestran las gráficas individuales generadas en el panel de Grafana, correspondientes a cada una de las variables simuladas. Se observan oscilaciones suaves en las variables periódicas (temperatura, humedad, presión y radiación), variaciones rápidas en el viento, y patrones de ruido controlado en precipitación y dirección. Estas visualizaciones confirman que el sistema responde adecuadamente a diferentes patrones de entrada, validando su funcionamiento global.

Figura 24: Evolución de la temperatura simulada (°C) con patrón senoidal.

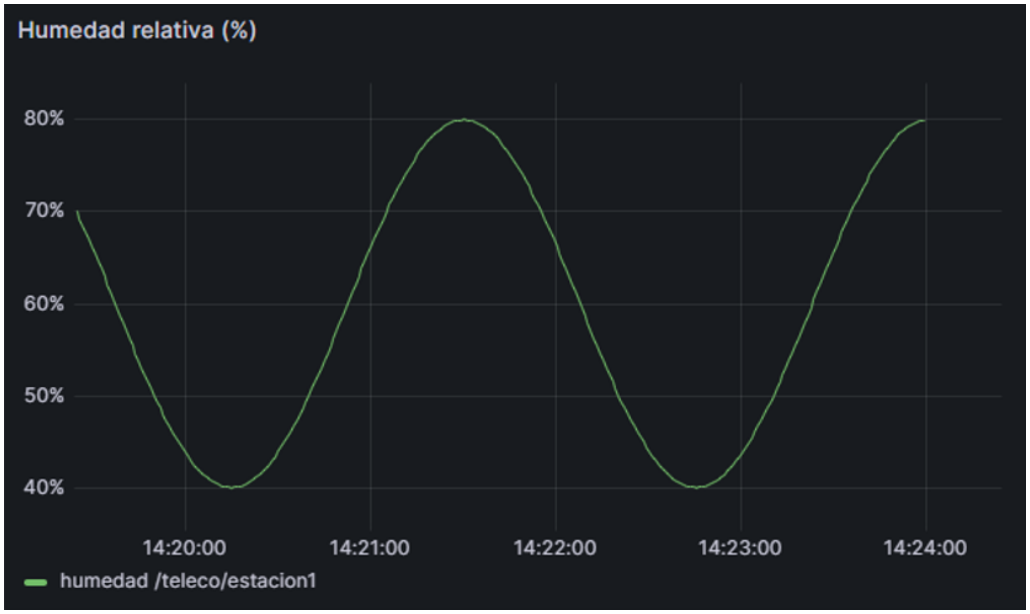

Figura 25: Humedad relativa (%): patrón cosenoidal con desfase respecto a la temperatura.

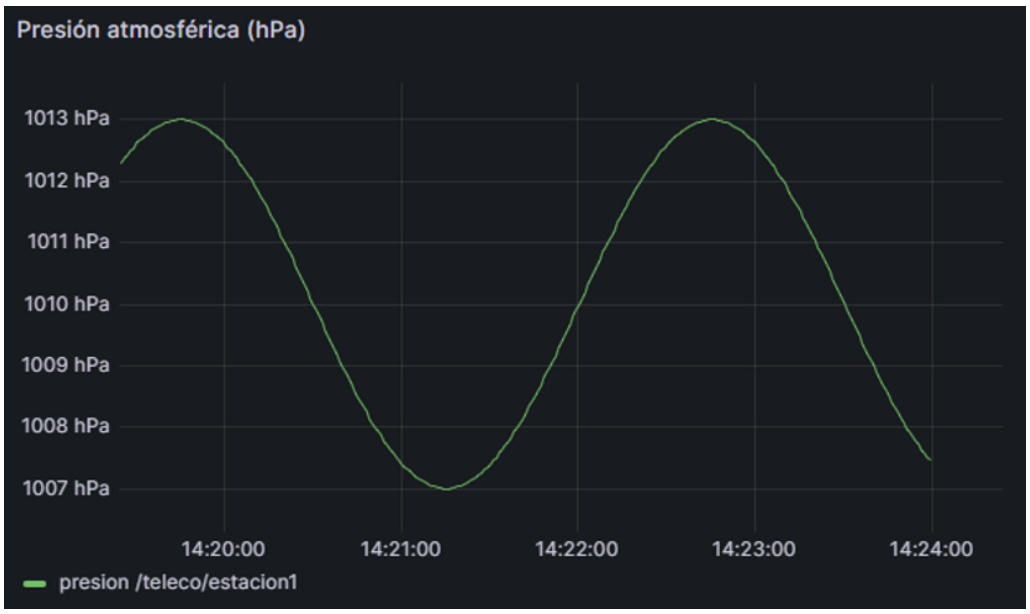

Figura 26: Presión atmosférica (hPa): variación suave y lenta dentro de un rango realista.

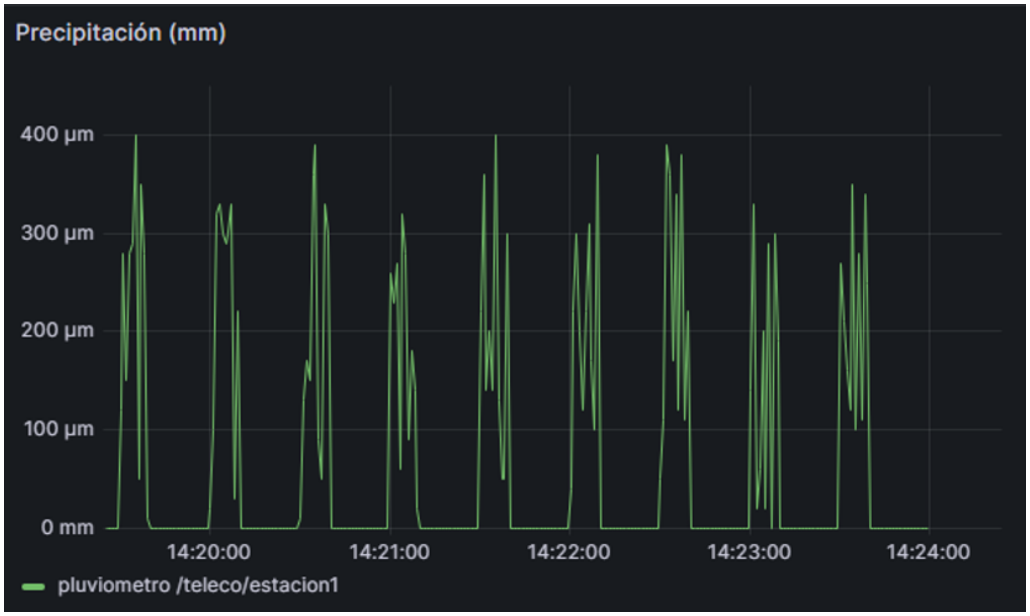

Figura 27: Precipitación (mm): ráfagas de valores aleatorios breves que simulan lluvia ligera.

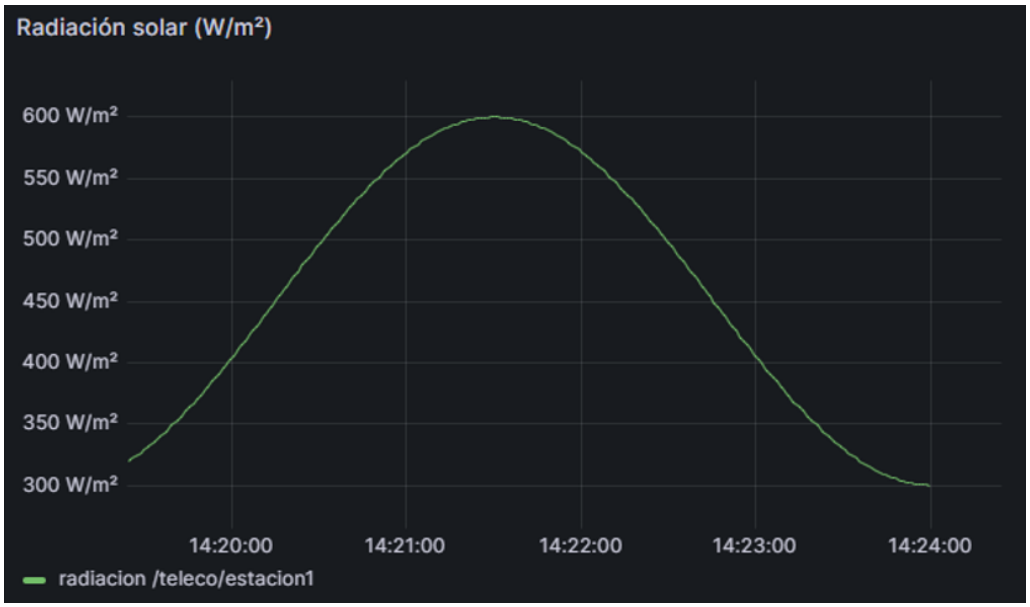

Figura 28: Radiación solar (W/m^2): curva $\sin^2$ invertida simulando el perfil solar diario.

Figura 29: Velocidad del viento (m/s): señal periódica rápida que representa rachas de viento.

Figura 30: Dirección del viento (°): valores aleatorios reflejando comportamiento irregular.

Conclusiones de la prueba:

- El sistema es capaz de recibir datos simulados en tiempo real, almacenarlos correctamente en InfluxDB y representarlos en Grafana sin errores ni retrasos perceptibles.
- El flujo de publicación MQTT, almacenamiento y visualización opera de forma fluida y coherente, lo que valida que la arquitectura es funcional y estable incluso con una tasa de envío de un mensaje por segundo.
- Las gráficas generadas en el panel de Grafana muestran una evolución continua y clara de los datos, lo que facilita su lectura e interpretación durante el proceso de validación.
- El uso de funciones matemáticas conocidas (como senos y cosenos) permitió verificar visualmente que las curvas no presentan distorsión ni errores durante su representación, aunque su objetivo fue únicamente ilustrativo.

5.3. Prueba 3: Monitorización del sistema bajo carga

Con el objetivo de evaluar la capacidad de la Raspberry Pi para procesar múltiples peticiones simultáneas sin degradación significativa, se diseñó una prueba de carga controlada. Para ello, se utilizó un PC externo encargado de generar los mensajes MQTT, evitando así que la propia Raspberry Pi asumiera tareas adicionales que pudieran distorsionar los resultados. El sistema se limitó a recibir, procesar e insertar los datos en la base de datos como ocurriría en condiciones reales.

Previamente, se instrumentó la Raspberry Pi con **Telegraf** para recolectar métricas del sistema operativo, y se configuró un dashboard personalizado en **Grafana** para visualizar los diferentes parámetros. A continuación, se ejecutó una simulación que imitaba el comportamiento de cientos de estaciones meteorológicas enviando datos, con el fin de comprobar la robustez del sistema ante una carga elevada.

5.3.1. Instrumentación con Telegraf

Para poder interpretar correctamente el comportamiento del sistema bajo estrés, fue necesario instrumentarlo previamente mediante un sistema de monitorización.

Para ello, se instaló **Telegraf**, un agente ligero desarrollado por InfluxData, que permite recolectar métricas del sistema operativo como uso de CPU, memoria, actividad de disco y red [16]. Telegraf se ejecuta como servicio en la Raspberry Pi y envía sus datos a InfluxDB en un *bucket* separado.

Con el fin de mejorar la resolución temporal de las métricas durante las pruebas de carga, se modificó el intervalo de muestreo predeterminado de Telegraf, reduciéndolo de 10 segundos a 1 segundo. Esta configuración permite capturar con mayor precisión los picos momentáneos de carga generados por el envío masivo de mensajes. Si bien sería posible reducir aún más el intervalo (por ejemplo, a 500 ms), este tipo de configuraciones puede provocar una sobrecarga adicional del sistema debido al aumento en la frecuencia de recopilación y escritura de datos. Por tanto, se consideró que un segundo representaba un equilibrio adecuado entre precisión y eficiencia.

Figura 31: Creación del bucket `Prueba` en la interfaz web de InfluxDB

Listing 15: Sección de configuración del archivo `telegraf.conf`

```
interval = "1s"
[[outputs.influxdb_v2]]
  urls = ["http://localhost:8086"]
  token = "**************"
  organization = "teleco"
  bucket = "telegraf"
```

El servicio fue habilitado y puesto en marcha mediante `systemd`:

Listing 16: Activación del servicio Telegraf

```
sudo systemctl enable telegraf
sudo systemctl start telegraf
```

A partir de este momento, comenzaron a recibirse métricas del sistema operativo cada 10 segundos, las cuales fueron confirmadas en la interfaz de InfluxDB.

5.3.2. Visualización mediante Grafana

Para representar gráficamente el rendimiento del sistema durante las pruebas de carga, se configuró un dashboard personalizado en Grafana vinculado al bucket de métricas generado por Telegraf.

Este panel fue diseñado específicamente para la prueba, incorporando visualizaciones que permiten analizar en tiempo real los principales parámetros del sistema operativo:

- **Uso de CPU (%)**: muestra el porcentaje real de utilización del procesador. Permite visualizar con precisión el impacto del envío de mensajes sobre los recursos de la Raspberry Pi, con picos claramente marcados en los momentos de carga.
- **RAM usada (%)**: refleja el porcentaje de memoria RAM utilizada en tiempo real. Aunque puede no presentar variaciones drásticas en pruebas breves, permite identificar tendencias de consumo sostenido o acumulación de buffers por parte del sistema operativo.
- **Tráfico de red (wlan0 - RX)**: muestra la velocidad de recepción de datos en la interfaz Wi-Fi. Este panel ofrece una visualización clara del ritmo de entrada de mensajes.
- **Escritura en disco**: representa la velocidad de escritura en la tarjeta SD, en bytes por segundo. El panel permite observar cómo se incrementa la actividad del sistema de almacenamiento durante la inserción de datos en InfluxDB, especialmente en las fases de envío continuo.
- **Lecturas de disco (read_bytes)**: muestra la velocidad de lectura en disco, aunque en este tipo de pruebas tiende a mantenerse en valores bajos o nulos, ya que el sistema se limita principalmente a operaciones de escritura. Su inclusión permite verificar la ausencia de procesos de lectura innecesarios durante la simulación.

- **Carga del sistema**: refleja el número promedio de procesos en ejecución o esperando CPU durante el último minuto. Al tratarse de una métrica de tipo promedio móvil, su respuesta es más gradual que la del uso de CPU, pero sigue siendo útil para identificar acumulaciones de carga sostenida o picos prolongados.

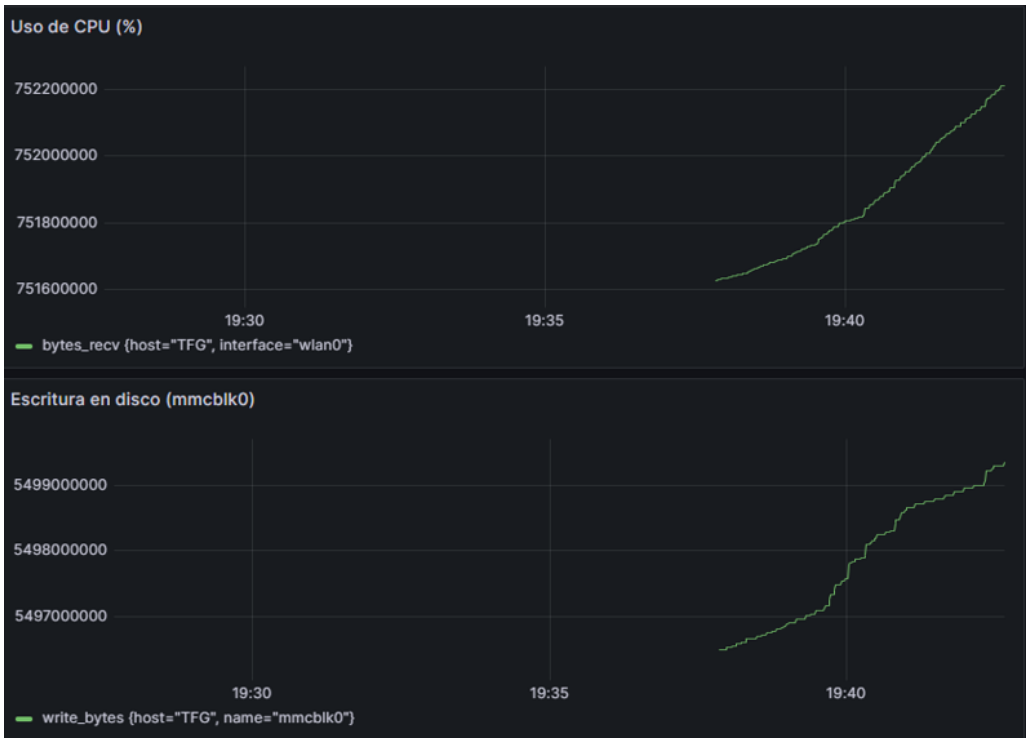

Figura 32: Panel de Grafana (parte 1)

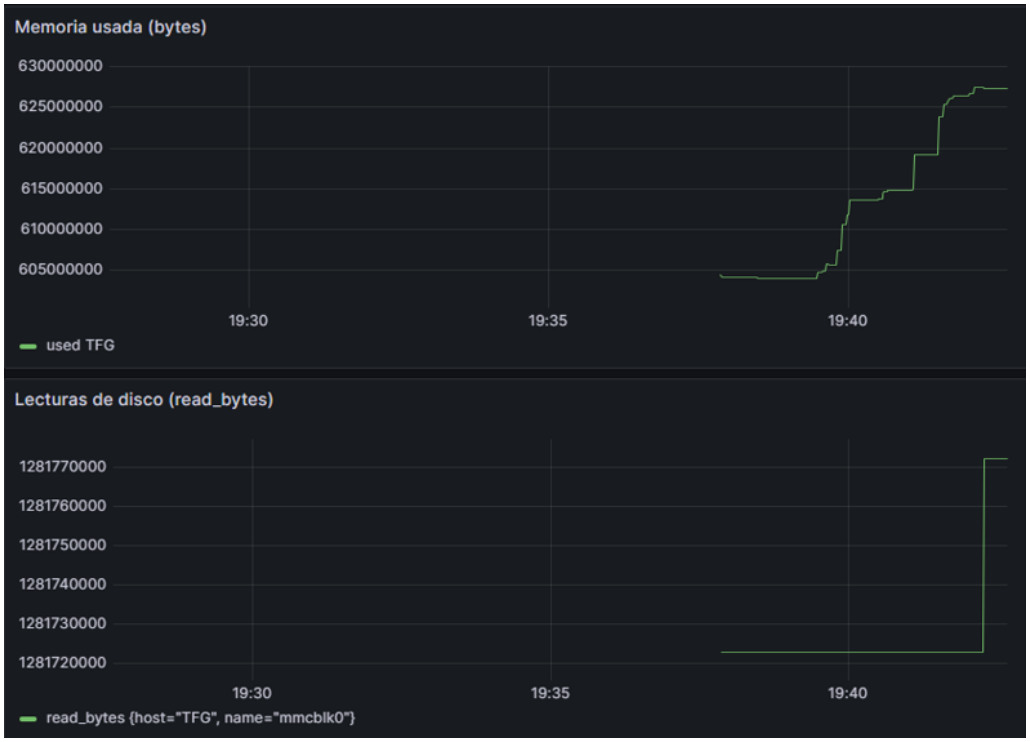

Figura 33: Panel de Grafana (parte 2)

Figura 34: Panel de Grafana (parte 3)

5.3.3. Prueba de carga con simulación de estaciones

Para evaluar la capacidad del sistema en escenarios de alta concurrencia, se diseñó una simulación en la que se enviaron mensajes MQTT a razón de **20 por segundo**, desde un equipo externo. Esto permitió generar una carga elevada sin afectar los recursos de la Raspberry Pi, que se centró únicamente en recibir, procesar e insertar los datos en InfluxDB.

En condiciones normales, una estación meteorológica envía un mensaje cada 5 minutos. Por tanto, se puede estimar el número de estaciones que podrían ser gestionadas simultáneamente mediante una simple regla de tres:

$$1 \text{ estación} \rightarrow 1 \text{ mensaje cada } 300 \text{ s} \Rightarrow 0.0033 \text{ msg/s}$$

$$20 \text{ msg/s} \div 0{,}0033 \approx \mathbf{6060 \text{ estaciones}}$$

Este ritmo de envío equivale a simular el comportamiento de **6060 estaciones meteorológicas** enviando datos cada cinco minutos, lo que permite evaluar el sistema bajo una carga altamente representativa de un escenario real a gran escala.

Para realizar la prueba, se utilizó un script Python que publicaba mensajes en el broker MQTT con una frecuencia controlada, puede consultarse en el Anexo E de esta memoria.

La simulación, con una duración total de 5 minutos, se estructuró en varios bloques temporales:

- 30 segundos iniciales sin envío.
- 90 segundos de envío continuo.
- 60 segundos de pausa.
- 90 segundos adicionales de envío.
- 30 segundos finales sin tráfico.

Durante la prueba, se configuró el panel de Grafana para mostrar exactamente los **últimos cinco minutos de actividad**, con una tasa de refresco de **5 segundos**. Esto provoca que las métricas se visualicen en forma de picos regulares, ya que los valores cambian coincidiendo con cada ciclo de publicación definido en el script. En realidad, los datos se reciben y procesan de forma continua, pero la naturaleza discreta del refresco visual puede dar la falsa impresión de una carga pulsada. Aun así, esto no representa un error, sino una forma muy útil de comprobar la correspondencia exacta entre el comportamiento del simulador y la respuesta del sistema.

A continuación, se presentan las gráficas capturadas, organizadas por parejas para facilitar su análisis comparativo:

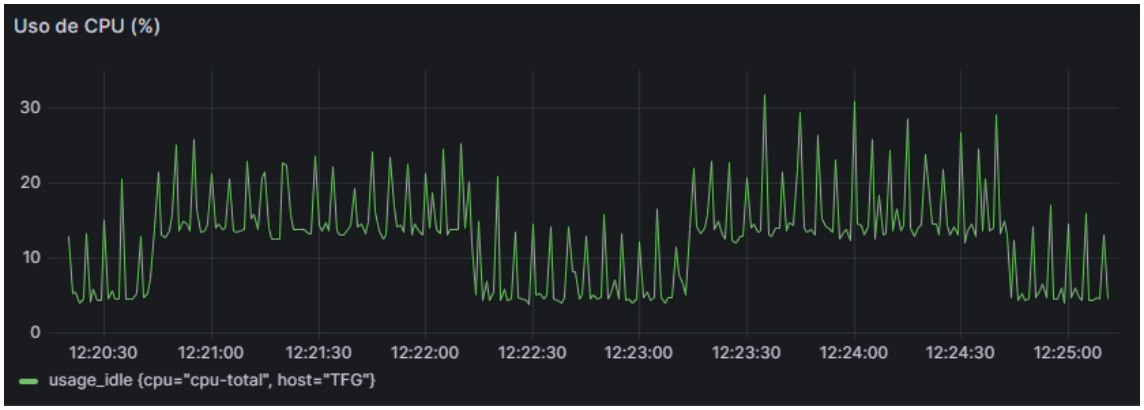

Figura 35: Porcentaje de uso de CPU durante la prueba de carga

Figura 36: Porcentaje de uso de RAM en tiempo real

Figura 37: Tráfico de red en la interfaz Wi-Fi (wlan0)

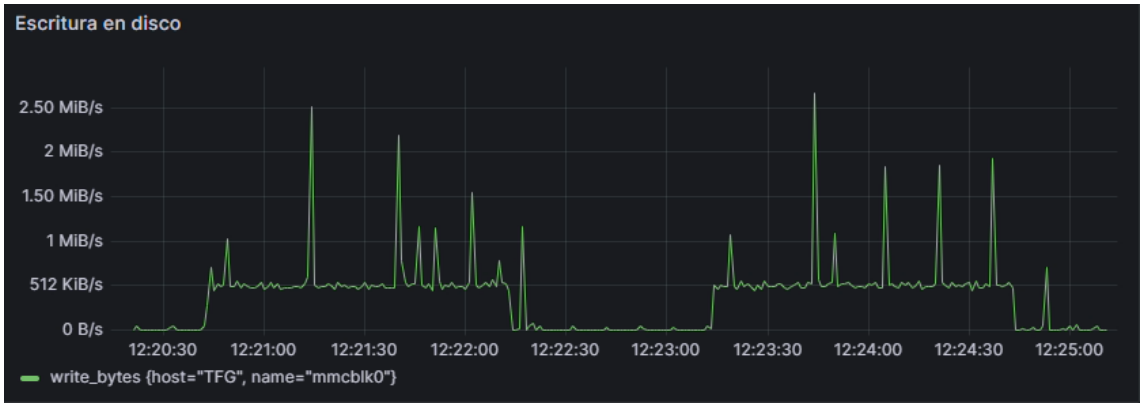

Figura 38: Velocidad de escritura en disco

Figura 39: Velocidad de lectura en disco

Figura 40: Carga media del sistema (load average)

Análisis de las métricas obtenidas:

- **Uso de CPU:** se aprecian claramente los intervalos de envío y pausa. El uso de CPU aumenta de forma notable durante el tráfico activo, pasando de valores base del 10 %

hasta picos del 25-30 %. Este comportamiento es coherente con el procesamiento de mensajes y la escritura a base de datos. La forma de picos regulares se debe al refresco de Grafana cada 5 segundos.

- **Uso de RAM:** el uso de memoria se mantiene estable durante la mayor parte del experimento, aunque muestra un incremento repentino en el segundo bloque de envío. Este salto puede deberse a buffers del sistema o al comportamiento interno de InfluxDB o Mosquitto. Aun así, el porcentaje total nunca supera el 16.5 %, por lo que no supone un riesgo de saturación.
- **Tráfico de red (RX - wlan0):** refleja de forma muy clara el patrón de la simulación. Se observa tráfico intenso durante los bloques de envío y silencio durante las pausas, confirmando la actividad del publicador MQTT y la recepción correspondiente por parte del sistema.
- **Escritura en disco:** presenta el patrón más claro de todos. Los picos de escritura corresponden directamente a los intervalos en los que se reciben datos y se almacenan en InfluxDB. Es una de las gráficas más útiles para validar que se están insertando los datos correctamente.
- **Lectura en disco:** aunque Grafana realiza consultas continuamente para actualizar los paneles, InfluxDB es capaz de responder a la mayoría de estas peticiones directamente desde memoria, sin necesidad de acceder al disco físico. Esto se debe a que los datos más recientes suelen estar en caché o en buffers de escritura. Por tanto, la baja actividad de lectura que muestra la gráfica no indica ausencia de consultas, sino que estas son resueltas de forma eficiente sin requerir operaciones de disco.
- **Carga del sistema:** también refleja el comportamiento esperado, aunque de forma algo más atenuada. La carga aumenta durante los envíos y desciende en las pausas. Al tratarse de una media móvil, el valor es más amortiguado que en la CPU.

Consideraciones sobre la sincronización de estaciones: Cabe señalar que, en la simulación realizada, los mensajes se enviaron de forma continua y controlada, manteniendo un ritmo constante de 20 publicaciones por segundo. Sin embargo, en un escenario real con miles de estaciones, no sería adecuado que todas estuvieran estrictamente sincronizadas y enviaran sus datos exactamente al mismo tiempo. Provocaría picos de carga muy elevados y concentrados en momentos concretos, afectando negativamente al rendimiento del sistema, especialmente en la red, el uso de CPU y la escritura en disco. Esto podría generar cuellos de botella y aumentar el riesgo de pérdida de datos si los servicios no son capaces de absorber ese tráfico puntual.

Por ello, resulta preferible que las estaciones estén ligeramente desincronizadas entre sí y envíen sus datos de manera distribuida a lo largo del tiempo. Esta estrategia no solo reduce la carga puntual, sino que mejora la estabilidad global del sistema. En términos de diseño de arquitectura IoT, fomentar este tipo de asincronía natural entre dispositivos es una buena práctica ampliamente recomendada.

Conclusiones preliminares de la prueba:

- El sistema se comportó de forma totalmente estable, sin pérdidas de mensajes ni interrupciones visibles.
- La CPU respondió con subidas controladas, y la RAM se mantuvo siempre dentro de márgenes seguros.
- Las gráficas confirman que la arquitectura soporta correctamente la carga simulada equivalente a más de 6000 estaciones activas.
- No se detectaron cuellos de botella en red ni en disco; las tasas observadas son perfectamente asumibles por el entorno hardware.
- El experimento demuestra que el sistema es escalable y viable para su uso en escenarios reales de monitorización meteorológica.

6. Líneas futuras

Este proyecto ha sentado las bases para un sistema robusto de monitorización meteorológica en tiempo real utilizando una Raspberry Pi y Grafana. Sin embargo, el potencial de crecimiento y mejora es considerable. Las siguientes líneas futuras proponen una expansión ambiciosa de las capacidades del sistema, abarcando desde la recolección de datos hasta la interacción con el usuario y la seguridad.
La evolución natural de este proyecto hacia una solución más robusta y escalable implica la migración de la infraestructura principal de la Raspberry Pi a un entorno de nube. Este paso estratégico permitiría superar las limitaciones de hardware y conectividad de un dispositivo local, garantizando una mayor disponibilidad del servicio, una escalabilidad elástica para manejar volúmenes crecientes de datos y un número elevado de usuarios concurrentes, y una redundancia inherente que minimiza los puntos únicos de fallo

I. Mejora y Expansión de Datos

- **Gestión de Cargas Simultáneas:** En escenarios en los que múltiples estaciones envían datos de forma sincronizada o casi simultánea, podrían producirse picos de tráfico que afecten al rendimiento del sistema o a la consistencia de las inserciones en la base de datos. Como línea futura, se propone la incorporación de un sistema de *buffering* o cola de procesamiento que permita desacoplar la llegada de los datos de su almacenamiento. Esto garantizaría una mayor tolerancia a picos de carga y una distribución más uniforme del trabajo, evitando cuellos de botella y pérdidas de mensajes en situaciones de alta concurrencia.
- **Integración con Fuentes de Datos Externas:** Para enriquecer el contexto y la validez de los datos locales, se contempla la integración con APIs de servicios meteorológicos externos. Esto permitiría comparar los datos obtenidos por la estación con pronósticos y mediciones de referencia, facilitando la detección de desviaciones y la validación cruzada. Adicionalmente, la incorporación de datos históricos y climatológicos de estaciones cercanas podría ofrecer una perspectiva a largo plazo, habilitando análisis de tendencias y patrones climáticos más allá de los datos recolectados *in situ*.

II. Interacción y Usabilidad

- **API Pública:** Una evolución significativa sería la definición de una API pública (por ejemplo, basada en REST) que permita a usuarios autenticados contribuir con datos de estaciones meteorológicas adicionales desde diferentes localidades. Esta funcionalidad no solo multiplicaría la cobertura geográfica de la red de monitorización, sino que también fomentaría una comunidad de colaboradores. Para garantizar la calidad y fiabilidad de los datos, se explorarían modelos de contribución con roles diferenciados (ej., observadores, validadores) y mecanismos robustos de validación previa antes de la inserción en la base de datos.
- **Visualización Contextual:** La interfaz de usuario de Grafana, aunque potente para la visualización de datos, podría complementarse con funcionalidades que mejoren

la experiencia del usuario y amplíen las capacidades de monitorización. Se propone la integración de una cámara IP que visualice la estación o su entorno (por ejemplo, el cielo), proporcionando un contexto visual en tiempo real de las condiciones meteorológicas. Más allá de la simple transmisión en vivo, se podría implementar la captura periódica de imágenes para la generación de *timelapses*, facilitando la observación de fenómenos a lo largo del tiempo.

- **Accesibilidad Móvil:** Asimismo, se podría considerar el desarrollo de una aplicación móvil dedicada o una *Progressive Web App* (PWA) que permita el acceso a la información meteorológica desde cualquier dispositivo, incorporando funcionalidades específicas como notificaciones *push* personalizadas, gestión de estaciones favoritas o ajustes visuales individuales. Esto facilitaría la interacción desde entornos móviles y mejoraría la accesibilidad del sistema en tiempo real.

III. Análisis y Procesamiento de Datos

- **Análisis Predictivo (Machine Learning):** Una de las líneas futuras más prometedoras es la aplicación de técnicas de *Machine Learning* para el análisis y la predicción de variables meteorológicas. Utilizando los datos históricos recopilados por la estación, se podrían implementar modelos de series temporales (por ejemplo, ARIMA o LSTM) capaces de anticipar comportamientos de temperatura, humedad o presión con unas horas de antelación. Para ello, los datos almacenados en InfluxDB podrían exportarse periódicamente en formato CSV y procesarse mediante herramientas como `pandas`, `scikit-learn` o `TensorFlow`. Este enfoque dotaría al sistema de una capacidad proactiva, pasando de la mera monitorización a la previsión a corto plazo.
- **Generación de Informes y Alertas:** Para hacer los datos más accesibles y relevantes, se contempla el desarrollo de funcionalidades para la generación automatizada de informes periódicos (diarios, semanales o mensuales) que incluyan resúmenes, estadísticas clave y gráficos generados dinámicamente en formatos amigables como PDF o HTML. Tecnologías como `matplotlib`, `plotly` o `WeasyPrint` podrían utilizarse en este contexto. Además, se propone la implementación de un sistema de alertas personalizables que permita a los usuarios configurar notificaciones (por ejemplo, por correo electrónico, Telegram o notificaciones *push*) en función de umbrales definidos para variables críticas, como temperaturas extremas, ráfagas de viento o cambios bruscos de presión.

V. Seguridad

- **Medidas de Seguridad Robustas:** Dada la potencial exposición de una API pública y la posibilidad de múltiples usuarios conectados, es fundamental implementar medidas de seguridad avanzadas. Se propone la aplicación de técnicas de *rate-limiting* no solo por dirección IP, sino también por geolocalización y por usuario autenticado, con el fin de mitigar ataques de denegación de servicio distribuido (DDoS) y prevenir abusos del sistema.

Además, se plantea el uso de mecanismos de autenticación seguros (como claves API o tokens JWT), el cifrado de las comunicaciones mediante HTTPS/TLS, y la validación estricta de los mensajes entrantes para evitar inyecciones o datos malformados. La incorporación de registros de actividad y auditoría permitiría detectar patrones anómalos y reforzar la trazabilidad. Estas medidas contribuirían a proteger tanto la integridad de los datos como la disponibilidad del sistema frente a accesos no autorizados o maliciosos.

V. Integración y Sostenibilidad

- **Integración con Servicios de la Escuela y Despliegue en la Nube:** La estación meteorológica ofrece una oportunidad única para su integración en el ecosistema digital de la institución educativa. Una posible línea de desarrollo sería la creación de un *widget* o panel simplificado que pudiera incrustarse directamente en la página web de la escuela, mostrando los datos más relevantes en tiempo real y aumentando la visibilidad del proyecto.

  Más allá de la mera visualización, los datos recopilados podrían ponerse a disposición de otros departamentos o iniciativas académicas, fomentando el uso interdisciplinar en proyectos educativos, científicos o de divulgación.

  Para garantizar la escalabilidad, disponibilidad y sostenibilidad del sistema, se propone el despliegue de los componentes clave —como InfluxDB, Grafana y la futura API— en una plataforma en la nube, ya sea bajo un modelo PaaS o IaaS (por ejemplo, InfluxDB Cloud, Fly.io o un servidor VPS). Esto permitiría desacoplar la operatividad de la Raspberry Pi respecto a la infraestructura de presentación y almacenamiento, facilitando el mantenimiento, la gestión de recursos y el crecimiento futuro ante un mayor volumen de datos o usuarios concurrentes.

- **Conexión con la red UVA-IoT:** Otra posible mejora sería la integración del sistema con la red institucional UVA-IoT, desplegada por la Universidad de Valladolid para proyectos de sensorización distribuida. Esta red permitiría conectar la Raspberry Pi de forma segura a través de una infraestructura ya configurada, evitando la necesidad de exponer el sistema mediante IP pública o port forwarding. Durante el desarrollo del proyecto se realizaron pruebas preliminares con esta red, aunque no se consiguió completar la conexión debido a restricciones de autenticación y configuración. Retomar esta línea permitiría aprovechar una plataforma robusta, segura y mantenida por la institución, facilitando la escalabilidad y la accesibilidad del sistema en contextos académicos y colaborativos.

En conjunto, estas líneas futuras no solo apuntan a extender las capacidades técnicas del sistema, sino también a consolidar su viabilidad como herramienta educativa, colaborativa y de investigación. Lejos de ser un proyecto cerrado, la arquitectura desarrollada permite iterar y evolucionar con facilidad, adaptándose a nuevos requerimientos, contextos de uso o avances tecnológicos. Esta flexibilidad constituye una de las principales fortalezas del trabajo y abre la puerta a su aplicación real y sostenida en el tiempo.

7. Conclusiones

El presente trabajo ha desarrollado un sistema completo de monitorización meteorológica en tiempo real, combinando tecnologías modernas como MQTT, InfluxDB, Docker y Grafana, todo desplegado sobre una Raspberry Pi como núcleo operativo. A lo largo del proyecto se han abordado tanto los aspectos técnicos de implementación como la validación práctica mediante simulaciones controladas.

La arquitectura propuesta, modular y desacoplada, ha demostrado ser robusta, eficiente y fácilmente escalable. Las pruebas realizadas —desde la comunicación básica hasta la simulación de carga equivalente a miles de estaciones— han permitido verificar la solidez del flujo de datos, desde la publicación MQTT hasta su representación gráfica en el panel de Grafana. El sistema respondió de forma estable incluso en escenarios de alta concurrencia, sin pérdidas de mensajes ni degradaciones relevantes del rendimiento.

Uno de los logros clave ha sido la automatización parcial del entorno, incluyendo mecanismos de respaldo, arranque de servicios y visualización web. Además, se ha validado la posibilidad de simular valores dinámicos y representarlos visualmente con precisión, lo que permite anticipar el funcionamiento del sistema una vez conectado a sensores reales.

Respecto al potencial de mejora, el trabajo deja sentadas bases sólidas para futuras extensiones: análisis predictivo mediante técnicas de *machine learning*, despliegue en la nube para mejorar la disponibilidad, y creación de una API pública que fomente la colaboración y expansión geográfica del sistema.

En resumen, el sistema cumple con los objetivos propuestos, aportando una solución funcional, educativa y técnicamente solvente para la adquisición, almacenamiento y visualización de datos meteorológicos. Este TFG constituye un punto de partida sólido para desarrollos más complejos, manteniendo siempre la filosofía de apertura, modularidad y escalabilidad.

Además, el sistema puede servir como base para nuevas iniciativas docentes o de investigación dentro del ámbito de la sensorización ambiental y la divulgación científica.

Referencias

[1] Davis Instruments. Davis instruments - weather monitoring, 2025. Consultado en mayo de 2025. URL: https://www.davisinstruments.com/.

[2] Docker Inc. *Compose Documentation*, 2025. Consultado en abril de 2025. URL: https://docs.docker.com/compose/.

[3] Docker Inc. *Docker Documentation*, 2025. Consultado en abril de 2025. URL: https://docs.docker.com/.

[4] Duck DNS. Duck dns - free dynamic dns, 2025. Consultado en junio de 2025. URL: https://www.duckdns.org.

[5] Elastic NV. Kibana documentation, 2025. Consultado en mayo de 2025. URL: https://www.elastic.co/guide/en/kibana/index.html.

[6] Eclipse Foundation. Eclipse mosquitto - an open source mqtt broker, 2025. Consultado en abril de 2025. URL: https://mosquitto.org.

[7] Eclipse Foundation. paho-mqtt 1.6.1 documentation, 2025. Consultado en abril de 2025. URL: https://www.eclipse.org/paho/index.php?page=clients/python/index.php.

[8] Python Software Foundation. datetime — basic date and time types, 2025. Consultado en abril de 2025. URL: https://docs.python.org/3/library/datetime.html.

[9] Python Software Foundation. json — json encoder and decoder, 2025. Consultado en abril de 2025. URL: https://docs.python.org/3/library/json.html.

[10] Python Software Foundation. time — time access and conversions, 2025. Consultado en abril de 2025. URL: https://docs.python.org/3/library/time.html.

[11] Raspberry Pi Foundation. Raspberry pi 4 model b, 2025. Consultado en abril de 2025. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[12] Grafana Labs. *Grafana Documentation*, 2025. Consultado en abril de 2025. URL: https://grafana.com/docs/grafana/latest/.

[13] IBM Corporation. Websocket applications, 2025. Consultado en mayo de 2025. URL: https://www.ibm.com/docs/es/was/9.0.5?topic=applications-websocket.

[14] InfluxData. *InfluxDB Documentation*, 2025. Consultado en abril de 2025. URL: https://docs.influxdata.com/influxdb/v2.7/.

[15] InfluxData. Influxdb python client library, 2025. Consultado en abril de 2025. URL: https://github.com/influxdata/influxdb-client-python.

[16] InfluxData. Telegraf documentation, 2025. Consultado en junio de 2025. URL: https://docs.influxdata.com/telegraf/.

[17] Canonical Ltd. Install ubuntu on a raspberry pi, 2025. Recuperado en abril de 2025. URL: https://ubuntu.com/download/raspberry-pi.

[18] Meteoclimatic. Meteoclimatic: Red de estaciones meteorológicas, 2025. Consultado en abril de 2025. URL: https://www.meteoclimatic.net.

[19] MongoDB Inc. *Time Series Collections*, 2025. Consultado en mayo de 2025. URL: https://www.mongodb.com/docs/manual/core/timeseries/.

[20] Netatmo. Netatmo - smart weather station, 2025. Consultado en mayo de 2025. URL: https://www.netatmo.com/.

[21] OASIS Standard. *MQTT Version 3.1.1*, 2014. Consultado en abril de 2025. URL: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[22] Oracle Corporation. *MySQL 8.4 Reference Manual*, 2025. Consultado en mayo de 2025. URL: https://dev.mysql.com/doc/refman/8.4/en/introduction.html.

[23] Red Hat, Inc. Podman - manage pods, containers and container images, 2025. Consultado en mayo de 2025. URL: https://podman.io/.

[24] Tableau Software. Tableau - business intelligence and analytics, 2025. Consultado en mayo de 2025. URL: https://www.tableau.com/.

[25] The Weather Company. Weather underground: Personal weather station network, 2025. Consultado en abril de 2025. URL: https://www.wunderground.com/pws.

[26] W3C. Http overview, 2025. Consultado en junio de 2025. URL: https://www.w3.org/Protocols/.

[27] Z. Shelby and K. Hartke and C. Bormann. The constrained application protocol (coap). https://datatracker.ietf.org/doc/html/rfc7252, 2014. RFC 7252, IETF. Consultado en mayo de 2025.

Anexos

Anexo A. Fichero docker-compose.yml

A continuación se muestra el contenido completo del fichero `docker-compose.yml` utilizado para el despliegue del sistema.

Listing 17: Contenido del fichero docker-compose.yml

```
version: '3.8'

services:
  mosquitto:
    image: eclipse-mosquitto:2.0
    container_name: mosquitto
    restart: always
    ports:
      - "1883:1883"
      - "9001:9001"
    volumes:
      - ./docker/mosquitto/config:/mosquitto/config
      - ./docker/mosquitto/data:/mosquitto/data

  influxdb:
    image: influxdb:2.7
    container_name: influxdb
    restart: always
    ports:
      - "8086:8086"
    volumes:
      - ./docker/influxdb/data:/var/lib/influxdb2
      - ./docker/influxdb/config:/etc/influxdb2
    environment:
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=**********
      - DOCKER_INFLUXDB_INIT_PASSWORD=**********
      - DOCKER_INFLUXDB_INIT_ORG=teleco
      - DOCKER_INFLUXDB_INIT_BUCKET=estacion1
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=supersecreto123

  grafana:
    image: grafana/grafana-oss:10.2.3
    container_name: grafana
    user: "472:472"   # <- Esta linea fuerza a usar el UID correcto
    environment:
      - GF_SECURITY_ADMIN_USER=**********
      - GF_SECURITY_ADMIN_PASSWORD=**********
    volumes:
      - ./docker/grafana/data:/var/lib/grafana
    ports:
      - "3000:3000"
    restart: always
```

Anexo B. Script mqtt_listener.py

A continuación se muestra el contenido completo del script `mqtt_listener.py`, encargado de recibir los datos desde Mosquitto, procesarlos y almacenarlos en InfluxDB.

Listing 18: Contenido del fichero mqtt_listener.py

```
import paho.mqtt.client as mqtt
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS
import datetime
import json
import time

# MQTT Config
broker = "localhost"
port = 1883
topic_base = "/teleco/#"

TOPICS_VALIDOS = [
    "/teleco/estacion1",
    "/teleco/estacion2"
]

username = "*****"
password = "*****"

# InfluxDB Config
influx_url = "http://localhost:8086"
influx_token = "*****"
influx_org = "*****"
influx_bucket = "*****"

# Mapeo de claves cortas a nombres descriptivos
MAPEO = {
    "t": "temperatura",
    "h": "humedad",
    "pn": "presion",
    "v": "veleta",
    "a": "anemometro",
    "pl": "pluviometro",
    "r": "radiacion"
}

# Crear cliente InfluxDB
influx_client = InfluxDBClient(url=influx_url, token=influx_token, org=
    influx_org)
write_api = influx_client.write_api(write_options=SYNCHRONOUS)

# Callback al recibir mensaje MQTT
def on_message(client, userdata, message):
    topic = message.topic
    payload = message.payload.decode()
```

```
    if topic in TOPICS_VALIDOS:
        print(f"[MQTT] {topic}: {payload}")

        try:
            datos = json.loads(payload.strip())

            # Convertir claves y valores
            campos = {}
            for clave, valor in datos.items():
                clave_larga = MAPEO.get(clave, clave)
                try:
                    campos[clave_larga] = float(valor)
                except ValueError:
                    print(f"Valor invalido para '{clave_larga}': {valor
                        }")

            # Crear punto InfluxDB con zona horaria UTC
            point = Point("mediciones").tag("estacion", topic).time(
                datetime.datetime.now>
            for k, v in campos.items():
                point = point.field(k, v)

            write_api.write(bucket=influx_bucket, org=influx_org,
                record=point)
            print("Datos guardados en InfluxDB")

        except json.JSONDecodeError as e:
            print(f"Error de formato JSON: {e}")

    else:
        print(f"Topic no permitido: {topic}")

# Configurar cliente MQTT
while True:
    try:
        client = mqtt.Client()
        client.username_pw_set(username, password)
        client.on_message = on_message

        client.connect(broker, port)
        client.subscribe(topic_base)
        print(f"Escuchando en {topic_base}...")
        client.loop_forever()  # Se bloquea aqui hasta que se pierde la
            conexion

    except Exception as e:
        print(f"Error durante conexion o ejecucion del loop MQTT: {e}")
        print("Reintentando en 5 segundos...")
        time.sleep(5)
```

Anexo C. Dashboard en formato JSON

A continuación se muestra un fragmento representativo del archivo JSON exportado desde Grafana. Este archivo contiene toda la configuración del dashboard utilizado en el proyecto, incluyendo los paneles, consultas, métricas, y opciones de visualización. El archivo completo puede exportarse directamente desde la opción `Dashboard settings → JSON model` dentro de la interfaz de Grafana.

Listing 19: Fragmento del archivo JSON del dashboard exportado

```
{
  "title": "Estacion Meteorologica - TFG (Completo)",
  "uid": "***********",
  "panels": [
    {
      "title": "Ultimos valores de todos los sensores",
      "type": "table",
      "targets": [
        {
          "query": "from(bucket: \"estacion1\")\r\n  |> range(start:
             -30d)\r\n  |> filter(fn: (r) => r._measurement == \"
             mediciones\")\r\n  |> last()\r\n"
        }
      ]
    },
    {
      "title": "Temperatura ( C ))",
      "type": "timeseries",
      "targets": [
        {
          "query": "from(bucket: \"estacion1\")\r\n  |> range(start: -7
             d)\r\n  |> filter(fn: (r) => r._field == \"temperatura\")
             "
        }
      ]
    },
    {
      "title": "Humedad relativa (%)",
      "type": "timeseries",
      "targets": [
        {
          "query": "from(bucket: \"estacion1\")\r\n  |> range(start: -7
             d)\r\n  |> filter(fn: (r) => r._field == \"humedad\")"
        }
      ]
    },
    ...
  ]
}
```

Nota: por motivos de espacio, se ha incluido solo una parte del archivo. El JSON completo puede ser regenerado desde la propia interfaz de Grafana.

Anexo D. Script de simulación para pruebas de visualización

A continuación se muestra el contenido completo del script utilizado para probar la visualización en el dashboard.

Listing 20: Script de simulación de carga MQTT

```
import paho.mqtt.client as mqtt
import time
import json
import math
import random

# Configuracion MQTT
broker = "192.168.1.29"
port = 1883
topic = "/teleco/estacion1"

username = "admin"
password = "admin"

client = mqtt.Client()
client.username_pw_set(username, password)
client.connect(broker, port)

# Duracion: 5 minutos, envio cada 1 segundo
duracion_total = 300
intervalo = 1
pasos = int(duracion_total / intervalo)

print("Iniciando simulacion visual con datos dinamicos...")

for i in range(pasos):
    t = i * intervalo

    # Valores dinamicos simulados
    temperatura = 22 + 5 * math.sin(4 * math.pi * t / duracion_total)
    humedad = 60 + 20 * math.cos(4 * math.pi * t / duracion_total)
    presion = 1010 + 3 * math.sin(2 * math.pi * t / 180)
    anemometro = 5 + 2 * math.sin(6 * math.pi * t / duracion_total)
    pluviometro = round(random.uniform(0, 0.4), 2) if i % 30 < 10 else
       0
    radiacion = 300 + 300 * math.pow(math.sin(math.pi * t /
       duracion_total), 2)
    direccion_viento = round(random.uniform(0, 360), 2)

    payload = {
        "t": round(temperatura, 2),
        "h": round(humedad, 2),
        "pn": round(presion, 2),
        "a": round(anemometro, 2),
        "pl": pluviometro,
        "r": round(radiacion, 2),
        "v": direccion_viento
```

```
    }

    client.publish(topic, json.dumps(payload))
    print(f"[MQTT] Enviado: {payload}")
    time.sleep(intervalo)

print("Simulacion finalizada.")
```

Anexo E. Script de simulación de carga MQTT

A continuación se muestra el contenido completo del script de simulación de carga MQTT..

Listing 21: Script de simulación de carga MQTT

```
import paho.mqtt.client as mqtt
import time
import random
import json

# Configuracion MQTT
broker = "localhost"
port = 1883
topic = "/teleco/estacion1"

client = mqtt.Client()
client.connect(broker, port)

# Funcion para generar datos aleatorios
def generar_datos():
    return {
        "t": round(random.uniform(15.0, 30.0), 1),
        "h": random.randint(30, 90),
        "pn": random.randint(980, 1030),
        "a": round(random.uniform(0, 15), 1),
        "pl": round(random.uniform(0, 2), 2),
        "r": random.randint(100, 900)
    }

# Enviar datos durante un periodo concreto
def enviar_durante(segundos, intervalo=5):
    fin = time.time() + segundos
    while time.time() < fin:
        payload = json.dumps(generar_datos())
        client.publish(topic, payload)
        print(f"[MQTT] Enviado: {payload}")
        time.sleep(intervalo)

# 1. Esperar 30 segundos
print("Esperando 30 segundos...")
time.sleep(30)

# 2. Enviar durante 1 minuto y medio
print("Enviando durante 90 segundos...")
enviar_durante(90)

# 3. Pausa 1 minuto
print("Pausa de 60 segundos...")
time.sleep(60)

# 4. Enviar otra vez durante 1 minuto y medio
print("Enviando otra vez durante 90 segundos...")
enviar_durante(90)
```

```
# 5. Pausa final de 30 segundos
print("Pausa final de 30 segundos...")
time.sleep(30)

print("Simulacion terminada.")
```