Desarrollo de un Sistema de Microservicios Orientado a Eventos con Message Broker para un Proveedor de Identidad (IdP) usando protocolo MQTT

Mateo Alejandro Bravo Revelo

Informe de práctica presentado para optar al título de Ingeniero Electrónico

Modalidad de Práctica

Semestre de Industria o Práctica Empresarial

Asesores:

Hernán Felipe García Arias, Doctor (PhD) en Ingeniería Electrónica

Sergio Nicolás Posado Chaparro, Profesional (Prof) en Ingeniería Electrónica

Universidad de Antioquia

Facultad de Ingeniería, Departamento de Ingeniería Electrónica y Telecomunicaciones

Ingeniería Electrónica

Medellín, Antioquia, Colombia

2025

Cita	(Bravo Revelo, 2025) [1]
Referencia Estilo IEEE (2020)	[1] Bravo Revelo M. (2025), "Desarrollo de un Sistema de Microservicios Orientado a Eventos con Message Broker para un Proveedor de Identidad (IdP) usando protocolo MQTT", Informe de práctica, Ingeniería Electrónica, Universidad de Antioquia Medellín, Antioquia, Colombia, 2025.

UNIVERSIDAD DE ANTIOQUIA
Vicerrectoría de Docencia

Sistema de Bibliotecas

Centro de Documentación Ingeniería (CENDOI)

Repositorio Institucional: http://bibliotecadigital.udea.edu.co

Universidad de Antioquia - www.udea.edu.co

Rector: John Jairo Arboleda Céspedes.

Decano/Director Julio César Saldarriaga Molina.

Jefe departamento: Eduard Emiro Rodríguez Ramírez.

TABLA DE CONTENIDO

LISTA DE FIGURAS

Siglas, acrónimos y abreviaturas

MQTT	Message Queuing Telemetry Transport
IoT	Internet de las cosas
IdP	Provedor de Identidad
API	Application Programming Interface
HTTP	HyperText Transfer Protocol
WiFi	Wireless Fidelity
DB	Base de datos
MCU	Microcontrolador
MHz	Megahertz
PhD	Philosophiae Doctor
UdeA	Universidad de Antioquia

RESUMEN

El proyecto se centró en el desarrollo de una arquitectura para un Proveedor de Identidad (IdP) basada en microservicios y orientada a eventos. Esta arquitectura utiliza un broker de mensajería y el protocolo MQTT para orquestar la comunicación entre los componentes, a través de la publicación y suscripción a tópicos. El IdP gestiona la autenticación y validación del dispositivo productor, mientras que el broker se encarga de distribuir los mensajes entre los distintos microservicios.

Cada microservicio tiene una responsabilidad específica en el sistema: procesamiento, análisis y almacenamiento de los datos generados por el productor. Este enfoque permite una estructura modular que facilita el mantenimiento, ya que cada componente puede ser actualizado o reemplazado de forma independiente, sin afectar al resto del sistema. Además, esta modularidad permite la integración de nuevas funcionalidades de forma escalable, a diferencia de los modelos monolíticos tradicionales, en los que todos los procesos están integrados en una única base de código. El proyecto se desarrolló en tres fases: primero, el diseño de la arquitectura; luego, su implementación; y finalmente, la validación mediante pruebas funcionales, comprobando la correcta comunicación entre los componentes y el cumplimiento de los principios de modularidad y escalabilidad del sistema.

Palabras clave **— Microservicios, Proveedor de Identidad, Broker de Mensajería, Protocolo MQTT, Productor, Consumidor, Internet de las Cosas**

ABSTRACT

The project focused on the development of an architecture for an Identity Provider (IdP) based on microservices and event-driven design. This architecture uses a messaging broker and the MQTT protocol to orchestrate communication between components through topic publication and subscription. The IdP is responsible for managing the authentication and validation of the producer device, while the broker handles the distribution of messages across the different microservices.

Each microservice has a specific role within the system: processing, analyzing, and storing the data generated by the producer. This approach enables a modular structure that simplifies maintenance, as each component can be updated or replaced independently without affecting the rest of the system. Furthermore, this modularity allows for scalable integration of new functionalities, in contrast to traditional monolithic models where all processes are tightly coupled within a single codebase. The project was carried out in three main phases: first, the design of the architecture; second, its implementation; and finally, validation through functional testing, verifying correct communication among components and compliance with the principles of modularity and scalability.

Keywords **— Microservices, Identity Provider (IdP), Message Broker, MQTT Protocol, Producer, Consumer, Internet of Things (IoT)**

1. INTRODUCCIÓN

Intertelco S.A.S. es una empresa dedicada al desarrollo de hardware y software para soluciones tecnológicas a la medida, respondiendo a las necesidades específicas de sus clientes y los entornos en los que operan. La empresa se enfoca especialmente en el área de la seguridad, desarrollando soluciones para fortalecer el control y la vigilancia en diferentes entornos, adaptándose a los requerimientos específicos de cada situación [1].

Dentro de los desarrollos de Inter-Telco S.A.S está en construcción un Proveedor de Identidad (IdP) un sistema que permite validar de forma segura la autenticación. Este desarrollo tiene un papel importante al ampliar las capacidades de las soluciones de seguridad que proporciona la empresa. Con el aumento de la complejidad y la cantidad de los dispositivos provoca el manejo de grandes volúmenes de información que se debe manejar existe la necesidad de replantear las arquitecturas convencionales. Reemplazar los modelos monolíticos por arquitecturas basadas en microservicios permitiendo desarrollar aplicaciones escalables y más fáciles de mantener, asi este enfoque facilita la integración de nuevas funcionalidades. Para abordar las anteriores necesidades, el backend del IdP se pretende desarrollarlo con el anterior enfoque empleando un broker de mensajería para gestionar la comunicación entre los microservicios, asegurando la entrega organizada y confiable de datos entre dispositivos.

Para alcanzar el desarrollo funcional del backend, es fundamental diseñar la arquitectura general que integra los dos microservicios involucrados y configurar de manera correcta el broker de mensajería para garantizar la comunicación entre productor y microservicios. Dentro de la implementación general incluye la configuración de productor y consumidor de información, utilizando el protocolo MQTT como base de toda la arquitectura, Además se prioriza el uso de programación asíncrona para obtener un buen rendimiento y capacidad de respuesta del sistema.

2. OBJETIVOS

A. Objetivo general

Desarrollar un backend para un Proveedor de Identidad (IdP) basado en una arquitectura de microservicios, que asegure la comunicación confiable entre productores y consumidores de datos, garantizando escalabilidad y modularidad del sistema.

B. Objetivos específicos

- Diseñar la arquitectura del sistema IoT, definiendo el proveedor de identidad (IdP) para gestionar el control de acceso del microcontrolador, así como la configuración del Message Broker y la funcionalidad de los microservicios, especificando el rol del consumidor en el proceso.
- Implementar un proveedor de identidad (IdP) para la gestión de la autenticación y validación del microcontrolador en el sistema IoT, incluyendo la configuración del Message Broker basado en MQTT para la publicación de información, el desarrollo del microcontrolador como productor de datos y la creación de microservicios para el almacenamiento y procesamiento de la información.
- Validar el funcionamiento de la arquitectura implementada mediante pruebas, asegurando la correcta autenticación, la comunicación entre el productor y los microservicios, y la correcta ejecución del procesamiento y almacenamiento de datos por parte de los microservicios.

3. MARCO TEÓRICO

A. Broker Active MQ Artemis

ActiveMQ Artemis es un broker de mensajería de código abierto, desarrollado bajo la licencia Apache 2.0, que es ampliamente utilizado debido a su compatibilidad con múltiples lenguajes y plataformas. En esta implementación, se utiliza con Python. Este broker permite gestionar dispositivos IoT mediante el protocolo MQTT. Además, ofrece opciones para configurar reglas de seguridad y control de acceso, lo que garantiza una gestión estricta en sistemas distribuidos. Este broker es adecuado para arquitecturas de microservicios altamente escalables, ya que emplea un patrón basado en eventos y mensajería asincrónica [2], lo que optimiza el rendimiento y la fiabilidad. En esta arquitectura, el broker cumple una función clave al recibir la información del microcontrolador ESP32 y enrutar los mensajes de manera adecuada hacia los dos microservicios [3].

B. Message Queuing Telemetry Transport (MQTT) :

Es un protocolo de mensajería diseñado para dispositivos de bajo consumo, especialmente en redes inestables o con ancho de banda limitado. Permite una comunicación asincrónica y funciona bajo el modelo publicar-suscribir (pub/sub). Esto significa que los dispositivos (productores y consumidores) no se comunican directamente entre sí, sino que lo hacen a través de un broker, que se encarga de distribuir los mensajes desde los productores hacia los suscriptores. Además de ser asincrónico, MQTT es escalable, lo que permite la comunicación entre miles de dispositivos. El protocolo se organiza mediante tópicos, que permiten clasificar y organizar los mensajes de manera eficiente [4]. En este proyecto, se emplearán cuatro tópicos, dos de los cuales estarán asignados al usuario 1 y los otros dos al usuario 2, con el fin de gestionar la autenticación y el acceso a los datos.

C. Arquitectura de microservicios:

Es un enfoque de diseño que divide las aplicaciones y desarrollos en módulos pequeños, donde cada módulo cumple una función específica. Esta división mejora la escalabilidad, mantenibilidad y despliegue de las aplicaciones [5], en comparación con las arquitecturas monolíticas, donde las funcionalidades están integradas en un solo bloque. En este desarrollo, los microservicios juegan un papel crucial en el procesamiento y almacenamiento de mensajes [6]. El microcontrolador, actuando como productor, envía la información que llega al Broker, el cual distribuye los datos a los microservicios de procesamiento. Estos microservicios se encargan de filtrar y analizar los datos recibidos, mientras que otro microservicio de almacenamiento guarda la información en una base de datos para consultas y publicaciones posteriores. La independencia entre servicios permite escalar o actualizar más microservicios sin la necesidad de modificar los demás, proporcionando flexibilidad para editar funcionalidades dentro de la arquitectura [7].

D. Sistemas de gestión y visualización de Bases de Datos:

En primer lugar, se encuentra InfluxDB, una base de datos diseñada para almacenar y gestionar series temporales de datos. Este tipo de base de datos está orientado a la recolección, almacenamiento y análisis de datos indexados por tiempo, lo que permite realizar un seguimiento de los mismos en intervalos regulares, como en el caso de los sensores IoT utilizados para monitoreo. InfluxDB cuenta con su propio lenguaje de consultas, denominado InfluxQL, que es similar a SQL [8], lo que facilita su integración con Grafana, una plataforma de código abierto destinada a la visualización y análisis de datos. Grafana permite crear paneles de visualización a partir de la integración con InfluxDB, lo que resulta útil para mostrar los datos en tiempo real [9]. En el contexto de este proyecto, un microservicio está dedicado tanto al almacenamiento de datos en InfluxDB como a su visualización a través de Grafana y otro microservicio está dedicado al procesamiento y almacenamiento [10].

E. Proveedor de Identidad (IdP) :

Es un sistema encargado de autenticar a los usuarios y gestionar sus identidades. Su función principal es verificar la identidad de los usuarios y permitir el acceso a recursos protegidos. Este proceso es esencial en sistemas distribuidos, ya que centraliza la gestión de usuarios, roles y permisos, garantizando un control de acceso seguro y eficiente [11]. En este proyecto, el IdP permite que los usuarios publiquen en un microservicio específico a través de un tópico determinado, utilizando su nombre de usuario y contraseña.

4. METODOLOGÍA

Para el cumplimiento de los objetivos propuestos, el desarrollo se organiza en tres etapas: diseño de la arquitectura, que incluye la selección de las tecnologías para la implementación, desarrollo de la implementación y pruebas y validaciones.

A. Diseño de la Arquitectura del Sistema

Dentro del diseño de la arquitectura del sistema, es fundamental considerar los requisitos funcionales y no funcionales, ya que estos definen tanto lo que el sistema debe realizar como las condiciones bajo las cuales debe operar.

Los requisitos funcionales especifican de manera puntual las funcionalidades que el sistema debe cumplir para satisfacer las necesidades del usuario, como la autenticación del productor, la publicación de datos a través del protocolo MQTT y el procesamiento de dicha información por los microservicios. Por su parte, los requisitos no funcionales establecen cómo debe comportarse el sistema, abarcando aspectos como el rendimiento, la confiabilidad, la escalabilidad, la disponibilidad y la seguridad. Estos requisitos permiten garantizar que el sistema no solo funcione correctamente, sino que lo haga de manera eficiente, segura y adaptable ante futuras ampliaciones.

La arquitectura del sistema se compone de varios componentes interconectados que colaboran entre sí para garantizar el funcionamiento adecuado del sistema. En primer lugar, se encuentra el productor de datos, para el cual se ha elegido un microcontrolador (MCU). Este MCU está diseñado para generar datos sintéticos provenientes de sensores, los cuales, en el futuro, podrían ser reemplazados por sensores reales. Para la elección del MCU, se optó por el ESP32, debido a varias ventajas. Este dispositivo cuenta con un módulo Wi-Fi y Bluetooth integrado en el chip, lo que lo hace especialmente adecuado para aplicaciones IoT, ya que permite una conexión fácil a redes locales. Además, ofrece una frecuencia de reloj superior en comparación con otros microcontroladores similares ya que alcanza hasta 240MHz, y su

costo es más bajo en comparación con plataformas más avanzadas, lo que genera una relación costo-beneficio muy favorable. El ESP32 también es compatible con protocolos IoT, como el caso de MQTT, lo que facilita su integración con brokers de mensajería.

Por otro lado, el broker de mensajería, que actúa como intermediario entre la publicación de datos desde el productor (ESP32) y la suscripción por parte de los microservicios, se eligió ActiveMQ Artemis con el protocolo MQTT. Esta solución, al ser software libre, es ideal para arquitecturas distribuidas y tiene la capacidad de manejar grandes volúmenes de datos, alcanzando miles de mensajes por segundo. Además, ActiveMQ Artemis ofrece alta disponibilidad, lo que garantiza la integridad de los datos al evitar pérdidas y permitir su recuperación frente a fallos. También presenta un bajo consumo de recursos, lo cual es importante, dado que aunque el ESP32 tiene un rendimiento adecuado, es preferible optimizar el uso de los recursos. ActiveMQ Artemis se distingue por su facilidad de integración y por ofrecer una interfaz de administración web que permite supervisar los brokers de manera sencilla mediante una interfaz gráfica.

En cuanto a los microservicios, se ha elegido Python como lenguaje de programación. Esta elección se debe a que Python permite un desarrollo ágil, facilitando la implementación rápida gracias a su enfoque en la simplicidad. Sin embargo, esto no limita el rendimiento del desarrollo, ya que Python ofrece soporte nativo para programación asincrónica, lo que mejora el rendimiento de los servicios que manejan múltiples solicitudes concurrentes. Además, la integración con FastAPI para la creación de APIs para los microservicios permite optimizar aún más el desarrollo y la eficiencia de las aplicaciones [12].

Para facilitar la comprensión de la arquitectura del sistema, se presenta el diagrama ilustrado en la **figura 1**, el cual muestra la interacción entre sus principales componentes. En este esquema, el ESP32-WROOM-32 cumple la función de productor de datos, generando y enviando información en forma de mensajes a través del protocolo MQTT. Estos mensajes son publicados en un tópico específico del broker de mensajería (ActiveMQ Artemis), que

actúa como intermediario en la comunicación. El broker se encarga de almacenar temporalmente los mensajes y distribuirlos a los consumidores que se encuentren suscritos al tópico correspondiente. En el diagrama se representan dos microservicios consumidores, los cuales serán desarrollados en Python utilizando el framework FastAPI. Estos microservicios tienen como objetivo almacenar los datos en una base de datos temporal orientada a series de tiempo (InfluxDB) y permitir su posterior visualización (Grafana).

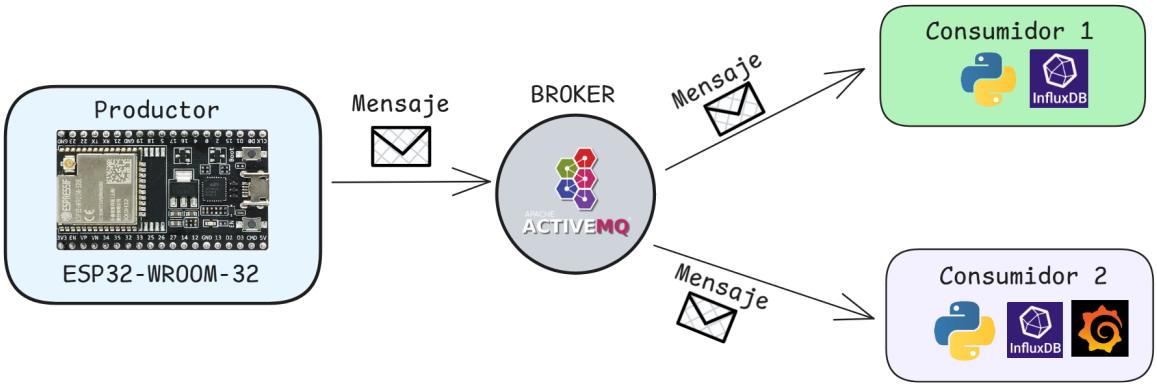


Fig. 1. Arquitectura del sistema propuesta, incluyendo el flujo de datos desde el productor hasta los microservicios y la visualización.

B. Desarrollo de la arquitectura del sistema

Una vez seleccionadas todas las tecnologías que conformaran el sistema, se procede al desarrollo de la arquitectura.

1) Conexión productor-broker MQTT: El primer paso consiste en establecer la conexión entre el productor de datos (ESP32) y el broker de mensajería. Para ello, se utiliza ActiveMQ Artemis como broker MQTT. Se consulta la documentación oficial y se descarga la versión 2.40.0 desde el sitio web del proyecto. El broker se ejecuta en una máquina local con sistema operativo Windows 11. El archivo descargado viene comprimido, por lo que se descomprime en una ruta específica. Posteriormente, se crea una instancia denominada ActiveMQ-IoT, en

la cual se configuran un nombre de usuario y una contraseña para el acceso a la administración del sistema.

Una vez finalizada la configuración inicial, se inicia el broker, el cual queda disponible en el host local (localhost), utilizando el puerto 8161. A través de una solicitud HTTP en el navegador (por ejemplo, http://localhost:8161), se accede a la interfaz gráfica de administración, desde donde es posible monitorear las conexiones activas, los tópicos configurados y el flujo de mensajes. Como se observa en la **figura 2**, se han definido cuatro tópicos en el broker MQTT: temperatura, wifi, humedad y presión. Cada uno de estos tópicos recibe mensajes publicados de forma separada por el módulo ESP32, que actúa como productor de datos. Estos datos son generados de manera sintética con fines prácticos. La lógica de publicación del ESP32 permite enviar periódicamente la información correspondiente a cada tópico, facilitando así su procesamiento posterior por parte de los microservicios suscritos.

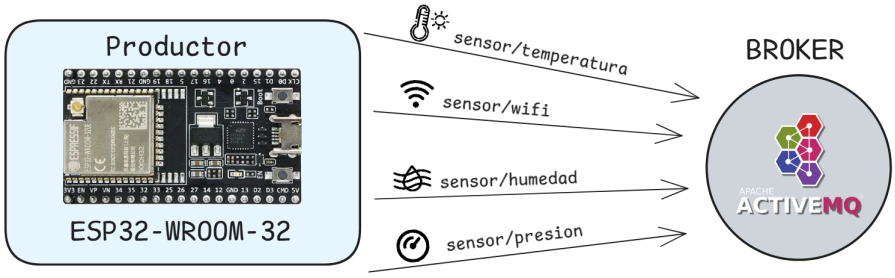


Fig. 2. Cuatro tópicos MQTT configurados para la publicación del productor.

2) Conexión broker MQTT - consumidor: Una vez verificada la correcta conexión entre el productor de datos (ESP32) y el broker MQTT, se procede con el desarrollo de los microservicios que cumplen el rol de consumidores. Estos microservicios se encargan de suscribirse a los tópicos definidos en el broker para recibir y procesar los mensajes publicados. Dado que los datos ya están siendo enviados periódicamente por el productor, el objetivo ahora es garantizar que los microservicios capturen correctamente esta información y la gestionen.

Como se observa en la **figura 3**, se implementan dos microservicios en Python utilizando la librería `paho.mqtt.client`, que permite establecer clientes MQTT eficientes bajo un enfoque de programación no bloqueante. El primer microservicio se suscribe a los tópicos `sensor/temperatura` y `sensor/wifi`. Su objetivo es almacenar los datos recibidos en una base de datos InfluxDB, la cual se ejecuta localmente y es accesible mediante `http://localhost:8086`. Los datos se registran en intervalos de 5 segundos. Posteriormente, se configura la herramienta Grafana, disponible en `http://localhost:3000`, para visualizar los datos en un panel de control (dashboard) mediante consultas a la base de datos.

El segundo microservicio se suscribe a los tópicos `sensor/humedad` y `sensor/presion`. Este servicio realiza un filtrado de los datos y los almacena en una base de datos distinta, también implementada con InfluxDB pero configurada por separado, lo que permite una separación lógica de los datos recolectados.

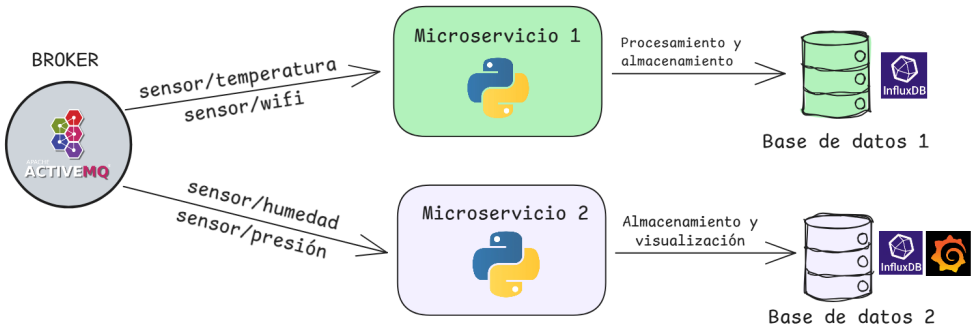


Fig. 3. Esquema del flujo de datos desde el broker MQTT hacia dos microservicios especializados.

3) Configuración de autenticación y control de acceso en el broker: Como parte del proceso de aseguramiento y segmentación de los datos publicados, se configura el broker ActiveMQ Artemis para implementar autenticación de usuarios y control de acceso por tópicos. Para ello, se definen dos usuarios en los archivos `artemis-users.properties` y `artemis-roles.properties`. Cada usuario se asocia con un rol específico:

- usuario1 con el rol *rol1*, autorizado únicamente para publicar en los tópicos `sensor/temperatura` y `sensor/wifi`.
- usuario2 con el rol *rol2*, autorizado para publicar en los tópicos `sensor/humedad` y `sensor/presion`.

Posteriormente, en el archivo `broker.xml`, se configuran las políticas de seguridad correspondientes, asignando permisos de publicación únicamente a los roles autorizados para cada tópico. Una vez configurado el broker, en el código fuente de la ESP32 se realiza el proceso de autenticación mediante el ingreso del usuario y contraseña correspondientes, lo cual garantiza que cada microservicio pueda acceder exclusivamente a los tópicos definidos para su rol.

La **figura 4** presenta la configuración de roles y permisos definidos en el broker MQTT. En ella se observa la existencia de tres usuarios con distintos niveles de acceso. El usuario Admin tiene privilegios completos, lo que le permite gestionar el broker y realizar configuraciones generales. Por otro lado, el *usuario1*, asociado al *rol1*, tiene permisos exclusivos para publicar en los tópicos de wifi y temperatura. Finalmente, el *usuario2*, con *rol2*, solo tiene autorización para publicar en los tópicos de humedad y presión. Esta asignación diferenciada de permisos permite segmentar y controlar el uso de los tópicos, de acuerdo con la lógica y necesidades específicas del sistema.

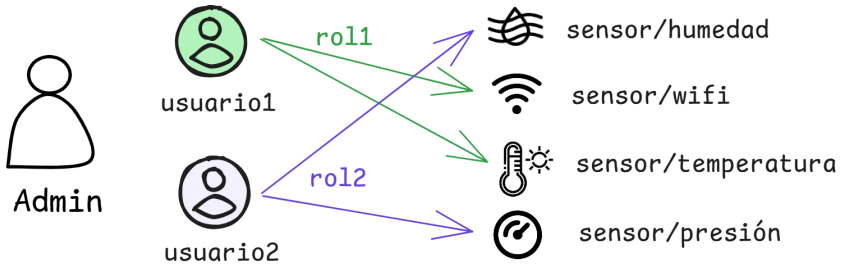


Fig. 4. Configuración de roles y permisos en el broker para el control de autenticación y publicación.

C. Validaciones de funcionamiento

Para evaluar el correcto desempeño del sistema propuesto, se realizaron diversas validaciones enfocadas tanto en el funcionamiento general de la arquitectura como en su comportamiento ante escenarios específicos.

En una primera fase, se validó la conectividad entre los distintos componentes del sistema: dispositivo productor (ESP32), broker MQTT y microservicios consumidores. Asimismo, se comprobó el almacenamiento adecuado de los datos en la base de datos InfluxDB, así como su correcta visualización en la plataforma Grafana. Posteriormente, se llevaron a cabo pruebas adicionales para analizar el comportamiento del sistema en condiciones particulares, tales como:

- Desconexión temporal del dispositivo productor.
- Integración de múltiples dispositivos productores (escalabilidad).
- Tolerancia del sistema ante un aumento en el tráfico de mensajes por unidad de tiempo.

5. RESULTADOS

A. Requerimiento del sistema:

Como resultado del proceso de planeación y delimitación del problema, se han identificado y definido los siguientes requerimientos que guiarán el desarrollo del sistema.

Requisitos funcionales:

- El microcontrolador debe generar y publicar datos cada 5 segundos a través del protocolo MQTT, enviándolos al broker (ActiveMQ Artemis).
- Los microservicios deben estar suscritos a los tópicos relevantes del broker y ser capaces de consumir los mensajes publicados por el productor.
- Una vez recibidos los datos, los microservicios deben almacenarlos en la base de datos temporal InfluxDB para su posterior análisis y visualización.
- Al menos uno de los microservicios debe exponer los datos almacenados en InfluxDB para su visualización mediante un dashboard en Grafana.

Requisitos no funcionales:

- El sistema debe ser modular, permitiendo la incorporación de nuevos microservicios sin necesidad de modificar los componentes existentes.
- El sistema debe ser tolerante a fallos, permitiendo la reconexión automática del productor (ESP32) en caso de pérdida de conectividad, garantizando así la continuidad del envío de datos cuando se restablezca la conexión.
- El proceso de autenticación del productor debe ser robusto, asegurando que únicamente dispositivos autorizados puedan publicar mensajes al broker de mensajería.
- Los microservicios deben ser capaces de operar de manera concurrente, procesando múltiples mensajes sin pérdida de rendimiento.

B. Funcionamiento del sistema:

Se logró establecer una arquitectura funcional para el sistema, en la que se integraron satisfactoriamente todos los componentes previstos: el productor de datos (ESP32), el broker de mensajería (ActiveMQ Artemis) y los microservicios consumidores. A continuación, se presentan evidencias que demuestran el correcto funcionamiento de cada etapa del sistema.

En primer lugar, se verificó que el broker MQTT se encontraba activo. Esta validación se realizó accediendo a la consola de administración web del broker mediante la URL `http://localhost:8161`, lo cual confirma que el servicio está disponible y en ejecución. La **figura 5** muestra la interfaz web del broker de mensajería. Se puede observar que el acceso se realiza a través del puerto 8161, y que la interfaz solicita un nombre de usuario y contraseña, los cuales fueron establecidos durante la creación inicial de la instancia. Posteriormente, se presenta una vista general de la consola de administración de ActiveMQ Artemis, en la que se destacan secciones como Artemis, JMX y Runtime. A través de estas secciones, es posible acceder a información detallada sobre los tópicos configurados, así como consultar documentación adicional sobre el funcionamiento del broker.

Posteriormente, el módulo ESP32 publica mensajes en los cuatro tópicos configurados: temperatura, humedad, presion y wifi. Para la visualización de estos mensajes se utiliza MQTT Explorer, una herramienta gráfica que permite observar en tiempo real los datos publicados, suscribirse a múltiples tópicos y visualizar el historial de mensajes recibidos. La **figura 6** muestra cómo la herramienta MQTT Explorer se suscribe correctamente a cada uno de los cuatro tópicos configurados, lo que confirma que la comunicación entre el productor (ESP32) y el broker MQTT se realiza de forma adecuada. Además, se puede observar una representación gráfica de los datos transmitidos, donde se visualiza el historial de cada tópico con su respectivo valor y marca de tiempo. Esta herramienta también permite navegar entre los diferentes tópicos publicados, lo que facilita el monitoreo y análisis del flujo de datos en tiempo real.

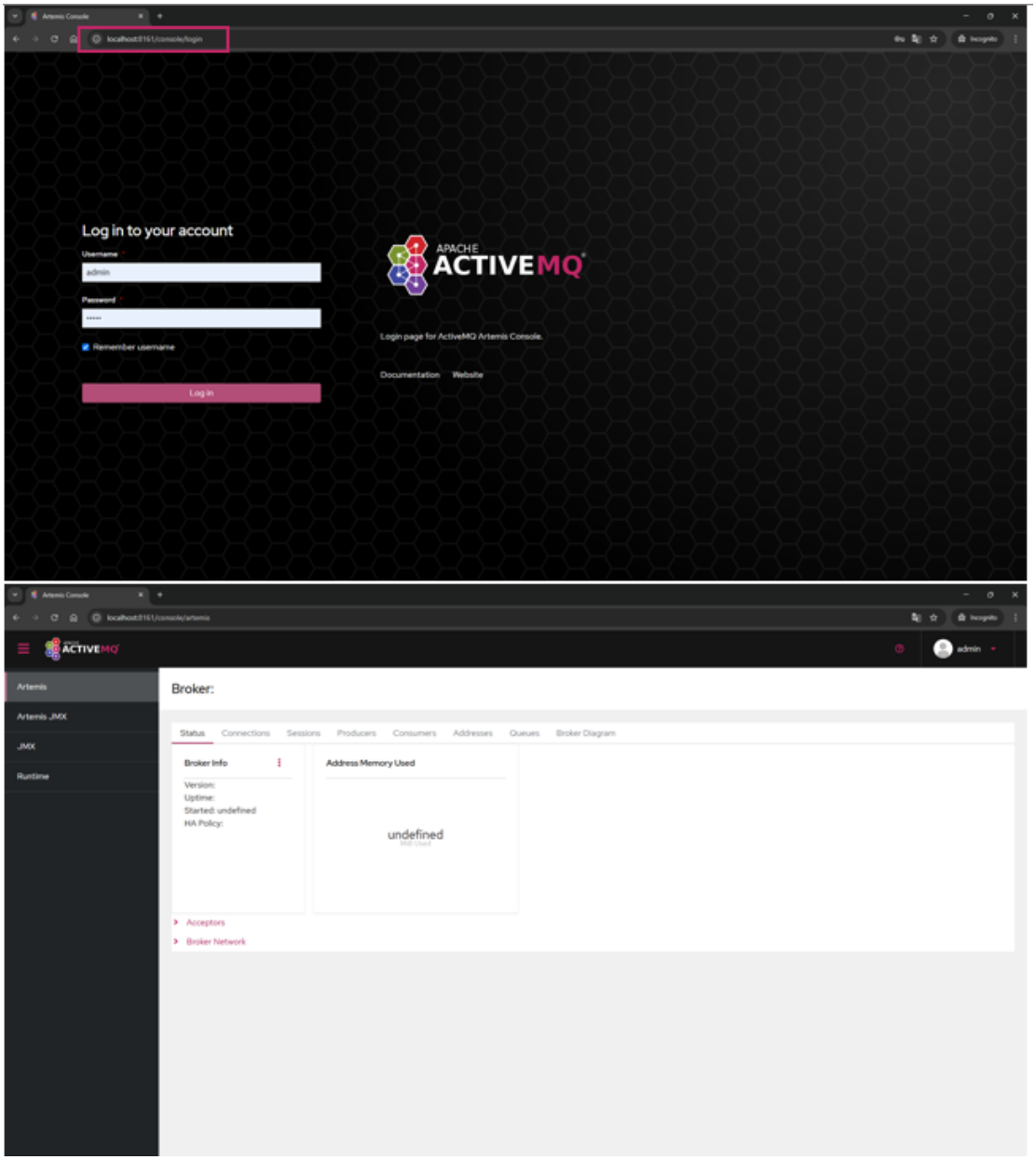


Fig. 5. Consola de gestión del broker MQTT vía navegador en el puerto 8161.

Una vez comprobado que los mensajes son enviados correctamente desde el módulo ESP32 al broker, se procede a su consumo mediante microservicios desarrollados en Python. Cada microservicio almacena la información en una base de datos InfluxDB independiente. Posteriormente, la herramienta Grafana consulta estas bases de datos para su visualización. La

Fig. 6. Visualización en MQTT Explorer de los mensajes publicados por el ESP32 en los cuatro tópicos definidos.

figura 7 muestra la ejecución de los microservicios, los cuales están desplegados como endpoints en una API construida con el framework **FastAPI**. Esta API se encuentra corriendo localmente en la dirección `http://127.0.0.1:8000`, donde se puede verificar que la aplicación está activa y funcionando correctamente. Asimismo, se observa cómo los microservicios reciben y procesan la información proveniente del broker de mensajería.

```
MATEO@ASUSTUFF15 MINGW64 /c/Users/MATEO/Desktop/mqtt/FastAPI
$ uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\MATEO\\Desktop\\mqtt\\FastAPI']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [27656] using StatReload
INFO:     Started server process [29040]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
[TWO] sensor/temperatura -> 17.1
[ONE] sensor/humedad -> 84.0
[TWO] sensor/wifi -> -36.0
[ONE] sensor/presion -> 1026.0
[TWO] sensor/temperatura -> 18.8
[TWO] sensor/wifi -> -37.0
[ONE] sensor/humedad -> 105.0
[ONE] sensor/presion -> 1025.0
```

Fig. 7. Ejecución local de los microservicios desarrollados con FastAPI.

Para verificar que la información llega a InfluxDB, la **figura 8** muestra cómo los datos correspondientes a los tópicos de temperatura y wifi, gestionados por el microservicio 1, son almacenados exitosamente en la base de datos. Esta visualización se realiza a través del Data Explorer, accesible mediante el puerto `8086`, donde los registros se presentan en formato tabular. Cada entrada incluye el valor del dato y su respectiva marca temporal. Este proceso de verificación se lleva a cabo mediante consultas (*queries*) que permiten acceder a los datos almacenados y comprobar el flujo entre el microservicio y la base de datos.

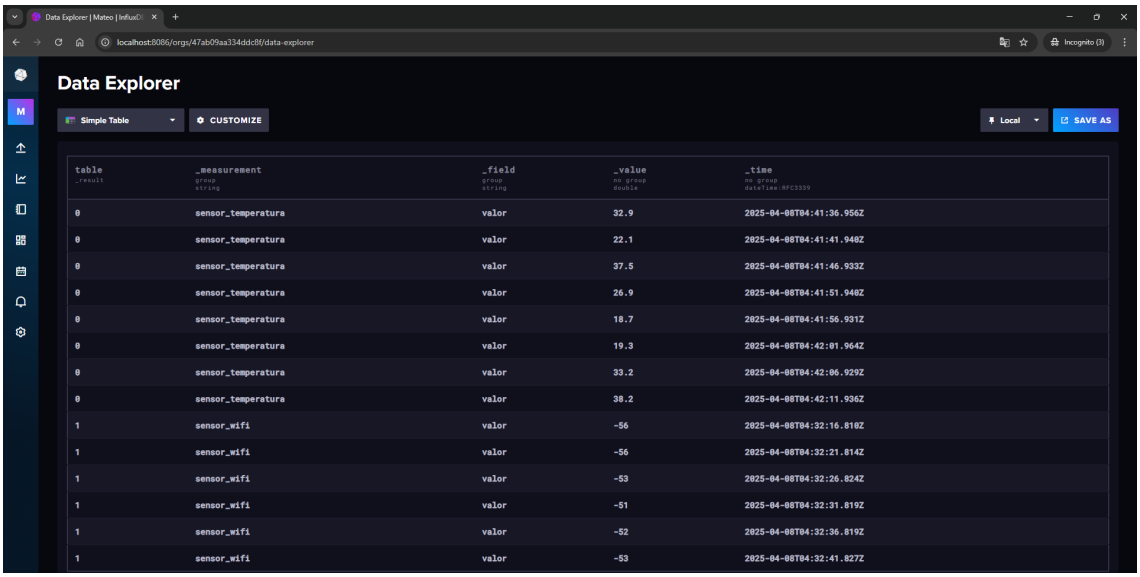


Fig. 8. Visualización de los datos de los tópicos de temperatura y WiFi almacenados en InfluxDB.

Por su parte, la **figura 9** presenta un dashboard de Grafana configurado para representar gráficamente estos mismos datos. En este panel se visualizan dos series temporales, una para la temperatura y otra para la intensidad de la señal WiFi. Además de dos indicadores que muestran el valor actual de cada parámetro. Las series temporales permiten observar la evolución de los datos a lo largo del tiempo, mientras que los indicadores proporcionan una lectura en tiempo real. Los datos se actualizan cada 5 segundos, que es el intervalo de envío configurado en el dispositivo productor.

Fig. 9. Visualización de los datos de los tópicos de temperatura y WiFi desde Grafana.

Para el microservicio 1 se implementó un proceso de filtrado de datos provenientes de los tópicos de humedad y presión. Los valores esperados se definieron dentro de los rangos de 55 % a 89 % para humedad y de 1015 hPa a 1036 hPa para presión. Sin embargo, durante la generación de los datos sintéticos se introdujo intencionalmente una variabilidad del 20 %, permitiendo que algunos valores se salieran de esos rangos con el objetivo de validar la capacidad del microservicio para identificar y descartar datos atípicos o erróneos.

La **figura 10** muestra los resultados del filtrado para el tópico de humedad. En dicha figura, los puntos azules representan los valores originales, mientras que los puntos rojos corresponden a los datos que han sido validados y mantenidos tras aplicar el filtrado. De forma similar, en la **figura 11**, los puntos amarillos representan los datos originales de presión, y los puntos verdes muestran los valores filtrados que cumplen con los criterios establecidos. Estas figuras permiten comprobar el correcto funcionamiento del microservicio, al evidenciar

que únicamente los datos dentro de los rangos definidos son procesados y almacenados.

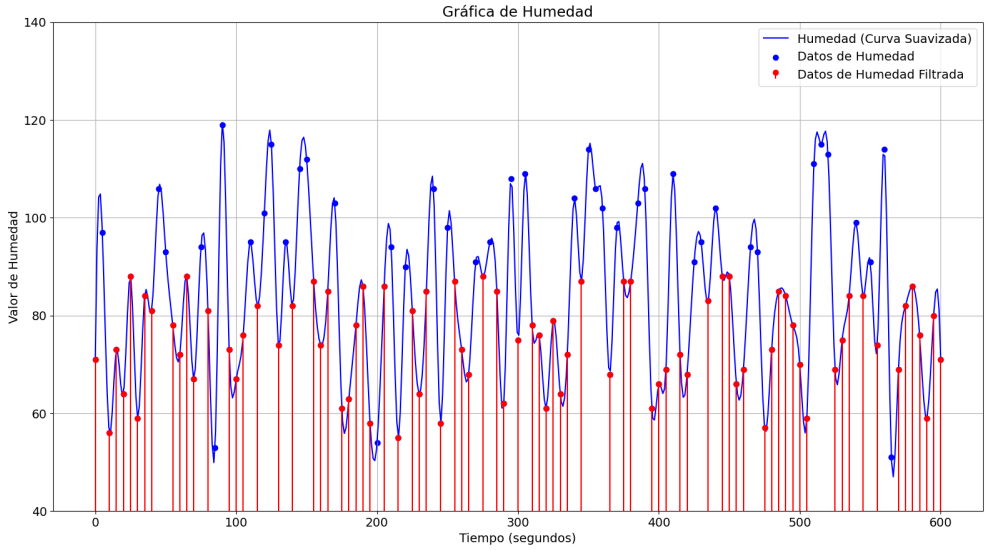


Fig. 10. Comparación de los datos de humedad durante un intervalo de 10 minutos, antes y después del proceso de filtrado.

Adicionalmente, en la **figura 12** se observa que los datos de presión y humedad, una vez filtrados, son almacenados correctamente en InfluxDB. La interfaz del Data Explorer, accesible a través del puerto `8086`, permite visualizar esta información en formato tabular. Cada registro contiene su respectivo valor y marca temporal. Esta visualización es posible tras el proceso de almacenamiento y posterior consulta de los datos mediante *queries*, lo que facilita la verificación y análisis de los datos dentro de la base.

C. Comportamiento del sistema ante condiciones variables

- **Desconexión del dispositivo productor:** Cuando el dispositivo productor de datos (ESP32) pierde la alimentación, sufre algún daño físico o presenta problemas de conectividad Wi-Fi, la transmisión de datos hacia el broker se interrumpe temporalmente. Sin embargo, los microservicios consumidores continúan en funcionamiento, aunque dejan de

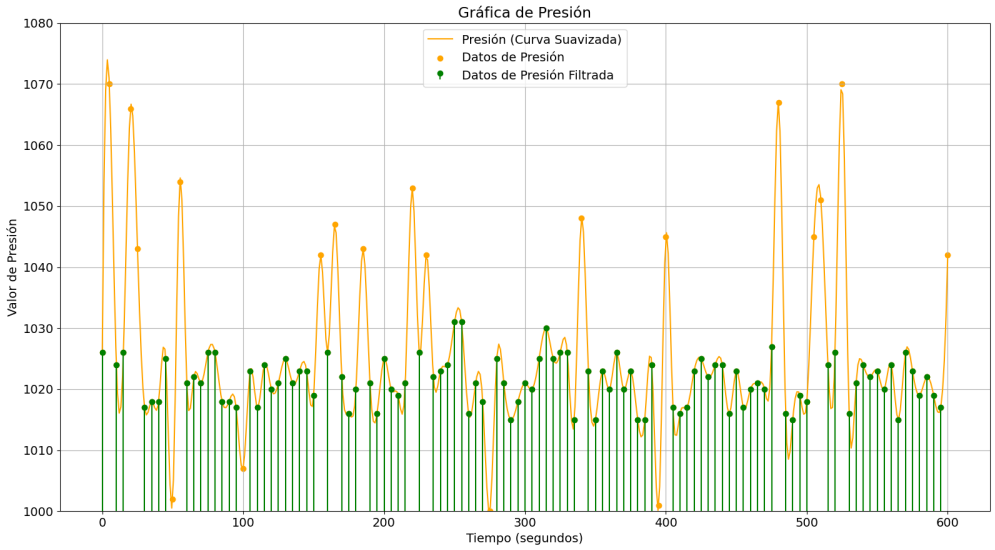


Fig. 11. Comparación de los datos de presión durante un intervalo de 10 minutos, antes y después del proceso de filtrado.

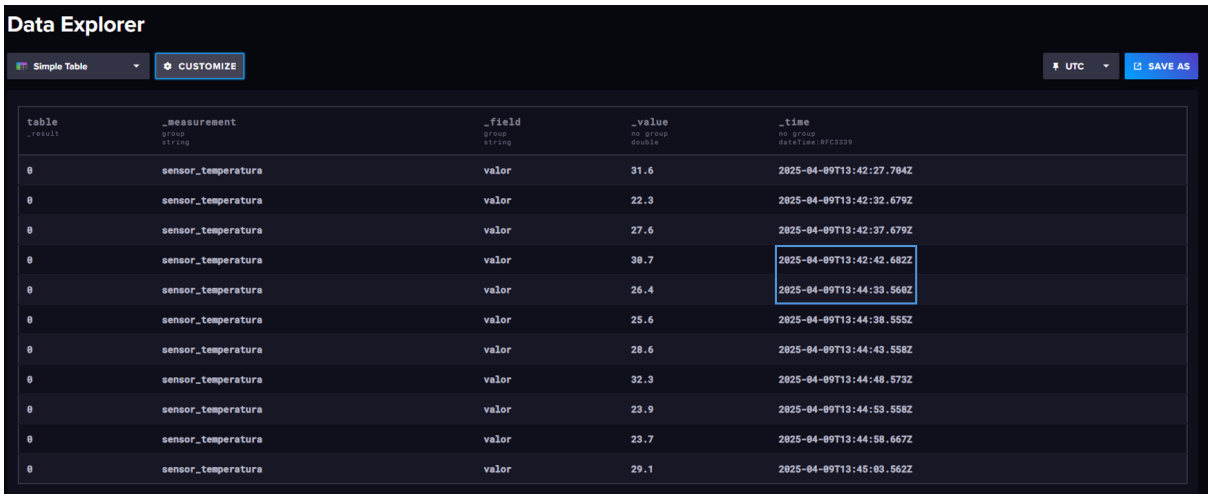


Fig. 12. Visualización de los datos de los tópicos de humedad y presión almacenados en InfluxDB

recibir información.

Como consecuencia, durante el periodo de desconexión no se almacenan registros en la

base de datos, lo que genera una discontinuidad en la serie temporal. Para validar este comportamiento, se realizó una prueba en la que se desconectó temporalmente la ESP32, retirando su alimentación por aproximadamente dos minutos. La **figura 13** muestra el resultado en la base de datos, donde se evidencia un salto temporal entre los registros: específicamente, el tiempo pasa de 42:42.68 a 44:33.56 sin ningún dato intermedio, lo que confirma la interrupción de la recepción.

A nivel gráfico, este comportamiento se representa en la **figura 14**, donde Grafana refleja visualmente el salto temporal. Como no se reciben datos en ese intervalo, la herramienta simplemente omite ese tramo, generando un vacío en la visualización. Una vez restablecida la alimentación de la ESP32, la transmisión de datos se reanuda y la visualización en Grafana vuelve a la normalidad.

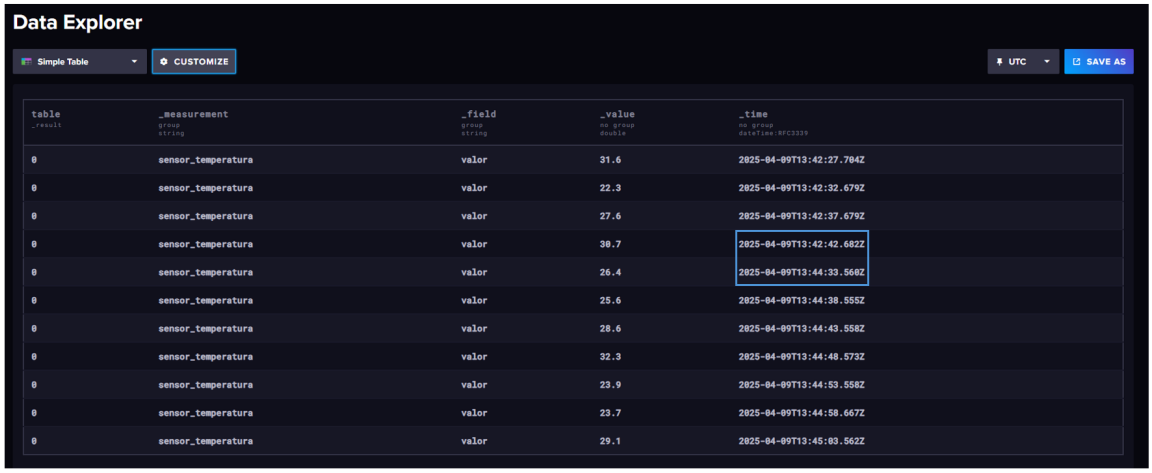


table	_measurement	_field	_value	_time
0	sensor_temperatura	valor	31.6	2025-04-09T13:42:27.704Z
0	sensor_temperatura	valor	22.3	2025-04-09T13:42:32.679Z
0	sensor_temperatura	valor	27.6	2025-04-09T13:42:37.679Z
0	sensor_temperatura	valor	30.7	2025-04-09T13:42:42.682Z
0	sensor_temperatura	valor	26.4	2025-04-09T13:44:33.560Z
0	sensor_temperatura	valor	25.6	2025-04-09T13:44:38.555Z
0	sensor_temperatura	valor	28.6	2025-04-09T13:44:43.558Z
0	sensor_temperatura	valor	32.3	2025-04-09T13:44:48.573Z
0	sensor_temperatura	valor	23.9	2025-04-09T13:44:53.558Z
0	sensor_temperatura	valor	23.7	2025-04-09T13:44:58.667Z
0	sensor_temperatura	valor	29.1	2025-04-09T13:45:03.562Z

Fig. 13. Efecto de la desconexión del dispositivo en el almacenamiento de datos.

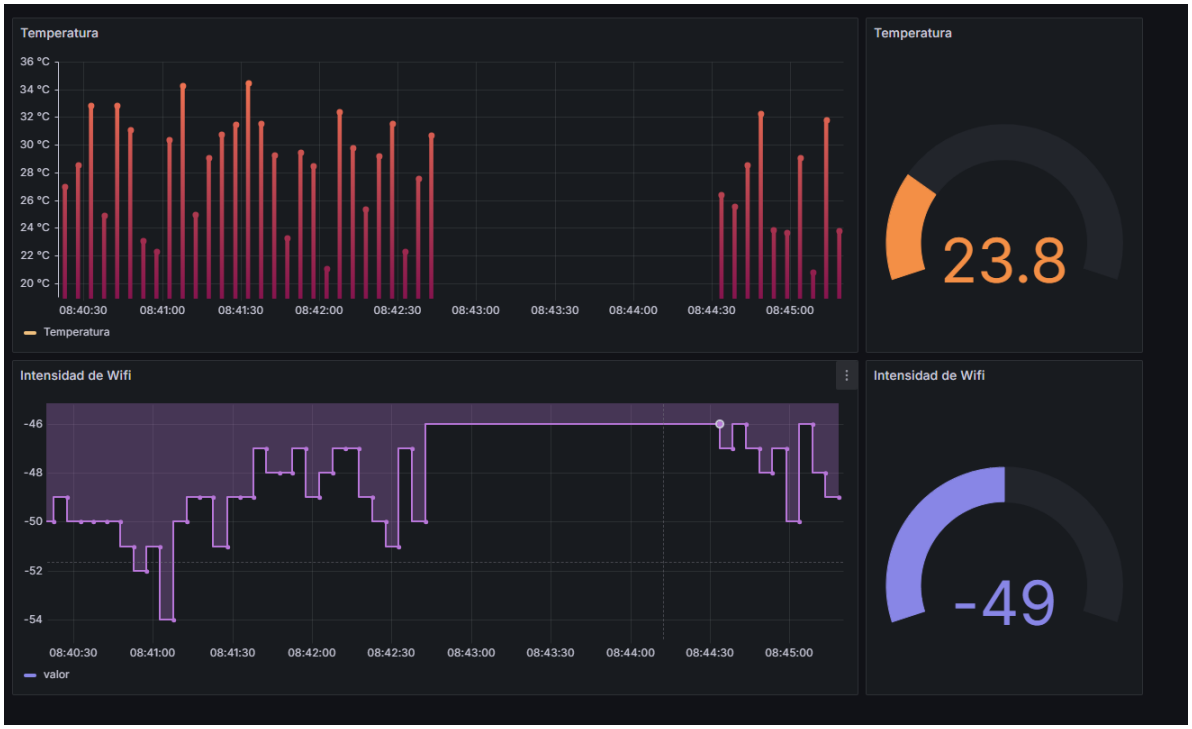


Fig. 14. Efecto de la desconexión del dispositivo en la visualización de datos.

- **Integración de dos productores:** La conexión de múltiples productores, como dos dispositivos ESP32, puede realizarse sin necesidad de modificar la arquitectura del sistema. Basta con que cada MCU publique en el tópico correspondiente utilizando un identificador de cliente distinto, lo cual evita conflictos en la comunicación. El broker MQTT se encarga de recibir los mensajes de todos los dispositivos sin inconvenientes y de enrutar correctamente la información a los consumidores.

  La **figura 15** muestra el resultado de la transmisión simultánea desde dos ESP32 que publican datos de temperatura. Para esta prueba, se simularon ubicaciones distintas: la ESP32 A genera datos en un rango de 15°C a 25°C, mientras que la ESP32 B emite valores entre 30°C y 35°C. En la gráfica se puede observar que ambos conjuntos de datos se visualizan correctamente, sin conflictos, lo que demuestra que ni el broker ni los microservicios presentan inconvenientes ante la incorporación de múltiples productores. Los puntos de

temperatura aparecen generalmente intercalados, ya que ambas ESP32 tienen el mismo intervalo de envío de datos. Sin embargo, en algunos casos pueden visualizarse secuencias consecutivas de una misma ESP32, lo cual se debe a ligeras diferencias en el tiempo de llegada de los mensajes al broker, causadas por variaciones en la red o procesamiento interno.

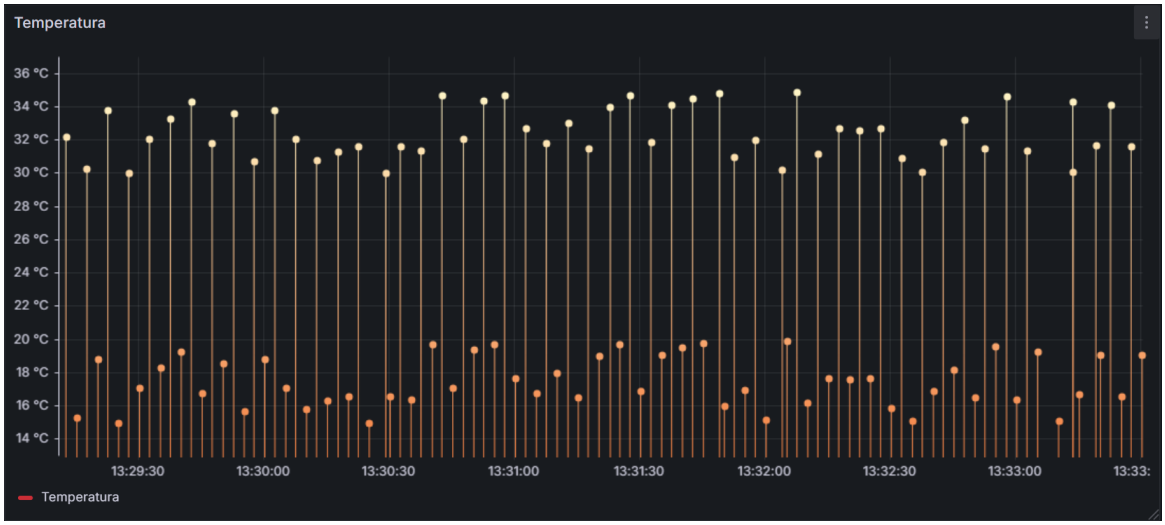


Fig. 15. Transmisión simultánea de datos desde dos dispositivos ESP32.

- **Tolerancia al aumento del tráfico de mensajes por tiempo:** Se realizó una prueba para evaluar el comportamiento del sistema ante un incremento en la frecuencia de envío de mensajes. Inicialmente, los mensajes se enviaban cada 5 segundos, y posteriormente se redujo el intervalo a 0.5 segundos por mensaje. Como se muestra en la **figura 17** Aunque el sistema mostró un desempeño favorable en términos de recepción, se observaron momentos puntuales donde la integridad temporal se vio afectada. Específicamente, algunos mensajes presentaron retrasos y fueron recibidos muy próximos al siguiente, lo que indica una pérdida de uniformidad en la temporización.

  No obstante, durante una prueba continua de 50 segundos, se enviaron un total de 100 mensajes y todos fueron correctamente recibidos, sin pérdida alguna. Esto demuestra que,

a pesar de ciertos retrasos, el sistema mantiene una alta fiabilidad en la entrega de mensajes incluso bajo condiciones de mayor carga.

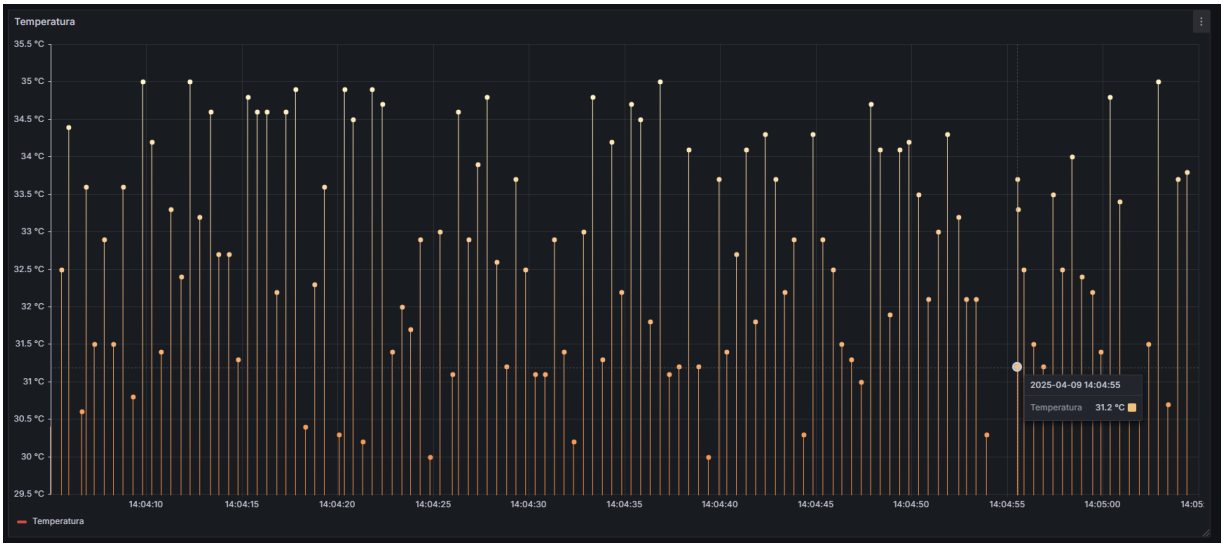


Fig. 16. Transmisión de mensajes cada 0,5 segundos.

6. CONCLUSIONES Y RECOMENDACIONES

- El sistema opera de manera efectiva bajo las condiciones establecidas, principalmente su modularidad, seguridad y capacidad de conexión fiable. Además, se ha demostrado que es capaz de manejar algunos fallos, como la desconexión temporal de alimentación o la pérdida de conexión de la red.
- El sistema es fácilmente escalable tanto en términos de productores como de consumidores de datos. Esto se debe a que varias ESP32 pueden comunicarse entre sí sin inconvenientes gracias al protocolo MQTT. Además, el diseño independiente de microservicios permite que un microservicio pueda estar en mantenimiento o fallar sin necesidad de afectar el funcionamiento de los demás.
- No se encontraron problemas en cuanto a la disponibilidad de los datos para su almacenamiento y visualización. Los sistemas de visualización demostraron ser efectivos para monitorear los datos.
- Se identificaron ciertos retrasos en los mensajes cuando el intervalo de tiempo entre mensajes disminuye. Estos retrasos ocasionalmente generan problemas como retrasos momentáneos.

Recomendaciones:

- Mejorar la seguridad del sistema: Para ambientes más inseguros, se recomienda implementar un sistema de autenticación basado en tokens de verificación en lugar de depender únicamente de un usuario y contraseña.
- Garantizar la disponibilidad de los datos de los productores: En caso de desconexiones prolongadas de la red, se sugiere implementar una técnica de almacenamiento temporal que permita guardar los datos del productor mientras esté fuera de la red. Para ello, se podría considerar el uso de almacenamiento en memoria no volátil, como EEPROM o tarjetas SD
- Monitoreo permanente del sistema: Para garantizar la disponibilidad continua del sistema y revisar su estado de manera eficiente, se recomienda utilizar herramientas de monitoreo, como Loghstash y Elastic searhc para la recolección de métricas y alertas.

Departamento de Ingeniería Electrónica y de Telecomunicaciones

Desarrollo de un Sistema de Microservicios Orientado a Eventos con Message Broker para un Proveedor de Identidad (IdP) usando protocolo MQTT

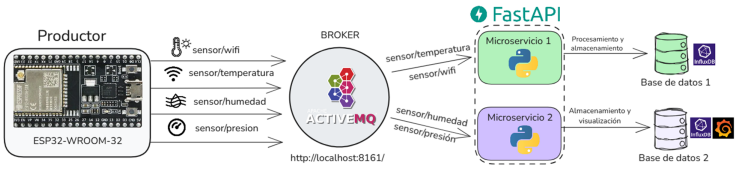
UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

PRACTICANTE: Mateo Alejandro Bravo Revelo
ASESORES: Hernan Felipe Garcia Arias, Nicolas Posada Chaparro

PROGRAMA: Ingeniería Electrónica
Semestre de la práctica: 2024-2

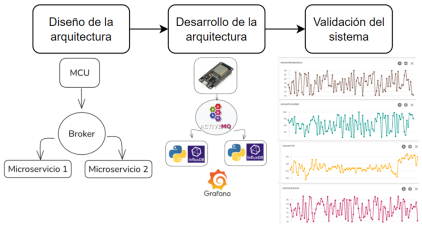
Introducción

Intertelco S.A.S. desarrolla un Proveedor de Identidad basado en arquitectura de microservicios, reemplazando sistemas monolíticos para mejorar la escalabilidad y el mantenimiento. Un ESP32 actúa como productor de datos, comunicándose con un broker MQTT en un servidor local. Dos microservicios consumen, procesan y almacenan los datos, utilizando programación asíncrona para garantizar alto rendimiento y eficiencia en los procesos de autenticación.



Metodología

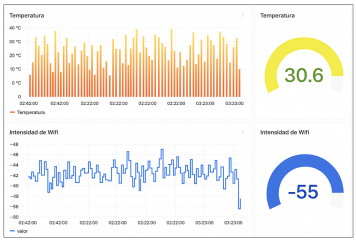
El desarrollo del proyecto se organizó en tres etapas. La primera consistió en el diseño de la arquitectura y la definición de los requerimientos. La segunda etapa abarcó la elección de tecnologías y la implementación de la conexión entre los componentes del sistema. Finalmente, en la tercera etapa se realizaron pruebas para validar el funcionamiento y la eficacia del sistema.



Resultados

Se logró establecer una arquitectura funcional para el sistema, integrando satisfactoriamente todos los componentes previstos: el productor de datos (ESP32), el broker de mensajería (ActiveMQ Artemis) y los microservicios consumidores. Como resultado, fue posible visualizar correctamente la información transmitida desde el productor, tanto en el dashboard como en la base de datos, validando la efectividad del flujo de datos en todo el sistema.

Grafana



InfluxDB

_measurement	_field	_value
sensor_temperatura	valor	25.4882337
sensor_temperatura	valor	25.4317394
sensor_temperatura	valor	25.5101529
sensor_temperatura	valor	24.1269320
sensor_temperatura	valor	24.7695198
sensor_temperatura	valor	24.9416666
sensor_temperatura	valor	24.9328166

Más información sobre el proyecto

Objetivos

- Diseñar la arquitectura del sistema IoT, definiendo el proveedor de identidad (IdP) para gestionar el control de acceso del microcontrolador, así como la configuración del Message Broker y la funcionalidad de los microservicios, especificando el rol del consumidor en el proceso.
- Implementar un proveedor de identidad (IdP) para la gestión de la autenticación y validación del microcontrolador en el sistema IoT, incluyendo la configuración del Message Broker basado en MQTT para la publicación de información, el desarrollo del microcontrolador como productor de datos y la creación de microservicios para el almacenamiento y procesamiento de la información.
- Validar el funcionamiento de la arquitectura implementada mediante pruebas, asegurando la correcta autenticación, la comunicación entre el productor y los microservicios, y la correcta ejecución del procesamiento y almacenamiento de datos por parte de los microservicios.

Conclusiones

- El sistema funciona correctamente bajo las condiciones establecidas, destacando su modularidad, seguridad y capacidad de conexión fiable. Además, maneja fallos como desconexiones de alimentación o de red.
- Es fácilmente escalable, permitiendo la comunicación entre múltiples ESP32 gracias al protocolo MQTT. El diseño modular de microservicios garantiza que el fallo de uno no afecta a los demás.
- No se presentaron problemas con la disponibilidad de datos para almacenamiento y tampoco para los sistemas de visualización.
- Se observaron retrasos en los mensajes cuando el intervalo entre ellos disminuye, lo que ocasionalmente causa retrasos momentáneos.

DATOS DE CONTACTO DEL AUTOR:
3188608403 mateo.bravor@udea.edu.co https://www.linkedin.com/in/mateobravo-r/

Fig. 17. Poster del proyecto.

REFERENCIAS

[1] Inter-Telco. (2025) Inter-telco inicio. Accedido: mayo 2025. [Online]. Available: https://inter-telco.com/nosotros/

[2] Apache Software Foundation. (2024) Flexible powerful open source multi-protocol messaging. Consultado el 15 de febrero de 2025. [Online]. Available: https://activemq.apache.org/

[3] IBM Corporation. (2024) ¿qué es un intermediario de mensajes? Consultado el 15 de febrero de 2025. [Online]. Available: https://www.ibm.com/mx-es/topics/message-brokers

[4] Amazon Web Services, Inc. (2024) ¿qué es mqtt? Consultado el 15 de febrero de 2025. [Online]. Available: https://aws.amazon.com/es/what-is/mqtt/

[5] Amazon Web Services. (2024) ¿qué son los microservicios? Consultado el 5 de enero de 2025. [Online]. Available: https://aws.amazon.com/es/microservices/

[6] R. M. Munaf, J. Ahmed, F. Khakwani, and T. Rana, "Microservices Architecture: Challenges and Proposed Conceptual Design," in *2019 International Conference on Communication Technologies (ComTech)*, 2019, pp. 82–87.

[7] Intel Corporation. (2024) Microservicios y arquitectura de microservicios. Consultado el 15 de febrero de 2025. [Online]. Available: https://www.intel.la/content/www/xl/es/cloud-computing/microservices.html

[8] OpenWebinars. (2023) ¿qué es influxdb y primeros pasos. Consultado el 25 de mayo de 2025. [Online]. Available: https://openwebinars.net/blog/que-es-influxdb-y-primeros-pasos/

[9] ——. (2023) ¿qué es grafana y primeros pasos. Consultado el 25 de mayo de 2025. [Online]. Available: https://openwebinars.net/blog/que-es-grafana-y-primeros-pasos/

[10] M. DÁloia, A. Longo, G. Guadagno, M. Pulpito, P. Fornarelli, P. N. Laera, D. Manni, and M. Rizzi, "Low Cost IoT Sensor System for Real-time Remote Monitoring," in *2020 IEEE International Workshop on Metrology for Industry 4.0 IoT*, 2020, pp. 576–580.

[11] Cloudflare, Inc. (2024) ¿qué es un proveedor de identidad (idp)? Consultado el 25 de mayo de 2025. [Online]. Available: https://www.cloudflare.com/es-es/learning/access-management/what-is-an-identity-provider/

[12] S. Nimkar. (2023) Building microservices with fastapi and rabbitmq - part 1. Consultado el 8 de febrero de 2025. [Online]. Available: https://snimkar1905.medium.com/building-microservices-with-fastapi-and-rabbitmq-part1-1104dbd4ad96