CPES SAN LUIS LH BHIP

Ciclo Formativo Grado Superior en Administración de Sistemas Informáticos en Red

Cloud & IoT

Curso lectivo:	2026
Especialidad:	2º curso de ASIR
Módulo:	Planificación y Administración de computación en la nube
Reto Ethazi:	Reto 5. Gran premio de Fórmula 1
Producto:	Producto 6. Cloud & IoT. Proyecto IoT
Realizado por:	GRUPO 3: Christian Manzambi, Mikel Gil, Izaro Ruiz, Jaime Iribarnegaray
Profesor:	Juan Carlos San Román
Fecha:	07/01/2025

Resumen

El presente proyecto tiene por objetivo el desarrollo de un sistema de monitorización de las condiciones de temperatura y humedad mediante la creación de una arquitectura de IoT que establece una conexión entre los dispositivos y las soluciones en la nube y locales.

Para ello, se hace uso de un microcontrolador ESP32 simulado en *Wokwi* que, a su vez, es programado en *MicroPython* para realizar la captura de la información del sensor DHT22 para, posteriormente, enviarla en formato *JSON* utilizando el protocolo de comunicaciones MQTT.

La gestión de la información se distribuye en dos rutas: una visualización indirecta utilizando ThingSpeak en el *cloud* y un *stack* local gestionado con Docker Compose. En el entorno local, Node-RED se encarga de procesar y enrutar los mensajes hacia la base de datos de series temporales InfluxDB. Finalmente, Grafana se encargará de realizar la representación gráfica avanzada y del sistema de alertas en tiempo real ante condiciones críticas. El sistema asegura la persistencia de la información y la supervisión de las métricas ambientales.

Índice de contenidos

Índice de figuras

1. Introducción

1.1. Justificación y planteamiento del problema

A continuación, la Introducción y los Objetivos de tu informe, escrito de manera técnica y profesional siguiendo el esquema del Grupo 2 y los requerimientos solicitados:

A día de hoy, realizar un seguimiento ambiental en espacios industriales y de domótica precisa de sistemas que sean capaces de llevar a cabo una gestión de datos que sea eficiente, escalable y en tiempo real. Usar cualquier tipo de solución IoT (Internet of Things) permite captar variables críticas como las relacionadas con la temperatura o la humedad para evitar fallos en infraestructuras o mejorar la sensación de confort.

Este proyecto toma impulso de la necesidad de integrar a dispositivos en infraestructuras de procesamiento. Para llevar a cabo esta integración el planteamiento está basado en el uso de tecnologías Open Source o tecnologías de contenedores que den lugar a la creación de un ecosistema robusto, sin anclarse a una única plataforma, asegurar la persistencia de la información y asegurar la tecnología ante alertas.

1.2. Objetivos

1.2.1. Objetivo general

Diseñar, programar y desplegar la arquitectura IoT completa para la monitorización distribuida de sensores ambientales validando el flujo de datos que va desde la captura en el microcontrolador hasta la visualización y almacenamiento del dato de forma profesional.

1.2.2. Objetivos específicos

Diseñar, programar y desplegar la arquitectura IoT completa para la monitorización distribuida de sensores ambientales validando el flujo de datos que va desde la captura en el microcontrolador hasta la visualización y almacenamiento del dato de forma profesional.

Configurar y validar el despliegue del protocolo MQTT como el estándar de transporte ligero entre el dispositivo y los servidores. Integración Cloud: Implementar una rama que despliegue la visualización rápida en la plataforma ThingSpeak para acceso remoto externo. Infraestructura Local: Orquestar, mediante Docker Compose, un stack tecnológico formado por Node-RED, InfluxDB y Grafana. Gestión de Datos y Alertas: Configurar flujos de procesamiento en Node-RED, almacenamiento en bases de datos de series temporales y un sistema visual de alertas en Grafana para la detección de anomalías térmicas.

1.3 Contextualización

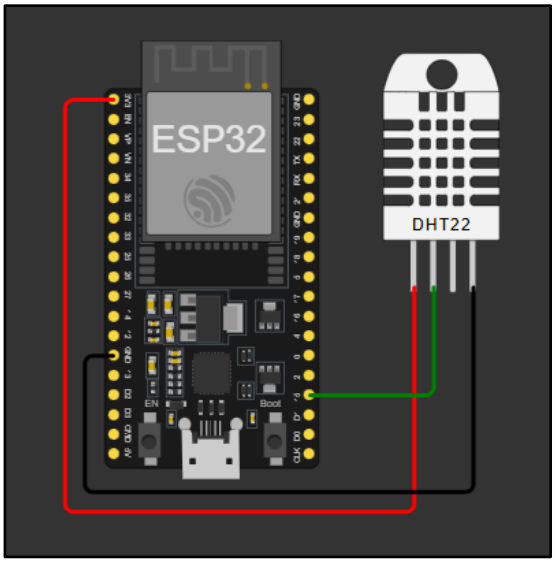
La correspondiente actuación gira a través del módulo "Planificación y Administración de Computación en la Nube" del ciclo ASIR y en la realización del proyecto “Gran Premio Fórmula 1”.

Utiliza herramientas de simulación tipo Wokwi para el hardware e instancias de virtualización para el servidor local, representando un escenario real de un despliegue industrial donde la redundancia de datos (Cloud y Local) y la manera de interaccionar con los recursos son sus basamentos fundamentales dentro de la administración de sistemas moderna.

2. Apartado técnico

2.1. Simulación y Firmware (Wokwi)

El proceso de captura de datos es ejecutado en la capa "Edge" con un microcontrolador ESP32 en el simulador Wokwi. La lectura de temperatura y humedad se recoge mediante un sensor DHT22.



Fuente: Figura 1 información recogida por el grupo 3

2.1.1 Configuración de hardware y sensor DHT22

El sensor DHT22 está conectado al pin GPIO 15 del ESP32. La alimentación, en el simulador, se realiza a 3.3V, que es el voltaje de alimentación conformado en el microcontrolador.

2.1.2. Código MicroPython y lógica de envío JSON

El firmware se ha programado en MicroPython. El flujo de la lógica principal es el siguiente: Conexión a la red WiFi virtual (Wokwi-GUEST). Inicialización del cliente MQTT apuntando al broker test.mosquitto.org.

Se obtienen los datos del sensor y se empaquetan en formato JSON. Y la publicación del mensaje en el tópico configurado (*asir/grupo3/sensores*).

Se puede ver el código completo en el apartado de Anexos.

```
# CONFIGURACIÓN DE REQUISITOS
MQTT_CLIENT_ID = "esp32-asir-cliente"
MQTT_BROKER    = "test.mosquitto.org"
MQTT_TOPIC     = "asir/grupo3/sensores"
```

Fuente: Figura 2 información recogida por el grupo 3

La salida de del sensor debe especificar tanto la temperatura como la humedad:

```
Conectado a Mosquitto
Midiendo...Enviando a asir/grupo3/sensores: {"temperatura": 24.0, "humedad": 40.0, "timestamp": 3}
Midiendo...Enviando a asir/grupo3/sensores: {"temperatura": 24.0, "humedad": 40.0, "timestamp": 13}
Midiendo...Enviando a asir/grupo3/sensores: {"temperatura": 79.8, "humedad": 55.5, "timestamp": 23}
Midiendo...Enviando a asir/grupo3/sensores: {"temperatura": 9.7, "humedad": 55.5, "timestamp": 34}
Midiendo...Enviando a asir/grupo3/sensores: {"temperatura": 11.8, "humedad": 71.5, "timestamp": 44}
Midiendo...Enviando a asir/grupo3/sensores: {"temperatura": 33.1, "humedad": 40.5, "timestamp": 54}
```

Fuente: Figura 3 información recogida por el grupo 3

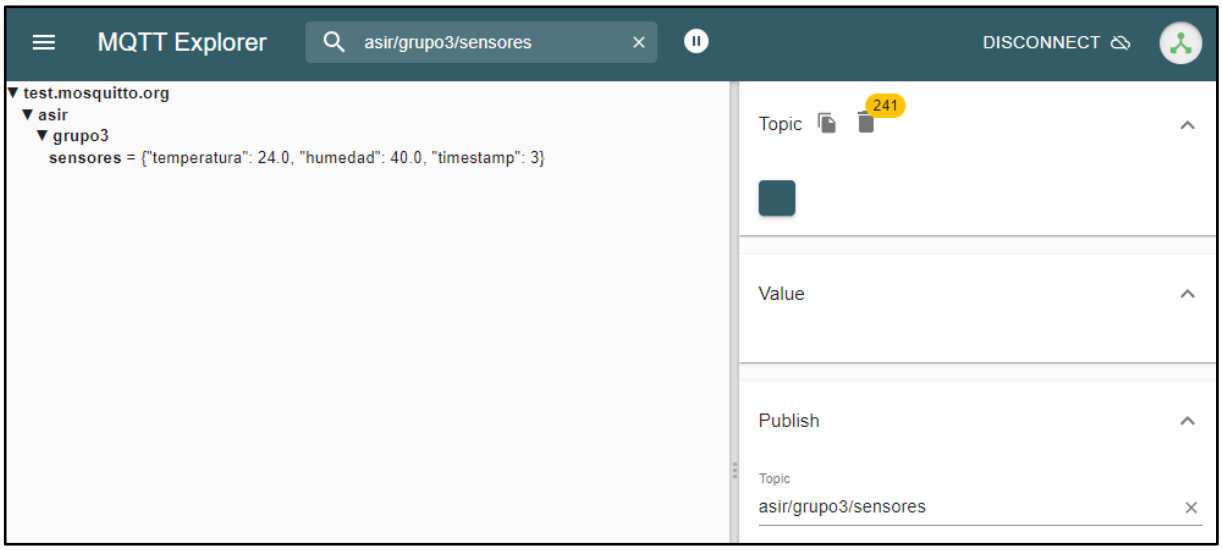
2.2. Zona de Transporte: Protocolo MQTT

1.2.3. Validación de mensajería con MQTT Explorer

Se hace uso de MQTT (Message Queuing Telemetry Transport) para la comunicación entre el dispositivo y el resto del sistema. Este protocolo es ideal para IoT por su ligereza y consumo de bajo ancho de banda. Se usa un modelo de Publicación/Suscripción.

La llegada de los mensajes se ha validado usando la herramienta MQTT Explorer, asegurando que el JSON llegue íntegro al broker.

Para que reconozca directamente el tópico el broker de MQTT que es test.mosquitto.org. De manera que se verá la salida de los datos recopilados por Wokwi

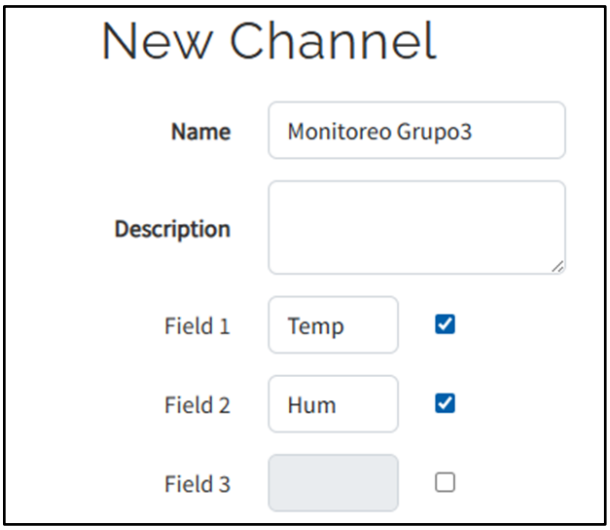


Fuente: Figura 4 información recogida por el grupo 3

2.3. Zona Cloud: ThingSpeak

El ThingSpeak, es una herramienta que permite recoger los datos creados y hacer una representación gráfica sincronizada a la BD.

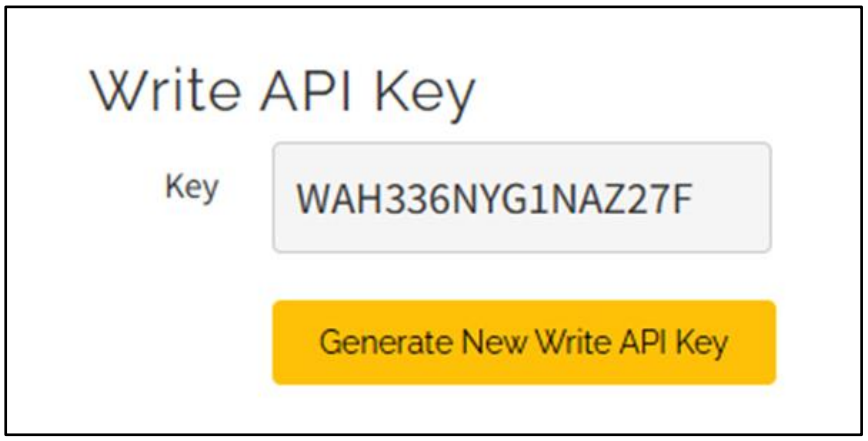
Para empezar, se deberá de crear una cuenta e iniciar sesión en ella. Una vez iniciada la cuenta, se tendrá que crear un “New Channel” y poner las características del proyecto.



Fuente: Figura 5 información recogida por el grupo 3

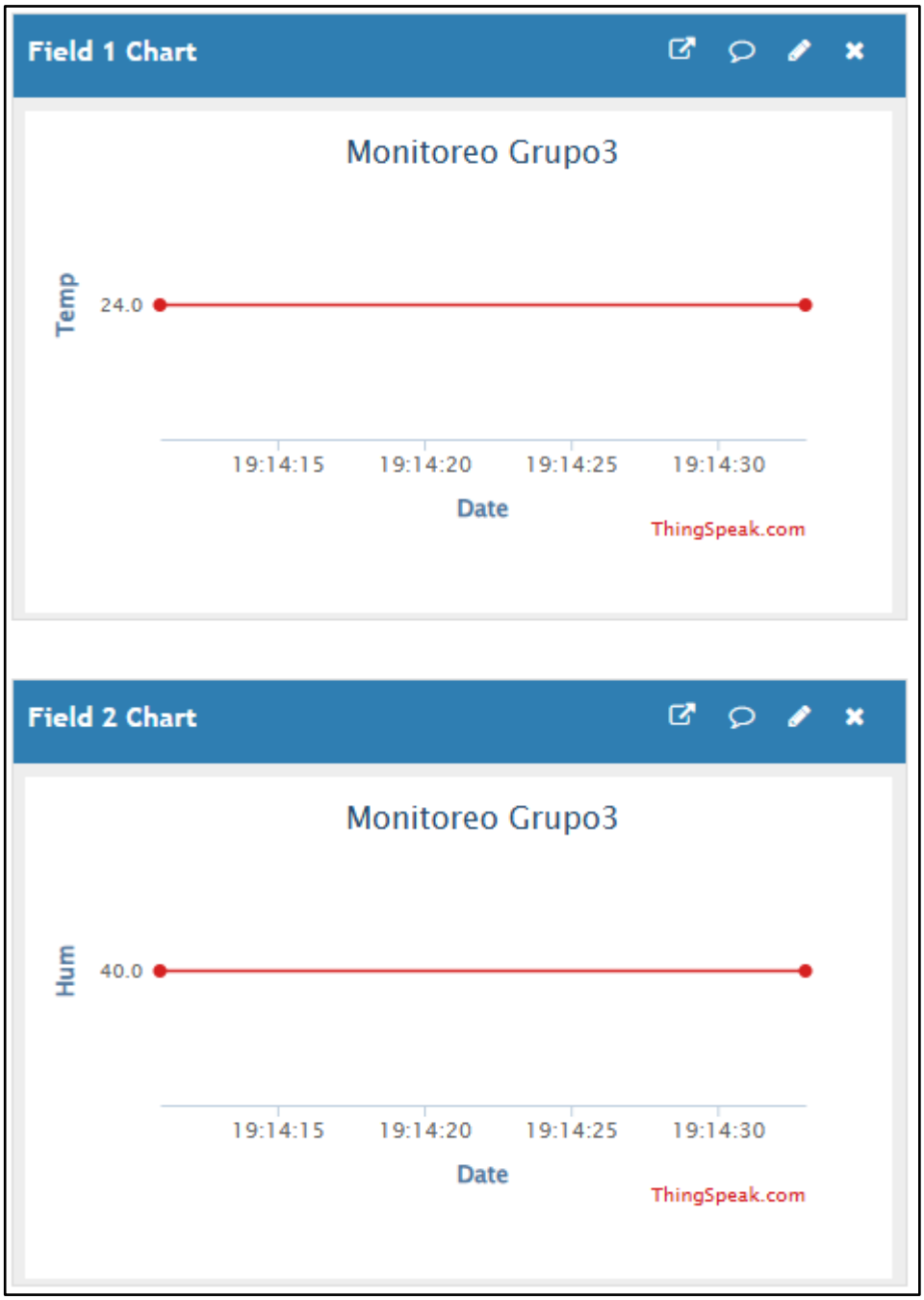
Una vez creado el *channel*, se deberá de añadir un *device* para crear las credenciales del cliente.

Es importante descargarse las credenciales y usarlas para nunca perderlas, para eso, hay que pinchar en el botón *Download Credentials*.



Fuente: Figura 6 información recogida por el grupo 3

Una vez terminado todo, se tendrá que refrescar la página de *ThingSpeak* para que vayan cargando los datos recogidos, de esa manera, se podrá comprobar mediante una representación gráfica los datos en vivo.



Fuente: Figura 7 información recogida por el grupo 3

2.4. Zona Local: Infraestructura (Docker Stack)

Como primera salida de datos se ha implementado ThingSpeak, que permite tener en la nube una visualización inmediata sin una infraestructura propia.

Se han configurado dos canales (Fields) para representar en tiempo real la temperatura y la humedad.

La parte más fuerte del proyecto está en el servidor local, donde se despliega un stack de

servicios mediante Docker Compose.

2.4.1. Orquestación con Docker Compose

En el archivo docker-compose.yml se definen tres servicios de importancia:

- Node-RED, InfluxDB y Grafana.

Primero para nodered se utiliza el puerto 1880 para la visualización en el navegador para realizar el diagrama de flujo.

```yaml
# Procesar y gestionar los datos
nodered:
  image: nodered/node-red:latest
  container_name: nodered
  restart: unless-stopped
  ports:
    - "1880:1880"
  volumes:
    - ./data/nodered:/data
  networks:
    - iot_network
```

Fuente: Figura 8 información recogida por el grupo 3

En el docker compose también se definirán los datos de inicio de sesión.

```yaml
# Almacenar el histórico de datos
influxdb:
  image: influxdb:1.8
  container_name: influxdb
  restart: unless-stopped
  ports:
    - "8086:8086"
  environment:
    - INFLUXDB_DB=iot
    - INFLUXDB_HTTP_AUTH_ENABLED=true
    - INFLUXDB_ADMIN_USER=admin
    - INFLUXDB_ADMIN_PASSWORD=sanluis
    - INFLUXDB_USER=iotuser
    - INFLUXDB_USER_PASSWORD=iot123
  volumes:
    - ./data/influxdb:/var/lib/influxdb
  networks:
    - iot_network
```

Fuente: Figura 9 información recogida por el grupo 3

Se utiliza grafana para la visualización de métricas que recoja tanto el Woki como la dependencia de Influxdb.

```
  # Visualización y alertar
  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    restart: unless-stopped
    ports:
      - "3000:3000"
    volumes:
      - ./data/grafana:/var/lib/grafana
    networks:
      - iot_network
    depends_on:
      - influxdb

networks:
  iot_network:
```

Fuente: Figura 10 información recogida por el grupo 3

2.4.2. Persistencia de datos y volúmenes

Persistencia y redes: se han configurado volúmenes para mantener los datos de la base de datos y los flujos de Node-RED para que no se pierdan durante el reinicio de los contenedores. Asimismo, todos los contenedores participan de una red virtual para comunicarse entre sí por nombre de servicio.

2.5. Procesamiento de datos: Node-RED

Node-red es el que intercepta los datos del MQTT y los redirige al InfluxDB. Por eso, hay que entrar al servicio de node-red con el puerto 1880 y crear un diagrama con los datos necesarios para el dashboard.

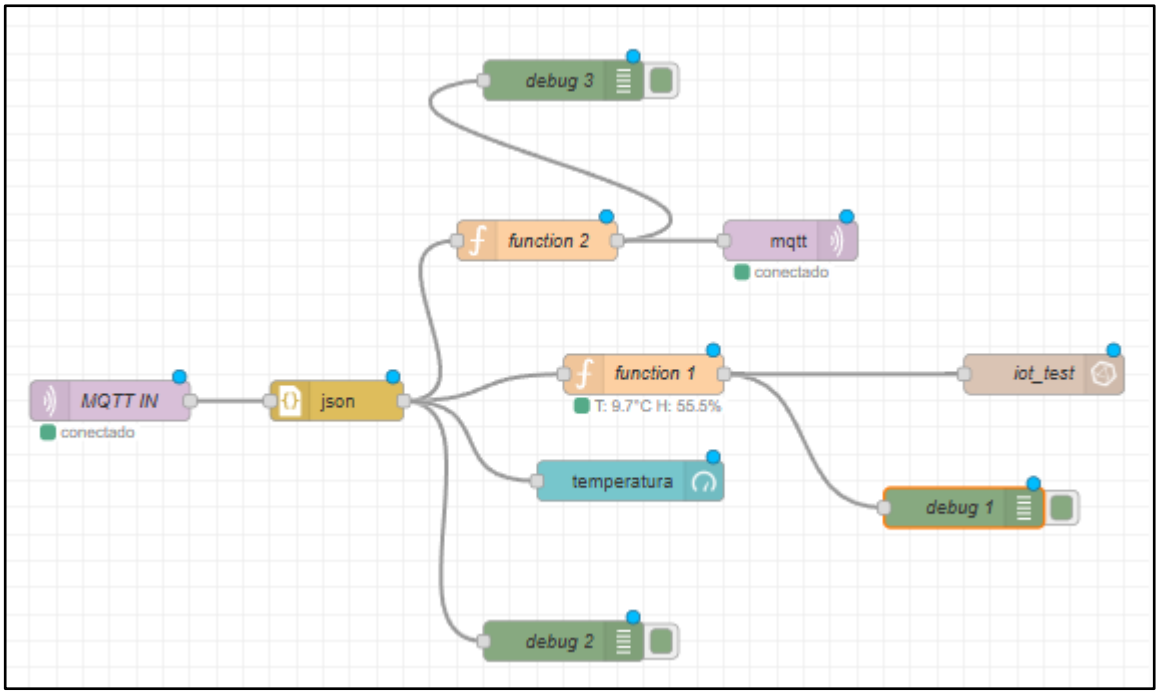
Para hacer el diseño del node-red, es importante instalar el plugin, de *node-red-contrib-influxdb* para poder agregar las funciones al diagrama.

Fuente: Figura 11 información recogida por el grupo 3

Debe aparecer de la siguiente manera:

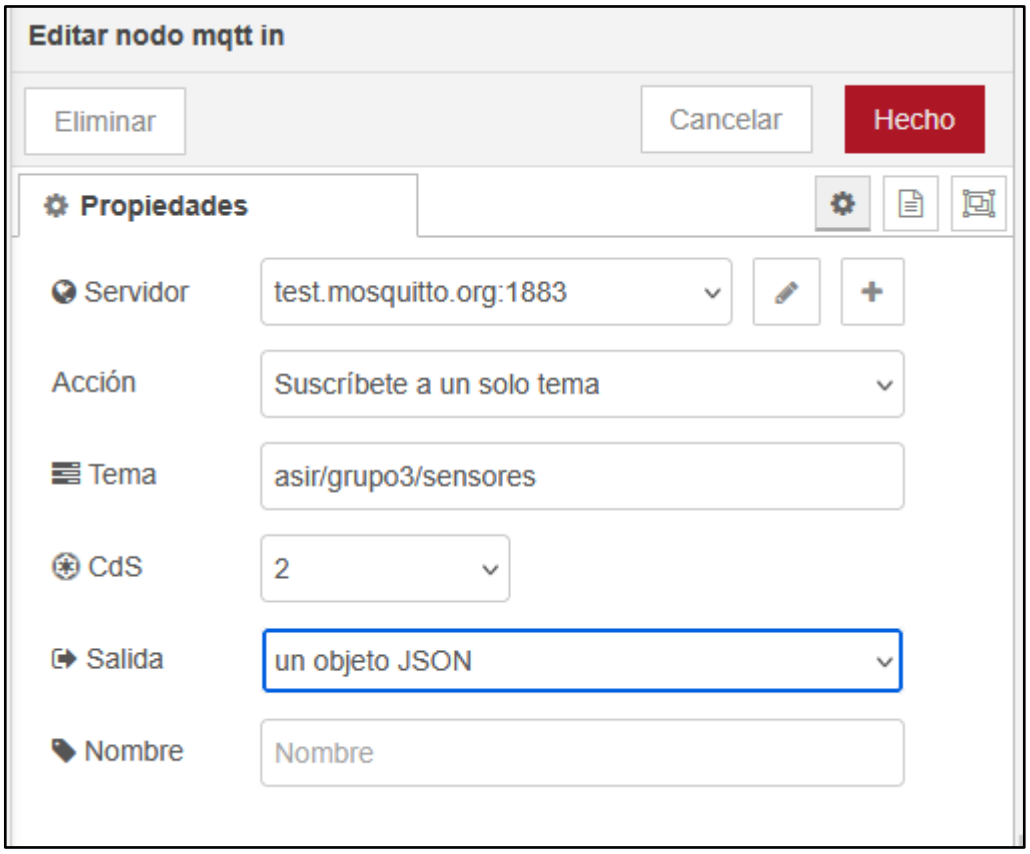
Fuente: Figura 12 información recogida por el grupo 3

A continuación se realizará paso por paso el diagrama de flujo y como se ha ido configurando en el proceso. El resultado final del diagrama de flujo fue el siguiente:



Fuente: Figura 13 información recogida por el grupo 3

1. En MQTT IN debemos poner el Broker de MQTT. En el apartado Tema debe ser el mismo que en Wokwi.



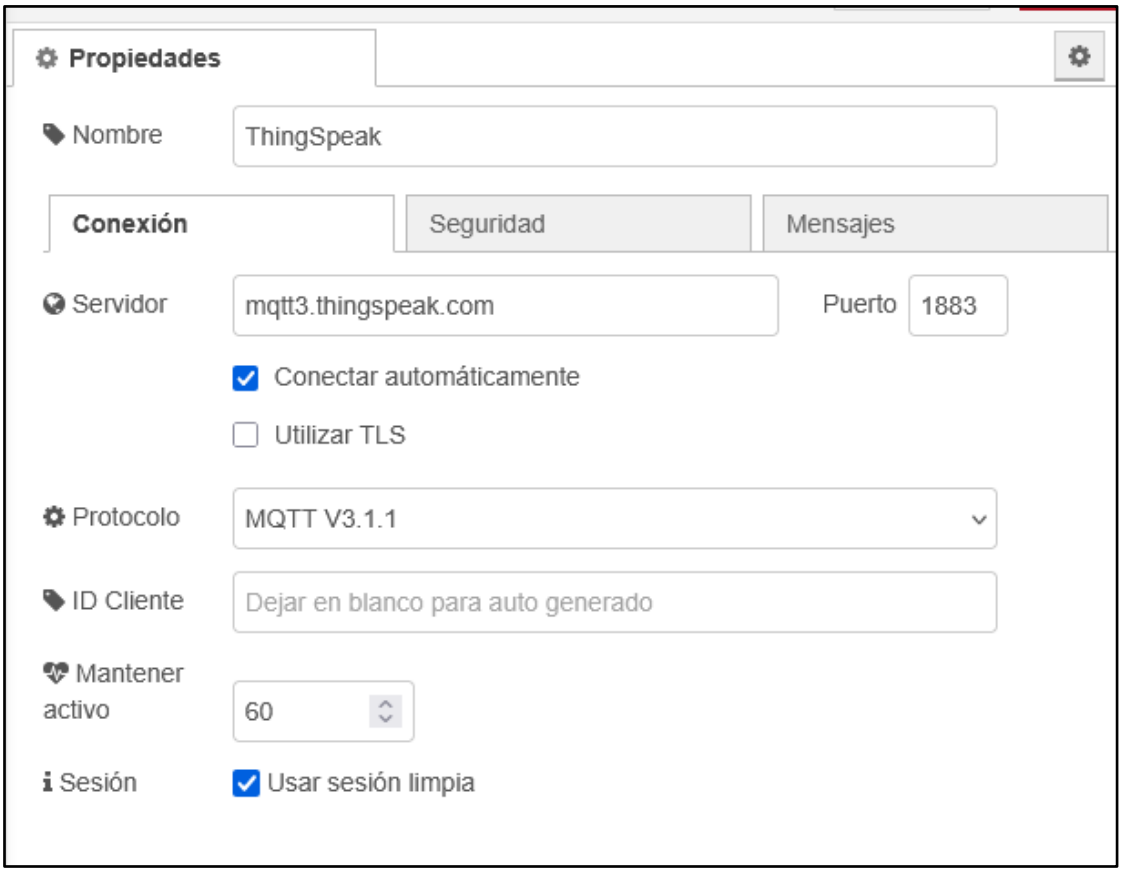
Fuente: Figura 14 información recogida por el grupo 3

2. Se realizaron también dos tipos de funciones, la función que está conectada MQTT es la *function 2* que es la que está unida al ThinkSpeak y la que nos permite la visualización de las gráficas mostradas. La función es la siguiente:

```
// 1. Extraer valores del payload que viene de Wokwi
const temp = msg.payload.temperatura;
const hum = msg.payload.humedad;

// 2. Construir el payload para ThingSpeak
msg.payload = `field1=${temp}&field2=${hum}`;

// 3. Configurar el topic de publicación de tu canal
msg.topic = "channels/3254060/publish";

return msg;
```

Fuente: Figura 15 información recogida por el grupo 3

3. La salida de MQTT es la siguiente, en el ID de cliente se debe de colocar la API obtenida anteriormente en ThingSpeak. Lo mismo en el apartado de seguridad, colocando los códigos correctos.



Fuente: Figura 16 información recogida por el grupo 3

4. La siguiente función está relacionada con la conexión de InfluxDB. La función recogerá los datos que se visualizan en InfluxDB.

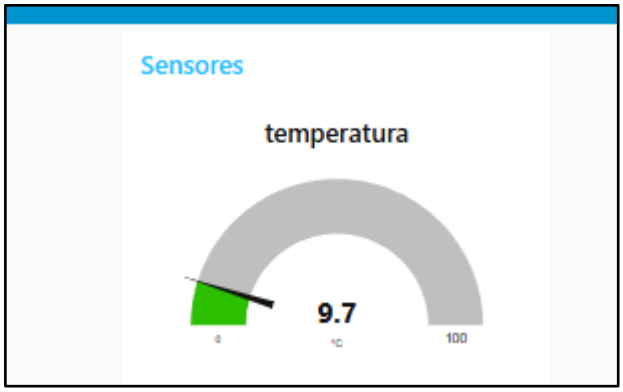
```
// 1. Convertir el payload de String/Buffer a Objeto JSON
let datos;
try {
    datos = typeof msg.payload === 'object' ? msg.payload : JSON.parse(msg.payload);
} catch (e) {
    node.error("Error al parsear JSON: " + e.message);
    return null;
}

// 2. Extraer valores usando los nombres correctos
let temp = datos.temperatura;
let hum = datos.humedad;

// 3. Definir campos para InfluxDB (asegurando números)
let fields = {
    humidity: parseFloat(hum),
    temperature: parseFloat(temp)
};

// 4. Etiquetas
let tags = {
    sensor_id: "ESP32_Grupo3",
    ubicacion: "Wokwi"
};

msg.measurement = "clima_sensores";
msg.payload = [fields, tags];

// Estado visual
node.status({ fill: "green", shape: "dot", text: `T: ${temp}°C H: ${hum}%` });

return msg;
```

Fuente: Figura 17 información recogida por el grupo 3

5. La salida de InfluxDB está configurada de la siguiente forma, debemos colocar los nombres ya establecidos.

Fuente: Figura 18 información recogida por el grupo 3

6. Por último se pide un visualizador de tipo Gauge que se ha unido a la función 1 que nos permite visualizar la temperatura.



Fuente: Figura 19 información recogida por el grupo 3

2.6. Almacenamiento: InfluxDB

Para comprobar si el InfluxDB está almacenando todos los datos, se deberá de comprobar entrando a la base de datos. Para ello, se tendrá que entrar con el usuario y contraseña previamente puesto en el fichero de docker-compose.yml

```
alumno@CSL-P4-A44-12:/mnt/d/RETO5/proyecto_iot/docker$ docker exec -it influxdb influx -username admin -password sanluis
Connected to http://localhost:8086 version 1.8.10
InfluxDB shell version: 1.8.10
> create database iot_test;
> USE iot_test
Using database iot_test
> INSERT prueba valor=10
> SELECT * FROM prueba
name: prueba
time                valor
----                -----
1770368808562649958 10
```

Fuente: Figura 20 información recogida por el grupo 3

Comprobamos la conexión:

```
alumno@CSL-P4-A44-12:/mnt/d/RETO5/proyecto_iot/docker$ curl -i localhost:8086/ping
HTTP/1.1 204 No Content
Content-Type: application/json
Request-Id: 5a93f233-033b-11f1-800d-7e59b3588643
X-Influxdb-Build: OSS
X-Influxdb-Version: 1.8.10
X-Request-Id: 5a93f233-033b-11f1-800d-7e59b3588643
Date: Fri, 06 Feb 2026 09:08:08 GMT
```

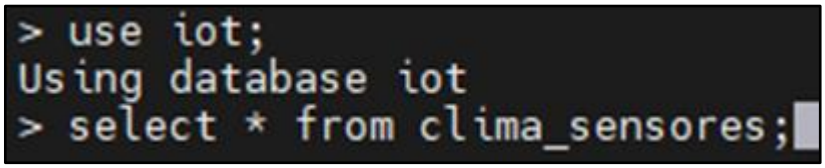
Fuente: Figura 21 información recogida por el grupo 3

2.6.1. Gestión de base de datos y series temporales

En cuanto al almacenamiento histórico, se hace uso de InfluxDB, la base de datos especializada en el manejo de series temporales.

Estructura de los datos: Las métricas se almacenan en la base de datos IoT bajo la medición (measurement) *clima_sensores*.

Validación de los datos insertados: Se ha podido comprobar la correcta inserción de datos desde la terminal, ejecutando las queries *SELECT* para verificar que cada registro contenga el la temperatura y la humedad registradas en la plataforma.



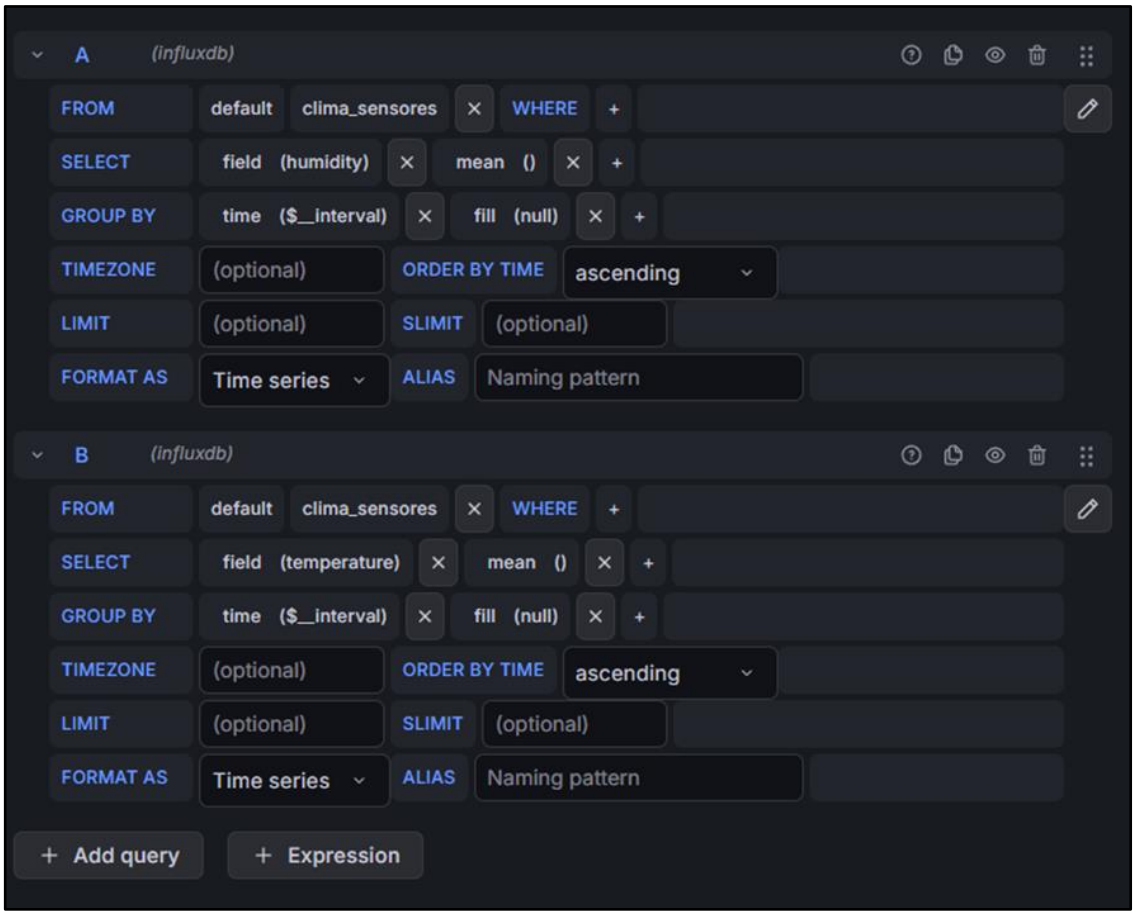
Fuente: Figura 22 información recogida por el grupo 3

2.7. Visualización y Alertas: Grafana

2.7.1. Dashboards, métricas y sistema de alertas

Grafana extiende la interfaz gráfica de usuario, ofreciendo la representación gráfica profesional de los datos que se han almacenado en InfluxDB.

Dashboards de Monitorización: Se ha preparado un dashboard que contiene gráficos de tipología Time series para poder observar la evolución histórica de la temperatura y la humedad.



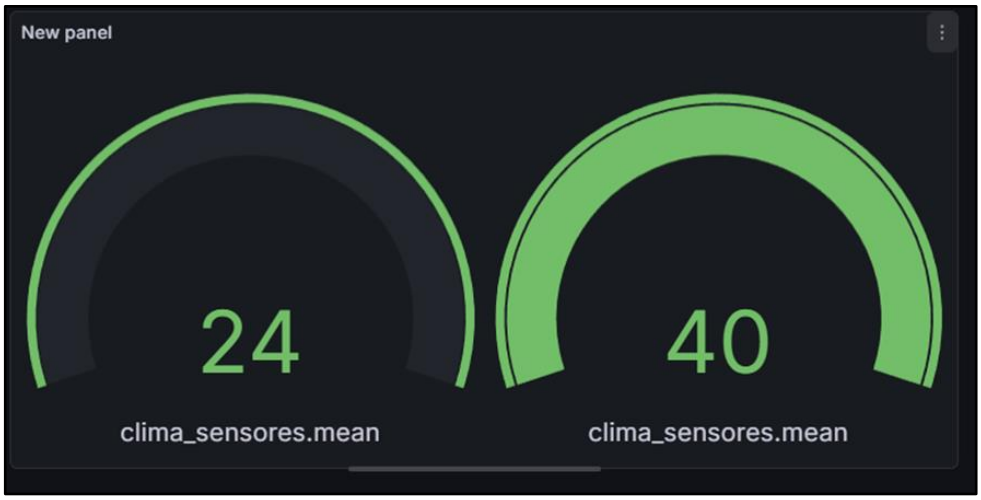
Fuente: Figura 23 información recogida por el grupo 3

Conectividad: El servicio está alimentado por InfluxDB a través del puerto 8086, utilizando el lenguaje de consulta InfluxQL para la extracción y la interconsultación de métricas.

Visualización de tipo Gráfica:

Fuente: Figura 24 información recogida por el grupo 3

Visualización de tipo Gauge:



Fuente: Figura 25 información recogida por el grupo 3

3. Conclusiones

La evolución de este trabajo ha servido para comprobar el ciclo de vida de un dato en un ecosistema IoT pleno. A partir de esto se puede concluir lo siguiente:

La Integración de tecnologías: se ha demostrado que era posible comunicar dispositivos de bajo coste (ESP32), con stacks de nivel industrial, mediante protocolos estándar como el MQTT y utilizando formatos ligeros como el JSON.

La Eficiencia de la contenerización: ha permitido que el despliegue de la infraestructura local sea drásticamente sencillo, ya que Docker Compose permite asegurar que los servicios (Node-RED, InfluxDB y Grafana) se ejecuten sin problemas y que tengan persistencia de los datos.

La Redundancia y Visibilidad: el sistema dual (Cloud con ThingSpeak y Local con el Docker Stack) aumenta la robustez necesaria en el caso de entornos profesionales al permitir hacer supervisión remota del sistema y realizar un análisis histórico.

La Automatización: la posibilidad de generar alertas automáticas permite que un sistema simple de lectura se convierta en un sistema activo de supervisión, necesaria para el mantenimiento preventivo que requieren entornos industriales.

4. Anexos

4.1. Script de MicroPython (*main.py*)

```
import network
import time
from machine import Pin
import dht
import ujson
from umqtt.simple import MQTTClient


# CONFIGURACIÓN DE REQUISITOS
MQTT_CLIENT_ID = "esp32-asir-cliente"
MQTT_BROKER    = "test.mosquitto.org"
MQTT_TOPIC     = "asir/grupo3/sensores"


sensor = dht.DHT22(Pin(15))


# Conexión a la red Wokwi-GUEST
def conectar_wifi():
    print("Conectando a WiFi", end="")
    sta_if = network.WLAN(network.STA_IF)
    sta_if.active(True)
    sta_if.connect('Wokwi-GUEST', '')
    while not sta_if.isconnected():
        print(".", end="")
        time.sleep(0.1)
    print(" ¡Conectado!")


# Configuración Cliente MQTT
def conectar_mqtt():
```

```
    client = MQTTClient(MQTT_CLIENT_ID, MQTT_BROKER)
    client.connect()
    print("Conectado a Mosquitto")
    return client

conectar_wifi()
cliente = conectar_mqtt()

# Bucle de envío temporizado
while True:
    try:
        print("Midiendo...", end="")
        sensor.measure() # Lectura DHT22

        # Empaquetado en JSON
        payload = ujson.dumps({
            "temperatura": sensor.temperature(),
            "humedad": sensor.humidity(),
            "timestamp": time.time()
        })

        # Publicación en el topic exacto
        print(f"Enviando a {MQTT_TOPIC}: {payload}")
        cliente.publish(MQTT_TOPIC, payload)

    except OSError as e:
        print("Error al leer sensor.")

    time.sleep(10)
```

4.2. Función 1

```
// 1. Convertir el payload de String/Buffer a Objeto JSON
let datos;
try {
    datos = typeof msg.payload === 'object' ? msg.payload :
JSON.parse(msg.payload);
} catch (e) {
    node.error("Error al parsear JSON: " + e.message);
    return null;
}

// 2. Extraer valores usando los nombres correctos
let temp = datos.temperatura;
let hum = datos.humedad;

// 3. Definir campos para InfluxDB (asegurando números)
let fields = {
    humidity: parseFloat(hum),
    temperature: parseFloat(temp)
};

// 4. Etiquetas
let tags = {
    sensor_id: "ESP32_Grupo3",
    ubicacion: "Wokwi"
};

msg.measurement = "clima_sensores";
msg.payload = [fields, tags];

// Estado visual
node.status({ fill: "green", shape: "dot", text: `T: ${temp}°C H:
${hum}%` });

return msg;
```

4.3. Función 2

```
// 1. Extraer valores del payload que viene de Wokwi
const temp = msg.payload.temperatura;
const hum = msg.payload.humedad;

// 2. Construir el payload para ThingSpeak
msg.payload = `field1=${temp}&field2=${hum}`;

// 3. Configurar el topic de publicación de tu canal
msg.topic = "channels/3254060/publish";

return msg;
```

4.4. Docker Compose

```
services:
  # Procesar y gestionar los datos
  nodered:
    image: nodered/node-red:latest
    container_name: nodered
    restart: unless-stopped
    ports:
      - "1880:1880"
    volumes:
      - ./data/nodered:/data
    networks:
      - iot_network

  # Almacenar el histórico de datos
  influxdb:
    image: influxdb:1.8
    container_name: influxdb
    restart: unless-stopped
    ports:
      - "8086:8086"
    environment:
      - INFLUXDB_DB=iot
      - INFLUXDB_HTTP_AUTH_ENABLED=true
      - INFLUXDB_ADMIN_USER=admin
      - INFLUXDB_ADMIN_PASSWORD=sanluis
      - INFLUXDB_USER=iotuser
```

```
      - INFLUXDB_USER_PASSWORD=iot123
    volumes:
      - ./data/influxdb:/var/lib/influxdb
    networks:
      - iot_network


  # Visualización y alertar
  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    restart: unless-stopped
    ports:
      - "3000:3000"
    volumes:
      - ./data/grafana:/var/lib/grafana
    networks:
      - iot_network
    depends_on:
      - influxdb


networks:
  iot_networks:
```