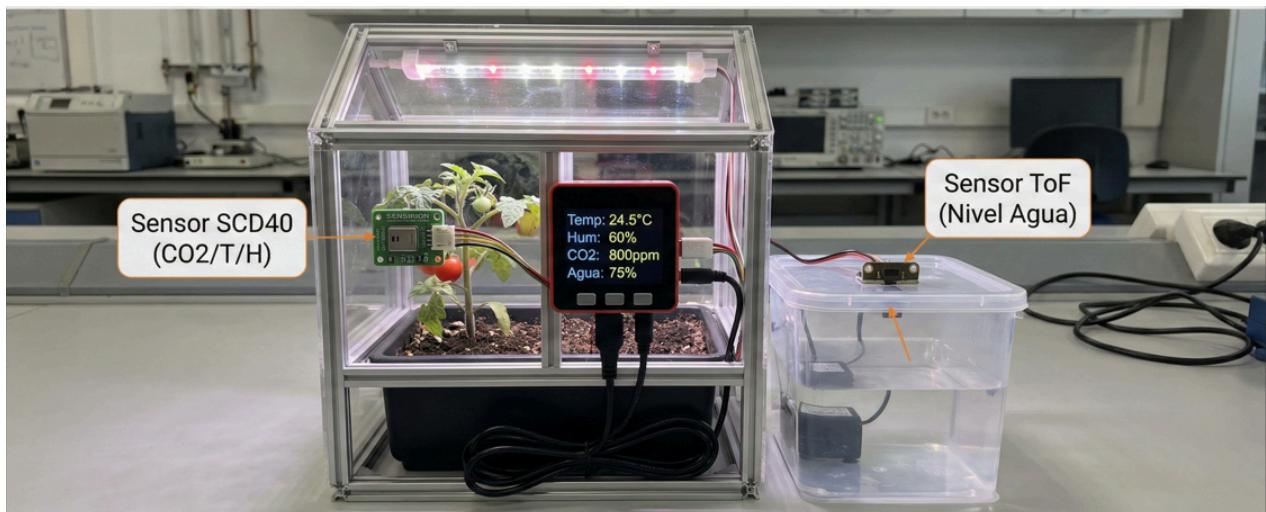


# MEMORIA DEL PROYECTO FINAL:

# SISTEMA IOT PARA LA GESTIÓN INTELIGENTE DE UN INVERNADERO



Raúl Sánchez Ibáñez y Carmen Sánchez del Vas  
Internet de Nueva Generación y Tratamiento de Datos

2025 / 2026

# Índice

1. DEFINICIÓN DEL CASO PRÁCTICO	3
1.1. Contexto y Justificación	3
1.2. Objetivos del Sistema	3
2. DISEÑO DEL SISTEMA IoT	3
2.1. Arquitectura del Sistema	3
2.2. Componentes	3
3. PROTOTIPO E IMPLEMENTACIÓN	4
3.1. Configuración y Conectividad Segura	4
3.2. Gestión de Actuadores y Alertas (Edge Computing)	5
3.2.1. Interfaz Visual	5
3.2.2. Perfiles de Alerta Multimodal	5
3.3. Ejecución Remota de Comandos (RPC y MQTT)	8
3.4. Inicialización de Hardware y Sensores (Protocolo I2C)	8
3.5. Bucle principal: Adquisición, Procesamiento y Visualización	10
3.5.1. Adquisición y Procesamiento de Señales (SCD40)	10
3.5.2. Monitorización de Recursos Hídricos (ToF)	11
3.5.3. Interfaz local rotativa	13
3.5.4. Comunicación Bidireccional con ThingsBoard	13
4. GESTIÓN DE DATOS Y VISUALIZACIÓN	14
4.1. Diseño del Dashboard de Supervisión	14
4.2. Motor de Reglas y Configuración de Alarmas	16
4.2.1. Gestión Térmica Escalonada	17
4.2.2. Gestión Crítica de Riego	18
4.2.3. Gestión umbrales de calidad del Aire (CO2 y Humedad)	19
4.3. Automatización y Respuesta Activa (Rule Chains)	21
4.3.1. Diagrama de Flujo (Root Rule Chain)	21
4.3.1. Transformación de Mensajes (Script Node)	23
4.4. Validación y registro de eventos	25

# 1. DEFINICIÓN DEL CASO PRÁCTICO

## 1.1. Contexto y Justificación

Este proyecto se sitúa en el ámbito de la agricultura moderna, enfocándose en la importancia crítica de la optimización de recursos y el control ambiental. Aprovechando los beneficios que el IoT aporta a la agricultura inteligente, nuestra propuesta se centra en el desarrollo de un Invernadero Inteligente.

Hemos seleccionado como caso de uso un Invernadero Inteligente. Este tipo de cultivo requiere mantener parámetros estrictos de temperatura, humedad, calidad del aire y disponibilidad de agua entre otros. Un fallo en el sistema de riego o un nivel inadecuado de CO<sub>2</sub> pueden arruinar una cosecha en cuestión de horas.

## 1.2. Objetivos del Sistema

El objetivo es diseñar un sistema automatizado que resuelva la monitorización manual ineficiente. El sistema debe:

1. Monitorizar variables ambientales: Temperatura, Humedad Relativa y concentración de CO<sub>2</sub>.
2. Gestionar el recurso hídrico: Monitorizar el nivel de llenado del tanque de agua de riego para evitar que la bomba trabaje en vacío o que las plantas se queden sin agua.
3. Actuar y Alertar: Generar alertas visuales y sonoras del dispositivo M5 y notificaciones remotas mediante ThingsBoard.

# 2. DISEÑO DEL SISTEMA IoT

Para el diseño de la solución, hemos seguido lo siguiente:

## 2.1. Arquitectura del Sistema

El sistema sigue una arquitectura híbrida Edge-Cloud:

- Capa de Percepción (Edge): Sensores conectados vía I2C al microcontrolador.
- Capa de Red: Conectividad Wi-Fi utilizando el protocolo MQTT sobre SSL para una transmisión segura.
- Capa de Aplicación (Cloud): Plataforma ThingsBoard para la ingesta, visualización y gestión de reglas de alarma (RPC).

## 2.2. Componentes

1. Dispositivo IoT (M5Stack Fire): capacidad de procesamiento, conectividad Wi-Fi, periféricos de salida (Pantalla, Altavoz, LEDs RGB) que facilitan el feedback local al operario del invernadero.

2. Sensor Ambiental (Unit SCD40): es un sensor de CO2 de alta precisión que también integra temperatura y humedad. Esto nos permite monitorizar tres variables críticas ocupando un solo puerto I2C y reduciendo el cableado.
3. Sensor de Nivel (Unit ToF): Para medir el agua del depósito de riego, el sensor mide la distancia hasta la superficie del agua sin contacto físico. Esto evita la corrosión de los sensores y la contaminación del agua de riego.

## 3. PROTOTIPO E IMPLEMENTACIÓN

El desarrollo del prototipo se ha realizado utilizando MicroPython sobre el dispositivo M5Stack Fire. El código fuente, contenido en el fichero proyecto.py, integra la lectura de sensores avanzados, la lógica de control local y la comunicación segura con la nube.

A continuación, se detallan los bloques funcionales del software desarrollado.

### 3.1. Configuración y Conectividad Segura

El primer paso en la lógica del sistema es el establecimiento de las comunicaciones. Se definen las constantes para la conexión Wi-Fi y las credenciales del servidor MQTT (ThingsBoard).

Es importante destacar que, para garantizar la seguridad de los datos del invernadero, la conexión MQTT se realiza utilizando SSL/TLS en el puerto 8883, asegurando que la telemetría no viaje en texto plano. Además, se definen los topics necesarios para la telemetría (v1/devices/me/telemetry) y para la recepción de comandos remotos (RPC).

```
mqttclient = None
mqttuser = "CarmenRaul"
mqttpass = "1102025data"
mqtturl = "mqtt.eu.thingsboard.cloud"
mqttclientid = "ingCarmenRaul"
puerto = 8883

topic_telemetria = "v1/devices/me/telemetry"
topic_atributos = "v1/devices/me/attributes"
topic_rpc_sub = "v1/devices/me/rpc/request/+"
```

### 3.2. Gestión de Actuadores y Alertas (Edge Computing)

Para dotar al sistema de capacidad de respuesta inmediata en M5 (Edge), se ha desarrollado un módulo de software encargado de interactuar con los actuadores físicos del M5Stack (Pantalla LCD, Tira de LEDs RGB y Altavoz).

### 3.2.1. Interfaz Visual

Dado que el sistema debe notificar diversos tipos de alarmas, se ha implementado una función auxiliar denominada `mostrar_alerta_pantalla(mensaje, color)`. Esta función encapsula la lógica de renderizado gráfico:

1. Limpieza: Borra el contenido previo (`M5.Lcd.clear()`) para asegurar que la alerta es legible.
2. Cabecera: Imprime el texto "**¡ALERTA!**" con un tamaño de fuente grande y en el color asociado a la urgencia (Rojo, Naranja, Azul, etc.).
3. Mensaje: Muestra el detalle del evento en color blanco.

```
def mostrar_alerta_pantalla(mensaje, color):  
    M5.Lcd.clear()  
    M5.Lcd.setCursor(10, 40)  
    M5.Lcd.setTextSize(3)  
    M5.Lcd.setTextColor(color)  
    M5.Lcd.print("¡ALERTA!")  
  
    M5.Lcd.setCursor(10, 90)  
    M5.Lcd.setTextSize(2)  
    M5.Lcd.setTextColor(0xFFFFFFFF)  
    M5.Lcd.print(mensaje)
```

### 3.2.2. Perfiles de Alerta Multimodal

Se han definido funciones específicas para cada tipo de evento ambiental. Estas funciones combinan feedback visual (LEDs y Pantalla) y feedback auditivo (Tonos) para garantizar que el operario perciba la alerta incluso si no está mirando el dispositivo.

- **Gestión del Recurso Hídrico:** Se ha diseñado un patrón de parpadeo visual sin sonido. La tira LED parpadea en color blanco tres veces (`range(3)`) con intervalos de 0.3 segundos, indicando la necesidad de mantenimiento (rellenado) y un mensaje “ALERTA: nivel de agua bajo”.

```
def alerta_agua_baja():  
    global rgb15  
    print("ALERTA: Nivel de agua bajo")  
    mostrar_alerta_pantalla("RELENAR AGUA", 0xFFFFFFFF)  
    for i in range(3):  
        rgb15.fill_color(0xFFFFFFFF)  
        time.sleep(0.3)  
        rgb15.fill_color(0x000000)  
        time.sleep(0.3)
```

- **Gestión de Temperatura:** Se distinguen tres niveles de severidad:
  - Crítica (`alerta_temp_critica`): Es la alerta de mayor prioridad. Ejecuta un bucle de 5 iteraciones donde sincroniza el parpadeo de LEDs rojos con un tono agudo de 2000Hz (`M5.Speaker.tone`), generando una señal de peligro inminente y un mensaje de “TEMP CRITICA”.

- Alta/Baja: Alertas informativas que cambian el color a Rojo o Azul respectivamente y emiten un tono continuo de distinta frecuencia (2000Hz para alta, 500Hz para baja) durante un segundo y un mensaje “TEMP MUY ALTA o MUY BAJA”

```

def alerta_temp_critica():
    global rgb15
    print("EJECUTANDO: Alerta Temperatura Muy Alta")
    mostrar_alerta_pantalla("TEMP CRITICA", 0xFF0000)
    for i in range(5):
        rgb15.fill_color(0xFF0000)
        M5.Speaker.tone(2000, 300)
        time.sleep(0.3)
        rgb15.fill_color(0x000000)
        time.sleep(0.3)

def alerta_temp_alta():
    global rgb15
    print("EJECUTANDO: Alerta Temperatura Alta")
    rgb15.fill_color(0xFF0000) # ROJO
    M5.Speaker.tone(2000, 1000)
    mostrar_alerta_pantalla("TEMP MUY ALTA", 0xFF0000)

def alerta_temp_baja():
    global rgb15
    print("EJECUTANDO: Alerta Temperatura Baja")
    rgb15.fill_color(0x0000FF) # AZUL
    M5.Speaker.tone(500, 1000)
    mostrar_alerta_pantalla("TEMP MUY BAJA", 0x0000FF)

```

- Gestión de Calidad del Aire (CO2 y Humedad):

Para gestionar estas alertas, hemos implementado cuatro funciones específicas:

#### A) Monitorización de Humedad:

Las alertas de humedad se han diseñado para indicar estados de saturación o sequedad extrema.

- Alerta de Humedad Alta (alerta\_hum\_alta):
  - Se utiliza el color Naranja (0xFFA500), asociado a "Precaución", emitiendo un tono continuo de frecuencia media-alta (2000Hz) durante 1 segundo y un mensaje: "HUMEDAD ALTA".
- Alerta de Humedad Baja (alerta\_hum\_baja):
  - Se emplea el color Cyan (0x00FFFF), evocando la falta de agua o necesidad de frescura. Se utiliza un tono grave de 500Hz durante 1 segundo, indicando una condición negativa pero no crítica junto con un mensaje: "HUMEDAD BAJA".

```

def alerta_hum_alta():
    global rgb15
    print("EJECUTANDO: Alerta Humedad Alta")
    rgb15.fill_color(0xFFA500) # NARANJA
    M5.Speaker.tone(2000, 1000)
    mostrar_alerta_pantalla("HUMEDAD ALTA", 0xFFA500)

```

```

def alerta_hum_baja():
    global rgb15
    print("EJECUTANDO: Alerta Humedad Baja")
    rgb15.fill_color(0x00FFFF) # CYAN
    M5.Speaker.tone(500, 1000)
    mostrar_alerta_pantalla("HUMEDAD BAJA", 0x00FFFF)

```

## B) Monitorización de CO2

El CO2 es un parámetro invisible y, en concentraciones muy altas, puede ser peligroso en espacios cerrados.

- Alerta Crítica de CO2 (alerta\_co2\_alta):
  - Se usa el color Magenta (0xFF00FF). A diferencia de las otras alertas que usan tonos continuos, aquí hemos implementado un patrón rítmico de doble pitido. El código ejecuta un tono de 3000Hz (muy agudo y penetrante) durante 200ms, hace una pausa de silencio de 300ms, y repite el tono.leza (plantas), indicando una anomalía química. Emite tambien un mensaje: "ALERTA CO2 CRITICO".
- Estado Óptimo (alerta\_co2\_baja):
  - Esta función actúa como confirmación positiva. Utiliza el color Verde (0x00FF00) y un tono suave y grave (500Hz), indicando que los niveles han returnedo a la normalidad y son seguros. Envía un mensaje ("CO2 BAJO (OK)")

```

def alerta_co2_alta():
    global rgb15
    print("EJECUTANDO: Alerta CO2 Alta")
    rgb15.fill_color(0xFF00FF) # MAGENTA
    M5.Speaker.tone(3000, 200)
    time.sleep(0.3)
    M5.Speaker.tone(3000, 200)
    mostrar_alerta_pantalla("NIVEL CO2 CRITICO", 0xFF00FF)

def alerta_co2_baja():
    global rgb15
    print("EJECUTANDO: Alerta CO2 Baja")
    rgb15.fill_color(0x00FF00) # VERDE
    M5.Speaker.tone(500, 500)
    mostrar_alerta_pantalla("CO2 BAJO (OK)", 0x00FF00)

```

## 3.3. Ejecución Remota de Comandos (RPC y MQTT)

Aquí mostramos el puente entre la herramienta ThingsBoard y los actuadores físicos. Todo esto es posible con la función de callback sub\_cb, mediante el cual el dispositivo se suscribe al topic de RPC de ThingsBoard.

Esta función actúa como un despachador de eventos. Cuando llega un mensaje MQTT al tópico de RPC (v1/devices/me/rpc/request/+), el código realiza los siguientes pasos:

1. Convierte el mensaje de bytes a cadena y crea un objeto JSON.
2. Extrae el campo method del JSON.
3. Mediante una estructura condicional compara el método solicitado con los disponibles y ejecuta la función Python correspondiente (alerta\_temp\_alta, alerta\_agua\_baja, etc.).

```
def sub_cb(topic, msg):
    print("RPC Recibido:", topic, msg)
    topic_str = topic.decode('utf-8')
    msg_str = msg.decode('utf-8')

    if topic_str.startswith('v1/devices/me/rpc/request/'):
        try:
            data = json.loads(msg_str)
            metodo = data['method']

            if metodo == 'alerta_temp_alta': alerta_temp_alta()
            elif metodo == 'alerta_temp_baja': alerta_temp_baja()
            elif metodo == 'alerta_hum_alta': alerta_hum_alta()
            elif metodo == 'alerta_hum_baja': alerta_hum_baja()
            elif metodo == 'alerta_co2_alta': alerta_co2_alta()
            elif metodo == 'alerta_co2_baja': alerta_co2_baja()
            elif metodo == 'alerta_agua_baja': alerta_agua_baja()
            elif metodo == 'alerta_temp_critica': alerta_temp_critica()
            else: print("Metodo RPC desconocido:", metodo)

        except Exception as e:
            print("Error procesando RPC:", e)
```

### 3.4. Inicialización de Hardware y Sensores (Protocolo I2C)

La función setup() actúa como el punto de entrada del sistema. Su objetivo es configurar el entorno de ejecución, establecer los buses de comunicación y asegurar la conexión con la nube antes de entrar en el bucle de control.

Primero se configura el hardware local (HMI y Sensores) y posteriormente la capa de red.

Al inicio, se inicializa el núcleo del M5Stack (M5.begin()) y se configuran los elementos de interacción con el usuario: la pantalla y los indicadores LED (Se instancia el objeto RGB asociado al pin 15).

Posteriormente, se inicia la comunicación con los sensores, a través del bus I2C estándar. La inicialización de los sensores incluye estrategias de manejo de errores (try-except) para evitar que el fallo de un sensor bloquee todo el sistema:

1. Arranque del SCD40 (Nivel Bajo): Para el sensor de CO<sub>2</sub>, interactuamos a bajo nivel enviando comandos hexadecimales, con la instrucción i2c0.writeto(SCD40\_ADDR, b'\x21\xb1') se envía el comando start\_periodic\_measurement al dispositivo.
2. Arranque del ToF (Abstracción): Para el sensor de distancia, utilizamos la clase ToFUnit, vinculándola al bus I2C previamente creado.

Una vez inicializado el hardware, el sistema establece el enlace con la nube (ThingsBoard). Este bloque destaca por la implementación de seguridad en la capa de transporte:

1. Conexión Wi-Fi: Se llama a la función auxiliar connect\_wifi que gestiona el handshake con el punto de acceso.
2. Cliente MQTT Seguro: Se instancia el cliente MQTTClient configurando parámetros esenciales para un entorno de producción IoT (keepalive para evitar desconexiones, encriptacion SSL/TLS para la seguridad el canal)
3. Configuración de Callbacks: Antes de conectar, se asigna la función sub\_cb mediante mqttclient.set\_callback(sub\_cb), que prepara al dispositivo para escuchar y reaccionar a las órdenes remotas (RPC)

```
def setup():
    global wifiFire, i2c0, mqttclient, rgb15, tof

    M5.begin()
    Widgets.setRotation(1)
    Widgets.fillScreen(0x222222)

    rgb15 = RGB(io=15, n=10, type="SK6812")
    rgb15.set_brightness(30)
    rgb15.fill_color(0x000000)

    print("Iniciando I2C...")
    i2c0 = I2C(0, scl=Pin(22), sda=Pin(21), freq=100000)

    # SCD40
    try:
        i2c0.writeto(SCD40_ADDR, b'\x21\xb1')
        print("SCD40 inicializado correctamente.")
        time.sleep(1)
    except Exception as e:
        print("Error iniciando SCD40:", e)
        M5.Lcd.print("Error Sensor SCD40")

    # TOF
    try:
        print("Iniciando ToF...")
        tof = ToFUnit(i2c=i2c0)
    except Exception as e: print("Error init ToF:", e)

    wifiFire = connect_wifi(SSID, PASSWORD)

    ssl_params = {"server_hostname": mqtturl}
    mqttclient = MQTTClient(client_id=mqttclientid,
                           server=mqtturl,
                           port=puerto,
                           user=mqttuser,
                           password=mqttpass,
                           keepalive=60,
                           ssl=True,
                           ssl_params=ssl_params)
    mqttclient.set_callback(sub_cb)
    ensure_mqtt_connected()
```

## 3.5. Bucle principal: Adquisición, Procesamiento y Visualización

El núcleo del firmware reside en la función loop(), este ciclo incluye la sincronización entre la lectura de sensores físicos, el procesamiento matemático de las señales, la actualización de la interfaz local y la transmisión de datos a la nube. El ciclo tiene una periodicidad definida de 5 segundos (time.sleep(5)) (al fin de la función)

### 3.5.1. Adquisición y Procesamiento de Señales (SCD40)

En este apartado, explicaremos la lectura del sensor SCD40, que no es instantánea; sino que requiere seguir el protocolo de comunicación I2C.

Primero, la fase de petición, en la cual el sensor no está enviando datos constantemente sino que nuestro M5Stack debe pedirle explícitamente la medición mediante i2c0.writeto(SCD40\_ADDR, b'\xec\x05')

Una vez el sensor está listo, le pedimos que nos envíe los datos, leyendo 9 bytes, ya que el sensor nos envía la información en bloques de 3 bytes: data = i2c0.readfrom(SCD40\_ADDR, 9)

La estructura que recibimos del sensor tiene la siguiente forma data:

- data[0] y data[1]: Valor de CO2.
- data[2]: CRC (Checksum) del CO2 (lo saltamos).
- data[3] y data[4]: Valor de Temperatura.
- data[5]: CRC de Temp (lo saltamos).
- data[6] y data[7]: Valor de Humedad.
- data[8]: CRC de Humedad (lo saltamos).

Una vez tenemos los datos, tenemos la reconstrucción matemática, ya que los sensores trabajan con números de 16 bits pero nuestro cable I2C solo puede transportar paquetes de 8 bits, por lo tanto hacemos una concatenación a nivel de bit: se desplaza el byte más significativo (MSB) 8 posiciones a la izquierda ( $\ll 8$ ) y se combina con el byte menos significativo (LSB) mediante una operación OR lógica ( $\|$ ).

Una vez reconstruido el valor entero (rango 0-65535), para el CO<sub>2</sub>, el dato resultante corresponde directamente a la concentración en ppm. Pero en cambio, para temperatura y Humedad, se requieren transformaciones finales para mapear el valor digital al rango físico correspondiente (Ecuaciones de conversión del fabricante Sensirion para obtener las magnitudes reales físicas).<sup>1</sup>

---

<sup>1</sup> Hoja de datos técnica del fabricante del sensor SCD40, sección 3.6 Basic Commands  
[https://sensirion.com/media/documents/48C4B7FB/67FE0194/CD\\_DS\\_SCD4x\\_Datasheet\\_D1.pdf](https://sensirion.com/media/documents/48C4B7FB/67FE0194/CD_DS_SCD4x_Datasheet_D1.pdf)

Write (hexadecimal)	Input parameter: -		Response parameter: CO <sub>2</sub> , Temperature, Relative Humidity		Max. command duration [ms]
	length [bytes]	signal conversion	length [bytes]	signal conversion	
0xec05	-	-	3	$CO_2 \text{ [ppm]} = word[0]$	1
			3	$T = -45 + 175 * \frac{word[1]}{2^{16} - 1}$	
			3	$RH = 100 * \frac{word[2]}{2^{16} - 1}$	

Figura 1: Tabla de Conversiones, Fuente: fabricante Sensirion SCD40

- Temperatura: se mapea el valor de 16 bits al rango de operación del sensor (-45 a 130 °C).
- Humedad Relativa: Se escala el valor al porcentaje de saturación.

```
try:
    i2c0.writeto(SCD40_ADDR, b'\xec\x05')
    time.sleep(0.1)
    data = i2c0.readfrom(SCD40_ADDR, 9)
    co2 = (data[0] << 8) | data[1]
    temp_raw = (data[3] << 8) | data[4]
    temperatura = -45 + 175 * (temp_raw / 65535.0)
    hum_raw = (data[6] << 8) | data[7]
    humedad = 100 * (hum_raw / 65535.0)
    lectura_ok = True
except Exception as e:
    print("Error leyendo sensor SCD40:", e)
```

### 3.5.2. Monitorización de Recursos Hídricos (ToF)

Para la gestión del tanque de riego, el sistema utiliza el sensor de distancia Time of Flight (ToF), el cuál imaginamos que se coloca en la tapa del depósito mirando hacia abajo. Por lo tanto, el sensor no mide la cantidad de agua, sino la distancia libre desde la tapa hasta la superficie del líquido (el aire). Por tanto, la relación entre la medida del sensor y el nivel de llenado es inversamente proporcional: cuanto mayor es la distancia medida, menor es la cantidad de agua.

Para implementar esto en software, hemos modelado el depósito cilíndrico con una altura total constante  $H_{total}$ , definida en el código como  $ALTURA_BOTE = 20$ . La deducción matemática empleada es la siguiente:

1. Cálculo de la columna de agua ( $h_{agua}$ ): Si el sensor mide una distancia  $d$ (aire), la altura real del agua es la diferencia entre la altura total del bote y esa medición:

$$h_{agua} = H_{total} - d$$

2. Conversión a Porcentaje (Nivel %): Para normalizar el dato, dividimos la altura de agua entre la altura total y multiplicamos por 100:

$$Nivel\% = \frac{h_{agua}}{H_{total}} \times 100$$

3. Sustitución y simplificación: Sustituimos  $h_{agua}$  por la igualdad del paso 1:

$$Nivel\% = \frac{H_{total} - d}{H_{total}} \times 100$$

Separamos los términos de la fracción

$$Nivel\% = \left( \frac{H_{total}}{H_{total}} - \frac{d}{H_{total}} \right) \times 100$$

Dado que el primer término es igual a 1, llegamos a la fórmula del código:

$$Nivel\% = \left( 1 - \frac{d}{H_{total}} \right) \times 100 \implies 100 - \left( \frac{d}{H_{total}} \times 100 \right)$$

```
if tof:
    try:
        dist_cm = tof.get_distance()

        if dist_cm < ALTURA_BOTE:
            nivel_agua = 100 - ((dist_cm / ALTURA_BOTE) * 100)
        else:
            nivel_agua = 0

        if nivel_agua < 0: nivel_agua = 0
    except: pass
```

De este modo, si la distancia medida es 0 cm (el agua toca el sensor), el resultado es  $100 - 0 = 100\%$ . Si la distancia es 20 cm (fondo del bote), el resultado es  $100 - 100 = 0\%$ .

### 3.5.3. Interfaz local rotativa

Dado que la pantalla del M5Stack tiene dimensiones limitadas para mostrar cinco variables simultáneamente de forma legible, se ha implementado una estrategia de visualización secuencial.

Se utiliza una variable de estado contador\_pantalla que incrementa su valor en cada iteración del bucle. Esto permite rotar la información mostrada cada 5 segundos:

1. Estado 0: Muestra Temperatura (Naranja).
2. Estado 1: Muestra Humedad (Azul).
3. Estado 2: Muestra CO2 (Verde o Rojo según nivel de peligro).

```

if lectura_ok:
    rgb15.fill_color(0x000000)

    # Lógica de Pantalla
    if contador_pantalla == 0:
        dibujar_pantalla("TEMP", temperatura, "Grados C", 0xffaa00)
        contador_pantalla = 1
    elif contador_pantalla == 1:
        dibujar_pantalla("HUMEDAD", humedad, "% RH", 0x00aaff)
        contador_pantalla = 2
    else:
        color_co2 = 0xff0000 if co2 > 1000 else 0x00ff00
        dibujar_pantalla("NIVEL CO2", co2, "ppm", color_co2)
        contador_pantalla = 0

```



### 3.5.4. Comunicación Bidireccional con ThingsBoard

La fase final del ciclo es la sincronización con la plataforma IoT ThingsBoard.

1. Construcción del Payload: Se genera un diccionario JSON que estructura todas las variables procesadas (temperature, humidity, co2, water\_level\_percent) y datos de diagnóstico (distance\_mm).
2. Publicación MQTT: Se utiliza el método publish() para enviar el JSON al topic v1/devices/me/telemetry.
3. Recepción de Comandos (RPC): con llamada a mqttclient.check\_msg() se verifica si hay mensajes entrantes en la cola de suscripción. Sin esta llamada, el dispositivo enviaría datos pero sería incapaz de recibir las órdenes de alerta (RPC) generadas por las reglas de la nube, rompiendo la bidireccionalidad del sistema.

```

if wifiFire and wifiFire.isconnected():
    ensure_mqtt_connected()
    try:
        telemetry_data = {

```

```

        "temperature": temperatura,
        "humidity": humedad,
        "co2": co2,
        "water_level_percent": nivel_agua,
        "distance_mm": dist_cm * 10
    }

    mqttclient.publish(topic_telemetria, json.dumps(telemetry_data))

    attributes_data = {"status": "Online"}
    mqttclient.publish(topic_atributos, json.dumps(attributes_data))

    mqttclient.check_msg()

    print(f"Enviado: CO2={co2}, T={temperatura:.1f}, H={humedad:.1f}%, WaterLevel={nivel_agua:.1f}%, Dist={dist_cm}cm")

except Exception as e:
    print("Error enviando MQTT:", e)

```

## 4. GESTIÓN DE DATOS Y VISUALIZACIÓN

La plataforma **ThingsBoard** se ha desplegado como el núcleo de gestión centralizada (Cloud) del sistema. Actúa como interfaz de supervisión para el operario (Dashboard) y como motor de decisiones automatizado (Rule Engine).

### 4.1. Diseño del Dashboard de Supervisión

Se ha implementado un panel de control en tiempo real que permite la monitorización remota de las variables críticas del invernadero.

El diseño se ha personalizado seleccionando widgets específicos según la naturaleza de cada dato, configurando sus propiedades para facilitar la lectura rápida por parte del operario:

1. Gráficas con Histórico (Temperatura y CO2) Para las variables críticas, se han empleado Chart Cards. Estos widgets combinan el valor numérico actual con una gráfica de línea en el fondo para mostrar la tendencia reciente.
2. Representación del Tanque de Riego Para monitorizar el recurso hídrico, se ha optado por una representación que imita el depósito real, mediante la forma "Vertical Cylinder" y el layout en modo "Percentage". Esto alinea la visualización gráfica con la lógica del sensor ToF implementada en el código, que transforma la distancia medida en un porcentaje de llenado (0-100%).
3. Monitorización de Humedad: hemos optado por una visualización de tipo Horizontal Card, con un ícono representativo de gran tamaño para una identificación inmediata.

← → ⌂ eu.thingsboard.cloud/dashboards/all/90d9fbf0-d063-11f0-b7b2-b38eabeeff08

+ Add widget ⏱ ⚙️ 🖍️ ⌂ ⌂ × Cancel ✓ Save

Title\* **Proyecto Final**

States Layouts

No alarms found

Created time ↓ Originator Type Severity Status Assignee

Items per page: 10 1 – 0 of 0 | < < > > |

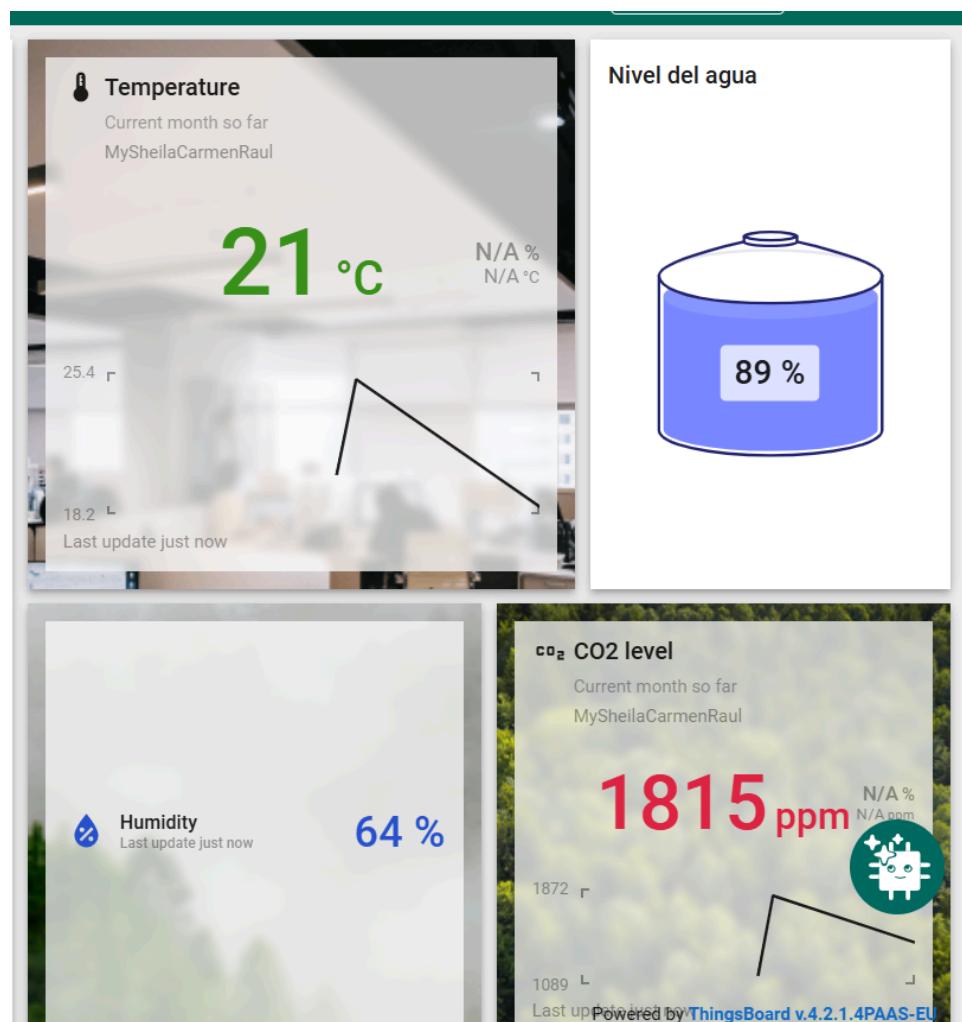
MySheilaCarmenRaul  
24 °c  
N/A % N/A °c  
25.4 F  
20 L Last update 1w ago

89 %  
Water tank icon

CO<sub>2</sub> level  
Current month so far MySheilaCarmenRaul  
704 ppm  
N/A % N/A ppm  
1872 F  
1089 L Last update 1w ago

Humidity 50 %  
Last update 1w ago

Powered by ThingsBoard v4.2.1.4PAAS-EU



## 4.2. Motor de Reglas y Configuración de Alarmas

Para el control histórico de incidencias, se ha integrado una tabla de alarmas (Alarm Widget) en el panel principal. Se han configurado 8 reglas de alarma distintas. Estas reglas monitorizan constantemente la telemetría entrante y determinan si los parámetros del invernadero están dentro de los rangos de seguridad establecidos.

Se ha diseñado una jerarquía de severidad para priorizar las incidencias:

### 4.2.1. Gestión Térmica Escalonada

Para la temperatura, se ha implementado un sistema de alerta escalonado, en las que el sistema discrimina entre desviaciones moderadas y peligrosas:

Rango de Advertencia (Warning):

- Regla "Cold" (Frío): Se activa si la temperatura es menor a 16 °C. Indica que el invernadero se está enfriando, pero no hay riesgo inmediato de helada.
- Regla "Hot" (Calor): Se activa si la temperatura es mayor a 26 °C, que sugiere la necesidad de ventilación pasiva.

The screenshot shows the configuration interface for creating alarm rules. At the top, it says 'Alarm type\*' and 'Cold'. Below that, there's a dropdown for 'Severity' set to 'Warning'. The first rule is for 'Cold' conditions: 'Condition: temperature less than 16' with a green status icon. It has a schedule 'Active all the time' and is linked to a 'Mobile dashboard' named 'Proyecto Final'. A note below says 'Used by mobile application as an alarm details dashboard'. The second rule is for 'Hot' conditions: 'Condition: temperature greater or equal 16' with a green status icon. It also has a schedule 'Active all the time'. A small circular icon with a gear and a plus sign is visible on the right side of the interface.

- Alerta de Calor Extremo (Extreme Hot): Si la temperatura supera los 30°C, el sistema eleva la severidad a Critical, indicando riesgo inminente para el cultivo.

Extreme Hot

Advanced settings ▾

Create alarm rules

Severity: Critical

Condition: temperature greater than 30

Schedule: Active all the time

Mobile dashboard: Proyecto Final

Used by mobile application as an alarm details dashboard

Clear alarm rule

Condition: temperature less or equal 30

Schedule: Active all the time

Gear icon

#### 4.2.2. Gestión Crítica de Riego

La regla de nivel de agua (Water Level) es fundamental para la protección de la bomba de riego. La alarma Critical salta cuando el nivel baja del 25%.

Limpieza (Clear Condition): la alarma no se desactiva hasta que el nivel supera el 50%. Esto evita que la alarma parpadee (se active y desactive continuamente) si el agua se mueve o si el llenado es insuficiente.

Water Level

Advanced settings ▾

Create alarm rules

Severity: Critical

Condition: water\_level\_percent less than 25

Schedule: Active all the time

Mobile dashboard: Proyecto Final

Used by mobile application as an alarm details dashboard

Clear alarm rule

Condition: water\_level\_percent greater than 50

Schedule: Active all the time

Gear icon

#### 4.2.3. Gestión umbrales de calidad del Aire (CO2 y Humedad)

La monitorización ambiental no se limita a la temperatura; el control preciso del CO2 y la humedad es determinante para la salud del cultivo.

##### A) Gestión Crítica de CO2

El dióxido de carbono es el "alimento" principal de las plantas. Las alarmas asociadas se han clasificado como Critical ya que, una desviación extrema detiene la fotosíntesis o indica un fallo grave de ventilación.

##### Nivel de CO2 bajo (Low CO2)

Si el nivel desciende por debajo de 200 ppm (el aire exterior suele tener ~400 ppm), la fotosíntesis se detiene por inanición de carbono. Esta alarma indica la urgencia de inyectar aire fresco.

Alarm type\*  
Low CO2

Advanced settings ▾

Create alarm rules

Severity  
Critical

Condition: co2 less than 200

Schedule: Active all the time

Mobile dashboard: Proyecto Final

Used by mobile application as an alarm details dashboard

Clear alarm rule

Condition: co2 greater or equal 200

Schedule: Active all the time

##### Nivel de CO2 alto (High CO2):

Niveles superiores a 1300 ppm, aunque no sean letales de inmediato, indican una acumulación excesiva de gases y falta de renovación de aire, lo cual es perjudicial para los cultivos pero también para el operario humano dentro del recinto.

Alarm type\*  
High CO2

Advanced settings ▾

Create alarm rules

Severity Critical	Condition: co2 greater than <b>1300</b>
	Schedule: Active all the time
	Mobile dashboard: <a href="#">Proyecto Final</a> <small>Used by mobile application as an alarm details dashboard</small>

Clear alarm rule

Condition: co2 less than <b>1300</b>
Schedule: Active all the time



## B) Gestión de Humedad Relativa

A diferencia del CO2 o la falta de agua, los problemas de humedad suelen causar daños a medio plazo por lo que se ha optado por una clasificación de severidad Minor (Menor). Esto evita saturar al con alertas críticas por eventos que no requieren acción inmediata instantánea.

Se monitorizan los extremos para evitar hongos ( $> 80\%$ ) o deshidratación ( $< 20\%$ ).

Alarm type\*  
High Humidity

Advanced settings ▾

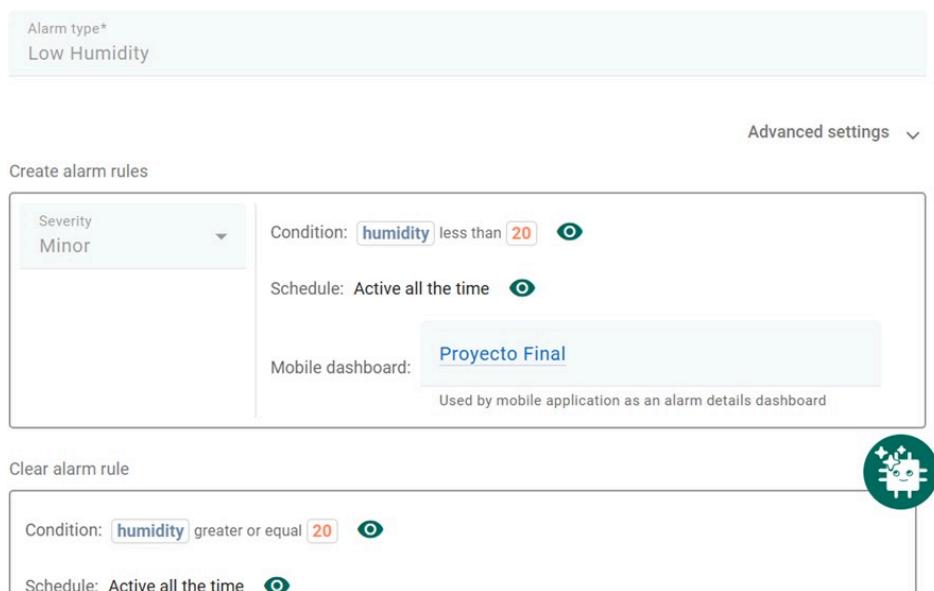
Create alarm rules

Severity Minor	Condition: humidity greater than <b>80</b>
	Schedule: Active all the time
	Mobile dashboard: <a href="#">Proyecto Final</a> <small>Used by mobile application as an alarm details dashboard</small>

Clear alarm rule

Condition: humidity less than <b>80</b>
Schedule: Active all the time





## 4.3. Automatización y Respuesta Activa (Rule Chains)

Una vez definidas las condiciones de alarma, es necesario establecer un mecanismo que cierre el ciclo de control, transformando una detección lógica (Alarma) en una acción física (Actuador). Para ello, hemos modificado la "Root Rule Chain" de ThingsBoard, implementando un flujo de automatización personalizado.

### 4.3.1. Diagrama de Flujo (Root Rule Chain)

Como se observa en la figura de abajo, la lógica de procesamiento de datos en ThingsBoard se organiza mediante nodos interconectados. El diagrama muestra cómo el sistema discrimina entre datos rutinarios (telemetría) y eventos críticos (alarmas).

A continuación, se describen los nodos y rutas principales visibles en el esquema:

#### Punto de Entrada (Input)

Todos los mensajes enviados por el M5Stack (telemetría, atributos, eventos) ingresan al sistema por el nodo verde situado a la izquierda. Desde aquí, el flujo se bifurca para ser analizado.

#### Evaluación de Perfil (Device Profile Node)

Es el primer nodo de procesamiento real (color rojo). Su función es alinear los datos entrantes con las reglas de alarma definidas en el Perfil del Dispositivo (ver apartado 4.2).

Si la telemetría cumple una condición de alarma , este nodo emite un evento específico por su salida "Alarm Created".

## La Ruta de Automatización (Rama Inferior)

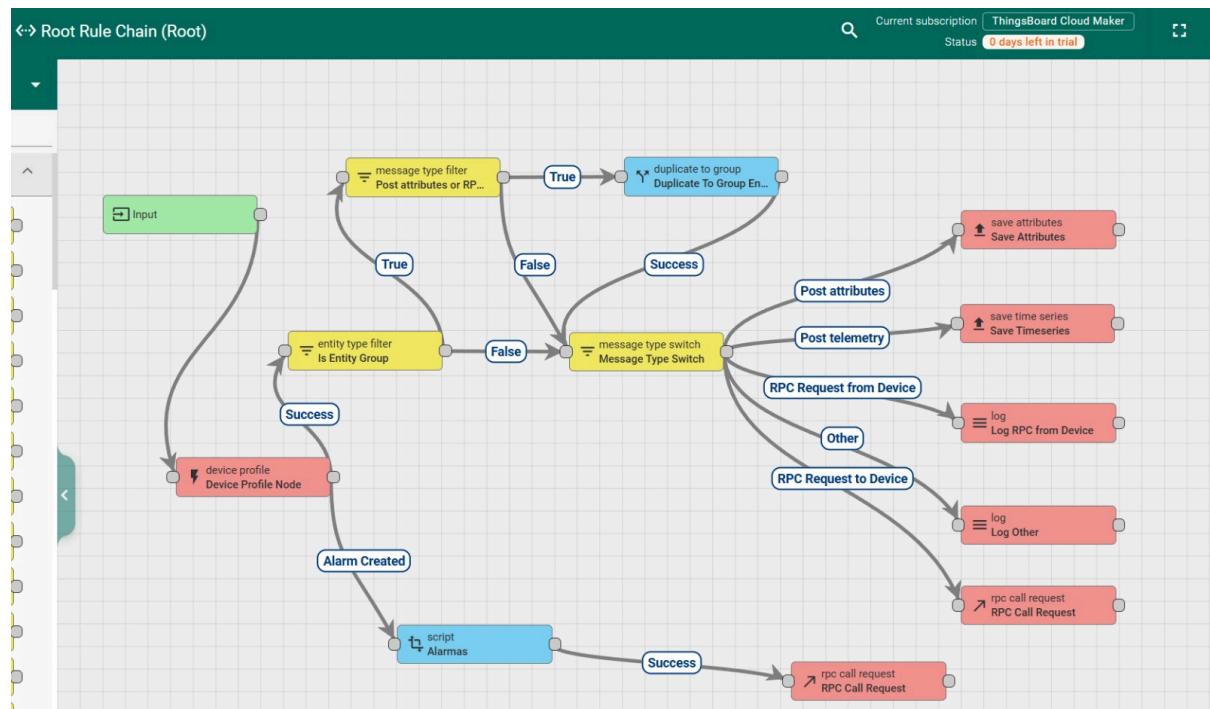
En lugar de limitarse a guardar la alarma, el sistema captura el evento de creación y lo deriva hacia una lógica de respuesta activa:

- Nodo de Transformación (script Alarmas): Este nodo azul contiene el código (que veremos en el siguiente apartado) encargado de traducir el nombre de la alarma (ej. "High CO2") al método RPC específico que el dispositivo entiende (ej. alerta\_co2\_alta). Actúa como un puente semántico entre la nube y el hardware.
- Nodo de Ejecución (RPC Call Request): El nodo final de esta rama (color rojo) toma el mensaje transformado y lo inyecta en el canal MQTT hacia el dispositivo, cerrando el bucle de control.

## La Ruta de Persistencia (Rama Central/Derecha)

Si el mensaje no genera una alarma, el flujo continúa hacia el nodo amarillo Message Type Switch. Este nodo , distribuyendo los datos según su naturaleza para su almacenamiento:

- Post telemetry Save Timeseries: Si son datos de sensores, se envían al nodo de guardado de series temporales para generar las gráficas históricas.
- Post attributes Save Attributes: Si son atributos de estado, se actualiza la base de datos del dispositivo.
- RPC Request from Device Log RPC: Si son respuestas a comandos, se registran en el log de auditoría.



#### 4.3.1. Transformación de Mensajes (Script Node)

Cuando se genera una alarma ("High CO2"), el motor de reglas invoca un nodo de transformación (Script Node)

La función de este script es actuar como un traductor que recibe el nombre de la alarma y genera un mensaje JSON con el método RPC específico que el firmware del M5Stack sabe interpretar.

A continuación, se muestra el código implementado para el mapeo de eventos:

```
// Estructura base del mensaje RPC
var newMsg = {
    method: "",
    params: {}
};

// Mapeo lógico: Nombre de Alarma -> Método del Dispositivo
if (msg.name == "Extreme Hot") {
    newMsg.method = "alerta_temp_critica";
}
else if (msg.name == "Hot") {
    newMsg.method = "alerta_temp_alta";
}
else if (msg.name == "Cold") {
    newMsg.method = "alerta_temp_baja";
}
else if (msg.name == "High Humidity") {
    newMsg.method = "alerta_hum_alta";
}
else if (msg.name == "Low Humidity") {
    newMsg.method = "alerta_hum_baja";
}
else if (msg.name == "High CO2") {
    newMsg.method = "alerta_co2_alta";
}
else if (msg.name == "Low CO2") {
    newMsg.method = "alerta_co2_baja";
}
else if (msg.name == "Water Level") {
    newMsg.method = "alerta_agua_baja";
}

// Retorno del mensaje formateado para el siguiente nodo
return {msg: newMsg, metadata: metadata, msgType: msgType};
```

Si en el futuro queremos cambiar cómo reacciona el invernadero ante el calor u otra alarma, solo tendríamos que modificar este script en la nube sin reprogramar el dispositivo físico.

Este script está en el siguiente apartado de ThingsBoard

**Alarms**  
Transformation - script

Details    Events    Help

Name\*  
Alarms



```
function Transform(msg, metadata, msgType) {  
    1 var newMsg = {  
    2     method: "",  
    3     params: {}  
    4 };  
    5  
    6  
    7 if (msg.name == "Extreme Hot") {  
    8     newMsg.method = "alerta_temp_critica";  
    9 }  
10 else if (msg.name == "Hot") {  
11     newMsg.method = "alerta_temp_alta";  
12 }  
13 }  
14  
15 }
```

#### 4.3.2. Ejecución y Persistencia (Nodo RPC Request)

El eslabón final de la cadena de automatización es el nodo "RPC Call Request". Este componente es el encargado de injectar el comando JSON generado por el sistema en la cola de mensajes MQTT del dispositivo.

Como se detalla en la figura de abajo, se ha realizado una configuración crítica en este nodo estableciendo el parámetro "Timeout in seconds" en 60.

**RPC Call Request**  
Action - rpc call request

Details    Events    Help

Name\*  
RPC Call Request

Timeout in seconds\*  
60

ess  
ess

Rule node description

Al establecer esta ventana de 60 segundos, instruimos al servidor ThingsBoard para que encole el mensaje. Si el dispositivo pierde la conexión pero se reconecta dentro del minuto siguiente, recibirá inmediatamente los comandos pendientes. Esto garantiza la integridad del sistema de alertas ante fallos de red transitorios.

#### 4.4. Validación y registro de eventos

Para verificar la fiabilidad del sistema, hemos realizado una serie de pruebas, forzando las condiciones físicas de disparo de cada alarma (cabe destacar que es ha resultado complejo hacer saltar algunas alarmas pues, no estamos en un escenario de invernadero con condiciones de temperatura, humedad y CO<sub>2</sub> reales y útiles).

Como podemos observar, el sistema ha sido capaz de detectar, clasificar y registrar múltiples incidencias consecutivas con precisión de segundos

Alarms						
🕒 Realtime - last day						
	Created time ↓	Originator	Type	Severity	Status	Assignee
<input type="checkbox"/>	2025-12-19 16:01:55	MySheilaCarmenRaul	High CO <sub>2</sub>	Critical	Active Unacknowledged	Una... ▾  ⏱  ✓
<input type="checkbox"/>	2025-12-19 15:58:44	MySheilaCarmenRaul	Extreme Hot	Critical	Cleared Acknowledged	Una... ▾  ⏱  ✓
<input type="checkbox"/>	2025-12-19 15:58:39	MySheilaCarmenRaul	Hot	Warning	Cleared Unacknowledged	Una... ▾  ⏱  ✓
<input type="checkbox"/>	2025-12-19 15:58:34	MySheilaCarmenRaul	High CO <sub>2</sub>	Critical	Cleared Unacknowledged	Una... ▾  ⏱  ✓
<input type="checkbox"/>	2025-12-19 15:58:28	MySheilaCarmenRaul	Low CO <sub>2</sub>	Critical	Cleared Unacknowledged	Una... ▾  ⏱  ✓
<input type="checkbox"/>	2025-12-19 15:54:45	MySheilaCarmenRaul	High Humidity	Minor	Cleared Unacknowledged	Una... ▾  ⏱  ✓
<input type="checkbox"/>	2025-12-19 15:54:07	MySheilaCarmenRaul	Water Level	Critical	Cleared Unacknowledged	Una... ▾  ⏱  ✓
<input type="checkbox"/>	2025-12-19 15:42:53	MySheilaCarmenRaul	Water Level	Critical	Cleared Acknowledged	Una... ▾  ⏱  ✓
<input type="checkbox"/>	2025-12-19 15:42:21	MySheilaCarmenRaul	High CO <sub>2</sub>	Critical	Cleared Acknowledged	Una... ▾  ⏱  ✓

#### Análisis de los Resultados:

1. Diversidad de Detección: El log muestra que el sistema ha gestionado correctamente alertas simultáneas de distinta naturaleza: Hídricas ("Water Level"), Térmicas ("Extreme Hot", "Hot") y de Calidad del Aire ("High CO<sub>2</sub>", "High Humidity").

2. Clasificación de Severidad: Se valida la correcta implementación de la jerarquía de riesgos diseñada, distinguiendo entre eventos críticos (en Rojo, ej: Extreme Hot a las 15:58:44) y advertencias menores (en Naranja/Amarillo, ej: High Humidity a las 15:54:45).
3. Ciclo de Vida del Evento (Cleared): Es importante destacar la columna "Status". La mayoría de alarmas aparecen como "Cleared" (Limpadas). Esto confirma que la lógica de recuperación funciona: cuando el sensor detectó que el nivel de agua o la temperatura volvieron a la normalidad, el sistema cerró la incidencia automáticamente, dejando de enviar alertas al dispositivo pero manteniendo el registro para futuras auditorías.
4. Interacción del Operario: Algunos eventos aparecen marcados como "Acknowledged" (Reconocidos), demostrando que la interfaz permite al usuario confirmar que ha visto la incidencia, una función clave para la gestión de equipos de mantenimiento.

Se adjunta evidencia gráfica de la respuesta física del prototipo ante situaciones de emergencia.

Como se observa en las imágenes de abajo, el dispositivo ha reaccionado correctamente a diferentes condiciones. Se validan visualmente que el ciclo rotativo de datos se detiene para mostrar el mensaje y que la tira LED lateral se ilumina en color alertando visualmente al usuario.

Dado que las alertas implementadas (especialmente las de CO2 y Nivel de Agua) incluyen patrones dinámicos de parpadeo y códigos sonoros específicos que no pueden apreciarse en una fotografía estática, el funcionamiento completo y la respuesta en tiempo real se detallan en un vídeo.

