

# Automatización de la clase

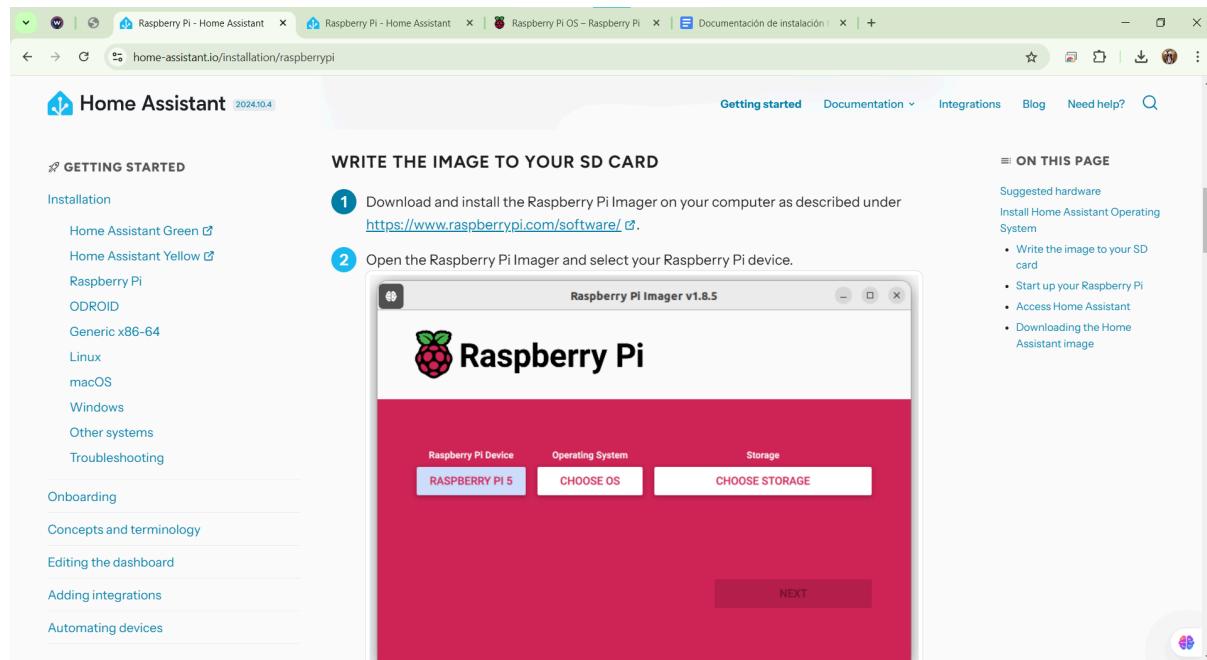


# Instalación en Raspberry Pi 5

## paso 1:

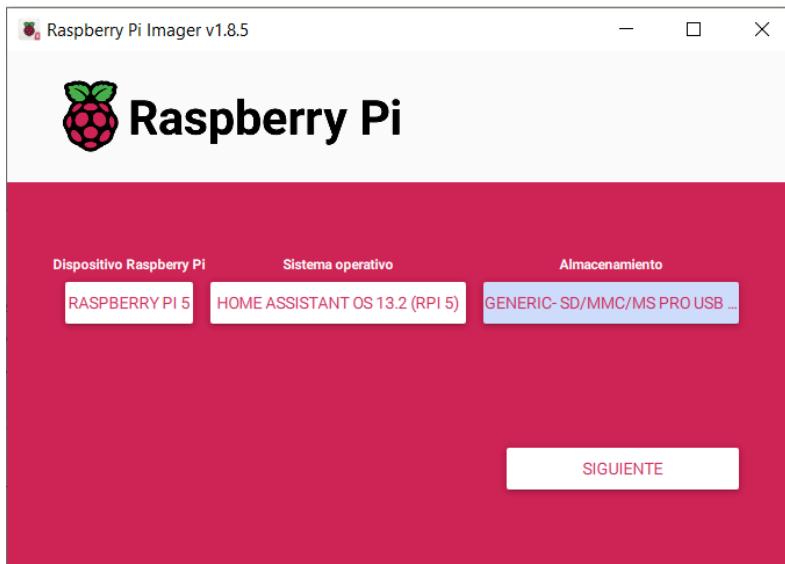
Desde la página oficial de home assistant buscamos cómo instalar el programa en nuestra Raspberry Pi 5. Nada más entrar en la siguiente URL podemos acceder a una guía. En ella podremos descargarnos en nuestro ordenador el programa que necesitamos.

<https://www.home-assistant.io/installation/raspberrypi/>



## paso 2:

Más tarde nos instalamos la imagen que se descarga y se nos abre el programa. Esto es para instalar la propia imagen en una micro-SD la cuál introduciremos dentro de la Raspberry Pi.

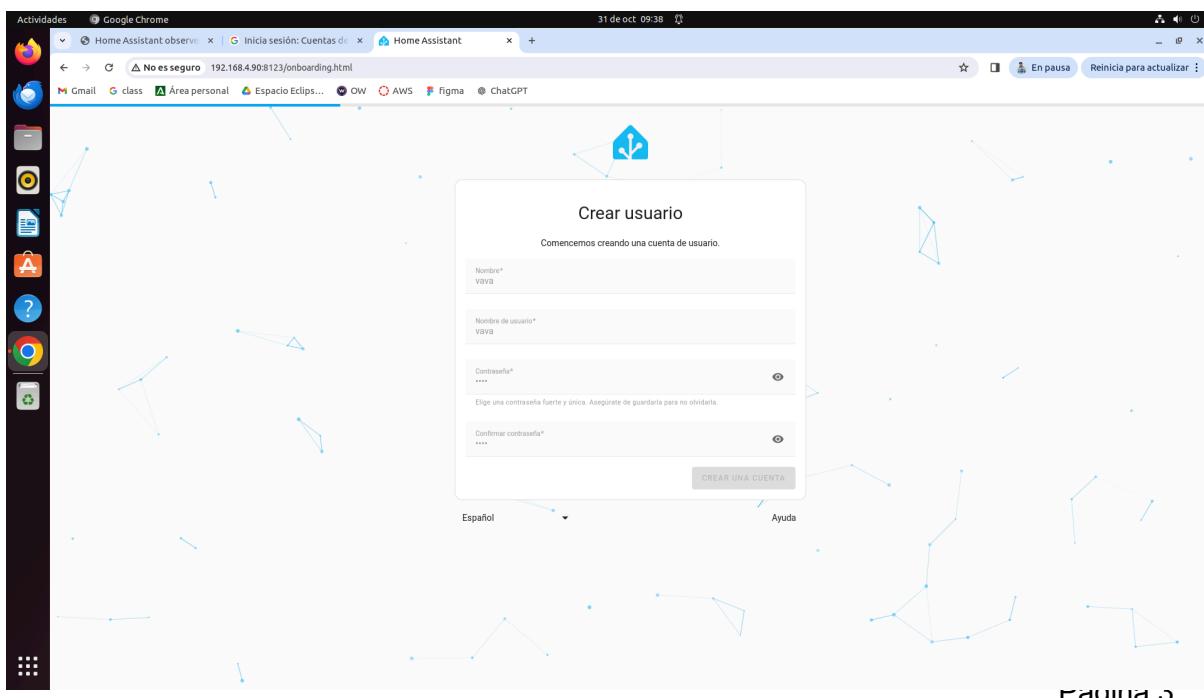


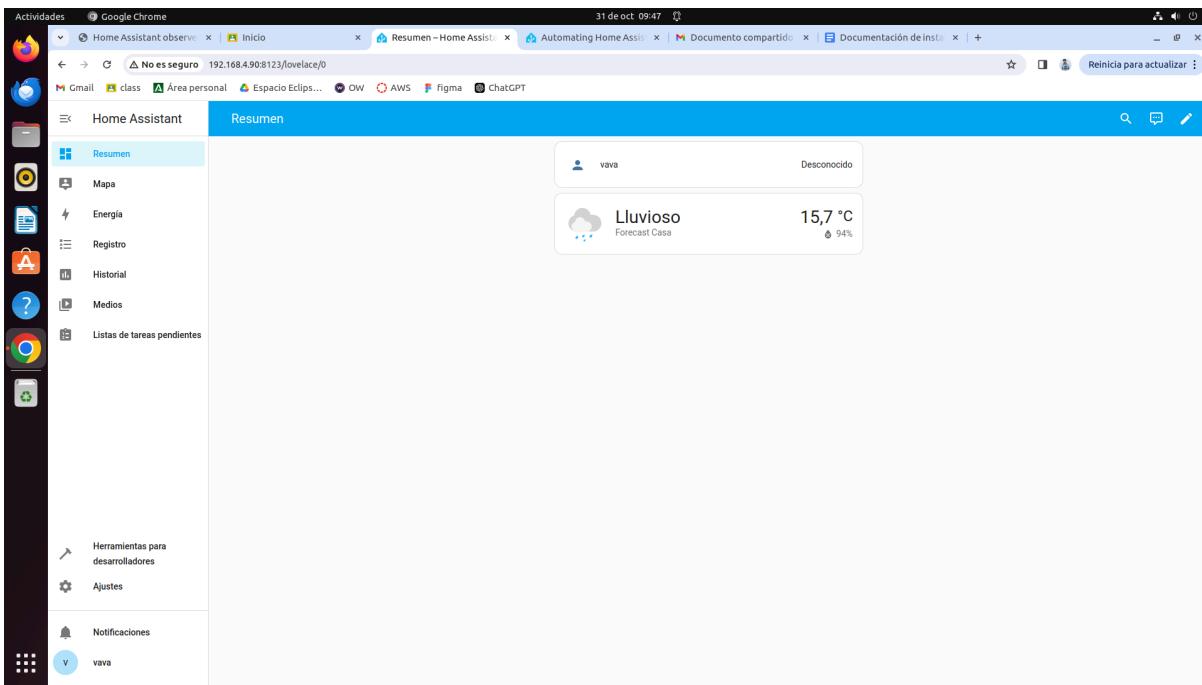
Una vez instalada nos saldrá el siguiente mensaje, el cual dirá que ya está instalada en la tarjeta.



## paso 3:

Una vez metamos la tarjeta SD a la raspberry, la conectaremos a la corriente eléctrica y le daremos conexión vía Ethernet. Para entrar dentro de la Raspberry necesitamos su dirección ip para evitar conectarnos a otra placa. Una vez hecho eso creamos una cuenta.

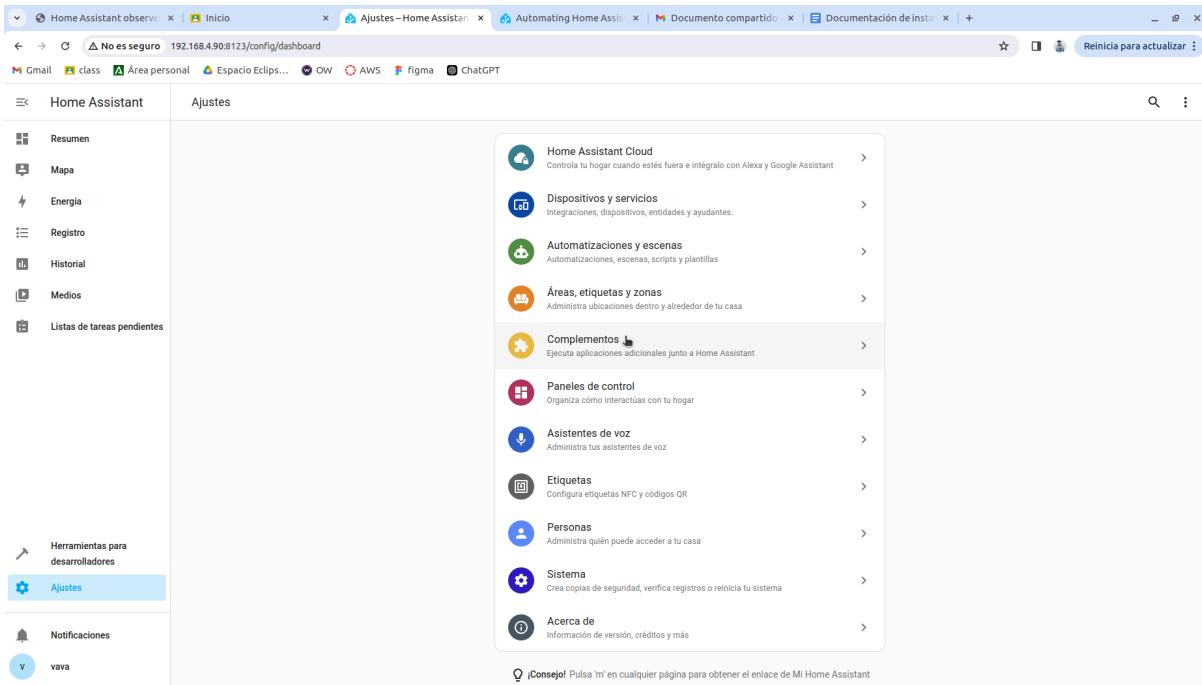




## Instalación de Complementos

### paso 1:

Una vez conectados procedemos a instalar todos los complementos necesarios desde el apartado de complementos.



## paso 2:

Para instalar los complementos se necesita ir a tienda de complementos:



## paso 3:

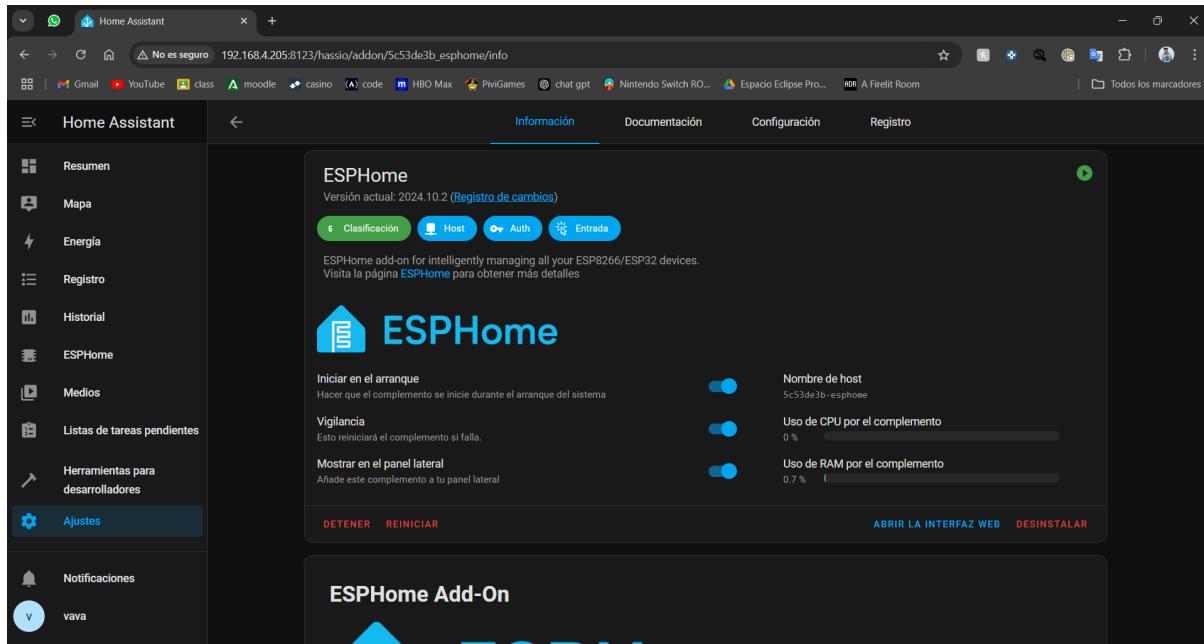
### ESP Home:

ESP Home es una herramienta que lee un fichero de configuración y crea un firmware personalizado que posteriormente puedes subir a tus ESP32 o ESP8266 con sus sensores conectados.

The screenshot shows the Home Assistant add-on store interface. On the left, there is a sidebar with various icons and sections like Resumen, Mapa, Energía, Registro, Historial, Medios, and Listas de tareas pendientes. The main content area is titled 'ESPHome' and shows the following details:

- Registro de cambios:** Shows a history of changes with a 'Clasificación' tab selected, along with Host, Auth, and Entrada tabs.
- Version:** 2024.10.2
- Description:** ESPHome add-on for intelligently managing all your ESP8266/ESP32 devices. It links to the official ESPHome website for more details.
- Install Button:** A large 'INSTALAR' button.
- ESPHome Add-On Section:** Displays the ESPHome logo, a statistics card (8.4k stars, 3.3k online), and an 'About' link.

Una vez buscada y descargada lo instalamos y ya la tenemos funcionando:

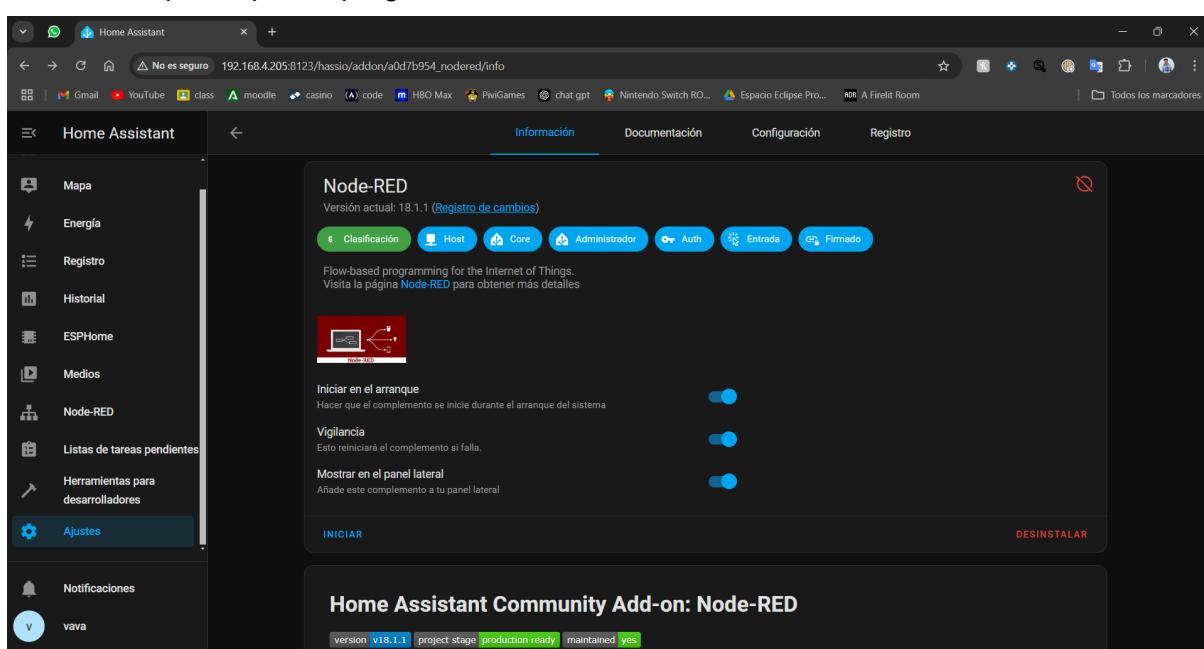


## Node\_Red:

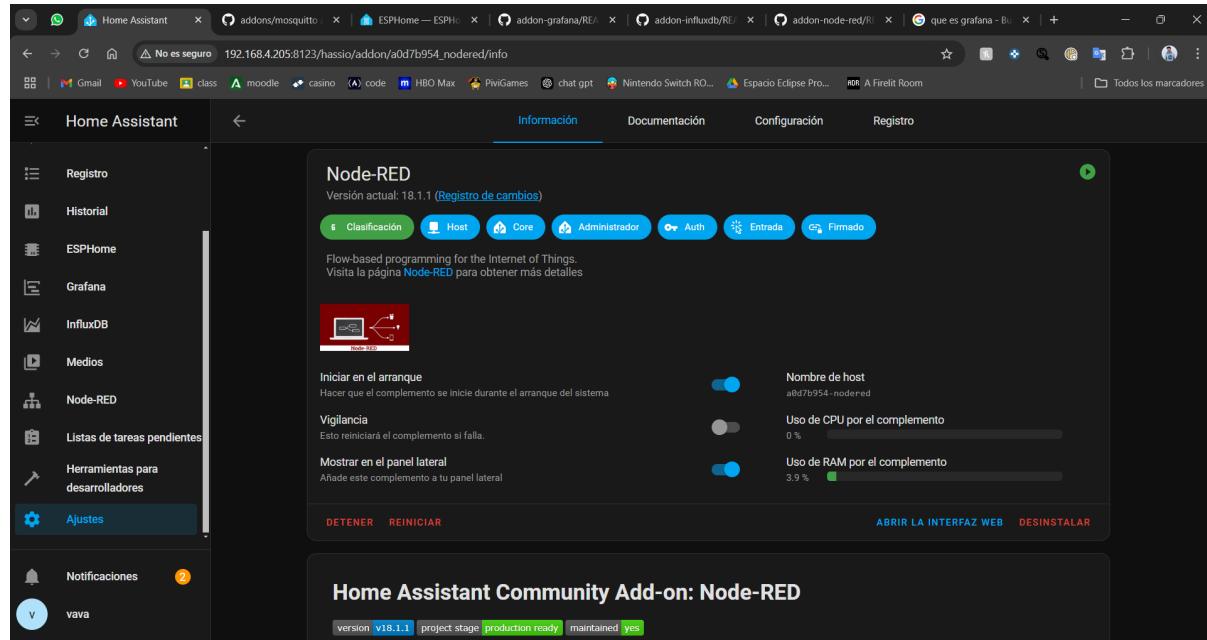
<https://docs.google.com/document/d/1dsLrLwtoHN4ku7yeuMV3IoMBnpVj5raWFAbTu19lbHE/edit?usp=sharing>

Proyecto domotizar la clase

Node-RED es una herramienta de programación visual que se implementa en dispositivos controladores de hardware. Trabaja mostrando de manera visual las relaciones y funciones de manera que se pueda programar sin escribir.

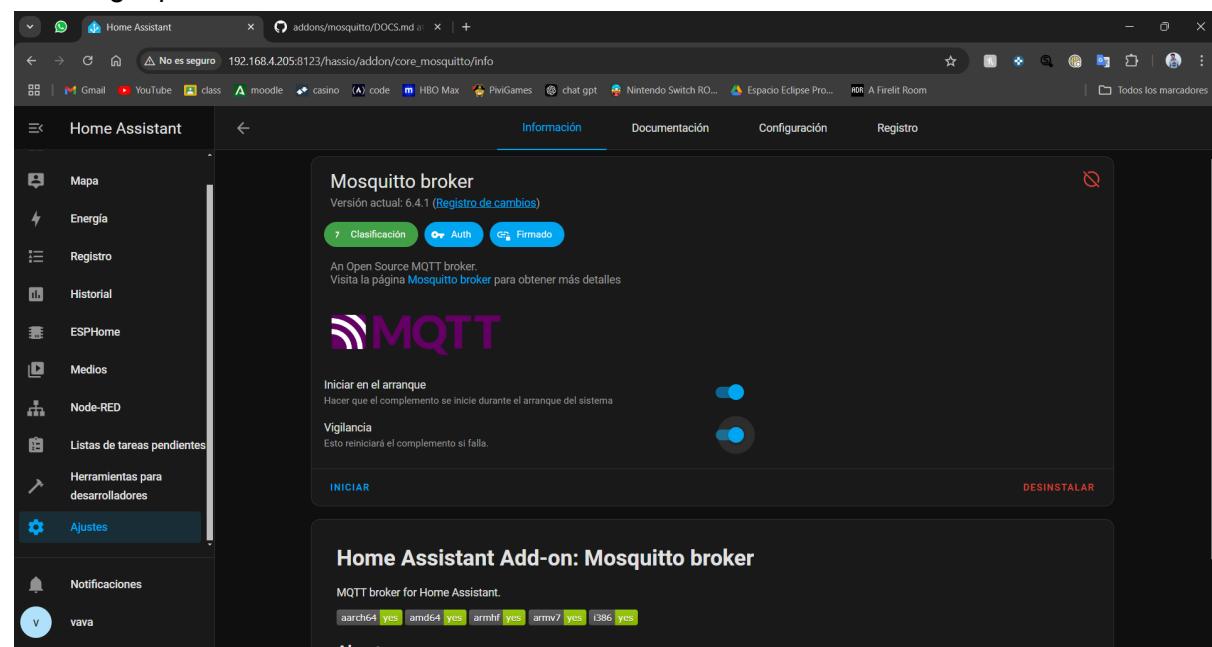


Tenía un problema al iniciar el Node-Red, después de varias investigaciones en línea hemos descubierto que el SSL de Node-Red no está bien configurado, así que de momento hemos optado por desactivarlo y hemos conseguido iniciararlo



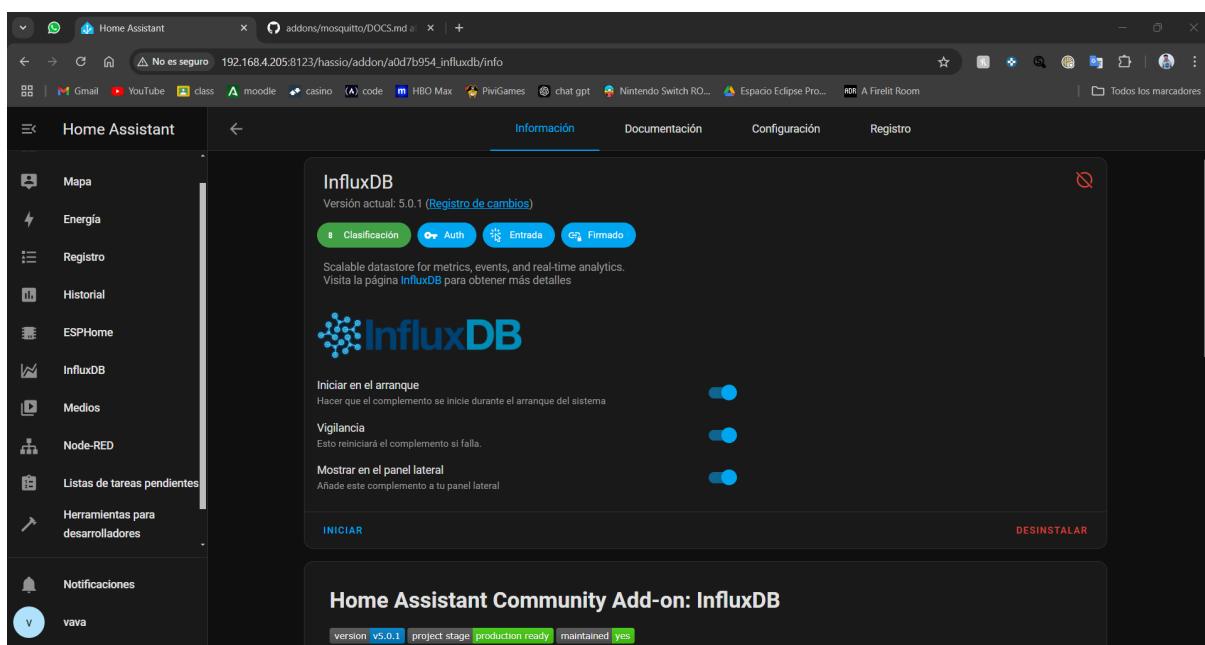
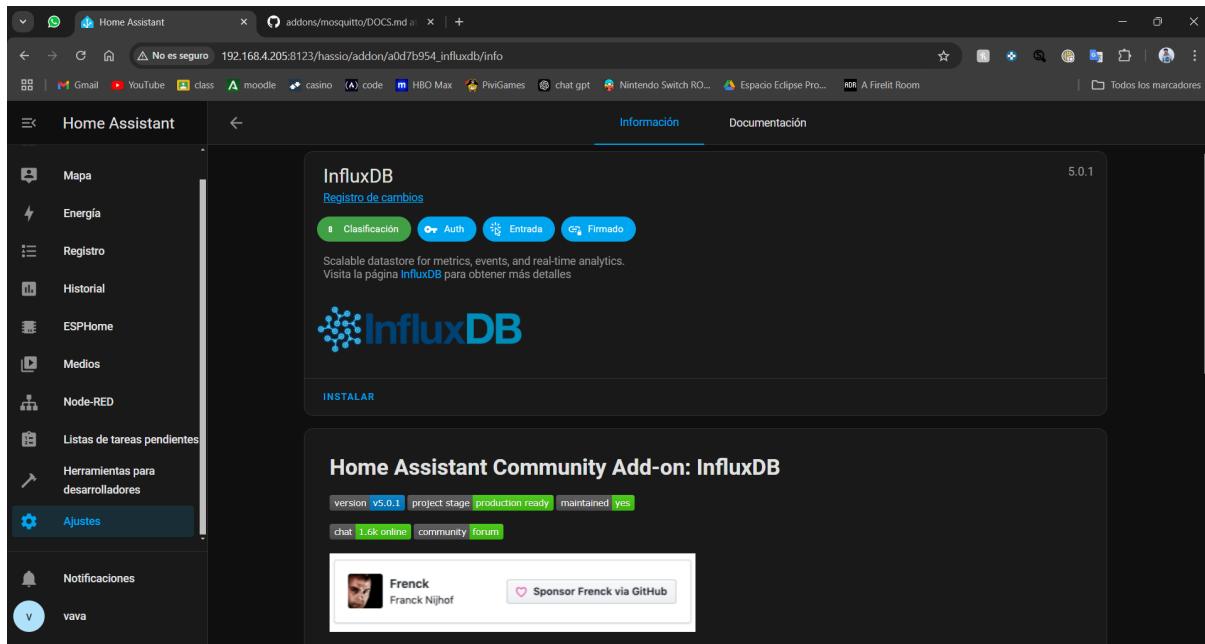
## Mosquitto broker:

Es un protocolo de mensajes para el Internet de las Cosas. Este protocolo permite enviar datos de una manera muy rápida y eficiente, para que todos los sensores y actuadores puedan comunicarse con la domótica del hogar de manera óptima, y con el menor consumo de energía posible.



## InfluxDB:

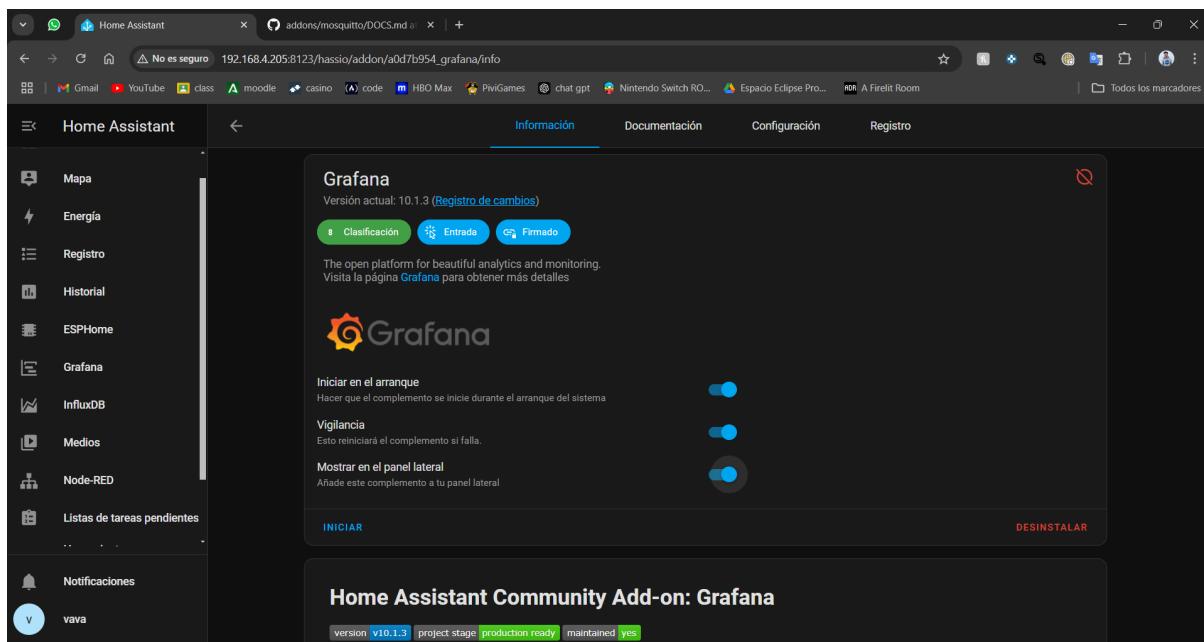
Generador de bases de datos mediante ficheros de distintos tipos



## Grafana:

Nos permite trabajar con grandes volúmenes de datos en tiempo real. Con Grafana es posible visualizar los análisis resultantes de la explotación de los datos en cuadros de mando intuitivos, obtener métricas, realizar consultas, monitorizar aplicaciones e infraestructuras hardware, y establecer alarmas parametrizables.

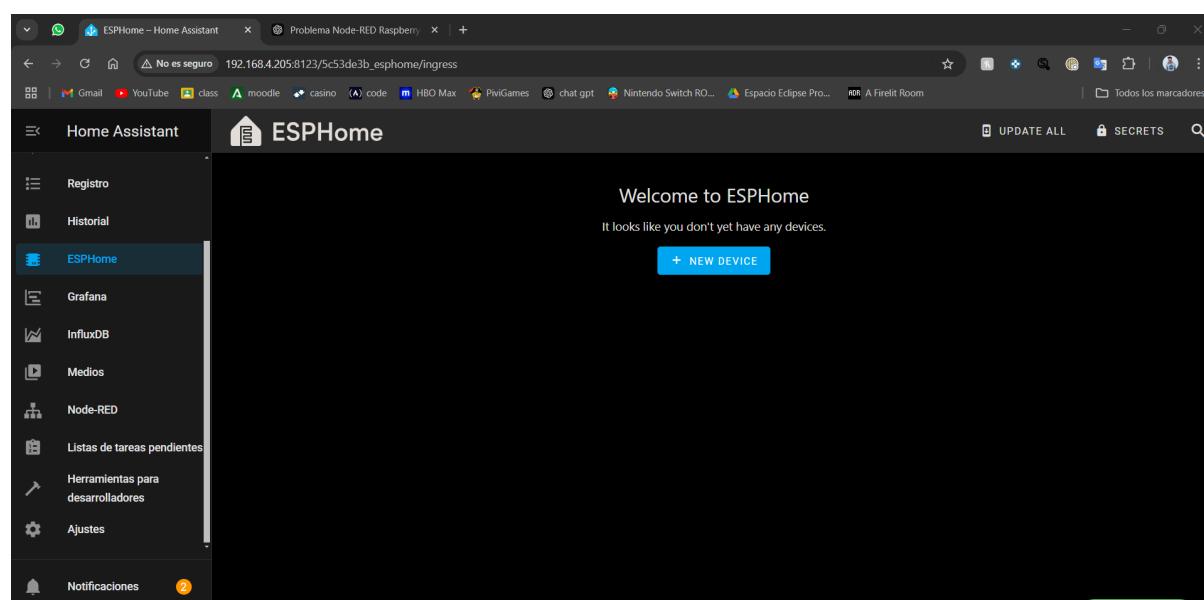
Se combina con InfluxDB

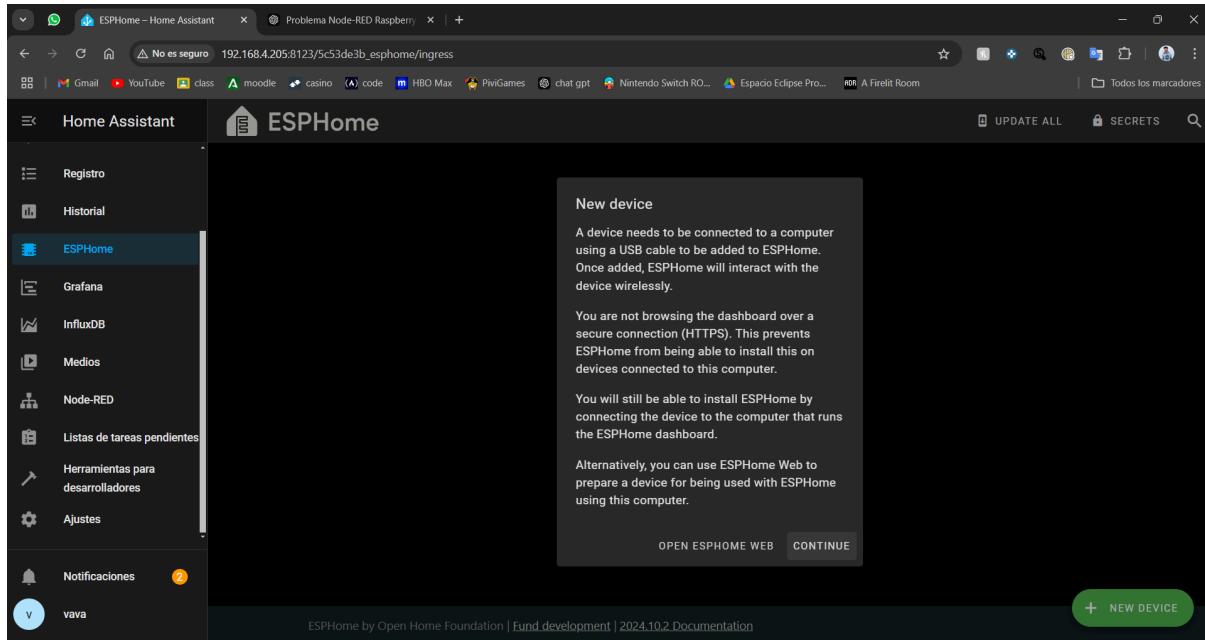


## CONECTAR ESP32 A ESP HOME

### paso 1:

Vamos a acceder al complemento llamado ESP Home:





## paso 2:

Después de introducir las credenciales wifi:

Nombre Wifi: 2DAW\_IoT

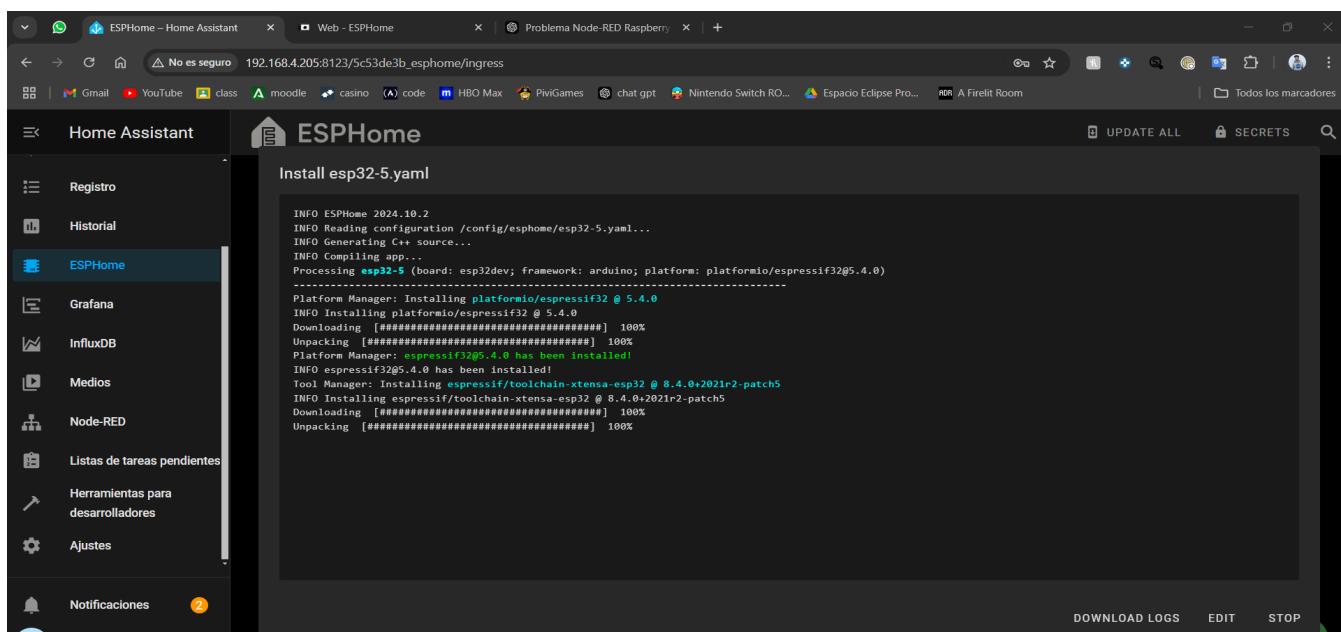
Contraseña: Somos2DAW

La clave es la siguiente:

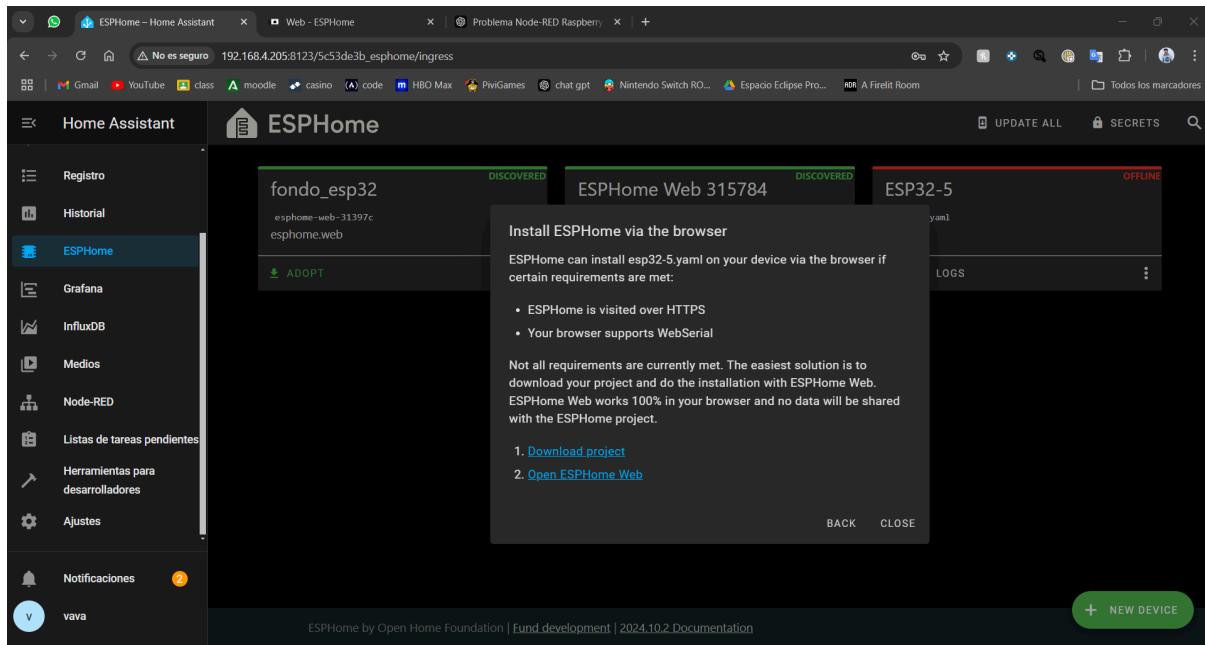
LXIdGwE1Ed9ydX32L50o0mlcF81njbT0JhUj0MQY9Uw=vlO7R0Z8JUIPg6v9S3V+hEm14n

hmBKmCEljDUx37o=

Hemos optado por la configuración inalámbrica y empezará a instalarse

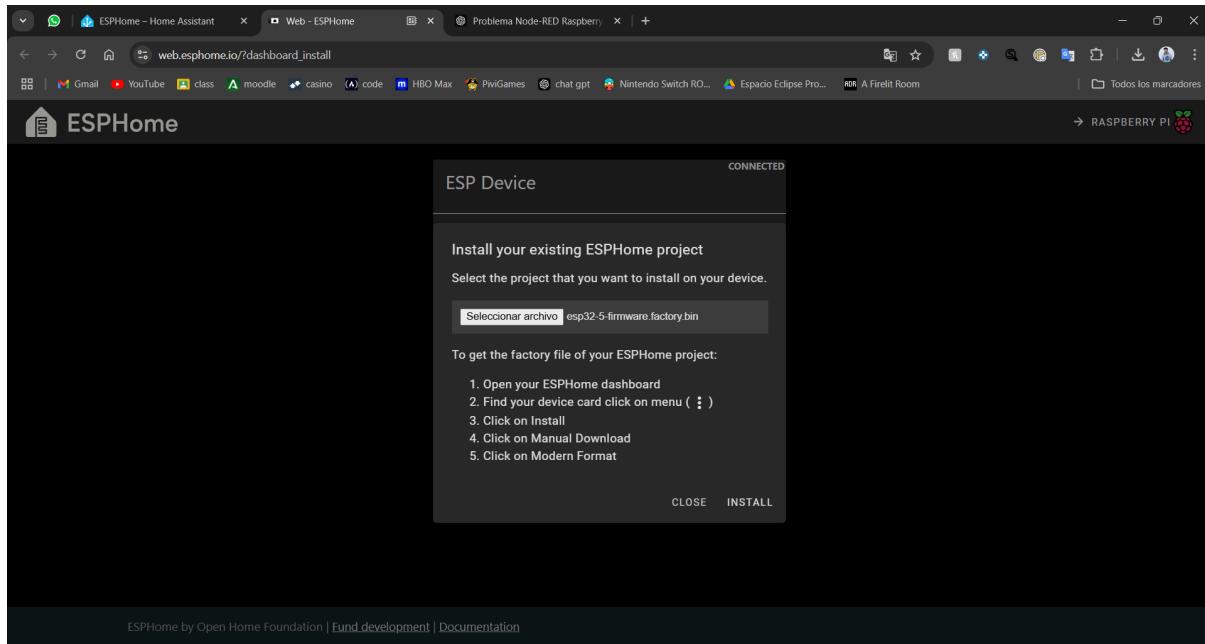


Ha salido un fallo en la IP, así que hemos optado por empezar de nuevo y hemos le hemos dado a instalar con el ordenador y nos sale esta pantalla:



## paso 3:

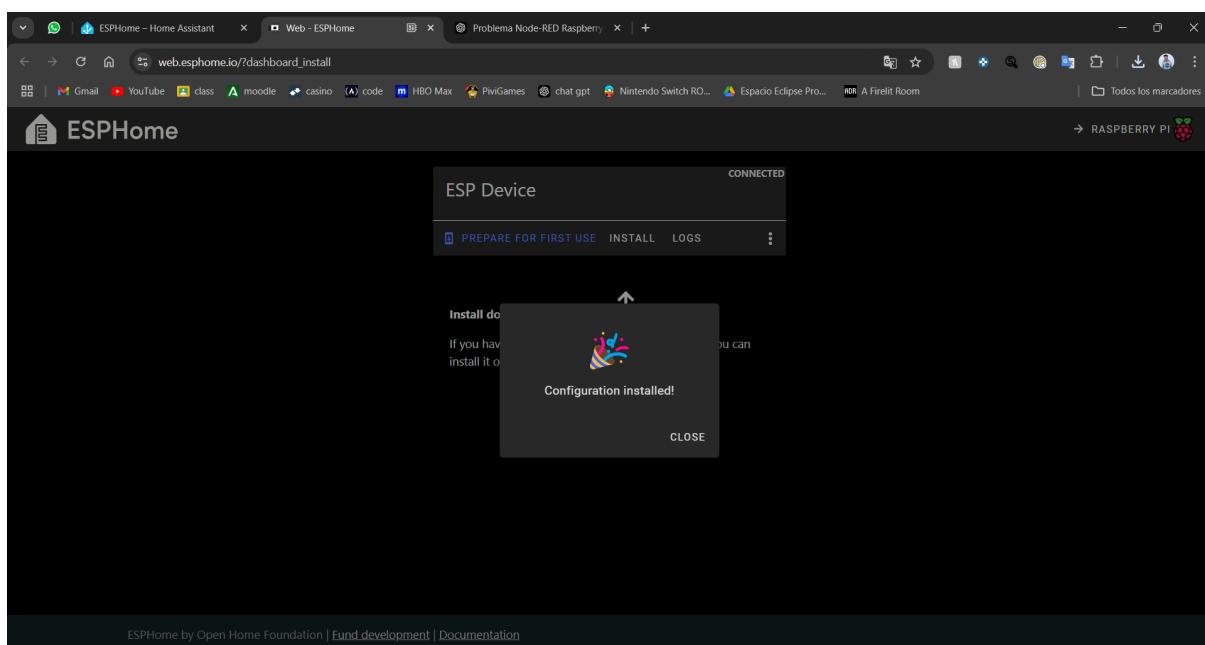
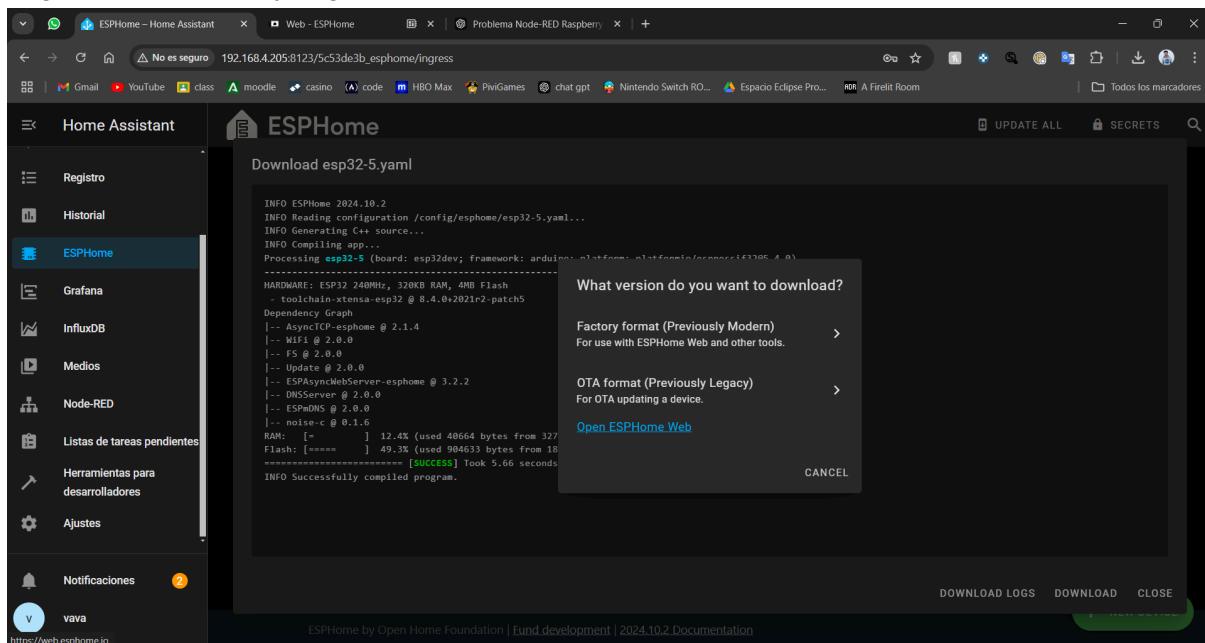
Abrimos el enlace y descargamos el proyecto:

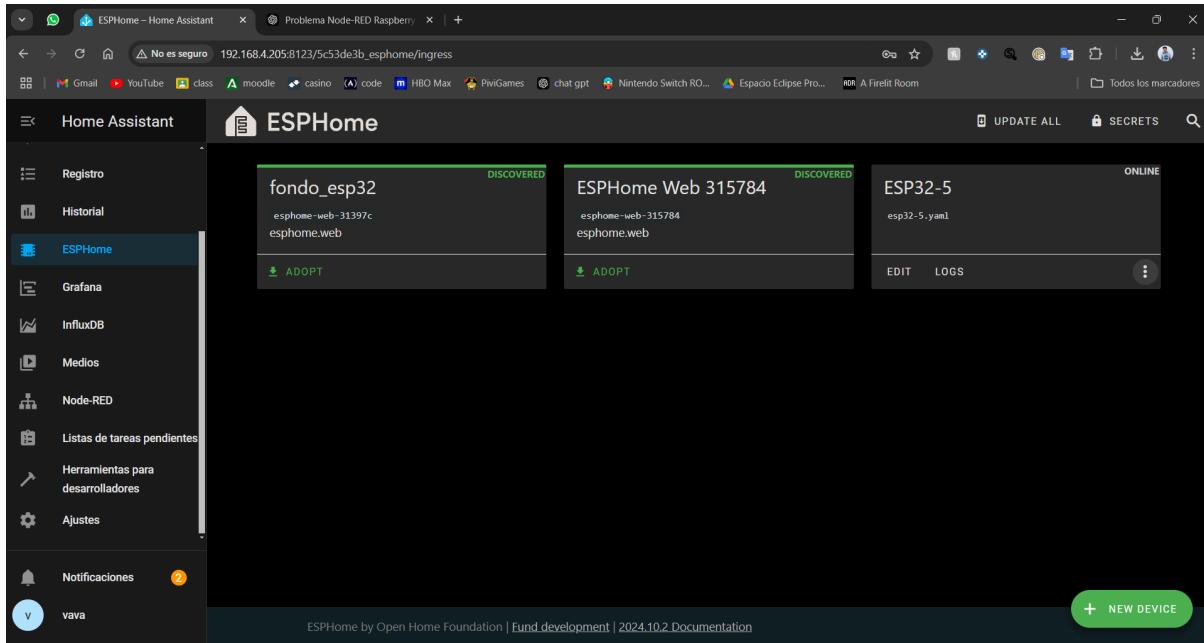


## paso 4:

En la página de ESP Home web conectamos mediante el puerto e instalamos el proyecto previamente instalado.

Seguimos los pasos y llegamos aquí





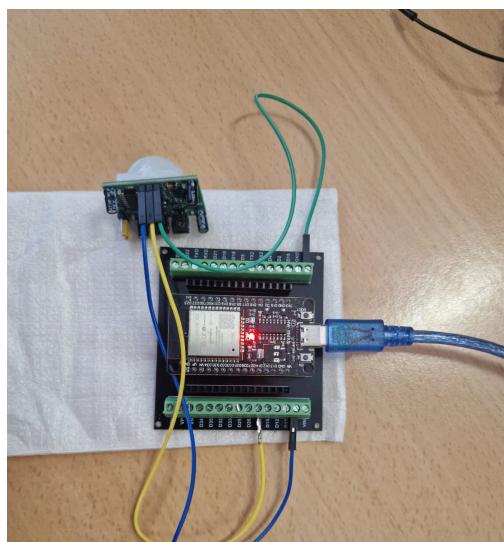
## Coneectar Sensores

### Sensor de Presencia (pir):

#### paso 1:

Vamos a realizar la instalación de un sensor pir el cuál detecta la presencia mediante infrarrojos emitidos por personas u objetos.

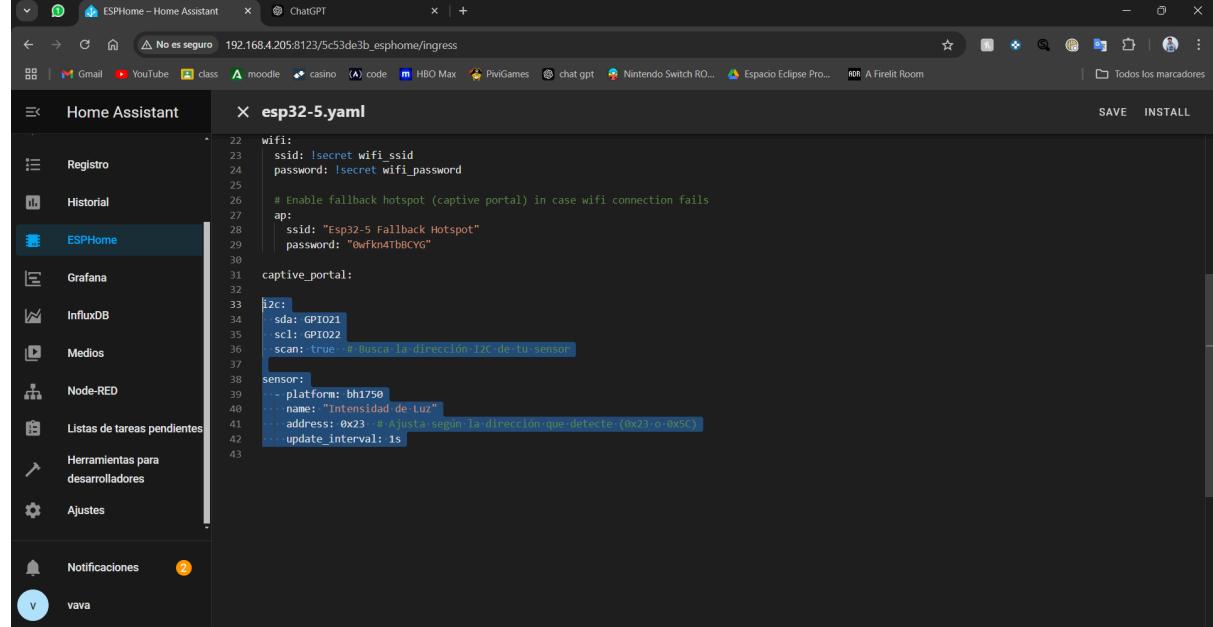
Las conexiones que debe de tener el sensor pir es el que se observa en la siguiente imagen



## paso 2:

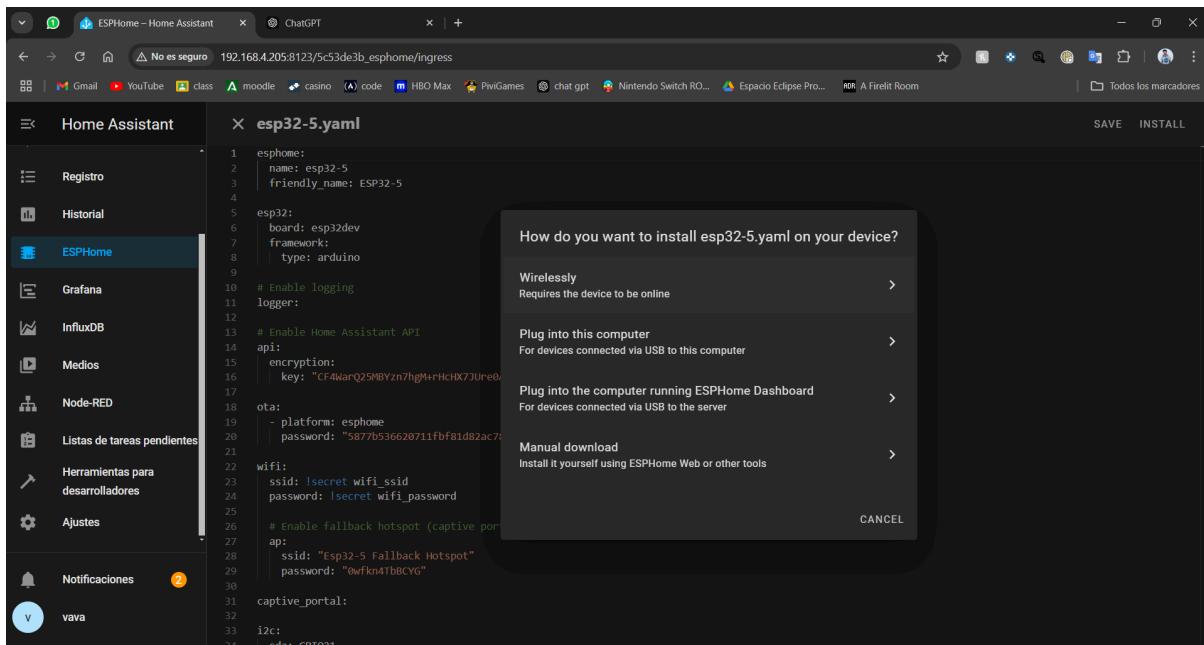
Modificamos el .yaml para que entienda que tiene que tener un sensor de presencia y en qué puerto lo guardamos.

```
i2c:  
  sda: GPIO21  
  scl: GPIO22  
  scan: True  
  
sensor:  
  - platform: bh1750  
    name: "Intensidad de Luz"  
    address: 0x23  
    update_interval:1s
```

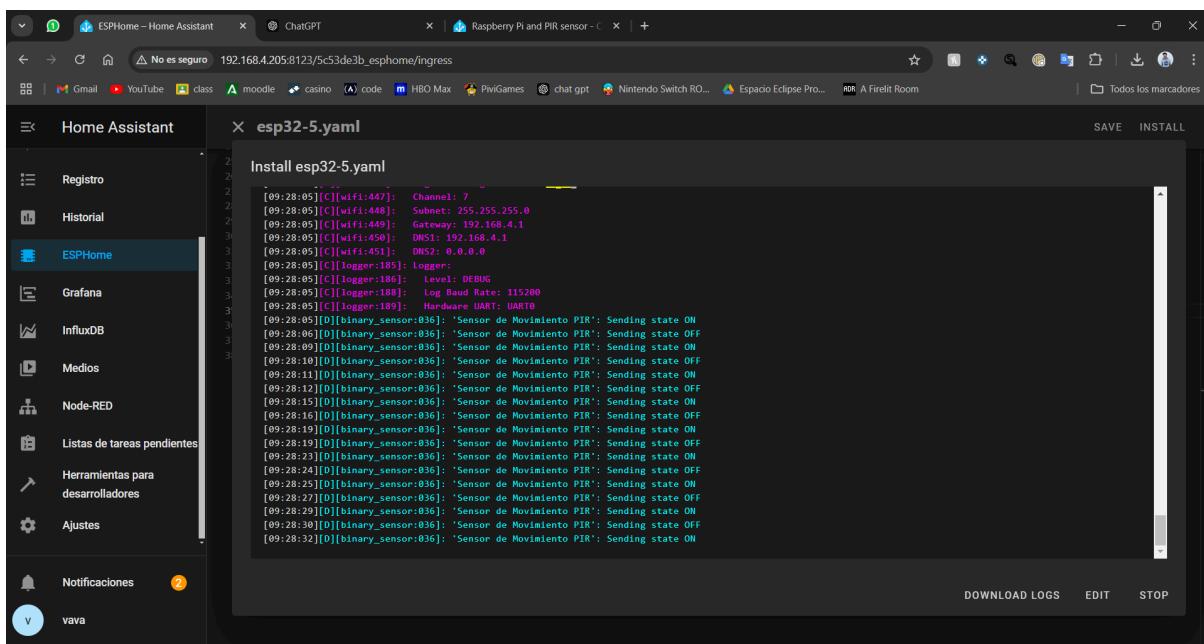


## paso 3:

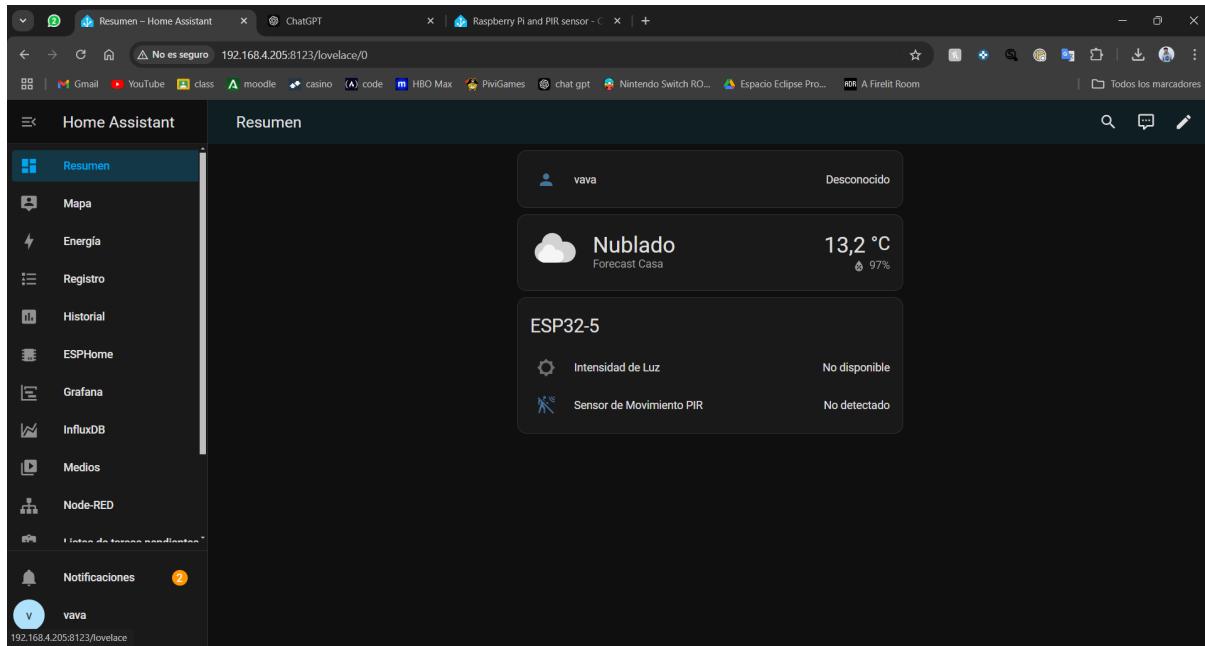
Le damos al método por el que queremos instalar el punto yaml.



En la siguiente imagen observamos el resultado de la instalación y de cómo se va instalando.



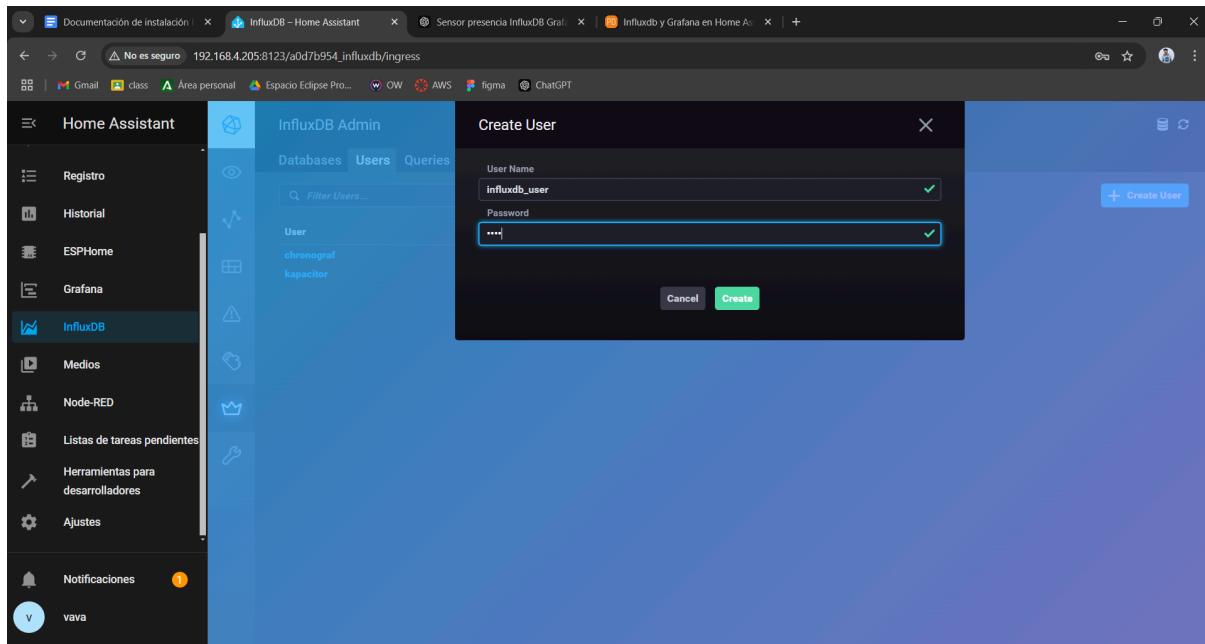
Una vez instalado vemos que ya nos sale en el panel de control.



## Mostrar datos del sensor pir

### paso 1:

Debemos iniciar en influxdb y creamos un usuario desde la parte de influxdb admin.



## paso 2:

Aquí vemos que tenemos el usuario creado anteriormente con permisos de lectura y escritura.

The screenshot shows the Home Assistant InfluxDB configuration page. On the left, there's a sidebar with various icons and links like Registro, Historial, ESPHome, Grafana, and InfluxDB. The InfluxDB section is currently selected. The main area is titled 'InfluxDB User' and shows a table for 'Database Permissions'. It lists a single user 'influxdb\_user' with 'ALL PRIVILEGES' granted. There are buttons for 'Change password', 'Revoke Admin', and 'Delete User' at the top right. A message at the bottom states: 'The user is an admin, ALL PRIVILEGES are granted irrespectively of database permissions.'

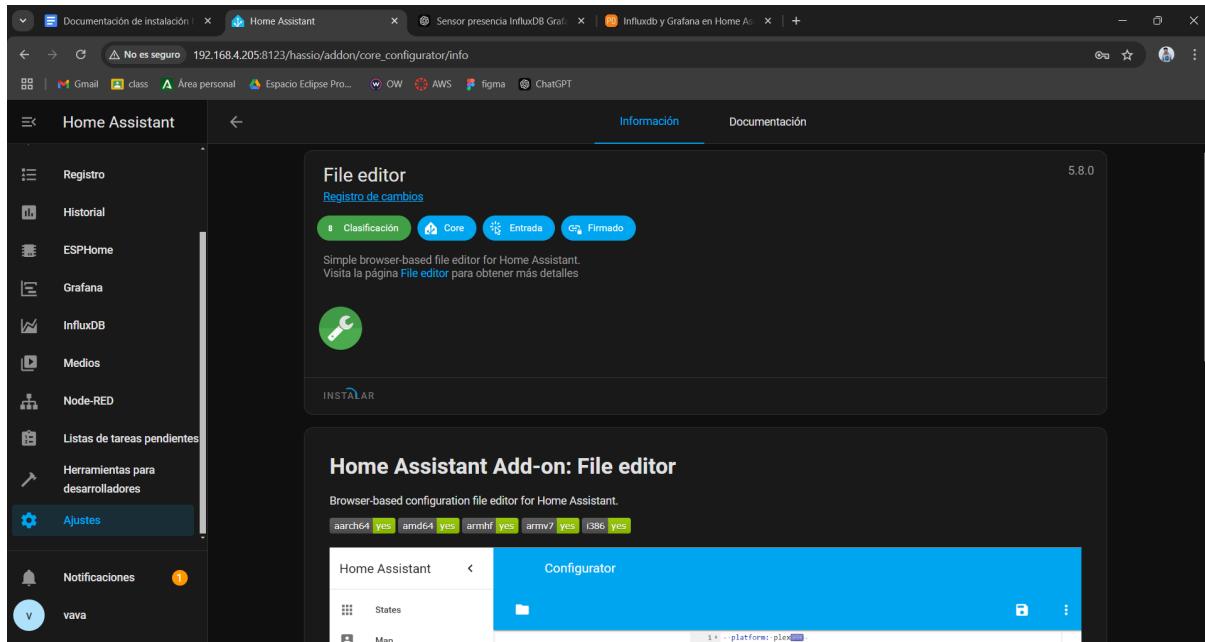
## paso 3:

Creamos la base de datos Prueba dentro del apartado Databases.

The screenshot shows the Home Assistant InfluxDB Admin interface. The sidebar is identical to the previous screen. The main area is titled 'InfluxDB Admin' and shows a table for 'Databases'. It lists two databases: '\_internal' and 'Prueba'. A success message 'Database created successfully.' is displayed above the 'Create Database' button. The 'Prueba' database has a retention policy for 'autogen' with 'default' settings and an infinite duration. The '\_internal' database has a retention policy for 'monitor' with 'default' settings and a 7-day duration.

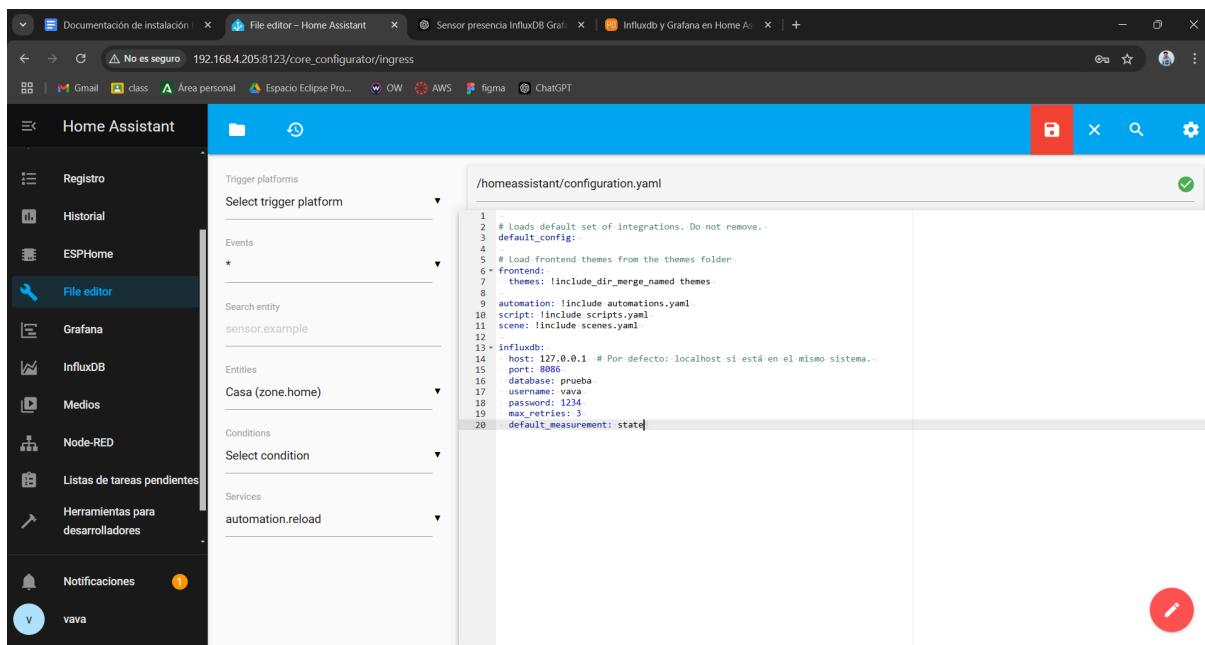
## paso 4:

Una vez creada la base de datos,nos dirigimos hacia ajustes. Dentro de ajustes le damos al apartado de información.



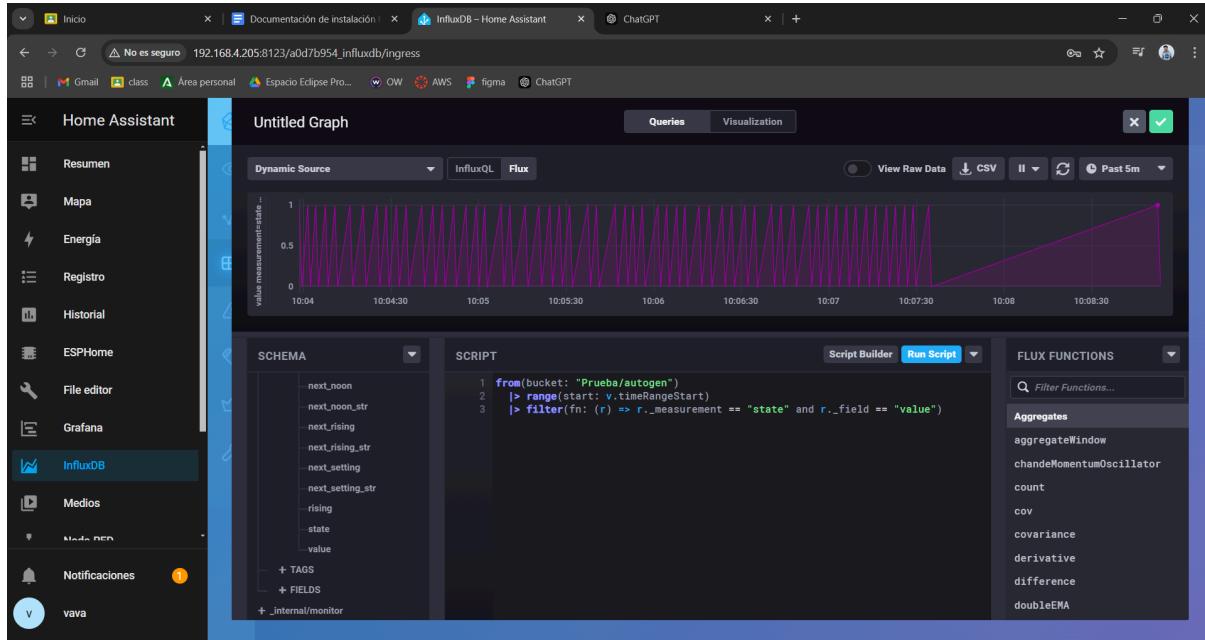
## paso 5:

Luego nos dirigimos a File editor y una vez allí editamos el archivo de configuración.yaml



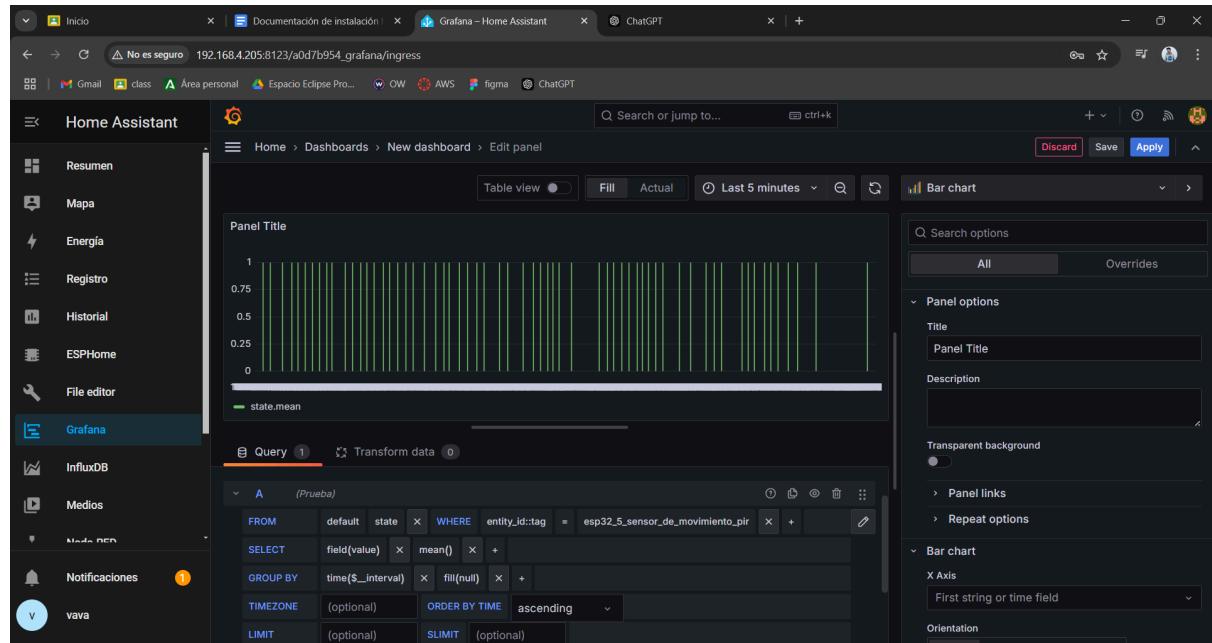
## paso 6:

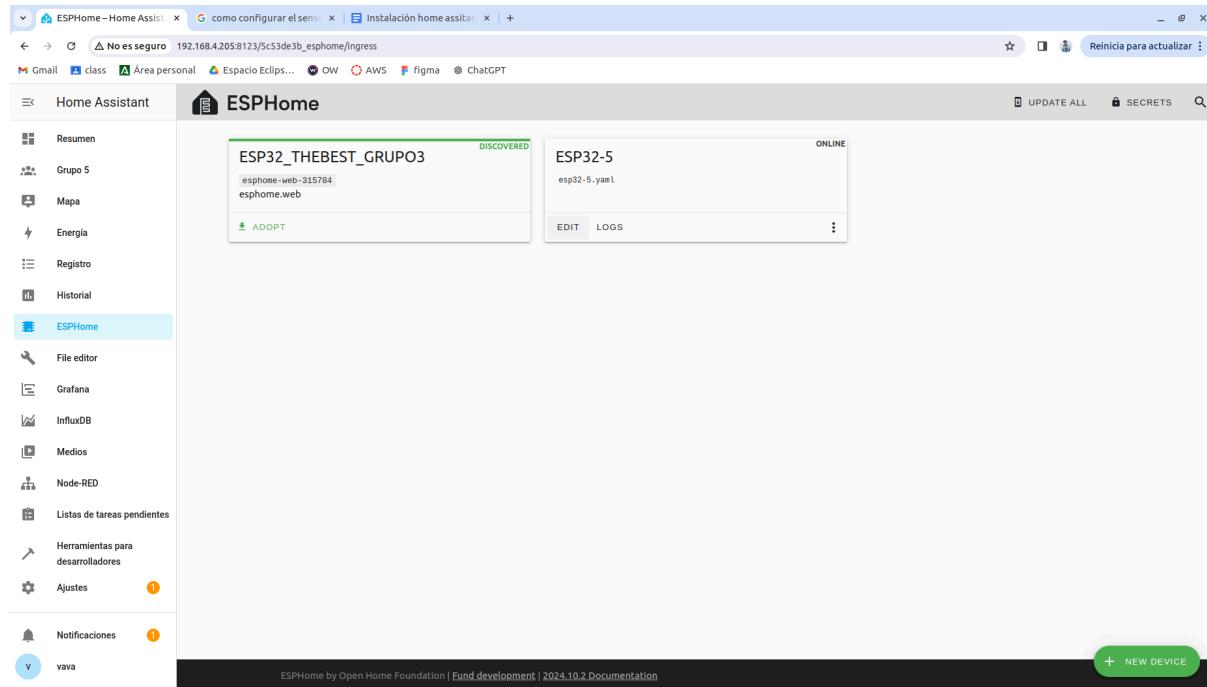
El siguiente paso es dirigirnos hacia influxdb y darle a la opción de explore, dentro veremos la gráfica para detectar el sensor de movimiento



## paso 7:

Nos dirigimos hacia grafana y vemos la gráfica del sensor de movimiento.





## Coneectar Sensores

### paso 1:

Este es el código que tenemos que añadir a la configuración del .yaml para activar nuestro sensor de temperatura.

```
 3   # Añade el sensor de temperatura interno del ESP32
 4   sensor:
 5     - platform: internal_temperature
 6       name: "Temperatura Interna ESP32"
 7       update_interval: 15s
```

Entramos en ajustes al panel de control. Una vez ahí clicamos en el botón de abajo a la derecha y creamos un panel de control.

Título	Método de configuración	Sol...	Mo...	
Resumen	Controlado por la IU	-	✓	<a href="#">ABRIR</a>
Energía	Controlado por la IU	-	✓	<a href="#">ABRIR</a>
Grupo 5	Controlado por la IU	-	✓	<a href="#">ABRIR</a>
Mapa	Controlado por la IU	-	✓	<a href="#">ABRIR</a>

# Módulo Relé 1 Canal 3v

## Optoacoplado

Model MKTX00002298\_S-7-4

### Descripción:

- 1- Gracias al optoacoplador, el módulo protege el microcontrolador de posibles picos de voltaje o interferencias provenientes del circuito de carga.
- 2- Interruptor de alimentación de relé de 1 canal de 3V con disparador de alto nivel de aislamiento óptico. Aislador optoacoplador-Módulo de relé de potencia de 3V/3,3 V compatible con control de aislamiento de fotoacoplador.
- 3- Disparador de alto nivel: el módulo de relé se activa mediante una señal de alto nivel (1,3 V), que se puede introducir desde el microcontrolador IO.
- 4- Larga vida útil: tiempo de respuesta rápido de menos de 20ms y larga vida útil de hasta 100.000 veces la acción a presión con protección de rueda libre de diodo.

### Conexiones básicas:

#### 1. Pines del lado de control:

- **VCC**: Alimentación del módulo (3.3V del microcontrolador).
- **GND**: Tierra común entre el microcontrolador y el módulo.
- **IN**: Señal de control (alta/baja) para activar o desactivar el relé.

#### 2. Lado de potencia (carga):

El relé tiene tres terminales en su lado de carga:

- **COM (común)**: Entrada de la señal que deseas controlar (fase o positivo).
- **NO (normalmente abierto)**: El circuito se cierra al activar el relé.
- **NC (normalmente cerrado)**: El circuito está cerrado cuando el relé está desactivado.

## Ejemplo de conexión con ESP32 o Arduino:

### Conexión básica:

- **VCC**: Conéctalo al pin de 3.3V del ESP32/Arduino.
- **GND**: Conéctalo a GND del ESP32/Arduino.
- **IN**: Conéctalo a un pin digital de tu ESP32/Arduino (por ejemplo, el D23 en ESP32).

### Circuito de carga (ejemplo para encender una lámpara):

1. Conecta el **COM** del relé a la fase (L) de la red eléctrica.
2. Conecta el terminal **NO** al lado positivo de la lámpara.
3. Conecta el neutro (N) directamente a la lámpara.

### Código de ejemplo (Arduino/ESP32):

cpp

Copiar código

---

```
#define RELAY_PIN 23 // Pin conectado al relé

void setup() {
    pinMode(RELAY_PIN, OUTPUT); // Configura el pin como salida
    digitalWrite(RELAY_PIN, LOW); // Asegúrate de que el relé esté
    apagado al inicio
}

void loop() {
    digitalWrite(RELAY_PIN, HIGH); // Activa el relé
    delay(1000); // Espera 1 segundo
    digitalWrite(RELAY_PIN, LOW); // Desactiva el relé
    delay(1000); // Espera 1 segundo
}
```

---

## Ventajas del optoacoplador:

- Protege el microcontrolador de altas corrientes o picos de voltaje provenientes del circuito de carga.
- Aumenta la confiabilidad del circuito en entornos ruidosos o con interferencias.

## Notas importantes:

- 1. Fuente de alimentación:** Verifica que la fuente de 3.3V pueda suministrar suficiente corriente al módulo.
  - 2. Protección del circuito de carga:**
    - Usa fusibles si controlas dispositivos de alta potencia.
    - Evita manejar conexiones eléctricas de AC sin experiencia.

## Sensor relé

The screenshot displays two separate desktop environments, likely running on different machines or virtual machines, both showing the Home Assistant interface.

**Top Monitor (Ubuntu 22.04 LTS):**

- Left Sidebar:** Includes links for Resumen, Grupo 5, Mapa, Energía, Registro, Historial, ESPHome, File editor, Grafana, InfluxDB, Medios, Node-RED, Listas de tareas pendientes, Herramientas para desarrolladores, Notificaciones, and vava.
- Central Area:** A code editor window titled "esp32-5.yaml" containing the following configuration:

```
ssid: "Esp32-5 Fallback Hotspot"
password: "0wfknd1b8CYG"
binary_sensor:
  platform: gpio
  pin: GPIO27 Cambia al pin que useaste
  name: "Sensor de Movimiento PIR"
  device_class: motion

# Añade el sensor de temperatura interno del ESP32
sensor:
  - platform: internal_temperature
    name: "Temperatura Interna ESP32"
    update_interval: 15s

#define RELAY_PIN 23 // Pin conectado al relé

void setup() {
  pinMode(RELAY_PIN, OUTPUT); // Configura el pin como salida
  digitalWrite(RELAY_PIN, LOW); // Asegurate de que el relé esté apagado al inicio
}

void loop() {
  digitalWrite(RELAY_PIN, HIGH); // Activa el relé
  delay(1000); // Espera 1 segundo
  digitalWrite(RELAY_PIN, LOW); // Desactiva el relé
  delay(1000); // Espera 1 segundo
}
```

- Bottom Taskbar:** Shows icons for Home Assistant, Google Chrome, and other system applications.

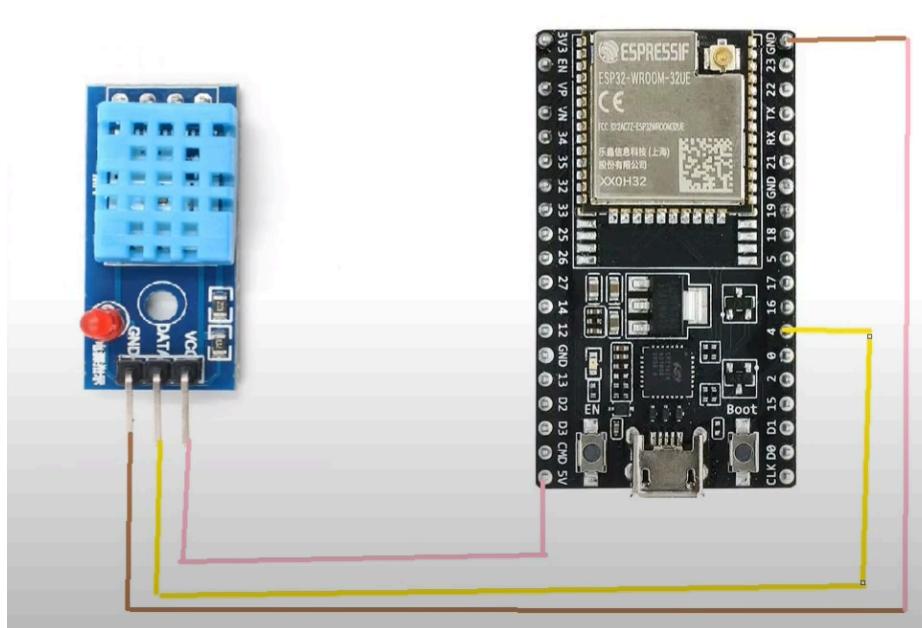
**Bottom Monitor (Windows 10):**

- Left Sidebar:** Same as the top monitor.
- Central Area:** A code editor window titled "esp32-5.yaml" showing the same configuration as the top monitor.
- Bottom Taskbar:** Shows icons for Home Assistant, Google Chrome, and other system applications.

# Sensor Az delivery ky-015 dht11

## Descripción:

El sensor de temperatura KY-015 DHT11 es un módulo electrónico utilizado para medir la temperatura y la humedad de un entorno. Contiene un sensor de temperatura y un sensor de humedad acoplados a un microcontrolador. Los valores medidos pueden transmitirse a un circuito externo a través de un bus de datos digital. El sensor de temperatura DHT11 KY-015 es una herramienta útil para controlar las condiciones ambientales en muchas aplicaciones, como la domótica, la agricultura y la industria.



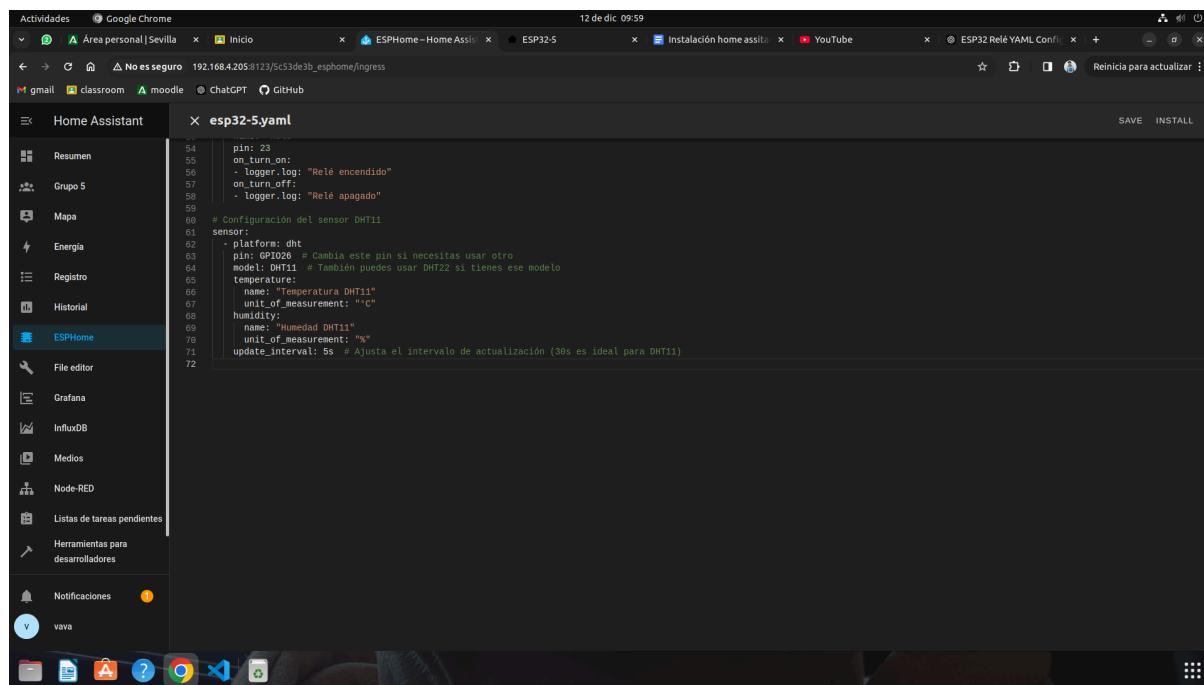
[https://www.google.com/search?q=az+delivery+ky-015+dht11+como+se+conecta+a+la+esp32+sin+arduino+&sca\\_esv=61ba496ce5dbe72c&ei=qJtaZ4q2OeLX7M8P2rHqqAQ&ved=0ahUKEwiKqrqN5aGKAxXiK\\_sDHdqYGkUQ4dUDCBA&uact=5&oq=az+delivery+ky-015+dht11+como+se+conecta+a+la+esp32+sin+arduino+&gs\\_lp=Egxn3Mtd2I6LXNIcnAiQGF6IGRlbGI2ZXJ5IGt5LTAxNSBkaHQxMSBjb21vIHNIIGNvbmVjdGEqYSBsYSBlc3AzMiBzaW4qYXJkdWIubyBishhQnwVYzhdwAXqAkAEAmAGsAaAB-QyqAQM1ljm4AQPIAQD4AQGYAqqgAuoJwgIKEAYsAMY1gQYR8ICBxAhGKABGAqYAwCIBqGQBgiSBwMxLjmgB\\_43&sclient=gws-wiz-serp#fpstate=ive&vld=cid:c642cbfd,vid:sERRresoqv4,st:159](https://www.google.com/search?q=az+delivery+ky-015+dht11+como+se+conecta+a+la+esp32+sin+arduino+&sca_esv=61ba496ce5dbe72c&ei=qJtaZ4q2OeLX7M8P2rHqqAQ&ved=0ahUKEwiKqrqN5aGKAxXiK_sDHdqYGkUQ4dUDCBA&uact=5&oq=az+delivery+ky-015+dht11+como+se+conecta+a+la+esp32+sin+arduino+&gs_lp=Egxn3Mtd2I6LXNIcnAiQGF6IGRlbGI2ZXJ5IGt5LTAxNSBkaHQxMSBjb21vIHNIIGNvbmVjdGEqYSBsYSBlc3AzMiBzaW4qYXJkdWIubyBishhQnwVYzhdwAXqAkAEAmAGsAaAB-QyqAQM1ljm4AQPIAQD4AQGYAqqgAuoJwgIKEAYsAMY1gQYR8ICBxAhGKABGAqYAwCIBqGQBgiSBwMxLjmgB_43&sclient=gws-wiz-serp#fpstate=ive&vld=cid:c642cbfd,vid:sERRresoqv4,st:159)

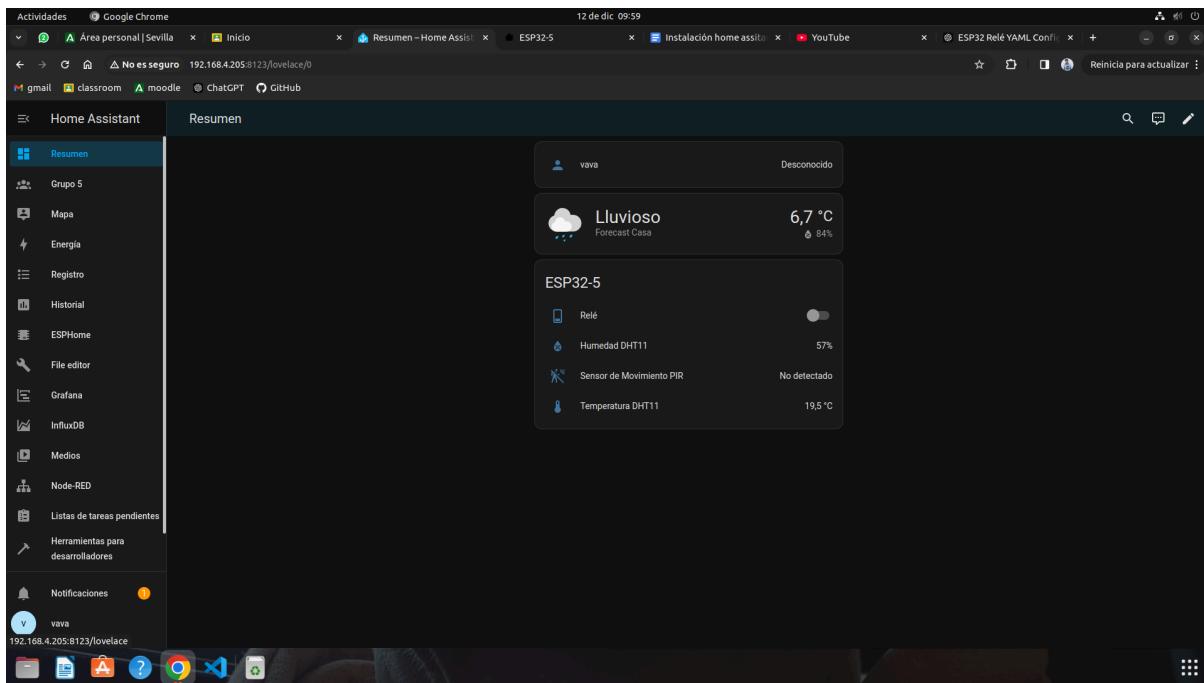
## paso 1:

Aquí está el yaml necesario dentro de esp Home para que funcione el sensor de temperatura y de humedad

```
# Configuración del sensor DHT11
sensor:
  - platform: dht
    pin: GPIO26 # Cambia este pin si necesitas usar otro
    model: DHT11 # También puedes usar DHT22 si tienes ese modelo
    temperature:
      name: "Temperatura DHT11"
      unit_of_measurement: "°C"
    humidity:
      name: "Humedad DHT11"
      unit_of_measurement: "%"
    update_interval: 5s # Ajusta el intervalo de actualización (30s es ideal para DHT11)
```

- pin: GPIO4**: Asegúrate de conectar el pin de datos del KY-015 (DHT11) al GPIO4 de la ESP32 (puedes cambiarlo si utilizas otro pin).
- Fuente de alimentación**: Conecta el pin VCC del KY-015 al pin de 3.3V o 5V de la ESP32 y GND a GND.
- Actualización de datos**: La frecuencia de actualización se configura en **update\_interval: 60s** (1 vez por minuto).





## MQTT

### ¿Qué es?

Es un protocolo de mensajería basado en estándares, o un conjunto de reglas, que se utiliza para la comunicación de un equipo a otro. Facilita a los desarrolladores el cifrado de mensajes y la autenticación de dispositivos y usuarios mediante protocolos de autenticación modernos

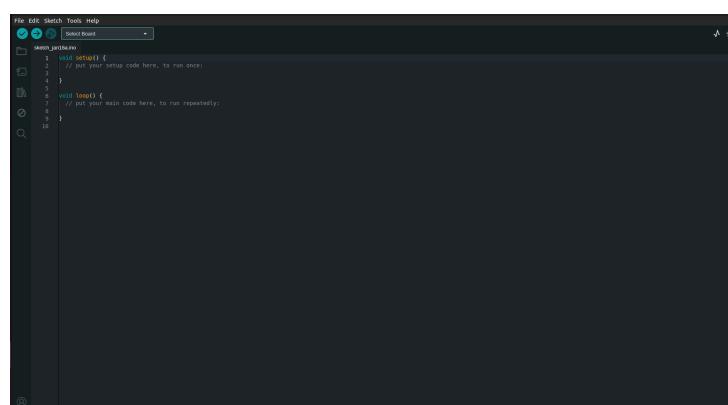
### Configuración:

#### paso 1:

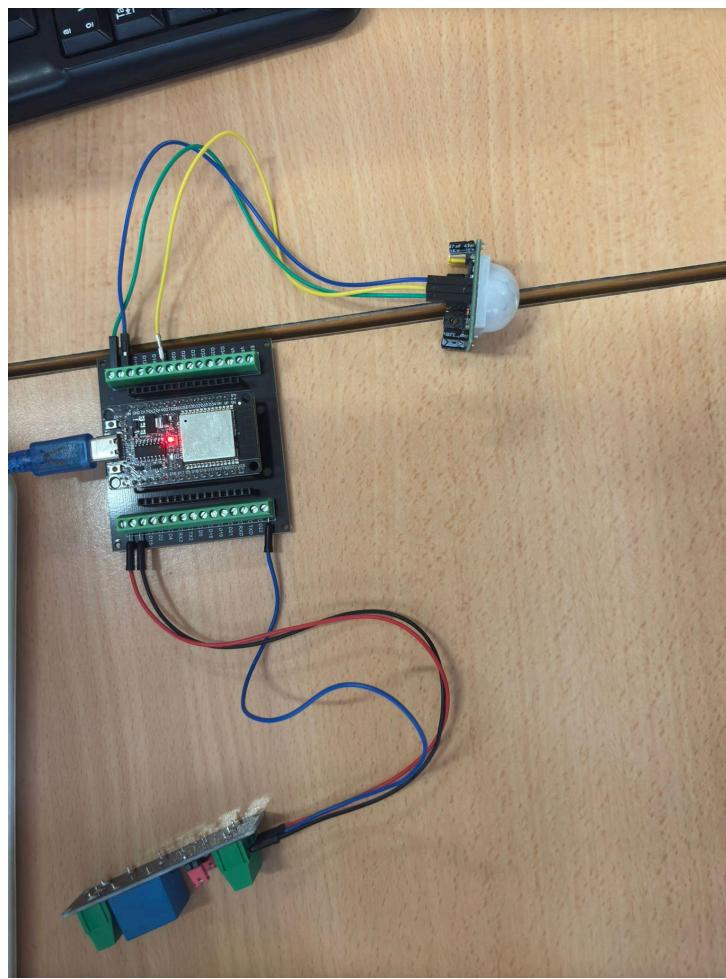
Para configurar la ESP-32 sin el uso de homeassistant es necesaria un IDE. El IDE se llama Arduino IDE y se descarga desde el siguiente enlace

<https://www.arduino.cc/en/software/download-thank-you/>

Una vez descargamos el entorno lo abrimos



Donde pone que se select board debemos buscar nuestra placa.  
Nuestra placa utiliza un relé y un sensor de movimiento pir



## código:

```

File Edit Sketch Tools Help
Select Board
sketch_jan16a.ino
4 // Configuración Wi-Fi
5 const char* ssid = "2DAW_IoT";
6 const char* password = "Somos2DAW";
7
8 // Configuración MQTT
9 const char* mqtt_server = "ha.ieshm.org";
10 const int mqtt_port = 1883;
11 const char* mqtt_user = "mqtt";
12 const char* mqtt_password = "mqtt";
13 const char* mqtt_topic = "gs/rele"; // Tópico para controlar el relé
14
15 // Pines
16 #define LED_BUILTIN 2 // LED integrado en la placa
17 #define RELAY_PIN 23 // Pin donde está conectado el relé (GPIO 23)
18 #define PIR_PIN 27 // Pin donde está conectado el sensor PIR (GPIO 27)
19
20 // Variables globales
21 WiFiClient espClient;
22 PubSubClient client(espClient);
23 bool pirState = false; // Estado actual del sensor PIR
24
25 // Función para conectarse a la red Wi-Fi
26 void setup_wifi() {
27   delay(10);
28   Serial.println("Conectando a Wi-Fi...");
29   WiFi.begin(ssid, password);
30   while (WiFi.status() != WL_CONNECTED) {
31     delay(1000);
32     Serial.print(".");
33   }
34   Serial.println("\nWi-Fi conectado.");
35   Serial.print("Dirección IP: ");
36   Serial.println(WiFi.localIP());
37 }
38
39 // Función que se ejecuta cuando se recibe un mensaje en el broker
40 void callback(char* topic, byte* payload, unsigned int length) {
41   String message = "";
42   for (unsigned int i = 0; i < length; i++) {
43     message += (char)payload[i];
44   }
45   Serial.print("Mensaje recibido: ");
46   Serial.println(message);
47 }
48
49 // Función para reconectar al broker MQTT si la conexión se pierde
50 void reconnect() {

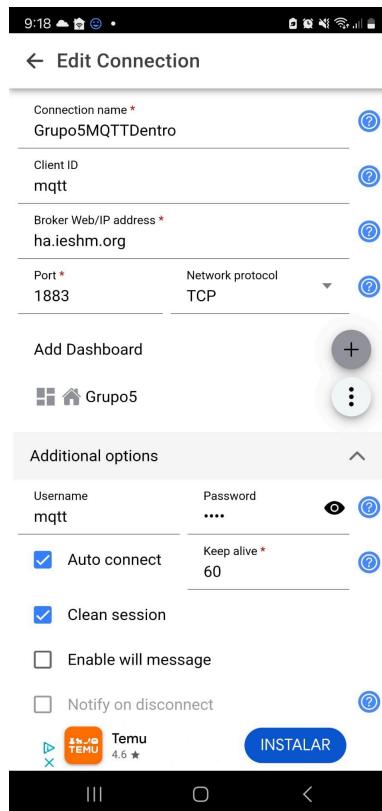
```

Este programa hace que cuando el sensor tiene presencia se encienda el relé

## Mediante WIFI

### paso 1:

Primero configuramos un dispositivo móvil a la red wifi que usaremos



### paso 2:

También El ESP-32 como vemos en la siguiente captura de pantalla

```
#include <WiFi.h>
#include <PubSubClient.h>

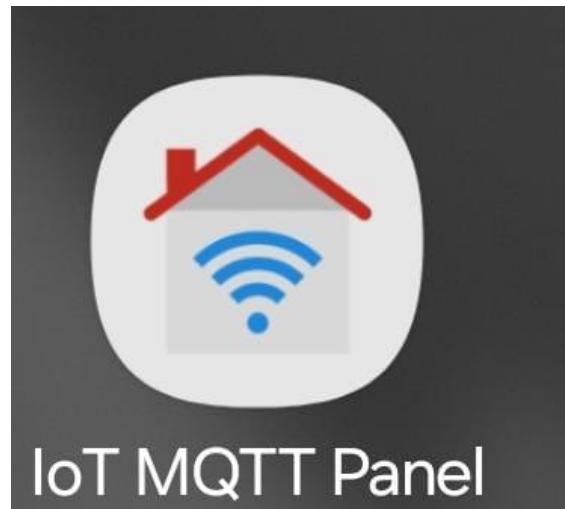
// Configuración Wi-Fi
const char* ssid = "2DAW_IoT";
const char* password = "Somos2DAW";

// Configuración MQTT
const char* mqtt_server = "ha.ieshm.org";
const int mqtt_port = 1883;
const char* mqtt_user = "mqtt";
const char* mqtt_password = "mqtt";
const char* mqtt_topic = "g5/rele"; // Tópico para controlar el relé
```

## paso 3:

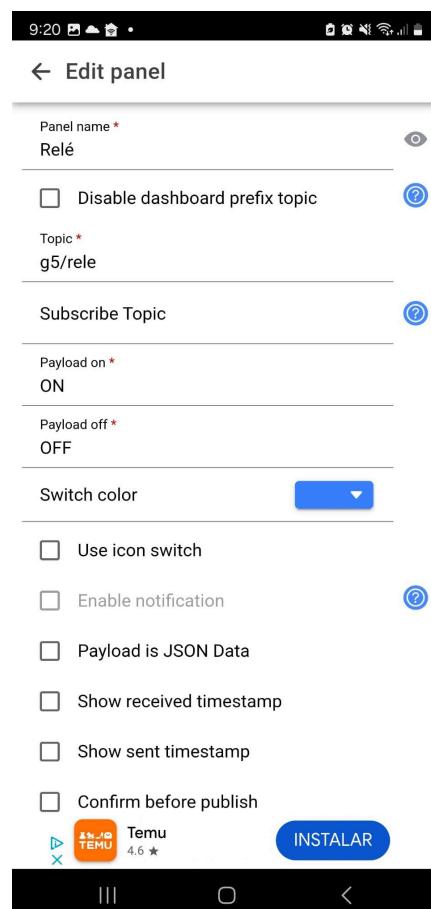
Configuramos en el móvil el relé

La aplicación que hemos utilizado para el móvil es IoT MQTT Panel.



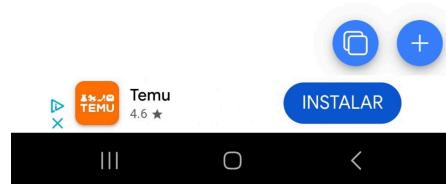
## paso 4:

Ahora vamos a conectar una lámpara y que se encienda si detecta movimiento cada 30 segundos y si no detecta que se apague, sensibilidad y luminosidad a través del sensor de presencia para saber si funciona o no



## paso 5:

Vemos cómo funciona encendiéndolo desde la app de móvil



```
#include <WiFi.h>
#include <PubSubClient.h>

// Configuración Wi-Fi
const char* ssid = "2DAW_IoT";
const char* password = "Somos2DAW";

// Configuración MQTT
const char* mqtt_server = "ha.ieshm.org";
const int mqtt_port = 1883;
const char* mqtt_user = "mqtt";
const char* mqtt_password = "mqtt";
const char* mqtt_topic = "g5/rele"; // Tópico para controlar el relé
const char* mqtt_log_topic = "g5/logs"; // Tópico para enviar logs

// Pines
#define LED_BUILTIN 2 // LED integrado en la placa
```

```
#define RELAY_PIN 23      // Pin donde está conectado el relé (GPIO 23)
#define PIR_PIN 27        // Pin donde está conectado el sensor PIR (GPIO 27)

// Variables globales
WiFiClient espClient;
PubSubClient client(espClient);
unsigned long relayEndTime = 0; // Tiempo en el que se debe apagar el relé
bool relayActive = false;     // Estado del relé
bool mqttControl = false;    // Prioridad de control desde MQTT

// Función para conectarse a la red Wi-Fi
void setup_wifi() {
    delay(10);
    Serial.println("Conectando a Wi-Fi...");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }
    Serial.println("\nWi-Fi conectado.");
    Serial.print("Dirección IP: ");
    Serial.println(WiFi.localIP());
}

// Función para enviar logs al broker MQTT
void sendLog(String logMessage) {
    if (client.connected()) {
        client.publish(mqtt_log_topic, logMessage.c_str());
    }
}

// Función que se ejecuta cuando se recibe un mensaje en el broker
void callback(char* topic, byte* payload, unsigned int length) {
    String message = "";
    for (unsigned int i = 0; i < length; i++) {
        message += (char)payload[i];
    }
    Serial.print("Mensaje recibido en el tópico ");
    Serial.print(topic);
    Serial.print(": ");
    Serial.println(message);

    // Enviar log al broker
    sendLog("Mensaje recibido: " + message);

    // Manejo de mensajes para controlar el relé
    if (message == "ON") {
        Serial.println("Encendiendo relé por MQTT.");
    }
}
```

```
digitalWrite(RELAY_PIN, HIGH);
digitalWrite(LED_BUILTIN, HIGH);
relayActive = true;
mqttControl = true; // MQTT toma el control
sendLog("Relé encendido por MQTT.");
} else if (message == "OFF") {
  Serial.println("Apagando relé por MQTT.");
  digitalWrite(RELAY_PIN, LOW);
  digitalWrite(LED_BUILTIN, LOW);
  relayActive = false;
  mqttControl = false; // MQTT toma el control
  sendLog("Relé apagado por MQTT.");
}
}

// Función para reconectar al broker MQTT si la conexión se pierde
void reconnect() {
  while (!client.connected()) {
    Serial.print("Intentando conectar al broker MQTT...");
    if (client.connect("ESP32", mqtt_user, mqtt_password)) {
      Serial.println("Conectado.");
      client.subscribe(mqtt_topic);
      sendLog("Conectado al broker MQTT.");
    } else {
      Serial.print("Fallo en la conexión, código de error: ");
      Serial.print(client.state());
      Serial.println(" Reintentando en 5 segundos...");
      sendLog("Error conectando al broker MQTT. Código: " + String(client.state()));
      delay(5000);
    }
  }
}

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

  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(RELAY_PIN, OUTPUT);
  pinMode(PIR_PIN, INPUT);

  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);

  digitalWrite(RELAY_PIN, LOW); // Apaga el relé al inicio
  digitalWrite(LED_BUILTIN, LOW); // Apaga el LED al inicio

  sendLog("ESP32 iniciado correctamente.");
```

```
}

void loop() {
    // Mantener la conexión con el broker MQTT
    if (!client.connected()) {
        reconnect();
    }
    client.loop(); // Procesar los mensajes MQTT

    // Leer el estado del sensor PIR
    bool pirDetected = digitalRead(PIR_PIN);

    // Depuración: Imprimir estado actual
    static unsigned long lastDebugTime = 0;
    if (millis() - lastDebugTime > 500) {
        lastDebugTime = millis();
        Serial.print("Estado del PIR: ");
        Serial.print(pirDetected);
        Serial.print(", Control MQTT: ");
        Serial.print(mqttControl);
        Serial.print(", Relé activo: ");
        Serial.println(relayActive);
    }

    // Lógica del relé controlado por MQTT
    if (mqttControl) { // Si MQTT manda ON
        if (pirDetected) {
            // Si el PIR detecta movimiento, activa el relé y reinicia el temporizador
            Serial.println("Movimiento detectado: Encendiendo relé y reiniciando temporizador.");
            digitalWrite(RELAY_PIN, HIGH);
            digitalWrite(LED_BUILTIN, HIGH);
            relayActive = true; // Marca el relé como activo
            relayEndTime = millis() + 5000; // Configura el temporizador a 5 segundos
            sendLog("Relé encendido por movimiento detectado.");
        }
    }

    // Si el temporizador se agota, apaga el relé
    if (relayActive && millis() > relayEndTime) {
        Serial.println("Tiempo completado: Apagando relé.");
        digitalWrite(RELAY_PIN, LOW);
        digitalWrite(LED_BUILTIN, LOW);
        relayActive = false; // Marca el relé como inactivo
        sendLog("Relé apagado tras 5 segundos.");
    }
} else {
    // Si MQTT manda OFF, apaga el relé inmediatamente
    if (relayActive || pirDetected) {
        Serial.println("MQTT OFF recibido: Apagando relé y desactivando PIR.");
    }
}
```

```
    digitalWrite(RELAY_PIN, LOW);
    digitalWrite(LED_BUILTIN, LOW);
    relayActive = false; // Marca el relé como inactivo
    sendLog("Relé apagado por MQTT OFF.");
}
}
```

Este es el código en el cual cuando lo activamos el MQTT cuando el sensor detecta movimiento se enciende en relé y si en 5 segundos no detecta movimiento se apaga.

## *Conectar 2 ESP32 con los sensores PIR*

### paso 1:

Configuramos el código a través de la aplicación Arduino con lo que introduciremos este primer código para configurar el primer relé.

```
[8:43, 30/1/2025] Abraham: #include <WiFi.h>
#include <PubSubClient.h>

// Configuración Wi-Fi
const char* ssid = "2DAW_IoT";
const char* password = "Somos2DAW";

// Configuración MQTT
const char* mqtt_server = "ha.ieshm.org";
```

```
const int mqtt_port = 1883;
const char* mqtt_user = "mqtt";
const char* mqtt_password = "mqtt";
const char* mqtt_topic = "g5/rele"; // Tópico para controlar el relé
const char* mqtt_log_topic = "g5/logs"; // Tópico para enviar logs
const char* mqtt_topic_pir2 = "g5/pir2"; // Tópico del PIR de la segunda ESP32

// Pines
#define LED_BUILTIN 2    // LED integrado en la placa
#define RELAY_PIN 23     // Pin donde está conectado el relé (GPIO 23)
#define PIR_PIN 27        // Pin donde está conectado el sensor PIR (GPIO 27)

// Variables globales
WiFiClient espClient;
PubSubClient client(espClient);
unsigned long relayEndTime = 0; // Tiempo en el que se debe apagar el relé
bool relayActive = false;      // Estado del relé
bool mqttControl = false;      // Prioridad de control desde MQTT

// Función para conectarse a la red Wi-Fi
void setup_wifi() {
    delay(10);
    Serial.println("Conectando a Wi-Fi...");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }
    Serial.println("\nWi-Fi conectado.");
    Serial.print("Dirección IP: ");
    Serial.println(WiFi.localIP());
}

// Función para enviar logs al broker MQTT
void sendLog(String logMessage) {
    if (client.connected()) {
        client.publish(mqtt_log_topic, logMessage.c_str());
    }
}

// Función que se ejecuta cuando se recibe un mensaje en el broker
void callback(char* topic, byte* payload, unsigned int length) {
    String message = "";
    for (unsigned int i = 0; i < length; i++) {
        message += (char)payload[i];
    }
    Serial.print("Mensaje recibido en el tópico ");
}
```

```
Serial.print(topic);
Serial.print(": ");
Serial.println(message);

// Manejo de mensajes MQTT
if (String(topic) == mqtt_topic) { // Mensajes para el control del relé
    if (message == "ON") {
        mqttControl = true; // Habilita el control por MQTT
        sendLog("Relé habilitado por MQTT.");
    } else if (message == "OFF") {
        mqttControl = false; // Deshabilita el control por MQTT
        sendLog("Relé deshabilitado por MQTT.");
    }
} else if (String(topic) == mqtt_topic_pir2) { // Mensajes del PIR de la segunda ESP32
    if (mqttControl && message == "DETECTED") {
        // Activa el relé si MQTT está en ON y hay movimiento en el PIR remoto
        Serial.println("Movimiento detectado en la segunda ESP32: Activando relé.");
        digitalWrite(RELAY_PIN, HIGH);
        digitalWrite(LED_BUILTIN, HIGH);
        relayActive = true;
        relayEndTime = millis() + 5000; // Temporizador de 5 segundos
        sendLog("Relé activado por segundo PIR.");
    }
}
}

// Función para reconectar al broker MQTT si la conexión se pierde
void reconnect() {
    while (!client.connected()) {
        Serial.print("Intentando conectar al broker MQTT...");
        if (client.connect("ESP32", mqtt_user, mqtt_password)) {
            Serial.println("Conectado.");
            client.subscribe(mqtt_topic); // Tópico para el control del relé
            client.subscribe(mqtt_topic_pir2); // Tópico del segundo PIR
            sendLog("Conectado al broker MQTT y suscrito a los tópicos.");
        } else {
            Serial.print("Fallo en la conexión, código de error: ");
            Serial.println(client.state());
            Serial.println("Reintentando en 5 segundos...");
            delay(5000);
        }
    }
}

void setup() {
    Serial.begin(115200);
```

```
pinMode(LED_BUILTIN, OUTPUT);
pinMode(RELAY_PIN, OUTPUT);
pinMode(PIR_PIN, INPUT);

setup_wifi();
client.setServer(mqtt_server, mqtt_port);
client.setCallback(callback);

digitalWrite(RELAY_PIN, LOW); // Apaga el relé al inicio
digitalWrite(LED_BUILTIN, LOW); // Apaga el LED al inicio

sendLog("ESP32 i...
[8:43, 30/1/2025] Abraham: #include <WiFi.h>
#include <PubSubClient.h>

// Configuración Wi-Fi
const char* ssid = "2DAW_IoT";
const char* password = "Somos2DAW";

// Configuración MQTT
const char* mqtt_server = "ha.ieshm.org";
const int mqtt_port = 1883;
const char* mqtt_user = "mqtt";
const char* mqtt_password = "mqtt";
const char* mqtt_topic_pir = "g5/pir2"; // Tópico para enviar el estado del PIR

// Pines
#define PIR_PIN 26 // Pin donde está conectado el sensor PIR (GPIO 27)

// Variables globales
WiFiClient espClient;
PubSubClient client(espClient);
bool lastPirState = LOW; // Estado anterior del PIR

// Función para conectarse a la red Wi-Fi
void setup_wifi() {
    delay(10);
    Serial.println("Conectando a Wi-Fi...");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }
    Serial.println("\nWi-Fi conectado.");
    Serial.print("Dirección IP: ");
    Serial.println(WiFi.localIP());
}
```

```
// Función para reconectar al broker MQTT
void reconnect() {
    while (!client.connected()) {
        Serial.print("Intentando conectar al broker MQTT...");
        if (client.connect("ESP32_PIR2", mqtt_user, mqtt_password)) {
            Serial.println("Conectado.");
        } else {
            Serial.print("Fallo en la conexión, código de error: ");
            Serial.println(client.state());
            Serial.println("Reintentando en 5 segundos...");
            delay(5000);
        }
    }
}

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

    pinMode(PIR_PIN, INPUT);

    setup_wifi();
    client.setServer(mqtt_server, mqtt_port);

    Serial.println("ESP32 con PIR lista.");
}

void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();

    // Leer el estado del sensor PIR
    bool pirState = digitalRead(PIR_PIN);

    // Enviar estado si hay un cambio
    if (pirState != lastPirState) {
        lastPirState = pirState;
        if (pirState == HIGH) {
            client.publish(mqtt_topic_pir, "DETECTED");
            Serial.println("Movimiento detectado, enviado mensaje a MQTT.");
        }
    }
    delay(100); // Reducir la frecuencia de lectura
}
```

Este primer código es el principal en el cuál conectamos y configuramos los dos sensores, el relé, su funcionamiento es una vez que pasen 5 segundos sin detectar movimiento se

apague, todo esto también se controla desde MQTT ya que su configuración está metida en el código.

Al segundo código se le mete un segundo sensor PIR a lo igual que al primero y su funcionamiento es igual pero actúa como una especie de repetidor si uno se enciende se le manda información al otro, lo detecta y enciende el relé.

```
#include <WiFi.h>
#include <PubSubClient.h>

// Configuración Wi-Fi
const char* ssid = "2DAW_IoT";
const char* password = "Somos2DAW";

// Configuración MQTT
const char* mqtt_server = "ha.ieshm.org";
const int mqtt_port = 1883;
const char* mqtt_user = "mqtt";
const char* mqtt_password = "mqtt";
const char* mqtt_topic_pir = "g5/pir2"; // Tópico para enviar el estado del PIR

// Pines
#define PIR_PIN 26 // Pin donde está conectado el sensor PIR (GPIO 27)

// Variables globales
WiFiClient espClient;
PubSubClient client(espClient);
bool lastPirState = LOW; // Estado anterior del PIR

// Función para conectarse a la red Wi-Fi
void setup_wifi() {
    delay(10);
    Serial.println("Conectando a Wi-Fi...");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }
    Serial.println("\nWi-Fi conectado.");
    Serial.print("Dirección IP: ");
    Serial.println(WiFi.localIP());
}

// Función para reconnectar al broker MQTT
void reconnect() {
    while (!client.connected()) {
```

```
Serial.print("Intentando conectar al broker MQTT...");  
if (client.connect("ESP32_PIR2", mqtt_user, mqtt_password)) {  
    Serial.println("Conectado.");  
} else {  
    Serial.print("Fallo en la conexión, código de error: ");  
    Serial.println(client.state());  
    Serial.println("Reintentando en 5 segundos...");  
    delay(5000);  
}  
}  
}  
  
void setup() {  
    Serial.begin(115200);  
  
    pinMode(PIR_PIN, INPUT);  
  
    setup_wifi();  
    client.setServer(mqtt_server, mqtt_port);  
  
    Serial.println("ESP32 con PIR lista.");  
}  
  
void loop() {  
    if (!client.connected()) {  
        reconnect();  
    }  
    client.loop();  
  
    // Leer el estado del sensor PIR  
    bool pirState = digitalRead(PIR_PIN);  
  
    // Enviar estado si hay un cambio  
    if (pirState != lastPirState) {  
        lastPirState = pirState;  
        if (pirState == HIGH) {  
            client.publish(mqtt_topic_pir, "DETECTED");  
            Serial.println("Movimiento detectado, enviado mensaje a MQTT.");  
        }  
    }  
    delay(100); // Reducir la frecuencia de lectura  
}
```

# Shelly EM

## Explicación:

La Shelly EM es una placa de medición de voltaje e intensidad. Pero lo que más necesitamos para nuestro proyecto es la potencia.

Vamos a utilizar la placa para medir el consumo de energía que va a utilizar el aula, este proyecto puede ser escalable para otros usos como programar un relé wifi. O simplemente medir y tratar de ahorrar energía que es el proyecto principal que tenemos en mente ahora mismo

## Requerimientos:

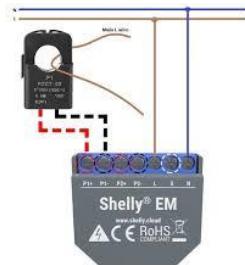
1. Shelly EM
2. Shelly Transformador De Corriente 50A
3. Conexión wifi
4. Cables
5. Dispositivo móvil

## Paso 1: Conectar pinza

Conectar la pinza a la Shelly a través de uno de los puertos, ya sea P1 o P2.

## Paso 2: Alimentar el Shelly EM

- Conecta el **Shelly EM** a la red eléctrica (230V AC) siguiendo el esquema del fabricante.
- Asegúrate de que la fase y el neutro estén bien conectados.



## Paso 3: Configurar la conexión WiFi

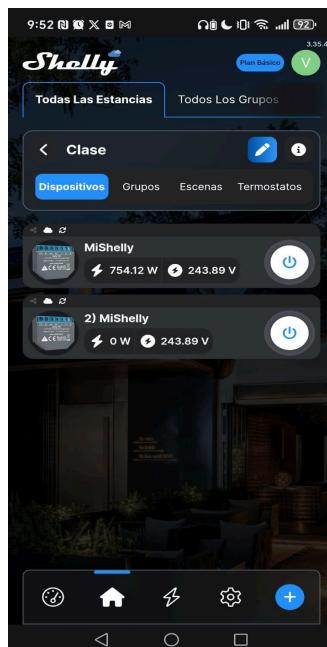
- Enciende el **Shelly EM** y conéctate a su red WiFi (Shelly-XXXXXX).

- Accede a la IP **192.168.33.1** desde un navegador o desde la **aplicación móvil** (recomendable)
- Configura la conexión WiFi para que se una a tu red doméstica.



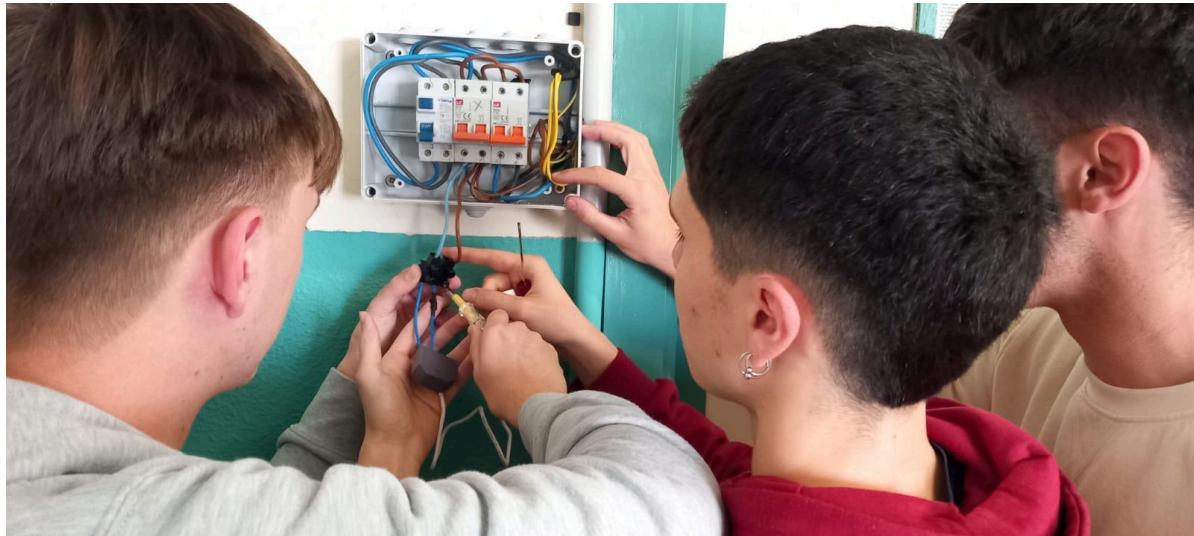
## Paso 4: Configurar en la App

- Si usas la **app de Shelly**, agrégalo a tu cuenta y personaliza las mediciones.
- También puedes añadir la sala en la que quieras configurar la shelly



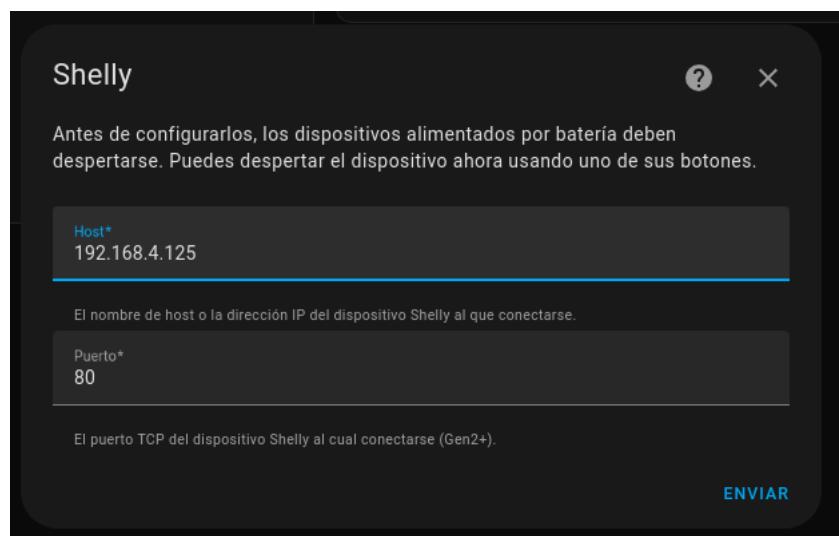
## Paso 5: Comprobar las mediciones

- Accede a la interfaz web o app para ver los valores en tiempo real.
- Asegúrate de que la pinza está correctamente colocada en el cable de fase.



## Paso 6: Configurar en Home Assistant

- Entra en tu Home Assistant y entra en **dispositivos y servicios** descarga su complemento
- Configura la dirección IP



- Configuración terminada

A screenshot of the Home Assistant interface showing the details for a Shelly EM device. The top navigation bar has "En 2DAW" and the device name "shellyem-244CAB418B7A". The main card displays "Información de Dispositivo" with the model "Shelly EM", firmware "Firmware: 20210429-104036/v1.10.4-2-gfa159c1fb-release-1.10", hardware "Hardware: gen1 (SHEM)", and MAC address "MAC: 24:4C:AB:41:8B:7A". It also shows a "Visitar" button and a "Automatizaciones" section. To the right are three cards: "Controles" (with a switch), "Registro" (empty log), and "Sensores" (listing Channel 1 and Channel 2 energy, power, voltage, and current values).

Página 44