



UNITED INTERNATIONAL UNIVERSITY (UIU)

Dept. of Computer Science & Engineering

Course No: CSE 4326

Course Title: Microprocessors and Microcontrollers Laboratory

Experiment 2: Wifi Communication and building IoT-based systems Using Arduino IoT Cloud

Objectives:

The objective of this lab experiment is to set up an ESP32 board to monitor gas sensor data and control an LED using the Arduino Cloud IoT platform. This comprehensive exercise encompasses multiple key steps, starting with the setup of the ESP32 device on the Arduino Cloud IoT platform, which involves device registration and the acquisition of essential credentials. Subsequently, students create a virtual "Thing" on the platform, linking it to the ESP32 and defining variables for data exchange, namely Gas_ppm and LED. Ensuring secure data transmission, students configure network credentials and incorporate the generated secret key. The lab progresses to programming the ESP32, enabling it to process gas sensor data and manage the LED using digital control logic. Finally, students design a user-friendly dashboard within the Arduino Cloud IoT platform, integrating widgets for real-time sensor data visualization and remote LED control. Final outcomes of this experiment will be-

- Proficient ESP32 setup for gas sensor monitoring.
- Secure IoT data transmission using Arduino Cloud.
- Effective use of virtual "Thing" for data exchange.
- Competent programming for sensor data processing and LED control.
- User-friendly IoT dashboard design for real-time data visualization.

Components required:

Hardware:

- 1) ESP-32 Development Board
- 2) LED
- 3) MQ-2 Gas Sensor
- 4) Breadboard
- 5) Jumper wires
- 6) 10k ohm Resistor

Software:

Arduino Cloud IoT

Theory:

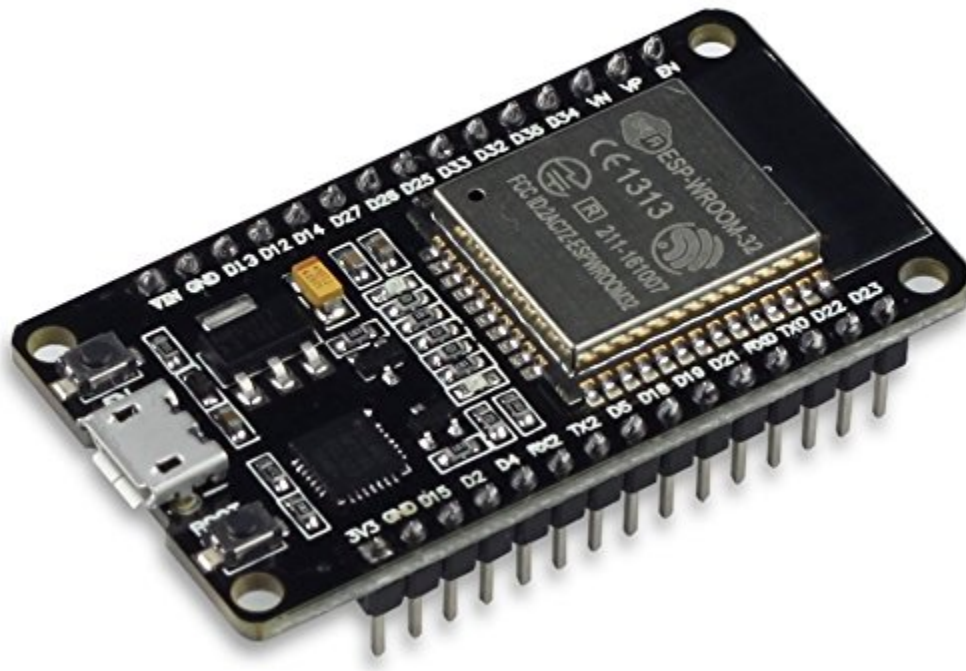


Fig. 1: ESP32

The ESP32 is a widely-used System on a Chip (SoC) that plays a key role in developing Internet of Things (IoT) devices. It boasts a range of features that make it a preferred choice for various IoT applications.

- **Built-in Connectivity:** The ESP32 comes equipped with both WiFi and Bluetooth capabilities, making it suitable for IoT devices that need to transmit data over the internet to operate

- **Security Requirements:** The ESP32 offers robust hardware encryption support, which is crucial for ensuring secure communication.
- **Dual Cores:** With two processing cores, the ESP32 enables multitasking and fast parallel processing, making it ideal for handling real-time applications in embedded systems and IoT devices.
- **Storage:** With 1.5 MB of SPIFFS storage, the ESP32 can efficiently store and manage files on external flash memory. This is particularly useful for IoT applications where data storage is a requirement.
- **RAM:** The ESP32 boasts 520 KB of RAM, which is essential for handling data and operations efficiently, especially in IoT devices.
- **Sensor Compatibility:** The ESP32 interfaces seamlessly with a wide range of IoT sensors, supporting UART, I2C, and SPI communication. It features ADC(Analog to digital conversion) and pulse counting for data acquisition and is compatible with capacitive touch sensors, ensuring adaptability to diverse sensor types.

For more information about the ESP32 [click here](#).

Pin Diagram of ESP32:

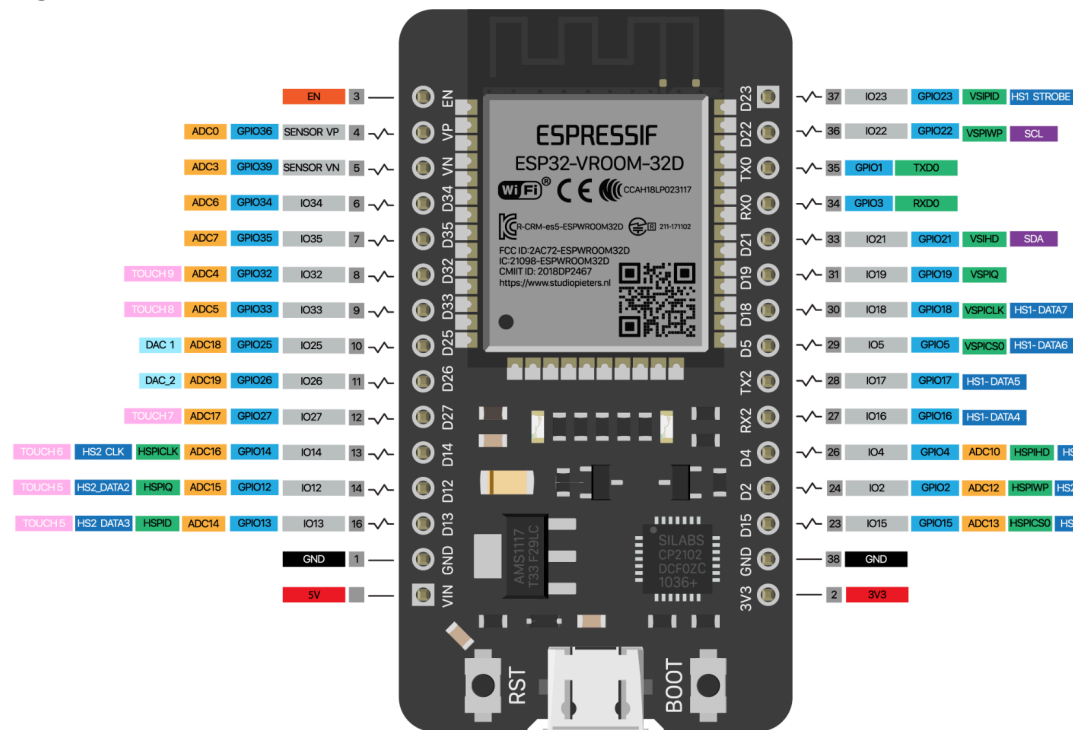


Fig. 2: ESP32 Pin Diagram

The ESP32 peripherals include:

- 18 Analog-to-Digital Converter (ADC) channels
- 3 SPI interfaces
- 3 UART interfaces
- 2 I2C interfaces
- 16 PWM output channels
- 2 Digital-to-Analog Converters (DAC)
- 2 I2S interfaces
- 10 Capacitive sensing GPIO's

Here, the ADC and DAC features are associated with specific static pins. However, UART, I2C, SPI, PWM, etc are assignable independently by code. A more detailed description of each pin is [found here](#).

In this experiment, we're going to explore how to create an IoT device using the versatile ESP32 microcontroller, an MQ2 gas sensor, and a simple LED. Our goal is to detect smoke using the MQ2 sensor and seamlessly transmit this data to the Arduino IoT Cloud platform. Moreover, we'll implement a convenient remote control feature via the cloud dashboard to toggle the LED

on and off. This experiment will provide you with valuable insights into the fundamental principles of IoT technology and hands-on experience in building IoT applications.

Circuit:

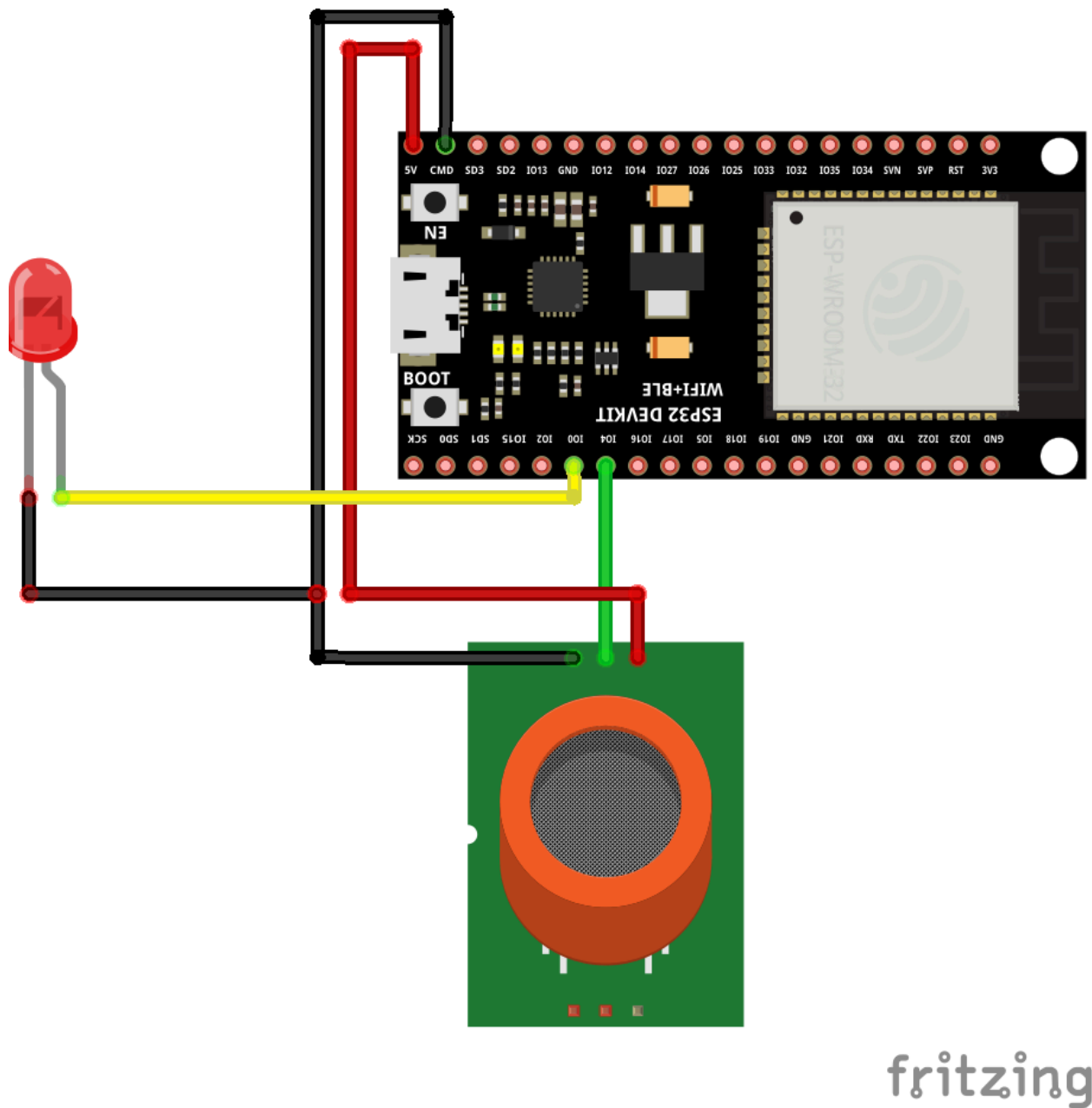


Fig. 3: Circuit

In this setup, you'll connect the VCC pin of the MQ2 sensor to the VIN pin of the ESP32. The A0 pin of the MQ2 will be linked to D4, and you'll connect the positive pin of the LED to D2 on

the ESP. Finally, the negative pin of the LED and the GND pin of the MQ2 will be connected to the GND of the ESP32.

Arduino Cloud IoT:

The Arduino IoT Cloud is an online platform that makes creating, deploying, and monitoring IoT projects easy.

To start, we will need to head over to the Arduino Cloud IoT and then Sign in to the Arduino account.

To sign up, follow the steps below:

Step 01: Sign In or Create Account

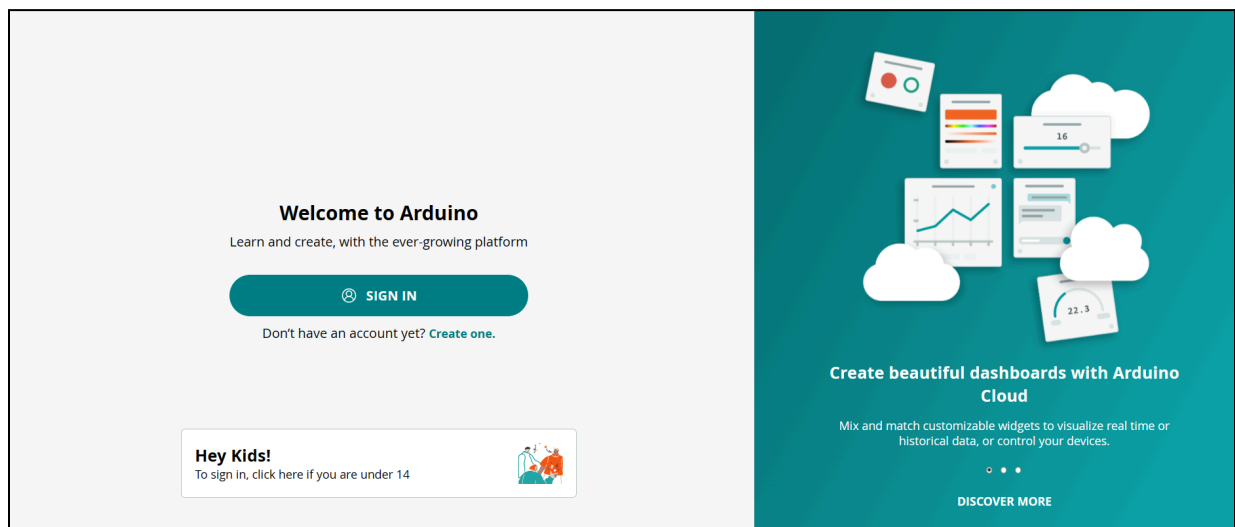


Fig. 4: Arduino IoT Cloud Sign In Page

To create an account, click on [Create one](#). Or if you already have an account, you can just simply click on the [Sign In](#) button as shown in *Fig. 4*. After Signing in, you will land to the page below shown in *Fig. 5*.

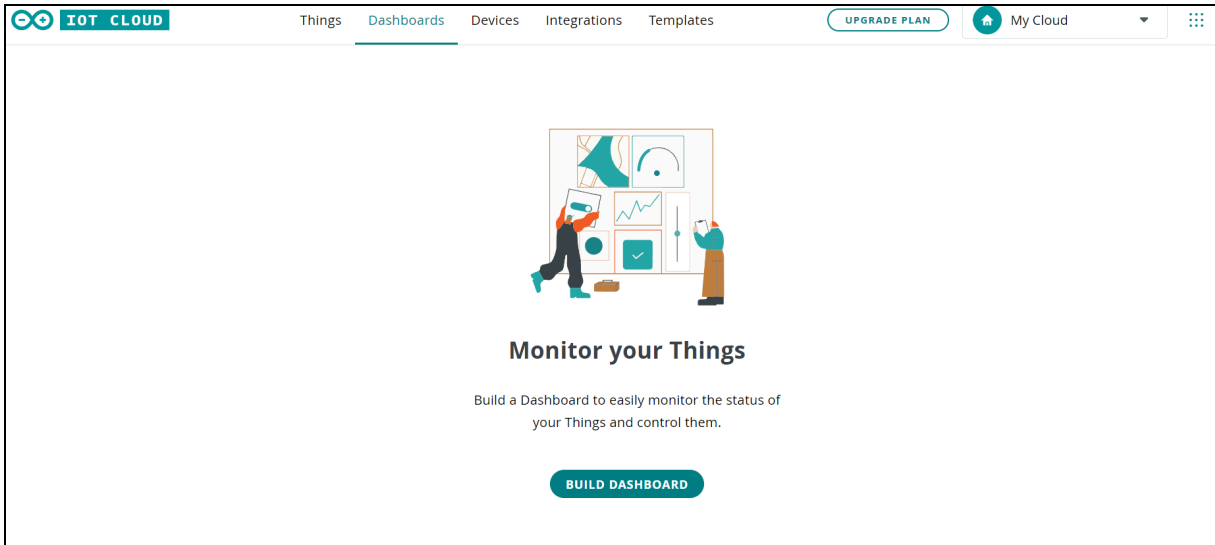


Fig. 5: Arduino IoT Cloud Home Page

Step 02: Setting up the device

1. In the Arduino Cloud IoT interface, click on the "[Devices](#)" tab as shown in Fig. 6.

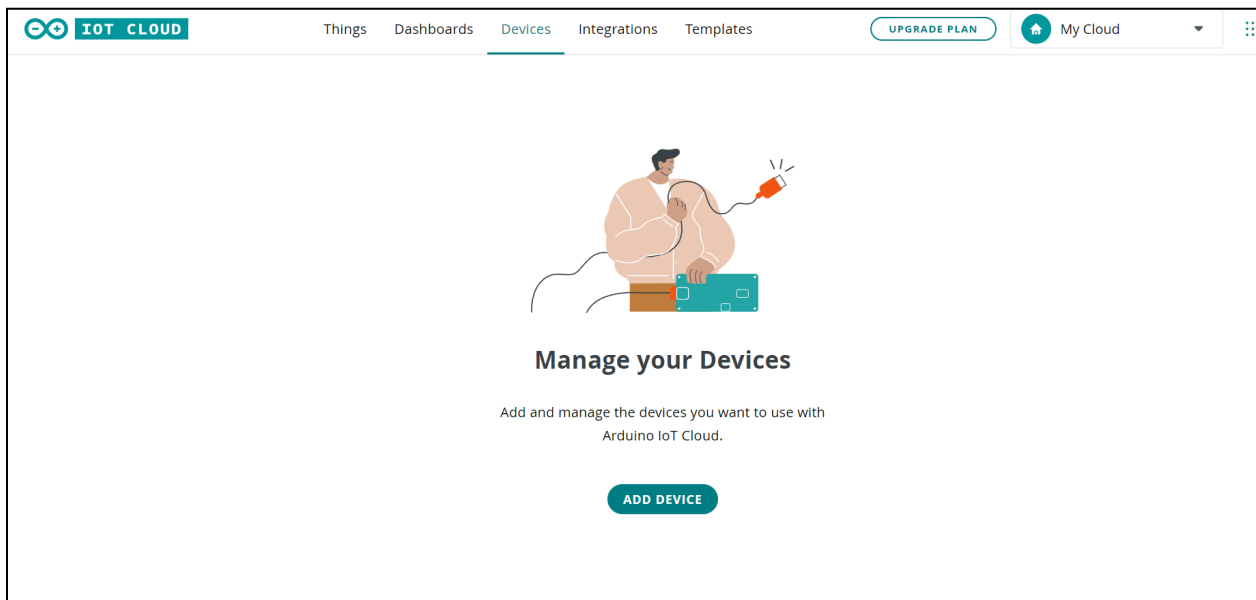


Fig. 6: Device Tab in the arduino cloud website

2. Click the "Add device" button in this "device tab" as shown in Fig. 6.

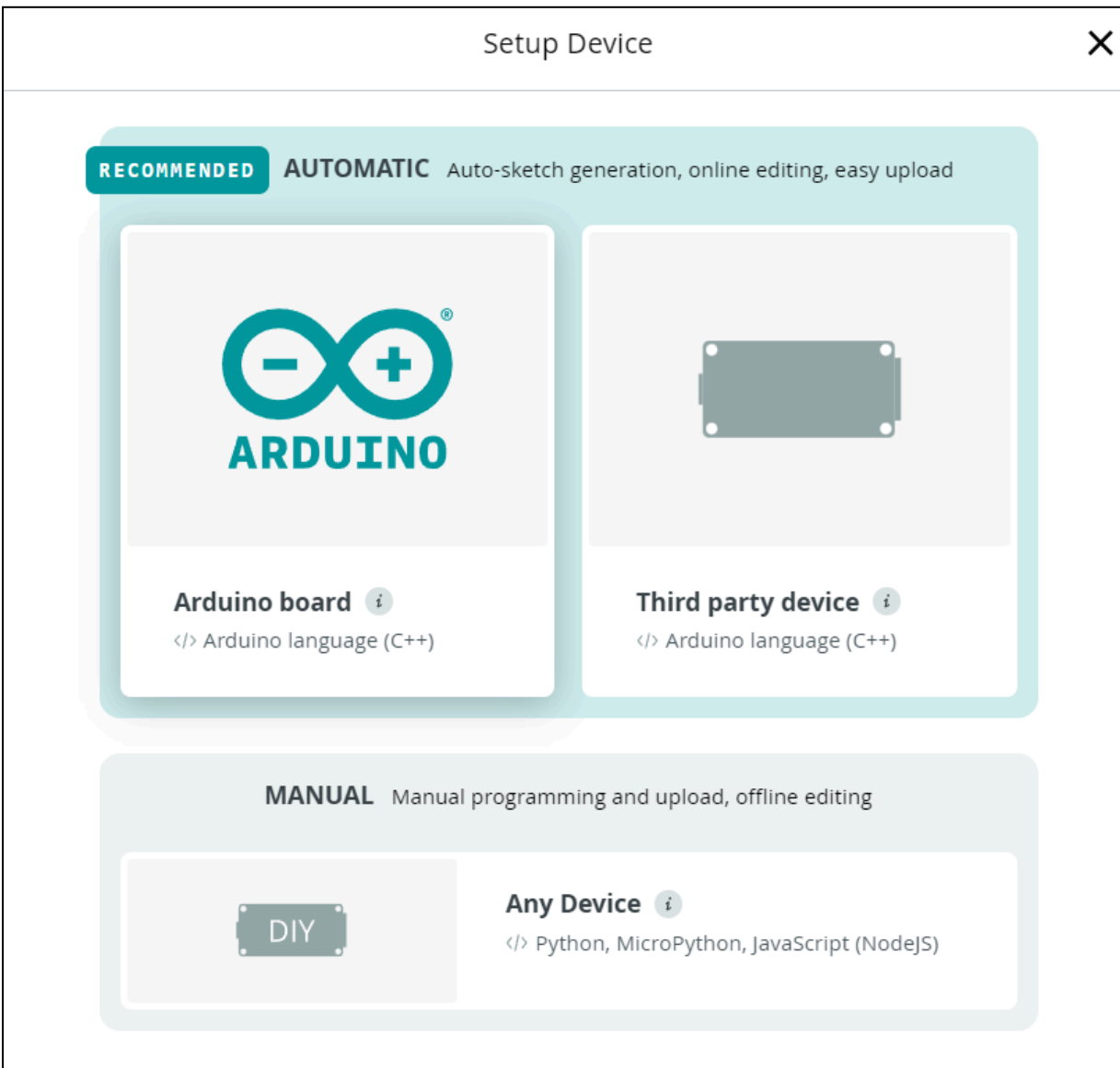


Fig. 7: Device Setup Page

3. Select "Set up a 3rd party device" as shown in Fig. 7.

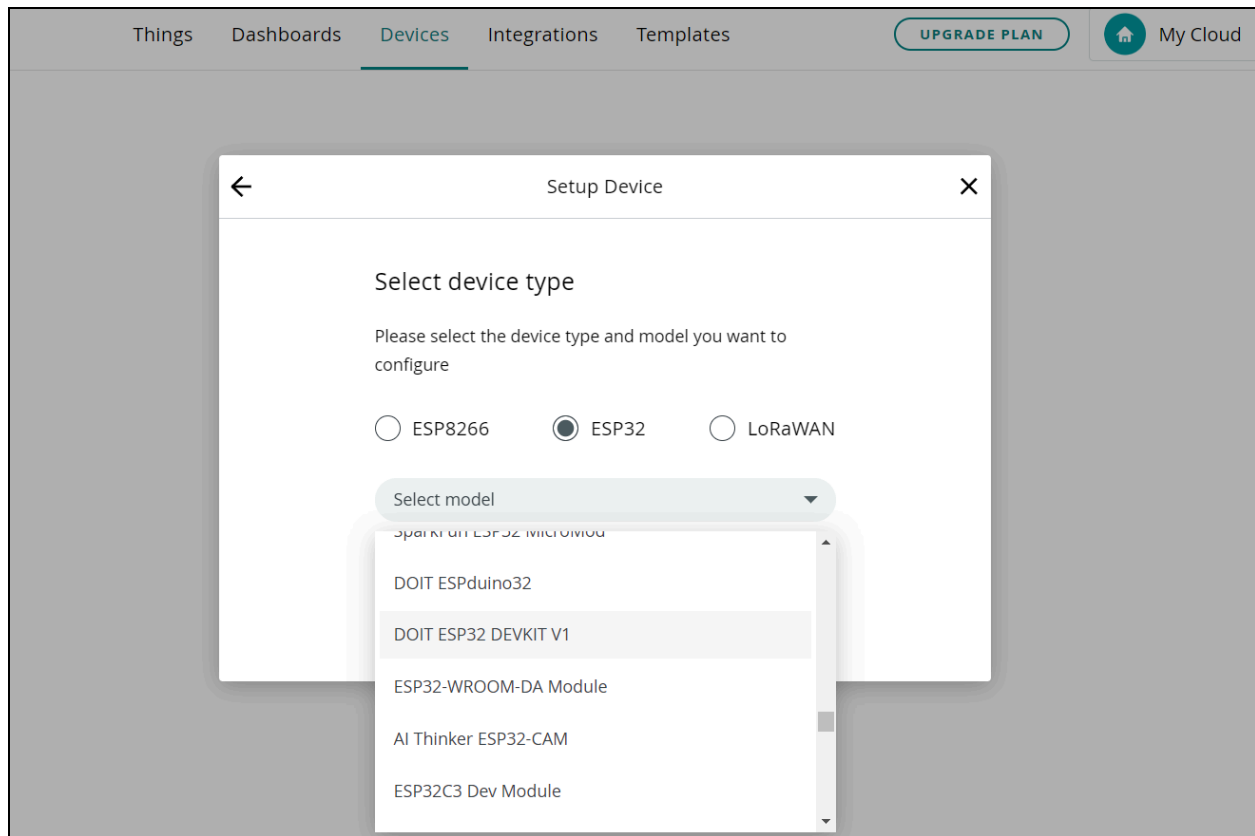



Fig. 8: Selecting Board

4. Then, Choose "ESP32" from the drop-down menu for the board selection. Then, Select the model " DOIT ESP32 DEVKIT V1" as shown in Fig. 8.

←

Setup Device

×



Give your device a name

Name your device so you will be able to recognize it.

Device Name

esp32


↻

NEXT

Fig. 9: Setting Up Device Name

- Now the board needs a name (Fig. 9). We chose *esp32*, but you may use any other creative name!

Setup Device




Make your device IoT-ready

To use this board you will need a Device ID and a Secret Key, please copy and save them or [download the PDF](#).

Also, keep in mind that this device authentication has a lower security level compared to other Arduino devices.

Device ID
cd11caff-43ef-4f97-87f8-59cd6f1a8f27

Secret Key
PUSD98JSHTT1BNYYPK0X

**Secret key cannot be recovered**
Please keep it safe, if you lose it you will have to delete and setup your device again.

☐ I saved my device ID and Secret Key

CONTINUE

Fig. 10: Secret Key for the esp32 device.

6. You will now receive your Device ID and Secret key as shown in Fig. 10. Please note that the *secret key cannot be recovered*, so make sure you note it down. You can also download a PDF with the information.
7. After saving it, tick the box at the bottom, and click on "Continue".

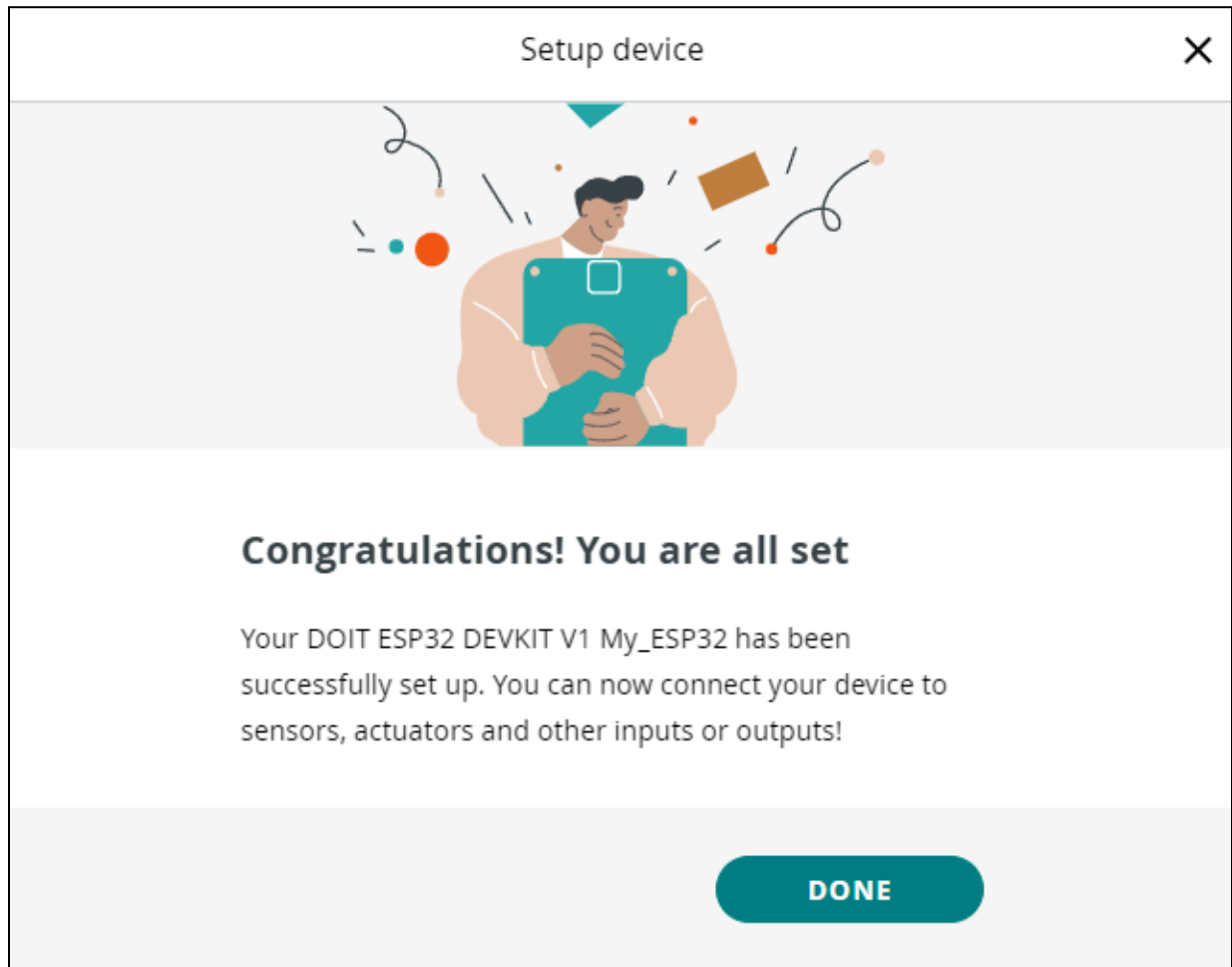


Fig. 11: Device Setup

By doing that, You have now configured your ESP32 board with the Arduino Cloud IoT. Your device will now appear in the list of devices.

Step 03: Creating a Thing

1. To create a Thing, you need to click on the "[Things](#)" to navigate to the “Things” tab as shown in Fig. 12

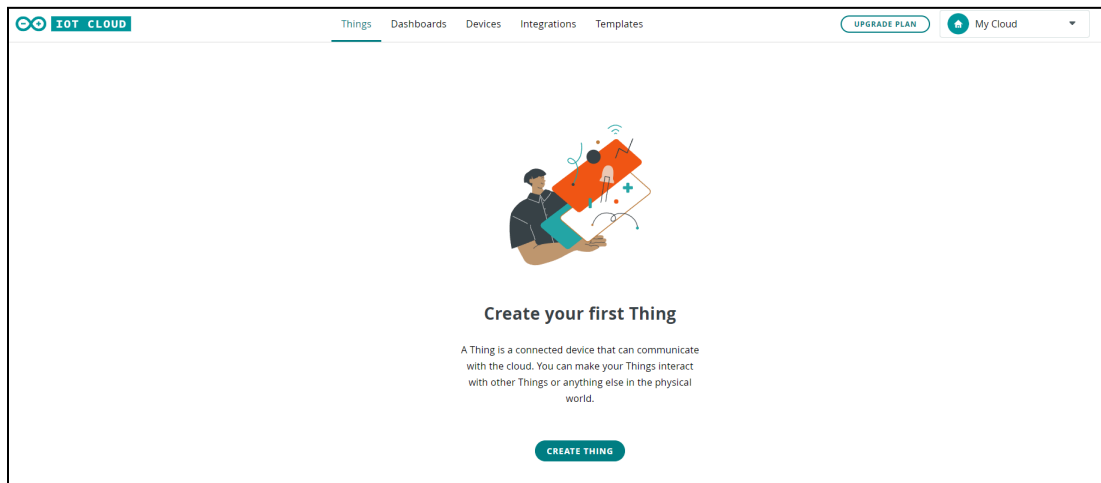


Fig. 12: Thing Tab

2. Then, click on the "Create Thing" button. We can then rename our Thing to something appropriate to what we are doing, in this case, we simply chose **Gas Data**. In this **Gas Data** things (Fig. 13), you will find **setup** and **sketch** options. You can think of **Gas Data** as a project where you can declare variables, add codes and upload it just like in regular arduino!

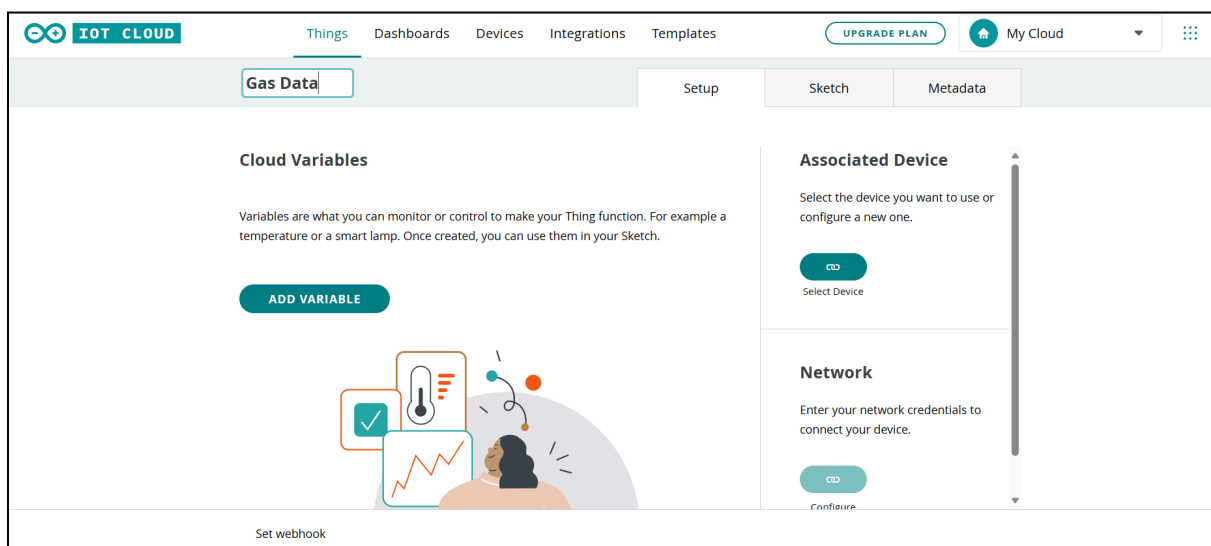


Fig. 13: Giving your project a name in the Things tab

3. Now, we need to link our device with our Thing. This is done by navigating to the "Device" section. This will open a window, where your ESP32 should be available to select as shown in Fig. 14.

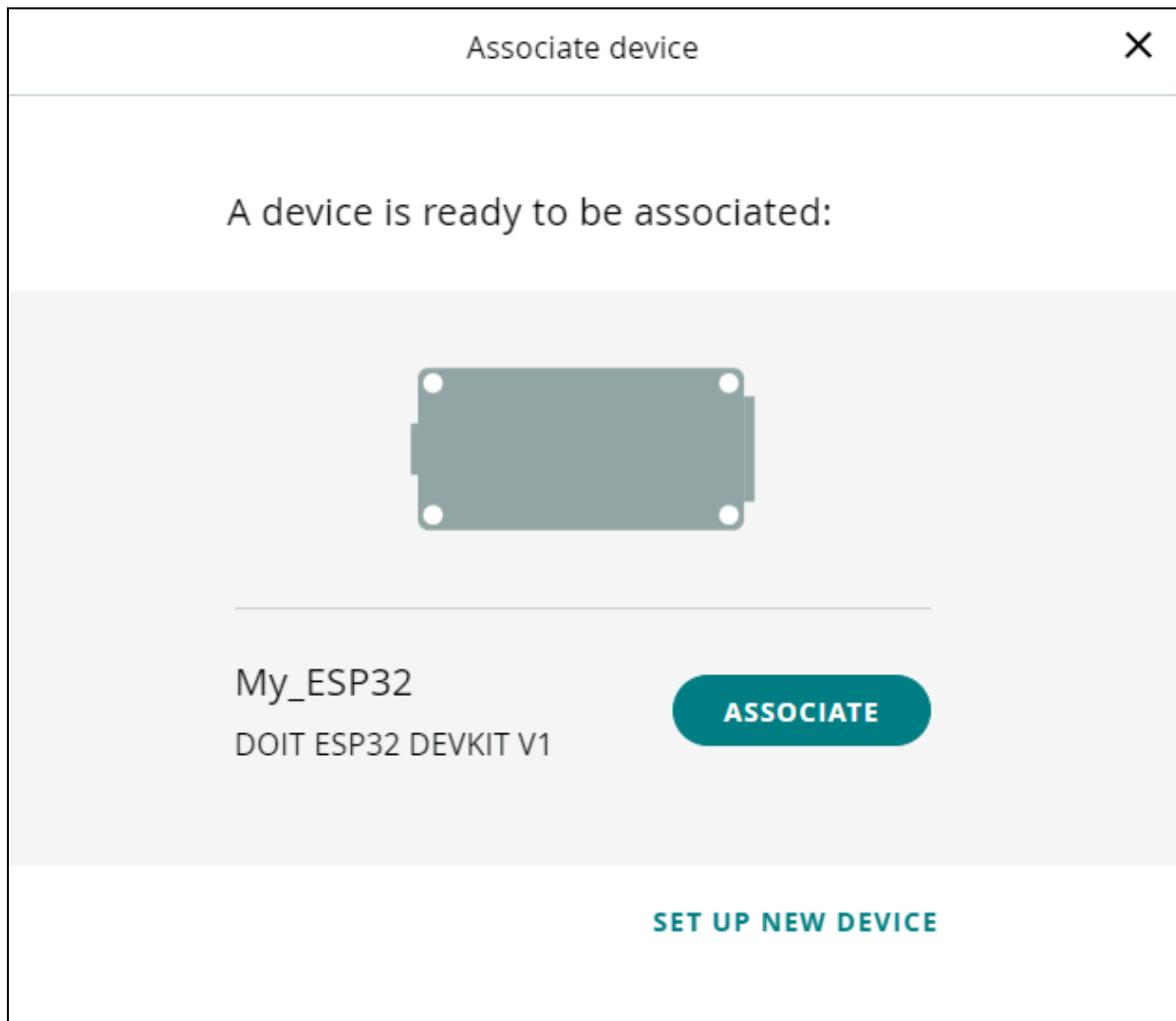


Fig. 14: Associate Device

4. Once the device is linked, we need to create two variables: ***Gas_ppm*** and ***LED***.
5. Click on the "Add variable" button. This will open a window where you need to fill in variable information.
 - I. Let's create the ***Gas_ppm*** first (Fig. 15). The data type is *int*, permission is *read-only* and the updated policy is *Periodically every 3s*. (Here ESP32 will send gas sensor data

every 3s) Once done, click on the "Add variable" button.

Add variable

Name

Gas_ppm

Sync with other Things

Integer Number eg. 1

Declaration

int gas_pp ;

Variable Permission

Read & Write

Read Only

Variable Update Policy

On change

Periodically

Every

3

s

ADD VARIABLE

CANCEL

Fig. 15: Setting Up Gas_ppm variable in the things.

II. Now, let's also add the **LED** variable. The data type for this variable is *boolean*, the permission is *read & write*, and the update policy is *on change*. Once done, click on the "Add variable" button.

The screenshot shows a modal window titled "Add variable" with a close button (X) in the top right corner. The form contains the following elements:

- Name:** A text input field containing "LED".
- Sync with other Things:** A button with a circular arrow icon and an information icon (i).
- Data Type:** A dropdown menu showing "Boolean" with a hint "eg. true" and a downward arrow.
- Declaration:** A text input field containing "bool LED ;" with an information icon (i) to its right.
- Variable Permission:** A section header with an information icon (i), followed by two radio buttons: "Read & Write" (selected) and "Read Only".
- Variable Update Policy:** A section header with an information icon (i), followed by two radio buttons: "On change" (selected) and "Periodically".
- Buttons:** At the bottom, there is a teal "ADD VARIABLE" button and a "CANCEL" button.

Fig. 16: Setting Up "LED" Variable in the *Gas Data* Project.

Step 04: Adding credentials

Now, we need to enter the credentials for our network and the secret key generated during the device configuration.

1. Click on the button in the "Network Section" at right bottom as shown in Fig. 17 (marked by red box)

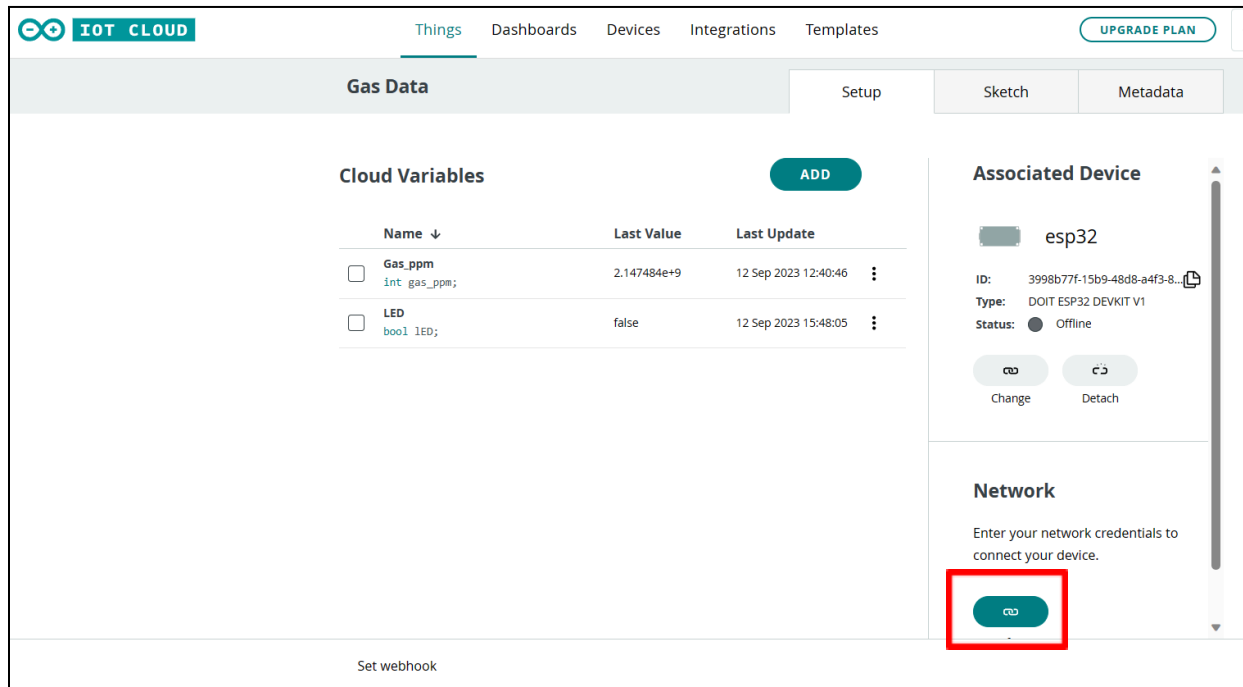


Fig. 17: Set Up Network

2. Then, enter the credentials (network name[Your WiFi you are connected to], network password[Your WiFi password], and secret key[Secret key you got while setting up things]) as shown in Fig. 18.. Click "Save" when finished.

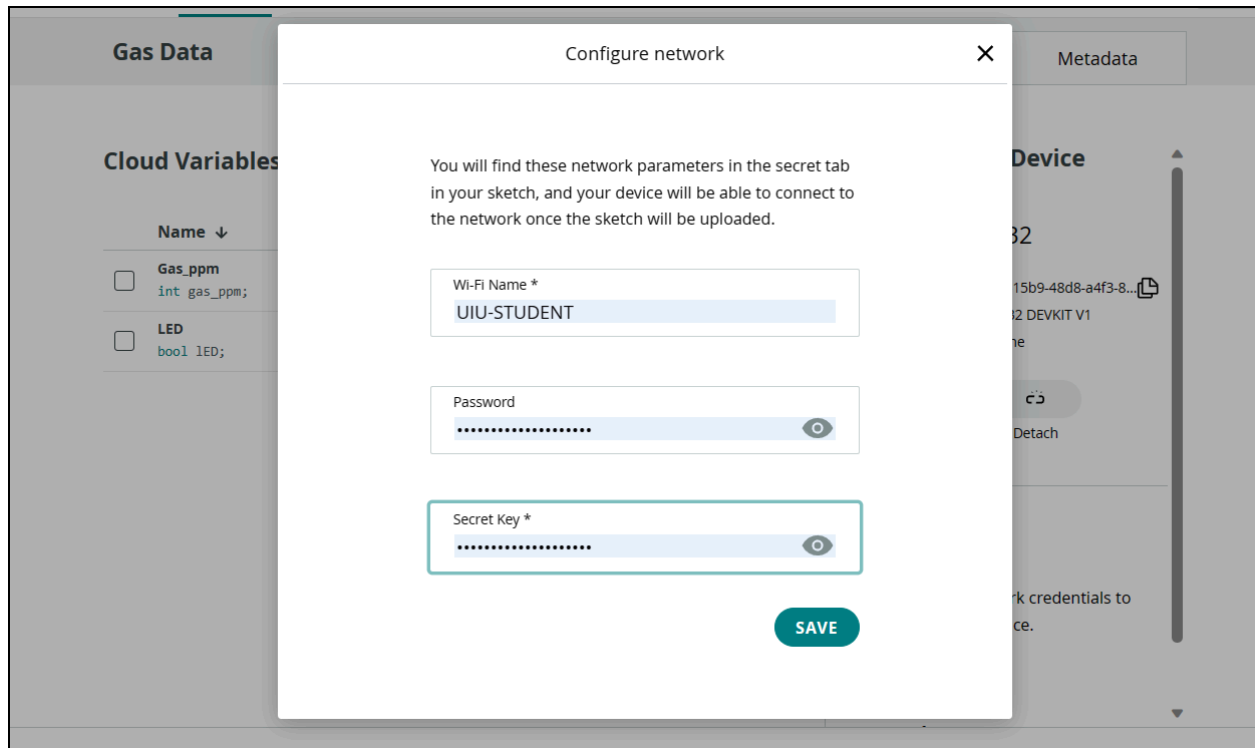


Fig. 18: Entering Network Credentials

3. The next step is to program the board. To do so, we need to go to the "*Sketch*" tab (Fig. 19).

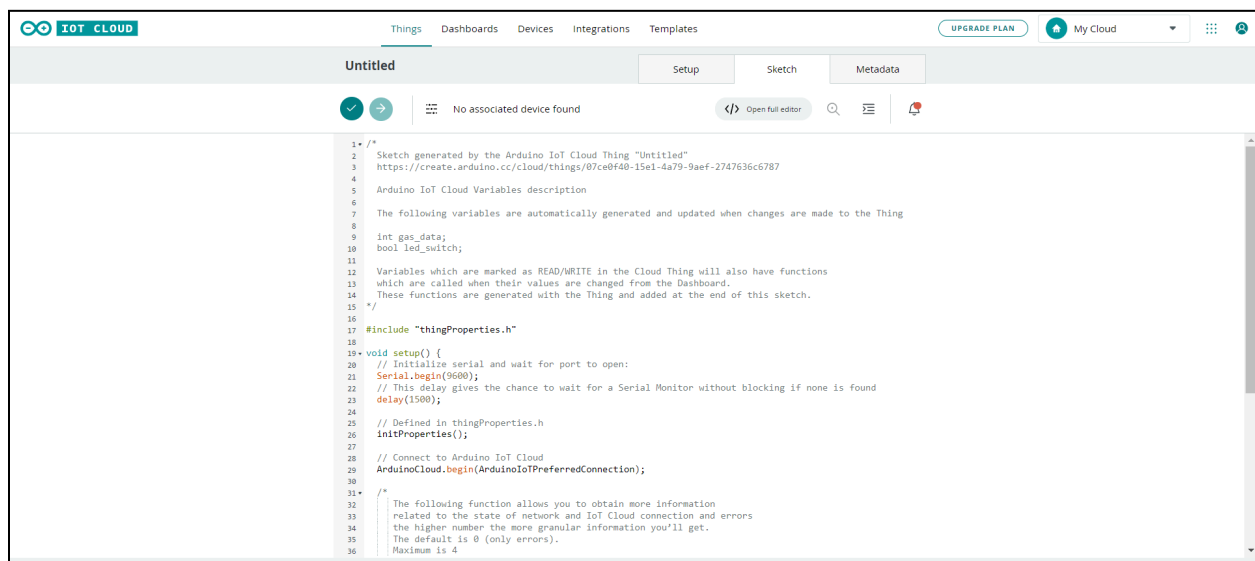


Fig. 19: Write in the Sketch (Code)

[Here all of the functions for the project will be automatically generated for the predefined variables.]

4. We have to add our code here for getting the gas sensor data and controlling the LED. Write the following code, then we will verify, then upload the sketch to your connected ESP32! We will see in the last section on how to connect your ESP32...

Code:

```
#include "thingProperties.h"

// Pin connected to the MQ-2 sensor

const int sensorPin = 4;

int led = 2;

// Define the load resistance value (in ohms) used in the circuit

#define RL 10 //Load resistance

#define m -0.263 //Calculated Slope

#define cb 0.42 //Calculated intercept

#define Ro 20 // Resistance on fresh air

void setup() {

    // Initialize serial and wait for port to open:

    Serial.begin(9600);

    // This delay gives the chance to wait for a Serial Monitor without
    blocking if none is found

    delay(1500);

    // Defined in thingProperties.h

    initProperties();
```

```

// Connect to Arduino IoT Cloud

ArduinoCloud.begin(ArduinoIoTPreferredConnection);

/*

The following function allows you to obtain more information
related to the state of network and IoT Cloud connection and errors
the higher number the more granular information you'll get.
The default is 0 (only errors).
Maximum is 4

*/

setDebugMessageLevel(4);

ArduinoCloud.printDebugInfo();

pinMode(sensorPin, INPUT);

pinMode(led, OUTPUT);

}

void loop() {

  ArduinoCloud.update();

  float VRL; //Voltage drop across the MQ sensor

  float Rs; //Sensor resistance at gas concentration

  float ratio; //Define variable for ratio

  float sensorValue = analogRead(sensorPin);

```

```

    Serial.println(sensorValue);

    VRL = sensorValue * (5.0/1023.0); //Measure the voltage drop and convert
to 0-5V

    Rs = ((5.0*RL)/VRL)-RL; //Use formula to get Rs value

    ratio = Rs/Ro; // find ratio Rs/Ro

    float ppm = pow(10, ((log10(ratio)-cb)/m)); //use formula to calculate
ppm

    Serial.print("PPM: ");

    Serial.println(ppm);

    gas_ppm = ppm;
}

/*

    Since LED is READ_WRITE variable, onLEDChange() is
    executed every time a new value is received from IoT Cloud.

*/

void onLEDChange() {

    if (LED == 0) {

        digitalWrite(led, HIGH);

    }

    else

    {

        digitalWrite(led, LOW);

```

```
}  
  
}
```

Step 05: Creating a dashboard

Now that the sketch is written on the ESP32 board, we can move on to the final step, which is creating a dashboard.

1. Navigate to the "[Dashboards](#)" tab as shown below in Fig. 20

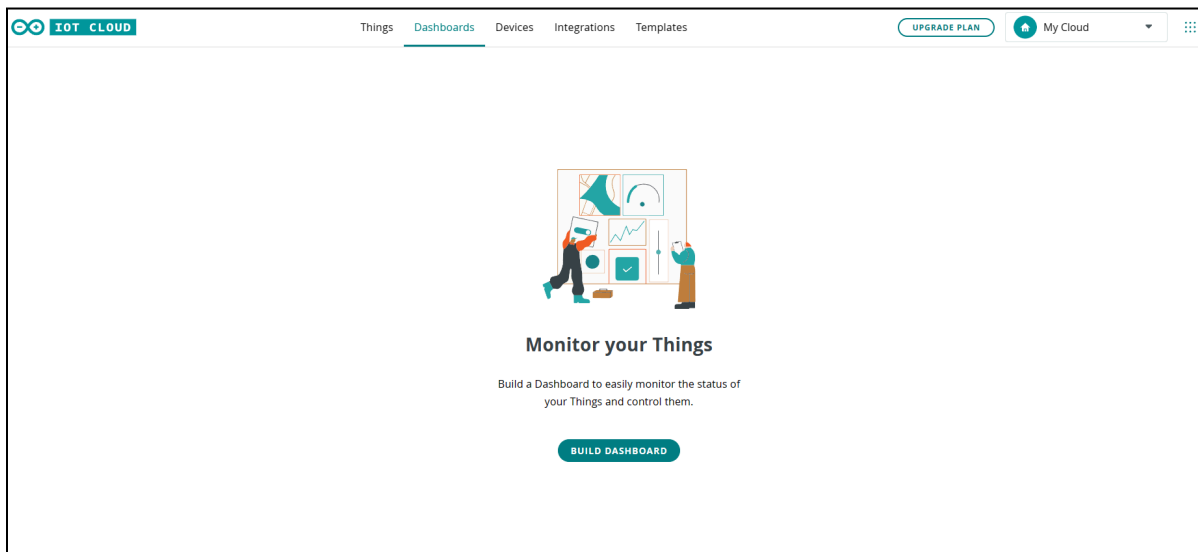


Fig. 20: Build Dashboard

2. Then, click on the "*Build dashboard*" button. A dashboard will appear as shown in Fig. 21. Here, you can observe your variables value, can control various parameters and can observe real time data!

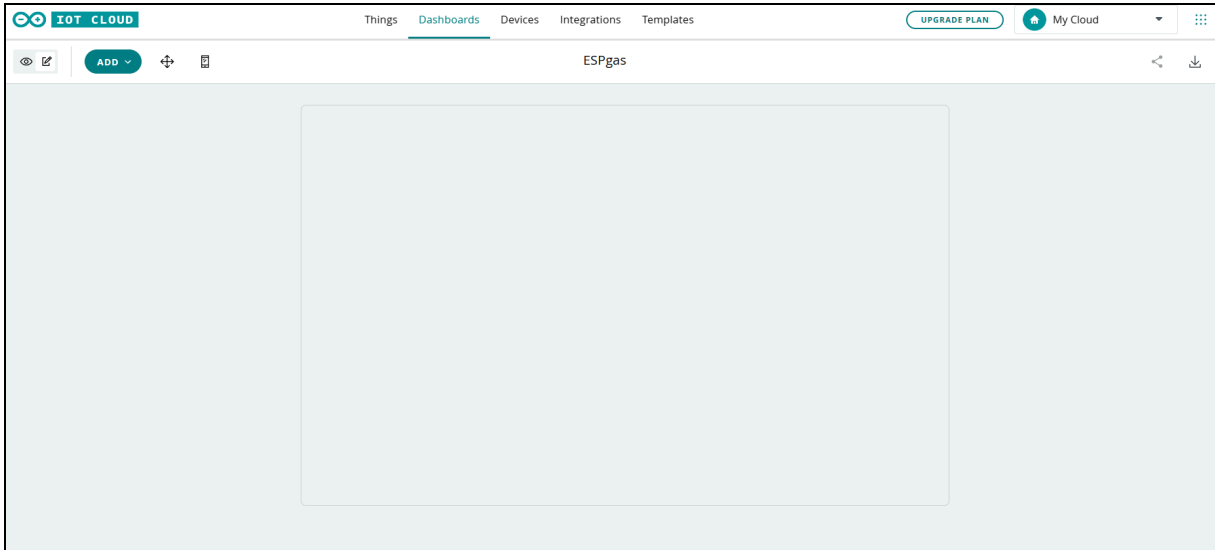


Fig. 21: Dashboard

3. To edit the dashboard, click on the pencil icon at the top left icon, select "*widgets*" and search for your Thing.

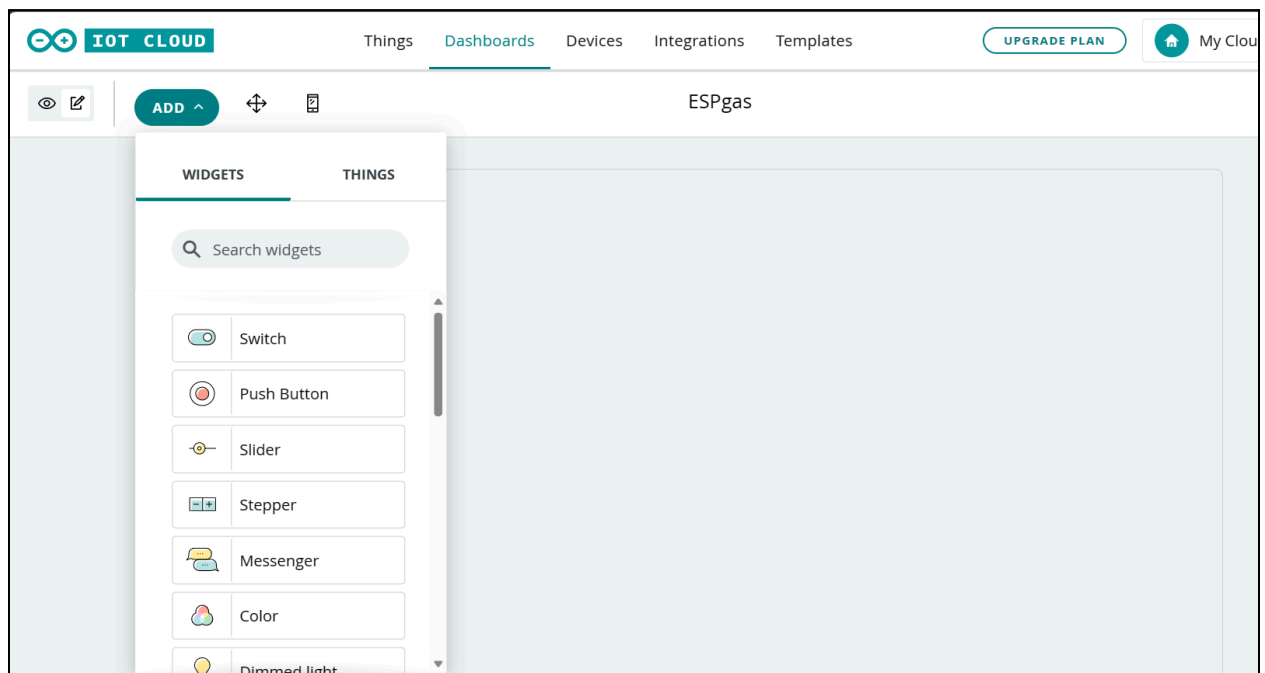


Fig. 22 : Add Widgets

- I. First of all, add a *switch*.

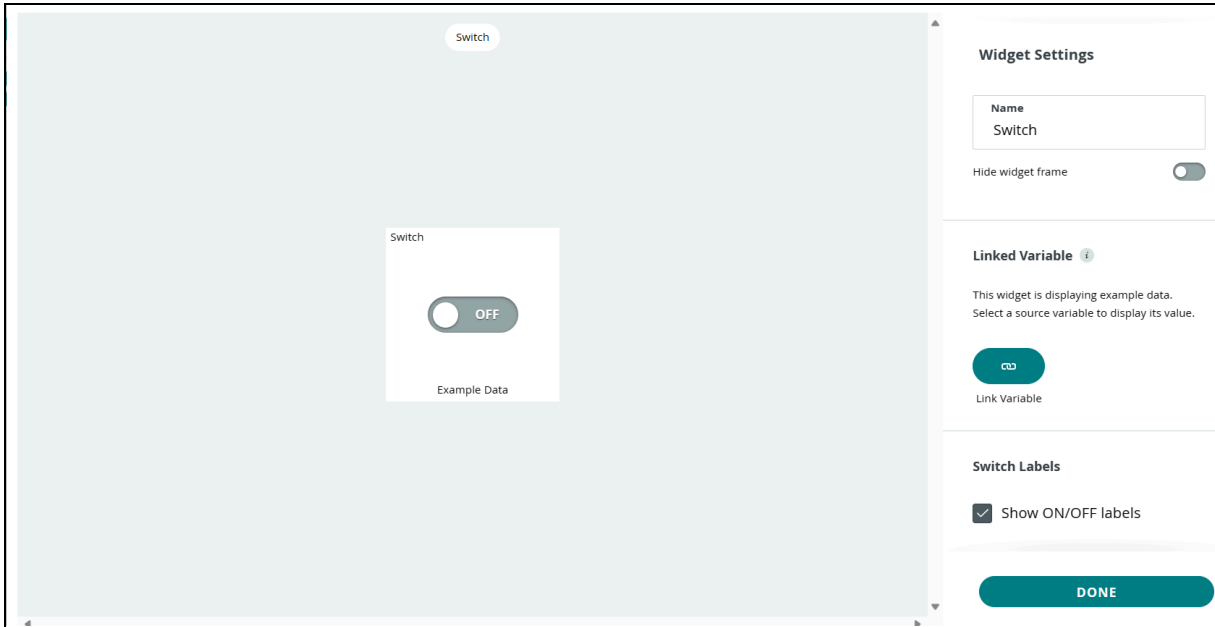


Fig. 23: Widget-Switch

- a) Click the link variable and link the LED variable

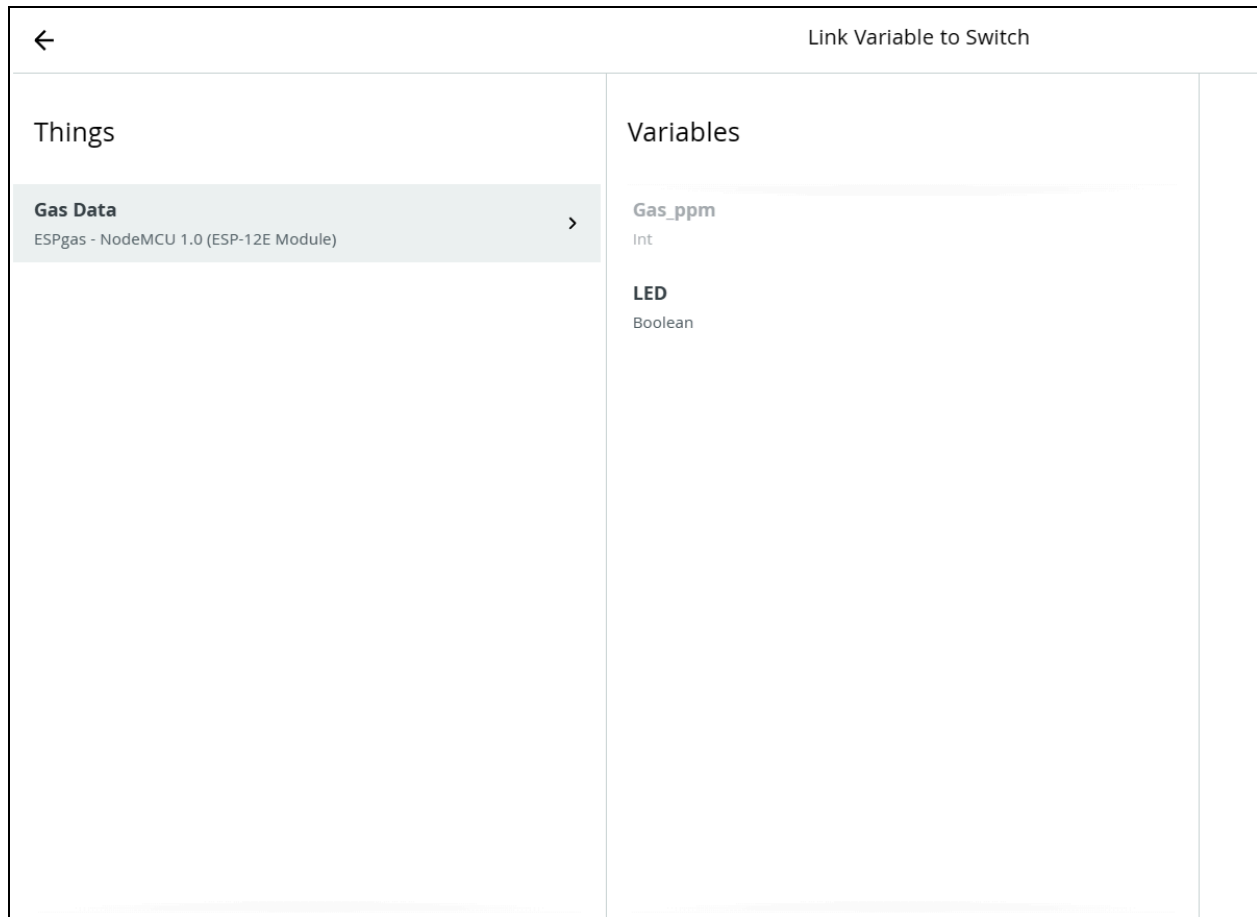


Fig. 24: Link Led

II. Then add a “*Chart*” widget and link variable to link the Gas_ppm

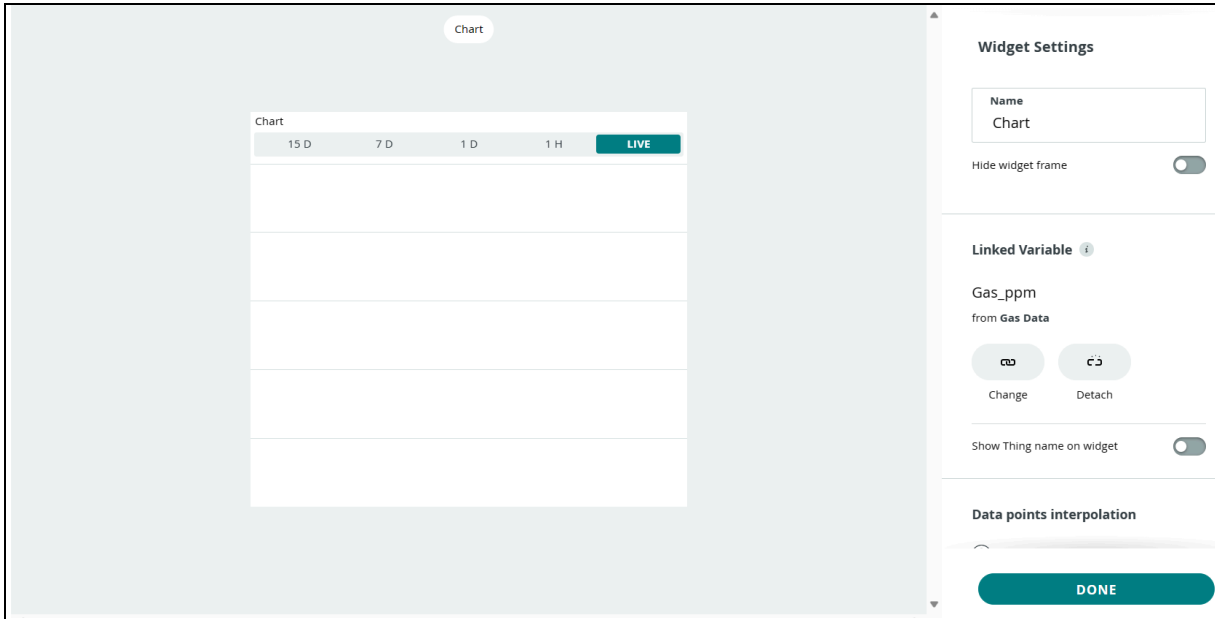


Fig. 25: Chart-Widget

III. Lastly, add “*Value*” Widget and link it to Gas_ppm

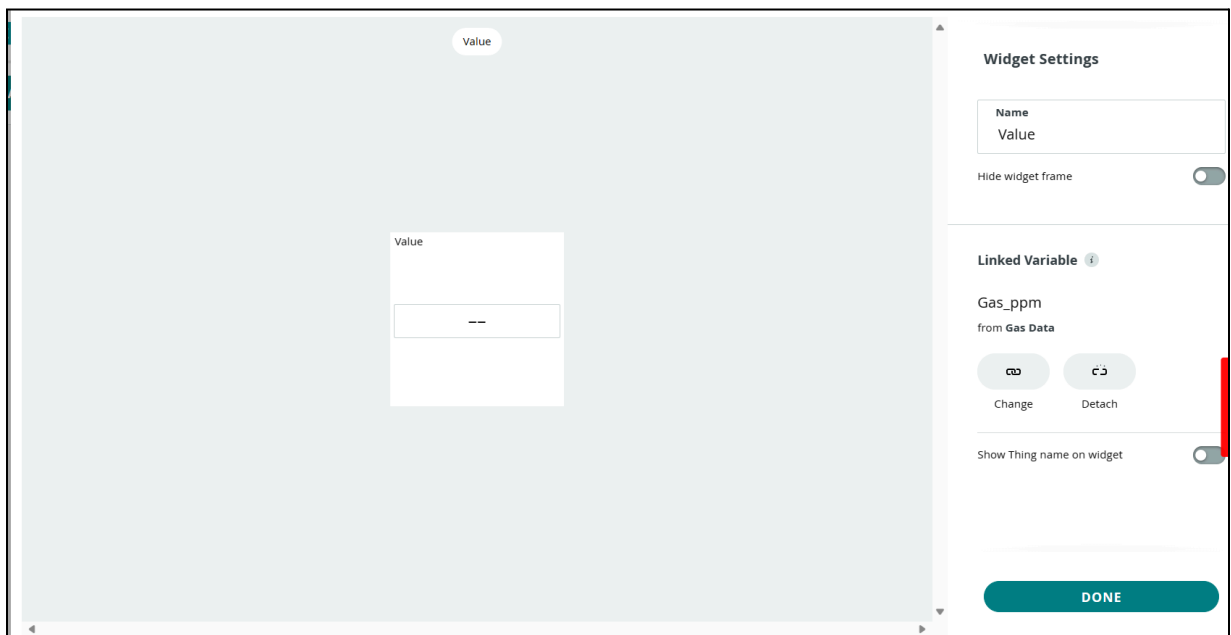


Fig. 26: Value-Widget

[You can arrange the widgets as desired on the dashboard interface.]

4. Customize the appearance and settings of the widgets as needed. Finally, the dashboard should look like this:

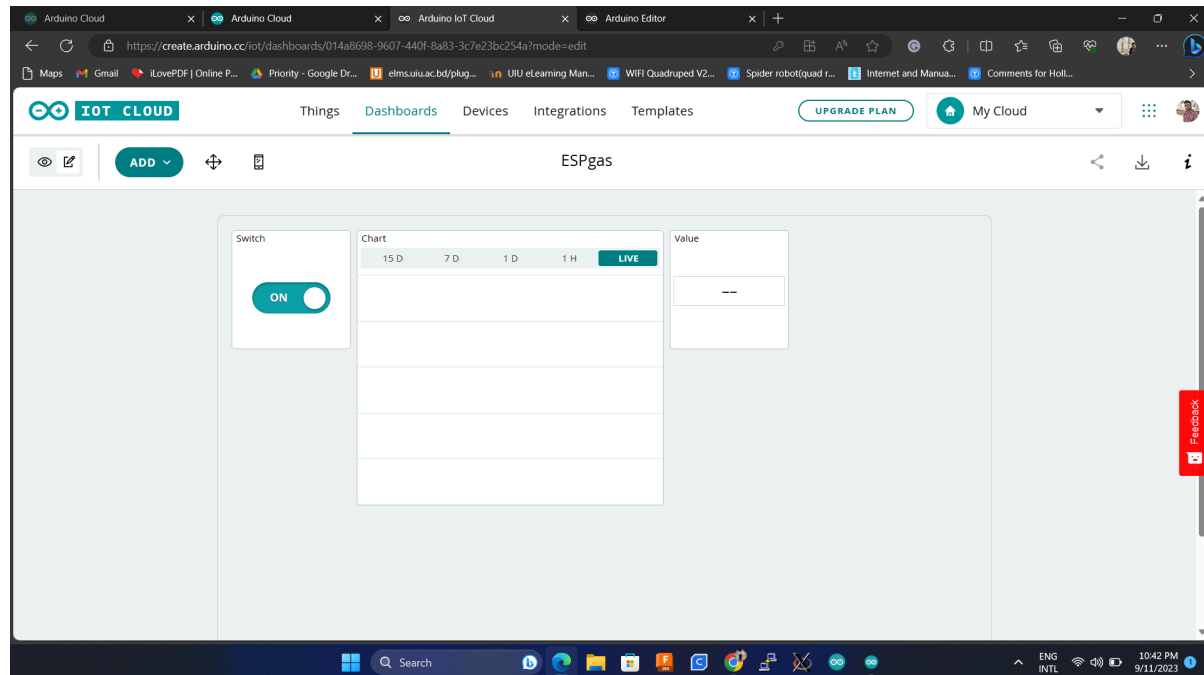


Fig. 27: Widget Added Dashboard

Upload code to the Board:

Now, finally it's time to upload our code to the ESP32 board and make the IoT device operational, follow these steps:

1. Begin by installing the [Arduino agent](#).
2. Next, connect the ESP32 to your computer. If you encounter an issue that says '*No associated device found*,' shown in Fig. 28 please [click here](#) to download and install the necessary [driver](#). Once installed, you should be ready to proceed.

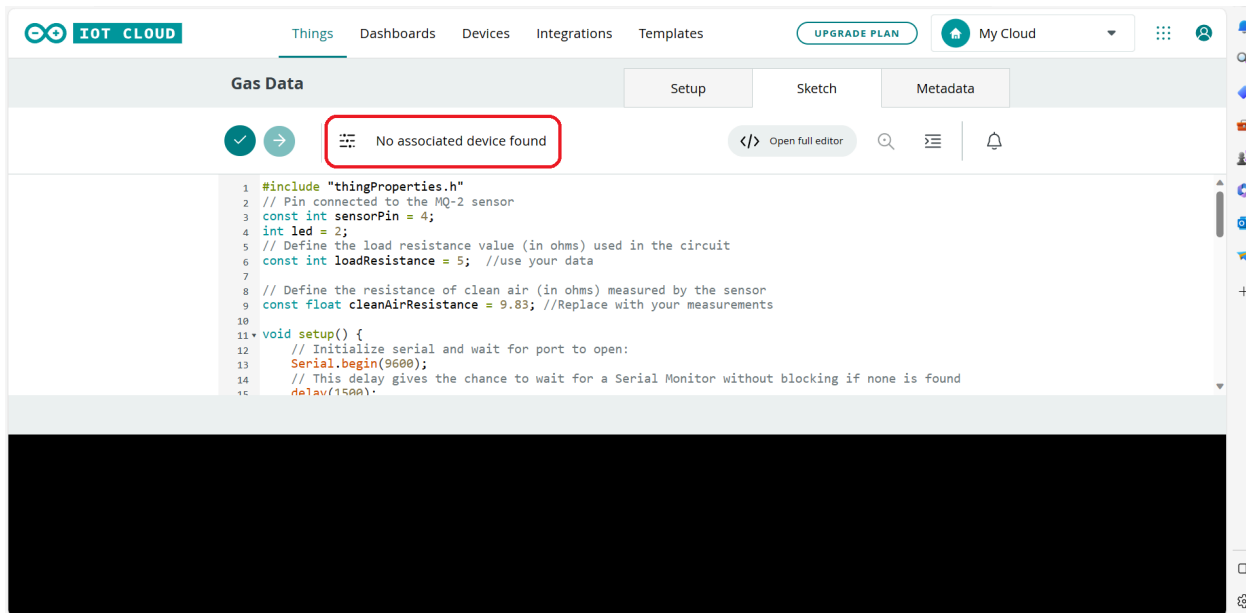


Fig. 28: No Associated Device Found

- To upload the code to the board, click the right arrow button as shown in Fig. 29. It's important to note that you should ***press and hold the boot button and press the EN button too*** on the ESP32, located at the bottom right and left corner shown in Fig. 30, ***then*** you click the code upload button on the screen. Wait for the code to upload while holding the boot button. The ESP32 may ask to install 3 additional drivers this time, notice that and click yes if asked.

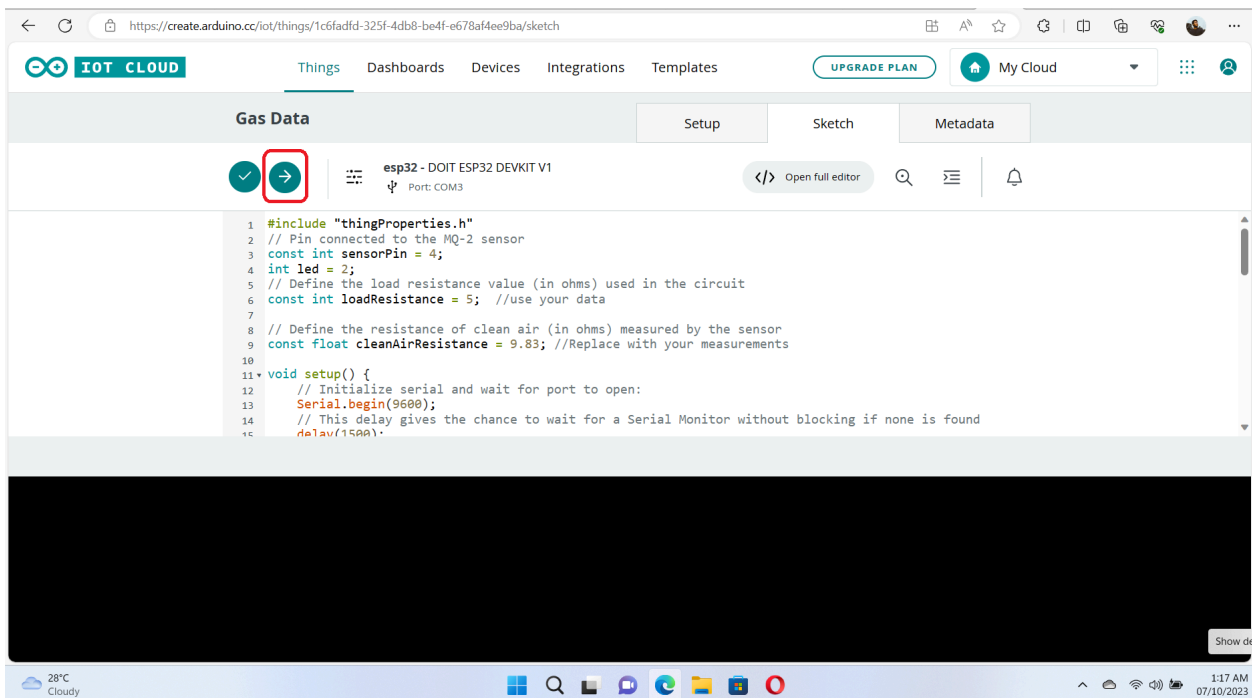


Fig. 29: Upload Button

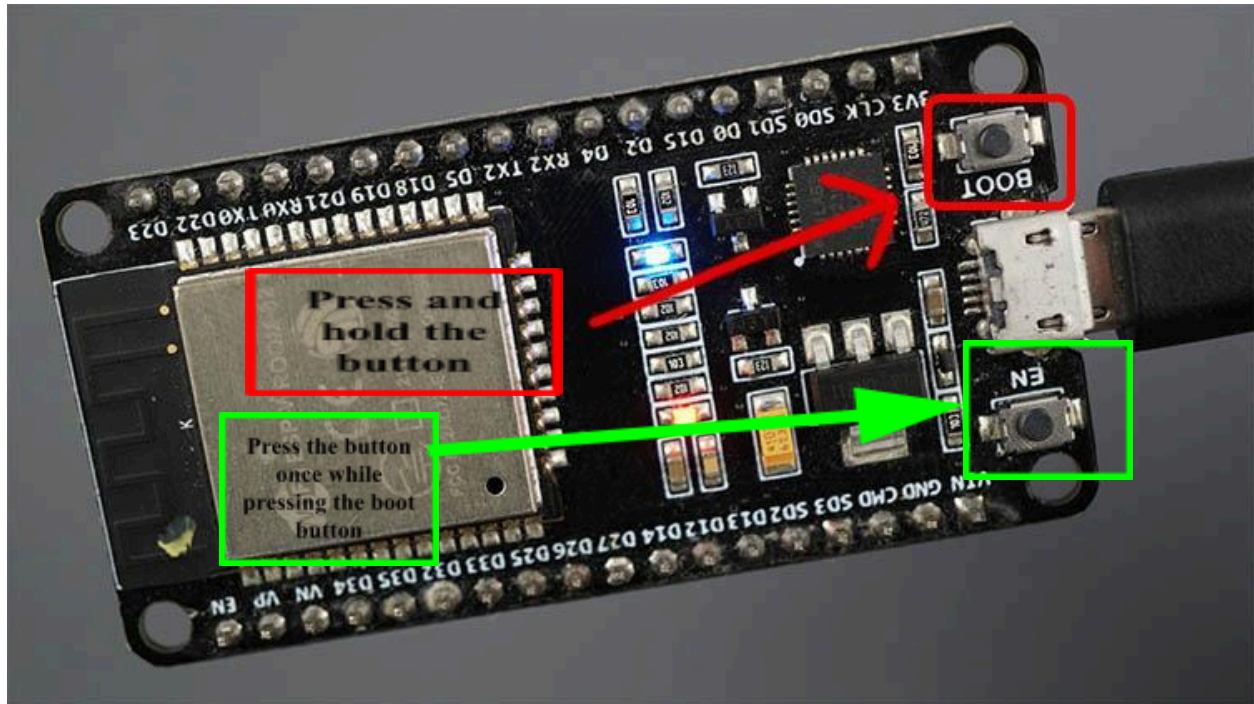


Fig. 30: Boot Button and EN button on ESP32

Local Server:

- 1) ESP32 itself working as a WiFi
- 2) Your sensor connected to ESP32
- 3) Now, you can connect your other devices to your ESP32-WiFi
- 4) Then, login to your devices using the IP address provided by your ESP32 WiFi to see the data from the sensor.

Code:

```
#include "WiFi.h"
#include "ESPAsyncWebServer.h"

#define AO_PIN 35

const char* ssid = "ESP32-Access-Point";
const char* password = "123456789";

AsyncWebServer server(80);
```

```

String readGas() {
    int Gas = analogRead(A0_PIN);
    return String(Gas);
}

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

    WiFi.softAP(ssid, password);

    IPAddress IP = WiFi.softAPIP();
    Serial.print("AP IP address: ");
    Serial.println(IP);

    server.on("/events", HTTP_GET, [](AsyncWebServerRequest *request) {
        request->send(200, "text/event-stream", "data: " + readGas() +
"\n\n");
    });

    server.on("/readGas", HTTP_GET, [](AsyncWebServerRequest *request) {
        request->send(200, "text/plain", readGas().c_str());
    });

    server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
        String page = "<html><head>";
        page += "<style>body { font-family: Arial, sans-serif;
background-color: #f4f4f4; display: flex; align-items: center;
justify-content: center; height: 100vh; margin: 0; }</style>";
        page += "<link rel='stylesheet'
href='https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/a
ll.min.css'
integrity='sha512-Zh2e5lR6b1E7G5kmC5q2/Jz2QHiyyZB49UoMKApSlEKNEz1o6eHA
h0FvsNpMOG/z6b1d00E7QxTUQA9QHJ5oQ=' crossorigin='anonymous' />";
        page += "<script>var eventSource = new EventSource('/events'); "
            "eventSource.onmessage = function(event) {
document.getElementById('temp').innerHTML = '<i class=\"fas
fa-thermometer-half\"></i> Gas: ' + event.data; };</script>";
        page += "</head><body>";
    });
}

```

```

    page += "<div id='temp' style='font-size: 100px; margin: 20px;
text-align: center;'></div>";
    page += "</body></html>";
    request→send(200, "text/html", page);
  });

  server.begin();
}

void loop() {
  // Your loop code goes here
}

```

Data Observation:

1. Access the dashboard from your Arduino Cloud IoT account.
2. Use the switch widget to control the LED on your ESP32 board.
3. Observe real-time gas sensor data displayed on the chart and value widgets.

Conclusion:

This IoT experiment demonstrated the entire lifecycle of an IoT project, from device configuration and variable setup to code implementation and user interface creation. The Arduino Cloud IoT platform served as a powerful tool for managing and controlling our ESP32-based project, highlighting the ease and effectiveness of using cloud-based services for IoT development. This experiment provided valuable hands-on experience in building and managing IoT solutions and can serve as a foundation for more advanced IoT projects and applications.

Lab Reports:

Normal Structure of the lab report (same for all the lab reports):

SL No.	Sections	What to Write
1.	Cover Page	Add a cover page and use the format uploaded in the drive (Lab Report Format (with cover page).docx)
2.	Objective	Write 2-3 lines using points to mention what was the main learning outcome of the report
3.	Components	Write what types of components you used.

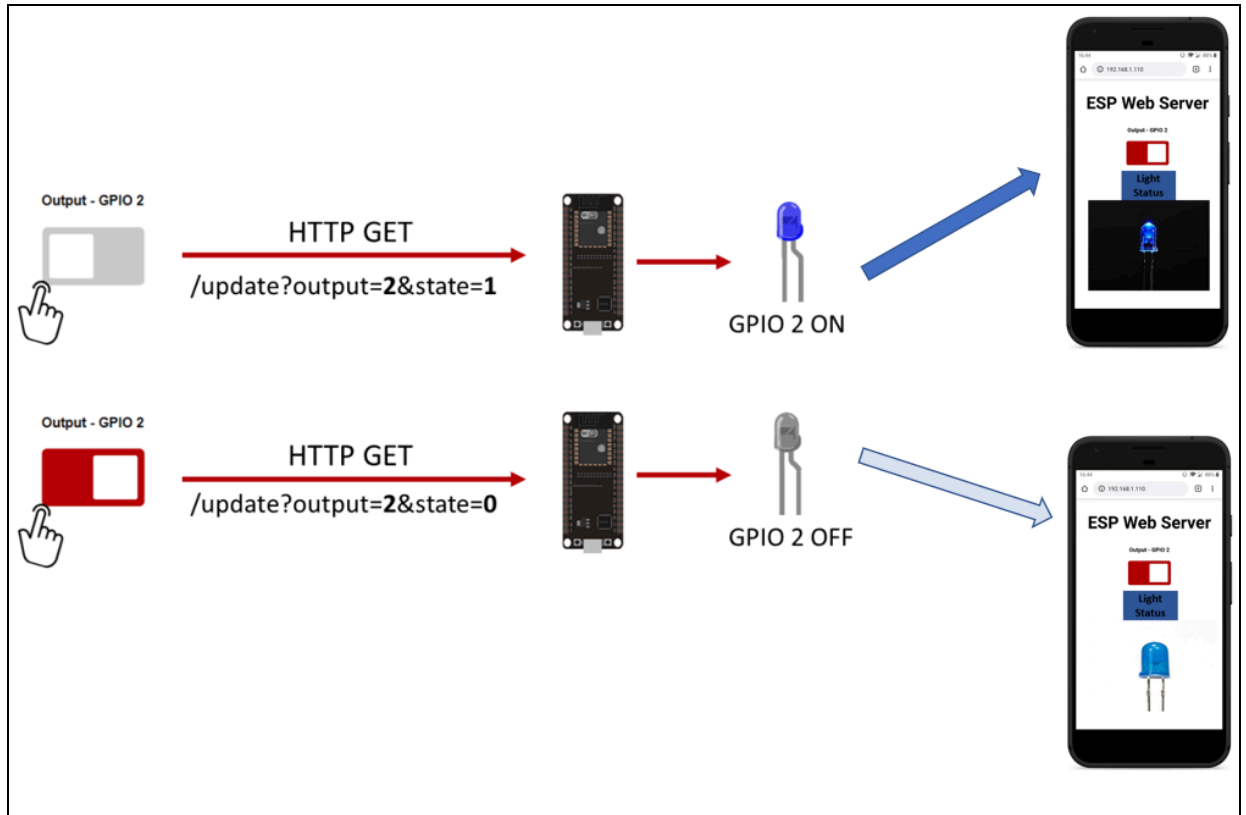
4.	Theory	Write some theoretical background if needed for the problems you are going to solve in this report
5.	Problems mention in the tasks	Solve all the problems, attach circuit diagrams (take screenshots or using camera phone), attach codes in detail. Use comments in the code so that we can understand what did you write
6.	Discussion	Write 5-7 lines on what you learn and what problems you faced when solving these problems.

Provide the relevant circuit connections and their codes for all the problems mentioned below:

- 1) Explain briefly how ESP32 is communicating via Arduino IoT Cloud. Suppose, you are observing the moisture in an agricultural farm using a sensor. You have to send the sensor data to a cloud. Based on the sensor data, the cloud would send a signal to the microcontroller used in the farm to turn on the motor pump. Explain how you can build such a system using the concept we have seen in this experiment. Use a proper block diagram to draw your system.
- 2) Write a program to send "Hello!" and "CSE 4326!" using WiFi to the cloud and then read the data from the cloud. A green and red led light is connected to another ESP32. If "Hello!" is sent then a green led would turn on and if "CSE 4326!" is sent then the red led would turn on.
- 3) Create a local ESP32 server on which two Widgets are shown, one is a button switch which can be pressed to turn ON or OFF. The other widget is a LED image/symbol which shows the status of the LED i.e., when the light in a room is ON, the LED button turns ON else OFF. The web server should look like this:



Now, the LED is connected to the GPIO 2 Pin of the ESP32. When the button on the server is pressed, the LED should turn ON else OFF. The interface of the web server should look like the following:



Provide detailed codes and explanations!

- 4) A gas sensor is connected in your kitchen to measure the gas in ppm. The sensor is connected to ESP32 to send the data in the cloud. Now, write the necessary programs to save the gas data from the cloud in a google sheet or excel file. Then, read from the Excel file and turn on a buzzer if the gas value crosses 50 ppm. You can follow these links:
<https://www.survivingwithandroid.com/integrate-arduino-google-sheets-iot-project/amp/>
https://projecthub.arduino.cc/Arduino_Genuino/arduino-iot-cloud-google-sheets-integration-3c0df1