

# Project Introduction and Implementation

## Contents

1.	Introduction of project.....	3
2.	Function of project.....	3
3.	Novelty of project .....	3
4.	Evaluation of project .....	4
4.1.	Function evaluation.....	4
4.2.	Interface evaluation .....	9
4.3.	Security evaluation.....	9
5.	Architecture .....	10
6.	Circuit Diagram and Hardware .....	10
6.1.	Circuit diagram and connected hardware.....	10
6.2.	Hardware.....	11
7.	Project Code.....	11
7.1.	Device Code.....	11
7.2.	Cloud Code .....	12
8.	Configure Device Code running environment.....	13
8.1.	Install the latest Arduino IDE version .....	13
8.2.	Install the ESP32 board support on Arduino IDE.....	13
8.3.	Install the Azure SDK for C library on Arduino IDE .....	13
8.4.	Install the DHT sensor library on Arduino IDE.....	13
8.5.	Install Adafruit Unified Sensor in the library manager.....	13
9.	Create the Azure IoT Cloud Components.....	13
9.1.	Create a new application in IoT Central Applications .....	13
9.2.	Create my IoT device on Azure IoT Central platform .....	13
9.3.	Create custom device template .....	13
9.4.	Create email alert by using Rules.....	14
9.5.	Create Blob Storage.....	15
10.	Configure wifi and IoT relevant authorization information of device code .....	16
10.1.	Navigate to folder “My_Final_Project_Fire_Detector” .....	16

10.2.	Open “My_Final_Project_Fire_Detector.ino” .....	16
10.3.	Configure “Azure_IoT_PnP_Template.cpp” for template model id.....	16
10.4.	Configure “iot_configs.h” .....	16
10.5.	Configure “Azure_IoT_PnP_Template.cpp” for device information.....	16
11.	Set running board and port.....	16
12.	Turn on hotspot.....	17
13.	Upload board .....	17
14.	Open monitor .....	17
15.	Check device status .....	17
16.	Check data visualization .....	17
17.	Check commands from cloud device to physical devices .....	18
17.1.	Trigger Buzzer .....	18
17.2.	Set Gas Alarm.....	18
17.3.	Set Temperature Alarm .....	18
17.4.	Commands effects in IoT Central .....	18
18.	View Raw data.....	19
19.	View Device Information .....	19
20.	Check email alert.....	19
21.	Check Blob Store .....	20

## 1. Introduction of project

Develop a smart and efficient system to detect the presence of fire or gas leak in indoor environments by using Esp32 and Azure IoT Central Cloud Service, and send email alert and sound alarm for relevant persons.

## 2. Function of project

- 1) Send DHT22 temperature and humidity data to IoT Central
- 2) Send MQ2 gas data (smoke, Methane or natural gas etc.) to IoT Central
- 3) View synchronized data from sensors on IoT Central platform
- 4) View raw data on IoT Central platform
- 5) View Client device information on IoT Central platform
- 6) Send command to control buzzer and led working or not working from IoT Central
- 7) Send command to change temperature alarm threshold from IoT Central
- 8) Send command to change gas alarm threshold from IoT Central
- 9) Send Fire or Gas Leak Email Alert and sound alarm to users
- 10) Store raw data into Blob Storage automatically

## 3. Novelty of project

- 1) Use MQTT and Wifi to establish communication between Esp32 and Azure IoT Central
- 2) First and only solution to use Azure IoT Central and Esp32 to detect fire and gas leak and send alert until now in the internet
  - No one uses IoT central as the cloud service
- 3) Use Azure IoT Central to send command to control buzzer and Led
- 4) Use Azure IoT Central to send command to change alarm thresholds for temperature and gas
- 5) Automatically upload Template for IoT device by using device code
- 6) Use Rules of IoT Central to send the Email Alert
- 7) Use any published templates resource by changing template model id.

## 4. Evaluation of project

### 4.1. Function evaluation

The function evaluation is mainly conducted by experiments. Here is the function evaluation table.

Function	Method and Metrics	Implementation Result	Analysis
1.Send DHT22 temperature and humidity data to IoT Central	Method: running system to test. Metrics: IoT Central receives data time costing, and if it is continuing.	Time costing: < 1s (second) when manually refreshing "Overview" and "Raw Data" page to view the data on IoT Central; < 2s when waiting for IoT Central refreshing. Continuing: continuing when hardware is keeping connection with computer and WIFI is stable.	Function works well. The time is mainly costing on refreshing not transmission. System is efficient and stable in receiving temperature and humidity data from DHT22 when hardware is keeping connection with computer and WIFI is stable.
2. Send MQ2 gas data to IoT Central.	Method: running system to test. Metrics: receives data time costing, and if it is continuing when WIFI is stable.	Time costing: < 1s (second) when manually refreshing "Overview" and "Raw Data" page to view the data on IoT Central; < 2s when waiting for IoT Central refreshing. Continuing: continuing when hardware is keeping connection with computer and WIFI is stable.	Function works well. The time is mainly costing on refreshing not transmission. System is efficient and stable in receiving gas data from MQ2 when hardware is keeping connection with computer and WIFI is stable.
3. View synchronized data chart on IoT Central platform.	Method: running system to test, putting hand close or far to DHT22, and making little smoke or blowing smoke away for MQ2 to view data changing on "Overview" page of IoT Central platform. Metrics: if data is changing from low to high, and then high to low.	Data is changing from low to high, and then high to low with test operations. And the line chart is reflecting this changing. Fig 1 shows this changing.	Function works well. The visualization effects are great.
4. View raw data on IoT Central platform.	Method: running system to test.	Update response time: < 1s when manually refreshing "Raw	Function works well.

	<p>Metrics: raw data update response time, and if it is continuing.</p>	<p>Data” page; &lt; 2s when waiting for IoT Central refreshing. Continuing: updating is continuing when hardware is keeping connection with computer and WIFI is stable. Fig 2 shows the raw data.</p>	<p>The time is mainly costing on refreshing not transmission. System is efficient and stable in updating raw data when hardware is keeping connection with computer and WIFI is stable.</p>
<p>5. View Client device information on IoT Central platform.</p>	<p>Method: running system to test, changing some device information in the code. Metrics: if the device information is as same as the definition in the code.</p>	<p>The device information is as same as the definition in the code file “Azure_IoT_PnP_Template.cpp”. Fig 3 shows they are same.</p>	<p>Function works well.</p>
<p>6. Send command to control buzzer and led working or not working from IoT Central.</p>	<p>Method: running system to test, clicking “Trigger Buzzer” button, making some smoke for MQ2. Metrics: If the LED can on and off when clicking button; If the buzzer can send alarm when the Led is on and the current data value is larger than alarm threshold value.</p>	<p>Led can on and off when clicking button without time delay. Buzzer can send alarm when the Led is on and the current smoke value is larger than alarm threshold value. Fig 4 shows the log out of Serial Monitor on Arduino IDE.</p>	<p>Function works well. <b>The buzzer sound continued for a little while after Led off. Because buzzer will finish its current running time and then stop. Buzzer sends sound using this function: <code>tone(Buzzer, Note_E4, 2000)</code>, here the buzzer will run for 2000 milliseconds, when this loop end, it won't get into the loop again because in “if (buzzer_status)” , the buzzer_status is false.</b></p>
<p>7. Send command to change temperature alarm threshold from IoT Central.</p>	<p>Method: running system to test, inputting “50” and clicking “Set Temperature Alarm” button. Metrics: If the “set_alarm_temperature” parameter is changed successfully.</p>	<p>Parameter is changed successfully without time delay, which can be shown by hearing buzzer sound of temperature alarm and be seen in the log out of Serial Monitor on Arduino IDE. Fig 4 shows the log out of Serial Monitor on Arduino IDE.</p>	<p>Function works well and is very easy to operate.</p>

8. Send command to change gas alarm threshold from IoT Central.	Method: running system to test, inputting "50" and clicking "Set Gas Alarm" button. Metrics: If the "set_alarm_gas" parameter is changed successfully.	Parameter is changed successfully without time delay, which can be shown by hearing buzzer sound of gas alarm and be seen in the log out of Serial Monitor on Arduino IDE. Fig 5 shows the log out of Serial Monitor on Arduino IDE.	Function works well and is very easy to operate.
9. Send Fire or Gas Leak Email Alert and sound alarm to users.	Method: running system to test, putting a heater (setting temperature as 135) near DHT22, making smoke near MQ2. Metrics: If the email account receives alert email successfully.	The email account received alert email successfully without time delay. Fig 5 shows gas alert email and fire alert email.	Function works well, and has no time delay.
10. Store raw data into Blob Storage automatically	Method: running system to test. Metrics: If the container of Blob Storage can store raw data automatically.	The container of Blob Storage can store raw data automatically. Fig 6 shows the raw data download from container.	Function works well, and has no time delay.

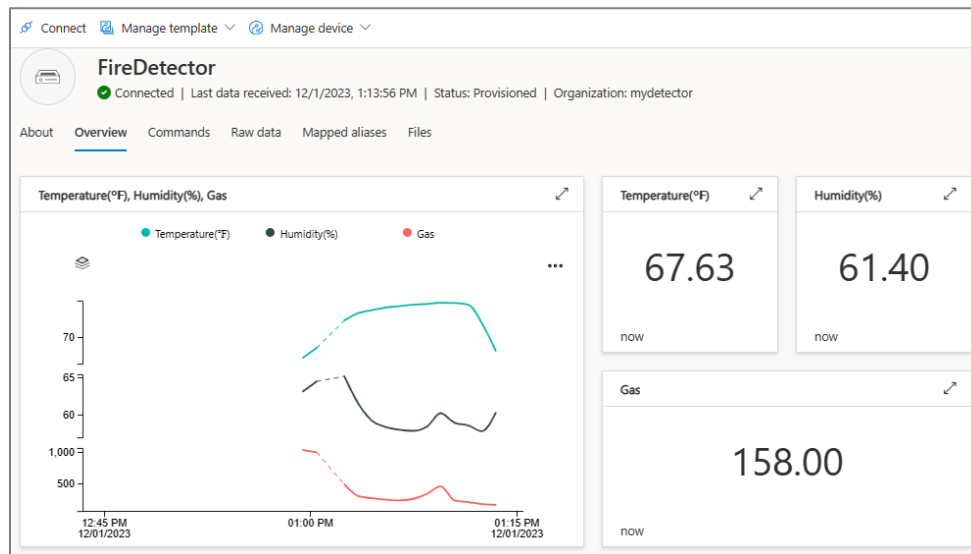


Fig 1, screen shot of Overview on Azure IoT Central platform

Devices > my\_temperature\_Gas\_Humidity > FireDetector

**FireDetector**  
 Connected | Last data received: 12/1/2023, 1:14:36 PM | Status: Provisioned | Organization: mydetector

About Overview Commands **Raw data** Mapped aliases Files

Timestamp ↓	Message type	Event creation time	Gas	Humidity(%)
> 12/1/2023, 1:14:36 PM	Command response	12/1/2023, 1:14:36 PM		
> 12/1/2023, 1:14:36 PM	Command request	12/1/2023, 1:14:35 PM		
> 12/1/2023, 1:14:36 PM	Telemetry		162	63.09
> 12/1/2023, 1:14:26 PM	Telemetry		165	62.4
> 12/1/2023, 1:14:16 PM	Telemetry		144	61.9
> 12/1/2023, 1:14:06 PM	Telemetry		164	61.4
> 12/1/2023, 1:13:56 PM	Telemetry		158	61.4
> 12/1/2023, 1:13:46 PM	Telemetry		162	61.29

Fig 2, screen shot of Raw data on Azure IoT Central platform

```
//device information
#define SAMPLE_MANUFACTURER_PROPERTY_VALUE "ESPRESSIF"
#define SAMPLE_MODEL_PROPERTY_VALUE "ESP32"
#define SAMPLE_VERSION_PROPERTY_VALUE "1.0.0"
#define SAMPLE_OS_NAME_PROPERTY_VALUE "FreeRTOS"
#define SAMPLE_ARCHITECTURE_PROPERTY_VALUE "ESP-WROOM-32"
#define SAMPLE_PROCESSOR_MANUFACTURER_PROPERTY_VALUE "ESPRESSIF"
#define SAMPLE_TOTAL_STORAGE_PROPERTY_VALUE 4096
```

Telemetry Frequency in Seconds, Device model, Softwar...

Telemetry Frequency in Seco... No Value

Device model ESP32  
read only device property

Software version 1.0.0  
read only device property

Operating system name FreeRTOS  
read only device property

Processor architecture ESP-WROOM-32  
read only device property

Total storage 4,096.00  
read only device property

Device manufacturer, Processor manufacturer

Device manufacturer ESPRESSIF  
read only device property

Processor manufacturer ESPRESSIF  
read only device property

Fig 3, screen shots of device information on Azure IoT Central platform and device information in the device code

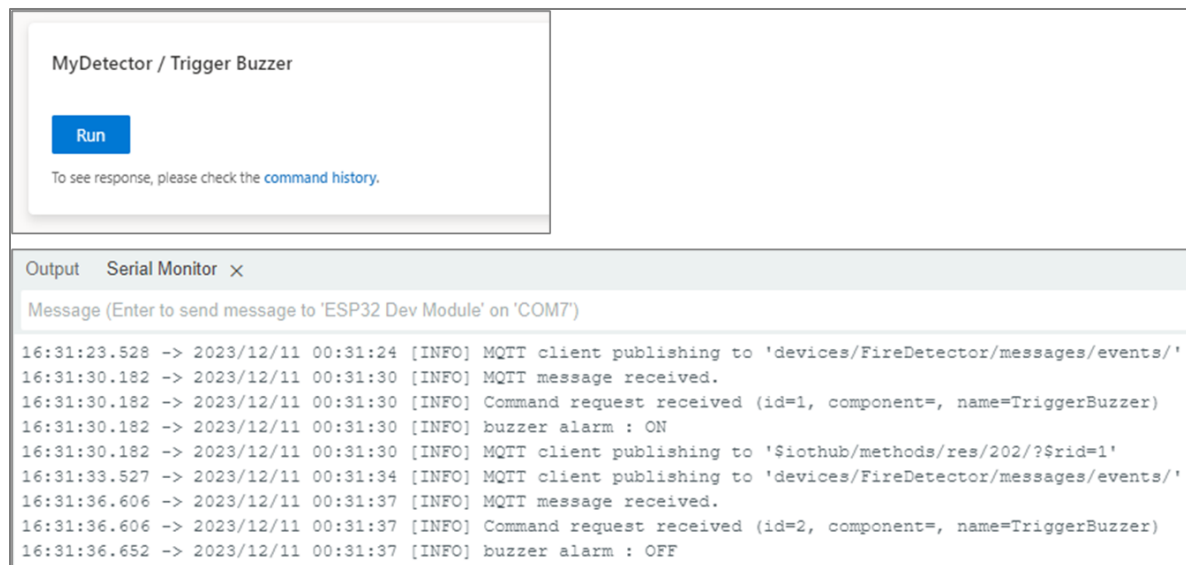


Fig 4, screen shots of Trigger Buzzer and command log out from Serial Monitor

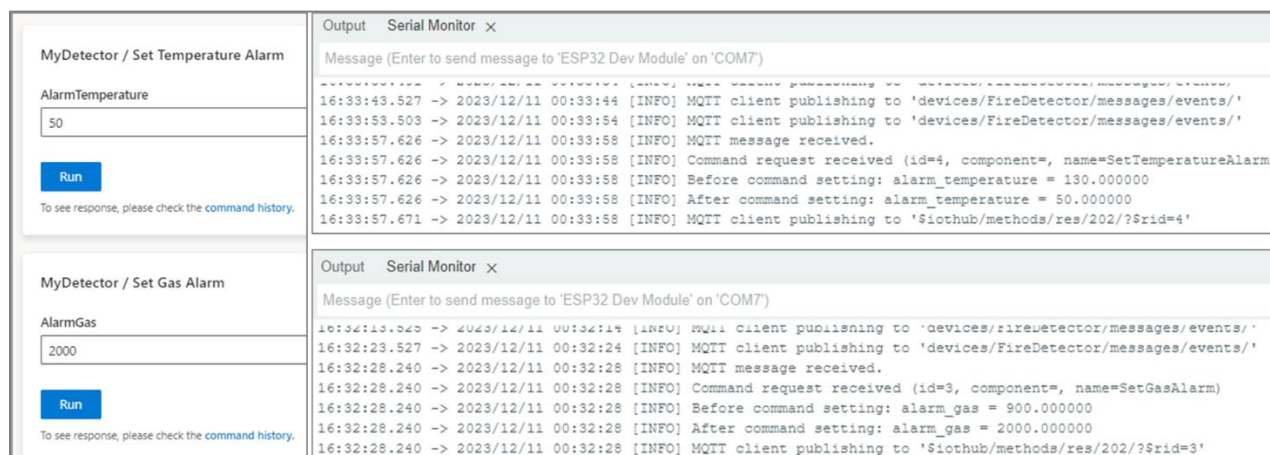


Fig 5, screen shots of Set Gas Alarm and Set Temperature Alarm and their command log out from Serial Monitor

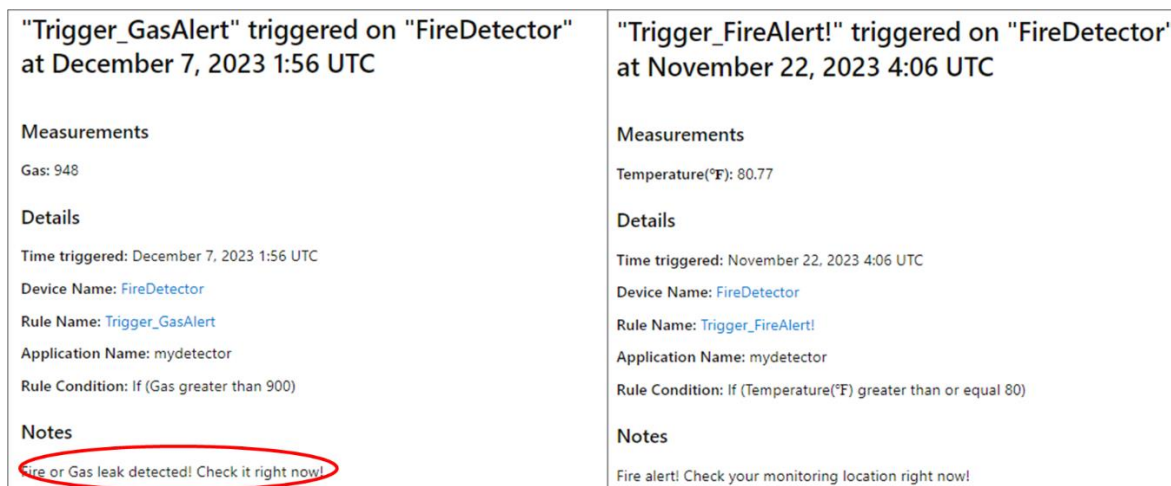


Fig 6, screen shots of fire and gas email alert.



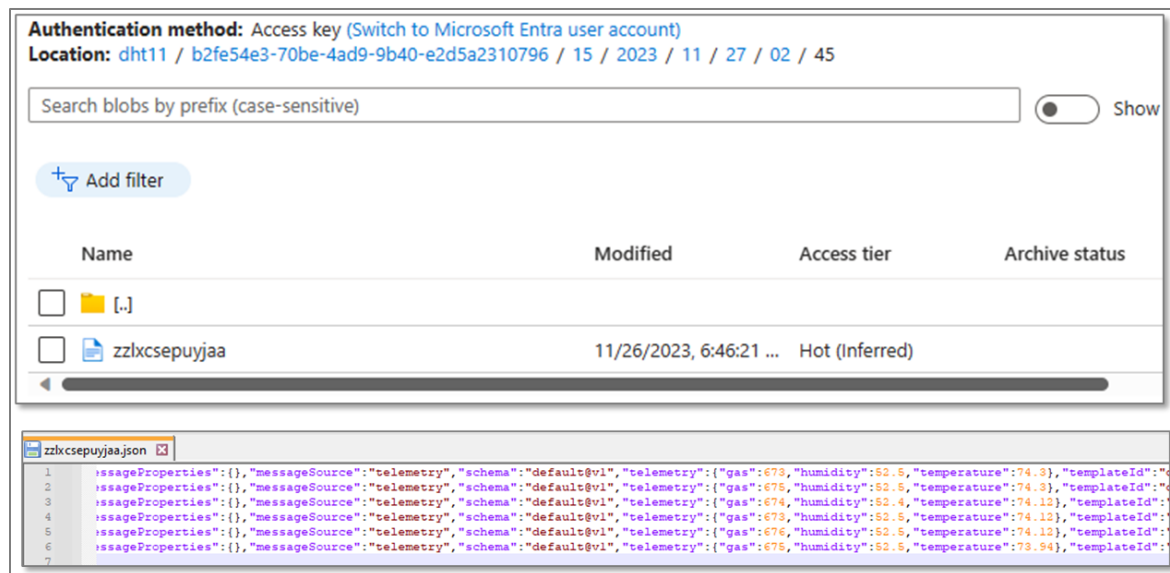


Fig 7, screen shots of raw data in Blob Storage.

## 4.2. Interface evaluation

The user interface of the project has been designed with a focus on user-friendliness. All features related to data display are prominently visible, ensuring easy access to crucial information. Additionally, operations are streamlined and effortlessly executed through the simple click of a button. This intuitive design enhances the overall user experience, making both data visualization and device control seamlessly accessible and user-friendly.

## 4.3. Security evaluation

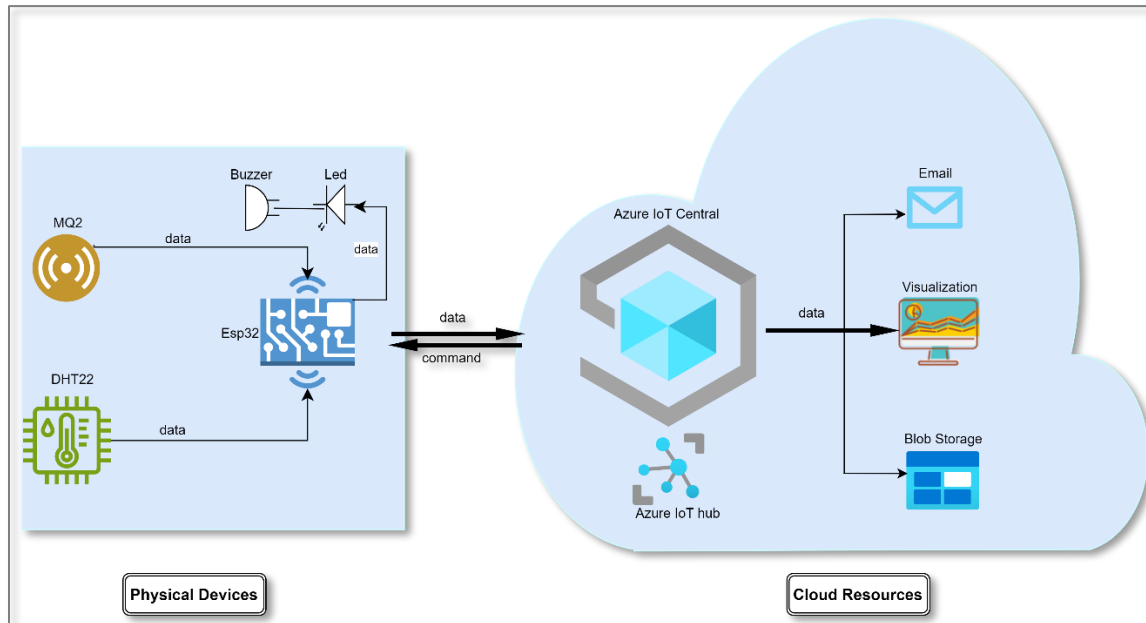
My project uses Azure relevant as the cloud service, which employs robust authentication mechanisms, Blob Storage supports encryption, adding an extra layer of protection to stored data. My project leverages Azure services as the chosen cloud platform, benefitting from robust authentication mechanisms inherent to Azure IoT Central. This ensures secure and authorized communication between the ESP32 device and the cloud, upholding the integrity and confidentiality of the data being transmitted.

Furthermore, by utilizing Azure Blob Storage for data storage, an additional layer of protection is applied through built-in encryption measures. These safeguards stored data, enhancing the overall security posture of the project.

Azure's commitment to industry-leading security standards, including ISO/IEC 27001 compliance, reinforces the project's dedication to maintaining a secure cloud environment. The comprehensive security practices integrated into Azure services, coupled with monitoring capabilities for incident response, contribute to a resilient end-to-end security framework.

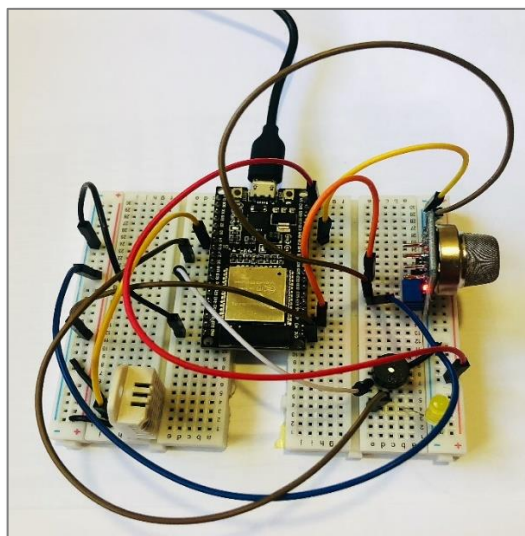
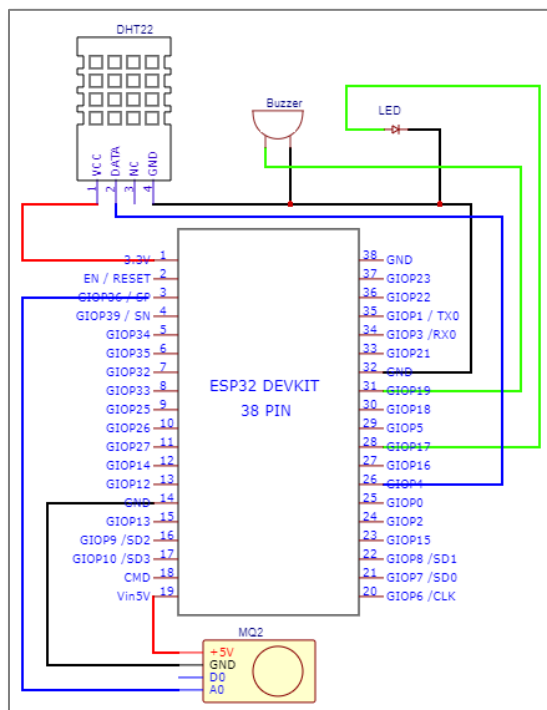
In summary, the project not only benefits from Azure's robust authentication mechanisms but also ensures data integrity and confidentiality through encryption in Blob Storage, aligning with the highest security standards in the industry.

## 5. Architecture



## 6. Circuit Diagram and Hardware

### 6.1. Circuit diagram and connected hardware



## 6.2. Hardware

- ESPRESSIF ESP32 board (Mine is esp32 with 30-pin) (1)
- Wi-Fi 2.4 GHz
  - Here, I create a Mobile hotspot using my laptop. The specific steps are as follows:
    - Open Settings: Press Windows key + I to open Settings.
    - Go to Network & Internet: Click on "Network & Internet."
    - Access Mobile Hotspot Settings: In the left sidebar, select "Mobile hotspot."
    - Turn On Mobile Hotspot: Toggle the switch under "Share my Internet connection with other devices" to turn on the mobile hotspot.
    - Configure Hotspot Settings (Optional): Click on "Edit" to configure your hotspot's network name (SSID) and password. Choose the brand "2.4 GHz".
- DHT22 (1)
- MQ2 (1)
- Buzzer (1)
- Led (1)
- USB 2.0 (1)
- Bread board (1)
- Wires (some)

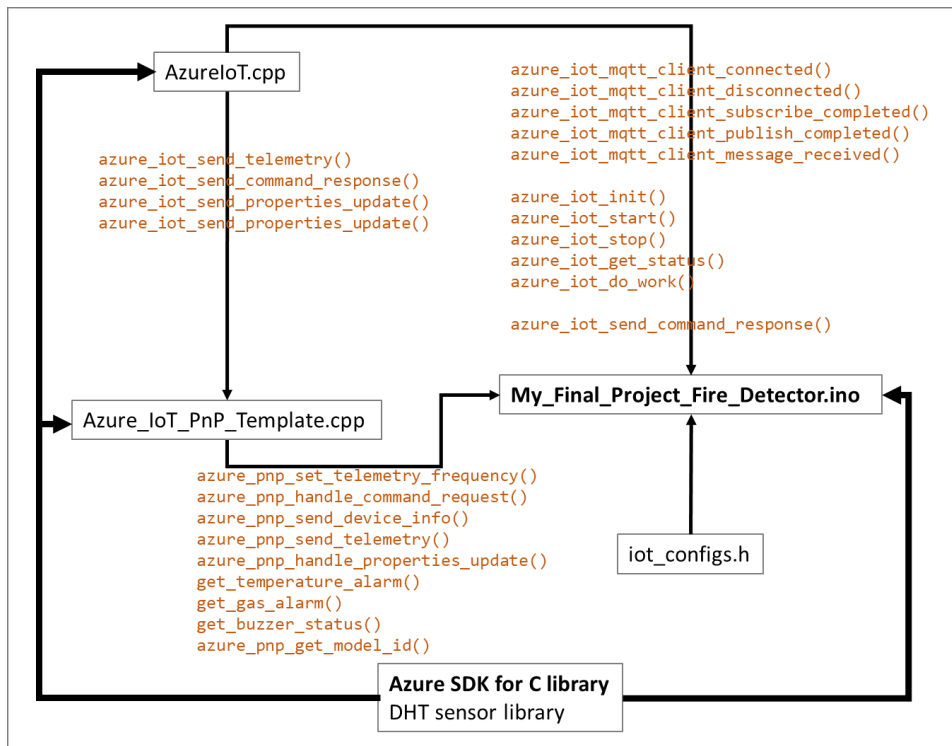
## 7. Project Code

### 7.1. Device Code

The device code files play a crucial role in implementing system functionalities. Executed on Arduino IDE, these files facilitate communication between the ESP32 and Azure IoT Central using the MQTT protocol and WiFi. Key functions include sending data from DHT22 and MQ2 sensors to Azure IoT Central, receiving and executing commands from the central platform, and sending command feedback back to Azure IoT Central.

The device code files primarily consist of the following: 1) `My_Final_Project_Fire_Detector.ino`: This Arduino file serves as the core of the project, containing the main implementation. 2) `AzureIoT.cpp`: This file provides essential functions for communication between the ESP32 and Azure IoT Central. 3) `Azure_IoT_PnP_Template.cpp`: Another critical file, it contributes functions that support the overall functionality of `My_Final_Project_Fire_Detector.ino`. 4) `iot_configs.h`: This file holds configuration parameters and settings essential for the proper execution of the device code.

Here is the visual representation of the relationship among these files.



## 7.2. Cloud Code

### 7.2.1. temperature\_Gas\_Humidity.json

The cloud code JSON file is crucial for establishing the custom template on Azure IoT Central. A segment of this file, illustrated in Fig 10, delineates the definition of the "Set Temperature Alarm" command capability. When a command is dispatched, the command name, such as "SetTemperatureAlarm," is transmitted to the device code file via the MQTT message. `azure_pnp_handle_command_request()` function within the "Azure\_IoT\_PnP\_Template.cpp" file is then invoked, directing the execution to the corresponding module to implement the requisite actions.

Display name	Name *	Capability type * ⓘ	Semantic type ⓘ
Temperature(°F)	temperature	Telemetry	Temperature
Humidity(%)	humidity	Telemetry	Humidity
Gas	gas	Telemetry	None
Set Temperature Alarm	SetTemperatureAlarm	Command	
Set Gas Alarm	SetGasAlarm	Command	
Trigger Buzzer	TriggerBuzzer	Command	
Telemetry Frequency in Seconds	telemetryFrequencySecs	Property	None

## 8. Configure Device Code running environment

### 8.1. Install the latest Arduino IDE version

### 8.2. Install the ESP32 board support on Arduino IDE

Navigate to Tools > Boards manager > Manage boards..., search ESP32 by Espressif and install it.

### 8.3. Install the Azure SDK for C library on Arduino IDE

Navigate to Sketch > Include Library > Manage Libraries..., search Azure SDK for C library and install it.

### 8.4. Install the DHT sensor library on Arduino IDE

Navigate to Sketch > Include Library > Manage Libraries..., search DHT sensor library and install it.

### 8.5. Install Adafruit Unified Sensor in the library manager

## 9. Create the Azure IoT Cloud Components

### 9.1. Create a new application in IoT Central Applications

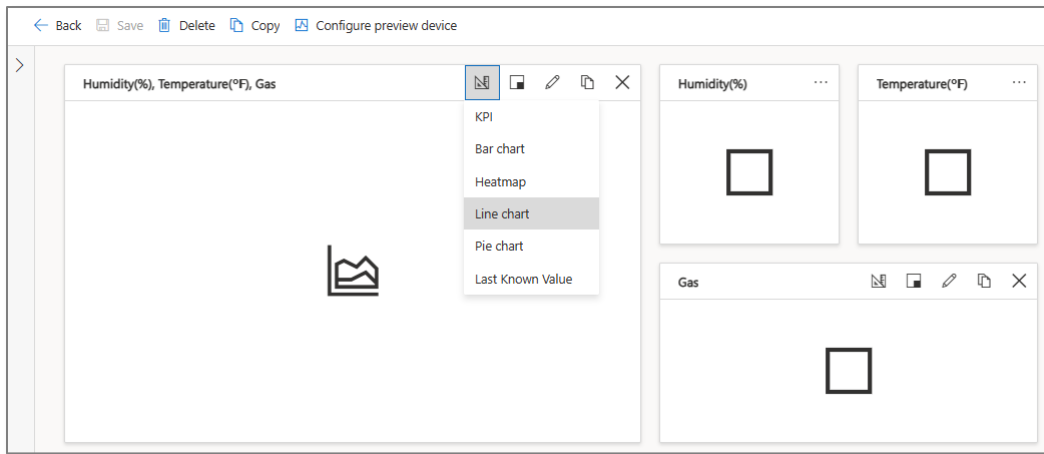
### 9.2. Create my IoT device on Azure IoT Central platform

### 9.3. Create custom device template

The screenshot shows the 'Add capabilities' dialog in the Azure IoT Central console. The device model is named 'test' and is in the 'Root' namespace. The dialog lists several capabilities to be added to the device model. Each capability has a display name, a name, a capability type, and a semantic type. The capabilities listed are:

Display name	Name *	Capability type *	Semantic type		
Humidity(%)	humidity	Telemetry	Humidity	X	✓
Set Temperature Alarm	SetTemperatureAlarm	Command		X	✓
Set Gas Alarm	SetGasAlarm	Command		X	✓
Temperature(°F)	temperature	Telemetry	Temperature	X	✓
Gas	gas	Telemetry	None	X	✓
Telemetry Frequency in Seconds	telemetryFrequencySecs	Property	None	X	✓
Toggle Buzzer	ToggleBuzzer	Command		X	✓

At the bottom of the dialog, there is a '+ Add capability' button.



## 9.4. Create email alert by using Rules

Save Cancel Rename

Rules > Trigger\_FireAlert

Trigger\_FireAlert

☒ Enabled

**Target devices**  
Select the device template your rule will use. If you need to narrow the rule's scope, add filters.

Device template \*  
my\_temperature\_Gas\_Humidity

+ Filter

**Conditions**  
Conditions define when your rule is triggered. Aggregation is optional—use it to cluster your data and trigger rules based on a time window.

Trigger the rule if

**Time aggregation**  
☒ Off

**Telemetry \***  **Operator \***

**Conditions**

Conditions define when your rule is triggered. Aggregation is optional—use it to cluster your data and trigger rules based on a time window.

Trigger the rule if all of the conditions are true

**Time aggregation**

☒ Off Select a time window

**Telemetry \*** Operator \*

Temperature(°F) Is greater than

☒ Enter a value ☐ Select a value

**Value \***

1000

## 9.5. Create Blob Storage

**Create a storage account** ...

**Basics** Advanced Networking Data protection Encryption Tags Review

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription \* Azure for Students

Resource group \* cloud-shell-storage-westus  
[Create new](#)

**Instance details**


**Storage accounts** ...

UW (cloud.washington.edu)

[+](#) Create [↶](#) Restore [⚙](#) Manage view [↺](#) Refresh [↓](#) Export to CSV [🔗](#) Open query | [🏷](#) Assign tags [🗑](#) Delete

Filter for any field... Subscription equals all Resource group equals all Location equals all [+ Add filter](#)

Showing 1 to 1 of 1 records.

<input type="checkbox"/> Name ↑↓	Type ↑↓	Kind ↑↓	Resource group ↑↓
<input type="checkbox"/>  blob4detector	Storage account	StorageV2	cloud-shell-storage-westus

**New container** ✕

**Name \***

dht11

## 10. Configure wifi and IoT relevant authorization information of device code

10.1. Navigate to folder “My\_Final\_Project\_Fire\_Detector”

10.2. Open “My\_Final\_Project\_Fire\_Detector.ino”

It will be opened in the Arduino IDE.

10.3. Configure “Azure\_IoT\_PnP\_Template.cpp” for template model id

Replace this:

```
#define AZURE_PNP_MODEL_ID "your custom template Model id"
```

10.4. Configure “iot\_configs.h”

Replace these:

```
#define IOT_CONFIG_WIFI_SSID "your Mobile hotspot SSID"
```

```
#define IOT_CONFIG_WIFI_PASSWORD "your Mobile hotspot password"
```

```
#define DPS_ID_SCOPE "your ID scope"
```

```
#define IOT_CONFIG_DEVICE_ID "your Device ID"
```

```
#define IOT_CONFIG_DEVICE_KEY "your Primary key"
```

10.5. Configure “Azure\_IoT\_PnP\_Template.cpp” for device information

Replace this if you want to define your own device information:

```
#define SAMPLE_MANUFACTURER_PROPERTY_VALUE "XXXX"
```

```
#define SAMPLE_MODEL_PROPERTY_VALUE " XXXX "
```

```
#define SAMPLE_VERSION_PROPERTY_VALUE " XXXX "
```

```
#define SAMPLE_OS_NAME_PROPERTY_VALUE " XXXX "
```

```
#define SAMPLE_ARCHITECTURE_PROPERTY_VALUE " XXXX "
```

```
#define SAMPLE_PROCESSOR_MANUFACTURER_PROPERTY_VALUE "XXXX "
```

## 11. Set running board and port

Navigate to Tools > Board > esp32, select **ESP32 Dev Module**;

Navigate to Tools > port, select port (Mine is COM6).



## 12. Turn on hotspot

Turn on “Mobile hotspot” in the network connection.

## 13. Upload board

Click “upload” for “My\_Final\_Project\_Fire\_Detector.ino” in Arduino IDE.

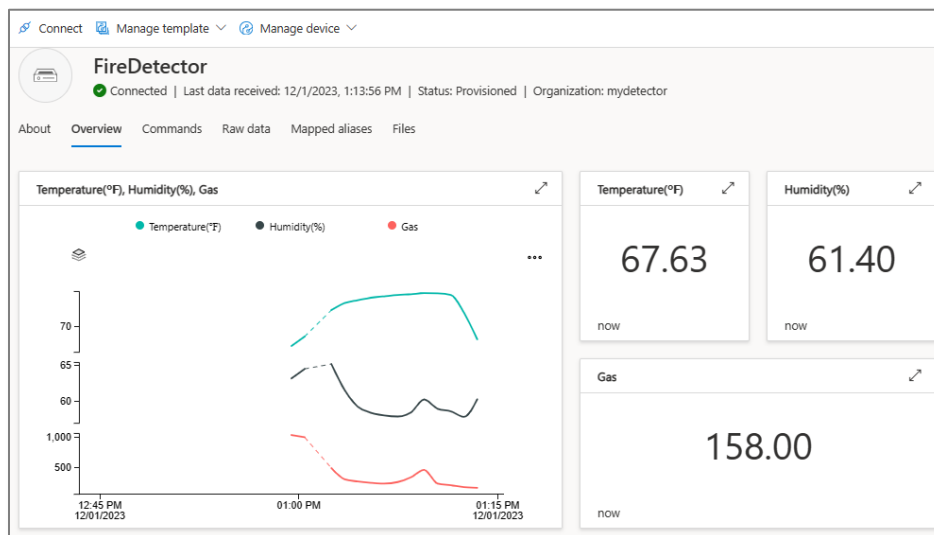
## 14. Open monitor

Click “Serial Monitor” to MCU (microcontroller) locally via the Serial Port after upload finishing.

## 15. Check device status

- From the application dashboard, select Devices on the side navigation menu.
- Check the Device status of the device is updated to Provisioned.
- Check the **Device template** of the device

## 16. Check data visualization



## 17. Check commands from cloud device to physical devices

You can use IoT Central to send a command to your device.

In my project, I realize 3 commands:

### 17.1. Trigger Buzzer

Turn on or off buzzer and led. If led light is on, the buzzer alarms once the detected data larger than limit values. If led light is off, the buzzer won't alarm no matter the detected data is larger or less than limit value.

### 17.2. Set Gas Alarm

Change the gas limit value, which makes my project to be more possible applications in different situations.

### 17.3. Set Temperature Alarm

Change the temperature limit value, which makes my project to be more possible applications in different situations.

### 17.4. Commands effects in IoT Central

The screenshot displays the IoT Central interface for a device named "FireDetector". The interface includes a top navigation bar with "Connect", "Manage template", and "Manage device" options. Below the navigation bar, the device name "FireDetector" is shown along with its status "Connected" and last data received timestamp "11/26/2023, 6:43:25 PM". The "Commands" tab is selected, showing three command cards:

- MyDetector / Set Temperature Alarm**: The "AlarmTemperature" field is set to "100". A "Run" button is present.
- MyDetector / Set Gas Alarm**: The "AlarmGas" field is set to "1000". A "Run" button is present.
- MyDetector / Trigger Buzzer**: A "Run" button is present.

Each card includes a note: "To see response, please check the [command history](#)."

## 18. View Raw data

You can use IoT Central to view Raw data from your physical devices.

## 19. View Device Information

The screenshot shows the IoT Central interface for a device named 'FireDetector'. The device is connected, with the last data received on 11/27/2023 at 10:33:57 PM. The status is 'Provisioned' and the organization is 'mydetector'. The 'About' tab is selected, showing a list of device properties. The 'Raw data' tab is also visible.

Telemetry Frequency in Seconds, Device model, Software version	
Telemetry Frequency in Seconds	No Value
Device model	ESP32 read only device property
Software version	1.0.0 read only device property
Operating system name	FreeRTOS read only device property
Processor architecture	ESP-WROOM-32 read only device property
Total storage	4,096.00 read only device property

Device manufacturer, Processor manufacturer	
Device manufacturer	ESPRESSIF read only device property
Processor manufacturer	ESPRESSIF read only device property

## 20. Check email alert

The screenshot shows an email alert from Microsoft Azure. The subject is '"Trigger\_GasAlert" triggered on "FireDetector" at December 1, 2023 20:59 UTC'. The email contains details about the triggered alert, including the device name, rule name, and application name.

Microsoft Azure

**"Trigger\_GasAlert" triggered on "FireDetector" at December 1, 2023 20:59 UTC**

**Measurements**

Gas: 1051

**Details**

Time triggered: December 1, 2023 20:59 UTC

Device Name: [FireDetector](#)

Rule Name: [Trigger\\_GasAlert](#)

Application Name: mydetector

Rule Condition: If (Gas greater than 1000)

**Notes**

Fire or Gas leak detected! Check it right now!

## 21. Check Blob Store

```
zzkcsepuvjaa.json
1  { "messageProperties": {}, "messageSource": "telemetry", "schema": "default@v1", "telemetry": { "gas": 673, "humidity": 52.5, "temperature": 74.3 }, "templateId": "d
2  { "messageProperties": {}, "messageSource": "telemetry", "schema": "default@v1", "telemetry": { "gas": 675, "humidity": 52.5, "temperature": 74.3 }, "templateId": "d
3  { "messageProperties": {}, "messageSource": "telemetry", "schema": "default@v1", "telemetry": { "gas": 674, "humidity": 52.4, "temperature": 74.12 }, "templateId": "d
4  { "messageProperties": {}, "messageSource": "telemetry", "schema": "default@v1", "telemetry": { "gas": 673, "humidity": 52.5, "temperature": 74.12 }, "templateId": "d
5  { "messageProperties": {}, "messageSource": "telemetry", "schema": "default@v1", "telemetry": { "gas": 676, "humidity": 52.5, "temperature": 74.12 }, "templateId": "d
6  { "messageProperties": {}, "messageSource": "telemetry", "schema": "default@v1", "telemetry": { "gas": 675, "humidity": 52.5, "temperature": 73.94 }, "templateId": "d
7
```