Project Introduction and Implementation

Contents

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 Could 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 of 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.

	Metrics: raw data update response time, and if it is continuing.	Data" page; < 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.	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.
5. View Client device information on IoT Central platform.	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.	The device information is as same as the definition in the code file "Azure_IoT_PnP_Template.cpp". Fig 3 shows they are same.	Function works well.
6. Send command to control buzzer and led working or not working from IoT Central.	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.	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.	Function works well. **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: tone(Buzzer, Note_E4, 2000), 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.**
7. Send command to change temperature alarm threshold from IoT Central.	Method: running system to test, inputting "50" and clicking "Set Temperature Alarm" button. Metrics: If the "set_alarm_temperature" parameter is changed successfully.	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.	Function works well and is very easy to operate.

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.

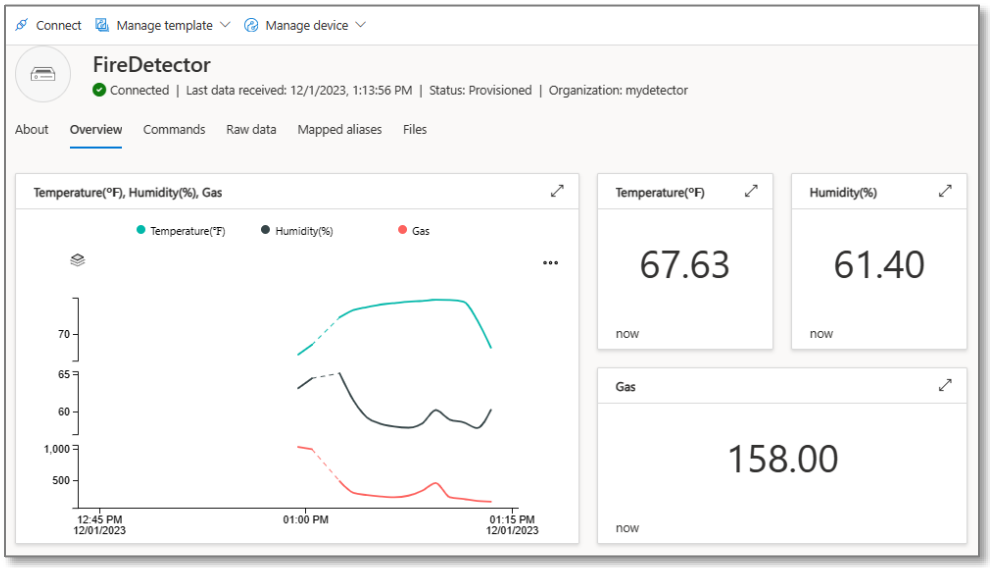
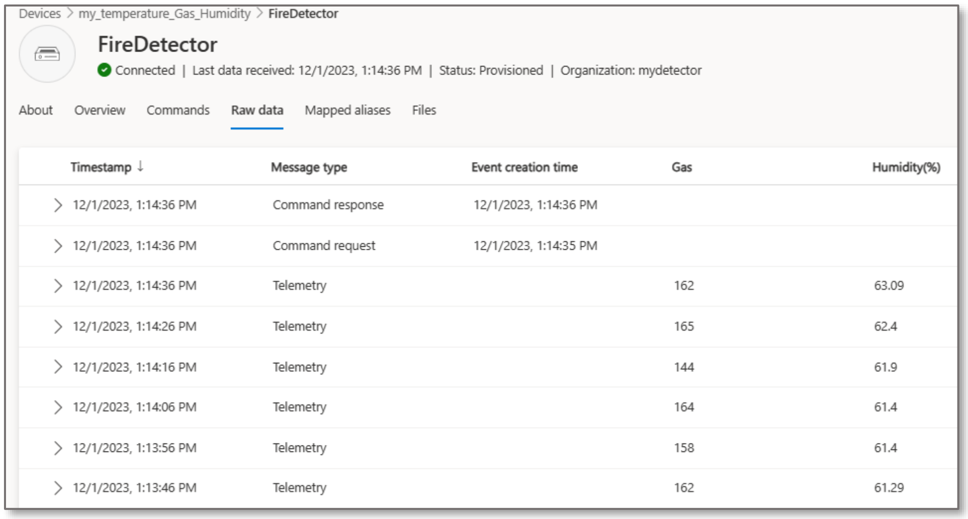


Fig 1, screen shot of Overview on Azure IoT Central platform



Devices > my_temperature_Gas_Humidity > FireDetector

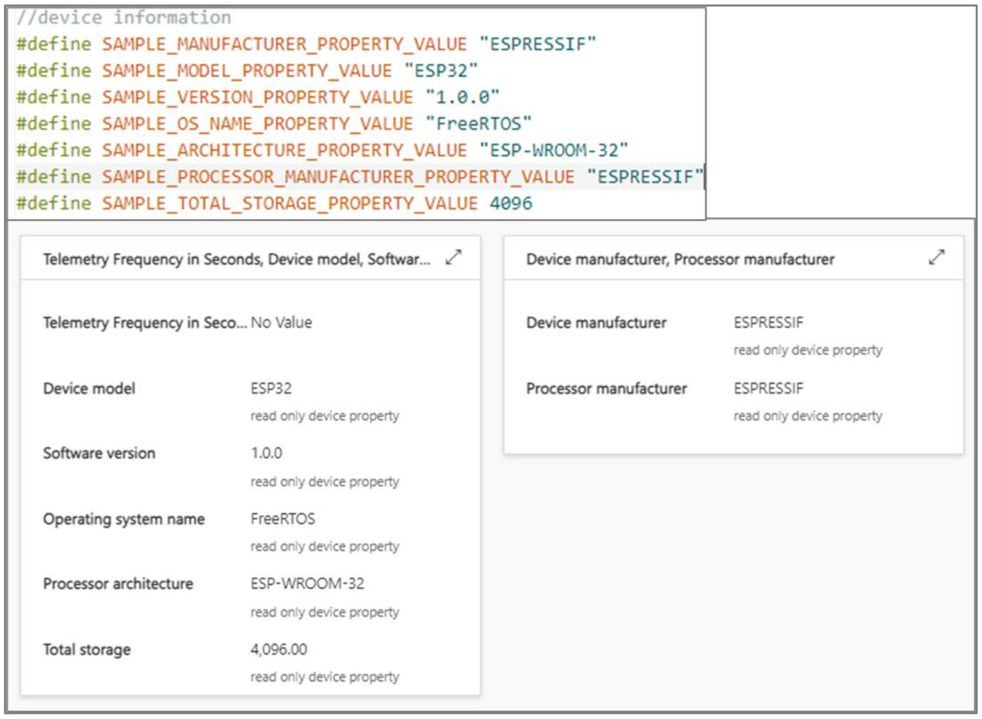
FireDetector

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
```

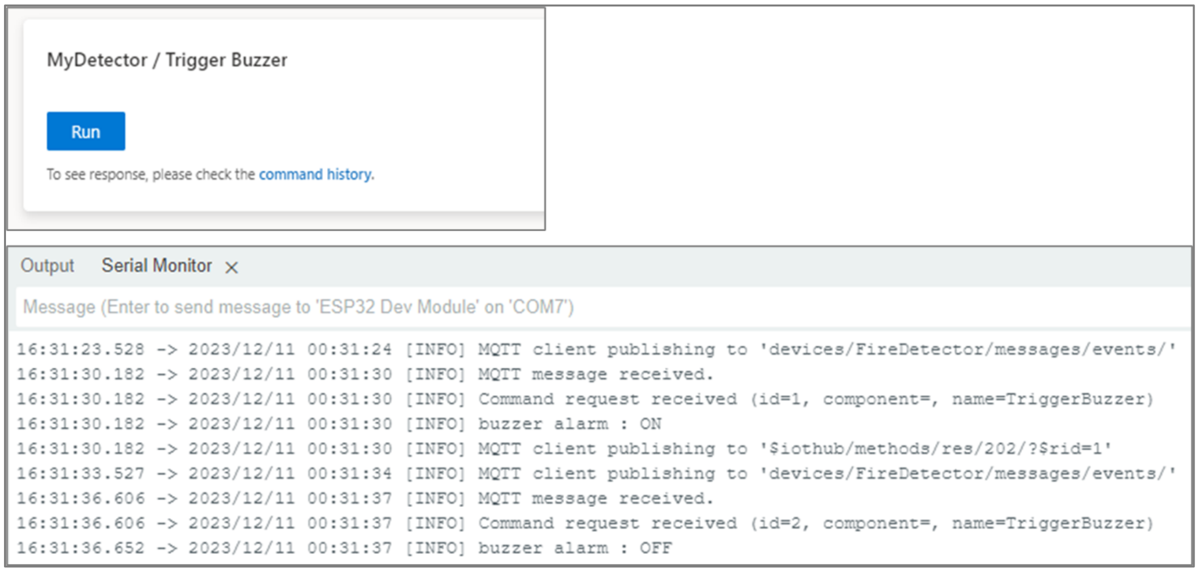
Telemetry Frequency in Seconds, Device model, Softwar...

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



MyDetector / Trigger Buzzer

Run

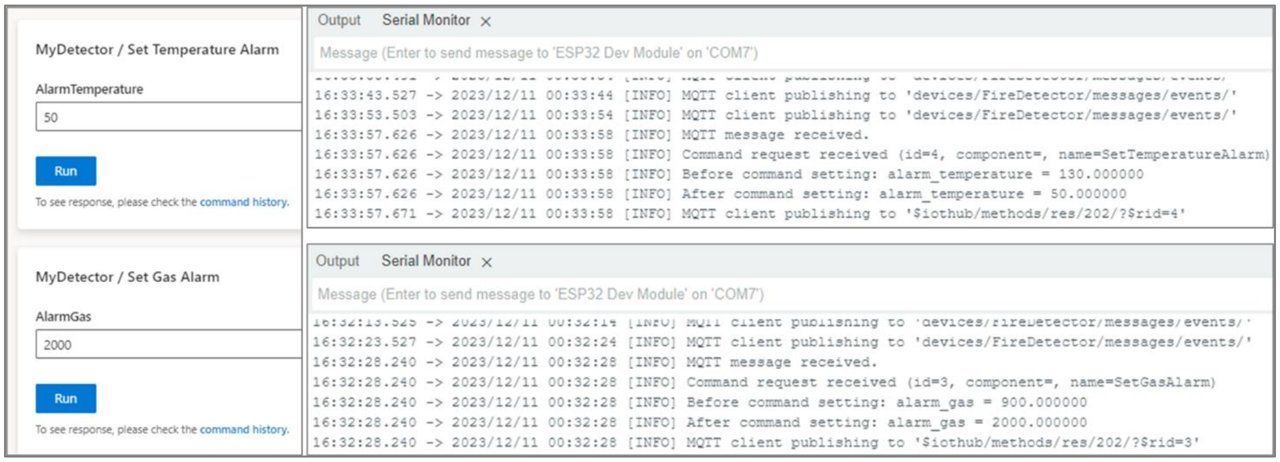
To see response, please check the command history.

Output Serial Monitor

Message (Enter to send message to 'ESP32 Dev Module' on 'COM7')

```
16:31:23.528 -> 2023/12/11 00:31:24 [INFO] MQTT client publishing to 'devices/FireDetector/messages/events/'
16:31:30.182 -> 2023/12/11 00:31:30 [INFO] MQTT message received.
16:31:30.182 -> 2023/12/11 00:31:30 [INFO] Command request received (id=1, component=, name=TriggerBuzzer)
16:31:30.182 -> 2023/12/11 00:31:30 [INFO] buzzer alarm : ON
16:31:30.182 -> 2023/12/11 00:31:30 [INFO] MQTT client publishing to '$iothub/methods/res/202/?$rid=1'
16:31:33.527 -> 2023/12/11 00:31:34 [INFO] MQTT client publishing to 'devices/FireDetector/messages/events/'
16:31:36.606 -> 2023/12/11 00:31:37 [INFO] MQTT message received.
16:31:36.606 -> 2023/12/11 00:31:37 [INFO] Command request received (id=2, component=, name=TriggerBuzzer)
16:31:36.652 -> 2023/12/11 00:31:37 [INFO] buzzer alarm : OFF
```

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



MyDetector / Set Temperature Alarm

AlarmTemperature

50

Run

To see response, please check the command history.

Output Serial Monitor

Message (Enter to send message to 'ESP32 Dev Module' on 'COM7')

```
16:33:43.527 -> 2023/12/11 00:33:44 [INFO] MQTT client publishing to 'devices/FireDetector/messages/events/'
16:33:53.503 -> 2023/12/11 00:33:54 [INFO] MQTT client publishing to 'devices/FireDetector/messages/events/'
16:33:57.626 -> 2023/12/11 00:33:58 [INFO] MQTT message received.
16:33:57.626 -> 2023/12/11 00:33:58 [INFO] Command request received (id=4, component=, name=SetTemperatureAlarm)
16:33:57.626 -> 2023/12/11 00:33:58 [INFO] Before command setting: alarm_temperature = 130.000000
16:33:57.626 -> 2023/12/11 00:33:58 [INFO] After command setting: alarm_temperature = 50.000000
16:33:57.671 -> 2023/12/11 00:33:58 [INFO] MQTT client publishing to '$iothub/methods/res/202/?$rid=4'
```

MyDetector / Set Gas Alarm

AlarmGas

2000

Run

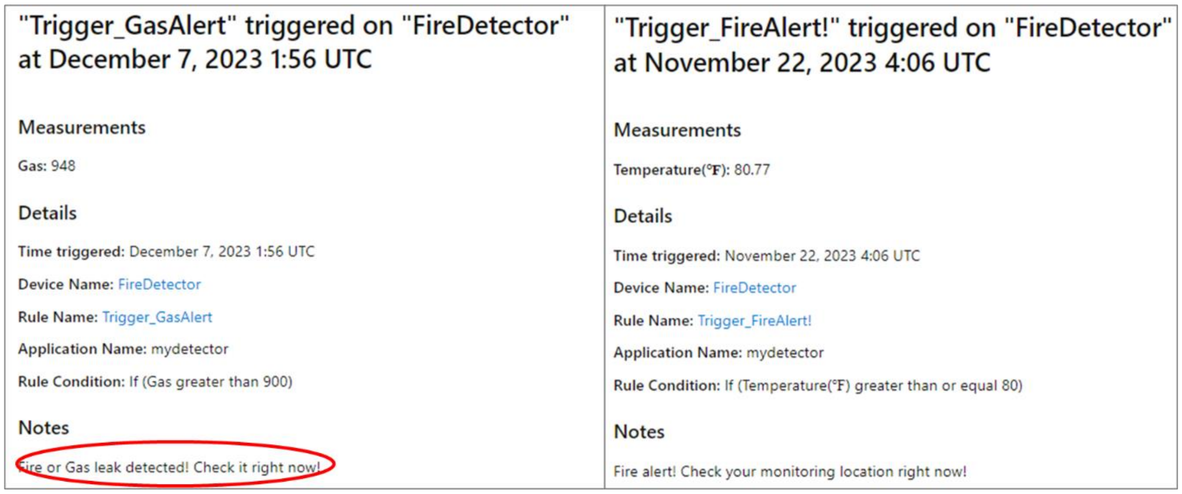
To see response, please check the command history.

Output Serial Monitor

Message (Enter to send message to 'ESP32 Dev Module' on 'COM7')

```
16:32:13.525 -> 2023/12/11 00:32:14 [INFO] MQTT client publishing to 'devices/FireDetector/messages/events/'
16:32:23.527 -> 2023/12/11 00:32:24 [INFO] MQTT client publishing to 'devices/FireDetector/messages/events/'
16:32:28.240 -> 2023/12/11 00:32:28 [INFO] MQTT message received.
16:32:28.240 -> 2023/12/11 00:32:28 [INFO] Command request received (id=3, component=, name=SetGasAlarm)
16:32:28.240 -> 2023/12/11 00:32:28 [INFO] Before command setting: alarm_gas = 900.000000
16:32:28.240 -> 2023/12/11 00:32:28 [INFO] After command setting: alarm_gas = 2000.000000
16:32:28.240 -> 2023/12/11 00:32:28 [INFO] MQTT client publishing to '$iothub/methods/res/202/?$rid=3'
```

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



"Trigger_GasAlert" triggered on "FireDetector" at December 7, 2023 1:56 UTC

Measurements

Gas: 948

Details

Time triggered: December 7, 2023 1:56 UTC

Device Name: FireDetector

Rule Name: Trigger_GasAlert

Application Name: mydetector

Rule Condition: If (Gas greater than 900)

Notes

Fire or Gas leak detected! Check it right now!

"Trigger_FireAlert!" triggered on "FireDetector" at November 22, 2023 4:06 UTC

Measurements

Temperature(℉): 80.77

Details

Time triggered: November 22, 2023 4:06 UTC

Device Name: FireDetector

Rule Name: Trigger_FireAlert!

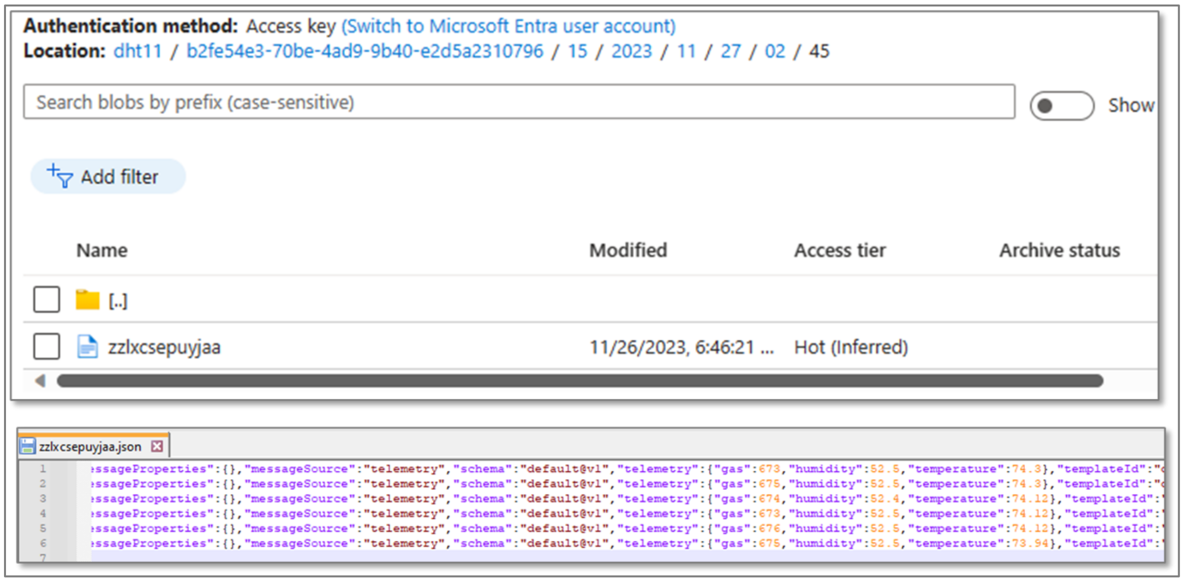
Application Name: mydetector

Rule Condition: If (Temperature(℉) greater than or equal 80)

Notes

Fire alert! Check your monitoring location right now!

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


Authentication method: Access key (Switch to Microsoft Entra user account)
Location: dht11 / b2fe54e3-70be-4ad9-9b40-e2d5a2310796 / 15 / 2023 / 11 / 27 / 02 / 45

Name	Modified	Access tier	Archive status
[..]			
zzlxcsepuyjaa	11/26/2023, 6:46:21 ...	Hot (Inferred)	

zzlxcsepuyjaa.json

```
ssageProperties":{},"messageSource":"telemetry","schema":"default@v1","telemetry":{"gas":673,"humidity":52.5,"temperature":74.3},"templateId":"
ssageProperties":{},"messageSource":"telemetry","schema":"default@v1","telemetry":{"gas":675,"humidity":52.5,"temperature":74.3},"templateId":"
ssageProperties":{},"messageSource":"telemetry","schema":"default@v1","telemetry":{"gas":674,"humidity":52.4,"temperature":74.12},"templateId":
ssageProperties":{},"messageSource":"telemetry","schema":"default@v1","telemetry":{"gas":673,"humidity":52.5,"temperature":74.12},"templateId":
ssageProperties":{},"messageSource":"telemetry","schema":"default@v1","telemetry":{"gas":676,"humidity":52.5,"temperature":74.12},"templateId":
ssageProperties":{},"messageSource":"telemetry","schema":"default@v1","telemetry":{"gas":675,"humidity":52.5,"temperature":73.94},"templateId":
```

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

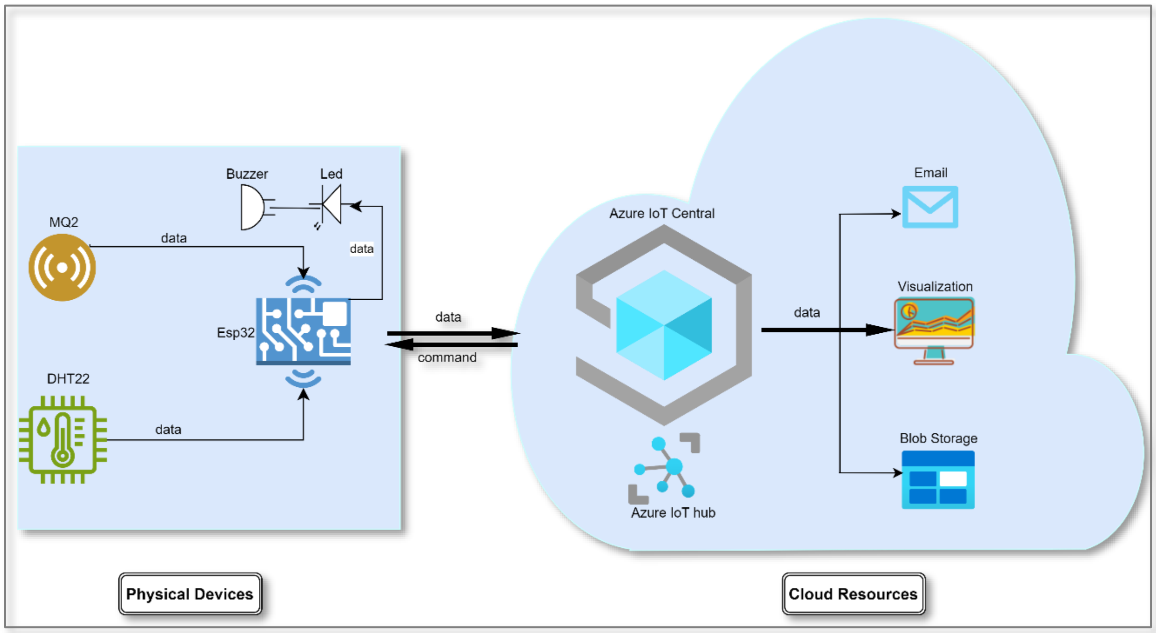
My project uses Azure relevant as the cloud service, which employs robust authentication mechanisms, Blob Storage supports encryption, adding an extra layer of protection to stored data. My project leverages Azure services as the chosen cloud platform, benefitting from robust authentication mechanisms 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.

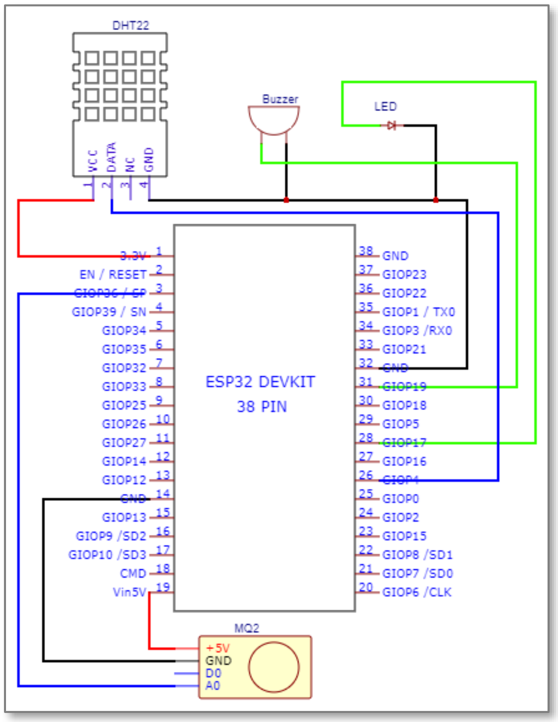
In summary, the project not only benefits from Azure's robust authentication mechanisms but also ensures data integrity and confidentiality through encryption in Blob Storage, aligning with the highest security standards in the industry.

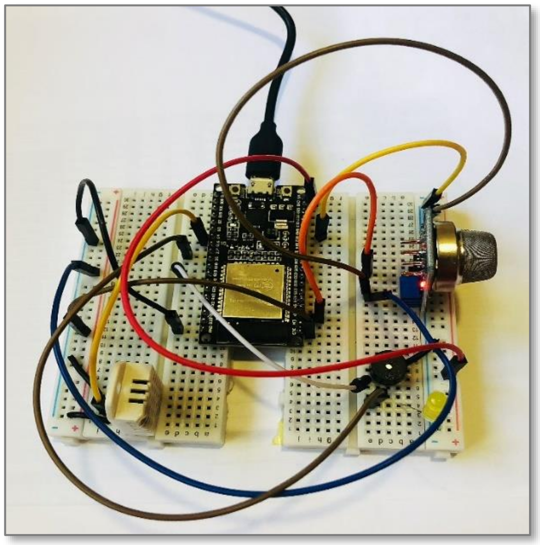
5. Architecture



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)

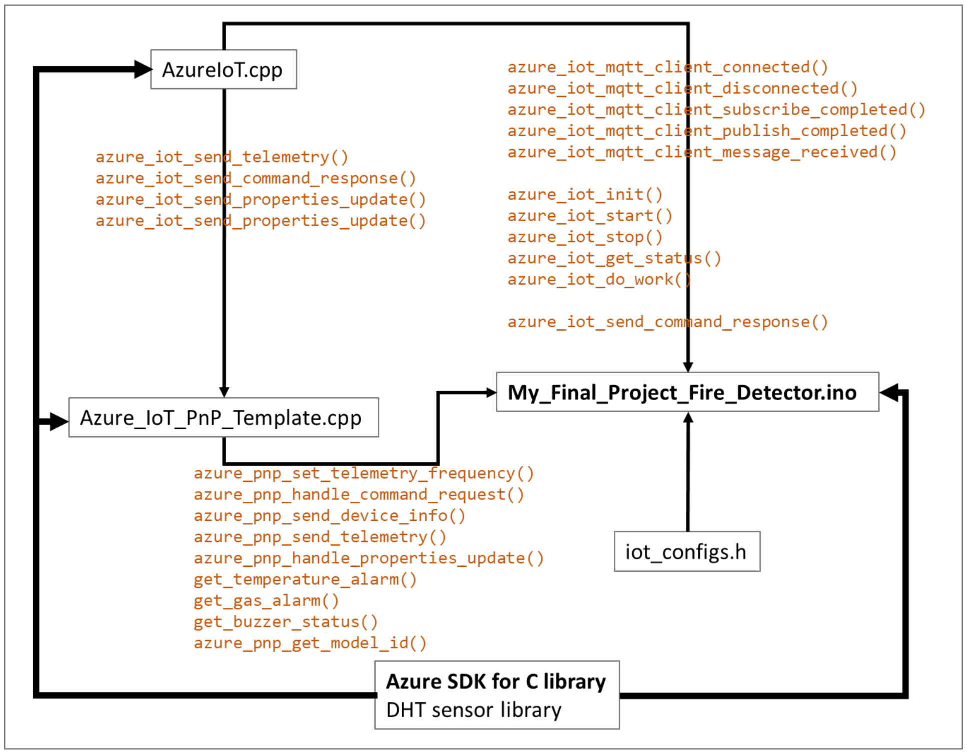
7. Project Code

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.


AzureIoT.cpp
azure_iot_send_telemetry()
azure_iot_send_command_response()
azure_iot_send_properties_update()
azure_iot_send_properties_update()
azure_iot_mqtt_client_connected()
azure_iot_mqtt_client_disconnected()
azure_iot_mqtt_client_subscribe_completed()
azure_iot_mqtt_client_publish_completed()
azure_iot_mqtt_client_message_received()
azure_iot_init()
azure_iot_start()
azure_iot_stop()
azure_iot_get_status()
azure_iot_do_work()
azure_iot_send_command_response()
Azure_IoT_PnP_Template.cpp
My_Final_Project_Fire_Detector.ino
azure_pnp_set_telemetry_frequency()
azure_pnp_handle_command_request()
azure_pnp_send_device_info()
azure_pnp_send_telemetry()
azure_pnp_handle_properties_update()
get_temperature_alarm()
get_gas_alarm()
get_buzzer_status()
azure_pnp_get_model_id()
iot_configs.h
Azure SDK for C library
DHT sensor library

7.1.1. Azure_IoT_Central_ESP32.ino

It is project core file running on Arduino IDE and has 2 main modules.

7.1.1.1. Setup module

It initializes the serial communication for logging, connects to Wi-Fi, synchronizes the device's clock with an NTP (Network Time Protocol) server, and sets up Azure IoT using configure_azure_iot() and azure_iot_start), DHT22 sensor, MQ2, Buzzer, and Led.

7.1.1.2. Loop module

The key functions:

– constantly checking the device's connection status, and handling reconnection if the device loses the Wi-Fi connection;

– Sending telemetry data to Azure IoT Central by mainly using azure_pnp_send_telemetry ();

– Causing the Azure IoT client to perform its tasks for connecting and working with Azure IoT by using azure_iot_do_work();

– Implementing commands and logical decision statements. Fig 8 shows commands implementation details in setup module.

Here is the main code about commands implementation in loop:

```
  alarm_temperature = get_temperature_alarm();
  alarm_gas = get_gas_alarm();
  buzzer_status = get_buzzer_status();

  if (buzzer_status) {
    digitalWrite(Led, HIGH); // turn the LED on, means turn on buzzer
monitoring.
  }
  if (!buzzer_status) {
    digitalWrite(Led, LOW);  // turn the LED off, means turn off buzzer
monitoring.
  }
  if (buzzer_status) {
    if (t > alarm_temperature) { // t is DHT22 sensor's realtime value
      Serial.println("Alert! Fire is detected!");
      tone(Buzzer, Note_B4, 2000);
      int pauseDuration = 2000;
      delay(pauseDuration);
      noTone(Buzzer);
    }else if(gas > alarm_gas) { // gas is MQ2 sensor's realtime value
      Serial.println("Alert! Gas leak or fire is detected!");
      tone(Buzzer, Note_E4, 2000);
      int pauseDuration = 1000 * 3;
      delay(pauseDuration);
      noTone(Buzzer);
    }
  }
```

7.1.2. AzureIoT.cpp

It acts as the interface for establishing and managing connections with Azure IoT Hub of Azure IoT Central platform. Its primary responsibilities include handling an MQTT client, which involves tasks such as connecting to IoT Hub, subscribing to specific topics, and publishing messages. In addition to configuring access to Azure IoT services, this file mandates the implementation of functions to:

- Manage the MQTT client to establish seamless communication with Azure IoT Hub;
- Perform Data Manipulations, including HMAC SHA256 encryption, Base64 decoding, and encoding;
- Receive Callbacks for Plug and Play by managing changes in device properties and responding to commands initiated by the IoT Hub.

By fulfilling these requirements, the AzureIoT.cpp file becomes instrumental in facilitating smooth integration with Azure IoT Hub, guaranteeing essential communication, ensuring data security, and handling IoT Hub features effectively.

7.1.2.1. Some Main Functions

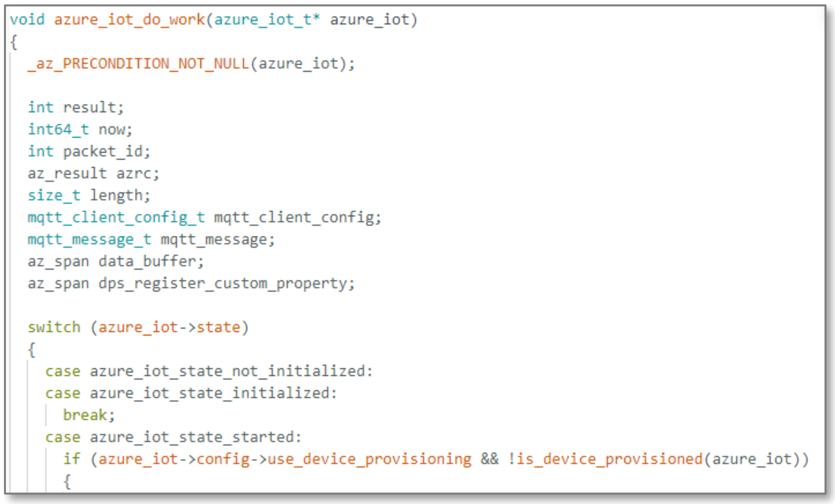
- generate_sas_token_for_dps()
- generate_sas_token_for_iot_hub()
- get_mqtt_client_config_for_dps()
- get_mqtt_client_config_for_iot_hub()

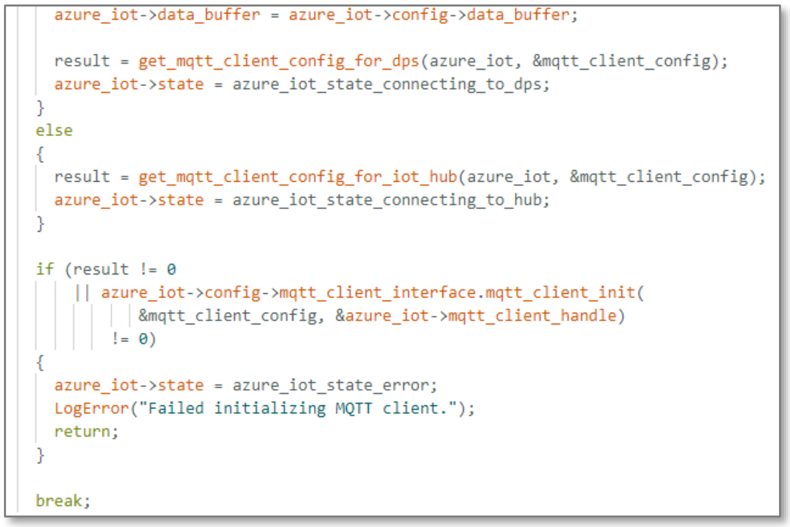
– generate_dps_register_custom_property()

Public API:

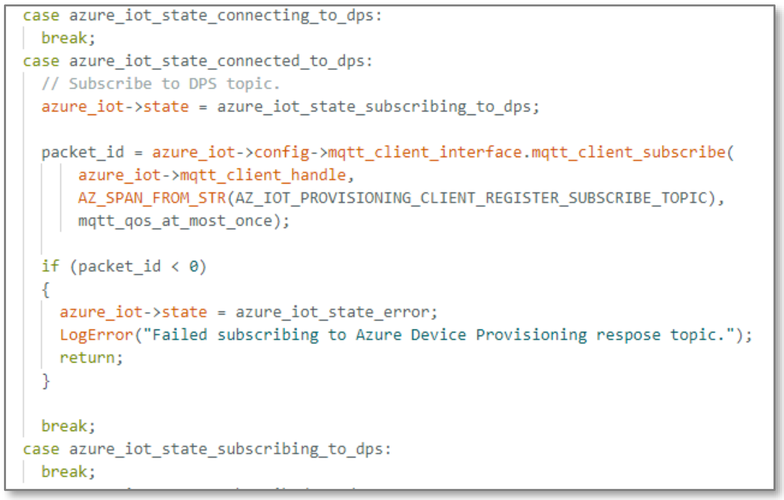
– azure_iot_init()
– azure_iot_start()
– azure_iot_stop()
– azure_iot_get_status()
– azure_iot_do_work()
– azure_iot_send_telemetry()
– azure_iot_send_properties_update()
– azure_iot_mqtt_client_connected()
– azure_iot_mqtt_client_disconnected()
– azure_iot_mqtt_client_subscribe_completed()
– azure_iot_mqtt_client_publish_completed()
– azure_iot_mqtt_client_message_received()
– azure_iot_send_command_response()

7.1.2.2. Parts of azure_iot_do_work()


```
void azure_iot_do_work(azure_iot_t* azure_iot)
{
  _az_PRECONDITION_NOT_NULL(azure_iot);

  int result;
  int64_t now;
  int packet_id;
  az_result azrc;
  size_t length;
  mqtt_client_config_t mqtt_client_config;
  mqtt_message_t mqtt_message;
  az_span data_buffer;
  az_span dps_register_custom_property;

  switch (azure_iot->state)
  {
    case azure_iot_state_not_initialized:
    case azure_iot_state_initialized:
      break;
    case azure_iot_state_started:
      if (azure_iot->config->use_device_provisioning && !is_device_provisioned(azure_iot))
      {
```

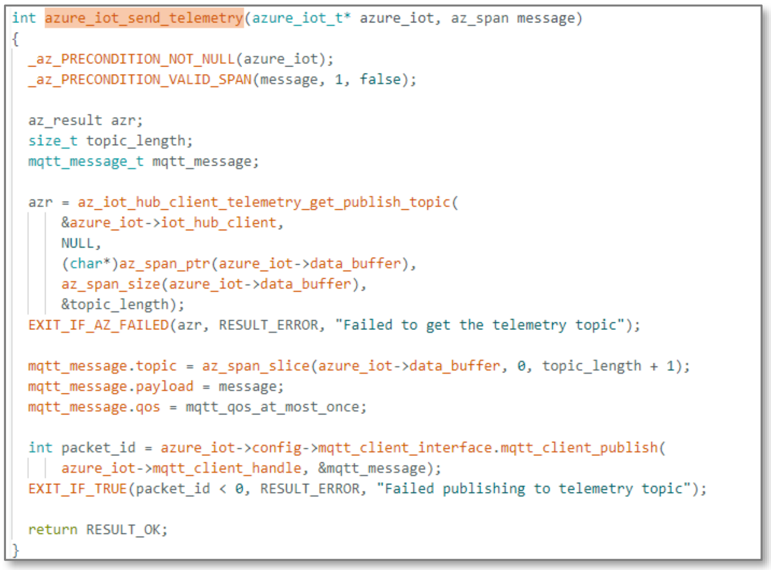


```
    azure_iot->data_buffer = azure_iot->config->data_buffer;

    result = get_mqtt_client_config_for_dps(azure_iot, &mqtt_client_config);
    azure_iot->state = azure_iot_state_connecting_to_dps;
  }
  else
  {
    result = get_mqtt_client_config_for_iot_hub(azure_iot, &mqtt_client_config);
    azure_iot->state = azure_iot_state_connecting_to_hub;
  }

  if (result != 0
      || azure_iot->config->mqtt_client_interface.mqtt_client_init(
             &mqtt_client_config, &azure_iot->mqtt_client_handle)
          != 0)
  {
    azure_iot->state = azure_iot_state_error;
    LogError("Failed initializing MQTT client.");
    return;
  }

  break;
```



```
case azure_iot_state_connecting_to_dps:
  break;
case azure_iot_state_connected_to_dps:
  // Subscribe to DPS topic.
  azure_iot->state = azure_iot_state_subscribing_to_dps;

  packet_id = azure_iot->config->mqtt_client_interface.mqtt_client_subscribe(
      azure_iot->mqtt_client_handle,
      AZ_SPAN_FROM_STR(AZ_IOT_PROVISIONING_CLIENT_REGISTER_SUBSCRIBE_TOPIC),
      mqtt_qos_at_most_once);

  if (packet_id < 0)
  {
    azure_iot->state = azure_iot_state_error;
    LogError("Failed subscribing to Azure Device Provisioning respose topic.");
    return;
  }

  break;
case azure_iot_state_subscribing_to_dps:
  break;
```

7.1.2.3. azure_iot_send_telemetry()



```
int azure_iot_send_telemetry(azure_iot_t* azure_iot, az_span message)
{
  _az_PRECONDITION_NOT_NULL(azure_iot);
  _az_PRECONDITION_VALID_SPAN(message, 1, false);

  az_result azr;
  size_t topic_length;
  mqtt_message_t mqtt_message;

  azr = az_iot_hub_client_telemetry_get_publish_topic(
      &azure_iot->iot_hub_client,
      NULL,
      (char*)az_span_ptr(azure_iot->data_buffer),
      az_span_size(azure_iot->data_buffer),
      &topic_length);
  EXIT_IF_AZ_FAILED(azr, RESULT_ERROR, "Failed to get the telemetry topic");

  mqtt_message.topic = az_span_slice(azure_iot->data_buffer, 0, topic_length + 1);
  mqtt_message.payload = message;
  mqtt_message.qos = mqtt_qos_at_most_once;

  int packet_id = azure_iot->config->mqtt_client_interface.mqtt_client_publish(
      azure_iot->mqtt_client_handle, &mqtt_message);
  EXIT_IF_TRUE(packet_id < 0, RESULT_ERROR, "Failed publishing to telemetry topic");

  return RESULT_OK;
}
```

7.1.3. Azure_IoT_PnP_Template.cpp

It contains the actual implementation of the IoT Plug and Play template. It fetches data from sensors and prepare payload for sending to the cloud, realize command handling.

7.1.3.1. Main Public Functions

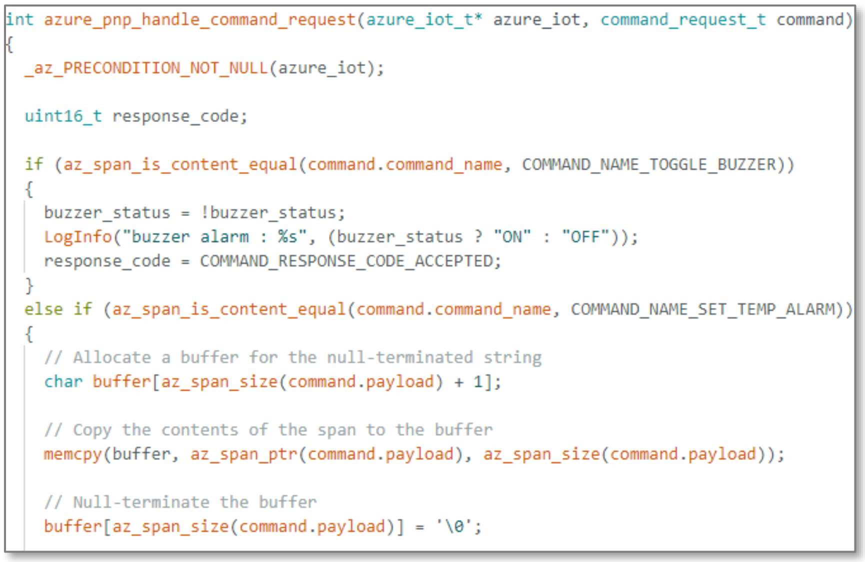
- azure_pnp_init(): Initializes the internal components of the IoT PnP module. This function should be called once by the application before any other Azure IoT-related function calls.
- azure_pnp_get_model_id(): Returns the model ID of the IoT Plug and Play template implemented by this device.
- azure_pnp_send_device_info(): Sends the device description to Azure IoT Central.
- azure_pnp_set_telemetry_frequency(): Sets with which minimum frequency this module should send telemetry to Azure IoT Central.
- azure_pnp_send_telemetry(): Sends telemetry data to Azure IoT Central.
- get_buzzer_status(): Set status for turning on or off led, and controlling buzzer working or not.
- get_temperature_alarm(): Get temperature limit value.
- get_gas_alarm(): Get gas limit value.
- azure_pnp_handle_command_request(): Executes the task requested by the command and sends a response back to Azure IoT Central.

7.1.3.2. Main Internal Functions

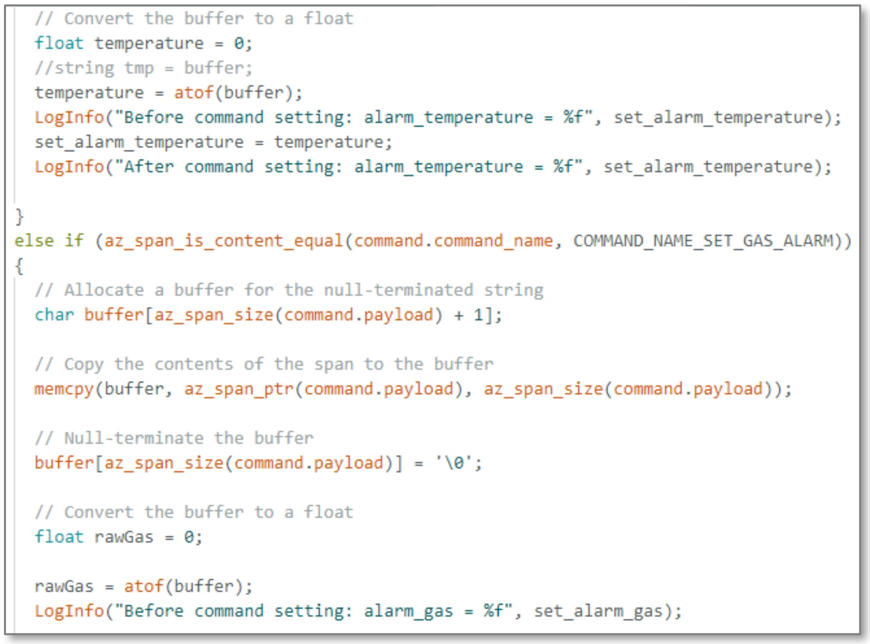
- generate_telemetry_payload()
- generate_device_info_payload()
- generate_properties_update_response()

7.1.3.3. azure_pnp_handle_command_request()

Here shows how to handle “Trigger Alarm” and “Set Temperature Alarm” command.

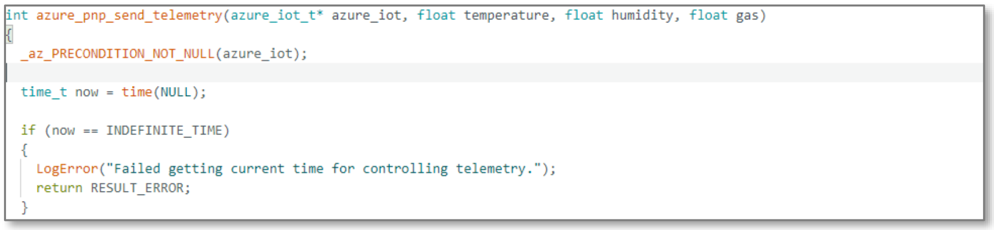


```
int azure_pnp_handle_command_request(azure_iot_t* azure_iot, command_request_t command)
{
  _az_PRECONDITION_NOT_NULL(azure_iot);

  uint16_t response_code;

  if (az_span_is_content_equal(command.command_name, COMMAND_NAME_TOGGLE_BUZZER))
  {
    buzzer_status = !buzzer_status;
    LogInfo("buzzer alarm : %s", (buzzer_status ? "ON" : "OFF"));
    response_code = COMMAND_RESPONSE_CODE_ACCEPTED;
  }
  else if (az_span_is_content_equal(command.command_name, COMMAND_NAME_SET_TEMP_ALARM))
  {
    // Allocate a buffer for the null-terminated string
    char buffer[az_span_size(command.payload) + 1];

    // Copy the contents of the span to the buffer
    memcpy(buffer, az_span_ptr(command.payload), az_span_size(command.payload));

    // Null-terminate the buffer
    buffer[az_span_size(command.payload)] = '\0';
```

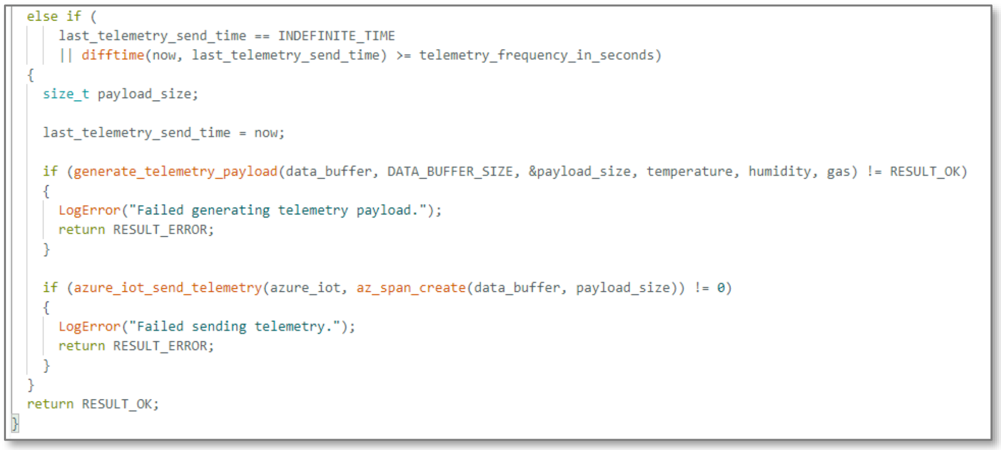


```
    // Convert the buffer to a float
    float temperature = 0;
    //string tmp = buffer;
    temperature = atof(buffer);
    LogInfo("Before command setting: alarm_temperature = %f", set_alarm_temperature);
    set_alarm_temperature = temperature;
    LogInfo("After command setting: alarm_temperature = %f", set_alarm_temperature);

  }
  else if (az_span_is_content_equal(command.command_name, COMMAND_NAME_SET_GAS_ALARM))
  {
    // Allocate a buffer for the null-terminated string
    char buffer[az_span_size(command.payload) + 1];

    // Copy the contents of the span to the buffer
    memcpy(buffer, az_span_ptr(command.payload), az_span_size(command.payload));

    // Null-terminate the buffer
    buffer[az_span_size(command.payload)] = '\0';

    // Convert the buffer to a float
    float rawGas = 0;

    rawGas = atof(buffer);
    LogInfo("Before command setting: alarm_gas = %f", set_alarm_gas);
```

7.1.3.4. azure_pnp_send_telemetry():



```
int azure_pnp_send_telemetry(azure_iot_t* azure_iot, float temperature, float humidity, float gas)
{
  _az_PRECONDITION_NOT_NULL(azure_iot);

  time_t now = time(NULL);

  if (now == INDEFINITE_TIME)
  {
    LogError("Failed getting current time for controlling telemetry.");
    return RESULT_ERROR;
  }
```



```
else if (
    last_telemetry_send_time == INDEFINITE_TIME
    || difftime(now, last_telemetry_send_time) >= telemetry_frequency_in_seconds)
{
  size_t payload_size;

  last_telemetry_send_time = now;

  if (generate_telemetry_payload(data_buffer, DATA_BUFFER_SIZE, &payload_size, temperature, humidity, gas) != RESULT_OK)
  {
    LogError("Failed generating telemetry payload.");
    return RESULT_ERROR;
  }

  if (azure_iot_send_telemetry(azure_iot, az_span_create(data_buffer, payload_size)) != 0)
  {
    LogError("Failed sending telemetry.");
    return RESULT_ERROR;
  }
}
return RESULT_OK;
}
```

7.1.4. iot_configs.h

This is a configuration file, sets up parameters for connecting an IoT device to Microsoft Azure IoT Central or Azure IoT Hub. It configures Wi-Fi, authentication, telemetry settings, and other important device-specific information.

– **Wi-Fi Configuration:** Specifies the Wi-Fi network's SSID and password for the IoT device to connect to the network.

– **Device Authentication Method:** Allows the choice of authentication method (X.509 certificates or a device key).

– **X.509 Certificate and Private Key (if X.509 is enabled):** Contains the device's X.509 certificate and private key (if X.509 authentication is used).

– **Azure IoT Central Configuration:** Specifies the Device Provisioning Service (DPS) ID scope, device ID, and device key (if not using X.509 certificates).

– **User-Agent Configuration:** Defines a user-agent string to identify the device when connecting to Azure IoT Services.

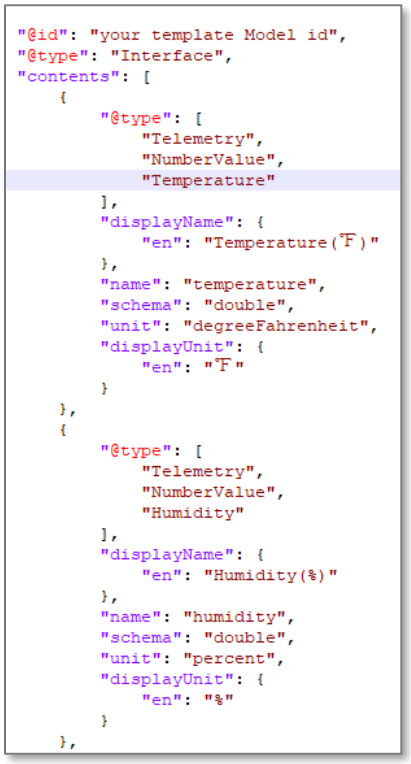
– **Telemetry and MQTT Password Lifetime Configuration:** Sets the telemetry message publishing frequency and the duration for which the MQTT password (SAS token) remains valid before regenerating it and reconnecting.

7.2. Cloud Code

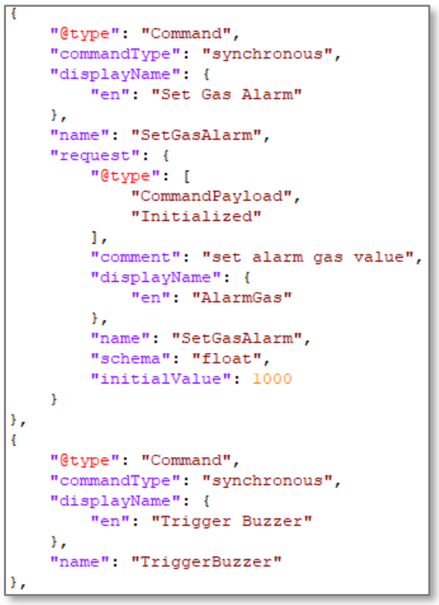
7.2.1. temperature_Gas_Humidity.json

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

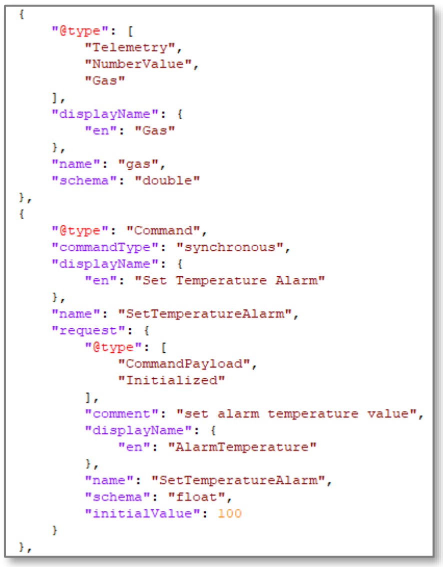
"Azure_IoT_PnP_Template.cpp" file is then invoked, directing the execution to the corresponding module to implement the requisite actions.



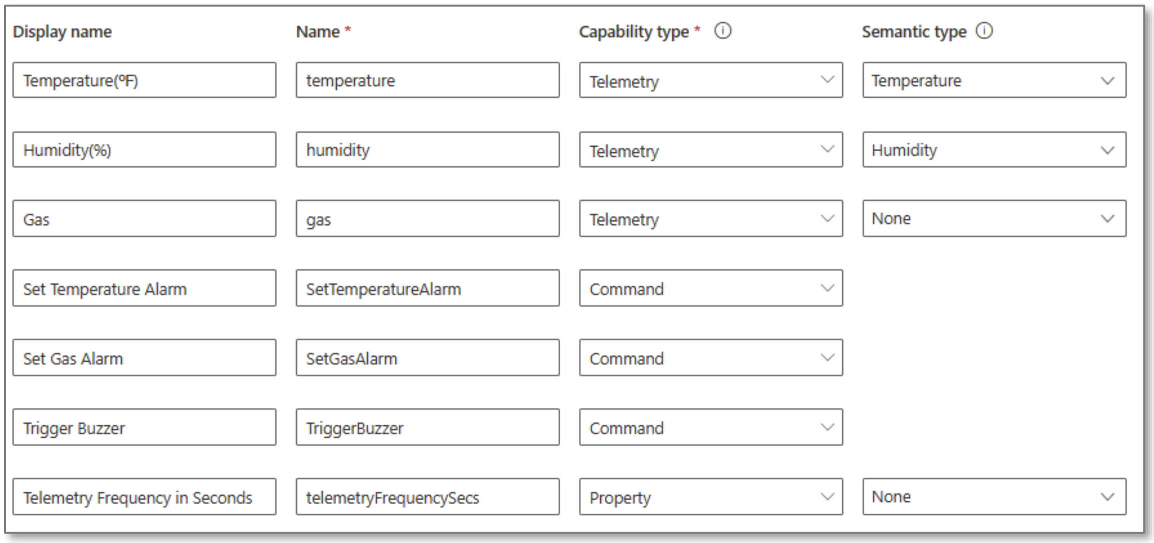
```
"@id": "your template Model id",
"@type": "Interface",
"contents": [
    {
        "@type": [
            "Telemetry",
            "NumberValue",
            "Temperature"
        ],
        "displayName": {
            "en": "Temperature(℉)"
        },
        "name": "temperature",
        "schema": "double",
        "unit": "degreeFahrenheit",
        "displayUnit": {
            "en": "℉"
        }
    },
    {
        "@type": [
            "Telemetry",
            "NumberValue",
            "Humidity"
        ],
        "displayName": {
            "en": "Humidity(%)"
        },
        "name": "humidity",
        "schema": "double",
        "unit": "percent",
        "displayUnit": {
            "en": "%"
        }
    },
```



```
{
    "@type": "Command",
    "commandType": "synchronous",
    "displayName": {
        "en": "Set Gas Alarm"
    },
    "name": "SetGasAlarm",
    "request": {
        "@type": [
            "CommandPayload",
            "Initialized"
        ],
        "comment": "set alarm gas value",
        "displayName": {
            "en": "AlarmGas"
        },
        "name": "SetGasAlarm",
        "schema": "float",
        "initialValue": 1000
    }
},
{
    "@type": "Command",
    "commandType": "synchronous",
    "displayName": {
        "en": "Trigger Buzzer"
    },
    "name": "TriggerBuzzer"
},
```



```
{
    "@type": [
        "Telemetry",
        "NumberValue",
        "Gas"
    ],
    "displayName": {
        "en": "Gas"
    },
    "name": "gas",
    "schema": "double"
},
{
    "@type": "Command",
    "commandType": "synchronous",
    "displayName": {
        "en": "Set Temperature Alarm"
    },
    "name": "SetTemperatureAlarm",
    "request": {
        "@type": [
            "CommandPayload",
            "Initialized"
        ],
        "comment": "set alarm temperature value",
        "displayName": {
            "en": "AlarmTemperature"
        },
        "name": "SetTemperatureAlarm",
        "schema": "float",
        "initialValue": 100
    }
},
```



Display name	Name *	Capability type *	Semantic type
Temperature(℉)	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.

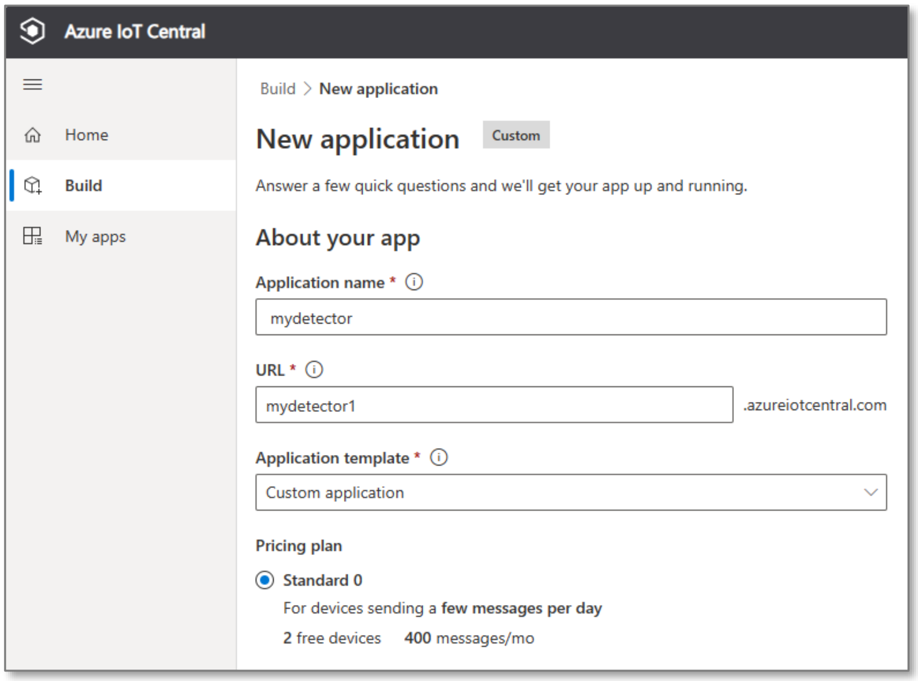
8.4. Install the DHT sensor library on Arduino IDE

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



New application

Answer a few quick questions and we'll get your app up and running.

About your app

Application name * : mydetector

URL * : mydetector1 .azureiotcentral.com

Application template * : Custom application

Pricing plan

Standard 0

For devices sending a few messages per day

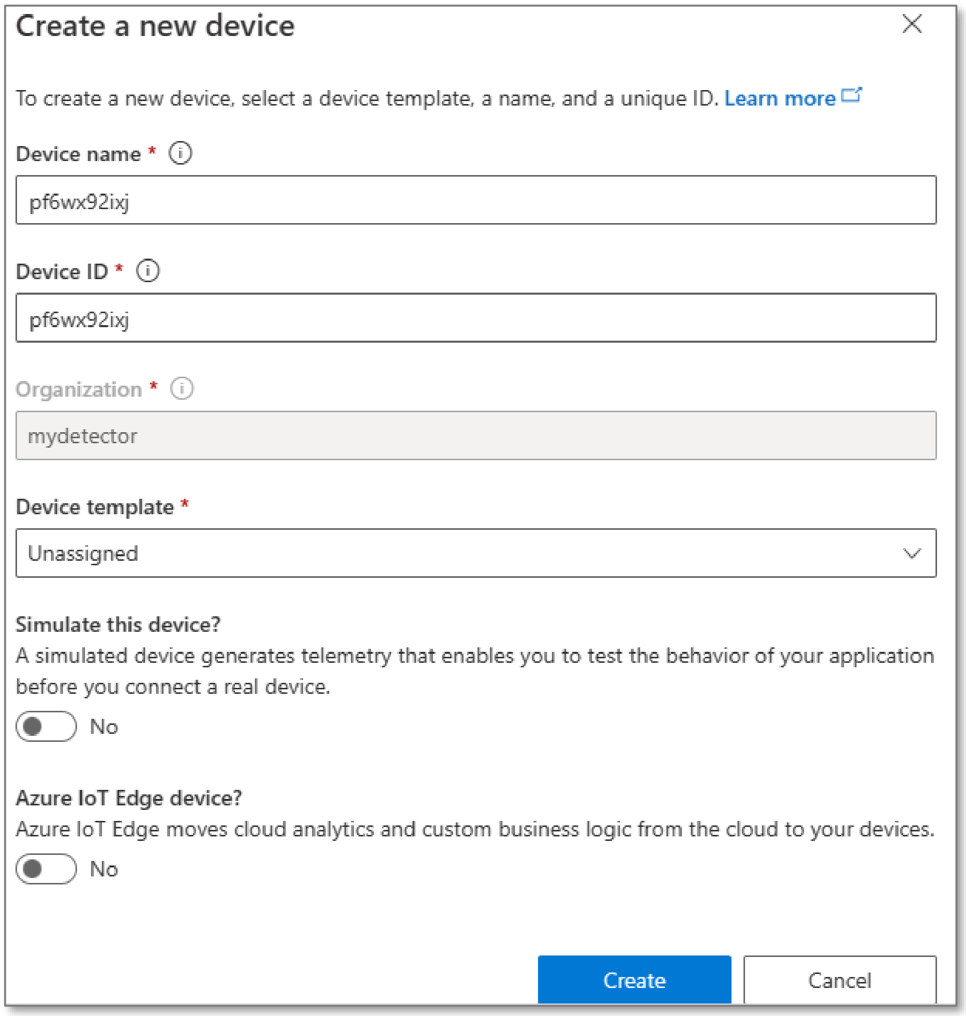
2 free devices 400 messages/mo

- Create Azure IoT account using student account of your university
- Navigate to Azure IoT Central portal, select **Build** on left menu.
- Select **Custom app**, click “**Create app”**.
- Define your Application **Name** and a **URL**.
- Choose Application template as “**Customer application**”
- Choose the pricing plan **Standard 0.**

– Click **Create**.

– After IoT Central provisions the application, it redirects automatically to the new application dashboard.

9.2. Create my IoT device on Azure IoT Central platform


Create a new device

To create a new device, select a device template, a name, and a unique ID. Learn more

Device name * pf6wx92ixj

Device ID * pf6wx92ixj

Organization * mydetector

Device template * Unassigned

Simulate this device?
A simulated device generates telemetry that enables you to test the behavior of your application before you connect a real device.
No

Azure IoT Edge device?
Azure IoT Edge moves cloud analytics and custom business logic from the cloud to your devices.
No

– Click **Devices** on the left menu.

– Click **+ New** to jump to the **Create a new device** window.

– Set **Device template** as **Unassigned**.

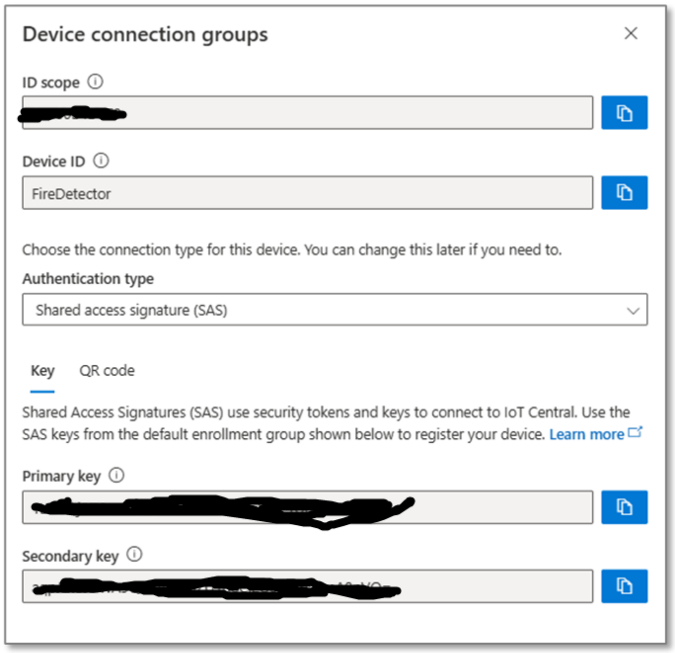
– Fill in **Device name** and **Device ID**.

– Select the **Create** button.

– The newly created device will be found in the **All devices**. Click the device name, we can check details of device.

– Click **Connect** in the top right menu and a new window will pop out. Take down:

1) **ID scope**
2) **Device ID**
3) **Primary key** (only if we choose Symmetric Key authentication)



Device connection groups

ID scope

[redacted]

Device ID

FireDetector

Choose the connection type for this device. You can change this later if you need to.

Authentication type

Shared access signature (SAS)

Shared Access Signatures (SAS) use security tokens and keys to connect to IoT Central. Use the SAS keys from the default enrollment group shown below to register your device. Learn more

Primary key

[redacted]

Secondary key

[redacted]

9.3. Create custom device template

Note: If you know my project **template Model id**, you don't need to create one, when you run the project .ino file in the Arduino IDE, it will automatically load the template for you based on the template Model id. However, I won't leak my project **template Model id** for security. So, please follow my steps to create your own template.

– Click **Devices** templates on the left menu.

– Click **+ New**, then choose IoT Device in the **Create a custom device template** area.

– Give a name for your device template, then click **View**.

– Click the **Create** button.

– The newly created device will be found in the **Device templates**. Click the device name, you can check details of device template.

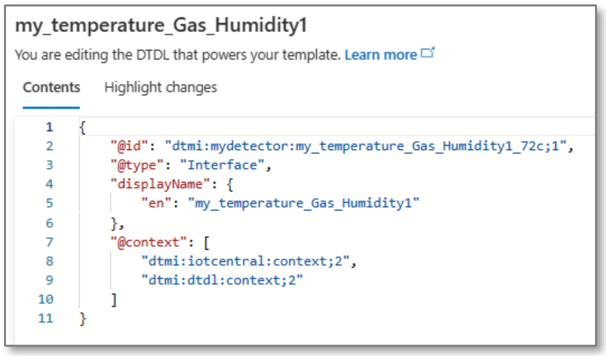
– Click **Edit DTDL**, there is some code.

– Copy **"@id": "dtmi:mydetector:my_temperature_Gas_Humidity1_72c;1"**, for example.

– Replace the first line in my code of "**temperature_Gas_Humidity.json**" with **"@id": "dtmi:mydetector:my_temperature_Gas_Humidity1_72c;1"**.

– Delete all the information automatically created for your custom template, and copy the whole replaced code of “**temperature_Gas_Humidity.json**” into your custom template.

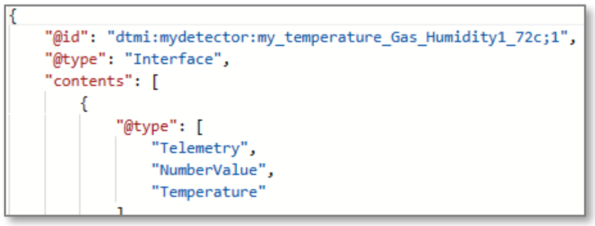
– Click **Save** button.

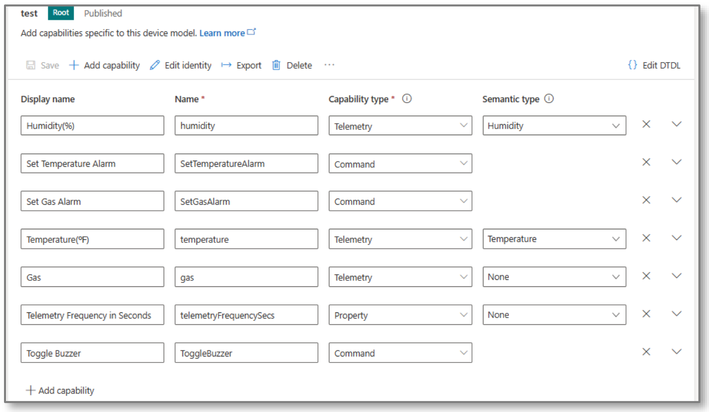

my_temperature_Gas_Humidity1

You are editing the DTDL that powers your template. Learn more

Contents Highlight changes

```
{
    "@id": "dtmi:mydetector:my_temperature_Gas_Humidity1_72c;1",
    "@type": "Interface",
    "displayName": {
        "en": "my_temperature_Gas_Humidity1"
    },
    "@context": [
        "dtmi:iotcentral:context;2",
        "dtmi:dtdl:context;2"
    ]
}
```


```
{
    "@id": "dtmi:mydetector:my_temperature_Gas_Humidity1_72c;1",
    "@type": "Interface",
    "contents": [
        {
            "@type": [
                "Telemetry",
                "NumberValue",
                "Temperature"
```


test Root Published

Add capabilities specific to this device model. Learn more

Save + Add capability Edit identity Export Delete ··· {} Edit DTDL

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

+ Add capability

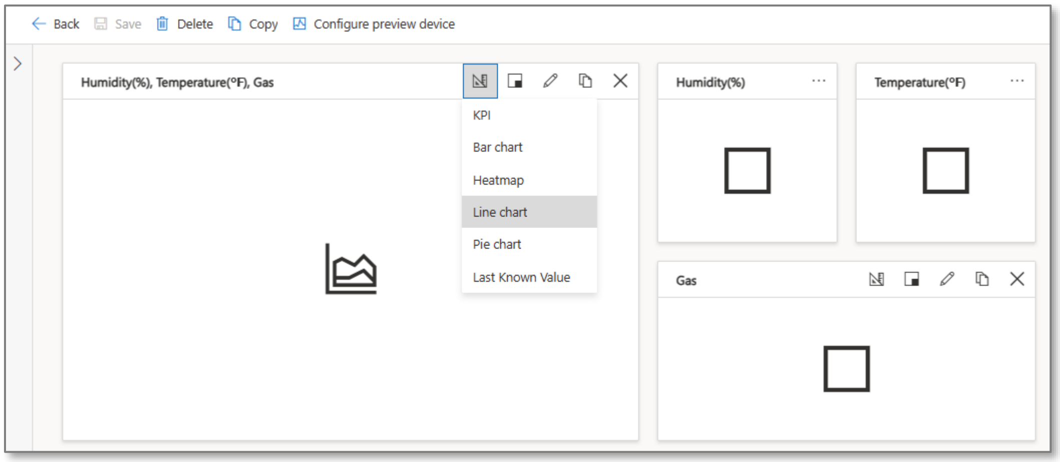
– Navigate to **Views**, click **Views**, select **Generate default views**.

– Click **Generate default dashboard view(s)** button

– Navigate to Views > Overview. Set the Humidity (%), Temperature (°F), Gas chart as “**Line chart**” in the **Change Visualization**. Set the independent Humidity (%), Temperature (°F), Gas charts as “**Last Known Value**” in their **Change Visualization**.

– Click the **Save** button.

– Click the **Publish** button.



Note:

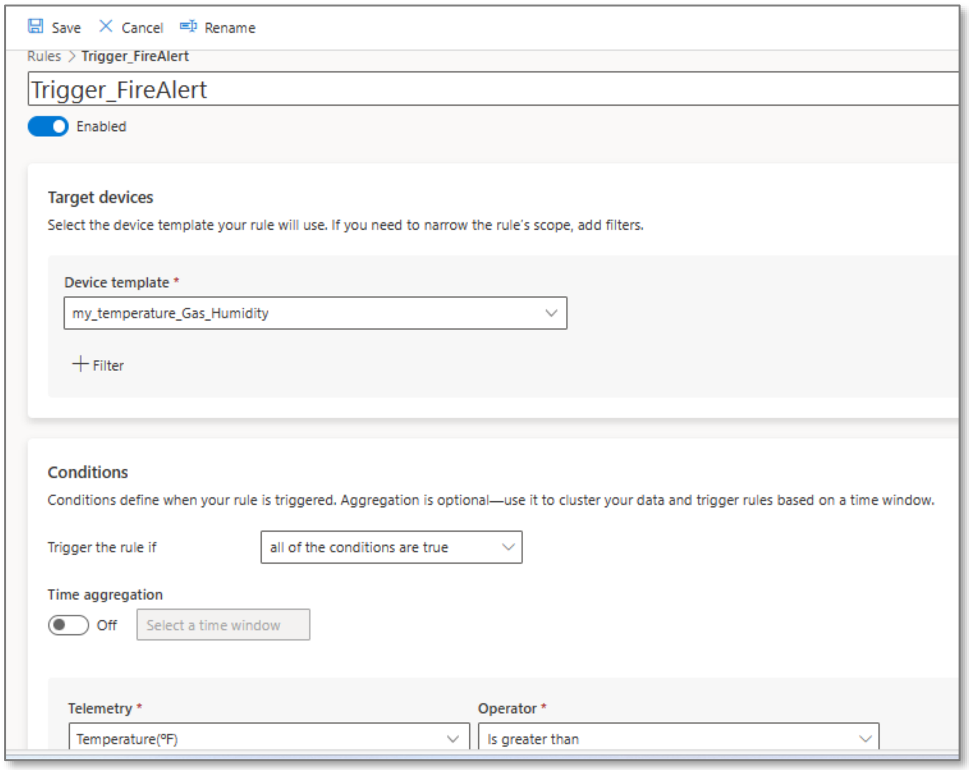
1) Get your template **Model id**, which is in the Json code “@id” line of template.
2) Get your device connection group information, take down “**ID scope**”, “**Device ID**” and “**Primary key**”.

9.4. Create email alert by using Rules

Navigate to Extend > Rules.

– Click **Rules** on the left menu.

– Click **+ New**, then create email alert 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

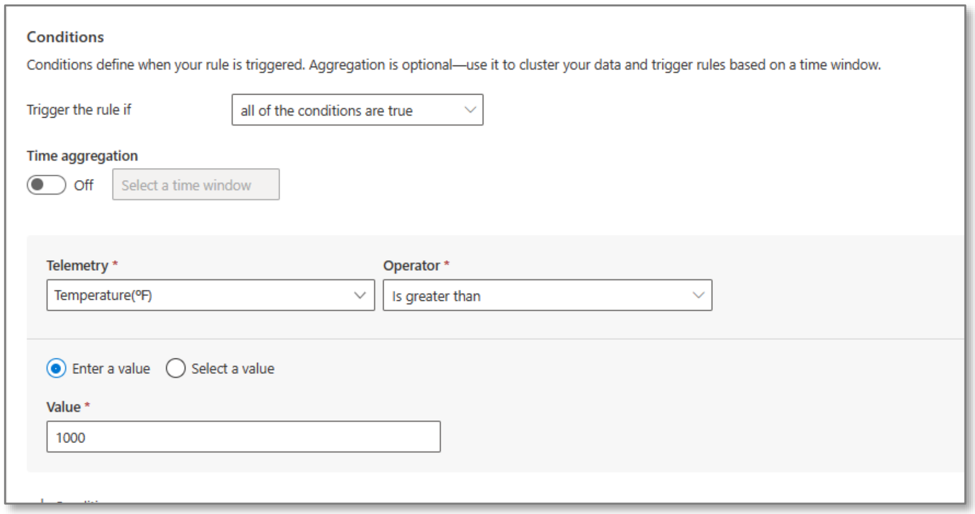
Conditions define when your rule is triggered. Aggregation is optional—use it to cluster your data and trigger rules based on a time window.

Trigger the rule if: all of the conditions are true

Time aggregation: Off — Select a time window

Telemetry *: Temperature(ºF)

Operator *: Is greater than


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

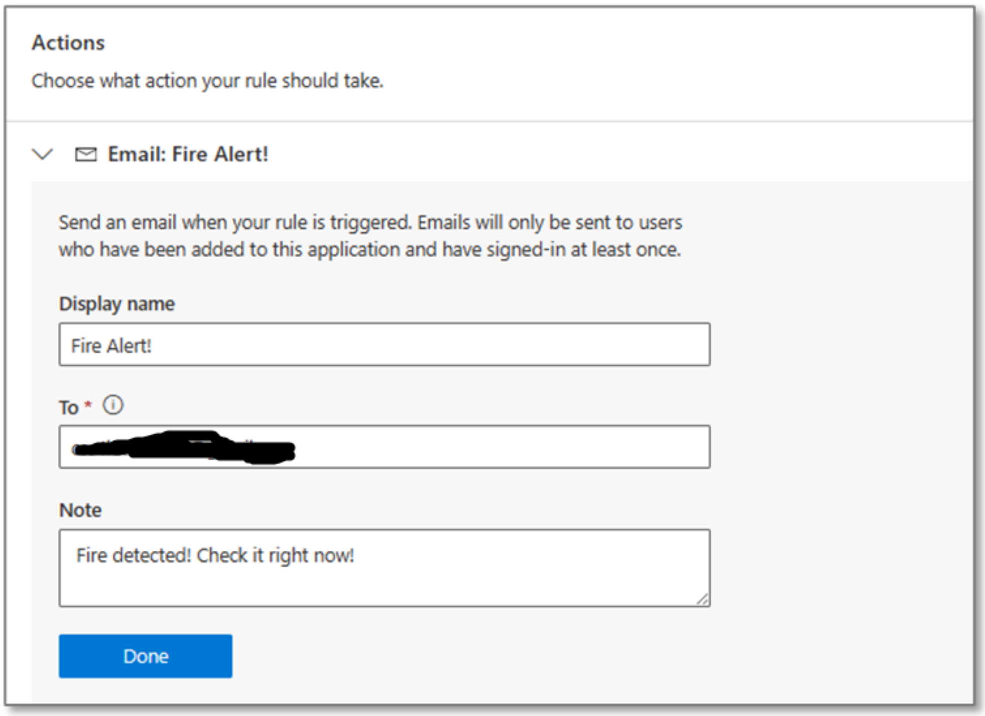
Time aggregation: Off — Select a time window

Telemetry *: Temperature(ºF)

Operator *: Is greater than

[✓ Enter a value] Select a value

Value *: 1000



Actions

Choose what action your rule should take.

Email: Fire Alert!

Send an email when your rule is triggered. Emails will only be sent to users who have been added to this application and have signed-in at least once.

Display name: Fire Alert!

To *: [redacted]

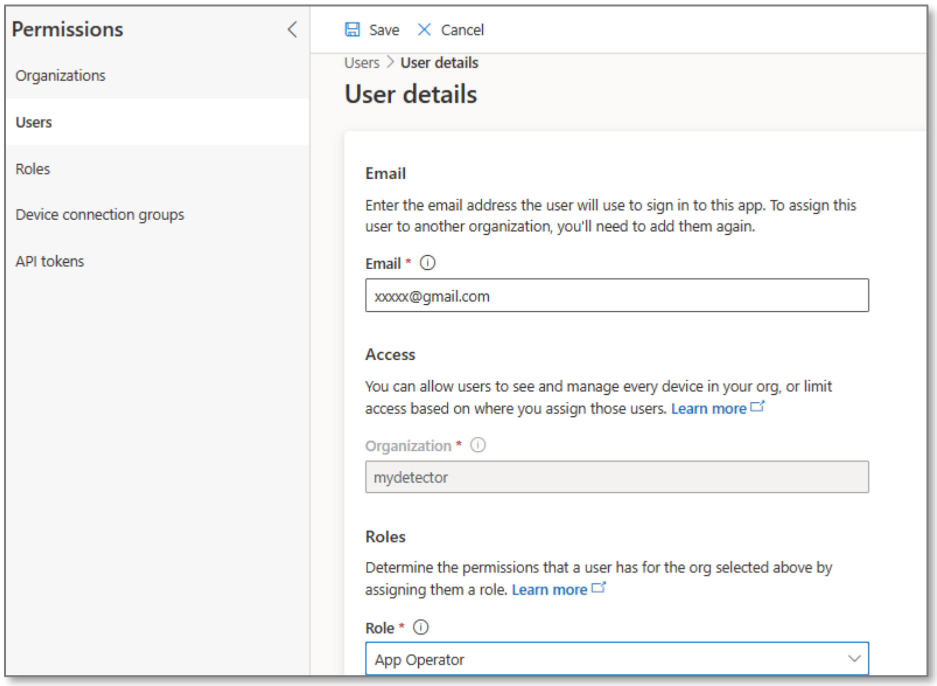
Note: Fire detected! Check it right now!

Note:

1) Apply an email account to receiving the email alert of this system.
2) The email address should have signed in to the application at least once.

   Navigate to: Security > Permissions > Users

   Click **+ Assign User** to add new user.



User details

Email

Enter the email address the user will use to sign in to this app. To assign this user to another organization, you'll need to add them again.

Email *: xxxxx@gmail.com

Access

You can allow users to see and manage every device in your org, or limit access based on where you assign those users. Learn more

Organization *: mydetector

Roles

Determine the permissions that a user has for the org selected above by assigning them a role. Learn more

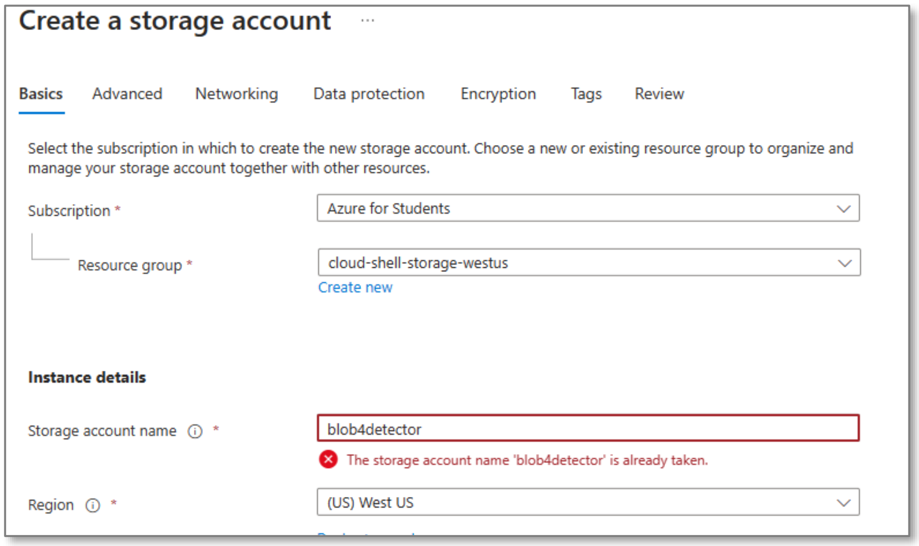
Role *: App Operator

9.5. Create Blob Storage

It is for storing received telemetry from physical devices.

Login to Azure account, open **Storage accounts.**

- Create a storage account.
- Open created storage account, create a container in the storage account.
- Navigate to IoT Central -> Data export -> New export
- Set Destination as your newly storage account.



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.

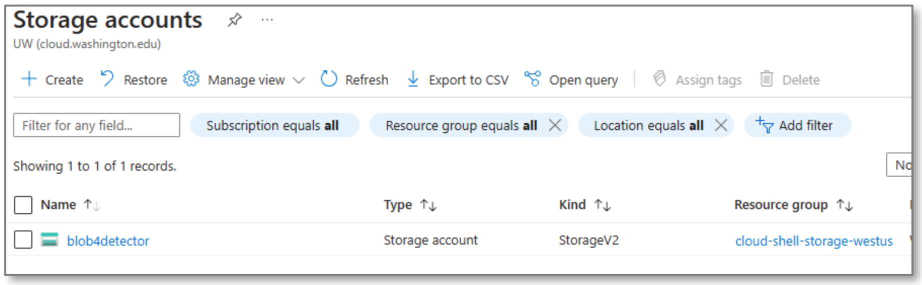
Subscription *	Azure for Students
Resource group *	cloud-shell-storage-westus

Create new

Instance details

Storage account name *	blob4detector
Region *	(US) West US

The storage account name 'blob4detector' is already taken.

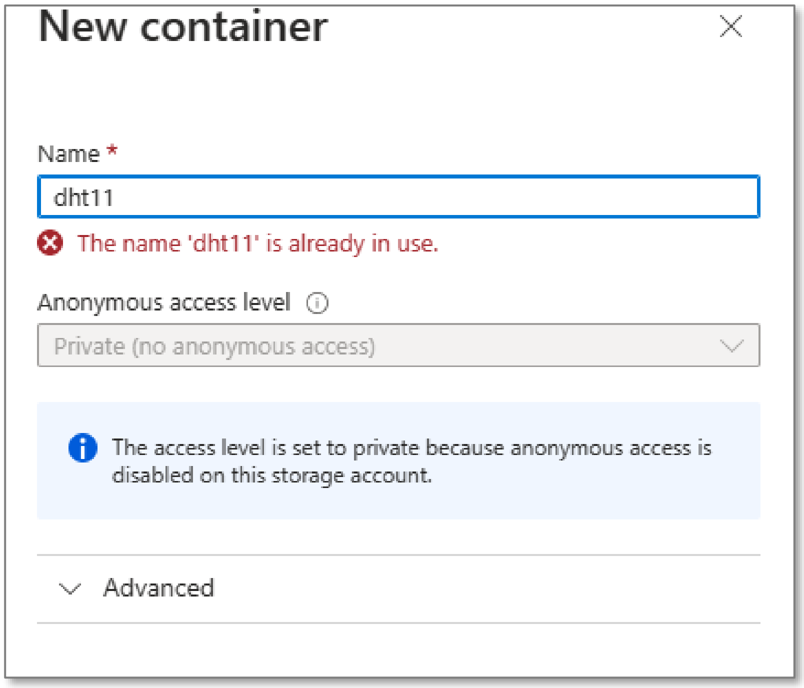


Storage accounts

UW (cloud.washington.edu)

Showing 1 to 1 of 1 records.

Name	Type	Kind	Resource group
blob4detector	Storage account	StorageV2	cloud-shell-storage-westus


New container

Name *

dht11

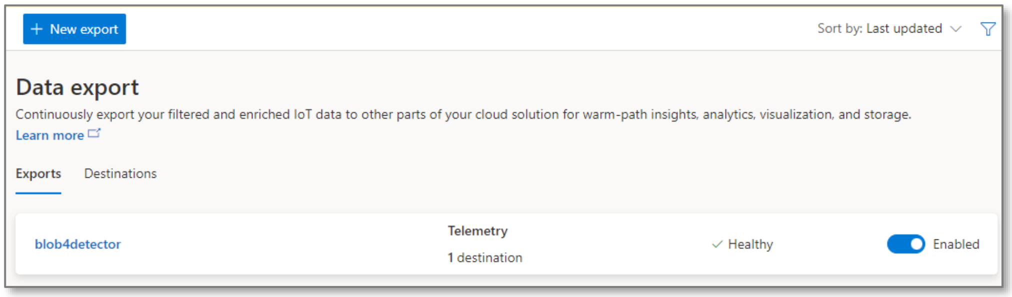
The name 'dht11' is already in use.

Anonymous access level

Private (no anonymous access)

The access level is set to private because anonymous access is disabled on this storage account.

Advanced


+ New export

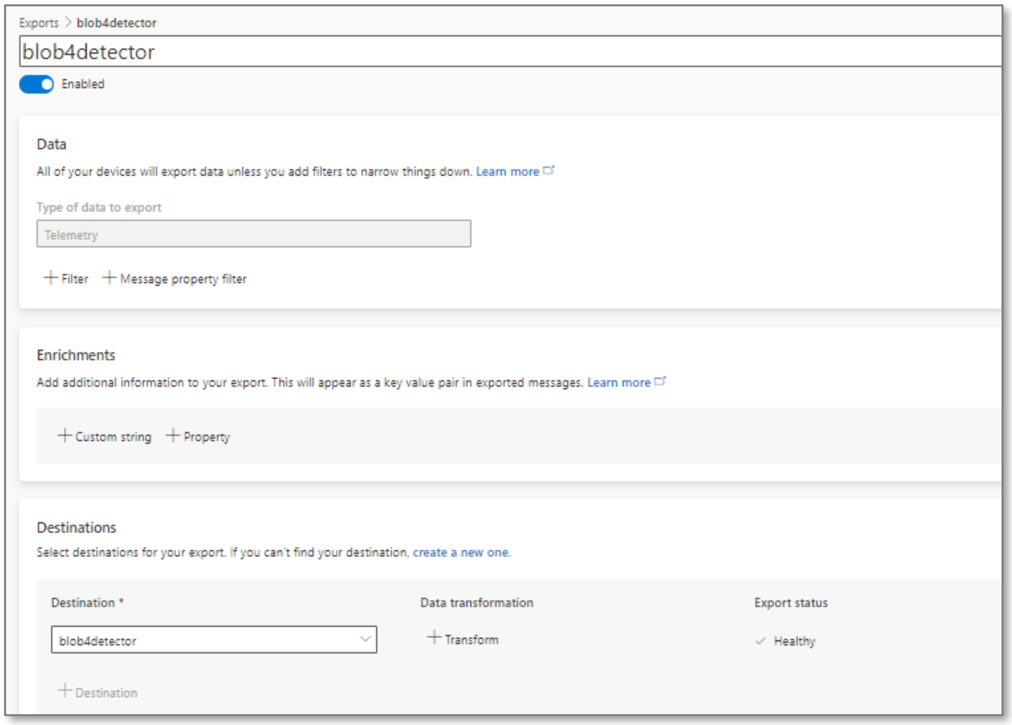
Sort by: Last updated

Data export

Continuously export your filtered and enriched IoT data to other parts of your cloud solution for warm-path insights, analytics, visualization, and storage. Learn more

Exports | Destinations

blob4detector	Telemetry 1 destination	✓ Healthy	Enabled


Exports > blob4detector

blob4detector

Enabled

Data

All of your devices will export data unless you add filters to narrow things down. Learn more

Type of data to export: Telemetry

+ Filter + Message property filter

Enrichments

Add additional information to your export. This will appear as a key value pair in exported messages. Learn more

+ Custom string + Property

Destinations

Select destinations for your export. If you can't find your destination, create a new one.

Destination *	Data transformation	Export status
blob4detector	+ Transform	✓ Healthy

+ Destination

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.

If all the connections are successful, you will see this information:

```
21:43:02.006 -> ..............................................................................
21:43:46.511 -> 1970/1/1 00:01:11 [INFO] WiFi connected, IP address: 192.168.137.83
21:43:46.511 -> 1970/1/1 00:01:11 [INFO] Setting time using SNTP
21:43:47.000 -> .......
21:43:49.975 -> 2023/11/26 05:43:48 [INFO] Time initialized!
21:43:49.975 -> 2023/11/26 05:43:48 [INFO] Telemetry frequency set to once every 10 seconds.
21:43:50.020 -> 2023/11/26 05:43:48 [INFO] Azure IoT client initialized (state=2)
21:43:50.020 -> 2023/11/26 05:43:48 [INFO] MQTT client target uri set to 'mqtts://global.azure-devices-
provisioning.net'
21:43:50.020 -> 2023/11/26 05:43:48 [INFO] MQTT client connecting.
21:43:52.338 -> 2023/11/26 05:43:51 [INFO] MQTT client connected (session_present=0).
21:43:52.338 -> 2023/11/26 05:43:51 [INFO] MQTT client subscribing to '$dps/registrations/res/#'
21:43:52.410 -> 2023/11/26 05:43:51 [INFO] MQTT topic subscribed (message id=24093).
21:43:52.410 -> 2023/11/26 05:43:51 [INFO] MQTT client publishing to '$dps/registrations/PUT/iotdps-
register/?$rid=1'
21:43:52.969 -> 2023/11/26 05:43:51 [INFO] MQTT message received.
21:43:52.969 -> 2023/11/26 05:43:51 [INFO] MQTT client publishing to '$dps/registrations/GET/iotdps-get-
operationstatus/?$rid=1&operationId=4.d4d9f80bcb64c6f9.ce57e2b8-43fa-484c-b103-bf866252b2a1'
21:43:53.280 -> 2023/11/26 05:43:51 [INFO] MQTT message received.
21:43:55.269 -> 2023/11/26 05:43:54 [INFO] MQTT client publishing to '$dps/registrations/GET/iotdps-get-
operationstatus/?$rid=1&operationId=4.d4d9f80bcb64c6f9.ce57e2b8-43fa-484c-b103-bf866252b2a1'
21:43:55.725 -> 2023/11/26 05:43:54 [INFO] MQTT message received.
21:43:55.725 -> 2023/11/26 05:43:54 [INFO] MQTT client being disconnected.
21:43:55.788 -> 2023/11/26 05:43:54 [INFO] MQTT client target uri set to 'mqtts://iotc-872c7377-d9b2-4f6e-
b62d-83ef036b7f8f.azure-devices.net'
21:43:55.825 -> 2023/11/26 05:43:54 [INFO] MQTT client connecting.
21:43:57.655 -> 2023/11/26 05:43:56 [INFO] MQTT client connected (session_present=0).
21:43:57.655 -> 2023/11/26 05:43:56 [INFO] MQTT client subscribing to '$iothub/methods/POST/#'
21:43:57.880 -> 2023/11/26 05:43:56 [INFO] MQTT topic subscribed (message id=3351).
21:43:57.880 -> 2023/11/26 05:43:56 [INFO] MQTT client subscribing to '$iothub/twin/res/#'
21:43:58.173 -> 2023/11/26 05:43:56 [INFO] MQTT topic subscribed (message id=11656).
21:43:58.173 -> 2023/11/26 05:43:56 [INFO] MQTT client subscribing to
'$iothub/twin/PATCH/properties/desired/#'
```

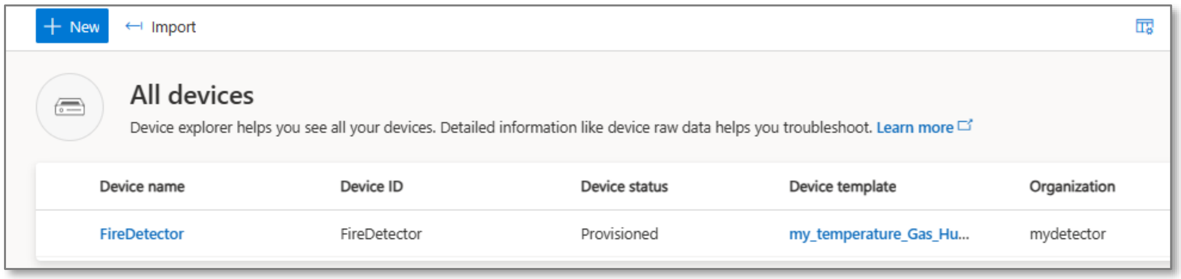
```
21:43:58.245 -> 2023/11/26 05:43:56 [INFO] MQTT topic subscribed (message id=9468).
21:43:58.245 -> 2023/11/26 05:43:56 [INFO] MQTT client publishing to 'devices/FireDetector/messages/events/'
```

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 has updated to **my_temperature_Gas_Humidity**.



All devices

Device explorer helps you see all your devices. Detailed information like device raw data helps you troubleshoot. Learn more

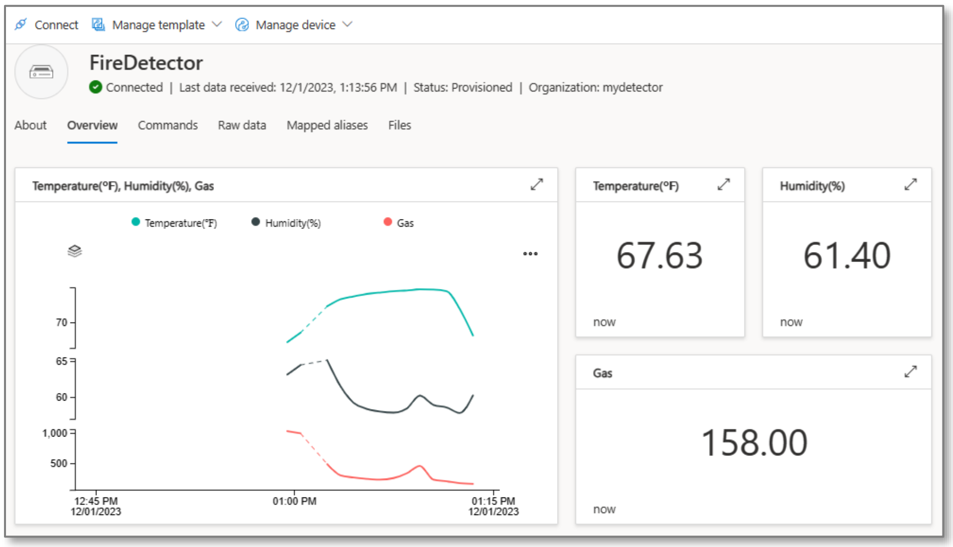
Device name	Device ID	Device status	Device template	Organization
FireDetector	FireDetector	Provisioned	my_temperature_Gas_Hu...	mydetector

16. Check data visualization

You can view the flow of telemetry from your device to the cloud with Azure IoT Central.

Navigate Devices > All devices > your device.

In my project, Gas value is detected by my MQ2 sensor, Humidity and Temperature values are detected by my DHT22 sensor.



FireDetector

Connected | Last data received: 12/1/2023, 1:13:56 PM | Status: Provisioned | Organization: mydetector

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.

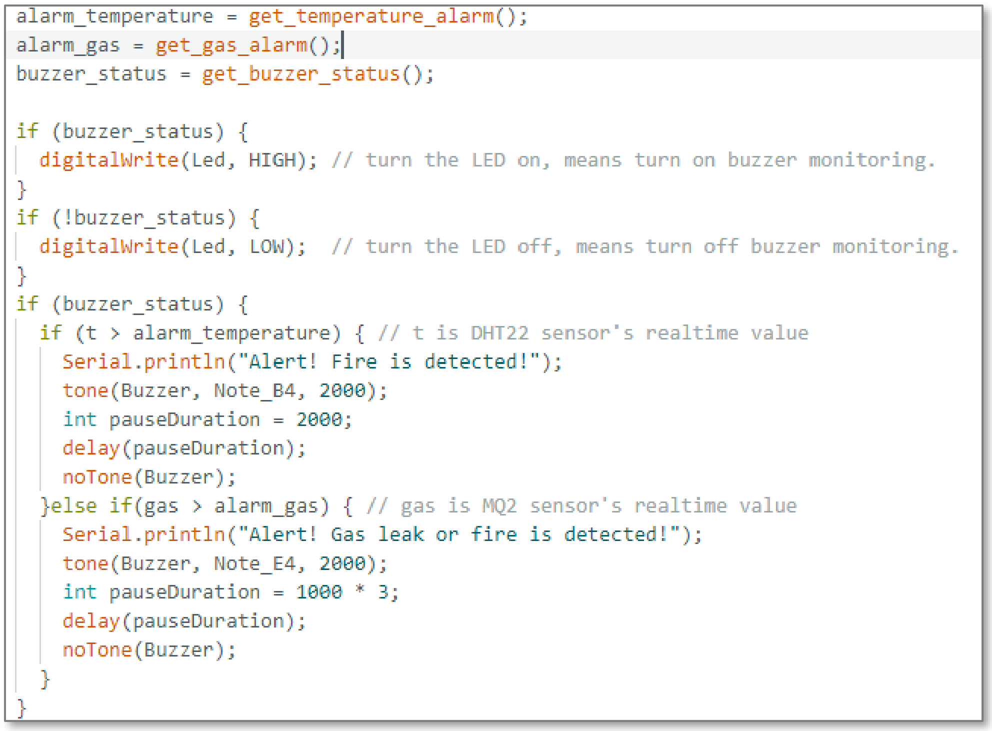
17.2. Set Gas Alarm

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

17.3. Set Temperature Alarm

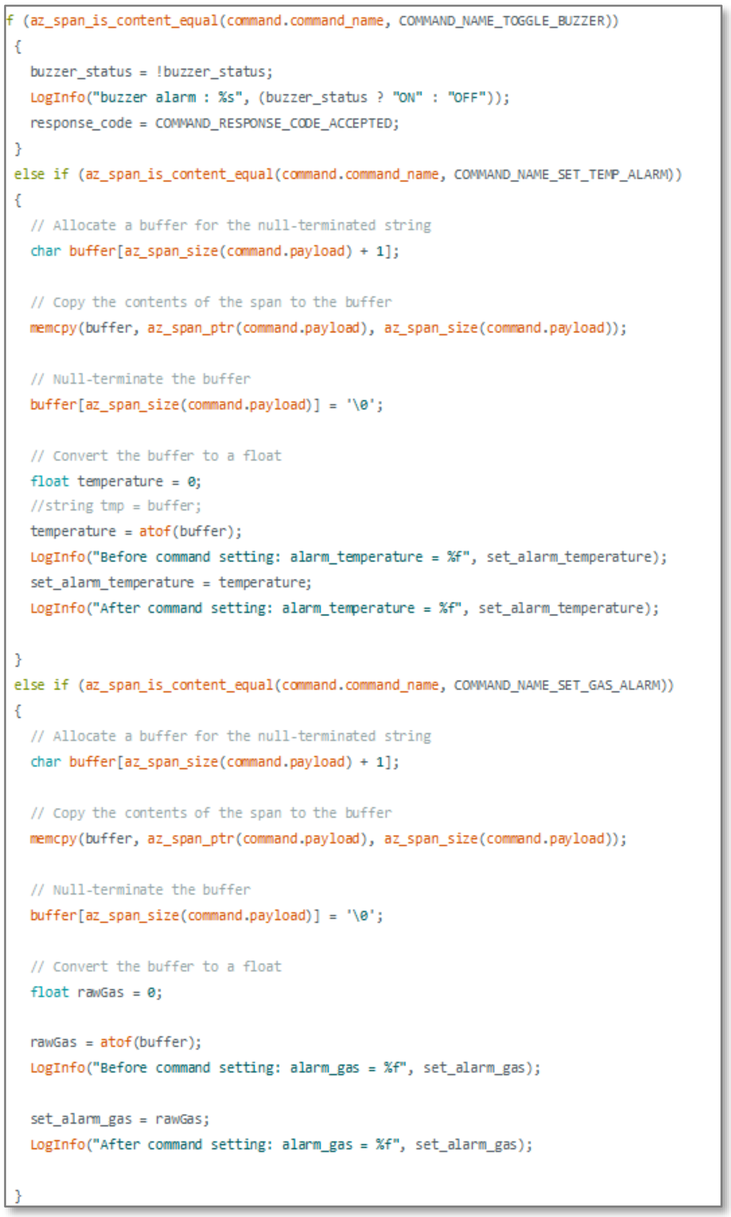
Change the temperature limit value, which makes my project to be more possible applications in different situations.

17.4. Buzzer alarm and controlling logic module



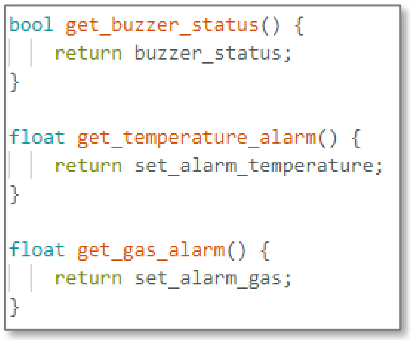
```
alarm_temperature = get_temperature_alarm();
alarm_gas = get_gas_alarm();
buzzer_status = get_buzzer_status();

if (buzzer_status) {
  digitalWrite(Led, HIGH); // turn the LED on, means turn on buzzer monitoring.
}
if (!buzzer_status) {
  digitalWrite(Led, LOW);  // turn the LED off, means turn off buzzer monitoring.
}
if (buzzer_status) {
  if (t > alarm_temperature) { // t is DHT22 sensor's realtime value
    Serial.println("Alert! Fire is detected!");
    tone(Buzzer, Note_B4, 2000);
    int pauseDuration = 2000;
    delay(pauseDuration);
    noTone(Buzzer);
  }else if(gas > alarm_gas) { // gas is MQ2 sensor's realtime value
    Serial.println("Alert! Gas leak or fire is detected!");
    tone(Buzzer, Note_E4, 2000);
    int pauseDuration = 1000 * 3;
    delay(pauseDuration);
    noTone(Buzzer);
  }
}
```

17.5. 3 commands handle code



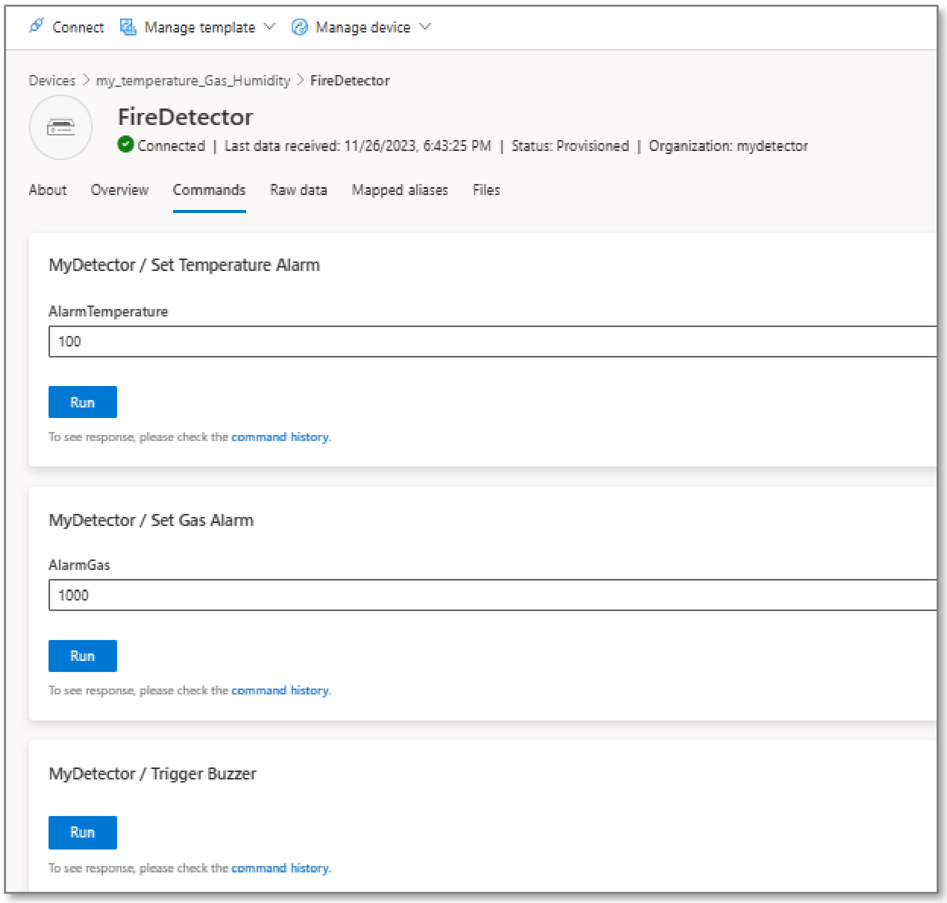
```
f (az_span_is_content_equal(command.command_name, COMMAND_NAME_TOGGLE_BUZZER))
{
  buzzer_status = !buzzer_status;
  LogInfo("buzzer alarm : %s", (buzzer_status ? "ON" : "OFF"));
  response_code = COMMAND_RESPONSE_CODE_ACCEPTED;
}
else if (az_span_is_content_equal(command.command_name, COMMAND_NAME_SET_TEMP_ALARM))
{
  // Allocate a buffer for the null-terminated string
  char buffer[az_span_size(command.payload) + 1];

  // Copy the contents of the span to the buffer
  memcpy(buffer, az_span_ptr(command.payload), az_span_size(command.payload));

  // Null-terminate the buffer
  buffer[az_span_size(command.payload)] = '\0';

  // Convert the buffer to a float
  float temperature = 0;
  //string tmp = buffer;
  temperature = atof(buffer);
  LogInfo("Before command setting: alarm_temperature = %f", set_alarm_temperature);
  set_alarm_temperature = temperature;
  LogInfo("After command setting: alarm_temperature = %f", set_alarm_temperature);

}
else if (az_span_is_content_equal(command.command_name, COMMAND_NAME_SET_GAS_ALARM))
{
  // Allocate a buffer for the null-terminated string
  char buffer[az_span_size(command.payload) + 1];

  // Copy the contents of the span to the buffer
  memcpy(buffer, az_span_ptr(command.payload), az_span_size(command.payload));

  // Null-terminate the buffer
  buffer[az_span_size(command.payload)] = '\0';

  // Convert the buffer to a float
  float rawGas = 0;

  rawGas = atof(buffer);
  LogInfo("Before command setting: alarm_gas = %f", set_alarm_gas);

  set_alarm_gas = rawGas;
  LogInfo("After command setting: alarm_gas = %f", set_alarm_gas);

}
```

17.6. Tool functions for commands



```
bool get_buzzer_status() {
    return buzzer_status;
}

float get_temperature_alarm() {
    return set_alarm_temperature;
}

float get_gas_alarm() {
    return set_alarm_gas;
}
```

17.7. Commands effects in IoT Central


Connect | Manage template | Manage device

Devices > my_temperature_Gas_Humidity > FireDetector

FireDetector

Connected | Last data received: 11/26/2023, 6:43:25 PM | Status: Provisioned | Organization: mydetector

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

MyDetector / Set Temperature Alarm

AlarmTemperature: 100

Run

To see response, please check the command history.

MyDetector / Set Gas Alarm

AlarmGas: 1000

Run

To see response, please check the command history.

MyDetector / Trigger Buzzer

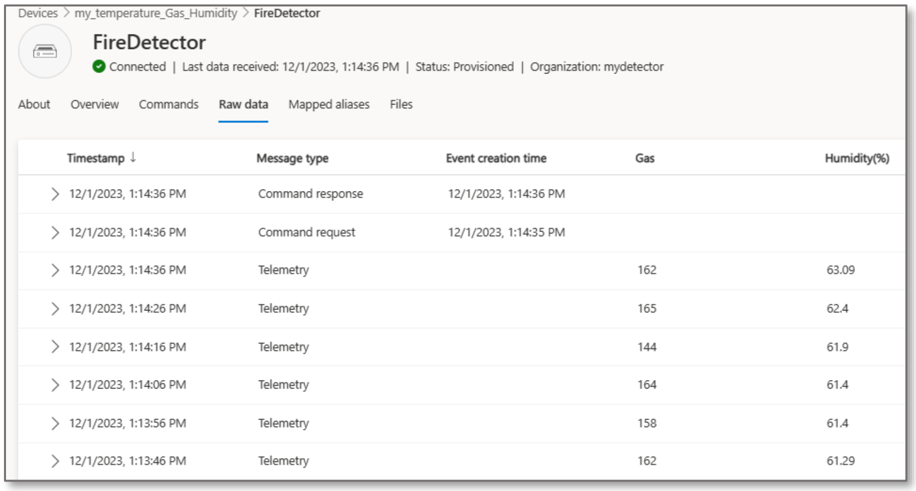
Run

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.

In my project, there are gas, humidity and temperature.


Devices > my_temperature_Gas_Humidity > FireDetector

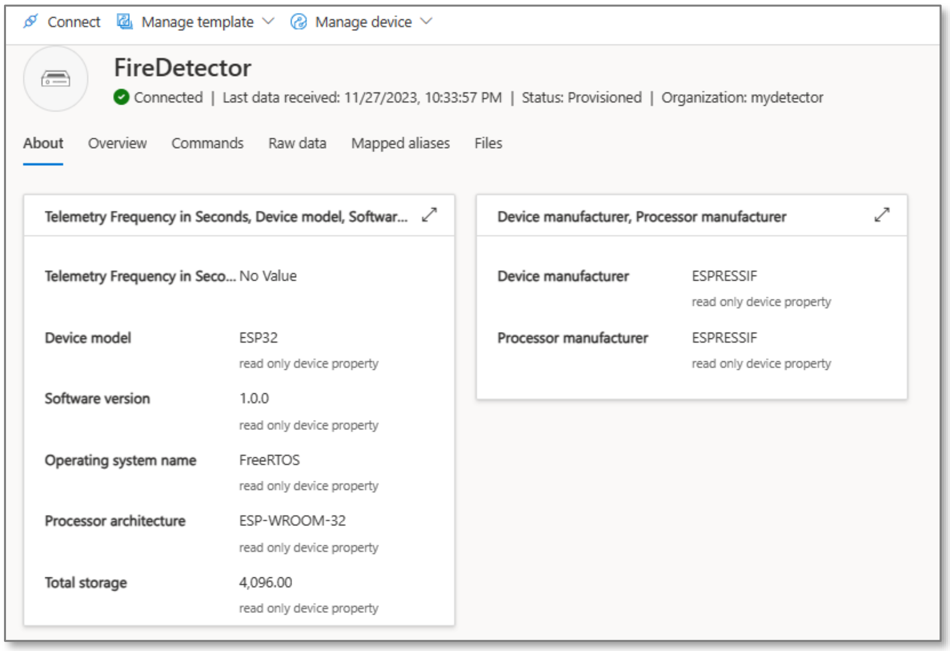
FireDetector

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

19. View Device Information



Connect | Manage template | Manage device

FireDetector

Connected | Last data received: 11/27/2023, 10:33:57 PM | Status: Provisioned | Organization: mydetector

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

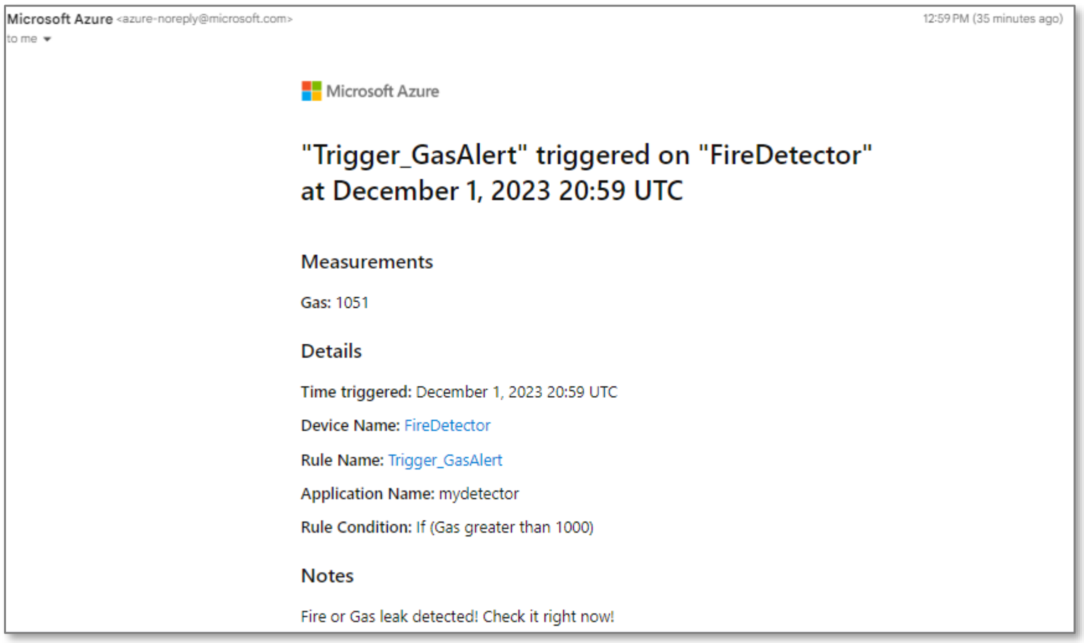
Telemetry Frequency in Seconds, Device model, Softwar...

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

20. Check email alert



Microsoft Azure <azure-noreply@microsoft.com>
to me
12:59 PM (35 minutes ago)

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

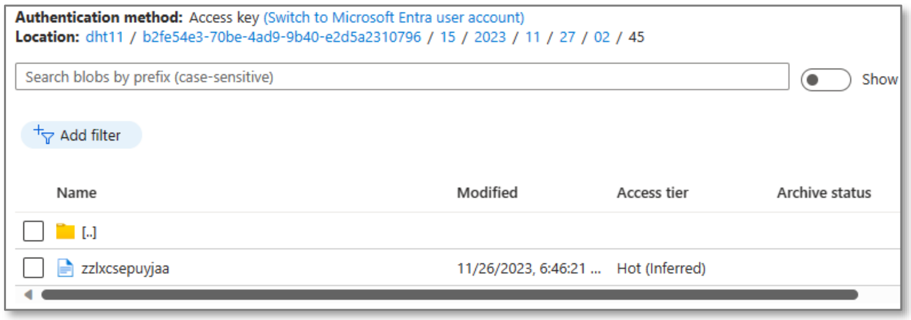
Application Name: mydetector

Rule Condition: If (Gas greater than 1000)

Notes

Fire or Gas leak detected! Check it right now!

21. Check Blob Store



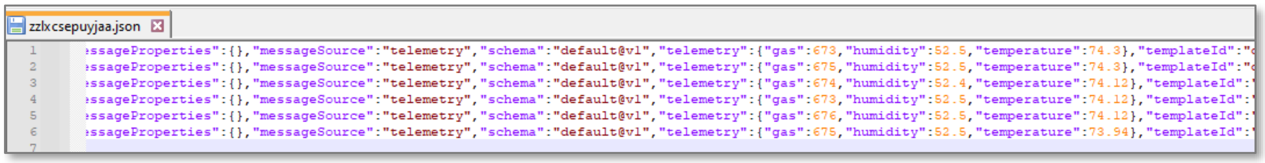
Authentication method: Access key (Switch to Microsoft Entra user account)
Location: dht11 / b2fe54e3-70be-4ad9-9b40-e2d5a2310796 / 15 / 2023 / 11 / 27 / 02 / 45

Search blobs by prefix (case-sensitive)

Show

Add filter

Name	Modified	Access tier	Archive status
[..]			
zzlxcsepuyjaa	11/26/2023, 6:46:21 ...	Hot (Inferred)	



zzlxcsepuyjaa.json

```
ssageProperties":{},"messageSource":"telemetry","schema":"default@v1","telemetry":{"gas":673,"humidity":52.5,"temperature":74.3},"templateId":"
ssageProperties":{},"messageSource":"telemetry","schema":"default@v1","telemetry":{"gas":675,"humidity":52.5,"temperature":74.3},"templateId":"
ssageProperties":{},"messageSource":"telemetry","schema":"default@v1","telemetry":{"gas":674,"humidity":52.4,"temperature":74.12},"templateId":"
ssageProperties":{},"messageSource":"telemetry","schema":"default@v1","telemetry":{"gas":673,"humidity":52.5,"temperature":74.12},"templateId":"
ssageProperties":{},"messageSource":"telemetry","schema":"default@v1","telemetry":{"gas":676,"humidity":52.5,"temperature":74.12},"templateId":"
ssageProperties":{},"messageSource":"telemetry","schema":"default@v1","telemetry":{"gas":675,"humidity":52.5,"temperature":73.94},"templateId":"
```