**IoT Security and Privacy**
**Term Project Phase 1 – Environmental Monitoring with AWS IoT + ESP32 + DHT11 + GPS (NEO-6M) + PMSA003**
**By:- Anmol Sureshkumar Panchal ; UID:4446829**

1. Document in detail the procedures updating the state of the connected DHT11, NEO-6M and PMSA003 continuously with the AWS IoT thing shadow. The data should be written into Amazon DynamoDB. The data should be in the format of (date and time from NEO-6M, geolocation from NEO-6M, DHT11 data, PMSA003 data). That is, explain how to code and send sensor data to Amazon AWS IoT. (9 points)

**Ans:**
The solution to this is same as Assignment 6-AWS_IoT, where we showed the setup and configuration of AWS with MQTT client shadow update and inserting data into Dynamodb table. So it's all the same for this assignment as well.
We have just created a new rule in Iot Core in AWS to insert new data which has a payload as json string comprising readings from Thermister, Neo6M-ublox GPS sensor, PMSA sensor and the current timestamp.

```
select * from 'Vira'
```

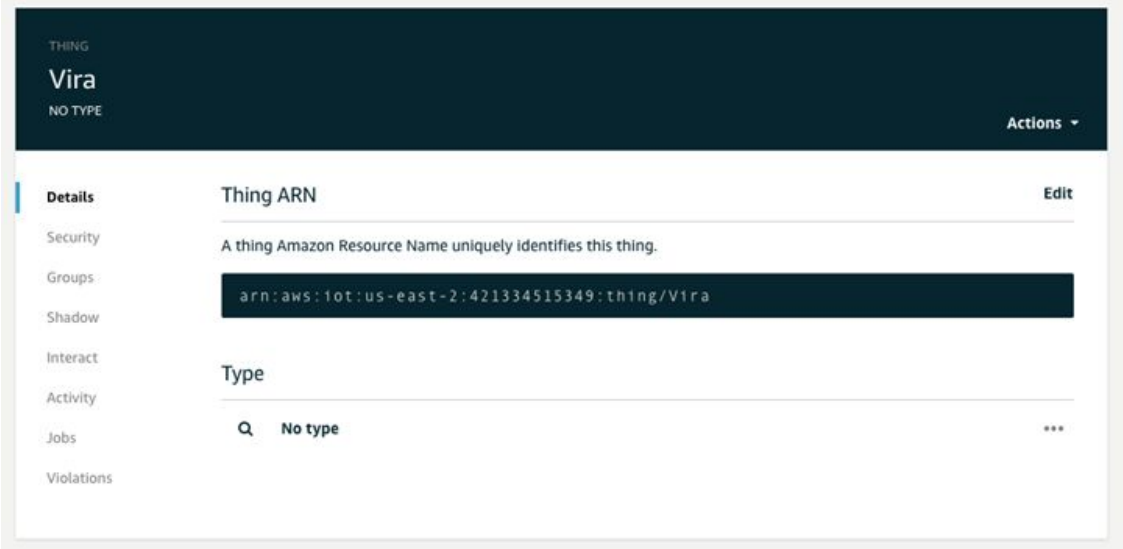
**Using SQL version 2016-03-23**

## Actions

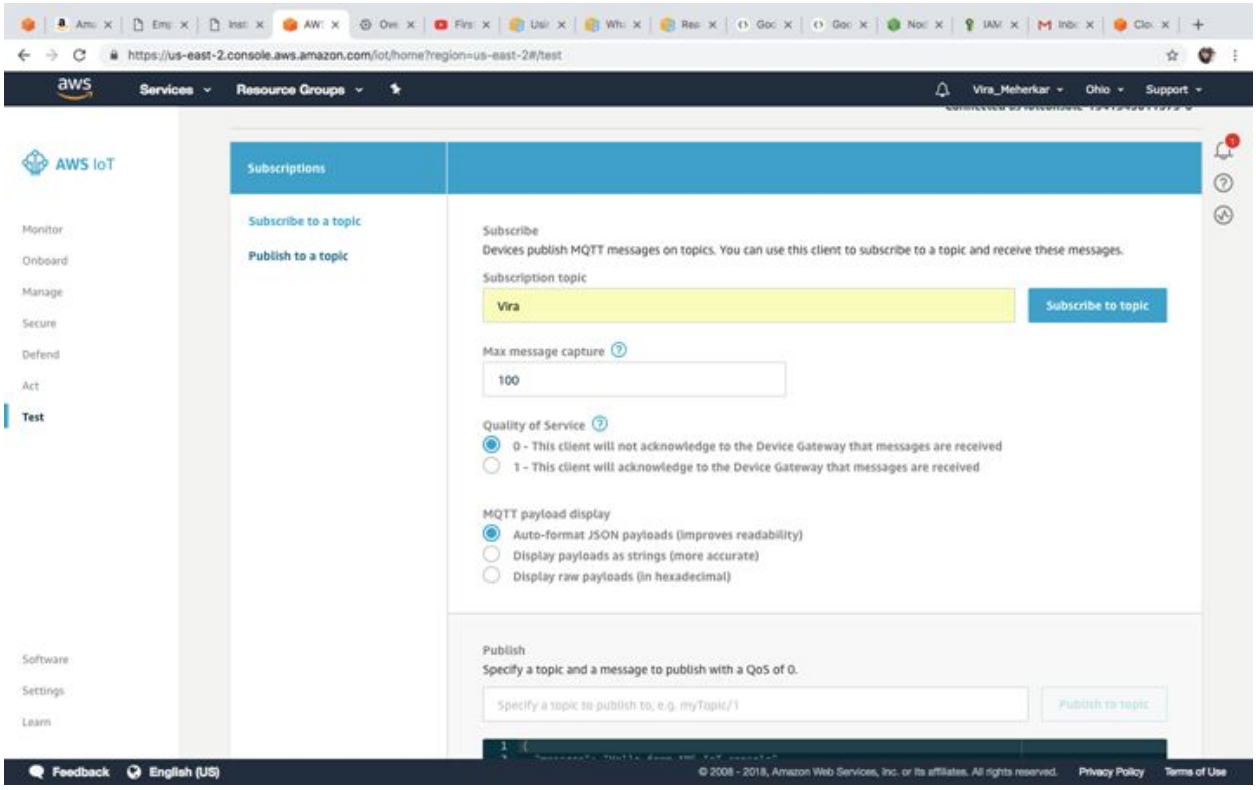
Actions are what happens when a rule is triggered. **Learn more**


**Add action**

In the ESP32 code, we use the AWS IOT library to connect to the AWS, which uses the HTTPS AWS endpoint to connect to the IOT



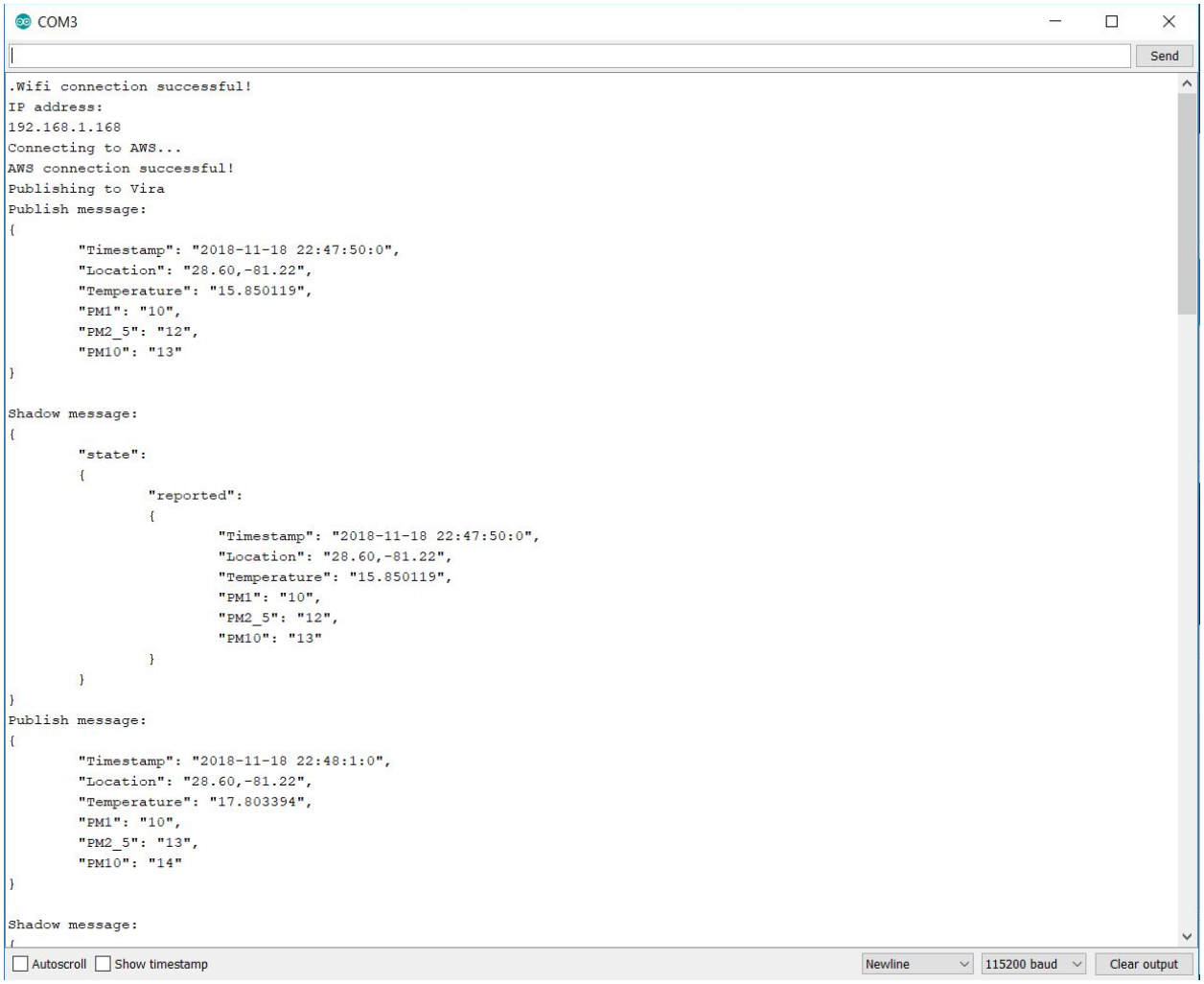
For the Publish and Subscribe it uses the MQTT shadow updates, also we should JSON format to ensure the data read and insertion in the Dynamo DB table, that we have created.



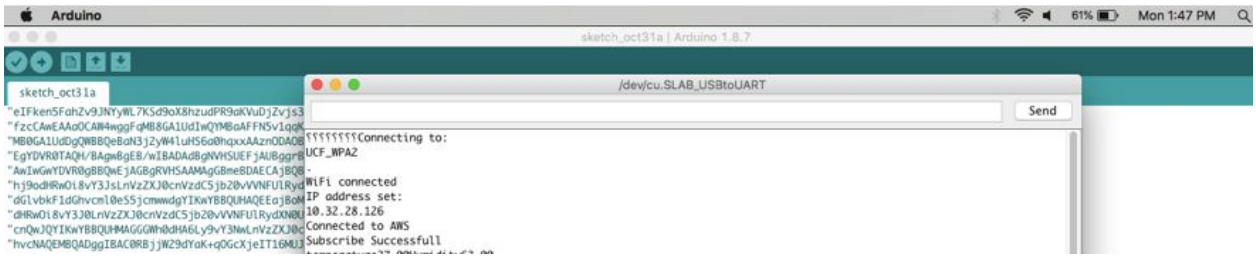

Also we need to update the certificates and key in a ‘C’ file to ensure connection to the IOT.

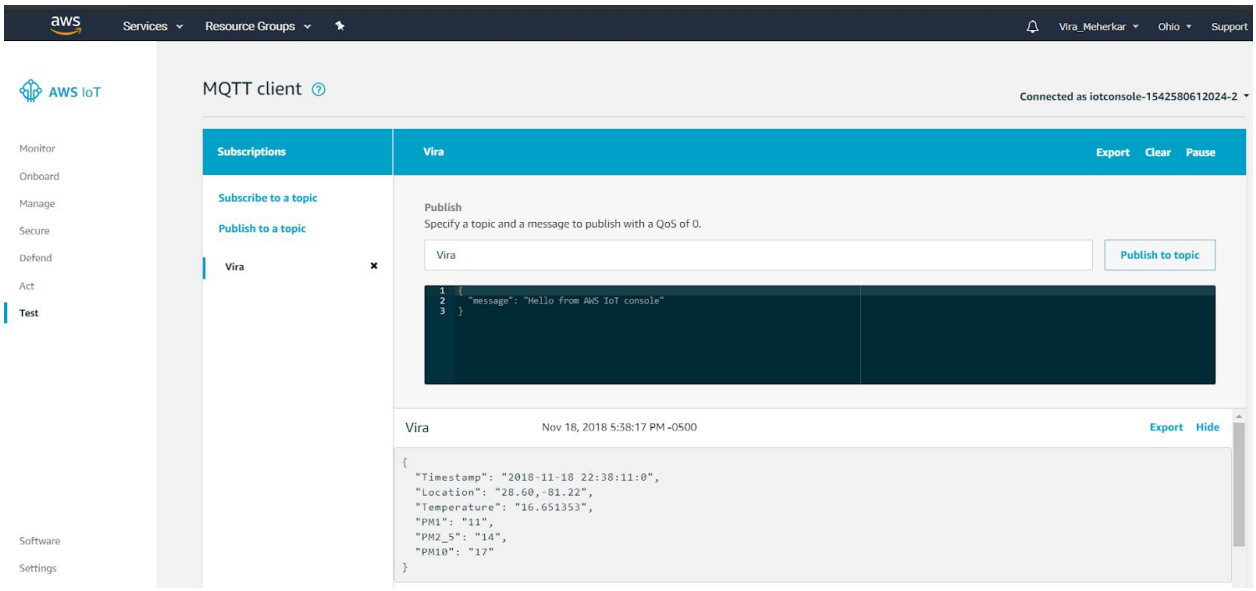

After you compiling and uploading the code my Arduino Console shows some output like this:

```
Writing at 0x0001c000... (13 %)
Writing at 0x00020000... (16 %)
Writing at 0x00024000... (20 %)
Writing at 0x00028000... (23 %)
Writing at 0x0002c000... (26 %)
Writing at 0x00030000... (30 %)
Writing at 0x00034000... (33 %)
Writing at 0x00038000... (36 %)
Writing at 0x0003c000... (40 %)
Writing at 0x00040000... (43 %)
Writing at 0x00044000... (46 %)
Writing at 0x00048000... (50 %)
Writing at 0x0004c000... (53 %)
Writing at 0x00050000... (56 %)
Writing at 0x00054000... (60 %)
Writing at 0x00058000... (63 %)
Writing at 0x0005c000... (66 %)
Writing at 0x00060000... (70 %)
Writing at 0x00064000... (73 %)
Writing at 0x00068000... (76 %)
Writing at 0x0006c000... (80 %)
Writing at 0x00070000... (83 %)
Writing at 0x00074000... (86 %)
Writing at 0x00078000... (90 %)
Writing at 0x0007c000... (93 %)
Writing at 0x00080000... (96 %)
Writing at 0x00084000... (100 %)
Wrote 802304 bytes (485548 compressed) at 0x00010000 in 43.4 seconds (effective 147.8 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 144...

Writing at 0x00008000... (100 %)
Wrote 3072 bytes (144 compressed) at 0x00008000 in 0.0 seconds (effective 877.7 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

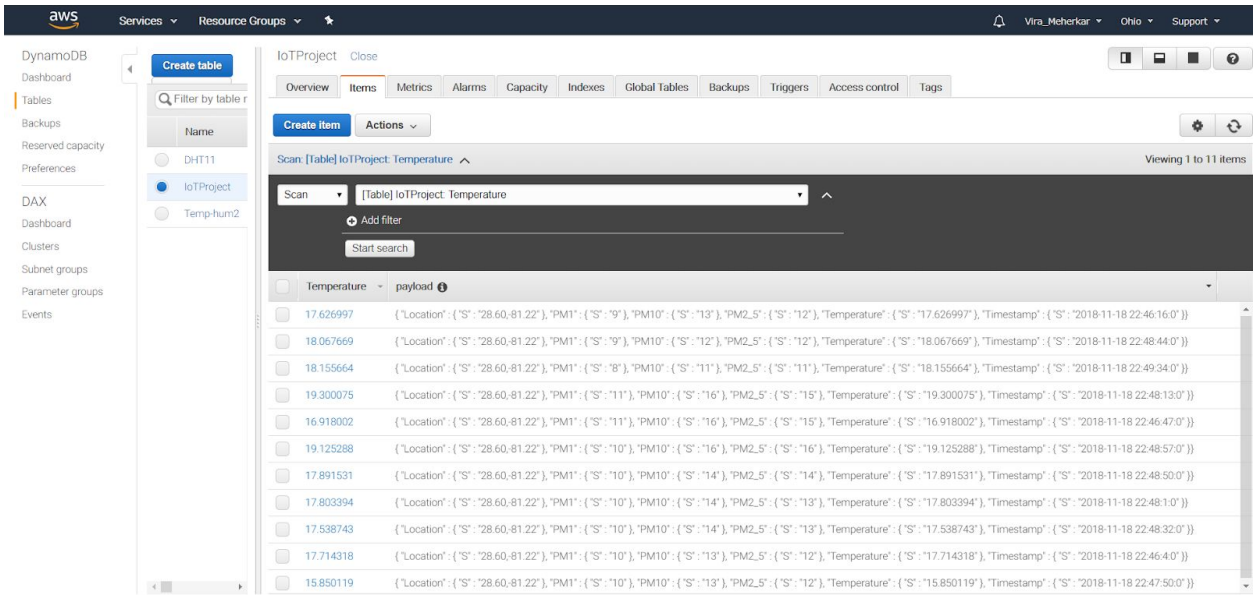



From the above screen shot we can see that the Values are same in Dynamodb as the Value from the ESP32 DHT sensor read on COM3 Serial Monitor.
When You connect to Eduroam or UCF_WPA2 you will see the following confirmation on Serial Monitor with the same readings.

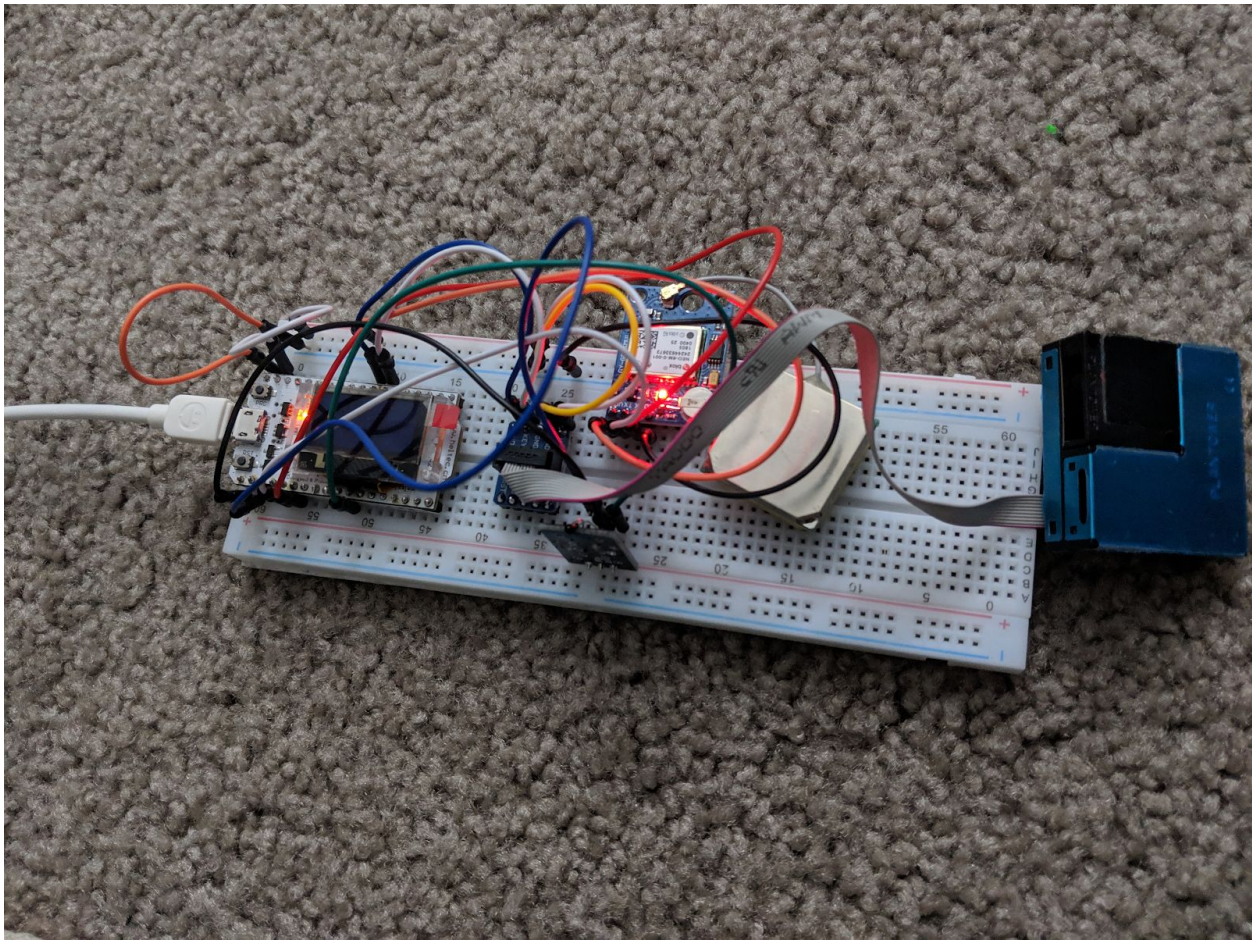





Once the data is seen from Subscribed IoT thing name , We have to check whether our data is correctly being stored in DynamoDb. And as you can see we have successfully done that part as well as illustrated by screenshot below:



2. Post a photo of the complete device below. (2 points)



3. Copy and paste the code for this assignment into this report. (2 points)

When executing on eduroam or UCF_WPA2 change debugging mode to TRUE and uncomment the credentials required for eduroam from the code.

```
const bool debugging = TRUE
//#define EAP_ID "UCF_WPA2"
//#define EAP_USERNAME "an370677"
//#define EAP_PASSWORD "********"
```

**CODE:-**

```
#include <math.h>
#include "esp_wpa2.h"
#include <WiFi.h>
#include <AWS_IOT.h>
#include <ctime>
#include <TinyGPS++.h>
#include <HardwareSerial.h>
#include <PMsensor.h>

// Change to false to connect to AWS
const bool debugging = false;

// wifi config
const char* ssid = "Vikings";
const char* pwd = "RagnarLothbrok";
//#define EAP_ID "UCF_WPA2"
//#define EAP_USERNAME "an370677"
//#define EAP_PASSWORD "********"

//########################//
//AWS Credentials//
//########################//
char HOST_ADDRESS[]="a2txpxlqpf0l2a-ats.iot.us-east-2.amazonaws.com";
char CLIENT_ID[]= "ViraPolicy";
char TOPIC_NAME[]= "Vira";
// GPS config
static const int gpsRx = 13, gpsTx = 12;
static const uint32_t gpsBaud = 9600;
TinyGPSPlus gps;
HardwareSerial ss(1);

#define ESP32
```

```
#ifdef ESP32
// PMSerial config
SerialPM pms(PMSA003);
HardwareSerial SerialAnmol(2);
#endif
// MQTT message config
int status = WL_IDLE_STATUS;
int tick = 0, msgCount = 0, msgReceived = 0;
char payload[512], shadowPayload[512];

//############################//
// UCF SERVER CERTIFICATE  //
//############################//
const char* UCF_server_ca_cert = "-----BEGIN CERTIFICATE-----\n"
"MIIF+TCCA+GgAwIBAgIQRyDQ+oVGGn4XoWQCkYRjdDANBgkqhkiG9w0BAQwFADCB\n"
"iDELMAkGA1UEBhMCVVMxEzARBgNVBAgTCk5ldyBKZXJzZXkxFDASBgNVBAcTC0pl\n"
"cnNleSBDaXR5MR4wHAYDVQQKExVUaGUgVVNFUlRSVVNUIE5ldHdvcmsxLjAsBgNV\n"
"BAMTJVVTRVJUcnVzdCBSU0EgQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkwHhcNMTQx\n"
"MDA2MDAwMDAwWhcNMjQxMDA1MjM1OTU5WjB2MQswCQYDVQQGEwJVUzELMAkGA1UE\n"
"CBMCTUkxEjAQBgNVBAcTCUFubiBBcmJvcjESMBAGA1UEChMJSW50ZXJuZXQyMREw\n"
"DwYDVQQLEwhJbkNvbW1vbjEfMB0GA1UEAxMWSW5Db21tb24gUlNBIFNlcnZlciBD\n"
"QTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAJwb8bsvf2MYFVFRVA+e\n"
"xU5NEFj6MJsXKZDmMwysE1N8VJG06thum4ltuzM+j9INpun5uukNDBqeso7JcC7v\n"
"HgV9lestjaKpTbOc5/MZNrun8XzmCB5hJ0R6lvSoNNviQsil2zfVtefkQnI/tBPP\n"
"iwckRR6MkYNGuQmm/BijBgLsNI0yZpUn6uGX6Ns1oytW61fo8BBZ321wDGZq0GTl\n"
"qKOYMa0dYtX6kuOaQ80tNfvZnjNbRX3EhigsZhLI2w8ZMA0/6fDqSl5AB8f2IHpT\n"
"eIFken5FahZv9JNYyWL7KSd9oX8hzudPR9aKVuDjZvjs3YncJowZaDuNi+L7RyML\n"
"fzcCAwEAAaOCAW4wggFqMB8GA1UdIwQYMBaAFFN5v1qqK0rPVIDh2JvAnfKyA2bL\n"
"MB0GA1UdDgQWBBQeBaN3j2yW4luHS6a0hqxxAAznODAOBgNVHQ8BAf8EBAMCAYYw\n"
"EgYDVR0TAQH/BAgwBgEB/wIBADAdBgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUH\n"
"AwIwGwYDVR0gBBQwEjAGBgRVHSAAMAgGBmeBDAECAjBQBgNVHR8ESTBHMEWgQ6BB\n"
"hj9odHRwOi8vY3JsLnVzZXJ0cnVzdC5jb20vVVNFUlRydXN0UlNBQ2VydGlmaWNh\n"
"dGlvbkF1dGhvcml0eS5jcmwwdgYIKwYBBQUHAQEEajBoMD8GCCsGAQUFBzAChjNo\n"
"dHRwOi8vY3J0LnVzZXJ0cnVzdC5jb20vVVNFUlRydXN0UlNBQWRkVHJ1c3RDQS5j\n"
"cnQwJQYIKwYBBQUHMAGGGWh0dHA6Ly9vY3NwLnVzZXJ0cnVzdC5jb20wDQYJKoZI\n"
"hvcNAQEMBQADggIBAC0RBjjW29dYaK+qOGcXjeIT16MUJNkGE+vrkS/fT2ctyNMU\n"
"11ZlUp5uH5gIjppIG8GLWZqjV5vbhvhZQPwZsHURKsISNrqOcooGTie3jVgU0W+0\n"
"+Wj8mN2knCVANt69F2YrA394gbGAdJ5fOrQmL2pIhDY0jqco74fzYefbZ/VS29fR\n"
"5jBxu4uj1P+5ZImem4Gbj1e4ZEzVBhmO55GFfBjRidj26h1oFBHZ7heDH1Bjzw72\n"
"hipu47Gkyfr2NEx3KoCGMLCj3Btx7ASn5Ji8FoU+hCazwOU1VX55mKPU1I2250Lo\n"
"RCASN18JyfsD5PVldJbtyrmz9gn/TKbRXTr80U2q5JhyvjhLf4lOJo/UzL5WCXED\n"
"Smyj4jWG3R7Z8TED9xNNCxGBMXnMete+3PvzdhssvbORDwBZByogQ9xL2LUZFI/i\n"
"eoQp0UM/L8zfP527vWjEzuDN5xwxMnhi+vCToh7J159o5ah29mP+aJnvujbXEnGa\n"
"nrNxHzu+AGOePV8hwrGGG7hOIcPDQwkuYwzN/xT29iLp/cqf9ZhEtkGcQcIImH3b\n"
"oJ8ifsCnSbu0GB9L06Yqh7IcyvKDTEADsIIaeSEINxhO2Y1fmcYFX/Fqrrp1WnhH\n"
"OjpIXuXE0OPa0utaKC25Aplgom88L2Z8mEWcyfoB7zKOfD759AN7JKZWCYwk\n"
"-----END CERTIFICATE-----\n";
```

```
/*
 * Convert thermal reading to accurate celcius value
 */
double Thermister(int RawADC)
{
  double Temp;
  Temp = log(((10240000/RawADC) - 10000));
  Temp = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * Temp * Temp ))* Temp );
  Temp = Temp - 270.15;
  return Temp;
}

/*
 * Connect to Wifi;
 */
void connectToWifi()
{
  WiFi.disconnect(true);
  WiFi.mode(WIFI_STA);

  //esp_wifi_sta_wpa2_ent_set_identity((uint8_t*) EAP_ID, strlen(EAP_ID));
  //esp_wifi_sta_wpa2_ent_set_username((uint8_t*) EAP_USERNAME, strlen(EAP_USERNAME));
  //esp_wifi_sta_wpa2_ent_set_password((uint8_t*) EAP_PASSWORD, strlen(EAP_PASSWORD));
  //esp_wifi_sta_wpa2_ent_set_ca_cert((const unsigned char*) ucf_certificate, strlen(ucf_certificate));

  //esp_wpa2_config_t conf = WPA2_CONFIG_INIT_DEFAULT();
  //esp_wifi_sta_wpa2_ent_enable(&conf);
  WiFi.begin(ssid,pwd);
  while(WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
}

/*
 * Connect to AWS
 */
void connectToAWS()
{
  if(aws_iot.connect(HOST_ADDRESS,CLIENT_ID)== 0) delay(1000);
  else
  {
    Serial.println("AWS connection failed, Check the HOST Address");
    while(1);
```

```
  }
}

/*
 * Program starts here
 */
void setup()
{
  Serial.begin(115200);


  Serial.print("Connecting to ");
  Serial.println(ssid);

  // Start by connecting to wifi
  connectToWifi();
  Serial.println("Wifi connection successful!");
  Serial.println("IP address:");
  Serial.println(WiFi.localIP());

   pms.begin(SerialAnmol);
  pms.init();

  ss.begin(gpsBaud, SERIAL_8N1, gpsRx, gpsTx);

  /*
   * In debug mode, we do not connect to AWS.
   */
  if(!debugging)
  {
    delay(5000);

    // Connect to AWS
    Serial.println("Connecting to AWS...");
    connectToAWS();
    Serial.println("AWS connection successful!");

    // Start MQTT
    Serial.print("Publishing to ");
    Serial.println(TOPIC_NAME);
  }
}

/*
 *Handle reading of sensors and publishing to MQTT topic
 */
void loop()
{
```

```
 delay(1000);
 int pm1, pm2_5, pm10;

 double lat, lng;
 uint16_t year, month, day, hour, minute, second, csecond;
 String timestamp = "";
 String loc = "";
 char locArray[512],timestampArray[512]
 ;

  // Read air quality
 pms.read();
 pm1 = pms.pm[0];
 pm2_5 = pms.pm[1];
 pm10 = pms.pm[2];
  //Read temperature
  double temperature = Thermister(analogRead(33));
 // Read GPS location, date, and time
 while(ss.available() > 0)
 {
   timestamp = "";
   gps.encode(ss.read());
   if(gps.location.isUpdated())
   {
     lat = gps.location.lat();
     lng = gps.location.lng();
     //timestamp = lat + lng;
     loc = String(lat) + String(",") + String(lng);
   }
     //loc = " 28.59, -81.22";
   if(gps.date.isValid())
   {
     year = gps.date.year();
     month = gps.date.month();
     day = gps.date.day();
     timestamp = String(year) + String("-") + String(month) + String("-") + String(day);
   }
   if(gps.time.isValid())
   {
     hour = gps.time.hour();
     minute = gps.time.minute();
     second = gps.time.second();
     csecond = gps.time.centisecond();
     timestamp+= String (" ") +String(hour) +  String(":") +String(minute) + String(":") +
String(second)+String(":")+String(csecond);
   }
   //delay(5000);
 }
```

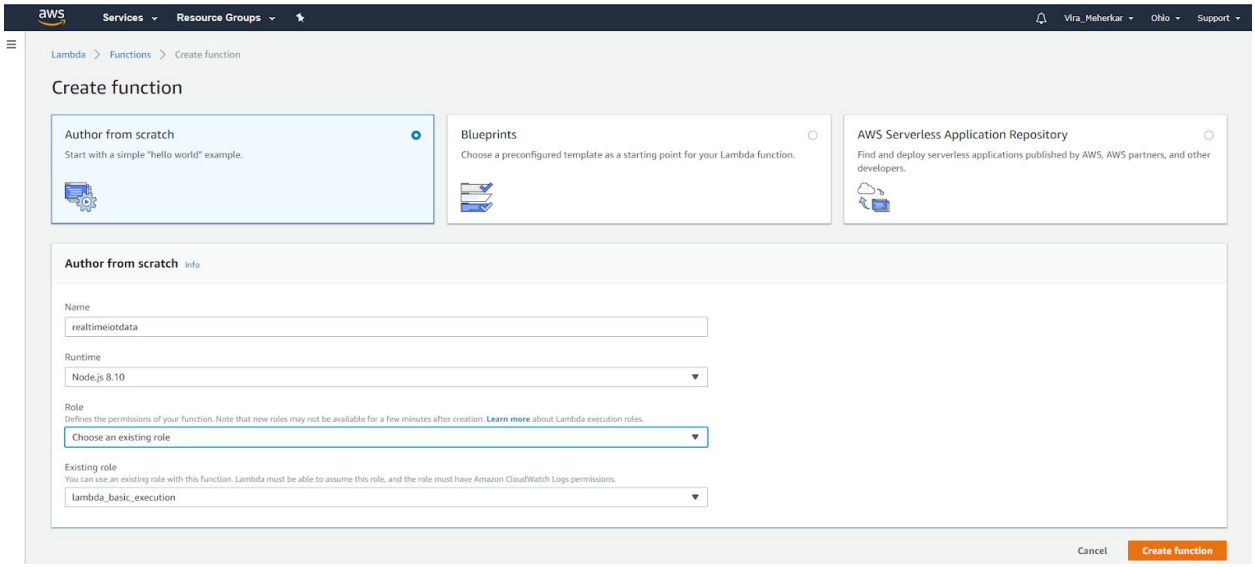
```
  if(debugging)
  {
    Serial.print("PMS 1: ");
    Serial.println(pm1);
    Serial.print("PMS 2.5: ");
    Serial.println(pm2_5);
    Serial.print("PMS 10: ");
    Serial.println(pm10);

    Serial.print("Temperature: ");
    Serial.println(temperature);
    Serial.print("Location:");
    Serial.println(loc);
    Serial.print("Timestamp: ");
    Serial.println(timestamp);
    //return;
  }
  timestamp.toCharArray(timestampArray, 512);
  loc.toCharArray(locArray, 512);
  //sprintf(payload, "{\n\t\"Timestamp\": \"%s\"\n\t\"Temperature\": \"%f\"\n\t\"Air quality data\": \"%d\"\n}",
timestamp,temperature,pm1);
  sprintf(payload, "{\n\t\"Timestamp\": \"%s\",\n\t\"Location\": \"%s\",\n\t\"Temperature\": \"%f\",\n\t\"PM1\":
\"%d\",\n\t\"PM2_5\": \"%d\",\n\t\"PM10\": \"%d\"\n}", timestampArray, locArray, temperature, pm1, pm2_5,
pm10);
  //sprintf(shadowPayload, "{\n\t\"state\": {\n\t\t\"reported\": {\n\t\t\t\"Timestamp\" :
\"%f\"\n\t\t\n\t\t\t\"Temperature\" : \"%f\"\n\t\t\n\t\t\t\"Air quality data\" : \"%d\"\n\t\t}\n\t}\n}",
timestamp,temperature,pm1);
  sprintf(shadowPayload, "{\n\t\"state\":\n\t{\n\t\t\"reported\":\n\t\t{\n\t\t\t\"Timestamp\":
\"%s\",\n\t\t\t\"Location\": \"%s\",\n\t\t\t\"Temperature\": \"%f\",\n\t\t\t\"PM1\": \"%d\",\n\t\t\t\"PM2_5\":
\"%d\",\n\t\t\t\"PM10\": \"%d\"\n\t\t}\n\t}\n}", timestampArray, locArray, temperature, pm1, pm2_5, pm10);
  // Publish to main topic
  if(aws_iot.publish(TOPIC_NAME, payload) == 0)
  {
    Serial.println("Publish message: ");
    Serial.println(payload);
  }
  else Serial.println("Publish failed");

  // Publish to shadow
  if(aws_iot.publish(SHADOW_TOPIC_NAME, shadowPayload) == 0)
  {
    Serial.println("\nShadow message: ");
    Serial.println(shadowPayload);
  }
  else Serial.println("Shadow publish failed");

  delay(5000);
}
```

4. Attach the code to WebCourse for this term project Phase One. (2 points)
Attached IoT Phase_1_Code.txt file

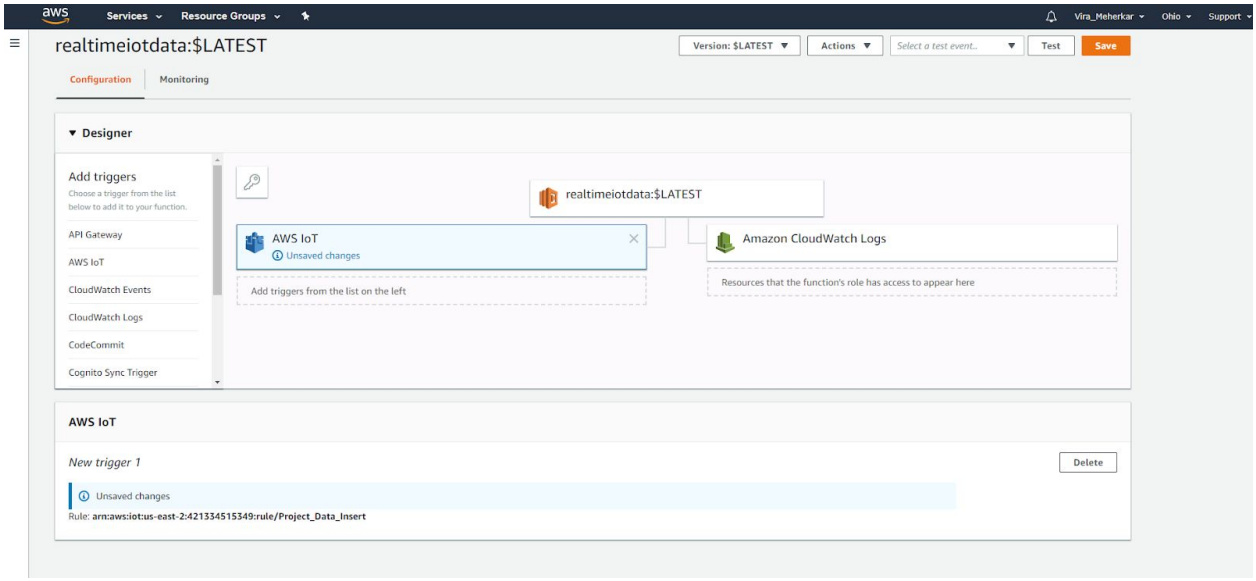
5. Bonus points: any application (web, smartphone app, etc) of displaying the real-time data will be awarded another 25 bonus points toward the final grade!Note: the display application must be proved secure. If the instructor can identify the security vulnerabilities of the application, there will be no bonus point at all. One example of such a vulnerability is an attacker can inject data into the database, or the attacker can change the data that is sent to the database.

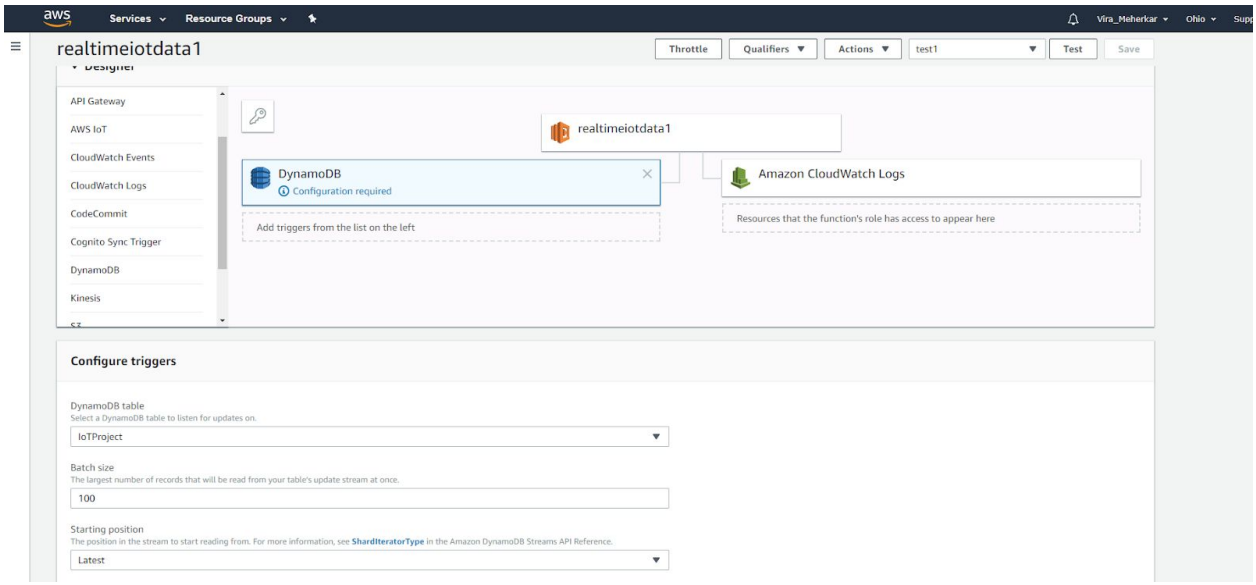
Solution:
I am trying to use lambda service from AWS to scan and read the items from the Dynamodb table (IoTProject) and after than setting up index.js an front end application (web page) which will be used to show us the real time data from our table. We will use a Google Map API key which is showing us these data on google maps in real time. You can follow detail steps to execute this part from the links given in the references.

First write a function to fetch or scan data from our Dynamodb table a shown below:

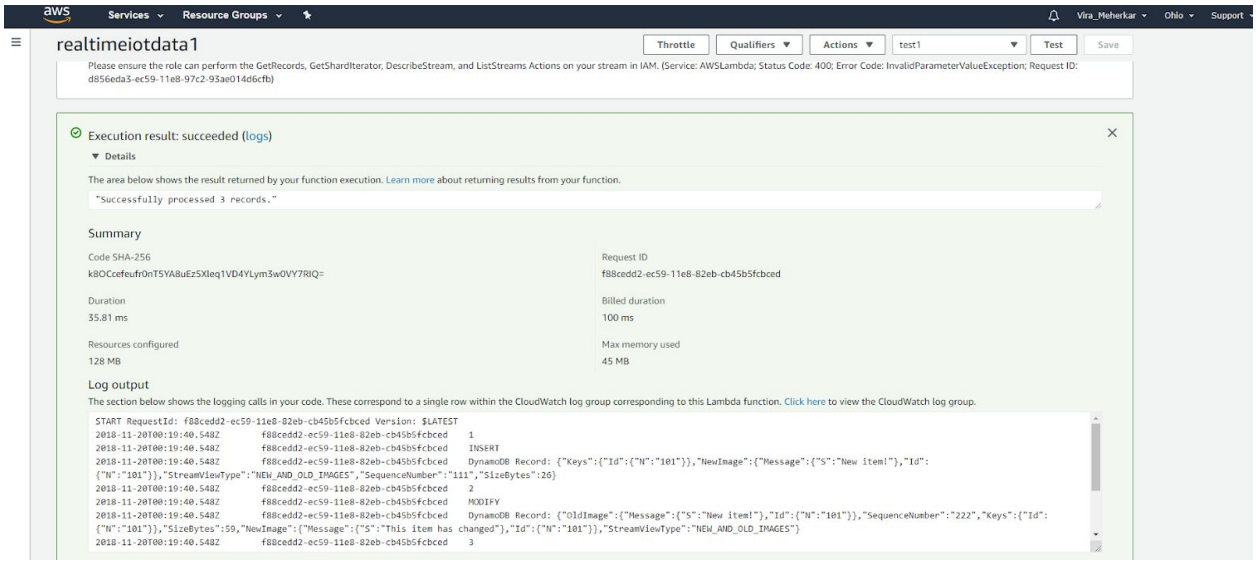


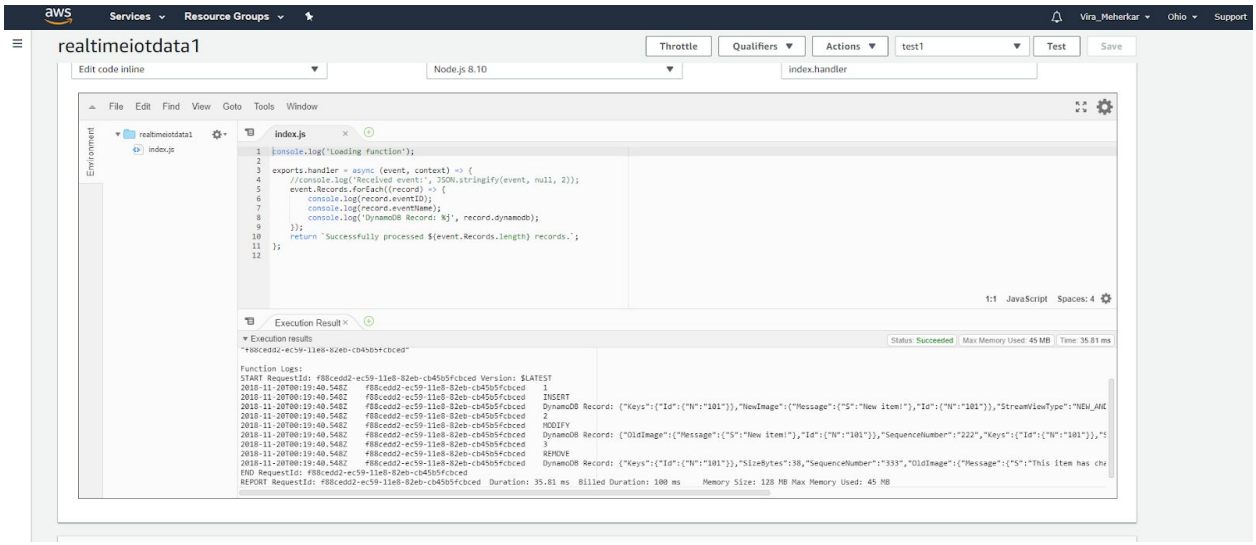

Connect your AWS IoT core to process the stream in real time from shadow update.



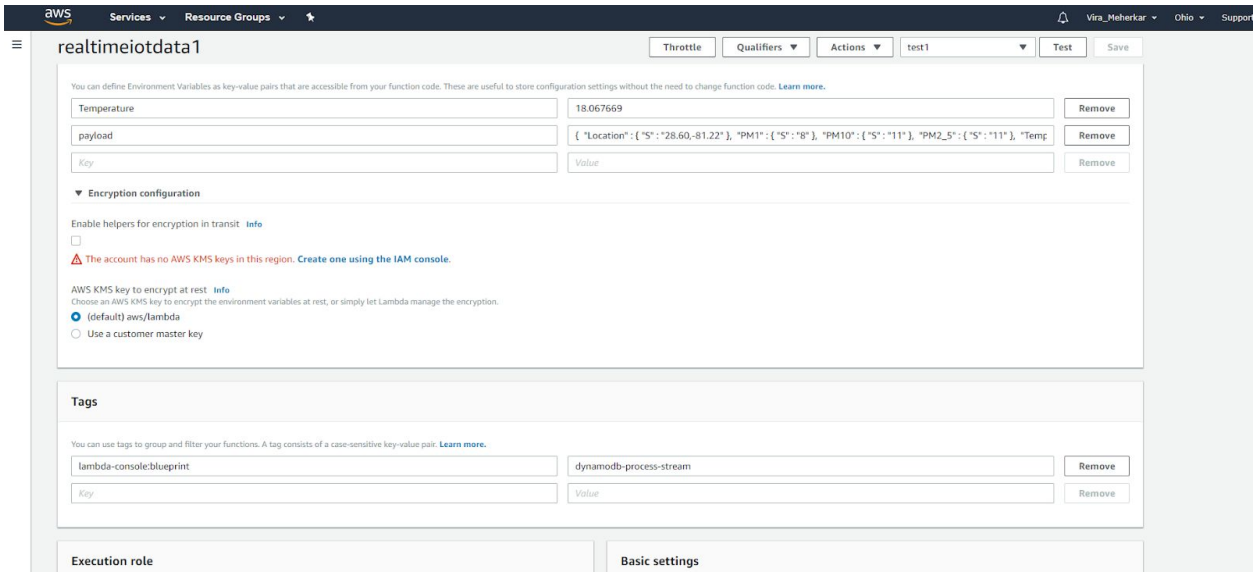




On successfully creation of the Lambda service function test it and save it. You can see the successful execution of the function as shown below:
This means that records are successfully scanned from our table.

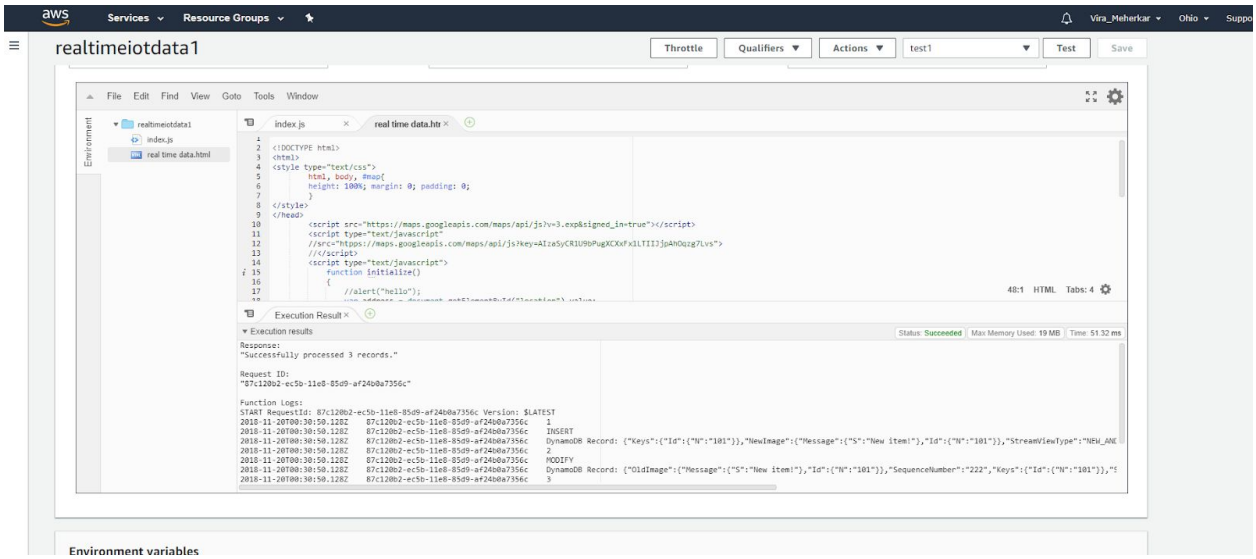



The variables are given as follows:




You then create a html file to view at the front end to show current location which is (28.56,-81.22).

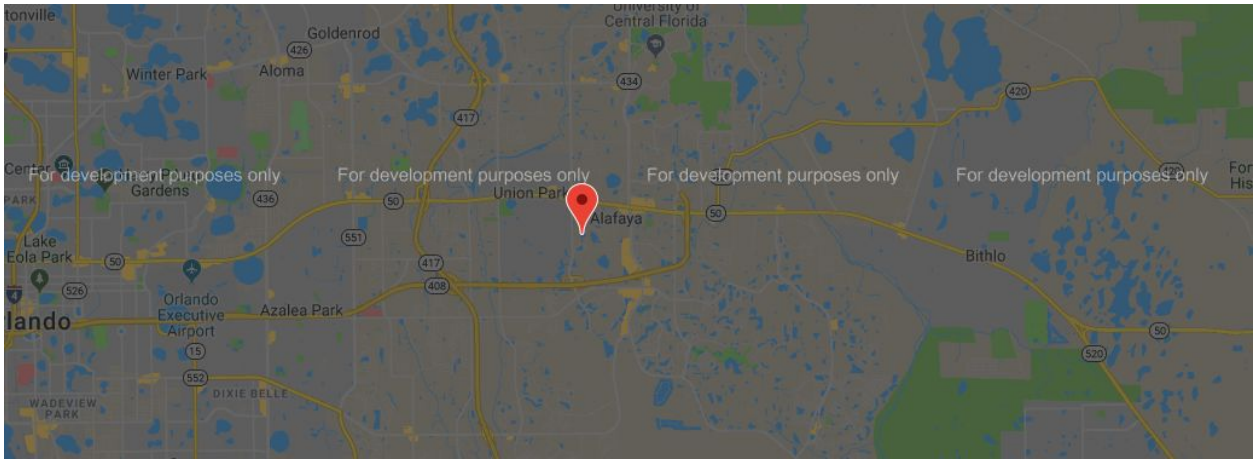



The output is somewhat like this. It shows " For development purpose only" because the api key is fetched from google maps and they automatically display like that.

Due to limited space we could'nt run our esp32 for a long time as we had to delete manually when records were going out of memory capacity as lambda service was also taking space, so in that scenario we had to manually delete the data from table and run again to show this output below:

Before this I tried from exporting data from Dynamodb manually to localhost XAMPP server and then displaying location on web page using php but the problem was with security and also it was not real time tracking.
So i searched this method on google and implemented it step by step from the links i referenced in later part of this report.

Don't worry about the safety of the front end web page as amazon handles the encryption part on itself so that your data is securely streamed from Dynamodb table to Lambda service function and show the output as desired.



References:

***https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html***

***https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/dynamodb-example-query-scan.html***

***https://thewebspark.com/2018/07/04/how-to-get-and-post-data-from-dynamodb-table-using-node-js-and-aws-lambdaserverless-function/***

***https://stackoverflow.com/questions/43241051/how-to-query-dynamodb-table-by-name-using-aws-lambda***

***https://www.youtube.com/watch?v=P8okmPWIAcQ***

***https://www.youtube.com/watch?v=mfAT38B_uhw&t=311s***