

University of Mississippi

eGrove

Honors Theses

Honors College (Sally McDonnell Barksdale
Honors College)

5-9-2019

Creating an Internet of Things Platform for Storing Smart Sensor Data Using Amazon Web Services

Jin Hyeok Noh
University of Mississippi

Follow this and additional works at: https://egrove.olemiss.edu/hon_thesis



Part of the [Management Information Systems Commons](#)

Recommended Citation

Noh, Jin Hyeok, "Creating an Internet of Things Platform for Storing Smart Sensor Data Using Amazon Web Services" (2019). *Honors Theses*. 1050.
https://egrove.olemiss.edu/hon_thesis/1050

This Undergraduate Thesis is brought to you for free and open access by the Honors College (Sally McDonnell Barksdale Honors College) at eGrove. It has been accepted for inclusion in Honors Theses by an authorized administrator of eGrove. For more information, please contact egrove@olemiss.edu.

CREATING AN INTERNET OF THINGS PLATFORM FOR
STORING SMART SENSOR DATA USING
AMAZON WEB SERVICES

A thesis submitted to the faculty of The University of Mississippi in partial fulfillment of
the requirements of the Sally McDonnell Barksdale Honors College.

by
Jin Hyeok Noh

Oxford
May 2019

Approved by

Advisor: Professor Brian J Reithel

Reader: Professor Bart L Garner

Reader: Professor Tony Ammeter

© 2019
Jin Hyeok Noh
ALL RIGHTS RESERVED

DEDICATION
For Yeon Seob Noh, Kyung Lan Moon

ACKNOWLEDGEMENTS

I want to thank Dr. Brian Reithel for his inexpressible guidance during the work on this thesis. In addition, I want to thank Dr. Bart Garner and Dr. Tony Ammeter for being readers of this thesis. In addition, I want to thank my parents for supporting the decision I chose. In addition, I want to thank my friends Khoa Tran and Amber Izzard for inspiring this thesis.

ABSTRACT

Creating an Internet of Things Platform for Storing Smart Sensor Data using
Amazon Web Services

by Jin Hyeok Noh

(Under the direction of Professor Brian J Reithel)

Water is one of the essential resources that we should be aware of conserving, but lawn sprinkler systems in Mississippi have inefficient resource management systems. During my fundamental research, I read an article about IoT (Internet of Things) and how important it will be in the future. Both of these topics inspired prototype research.

In order to create the prototype, a Raspberry Pi, breadboard, keyboard, monitor, and cables were used. Using the Raspberry Pi, monitors, and keyboard, the software was developed in order to create a RESTFUL API gateway to connect to AWS (Amazon Web Service). In the AWS console, an AWS Lambda function was created to store data to AWS DynamoDB. Then, a breadboard and sensor were connected to the Raspberry Pi.

When the sensor detects water, the data is transferred to the Raspberry Pi, and the Raspberry Pi sends the data to the AWS API Gateway. The API Gateway then sends the data to AWS Lambda, and the process continues from AWS Lambda to AWS DynamoDB. DynamoDB can display the type of sensor, date, IP address, user id, and eventid. This platform will make it easy to monitor energy efficiency and water waste.

TABLE OF CONTENTS

LIST OF TABLES	7
LIST OF ABBREVIATIONS.....	8
INTRODUCTION	9
CHAPTER I: BRIEF OVERVIEW OF CHAPTERS.....	10
CHAPTER II: PREVIOUS RESEARCH	11
CHAPTER III: RESEARCH METHODOLOGY	15
CHAPTER IV: DISCUSSION OF RESULTS	16
CHAPTER V: CONCLUSION.....	22
CHAPTER VI: DIRECTION FOR FUTURE THESIS.....	23

LIST OF FIGURES

- Figure 1 Picture of prototype
- Figure 2 Raspberry Pi code setup with Python
- Figure 3 Raspberry Pi GPIO Header block
- Figure 4 Raspberry Pi GPIO Pin diagram
- Figure 5 AWS API gateway interface setup to AWS Lambda
- Figure 6 AWS Lambda function code to send data to AWS DynamoDB to store data
- Figure 7 The result of AWS DynamoDB

LIST OF ABBREVIATIONS

- AWS Amazon Web Service
- IoT Internet Of Things
- IaaS Infrastructure as service
- PaaS Platform as service
- SaaS Software as service
- UI User Interface
- IP Internet Protocol
- RESTFUL Representational state transfer
- API Application program interface
- OS Operating System

INTRODUCTION

The Internet of Things (IoT) is a form of technology that produces data then interacts, exchanges, and connects to the Internet and stores data to a database. For example, a smartwatch is a favorite IoT device these days. It produces medical information such as heartbeats and contains a step tracker for users to check their health status. IoT devices must be connected to the Internet with a distinguishable IP address. The IoT has become more sophisticated and technologically advanced, the computer.

With the growth of IoT devices, our lives are often more connected to data. We have the capability now to check the weather, download photos, make phone calls, and more through our IoT smartwatches. However, people still think IoT devices are considered expensive equipment to upgrade their lives, even though through our daily lives there is much energy wasted due to automated systems. The cost for an IoT Sprinkler, for example, is more than \$200. While watching a lawn sprinkler system operate during a rainstorm, I found my inspiration to create a budget-friendly, water sensing device that connects with the IoT.

Can we create an affordable, functional prototype of an inexpensive IoT device that stores sensor data through a serverless cloud environment in order to observe the sensor's data? That is the research question that will be explored in this thesis.

Ch. 1 BRIEF OVERVIEW OF CHAPTERS

This chapter describes a brief overview of the chapters that make up this thesis. Chapter 2 explains the previous knowledge learned before conducting the research. Chapter 3 describes the prototype research methodology. First, it explains how the research question originated and describes the methodology used to proceed with research. Chapter 4 discusses, and the results and shows that the prototype is viable, as well as the lessons learned in this research. Chapter 5 is a summary of the research. It provides a guideline for future researchers, discusses how this research will be practical for society, and presents some limitations and assumptions of the research.

Ch. 2 PREVIOUS RESEARCH

As stated in the introduction, the IoT is a form of technology that allows people to connect to the Internet to exchange and store data. It analyzes the information that it receives and provides users the ability to access information remotely. The IoT includes a variety of embedded systems such as home appliances, mobile equipment, and wearables. The prospects for IoT are quite promising. Jown Woo Choi argued that according to the research organization Machina, “IoT growth rate will be at 21.8% per year until 2022, and total market share will be around \$1.2 trillion.” “In 2020, IoT products will cover 31.8% of electric devices, IT, and service areas. Also, the development of various platforms of IoT, data storage, and new software development is expected. In 2015, products patented by IoT were \$50 billion. The number is expected to reach \$ 250 billion by 2020,” estimated the research organization Gartner (as cited in KOTRA, 2016).

On the other hand, there are three crucial issues for IoT devices. First, there is no standardization because everything serves a different purpose, but this is a regulatory issue.

The second problem is the electronic waste (e-waste) issue. The IoT revolution needs a massive amount of new electronic parts. Old parts are often discarded, and this electronic waste will cause environmental problems. The third problem is that IoT is still insecure. Current IoT devices are vulnerable to hackers to attack through methods such as SQL injection attack.

For this thesis, a project was undertaken to create a prototype for an IoT device that connects a smart sensor to a Raspberry Pi. The Raspberry Pi is an example of an

inexpensive IoT device. It was created by the Raspberry Pi Foundation to enhance the availability of computer science in developing countries. Raspberry Pi runs on the Linux operating system. Python is one of the available computer languages that run inside a Raspberry Pi. Features of Python include the fact that it is easy to read and write computer programs compared to other computer languages. Also, Python has high compatibility with other programming languages.

Another resource used in this project is the Cloud platform, which is an approach that provides businesses with the components to build their cloud computing environments. It provides different services depending on the needs of the business. Cloud computing uses, “a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server or a personal computer” (Dillon-Roberts, 2018). The first advantage of Cloud computing is that it has scalability because the user does not own hardware or software. The second advantage is that it is particularly useful for start-up and small business since they do not have to invest in expensive infrastructure, hardware, or software.

There are different types of cloud computing services. For example, there is Software-as-a-service (SaaS), Platform-as-service (PaaS), and Infrastructure-as-service (IaaS). SaaS uses a web browser to deliver applications that are managed by a third-party vendor and whose interface is accessed on the client's side. According to Apprenda, *IaaS, PaaS, SaaS (Explained and Compared)*, most SaaS applications can be run directly from a Web browser without requiring any downloads or installations, although some require plugins. PaaS is used for applications and other development while providing cloud components to software. The advantage of PaaS is that user can develop or customize

applications for their needs. IaaS is a self-service model for accessing, monitoring, and managing remote datacenter infrastructures, such as compute (virtualized or bare metal), storage, networking, and networking services (e.g., firewalls). Users can use IaaS and be charged for their consumption depending on their IaaS usage. For this research in constructing a prototype, DynamoDB from Amazon Web Service (AWS) is a suitable choice. DynamoDB is an IaaS cloud service provided by AWS. DynamoDB is a cloud service that specializes in NoSQL data.

Chang-Soo Kim (2014) defines a sensor as a mechanical part that converts environmental information such as temperature, smoke, moisture, and image changes to an electrical signal that a person or device can identify. The configuration of the sensor consists of three sections: sensing, signal processing, and control. The information is detected by the sensor unit and converted into an electrical signal as either analog or digital in separate ways in the single processing section. Then, the control section acquires advanced information by using the software, or connects with another device and operates immediately. The control section utilizes software to acquire advanced data or connects with another device to function immediately. By transforming surrounding information into electrical information, sensors can be classified into four different kinds. Pressure and acceleration can be detected by some types of physical sensors. CIS, IR, and illumination are converted into electrical information by optical sensors. CO₂, NO_x, and Ph can be detected by chemical sensors. DNA and protein levels can be retrieved by biosensors.

Node.js is an open source development platform for running JavaScript code on the server-side. Node.js is useful for developing applications that require a persistent connection from the browser to the server.

RESTFUL API is an API that uses a program at the endpoint that is internet access that other internet-connected computers can shift data to via HTTP server request to requests to get, put, post, and delete data.

Ch. 3 RESEARCH METHODOLOGY

The water crisis is one of the most critical global issues throughout the world. However, in the state of Mississippi, the water management system is very inefficient. For example, Tupelo's sprinkler systems operate even during rain. Observation of these ineffective management systems demands a need for alternative options. During the research, a methodology prototype was determined to be appropriate. When constructing a prototype, it was decided that the purchase of a wireless keyboard, a small monitor, breadboard, a moisture sensor, cables, and a raspberry pi was necessary. First, to construct the server-side prototype, one must access AWS API-gateway, Lambda, and DynamoDB to store data. Second, the moisture sensor is connected to a breadboard with cables. Third, connect the breadboard and sensors to the raspberry pi. Then raspberry pi connects to the monitor to display Python in order to start writing the Python code to send sensor data to AWS API gateway. Then AWS API gateway sends data to Lambda function. Inside the Lambda function, the code is written in the Node.js format. The code stores the sensor data into DynamoDB. The DynamoDB displays stored data about sensor events, such as water and time. Assuming this sensor is inside of the factory, it means there is a water leak that needs to be fixed. Having a sensor in each building could allow maintenance to identify the leak source and repair it.

Ch. 4 DISCUSSION OF RESULT



<Figure 1 Picture of prototype>

While Raspberry Pi is plugged-in, the sensor detects water and registers an electrical signal. The electrical signals go to the raspberry pi which converts the signal into data.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time
import datetime
import requests
from requests_aws4auth import AWS4Auth

url = 'https://ce04ezqlg3.execute-api.us-east-2.amazonaws.com/dev/compare-yourself'

#GPIO SETUP
channel = 21
GPIO.setmode(GPIO.BCM)
GPIO.setup(channel, GPIO.IN)

def callback(channel):
    if GPIO.input(channel):
        currentDT = datetime.datetime.now()
        print (currentDT)
        print ("Water Detected!")
        auth = AWS4Auth('AKIAJCS5VFK3JDFPU24GAQ',
            'qxJjrUumF3U8VME3FtOnwkWIKfWccoENToF2WE/U',
            'us-east-2',
            'execute-api')
        data = { "userid": "12345.fg", "a": "H2O" }
        r = requests.post(url, auth=auth, json=data)
        print(r.status_code)
        print(r.json())
        print(r.text)
    else:
        currentDT = datetime.datetime.now()
        # The formatting of this line of code is not good. Change it.
        print (currentDT)
        print ("Water stopped!")

GPIO.add_event_detect(channel, GPIO.BOTH, bouncetime=300) # let us know when the pin goes HIGH or LOW
GPIO.add_event_callback(channel, callback) # assign function to GPIO PIN, Run function on change

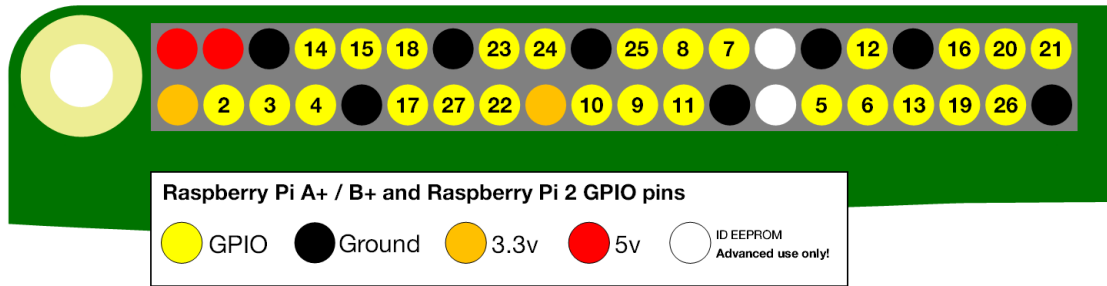
# infinite loop
while True:
    time.sleep(1)
```

<Figure 2 Raspberry Pi Python code about read signal and convert into data.>



<figure 3 Raspberry pi GPIO Header block>--

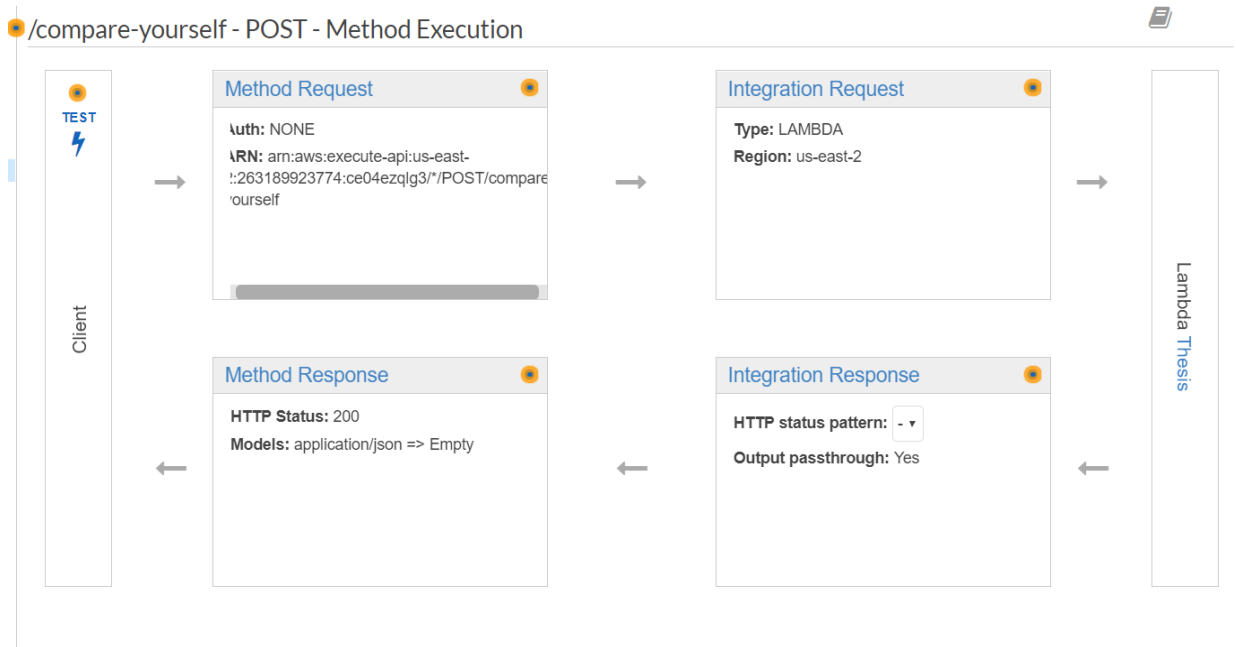
(image from <https://www.raspberrypi.org/documentation/usage/gpio/> retrieved March 27, 2019)



<figure 4 Raspberry pi GPIO Pin diagram>--

(image from <https://www.raspberrypi.org/documentation/usage/gpio/> retrieved March 27, 2019)

Figure 2 presents the client code that is executed on the Raspberry Pi, written in Python. This code sends moisture sensor data to AWS API gateway. The client code imports Python classes for GPIO, time, and AWS authentication. The GPIO features of a Raspberry Pi motherboard provides an electrical connection point for external sensors and devices that interact with Raspberry Pi. In the current project, the code senses electrical voltages that come from the sensors to detect water and converts those voltages to data. The AWS authentication code is sent to AWS API Gateway to store the sensor data. The information transmitted to AWS API Gateway must be sent in the right format. The data was transmitted with the Node.js data format in mind so that it would be processed by AWS Lambda functions.



<Figure 5 AWS API interface setting >

Figure 5 is the UI of the AWS API gateway. The client is the raspberry pi, and it sends the data to a method request. A method request deals with the post method to allow the Python client to post the data. Integration request integrates the data into a particular format. In this research, Lambda was chosen, and the data goes to AWS Lambda function.

```

const AWS = require('aws-sdk');
const dynamodb = new AWS.DynamoDB({region: 'us-east-2', apiVersion: '2012-08-10'});

exports.handler = function(event, context, callback)
{
  'use strict';
  console.log('Loading function');
  exports.handler = (event, context, callback) => {
    console.log('SourceIP =', event.sourceIP);
    callback(null, event.sourceIP);
  };

  function generateRowId(shardId /* range 0-64 for shard/slot */) {
    var ts = new Date().getTime() - CUSTOMEPOCH; // limit to recent
    var randid = Math.floor(Math.random() * 512);
    ts = (ts * 64); // bit-shift << 6
    ts = ts + shardId;
    return (ts * 512) + (randid % 512);
  }

  console.log('Received event:', JSON.stringify(event, null, 2));
  if (event.a === undefined)
  {
    callback("400 Invalid Input");
  }

  var res = {};
  res.sensortype = String(event.a);
  var datetime = new Date().toISOString();
  res.Date = datetime;
  res.userid = String(event.userid)
  var CUSTOMEPOCH = 1300000000000; // artificial epoch
  var neweventidprimarykey = "event:" + generateRowId(4);
  res.eventid = String(neweventidprimarykey);

  if (!(event.a == "H2O"))
  {
    callback("400 Invalid Operand");
  }
}

```

<Figure 6 AWS Lambda function code to send data to AWS DynamoDB to store data >

```

if (!(event.userid == "12345.fg"))
{
  callback("400 Invalid Credentials");
} else {
  var params =
  {
    Item:
    {
      "Eventid":
      {
        "S": res.eventid
      },
      "UserId":
      {
        "S": res.userid
      },
      "SensorType":
      {
        "S": res.sensortype
      },
      "Date":
      {
        "S": res.Date
      },
      "SourceIP":
      {
        "S": "130.130.130.130"
      }
    },
    TableName: "sensorlog"
  };
  dynamodb.putItem(params, function(err, data)
  {
    if (err)
    {
      console.log(err);
      callback(err);
    }
    else
    {
      console.log(data);
      callback(null, data);
    }
  });
  callback(null, res);
}
};

```

This Node.js code receives data from Raspberry Pi Python code that was received by AWS API Gateway. This Node.js code acknowledges the event notification from Raspberry Pi. The AWS Lambda function stores data into AWS DynamoDB. A Lambda function is an unnamed function that can operate without backend service or application maintenance.

The second page of the figure 6 code creates parameters in a structure for sending value to AWS DynamoDB. For example, event.id was created for primary key value, and the user id was from the client device for identifying the client device. The sensor type is for describing the type of the sensor detected, DateTime is for identifying the date, and time. SourceIP means for displaying an IP address for the client device.

Scan: [Table] sensorlog: Eventid ^ Viewing 1 to 24 items

Eventid	Date	SensorType	SourceIP	Userid
event:8234731403544876	2019-02-27T21:47:38.946Z	H2O	130.130.130.130	12345.fg
event:8234731417897238	2019-02-27T21:47:39.384Z	H2O	130.130.130.130	12345.fg
event:8234731433527552	2019-02-27T21:47:39.861Z	H2O	130.130.130.130	12345.fg
event:8234731445455249	2019-02-27T21:47:40.225Z	H2O	130.130.130.130	12345.fg
event:8234731460626435	2019-02-27T21:47:40.688Z	H2O	130.130.130.130	12345.fg
event:8234731475765627	2019-02-27T21:47:41.150Z	H2O	130.130.130.130	12345.fg
event:8234732136499683	2019-02-27T21:48:01.314Z	H2O	130.130.130.130	12345.fg
event:8234732150491552	2019-02-27T21:48:01.741Z	H2O	130.130.130.130	12345.fg
event:8234732166515105	2019-02-27T21:48:02.230Z	H2O	130.130.130.130	12345.fg
event:8234732179523613	2019-02-27T21:48:02.627Z	H2O	130.130.130.130	12345.fg
event:8234732194924855	2019-02-27T21:48:03.097Z	H2O	130.130.130.130	12345.fg

<Figure 7 Displaying the result DynamoDB>

In figure 7 DynamoDB displays the converted data lists from the moisture sensors. It displays date, time sensor type, user id number, and event id number.

Ch. 5 CONCLUSION

The prototype that was constructed can monitor data by watching incoming event calls through AWS API Gateway. This monitoring system could be used on a mass scale and could be building-coded to benefit maintenance workers to find out the problems with products or buildings. The prototype constructed in AWS DynamoDB can be extended to store other different sensor data at the same time. Another way this research can operate daily is by modifying it in different ways. For example, implementing this research into a fish tank would be beneficial: the only thing that has to be done is changing the sensors and monitoring the data.

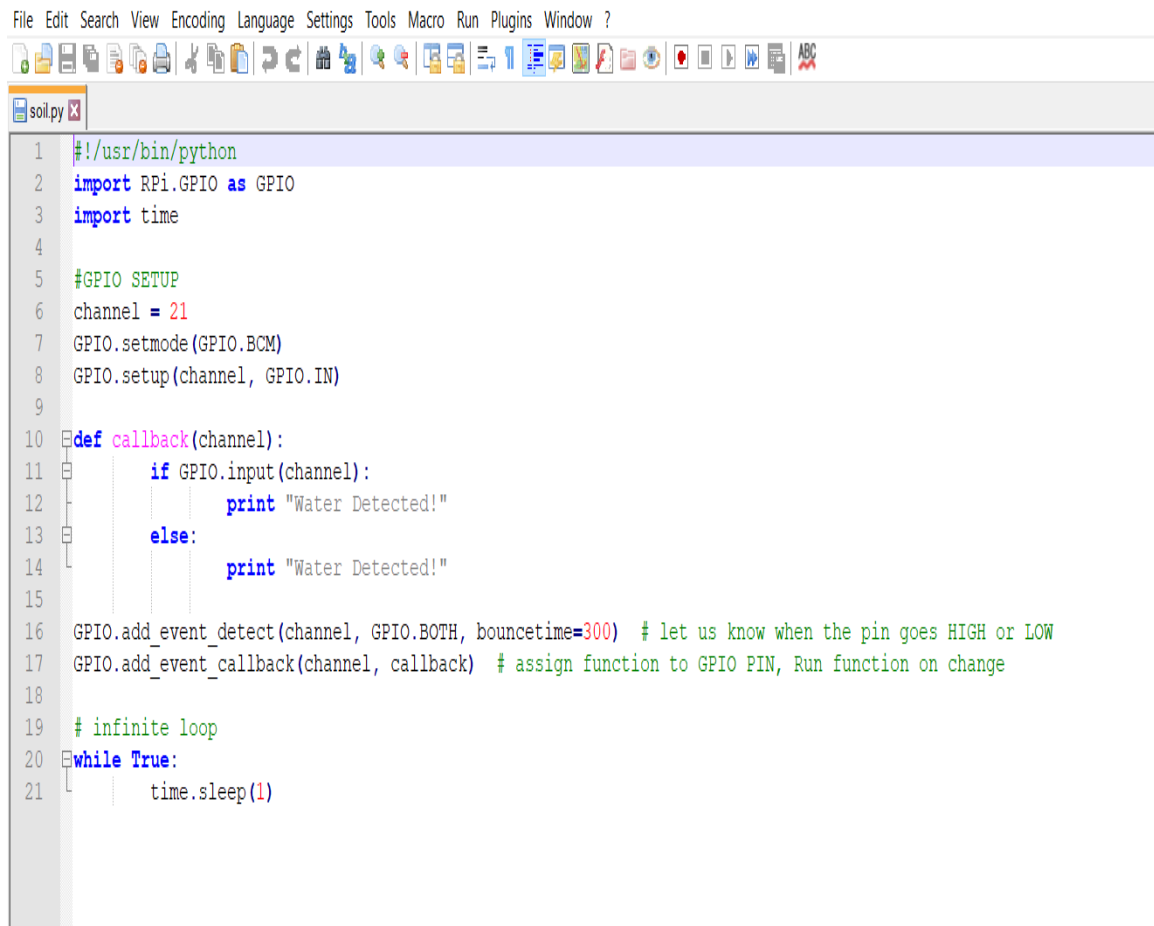
One of the unexpected outcomes was that the data transmission did not work correctly because the cable connections sometimes could not transmit data. It was not able to store DynamoDB. Additionally, because of bad electrical connections, the sensor was not always able to detect water. Thus, it could send the wrong data. Therefore, further development of the research needs a better cable in order to transfer data correctly. It also needs to output valid IP addresses. Another limitation on the research was the discomfort of the user interface. Because the prototype was built on Raspberry Pi, it needed a separate monitor to interact with it. Furthermore, the Raspberry Pi needs a Wi-Fi connection in order to operate.

Ch. 6 DIRECTION FOR FUTURE THESIS

There are some more directions for future research. Figure 5 displays that the integration and method response was not used at the AWS API Gateway. In the future, callback function needs to be implemented so that a user can get a notification when there is a change. To create callback function, DynamoDB needs to be triggered to invoke AWS Lambda function. The AWS Lambda function will be set so that it will be redirected to AWS SNS, which stands for simple notification system, and data will be sent from AWS SNS to the user's mobile device or email. Future studies need to find a better way to identify client devices from a software perspective since AWS API Gateway does not allow the display of valid IP address. An alternative way is needed. From a hardware perspective, it needs integrated IoT devices with sensors on it using breadboard and cables to connect the sensor with IoT. The connection problem is one of the main issues. Furthermore, in the future, researchers can create a mobile application in order to check the data status more efficiently.

APPENDIX A

Soil Moisture Sensor (Raspberry Pi)



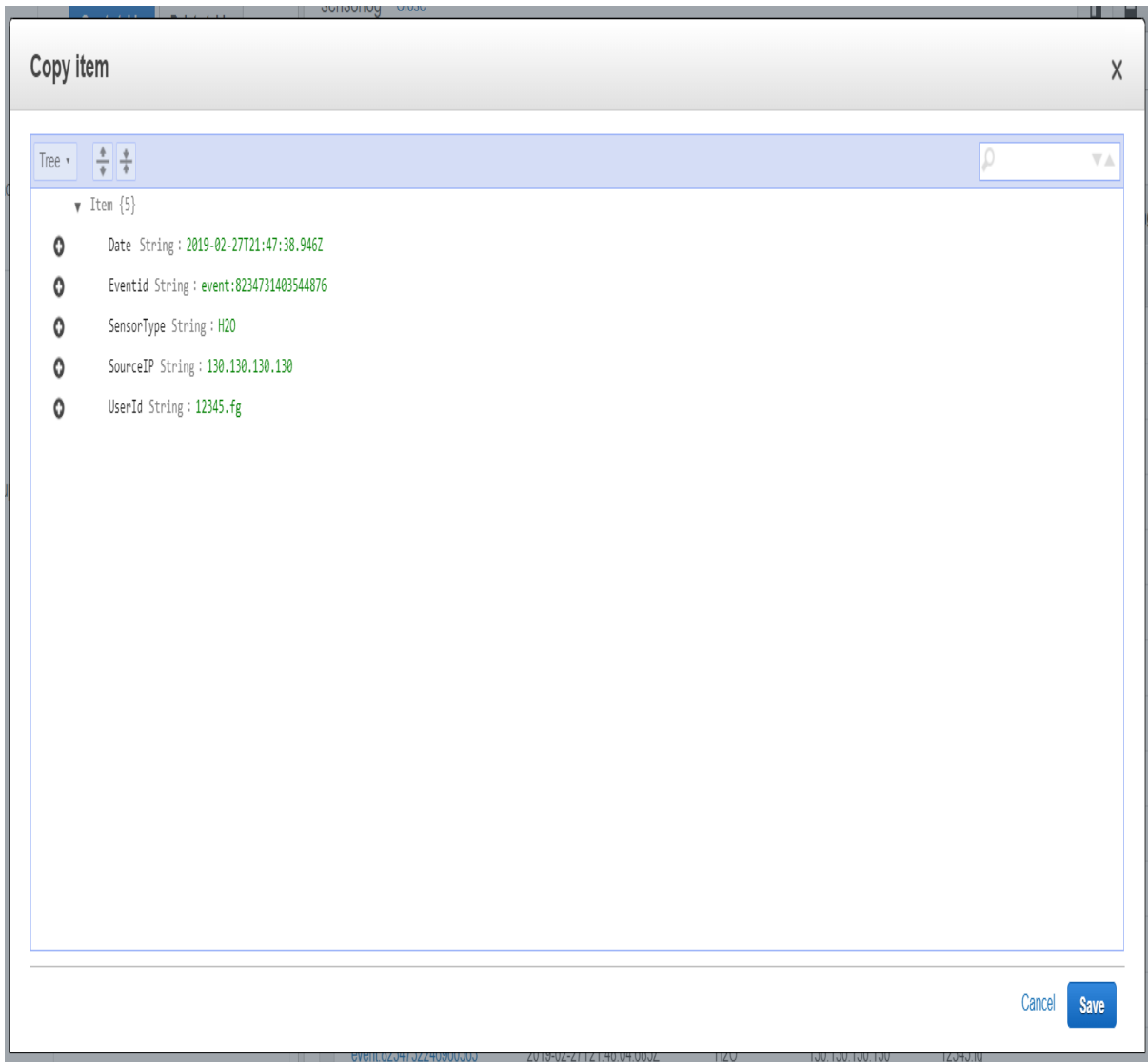
The image shows a screenshot of a Python IDE window titled 'soil.py'. The code is as follows:

```
1  #!/usr/bin/python
2  import RPi.GPIO as GPIO
3  import time
4
5  #GPIO SETUP
6  channel = 21
7  GPIO.setmode(GPIO.BCM)
8  GPIO.setup(channel, GPIO.IN)
9
10 def callback(channel):
11     if GPIO.input(channel):
12         print "Water Detected!"
13     else:
14         print "Water Detected!"
15
16 GPIO.add_event_detect(channel, GPIO.BOTH, bouncetime=300) # let us know when the pin goes HIGH or LOW
17 GPIO.add_event_callback(channel, callback) # assign function to GPIO PIN, Run function on change
18
19 # infinite loop
20 while True:
21     time.sleep(1)
```

(Retrieved from <https://www.instructables.com/id/Soil-Moisture-Sensor-Raspberry-Pi/>)

APPENDIX B

Table structure and format for AWS DynamoDB



APPENDIX C

Simple Calculator Lambda Function

```
1 console.log('Loading the Calc function');
2
3 exports.handler = function(event, context, callback) {
4     console.log('Received event:', JSON.stringify(event, null, 2));
5     if (event.a === undefined || event.b === undefined || event.op === undefined) {
6         callback("400 Invalid Input");
7     }
8
9     var res = {};
10    res.a = Number(event.a);
11    res.b = Number(event.b);
12    res.op = event.op;
13
14    if (isNaN(event.a) || isNaN(event.b)) {
15        callback("400 Invalid Operand");
16    }
17
18    switch(event.op)
19    {
20        case "+":
21        case "add":
22            res.c = res.a + res.b;
23            break;
24        case "-":
25        case "sub":
26            res.c = res.a - res.b;
27            break;
28        case "*":
29        case "mul":
30            res.c = res.a * res.b;
31            break;
32        case "/":
33        case "div":
34            res.c = res.b===0 ? NaN : Number(event.a) / Number(event.b);
35            break;
36        default:
37            callback("400 Invalid Operator");
38            break;
39    }
40    callback(null, res);
41};
```

(Retrieved from <https://docs.aws.amazon.com/apigateway/latest/developerguide/simple-calc-nodejs-lambda-function.html>)

REFERENCE

- Choi, J.W. (2016). □□□□□ (IoT) □□□ □□□ □□ [Current Status and Prospects of the Internet (IoT) Market KOTRA]. Retrieved from <http://news.kotra.or.kr/user/globalAllBbs/kotranews/list/2/globalBbsDataAllView.do?dataIdx=148067&column=&search=&searchAreaCd=&searchNationCd=&searchTradeCd=&searchStartDate=&searchEndDate=&searchCategoryIdxs=&searchIndustryCateIdx=&page=54&row=100>
- Dillon-Roberts, S. (2018). An overview of cloud computing. Retrieved from <http://digitalresources.nz/article/OZcgU6Y>
- Getting Started with REST APIs in Amazon API Gateway. (n.d.). Retrieved from <https://docs.aws.amazon.com/apigateway/latest/developerguide/getting-started.html>
- GPIO(general-purpose input/output). (n.d.). Retrieved from <https://www.raspberrypi.org/documentation/usage/gpio>
- IaaS, PaaS, SaaS (Explained and Compared). (n.d.). Retrieved from <https://apprenda.com/library/paas/iaas-paas-saas-explained-compared/>
- Kim, C.S. (2014). □□□ □□ □□ □□ [Smart sensor technology trend]. Retrieved from <http://www.elec4.co.kr/article/articleView.asp?idx=7276>

notionquest (2017, March 6). AWS Lambda function write to DynamoDB [MSG 1].

Message posted to <https://stackoverflow.com/questions/42623084/aws-lambda-function-write-to-dynamodb>

piddlerintheroot (n.d). Soil Moisture Sensor (Raspberry Pi). Retrieved from

<https://www.instructables.com/id/Soil-Moisture-Sensor-Raspberry-Pi/>

Schwarz Müller, M. (2019, April 20). AWS Serverless APIs & Apps - A Complete

Introduction. [udemy.com online course]. Retrieved from

<https://www.udemy.com/aws-serverless-a-complete-introduction/>

Simple Calculator Lambda Function. (n.d.). Retrieved from

<https://docs.aws.amazon.com/apigateway/latest/developerguide/simple-calc-nodejs-lambda-function.html>

What Is AWS Lambda? (n.d.). Retrieved from

<https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>