Cloud-based Remote Monitoring System

Tanmay Chandavarkar
Dept. of Computer Engineering
San Jose State University
San Jose, CA, USA
tanmay.chandavarkar@sjsu.edu

***Abstract*—In this paper, cloud-based services are used to implement and augment the field of remote monitoring. This kind of monitoring requires a huge network of hardware and storage devices needs to be setup for exchange of data, which is further analyzed using algorithms to further enhance user experience. These algorithms require complex custom-made applications to be developed by every user working in this field. This is made easier by using services provided by some cloud-based companies like Amazon AWS, helping the user to connect to their cloud system using their tailored AWS IoT approach which is demonstrated in this paper. This made is easy to use, reliable, fully scalable and can also handle a large amount a data. This has been successfully implemented in this paper using a temperature-humidity sensor (DHT22) interfaced with a Raspberry Pi connected to AWS and retrieved on the client-side to plot graphs over time.**

***Index Terms*— AWS IoT Core, Lambda Function, PyQT Graphics Library, SQS FIFO Queue.**

I. INTRODUCTION

Internet of things (IoT) is a system of connected devices (computing/mechanical/digital) or "Things" with unique identifiers and which are able to autonomously transfer data. IoT is emerging as a growing topic of interest for many top multinational companies which has the potential to impact out personal and work life. The ever-growing number of devices has led to the necessity for the presence a system where all the devices on the network would be connected by a common protocol and can communicate with each other and exchange data within the system to enhance user experience and make human lives simpler.

Cloud technology plays a very important role in IoT applications. Cloud based services allow programmers to use a Client-Server model so that data can be sent or received irrespective of the distance between them. It also enables organizations to use a virtual computational resource like a virtual machine, instead of constructing a computing infrastructure in each of the company's campuses. Many top cloud-based companies like Google Cloud, AWS, Microsoft Azure provide IoT services to connect devices and collect data from these devices and analyze them. AWS provides IoT services for embedded devices like Raspberry Pi to connect to like Lambda Functions, Simple Queue Service (SQS), Simple Notification Service (SNS) to monitor data over the cloud take actions accordingly. These cloud providers also make sure that the data is fully secure and protected from other users.

II. THEORY

A. Amazon Web Services

Amazon Web Services (AWS) is affiliated to Amazon and provides cloud computing platforms and related APIs to developers. These cloud computing services are provided as a set of technical infrastructure and distributed computing building blocks and tools. Their virtual computers emulate real computers including their hardware like the CPUs and GPUs for mathematical and graphical processing, local and cache memory, secondary memory, operating systems with networking and application software such as databases, web servers and sockets, etc.

B. PyQt Library

Qt is an open-source library developed in C++ with APIs to develop graphical user interfaces and cross-platform applications that can run on various software platforms and operating systems like Windows, macOS, Linux/UNIX based systems, etc. These APIs can even run on embedded systems without any changes to the core codebase. PyQt extends this library to Python version2 and version3. It combines the best of both the Qt and Python worlds. With the power of the Qt toolkit and the simplicity of Python, a developer can create complex GUIs with relatively easier code.

C. Lambda Function

Starting November 2014, AWS has been providing developers with an event-driven and serverless platform to perform computing called the Lambda. It runs code according the to the triggers set it and displays the incoming data. Its purpose is to simplify applications that respond to events and new incoming data. An instance of the Lambda is triggered within a millisecond of an occurring event. It supports Node.js, Python, Go, etc. and a few more are been targeted by AWS. Fig.1 shows the structure of a lambda function in AWS.

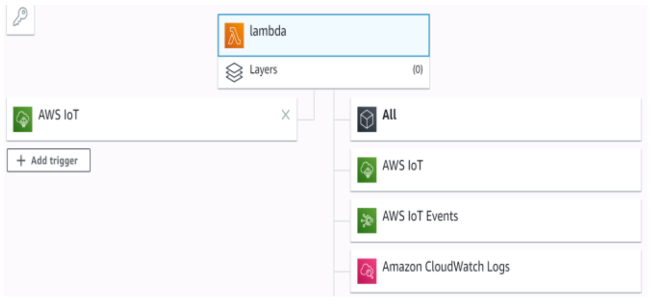


Fig.1: Structure of an AWS Lambda Function

D. SQS

The Simple Queue Service is a distributed messaging system started by the AWS in 2004. It provided messaging queues to redundantly store data across multiple Amazon servers. It provides Standard and FIFO queues. Standard queues ensure at least one delivery of each message but can send copies of the same message more than once. They are designed to support unlimited exchanges of data per second per API call. FIFO queues possess all the attributes of standard queues but ensure the message delivery order in the same order as entered by the user. FIFO queues use batching which limits these to 3000 transaction per sec. It also has a deduplication feature to remove duplicate messages.

E. BOTO

Boto is a Python based software development kit for AWS, which enables innovators to manage AWS services.[3] It also provides a simpler object-based API along with a firmware level access to AWS services. It can be used to send and receive data to/from an SQS queue, among its many other functionalities.

III. Hardware Implementation

A. Components

i. Raspberry Pi

Raspberry Pi is a small single-board computer used in many popular embedded system applications. Many different versions of the board are released by the company available to be used by developers.

The model used for this project is the Raspberry Pi 3 Model B+. It is a 40-pin micro-computer with a Broadcom SoC with a 700 MHz ARM CPU and an on-chip GPU as well. It has an onboard Wi-Fi module and an Ethernet connector to provide for network connectivity over the internet. It also has and audio jack for audio streaming. It also provides 4 USB port to connect peripherals like keyboards, mice, etc. needed to complete a development system. It can be powered on using a 3.3V adapter connected via a micro-USB port. Its 40-pins are well spaced and provide GPIO, SPI, UART, I2C functionalities that are essential in embedded system applications. It also outputs 0V, 3.3V & 5V for external hardware.

Fig.2: Raspberry Pi 3 Model B+

Table 1: Raspberry Pi 3 Model B+ Specifications

CPU	Core	64-bit ARM Cortex-A53
	SoC	Broadcom BCM2837B0
	Speed	1.4GHz
	Power	5V/2.5A DC
MEMORY	RAM	1GB LPDDR2 SDRAM
	ROM	512kB on chip flash
NETWORK	WIFI	Dual band 802.11ac WLAN
	Ethernet	Gigabit Ethernet
	Bluetooth	Version 4.2

The Raspberry Pi Foundation provides a Raspbian OS for the developers which is a GUI designed to interact with the computer. It is a Debian-based operating system which has over 30,000 packages that are pre-compiled and can easily be imported on the hardware. It has inbuilt applications like a file system explorer, Chromium (lightweight browser based on Google Chrome), terminal and Scratch for developers. It supports coding languages like C++, Python, HTML5, Javascript, Java, JQuery, Perl, Erlang, etc.

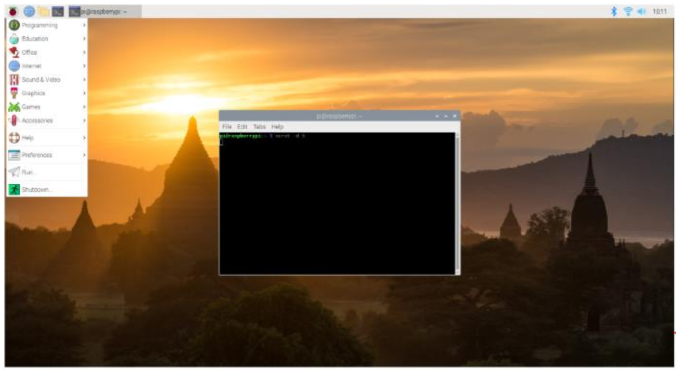


Fig.3: Home screen on Raspbian OS

ii. DHT22

DHT22 is low-cost temperature and humidity sensor. It is a 4-pin digital sensor which uses thermistor and a capacitor-based humidity sensor to measure the temperature and humidity respectively of the surrounding air. It outputs a digital signal on the data pin, which removes the need for analog pins and analog-to-digital conversion.

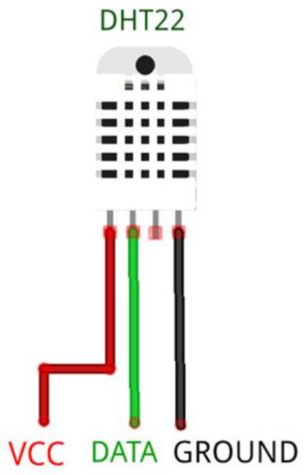


Fig.4: DHT22 Pinout structure

B. *Hardware Connectivity*

This project utilizes the GPIO pin connectivity of the Raspberry Pi pins to connect the sensor and read values. As seen from the sensor pinout diagram in Fig. 4, the V_{cc} pin is connected to the 3.3V output pin on the raspberry pi, the Data pint is connected to the pin number 4 of the pi and a common ground is connected for proper synchronization. Fig. 5 shows the graphical representation of the pin connections between the sensor and the raspberry pi module.

Table 2: Hardware Connections

Raspberry Pi		DHT22 Pin
Pin Name	Pin Number	
DC Power (3.3V)	01	Vcc
Ground	06	Gnd
GPIO04(GPIO_GCLK)	07	Data

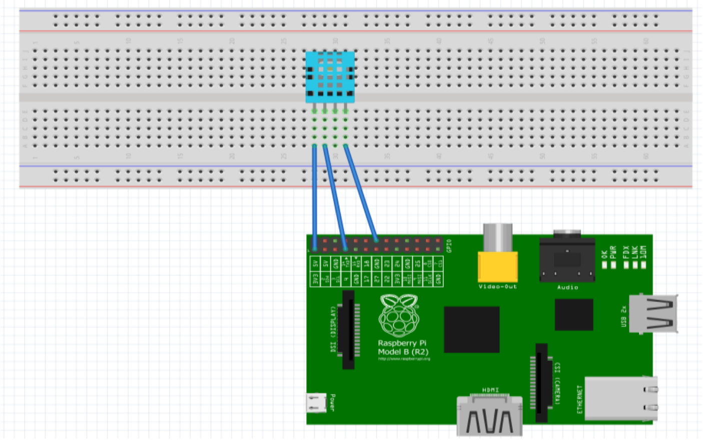


Fig.5: Hardware Layout

IV. System Workflow

This project uses a Client-Server Architecture model for transferring data.
On the Server side, there is a Raspberry Pi which get sensor values from the DHT22 sensor. These values are appended into a '.csv' file with their timestamp. A connection to the AWS is established after procuring the required certificates from AWS. The AWS IoT Core is set as input terminal of data for the cloud. This is where the data is visible on AWS. This is then added as a Trigger point for the Lambda function. This data is then programmed to be sent into a Simple Queue Service (SQS) where data in aligned as per its arrival in the FIFO queue.
On the Client side, there is a Raspberry Pi using a Boto3 client to connect to the FIFO queue to get sensor values. These values are received and displayed as unit values on the client-side GUI and also as graphs to show the variance over time. Fig. 6 shows the graphical block diagram of the system Workflow.

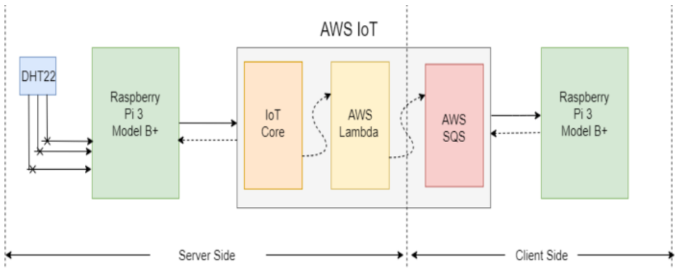


Fig.6: System Workflow

V. Server-Side Software Implementation

A. Reading sensor values

After downloading the Raspbian OS and getting familiarized with the environment, the Adafruit_DHT library was installed on the Pi using:
>> `sudo pip3 install Adafruit_DHT`, where Python Package Index (pip3) is a package installer for Python3. Adafruit_DHT is a ready-to-use Python library developed for the DHT22 sensor to be used for the Raspberry Pi & Beaglebone black boards. It reads the pin to which the Data pin of DHT22 is connected. The drivers are written in C and are called in the mentioned library.
The sensor values are read along with the time at which it is recorded using the following:

```
rh, temp = Adafruit_DHT.read_retry(sensor, pin)
timstamp=datetime.now().strftime('%d%b%Y%H:%M:%S')
```

B. Connecting to AWS

To connect to AWS, firstly an account is required on AWS (student or member). After creating a free student account, the hardware to be used, Raspberry Pi in this case, was registered on the IoT Core platform on the AWS. The AWS then generates certificates and public and private keys for the registered device. These are downloaded as a '.zip' file and extracted on the local drive on the Raspberry Pi. A basic publish subscribe method is used to send data for which a generic code can be found in the AWS git. This basicpubsub file was customized to access the '.csv' file (which has all the sensor data with the timestamp) and initializing an instance of AWSIoTMQTTClient as follows.

Pseudo Code:
```
myAWSIoTMQTTClient.connect()
myAWSIoTMQTTClient.subscribe(topic,customCallback)
```

Further, modifying the and running start shell script provided in AWS credentials packet to run the modified pubsub code in the Linux terminal window, a successful connection to the AWS was established.

C. Triggering the AWS Lambda Function

For cloud integration, AWS Lambda has to be configured to received and forward messages. A function must be created and linked to AWS IoT core to be able to receive sensor data messages. As shown in Fig 1, SQS Service must be added to the destination parameters of Lambda so forward the incoming data to the queue. A Node.js script must be entered in the console embedded on the Lambda function web page.
Pseudo Code:
```
var q_url='sqs_fifo_URL';
exports.handler = function(event) {
    var params = {
      MessageBody: JSON.stringify(event),
      QueueUrl: q_url,
    };
    sqs.sendMessage(params, function(data){
      console.log('data:',data.MessageId);
      };
};
```
Here, the handler waits on an external event to occur and converts that incoming data to string using JSON.stringify() and sends it to SQS using sendMessage().

CloudWatch Metrics also can be used to ensure and monitor successful connections to the Lambda function as shown in Fig.7. It shows the number of invocations, the time and duration of the function invocations and the error to success rate percentage.

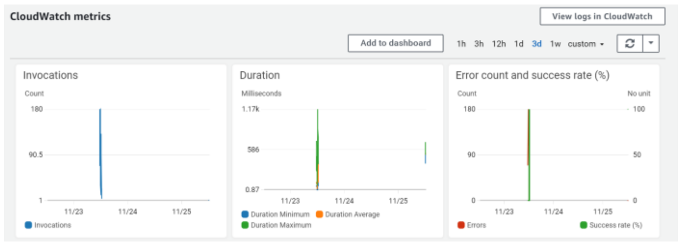


Fig.7: CloudMetrics Monitor for AWS Lambda

VI. Client-Side Software Implementation

A. *Connecting to Boto client*

To connect to the Boto3 service, first a resource has to be initialized as shown in the pseudo code below. Further the AWS FIFO queue has to be instantiated to receive the messages. Then the messages can be received using the receive_message() method.

Pseudo Code:

```
res_var=boto3.resource('resource_name')
q=res_var.get_queue_by_name('Queue_name.fifo')
rec=q.receive_messages('Queue_URL')
```

B. *Creating the GUI-Login Window*

After importing the required libraries like QtCore, QtWidgets, QDialog, a class "Login" is created that inherits properties of QtWidget class. `pyqtSignal()` is used to create a new window from the terminal window. Is used to set the title of the window. Buttons are created inside the login window using `QPushButton()`. Widgets are added using `layout.addWidget()`. Credentials are set and verified for authentication.

Pseudo Code:

```
from PyQt5 import QtCore, QtWidgets
class Login(QtWidgets):
    switch_window = QtCore.pyqtSignal()
    self.setWindowTitle('Login')
self.button= QtWidgets.QPushButton('A')
    self.username.setPlaceholderText('B')
layout.addWidget(self.username)
```

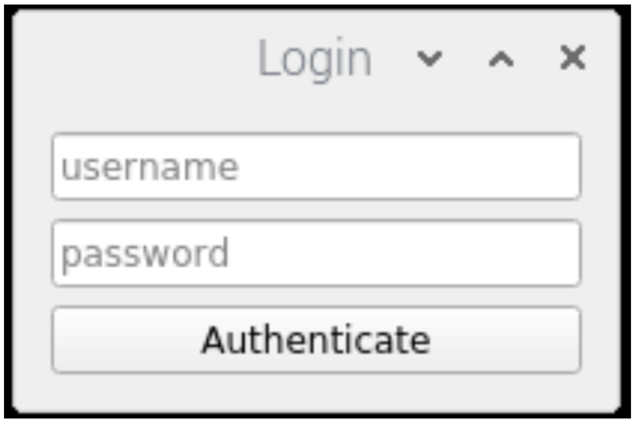


Fig.8: Login Window

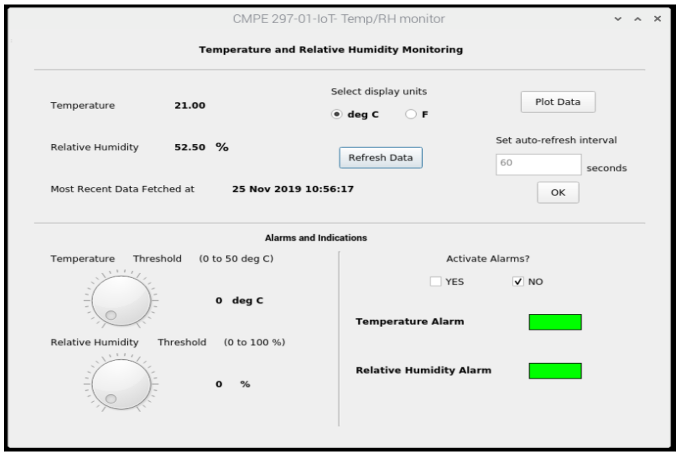
C. *Creating the GUI- Main Window*

This is the part where the data is going to be displayed and the graphs will be plotted. Matplot library is used to plot the graphs as shown below.

Pseudo Code:

```
import matplotlib.pyplot as mpp
fig = mpp.figure()
mpp.subplot(2,1,1) #temperature plot
mpp.plot(var1,var2)
mpp.subplot(2,1,2) #Humidity plot
mpp.xlabel('A')
mpp.ylabel('B')
```

Here, figure() invokes an instance of the class figure included in matplot library. Subplot() divided the figure in to two halves and the coordinates (2,1,1) denote the upper half. All the code for the upper graph has to be written before the next subplot() call. Plot() denotes what values are to plotted on the X and Y axis. Xlabel() and ylabel() are used to label the X and Y axis with desired names. Matplot is the same library which is used in MatLab to plot graphs for signal processing circuits. Pyplot is the Python extension of the matplot library.



Temperature and Relative Humidity Monitoring

Temperature 21.00

Relative Humidity 52.50 %

Most Recent Data Fetched at 25 Nov 2019 10:56:17

Select display units: [✓ deg C] F

Set auto-refresh interval 60 seconds

Alarms and Indications

Temperature Threshold (0 to 50 deg C) 0 deg C

Relative Humidity Threshold (0 to 100 %) 0 %

Activate Alarms? YES [✓ NO]

Temperature Alarm 🟩

Relative Humidity Alarm 🟩

Fig.9: Main Window

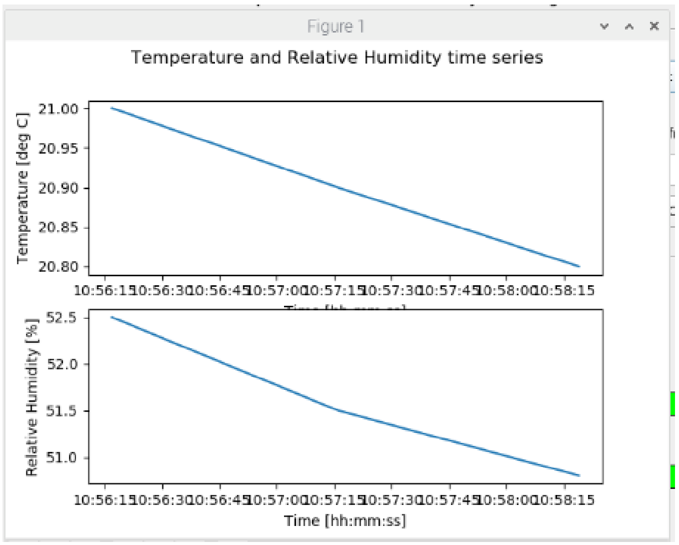


Fig.10: Graphs showing Temperature & Humidity varying over time

VII. Testing And Verification

The project ran successfully without any errors. Data Transfer was successfully established between the Raspberry Pi and the AWS IoT Core. Incoming data was monitored in the AWS terminal. Boto3 client was used to successfully connect to the SQS FIFO queue on the client side. Sensor data was extracted from the SQS FIFO queue and used to plot the graphs versus time, as shown in Fig.10.

VIII. Industrial Applications

The work presented in this paper has been implemented as a prototype and similar technology is being used in many IoT applications developed by companies like Google for their smart-home products like Google Nest. This can also be extended to smartwatches and smart apparel to give more control to the users, wherein actuators can be triggered using software applications developed for smartwatches to control home temperature, humidity, setup theft alarms, etc.

IX. Conclusion

The aim of the project was to establish the communication between embedded hardware and a cloud-based service. This was successfully achieved using a Raspberry Pi interfaced with the AWS IoT Core.

Acknowledgment

I would like to thank Dr. Ammar Rayes for sharing his knowledge on Internet of Things and for guiding me through his knowledge and his book during the various steps of the project.

References

[1] Getting started with AWS, AWS Open-source Documentation
[2] SQS Developer Guide, AWS Open-source Documentation
[3] Boto3 Documentation and Manual, AWS Open-source Documentation
[4] Raspberry Pi Manual and Official Documentation, Raspberry Pi foundation
[5] Digital-output relative humidity & temperature sensor/module, Aosong Electronics Co., Ltd.
[6] AWS Lambda Documentation, AWS.
[7] PyQT Reference Guide, RiverBank Computing.