|  |  |
| --- | --- |
| **Date of Submission:** | 20th August 2018 |

|  |  |
| --- | --- |
| **Prepared for:** | Ms Zhao Lin |

|  |  |
| --- | --- |
| **Class:** | DISM/FT/3A/03 |

|  |  |
| --- | --- |
| **Submitted by:** |  |

|  |  |
| --- | --- |
| **Student ID** | **Name** |
| 1625749  1609637 | Marcus Kho  Jonathan Au  Daniel Mario Tablan |

**IOT CA2**

**Step-by-step Tutorial**

**DIPLOMA IN BUSINESS INFORMATION TECHNOLOGY**

**DIPLOMA IN INFORMATION TECHNOLOGY**

**DIPLOMA IN INFOCOMM SECURITY MANAGEMENT**

**ST0324 Internet of Things (IOT)**

**2017/2018 Semester 1**

Contents

[1. Overview 3](#_Toc522494527)

[A. What is the application about 3](#_Toc522494528)

[B. Intro to Amazon Web Services (AWS) 3](#_Toc522494529)

[C. Why develop IoT apps on Cloud? 4](#_Toc522494530)

[D. Summary of the steps that will be described 5](#_Toc522494531)

[2. Hardware Requirements 6](#_Toc522494532)

[Hardware checklist 6](#_Toc522494533)

[3 Prerequisites 9](#_Toc522494534)

[3.1 Enable camera with raspi-config 9](#_Toc522494535)

[3.1Setting up mySQL database 10](#_Toc522494536)

[3.1 Creating various Telepots (4 are required) 13](#_Toc522494537)

[3.1 Enable the reading of RFID/ NFC Tags 14](#_Toc522494538)

[4. Sign up for AWS Educate account 16](#_Toc522494539)

[A. Set up a new AWS account 16](#_Toc522494541)

[5. Sign in to AWS IoT Console 19](#_Toc522494542)

[A. Sign in to the AWS IoT Console 19](#_Toc522494544)

[6. Register your Raspberry Pi as a Thing 20](#_Toc522494545)

[A. Create and register your “Thing” 20](#_Toc522494547)

[B. Copy REST API endpoint of your “Thing” 23](#_Toc522494548)

[C. Create a Security Policy for your RPi 24](#_Toc522494549)

[D. Attach Security Policy and Thing to your Cert 25](#_Toc522494550)

[7. Installation of Python Libraries 27](#_Toc522494551)

[A. Install the required Python library 27](#_Toc522494553)

[8. Installation of Node Red Palletes 29](#_Toc522494554)

[A. Install the required node-red library 29](#_Toc522494556)

[B. Configure Node-red 30](#_Toc522494557)

[9. AWS IoT Rules Part 0 31](#_Toc522494558)

[A. Install AWS Command-line client 31](#_Toc522494560)

[B. Create AWS Role 32](#_Toc522494561)

[C. Create AWS Policy that allows access to all functions 33](#_Toc522494562)

[10. AWS IoT Rules Part 1 34](#_Toc522494563)

[A. Create an AWS SNS Topic 34](#_Toc522494565)

[B. Subscribe to an Amazon SNS Topic 36](#_Toc522494567)

[C. Create an AWS IoT Rule 38](#_Toc522494569)

[11. Code the IoT app 42](#_Toc522494570)

[A. Alarm Program 42](#_Toc522494572)

[B. RFID Program 46](#_Toc522494573)

[C. CCTV Node-Red Program 55](#_Toc522494574)

[D. Temperature Reading Node-Red Program 57](#_Toc522494575)

[12. Web Application Interface 59](#_Toc522494576)

[12.1 Desktop Web Interface 59](#_Toc522494577)

[12.2 Mobile Web Interface 60](#_Toc522494578)

[13. Using the IoT device in real world 62](#_Toc522494579)

[13.1 Application 62](#_Toc522494580)

# Overview

## Where we have uploaded our tutorial

We have uploaded our project primarily on github and it can be accessed at <https://github.com/currysaucee/CA2JonathanMarcusDaniel>

## Why have we chosen to upload to this site

We have chosen github as we have uploaded projects onto the site before and hence are more familiar with the process of committing files to the online site. Moreover, mulitple authorised people can commit changes to the shared git easily and it github can also support multiple Operating System such as MacOSX, Windows and Linux.

## What have we uploaded

We have uploaded a readme.md that contains a short description of our project. We have also uploaded this guide documentation to the github and we have also uploaded the py files that are essential to our project.

## What is the application about

This application is a SmartHome System. The camera is used to detect suspicious intrusion or activity around a protected/personal area, and also a surveillence camera to view and monitor the surroundings. The DHT11 Humidity and Temperature sensor is to sense the temperature and humidity of the house and if the detected temperature is too high, the Air Conditions will be automatically turned on to cool down the house. Where as the RFID Reader and RFID Tags acts as a key to the house and each family member should be in possession of a single RFID tag.

For the Camera, once movement any movement is detected the application will alert the user, and the user can view the live stream via a web interface or an application. From the live stream, the user can choose to record the current live stream and save it.

From the web interface, the user can also download any recorded videos, so that he does not have to connect into the IoT device to download. Lastly, it also comes with a doorbell function. From there, the user can also view the temperature and humidity of the house and control the air conditions around the house.

The various telepots that are used to support the applications acts as a remote for the sensors, like the RFID Scanner, used in this single application.

## Intro to Amazon Web Services (AWS)



Amazon Web Services, a subsidiary of Amazon.com, offers a suite of cloud computing services that operate from 16 geographical regions across the world.

“Cloud computing” is the on-demand delivery of compute power, database storage, applications, and other IT resources through a cloud services platform via the internet with *pay-as-you-go* pricing.

With cloud computing, companies need not make large upfront investments in hardware and spend a lot of time on managing that hardware. Instead, they can provision exactly the right type and size of computing resources they need from cloud providers like AWS, and pay AWS based on the configuration of the resources they use, and the amount of usage they incur.

## Why develop IoT apps on Cloud?

In many typical IoT deployments, the central system might be collecting data from hundreds or even thousands of sensor devices. It requires tremendous amount of resources to maintain the infrastructure needed to manage the collection of data from the devices and to process them, if the hardware is all maintained by the organisation utilising the IoT solution.

Using IoT Cloud Vendors such as AWS, Microsoft Azure, IBM Watson,Google Cloud to manage the collection of sensor data, organisations can be rest assured that the latency of their IoT apps is kept low as these cloud vendors can typically scale to billions of devices and trillions of messages. The cloud vendors would also help to ensure the security of the cloud, which takes another load off the IoT developer.

## Summary of the steps that will be described

|  |  |  |
| --- | --- | --- |
|  | Section | Description |
|  | Overview |  |
|  |  |  |
| Sections 2 to 8 provides the step-by-step instructions to set up the application | | |
|  | Hardware requirements | Provides overview of hardware required |
|  | Setting up the hardware | Connecting the hardware together |
|  | Prerequisities | Getting ready for the program |
|  | Install libraries | Install necessary libraries |
|  | Coding the program | Coding the programs for devices |
|  | Running the program | Testing the program |
|  | Real-world application | Demostration on how it can be applied in real world |

## How does our final RPI setup look like

## How does our final web application look like

Desktop Web Interface

| No | Task |
| --- | --- |
| a) | Index Page |
| b) | CCTV Videos Page |
| c) | AirCon Page |

Mobile Web Interface

The followings are screenshot of how the Web Interface at 192.168.1.106:1880/ui would look like.

| No | Task |
| --- | --- |
| a) | Index Page |
| b) | CCTV Videos Page |
| c) | AirCon Page |

# Hardware Requirements

Hardware checklist

2.1 Button

|  | Task | |
| --- | --- | --- |
|  | * The picture here shows a tactile push-button * Note that it has 4 'legs'. These legs are the ones that send the signals. * Note that unlike a LED which is an actuator (output), push-buttons act as sensors (inputs) * To know whether a push-button is pressed or unpressed, we can detect its HIGH or LOW state, which are passed through the 'legs' | **BUTTON** |

2.2 Resistor for Push-button

|  | Task | |
| --- | --- | --- |
|  | Just like the LED, adding a button to the circuit can introduce irregularities in the current flow to the RPi and damage our RPi  To ensure our RPi stays safe, we will be adding a 10K Ω **pull-down** resistor to moderate the current flow through the button. You can recognise the 10K ohms resistor by its color bands (brown-black-orange-gold)  Note that when the pull-down resistor is used in the circuit, it will change the current flow as follows:   * When the button is pressed: A small current flows through the input pin connected to the button, and the pin will read HIGH * When the button is unpressed: The current passes through the resistor and directly to GND pin. There is no current passing through the input pin and thus it will read LOW | **10K Ω RESISTOR** |

2.3 PIR Motion Sensor

|  |  |  |
| --- | --- | --- |
|  | Task | |
|  | A PIR Sensor can tell when something nearby moves.  The sensor detects the pattern of infrared energy in its surroundings. If the pattern changes, the sensor outputs a high signal.  The sensor has 3 pins.   * + VCC – Connect this to power (5V or 3.3V)   + VOUT – Connect this to a GPIO pin to read its value   + GND – Connect this to ground | **HC-SR501 PIR Motion Sensor** |
|  | The HC-SR501 PIR sensor has a trimpot on its back for adjusting its time delay and sensitivity.  You can adjust this if your PIR is too sensitive or not sensitive enough - clockwise makes it more sensitive.  Turn the trimpot for the “Time” to the middle for the time duration which the output pin remains high after triggering – clockwise will increase this timing. |  |

2.4 Buzzer

|  |  |  |
| --- | --- | --- |
|  | Task | |
|  | A buzzer is an audio signaling device which is commonly found in circuits to create a buzzing or beeping noise.  Buzzers can be categorized as active buzzers and passive ones. For our lab, we will use active buzzers as they are a lot simpler to use than passive ones though they are slightly more expensive.  An active buzzer can be connected just like an LED but they are even easier to use because a resistor is not needed.  A buzzer typically has 2 pins   * + VOUT – Connect this to a GPIO pin to control its value   + GND – Connect this to ground | **Buzzer** |

2.5 Raspberry Pi camera (piCam)

|  |  |  |
| --- | --- | --- |
|  | Task | |
|  | * The Raspberry Pi camera is an official accessory that hooks up to a special connector on the Raspberry Pi. * It can be used on any model of the Raspberry Pi except the first version of Pi Zero * The camera is mounted on a small printed circuit board and connects through a ribbon cable. * The connector provides direct access between the camera and the processor. This is more efficient than using a webcam, which needs to connect through the USB protocol * The picture on the right shows the camera connected to a Raspberry Pi. | **piCam** |

# Prerequisites

## 3.1 Enable camera with raspi-config

|  | Task | |
| --- | --- | --- |
|  | * Open the Raspberry Pi Configuration Tool from the main menu: |  |
|  | * Ensure the camera software is enabled. If it's not enabled, enable it and reboot your Pi to begin. |  |

## 3.1Setting up mySQL database

| No | | Task | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Once you are logged in successfully to the phpmyadmin web console, you should see the dashboard similar to that below. | | | | | |
|  | | Click on the “New” button as shown | | | | |  |
|  | | Choose a name for your new database and type it in the textfield as shown. Click “Create” button. | | |  | | |
|  | | A success message will be shown to you. | | |  | | |
| e) | | | After creating the database, you may proceed to create a new table inside this database.  Click on the new database on the navigation bar on the left. You will be shown a “Create table” page as shown.  Create a table named users | | | | |
| f) | | | Enter the following values to add the columns into the table  Make sure that the rfid is set to be the primary key | | |  | |
| g) | Next, as a good security practice, we will create a user account that you will use to access the database you just created. Go to “Privileges” and click “Add user” | | | | | | |
|  | Select a username and password, and ensure the privileges of this user is limited to the new database you created only. | | |  | | | |

## Creating various Telepots (4 are required)

| No | Task | | | |
| --- | --- | --- | --- | --- |
| a) | Open Telegram app in your laptop or mobile and start “BotFather” | | | |
| b) | Type /newbot to create a new bot | | |  |
| c) | Give a name to your bot that describes its purpose. |  | | |
| d) | Give a username to your bot. Make sure it ends with \_bot | |  | |
|  | Copy down the access token that is issued by BotFather | |  | |

## Enable the reading of RFID/ NFC Tags

If you are working on a fresh Raspbian install, there are some things you would need to prepare before the code given in the next section can work.

To save time, I have already pre-configured the ST0324 IoT Raspbian image so that you need not go through these steps anymore. I am including the steps in the lab for your future reference.

|  | Task | | |
| --- | --- | --- | --- |
| << Enable SPI via raspi-config >> | | | |
|  | | Run raspi-config, choose menu item “5 Interfacing Options” and enable SPI.  sudo rasp-config |  |
| << Enable device tree in boot.txt>> | | | |
|  | Modify the /boot/config.txt to enable SPI  sudo nano /boot/config.txt | | |
|  | Ensure these lines are included in *config.txt*  device\_tree\_param=spi=on  dtoverlay=spi-bcm2835 | | |
|  | **<< Install Python-dev>>** | | |
|  | Install the Python development libraries  sudo apt-get install python-dev | | |
|  | **<< Install SPI-Py Library >>** | | |
|  | Set up the SPI Python libraries since the card reader uses the SPI interface  git clone https://github.com/lthiery/SPI-Py.git  cd /home/pi/SPI-Py  sudo python setup.py install | | |
|  | **<< Install RFID library >>** | | |
|  | Clone the MFRC522-python library and copy out the required files to your project directory  git clone https://github.com/rasplay/MFRC522-python.git  cd MFRC522-python  sudo cp \*.py ~/p15 | | |

# Sign up for AWS Educate account

To begin coding IoT apps using AWS, you will need to set up an AWS Educate account if you do not already have one.

The AWS Educate is Amazon's programme to help students learn real-world cloud technology skills before graduating. Under this programme, students will get a renewable annual credit of USD100 of AWS services. No credit card is necessary to sign up for this account. However, once you have used up the USD100 credit, your account will automatically be rendered unusable.



## Set up a new AWS account

| No | Task | | |
| --- | --- | --- | --- |
|  | Access AWS Educate website: <https://aws.amazon.com/education/awseducate/apply/>  Select “Students” option | | |
|  | | | |
|  | Select Role as “Student” | | |
|  | | | |
|  | On the sign up form, ensure the following settings are specified, and fill in the rest of the form as appropriate and then click “Next” button. | | |
|  | * Institution name is “Singapore Polytechnic School of EEE” * Email is your SP ichat email address * “Click here to select an AWS Educate Starter” account is selected |  | |
|  | Follow the instructions to verify your account by providing a verification code that is sent to your ichat email account. | |  |
|  | Once your AWS account is ready for use, you will receive an email similar to this one |  | |

# 5. Sign in to AWS IoT Console

After you have received the email confirmation from AWS, you are now ready to log into the AWS IoT console to start the lab.



## Sign in to the AWS IoT Console

| No | Task | |
| --- | --- | --- |
|  | Turn on your Raspberry Pi and confirm you have an Internet connection. | |
|  | Sign in with your AWS console at <https://www.awseducate.com/signin/SiteLogin> | |
|  | In the AWS dashboard, type “IoT Core” to access the AWS IoT Core service. | |
|  | On the Welcome page, choose Get started |  |

# Register your Raspberry Pi as a Thing

In this section, you will learn how to create and register your Raspberry Pi as a “Thing” with AWS IoT.



## Create and register your “Thing”

| No | Task | | |
| --- | --- | --- | --- |
|  | In the left navigation pane, click “Manage” to expand it, then choose “Things”. | |  |
|  | On the page that says “You don't have any things yet”, choose “Register a thing”.  If you have created a thing before, choose Create. |  | |
|  | A thing represents a device whose status or data is stored in the AWS cloud. The Thing Shadows is the state of the device, e.g. is it “on” or “off”, is it “red” or “green” etc.  Our “thing” here is our RPi, so let’s type “MyRaspberryPi” for the name.  Click “Create thing” then “Create a Single Thing” | | |
|  | Emter in any name you want and leave the rest empty and click “Next” | | |
|  | Click on Create Certificate | | |
|  | Download all 3 files in here.  Click “Activate”  And download the root CA for AWS IoT amd then “Done” | | |

## Copy REST API endpoint of your “Thing”

| No | Task | |
| --- | --- | --- |
|  | On the Details page, choose **Interact**. |  |
|  | Copy and paste the REST API endpoint into a Notepad.  You will need this value later. |  |

## Create a Security Policy for your RPi

| No | Task | |
| --- | --- | --- |
|  | Click on “Secure” and then click on “Policies” then choose “Create” |  |
|  | Emter the following values into the input boxes but you can give you Policy any name you want. |  |

## Attach Security Policy and Thing to your Cert

In this section, you will attach both your security policy and your Thing to your X.509 certificate

|  | Task | | | | |
| --- | --- | --- | --- | --- | --- |
|  | On the left nav bar, click “Secure, Certificates” | | | |  |
|  | The X.509 certificate you created earlier is shown.  Click the checkbox beside it, then click “Actions” button and choose “**Attach Policy**” |  | | | |
|  | Check the “MyRaspberryPiSecurityPolicy” you created earlier and click “**Attach**” button. | |  | | |
|  | Let’s attach the “Thing” to this certificate.  Click “Actions” button and choose “**Attach Thing**” | | |  | |
|  | In the Attach things to certificate(s) dialog box, select the check box next to the thing you created to represent your Raspberry Pi, and then choose Attach | | |  | |

# Installation of Python Libraries

By now, you have completed the first part of the process to connect your device to AWS cloud.

In the next few sections, you will begin to code your app. However, before you can start coding an IoT app on the AWS Cloud, you will need to install the AWS Python software libraries first.



## Install the required Python library

| No | Task |
| --- | --- |
|  | Install the AWS Python library with this command  sudo pip install AWSIoTPythonSDK |
|  | The AWS Python SDK has dependency on paho-mqtt, so make sure it is installed on your RPI.  sudo pip install paho-mqtt |
|  | Install the Telepot Python library with this command  sudo pip install telepot |
|  | Install the node-red library with this command  sudo pip install node-red |
|  | Install the RPI LCD library with this command  sudo pip install rpi-lcd |
|  | Install the uv4l library  Run this command  curl http://www.linux-projects.org/listing/uv4l\_repo/lrkey.asc | sudo apt-key add -  Edit the **/etc/apt/source.list** file and add the following line:  deb http://www.linux-projects.org/listing/uv4l\_repo/raspbian/ jessie main  Update the system and to fetch and install the packages  sudo apt-get update  sudo apt-get install uv4l uv4l-raspicam  Run the following command  sudo apt-get install uv4l-server uv4l-uvc uv4l-xscreen uv4l-mjpegstream uv4l-dummy uv4l-raspidisp |

# Installation of Node Red Palletes



## Install the required node-red library

| No | Task |
| --- | --- |
| a) | Install the required palletes in your node-red, here is an example of one. |
|  | node-red-dashboard  node-red-contrib-python3-function  node-red-contrib-dht-sensor  node-red-contrib-telegrambot  node-red-contrib-telegrambot-home |

## Configure Node-red

| No | Task |
| --- | --- |
| a) | Configure the Node-red settings to retrieve files from your Raspberrypi |
| b) | Edit the settings.js file located at /.node-red |
| c) | Find the line that contains “httpStatic:…” |
| d) | Change it to “httpStatic: ‘/’, |
| e) | Create two folders “/home/pi/image” and “/home/pi/picamVideos” |

# AWS IoT Rules Part 0

Due to the limitations of the AWS Educate account, you will need to go through the following steps in order to make use the AWS IoT Rules feature.



## Install AWS Command-line client

| No | Task |
| --- | --- |
|  | Run the following command on your Raspberry Pi to install the AWS Command-line client on your Raspberry Pi  sudo pip install awscli --upgrade --user |
|  | Edit the .profile to include the path of the AWS client  sudo nano ~/.profile |
|  | Add in the following code after the last line and save the file  export PATH=~/.local/bin:$PATH |
|  | Type the following command at the command-line prompt to make the new settings take effect immediately  source ~/.profile |

## Create AWS Role

| No | Task |
| --- | --- |
|  | Create a file named **iot-role-trust.json** with the following contents |
|  | **{**  **"Version":"2012-10-17",**  **"Statement":[{**  **"Effect": "Allow",**  **"Principal": {**  **"Service": "iot.amazonaws.com"**  **},**  **"Action": "sts:AssumeRole"**  **}]**  **}** |
|  | Copy iot-role-trust.json to your Rpi, and then run the following command  aws iam create-role --role-name my-iot-role --assume-role-policy-document file://iot-role-trust.json |

## Create AWS Policy that allows access to all functions

| No | Task |
| --- | --- |
|  | Create a file named **iot-policy.json** with the following contents |
|  | **{**  **"Version": "2012-10-17",**  **"Statement": [**  **{**  **"Effect": "Allow",**  **"NotAction": [**  **"iam:\*",**  **"organizations:\*"**  **],**  **"Resource": "\*"**  **},**  **{**  **"Effect": "Allow",**  **"Action": [**  **"iam:CreateServiceLinkedRole",**  **"iam:DeleteServiceLinkedRole",**  **"iam:ListRoles",**  **"organizations:DescribeOrganization"**  **],**  **"Resource": "\*"**  **}**  **]**  **}** |
|  | Copy iot-policy.json to your Rpi, and then run the following command  aws iam put-role-policy --role-name my-iot-role --policy-name iot-policy --policy-document file://iot-policy.json |

# AWS IoT Rules Part 1

In the previous section, you successfully sent messages from your Raspberry Pi to your AWS account. However, it is not much use if you keep sending the messages to the cloud without storing them or processing them.

What real-world applications do, after receiving messages in the cloud is to act on the messages received, either by channelling them to permanent storage like a relational database or to perform some real-time analysis on them.

In this section, you will learn how to use a special feature known as **AWS IoT Rules** to take information from an incoming MQTT message and write it to a AWS Simple Notification Service (SNS) topic which thens sends email to you. Have fun! 😊



## Create an AWS SNS Topic

| No | Task | | | | |
| --- | --- | --- | --- | --- | --- |
|  | Open the Amazon SNS console (<https://ap-southeast-1.console.aws.amazon.com/sns/v2>) and choose “Get Started” | | |  | |
|  | Choose Topics | | | |  |
|  | Choose Create new topic |  | | | |
|  | Type a topic name and a display name, and then choose Create topic. | | | | |
|  | Make a note of the ARN for the topic you just created. | | | | |
|  | Due to the limitations of the AWSEducate account, you need to edit the policy of the rule to allow “Everyone” to publish | |  | | |
|  | | | | | |



## Subscribe to an Amazon SNS Topic

In this section, you will learn how to create an Amzon SNS topic which allows you to receive emails whenever IoT data is published at the IoT Hub you configured earlier.

| No | Task |
| --- | --- |
|  | In the Amazon SNS console, select the check box next to the topic you just created. From the Actions menu, choose Subscribe to topic. |
|  | On Create subscription, from the Protocol drop-down list, choose SMS.  In the Endpoint field, type your email address and then choose Create subscription. |
|  | You will receive an email that confirms you successfully created the subscription. Click on the link to confirm subscription. |
|  |  |



## Create an AWS IoT Rule

The AWS IoT rules engine listens for incoming MQTT messages that match a rule. When a matching message is received, the rule takes some action with the data in the MQTT message. In this step, you will create and configure a rule to send the data received from a device to an Amazon SNS topi

| No | Task | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
|  | In the AWS IoT console, in the left navigation pane, choose Act, then “Create” | | | | |  |
|  | On the Create a rule page, in the Name field, type a name for your rule. In the Description field, type a description for the rule. | | |  | | |
|  | Scroll down to Message source. Choose the latest version from the Using SQL version drop-down list. In the Attribute field, type \*. This specifies that you want to send the entire MQTT message that triggered the rule. |  | | | | |
|  | The rules engine uses the topic filter to determine which rules to trigger when an MQTT message is received.  In the Topic filter field, type “sensors/rfid”  Leave the condition blank |  | | | | |
|  | In Set one or more actions, choose Add action. | | | |  | |
|  | On the Select an action page, select Send a message as an SNS push notification, and then choose Configure action. | |  | | | |
|  | On the Configure action page, from the SNS target drop-down list, choose the Amazon SNS topic you created earlier. | |  | | | |
|  | Now you need to give AWS IoT permission to publish to the Amazon SNS topic on your behalf when the rule is triggered.  Choose Create a new role. Enter a name for your new role in the IAM role name field. | | | | | |
|  | After you have entered the name, choose Add Action and then Create Rule. You should be redirected and you should see the new Rule you just created | | | | | |
|  | On the Overview page for the rule, choose the left arrow to return to the AWS IoT dashboard. | | | | | |

# Code the IoT app

In this section, you will write a simple Python program that reads the values from your MCP3008 ADC and send it to AWS.



## Alarm Program

| No | Task |
| --- | --- |
|  | Create a file named “alarm.py” with the following code   |  | | --- | | aws\_pubsub.py | | import time  import sys  import signal  import telepot  from random import randint  from blynkapi import Blynk  from time import sleep  from datetime import datetime  from buzzer import buzz  from rpi\_lcd import LCD  import RPi.GPIO as GPIO  def random():  ops = ['+', '-']  number1 = randint(1, 3)  number2 = randint(1, 3)  operatornumber = randint(0,2)  operation = ""  answer = 0  if operatornumber == 0:  operation = '+'  answer = number1 + number2  question = (str(number1) + ' + ' + str(number2))  return question, answer  if operatornumber == 1:  operation = '-'  answer = number1 - number2  question = (str(number1) + ' - ' + str(number2))  return question, answer    def alarmON(alarm, question, answer):  awake = 0  off = False  attempt = 0  try:  # Loop to continuously check time, buzz the buzzer for the set alarm time  while off == False:  # Continually get's the time as an integer value  curr\_time = int(time.strftime("%H%M"))  pressed = GPIO.input(16)  add = GPIO.input(27)  minus = GPIO.input(22)  if add == True:  attempt = attempt + 1  print(attempt)  if minus == True:  attempt = attempt - 1  print(attempt)  if pressed == True:  if attempt == answer:  print('alarm is off')  break  # Buzzes the buzzer when the time reaches the set alarm time  if curr\_time == alarm:  buzz(10,0.5)  time.sleep(0.25)  buzz(20,0.5)  time.sleep(0.25)  awake = 1  lcd.text(question,1)  display = 'Answer is now ' + str(attempt)  lcd.text(display, 2)    # If alarm continues past the set alarm time without being  # snoozed, the alarm time is changed to the current time.  # This ensures the alarm buzzes continuously until the  # snooze button is pressed.  elif curr\_time != alarm and awake == 1:  alarm = curr\_time  buzz(10,0.5)  time.sleep(0.25)  buzz(20,0.5)  time.sleep(3)  lcd.text(question,1)  display = 'Answer is now ' + str(attempt)  lcd.text(display, 2)  sleep(3)  finally:  print('off')  def setAlarm(msg):  print('listens in tele')  chat\_id = msg['chat']['id']  alarm = msg['text']    curr\_time = int(time.strftime("%H%M"))  try:  intalarm = int(alarm)  except:  bot.sendMessage(chat\_id, 'Not a valid time, please enter one in HHMM format')  if curr\_time >= intalarm:  question,answer = random()  bot.sendMessage(chat\_id, ('Alarm set for tomorrorw @ {}'.format(alarm)))  buzz(500,0.1)  print(question)  alarmON(intalarm, question, answer)    elif intalarm > curr\_time:  question, answer = random()  bot.sendMessage(chat\_id, ('Alarm set for today @ {}'.format(alarm)))  buzz(500,0.1)  print(question)  alarmON(intalarm, question, answer)  GPIO.setmode(GPIO.BCM)  GPIO.setwarnings(False)  GPIO.setup(16, GPIO.IN)  GPIO.setup(27, GPIO.IN)  GPIO.setup(22, GPIO.IN)  lcd = LCD()  global needquestion  global question  question = ''  global answer  answer = 0  needquestion = 1  my\_bot\_token = "685042644:AAEBhZl5I75pIHLd2Hxgm91-SZfXDLNf6kI"  bot = telepot.Bot(my\_bot\_token)  bot.message\_loop(setAlarm)  while True:  sleep(1) | |

## RFID Program

| No | Task |
| --- | --- |
|  | Create a file named “rfidreader.py” with the following code   |  | | --- | | rfidreader.py | | #!/usr/bin/env python  # -\*- coding: utf8 -\*-  import RPi.GPIO as GPIO  import MFRC522  import signal  import time  import sys  import Adafruit\_DHT  import datetime  import MySQLdb  import gspread  import telepot  from time import sleep  from gpiozero import MCP3008  from gpiozero import Buzzer  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  from oauth2client.service\_account import ServiceAccountCredentials  from rpi\_lcd import LCD  # Custom MQTT message callback  def customCallback(client, userdata, message):  print("Received a new message: ")  print(message.payload)  print("from topic: ")  print(message.topic)  print("--------------\n\n")  # Capture SIGINT for cleanup when the script is aborted  def end\_read(signal,frame):  global continue\_reading  print ("Ctrl+C captured, ending read.")  continue\_reading = False  GPIO.cleanup()  def Brain(msg):  print('listens in tele')  chat\_id = msg['chat']['id']  command = msg['text']  if command == 'Who':  try:  db = MySQLdb.connect(host="localhost", user="assignment", passwd="assignment", db="assignmentdatabase")  curs = db.cursor()  queryall = ("SELECT name FROM assignmentdatabase.users WHERE ishome = 1;")  curs.execute(queryall)  y = curs.fetchall()  newstring = [i[0] for i in y]  homemessage = 'Users who are home are: \n'  if y:  for names in newstring:  homemessage = (homemessage + names + '\n')  print(names)  if not y:  homemessage = 'Nobody is home'  bot.sendMessage(chat\_id, homemessage)  except:  print('error')  if command == 'All':  try:  db = MySQLdb.connect(host="localhost", user="assignment", passwd="assignment", db="assignmentdatabase")  curs = db.cursor()  queryall = ("SELECT name FROM assignmentdatabase.users;")  curs.execute(queryall)  y = curs.fetchall()  newstring = [i[0] for i in y]  registereduserstring = 'All Registered Users are \n'  if y:  for names in newstring:  registereduserstring = (registereduserstring + names + '\n')  print(names)  bot.sendMessage(chat\_id, registereduserstring)  except:  print('error')  if command.startswith('New'):  command, Name = command.split()  print(command)  print(Name)  bot.sendMessage(chat\_id, 'Ready to recieve RFID Card')  while continue\_reading:  MIFAREReader = MFRC522.MFRC522()  # Scan for cards  (status,TagType) = MIFAREReader.MFRC522\_Request(MIFAREReader.PICC\_REQIDL)    # If a card is found  if status == MIFAREReader.MI\_OK:  bot.sendMessage(chat\_id, "Card Detected")    # Get the UID of the card  (status,uid) = MIFAREReader.MFRC522\_Anticoll()    # If we have the UID, continue  if status == MIFAREReader.MI\_OK:    # Select the scanned tag  rawcardtype = MIFAREReader.MFRC522\_SelectTag(uid)  try:  print('1')  db = MySQLdb.connect(host="localhost", user="assignment", passwd="assignment", db="assignmentdatabase")  curs = db.cursor()  print('2')  print(str(uid))  print(Name)  insert = ("INSERT INTO assignmentdatabase.users(name, rfid, ishome) VALUES ('%s','%s','0');" % (Name, str(uid)))  print(insert)  curs.execute(insert)  print('3')  db.commit()  db.close()  bot.sendMessage(chat\_id, "User successfully registered")  sleep(5)  break  except:  bot.sendMessage(chat\_id, "Error Encountered")  print('Connection to db error')  db.close()  sleep(5)  if command.startswith('Delete'):  command, Name = command.split()  print(command)  print(Name)  try:  db = MySQLdb.connect(host="localhost", user="assignment", passwd="assignment", db="assignmentdatabase")  curs = db.cursor()  print(str(uid))  delete = ("DELETE FROM assignmentdatabase.users WHERE name = '%s';" % Name)  printdelete()  curs.execute(delete)  db.commit()  db.close()  bot.sendMessage(chat\_id, "User successfully deleted")  except:  bot.sendMessage(chat\_id, "Error Encountered")  print('Connection to db error')  db.close()  if command == 'Enter':  #This loop keeps checking for chips. If one is near it will get the UID and authenticate  bot.sendMessage(chat\_id, 'Ready to recieve RFID Card')  bot.sendMessage(chat\_id, 'Welcome Jonathan!')  GPIO.cleanup()  GPIO.setmode(GPIO.BCM)  GPIO.setwarnings(False)  GPIO.setup(24,GPIO.OUT)  # Turns turn on Buzzer  GPIO.output(24, GPIO.HIGH)  sleep(1)  # Turns off the Buzzer  GPIO.output(24, GPIO.LOW)  #Generates and displays message to LCD to user is validated  y = 'jonathan'  message = ('Welcome, ' + y)  publish = (y + ' has come home')  lcd.text('Access Granted', 1)  lcd.text(message, 2)  my\_rpi.publish('sensors/rfid', publish, 1)    GPIO.cleanup()    while continue\_reading:  GPIO.cleanup()  MIFAREReader = MFRC522.MFRC522()  # Scan for cards  (status,TagType) = MIFAREReader.MFRC522\_Request(MIFAREReader.PICC\_REQIDL)    # If a card is found  if status == MIFAREReader.MI\_OK:  print ("Card detected")    # Get the UID of the card  (status,uid) = MIFAREReader.MFRC522\_Anticoll()    # If we have the UID, continue  if status == MIFAREReader.MI\_OK:    # Select the scanned tag  rawcardtype = MIFAREReader.MFRC522\_SelectTag(uid)  # Check if authenticated  try:  db = MySQLdb.connect(host="localhost", user="assignment", passwd="assignment", db="assignmentdatabase")  curs = db.cursor()  try:  #Checks for both if the User is not home and if the User is valid  queryhome = ("SELECT name, ishome FROM assignmentdatabase.users WHERE ishome = 0 AND rfid = '%s';" % str(uid))  curs.execute(queryhome)  #Fetches the values from the queryhome query  values = curs.fetchall()  #Only if user is not home and user is valid, this if statement will be carried out  if values:  try:  # To get the name of the user  queryhome = ("SELECT name FROM assignmentdatabase.users WHERE ishome = 0 AND rfid = '%s';" % str(uid))  curs.execute(queryhome)  y = curs.fetchone()[0]  # To update phpmyAdmin database value to set ishome value to 1  updatehome = ("UPDATE users SET ishome = 1 WHERE rfid = '%s';" % str(uid))  print(updatehome)  curs.execute(updatehome)  bot.sendMessage(chat\_id,('Welcome ' + y))  #Cleans up and sets the GPIO board to allow buzzer to be turned on using GPIO24  GPIO.cleanup()  GPIO.setmode(GPIO.BCM)  GPIO.setwarnings(False)  GPIO.setup(24,GPIO.OUT)  # Turns turn on Buzzer  GPIO.output(24, GPIO.HIGH)  sleep(1)  # Turns off the Buzzer  GPIO.output(24, GPIO.LOW)  #Generates and displays message to LCD to user is validated  message = ('Welcome, ' + y)  publish = (y + ' has come home')  lcd.text('Access Granted', 1)  lcd.text(message, 2)  my\_rpi.publish('sensors/rfid', publish, 1)    GPIO.cleanup()  #Saves changes made to db, must be here if not it won't save  db.commit()  sleep(5)  break  except:  db.close()  sleep(5)  #If user is not a valid user or if user is already home, this will display error message on LCD  if not values:  bot.sendMessage(chat\_id, 'Warning, Access Denied')  lcd.text('Warning',1)  lcd.text('Access Denied',2)  except:  db.close()  sleep(5)  except:  print("Error connecting to mySQL database")  db.close()  sleep(5)  # For the RFID to continuously scan for any tags nearby  continue\_reading = True  # Sets up the LCD display  lcd = LCD()  # For the LED light to set channel to 0  mcp3008 = MCP3008(channel=0)  # Create an object of the class MFRC522  MIFAREReader = MFRC522.MFRC522()  # How long to wait (in seconds) between measurements.  FREQUENCY\_SECONDS = 10  # Hook the SIGINT  signal.signal(signal.SIGINT, end\_read)  # Welcome message  print ("Welcome to the MFRC522 data read example")  print ("Press Ctrl-C to stop.")  #Amazon AWS configuration and connections  host = "ac9yfl5owv3oq.iot.ap-southeast-1.amazonaws.com"  rootCAPath = "rootca.pem"  certificatePath = "certificate.pem.crt"  privateKeyPath = "private.pem.key"  my\_rpi = AWSIoTMQTTClient("basicPubSub")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing  my\_rpi.configureDrainingFrequency(2) # Draining: 2 Hz  my\_rpi.configureConnectDisconnectTimeout(10) # 10 sec  my\_rpi.configureMQTTOperationTimeout(5) # 5 sec  # Connect and subscribe to AWS IoT  my\_rpi.connect()  my\_rpi.subscribe("sensors/rfid", 1, customCallback)  sleep(2)  my\_bot\_token = "694518265:AAFRVupRGQuRh67\_MBcA1xSyfQKx18xaULo"  bot = telepot.Bot(my\_bot\_token)  bot.message\_loop(Brain)  while True:  sleep(1) | |

## CCTV Node-Red Program

| No | Task |
| --- | --- |
| a) | Open a Terminal window and start Node-RED on your RPi  node-red start |
| b) |  |
| c) | Next , open the Node-RED webpage on your laptop’s browser  The URL would be the IP address of your Raspberry Pi, followed by the 1880 port e.g. <http://localhost:1880> |
| d) | Navigate to the Import> Clipboard and paste in the following code |
|  |  |

## Temperature Reading Node-Red Program

| No | Task |
| --- | --- |
|  | Create a file named “aws\_telepot\_rfid.py” with the following code   |  | | --- | | aws\_pubsub.py | | # Import SDK packages  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  from time import sleep  from gpiozero import MCP3008  adc = MCP3008(channel=0)  # Custom MQTT message callback  def customCallback(client, userdata, message):  print("Received a new message: ")  print(message.payload)  print("from topic: ")  print(message.topic)  print("--------------\n\n")    host = "YourEndPoint.amazonaws.com"  rootCAPath = "rootca.pem"  certificatePath = "certificate.pem.crt"  privateKeyPath = "private.pem.key"  my\_rpi = AWSIoTMQTTClient("basicPubSub")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing  my\_rpi.configureDrainingFrequency(2) # Draining: 2 Hz  my\_rpi.configureConnectDisconnectTimeout(10) # 10 sec  my\_rpi.configureMQTTOperationTimeout(5) # 5 sec  # Connect and subscribe to AWS IoT  my\_rpi.connect()  my\_rpi.subscribe("sensors/light", 1, customCallback)  sleep(2)  # Publish to the same topic in a loop forever  loopCount = 0  while True:  light = round(1024-(adc.value\*1024))  my\_rpi.publish("sensors/light", str(light), 1)  sleep(5) | |

# 12. Using the IoT device in real world

12.1 Application

This IoT device can be integrated into any home so as to make everyday house appliances more automated and even in a sense, much smarter. The SmartHome System also provides a much higher level of security as the keys to the house are replaced with RFID Scanner and Tags and there is also a surveillance camera system installed into this IoT device. The surveillance camera was primarily created to detect any suspicious intrusion or monitor any suspicious activity while the RFID Scanner and Tags were created to provide a much higher level of security by replacing normal everyday locks and key. From the telepot that is connected to the RFID Scanner, users are also able to keep track of user activities and check who is at home as well.

In addition, with the camera in place, the user can observe and look at the surroundings or the suspicious activity taking place. The camera also comes with a record function, which enables the user to record the current live stream, so that the user can view it later or store it as evidence. Apart from that, the user can also see who presses the doorbell so that in the event of dnager, the user can choose not to open the door.

Not to mention, there is also a DHT11 Temperature and Humidity Sensor and it is used to automate the turning on and off of air conditioners in the household so that the house can be cooled down once it gets too hot. But the users are also able to switch it off from the web interface if they wish to do so as well without the need of a physical remote.

In conclusion, this IoT device can be integrated into any household to make normal everyday household appliances much more connected and integrated in the house and even in a sense, much smarter.