SOMS CA2 (Part Two) Step-by-step Tutorial

Table of Contents

Section 1
Overview of project

A. What is the application about?

Smart Office Management System (SOMS) is a Node.js web application that is used to manage certain processes required by people in an office environment. The web application interfaces with the Raspberry Pi 3 and various modules connected to it to carry out its functionalities.

SOMS is an application to manage and track employees' daily attendance, monitor and control the lights and camera in the office.

For CA2, we will be extending from CA1 to include the functionality to allow users to track the number of people in a particular room. We will be utilising a lightweight communications protocol called Message Queuing Telemetry Transport (MQTT) to publish and subscribe messages to and from certain topics from the AWS IoT broker.

Imagine that you want to know the number of people present in two rooms, T2031 (Main Office) and T2032 (Meeting Room 1).

This is achieved by users tapping their NFC cards on the RFID reader located at the entry and/or exit doors of a room. Once a NFC card is detected, the Raspberry Pi publishes to the topic 'rooms/t2031' based on whether if its an entry or exit. Other Raspberry Pi which subscribes to the topics will receive the message to update the data.

The value of the number of people in the affected room will be incremented by 1 when a person enters a room. Conversely, the value will decrement if a person exits a room. These data will be stored in Amazon DynamoDB, which is a fast and scalable NoSQL cloud database.

You can deploy the web application on a EC2 instance on the Ubuntu Server 14.04 LTS.

One additional advanced feature we have implemented is the usage of Telegram chatbots to complement with this functionality.

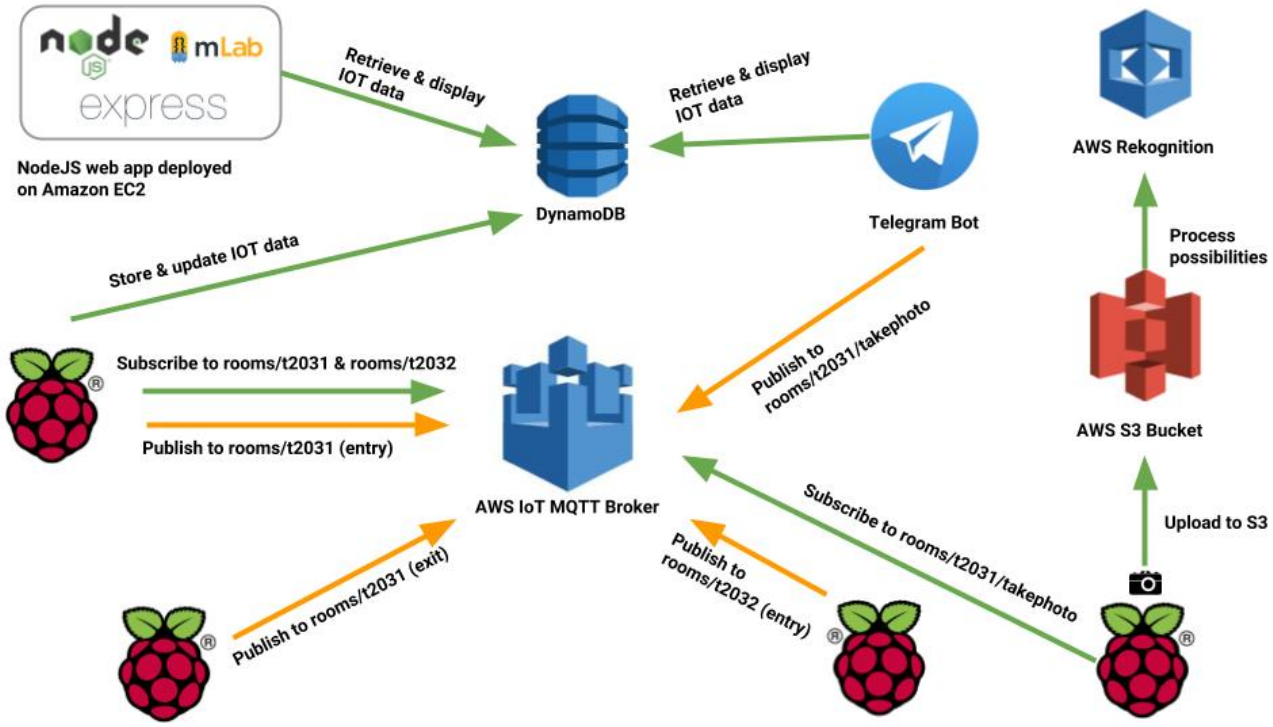
With the chatbot, users can easily view the number of people in these two rooms and capture image from T2031 and process it using Amazon Rekognition to detect faces or objects.

The following table shows a list of commands available.

Operation	Command
List all commands	help
Take photo in T2031	take photo
Get number of people in T2031	t2031
Get number of people in T2032	t2032

B. System Architecture

The following diagram illustrates the System Architecture.



The AWS IoT MQTT Broker publishes messages to all the Raspberry Pi that are subscribed to the topics in it. At the same time, the Raspberry Pis itself can publish their IoT data to the topics and it will be received by others. All IoT data from CA2 is stored on DynamoDB, a free NoSQL cloud service provided by Amazon.

The SOMS web application consists of Node.js, Express, and a free NoSQL cloud database called mLab. All the data that was previously stored in the local MongoDB on the Raspberry Pi has now been migrated to mLab on the cloud.

This is because storing large amounts of data on a local database is not feasible and it cannot be accessed by other applications publicly. Lastly, the entire application itself is hosted on Amazon EC2 instance, on the Ubuntu 14.04.04 LTS so that the application is accessible to the public using http protocol. The rooms page retrieves the IoT data that is stored on DynamoDB.

The Telegram chatbot communicates with the DynamoDB to retrieve IoT data. The chatbot itselfalso publishes to rooms/t2031/takephoto. The raspberrry pi that is listening to that topic will receive the message, trigger the PiCam to take an photo. After taking the photo, it stores the photo on AWS S3 Bucket. Subsequently, the photo is sent to AWS Rekognition service to detect for faces or objects. The results are returned to the Telegram chatbot to display the image and recognition results.

C. Technology Stack

The following describes that technology stack that we have used in our project.

The main bulk of the SOMS application is written using Mongo, Express.js and Node.js, also denoted as MEN stack. In addition, we also used python as well for the MQTT communications.

D. Cloud Services

We have also leveraged on a handful of useful cloud services in order to build our application's functionalities:

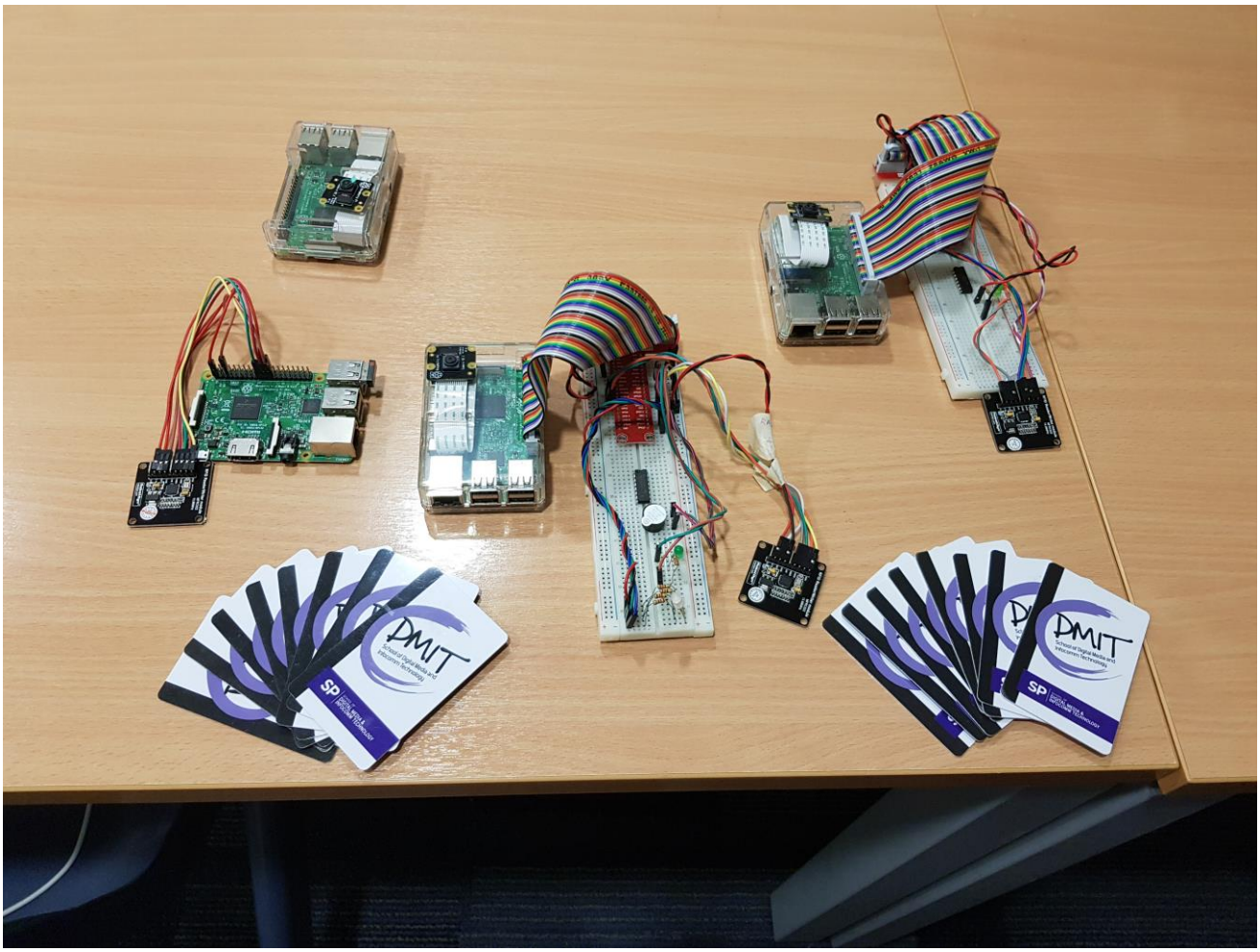
- AWS IoT MQTT Broker
- AWS Rekognition
- AWS S3 Bucket
- DynamoDB
- mLab

E. Summary of the steps that will be described

We will be covering the following in this documentation.

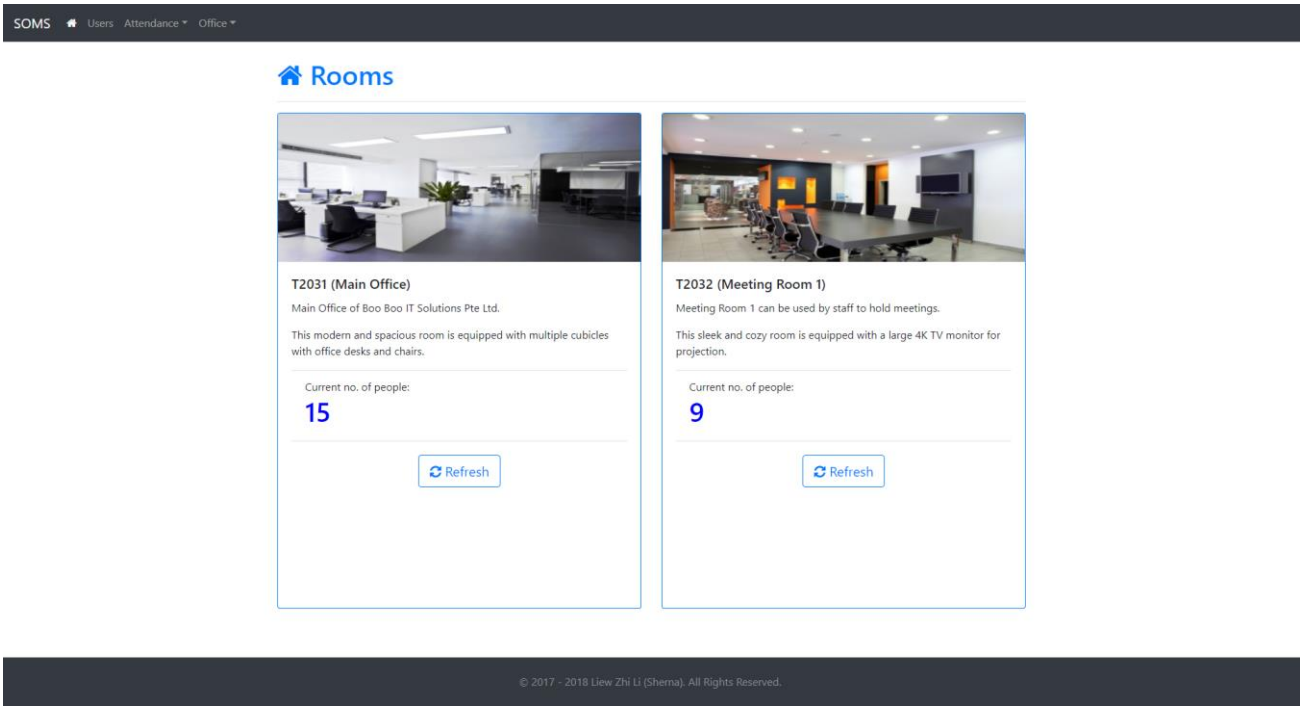
	Section	Description
1)	Overview	Provides an overview of an enhanced version of SOMS, the system architecture, technology stack and cloud services used and how the RPI set up and web and mobile application looks like.
Sections 2 to 8 provides the step-by-step instructions to set up the application		
2)	Hardware requirements	Provides overview of hardware required
3)	AWS Setup Tutorial	Provides a comprehensive step-by-step tutorial on how to setup AWS for the project
4)	Installation of required libraries	Provides instructions on installing the required python libraries on the Raspberry Pi
5)	Coding pub-sub program	Provides instructions on how to code the Node.js and Python MQTT publish-subscribe programs
6)	Coding Telegram Bot	Provides instructions on how to code the Telegram Bot and deploy it on an EC2 instance
7)	Coding Raspberry Pi Camera	Provides instructions on how to code to trigger PiCam to capture image and send to AWS Rekognition to get results
8)	Deploy SOMS on Amazon EC2	Provides instructions on how to deploy the main web application SOMS on a Amazon EC2 instance

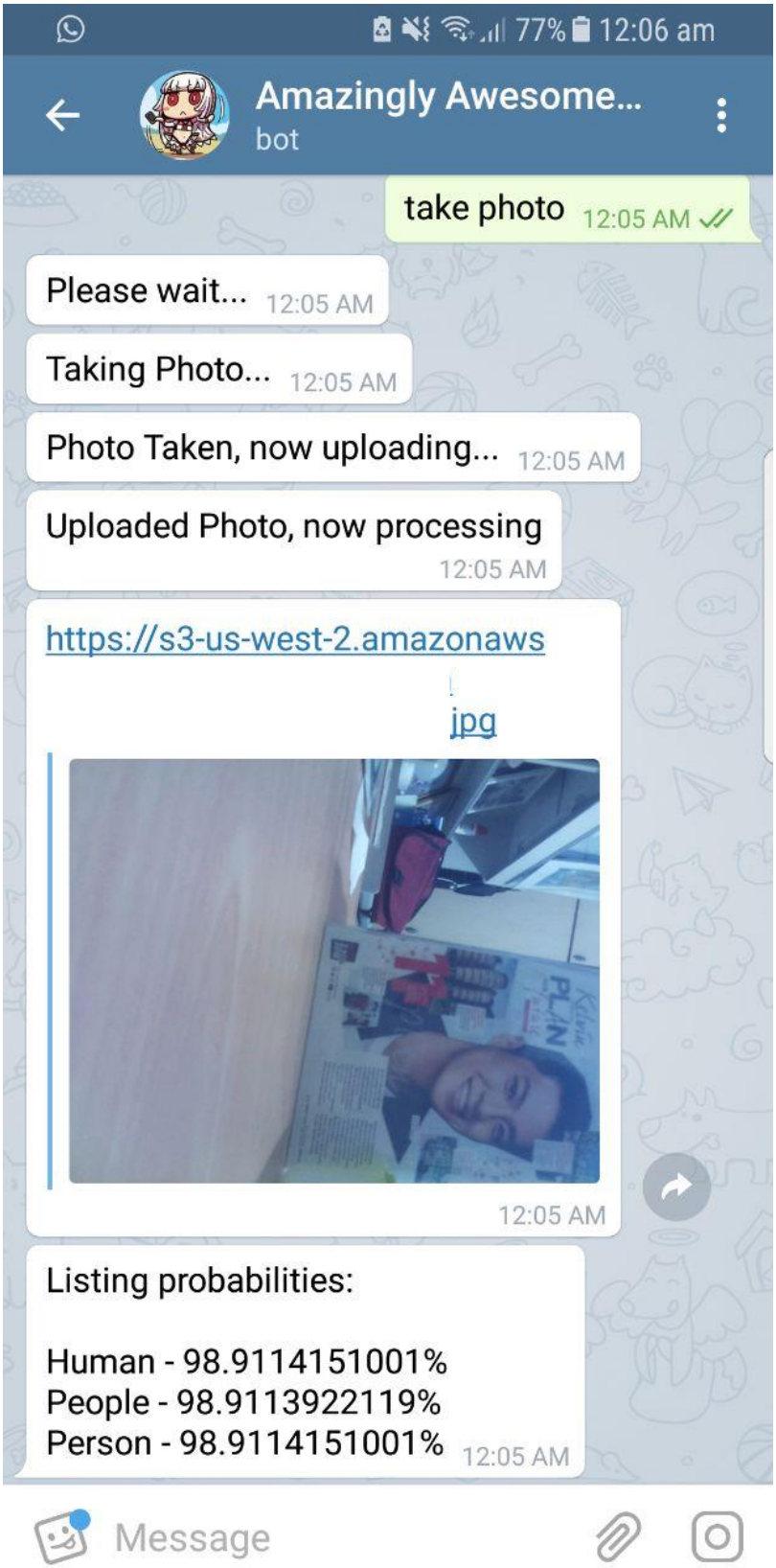
F. How does the final RPI set-up looks like?

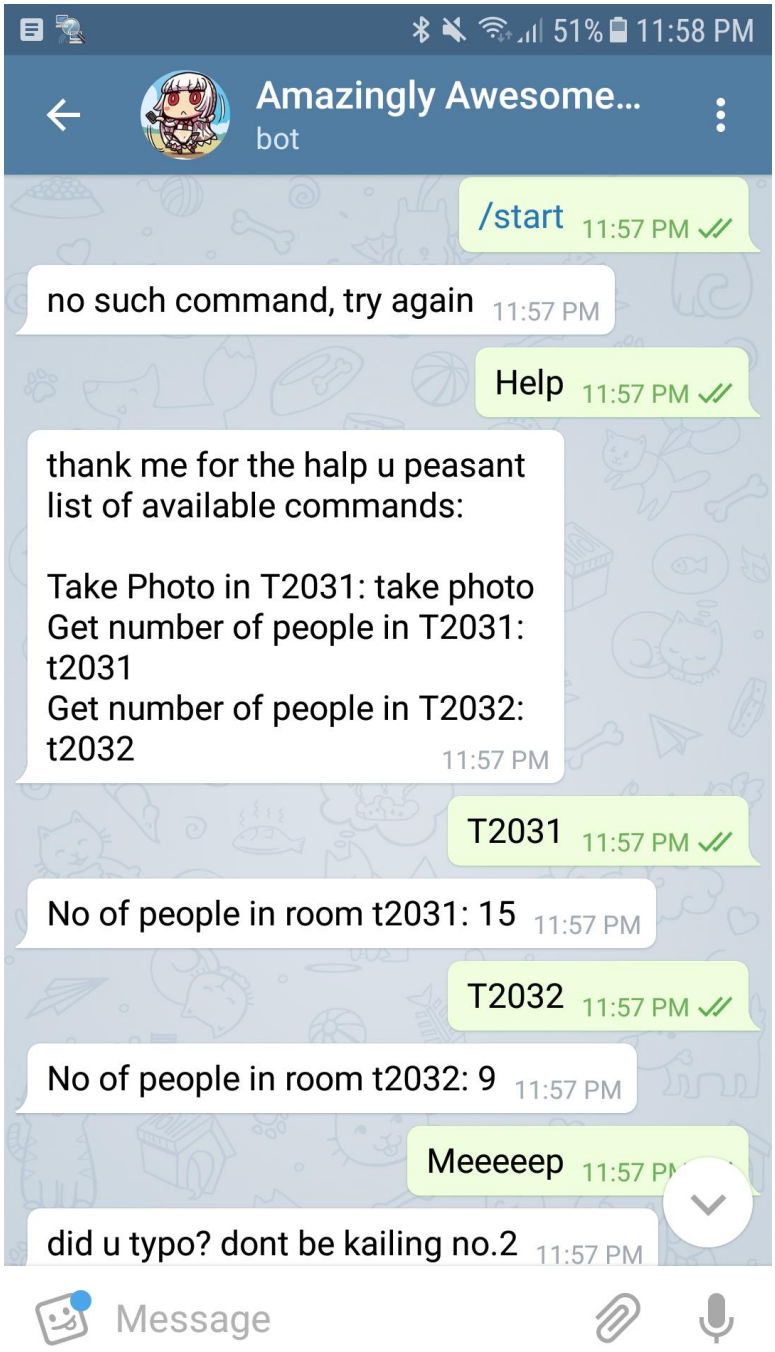


G. How does the web and mobile application look like?

The Rooms page (accessed by going to Office > Rooms in the navbar) displays the number of people in T2031 and T2032.




77% 12:06 am
Amazingly Awesome...
bot
take photo 12:05 AM
Please wait... 12:05 AM
Taking Photo... 12:05 AM
Photo Taken, now uploading... 12:05 AM
Uploaded Photo, now processing
12:05 AM
https://s3-us-west-2.amazonaws
jpg
12:05 AM
Listing probabilities:
Human - 98.9114151001%
People - 98.9113922119%
Person - 98.9114151001%
12:05 AM
Message


51% 11:58 PM
Amazingly Awesome...
bot
/start 11:57 PM
no such command, try again 11:57 PM
Help 11:57 PM
thank me for the halp u peasant
list of available commands:
Take Photo in T2031: take photo
Get number of people in T2031: t2031
Get number of people in T2032: t2032
11:57 PM
T2031 11:57 PM
No of people in room t2031: 15 11:57 PM
T2032 11:57 PM
No of people in room t2032: 9 11:57 PM
Meeeeep 11:57 PM
did u typo? dont be kailing no.2 11:57 PM
Message


51% 11:58 PM
Amazingly Awesome...
bot
No of people in room t2032: 9 11:57 PM
Meeeeep 11:57 PM
did u typo? dont be kailing no.2 11:57 PM
Meep 11:57 PM
did u typo? dont be kailing no.2 11:57 PM
Meeeeep 11:57 PM
no such command, try again 11:57 PM
Mrup 11:57 PM
no such command, try again 11:58 PM
Md 11:58 PM
no such command, try again 11:58 PM
Dj 11:58 PM
crying meep no such command, dont bully me T_T 11:58 PM
Message

Section 2
Hardware requirements

Hardware checklist

The following table shows the hardware required to replicate my project.

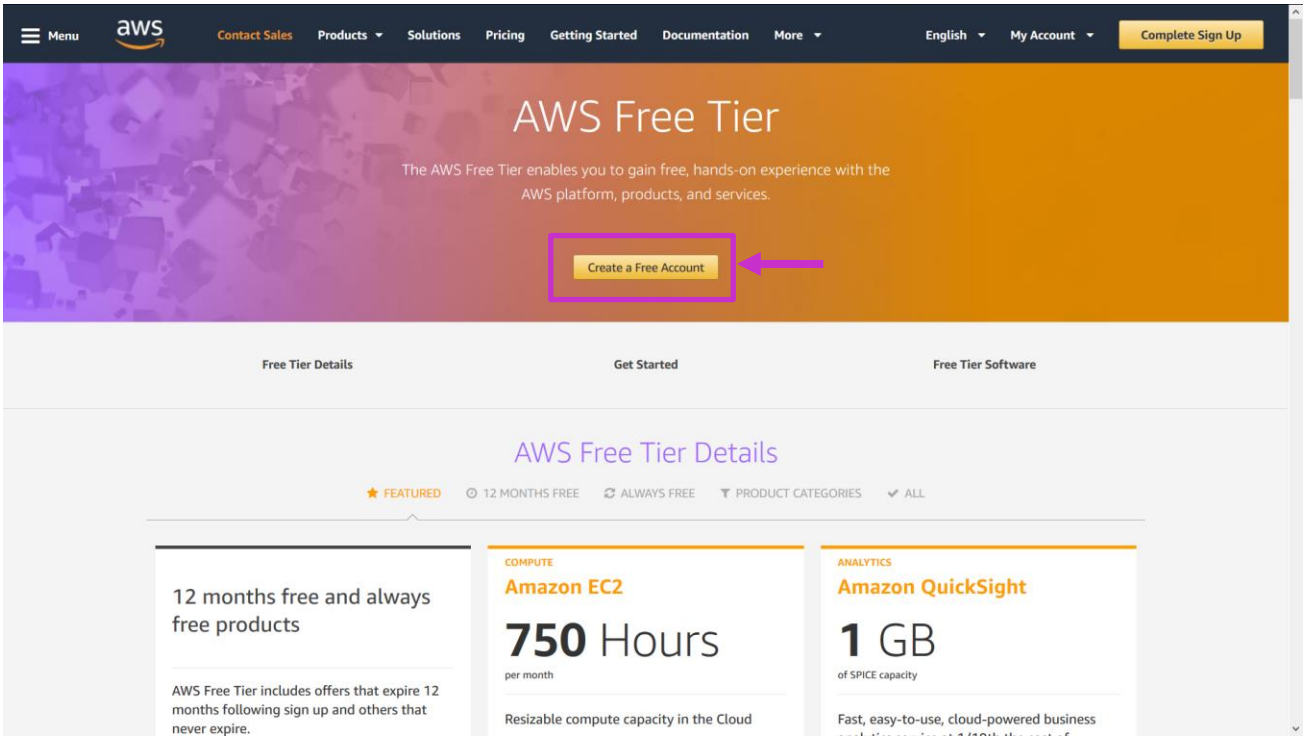
No.	Hardware Module	Model Name	Quantity	Price/unit ($)	Where to buy?
1	Raspberry Pi 3	Model B	3	52.21	Element14
2	PiCam	Version 2	1	37.29	Element14
2	Active Buzzer		1	1.08	Banggood
3	Common Anode RGB LED		1	0.06	AliExpress
4	Any colour LED		3	0.02	AliExpress
5	RFID Reader	MFRC522	3	5.43	AliExpress
6	Full-length Breadboard		2	2.24	AliExpress
7	Half-length Breadboard		1	1.23	AliExpress
8	Jumper Wires (M-to-M, M-to-F, F-to-F)		Up to you	4.70	AliExpress
9	220Ω Resistors		5	0.02	AliExpress

Section 3
AWS Setup Tutorial

A. Create a Amazon AWS Account

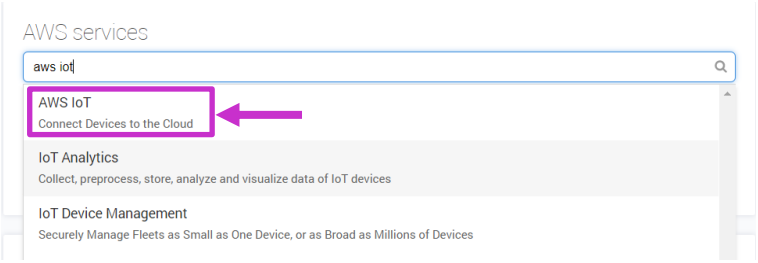
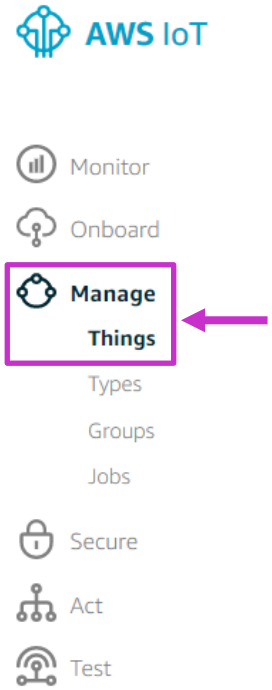
You can register for a Amazon AWS account using its Free Tier here.
Click on “Get Started” to register your account.

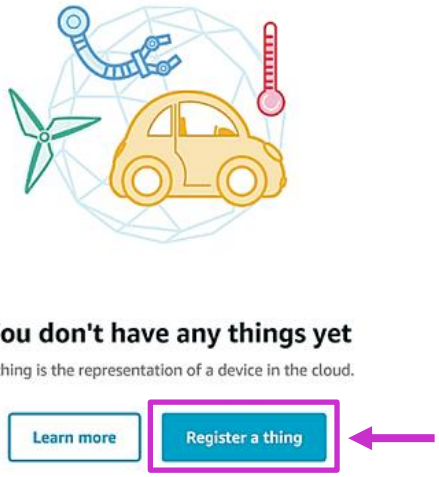
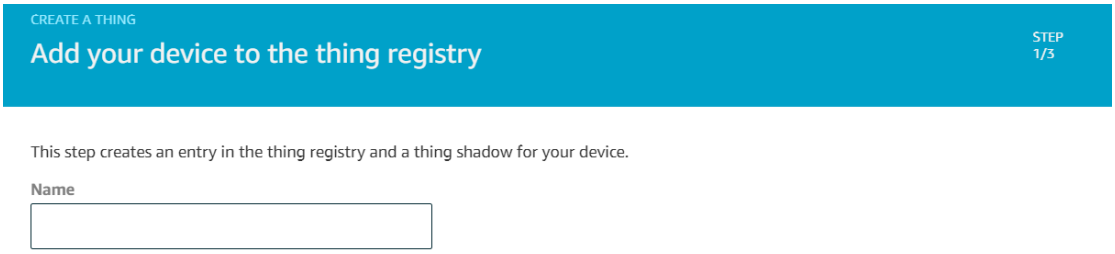
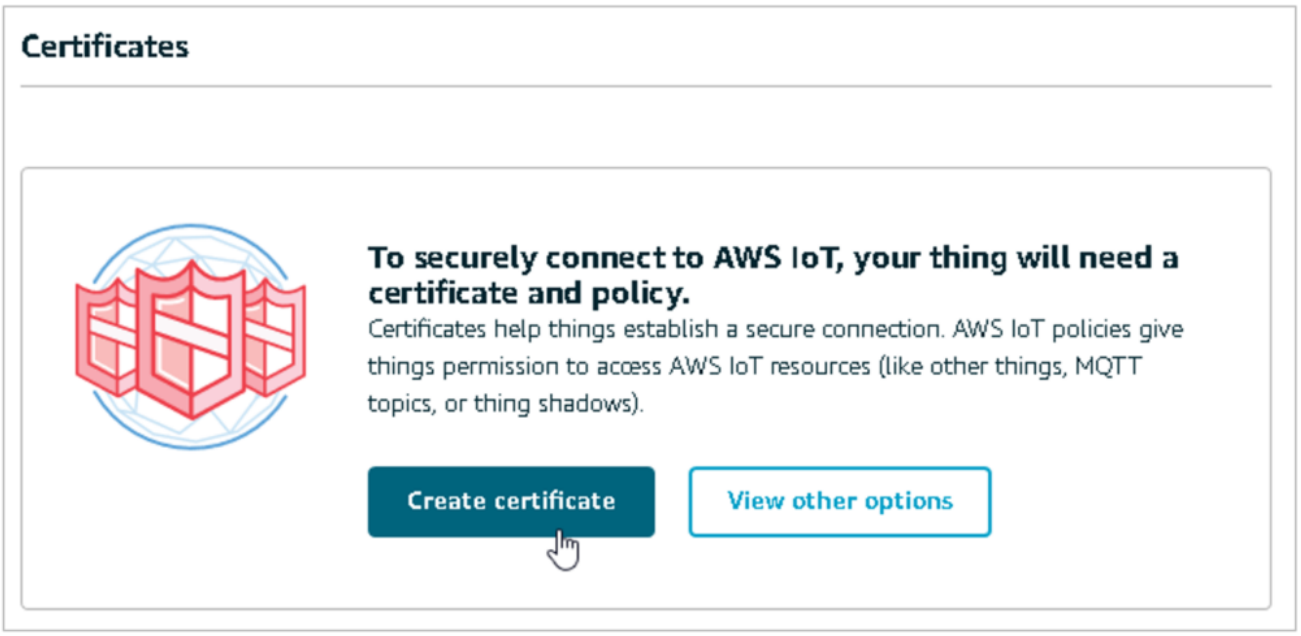
Note that you will need to input your credit card and billing address. As long as your usage remains in the Free Tier, you will not be charged.



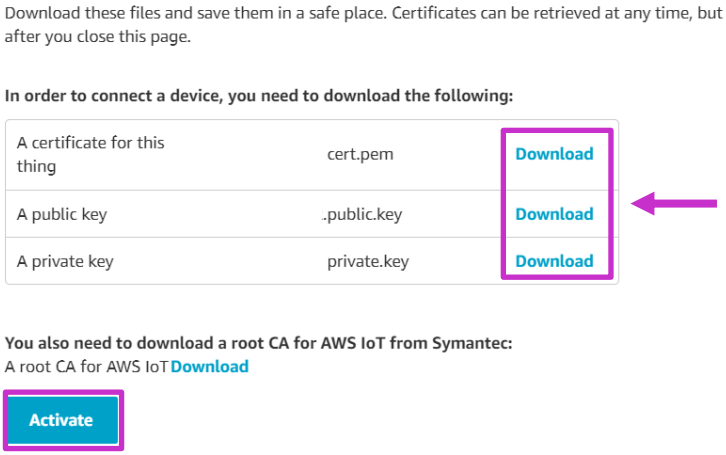
Once you have completed your registration, sign in to your AWS account here.

B. Create a Raspberry Pi as a Thing

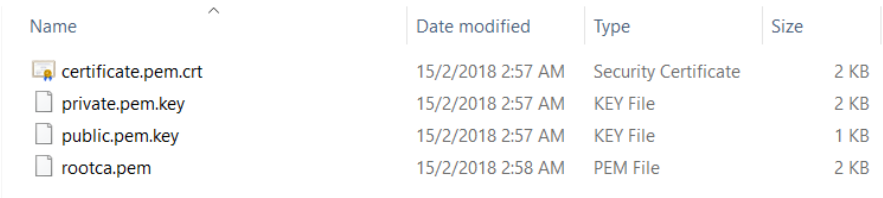
No	Task
1	Sign in with your AWS account at https://aws.amazon.com
2	In the main AWS Dashboard, search for "AWS IoT" and click to access the AWS IoT service. 
3	In this page, click "Get Started". 
4	On your left, you will see a navigation menu for AWS IoT. Go to Manage > Thing. 

5	In this page, click “Register a thing”. 
6	In AWS IoT, a thing represents either a device or a logical entity. For example, physical device or sensors such as LED lights, or logical entity like an application instance. Our “thing” here is our RPi, so you an choose any name you want. Click “Create thing”. 
7	Next, click “Create certificate” to generate an X.509 certificate and key pair. 

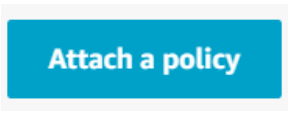
9 After a while, you should see the following screen, where there are a total of four download links.
At your desktop, create a folder named “AWS-IOT-files”.
Click “Download” for the 3 items in the table to the folder you just created.
Next, click “Activate” button at the bottom, then click “Download” for the root CA.



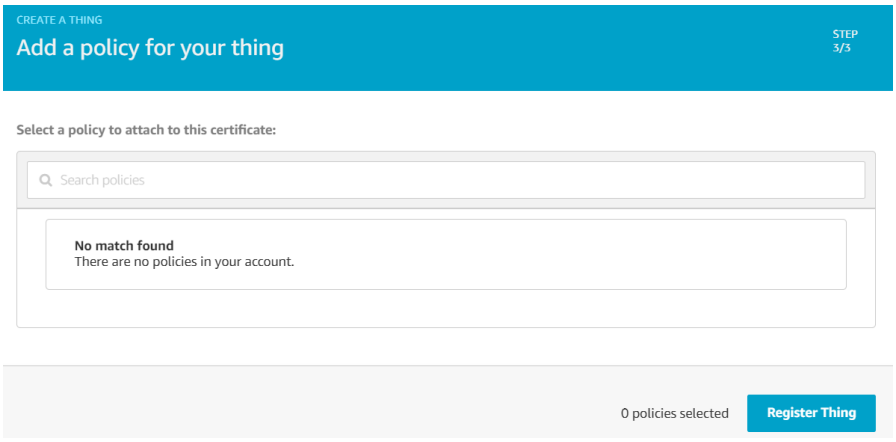
10 Rename the files to a more friendly name for usage later.



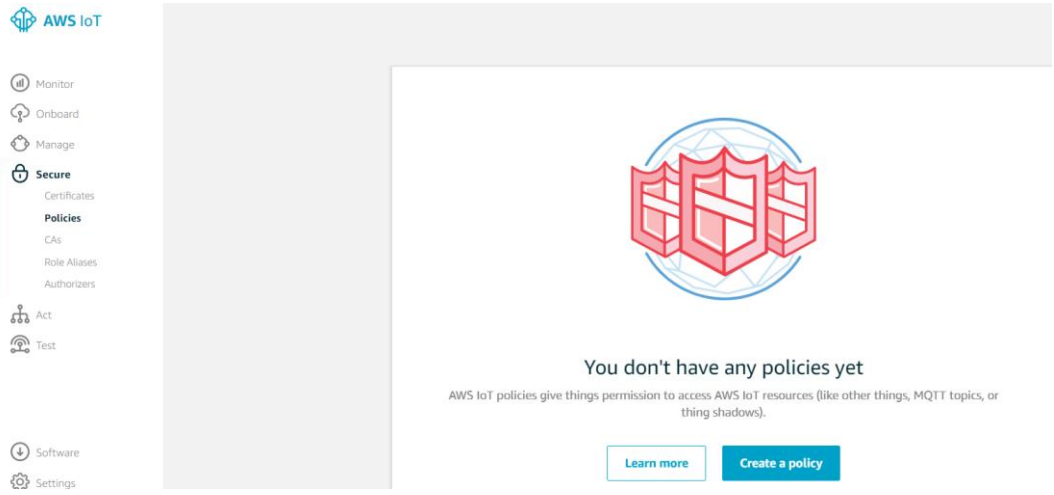
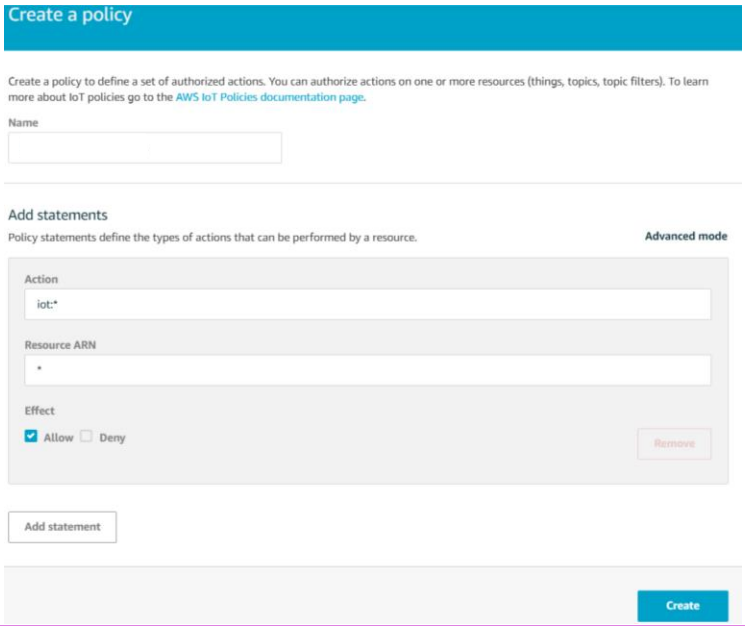
11 Click “Attach Policy” at the bottom of the page.



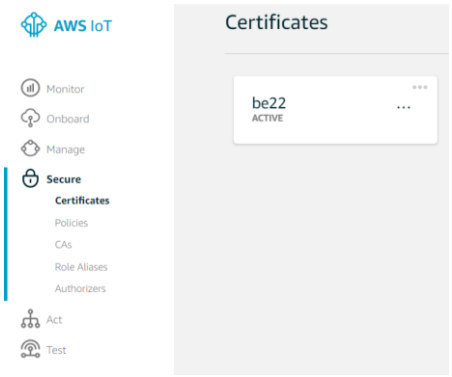
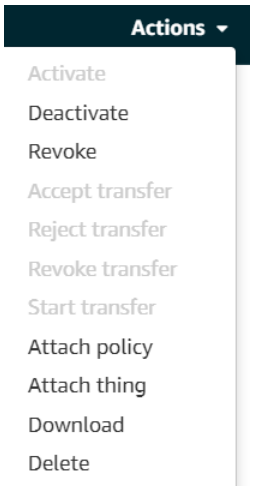
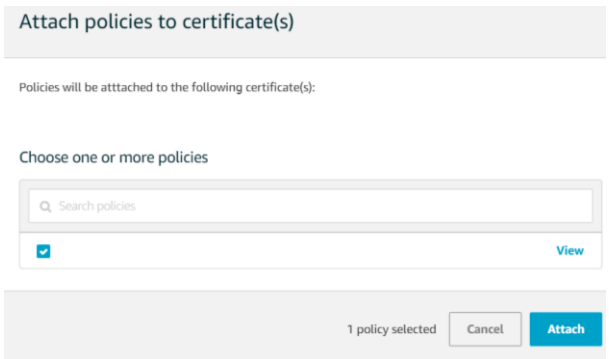
12 Click “Register Thing” to complete the registration as we do not have a policy yet.



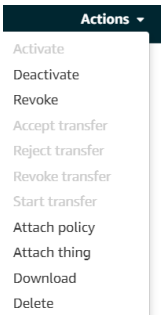
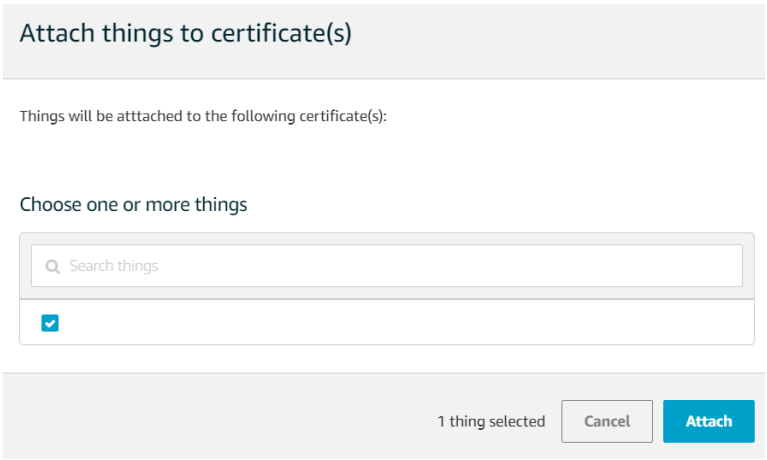
C. Create a Security Policy for your Thing

No	Task
1	Go to your AWS IoT Dashboard. Go to Securities > Policies and click “Create a Policy”. 
2	Fill in the following information and click “Create”. Name: <any name you want> Action: iot:* Resource ARN: * Effect: Allow Click “Create” at the bottom of the page. 

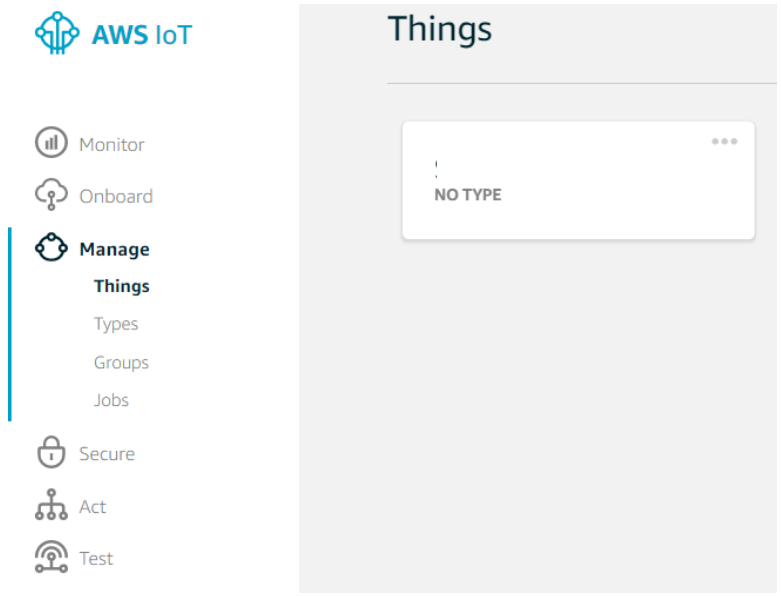
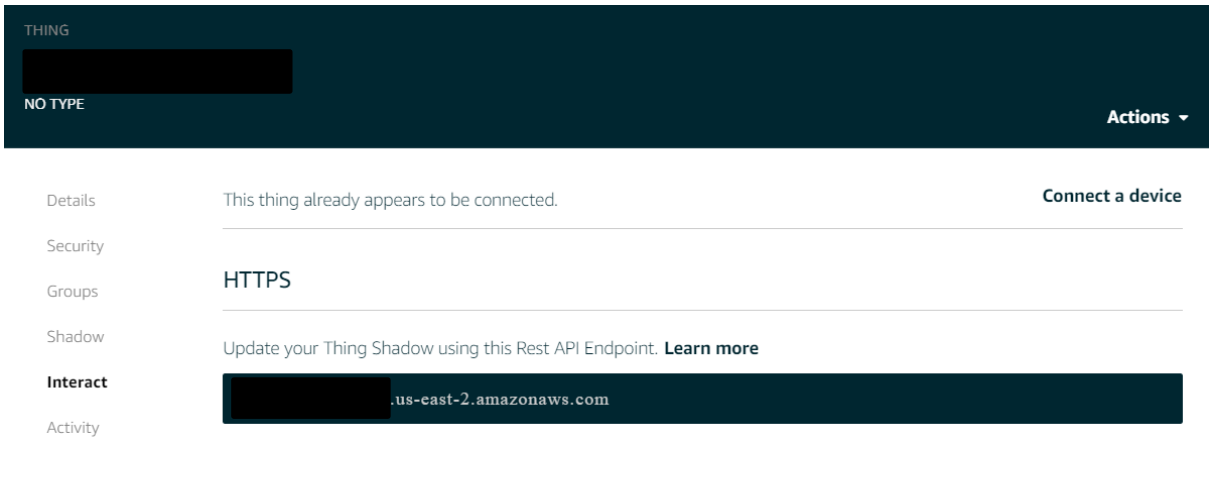
D. Attach policy to Certificate

No	Task
1	Once you click “Create”, you will be brought back to your Dashboard. Go to Secure > Certificates Click on the box that says “be22…” 
2	Click on “Actions” Click on “Attach policy” 
3	Check the policy you created earlier and click “Attach” button. 

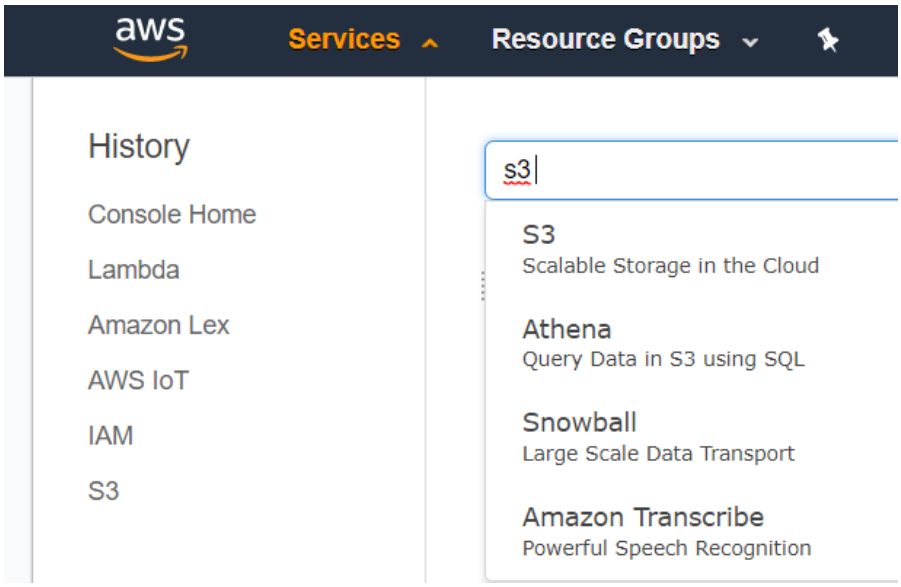
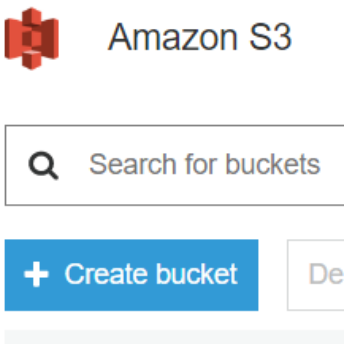
E. Attach Thing to Certificate

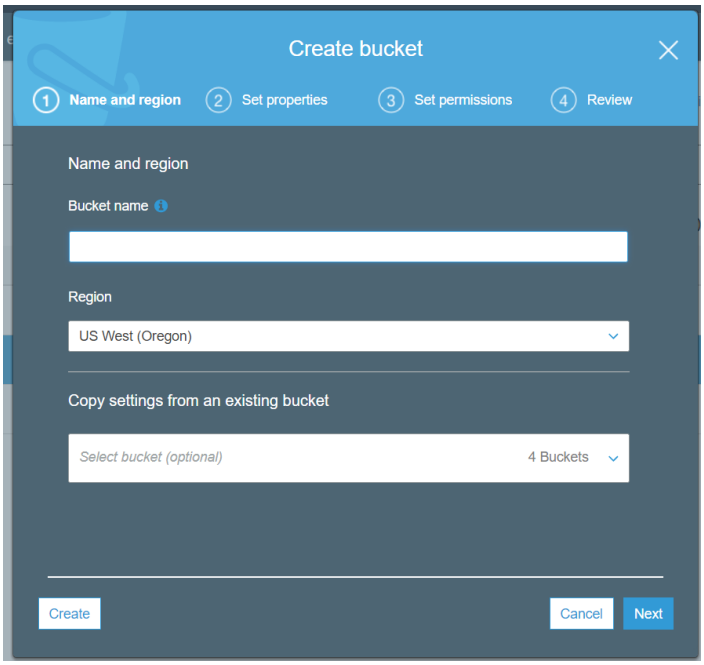
No	Task
1	Click on the “Actions” and “Attach Thing”. 
2	Click on the thing you created and click “Attach”. 

F. Get your REST API Endpoint

No	Task
1	Click on Manage > Things Click on your thing 
2	Go to Interact tab Copy and paste the REST API endpoint into a Notepad. You will need this value later. Endpoint: xxx.us-east-2.amazonaws.com 

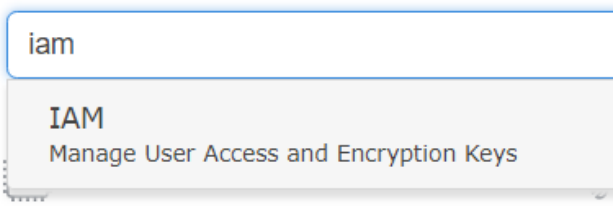
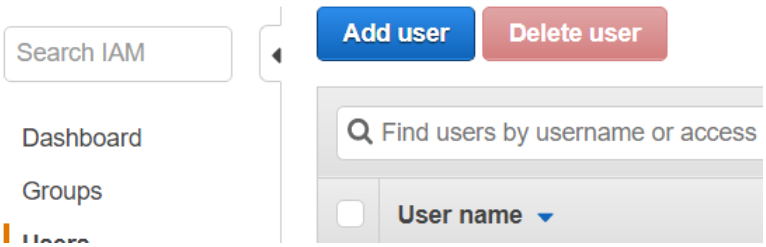
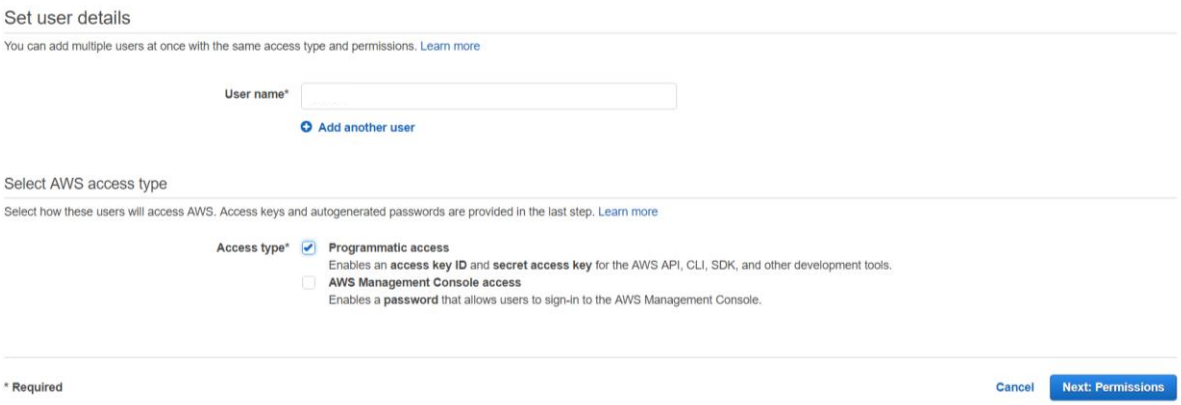
G. Create Bucket on AWS S3

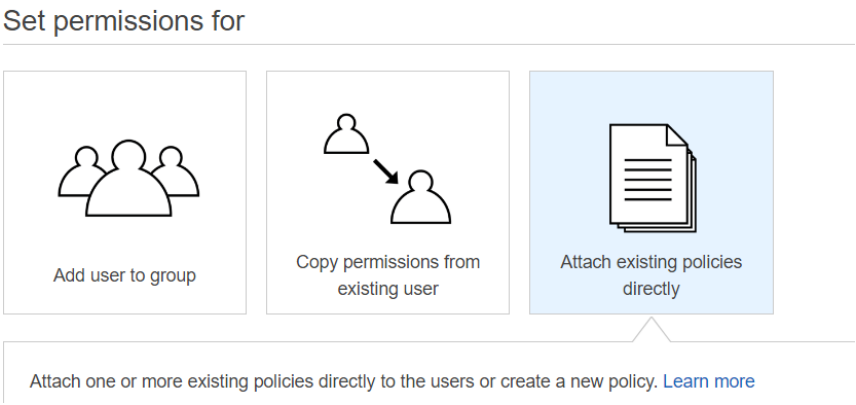
No	Task
1	Click on service and search for S3 
2	Create bucket 

3	Type in a unique name for your bucket and choose Region as “US East (Ohio)” which is us-west-2  Click “Create” button
4	After a while, your newly created bucket should appear in the list. 

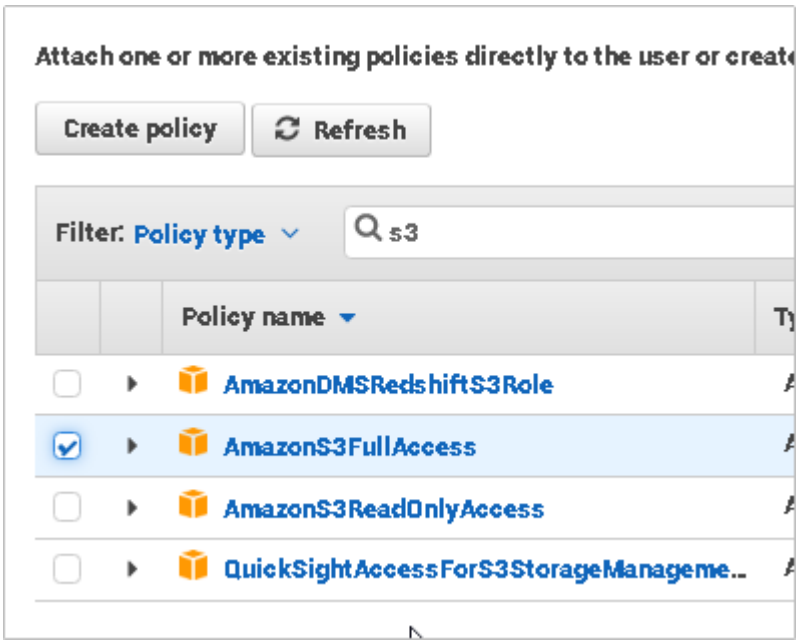
H. Setup Authentication Credentials

In order to be able to use API calls to write to Amazon S3 buckets, you will need to set up credentials for it.

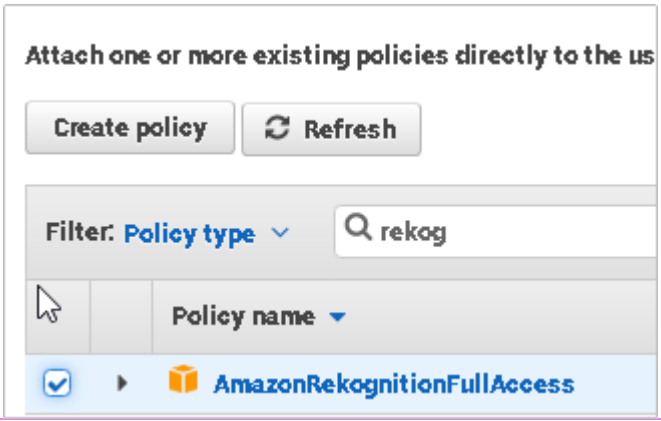
No	Task
1	Click on service and search for IAM 
2	Click on “Users” submenu, “Add user” 
3	Create a new user **(any name)** and enable **Programmatic access**  Click next
4	Click “Attach existing policies directly”

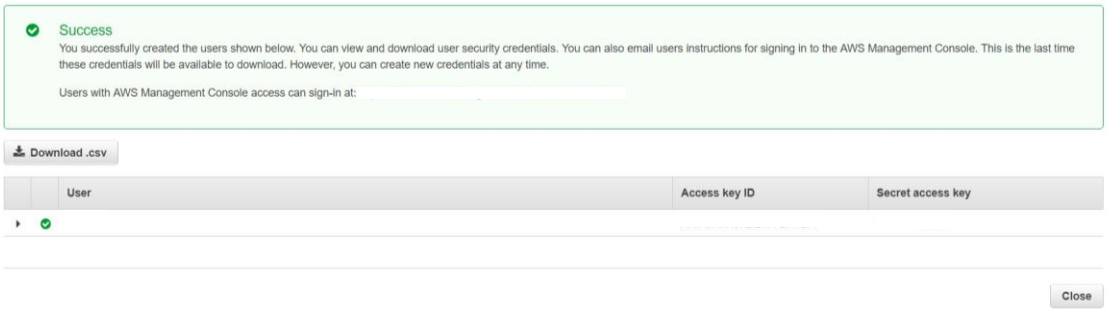


5 Scroll to the bottom of the page, search for “s3” and then check the “AmazonS3FullAccess” option



6 Next, search for “Rekognition” and then check the “AmazonRekognitionFullAccess” option



7	Click the “Next: Review” button at the bottom of the page on the right, then on the next page, click “Create user” 
8	 You should see the success page above with a link to download a csv file. **Make sure you download the .csv file to your laptop** You will need to refer to them later.

Section 4
Installation of required libraries

After we have successfully set up the AWS Cloud configurations, we need to install the required libraries to use the AWS Python SDK.

A. Install required libraries in Terminal

Execute the following command in Terminal to install AWS Python SDK.

```
sudo  pip  install  AWSIoTPythonSDK
```

Paho MQTT is a dependency of the AWS Python SDK, thus we need to install it.
Execute the following command in Terminal to install Paho MQTT.

```
Sudo pip install paho-mqtt
```

Execute the following command in Terminal to install Boto, the Python library for AWS

```
sudo pip install boto3
```

Execute the following command in Terminal to install the AWS Command-Line Interface Client

```
sudo pip install awscli
```

Execute the following command in Terminal to install the Telegram API

```
sudo pip install telepot
```

That's all! 😊

Section 5
Coding pub-sub program

A. Node.js & Python Pub-Sub program

This is where it gets exciting! We will code a Node.js program to connect to MQTT, detect and read from the RFID reader, publish and subscribe to the 2 topics: rooms/t2031 and rooms/t2032. Server.js will be running on the main RPI.

Server.js

```
var express = require('express')
var bodyParser = require('body-parser')
var app = express()
var http = require('http').Server(app)
var io = require('socket.io')(http)
var ip = require("ip")
var mongoose = require('mongoose')
var rc522 = require("rc522")
var rpio = require('rpio')
var awsIot = require('aws-iot-device-sdk');
var AWS = require('aws-sdk');
mongoose.Promise = Promise

//
// Register Node.js middleware
// -------------------------------------------------------------------------
app.disable('etag'); // Resolve HTTP status of 304 Modified due to caching
https://stackoverflow.com/questions/18811286/nodejs-express-cache-and-304-status-
code
app.use(express.static('../front_end'))
app.use(bodyParser.json())
app.use(bodyParser.urlencoded({ extended: false }))

var dbUrl = '<insert your mlab url endpoint>'

//
// Import routes
// -------------------------------------------------------------------------
var user = require('./routes/user');
var light = require('./routes/light');
var attendance = require('./routes/attendance');
var mqtt = require('./routes/mqtt');

//
// Use routes
// -------------------------------------------------------------------------
```

```
app.use('/api', user);
app.use('/api', light);
app.use('/api', attendance);
app.use('/api', mqtt);

// --- Database Connection to MongoDB ---
mongoose.connect(dbUrl, { useMongoClient: true }, (err) => {
    console.log('MongoDB connection err: ', err)
})

io.on('connection', (socket) => {
    console.log('a user connected')
    // emit ip address
    io.sockets.emit("ip", ip.address())
})

// ------------------------------------------------------------------------
// AWS IOT MQTT CONNECTIONS - START ---------------------------------------
// Configuration
var device = awsIot.device({
    keyPath: '<insert your private key path>',
    certPath: '<insert your cert path>',
    caPath: '<insert your root ca path>',
    clientId: '<insert  your client id>',
    host: '<insert your aws host>'
});


// Device is an instance returned by mqtt.Client(), see mqtt.js for full
// documentation.

device
    .on('connect', function () {
        // Subscribe to both rooms
        device.subscribe('rooms/t2031', 1);
        device.subscribe('rooms/t2032', 1);
        console.log('MQTT: Subscribed to rooms/t2031!');
        console.log('MQTT: Subscribed to rooms/t2032!');
    });

device
    .on('message', function (topic, payload) {
        console.log("NEWM MESSAGE COMING IN!")
        console.log('message', topic, payload.toString());

        var payload = JSON.parse(payload.toString());

        // show the incoming message
        if (topic === 'rooms/t2031') {
```

```
            if (payload.isEnter === "true") {
                // get record from dynamodb & increment & update value
                getData("t2031", true);
                buzz_rfid();
                console.log("1 user has entered T2031!");
            } else {
                // get record from dynamodb & decrement & update value
                getData("t2031", false);
                buzzer_ring();
                console.log("1 user has exited T2031!");
            }
        }else if (topic === 'rooms/t2032'){ // T2032
            console.log("t2032")
            console.log(payload.isEnter)
            // get record from dynamodb & increment & update value
            getData("t2032", true);
            buzz_rfid();
            console.log("1 user has entered T2032!");


        }
    });
// AWS IOT MQTT CONNECTIONS - END  ----------------------------------------
// ------------------------------------------------------------------------

// ------------------------------------------------------------------------
// AWS SDK TO COMMUNICATE WITH DYNAMODB - START ---------------------------
AWS.config.update({
    region: "<insert your region>",
    // The endpoint should point to the local or remote computer where DynamoDB
(downloadable) is running.
    endpoint: "<insert your endpoint>",
    /*
    accessKeyId and secretAccessKey defaults can be used while using the
downloadable version of DynamoDB.
    For security reasons, do not store AWS Credentials in your files. Use Amazon
Cognito instead.
    */
    accessKeyId: '<insert your access key>',
    secretAccessKey: '<insert your secret access key>'
});

var dynamodb = new AWS.DynamoDB({ apiVersion: '2012-08-10', correctClockSkew:
true });
var docClient = new AWS.DynamoDB.DocumentClient();
var tablename = "<insert your table name>";

// Function to retrieve the data from table in DynamoDB
```

```
function getData(room, isEnter) {
    console.log("Retrieving record from table in DynamoDB!")

    var params = {
        TableName: tablename,
        Key: {
            "room": {
                S: room
            }
        }
    };

    dynamodb.getItem(params, function (err, data) {
        if (err) console.log(err, err.stack); // an error occurred
        else {
            // Need to parse string to integer
            var currentNoOfPpl = parseInt(data.Item.entered.N);
            console.log("Old value of no. of ppl in room: " + currentNoOfPpl)

            // If isEnter, then increment. Else, decrement.
            if (isEnter){
                currentNoOfPpl += 1;
            }else{
                currentNoOfPpl -= 1;
            }

            console.log("New value of no. of ppl in room: " + currentNoOfPpl);

            // Update new value of no. of ppl in room in database
            var params = {
                TableName: tablename,
                Item: {
                    "room": room,
                    "entered": currentNoOfPpl,
                }
            };

            docClient.put(params, function (err, data) {
                if (err) console.log(err, err.stack); // an error occurred
                else console.log("Successfully updated new value!");           //
successful response
            });

        }
    });
}

// AWS SDK TO COMMUNICATE WITH DYNAMODB - END -----------------------------
```

```
// --------------------------------------------------------------------------

rc522(function (rfidSerialNumber) {
    io.sockets.emit("rfid", rfidSerialNumber); // Sends the RFID Serial Number
through Socket.IO
    buzz_rfid();
    console.log(rfidSerialNumber);

    // Publisher #1 : Entry
    // Publish using AWS IOT
    device.publish('rooms/t2031', JSON.stringify({ 'isEnter': 'true' }));
    console.log('MQTT: Published entry to rooms/t2031!');
});

/*
 * Init device (Buzzer) on Pin 12 (GPIO18)
 * Init device (Green LED) on Pin 18 (GPIO24)
 * Set the initial state to low.  The state is set prior to the pin becoming
 * active, so is safe for devices which require a stable setup.
 */
var buzzer_pin = 12;
var led_pin = 18;
rpio.open(buzzer_pin, rpio.OUTPUT, rpio.LOW);
rpio.open(led_pin, rpio.OUTPUT, rpio.LOW);

// Function that beeps the buzzer & lit the green LED for 0.5 seconds
function buzzer_ring() {
    console.log("buzz")
    /* On buzzer for 50ms */
    rpio.write(buzzer_pin, rpio.HIGH);
    rpio.msleep(100);

    /* Off both devices for 50ms */
    rpio.write(buzzer_pin, rpio.LOW);
    rpio.msleep(100);

    rpio.msleep(250);

    rpio.write(buzzer_pin, rpio.HIGH);
    rpio.msleep(150);

    /* Off both devices for 500ms */
    rpio.write(buzzer_pin, rpio.LOW);
    rpio.msleep(150);


}

// Function that beeps the buzzer & lit the green LED in a cute pattern
```

```
function buzz_rfid() {
    for (x = 0; x <= 5; x++) {
        /* On both devices for half of a half of a second (50ms) */
        rpio.write(buzzer_pin, rpio.HIGH);
        rpio.write(led_pin, rpio.HIGH);
        rpio.msleep(50);

        /* Off both devices for half of a half of a second (50ms) */
        rpio.write(buzzer_pin, rpio.LOW);
        rpio.write(led_pin, rpio.LOW);
        rpio.msleep(50);
    }
}

var server = http.listen(3000, () => {
    console.log('NodeJS server is running on ' + ip.address() + ':' +
server.address().port)
    console.log(io.sockets.name)
})
```

Now, lets create the Rooms.html page to retrieve data from DynamoDB and display them!

Rooms.html

```
<!doctype HTML>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">
    <meta name="description" content="IoT CA1 Assignment">
    <meta name="author" content="Sherna Liew">
    <title>SOMS</title>
    <!-- CSS -->
    <link href="node_modules/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet">
    <link href="colours.css" rel="stylesheet">
    <link href="starter-template.css" rel="stylesheet">
    <link href="node_modules/font-awesome/css/font-awesome.min.css"
rel="stylesheet">
    <link href="node_modules/bootstrap-toggle/css/bootstrap-toggle.css"
rel="stylesheet">
    <link href="https://cdn.datatables.net/1.10.16/css/jquery.dataTables.min.css"
rel="stylesheet">
    <link href="node_modules/sweetalert2/dist/sweetalert2.min.css" rel="stylesheet">
    <!-- Javascript libraries -->
    <script src="node_modules/jquery/dist/jquery.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.13.0/umd/popper.js"></script
>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/tether/1.4.0/js/tether.min.js"
integrity="sha384-DztdAPBWPRXSA/3eYEEUWrWCy7G5KFbe8fFjk5JAIxUYHKkDx6Qin1DkWx51bBrb"
        crossorigin="anonymous"></script>
    <script src="node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="node_modules/bootstrap/dist/js/bootstrap.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.20.1/moment.min.js"></script
>
    <script src="node_modules/bootstrap-toggle/js/bootstrap-toggle.min.js"></script>
    <script src="node_modules/sweetalert2/dist/sweetalert2.all.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/core-
js/2.4.1/core.js"></script>
    <script
src="https://cdn.datatables.net/1.10.16/js/jquery.dataTables.min.js"></script>
    <script src="/socket.io/socket.io.js"></script>
    <script src="node_modules/whatwg-fetch/fetch.js"></script>
    <script src="https://sdk.amazonaws.com/js/aws-sdk-2.7.16.min.js"></script>
```

```
</head>

<body>
    <nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
        <a class="navbar-brand" href="/">SOMS</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarsExampleDefault" aria-controls="navbarsExampleDefault"
            aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse" id="navbarsExampleDefault">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item active">
                    <a class="nav-link" href="/">
                        <i class="fa fa-home" aria-hidden="true"></i>
                        <span class="sr-only">(current)</span>
                    </a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="user_management.html">Users</a>
                </li>
                <li class="nav-item dropdown">
                    <a class="nav-link dropdown-toggle" id="dropdown01" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false">Attendance</a>
                    <div class="dropdown-menu" aria-labelledby="dropdown01">
                        <a class="dropdown-item"
href="take_attendance.html">Attendance Taking</a>
                        <a class="dropdown-item" href="attendance.html">Attendance
Log</a>
                    </div>
                </li>
                <li class="nav-item dropdown">
                    <a class="nav-link dropdown-toggle" id="dropdown01" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false">Office</a>
                    <div class="dropdown-menu" aria-labelledby="dropdown01">
                        <a class="dropdown-item" href="lights.html">Lights</a>
                        <a class="dropdown-item" href="cctv.html">CCTV</a>
                        <a class="dropdown-item" href="rooms.html">Rooms</a>
                    </div>
                </li>
            </ul>
        </div>
    </nav>

    <main role="main" class="container">
        <h1 class="text-primary">
            <i class="fa fa-home" aria-hidden="true"></i> Rooms</h1>
```

```
        <hr />
    </main>

    <div class="container">
        <div class="card-deck">
            <!-- Rooms T2031-->
            <div class="card border-primary">
                <img class="card-img-top" src="http://aymdesign.com.sg/wp-
content/uploads/2016/02/Feature-Image-1024x683.png" alt="Card image cap"
                    height="30%">
                <div class="card-body">
                    <h5 class="card-title">T2031 (Main Office)</h5>
                    <p class="card-text">Main Office of Boo Boo IT Solutions Pte
Ltd.</p>
                    <p class="card-text">This modern and spacious room is equipped
with multiple cubicles with office desks and chairs.</p>
                    <ul class="list-group list-group-flush">
                        <li class="list-group-item">Current no. of people:
                            <h1 id="t2031People" style="color: rgb(0,0,255);">0</h1>
                        </li>
                    </ul>
                    <div class="card-body text-center">
                        <button id="refreshT2031" type="button" class="btn btn-
outline-primary btn-lg">
                            <i class="fa fa-refresh" aria-
hidden="true"></i> Refresh</button>
                    </div>
                </div>
            </div>
            <!-- Rooms T2032-->
            <div class="card border-primary">
                <img class="card-img-top" src="http://www.spandanindia.com/blog/wp-
content/uploads/2016/01/Modern-Office-Interior-Designers-Surat-624x395.jpg"
                    alt="Card image cap" height="30%">
                <div class="card-body">
                    <h5 class="card-title">T2032 (Meeting Room 1)</h5>
                    <p class="card-text">Meeting Room 1 can be used by staff to hold
meetings.</p>
                    <p class="card-text">This sleek and cozy room is equipped with a
large 4K TV monitor for projection.</p>

                    <ul class="list-group list-group-flush">
                        <li class="list-group-item">Current no. of people:
                            <h1 id="t2032People" style="color: rgb(0,0,255);">0</h1>
                        </li>
                    </ul>
                    <div class="card-body text-center">
```

```
                        <button id="refreshT2032" type="button" class="btn btn-
outline-primary btn-lg">
                            <i class="fa fa-refresh" aria-
hidden="true"></i> Refresh</button>
                    </div>
                </div>
            </div>

        </div>
        <br />
        <br />
        <br />
    </div>

    <!-- End of container -->
    <footer class="footer bg-dark text-center">
        <div class="container">
            <br />
            <span class="text-muted">&copy; 2017 - 2018 Liew Zhi Li (Sherna). All
Rights Reserved.</span>
            <br />
            <br />
        </div>
    </footer>
    <script>
        var t2031People = 0;
        var t2032People = 0;

        // Init Socket.IO to make connection to the Socket.IO server at the same URL
the page is being hosted on
        var socket = io()

        // Register Socket.IO event when rfid is emitted
        socket.on('rfid', function (nfcCard) {
            console.log("NFC Serial number: " + nfcCard);
        });

        AWS.config.update({
            region: "<insert your region>",
            endpoint: "<insert your endpoint>",
            accessKeyId: '<insert your accesskeyid>',
            secretAccessKey: '<insert your secretaccesskey>'
        });

        var dynamodb = new AWS.DynamoDB({ apiVersion: '2012-08-10' });
        var docClient = new AWS.DynamoDB.DocumentClient();
        var tablename = "<insert your table name>";
```

```
        // Function to get the number of people in the room stored in DynamoDB
        function getData(room) {
            console.log("get data called!")

            var params = {
                TableName: tablename,
                Key: {
                    "room": {
                        S: room
                    }
                }
            };

            dynamodb.getItem(params, function (err, data) {
                if (err) console.log(err, err.stack); // an error occurred
                else {
                    // Need to parse string to integer
                    var currentNoOfPpl = parseInt(data.Item.entered.N);
                    if (room === "t2031") {
                        $('#t2031People').html(currentNoOfPpl);
                    } else {
                        $('#t2032People').html(currentNoOfPpl);
                    }

                }
            });
        }

        // Refresh T2031 Button onclick handler
        $("#refreshT2031").click(function () {
            getData("t2031");
            swal("Refresh", "Current no. of people refreshed for T2031!",
"success");
        });

        // Refresh T2032 Button onclick handler
        $("#refreshT2032").click(function () {
            getData("t2032");
            swal("Refresh", "Current no. of people refreshed for T2032!",
"success");
        });

        $(() => {
            getData("t2031");
            getData("t2032");

        })
    </script>
```

```
</body>

</html>
```

Now, we will be coding for a MQTT publish program using python. This program can be run on another raspberry pi, RPI #1.

Rfid-publish-1.py

```
import RPi.GPIO as GPIO
import MFRC522
import signal
import paho.mqtt.client as paho
import os
import socket
import ssl
from time import sleep

connflag = False

def on_connect(client, userdata, flags, rc):
    global connflag
    connflag = True
    print("Connection returned result: " + str(rc) )

def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

mqttc = paho.Client()
mqttc.on_connect = on_connect
mqttc.on_message = on_message

# aws credentials
awshost = "<insert your awshost>"
awsport = 8883
clientId = "<insert your clientId>"
thingName = "<insert your thingName>"
caPath = "<insert your caPath>"
certPath = "<insert your certPath>"
keyPath = "<insert your keyPath>"

mqttc.tls_set(caPath, certfile=certPath, keyfile=keyPath,
cert_reqs=ssl.CERT_REQUIRED, tls_version=ssl.PROTOCOL_TLSv1_2, ciphers=None)

mqttc.connect(awshost, awsport, keepalive=60)

mqttc.loop_start()
```

```
uid = None
prev_uid = None
continue_reading = True


# 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()


# Hook the SIGINT
signal.signal(signal.SIGINT, end_read)

# Create an object of the class MFRC522
mfrc522 = MFRC522.MFRC522()

# Welcome message
print "Welcome to the MFRC522 data read example"
print "Pres Ctrl+C to stop."

# This loop keeps checking for chips.
# If one is near it will get th UID

while continue_reading:
    # Scan for cards
    (status, TagType) = mfrc522.MFRC522_Request(mfrc522.PICC_REQIDL)

    # If a card is found
    if status == mfrc522.MI_OK:
        # Get the UID of the card
        (status, uid) = mfrc522.MFRC522_Anticoll()
        if connflag == True:
            json = "{ \"isEnter\":" + "false" + "}"
            mqttc.publish("rooms/t2031", json, qos=1)
            print("MQTT Published:  " + json)
        # if uid != prev_uid:
        #     prev_uid = uid
        #     print("New card detected! UID of card is {}".format(uid))
        sleep(5)
```

You can create another duplicate of Rfid-publish-1.py and modify the json to ‘true’ and topic to ‘rooms/t2032’ to publish to another topic.

Section 6
Coding Telegram Bot

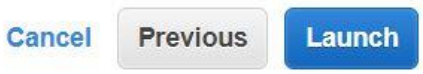
A. Create a Telegram Bot

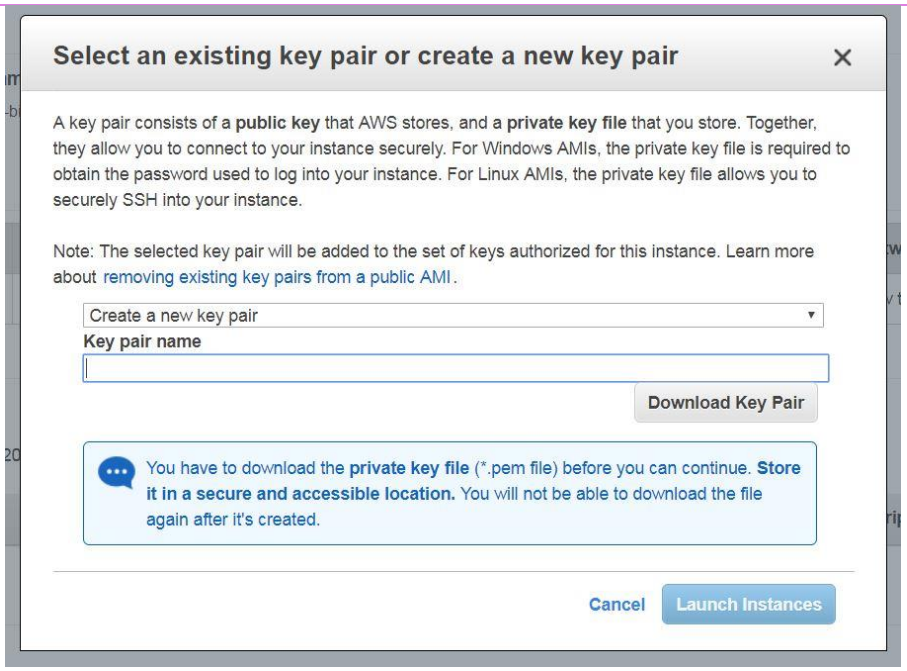
We will need to have a telegram bot so that people are able to interact with our application through telegram.

No	Task
1	On Telegram, search and start chatting with **BotFather**
2	Enter \newbot to create a new bot
3	Give it a name, this name will be shown to other people
4	Give it a username, this is how other people find this bot
5	Copy the **token** and save it for later use

B. Create AWS EC2 Instance for Telegram Bot

For the codes of the telegram bot, we will execute them on a AWS EC2 Instance so that it is always running.

No	Task
1	On AWS Console, search and go to EC2 Region does not matter.
2	Click Launch Instance to create a new instance 
3	Choose Microsoft Windows Server 2012 Base 
4	Click Review and Launch at the bottom right We are skipping steps because we do not need to do any additional configurations  Click Launch at the bottom right 
5	Create a new key pair if you do not have one. Name it any Key pair name you want. Save this file on your computer where you can find it, we will need it later on.

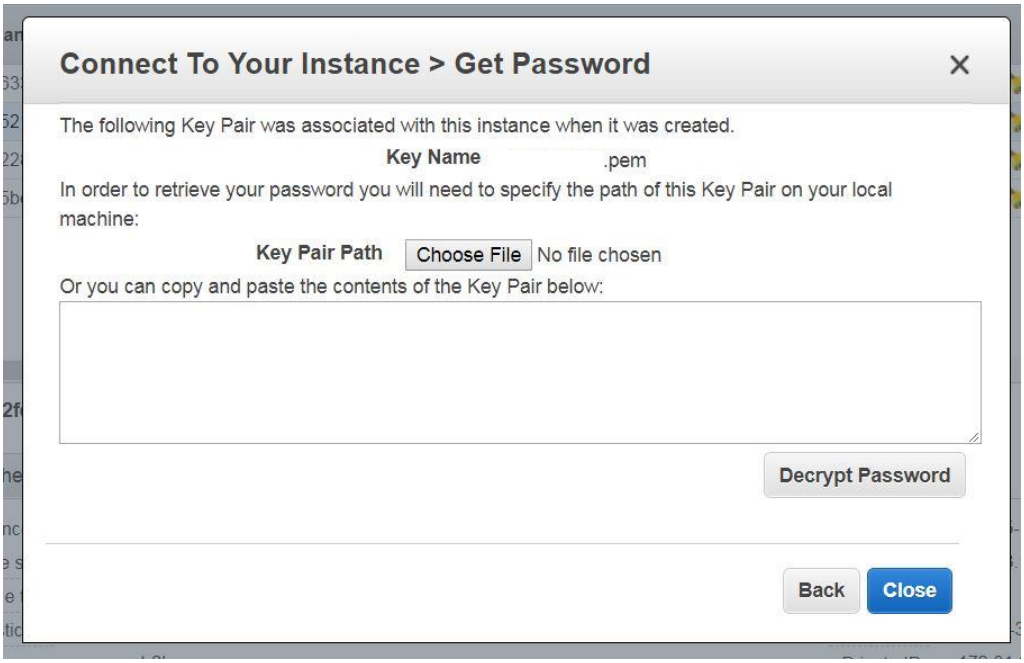


6 At the Instances page, wait for the newly created instance to be ready.
Takes a few minutes.

When it is ready, it will look like this:

C. Connect AWS EC2 Instance for Telegram Bot

Now, let's connect our AWS EC2 instance so that our Telegram Bot can use it! ☺

No	Task
1	At the EC2 Instances page, select the instance you created. Click Connect and then click Get Password. Upload the pem file we saved previously to get the password.  We will need the values of **Public DNS**, **User name**, **Password**, save it somewhere for convenience sake if you want
2	On your Windows OS, open Remote Desktop Connection. For the Computer, enter the **Public DNS** eg: xxx.us-west-2.compute.amazonaws.com For User and Password, enter **User name** and **Password** obtained from previous step.

D. Configure AWS EC2 Instance for Telegram Bot

For the instance to be able to run the codes, we will need to install and configure a few things.

No	Task
1	Either press the windows key on the keyboard or move the mouse pointer to the corner left and click Open Internet Explorer to search and download installers
2	Search Python and download Python 2.7 Install it and follow instructions

3	Start a command prompt or powershell and enter the follow lines to install the libraries:
	`C:\Python27\Scripts\pip.exe install boto3`
	`C:\Python27\Scripts\pip.exe install telepot`
	`C:\Python27\Scripts\pip.exe install AWSIoTPythonSDK`
	`C:\Python27\Scripts\pip.exe install paho-mqtt`

4	Go to https://docs.aws.amazon.com/cli/latest/userguide/awscli-install-windows.html and install the AWS CLI MSI installer for Windows (64-bit). **To install the AWS CLI using the MSI installer** 1. Download the appropriate MSI installer. • Download the AWS CLI MSI installer for Windows (64-bit) • Download the AWS CLI MSI installer for Windows (32-bit) **Note** The MSI installer for the AWS CLI does not work with Windows Server 2008 (version 6.0.6002). Use pip to install with this version of Windows. Run this on command prompt or powershell `C:\Python27\Scripts\aws configure` Enter your own **Access Id, Secret Key and Region**
5	Create a new Folder at C:/ and name it **telepotserver** Use your creativity and transfer your **certificate.pem.crt**, **private.pem.key**, **public.pem.key** and **rootca.pem** files into this folder.
6	In telepotserver folder, create a new file called **s3dynamodb.py** Enter the following codes:

```
from __future__ import print_function # Python 2/3 compatibility
import boto3
import json
import decimal
from boto3.dynamodb.conditions import Key, Attr
from botocore.exceptions import ClientError

# Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            if o % 1 > 0:
                return float(o)
            else:
                return int(o)
        return super(DecimalEncoder, self).default(o)

dynamodb = boto3.resource("dynamodb", region_name='<insert your region>',
endpoint_url="<insert your endpoint>")

table = dynamodb.Table('<insert your table>')

def readDynamoDBItem(inRoom):
    try:
        response = table.get_item(
            Key={
                'room': str(inRoom)
            }
        )
    except ClientError as e:
        print(e.response['Error']['Message'])
        return "Error in getting item from dynamodb"
    else:
        item = response['Item']
        print("GetItem succeeded:")
        #mItem = json.dumps(item, indent=4, cls=DecimalEncoder)
        return str(item['entered'])
```

7 In telepotserver folder, create a second new file called **onlytelebot.py**
Enter the following codes:

```
import telepot
from time import sleep
import random
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
from s3dynamodb import *
import paho.mqtt.client as paho
```

```
import os
import socket
import ssl

connflag = False

def on_connect(client, userdata, flags, rc):
    global connflag
    connflag = True
    print("Connection returned result: " + str(rc) )

def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

mqttc = paho.Client()
mqttc.on_connect = on_connect
mqttc.on_message = on_message
#mqttc.on_log = on_log

awshost = "<insert your awshost>"
awsport = 8883
clientId = "<insert your clientId>"
thingName = "<insert your thingName>"
caPath = "<insert your caPath>"
certPath = "<insert your certPath>"
keyPath = "<insert your keyPath>"

mqttc.tls_set(caPath, certfile=certPath, keyfile=keyPath,
cert_reqs=ssl.CERT_REQUIRED, tls_version=ssl.PROTOCOL_TLSv1_2, ciphers=None)

mqttc.connect(awshost, awsport, keepalive=60)

mqttc.loop_start()

# Connect and subscribe to AWS IoT
#my_rpi.connect()

#telegram bot
my_bot_token = '<insert your bot token>'

#s3
BUCKET = '<insert your s3 bucket name>'

def allCommandsMsg():
    return "thank me for the halp u peasant\nlist of available
commands:\n\nTake Photo in T2031: take photo\nGet number of people in T2031:
t2031\nGet number of people in T2032: t2032"
def noCommandMsg():
```

```
    num = random.randint(0, 15)
    if num < 5:
        return "no such command, try again"
    elif num < 10:
        return "**crying meep** no such command, dont bully me T_T"
    else:
        return "did u typo? dont be kailing no.2"

def respondToMsg(msg):
    chat_id = msg['chat']['id']
    command = msg['text']
    print('Got command: {}'.format(command))

    command = command.lower() #lower case it
    # filter commands
    if command == 'help':
        bot.sendMessage(chat_id, allCommandsMsg())
    elif command == 'thankyou' or command =='thx' or command =='thank you':
        bot.sendMessage(chat_id, "awww im touched blush blush")
    elif command == "t2031" or command == "t2032":
        noOfPpl = readDynamoDBItem(command)
        bot.sendMessage(chat_id, "No of people in room " + command + ": " +
noOfPpl)
    elif command == "take photo":
        file_name = str(msg['date']) + ".jpg"
        s3filepath = str(chat_id) + "/" + file_name
        mValues =
"\"file_name\":\"{0}\",\"chat_id\":\"{1}\",\"s3filepath\":\"{2}\"".format(str(
file_name),str(chat_id),str(s3filepath))
        mqttc.publish("rooms/t2031/takephoto", "{"+mValues+"}", qos=1)
        bot.sendMessage(chat_id, "Please wait...")
    else:
        bot.sendMessage(chat_id, noCommandMsg())

bot = telepot.Bot(my_bot_token)
bot.message_loop(respondToMsg)
print('Listening for RPi commands...')
while True:
    sleep(1)
```

8 Start a command prompt or powershell and enter this to run the codes for telegram bot

```
C:\Python27\Python.exe C:\telepotserver\onlytelebot.py
```

Pray and it should be running without problems. Start sending messages to the telegram bot and test it out! 😊

	Note that at this point, the take photo command is not ready yet. However, the t2031 and t2032 commands are ready.

Section 7
Coding Raspberry Pi Camera

This section is for a raspberry pi that will only use its camera.

When it receives a message from the telegram bot through MQTT subscribe, it will take a photo with its camera, upload to S3 Bucket, process using AWS Rekognition and return the results to the user through telegram bot.

A. Setup folder and files

No	Task
1	Install the libraries as stated in Section 4 if you have not done so
2	At your raspberry pi desktop, create a folder called **sfsphoto** This folder will store all the photos taken by the raspberry pi camera
3	Create a folder called **rasp-camera**
4	Transfer your **certificate.pem.crt**, **private.pem.key**, **public.pem.key** and **rootca.pem** files into rasp-camera folder.

5	In rasp-camera folder, create a new file called **picam_s3_rekognition_2.py**
	Enter the following codes:
	
	`import boto3`
	`import botocore`
	`from picamera import PiCamera`
	`from time import sleep`
	
	`# Set the filename and bucket name`
	`BUCKET = '<insert your BUCKET>' # replace with your own unique bucket name`
	`location = {'LocationConstraint': '<insert your location>'}`
	`file_path = "/home/pi/Desktop"`
	`file_name = "test1.jpg"`
	
	`def takePhoto(file_path,file_name):`
	
	`with PiCamera() as camera:`
	`#camera.resolution = (1024, 768)`
	`full_path = file_path + "/" + file_name`
	`camera.capture(full_path)`
	`sleep(3)`
	
	`def uploadToS3(file_path,file_name,bucket_name,location,s3filepath):`

```
    s3 = boto3.resource('s3') # Create an S3 resource
    exists = True

    try:
        s3.meta.client.head_bucket(Bucket=bucket_name)
    except botocore.exceptions.ClientError as e:
        error_code = int(e.response['Error']['Code'])
        if error_code == 404:
            exists = False

    if exists == False:

s3.create_bucket(Bucket=bucket_name,CreateBucketConfiguration=location)

    # Upload the file
    full_path = file_path + "/" + file_name
    s3.Object(bucket_name, s3filepath).put(ACL="public-
read",Body=open(full_path, 'rb'))
    print("File uploaded")


def detect_labels(bucket, key, max_labels=10, min_confidence=90,
region="<insert your region>"):
    rekognition = boto3.client("rekognition", region)
    response = rekognition.detect_labels(
        Image={
            "S3Object": {
                "Bucket": bucket,
                "Name": key,
            }
        },
        MaxLabels=max_labels,
        MinConfidence=min_confidence,
    )
    return response['Labels']
```

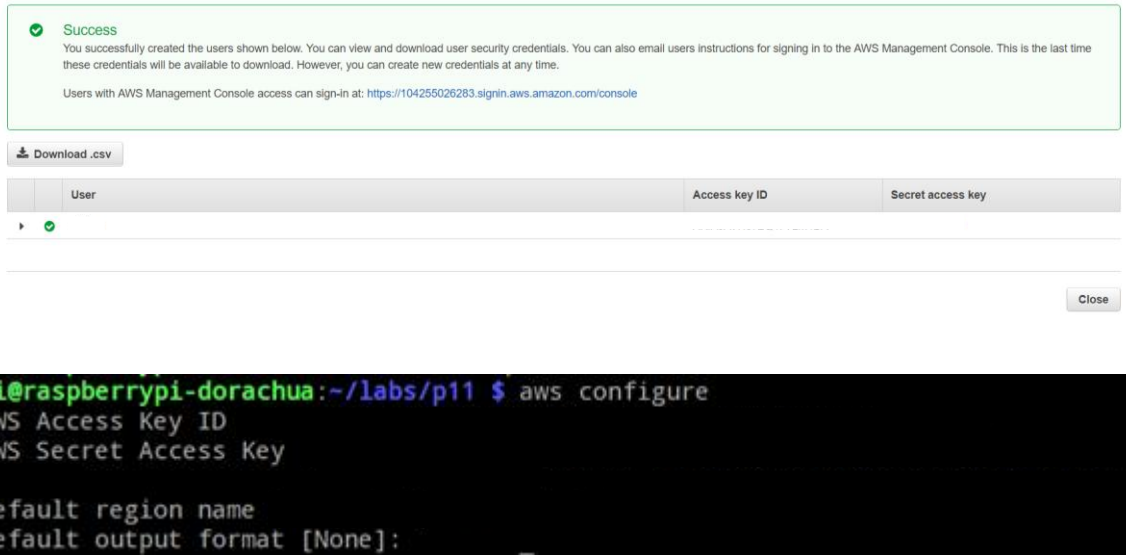
6 In rasp-camera folder, create a new file called **rasppublish.py**
Enter the following codes:

```
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
from time import sleep
import ast
#camera
from picam_s3_rekognition_2 import *

import telepot

import paho.mqtt.client as paho
```

```
import os
import socket
import ssl

BUCKET = '<insert your BUCKET>'
location = {'LocationConstraint': '<insert your location>'}
file_path = "/home/pi/Desktop/sfsphotos"

my_bot_token = '<insert your my_bot_token>'

bot = telepot.Bot(my_bot_token)

def on_connect(client, userdata, flags, rc):
    print("Connection returned result: " + str(rc) )
    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe("rooms/t2031/takephoto" , 1 )

def on_message(client, userdata, msg):
    print("topic: "+msg.topic)
    print("payload: "+str(msg.payload))
    result = ast.literal_eval(msg.payload)
    print(result)
    chat_id = result['chat_id']
    file_name = result['file_name']
    s3filepath = result['s3filepath']

    #s3 and stuff
    try:
        bot.sendMessage(chat_id, "Taking Photo...")
        takePhoto(file_path,file_name)
        print("take photo success")
        bot.sendMessage(chat_id, "Photo Taken, now uploading...")
        uploadToS3(file_path,file_name, BUCKET,location,s3filepath)
        print("uploaded to s3")
        bot.sendMessage(chat_id, "Uploaded Photo, now processing")
        bot.sendMessage(chat_id, "https://s3-us-west-
2.amazonaws.com/"+BUCKET+"/"+s3filepath)

        replyMsg = ""
        for label in detect_labels(BUCKET, s3filepath):
            print("{Name} - {Confidence}%".format(**label))
            replyMsg += "\n{Name} - {Confidence}%".format(**label)
        if replyMsg == "":
            print("nth found")
            bot.sendMessage(chat_id, "Did not detect anything")
        else:
            print("Listing probabilities:\n" + replyMsg)
```

```
            bot.sendMessage(chat_id, "Listing probabilities:\n" + replyMsg)
    except:
        print("Error in takePhotoCommand")
        bot.sendMessage(chat_id, "Error! D:")

mqttc = paho.Client()
mqttc.on_connect = on_connect
mqttc.on_message = on_message
#mqttc.on_log = on_log

awshost = "<insert your awshost>"
awsport = 8883
clientId = "<insert your clientId>"
thingName = "<insert your thingName>"
caPath = "<insert your caPath>"
certPath = "<insert your certPath>"
keyPath = "<insert your keyPath>"

mqttc.tls_set(caPath, certfile=certPath, keyfile=keyPath,
cert_reqs=ssl.CERT_REQUIRED, tls_version=ssl.PROTOCOL_TLSv1_2, ciphers=None)

mqttc.connect(awshost, awsport, keepalive=60)

mqttc.loop_forever()

while True:
    sleep(1)
```

B. Configure AWS Credentials

No	Task
1	Type the following command in your Raspberry Pi terminal at **rasp-camera** folder so that you can use the AWS CLI to configure your credentials file: `aws configure`
2	Enter the Access Key ID and Secret Access Key id you obtained earlier when you created the account by referring to the .csv file you downloaded. 

In rasp-camera folder, run the following code to start the program.

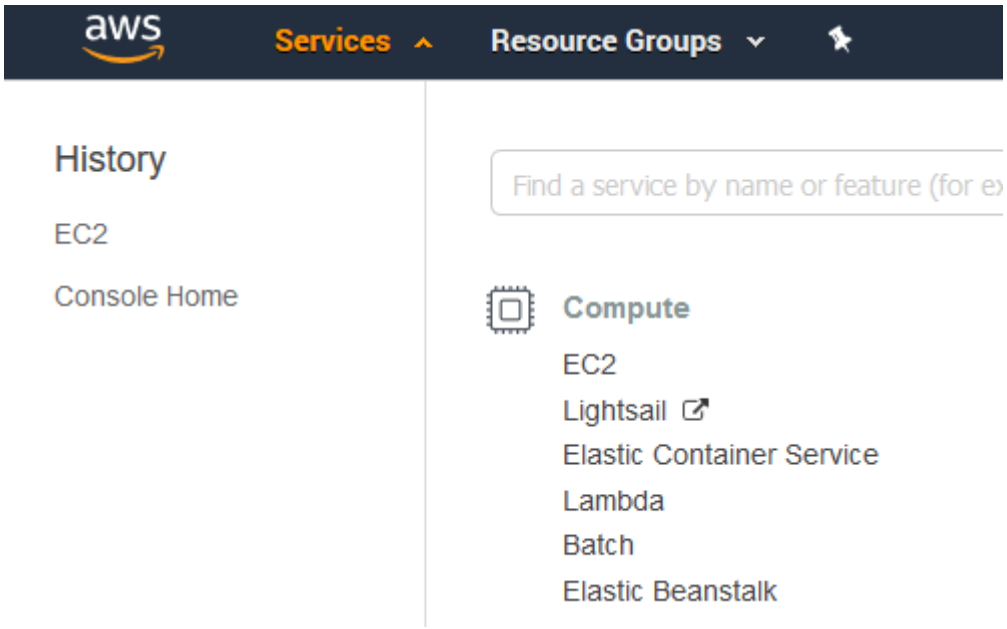
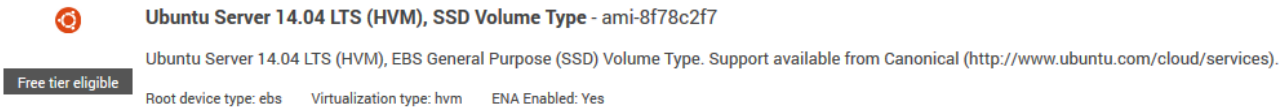
```
python rasppublish.py
```

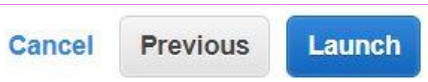
Your raspberry pi is now ready to take photo!
Message your telegram bot take photo and test it out! 😊

Section 8
Deploy SOMS on Amazon EC2

A. Create Amazon EC2 Instance

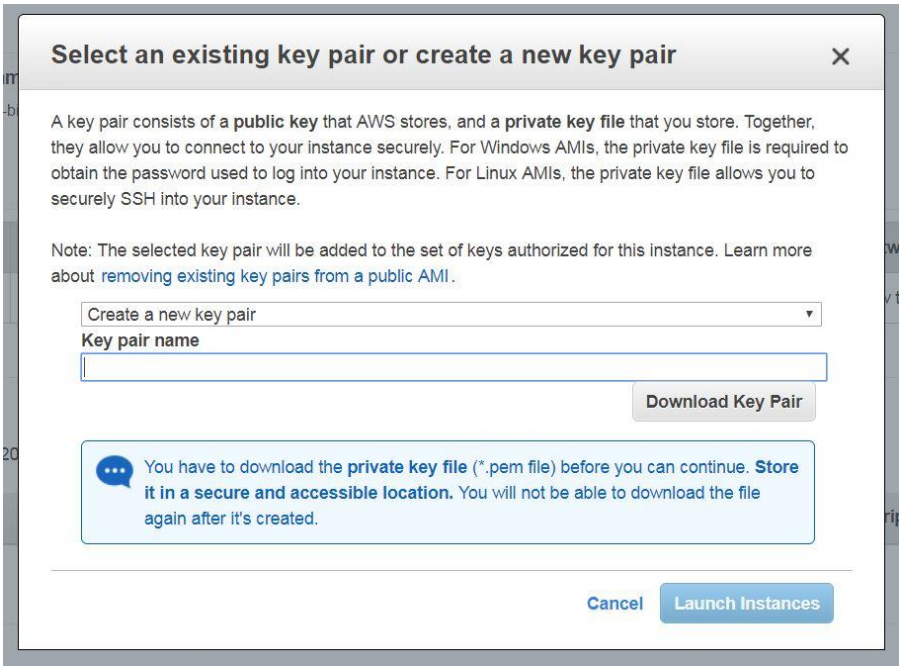
In order to access our SOMS web application at a publicly available URL to share with the world, we chose to host our web application on the cloud.

No	Task
1	First, let's create a Amazon EC2 instance to deploy our SOMS NodeJS web application. In your AWS Dashboard, click on Services > EC2. 
2	Click "Launch Instance". Select "Ubuntu Server 14.04 LTS". 
3	Click "Review and Launch", then "Launch". 



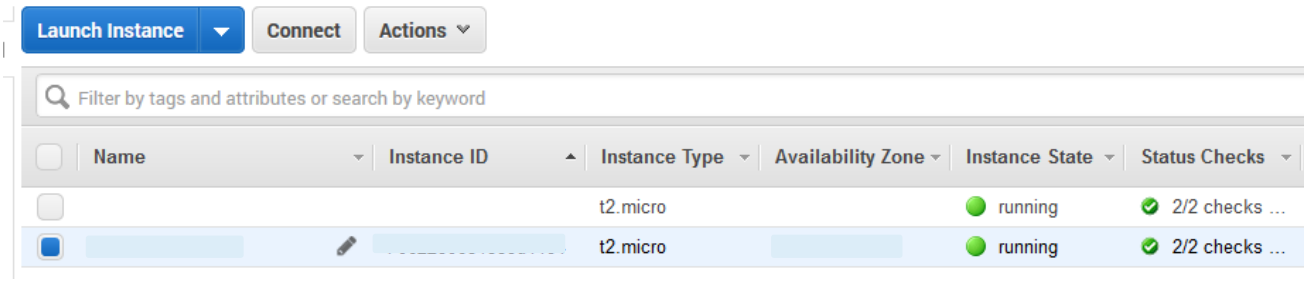
4 You will be prompted to select an existing key pair or create a new key pair. At this stage, I chose to create a new key pair.

You can name the key pair name to anything you want and then click “Download Key Pair”.



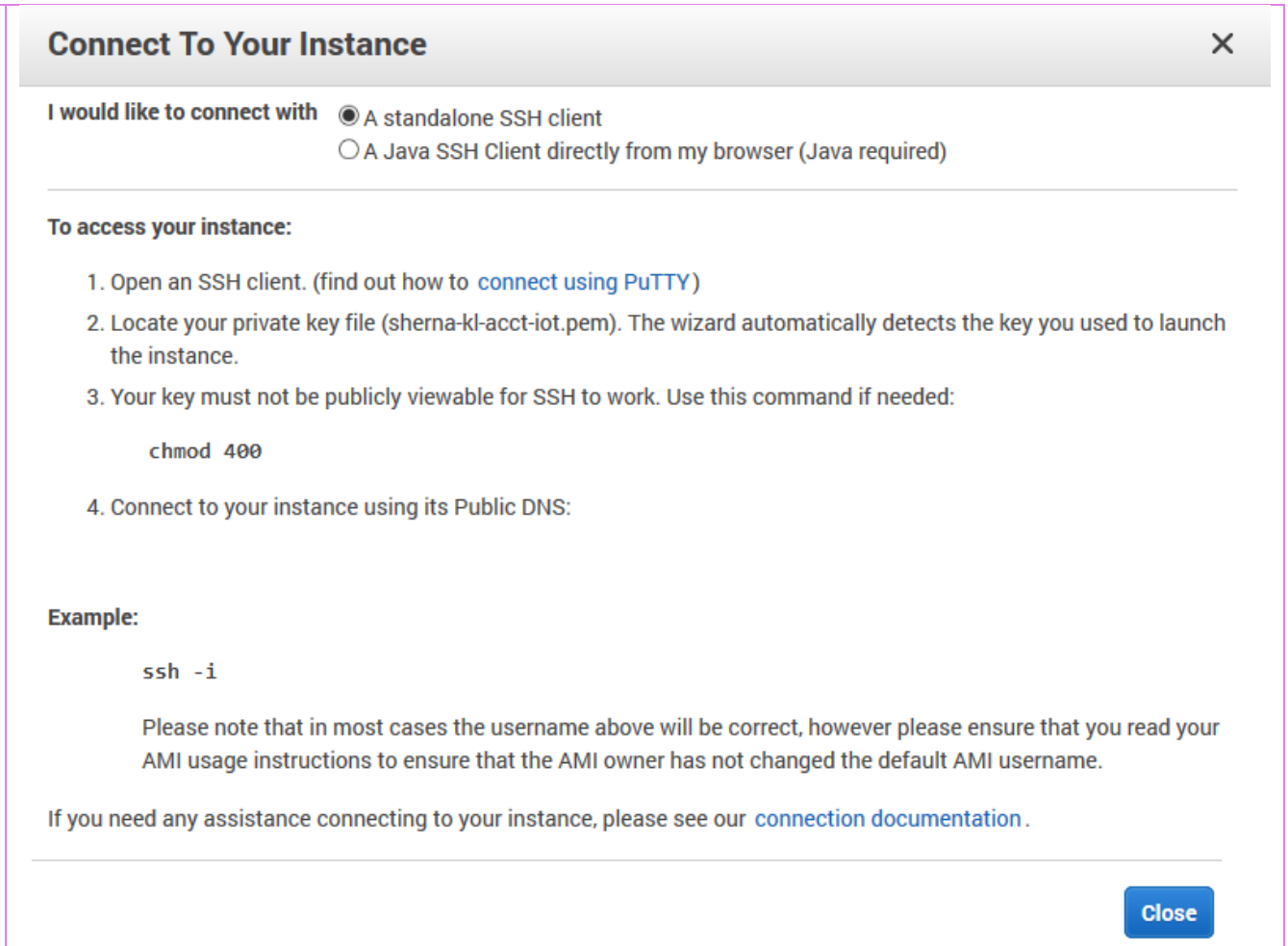
Finally, click “Launch instances”.

5 Go to your EC2 dashboard and you can rename the instance to whatever you fancy.
When the Instance State is “running” and the Status Checks is no longer “initializing”, you can connect to it!



6 Check the instance and click “Connect”.

Copy the command listed in the example.



7 Now, open the Terminal (if you are on Mac) or Command Prompt (Windows). Since I am on Windows, I will use a better version of Command Prompt called Cmder.

Simply execute the command you copied in the same location where you have downloaded your .pem file.

Now, it will prompt you if you want to continue. Enter "yes" and once you see this, you are successfully connected to your EC2 instance!

```
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-141-generic x86_64)
```

B. Setup and deploy SOMS on EC2

Since you have a fresh Ubuntu EC2 instance, we need to install our web application Node.js environments and other stuff in order to deploy our web application.

No	Task

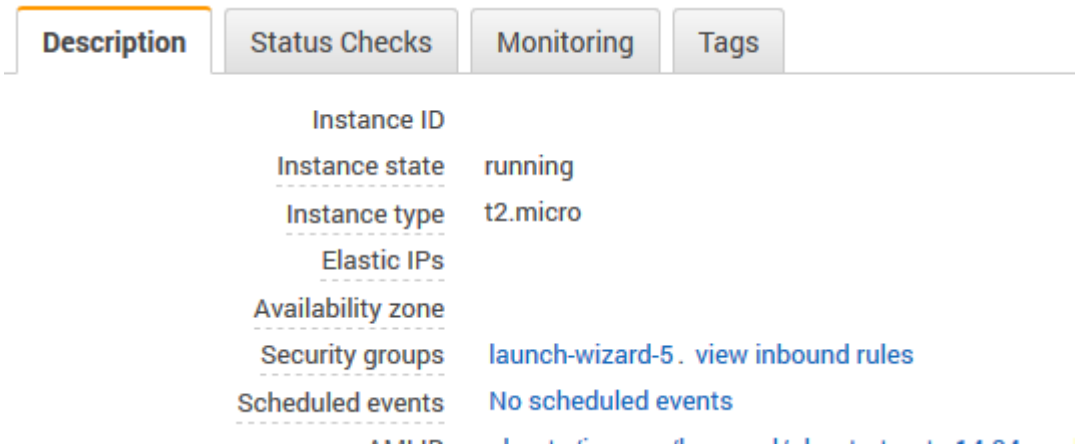
1	Execute the following commands to update your apt package manager.
	
	`sudo apt-get update`
	
	`sudo apt-get upgrade`

2	Let's proceed with an installation of the latest version of Node at the moment which is Node 9.5.0. `curl -sL https://deb.nodesource.com/setup_9.x | sudo -E bash -`
3	Now that we have added the NodeSource package repository, we can move on and install Node.js! `sudo apt install nodejs`
4	We can then test and see what version of Node we are running and launch the Node REPL as we discussed in the previous article as a quick test to confirm the installation was successful. 

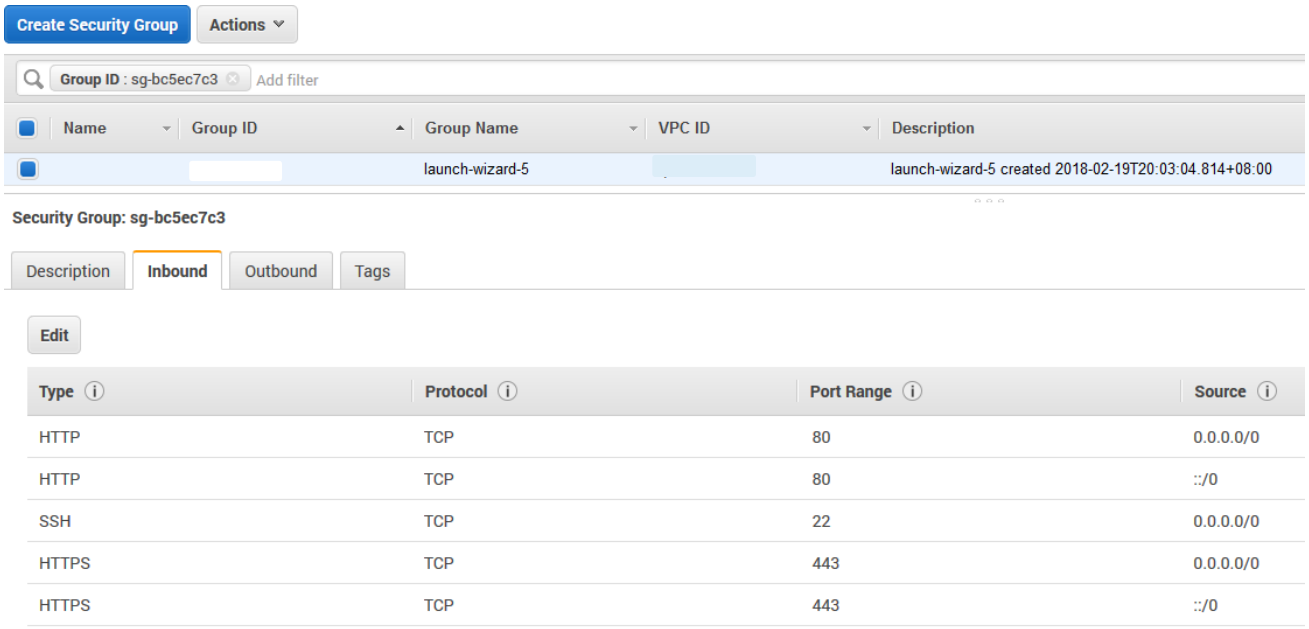
5	Now, lets install git so that you can clone my project!
	
	`sudo apt-get install git`
	
	`git clone https://github.com/shernaliu/.git`

6	Enter your credentials!
7	Now, lets install the dependencies the web application requires! In the back_end directory and front_end directory, execute npm install
8	One last step before we start up the web app. Remember in your Ubuntu EC2 instance, you need to set the inbound rules to allow access from SSH, HTTP (Port 80) and HTTPS (Port 443).

Go to EC2 dashboard, click on your EC2 instance. In the bottom section, scroll down to the Security groups section.
Click on “launch-wizard-5” to view your security group.



Click “Inbound” and “Edit”.



Add “HTTP” and “HTTPS” rules and click “Save”.

	In server.js, our NodeJS application must be running on port 80. Remember to modify server.js to change it to run NodeJS application on port 80.

9	Finally, simply execute the following command to start up our application!
	
	`sudo node server.js`
	
	Now, simply access the application at the EC2 url.
	
	Note: If you close the Terminal, Node.js will stop running.
	
	You can run the Node.js in the background by executing the following commands.
	
	Ctrl + Z
	
	`bg`
	
	Reference: https://askubuntu.com/questions/8653/how-to-keep-processes-running-after-ending-ssh-session

-- End of CA2 Step-by-step tutorial --