|  |  |
| --- | --- |
| https://lh6.googleusercontent.com/zMKcj5jF8F1Z9U2kA-dfQLhgdQAPi_euo8tiepSP2GUcSW22LSagZK3NIV2zp0SfYBYUSH_9gntT0CRTwXFRsieF0dRhNBX-xt_YL8sSYpIhf4dDVYuK8VAj9oik8uzwDQ58r5iA  IoT Doorbell  Submitted on May 24, 2019 | CET 4925 – Internet of things Professor xiaohai li  Section D562  Eric Chan and Oscar Familia  Spring 2019 |

Table of Contents

[Abstract and Description 2](#_Toc9553215)

[System Design 4](#_Toc9553216)

[Components Design 6](#_Toc9553217)

[Testing Procedure and Analysis 6](#_Toc9553218)

[Wiring Schematics 9](#_Toc9553219)

[List of Devices and Parts Used 9](#_Toc9553220)

[Discussion and Conclusion 10](#_Toc9553221)

[Source Code 12](#_Toc9553222)

[Main Code 12](#_Toc9553223)

[Publisher Test Code 14](#_Toc9553224)

[References 15](#_Toc9553225)

# Abstract and Description

In this IoT project, the group will be constructing an Internet connected doorbell, mainly consisting of a Raspberry Pi Zero W, a Pi supported camera module, and the various tools available by the Amazon Web Services or simply AWS. The doorbell remains on standby until when the doorbell, which is represented by a push button, is pressed and released. This action signifies a simple clock pulse, changing the connection from a digital zero to a digital one and then back to a digital zero. That return to digital zero creates a cascade of functions for the doorbell to properly execute its duties. Firstly, it notes that Pi camera to capture an image of that instance, imaging the whatever’s outside of the door, and stores the picture with the current date and time as its file name. Next, it will create an email message, attached the image file to the message, and send the email using preset credentials within the code that contains the sending address, receiving address, email subject, and the email login information to which the doorbell uses to send that email. Lastly, after the email the Pi will publish a message to AWS IoT with a topic to follow and proceeds to send this message over to the cloud to then which will then be picked up if it is subscribed to that same topic, notifying the user that the doorbell has been pressed.

The primary component of this project starts with the small board computer or SBC, the Raspberry Pi Zero W. The W, which stands for wireless, is a continuation to the Raspberry Zero 1.3, with newly added onboard 802.11b/g/n single band 2.4 GHz Wi-Fi and 4.1 BLE Bluetooth. Raspberry Pi Zero W has a 1 GHz [ARM11](https://en.wikipedia.org/wiki/ARM11)76JZF-S CPU, 512 MB of RAM, two micro USB ports with one dedicated solely to power, a mini HDMI port, composite video and reset headers, and the port that the project revolves around: the Camera Serial Interface or CSI camera port. This CSI camera connector is what enables the Pi to connect with the Pi Camera module and natively support it right on the spot, with a setting change. The same forty HAT (Hardware Attached on Top) header pins, that are present on the larger Raspberry Pi 3 models, are still available to use on the Pi Zero W. It’s small form factor and with the newly added wireless capabilities makes the Pi Zero W ideal for small IoT (Internet of Things) projects and it was what gravitated the group towards using this SBC. The group planned on using its small size to pack into spaces that were not viable with its Raspberry’s flagship models, and with the same forty GPIO pins that are available to use, there are possibilities to connect different devices to the Zero W. The drawbacks we considered was the lack of ports on the device, to which one has already been occupied due to it being a power supply, and the second drawback was its weak CPU processing power and poor amount of RAM, these limited expectations of the Pi Zero W processing anything intensive as it would overheat the system and potentially shut down. The consensus was to attach simple components that operator together and avoid graphically intensive or power-hungry components to prolong the life of the device.

The next crucial component that’s present in the project is the Raspberry Pi Camera Module, specifically the second version. Version 2 replaces the five-megapixel camera sensor for an eight-megapixel sensor, so it can take higher resolution pictures and videos. In specifics, the camera module can capture a maximum picture resolution of 3280 pixels by 2464 pixels with an aspect ratio of 4:3 and can take video at that same resolution at maximum fifteen frames per second, at 1920 x 1080 16:9 it’s thirty frames per second and at 1280 x 720 16:9 & 640 x 480 4:3 at ninety frames per second. There are two official Pi Camera Modules from Raspberry, one is the regular color camera and one night-vision, both contain the same CMOS (complementary metal oxide semiconductor) Sony IMX219PQ sensor but the night vision does not contain an infrared sensor, so it is more sensitive in low light, dubbing it “night vision.” For the project, the regular color Pi Camera Module will suffice as there’s no niche use of the night vision version. Despite its standard 15-pin MIPI CSI standard interface, the Pi Camera Module usually comes standard with a 1.0 mm fifteen pin interface ribbon cable, but the Pi Zero W uses a smaller 0.5 mm twenty-two pin connector to shrink the footprint of the device. To be able to fully utilize the camera module on the Pi Zero W, an adapter is needed to connect the larger fifteen-pin connector to the smaller twenty-two pin.

The web service that the group is employing and that will make the project truly IoT is Amazon Web Services. AWS is a remote cloud computing platform that provides a suite of services at any level from hobbyists to companies to whole governments. Notable companies that use AWS include Autodesk, Lyft, NASA, Netflix, Reddit, Samsung, Spotify, Yelp, and Disney. This cloud computing platform has general services like blockchain, database storage, machine learning, network delivery, and security. The main general service that the group will be using is IoT, especially the AWS IoT Core function, which allows devices, in this case, the Raspberry Pi Zero W, to be connected to their cloud service. Once connected to the cloud, any data that is published by the Pi Zero W will show up on AWS, there is a service that can track any events happening from the device, AWS IoT events, and even graph out multiple devices and web services to build IoT applications, AWS IoT Things Graph, which very similar to IBM’s Node-RED. IoT connected devices typically will use MQTT (message queuing telemetry transport), an ISO standard publish and subscribe based protocol, to communicate to and from both devices and the cloud. Before even beginning to send these messages, the connecting device must contain access keys and certificates, these security measures are to prevent tampering with the encrypted messages. Public keys allow for a message to be encrypted and sent, but it’s the private key that allows that message to be decrypted and read. Much like the keys, the certificates operate in a similar way, proving ownership that the device in possession is truly yours, to begin with, as well as providing ownership to the private key signing it off as your own. In all above, the device importantly must have a root certificate, it’s a public key certificate that self-signs any device trying to connect to its respective web service, the purpose of this is to legitimize the devices and gives existence in its network.

# System Design

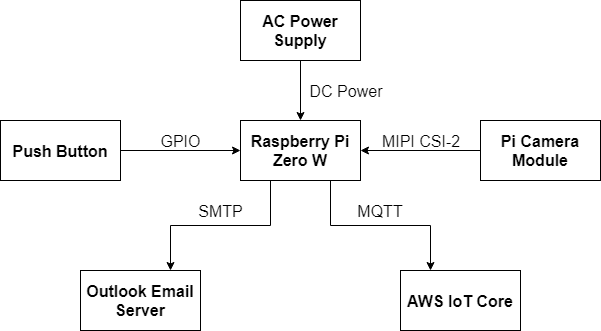
****

Figure : Project Block Diagram

For the project, components connected with the Pi in a star topology with the Pi Zero W in the middle and all connected components and services on the edges. The AC power supply provides DC current to the Pi which power not only the Pi but all other components as well. The push button imitating as the doorbell is connected via the GPIO header pins on the Pi. The Pi Camera module is connected to the dedicated CSI connector on the side of the Pi. When the Pi is connected to the Internet, it can connect to the Outlook email server using the Simple Mail Transfer Protocol (SMTP) by providing the correct server name, port, and encryption method in the parameters. The Pi will also be connected to the AWS IoT Core service by communicated with the Amazon cloud via MQTT.

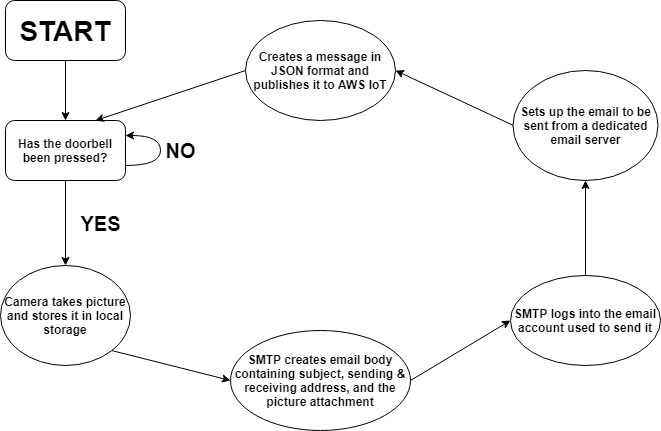
****

Figure : Project Flowchart

The ‘START’ represents when the Pi is powered on, connected to a network, and has the Python shell script started with the setup running. As soon as the loop starts, the Pi will be detecting a change in the state of the button, if it hasn’t at any point it will remain dormant, but if it’s pressed it will start the loop. First, the camera takes a picture and stores it in the desired directory formatted with the current date and time as the file name. Next, the imported SMTP module will create an email body containing the message subject, both the sending and receiving email address, and the recently took picture as the attachment. SMTP then logs into the doorbell sending email account to prepare for the sending. The email is then linked to the its own email server to establish connection and sends off the email. Lastly, the code creates a message in string, transform it to JSON format, and publishes the message off to AWS IoT.

# Components Design

The Pi Zero W has a forty-pin header soldered onto it as right out of the box, it is not included with one. The pin header makes adding and subtracting cables easier and more intuitive, with traditional soldering, pins can be burnt or inoperable, so de-soldering and re-soldering would be inefficient. The Pi Camera Module used for this project specifically supports the Zero’s 0.5 CSI connector, this reduces the addition of an adapter. The main code to which the Pi runs off a Python based Bash shell script, containing the modules utilized in the code, the setup for the camera, SMTP, and AWS, and lastly the loop section that’s always on standby when the doorbell is pressed. All the electrical and programming components are incased in a makeshift room imitating a house, with cardboard walls and plywood beams for support. The Pi and camera are mounting on the back of the cutout door, the doorbell is mounting on the side, and the AC power supply is routed through the back of the “house.”

# Testing Procedure and Analysis

To ensure that the Pi is functioning properly, connect it with one end to the dedicated power supply micro USB port and one end to a USB A port of a Windows PC, if working the computer will attempt to download driver software called “BCM2708 Boot” and on Device Manager, there should be that same name in “Other devices.” For the Pi camera module, simply connect to the Pi and enabled, enter the command “raspistill -v -o cam.jpg” this will use the native camera tool to take a picture, name it “cam”, format it as a JPEG file, and sent to the root directory of the Pi. Both components were functional right out of their packaging. As for the SMTP and AWS, using preexisting test code allows the Pi to test the connection of both.

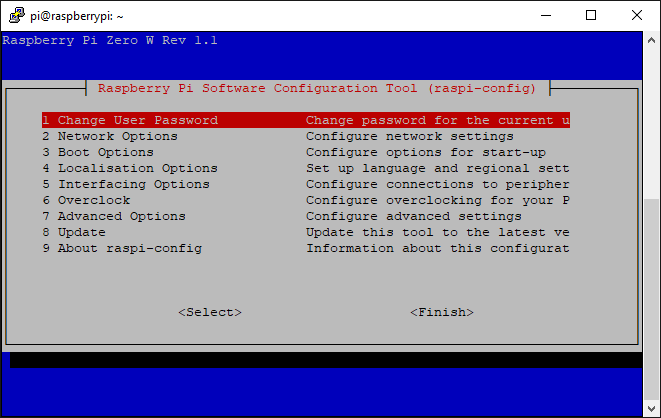


Figure : Raspberry Pi Software Configuration Tool

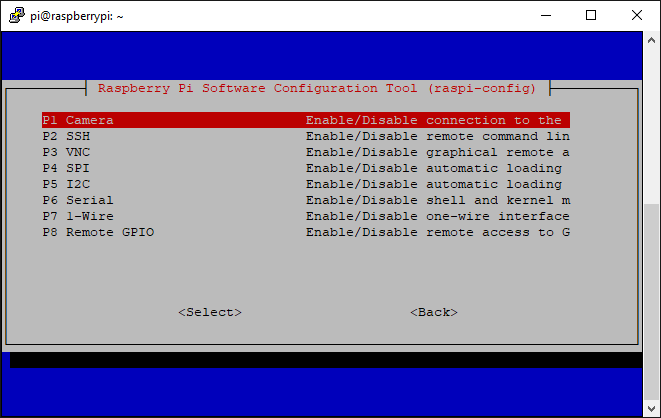


Figure : After selecting Interfacing Options

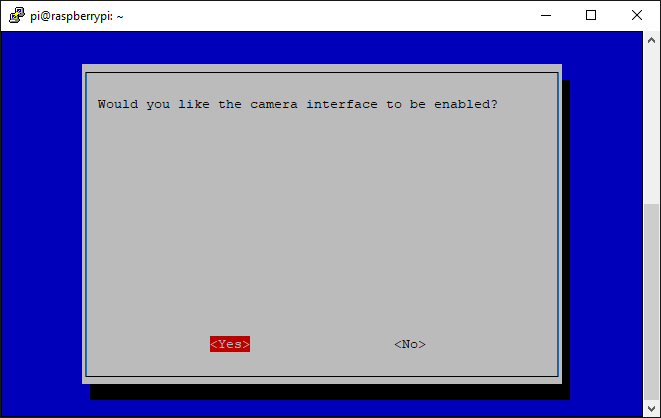


Figure : Enabling the camera interface

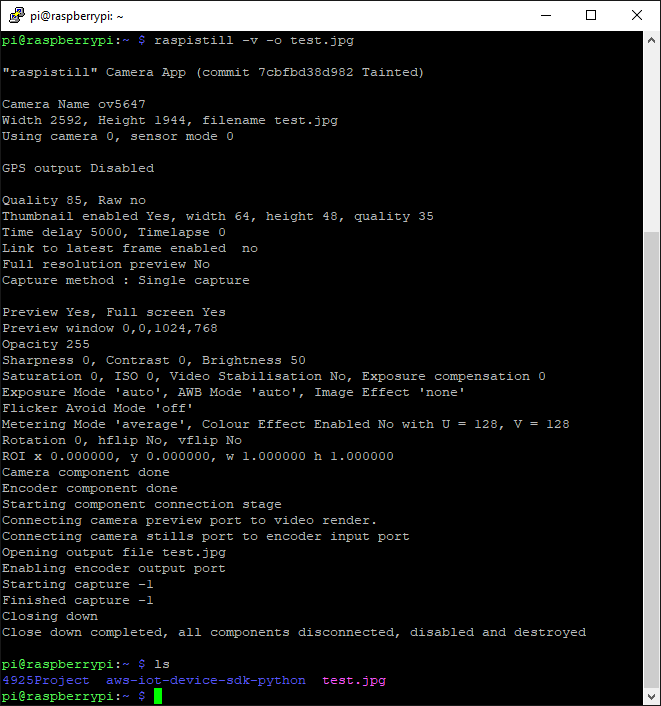
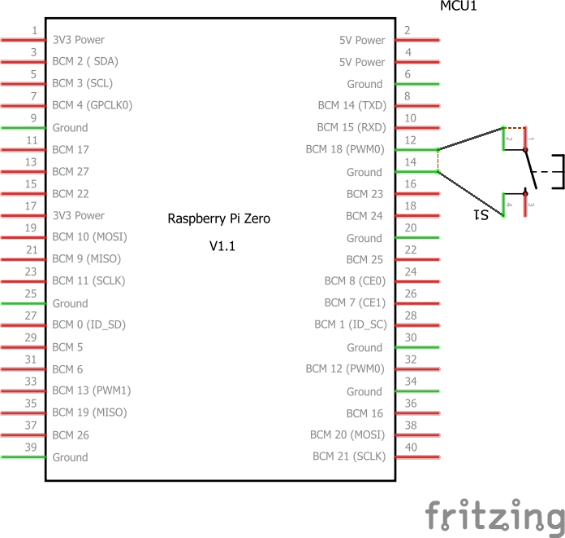
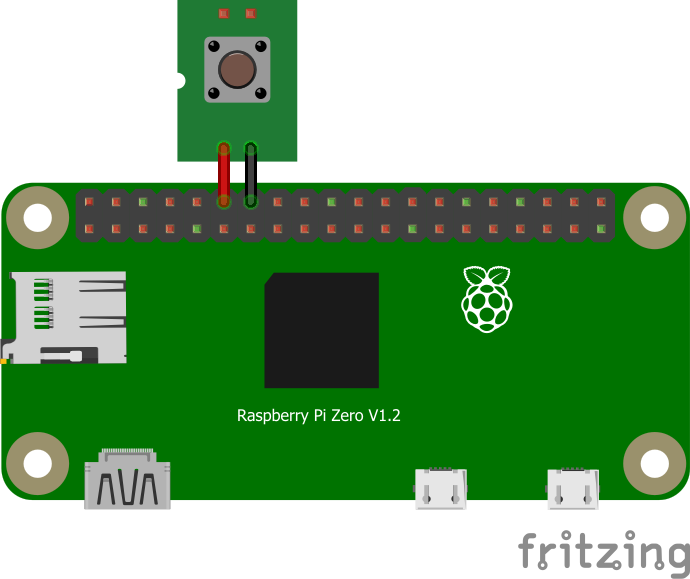


Figure : Testing the camera module

Many of the issues arise in the programming aspect, there were various errors along compilation that created different ways of solving them. First error was that the code came to a halt as soon as it encountered the ‘@’ symbol for the email address login info, the compiler deems it unrecognizable and stops the code. The solution required some searching but eventually it came down to adding an extra line of code preceding the main code, the line: “# -\*- coding: utf-8 -\*-“ ensures that the rest of code is encoded in the UTF-8 standard in a Python script, thus able to understand the letters and symbols in the script. The next error happened after transporting sections of code sent through a messaging service like Facebook Messenger, all sudden simple letters became foreign to the compiler, but as I removed those sections the code compiled normally again. The group hypothesized that the way the messaging service encodes then decodes the messages sent changed the format of the text itself. To test this, the group sent Python modules through the service and pasted them on the text editor. Normally, when entered by hand the ‘import’ keyword would be highlighted with a cyan color, but when pasted from the messaging service, the keyword was not highlighted. It did turn out to be the messaging service after all, and all subsequent code were sent stored inside a text document to preserve the correct format for the editor to use.

# Wiring Schematics



The parts that were available on Fritzing were the Raspberry Pi Zero W and the push button, one end of the push button is connected to GPIO BCM pin 18 and the other is connected to the pin right below for ground.

# List of Devices and Parts Used

* Raspberry Pi Zero W
* Raspberry Pi Camera Module
* Mini HDMI to HDMI adapter (to initialize the setup)
* Display devise with an available HDMI port
* One push button
* Female to male and female to female breadboard jumper cables
* Amazon Web Services subscription
* (Optional) 2 x 20 pin header (ease of attaching components without soldering)
* (Optional) Soldering iron
* (Optional) Solder with flux

# Discussion and Conclusion

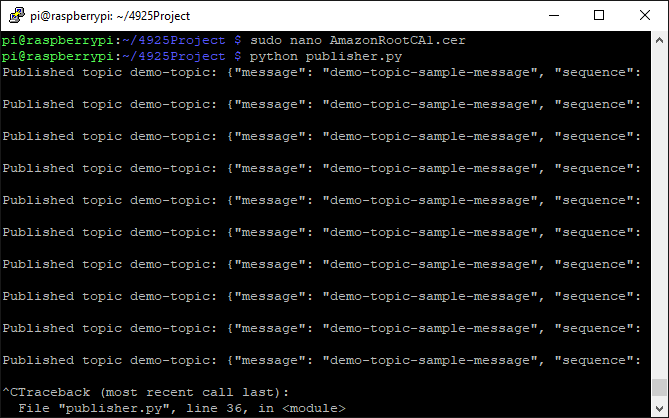


Figure : Testing the AWS publish feature

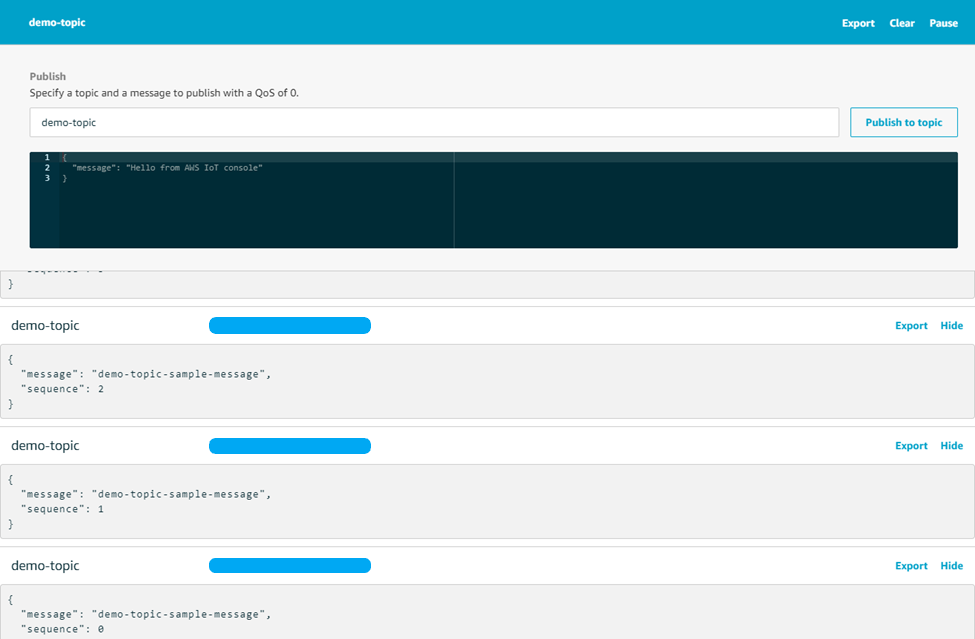


Figure : Successfully subscribing to the topic published in the Pi



Figure : Email successfully sent from the Pi with attached live picture

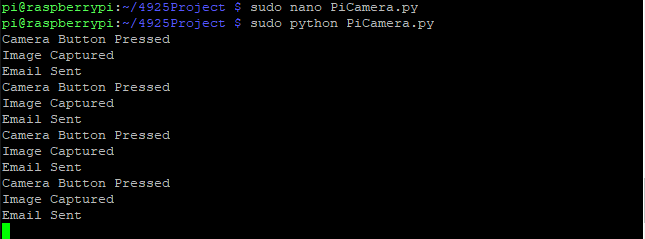


Figure :The main code looping after each button press

There were some aspects that could’ve been improved upon. One would be to always keep an extra Pi camera module at all time, as when it was time for presentation, it had the most unfortunate timing of malfunctioning, the Pi was unable to detect the camera module. It most likely broke during the setup for presentation. Secondly, there were services that could’ve been fully expedited to AWS for this project to become more IoT. The storage of all the real-time snapshots could’ve been stored using the AWS Simple Storage Service (S3) and AWS Simple Notification Service (SNS) could have been implemented to send the emails instead of relying on SMTP. Overall, the project succeeds in what it has sought out to do, grabbing a live picture and notifying the user of a presence at the door. When the code ran, it waited for the press, and as it did, it captures an image, sends that image off to be emailed as an attachment, and notifying the user on AWS.

# Source Code

## Main Code

# -\*- coding: utf-8 -\*-

# Main Modules

from email.mime.image import MIMEImage

from email.mime.multipart import MIMEMultipart

from picamera import PiCamera

import datetime

import smtplib

from time import sleep

import RPi.GPIO as gpio

import datetime

# AWS Modules

from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

import logging

import time

import argparse

import json

# AWS Configuration

host = "a1cai3u3kh553t-ats.iot.us-east-2.amazonaws.com"

certPath = "/home/pi/4925Project/"

clientId = "photon"

topic = "doorbell"

# Init AWSIoTMQTTClient

myAWSIoTMQTTClient = None

myAWSIoTMQTTClient = AWSIoTMQTTClient(clientId)

myAWSIoTMQTTClient.configureEndpoint(host, 8883)

myAWSIoTMQTTClient.configureCredentials("{}AmazonRootCA1.cer".format(certPath), "{}iot-private.pem.key".format(certPath), "{}iot-certificate.pem.crt".format(certPath))

# AWSIoTMQTTClient connection configuration

myAWSIoTMQTTClient.configureAutoReconnectBackoffTime(1, 32, 20)

myAWSIoTMQTTClient.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing

myAWSIoTMQTTClient.configureDrainingFrequency(2) # Draining: 2 Hz

myAWSIoTMQTTClient.configureConnectDisconnectTimeout(10) # 10 sec

myAWSIoTMQTTClient.configureMQTTOperationTimeout(5) # 5 sec

myAWSIoTMQTTClient.connect()

# Set the pin mode

gpio.setmode(gpio.BCM)

# Connecting push button to GPIO Pin 18

gpio.setup(18, gpio.IN, pull\_up\_down=gpio.PUD\_UP)

# Initializes the Pi Camera and sets it for night time photos

camera = PiCamera(resolution = (1280, 720))

#Setting up email parameters

toAddr = 'iotdoorbell@outlook.com'

fromAddr = 'iotdoorbell@outlook.com'

subject = 'Doorbell pressed, someone is at the door'

# Publish to the same topic in a loop forever

loopCount = 0

while True:

# Obtains the value from button press

inputCamera = gpio.input(18)

# When pressed, it saves the current date and time as the file name

if inputCamera == False:

print("Camera Button Pressed")

picturefile = camera.capture('/home/pi/4925Project/' + datetime.datetime.now().strftime('%Y-%m-%d%H:%M:%S') + '.png')

sleep(.2)

camera.capture(picturefile)

print("Image Captured")

# Create the message

msg = MIMEMultipart()

msg['Subject'] = subject

msg['From'] = fromAddr

meg['To'] = toAddr

# Attaches picture to email

File = open(picturefile, 'rb')

img = MIMEImage(File.read())

File.close()

msg.attach(img)

# Credentials for fromAddr

username = 'iotdoorbell@outlook.com'

password = 'password4925'

# Setting up the email to be sent from a Microsoft account

server = smtplib.SMTP('smtp-mail.outlook.com:587')

server.starttls()

server.login(username,password)

server.sendmail(fromAddr, toAddr, msg.as\_string)

server.quit()

print("Email Sent")

message = {}

message['message'] = "Someone is at the door!"

messageJson = json.dumps(message)

myAWSIoTMQTTClient.publish(topic, messageJson, 1)

print('Published topic %s: %s\n' % (topic, messageJson))

loopCount += 1

time.sleep(10)

myAWSIoTMQTTClient.disconnect()

## Publisher Test Code

from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

import logging

import time

import argparse

import json

host = "a1cai3u3kh553t-ats.iot.us-east-2.amazonaws.com"

certPath = "/home/pi/4925Project/"

clientId = "photon"

topic = "doorbell"

# Init AWSIoTMQTTClient

myAWSIoTMQTTClient = None

myAWSIoTMQTTClient = AWSIoTMQTTClient(clientId)

myAWSIoTMQTTClient.configureEndpoint(host, 8883)

myAWSIoTMQTTClient.configureCredentials("{}AmazonRootCA1.cer".format(certPath), "{}iot-private.pem.key".format(certPath), "{}iot-certificate.pem.crt".format(certPath))

# AWSIoTMQTTClient connection configuration

myAWSIoTMQTTClient.configureAutoReconnectBackoffTime(1, 32, 20)

myAWSIoTMQTTClient.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing

myAWSIoTMQTTClient.configureDrainingFrequency(2) # Draining: 2 Hz

myAWSIoTMQTTClient.configureConnectDisconnectTimeout(10) # 10 sec

myAWSIoTMQTTClient.configureMQTTOperationTimeout(5) # 5 sec

myAWSIoTMQTTClient.connect()

# Publish to the same topic in a loop forever

loopCount = 0

while True:

message = {}

message['message'] = "Someone is at the door!"

message['sequence'] = loopCount

messageJson = json.dumps(message)

myAWSIoTMQTTClient.publish(topic, messageJson, 1)

print('Published topic %s: %s\n' % (topic, messageJson))

loopCount += 1

time.sleep(10)

myAWSIoTMQTTClient.disconnect()

# References

<https://aws.amazon.com/iot/>

<https://aws.amazon.com/iot-core/features/>

<https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html>

<https://docs.aws.amazon.com/aws-technical-content/latest/aws-overview/internet-of-things-services.html>

<https://www.raspberrypi.org/products/raspberry-pi-zero-w/>

<https://www.raspberrypi.org/products/camera-module-v2/>

<https://support.office.com/en-us/article/pop-imap-and-smtp-settings-for-outlook-com-d088b986-291d-42b8-9564-9c414e2aa040>

<https://www.raspberrypi.org/documentation/usage/camera/raspicam/raspistill.md>

<https://learn.adafruit.com/raspberry-pi-zero-creation/is-my-pi-zero-dead>

<https://medium.com/@sajithswa/connect-your-raspberry-pi-with-aws-iot-28920dbc0e88>