

# IOT PIPELINE FOR AWS RECOGNITION DEPLOYMENT

Larry Fuller and Blake Washburn

# PRESENTATION OUTLINE

- Objectives
- Project architecture
- Overview of services employed
- Camera
- IoT Core
- S3, SNS, and Lambda Communication Setup
- Lamda and Rekognition
- Updating IoT devices from AWS IoT Core
- Future work



# OBJECTIVES

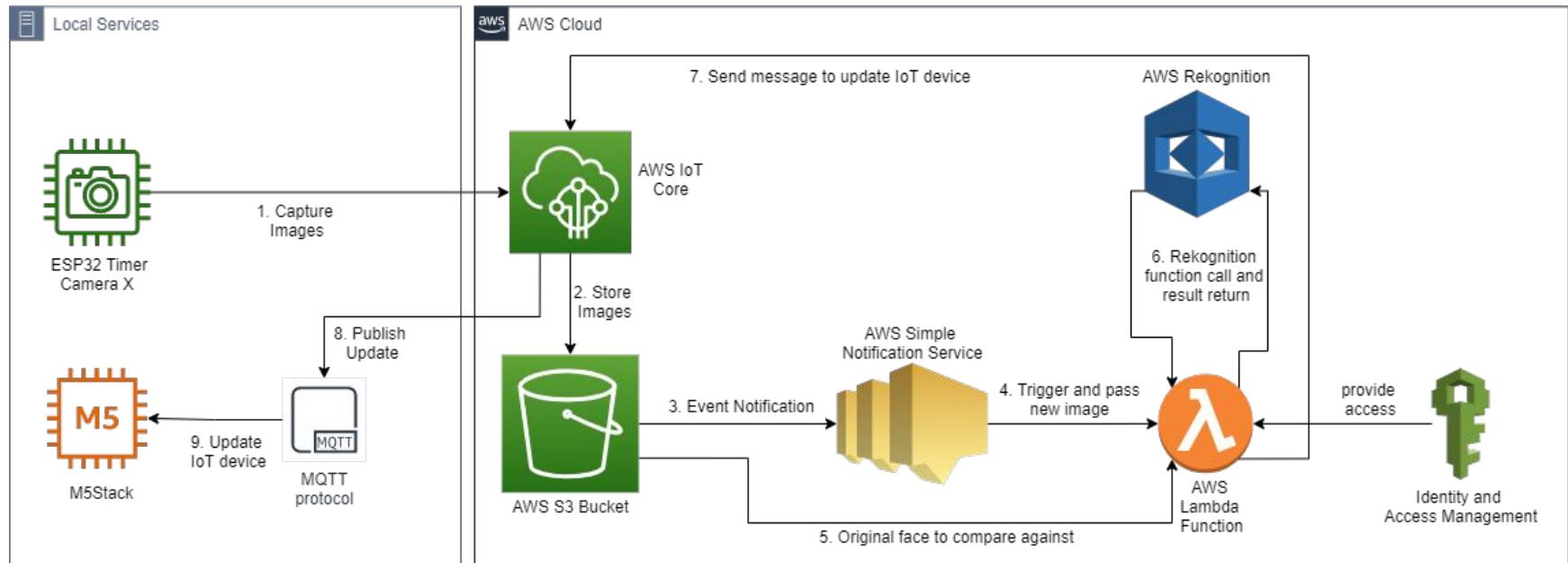
- 1. Demonstrate knowledge of cloud services concepts
- 2. Demonstrate use case for ~~ML and AI in~~ IoT and cloud computing
- 3. Demonstrate ~~low-cost proof~~ that can be used for undergraduate level training



# PROJECT PROPOSAL

- Original Project Proposal:  
Use an AWS SageMaker trained model with AWS Greengrass ML inference, and hardware including AWS IoT Core2 M5Stack with PSRAM Camera, to perform local facial recognition for home surveillance.
- Actual Project: Use an IoT Edge Device (Camera) to connect an end-to-end pipeline for the deployment of AWS Rekognition





# ARCHITECTURE



# DESCRIPTION OF SERVICES USED

- **ESP32 Timer Camera X**
  - Low-power camera integrated with ESP32 chip containing 8M PSRAM
- **AWS IoT Core**
  - IoT thing manager and message broker
- **AWS Simple Storage Service (S3)**
  - All-purpose, web-based storage service
- **AWS Simple Notification Service (SNS)**
  - Message deliver service allowing for publishers and subscribers
- **AWS Lambda**
  - Event-driven, serverless computing platform
- **AWS Rekognition**
  - Deep learning API that enables images classification
- **AWS Identity and Access Management**
  - Control and regulate access to AWS services and resources



CameraWebServer | Arduino 1.8.13

CameraWebServer \$ app\_httpd.cpp camera\_index.h camera\_pi.h

```
// Partial images will be transmitted if image exceeds buffer size

// Select camera model
#ifndef CAMERA_MODEL_WROVER_KIT // Has PSRAM
#ifndef CAMERA_MODEL_ESP_EYE // Has PSRAM
#define CAMERA_MODEL_MSSTACK_PSRAM // Has PSRAM
#define CAMERA_MODEL_MSSTACK_V2_PSRAM // MS Camera version B Has PSRAM
#define CAMERA_MODEL_MSSTACK_WIDE // Has PSRAM
#define CAMERA_MODEL_MSSTACK_ESP32CAM // No PSRAM
#define CAMERA_MODEL_AI_THINKER // Has PSRAM
#define CAMERA_MODEL_TTGO_T_JOURNAL // No PSRAM

#include "camera_pins.h"

const char* ssid = "iPhone";
```

Done uploading.

```
Writing at 0x001fc000... (96 %)
Writing at 0x001fc000... (97 %)
Writing at 0x00200000... (98 %)
Writing at 0x00204000... (99 %)
Writing at 0x00208000... (100 %)
Wrote 2594496 bytes (2073064 compressed) at 0x00010000 in 27.0 seconds.
Hash of data verified.
Compressed 3072 bytes to 119...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (119 compressed) at 0x00008000 in 0.0 seconds (effective)
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

13 M5Stack-Timer-CAM on /dev/cu.usbserial-3152740BF3



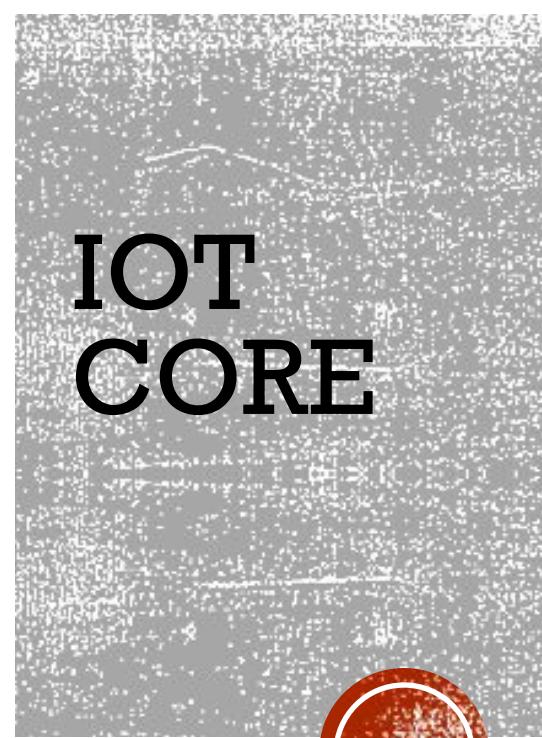
The screenshot shows the AWS CloudFormation console interface. On the left, a sidebar lists "Resource types" including ACMPCA, AccessAnalyzer, AmazonMQ, Amplify, AppGateway, AppGatewayV2, AppConfig, AppFlow, AppMesh, AppSync, ApplicationAutoScaling, ApplicationInsights, Athena, and AuditManager. Below this is a "template1" section with a "template.yaml" file and a "Choose template type" dropdown set to "AWS CloudFormation template". A preview area shows two stacks: "CopyLambda... Stack" and "Provision... Stack", with a dependency arrow from the latter to the former. On the right, the main workspace displays "Step 2: Specify stack details" for the "template1" stack. It includes a "Parameters (4)" section with a search bar and a table:

Key	Value
CreateProvisioningKey	true
QSS3BucketName	aws-quickstart
QSS3BucketRegion	us-east-1
QSS3KeyPrefix	quickstart-onica-connected-camera/

Below the parameters is a "Your AWS Account Status" summary box:

- Active** full access
- \$100** remaining credits (estimated)
- 2:60** session time

At the bottom are "Account Details" and "AWS Console" buttons.



AWS IoT > Policies

## Policies

Search policies

Name  ESP32Policy

IoT's certificate authority.

Create with CSR  
Upload your own certificate signing request (CSR) based on a private key you own. [Create](#)

Use my certificate  
Register your CA certificate and use your own certificates for one or many devices. [Get](#)

Skip certificate and create thing  
You will need to add a certificate to your thing later before your device can connect to AWS IoT. [Create thing](#)

Introducing the new AWS IoT console experience  
We're updating the console experience for you. Learn more [Try the new experiences and let us know what you think.](#)

Success  
Successfully created thing.

Success  
Successfully generated certificate. Please download certificate files.

Success  
Successfully activated certificate.

Certificate created!

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

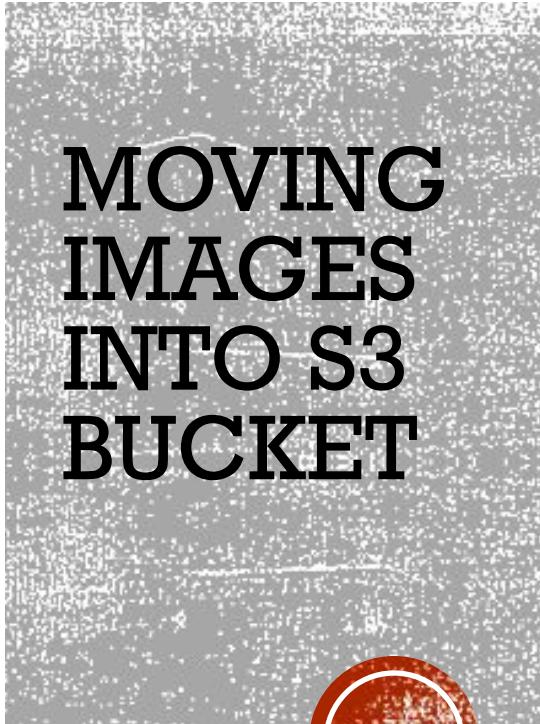
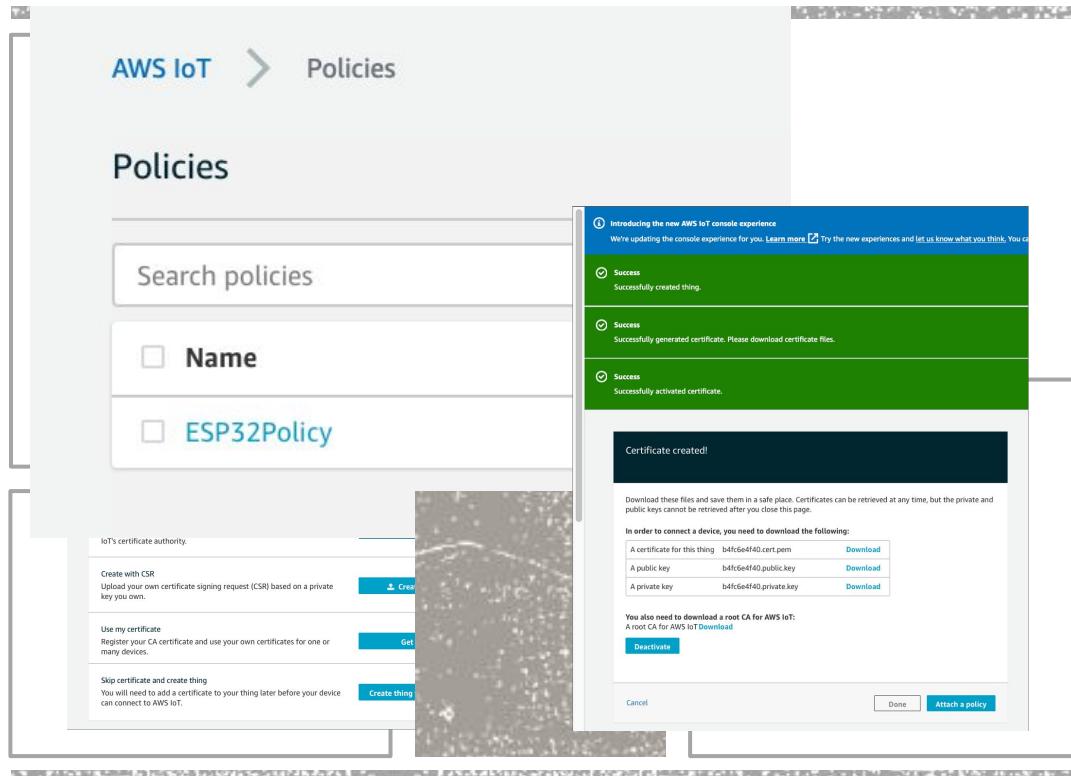
In order to connect a device, you need to download the following:

A certificate for this thing	b4fc6d4f40.cert.pem	<a href="#">Download</a>
A public key	b4fc6d4f40.public.key	<a href="#">Download</a>
A private key	b4fc6d4f40.private.key	<a href="#">Download</a>

You also need to download a root CA for AWS IoT:  
A root CA for AWS IoT [Download](#)

[Deactivate](#)

[Cancel](#) [Done](#) [Attach a policy](#)



Search for services, features, marketplace products, and docs [Option+S]   vocstartsoft/user987960=lwfulle@clemso

Amazon S3 > esp32-661616503187

## esp32-661616503187

Objects Properties Permissions Metrics Management Access Points

### Objects (0)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. You need to explicitly grant them permissions. [Learn more](#)

 Copy URL Delete Actions Create folder Upload

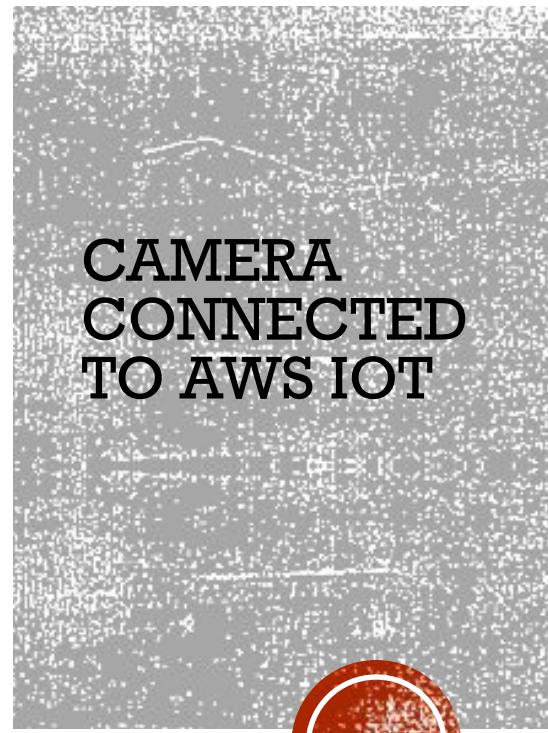
Name	Type	Last modified	Size
No objects			
You don't have any objects in this bucket.			
<a href="#">Upload</a>			



```
Received Message:Hello from hornbill ESP32 : 299
Publish Message:Hello from hornbill ESP32 : 300
Received Message:Hello from hornbill ESP32 : 300
Publish Message:Hello from hornbill ESP32 : 301
Received Message:Hello from hornbill ESP32 : 301
Publish Message:Hello from hornbill ESP32 : 302
Received Message:Hello from hornbill ESP32 : 302
Publish Message:Hello from hornbill ESP32 : 303
Received Message:Hello from hornbill ESP32 : 303

.....
WiFi connected
Starting web server on port: '80'
Starting stream server on port: '81'
Camera Ready! Use 'http://172.20.10.4' to connect
Connected to AWS
Subscribe Successfull
Received Message:{  
    "message": "Hello from AWS IoT console"  
}
```

Autoscroll  Show timestamp      [Newline](#)





1. Create S3 bucket
2. Create SNS service topic
3. Update topic's AccessPolicy so S3 can send events
4. Create event notification in bucket to notify topic
5. Create IAM role with S3 and Rekognition access
6. Create Lambda function
7. Set Lambda's execution role to IAM role
8. Add SNS trigger to Lambda function
9. Edit S3's BucketPolicy to allow services with IAM role to access its contents

# S3, SNS, AND LAMBDA COMMUNICATION SETUP



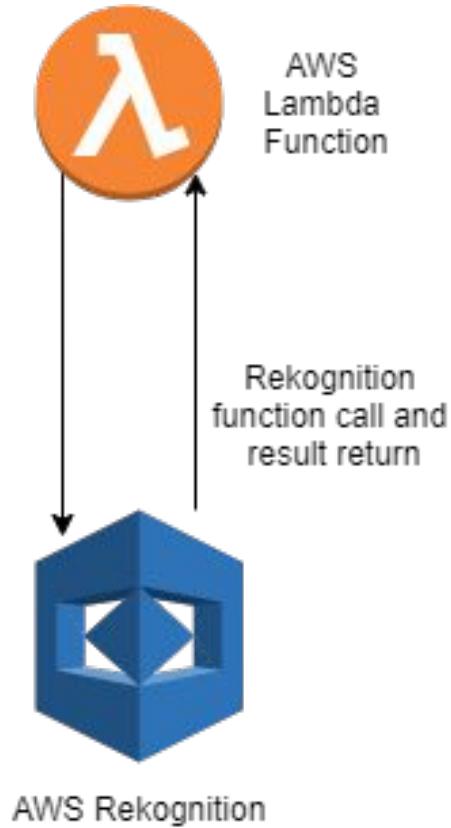


# S3, SNS, AND LAMBDA DEMO

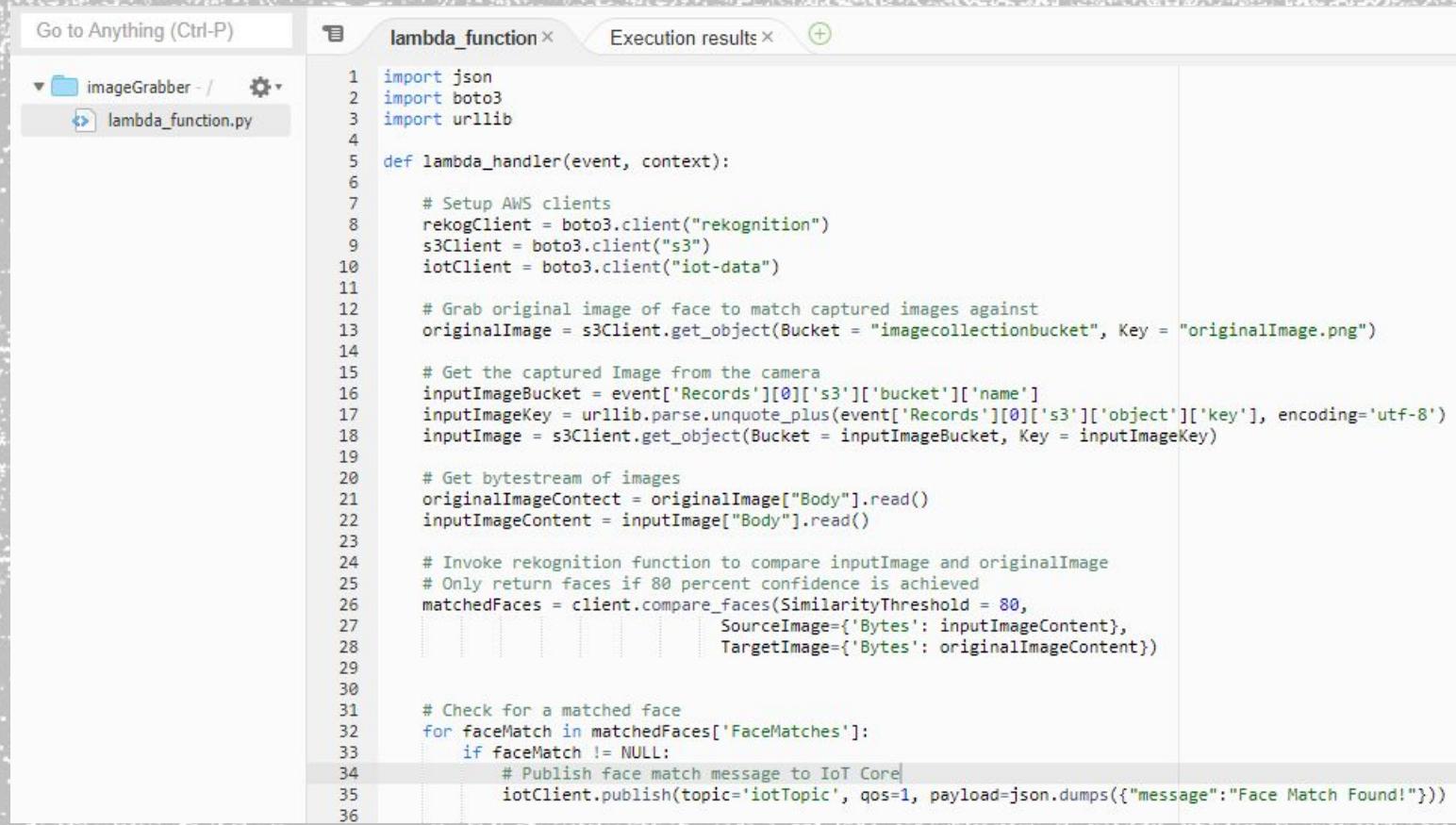


# LAMBDA AND RECOGNITION

- SNS triggers the execution of Lambda function
  - Passes the captured image as an event
- Lambda function grabs the captured image and the original image from S3
- Evokes Rekognition function compare\_faces
  - Returns matched faces with a confidence score
- If faces are found, publish message to IoT core for device updates



# LAMBDA AND RECOGNITION CODE



The screenshot shows a code editor interface with the following details:

- Project Structure:** The left sidebar shows a project named "imageGrabber" containing a folder "lambda\_function" which contains a file "lambda\_function.py".
- Code Editor:** The main area displays the contents of "lambda\_function.py".
- Code Content:** The code is a Python script for a Lambda function. It imports json, boto3, and urllib. It defines a lambda\_handler function that sets up AWS clients for rekognition, s3, and iot-data. It then grabs the original image from an S3 bucket ("imagecollectionbucket") and the captured image from the camera. It reads the byte streams of both images. It invokes the rekognition function to compare the images, setting a similarity threshold of 80%. It then checks for a matched face and publishes a message to an IoT Core topic ("iotTopic") if a match is found.

```
1 import json
2 import boto3
3 import urllib
4
5 def lambda_handler(event, context):
6
7     # Setup AWS clients
8     rekogClient = boto3.client("rekognition")
9     s3Client = boto3.client("s3")
10    iotClient = boto3.client("iot-data")
11
12    # Grab original image of face to match captured images against
13    originalImage = s3Client.get_object(Bucket = "imagecollectionbucket", Key = "originalImage.png")
14
15    # Get the captured Image from the camera
16    inputImageBucket = event['Records'][0]['s3']['bucket']['name']
17    inputImageKey = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')
18    inputImage = s3Client.get_object(Bucket = inputImageBucket, Key = inputImageKey)
19
20    # Get bytestream of images
21    originalImageContent = originalImage["Body"].read()
22    inputImageContent = inputImage["Body"].read()
23
24    # Invoke rekognition function to compare inputImage and originalImage
25    # Only return faces if 80 percent confidence is achieved
26    matchedFaces = client.compare_faces(SimilarityThreshold = 80,
27                                         SourceImage={'Bytes': inputImageContent},
28                                         TargetImage={'Bytes': originalImageContent})
29
30
31    # Check for a matched face
32    for faceMatch in matchedFaces['FaceMatches']:
33        if faceMatch != NULL:
34            # Publish face match message to IoT Core
35            iotClient.publish(topic='iotTopic', qos=1, payload=json.dumps({"message":"Face Match Found!"}))
36
```

# UPDATE IOT DEVICE FROM AWS IOT CORE

- Lambda publishes a message to IoT Core
- Next Steps:
  - AWS IoT core receives message
  - Publish MQTT message to M5Stack to take some action



# FUTURE WORK

- Get IoT Core working and publish messages to and from devices
- Separate incoming images and original faces into different datasets (S3 buckets)
- Enable code to search through a list of faces for a match rather than a single face from a single image
- Create custom IoT updates depending on which faces are detected



# GITHUB REPOSITORY

- [https://github.com/blekewashburn/cc4iot\\_FinalProject](https://github.com/blekewashburn/cc4iot_FinalProject)



# QUESTIONS?