IOT PIPELINE FOR AWS REKOGNITION 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

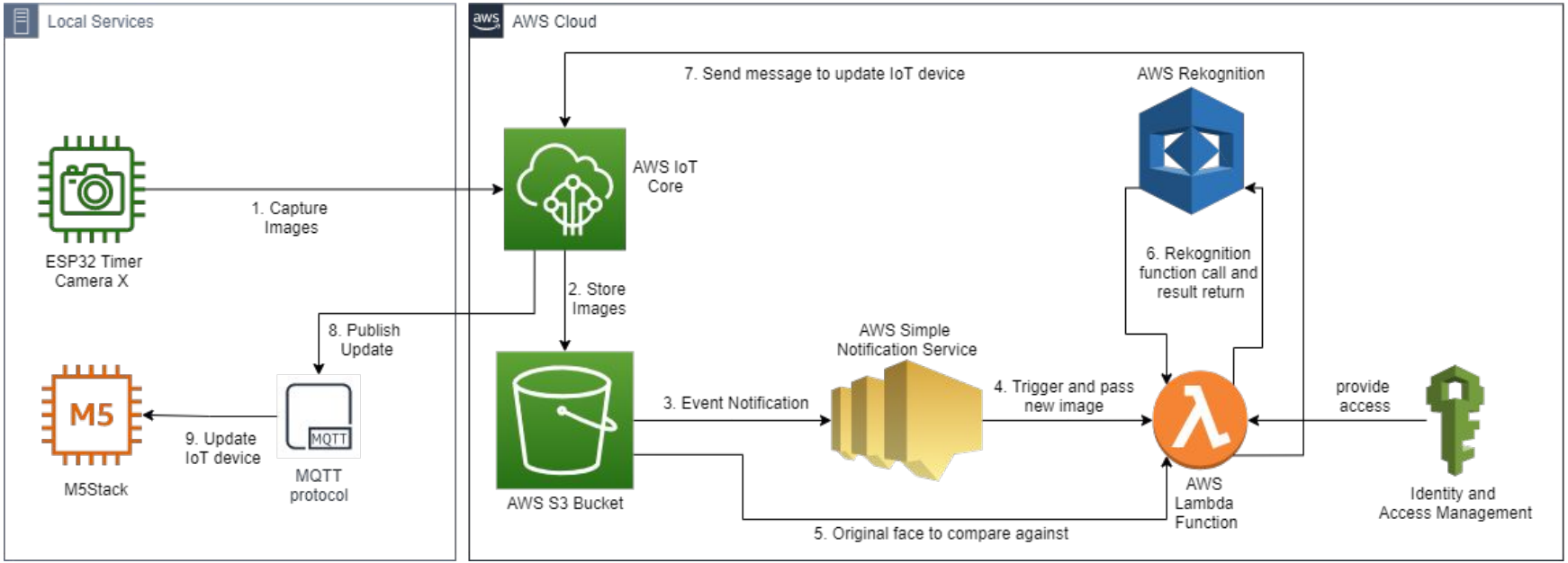
OBJECTIVES

- 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.

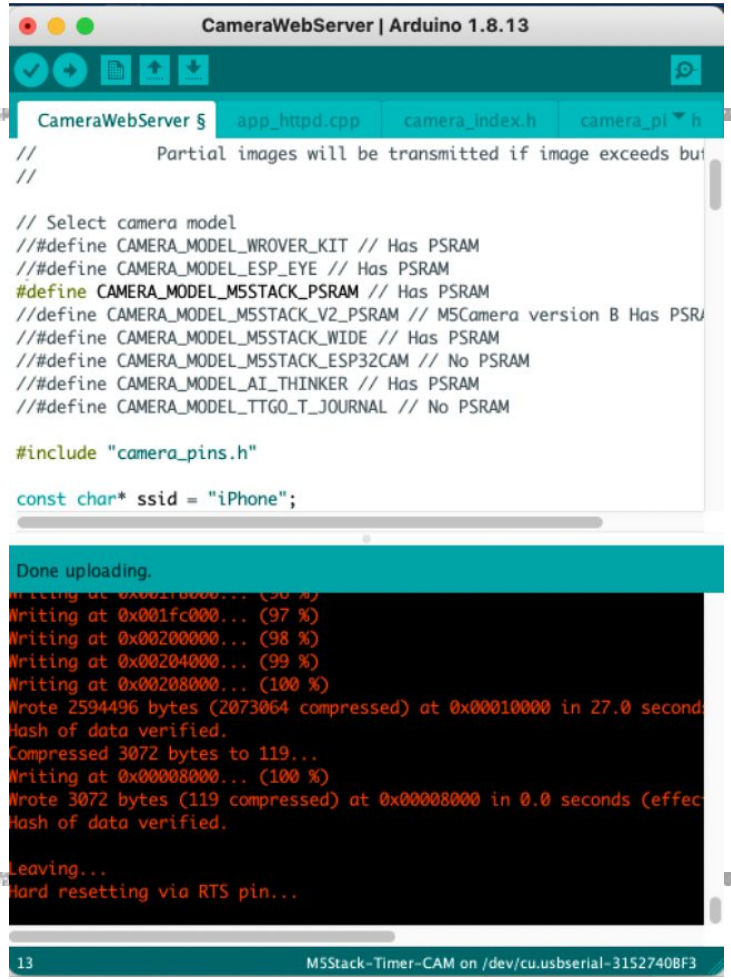
- Actual Project: Use an IoT Edge Device (Camera) to connect an end-to-end pipeline for the deployment of AWS Rekognition



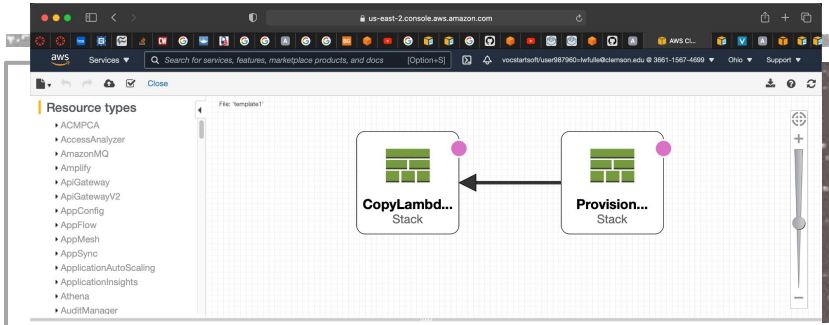
ARCHITECTURE

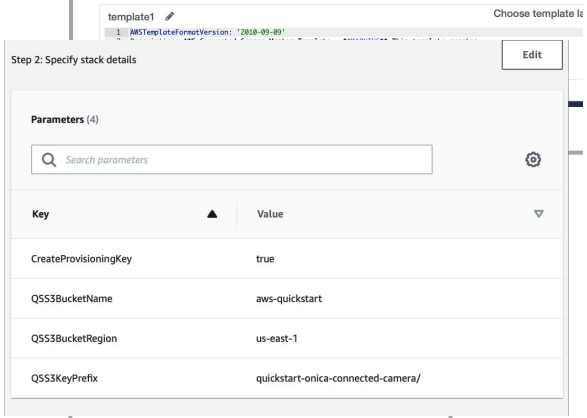
DESCRIPTION OF SERVICES USED

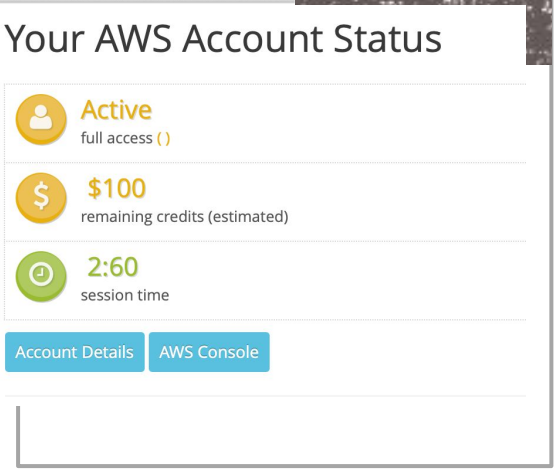
- ESP32 Timer Camera X
  - Low-power camera integrated with ESP32 chip containing 8M PSRAM
- AWS IoT Core
  - IoT thing manager and message broker
- AWS Simple Storage Service (S3)
  - All-purpose, web-based storage service
- AWS Simple Notification Service (SNS)
  - Message deliver service allowing 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



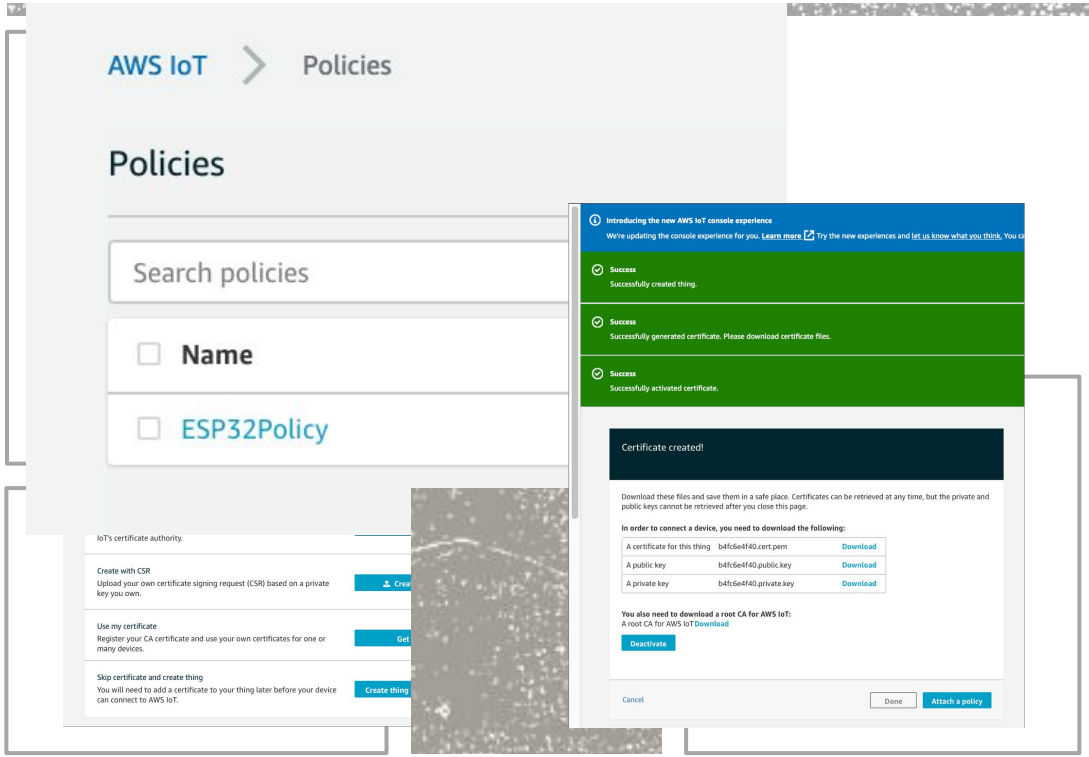
CAMERA SETUP


Resource types
ACMPCA
AccessAnalyzer
AmazonMQ
Amplify
ApiGateway
ApiGatewayV2
AppConfig
AppFlow
AppMesh
AppSync
ApplicationAutoScaling
ApplicationInsights
Athena
AuditManager
CopyLambd...
Stack
Provision...
Stack

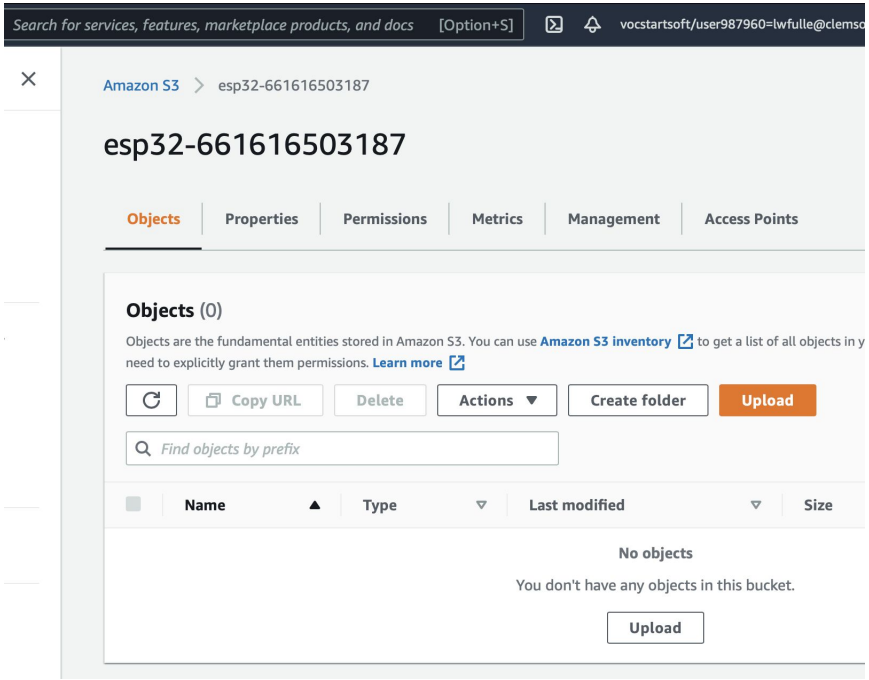

template1
Choose template
Step 2: Specify stack details
Edit
Parameters (4)
Search parameters
Key
Value
CreateProvisioningKey
true
QSS3BucketName
aws-quickstart
QSS3BucketRegion
us-east-1
QSS3KeyPrefix
quickstart-onica-connected-camera/


Your AWS Account Status
Active
full access ( )
$100
remaining credits (estimated)
2:60
session time
Account Details
AWS Console

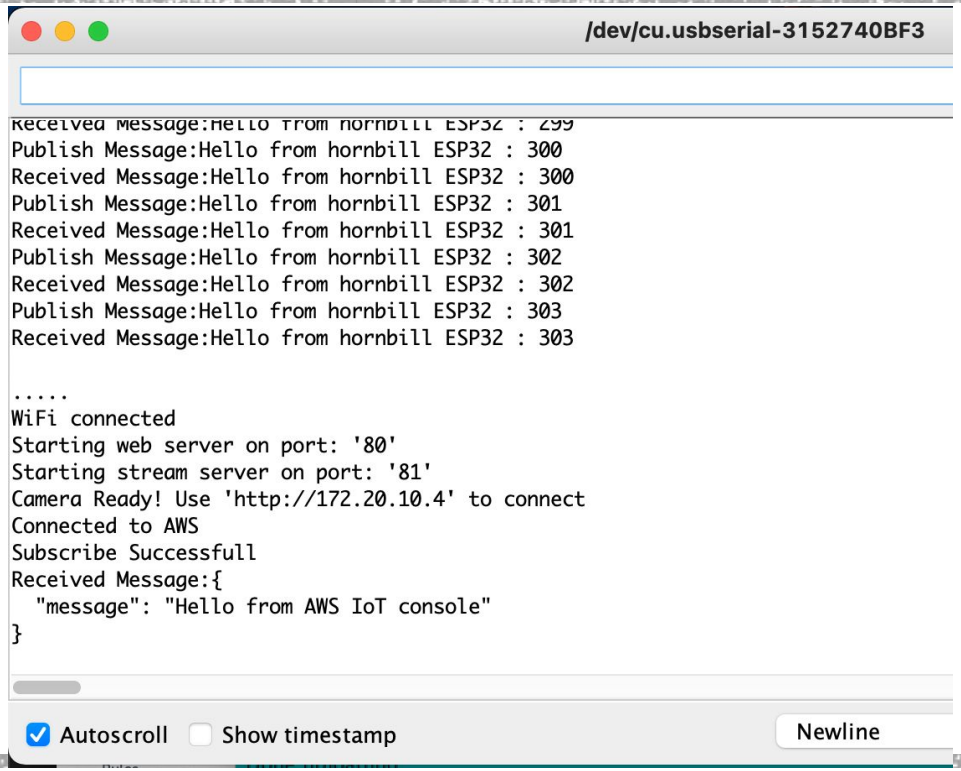
IOT CORE



MOVING IMAGES INTO S3 BUCKET



MOVING IMAGES INTO S3 BUCKET



CAMERA CONNECTED TO AWS IOT

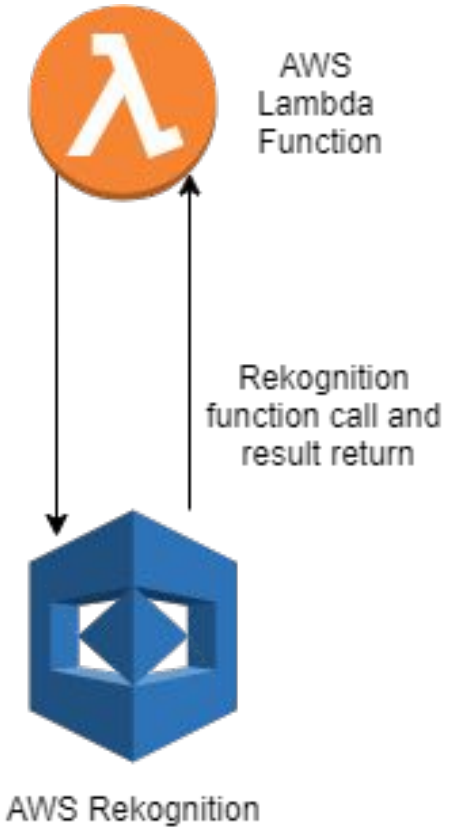
1. Create S3 bucket
2. Create SNS service topic
3. Update topic's AccessPolicy so S3 can send events
4. Create event notification in bucket to notify topic
5. Create IAM role with S3 and Rekognition access
6. Create Lambda function
7. Set Lambda's execution role to IAM role
8. Add SNS trigger to Lambda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

- 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 REKOGNITION CODE

Go to Anything (Ctrl-P)

imageGrabber - /

lambda_function.py

lambda_function × Execution results ×

```python
import json
import boto3
import urllib

def lambda_handler(event, context):

    # Setup AWS clients
    rekogClient = boto3.client("rekognition")
    s3Client = boto3.client("s3")
    iotClient = boto3.client("iot-data")

    # Grab original image of face to match captured images against
    originalImage = s3Client.get_object(Bucket = "imagecollectionbucket", Key = "originalImage.png")

    # Get the captured Image from the camera
    inputImageBucket = event['Records'][0]['s3']['bucket']['name']
    inputImageKey = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')
    inputImage = s3Client.get_object(Bucket = inputImageBucket, Key = inputImageKey)

    # Get bytestream of images
    originalImageContect = originalImage["Body"].read()
    inputImageContent = inputImage["Body"].read()

    # Invoke rekognition function to compare inputImage and originalImage
    # Only return faces if 80 percent confidence is achieved
    matchedFaces = client.compare_faces(SimilarityThreshold = 80,
                                SourceImage={'Bytes': inputImageContent},
                                TargetImage={'Bytes': originalImageContent})


    # Check for a matched face
    for faceMatch in matchedFaces['FaceMatches']:
        if faceMatch != NULL:
            # Publish face match message to IoT Core
            iotClient.publish(topic='iotTopic', qos=1, payload=json.dumps({"message":"Face Match Found!"}))

```

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 the which faces are detected

GITHUB REPOSITORY

- https://github.com/blakewashburn/cc4iot_FinalProject


QUESTIONS?