**Homework 12: AWS Internet of Things (Total Points: 100)**

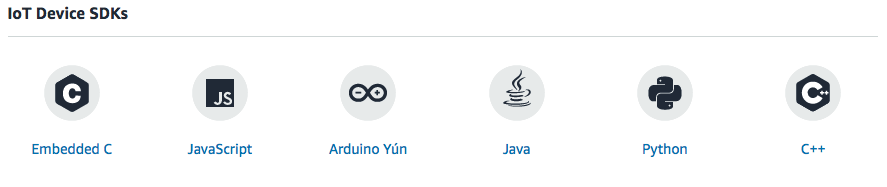
Due: Sunday December 9 11:59PM ET

**Problem 1 - Set Up a Device (40 points)**

Create a Thing that will represent a multisensor which transmits the following information:

* temperature
* humidity
* light
* motion

Technically your “thing” is going to be your computer in this setup, and you will use the AWS IoT SDK to send data to the IoT platform. You may use any of the IoT SDKs Amazon provides, which means you can select from:



The Python script demoed in lecture is also available for download and has everything you need to complete this step. The only modifications required will be 1) your configuration settings and 2) in how you publish your data.

The data will be provided in a separate CSV file. Your program should use the same data to publish to the message broker.

Paste in the box below the source code of your device simulator:

|  |
| --- |
| import csv  import random  import json  import time  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  from time import sleep  from datetime import date, datetime  CLIENT\_NAME = "HW12"  TOPIC = "HW12/1"  # Broker path is under AWS IoT > Settings (at the bottom left)  BROKER\_PATH = "a3uogk40zc917y-ats.iot.us-east-1.amazonaws.com"  # RSA 2048 bit key: Amazon Root CA 1 found here:  # https://docs.aws.amazon.com/iot/latest/developerguide/managing-device-certs.html  ROOT\_CA\_PATH = './AmazonRootCA1.pem'  PRIVATE\_KEY\_PATH = './b912f70074-private.pem.key'  CERTIFICATE\_PATH = './b912f70074-certificate.pem.crt'  IoTclient = AWSIoTMQTTClient(CLIENT\_NAME)  IoTclient.configureEndpoint(BROKER\_PATH, 8883)  IoTclient.configureCredentials(  ROOT\_CA\_PATH,  PRIVATE\_KEY\_PATH,  CERTIFICATE\_PATH  )  # Allow the device to queue infinite messages  IoTclient.configureOfflinePublishQueueing(-1)  # Number of messages to send after a connection returns  IoTclient.configureDrainingFrequency(2) # 2 requests/second  # How long to wait for a [dis]connection to complete (in seconds)  IoTclient.configureConnectDisconnectTimeout(10)  # How long to wait for publish/[un]subscribe (in seconds)  IoTclient.configureMQTTOperationTimeout(5)  IoTclient.connect()  IoTclient.publish(TOPIC, "connected", 0)  with open('multi-sensor-data.csv', newline='') as csvfile:  csv.reader(csvfile, delimiter=',', quotechar='"')  next(csvfile)  for row in csvfile:  time.sleep(1)  data = row.split(',')  i=0  while i < len(data):  data[i]=data[i].replace('"','')  data[i]=data[i].replace('\n','')  data[i]=data[i].replace('\r','')  i=i+1  payload = json.dumps({  "name": data[0],  "temp": data[1],  "humidity": data[2],  "luminosity": data[3],  "motion": data[4]  })  IoTclient.publish(TOPIC, payload, 0) |

Paste in the box below a screenshot of the topic subscriber displaying some of the received messages:

|  |
| --- |
|  |

**Problem 2 - Configure Two Rules (60 points)**

Now that you have a device successfully publishing data to the message broker, we should make use of that data in some meaningful way.

Rule 1

Set up a rule that looks for any messages where the temperature is above 50 degrees, and then writes that message to an S3 bucket.

Paste in the box below the SQL Query for your Rule.

|  |
| --- |
|  |

Paste in the box below the S3 Bucket Key for your bucket.

|  |
| --- |
| ${topic()}/${timestamp()} |

After feeding all data from your device to the message broker, what are the contents of your S3 Bucket? (Paste a screenshot of the bucket contents in the box below):

|  |
| --- |
|  |

Rule 2

Set up a rule that looks for any message indicating motion was detected, and send out a notification through an SNS topic. Remember, with SNS you will need to create a topic, and then create an email subscriber to that topic. The message does not need to be formatted, it just needs to send the data through an SNS notification.

Paste in the box below the SQL Query for your Rule.

|  |
| --- |
| SELECT \* from 'HW12/1' WHERE motion = 'TRUE' |

Paste in the box below a screenshot of a message you received through email. Make sure the from and to fields are visible in the screenshot:

|  |
| --- |
|  |

**Bonus (10 points)**

Create a rule that persists all device data in DynamoDB. Provide the details of your action configuration. Also show the data in your DynamoDB.

Save all data under primary key ‘name’ which is the timestamp of the message.

All data goes under the message field in the database for opaque data storage, and also because the setup is easier.

