ECEA 5348- Project 2

Implementation/Assumption Notes

Objective- Create an extended Amazon Web Services connection on top of project 1’s infrastructure that will:

1. Add a rule to the AWS IoT Thing to send incoming data to AWS Lambda, in addition to the existing rule for sending data to the SQS Queue.

2. Create a Lambda rule that routes the incoming messages to an AWS database of your choice.

3. Develop a Microservice that will provide a REST API to request data from the data stored in the AWS database. Decide how to request data (get the latest value, allow for a query, etc.). The Microservice will use AWS API Gateway and another Lambda program, as well as the AWS database you chose, to combine to provide this functionality.

4. Create a simple client in Python to exercise the REST API you created above, and demonstrate requesting and receiving data from the AWS database.

5. Data from the pseudo sensor is humidity between 0 and 100%, and temperature between -20 and 100 degrees Fahrenheit.

Additional Notes

1. IDE is set up with: AWS Cloud9, Win10, Python, MQTT, DynamoDB.

2. PDF submission will include:

   a. The data server tracing its connection to AWS

   b. The data server sending messages (JSON timestamps, temperature, and humidity values) to AWS

   c. The client connecting to the REST API

   d. The client showing the retrieved messages from API transactions

   e. Captures of failed communication for the server or client.

   f. Use of the previously developed HTML UI as an alternate client.

Publish.py (privacy keys removed)

```
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
from pseudoSensor import PseudoSensor
from datetime import datetime
import time
import boto3
import json
import os
import sys

#Define MQTT client
myMQTTClient = AWSIoTMQTTClient("dojodevice1")
myMQTTClient.configureEndpoint("a282wfrznzqkn3-ats.iot.us-east-2.amazonaws.com", 8883)
myMQTTClient.configureCredentials("./AmazonRootCA1 (1).pem","#####.pem.key", "#####.pem.crt")
myMQTTClient.connect()
print("Client Connected")

#Create Simple Notofication Service client with Boto3
client = boto3.client('sns',
region_name='us-east-2',aws_access_key_id='########',aws_secret_access_key='SecretSanta')
response = client.list_topics()

#Get timestamp
rn=datetime.now()
timein=str(rn.strftime("%H:%M:%S"))

#Generate weather value, store in 'foo'
ps=PseudoSensor()
for i in range(1):
h,t = ps.generate_values()
foo={"Time":timein, "Humidity":h,"Temperature":t}

#Reformat json to string, so it can be used by 'publish'
json_dump=json.dumps(foo)
msg = json_dump

#Publish to general/inbound topic
topic = "general/inbound"
myMQTTClient.publish(topic, msg, 0)

client.publish(TopicArn='arn:aws:sns:us-east-
2:305202504581:Weather',Message=msg,Subject='Weather
',)
print("Message Sent")
time.sleep(2)
myMQTTClient.disconnect()
print("Client Disconnected")
```

Lambda to DDB.py

```python
import boto3
import json
import time
from pprint import pprint
from decimal import Decimal
# Get the service resources
sqs = boto3.resource('sqs')
dynamodb= boto3.resource('dynamodb')
# Get the queue
queue = sqs.get_queue_by_name(QueueName='5318_Proj1')
table_name = 'WeatherTable'
#DynamoDB client code
dynamodb_client = boto3.client('dynamodb')
table = dynamodb.Table('WeatherTable')
def lambda_handler(event, context):
# Put item into db, combining sqs and dynamoDB resources
message_bodies=[]
messages_to_delete = []
# Put item into db, combining sqs and dynamoDB resources
for message in queue.receive_messages(MaxNumberOfMessages=10):

#Process message body until there are none left
weatherItem = json.loads(message.body, parse_float=Decimal)
message_bodies.append(weatherItem)
#Add message to delete
messages_to_delete.append({'Id': message.message_id,'ReceiptHandle': message.receipt_handle})

for i in message_bodies:
response = table.put_item(Item=i)

if len(messages_to_delete) == 0:
break
else:
delete_response = queue.delete_messages(Entries=messages_to_delete)
```

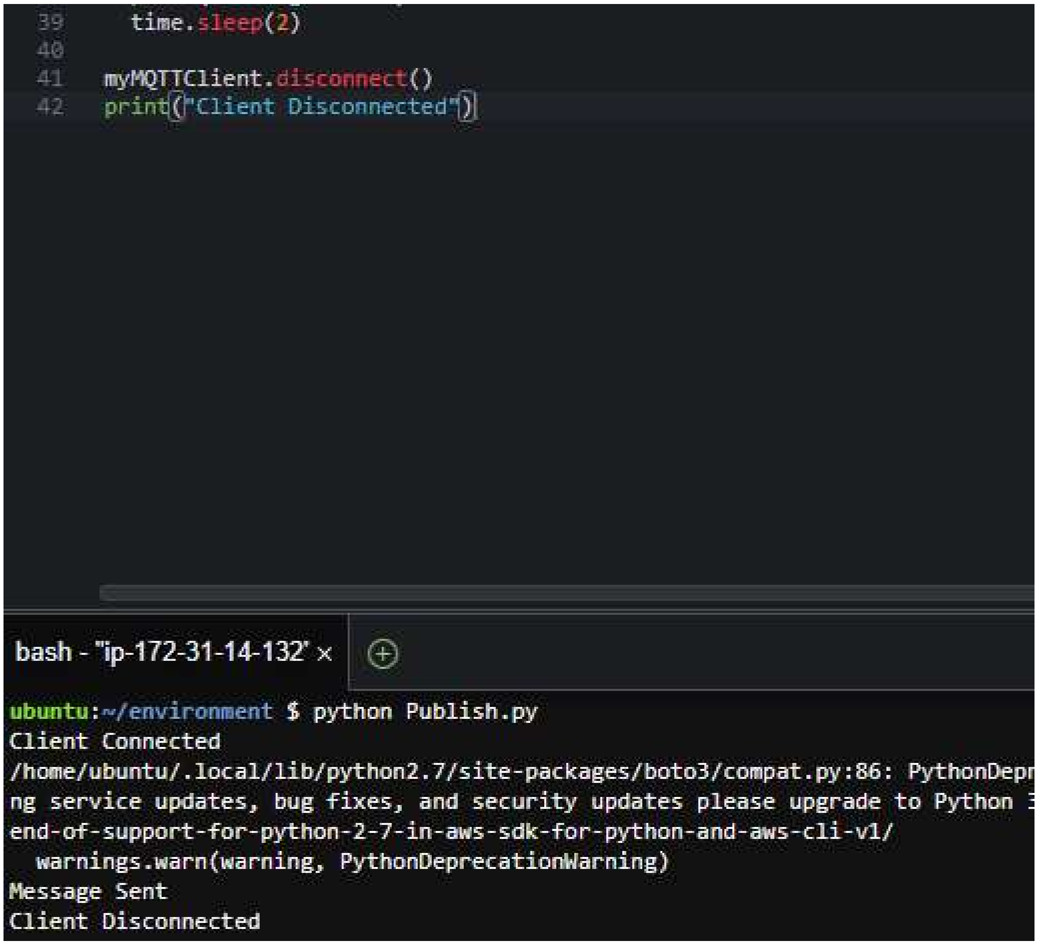
Lambda Gateway Send.py

```
import json
import boto3
from decimal import Decimal
dynamodb = boto3.resource('dynamodb', region_name='us-east-2')
table = dynamodb.Table('WeatherTable')
#Fix issue with json and decimals
class DecimalEncoder(json.JSONEncoder):
def default(self, obj):
if isinstance(obj, Decimal):
return float(obj)
return json.JSONEncoder.default(self, obj)
def lambda_handler(event, context):
# TODO implement
response = table.scan()
#Gimme my stuff
return {
'statusCode': 200,
'body': json.dumps(response['Items'], cls=DecimalEncoder)
}
```

Get Weather.py

```
import requests
response = requests.get("https://dvmjby15r5.execute-api.us-east-
2.amazonaws.com/default/Gate_Send")
print(response.text)
```

1. Data server tracing connection to AWS:

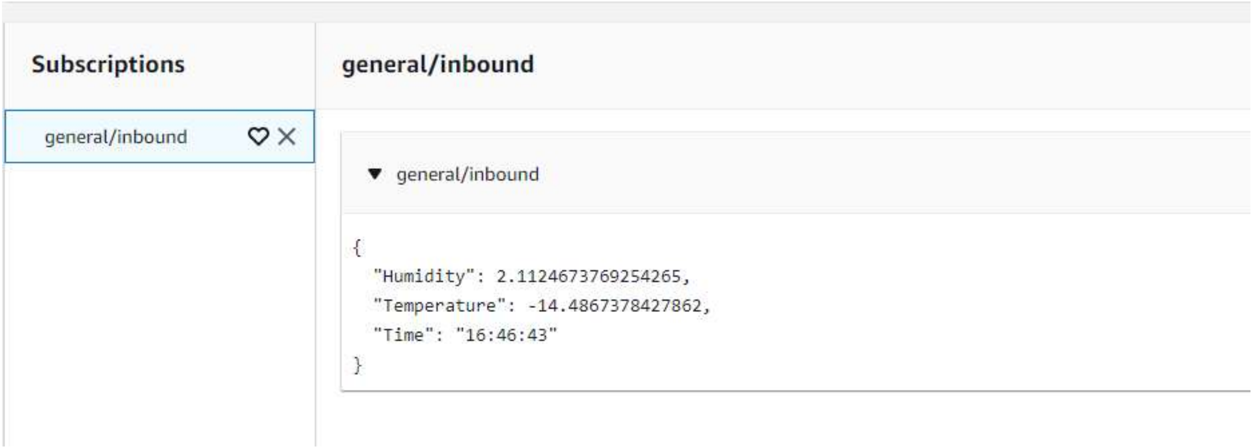


```
  time.sleep(2)

myMQTTClient.disconnect()
print("Client Disconnected")
```

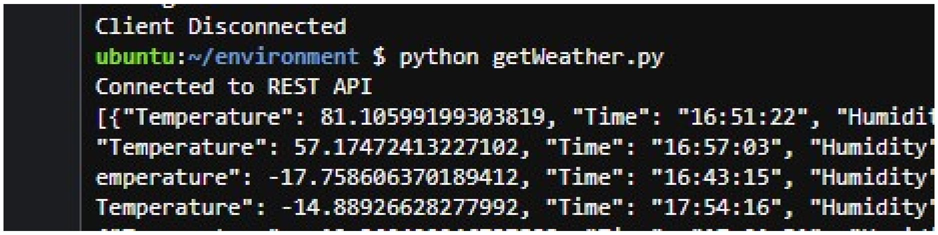
```
ubuntu:~/environment $ python Publish.py
Client Connected
/home/ubuntu/.local/lib/python2.7/site-packages/boto3/compat.py:86: PythonDepr
ng service updates, bug fixes, and security updates please upgrade to Python 3
end-of-support-for-python-2-7-in-aws-sdk-for-python-and-aws-cli-v1/
  warnings.warn(warning, PythonDeprecationWarning)
Message Sent
Client Disconnected
```

2. Server sending data to AWS:



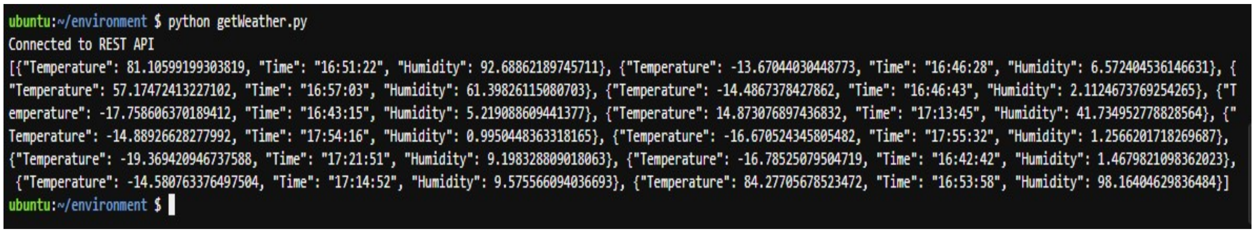
```
{
  "Humidity": 2.1124673769254265,
  "Temperature": -14.4867378427862,
  "Time": "16:46:43"
}
```

3. Client connecting to REST API:



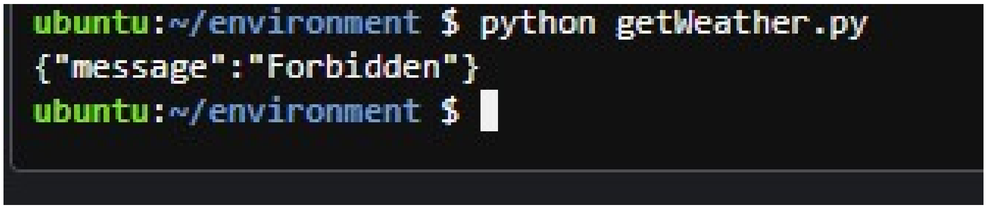
```
Client Disconnected
ubuntu:~/environment $ python getWeather.py
Connected to REST API
[{"Temperature": 81.10599199303819, "Time": "16:51:22", "Humidit
"Temperature": 57.17472413227102, "Time": "16:57:03", "Humidity"
emperature": -17.758606370189412, "Time": "16:43:15", "Humidity"
Temperature": -14.88926628277992, "Time": "17:54:16", "Humidity"
```

4. Client showing retrieved messages from API transactions:



```
ubuntu:~/environment $ python getWeather.py
Connected to REST API
[{"Temperature": 81.10599199303819, "Time": "16:51:22", "Humidity": 92.68862189745711}, {"Temperature": -13.67044030448773, "Time": "16:46:28", "Humidity": 6.57240453614663 1}, {
"Temperature": 57.17472413227102, "Time": "16:57:03", "Humidity": 61.39826115080703}, {"Temperature": -14.4867378427862, "Time": "16:46:43", "Humidity": 2.1124673769254265}, {"T
emperature": -17.758606370189412, "Time": "16:43:15", "Humidity": 5.219088609441377}, {"Temperature": 14.873076897436832, "Time": "17:13:45", "Humidity": 41.734952778828564}, {"
Temperature": -14.88926628277992, "Time": "17:54:16", "Humidity": 0.9950448363318165}, {"Temperature": -16.670524345805482, "Time": "17:55:32", "Humidity": 1.2566201718269687},
{"Temperature": -19.369420946737588, "Time": "17:21:51", "Humidity": 9.198328809018063}, {"Temperature": -16.78525079504719, "Time": "16:42:42", "Humidity": 1.4679821098362023},
 {"Temperature": -14.580763376497504, "Time": "17:14:52", "Humidity": 9.575566094036693}, {"Temperature": 84.27705678523472, "Time": "16:53:58", "Humidity": 98.16404629836484}]
ubuntu:~/environment $ 
```

5. Failed communication to server:



```
ubuntu:~/environment $ python getWeather.py
{"message":"Forbidden"}
ubuntu:~/environment $ 
```