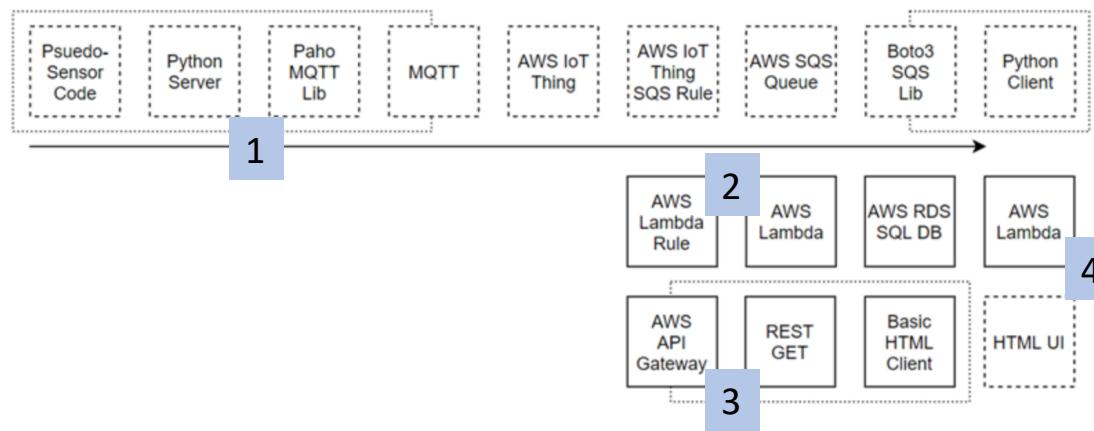


Project 2: Extended AWS IoT Connections

Kaylin Punotai

11 March 2021

Part 1. Implementation/Assumption Notes



1. Python server gets sensor readings and sends to AWS SQS
2. AWS Lambda rule subscribes first lambda to read incoming SQS messages and create entries in DynamoDB
3. A client sends REST GET request to API Gateway
4. API Gateway triggers second lambda to scan DynamoDB and return all entries to client

Part 2. Code: pseudoSensor.py

```
1 import random
2
3 class PseudoSensor:
4
5     h_range = [0, 20, 20, 40, 40, 60, 60, 80, 80, 90, 70, 70, 50, 50, 30, 30, 10, 10]
6
7     t_range = [-20, -10, 0, 10, 30, 50, 70, 80, 90, 80, 60, 40, 20, 10, 0, -10]
8
9     h_range_index = 0
10
11    t_range_index = 0
12
13    humVal = 0
14
15    tempVal = 0
16
17
18
19    def __init__(self):
20
21        self.humVal = self.h_range[self.h_range_index]
22
23        self.tempVal = self.t_range[self.t_range_index]
24
25
26
27    def generate_values(self):
28
29        self.humVal = self.h_range[self.h_range_index] + random.uniform(0, 10);
30
31        self.tempVal = self.t_range[self.t_range_index] + random.uniform(0, 10);
32
33        self.h_range_index += 1
34
35        if self.h_range_index > len(self.h_range) - 1:
36
37            self.h_range_index = 0
38
39        self.t_range_index += 1
40
41        if self.t_range_index > len(self.t_range) - 1:
42
43            self.t_range_index = 0
44
45    return self.humVal, self.tempVal
46
```

Each index has a
starter magnitude

Generated values are
randomized based on the
starter magnitude at each
index

This code is from Week 1 and
generates random humidity
and temperature readings

Part 2. Code: server.py

```
1 import sys, time
2 from datetime import datetime
3
4 from psuedoSensor import PseudoSensor
5 ps = PseudoSensor()          Import pseudoSensor
6
7 import paho.mqtt.client as paho
8 import os
9 import socket
10 import ssl
11 from time import sleep
12 from random import uniform
13
14 connflag = False  connflag indicates whether server.py is connected to MQTT
15
16 def on_connect(client, userdata, flags, rc):
17     global connflag
18     connflag = True
19     print("Connection returned result: " + str(rc) ) When connected, connflag becomes true
110 and connection is printed to console
20
21 def on_message(client, userdata, msg): MQTT messages print on the client console
22     print(msg.topic+" "+str(msg.payload))
23
24 mqttc = paho.Client()
25 mqttc.on_connect = on_connect
26 mqttc.on_message = on_message
```

```
28 awshost = AWS connection
29 awsport =
30 clientId =
31 thingName
32 caPath =
33 certPath =
34 keyPath =
35
36 mqttc.tls_set(caPath, certfile=certPath, keyfile=keyPath, cert_reqs=ssl.CERT_REQUIRED, tls_version=ssl.PROTOCOL_TLS
37
38 mqttc.connect(awshost, awsport, keepalive=60)
39
40 mqttc.loop_start()
41 i = 0      Iterate 10 times to send 10 readings
42 while i<10:
43     sleep(1)  Each reading has a 1 second delay
44     if connflag == True:
45         h,t = ps.generate_values()
46         now = datetime.now()
47         timestamp = now.strftime("%m/%d/%Y %H:%M:%S")
48         message = "{ \"time\": \"%s\", \"temperature\": \"% .5f\", \"humidity\": \"% .5f\" }" % (timestamp, t, h)
49         mqttc.publish("sensor", message, qos=1)
50         print("[ " + str(i) + "]\n" + message)
51     else:
52         print("waiting for connection...")
53     i += 1
```

Generate sensor readings and publish the time, temp, and humidity to MQTT

Part 2. Code: LambdaToDB.js

```
1 const AWS = require('aws-sdk');
2
3 AWS.config.update({endpoint: "https://dynamodb.us-west-2.amazonaws.com"});
4
5 var dynamo = new AWS.DynamoDB();
6
7 exports.handler = (event) => {
8     event.Records.forEach(record => {
9
10         var parser = JSON.parse(record.body);
11
12         var params = {
13             TableName: "pseudosensor",
14             Item: {
15                 "id": {S: record.messageId},
16                 "time": {S: parser.time},
17                 "temperature": {N: parser.temperature},
18                 "humidity": {N: parser.humidity}
19             }
20         };
21
22         dynamo.putItem(params, function(err, data) {
23             if (err) {
24                 console.error("Unable to add item. Error JSON:", JSON.stringify(err, null, 2));
25             } else {
26                 console.log("Added item:", JSON.stringify(data, null, 2));
27             }
28         });
29     });
30 });
31 };
32 }
```

New SQS messages trigger this Lambda (based on lambda rule), which parses the message and adds them to the "pseudosensor" table

Message body contains time, temp, and humidity. ID comes from the SQS message ID

Part 2. Code: LambdaToClient.js

This Lambda is triggered by the API Gateway, which sends REST requests to the Lambda

```
1 const AWS = require('aws-sdk');
2
3 const dynamo = new AWS.DynamoDB.DocumentClient();
4
5 exports.handler = async (event, context) => {
6     //console.log('Received event:', JSON.stringify(event, null, 2));
7     let body;
8     let statusCode = '200';
9     const headers = {
10         'Content-Type': 'application/json',
11     };
12
13     try {
14         switch (event.httpMethod) {    REST methods
15             case 'DELETE':
16                 body = await dynamo.delete(JSON.parse(event.body)).promise();
17                 break;
18             case 'GET':
19                 body = await dynamo.scan({ TableName:"pseudosensor" }).promise();
20                 break;
21             case 'POST':
22                 body = await dynamo.put(JSON.parse(event.body)).promise();
23                 break;
24             case 'PUT':
25                 body = await dynamo.update(JSON.parse(event.body)).promise();
26                 break;
27             default:
28                 throw new Error(`Unsupported method "${event.httpMethod}"`);
29     }
```

```
30     }
31     catch (err) {
32         statusCode = '400';
33         body = err.message;
34     } finally {
35         body = JSON.stringify(body);
36     }
37     return { body };
38 };
39 }
```

For "GET" requests,
the Lambda returns all
entries from the
"pseudosensor" table

Part 2. Code: client.py

Client sends the “GET” response
and prints all “pseudosensor”
entries to the console

```
1 import requests
2
3 URL = "https:// URL for API Gateway i.us-west-2.amazonaws.com/default/LambdaToClient"
4
5 response = requests.request("GET", URL)
6
7 print(response.text)
```

Part 3. Demonstration

Server.py connecting to AWS
and sending JSON messages

```
pi@raspberrypi:~/Desktop/Project_2 $ python server.py
Connection returned result: 0
[0]
{ "time": "03/11/2021 10:16:31", "temperature": "-17.53123", "humidity": "0.46402" }
[1]
{ "time": "03/11/2021 10:16:32", "temperature": "-5.77072", "humidity": "21.34948" }
[2]
{ "time": "03/11/2021 10:16:33", "temperature": "2.42055", "humidity": "28.09472" }
[3]
{ "time": "03/11/2021 10:16:34", "temperature": "12.89177", "humidity": "43.03383" }
[4]
{ "time": "03/11/2021 10:16:35", "temperature": "38.09131", "humidity": "45.42734" }
[5]
{ "time": "03/11/2021 10:16:36", "temperature": "54.30162", "humidity": "67.29430" }
[6]
{ "time": "03/11/2021 10:16:37", "temperature": "78.98567", "humidity": "63.61119" }
[7]
{ "time": "03/11/2021 10:16:38", "temperature": "81.82157", "humidity": "80.81345" }
[8]
{ "time": "03/11/2021 10:16:39", "temperature": "94.47528", "humidity": "83.58183" }
[9]
{ "time": "03/11/2021 10:16:40", "temperature": "89.05789", "humidity": "94.97806" }
pi@raspberrypi:~/Desktop/Project_2 $
```

SQS receives all messages and triggers LambdaToDB to create entries in “pseudosensor” table

This LambdaToDB CloudWatch trace shows entries were successfully added to the table

▶	2021-03-11T10:16:38.523-08:00	START RequestId: 981a8c83-61a9-53ae-b601-d447016833a5 Version: \$LATEST
▶	2021-03-11T10:16:38.556-08:00	2021-03-11T18:16:38.556Z 981a8c83-61a9-53ae-b601-d447016833a5 INFO Added item: {}
▶	2021-03-11T10:16:38.557-08:00	END RequestId: 981a8c83-61a9-53ae-b601-d447016833a5
▶	2021-03-11T10:16:38.557-08:00	REPORT RequestId: 981a8c83-61a9-53ae-b601-d447016833a5 Duration: 31.60 ms Billed Duration: 32 ms M
▶	2021-03-11T10:16:39.523-08:00	START RequestId: dd63b2d1-67be-58d5-83b5-ea1507038f97 Version: \$LATEST
▶	2021-03-11T10:16:39.576-08:00	2021-03-11T18:16:39.576Z dd63b2d1-67be-58d5-83b5-ea1507038f97 INFO Added item: {}
▶	2021-03-11T10:16:39.577-08:00	END RequestId: dd63b2d1-67be-58d5-83b5-ea1507038f97
▶	2021-03-11T10:16:39.577-08:00	REPORT RequestId: dd63b2d1-67be-58d5-83b5-ea1507038f97 Duration: 51.22 ms Billed Duration: 52 ms M
▶	2021-03-11T10:16:40.566-08:00	START RequestId: 32796984-ff49-5f9e-9d3f-31409f8cfe67 Version: \$LATEST
▶	2021-03-11T10:16:40.594-08:00	2021-03-11T18:16:40.594Z 32796984-ff49-5f9e-9d3f-31409f8cfe67 INFO Added item: {}
▶	2021-03-11T10:16:40.616-08:00	END RequestId: 32796984-ff49-5f9e-9d3f-31409f8cfe67
▶	2021-03-11T10:16:40.616-08:00	REPORT RequestId: 32796984-ff49-5f9e-9d3f-31409f8cfe67 Duration: 46.86 ms Billed Duration: 47 ms M
No newer events at this moment. <i>Auto retry paused.</i> Resume		

This is the “pseudosensor” table in DynamoDB

Scan: [Table] pseudosensor: id ^

Scan [Table] pseudosensor: id ^

+ Add filter

Start search

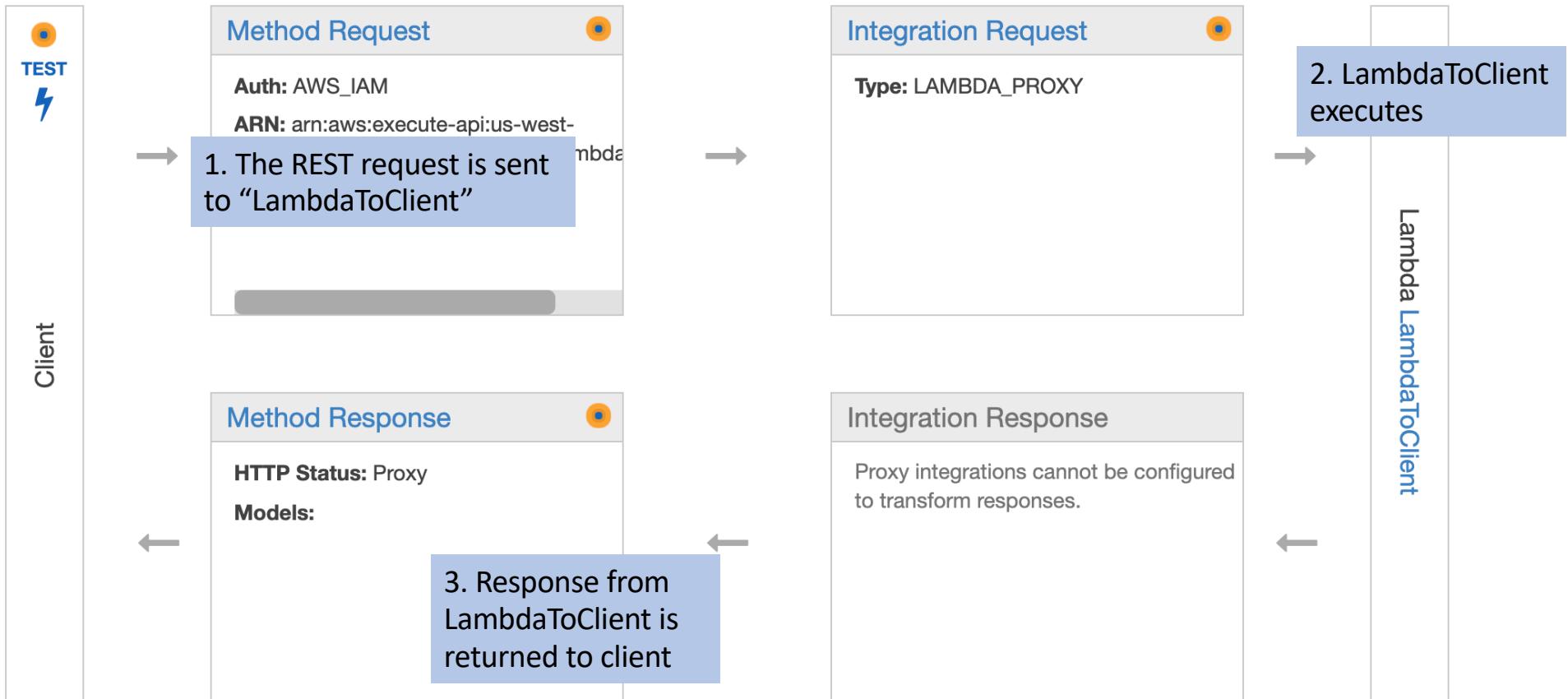
id is the messageID from SQS

	id ⓘ	humidity	temperature	time
	17c1db40-7eb7-4895-9cc0-2693049d2b57	63.61119	78.98567	03/11/2021 10:16:37
	1af4f6ae-35ea-4dde-acee-3f9277935724	0.46402	-17.53123	03/11/2021 10:16:31
	1c35ee2b-2a65-4667-ae19-37ef6103049d	67.2943	54.30162	03/11/2021 10:16:36
	36302a67-0bc5-48af-8d91-399390c46a2f	45.42734	38.09131	03/11/2021 10:16:35
	9c5ed57a-6962-4db3-ac5f-7a56f8bf362a	83.58183	94.47528	03/11/2021 10:16:39
	9cbc3b46-b059-4c9c-b20f-8a7f585d6f30	43.03383	12.89177	03/11/2021 10:16:34
	9d891639-9985-4d33-a343-2ba25f4400f1	94.97806	89.05789	03/11/2021 10:16:40
	c0d75927-ef7c-4d54-92c9-72a8abfdf290	80.81345	81.82157	03/11/2021 10:16:38
	cf497328-787c-452b-abd2-4a0bfd5779c1	28.09472	2.42055	03/11/2021 10:16:33
	d1f1a1b0-2161-1022-0150-20165051769d	21.21010	5.77070	03/11/2021 10:16:22

All entries were successfully entered in table

This is the API test for LambdaToClient

/LambdaToClient - ANY - Method Execution



This is the result of the API test

[Method Execution](#)

/LambdaToClient - ANY - Method Test



Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method

Method

GET

Request: /LambdaToClient

Status: 200

Latency: 717 ms

Path

No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.

Response Body

Using method “GET”, all entries are returned to the client

```
{
  "Items": [
    {
      "humidity": 28.09472,
      "time": "03/11/2021 10:16:33",
      "id": "cf497328-787c-452b-abd2-4a0bfd5779c1",
      "temperature": 2.42055
    },
    {
      "humidity": 45.42734,
      "time": "03/11/2021 10:16:35",
      "id": "36302a67-0bc5-48af-8d91-399390c46a2f",
      "temperature": 38.09131
    },
    {
      "humidity": 83.58183,
      "time": "03/11/2021 10:16:39",
      "id": "9c5ed57a-6962-4db3-ac5f-7a56f8bf362a",
      "temperature": 94.47528
    }
  ]
}
```

Query Strings

{LambdaToClient}

param1=value1¶m2=value2

Headers

{LambdaToClient}

Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg.
Accept:application/json.

Stage Variables

When the client Python runs, it invokes the API and prints the response to the console

```
pi@raspberrypi:~/Desktop/Project 2 $ python client.py
{"Items": [{"humidity": 28.09472, "time": "03/11/2021 10:16:33", "id": "cf497328-787c-452b-abd2-4a0bfd5779c1", "temperature": 2.42055}, {"humidity": 45.42734, "time": "03/11/2021 10:16:35", "id": "36302a67-0bc5-48af-8d91-399390c46a2f", "temperature": 38.09131}, {"humidity": 83.58183, "time": "03/11/2021 10:16:39", "id": "9c5ed57a-6962-4db3-ac5f-7a56f8bf362a", "temperature": 94.47528}, {"humidity": 67.2943, "time": "03/11/2021 10:16:36", "id": "1c35ee2b-2a65-4667-ae19-37ef6103049d", "temperature": 54.30162}, {"humidity": 0.46402, "time": "03/11/2021 10:16:31", "id": "1af4f6ae-35ea-4dde-acee-3f9277935724", "temperature": -17.53123}, {"humidity": 80.81345, "time": "03/11/2021 10:16:38", "id": "c0d75927-ef7c-4d54-92c9-72a8abfdf290", "temperature": 81.82157}, {"humidity": 43.03383, "time": "03/11/2021 10:16:34", "id": "9cbc3b46-b059-4c9c-b20f-8a7f585d6f30", "temperature": 12.89177}, {"humidity": 63.61119, "time": "03/11/2021 10:16:37", "id": "17c1db40-7eb7-4895-9cc0-2693049d2b57", "temperature": 78.98567}, {"humidity": 21.34948, "time": "03/11/2021 10:16:32", "id": "d1f1e4b9-2164-4823-915e-324659517e8d", "temperature": -5.77072}, {"humidity": 94.97806, "time": "03/11/2021 10:16:40", "id": "9d891639-9985-4d33-a343-2ba25f4400f1", "temperature": 89.05789}], "Count": 10, "ScannedCount": 10}
```