**Lambda**

**-AWS Lambda is a compute service that lets you run code without provisioning or managing servers.**

**June 5-Thursday**

1. Create an AWS IAM Policy to Grant AWS Lambda Access to an Amazon DynamoDB Table
2. Go to IAM services-->roles-->AWS access policies
3. The lambda function uses an IAM role that has an IAMpolicy attached that grants access to DynamoDB
4. Created a lambda function in AWS Lambda - Adding new data to DynamoDB table

import boto3 def lambda\_handler(event, context):

# this will create dynamodb resource object and

# here dynamodb is resource name

client = boto3.resource('dynamodb')

# this will search for dynamoDB table

# your table name may be different

table = client.Table("dynamoDB")

print(table.table\_status)

table.put\_item(Item= {'id': '34','company': 'microsoft'})

**June 12**

1.IAM role policies

2. Created **scrapeYelpData** Lambda function to insert yelp API- restaurant details to DynamoDB table called yelp\_restaurant

**scrapeYelpData (python 2.7)**

**import boto3**

**import datetime**

**import json**

**from time import sleep**

**from decimal import \***

**from botocore.vendored import requests**

**from urlparse import urljoin**

**API\_KEY = "T-K64h1YLhBnreA69A0zy\_ZPk518bg0\_TFzlaSL2DGt2vhb3XCpsUxDiKFDR2qQTa2yIEayQqECso-wxMotP5bUAJz\_GmE0e4lp0iDIisBAQlmfX8TtaKhfbFmXWXnYx"**

**dynamodb = boto3.resource('dynamodb', region\_name='us-east-1')**

**table = dynamodb.Table('yelp\_restaurant')**

**API\_HOST = 'https://api.yelp.com'**

**SEARCH\_PATH = '/v3/businesses/search'**

**TOKEN\_PATH = '/oauth2/token'**

**GRANT\_TYPE = 'client\_credentials'**

**# Defaults for our simple example.**

**DEFAULT\_TERM = 'dinner'**

**DEFAULT\_LOCATION = 'Toronto'**

**restaurants = {}**

**def search(cuisine, offset):**

**url\_params = {**

**'location': DEFAULT\_LOCATION,**

**'offset': offset,**

**'limit': 50,**

**'term': cuisine + " restaurants",**

**'sort\_by': 'rating'**

**}**

**return request(API\_HOST, SEARCH\_PATH, url\_params=url\_params)**

**def request(host, path, url\_params=None):**

**url\_params = url\_params or {}**

**url = urljoin(host, path)**

**headers = {**

**'Authorization': 'Bearer ' + API\_KEY,**

**}**

**response = requests.request('GET', url, headers=headers, params=url\_params)**

**rjson = response.json()**

**# business\_list = rjson['businesses']**

**return rjson**

**def addItems(data, cuisine):**

**global restaurants**

**with table.batch\_writer() as batch:**

**for rec in data:**

**try:**

**if rec["alias"] in restaurants:**

**continue;**

**rec["rating"] = Decimal(str(rec["rating"]))**

**restaurants[rec["alias"]] = 0**

**rec['cuisine'] = cuisine**

**rec['insertedAtTimestamp'] = str(datetime.datetime.now())**

**rec["coordinates"]["latitude"] = Decimal(str(rec["coordinates"]["latitude"]))**

**rec["coordinates"]["longitude"] = Decimal(str(rec["coordinates"]["longitude"]))**

**rec['address'] = rec['location']['display\_address']**

**rec.pop("distance", None)**

**rec.pop("location", None)**

**rec.pop("transactions", None)**

**rec.pop("display\_phone", None)**

**rec.pop("categories", None)**

**if rec["phone"] == "":**

**rec.pop("phone", None)**

**if rec["image\_url"] == "":**

**rec.pop("image\_url", None)**

**# print(rec)**

**batch.put\_item(Item=rec)**

**sleep(0.001)**

**except Exception as e:**

**print(e)**

**print(rec)**

**def scrape():**

**cuisines = ['italian', 'chinese', 'indian', 'american', 'mexican', 'spanish', 'greek', 'latin', 'Persian']**

**for cuisine in cuisines:**

**offset = 0**

**while offset < 1000:**

**js = search(cuisine, offset)**

**addItems(js["businesses"], cuisine)**

**#print(cuisine)**

**#print(js["businesses"][0])**

**#print("----------------")**

**offset += 50**

**def lambda\_handler(event, context):**

**scrape()**

**return {**

**'statusCode': 200,**

**'body': json.dumps('Hello from Lambda!')**

**}**

1. **Error**

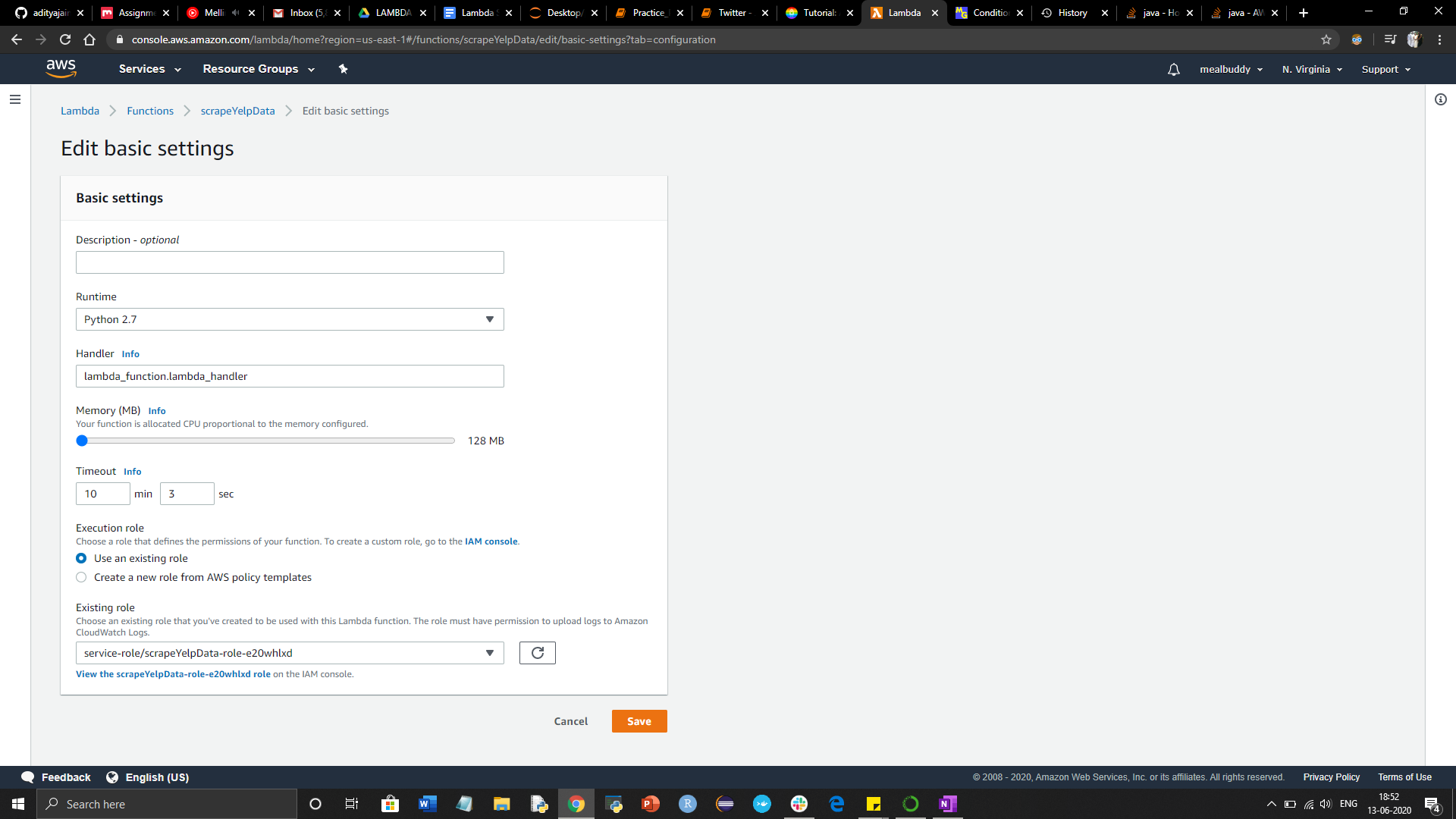
**when I run the code in Lambda, I get the following error:**

**Task timed out after 15.00 seconds**

1. **Solution**

**Increase the timeout for AWS Lambda Client**

**Permissions--->Edit basic settings**

****

**JUNE 18**

**LF2**

**Get data from SQS**

[**https://github.com/tainanboy/dining\_chatbot/blob/master/LF2.py**](https://github.com/tainanboy/dining_chatbot/blob/master/LF2.py)

1. **-pushing data from lex (lf2) to sqs**
2. **-read data from sqs(LF2)**

**SQStest- lambda function to get messages from SQS queue.**

**SQStest**

**import json**

**import boto3**

**from botocore.vendored import requests**

**from boto3.dynamodb.conditions import Key, Attr**

**import logging**

**logger = logging.getLogger()**

**logger.setLevel(logging.DEBUG)**

**def lambda\_handler(event, context):**

**sqs = boto3.client('sqs')**

**# Get URL for SQS queue**

**response = sqs.get\_queue\_url(QueueName='LF1SQSLF2')**

**queue\_url = response['QueueUrl']**

**#print(queue\_url)**

**message = None**

**# Receive a message from SQS queue**

**response = sqs.receive\_message(**

**QueueUrl="https://sqs.us-east-1.amazonaws.com/702727121783/LF1SQSLF2",**

**AttributeNames=[**

**'SentTimestamp'**

**],**

**MaxNumberOfMessages=1,**

**MessageAttributeNames=[**

**'All'**

**],**

**VisibilityTimeout=0,**

**WaitTimeSeconds=0**

**)**

**logger.debug(response)**

**message = response**

**'''**

**try:**

**message = response['Messages'][0]**

**receipt\_handle = message['ReceiptHandle']**

**# Delete received message from queue**

**sqs.delete\_message(**

**QueueUrl=queue\_url,**

**ReceiptHandle=receipt\_handle**

**)**

**#print('Received and deleted message: %s' % message)**

**'''**

**#message=event**

**"""location = message['MessageAttributes']['location']['StringValue']**

**cuisine = message['MessageAttributes']['cuisine']['StringValue']**

**dining\_date = message['MessageAttributes']['dining\_date']['StringValue']**

**dining\_time = message['MessageAttributes']['dining\_time']['StringValue']**

**num\_people = message['MessageAttributes']['num\_people']['StringValue']**

**phone = message['MessageAttributes']['phone']['StringValue']**

**print(location, cuisine, dining\_date, dining\_time, num\_people, phone)"""**

**return {**

**'message' : message**

**}**

1. **Get data from sqs**
2. **Queue name: LF1SQSLF2.fifo**
3. **QueueUrl:** [**https://sqs.us-east-1.amazonaws.com/702727121783/LF1SQSLF2.fifo**](https://sqs.us-east-1.amazonaws.com/702727121783/LF1SQSLF2.fifo)

**Import packages**

**import json**

**import boto3**

**import logging**

**logger = logging.getLogger()**

**logger.setLevel(logging.DEBUG)**

**Lambda Handler**

**At the time you create a Lambda function, you specify a *handler*, which is a function in your code, that AWS Lambda can invoke when the service executes your code**

|  |  |
| --- | --- |
| 1. **pulls a message from the SQS queue** |  |
| 1. **Create SQS client** |  |

**def lambda\_handler(event, context):**

**sqs = boto3.client('sqs')**

1. **Get URL for SQS queue**

**response = sqs.get\_queue\_url(QueueName='LF1SQSLF2.fifo')**

**queue\_url = response['QueueUrl']**

**sendMessage = {}**

**message = {}**

1. **Receive a message from SQS queue**

**QueueUrl (*string*) :The URL of the Amazon SQS queue to which permissions are added.**

**AttributeName.N:A list of attributes that need to be returned along with**

**each message**

* **All – Returns all values.**

## **Response Elements**

**The following element is returned by the service.**

* **Message.N**
* **A list of messages.**
* **Type: Array of** [**Message**](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_Message.html) **objects**

**try:**

**response = sqs.receive\_message(**

**QueueUrl="https://sqs.us-east-1.amazonaws.com/702727121783/LF1SQSLF2.fifo",**

**AttributeNames=['All'],**

**MaxNumberOfMessages=10**

**)**

**logger.debug("sqs response :")**

1. **Getting the first message from queue**

**message = response['Messages'][0]**

**logger.debug(message)**

**body = message['Body']**

**logger.debug(body)**

**message = json.loads(body)**

**location = message['Location']['StringValue']**

**cuisine = message['Cuisine']['StringValue']**

**dining\_date = message['DiningDate']['StringValue']**

**dining\_time = message['DiningTime']['StringValue']**

**num\_people = message['NumPeople']['StringValue']**

**phone = message['PhoneNum']['StringValue']**

**logger.debug(location+' '+cuisine+' '+dining\_date+' '+dining\_time+' '+num\_people+' '+phone)**

**#sendMessage = location+' '+cuisine+' '+dining\_date+' '+dining\_time+' '+num\_people+' '+phone**

**#call elastic search**

**#after elastic search**

**name = "Filgy Resturant"**

**address = "Country 1399"**

**num\_reviews = "1000"**

**rating = 5**

**#name = response['Items'][0]['Name']**

**#address = response['Items'][0]['Address']**

**#num\_reviews = response['Items'][0]['Num\_of\_Reviews']**

**#rating = response['Items'][0]['Rating']**

**sendMessage = "Hello! For {}, we recommend the {} {} restaurant on {}. The place has {} of reviews and an average score of {} on Yelp. Enjoy!".format(location, name, cuisine, address, num\_reviews, rating)**

**logger.debug(sendMessage)**

**except Exception as ex:**

**template = "An exception of type {0} occurred. Arguments:\n{1!r}"**

**errormessage = template.format(type(ex).\_\_name\_\_, ex.args)**

**if errormessage.find("An exception of type KeyError occurred. Arguments:") == -1:**

**logger.debug("Error:- Empty sqs queue")**

**else:**

**logger.debug("Error:- "+errormessage)**

**return {**

**'statusCode': 200,**

**'body': json.dumps(sendMessage)**

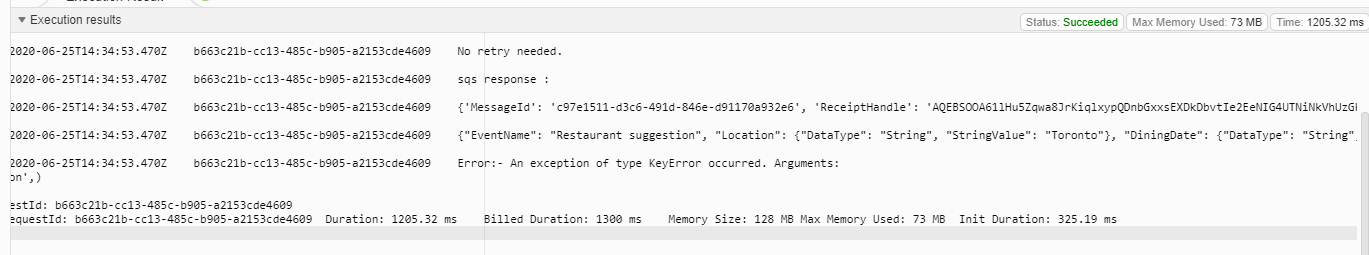
**}**

**ERROR:**

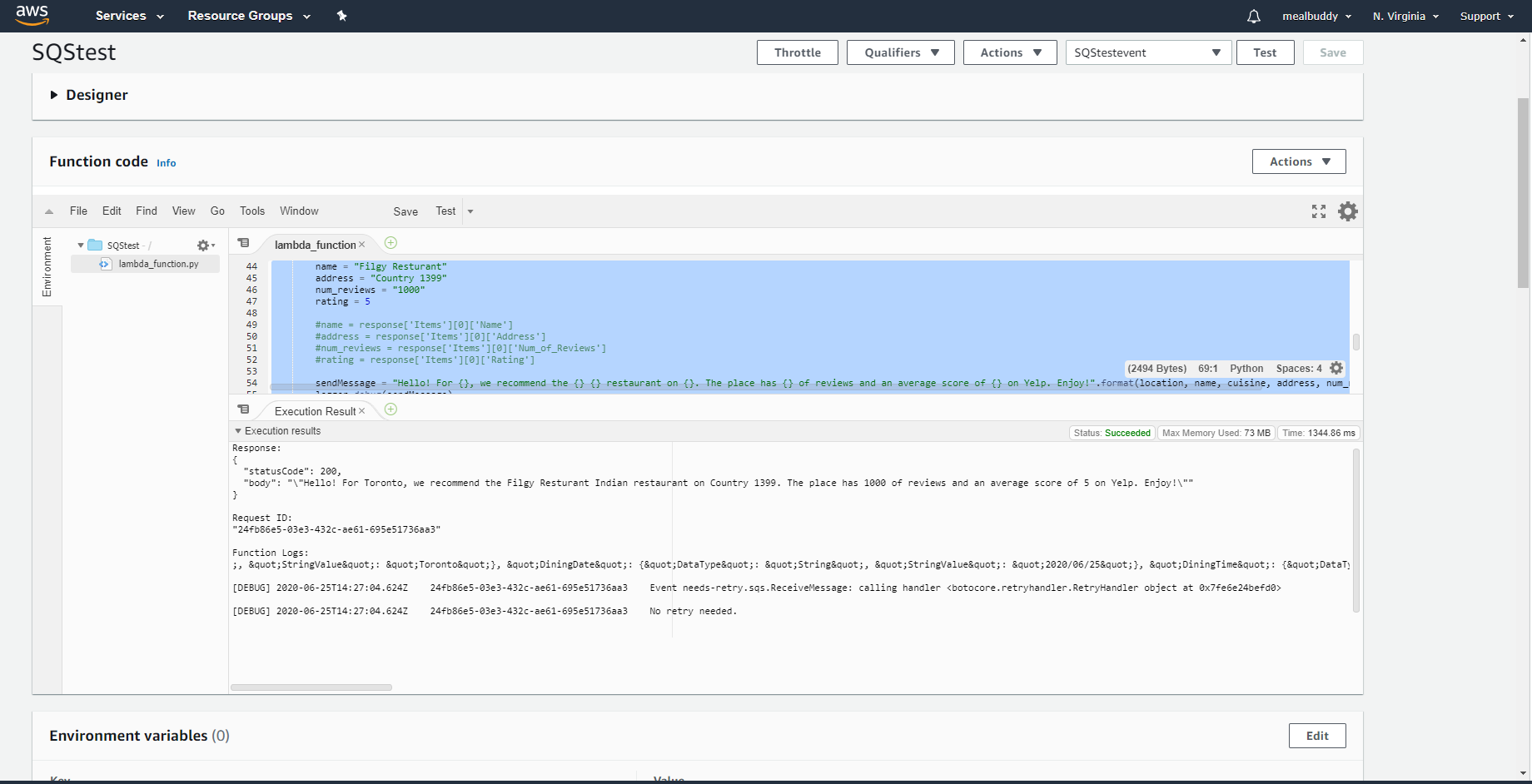
**Since response from sqs is a Nested JSON and in python it appears as a Dictionary with keys and values**

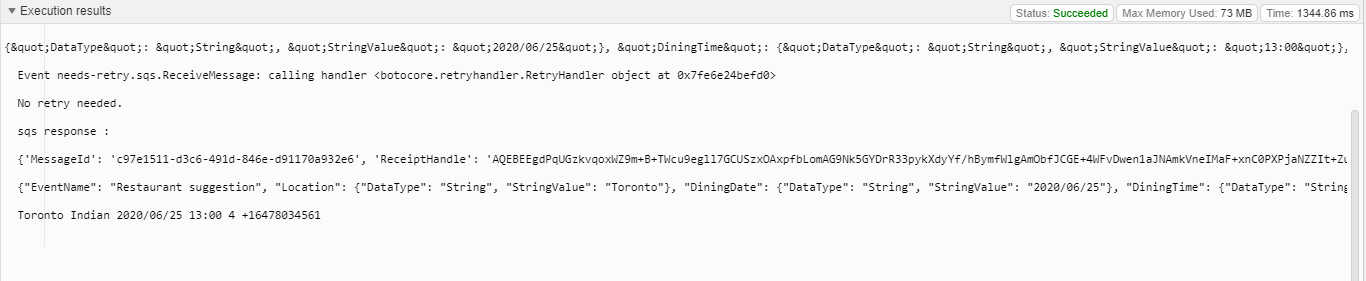
**Dictionary contains keys and values in string, but expected in json**

**Change the Key’s values into json format again**

****

**Output**

****

****