AWS SQS Project

Table of Contents

[1. Project Brief 3](#_Toc38538741)

[2. Requirements 3](#_Toc38538742)

[3. Architectural Overview 3](#_Toc38538743)

[4. Task 4](#_Toc38538744)

[4.1 Setting up DyanmoDB 4](#_Toc38538745)

[4.2 Setting up SQS 4](#_Toc38538746)

[4.3 Setting up an EC2 Instance 4](#_Toc38538747)

[4.4 Setting up IAM roles policy 5](#_Toc38538748)

[4.5 Setting up Lambda function 5](#_Toc38538749)

[5. Running the application 5](#_Toc38538750)

[6. Code 6](#_Toc38538751)

[6.1 Code 1a 6](#_Toc38538752)

[6.2 Code 1.b 7](#_Toc38538753)

[6.3 Code 2a 8](#_Toc38538754)

# Project Brief

Create a system that produces messages random and is placed in SQS ready to be consumed. The objective of this project is to utilise AWS services to simulate a situation where there are too many messages are coming through and needs to be added to a database. This can be seen as messaging IT support for a company, where there are many clients messaging IT support for a company, and this all needs to be added to a database ready to be picked up by the employee.

# Requirements

These are the list of topics that needs to be covered:

* EC2
* IAM Roles and creating policies from JSON
* IAM Users with programmatic Access to SQS
* Lambda
* SQS
* DynamoDB
* Basic Python
* Basic Linux

# Architectural Overview

A screenshot of a cell phone

Description automatically generated

# Task

Task instructions.

## Setting up DyanmoDB

Create a table in DyanmoDB. Call this table Message and the partition/primary key should be called MessageId.

## Setting up SQS

Create an SQS queue. Call this queue Messages, and leave the rest as default settings. This default setting should be the **standard** queue.

## Setting up an EC2 Instance

Create an EC2 instance with the following specification:

* Ubuntu 18.04 LTS
* T2.micro

Leave the rest as default settings. Ensure the security group allows SSH access.

Installation:

* Python3
* Pip3
* Awscli

Ensure you use aws configure to log in as the user that has SQS access as this server will be sending messages over to SQS.

Create a python file and copy and paste the code provided in Code 1.a section of this brief.

The code does not require any modifications. To run the code, you will require to pass on some arguments. For example, if the file is called send\_messages.py. You will need to run this code by:

./send\_messages.py -q Messages -i 0.1

## Setting up IAM roles policy

You will need to create a custom role as this will be attached to the Lambda function. This is needed so SQS can be a trigger for the Lambda function as well as adding items to DynamoDB.

The policy can be found in Code 2.a.

This policy provided needs to be modified. The ARN section is missing and has ‘…’ places in. You will need to provide the arn for the DynamoDB table and SQS Queue in their respective section.

## Setting up Lambda function

Attach the role created in section 4.4.

Create a lambda function and paste in the code that is provided in Code 1.b. This code does not need to be modified. You will need to create **environment** variables.

QUEUE\_NAME: This should be the name of the SQS Queue created.

MAX\_QUEUE\_MESSAGES: This should be set to 10

DYNAMODB\_TABLE: This should be the name of the DyanamoDB table created.

Add a **trigger** to the Lambda function. This trigger is going to be an SQS trigger, select the name of the queue you’ve created and leave the rest as default.

# Running the application

Once everything has been set up, go back into the EC2 instance that has the code sending messages to SQS, and run the code.

What you should be expecting is to see on your SQS queue, some messages appearing and the lambda function should automatically start consuming those messages very quickly. The code for the lambda function is taking those messages and pushing them onto the DynamoDB table you’ve created.

# Code

## Code 1a

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

import argparse

import logging

import sys

from time import sleep

import boto3

from faker import Faker

parser = argparse.ArgumentParser()

parser.add\_argument("--queue-name", "-q", required=True,

help="SQS queue name")

parser.add\_argument("--interval", "-i", required=True,

help="timer interval", type=float)

parser.add\_argument("--message", "-m", help="message to send")

parser.add\_argument("--log", "-l", default="INFO",

help="logging level")

args = parser.parse\_args()

if args.log:

logging.basicConfig(

format='[%(levelname)s] %(message)s', level=args.log)

else:

parser.print\_help(sys.stderr)

sqs = boto3.client('sqs')

response = sqs.get\_queue\_url(QueueName=args.queue\_name)

queue\_url = response['QueueUrl']

logging.info(queue\_url)

while True:

message = args.message

if not args.message:

fake = Faker()

message = fake.text()

logging.info('Sending message: ' + message)

response = sqs.send\_message(

QueueUrl=queue\_url, MessageBody=message)

logging.info('MessageId: ' + response['MessageId'])

sleep(args.interval)

## Code 1b

import json

import os

from datetime import datetime

import boto3

QUEUE\_NAME = os.environ['QUEUE\_NAME']

MAX\_QUEUE\_MESSAGES = os.environ['MAX\_QUEUE\_MESSAGES']

DYNAMODB\_TABLE = os.environ['DYNAMODB\_TABLE']

sqs = boto3.resource('sqs')

dynamodb = boto3.resource('dynamodb')

def lambda\_handler(event, context):

# Receive messages from SQS queue

queue = sqs.get\_queue\_by\_name(QueueName=QUEUE\_NAME)

print("ApproximateNumberOfMessages:",

queue.attributes.get('ApproximateNumberOfMessages'))

for message in queue.receive\_messages(

MaxNumberOfMessages=int(MAX\_QUEUE\_MESSAGES)):

print(message)

# Write message to DynamoDB

table = dynamodb.Table(DYNAMODB\_TABLE)

response = table.put\_item(

Item={

'MessageId': message.message\_id,

'Body': message.body,

'Timestamp': datetime.now().isoformat()

}

)

print("Wrote message to DynamoDB:", json.dumps(response))

# Delete SQS message

message.delete()

print("Deleted message:", message.message\_id)

## Code 2a

{

"Action": [

"dynamodb:PutItem"

],

"Effect": "Allow",

"Resource": "…"

},

{

"Action": [

"sqs:Describe\*",

"sqs:Get\*",

"sqs:List\*",

"sqs:DeleteMessage",

"sqs:ReceiveMessage"

],

"Effect": "Allow",

"Resource": "…"

}