



***ASSIGNMENT 5***

*In  
The Class of*

***CSCI5710: SERVERLESS DATA PROCESSING***

*by*

*Janvi Patel [B00863421]*

***Submitted to***  
*Prof. Saurabh Dey*  
*Department of Computer Science*  
*Dalhousie university.*

*Date: 30<sup>th</sup> July 2021*

## Part A. Explore & Build a Use Case:

### AWS Kinesis

Amazon Kinesis is a service offered by Amazon to make it easy to process and analyze the real time streaming data to get the timely insights and react quickly to new information. This provides flexibility to choose the tools that provides streaming data at any scale with cost effective. Moreover, Amazon Kinesis provides a respond to process and analyze data to respond and processing can begin.

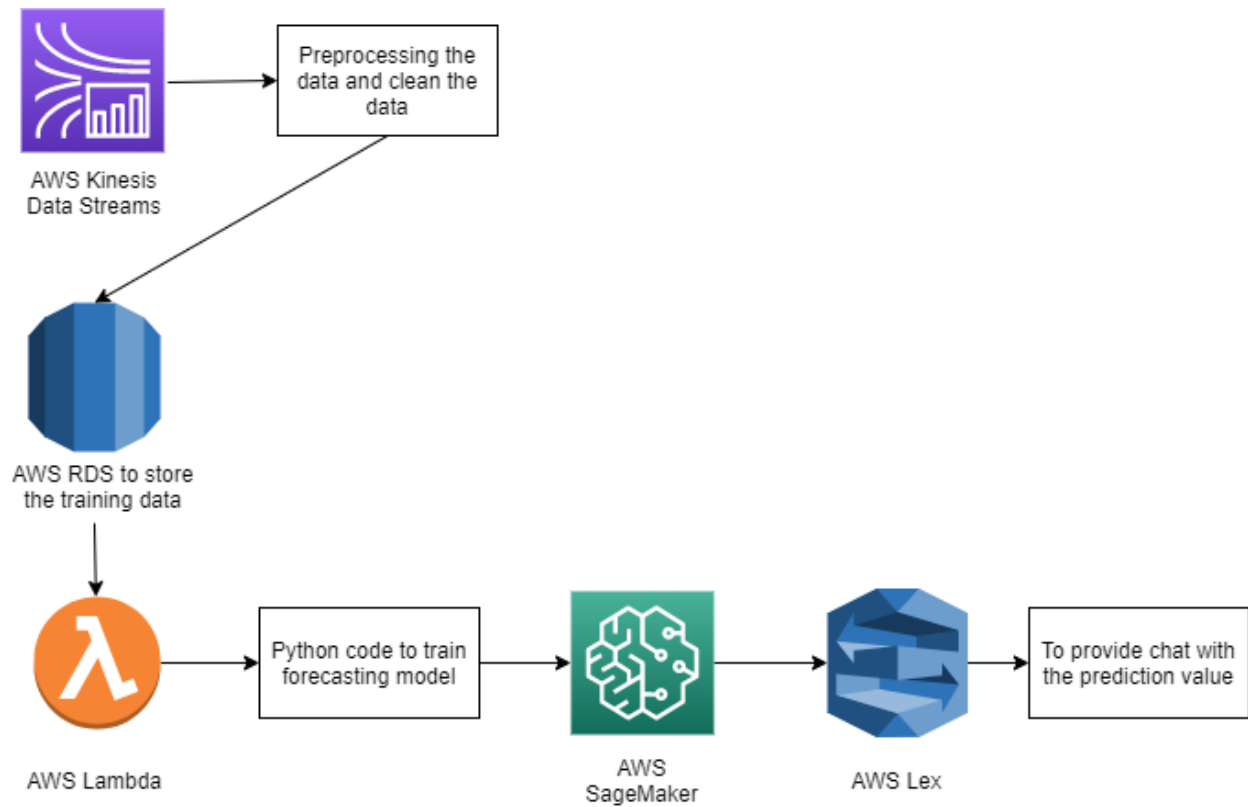
There are 3 basic components described below:

1. Kinesis data/video stream receives data from multiple sources (click streams, IoT devices, video recording devices, etc.) and divides it in shards (or partitions). It allows multiple applications to consume the same data.
2. Kinesis data firehose is service loads the streaming data (from Kinesis streams) into various data lakes, stores, AWS S3, MongoDB, etc. It is a fully managed service which scales according to the data.
3. Kinesis data analytics is service allows for advanced analytics of the data. It allows users to respond to data in real-time.

Scenario – A machine that works on heavy equipment in which depends on massively on the machines. That has some sensors that continuously provides data for example pressure, temperature, humidity, light sensor. They now want to predict that based on the sensor the machine when the machine is going to fail. Further, the company has already provided that past data with all sensor values and what was the exact sensor value when the machine has failed. This means that bots should be able to respond to chat messages.

Use-Case – The company can leverage the processing power and infrastructure provided by different AWS Services. Having huge machinery and maintaining the machine is it's self a big task. The sensor data is being fetched real time and that is streaming data fetched using AWS Kinesis streams. Further, AWS Lambda when coupled with Kinesis the streaming data will be fetched and then store them in the AWS RDS database. Training of the model will be done on AWS Sage Maker and use Machine learning forecasting. These models could be continuously updated by training and testing them on live data provided by Kinesis streams. AWS Lex could be integrated with the system to provide interaction of predicting the sensor values for machine that can be stopped in the neared future. This will make the employees life easy to find alert that this machine is going to fail.

## Block Diagram –



## Part B. AWS SNS and AWS SQS:

- 1) Create a queue in AWS SQS in figure 1.

Amazon SQS > Queues > Create queue

### Create queue

**Details**

Type  
Choose the queue type for your application or cloud infrastructure.

**You can't change the queue type after you create a queue.**

☒ **Standard** [Info](#)

At-least-once delivery, message ordering isn't preserved

- At-least once delivery
- Best-effort ordering

☐ **FIFO** [Info](#)

First-in-first-out delivery, message ordering is preserved

- First-in-first-out delivery
- Exactly-once processing

Name

assignment5B00863421

A queue name is case-sensitive and can have up to 80 characters. You can use alphanumeric characters, hyphens (-), and underscores (\_).

Figure 1 create queue

aws Services Search for services, features, marketplace products, and docs [Alt+S] vocstartsoft/user1357694=jn410076@dal.ca @ 4631-1347-4338 N. Virginia Support

**Queue assignment5B00863421 created successfully**  
You can now send and receive messages.

Amazon SQS > Queues > assignment5B00863421

assignment5B00863421 Edit Delete Purge Send and receive messages

**Details** [Info](#)

Name	Type	ARN
assignment5B00863421	Standard	arn:aws:sqs:us-east-1:463113474338:assignment5B00863421
Encryption	URL	Dead-letter queue
-	https://sqs.us-east-1.amazonaws.com/463113474338/assignment5B00863421	-

► More

Figure 2 Successfully created SQS

The code below provides details of AWS SQS code:

```
# -*- coding: utf-8 -*-

import boto3
import time
import random

client = boto3.resource('sqs', region_name='us-east-1')

queue = client.get_queue_by_name(QueueName='assignment5B00863421')

OrderMessage = ['Please confirm order for a fabulous collection of white and
cream flowers make this the perfect gift',
                '5 Mamma Mia order is placed',
                'Can you provide Orchid bouquet at 1991 brunswick street?',
                'I have received wrong order for Tulip bouquet',
                'Do you have white peaceful lily bouquet today?']

Size = ['LARGE', 'MEDIUM', 'SMALL', 'JUNIOR', 'MINI', 'TOSS', 'WAND']

def sendToSQS():
    while True:
        bouquetOrderMessage = (random.choice(OrderMessage))
        bouquetSize = random.choice(Size)
        sendData = bouquetOrderMessage + " " + bouquetSize
        response = queue.send_message(MessageBody= sendData)
        print(response)
        time.sleep(300)

sendToSQS()
```

Python Code for AWS SQS

Message received in SQS mentioned in figure 3.

Receive messages [Info](#)

Edit poll settings

Stop polling

Poll for messages

Messages available

7

Polling duration

30

Maximum message count

10

Polling progress

1.8 receives/second

Messages (7)

View details

Delete

Q Search messages

< 1 > ⚙

<input type="checkbox"/>	ID	Sent	Size	Receive count
<input type="checkbox"/>	3bc41255-c861-480a-897e-3dad1e9ddfb0	7/29/2021, 23:28:01 ADT	51 bytes	1
<input type="checkbox"/>	7d527cb5-2f0e-4dc9-8b06-c20b105898d0	7/29/2021, 23:27:44 ADT	51 bytes	1
<input type="checkbox"/>	56a178c7-f5d9-4da7-a685-37f3ea7010bc	7/29/2021, 23:27:36 ADT	51 bytes	1
<input type="checkbox"/>	30a2a087-b85a-4ef4-ab7f-c0e5a938fca4	7/29/2021, 23:25:47 ADT	16 bytes	1
<input type="checkbox"/>	9403f59e-f5f6-42f1-aeb2-e351df813b87	7/29/2021, 23:21:47 ADT	14 bytes	1
<input type="checkbox"/>	5cb39610-41b5-4e1f-9134-1c75e5214179	7/29/2021, 23:17:47 ADT	17 bytes	1
<input type="checkbox"/>	da1f59ef-49e3-4549-bc0e-cb9118a516f0	7/29/2021, 23:13:46 ADT	18 bytes	1

Figure 3: Message received on polling

Message: 7d527cb5-2f0e-4dc9-8b06-c20b105898d0

Details

Body

Attributes

Do you have white peaceful lily bouquet today? TOSS

Done

Figure 4: Body message of one of the page

AWS SNS with the Standard Type of the topic mentioned in figure 5:

Amazon SNS > Topics > Create topic

## Create topic

### Details

**Type** [Info](#)  
Topic type cannot be modified after topic is created

☐ FIFO (first-in, first-out)

- Strictly-preserved message ordering
- Exactly-once message delivery
- High throughput, up to 300 publishes/second
- Subscription protocols: SQS

☒ Standard

- Best-effort message ordering
- At-least once message delivery
- Highest throughput in publishes/second
- Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints

**Name**

Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (\_).

**Display name - optional**  
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message. [Info](#)

Maximum 100 characters, including hyphens (-) and underscores (\_).

Figure 5: create topic

✕ 🟢 Topic assignment5PartB created successfully.  
You can create subscriptions and send messages to them from this topic. Publish message ✕ ⓘ

Amazon SNS > Topics > assignment5PartB

## assignment5PartB

Edit Delete Publish message

### Details

Name	assignment5PartB	Display name	-
ARN	arn:aws:sns:us-east-1:463113474338:assignment5PartB	Topic owner	463113474338
Type	Standard		

Subscriptions Access policy Delivery retry policy (HTTP/S) Delivery status logging Encryption Tags

**Subscriptions (0)**

Edit Delete Request confirmation Confirm subscription Create subscription

Figure 6: Successfully generated SNS

## Create subscription

**Details**

Topic ARN

Protocol

The type of endpoint to subscribe

Email

Endpoint

An email address that can receive notifications from Amazon SNS.

After your subscription is created, you must confirm it. [Info](#)

Figure 7: Provided Email as Endpoint

assignment5PartB
Edit
Delete
Publish message

**Details**

Name  
assignment5PartB

Display name  
-

ARN  
arn:aws:sns:us-east-1:463113474338:assignment5PartB

Topic owner  
463113474338

Type  
Standard

Subscriptions
Access policy
Delivery retry policy (HTTP/S)
Delivery status logging
Encryption
Tags

**Subscriptions (1)**
Edit
Delete
Request confirmation
Confirm subscription
Create subscription

	ID	Endpoint	Status	Protocol
<input type="radio"/>	Pending confirmation	janvipatel4199@gmail.com	<input type="checkbox"/> Pending confirmation	EMAIL

Figure 8: Details of ARN



assignment5PartB

EditDeletePublish message

Details

Name

assignment5PartB

ARN

arn:aws:sns:us-east-1:463113474338:assignment5PartB

Type

Standard

Display name

-

Topic owner

463113474338

Subscriptions

Access policy

Delivery retry policy (HTTP/S)

Delivery status logging

Encryption

Tags

Subscriptions (1)

EditDeleteRequest confirmationConfirm subscriptionCreate subscription

Q Search

< 1 > ⚙

ID	Endpoint	Status	Protocol
4768cf6d-e408-4fc8-b699-d02d3134b0b0	janvipatel4199@gmail.com	Confirmed	EMAIL

Figure 9: ARN details of SNS

IAM role:

Create role

1234

Review

Provide the required information below and review this role before you create it.

Role name\*

SNSrole

Use alphanumeric and '+,=, @, \_' characters. Maximum 64 characters.

Role description

Allows Lambda functions to call AWS services on your behalf.

Maximum 1000 characters. Use alphanumeric and '+,=, @, \_' characters.

Trusted entities

AWS service: lambda.amazonaws.com

Policies

AWSLambdaBasicExecutionRole

AmazonSNSFullAccess

Permissions boundary

Permissions boundary is not set

No tags were added.

\* Required

CancelPreviousCreate role

Figure 10: Provide Lambda Access

## Lambda Creation:

**Basic information**

**Function name**  
Enter a name that describes the purpose of your function.  
  
Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime** [Info](#)  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

**Permissions** [Info](#)  
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

**▼ Change default execution role**  
**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).  
☐ Create a new role with basic Lambda permissions  
☒ Use an existing role  
☐ Create a new role from AWS policy templates  
**Existing role**  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.  
  
[View the SNSrole role](#) on the IAM console.

Figure 11: Lambda creation

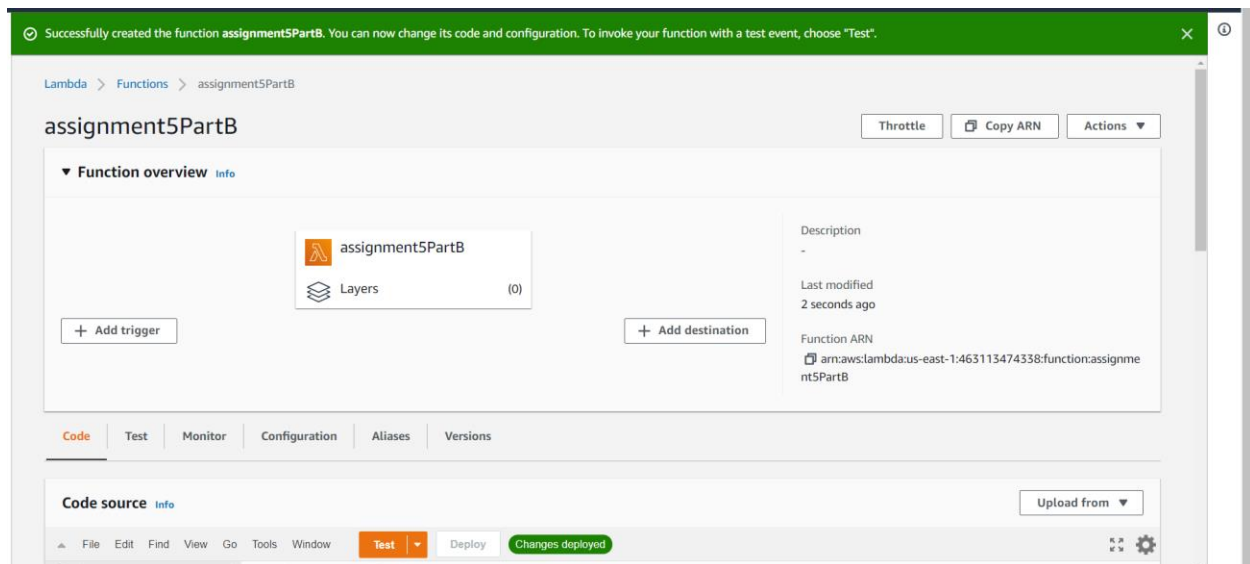



Figure 12: Lambda successfully generation

### Trigger configuration

 **EventBridge (CloudWatch Events)**  
aws events management-tools

**Rule**  
Pick an existing rule, or create a new one.

☒ Create a new rule

☐ Existing rules

**Rule name\***  
Enter a name to uniquely identify your rule.

partbRole

**Rule description**  
Provide an optional description for your rule.

**Rule type**  
Trigger your target based on an event pattern, or based on an automated schedule.

☐ Event pattern

☒ Schedule expression

**Schedule expression\***  
Self-trigger your target on an automated schedule using Cron or rate expressions. Cron expressions are in UTC.

rate(5 min)

e.g. rate(1 day), cron(0 17 ? \* MON-FRI \*)

Figure 13: CloudWatch Events

Lambda > Functions

Functions (3)

Last fetched in 0 seconds

Actions

Create function

Filter by tags and attributes or search by keyword

< 1 >

	Function name	Description	Package type	Runtime	Code size	Last modified
<input type="radio"/>	assignment5PartB		Zip	Python 3.8	299.0 byte	4 minutes ago
<input type="radio"/>	wordCloudGroup11		Zip	Python 3.8	517.0 byte	3 days ago
<input type="radio"/>	wordCloudGetDataGroup11		Zip	Python 3.8	1.2 kB	2 days ago

Figure 14: Lambda successfully generated

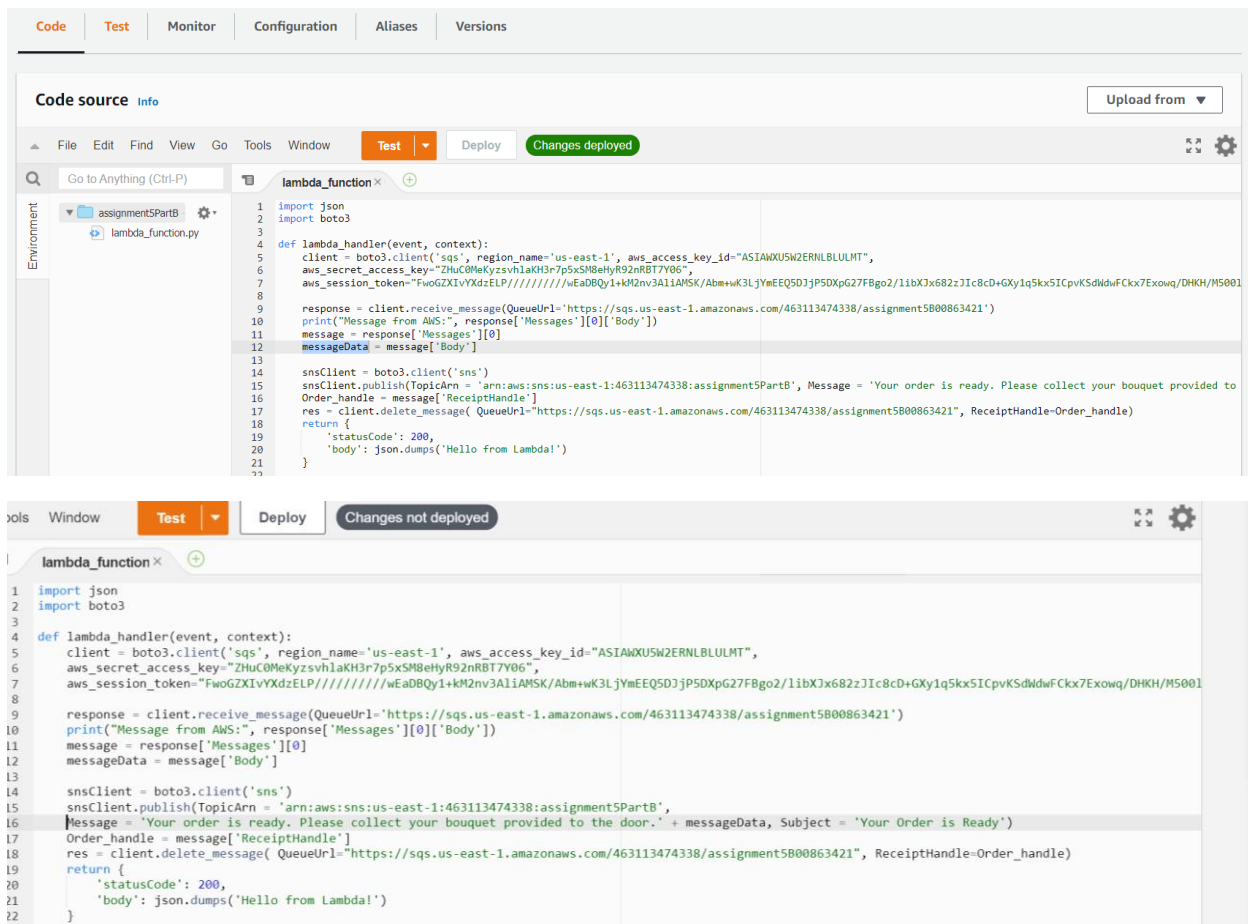


Figure 15: Lambda code

Logs that Lambda code is successfully running:

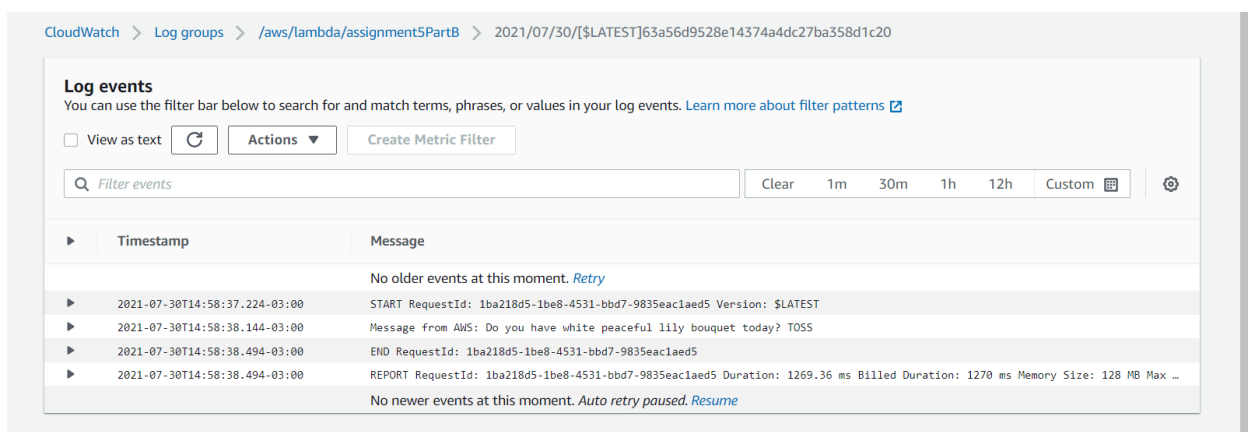
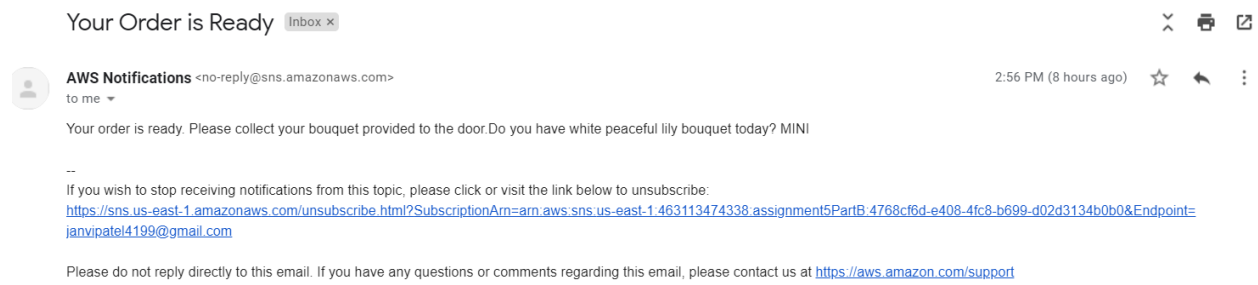


Figure 16: Cloud Watch Logs

## SNS Mail received:



*Figure 17: Mail received from SNS*