LOW LEVEL DOCUMENTATION -3

PROJECT-3

1. Create Bucket in S3
   1. Create Bucket with Name : “groupno6”
      1. ACL Enabled
      2. Create folder :- data
      3. Create folder :- logs
      4. Create folder :- scripts
      5. Create folder :- results
      6. Create folder:- archive
      7. Create folder:-failed-records
2. Create Queue :- group\_6\_project\_3\_queue
3. Enable Dead-letter queue group
4. Create Lambda Function:- group\_6\_project\_3
   * 1. Select Runtime Python 3.10
     2. Change default execution role : i.Create new Role

ii.Use existing Role

* + 1. Add Trigger SQS

1. Create new Role

Create an Admin full access IAM role “Glue\_AdminFullAccessRole”

* 1. Go to IAM 🡪 Roles 🡪 Create role
  2. Search policy 🡪 AdministratorAccess 🡪 Next:Tags 🡪 Next:Review
  3. Role Name : Glue\_AdminFullAccessRole 🡪 Create Role

ii. Use existing Role we use InfoceptsLambdaRole

Select which role you want to use

1. Create Dynamo DB table:- group\_6\_transaction\_table

**Sending the Message to Queue we use Spyder to run python code** :

* + **random**: This library provides functions for generating random numbers, selecting random elements from a sequence, shuffling sequences, and more.
  + **string**: This module contains various string constants and functions, such as generating random strings or manipulating strings.
  + **json**: This module provides functions for working with JSON data, including parsing JSON strings into Python objects and serializing Python objects into JSON strings.
  + **datetime, date, timedelta**: These modules provide classes and functions for working with dates and times in Python, including creating and manipulating dates, calculating time differences, and formatting dates and times.
  + **randrange**: This function is part of the random module and allows you to generate random numbers within a specified range.
  + **boto3**: This library is the AWS SDK for Python and provides interfaces to interact with various AWS services, such as Amazon S3, Amazon EC2, Amazon DynamoDB, and more.
  + **uuid**: This module provides functions for generating universally unique identifiers (UUIDs). UUIDs are unique identifiers that can be used to identify objects in a distributed computing environment.
  + **sqs\_url:** This variable stores the URL of an Amazon Simple Queue Service (SQS) queue.
  + It specifies the destination queue to which the message will be sent.
  + **msg:** This variable is assigned the result of a function call to get\_transaction\_message().
  + It presumably retrieves a transaction message or data that will be sent to the SQS queue.
  + **msgstr:** This variable stores the JSON string representation of the msg object.
  + It uses the json.dumps() function to convert the Python object to a JSON string.
  + **send\_message\_to\_q():** This function is not defined in the code snippet you provided,
  + The **get\_transaction\_message()** function generates a dictionary representing a transaction message.
  + It includes random values for transaction ID, transaction number, transaction date, transaction amount, GST, and excise duty.
  + The function returns the generated message dictionary.
  + The **tn\_rf\_id()** function generates a unique five-digit ID using UUID and returns it as a string.
  + The **Txn\_no()** function generates a random ten-digit transaction number by sampling digits from the string of digits (string.digits)
  + and then concatenates them together to form the transaction number. The function returns the generated transaction number as a string.
  + The **tn\_dt()** function generates a random datetime between two specified datetime objects.
  + It calculates the time difference in seconds between the two datetime objects, generates a random number of seconds within that range,
  + and adds it to the starting datetime. The function returns the generated datetime formatted as a string in the format 'YYYY-MM-DD HH:MM AM/PM'.
  + The **send\_message\_to\_q()** function sends a message to an SQS queue using the provided message and queue\_url parameters.
    - It initializes an SQS client with the provided AWS credentials and region,
  + and then calls the send\_message() method of the SQS client to send the message to the specified queue.
  + The function includes additional parameters such as DelaySeconds and MessageAttributes to control the message delivery and metadata.
  + The **lambda\_handler()** function is called with empty strings as the event and context parameters, triggering the execution of the code within it.
  + **After putting the message in queue, Lambda Function will trigger the Message from SQS And store it in Dynamo DB table**
  + In AWS, if a message sent by a producer reaches a consumer successfully,
    - it will be removed from the queue. However,
  + if there are any issues or inconsistencies during the message delivery process,
  + AWS provides a feature called Dead Letter Queues (DLQs) to handle such scenarios.
  + The code begins by importing the necessary libraries and clients from the Boto3 AWS SDK for Python. It imports the DynamoDB client as dyn and the SQS client as sqs.
  + The table\_name variable is set to the name of the DynamoDB table (group\_6\_transaction\_table) where the data will be stored.
  + The **generate\_unique\_id** function generates a unique ID using the uuid library. It converts the UUID to an integer and returns the first 5 digits.
  + The **id\_int** variable is set by calling the generate\_unique\_id function, generating a unique ID.
  + The **generate\_txn\_id** function is used to generate a transaction ID for each record. It retrieves the previous transaction ID from the environment variable LAST\_TXN\_ID (which is initially set to id\_int). It increments the previous ID, stores it in the environment variable, and returns the current ID.
  + The **add\_in\_ddb** function takes a record and a source system ID as input. It constructs the necessary DynamoDB item for each accounting code (A101, A104, A103) and adds it to the items list. Then, it uses the DynamoDB client to put each item into the specified DynamoDB table.
  + The **lambda\_handler** function is the entry point for the Lambda function. It takes the event and context parameters.
  + Inside the **lambda\_handler**, the function prints the received message from the SQS queue and the name of the queue it came from.
  + The received message is parsed as JSON and stored in the record variable.
  + The code checks for any inconsistencies in the record. If any key in the record is None, an empty string, or if the "amt" value is None or less than 0, it sends the record to a Dead Letter Queue (DLQ) by sending it as a message to the SQS queue specified by the DLQ URL.
  + If the record passes the consistency checks, it is considered a valid message from the queue. The add\_in\_ddb function is called to add the record to the DynamoDB table with the source system ID of 3.
  + Finally, the Lambda function returns a response with a status code of 200 and a simple JSON body as a confirmation.
  + The overall purpose of this Lambda function is to handle incoming messages from an SQS queue, process the messages, and store the data in a DynamoDB table. It performs some validations and sends inconsistent records to a DLQ for further analysis.

**Spyder console code**

The code provided is an AWS Lambda function written in Python. Here is a brief explanation of the code:

1. The necessary libraries are imported, including `random`, `string`, `json`, `datetime`, `date`, `timedelta`, and `boto3`.

2. The `lambda\_handler()` function is defined, which serves as the entry point for the Lambda function. It is triggered by an event and context.

3. Inside the `lambda\_handler()` function, an SQS URL is specified. A transaction message is generated using the `get\_transaction\_message()` function. The message is then converted to a JSON string using `json.dumps()`. Finally, the message is sent to the specified SQS queue using the `send\_message\_to\_q()` function.

4. The `get\_transaction\_message()` function generates a transaction message with random values for transaction ID, transaction number, transaction date, amount, GST, and excise duty.

5. The `tn\_rf\_id()` function generates a random string of length 14 using a combination of letters and digits.

6. The `Txn\_no()` function generates a random string of length 10 using digits.

7. The `tn\_dt()` function generates a random datetime between January 1, 2000, and December 31, 2009.

8. The `send\_message\_to\_q()` function creates an SQS client using the provided credentials and region. It sends a message to the specified SQS queue, including message attributes such as title and format.

9. The `lambda\_handler('','')` line at the end invokes the `lambda\_handler()` function with empty event and context parameters. This line is commented out in the code snippet provided.

In summary, the code generates a transaction message with random values and sends it to an SQS queue using the AWS Lambda function. The function can be triggered by various events, such as an API Gateway request, a scheduled event, or an S3 event.

**Lambda Function**

The provided code is an AWS Lambda function written in Python. Here is a brief explanation of the code:

1. The necessary libraries are imported, including `json`, `os`, and `boto3` (for DynamoDB and SQS).

2. The DynamoDB and SQS clients are created using the `boto3.client()` function.

3. The `table\_name` variable is set to the name of the DynamoDB table where the records will be inserted.

4. The `generate\_txn\_id()` function retrieves the last transaction ID from the environment variable `LAST\_TXN\_ID`, increments it by 1, and updates the environment variable with the new value. It returns the current transaction ID.

5. The `add\_in\_ddb()` function takes a `record` and `source\_system\_id` as input. It creates items to be inserted into DynamoDB for each account type (A101, A104, A103). Each item contains the necessary attributes based on the values in the `record` and `source\_system\_id`. The items are added to the DynamoDB table using the `dyn.put\_item()` function.

6. The `lambda\_handler()` function is the entry point for the Lambda function. It receives the `event` and `context` parameters.

7. Inside the `lambda\_handler()` function, the received message and source information are printed for debugging purposes.

8. The message received from the SQS queue is parsed as JSON and stored in the `record` variable.

9. The code checks for any inconsistencies or missing values in the `record`. If any inconsistency is found, the record is sent to a Dead Letter Queue (DLQ) using the `sqs.send\_message()` function. A DLQ is a queue that stores messages that couldn't be processed successfully.

10. If no inconsistencies are found, the code proceeds to insert the record into DynamoDB using the `add\_in\_ddb()` function. The `source\_system\_id` is set to 3.

11. The function returns a response with a 200 status code and a simple JSON message.

In summary, this Lambda function is triggered by messages in an SQS queue. It processes the messages by inserting the data into a DynamoDB table. If any inconsistencies are found, the messages are sent to a Dead Letter Queue for further analysis.

**OUTLOOK OF THE STRUCTURE**

