

Assignment: AWS Lambda with SQS Trigger

Submitted by:

Name: Vikram

Problem Statement

You work for XYZ Corporation. Your corporation wants to launch a new web-based application and they do not want their servers to be running all the time. It should also be managed by AWS. Implement suitable solutions.

Tasks To Be Performed:

1. Create a sample Python Lambda function.
 2. Set the Lambda Trigger as SQS and send a message to test invocations
-

Solution Explanation

To meet the requirement of a serverless and fully managed setup, we used AWS Lambda (a serverless compute service) and Amazon SQS (Simple Queue Service) as a trigger.

Whenever a message is sent to the SQS queue, Lambda automatically executes and processes that message.

This ensures servers don't need to run continuously, reducing cost and management overhead.

Tasks Performed

Step 1: Create IAM Role for Lambda

Go to IAM → Roles → Create role.

Choose AWS Service → Lambda.

Attach these policies:

AWSLambdaBasicExecutionRole

AmazonSQSFullAccess

Role name: LambdaSQSTriggerRole.

Step 2
Add permissions

Step 3
Name, review, and create

Trusted entity type

- AWS service

Allow AWS services like EC2, Lambda, or others to perform actions in this account.
- AWS account

Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- Web identity

Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.
- SAML 2.0 federation

Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.
- Custom trust policy

Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case

Lambda

Choose a use case for the specified service.

Use case

Lambda
Allows Lambda functions to call AWS services on your behalf.

LambdaSQSTriggerRole

Role LambdaSQSTriggerRole created.

Summary

Creation date
October 23, 2025, 16:04 (UTC+05:30)

Last activity
-

ARN
arn:aws:iam::062250062838:role/LambdaSQSTriggerRole

Maximum session duration
1 hour

Permissions **Trust relationships** **Tags** **Last Accessed** **Revoke sessions**

Permissions policies (2) info
You can attach up to 10 managed policies.

Filter by Type

Policy name	Type	Attached entities
<input type="checkbox"/> AmazonSQSFullAccess	AWS managed	1
<input type="checkbox"/> AWSLambdaBasicExecutionRole-94ca93a2...	Customer managed	2

Step 2: Create Lambda Function

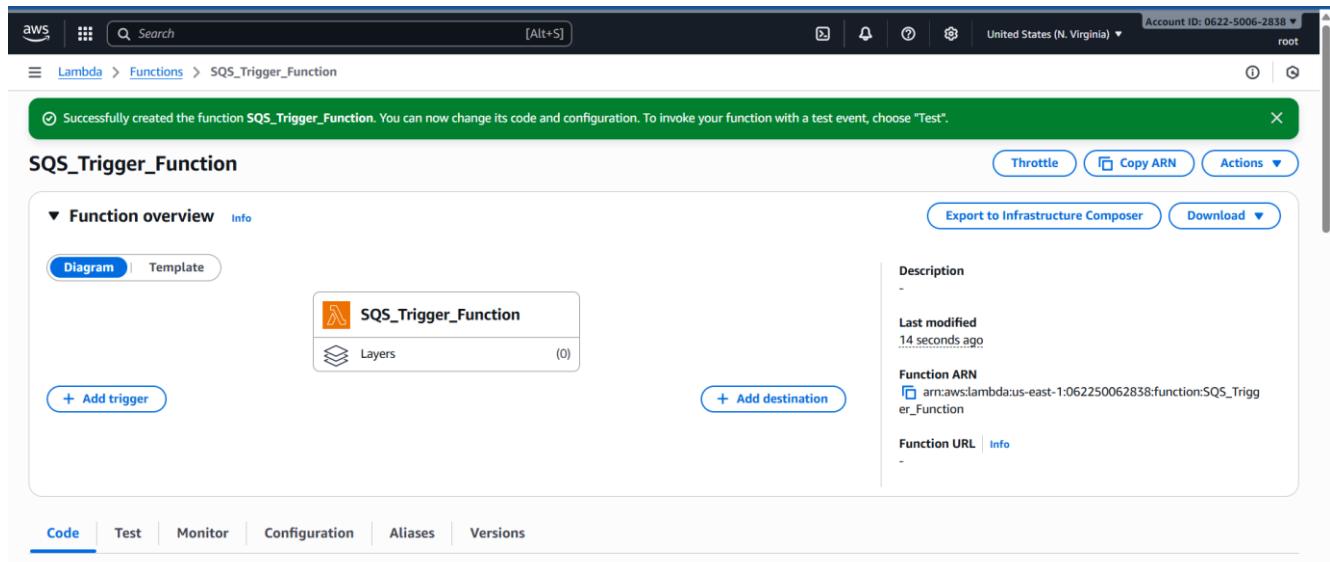
Go to AWS Lambda → Create Function → Author from scratch.

Function Name: SQS_Trigger_Function

Runtime: Python 3.13

Execution Role: Use existing role → LambdaSQSTriggerRole

Click Create Function.



Step 3: Add Python Code

Replace the default handler code with the following and click **Deploy**:

```
import json
```

```
def lambda_handler(event, context):
    for record in event['Records']:
        print("Message Body:", record['body'])
    return {
        'statusCode': 200,
        'body': json.dumps('Message processed successfully!')
    }
```

- This code prints every message received from SQS into the CloudWatch logs.

The screenshot shows the AWS Lambda console interface. At the top, a green banner displays the message "Successfully updated the function SQS_Trigger_Function." Below the banner, a navigation bar includes tabs for Code, Test, Monitor, Configuration, Aliases, and Versions. The Code tab is selected. On the left, the EXPLORER sidebar shows a project named "SQS_TRIGGER_FUNCTION" with a "DEPLOY" section containing "Deploy (Ctrl+Shift+U)" and "Test (Ctrl+Shift+I)" buttons. The main area is titled "Code source" and contains a code editor for "lambda_function.py". The code is as follows:

```
import json

def lambda_handler(event, context):
    for record in event['Records']:
        print("Message Body:", record['body'])
    return {
        'statusCode': 200,
        'body': json.dumps('Message processed successfully!')
    }
```

At the top right of the code editor, there are buttons for "Open in Visual Studio Code" and "Upload from".

Step 4: Test Lambda Manually

After deploying the code, click **Test** on the Lambda console.

It will ask you to **create a new test event**.

Create the event by **replacing the existing JSON code** with:

```
{
  "Records": [
    {
      "body": "Manual test message"
    }
  ]
}
```

Click **Test**.

This will run the Lambda function once and **automatically create the CloudWatch log group** /aws/lambda/SQS_Trigger_Function.

Note: Change the Cloudwatch Log Format to JSON under configuration settings in lambda function

The screenshot shows the AWS Lambda console interface. On the left, there's a sidebar with 'EXPLORER' and 'DEPLOY' sections, and buttons for 'Deploy' and 'Test'. The main area has tabs for 'lambda_function.py' and 'Create new test event'. The code editor contains Python code for a lambda function. To the right, there's a 'TEST EVENTS [SELECTED: DEMO]' section with a 'Create new test event' button and a 'Private saved events' dropdown. Below the code editor, tabs for 'PROBLEMS', 'OUTPUT', 'CODE REFERENCE LOG', and 'TERMINAL' are visible, along with status messages like 'Status: Succeeded' and 'Test Event Name: demo'. A 'Response:' field is also present.

Step 5: Create an SQS Queue

Go to Amazon SQS → Create Queue → Standard queue.

Queue Name: LambdaTriggerQueue.

Keep all default configurations and create the queue.

The screenshot shows the AWS SQS 'Queues' page. A green success message at the top says 'Queue LambdaTriggerQueue created successfully. You can now send and receive messages.' Below it, the queue details are shown: Name (LambdaTriggerQueue), Type (Standard), ARN (arn:aws:sqs:us-east-1:062250062838:LambdaTriggerQueue), URL (https://sqs.us-east-1.amazonaws.com/062250062838/LambdaTriggerQueue), and Dead-letter queue (-). There are buttons for 'Edit', 'Delete', 'Purge', 'Send and receive messages', and 'Start DLQ redrive'. At the bottom, tabs for 'Queue policies', 'Monitoring', 'SNS subscriptions', 'Lambda triggers', 'EventBridge Pipes', 'Dead-letter queue', 'Tagging', 'Encryption', and 'Dead-letter queue redri' are visible. An 'Access policy' section with an 'Edit' button is also present.

Step 6: Configure SQS as Lambda Trigger

Open the Lambda function → Configuration → Triggers → Add trigger.

Choose SQS, select LambdaTriggerQueue, click Add.

Now Lambda executes automatically whenever a message arrives in SQS.

The screenshot shows two pages from the AWS Lambda console:

- Top Page:** Shows the configuration for the "SQS_Trigger_Function". The "Configuration" tab is selected. Under the "Triggers" section, it says "No triggers. No triggers are configured." with a blue "Add trigger" button.
- Bottom Page:** A modal dialog titled "Add trigger" for configuring a new trigger. It has a "Trigger configuration" section with a dropdown menu showing "SQS" selected. Below it is a "SQS queue" input field containing "arn:aws:sqs:us-east-1:062250062838:LambdaTriggerQueue".
 - Event poller configuration:** Contains options for "Activate trigger" (checked), "Enable metrics" (unchecked), and a "Batch size - optional" input field set to "10".

The screenshot shows the AWS Lambda Function Configuration page. On the left, a sidebar lists various configuration options: General configuration, Triggers (selected), Permissions, Destinations, Function URL, Environment variables, Tags, VPC, RDS databases, and Monitoring and operations tools. The main panel is titled "Triggers (1) Info". It contains a search bar with placeholder text "Find triggers", a "Trigger" button, and a single item listed: "SQS: LambdaTriggerQueue" with state "Creating". There are "Details" and "Edit" buttons next to it. A navigation bar at the top right includes "Fix errors", "Edit", "Delete", and "Add trigger" buttons.

Step 7: Test the Setup with SQS

Go to Amazon SQS → LambdaTriggerQueue → Send and receive messages.

Message Body:

```
{"name": "Vikram", "message": "Hello from SQS!"}
```

The screenshot shows the "Send and receive messages" page for the LambdaTriggerQueue. The title is "Send and receive messages". Below it is a sub-header "Use this page to send, retrieve and view messages, helping you experiment with various queue features." The main form is titled "Send message". It has a "Message body" field containing the JSON message {"name": "Vikram", "message": "Hello from SQS!"}. There are optional fields: "Message group ID - optional" (with a note about character limits and punctuation), "Delivery delay" (set to 0 seconds), and "Message attributes - optional". At the bottom are "Clear content" and "Send message" buttons.

Click Send Message.

Check CloudWatch Logs → /aws/lambda/SQS_Trigger_Function → latest log stream → you should see:

Message Body: {"name": "Vikram", "message": "Hello from SQS!"}

The screenshot shows the CloudWatch Log groups interface. On the left, there's a sidebar with navigation links like 'CloudWatch' (selected), 'Log groups', 'Favorites and recents', 'Dashboards', 'Operations (New)', 'Overview', 'Investigations', 'Configuration', and 'Alarms'. The main area is titled 'Log groups (1)' and displays a single log group entry: '/aws/lambda/SQS_Trigger_Function'. The entry includes a checkbox, a 'Log group' button, a 'Standard' retention period ('Never expire'), and a 'Configure' button. There are also buttons for 'Actions', 'View in Logs Insights', 'Start tailing', and 'Create log group'.

The screenshot shows the CloudWatch Logs Insights interface. On the left, there's a sidebar with navigation links like 'Logs Insights' (selected), 'Logs (2)', 'Patterns (1)', and 'Visualization'. The main area is titled 'Logs (2)' and shows a histogram with 2 records matched. Below the histogram, the log entries are listed:

#	@timestamp	@message
► 1	2025-10-23T16:42:49.934+00:00	Message Body: Message Body: {"name": "Vikram", "message": "Hello from SQS!"}
► 2	2025-10-23T16:40:00.839+00:00	Message Body: Message Body: {"name": "Vikram", "message": "Hello from SQS!"}

✓ Result

Successfully implemented a **serverless solution** using **AWS Lambda** and **SQS**.

Whenever a message is sent to the queue, the Lambda function executes automatically and logs the message body in **CloudWatch** — fulfilling the requirement of a fully managed, non-continuous compute setup.