🛠️ Complete Guide: AWS Lambda + SNS + SQS Integration

Introduction

In this guide, we will create a simple event-driven architecture using **AWS Lambda**, **SNS (Simple Notification Service)**, and **SQS (Simple Queue Service)**.

We will:

- Trigger a Lambda function to publish a message to an SNS Topic.
- The SNS Topic will fan-out this message to multiple SQS Queues.
- Each SQS Queue will invoke another Lambda function to process the message.

✅ Prerequisites

- AWS Account
- AWS CLI installed and configured (`aws configure`)
- Node.js installed for optional CLI testing

🔢 Architecture Overview

```
[ PublishToSNS Lambda ]
          |
          v
[   SNS Topic (UserMessageTopic)   ]
        /        \
       v          v
[ SQS Queue 1 ]  [ SQS Queue 2 ]
     |                |
     v                v
[ Lambda 1 ]    [ Lambda 2 ]
```

🔄 Step-by-Step Setup

Step 1: Create an SNS Topic

- Go to AWS Console > SNS > Topics > Create Topic.
- Select **Type**: *Standard*.
- Name: `UserMessageTopic`.
- Click **Create Topic**.

Step 2: Create Two SQS Queues

- Go to AWS Console > SQS > Create Queue.
- Create two **Standard Queues**:
  - `UserMessageQueue1`
  - `UserMessageQueue2`
- Leave other settings default.

Step 3: Subscribe SQS Queues to SNS Topic

- Go to SNS > Topics > `UserMessageTopic` > Create subscription.
- Protocol: **Amazon SQS**.
- Endpoint: select **UserMessageQueue1 ARN**.
- Repeat for **UserMessageQueue2**.
- **Enable Raw Message Delivery** (Important!).

Step 4: Create Lambda Function to Publish Message

- AWS Console > Lambda > Create Function.
- Name: `PublishToSNSFunction`.
- Runtime: Python 3.13.
- Create a new role with permissions:
  - `AWSLambdaBasicExecutionRole`
  - `AmazonSNSFullAccess`.

Environment Variable:

- `SNS_TOPIC_ARN = <Your SNS Topic ARN>`

Sample Python 3.13 Code:

```
import boto3
```

```
import os

def lambda_handler(event, context):
    sns_client = boto3.client('sns')
    topic_arn = os.environ['SNS_TOPIC_ARN']
    message = "Hello, this is a user message!"

    response = sns_client.publish(
        TopicArn=topic_arn,
        Message=message,
        Subject='New User Message'
    )

    return {
        'statusCode': 200,
        'body': f'Message sent! ID: {response["MessageId"]}'
    }
```

Step 5: Create Processor Lambda Functions

Create two separate Lambdas:

- `ProcessQueue1Message`
- `ProcessQueue2Message`

Each should have permissions:

- `AWSLambdaBasicExecutionRole`
- `AmazonSQSFullAccess`

Sample Processor Code:

```
def lambda_handler(event, context):
    for record in event['Records']:
        print(f"Received message: {record['body']}")
```

Step 6: Configure SQS as Trigger for Processor Lambdas

- Go to each Processor Lambda.
- Add Trigger > Choose **SQS** > Select the correct queue.
- Save.

Lambda will automatically get permissions to poll the SQS queue.

Step 7: Test the Entire Flow

1. Invoke Publish Lambda

- Go to Lambda > `PublishToSNSFunction` > Test.
- Create a simple event and run the function.

2. Verify SQS Processing

- Check CloudWatch logs for:
  - `ProcessQueue1Message`
  - `ProcessQueue2Message`
- Look for log lines like:

```
Received message: Hello, this is a user message!
```

📊 Useful AWS CLI Commands (Optional)

Invoke Lambda:

```
aws lambda invoke --function-name PublishToSNSFunction output.json
```

List SQS Messages:

```
aws sqs receive-message --queue-url <queue-url>
```

View Logs:

Use AWS Console CloudWatch or AWS SDK to pull logs.