CSCI 5410 Serverless Data Processing

Assignment 3 Report

Prepared by:
Vikram Venkatapathi - B00936916

Master of Applied Computer Science (Summer'23)
Faculty of Computer Science
Dalhousie University

GitLab Repo link : **https://git.cs.dal.ca/vikramv/csci5410-summer-23-b00936916/-/tree/A3**

Contents

Part I.

PART A

Explore & Build a Use Case

1. Introduction

As a former cashier at the Atlantic Superstore[2], I marveled at the complexities of inventory management. This inspired me to envision a scenario where AWS Kinesis[1] revolutionizes the process, ensuring precise stock numbers and minimizing wastage. In this hypothetical scenario, real-time data streams from IoT devices and point-of-sale systems are leveraged to optimize inventory, prompt reordering, and deliver an exceptional shopping experience for customers.

2. Hypothetical Scenario and the Use Case

Hypothetical Scenario: Smart Inventory Management at Atlantic Superstore

In this hypothetical scenario, the Atlantic Superstore[2], a large retail chain, faces challenges with manual inventory management, leading to inefficiencies and occasional stockouts. The store envisions implementing an intelligent inventory management system using AWS Kinesis[1] and other AWS services to address these issues.

Usecase:

The Atlantic Superstore decides to leverage AWS Kinesis[1] for real-time data streaming to capture continuous updates from IoT devices and point-of-sale systems placed strategically across the store. These devices will transmit product movement, stock levels, and sales data in real-time to AWS Kinesis Data Streams[15]. With AWS Kinesis Data Analytics[16], the store processes and analyzes the streaming data on the fly. Advanced analytics and machine learning algorithms are applied to forecast demand, identify trending products, and detect potential stock shortages. The system can instantly update inventory records and provide a comprehensive view of stock levels. To ensure efficient stock management, the Atlantic Superstore integrates AWS Lambda functions[17] into the system. These functions automatically trigger reordering processes for low-stock items, streamlining inventory replenishment and preventing stockouts. Additionally, the store incorporates AWS Comprehend[18] for sentiment analysis to understand customer preferences and feedback. By integrating customer sentiment with sales data, the store can optimize its inventory based on real-time demand and adapt to changing customer needs. The smart inventory management system using AWS Kinesis allows the Atlantic Superstore to optimize inventory levels, reduce wastage, and deliver a seamless shopping experience for its customers. With real-time insights and automated processes, the store can maintain accurate stock numbers, minimize manual efforts, and ensure products are readily available for customers, ultimately driving increased customer satisfaction and operational efficiency.

2.1. AWS Architecture diagram

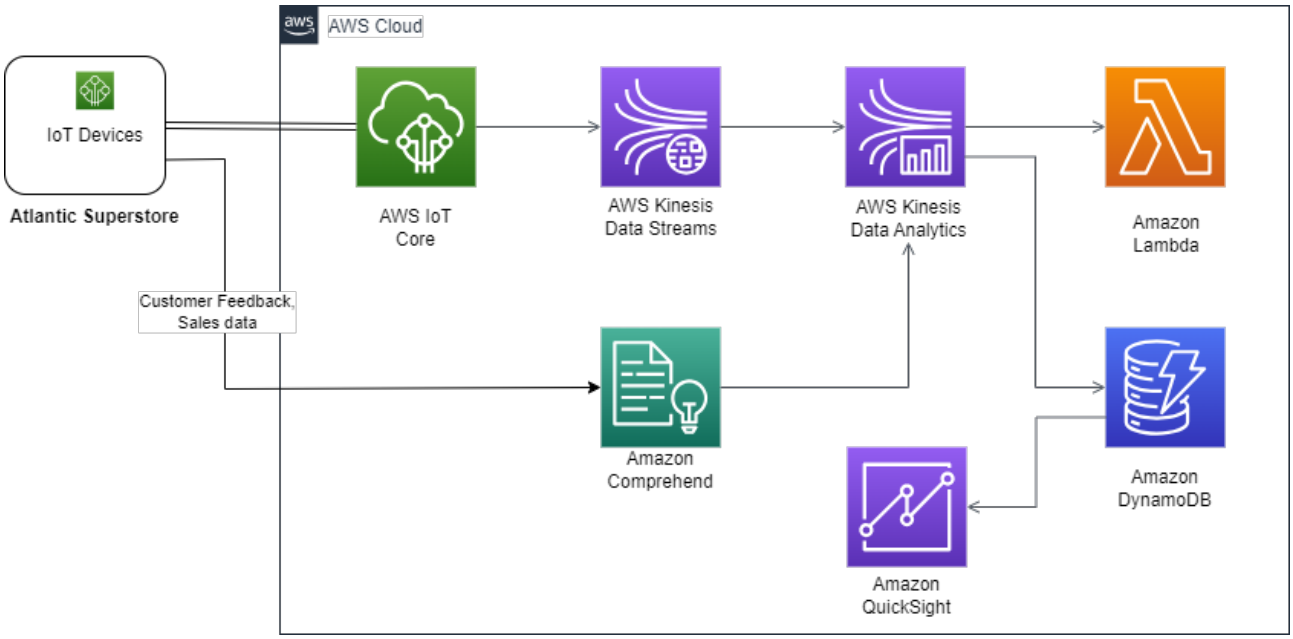

Figure 2.1.: ***Smart Inventory Management Architecture at Atlantic Superstore using AWS Services (Created using: draw.io [23])***

1. **IoT Devices**: These devices are strategically placed on shelves and equipped with sensors to detect product movement and measure stock levels.
2. **AWS IoT Core**: The IoT devices securely connect and transmit data to AWS Kinesis Data Streams, ensuring smooth communication with the cloud.[14]
3. **AWS Kinesis Data Streams**: This service captures and streams real-time data from the IoT devices, including product IDs, timestamps, stock levels, and sales data.[15]
4. **AWS Kinesis Data Analytics**: The streaming data is processed and analyzed in real-time to forecast demand, identify trending products, and detect potential stock shortages. It also triggers Amazon Lambda functions for automatic reordering and anomaly detection.[16]
5. **Amazon Lambda**: Lambda functions automatically trigger reordering processes for low-stock items and flag any irregularities in stock movements.[17]
6. **Amazon Comprehend**: This service is used for sentiment analysis, integrating customer feedback with sales data to understand customer preferences and optimize inventory accordingly.[18]
7. **Amazon DynamoDB**: The analyzed data is stored in DynamoDB, a NoSQL database, offering fast and scalable access to historical inventory data.[19]

8. **Amazon QuickSight**: The data in DynamoDB is visualized and analyzed through QuickSight, enabling the store to create interactive dashboards and gain real-time insights into stock levels, popular products, and inventory performance.[20]

Part II.

PART B

Build an event-driven serverless application using AWS Lambda

3. NOTE

3.1. Usage of AWS account

On the account of Novelty, I attempted to create all the services required for this assignment, using the Serverless Framework[3]. Running this YAML file creates an IAM role for logging purposes. Since I don't have permission to create an IAM role in the AWS academy account, I have used my personal AWS account to deploy the services for this entire assignment.

4. Procedure followed for the given experiment

4.1. AWS Architecture diagram

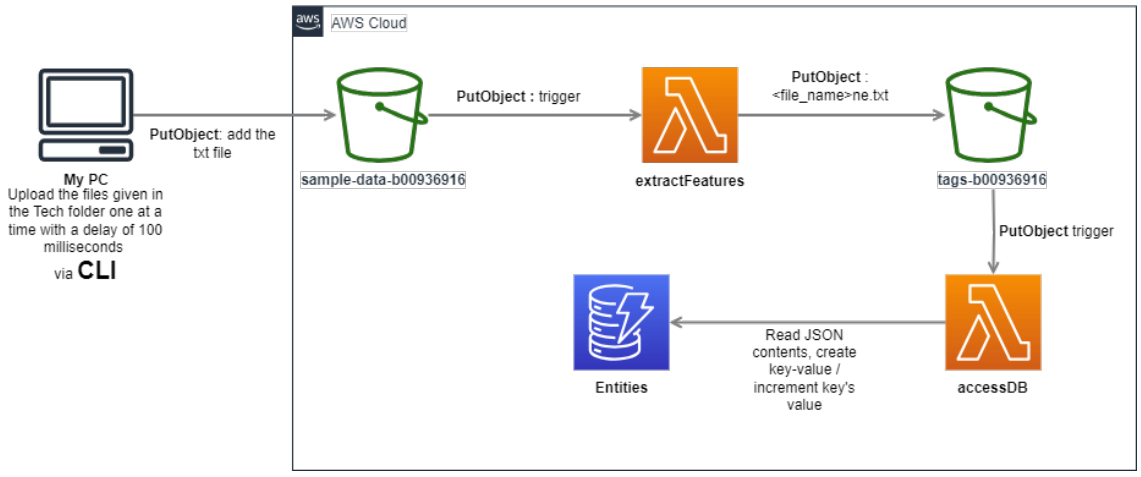

Figure 4.1.: ***AWS Architecture diagram for the given work flow (Created using: draw.io [23])***

4.2. Steps

1. Configure the AWS credentials in the CLI using the command **"aws configure"**.

2. Create an IAM role with the following permission provided by AWS, Role_Name = **lambdaRole**: [21]
   a) AWSLambdaExecute: Full access to CloudWatch Logs; Read, Write access to S3
3. Create an IAM role with the following permissions provided by AWS, Role_Name = **lambdaRole_accessDB**: [21]
   a) AWSLambdaExecute: Full access to CloudWatch Logs; Read, Write access to S3
   b) AmazonDynamoDBFullAccess (as the name suggests)
4. Role = lambdaRole is for lambda extractFeatures
5. Role = lambdaRole_accessDB is for lambda accessDB
6. Run the following command to create a serverless-framework project

```
# Create a new AWS Python3 Serverless project at path 'event-driven-
    serverless-app-a3'.
serverless create --template aws-python3 --path event-driven-serverless-
    app-a3
```

7. Create the required resources using the following serverless framework script[3] (serverless.yml)[4]

```
service: event-driven-serverless-app-a3

provider:
  name: aws
  runtime: python3.8
  stage: dev
  region: us-east-1

resources:
  Resources:
    # 1st S3 Bucket
    SampleDataBucket:
      Type: AWS::S3::Bucket
      Properties:
        BucketName: sample-data-b00936916 # Actual Bucket Name :
            SampleDataB00936916 -> Bucket name should not contain
            uppercase characters

    # 2nd S3 Bucket
    TagsBucket:
      Type: AWS::S3::Bucket
      Properties:
        BucketName: tags-b00936916 # Actual Bucket Name : TagsB00936916
            -> Bucket name should not contain uppercase characters
```

```
# Lambda Function to Extract Named Entities
ExtractFeaturesLambda:
  Type: AWS::Lambda::Function
  Properties:
    FunctionName: extractFeatures
    Code:
      ZipFile: | #sample code
        def lambda_handler(event, context):
        # TODO implement
          return {
            'statusCode': 200,
            'body': json.dumps('Hello␣from␣Lambda!')
          }
    Handler: app.lambda_handler
    Runtime: python3.8   # Specify the runtime for the Lambda
       function
    MemorySize: 512
    Timeout: 10
    Role: arn:aws:iam::000966082997:role/lambdaRole #role with
       permissions : AWSLambdaExecute, AmazonDynamoDBFullAccess

# Lambda Function to Access DynamoDB and Update
AccessDBLambda:
  Type: AWS::Lambda::Function
  Properties:
    FunctionName: accessDB
    Code:
      ZipFile: | #sample code
        def lambda_handler(event, context):
        # TODO implement
          return {
            'statusCode': 200,
            'body': json.dumps('Hello␣from␣Lambda!')
          }
    Handler: app.lambda_handler
    Runtime: python3.8   # Specify the runtime for the Lambda
       function
    MemorySize: 512
    Timeout: 10
    Role: arn:aws:iam::000966082997:role/lambdaRole_accessDB #role
       with permissions : AWSLambdaExecute, AmazonDynamoDBFullAccess

# DynamoDB Table
DynamoDBTable:
  Type: AWS::DynamoDB::Table
  Properties:
    TableName: Entities
```

```
        AttributeDefinitions:
          - AttributeName: entity_name
            AttributeType: S # Set the AttributeType to String for the
                partition key "entity_name"
        KeySchema:
          - AttributeName: entity_name # Use file_name as the partition
              key
            KeyType: HASH  # Set the KeyType to HASH for the partition
                key "entity_name"
        ProvisionedThroughput:
          ReadCapacityUnits: 1
          WriteCapacityUnits: 1
```

8. The above script does the following
   a) Create 2 - S3 buckets: sample-data-b00936916, tags-b00936916
   b) Create 2 - lambdas: extractFeatures(with IAM role: lambdaRole), accessDB(with IAM role: lambdaRole_accessDB)
   c) Create 1 - DynamoDB table: Entities (with Partition key: entity_name)
9. Run the command **serverless deploy** to deploy the .yml file to create the resources in CloudFormation
10. Run the following commands to **add the InvokeFunction permission** to both the lambdas **via CLI** [5]

```
Lambda_1:

Command = aws lambda add-permission --function-name arn:aws:lambda:us-
    east-1:000966082997:function:extractFeatures --action lambda:
    InvokeFunction --principal 000966082997 --source-arn arn:aws:s3:::
    sample-data-b00936916 --statement-id s3_trigger

Lambda_2:

Command = aws lambda add-permission --function-name arn:aws:lambda:us-
    east-1:000966082997:function:accessDB --action lambda:InvokeFunction
    --principal 000966082997 --source-arn arn:aws:s3:::tags-b00936916 --
    statement-id s3_trigger
```

11. Now, create separate JSON files with the below contents, to specify the LambdaFunctionConfigurations, to add S3 event notification.[7]
    a) **sample-data-b00936916** bucket (trigger_configuration_sample-data-b00936916.json)

```
{
  "LambdaFunctionConfigurations": [
    {
      "LambdaFunctionArn": "arn:aws:lambda:us-east-1:000966082997:
         function:extractFeatures",
      "Events": ["s3:ObjectCreated:*"]

  ]
}
```

b) **tags-b00936916** bucket (trigger_configuration_tags-b00936916.json)

```
{
    "LambdaFunctionConfigurations": [
      {
        "LambdaFunctionArn": "arn:aws:lambda:us-east-1:000966082997:
           function:accessDB",
        "Events": ["s3:ObjectCreated:*"]

    ]
}
```

12. Now, run the following commands to add the S3 event notification to each bucket[11], **specifying the path** to the above created json file using **file://</path/to/file_name>.json** [6]

```
Bucket1:

Command = aws s3api put-bucket-notification-configuration --bucket
    sample-data-b00936916 --notification-configuration file://
    trigger_configuration_sample-data-b00936916.json


Bucket2:

Command = aws s3api put-bucket-notification-configuration --bucket tags-
    b00936916 --notification-configuration file://
    trigger_configuration_tags-b00936916.json
```

13. Download the tech.zip file, extract it, change directory to folder 'tech' [22], open the terminal, run the following script[12]

```
foreach ($file in Get-ChildItem -File) {
    aws s3 cp "$file" s3://sample-data-b00936916/
    Start-Sleep -Milliseconds 100
}
```

14. Now, all the 401 files will be uploaded with a 100 milliseconds delay.

15. Upon each upload, the flow of events mentioned in the architecture will be triggered, and all the names entities will be added to the 'Entities' dynamoDB databases, with the **key as the 'entity_name'** and **value as its frequency**.

4.3. Lamda code

4.3.1. extractFeatures

```
import json
import re
import boto3

def lambda_handler(event, context):
    # Retrieve the S3 event information
    records = event['Records']
    for record in records:
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']

        """
            Boto3 - Amazon Simple Storage Service (S3) API Documentation
            Reference: `Boto3 - Amazon S3 API <https://boto3.amazonaws.com/
               v1/documentation/api/latest/reference/services/s3.html>`_
        """

        # Read the contents of the file from S3
        s3_client = boto3.client('s3')
        response = s3_client.get_object(Bucket=bucket, Key=key)
        file_content = response['Body'].read().decode('utf-8')

        """
            Python Standard Library - re (Regular Expression) module
            Reference: `Python re (Regular Expression) module <https://docs.
               python.org/3/library/re.html>`_
        """

        # Extract words starting with a capital letter
        capital_words = re.findall(r'\b[A-Z][a-zA-Z]*\b', file_content)

        # Extract all caps words
        all_caps_words = re.findall(r'\b[A-Z]+\b', file_content)

        # Create a dictionary to store the named entities and their counts
        named_entities = {}

        # Count the occurrences of each word
        for word in capital_words + all_caps_words:
            named_entities[word] = named_entities.get(word, 0) + 1

        # Remove the file extension from the key
        file_name = key.split('.')[0]
```

```
        # Create the JSON object with the key as the file name and the named
            entities dictionary as the value
        result = {f"{file_name}ne": named_entities}

        # Convert the result to a JSON string
        named_entities_json = json.dumps(result)

        destination_s3_bucket = "tags-b00936916"

        # Upload the JSON to the tags-b00936916 bucket
        s3_client.put_object(Bucket=destination_s3_bucket, Key=f"{file_name}
            ne.txt", Body=named_entities_json)


    return {
        'statusCode': 200,
        'body': json.dumps('Named entities extracted successfully.')
    }


"""
All references:
    Python Standard Library - re (Regular Expression) module
    Reference: `Python re (Regular Expression) module <https://docs.python.
        org/3/library/re.html>`_

    Boto3 - Amazon Simple Storage Service (S3) API Documentation
    Reference: `Boto3 - Amazon S3 API <https://boto3.amazonaws.com/v1/
        documentation/api/latest/reference/services/s3.html>`_

"""
```

4.3.2. accessDB

```
import json
import boto3

def lambda_handler(event, context):
    # Retrieve the S3 event information
    records = event['Records']
    for record in records:
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']

        # Check if the file is in the format "001ne.txt"
```

```
        if key.endswith("ne.txt"):
            """
                Boto3 - Amazon Simple Storage Service (S3) API Documentation
                Reference: `Boto3 - Amazon S3 API <https://boto3.amazonaws.
                    com/v1/documentation/api/latest/reference/services/s3.
                    html>`_
            """
            # Read the contents of the file from S3
            s3_client = boto3.client('s3')

            response = s3_client.get_object(Bucket=bucket, Key=key)
            file_content = response['Body'].read().decode('utf-8')

            # Convert the JSON content to a dictionary
            named_entities = json.loads(file_content)

            # Get the file name from the key (e.g., "001ne.txt" -> "001ne")
            file_name = key.split('.')[0]

            """
                Boto3 - Amazon DynamoDB API Documentation
                Reference: `Boto3 - Amazon DynamoDB API <https://boto3.
                    amazonaws.com/v1/documentation/api/latest/reference/
                    services/dynamodb.html>`_
            """
            # Update the DynamoDB table with the named entities
            dynamodb = boto3.resource('dynamodb')
            table = dynamodb.Table('Entities')

            for entity, count in named_entities[file_name].items():
                # Convert the count to an integer before updating the table
                count = int(count)
                # Update the table with the named entity and its count
                table.update_item(
                    Key={'entity_name': entity},
                    UpdateExpression='SET frequency = if_not_exists(
                        frequency, :zero) + :val',
                    ExpressionAttributeValues={':zero': 0, ':val': count}
                )

    return {
        'statusCode': 200,
        'body': json.dumps('DynamoDB table updated successfully.')
    }


"""
```

```
All references:
    Boto3 - Amazon Simple Storage Service (S3) API Documentation
    Reference: `Boto3 - Amazon S3 API <https://boto3.amazonaws.com/v1/
        documentation/api/latest/reference/services/s3.html>`_

    Boto3 - Amazon DynamoDB API Documentation
    Reference: `Boto3 - Amazon DynamoDB API <https://boto3.amazonaws.com/v1/
        documentation/api/latest/reference/services/dynamodb.html>`_
"""
```

4.3.3. Screenshots

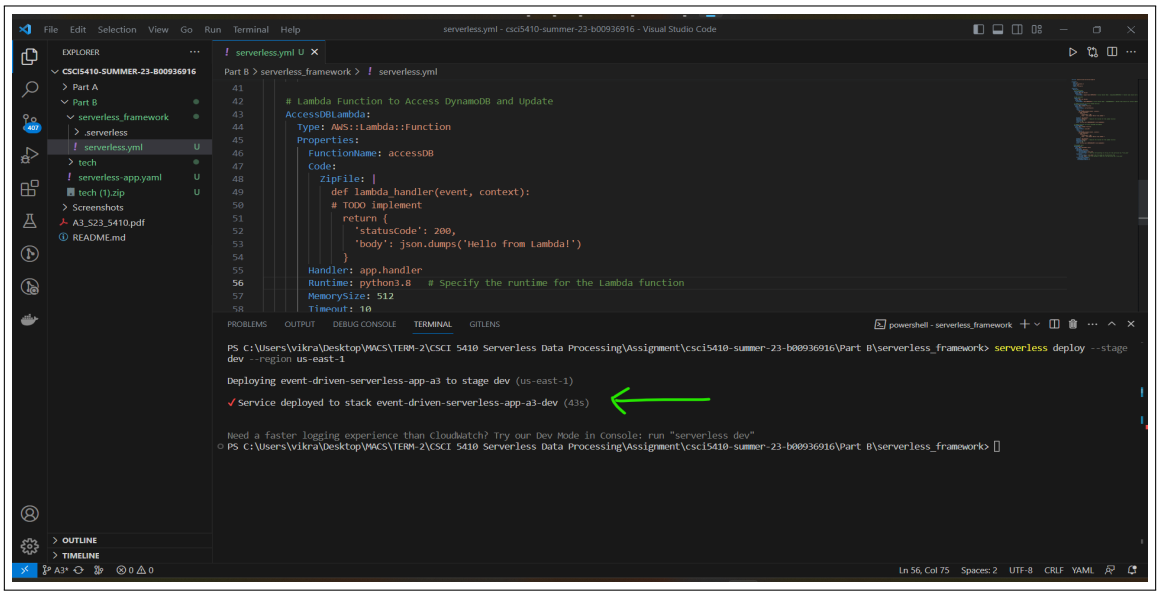

Figure 4.2.: ***Running Serverless-Framework script in CLI***

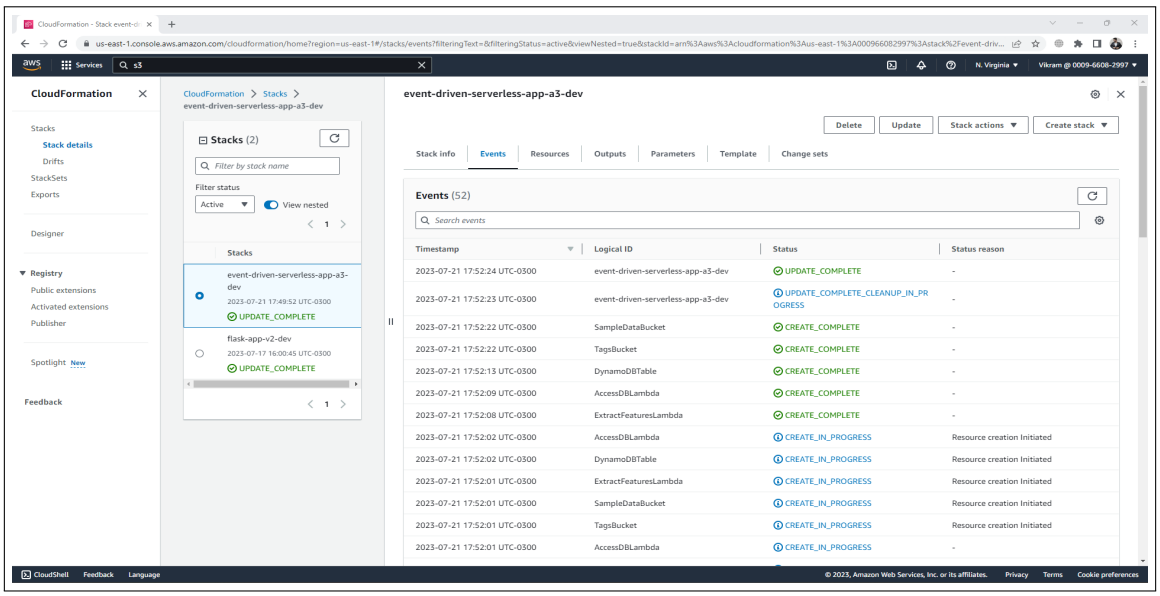

Figure 4.3.: ***Resources stack created - CloudFormation in Console***

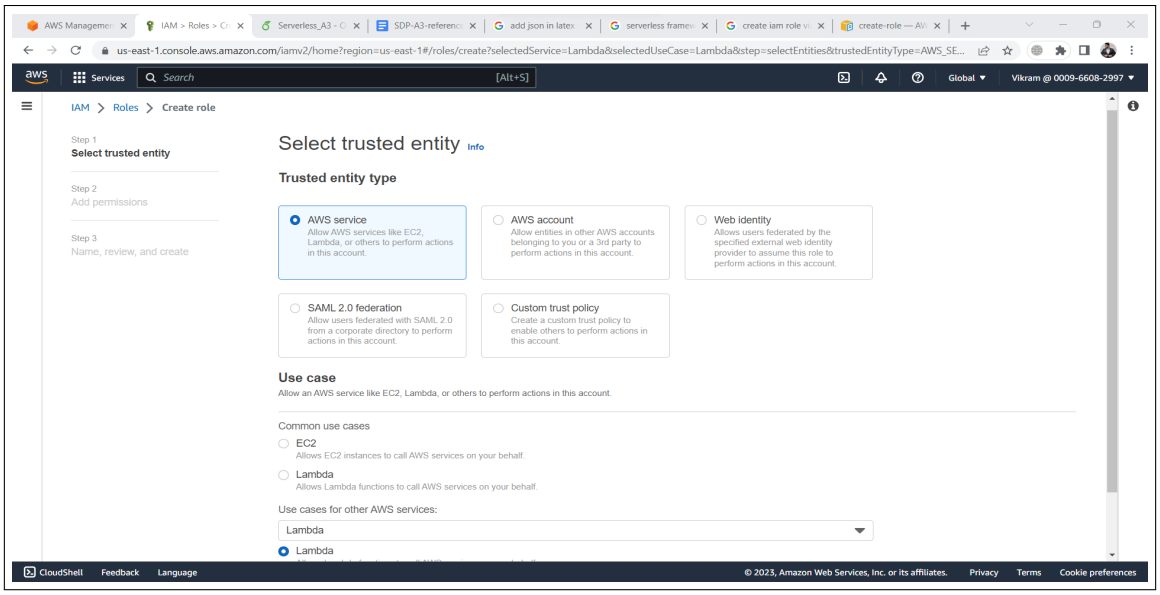

Figure 4.4.: ***Create IAM role for lambda extractFeatures***

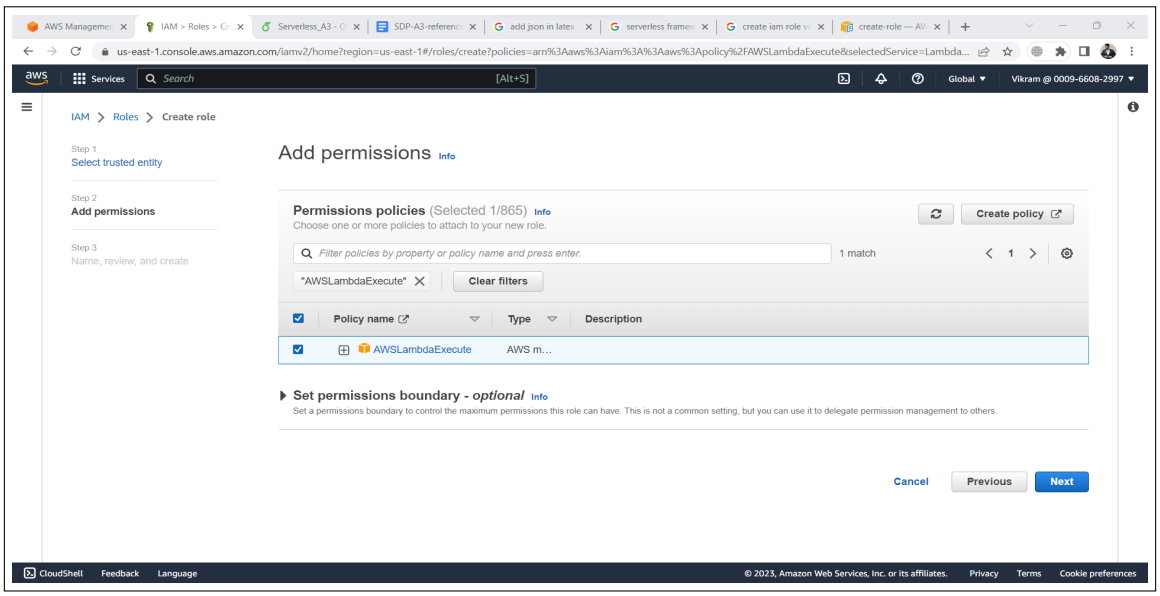

Figure 4.5.: ***Create IAM role for lambda extractFeatures***

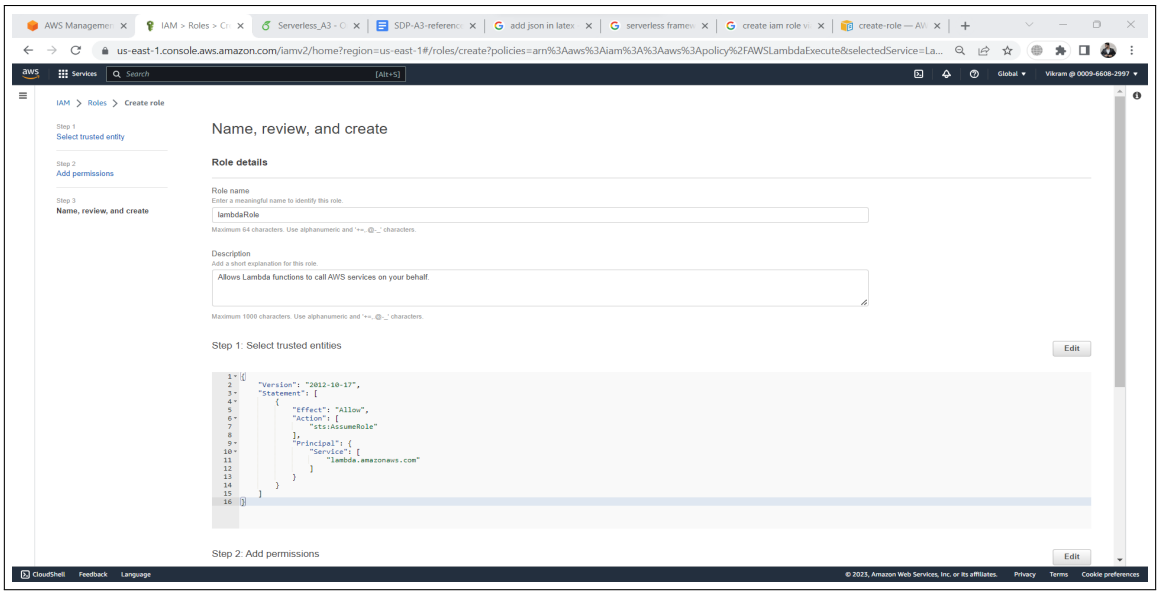

Figure 4.6.: ***Create IAM role for lambda extractFeatures***

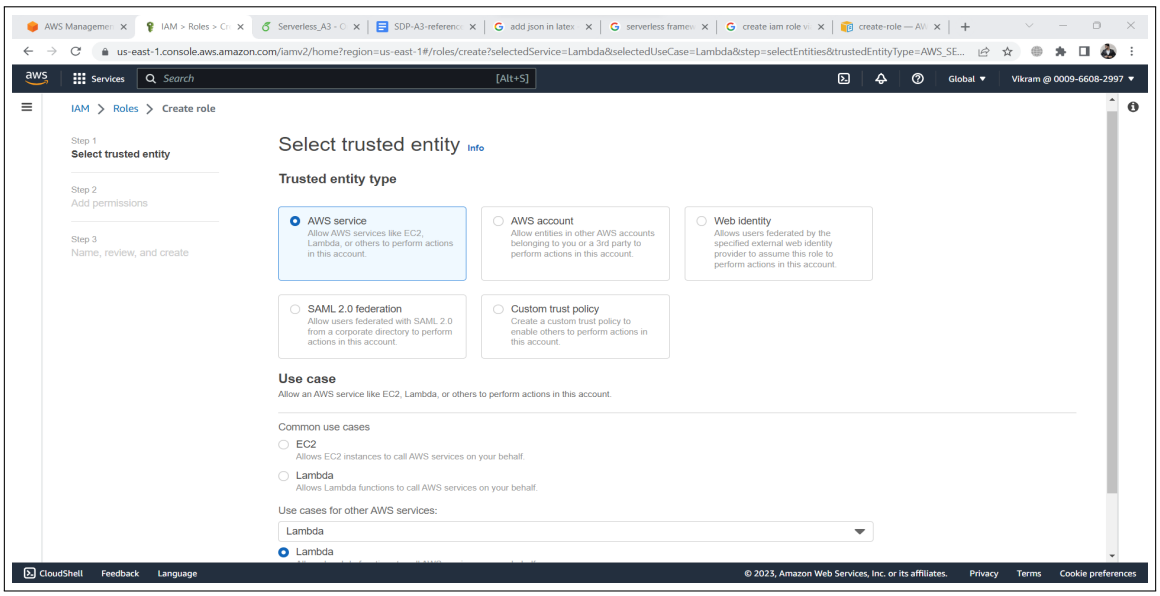

Figure 4.7.: ***Create IAM role for lambda accessDB***

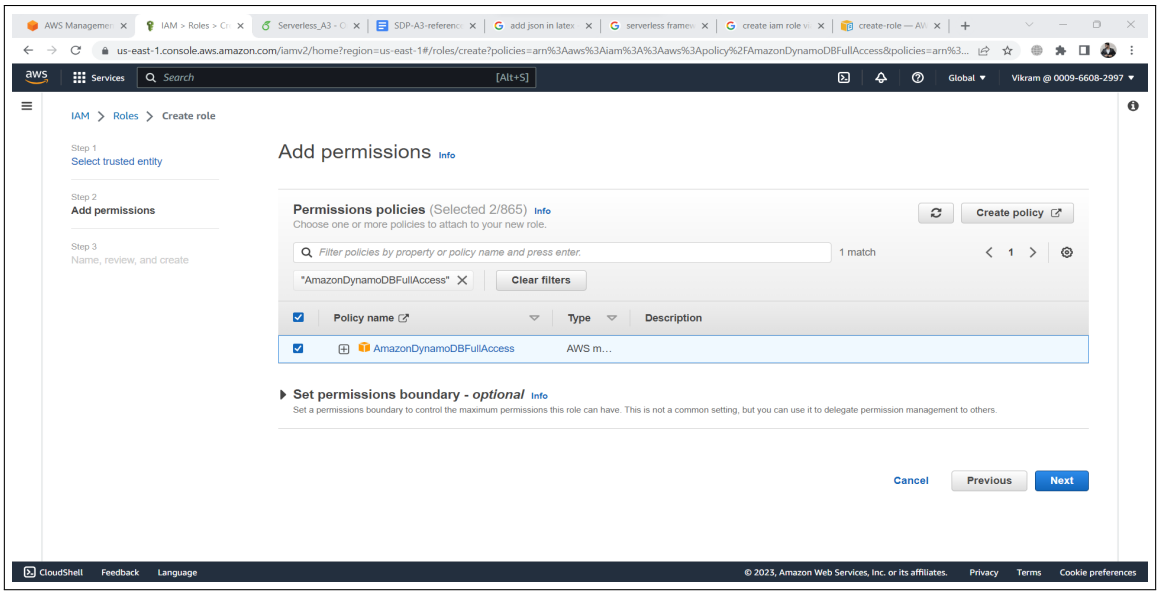

Figure 4.8.: ***Create IAM role for lambda accessDB***

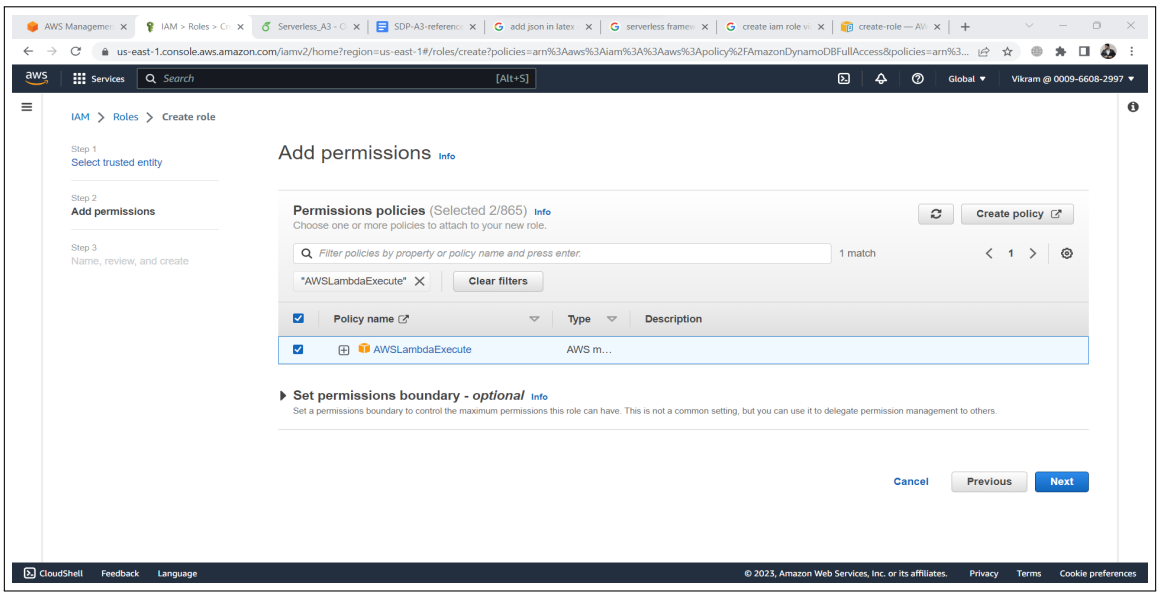

Figure 4.9.: ***Create IAM role for lambda accessDB***

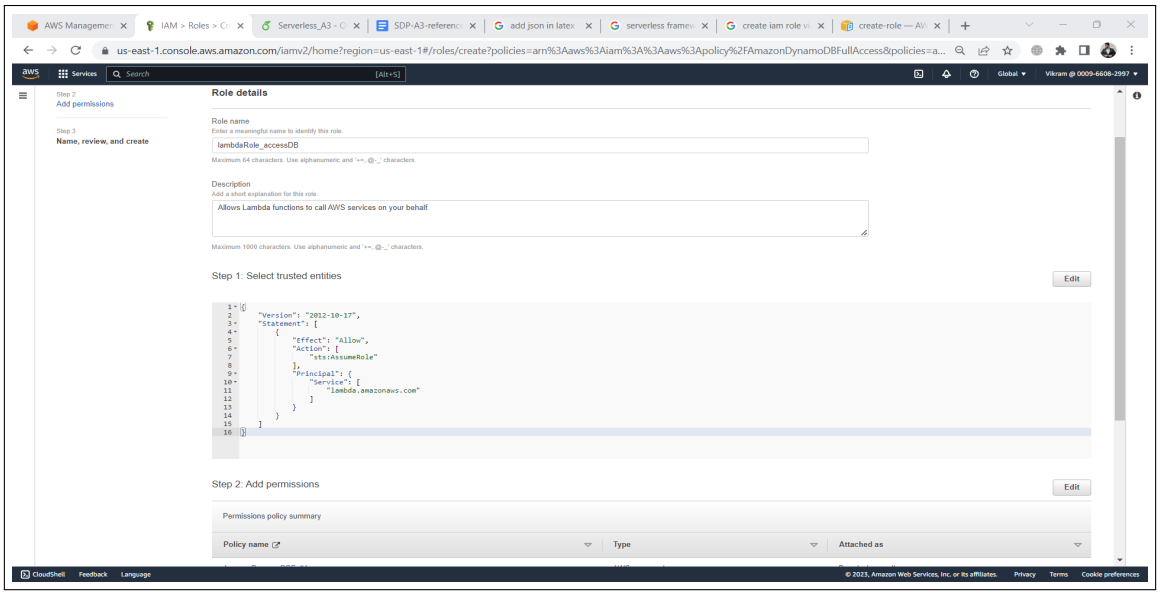

Figure 4.10.: ***Create IAM role for lambda accessDB***

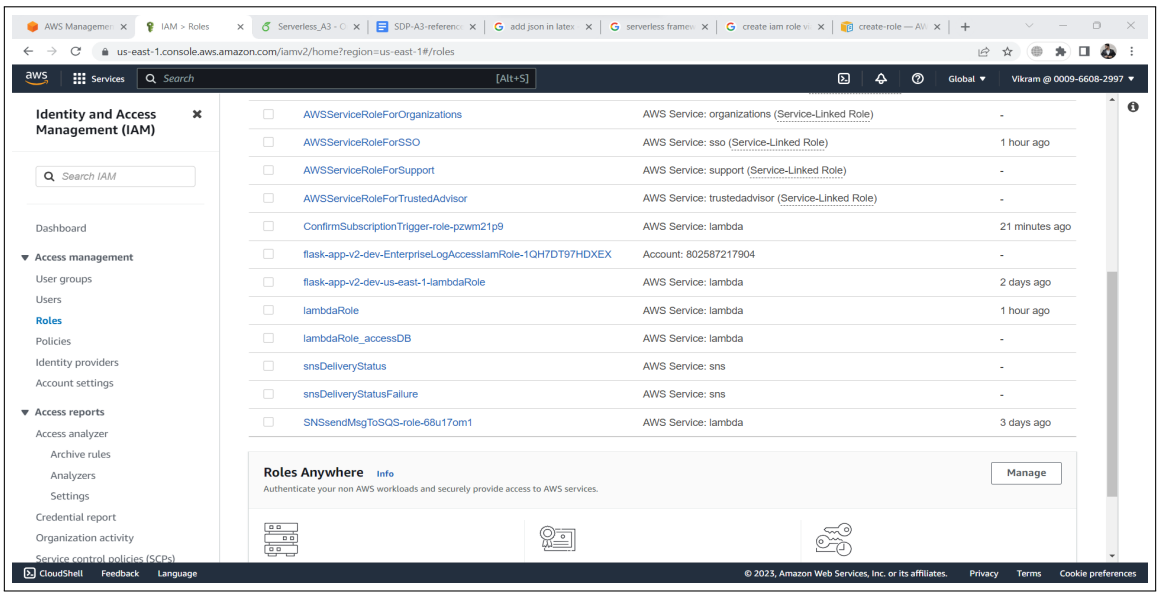

Figure 4.11.: ***Both the Roles created***

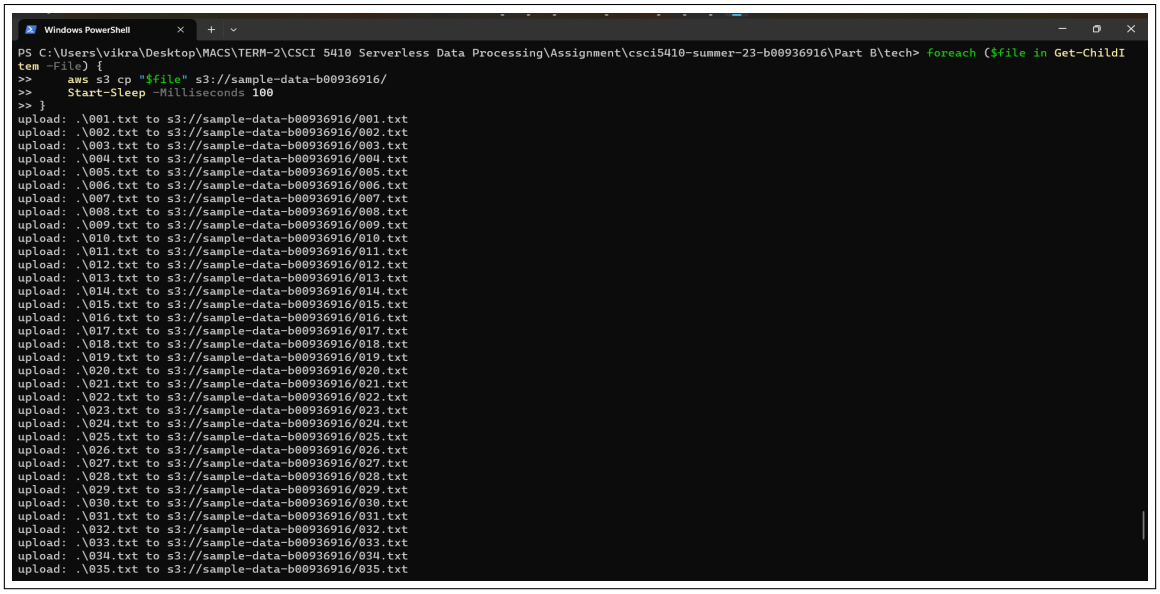

Figure 4.12.: ***Upload the files in folder 'tech' to bucket sample-data-b00936916 via CLI***

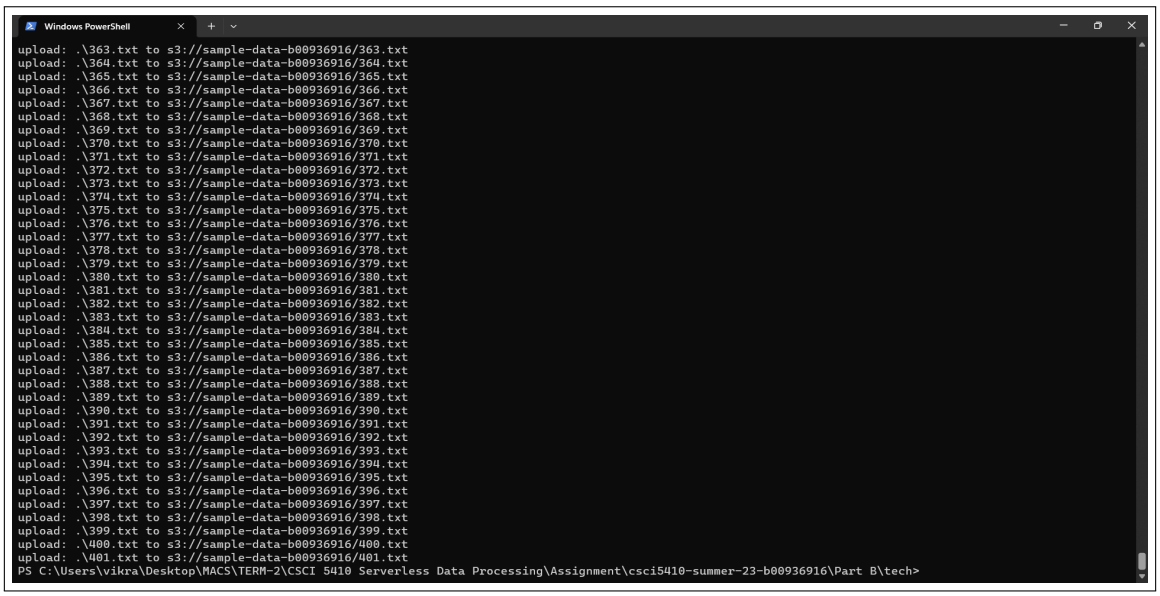

Figure 4.13.: ***Upload the files in folder 'tech' to bucket sample-data-b00936916 via CLI - completed***

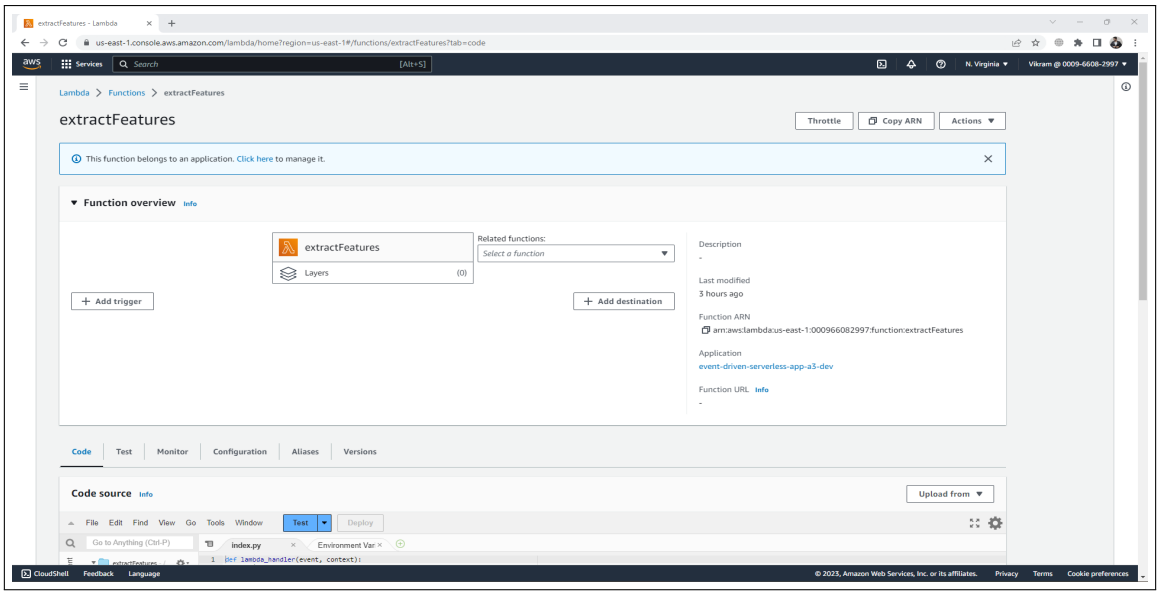

Figure 4.14.: ***extractFeatures lambda before adding trigger***

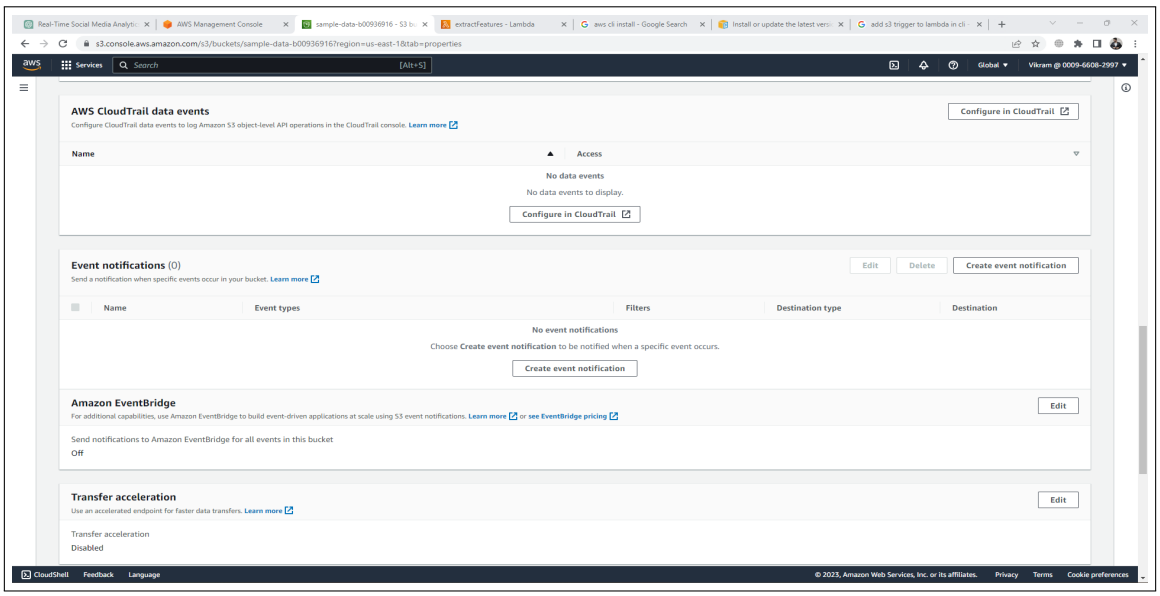

Figure 4.15.: ***sample-data-b00936916 before adding event notification***

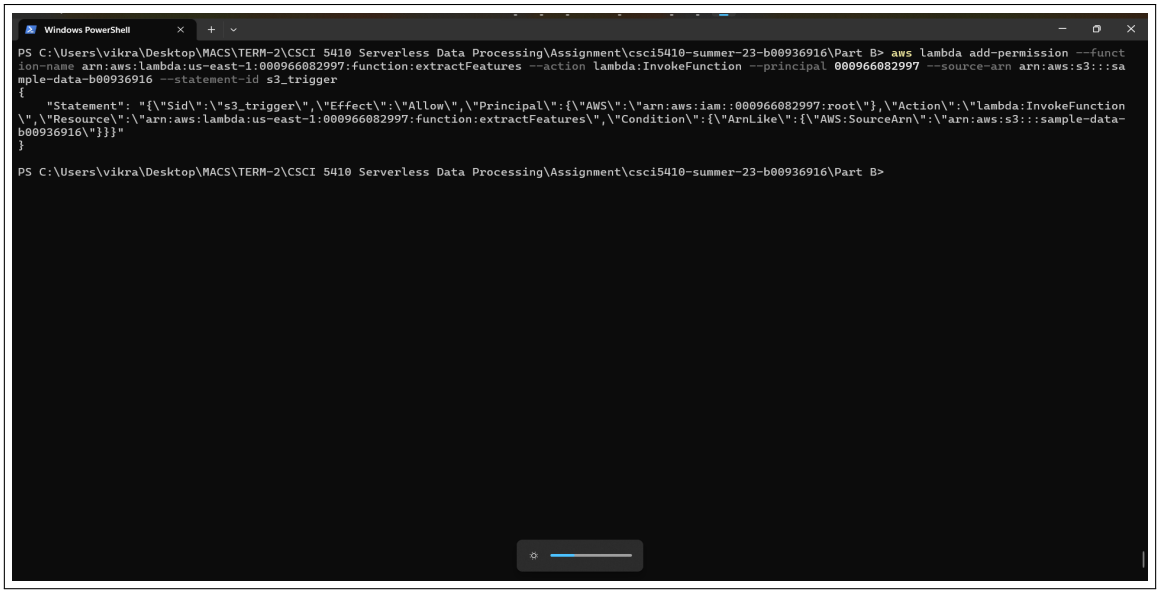

Figure 4.16.: ***add InvokeFunction permission for lambda sample-data-b00936916 via CLI***

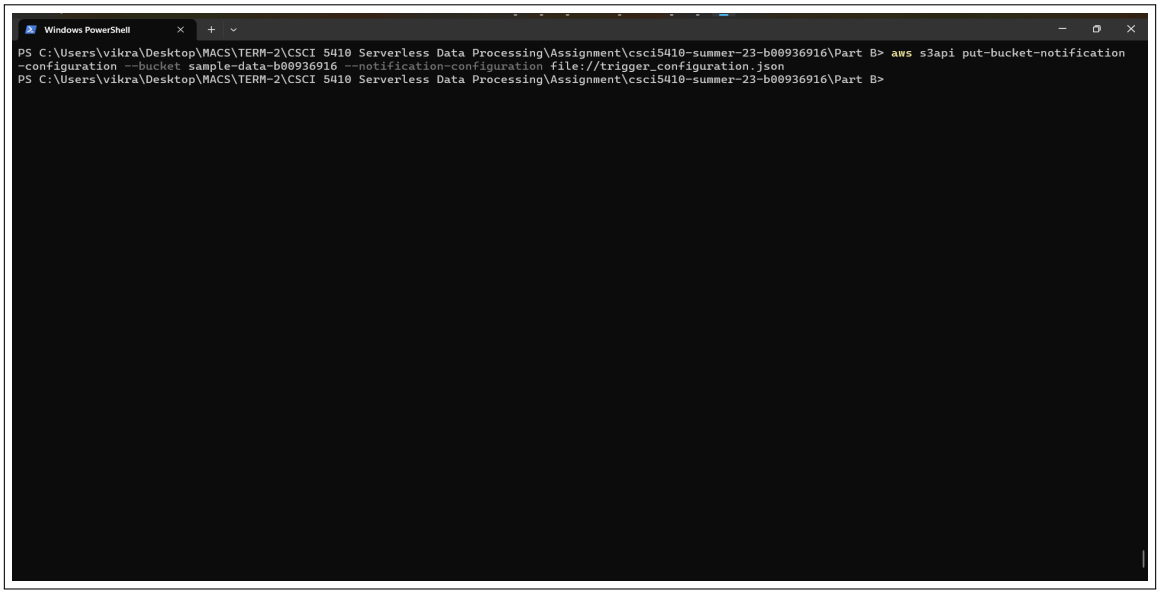

Figure 4.17.: ***Add s3 trigger via CLI for extractFeatures lambda***

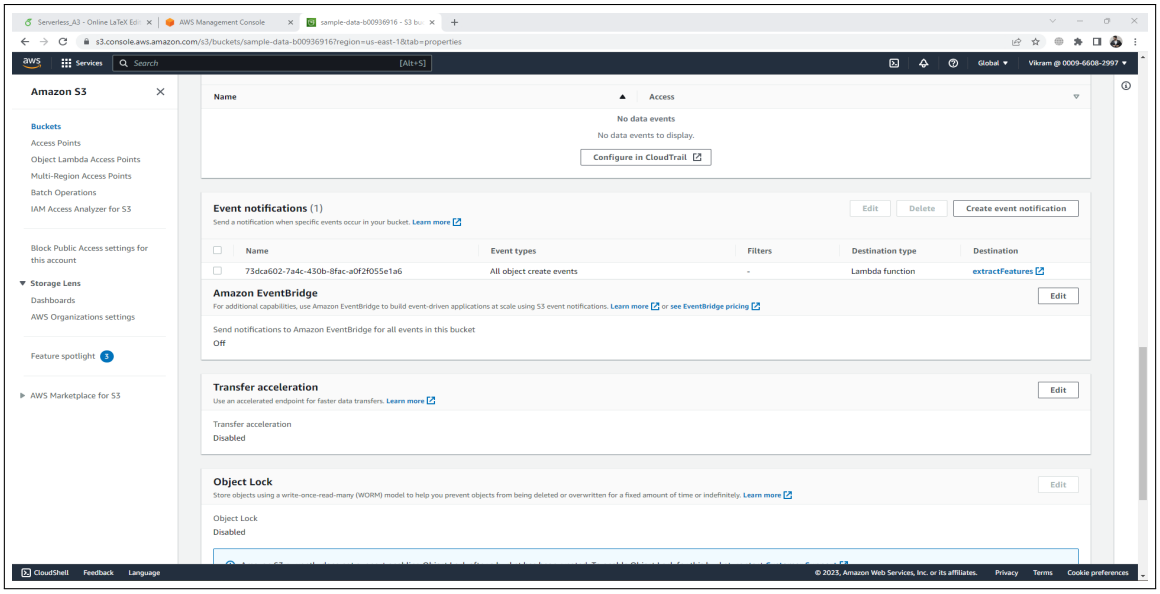

Figure 4.18.: ***S3 event notification - bucket sample-data-b00936916***

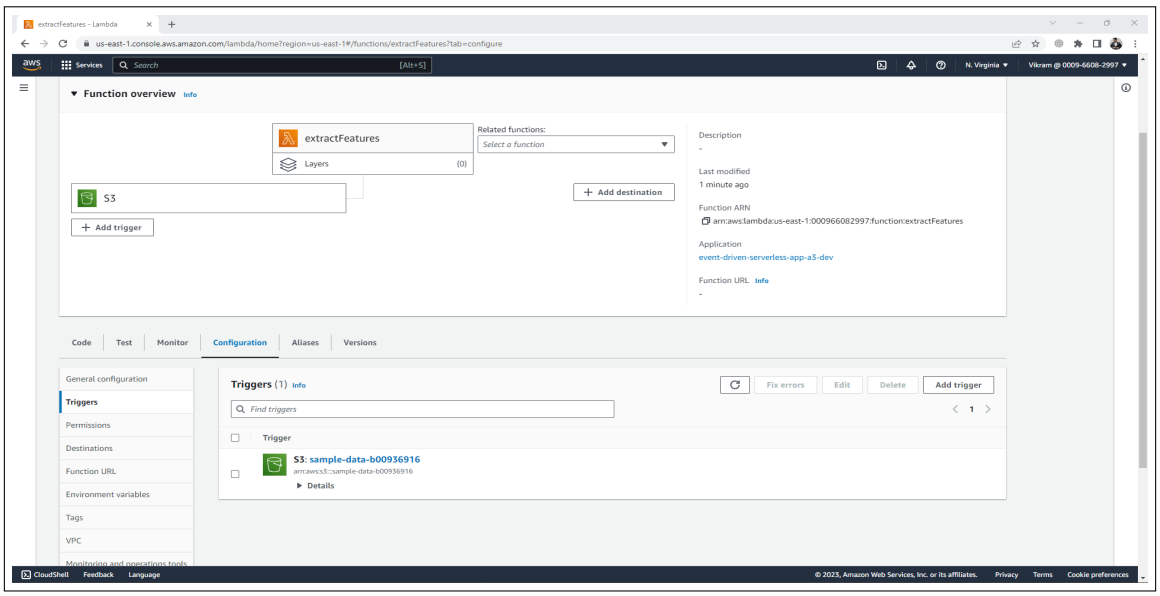

Figure 4.19.: ***S3 trigger added for lambda extractFeatures***

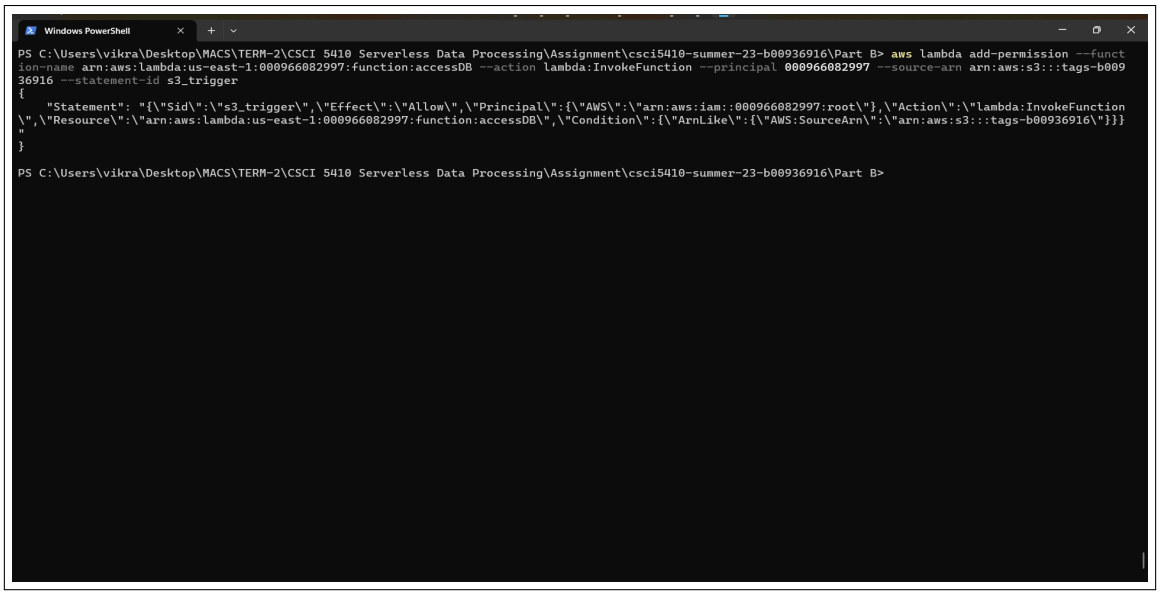

Figure 4.20.: ***add InvokeFunction permission for lambda tags-b00936916 via CLI***

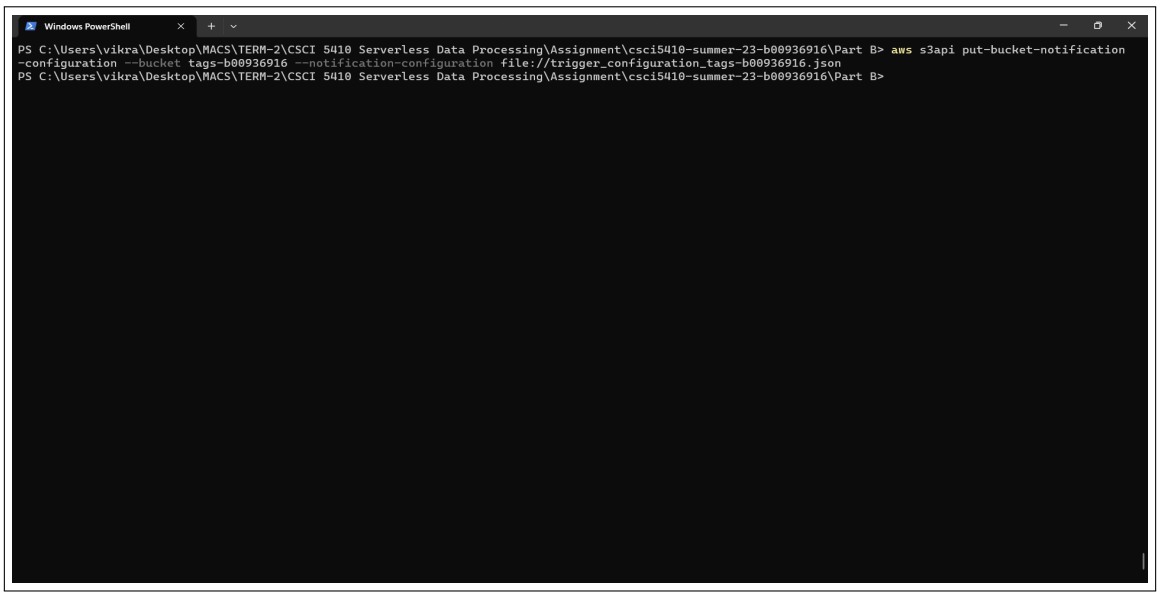

Figure 4.21.: ***Add s3 trigger via CLI for accessDB lambda***

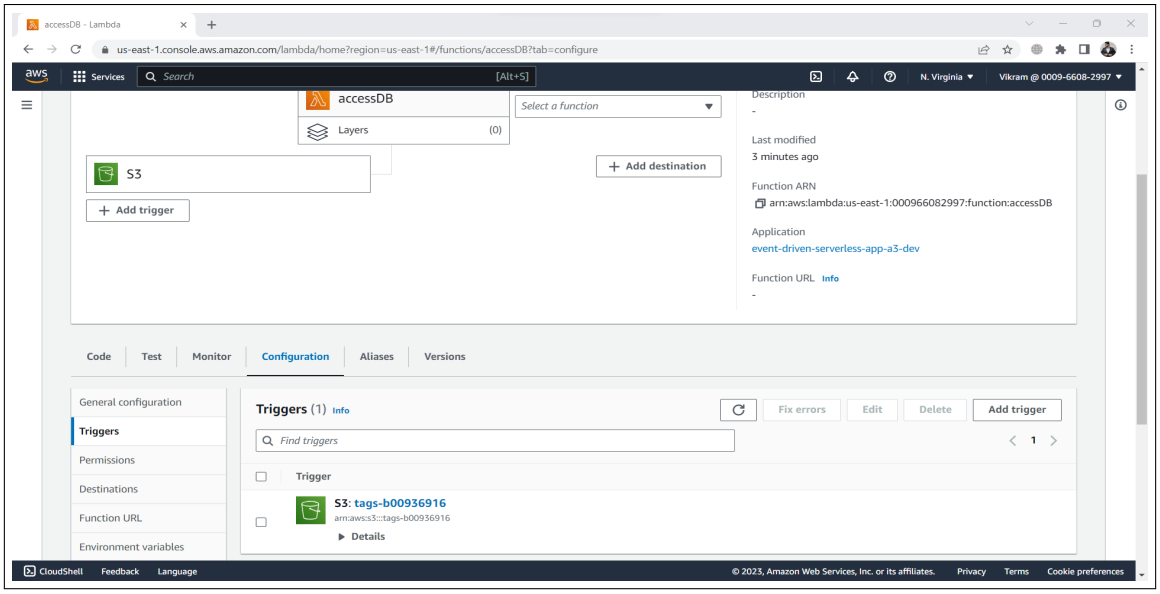

Figure 4.22.: ***S3 trigger added for lambda accessDB***

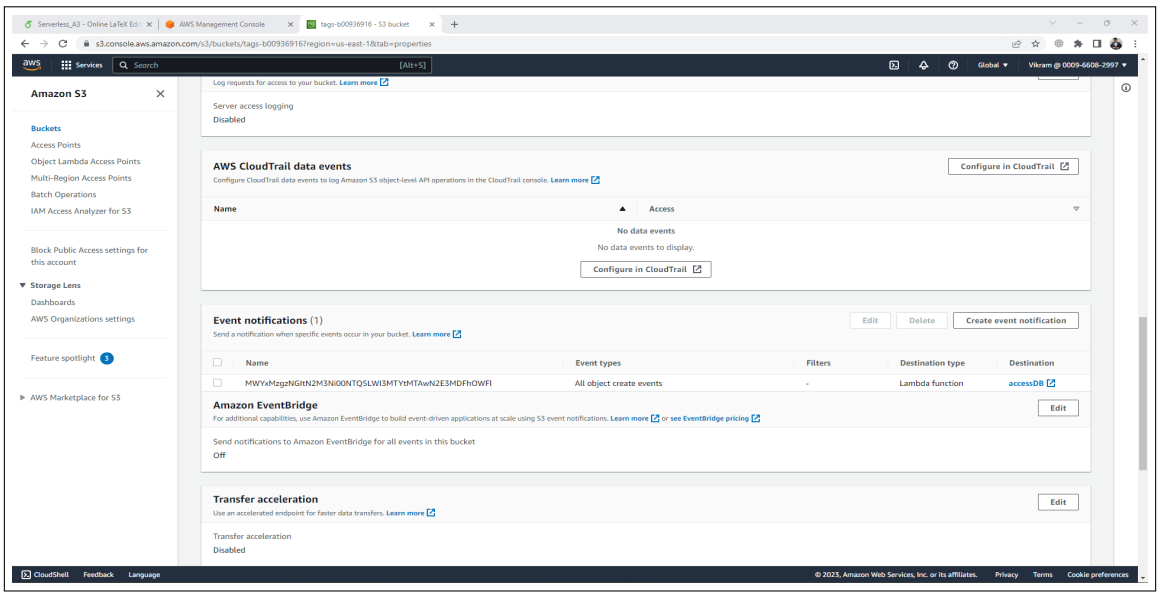

Figure 4.23.: ***S3 event notification for bucket tags-b00936916***

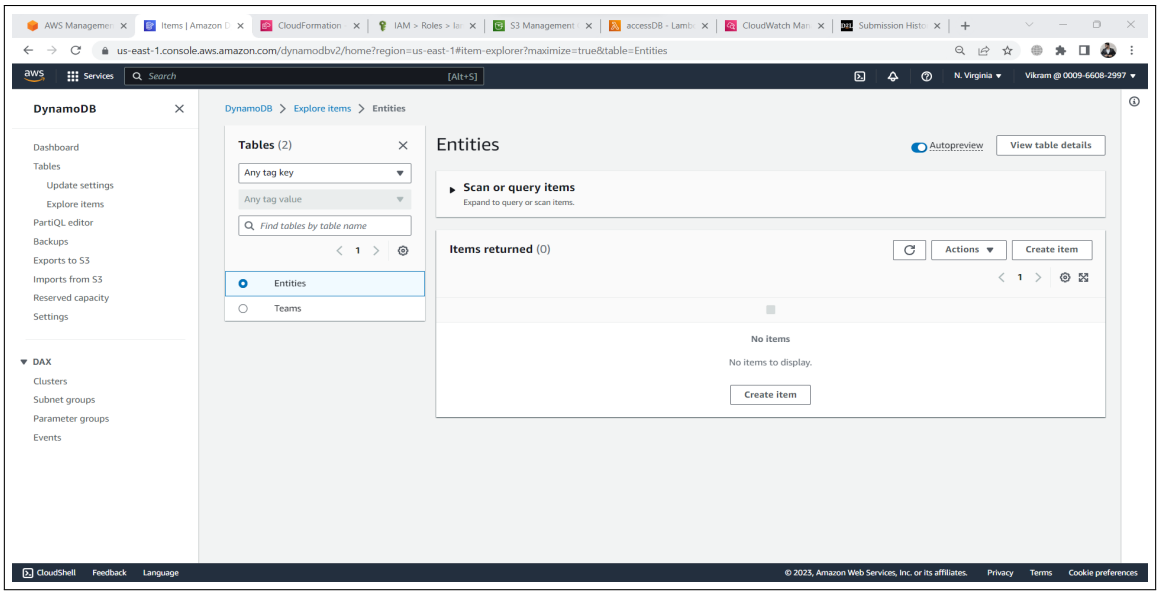

Figure 4.24.: ***Empty DynamoDB table 'Entities'***

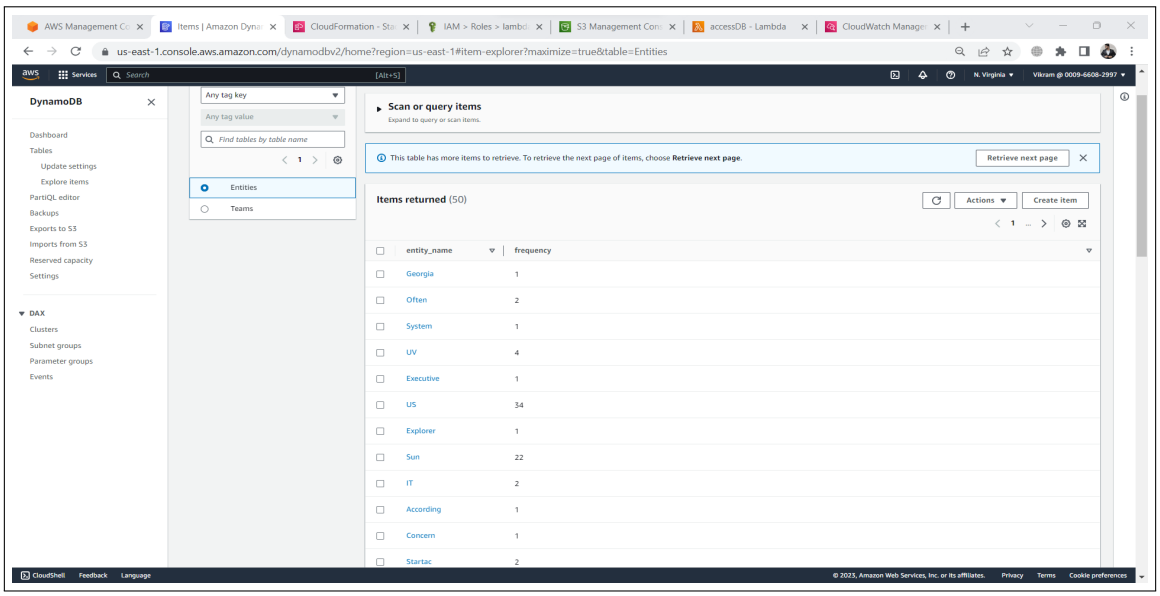

Figure 4.25.: ***Final DynamoDB table 'Entities' after all entities are extracted***

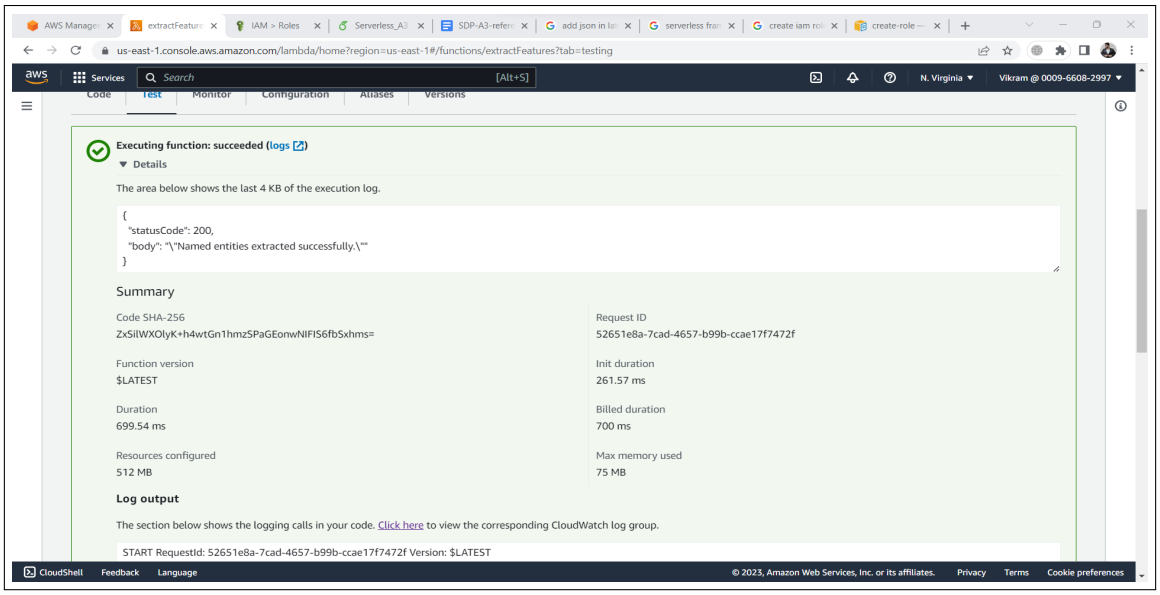

Figure 4.26.: ***extractFeatures lambda unit test***

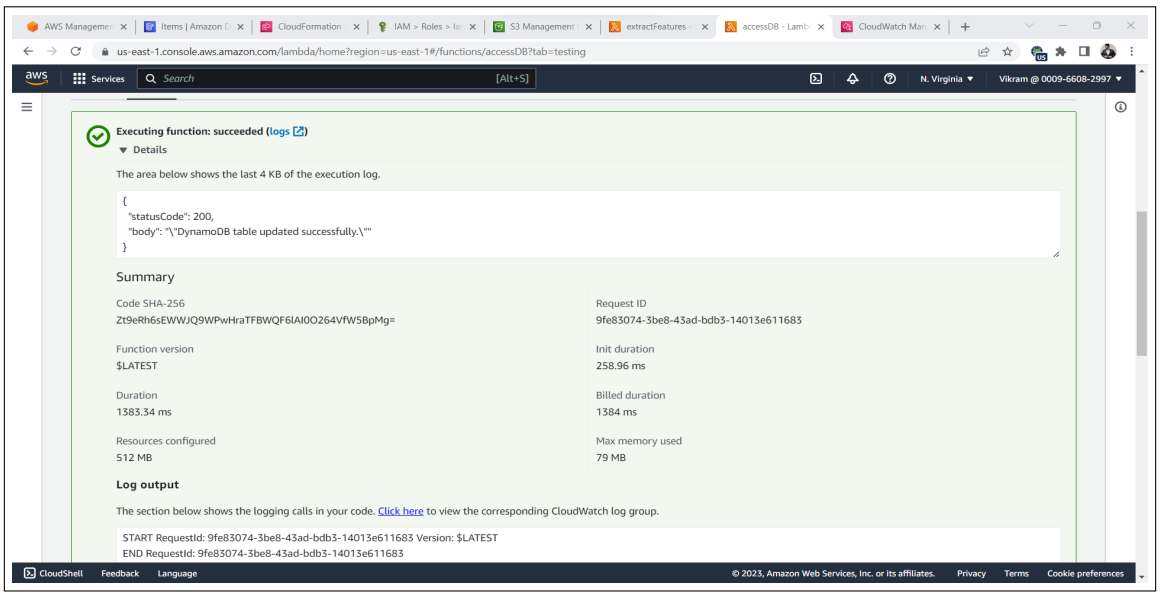

Figure 4.27.: ***accessDB lambda unit test***

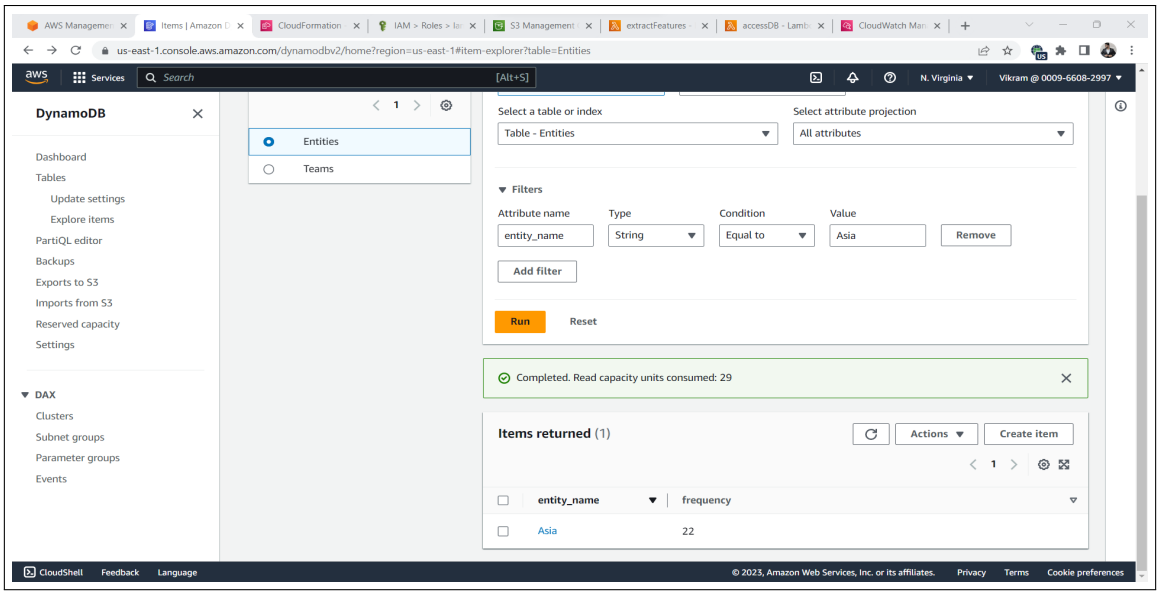

Figure 4.28.: ***Sample query with entity_name="Asia"***

4.4. Testing

4.4.1. Procedure

1. Created a new file named '000.txt' with the following content:

```
sample txt file for testing purposes:

all lowercase
                : ggggggg
all uppercase
                : GGGGGGG
Starting uppercase,
other lowercase
                : Ggggggg
numbers
                : 0000000
```

2. Named entities in this file are: GGGGGGG, Starting, Ggggggg
3. Uploaded the file via CLI.
4. Verfied the file upload in bucket sample-data-b00936916 on console.
5. Verified the triggering of events by checking the logs of lambda functions: extractFeatures, accessDB.
6. Verified the file contents of generated '000ne.txt' file in bucket tags-b00936916 on console, by downloading the file.
7. Queried the extracted entities in DynamoDB and verified their frequency.

4.4.2. Screenshots

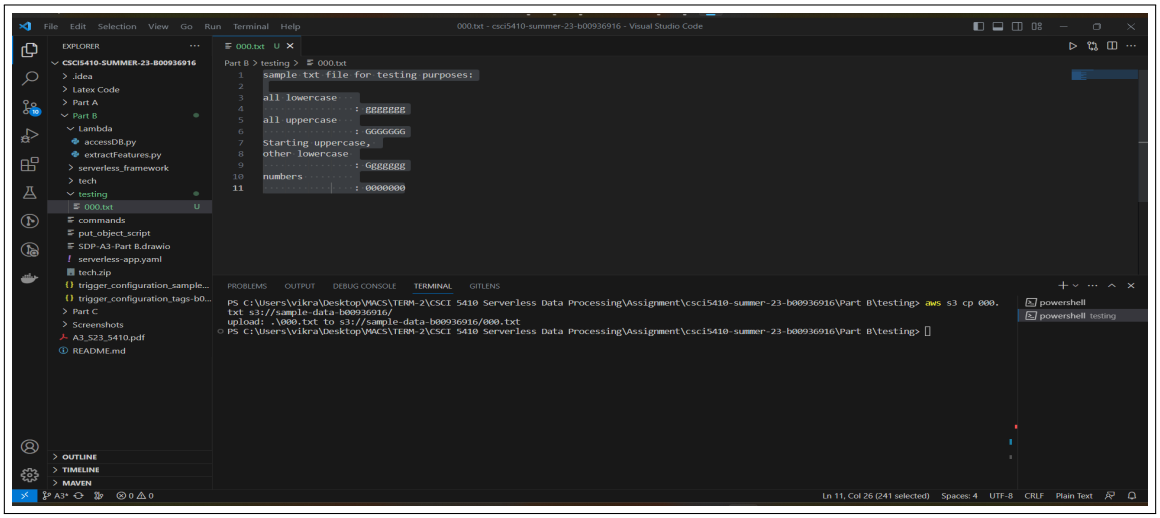

Figure 4.29.: ***Upload 000.txt via CLI***

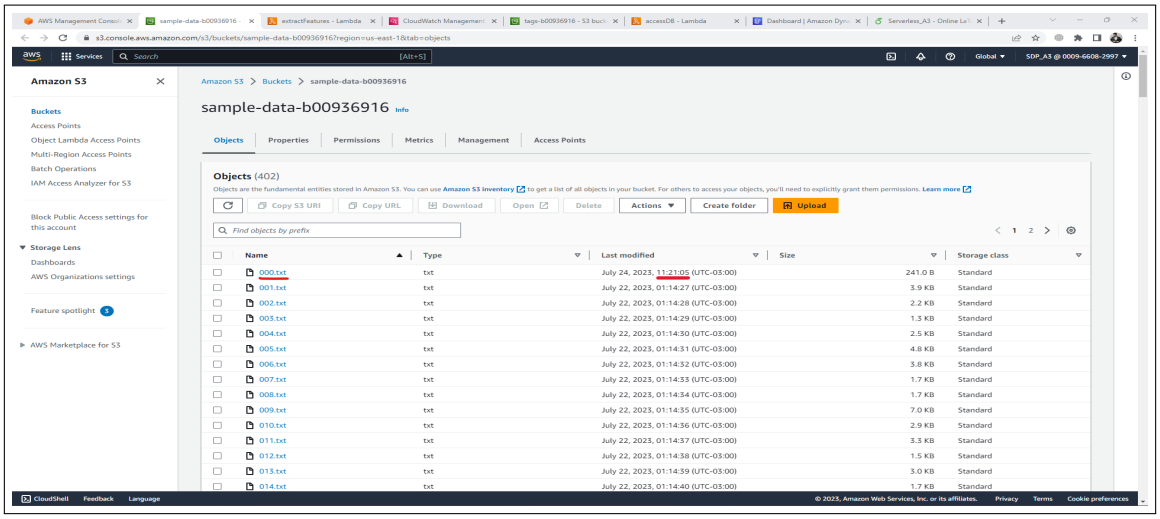

Figure 4.30.: ***File uploaded in bucket sample-data-b00936916 - console***

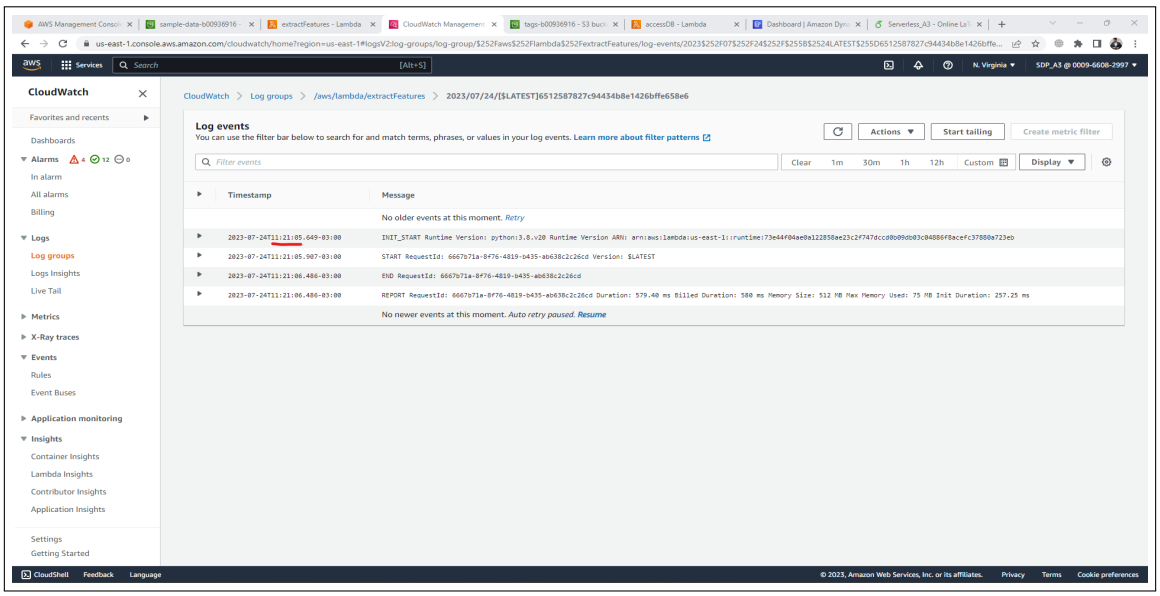

Figure 4.31.: ***Logs in extractFeatures lambda - notice the timestamp***

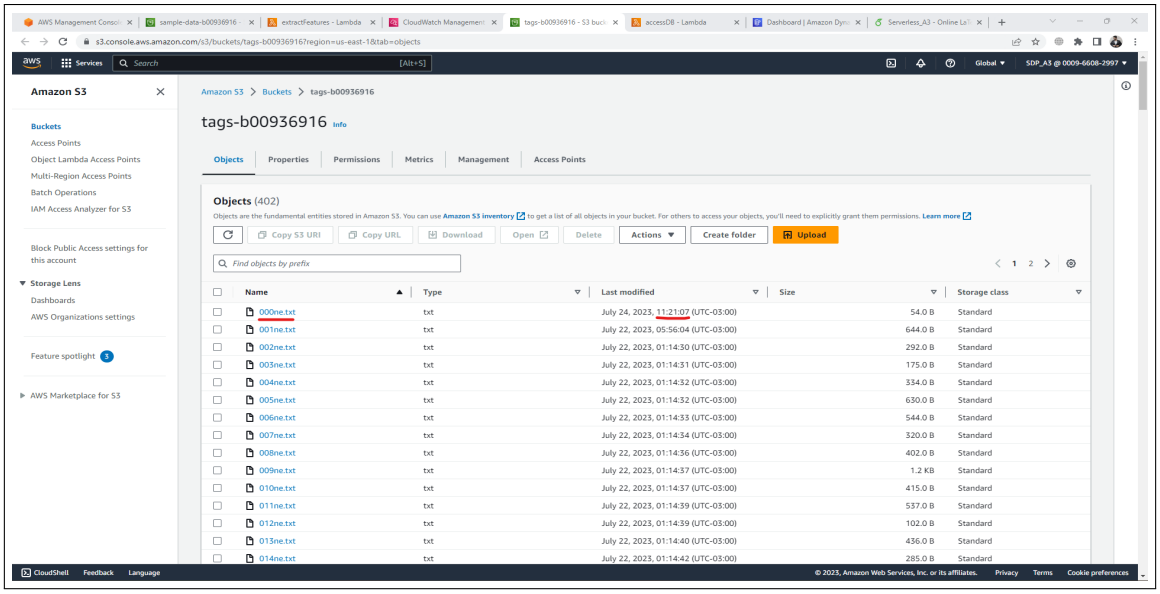

Figure 4.32.: ***000ne.txt file in bucket tags-b00936916***

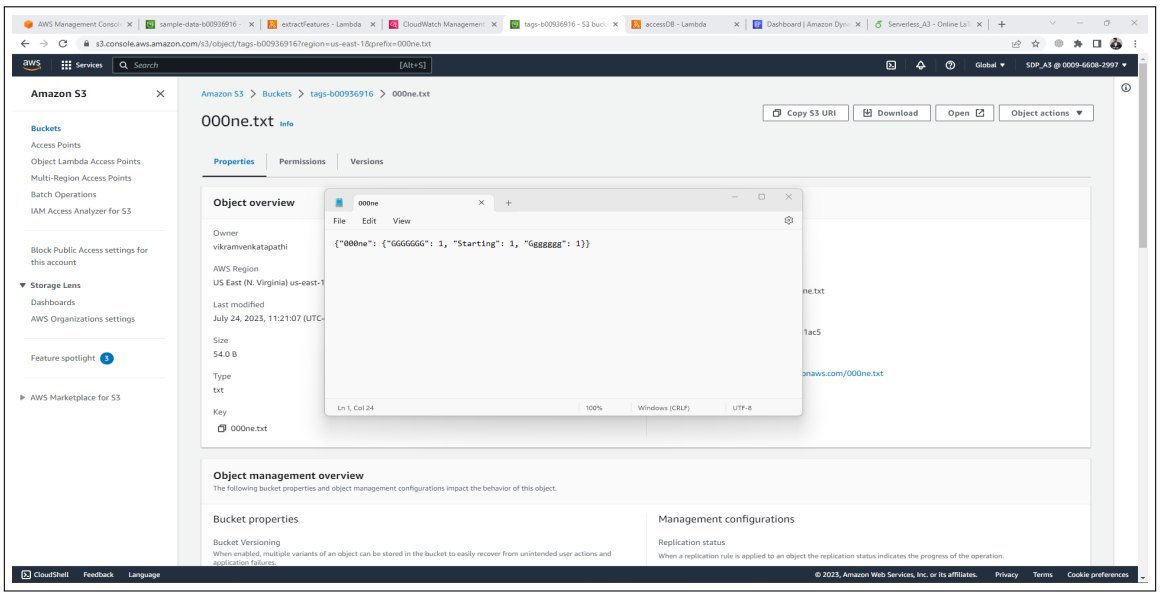

Figure 4.33.: ***000ne.txt file contents - with named entities***

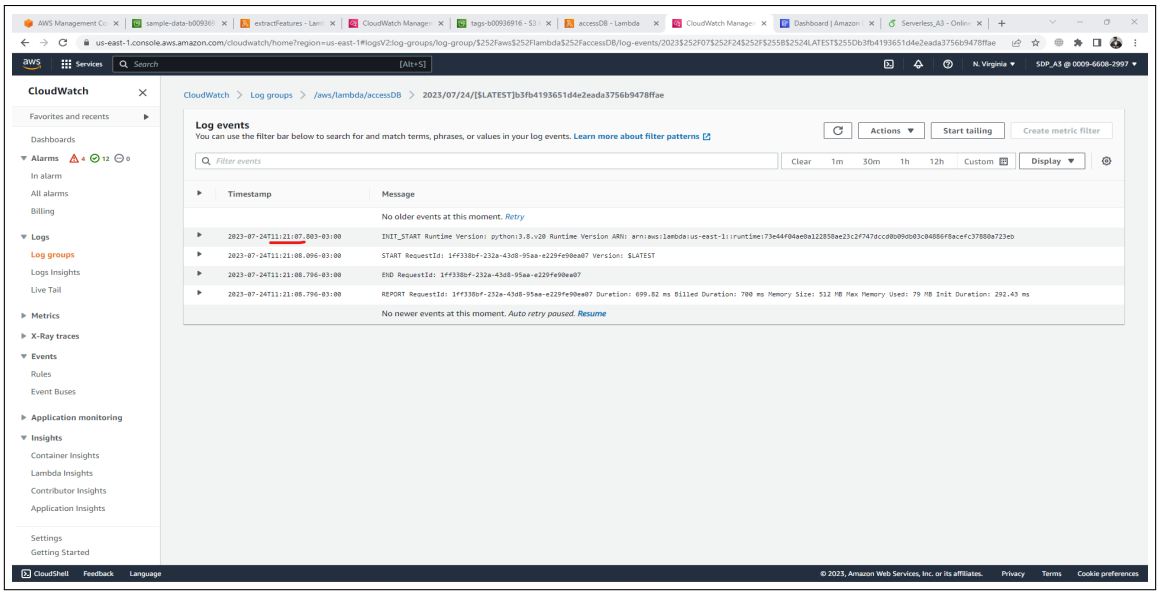

Figure 4.34.: ***Logs in accessDB lambda - notice the timestamp***

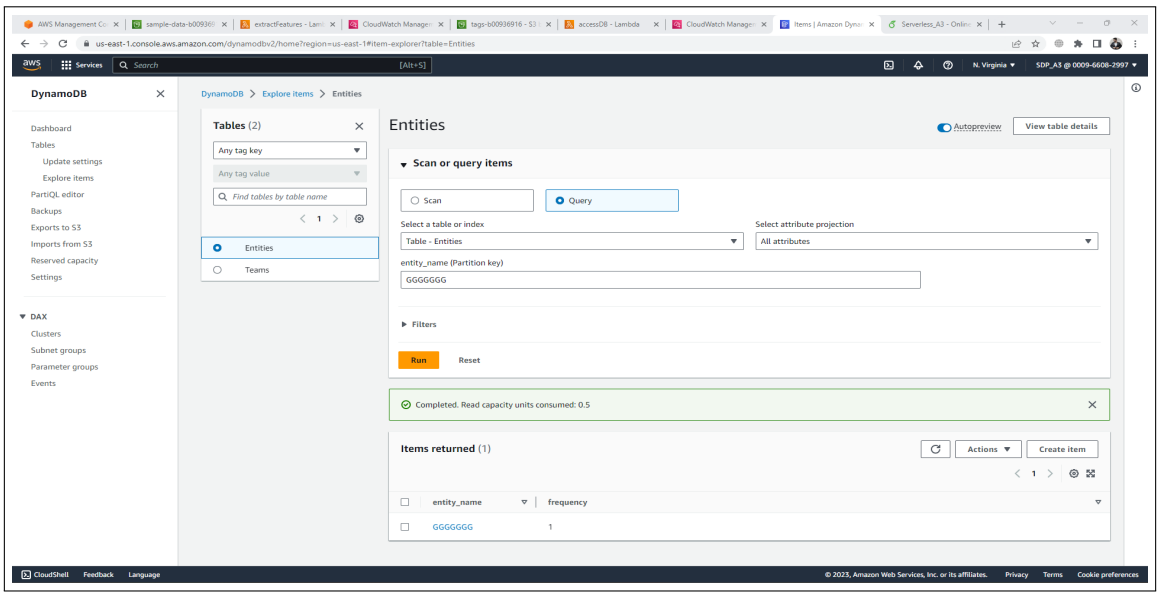

Figure 4.35.: ***Query : entity_name=GGGGGGG, frequency=1 in table***

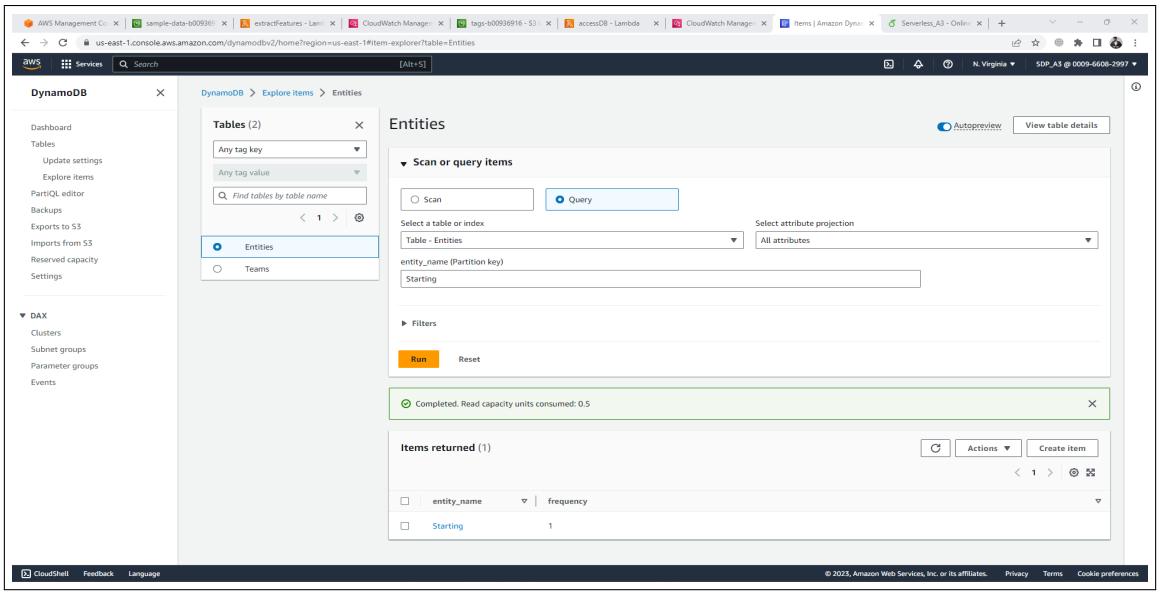

Figure 4.36.: ***Query : entity_name=Starting, frequency=1 in table***

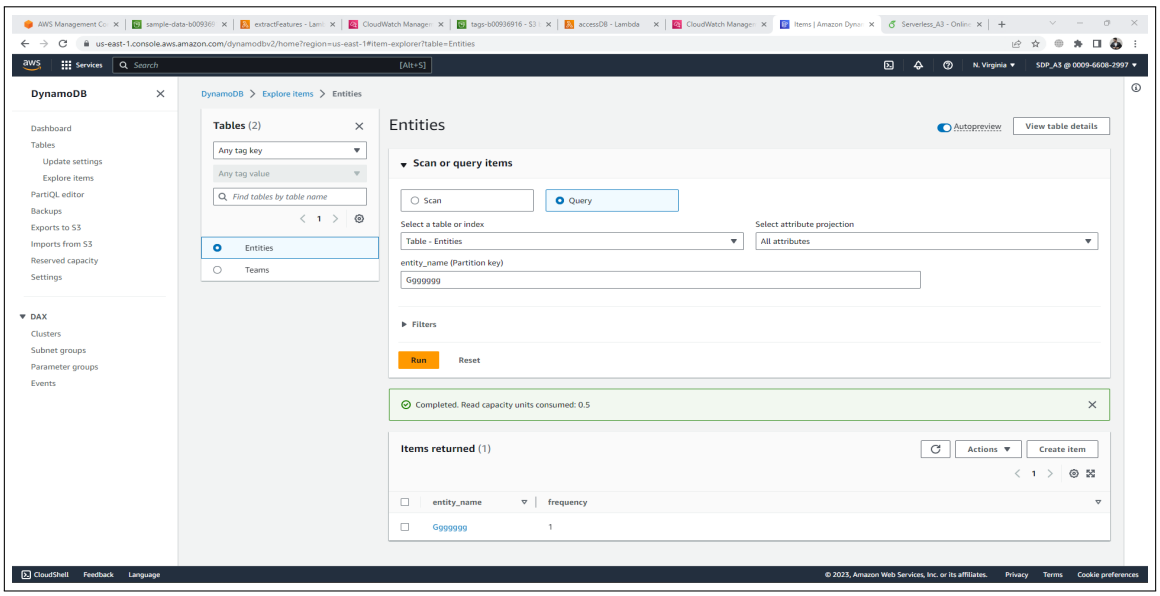

Figure 4.37.: ***Query : entity_name=Ggggggg, frequency=1 in table***

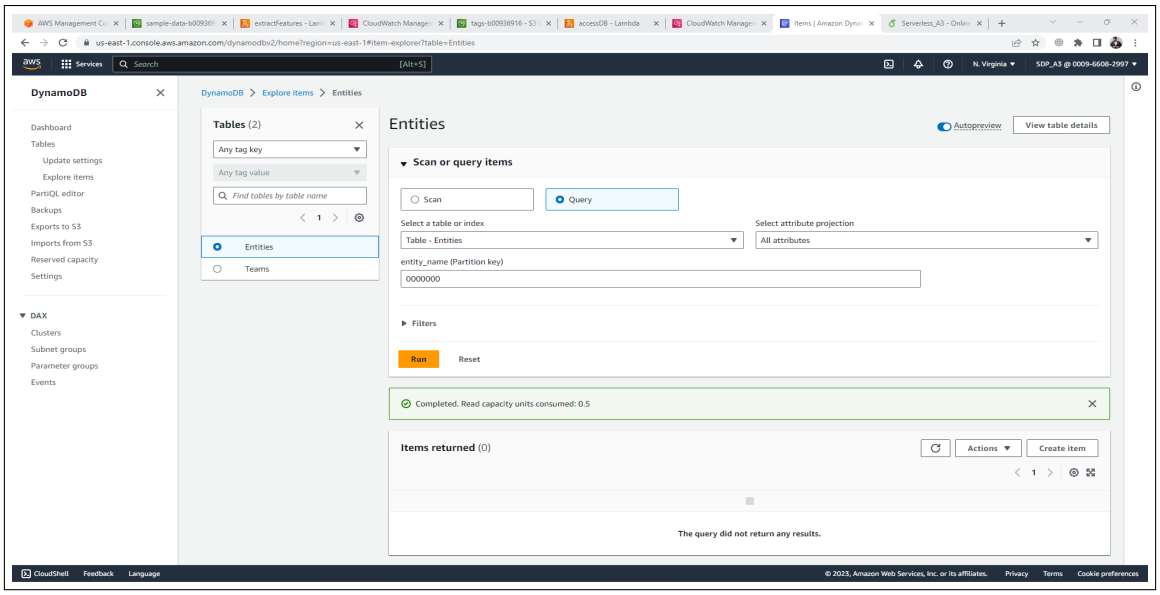

Figure 4.38.: ***Query : entity_name=0000000, frequency=0 in table***

Part III.

PART C

Use AWS Lambda-SQS-SNS

5. Car Service

5.1. AWS Architecture diagram of my implementation

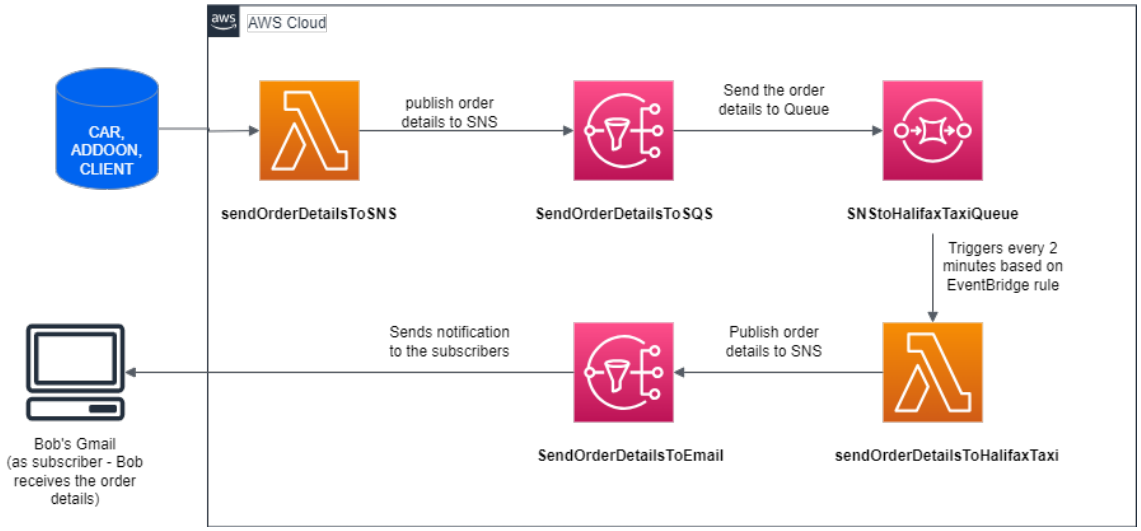

Figure 5.1.: ***AWS Architecture diagram for the given work flow (Created using: draw.io [23])***

5.2. Steps

1. Create an IAM role with permissions to be : invoked, publish messages to SNS, push messages to SQS and attach to the lambda functions.

2. Create a JSON file: **AssumeRolePolicyDocument.json** with the following contents (IAM role creation) [24]

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
```

```
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

3. Create another JSON file: **invokeFunction.json** with the following contents (add invokeFunction permission)

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "*"

    ]
}
```

4. Now, run the following commands to create the IAM role: **lambdaRole-Part-C specifying the path** to the above-created json file using **file://</path/to/file_name>.json** [6]

```
aws iam create-role --role-name lambdaRole-Part-C --assume-role-policy-
    document file://AssumeRolePolicyDocument.json

aws iam put-role-policy --role-name lambdaRole-Part-C --policy-name
    LambdaExecutionPolicy --policy-document file://invokeFunction.json
```

5. Run the following commands to add the permissions to the created lambda role **lambdaRole-Part-C** [25]

```
# AWSLambdaExecute role with permissions - Put, Get access to S3 and
    full access to CloudWatch Logs.
aws iam attach-role-policy --role-name lambdaRole-Part-C --policy-arn
    arn:aws:iam::aws:policy/AWSLambdaExecute

# AmazonSNSFullAccess role with permissions - Full access to SNS
aws iam attach-role-policy --role-name lambdaRole-Part-C --policy-arn
    arn:aws:iam::aws:policy/AmazonSNSFullAccess

# AmazonSQSFullAccess role with permissions - Full access to SQS
aws iam attach-role-policy --role-name lambdaRole-Part-C --policy-arn
```

```
    arn:aws:iam::aws:policy/AmazonSQSFullAccess
```

6. Run the following command to create a serverless-framework project [3]

```
# Create a new AWS Python3 Serverless project at path 'CarService'.
serverless create --template aws-python3 --path CarService
```

7. Create the required resources using the following serverless framework script[3] (serverless.yml)[4]

```
service: CarService
provider:
  name: aws
  runtime: python3.8
  region: us-east-1
  stage: dev

resources:
  Resources:
    # 1st Lambda Function to Send order details to SNS
    SendOrderDetailsToSNS:
      Type: AWS::Lambda::Function
      Properties:
        FunctionName: sendOrderDetailsToSNS
        Code:
          ZipFile: |
            import json
            def lambda_handler(event, context):
                # TODO implement
                return {
                    'statusCode': 200,
                    'body': json.dumps('Hello␣from␣Lambda!')
                }
        Handler: app.lambda_handler  # Ensure that the handler function
            name matches the code
        Runtime: python3.8
        MemorySize: 512
        Timeout: 10
        Role: arn:aws:iam::000966082997:role/lambdaRole-Part-C

    # 2nd Lambda Function to Send Order Details to Halifax Taxi
    SendOrderDetailsToHalifaxTaxi:
      Type: AWS::Lambda::Function
      Properties:
        FunctionName: sendOrderDetailsToHalifaxTaxi
        Code:
          ZipFile: |
```

```
        import json
        def lambda_handler(event, context):
            # TODO implement
            return {
                'statusCode': 200,
                'body': json.dumps('Hello from Lambda!')
            }
    Handler: app.handler  # Ensure that the handler function name
       matches the code
    Runtime: python3.8
    MemorySize: 512
    Timeout: 10
    Role: arn:aws:iam::000966082997:role/lambdaRole-Part-C

# 1st SNS Topic - To take order details from Lambda and send to SQS
SnsTopicToSqs:
  Type: AWS::SNS::Topic
  Properties:
    TopicName: SendOrderDetailsToSQS

# 1st SNS Topic Subscription - Lambda subscribing to the SNS topic
SnsSubscriptionToSqs:
  Type: AWS::SNS::Subscription
  Properties:
    TopicArn: !Ref SnsTopicToSqs
    Protocol: lambda
    Endpoint:
      Fn::GetAtt:
        - SendOrderDetailsToSNS
        - Arn

# 2nd SNS Topic - To poll order details from SQS and publish to
   email
SnsTopicToEmail:
  Type: AWS::SNS::Topic
  Properties:
    TopicName: SendOrderDetailsToEmail
    Subscription:
      - Protocol: email
        Endpoint: vikramvenkatapathi@gmail.com

# 2nd SNS Topic Subscription - Lambda subscribing to the SNS topic
SnsSubscriptionToEmail:
  Type: AWS::SNS::Subscription
  Properties:
    TopicArn: !Ref SnsTopicToEmail
    Protocol: lambda
    Endpoint:
```

```
        Fn::GetAtt:
          - SendOrderDetailsToHalifaxTaxi
          - Arn
  SNStoHalifaxTaxiQueue:
    Type: AWS::SQS::Queue
    Properties:
      QueueName: SNStoHalifaxTaxiQueue

  # SQS Queue Subscription - Subscribe SQS to SNS topic
  SqsSubscriptionToSns:
    Type: AWS::SNS::Subscription
    Properties:
      TopicArn: !Ref SnsTopicToSqs
      Protocol: sqs
      Endpoint:
        Fn::GetAtt:
          - SNStoHalifaxTaxiQueue
          - Arn

  # CloudWatch Events Rule to trigger Lambda function periodically
     every 2 minutes
  SendOrderDetailsToHalifaxTaxiSchedule:
    Type: AWS::Events::Rule
    Properties:
      ScheduleExpression: rate(2 minutes)
      Targets:
        - Arn: arn:aws:lambda:us-east-1:000966082997:function:
            sendOrderDetailsToHalifaxTaxi
          Id: SendOrderDetailsToHalifaxTaxiScheduleTarget
```

8. The above script does the following
   a) Create 2 - lambdas: SendOrderDetailsToSNS, SendOrderDetailsToHalifaxTaxi
   b) Create 2 - SNS topics: SendOrderDetailsToSQS, SendOrderDetailsToEmail
   c) Create 1 - SQS Queue: SNStoHalifaxTaxiQueue
   d) Create 2 subscriptions for SNS topic SendOrderDetailsToSQS
      i. Lambda - SendOrderDetailsToSNS
      ii. SQS - SNStoHalifaxTaxiQueue
   e) Create 2 subscriptions for SNS topic SendOrderDetailsToEmail
      i. Lambda - sendOrderDetailsToHalifaxTaxi
      ii. Bob's Email - vikramvenkatapathi@gmail.com (my personal email)
   f) Create a CloudWatch Events Rule - SendOrderDetailsToHalifaxTaxiSchedule, to trigger Lambda function: sendOrderDetailsToHalifaxTaxi, periodically every 2 minutes.

9. Run the command **serverless deploy** to deploy the .yml file to create the resources in CloudFormation.

5.3. Lambda Code

5.3.1. sendOrderDetailsToSNS

```
import json
import boto3
import random

# Sample data lists
car_types = ["Compact", "Mid-size Sedan", "SUV", "Luxury"]
car_accessories = ["GPS", "Camera", "Sun shades", "Vehicle Seat Cover"]
client_addresses = ["123 Main St", "456 Elm St", "789 Oak St", "1333
    South park st"]

def lambda_handler(event, context):

    """
        W3schools - Python Random Module Documentation
        Reference: `W3schools - Python Random Module <https://www.
            w3schools.com/python/module_random.asp>`_
    """
    # Generate a single message by combining data from all three lists
    order_details = {
        "type": "ORDER",
        "data": {
            "CAR_TYPE": random.choice(car_types),
            "CAR_ACCESSORY": random.choice(car_accessories),
            "CLIENT_ADDRESS": random.choice(client_addresses)
        }
    }

    # Publish the combined message to the SNS topic
    sns_topic_arn = "arn:aws:sns:us-east-1:000966082997:
        SendOrderDetailsToSQS"
    publish_to_sns(sns_topic_arn, order_details)

    print(order_details)
    return {
        'statusCode': 200,
        'body': json.dumps('Message sent to SNS topic=
            SendOrderDetailsToSQS!')
    }
```

```
def publish_to_sns(topic_arn, message):

    """
        Boto3 - Amazon Amazon Simple Notification Service (SNS)
           Documentation
        Reference: `Boto3 - Amazon SNS API <https://boto3.amazonaws.com/
           v1/documentation/api/latest/reference/services/sns.html>`_
    """
    sns_client = boto3.client('sns')

    sns_client.publish(
        TopicArn=topic_arn,
        Message=json.dumps(message)
    )

"""
All references:
    Boto3 - Amazon Amazon Simple Notification Service (SNS)
       Documentation
    Reference: `Boto3 - Amazon SNS API <https://boto3.amazonaws.com/v1/
       documentation/api/latest/reference/services/sns.html>`_

    W3schools - Python Random Module Documentation
    Reference: `W3schools - Python Random Module <https://www.w3schools.
       com/python/module_random.asp>`_

"""
```

5.3.2. sendOrderDetailsToHalifaxTaxi

```
import json
import boto3

def handler(event, context):
    # Get the SQS queue URL
    queue_url = "https://sqs.us-east-1.amazonaws.com/000966082997/
       SNStoHalifaxTaxiQueue"

    """
        Boto3 - Amazon Amazon Simple Queue  Service (SQS) Documentation
        Reference: `Boto3 - Amazon SQS API <https://boto3.amazonaws.com/
           v1/documentation/api/latest/reference/services/sqs.html>`_
    """

    # Create an SQS client
    sqs = boto3.client('sqs')
```

```
    # Receive messages from the SQS queue (max 1 message)
    response = sqs.receive_message(
        QueueUrl=queue_url,
        MaxNumberOfMessages=1,
        VisibilityTimeout=30,
        WaitTimeSeconds=0  # Set to 0 for non-blocking receive
    )

    # Check if any messages were received
    if 'Messages' in response:
        # Extract the message
        message = response['Messages'][0]
        body = json.loads(message['Body'])
        order_details = body['Message']
        # print(order_details)

        order_details_json = json.loads(order_details)

        # Template for the email body
        email_body_template = """Car Type : {car_type}, Car Accessory :
            {car_accessory}, Client Address : {client_address}"""

        """
            W3schools - Python String format() Method Documentation
            Reference: `W3schools - Python String format() Method <https
                ://www.w3schools.com/python/ref_string_format.asp>`_
        """
        # Replace the placeholders in the email_body template
        email_body = email_body_template.format(
            car_type=order_details_json['data']['CAR_TYPE'],
            car_accessory=order_details_json['data']['CAR_ACCESSORY'],
            client_address=order_details_json['data']['CLIENT_ADDRESS']
        )

        # Publish the message to the SNS topic
        sns_topic_arn = "arn:aws:sns:us-east-1:000966082997:
            SendOrderDetailsToEmail"

        publish_to_sns(sns_topic_arn, email_body)

        # Delete the message from the SQS queue
        sqs.delete_message(
            QueueUrl=queue_url,
            ReceiptHandle=message['ReceiptHandle']
        )

    return {
```

```
        'statusCode': 200,
        'body': json.dumps('Message sent to subscriber')
    }

def publish_to_sns(topic_arn, message):
    # Create an SNS client
    sns = boto3.client('sns')

    """
        Boto3 - Amazon Amazon Simple Notification Service (SNS)
            Documentation
        Reference: `Boto3 - Amazon SNS API <https://boto3.amazonaws.com/
            v1/documentation/api/latest/reference/services/sns.html>`_
    """
    # Publish the message to the SNS topic
    sns.publish(
        TopicArn=topic_arn,
        Subject="Your order details",
        Message=json.dumps(message)
    )

"""
All references:
    Boto3 - Amazon Amazon Simple Notification Service (SNS)
        Documentation
    Reference: `Boto3 - Amazon SNS API <https://boto3.amazonaws.com/v1/
        documentation/api/latest/reference/services/sns.html>`_

    Boto3 - Amazon Amazon Simple Queue  Service (SQS) Documentation
    Reference: `Boto3 - Amazon SQS API <https://boto3.amazonaws.com/v1/
        documentation/api/latest/reference/services/sqs.html>`_

    W3schools - Python Random Module Documentation
    Reference: `W3schools - Python Random Module <https://www.w3schools.
        com/python/module_random.asp>`_

"""
```

5.4. Testing

I followed the below testing procedure

a) Triggered the 1st lambda **SendOrderDetailsToSNS** 2 times.

b) Checked it logs - 2 messageBody's were present.

c) Check if it published the messages to the SNS topic **SnsTopicToSqs**.

d) Polled the messages in SQS **SNStoHalifaxTaxiQueue**, and verified that I received the 2 messages.

e) Checked the logs of 2nd lambda **SendOrderDetailsToHalifaxTaxi**, and verified that messages are received periodically every 2 minutes based on the EventBridge rule.

f) Verified the receipt of mail in Gmail.

5.5. Screenshots

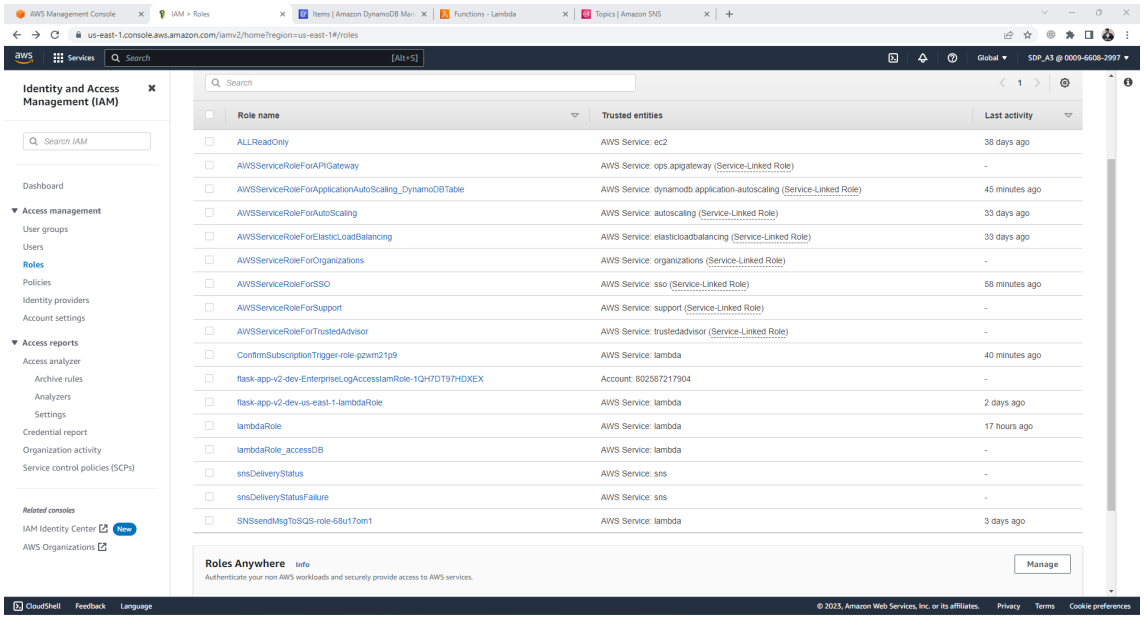

Figure 5.2.: ***Before IAM Role creation***

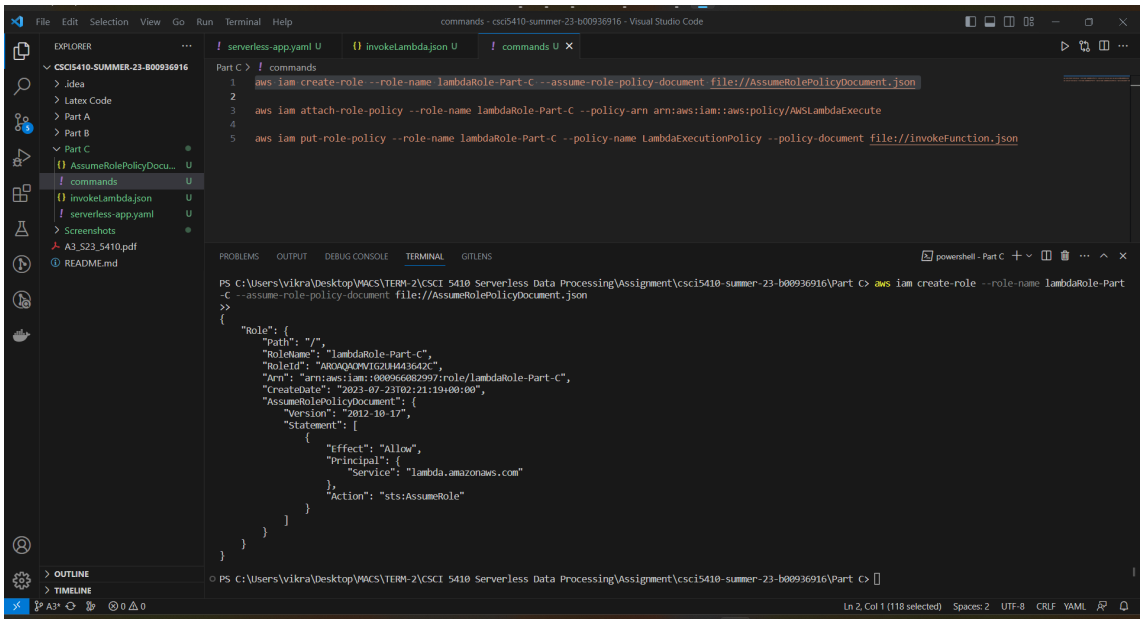

Figure 5.3.: ***Created IAM role lambdaRole-Part-C in CLI***

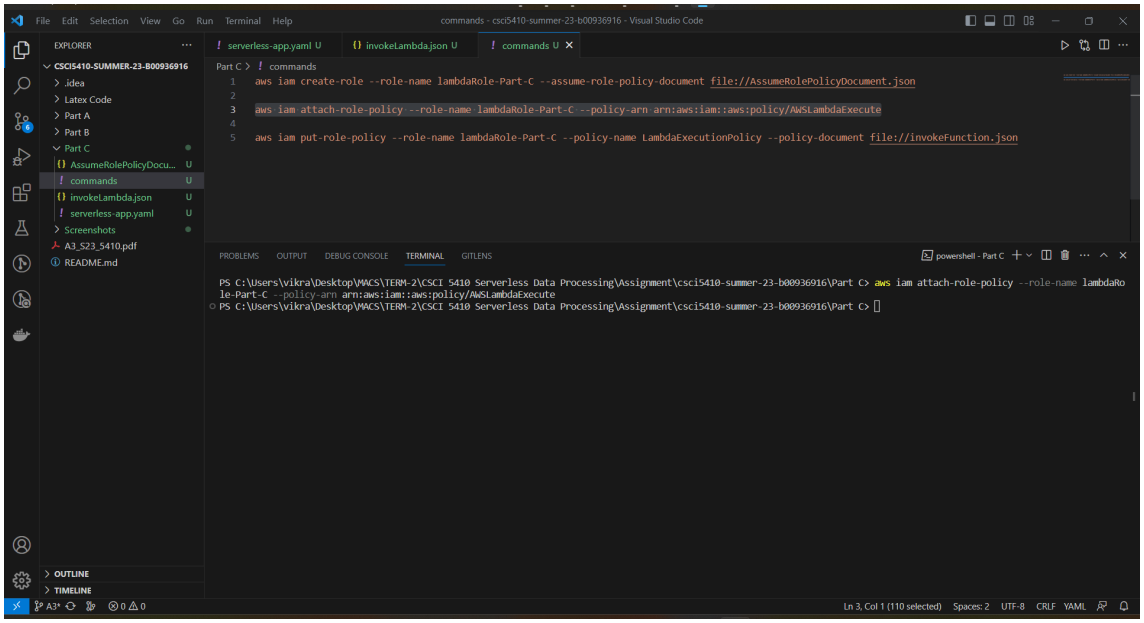

Figure 5.4.: ***Attach AWSLambdaExecute permission to role: lambdaRole-Part-C***

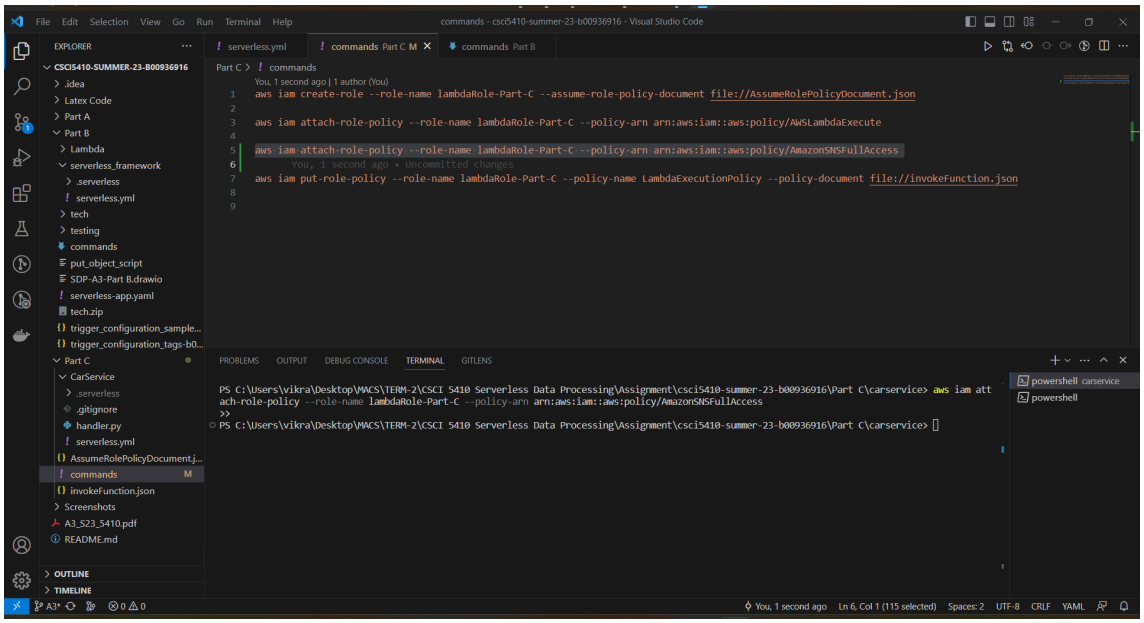

Figure 5.5.: ***Attach AmazonSNSFullAccess permission to role: lambdaRole-Part-C***

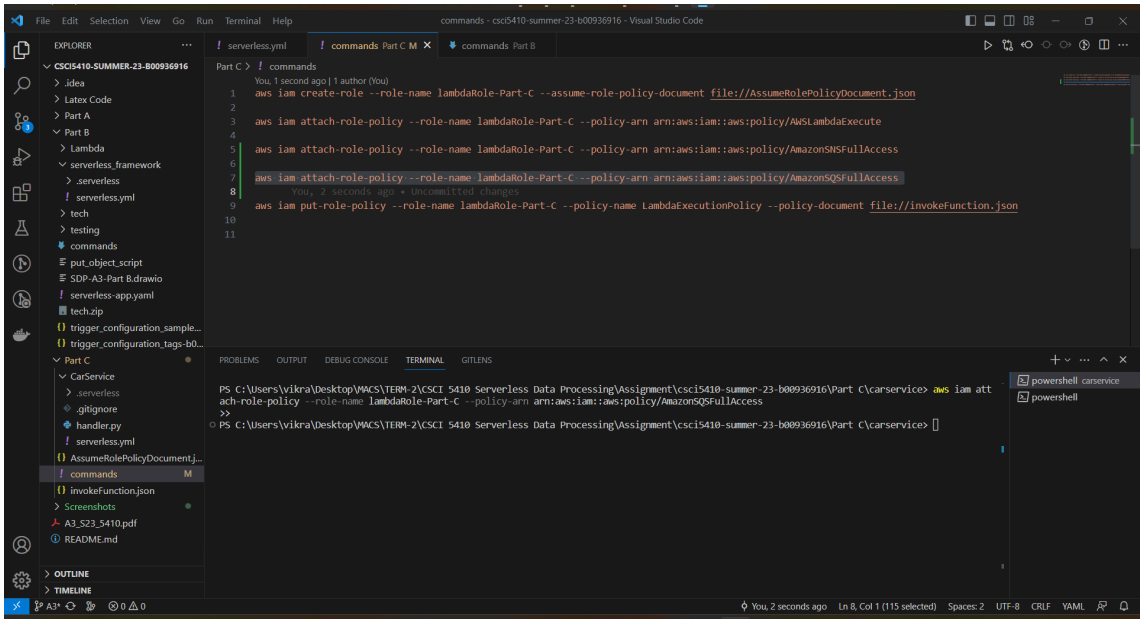

Figure 5.6.: ***Attach AmazonSQSFullAccess permission to role: lambdaRole-Part-C***

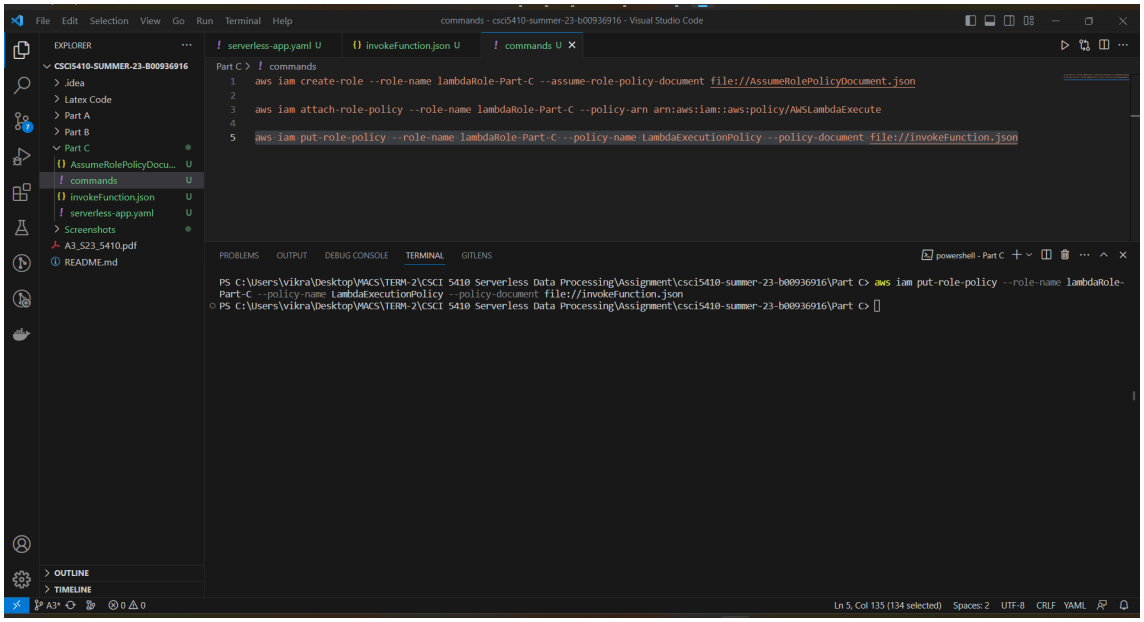

Figure 5.7.: ***Added invokeFunction permission to role: lambdaRole-Part-C***

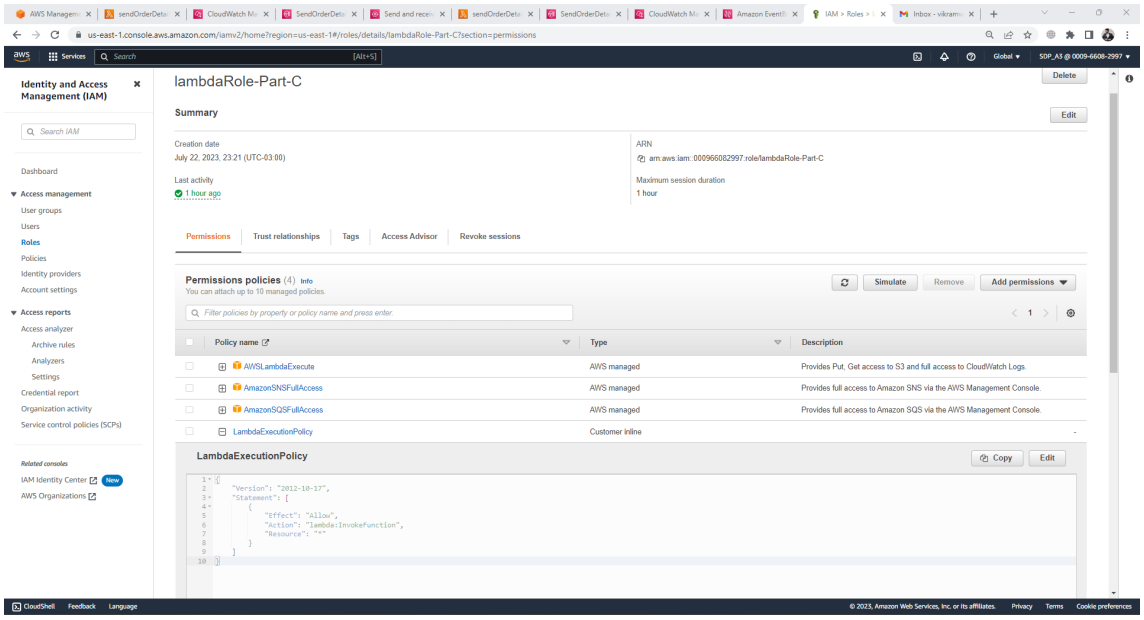

Figure 5.8.: ***Created IAM role lambdaRole-Part-C's permissions shown in console***

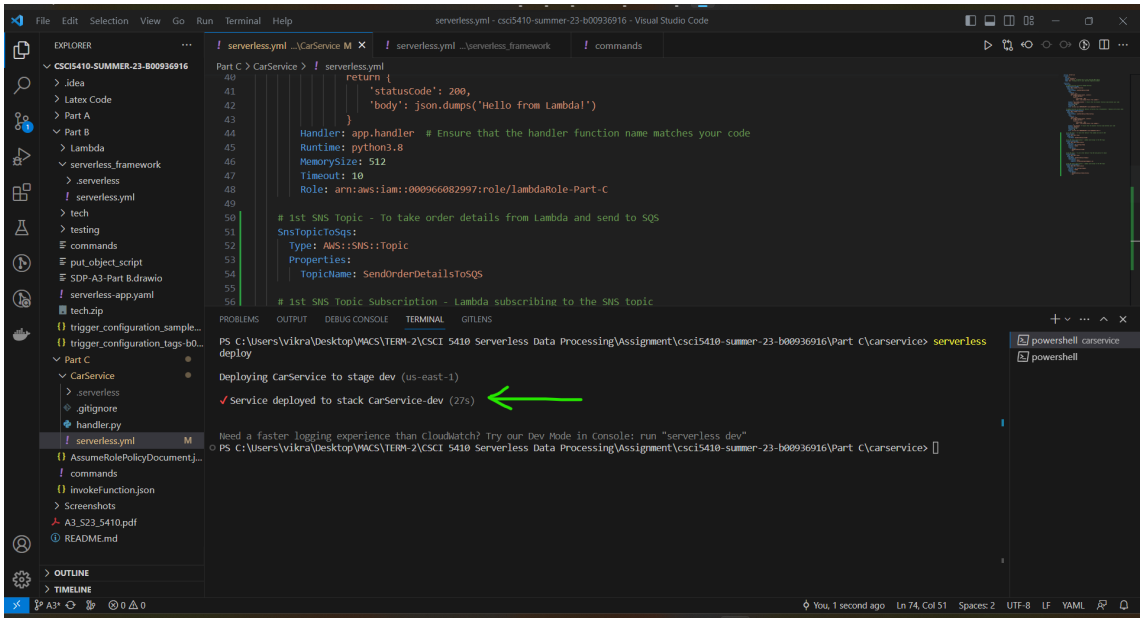

Figure 5.9.: ***serverless.yml file executed in CLI***

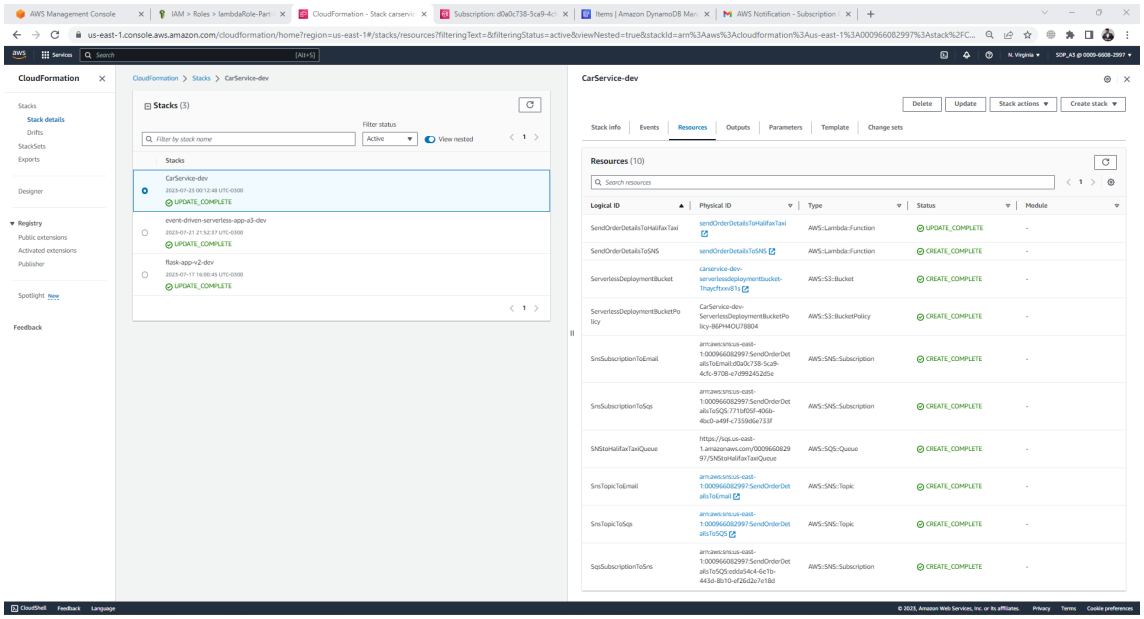

Figure 5.10.: ***Cloud formation stack created - shown in console***

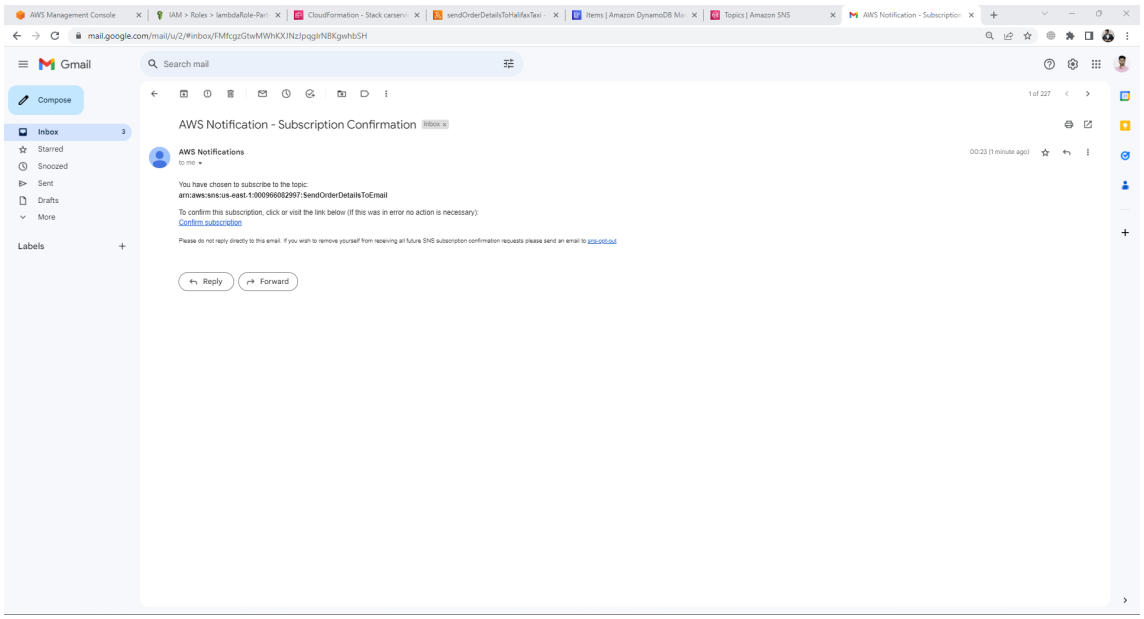

Figure 5.11.: ***Subscription mail received***

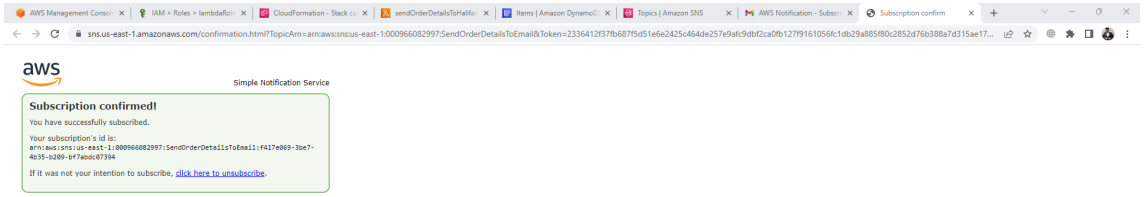

Figure 5.12.: ***Confirmed subscription***

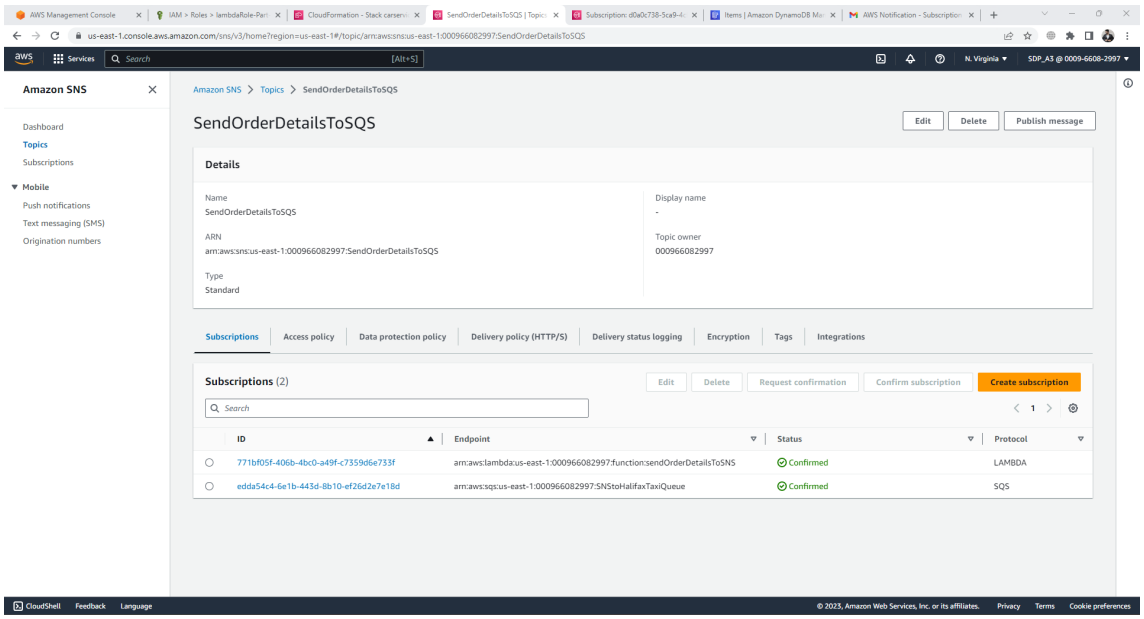

Figure 5.13.: ***Subscriptions of topic=SendOrderDetailsToSQS***

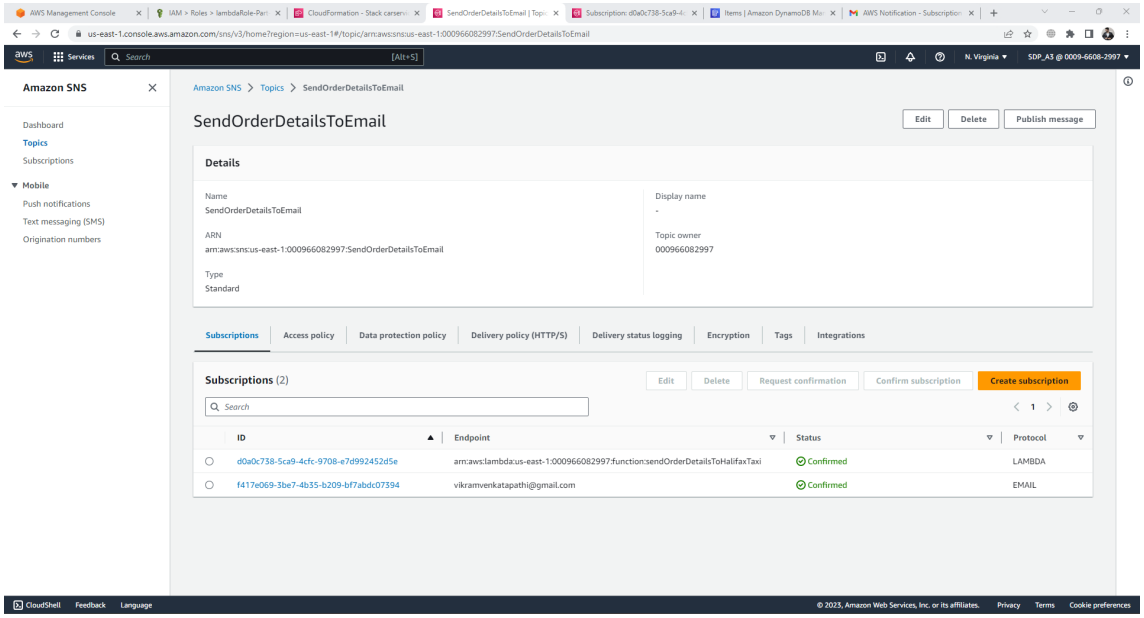

Figure 5.14.: ***Subscriptions of topic=SendOrderDetailsToEmail***

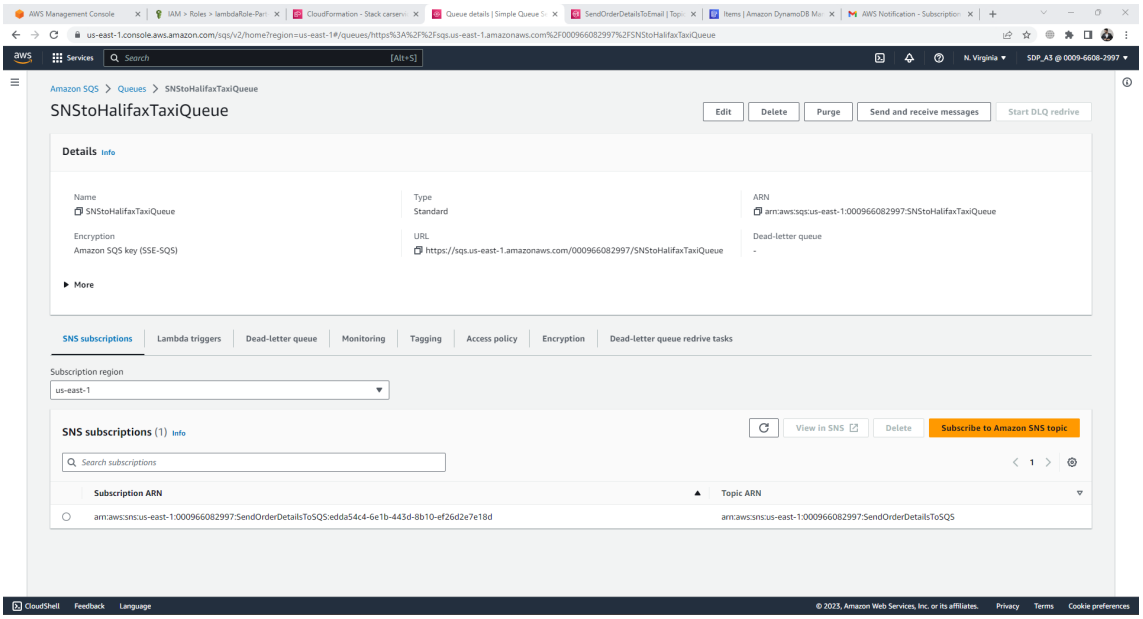

Figure 5.15.: ***SNStoHalifaxTaxiQueue - SNS subscription***

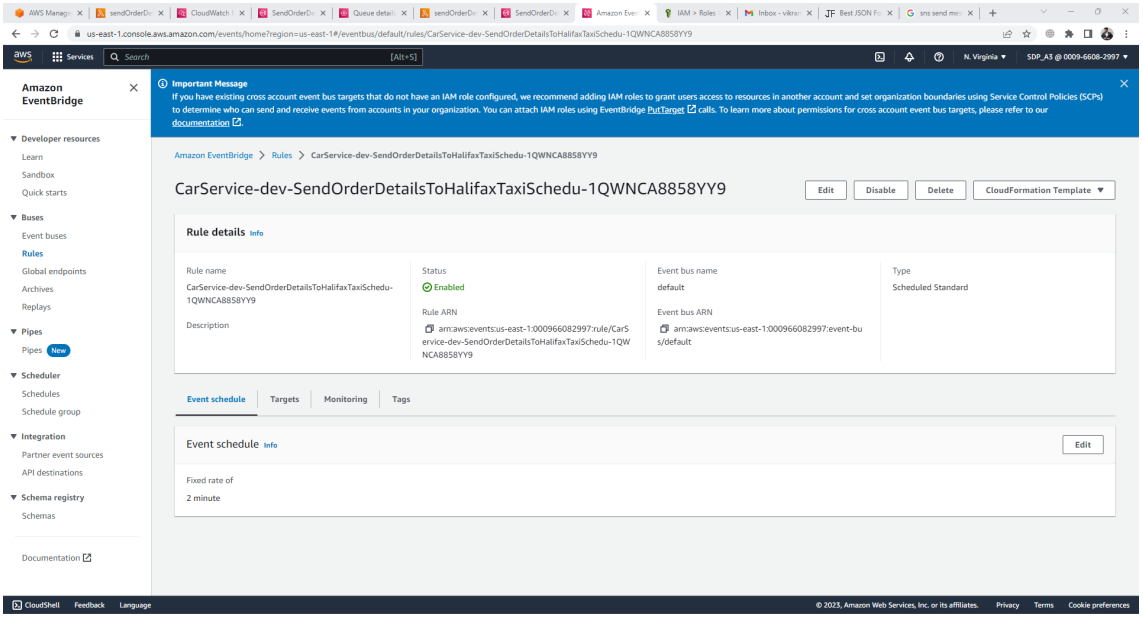

Figure 5.16.: ***Event rule "CarService-dev-SendOrderDetailsToHalifaxTaxiSchedu-1QWNCA8858YY9", created for Lambda: SendOrderDetailsToHalifaxTaxi***

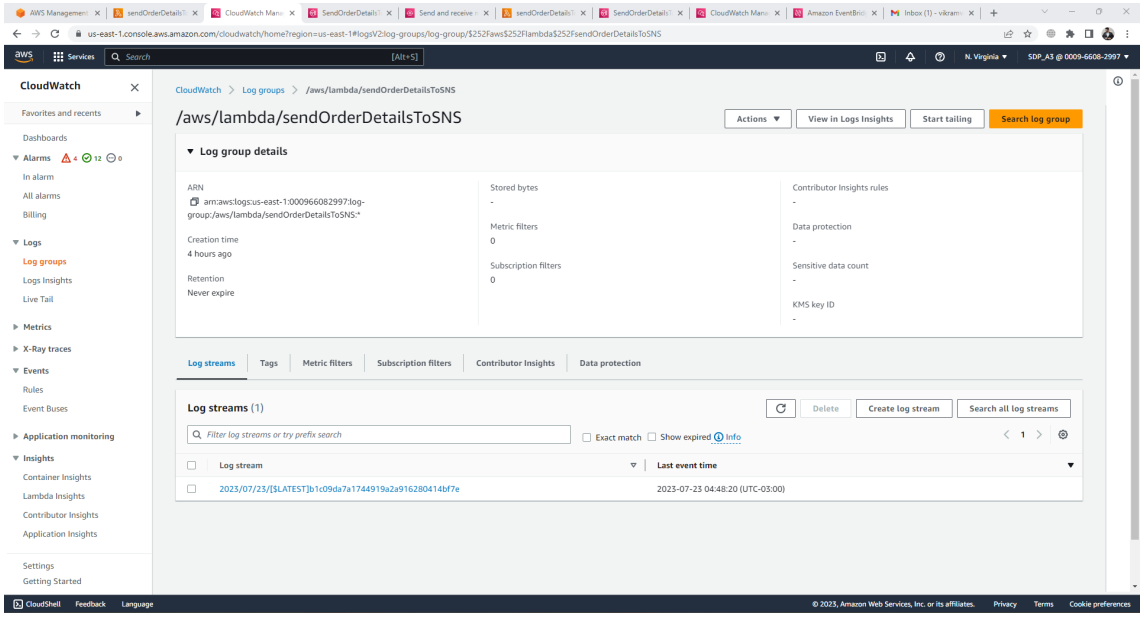

Figure 5.17.: ***sendOrderDetailsToSNS lambda - send message logs***

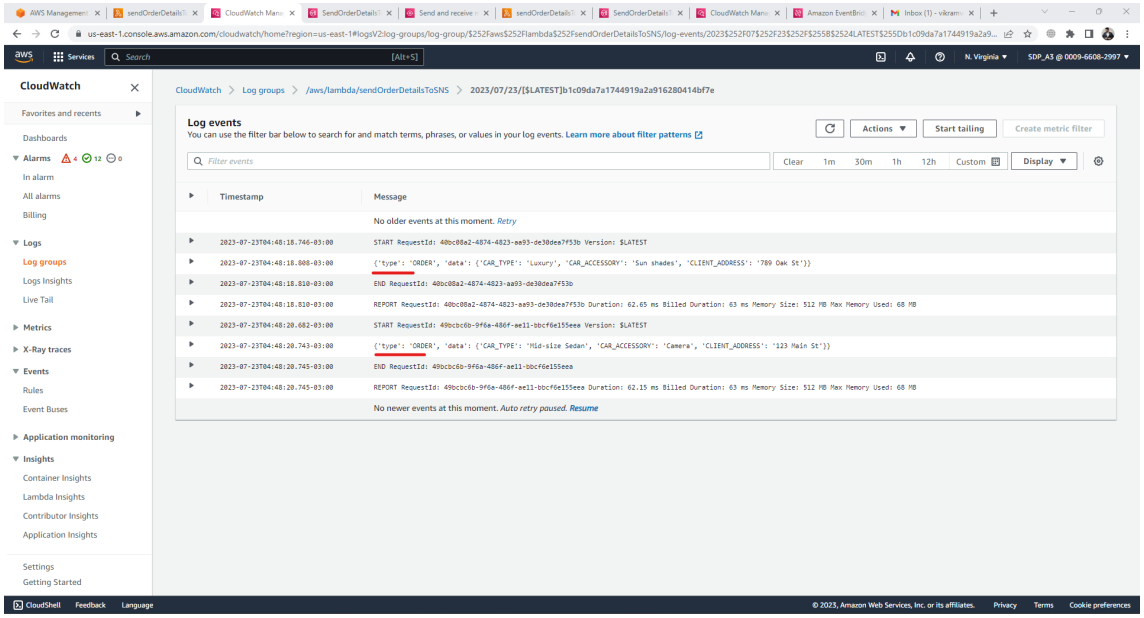

Figure 5.18.: ***sendOrderDetailsToSNS lambda - send message logs- 2 messages sent***

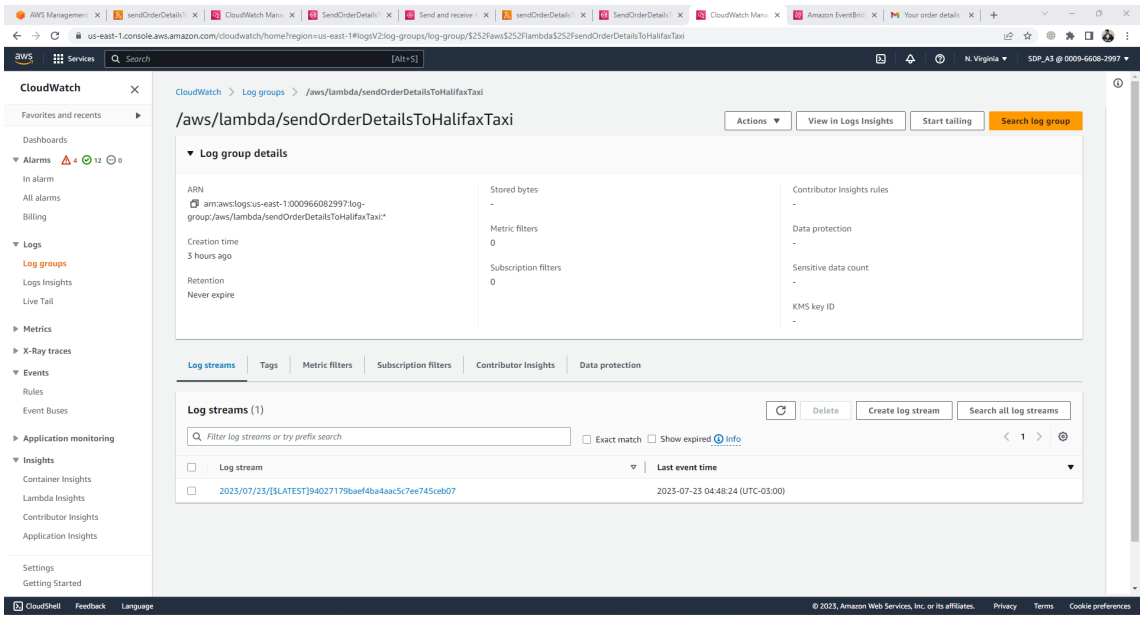

Figure 5.19.: ***sendOrderDetailsToHalifaxTaxi lambda - receive msgs logs***

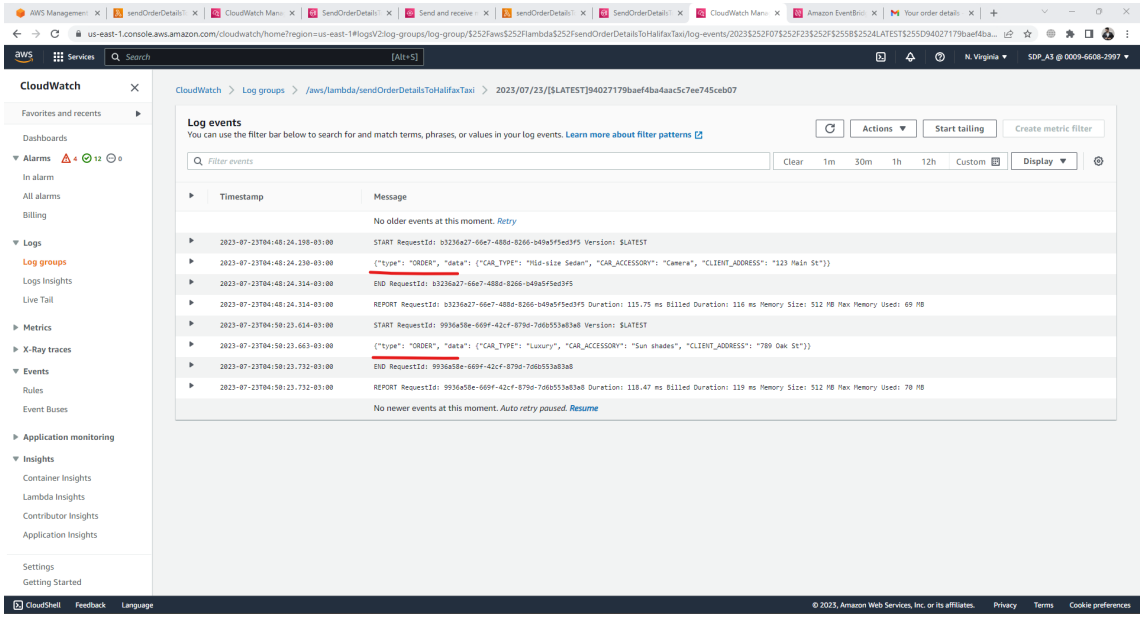

Figure 5.20.: ***sendOrderDetailsToHalifaxTaxi lambda - receive msgs logs - 2 msgs received***

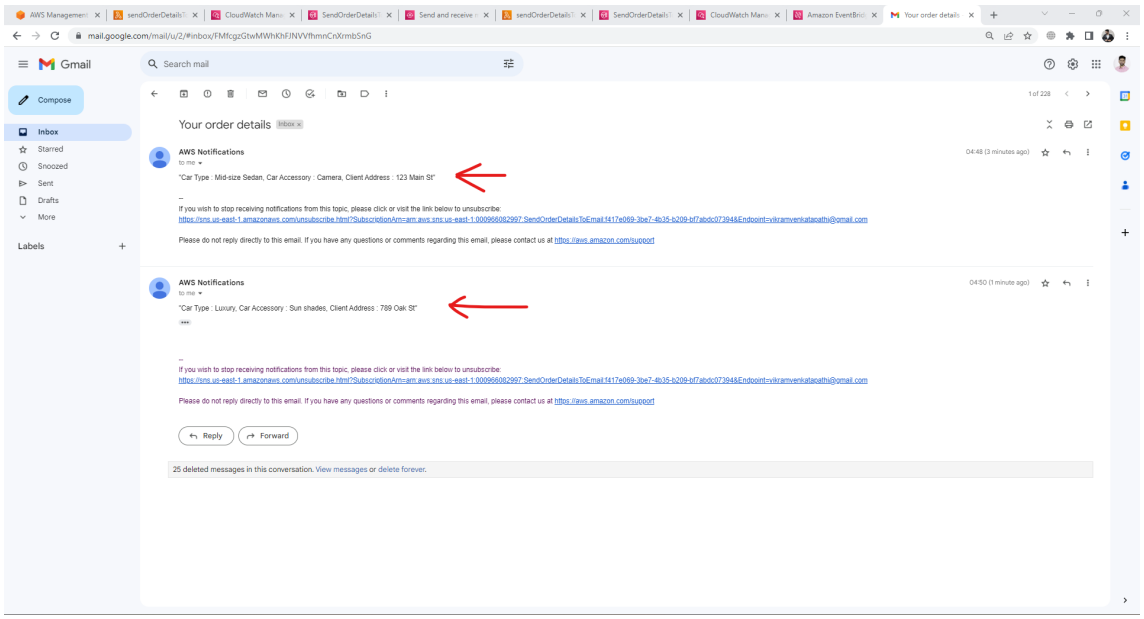

Figure 5.21.: *2 Emails with order details - received*

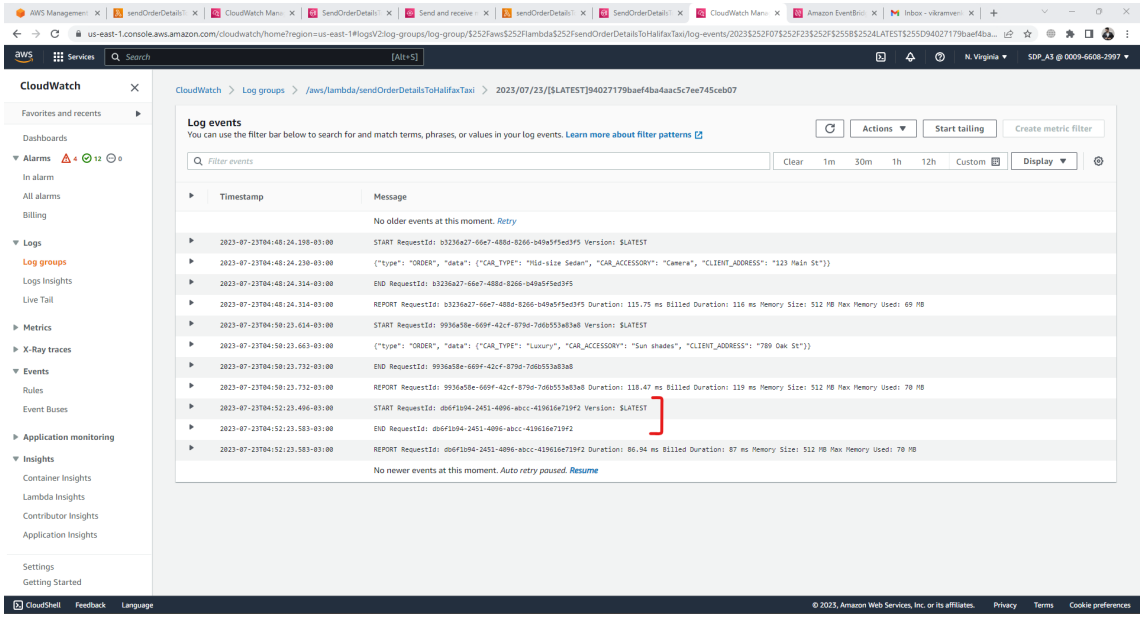

Figure 5.22.: ***No more msgs in the queue - so empty lambda invoke- didn't send any mail***

Part IV.

REFERENCES

References

[1] "Amazon Kinesis," *Amazon Web Services*, Inc., 2019. Available: https://aws.amazon.com/kinesis.[Accessed: 20 July 2023].

[2] "Atlantic Superstore supermarket: Grocery shop online or instore," *Atlantic Superstore Supermarket — Grocery shop online or instore*, [Online]. Available: https://www.atlanticsuperstore.ca/ [Accessed: July 22, 2023].

[3] "Zero-Friction Serverless Apps on AWS," *Serverless*, [Online]. Available: https://www.serverless.com/ [Accessed: July 22, 2023].

[4] "Develop & Monitor apps on AWS Lambda," *Serverless*, [Online]. Available: https://www.serverless.com/ [Accessed: July 22, 2023].

[5] Abhay Singh, "AWS Lambda S3 trigger using AWS CLI: A step-by-step guide," *Abhay Singh*, [Online]. Available: https://abhayksingh.com/aws-lambda-s3-trigger-configuration-using-aws-cli-a-step-by-step-guide/ [Accessed: July 22, 2023].

[6] "AWS CLI Command Reference - Load AWS CLI parameters from a file," *Amazon*, [Online]. Available: https://docs.aws.amazon.com/cli/latest/userguide/cli-usage-parameters-file.html [Accessed: July 22, 2023].

[7] D. Musgrave, "Lambda," *AWS*, [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html [Accessed: July 22, 2023].

[8] "DynamoDB#," *DynamoDB - Boto3 1.28.9 documentation*, [Online]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/dynamodb.html [Accessed: July 22, 2023].

[9] "RE - regular expression operations," *Python documentation*, [Online]. Available: https://docs.python.org/3/library/re.html [Accessed: July 22, 2023].

[10] "Amazon S3 REST API introduction - LambdaFunctionConfiguration," *AWS*, [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html [Accessed: July 22, 2023].

[11] "AWS CLI Command Reference - put-bucket-notification-configuration," *AWS*, [Online]. Available: https://docs.aws.amazon.com/cli/latest/reference/s3api/put-bucket-notification-configuration.html [Accessed: July 22, 2023].

[12] "How to use bash for loop in one line - nixcraft," *Nixcraft*, [Online]. Available: https://www.cyberciti.biz/faq/linux-unix-bash-for-loop-one-line-command/ [Accessed: July 22, 2023].

[13] "S3#," *S3 - Boto3 1.28.9 documentation*, [Online]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html [Accessed: July 22, 2023].

[14] "AWS IoT Core," *AWS* ,[Online]. Available: https://aws.amazon.com/iot-core/ [Accessed: July 22, 2023].

[15] "Amazon Kinesis Data Streams", *AWS* ,[Online]. Available: https://aws.amazon.com/kinesis/data-streams/ [Accessed: July 22, 2023].

[16] "Amazon Kinesis Data Analytics", *AWS* ,[Online]. Available: https://aws.amazon.com/kinesis/data-analytics/ [Accessed: July 22, 2023].

[17] "AWS Lambda" *AWS* ,[Online]. Available: https://aws.amazon.com/lambda/ [Accessed: July 22, 2023].

[18] "Amazon Comprehend", *AWS* ,[Online]. Available: https://aws.amazon.com/comprehend/ [Accessed: July 22, 2023].

[19] "Amazon DynamoDB", *AWS* ,[Online]. Available: https://aws.amazon.com/dynamodb/ [Accessed: July 22, 2023].

[20] "Amazon QuickSight", *AWS* ,[Online]. Available: https://aws.amazon.com/quicksight/ [Accessed: July 22, 2023].

[21] "AWS creating IAM roles - *javatpoint*," javatpoint, [Online]. Available: https://www.javatpoint.com/aws-creating-iam-roles [Accessed: July 22, 2023].

[22] D. Greene and P. Cunningham, "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering," *Proceedings of the 23rd International Conference on Machine Learning (ICML '06),* Pittsburgh, Pennsylvania, USA, 2006, pp. 377-384. doi: 10.1145/1143844.1143892

[23] P. Melly, "AWS Architecture and Draw.io, a match made in developer heaven," draw.io, https://drawio-app.com/blog/aws-architecture-and-draw-io-a-match-made-in-developer-heaven/ (Accessed July 24, 2023).

[24] "AWS Command Line Reference - create-role," Amazon, [Online]. Available: https://docs.aws.amazon.com/cli/latest/reference/iam/create-role.html (Accessed: July 24, 2023).

[25] "AWS Command Line Reference - attach-role-policy," Amazon, [Online]. Available: https://docs.aws.amazon.com/cli/latest/reference/iam/attach-role-policy.html (Accessed: July 24, 2023).