



CSCI 5410

Serverless Data Processing (Summer 2023)

MASTER OF APPLIED COMPUTER
SCIENCE

Assignment-3 (Part C):

Name: **Jainil Sevalia** | Banner Id : **B00925445** |
Email: jn498899@dal.ca

Table of figures:

Figure 1 : Overall Architecture of Halifax Taxi	3
Figure 2 : HalifaxTaxiQueue created (AWS SQS).	3
Figure 3 : Queue created and got Queue URL.....	4
Figure 4 : Creating Publish Order SNS topic.....	5
Figure 5: Successfully created PublishOrder SNS topic.....	5
Figure 6 : SQS subscribed to SNS topic.	6
Figure 7 : Create ECR Repository for Lambda Docker Image.....	7
Figure 8 : Successfully created ECR Repository.	7
Figure 9 : PublishOrder lambda function Code.	8
Figure 10 : Docker Image of Lambda code build, tagged and Pushed to ECR.	9
Figure 11 : ECR repository AWS console.....	9
Figure 12 : Creating Lambda function using docker image pushed in ECR.	10
Figure 13 : Successfully created Lambda Function.	10
Figure 14 : Send Mail SNS topic.	11
Figure 15 : Successfully created SNS topic. (AWS Console).....	12
Figure 16 : Subscribe to email in SendMail SNS topic.	13
Figure 17 : Subscription list of SNS topic.	13
Figure 18 : ECR Repository for second Lambda Docker Image.	14
Figure 19 : Code for (poll_msg_send_mail) lambda function.....	15
Figure 20 : Docker image for second Lambda is build, tagged, and pushed to ECR repository.	16
Figure 21 : Successfully Second lambda docker image is pushed. (AWS console).	16
Figure 22 : Creating Second Lambda function poll_msg_send_mail. (AWS console).	17
Figure 23 : Configure AWS Event bridge for invoking lambda every 2 min.	18
Figure 24 : Second lambda(poll_msg_send_mail) AWS console.	18
Figure 25: Testing - Triggering first Lambda function to generate order and send to SNS to SQS.	19
Figure 26: Message that sent to SNS topic. (Cloud watch AWS console).....	19
Figure 27 : Got email of with Taxi order Information.	20
Figure 28 : Taxi Order Information got in mail.	20

Halifax Taxi:

Architecture:

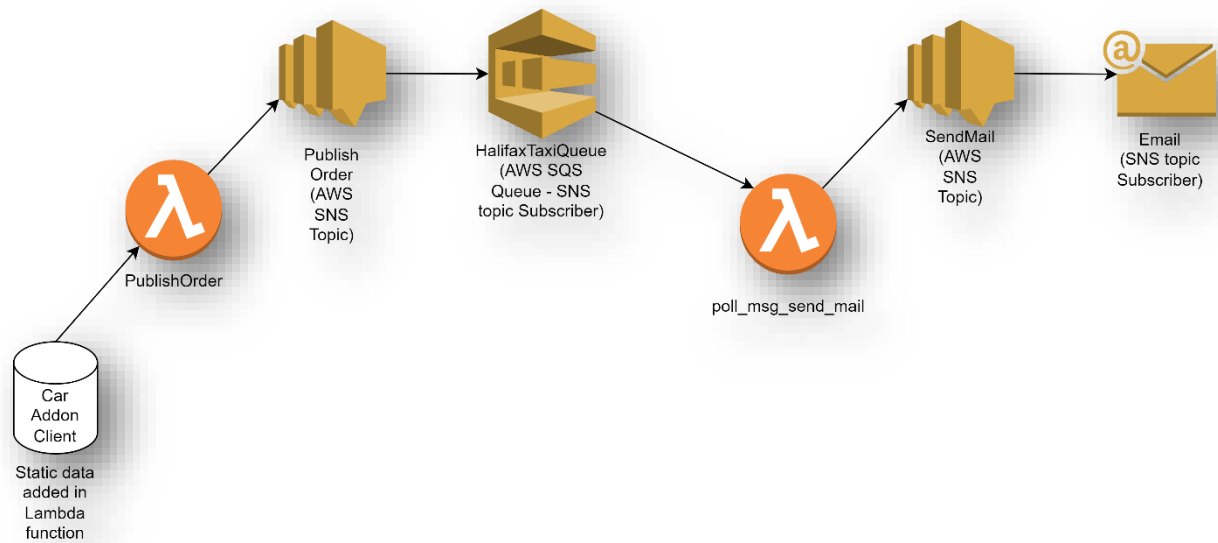


Figure 1 : Overall Architecture of Halifax Taxi

- Created a HalifaxTaxiQueue using AWS SQS service. This Queue will store all messages(Orders Information).

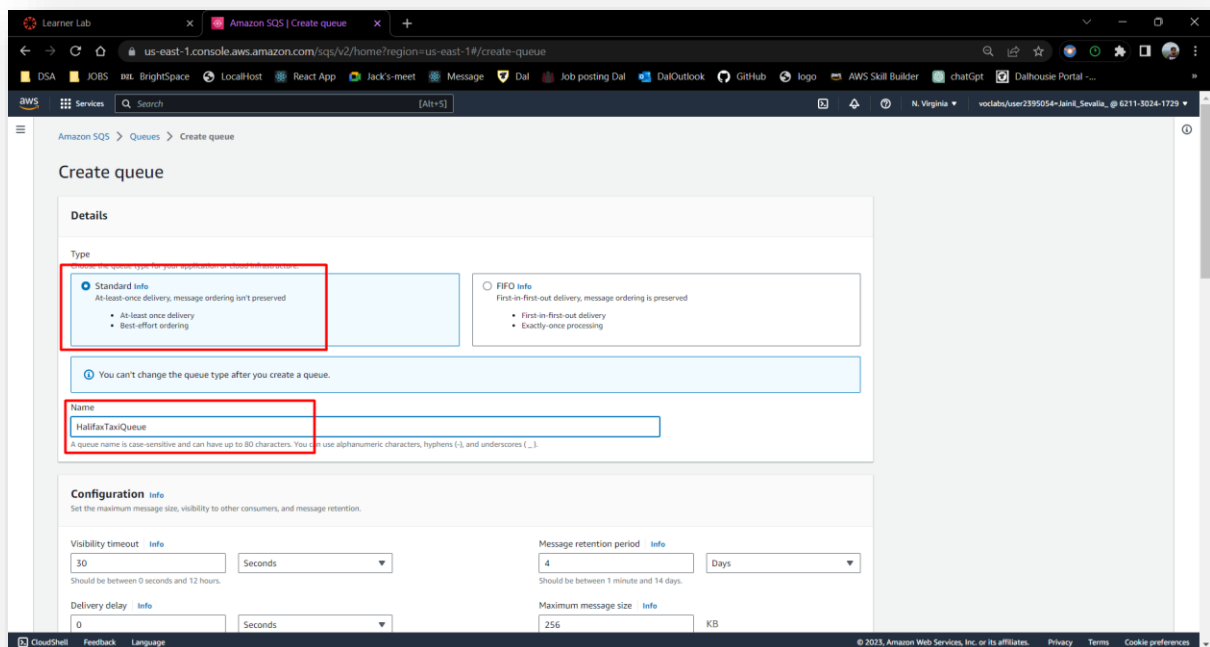


Figure 2 : HalifaxTaxiQueue created (AWS SQS).

- Queue Created successfully.

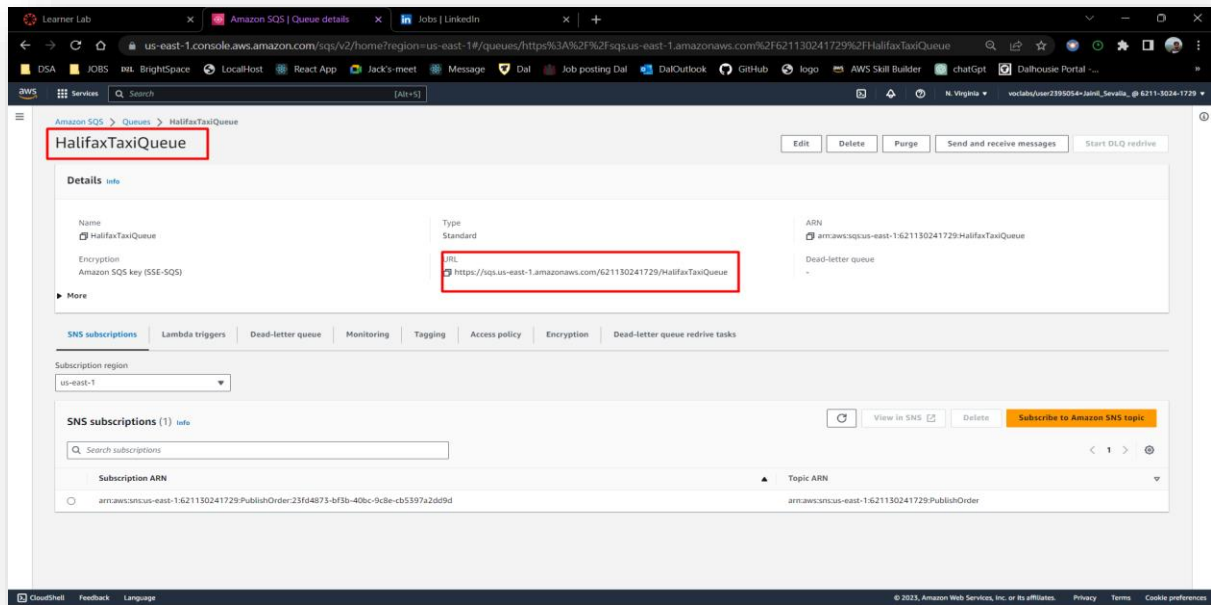


Figure 3 : Queue created and got Queue URL.

- Create AWS SNS topic. First AWS Lambda(Which randomly generate orders) will send order message to this SNS topic.

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

PublishOrder

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

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

My Topic

Maximum 100 characters.

► **Encryption - optional**
Amazon SNS provides in-transit encryption by default. Enabling server-side encryption adds at-rest encryption to your topic.

► **Access policy - optional** [Info](#)
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

Figure 4 : Creating Publish Order SNS topic.

- Successfully created PublishOrder SNS topic.

Amazon SNS > Topics > PublishOrder

PublishOrder

Details

Name
PublishOrder

ARN
arn:aws:sns:us-east-1:621130241729:PublishOrder

Type
Standard

Display name
-

Topic owner
621130241729

Edit Delete Publish message

Figure 5: Successfully created PublishOrder SNS topic.

- SQS Queue is Subscribed to SNS topic. So, message will publish to SQS.

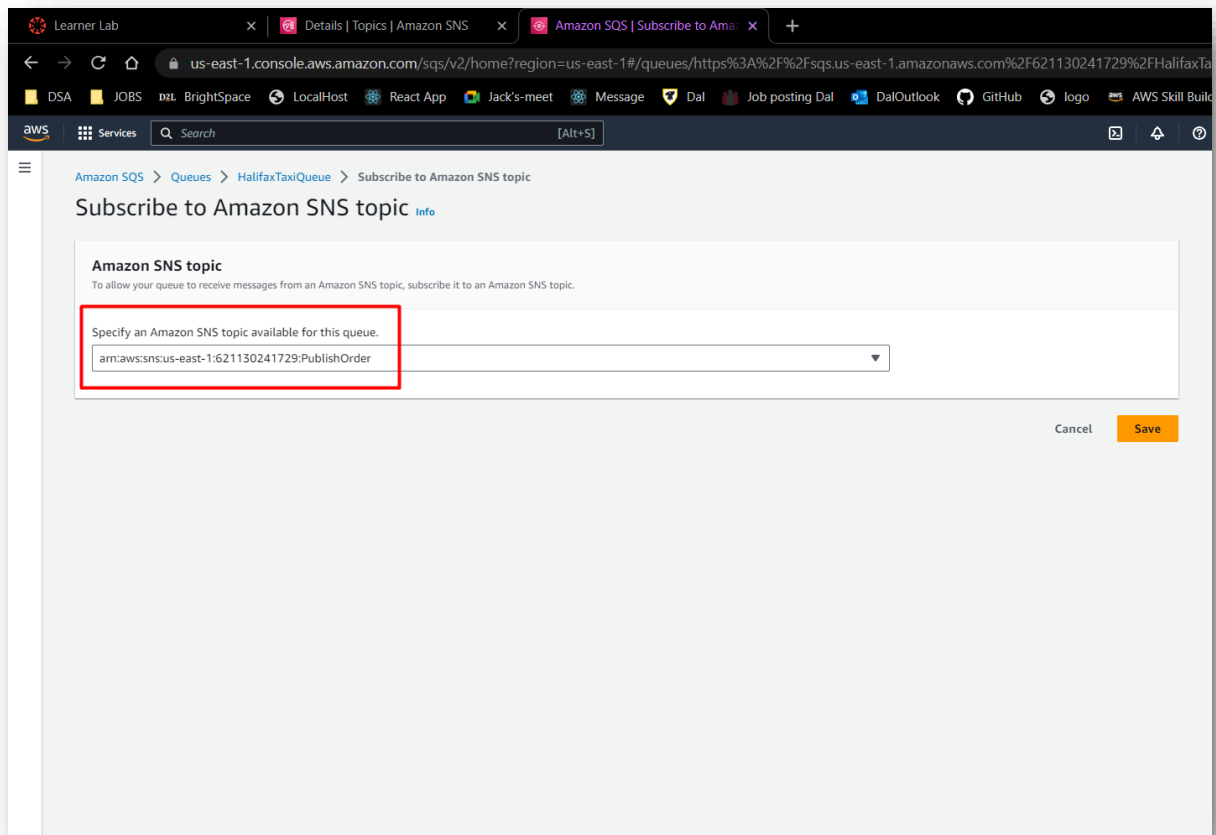


Figure 6 : SQS subscribed to SNS topic.

- Create ECR Repository for Lambda Docker Image.

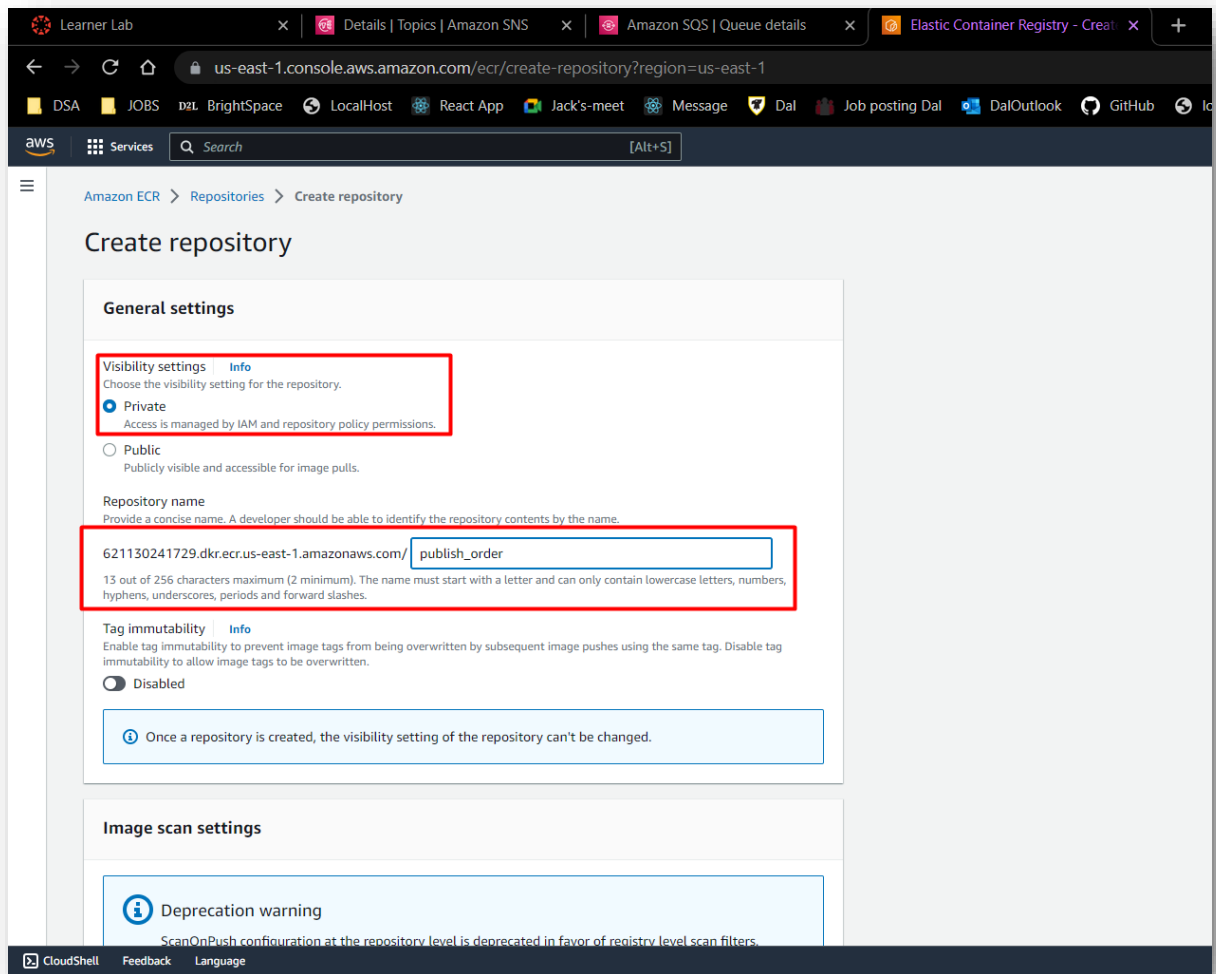


Figure 7 : Create ECR Repository for Lambda Docker Image.

- Created ECR Repository for Docker Image.

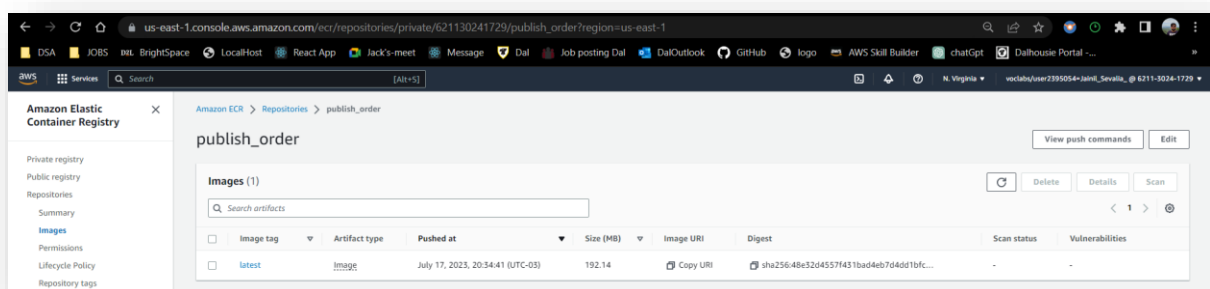
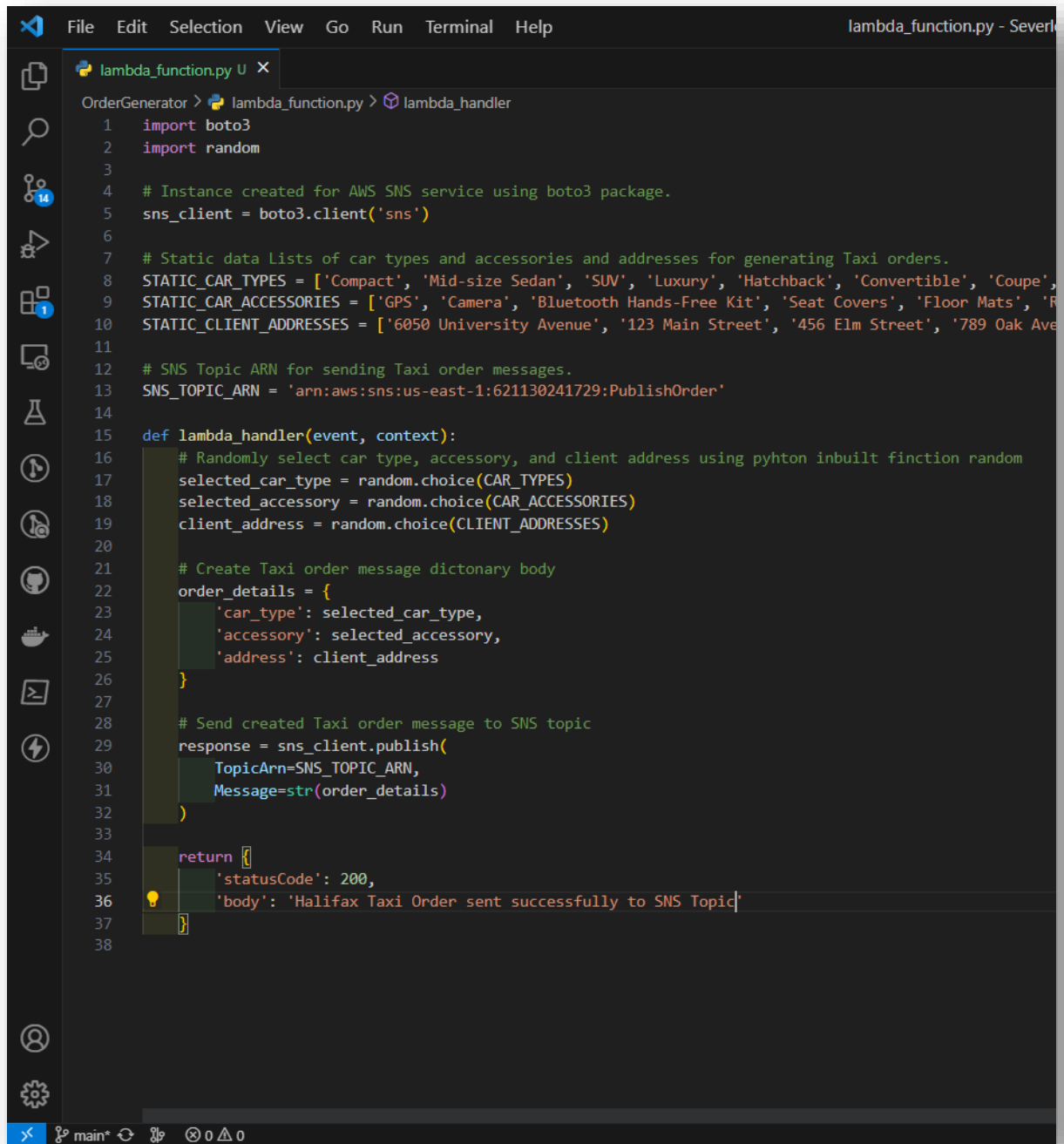


Figure 8 : Successfully created ECR Repository.

- Lambda function code is mentioned below, which will generate Taxi Order message and send it to SNS Topic(PublishOrder).



```
File Edit Selection View Go Run Terminal Help
lambda_function.py - Severl

lambda_function.py U X
OrderGenerator > lambda_function.py > lambda_handler
1 import boto3
2 import random
3
4 # Instance created for AWS SNS service using boto3 package.
5 sns_client = boto3.client('sns')
6
7 # Static data Lists of car types and accessories and addresses for generating Taxi orders.
8 STATIC_CAR_TYPES = ['Compact', 'Mid-size Sedan', 'SUV', 'Luxury', 'Hatchback', 'Convertible', 'Coupe',
9 STATIC_CAR_ACCESSORIES = ['GPS', 'Camera', 'Bluetooth Hands-Free Kit', 'Seat Covers', 'Floor Mats', 'R
10 STATIC_CLIENT_ADDRESSES = ['6050 University Avenue', '123 Main Street', '456 Elm Street', '789 Oak Ave
11
12 # SNS Topic ARN for sending Taxi order messages.
13 SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:621130241729:PublishOrder'
14
15 def lambda_handler(event, context):
16     # Randomly select car type, accessory, and client address using python inbuilt function random
17     selected_car_type = random.choice(CAR_TYPES)
18     selected_accessory = random.choice(CAR_ACCESSORIES)
19     client_address = random.choice(CLIENT_ADDRESSES)
20
21     # Create Taxi order message dictionary body
22     order_details = {
23         'car_type': selected_car_type,
24         'accessory': selected_accessory,
25         'address': client_address
26     }
27
28     # Send created Taxi order message to SNS topic
29     response = sns_client.publish(
30         TopicArn=SNS_TOPIC_ARN,
31         Message=str(order_details)
32     )
33
34     return {
35         'statusCode': 200,
36         'body': 'Halifax Taxi Order sent successfully to SNS Topic'
37     }
38
```

Figure 9 : PublishOrder lambda function Code.

- Docker image build, tagged and pushed to ECR newly created repository using custom deploy.sh script.

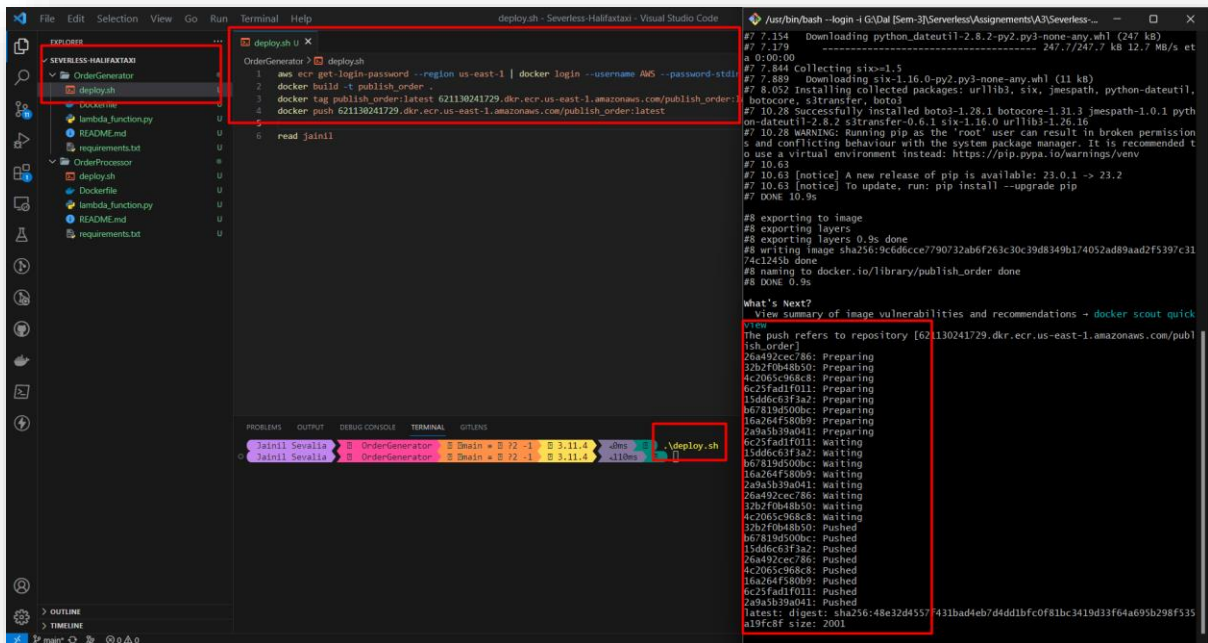


Figure 10 : Docker Image of Lambda code build, tagged and Pushed to ECR.

- Docker image is visible in the ECR repository in AWS Console.

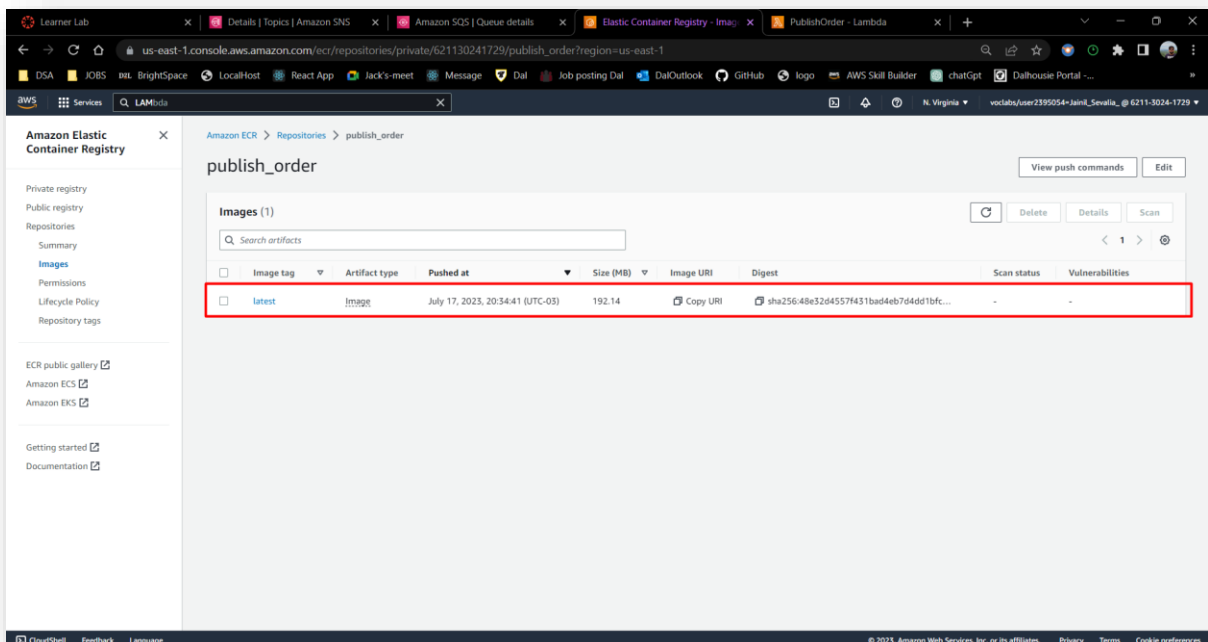


Figure 11 : ECR repository AWS console.

- Created AWS Lambda function using newly pushed docker image on ECR publish_order Repository.

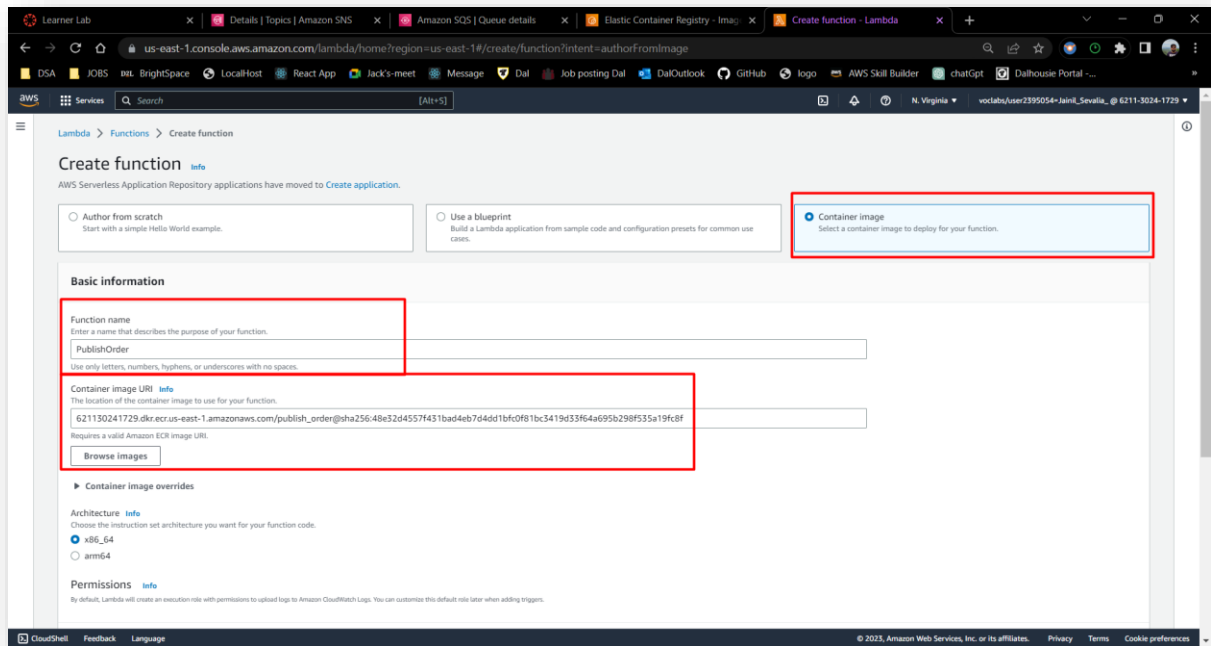


Figure 12 : Creating Lambda function using docker image pushed in ECR.

- Successfully created Lambda function.

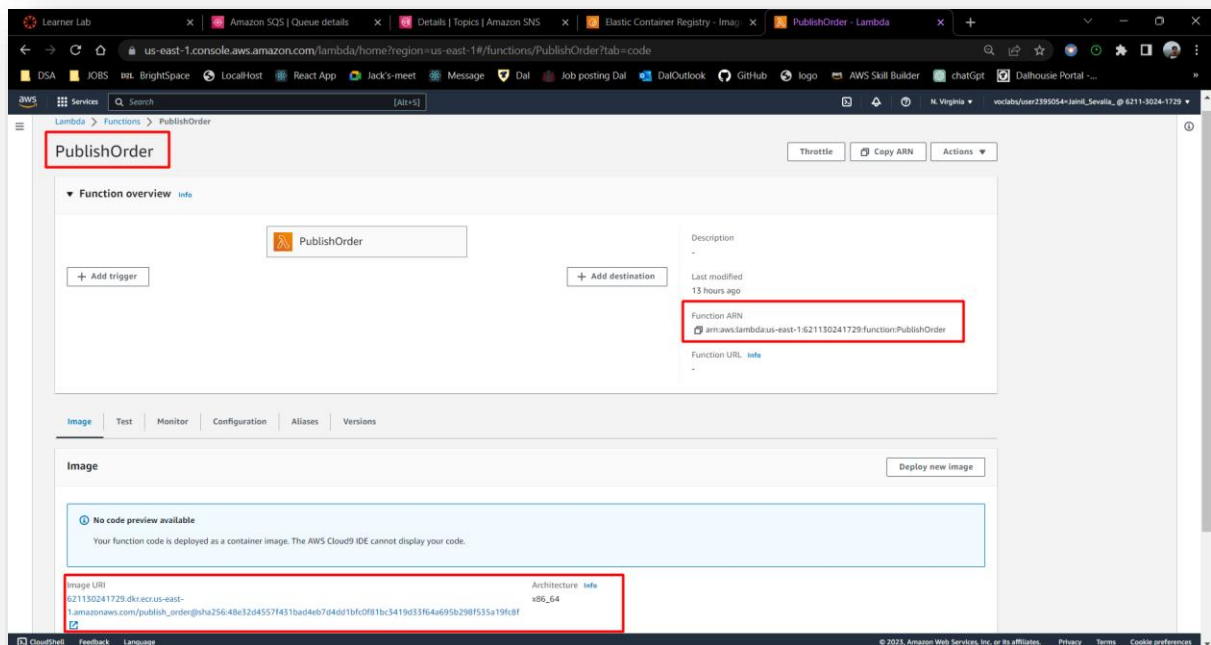


Figure 13 : Successfully created Lambda Function.

- Create a new SNS topic for sending mail. Second Lambda will poll message from the SQS and send it to this SNS topic.

The screenshot shows the AWS Management Console interface for creating a new SNS topic. The browser tabs include 'Learner Lab', 'Create topic | Amazon SNS', 'Amazon SQS | Queue details', and 'Elastic Container Registry - Image'. The URL is 'us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/create-topic'. The page title is 'Create topic'. The 'Details' section shows two topic types: 'FIFO (first-in, first-out)' and 'Standard'. The 'Standard' type is selected and highlighted with a red box. Below the types, the 'Name' field is filled with 'SendMail' and is also highlighted with a red box. The 'Display name - optional' field is filled with 'My Topic'. The 'Encryption - optional' and 'Access policy - optional' sections are visible at the bottom.

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

SendMail

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

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

My Topic

Maximum 100 characters.

► **Encryption - optional**
Amazon SNS provides in-transit encryption by default. Enabling server-side encryption adds at-rest encryption to your topic.

► **Access policy - optional** [Info](#)
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

Figure 14 : Send Mail SNS topic.

- Successfully SNS topic created for sending mail for car delivery.

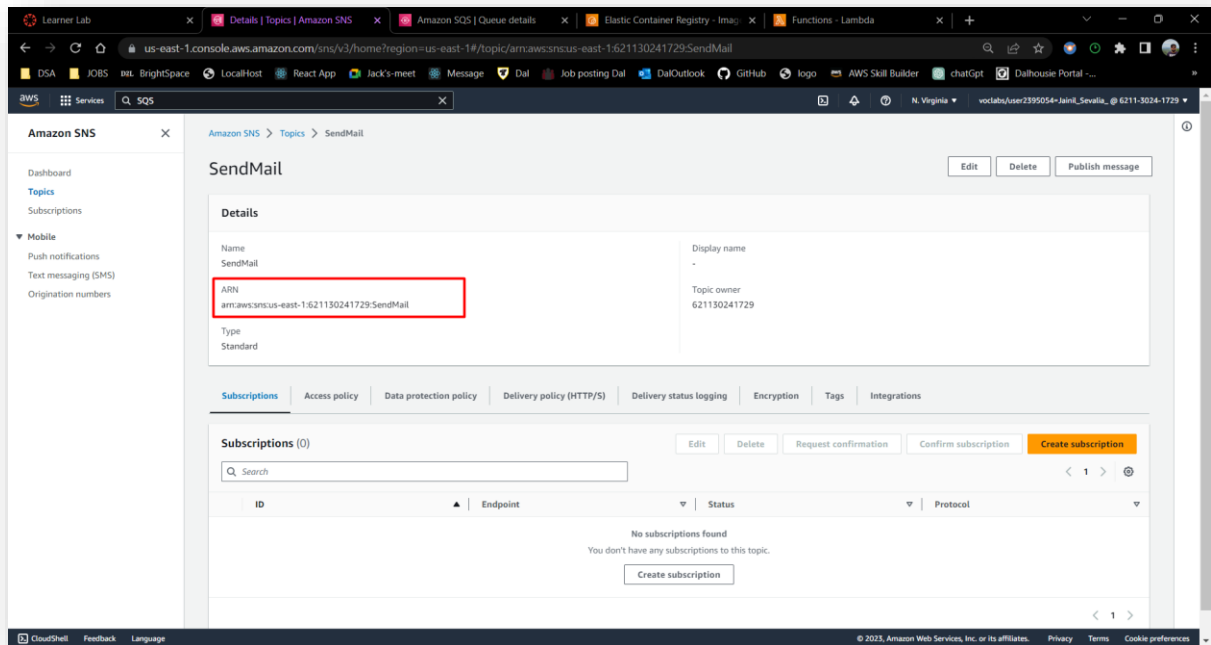


Figure 15 : Successfully created SNS topic. (AWS Console).

- Subscribe to email using SNS Subscription functionality.

The screenshot shows the AWS Management Console 'Create subscription' page for an SNS topic. The browser tabs include 'Learner Lab', 'Create subscription | Amazon SNS', 'Amazon SQS | Queue details', 'Elastic Container Registry', and 'PublishOrder - Lambda'. The URL is 'us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/create-subscription'. A warning banner at the top mentions 'Important changes for sending text messages (SMS) to US destinations'. The page title is 'Create subscription'. The 'Details' section is highlighted with a red box and contains the following fields:

- Topic ARN:** A text input field containing 'arn:aws:sns:us-east-1:621130241729:SendMail'.
- Protocol:** A dropdown menu with 'Email' selected.
- Endpoint:** A text input field containing 'jainil.sevalia@dal.ca'.

Below the 'Details' section, there are two optional policy sections:

- Subscription filter policy - optional:** This policy filters the messages that a subscriber receives.
- Redrive policy (dead-letter queue) - optional:** Send undeliverable messages to a dead-letter queue.

At the bottom right, there are 'Cancel' and 'Create subscription' buttons. A blue information box states: 'After your subscription is created, you must confirm it. Info'.

Figure 16 : Subscribe to email in SendMail SNS topic.

The screenshot shows the AWS Management Console 'Subscriptions' page for an SNS topic. The page title is 'Subscriptions (2)'. There is a search bar and a table of subscriptions. The table has columns for ID, Endpoint, Status, and Protocol. Two subscriptions are listed, both with a status of 'Confirmed' and a protocol of 'EMAIL'. The 'Endpoint' column is highlighted with a red box.

ID	Endpoint	Status	Protocol
2b4dc357-a9c3-43a0-95e6-018bc9ba3caa	jainilsevalia@gmail.com	Confirmed	EMAIL
9b49d03c-765b-48ff-bc08-9b0d0b0b0b0b	jainil.sevalia@dal.ca	Confirmed	EMAIL

Figure 17 : Subscription list of SNS topic.

- Create ECR repository for second lambda function. This function will poll the message from SQS. This function will trigger every 2 min.

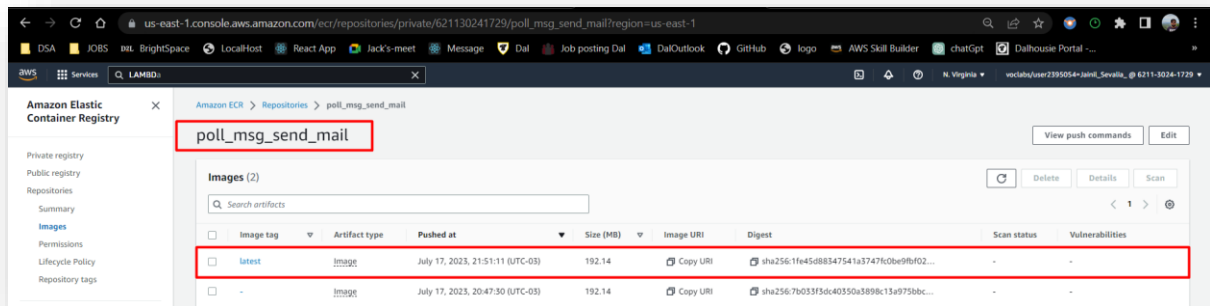
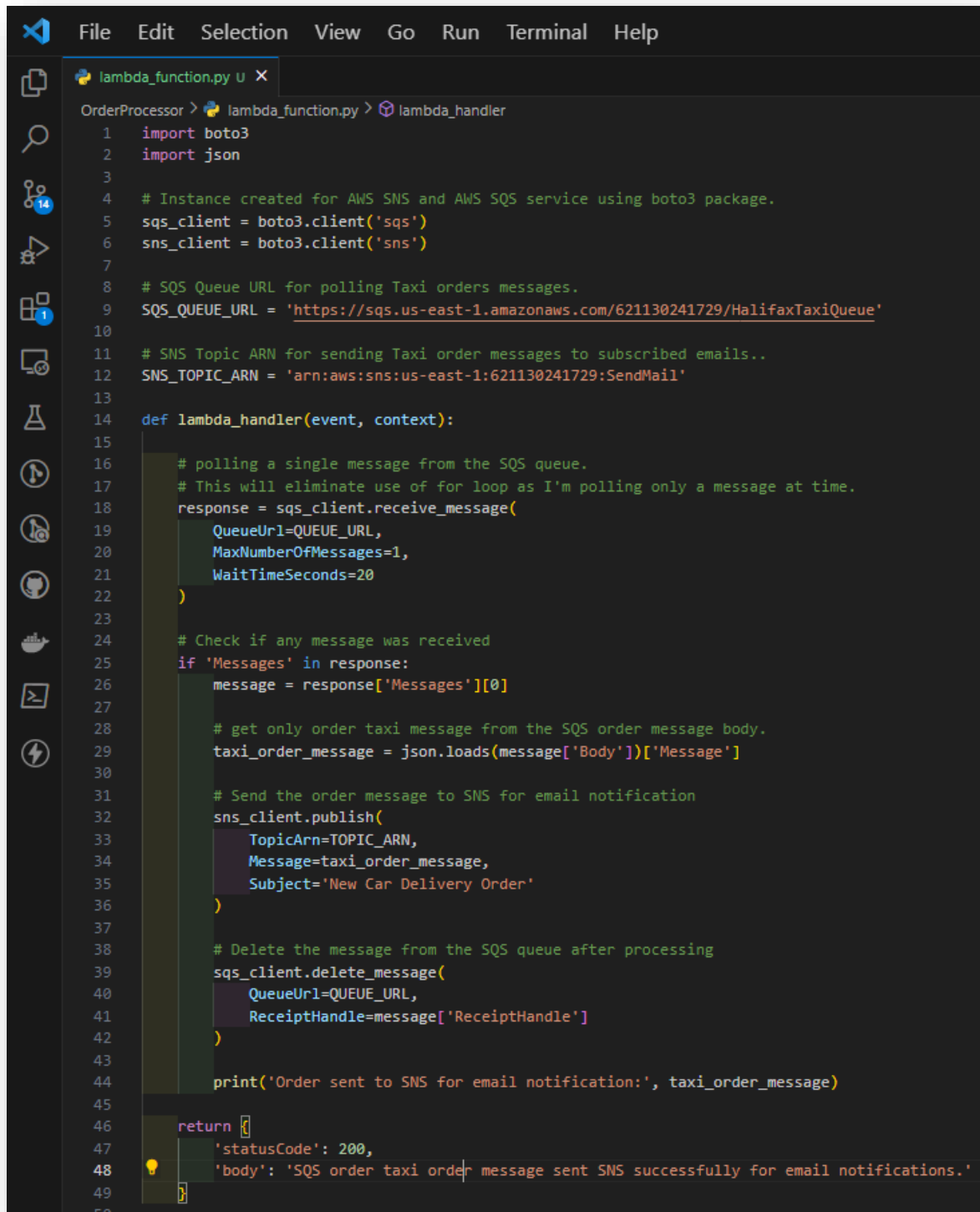


Figure 18 : ECR Repository for second Lambda Docker Image.

- Lambda function code is mentioned below, which will poll Taxi Order message from SQS(Every 2 min) and send it to SNS Topic(SendMail).



```
File Edit Selection View Go Run Terminal Help

lambda_function.py U X
OrderProcessor > lambda_function.py > lambda_handler
1 import boto3
2 import json
3
4 # Instance created for AWS SNS and AWS SQS service using boto3 package.
5 sqs_client = boto3.client('sqs')
6 sns_client = boto3.client('sns')
7
8 # SQS Queue URL for polling Taxi orders messages.
9 SQS_QUEUE_URL = 'https://sqs.us-east-1.amazonaws.com/621130241729/HalifaxTaxiQueue'
10
11 # SNS Topic ARN for sending Taxi order messages to subscribed emails..
12 SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:621130241729:SendMail'
13
14 def lambda_handler(event, context):
15
16     # polling a single message from the SQS queue.
17     # This will eliminate use of for loop as I'm polling only a message at time.
18     response = sqs_client.receive_message(
19         QueueUrl=QUEUE_URL,
20         MaxNumberOfMessages=1,
21         WaitTimeSeconds=20
22     )
23
24     # Check if any message was received
25     if 'Messages' in response:
26         message = response['Messages'][0]
27
28         # get only order taxi message from the SQS order message body.
29         taxi_order_message = json.loads(message['Body'])['Message']
30
31         # Send the order message to SNS for email notification
32         sns_client.publish(
33             TopicArn=TOPIC_ARN,
34             Message=taxi_order_message,
35             Subject='New Car Delivery Order'
36         )
37
38         # Delete the message from the SQS queue after processing
39         sqs_client.delete_message(
40             QueueUrl=QUEUE_URL,
41             ReceiptHandle=message['ReceiptHandle']
42         )
43
44         print('Order sent to SNS for email notification:', taxi_order_message)
45
46     return {
47         'statusCode': 200,
48         'body': 'SQS order taxi order message sent SNS successfully for email notifications.'
49     }
```

Figure 19 : Code for (poll_msg_send_mail) lambda function.

- Docker image for second lambda build, tagged and pushed to ECR newly created repository using custom deploy.sh script.

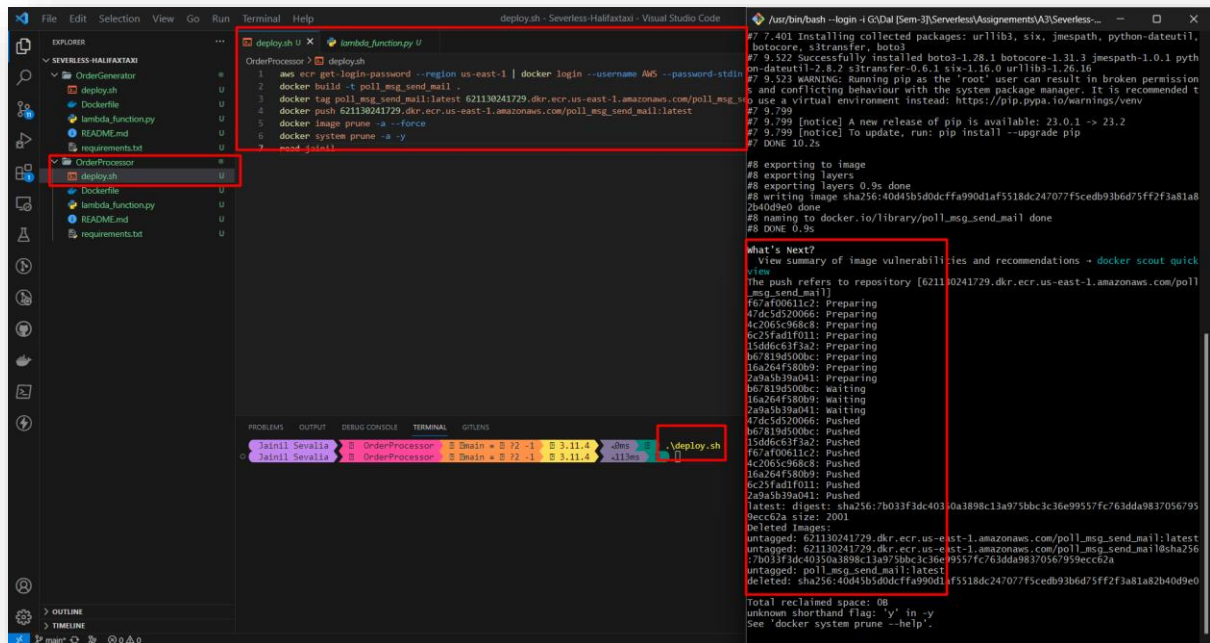


Figure 20 : Docker image for second Lambda is build, tagged, and pushed to ECR repository.

- ECR Repository AWS console, second lambda Docker image is successfully pushed.

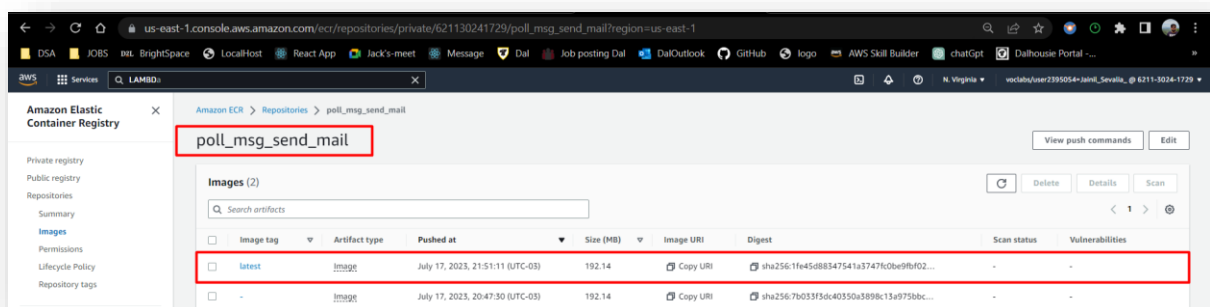


Figure 21 : Successfully Second lambda docker image is pushed. (AWS console).

- Create second lambda function which will poll the message from SQS queue and pass it to SNS Topic(SendMail).

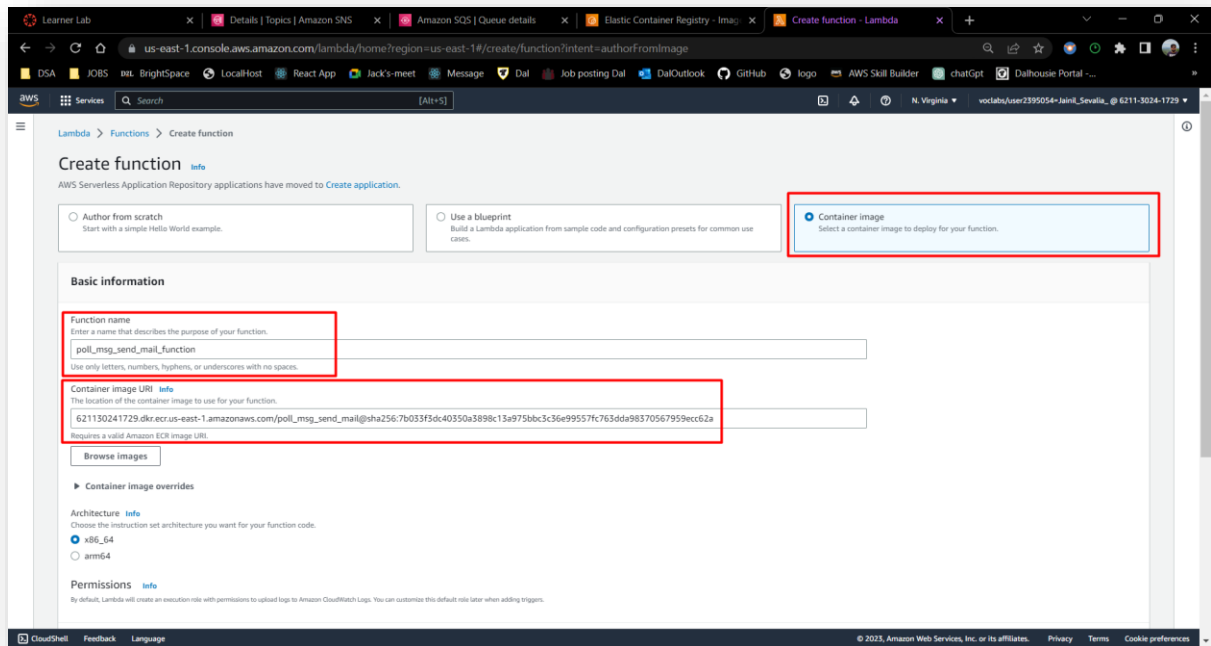


Figure 22 : Creating Second Lambda function poll_msg_send_mail. (AWS console).

- Configure Event Bridge which will invoke this lambda function every 2 min to check the SQS Queue for message. Expression cron(*/* * * * ? *) will invoke lambda every 2 min.

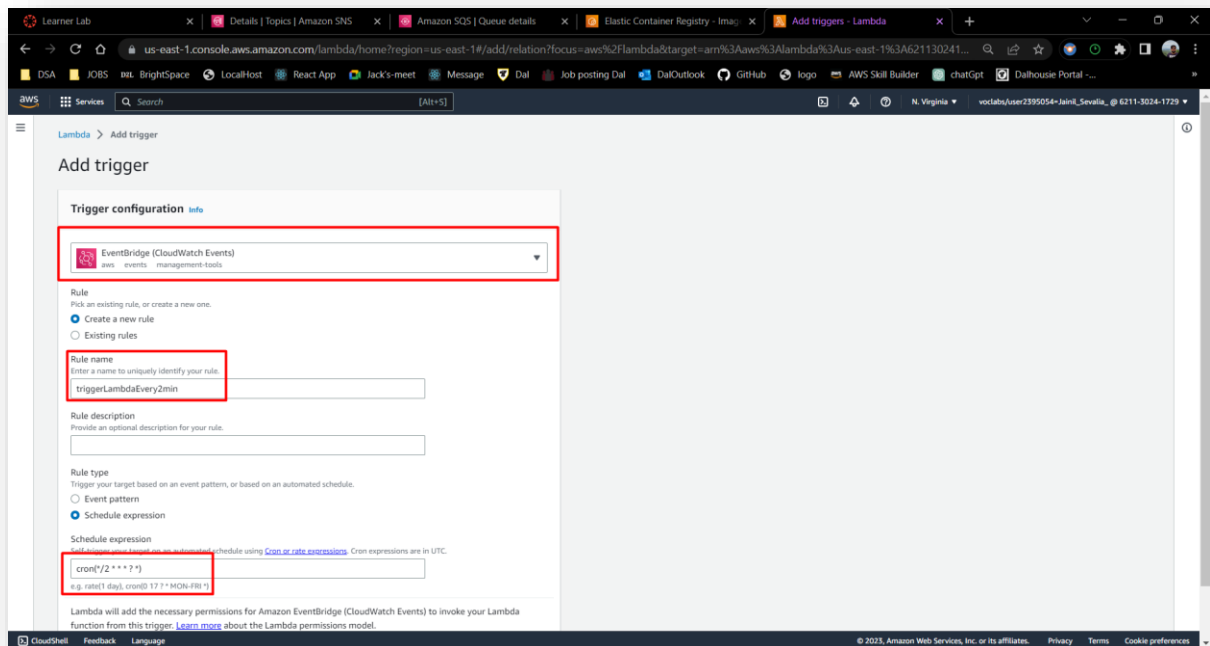


Figure 23 : Configure AWS Event bridge for invoking lambda every 2 min.

- Final Architecture for second lambda(poll_msg_send_mail) is mentioned below.

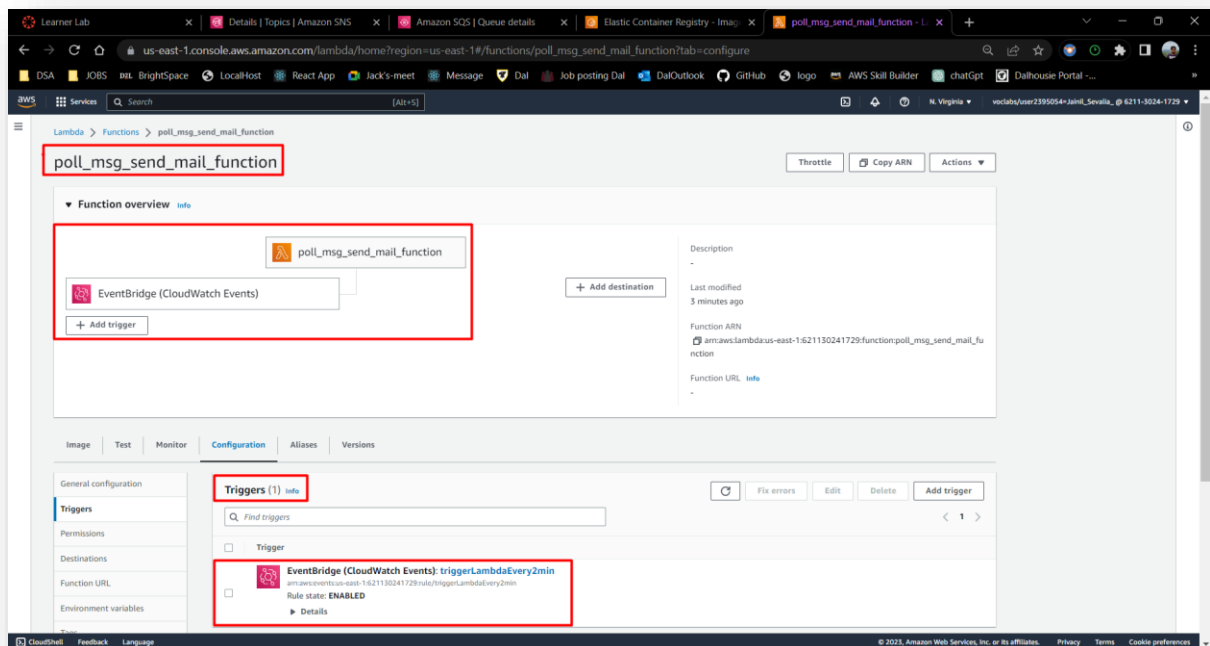


Figure 24 : Second lambda(poll_msg_send_mail) AWS console.

- Everything is set up as mentioned in the assignment guidelines. It's time for testing this HalifaxTaxi cloud serverless architecture.
- Let's hit the or trigger manually first lambda function(PublishOrder). This function will randomly generate order for taxi. This order will send to SNS topic(PublishOrder) and lead to SQS(HalifaxTaxiQueue) as that queue is subscribed to SNS topic.

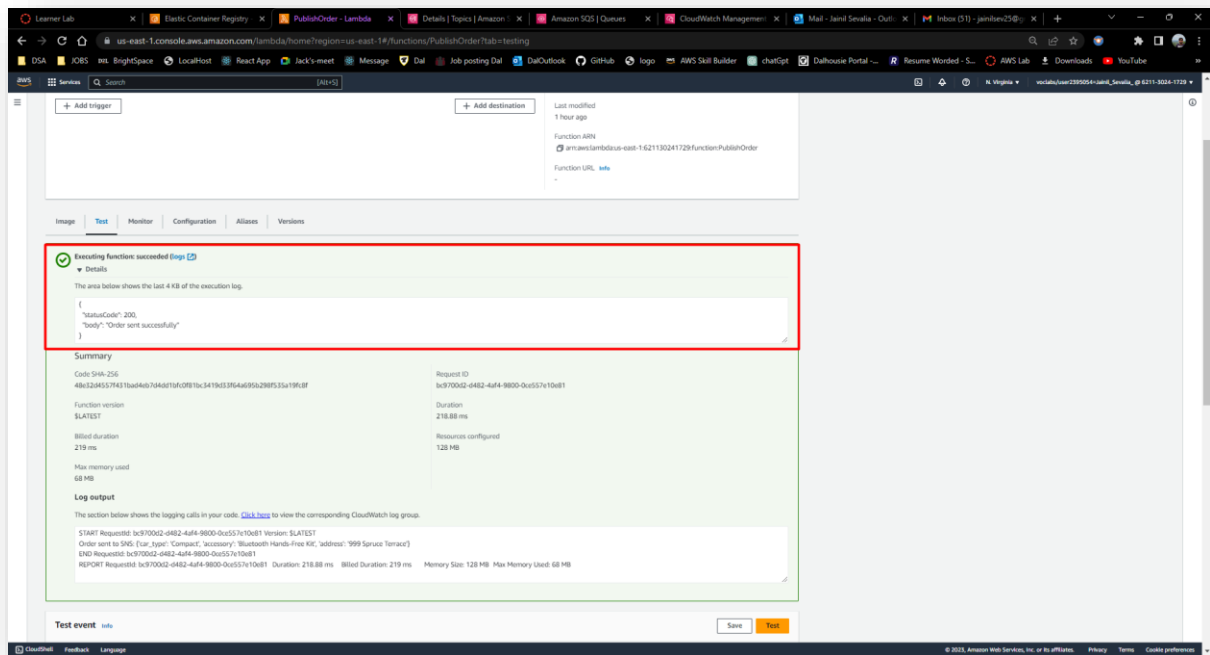


Figure 25: Testing - Triggering first Lambda function to generate order and send to SNS to SQS.

- At this point SQS(halifaxTaxiQueue) will receive random order message in the queue.
- After every 2min, Second Lambda(poll_msg_send_mail) will trigger. This function will poll message form SQS(HalifaxTaxiQueue) and Send that Message to SNS topic(SendMail). As 2 emails are added as subscriber, that emails will receive taxi order details.
- Here, I'm adding cloud watch log image and received email images as a part of testing.



Figure 26: Message that sent to SNS topic. (Cloud watch AWS console).

- Mail got into my account(Subscribed to SNS topic).

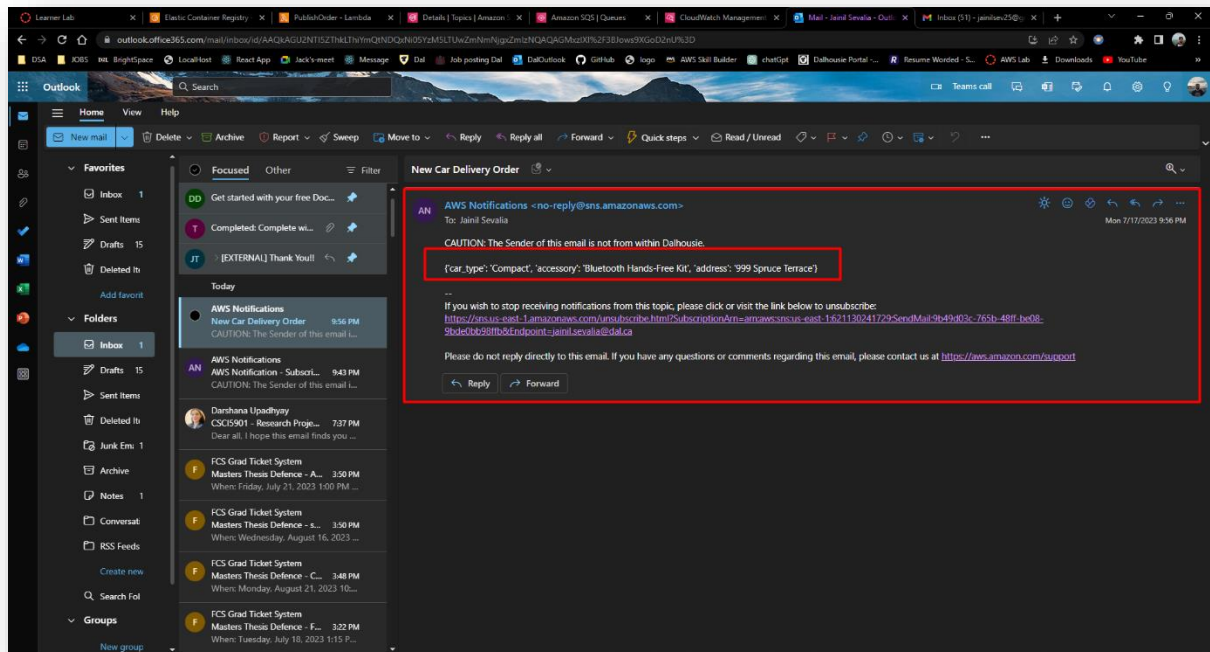


Figure 27 : Got email of with Taxi order Information.

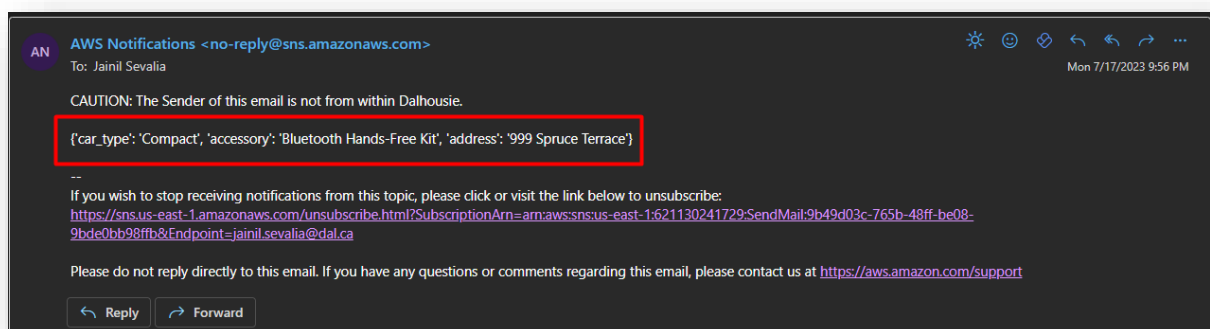


Figure 28 : Taxi Order Information got in mail.