CSCI 5410: Assignment 3: Part C:

Waleed R. Alhindi (B00919848)

GitLab A3 Part C Repository:

The A3-Part C code has been pushed to the GitLab repository's "A3" branch under the "Part_C" folder, which can be found at the following URL:
https://git.cs.dal.ca/alhindi/csci5410-summer-23-b00919848/-/tree/A3/Part_C

OR

Under the "A3" folder's "Part_C" sub-folder in the "main" branch, which can be found here:
https://git.cs.dal.ca/alhindi/csci5410-summer-23-b00919848/-/tree/main/A3/Part_C

Additionally, the professor and all TAs have been granted "Maintainer" access to this GitLab repository. This includes assigning "Maintainer" roles to the following GitLab accounts:

- *@saurabh (Dr. Saurabh Dey)*
- *@mudgal (Ankush Mudgal)*
- *@bharatwaaj (Bharatwaaj Shankanarayanan)*
- *@rmacwan (Rahul Ashokkumar Macwan)*

Architecture and Design:

Part C of this assignment tasks us with using AWS SQS [1], SNS [2], and Lambda [3] to design an event-driven system to simulate the ordering and fulfillment of taxi requests. As seen in Figure 1 below, an "orderTaxi" Lambda function [3] pushes a message containing information about a new taxi order to the "HalifaxTaxi" SNS Topic [2]. Three SQS Queues [1] are subscribed to this topic which queues the order messages for our next step. The second Lambda function [3], "fulfillTaxiRequest", is periodically called every 2 minutes to check whether a request message is in the queues. If a message is available, then this second Lambda function [3] will extract information from the queues' messages and publish the taxi order information to a second SNS Topic [2], "SendEmailNotification", that an email is subscribed to. Once a message has been published to the second topic, an email notification containing that message should be sent out.

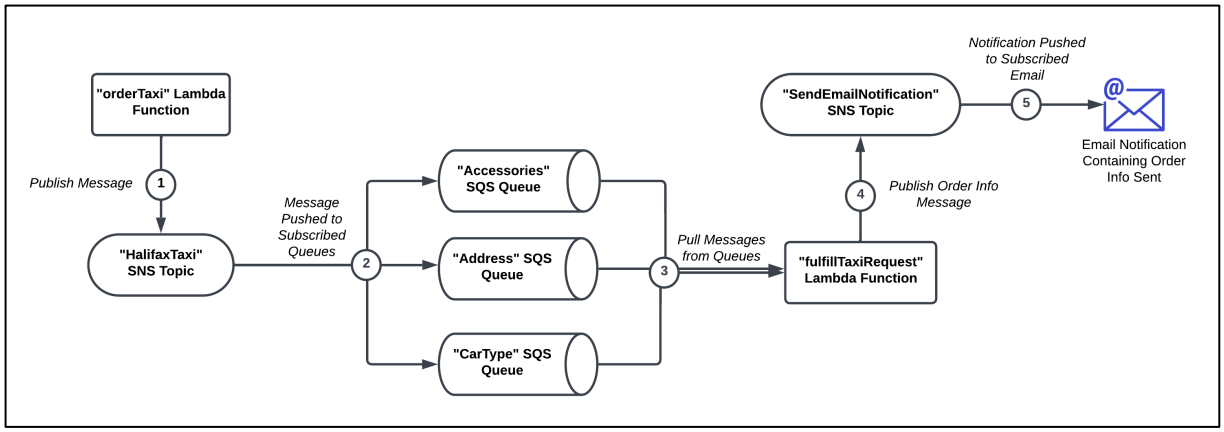


Fig 1. High-Level System Workflow [4]

To develop the system depicted in Figure 1, the following code files were composed and pushed to the designated Gitlab repository:

- *orderTaxi.py* – Source code of 1st Lambda [3] function, which publishes new taxi order messages to the "HalifaxTaxi" topic.
- *fulfillTaxiRequest.py* – Source code of 2nd Lambda [3] function that is periodically invoked by the "checkNewOrders.py" script to check whether the SQS Queues [1] have messages publishing that information to the "SendEmailNotification" topic.
- *checkNewOrders.py* – Python script used to invoke the "fulfillTaxiRequest" Lambda [3] every 2 minutes. This script is run locally.

<u>Operations Performed:</u>

1. Creating "Accessories", "Address", and "CarType" SQS Queues:

Before we create an SNS Topic [2], we first create three SQS Queues [1] that will be subscribed to the SNS Topic [2] and will queue published messages for our Lambda [3] functions to access. As such, three SQS Queues [1] were created; "Accessories", "Address", and "CarType", as seen in Figures 2, 3, and 4:

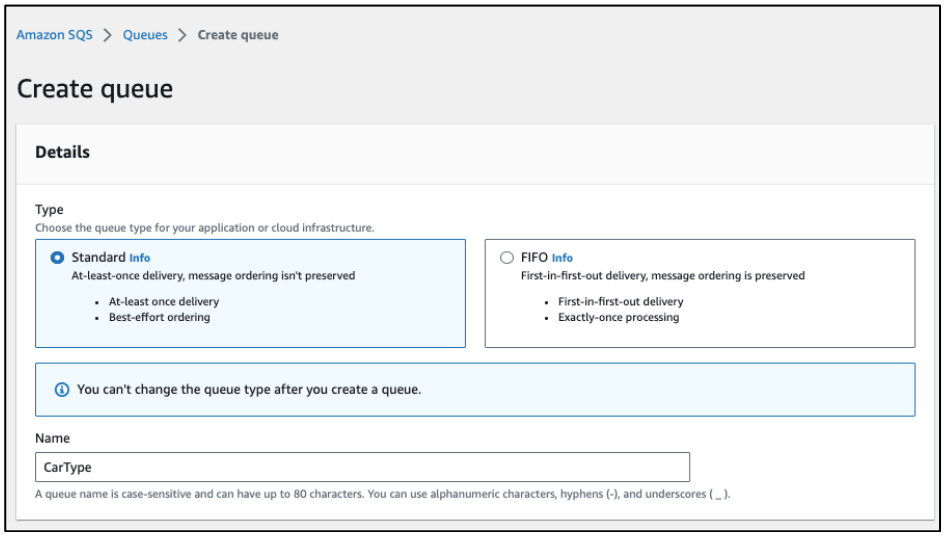


Fig 2. Creating "CarType" SQS Queue [1]

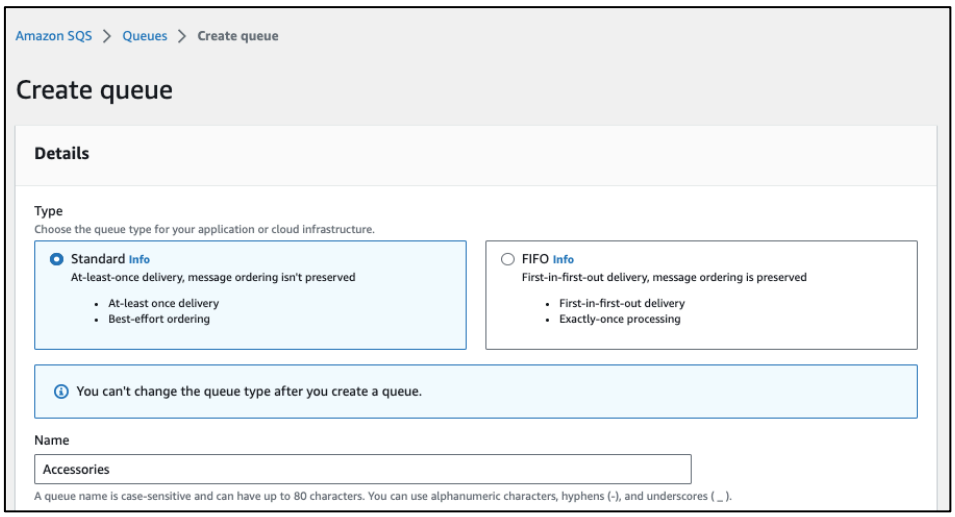


Fig 3. Creating "Accessories" SQS Queue [1]

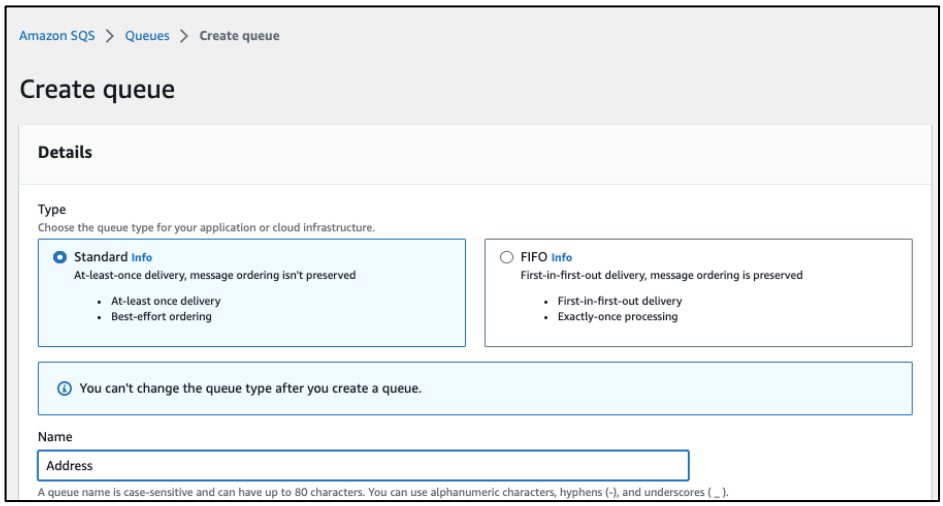


Fig 4. Creating "Address" SQS Queue [1]

We can verify that the SQS Queues [1] were created successfully by observing our SQS dashboard which displays the three created queues:

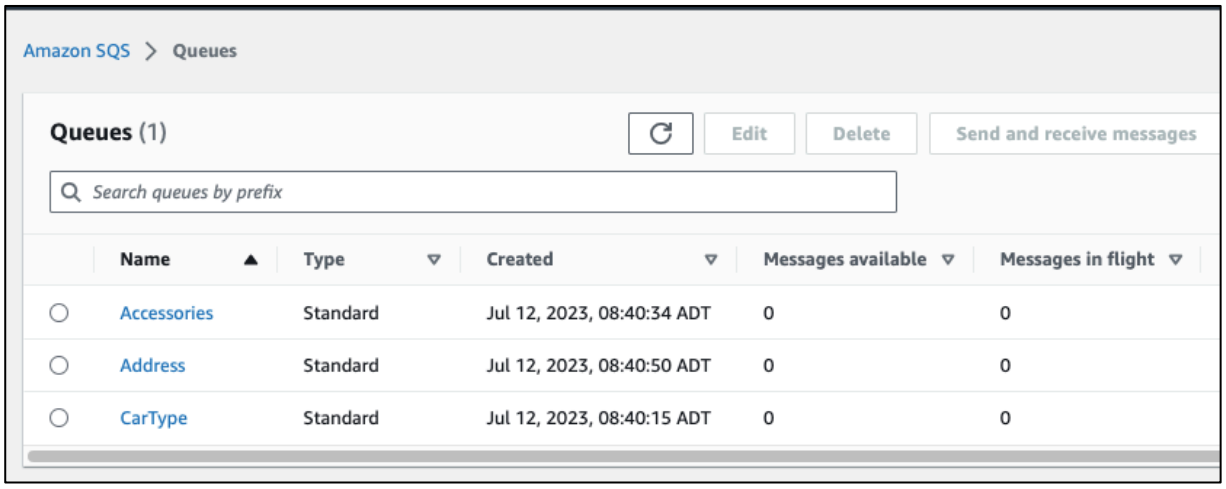


Fig 5. "Accessories", "Address", and "CarType" Queues Created Successfully [1]

2. ***Creating "HalifaxTaxi" SNS Topic:***

Next, we need to create the SNS Topic [2] that our first Lambda function [3] will be publishing messages to. Additionally, the three created SQS Queues [1] will subscribe to this topic to queue published messages for our second Lambda function [3] that fulfills taxi orders by sending an email notification containing the order details.

As such, we navigate to the SNS Console [2] and create a new topic called "HalifaxTaxi" as seen below:

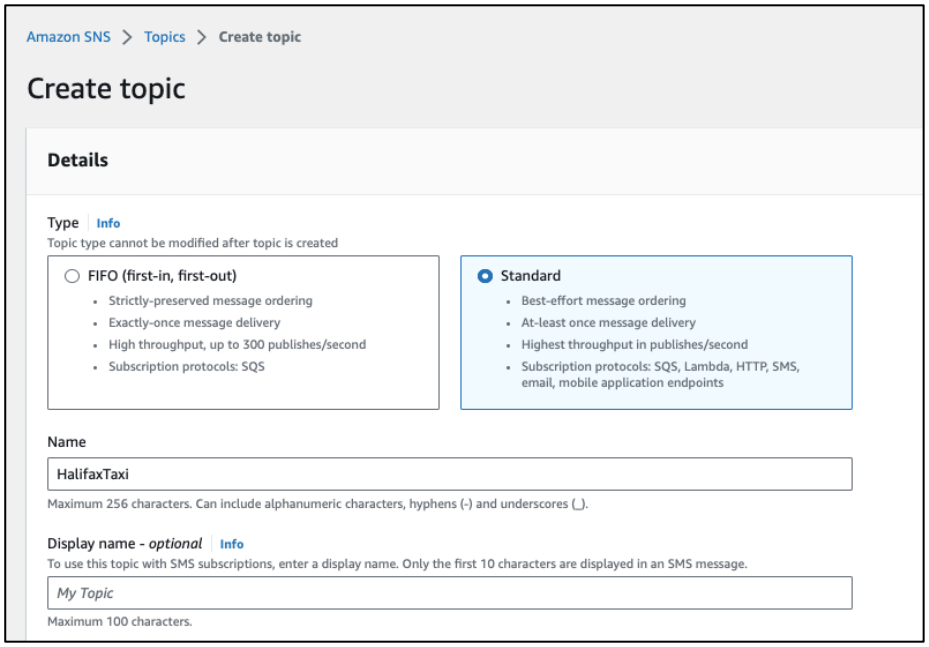


Fig 6. Creating "HalifaxTaxi" SNS Topic [2]

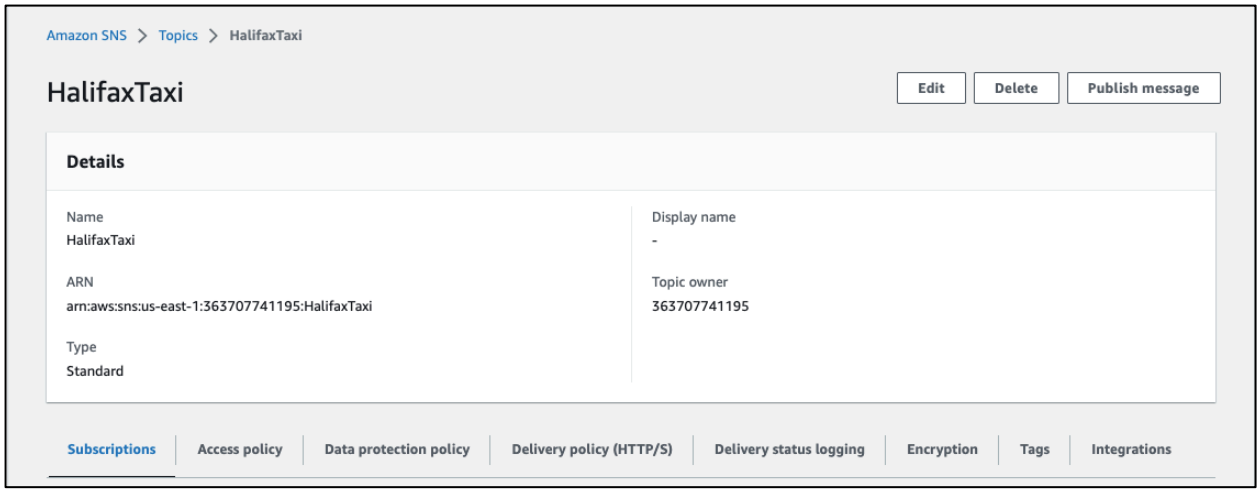


Fig 7. "HalifaxTaxi" SNS Topic Successfully Created [2]

3. *Subscribing Queues to "HalifaxTaxi" Topic:*

In order for the three SQS Queues [1] to receive messages from the "HalifaxTaxi" SNS Topic [2], we need to subscribe each of the three queues to the topic. The process depicted in Figure 8 is repeated for each of the three queues:

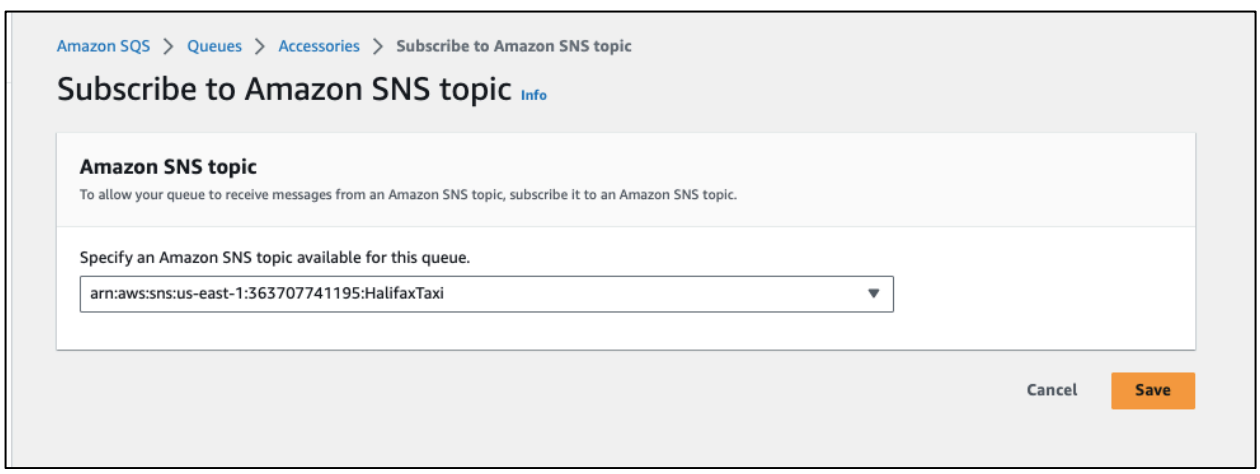


Fig 8. Subscribing "Accessories" Queue to "HalifaxTaxi" Topic [1]

Once we have subscribed each of the queues, we can verify the subscriptions back navigating back to "HalifaxTaxi" in the SNS Console [2], where we can see the three newly added subscription:

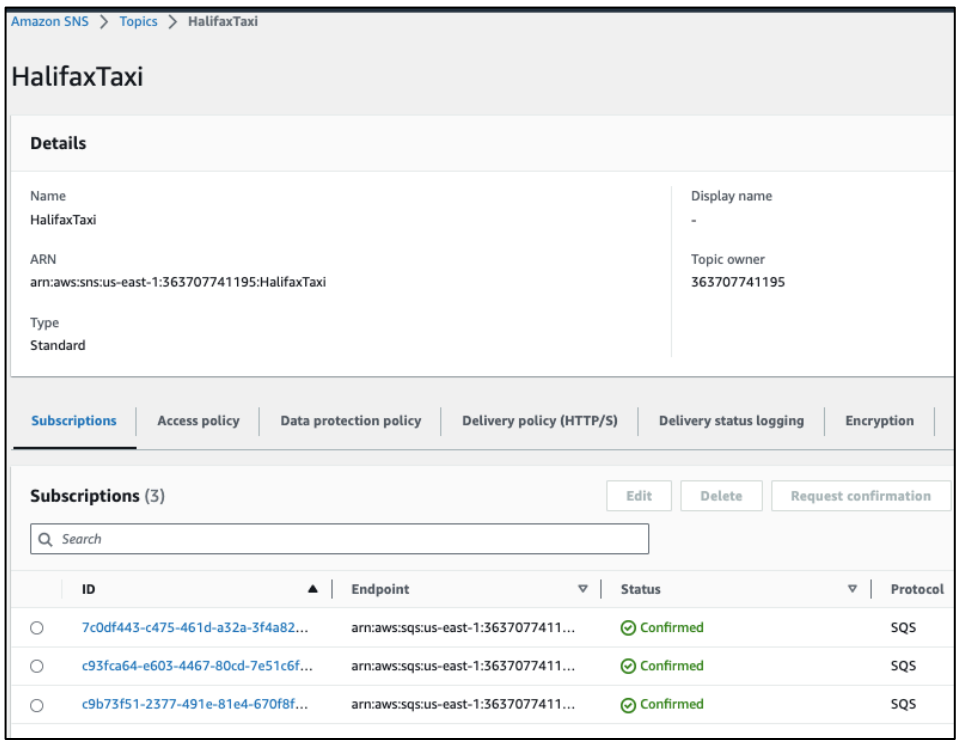


Fig 9. SQS Queues Successfully Subscribed to "HalifaxTaxi" SNS Topic [2]

4. *Creating "SendEmailNotification" SNS Topic:*

Next we need to create a second SNS Topic [2], named "SendEmailNotification", that will send an email notification containing the order information once our second Lambda function ("fulfillTaxiRequest") publishes a message to it.

Similarly to creating the "HalifaxTaxi" SQS Topic, we create the "SendEmailNotification" topic as seen below:

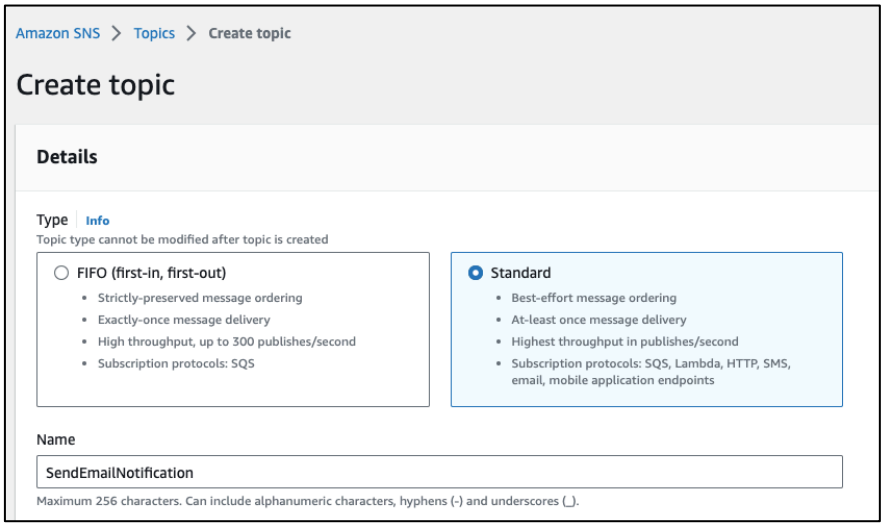


Fig 10. Creating "SendEmailNotification" SNS Topic [2]

5. ***Subscribe Email to "SendEmailNotification" Topic:***

However, unlike the "HalifaxTaxi" topic, we add an email subscription to "SendEmailNotification" rather than subscribing SQS Queues [1]. In this case, we will be using my Dalhousie Student Email (wl392785@dal.ca). Thus, we add that email subscription to the "SendEmailNotification" topic as seen below:

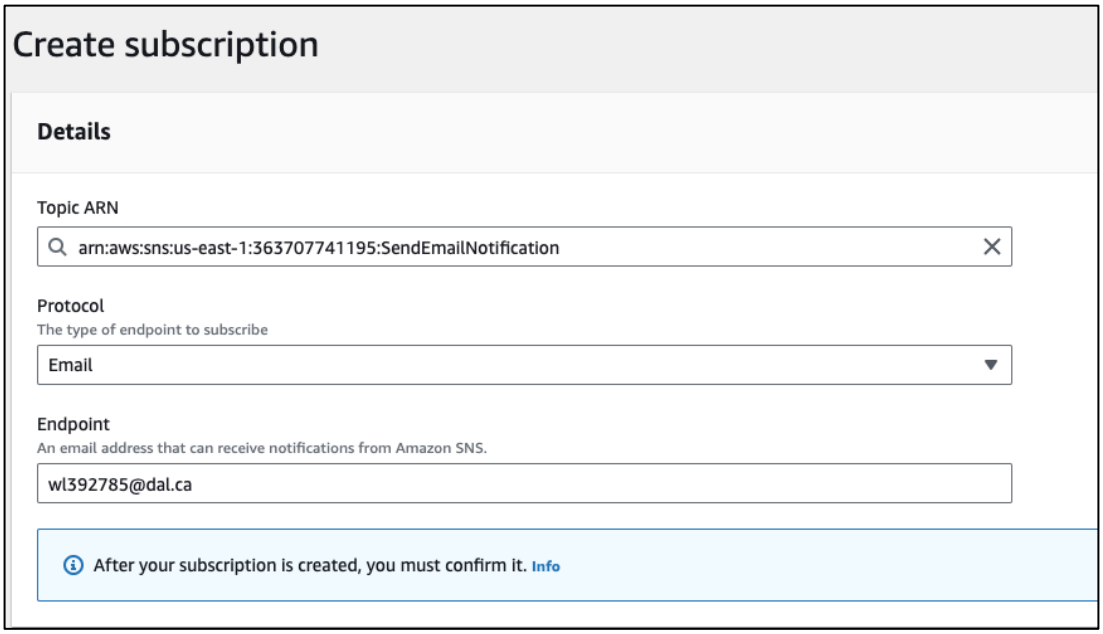


Fig 11. Adding Email Subscription to "SendEmailNotification" Topic [2]

Upon adding the subscription, we receive an email asking us to confirm this subscription as seen in Figure 12:

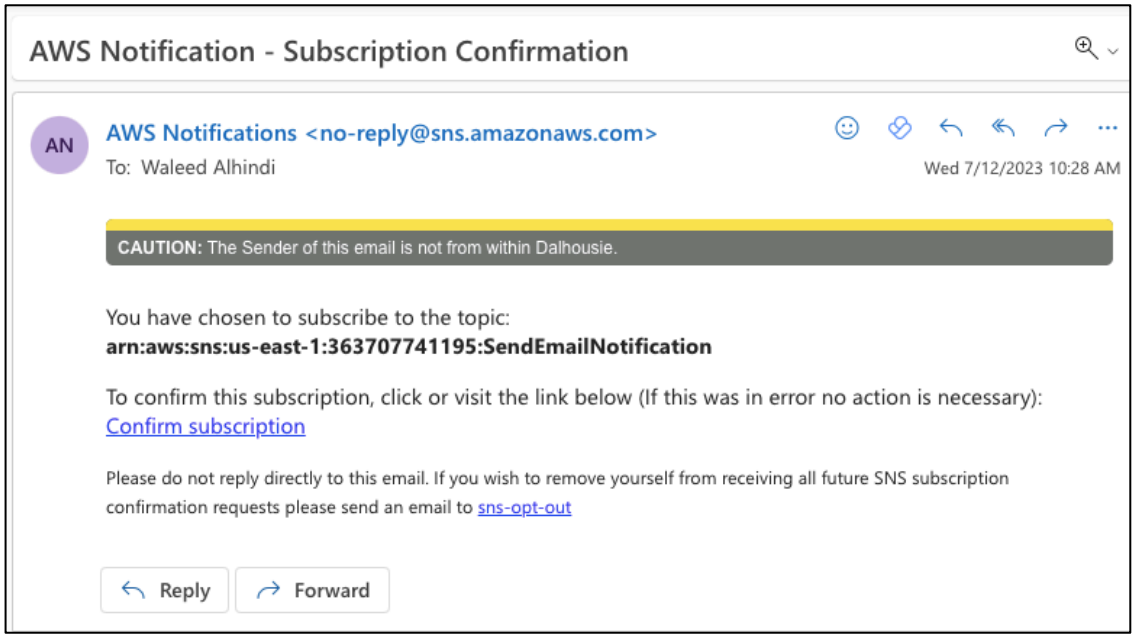


Fig 12. SNS Topic Email Subscription Confirmation Email

We confirm this subscription by clicking the “Confirm subscription” link in Figure 12, which will display the following confirmation message verifying our subscription:

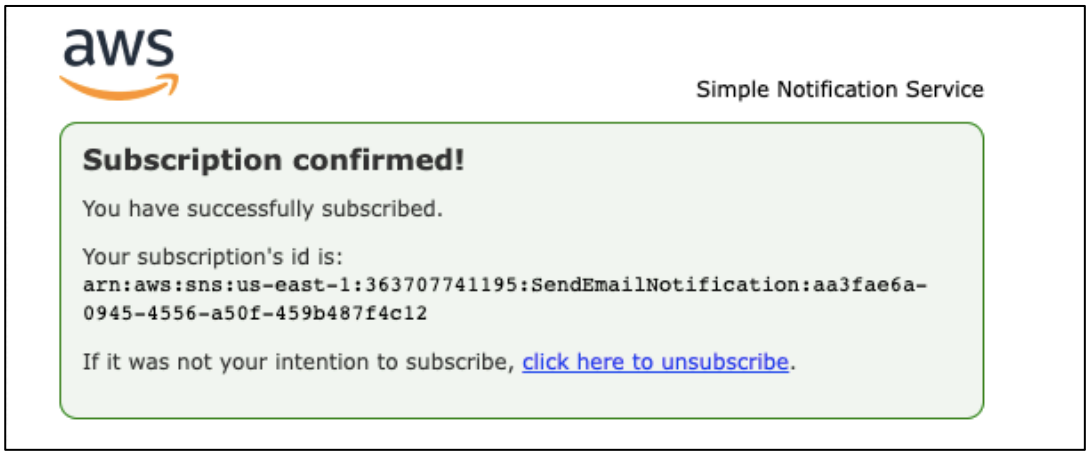


Fig 13. SNS Topic Email Subscription Confirmed

Lastly, we can confirm that the email subscription has been applied to the “SendEmailNotification” topic back navigating back to the SNS Console [2] where we can see that the subscription has indeed been created:

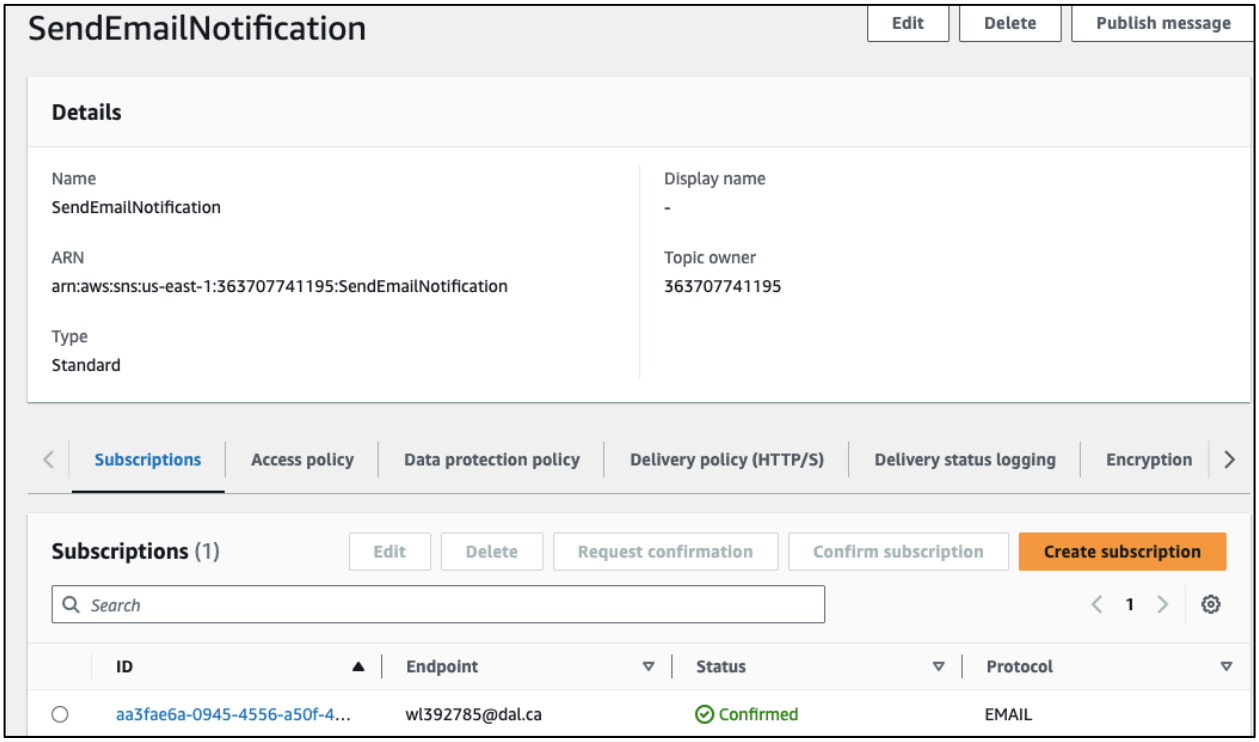
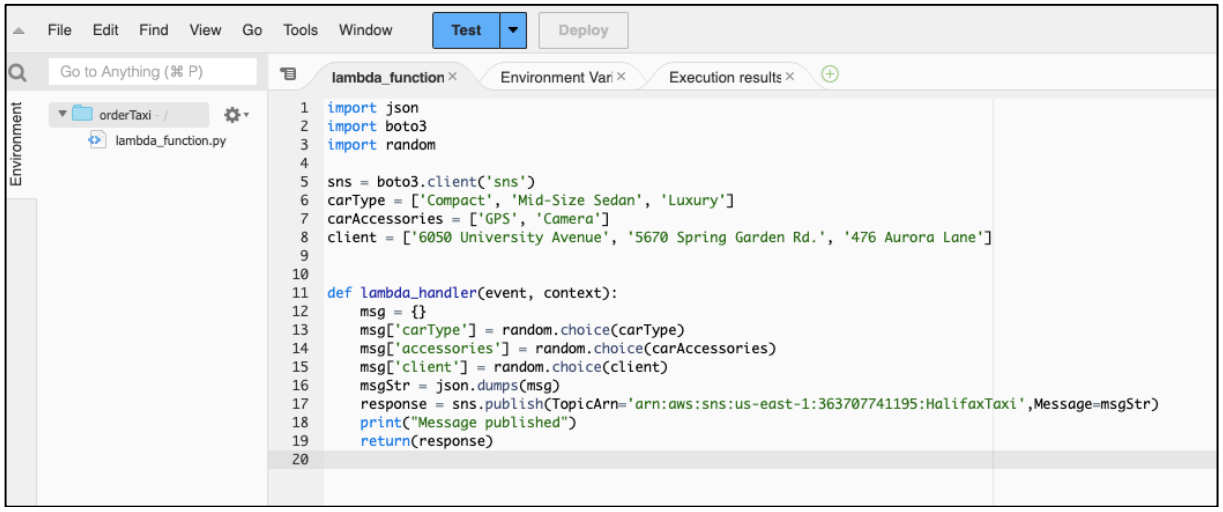


Fig 14. Email Subscription Created for “SendEmailNotification” Topic [2]

6. *Implementing "orderTaxi" Lambda:*

The "orderTaxi" Lambda [3] function simply generates a message by selecting options from three lists at random. This message is then published to the "HalifaxTaxi" SNS Topic [2]. The source code of this function is depicted below:




```
import json
import boto3
import random

sns = boto3.client('sns')
carType = ['Compact', 'Mid-Size Sedan', 'Luxury']
carAccessories = ['GPS', 'Camera']
client = ['6050 University Avenue', '5670 Spring Garden Rd.', '476 Aurora Lane']


def lambda_handler(event, context):
    msg = {}
    msg['carType'] = random.choice(carType)
    msg['accessories'] = random.choice(carAccessories)
    msg['client'] = random.choice(client)
    msgStr = json.dumps(msg)
    response = sns.publish(TopicArn='arn:aws:sns:us-east-1:363707741195:HalifaxTaxi',Message=msgStr)
    print("Message published")
    return(response)
```

Fig 15. "orderTaxi" Lambda Function's Source Code [3]

7. *Implementing "fulfillTaxiRequest" Lambda:*

Our 2nd Lambda function [3], "fulfullTaxiRequest" is invoked every 2 minutes and retrieves the messages in the SQS Queues [1]. Then, it generates a new message that contains the order information and publishes it to our 2nd SNS Topic [2], "SendEmailNotification".

```python
import json
import boto3

def lambda_handler(event, context):

    sqs = boto3.client('sqs')
    carTypeRes = sqs.receive_message(
        QueueUrl="https://sqs.us-east-1.amazonaws.com/363707741195/CarType",
        MaxNumberOfMessages = 1,
        WaitTimeSeconds = 10
    )

    accessoriesRes = sqs.receive_message(
        QueueUrl="https://sqs.us-east-1.amazonaws.com/363707741195/Accessories",
        MaxNumberOfMessages = 1,
        WaitTimeSeconds = 10
    )

    addressRes = sqs.receive_message(
        QueueUrl="https://sqs.us-east-1.amazonaws.com/363707741195/Address",
        MaxNumberOfMessages = 1,
        WaitTimeSeconds = 10
    )

    print(carTypeRes)
    ct1 = json.loads(carTypeRes['Messages'][0]['Body'])
    ct2 = json.loads(ct1['Message'])
    carType = ct2['carType']

    print(accessoriesRes)
    ar1 = json.loads(accessoriesRes['Messages'][0]['Body'])
    ar2 = json.loads(ar1['Message'])
    accessories = ar2['accessories']

    print(addressRes)
    adr1 = json.loads(addressRes['Messages'][0]['Body'])
    adr2 = json.loads(adr1['Message'])
    address = adr2['client']

    emailStr = "New Order!\nCar Type: " + carType + "\nAccessories: " + accessories + "\nClient Address: " + address

    emailSNS = boto3.client('sns')
    emailSNSResponse = emailSNS.publish(
        TargetArn = 'arn:aws:sns:us-east-1:363707741195:SendEmailNotification',
        Message = emailStr,
        MessageStructure = 'text'
    )

    return {
      'statusCode': 200,
      'body': json.dumps(emailSNSResponse)
    }
```

Fig 16. "fulfillTaxiRequest" Lambda Function's Source Code [3]

8. *Implementing Script to Invoke "fulfillTaxiRequest" Every 2 Minutes:*

The last piece of code we need to implement is the script that will periodically invoke the "fulfillTaxiRequest" Lambda [3] every 2 minutes. This code shown in Figure 17 depicts the simple Python script, "checkNewOrders", which will be run locally to remotely invoke "fulfillTaxiRequest":

```
Part_C > checkNewOrders.py
import json
import boto3
import time
import time


lambdaClient = boto3.client('lambda')

while True:
    response = lambdaClient.invoke(
        FunctionName='arn:aws:lambda:us-east-1:363707741195:function:fulfillTaxiRequest'
    )
    currentTime = time.localtime()
    currTimeStr = time.strftime("%H:%M:%S", currentTime)
    print(currTimeStr)
    print(response)
    print("----------")
    time.sleep(120)
```

Fig 17. "checkNewOrder" Python Script

9. *Verification and Testing:*

Now that we have set up all the necessary pieces, we need to verify our event-driven service by testing its workflow.

First, we invoke the "orderTaxi" Lambda [3]. After invoking the Lambda [3], we observe the "HalifaxTaxi" SNS Topic [2] for new messages. As seen in Figure 18, polling the "Accessories" queue, for example, shows that messages have indeed been received:

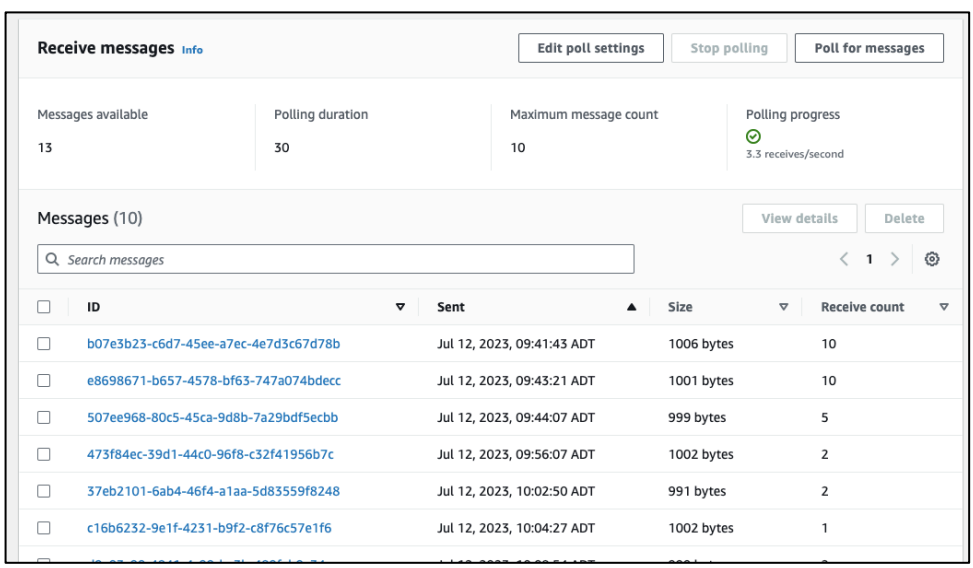


Fig 18. Messages Polled from "Accessories" Queue After Lambda Invocations [1]

We can verify the contents of the messages by navigating into one of the messages as a sample as seen in Figure 19:

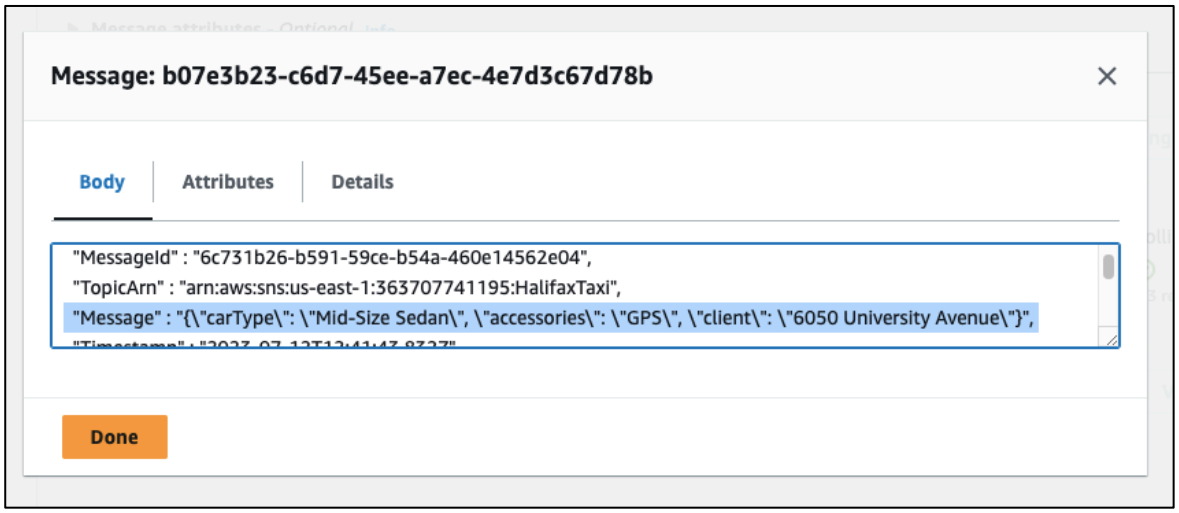


Fig 19. Verifying Sample Queue Message Content [1]

Additionally, we can verify "orderTaxi" execution by examining its CloudWatch [5] logs and examining the log stream of the recent Lambda [3] invocation:

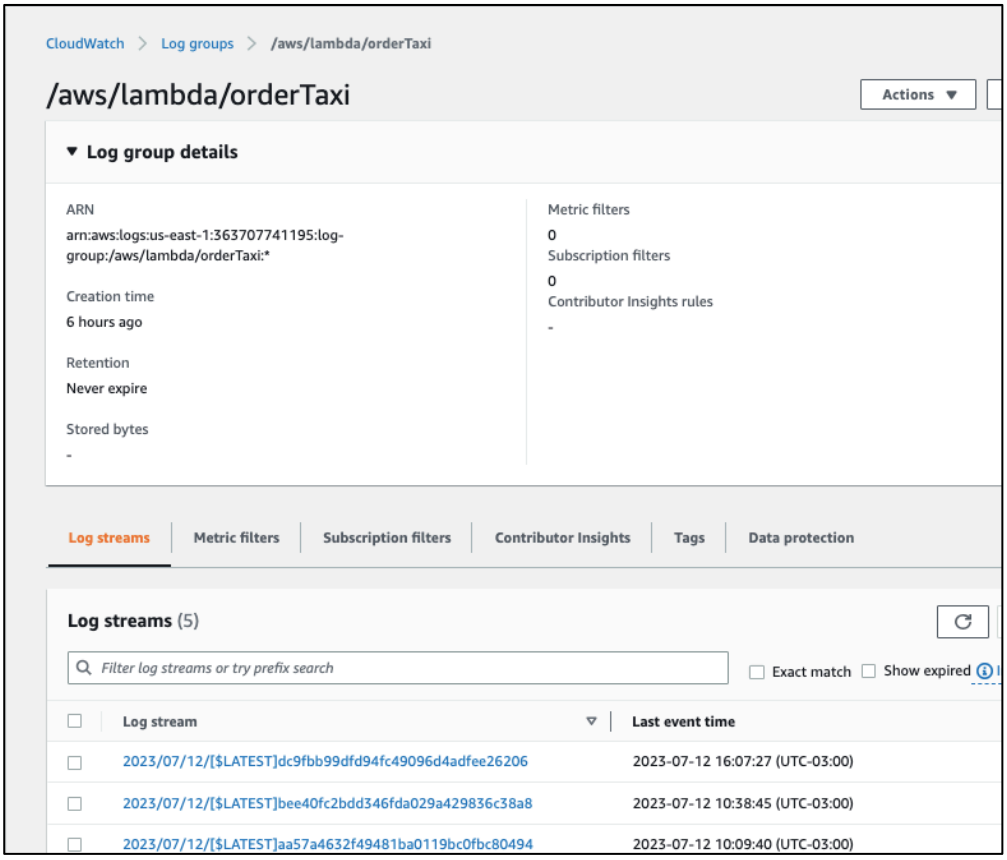


Fig 20. "orderTaxi" Execution Logs in CloudWatch [5]

```
2023-07-12T16:07:23.394-03:00	INIT_START Runtime Version: python:3.10.v5 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:51b…
2023-07-12T16:07:23.838-03:00	START RequestId: 44296813-5497-4b62-b2f0-b7c5fc699422 Version: $LATEST

START RequestId: 44296813-5497-4b62-b2f0-b7c5fc699422 Version: $LATEST
Copy
2023-07-12T16:07:24.104-03:00	Message published

Message published
Copy
2023-07-12T16:07:24.119-03:00	END RequestId: 44296813-5497-4b62-b2f0-b7c5fc699422

END RequestId: 44296813-5497-4b62-b2f0-b7c5fc699422
Copy
2023-07-12T16:07:24.119-03:00	REPORT RequestId: 44296813-5497-4b62-b2f0-b7c5fc699422 Duration: 279.40 ms Billed Duration: 280 ms Me…

REPORT RequestId: 44296813-5497-4b62-b2f0-b7c5fc699422  Duration: 279.40 ms	Billed Duration: 280 ms Memory Size: 128 MB
Max Memory Used: 68 MB  Init Duration: 443.43 ms
Copy
```

Fig 21. Sample "orderTaxi" Invocation Log Stream [5]

Next, we must verify our second Lambda [3], "fulfillTaxiOrder". This function is invoked every 2 minutes by locally running the "checkNewOrder" Python script. Thus, we run the "checkNewOrder" script and keep it running for about 8 minutes, which invokes the Lambda [3] four times:

```
checkNewOrders.py — A3
checkNewOrders.py U
Part_C > checkNewOrders.py
import boto3
import time
import time


lambdaClient = boto3.client('lambda')

while True:
    response = lambdaClient.invoke(
        FunctionName='arn:aws:lambda:us-east-1:363707741195:function:fulfillTaxiRequest'
    )
    currentTime = time.localtime()
    currTimeStr = time.strftime("%H:%M:%S", currentTime)
    print(currTimeStr)
    print(response)
    print("----------")
    time.sleep(120)
```

```
Part_C — Python checkNewOrders.py — 80×43
Waleeds-MacBook-Air:Part_C waleedalhindi$ python3 checkNewOrders.py
16:19:39
{'ResponseMetadata': {'RequestId': '90084aa6-876c-4916-9f6f-1ad0f81830d0', 'HTTP
StatusCode': 200, 'HTTPHeaders': {'date': 'Wed, 12 Jul 2023 19:19:39 GMT', 'cont
ent-type': 'application/json', 'content-length': '405', 'connection': 'keep-aliv
e', 'x-amzn-requestid': '90084aa6-876c-4916-9f6f-1ad0f81830d0', 'x-amzn-remapped
-content-length': '0', 'x-amz-executed-version': '$LATEST', 'x-amzn-trace-id': '
root=1-64aefcca-4f1c1fe500d2fd705344f5fd;sampled=0;lineage=a292b208:0'}, 'RetryA
ttempts': 0}, 'StatusCode': 200, 'ExecutedVersion': '$LATEST', 'Payload': <botoc
ore.response.StreamingBody object at 0x7febb00351f0>}
----------
16:21:39
{'ResponseMetadata': {'RequestId': '2d58ebed-363a-46b3-8811-934a4cda96a1', 'HTTP
StatusCode': 200, 'HTTPHeaders': {'date': 'Wed, 12 Jul 2023 19:21:39 GMT', 'cont
ent-type': 'application/json', 'content-length': '405', 'connection': 'keep-aliv
e', 'x-amzn-requestid': '2d58ebed-363a-46b3-8811-934a4cda96a1', 'x-amzn-remapped
-content-length': '0', 'x-amz-executed-version': '$LATEST', 'x-amzn-trace-id': '
root=1-64aefd43-46d65c9361a25bd92a7540c7;sampled=0;lineage=a292b208:0'}, 'RetryA
ttempts': 0}, 'StatusCode': 200, 'ExecutedVersion': '$LATEST', 'Payload': <botoc
ore.response.StreamingBody object at 0x7febb0035430>}
----------
16:23:40
{'ResponseMetadata': {'RequestId': '6856fab8-5314-4bca-94ba-18715d0c397a', 'HTTP
StatusCode': 200, 'HTTPHeaders': {'date': 'Wed, 12 Jul 2023 19:23:40 GMT', 'cont
ent-type': 'application/json', 'content-length': '405', 'connection': 'keep-aliv
e', 'x-amzn-requestid': '6856fab8-5314-4bca-94ba-18715d0c397a', 'x-amzn-remapped
-content-length': '0', 'x-amz-executed-version': '$LATEST', 'x-amzn-trace-id': '
root=1-64aefdbc-1fc640621344a84c06662074;sampled=0;lineage=a292b208:0'}, 'RetryA
ttempts': 0}, 'StatusCode': 200, 'ExecutedVersion': '$LATEST', 'Payload': <botoc
ore.response.StreamingBody object at 0x7febb00355e0>}
----------
16:25:41
{'ResponseMetadata': {'RequestId': '0d255bd5-e0f7-462f-96fa-04a0e7703a6f', 'HTTP
StatusCode': 200, 'HTTPHeaders': {'date': 'Wed, 12 Jul 2023 19:25:41 GMT', 'cont
ent-type': 'application/json', 'content-length': '405', 'connection': 'keep-aliv
e', 'x-amzn-requestid': '0d255bd5-e0f7-462f-96fa-04a0e7703a6f', 'x-amzn-remapped
-content-length': '0', 'x-amz-executed-version': '$LATEST', 'x-amzn-trace-id': '
root=1-64aefe35-7cf102377eb72ba23716d6d;sampled=0;lineage=a292b208:0'}, 'RetryA
ttempts': 0}, 'StatusCode': 200, 'ExecutedVersion': '$LATEST', 'Payload': <botoc
ore.response.StreamingBody object at 0x7febb0035670>}
----------
```

Fig 22. Executing "checkNewOrder" Script to Invoke Lambda Every 20 Minutes

Since the "fulfillTaxiRequest" Lambda [3] was invoked four times, we can see that four emails have been received in my Dalhousie email's inbox. What's more, examining the contents of one of the emails verifies that the message contains the taxi request information as expected:

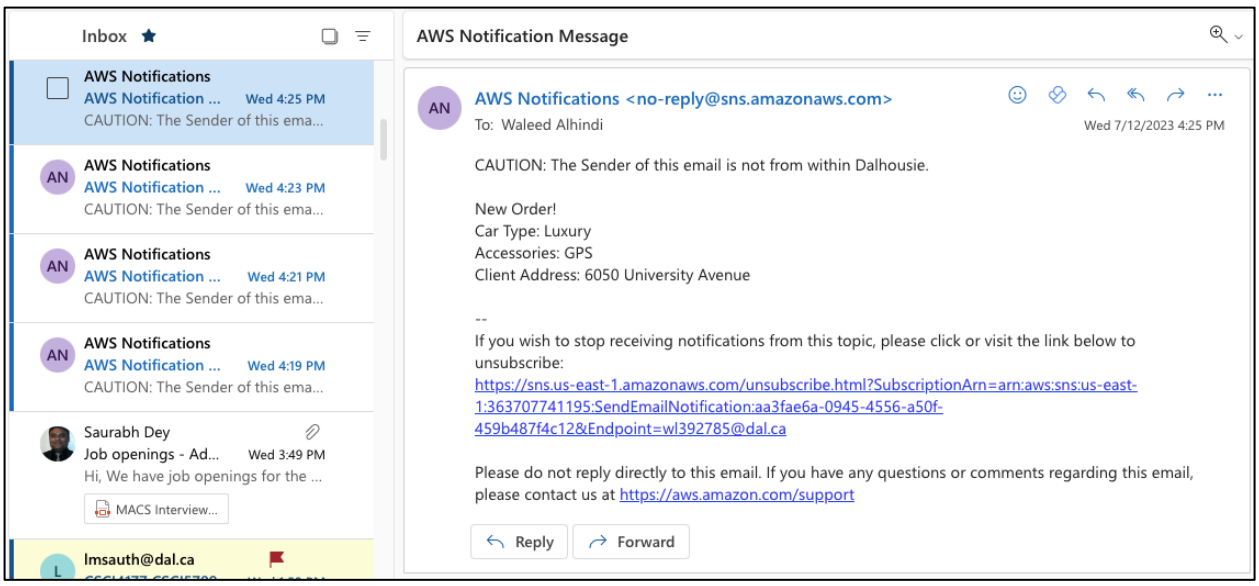


Fig 23. SNS Emails Successfully Received and Contents Verified

Lastly, let us check the execution logs of the "fulfillTaxiRequest" Lambda [3] in CloudWatch [5] and examining the latest log stream:

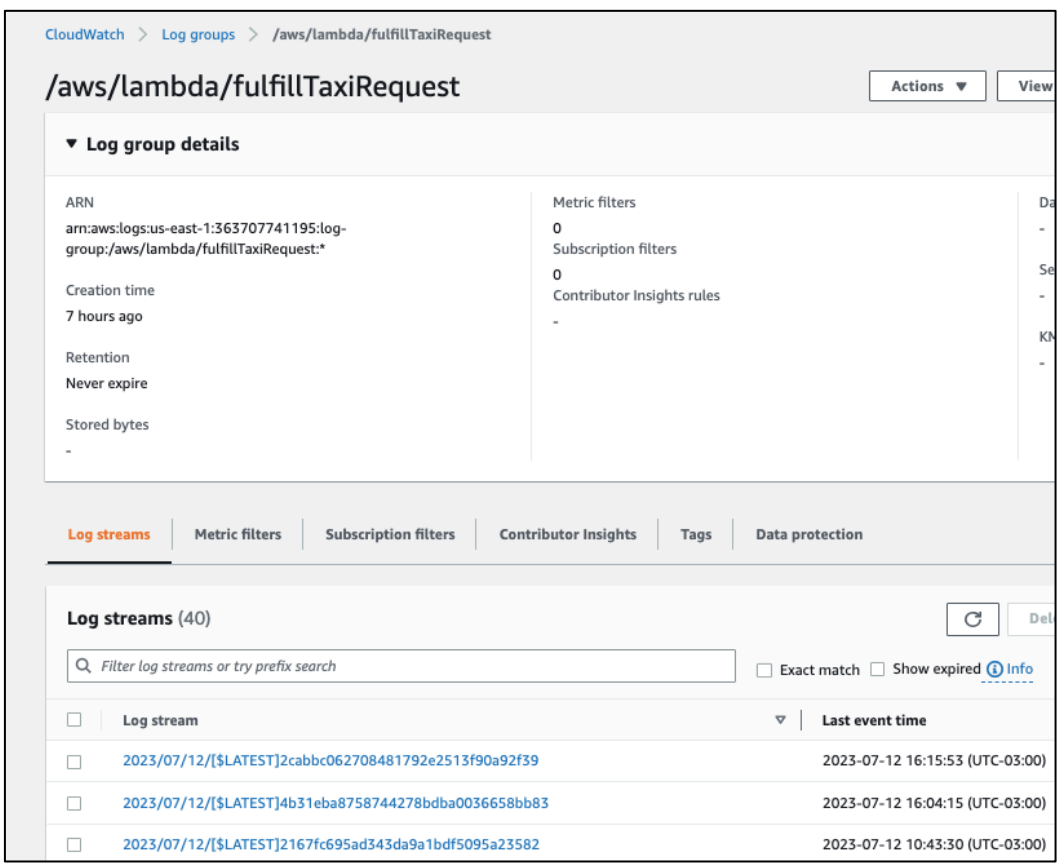
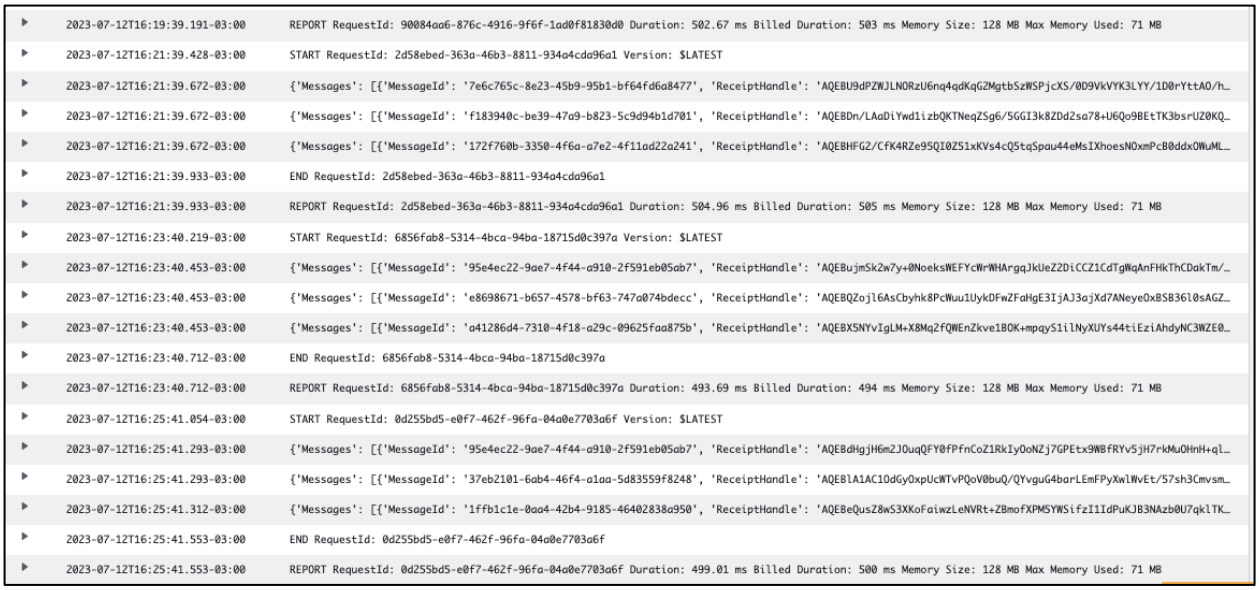


Fig 24. "fulfillTaxiRequest" Log Streams in CloudWatch [5]



```
2023-07-12T16:19:39.191-03:00    REPORT RequestId: 90084aa6-876c-4916-9f6f-1ad0f81830d0 Duration: 502.67 ms Billed Duration: 503 ms Memory Size: 128 MB Max Memory Used: 71 MB
2023-07-12T16:21:39.428-03:00    START RequestId: 2d58ebed-363a-46b3-8811-934a4cda96a1 Version: $LATEST
2023-07-12T16:21:39.672-03:00    {'Messages': [{'MessageId': '7e6c765c-8e23-45b9-95b1-bf64fd6a8477', 'ReceiptHandle': 'AQEBU9dPZWJLNORzU6nq4qdKqG2MgtbSzWSPjcXS/0D9VkVYK3LYY/1D0rYttAO/h…
2023-07-12T16:21:39.672-03:00    {'Messages': [{'MessageId': 'f183940c-be39-47a9-b823-5c9d94b1d701', 'ReceiptHandle': 'AQEBDn/LAaDiYwd1izbQKTNeqZSg6/5GGI3k8ZDd2sa78+U6Qo9BEtTK3bsrUZ0KQ…
2023-07-12T16:21:39.672-03:00    {'Messages': [{'MessageId': '172f760b-3350-4f6a-a7e2-4f11ad22a241', 'ReceiptHandle': 'AQEBHFG2/CfK4RZe9SQI0Z51xKVs4cQ5tqSpau44eMsIXhoesNOxmPcB0ddxOWuML…
2023-07-12T16:21:39.933-03:00    END RequestId: 2d58ebed-363a-46b3-8811-934a4cda96a1
2023-07-12T16:21:39.933-03:00    REPORT RequestId: 2d58ebed-363a-46b3-8811-934a4cda96a1 Duration: 504.96 ms Billed Duration: 505 ms Memory Size: 128 MB Max Memory Used: 71 MB
2023-07-12T16:23:40.219-03:00    START RequestId: 6856fab8-5314-4bca-94ba-18715d0c397a Version: $LATEST
2023-07-12T16:23:40.453-03:00    {'Messages': [{'MessageId': '95e4ec22-9ae7-4f44-a910-2f591eb05ab7', 'ReceiptHandle': 'AQEBujmSk2w7y+0NoeksWEFYcWrWHArgqJkUeZ2DiCCZ1CdTgWqAnFHkThCDakTm/…
2023-07-12T16:23:40.453-03:00    {'Messages': [{'MessageId': 'e8698671-b657-4578-bf63-747a074bdecc', 'ReceiptHandle': 'AQEBQZojl6AsCbyhk8PcWuu1UykDFwZFaHgE3IjAJ3ajXd7ANeyeOxBSB36l0sAGZ…
2023-07-12T16:23:40.453-03:00    {'Messages': [{'MessageId': 'a41286d4-7310-4f18-a29c-09625faa875b', 'ReceiptHandle': 'AQEBX5NYvIgLM+X8Mq2fQWEnZkve1BOK+mpqyS1ilNyXUYs44tiEziAhdyNC3WZE0…
2023-07-12T16:23:40.712-03:00    END RequestId: 6856fab8-5314-4bca-94ba-18715d0c397a
2023-07-12T16:23:40.712-03:00    REPORT RequestId: 6856fab8-5314-4bca-94ba-18715d0c397a Duration: 493.69 ms Billed Duration: 494 ms Memory Size: 128 MB Max Memory Used: 71 MB
2023-07-12T16:25:41.054-03:00    START RequestId: 0d255bd5-e0f7-462f-96fa-04a0e7703a6f Version: $LATEST
2023-07-12T16:25:41.293-03:00    {'Messages': [{'MessageId': '95e4ec22-9ae7-4f44-a910-2f591eb05ab7', 'ReceiptHandle': 'AQEBdHgjH6m2JOuqQFY0fPfnCoZ1RkIyOoNZj7GPEtx9WBfRYv5jH7rkMuOHnH+ql…
2023-07-12T16:25:41.293-03:00    {'Messages': [{'MessageId': '37eb2101-6ab4-46f4-a1aa-5d83559f8248', 'ReceiptHandle': 'AQEBlA1AC1OdGyOxpUcWTvPQoV0buQ/QYvguG4barLEmFPyXwlWvEt/57sh3Cmvsm…
2023-07-12T16:25:41.312-03:00    {'Messages': [{'MessageId': '1ffb1c1e-0aa4-42b4-9185-46402838a950', 'ReceiptHandle': 'AQEBeQusZ8wS3XKoFaiwzLeNVRt+ZBmofXPM5YWSifzI1IdPuKJB3NAzb0U7qklTK…
2023-07-12T16:25:41.553-03:00    END RequestId: 0d255bd5-e0f7-462f-96fa-04a0e7703a6f
2023-07-12T16:25:41.553-03:00    REPORT RequestId: 0d255bd5-e0f7-462f-96fa-04a0e7703a6f Duration: 499.01 ms Billed Duration: 500 ms Memory Size: 128 MB Max Memory Used: 71 MB
```

Fig 25. Sample "fulfillTaxiRequest" Execution Log [5]

What's more, notice in Figure 25 that there is indeed about a 2-minute delay between the start of each new request.

Thus, we have verified that our "orderTaxi" Lambda [3] correctly publishes to "HalifaxTaxi" SNS Topic [2] and that the three SQS Queues [1] subscribed to that topic receive the messages. Additionally, we have verified that the second Lambda, "fulfillTaxiRequest", is invoked every 2 minutes and correctly polls the three SQS Queues [1] and publishes a new message to the "SendEmailNotification" SNS Topic [2]. Finally, we have verified that email notifications were correctly sent out with the expected content once "fulfillTaxiRequest" publishes an order message to the "SendEmailNotification" SNS Topic [2]

References:

[1] Amazon Web Services Inc., "Amazon Simple Queue Service," *Amazon Web Services Inc.* [Online], Available: https://aws.amazon.com/sqs/ [Accessed: July 13, 2023].

[2] Amazon Web Services Inc., "Amazon Simple Notification Service," *Amazon Web Services Inc.* [Online], Available: https://aws.amazon.com/sns/ [Accessed: July 13, 2023].

[3] Amazon Web Services Inc., "AWS Lambda," *Amazon Web Services Inc.* [Online], Available: https://aws.amazon.com/lambda/ [Accessed: July 13, 2023].

[4] Lucid Software Inc., "Lucidchart," *Lucid Software Inc.* [Online], Available: https://www.lucidchart.com/pages/ [Accessed: July 13, 2023].

[5] Amazon Web Services Inc., "Amazon CloudWatch," *Amazon Web Services Inc.* [Online], Available: https://aws.amazon.com/cloudwatch/ [Accessed: July 13, 2023]