

Use AWS Lambda-SQS-SNS for online Car delivery service.

Step 1: Login to AWS Creds and open SNS to create a Topic named Car_Order and select Standard option and click on create as shown in Fig 100, 101. And similarly, we will create another Topic named order_generation as shown in Fig 102.

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:

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.

Maximum 100 characters.

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

Fig 100

Topic: Car_Order created successfully
You can create subscriptions and send messages to them from this topic.

Amazon SNS > Topics > Car_Order

Car_Order

[Edit](#) [Delete](#) [Publish message](#)

Details

Name: Car_Order
ARN: arn:aws:sns:us-east-1:408121071803:Car_Order
Type: Standard
Display name: Car_Order
Topic owner: 408121071803

[Subscriptions](#) [Access policy](#) [Data protection policy](#) [Delivery policy \(SQS\)](#) [Delivery status logging](#) [Encryption](#) [Tags](#) [Integrations](#)

Subscriptions (0)

[Create subscription](#)

No subscriptions found
You don't have any subscriptions to this topic.

[Create subscription](#)

Fig 101

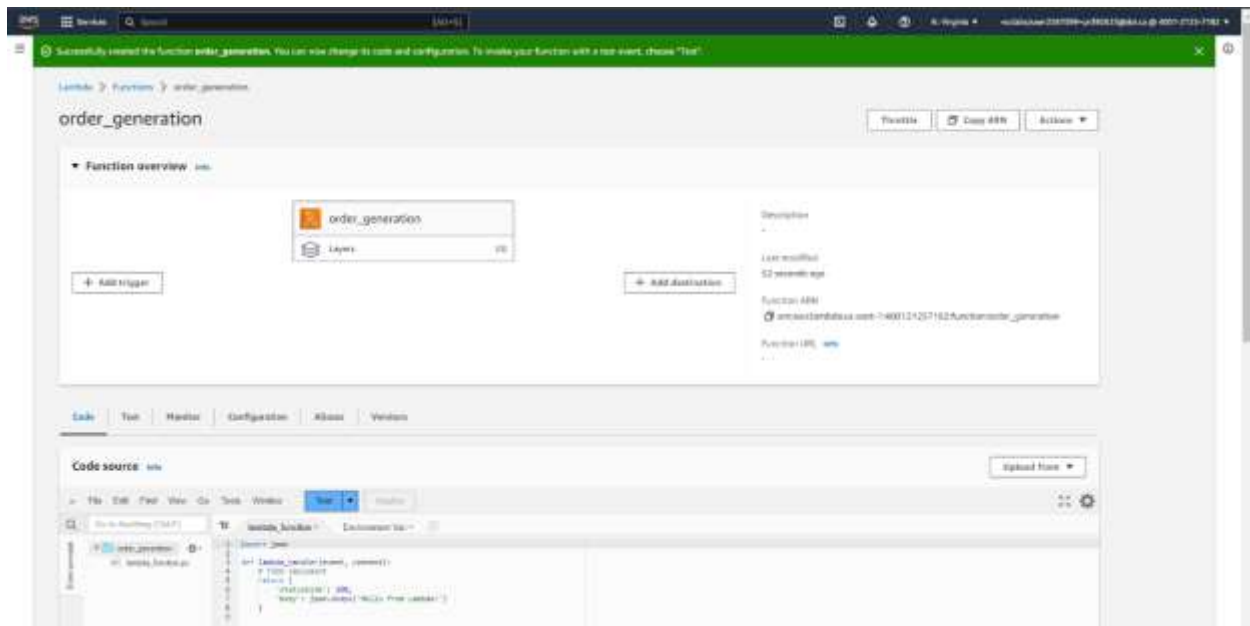


Fig 102

Step 2: Now we will go to SQS and create a Queue named Car_type, Car_Accessories and Client_address with standard option as shown in Fig 103, 104, 105, 106, 107, 108, 109.

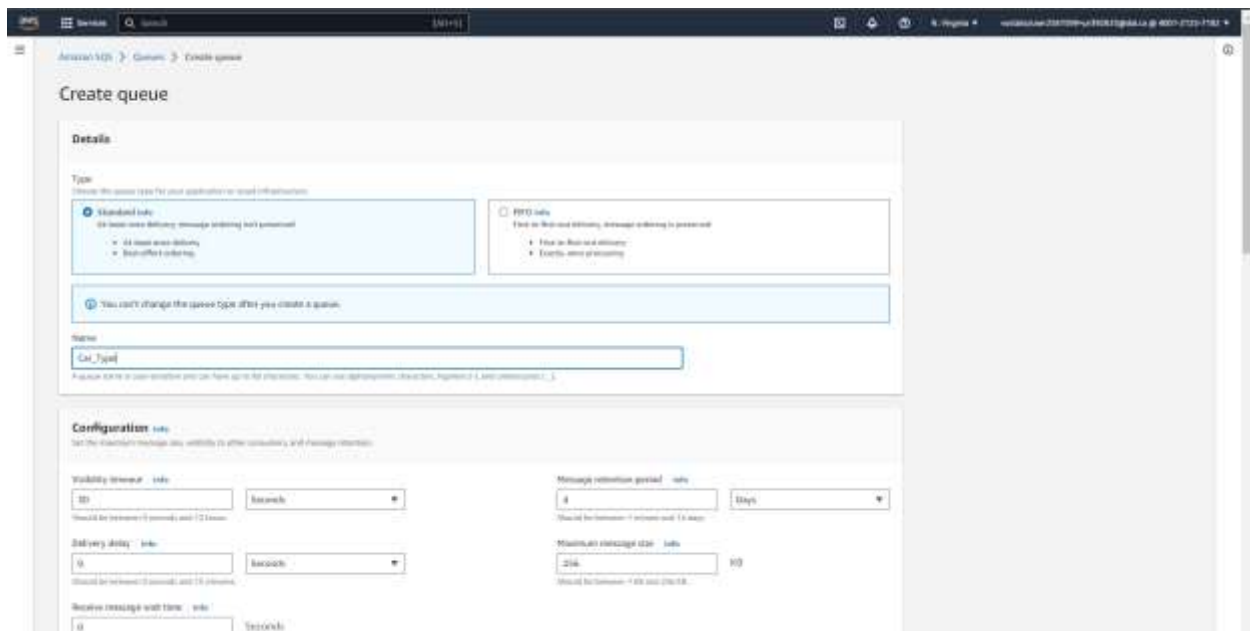


Fig 103

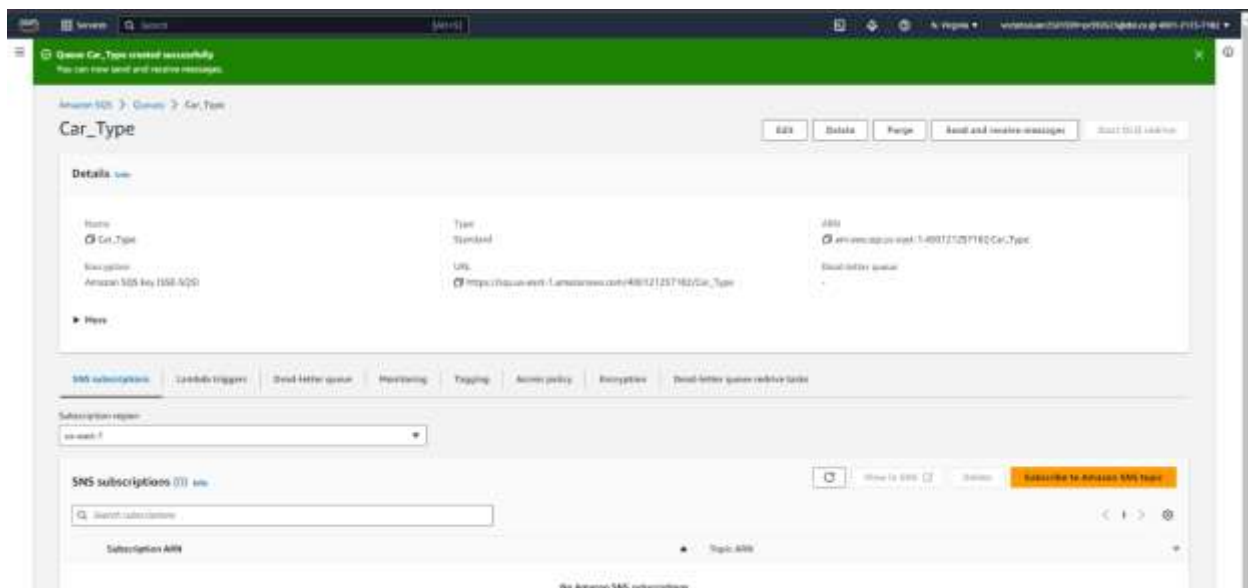


Fig 104

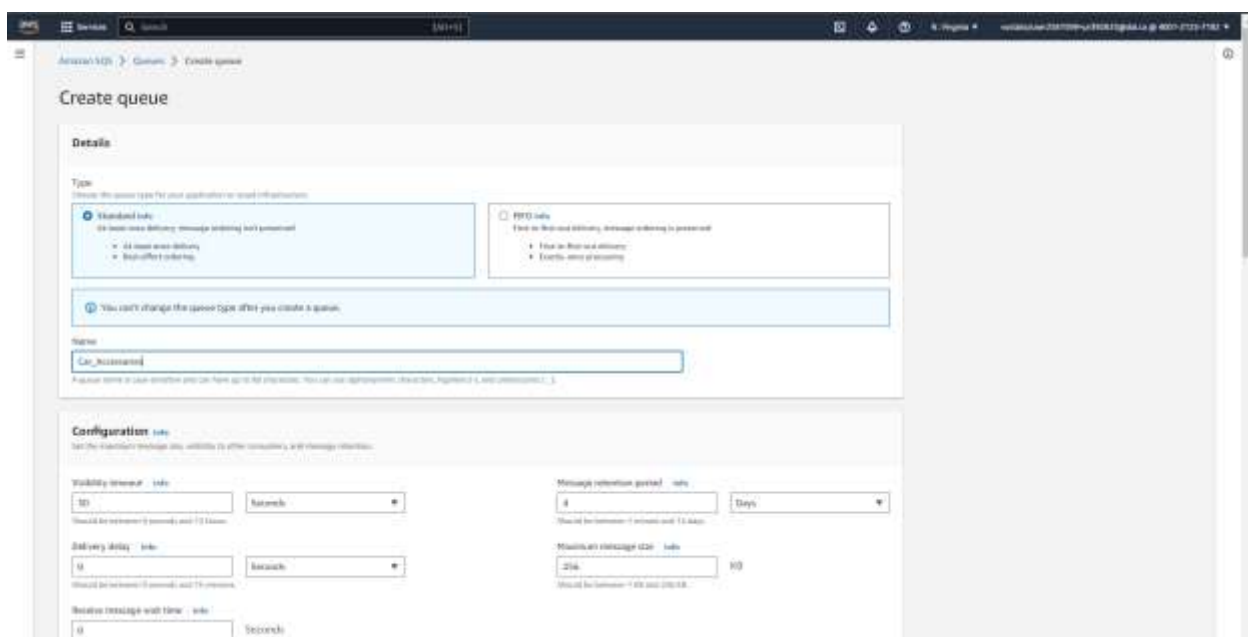


Fig 105

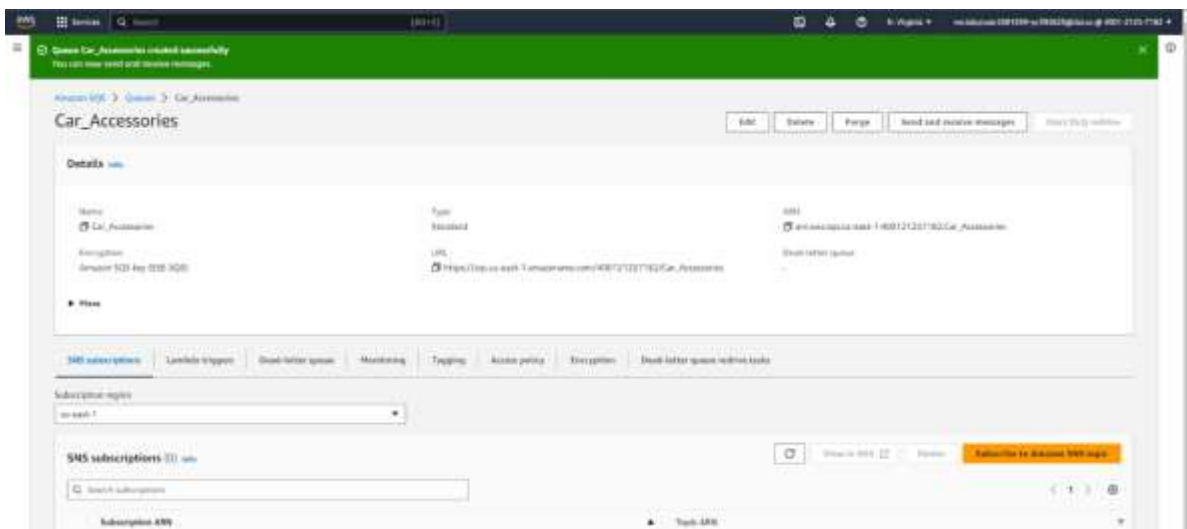


Fig 106

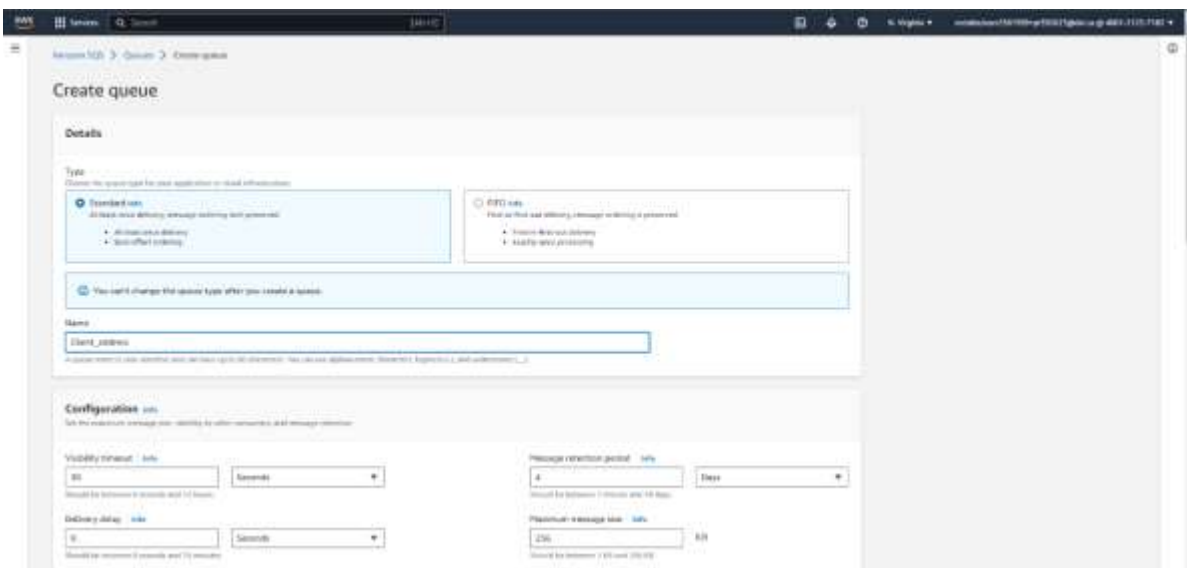


Fig 107

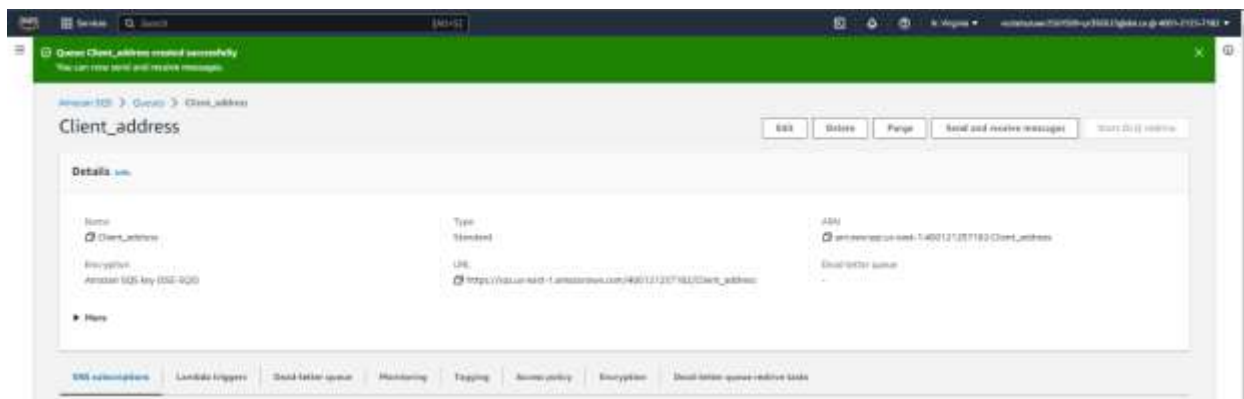


Fig 108



Fig 109

Step 3: Now we will go to Lambda functions and create functions named Car_Updates and create a Trigger with cloud watch events as shown in Fig 110, 111, 112, 113. Now we will add code to this lambda function as shown in Fig 114.

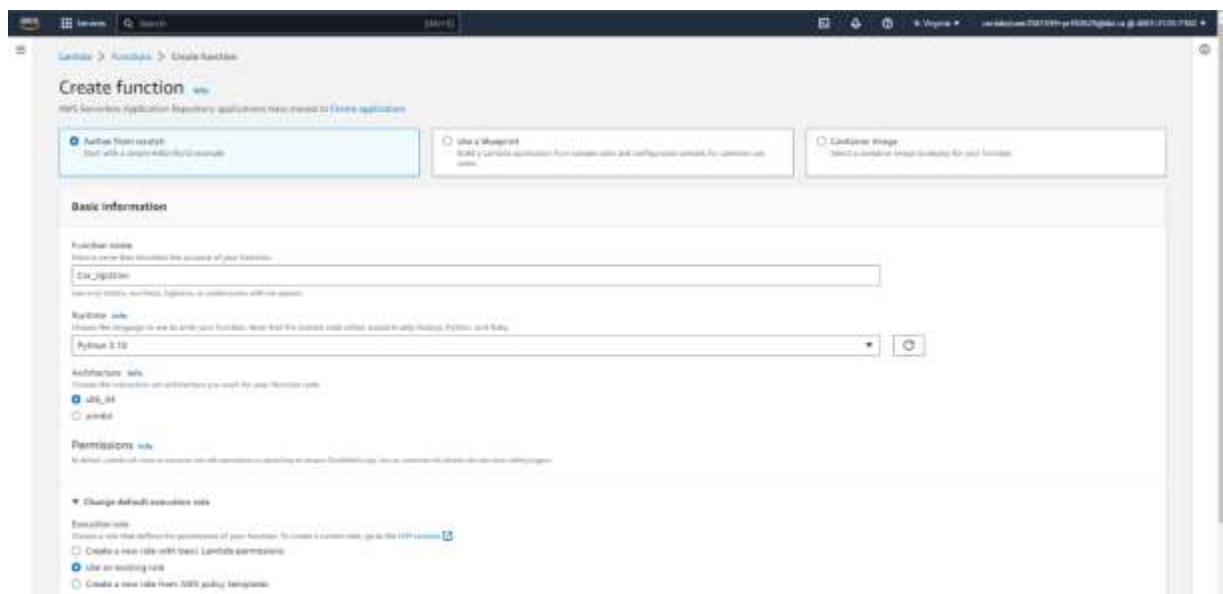


Fig 110

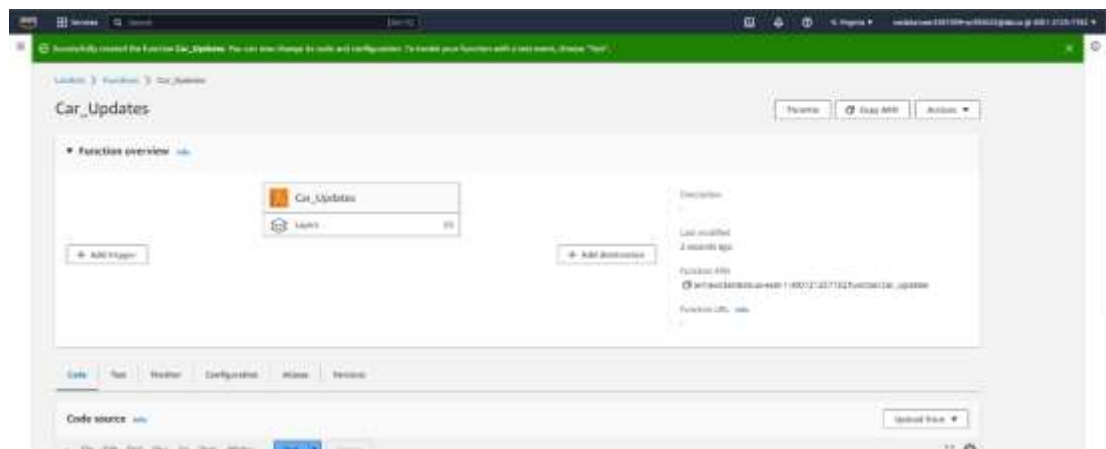


Fig 111

aws

Services


Search

[Alt+S]

Lambda > Add trigger

Add trigger

Trigger configuration [Info](#)

 EventBridge (CloudWatch Events)
asynchronous · aws · management-tools · schedule

Rule

Pick an existing rule, or create a new one.

☒ Create a new rule

☐ Existing rules

Rule name

Enter a name to uniquely identify your rule.

Orders

Rule description

Provide an optional description for your rule.

Rule type

Trigger your target based on an event pattern, or based on an automated schedule.

☐ Event pattern

☒ Schedule expression

Schedule expression

Self-trigger your target on an automated schedule using [Cron or rate expressions](#). Cron expressions are in UTC.

rate(2 minutes)

e.g. rate(1 day), cron(0 17 ? * MON-FRI *)

Lambda will add the necessary permissions for Amazon EventBridge (CloudWatch Events) to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

Cancel

Add

Fig 112

6 | Page

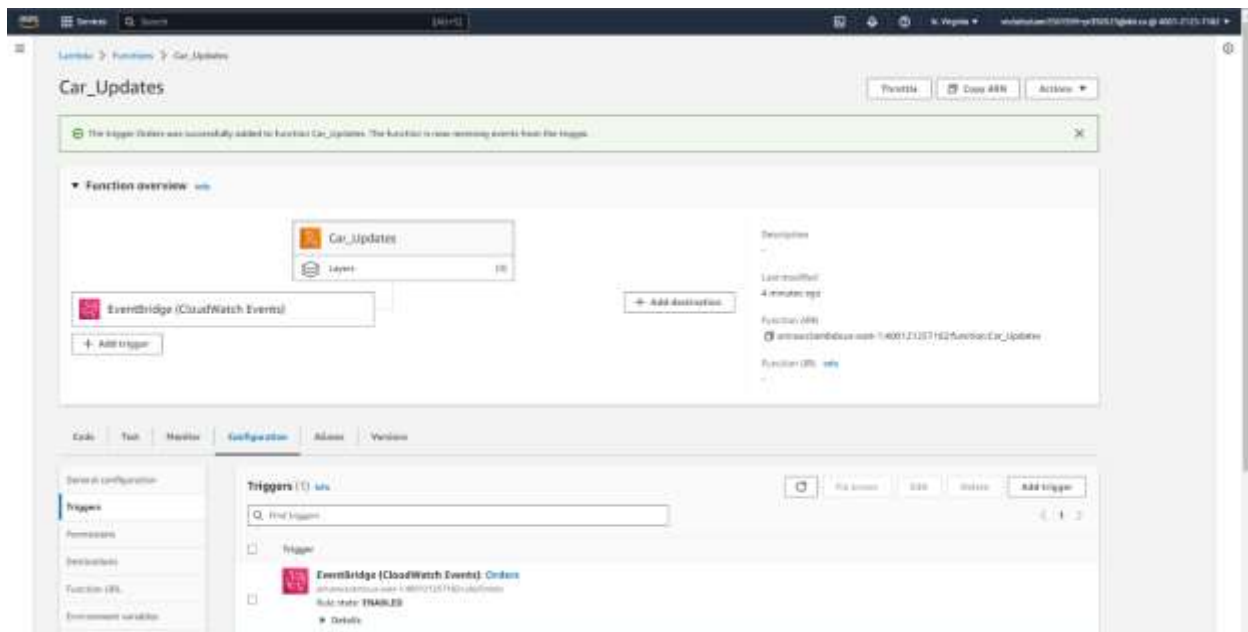


Fig 113

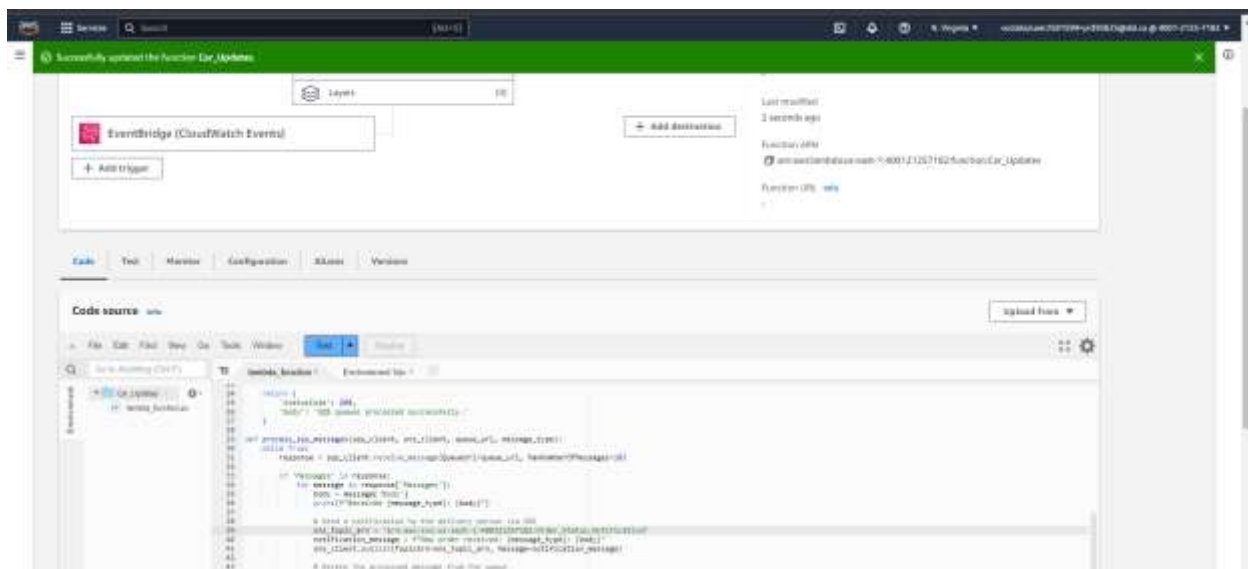


Fig 114

Code added:

import boto3

def lambda_handler(event, context):

Initialize SQS client
sqs_client = boto3.client('sqs')

Initialize SNS client
sns_client = boto3.client('sns')

```

# SQS queue URLs
car_type_queue_url = 'https://sqs.us-east-1.amazonaws.com/400121257182/Car_Type'
accessory_queue_url = 'https://sqs.us-east-
1.amazonaws.com/400121257182/Car_Accessories'
address_queue_url = 'https://sqs.us-east-1.amazonaws.com/400121257182/Client_address'

# Process car types
process_sqs_messages(sqs_client, sns_client, car_type_queue_url, "Car Type")

# Process accessories
process_sqs_messages(sqs_client, sns_client, accessory_queue_url, "Accessory")

# Process delivery addresses
process_sqs_messages(sqs_client, sns_client, address_queue_url, "Delivery Address")

return {
    'statusCode': 200,
    'body': 'SQS queues processed successfully.'
}

def process_sqs_messages(sqs_client, sns_client, queue_url, message_type):
    while True:
        response = sqs_client.receive_message(QueueUrl=queue_url,
MaxNumberOfMessages=10)

        if 'Messages' in response:
            for message in response['Messages']:
                body = message['Body']
                print(f"Received {message_type}: {body}")

                # Send a notification to the delivery person via SNS
                sns_topic_arn = 'arn:aws:sns:us-east-1:400121257182:Order_Status_Notification'
                notification_message = f"New order received! {message_type}: {body}"
                sns_client.publish(TopicArn=sns_topic_arn, Message=notification_message)

                # Delete the processed message from the queue
                sqs_client.delete_message(QueueUrl=queue_url,
ReceiptHandle=message['ReceiptHandle'])
            else:
                break

```

Step 4: Similarly, we will create another Lambda function named order_generation and with this we will add trigger to SNS Car_order that we created earlier. Also, we will add the code too in the process as shown in Fig 115, 116, 117.

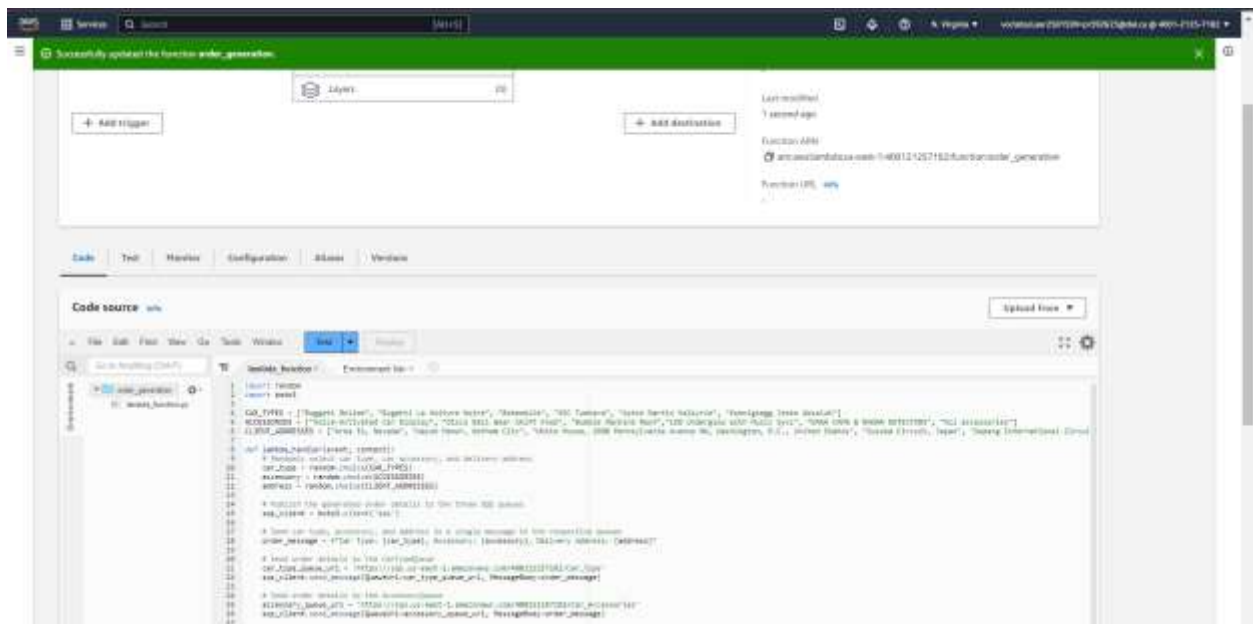


Fig 115

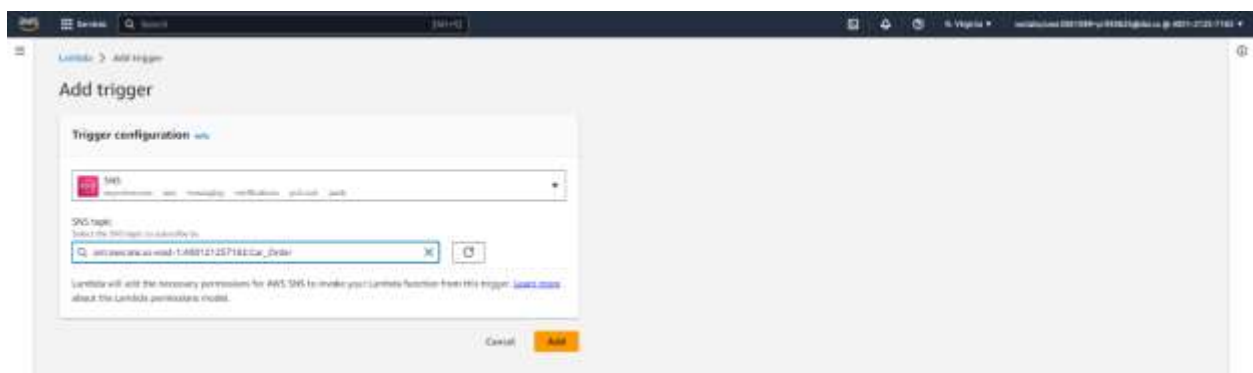


Fig 116

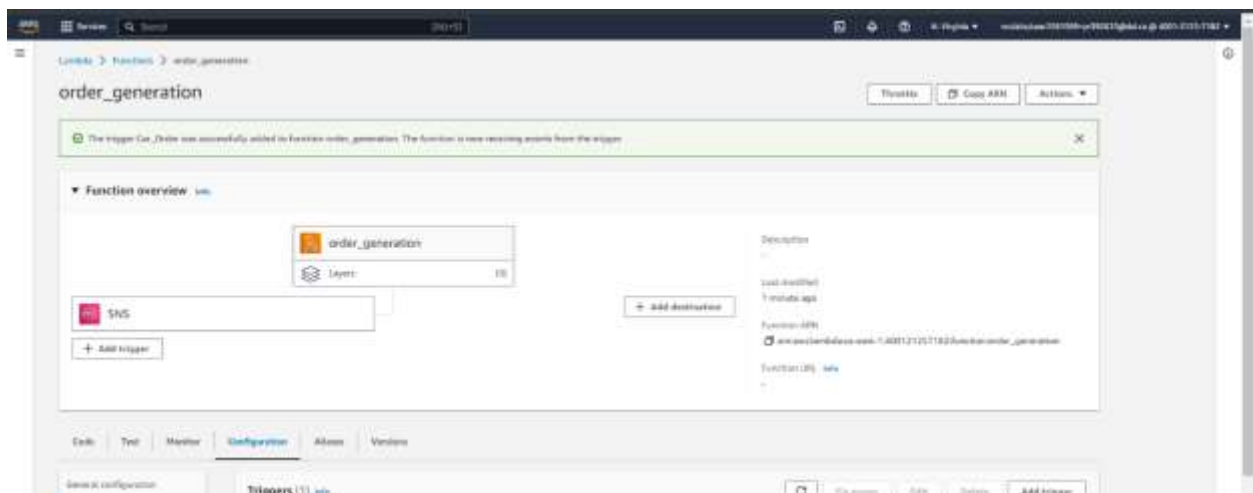


Fig 117

Code added:

Import necessary libraries

import random

import boto3

Define lists of car types, accessories, and client addresses

CAR_TYPES = ["Bugatti Bolide", "Bugatti La Voiture Noire", "Batmobile", "SSC Tuatara", "Aston Martin Valkyrie", "Koenigsegg Jesko Absolut"]

ACCESSORIES = ["Voice-Activated Car Display", "Disco Ball Gear Shift Knob", "Bubble Machine Roof", "LED Underglow with Music Sync", "DASH CAMS & RADAR

DETECTORS", "All accessories"]

CLIENT_ADDRESSES = ["Area 51, Nevada", "Wayne Manor, Gotham City", "White House, 1600 Pennsylvania Avenue NW, Washington, D.C., United States", "Suzuka Circuit, Japan", "Sepang International Circuit, Malaysia", "Circuit Gilles Villeneuve, Canada"]

def lambda_handler(event, context):

Randomly select car type, car accessory, and delivery address

car_type = random.choice(CAR_TYPES)

accessory = random.choice(ACCESSORIES)

address = random.choice(CLIENT_ADDRESSES)

Publish the generated order details to the three SQS queues

sqs_client = boto3.client('sqs')

Construct the order message with car type, accessory, and delivery address

order_message = f'Car Type: {car_type}, Accessory: {accessory}, Delivery Address: {address}'

Send order details to the CarTypeQueue

car_type_queue_url = 'https://sqs.us-east-1.amazonaws.com/400121257182/Car_Type'

sqs_client.send_message(QueueUrl=car_type_queue_url, MessageBody=order_message)

Send order details to the AccessoryQueue

accessory_queue_url = 'https://sqs.us-east-1.amazonaws.com/400121257182/Car_Accessories'

sqs_client.send_message(QueueUrl=accessory_queue_url, MessageBody=order_message)

Send order details to the AddressQueue

```

address_queue_url = 'https://sqs.us-east-
1.amazonaws.com/400121257182/Client_address'
sns_client.send_message(QueueUrl=address_queue_url,
MessageBody=order_message)

# Publish the generated order details to the SNS topic (optional)
sns_client = boto3.client('sns')
sns_topic_arn = 'arn:aws:sns:us-east-1:400121257182:Car_Order'
sns_client.publish(TopicArn=sns_topic_arn, Message=order_message)

return {
    'statusCode': 200,
    'body': 'Order generated and sent to SQS queues.'
}

```

Step 5: Now we will create another SNS topic named Order_Status_Notification with a display name of Halifax Ultimate Car Services as shown in Fig 118, 119. Now we will click on create Subscription in which we will add an endpoint which is the email of client as shown in 120. This will also send a confirmation email to the client and after we click on confirm we will be subscribed to receive messages from the aws as shown in Fig 121, 122.

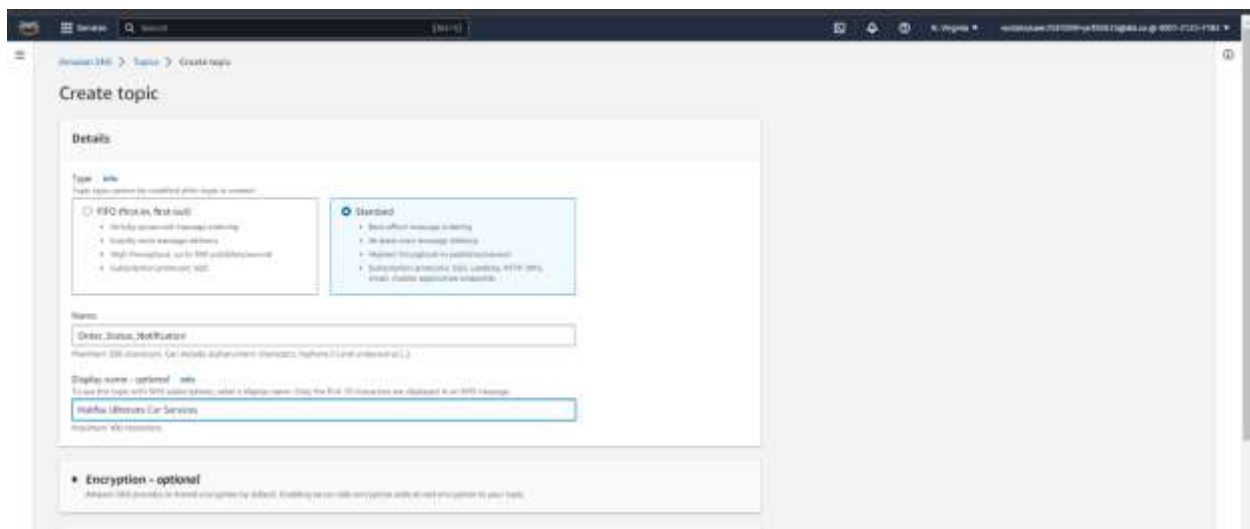


Fig 118

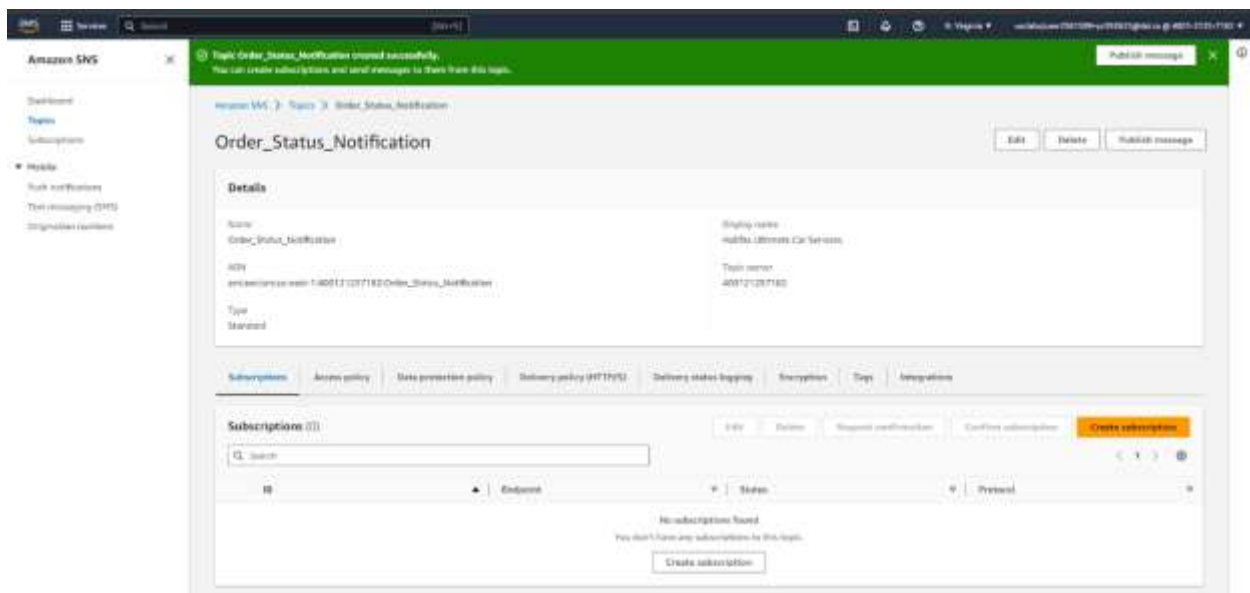


Fig 119



Fig 120

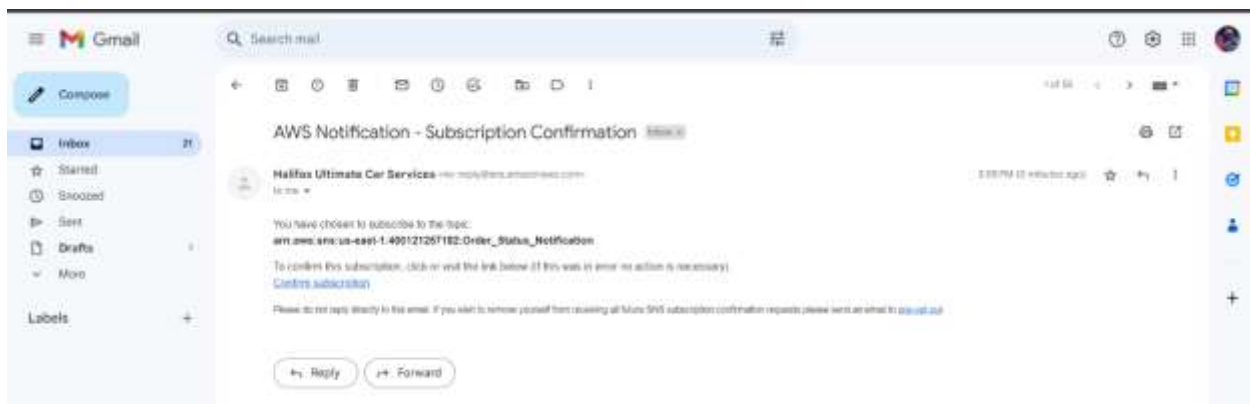


Fig 121

Configure test event

×

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, modify the event, then choose Test. Lambda uses the modified event to invoke your function, but does not overwrite the original event until you choose Save changes.

Test event action

☐ Create new event
 ☒ Edit saved event

Event name

Event JSON

Format JSON

```

3  {
4    "EventSource": "aws:sns",
5    "EventVersion": "1.0",
6    "EventSubscriptionArn": "arn:aws:sns:us-east-1:{{accountId}}:ExampleTopic",
7    "Sns": {
8      "Type": "Notification",
9      "MessageId": "85d981b4-e088-5cb0-9003-4c221d41eb5e",
10     "TopicArn": "arn:aws:sns:us-east-1:40012357182:Car_Order",
11     "Subject": "example subject",
12     "Message": "example message",
13     "Timestamp": "1970-01-01T00:00:00.000Z",
14     "SignatureVersion": "1",
15     "Signature": "EXAMPLE",
16     "SigningCertUrl": "EXAMPLE",
17     "UnsubscribeUrl": "EXAMPLE",
18     "MessageAttributes": {
19       "Test": {
20         "Type": "String",
21         "Value": "TestString"
22       },
23       "TestBinary": {
24         "Type": "Binary",
25         "Value": "TestBinary"
26       }
27     }
28   }
29 }
  
```

Fig 124

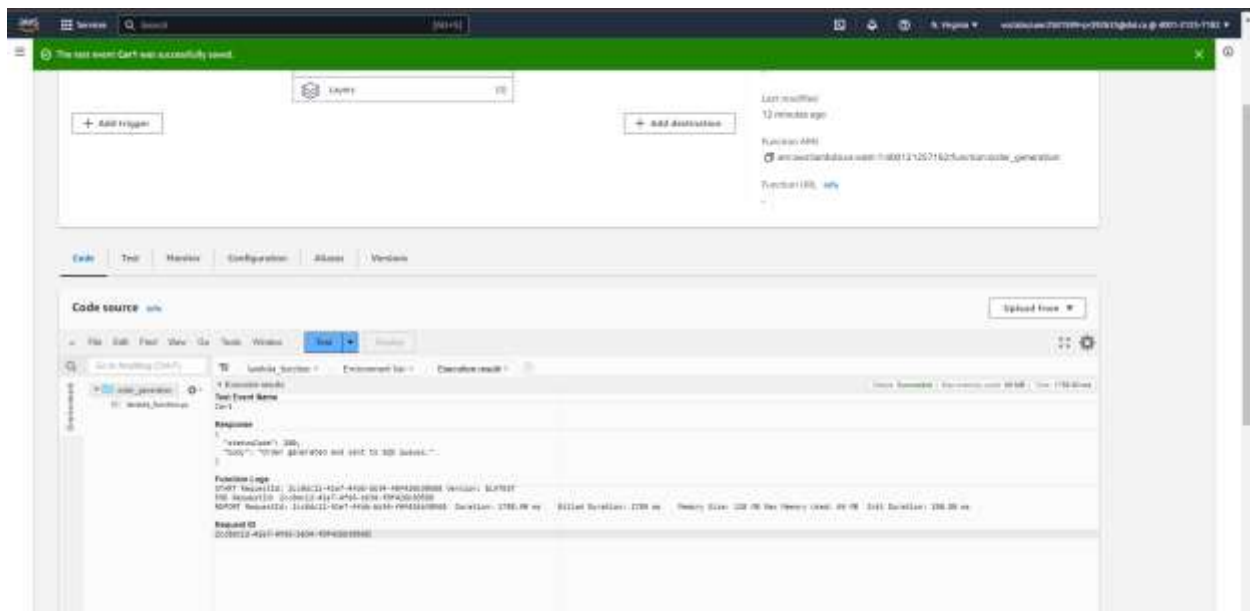


Fig 125

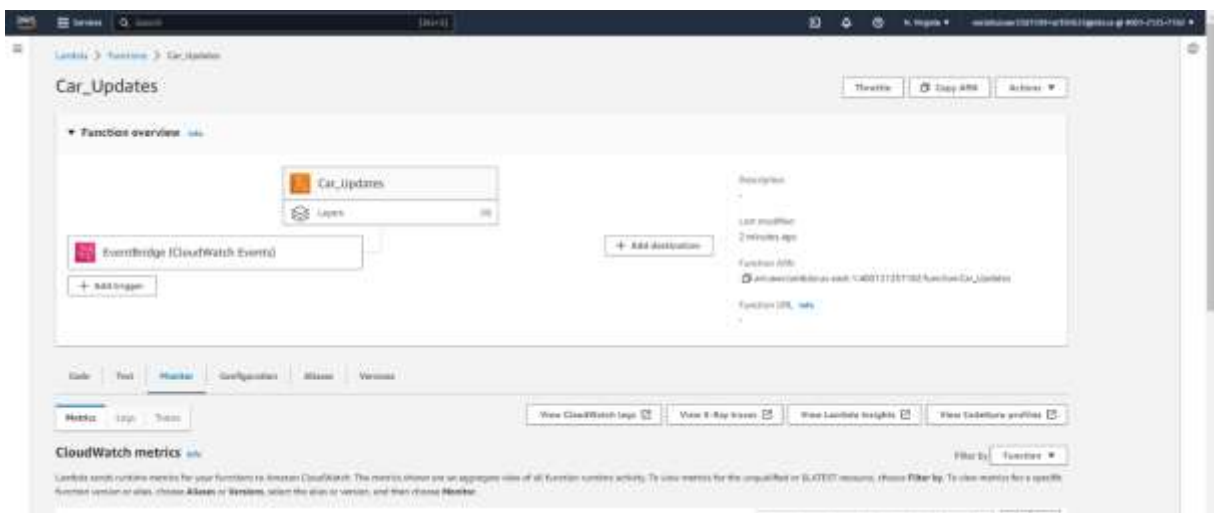


Fig 126

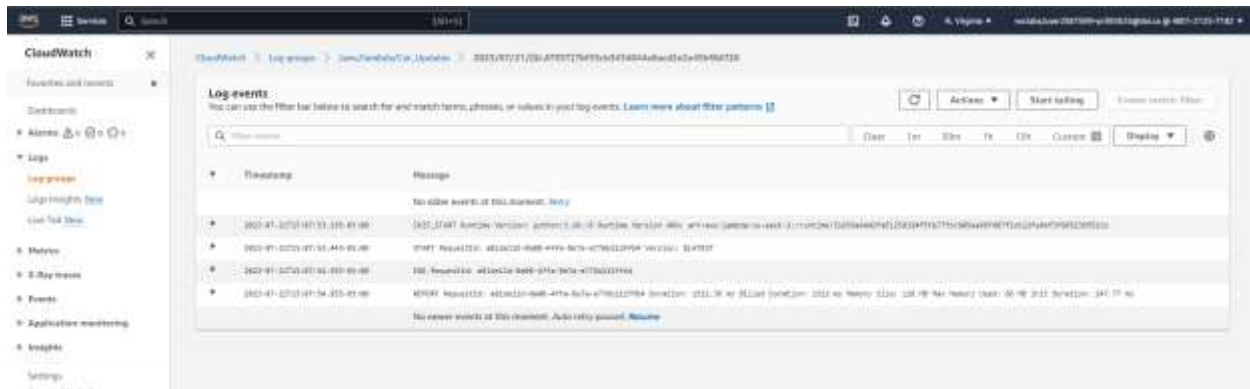


Fig 127

Step 7: We can verify that the notification has been sent by going to SQS which will show messages available 1 as shown in Fig 128. We can also see that we have received an email of the order as shown in Fig 129.

The screenshot shows the Amazon SQS console for a queue named 'Queue-1'. The queue is of type 'Standard' and was created on 2023-07-21T14:53:03.000. There are three messages in the queue, all of type 'Text'. The messages are listed with their IDs, types, and creation dates.

Name	Type	Created	Messages available	Messages in flight	Expiration	Current-based deduplication
Msg_1	Text	2023-07-21T14:53:03.000	1	0	Amazon SQS key (336-5Q3)	-
Msg_2	Text	2023-07-21T14:53:03.000	1	0	Amazon SQS key (336-5Q3)	-
Msg_3	Text	2023-07-21T14:53:03.000	1	0	Amazon SQS key (336-5Q3)	-

Fig 128

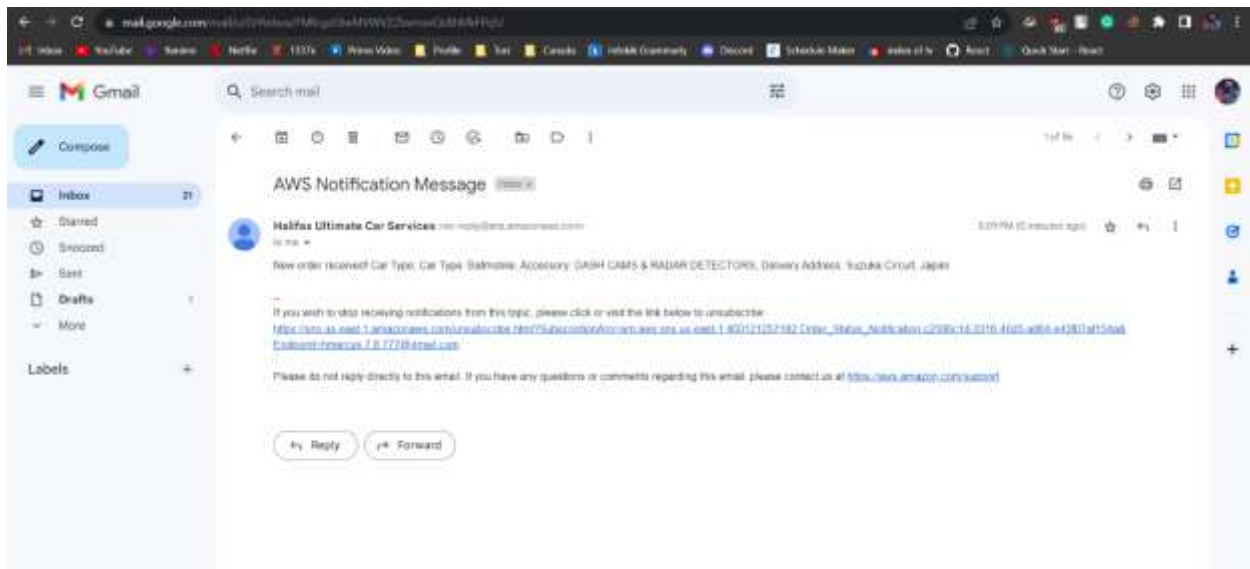


Fig 129

Step 8: I have sent a bunch of Tests like 5 and we can see that we have different results each time as shown in Fig 130, 131.

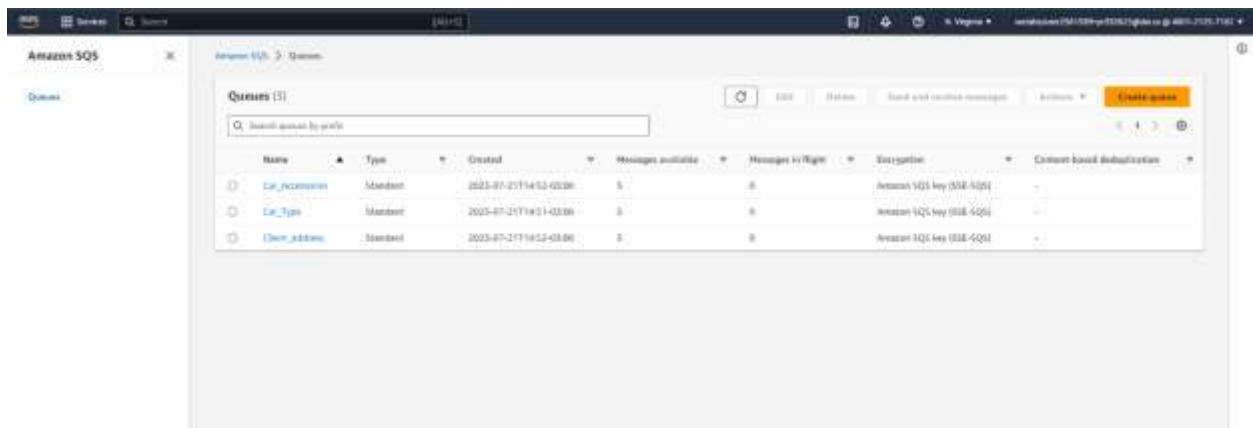


Fig 130

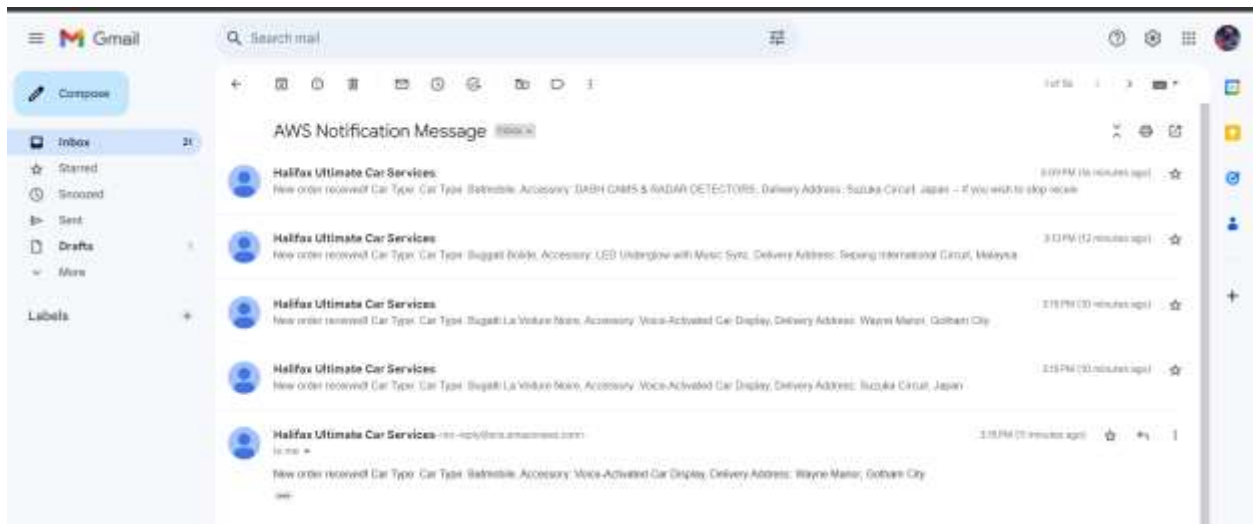


Fig 131