

Stock market Efficiency

Problem Statement:

Develop a pipeline where we hit NSE Stock Market API from RAPIDAPI website each minute & send data to AWS SQS. Once 10 records are accumulated in SQS, a Lambda function will be triggered to process the data & store it in Dynamo DB with current date & time.

Approach:

Following steps are describing the approach:

- Hit NSE Stock Market API every 1 min
- Store Stock Data for given company in SQS
- After 10 records are accumulated in SQS, trigger Lambda function to fetch the 10 records
- Store the current stock price in Dynamo DB

Technologies:

Python 3.9, Google Colab, Requests, boto3, AWS SQS, AWS Lambda, AWS DynamoDB.

Note:

- Google Colab is used since most libraries are pre-installed.

Code Explanation:

- Open 'Stock market Efficiency.ipynb' in Google Colab & run the 1st & 2nd cell.

```

  ▾ Stock Market Efficiency

  Develop a pipeline where we hit NSE Stock Market API from RAPIDAPI website each minute & send data to AWS SQS. Once 10 records are accumulated in SQS, a Lambda function will be triggered to process the data & store it in Dynamo DB with current date & time.

  ▾ Installing Boto Library

  [ ] ! pip install boto3

  ▾ Importing All Neccessary Libraries

  [ ] import requests
      import boto3
      import time
      import json

```

This will download boto3 library & import all the necessary libraries.

- Here in this cell we have to give our access key & secret access key to connect to AWS using boto3 client.

```

  ▾ Connect to AWS using Boto Client

  [ ] access_key="YOUR_ACCESS_KEY"
      secret_access_key="YOUR_SECRET_ACCESS_KEY"
      region = 'us-east-1'

      #Use boto3 client to connect with S3
      client = boto3.client(
          'sqs',
          aws_access_key_id=access_key,
          aws_secret_access_key=secret_access_key,
          region_name=region
      )

```

- This cell will create a Standard SQS Queue with name 'my-sqs-queue'.

```

  ▾ Create SQS Queue

  [ ] def create_queue():
      response = client.create_queue(
          QueueName="my-sqs-queue",
          Attributes={
              "DelaySeconds": "0",
              "VisibilityTimeout": "60",
          }
      )
      print(response)

```

- Get the Name of the stock whose data we need to store in DynamoDB.

```

  ▾ Getting the Name of stock

  [ ] def getName():
      ip = input("Please Enter the Stock Name")
      stock = {"symbol": ip}
      return stock

```

- This function will hit the NSE MARKET API for the mentioned stock name & get the stock data such as name, current_price etc in the json format.

▼ Fetching Stock Data from NSE Market API

```
[ ] def getData(stock):  
    url = "https://nse-market.p.rapidapi.com/stock_metrics"  
    headers = {  
        "x-rapidapi-key": "84948a2b71mshcfc40c32a46b15ep119225jsn279efff5e964",  
        "x-rapidapi-host": "nse-market.p.rapidapi.com"  
    }  
    res = requests.get(url, headers=headers, params=stock)  
    return res.json()
```

- This function will send the stock json data to SQS queue.

▼ Passing Data to AWS SQS

```
[ ] def sendMessage(data):  
    try:  
        queue = client.get_queue_url(QueueName='mytestsqssqs')  
        response = client.send_message(  
            QueueUrl=queue['QueueUrl'],  
            MessageBody=json.dumps(data)  
        )  
        print(response)  
    except Exception as e:  
        print("Error in sending message \n{}".format(e))  
    return response
```

- Finally main function which will invoke all function & run the code.

```
▼ Main Function

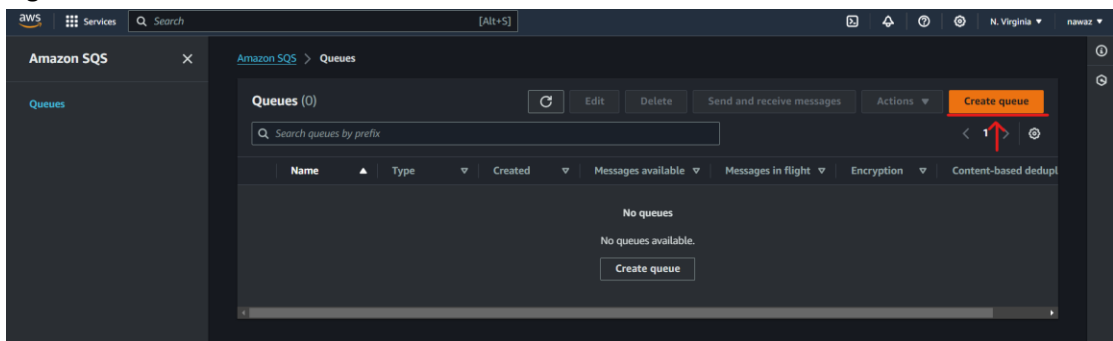
[ ] if __name__ == "__main__":
    name = getName()
    for i in range(10):
        time.sleep(30)
        data = getData(name)
        response = sendMessage(data)
        print(response)
```

Code Working:

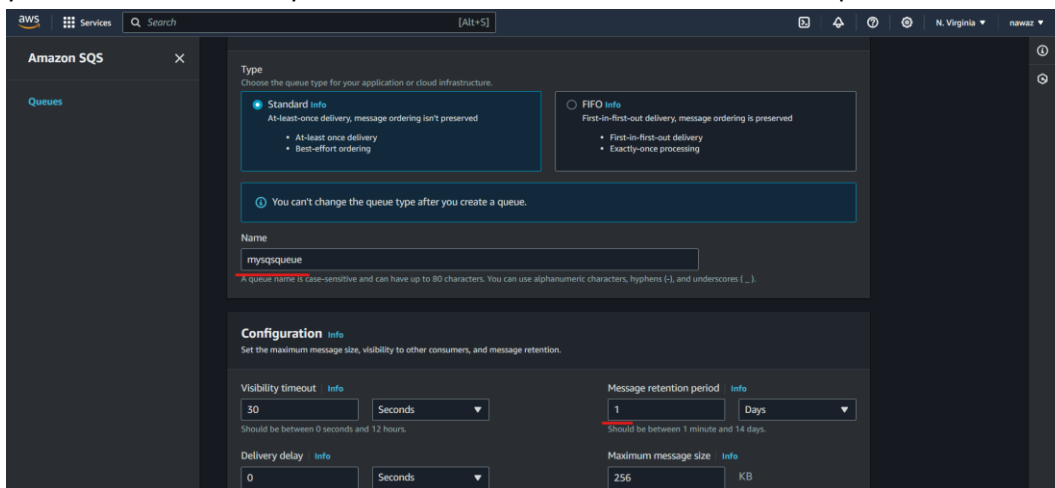
Before we run the code, we need to create a SQS queue, lambda function & DynamoDB table to store Data.

Step 1: Creating SQS queue

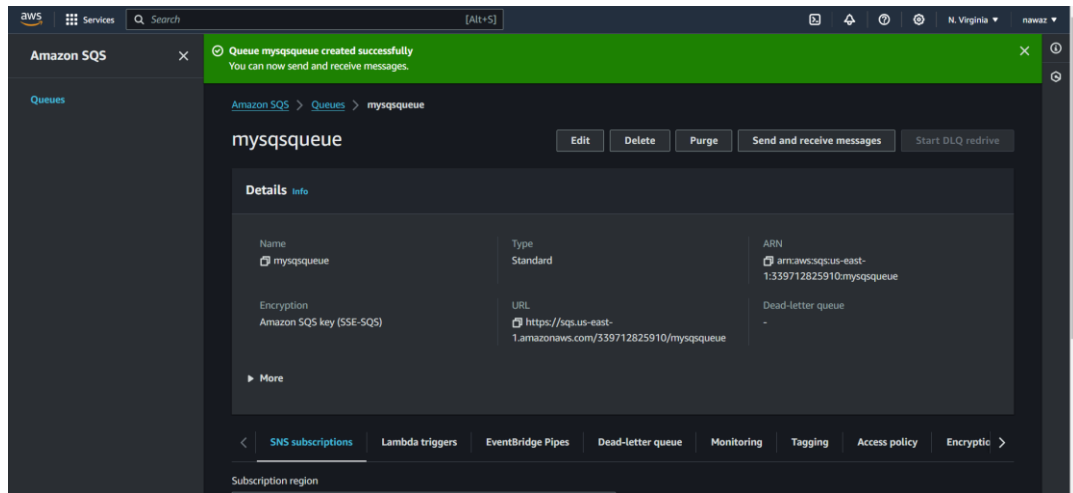
- Sign in to the AWS console & search SQS & click it. Now click on Create Queue.



- Give the Queue name whatever you want and maintain message retention period as 1 day or you can set default 4 days also. Then scroll to down & click on Create queue.

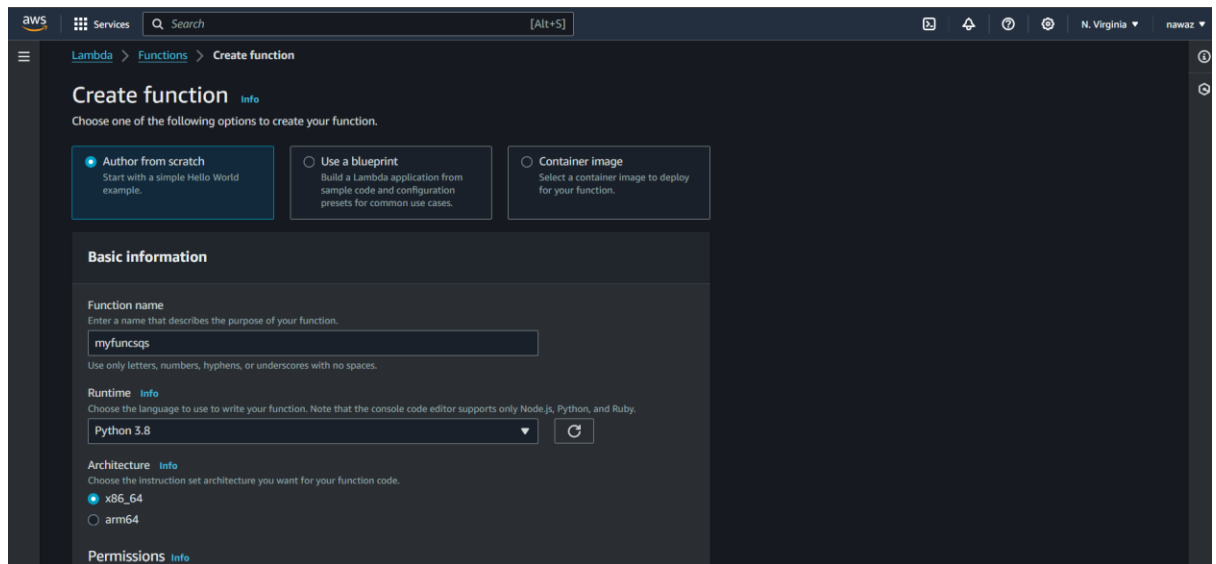


- Now our queue is up & running.



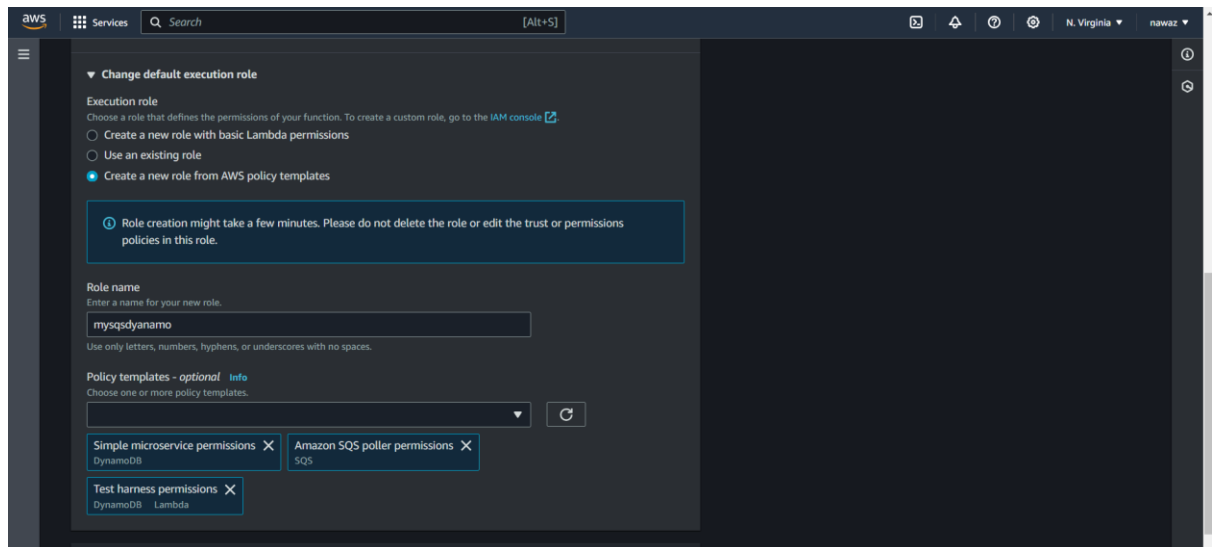
Step 2: Create Lambda Function & add a trigger that points to SQS queue.

- Search for Lambda in console & click on create function.



Give your function name & select the runtime. Below open the drag down 'Change default execution role'. Then select 'Create a new role from AWS policy templates'. You can role name & then for Policy add these 3:

- | | |
|------------------------------------|-------------------|
| 1. Amazon SQS poller permissions | SQS |
| 2. Simple microservice permissions | DynamoDB |
| 3. Test harness permissions | DynamoDB & Lambda |



Now click on create function.

Once function is created add the below code in the function & click o deploy.

```
import json

import boto3

from decimal import Decimal

from datetime import datetime

dydb = boto3.resource(service_name = 'dynamodb')

table = 'stocktest'

table = dydb.Table(table)

def lambda_handler(event, context):

    print(event)

    records = event['Records']

    lx = []

    now = datetime.now()

    now = now.strftime("%d_%m_%Y_%H_%M_%S")

    for rec in records:

        x = rec['body']
```

```

y = json.loads(x)

lx.append(y['currentPrice'])

price = lx[::-1][0]

data = {'now':str(now),

'price' : price}

jsondata = json.loads(json.dumps(data), parse_float=Decimal)

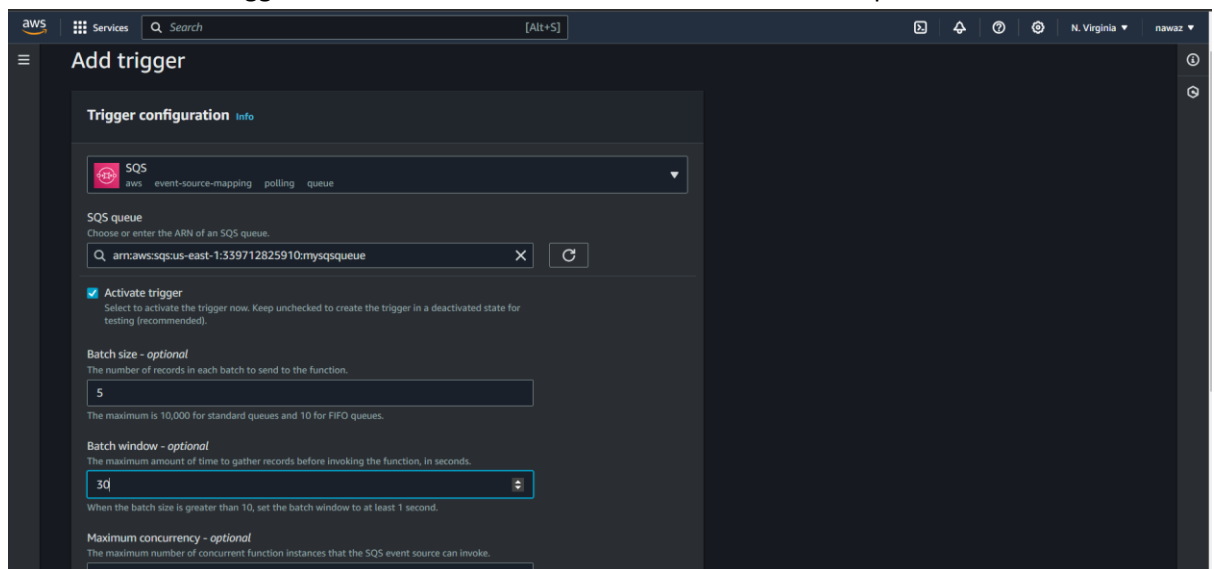

print(type(jsondata))

print(jsondata)


table.put_item(Item = jsondata)

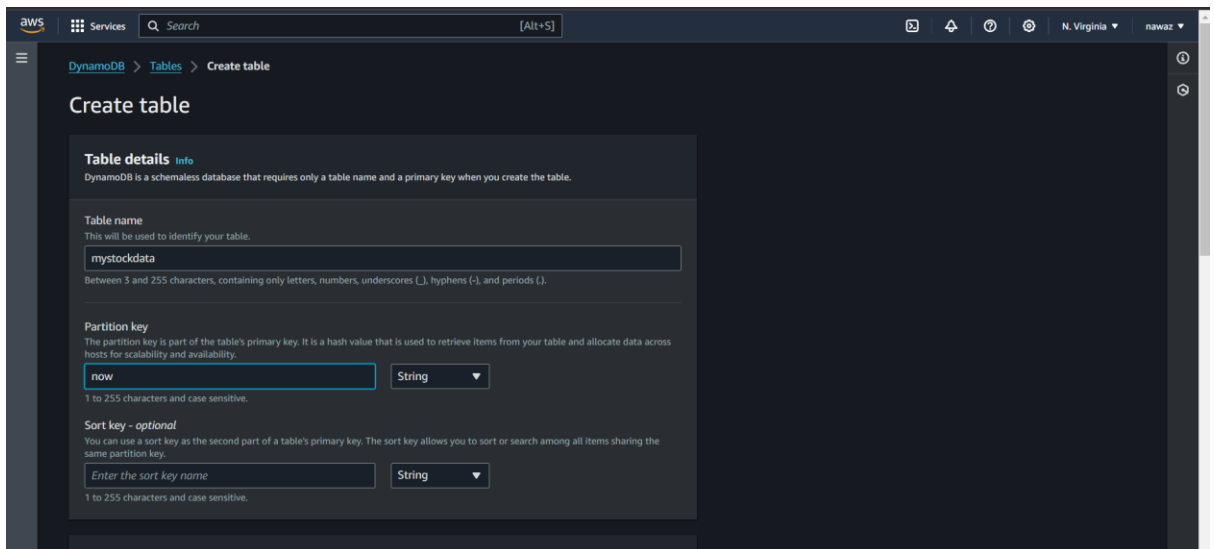
```

- Now click on add trigger and select SQS and then select the our created queue.

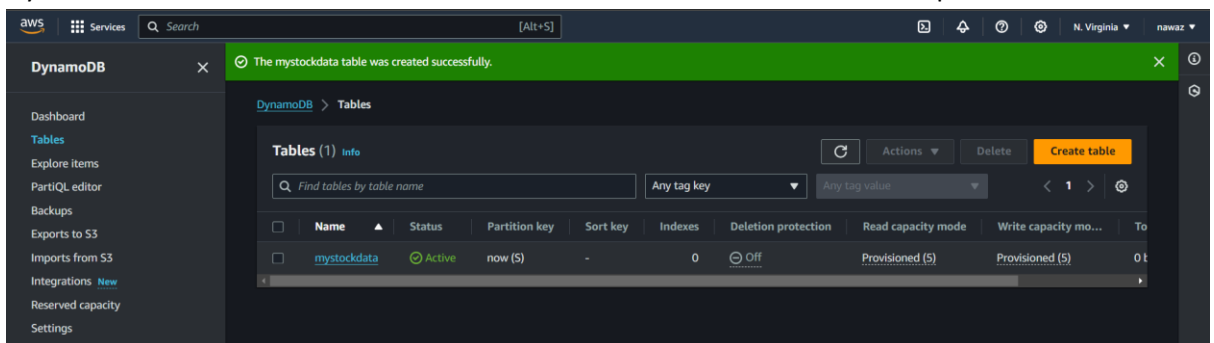


Give batch size as 10 but as of now batch size is given as 5 for testing & Give Batch Window as 60 sec where the lambda will check the queue every 60 seconds. But here for testing I have taken it as 30 sec.

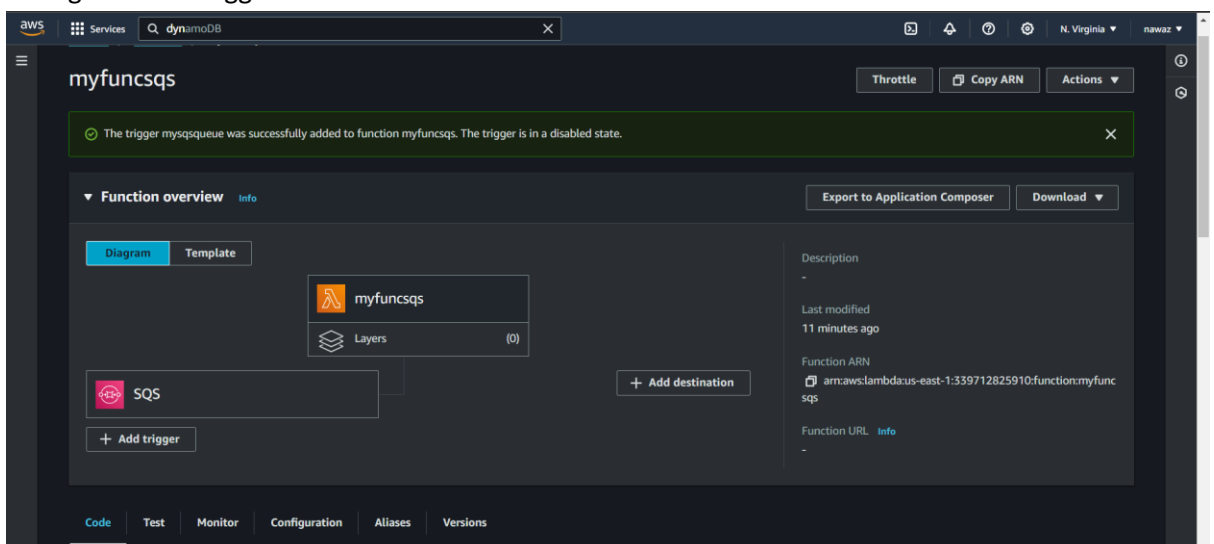
- Before clicking on add trigger, In AWS console search DynamoDB & click on create table.



Give name & partition key here given as 'now', this is current date in json data passed to DynamoDB from lambda. Next click on create table. It will take some time to up and run.



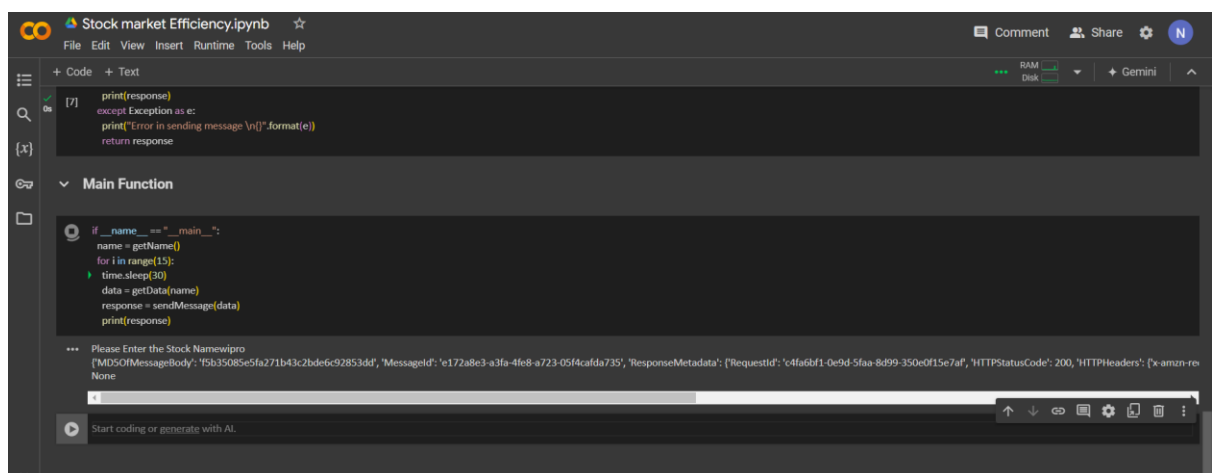
Now go back to trigger and create it.



Now trigger is added to the function. We just need to go back to code and run it.

Running the code:

Now run all the cells in the code.



The screenshot shows a Jupyter Notebook titled "Stock market Efficiency.ipynb". The interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a toolbar with icons for adding code or text, and a sidebar with navigation icons. The main area displays two code cells. The first cell, labeled [7], contains a try-except block for handling exceptions during message sending. The second cell, titled "Main Function", contains a loop that sends 15 messages, each with a 30-second delay. Below the code, the output of the "Main Function" cell is visible, showing a detailed JSON response from the AWS SDK, including fields like "RequestId", "HTTPStatusCode", and "ResponseMetadata". At the bottom, there is a prompt to "Start coding or generate with AI".

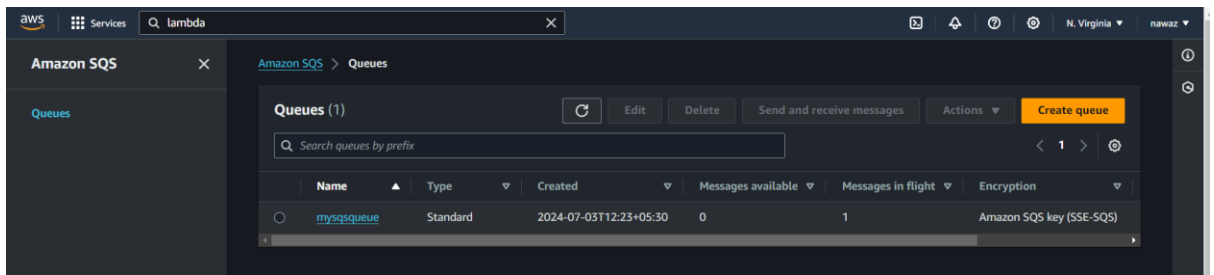
```
[7] print(response)
    except Exception as e:
        print("Error in sending message {}".format(e))
    return response
```

```
def main():
    if __name__ == "__main__":
        name = getName()
        for i in range(15):
            time.sleep(30)
            data = getData(name)
            response = sendMessage(data)
            print(response)
```

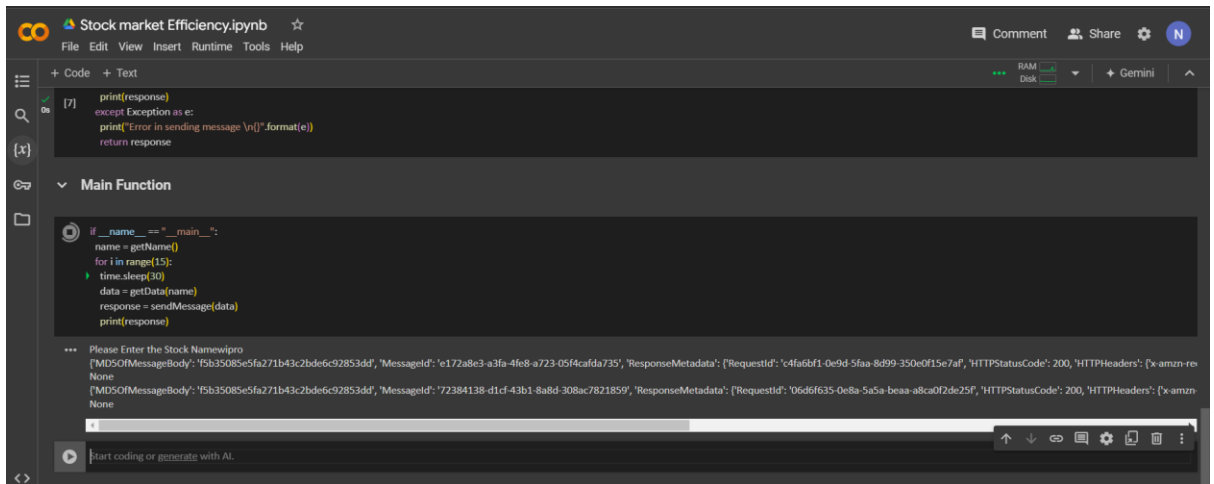
```
{
  "Request": {
    "Name": "AAPL",
    "Message": "AAPL stock price is 150.00"
  },
  "Response": {
    "MessageId": "e172a8e3-a3fa-4fe8-a723-05f4cdda735",
    "ResponseMetadata": {
      "RequestId": "c4fa6bf1-0e9d-5faa-8d99-350c0f15e7af",
      "HTTPStatusCode": 200,
      "HTTPHeaders": {
        "x-amzn-req-id": "c4fa6bf1-0e9d-5faa-8d99-350c0f15e7af"
      }
    }
  }
}
```

Start coding or generate with AI.

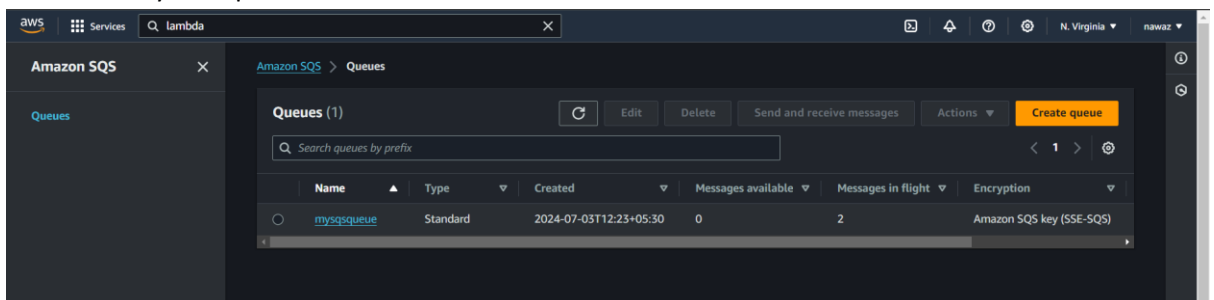
We can see in the code that 1 message is sent to SQS and we can verify it in queue.



Similarly another message is triggered.



We can verify it in queue.



Once 5 messages are triggered,

```
Stock market Efficiency.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[7] print(response)
    except Exception as e:
    print("Error in sending message {}".format(e))
    return response

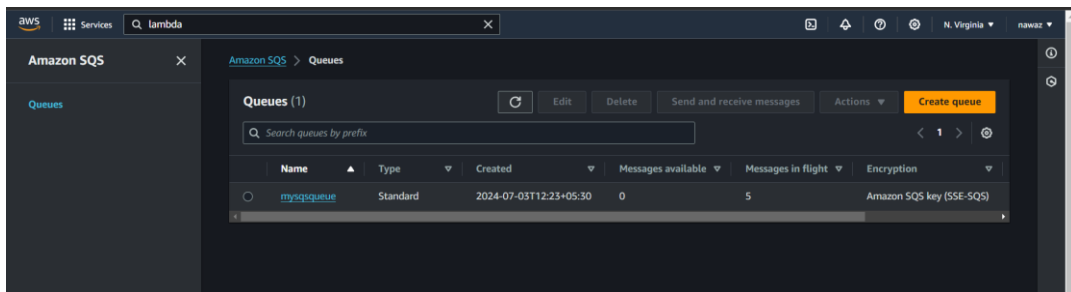
▼ Main Function

if __name__ == "__main__":
    name = getName()
    for i in range(15):
        time.sleep(30)
        data = getData(name)
        response = sendMessage(data)
        print(response)

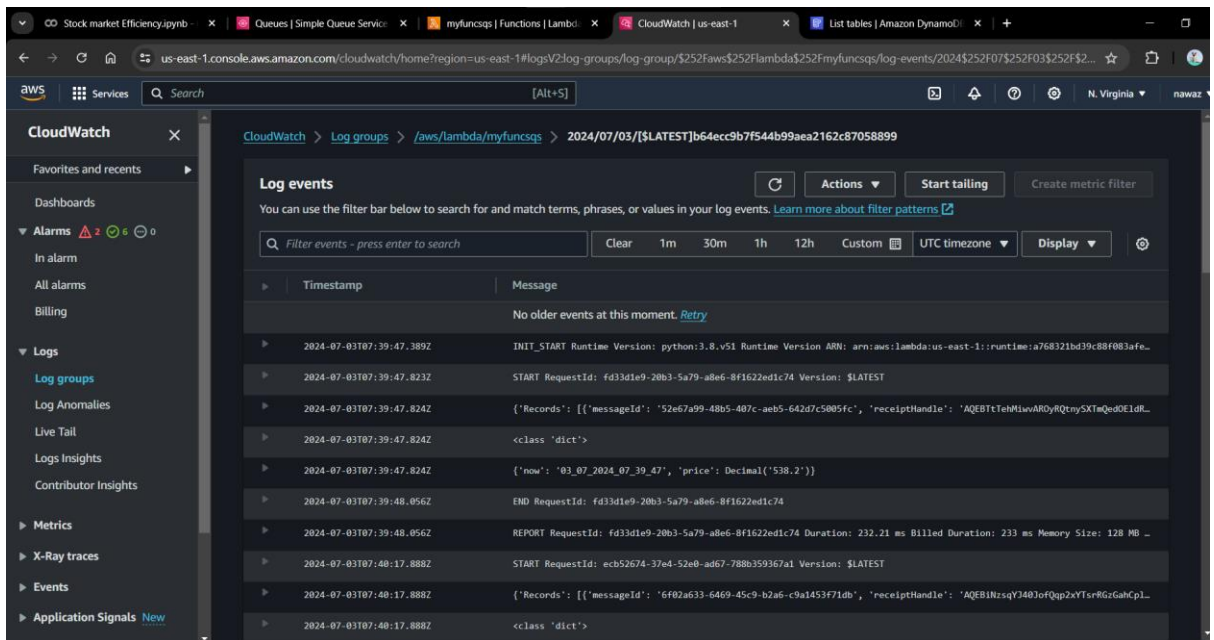
... Please Enter the Stock Name/wipro
{"[\"MSOFSMessageBody\": \"5b3508Se5fa271b43c2bde6c92853dd\", \"MessageId\": \"e172a8e3-a3fa-4fe8-a723-05f4cfdaf735\", \"ResponseMetadata\": {\"RequestId\": \"cf466bf1-0e9d-5faa-8d99-350e0f15e7af\", \"HTTPStatusCode\": 200, \"HTTPHeaders\": {\"x-amzn-re\"
None
\"[\"MSOFSMessageBody\": \"5b3508Se5fa271b43c2bde6c92853dd\", \"MessageId\": \"72384138-d1cf-43b1-8a8d-308ac7821859\", \"ResponseMetadata\": {\"RequestId\": \"06d6f535-0e8a-5a5a-beaa-a8caf02de25f\", \"HTTPStatusCode\": 200, \"HTTPHeaders\": {\"x-amzn\"
None
\"[\"MSOFSMessageBody\": \"5b3508Se5fa271b43c2bde6c92853dd\", \"MessageId\": \"f502260-4e57-43bb-b822-d86b98aea2b1\", \"ResponseMetadata\": {\"RequestId\": \"409a52d6-33ac-5c6d-adaf-0c29326a1468\", \"HTTPStatusCode\": 200, \"HTTPHeaders\": {\"x-amz\"
None
\"[\"MSOFSMessageBody\": \"5b3508Se5fa271b43c2bde6c92853dd\", \"MessageId\": \"52e67a99-48b5-407c-ae5b-642d7c5005fc\", \"ResponseMetadata\": {\"RequestId\": \"1be9ac55-6a46-5411-bee1-f20a6ba01c14\", \"HTTPStatusCode\": 200, \"HTTPHeaders\": {\"x-amzn\"
None
\"[\"MSOFSMessageBody\": \"5b3508Se5fa271b43c2bde6c92853dd\", \"MessageId\": \"b02a633-6469-45c9-b2af-c9a1453f71db\", \"ResponseMetadata\": {\"RequestId\": \"51c280e1-aeed-577d-bbed-de1eace8f878\", \"HTTPStatusCode\": 200, \"HTTPHeaders\": {\"x-amzn\"
None

Start coding or generate with AI.
```

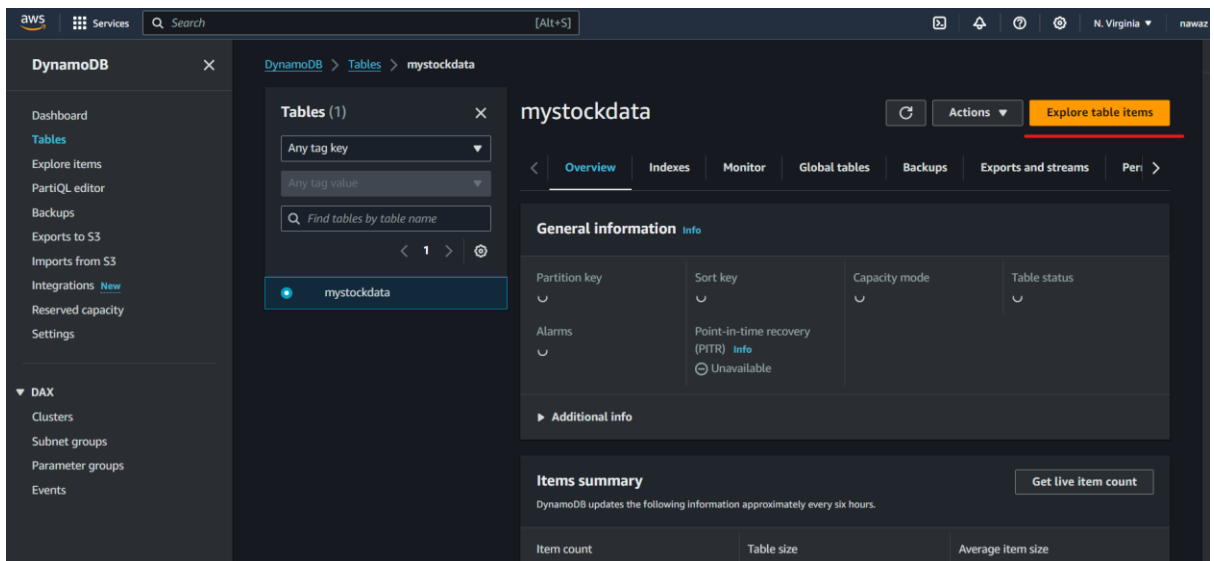
We can verify it in queue and lambda is triggered.



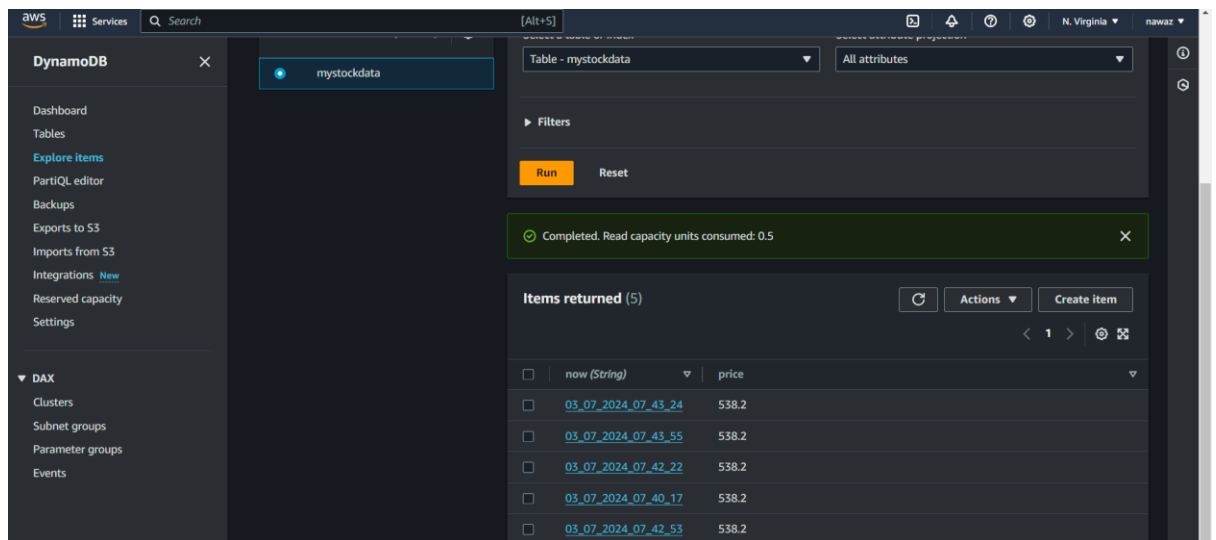
We can verify this in opening lambda & going into monitoring tab & clicking on View Cloud watch. We can see log is generated and data is inserted into DynamoDB.



Go to DynamoDB, click on the table which we created and click on Explore Table Items.



We can see that data is inserted into the table with timestamp.



We tweak the lambda trigger and fetch for 10 records also.