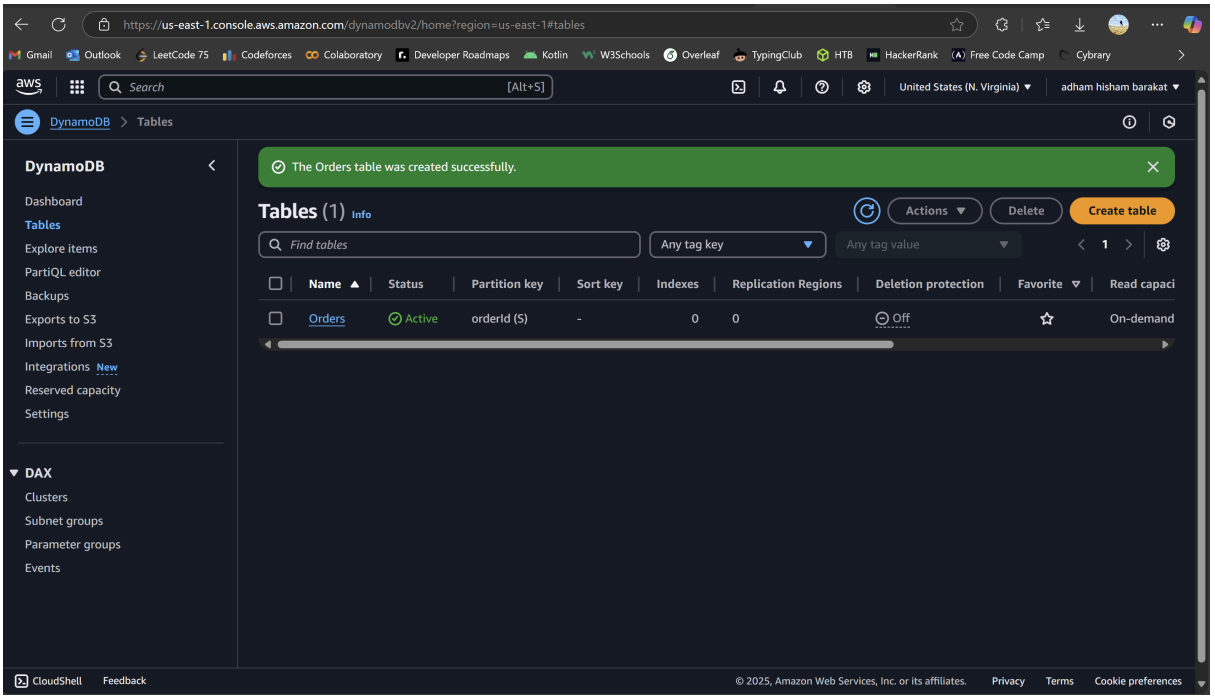
AWS Assignment 2 Screenshots

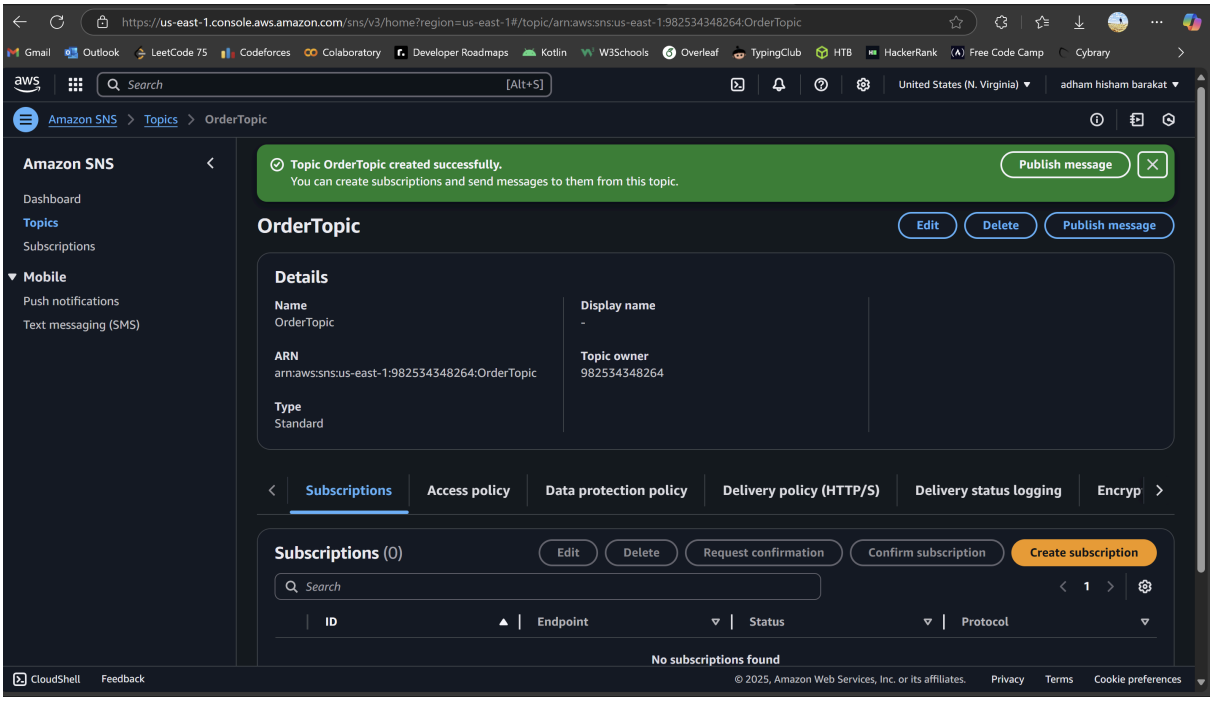
Adham Hisham Salah Hassan

10000480 | T-20

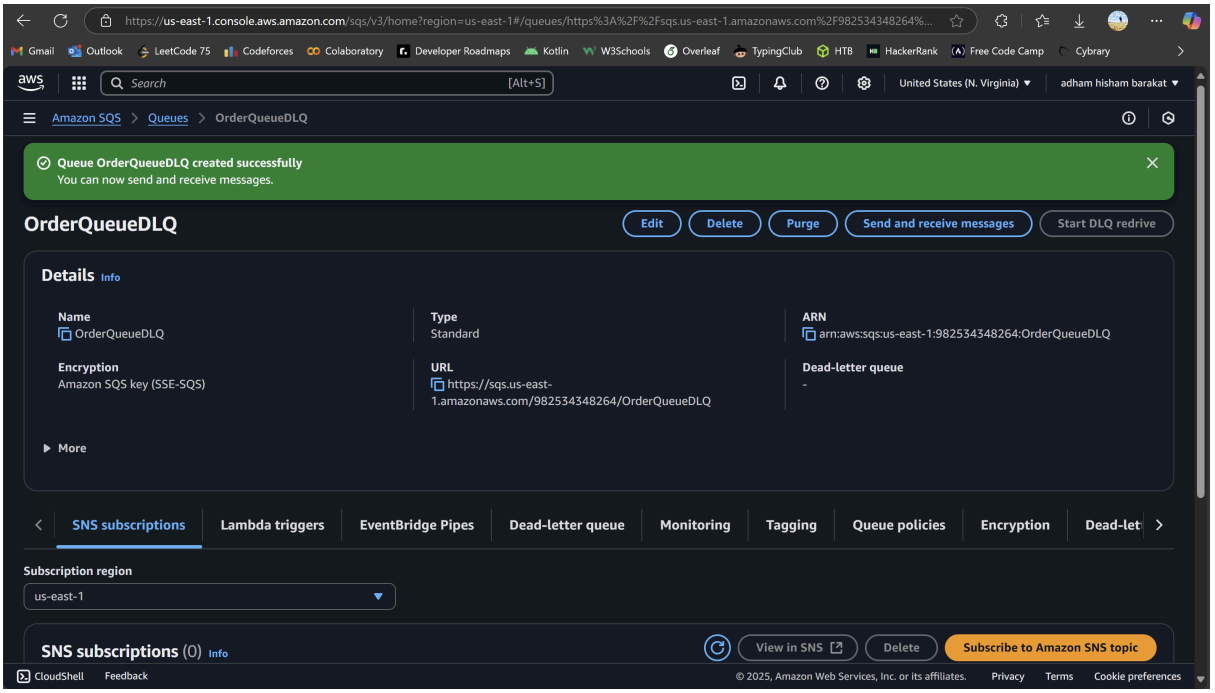
Created a DynamoDB table called orders:



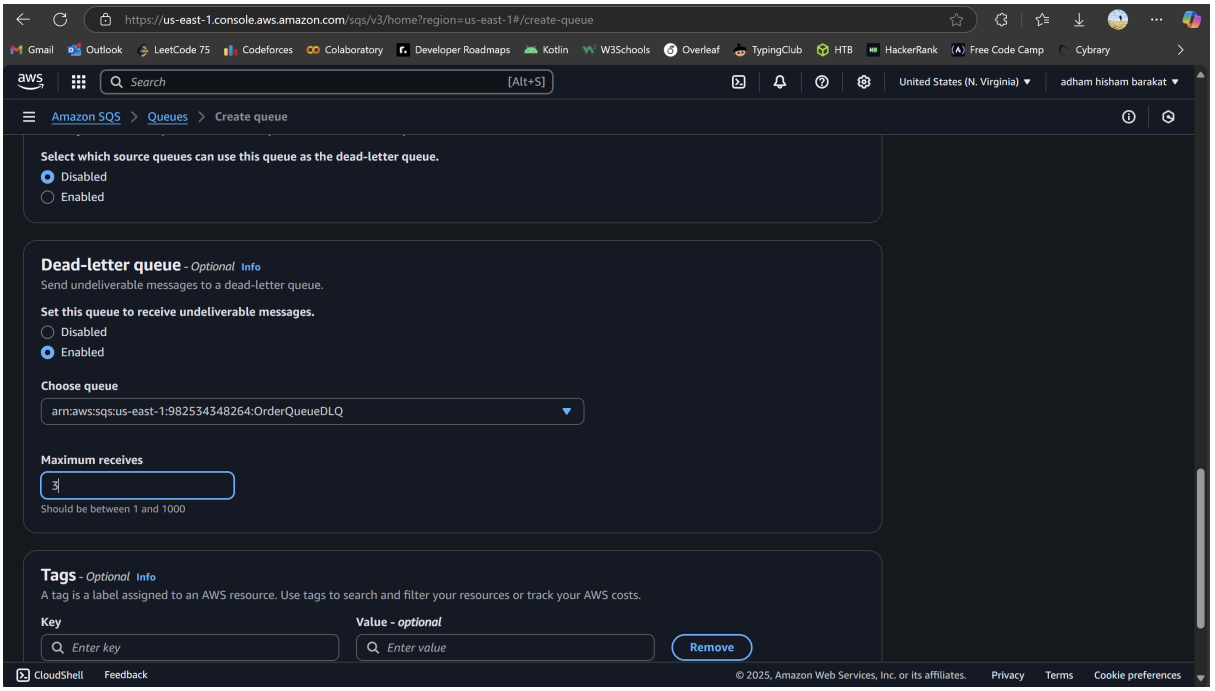
created an sns called OrderTopic:

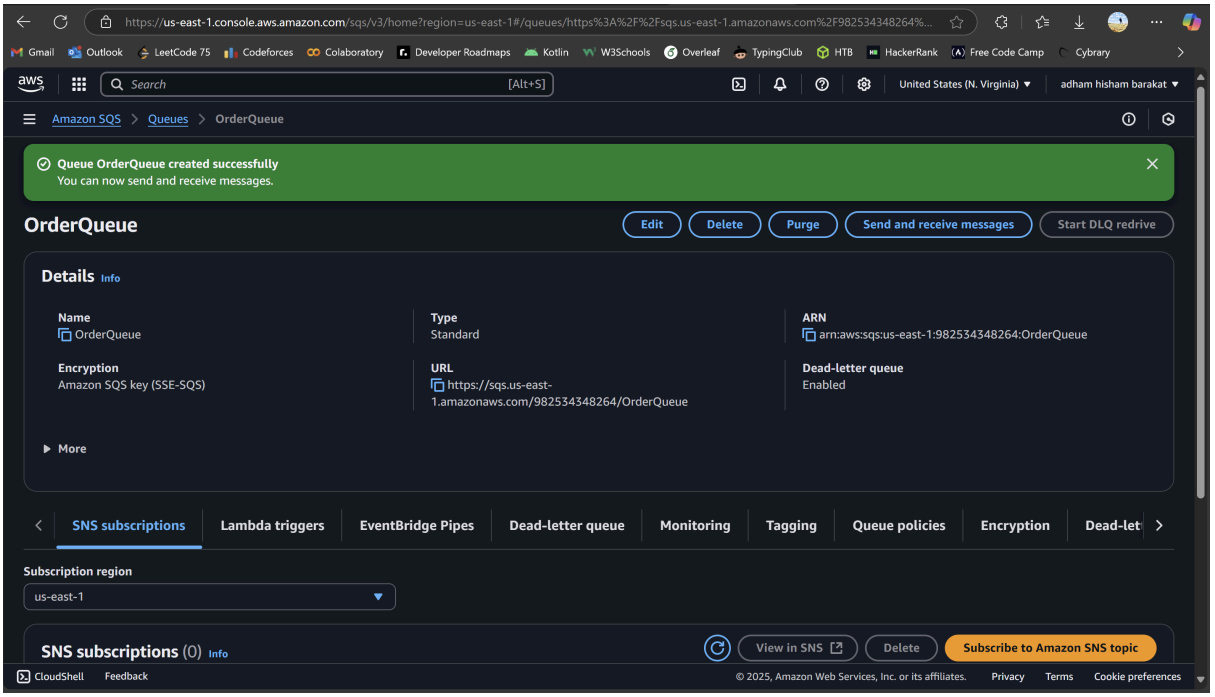


Created Dead Letter Queue:

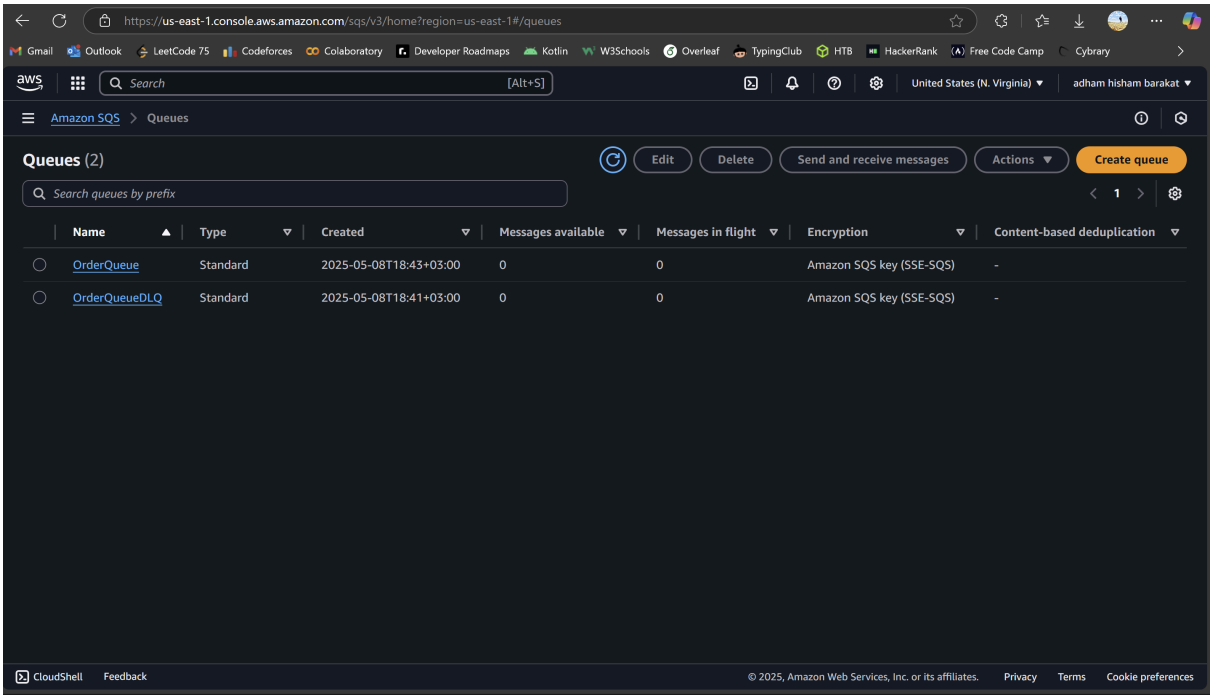


create Main Queue called OrdereQueue:

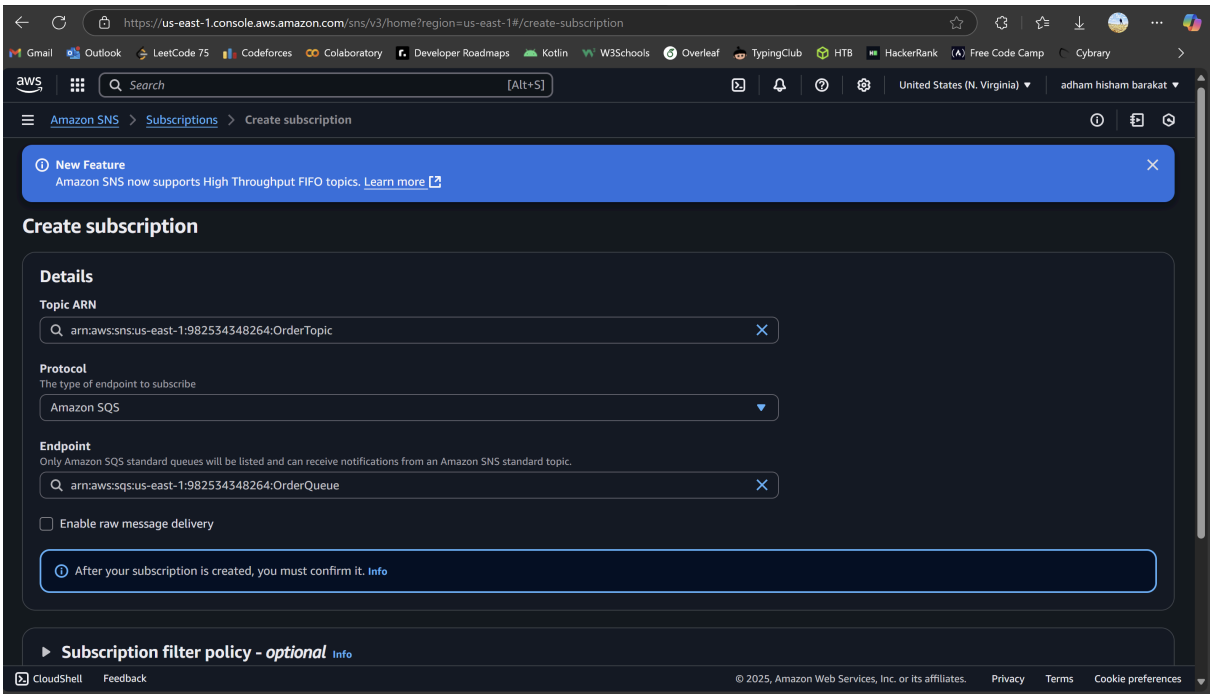

https://us-east-1.console.aws.amazon.com/sqs/v3/home?region=us-east-1#/create-queue
Amazon SQS > Queues > Create queue
Select which source queues can use this queue as the dead-letter queue.
Disabled
Enabled
Dead-letter queue - Optional Info
Send undeliverable messages to a dead-letter queue.
Set this queue to receive undeliverable messages.
Disabled
Enabled
Choose queue
arn:aws:sqs:us-east-1:982534348264:OrderQueueDLQ
Maximum receives
3
Should be between 1 and 1000
Tags - Optional Info
A tag is a label assigned to an AWS resource. Use tags to search and filter your resources or track your AWS costs.
Key
Enter key
Value - optional
Enter value
Remove
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

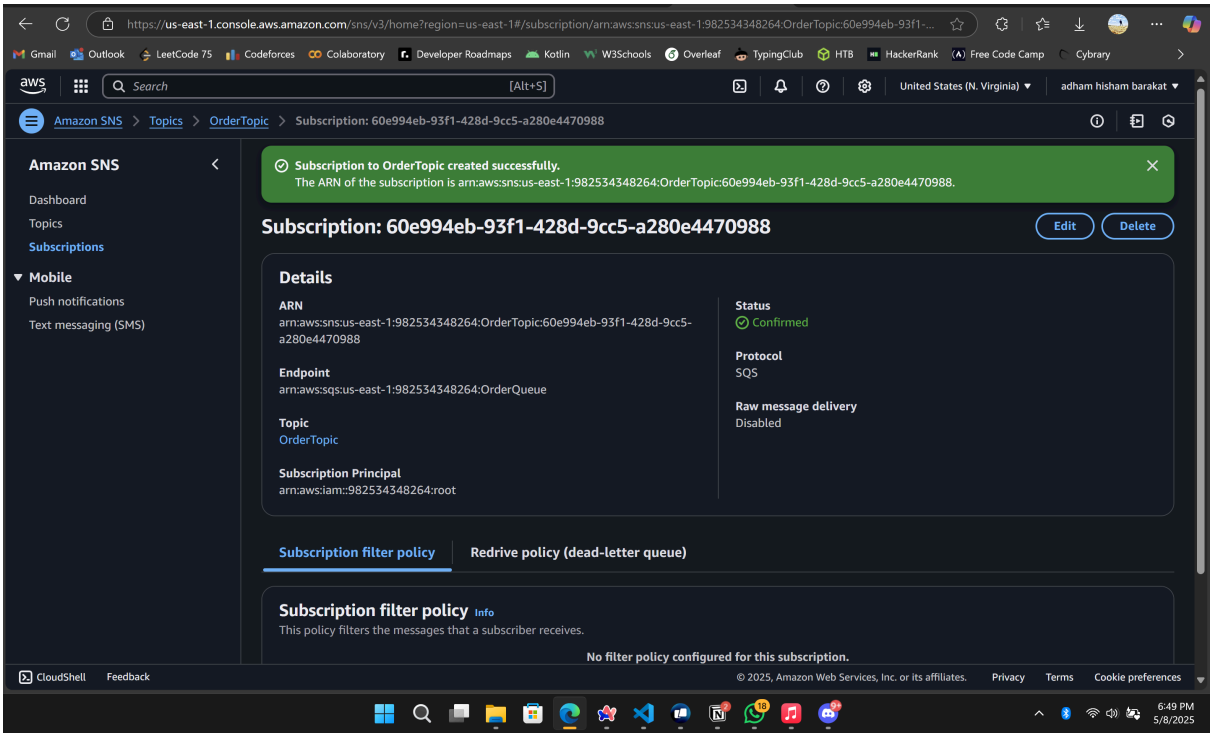

https://us-east-1.console.aws.amazon.com/sqs/v3/home?region=us-east-1#/queues/https%3A%2F%2Fsqs.us-east-1.amazonaws.com%2F982534348264%...
Amazon SQS > Queues > OrderQueue
Queue OrderQueue created successfully
You can now send and receive messages.
OrderQueue
Edit
Delete
Purge
Send and receive messages
Start DLQ redrive
Details Info
Name
OrderQueue
Type
Standard
ARN
arn:aws:sqs:us-east-1:982534348264:OrderQueue
Encryption
Amazon SQS key (SSE-SQS)
URL
https://sqs.us-east-1.amazonaws.com/982534348264/OrderQueue
Dead-letter queue
Enabled
More
SNS subscriptions
Lambda triggers
EventBridge Pipes
Dead-letter queue
Monitoring
Tagging
Queue policies
Encryption
Subscription region
us-east-1
SNS subscriptions (0) Info
View in SNS
Delete
Subscribe to Amazon SNS topic
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Both queues:

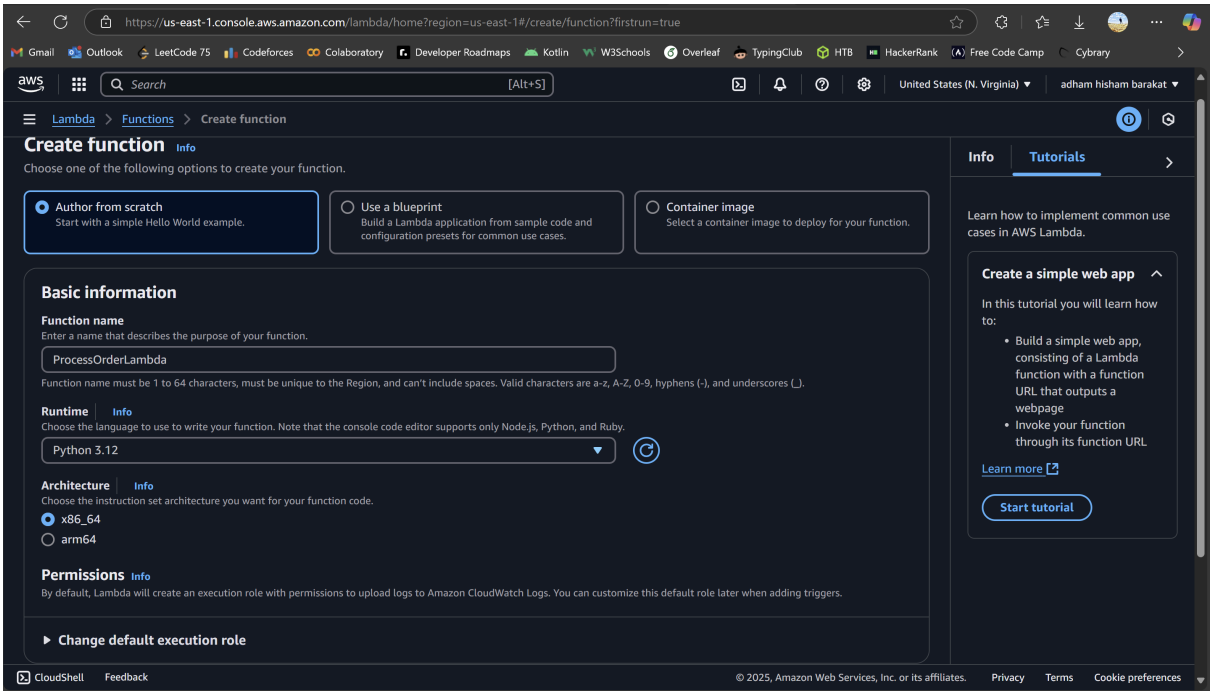


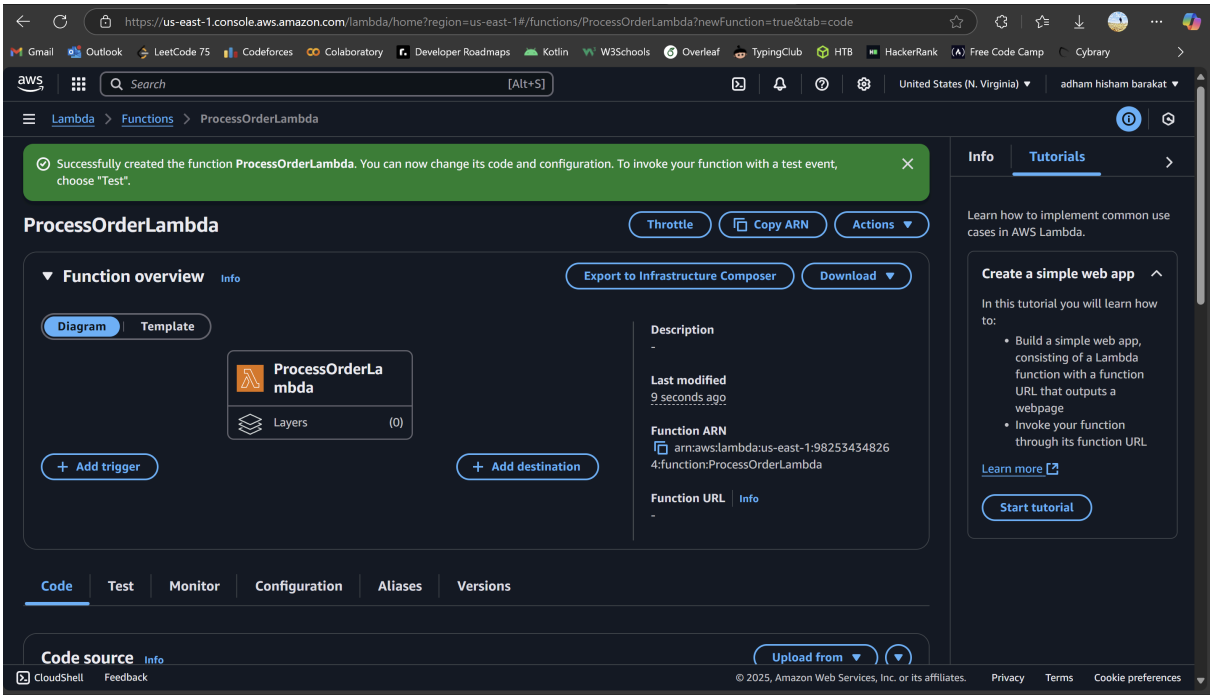
Subscribe SQS Queue to SNS Topic

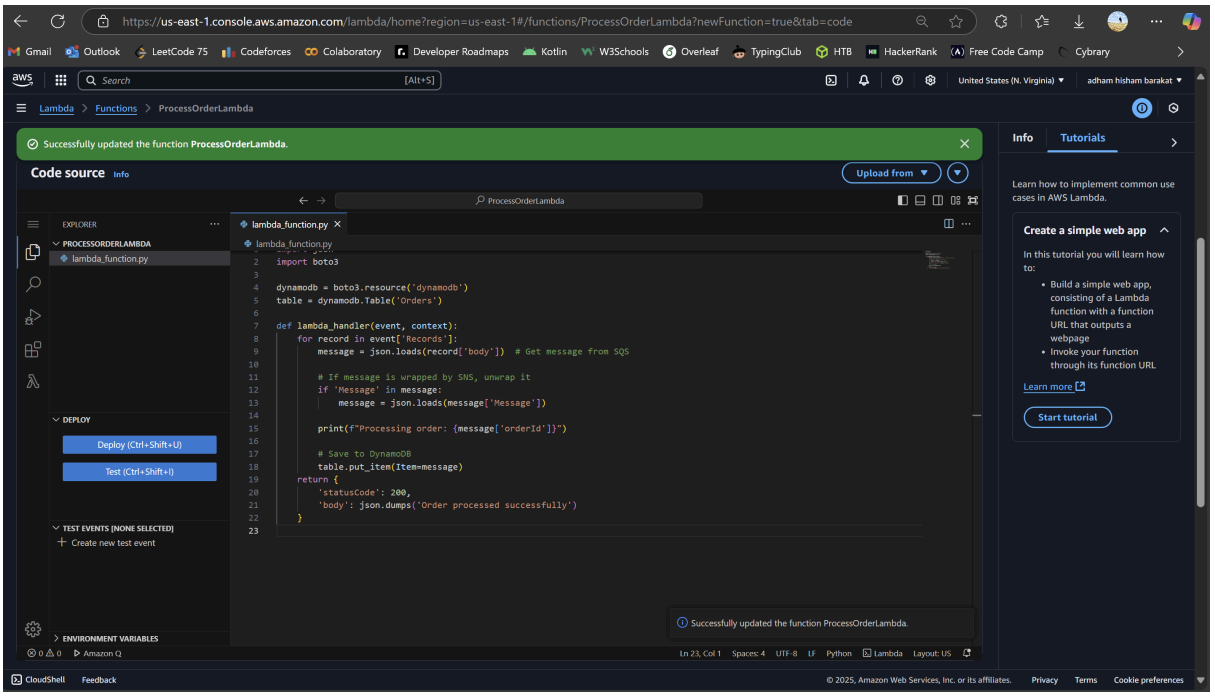




create lambda function



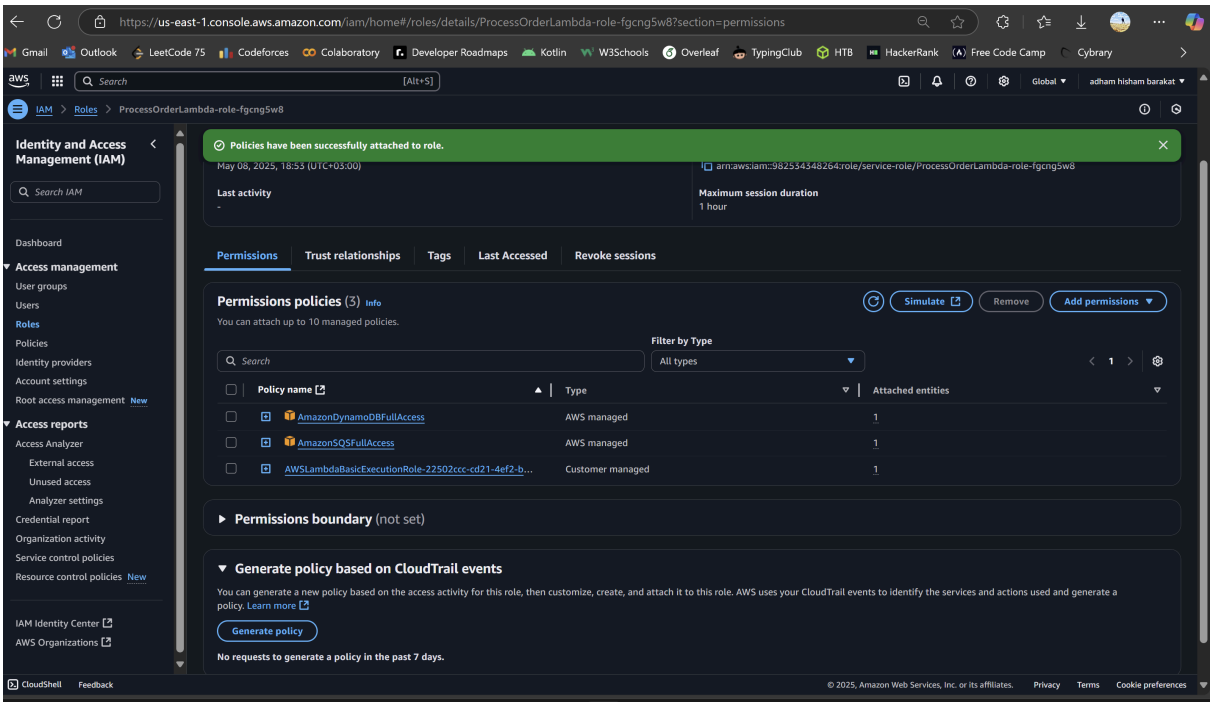




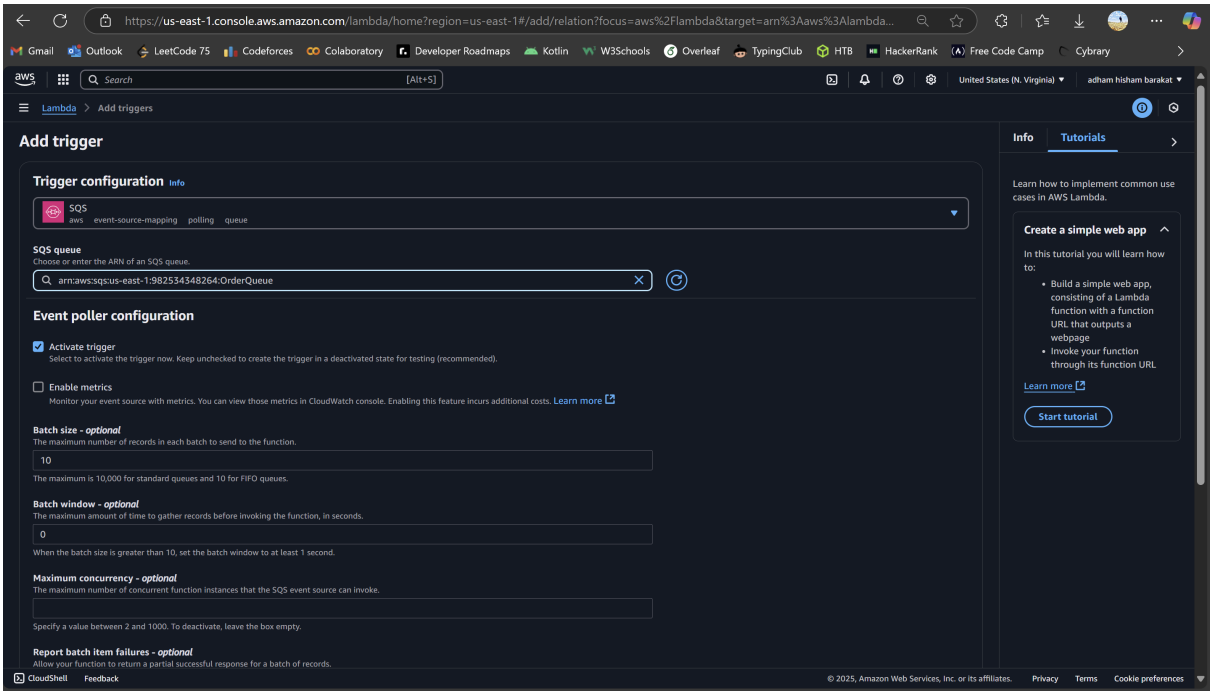

```
import json
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('Orders')

def lambda_handler(event, context):
```

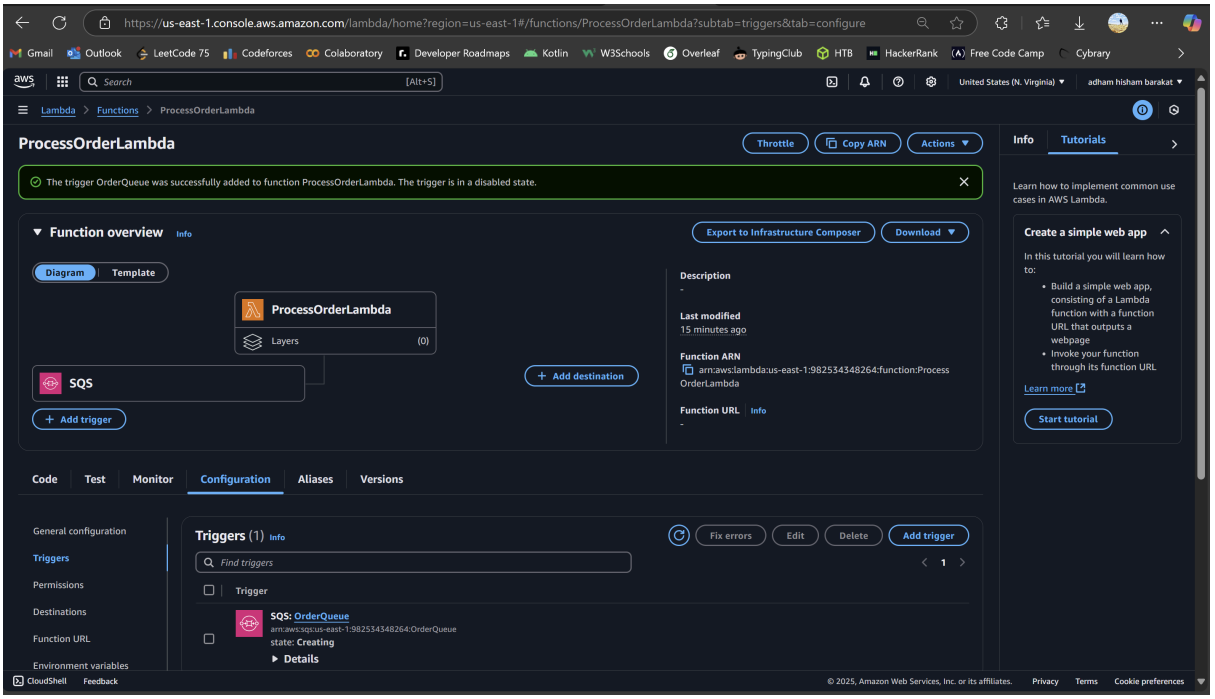
```
for record in event['Records']:
    message = json.loads(record['body'])  # Get message from SQS

    # If message is wrapped by SNS, unwrap it
    if 'Message' in message:
        message = json.loads(message['Message'])

    print(f"Processing order: {message['orderId']}")

    # Save to DynamoDB
    table.put_item(Item=message)

return {
    'statusCode': 200,
    'body': json.dumps('Order processed successfully')
}
```

Add to the lambda the needed permissions

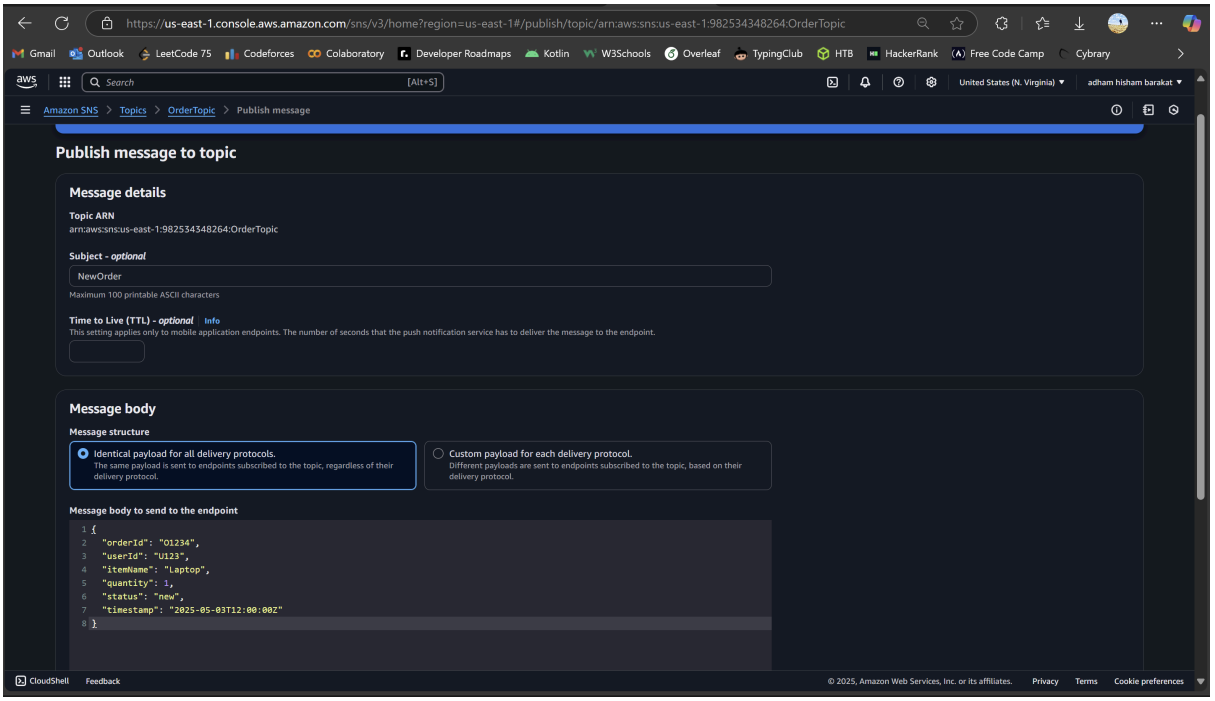


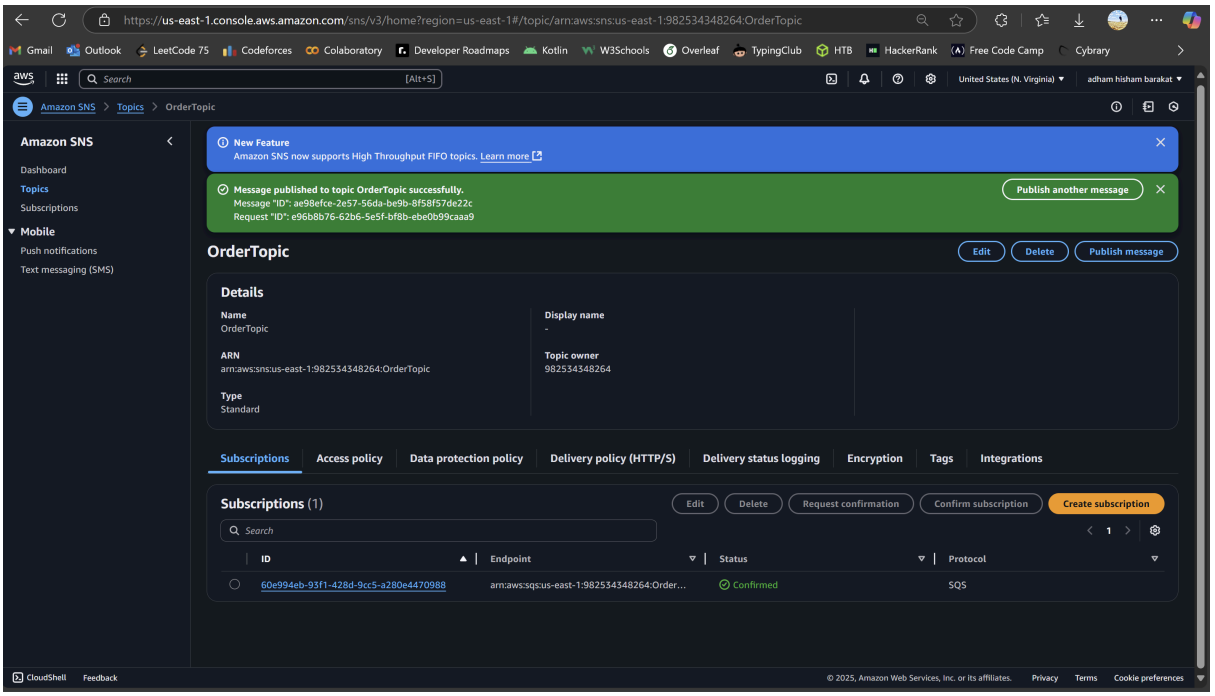
Add a trigger:



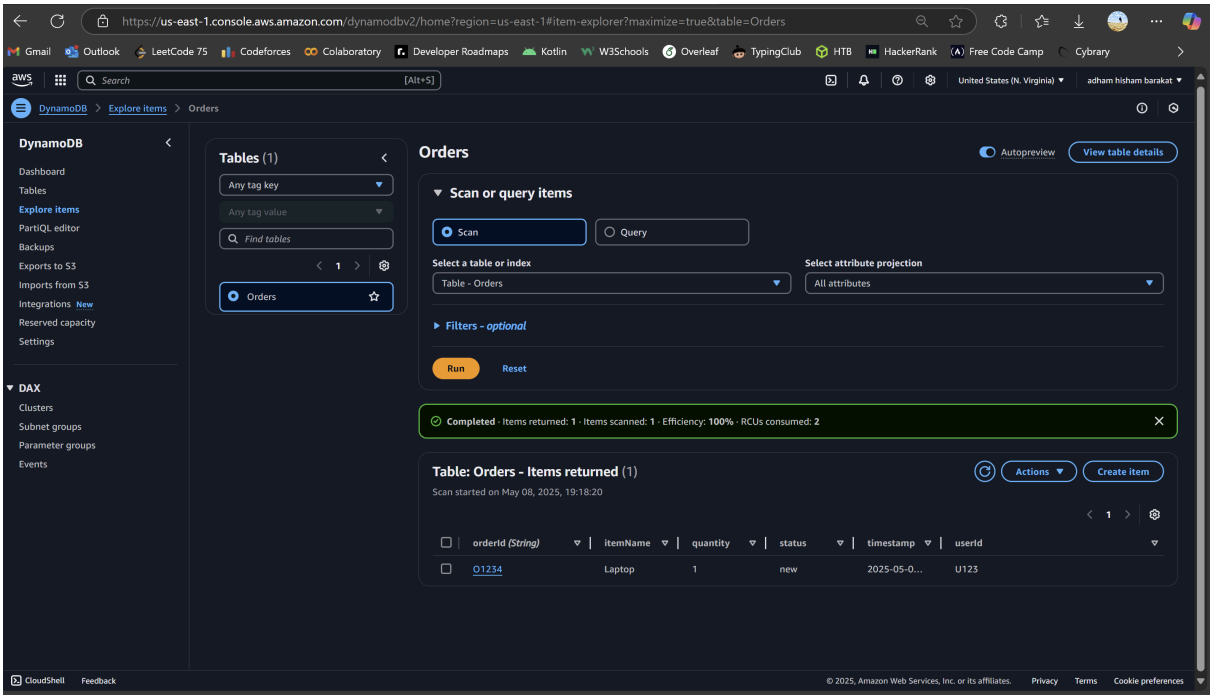


Publish Test Message to SNS

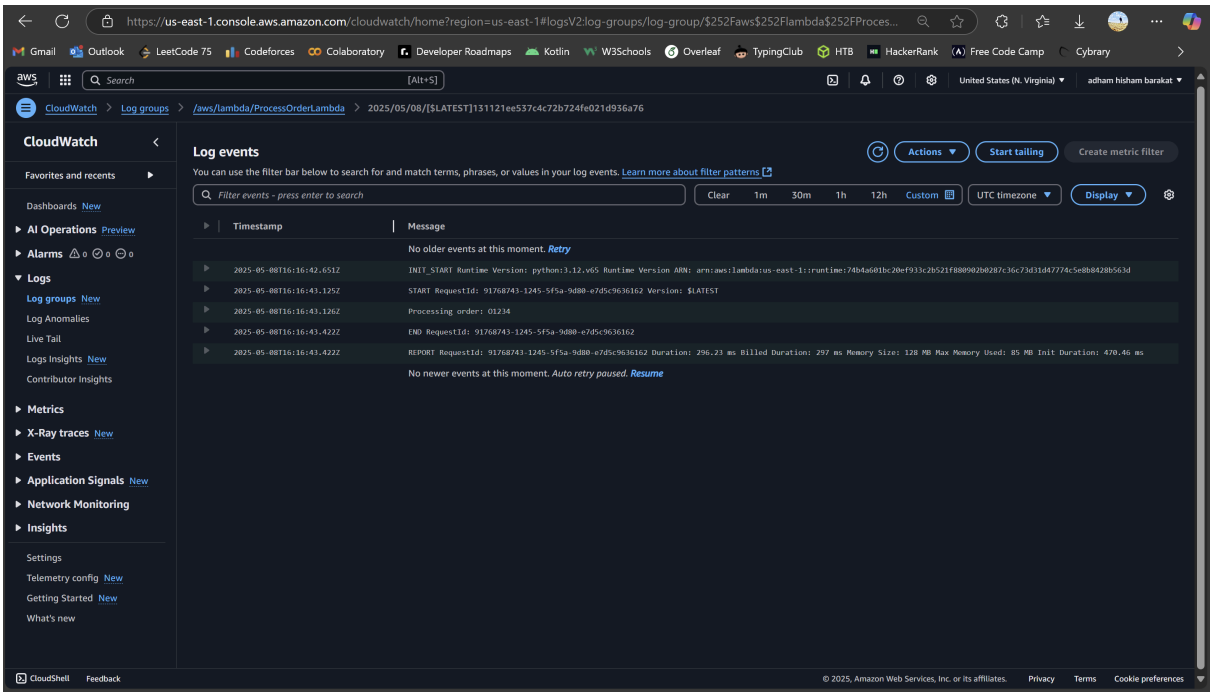




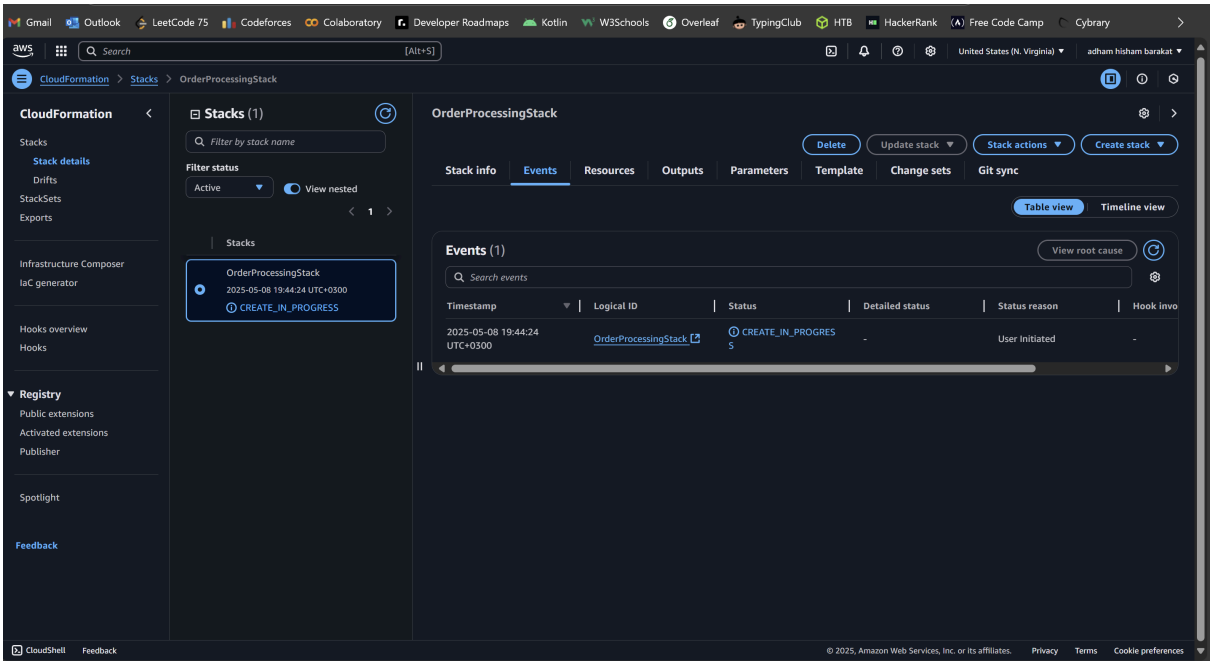
check dynamodb for the order as it has been placed

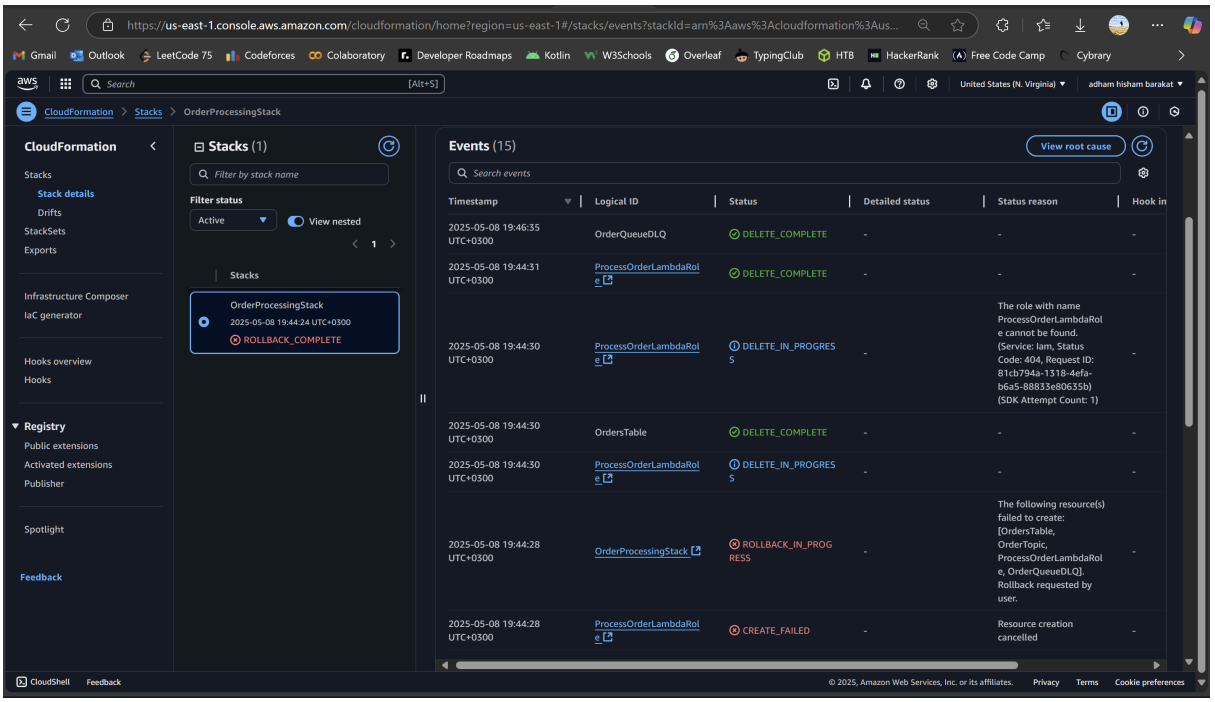


check cloudwatch group logs for the order 01234



cloudformation template (Bonus)





Script .yaml file for bonus Exported CloudFormation couse i did all using the console manually:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Event-Driven Order Notification System

Resources:
```

```
OrdersTable:
  Type: AWS::DynamoDB::Table
  Properties:
    TableName: Orders
    AttributeDefinitions:
      - AttributeName: orderId
        AttributeType: S
    KeySchema:
      - AttributeName: orderId
        KeyType: HASH
    BillingMode: PAY_PER_REQUEST

OrderTopic:
  Type: AWS::SNS::Topic
  Properties:
    TopicName: OrderTopic

OrderQueueDLQ:
  Type: AWS::SQS::Queue
  Properties:
    QueueName: OrderQueueDLQ-CF  # added suffix to avoid name conflict

OrderQueue:
  Type: AWS::SQS::Queue
  Properties:
    QueueName: OrderQueue-CF  # added suffix to avoid name conflict
    RedrivePolicy:
      deadLetterTargetArn: !GetAtt OrderQueueDLQ.Arn
      maxReceiveCount: 3

OrderQueueSubscription:
  Type: AWS::SNS::Subscription
  Properties:
    TopicArn: !Ref OrderTopic
    Protocol: sqs
    Endpoint: !GetAtt OrderQueue.Arn
    RawMessageDelivery: false
```

```
OrderQueuePolicy:
  Type: AWS::SQS::QueuePolicy
  Properties:
    Queues:
      - !Ref OrderQueue
    PolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: Allow
          Principal: "*"
          Action: "SQS:SendMessage"
          Resource: !GetAtt OrderQueue.Arn
          Condition:
            ArnEquals:
              aws:SourceArn: !Ref OrderTopic

ProcessOrderLambdaRole:
  Type: AWS::IAM::Role
  Properties:
    # No RoleName → auto-generated by CloudFormation to avoid naming cor
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: Allow
          Principal:
            Service: lambda.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
      - arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess
      - arn:aws:iam::aws:policy/AmazonSQSFullAccess

ProcessOrderLambda:
  Type: AWS::Lambda::Function
  Properties:
    FunctionName: ProcessOrderLambda
    Runtime: python3.12
```

```
      Handler: index.lambda_handler
      Role: !GetAtt ProcessOrderLambdaRole.Arn
      Code:
        ZipFile: |
          import json
          import boto3

          dynamodb = boto3.resource('dynamodb')
          table = dynamodb.Table('Orders')

          def lambda_handler(event, context):
              for record in event['Records']:
                  message = json.loads(record['body'])
                  if 'Message' in message:
                      message = json.loads(message['Message'])
                  print(f"Processing order: {message['orderId']}")
                  table.put_item(Item=message)
              return {
                  'statusCode': 200,
                  'body': json.dumps('Order processed successfully')
              }

  LambdaSQSTrigger:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      EventSourceArn: !GetAtt OrderQueue.Arn
      FunctionName: !Ref ProcessOrderLambda
      Enabled: true
      BatchSize: 1

Outputs:
  OrdersTableName:
    Description: DynamoDB table name
    Value: !Ref OrdersTable
  OrderTopicArn:
    Description: SNS topic ARN
    Value: !Ref OrderTopic
  OrderQueueURL:
```

```
  Description: SQS Queue URL
  Value: !Ref OrderQueue
OrderQueueDLQURL:
  Description: SQS Dead Letter Queue URL
  Value: !Ref OrderQueueDLQ
LambdaFunctionName:
  Description: Lambda function name
  Value: !Ref ProcessOrderLambda
```