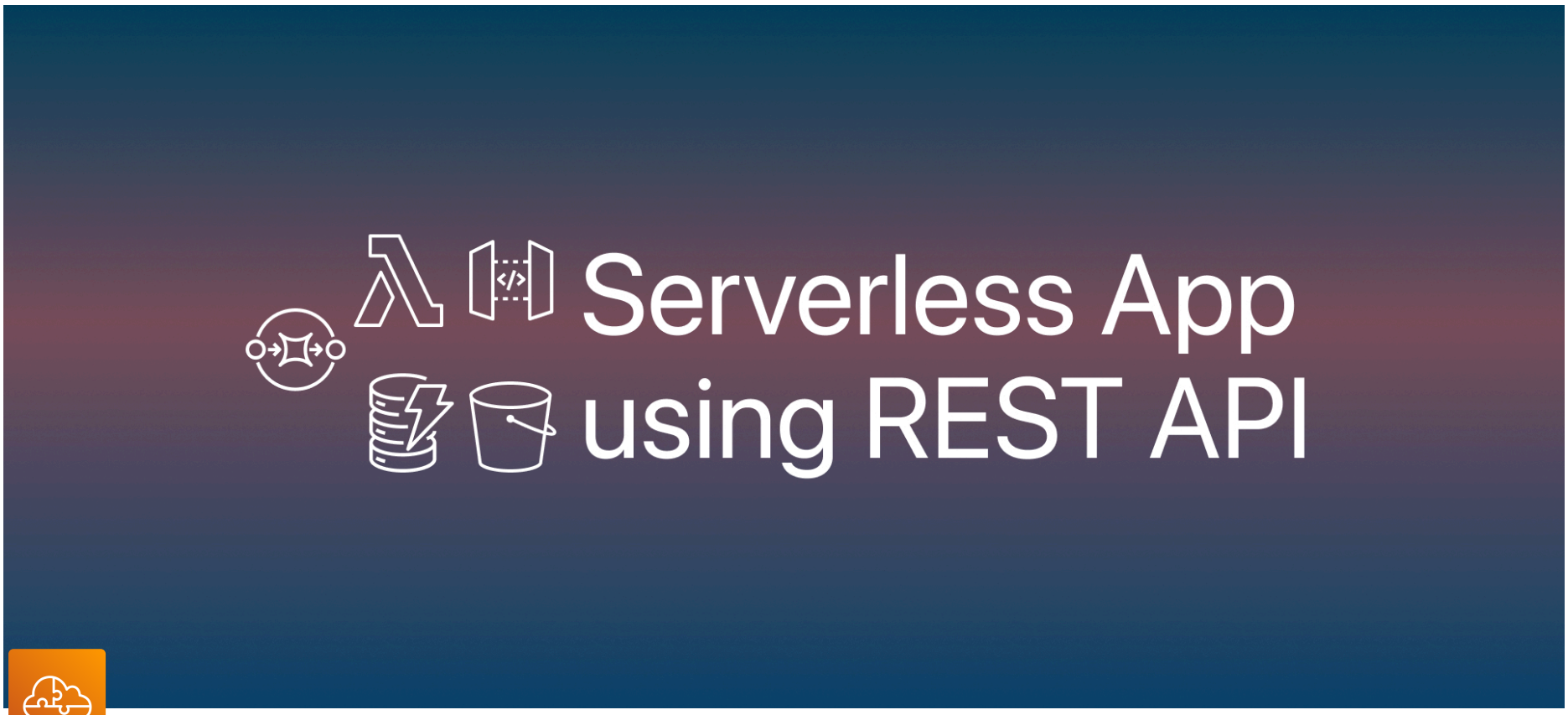
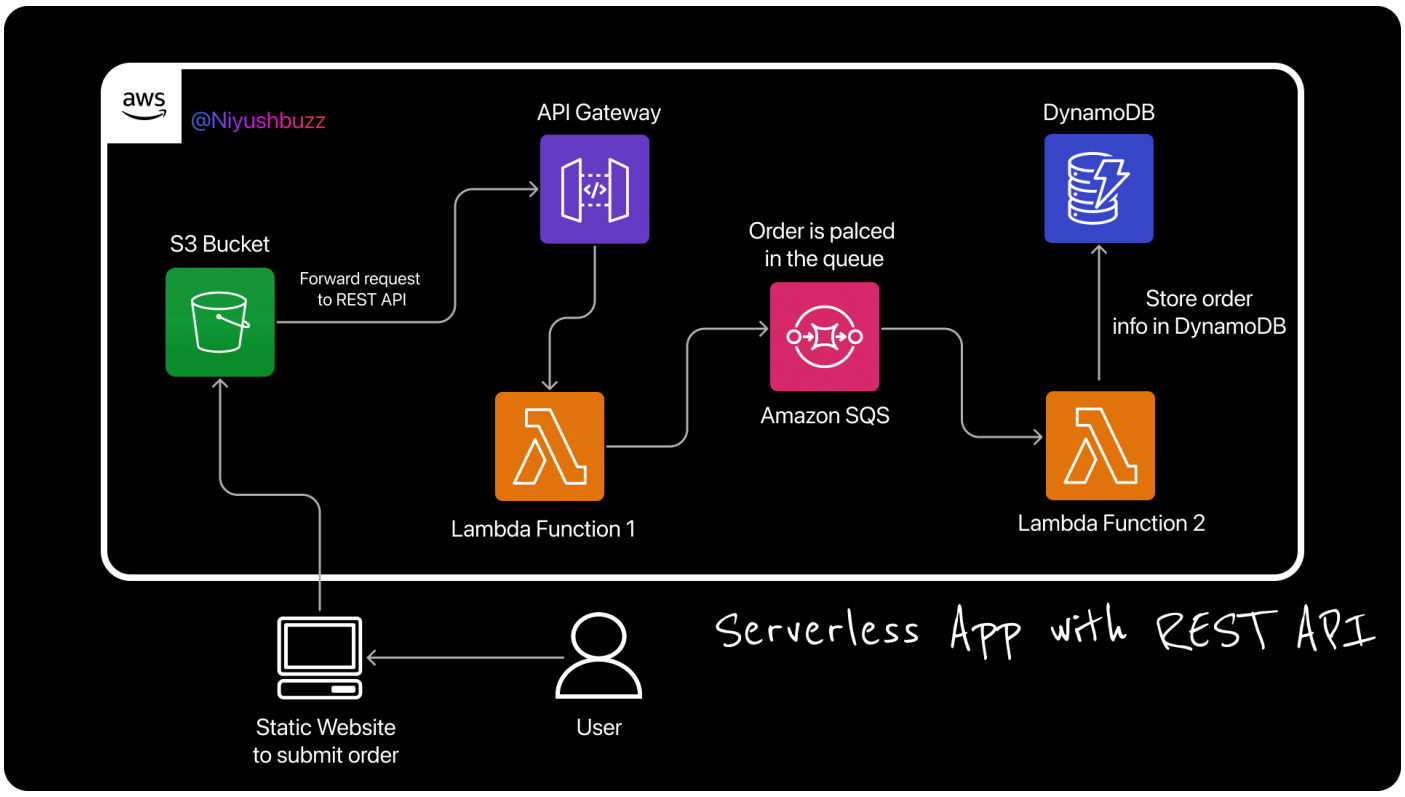


Serverless App with REST API

Created by	Ⓝ Niyush Bjr
Created time	@26 July 2025 00:21
level	Intermediate
Description	Building a Serverless Application with REST API
AWS Services	API Gateway, DynamoDB, Lambda, S3, SQS
Other tools	VsCode
Category	Compute
Buy-Me-a-Coffee	http://buymeacoffee.com/niyushbjr1L

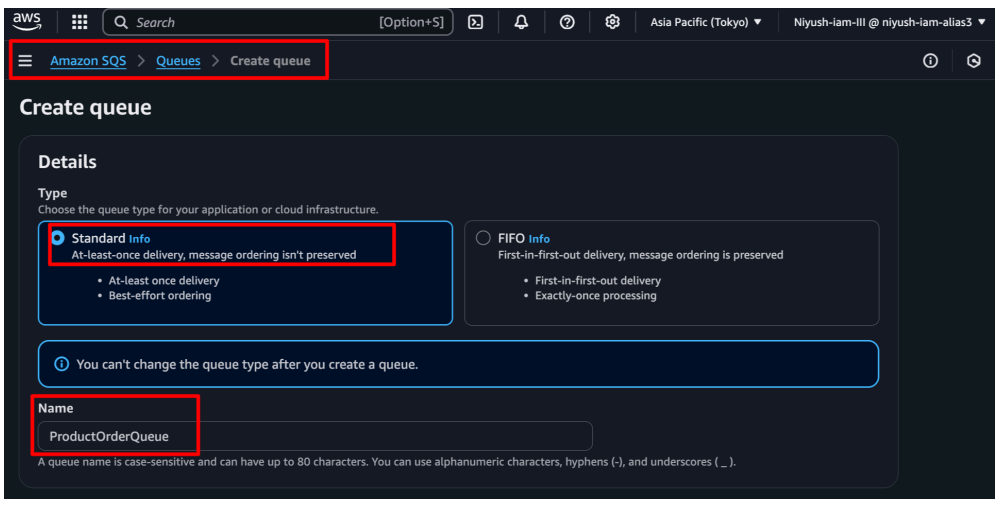
Overall Architecture.

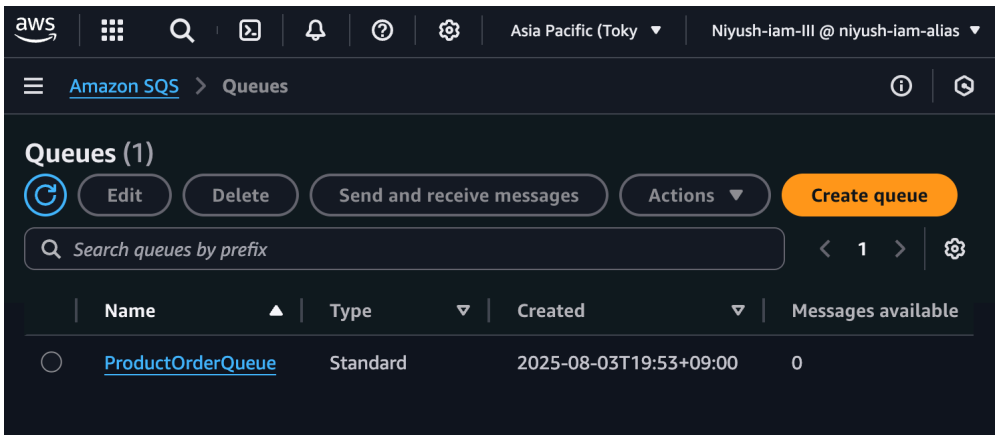


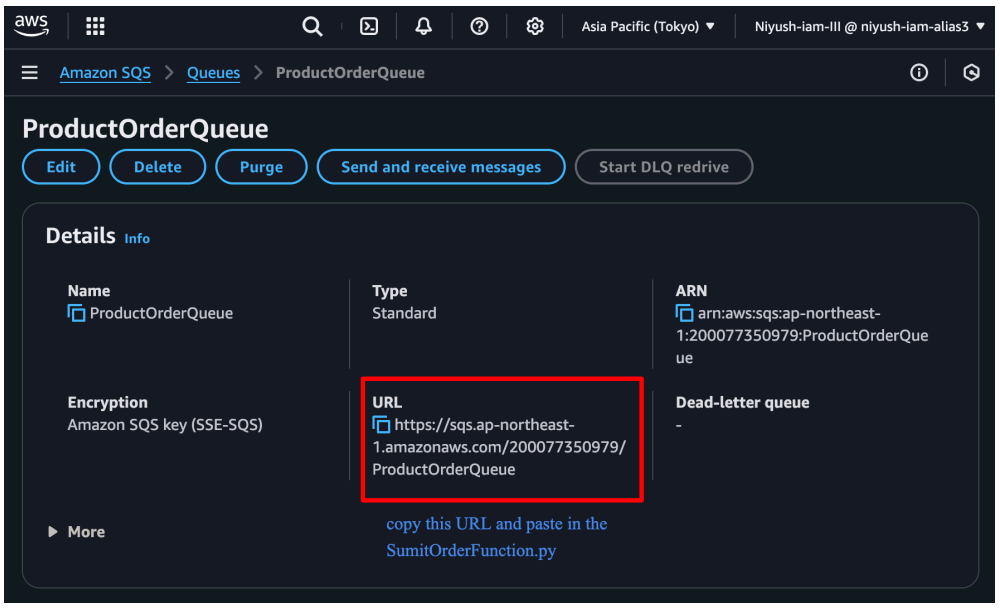
Step 1 : Create the SQS queue.

- Head to SQS console.
- Queue Type : Standard queue.
- Name : `ProductOrderQueue`
- Leave all the Remaining setting to `Default` and then click on `create queue`
- Copy the URL of the queue.
  (this will be used later on with lambda function)

▼ `[Image]` Create the queue



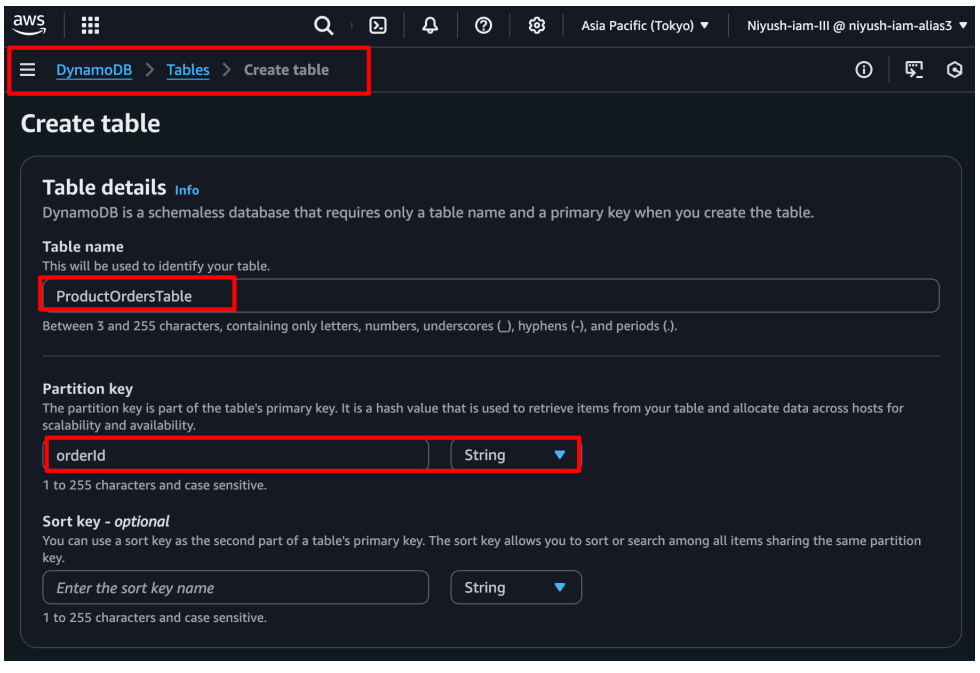




Step 2 : Create the DynamoDB table.

- Head to DynamoDB console → `Create Table`
- Table Name : `ProductOrdersTable`
- Primary Key : orderId
- Leave the rest to default and hit `Create Table`

▼ **[Image]** Create a dynamoDB table



Step 3 : Create the Lambda Functions.

Creating the first lambda function.

- Head over to Lambda Console → `Create Function`
- Function Name : `SubmitOrderFunction`
  Runtime : Python 3.9

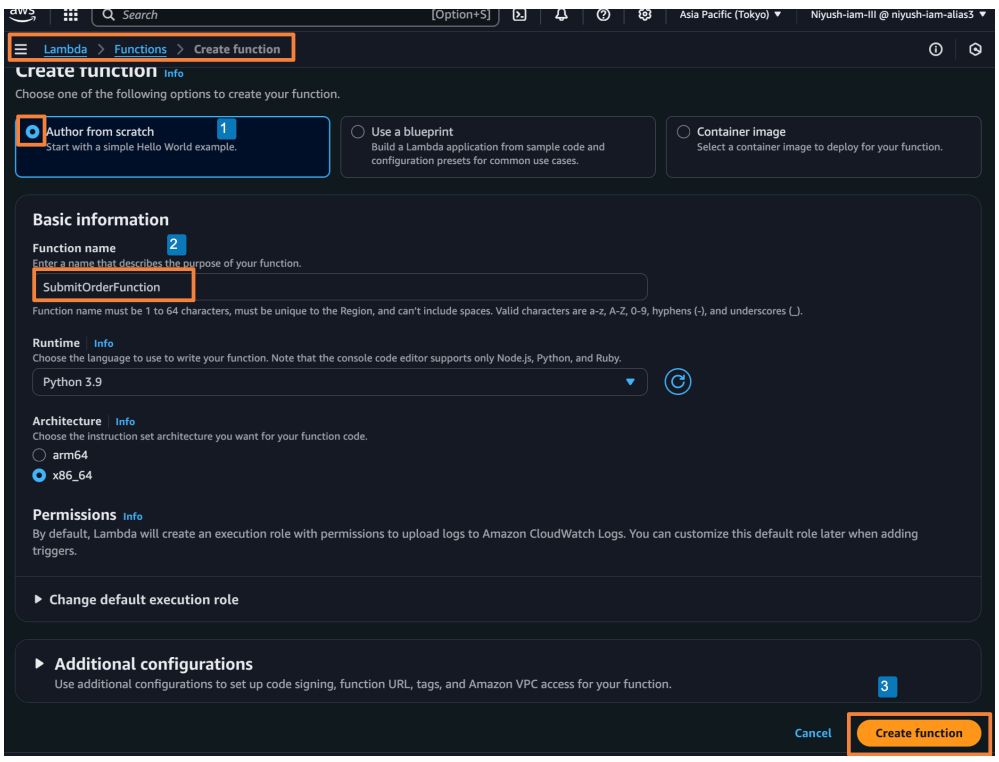
▼ **[Image]** Creating the lambda Function Image

▼ **Python Code Here ↓**

```
import json
import boto3

sqs = boto3.client('sqs')
queue_url = 'YOUR_SQS_QUEUE_URL'

def lambda_handler(event, context):
```

```
    try:
        order_details = json.loads(event['body'])
        response = sqs.send_message(QueueUrl=queue_url, MessageBody=json.dumps(order_details))
        return {'statusCode': 200, 'headers': {'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Headers': 'Content-Type', 'Access-Control-Allow-Methods': 'OPTIONS,POST'}, 'body': json.dumps({'message': 'Order submitted to queue successfully'})}
    except Exception as e:
        return {'statusCode': 400, 'body': json.dumps({'error': str(e)})}
```

- Add the URL of the SQS queue to the python code.



Creating the lambda function

jj

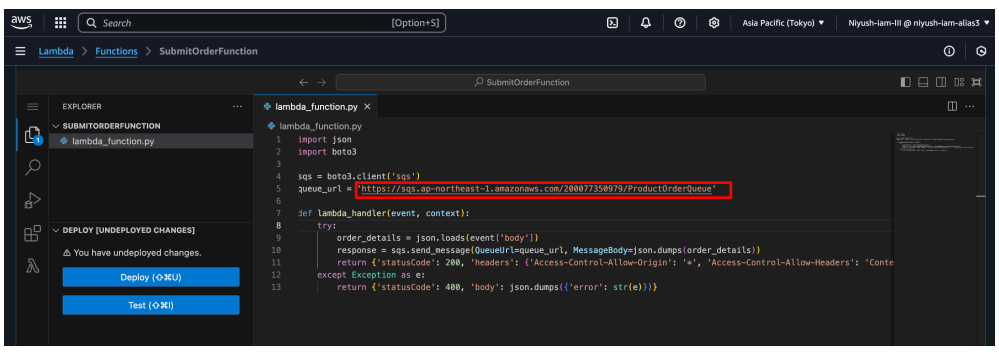
- Add permission to the Lambda function
  - `AmazonSQSFullAccess`
- Then finally hit Deploy
- Once the Function has been successfully Created Check or test the function.

▼ **Test Code for Lambda Function 1.**

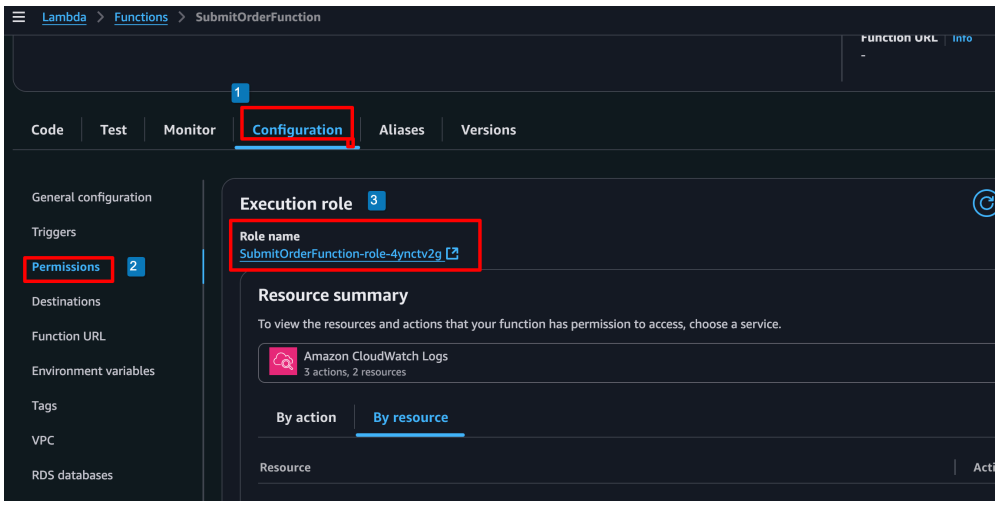
```
{
  "body": "{\"productName\":\"THIS_IS_A_SAMPLE_TEST\",\"QUANTITY\":100}"
}
```

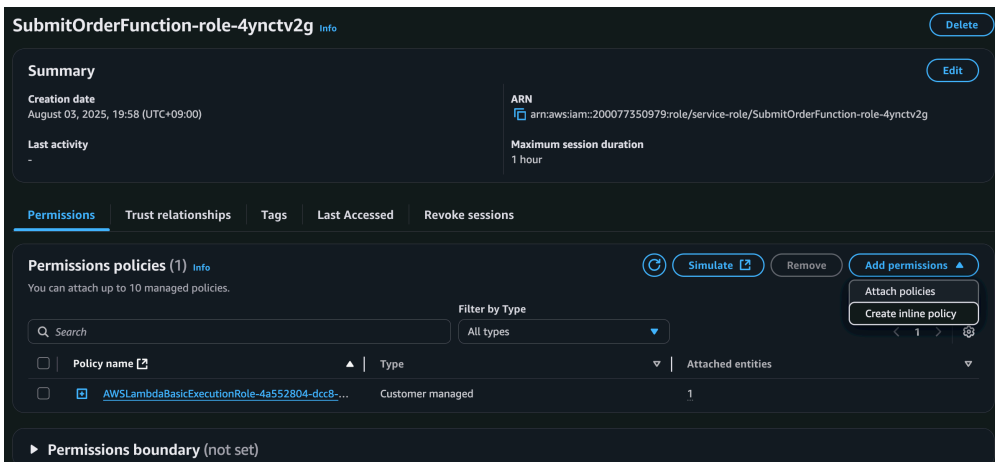
- Go the SQS and poll for messages
  - there you should see a message waiting the queue.

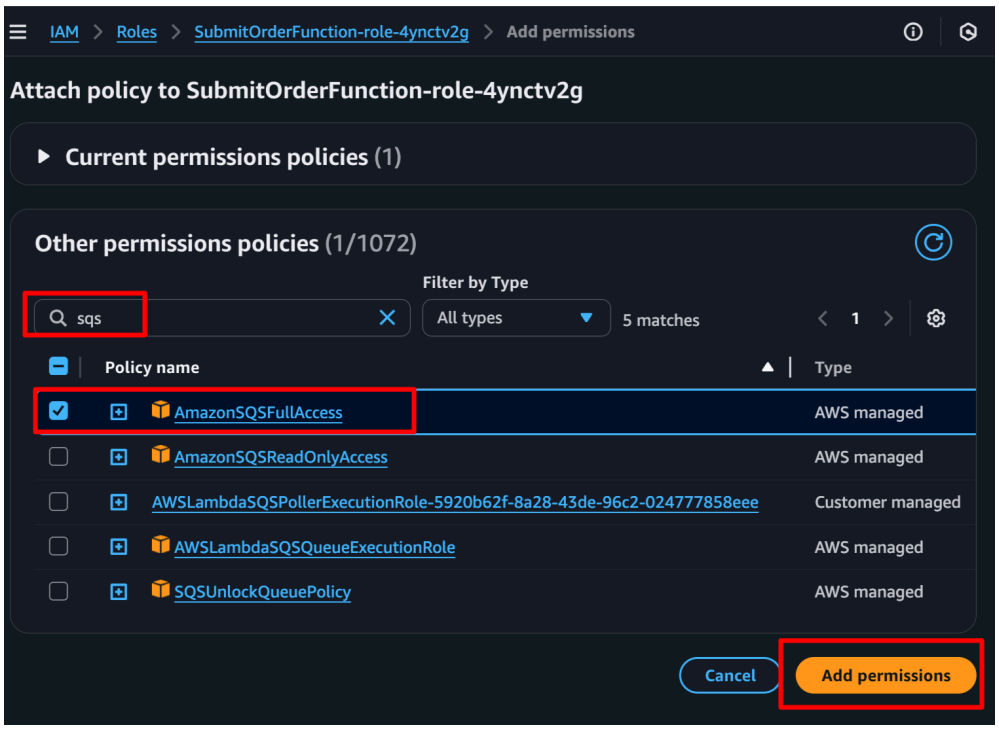
▼ [Image] adding code to the lambda function and updating the URL



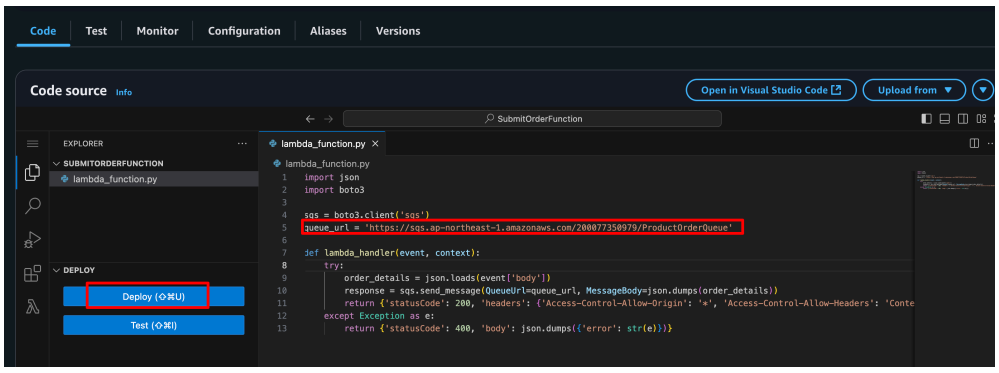
▼ [Image] adding permission to Lambda function.

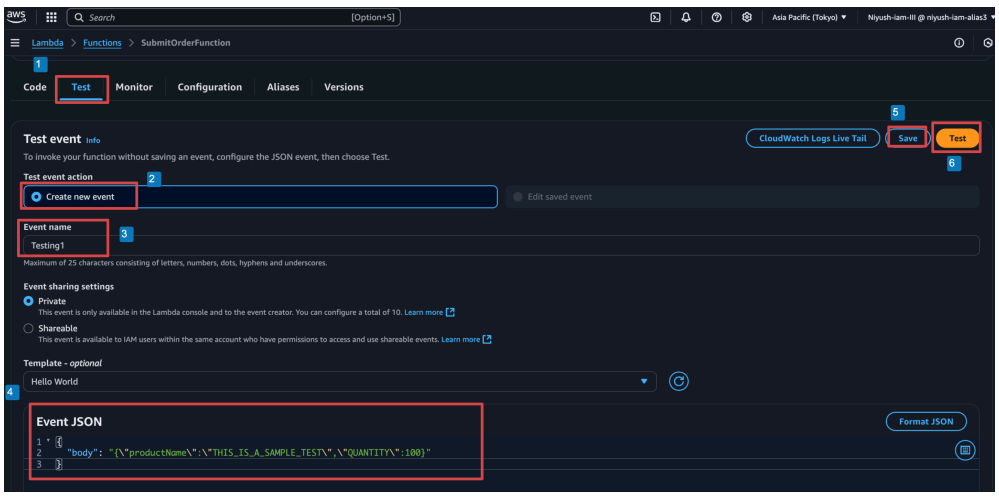


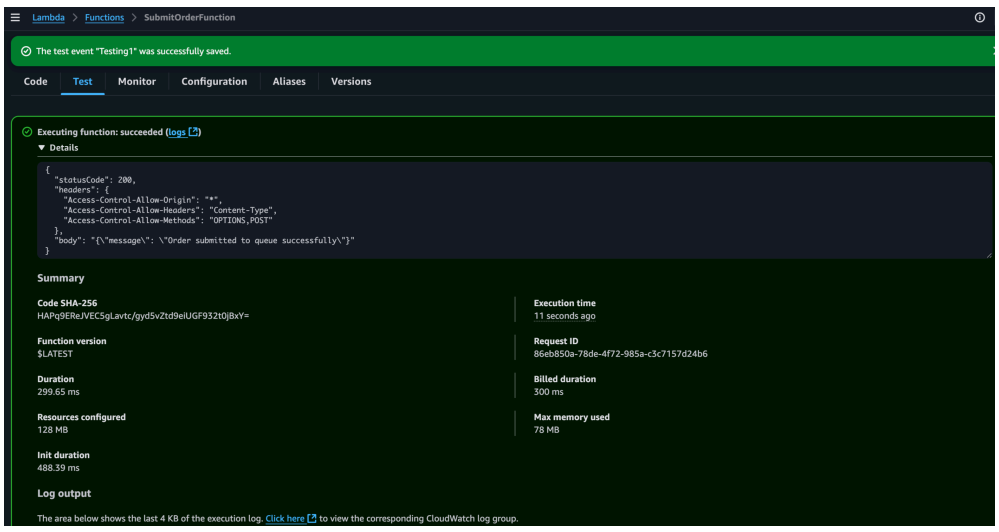




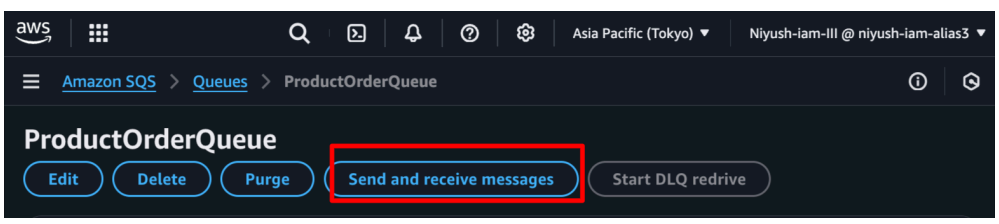
▼ [Image] Deploy and test the lambda function

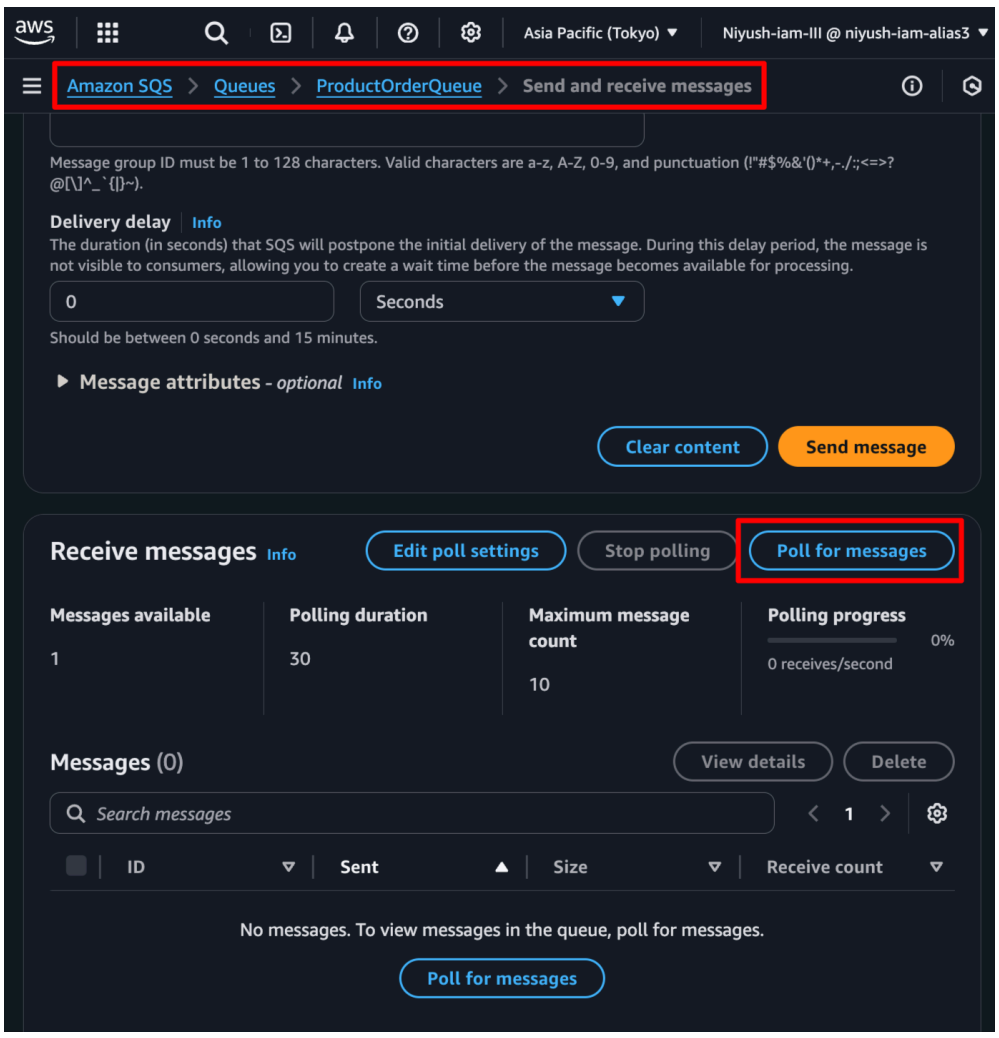


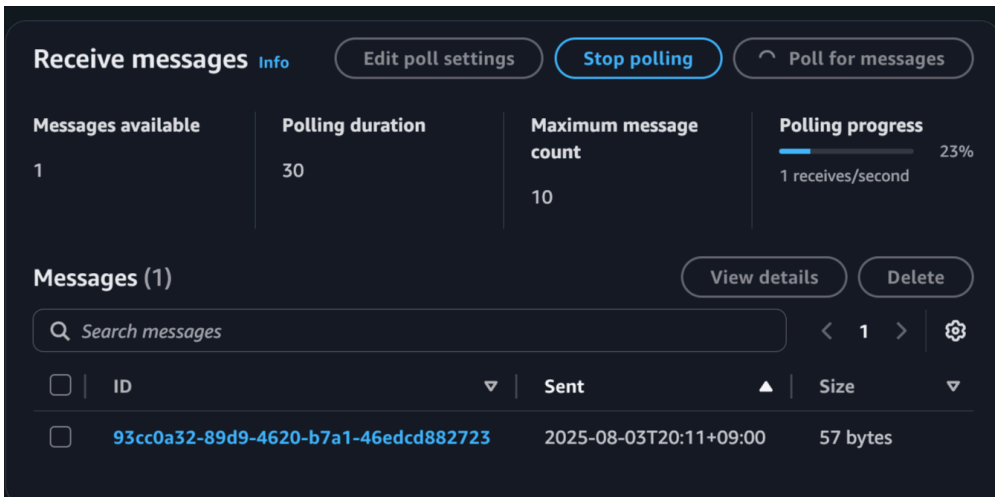


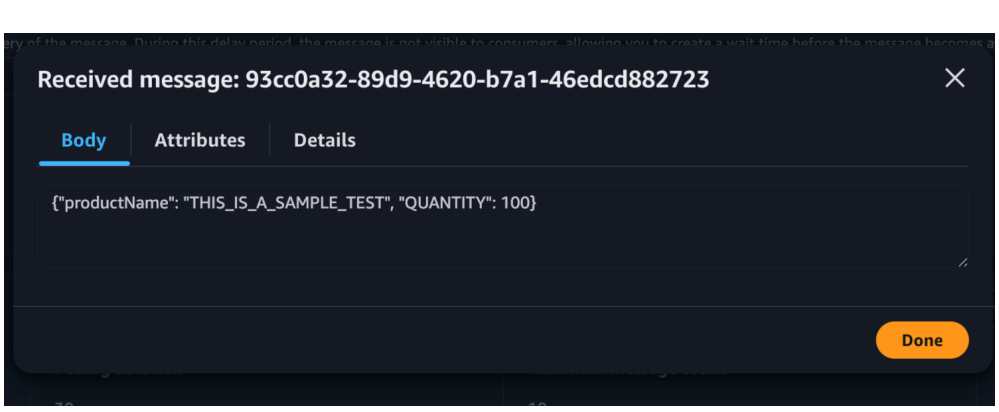


▼ [Image] Poll for message in the SQS queue









Creating the Second Lambda Function.

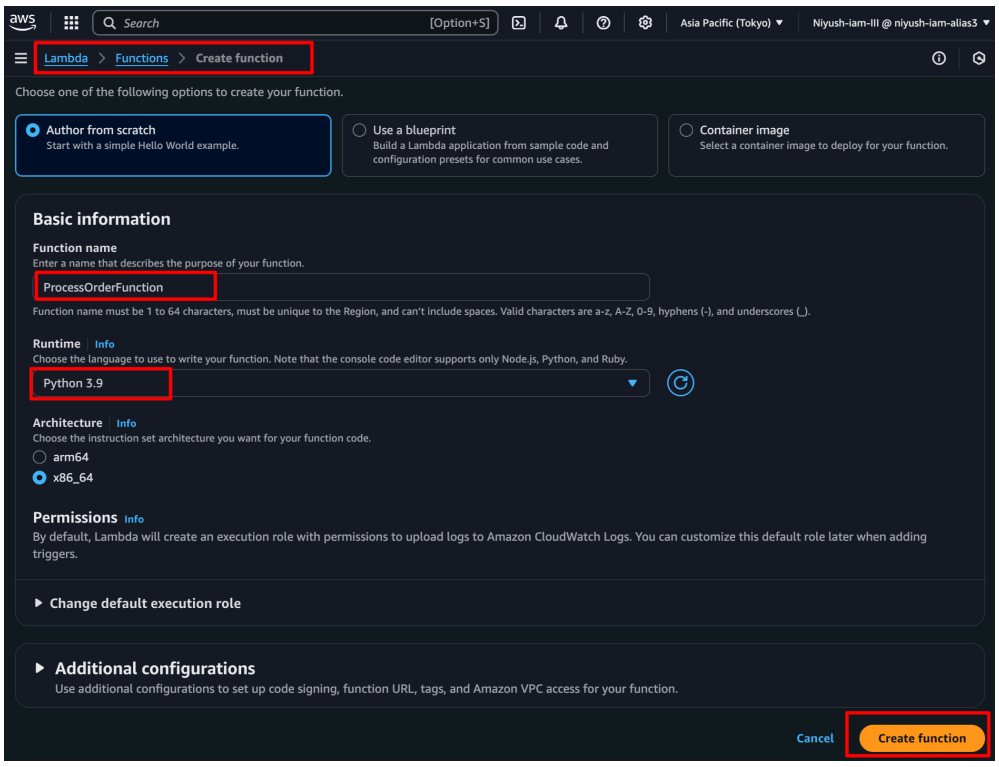
- Go the Lambda console and create another function.
- Name : `ProcessOrderFunction`
  Runtime : Python 3.9

▼ [Image] Making the second lambda function

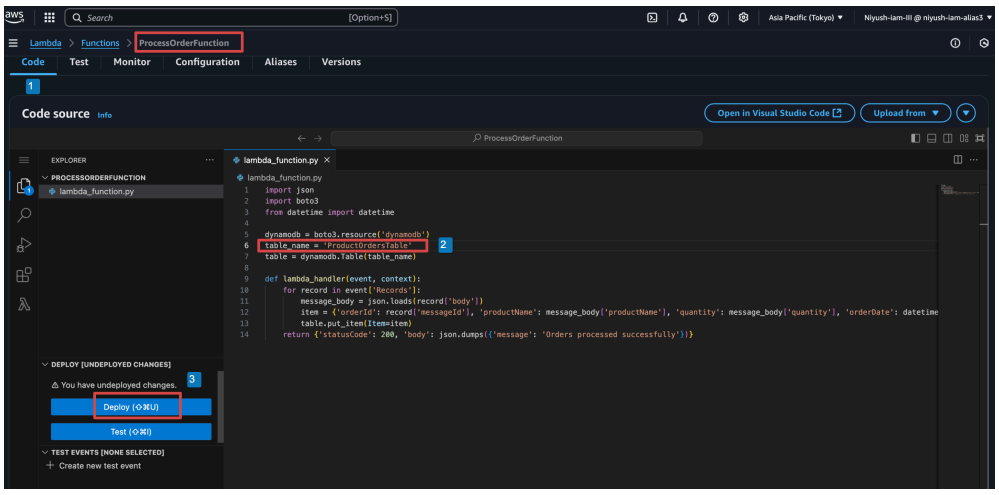
▼ ***Code for ProcessOrderFunction.***

```
import json
import boto3
from datetime import datetime

dynamodb = boto3.resource('dynamodb')
table_name = 'YOUR_DYNAMODB_TABLE_NAME'
table = dynamodb.Table(table_name)

def lambda_handler(event, context):
    for record in event['Records']:
        message_body = json.loads(record['body'])
        item = {'orderId': record['messageId'], 'productName': message_body['productName'], 'quantity': message_body['quantity'], 'orderDate': datetime.now().isoformat()}
        table.put_item(Item=item)
    return {'statusCode': 200, 'body': json.dumps({'message': 'Orders processed successfully'})}
```

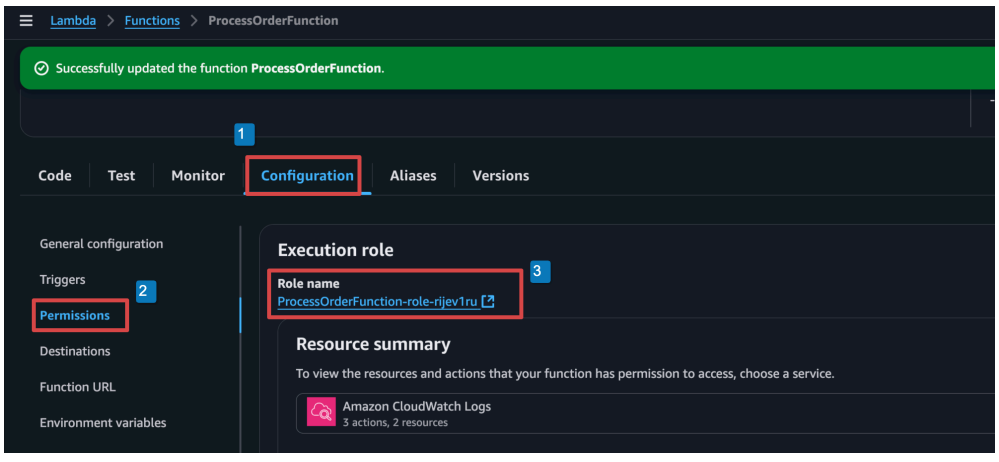
- In the above make sure to replace the table_name with
  the actual DynamoDB table name.
- Add the following permission policies to the execution role.
  - AmazonSQSFullAccess
  - AmazonDynamoDBFullAccess

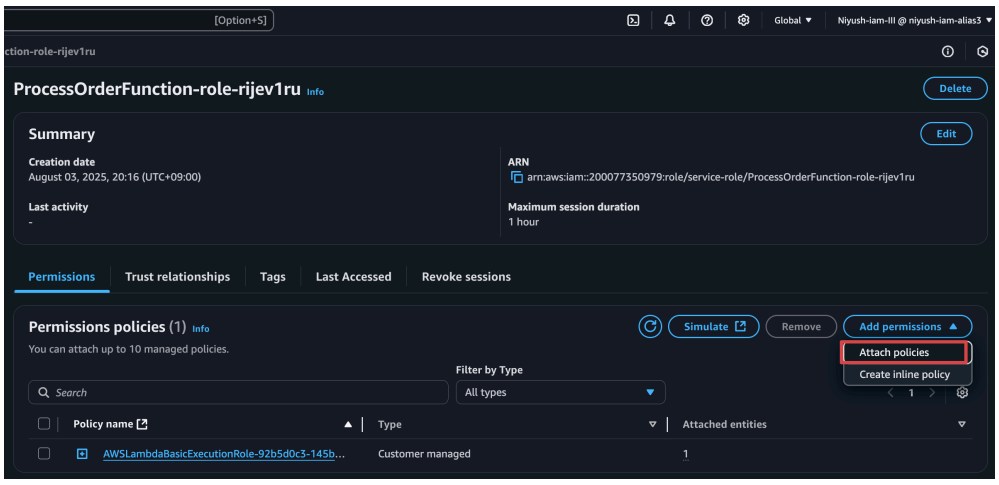


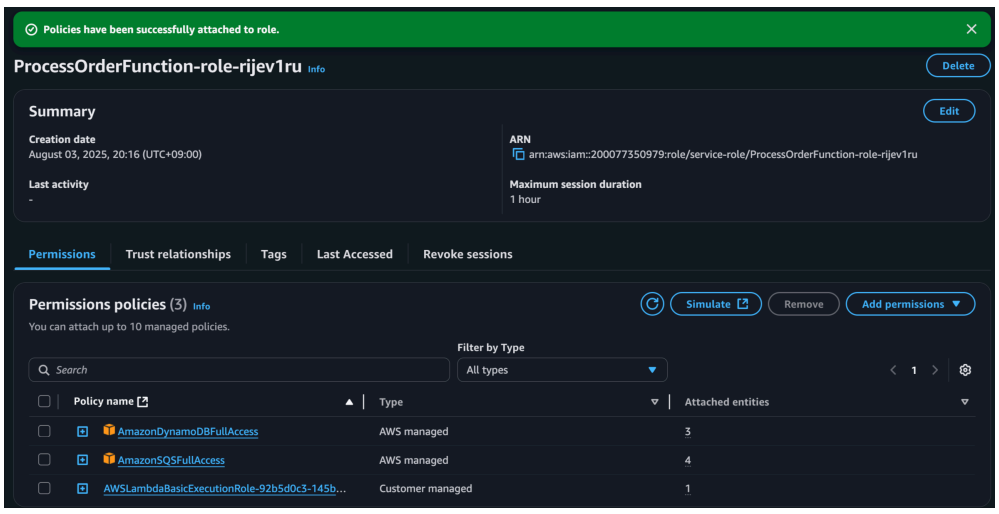
▼ [Image] Replacing the table name with actual table name



▼ [Image] Adding permission to the function2



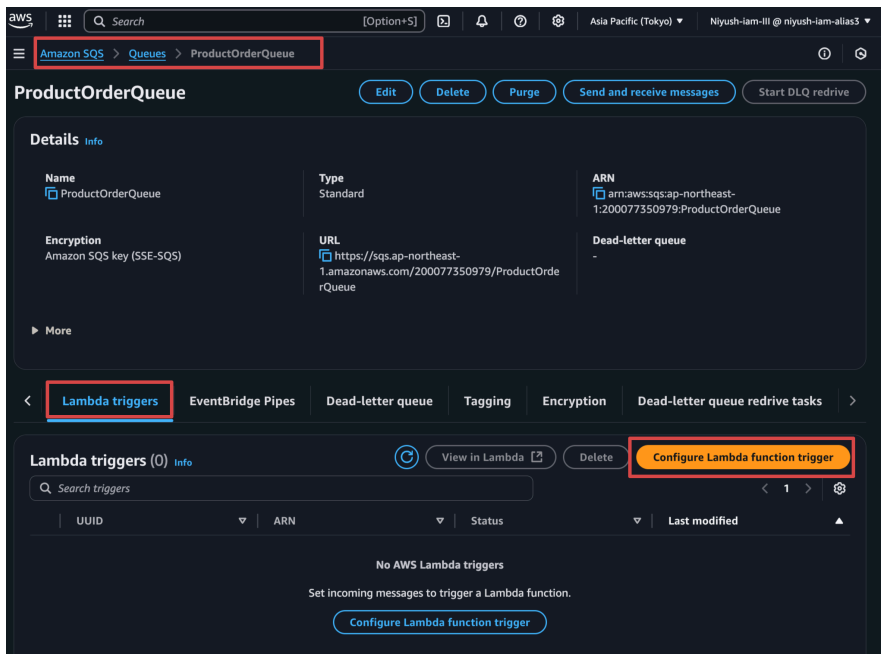




Step 4 : Deploy and test the application.

- Go to SQS console and configure lambda trigger and specify the `ProcessOrderFunction`.
- Check the DynamoDB table to see it if the first test event was processed.
- Test the code using the cloudshell.
  - create a file name `input.json`

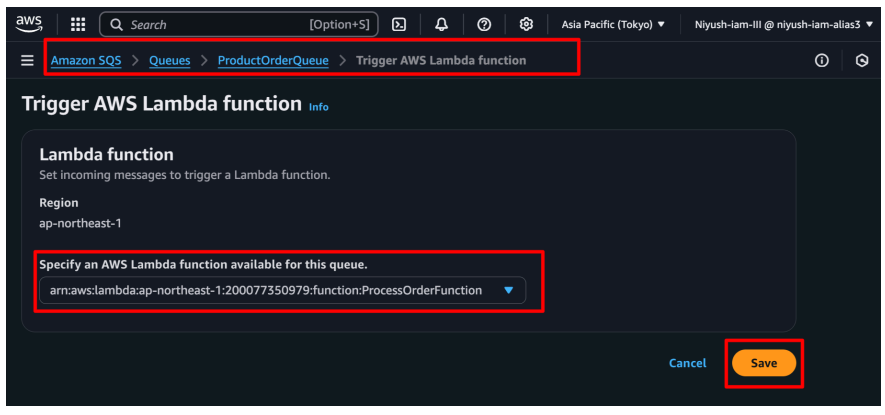
▼ [Image] Configure lambda trigger

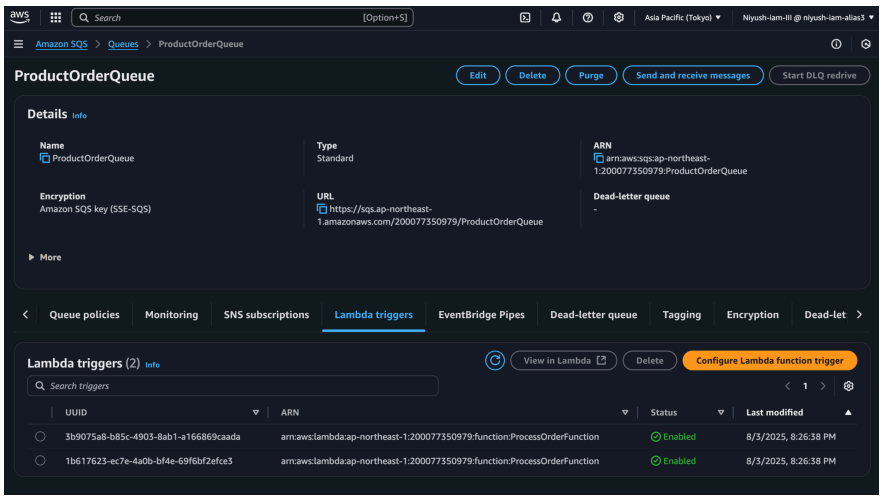



```
pwd
ls
touch input.json
nano input.json
{
  "body": "{\"productName\":\"Test Product 2\",\"quantity\":2}"
}


ctrl +z → Y ⇒ enter
```

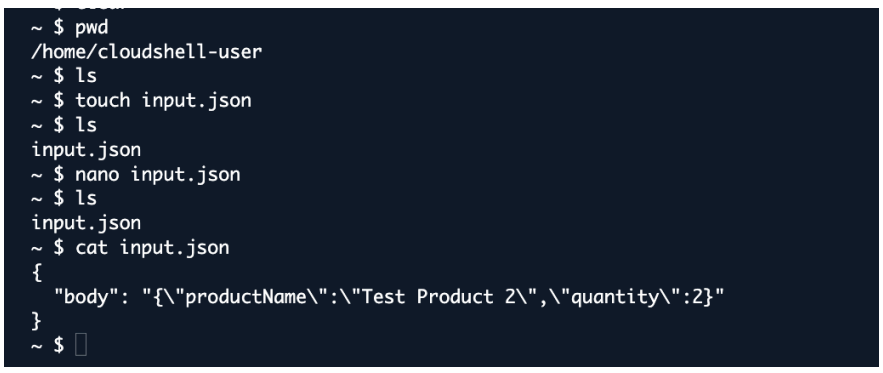
- Invoke the lambda function using the following command.

```
aws lambda invoke --function-name SubmitOrderFunction --payload fileb://input.json output.json
```



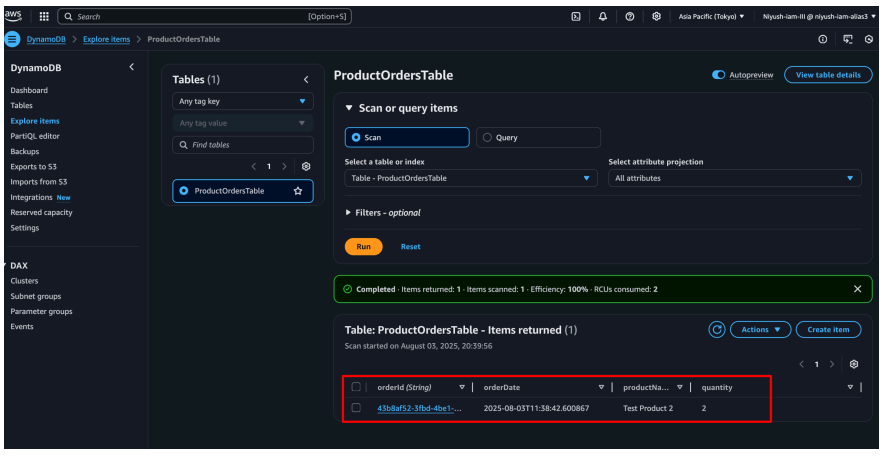


▼ [Image] input.json file



▼ [Image] invoke the function.

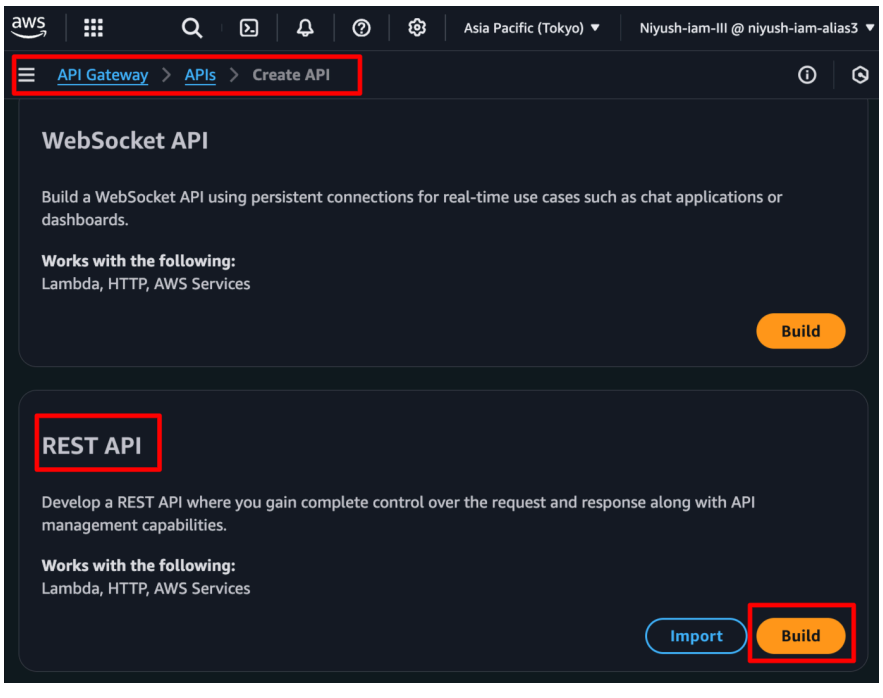
▼ [Image] check the invoked data in dynamoDB

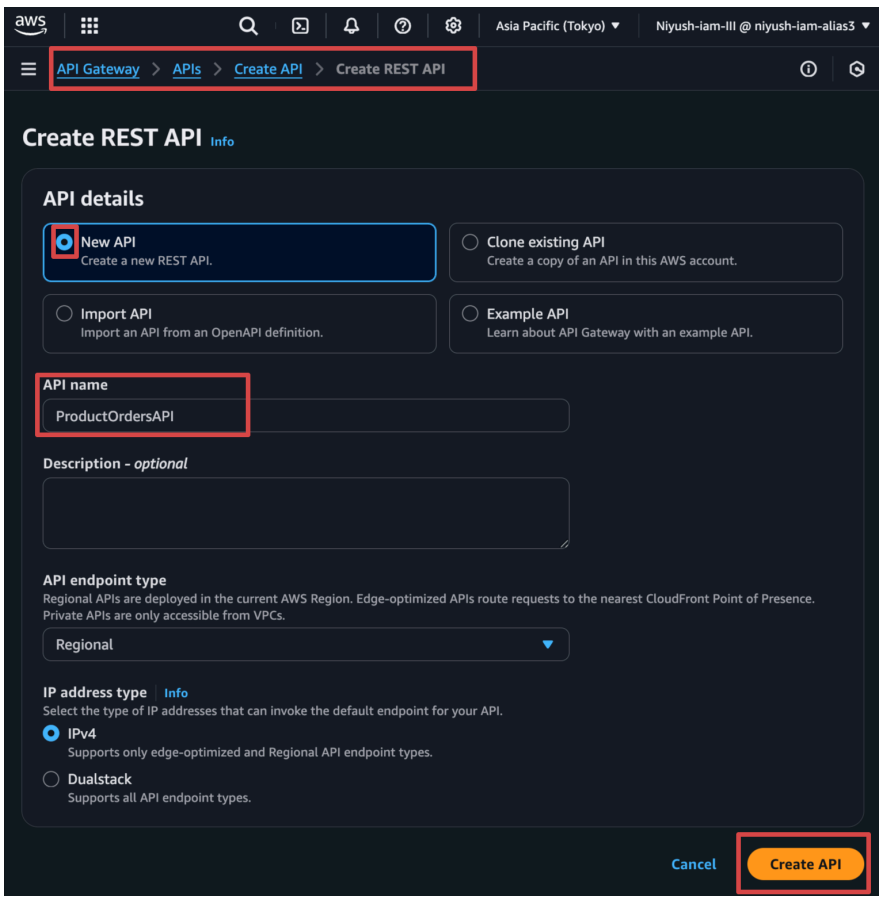


Step 5 : Creating the REST API

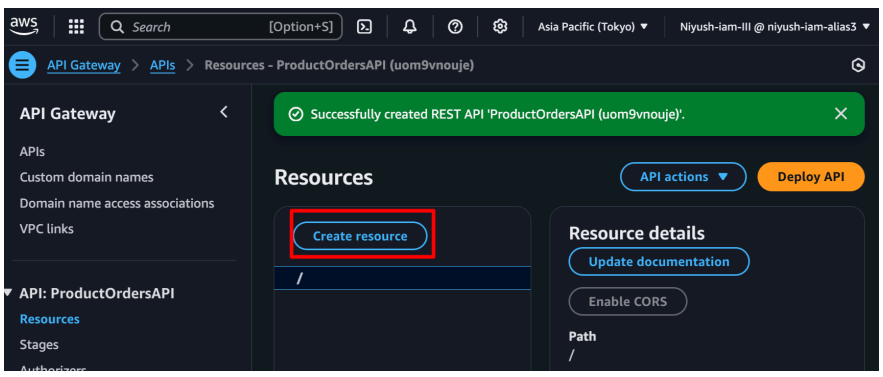
- Go to API Gateway → Create API → REST API
  API name : ProductOrdersAPI
- create a new resource name : orders
  enable : CORS
- create a POST method for /orders integrated with the SubmitOrderFunction
- Enter a Lambda proxy integration.
- Click back up to the /orders resource and click “enable CORS”
- Select all CORS options.
- Deploy API to a new stage named prod
- Update the invoke URL in index.html

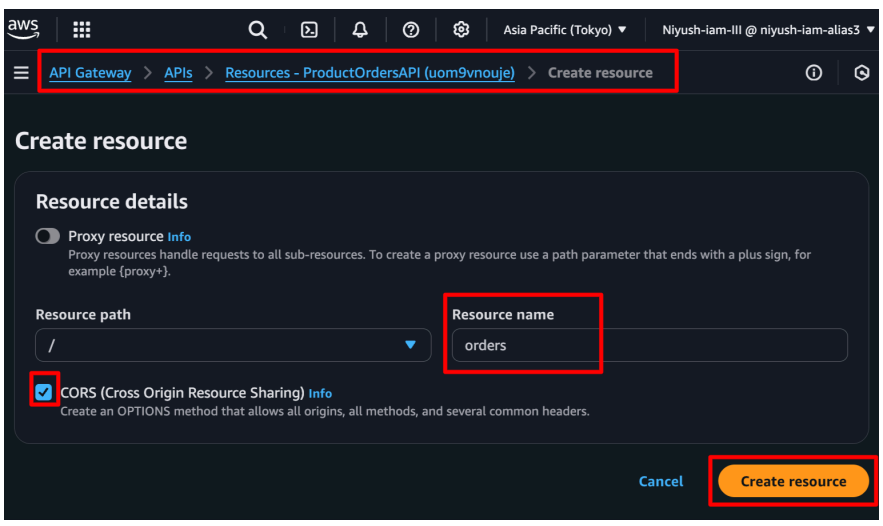
▼ [Image] Creating the API



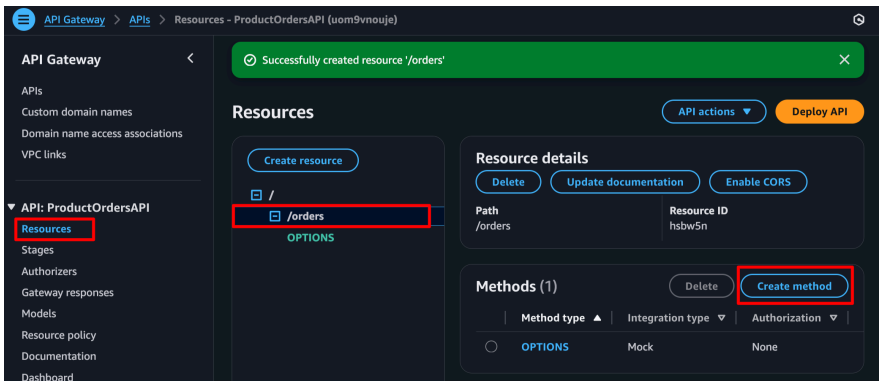


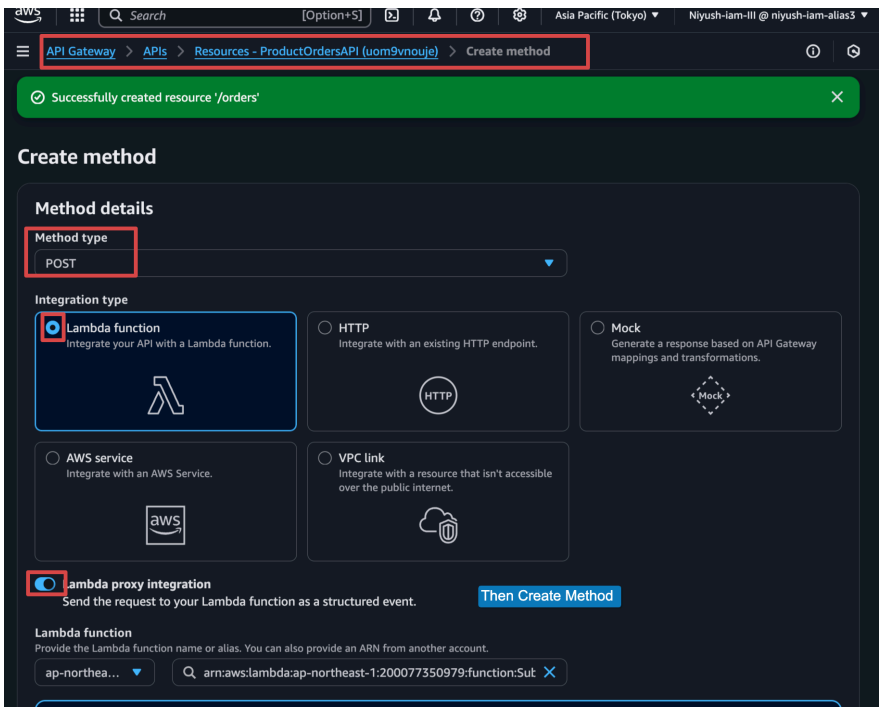
▼ [Image] Creating Resources



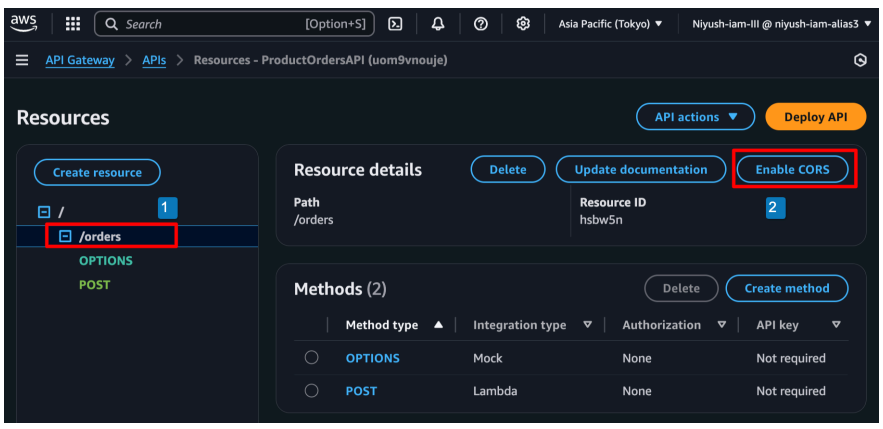


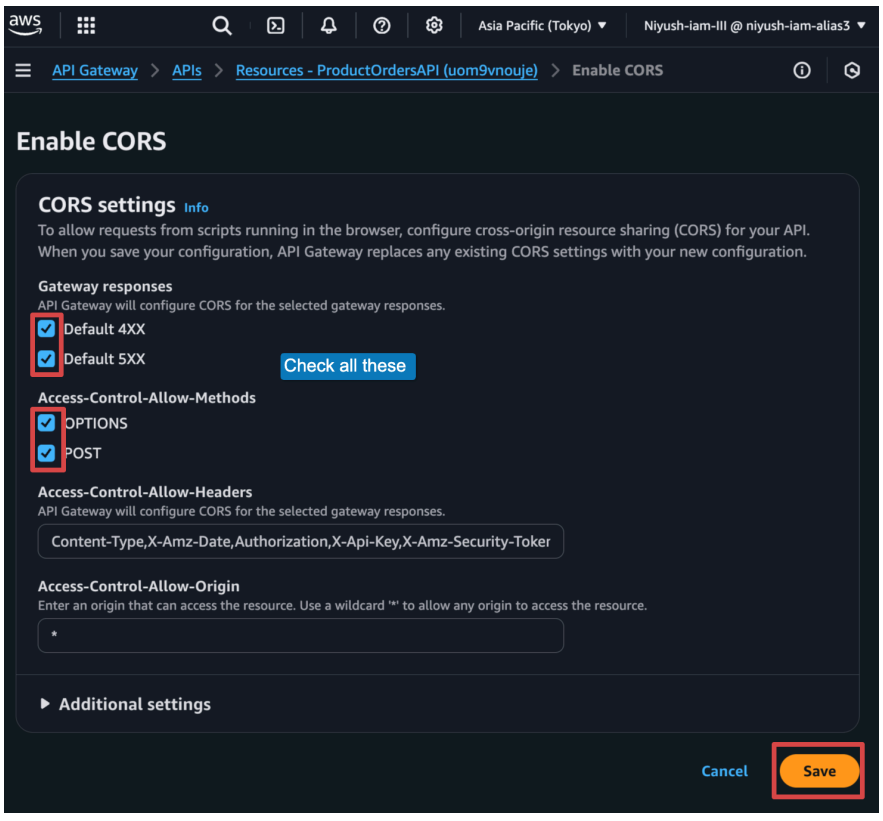
▼ [Image] Creating POSt method



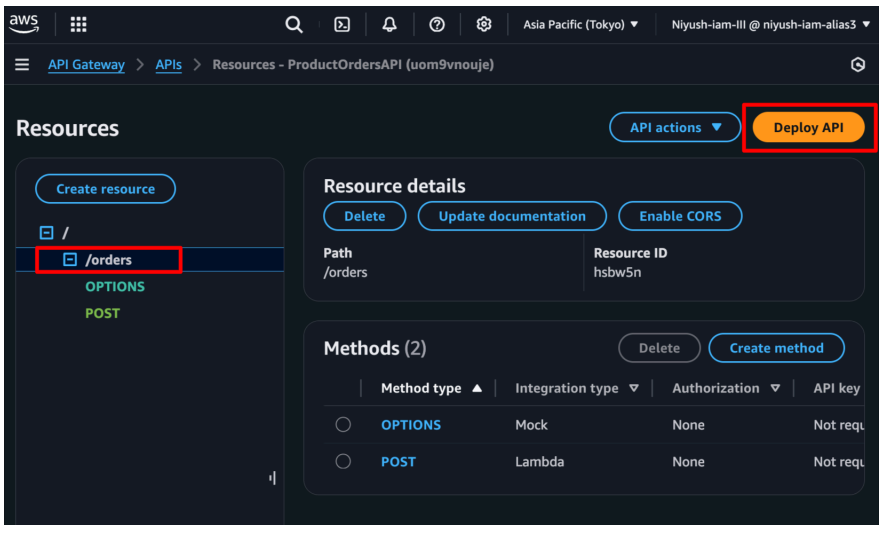


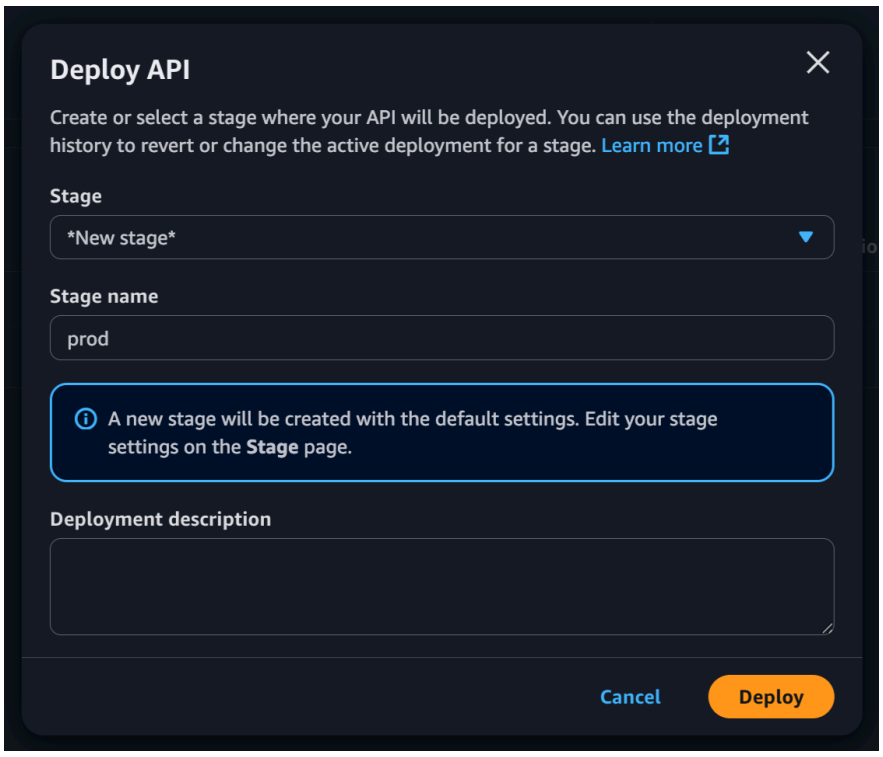
▼ [Image] Enabling CORS

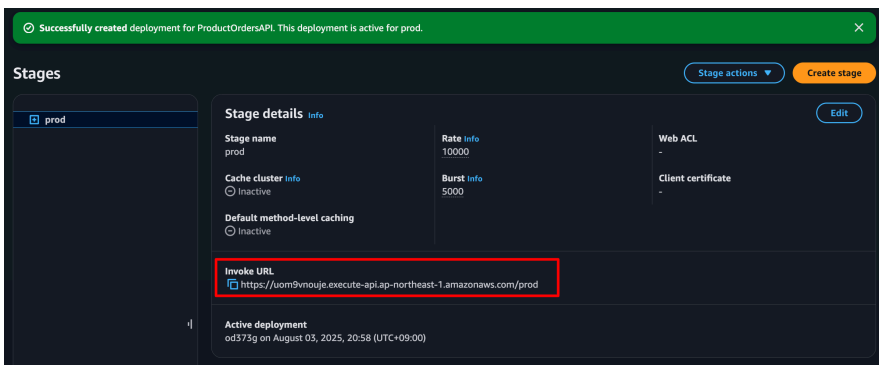




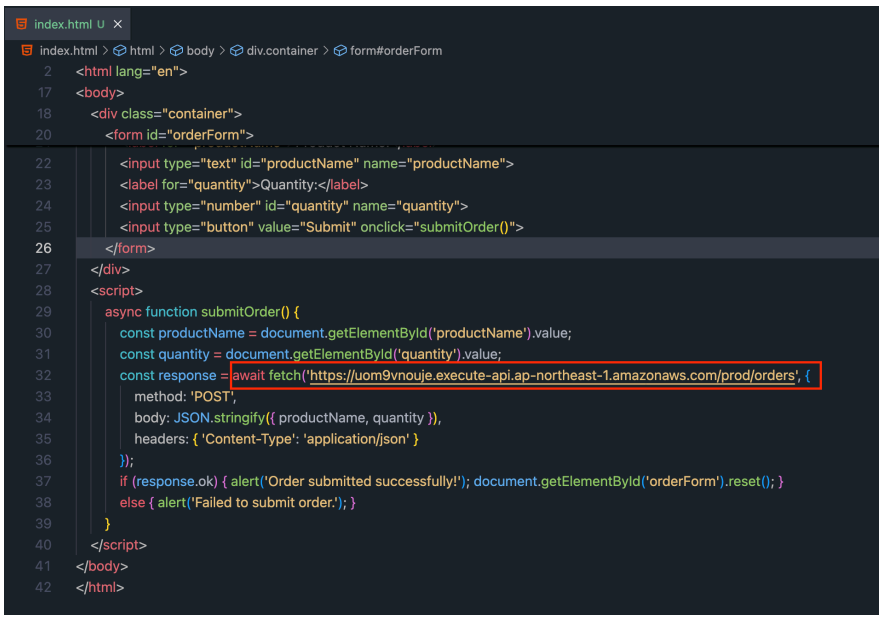
▼ [Image] Deploy API and update the index.html







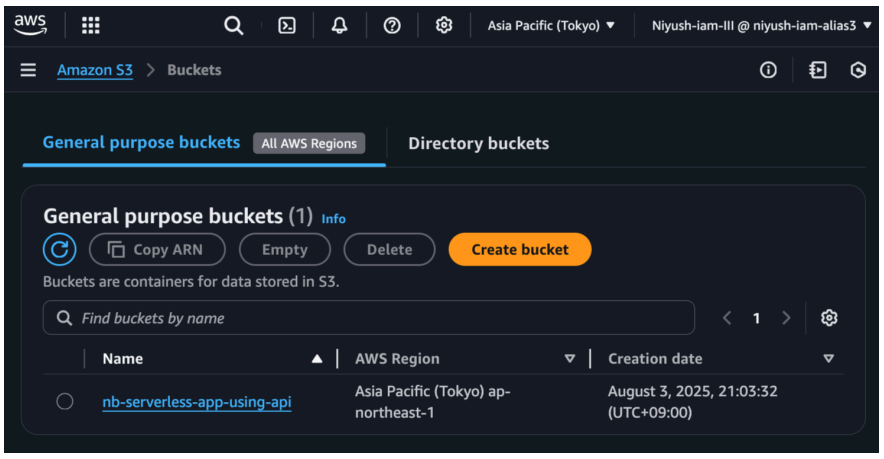
▼ [Image] Update the API of index.html



Step 6 : Create the S3 bucket

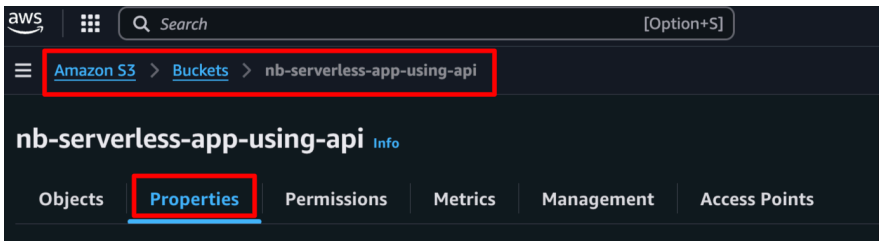
- Go to the S3 bucket console.
- Name the bucket
- Configure the bucket for static website for hosting.
- Make sure to set default document to `index.html`
- updated `index.html`
- Enable public access using a bucket policy.

▼ [Image] Creating bucket and configure settings



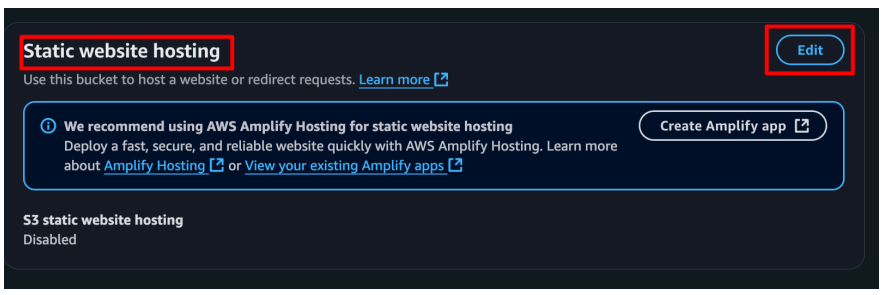
▼ **The required policy to allow public access.**

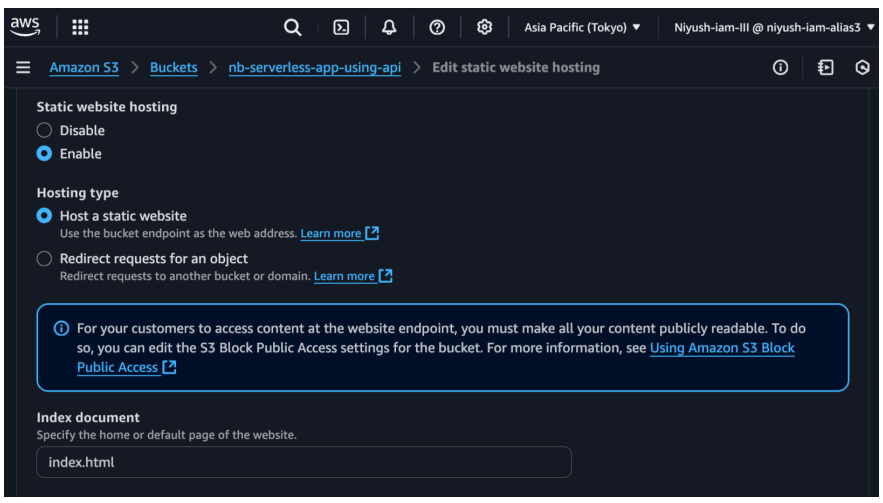
```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "PublicReadGetObject",
         "Effect": "Allow",
         "Principal": "*",
         "Action": "s3:GetObject",
         "Resource": "<YOUR-BUCKET-ARN>/*"
      }
```



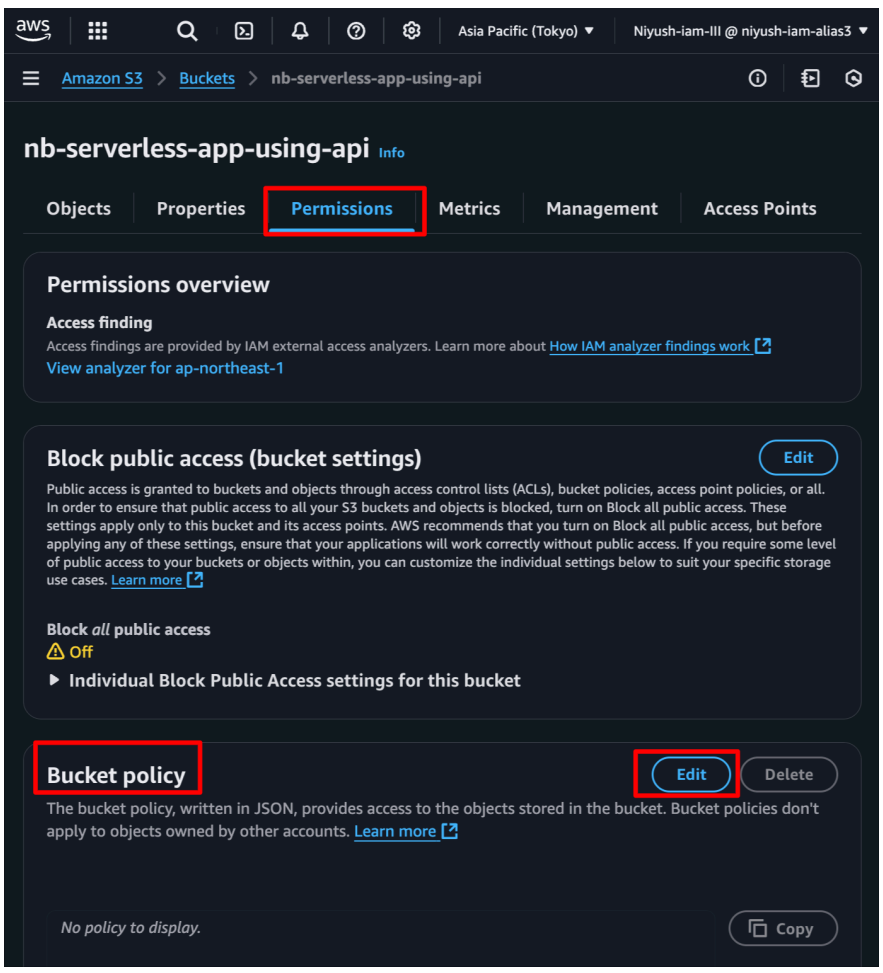
```
    ]
}
```

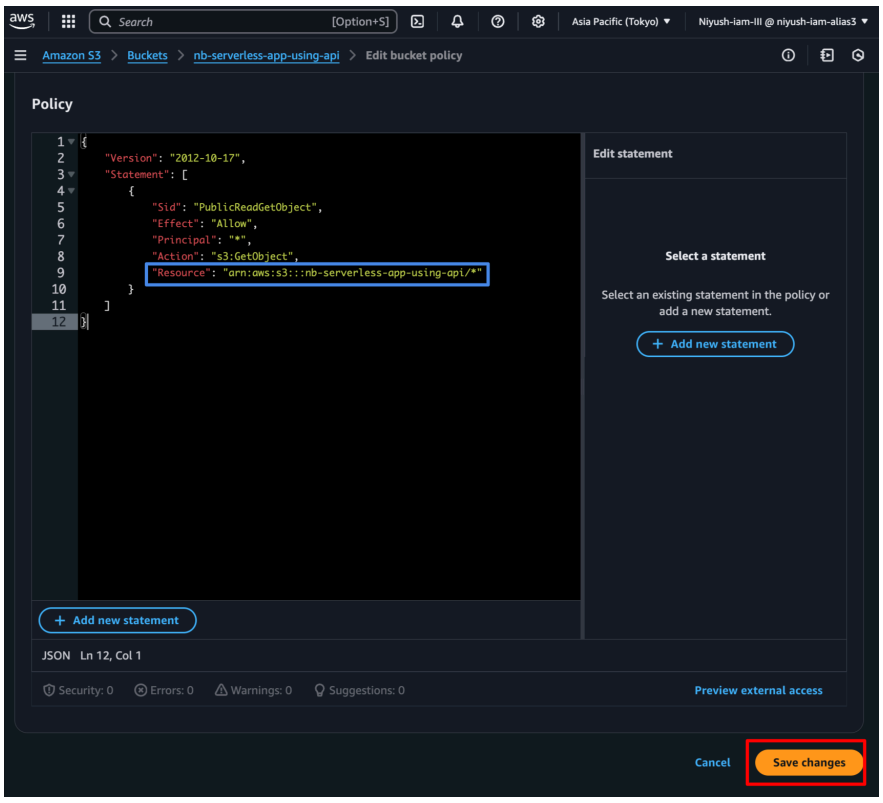
- once all done finally upload the updated index.html
- Once the upload is completed head to `properites` → `static website hosting` → click on `Bucket website endpoint`



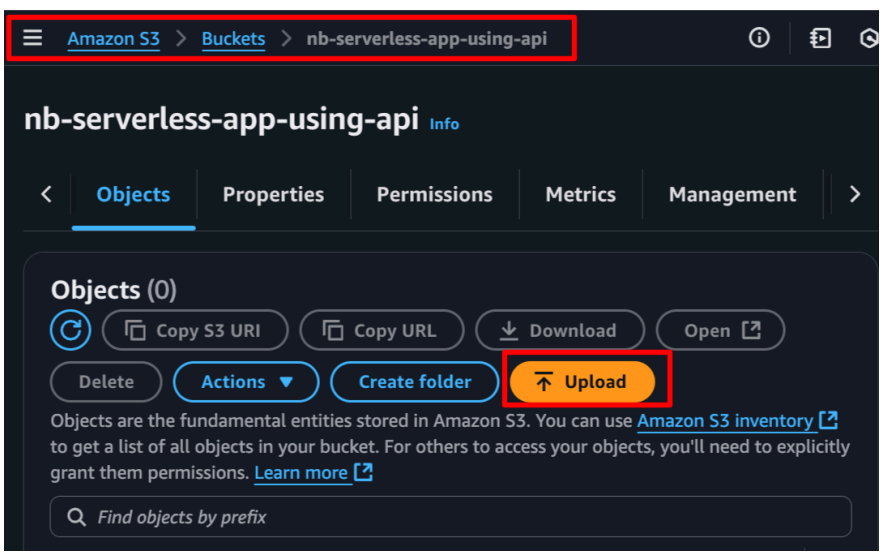


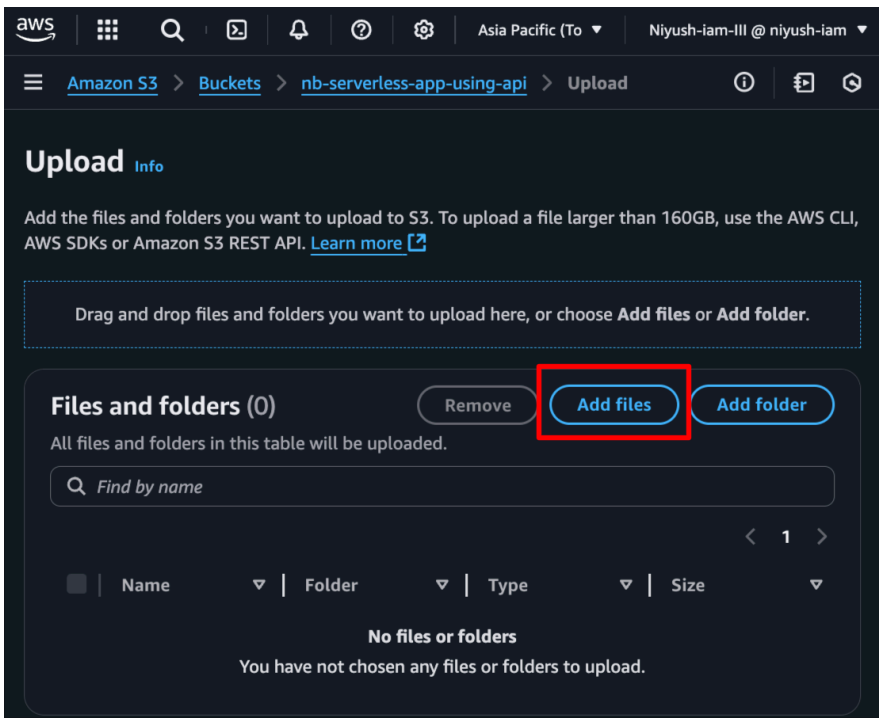
▼ `[Image]` adding permission and editing bucket policy

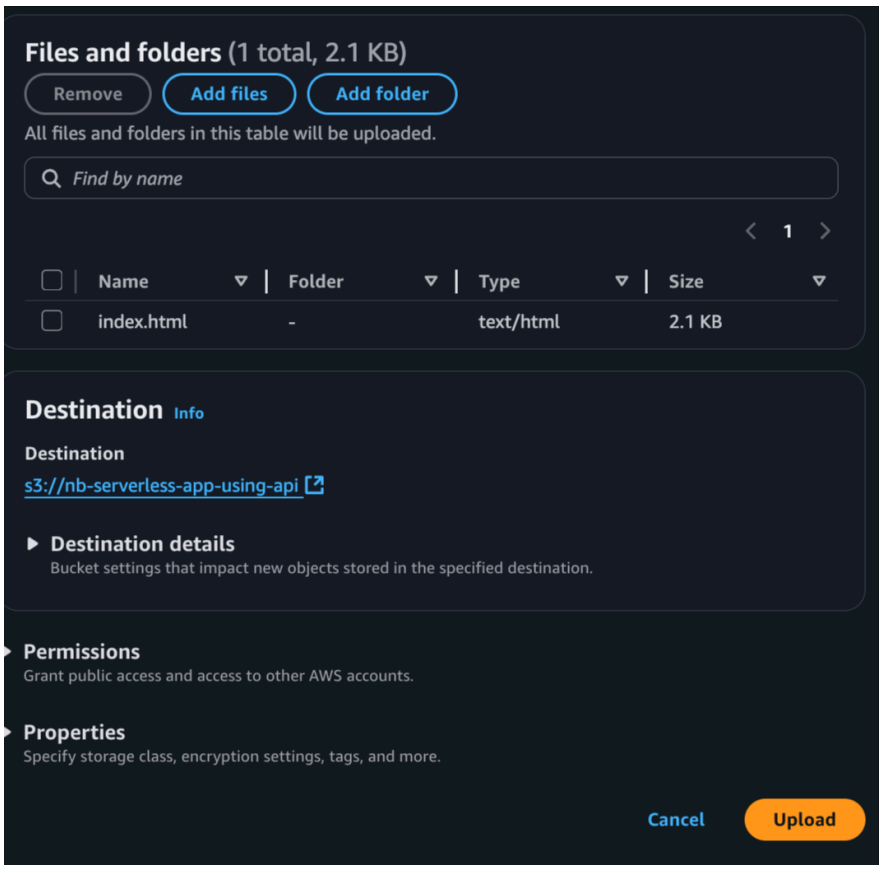




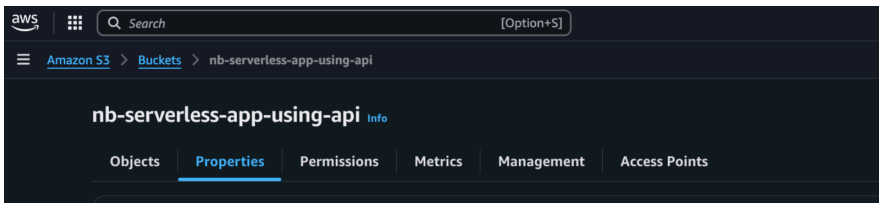
▼ [Image] upload the index.html

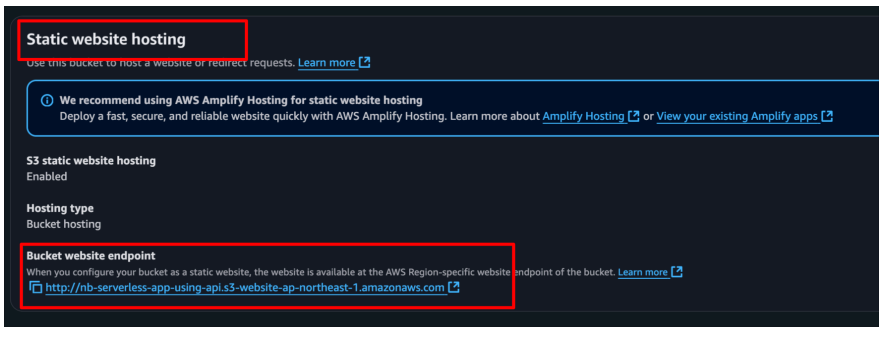


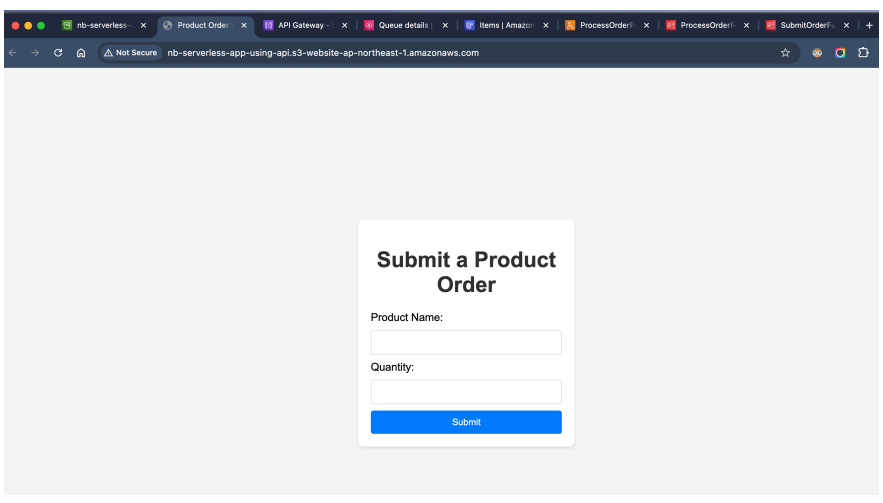




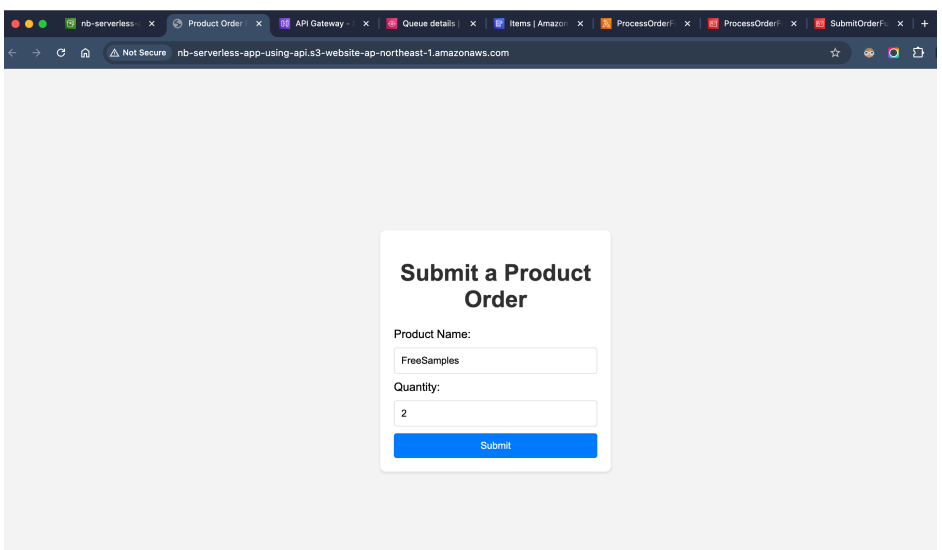
▼ [Image] Accessing the static website

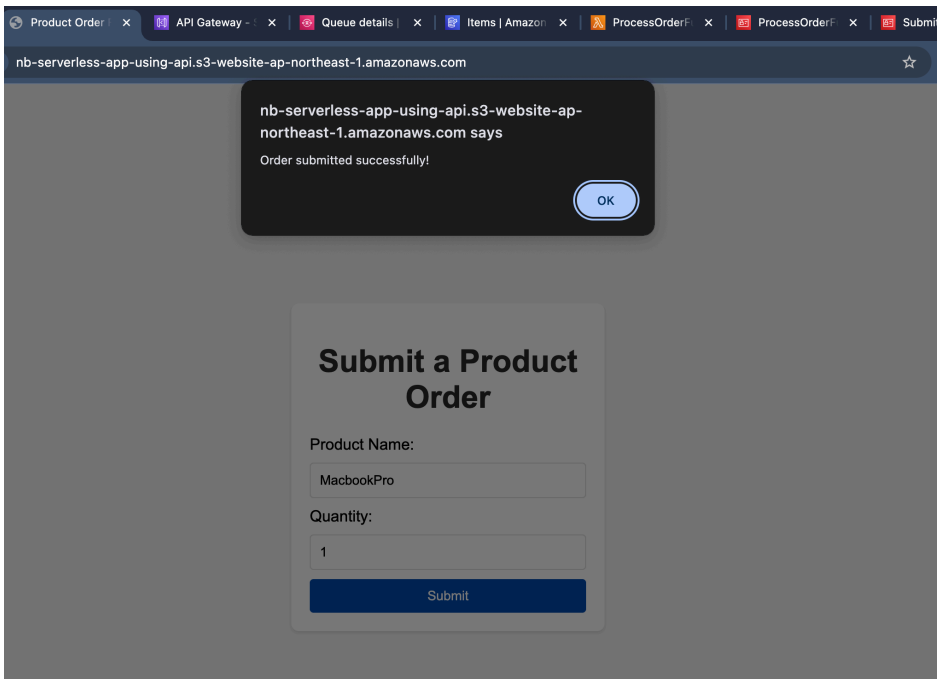






Step 7 : Testing time

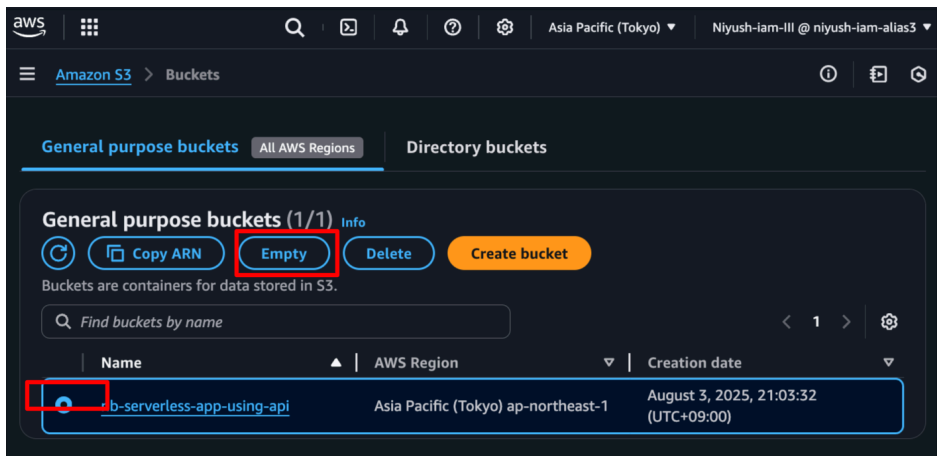


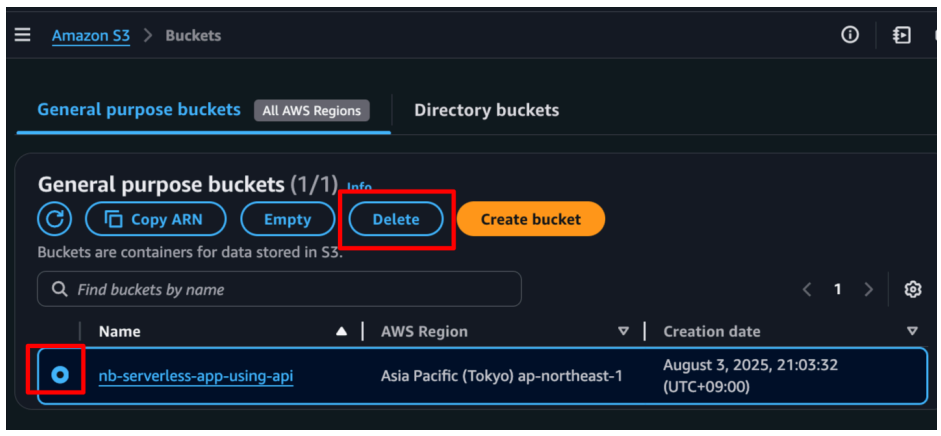


- Head to DynamoDb and check if the orders are successfully stored or not

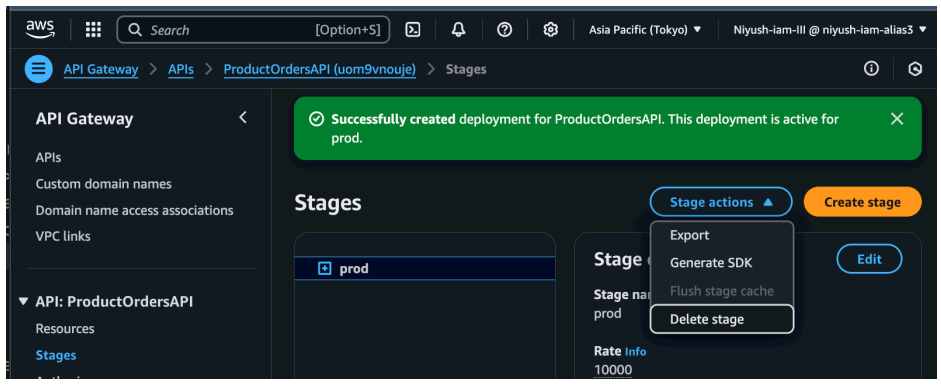
▼ [Image] Check the orders in DynamoDB

Step 8 : Delete the resources

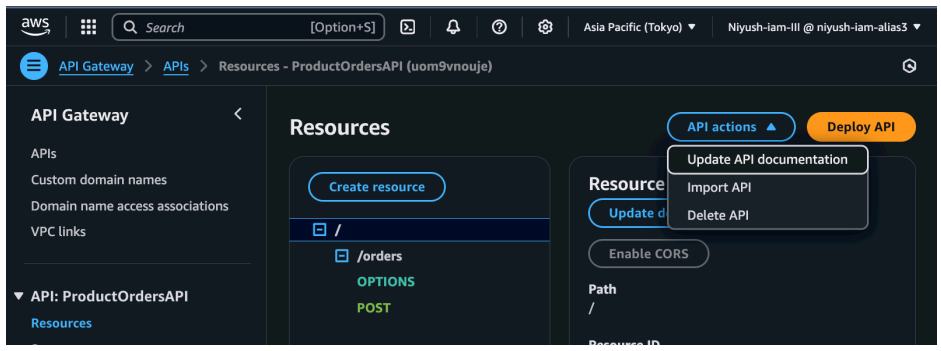




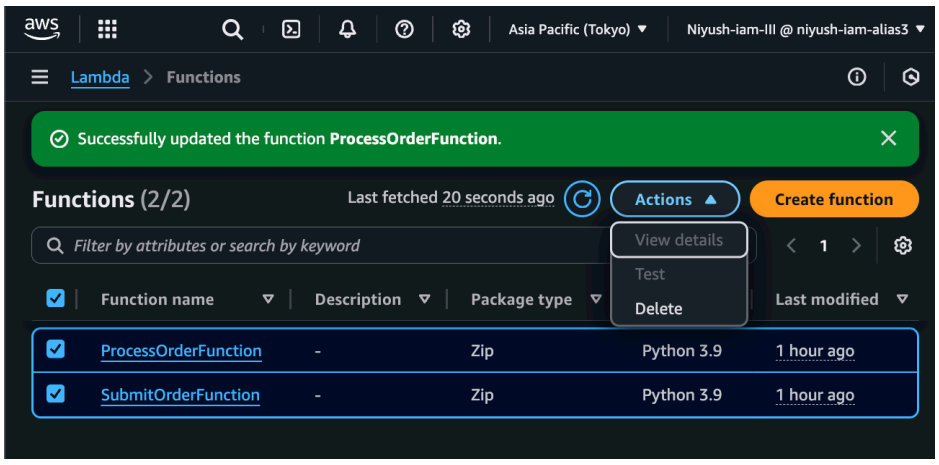
deleting the stage

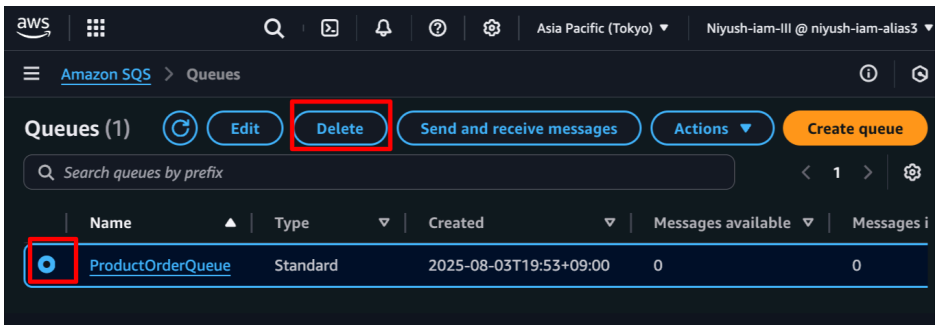


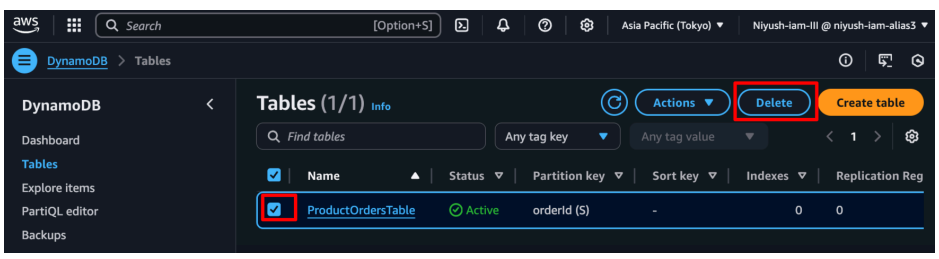
deleting the resources

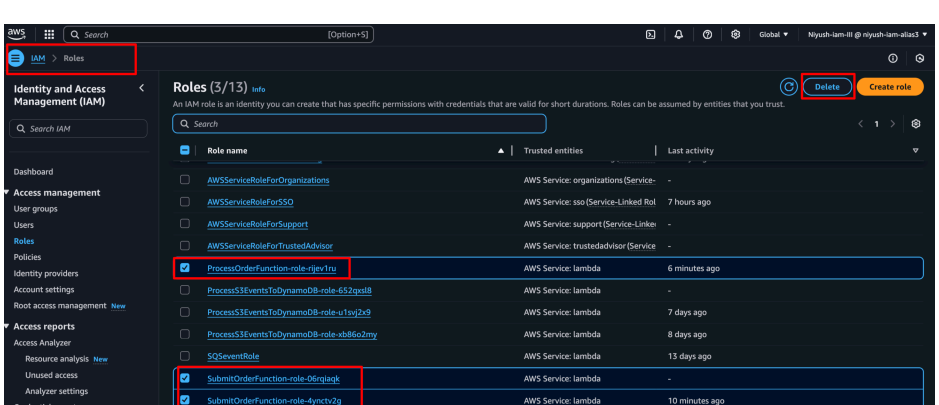


deleting lambda functions.









And that’s a wrap!!!!!!!!!!!

niyush

buy me some caffeine

http://buymeacoffee.com/niyushbjr1L