

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3:\*",

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents",

"sqs:ReceiveMessage",

"sqs:DeleteMessage",

"sqs:GetQueueAttributes",

"rekognition:DetectLabels",

"dynamodb:PutItem",

"comprehend:DetectSentiment",

"textract:StartDocumentTextDetection",

"textract:GetDocumentTextDetection"

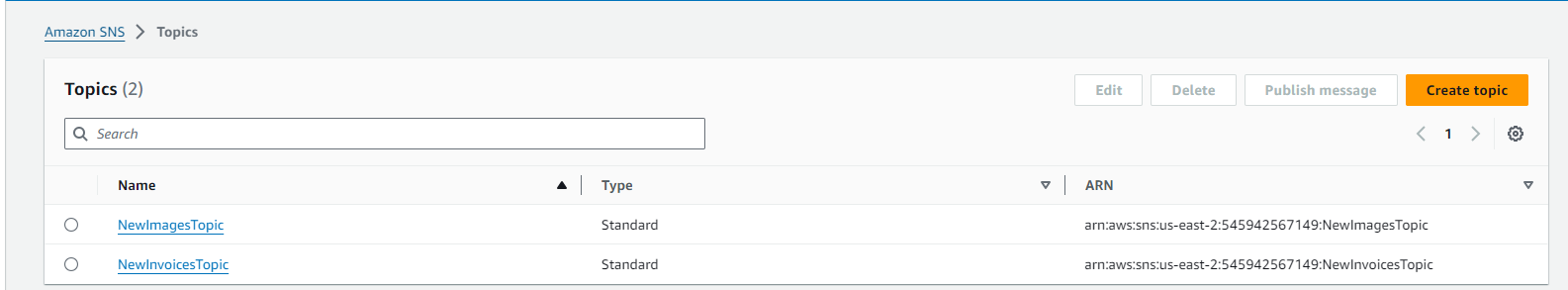
],

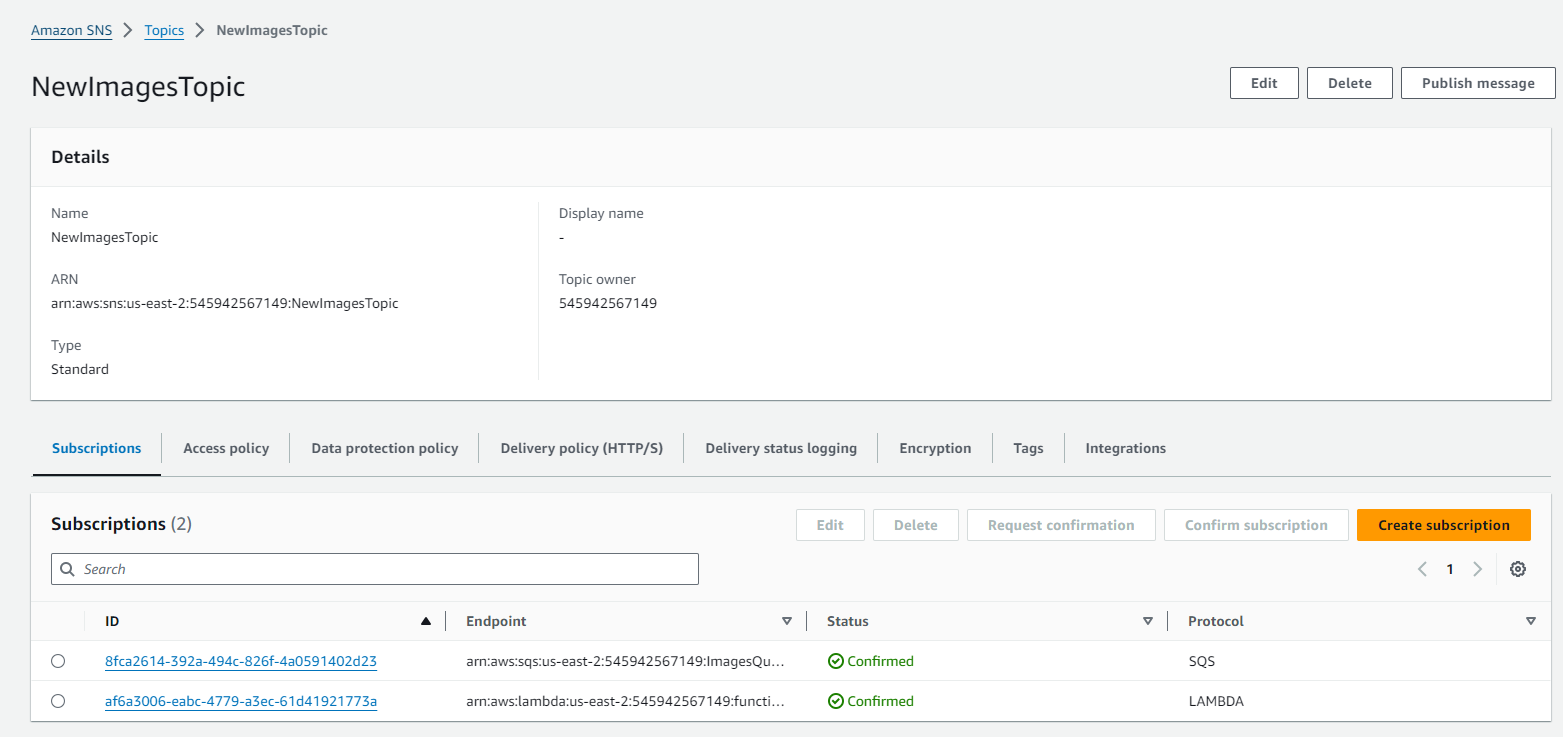
"Resource": "\*"

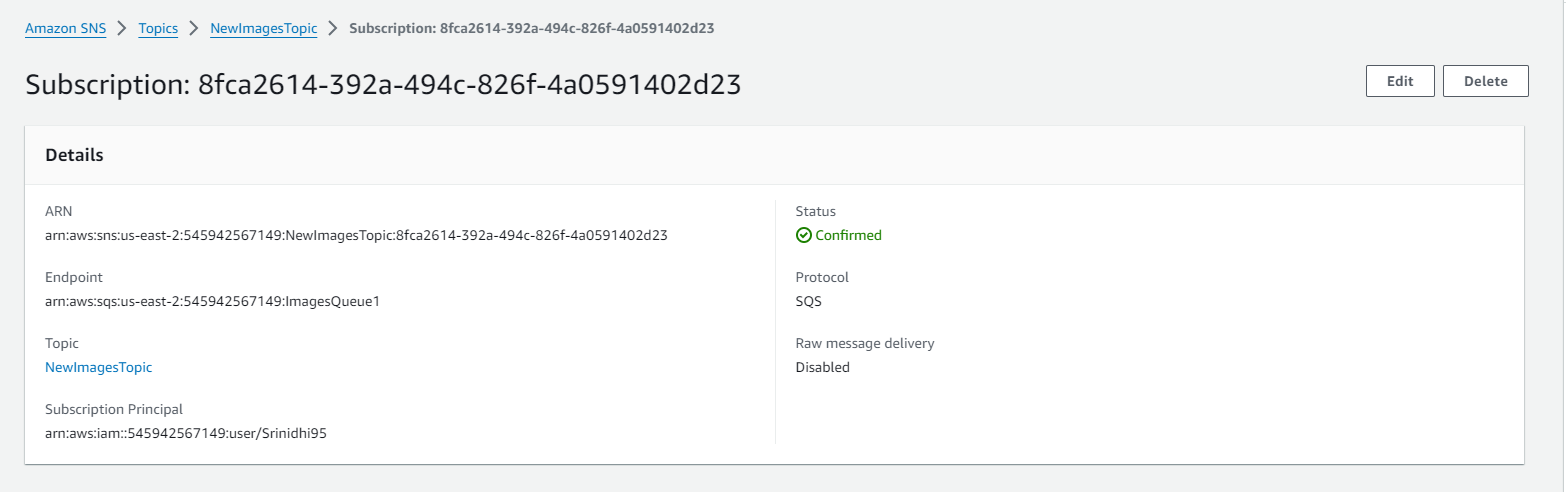
}

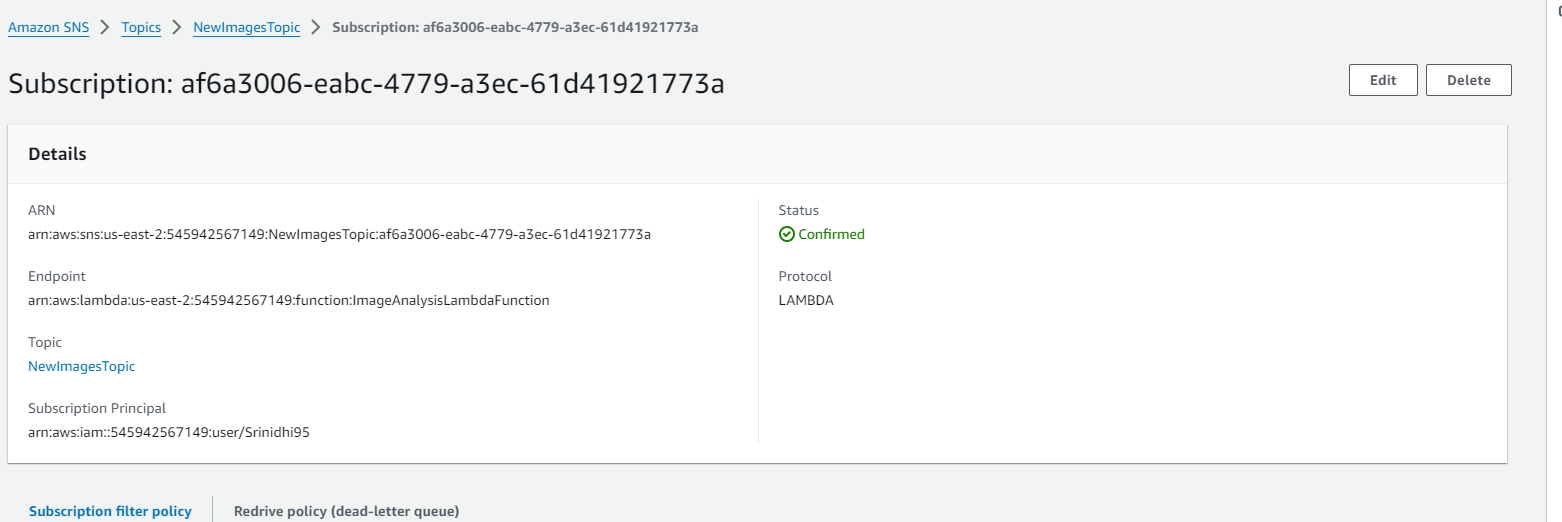
]

}











{

"Version": "2012-10-17",

"Id": "\_\_default\_policy\_ID",

"Statement": [

{

"Sid": "\_\_default\_statement\_ID",

"Effect": "Allow",

"Principal": {

"Service": "s3.amazonaws.com"

},

"Action": "sns:Publish",

"Resource": "arn:aws:sns:us-east-2:545942567149:NewImagesTopic",

"Condition": {

"ArnLike": {

"aws:SourceArn": "arn:aws:s3:::my-images-bucket-srinidhi"

}

}

},

{

"Sid": "OtherSNSActions",

"Effect": "Allow",

"Principal": {

"AWS": "\*"

},

"Action": [

"SNS:GetTopicAttributes",

"SNS:SetTopicAttributes",

"SNS:AddPermission",

"SNS:RemovePermission",

"SNS:DeleteTopic",

"SNS:Subscribe",

"SNS:ListSubscriptionsByTopic"

],

"Resource": "arn:aws:sns:us-east-2:545942567149:NewImagesTopic",

"Condition": {

"StringEquals": {

"AWS:SourceOwner": "545942567149"

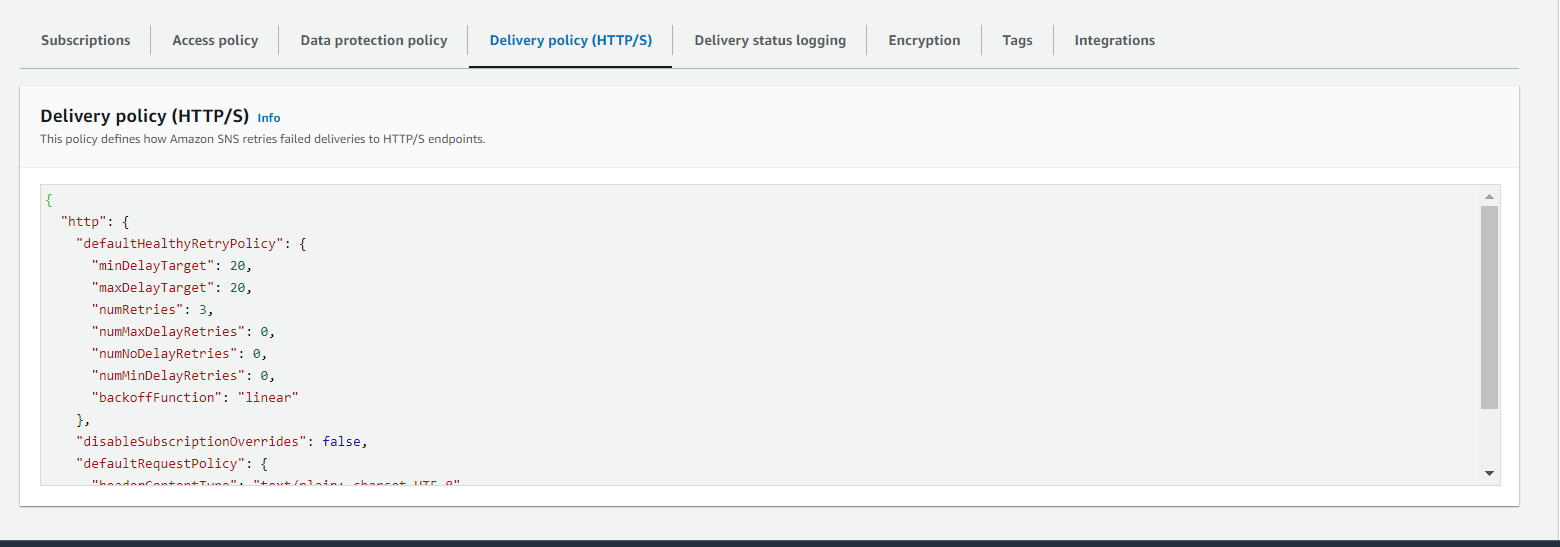
}

}

}

]

}



{

"http": {

"defaultHealthyRetryPolicy": {

"minDelayTarget": 20,

"maxDelayTarget": 20,

"numRetries": 3,

"numMaxDelayRetries": 0,

"numNoDelayRetries": 0,

"numMinDelayRetries": 0,

"backoffFunction": "linear"

},

"disableSubscriptionOverrides": false,

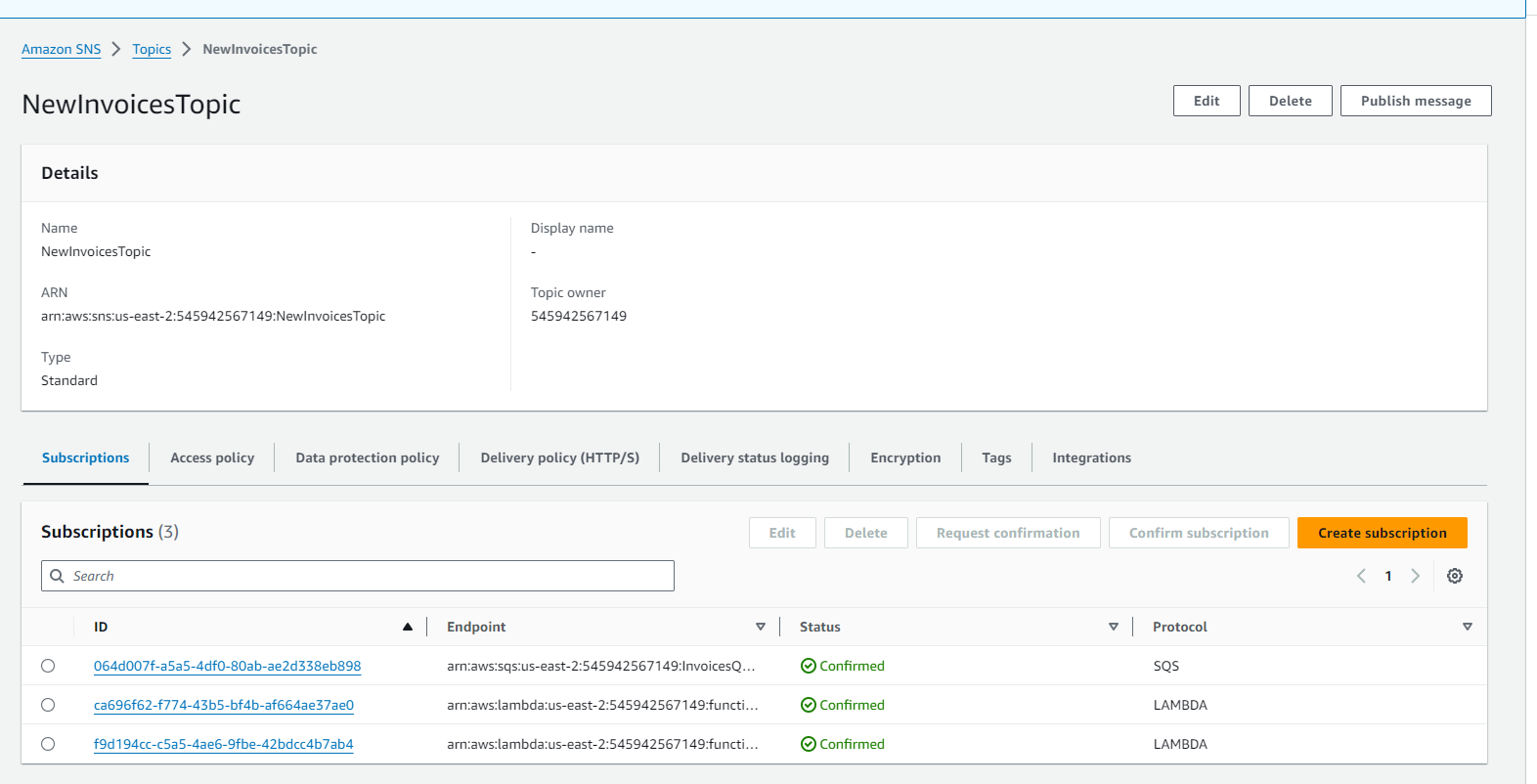
"defaultRequestPolicy": {

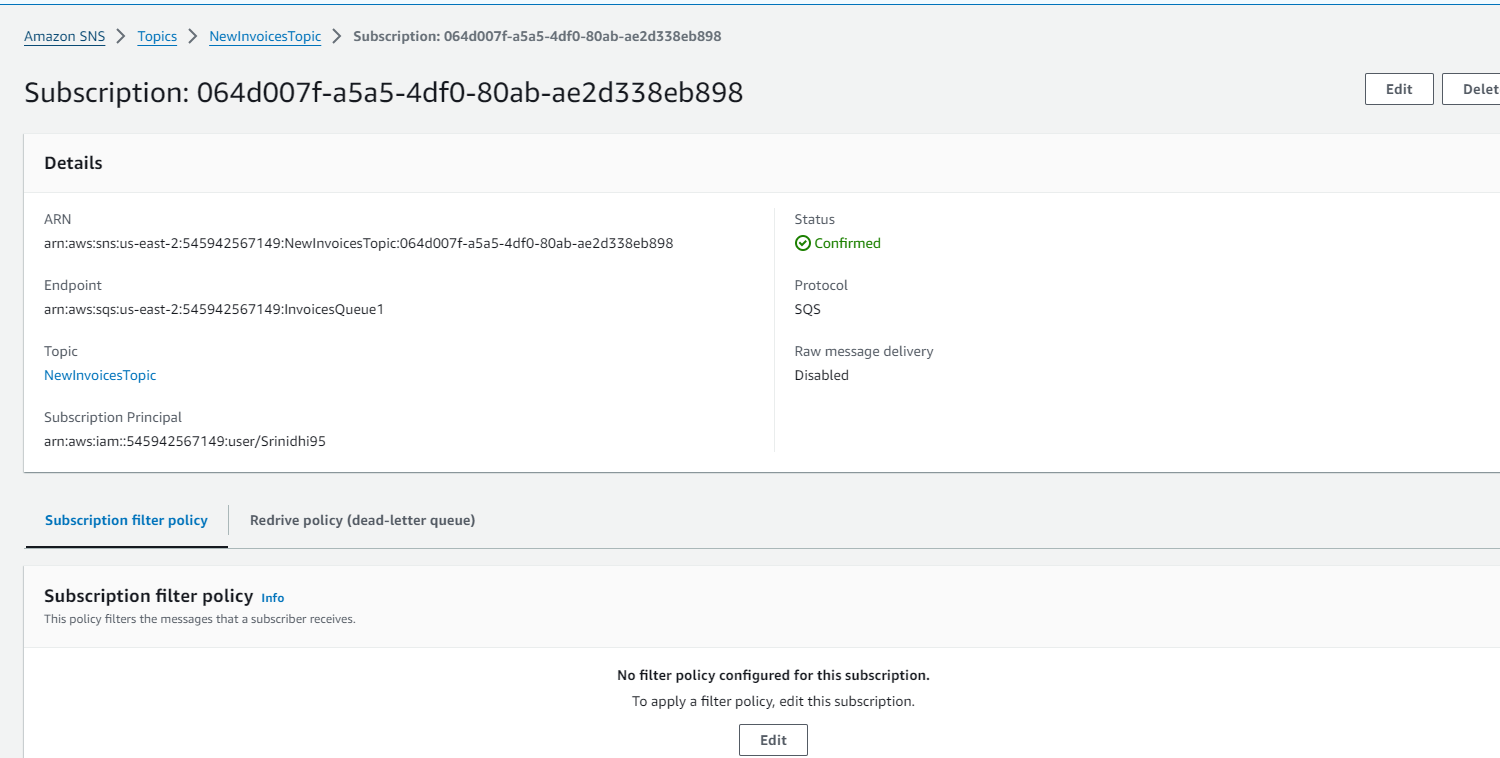
"headerContentType": "text/plain; charset=UTF-8"

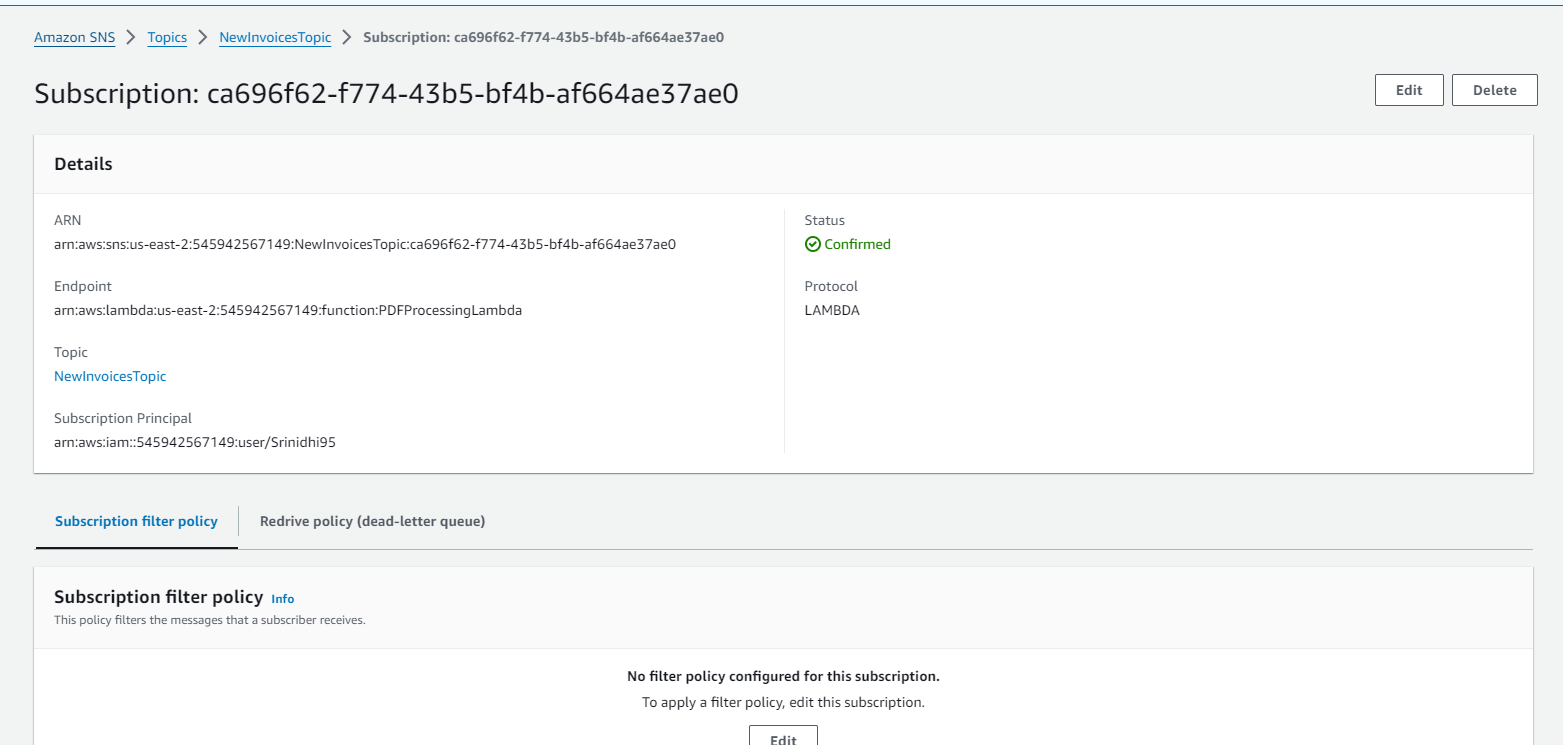
}

}

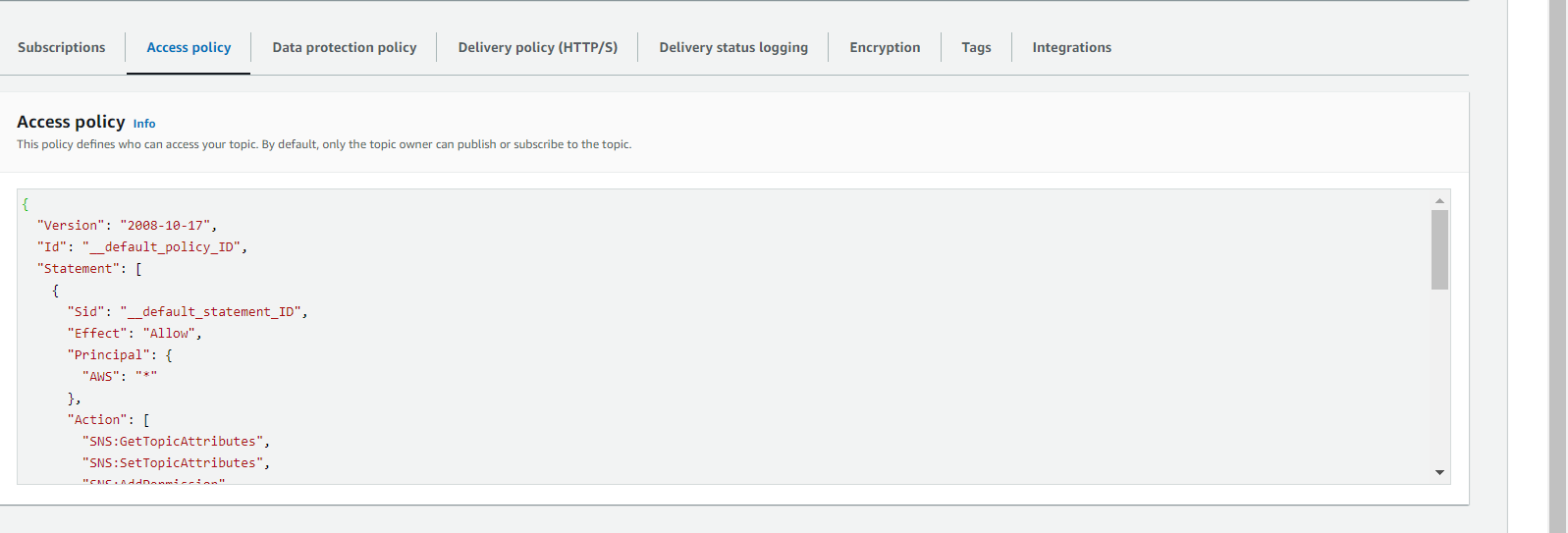
}











{

"Version": "2008-10-17",

"Id": "\_\_default\_policy\_ID",

"Statement": [

{

"Sid": "\_\_default\_statement\_ID",

"Effect": "Allow",

"Principal": {

"AWS": "\*"

},

"Action": [

"SNS:GetTopicAttributes",

"SNS:SetTopicAttributes",

"SNS:AddPermission",

"SNS:RemovePermission",

"SNS:DeleteTopic",

"SNS:Subscribe",

"SNS:ListSubscriptionsByTopic",

"SNS:Publish"

],

"Resource": "arn:aws:sns:us-east-2:545942567149:NewInvoicesTopic",

"Condition": {

"StringEquals": {

"AWS:SourceOwner": "545942567149"

}

}

},

{

"Sid": "AllowS3BucketPublish",

"Effect": "Allow",

"Principal": {

"Service": "s3.amazonaws.com"

},

"Action": "sns:Publish",

"Resource": "arn:aws:sns:us-east-2:545942567149:NewInvoicesTopic",

"Condition": {

"ArnLike": {

"aws:SourceArn": "arn:aws:s3:::my-invoices-bucket-srinidhi"

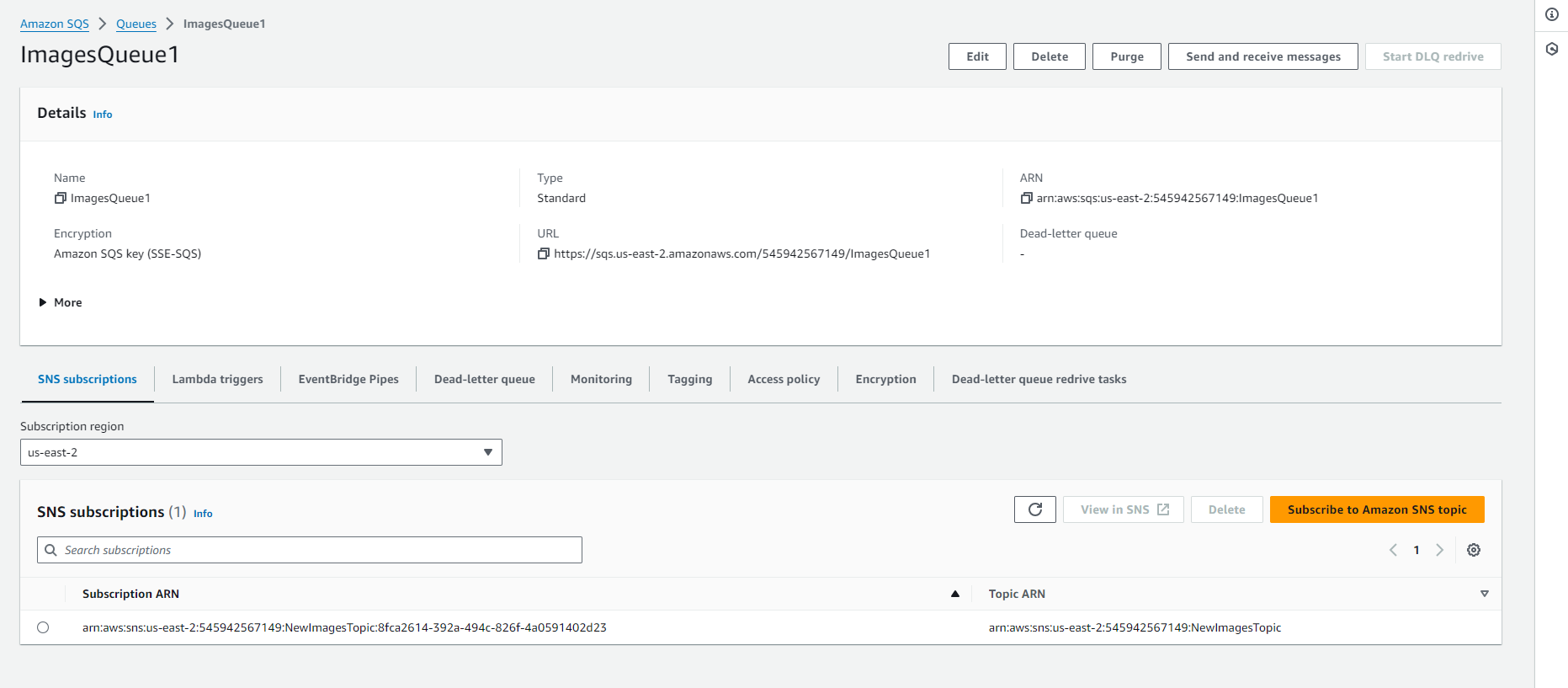
}

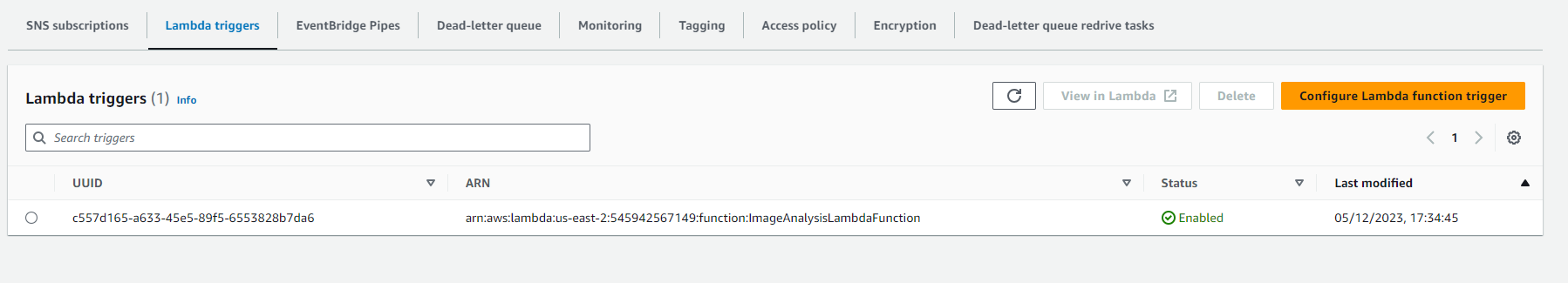
}

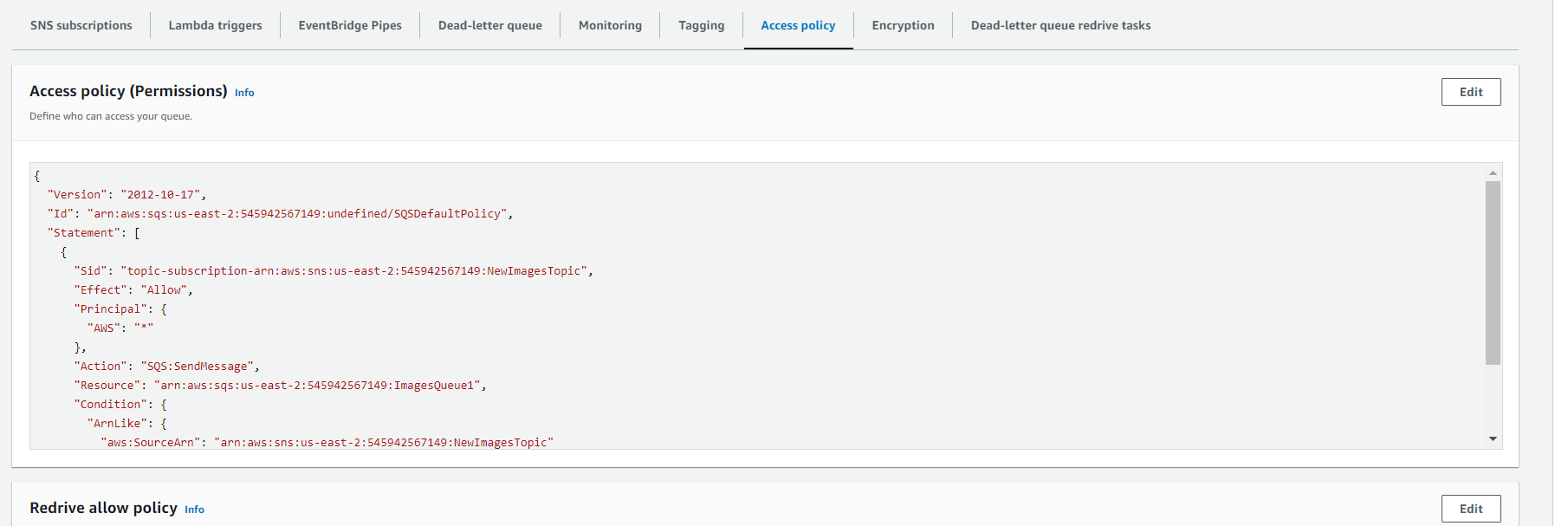
}

]

}







{

 "Version": "2012-10-17",

 "Id": "arn:aws:sqs:us-east-2:545942567149:undefined/SQSDefaultPolicy",

 "Statement": [

  {

     "Sid": "topic-subscription-arn:aws:sns:us-east-2:545942567149:NewImagesTopic",

     "Effect": "Allow",

     "Principal": {

       "AWS": "\*"

    },

     "Action": "SQS:SendMessage",

     "Resource": "arn:aws:sqs:us-east-2:545942567149:ImagesQueue1",

     "Condition": {

       "ArnLike": {

         "aws:SourceArn": "arn:aws:sns:us-east-2:545942567149:NewImagesTopic"

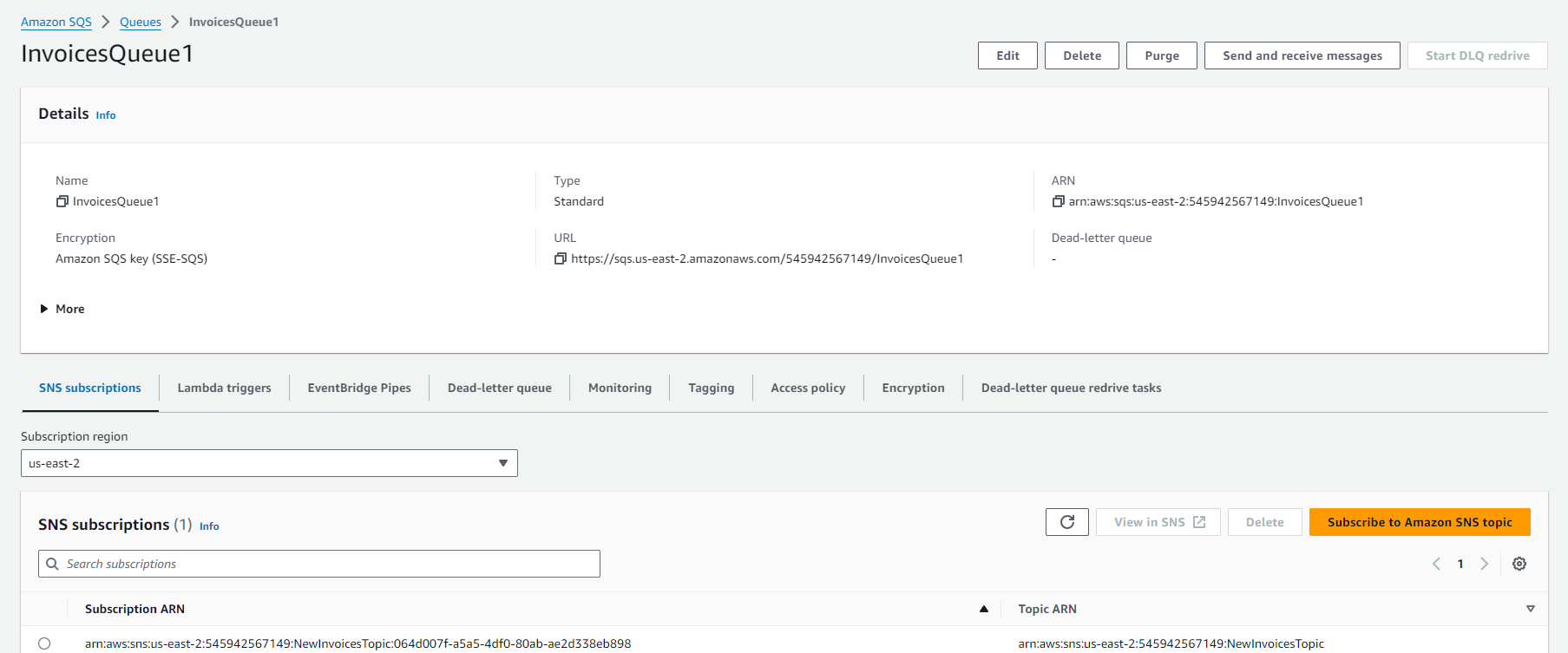
      }

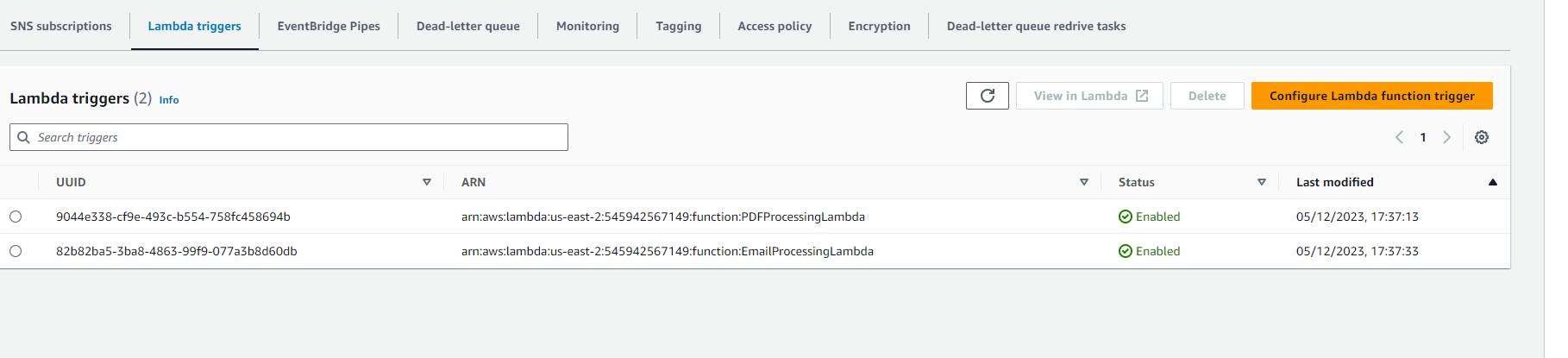
    }

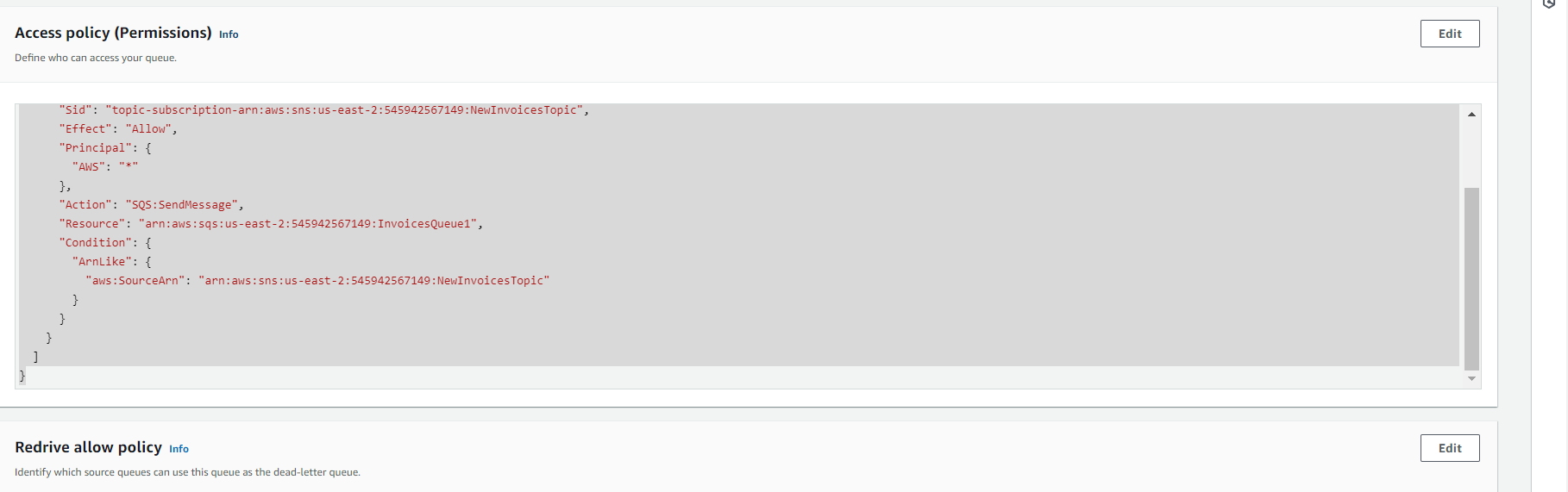
  }

]

}







{

"Version": "2012-10-17",

"Id": "arn:aws:sqs:us-east-2:545942567149:undefined/SQSDefaultPolicy",

"Statement": [

{

"Sid": "topic-subscription-arn:aws:sns:us-east-2:545942567149:NewInvoicesTopic",

"Effect": "Allow",

"Principal": {

"AWS": "\*"

},

"Action": "SQS:SendMessage",

"Resource": "arn:aws:sqs:us-east-2:545942567149:InvoicesQueue1",

"Condition": {

"ArnLike": {

"aws:SourceArn": "arn:aws:sns:us-east-2:545942567149:NewInvoicesTopic"

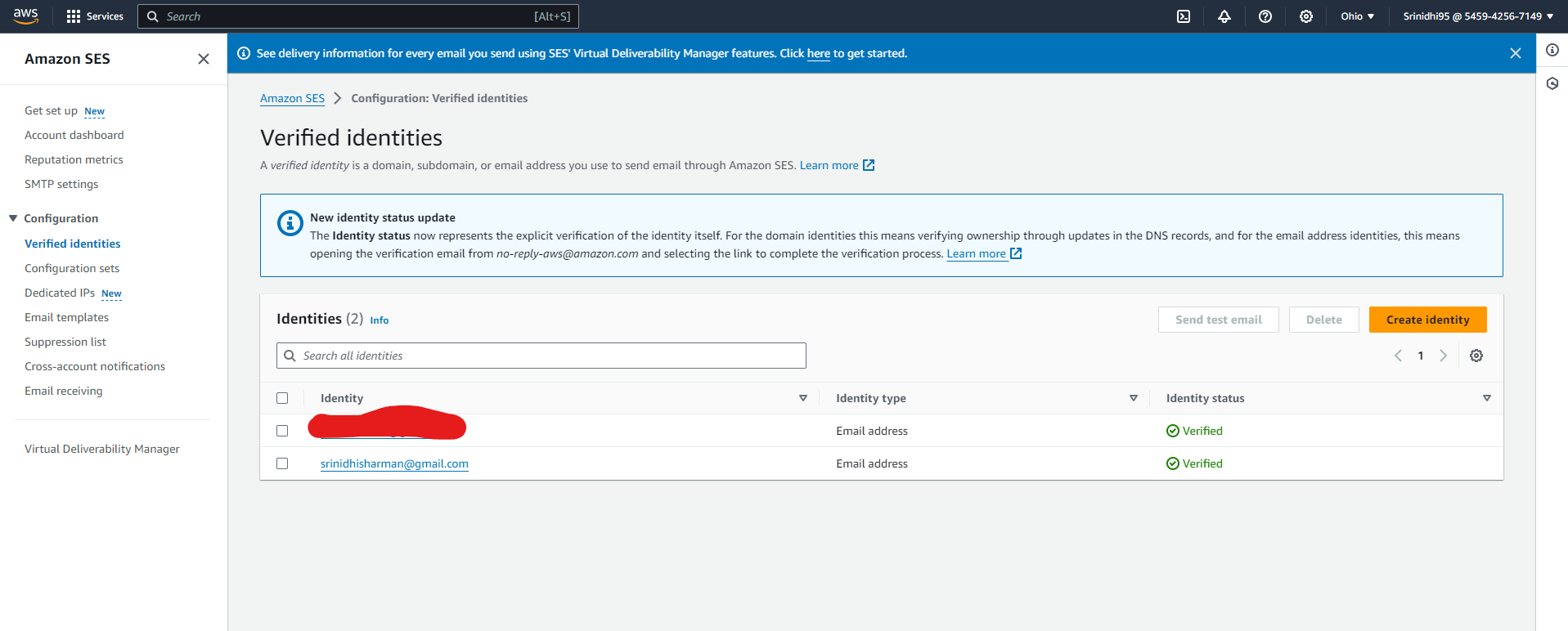
}

}

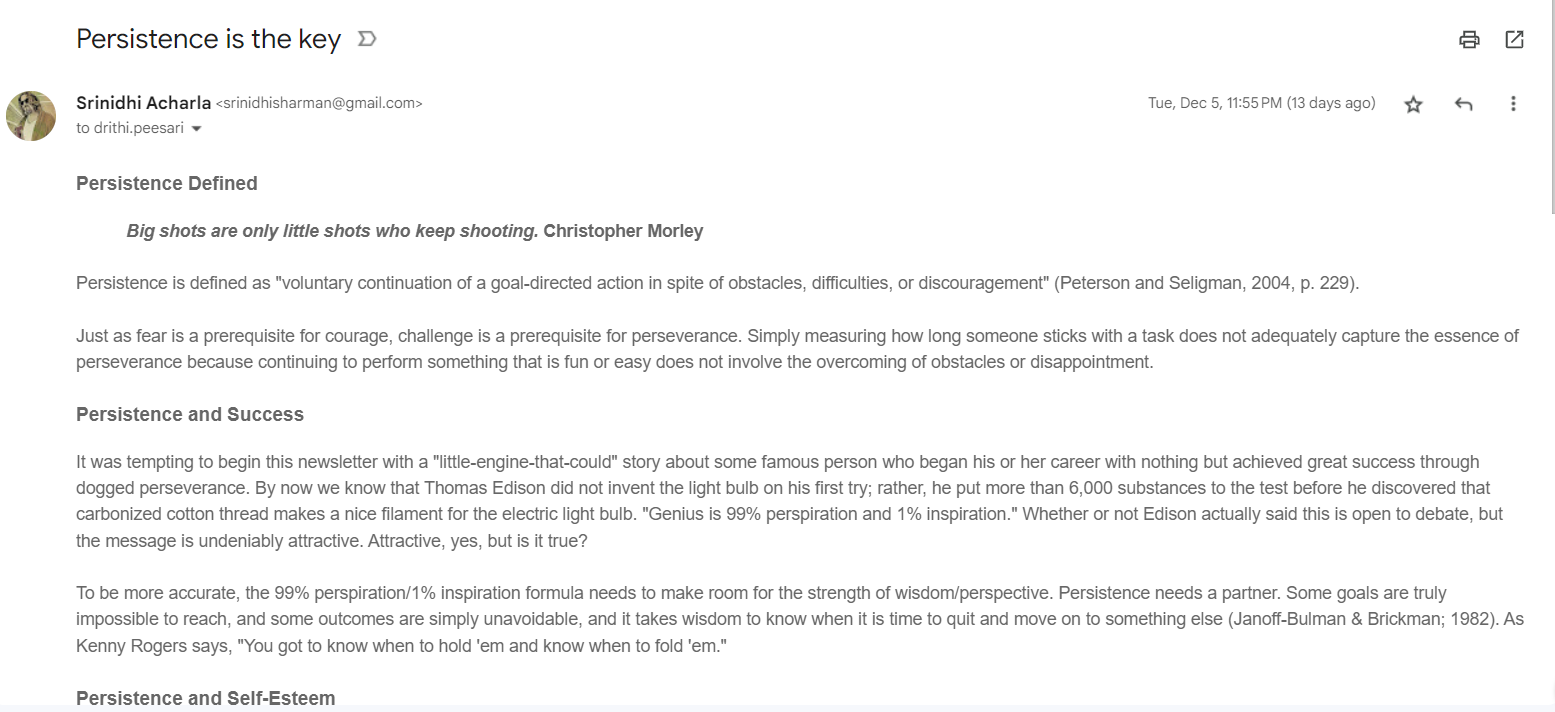
}

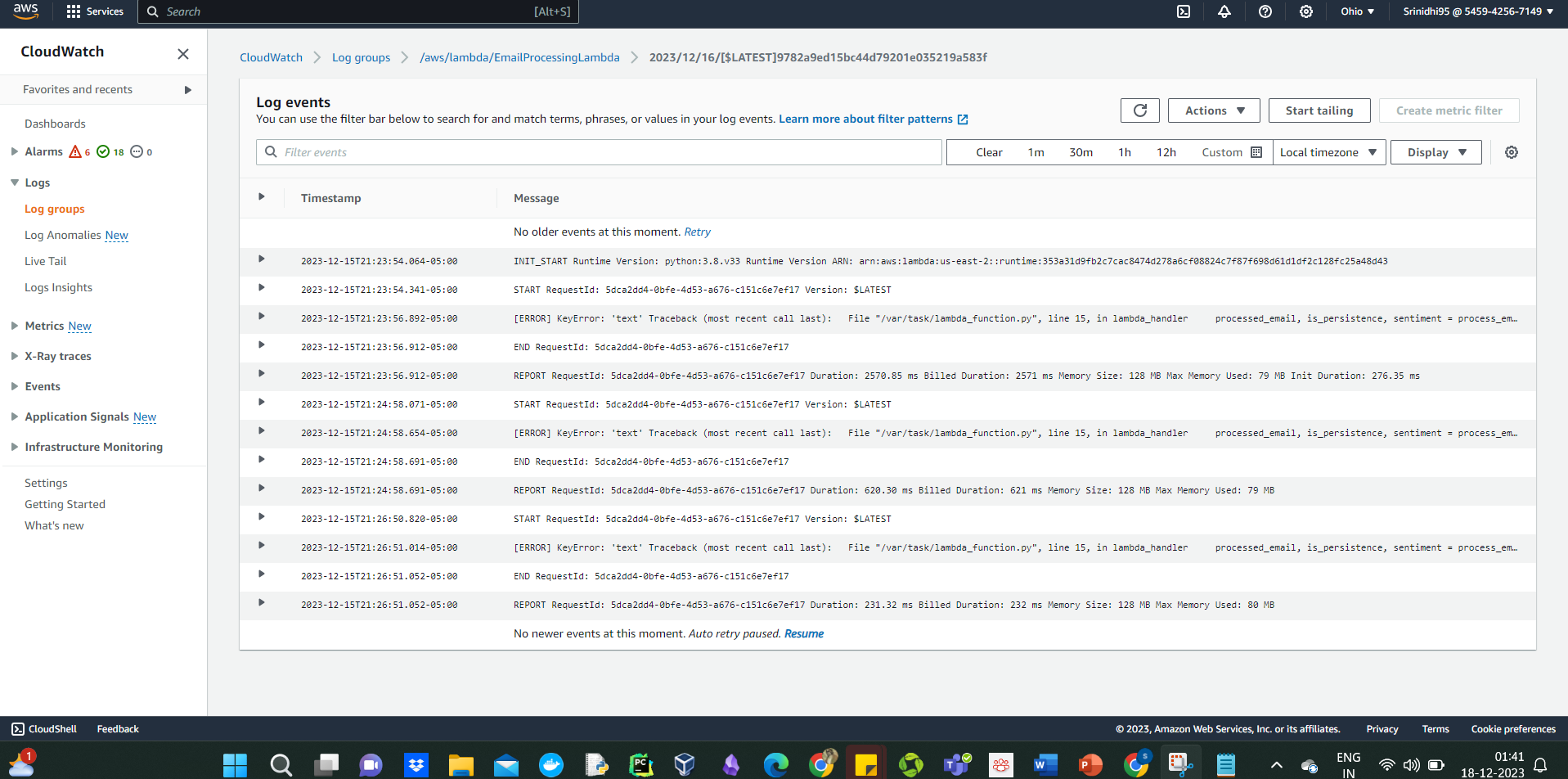
]

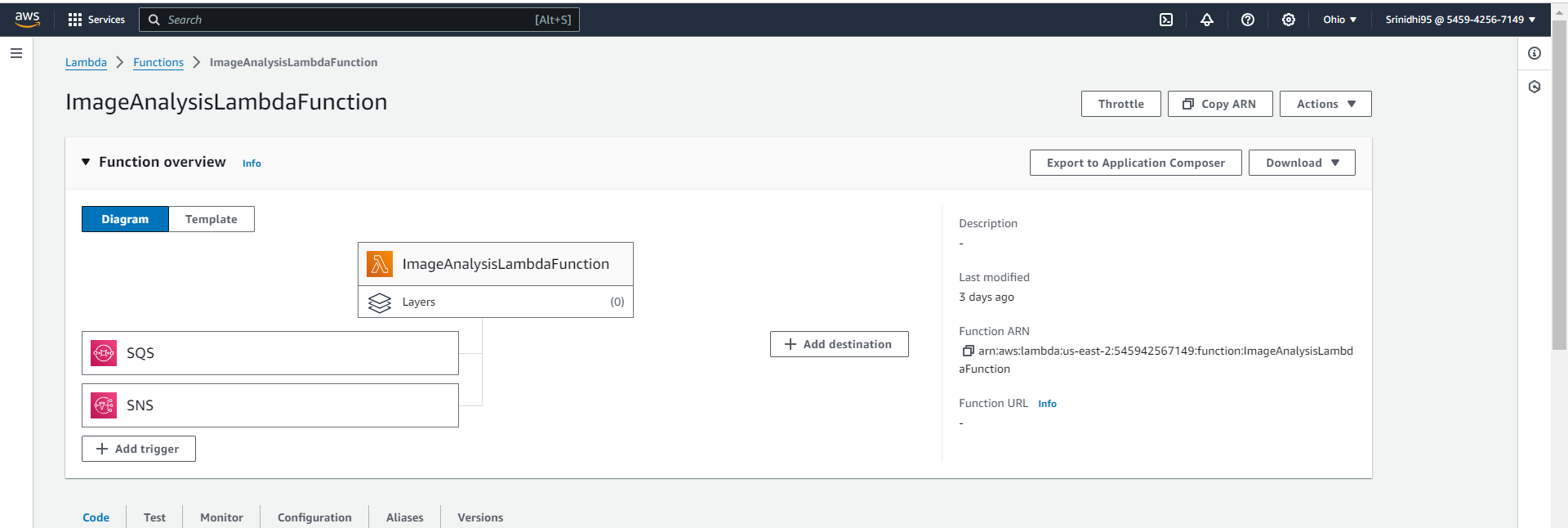
}

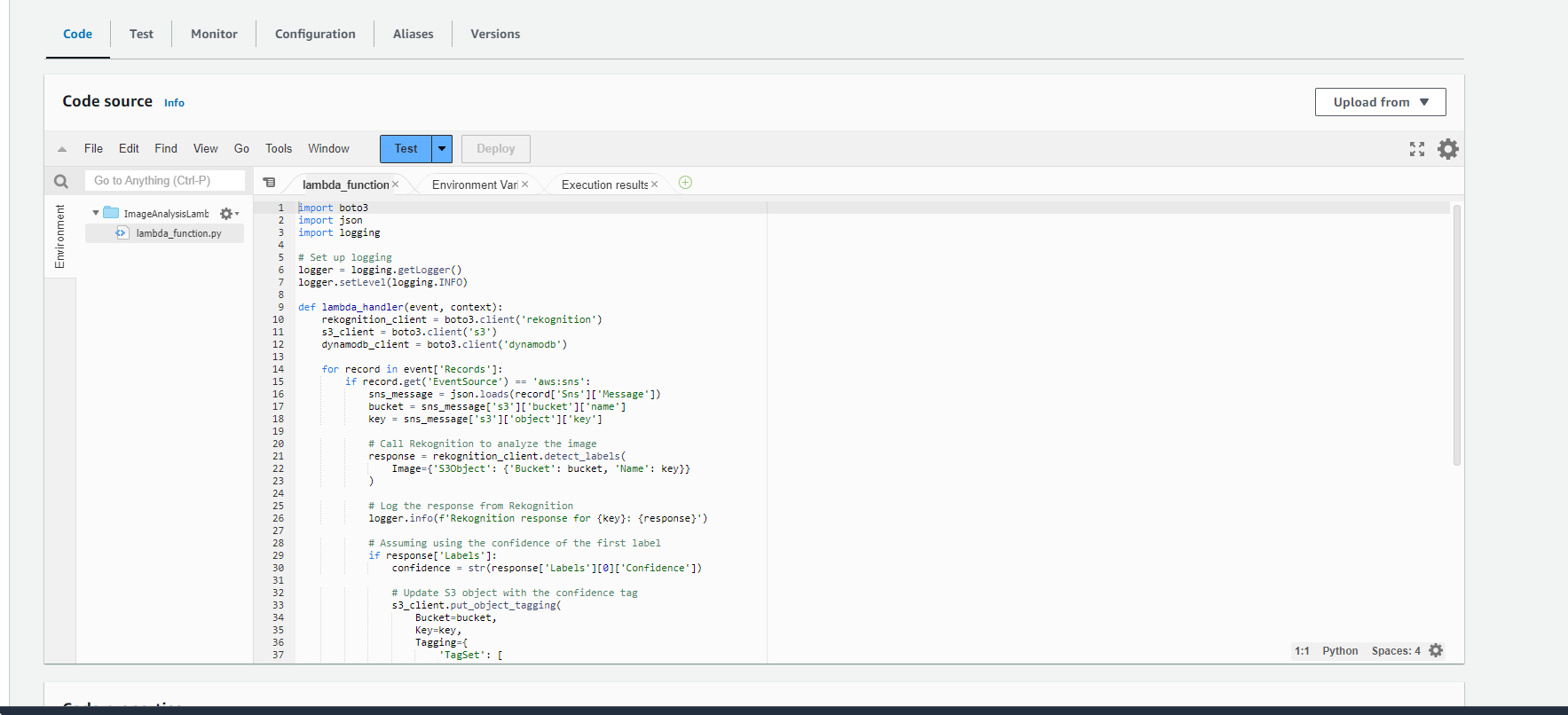












import boto3

import json

import logging

# Set up logging

logger = logging.getLogger()

logger.setLevel(logging.INFO)

def lambda\_handler(event, context):

rekognition\_client = boto3.client('rekognition')

s3\_client = boto3.client('s3')

dynamodb\_client = boto3.client('dynamodb')

for record in event['Records']:

if record.get('EventSource') == 'aws:sns':

sns\_message = json.loads(record['Sns']['Message'])

bucket = sns\_message['s3']['bucket']['name']

key = sns\_message['s3']['object']['key']

# Call Rekognition to analyze the image

response = rekognition\_client.detect\_labels(

Image={'S3Object': {'Bucket': bucket, 'Name': key}}

)

# Log the response from Rekognition

logger.info(f'Rekognition response for {key}: {response}')

# Assuming using the confidence of the first label

if response['Labels']:

confidence = str(response['Labels'][0]['Confidence'])

# Update S3 object with the confidence tag

s3\_client.put\_object\_tagging(

Bucket=bucket,

Key=key,

Tagging={

'TagSet': [

{

'Key': 'Confidence',

'Value': confidence

}

]

}

)

# Prepare the item to put in DynamoDB

item = {

'ImageKey': {'S': key},

'Labels': {'S': json.dumps(response['Labels'])}

}

# Log the item to be put in DynamoDB

logger.info(f'Item to be put in DynamoDB: {item}')

# Store the analysis results in DynamoDB

dynamodb\_client.put\_item(

TableName='ImageAnalysisResults',

Item=item

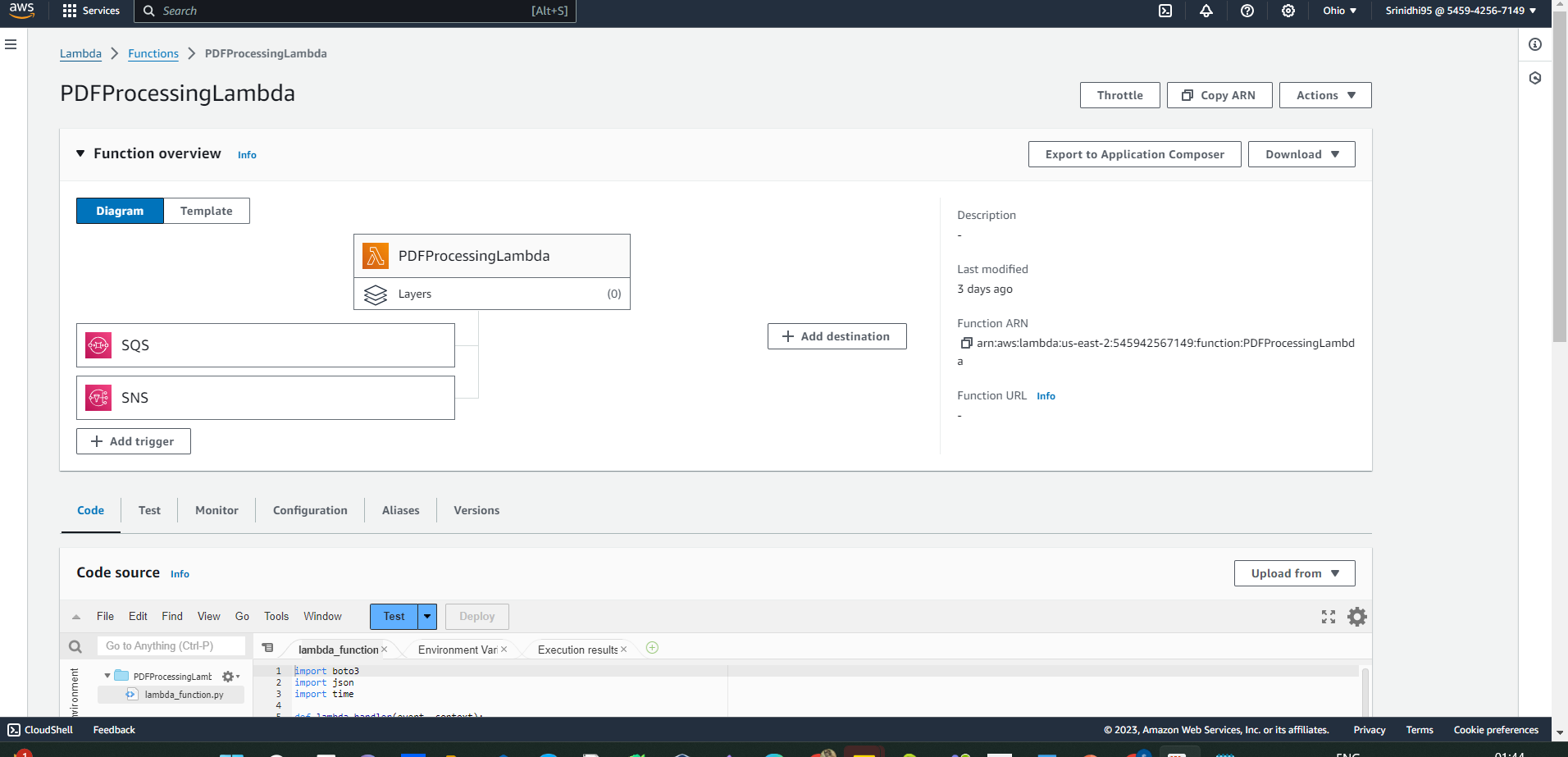
)

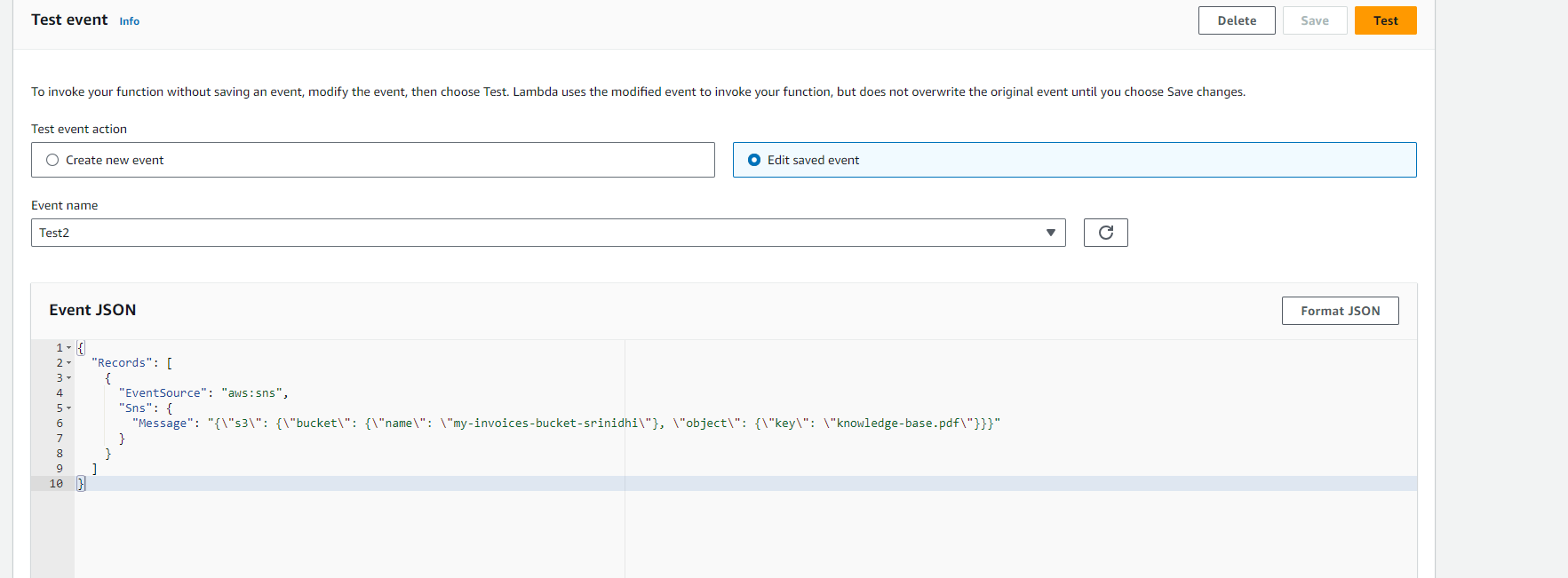
return {

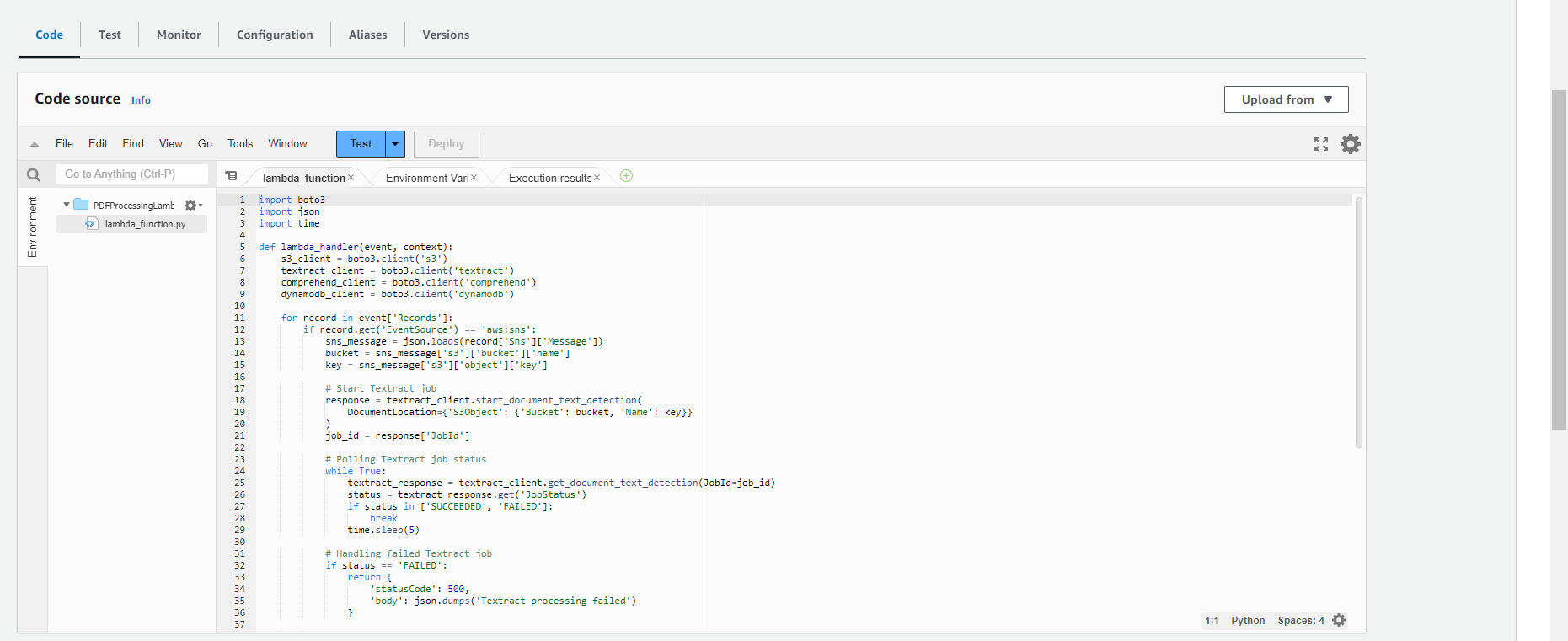
'statusCode': 200,

'body': json.dumps('Image analysis complete')

}







import boto3

import json

import time

def lambda\_handler(event, context):

s3\_client = boto3.client('s3')

textract\_client = boto3.client('textract')

comprehend\_client = boto3.client('comprehend')

dynamodb\_client = boto3.client('dynamodb')

for record in event['Records']:

if record.get('EventSource') == 'aws:sns':

sns\_message = json.loads(record['Sns']['Message'])

bucket = sns\_message['s3']['bucket']['name']

key = sns\_message['s3']['object']['key']

# Start Textract job

response = textract\_client.start\_document\_text\_detection(

DocumentLocation={'S3Object': {'Bucket': bucket, 'Name': key}}

)

job\_id = response['JobId']

# Polling Textract job status

while True:

textract\_response = textract\_client.get\_document\_text\_detection(JobId=job\_id)

status = textract\_response.get('JobStatus')

if status in ['SUCCEEDED', 'FAILED']:

break

time.sleep(5)

# Handling failed Textract job

if status == 'FAILED':

return {

'statusCode': 500,

'body': json.dumps('Textract processing failed')

}

# Extract text from Textract response

extracted\_text = ' '.join([block['Text'] for block in textract\_response['Blocks'] if block['BlockType'] == 'LINE'])

# Perform sentiment analysis

sentiment\_response = comprehend\_client.detect\_sentiment(

Text=extracted\_text[:5000], # Comprehend supports up to 5000 bytes per request

LanguageCode='en'

)

contains\_s3 = 's3' in extracted\_text.lower()

# Storing results in DynamoDB

dynamodb\_client.put\_item(

TableName='ProcessedEmails',

Item={

'PersistenceKeyword': {'S': key}, # Assuming 'key' as the primary key value

'Sentiment': {'S': sentiment\_response['Sentiment']},

'SentimentScore': {'S': json.dumps(sentiment\_response['SentimentScore'])},

'ContainsS3': {'BOOL': contains\_s3}

}

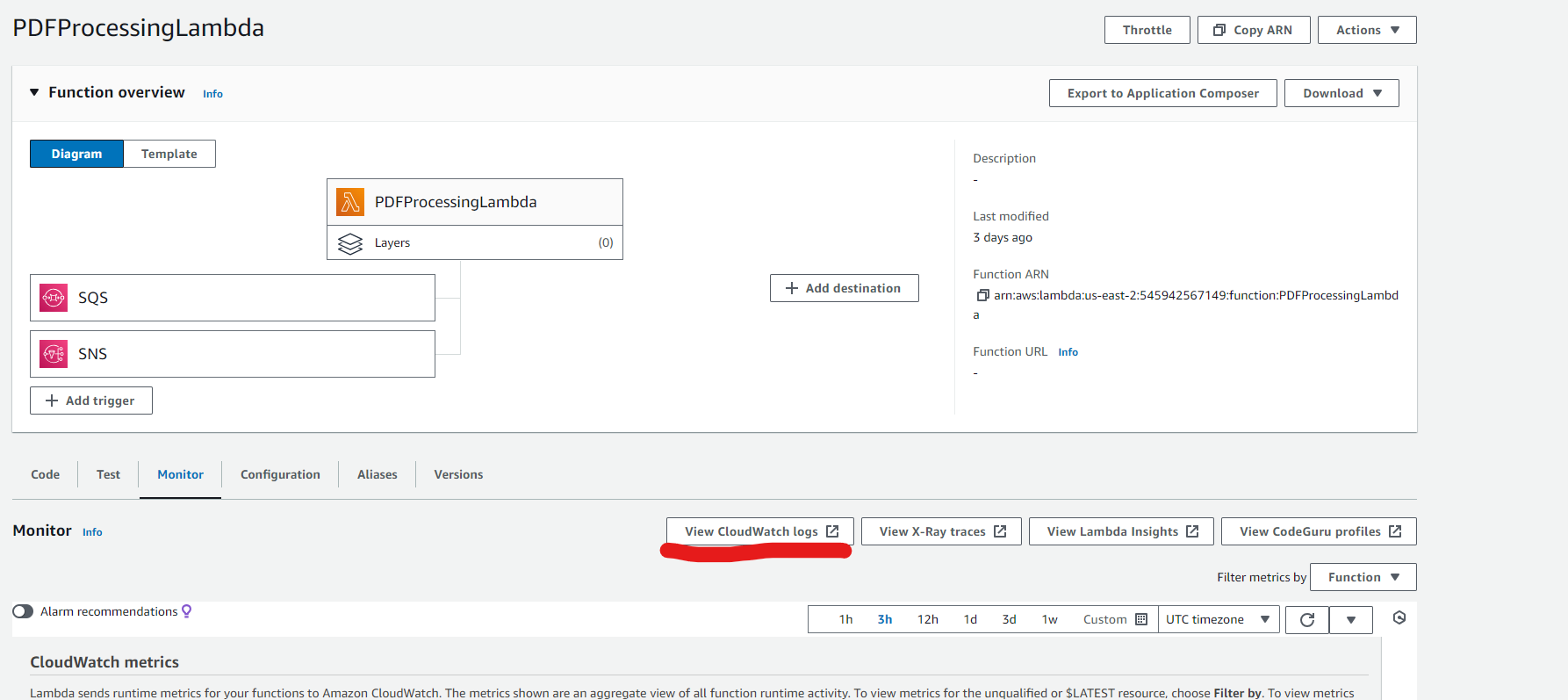
)

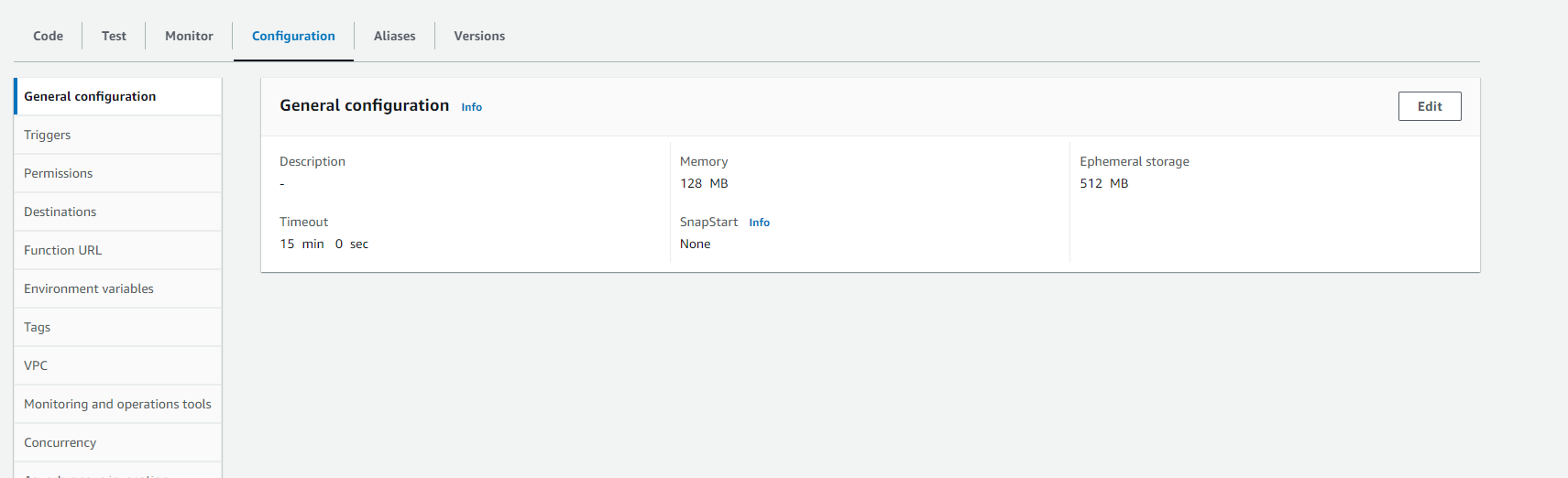
return {

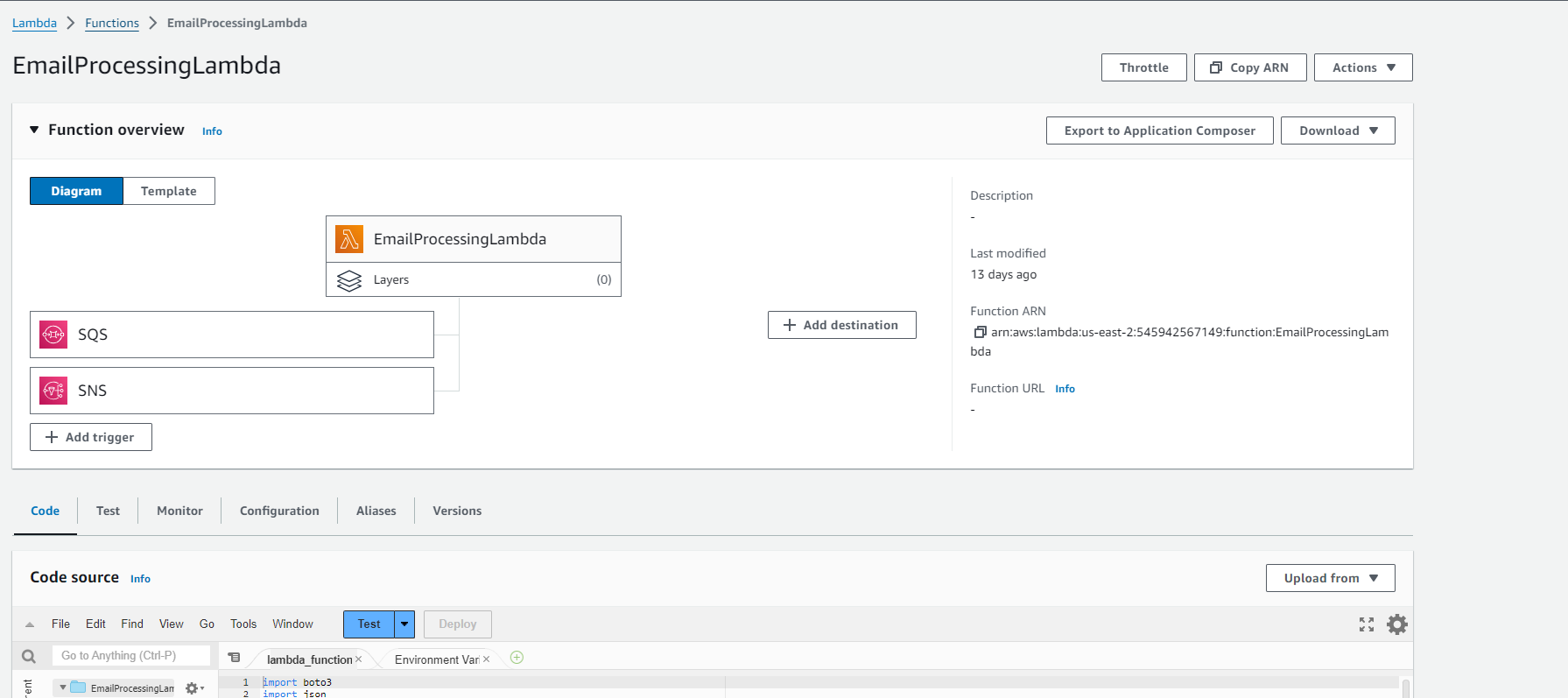
'statusCode': 200,

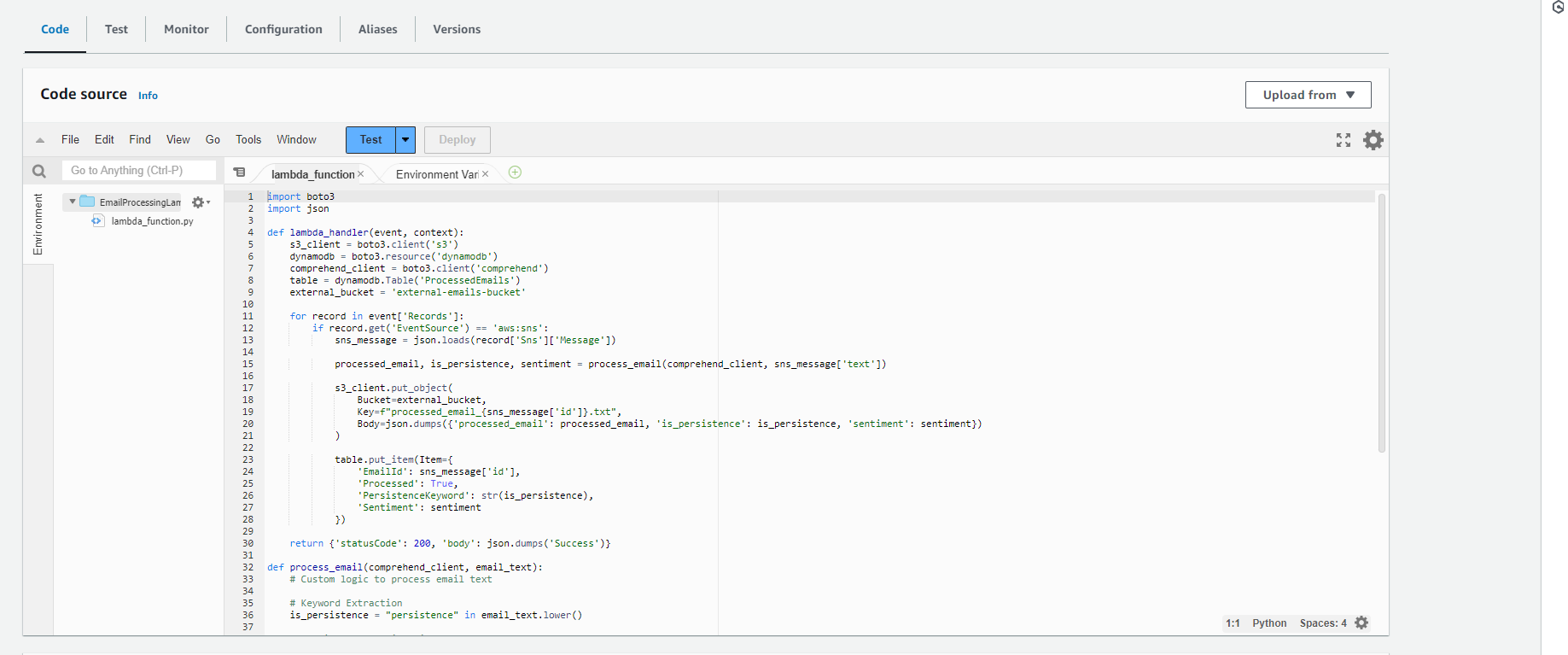
'body': json.dumps('PDF processing complete')

}









import boto3

import json

def lambda\_handler(event, context):

s3\_client = boto3.client('s3')

dynamodb = boto3.resource('dynamodb')

comprehend\_client = boto3.client('comprehend')

table = dynamodb.Table('ProcessedEmails')

external\_bucket = 'external-emails-bucket'

for record in event['Records']:

if record.get('EventSource') == 'aws:sns':

sns\_message = json.loads(record['Sns']['Message'])

processed\_email, is\_persistence, sentiment = process\_email(comprehend\_client, sns\_message['text'])

s3\_client.put\_object(

Bucket=external\_bucket,

Key=f"processed\_email\_{sns\_message['id']}.txt",

Body=json.dumps({'processed\_email': processed\_email, 'is\_persistence': is\_persistence, 'sentiment': sentiment})

)

table.put\_item(Item={

'EmailId': sns\_message['id'],

'Processed': True,

'PersistenceKeyword': str(is\_persistence),

'Sentiment': sentiment

})

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

def process\_email(comprehend\_client, email\_text):

# Custom logic to process email text

# Keyword Extraction

is\_persistence = "persistence" in email\_text.lower()

# Sentiment Analysis using AWS Comprehend

sentiment = analyze\_sentiment(comprehend\_client, email\_text)

return email\_text, is\_persistence, sentiment

def analyze\_sentiment(comprehend\_client, text):

response = comprehend\_client.detect\_sentiment(Text=text, LanguageCode='en')

sentiment = response['Sentiment']

return sentiment

# Example usage for testing with SNS-like event

if \_\_name\_\_ == "\_\_main\_\_":

example\_email = "This project requires persistence and hard work. We're making great progress!"

test\_event = {

'Records': [

{

'EventSource': 'aws:sns',

'Sns': {

'Message': json.dumps({'text': example\_email, 'id': '123'})

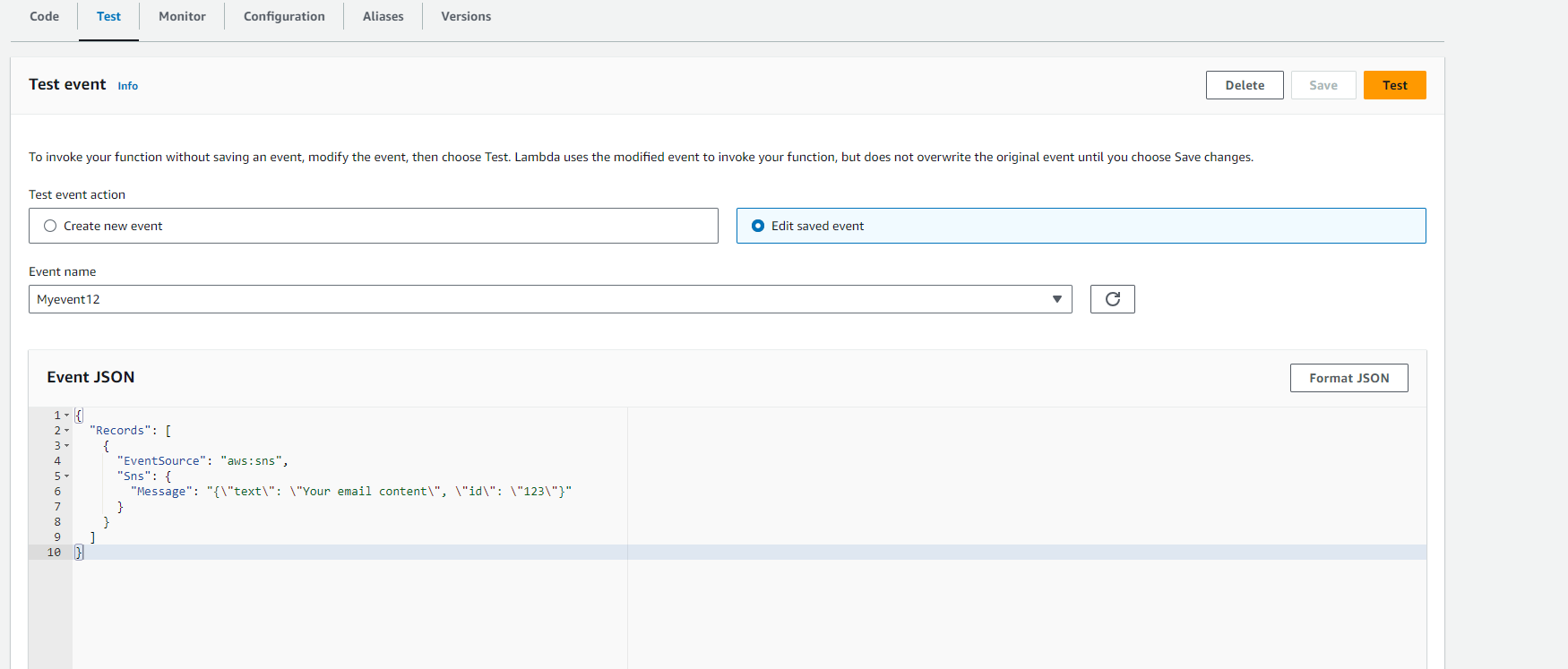
}

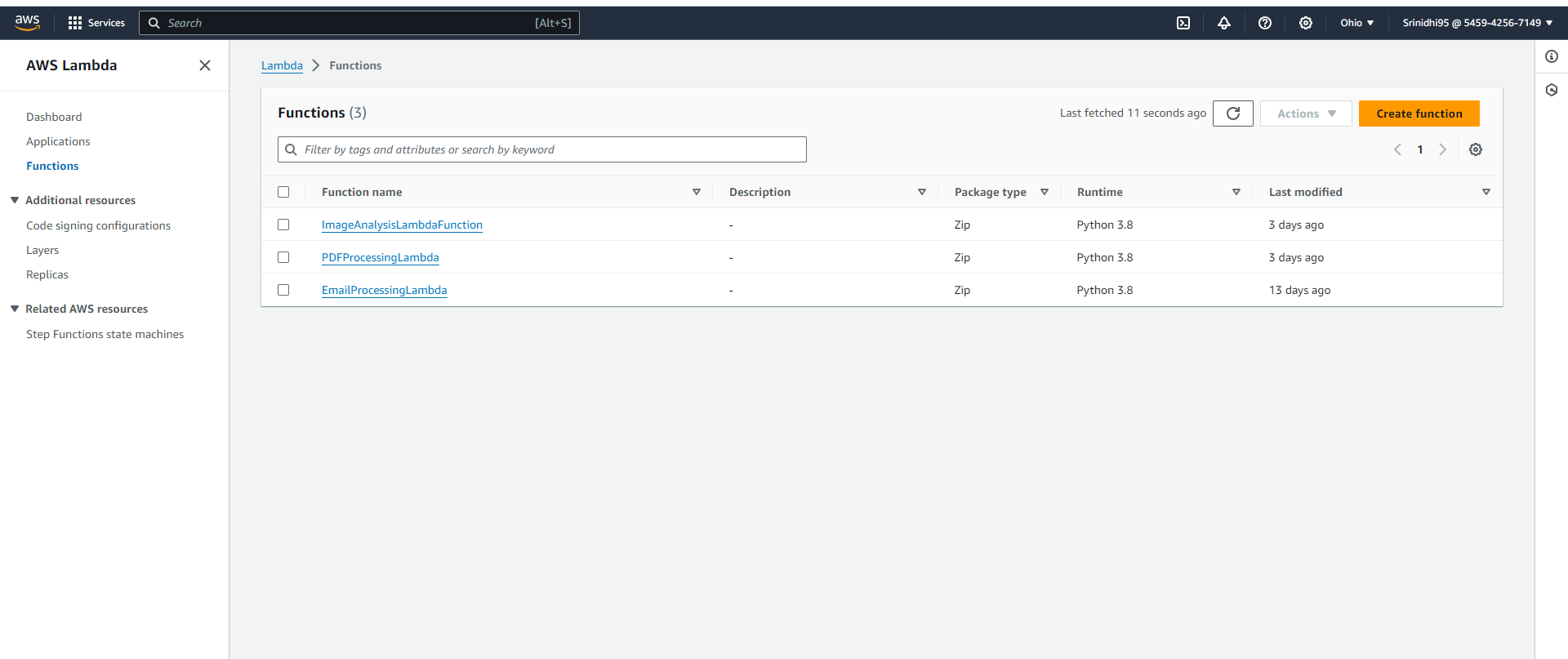
}

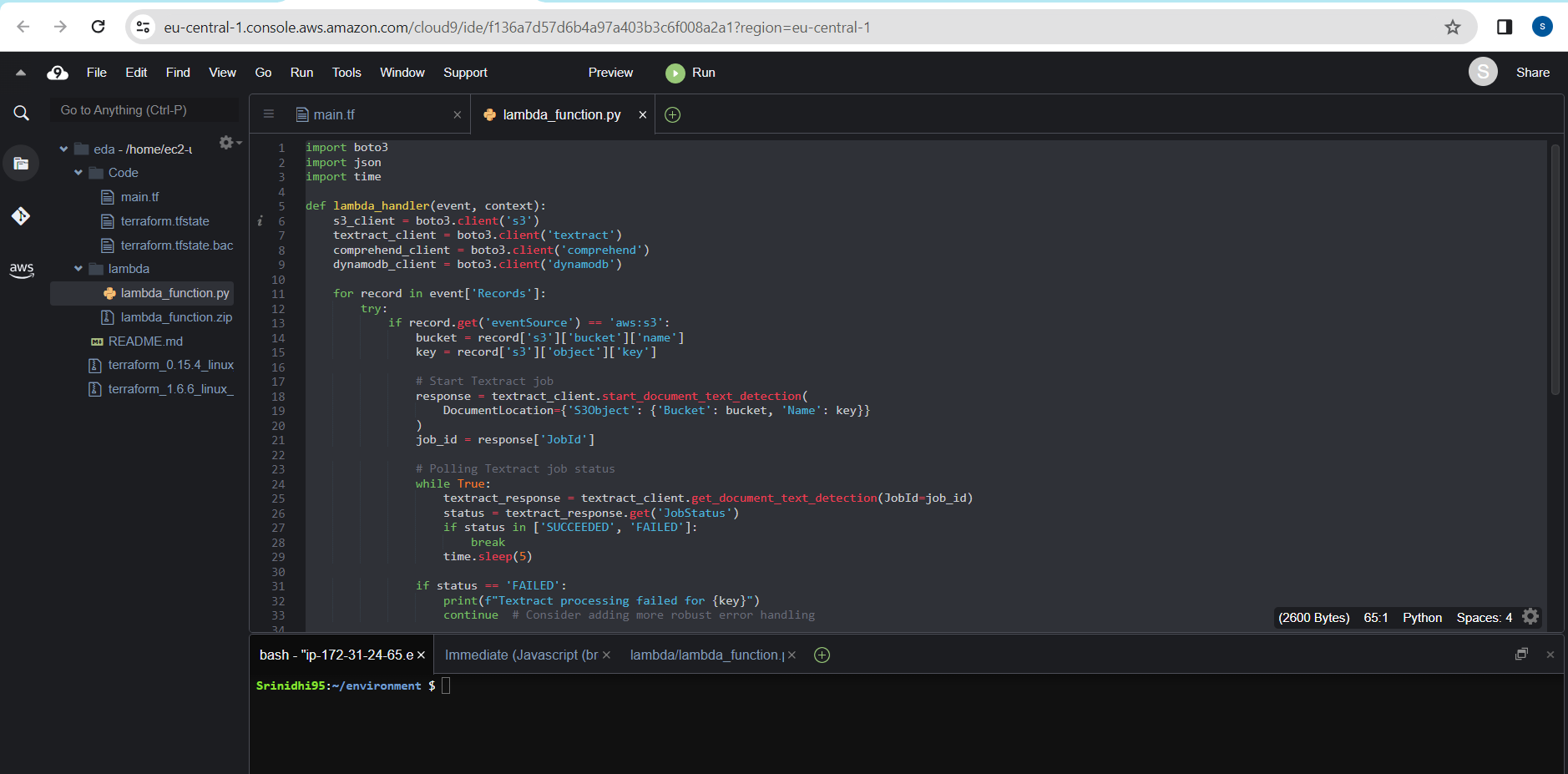
]

}

print(lambda\_handler(test\_event, None))







import boto3

import json

import time

def lambda\_handler(event, context):

s3\_client = boto3.client('s3')

textract\_client = boto3.client('textract')

comprehend\_client = boto3.client('comprehend')

dynamodb\_client = boto3.client('dynamodb')

for record in event['Records']:

try:

if record.get('eventSource') == 'aws:s3':

bucket = record['s3']['bucket']['name']

key = record['s3']['object']['key']

# Start Textract job

response = textract\_client.start\_document\_text\_detection(

DocumentLocation={'S3Object': {'Bucket': bucket, 'Name': key}}

)

job\_id = response['JobId']

# Polling Textract job status

while True:

textract\_response = textract\_client.get\_document\_text\_detection(JobId=job\_id)

status = textract\_response.get('JobStatus')

if status in ['SUCCEEDED', 'FAILED']:

break

time.sleep(5)

if status == 'FAILED':

print(f"Textract processing failed for {key}")

continue # Consider adding more robust error handling

# Extract text from Textract response

extracted\_text = ' '.join([block['Text'] for block in textract\_response['Blocks'] if block['BlockType'] == 'LINE'])

# Perform sentiment analysis

sentiment\_response = comprehend\_client.detect\_sentiment(

Text=extracted\_text[:5000], # Comprehend supports up to 5000 bytes per request

LanguageCode='en'

)

contains\_s3 = 's3' in extracted\_text.lower()

# Storing results in DynamoDB

dynamodb\_client.put\_item(

TableName='pdf-metadata', # Ensure this matches your DynamoDB table name

Item={

'id': {'S': key}, # Adjust the primary key field as per your table's schema

'Sentiment': {'S': sentiment\_response['Sentiment']},

'SentimentScore': {'S': json.dumps(sentiment\_response['SentimentScore'])},

'ContainsS3': {'BOOL': contains\_s3}

}

)

except Exception as e:

# Log the error

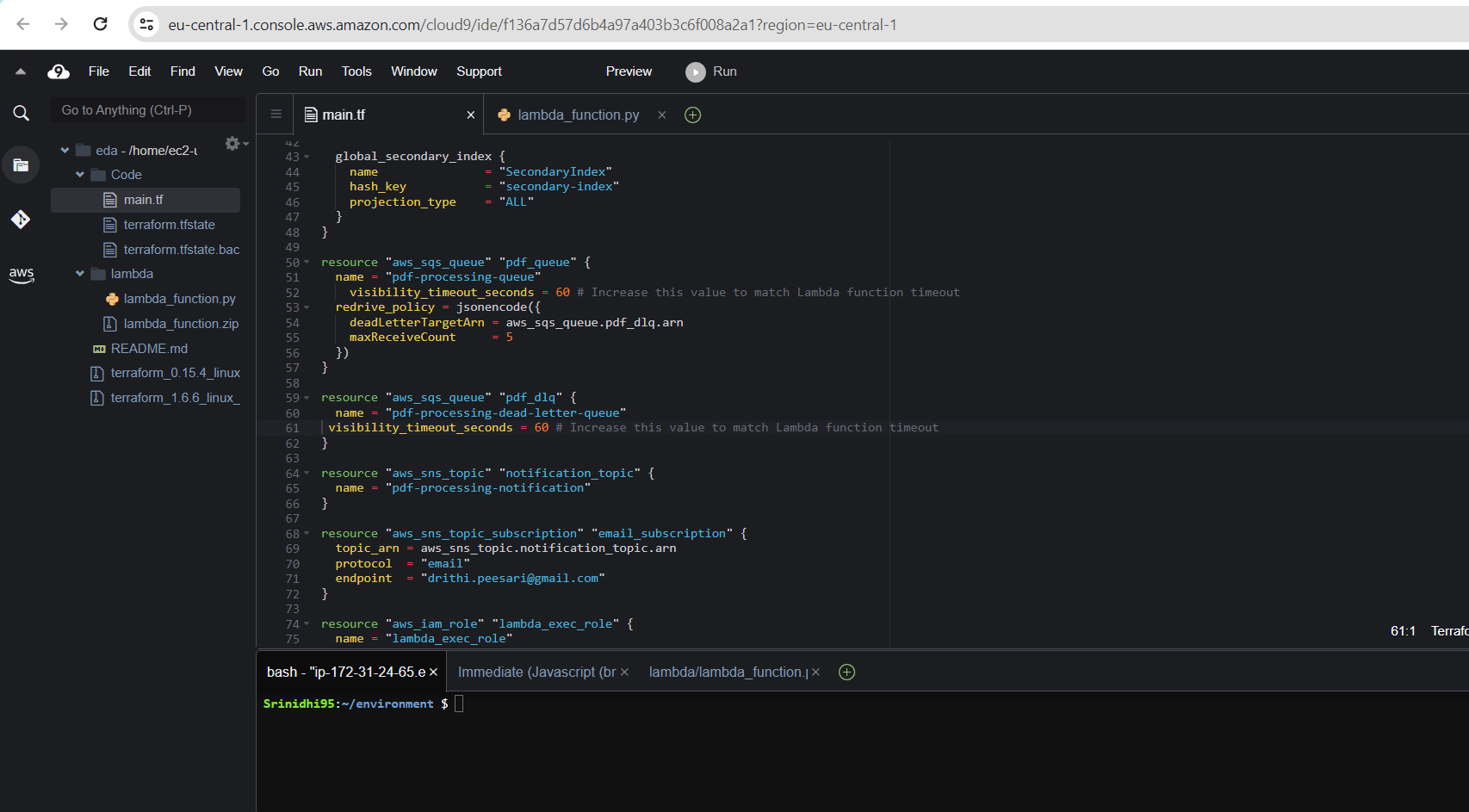
print(f"Error processing record {record}: {str(e)}")

return {

'statusCode': 200,

'body': json.dumps('PDF processing complete')

}



provider "aws" {

region = "eu-central-1" # Set the desired region for all resources

}

resource "aws\_s3\_bucket" "pdf\_bucket" {

bucket = "pdf-processing-bucket-srinidhi"

}

resource "aws\_s3\_bucket\_versioning" "pdf\_bucket\_versioning" {

bucket = aws\_s3\_bucket.pdf\_bucket.id

versioning\_configuration {

status = "Enabled"

}

}

resource "aws\_s3\_bucket\_server\_side\_encryption\_configuration" "pdf\_bucket\_sse" {

bucket = aws\_s3\_bucket.pdf\_bucket.id

rule {

apply\_server\_side\_encryption\_by\_default {

sse\_algorithm = "AES256"

}

}

}

resource "aws\_dynamodb\_table" "metadata\_table" {

name = "pdf-metadata"

hash\_key = "id"

billing\_mode = "PAY\_PER\_REQUEST"

attribute {

name = "id"

type = "S"

}

attribute {

name = "secondary-index"

type = "S"

}

global\_secondary\_index {

name = "SecondaryIndex"

hash\_key = "secondary-index"

projection\_type = "ALL"

}

}

resource "aws\_sqs\_queue" "pdf\_queue" {

name = "pdf-processing-queue"

visibility\_timeout\_seconds = 60 # Increase this value to match Lambda function timeout

redrive\_policy = jsonencode({

deadLetterTargetArn = aws\_sqs\_queue.pdf\_dlq.arn

maxReceiveCount = 5

})

}

resource "aws\_sqs\_queue" "pdf\_dlq" {

name = "pdf-processing-dead-letter-queue"

visibility\_timeout\_seconds = 60 # Increase this value to match Lambda function timeout

}

resource "aws\_sns\_topic" "notification\_topic" {

name = "pdf-processing-notification"

}

resource "aws\_sns\_topic\_subscription" "email\_subscription" {

topic\_arn = aws\_sns\_topic.notification\_topic.arn

protocol = "email"

endpoint = "drithi.peesari@gmail.com"

}

resource "aws\_iam\_role" "lambda\_exec\_role" {

name = "lambda\_exec\_role"

assume\_role\_policy = jsonencode({

Version = "2012-10-17",

Statement = [

{

Action = "sts:AssumeRole",

Effect = "Allow",

Principal = {

Service = "lambda.amazonaws.com"

}

}

]

})

}

resource "aws\_iam\_policy" "lambda\_policy" {

name = "lambda\_pdf\_processing\_policy\_9908"

path = "/"

description = "IAM policy for PDF processing Lambda function"

policy = jsonencode({

Version = "2012-10-17",

Statement = [

{

Effect = "Allow",

Action = [

"s3:\*",

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents",

"sqs:ReceiveMessage",

"sqs:DeleteMessage",

"sqs:GetQueueAttributes",

"rekognition:DetectLabels",

"dynamodb:PutItem",

"comprehend:DetectSentiment",

"textract:StartDocumentTextDetection",

"textract:GetDocumentTextDetection"

],

Resource = "\*"

}

]

})

}

resource "aws\_iam\_role\_policy\_attachment" "lambda\_policy\_attach" {

policy\_arn = aws\_iam\_policy.lambda\_policy.arn

role = aws\_iam\_role.lambda\_exec\_role.name

}

resource "aws\_lambda\_function" "pdf\_processor" {

function\_name = "pdfProcessorFunction"

runtime = "python3.8"

handler = "lambda\_function.lambda\_handler"

s3\_bucket = aws\_s3\_bucket.pdf\_bucket.bucket

s3\_key = "lambda\_function.zip"

role = aws\_iam\_role.lambda\_exec\_role.arn

environment {

variables = {

S3\_BUCKET = aws\_s3\_bucket.pdf\_bucket.bucket

DYNAMODB\_TABLE = aws\_dynamodb\_table.metadata\_table.name

SQS\_QUEUE = aws\_sqs\_queue.pdf\_queue.name

SNS\_TOPIC = aws\_sns\_topic.notification\_topic.arn

}

}

timeout = 60

memory\_size = 128

}

resource "aws\_lambda\_permission" "allow\_bucket" {

statement\_id = "AllowExecutionFromS3Bucket"

action = "lambda:InvokeFunction"

function\_name = aws\_lambda\_function.pdf\_processor.arn

principal = "s3.amazonaws.com"

source\_arn = aws\_s3\_bucket.pdf\_bucket.arn

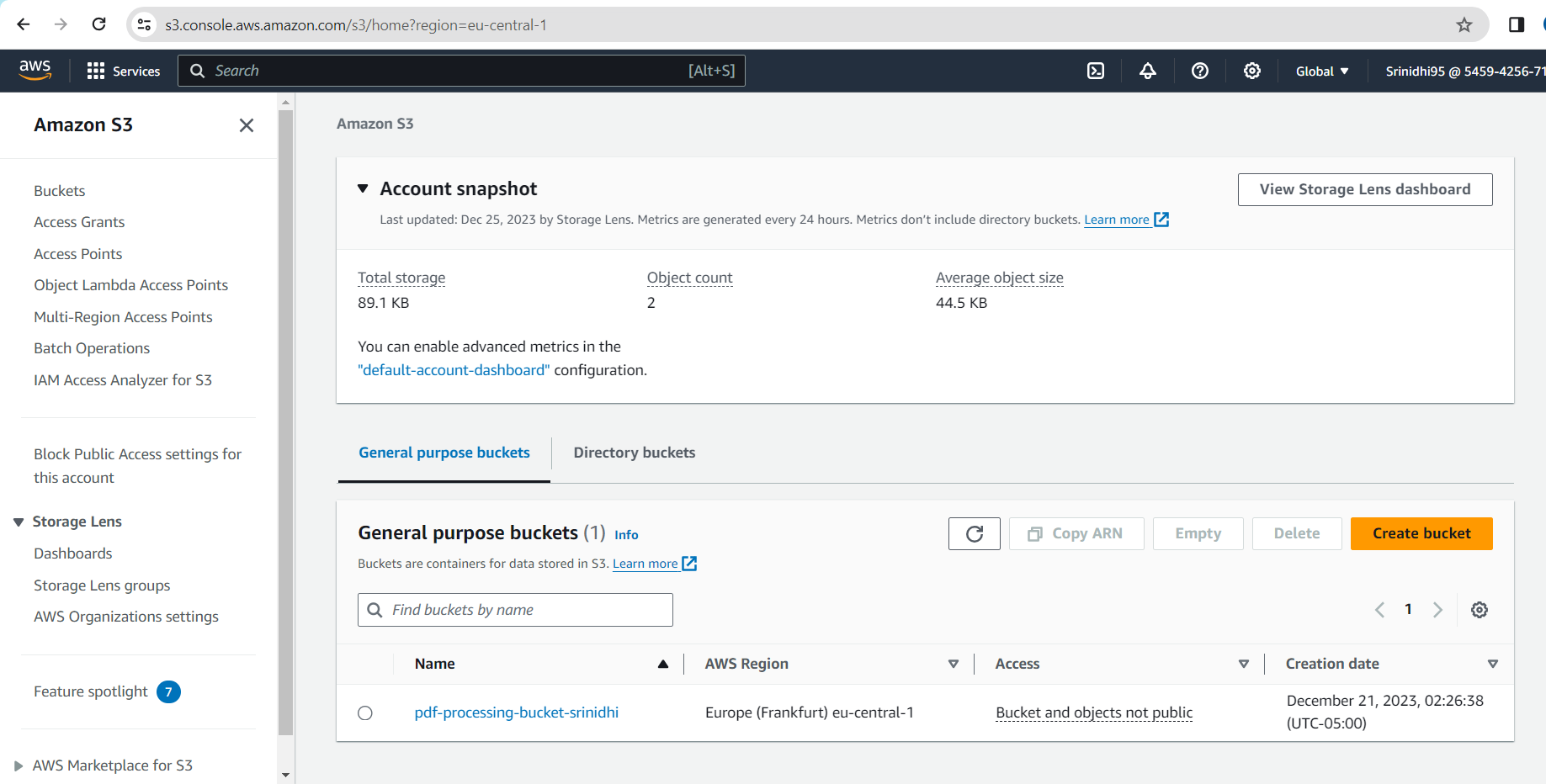
}

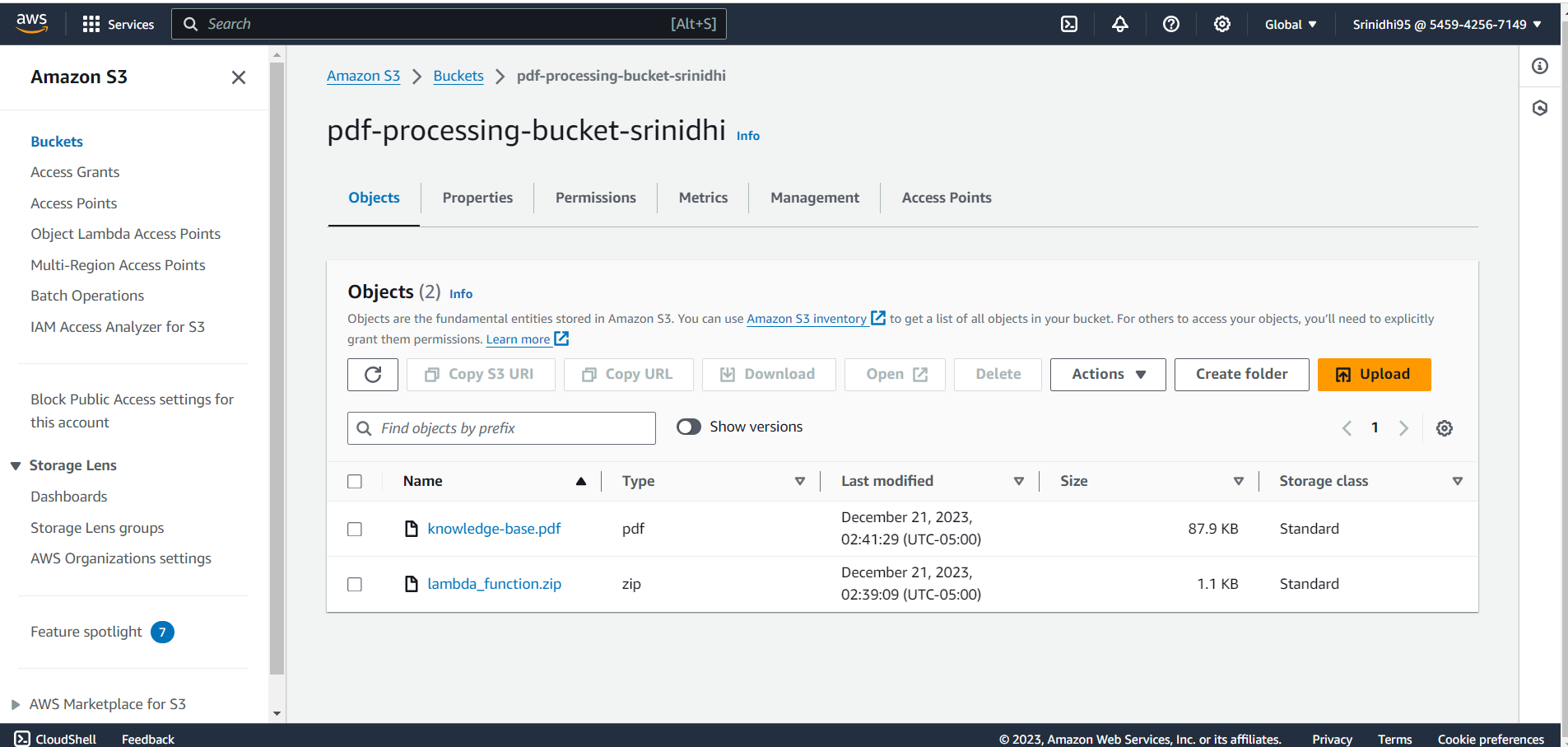
resource "aws\_cloudwatch\_log\_group" "lambda\_log\_group" {

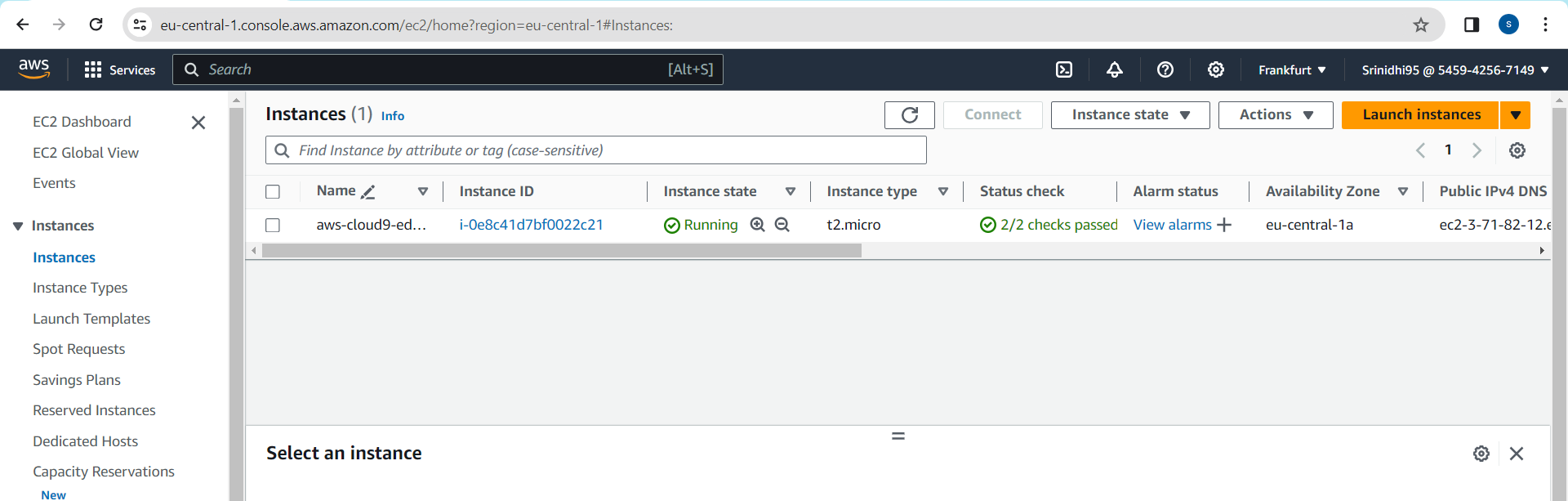
name = "/aws/lambda/${aws\_lambda\_function.pdf\_processor.function\_name}"

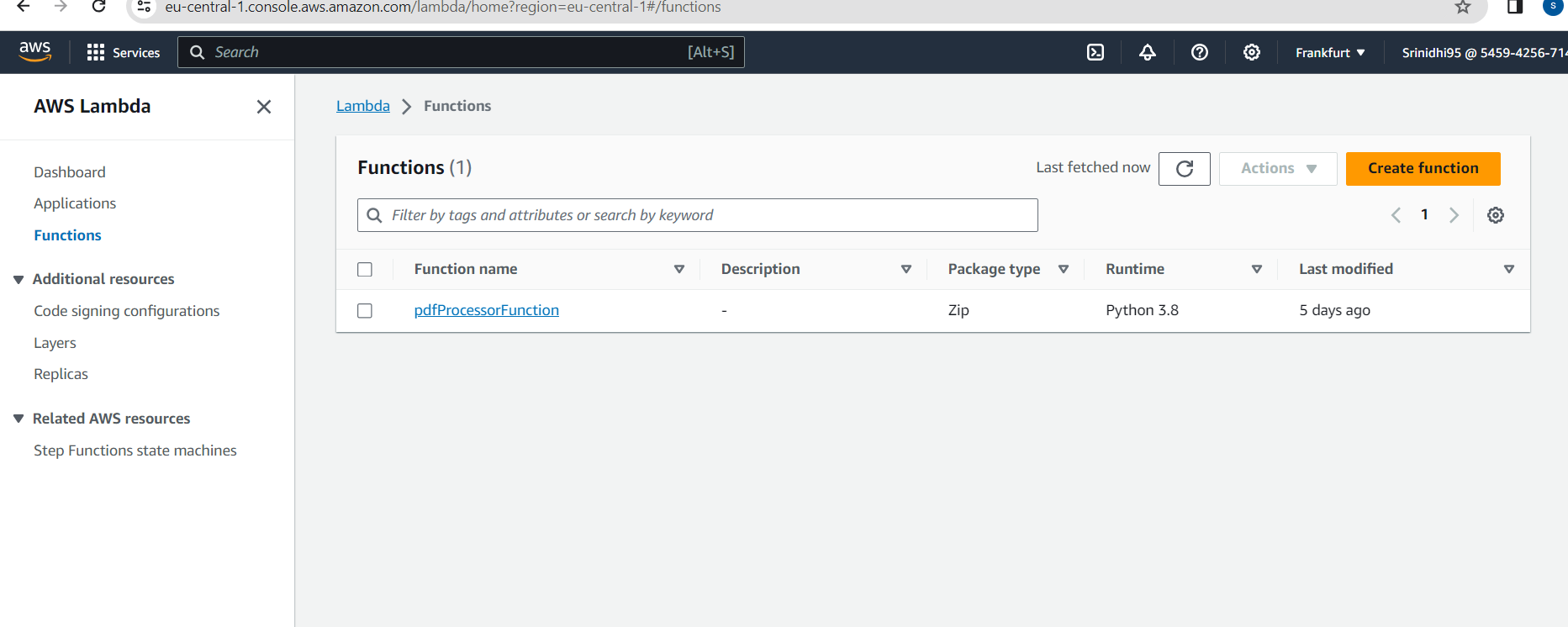
retention\_in\_days = 14

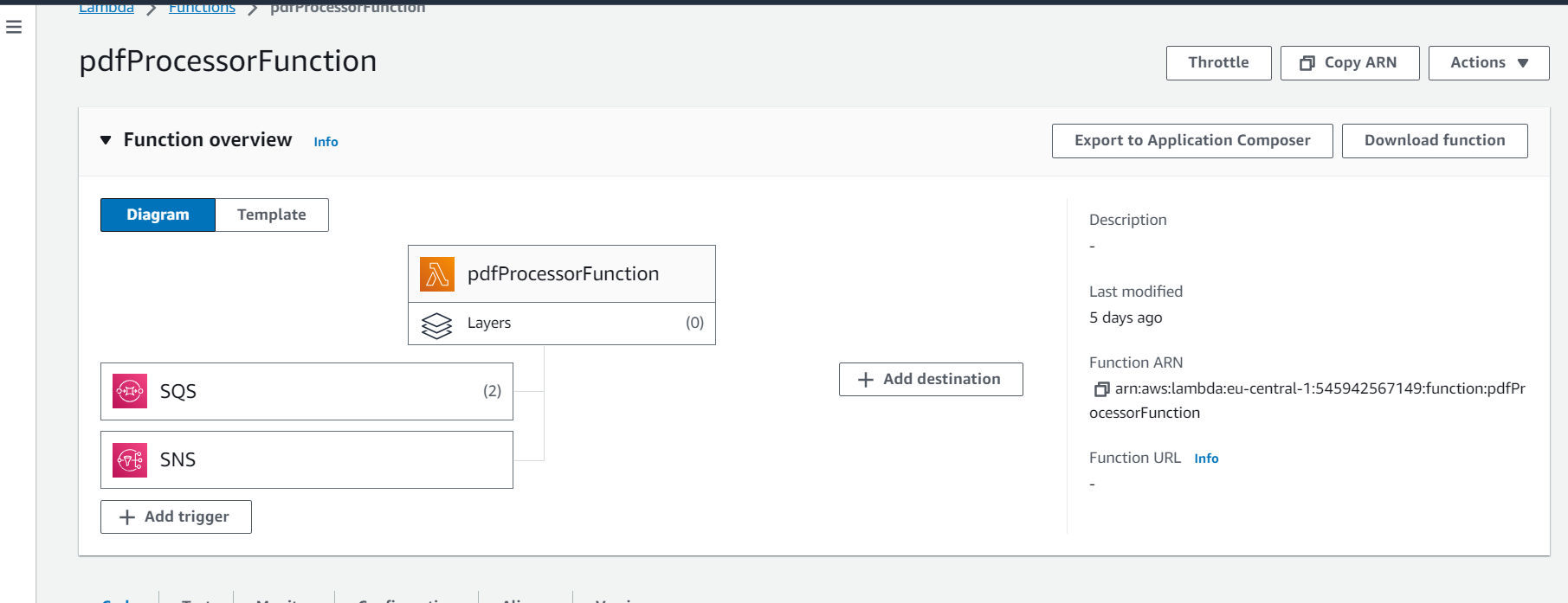
}

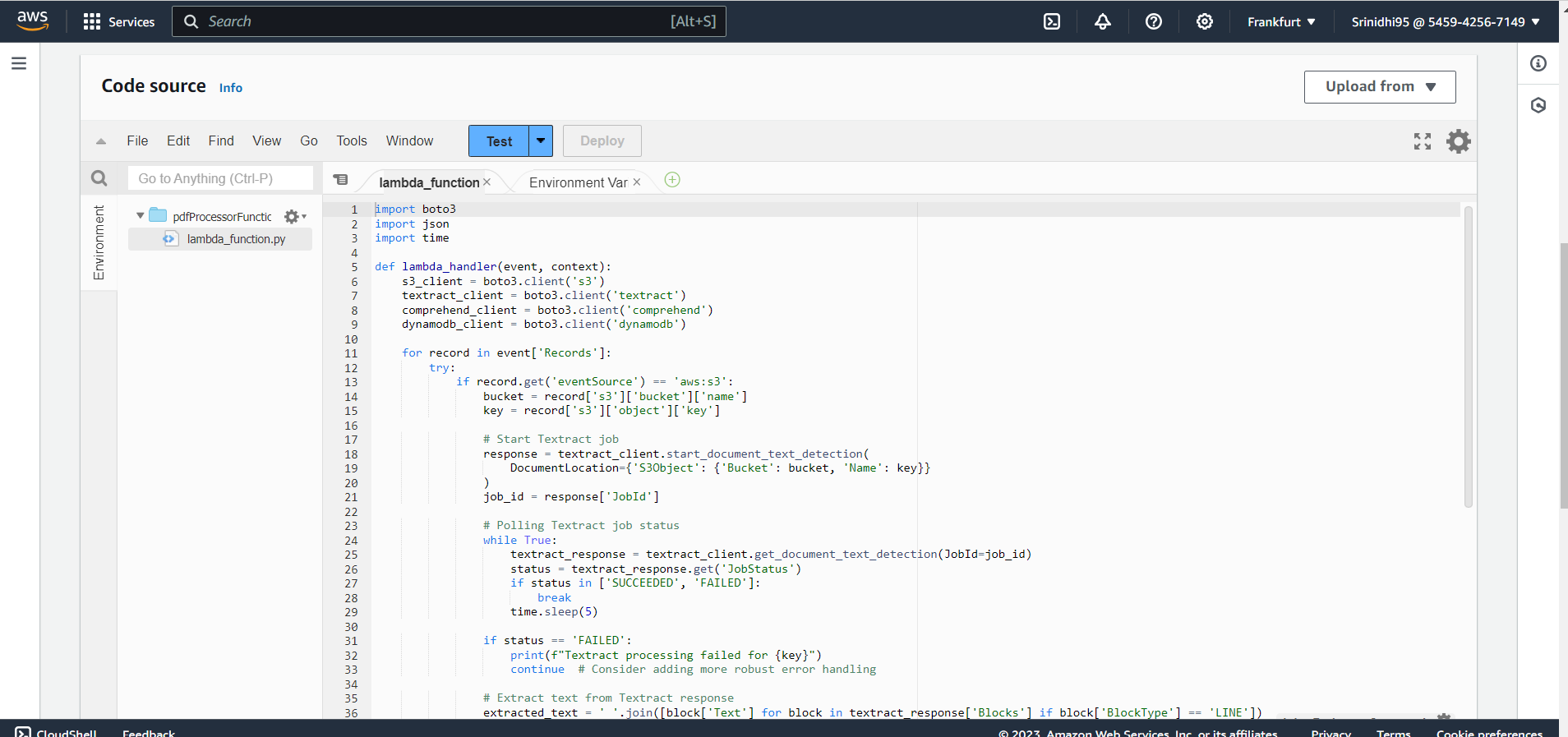


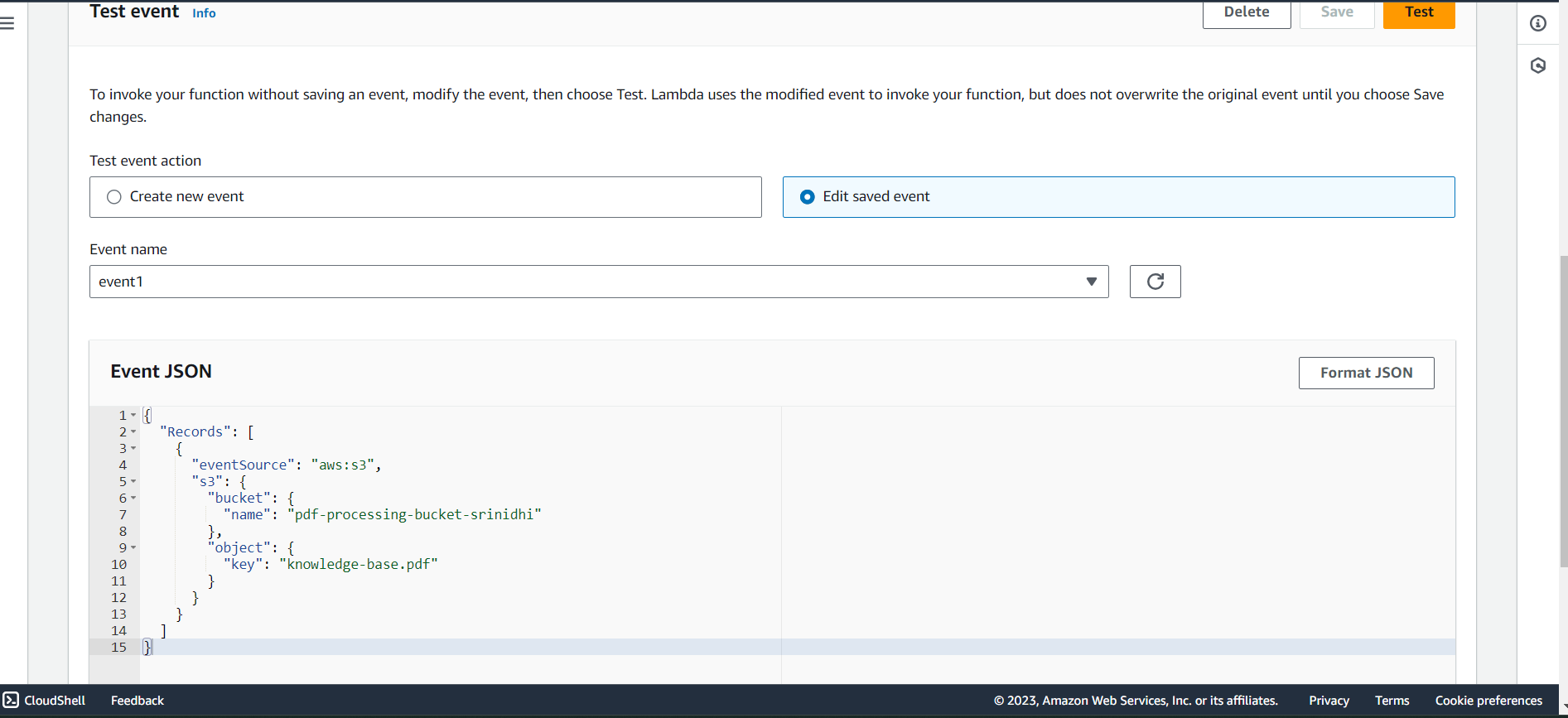












{

"Records": [

{

"eventSource": "aws:s3",

"s3": {

"bucket": {

"name": "pdf-processing-bucket-srinidhi"

},

"object": {

"key": "knowledge-base.pdf"

}

}

}

]

}

