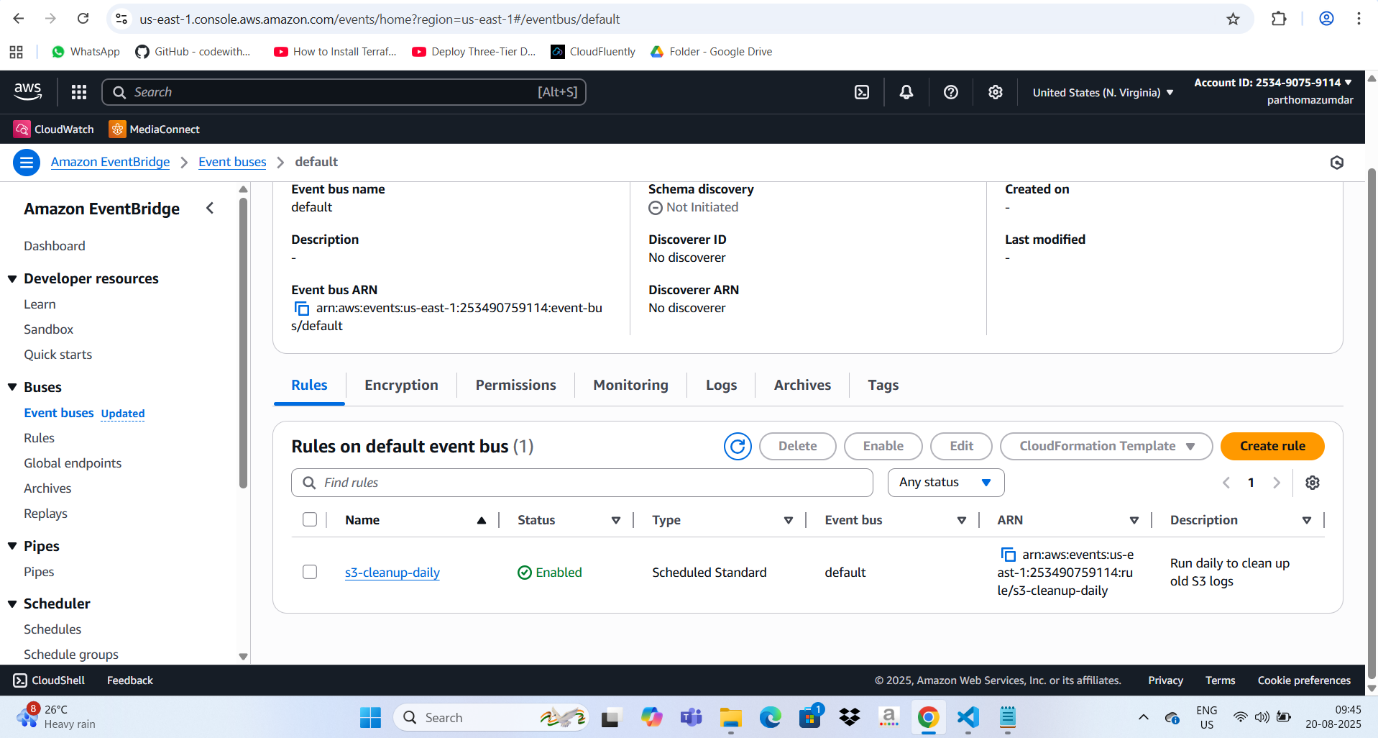
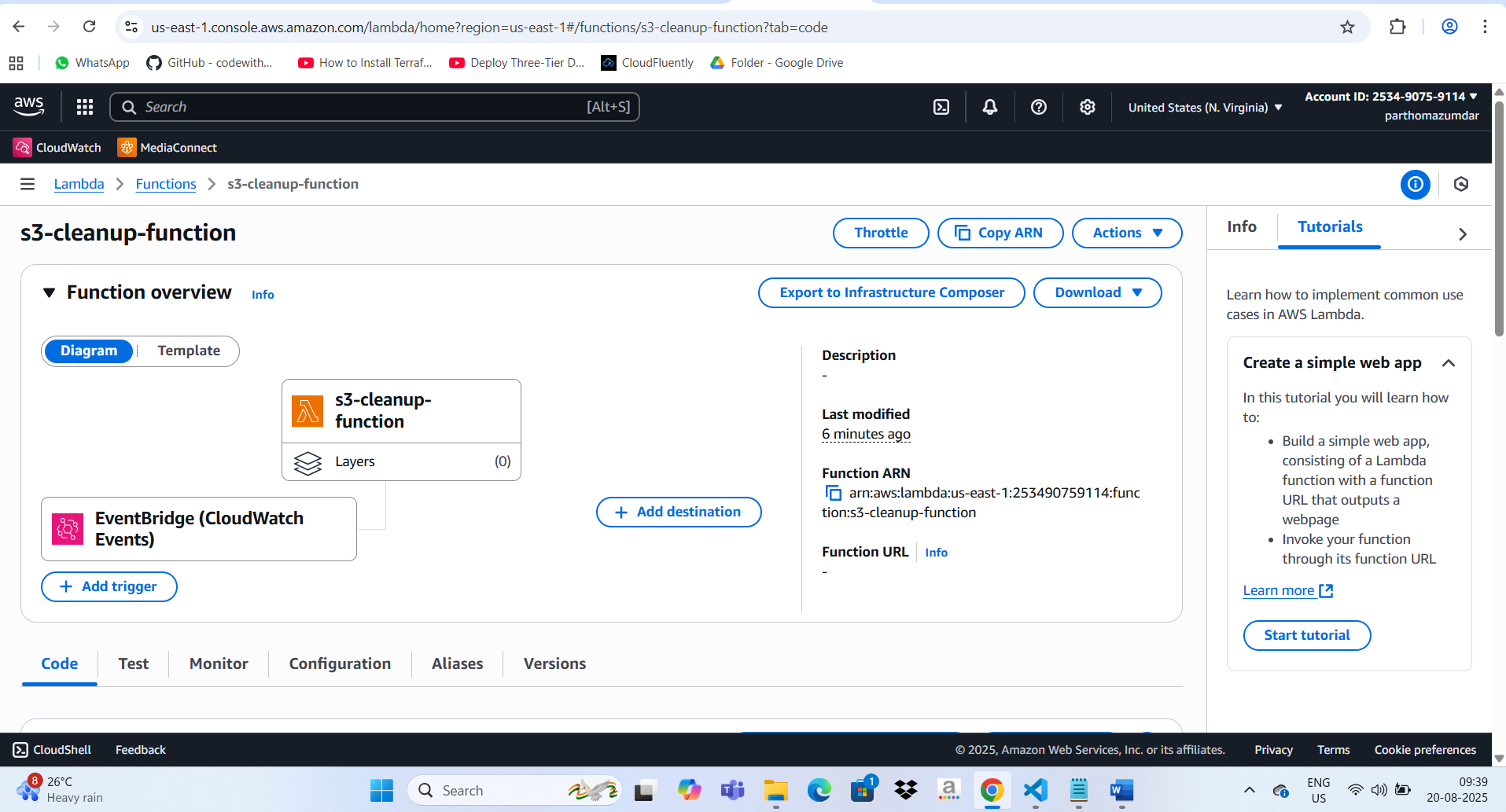
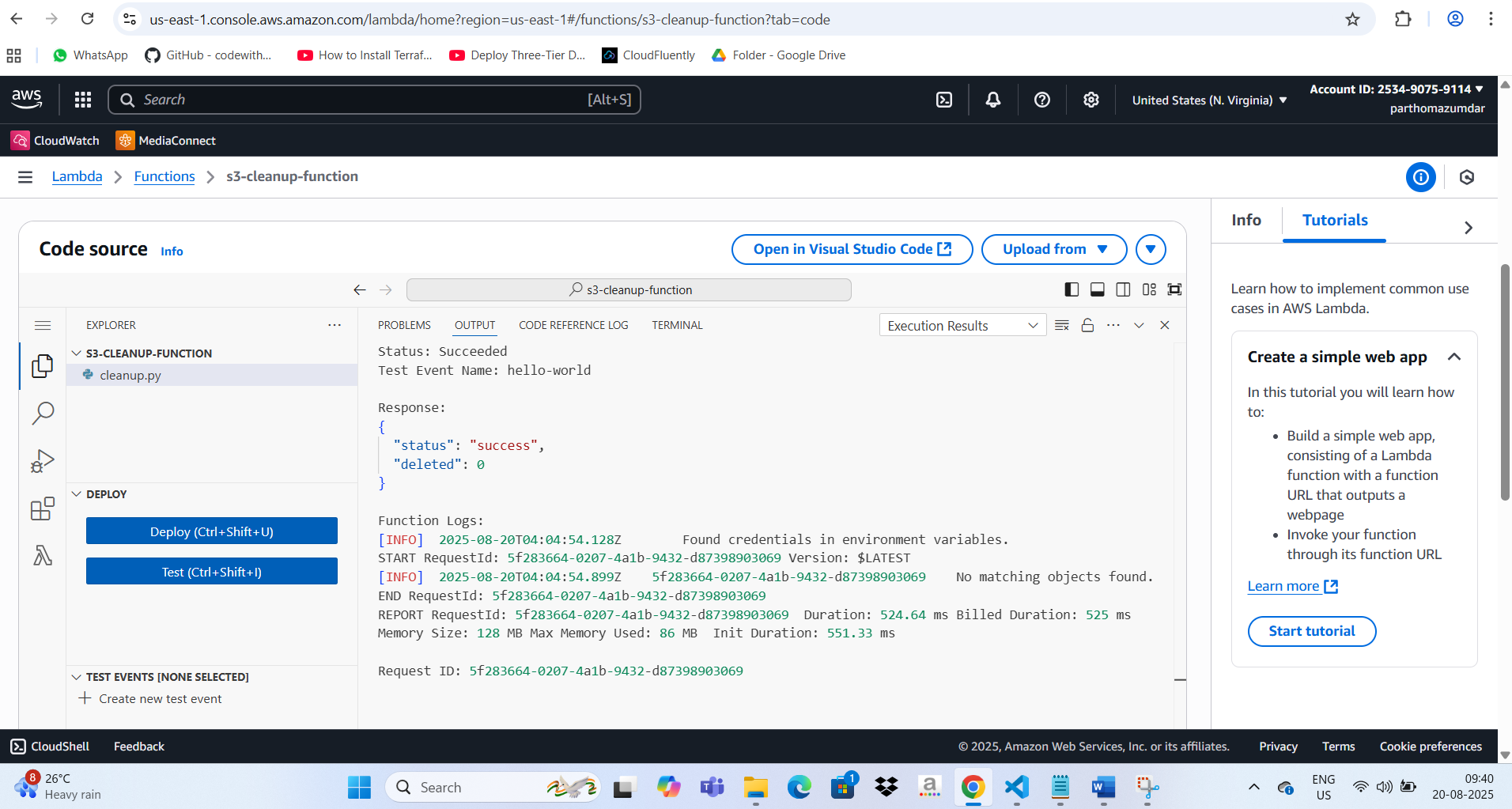
  
  
  
AWS S3 bucket is in us-east 2  
  
  
  
  
  
Amazon Eventbridge is also in us-east 2  
  
  
  
  
  


AWS Lambda invoked manually for testing yielding a successful response.

Perfect 🚀 Let’s build a **Terraform template** that provisions:

* An **S3 bucket** (for logs)
* An **IAM role + policy** (for Lambda permissions)
* An **AWS Lambda function** (Python) to delete logs >7 days old & >10 KB
* An **Amazon EventBridge rule** to trigger the Lambda daily

**📂 Terraform Setup**

**1. Project Structure**

s3-cleanup-project/

├── main.tf

├── variables.tf

├── outputs.tf

└── lambda/

└── cleanup.py

**2. Lambda Code (lambda/cleanup.py)**

import boto3

import datetime

import os

s3 = boto3.client('s3')

BUCKET\_NAME = os.environ['BUCKET\_NAME']

def lambda\_handler(event, context):

today = datetime.datetime.now(datetime.timezone.utc)

paginator = s3.get\_paginator('list\_objects\_v2')

page\_iterator = paginator.paginate(Bucket=BUCKET\_NAME)

objects\_to\_delete = []

for page in page\_iterator:

if "Contents" in page:

for obj in page["Contents"]:

key = obj["Key"]

size = obj["Size"]

last\_modified = obj["LastModified"]

age\_days = (today - last\_modified).days

if age\_days > 7 and size > 10 \* 1024:

objects\_to\_delete.append({"Key": key})

if objects\_to\_delete:

for i in range(0, len(objects\_to\_delete), 1000):

batch = objects\_to\_delete[i:i+1000]

s3.delete\_objects(Bucket=BUCKET\_NAME, Delete={"Objects": batch})

print(f"Deleted {len(batch)} objects")

else:

print("No matching objects found.")

**3. Terraform Code (main.tf)**

provider "aws" {

region = "us-east-1"

}

resource "aws\_s3\_bucket" "logs" {

bucket = "my-log-bucket-12345"

}

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

name = "s3-cleanup-lambda-role"

assume\_role\_policy = jsonencode({

Version = "2012-10-17",

Statement = [{

Action = "sts:AssumeRole",

Effect = "Allow",

Principal = {

Service = "lambda.amazonaws.com"

}

}]

})

}

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

name = "s3-cleanup-policy"

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

policy = jsonencode({

Version = "2012-10-17",

Statement = [

{

Effect = "Allow",

Action = [

"s3:ListBucket",

"s3:DeleteObject"

],

Resource = [

aws\_s3\_bucket.logs.arn,

"${aws\_s3\_bucket.logs.arn}/\*"

]

},

{

Effect = "Allow",

Action = [

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents"

],

Resource = "\*"

}

]

})

}

data "archive\_file" "lambda\_zip" {

type = "zip"

source\_dir = "${path.module}/lambda"

output\_path = "${path.module}/lambda.zip"

}

resource "aws\_lambda\_function" "cleanup" {

function\_name = "s3-cleanup-lambda"

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

handler = "cleanup.lambda\_handler"

runtime = "python3.9"

filename = data.archive\_file.lambda\_zip.output\_path

environment {

variables = {

BUCKET\_NAME = aws\_s3\_bucket.logs.bucket

}

}

}

resource "aws\_cloudwatch\_event\_rule" "daily" {

name = "daily-s3-cleanup"

schedule\_expression = "cron(0 0 \* \* ? \*)" # Every day at midnight UTC

}

resource "aws\_cloudwatch\_event\_target" "lambda\_target" {

rule = aws\_cloudwatch\_event\_rule.daily.name

target\_id = "s3-cleanup"

arn = aws\_lambda\_function.cleanup.arn

}

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

statement\_id = "AllowExecutionFromEventBridge"

action = "lambda:InvokeFunction"

function\_name = aws\_lambda\_function.cleanup.function\_name

principal = "events.amazonaws.com"

source\_arn = aws\_cloudwatch\_event\_rule.daily.arn

}

**4. Deploy with Terraform**

terraform init

terraform apply -auto-approve

✅ **What this does**:

* Creates an S3 bucket for logs
* Deploys a Lambda function with cleanup logic
* Sets up EventBridge to trigger Lambda daily
* Deletes S3 objects older than 7 days AND larger than 10 KB

👉 Do you want me to also add an **SNS notification** so you get an email every time logs are deleted (with number of files deleted)?

**You said:**

import boto3 import datetime import os s3 = boto3.client('s3') BUCKET\_NAME = os.environ['BUCKET\_NAME'] def lambda\_handler(event, context): today = datetime.datetime.now(datetime.timezone.utc) paginator = s3.get\_paginator('list\_objects\_v2') page\_iterator = paginator.paginate(Bucket=BUCKET\_NAME) objects\_to\_delete = [] for page in page\_iterator: if "Contents" in page: for obj in page["Contents"]: key = obj["Key"] size = obj["Size"] last\_modified = obj["LastModified"] age\_days = (today - last\_modified).days if age\_days > 7 and size > 10 \* 1024: objects\_to\_delete.append({"Key": key}) if objects\_to\_delete: for i in range(0, len(objects\_to\_delete), 1000): batch = objects\_to\_delete[i:i+1000] s3.delete\_objects(Bucket=BUCKET\_NAME, Delete={"Objects": batch}) print(f"Deleted {len(batch)} objects") else: print("No matching objects found.") rewrite the code with error handling for clean.py

**ChatGPT said:**

Got it ✅ — I’ll **rewrite your Lambda Python code (cleanup.py)** with proper error handling, logging, and safe pagination.

Here’s the improved version:

import boto3

import datetime

import os

import logging

from botocore.exceptions import ClientError

# Setup logging

logger = logging.getLogger()

logger.setLevel(logging.INFO)

s3 = boto3.client('s3')

BUCKET\_NAME = os.environ.get('BUCKET\_NAME', '')

def lambda\_handler(event, context):

if not BUCKET\_NAME:

logger.error("❌ BUCKET\_NAME environment variable is not set.")

return {"status": "error", "message": "Missing bucket name"}

today = datetime.datetime.now(datetime.timezone.utc)

objects\_to\_delete = []

try:

paginator = s3.get\_paginator('list\_objects\_v2')

page\_iterator = paginator.paginate(Bucket=BUCKET\_NAME)

for page in page\_iterator:

if "Contents" not in page:

continue

for obj in page["Contents"]:

try:

key = obj["Key"]

size = obj["Size"]

last\_modified = obj["LastModified"]

age\_days = (today - last\_modified).days

if age\_days > 7 and size > 10 \* 1024:

objects\_to\_delete.append({"Key": key})

except KeyError as e:

logger.warning(f"⚠️ Skipping object due to missing field: {e}")

except ClientError as e:

logger.error(f"❌ Failed to list objects from bucket {BUCKET\_NAME}: {e}")

return {"status": "error", "message": str(e)}

# Delete in batches of 1000 (S3 limit)

if objects\_to\_delete:

try:

for i in range(0, len(objects\_to\_delete), 1000):

batch = objects\_to\_delete[i:i+1000]

response = s3.delete\_objects(

Bucket=BUCKET\_NAME,

Delete={"Objects": batch}

)

deleted = response.get("Deleted", [])

errors = response.get("Errors", [])

logger.info(f"✅ Deleted {len(deleted)} objects")

if errors:

logger.error(f"❌ Errors while deleting: {errors}")

return {"status": "success", "deleted": len(objects\_to\_delete)}

except ClientError as e:

logger.error(f"❌ Error during delete operation: {e}")

return {"status": "error", "message": str(e)}

else:

logger.info("ℹ️ No matching objects found.")

return {"status": "success", "deleted": 0}

**🔹 Improvements Added**

* **Environment variable check** → fails gracefully if BUCKET\_NAME not set.
* **Try/except with ClientError** → catches AWS API errors.
* **Safe object parsing** → avoids crash if an object has missing metadata.
* **Batch deletion handling** → logs successful deletes & errors.
* **Structured return values** → makes debugging easier in CloudWatch logs.