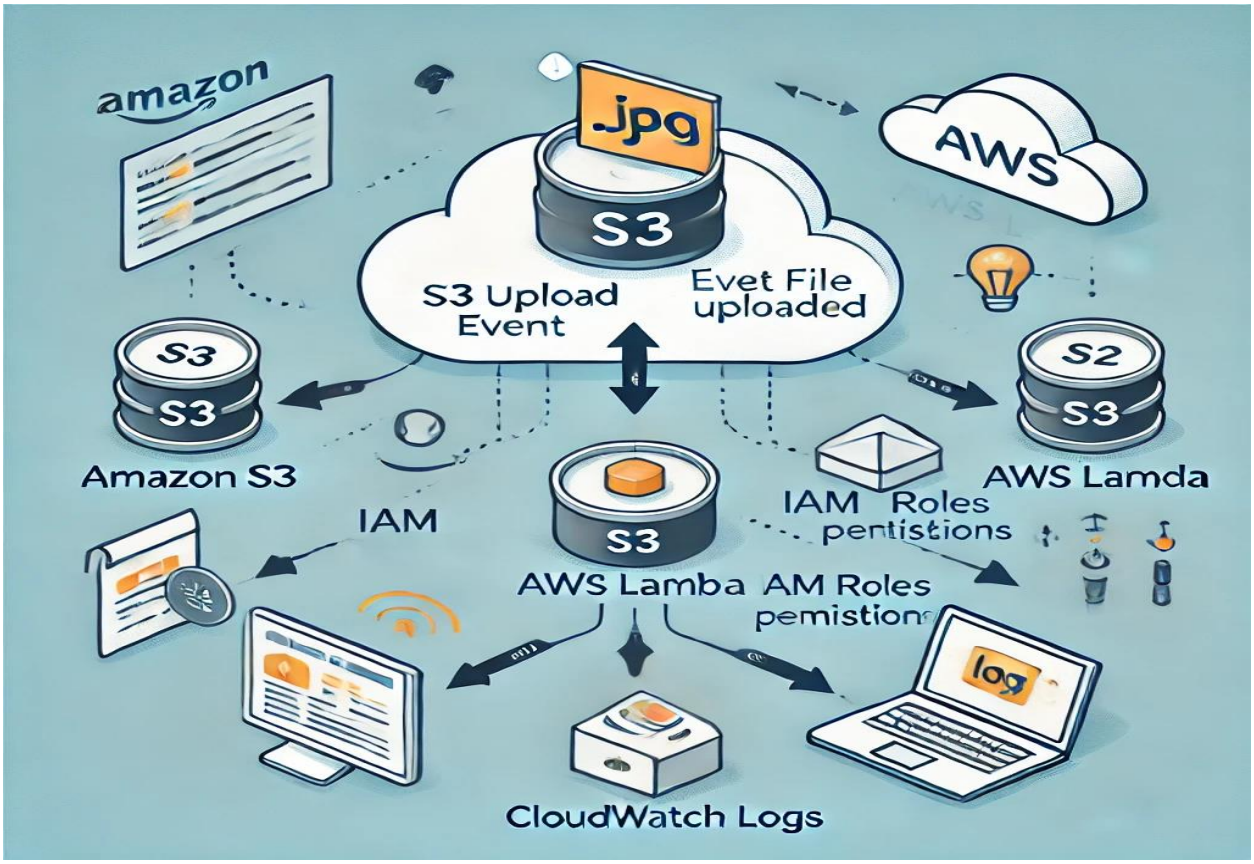
cprime

Event-Driven File Processing with Amazon S3 and AWS Lambda



Name : T MANOJ

EMPID : LYAKE2KHS

Objective

The main purpose of this project is to establish an event-driven design in which an Amazon S3 bucket invokes an AWS Lambda function whenever there is an upload of an object (e.g.,.jpg file). The Lambda function processes the object based on the configured logic. The solution automates the workflow without the need to provision server infrastructure, using IAM roles for permission management and CloudWatch Logs for logging.

Advantages

- Serverless and Event-Driven: No server maintenance needed; Lambda executes only when invoked.
- Cost Efficiency: Pay only for compute time used by Lambda.
- Scalability: Scales automatically with the number of S3 object uploads.
- Low Latency Processing: Low-latency processing of S3 object events.
- Highly Available: AWS manages Lambda and S3 with intrinsic fault tolerance.
- Secure: Fine-grained access control with IAM roles and policies.

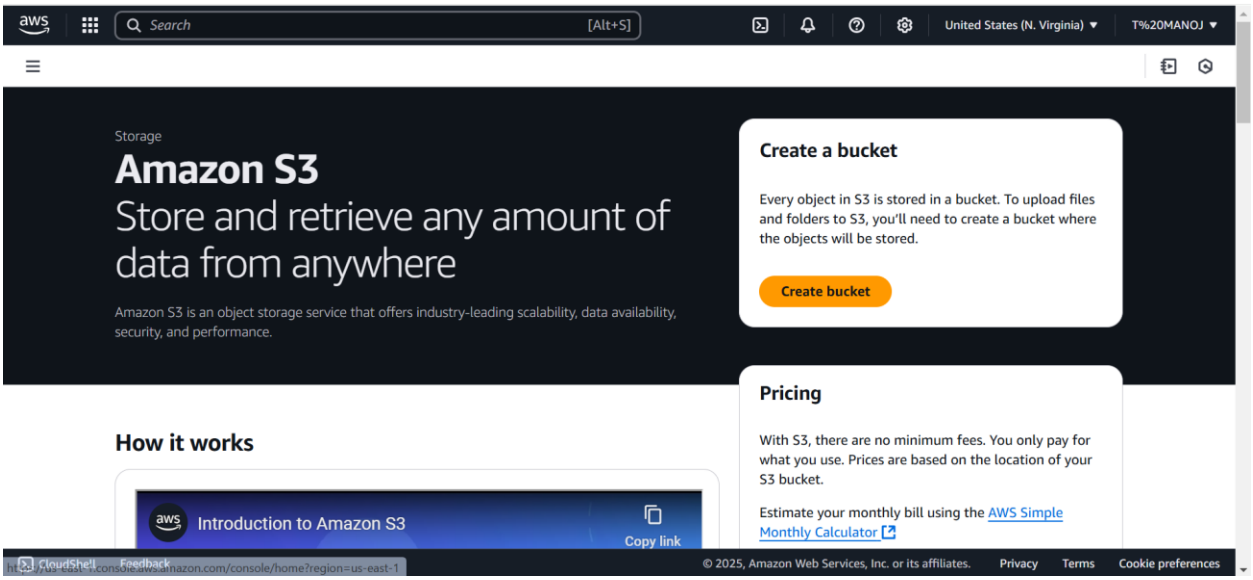
Project Steps:

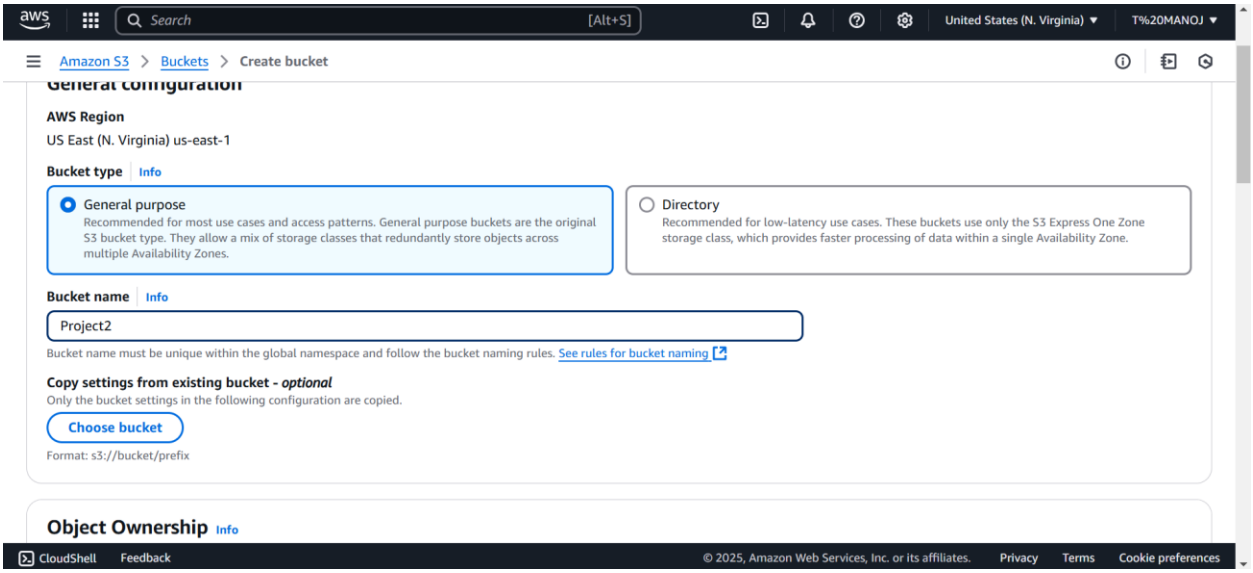
1. Create an Amazon S3 Bucket

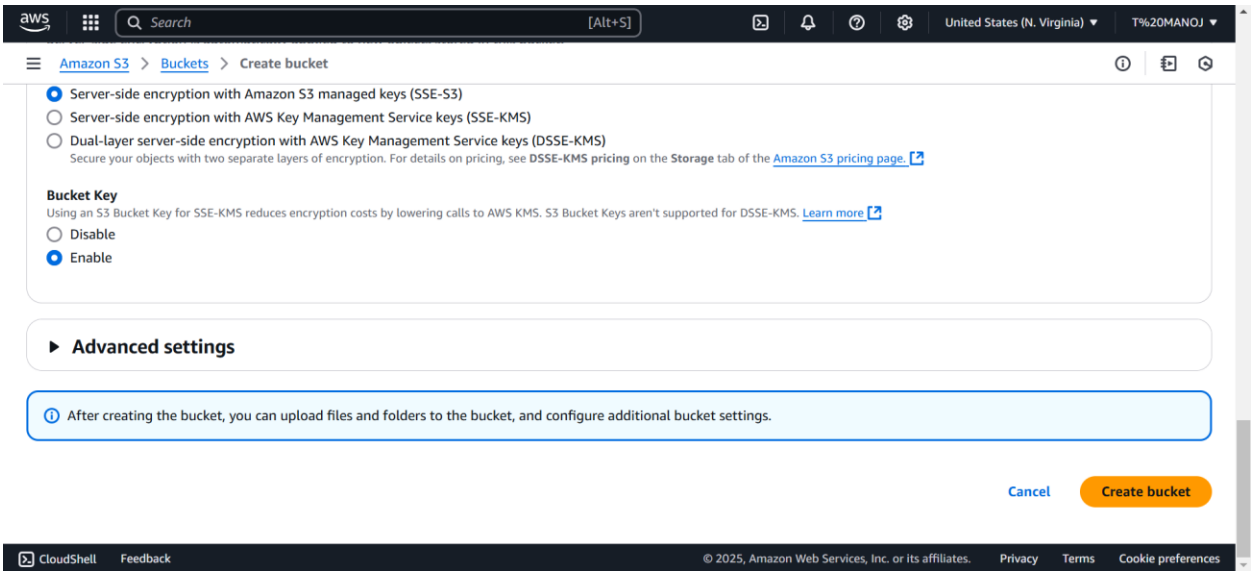
    Service: Amazon S3

    Steps:

    - Navigate to S3 in AWS Console.
    - Click on Create bucket.
    - Enter a unique bucket name .
    - Select the AWS Region nearest to your deployment resources.
    - Leave other options as default .
    - Click on Create bucket.


Search
[Alt+S]
United States (N. Virginia)
T%20MANOJ
Storage
Amazon S3
Store and retrieve any amount of data from anywhere
Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance.
Create a bucket
Every object in S3 is stored in a bucket. To upload files and folders to S3, you'll need to create a bucket where the objects will be stored.
Create bucket
Pricing
With S3, there are no minimum fees. You only pay for what you use. Prices are based on the location of your S3 bucket.
Estimate your monthly bill using the AWS Simple Monthly Calculator
How it works
Introduction to Amazon S3
Copy link
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

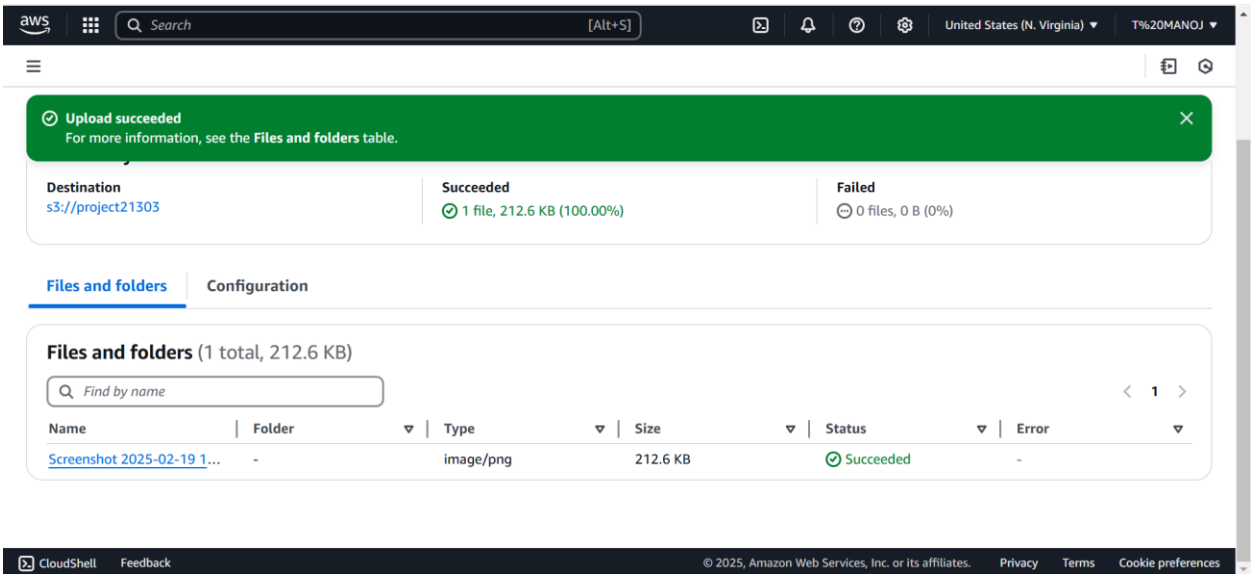

Search
[Alt+S]
United States (N. Virginia)
T%20MANOJ
Amazon S3 > Buckets > Create bucket
General configuration
AWS Region
US East (N. Virginia) us-east-1
Bucket type Info
General purpose
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.
Directory
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.
Bucket name Info
Project2
Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming
Copy settings from existing bucket - optional
Only the bucket settings in the following configuration are copied.
Choose bucket
Format: s3://bucket/prefix
Object Ownership Info
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



2. Upload a Test Object (.jpg File)

Select the bucket created.

- Click on Upload → Add files → select any.jpg file from your local machine.
- Click on Upload.



3. Create an IAM Policy

Service: IAM → Policies

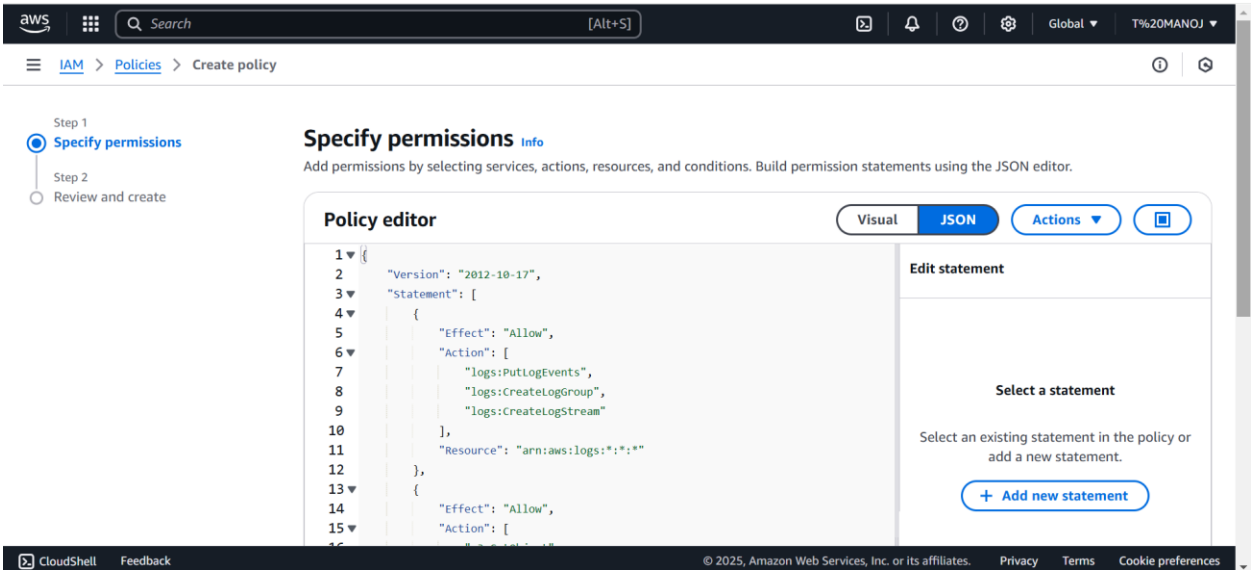
Steps:

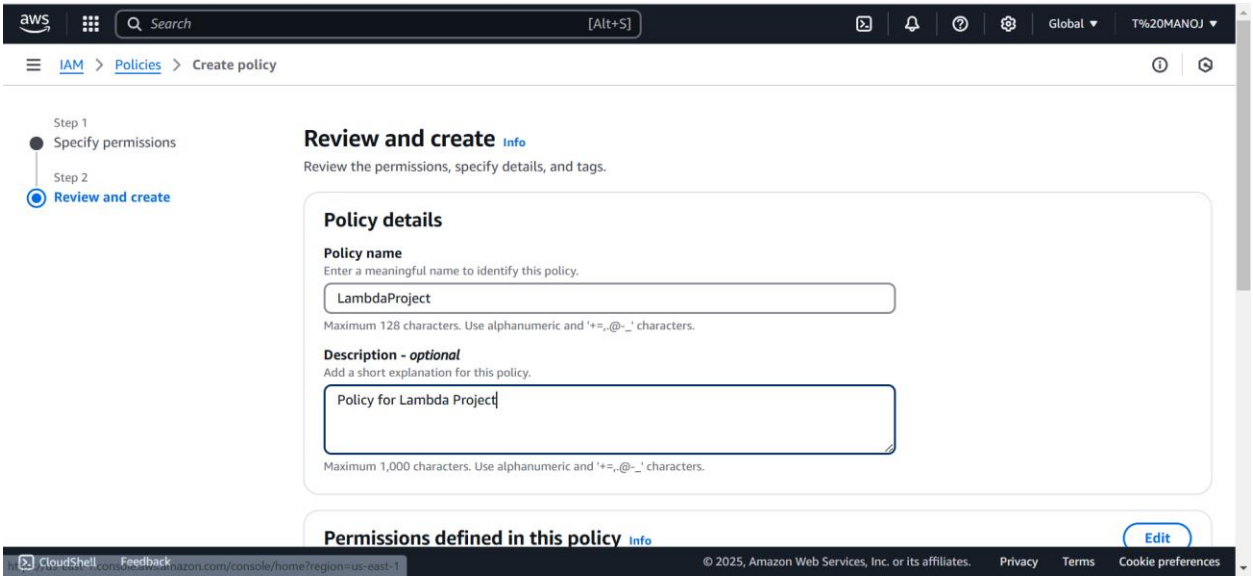
- Click on Create policy.
- Select the JSON tab.
- Paste the following example JSON policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:CreateLogStream"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::*/*"
    }
  ]
}
```

- Policy Name: LambdaS3AccessPolicy.

- Description: Policy for Lambda to read from S3 and write to CloudWatch Logs.
- Click Create policy.



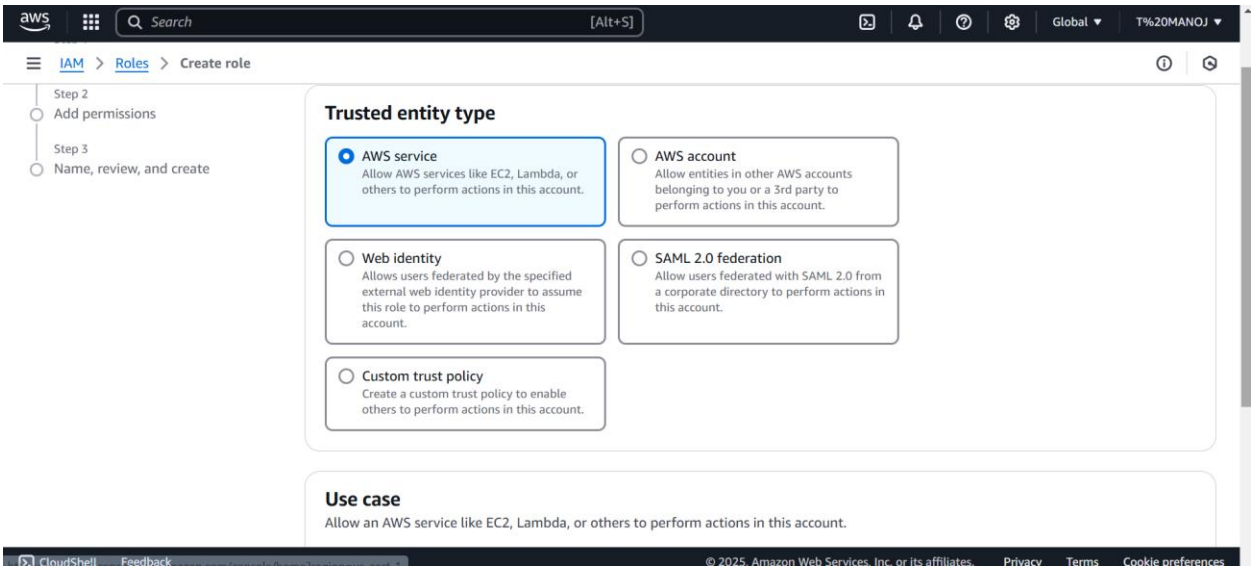


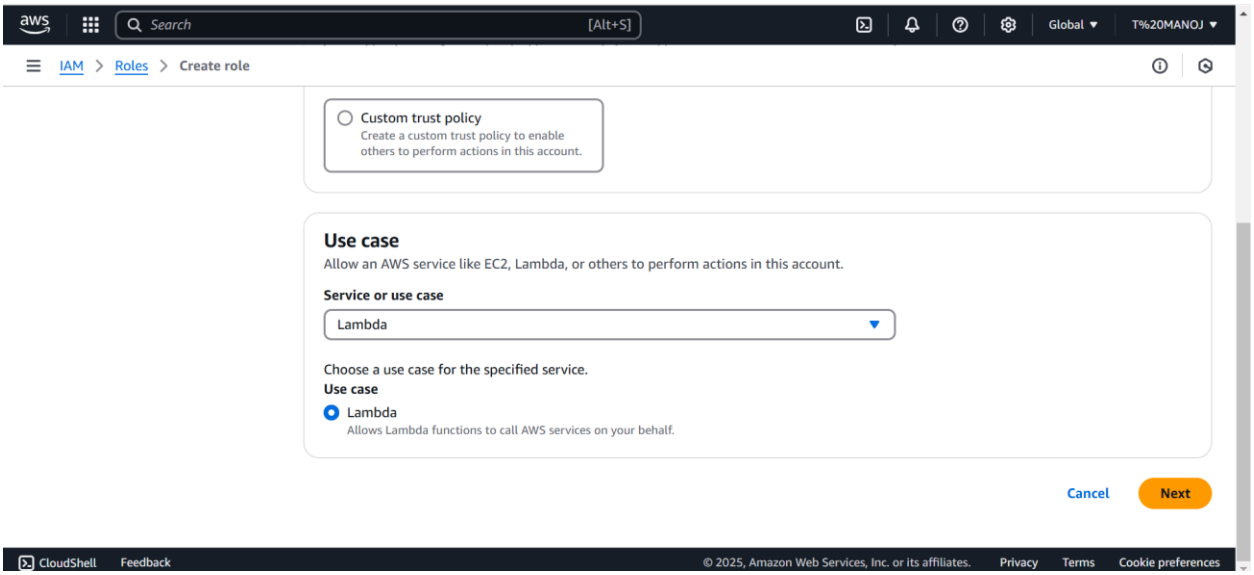
4. Create an IAM Role for Lambda

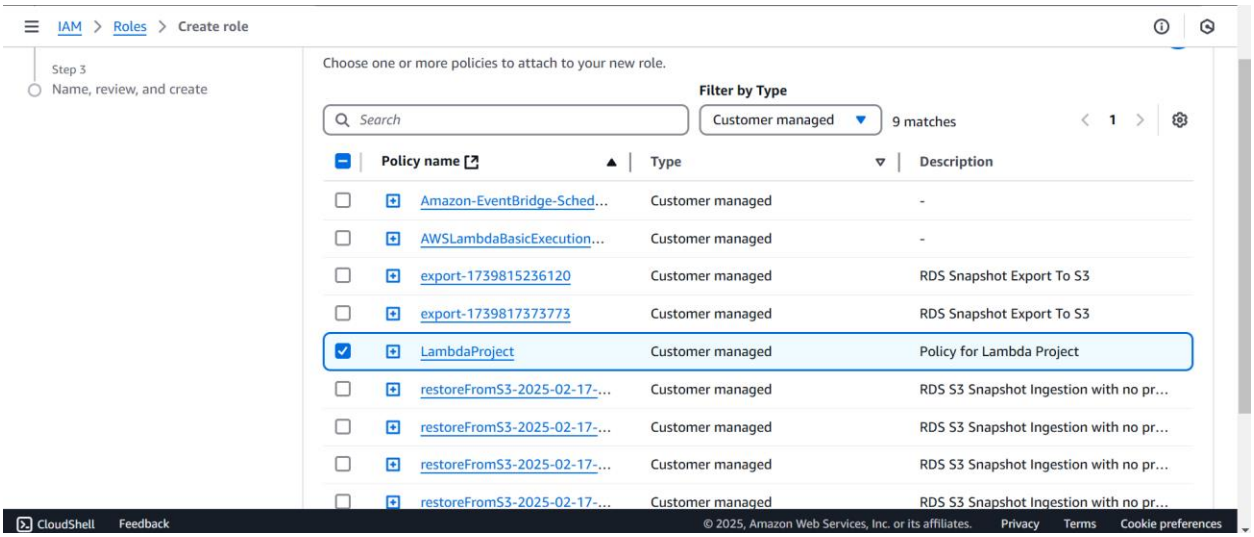
Service: IAM → Roles

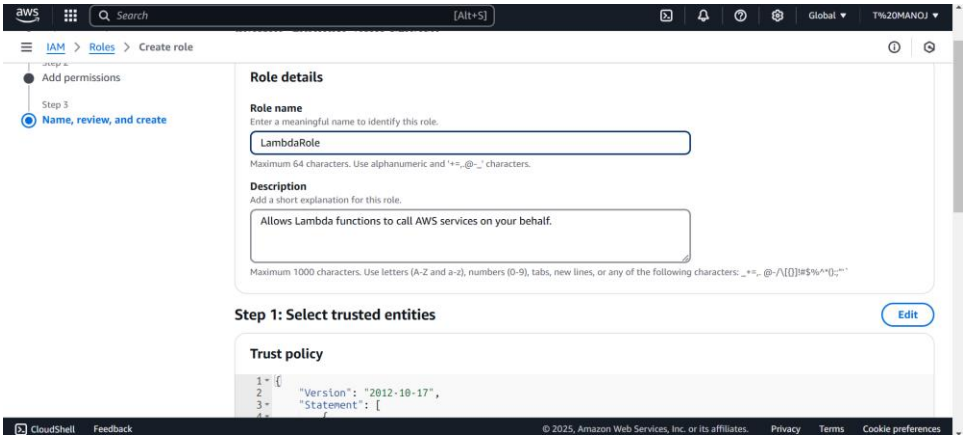
Steps:

- Click Create role.
- Use case: Select Lambda.
- Click Next.
- Attach the LambdaS3AccessPolicy created above.
- Role name: LambdaS3TriggerRole.
- Click Create role.







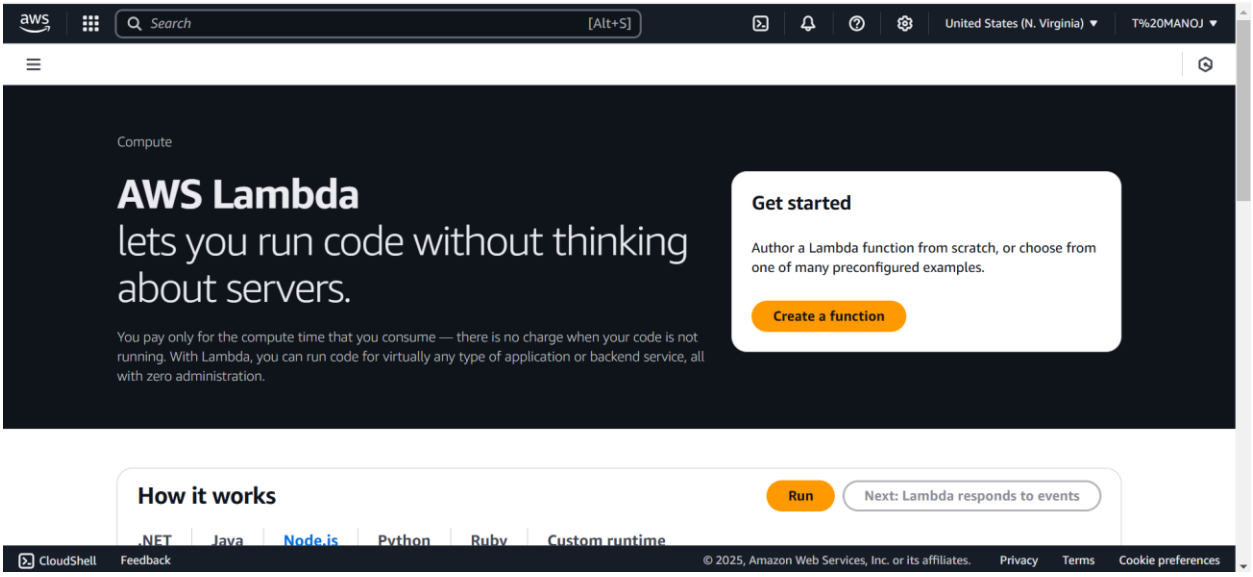


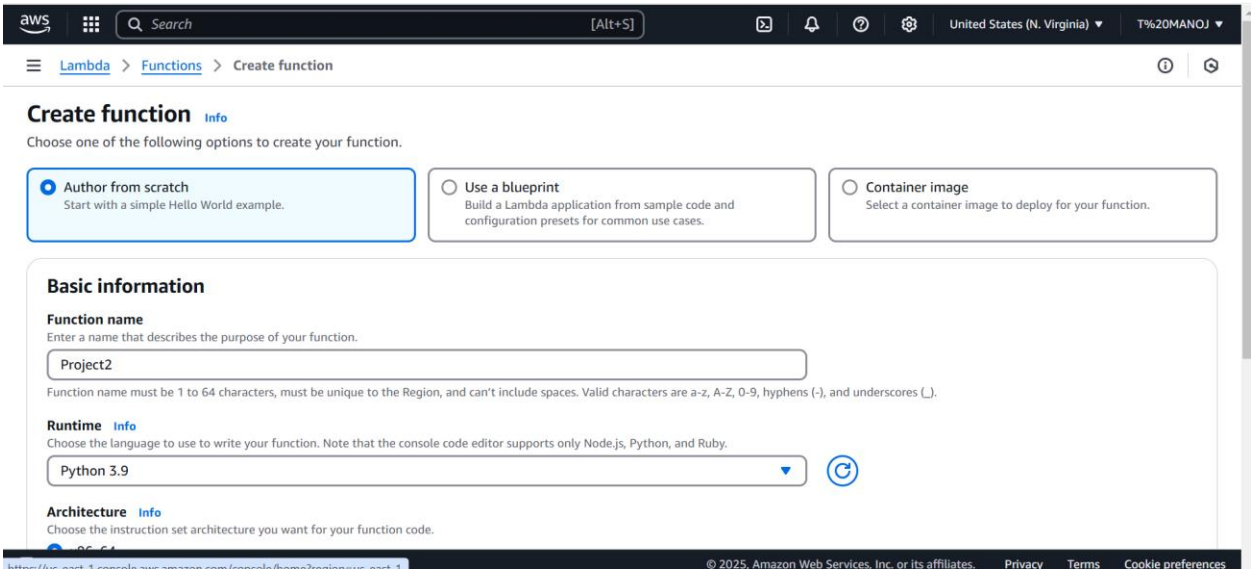
5. Create the Lambda Function

Service: Lambda

Steps:

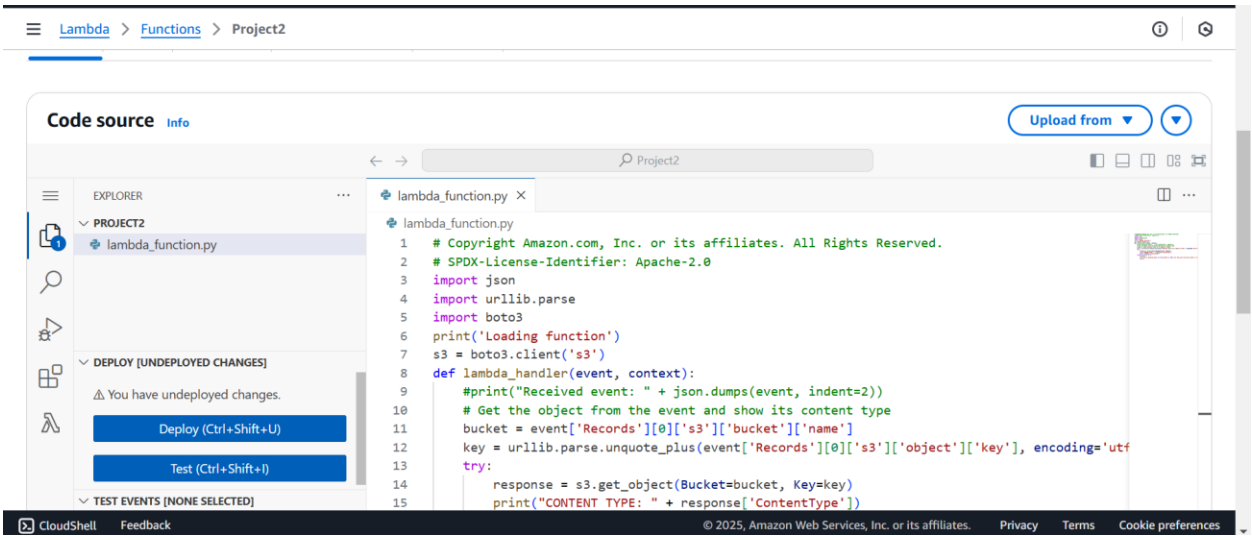
- Click Create function → Author from scratch.
- Function name: S3ObjectProcessor.
- Runtime: Python (3.13).
- Execution role: Choose Use an existing role → select LambdaS3TriggerRole.
- Click Create function.





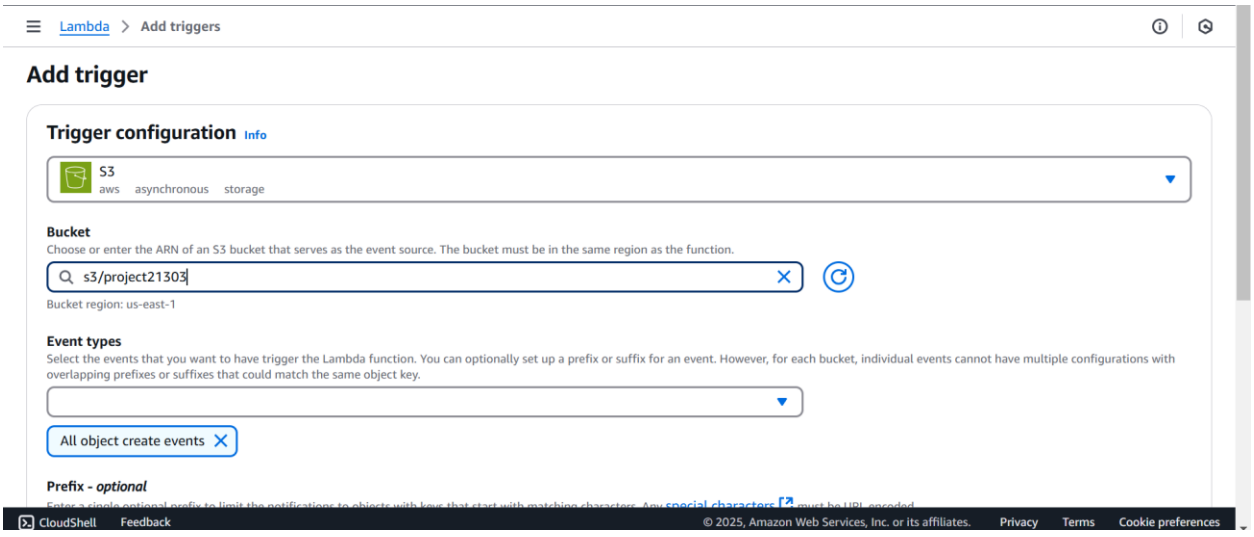
6. Deploy Lambda Code

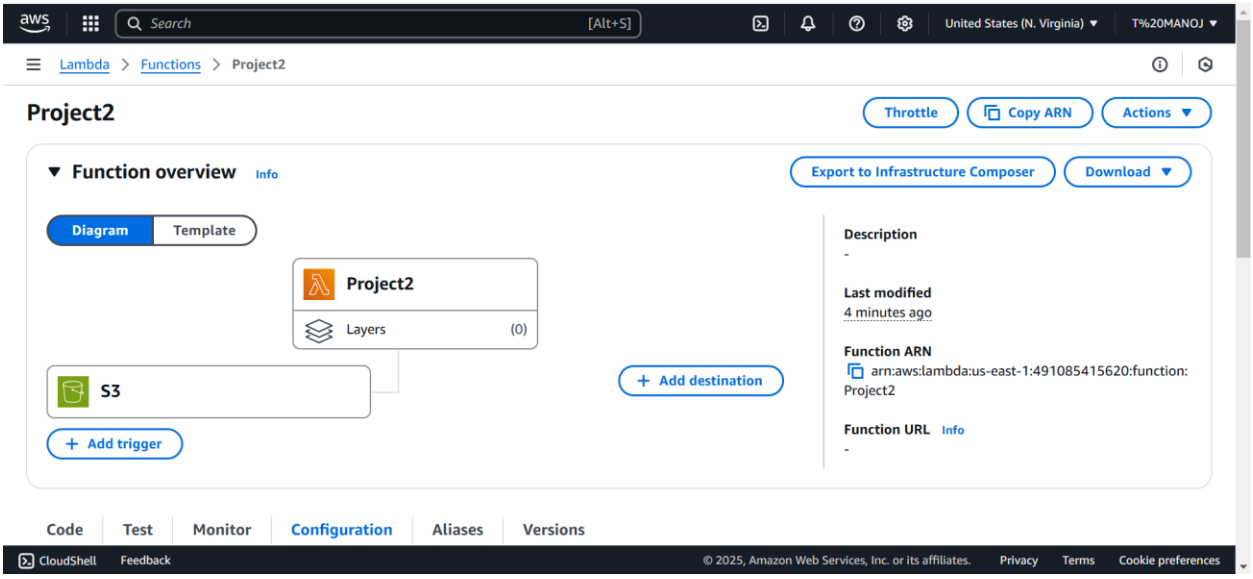
- Sample Lambda Code:
- Steps:
- Paste the code in the Code source section.
- Click Deploy.

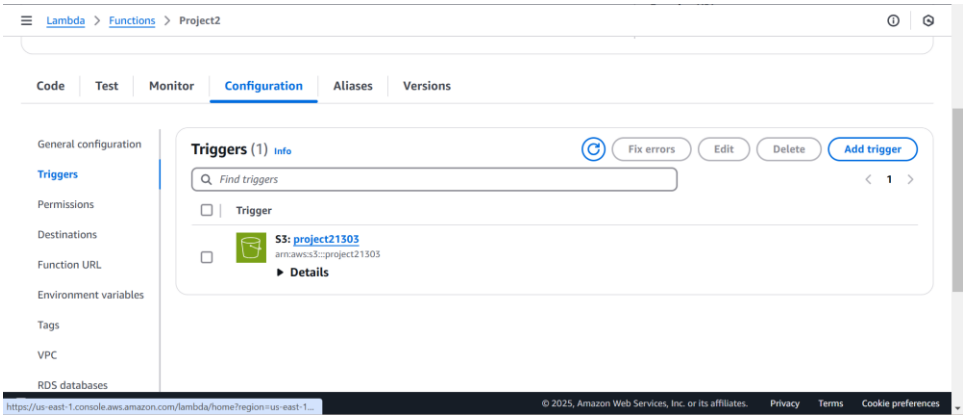


7. Set up S3 Trigger

- In the page of the Lambda function, click Configuration → Triggers → Add trigger.
- Trigger configuration:
- Source: S3
- Bucket: Choose the created bucket.
- Event type: PUT (for object uploads).
- Acknowledge the warning regarding recursive invocation.
- Click Add.







8. Test the Lambda Function

- Test with Dummy Event:
- Click Test → Configure test event.
- Use the following JSON and modify the bucket name and key:

```
{
 "Records": [
  {
   "eventVersion": "2.0",
   "eventSource": "aws:s3",
   "awsRegion": "Bucket region",
```

```
    "eventTime": "1970-01-01T00:00:00.000Z",
    "eventName": "ObjectCreated:Put",
    "userIdentity": {
     "principalId": "EXAMPLE"
    },
    "requestParameters": {
     "sourceIPAddress": "127.0.0.1"
    },
    "responseElements": {
     "x-amz-request-id": "EXAMPLE123456789",
     "x-amz-id-2":
"EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEFGH"
    },
    "s3": {
     "s3SchemaVersion": "1.0",
     "configurationId": "testConfigRule",
     "bucket": {
      "name": "S3 bucket name",
      "ownerIdentity": {
       "principalId": "EXAMPLE"
      },
      "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
     },
     "object": {
      "key": "Object_Key",
      "size": 1024,
      "eTag": "0123456789abcdef0123456789abcdef",
      "sequencer": "0A1B2C3D4E5F678901"
     }
    }
   }
```

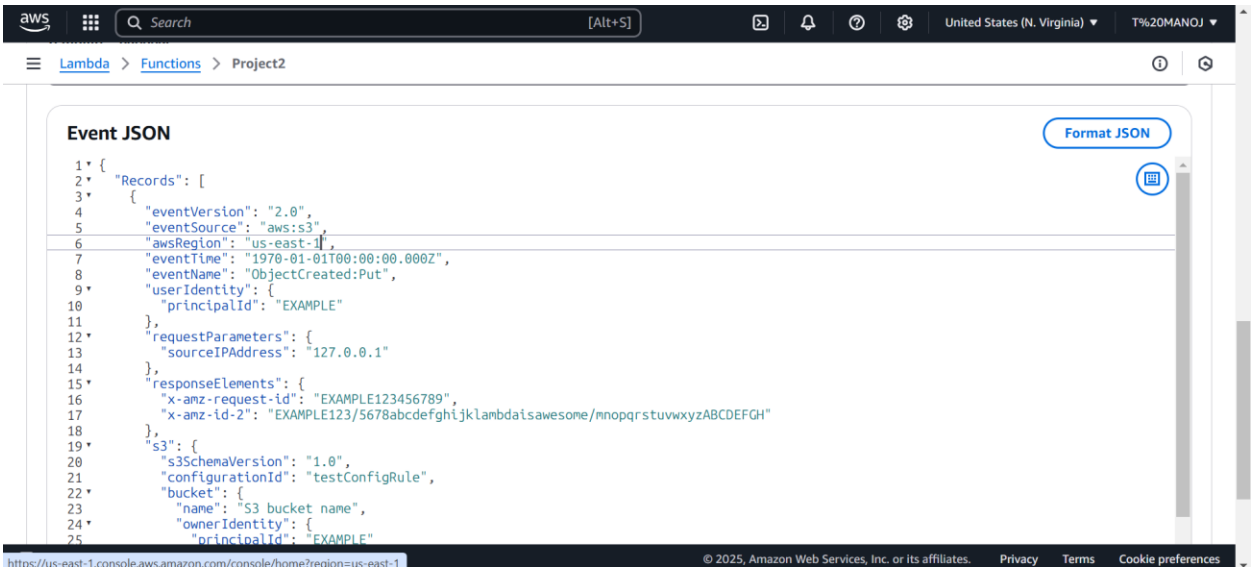
]

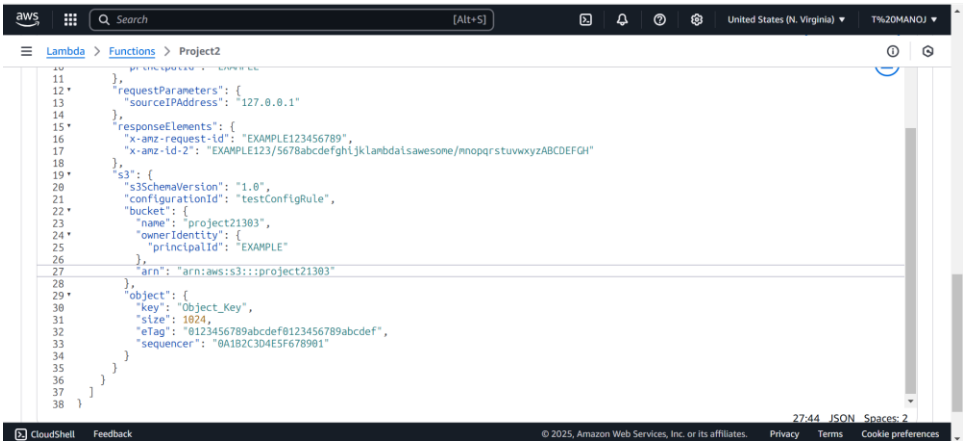
}

DEPLOY CODE :

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import json
import urllib.parse
import boto3print('Loading function')s3 = boto3.client('s3')def lambda_handler(event, context):
  #print("Received event: " + json.dumps(event, indent=2))  # Get the object from the event and show its content type
  bucket = event['Records'][0]['s3']['bucket']['name']
  key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')
  try:
    response = s3.get_object(Bucket=bucket, Key=key)
    print("CONTENT TYPE: " + response['ContentType'])
    return response['ContentType']
  except Exception as e:
    print(e)
    print('Error getting object {} from bucket {}. Make sure they exist and your bucket is in the same region as this function.'.format(key, bucket))
    raise e
```

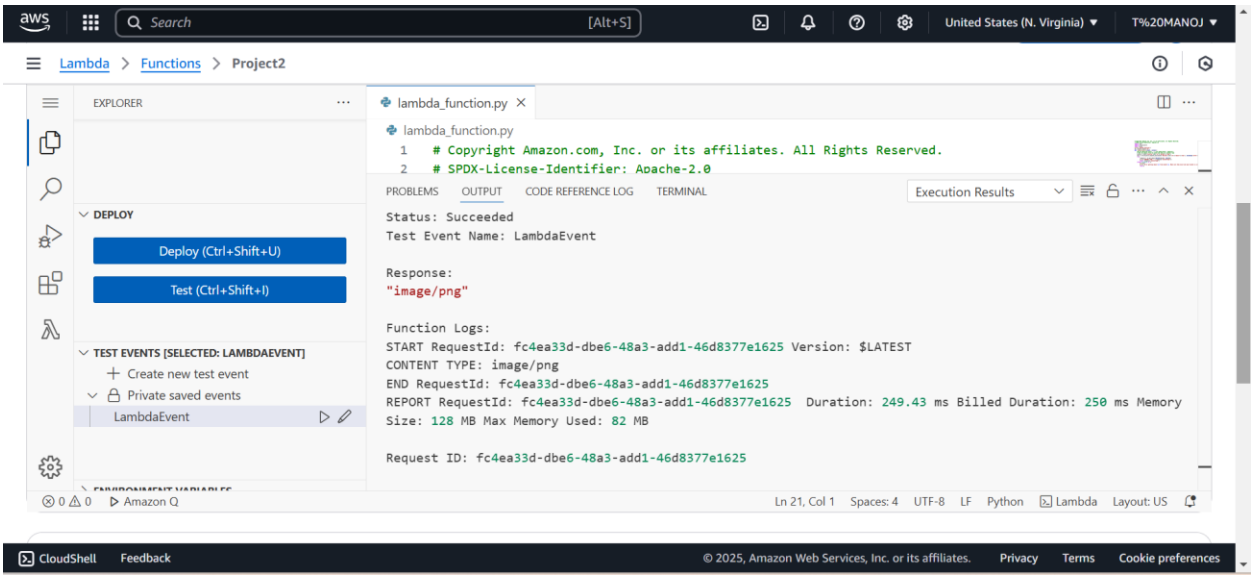
- Test with Actual Trigger:
- Upload a second.jpg file to the S3 bucket.
- Check the Lambda execution logs in CloudWatch.

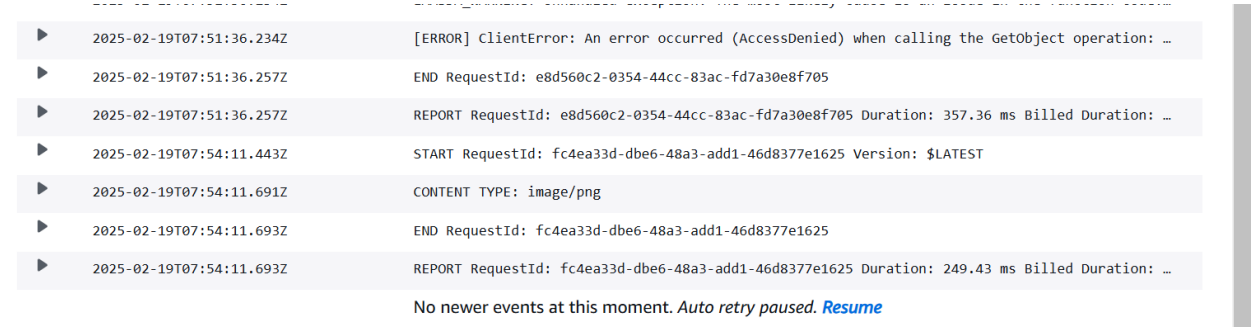




9. View Logs in CloudWatch

- Service: CloudWatch → Logs → Log groups.
- Find the log group for the Lambda function (e.g., /aws/lambda/S3ObjectProcessor).
- Click on the Log stream to see the detailed execution logs.





Use Cases

- Image Processing Pipelines: Resize images automatically upon upload.
- Data Processing: Initiate ETL jobs upon new data upload.
- Document Conversion: Translate uploaded documents (e.g., .docx to .pdf).
- Real-Time Notifications: Provide alerts or notifications upon adding of certain files.
- Audit and Logging: Automatically log upload information for compliance.

Conclusion :

- This project showcases the easy integration of AWS Lambda with Amazon S3 to develop real-time event-driven applications. Utilizing IAM roles for access security, CloudWatch Logs for monitoring, and Python as runtime for Lambda, the solution creates a scalable, cost-saving, and automated file processing workflow. These kinds of architectures are most suitable for current applications needing rapid, consistent, and serverless event processing.