Terraform To Deploy AWS Lambda Function With S3 Trigger

Scenario

Our team receives numerous files from a third-party vendor, who uploads them to an S3 bucket. These files are suffixed with date stamps. Over time, we accumulated over a thousand files, which presented a challenge since S3 doesn’t allow sorting objects by date when there are over 1,000 objects.

Our team performs daily checks, downloading the current day’s file to process the information. However, they struggled to sort and locate the latest files efficiently. To address this issue, we developed a Lambda function that organizes files in a specific path into folders structured by year/month/day.

Implementation

- I will use Terraform to provision the Lambda function.
- I will use Python as Lambda runtime.
- Python script will pick the files uploaded to a path and move them to their respective folder with year, month, and date.
- S3 notification will trigger the Lambda (When any new files get uploaded to the bucket on a path)

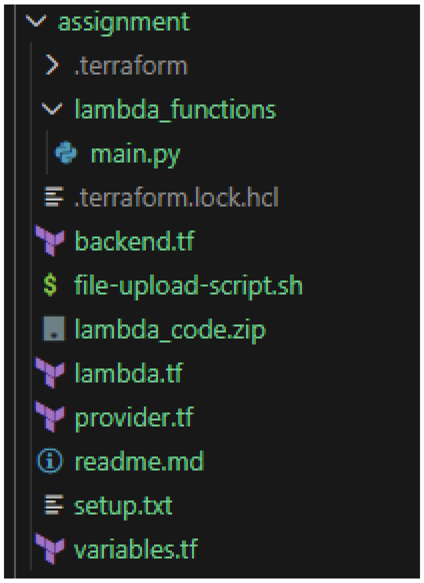
Prerequisite

- Basic understanding of AWS services such as Lambda, S3, IAM, etc.
- Basic understanding of Python and boto3 SDK
- Basic knowledge of Terraform.

Project setup

Creating the file structure as below. .tf files will store the Terraform code, and the Python code will be in lambda_functions/main.py

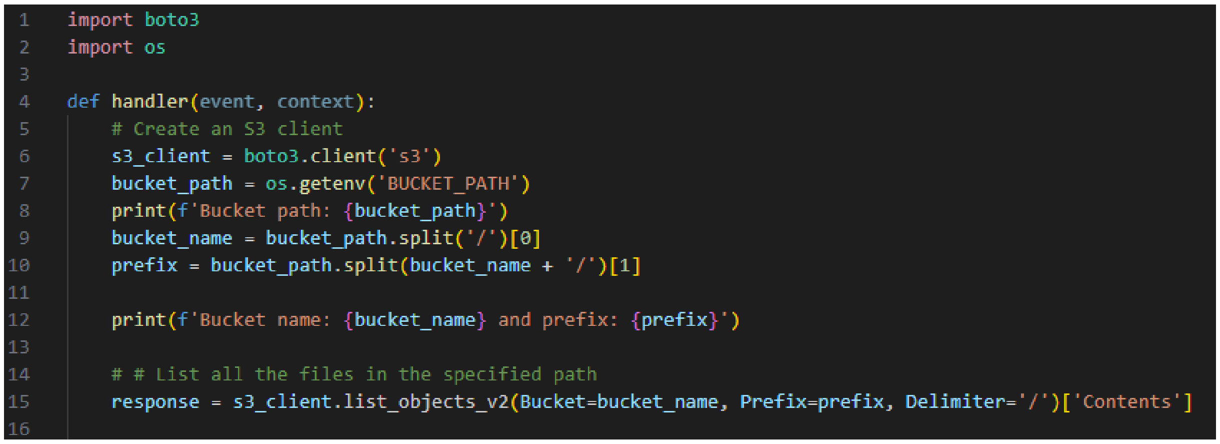
pre-setup-script.sh will be used to test the Lambda function. This script will create random files in the bucket and this event will trigger the Lambda function.



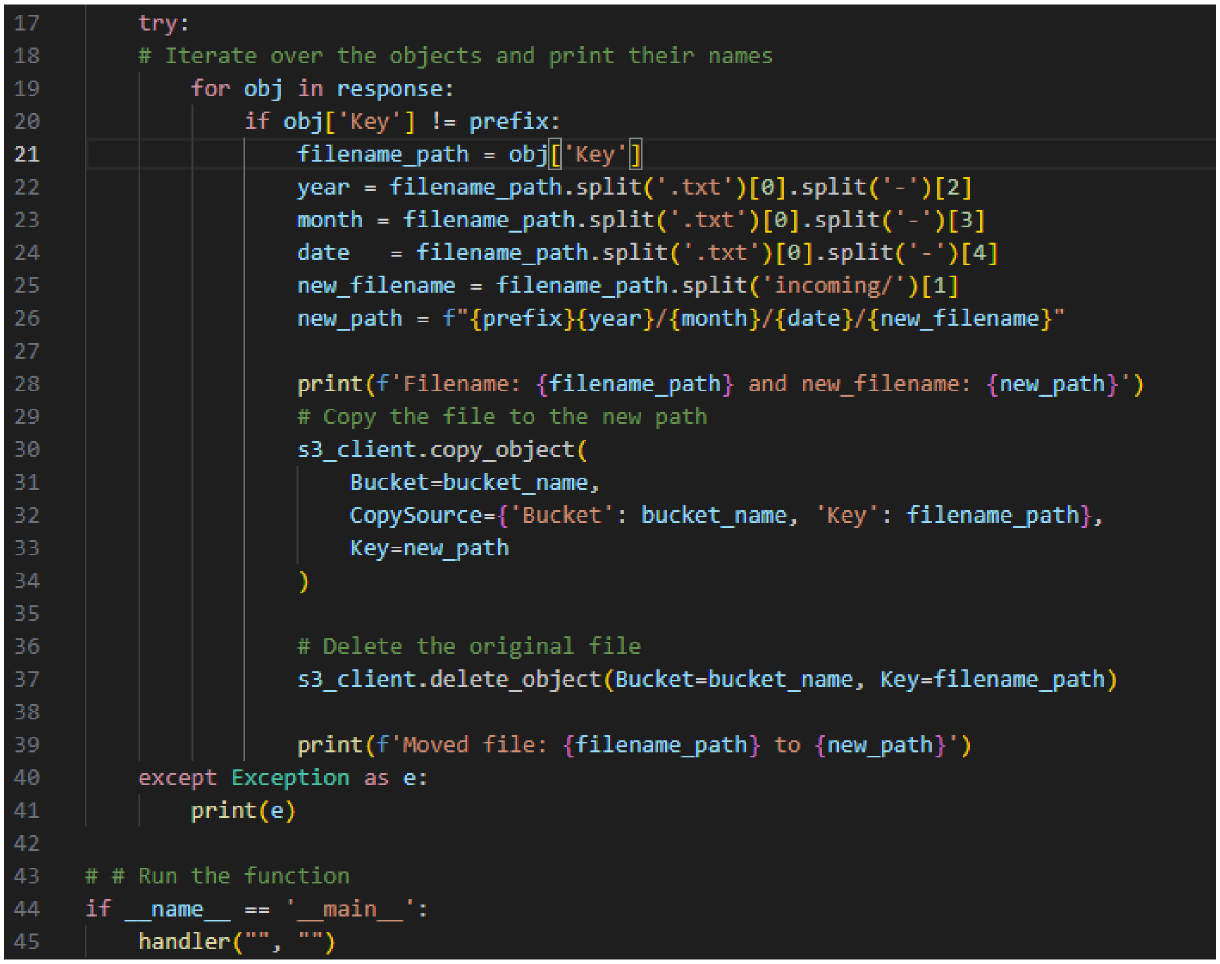
```
assignment
  > .terraform
  lambda_functions
    main.py
  .terraform.lock.hcl
  backend.tf
  file-upload-script.sh
  lambda_code.zip
  lambda.tf
  provider.tf
  readme.md
  setup.txt
  variables.tf
```

Before we write Terraform, let me create a bucket with the name **customer-bucket-307946636515** and a folder **incoming** . I will also create a bucket to store Terraform state—my-backend-devops101-terraform

Python Script for Lambda Function



```python
import boto3
import os

def handler(event, context):
    # Create an S3 client
    s3_client = boto3.client('s3')
    bucket_path = os.getenv('BUCKET_PATH')
    print(f'Bucket path: {bucket_path}')
    bucket_name = bucket_path.split('/')[0]
    prefix = bucket_path.split(bucket_name + '/')[1]

    print(f'Bucket name: {bucket_name} and prefix: {prefix}')

    # # List all the files in the specified path
    response = s3_client.list_objects_v2(Bucket=bucket_name, Prefix=prefix, Delimiter='/')['Contents']

```



```python
    try:
    # Iterate over the objects and print their names
        for obj in response:
            if obj['Key'] != prefix:
                filename_path = obj['Key']
                year = filename_path.split('.txt')[0].split('-')[2]
                month = filename_path.split('.txt')[0].split('-')[3]
                date   = filename_path.split('.txt')[0].split('-')[4]
                new_filename = filename_path.split('incoming/')[1]
                new_path = f"{prefix}{year}/{month}/{date}/{new_filename}"

                print(f'Filename: {filename_path} and new_filename: {new_path}')
                # Copy the file to the new path
                s3_client.copy_object(
                    Bucket=bucket_name,
                    CopySource={'Bucket': bucket_name, 'Key': filename_path},
                    Key=new_path
                )

                # Delete the original file
                s3_client.delete_object(Bucket=bucket_name, Key=filename_path)

                print(f'Moved file: {filename_path} to {new_path}')
    except Exception as e:
        print(e)

# # Run the function
if __name__ == '__main__':
    handler("", "")
```

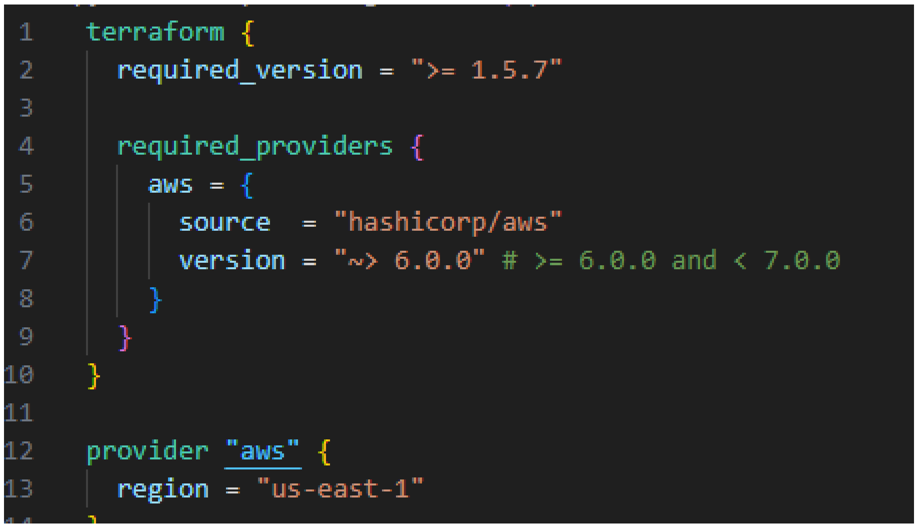
This code will perform the below functions.

- Retrieves the S3 bucket and file path from an environment variable (BUCKET_PATH).
- Lists all files in a specific directory (prefix) within the S3 bucket.

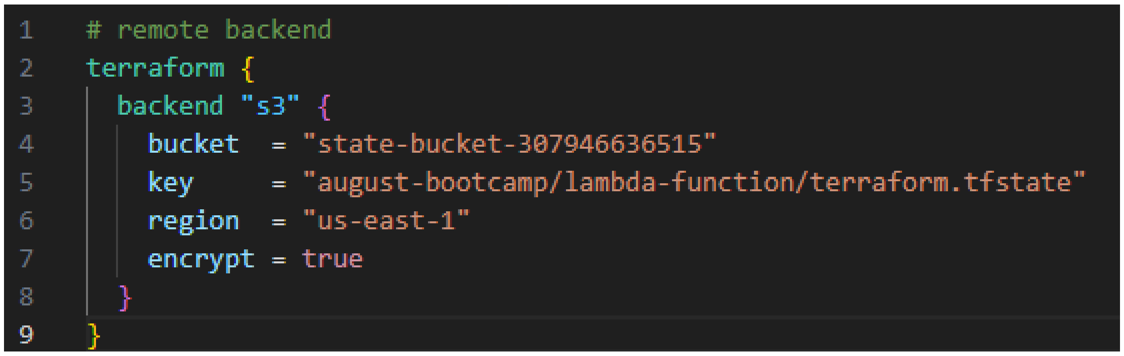
For each file:

- Extracts the year, month, and date from the file name.
- Constructs a new path for the file based on this date information.
- Copies the file to the new path.
- Deletes the original file.

Terraform setup



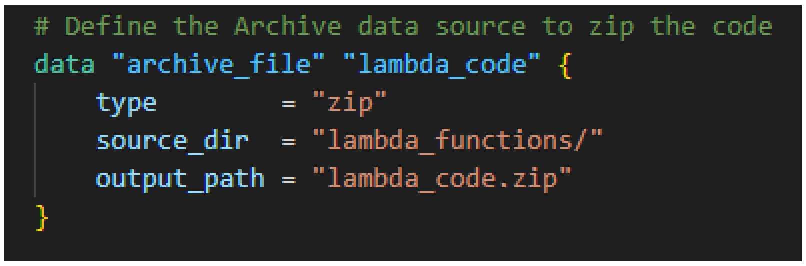
```
terraform {
  required_version = ">= 1.5.7"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 6.0.0" # >= 6.0.0 and < 7.0.0
    }
  }
}

provider "aws" {
  region = "us-east-1"
```



```
# remote backend
terraform {
  backend "s3" {
    bucket  = "state-bucket-307946636515"
    key     = "august-bootcamp/lambda-function/terraform.tfstate"
    region  = "us-east-1"
    encrypt = true
  }
}
```

Package the Python code as a zip file

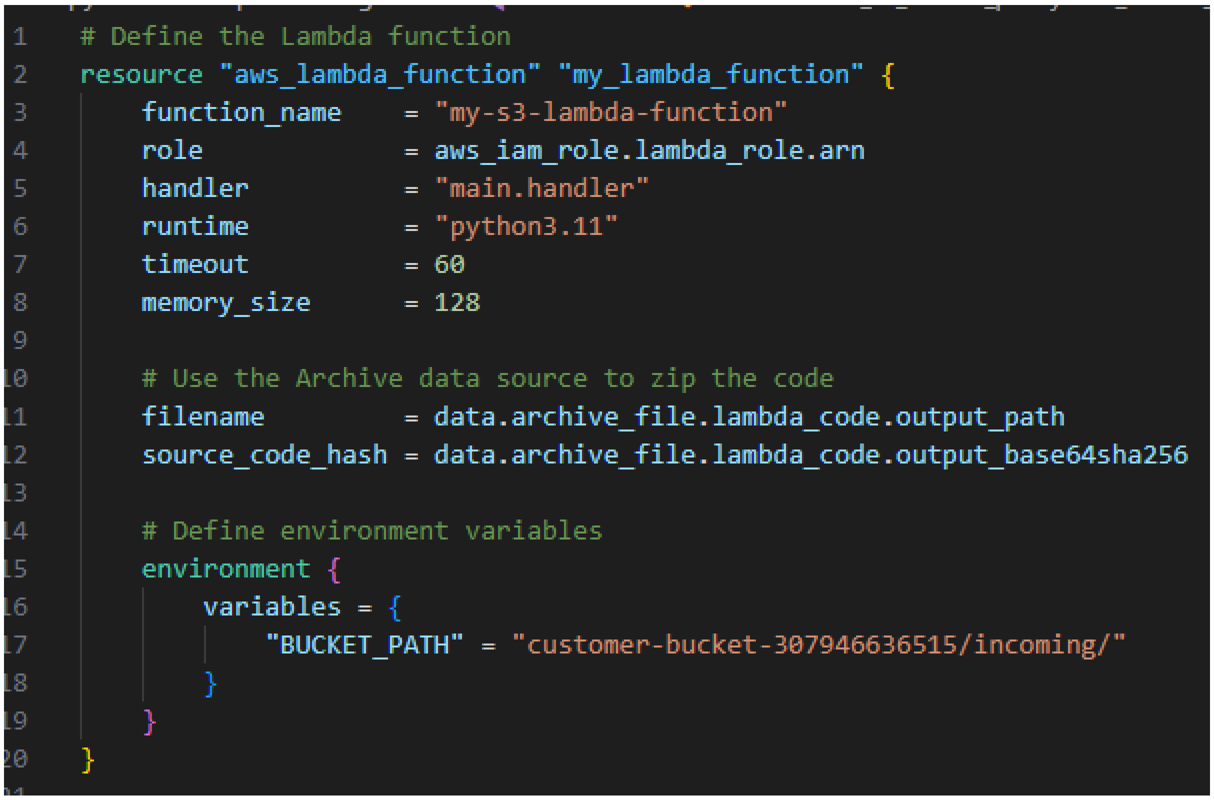
The lambda function will require the zip file with Python code. I have used the Terraform data source archive_file to zip the Python function code. We have provided the path to the Lambda function and the destination where to store the zip file.



```
# Define the Archive data source to zip the code
data "archive_file" "lambda_code" {
    type        = "zip"
    source_dir  = "lambda_functions/"
    output_path = "lambda_code.zip"
}
```

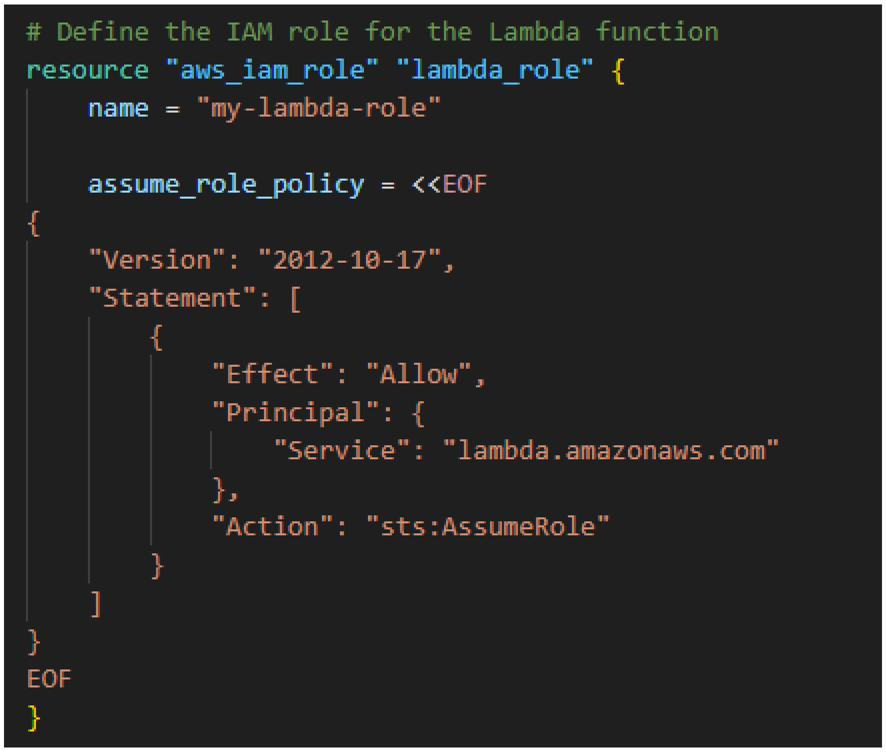
runtime -> python3.11
filename -> location of the zip file with Python code
source_code_hash -> This allows the Terraform to update the Lambda with the code changes

Lambda Creation TF



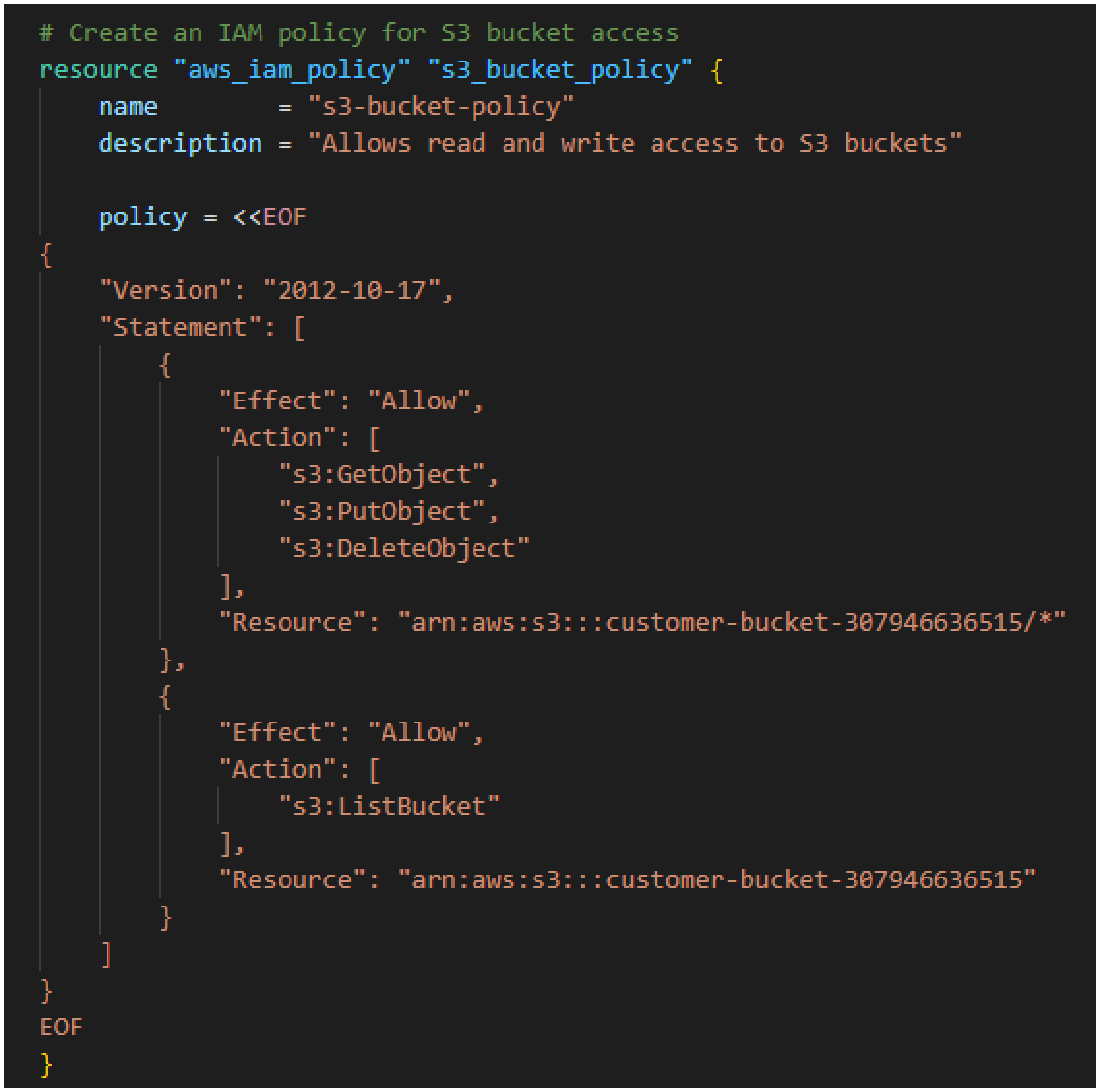
```
# Define the Lambda function
resource "aws_lambda_function" "my_lambda_function" {
    function_name    = "my-s3-lambda-function"
    role             = aws_iam_role.lambda_role.arn
    handler          = "main.handler"
    runtime          = "python3.11"
    timeout          = 60
    memory_size      = 128

    # Use the Archive data source to zip the code
    filename         = data.archive_file.lambda_code.output_path
    source_code_hash = data.archive_file.lambda_code.output_base64sha256

    # Define environment variables
    environment {
        variables = {
            "BUCKET_PATH" = "customer-bucket-307946636515/incoming/"
        }
    }
}
```

Lambda execution role



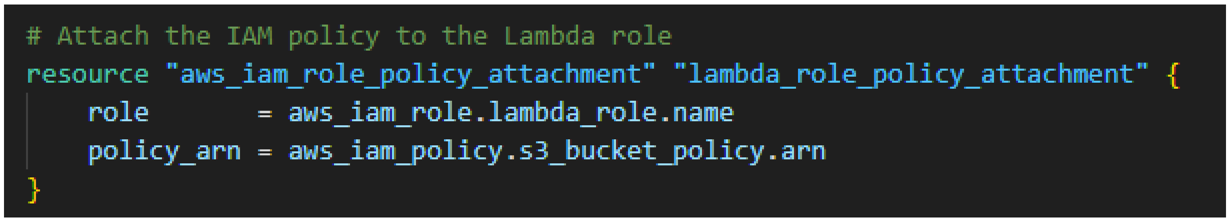
```
# Define the IAM role for the Lambda function
resource "aws_iam_role" "lambda_role" {
    name = "my-lambda-role"

    assume_role_policy = <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
EOF
}
```

Iam policy that allows Lambda to access the files from the bucket



```
# Create an IAM policy for S3 bucket access
resource "aws_iam_policy" "s3_bucket_policy" {
    name        = "s3-bucket-policy"
    description = "Allows read and write access to S3 buckets"

    policy = <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": "arn:aws:s3:::customer-bucket-307946636515/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::customer-bucket-307946636515"
        }
    ]
}
EOF
}
```

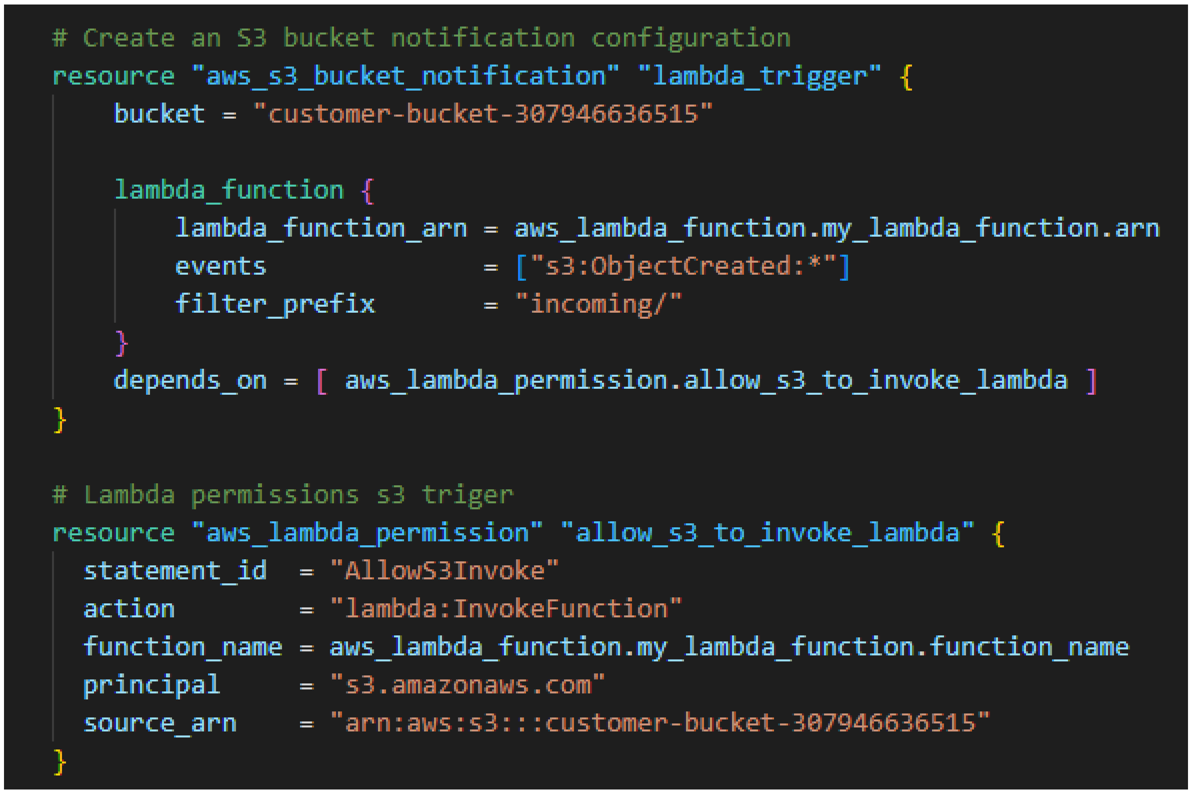
Attach policy with required permissions to the Lambda execution role



```
# Attach the IAM policy to the Lambda role
resource "aws_iam_role_policy_attachment" "lambda_role_policy_attachment" {
    role       = aws_iam_role.lambda_role.name
    policy_arn = aws_iam_policy.s3_bucket_policy.arn
}
```

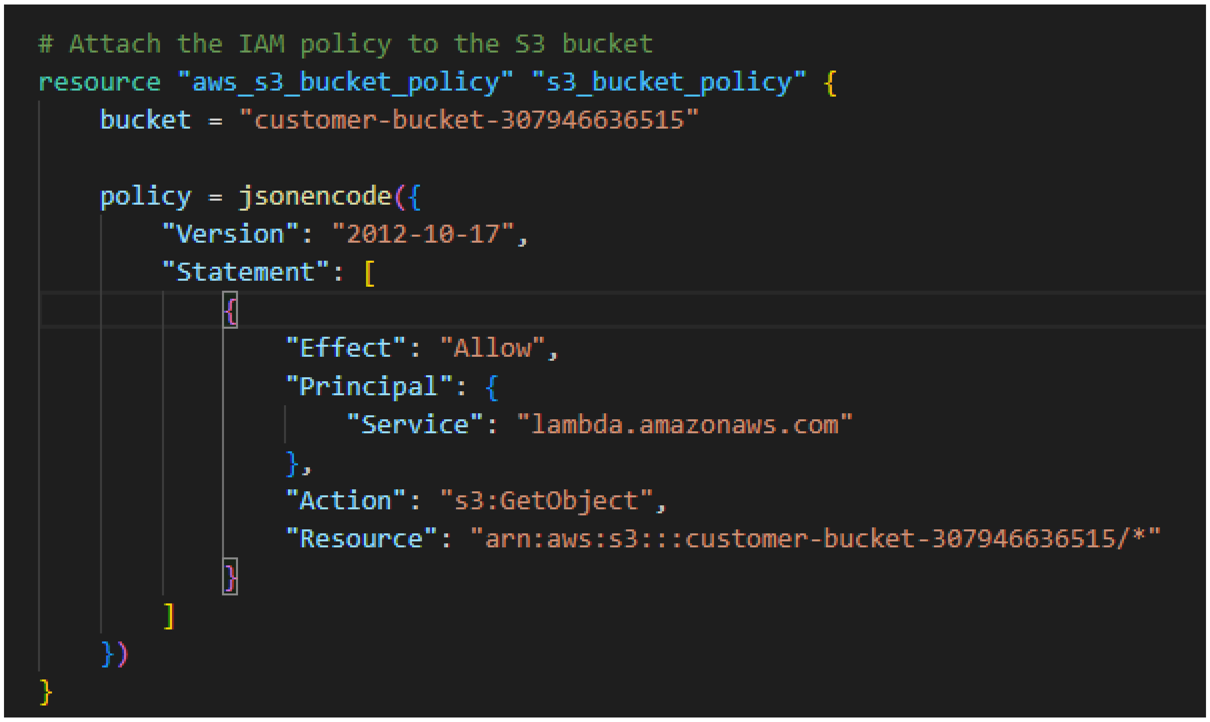
Create an S3 trigger for the Lambda function

Lambda needs explicit permission to be triggered by S3 hence we will create lambda permissions before creating the S3 trigger



```
# Create an S3 bucket notification configuration
resource "aws_s3_bucket_notification" "lambda_trigger" {
    bucket = "customer-bucket-307946636515"

    lambda_function {
        lambda_function_arn = aws_lambda_function.my_lambda_function.arn
        events              = ["s3:ObjectCreated:*"]
        filter_prefix       = "incoming/"
    }
    depends_on = [ aws_lambda_permission.allow_s3_to_invoke_lambda ]
}

# Lambda permissions s3 triger
resource "aws_lambda_permission" "allow_s3_to_invoke_lambda" {
  statement_id  = "AllowS3Invoke"
  action        = "lambda:InvokeFunction"
  function_name = aws_lambda_function.my_lambda_function.function_name
  principal     = "s3.amazonaws.com"
  source_arn    = "arn:aws:s3:::customer-bucket-307946636515"
}
```

Create A bucket policy that will allow Lambda to get the bucket objects for the S3 notification trigger



```
# Attach the IAM policy to the S3 bucket
resource "aws_s3_bucket_policy" "s3_bucket_policy" {
    bucket = "customer-bucket-307946636515"

    policy = jsonencode({
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "lambda.amazonaws.com"
                },
                "Action": "s3:GetObject",
                "Resource": "arn:aws:s3:::customer-bucket-307946636515/*"
            }
        ]
    })
}
```

Configure the AWS credentials

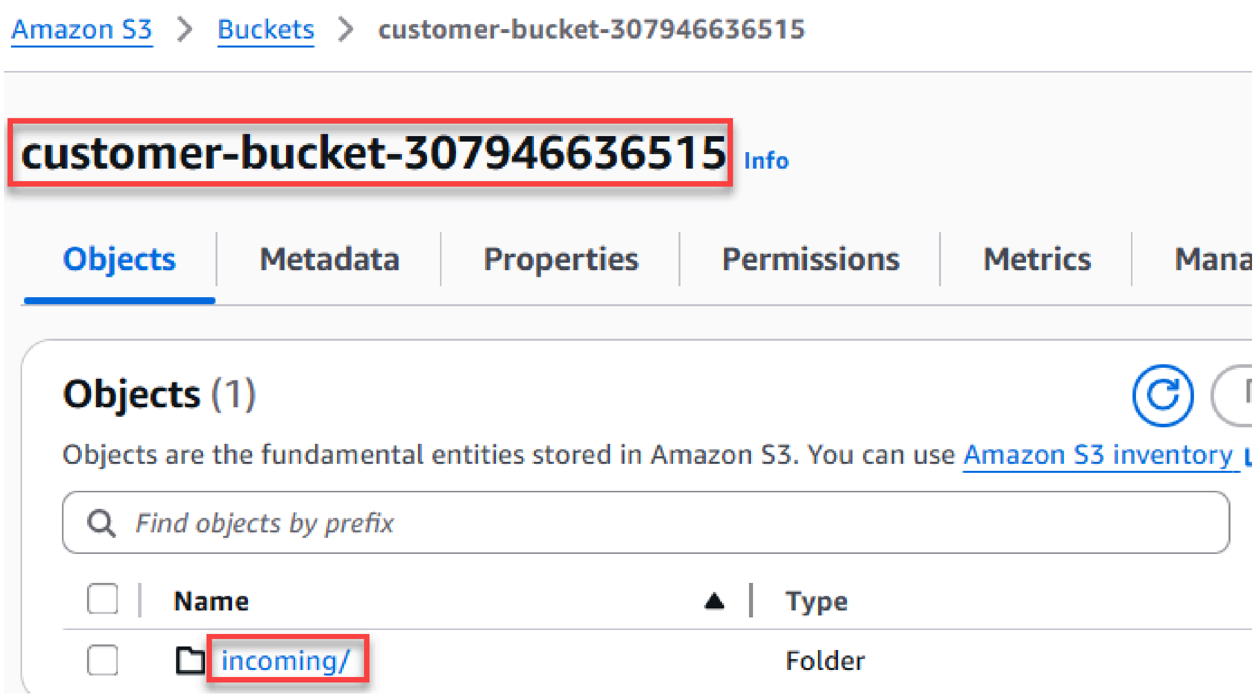
- Create an IAM user with only CLI access
- Create Access and Security key
- Install AWS CLI
- Run aws configure

Deploy the Terraform

```
# Navigate to project directory
cd aws-lambda-with-s3-trigger-terraform

# Initialize Terraform
terraform init

# Review the deployment plan
terraform plan

# Deploy the infrastructure
terraform apply
```

This will deploy all the resources.

S3 Bucket



customer-bucket-307946636515

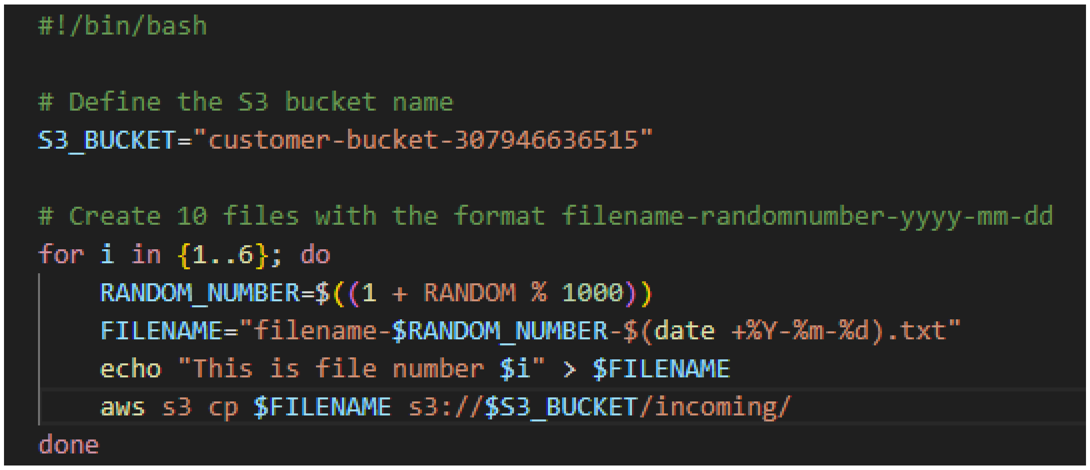
Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory

Find objects by prefix

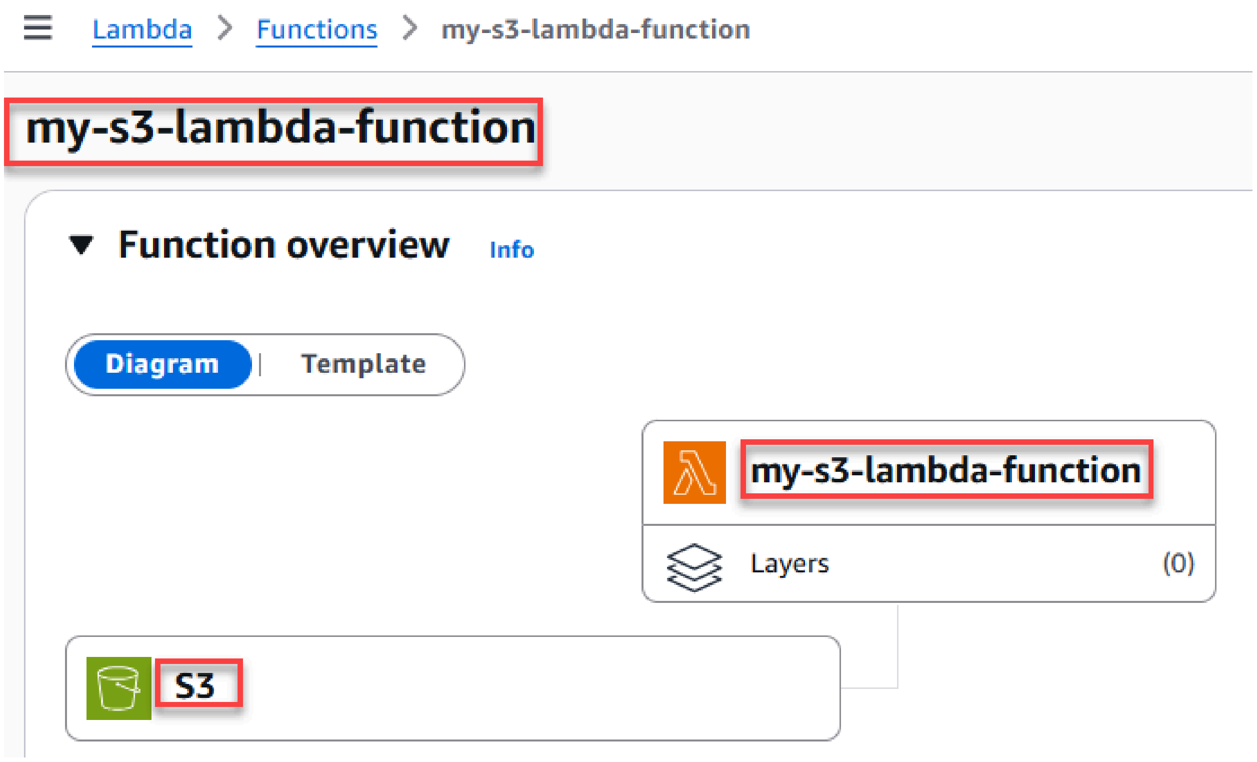
Name	Type
incoming/	Folder

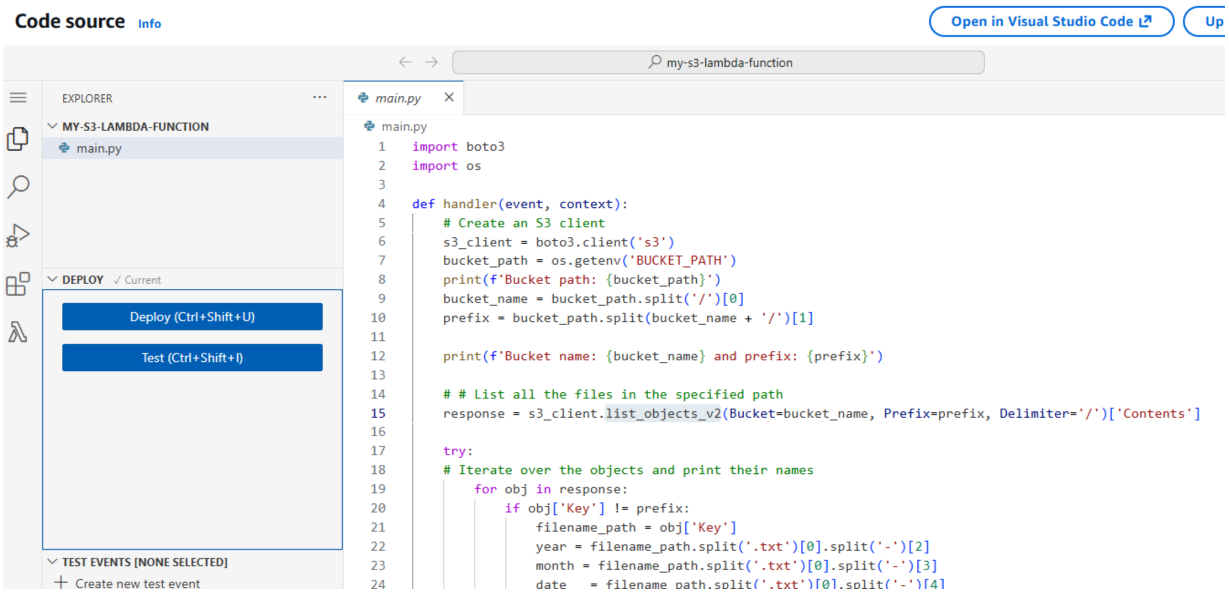
Script to Upload Files into S3 bucket



```bash
#!/bin/bash

# Define the S3 bucket name
S3_BUCKET="customer-bucket-307946636515"

# Create 10 files with the format filename-randomnumber-yyyy-mm-dd
for i in {1..6}; do
    RANDOM_NUMBER=$((1 + RANDOM % 1000))
    FILENAME="filename-$RANDOM_NUMBER-$(date +%Y-%m-%d).txt"
    echo "This is file number $i" > $FILENAME
    aws s3 cp $FILENAME s3://$S3_BUCKET/incoming/
done
```

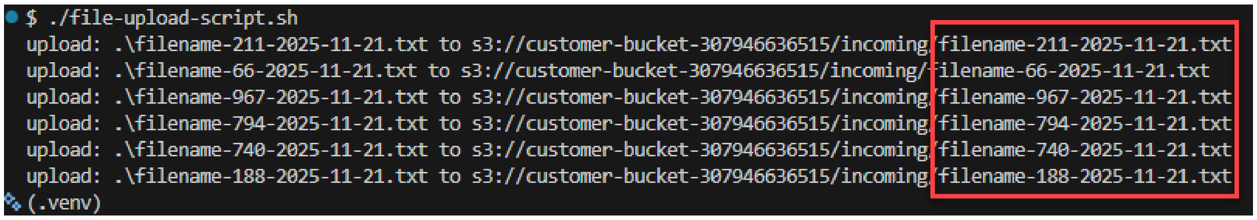
Lambda Function





```python
import boto3
import os

def handler(event, context):
    # Create an S3 client
    s3_client = boto3.client('s3')
    bucket_path = os.getenv('BUCKET_PATH')
    print(f'Bucket path: {bucket_path}')
    bucket_name = bucket_path.split('/')[0]
    prefix = bucket_path.split(bucket_name + '/')[1]

    print(f'Bucket name: {bucket_name} and prefix: {prefix}')

    # # List all the files in the specified path
    response = s3_client.list_objects_v2(Bucket=bucket_name, Prefix=prefix, Delimiter='/')['Contents']

    try:
    # Iterate over the objects and print their names
        for obj in response:
            if obj['Key'] != prefix:
                filename_path = obj['Key']
                year = filename_path.split('.txt')[0].split('-')[2]
                month = filename_path.split('.txt')[0].split('-')[3]
                date   = filename_path.split('.txt')[0].split('-')[4]
```

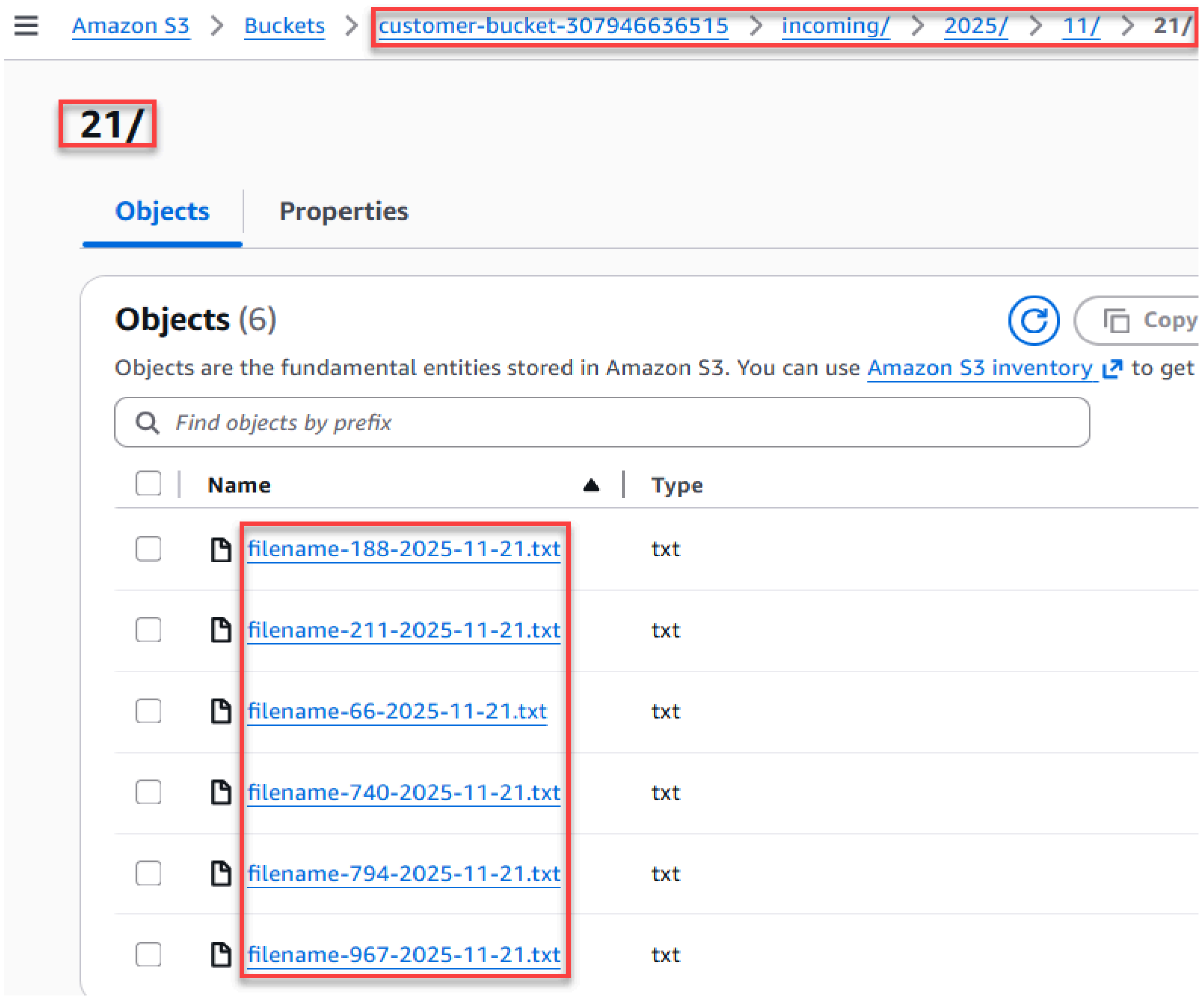
Uploaded Files



```
$ ./file-upload-script.sh
upload: .\filename-211-2025-11-21.txt to s3://customer-bucket-307946636515/incoming/filename-211-2025-11-21.txt
upload: .\filename-66-2025-11-21.txt to s3://customer-bucket-307946636515/incoming/filename-66-2025-11-21.txt
upload: .\filename-967-2025-11-21.txt to s3://customer-bucket-307946636515/incoming/filename-967-2025-11-21.txt
upload: .\filename-794-2025-11-21.txt to s3://customer-bucket-307946636515/incoming/filename-794-2025-11-21.txt
upload: .\filename-740-2025-11-21.txt to s3://customer-bucket-307946636515/incoming/filename-740-2025-11-21.txt
upload: .\filename-188-2025-11-21.txt to s3://customer-bucket-307946636515/incoming/filename-188-2025-11-21.txt
(.venv)
```

This will trigger the lambda function, which will move these files into the new folder structure.

Output



Amazon S3 > Buckets > customer-bucket-307946636515 > incoming/ > 2025/ > 11/ > 21/

21/

Objects | Properties

Objects (6)

Copy

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get

Find objects by prefix

Name	Type
filename-188-2025-11-21.txt	txt
filename-211-2025-11-21.txt	txt
filename-66-2025-11-21.txt	txt
filename-740-2025-11-21.txt	txt
filename-794-2025-11-21.txt	txt
filename-967-2025-11-21.txt	txt