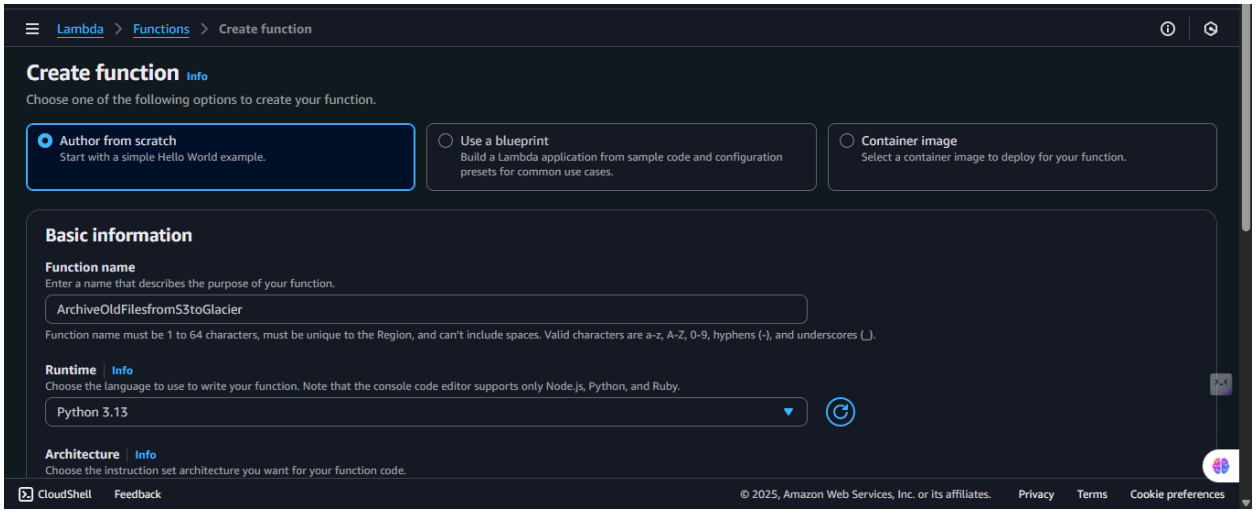
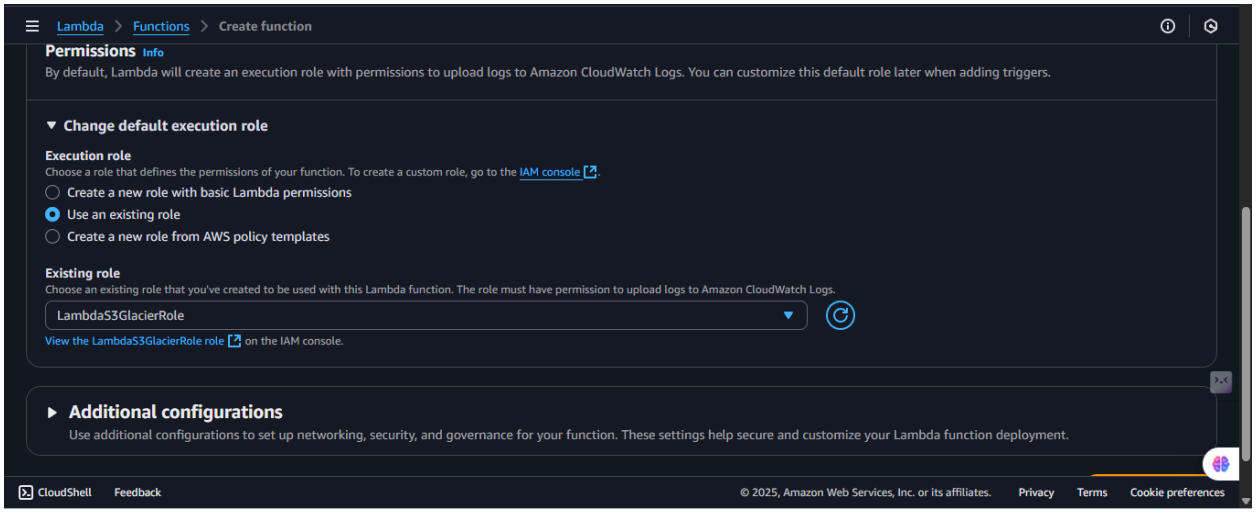
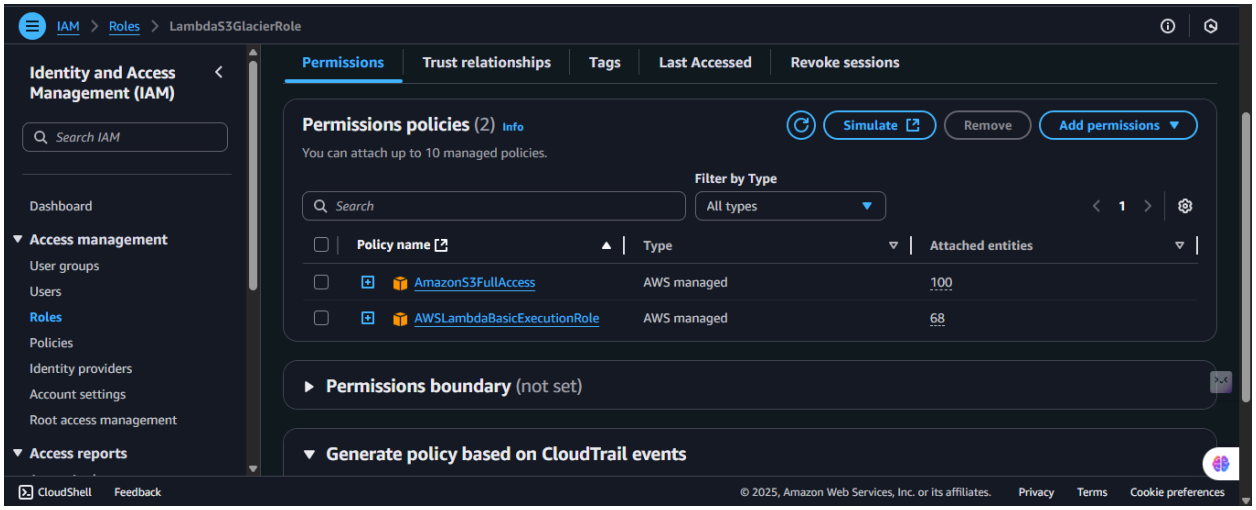
1. Prerequisites

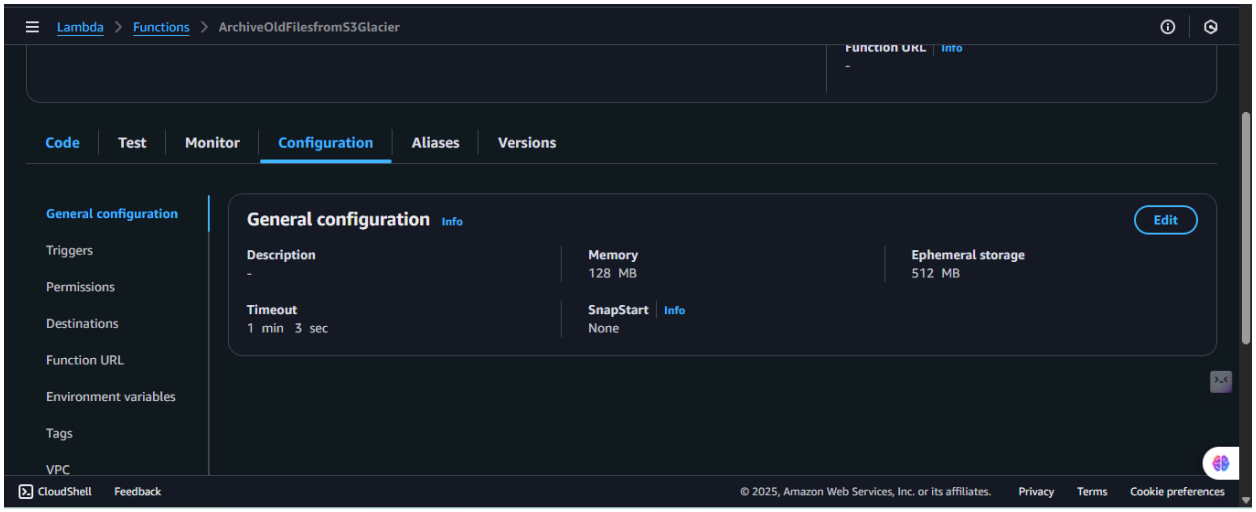
- S3 bucket with test files (some older than 6 months — you can manually edit “Last Modified” date for testing in some cases or use backdated test files).
- IAM Role for Lambda with `AmazonS3FullAccess` and **basic Lambda execution permissions** (`AWSLambdaBasicExecutionRole`).

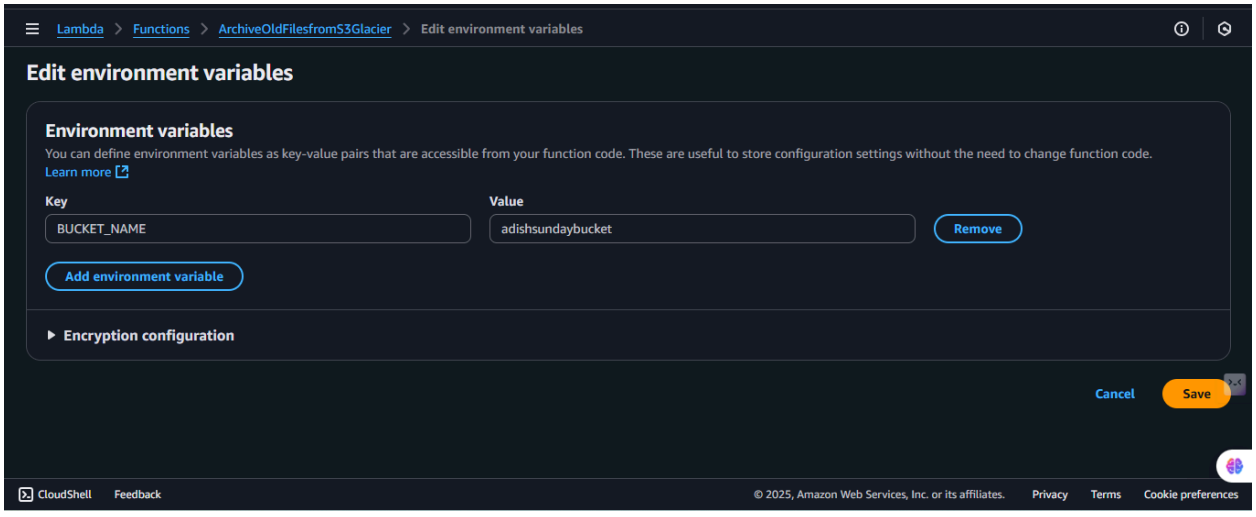
2. Lambda Function Code (Python 3.x)



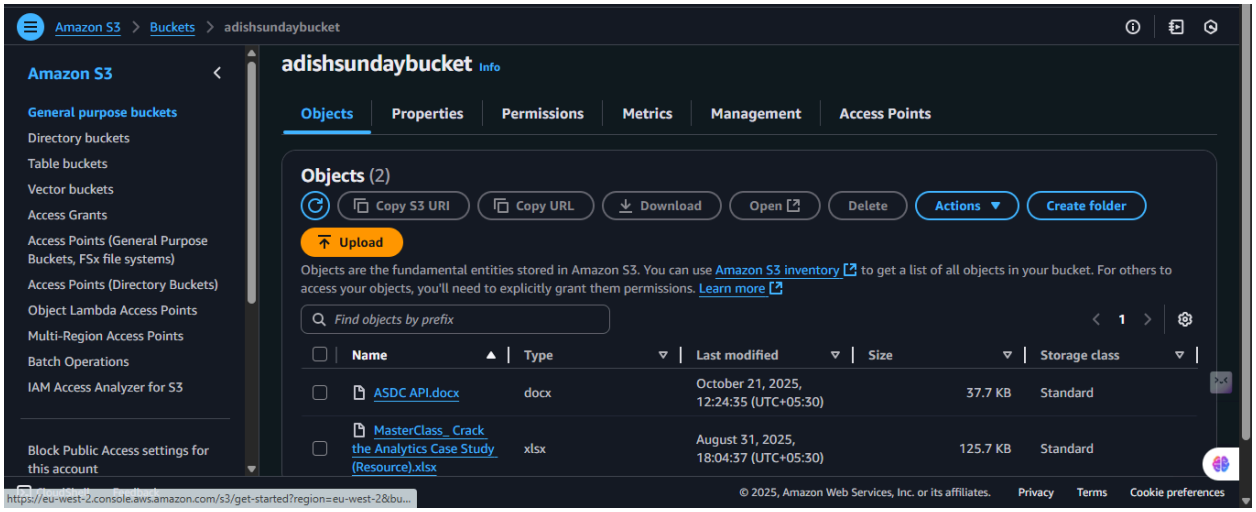


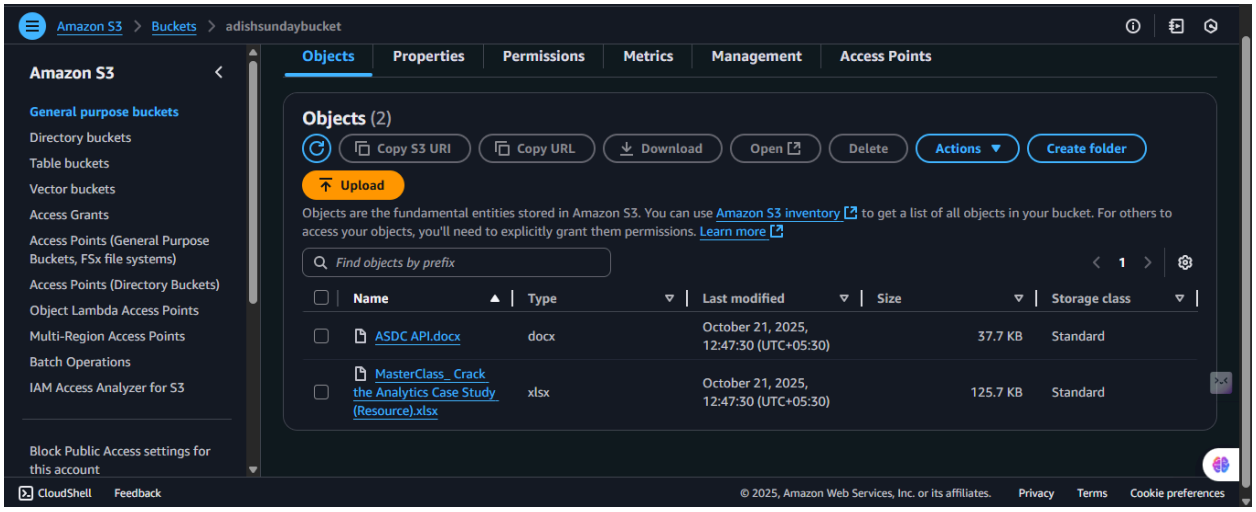






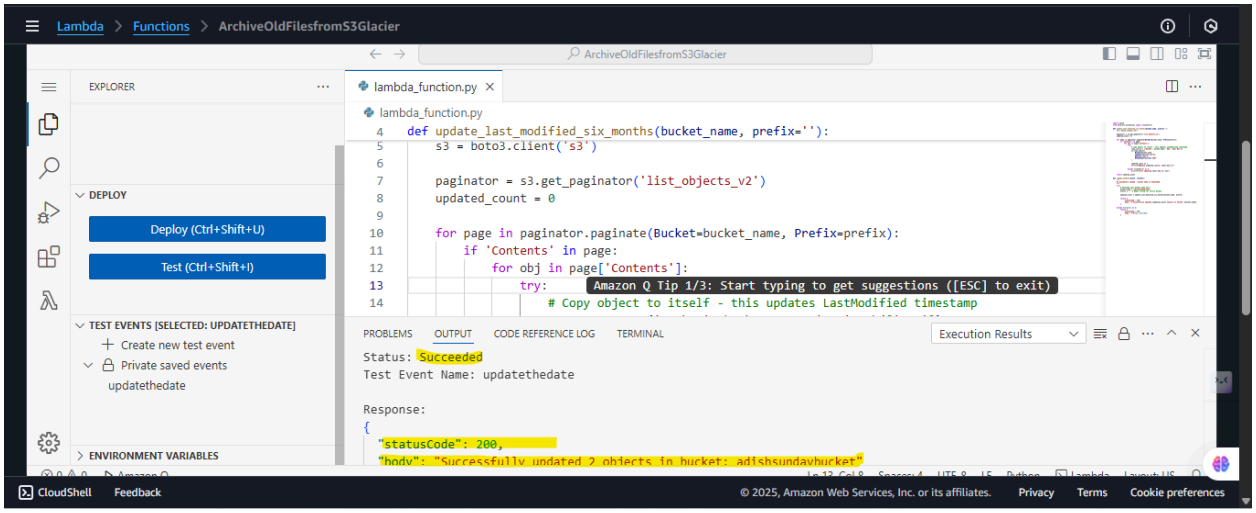
2 . S3 bucket list






```
import boto3
from botocore.exceptions import ClientError

def update_last_modified_six_months(bucket_name, prefix=''):
    s3 = boto3.client('s3')

    paginator = s3.get_paginator('list_objects_v2')
    updated_count = 0

    for page in paginator.paginate(Bucket=bucket_name, Prefix=prefix):
        if 'Contents' in page:
            for obj in page['Contents']:
                try:
                    # Copy object to itself - this updates LastModified
timestamp
```

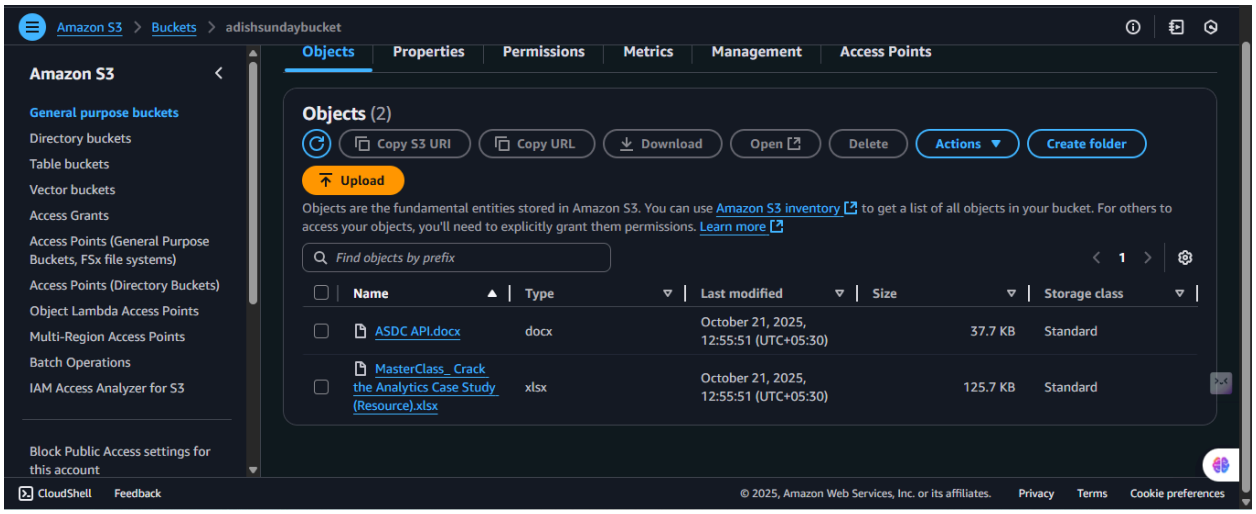
```
                    copy_source = {'Bucket': bucket_name, 'Key':
obj['Key']}
                    s3.copy_object(
                        Bucket=bucket_name,
                        CopySource=copy_source,
                        Key=obj['Key'],
                        MetadataDirective='COPY'
                    )

                    updated_count += 1
                    print(f"Updated {updated_count}: {obj['Key']}")

                except ClientError as e:
                    print(f"Error updating {obj['Key']}: {e}")

    return updated_count

def lambda_handler(event, context):
    """
    No parameters needed - bucket name is hardcoded
    """
    try:
        # Hardcode your bucket name here
        bucket_name = "adishsundaybucket"
        prefix = ""  # Empty string for entire bucket

        updated_count = update_last_modified_six_months(bucket_name,
prefix)

        return {
            'statusCode': 200,
            'body': f'Successfully updated {updated_count} objects in
bucket: {bucket_name}'
        }

    except Exception as e:
        return {
            'statusCode': 500,
            'body': f'Error: {str(e)}'
        }
```

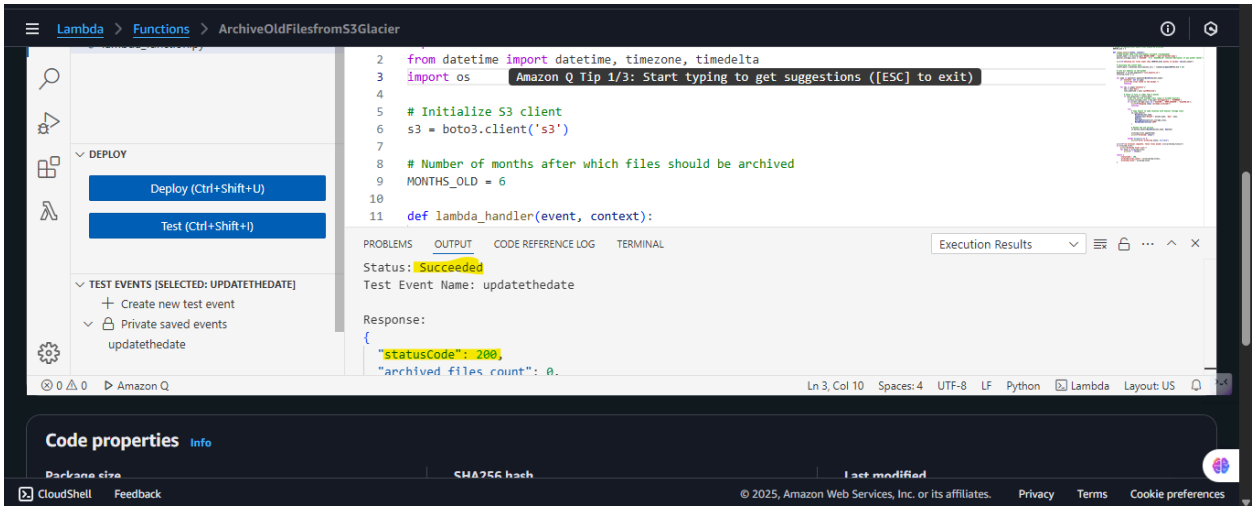


3. Lambda Environment Variables

Set the following environment variable in Lambda configuration:

Key	Value
`BUCKET_NAME`	your-s3-bucket-name






```
import boto3
from datetime import datetime, timezone, timedelta
import os

# Initialize S3 client
s3 = boto3.client('s3')

# Number of months after which files should be archived
MONTHS_OLD = 6

def lambda_handler(event, context):
    # Read bucket name from environment variable (recommended)
    bucket_name = os.environ.get('BUCKET_NAME', 'your-s3-bucket-name')
    glacier_storage_class = 'GLACIER'  # or 'GLACIER_IR' (Instant Retrieval) if you prefer faster retrieval

    print(f"Checking for files older than {MONTHS_OLD} months in bucket: {bucket_name}")

    # Calculate the cutoff date
    cutoff_date = datetime.now(timezone.utc) - timedelta(days=MONTHS_OLD * 30)

    # List all objects in the bucket
    paginator = s3.get_paginator('list_objects_v2')
    archived_files = []

    for page in paginator.paginate(Bucket=bucket_name):
```

```
        if 'Contents' not in page:
            print("No files found in the bucket.")
            continue

        for obj in page['Contents']:
            key = obj['Key']
            last_modified = obj['LastModified']

            # Check if file is older than 6 months
            if last_modified < cutoff_date:
                # Get the current storage class (skip if already Glacier)
                current_storage_class = obj.get('StorageClass',
'STANDARD')
                if current_storage_class in ['GLACIER', 'DEEP_ARCHIVE',
'GLACIER_IR']:
                    print(f"Skipping {key} (already archived)")
                    continue

                try:
                    # Copy object to same location with Glacier storage
class
                    s3.copy_object(
                        Bucket=bucket_name,
                        CopySource={'Bucket': bucket_name, 'Key': key},
                        Key=key,
                        StorageClass=glacier_storage_class,
                        MetadataDirective='COPY'
                    )

                    # Delete the old version
                    s3.delete_object(Bucket=bucket_name, Key=key)

                    archived_files.append(key)
                    print(f"Archived: {key}")

                except Exception as e:
                    print(f"Error archiving {key}: {str(e)}")

    print(f"\n✅ Archival complete. Total files moved:
{len(archived_files)}")
```

```
    if archived_files:
        print("Archived files list:")
        for file in archived_files:
            print(f" - {file}")

    return {
        'statusCode': 200,
        'archived_files_count': len(archived_files),
        'archived_files': archived_files
    }
```

4. IAM Role Policy

Attach the following policies:

- `AmazonS3FullAccess`
- `AWSLambdaBasicExecutionRole`