

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)

The screenshot shows the 'Create function' page in the AWS Lambda console. The 'Basic information' tab is active. Under 'Function name', the text 'ArchiveOldFilesfromS3toGlacier' is entered. The 'Runtime' is set to 'Python 3.13'. The 'Architecture' is set to 'x86_64'. The page includes a breadcrumb trail 'Lambda > Functions > Create function' and a footer with 'CloudShell', 'Feedback', and copyright information.

Create function [Info](#)

Choose one of the following options to create your function.

- ☒ **Author from scratch**
Start with a simple Hello World example.
- ☐ **Use a blueprint**
Build a Lambda application from sample code and configuration presets for common use cases.
- ☐ **Container image**
Select a container image to deploy for your function.

Basic information

Function name
Enter a name that describes the purpose of your function.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

The screenshot shows the 'Permissions' tab of the 'Create function' page. It details the 'Change default execution role' section, where 'Use an existing role' is selected, and 'LambdaS3GlacierRole' is chosen from the dropdown. The 'Additional configurations' section is partially visible at the bottom. The page includes a breadcrumb trail 'Lambda > Functions > Create function' and a footer with 'CloudShell', 'Feedback', and copyright information.

Permissions [Info](#)

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ Change default execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

- ☐ Create a new role with basic Lambda permissions
- ☒ Use an existing role
- ☐ Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

[View the LambdaS3GlacierRole role](#) on the IAM console.

► Additional configurations
Use additional configurations to set up networking, security, and governance for your function. These settings help secure and customize your Lambda function deployment.

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Identity and Access Management (IAM)

Search IAM

Dashboard

Access management

- User groups
- Users
- Roles**
- Policies
- Identity providers
- Account settings
- Root access management

Access reports

CloudShell Feedback

Permissions Trust relationships Tags Last Accessed Revoke sessions

Permissions policies (2) Info



You can attach up to 10 managed policies.

Filter by Type

Search

All types

< 1 >

<input type="checkbox"/>	Policy name	Type	Attached entities
<input type="checkbox"/>	 AmazonS3FullAccess	AWS managed	100
<input type="checkbox"/>	 AWSLambdaBasicExecutionRole	AWS managed	68

Permissions boundary (not set)

Generate policy based on CloudTrail events

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Lambda > **Functions** > ArchiveOldFilesfromS3Glacier

FUNCTION URL Info

Code Test Monitor **Configuration** Aliases Versions

General configuration Info

Description

-

Timeout

1 min 3 sec

Memory

128 MB

Ephemeral storage

512 MB

SnapStart Info

None

Edit

CloudShell Feedback

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Lambda > **Functions** > **ArchiveOldFilesfromS3Glacier** > Edit environment variables

Edit environment variables

Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#)

Key	Value	
BUCKET_NAME	adishundaybucket	Remove

Add environment variable

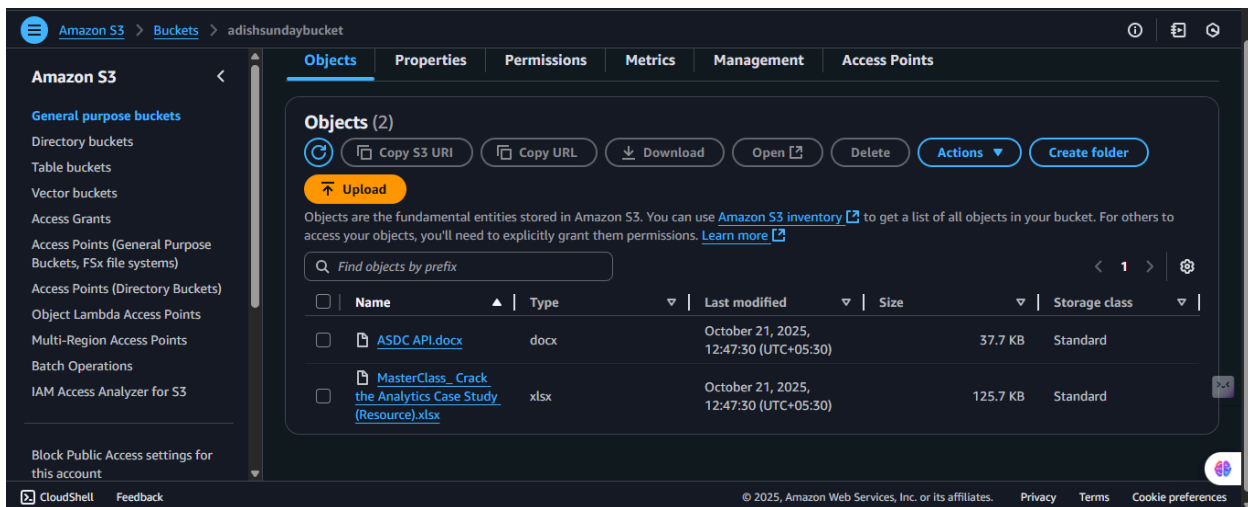
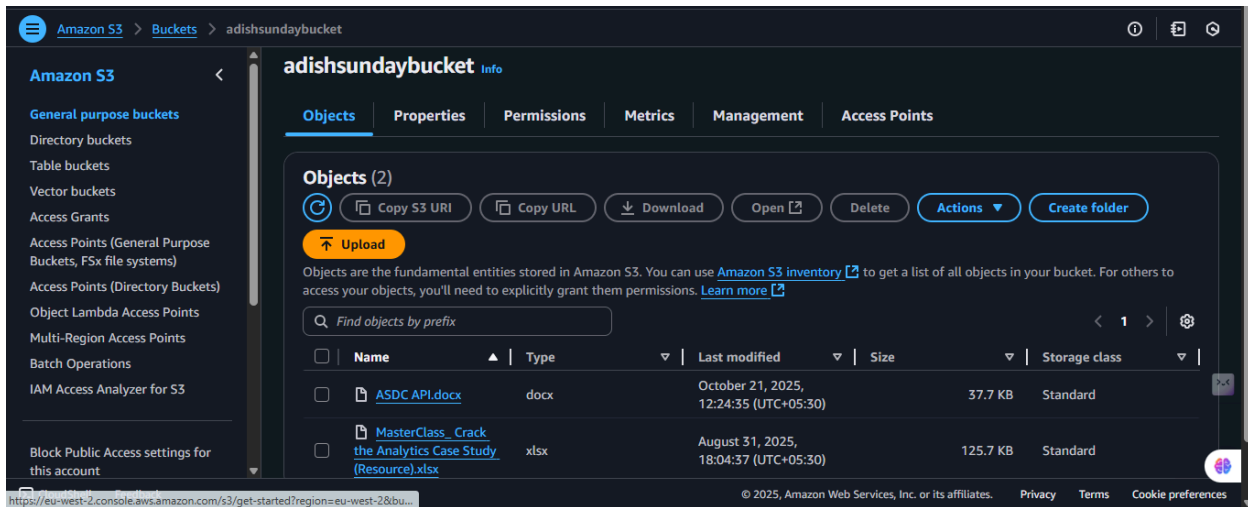
Encryption configuration

Cancel **Save**

CloudShell Feedback

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

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

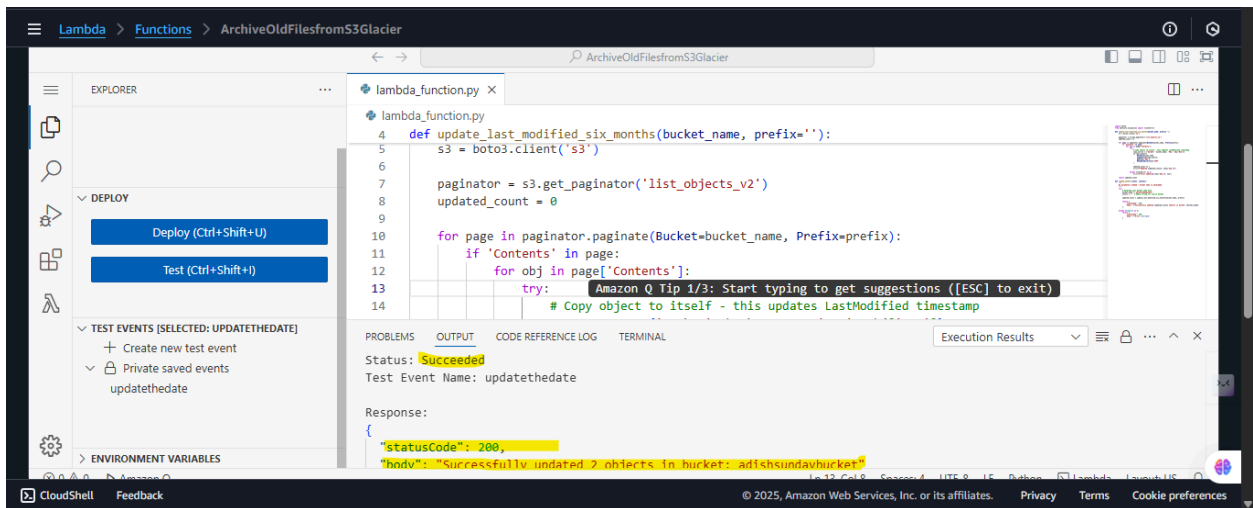
def lambda_handler(event, context):
    """
    No parameters needed - bucket name is hardcoded
    """
    try:
        # Hardcode your bucket name here
        bucket_name = "adishsundaybucket"
        prefix = "" # Empty string for entire bucket

        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)}'
        }

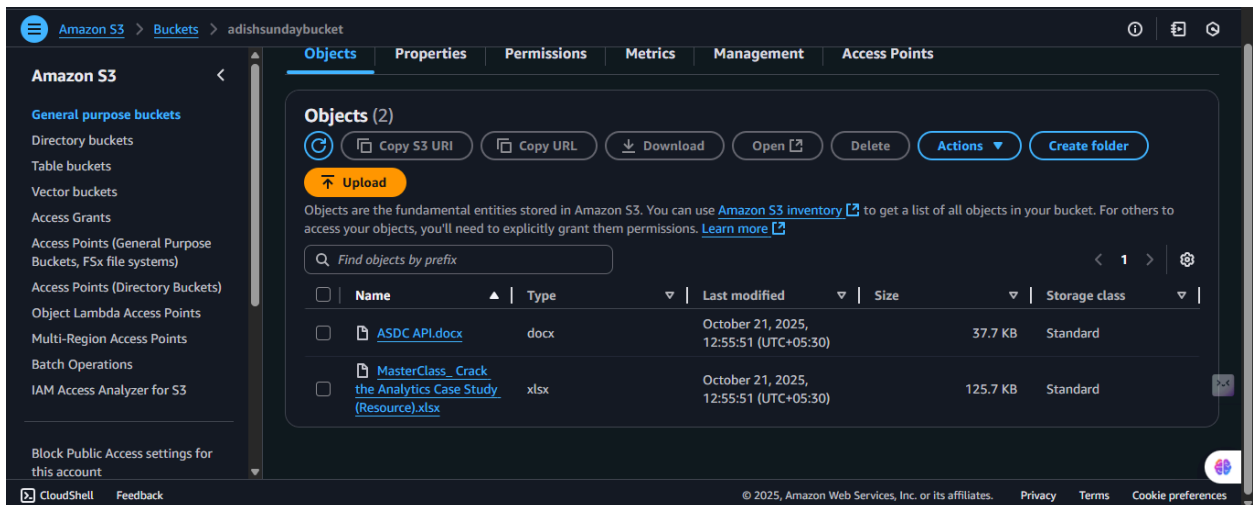
```

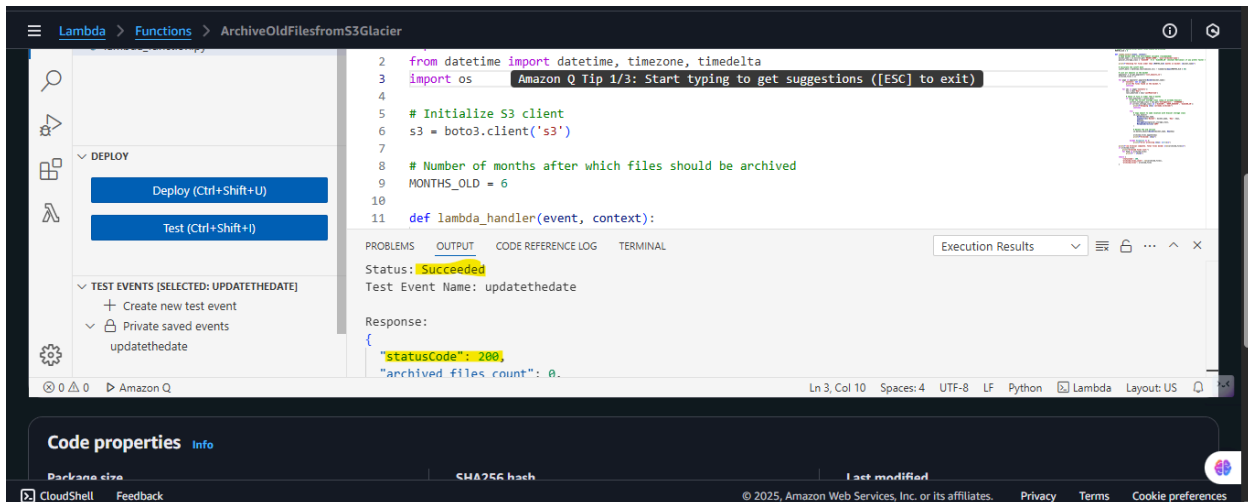


3. Lambda Environment Variables

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**