

## 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' wizard in the AWS Lambda console. The top navigation bar includes 'Lambda > Functions > Create function'. The main section is titled 'Create function' with an 'Info' link. It says 'Choose one of the following options to create your function.' Three options are shown in boxes: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. Below this is the 'Basic information' section. Under 'Function name', the input field contains 'ArchiveOldFilesfromS3toGlacier'. A note below it states: '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 (\_).'. Under 'Runtime', a dropdown menu is set to 'Python 3.13'. Under 'Architecture', there's a note about instruction sets and a dropdown menu currently at 'Python'. At the bottom of this section are links for 'CloudShell' and 'Feedback', along with copyright and privacy information.

The screenshot shows the 'Create function' wizard in the AWS Lambda console, specifically the 'Permissions' step. The top navigation bar includes 'Lambda > Functions > Create function'. The main section is titled 'Permissions' with an 'Info' link. It says '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.' Below this is the 'Change default execution role' section. Under 'Execution role', it says 'Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#)'. Three options are listed: 'Create a new role with basic Lambda permissions' (unchecked), 'Use an existing role' (selected), and 'Create a new role from AWS policy templates' (unchecked). Under 'Existing role', a dropdown menu is set to 'LambdaS3GlacierRole'. A note below it says 'View the [LambdaS3GlacierRole role](#) on the IAM console.' At the bottom of this section is the 'Additional configurations' section, which includes a note about setting up networking, security, and governance. At the very bottom are links for 'CloudShell' and 'Feedback', along with copyright and privacy information.

**IAM > Roles > LambdaS3GlacierRole**

**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

Policy name	Type	Attached entities
AmazonS3FullAccess	AWS managed	100
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: -

Code Test Monitor Configuration Aliases Versions

**General configuration**

Description: -    Memory: 128 MB    Ephemeral storage: 512 MB

Triggers    Permissions    Destinations    Function URL    Environment variables    Tags    VPC

CloudShell Feedback

**Configuration**

**General configuration**

Description: -    Memory: 128 MB    Ephemeral storage: 512 MB

Timeout: 1 min 3 sec    SnapStart: None

Edit

© 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	adishsundaybucket

Add environment variable Remove

**Encryption configuration**

Cancel Save

CloudShell Feedback

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

## 2 . S3 bucket list

Amazon S3 > Buckets > adishsundaybucket

adishsundaybucket info

Objects (2)

Name	Type	Last modified	Size	Storage class
ASDC API.docx	docx	October 21, 2025, 12:24:35 (UTC+05:30)	37.7 KB	Standard
MasterClass_Crack_the Analytics Case Study_(Resource).xlsx	xlsx	August 31, 2025, 18:04:37 (UTC+05:30)	125.7 KB	Standard

Find objects by prefix

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

Amazon S3 > Buckets > adishsundaybucket

adishsundaybucket info

Objects (2)

Name	Type	Last modified	Size	Storage class
ASDC API.docx	docx	October 21, 2025, 12:47:30 (UTC+05:30)	37.7 KB	Standard
MasterClass_Crack_the Analytics Case Study_(Resource).xlsx	xlsx	October 21, 2025, 12:47:30 (UTC+05:30)	125.7 KB	Standard

Find objects by prefix

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

```
import boto3
from botocore.exceptions import ClientError

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

    paginator = s3.getPaginator('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)}'
        }

```

```

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:
                    # Amazon Q Tip 1/3: Start typing to get suggestions ([ESC] to exit)
                    # Copy object to itself - this updates LastModified timestamp
                    s3.put_object(Bucket=bucket_name, Key=obj['Key'], Body='')
                    updated_count += 1
                except Exception as e:
                    print(f"Error updating object {obj['Key']}: {e}")

```

### 3. Lambda Environment Variables

Set the following environment variable in Lambda configuration:

Key	Value
BUCKET_NAME	your-s3-bucket-name
ME	me

Name	Type	Last modified	Size	Storage class
ASDC API.docx	docx	October 21, 2025, 12:55:51 (UTC+05:30)	37.7 KB	Standard
MasterClass_Crack the Analytics Case Study (Resource).xlsx	xlsx	October 21, 2025, 12:55:51 (UTC+05:30)	125.7 KB	Standard

```

    2: from datetime import datetime, timezone, timedelta
  3: import os [Amazon Q Tip 1/3: Start typing to get suggestions ([ESC] to exit)]
  4:
  5: # Initialize S3 client
  6: s3 = boto3.client('s3')
  7:
  8: # Number of months after which files should be archived
  9: MONTHS_OLD = 6
10:
11: def lambda_handler(event, context):

```

PROBLEMS OUTPUT CODE REFERENCE LOG TERMINAL

Status: **Succeeded**

Test Event Name: updatethedate

Response:

```
{
  "statusCode": 200,
  "archived_files_count": 0.
```

Ln 3, Col 10 Spaces:4 UTF-8 LF Python Lambda Layout:US

```

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**