Secure File Sharing using Amazon S3 and CloudFront

1. Project Overview

This project implements a **secure file-sharing system** using **Amazon S3** and **Amazon CloudFront**. Files are stored securely in a **private S3 bucket** and delivered to users through **CloudFront** using **signed URLs**. Direct access to S3 objects is blocked, ensuring controlled and secure file distribution.

---

2. AWS Services Used

Service	Purpose
Amazon S3	Secure object storage
Amazon CloudFront	Content Delivery Network (CDN)
CloudFront Origin Access Control (OAC)	Secure access from CloudFront to S3
CloudFront Key Groups	Validation of signed URLs
AWS IAM	Access control and permissions

---

3. Architecture Summary

- Users **cannot access S3 directly**
- CloudFront acts as the **only access layer**
- Signed URLs are required to download files
- Origin Access Control ensures **only CloudFront** can read S3 objects

4. Step-by-Step Implementation (Console-Based)

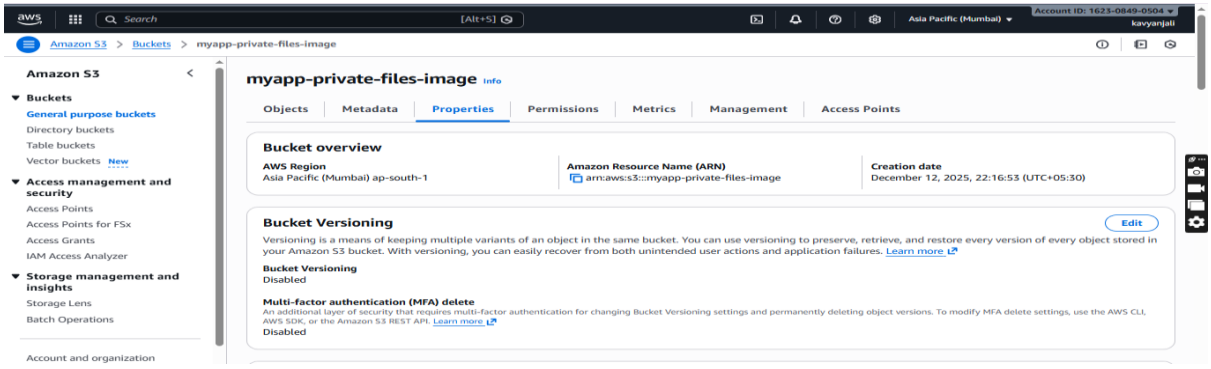
STEP 1 — Create a Private S3 Bucket

Objective: Store files securely with no public access.

Actions performed:

- Created an S3 bucket with a unique name.
- Disabled all public access.
- Enabled bucket ownership to prevent ACL misuse.

Result:S3 bucket is private and inaccessible from the internet.



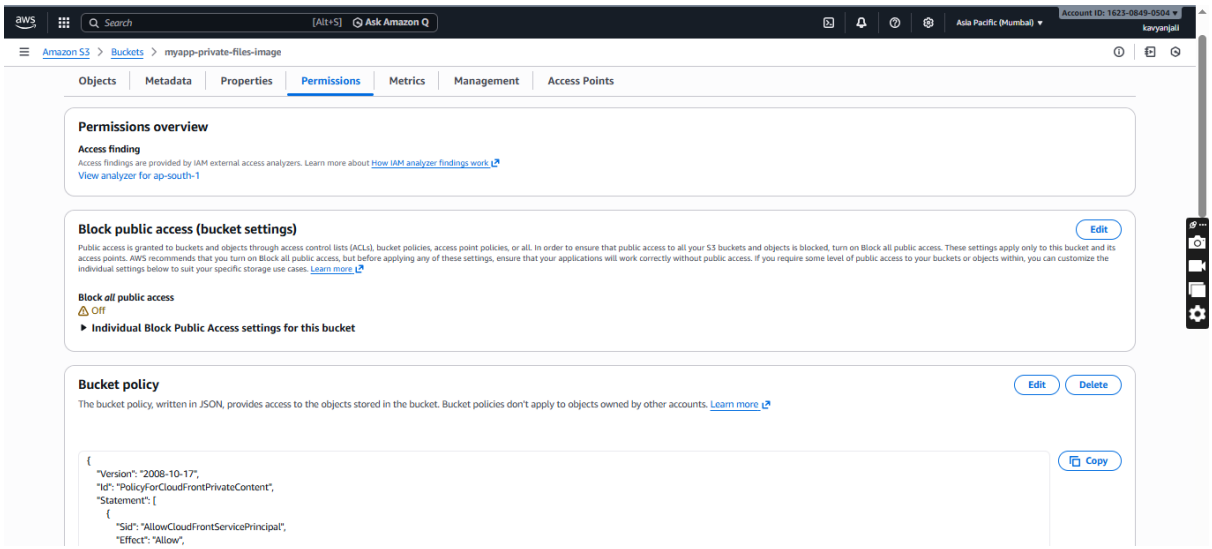
STEP 2 — Upload Policy Preparation (Access Control)

Objective: Ensure uploads and downloads are controlled.

Actions performed:

- Bucket permissions restricted.
- No public read or write access allowed.

Result: Only authorized AWS services can interact with the bucket.

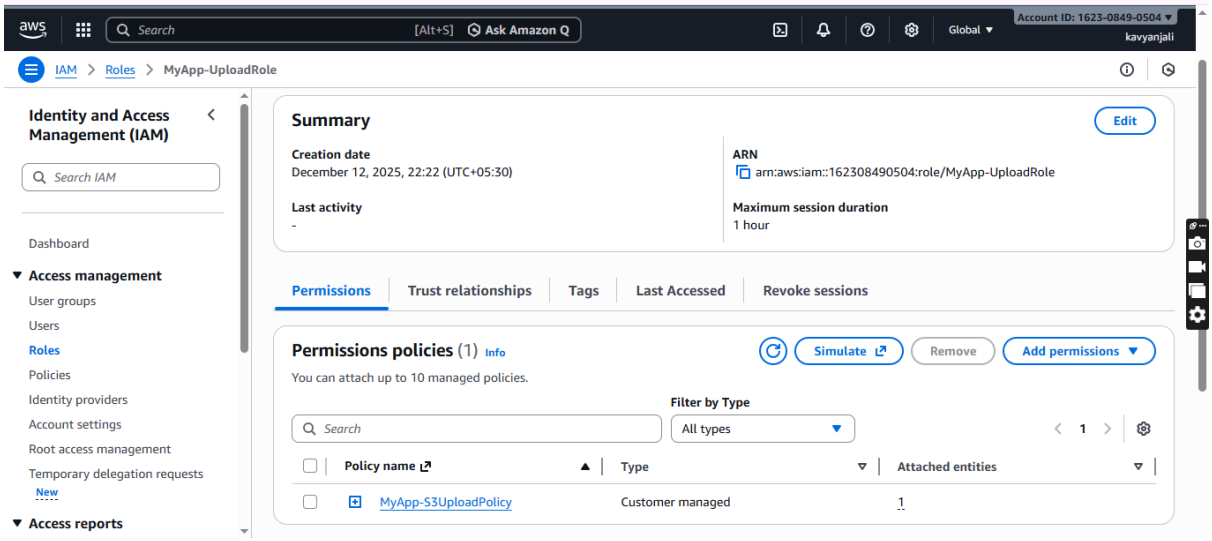


2.1 — Create an IAM policy & role for server uploads (console)

1. Console → Services → **IAM** → left nav → **Policies** → **Create policy**.
2. Choose the **JSON** tab and paste the policy (replace bucket name):

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": ["s3:PutObject", "s3:PutObjectAcl"],
   "Resource": "arn:aws:s3:::myapp-private-files-image"
  }
 ]
}
```

- Click **Next** → **Tags** (optional) → **Next** → **Review policy**.
- **Name:** MyApp-S3UploadPolicy → **Create policy**.
- Now create a role: IAM → **Roles** → **Create role**.
- **Trusted entity type:** Choose appropriate (e.g., **AWS service** → EC2 / Lambda /
- Attach the policy MyApp-S3UploadPolicy you created.
- **Name:** MyApp-UploadRole → **Create role**.



STEP 3 — Create CloudFront Origin Access Control (OAC)

Objective: Allow CloudFront to securely access private S3 objects.

Actions performed:

- Created an Origin Access Control (OAC) in CloudFront.
- Selected **Amazon S3** as origin type.
- Enabled request signing.

Result: CloudFront can securely fetch objects from S3 using signed requests.

3.1 Create an Origin Access Control (OAC) and update S3 bucket policy to allow Cloud Front only

Important: AWS recommends using an **Origin Access Control (OAC)**. We will create an OAC and let Cloud Front sign requests to S3.

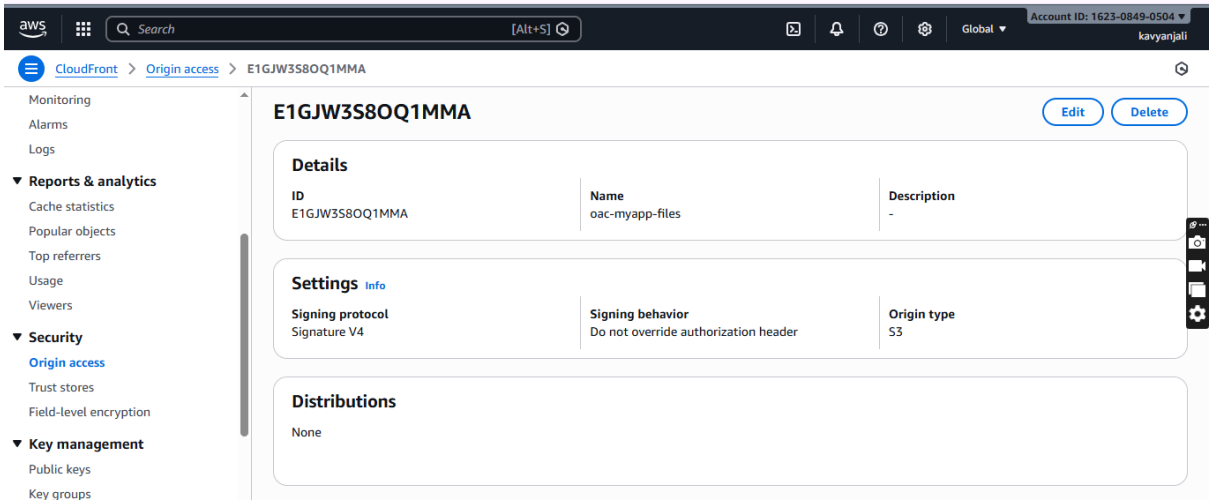
Create OAC in Console

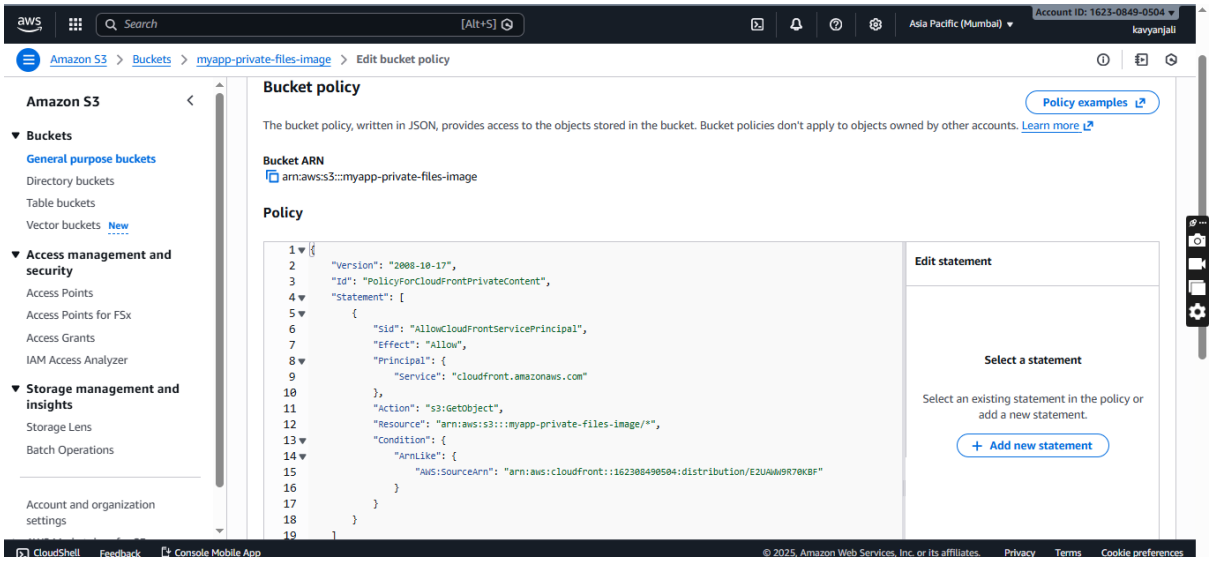
1. Console → **Services** → **CloudFront** → left nav → **Origin access** (or **Origin access control**).
2. Click **Create control setting** (or **Create origin access control**).
    - Name: oac-myapp-files
    - Signing protocol: **SigV4** (recommended)
    - Description: optional
3. Create.

Update S3 bucket policy to allow CloudFront OAC only

1. Console → **Services** → **S3** → open myapp-private-files-image→ **Permissions** → **Bucket policy**.
2. Add policy that allows the CloudFront OAC principal access to s3:GetObject. The CloudFront console provides the exact statement when you attach the OAC to the distribution. If you prefer, you can use the CloudFront console when creating the distribution and it will update bucket permissions for you.

Why: This ensures S3 objects are ac





STEP 4 — Create CloudFront Public Key

Objective: Enable signed URL validation.

Actions performed:

- Created a CloudFront public key.
- Uploaded an RSA public key (PEM format).
- Associated the public key with CloudFront.

Result: CloudFront can verify signed URLs generated by the private key.

4.1 Create a CloudFront key pair (Trusted Key Group) — public key + key group

To use **CloudFront signed URLs**, create a Key Group that holds the public key(s) CloudFront will trust. You keep the private key offline (or in a secure KMS/secret store) to sign URLs.

Create Public Key in CloudFront

1. Console → **Services** → **CloudFront** → left nav → **Public keys**.
2. Click **Create public key**.
    - Name: pk-myapp-signer

- Public key format: paste the RSA public key (PEM).
- (If you don't have a key pair locally: create one with OpenSSL: openssl genrsa -out private.pem 2048 and openssl rsa -in private.pem -pubout -out public.pem. Keep private.pem safe.)
- Click **Create public key**.

-----BEGIN PUBLIC KEY-----

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqd9ucSKQhl93hD9YSqAh

M8Th2pcw4M70P7+5x13RntmEX0jF+3JOMWRXIVCE1HqFw0AUEqbywYYkoBtelZP6

oYvdKqYQQ0CWWm5iL/KOhrzDZeOzv19jpvd9MuX0rxboIA46eg7HQe+14oAWjdmk

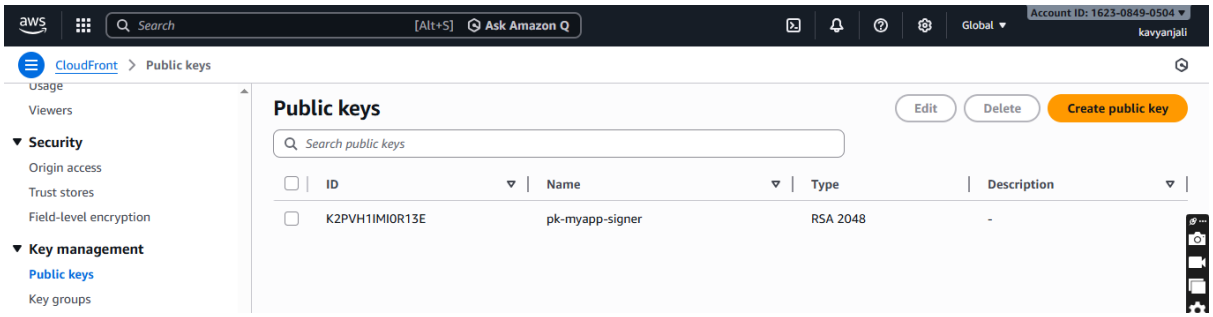
IfSjM49HOGu3VQCS4PD4dD/zztmtryZffpKP4T6MceL/AUv81JCEVkEFHzo+uMNZ

hk9AZPsSgB64VkF57T9BLDtgiT0L/NV9UVTJjFVj178VF1z9g09FPQIV21guzNoz

dVPfzkKfGRZNz+Os1jqM7Oso4tszgfMd/V6o2MjX2yjYM5lGytSDxnvLzL8+fwYx

owIDAQAB

-----END PUBLIC KEY-----



4.2 Create CloudFront Key Group

Objective: Group trusted public keys.

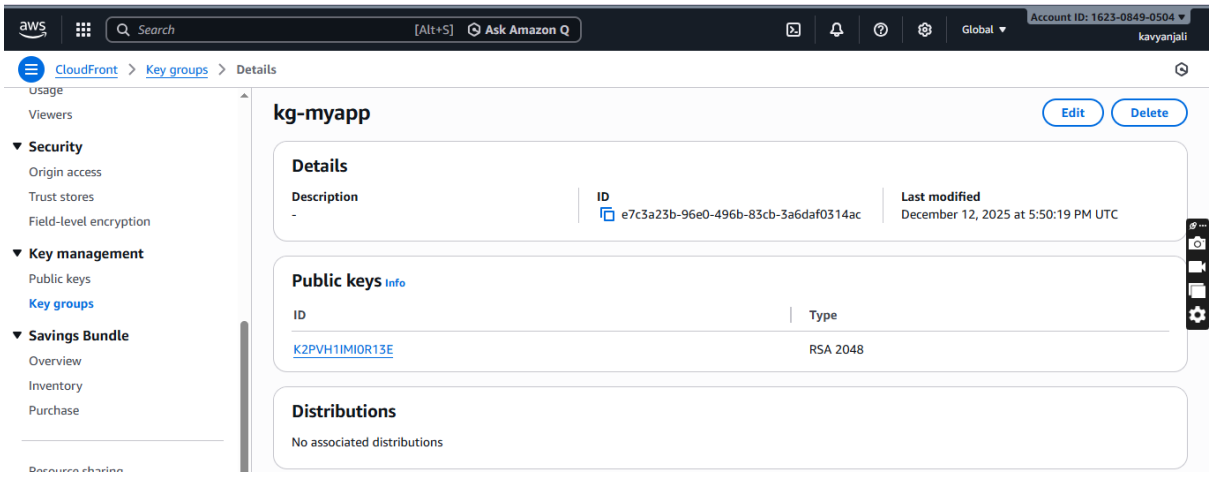
Actions performed:

- Created a Key Group.
- Added the CloudFront public key to the group.
- **Result:**Key Group is ready to validate signed URLs.

Create Key Group

1. Console → **CloudFront** → left nav → **Key groups**.
2. Click **Create key group**.
   - Name: kg-myapp
   - Add the public key you just created.
3. Create.

Why: CloudFront verifies signed URLs using the public key in the key group. Use Key Groups (recommended) rather than account key pairs.



STEP 5 — Create CloudFront Distribution

Objective: Serve private S3 content through CloudFront.

Actions performed:

- Created a CloudFront distribution.
- Selected S3 bucket as origin.
- Attached **Origin Access Control (OAC)**.
- Allowed CloudFront to update the S3 bucket policy automatically.

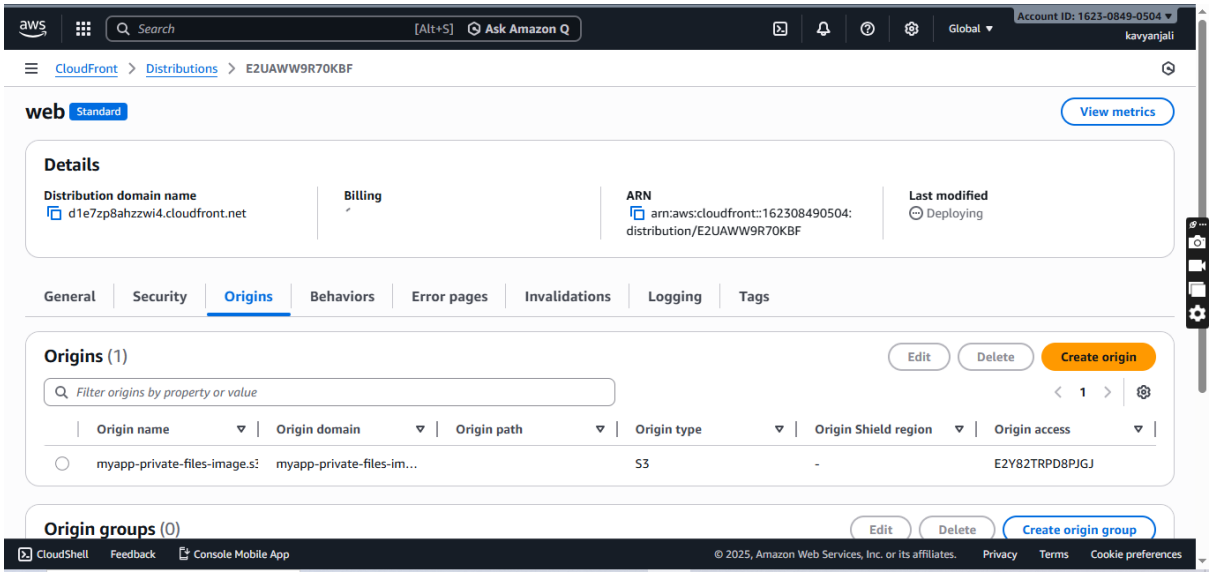
Result: CloudFront became the only gateway to the S3 bucket.

5.1 Create a CloudFront distribution pointing to your S3 bucket (using new OAC) and associate the Key Group

Create Distribution

1. Console → **CloudFront → Create distribution → Web (or Distributions → Create distribution)**.
2. **Origin settings**:
   - Origin domain: choose your S3 bucket from dropdown (it'll appear as myapp-private-files-<suffix>.s3.<region>.amazonaws.com).
   - Origin access: select **Origin access control (recommended)** → pick oac-myapp-files. If console prompts to update bucket permissions, allow it (or paste the bucket policy manually).
3. **Default Cache Behavior**:
   - Viewer protocol policy: **Redirect HTTP to HTTPS** (recommended).
   - Allowed HTTP methods: GET, HEAD (and OPTIONS if needed).
4. **Restrict Viewer Access** (Private content):
   - In **Cache behavior** look for **Signed URL / Signed cookie** options. (Exact label sometimes shown as **Require signed URLs** or you will later choose how to generate signed URLs when creating links.)
5. **Add Trusted Key Group**:
   - In the distribution's settings/behaviour, under **Trusted key groups** or **Trusted signers** select the Key Group kg-myapp you created. This tells CloudFront to accept signed URLs/cookies signed by the private key that corresponds to the public key in that key group.
6. Finish creating the distribution. Note the **Domain Name** (e.g., d1234abcd.cloudfront.net).

Why: CloudFront will now be the only way users can fetch objects from S3 (because S3 allows CloudFront OAC) and CloudFront will enforce signed URL validation using the trusted key group.



STEP 5 — Verify S3 Bucket Policy

Objective: Ensure S3 only allows CloudFront access.

Actions performed:

- Verified auto-generated bucket policy.
- Confirmed cloudfront.amazonaws.com is the only principal allowed.
- Verified access restricted by CloudFront distribution ID.

Result: Direct S3 access is blocked.

STEP 5.1 — Verify S3 Bucket Policy (CloudFront OAC)

This ensures **only CloudFront** can read objects.

1 Open S3

1. AWS Console → **Services**
2. Click **S3**
3. Open your bucket
   Example: myapp-private-files-image

2 Open Bucket Policy

1. Go to **Permissions** tab
2. Scroll to **Bucket policy**
3. Click **Edit**

3 Confirm Policy Exists

You should already see a policy added automatically by CloudFront.

✔ Confirm:

- **Principal** contains cloudfront.amazonaws.com
- **Action** includes s3:GetObject

- **Condition** includes your **CloudFront distribution ID**
- Click **Save changes** only if CloudFront added it automatically

STEP 6 — Upload Test Object via Console

Objective: Test secure file storage.

Actions performed:

Uploaded a test file using the S3 console.

Result: File successfully stored in the private bucket.

STEP 6.1 — Upload Test Object (S3 Console)

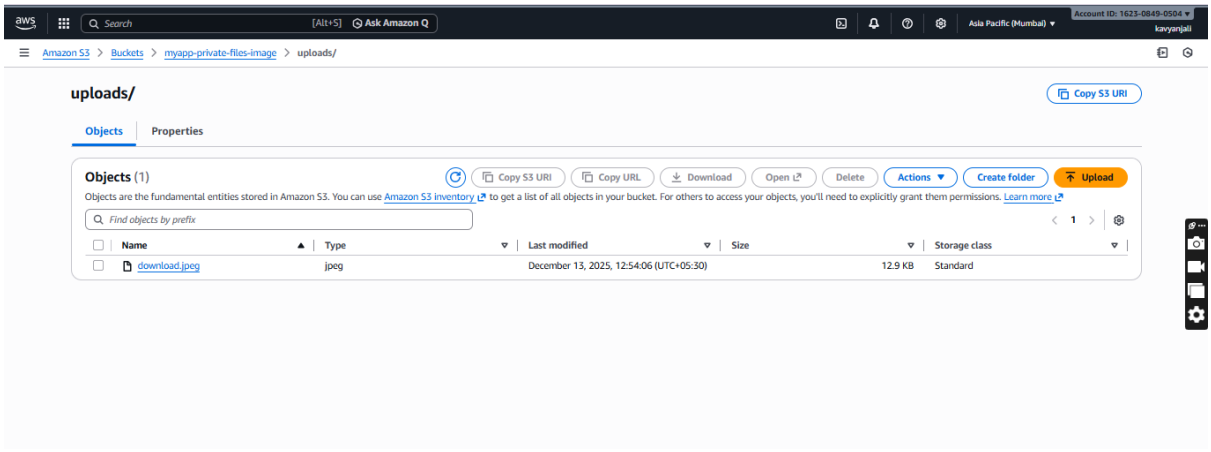
1 Go to Objects

1. S3 → your bucket
2. Click **Objects**
3. Click **Upload**

2 Upload File

1. Click **Add files**
2. Select a test file
   Example: test.pdf or image.jpg
3. Click **Upload**

File appears in the bucket



STEP 7 — Test Direct S3 Access

Objective: Confirm S3 security.

Actions performed: Accessed object using S3 object URL.

Observed result:
❌Access Denied

Conclusion:
S3 is correctly secured.

STEP 7.1 — Test Direct S3 Access (Should FAIL)

1 Copy S3 Object URL

1. Click the uploaded file
2. Copy **Object URL**

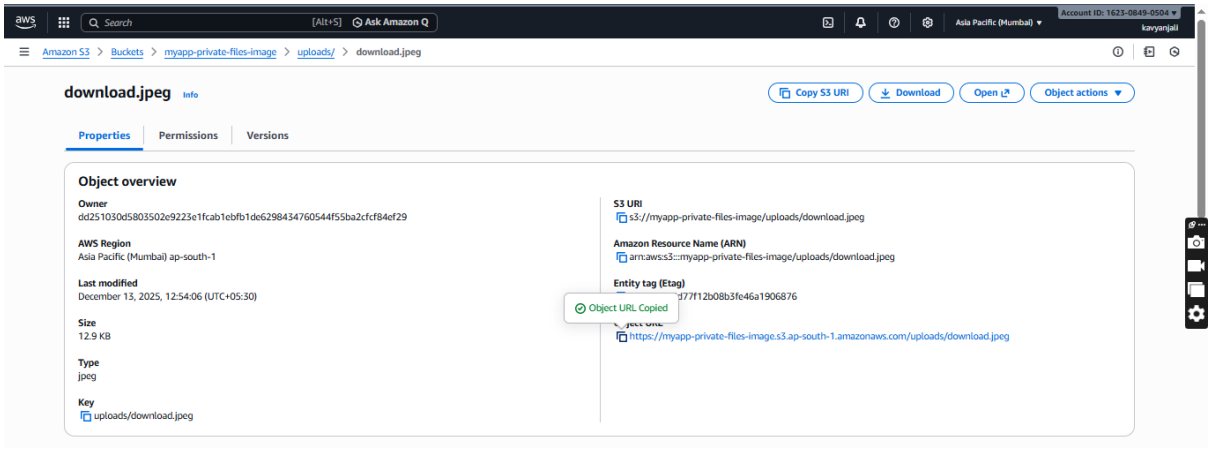
2 Open in Browser

1. Paste URL in browser
2. Press Enter

Expected Result: AccessDenied

This confirms:

- Bucket is private
- S3 blocks public access



STEP 8 — Test CloudFront Access without Signed URL

Objective: Validate CloudFront restrictions.

Actions performed: Accessed file using CloudFront domain without a signed URL.

Observed result:
❌403 Forbidden

Conclusion:
CloudFront enforces signed URL validation.

STEP 8.1 — Test CloudFront Access (Signed URL Required)

1 Copy CloudFront Domain

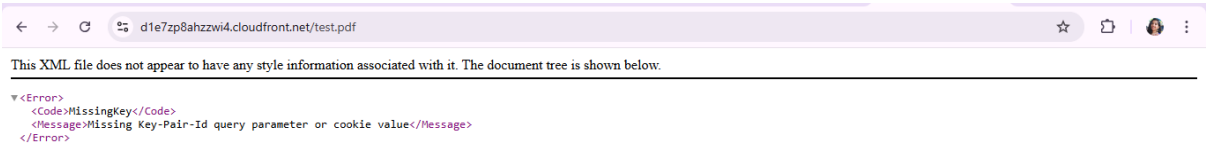
1. CloudFront → **Distributions**
2. Copy **Domain name**
   Example: d1e7zp8ahzzwi4.cloudfront.net

2 Try Without Signed URL

1. Open browser
2. Enter:
3. https://d1e7zp8ahzzwi4.cloudfront.net/test.pdf

Expected Result: 403 Forbidden

This confirms: CloudFront is enforcing **signed URLs**



STEP 9— Security Validation Summary

✔ Private S3 bucket
✔ CloudFront-only access
✔ Signed URLs required
✔ OAC correctly configured

STEP 10 — Lambda@Edge Integration

Objective: Add authentication or request validation at the CDN edge.

Actions Performed:

- Created Lambda function in **us-east-1 (N. Virginia)**
- Published a Lambda version (required for Lambda@Edge)
- Associated Lambda function with CloudFront distribution
- Attached function to **Viewer Request** or **Viewer Response** event

Purpose:

- Validate requests
- Issue signed cookies
- Apply custom access logic before content delivery

Result: Lambda@Edge executes globally at CloudFront edge locations before serving content.

STEP 10.1 — Switch Region

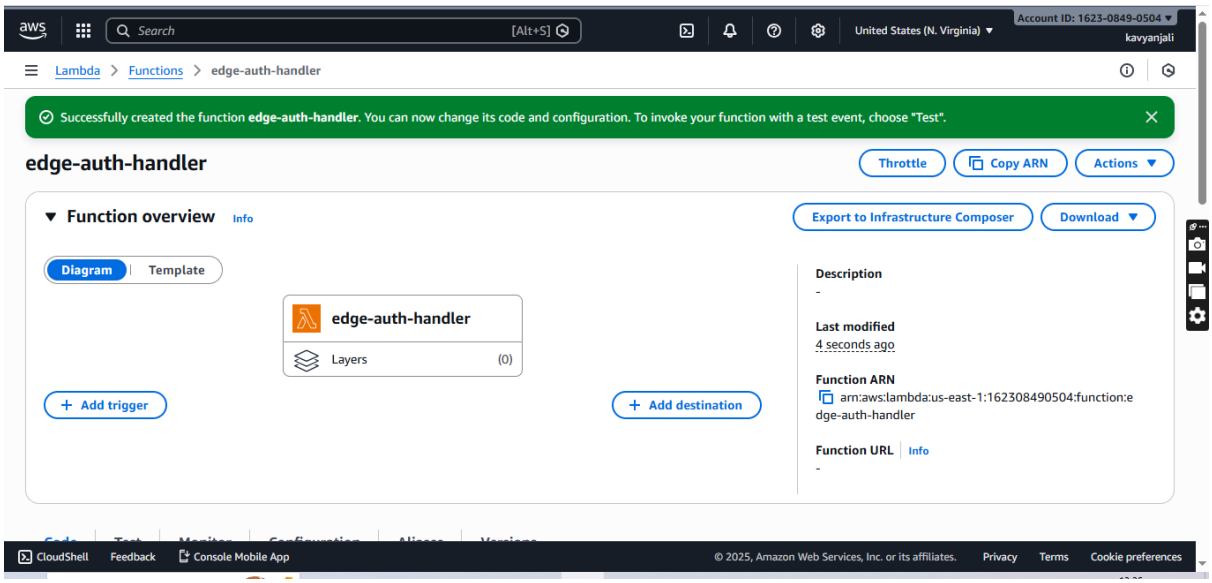
1. AWS Console (top-right region selector)
2. Select **US East (N. Virginia) – us-east-1**

STEP 10.2 — Create Lambda Function

1. AWS Console → **Services**
2. Click **Lambda**
3. Click **Create function**

Create function page

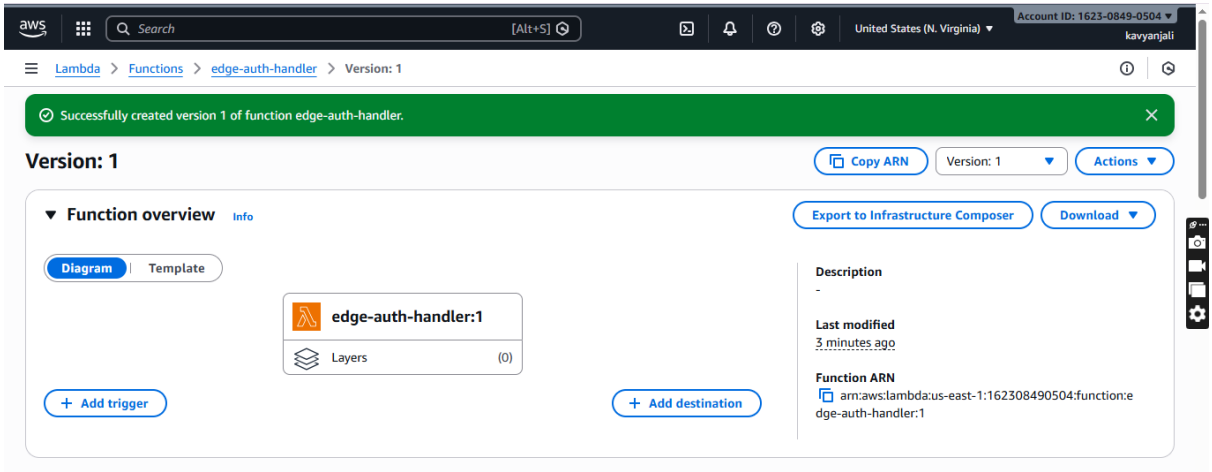
1. Select **Author from scratch**
2. **Function name**: edge-auth-handler or arn id
3. **Runtime**: Python
4. **Architecture**: x86_64
5. **Permissions** : Select **Create a new role with basic Lambda permissions**
6. Click **Create function**



STEP 10.3 — Publish a Lambda Version

Lambda@Edge **requires a published version**.

1. Inside the Lambda function page
2. Click **Actions**
3. Click **Publish new version**
4. Version description: optional
5. Click **Publish** **Note** the **Version number** (example: 1)



STEP 10.4 — Attach Lambda@Edge to CloudFront

1. AWS Console → **Services**
2. Click **CloudFront**
3. Open **Distributions**
4. Click your distribution ID

STEP 10.5 — Edit Default Cache Behavior

1. Go to **Behaviors** tab
2. Select the **Default behavior**
3. Click **Edit**

STEP 10.6 — Add Lambda@Edge Association

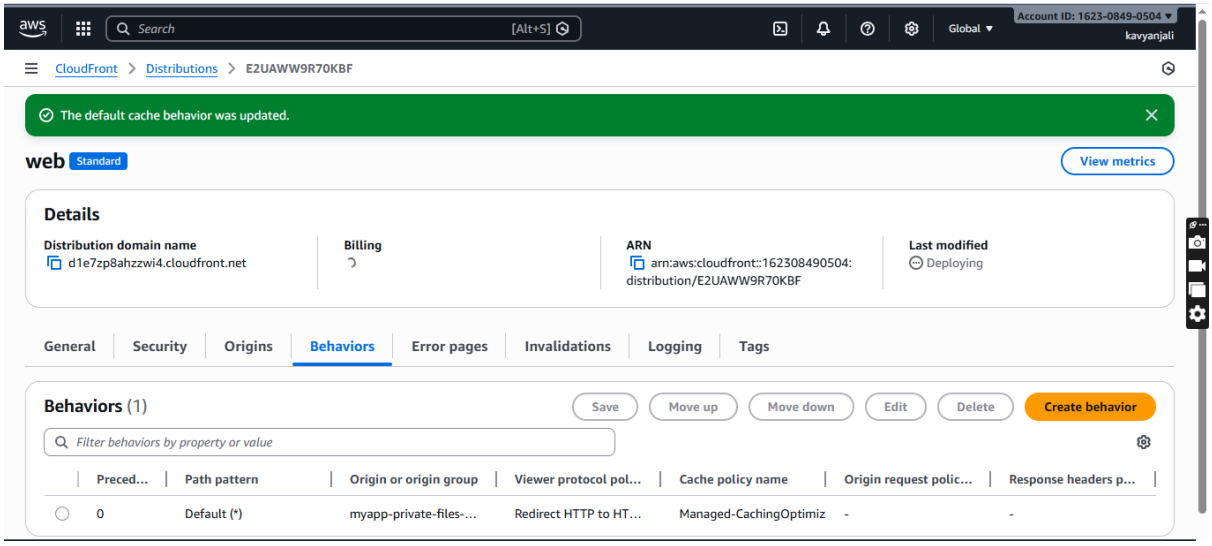
Scroll to **Function associations**:

1. Click **Add association**
2. **Event type**
   - Choose **Viewer request**
     (or **Viewer response** if setting cookies)
3. **Lambda function**
   - Select your function: edge-auth-handler
4. **Lambda version**
   - Select the **published version number** (NOT $LATEST)
5. Click **Save changes**

STEP 10.7 — Wait for Deployment

1. CloudFront will show **Deploying**
2. Wait until status becomes **Deployed**

✔ Lambda@Edge is now replicated globally



11. Final Outcome

The project successfully delivers a **secure, scalable, and controlled file-sharing solution** using AWS-managed services. Content is protected end-to-end with strong access controls.

12. Skills Demonstrated

- Secure S3 bucket configuration
- CloudFront CDN setup
- Origin Access Control (OAC)
- Signed URL enforcement
- Lambda@Edge integration
- IAM security best practices

13. Conclusion

This solution ensures:

- Zero public exposure of files
- Secure global content delivery
- Fine-grained access control
- Enterprise-grade security

It is suitable for **real-world production use cases** such as document sharing, media distribution, and secure downloads.