AWS S3

Why use Amazon S3? — Full list (every point included)

When comparing **EBS, EFS, and S3**, here's a complete, point-by-point explanation of why we use **S3** for object storage.

Key reasons

1. **Limited storage in EBS.**

   EBS volumes are fixed-size — you pay for the full volume even if you don't use it.

2. **EFS is expensive.**

   Example (approx):

   - EFS → 1 GB ≈ ₹30
   - S3 → 1 GB ≈ ₹2

3. **Hard to manage / track multiple EBS volumes.**

   When you store many files across multiple EBS volumes it becomes hard to track. S3 centralizes objects in a bucket and is easier to manage.

4. **EFS doesn't provide direct HTTP URLs like S3.**

   S3 gives object endpoints / URLs you can open in a browser.

5. **Different primary use-cases.**

   - EFS → application data (shared filesystem, source code)
   - S3 → object storage (files, images, backups, media, static websites)

6. **Unlimited storage.**

   S3 scales automatically — you pay only for what you use.

7. **Common/shared storage.**

   Multiple applications or users can access the same bucket.

8. **Redundant storage across AZs.**

   Objects are stored redundantly across 3 or more Availability Zones.

9. **Very high availability.**

   Availability is very high (e.g., ~99.999% or better depending on class).

10. **Very high durability.**

    “11 nines” durability — 99.999999999% — designed to prevent data corruption.

11. **Data in S3 is encrypted by default.**

    This helps keep data secure at rest.

12. **Fine-grained permissions.**

    Unlike EBS folders, S3 allows policies at object-level (per file) for precise access control.

13. **Protection against accidental deletes.**

    Use Versioning and Object Lock (or bucket policies) to prevent accidental deletion.

Short summary

- Unlimited storage
- Common shared storage
- 11 nines → durability
- Very high availability (~99.999%)
- Low cost — pay only for what you use

S3 object basics

- **Bucket name must be globally unique.**
- S3 stores **objects**. Each object contains:
  1. File
  2. Metadata
  3. Key → the “path” of the file inside the bucket

- **S3 does not have real folders.**

  Folders are a UI concept created from object keys.

Bucket endpoint example (shows folder-like path)

```
http://<bucket_name>.s3.<region>.amazonaws.com/images/abc.jpg
```

`/images/abc.jpg` looks like a folder path but it is actually part of the object key.

Default access & public access

- **By default, S3 buckets are private.**
- To access files publicly you must set bucket or object permissions.
- You can make the **bucket public** and then allow public access only to selected objects via policies.

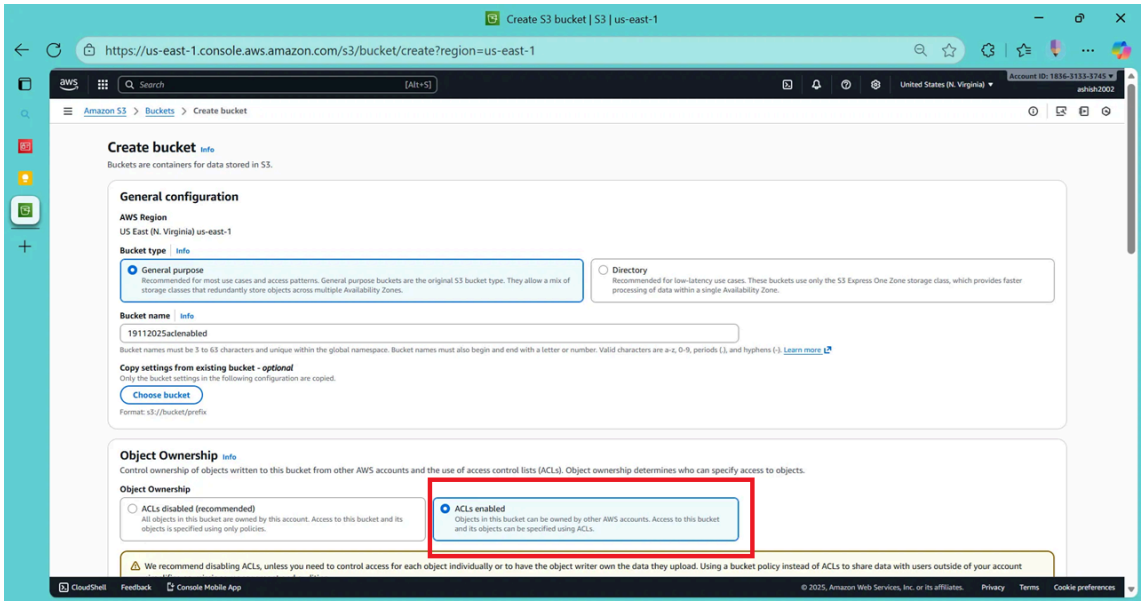
ACL disabled vs ACL enabled

1. **ACL disabled (recommended)**
   - Use bucket policies / IAM for fine-grained control.
   - Everything is private by default.
   - Allows to define permission at deep level.
   - You can allow or deny specific actions (upload, read, delete).
2. **ACL enabled**
   - Simple Read/Write permissions via ACLs.
   - Less flexible than policies. Good for simple use-cases.

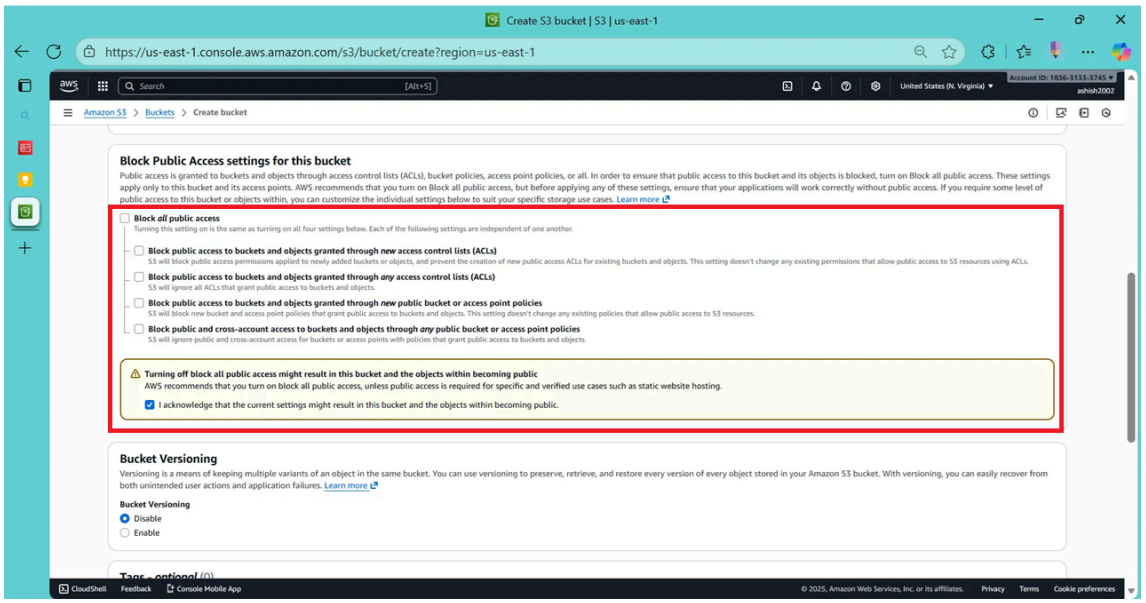
Creating ACL Enabled Bucket (Practical)

1. Go to AWS Console → S3.

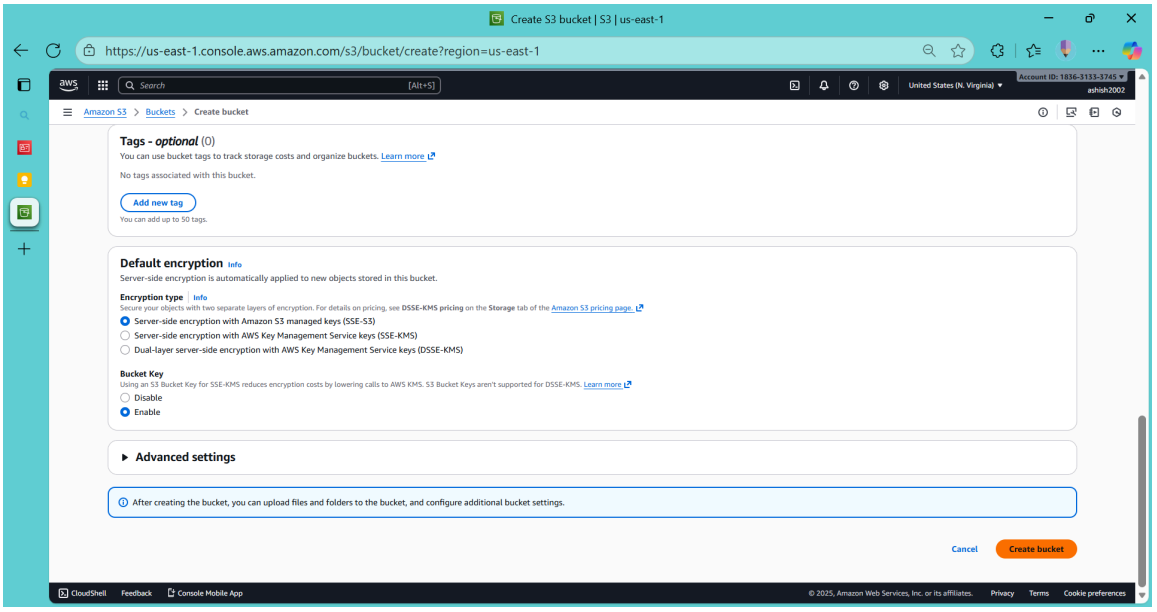
   Write bucket name → select **ACL enabled**.



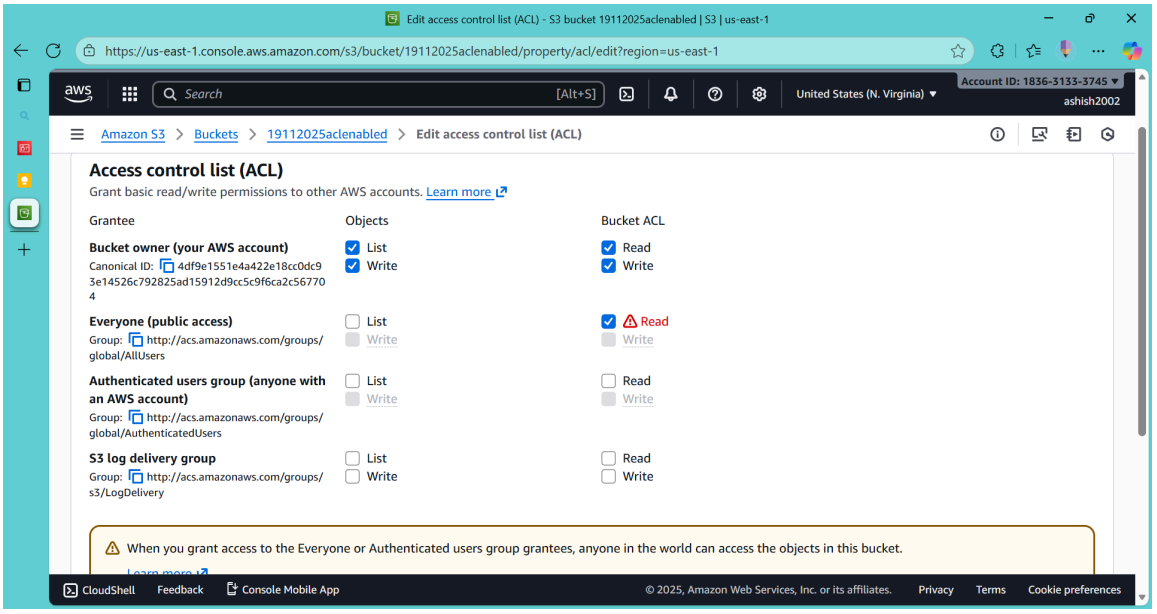
2. Allow public access for the bucket if needed.



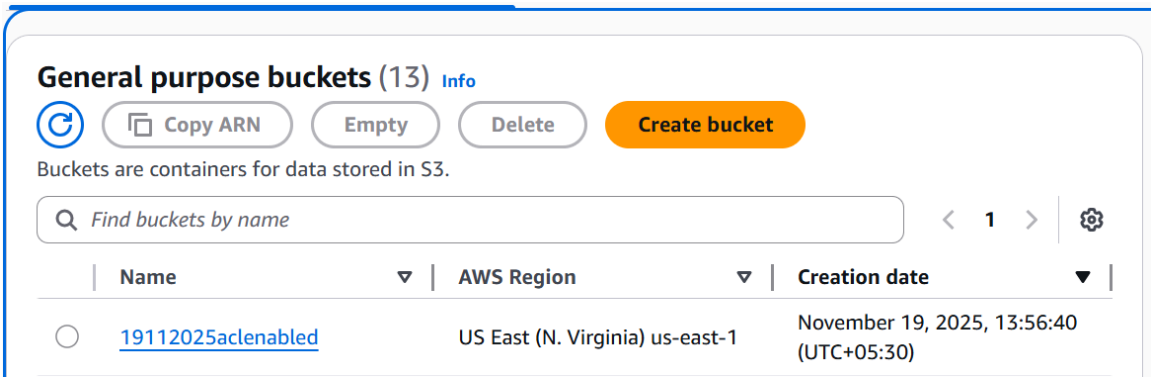
3. Create the bucket.



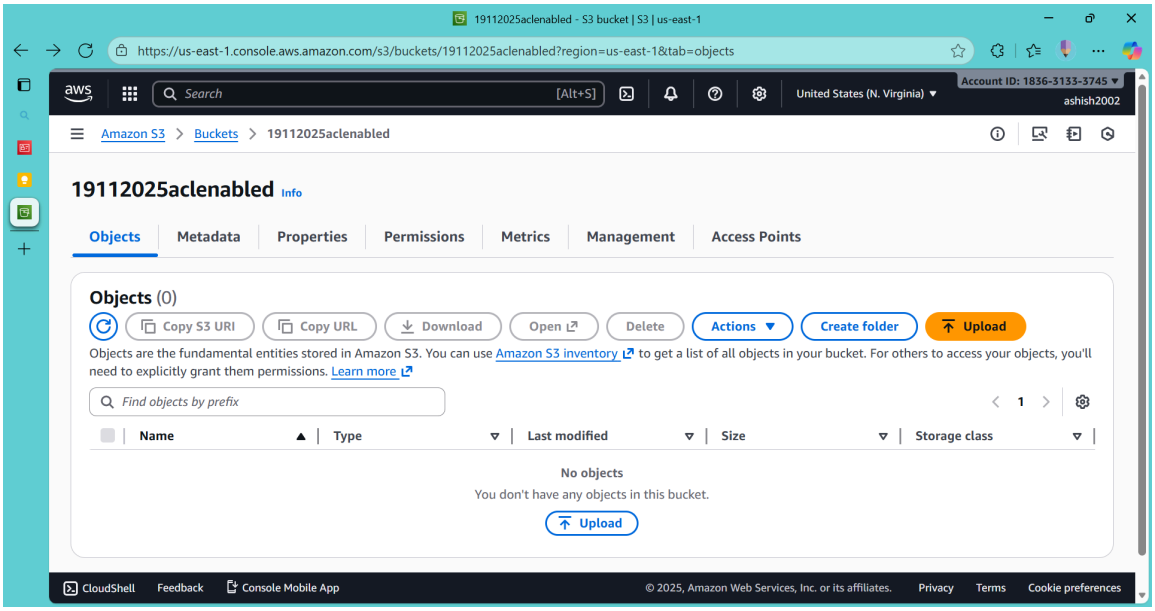
4. Give Read permission to the bucket if you want objects to be publicly readable.



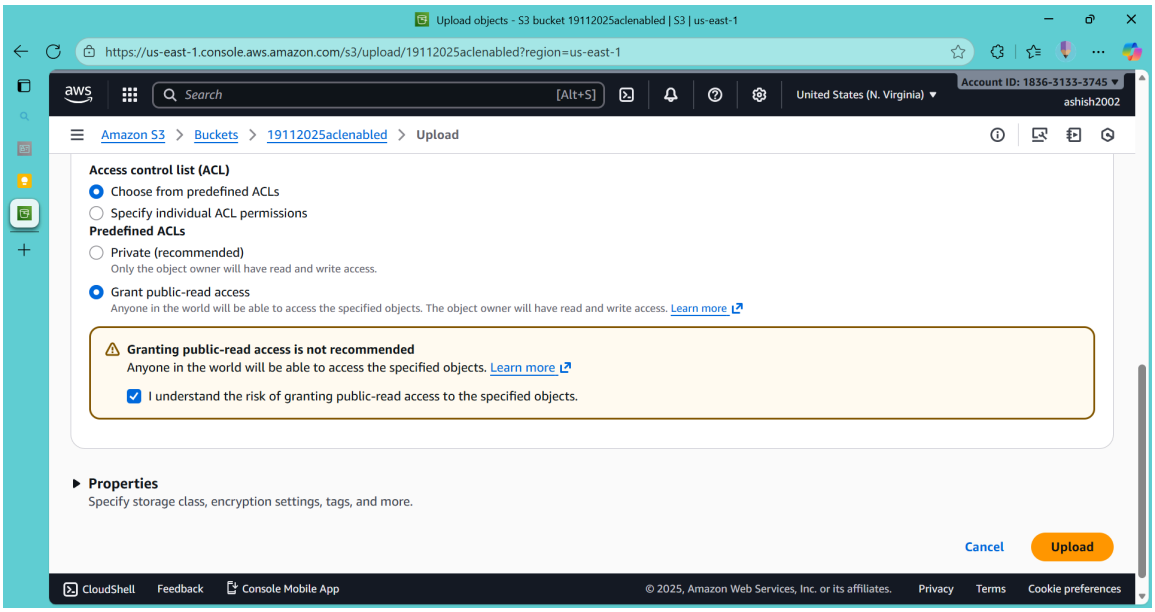
5. Upload the image or file.



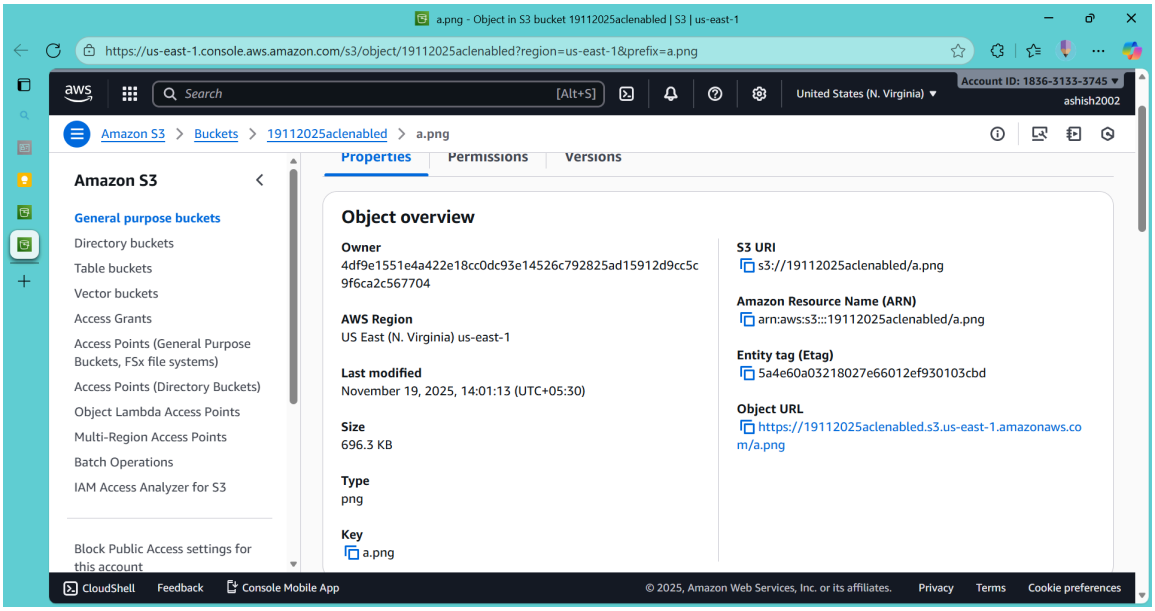
Click **Upload** and upload the file.



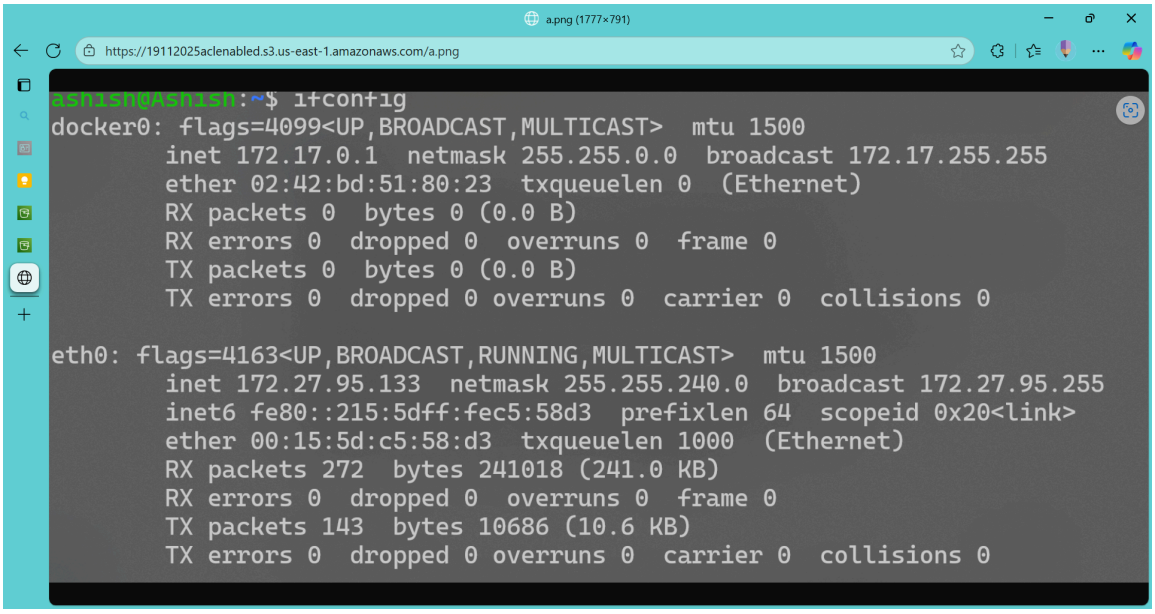
6. Grant public read access for that uploaded object.



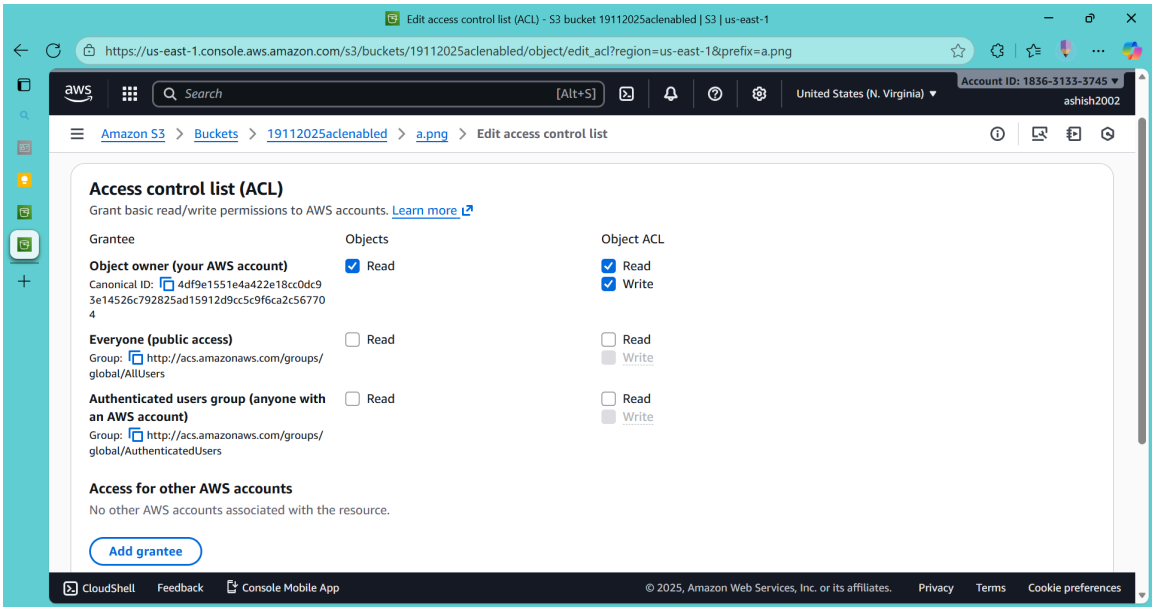
7. Copy the object URL and paste it into a browser.

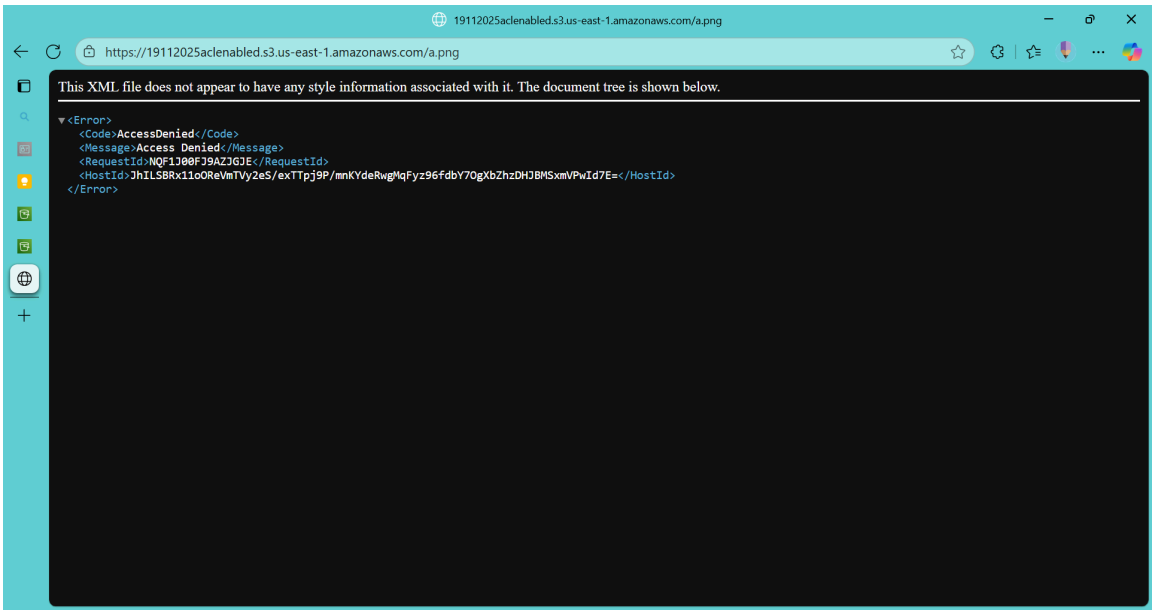


You will see the uploaded image.



8. Revoke permission for everyone to test access — you will see **Access Denied** for that object.



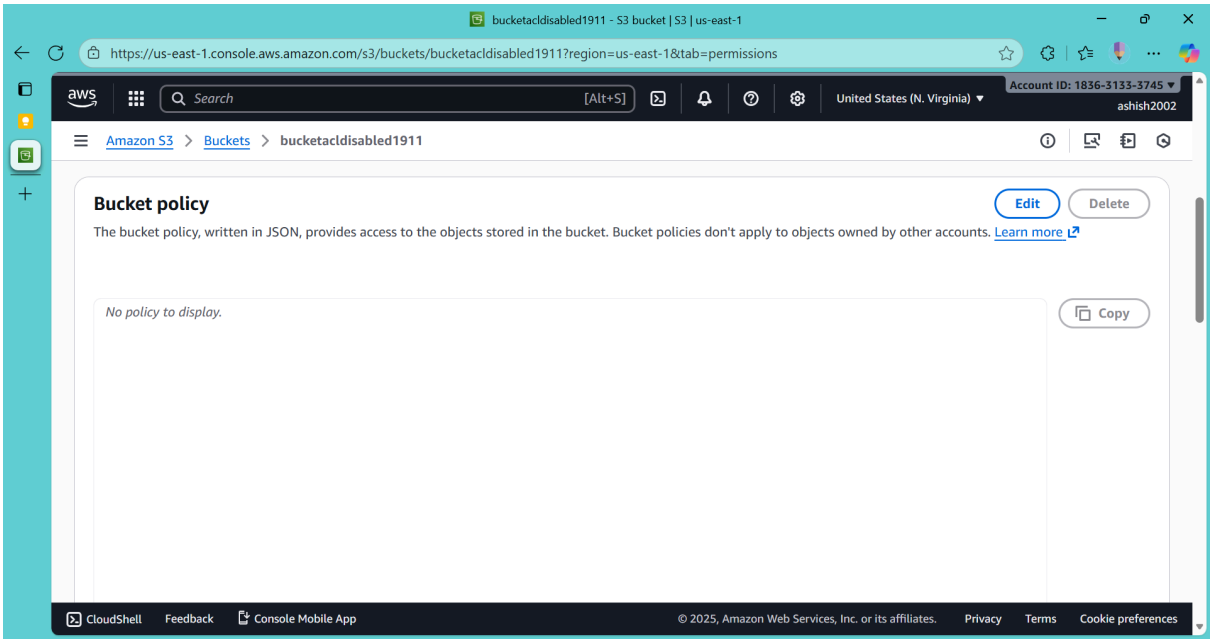


Creating ACL Disabled Bucket (Practical)

Follow the same steps above but choose **ACL disabled** when creating the bucket.

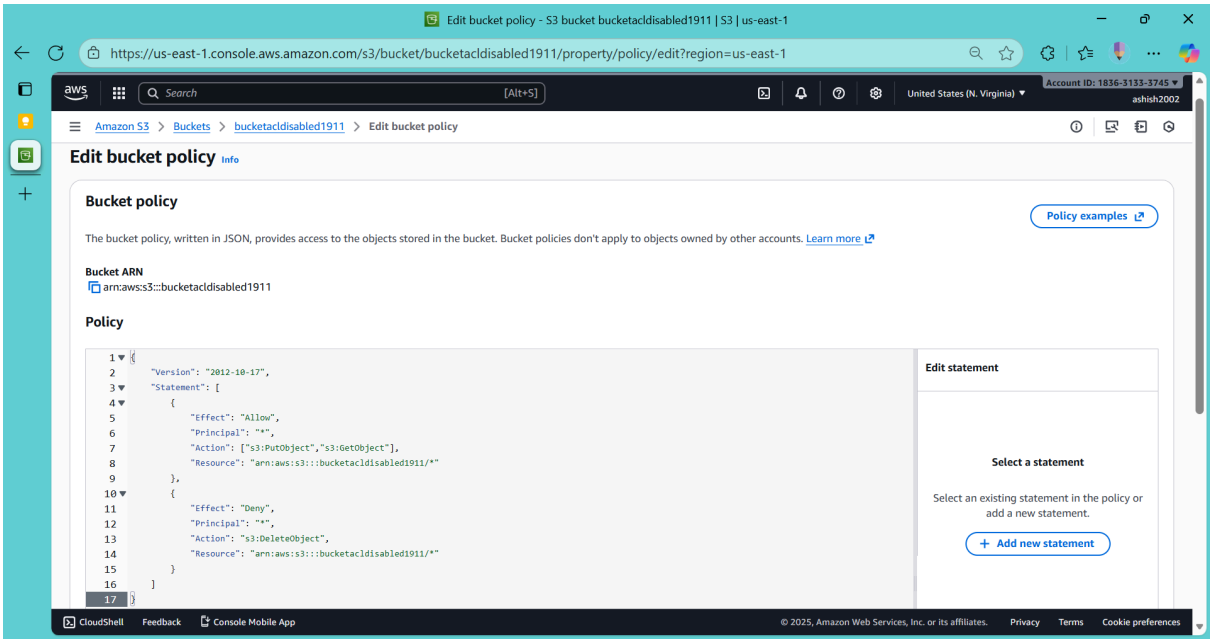
With ACL disabled you must use a **Bucket Policy** to control access.

Inside the bucket → Permissions → Bucket policy.



Example policy (fig 2.3):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": ["s3:PutObject", "s3:GetObject"],
      "Resource": "arn:aws:s3:::bucketacldisabled1911/*"
    },
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:DeleteObject",
      "Resource": "arn:aws:s3:::bucketacldisabled1911/*"
    }
  ]
}
```



ACL disabled gives minute-level control through policies. For example: allow uploading and reading, but deny delete.

Elements inside a policy

- **Effect** → `Allow` or `Deny`
- **Resource(s)** → which object(s) the policy applies to
- **Action(s)** → e.g., `s3:GetObject`, `s3:DeleteObject`, `s3:PutObject`
- **Principal** → who can access (IAM user, role, or for everyone)

Amazon Resource Name (ARN)

Used to identify AWS resources. Format:

```
arn:<partition>:<service>:<region>:<account-id>:<resourceType>/<resourceId>
```

S3 example (S3 is global):

```
arn:aws:s3:::bucketname
```

Partitions include `aws`, `aws-cn`, `aws-us-gov`.

Preventing object deletion — alternatives

- Use **bucket policies** to `Deny` `s3:DeleteObject` for certain principals.

  → But if someone can change the policy, they could remove that deny, so lock down who can edit policies via IAM.

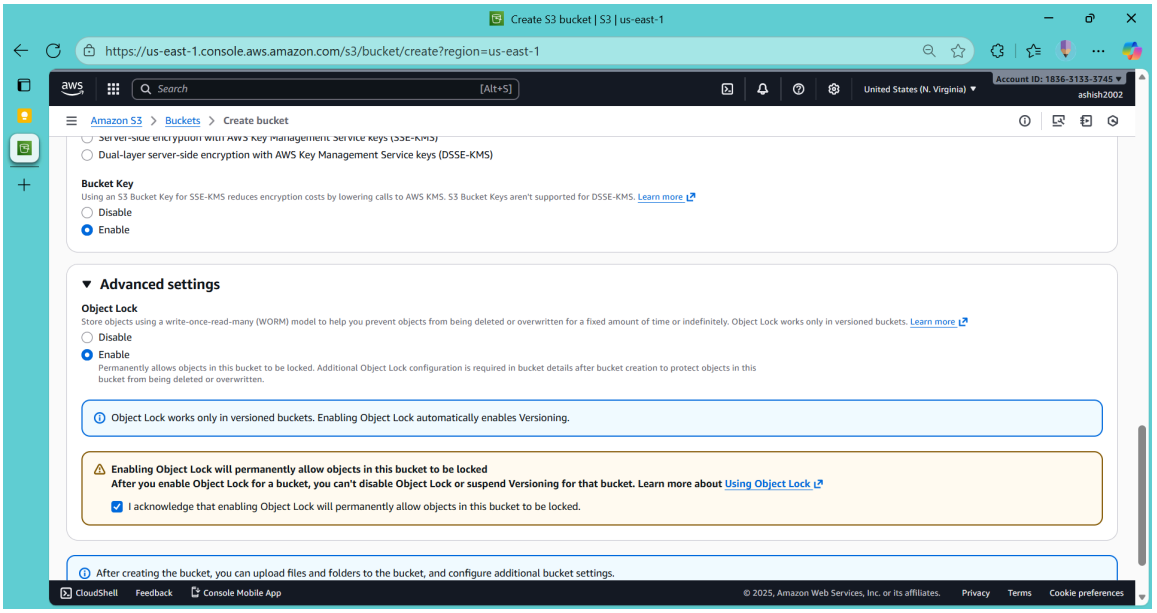
- Use **Object Lock** and **Versioning** for stronger protection (recommended for compliance or immutability).

Object Lock (enable at bucket creation)

Important: Object Lock must be enabled when you create the bucket. You *cannot* enable it later. Object Lock requires versioning.

Steps:

1. Enable **Object Lock** while creating the bucket.



2. Optionally set **Default Retention** rules.

Behavior notes:

- Enabling Object Lock automatically enables bucket **versioning**.
- Object Lock prevents objects from being deleted or overwritten according to retention rules.

Object Lock Modes & Cases

Case 1: Object Lock enabled, Default Retention disabled

- Behavior: Works like versioning. If someone deletes the object from the object list, a **delete marker** is created and older versions remain available. You can restore by removing the delete marker.

Case 2: Object Lock enabled, Default Retention enabled

- You must choose a retention mode:
  - **Governance mode** → Privileged IAM users (with special permissions) can bypass retention and delete during the retention period.
  - **Compliance mode** → No one can delete the object or remove retention until the retention period expires (strongest protection).

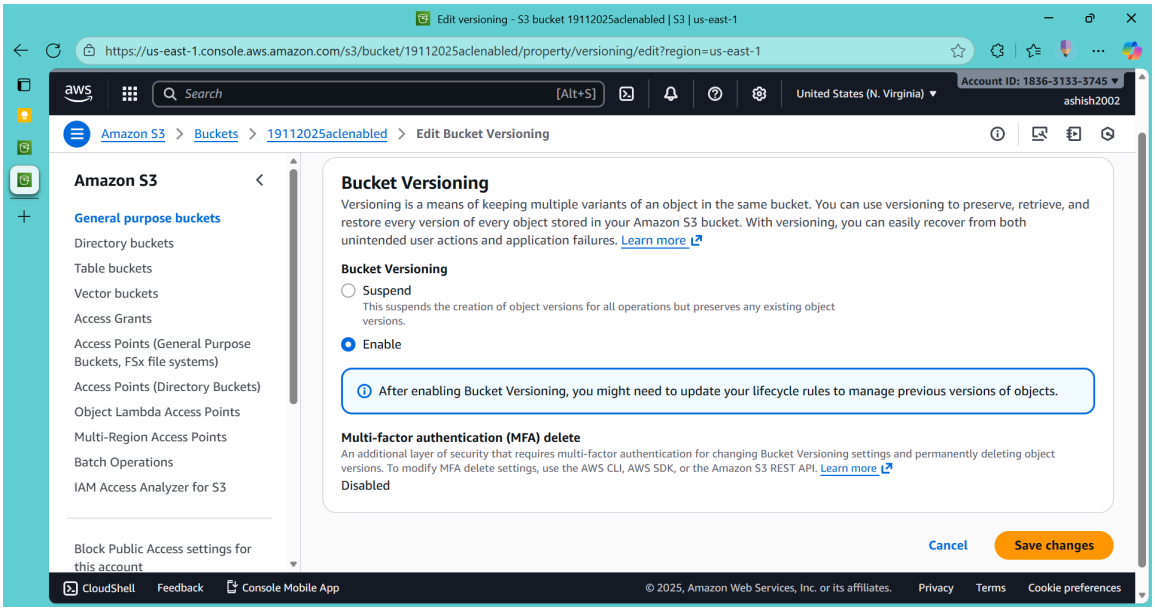
Use Compliance mode for strict immutability. Use Governance if you need controlled flexibility.

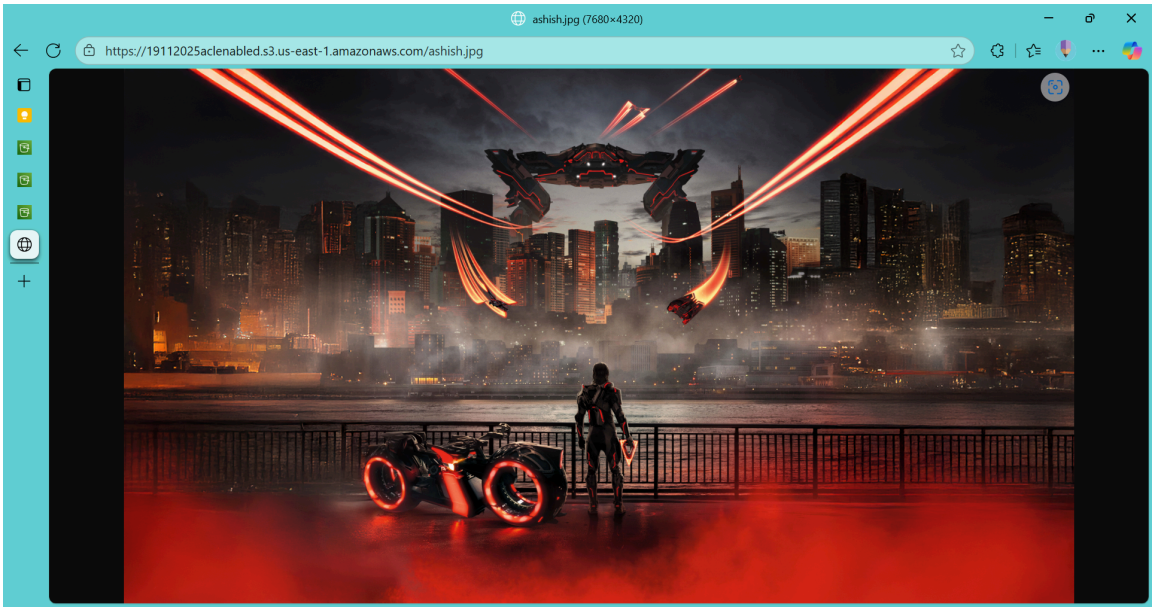
Bucket Versioning (how it works)

- **Versioning can be enabled anytime.**
- When versioning is on, the same object key can have multiple versions.

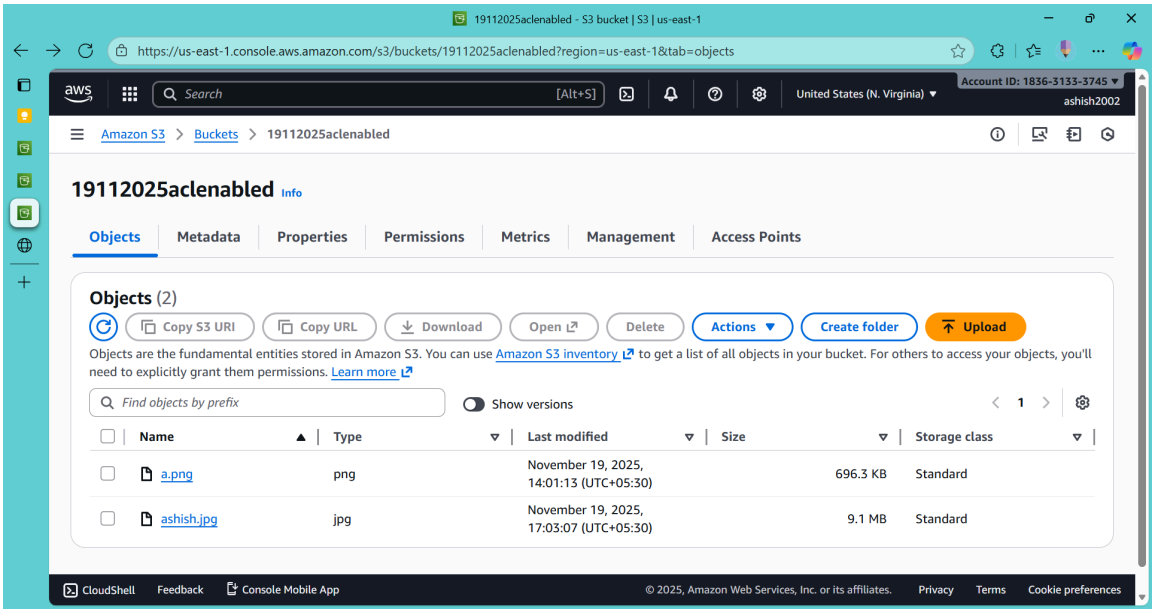
Example workflow:

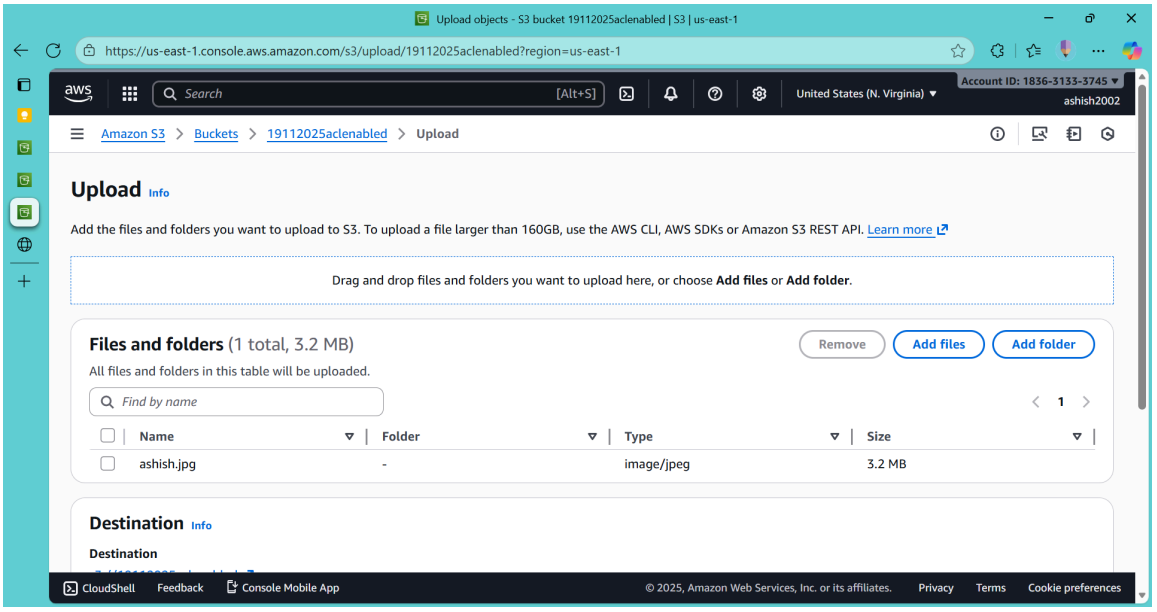
1. Upload `ashish.png` → version V1.



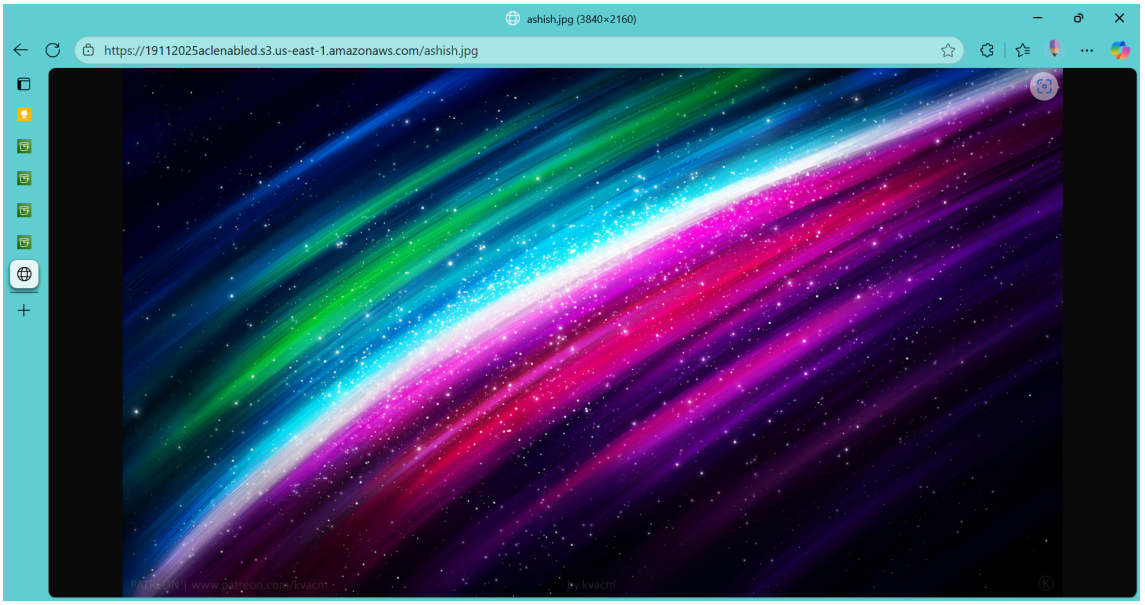


2. Upload another file with the same name `ashish.jpg` → new version V2; older V1 remains visible under versions.





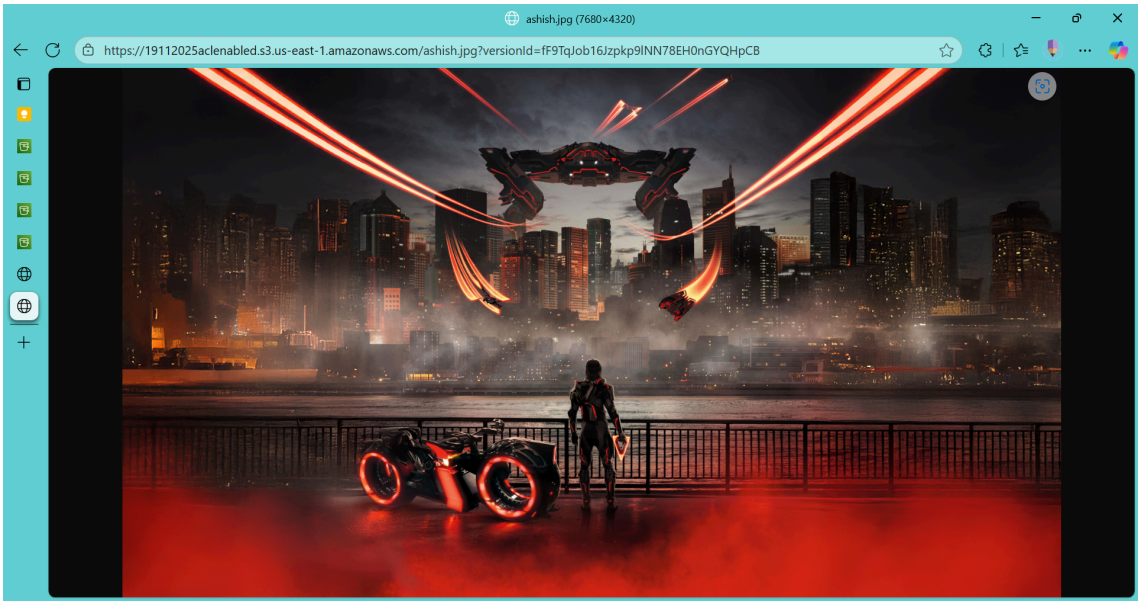
3. The object URL without a version shows the **latest version**.



4. To access an older version:

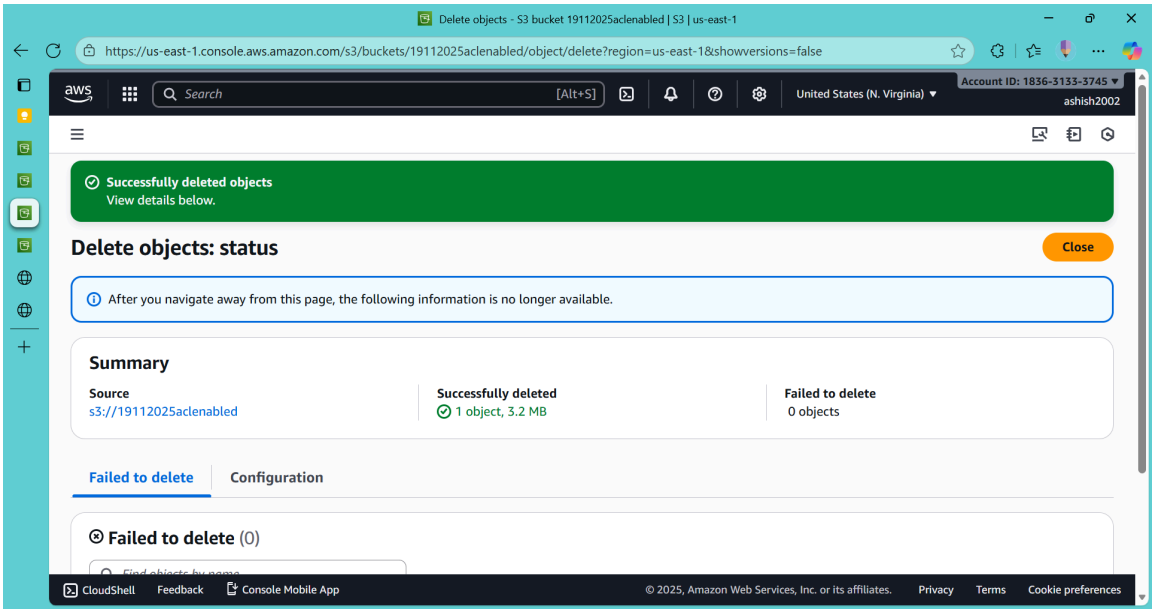
```
bucketname.region.amazonaws.com/ashish.jpg?versionId=4344
```

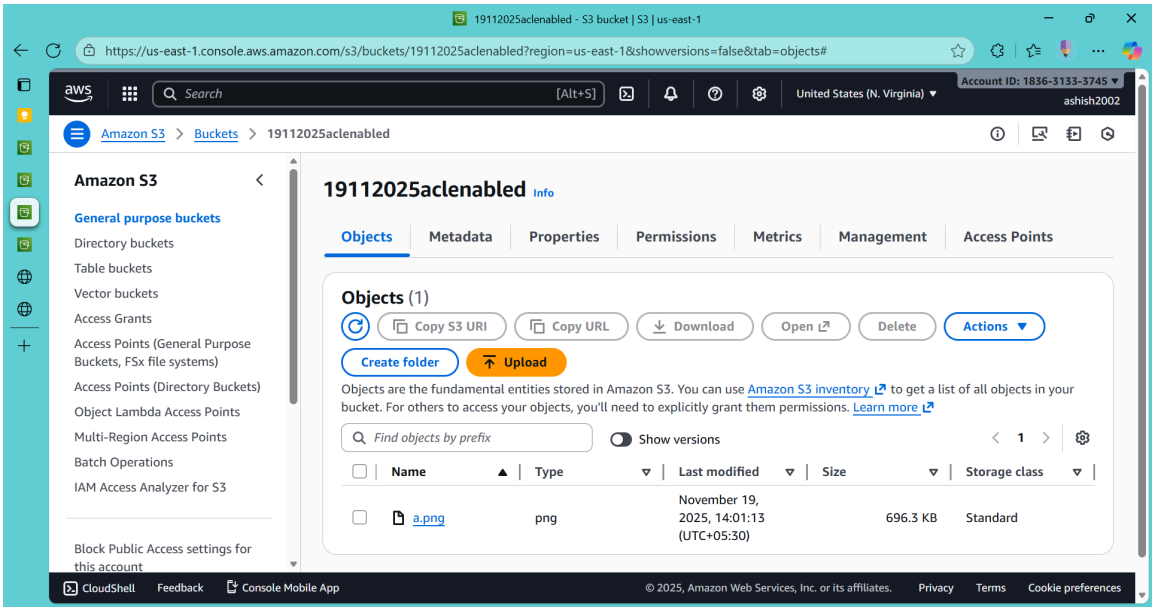
(Each version has a `versionId`.) (fig 3.1)



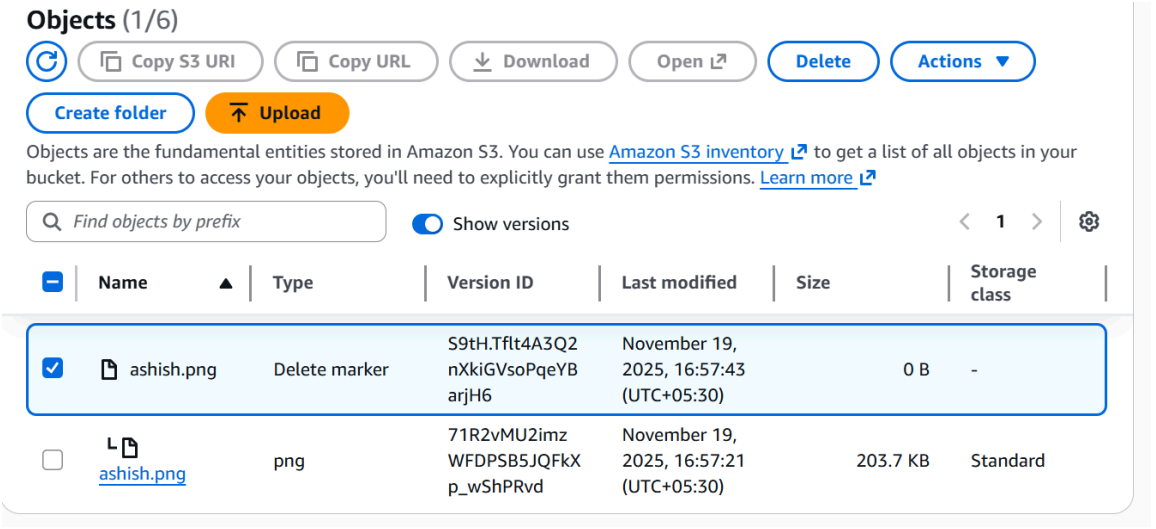
Delete behavior with versioning:

- Deleting the object from the object list adds a **delete marker**; the object disappears from the normal list but older versions are still stored.





- To restore the object in the object list, remove the delete marker (it acts like a recycle bin).



- Deleting a specific version from the versions view permanently removes that version.
- If **versioning is disabled**, uploading a file with the same name replaces the old file (no version history).

API note: To read object versions via API you may need `s3:GetObjectVersion` permission.

S3 Static Website Hosting

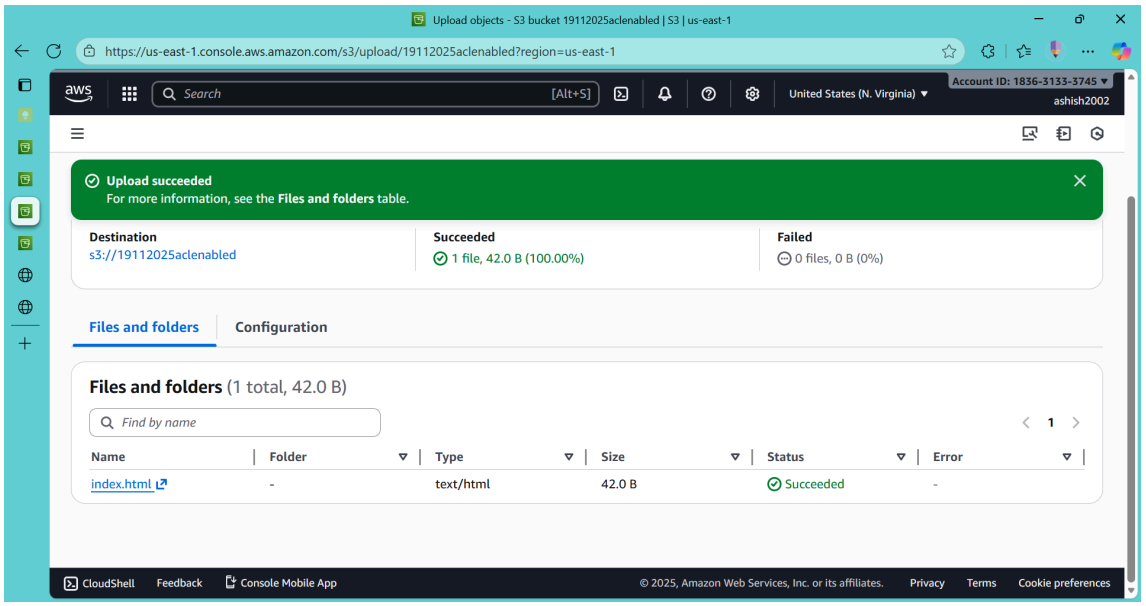
- Website URL format:

```
<bucket_name>.s3-website.<region>.amazonaws.com
```

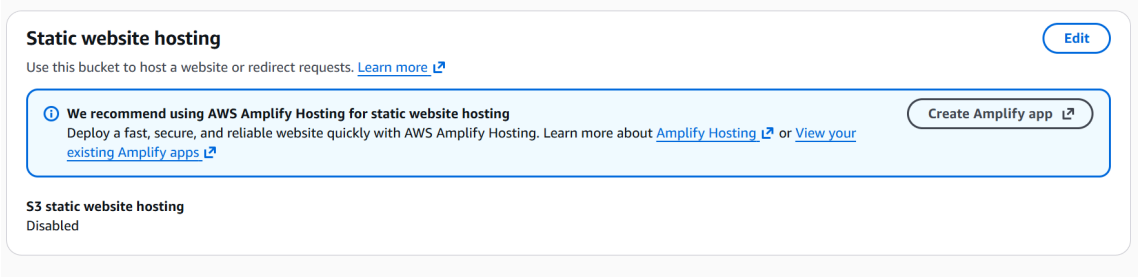
- S3 static websites are **HTTP only**. To serve over **HTTPS**, put **CloudFront** in front of S3.

Practical

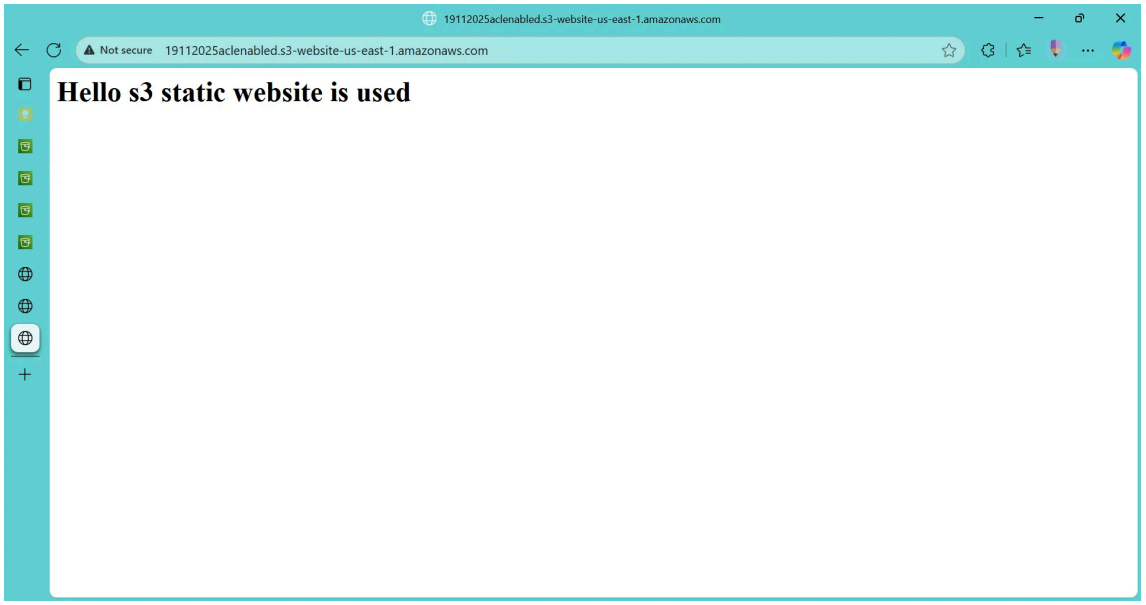
1. Upload `index.html` to your bucket.



2. Enable static website hosting from bucket properties.



3. You will get a website URL to open.



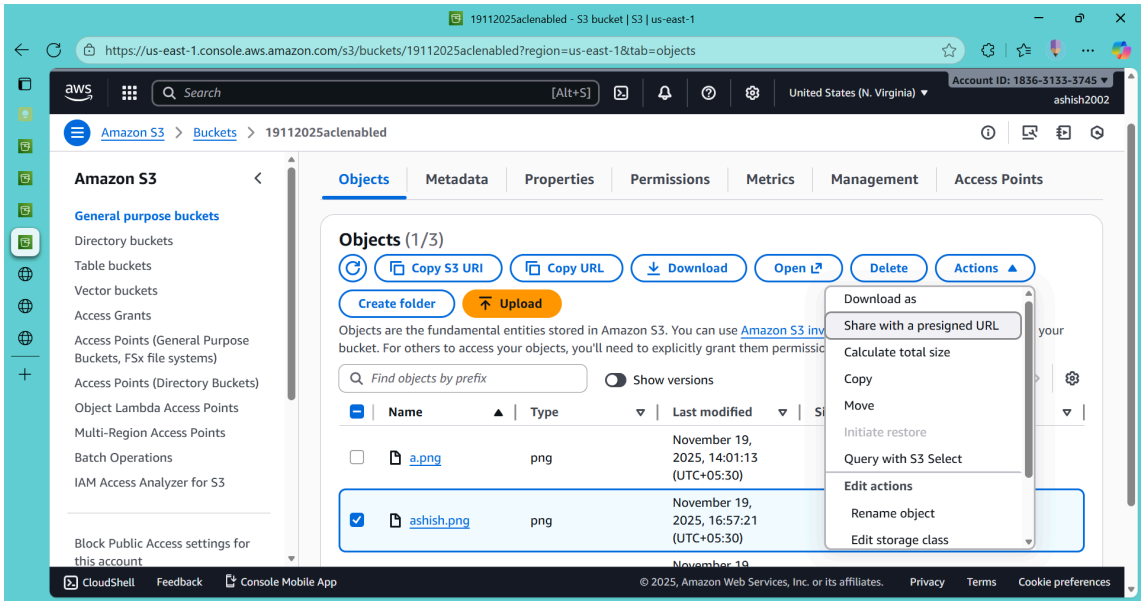
It’s recommended to use S3 (with CloudFront for HTTPS) for static websites instead of running EC2 for static hosting.

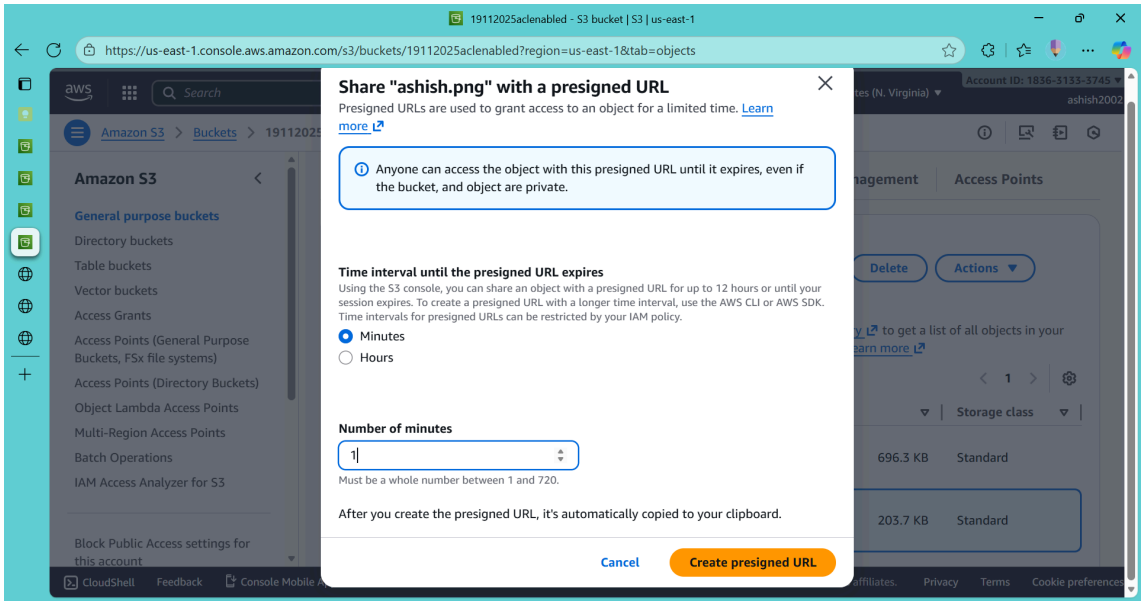
Pre-signed URLs

- A **pre-signed URL** is a temporary link that expires after a set time.
- Common use: give time-limited access for streaming or downloads.

Practical

- In the Console: select `ashish.jpg` → Actions → Share with a presigned URL.





- Example: generate a URL that is valid for **1 minute** — after that it will expire.

MFA Delete (Multi-Factor Authentication for versioning)

- **Requirement:** Bucket versioning must be ON.
- Steps:
  1. Enable MFA device in AWS Security Credentials (authenticator app or hardware).
  2. Note the MFA device ARN.
  3. Use AWS CLI with configured credentials to enable MFA Delete.

Example CLI command:

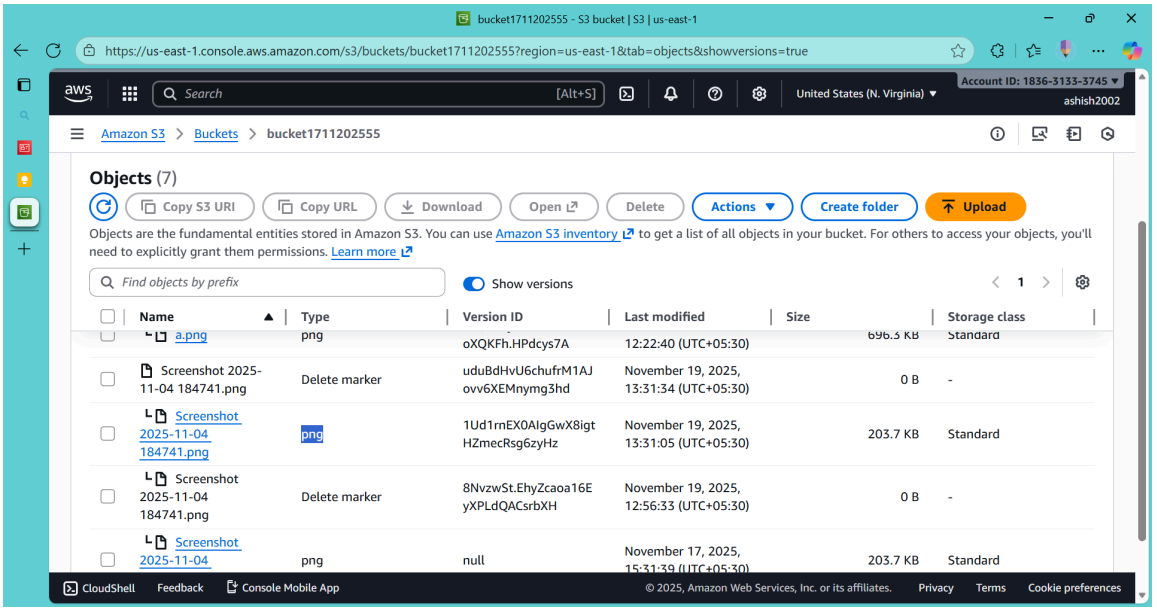
```
aws s3api put-bucket-versioning --bucket bucket1711202555 \
--versioning-configuration Status=Enabled,MFADelete=Enabled \
--mfa "arn:aws:iam::183631333745:mfa/Authapp 123456"
```

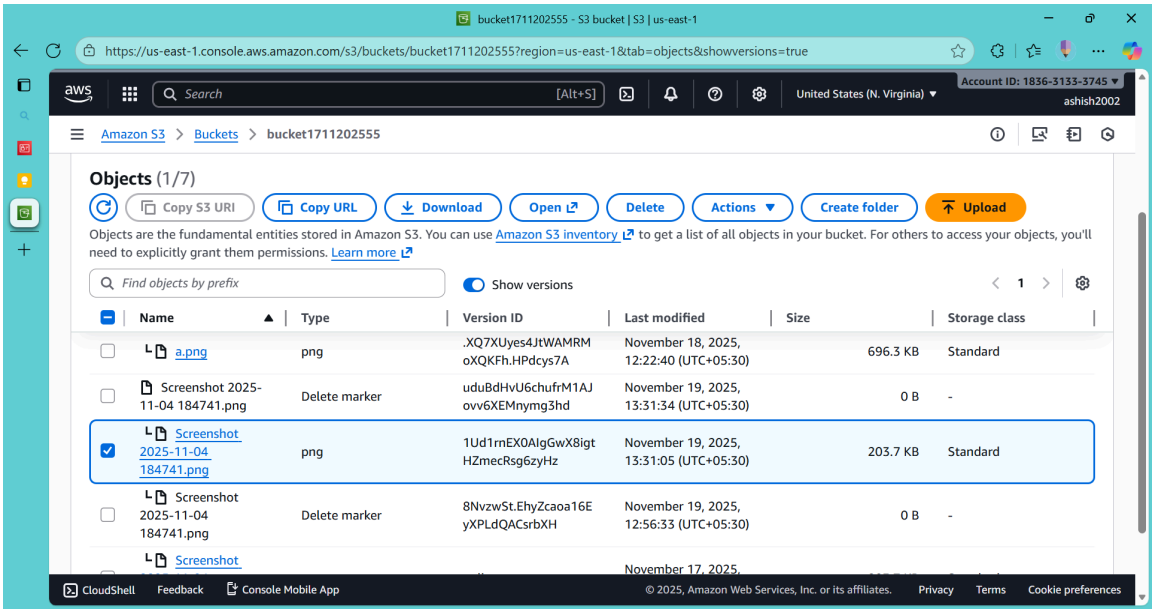
Ways to implement MFA:

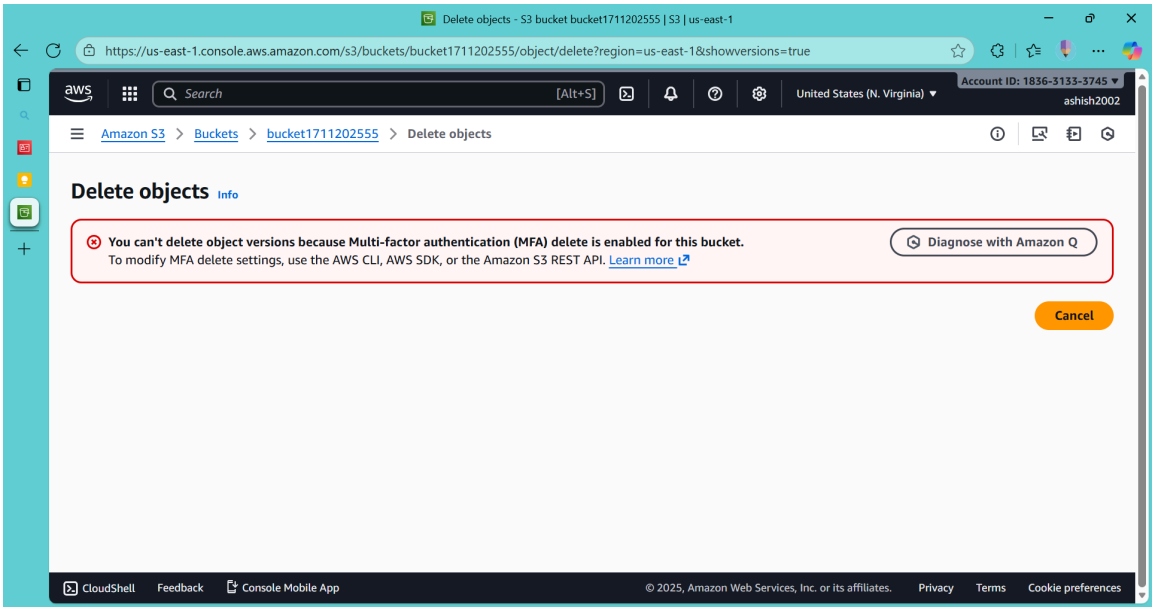
- Software TOTP (authenticator app)
- YubiKey (hardware)
- Other hardware TOTP devices

After enabling MFA Delete:

- Deleting from the main object list still puts a delete marker (versioning behavior).
- Deleting a **specific version** or removing delete markers requires MFA — extra protection.







Final practical reminders

- Enable **Versioning** to keep older copies and recover from accidental deletes.
- Enable **Object Lock** at bucket creation for immutable storage (compliance needs).
- Use **Bucket Policies** and **IAM** to restrict who can edit policies and who can delete objects.
- Use **Pre-signed URLs** for temporary, time-limited access.
- Use **CloudFront** if you need HTTPS for a static S3 website.

Thank you.