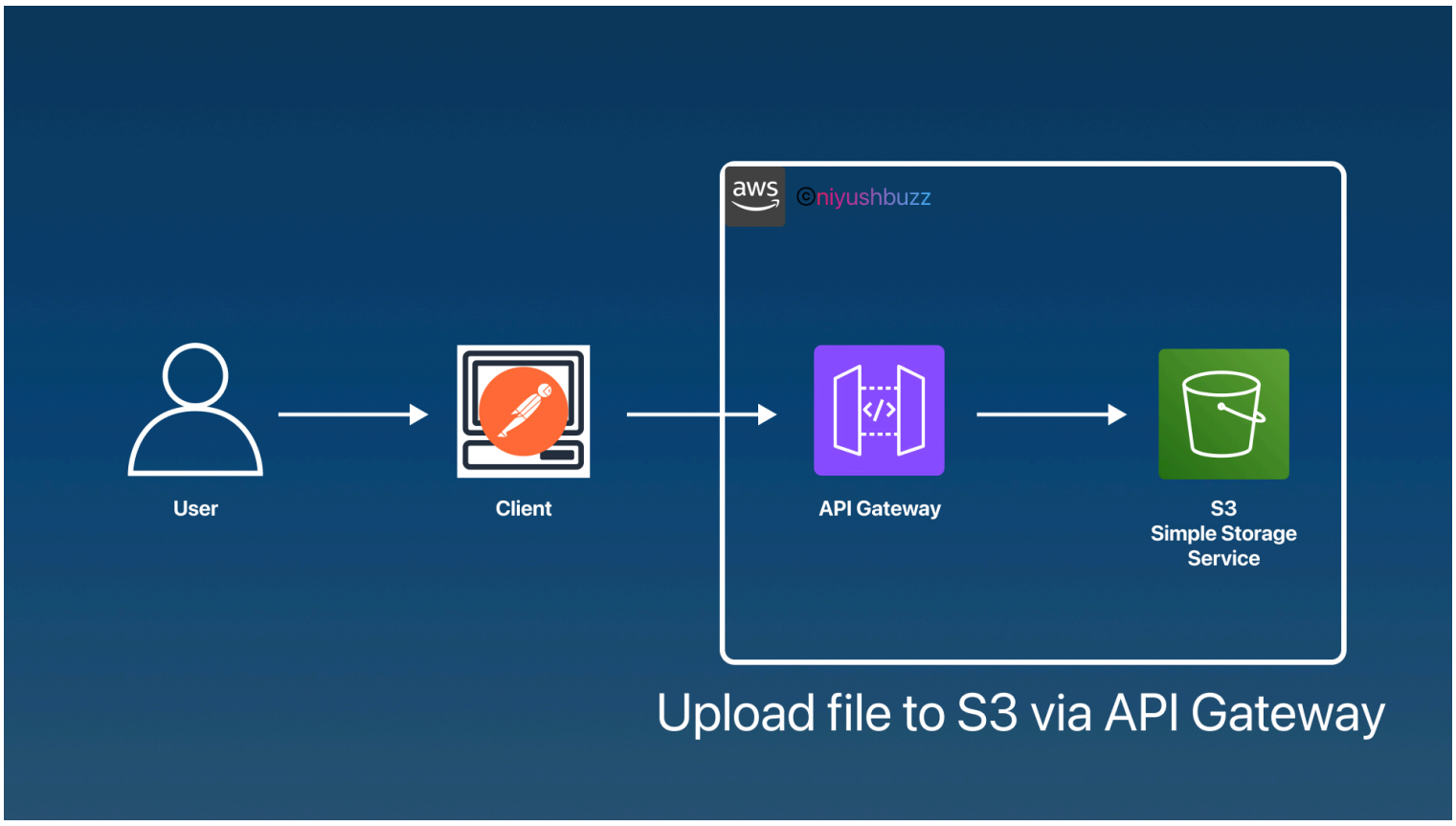
Uploading to S3 via API Gateway

Created	@21 December 2025 19:28
Tags	

Architecture

Architecture



Client → API Gateway → S3

> In this project, we are going to setup an API Gateway endpoint that uses file name parameter as the S3 object key to upload files to an S3 bucket.

This project Includes the Following steps :

- [Creating an S3 Bucket]
- **Creating IAM Roles and Policies For the bucket.**
- **Creating an API Gateway Endpoints**
- **Test Uploading a file via Postman**
- **Deletion of Resources**

Pros and Cons of Using this Architecture

▼ Pros:

- Serverless, no lambda cost, cheapest option.
- Low Latency and Simple
- Great for Large uploads - client streams through API Gateway into S3.

▼ Cons :

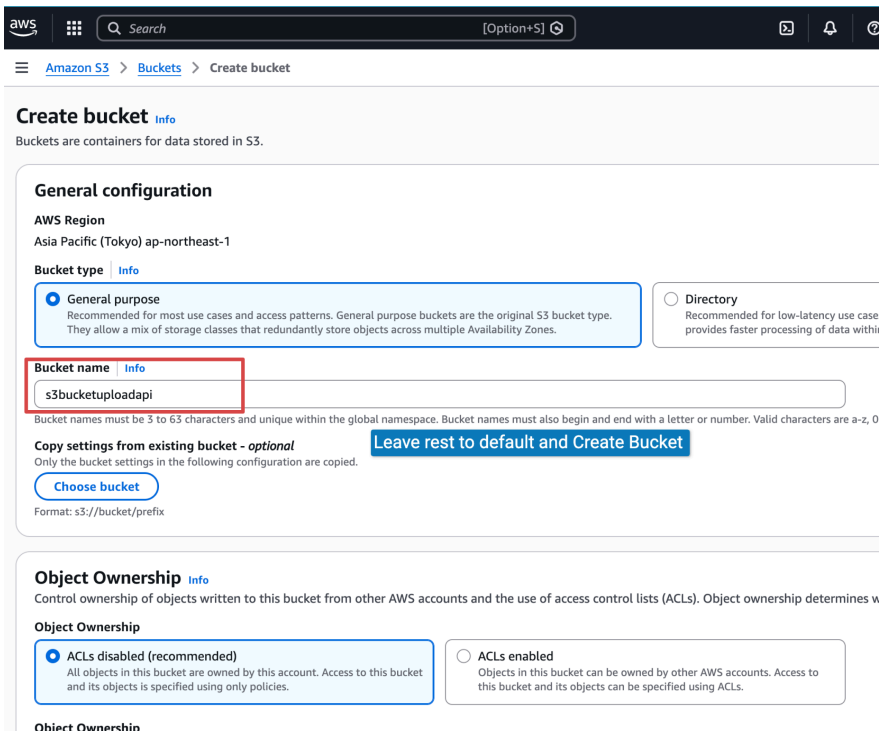
- Hard to secure
  - must configure IAM role
  - Resource policy
  - Correct S3 permissions
- No file validation or custom logic.
- API Gateway max payload : 10MB (REST) or 6 MB (HTTPS)
  - cannot support large uploads directly unless multipart + special handling.

> ⚠ `Not recommended for General or large file uploads.

⌘ Step 1 : Creating S3 Buckets.

- Navigate to `AWS Console` → `S3`
- `Create Bucket` → give it a name (a unique one)
- Bucket Type : `General Purpose`
- Block public access : `True` → `Create Bucket`

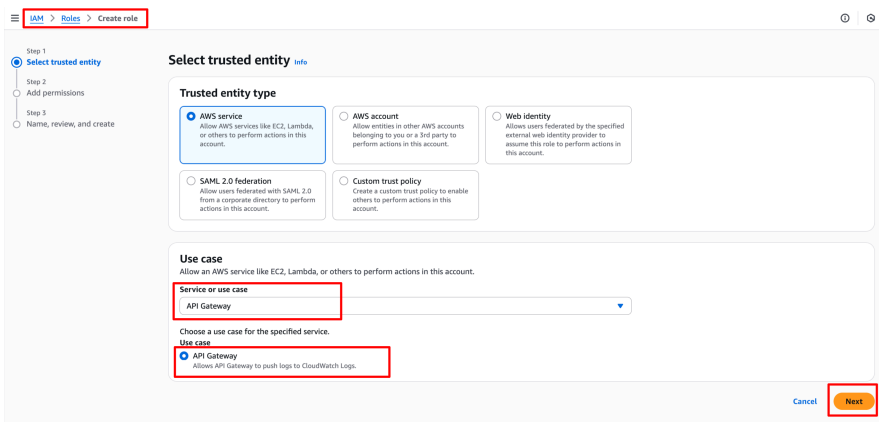
▼ Image : Creating Bucket

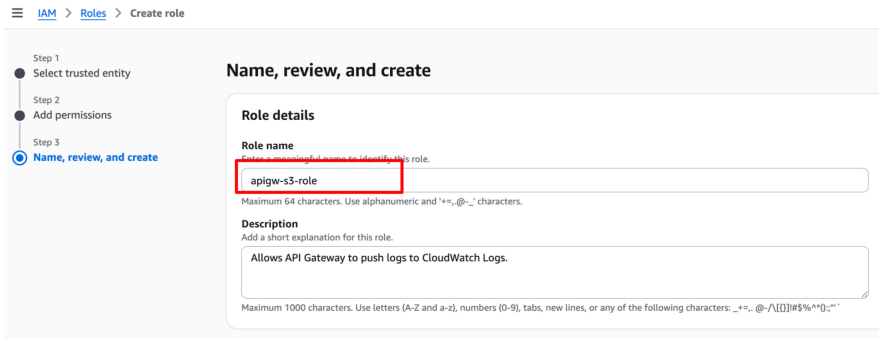


⌘ Step 2 : Creating IAM Roles and Policy

- Headover to `IAM` → `Roles` → `Create role`
- Trusted Entity Type : `AWS Service`
  - Use Case : API Gateway → `Next`
- Name, Review and create:
  - Role name : `apiGW-s3-upload`
  - Description : (optional)
- Finally click on `Create Role`

▼ Image : IAM role Creation



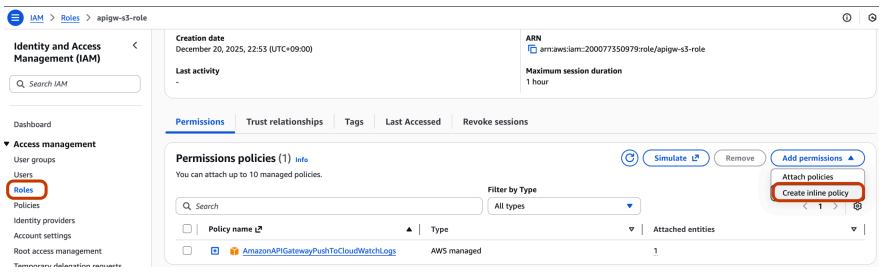


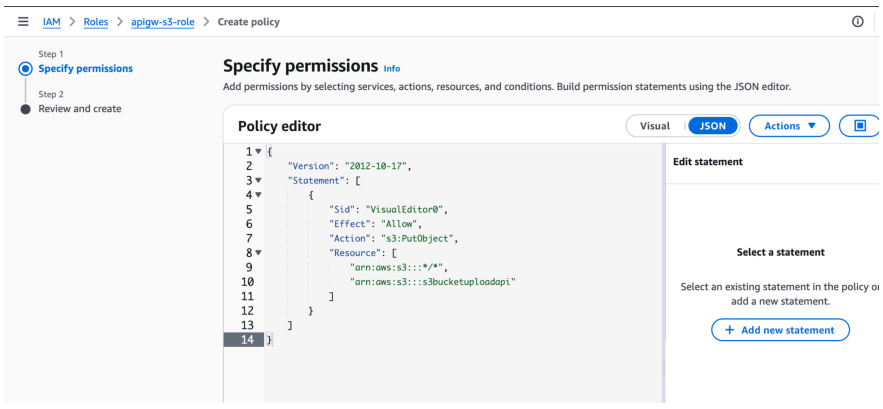
Step 2a : Attaching Policy to the Role.

- Navigate to `Permissions` tab in `apiGW-s3-upload` Role.
- Under Permissions policies → `Add permissions` → `Create inline policy`

▼ Image : attaching policy to the role

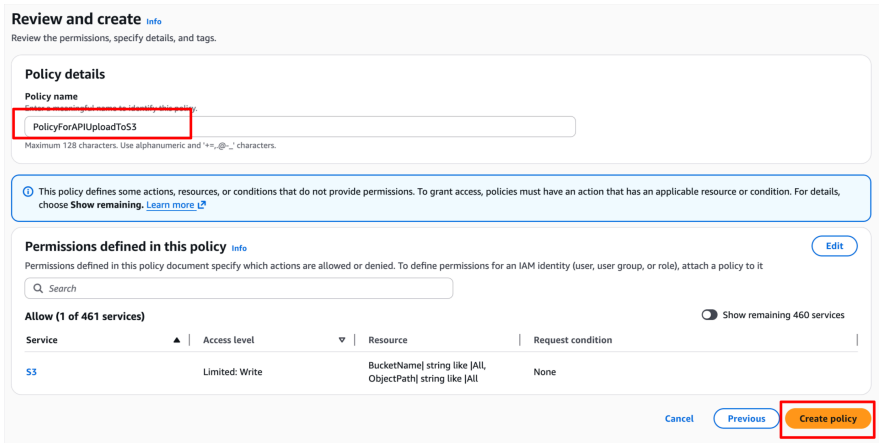
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": [
        "arn:aws:s3:::*/*",
        "arn:aws:s3:::s3bucketuploadapi"
      ]
    }
  ]
}
```

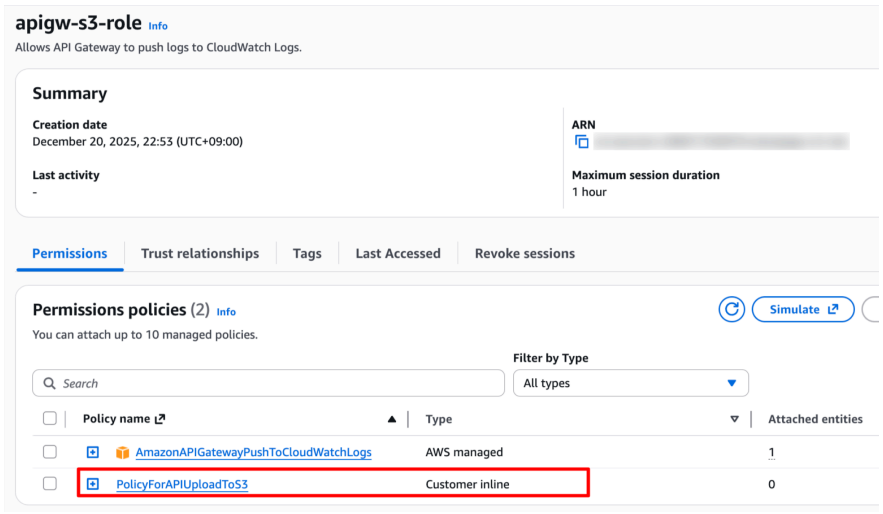




- Policy name : `API-GW-UploadPurpose-Policy` → `Create role`

▼ Image : policy





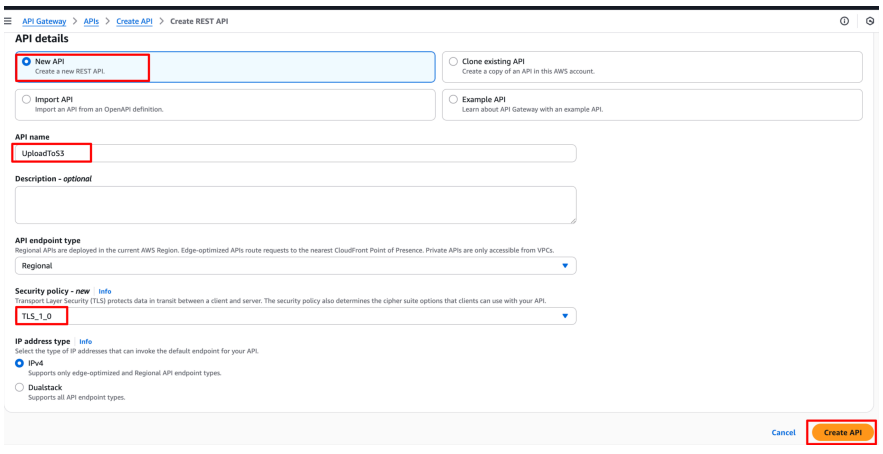
⌘ Step 3 : Setting up the API.

- Head over to `API GateWay` → `Create API`
- API Type : `REST API` → `Build`
- Click on `New API` → API Name : `UploadToS3` → `Create API`

▼ Image : setting up the API

Once the API is created click on Resource path : `/`

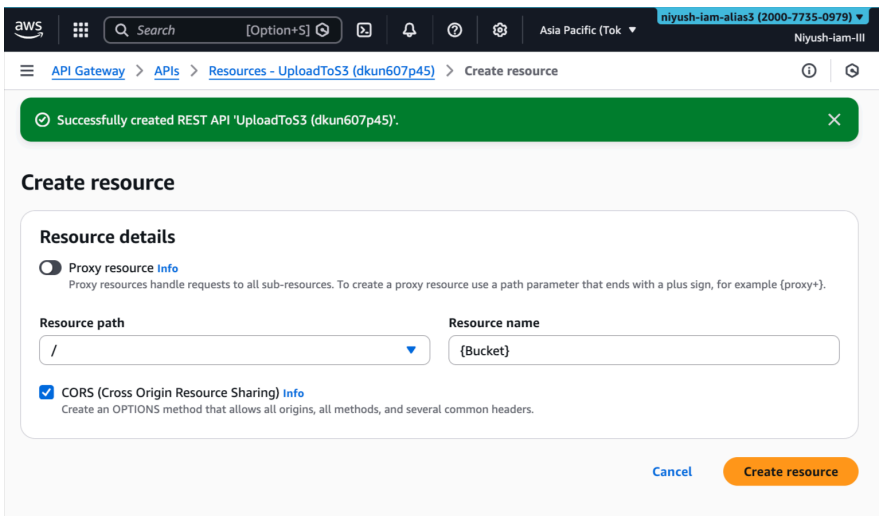
- Resource name : {Bucket}
- Enable CORS
- in the {bucket} click on create resource
- Resource path : /{bucket}/
- Resource name : {filename}
- enable CORS and hit Create resource
- in the {filename} resource, create a method
- Method PUT
- AWS service : s3
- Action type : use path override
  - Path override : {bucket}/{filename}
  - Execution role : **Provide the ARN of the role created earlier**



▼ Image : Creating resources

In the PUT Method

- Integration Request → Edit
- URL path parameters :
  - Name : bucket , Mapped from : method.request.path.bucket
  - Name : filename , Mapped from : method.request.path.filename
  - Click Save
  - Finally click on Deploy API

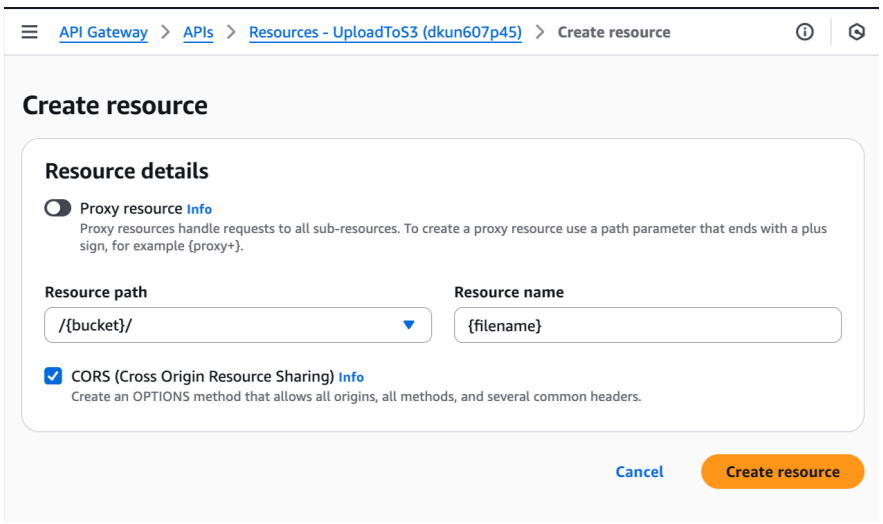


Binary Media Type Settings.

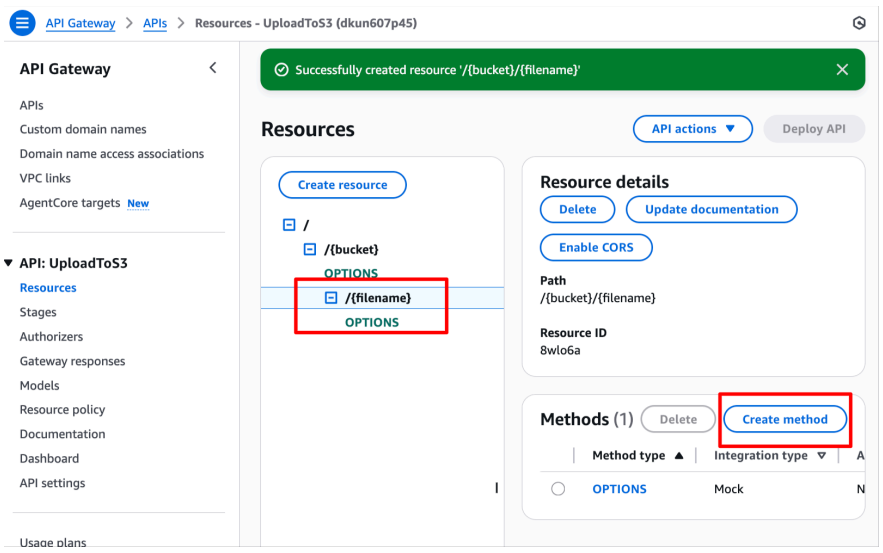
In the API settings > Binary media types → manage the settings to allow any file types to be uploaded.

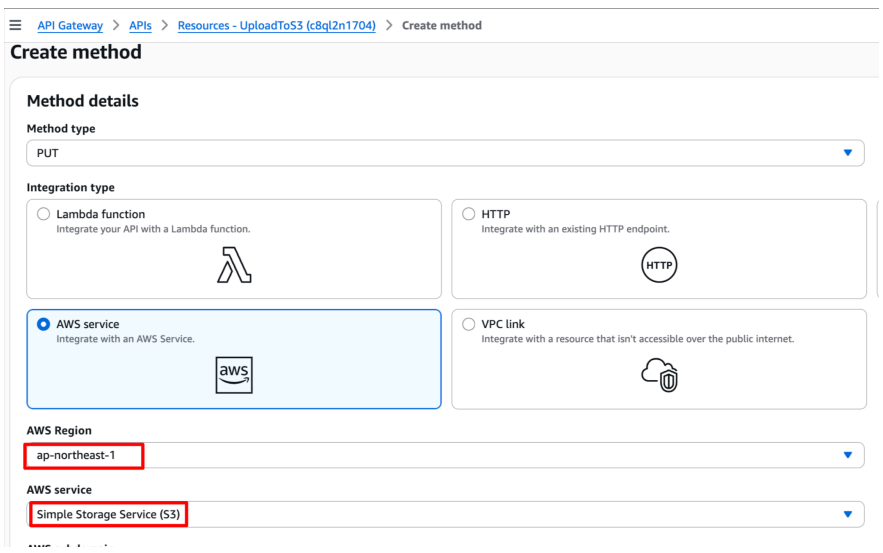
- API settings → Binary media types → Manage media types
- */* → Save Changes

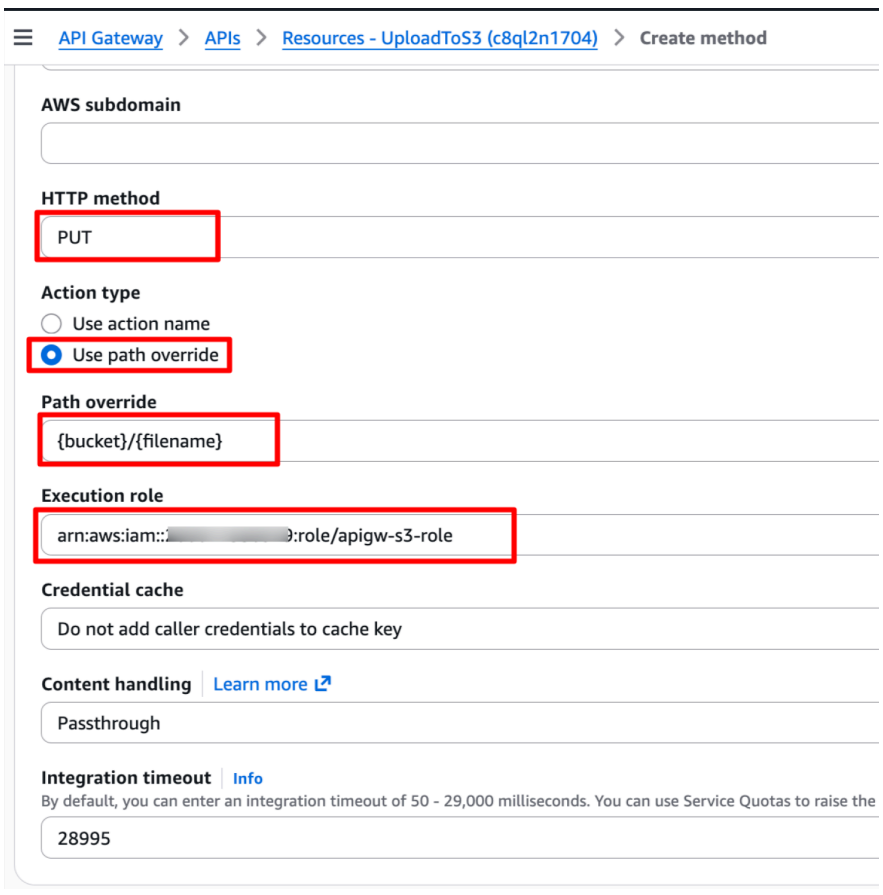
Go to Stages , select PUT method and copy the Invoke URL



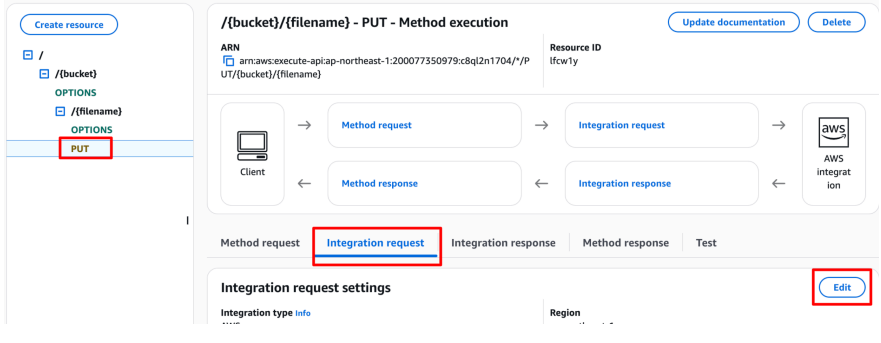
▼ image : Crating method for {filename}

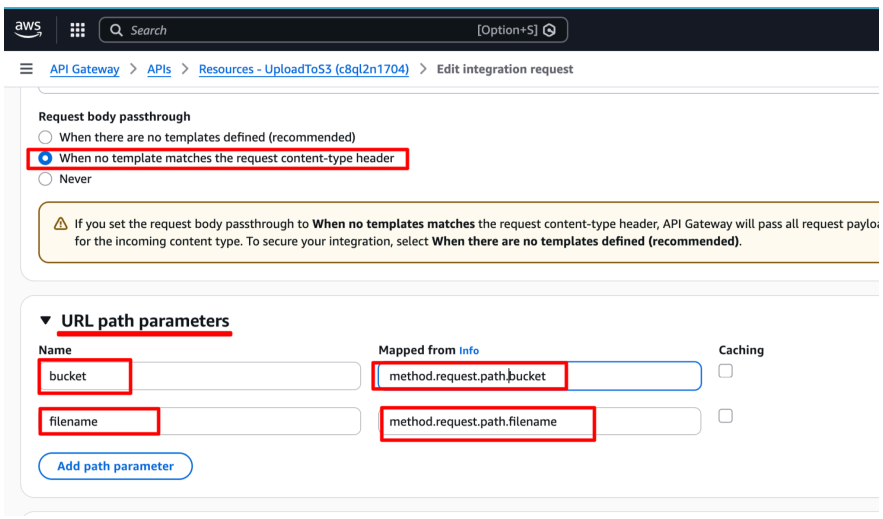

API Gateway > APIs > Resources - UploadToS3 (dkun607p45)
API Gateway
APIs
Custom domain names
Domain name access associations
VPC links
AgentCore targets New
API: UploadToS3
Resources
Stages
Authorizers
Gateway responses
Models
Resource policy
Documentation
Dashboard
API settings
Usage plans
Successfully created resource '/{bucket}/{filename}'
Resources
API actions
Deploy API
Create resource
/
/{bucket}
OPTIONS
/{filename}
OPTIONS
Resource details
Delete
Update documentation
Enable CORS
Path
/{bucket}/{filename}
Resource ID
8wlo6a
Methods (1)
Delete
Create method
Method type
Integration type
OPTIONS
Mock


API Gateway > APIs > Resources - UploadToS3 (c8ql2n1704) > Create method
Create method
Method details
Method type
PUT
Integration type
Lambda function
Integrate your API with a Lambda function.
HTTP
Integrate with an existing HTTP endpoint.
AWS service
Integrate with an AWS Service.
VPC link
Integrate with a resource that isn't accessible over the public internet.
AWS Region
ap-northeast-1
AWS service
Simple Storage Service (S3)

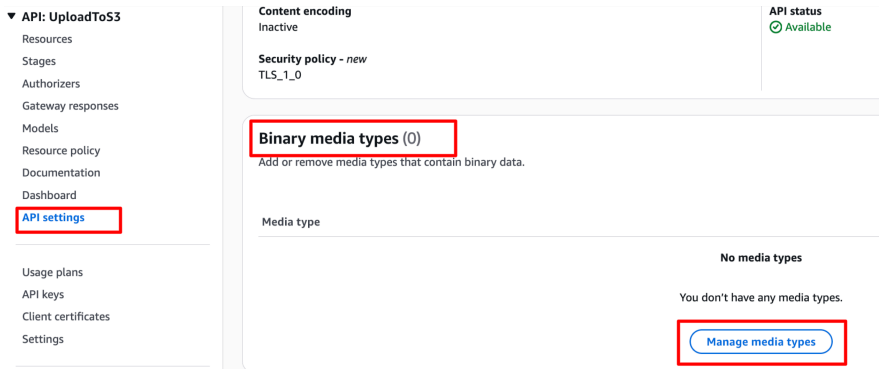

API Gateway > APIs > Resources - UploadToS3 (c8ql2n1704) > Create method
AWS subdomain
HTTP method
PUT
Action type
Use action name
Use path override
Path override
{bucket}/{filename}
Execution role
arn:aws:iam::
:role/apigw-s3-role
Credential cache
Do not add caller credentials to cache key
Content handling
Learn more
Passthrough
Integration timeout
Info
By default, you can enter an integration timeout of 50 - 29,000 milliseconds. You can use Service Quotas to raise the
28995

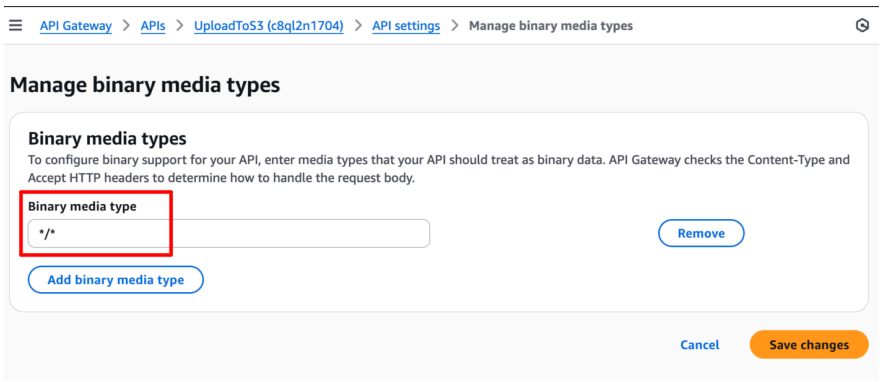
▼ PUT Method Integration:

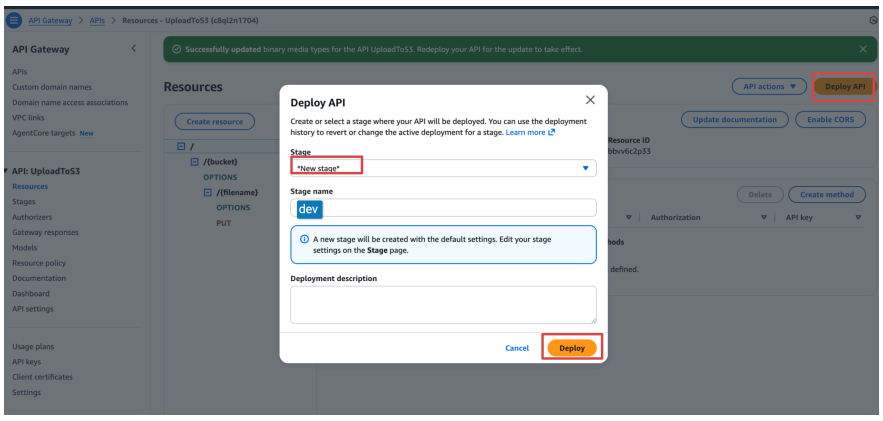


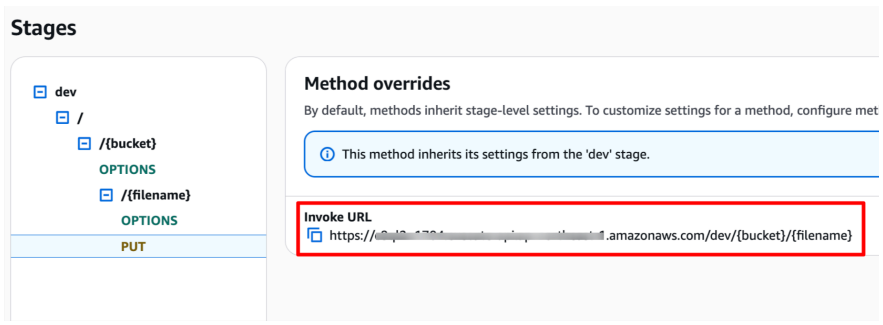


▼ Image : Binary Media Settings





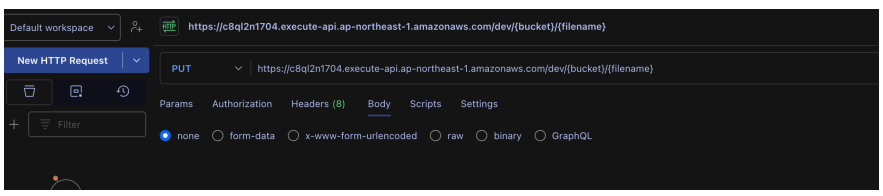




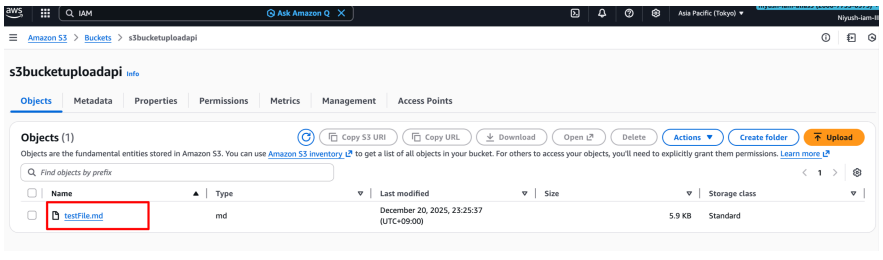
⌘ Step 4 : Testing the API via Postman

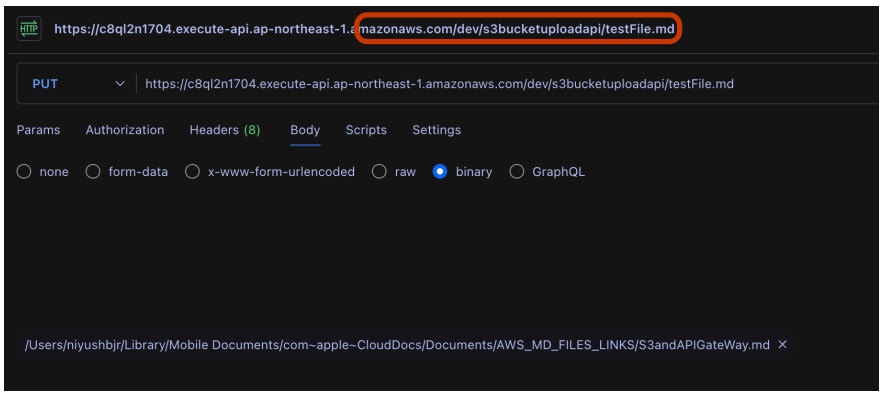
- Open Postman via app or VS Code. → `New Request` → `PUT`
- Replace `{bucket}` with your S3 Bucket name and `{filename}` with file you wish to upload.
- In postman, select `body` , choose `binary` and upload the file.
- If successfully uploaded it should return `Status code : 200`
- Also Check and confirm via AWS Console , whether the file has been uploaded to S3.

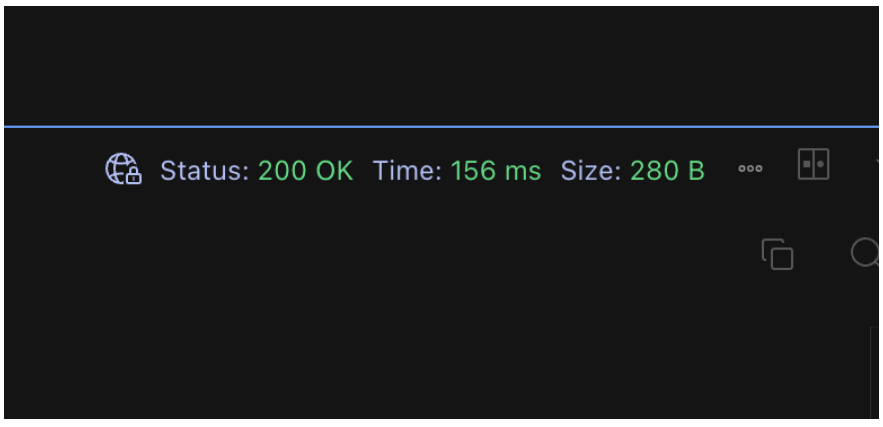
▼ POSt man test



▼ Checking the file upload confirmation via s3 console.

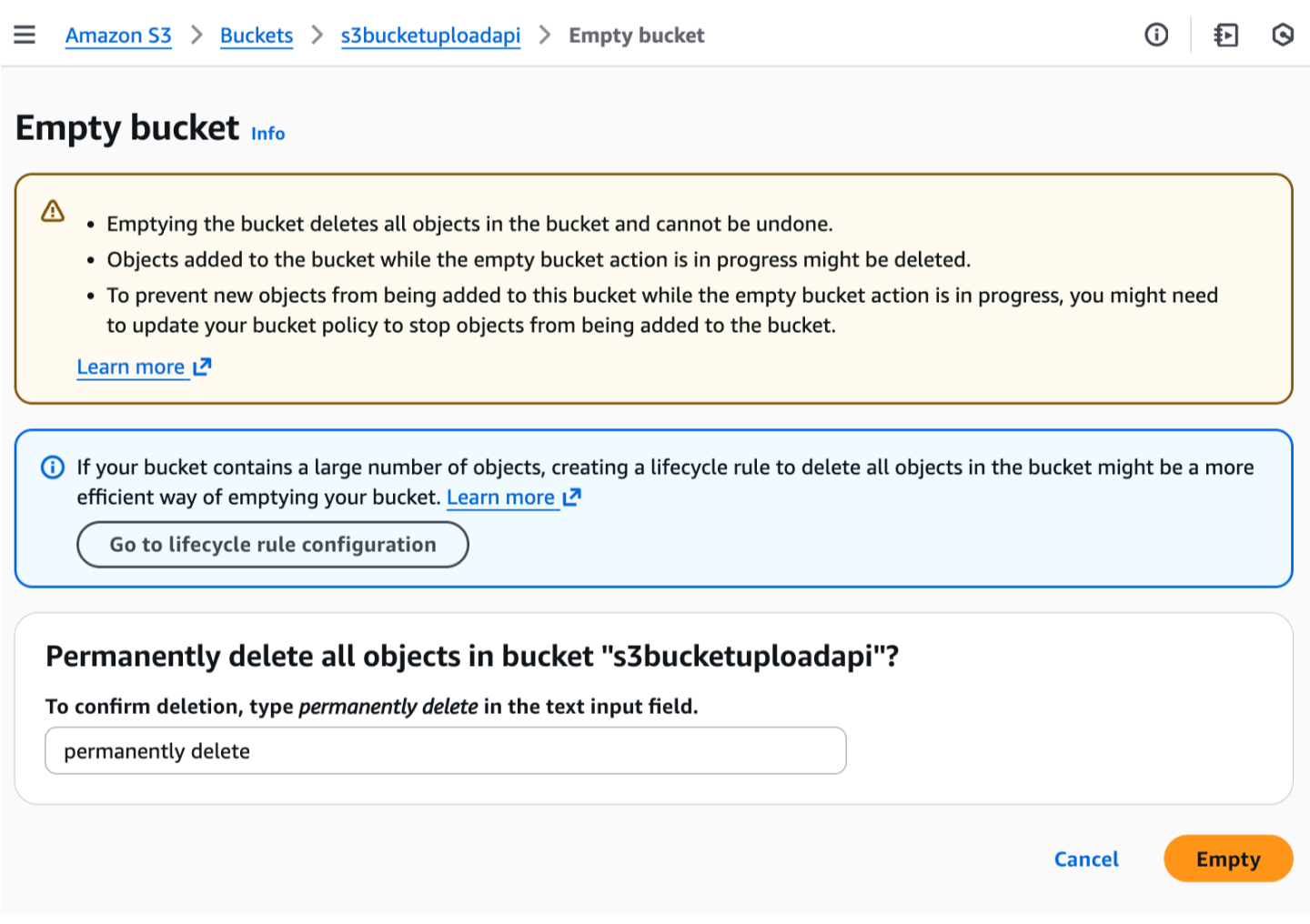


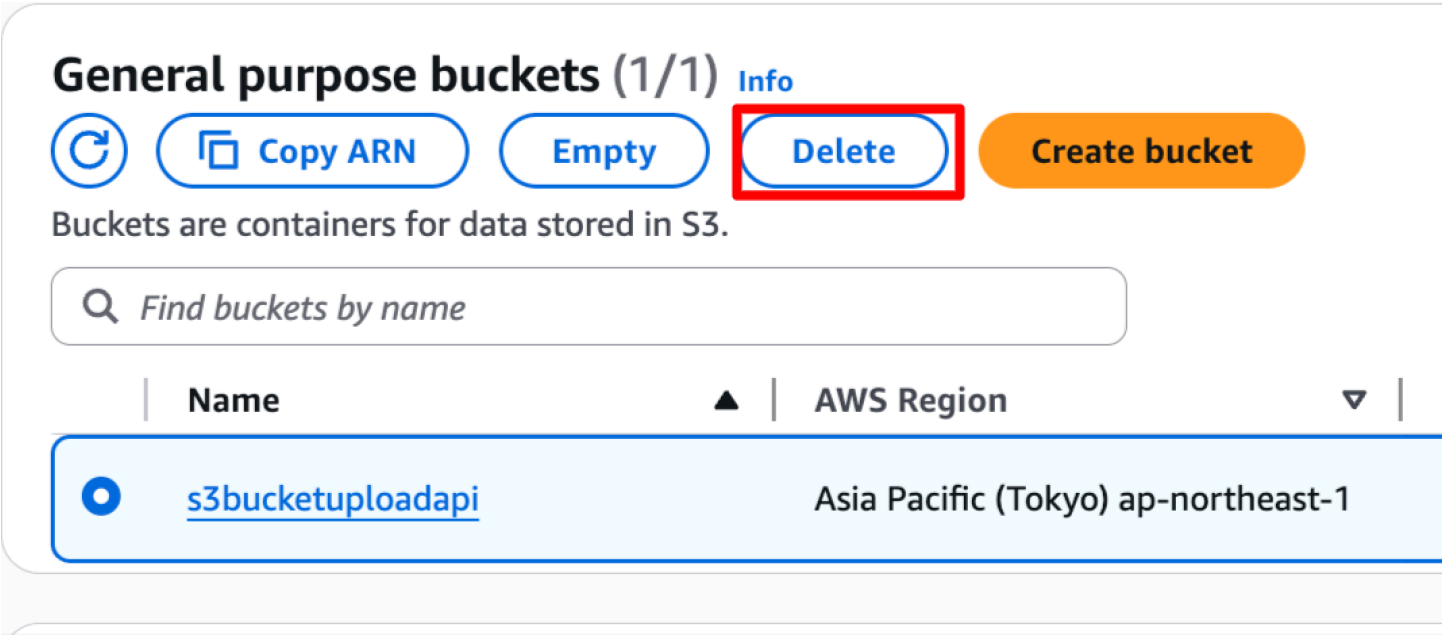


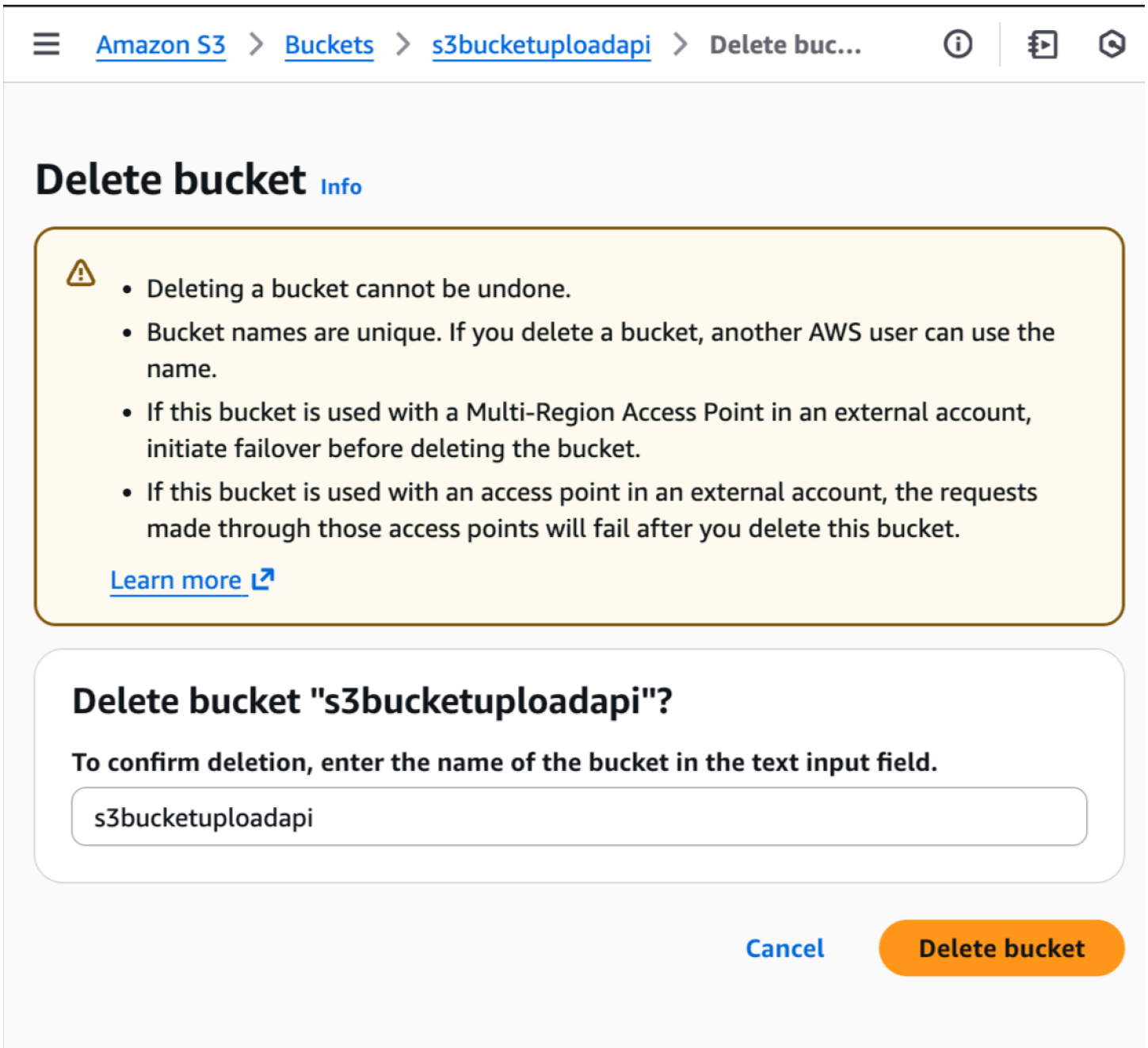


⌘ Step 5 : Deletion Time.

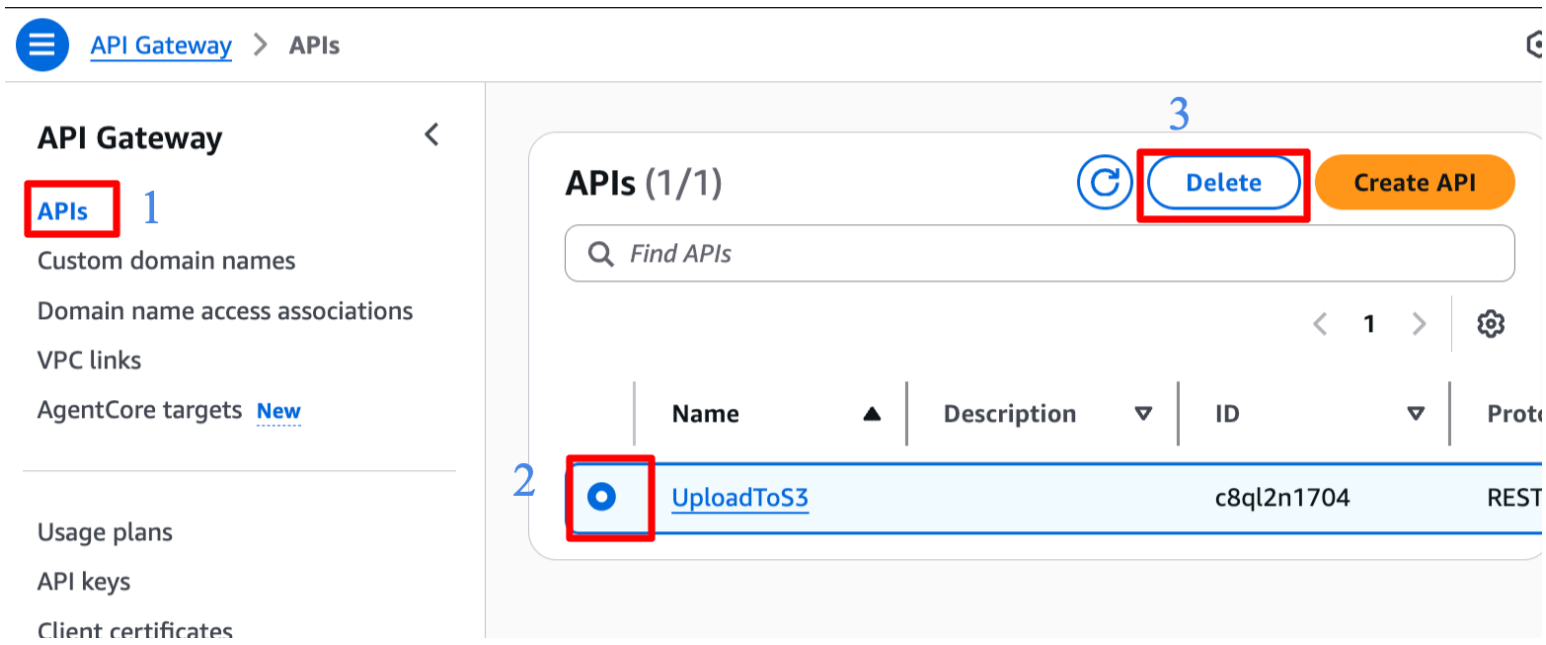
▼ Delete the S3 Bucket.



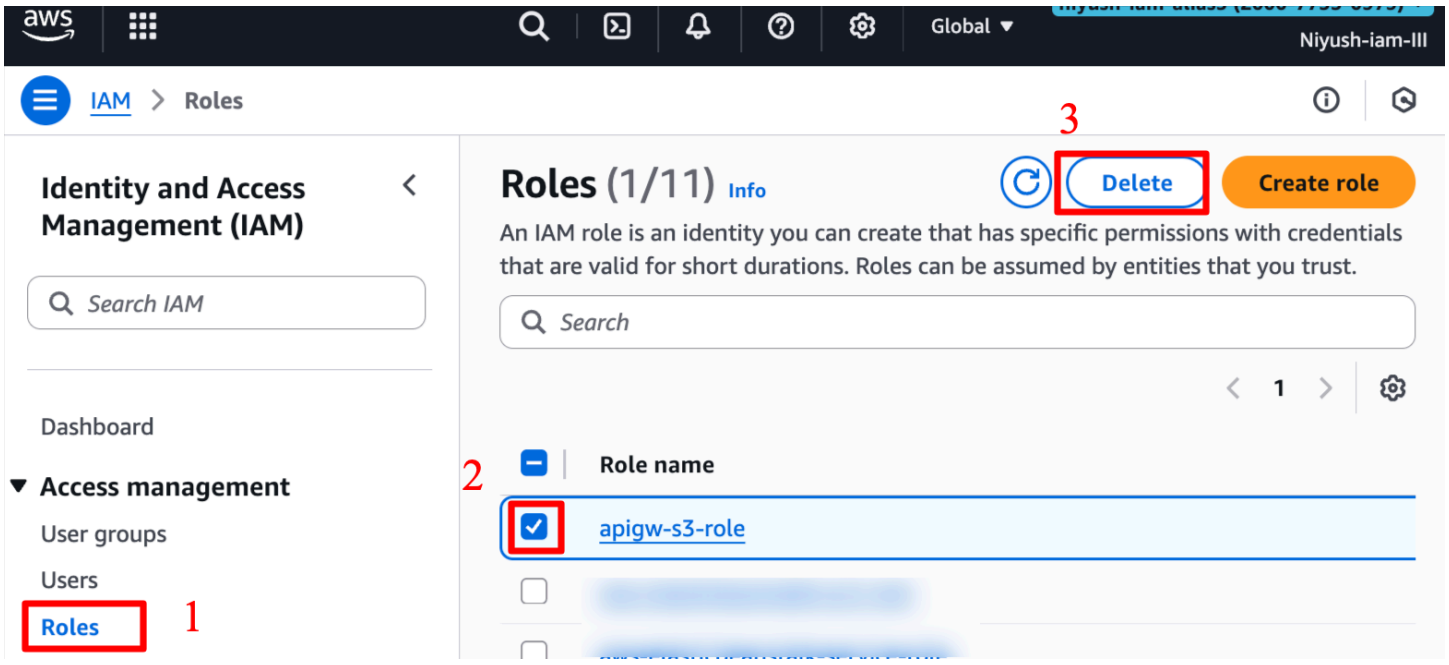




▼ Delete the API Gateway



▼ Delete the IAM Role (optional)



And that’s a wrap for this section.