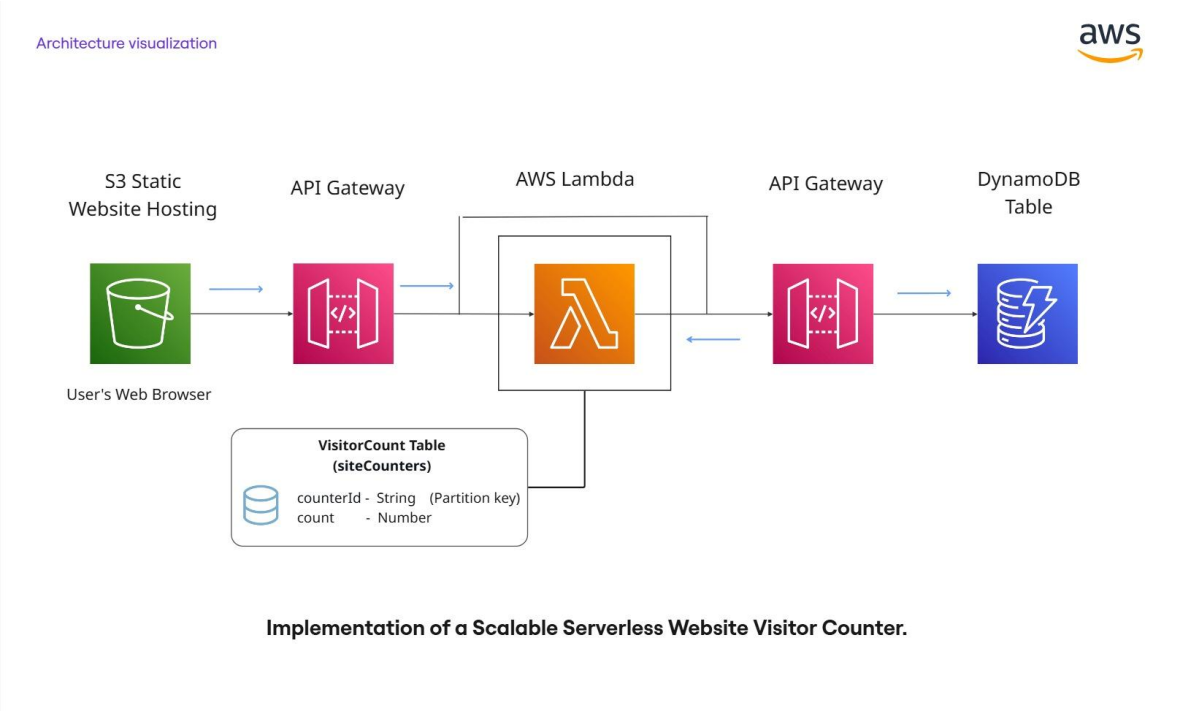
Implementation of a Scalable Serverless Website Visitor Counter.

AWS SERVICES USED:

1. AWS Lambda
2. API Gateway
3. DynamoDB
4. S3 (Simple Storage Service).

AWS ARCHITECTURAL DIAGRAM:



Implementation of a Scalable Serverless Website Visitor Counter.

- This diagram shows a **serverless website visitor counter architecture** using AWS services.

- A static site on S3 calls API Gateway, which triggers a Lambda function that reads and updates visitor counts stored in DynamoDB.

<u>OVERVIEW (ONE-LINE)</u>

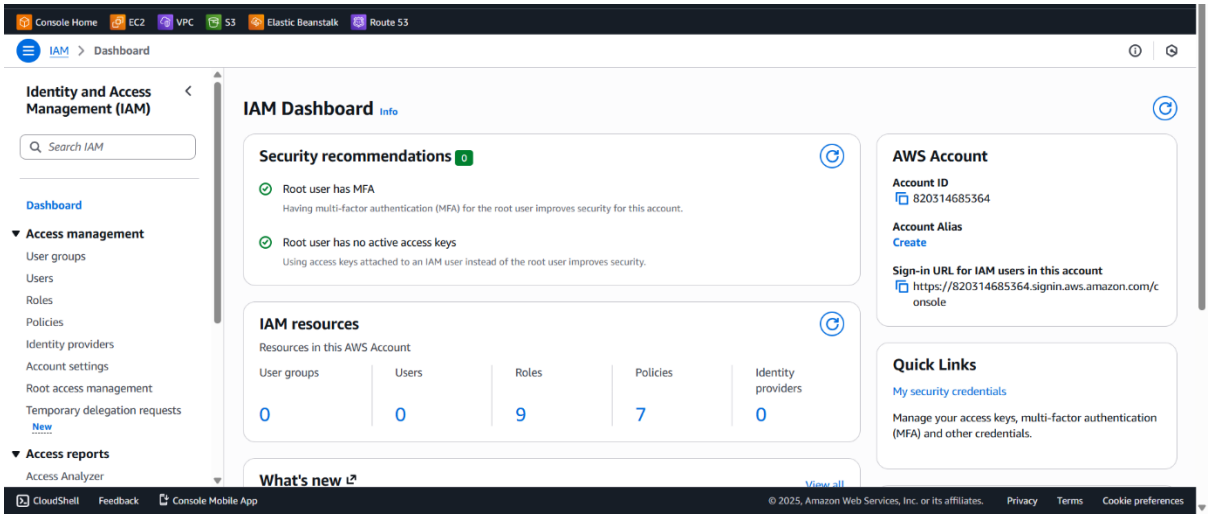
User visits site → browser `fetch()` hits API Gateway → API Gateway calls Lambda → Lambda atomically `ADD`'s 1 to a DynamoDB item and returns the new count → browser displays it.

→ Here, we will see step by step process of how this architecture works in detail.

AWS IAM:

- IAM - Identity and Access Management is a service that lets you securely manage access to AWS resources.

- It enables you to control who can do what in your AWS account through users, roles, and policies.



Step:1 - Create IAM role for Lambda:

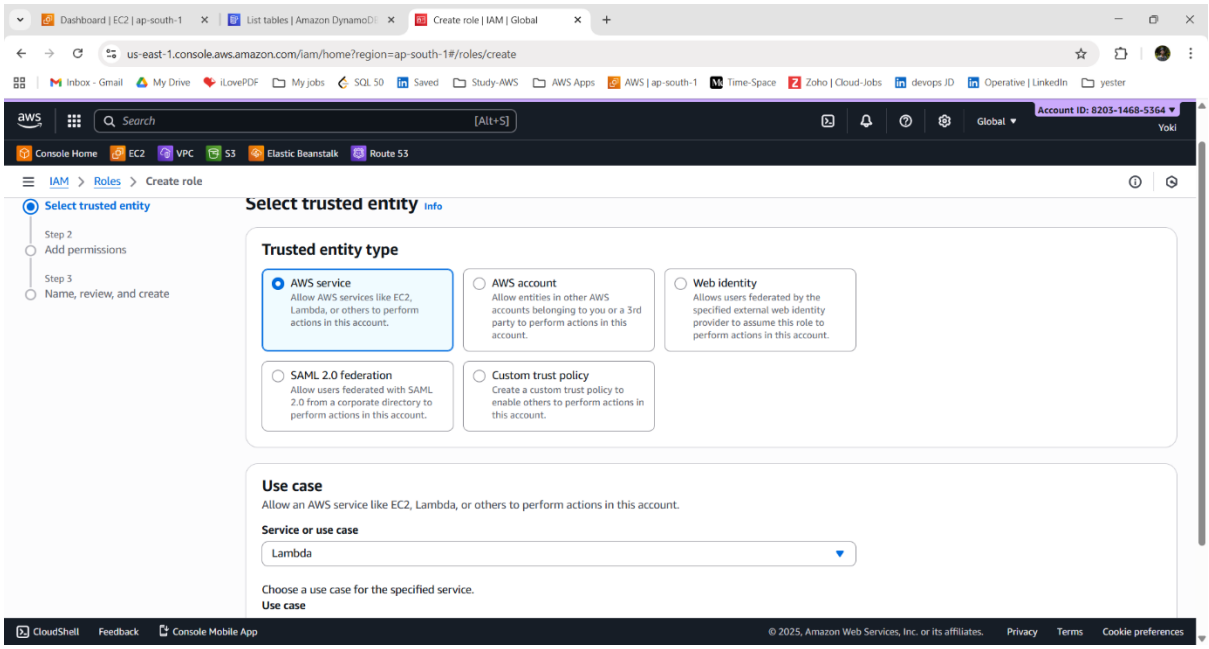
- Minimal policy — allows Lambda to update the table and write logs.

- IAM policy JSON (attach to role).

1. Choose trusted entity

    Select: **AWS service**

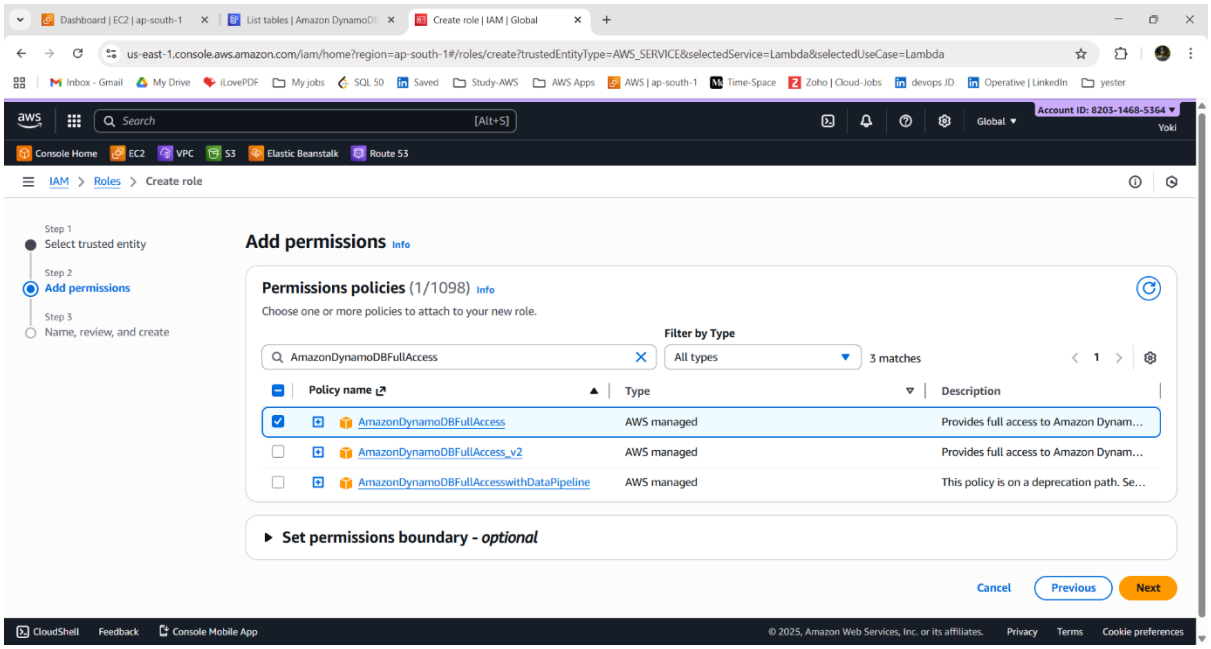
    Choose: **Lambda and** Click **Next**



2. Attach permissions: - Search for these policies

AmazonDynamoDBFullAccess - Lambda can update the counter

CloudWatchLogsFullAccess - Lambda can write logs

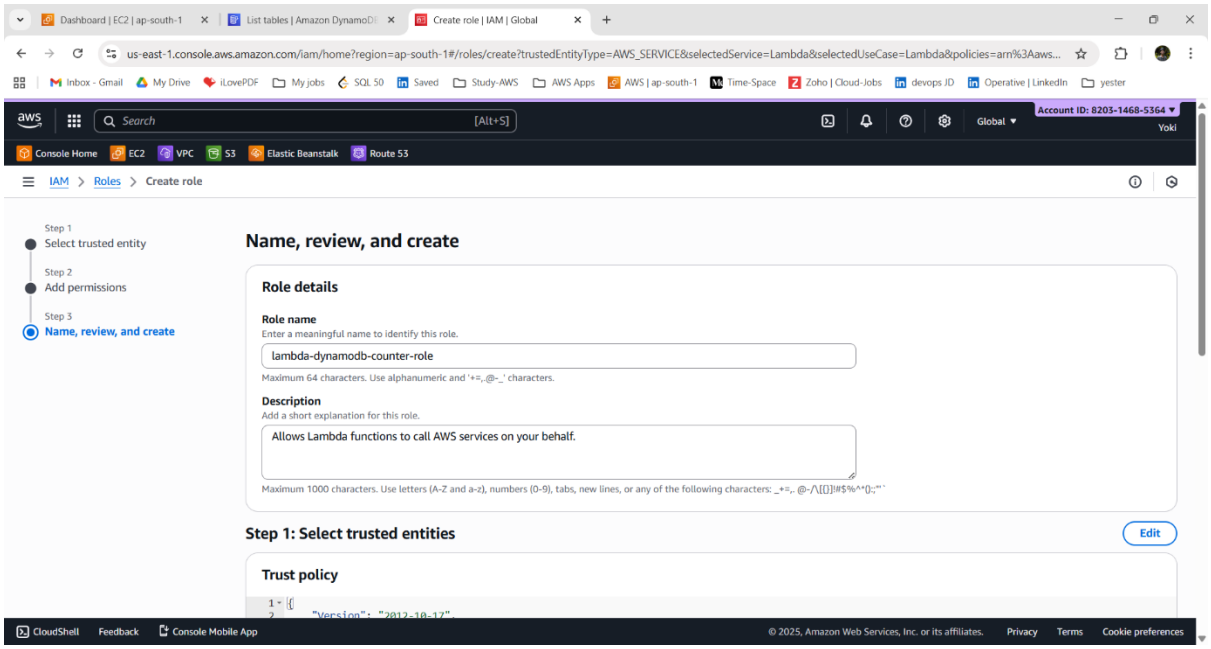


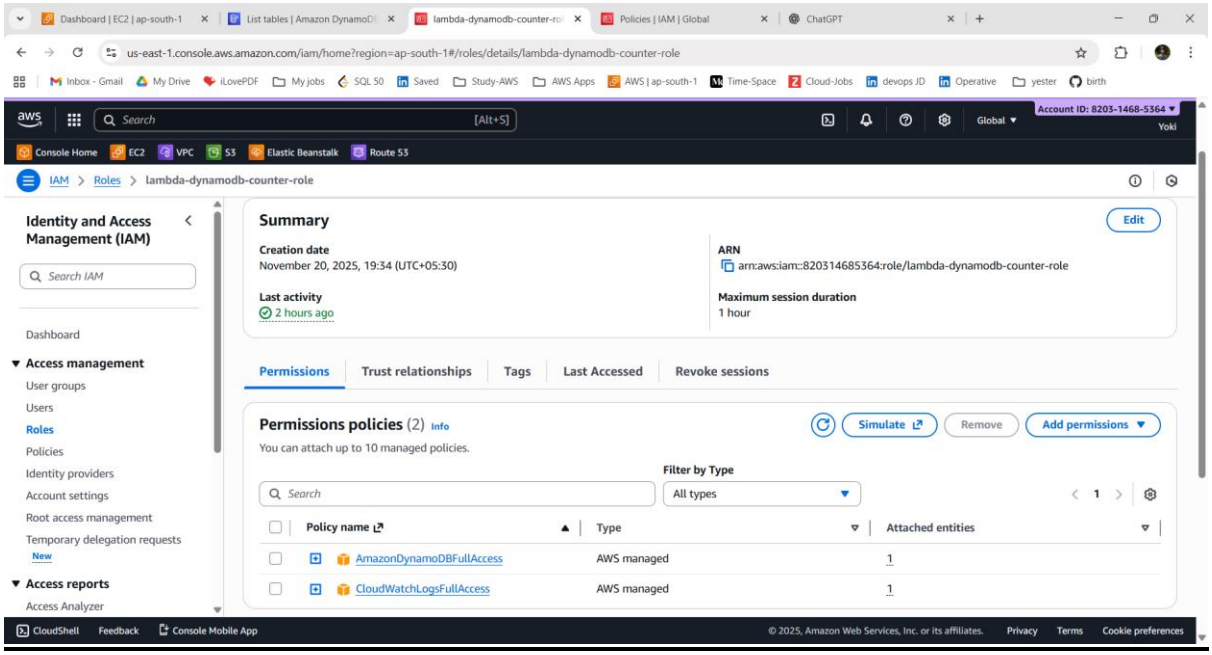
~ Click **Next**.

3. Name the role:

Role name: **lambda-dynamodb-counter-role**

- Click **"Create Role".**

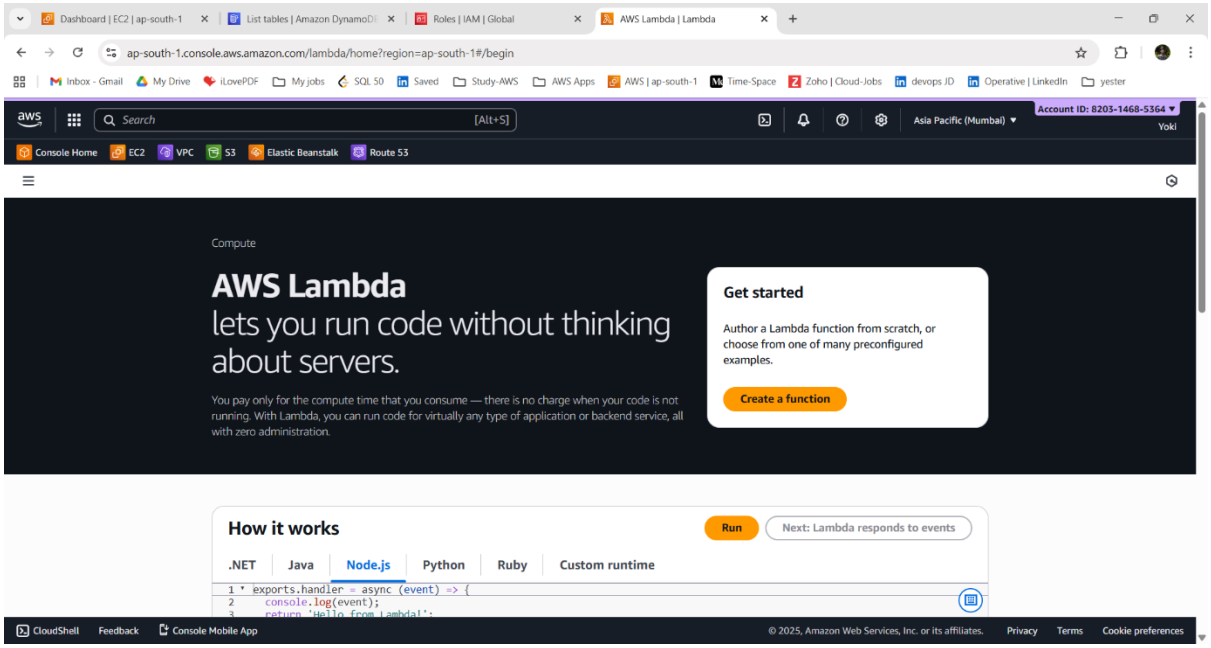




AWS Lambda:

- AWS Lambda is a serverless compute service that runs your code automatically in response to events without provisioning or managing servers.

- It lets you execute code on-demand and scale automatically, with billing based only on compute time used.



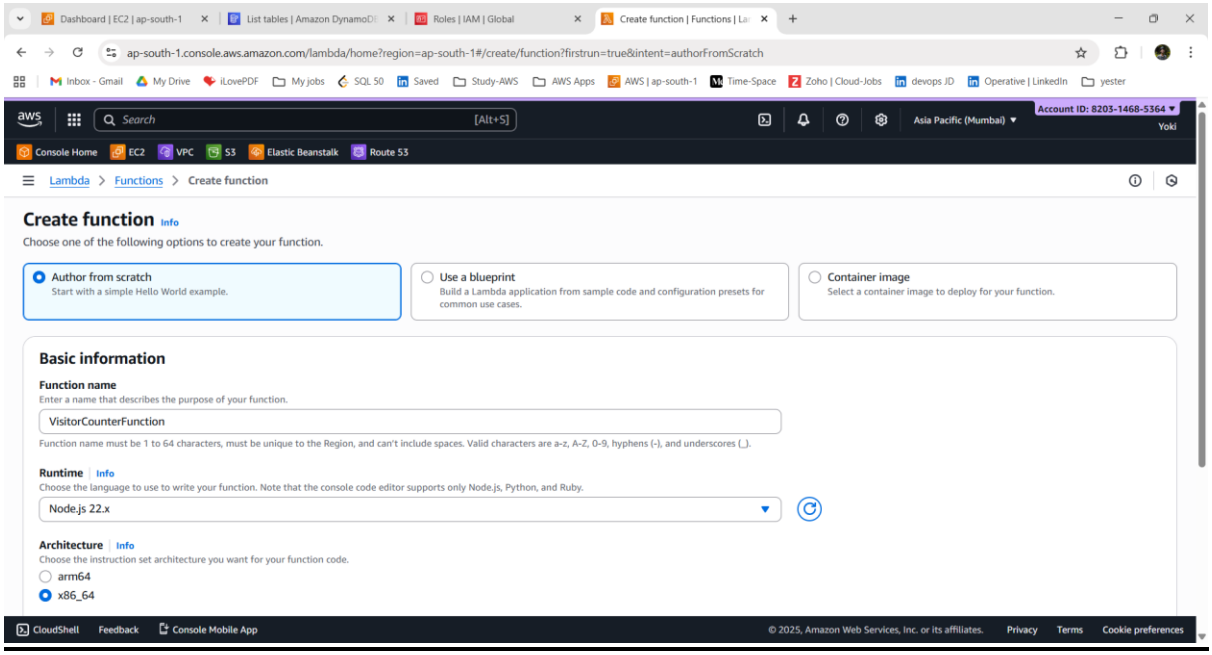
Step:2 - Create Lambda Function:

1. Create function.

Choose: **Author from scratch**

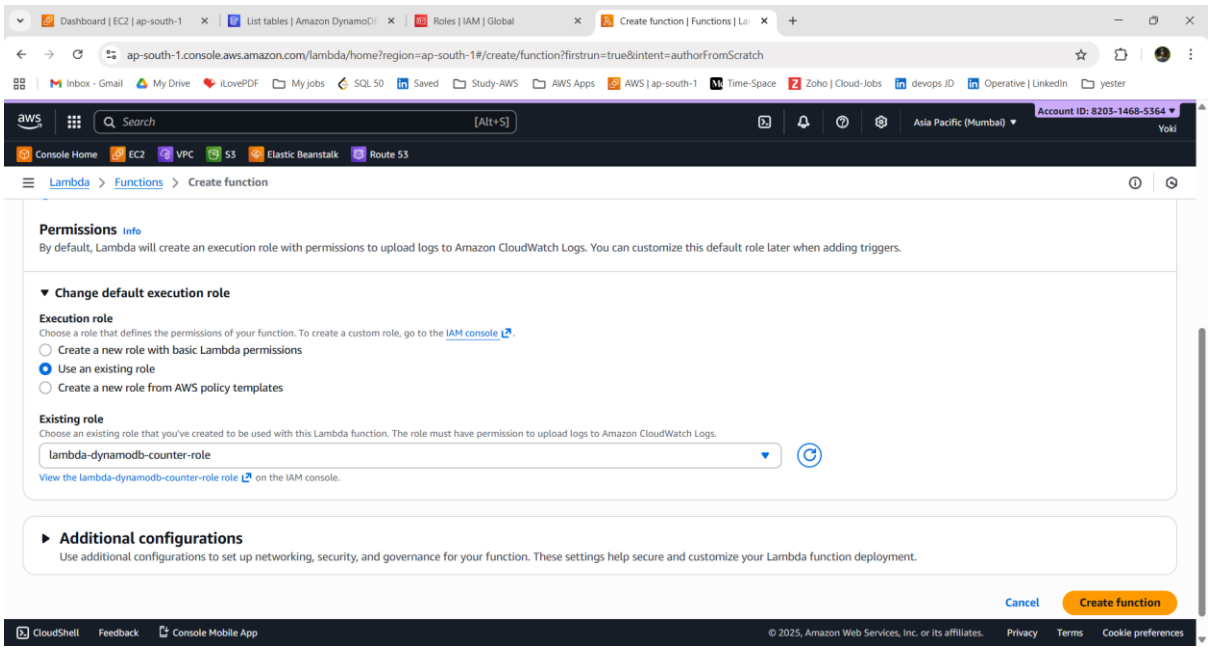
Function name: **VisitorCounterFunction**

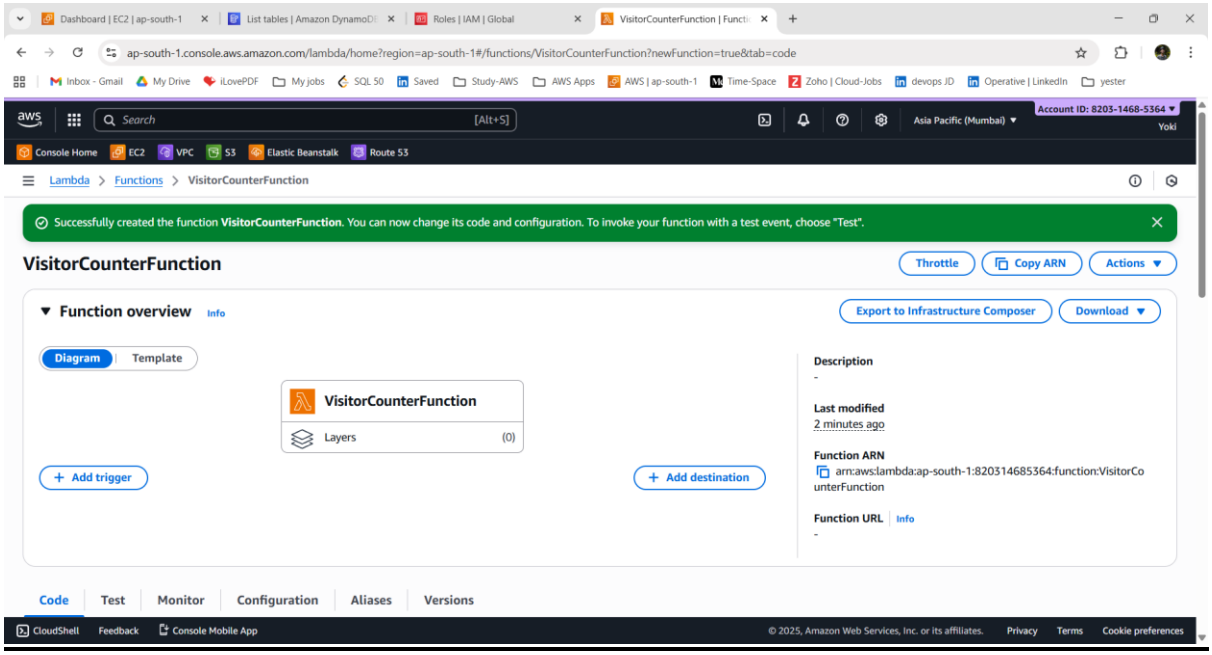
Runtime: **Node.js 22.x**



2. Change "**Execution Role**" section: Select Use an existing role

- Choose the role you created: **lambda-dynamodb-counter-role**
- Click **Create Function.**





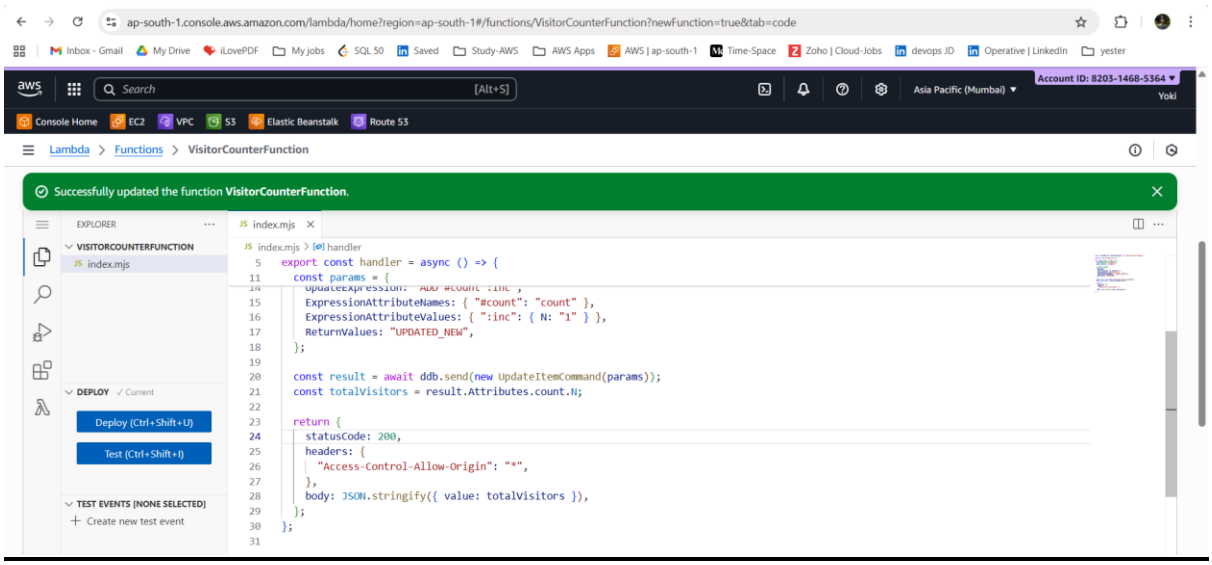
Step:3 - Add Lambda Code:

1. Go to → **"Code" section**
2. You will see a file called: **index.mjs (or index.js)**
3. Delete the default code.
4. Paste the below code.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
const ddb = new DynamoDBClient({});
export const handler = async () => {
  // DynamoDB table and counter ID
  const TableName = "siteCounters";
  const CounterId = "visitors";
  // Atomic increment
  const params = {
    TableName,
    Key: { counterId: { S: CounterId } },
    UpdateExpression: "ADD #count :inc",
    ExpressionAttributeNames: { "#count": "count" },
    ExpressionAttributeValues: { ":inc": { N: "1" } },
    ReturnValues: "UPDATED_NEW",   };
  const result = await ddb.send(new UpdateItemCommand(params));
  const totalVisitors = result.Attributes.count.N;
```

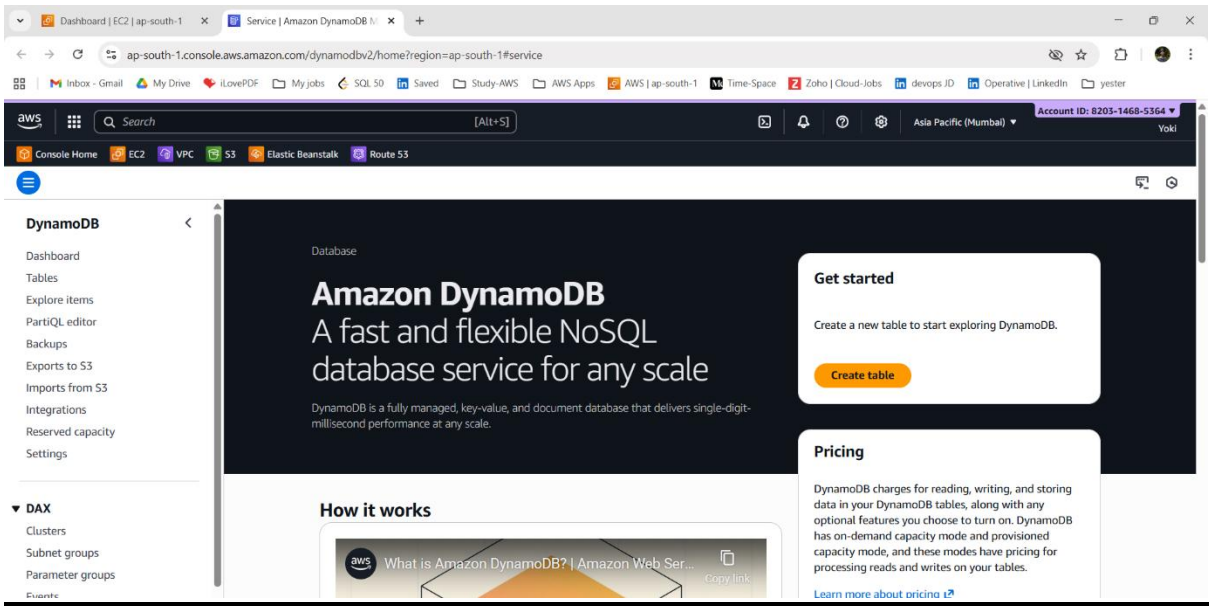
```
  return {
    statusCode: 200,
    headers: {       "Access-Control-Allow-Origin": "*",     },
    body: JSON.stringify({ value: totalVisitors }),
  };
};
```

5. Click **“Deploy”.**



AMAZON DynamoDB:

- Amazon DynamoDB is a fully managed NoSQL database service from AWS that provides fast, predictable performance at any scale. It offers key-value and document data storage with built-in availability, security, and automatic scaling.

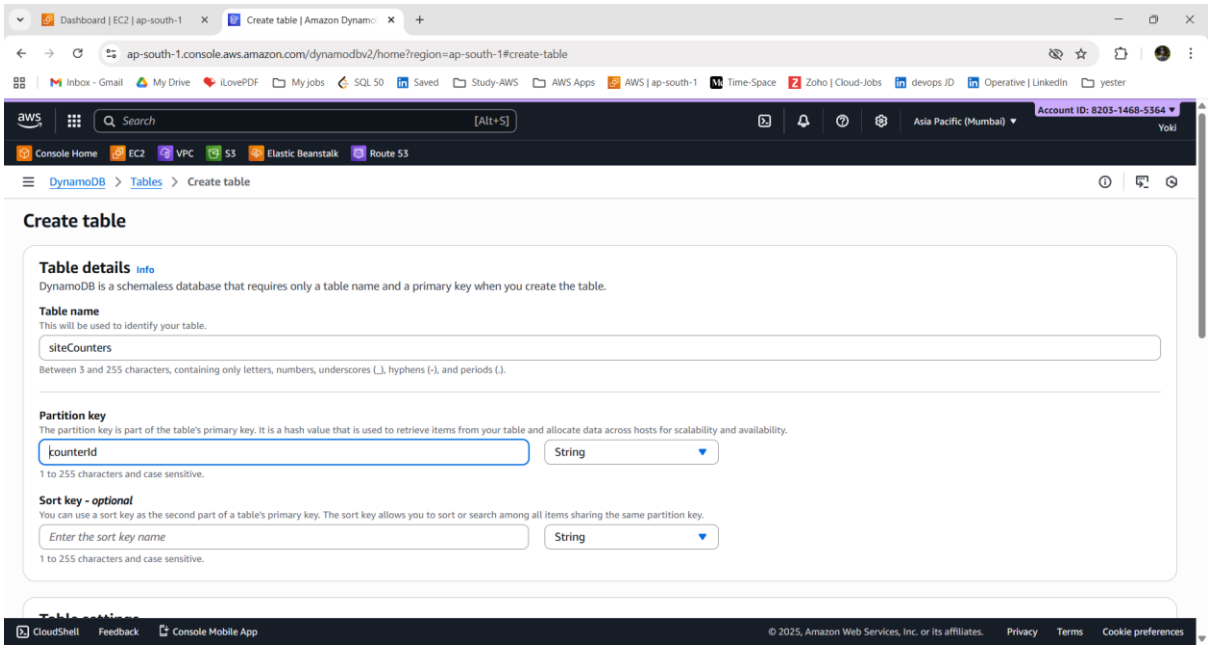


Step:4 - Create DynamoDB table:

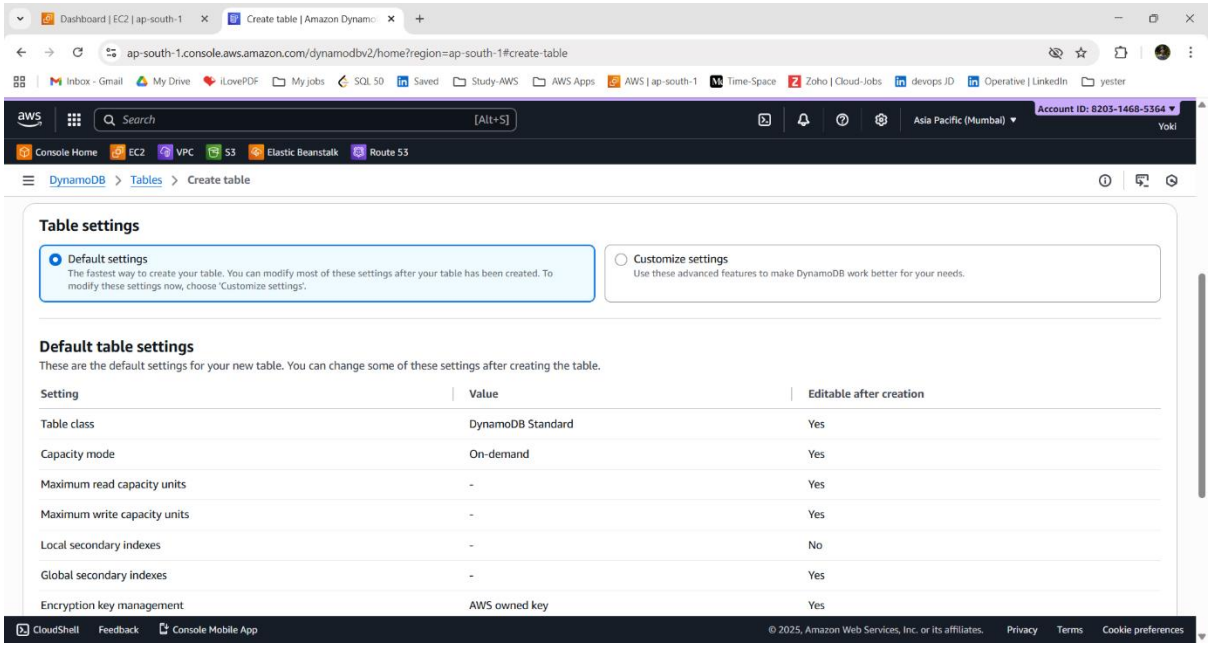
1. Go to DynamoDB → Create table.

Table name: siteCounters

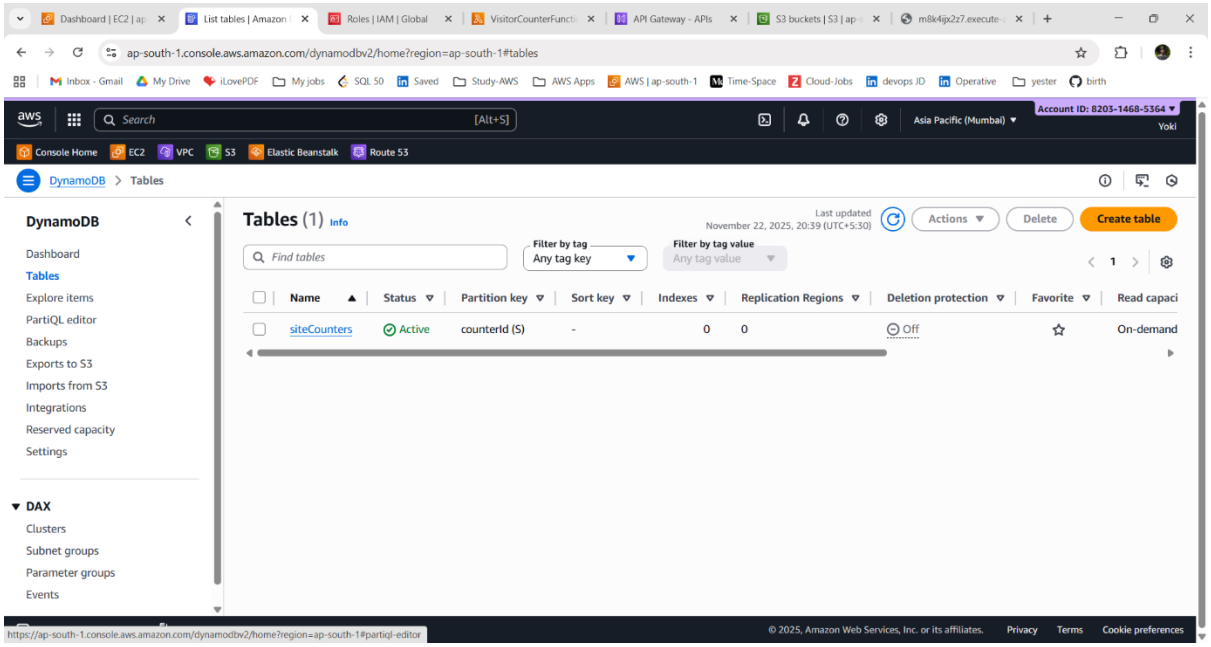
Partition key: counterId (String)



2. Billing mode: Capacity mode = **On-demand**



3. click **“Create table”.**



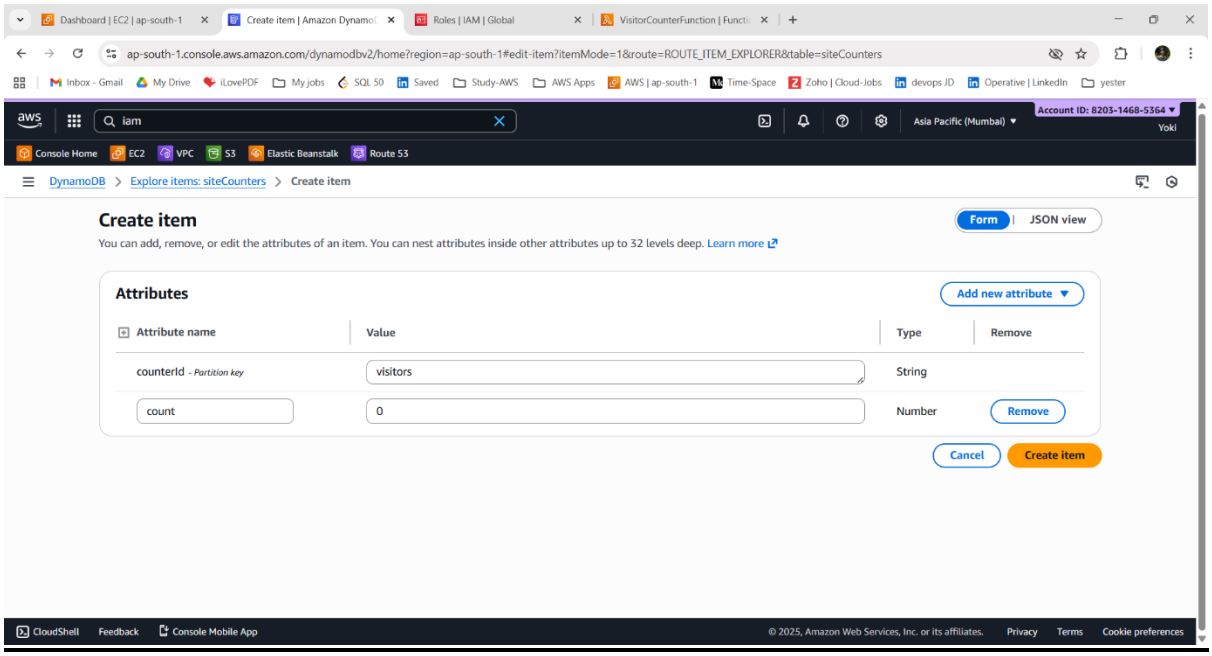
4. Add the first item.

5. Go to: Items → Create item

counterId (Partition key) : visitors - string

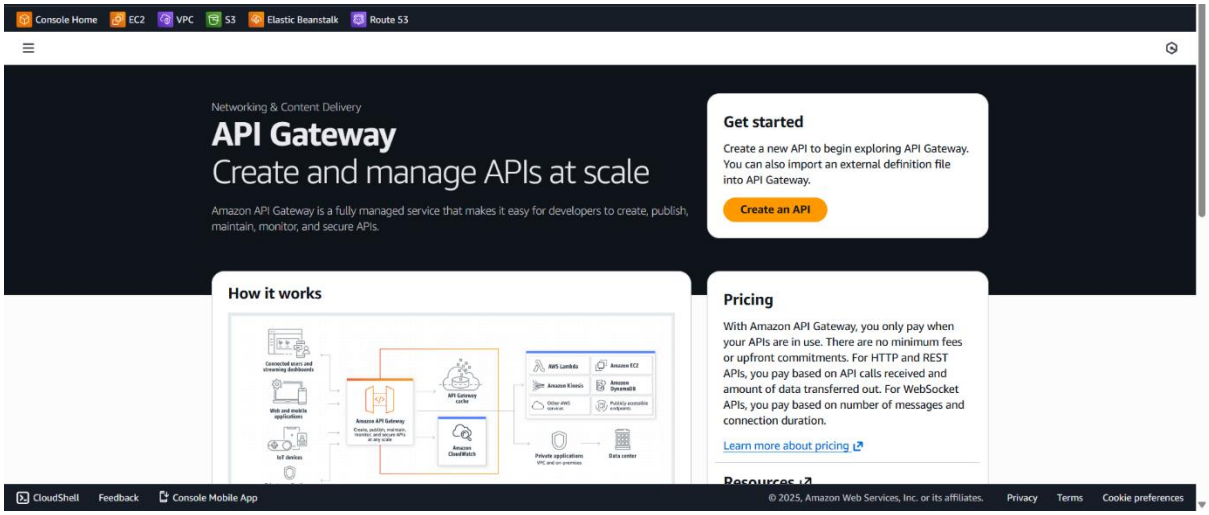
count : 0 - number

6. Click **“Create item”.**



API GATEWAY:

- It is a fully managed service that lets you create, publish, secure, and monitor APIs at any scale.
- It acts as a front door for applications to access backend services like Lambda, EC2, or microservices.



Step:5- Create API Gateway:

1. Go to API Gateway → Create API.
2. enter API details:

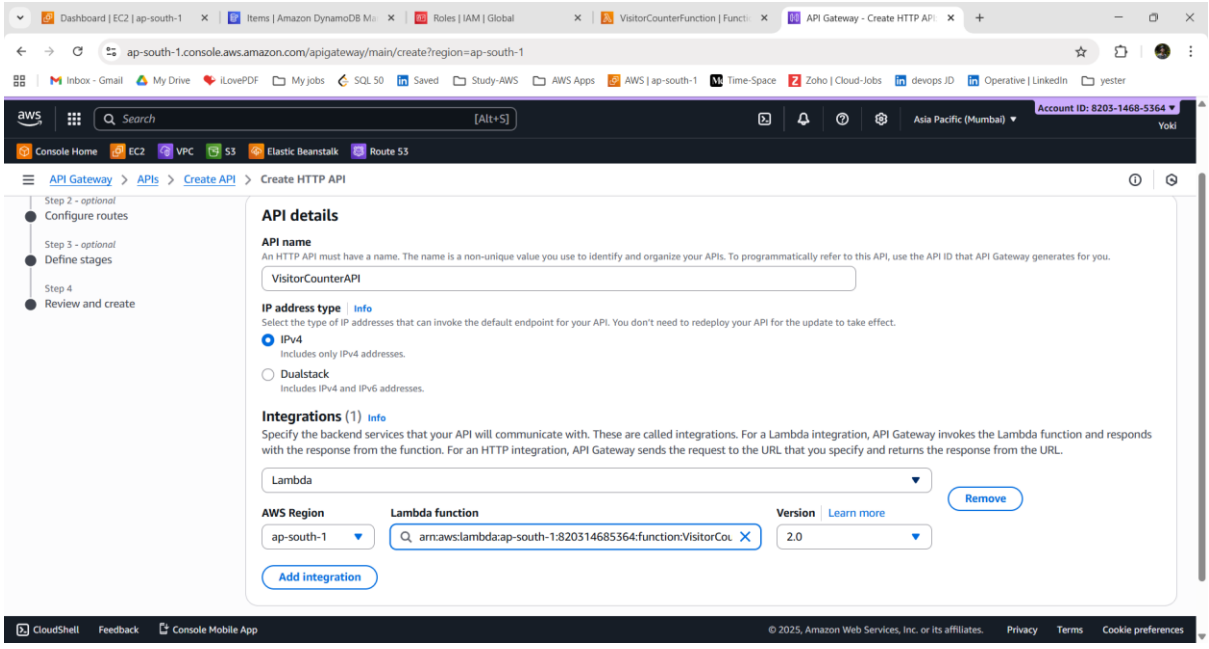
   API name: **VisitorCounterAPI**

   IP Address: IPv4

3. Integrations → Choose: **Lambda**

   AWS Region: **ap-south-1**

   Lambda Function: **arn:aws:lambda-functionVisitorCounter**



5. Configure routes:

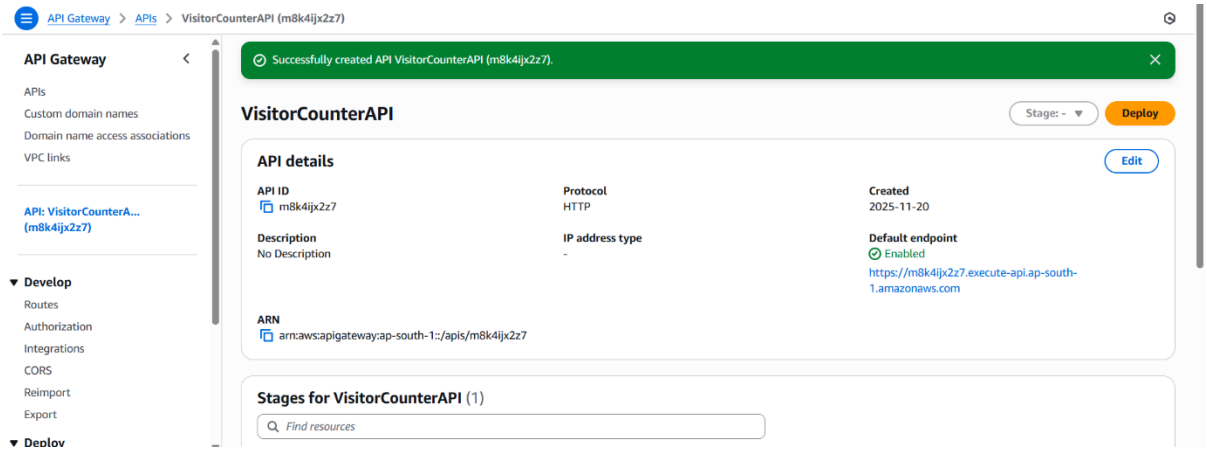
Method: **GET** Path

Resource path: **/count**

Integration Target: **VisitorCounterFunction.**

6. Define stages = default

7. Review and create.



Step:6- Copy Endpoint:

1. In your API → go to Stages → Select default.

2. Copy the Invoke URL.

e.g., https://abcdedf.execute-api.ap-south -1.amazonaws.com .

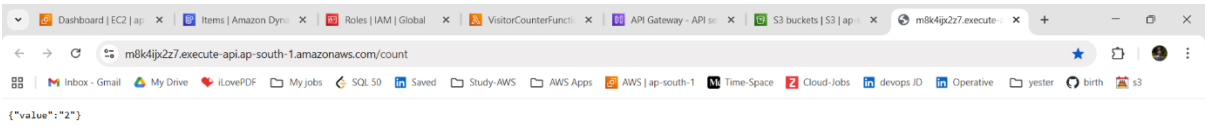
3. Open your browser/Postman and test:

4. And add the route path **"/count".**

https://m8k4ijx2z7.execute-api.ap-south-1.amazonaws.com/count

→ You should see something like:

{"value":"2"}



OUTCOME:

Here's an outcome description of the project *before hosting the website on S3*, based on the steps listed:

~ The project successfully set up a scalable backend system by creating an IAM role, building a Lambda function, and adding the necessary code to process visitor counts.

~ A database table was created to store visit data, and APIs were configured to allow secure access to the counter.

~ The API endpoint was generated and ready to be integrated into a frontend or S3-hosted static website.

STATIC WEBSITE HOSTING ON AMAZON S3:

~ project-name: **s3-dynamic-visitor-counter**

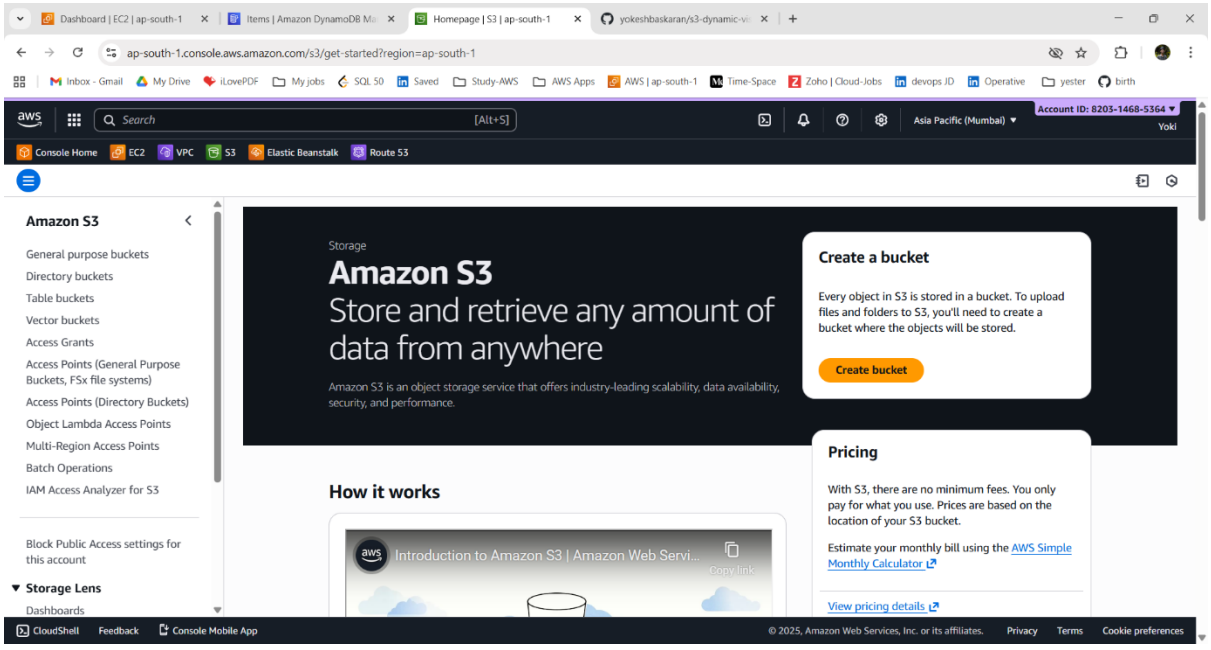
- This document describes how to host a static website using Amazon S3's built-in static hosting feature.

- This approach is suitable for simple static sites that do not require HTTPS, backend logic, authentication, or private hosting.

S3 STATIC HOSTING:

- S3 static hosting in AWS is a feature that lets you host static websites (HTML, CSS, JavaScript, images) directly from an Amazon S3 bucket without needing a web server.

- It serves your site via HTTP/HTTPS with high availability and low cost.



Here, we will see how to host a static website using Amazon S3's built-in static hosting feature. This approach is suitable for simple static sites that do not require HTTPS, backend logic, authentication, or private hosting.

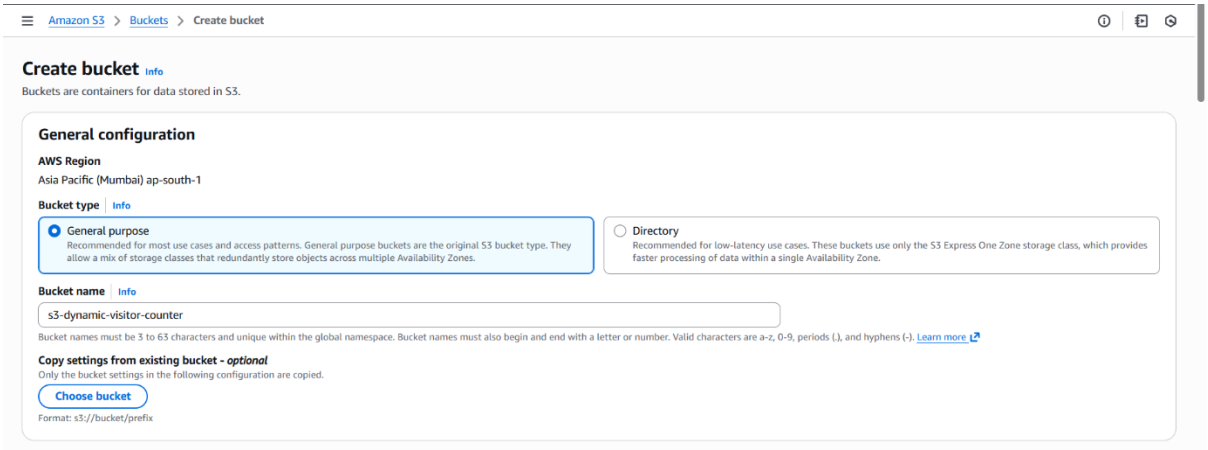
~ Go → AWS Management Console and Open the S3 service.

1. Create an S3 Bucket:

i) Choose **"Create bucket".**

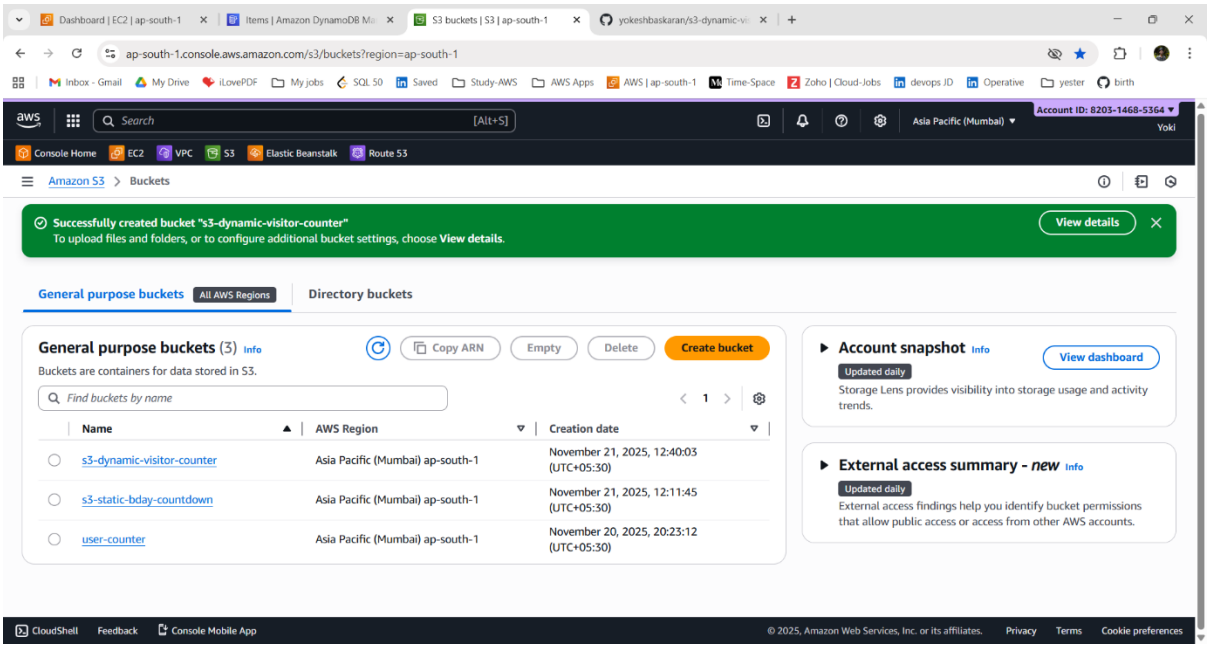
ii) Enter the bucket name (must be globally unique).

Bucket-name: **s3-dynamic-visitor-counter**



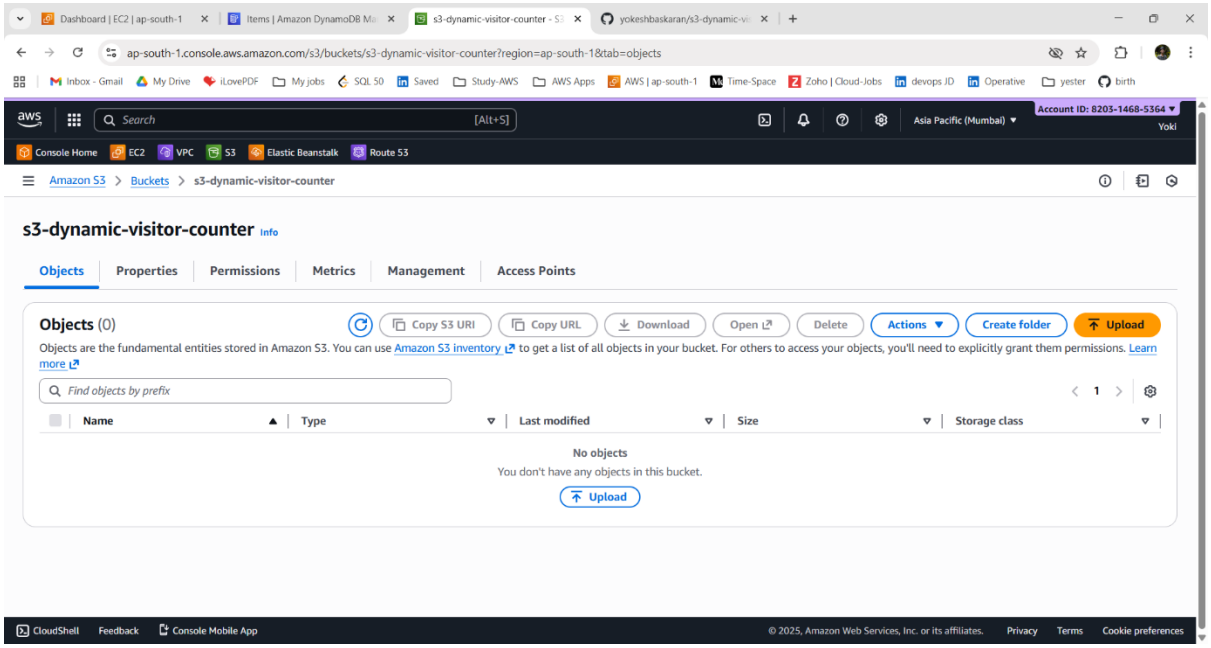
iii) Select an AWS Region - **ap-south-1**

iv) **Create the bucket.**



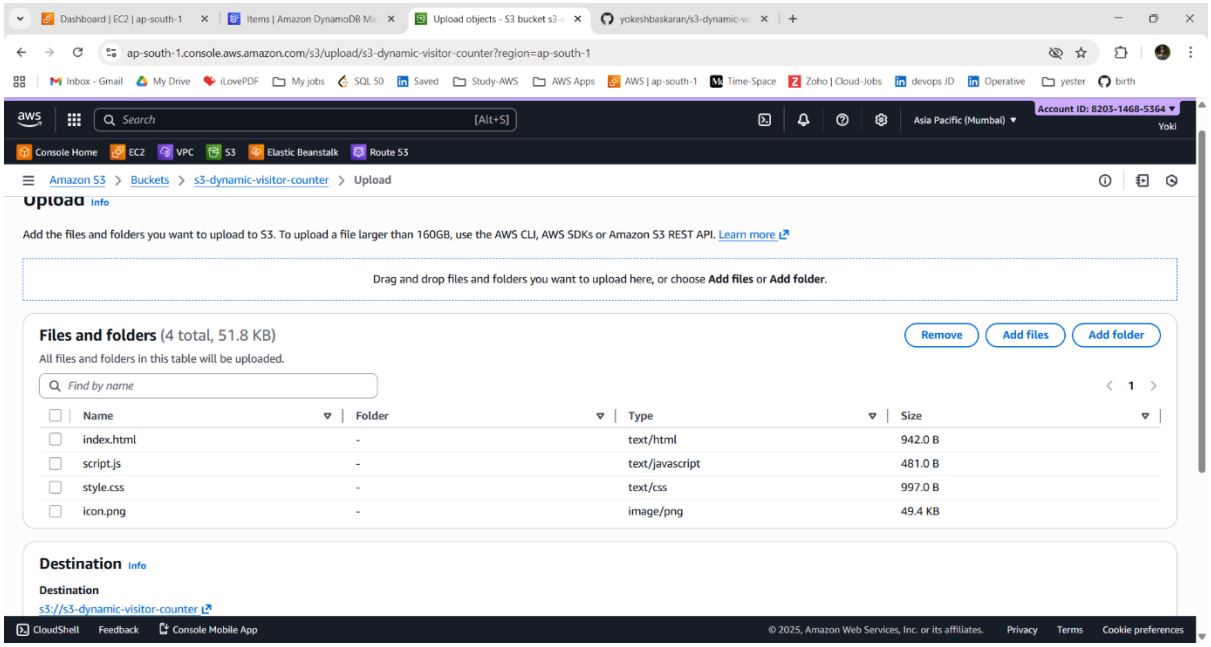
2. Upload Your Website Files:

i) Open the s3-dynamic-visitor-counter bucket.

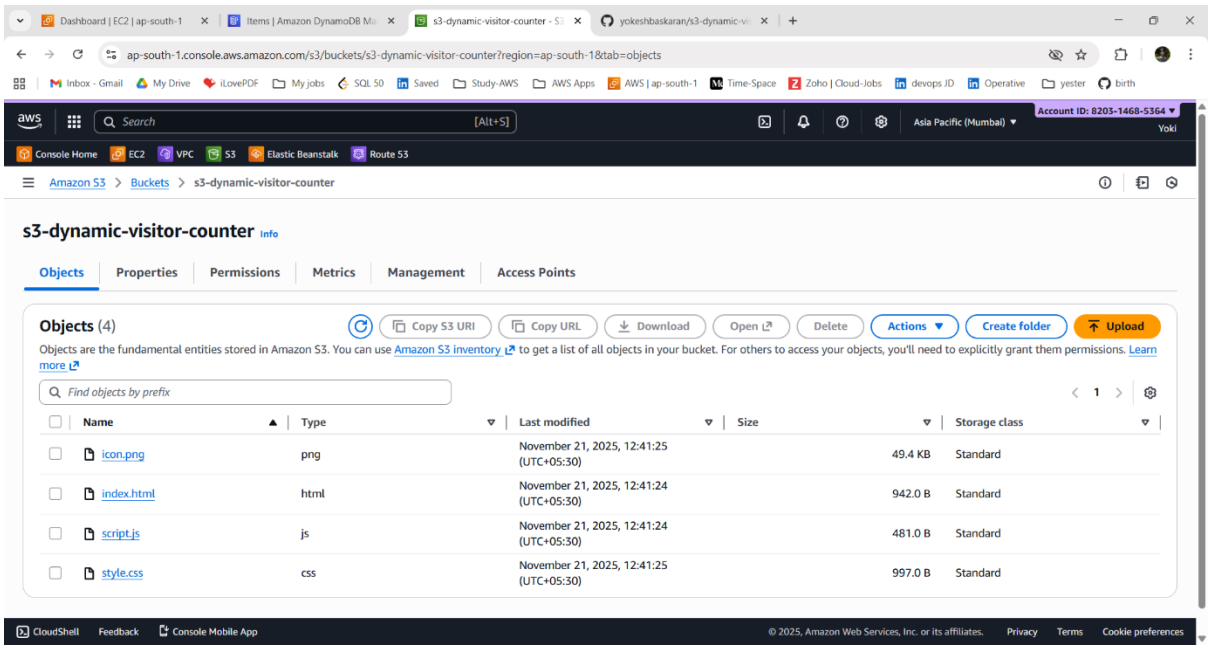


ii) Choose **Upload** to upload the files.

iii) Add all files related to your website (e.g., index.html, error.html, CSS, JS files).



iv) Choose **Upload** again to complete the process.



3. Enable Static Website Hosting:

i) Go to the **"Properties"** tab of the bucket.

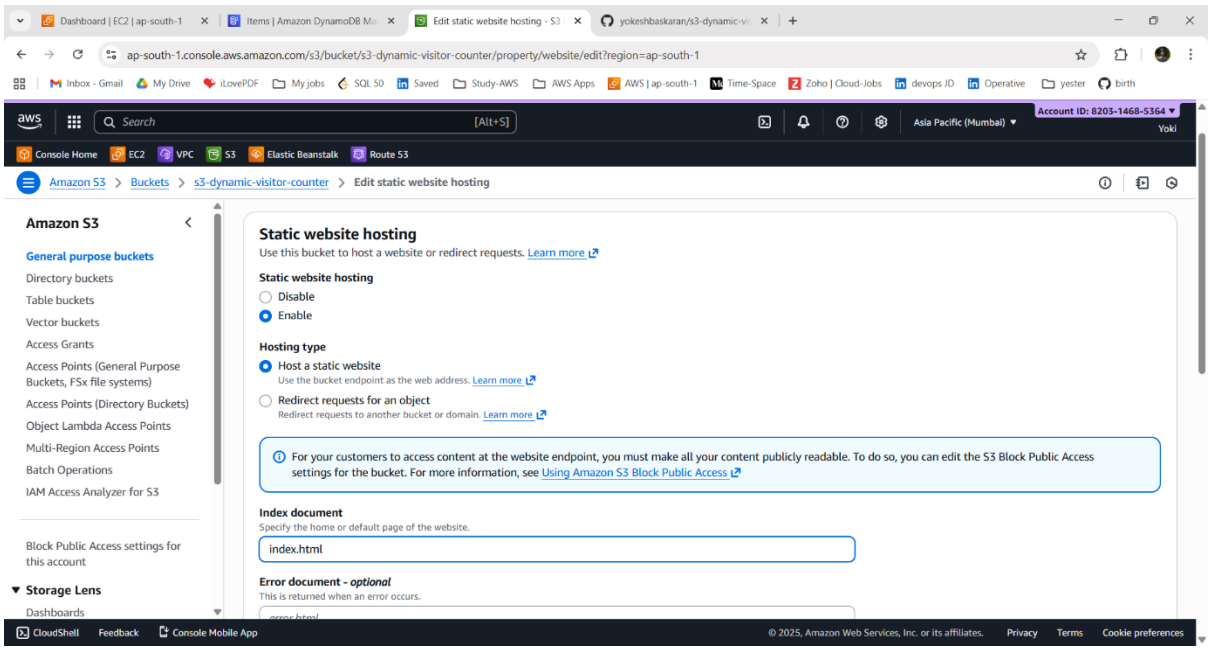
ii) Scroll to **Static website hosting.**

iii) Choose **Edit** and then Select **Enable**.

iv) Choose Host a static website.

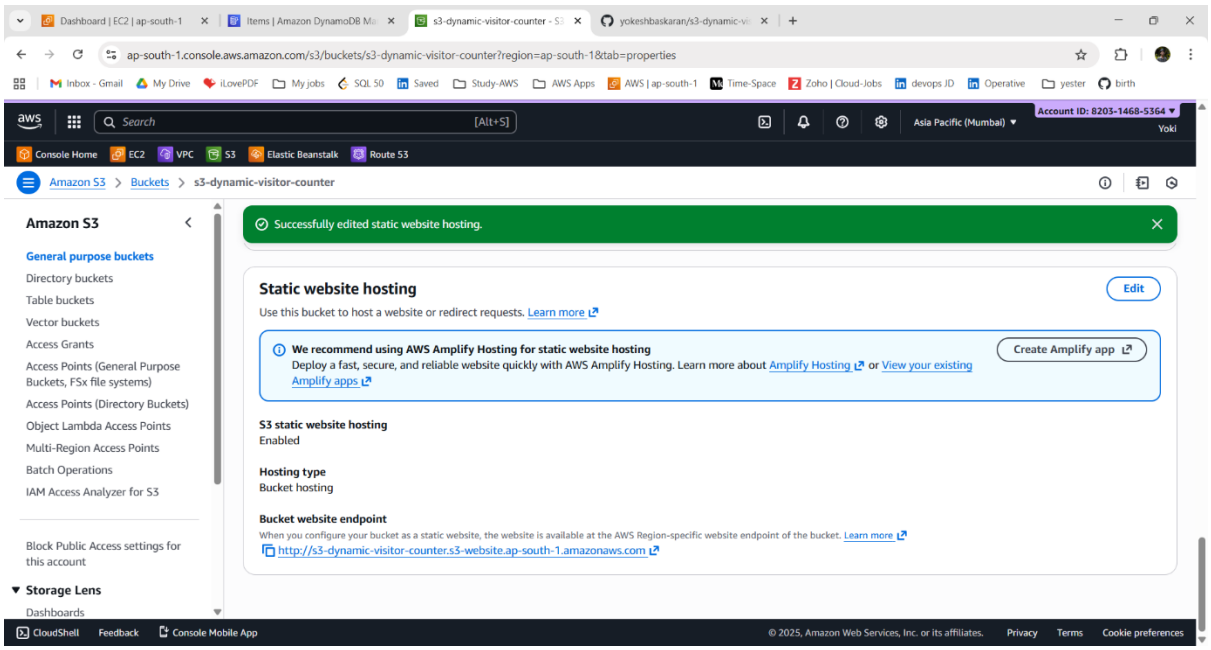
v) Set your Index document (e.g., **index.html**).

vi) click **Save changes.**

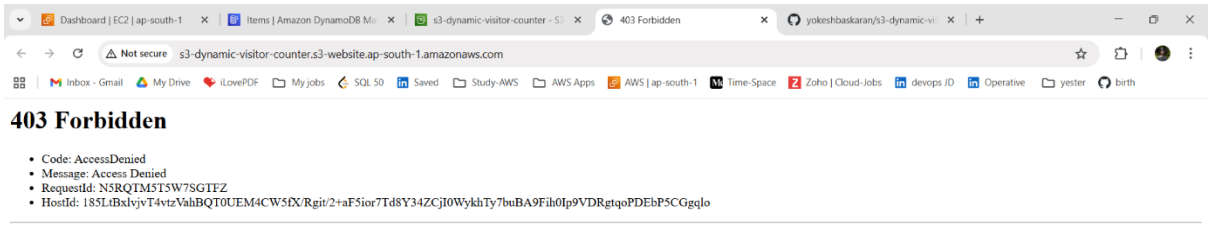


vii) Copy the Website endpoint shown after saving — this is your public website URL.

URL: http://s3-dynamic-visitor-counter.s3-website.ap-south-1.amazonaws.com



~ If we get 403 Forbidden or access denied. We need to check the bucket settings.



~ we need to **Turn off "Block Public Access"** (bucket settings).

~ Also, give the **Bucket policy.**

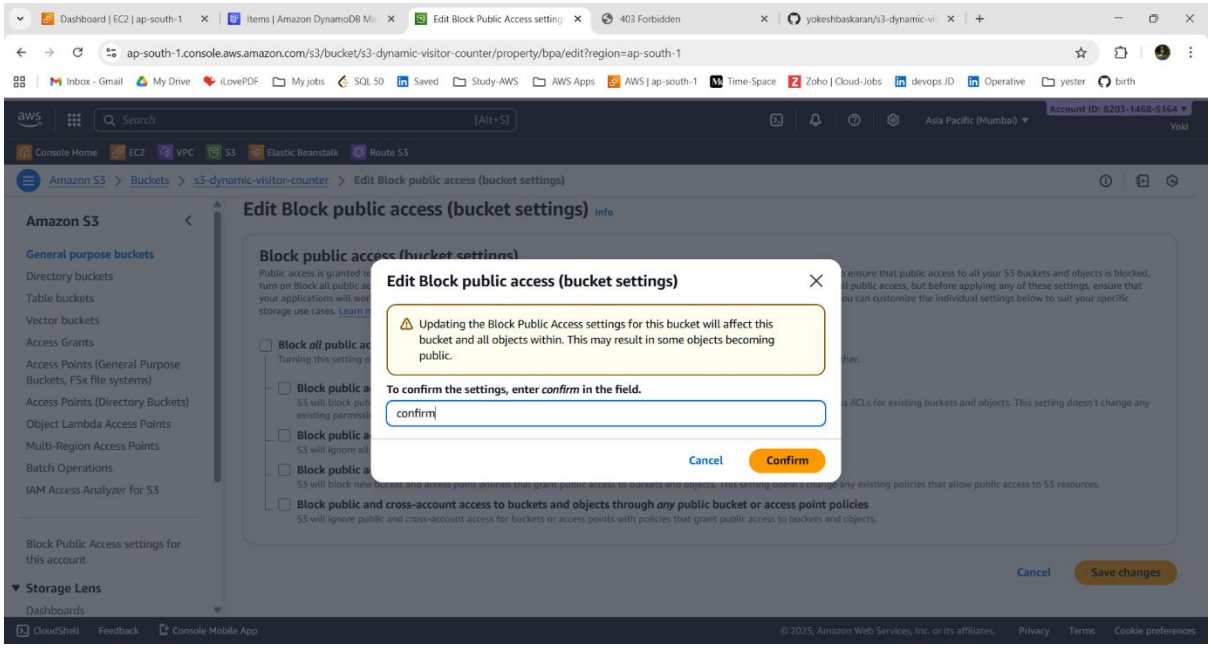
4. Allow Public Access to Website Files and add Bucket Policy:

i) Go to the **"Permissions"** tab of the bucket.

ii) Scroll to **"Block Public Access"** (bucket settings).

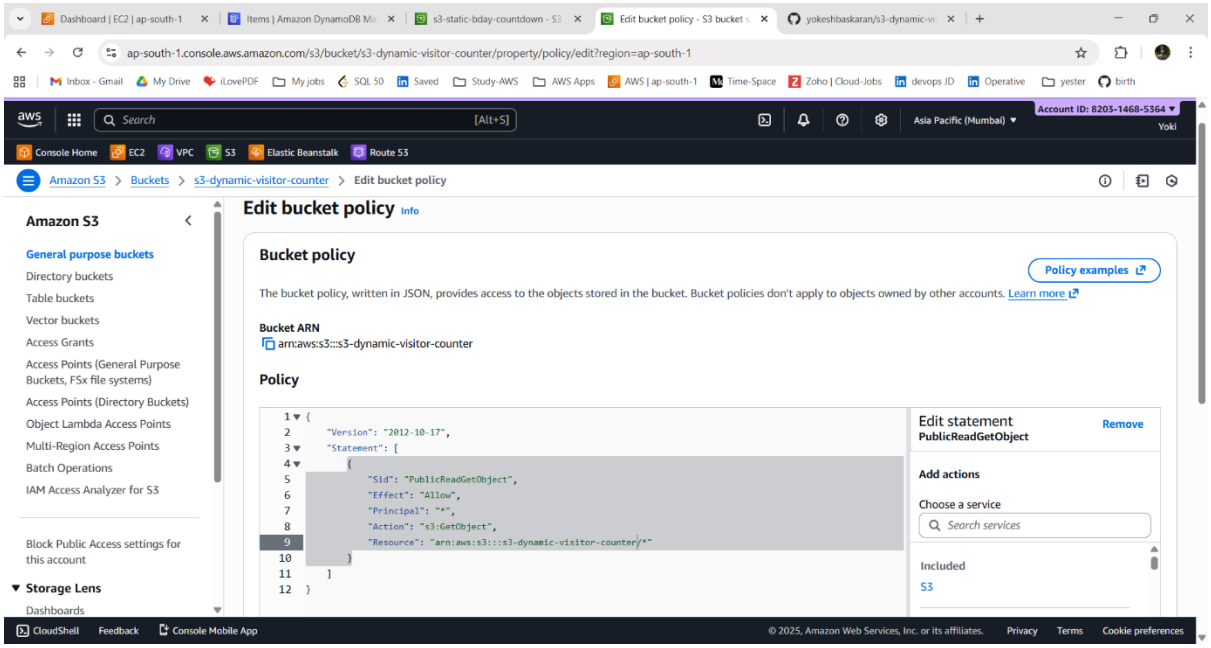
iii) Choose **Edit** and **uncheck** the "Block all Public Access"**.**

iv) type "confirm" and click Save Changes.



v) In same page of "**Permissions"** tab of the bucket. Scroll to **Bucket policy.**

vi) Choose **Edit** and add a policy that allows public read of all objects in the bucket.



BUCKET POLICY:

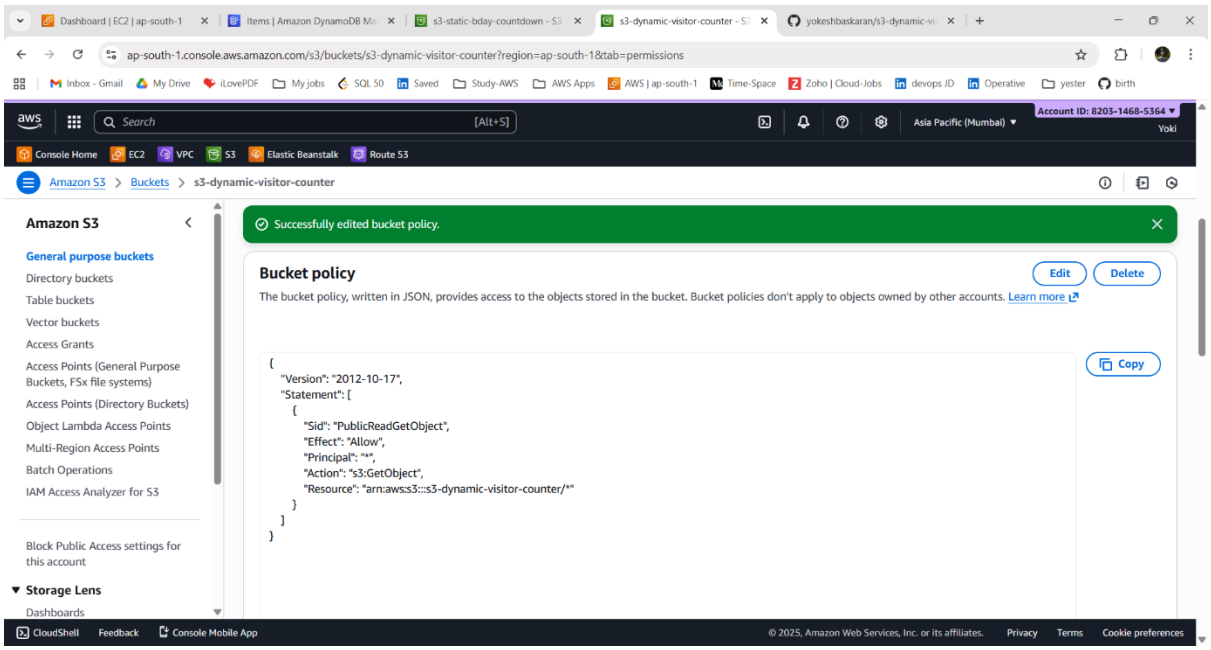
```

{
	"Version": "2012-10-17",
	"Statement": [
		{
			"Sid": "PublicReadGetObject",
			"Effect": "Allow",
			"Principal": "*",
			"Action": "s3:GetObject",
			"Resource": "arn:aws:s3:::s3-dynamic-visitor-counter/*"
		}
	]
}

```

vii) Save the policy.

viii) Verify that Public Access indicators show that the bucket allows public reads.



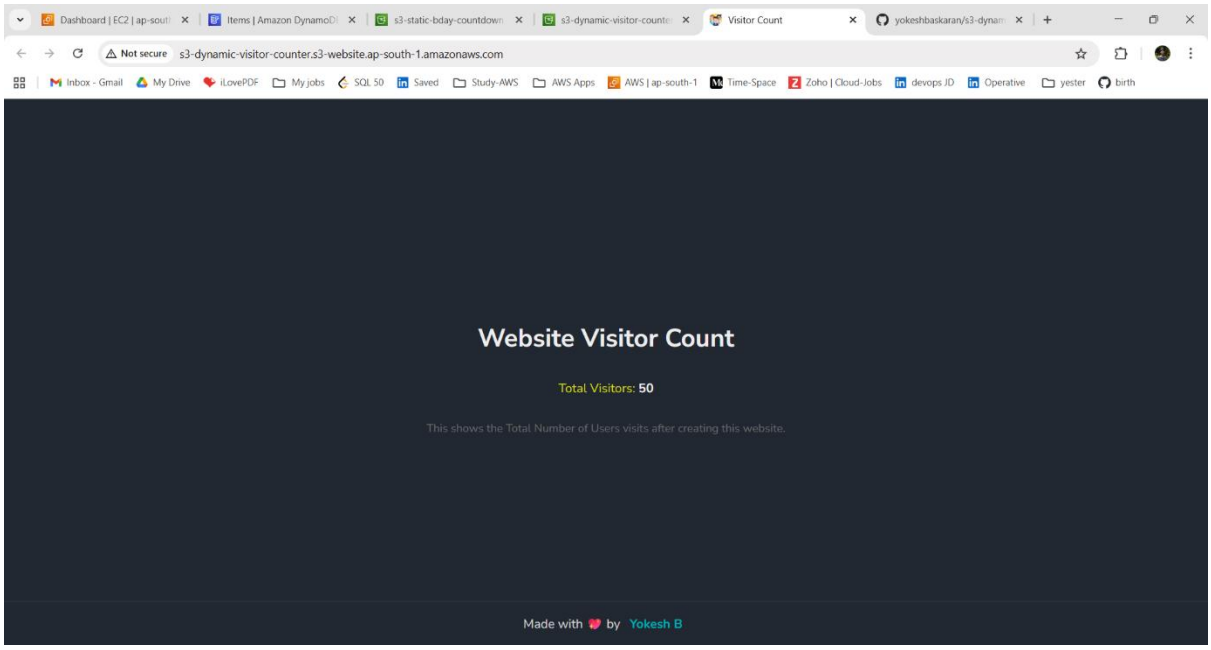
5. Verify Website Accessibility:

i) Open the S3 Website endpoint URL in your browser.

ii) Confirm that the site loads and that resources (CSS, JS files) work properly.

iii) If something doesn't load, check:

- Files are uploaded at the correct paths
- Permissions allow public read
- File names match exactly (case-sensitive)



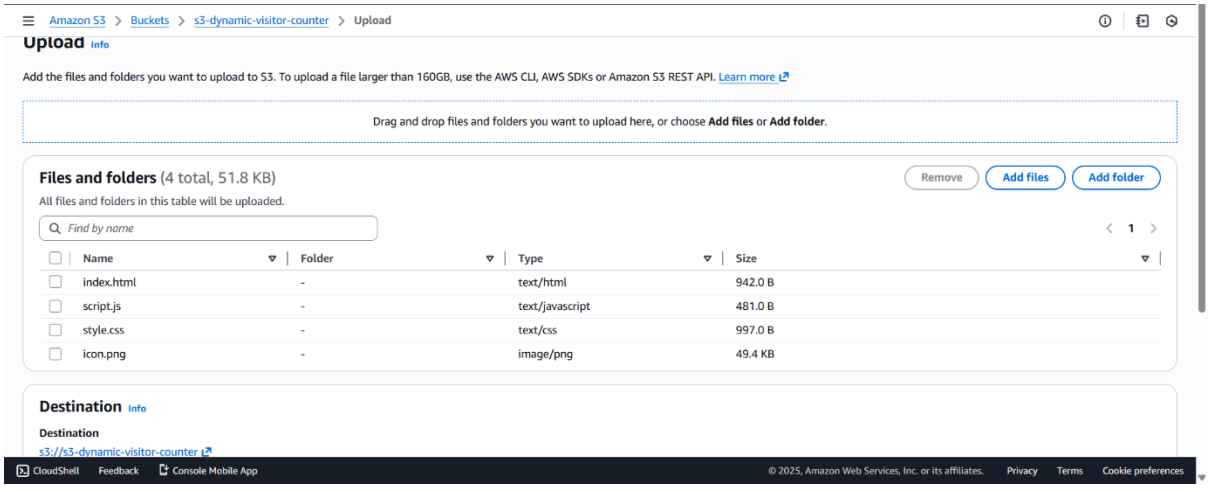
6. Maintenance & Updates:

i) To update your site, upload new versions of your files to the S3 bucket.

ii) Overwrite existing objects or delete and re-upload.

iii) Refresh the website to view changes (sometimes a hard refresh is needed: Ctrl+Shift+R).

~ updated code version.



VISUAL ARCHITECTURE DIAGRAM:

User Browser ← (user visits the website)

↓

S3 Static Hosting ← (serves HTML/JS assets)

↓

API Gateway ← (visitor counter API endpoint)

↓

Lambda Function ← (reads & updates visitor count)

↓

DynamoDB Table ← (atomic counter storage)

CONCLUSION

The project successfully implemented a **scalable serverless website visitor counter** using AWS Lambda, API Gateway, DynamoDB, and S3. The solution eliminates server management, scales automatically with traffic, and stores visitor data reliably.

By combining these AWS services, the system delivers a lightweight, cost-efficient, and highly available way to track website visits in real time, demonstrating the effectiveness of serverless architecture for modern web applications.