

## Project Title:

# Serverless Notes Web Application

### Overview:

A simple web application that allows users to create, view, update, and delete notes. The entire application is built using AWS services in a serverless architecture.

---

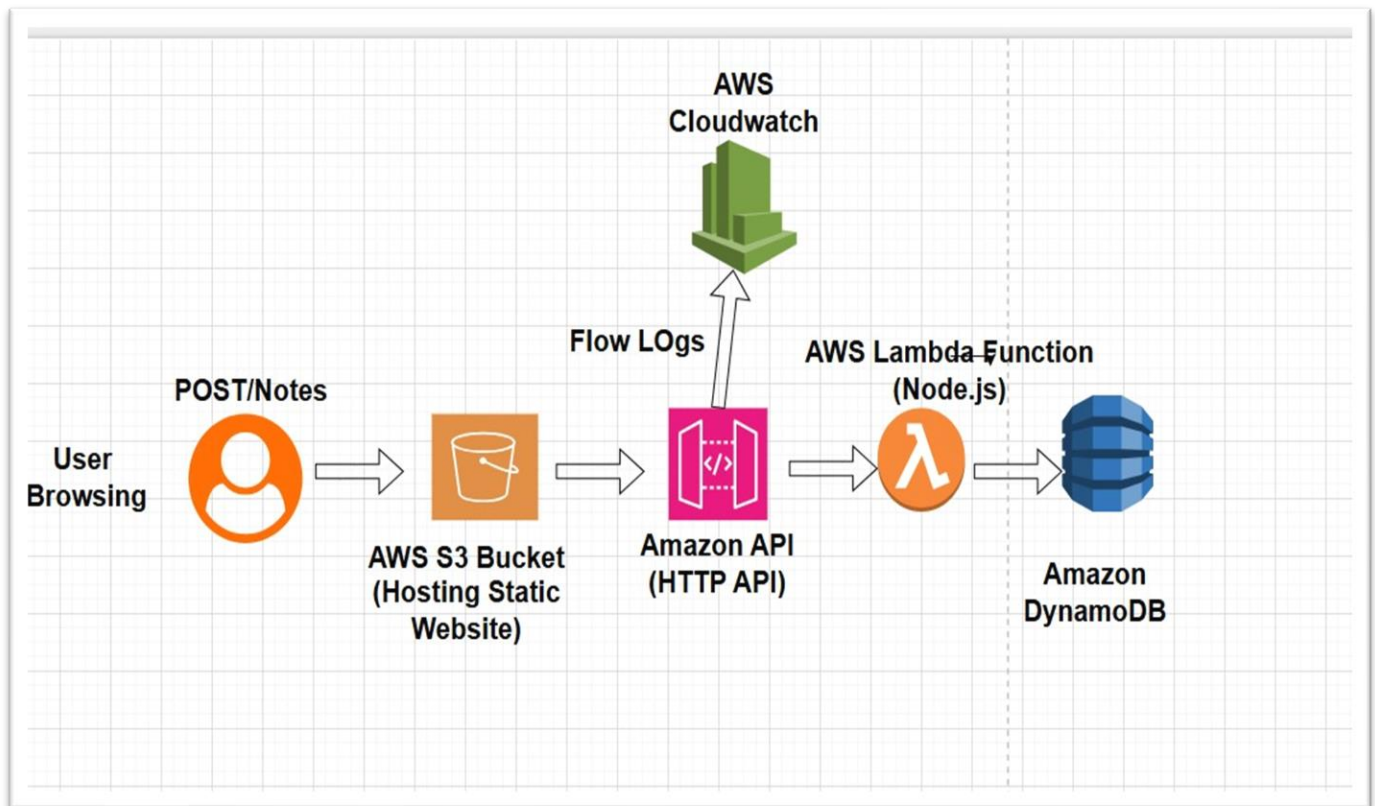
### Tech Stack:

- **Frontend:** HTML, CSS, JavaScript (Hosted on Amazon S3)
  - **Backend:** AWS Lambda + API Gateway
  - **Database:** Amazon DynamoDB
  - **Other Services:** IAM for access control
- 

### Main Features:

- Add a note
- View all notes
- Edit a note
- Delete a note
- Notes have timestamps

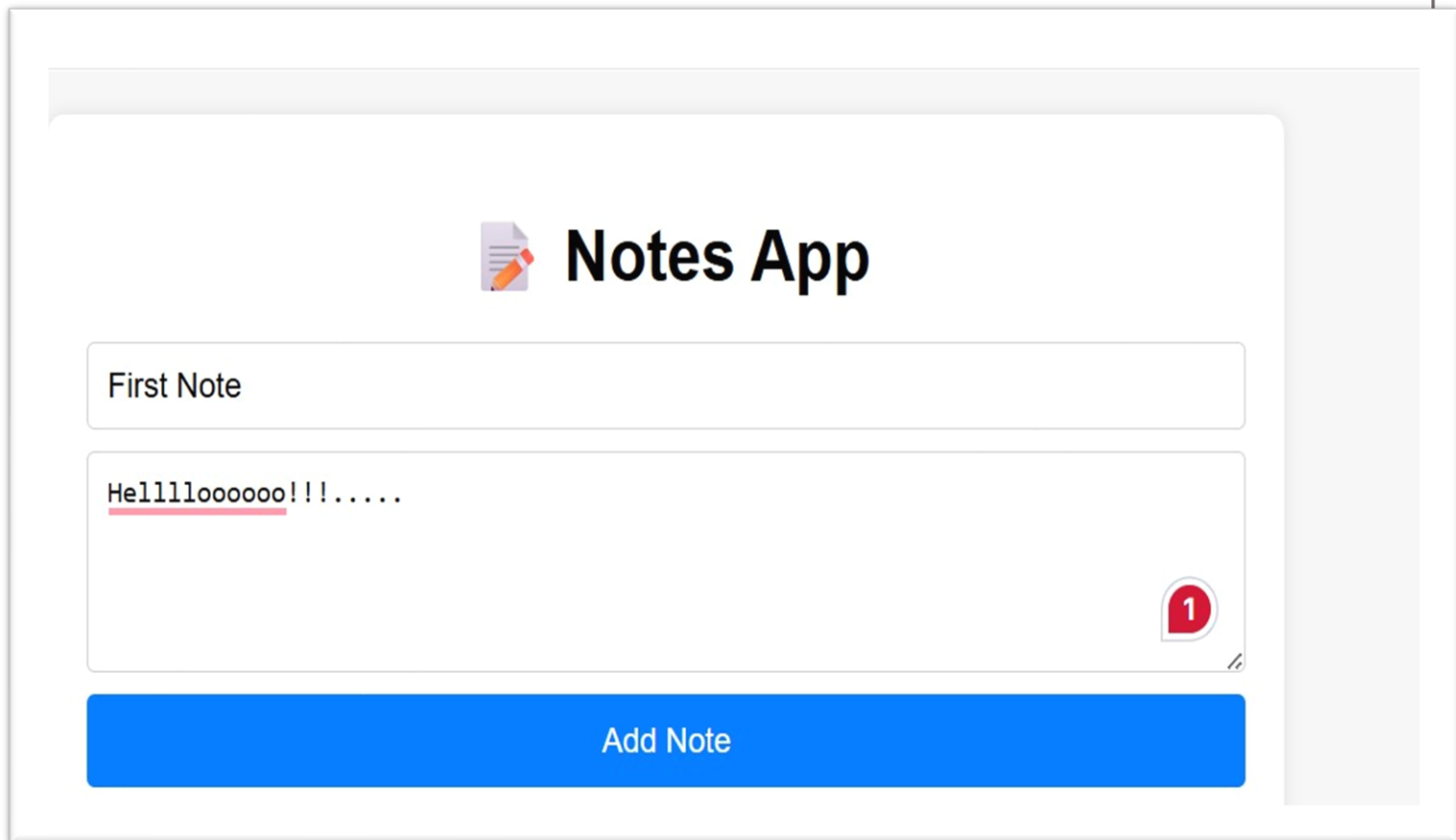
## Architecture Summary:



1. Static frontend hosted on **S3**
2. Frontend connects to **API Gateway**
3. **API Gateway** triggers **Lambda functions**
4. **Lambda** interacts with **DynamoDB**
5. Response is returned to frontend

## Screenshots:

### ➤ Frontend of Notes Application.



➤ Here is the DynamoDB table where our notes have been saved

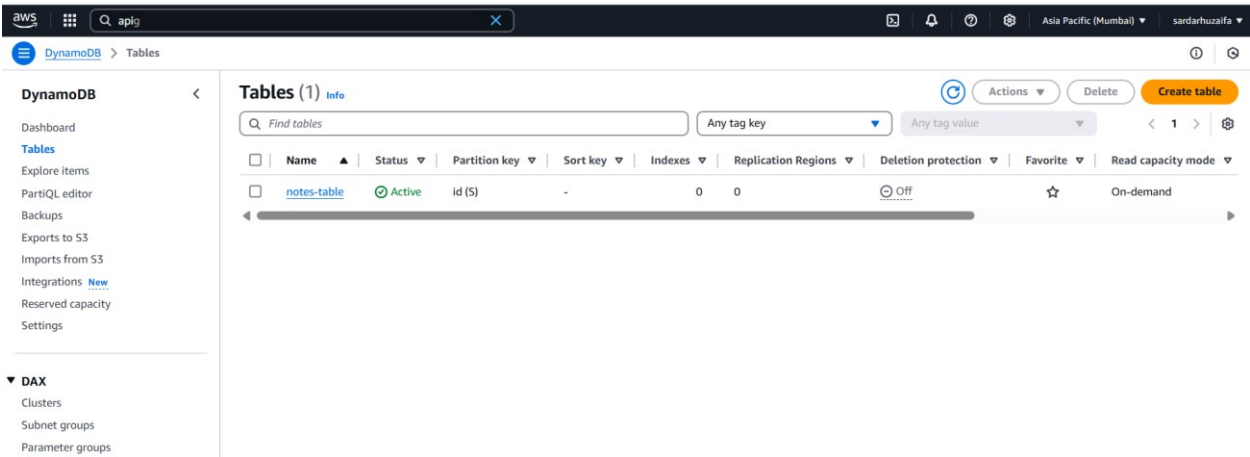


Table: notes-table - Items returned (5)

Actions

Create item

Scan started on June 11, 2025, 16:38:20

< 1 >

<input type="checkbox"/>	id (String)	content	createdAt	title
<input type="checkbox"/>	<a href="#">1749637929046</a>	Stored in D...	2025-06-1...	Test Note from Lambda
<input type="checkbox"/>	<a href="#">1749641329437</a>	first note a...	2025-06-1...	first note app on cloud
<input type="checkbox"/>	<a href="#">1749639502796</a>	This is my fi...	2025-06-1...	Amazon HTTP API
<input type="checkbox"/>	<a href="#">1749641804517</a>	heyyy	2025-06-1...	hhh
<input type="checkbox"/>	<a href="#">1749641266404</a>	Hellllooooo...	2025-06-1...	First Note

## Serverless Notes Web Application

### ➤ S3 Bucket Policy allowing public access.

Block *all* public access

⚠ Off

► Individual Block Public Access settings for this bucket

### Bucket policy

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. [Learn more](#)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::notes-web-app-22/*"
    }
  ]
}
```

Copy

Activate Windows  
Go to Settings to activate Windows.

### ➤ Uploaded objects i.e. for front end on S3 for hosting.

### notes-web-app-22

Info

Objects Properties Permissions Metrics Management Access Points

Objects (3)

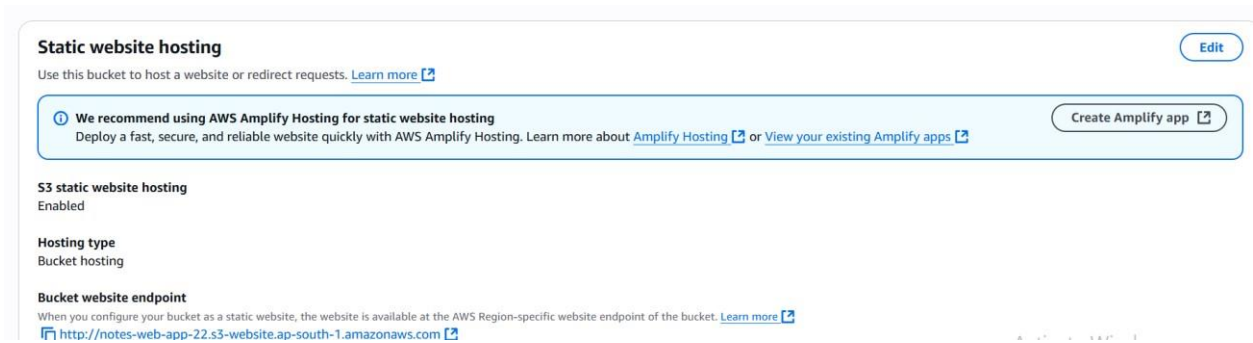
Copy S3 URI Copy URL Download Open Delete Actions Create folder

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicit permissions. [Learn more](#)

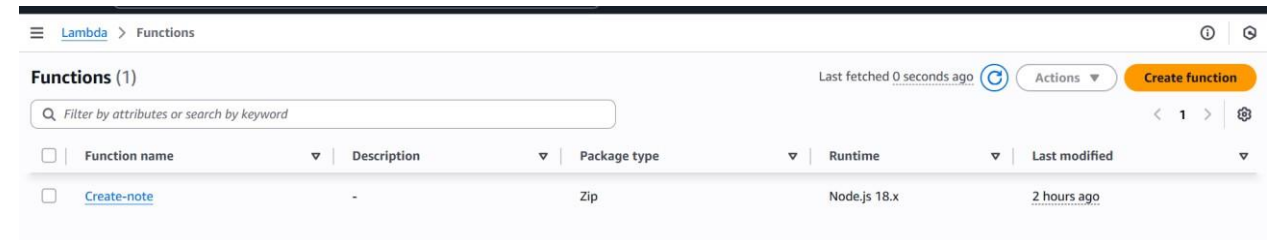
Find objects by prefix

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	<a href="#">index.html</a>	html	June 11, 2025, 12:55:11 (UTC+05:00)	676.0 B	Standard
<input type="checkbox"/>	<a href="#">script.js</a>	js	June 11, 2025, 16:05:37 (UTC+05:00)	2.4 KB	Standard
<input type="checkbox"/>	<a href="#">style.css</a>	css	June 11, 2025, 12:55:12 (UTC+05:00)	1.7 KB	Standard

➤ Enabling static website hosting on S3.



➤ Create Note Function for AWS Lambda.



- **The code inside the Lambda function's `index.mjs` file is responsible for handling the backend CRUD operations interacting with DynamoDB**



```
JS index.mjs > ...
1 import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";
2 const client = new DynamoDBClient({});
3 export const handler = async (event) => {
4   try {
5     const body = JSON.parse(event.body);
6
7     const id = Date.now().toString(); // 📅 now matches DynamoDB key
8
9     const params = {
10       TableName: "notes-table",
11       Item: {
12         id: { S: id }, // 📅 correct key
13         title: { S: body.title || "" },
14         content: { S: body.content || "" },
15         createdAt: { S: new Date().toISOString() }
16       }
17     };
18
19     console.log("Saving note:", JSON.stringify(params, null, 2));
20
21     await client.send(new PutItemCommand(params));
22
23     return {
24       statusCode: 200,
25       body: JSON.stringify({ message: "Note created!", id })
26     };
27   } catch (err) {
28     console.error(err);
29     return {
30       statusCode: 500,
```

- **It is the test case A Status code of 200 means the request was successfully processed without any errors.**

The screenshot displays the AWS Lambda console interface for a function named `index.mjs`. The left sidebar contains navigation options: **CREATE-NOTE** (selected), **DEPLOY [UNDEPLOYED CHANGES]**, **TEST EVENTS [SELECTED: HELLOCREATE]**, and **ENVIRONMENT VARIABLES**. The **DEPLOY** section shows a message "You have undeployed changes." and buttons for **Deploy (Ctrl+Shift+U)** and **Test (Ctrl+Shift+I)**. The **TEST EVENTS** section shows a list of events, with `hellocreate` selected. The main area displays the `index.mjs` code, which uses the `@aws-sdk/client-dynamodb` library to create a new item in a DynamoDB table named `notes-table`. The code includes comments for key values and uses `Date.now().toString()` for the `id` field. Below the code, the **OUTPUT** tab shows the test results for the `hellocreate` event. The status is **Succeeded**, and the response is a JSON object with `statusCode: 200` and a body containing a success message and the generated `id`. The **FUNCTION LOGS** section shows the start of the function execution with the request ID and version.

```
JS index.mjs > ...
1 import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";
2 const client = new DynamoDBClient({});
3 export const handler = async (event) => {
4   try {
5     const body = JSON.parse(event.body);
6
7     const id = Date.now().toString(); // 🏆 now matches DynamoDB key
8
9     const params = {
10       TableName: "notes-table",
11       Item: {
12         id: { S: id }, // 🏆 correct key
13         title: { S: body.title || "" },
14         content: { S: body.content || "" },
15         createdAt: { S: new Date().toISOString() }
16       }
17     };
18   } catch (error) {
19     console.error("Error creating note:", error);
20     return {
21       statusCode: 500,
22       body: JSON.stringify({ message: "Error creating note" })
23     };
24   }
25   return {
26     statusCode: 200,
27     body: JSON.stringify({ message: "Note created!", id })
28   };
29 }
```

PROBLEMS OUTPUT CODE REFERENCE LOG TERMINAL

Status: Succeeded  
Test Event Name: hellocreate

Response:

```
{
  "statusCode": 200,
  "body": "{\"message\":\"Note created!\",\"id\":\"1749646452440\"}"
}
```

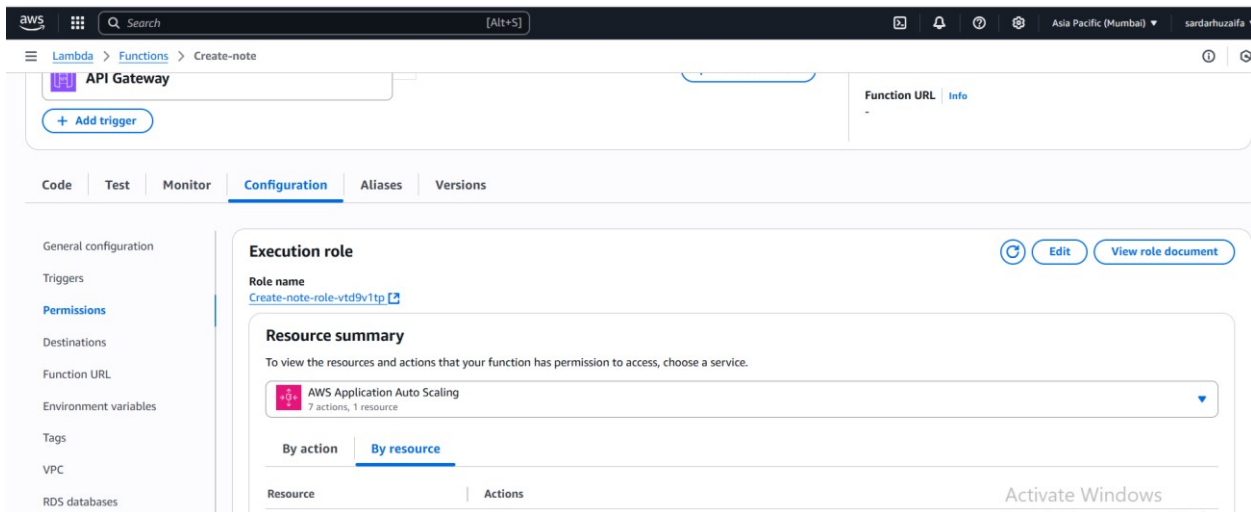
Function Logs:

```
START RequestId: 9eb86ca1-7ae9-4a5c-87b0-462e8accf20f Version: $LATEST
2025-06-11T12:54:12.441Z 9eb86ca1-7ae9-4a5c-87b0-462e8accf20f INFO Saving note: {
  "title": "Note",
  "content": "Content",
  "createdAt": "2025-06-11T12:54:12.441Z"
}
```

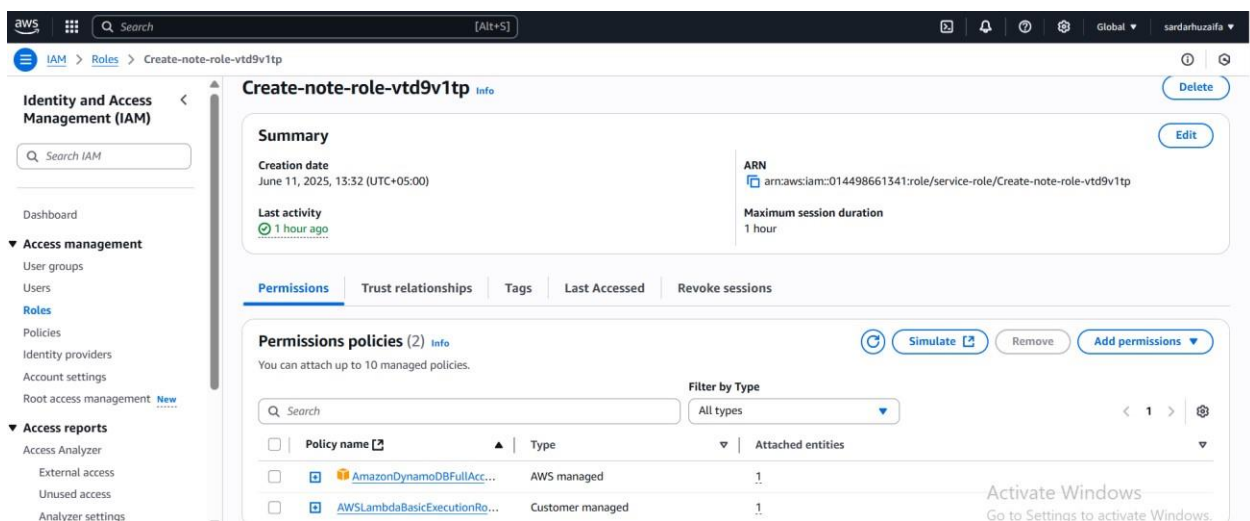


## Serverless Notes Web Application

- We created an AWS IAM Role to allow the Lambda function to securely read from and write to the DynamoDB table.

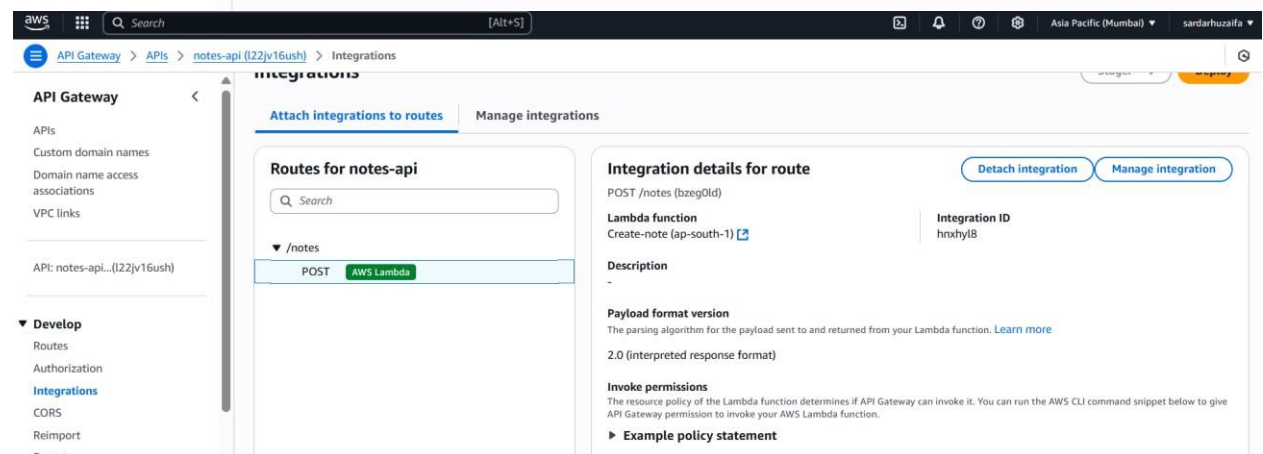
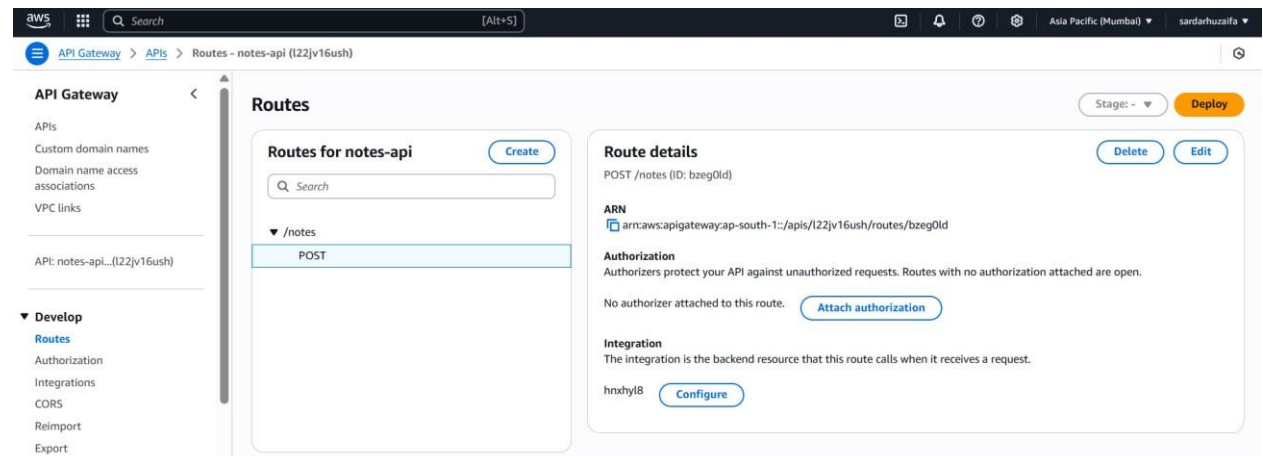
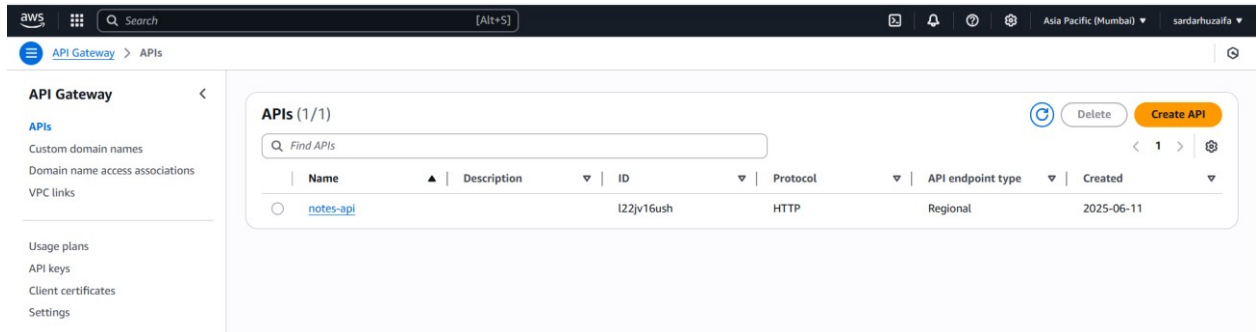


- Here is the policy for AWS IAM Role allowing AWS DynamoDB Full Access.



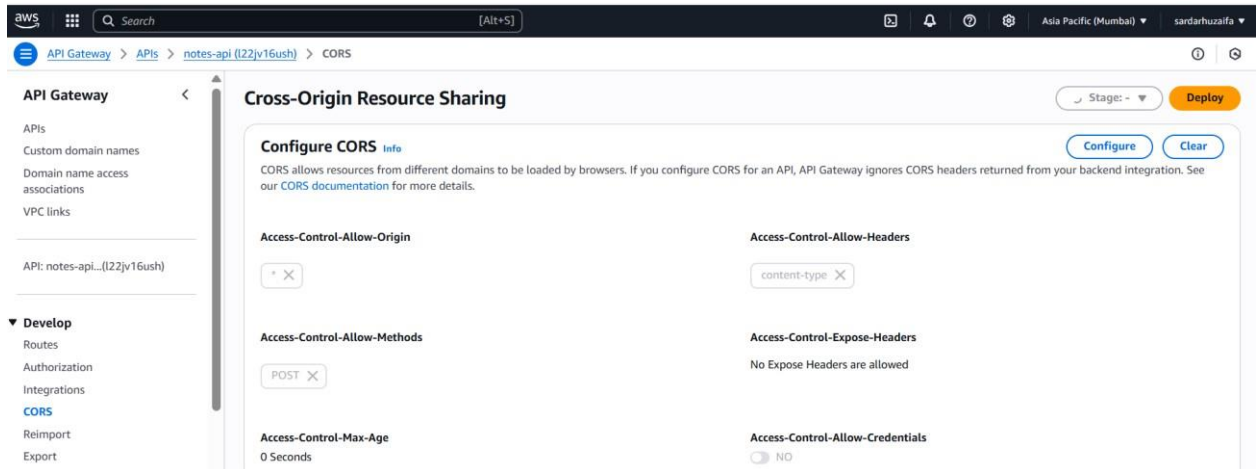
## Serverless Notes Web Application

- Now we have created API for Notes, setting up POST routes and integrating it with AWS Lambda



## Serverless Notes Web Application

- **CORS (Cross-Origin Resource Sharing)** was enabled in API Gateway to allow the frontend (hosted on S3) to communicate with the backend (API Gateway) from a different domain.



- **CloudWatch Logs** were used to monitor and debug Lambda function executions by capturing request and error logs.

