

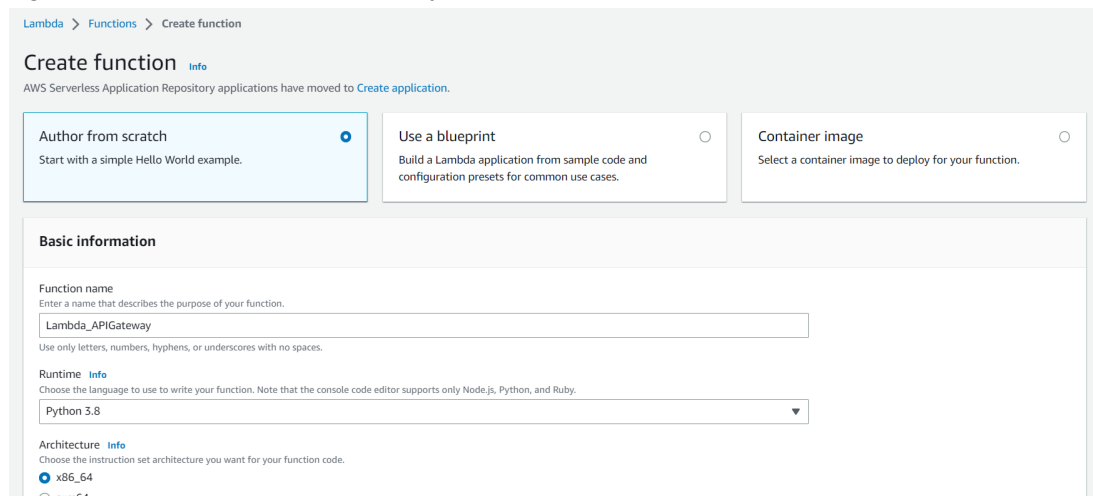
API Gateway

Problem Statement:

1. Create and configure a Lambda function in Python or Node.js to perform operations on a DynamoDB table.
2. Create a REST API in API Gateway to connect to your Lambda function.
3. Create a DynamoDB table and test it with your Lambda function in the console.
4. Deploy your API and test the full setup using curl in a terminal.

Solution:

Creating a Lambda function with the Python 3.8 runtime



The screenshot shows the 'Create function' page in the AWS Lambda console. The 'Author from scratch' option is selected. The 'Basic information' section is visible, showing the function name 'Lambda_APIGateway', the runtime 'Python 3.8', and the architecture 'x86_64'.

Create function [Info](#)

AWS Serverless Application Repository applications have moved to [Create application](#).

Author from scratch ☒ Start with a simple Hello World example.

Use a blueprint ☐ Build a Lambda application from sample code and configuration presets for common use cases.

Container image ☐ Select a container image to deploy for your function.

Basic information

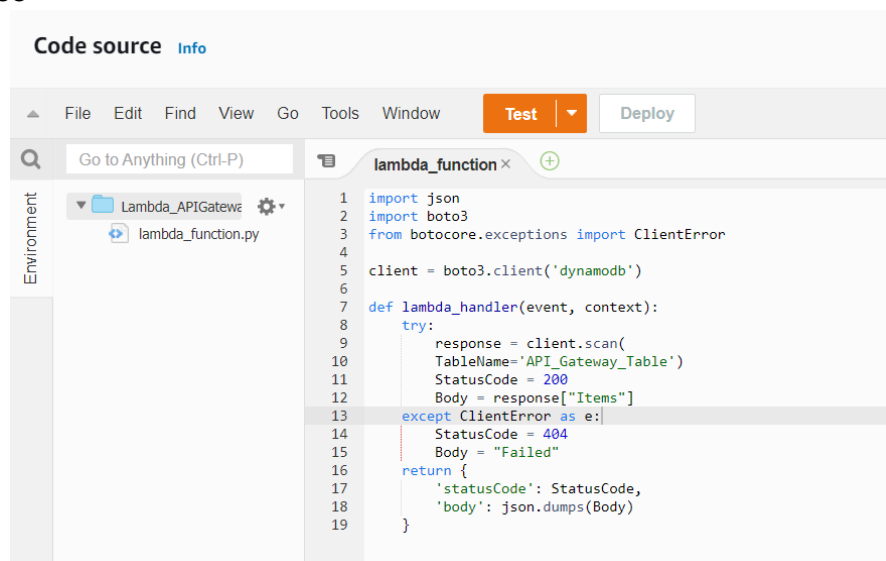
Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.
☒ x86_64
☐ arm64



Updated the code in the lambda function to get items from the DynamoDB table whenever the function is triggered.



The screenshot shows the 'Code source' editor in the AWS Lambda console. The code is a Python lambda function that uses boto3 to scan a DynamoDB table named 'API_Gateway_Table' and return the items as a JSON response. The code is as follows:

```
1 import json
2 import boto3
3 from botocore.exceptions import ClientError
4
5 client = boto3.client('dynamodb')
6
7 def lambda_handler(event, context):
8     try:
9         response = client.scan(
10             TableName='API_Gateway_Table')
11         StatusCode = 200
12         Body = response["Items"]
13     except ClientError as e:
14         StatusCode = 404
15         Body = "Failed"
16     return {
17         'statusCode': StatusCode,
18         'body': json.dumps(Body)
19     }
```


Created version for the lambda function code which implements the **GET method operation**.


 Successfully created version 1 of function Lambda_APIGateway. 

[Lambda](#) > [Functions](#) > [Lambda_APIGateway](#) > Version: 1

Version: 1 Copy ARN Actions

▼ **Function overview** Info

 Lambda_APIGateway:1


 Layers (0)

+ Add trigger

+ Add destination

Description
GET

Last modified
2 minutes ago

Function ARN
 arn:aws:lambda:ap-south-1:818119396514:function:Lambda_APIGateway:1

Creating an alias to the previously created version.

[Lambda](#) > [Functions](#) > [Lambda_APIGateway](#) > [Create alias](#)

Create alias

Alias configuration

An alias is a pointer to one or two versions. Choose each version that you want the alias to point to.

Name

GET

Description - *optional*

Version

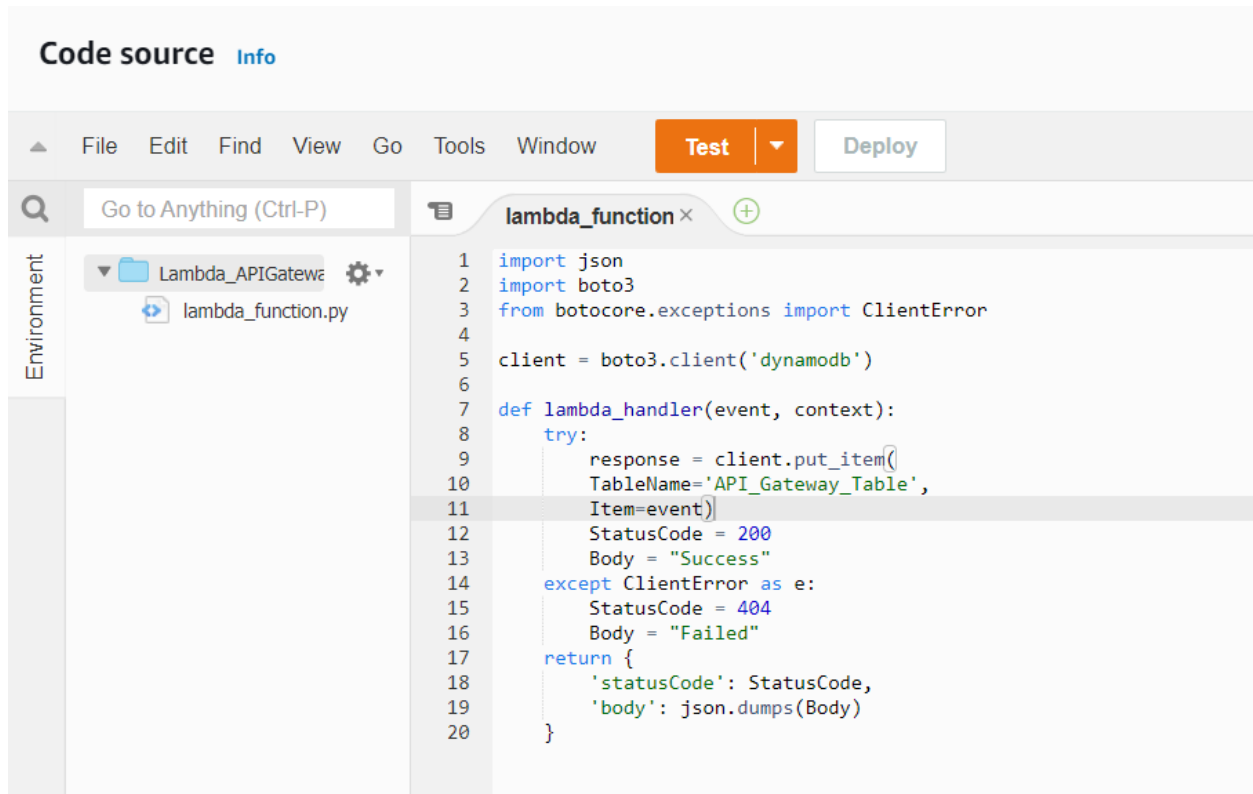
1 ▼

► Weighted alias

Cancel

Save

Updated the lambda function code to perform the **post method operation**. After that, I created the version and alias to the existing code.



```
Code source Info

File Edit Find View Go Tools Window Test Deploy

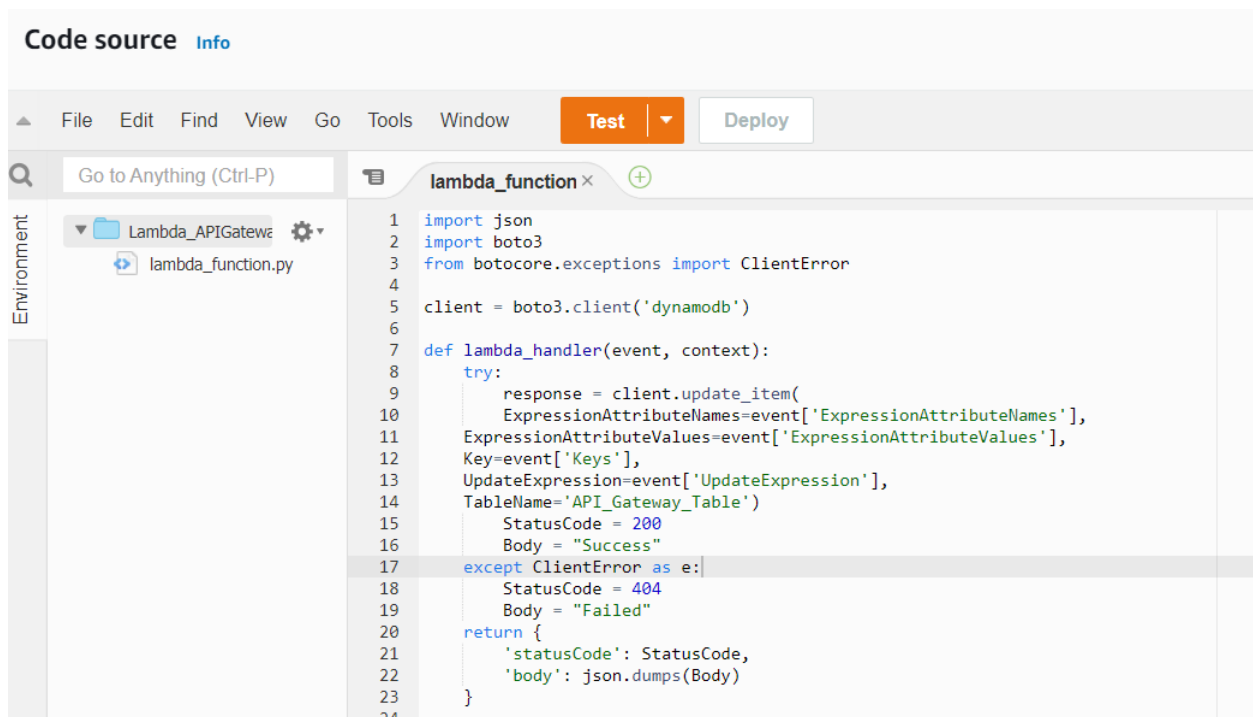
Go to Anything (Ctrl-P)

Environment

Lambda_APIGateway
lambda_function.py

1 import json
2 import boto3
3 from botocore.exceptions import ClientError
4
5 client = boto3.client('dynamodb')
6
7 def lambda_handler(event, context):
8     try:
9         response = client.put_item(
10             TableName='API_Gateway_Table',
11             Item=event)
12         StatusCode = 200
13         Body = "Success"
14     except ClientError as e:
15         StatusCode = 404
16         Body = "Failed"
17     return {
18         'statusCode': StatusCode,
19         'body': json.dumps(Body)
20     }
```

Updated the lambda function code to perform the **put method operation**. After that, I created the version and alias to the existing code.



```
Code source Info

File Edit Find View Go Tools Window Test Deploy

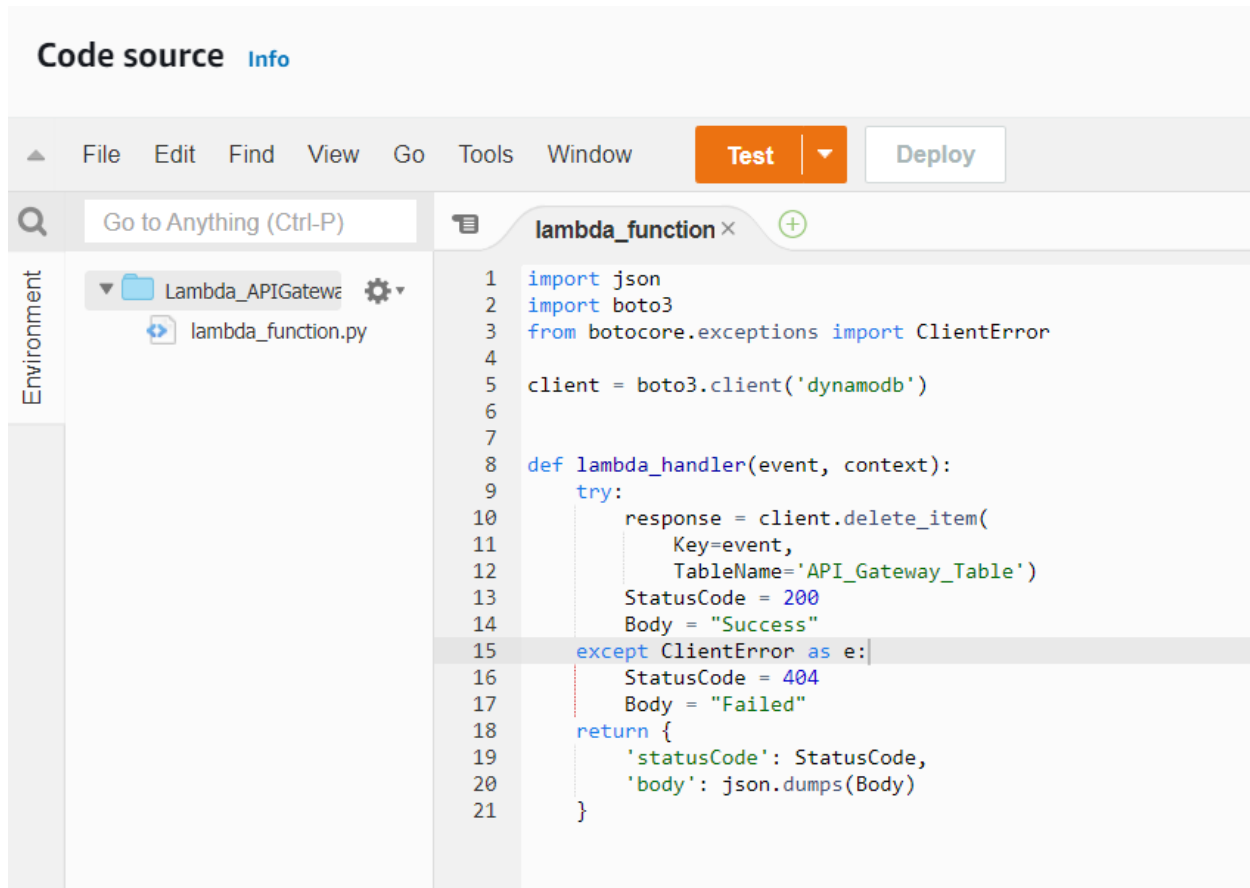
Go to Anything (Ctrl-P)

Environment

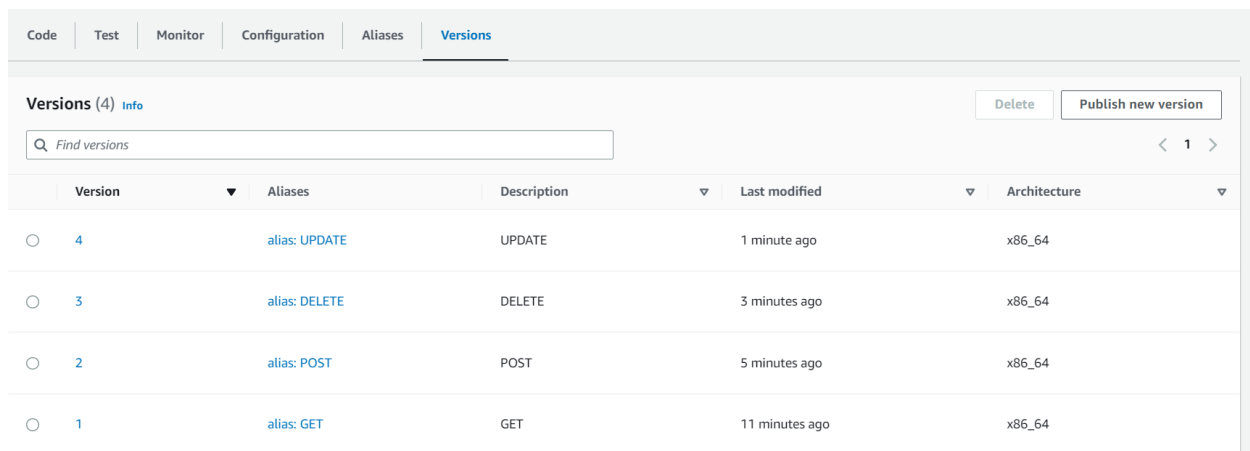
Lambda_APIGateway
lambda_function.py

1 import json
2 import boto3
3 from botocore.exceptions import ClientError
4
5 client = boto3.client('dynamodb')
6
7 def lambda_handler(event, context):
8     try:
9         response = client.update_item(
10             ExpressionAttributeNames=event['ExpressionAttributeNames'],
11             ExpressionAttributeValues=event['ExpressionAttributeValues'],
12             Key=event['Keys'],
13             UpdateExpression=event['UpdateExpression'],
14             TableName='API_Gateway_Table')
15         StatusCode = 200
16         Body = "Success"
17     except ClientError as e:
18         StatusCode = 404
19         Body = "Failed"
20     return {
21         'statusCode': StatusCode,
22         'body': json.dumps(Body)
23     }
```

Updated the lambda function code to perform the **delete method operation**. After that, I created the version and alias to the existing code.



```
1 import json
2 import boto3
3 from botocore.exceptions import ClientError
4
5 client = boto3.client('dynamodb')
6
7
8 def lambda_handler(event, context):
9     try:
10         response = client.delete_item(
11             Key=event,
12             TableName='API_Gateway_Table')
13         StatusCode = 200
14         Body = "Success"
15     except ClientError as e:
16         StatusCode = 404
17         Body = "Failed"
18     return {
19         'statusCode': StatusCode,
20         'body': json.dumps(Body)
21     }
```



Version	Aliases	Description	Last modified	Architecture
4	alias: UPDATE	UPDATE	1 minute ago	x86_64
3	alias: DELETE	DELETE	3 minutes ago	x86_64
2	alias: POST	POST	5 minutes ago	x86_64
1	alias: GET	GET	11 minutes ago	x86_64

Finally to perform the CRUD operation on the Dynamodb table created all versions of the lambda function code.

Creating a API Gateway Rest API

REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:

Lambda, HTTP, AWS Services

Import

Build

Amazon API Gateway APIs > Create

Hide hints ?

Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

☒ REST ☐ WebSocket

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

☒ New API ☐ Import from Swagger or Open API 3 ☐ Example API

Settings

Choose a friendly name and description for your API.

API name*

DynamoDB_CURD

Description

Endpoint Type

Regional

* Required

Create API

After creating the rest api creating a resource to the api which helps to perform the routing operation.

APIs > DynamoDB_CURD (ehvq37r920) > Resources > / (rlwx7lkul) > Create

Hide hints ?

Resources

Actions New Child Resource

RESOURCE ACTIONS

Create Method

Create Resource

Enable CORS

Edit Resource Documentation

API ACTIONS

Deploy API

Import API

Edit API Documentation

Delete API

Create a new child resource for your resource.

☒ proxy resource

Resource Name*

POST

Resource Path*

/ post

You can add path parameters using brackets. For example, the resource path {username} represents a path parameter called 'username'. Configuring /{proxy+} as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.

API Gateway CORS

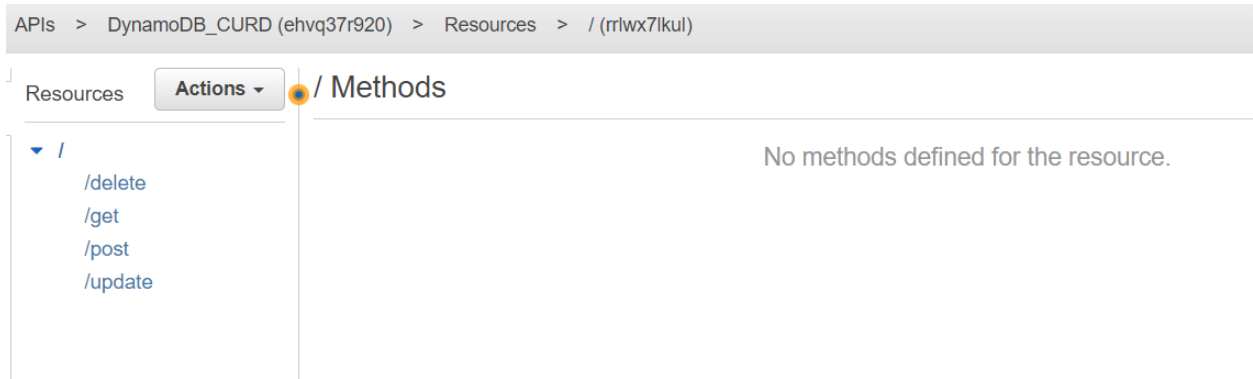
☐

* Required

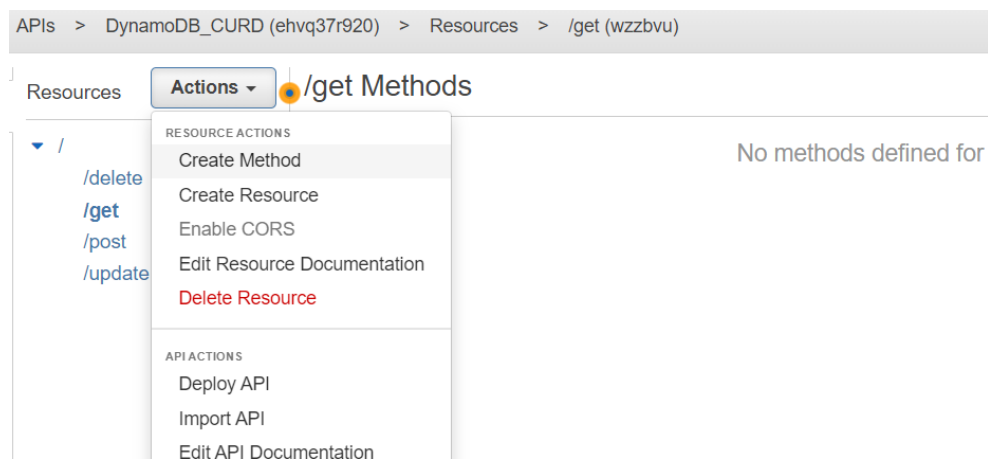
Cancel

Create Resource

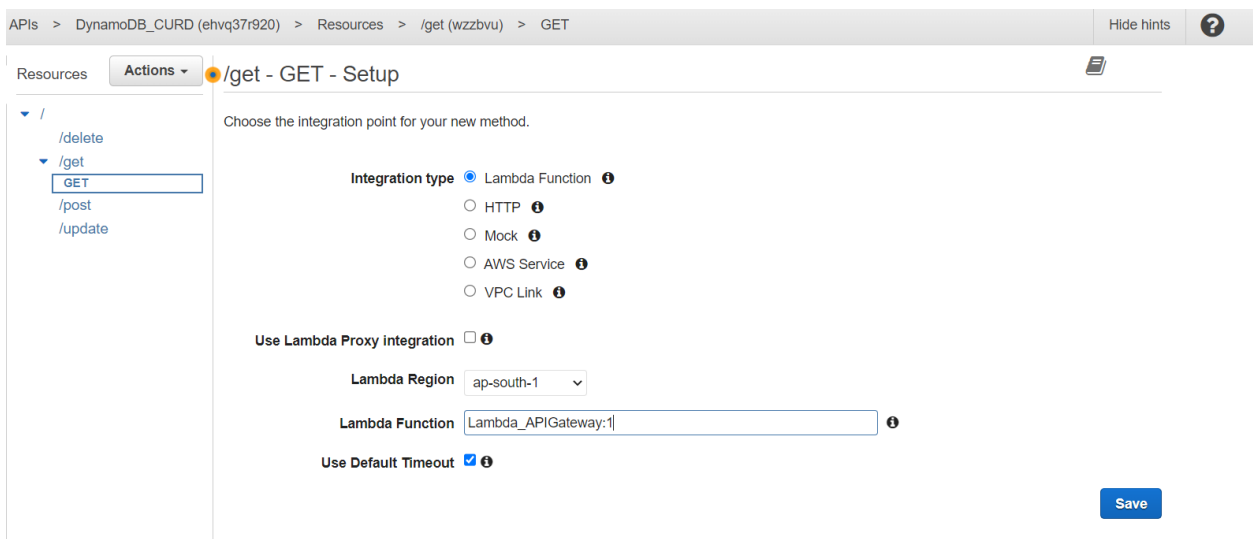
Same way finally created all routes resources now we can perform the routing with the following routes after deploying the API.



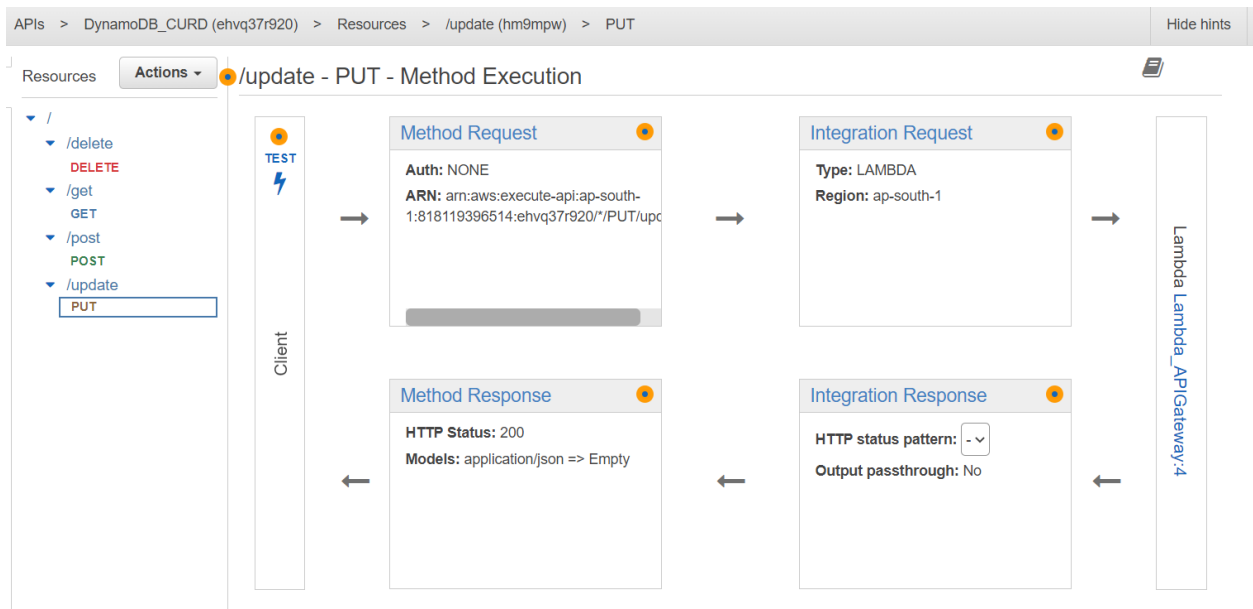
Creating a get method to the resource



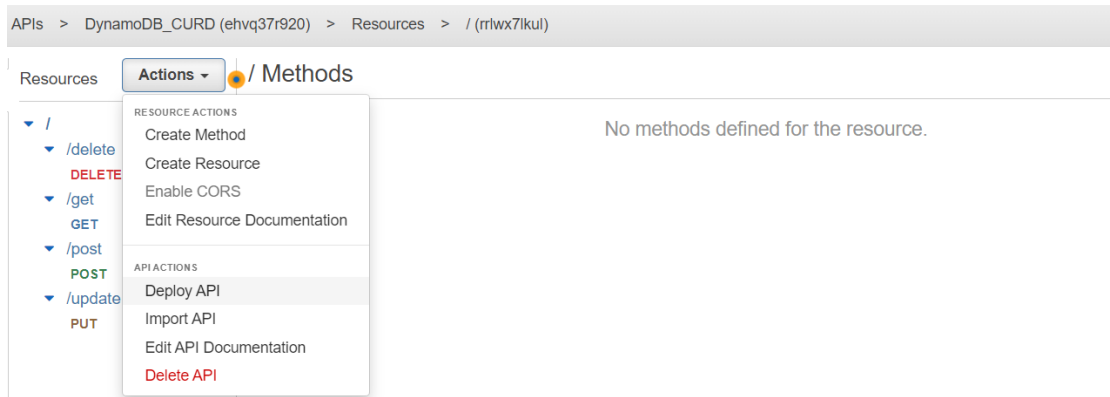
After that integrating the lambda function version to the get method



Sameway created methods and added the respective lambda function version to them.



Finally deploying the api so we can trigger lambda function with the rest api.



Deploying a API in the Dev stage

Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage

[New Stage]

Stage name*

dev

Stage description

Deployment description

Cancel

Deploy

Created a Dynamodb table with ID as partition key and Name as a Sort key.

The screenshot shows the AWS Management Console interface for a DynamoDB table named **API_Gateway_Table**. The table is configured with **ID (String)** as the Partition key and **Name (String)** as the Sort key. The Capacity mode is **Provisioned** and the Table status is **Active**. There are **No active alarms** and **Point-in-time recovery (PITR)** is **Off**. A notification banner at the top encourages enabling PITR to protect the table from accidental writes and deletes.

General information			
Partition key ID (String)	Sort key Name (String)	Capacity mode Provisioned	Table status Active
Alarms No active alarms	Point-in-time recovery (PITR) Off		

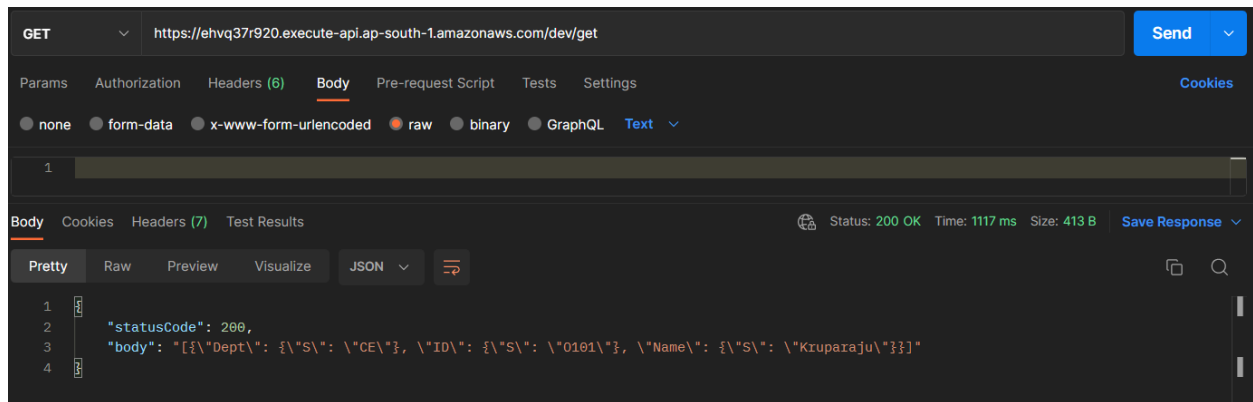
Performing CRUD operations with the help of a postman on the Dynamodb table. For the first time performing the post operation it will insert a new item in the DynamoDB table.

The screenshot shows a Postman interface with a POST request to the endpoint `https://ehvq37r920.execute-api.ap-south-1.amazonaws.com/dev/post`. The request body is a JSON object representing a new item to be inserted into the DynamoDB table. The response is a 200 OK status with a JSON body indicating success.

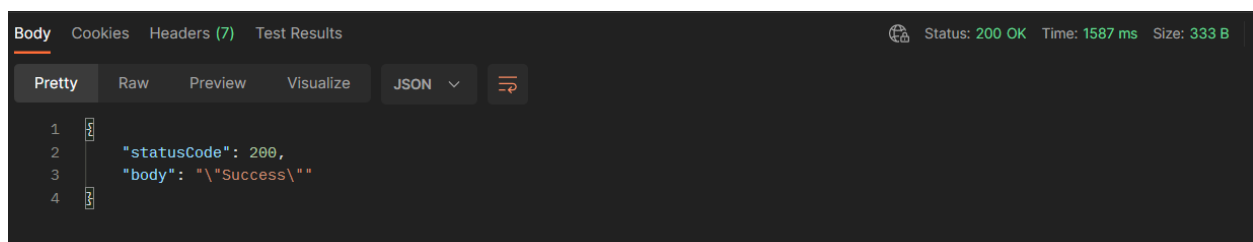
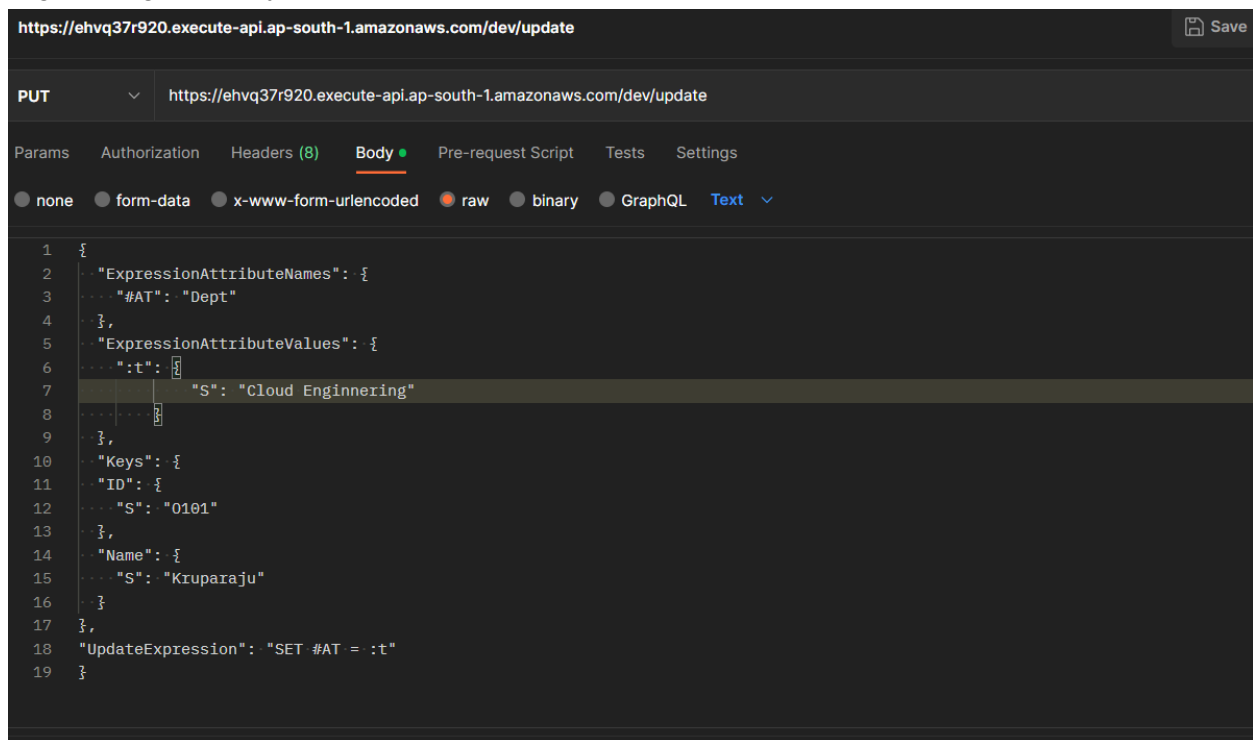
```
1 {
2   "ID": {
3     "S": "0101"
4   },
5   "Name": {
6     "S": "Kruparaju"
7   },
8   "Dept": {
9     "S": "CE"
10  }
11 }
```

```
1 {
2   "statusCode": 200,
3   "body": "\"Success\""
4 }
```

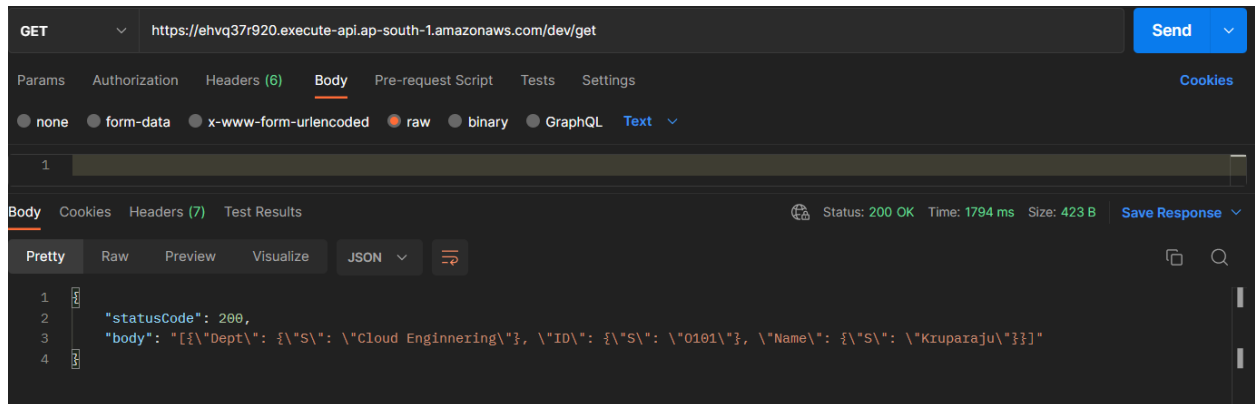

After performing the get operation it will list all items in the DynamoDB table.



After performing the put operation it will update the Dept value of the item from CE to cloud engineering in the DynamoDB table.



After that performed the get operation to list the changes done.



Finally deleting the item from the dynamo db table.

