Name: Atharva Patil Class: D15C Roll No: 39

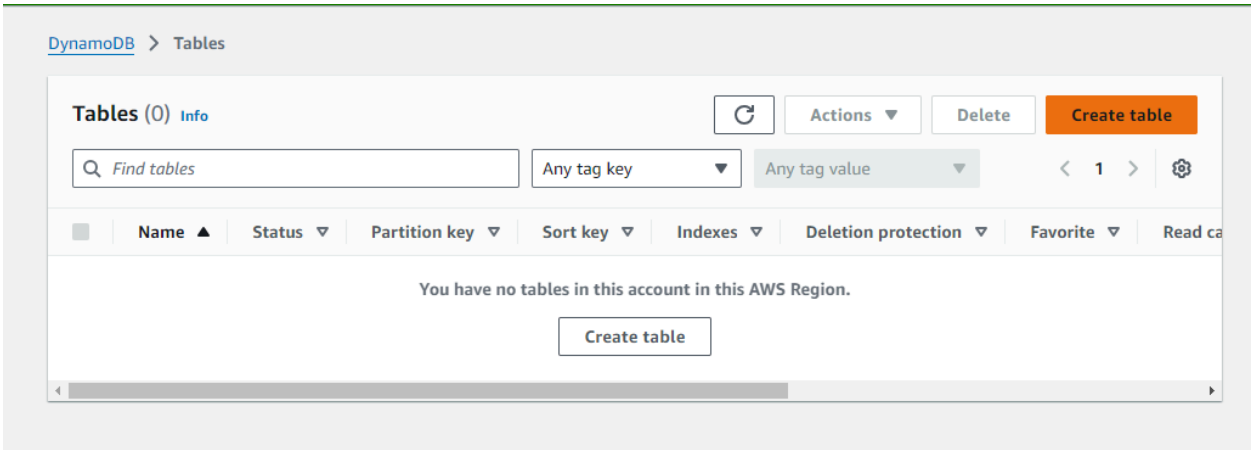
Advance DevOps Practical Examination Case Study Assignment

Topic: 19. Building a Serverless REST API
● Concepts Used: AWS Lambda, API Gateway, DynamoDB.
● Problem Statement: "Create a simple serverless REST API using AWS Lambda and
API Gateway to manage user data in a DynamoDB table. The API should support adding a new user and retrieving user details."
● Tasks:
○ Create a DynamoDB table to store user data.
○ Write two Lambda functions: one for adding a user to the table and another for retrieving user details by ID.
○ Set up an API Gateway to trigger these Lambda functions based on HTTP methods (POST for adding and GET for retrieving).
○ Test the API using curl or Postman.

Steps to perform the practical:

Step 1: Set Up the DynamoDB Table

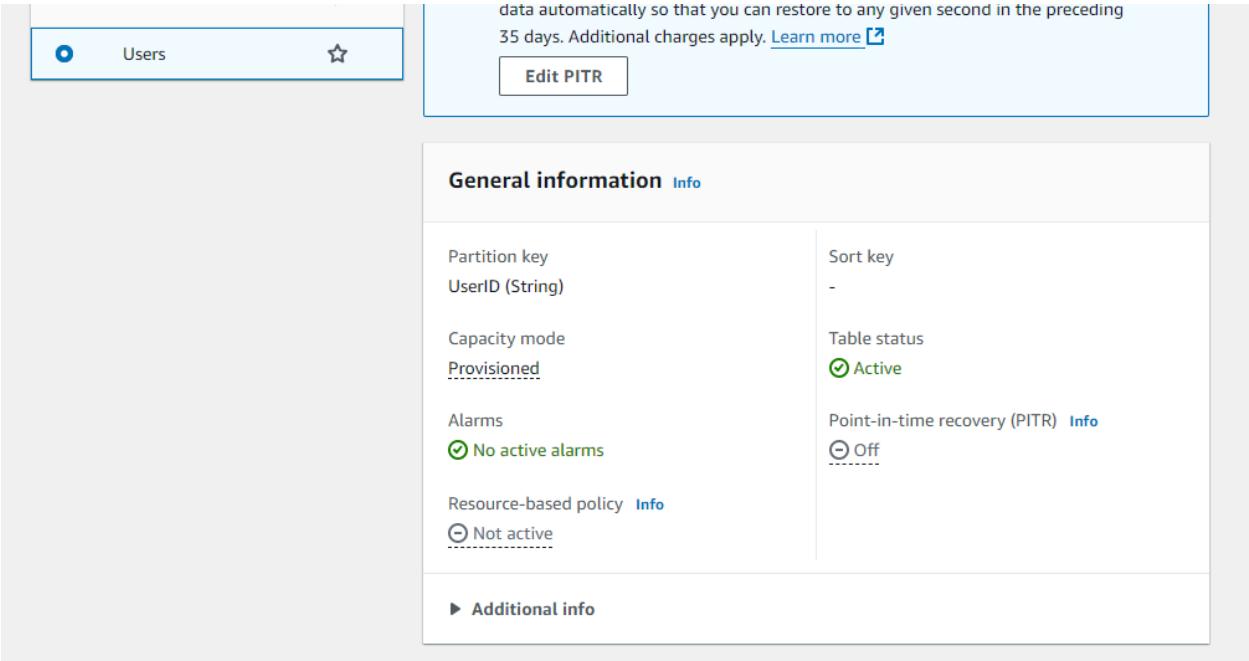
Click on **Create Table**.



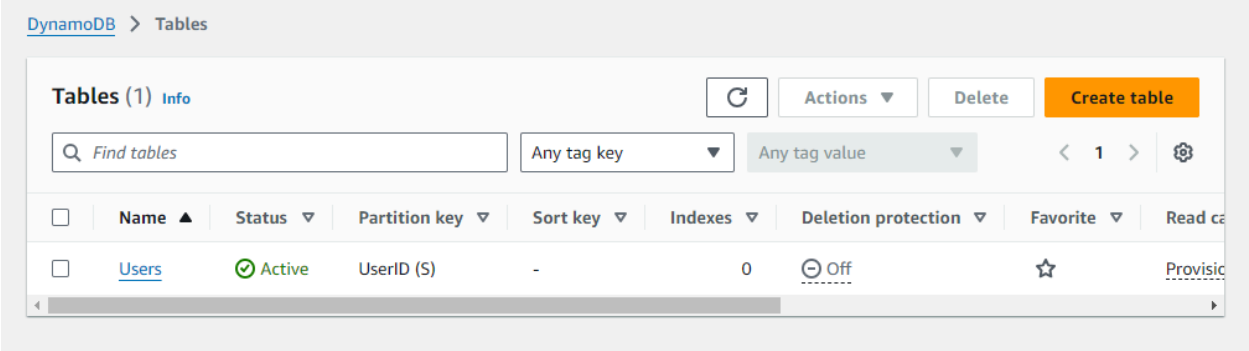
Name: Atharva Patil Class: D15C Roll No: 39

Table Name: `Users`

`Partition key`: `UserID`



`Table Created Successfully`

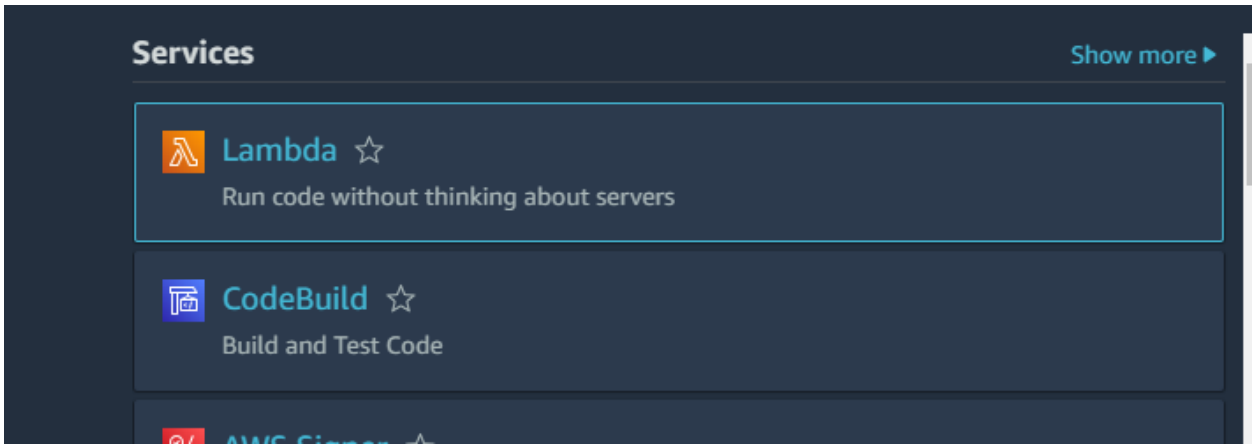


Name: Atharva Patil Class: D15C Roll No: 39

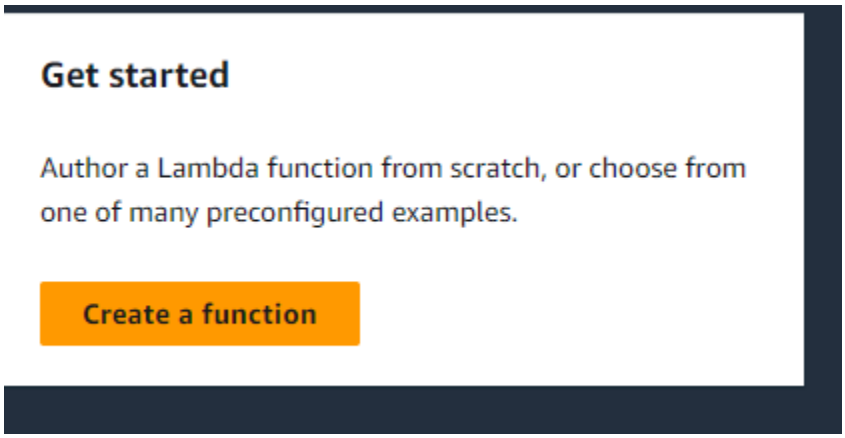
Step 2: Write Lambda Functions

Function 1: Add User

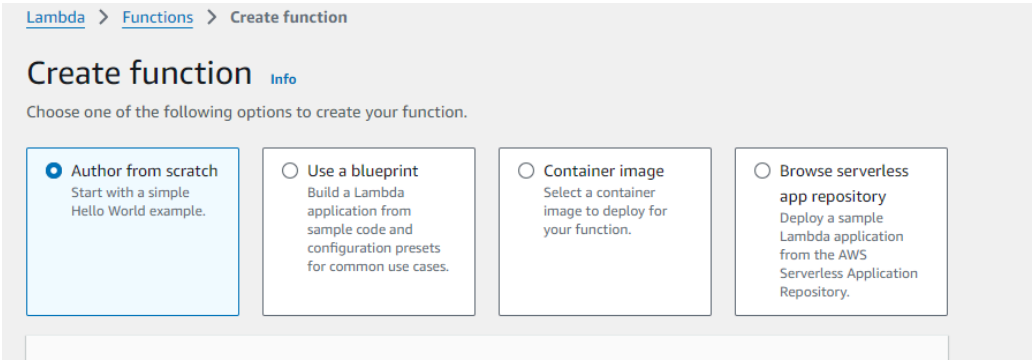
1. Navigate to **AWS Lambda**.



2. Click on **Create Function**.

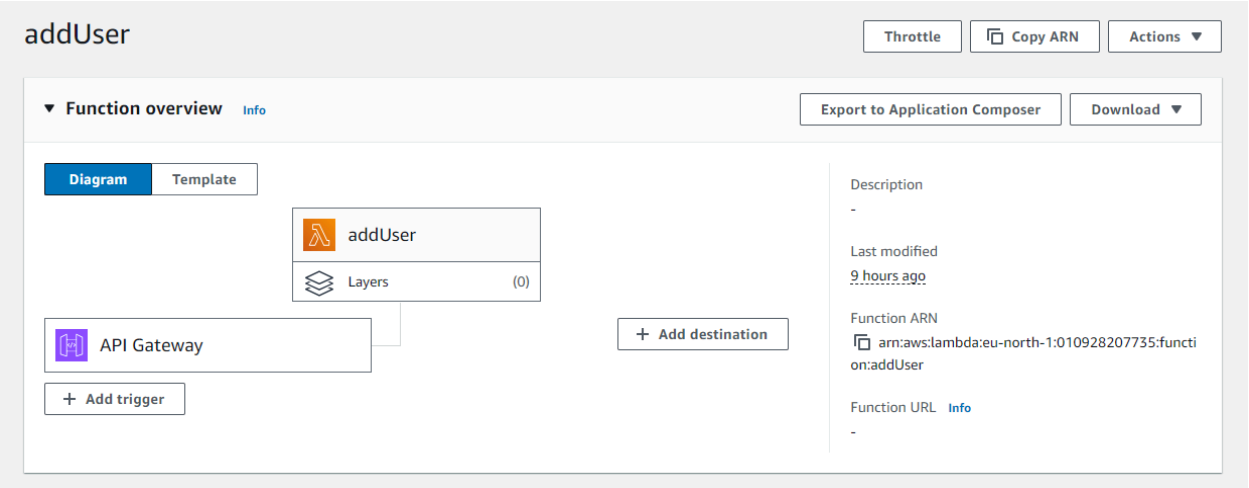


3. Select **Author from scratch**.

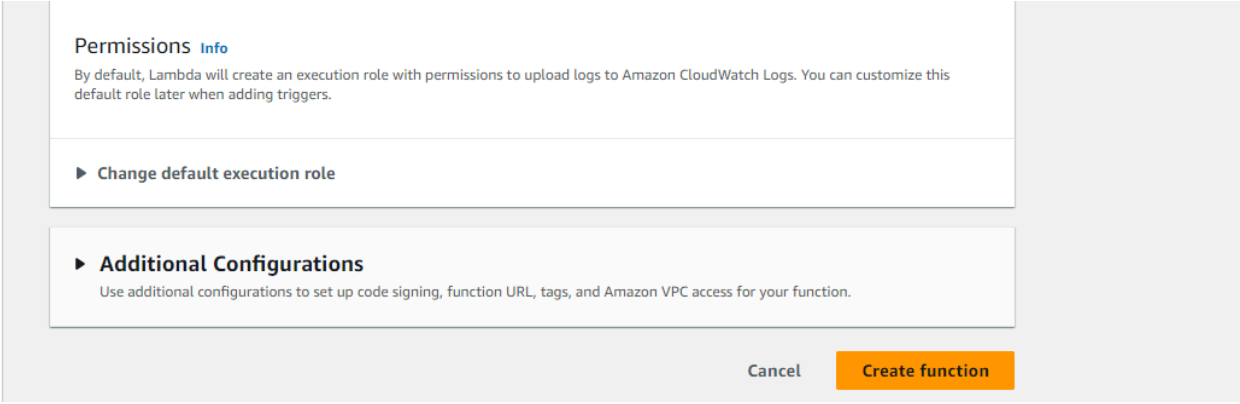


Name: Atharva Patil Class: D15C Roll No: 39

4. **Function Name**: `addUser`
5. **Runtime**: Node.js 20.x



6. Click **Create function**.



7. In the function code editor, replace the default code with the following:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
// Lambda handler function
export const handler = async (event) => {
// Extract parameters from the incoming event (from API Gateway)
const { userID, name, email } = JSON.parse(event.body);
const command = new PutCommand({
TableName: "Users",
Item: {
UserID: userID,
Name: name,
Email: email,
},
});
// Insert data into DynamoDB
const response = await docClient.send(command);
// Return a response for API Gateway
return {
statusCode: 200,
body: JSON.stringify({
```

message: "User added successfully",

data: response,

}),

};

};

Environment Vari × index.mjs ×

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
// Lambda handler function
export const handler = async (event) => {
// Extract parameters from the incoming event (from API Gateway)
const { userID, name, email } = JSON.parse(event.body);
const command = new PutCommand({
TableName: "Users",
Item: {
UserID: userID,
Name: name,
Email: email,
},
});
// Insert data into DynamoDB
const response = await docClient.send(command);
// Return a response for API Gateway
return {
statusCode: 200,
body: JSON.stringify({
message: "User added successfully",
data: response,
}),
```

8. Add DynamoDBFullAccess policy to the addUser Function.
Go to configuration → addUser-role→policies

Permissions policies (2) Info

Simulate | Remove | Add permissions

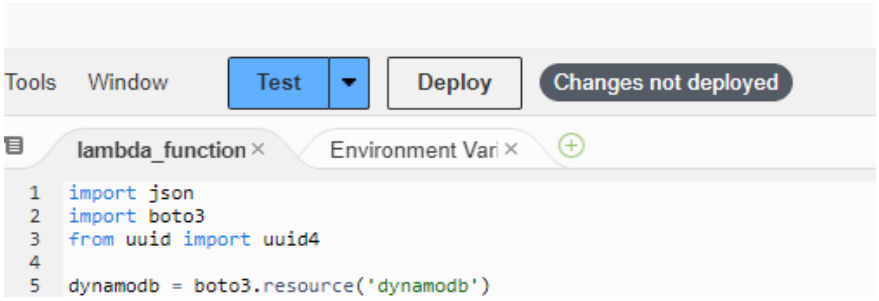
You can attach up to 10 managed policies.

Search | Filter by Type: All types | 1

Policy name	Type	Attached entities
AmazonDynamoDBFullAccess	AWS managed	2
AWSLambdaBasicExecutionRole...	Customer managed	1

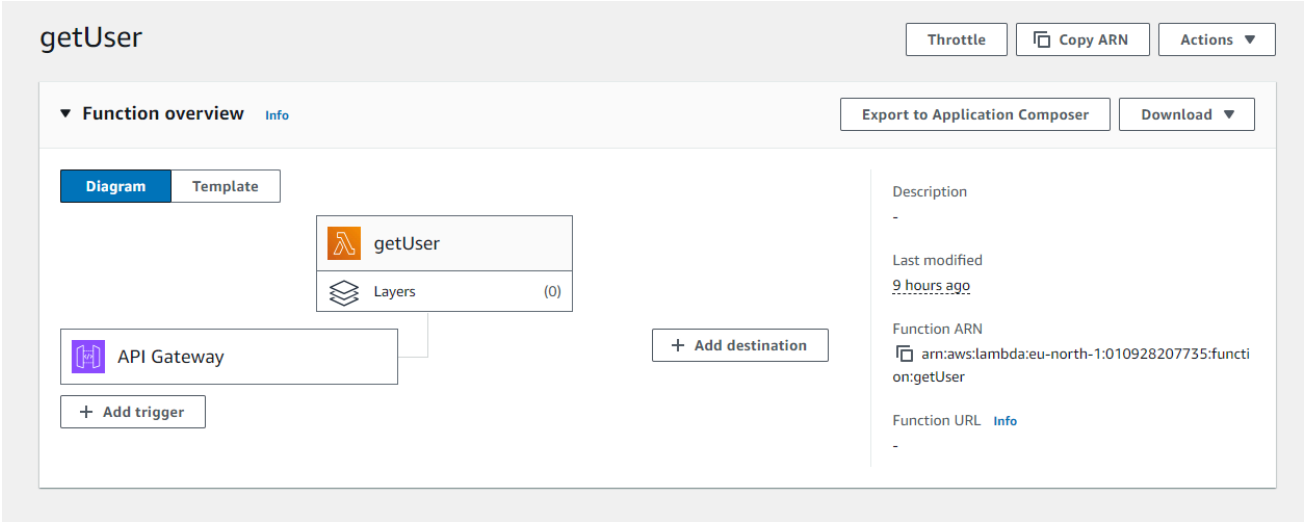
Name: Atharva Patil Class: D15C Roll No: 39

9. Click Deploy.



Function 2: Get User

1. Create another Lambda function.
2. **Function Name**: `getUser`



Keep everything default except this time select an existing role which we noted earlier

3. Click **Create function**.

4. In the function code editor, replace the default code with the following:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import { DynamoDBDocumentClient, GetCommand } from
"@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

// Lambda handler function

export const handler = async (event) => {

// Extract UserID from the query string parameters of the API Gateway event

const { userID } = event.queryStringParameters;

const command = new GetCommand({

TableName: "Users",

Key: {

UserID: userID, // Use the UserID from the event

},

});

// Get item from DynamoDB

const response = await docClient.send(command);

// Return the item as the API Gateway response

return {

statusCode: 200,
```

```
body: JSON.stringify({
message: "User retrieved successfully",
data: response.Item,
}),
};
}
```

5. Click **Deploy**.

Environment Vari × index.mjs ×

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
// Lambda handler function
export const handler = async (event) => {
// Extract UserID from the query string parameters of the API Gateway event
const { userID } = event.queryStringParameters;
const command = new GetCommand({
TableName: "Users",
Key: {
UserID: userID, // Use the UserID from the event
},
});
// Get item from DynamoDB
const response = await docClient.send(command);
// Return the item as the API Gateway response
return {
statusCode: 200,
body: JSON.stringify({
message: "User retrieved successfully",
data: response.Item,
}),
```

Step 3: Test the Lambda Section:
addUser: Navigate to the test section of the addUser function page and. Paste this event json, in the event json code console, for testing the addUser lambda functions


```
{
"body": "{\"userID\":\"001\", \"name\":\"Alice Smith\", \"email\":\"alice@example.com\"}",
"headers": {
"Content-Type": "application/json"
},
"httpMethod": "POST",
"isBase64Encoded": false,
"path": "/addUser",
"queryStringParameters": null,
"requestContext": {
"httpMethod": "POST",
"requestId": "example-request-id"
},
"resource": "/addUser",
"stageVariables": null
}
```

Name: Atharva Patil Class: D15C Roll No: 39

Event JSON Format JSON

```
{
"body": "{\"userID\":\"001\", \"name\":\"Alice Smith\", \"email\":\"alice@example.com\"}",
"headers": {
"Content-Type": "application/json"
},
"httpMethod": "POST",
"isBase64Encoded": false,
"path": "/addUser",
"queryStringParameters": null,
"requestContext": {
"httpMethod": "POST",
"requestId": "example-request-id"
},
"resource": "/addUser",
"stageVariables": null
}
```

16:3 JSON Spaces: 2

Cancel Invoke Save

Executing function: succeeded (logs)

Details

Test event Info

CloudWatch Logs Live Tail Save Test

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

Create new event Edit saved event

Event name

MyEventName

Check table for addition:

UserID *(String)*	Email	Name
001	alice@exa...	Alice Smith

Name: Atharva Patil Class: D15C Roll No: 39

GetUser Navigate to the test section of getUser function and add the event json


```
{
 "queryStringParameters": {
 "userID": "001"
 },
 "httpMethod": "GET",
 "path": "/getUser",
 "headers": {
 "Content-Type": "application/json"
 },
 "requestContext": {
 "httpMethod": "GET",
 "requestId": "example-request-id",
 "path": "/getUser"
 },
 "resource": "/getUser",
 "stageVariables": null
}
```

Name: Atharva Patil Class: D15C Roll No: 39

Event JSON

```
{

"queryStringParameters": {
"userID": "001"
},
"httpMethod": "GET",
"path": "/getUser",
"headers": {
"Content-Type": "application/json"
},
"requestContext": {
"httpMethod": "GET",
"requestId": "example-request-id",
"path": "/getUser"
},
"resource": "/getUser",
"stageVariables": null
}
```

Code | Test | Monitor | Configuration | Aliases | Versions

Executing function: succeeded (logs)

Details

Test event Info

CloudWatch Logs Live Tail | Save | Test

Step 4: Set Up API Gateway

1. **Navigate to API Gateway.**

Networking & Content Delivery

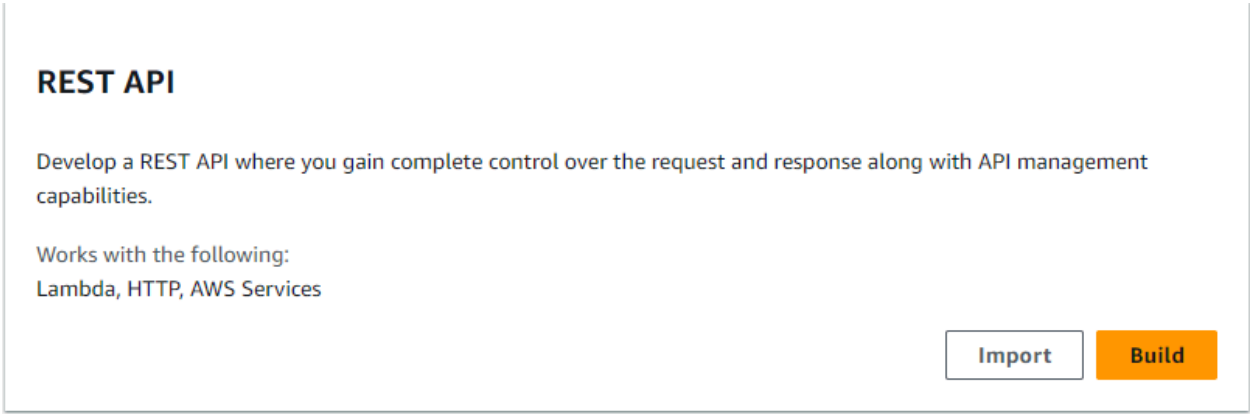
Amazon API Gateway

create, maintain, and secure APIs at any scale

Amazon API Gateway helps developers to create and manage APIs to back-end systems running on Amazon EC2, AWS Lambda, or any publicly addressable web service. With Amazon API Gateway, you can generate custom client SDKs for your APIs, to connect your back-end systems to mobile, web, and server applications or services.

2. **Click on Create API.**

3. **Choose REST API and select Build.**



4. **API Name: `users`**
5. **Click Create API.**

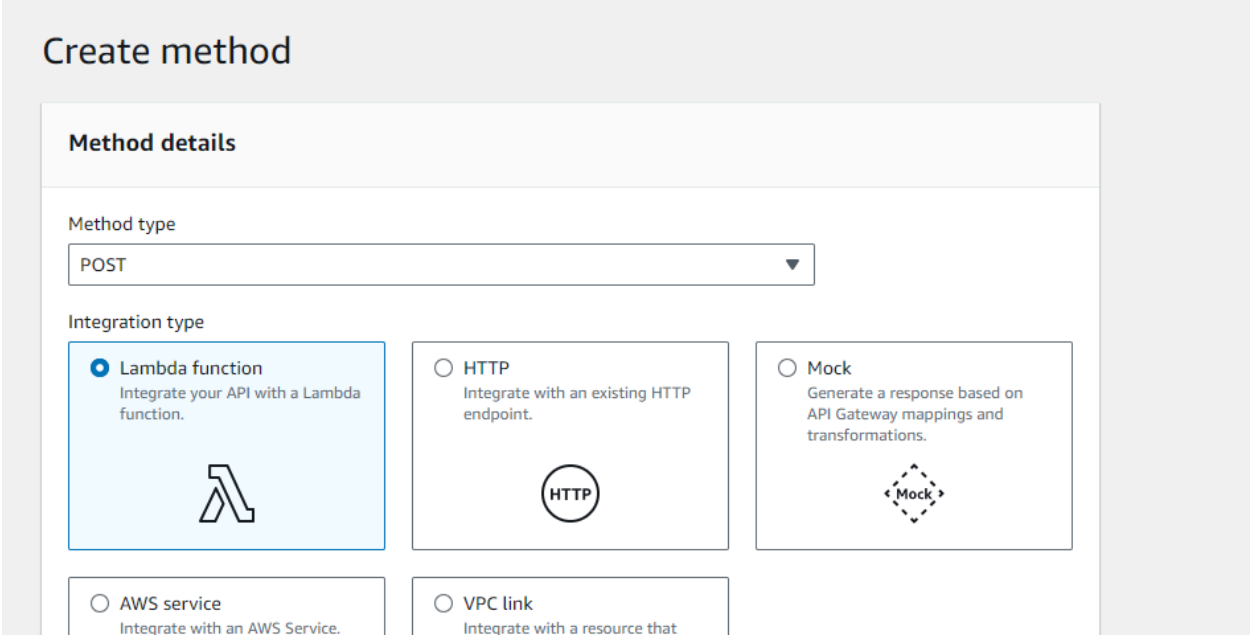
Create POST Method for Adding User

1. **Keep everything else default**

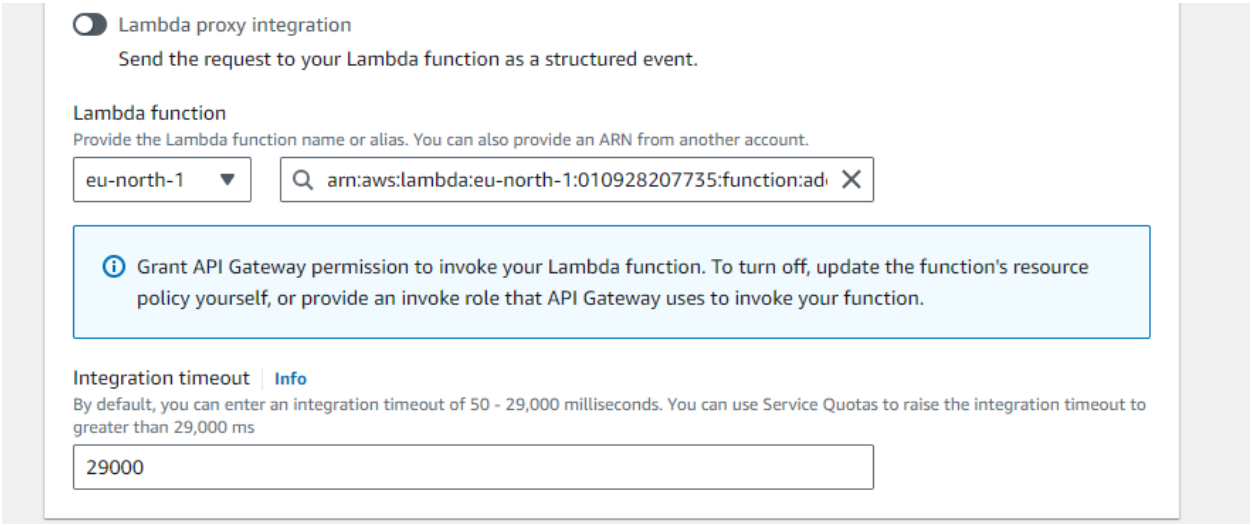
2. **Click Actions > Create Method > select POST > click the checkmark.**

Name: Atharva Patil Class: D15C Roll No: 39

3. **Integration type: Lambda Function.**



4. **Select your addUser function.**



Name: Atharva Patil Class: D15C Roll No: 39

Click on Test Add a new json

Request body

```
1  {
2   "body": "{\"userID\":\"002\", \"name\":\"Alice Smmmmmmith\",
        \"email\":\"aliceeeee@example.com\"}",
3   "headers": {
4   "Content-Type": "application/json"
5   },
6   "httpMethod": "POST",
7   "isBase64Encoded": false,
8   "path": "/addUser",
9   "queryStringParameters": null,
10  "requestContext": {
11  "httpMethod": "POST",
12  "requestId": "example-request-id"
13  },
14  "resource": "/addUser",
15  "stageVariables": null
16  }
17
```

{

"body": "{\"userID\":\"002\", \"name\":\"Alice Smmmmmmith\", \"email\":\"aliceeeee@example.com\"}",

"headers": {

"Content-Type": "application/json"

},

"httpMethod": "POST",

"isBase64Encoded": false,

"path": "/addUser",

"queryStringParameters": null,

"requestContext": {

"httpMethod": "POST",

"requestId": "example-request-id"

},

"resource": "/addUser",

"stageVariables": null

}

/ - POST method test results

Request

/

Status

200

Latency ms

1453

Response body

{"statusCode":200,"body":"{\"message\":\"User added successfully\",\"data\":{\"$metadata\":{\"httpStatusCode\":200,\"requestId\":\"57PFSO6CHVE9PH5D43RA53KF7JVV4KQNSO5AEMVJF66Q9ASUAAJG\",\"attempts\":1,\"totalRetryDelay\":0}}}"}

Response headers

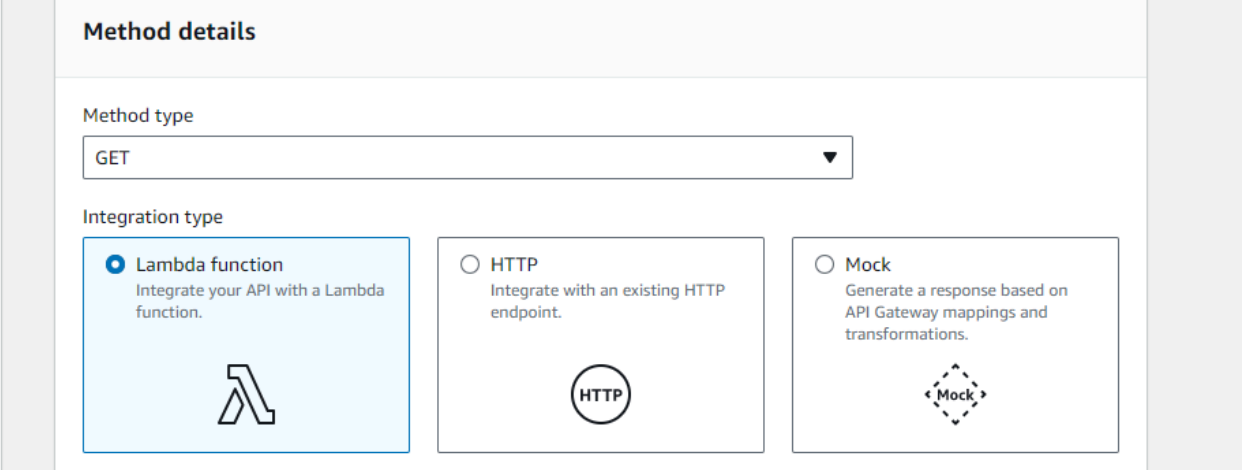
{
"Content-Type": "application/json",
"X-Amzn-Trace-Id": "Root=1-67168c90-fcba048bdb0cc0dfb1cac5c2;Parent=22278c979a87928d;Sampled=0;Lineage=1:25294191:0"

Create GET Method for Retrieving User

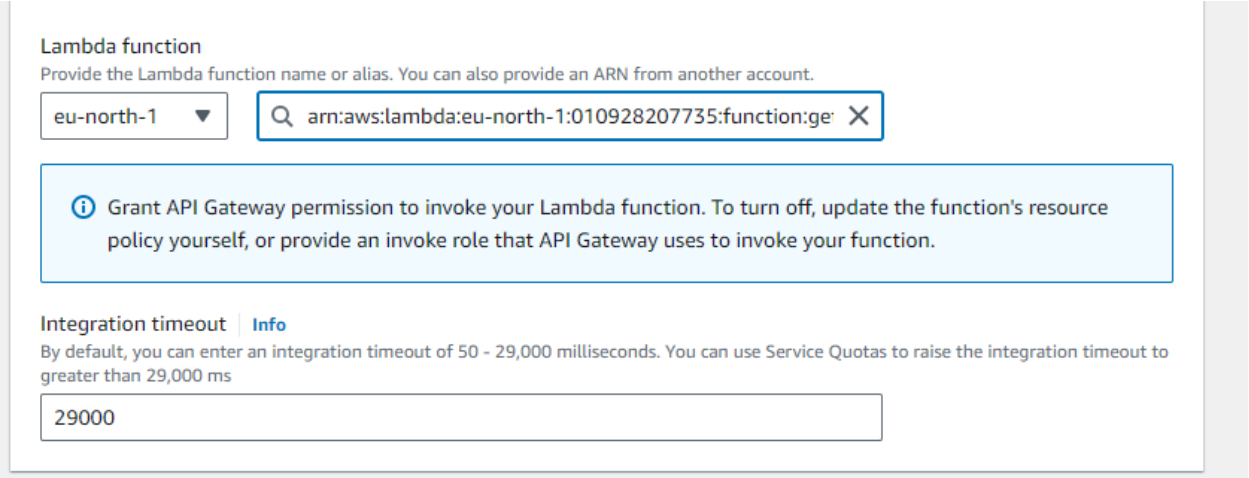
1. **Select the `/users` resource.**

2. **Click Actions > Create Method > select GET > click the checkmark.**

3. **Integration type: Lambda Function.**



4. **Select your `getUser` function.**



5. **Go to the integration request section**
   Click on edit. Scroll down> click on mapping templates> then click on add mapping template Write "application/json" in content type. And write this code in the Template body:

```
{

"queryStringParameters": {

"userID": "$input.params('UserID')"

}

}
```

Mapping templates (1)

application/json

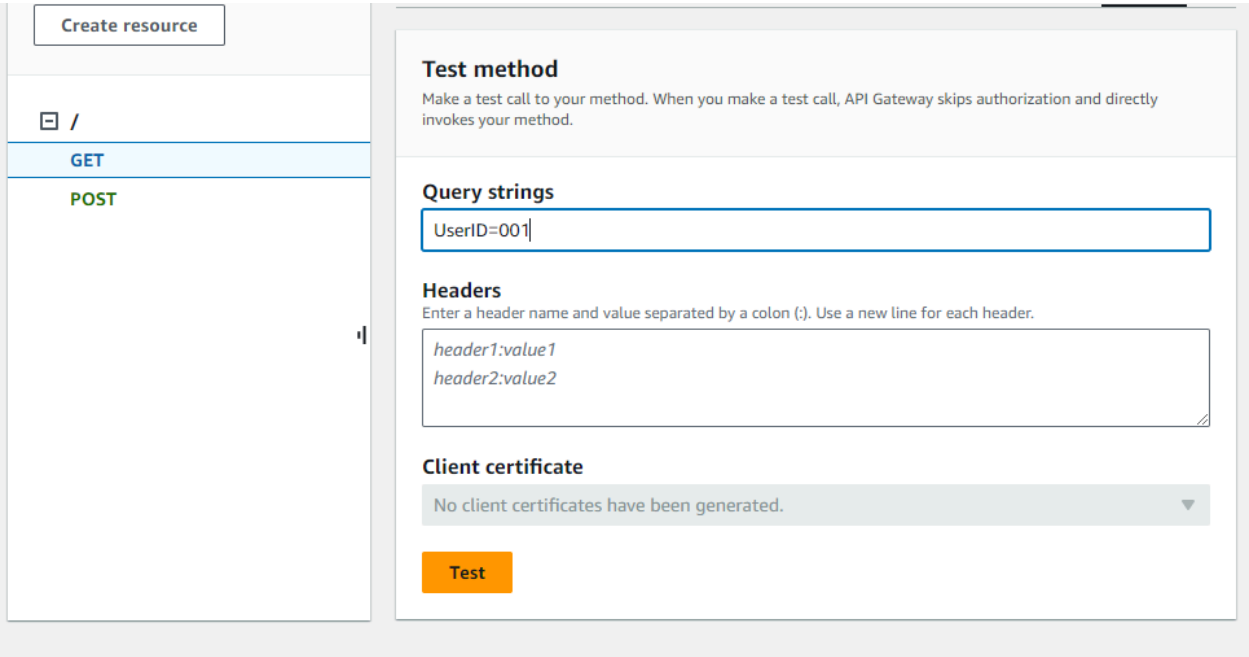
Copied

```
{
"queryStringParameters": {
"userID": "$input.params('UserID')"
}
}
```

6. **Click Save and grant API Gateway permissions to invoke the Lambda function.**

7. **Now navigate to the test section of this GetMethod and paste the query string here: Eg: Myquery string : UserID=001**

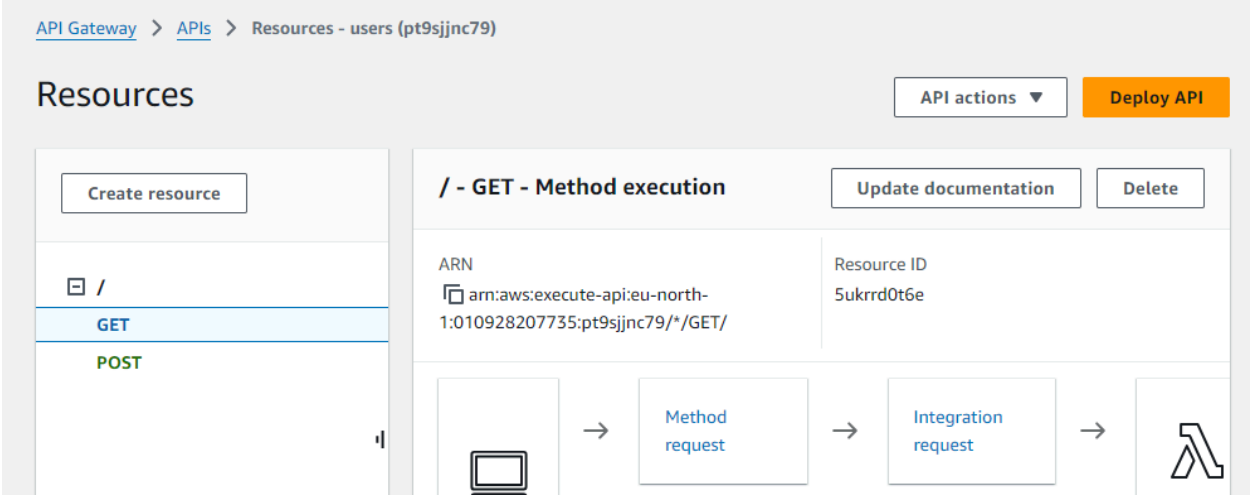
Name: Atharva Patil Class: D15C Roll No: 39



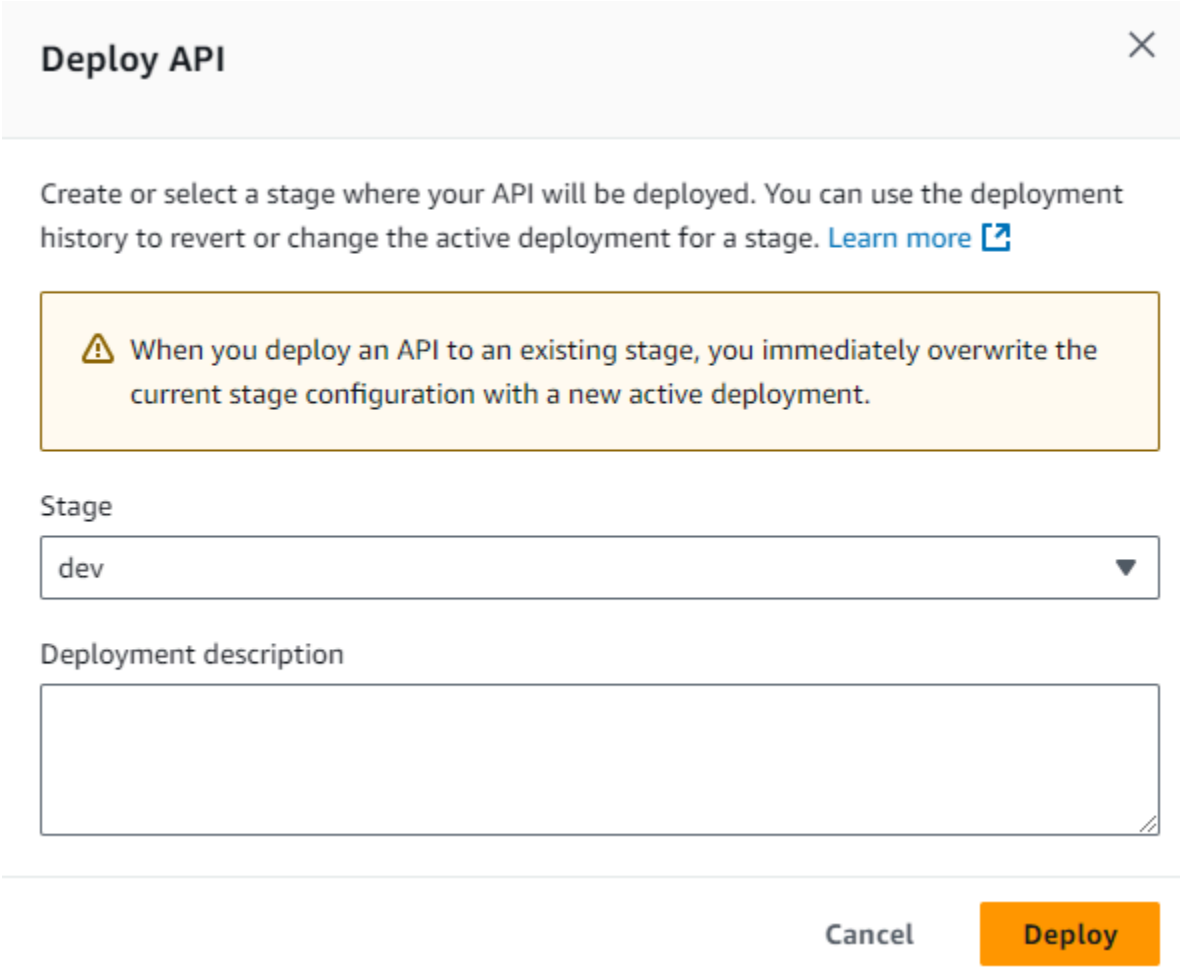
Step 4: Deploy the API

Name: Atharva Patil Class: D15C Roll No: 39

1. **Click on Actions > Deploy API.**

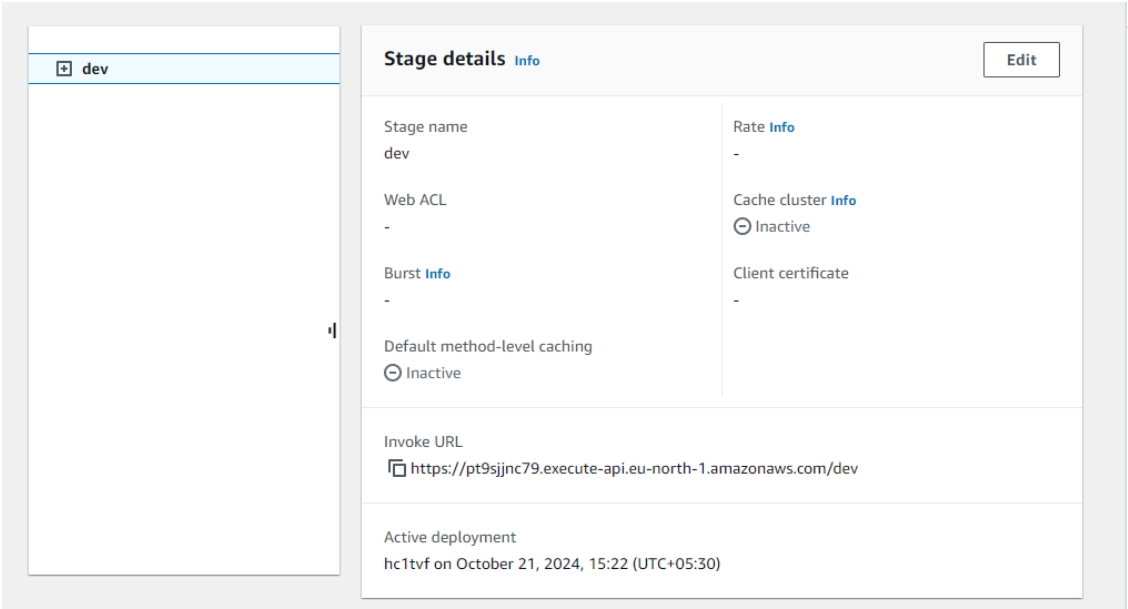


2. **Create a new stage (e.g., dev).**



Name: Atharva Patil Class: D15C Roll No: 39

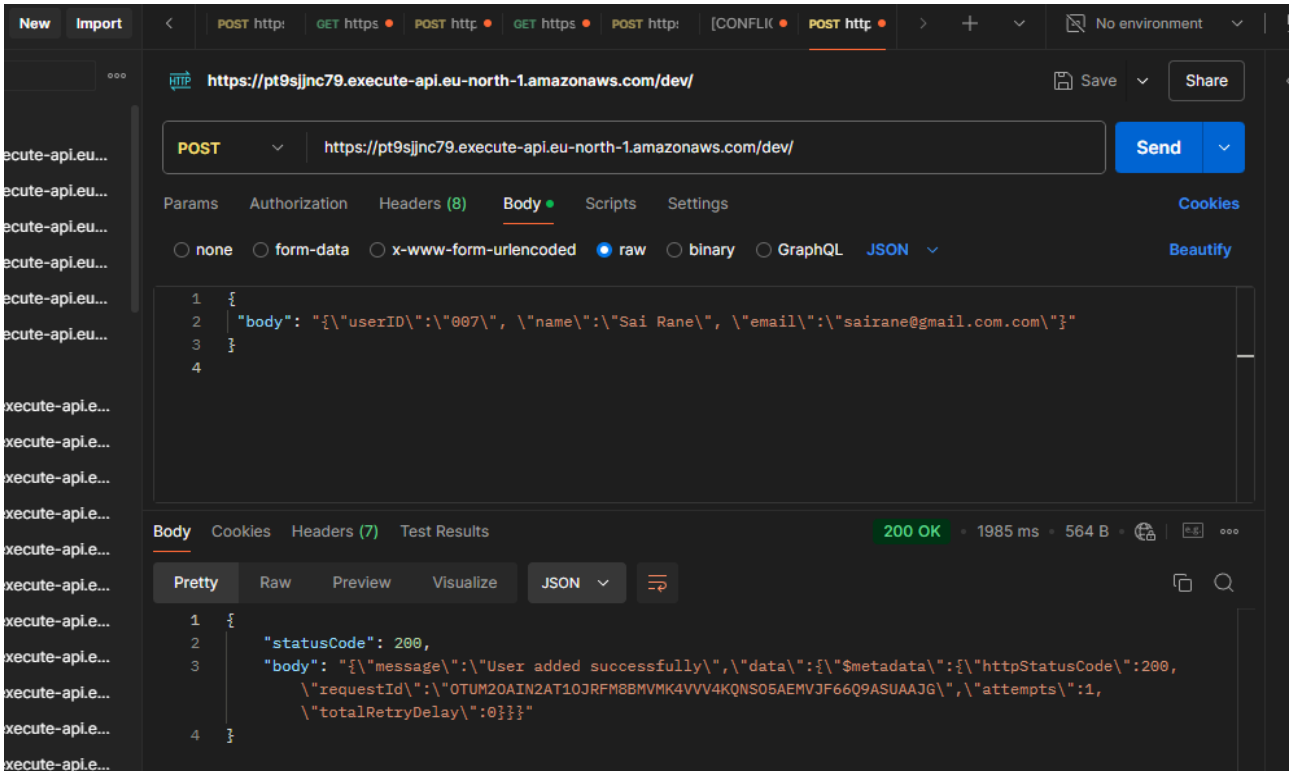
3. **Note the Invoke URL provided after deployment.**



Step 5: Test the API

You can test your API using Postman.

Add User



Name: Atharva Patil Class: D15C Roll No: 39

Check the Database Table:

Items returned (6)

Actions ▼ | Create item

1

UserID *(String)*	Email	Name
001	alice@exa...	Alice Smith
007	sairane@g...	Sai Rane
002	aliceeeee@...	Alice Smmmmmmith
004	atharvapati...	Atharva Patil
005	atharvnika...	Atharv Nikam
006	pratikpatil...	Pratik Patil

Get User

https://pt9sjjnc79.execute-api.eu-north-1.amazonaws.com/dev/?UserID=004

Save | Share

GET https://pt9sjjnc79.execute-api.eu-north-1.amazonaws.com/dev/?UserID=007 Send

Params • Authorization Headers (8) Body • Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
UserID	007		
Key	Value	Description	

Body Cookies Headers (7) Test Results 200 OK • 1966 ms • 493 B

Pretty Raw Preview Visualize JSON


```
{
    "statusCode": 200,
    "body": "{\"message\":\"User retrieved successfully\",\"data\":{\"UserID\":\"007\",\"Name\":\"Sai Rane\",\"Email\":\"sairane@gmail.com.com\"}}"
}
```