

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."

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**.

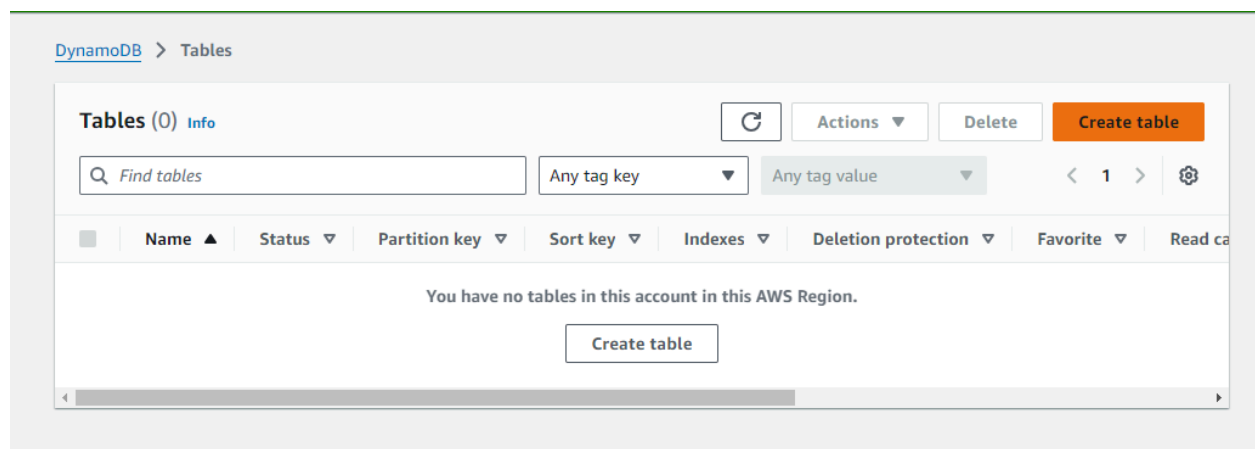


Table Name: Users**Partition key:** UserID

The screenshot shows the AWS IAM console interface for a table named 'Users'. On the left, there's a sidebar with a 'Users' link and a star icon. The main content area has a light blue header with a warning message: 'data automatically so that you can restore to any given second in the preceding 35 days. Additional charges apply. [Learn more](#)'. Below this is an 'Edit PITR' button. The 'General information' section displays the following details:

General information Info	
Partition key UserID (String)	Sort key -
Capacity mode <u>Provisioned</u>	Table status ✔ Active
Alarms ✔ No active alarms	Point-in-time recovery (PITR) Info ⊖ Off
Resource-based policy Info ⊖ Not active	

At the bottom, there is a section for 'Additional info' with a right-pointing arrow.

Table Created Successfully

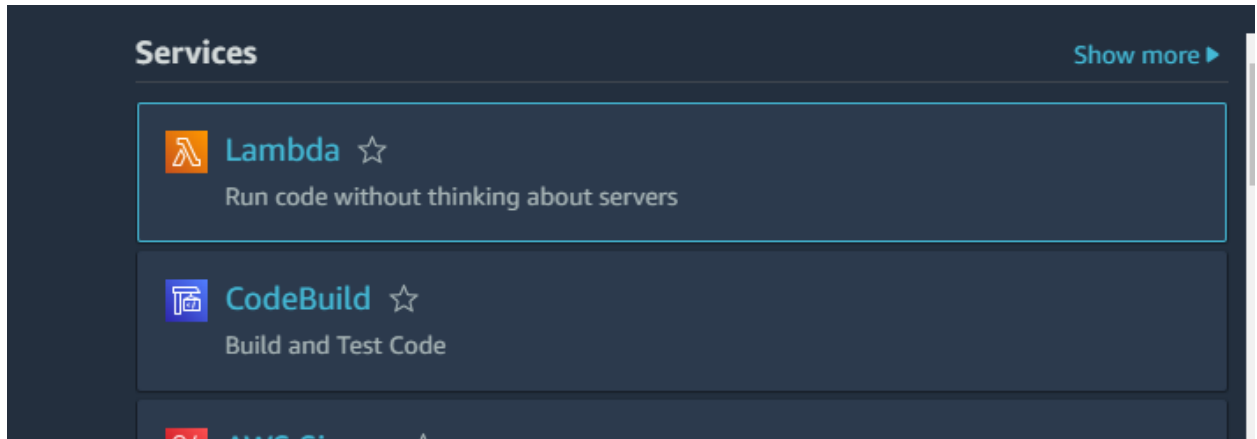
The screenshot shows the AWS IAM console 'Tables' page. The breadcrumb navigation is 'DynamoDB > Tables'. The page title is 'Tables (1) [Info](#)'. There are buttons for 'Actions', 'Delete', and 'Create table'. A search bar contains 'Find tables'. Below the search bar, there are filters for 'Any tag key' and 'Any tag value'. A table lists the details of the 'Users' table:

	Name ▲	Status ▼	Partition key ▼	Sort key ▼	Indexes ▼	Deletion protection ▼	Favorite ▼	Read ca
<input type="checkbox"/>	Users	✔ Active	UserID (S)	-	0	⊖ Off	☆	Provisio

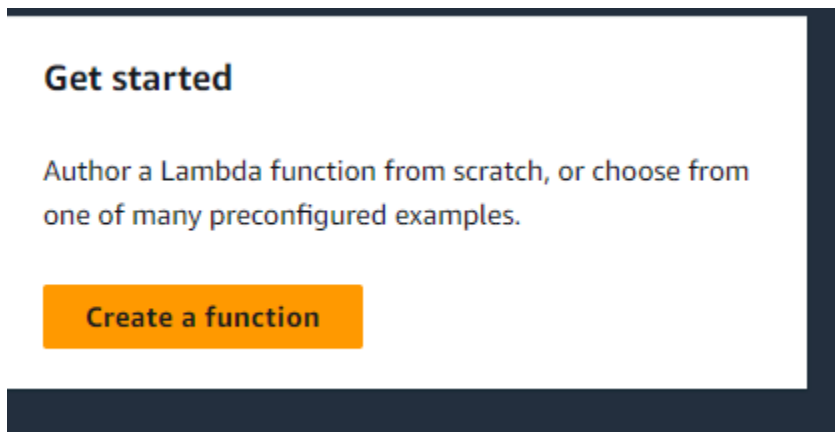
Step 2: Write Lambda Functions

Function 1: Add User

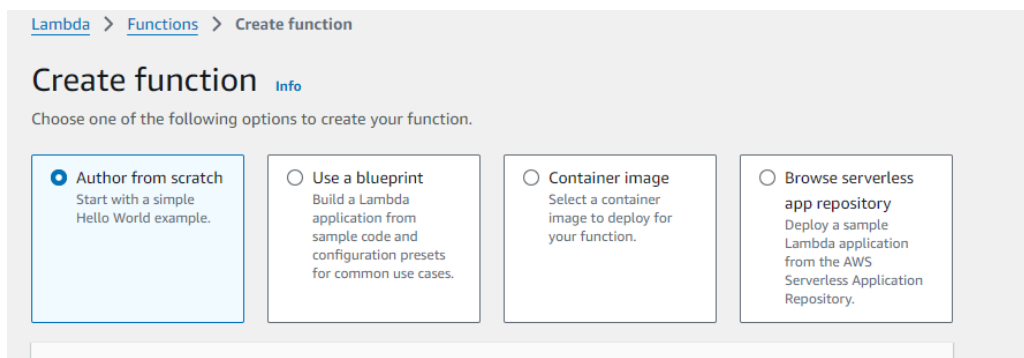
1. Navigate to **AWS Lambda**.



2. Click on **Create Function**.



3. Select **Author from scratch**.



4. **Function Name:** addUser

5. **Runtime:** Node.js 20.x

The screenshot shows the 'addUser' function overview in the AWS Lambda console. At the top, there are buttons for 'Throttle', 'Copy ARN', and 'Actions'. Below this is a tabbed interface with 'Function overview' selected. On the left, there's a 'Diagram' tab showing a visual representation of the function being triggered by an 'API Gateway' and having 'Layers' attached. On the right, there's a 'Template' tab and a 'Description' section. The 'Description' section contains the following information:

- Description: -
- Last modified: 9 hours ago
- Function ARN: `arn:aws:lambda:eu-north-1:010928207735:function:addUser`
- Function URL: [Info](#)

At the bottom of the 'Diagram' tab, there are buttons for '+ Add trigger' and '+ Add destination'.

6. Click **Create function**.

The screenshot shows the 'Create function' page in the AWS Lambda console. It has a 'Permissions' section with an 'Info' link. The text states: 'By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.' Below this is a button labeled 'Change default execution role'. There is also an 'Additional Configurations' section with a right-pointing arrow. The text states: 'Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.' At the bottom right, there are two buttons: 'Cancel' and '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({
```

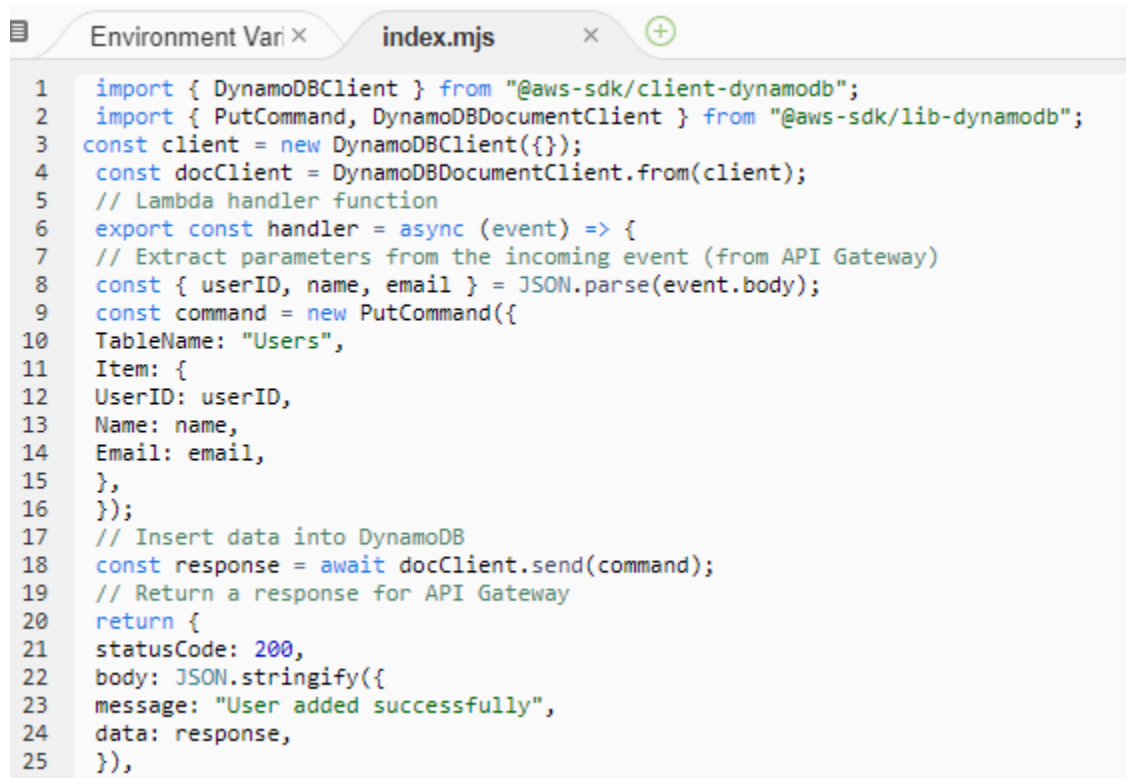
message: "User added successfully",

data: response,

}},

};

};

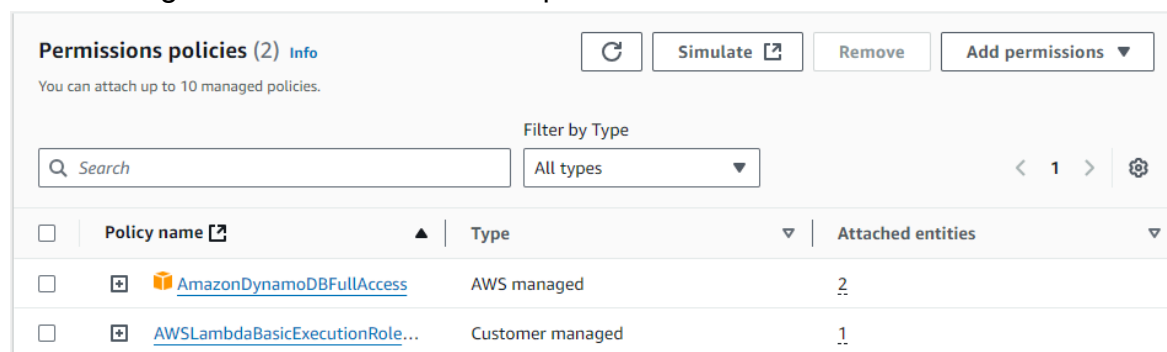




```

1  import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
2  import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";
3  const client = new DynamoDBClient({});
4  const docClient = DynamoDBDocumentClient.from(client);
5  // Lambda handler function
6  export const handler = async (event) => {
7    // Extract parameters from the incoming event (from API Gateway)
8    const { userID, name, email } = JSON.parse(event.body);
9    const command = new PutCommand({
10     TableName: "Users",
11     Item: {
12       UserID: userID,
13       Name: name,
14       Email: email,
15     },
16   });
17   // Insert data into DynamoDB
18   const response = await docClient.send(command);
19   // Return a response for API Gateway
20   return {
21     statusCode: 200,
22     body: JSON.stringify({
23       message: "User added successfully",
24       data: response,
25     }),
  
```

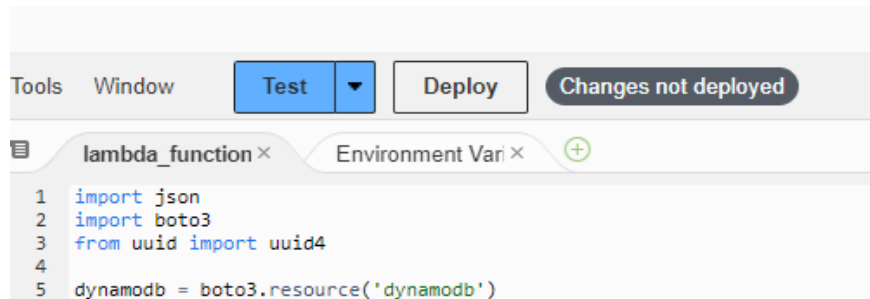
8. Add DynamoDBFullAccess policy to the addUser Function.

Go to configuration → addUser-role→policies



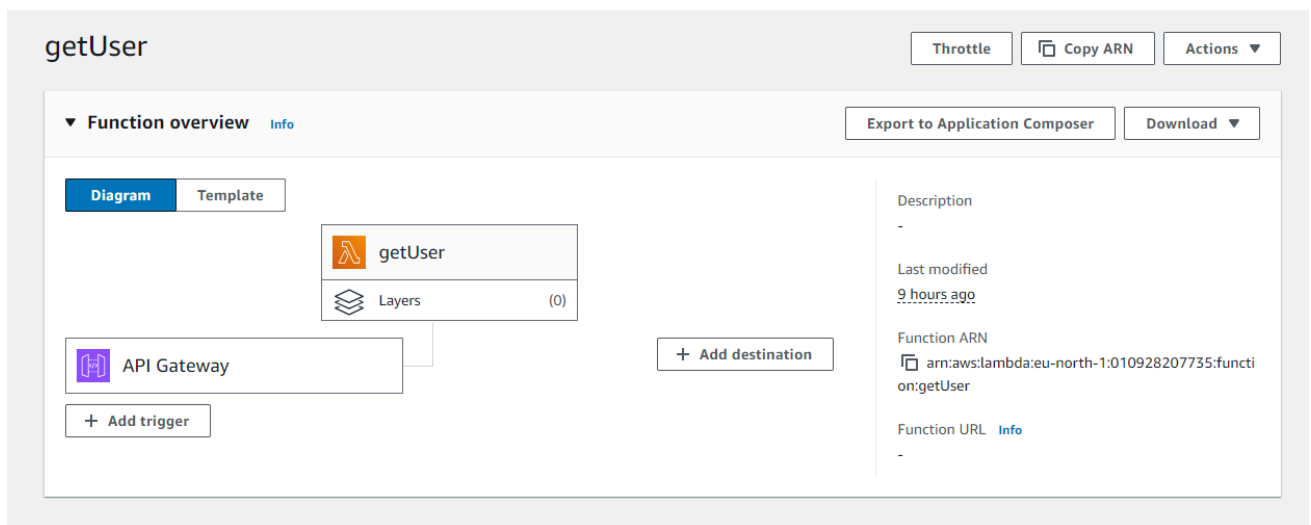
Permissions policies (2) Info			
You can attach up to 10 managed policies.			
Filter by Type			
Q Search		All types	
<input type="checkbox"/>	Policy name ?	Type	Attached entities
<input type="checkbox"/>	 AmazonDynamoDBFullAccess	AWS managed	2
<input type="checkbox"/>	 AWSLambdaBasicExecutionRole...	Customer managed	1

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

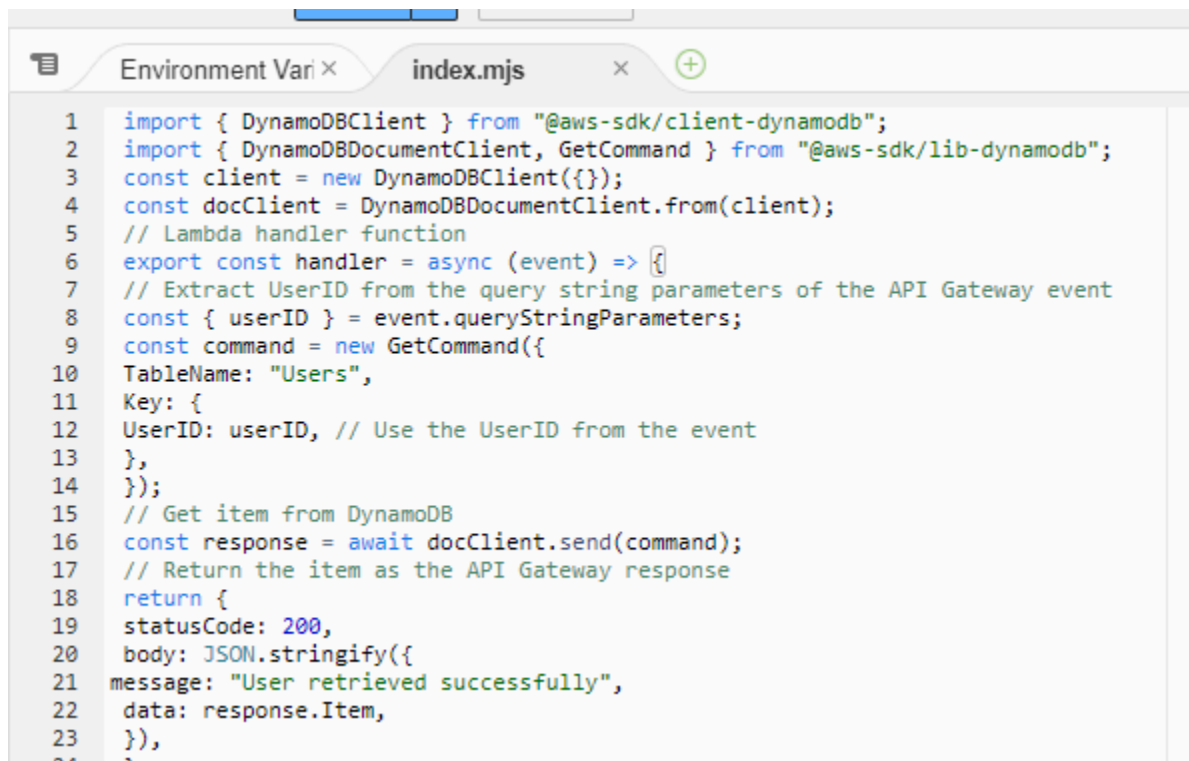
return {

statusCode: 200,
```



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

5. Click **Deploy**.



```
Environment Vari x index.mjs x +  
1 import { DynamoDBClient } from "@aws-sdk/client-dynamodb";  
2 import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";  
3 const client = new DynamoDBClient({});  
4 const docClient = DynamoDBDocumentClient.from(client);  
5 // Lambda handler function  
6 export const handler = async (event) => {  
7   // Extract UserID from the query string parameters of the API Gateway event  
8   const { userID } = event.queryStringParameters;  
9   const command = new GetCommand({  
10    TableName: "Users",  
11    Key: {  
12      UserID: userID, // Use the UserID from the event  
13    },  
14  });  
15  // Get item from DynamoDB  
16  const response = await docClient.send(command);  
17  // Return the item as the API Gateway response  
18  return {  
19    statusCode: 200,  
20    body: JSON.stringify({  
21      message: "User retrieved successfully",  
22      data: response.Item,  
23    }),  
24  };
```

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
}
```

Event JSON Format JSON

```
1 {
2   "body": "{\"userID\":\"001\", \"name\":\"Alice Smith\", \"email\":\"alice@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 }
```

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

Check table for addition:

<input type="checkbox"/>	UserID (String) ▾	Email ▾	Name ▾
<input type="checkbox"/>	001	alice@exa...	Alice Smith


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  
}
```

Event JSON

```
1 {  
2  
3   "queryStringParameters": {  
4     "userID": "001"  
5   },  
6   "httpMethod": "GET",  
7   "path": "/getUser",  
8   "headers": {  
9     "Content-Type": "application/json"  
10  },  
11  "requestContext": {  
12    "httpMethod": "GET",  
13    "requestId": "example-request-id",  
14    "path": "/getUser"  
15  },  
16  "resource": "/getUser",  
17  "stageVariables": null  
18 }
```

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.

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

ImportBuild

4. API Name: users

5. Click Create API.

APIs (1/1)

Find APIs

< 1 > ⚙

⌂

Delete

Create API

	Name ▲	Description ▼	ID ▼	Protocol ▼	API endpoint type ▼	Created ▼
<input type="radio"/>	users		pt9sjjnc79	REST	Regional	2024-10-21

Create POST Method for Adding User

1. Keep everything else default

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

3. Integration type: Lambda Function.


Create method


Method details


Method type

POST ▼

Integration type

☒ **Lambda function**
Integrate your API with a Lambda function.


☐ **HTTP**
Integrate with an existing HTTP endpoint.


☐ **Mock**
Generate a response based on API Gateway mappings and transformations.


☐ **AWS service**
Integrate with an AWS Service.

☐ **VPC link**
Integrate with a resource that

4. Select your addUser function.


☒ **Lambda proxy integration**
Send the request to your Lambda function as a structured event.

Lambda function

Provide the Lambda function name or alias. You can also provide an ARN from another account.

eu-north-1 ▼

X

 Grant API Gateway permission to invoke your Lambda function. To turn off, update the function's resource policy yourself, or provide an invoke role that API Gateway uses to invoke your function.

Integration timeout | [Info](#)

By default, you can enter an integration timeout of 50 - 29,000 milliseconds. You can use Service Quotas to raise the integration timeout to greater than 29,000 ms


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	Latency ms
/	1453

Status
200

Response body

```
{
  "statusCode": 200,
  "body": {
    "\message\": "\User added successfully\\"",
    "\data\": {
      "\$metadata\": {
        "\httpStatusCode\": 200,
        "\requestId\": "\57PFS06CHVE9PH5D43RA53KF7JVV4KQNS05AEMVJF66Q9ASUAAJG\\"",
        "\attempts\": 1,
        "\totalRetryDelay\": 0
      }
    }
  }
}
```

Response headers

```
{
  "Content-Type": "application/json",
  "X-Amzn-Trace-Id": "Root=1-67168c90-fc
ba048bdb0cc0dfb1cac5c2;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.


Method details

Method type
GET

Integration type

☒ **Lambda function**
Integrate your API with a Lambda function.


☐ **HTTP**
Integrate with an existing HTTP endpoint.


☐ **Mock**
Generate a response based on API Gateway mappings and transformations.


4. Select your getUser function.

Lambda function
Provide the Lambda function name or alias. You can also provide an ARN from another account.

eu-north-1

Grant API Gateway permission to invoke your Lambda function. To turn off, update the function's resource policy yourself, or provide an invoke role that API Gateway uses to invoke your function.

Integration timeout [Info](#)
By default, you can enter an integration timeout of 50 - 29,000 milliseconds. You can use Service Quotas to raise the integration timeout to greater than 29,000 ms

29000

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

```
1  {
2    "queryStringParameters": {
3      "userID": "$input.params('UserID')"
```

4 }
5 }
6

✓ Copied

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

Create resource

/

GET

POST

Test method
Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method.

Query strings

Headers
Enter a header name and value separated by a colon (:). Use a new line for each header.

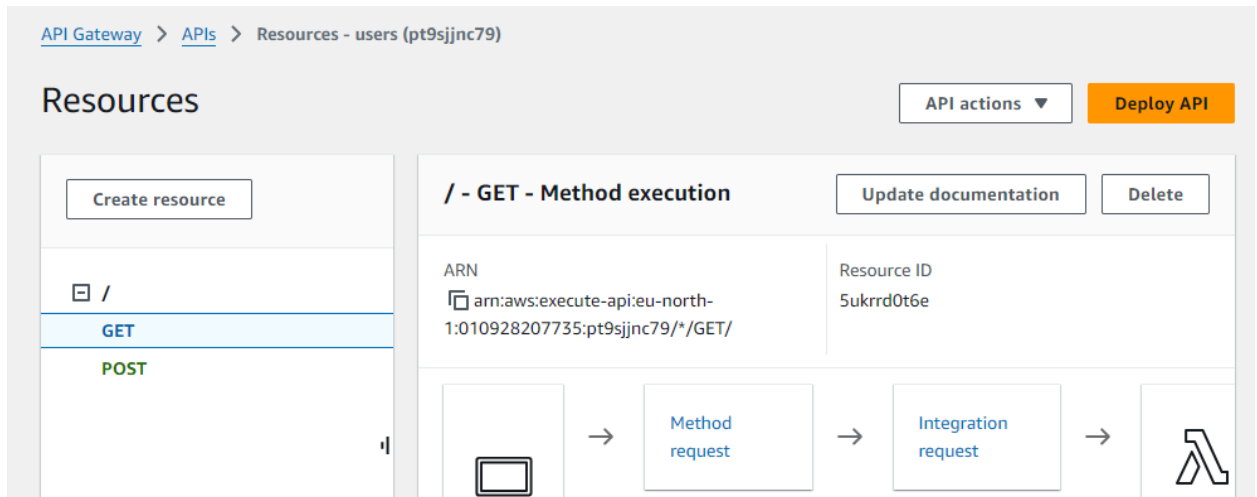
header1:value1
header2:value2

Client certificate
No client certificates have been generated.

Test

/ - GET method test results	
Request	Latency ms
/?UserID=001	1431
Status	
200	
Response body	
<pre>{"statusCode":200,"body":{"message":"User retrieved successfully","data":{"UserID":"001","Name":"Alice Smith","Email":"alice@example.com"}}</pre>	
Response headers	

Step 4: Deploy the API



Deploy API

Create or select a stage where your API will be deployed. You can use the deployment history to revert or change the active deployment for a stage. [Learn more](#)

When you deploy an API to an existing stage, you immediately overwrite the current stage configuration with a new active deployment.

Stage

dev

Deployment description

Cancel

Deploy

3. Note the Invoke URL provided after deployment.

The screenshot shows the AWS API Gateway console. On the left, a list of stages includes 'dev'. The main panel displays the 'Stage details' for the 'dev' stage. The details include:

- Stage name: dev
- Web ACL: -
- Burst: -
- Default method-level caching: Inactive
- Rate: -
- Cache cluster: Inactive
- Client certificate: -
- Invoke URL: <https://pt9sjjnc79.execute-api.eu-north-1.amazonaws.com/dev>
- Active deployment: hc1tvf on October 21, 2024, 15:22 (UTC+05:30)

Step 5: Test the API

You can test your API using Postman.

Add User

The screenshot shows the Postman interface. The request is a POST to `https://pt9sjjnc79.execute-api.eu-north-1.amazonaws.com/dev/`. The body is raw JSON:

```
{
  "body": "{\"userID\":\"007\", \"name\":\"Sai Rane\", \"email\":\"sairane@gmail.com.com\"}"
}
```

The response is a 200 OK status with a response time of 1985 ms and a body size of 564 B. The response body is shown in JSON format:

```
{
  "statusCode": 200,
  "body": {
    "message": "User added successfully",
    "data": {
      "metadata": {
        "statusCode": 200,
        "requestId": "0TUM20AIN2AT10JRFM8BMVMK4VVV4KQNS05AEMVJF66Q9ASUAAJG",
        "attempts": 1,
        "totalRetryDelay": 0
      }
    }
  }
}
```

Check the Database Table:

Items returned (6)			
<div><div>⌂</div><div>Actions ▼</div><div>Create item</div></div>			
<div>< 1 > ⚙️ 🗖️</div>			
<input type="checkbox"/>	UserID (String) ▼	Email ▼	Name ▼
<input type="checkbox"/>	001	alice@exa...	Alice Smith
<input type="checkbox"/>	007	sairane@g...	Sai Rane
<input type="checkbox"/>	002	aliceeeee@...	Alice Smmmmmmmith
<input type="checkbox"/>	004	atharvapati...	Atharva Patil
<input type="checkbox"/>	005	atharvnika...	Atharv Nikam
<input type="checkbox"/>	006	pratikpatil...	Pratik Patil

Get User

https://pt9sjnc79.execute-api.eu-north-1.amazonaws.com/dev/?UserID=004

Save ▼ Share

GET ▼

https://pt9sjnc79.execute-api.eu-north-1.amazonaws.com/dev/?UserID=007

Send ▼

Params •

Authorization

Headers (8)

Body •

Scripts

Settings

Cookies

Query Params

<input checked="" type="checkbox"/>	Key	Value	Description	⋮ Bulk Edit
<input checked="" type="checkbox"/>	UserID	007		
	Key	Value	Description	

Body

Cookies

Headers (7)

Test Results

200 OK • 1966 ms • 493 B • 🌐 🗖️ ⋮

Pretty

Raw

Preview

Visualize

JSON ▼

🔍

```
1 {
2   "statusCode": 200,
3   "body": "{\\\"message\\\":\\\"User retrieved successfully\\\",\\\"data\\\":{\\\"UserID\\\":\\\"007\\\",\\\"Name\\\":\\\"Sai Rane\\\",\\\"Email\\\":\\\"sairane@gmail.com.com\\\"}}}"
4 }
```