😊 AWS API Gateway

Amazon API Gateway is a fully managed service provided by Amazon Web Services (AWS) that allows developers to create, publish, maintain, monitor, and secure APIs at any scale. It acts as a gateway for backend services such as Lambda functions, EC2 instances, or any other web application, allowing them to be exposed to the internet through RESTful APIs or WebSocket APIs.

Here are some key features and functionalities of AWS API Gateway:

1. **API Creation:** Developers can define RESTful APIs or WebSocket APIs using API Gateway's intuitive interface or by importing API definitions in formats like Swagger or OpenAPI.
2. **Integration with Backend Services:** API Gateway can integrate with various backend services including AWS Lambda, Amazon EC2, AWS Fargate, HTTP endpoints, and more, allowing developers to easily connect their APIs to the desired backend.
3. **API Security:** AWS API Gateway provides features for securing APIs such as IAM authentication, Lambda authorizers, OAuth, and custom authorizers. It also supports features like API keys and usage plans for access control and rate limiting.
4. **Monitoring and Analytics:** Developers can monitor API usage, performance, and error rates using AWS CloudWatch metrics and logs. API Gateway also integrates with AWS X-Ray for tracing and analyzing requests.
5. **Scalability and High Availability:** API Gateway is designed to scale automatically to handle any amount of traffic, and it is deployed across multiple availability zones for high availability.
6. **Cost Optimization:** AWS API Gateway offers a pay-as-you-go pricing model, allowing developers to optimize costs based on API usage and traffic patterns. It also provides features like caching to reduce the number of calls to backend services.

😄 Use Cases of API Gateway:

AWS API Gateway can be used in various scenarios across different industries and applications. Here are some common use cases:

1. **Microservices Architecture:** API Gateway is often used in microservices architectures to expose individual services as APIs. Each microservice can have its own API, allowing for better modularity, scalability, and flexibility in managing and deploying services independently.
2. **Mobile and Web Applications:** API Gateway can serve as the backend for mobile and web applications, providing a centralized API layer for accessing backend services. It allows developers to build rich client applications that interact with server-side resources via RESTful APIs or WebSocket APIs.
3. **Serverless Applications:** API Gateway is commonly used in serverless architectures, where it acts as the entry point for serverless functions like AWS Lambda. It enables

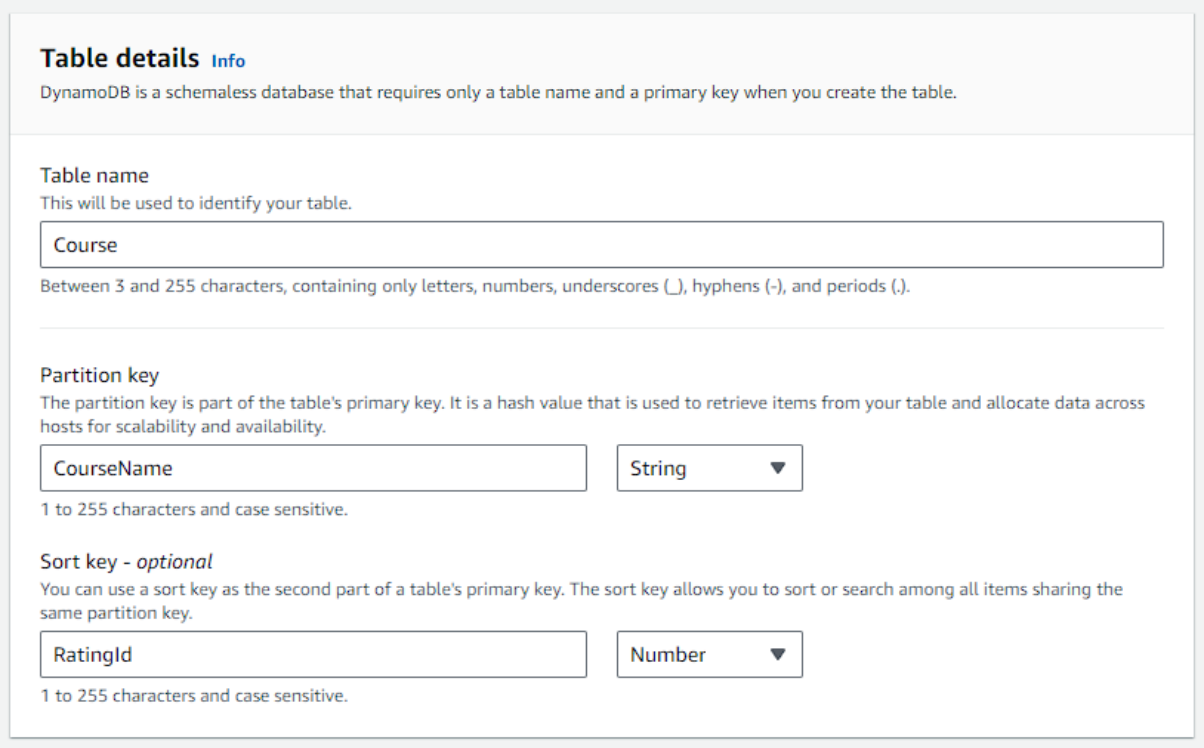
developers to create serverless APIs without managing servers, making it easy to build and deploy event-driven applications at scale.

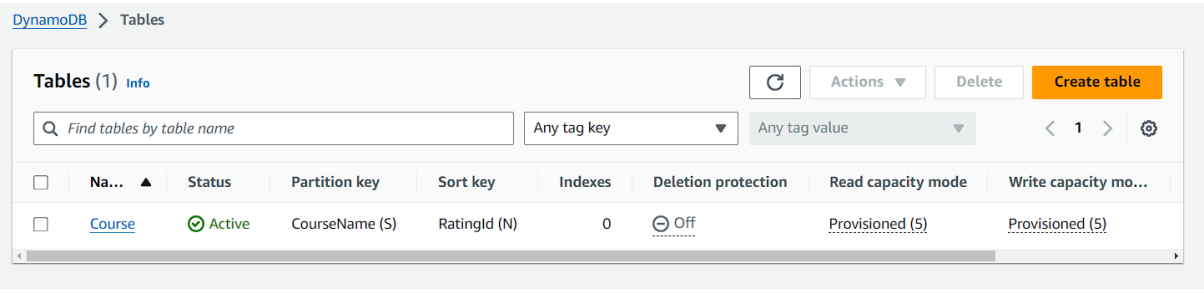
4. **IoT (Internet of Things):** API Gateway can be used to expose APIs for IoT devices, allowing them to securely communicate with backend services in the cloud. It provides features like authentication, authorization, and throttling to control access and ensure the security of IoT data.
5. **Integration with Third-Party Services:** API Gateway can be used to integrate with third-party APIs and services, acting as a proxy for routing requests to external endpoints. This enables organizations to create unified APIs that aggregate data from multiple sources or extend the functionality of existing services.
6. **API Monetization:** Organizations can use API Gateway to monetize their APIs by implementing features like usage plans, API keys, and rate limiting. This allows them to charge developers based on API usage, generate revenue from API consumption, and create new business models around their APIs.
7. **Real-Time Applications:** API Gateway supports WebSocket APIs, making it suitable for building real-time applications such as chat applications, gaming platforms, or collaborative editing tools. WebSocket APIs enable bidirectional communication between clients and servers, allowing for instant updates and notifications.
8. **Content Delivery:** API Gateway can be used to deliver static content such as HTML, CSS, and JavaScript files by integrating with Amazon S3 or other content delivery networks (CDNs). It enables developers to build serverless websites or web applications with low latency and high scalability.

😄 To begin with the Lab:

😎 Step 1: Create DynamoDB table

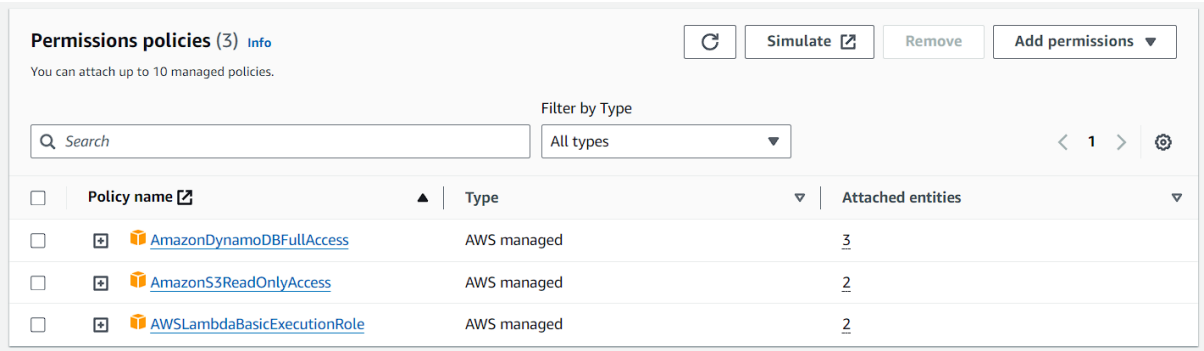
1. Log in to AWS Console. Then navigate to DynamoDB.
2. There you need to create a Table.
3. Now give this table a name, partition key and sort key.
4. Then just create your table.





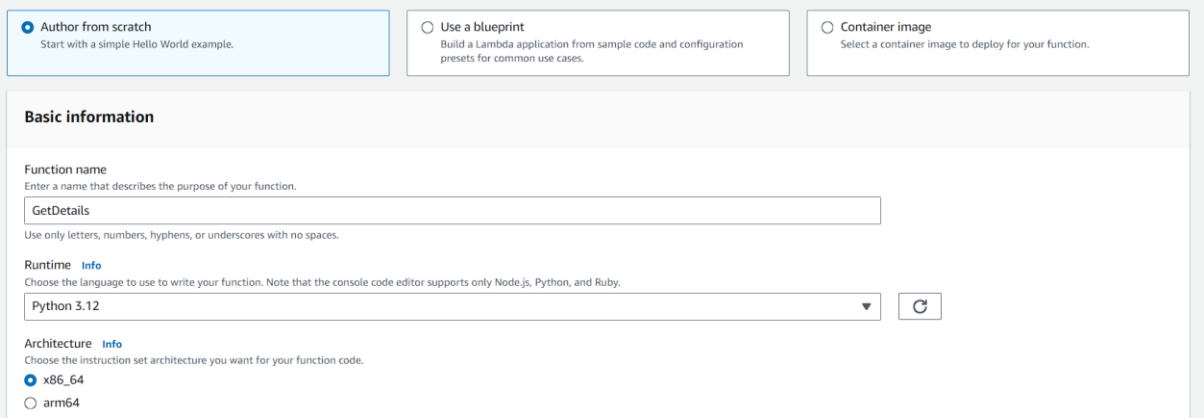
Step 2: Create IAM Role

1. Now navigate to IAM. Then you need to create a role.
2. Now attach these policies with the role and create it.

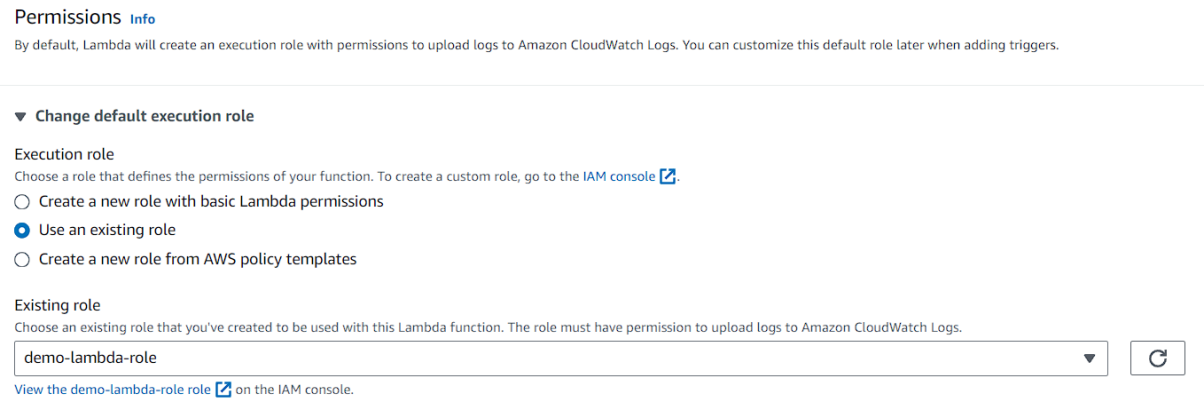


Step 3: Create 2 Lambda functions

1. Now go to Lambda and click on create function.
2. Select author from scratch. Select runtime as Python.



3. Now in the permissions you need to click on use an existing role.
4. Then choose your IAM role which you just created and attach it with the function.



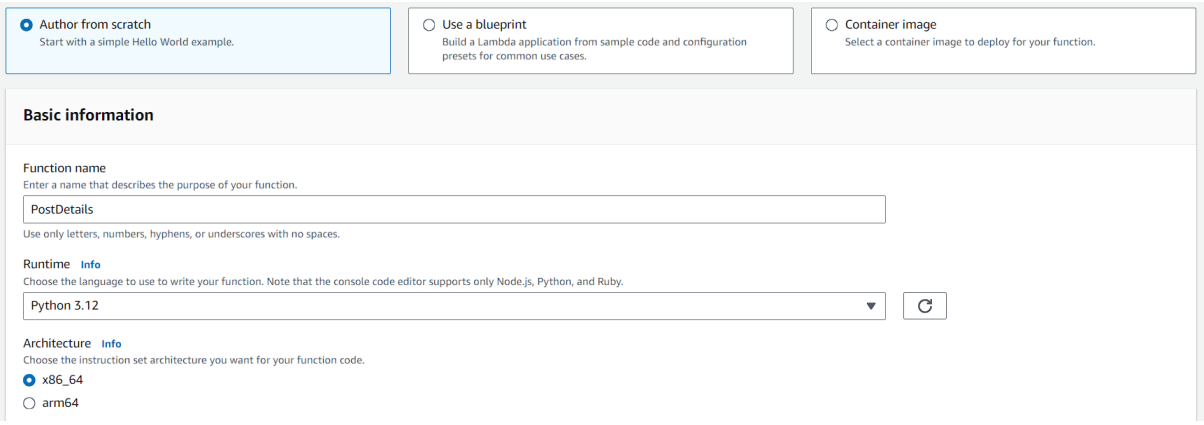
5. Then just create your role.
6. Once your function is created you need to scroll down to the code section and change the code to this. Then click on deploy.

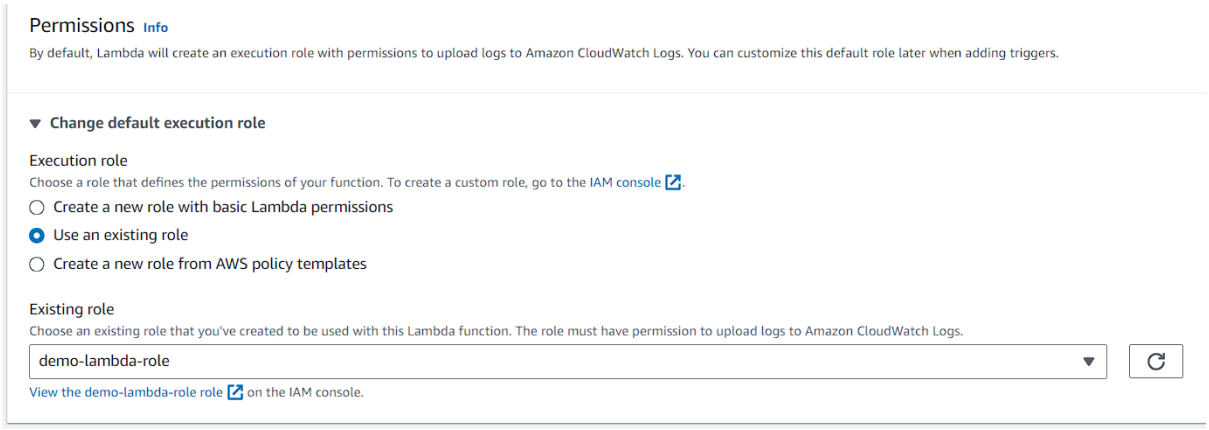
```
import boto3

# Create a DynamoDB client
client = boto3.client('dynamodb', region_name='ap-south-1')

def lambda_handler(event, context):
    # Define input parameters
    input_params = {
        "TableName": "Course"
    }

    # Execute the scan command
    response = client.scan(**input_params)
    print(response)
    # Prepare the response
    reply = {
```

"statusCode": 200,
"body": response['Items'][0]
}

return reply

7. Now you need to another function the details are same as before you need to use the different code here.





8. Now just create your function and scroll down to the code section, use this code and deploy it.

```
import json
import boto3

# Create a DynamoDB client
client = boto3.client('dynamodb', region_name='ap-south-1')

def lambda_handler(event, context):
    # Parse the event body
    eventData = json.loads(event['body'])
    courseName = eventData['courseName']
    ratingId = eventData['ratingId']
    rating = eventData['rating']

    # Define parameters for PutItem command
    params = {
        "Item": {
            "CourseName": {"S": courseName},
            "RatingId": {"N": str(ratingId)},
            "Rating": {"N": str(rating)}
        },
        "TableName": "Course"
    }

    # Execute the PutItem command
    response = client.put_item(**params)

    # Prepare the response
    reply = {
        "statusCode": 200,
        "body": json.dumps("Item added")
```

}

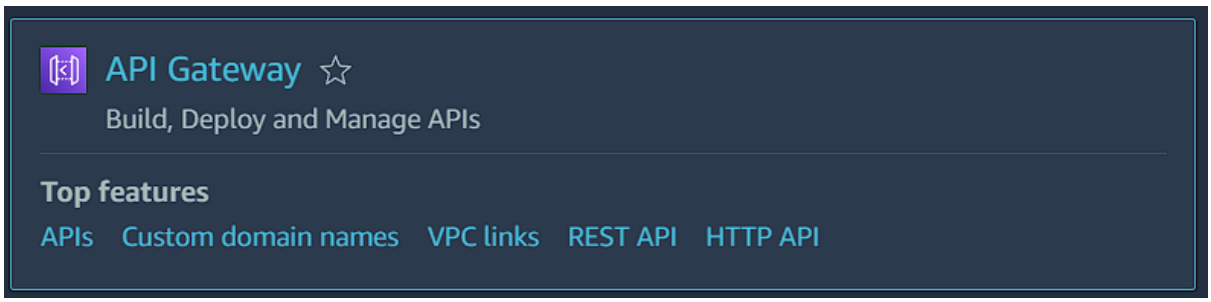
return reply

lambda_function × Environment Vari × Execution results ×

```python
import json
import boto3

# Create a DynamoDB client
client = boto3.client('dynamodb', region_name='ap-south-1')

def lambda_handler(event, context):
    # Parse the event body
    eventData = json.loads(event['body'])
    courseName = eventData['courseName']
    ratingId = eventData['ratingId']
    rating = eventData['rating']

    # Define parameters for PutItem command
    params = {
        "Item": {
            "CourseName": {"S": courseName},
            "RatingId": {"N": str(ratingId)},
            "Rating": {"N": str(rating)}
        },
        "TableName": "Course"
    }

    # Execute the PutItem command
    response = client.put_item(**params)

    # Prepare the response
    reply = {
        "statusCode": 200,
        "body": json.dumps("Item added")
    }

    return reply

```

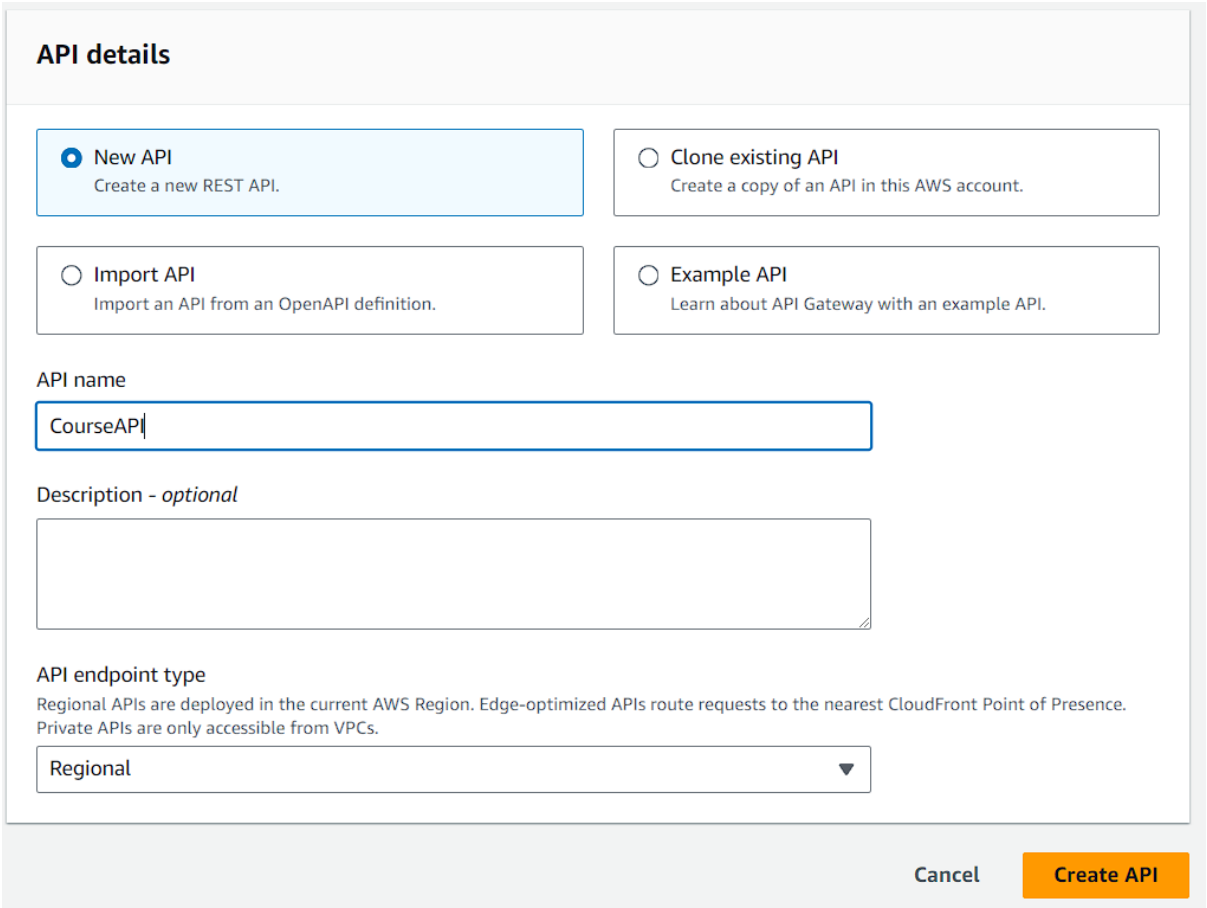
Step 3: Create API Gateway

1. Now you need to navigate to API Gateway. Choose this service accordingly.

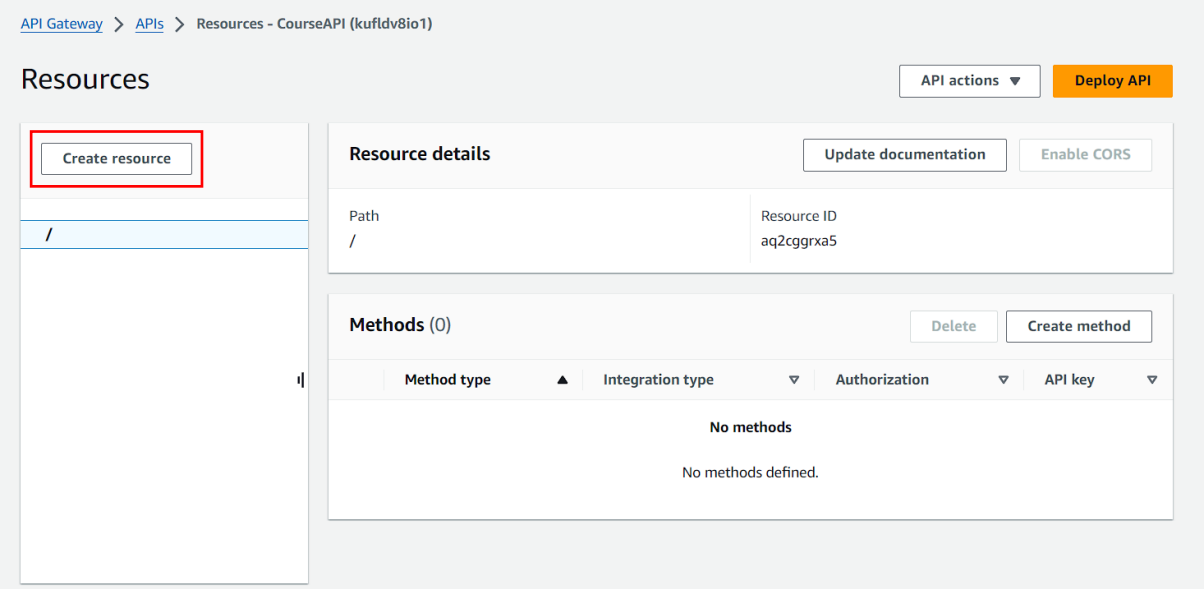


2. Now you need to choose API, which API do you want. There you need to choose REST API. Click on Build.

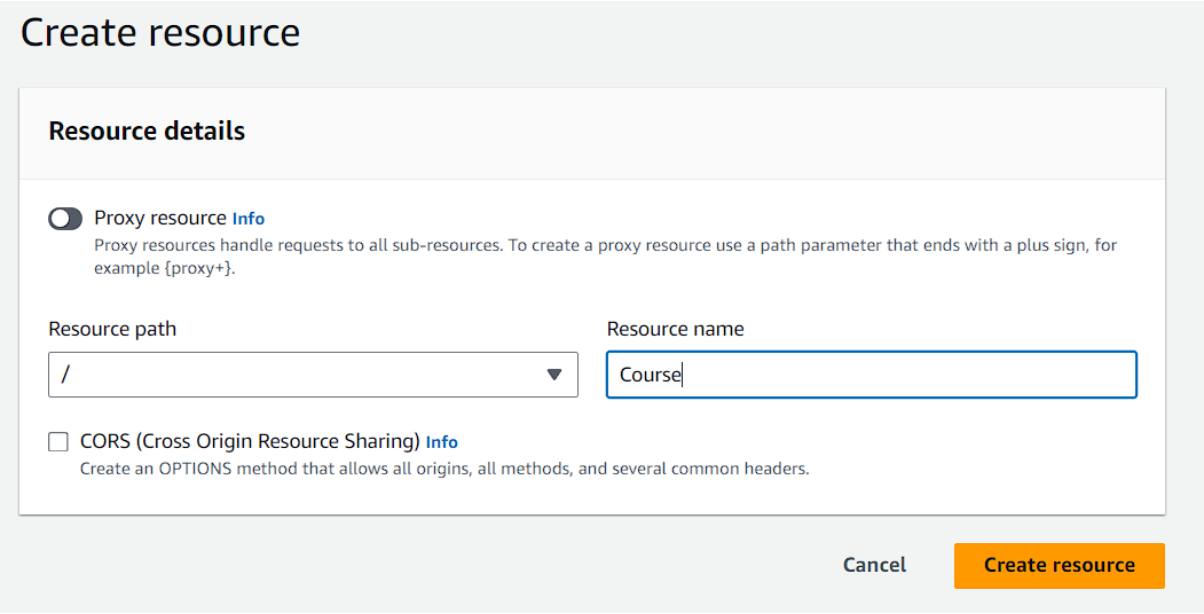
3. Now you need to select New API. Then give it a name and select end point to Regional.
4. Then just hit on Create API.



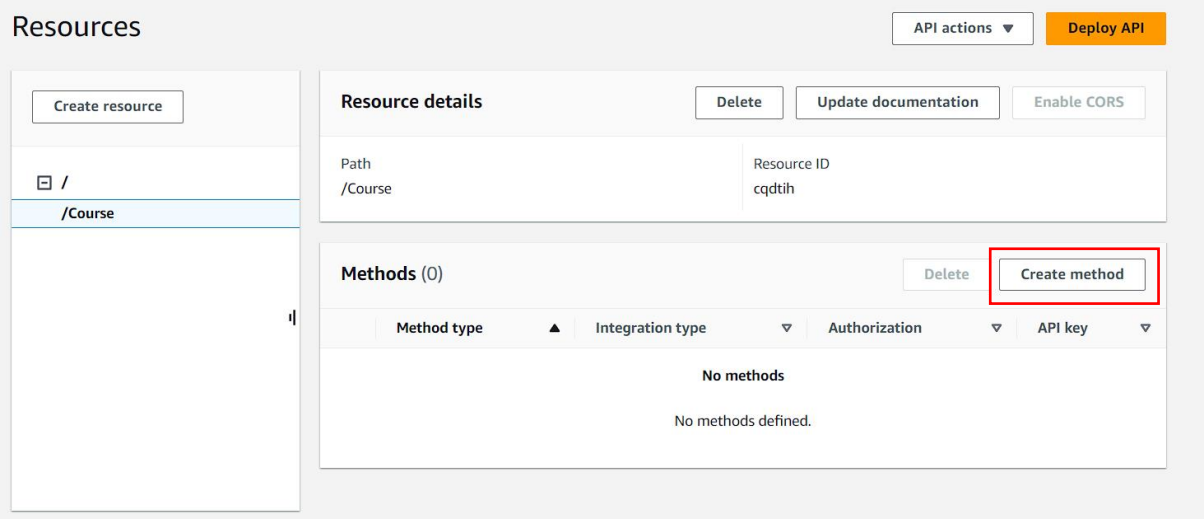
5. Once you have this in place, you can now create your resources.
6. Click on create resource.



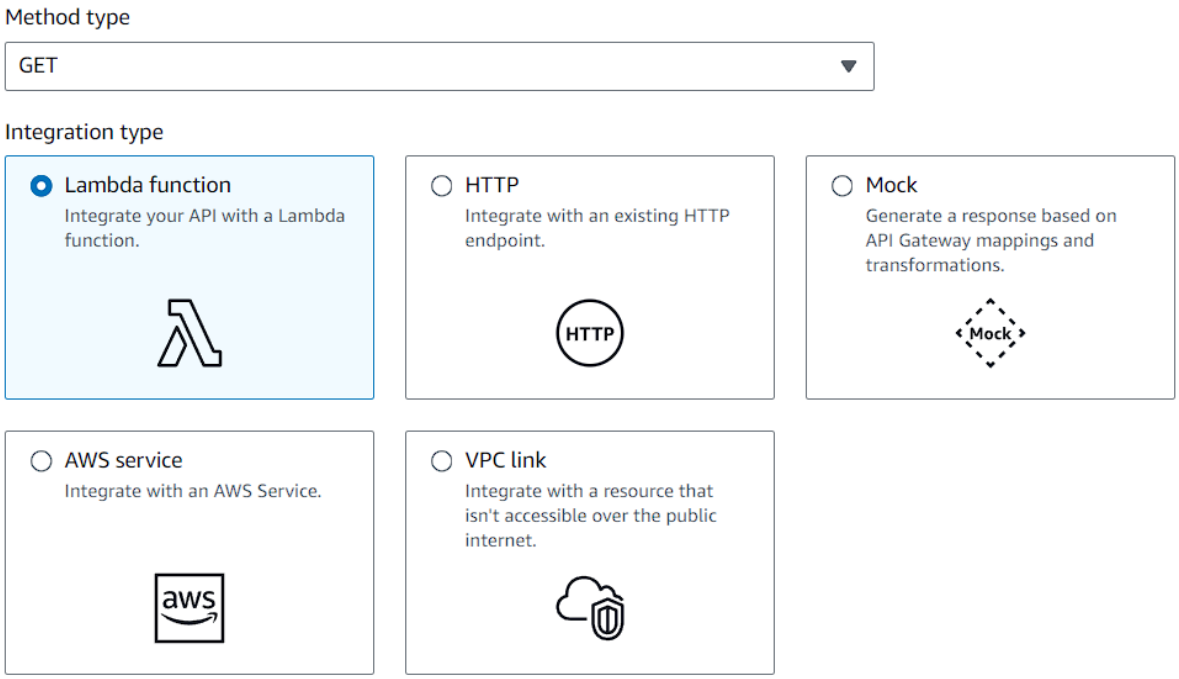
7. Here you just need to give the Resource name and click on create resource.



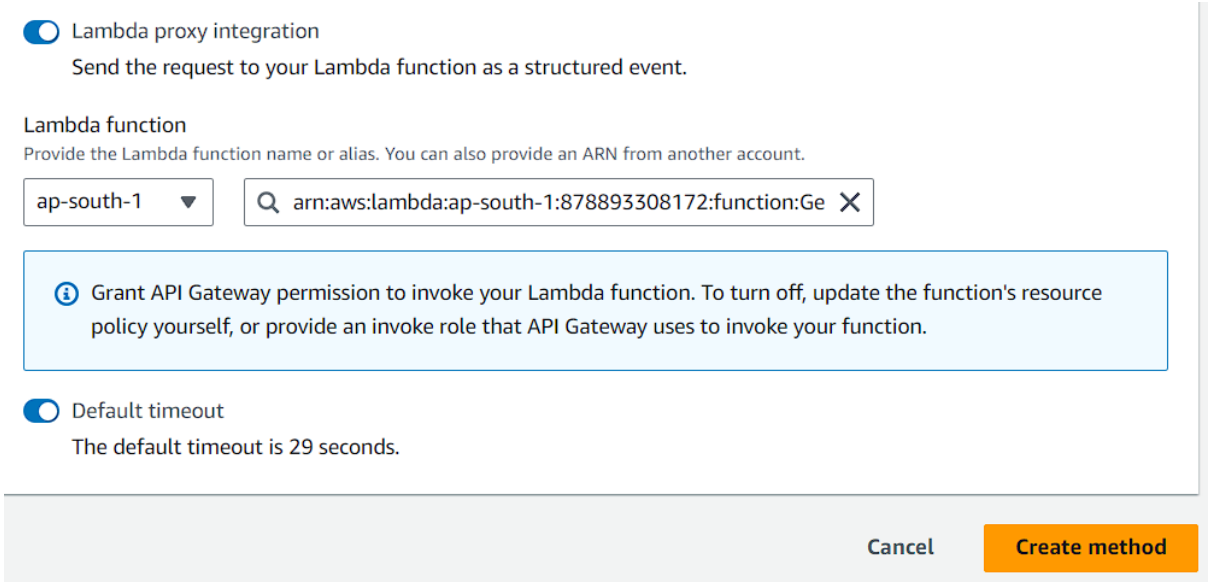
8. Now you can define your methods. Click on create method.



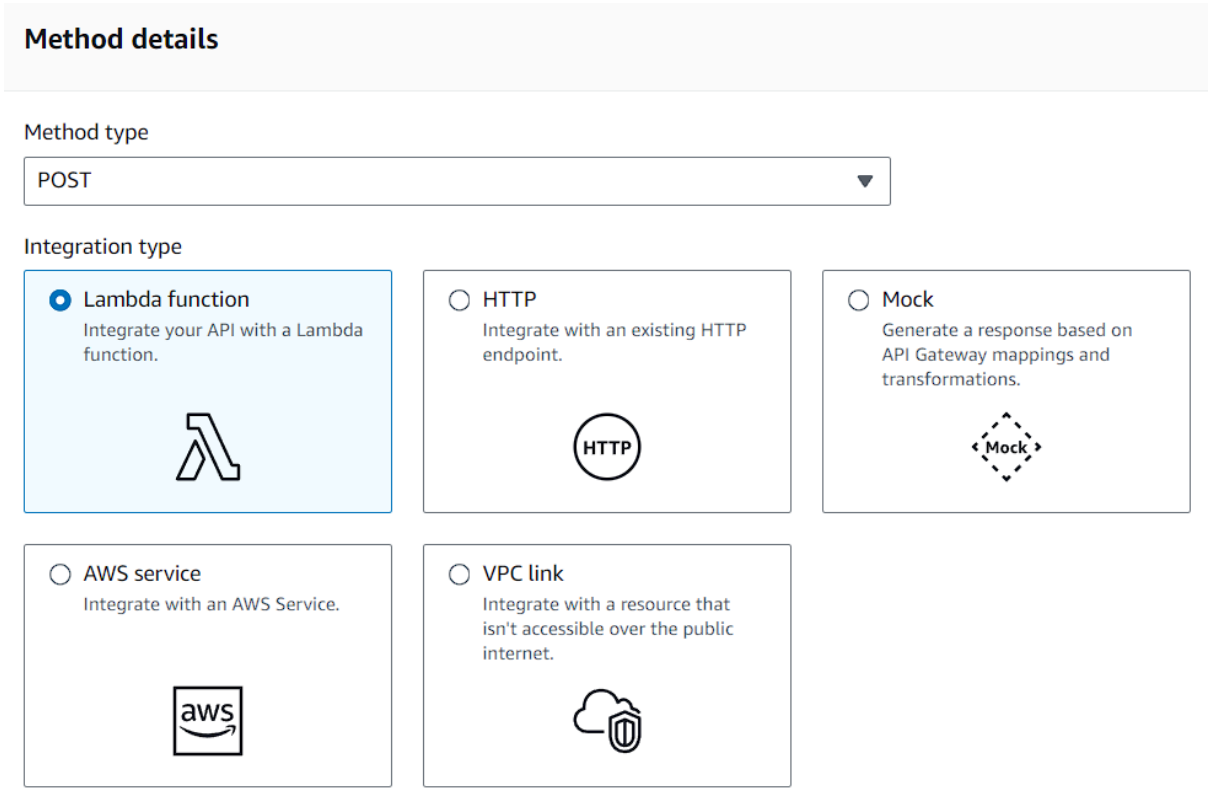
9. Now you need to choose the method type as GET.
10. The integration type as lambda.

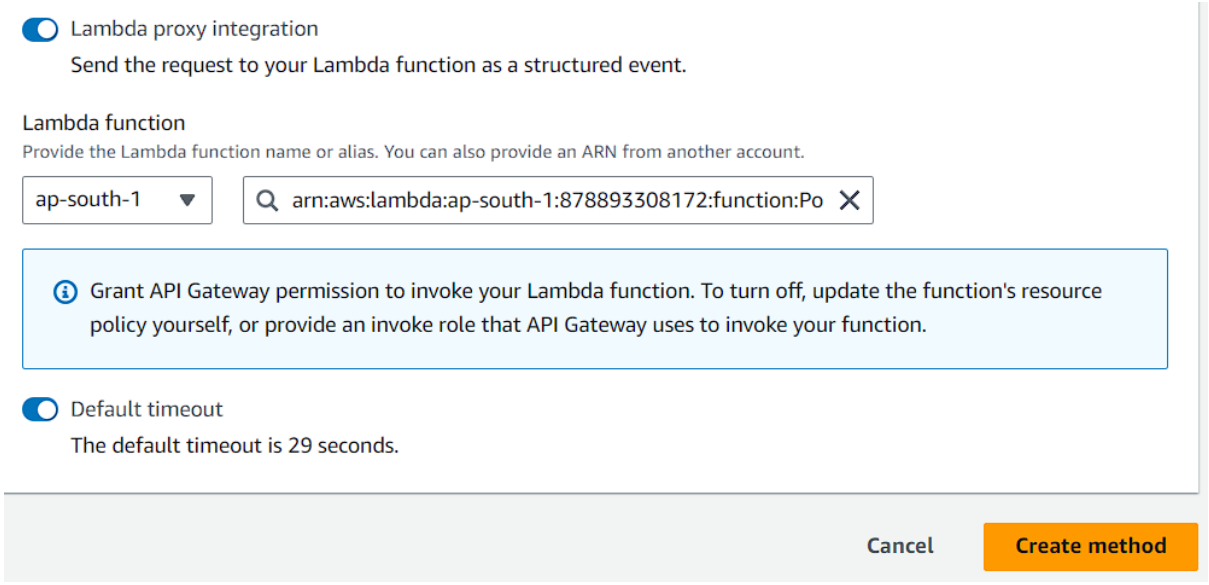


11. Now turn on Lambda proxy integration and choose you Get Lambda function.
12. Then just create your method.

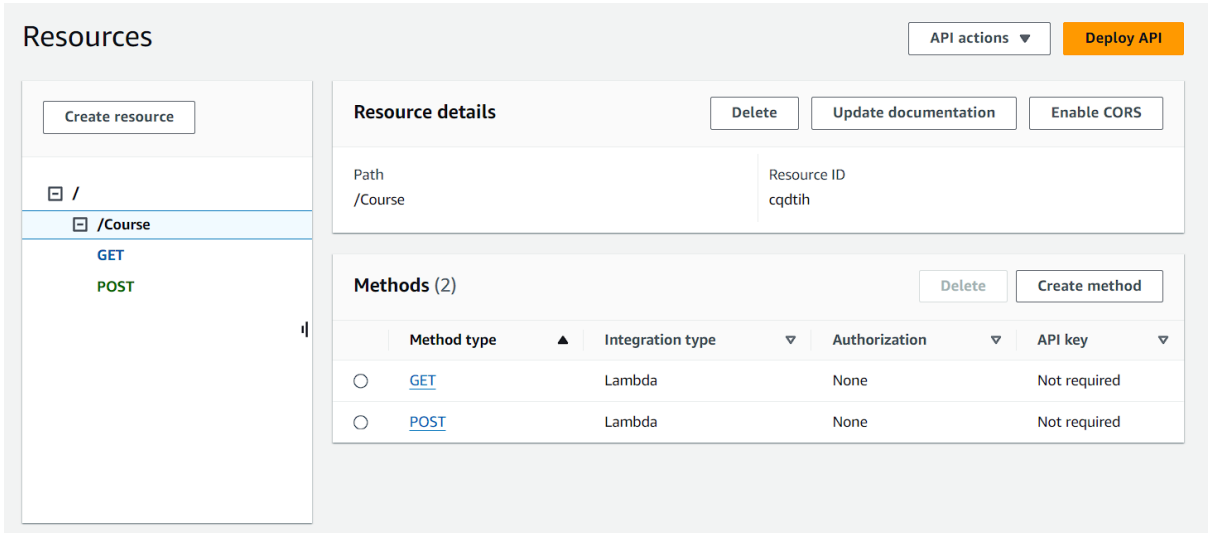


13. Now you need to create another method for Post.

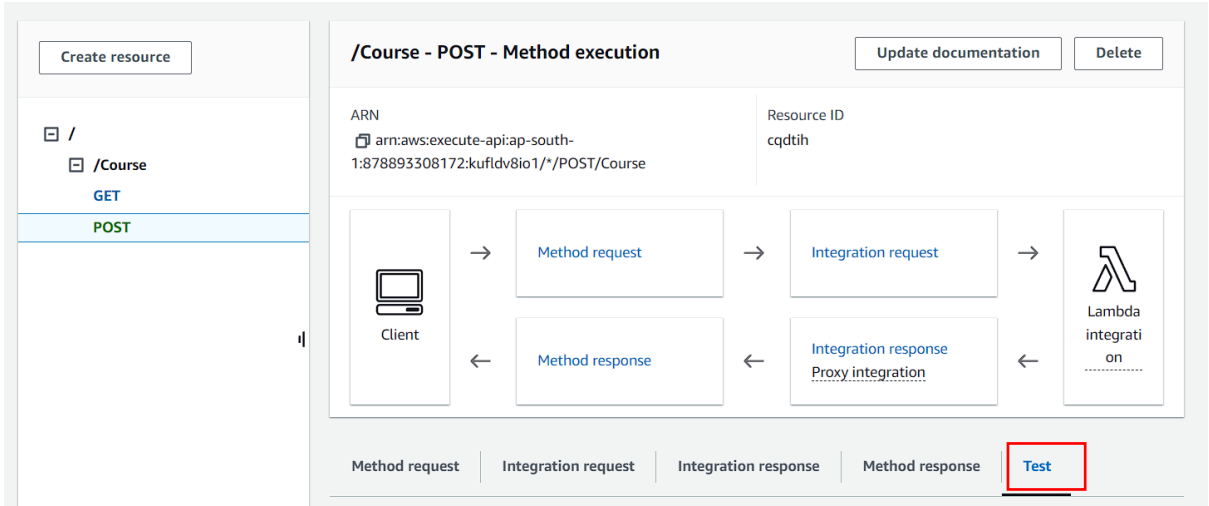




14. Here you can see that you have two methods in place.



15. Now you need to select your POST method and go to Test.



16. Now in the test request body you need to write this command.

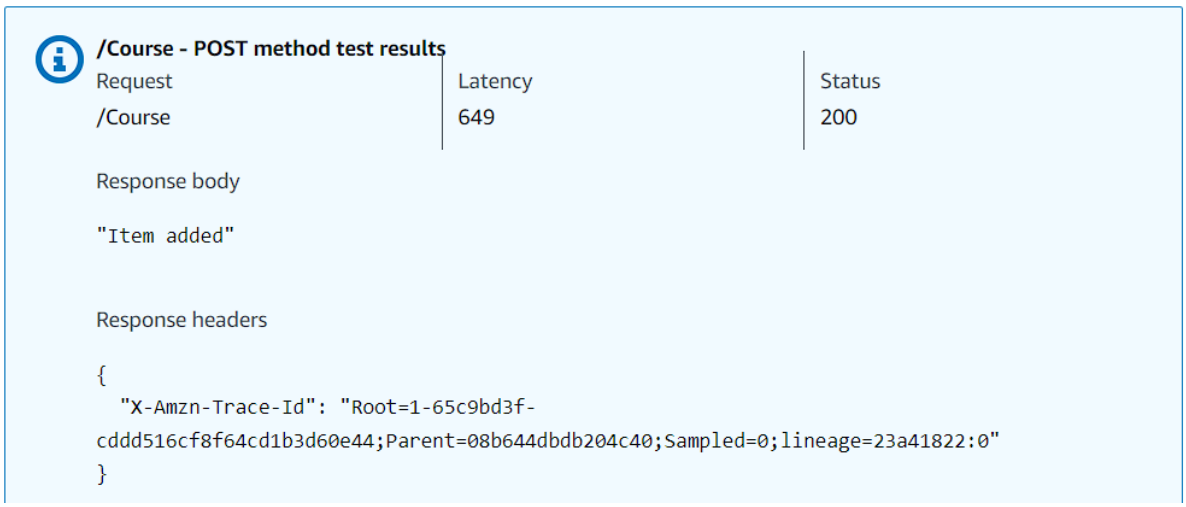
17. And then click on test.

```
{
  "courseName": "Computer Science Engineering",
  "ratingId": "1",
  "rating": "4.8"
}
```

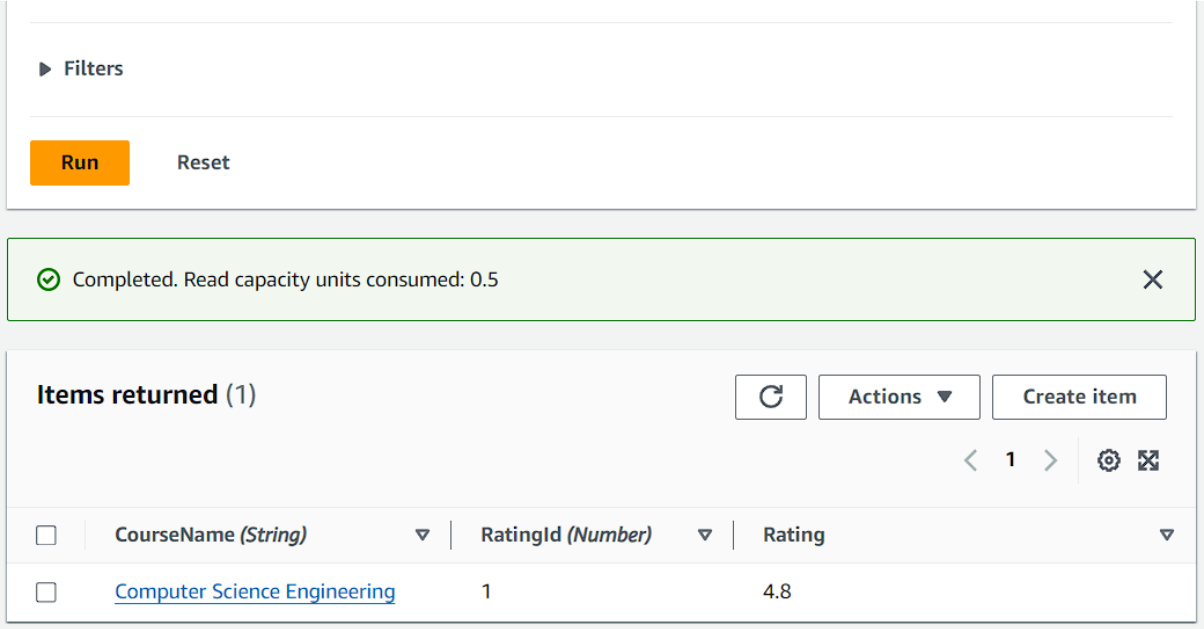
Request body

Test

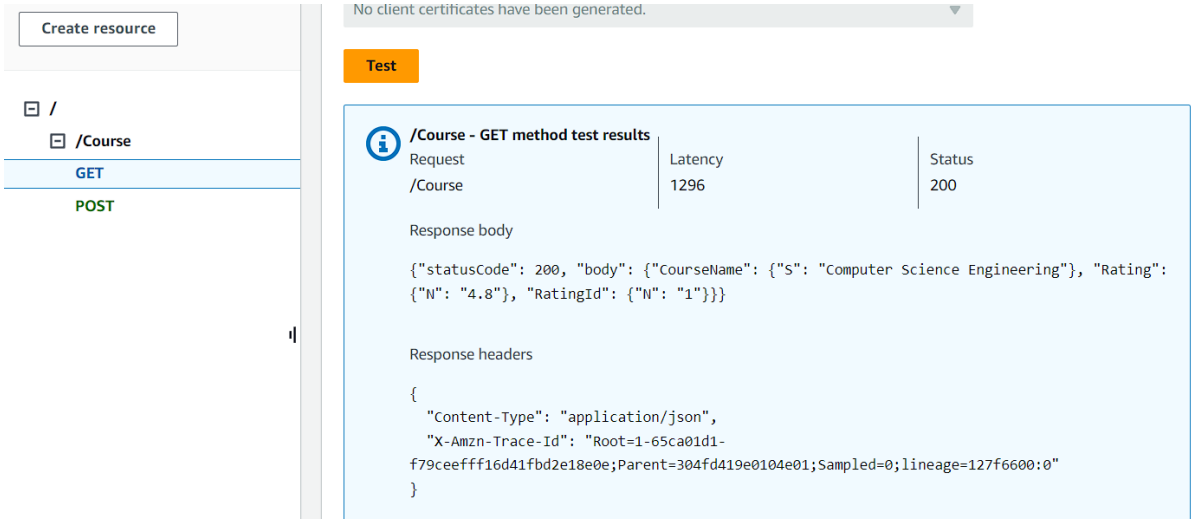
18. You will see that there is a response body which says Item added.
19. This means that the item is successfully added.



20. Now go to DynamoDB and click on your table then on explore table there you will see a record.

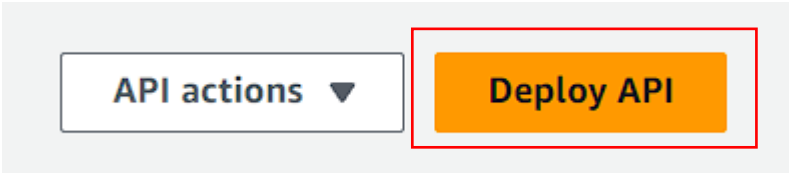


21. After that go to GET method and run the test there too.
22. You will see the response body in the JSON format.



Step 3: Deploy API Gateway

1. Now you are going to deploy your API.
2. So, now if you scroll up to the top on the API Gateway page. You'll see an option to deploy API. Click on it.



3. Then you need to choose a stage and give the stage name. Then click on Deploy.

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.

Stage

New stage

Stage name

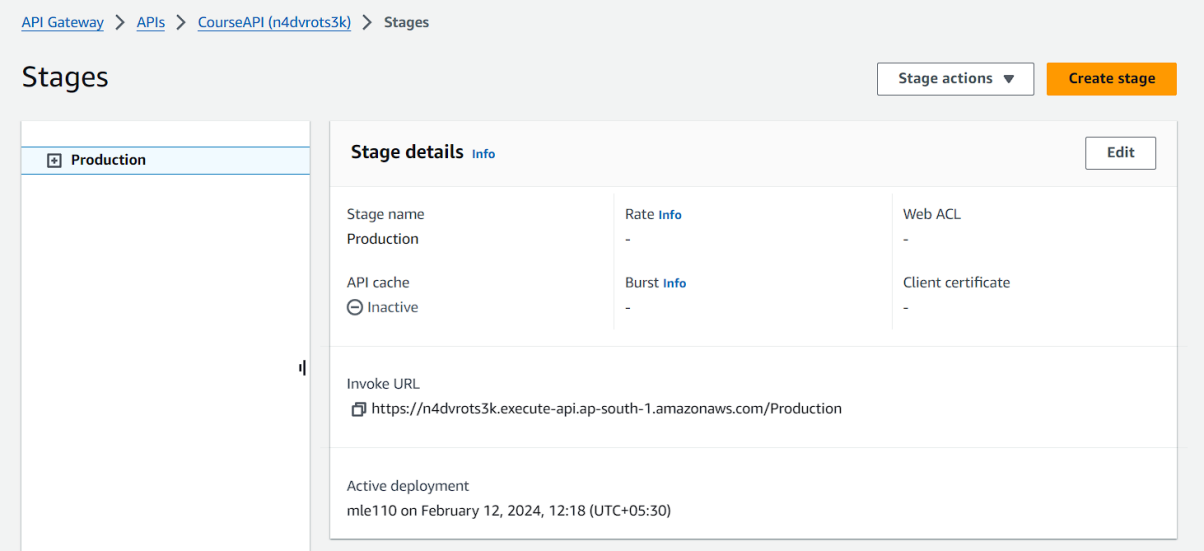
Production

A new stage will be created with the default settings. Edit your stage settings on the **Stage** page.

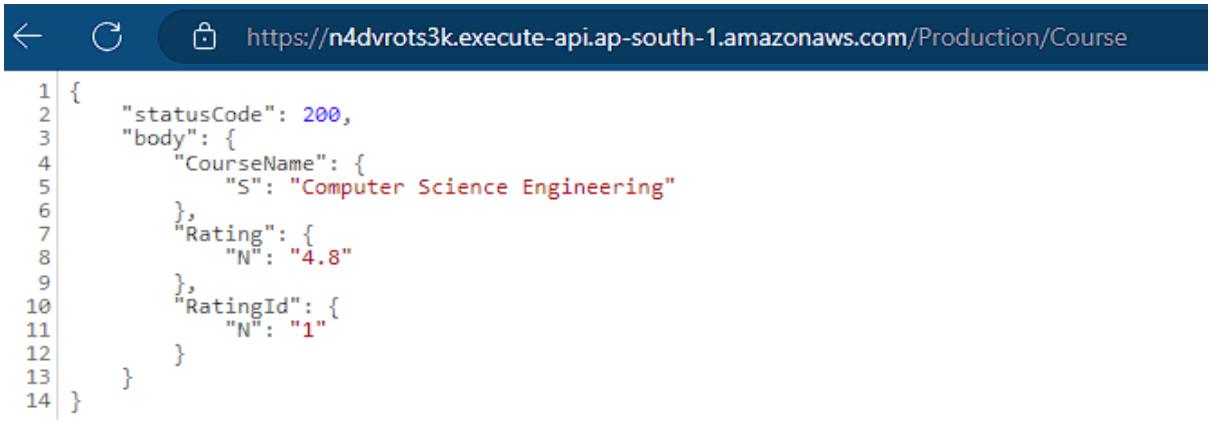
Deployment description

Cancel Deploy

4. Afterwards you will see that you are on the stages page. Now here you will get an URL. Copy this URL and paste it in a new tab.

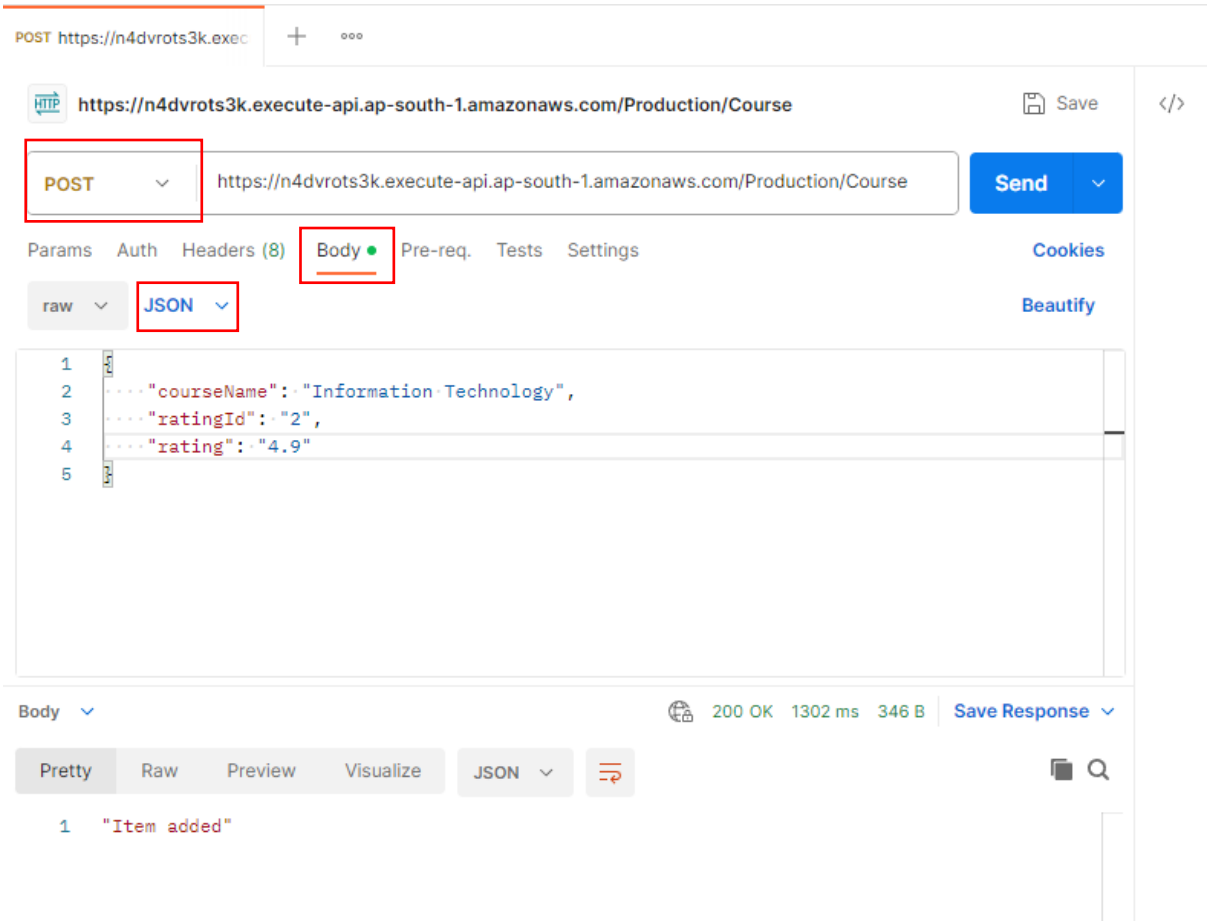


5. There you can see the data which is from the DynamoDB table.

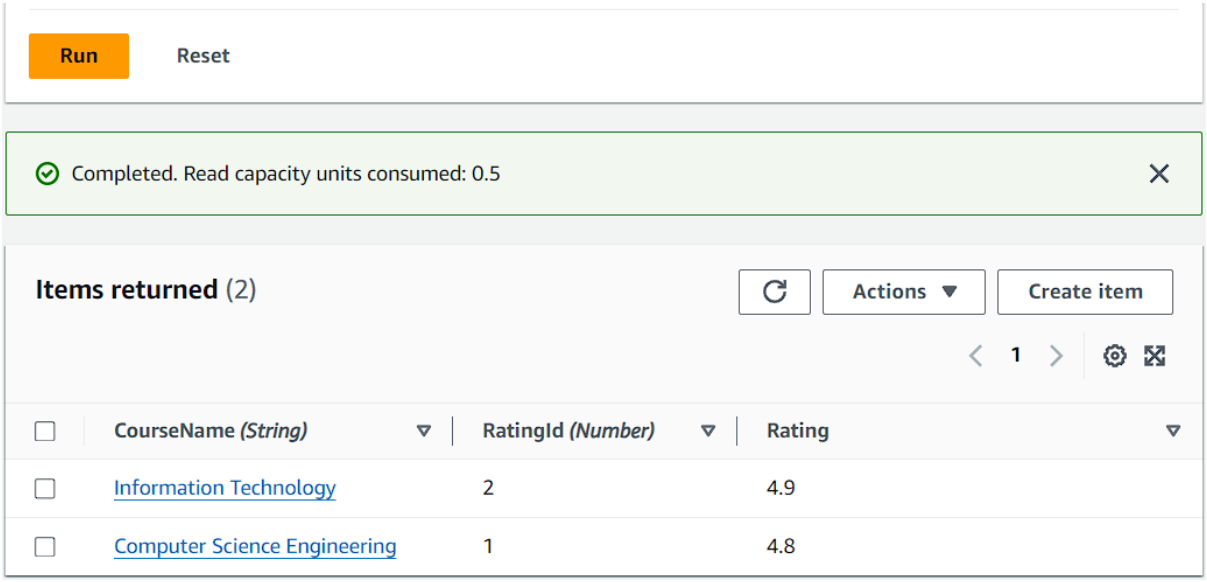


6. If you want to now invoke the post request, you have to use another tool.
7. So, we can't use the browser. And that's because we want to send in some data along with the post request. For this, we can make use of a tool known as the postman tool.
8. Now you need to download the postman tool and install it then after you need to copy the URL which API Gateway gave you and paste it in the Postman.
9. After that select POST then body and JSON as your format.
10. Then write this and click on send. You will get a message that the item has been added.

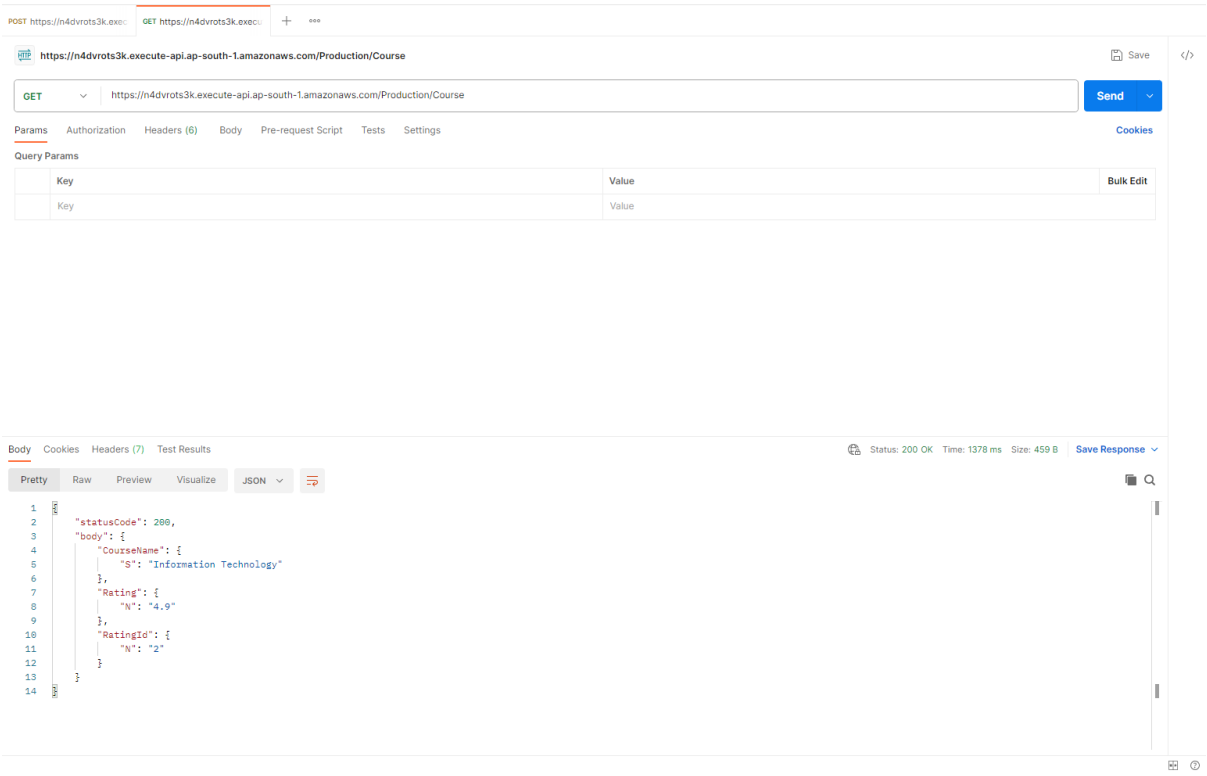
{
"courseName": "Information Technology",
"ratingId": "2",
"rating": "4.9"
}



11. After that if you will go to DynamoDB and Run it again. You can see a new record has been added to the list.



12. After that if you will go back to postman and create a new request for GET and paste the URL and click on Send. You will get all the information needed.




```
{
    "statusCode": 200,
    "body": {
        "CourseName": {
            "S": "Information Technology"
        },
        "Rating": {
            "N": "4.9"
        },
        "RatingId": {
            "N": "2"
        }
    }
}
```