



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.

Table details [Info](#)

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.

Course

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

CourseName

String

▼

1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

RatingId

Number

▼

1 to 255 characters and case sensitive.

[DynamoDB](#) > Tables

Tables (1) Info									C	Actions ▾	Delete	Create table
	Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode				
<input type="checkbox"/>	Course	Active	CourseName (S)	RatingId (N)	0	Off	Provisioned (5)	Provisioned (5)				

😎 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.

Permissions policies (3) [Info](#)

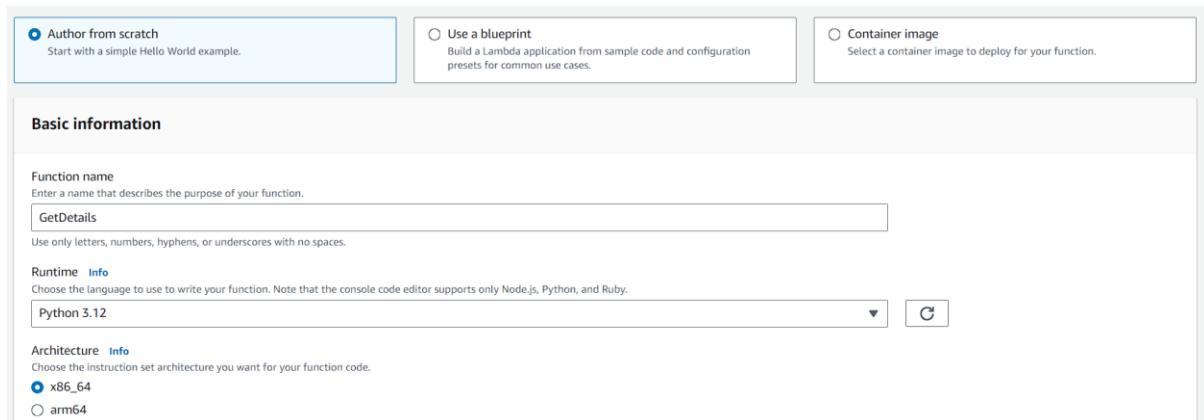
You can attach up to 10 managed policies.

Filter by Type

	Policy name	Type	Attached entities
<input type="checkbox"/>	AmazonDynamoDBFullAccess	AWS managed	3
<input type="checkbox"/>	AmazonS3ReadOnlyAccess	AWS managed	2
<input type="checkbox"/>	AWSLambdaBasicExecutionRole	AWS managed	2

😎 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.

Permissions [Info](#)

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.

▼ Change default execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Create a new role with basic Lambda permissions
 Use an existing role
 Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

demo-lambda-role [View](#) [Edit](#)

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
```

The screenshot shows a Lambda function named "lambda_function" in the AWS Lambda console. The code is written in Python and performs a scan operation on a DynamoDB table named "Course". The function returns a response object with a status code of 200 and the body of the first item from the scan result.

```

1 import boto3
2
3 # Create a DynamoDB client
4 client = boto3.client('dynamodb', region_name='ap-south-1')
5
6 def lambda_handler(event, context):
7     # Define input parameters
8     input_params = {
9         "TableName": "Course"
10    }
11
12    # Execute the scan command
13    response = client.scan(**input_params)
14    print(response)
15    # Prepare the response
16    reply = {
17        "statusCode": 200,
18        "body": response['Items'][0]
19    }
20
21 return reply

```

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

The screenshot shows the "Create New Function" wizard in the AWS Lambda console. It includes sections for "Basic information", "Function code", "Runtime", "Architecture", and "Container image". The "Basic information" section shows the function name "PostDetails" and runtime "Python 3.12". The "Architecture" section shows "x86_64" selected. The "Container image" section is empty.

Basic information

Function name: PostDetails

Runtime: Python 3.12

Architecture: x86_64

Permissions Info

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.

▼ Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console [↗](#).

- Create a new role with basic Lambda permissions
- Use an existing role
- Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

▼C

[View the demo-lambda-role role ↗](#) on the IAM console.

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
```

The screenshot shows the AWS Lambda function editor. The tab bar at the top has 'lambda_function' selected, followed by 'Environment Vari' and 'Execution results'. A green '+' button is on the far right. The code area contains the following Python script:

```
1 import json
2 import boto3
3
4 # Create a DynamoDB client
5 client = boto3.client('dynamodb', region_name='ap-south-1')
6
7 def lambda_handler(event, context):
8     # Parse the event body
9     eventData = json.loads(event['body'])
10    courseName = eventData['courseName']
11    ratingId = eventData['ratingId']
12    rating = eventData['rating']
13
14    # Define parameters for PutItem command
15    params = {
16        "Item": {
17            "CourseName": {"S": courseName},
18            "RatingId": {"N": str(ratingId)},
19            "Rating": {"N": str(rating)}
20        },
21        "TableName": "Course"
22    }
23
24    # Execute the PutItem command
25    response = client.put_item(**params)
26
27    # Prepare the response
28    reply = {
29        "statusCode": 200,
30        "body": json.dumps("Item added")
31    }
32
33    return reply
34
```

😎 Step 3: Create API Gateway

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

The screenshot shows the AWS API Gateway service page. The top navigation bar has 'API Gateway' with a star icon and 'Build, Deploy and Manage APIs'. Below the navigation is a section titled 'Top features' with links for 'APIs', 'Custom domain names', 'VPC links', 'REST API', and 'HTTP API'.

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

Import

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.

API details

New API

Create a new REST API.

Clone existing API

Create a copy of an API in this AWS account.

Import API

Import an API from an OpenAPI definition.

Example API

Learn about API Gateway with an example API.

API name

CourseAPI

Description - optional

API endpoint type

Regional APIs are deployed in the current AWS Region. Edge-optimized APIs route requests to the nearest CloudFront Point of Presence.
Private APIs are only accessible from VPCs.

Regional

Cancel

Create API

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

The screenshot shows the AWS API Gateway interface. In the top navigation bar, it says "API Gateway > APIs > Resources - CourseAPI (kufldv8io1)". On the left, there's a sidebar with a "Create resource" button highlighted by a red box. The main area is titled "Resource details" with a path of "/". It includes buttons for "Update documentation" and "Enable CORS". Below this is a section for "Methods (0)" with a "Create method" button. A message states "No methods defined." At the top right, there are "API actions" and a "Deploy API" button.

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

The screenshot shows the "Create resource" dialog. It has a "Resource details" section where "Proxy resource" is selected. The "Resource path" is set to "/" and the "Resource name" is "Course". There's also a "CORS (Cross Origin Resource Sharing)" checkbox. At the bottom are "Cancel" and "Create resource" buttons.

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

Resources

API actions ▾ Deploy API

Create resource	Resource details														
/	Path /Course	Resource ID cqdtih	Delete Update documentation Enable CORS												
Methods (0) <table border="1"> <thead> <tr> <th>Method type</th> <th>Integration type</th> <th>Authorization</th> <th>API key</th> </tr> </thead> <tbody> <tr> <td colspan="4">No methods</td> </tr> <tr> <td colspan="4">No methods defined.</td> </tr> </tbody> </table>				Method type	Integration type	Authorization	API key	No methods				No methods defined.			
Method type	Integration type	Authorization	API key												
No methods															
No methods defined.															

9. Now you need to choose the method type as GET.

10. The integration type as lambda.

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.



AWS service

Integrate with an AWS Service.



VPC link

Integrate with a resource that isn't accessible over the public internet.



11. Now turn on Lambda proxy integration and choose you Get Lambda function.

12. Then just create your method.

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.

ap-south-1 ▾

arn:aws:lambda:ap-south-1:878893308172:function:Ge 

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.

Default timeout

The default timeout is 29 seconds.

Cancel

Create method

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

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 isn't accessible over the public internet.



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.

ap-south-1 ▾

arn:aws:lambda:ap-south-1:878893308172:function:Po 

 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.

Default timeout

The default timeout is 29 seconds.

Cancel

Create method

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

Resources

API actions ▾ Deploy API

Create resource

/
 └ /Course
 └ GET
 └ POST

Resource details

Delete

Update documentation

Enable CORS

Path
/Course

Resource ID
cqdth

Methods (2)

Delete

Create method

Method type	Integration type	Authorization	API key
GET	Lambda	None	Not required
POST	Lambda	None	Not required

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

Create resource

/
 └ /Course
 └ GET
 └ POST

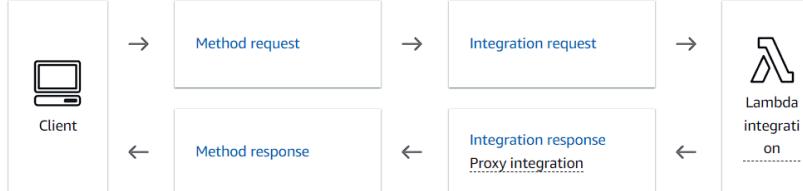
/Course - POST - Method execution

Update documentation

Delete

ARN
arn:aws:execute-api:ap-south-1:878893308172:kuflvd8io1/*/POST/Course

Resource ID
cqdth



Method request

Integration request

Integration response

Method response

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

```
1 ▼ {  
2   "courseName": "Computer Science Engineering",  
3   "ratingID": "1",  
4   "rating": "4.8"  
5 }
```

Test

18. You will see that there is a response body which says Item added.

19. This means that the item is successfully added.

 /Course - POST method test results		
Request	Latency	Status
/Course	649	200
Response body		
"Item added"		
Response headers		
{ "X-Amzn-Trace-Id": "Root=1-65c9bd3f- cddd516cf8f64cd1b3d60e44;Parent=08b644dbdb204c40;Sampled=0;lineage=23a41822:0" }		

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



The screenshot shows the AWS Lambda Test Results interface. At the top, there's a summary table for a POST method test. Below it, the response body is shown as a JSON object: "Item added". The response headers section shows a single header: "X-Amzn-Trace-Id".

Below the test results, there's a screenshot of the AWS DynamoDB Explore table interface. It shows a single item returned from a query. The item details are:

CourseName (String)	RatingId (Number)	Rating
Computer Science Engineering	1	4.8

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

The screenshot shows the AWS API Gateway Test interface. On the left, there's a sidebar with a 'Create resource' button and a tree view showing a root node with a 'GET' method selected. Below it is a 'POST' method. On the right, under the 'Test' tab, there's a message stating 'No client certificates have been generated.' A table titled '/Course - GET method test results' shows a single row: Request /Course, Latency 1296, and Status 200. Below the table, the 'Response body' is displayed as a JSON object:

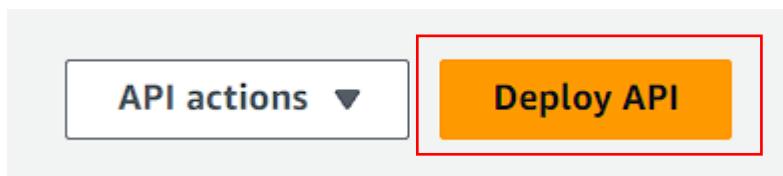
```
{"statusCode": 200, "body": {"CourseName": {"$": "Computer Science Engineering"}, "Rating": {"N": "4.8"}, "RatingId": {"N": "1"}}}
```

Under 'Response headers', there is a JSON object:

```
{ "Content-Type": "application/json", "X-Amzn-Trace-Id": "Root=1-65ca01d1-f79ceffff16d41fbd2e18e0e;Parent=304fd419e0104e01;Sampled=0;lineage=127f6600:0" }
```

😊 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

- 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.

API Gateway > APIs > CourseAPI (n4dvrots3k) > Stages

Stages

Stage actions ▾ Create stage

Stage details Info		
Stage name Production	Rate Info -	Web ACL -
API cache <input checked="" type="checkbox"/> Inactive	Burst Info -	Client certificate -
Invoke URL https://n4dvrots3k.execute-api.ap-south-1.amazonaws.com/Production		
Active deployment mle110 on February 12, 2024, 12:18 (UTC+05:30)		

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

A screenshot of a web browser window. The address bar shows the URL: <https://n4dvrots3k.execute-api.ap-south-1.amazonaws.com/Production/Course>. The main content area displays a JSON response with line numbers 1 through 14 on the left. The JSON data includes course details like name and rating.

```
1 {  
2     "statusCode": 200,  
3     "body": {  
4         "CourseName": {  
5             "S": "Computer Science Engineering"  
6         },  
7         "Rating": {  
8             "N": "4.8"  
9         },  
10        "RatingId": {  
11            "N": "1"  
12        }  
13    }  
14 }
```

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"  
}
```

The screenshot shows the Postman interface with a successful API call. The URL is `https://n4dvrots3k.execute-api.ap-south-1.amazonaws.com/Production/Course`. The method is set to `POST`, and the body is in `JSON` format. The response status is `200 OK` with `1302 ms` latency and `346 B` size. The response body is "Item added".

```
1
2     "courseName": "Information Technology",
3     "ratingId": "2",
4     "rating": "4.9"
5
```

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.

The screenshot shows the AWS DynamoDB Items page. A success message indicates "Completed. Read capacity units consumed: 0.5". The table displays two items:

	CourseName (String)	RatingId (Number)	Rating
<input type="checkbox"/>	Information Technology	2	4.9
<input type="checkbox"/>	Computer Science Engineering	1	4.8

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.

POST https://n4dvrots3k.execute-api.ap-south-1.amazonaws.com/Production/Course

GET https://n4dvrots3k.execute-api.ap-south-1.amazonaws.com/Production/Course

Send   Save 

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value
Key	Value

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 1378 ms Size: 459 B Save Response 

Pretty Raw Preview Visualize JSON 

```
1  "statusCode": 200,
2  "body": {
3    "CourseName": {
4      "S": "Information Technology"
5    },
6    "Rating": {
7      "N": "4.9"
8    },
9    "RatingId": {
10      "N": "2"
11    }
12  }
13}
14}
```