😊 AWS API Gateway & Lambda

AWS API Gateway

AWS API Gateway is a fully managed service from Amazon Web Services (AWS) that lets developers create, publish, maintain, monitor, and secure APIs (Application Programming Interfaces) easily. Here's a breakdown of what that means and how it works in simpler terms:

What is an API?

An API, or Application Programming Interface, is like a messenger that allows two applications to communicate with each other. Imagine you're ordering a meal at a restaurant: you (the customer) tell the waiter (the API) what you want from the menu, and the waiter brings your order from the kitchen (the application/server) back to you. In technical terms, the API is how your application or service can "ask" another application to perform certain actions or provide information.

What Does API Gateway Do?

In the context of AWS, API Gateway acts as a central point through which your applications can communicate with AWS services or even with each other.

1. **Request Management**: API Gateway handles incoming requests from users and forwards them to the appropriate backend resources, which can be AWS Lambda functions, EC2 instances, or even other web services.
2. **Security and Authorization**: API Gateway helps secure your API by enforcing who can access your API. You can set up authorization to ensure only authorized users (like registered customers) can access specific data or perform certain actions.
3. **Traffic Control and Scaling**: It helps handle heavy traffic by managing requests efficiently, making sure your application doesn't slow down under a load of users. API Gateway can handle thousands of requests per second.
4. **Monitoring and Analytics**: It provides detailed metrics and monitoring, allowing you to see how many people are using your API, track response times, and monitor any issues.
5. **Custom Responses**: API Gateway can transform data so that it's presented in the right format for users or devices. For instance, it can take data in a technical format and output it in a simpler format.

Why Use AWS API Gateway?

Here's why you might want to use API Gateway for your applications:

- **Simplicity**: It simplifies the process of creating and deploying an API.
- **Cost-Effectiveness**: You pay only for the number of requests received and the amount of data processed, so it's affordable for low- and high-traffic apps.

- **Security**: It offers multiple security layers, including authorization, so your data and services are protected.
- **Scalability**: API Gateway automatically adjusts to handle more users without you needing to make adjustments manually.

Real-World Example

Imagine you have a mobile app for an e-commerce store, and your users want to view products, place orders, or check order statuses. You can use API Gateway to handle these requests:

1. When a user views products, API Gateway takes that request and sends it to a backend service that fetches product data from a database.
2. When a user places an order, the request goes through API Gateway, which ensures they're authenticated, then passes the request to a service that handles order processing.
3. When the user checks the status of an order, API Gateway fetches that data securely from your backend.

With API Gateway, all these requests are handled smoothly and securely, allowing you to focus on building your application without managing infrastructure.

In short, AWS API Gateway is like a super-organized, high-performing gatekeeper that ensures your applications communicate efficiently and securely with each other and with users.

AWS Lambda

AWS Lambda is a cloud service that lets you run code without needing to manage servers. It's part of the "serverless" computing model, where AWS handles all the infrastructure for you, and you simply focus on writing the code that does what you want. Here's a deep dive into what it is, how it works, and why it's valuable, all in layman's terms.

What is AWS Lambda?

AWS Lambda is like a piece of code you write that AWS can run for you whenever certain events occur, like when someone uploads a photo, clicks a button on your website, or even at a scheduled time of day. This code is called a *Lambda function*, and it's designed to handle specific tasks.

Example: Imagine you're running an online photo-sharing app. Every time someone uploads a photo, you might want to automatically resize it to create a thumbnail. A Lambda function can do exactly that without you needing to maintain a dedicated server for the job.

How Does AWS Lambda Work?

Here's how Lambda operates step-by-step:

1. **Write Your Code**: You write a piece of code in a programming language that Lambda supports, like Python, JavaScript (Node.js), Java, or several others. This code is the Lambda function, which performs a specific task (e.g., resizing an image, processing data, etc.).

2. **Define an Event**: Lambda functions are "event-driven," meaning they only run when triggered by an event. Events can be from various sources, like:
    - An HTTP request coming through API Gateway (such as a button click on your app)
    - A new file (like an image) uploaded to Amazon S3
    - A message added to an AWS SQS queue
    - A scheduled time, like a cron job that runs daily at midnight
3. **Function Execution**: When an event triggers the Lambda function, AWS quickly allocates the resources needed, runs your code, and then releases the resources once the code finishes executing. This happens within milliseconds.
4. **Automatic Scaling**: If thousands of events trigger the function at the same time, Lambda automatically scales to handle each request without you needing to intervene. It can process multiple requests concurrently, so if 100 users upload photos simultaneously, Lambda will create multiple instances of your function to handle all of them at once.
5. **Pay-as-You-Go**: With Lambda, you only pay for the time your code actually runs, measured in milliseconds. So, if your code runs for 200 milliseconds, you're billed for just those 200 milliseconds and nothing else. If no events occur, you don't pay anything.

Why Use AWS Lambda?

Lambda's serverless nature and automatic scaling provide multiple advantages:

- **Cost-Efficiency**: You pay only when your function is running, which can be much cheaper than keeping a server running all the time.
- **Simplicity**: You don't need to manage, configure, or worry about servers—AWS does that for you.
- **Scalability**: It scales automatically to handle any increase or decrease in the number of requests, making it ideal for unpredictable workloads.
- **Event-Driven Processing**: Lambda is excellent for tasks triggered by specific events, like data processing, handling HTTP requests, or even automating regular maintenance tasks.

Common Use Cases

Here are some real-world scenarios where AWS Lambda is a great fit:

- **Data Processing**: When a new file (such as a log file or image) is uploaded to an S3 bucket, a Lambda function can automatically process it, extracting information or transforming it.
- **Backend for Web or Mobile Apps**: Lambda functions can handle backend logic for apps. Combined with API Gateway, it can process user requests like submitting forms, processing payments, or updating profiles.

- **Real-time Notifications**: Lambda can send out notifications when certain conditions are met, like alerting an admin if there's unusual activity in an application.
- **IoT Device Management**: Lambda can handle data streaming from IoT devices, allowing for real-time processing and analysis.

Lambda Example Scenario

Let's say you run a news website where users can submit their own articles. When a user submits an article, you want to scan it for offensive words or check that it's within a certain word count before publishing.

1. **Trigger**: A user submits an article through your website, which gets saved in an S3 bucket.
2. **Lambda Function**: A Lambda function is triggered when the article is uploaded. The function scans the content for inappropriate words and counts the words.
3. **Result**: If the article passes the checks, Lambda can pass it to a publishing system; if not, it can alert the user or an editor.

Summary

AWS Lambda is a powerful, flexible tool for handling tasks without managing servers. It's well-suited for scenarios where you need on-demand computing that responds to events or sudden traffic spikes. With Lambda, you write code to perform specific tasks, AWS takes care of the rest, and you pay only for what you use. It makes development simpler, cheaper, and more scalable.

In this lab, you'll create a basic serverless application by setting up an API Gateway and a Lambda function on AWS, linking them to deliver random messages via a web URL. You start by creating a REST API using API Gateway, where you set up a basic "GET" request method that initially returns a test message. After successfully testing the initial setup, you deploy this API so it's accessible online.

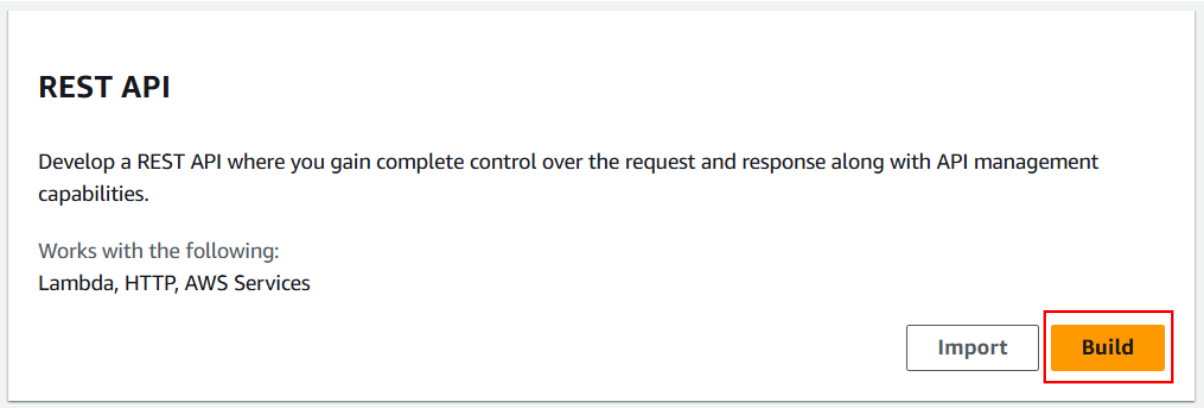
Next, you create a Lambda function that selects a random message each time it's triggered. After testing the Lambda function, you integrate it with the API Gateway, replacing the test response with your Lambda's random message. Finally, you deploy the updated API, allowing anyone to access it via a browser link, where each page refresh delivers a new message from the Lambda function.

End Goal: A fully functioning API that returns a random message every time it's accessed online.

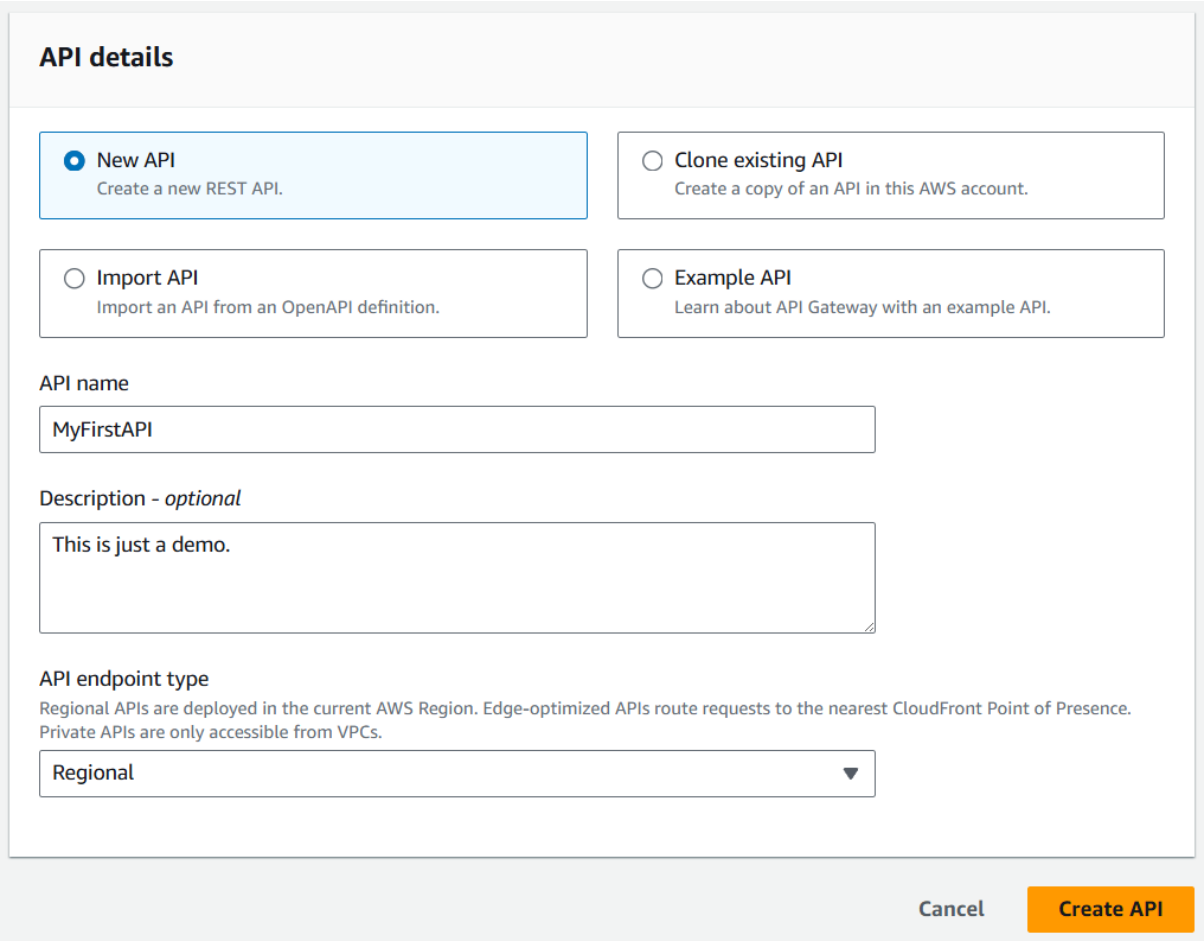
😄 To begin with the Lab:

1. In this lab, we will create an API Gateway and a lambda function, and then we will integrate both to create a serverless application.

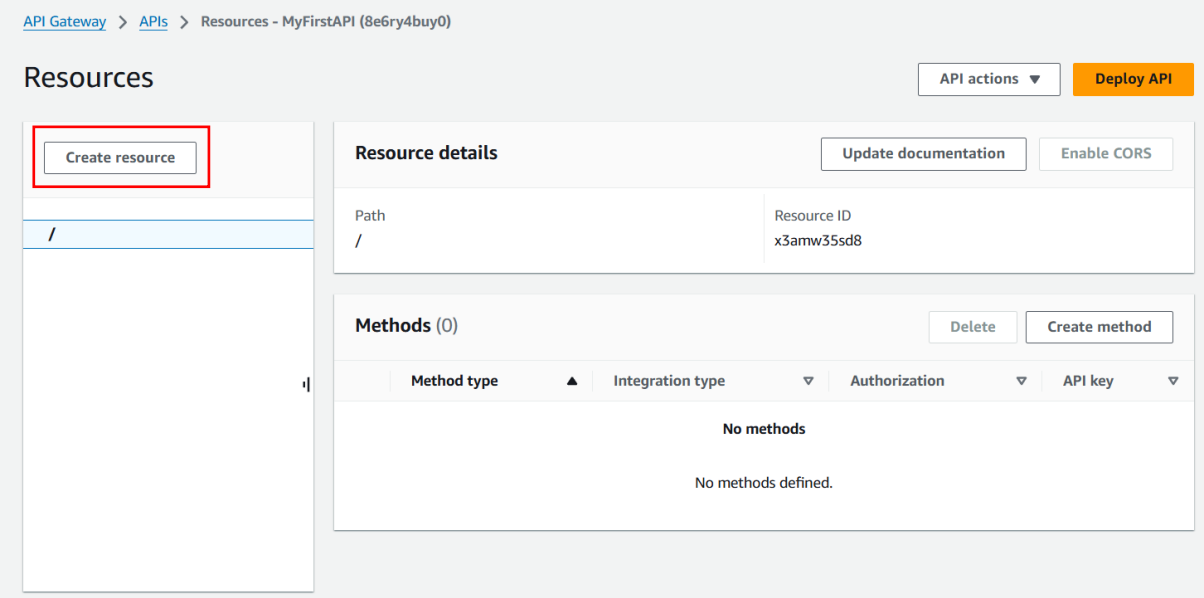
2. First, log in to your AWS Console, then navigate to API Gateway and click on Create. After that scroll down to REST API and click on Build.



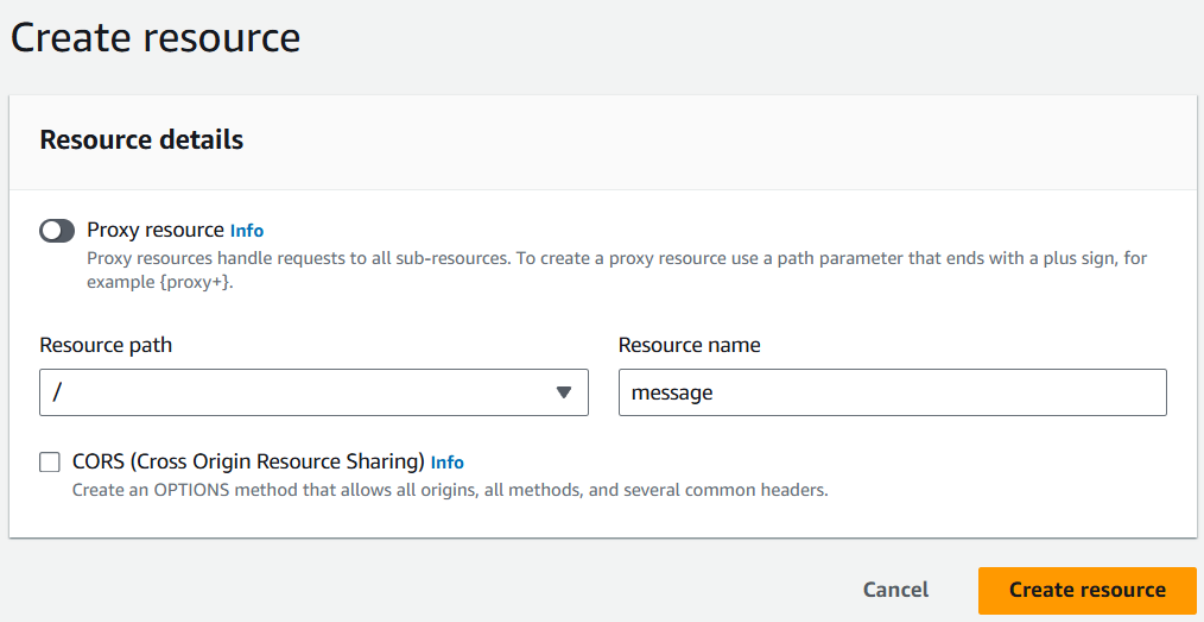
3. Then choose New API and give it a name and description after that in the Endpoint type choose Regional and click on Create API.



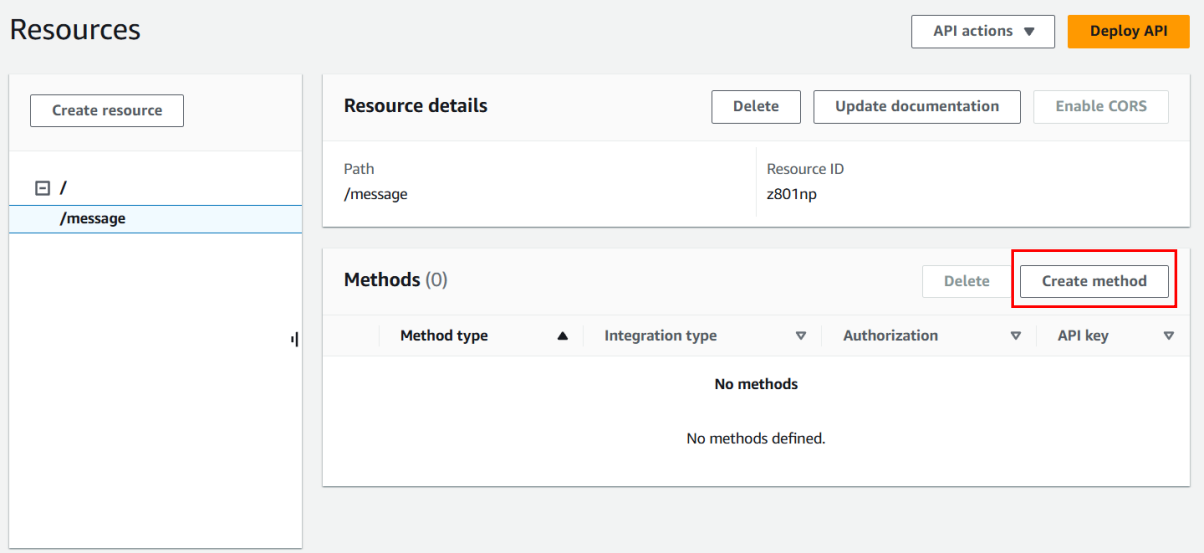
4. Once your API is created you need to click on Create resource.



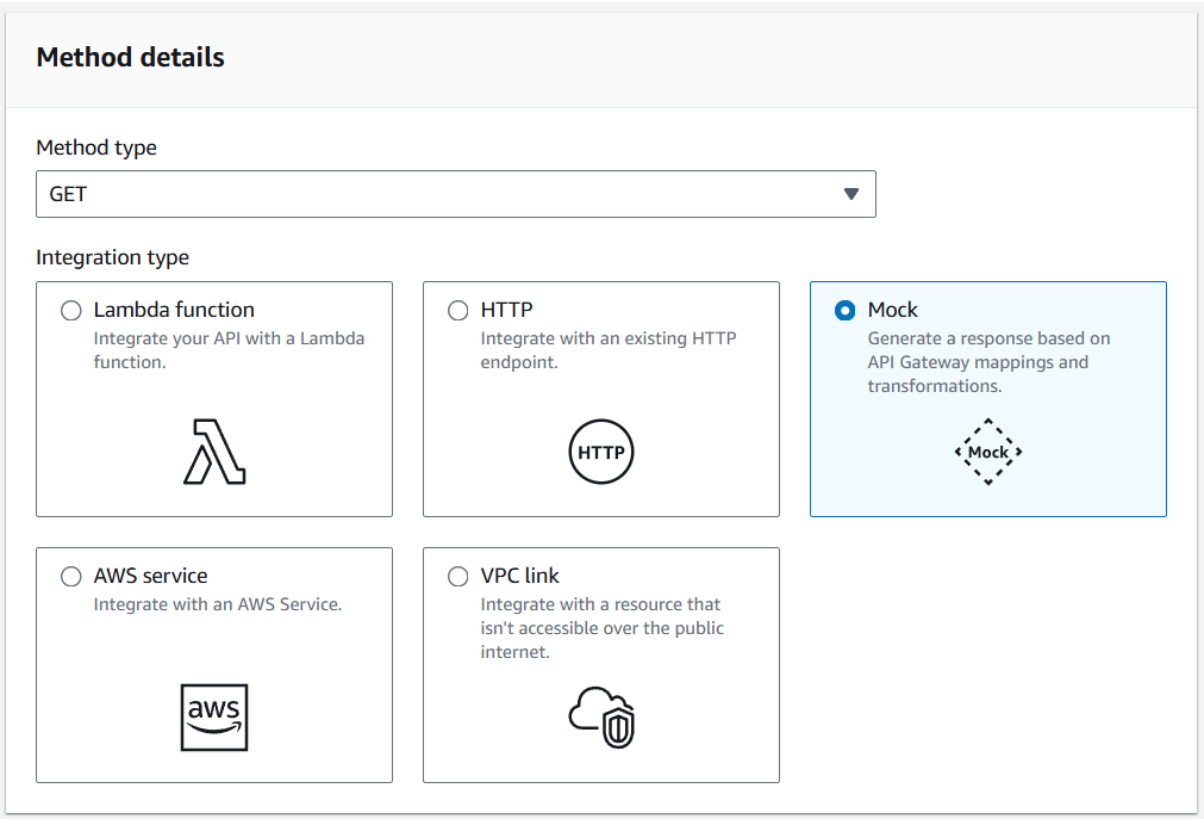
5. Here you need to give it a resource name and click on Create resource.



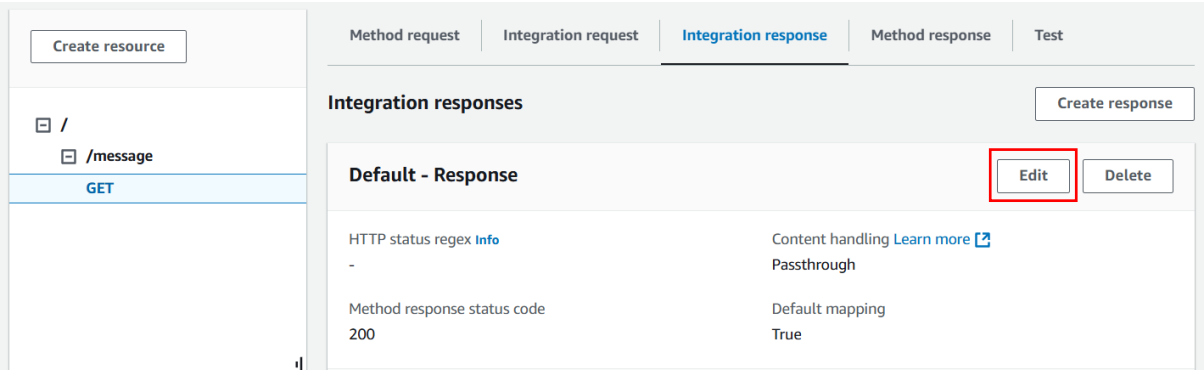
6. Below you can see that our resource has been created and from this resource, we will create a method. A method is an HTTP verb like GET, POST, PATCH, DELETE, and so on. Click on the Create method.



7. Here from the method details you need to choose GET as your Method type and then in the integration type choose Mock. So, using Mock as our integration type, we simply hard-code and return a dummy response from within the API Gateway.
8. Then scroll down and click on the Create method button.



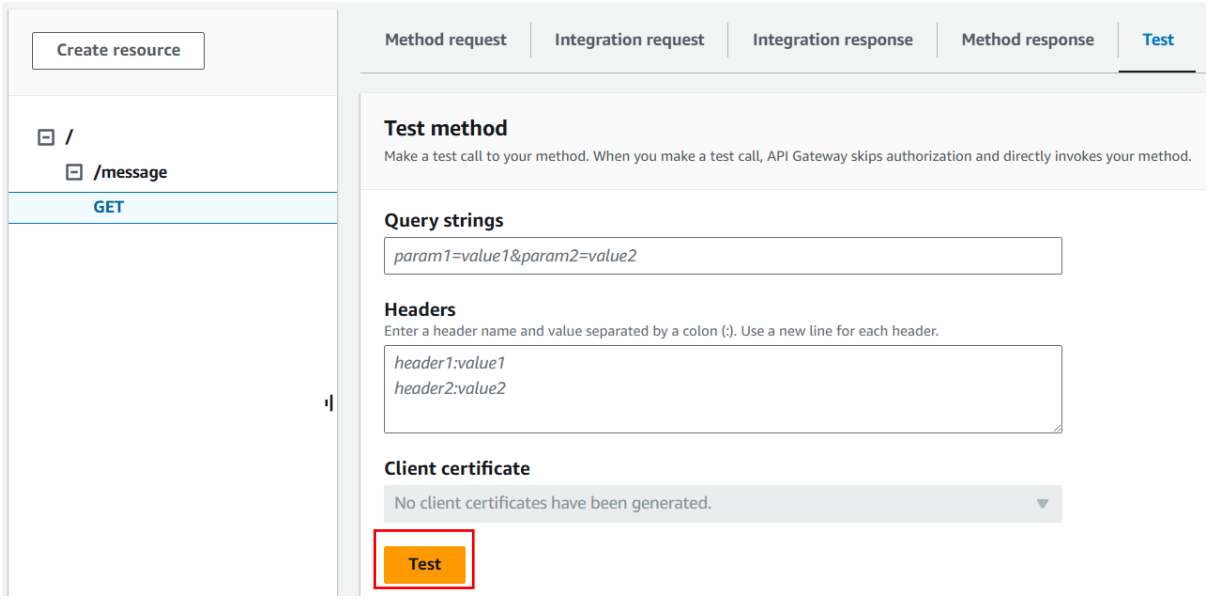
9. From our GET method come to Integration response and click on Edit.

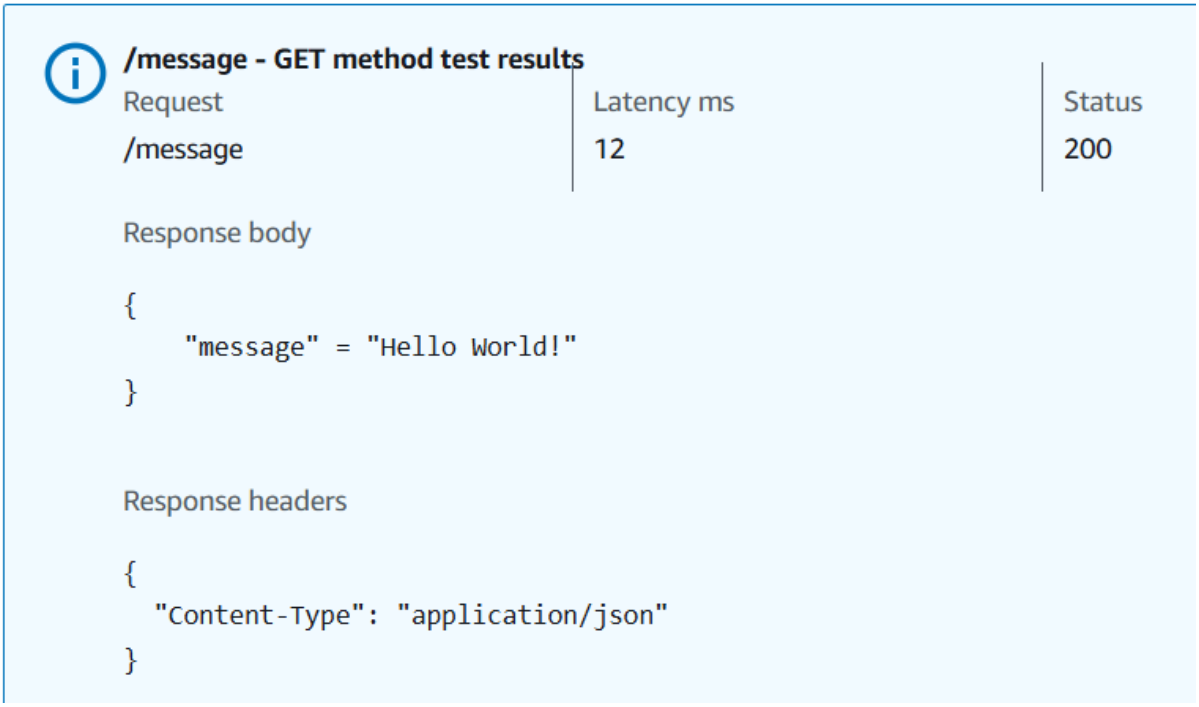


10. Then in the Edit integration response page you need to expand Mapping Templates. In the content type leave it to application/JSON and in the template body, you need to write this code. Click on Save.

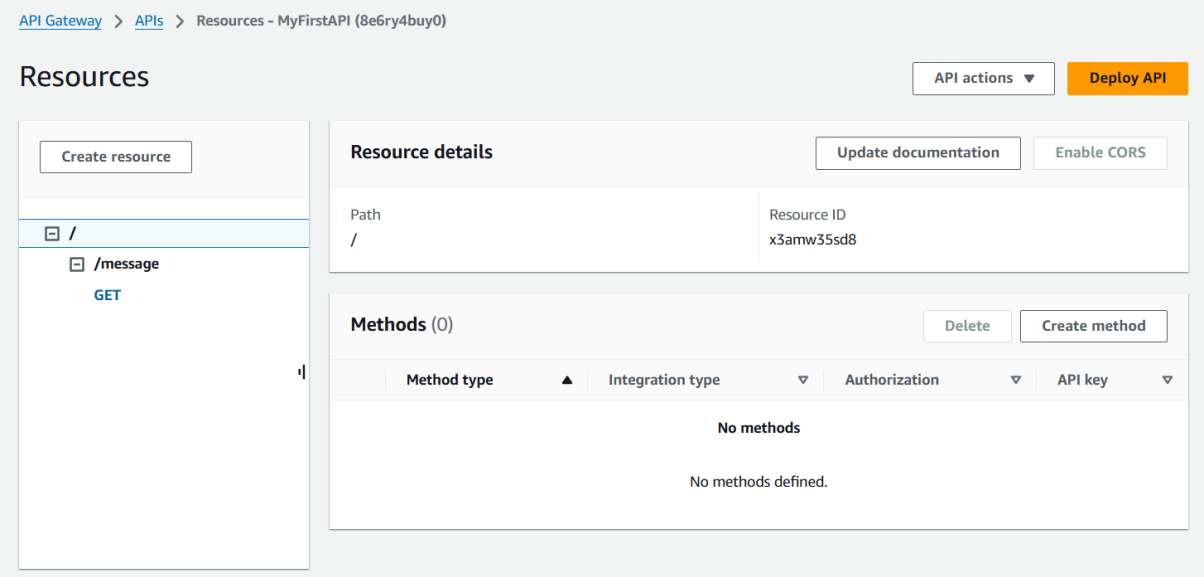
```
{
   "message" = "Hello World!"
}
```

11. Now from your GET method go to Test and click on Test. You will see that you have a response.





12. Once you get the response now you need to deploy your API. So, click on Deploy API then choose a new stage give it a name and finally click on Deploy.



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

Stage

New stage

Stage name

dev

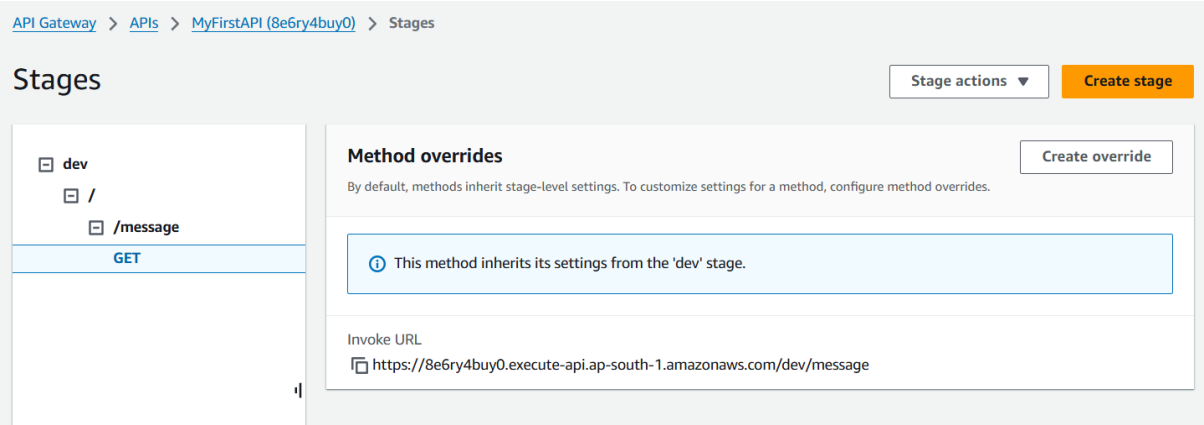
A new stage will be created with the default settings. Edit your stage settings on the **Stage** page.

Deployment description

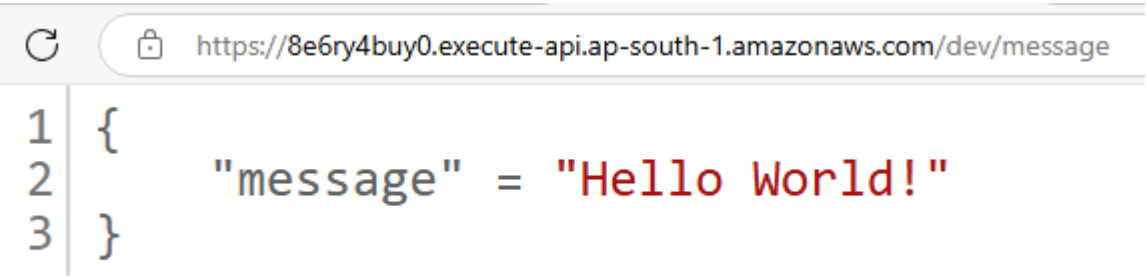
Cancel

Deploy

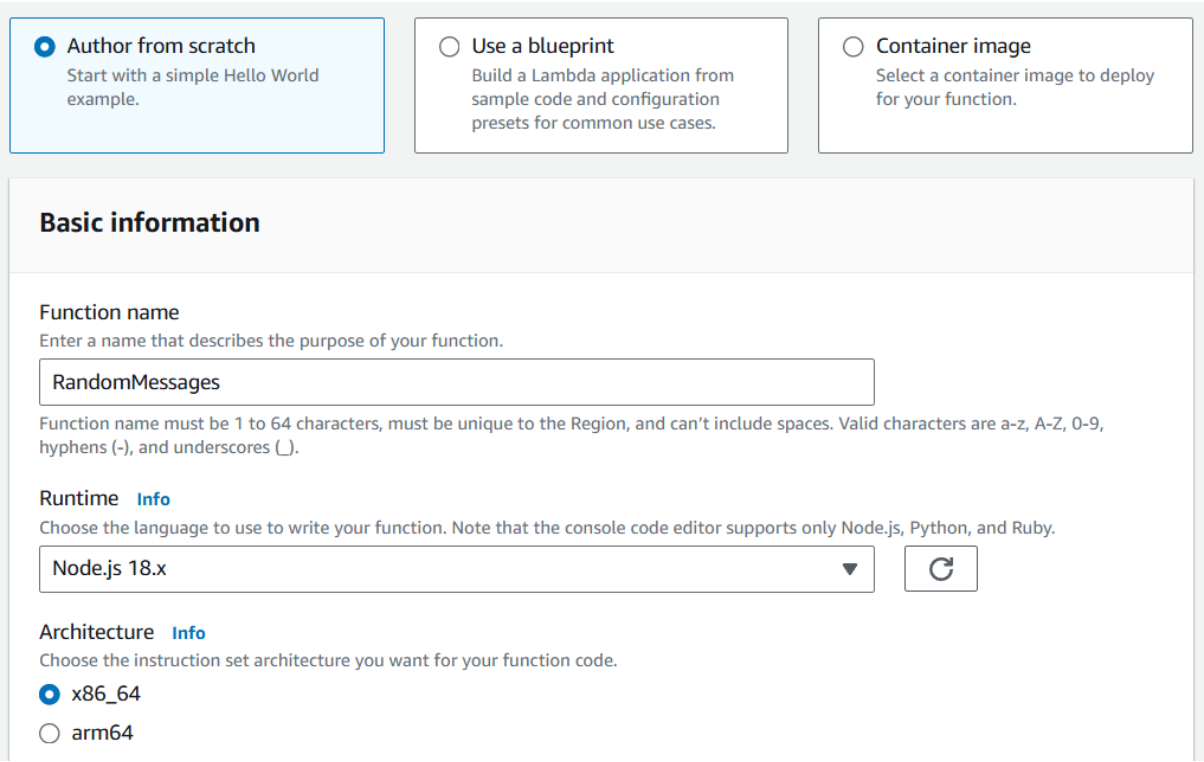
13. You will see that you are on the stages page, expand your dev stage until you come to the GET method, copy the Invoke URL, and paste it into a new tab.



14. Below you can see that you have successfully deployed your API.



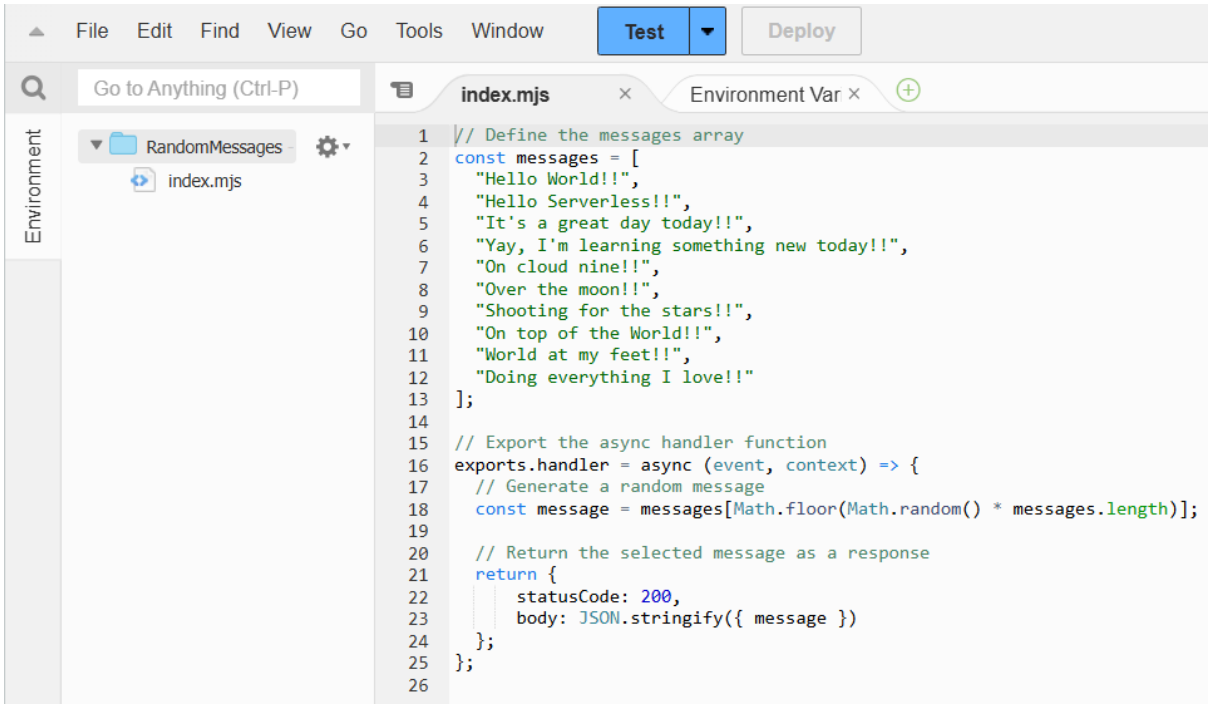
15. Now we are going to search for AWS Lambda and click on the Create function. Then on the create lambda function page, you need to give your function a name choose your runtime as Node.js, and click on create function.



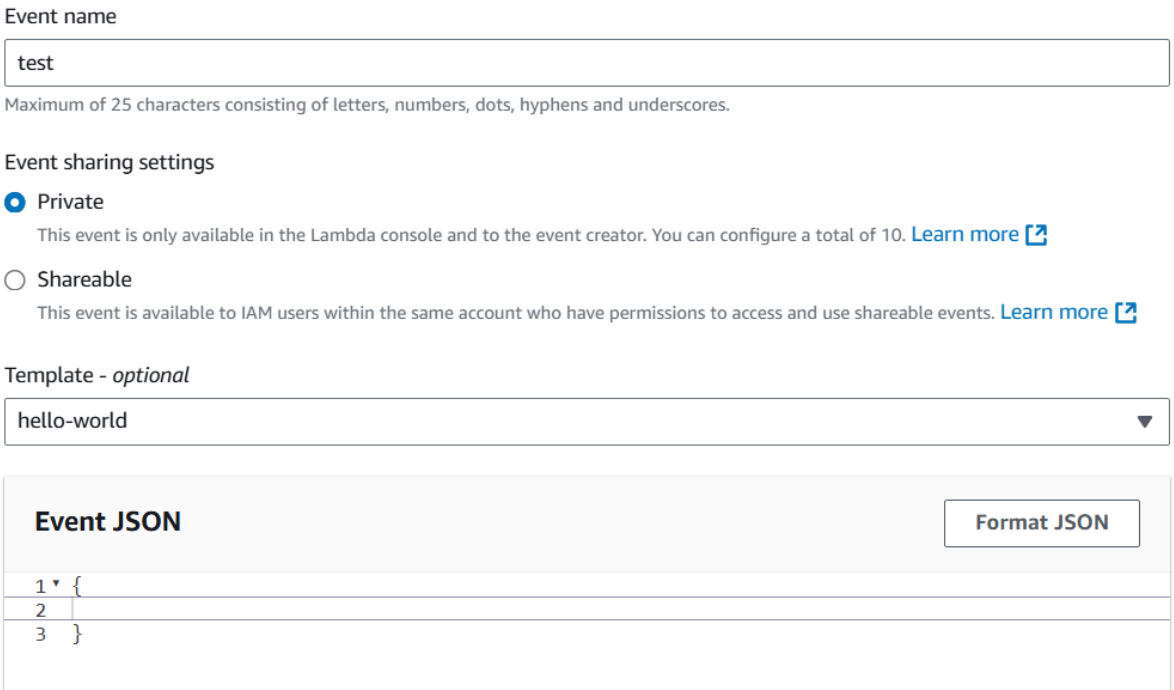
16. Then you need to copy the code given below and paste it in the code section of your lambda function and click on deploy.

```
// Define the messages array
const messages = [
  "Hello World!!",
  "Hello Serverless!!",
  "It's a great day today!!",
  "Yay, I'm learning something new today!!",
  "On cloud nine!!",
  "Over the moon!!",
  "Shooting for the stars!!",
  "On top of the World!!",
  "World at my feet!!",
  "Doing everything I love!!"
];

// Export the async handler function
exports.handler = async (event, context) => {
  // Generate a random message
  const message = messages[Math.floor(Math.random() * messages.length)];

  // Return the selected message as a response
  return {
    statusCode: 200,
    body: JSON.stringify({ message })
  };
```

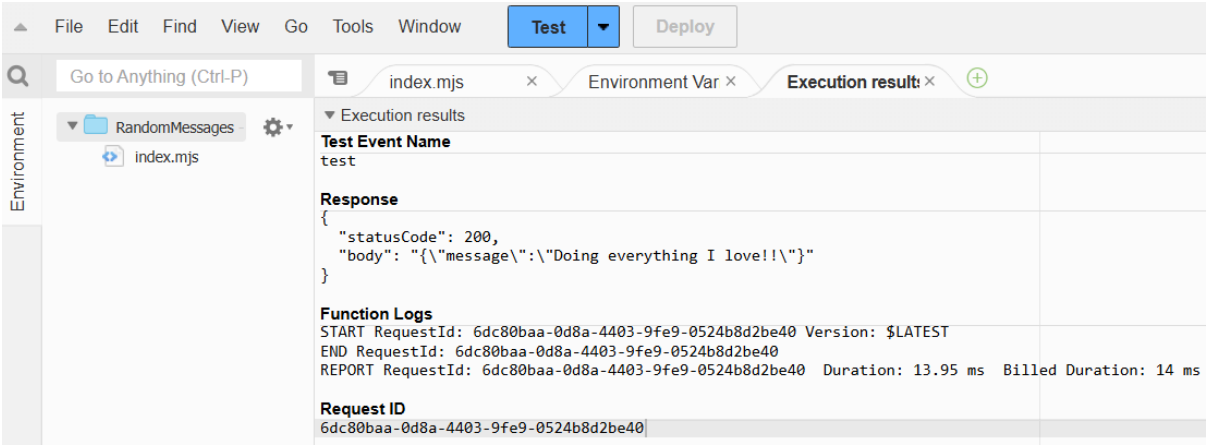
};



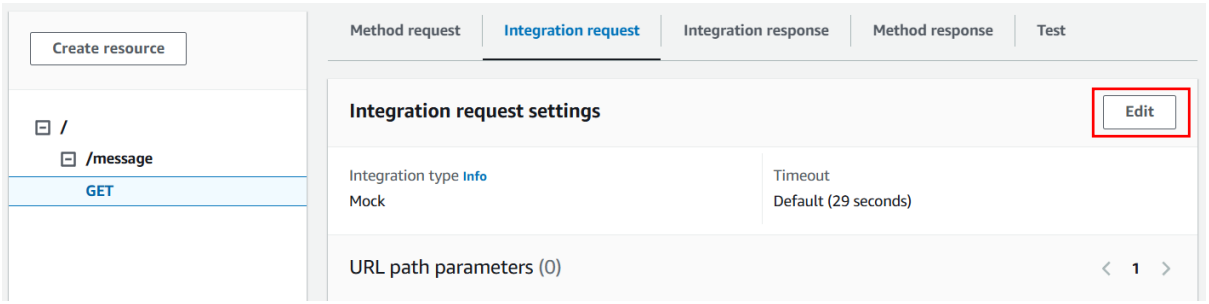
17. Once your code is deployed then we are going to test it. So, click on test to create a new test event. First, you need to create a test event and then give it a name, in the event JSON you need to keep an empty string as you can see down below. Click on Save.



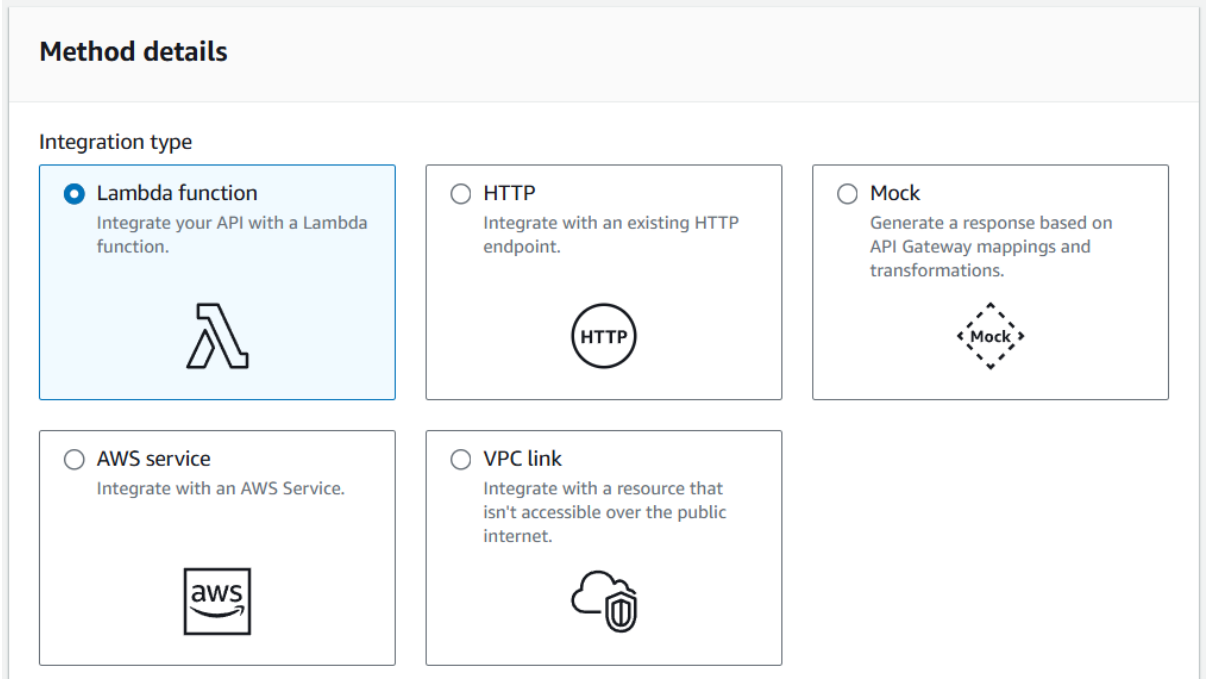
18. After that you need to click on Test and you will see that you have a response. Every time you click on test you will get a new response.

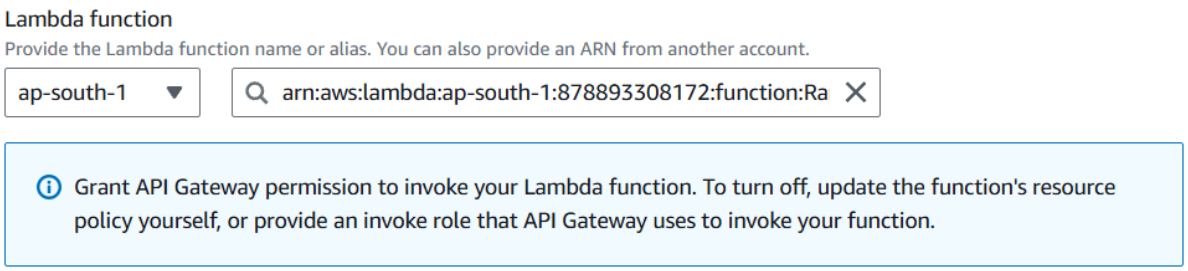


19. Now you need to come back to API Gateway because we are going to integrate our lambda function with API Gateway.
20. In your API gateway open your open your API and come to the GET method. Here you need to come to the Integration request and click on Edit.

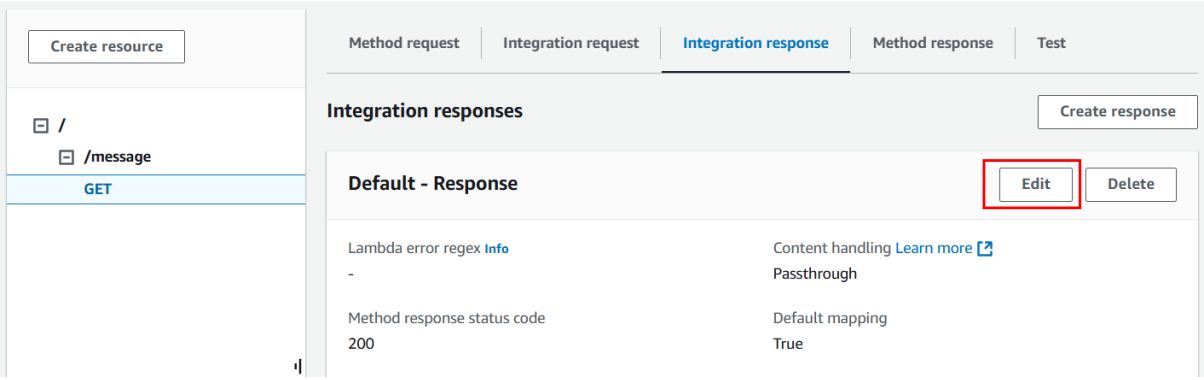


21. Firstly, you need to change the integration type to the Lambda function. Then scroll down and choose your lambda function which you just created. Click on save.





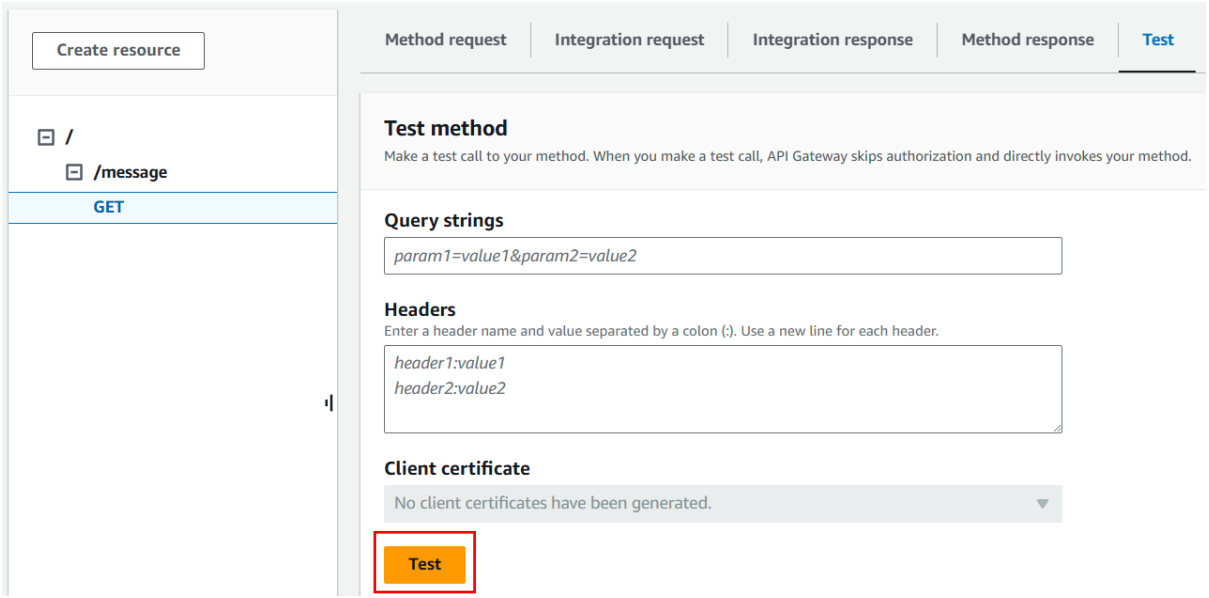
22. Then choose integration response and click on edit.



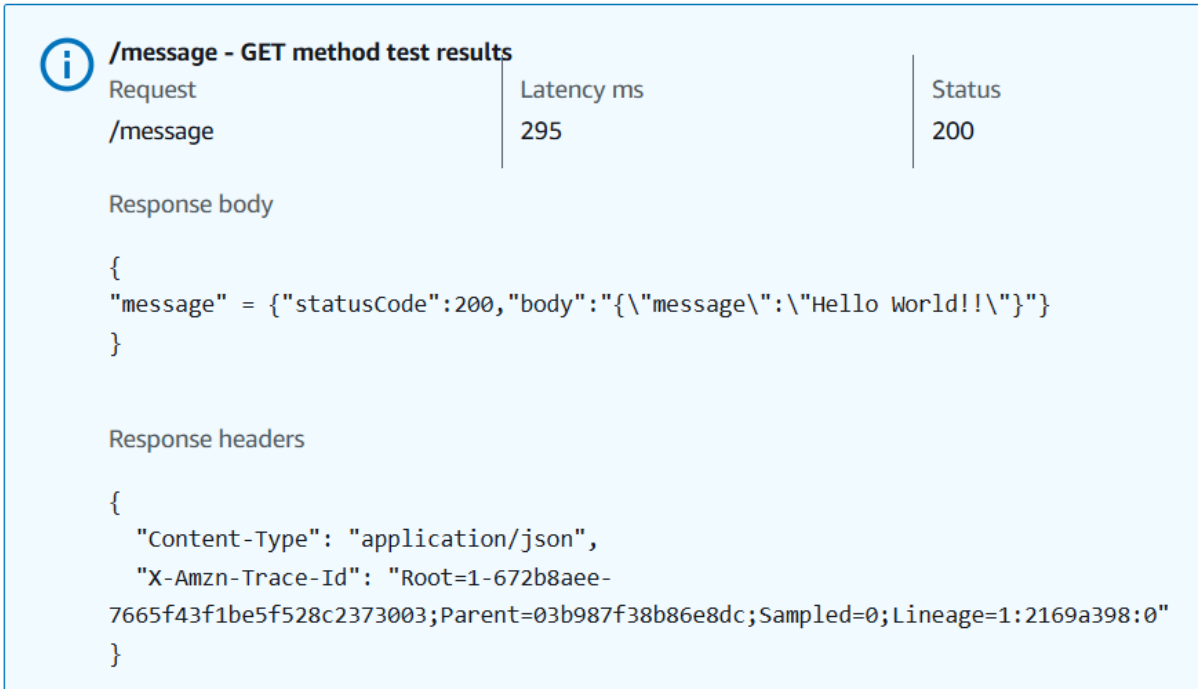
23. Scroll down to Mapping templates and in the template body you need to write the same code snippet. Click on Save.

{
"message" : $input.body
}

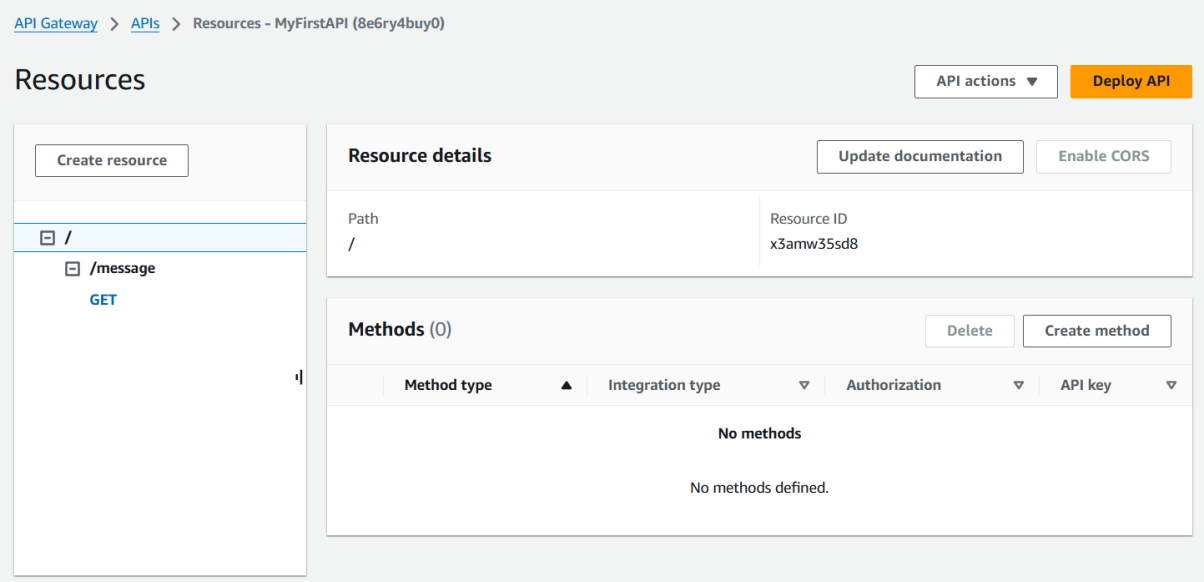
24. In the GET method you need to come to Test and click on Test.



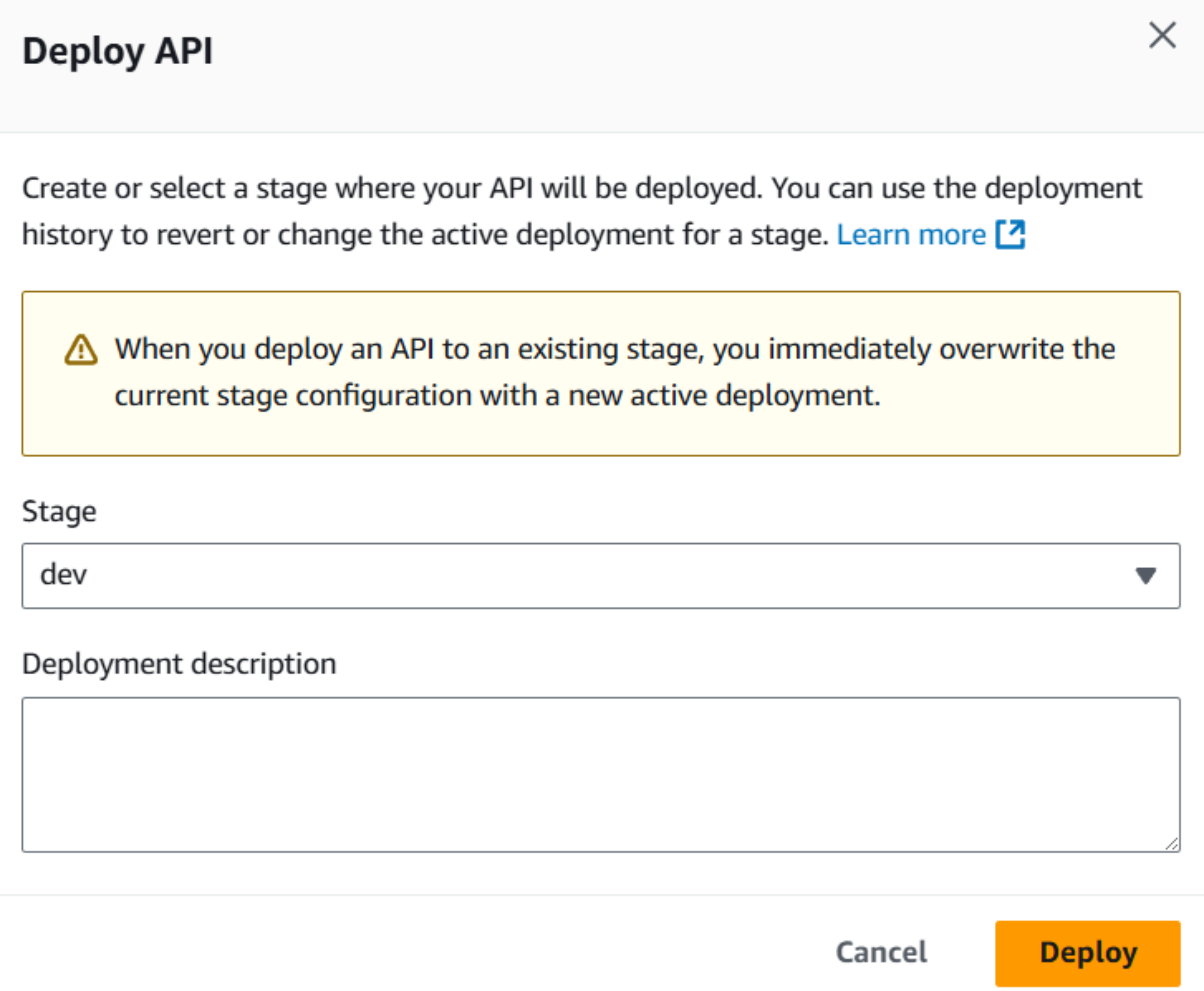
25. Like before you get the response but this time you get the response from the lambda function.



26. After that you need to deploy your API again to deploy the changes you made to your API.



27. Then choose your previous stage and click on Deploy.



28. Now you need to go back to the browser where you opened your API Gateway URL or you can copy it again from the stages page in your API gateway.
29. Below you can see that the changes have been deployed successfully. Every time you refresh your page you will see that a new message is coming up.

30. Once you are done with the lab delete all the resources you created.