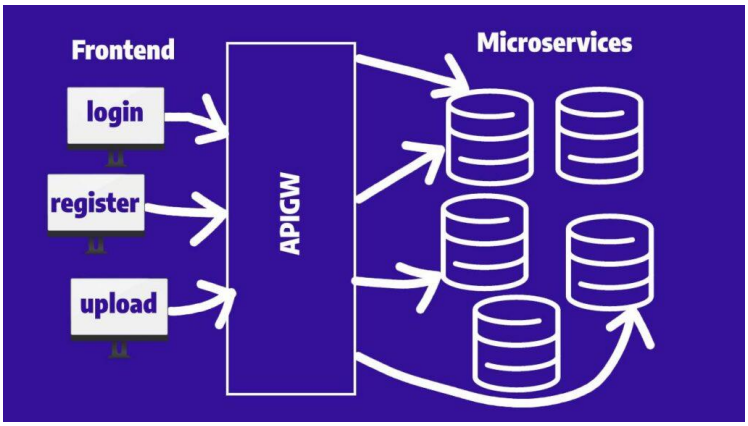
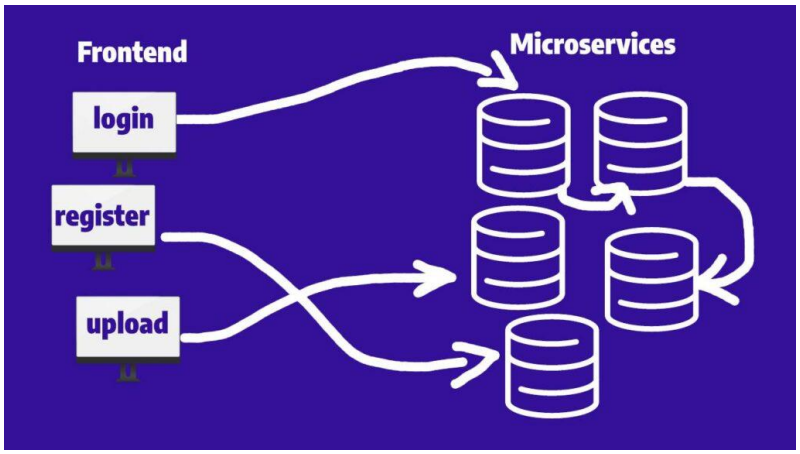
TABLE OF CONTENTS

WHAT IS API GATEWAY?

An API Gateway is an interface that sits in between the application and the microservices. Developers use them to create, publish, maintain, monitor and secure APIs. Its primary role is to act as a single-entry point and standardized process for interactions between an organization's apps, data and services and internal and external customers.



Without using an API Gateway you'd have to connect all your API resources directly with your user-facing applications which would make it more difficult to manage responses, implement updates to your business logic or even secure your API.



The API gateway can also perform various other functions to support and manage API usage, from authentication to rate limiting to analytics.

WHY DO WE NEED AN API GATEWAY?

It provides a unified entry point across internal APIs. It allows you to control user access. An API gateway is especially important for securing microservices.

- Firstly, it acts as a security layer, authenticating and authorizing incoming requests, ensuring that only authorized users or systems can access specific resources.
- Secondly, it simplifies the client experience by providing a unified API that abstracts the complexities of interacting with multiple services, reducing the need for clients to make numerous requests.
- Additionally, an API Gateway can handle tasks like load balancing, caching, and request transformation, optimizing the performance of the overall system.
- Lastly, it serves as a monitoring and analytics point, providing valuable insights into API usage and performance, aiding in troubleshooting and capacity planning. In essence, an API Gateway enhances the security, scalability, and manageability of modern distributed systems, making it an indispensable tool in building robust and efficient software architectures.

WHERE DO WE USE AN API GATEWAY?

API Gateways find applications in various scenarios, such as:

- **Microservices Architecture:** They act as a central entry point, routing requests to relevant microservices and providing load balancing.
- **Security and Authentication:** Enforcing security measures, generating access tokens, and protecting against threats.
- **Traffic Management:** Controlling the flow of traffic, implementing rate limiting, throttling, and caching.
- **Request and Response Transformation:** Modifying data formats and aggregating data from multiple sources.
- **Logging and Monitoring:** Providing centralized tracking of API usage and performance metrics.
- **Load Balancing:** Distributing requests across backend servers for scalability.

WHEN DO WE USE AN API GATEWAY?

API Gateways are used when building API-driven applications and services to ensure proper management, security, and optimization of APIs. They are typically employed during the design and development phase of software projects to provide a centralized point of control for all API-related functions, from security to traffic management.

INTRODUCTION

API stands for Application Program Interface. An **API Gateway** is a management tool that acts as an interface between users and microservices. The **Amazon API Gateway** is an **AWS service** that allows users to create, publish, secure, maintain and monitor APIs at any scale. You can create APIs in your own client application and can also make them available to third-party developers. It is serverless. It supports **HTTP, WebSocket, and REST APIs**. API Gateway is a type of server that provides the service of API which is used to customize each client. Amazon API Gateway will take care of all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management. Deployable with a couple of clicks, your Amazon API Gateway will act as the single point of entry to your backend services and will handle the data management, business logic or any other type of functionality or workloads running on services like EC2, AWS Lambda and many more.

HISTORY OF AMAZON API GATEWAY

Amazon API Gateway was launched in July 2015 as a managed service by Amazon Web Services (AWS). It was introduced to simplify the process of creating, deploying, and managing APIs, enabling developers to build secure and scalable interfaces for their applications. Here's a chronological history of Amazon API Gateway's key developments and milestones:

- **2015:** Amazon API Gateway was officially launched. It provided developers with a platform to create, publish, and manage APIs. The service aimed to address the challenges of building and maintaining APIs, offering features like traffic management, security, and integration with backend services.

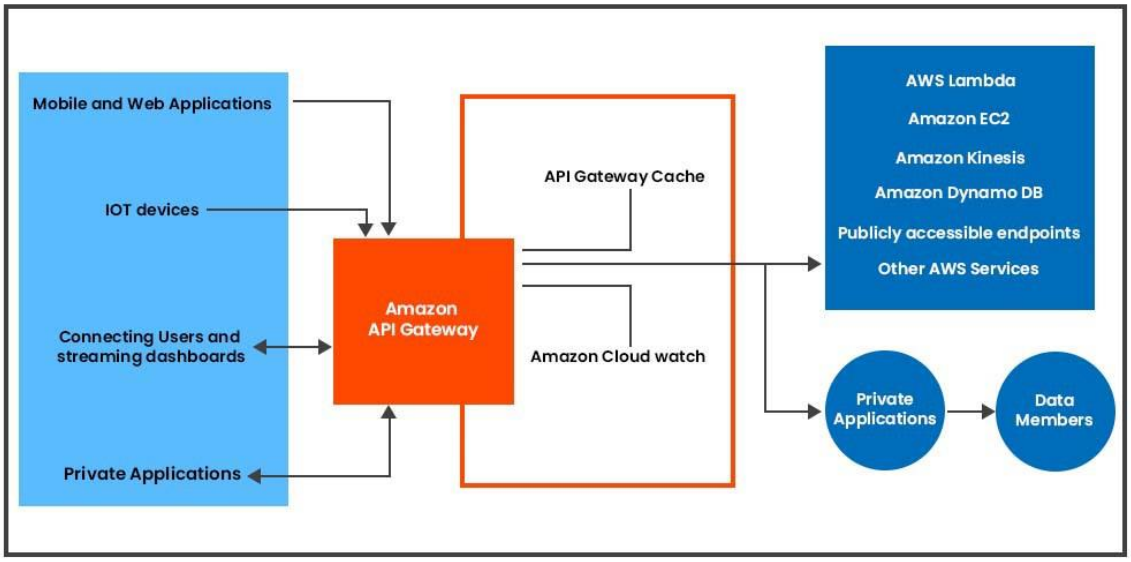
- **2016:** WebSocket API support was introduced. This feature allowed developers to create APIs that supported real-time, bidirectional communication between clients and servers over a single connection. AWS launched the "REST APIs with Private Integrations" feature, allowing users to integrate their API Gateway with resources within a Virtual Private Cloud (VPC), enabling private access to backend services.
- **2017:** Amazon API Gateway introduced support for regional endpoints. This allowed developers to deploy APIs in specific AWS regions to reduce latency for users located within the same region.
- **2018:** AWS introduced "HTTP APIs" as a new, simpler way to create HTTP-based APIs. These APIs aimed to provide optimized performance for HTTP services with reduced overhead compared to traditional REST APIs.
- **2019:** WebSocket APIs received route management enhancements, making it easier to manage different types of real-time data and interactions. Amazon API Gateway launched custom domain name support, allowing users to associate their APIs with branded domain names for a more professional appearance.
- **2020:** The "REST APIs with Private Integrations" feature was expanded to support more AWS services, providing more options for integrating with backend resources privately.
- **2021:** WebSocket APIs gained support for "Payload Compression," reducing the size of data transferred over WebSocket connections. API Gateway announced the availability of "HTTP APIs with Private Integrations," expanding the capabilities of private integrations to the newer HTTP APIs.
- **2022:** Amazon API Gateway added support for JWT-based authorizers for WebSocket APIs, enabling developers to implement authentication and authorization for WebSocket connections.

Throughout its history, Amazon API Gateway has evolved to meet the changing needs of developers, providing a range of features to simplify API management, enhance security, and enable efficient communication between clients and backend services.

ARCHITECTURE OF API GATEWAY

The below diagram shows how the APIs built-in Amazon API Gateway provides customers with an integrated and consistent developer experience for

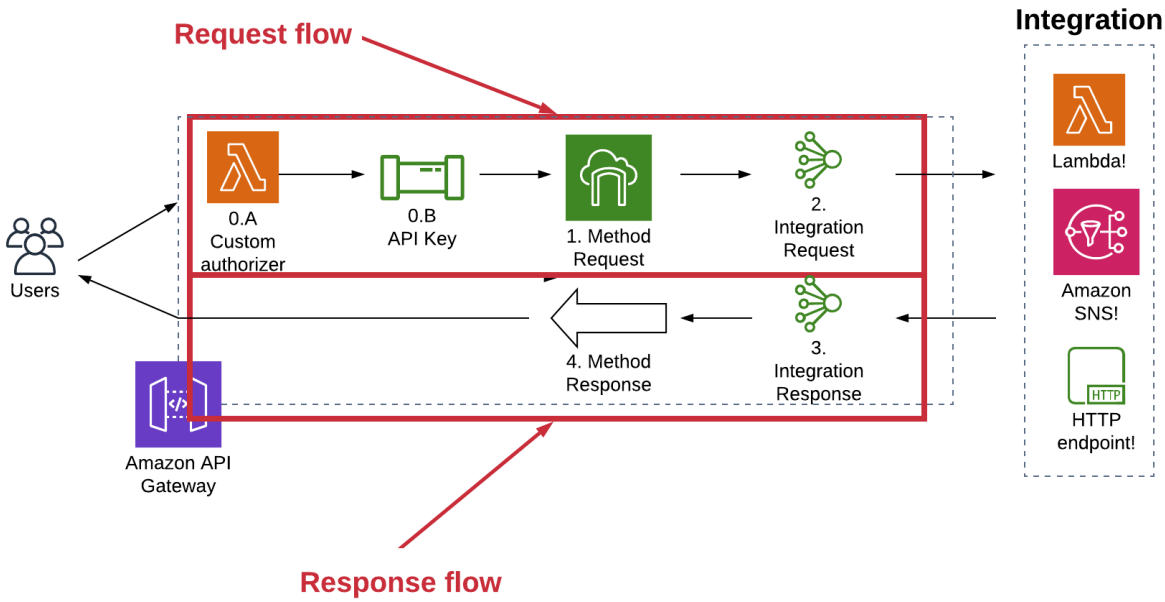
building AWS serverless applications. It acts as a gateway for applications to access data from your backend services such as code running on Amazon Elastic Compute Cloud (Amazon EC2), AWS Lambda, any web applications, etc.



The entire architecture of AWS API Gateway consists of the below key components:

1. **Amazon API Gateway:** It is used to create, publish, secure, maintain and monitor APIs.
2. **API Gateway Cache:** Users can enable API caching to cache their endpoint responses, which can reduce the number of calls made to the endpoint and improve the latency of API requests.
3. **Amazon Cloud Watch:** It is a monitoring and observability service. It collects monitoring and operational data and visualizes it using automated dashboards, which allows users to visually monitor calls to their services.

WORKFLOW OF API GATEWAY



- **Request Flow:** When a client sends a request to the API Gateway, it first passes through the authorization process, where the client's credentials are verified. Then, the request goes through validation and routing stages, which ensure the request is properly formed and directed to the appropriate backend.
- **Integration:** Once the request is validated, it's sent to the configured integration, which can be a Lambda function, an HTTP endpoint, or an AWS service.
- **Response Flow:** The response from the backend follows a similar flow. It passes through validation, authorization, and routing before being returned to the client.

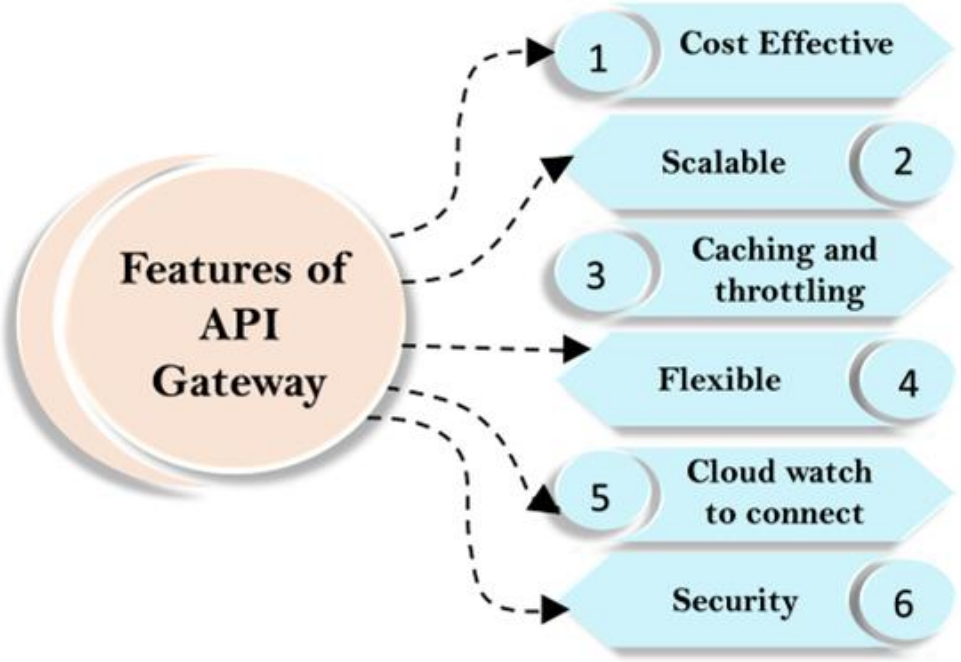
INTEGRATING OPTIONS

- **AWS Lambda:** You can directly integrate API Gateway with AWS Lambda functions. This allows you to execute serverless backend logic in response to API requests. Lambda functions can be triggered by API Gateway, enabling dynamic and customized responses.
- **HTTP Endpoints:** API Gateway can also integrate with HTTP/HTTPS endpoints. This means you can connect to existing resources, like Amazon

EC2 instances or S3 buckets, to expose their functionalities through the API Gateway.

- **AWS Services:** Through integration with AWS services, you can create APIs that interact directly with services like DynamoDB, SNS (Simple Notification Service), or SQS (Simple Queue Service).

FEATURES OF API GATEWAY



- **Cost Effective**: It is very low cost and efficient as an API Gateway provides a tiered pricing model for API requests. The price of an API request is as low as $1.51 per million requests, you can also decrease the costs by decreasing the number of requests.
- **Scalable**: You do not have to worry about having EC2 service or Autoscaling groups responding to API requests. An API Gateway scales automatically.
- **Caching and Throttling**: Caching is the most important feature of API Gateway. Caching is used to cache the endpoint's responses which improve the latency of requests to your API. It is also a primary factor that determines the price of the service.
  You can also prevent the security risks to your API Gateway. If you want to

prevent from flooding with the fraud API calls to API Gateway, you can configure throttle service that can throttle requests to prevent attacks.

- **Flexible**: To implement the API Gateway, you do not have to launch an EC2 instance or setting up the Gateway software. API Gateway can be implemented in few minutes through the AWS Management Console.
- **CloudWatch to Connect**: An Amazon API Gateway is integrated with the CloudWatch service which is a monitoring service. This tool is used to monitor the metrics of incoming API calls, latency and errors.
- **Security:** You can authorize access to your APIs. API Gateway is used to verify incoming requests by executing various authorization options such as Lambda function and Identity Access Management service (IAM). An IAM is integrated with a gateway that provides tools such as AWS credentials, i.e., access and secret keys to access an API. A Lambda function is used to verify tokens, and if tokens are successfully verified, then access to an API will be granted.

COST OF API GATEWAY

You pay for use only, i.e you pay for the API calls you receive and the amount of data transfer. There is an optional data caching charged at an hourly rate that varies based on the cache size you select.

Amazon also provides free tier services for **up to 12 months**, which includes:

- 1 million HTTP API calls
- 1 million REST API calls
- 1 million messages
- 750,000 connection minutes per month

Pricing for the API Calls		
APIs	**No. of requests per month**	**Price (per million)**
HTTP	First 300 million	$1.00
	300+ million	$0.90

Pricing for the API Calls		
APIs	**No. of requests per month**	**Price (per million)**
REST	First 333 million	$3.50
	Next 667 million	$2.80
	Next 19 billion	$2.38
	Over 20 billion	$1.51
WebSocket	First 1 billion	$1.00
	Over 1 billion	$0.80

ADVANTAGE OF USING API GATEWAY

- **Centralized Management:** API Gateways allow you to manage multiple APIs from a single location. This makes it easier to configure, monitor, and maintain your APIs.
- **Security and Access Control:** API Gateways can enforce security mechanisms such as authentication and authorization. They can also protect against attacks like cross-site scripting (XSS) and SQL injection by validating and sanitizing requests.
- **Rate Limiting and Throttling:** API Gateways can enforce rate limits and throttling on incoming requests. This prevents abuse of APIs and ensures fair usage.
- **Caching:** API Gateway can cache responses from APIs, reducing the load on the backend servers and improving response times.

- **Load Balancing**: API Gateways can distribute incoming requests across multiple instances of your APIs, ensuring high availability and optimal resource utilization.
- **Protocol Translation:** API Gateways can handle protocol translations, allowing clients to use different communication protocols while the backend services can use a different protocol.
- **Request and Response Transformation:** API Gateways can modify request and response payloads to match the requirements of the client or the backend service. This can include data transformation, validation, and formatting.
- **Analytics and Monitoring:** API Gateways provide tools for monitoring API usage, performance, and errors. This helps in identifying issues and optimizing APIs.
- **Version Management:** API Gateways can help manage different versions of APIs, allowing clients to use the version they are compatible with while the backend service code evolves.
- **Simplifying Client-Side Logic:** API Gateways can aggregate data from multiple backend services into a single response, reducing the complexity and number of requests on the client side.
- **Cross-Cutting Concerns:** They allow you to handle cross-cutting concerns like logging, auditing, and analytics in a centralized manner.
- **Microservices Architecture Support:** For organizations using microservices, an API Gateway can simplify the communication between services and provide a consistent API interface to external clients.
- **Scalability:** API Gateways can horizontally scale to handle increased traffic and distribute it across backend services.
- **Code Duplication Reduction:** By handling common tasks like authentication, authorization, and request/response transformation in a centralized manner, API Gateway reduces the need for these functionalities to be implemented redundantly across multiple services.
- **Ease of Deployment:** Changes to security policies, routing rules, or transformations can be implemented in the API Gateway without requiring changes to the individual microservices.

DISADVANTAGE OF USING API GATEWAY

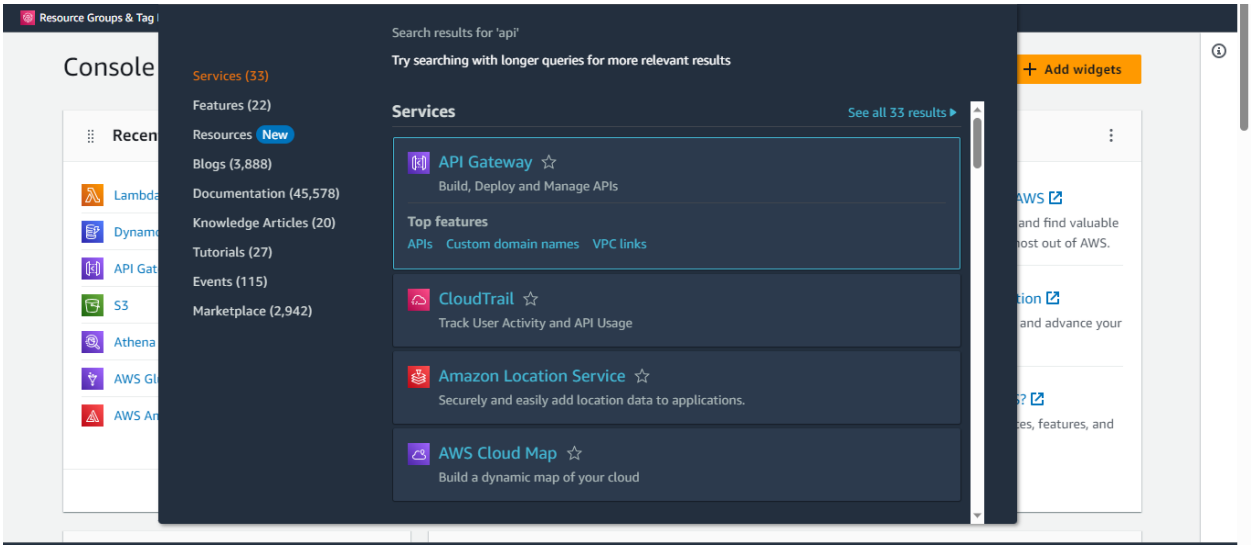
- **Single Point of Failure:** Since an API Gateway act as a central entry point for all API traffic, if the API Gateway goes down, all API communication may be affected, leading to potential service disruptions.
- **Performance Bottleneck:** If not properly designed or if the API Gateway becomes a performance bottleneck due to high traffic, it can degrade the overall system's performance.
- **Complexity:** Implementing and managing an API Gateway can add complexity to your architecture. Configuration, monitoring, and maintenance of the gateway itself can become challenging as the number of APIs and services grows.
- **Increased Latency:** API Gateway adds an additional layer of processing and potential network hops. This can lead to increased latency compared to direct communication between clients and services.
- **Deployment Challenges**: Deploying updates to the API Gateway requires careful coordination, as changes can affect multiple APIs and services simultaneously.
- **Vendor Lock-in:** Some API Gateway solutions are proprietary, which can lead to vendor lock-in. Switching to a different solution could involve significant effort and cost.
- **Limited Protocol Support:** Some API Gateways might have limitations on the protocols and communication patterns they support, potentially causing issues if your services need to use specific protocols.
- **Maintenance Overhead:** Maintaining and upgrading the API Gateway software can require dedicated resources and expertise.
- **Increased Complexity in Troubleshooting:** When issues arise, debugging and troubleshooting might become more complex due to the added layer of the API Gateway.
- **Resource Consumption:** The API Gateway itself consumes resources (CPU, memory, etc.). In high-traffic scenarios, this can have an impact on the overall infrastructure costs.

- **Customization Limitations:** Some API Gateways might not support all the customization needs of your APIs. Complex or unique requirements might be difficult to implement within the constraints of the Gateway.
- **Scalability Challenges:** While API Gateways can help distribute traffic to backend services, scaling the API Gateway itself can become challenging as the traffic grows.
- **Learning Curve:** If your team is not familiar with the chosen API Gateway solution, there might be a learning curve to understand and effectively manage it.
- **Dependency on Third-party Solutions:** By relying heavily on an API Gateway, you become dependent on its stability, performance, and feature set. Any issues with the Gateway can directly impact your services.
- **Network Overhead:** Depending on the Gateway's architecture and features, there might be additional network overhead introduced for requests to pass through the Gateway.

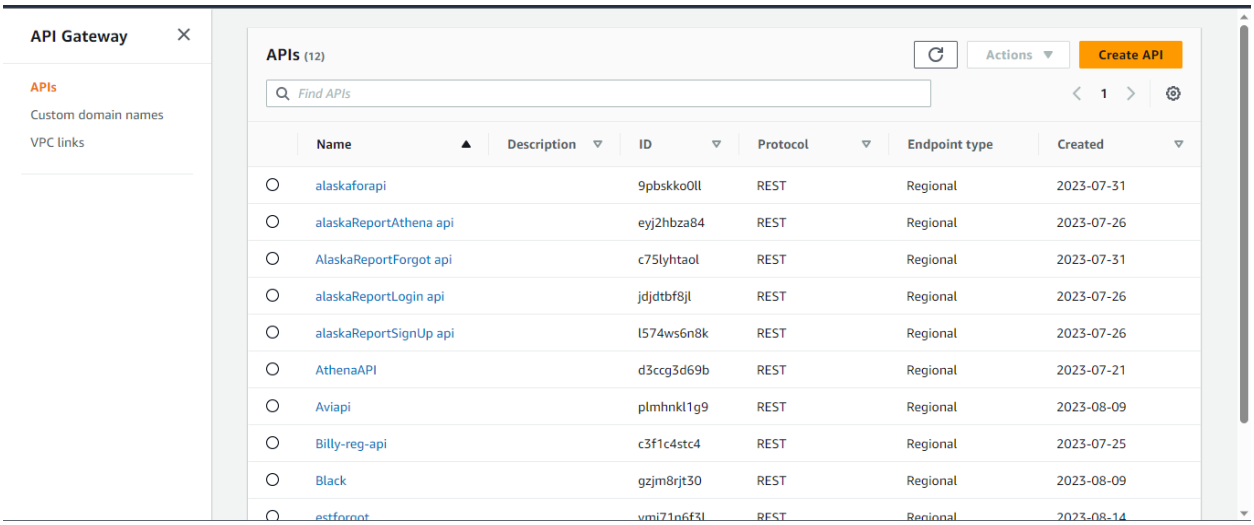
HOW TO CONNECT API GATEWAY

STEPS TO CREATE API GATEWAY

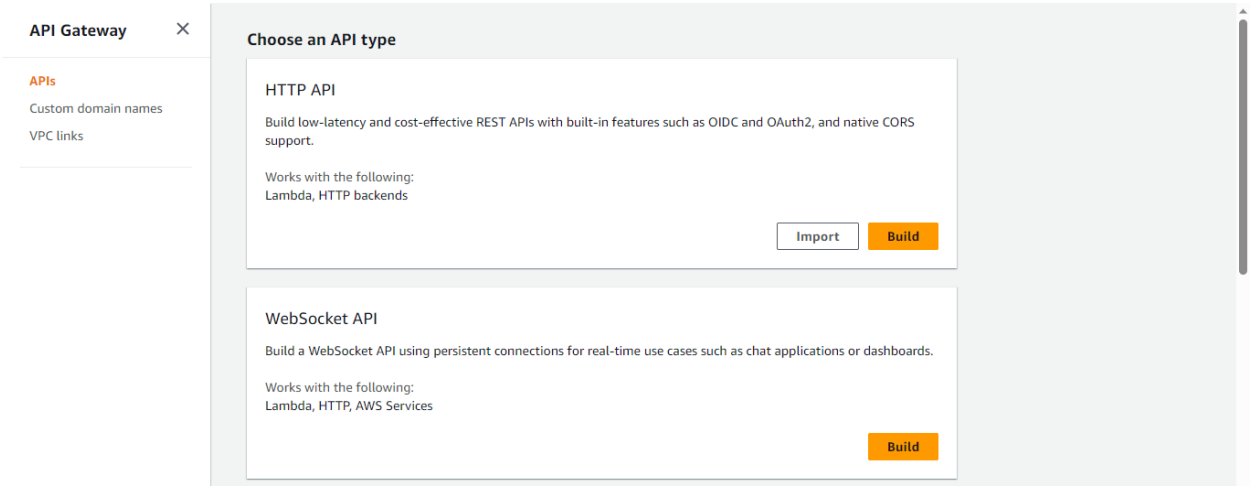
STEP 1: Log in to your AWS Management Console using your AWS account credentials. Once logged in, navigate to the "API Gateway" service. You can find this under the "Networking & Content Delivery" section.

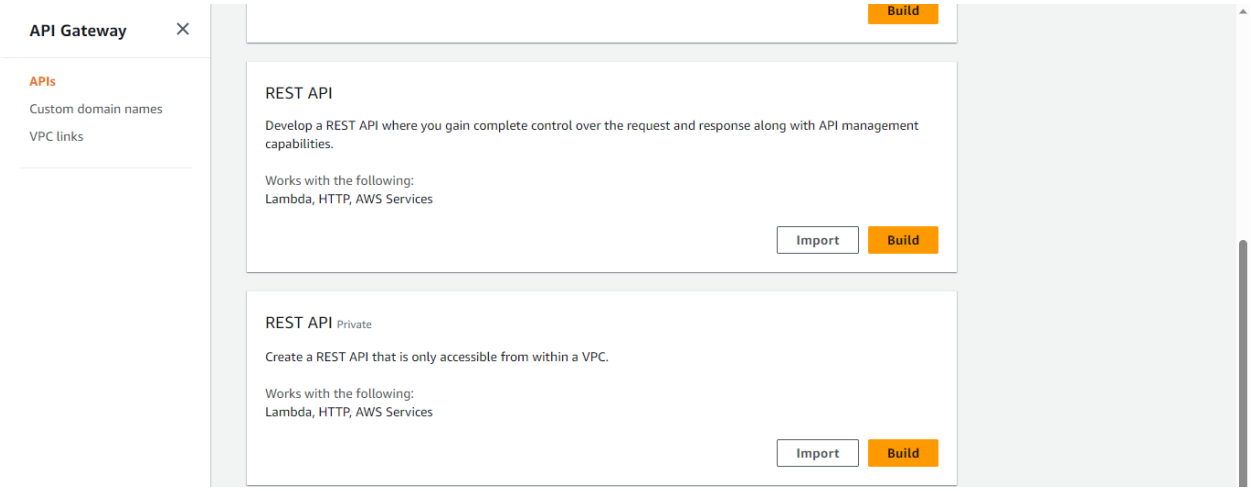


STEP 2: Click on the "Create API" button to start creating a new API.

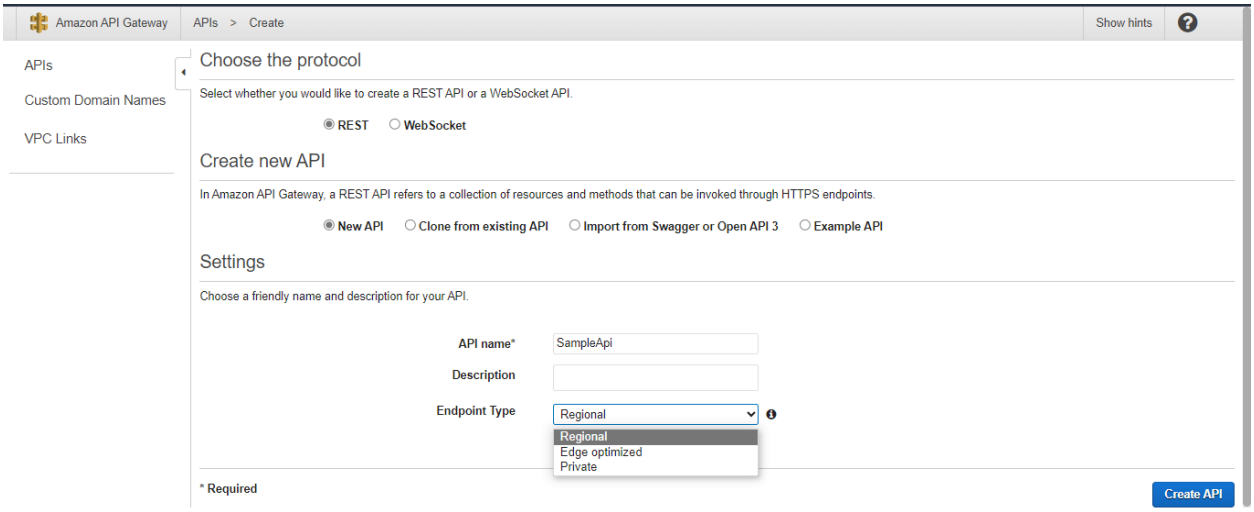


STEP 3: Select the protocol for your API. You can choose between HTTP and WebSocket, depending on the type of communication your API will handle.

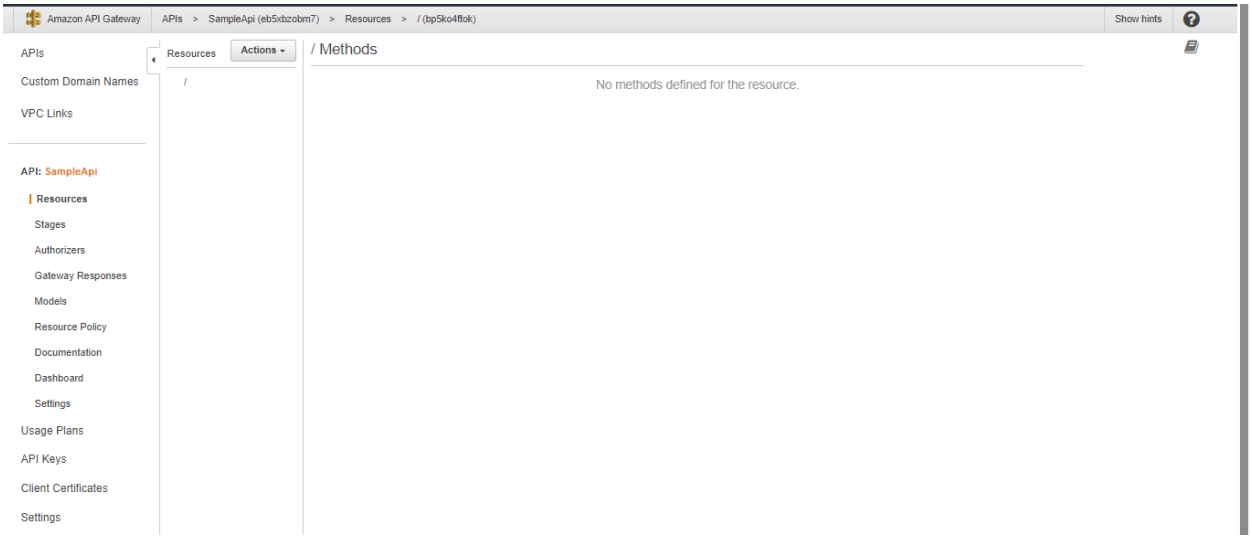




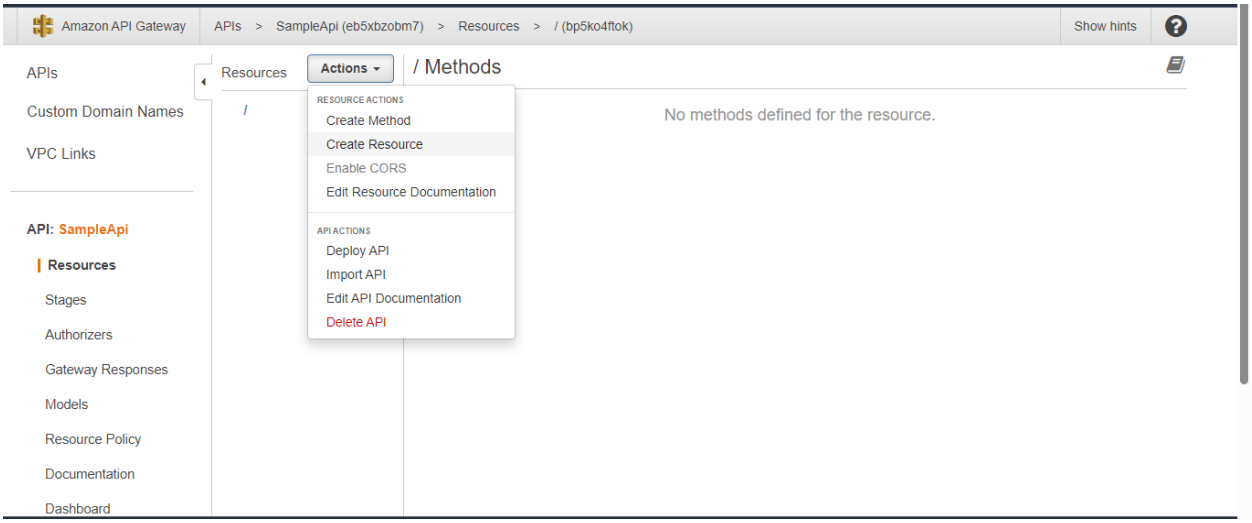
STEP 4: Provide a name for your API and a description. You can also set up optional features like API endpoint type (edge-optimized, regional, or private), and whether to enable CORS (Cross-Origin Resource Sharing) for web browser clients.



STEP 5: After creating the API, you'll need to define resources and methods. Resources represent the different endpoints of your API, and methods define the actions that can be performed on those endpoints (e.g., GET, POST, PUT, DELETE).

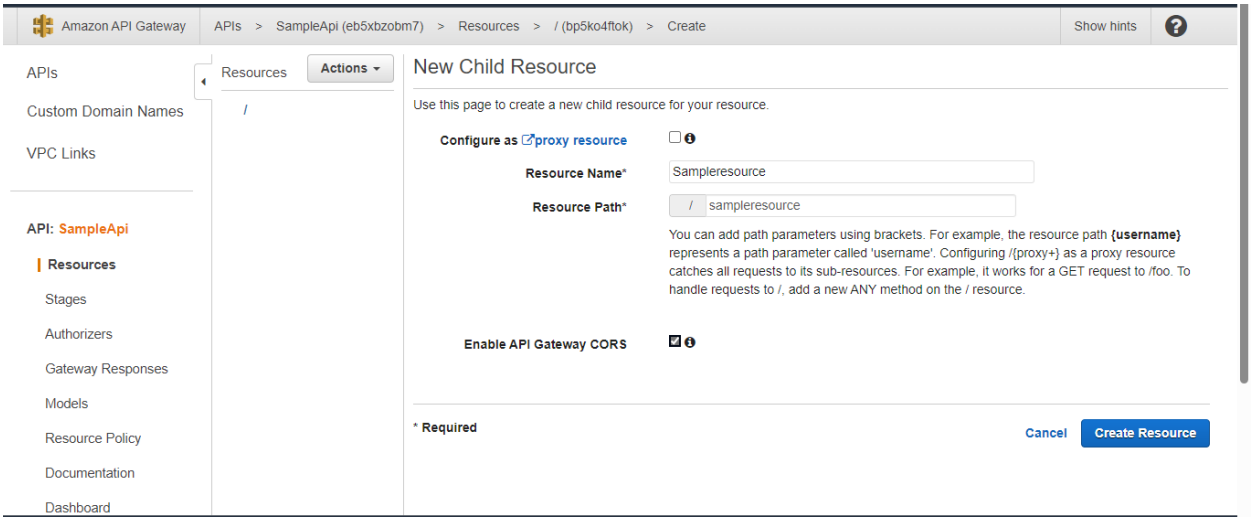


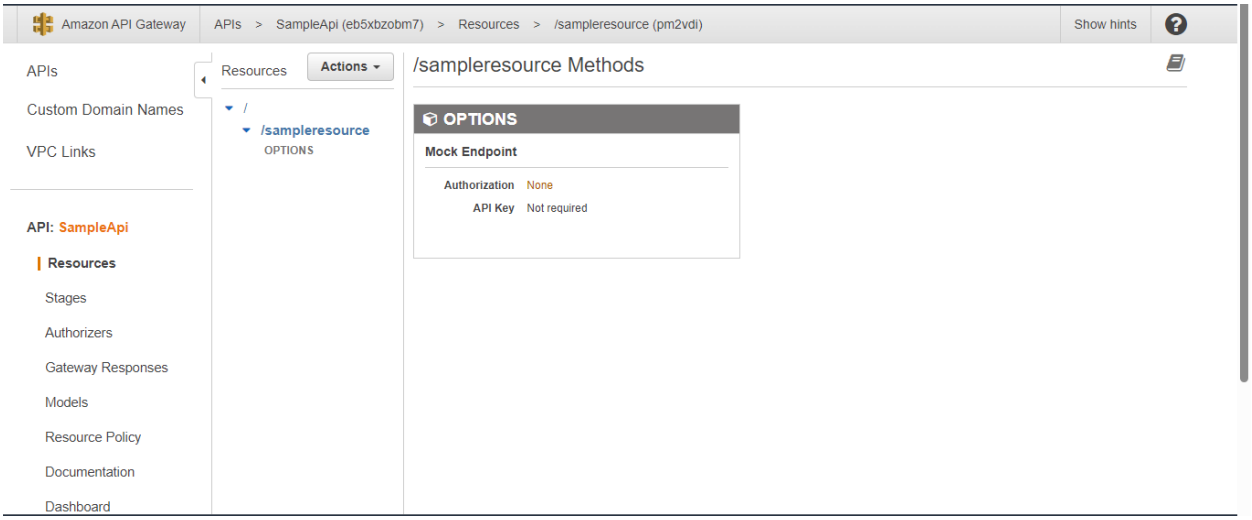
STEP 6: After selecting an API or creating a new one, you'll be in the API's main dashboard. On the left sidebar, click on the "Resources" link



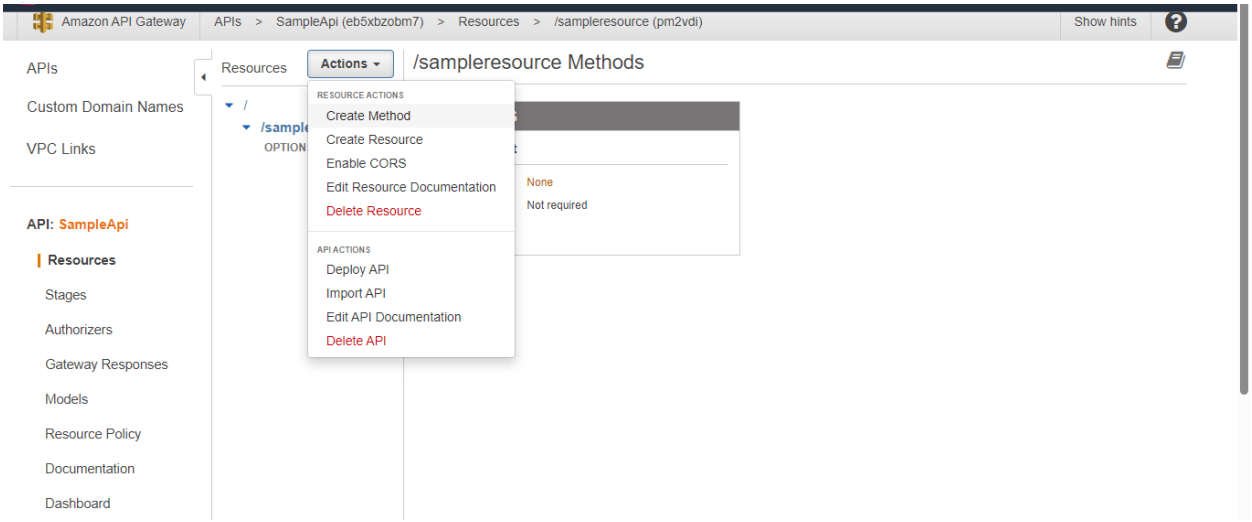
STEP 7: Click on the "Create Resource" button. In the "Resource Name" field, enter a name for your resource. This name becomes a part of the URL path for your API endpoint. For example, if you enter "users", your endpoint could be something like `https://your-api-id.execute-api.your-region.amazonaws.com/users`.

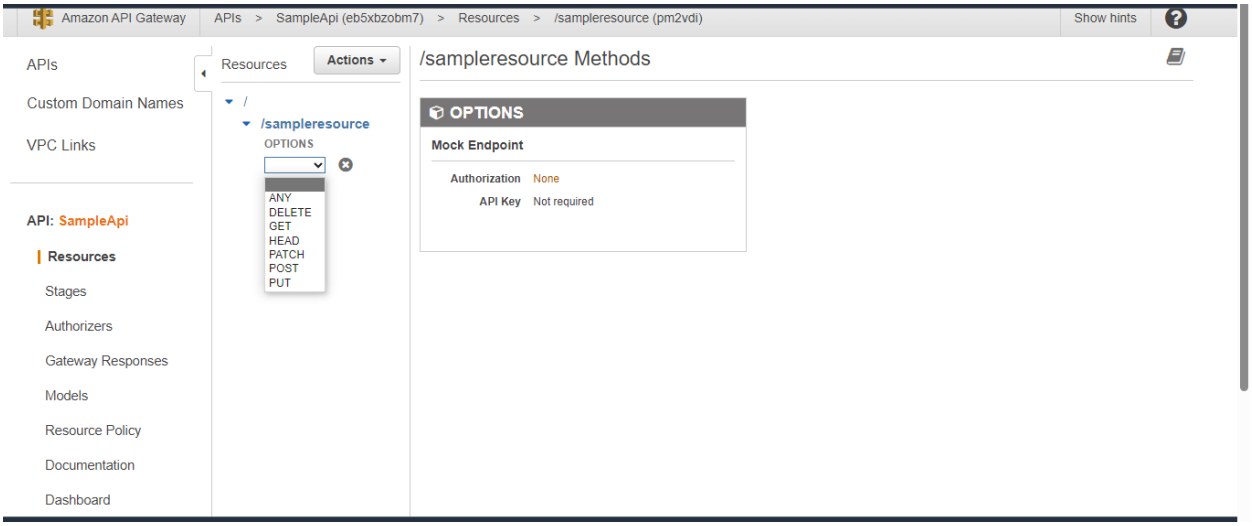
STEP 8: Once you've named the resource, you can configure additional settings such as enabling API Gateway CORS support, setting up request and response models, and more.





STEP 9: With the resource created, you can now define methods that correspond to HTTP verbs (e.g., GET, POST, PUT, DELETE). Click on the resource you just created, then click on the "Create Method" button. Choose the HTTP verb you want to associate with the method.

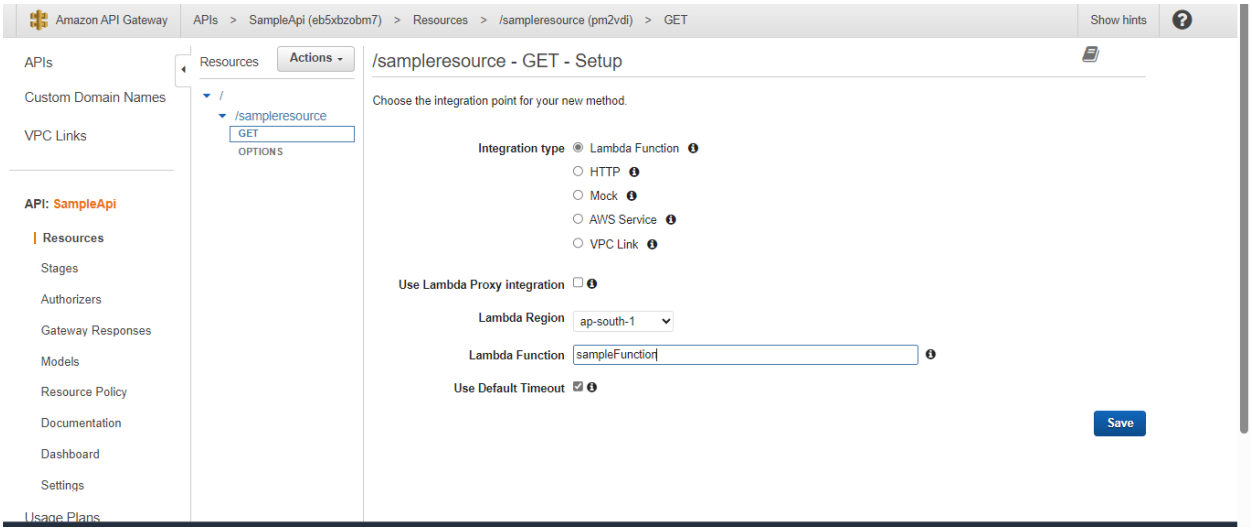


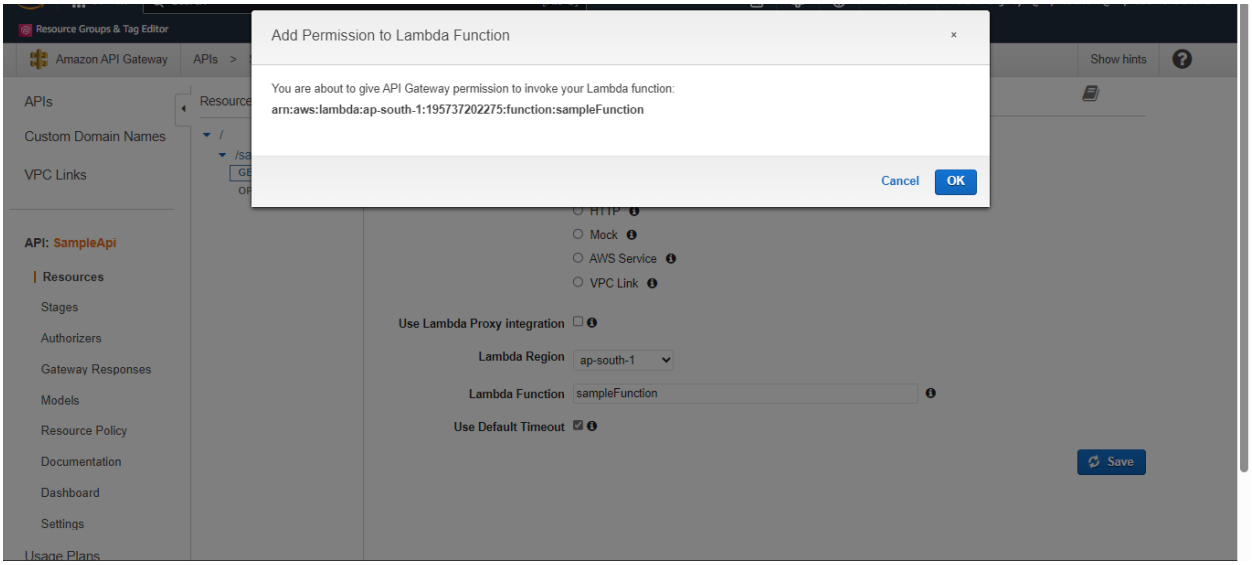


Other options for an API method include:

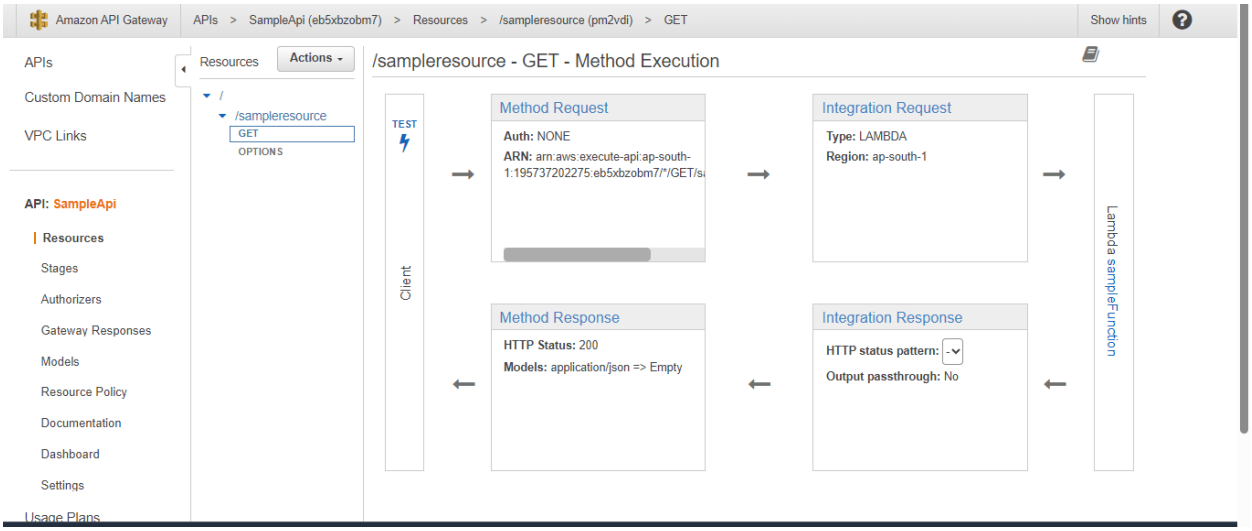
- **POST**, primarily used to create child resources.
- **PUT**, primarily used to update existing resources (and, although not recommended, can be used to create child resources).
- **DELETE**, used to delete resources.
- **PATCH**, used to update resources.
- **HEAD**, primarily used in testing scenarios. It is the same as GET but does not return the resource representation.
- **OPTIONS**, which can be used by callers to get information about available communication options for the target service.

STEP 10: For each method, you'll need to configure how the method integrates with your backend service or Lambda function. You can choose from various integration types such as Lambda Function, HTTP, Mock, AWS Service, etc.

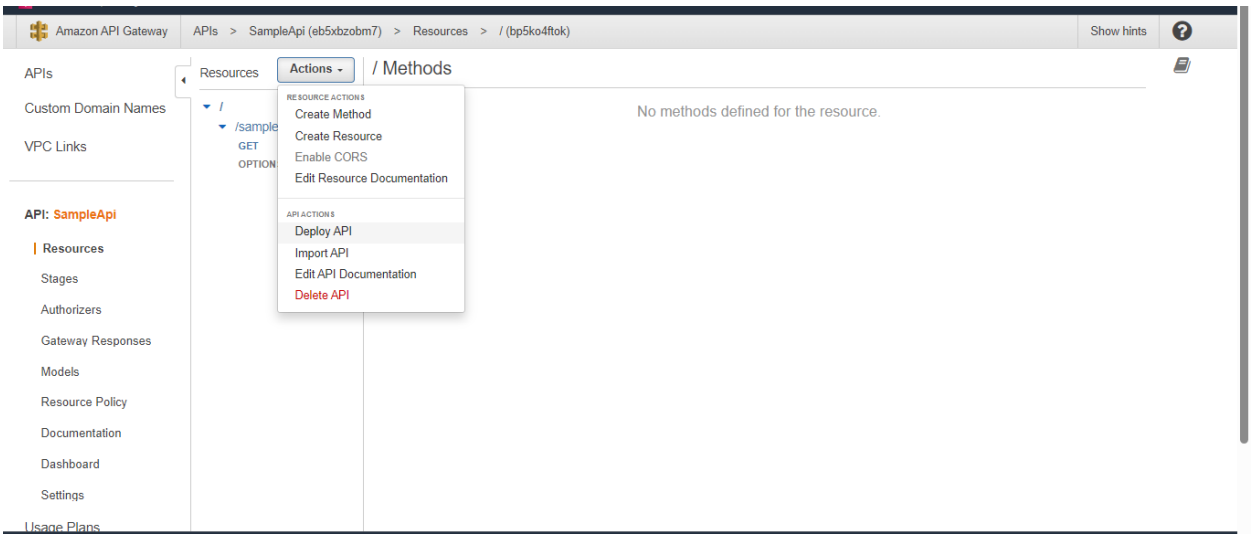


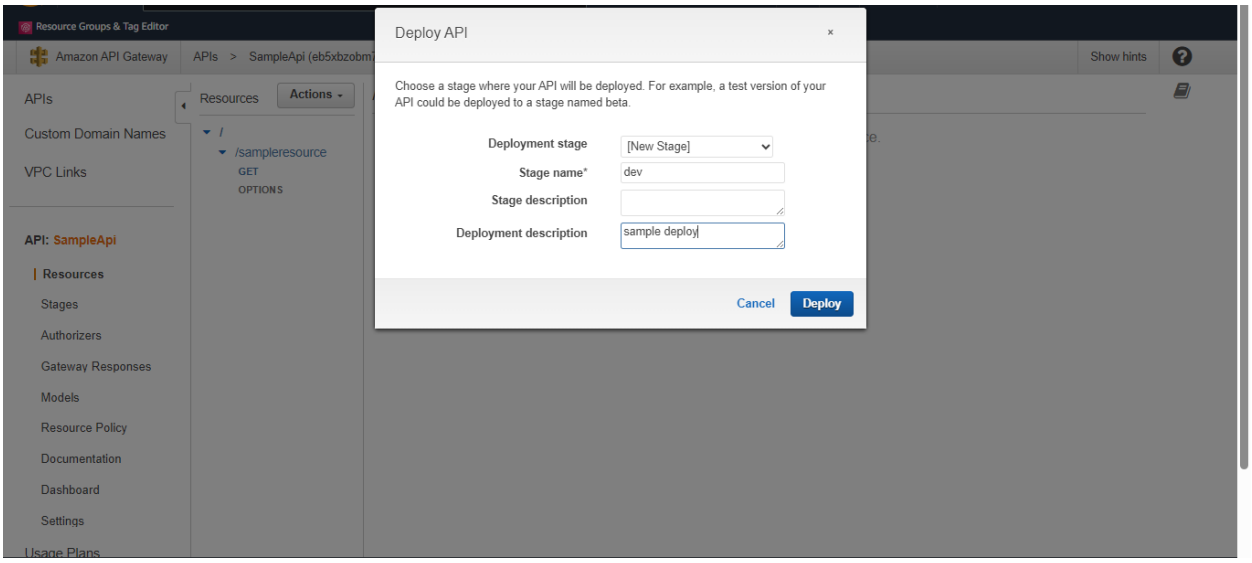


STEP 11: Define the method request and response settings. This includes configuring request parameters, headers, query strings, and mapping templates if needed.

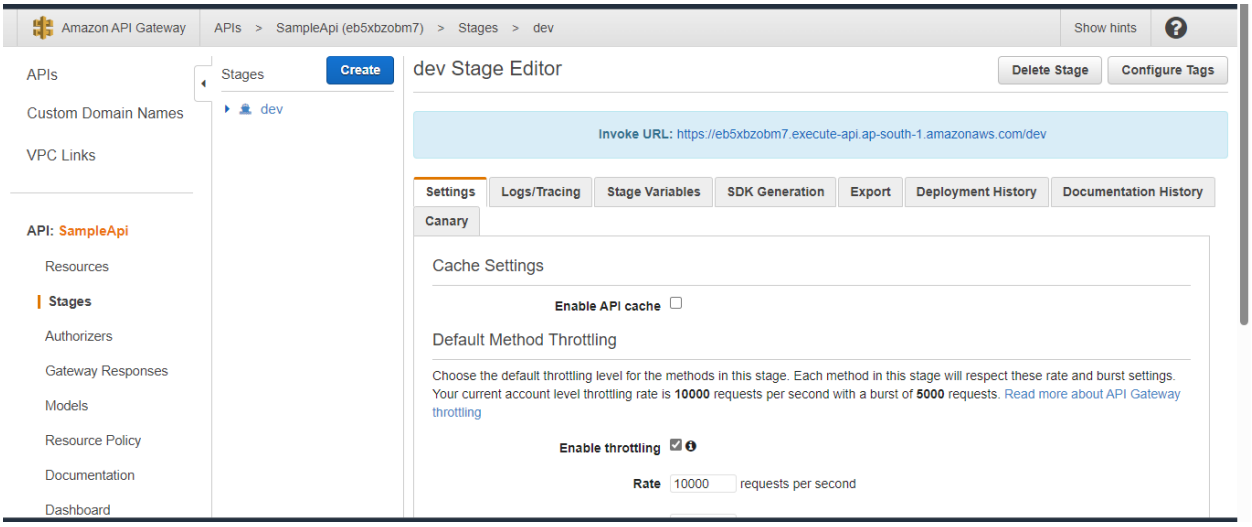


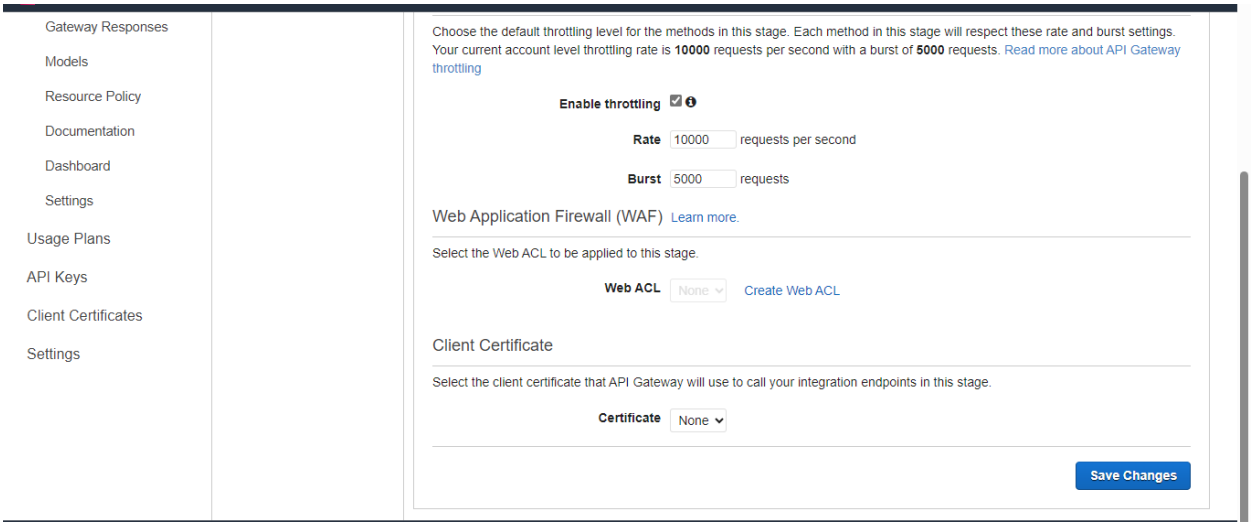
STEP 12: Once your resource and methods are configured, you need to deploy the API to make it accessible. Click on the "Deploy API" button and create a new stage (e.g., "prod" for production). This will provide you with a URL where clients can access your API.





STEP 13: Use the provided URL or a tool like Postman to test the resource and methods you've created. Send requests to the appropriate endpoints using the defined HTTP verbs and check the responses.





INTEGRATING TYPES

API Gateway supports various integration types to connect your APIs to backend services, and these integration types allow you to define how requests from clients are forwarded and processed. Here are the common integration types in API Gateway:

1. **HTTP:** This integration type allows you to connect your API to an HTTP endpoint. It can be an HTTP web service, a Lambda function, or any other HTTP resource.
2. **Lambda Function:** This type of integration connects your API directly to an AWS Lambda function. When a request is made to the API, it triggers the corresponding Lambda function.
3. **Mock:** The Mock integration type is used for testing and mocking purposes. It allows you to create API methods that return static responses without connecting to a backend service.
4. **AWS Service:** This integration type lets you integrate your API directly with AWS services like AWS Step Functions or AWS S3.
5. **VPC Link:** VPC (Virtual Private Cloud) Link integration enables you to connect your API to private resources in your Virtual Private Cloud, like EC2 instances or containers.

CONFIGURATION STEPS FOR DIFFERENT INTEGRATION TYPES WHEN SETTING UP AN API GATEWAY ENDPOINT

1. HTTP Integration:

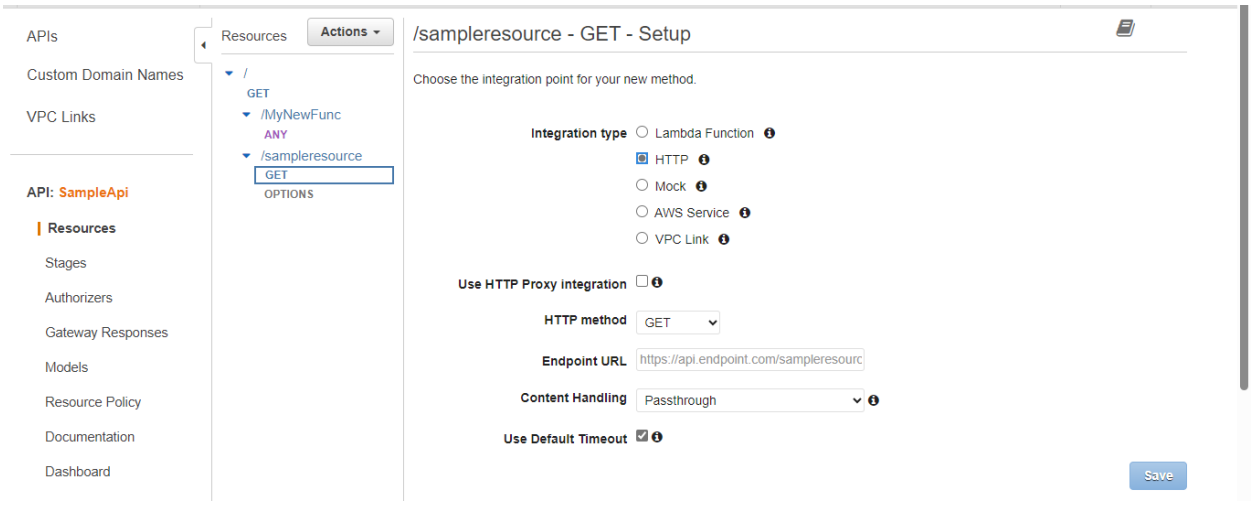
- Integration Type: HTTP

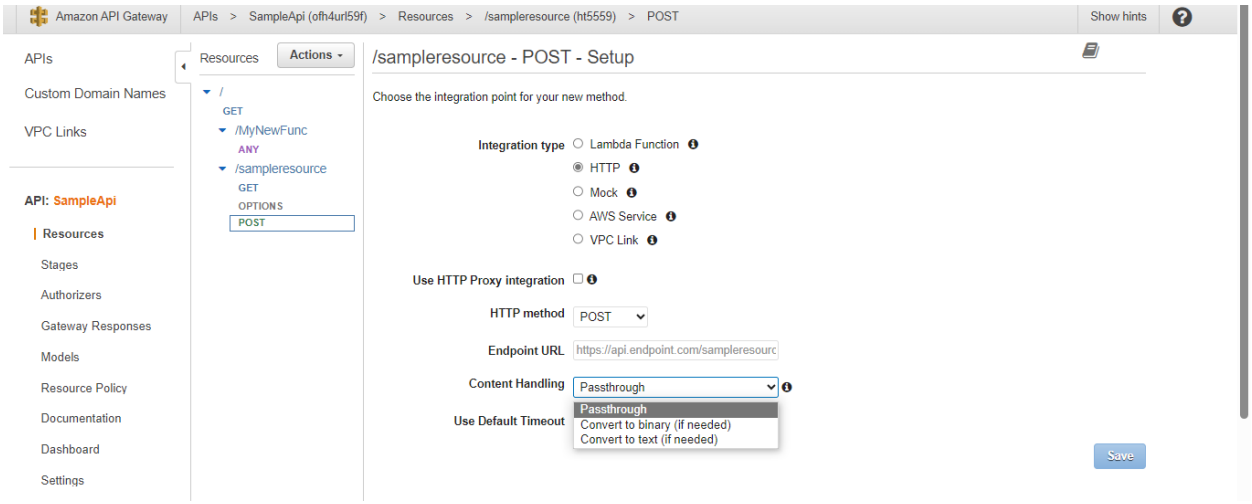
STEP 1: Select your API in the API Gateway console.

STEP 2: Choose the desired resource and method for which you want to set up the HTTP integration.

STEP 3: Under the "Integration" section, choose "HTTP" as the integration type.

STEP 4: In the "Endpoint URL" field, provide the URL of the backend HTTP service.





In the context of an API gateway, the "Content Handling" option typically refers to how the API gateway handles the content (payload) of incoming requests and outgoing responses. This option allows you to configure how the API gateway processes and transforms the data as it passes between clients and backend services.

1. Passthrough:

- With "Passthrough" content handling, the API gateway acts as a conduit, forwarding data between clients and backend services without modifying its content.
- This is useful when you want minimal interference with the data and need to maintain its integrity during transfer.

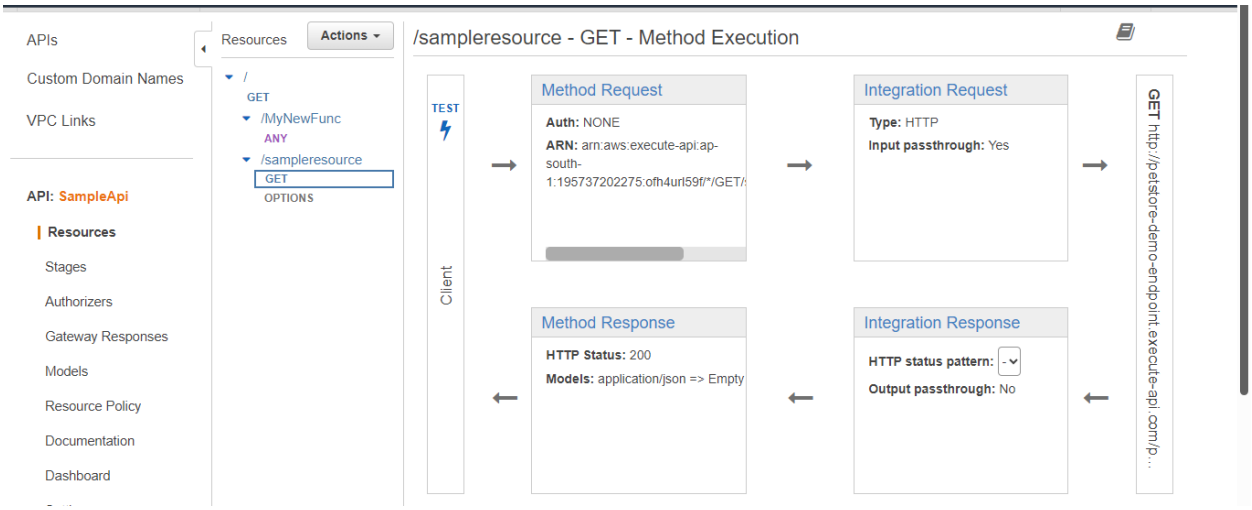
2. Convert to Binary:

- "Convert to Binary" treats all data as binary content, regardless of its original format.
- This option is useful when you want to prevent the API gateway from interpreting or modifying data structure, often used for custom or proprietary formats.

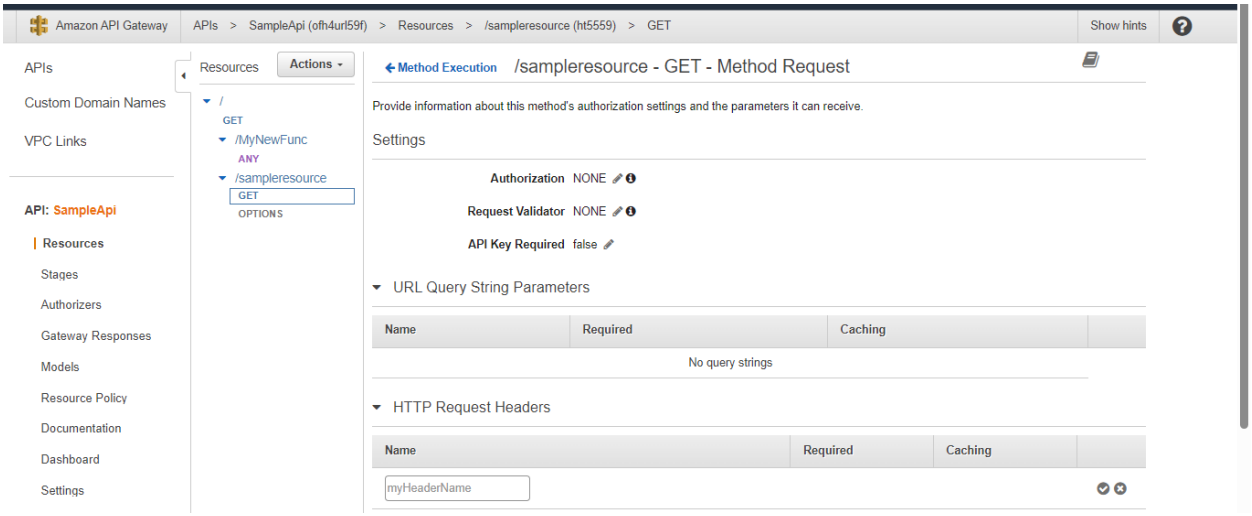
3. Convert to Text:

- "Convert to Text" interprets incoming binary data as text and might involve decoding operations.
- This option is suitable when you have binary data that should be transformed into a textual format before being passed to backend services.

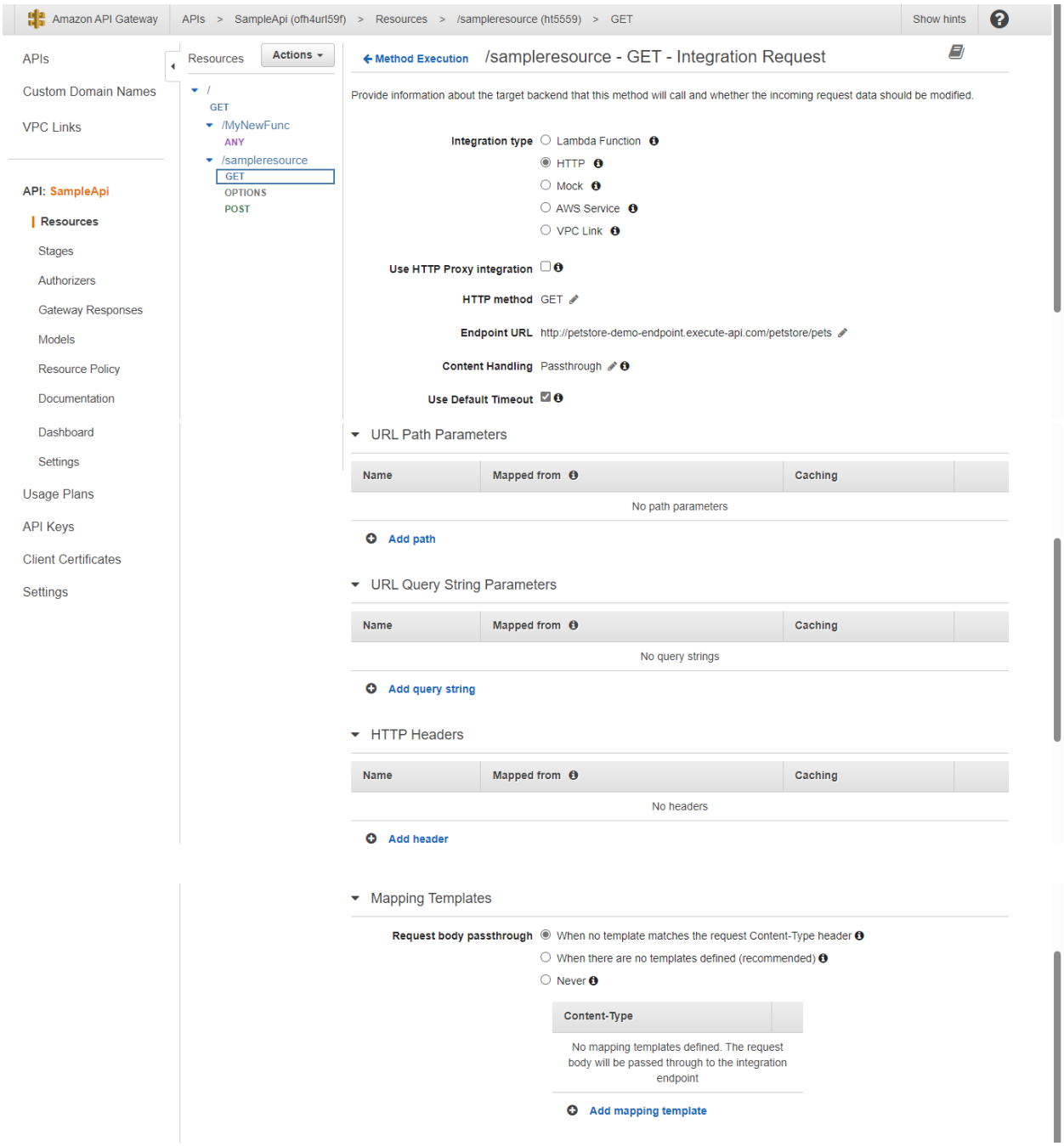
STEP 5: When the method setup finishes, you are presented with the **Method Execution** pane.



STEP 5: you can further configure the method request to add query string or custom header parameters.



STEP 6: You can also update the integration request to map input data from the method request to the format required by the back end.



STEP 7: Test the integration using the "Test" button.

STEP 8: Deploy the API to a stage to make it accessible via an endpoint URL.

2. Lambda Function Integration:

- Integration Type: Lambda Function

STEP 1: Select your API in the API Gateway console.

STEP 2: Choose the desired resource and method for which you want to set up the Lambda integration.

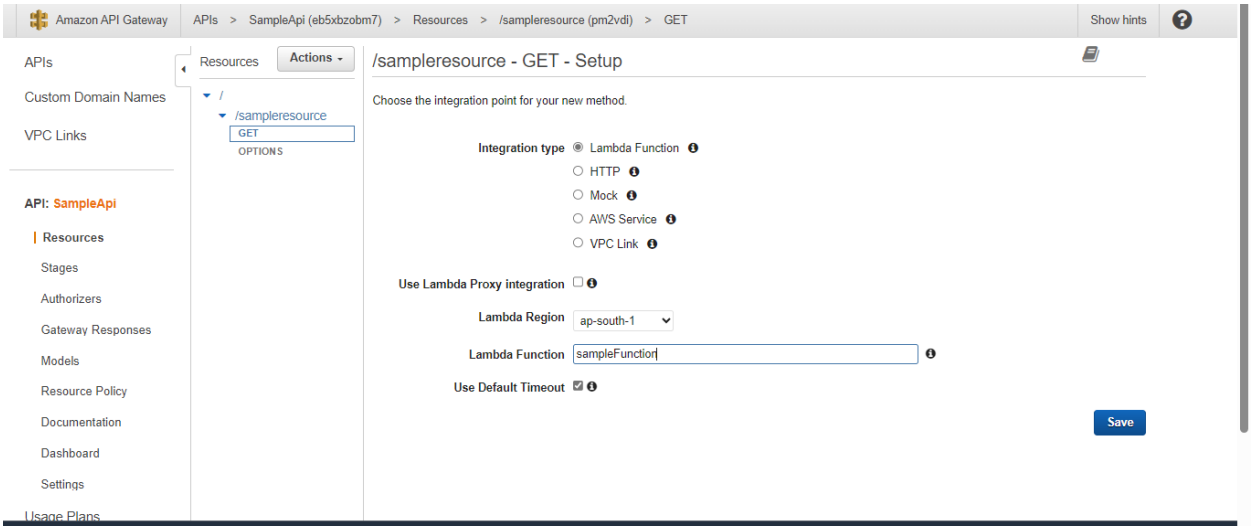
STEP 3: Under the "Integration" section, choose "Lambda Function" as the integration type.

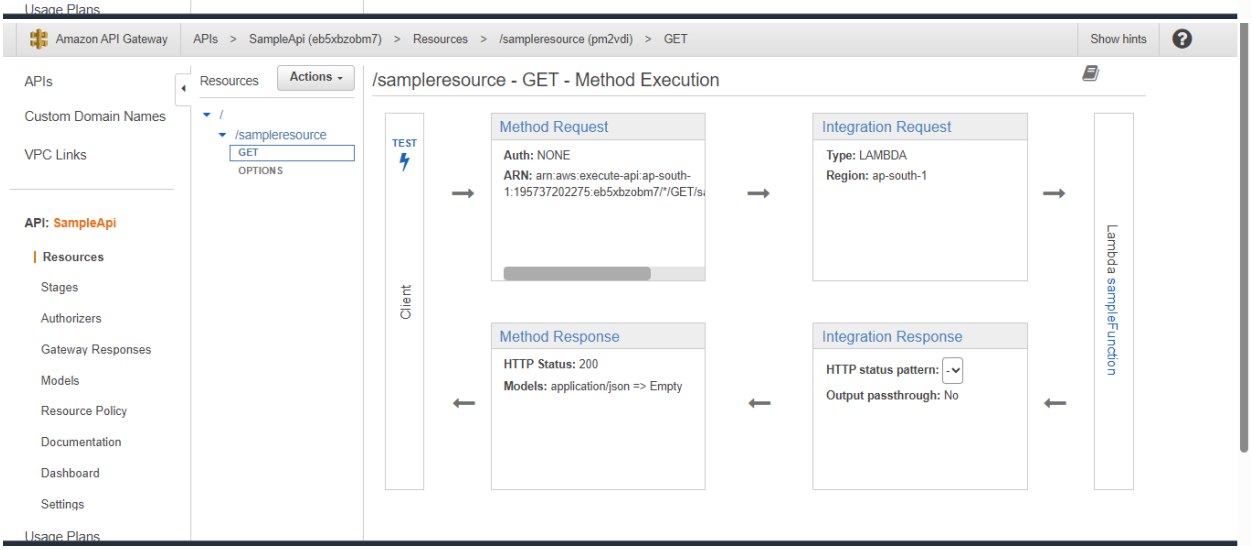
STEP 4: Select the Lambda function you want to invoke from the dropdown list.

STEP 5: Configure any necessary input mappings in the "Mapping Templates" section to map the incoming request to the Lambda function's event input.

STEP 6: Test the integration using the "Test" button.

STEP 7: Deploy the API to a stage to make it accessible via an endpoint URL.





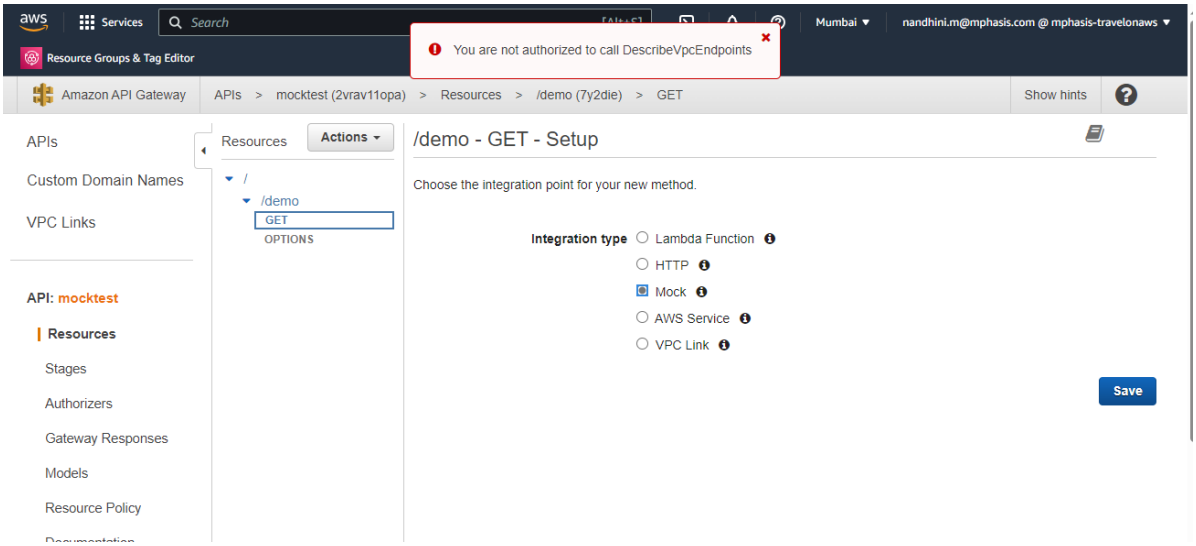
3. Mock Integration:

- Integration Type: Mock

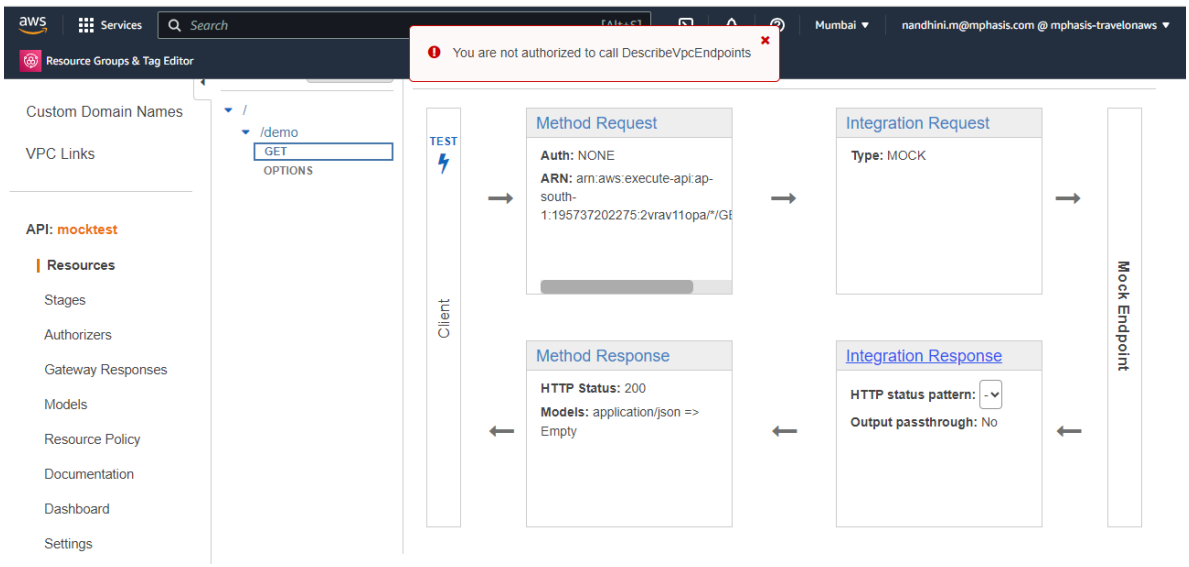
STEP 1: Select your API in the API Gateway console.

STEP 2: Choose the desired resource and method for which you want to set up the mock integration.

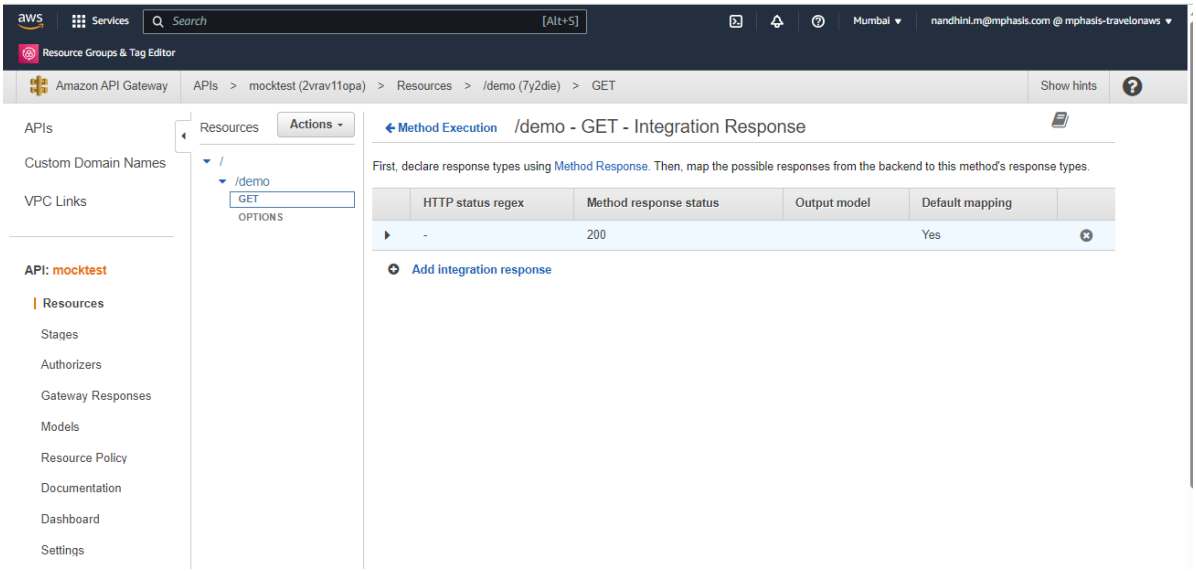
STEP 3: Under the "Integration" section, choose "Mock" as the integration type.



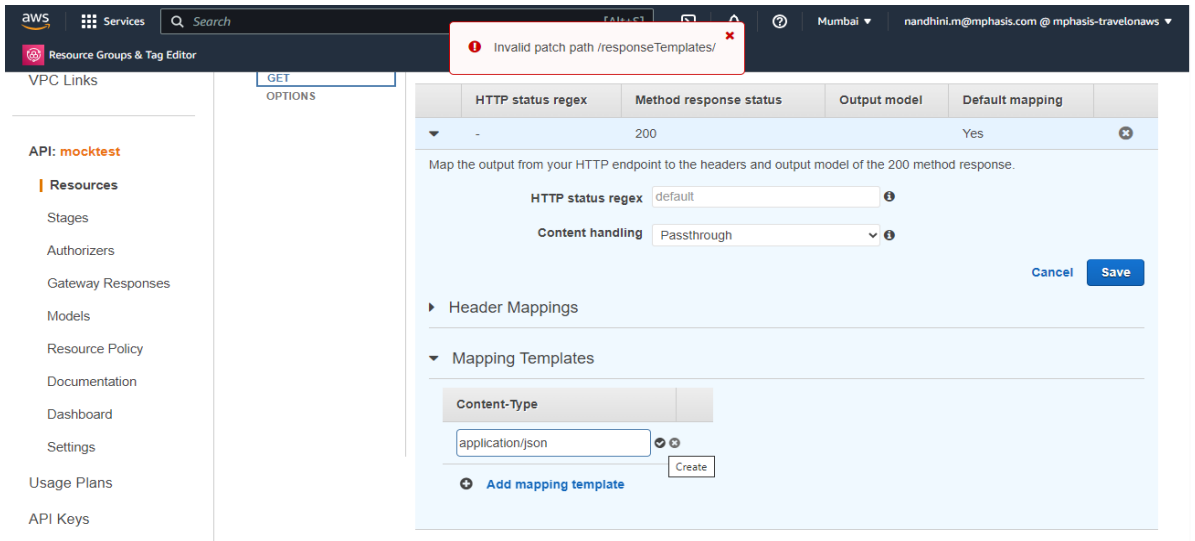
STEP 4: Click on "Add Integration Response" to configure mock responses for different HTTP status codes.



STEP 5: Click on the HTTP status code "200" to configure the mock response for success.



STEP 6: In the "Body Mapping Templates" section, click on "application/json" to configure a response in JSON format.



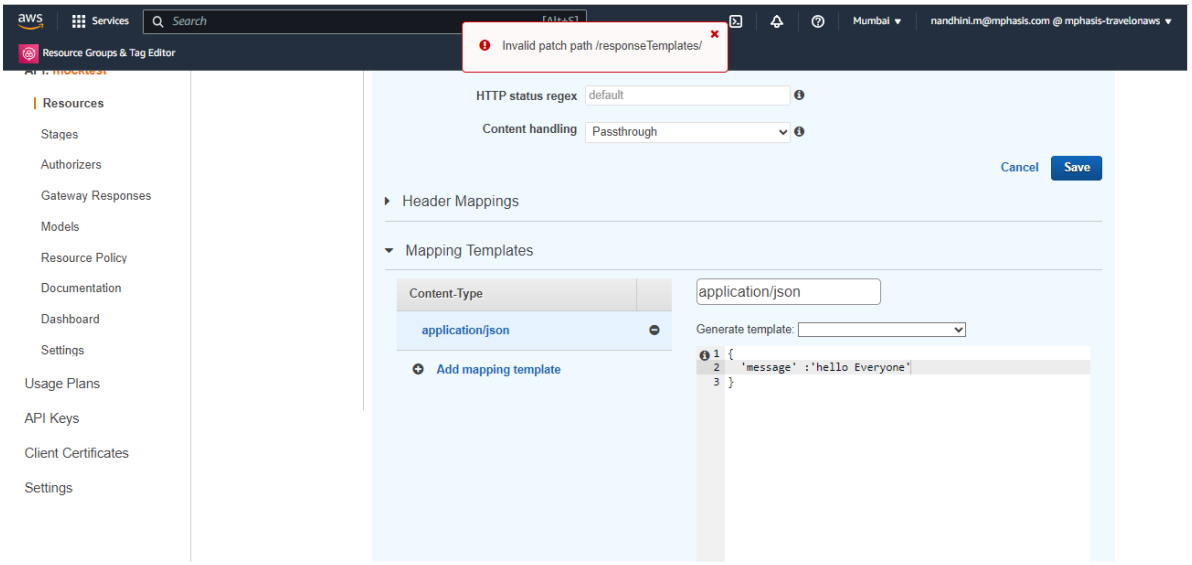
STEP 7: In the "Generate Template" box, enter the following JSON response template:

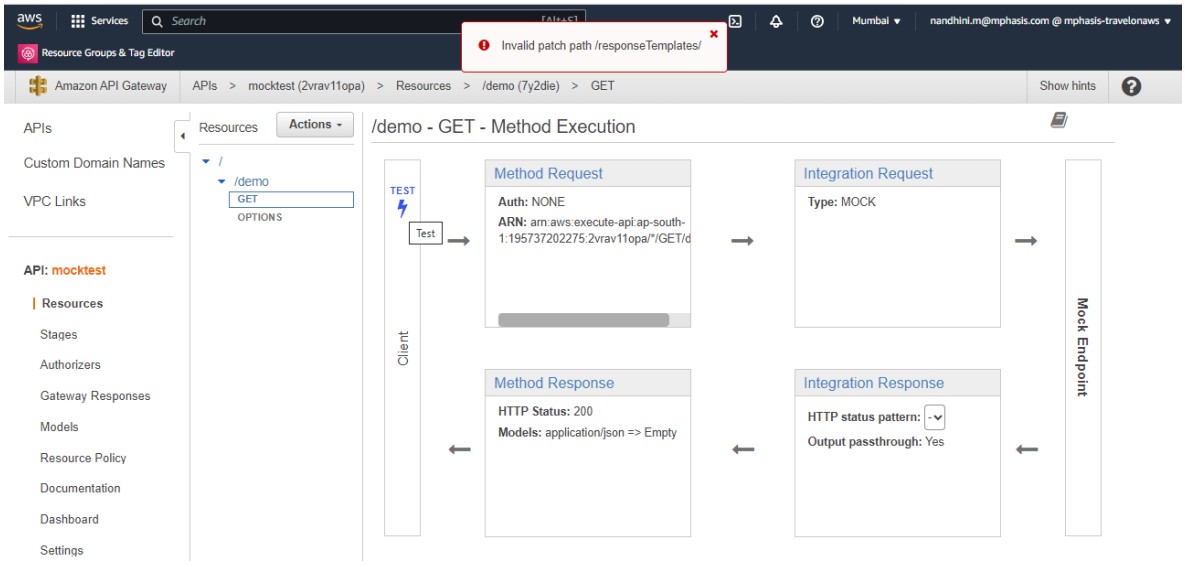
{

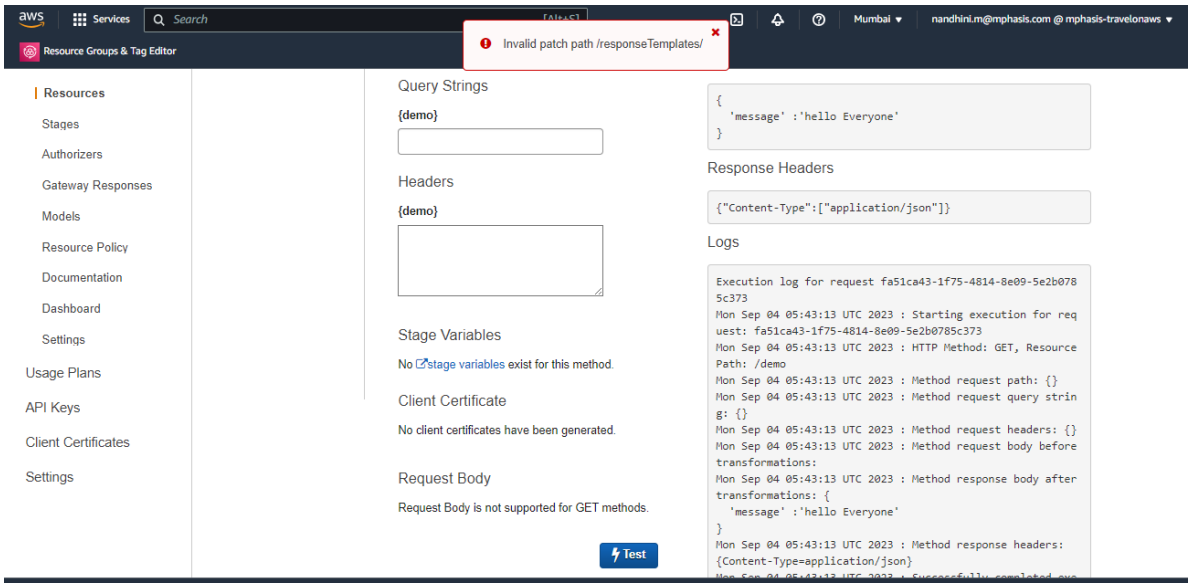
"message": "Hello Everyone"

}

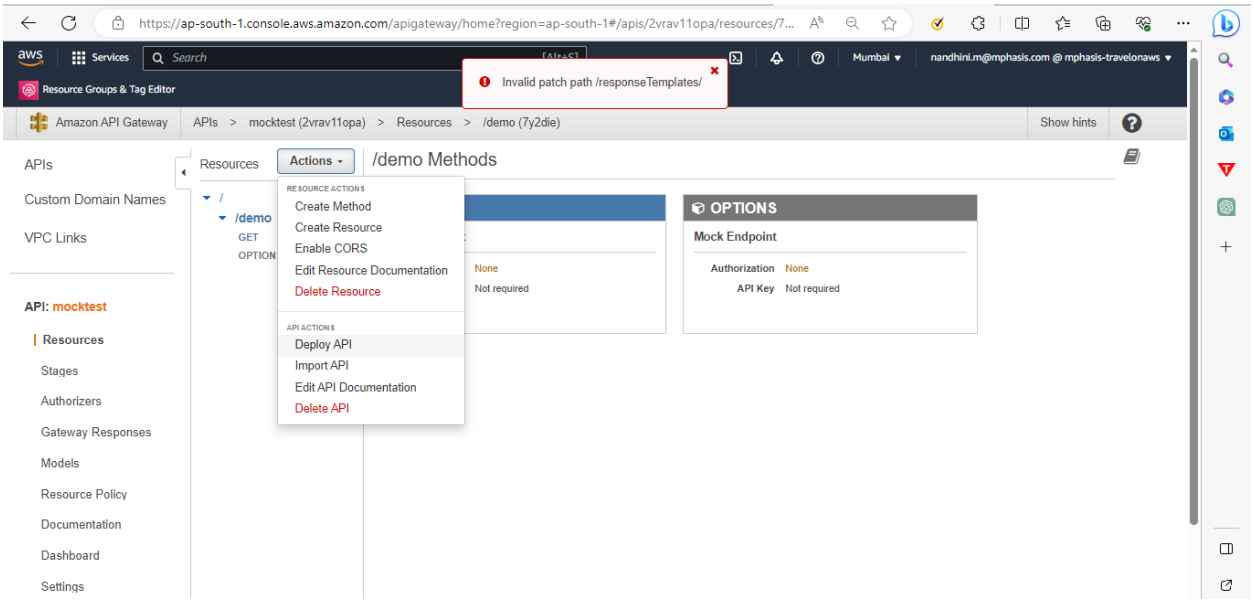
This template defines a simple JSON object with a "message" field containing the "Hello Everyone" message.



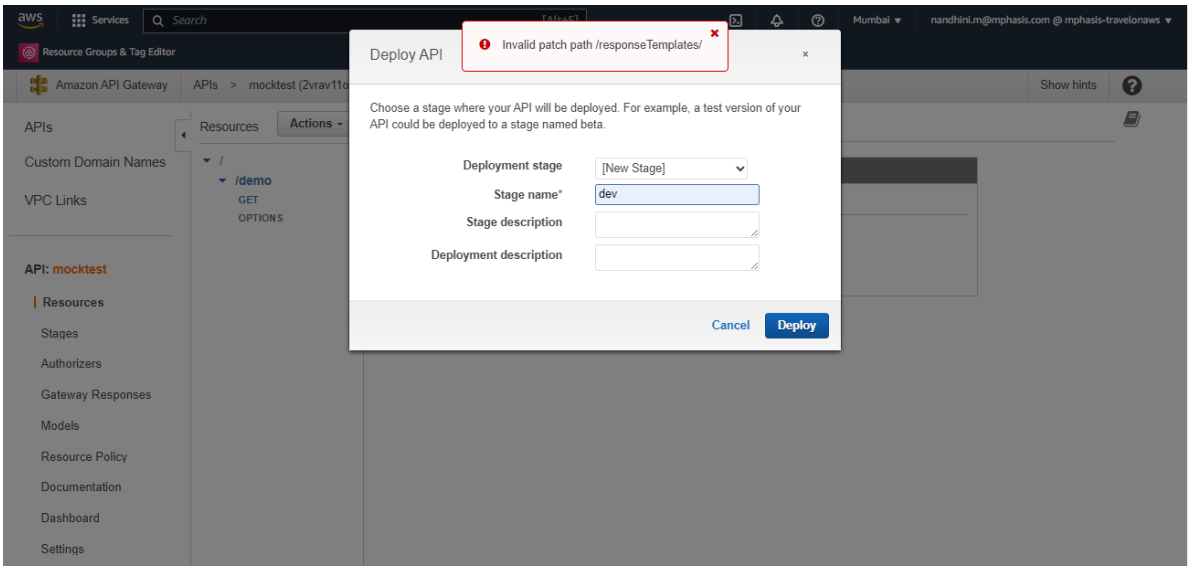




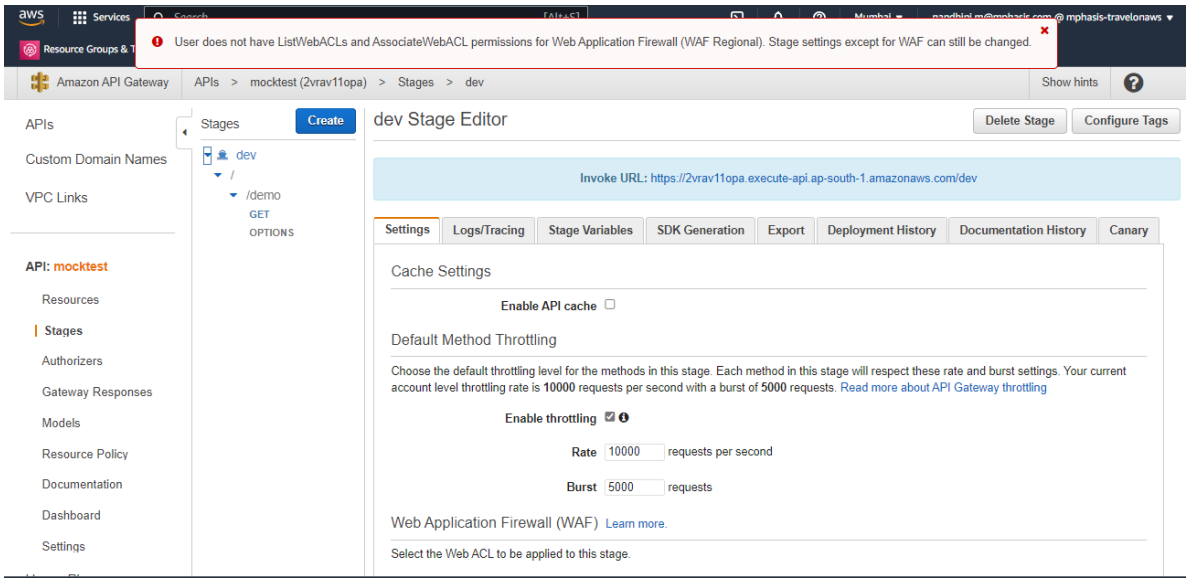
STEP 8: After configuring the mock integration response, deploy your API to make it accessible via a URL. Click on "Deploy API" under the "Actions" menu.



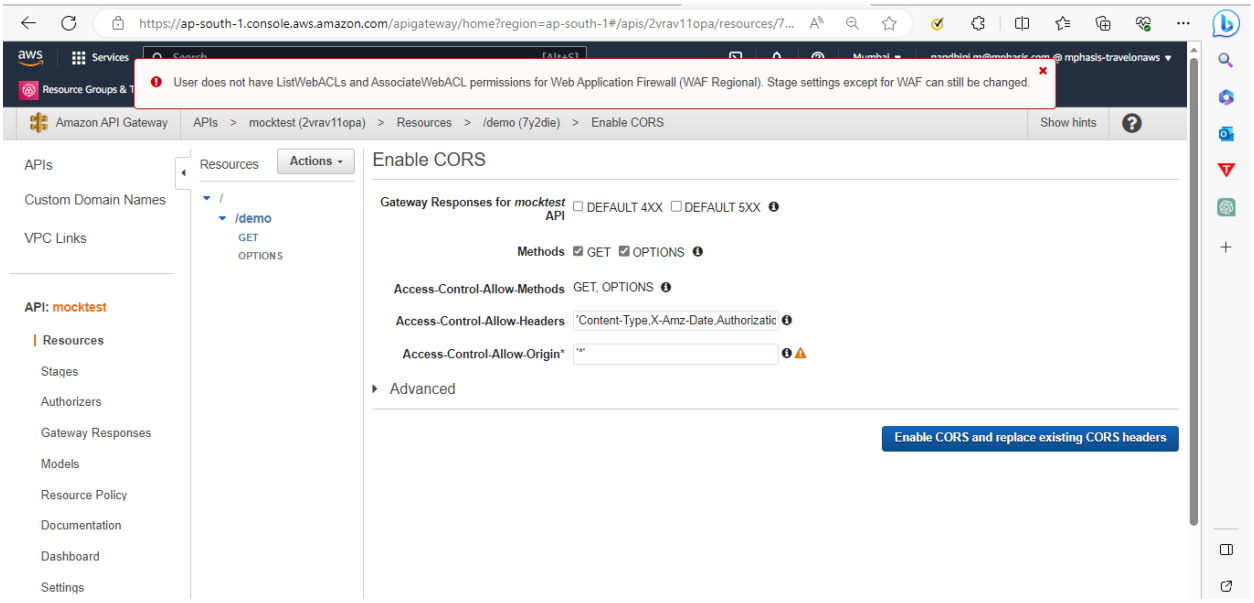
STEP 9: Choose or create a deployment stage (e.g., "test" or "dev") and click "Deploy."

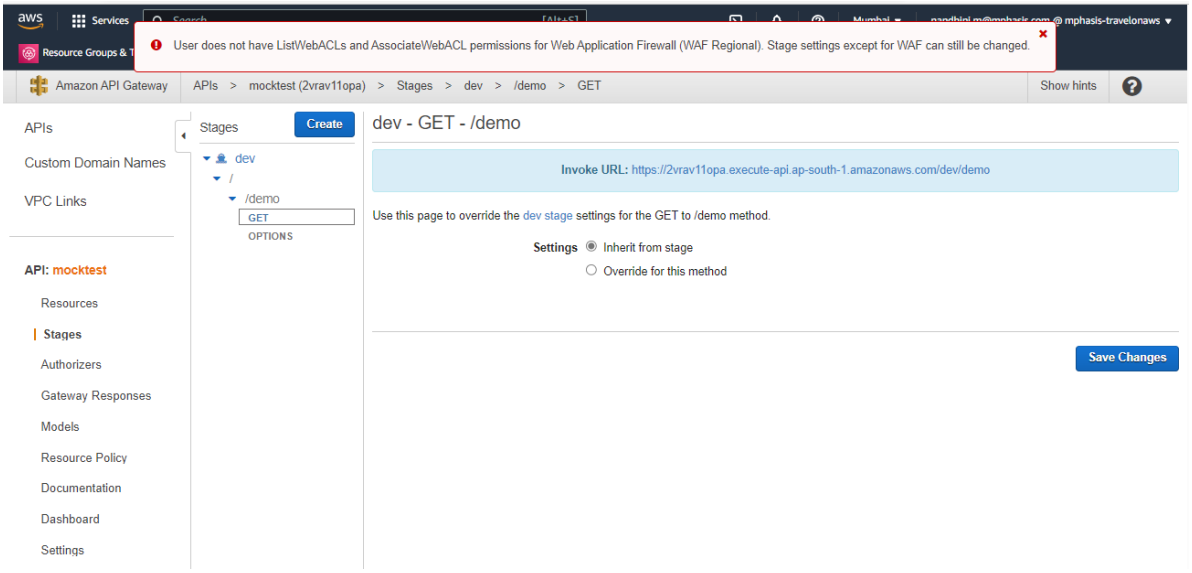


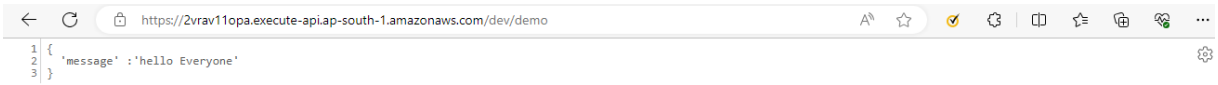
STEP 10: Once the API is deployed, you will receive an invoke URL for your mock API stage.

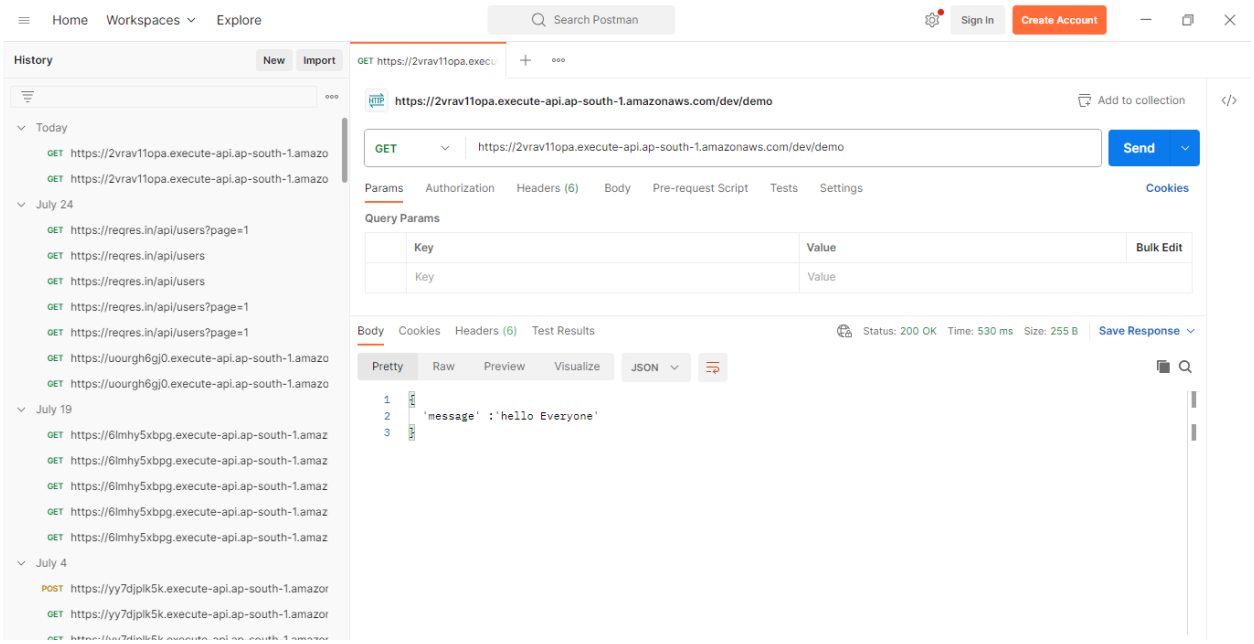


STEP 11: After deploying click on "Enable CORS" under the "Actions" menu.






https://2vrav11opa.execute-api.ap-south-1.amazonaws.com/dev/demo
{
'message' :'hello Everyone'
}


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
History
New
Import
GET https://2vrav11opa.execu
https://2vrav11opa.execute-api.ap-south-1.amazonaws.com/dev/demo
Add to collection
Today
GET https://2vrav11opa.execute-api.ap-south-1.amazo
GET https://2vrav11opa.execute-api.ap-south-1.amazo
July 24
GET https://reqres.in/api/users?page=1
GET https://reqres.in/api/users
GET https://reqres.in/api/users
GET https://reqres.in/api/users?page=1
GET https://reqres.in/api/users?page=1
GET https://uourgh6gj0.execute-api.ap-south-1.amazo
GET https://uourgh6gj0.execute-api.ap-south-1.amazo
July 19
GET https://6lmhy5xbpg.execute-api.ap-south-1.amaz
GET https://6lmhy5xbpg.execute-api.ap-south-1.amaz
GET https://6lmhy5xbpg.execute-api.ap-south-1.amaz
GET https://6lmhy5xbpg.execute-api.ap-south-1.amaz
GET https://6lmhy5xbpg.execute-api.ap-south-1.amaz
July 4
POST https://yy7djplk5k.execute-api.ap-south-1.amazor
GET https://yy7djplk5k.execute-api.ap-south-1.amazor
GET https://6lmhy5xbpg.execute-api.ap-south-1.amazonaws.com/dev/demo
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Bulk Edit
Body
Cookies
Headers (6)
Test Results
Status: 200 OK
Time: 530 ms
Size: 255 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
'message' :'hello Everyone'

4. AWS Service Integration:

- Integration Type: AWS Service

STEP 1: Select your API in the API Gateway console.

STEP 2: Choose the desired resource and method for which you want to set up the AWS service integration.

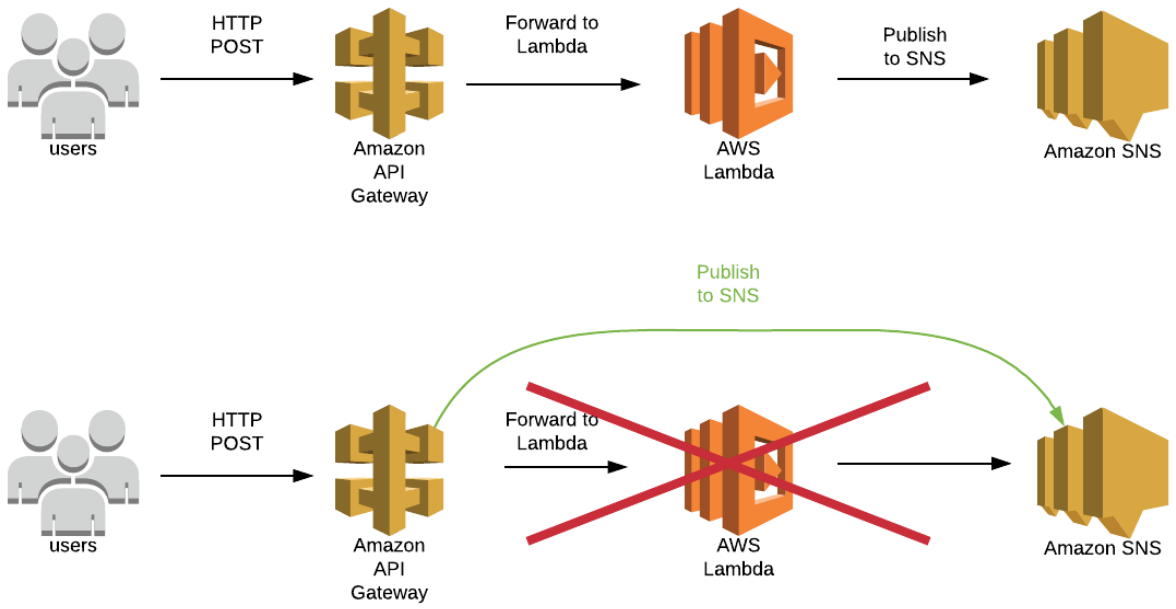
STEP 3: Under the "Integration" section, choose "AWS Service" as the integration type.

STEP 4: Select the AWS service you want to integrate with from the dropdown list.

STEP 5: Configure any necessary input parameters or options for the AWS service.

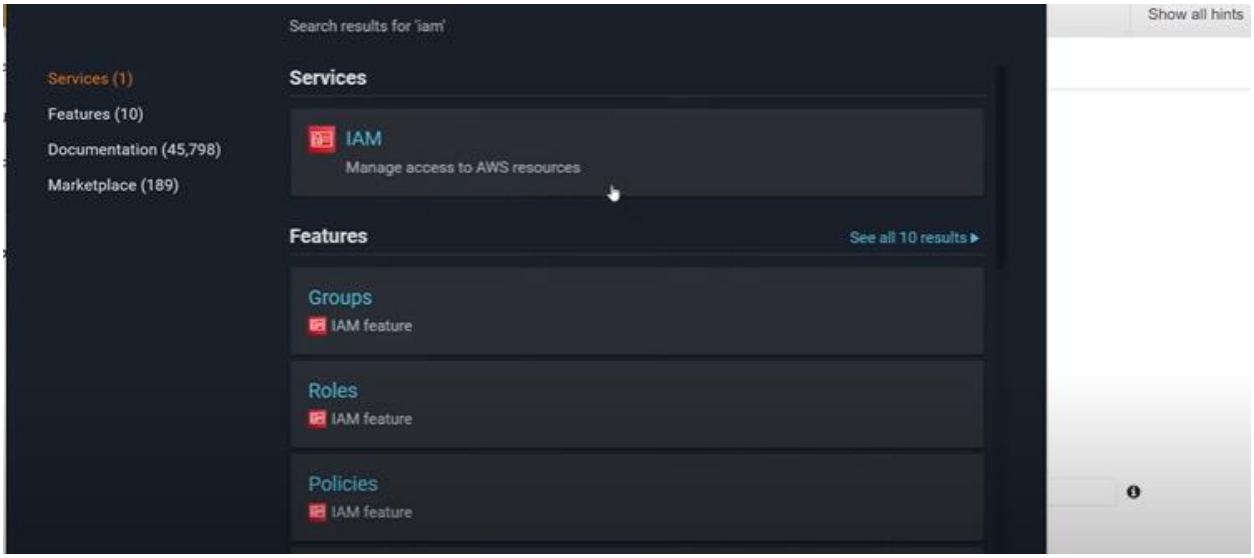
STEP 6: Test the integration using the "Test" button.

STEP 7: Deploy the API to a stage to make it accessible via an endpoint URL.



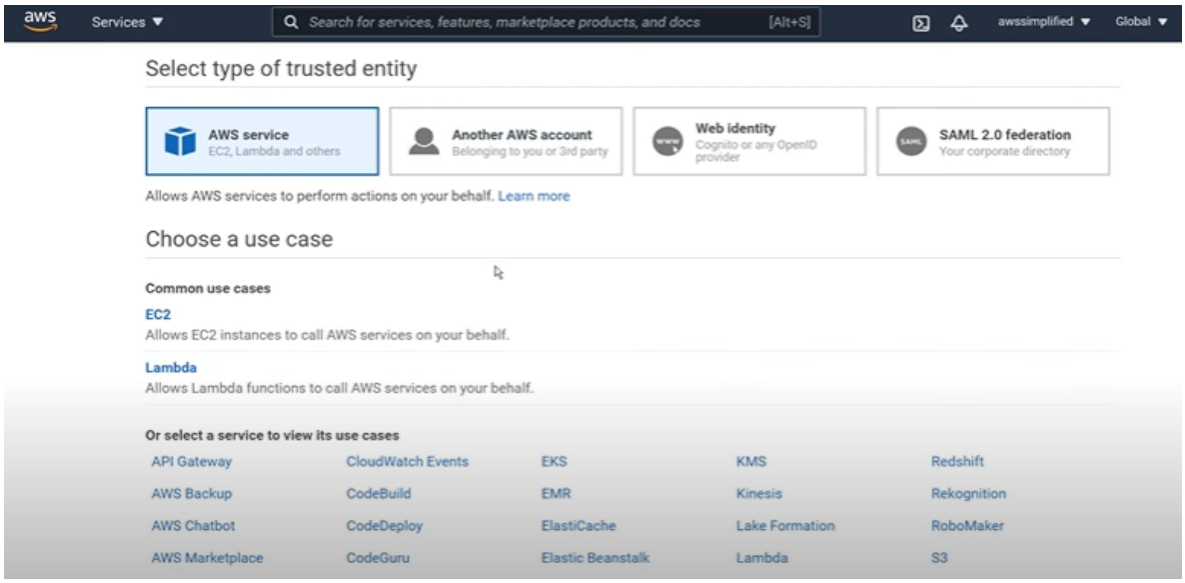
DIRECT INTEGRATION - API GATEWAY & DYNAMO DB

STEP 1: In the AWS Management Console, go to the service.

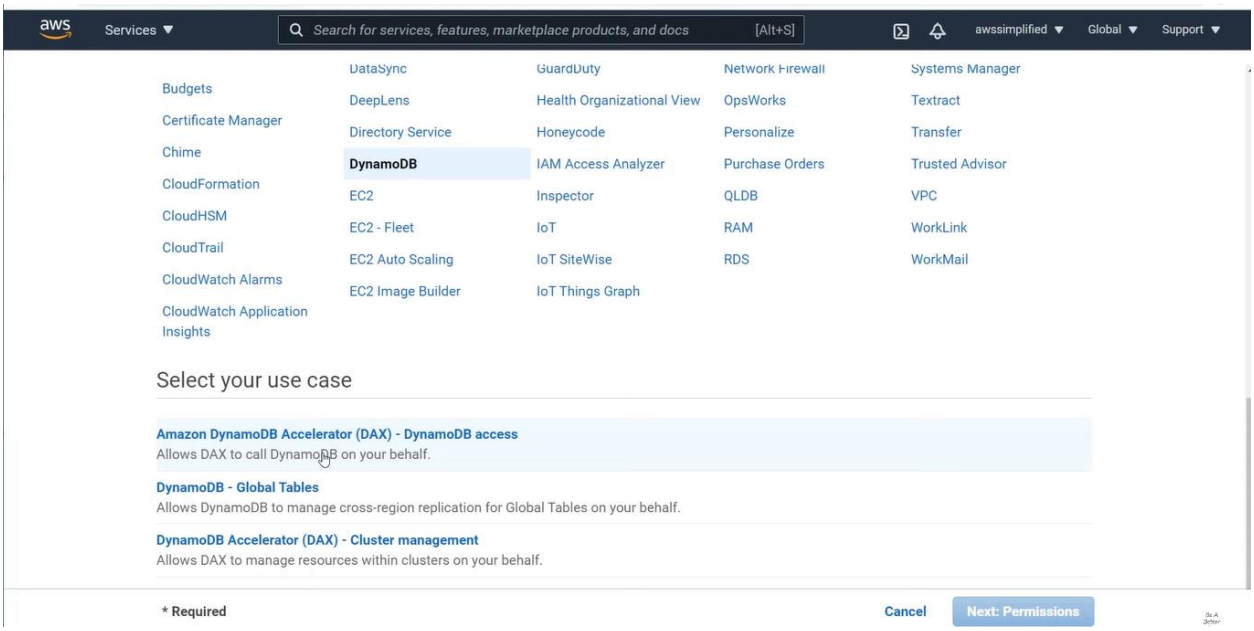


STEP 2: In the IAM Dashboard, click on "Roles" in the left navigation pane. Click the "Create role" button.

STEP 3: In the "Select type of trusted entity" section, select "AWS service" as the trusted entity type.

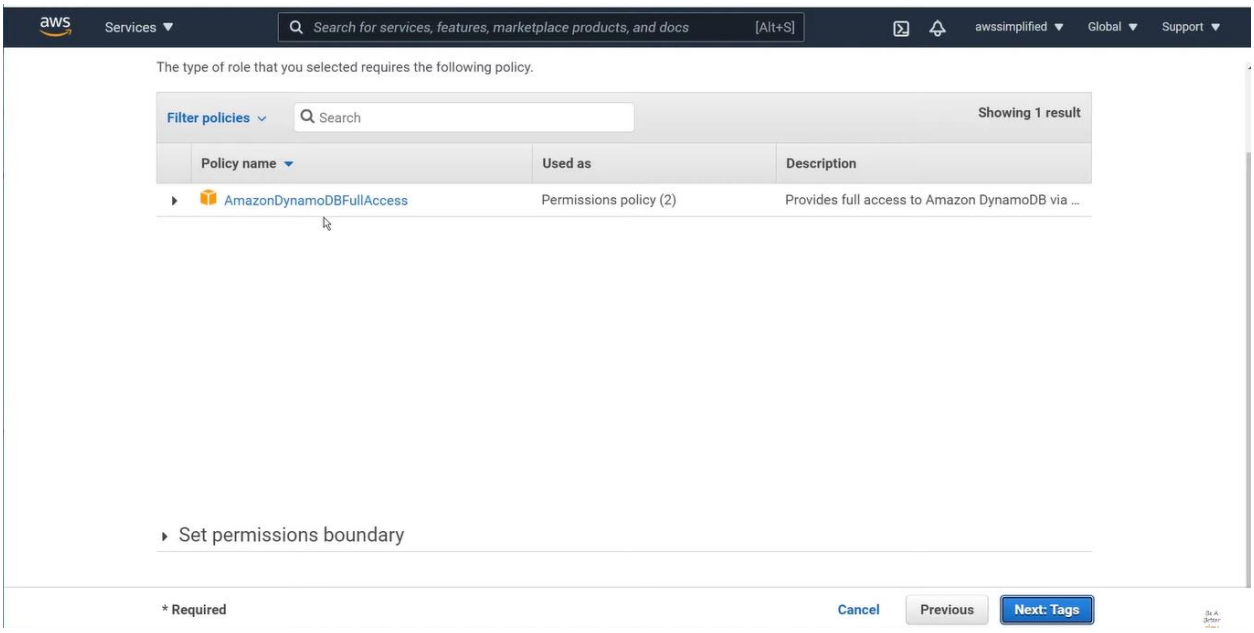


STEP 4: In the "Choose a use case" section, select the "DynamoDB" use case since you want to create a role for DynamoDB Accelerator (DAX).

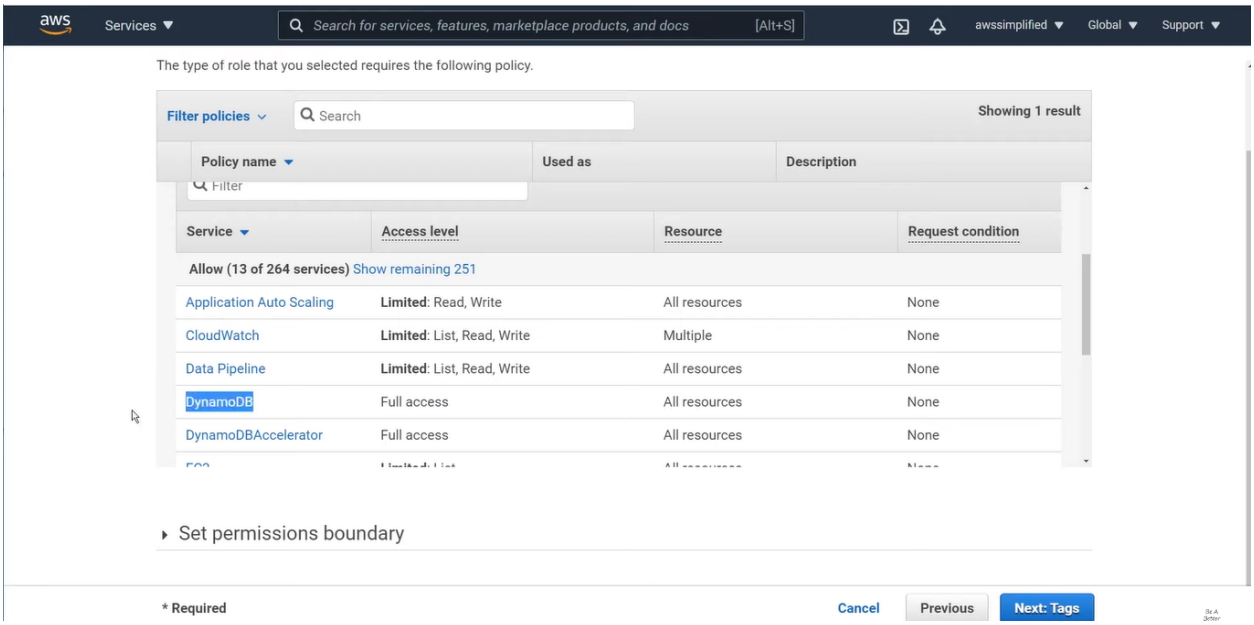


STEP 5: In the "Permissions" section, you need to attach policies that grant full access to DynamoDB, to do this:

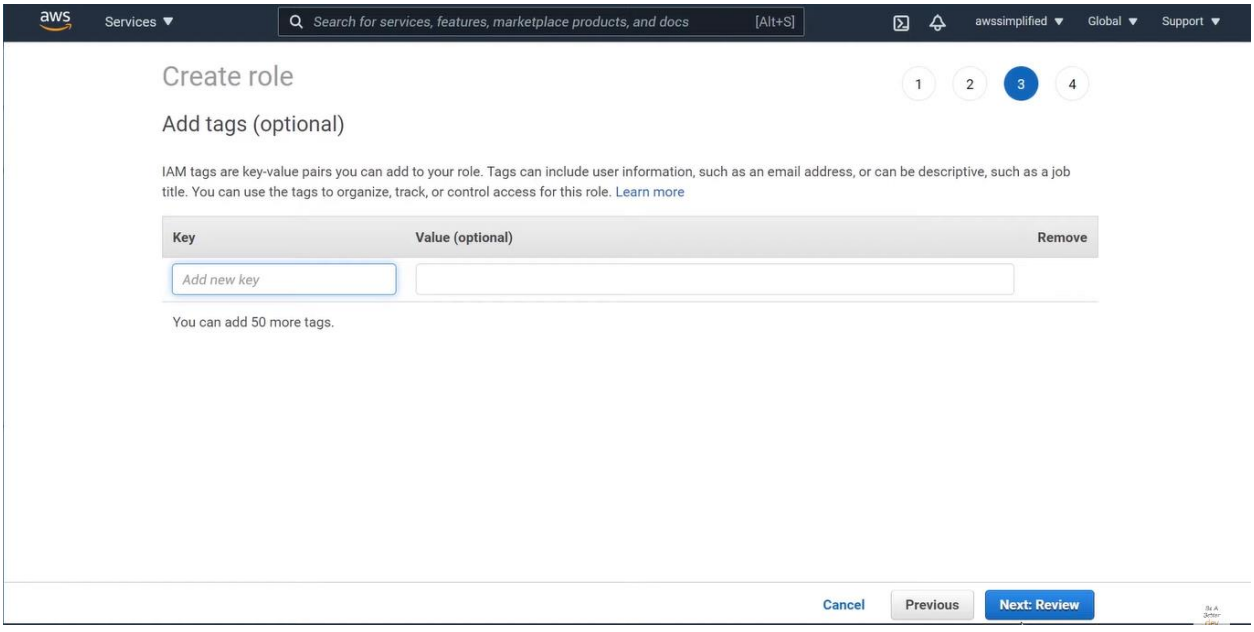
- Click the "Next: Permissions" button.
- Search for and select the "AmazonDynamoDBFullAccess" policy to grant full access to DynamoDB.



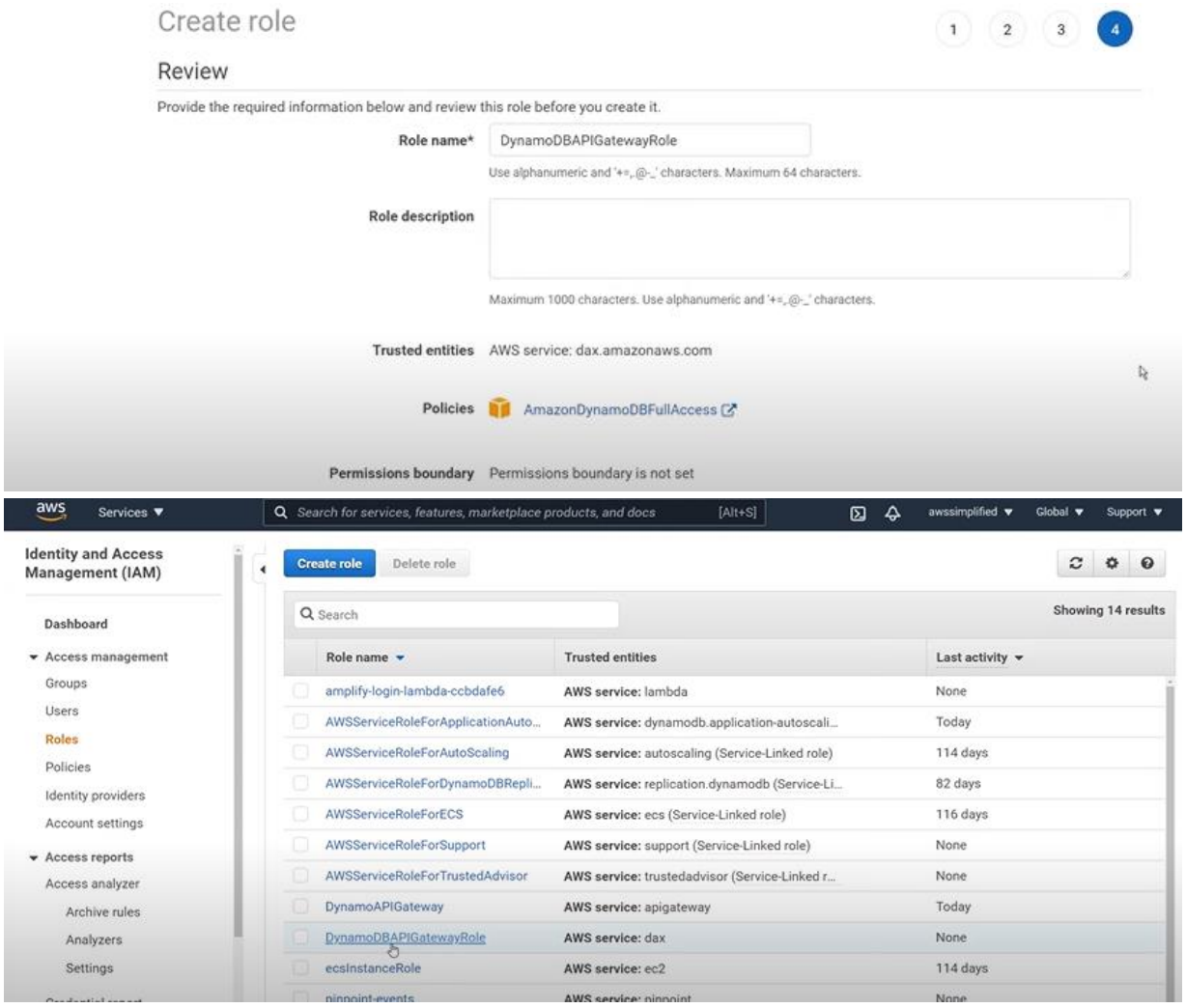
STEP 6: Review the role configuration to ensure it grants the desired permissions for both DynamoDB.

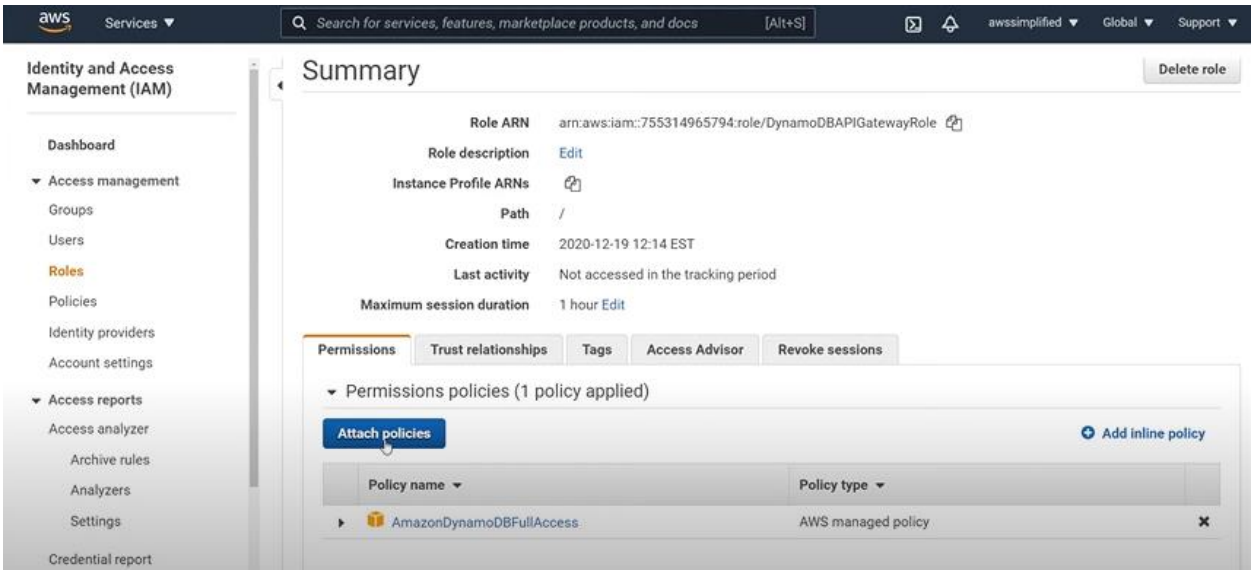


STEP 7: You can optionally add tags to the role for better organization and tracking



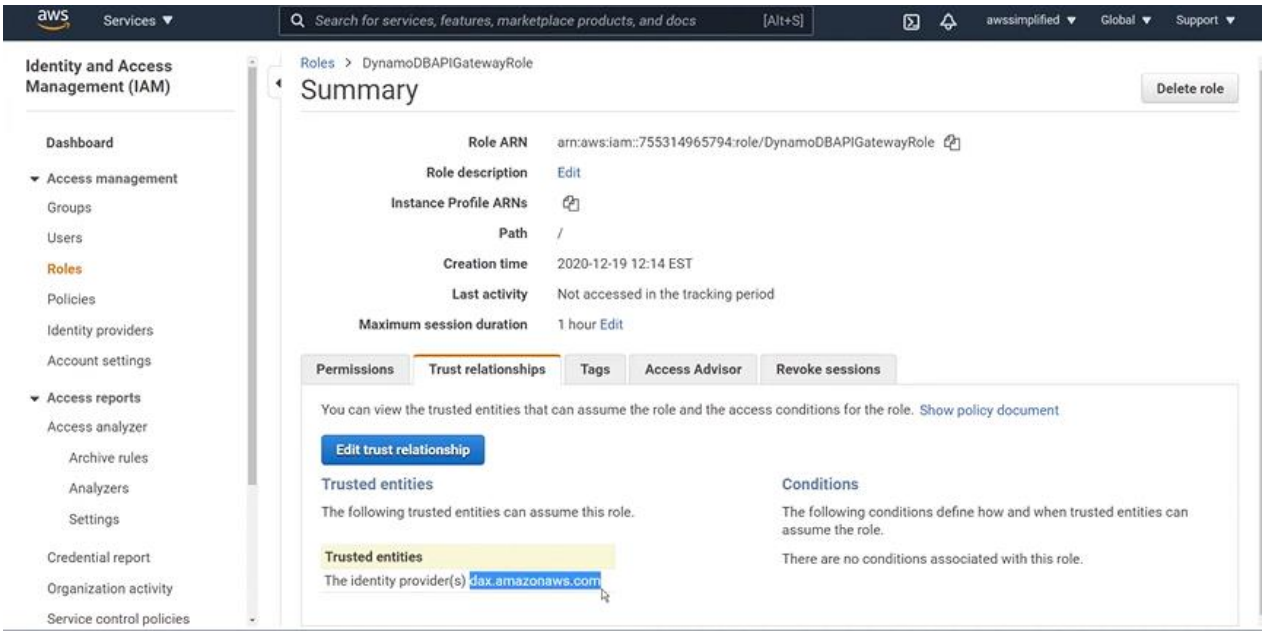
STEP 8: Provide a name for your role (e.g., "DynamoDBAPIGatewayRole")



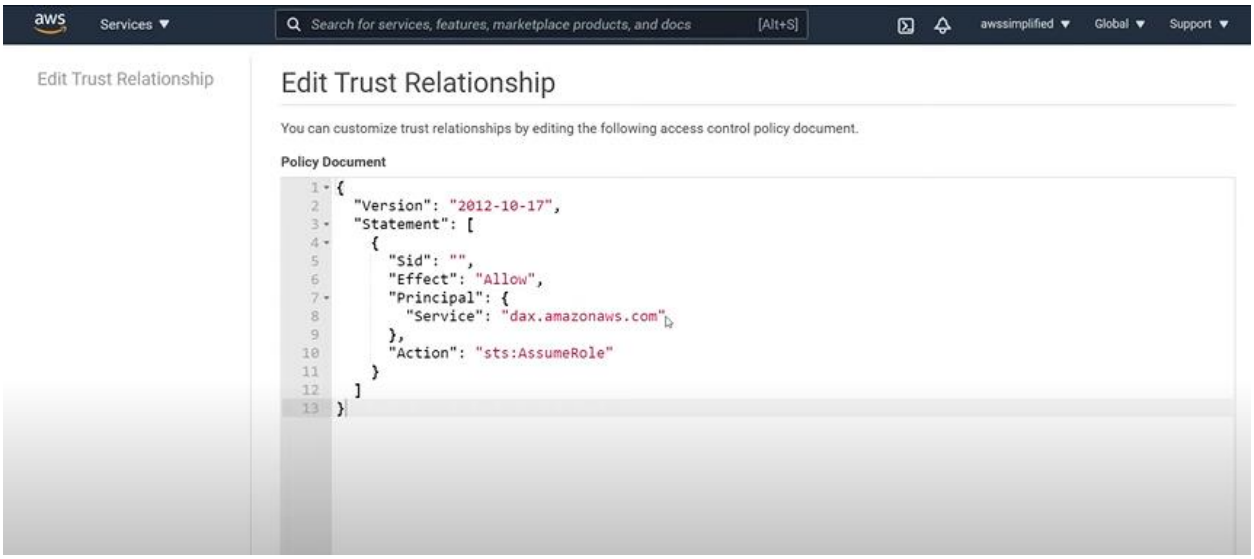


NOTE : Change Trust Relationship for API Gateway-Now that you've created the role with DAX permissions, you can update the trust relationship to allow API Gateway to assume this role. Here's how to do it:

STEP 9: In the "Summary" tab of the role, click on the "Edit trust relationship" button.



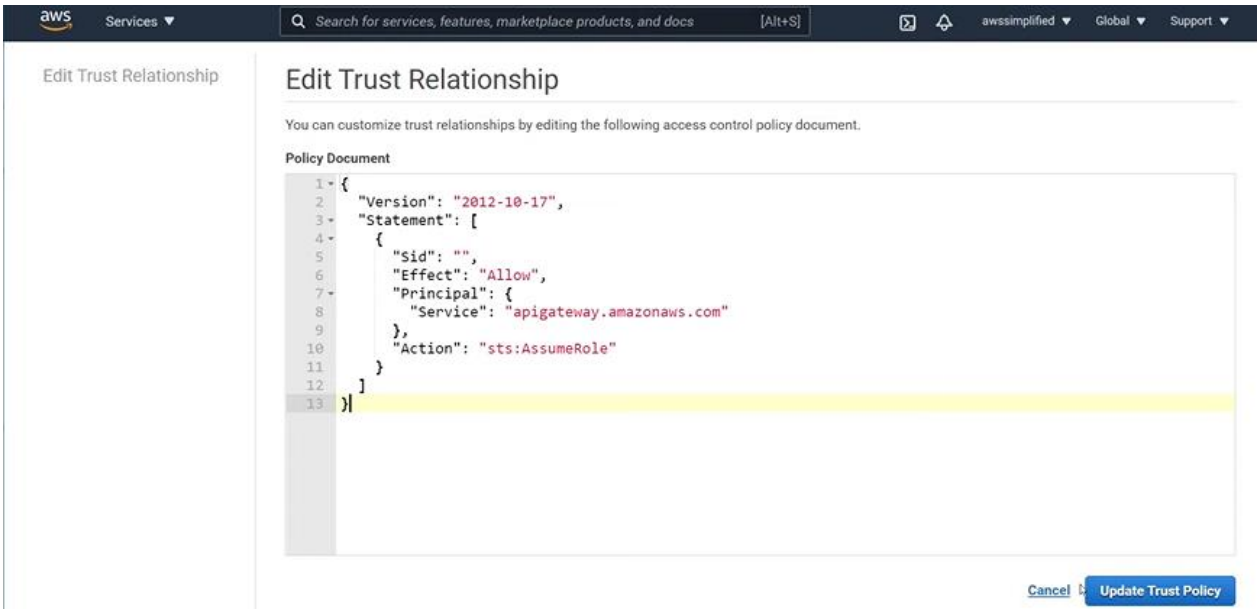
In the "Trust relationships" tab, you'll see a policy document.



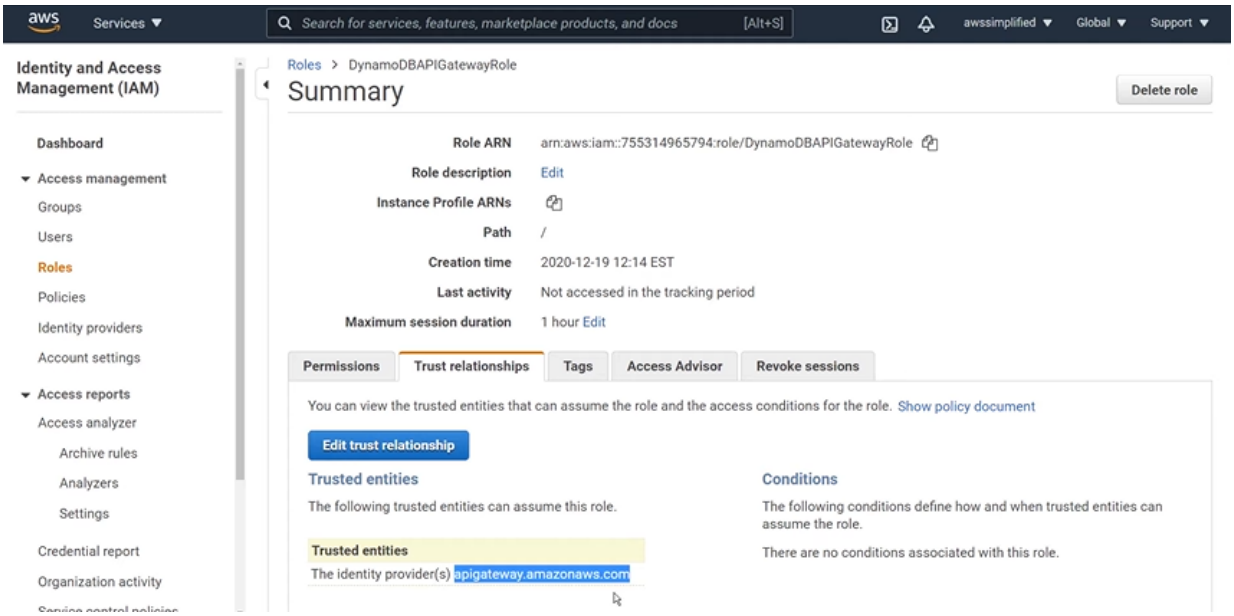
STEP 10: Replace the policy document with the following JSON policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
     {
        "Effect": "Allow",
        "Principal": {
          "Service": "apigateway.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
      }
    ]
  }
```

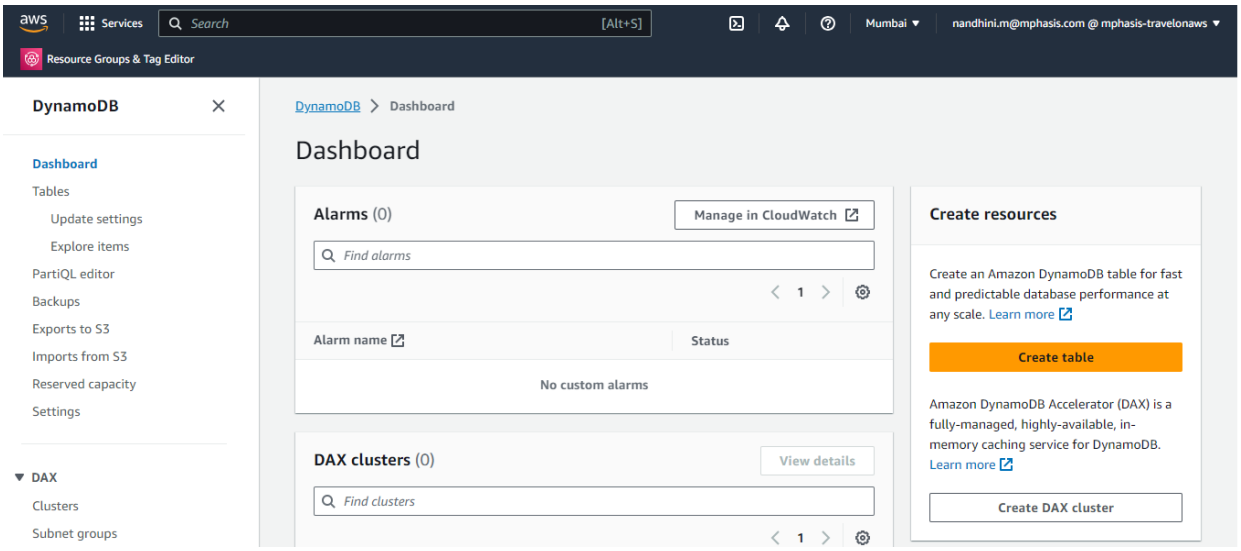
STEP 11: After editing the trust policy, click the "Update Trust Policy" button to save the changes.



Now, the IAM role you created can be assumed by API Gateway.



STEP 12: In the DynamoDB Dashboard, click on "Create table."



STEP 13: Configure the Table

a. Table Settings:

- Enter a unique table name.

- Define the primary key for your table. A primary key consists of one or both of the following:

  - Partition Key (required): A single attribute that uniquely identifies an item.

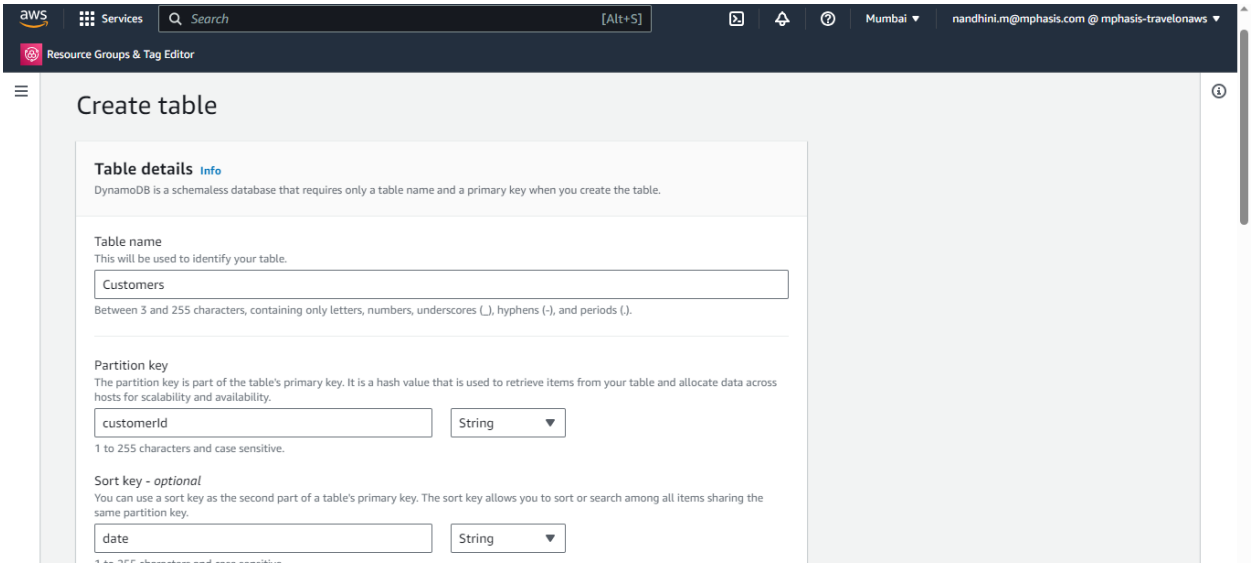
  - Sort Key (optional): If specified, it's used to organize items with the same partition key.

b. Provisioned or On-Demand Capacity Mode:

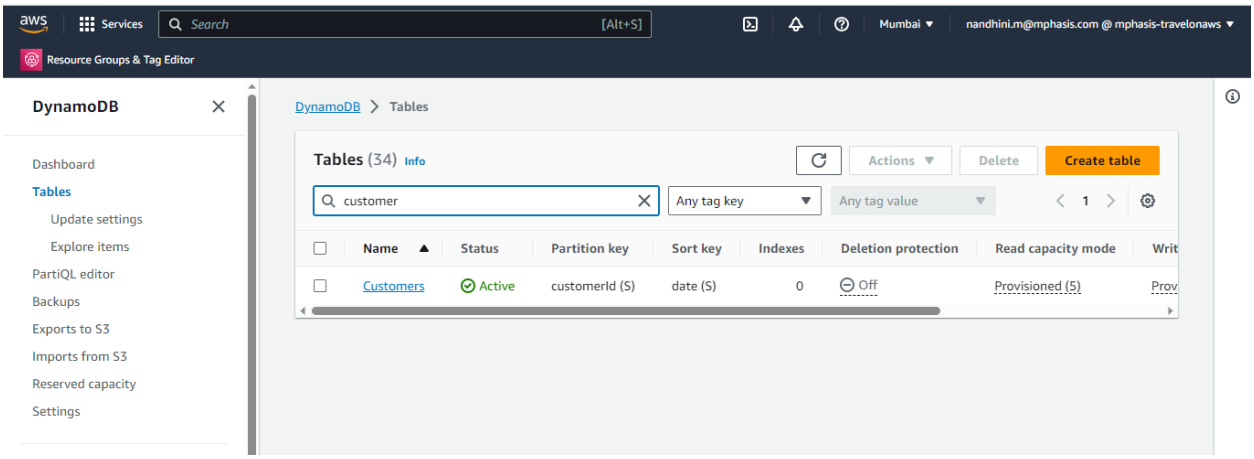
- Choose either "Provisioned" or "On-Demand" capacity mode. Provisioned allows you to specify read and write capacity units, while On-Demand handles capacity automatically.

c. Additional Settings (Optional):

- You can configure options like encryption, stream settings, and TTL (Time to Live) for your table as needed.

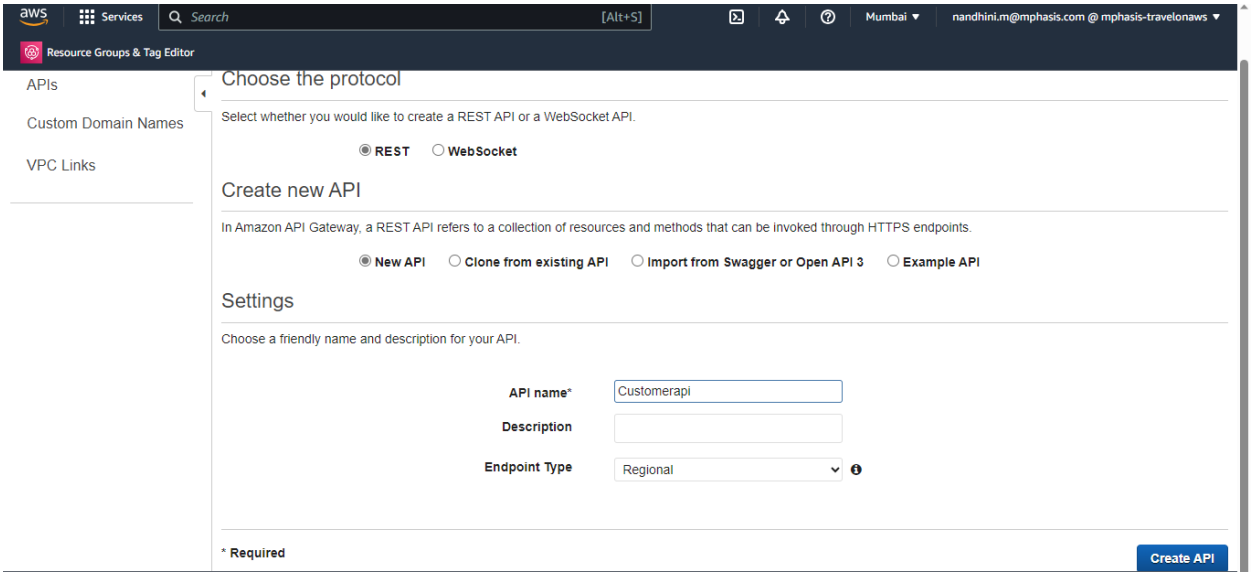


STEP 14: Click the "Create" button to create your DynamoDB table, DynamoDB will create the table, which may take a few moments. You can monitor the status in the DynamoDB console.



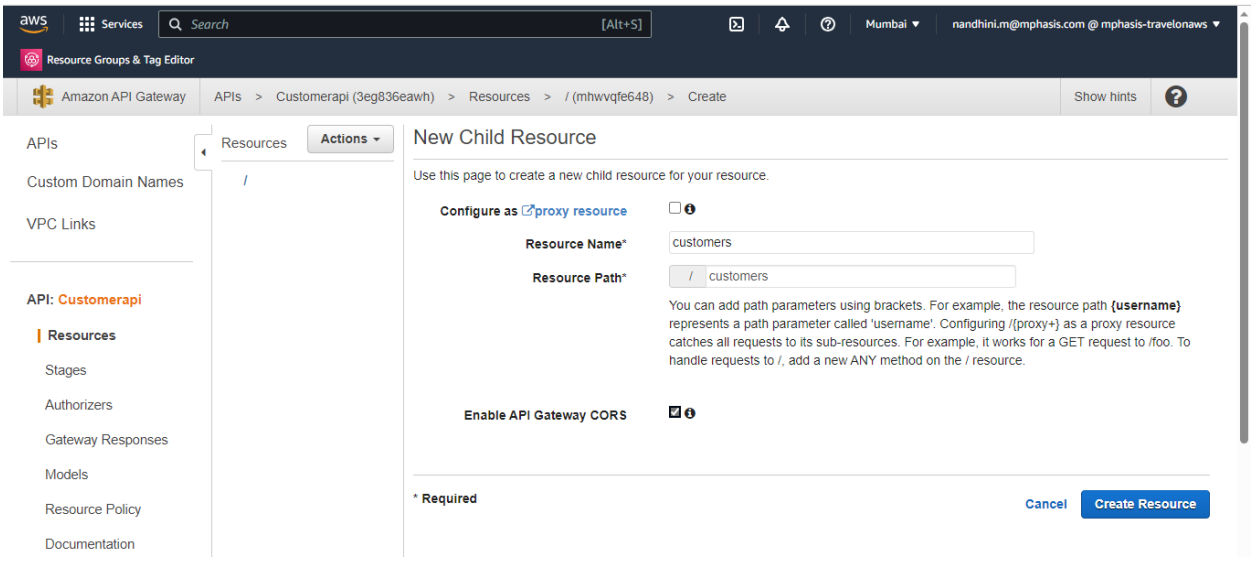
STEP 15: Create a REST API

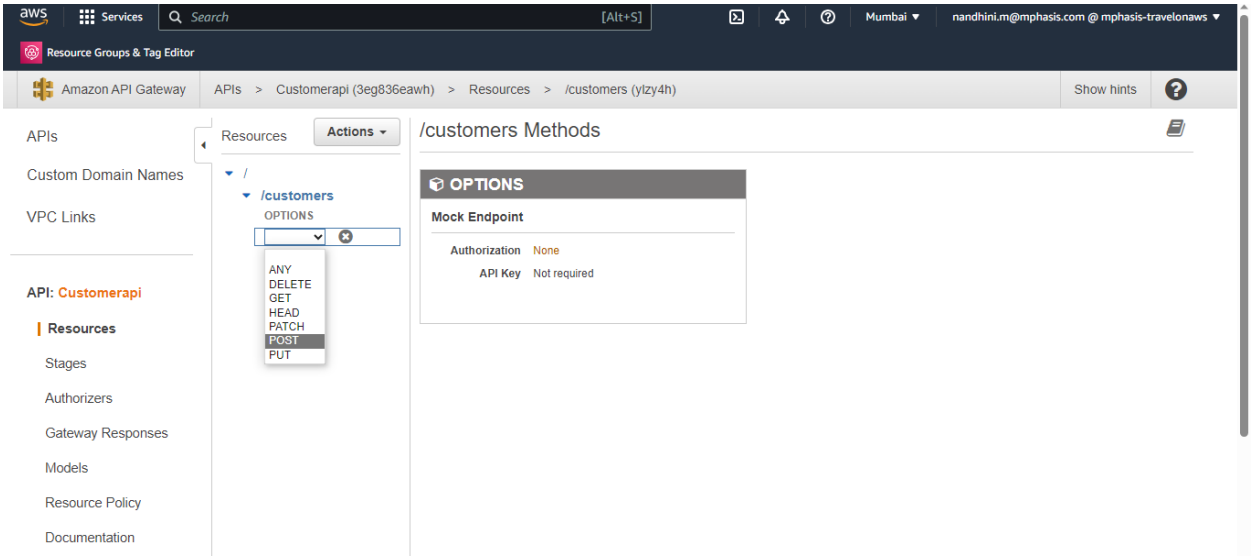
- Click "Create API."
- Choose "HTTP API" or "REST API," depending on your preference and requirements.



STEP 16: Create a Resource and Method

- In your API Gateway configuration, create a new resource (e.g., `/dynamodb`) and an HTTP method (e.g., POST, GET) for your DynamoDB operations.





STEP 17: Set Up AWS Service Integration

- In the "AWS Service" settings:

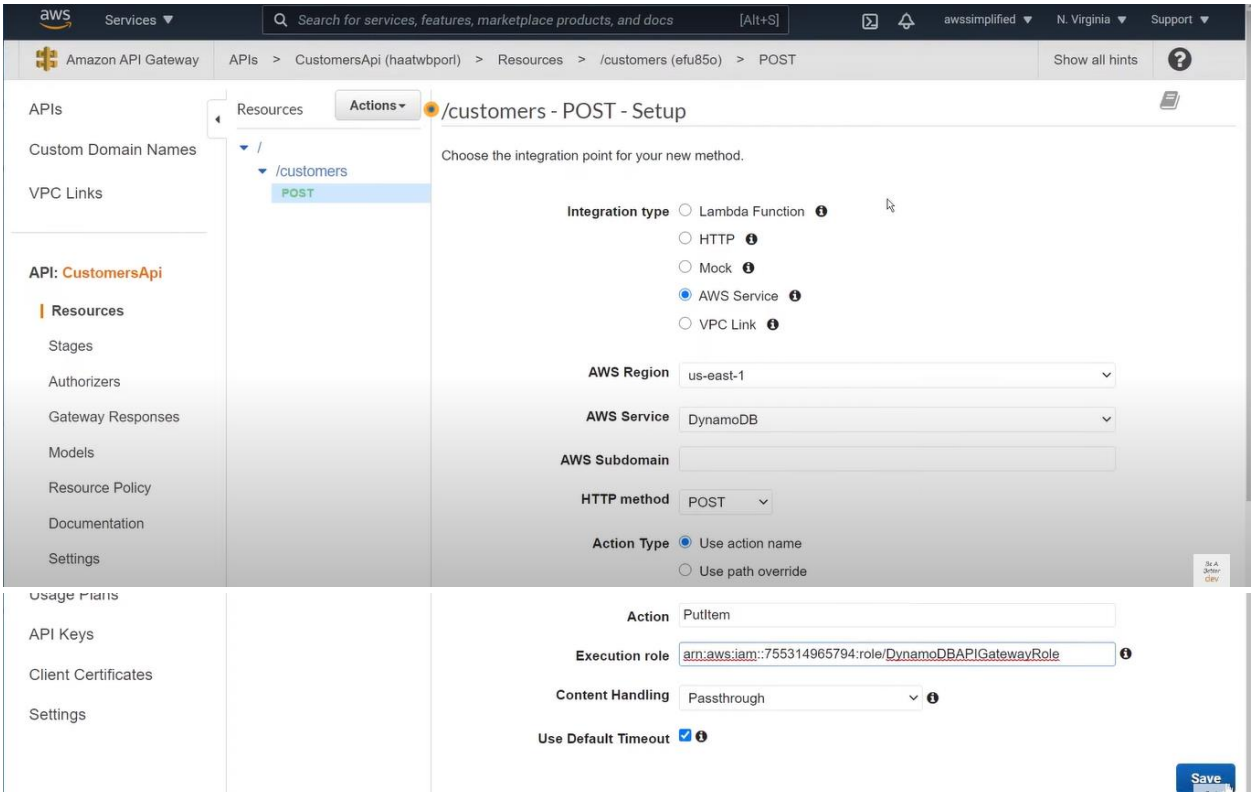
- AWS Region: Choose the AWS region where your DynamoDB table is located.

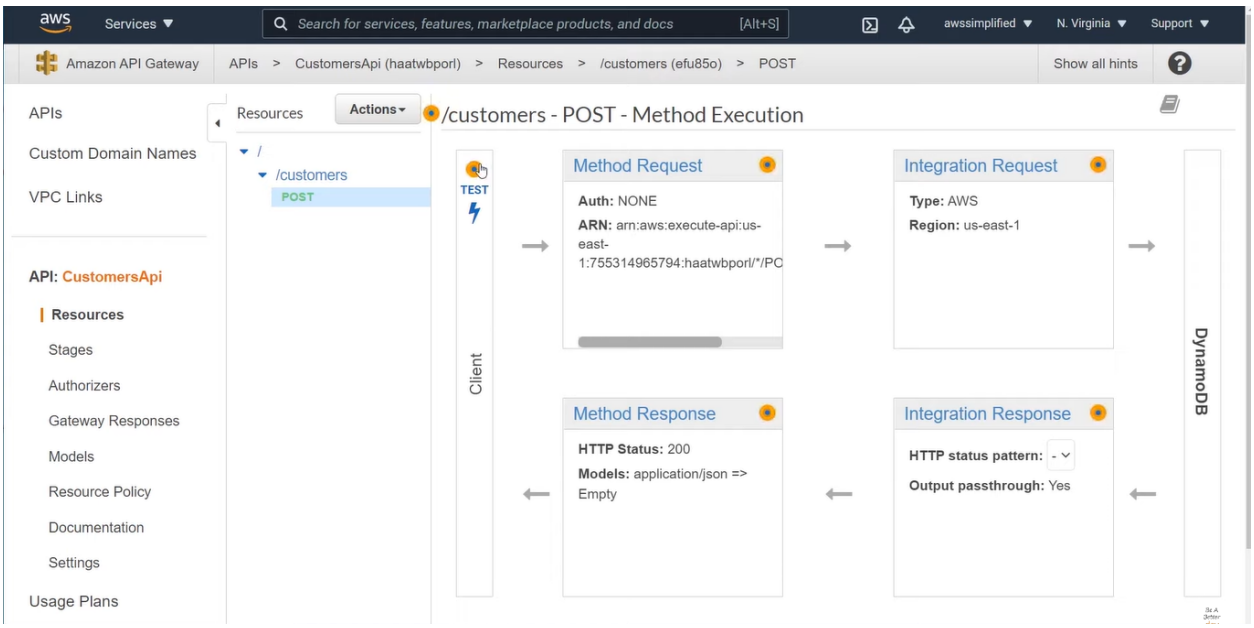
- AWS Service: Select "DynamoDB."

- HTTP Method: Choose the appropriate method for your DynamoDB operation (e.g., "POST" for PutItem, "GET" for GetItem).

- Action: Choose the DynamoDB action you want to perform (e.g., "PutItem," "GetItem").

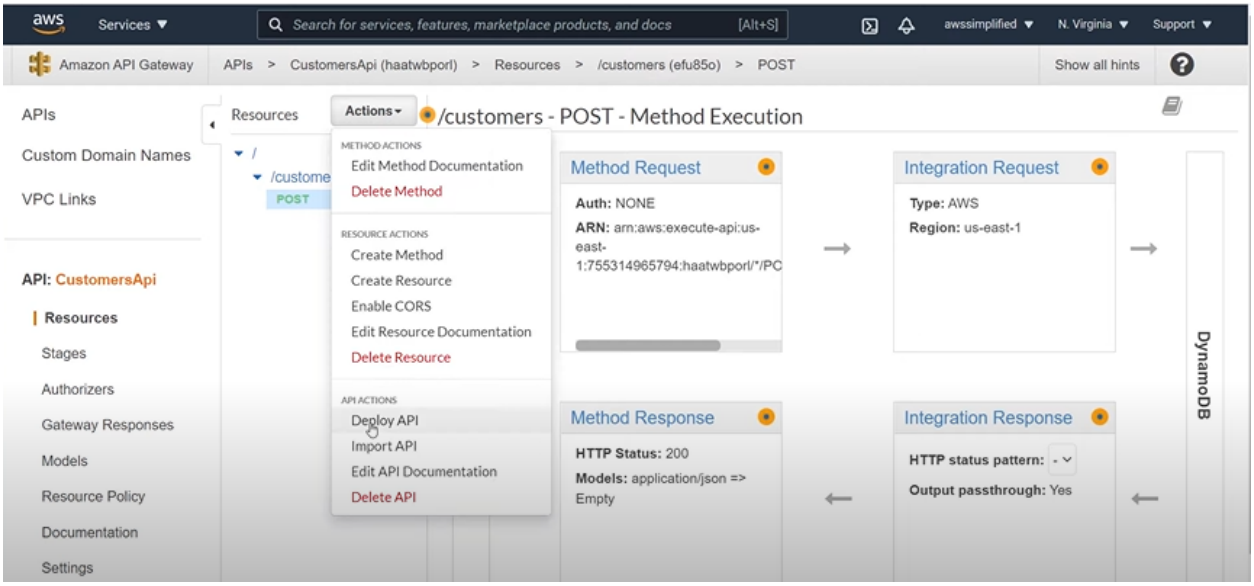
- Execution Role: Create or select an IAM Role that grants permissions to access DynamoDB. This role should have policies that allow the necessary actions on your DynamoDB table.

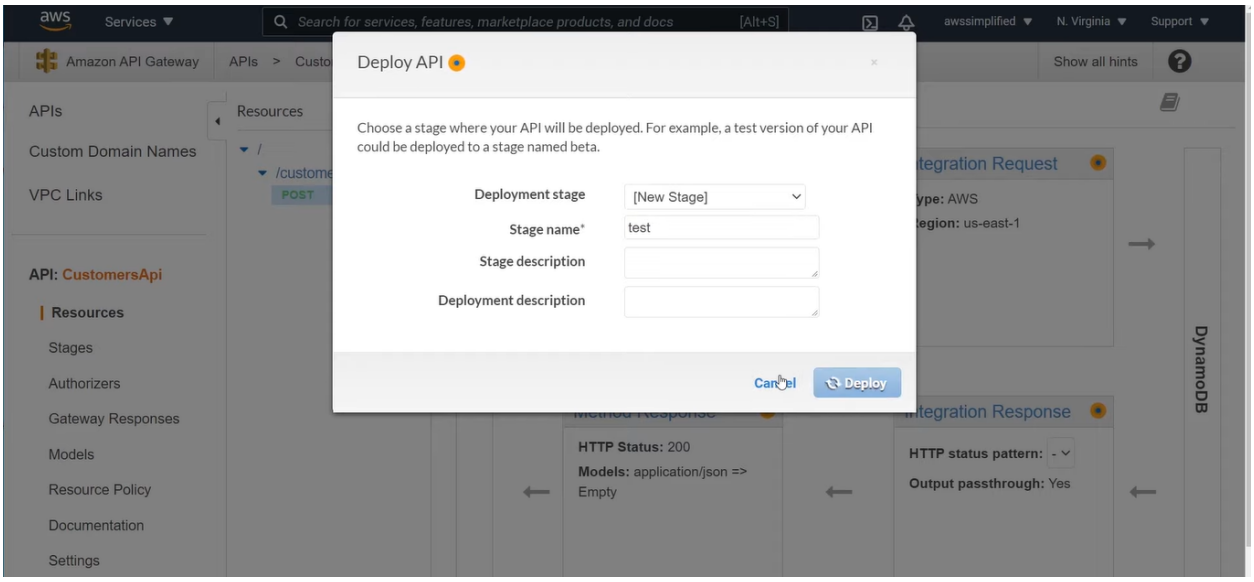

Services
Search for services, features, marketplace products, and docs [Alt+S]
awssimplified
N. Virginia
Support
Amazon API Gateway
APIs > CustomersApi (haatwbporl) > Resources > /customers (efu85o) > POST
Show all hints
APIs
Custom Domain Names
VPC Links
API: CustomersApi
Resources
Stages
Authorizers
Gateway Responses
Models
Resource Policy
Documentation
Settings
Usage Plans
API Keys
Client Certificates
Settings
Resources
Actions
/customers - POST - Setup
Choose the integration point for your new method.
Integration type
Lambda Function
HTTP
Mock
AWS Service
VPC Link
AWS Region
us-east-1
AWS Service
DynamoDB
AWS Subdomain
HTTP method
POST
Action Type
Use action name
Use path override
Action
PutItem
Execution role
arn:aws:iam::755314965794:role/DynamoDBAPIGatewayRole
Content Handling
Passthrough
Use Default Timeout
Save

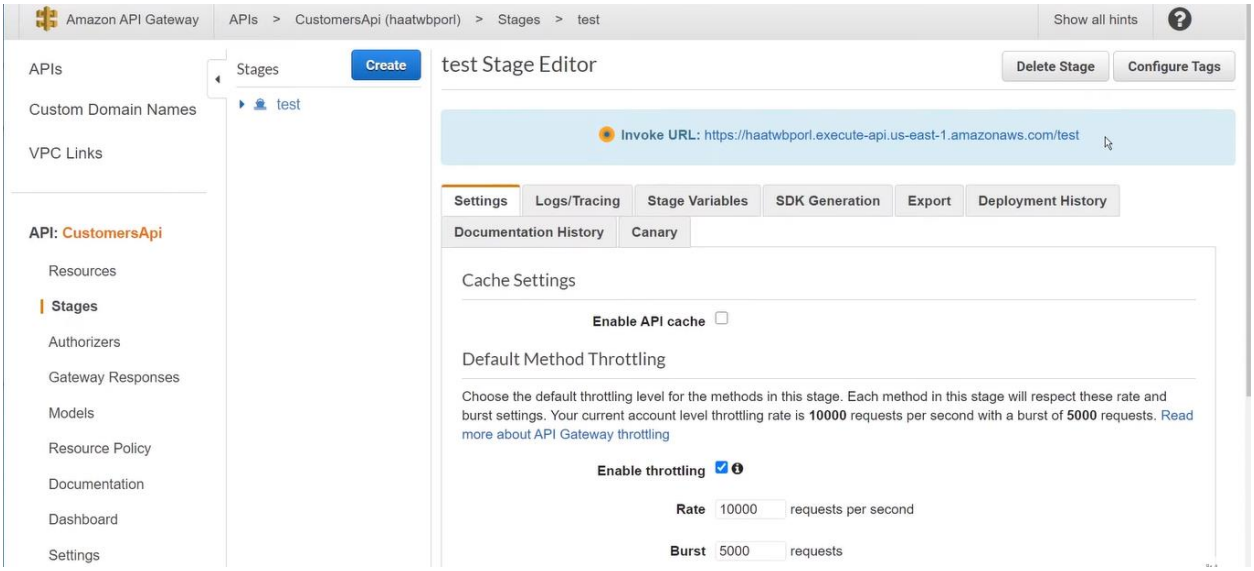

Services
Search for services, features, marketplace products, and docs [Alt+S]
awssimplified
N. Virginia
Support
Amazon API Gateway
APIs > CustomersApi (haatwbporl) > Resources > /customers (efu85o) > POST
Show all hints
APIs
Custom Domain Names
VPC Links
API: CustomersApi
Resources
Stages
Authorizers
Gateway Responses
Models
Resource Policy
Documentation
Settings
Usage Plans
Resources
Actions
/customers - POST - Method Execution
TEST
Client
Method Request
Auth: NONE
ARN: arn:aws:execute-api:us-east-1:755314965794:haatwbporl/*/PO
Integration Request
Type: AWS
Region: us-east-1
DynamoDB
Method Response
HTTP Status: 200
Models: application/json => Empty
Integration Response
HTTP status pattern: -
Output passthrough: Yes

STEP 18: Deploy Your API

- Click "Actions" and choose "Deploy API."
- Select a deployment stage (e.g., "prod," "test") and click "Deploy."
- Note the URL of your API endpoint.

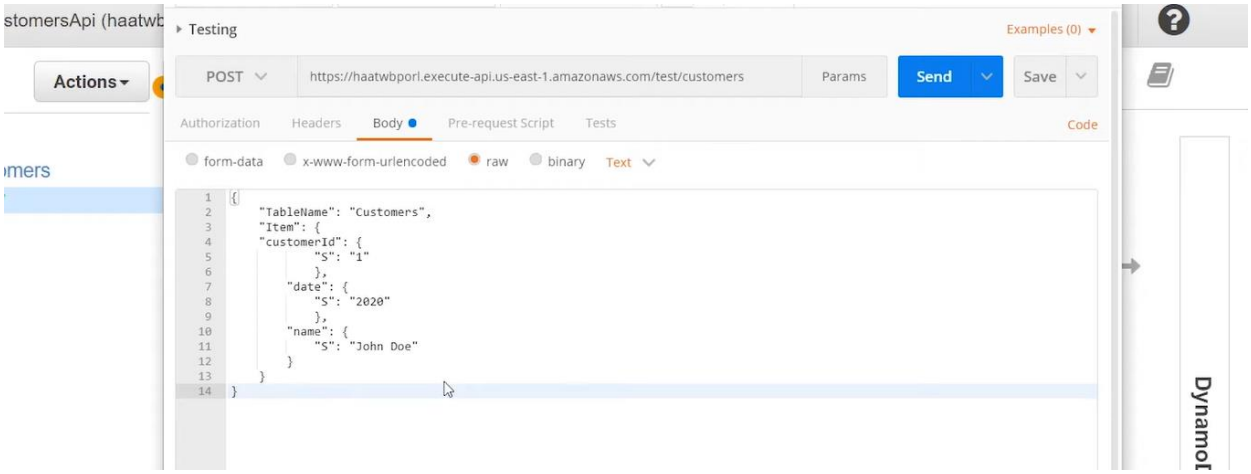




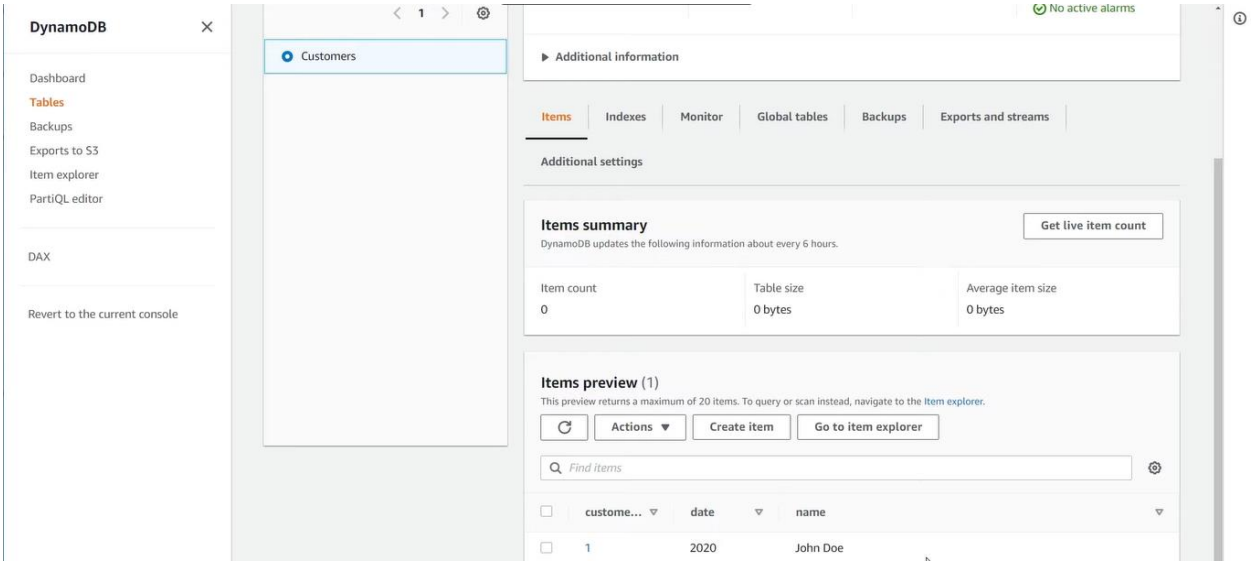


STEP 19: Test Your Integration

- Use tools like Postman, or a web browser to send HTTP requests to the API Gateway endpoint.

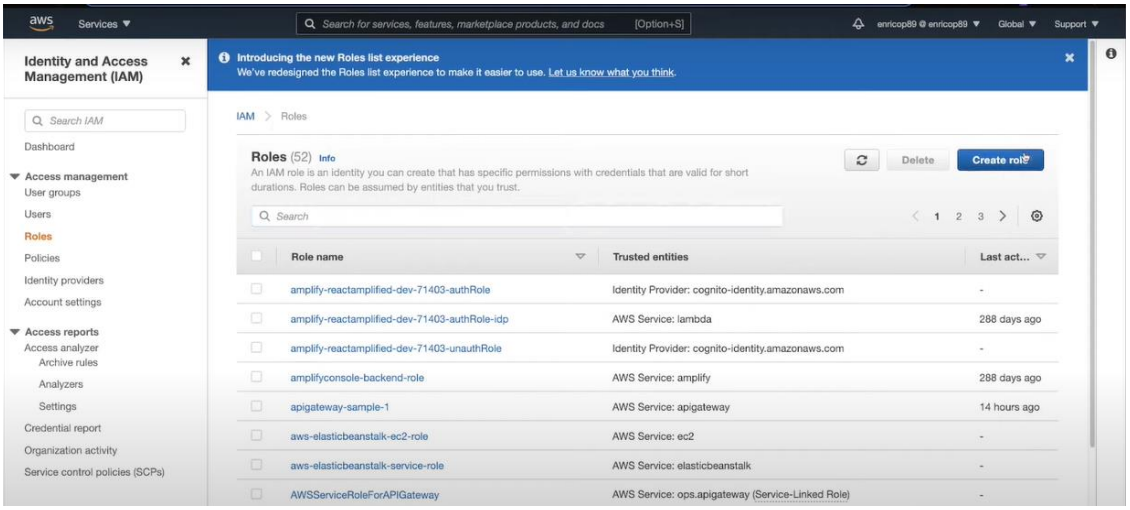


STEP 20: Verify that requests are forwarded to DynamoDB, and responses are returned correctly.

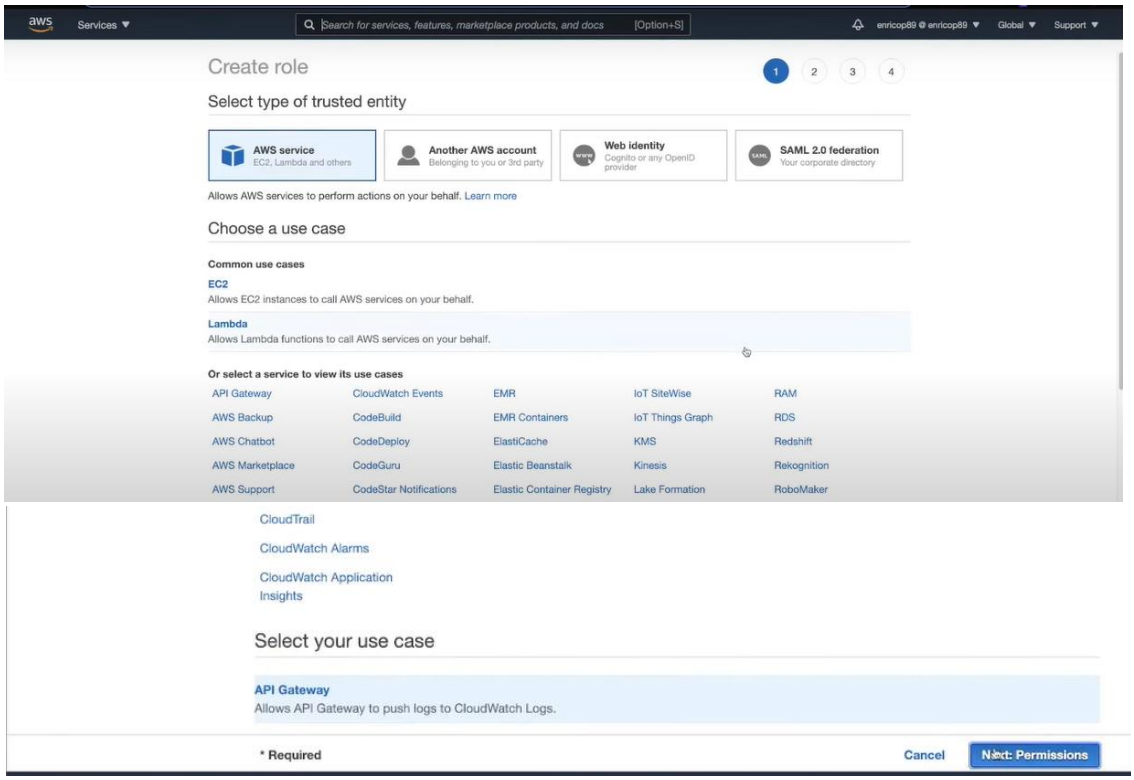


DIRECT INTEGRATION – API GATEWAY & S3

STEP 1: In the IAM dashboard, click on "Roles" in the left-hand navigation pane. Click the "Create role" button.

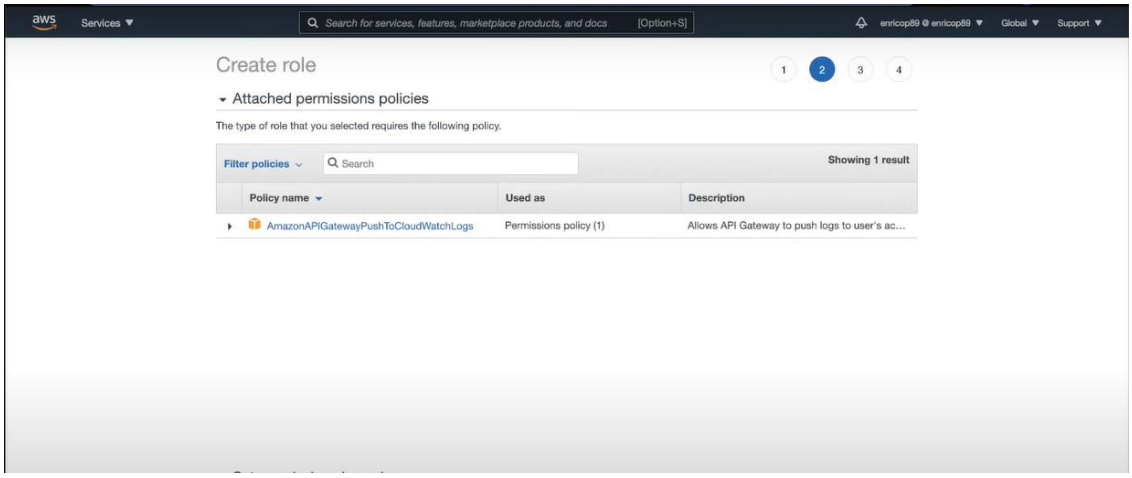


STEP 2: In the "Select type of trusted entity" section, choose "AWS service". In the "Choose a use case" section, select "API Gateway". Click "Next: Permissions"

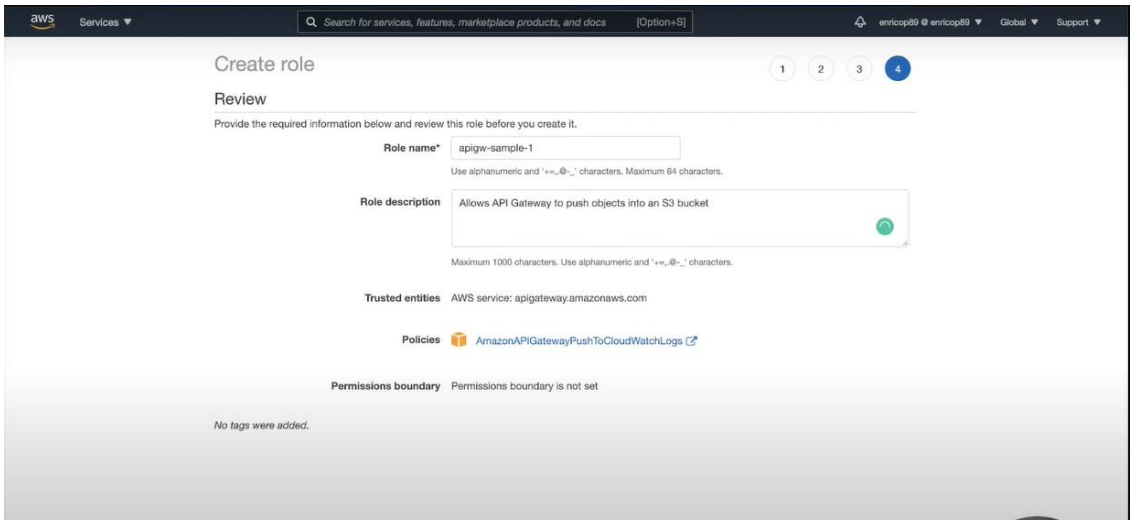


STEP 3: In the "Permissions" step, you can choose an existing policy that grants permissions to push logs to CloudWatch Logs.

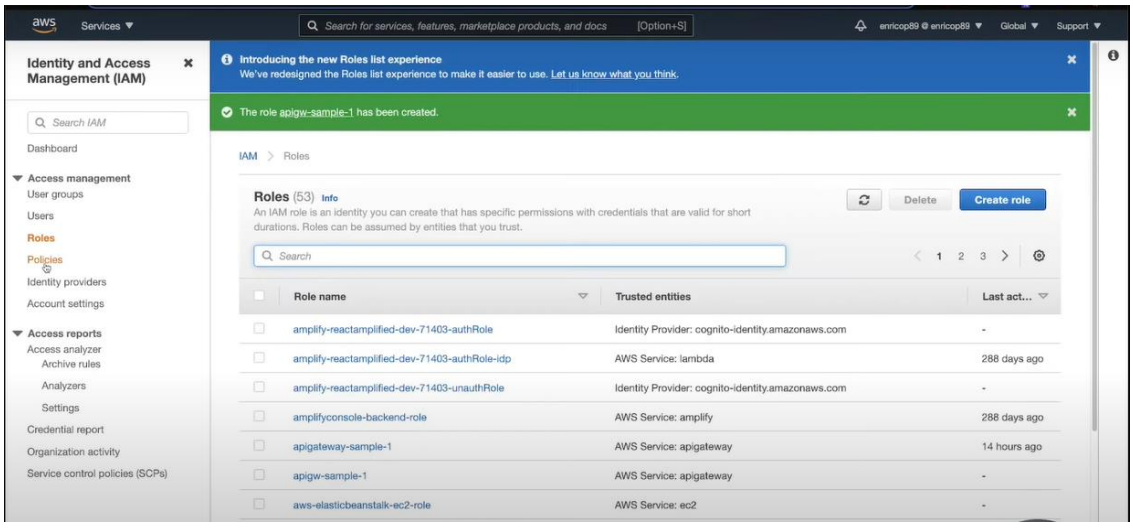
a. Click the "Attach permissions policies" button.
b. In the "Filter policies" search box, type "CloudWatchLogs" to find the built-in policy "CloudWatchLogs_FullAccess." Select this policy if it provides the necessary permissions



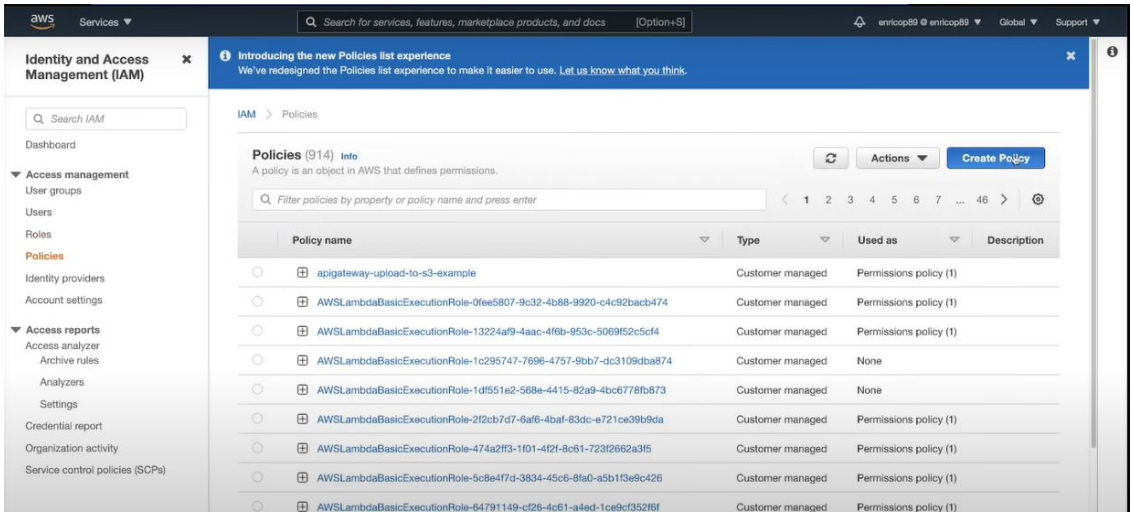
STEP 4 : Give your role a meaningful name and add a description (optional).Click "Next: Tags" if you want to add tags (optional).



Now, you have created an IAM role that allows Amazon API Gateway to push logs to CloudWatch Logs. You can associate this role with your API Gateway stage or deployment to enable logging.



STEP 5: In the IAM dashboard, click on "Policies" in the left-hand navigation pane. Click the "Create policy" button.



STEP 6: In the "Create Policy" wizard, select the "Visual editor" tab or choose the "JSON" tab to create a custom policy.

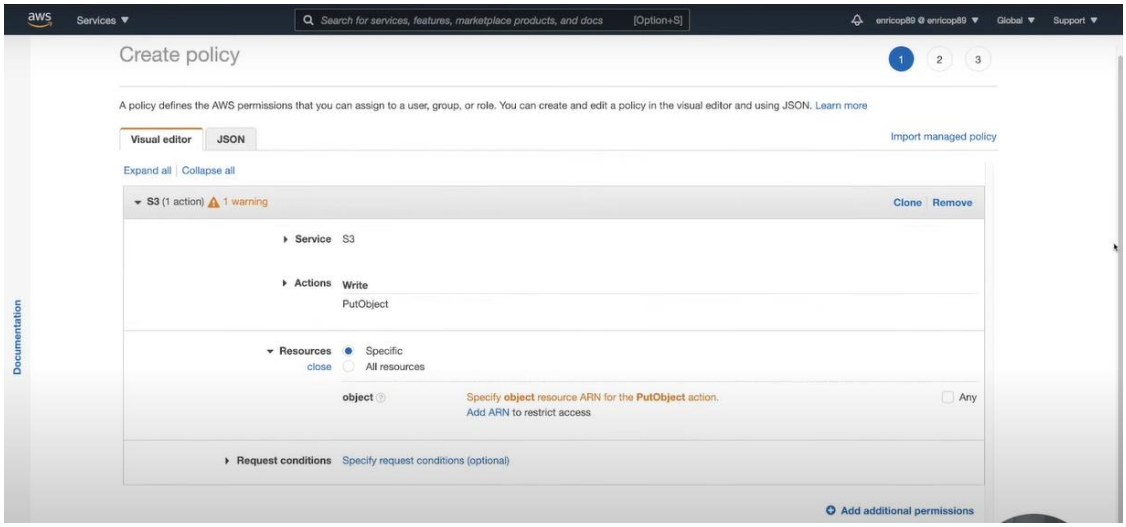
In Visual Editor:

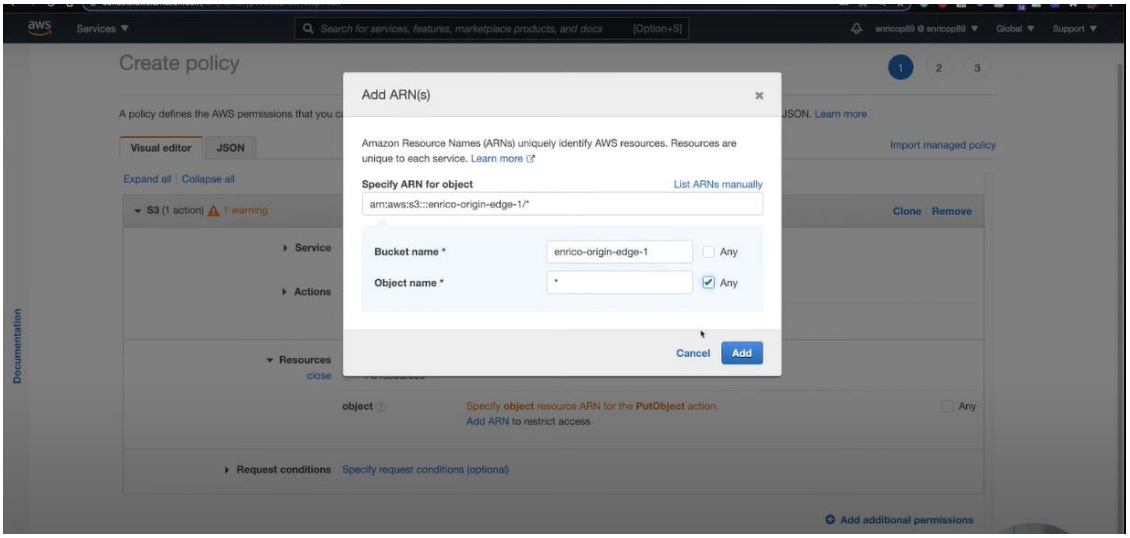
a. Select "S3" from the list of services**.**

b. Select the "PutObject"as an action. Because you want to grant "Write" permissions for "PutObject."

c. you can specify resources by clicking the "Add ARN" button. Enter the ARN for your S3 bucket name and object name.

NOTE: The ARN for an S3 object typically follows this format- **`arn: aws:s3:::bucket-name/object-key`**.



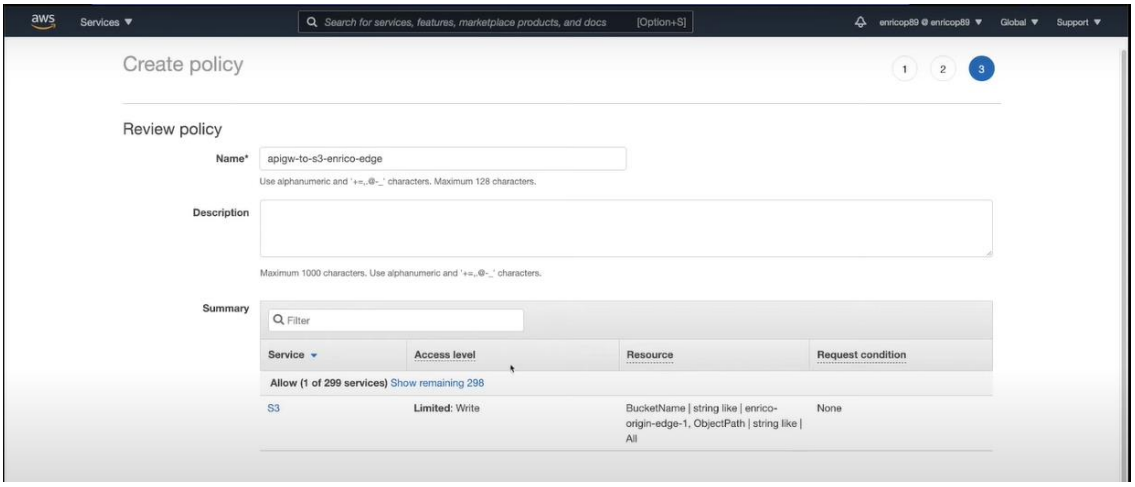


In the JSON editor, define the policy. Below is an example of a policy that allows Amazon API Gateway to perform the "s3: PutObject" action on a specific S3 bucket and object.

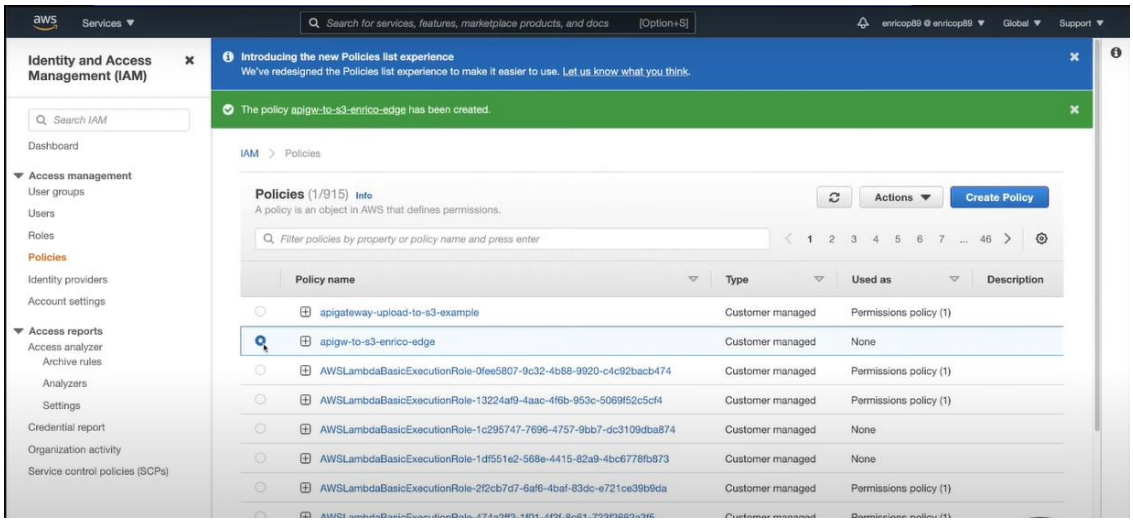
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::YOUR_BUCKET_NAME/YOUR_OBJECT_NAME"
        }
    ]
}
```

STEP 7: Review the permissions you've added in the visual editor to ensure they match your requirements.

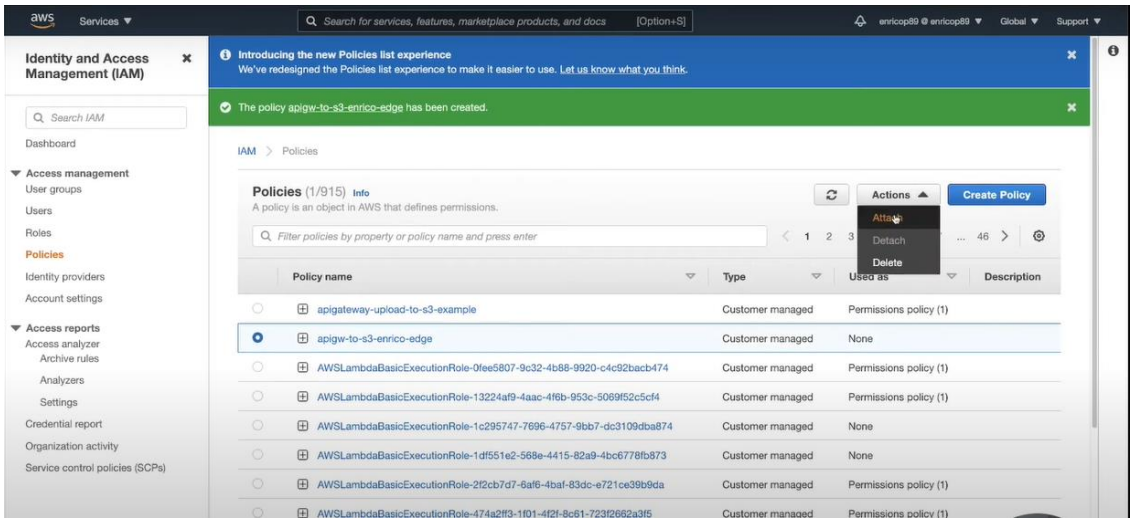
STEP 8: Give your policy a meaningful name and description.



STEP 9: Click the "Create policy" button to create the custom IAM policy.

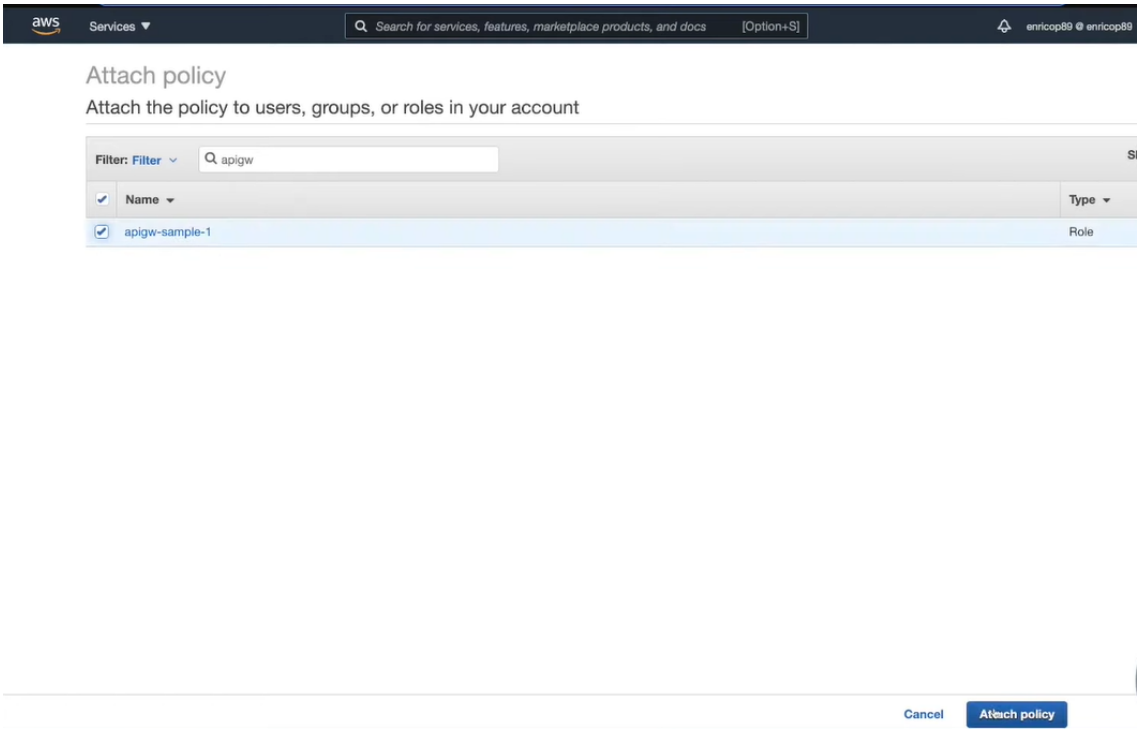


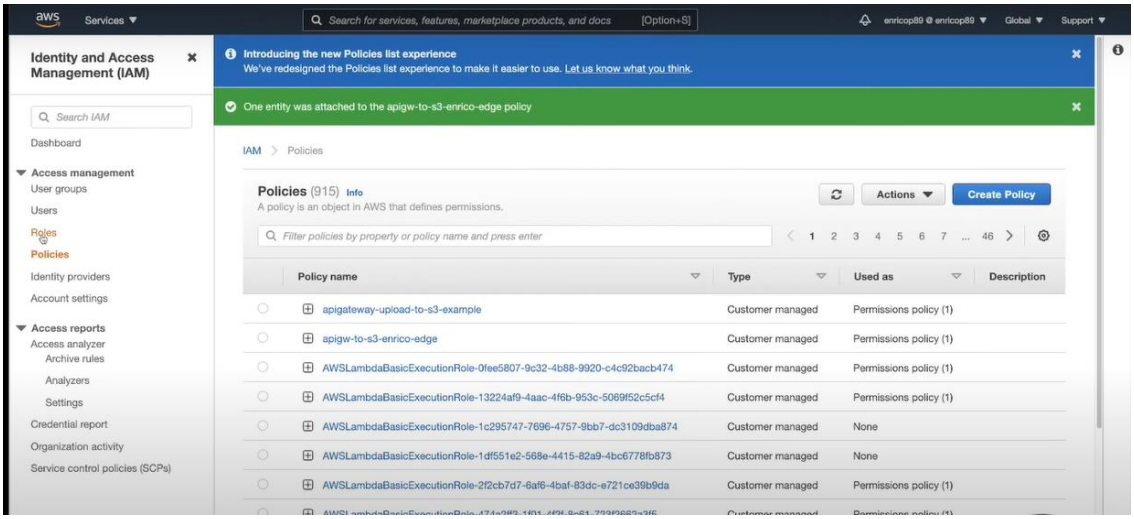
STEP 10: Click on the policy name to select it. After selecting the policy, you'll see an "Actions" menu at the top. From the "Policy actions" menu, choose "Attach."



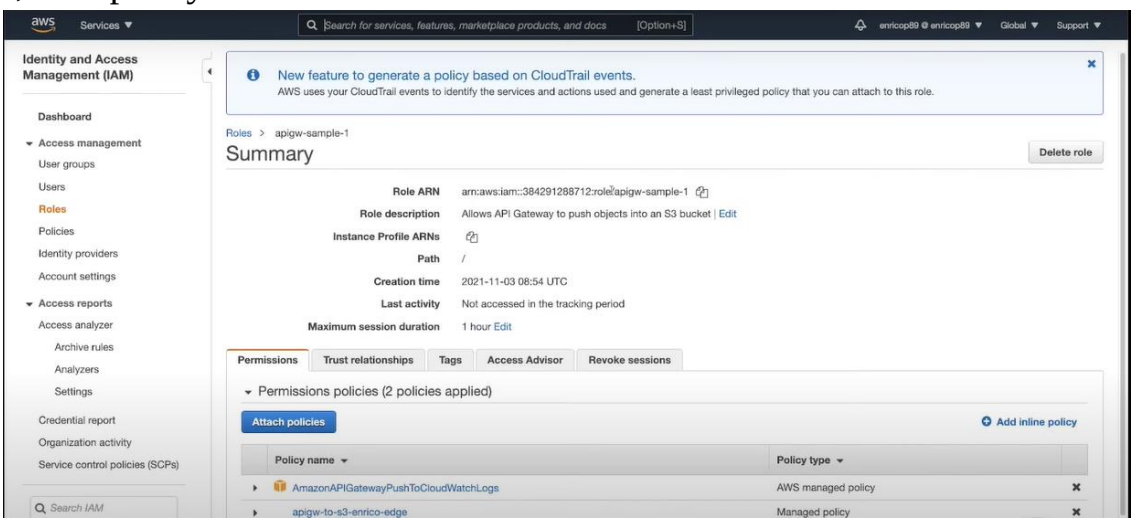
STEP 11: You will be prompted to select the role to which you want to attach the policy.

STEP 12: Click the "Attach policy" button to complete the process.

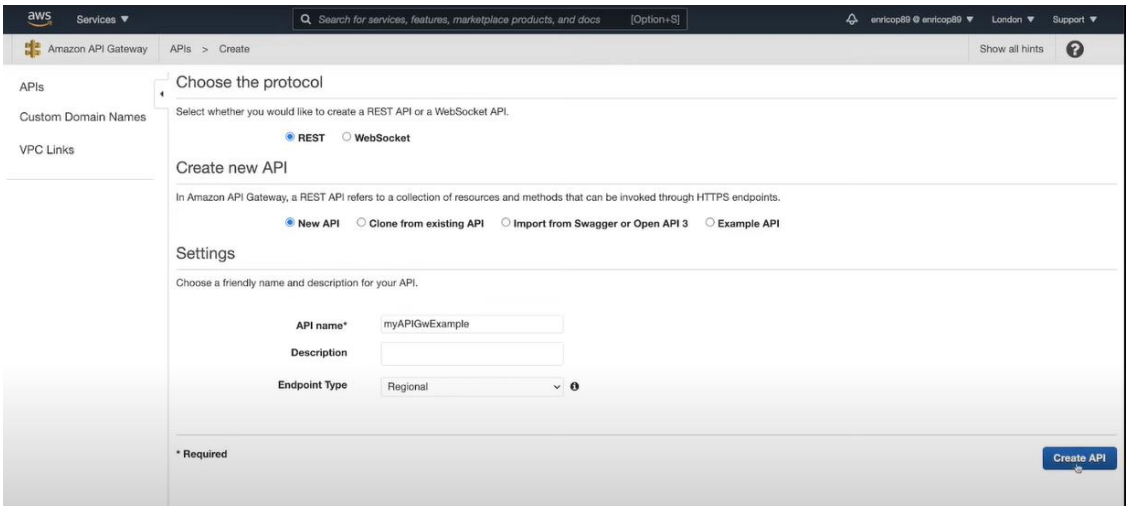




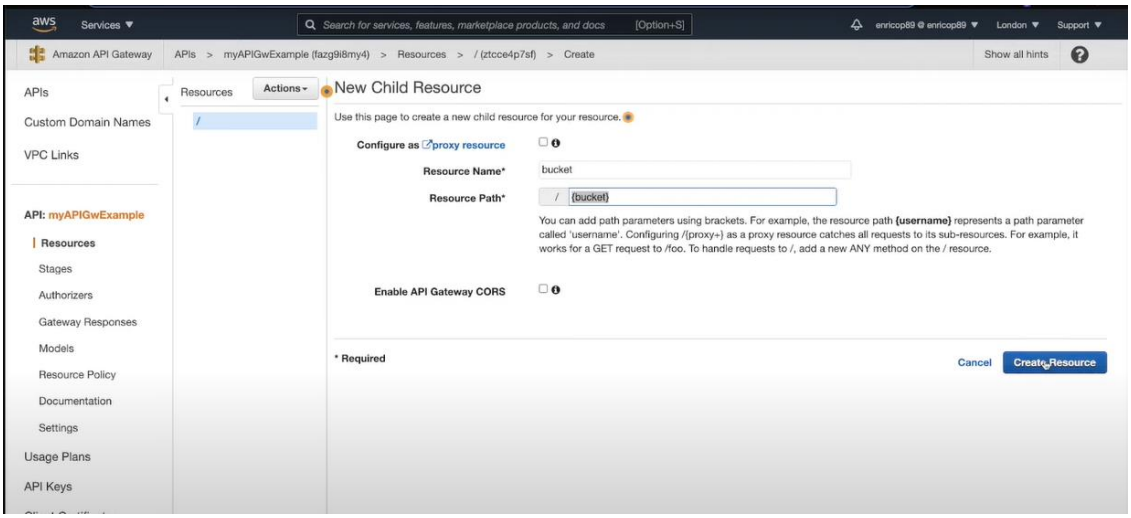
Now, The policy will now be attached to the selected IAM role.

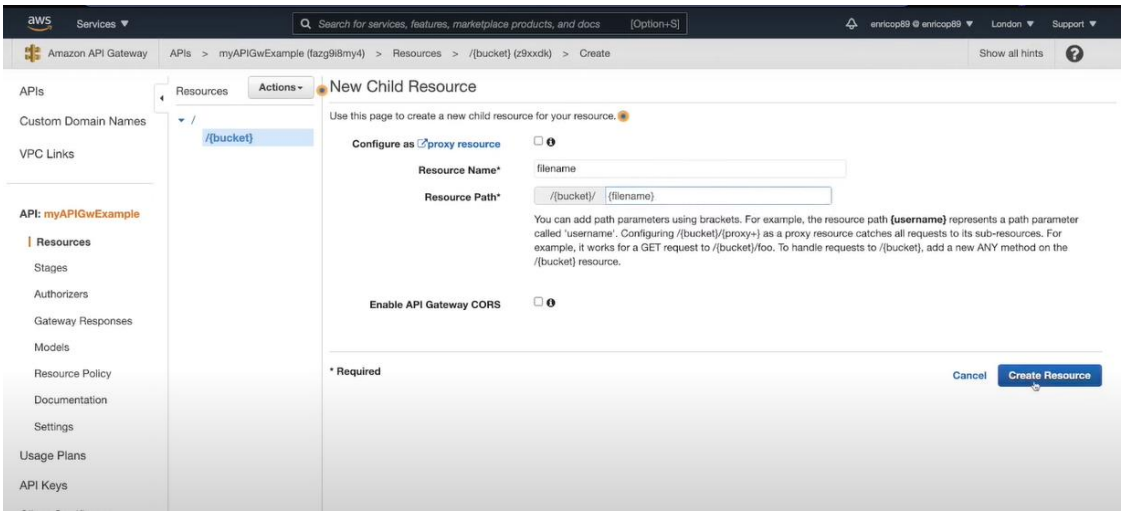


STEP 13: Give your API a name and description. Click "Create API."



STEP 14: In the left sidebar, click on the "Resources" section. Under Actions menu, click "Create" to create a new resource. Enter a resource name, e.g., "/bucket" or any path you prefer. Click "Create Resource".

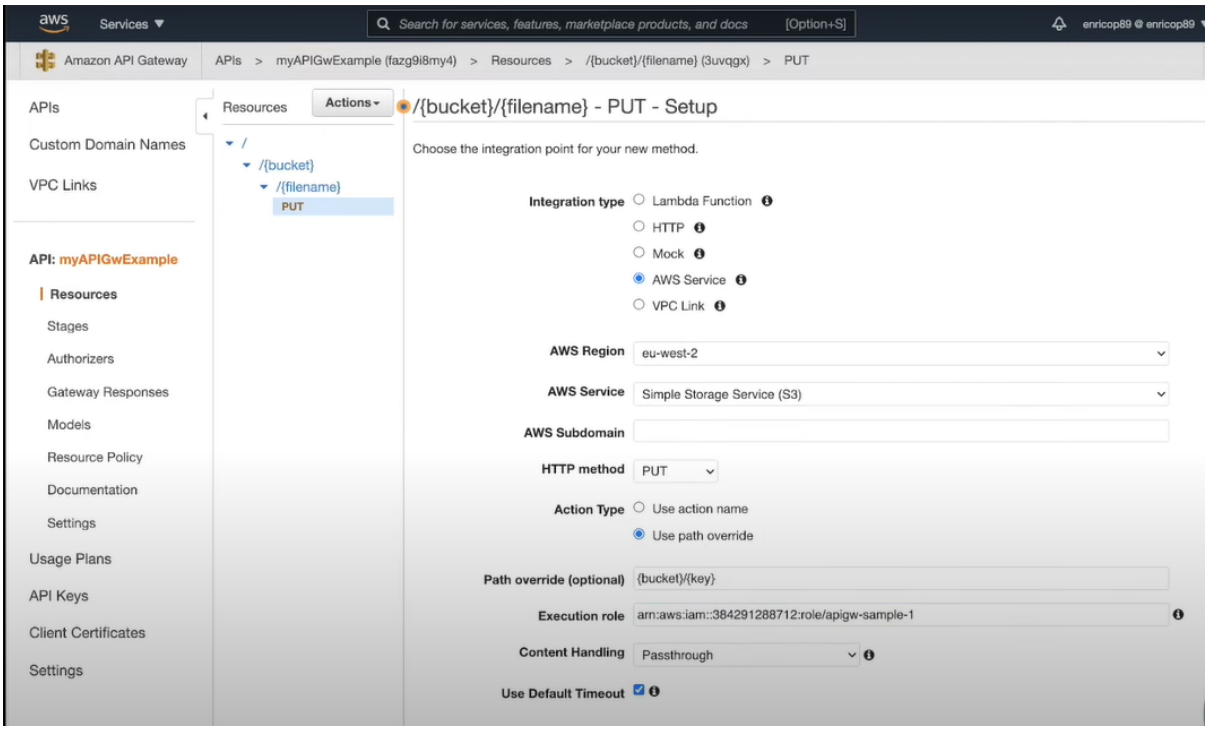


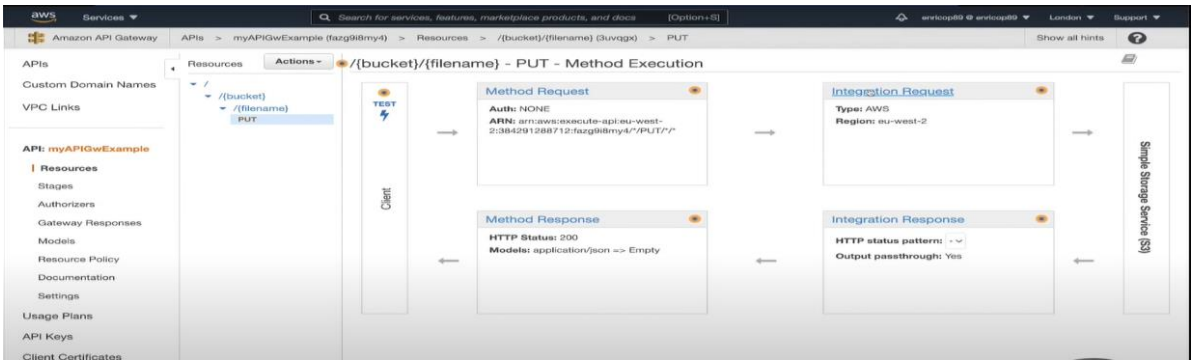


STEP 15: Set Up AWS Service Integration

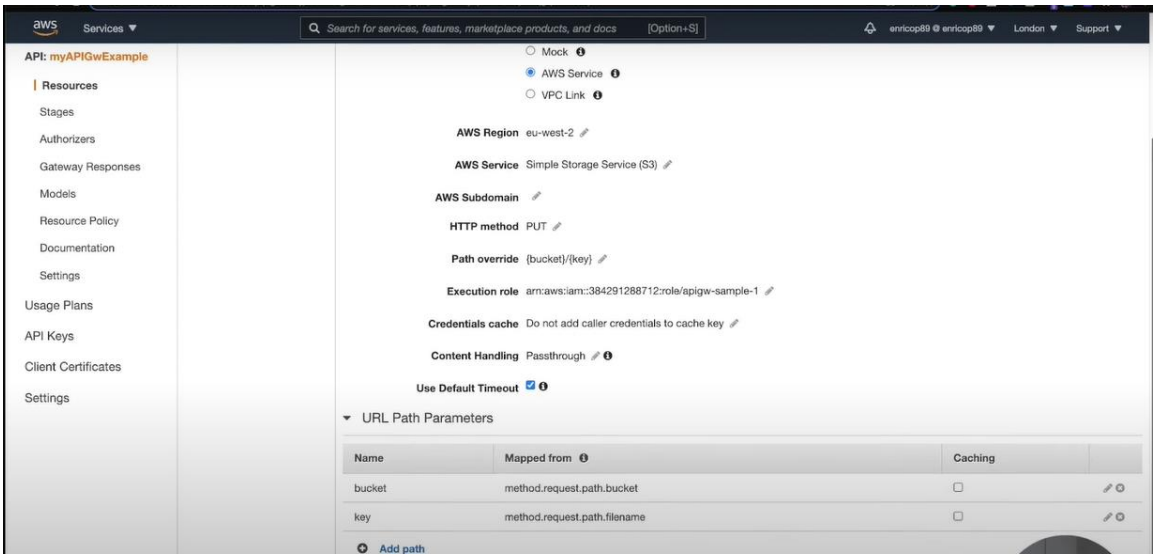
- **In the "AWS Service" settings:**
  - **AWS Region:** Choose the AWS region where your S3 Bucket is located.
  - **AWS Service:** Select "S3".
  - **HTTP Method:** Choose the appropriate method for your S3 operation.
  - **Action:** Set up the "Action" to use the "Path Override."
- **Path Override: I**n the "Path Override" field, specify the path you want to use for the S3 request. This allows you to modify the path portion of the URL.

- **Execution Role:** Create or select an IAM Role that grants permissions to access S3. This role should have policies that allow the necessary actions on your S3 Bucket.

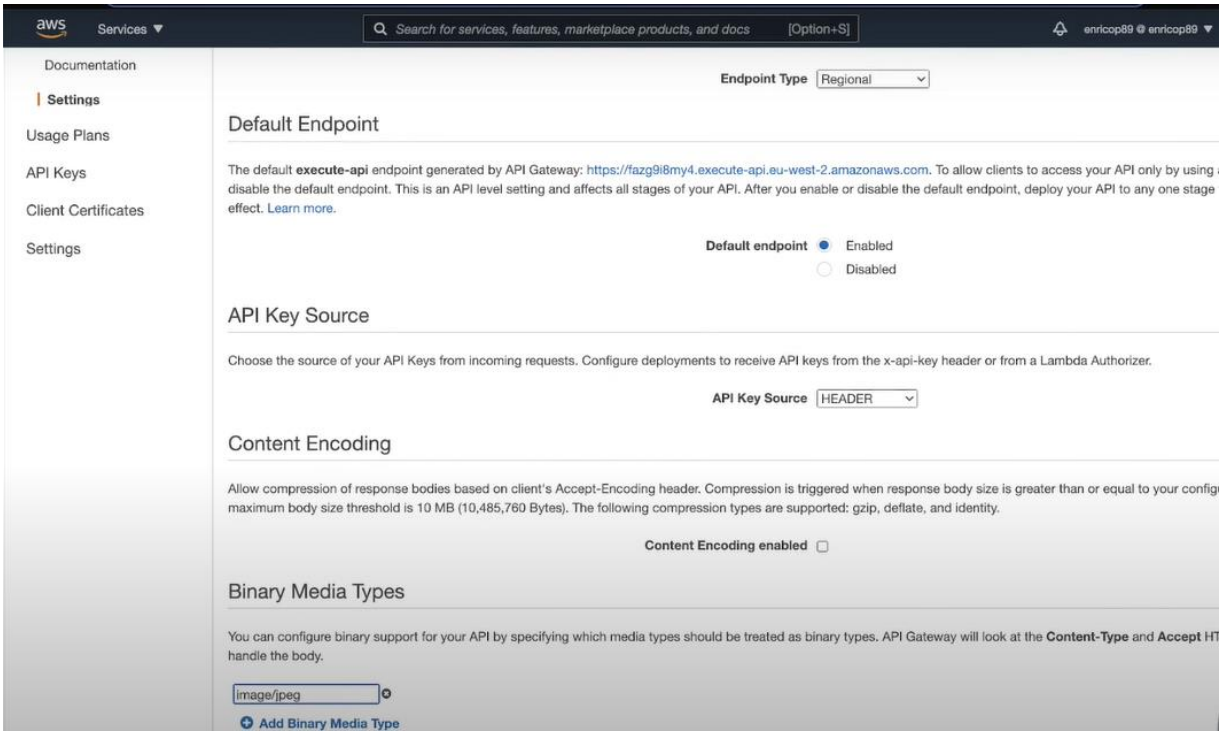




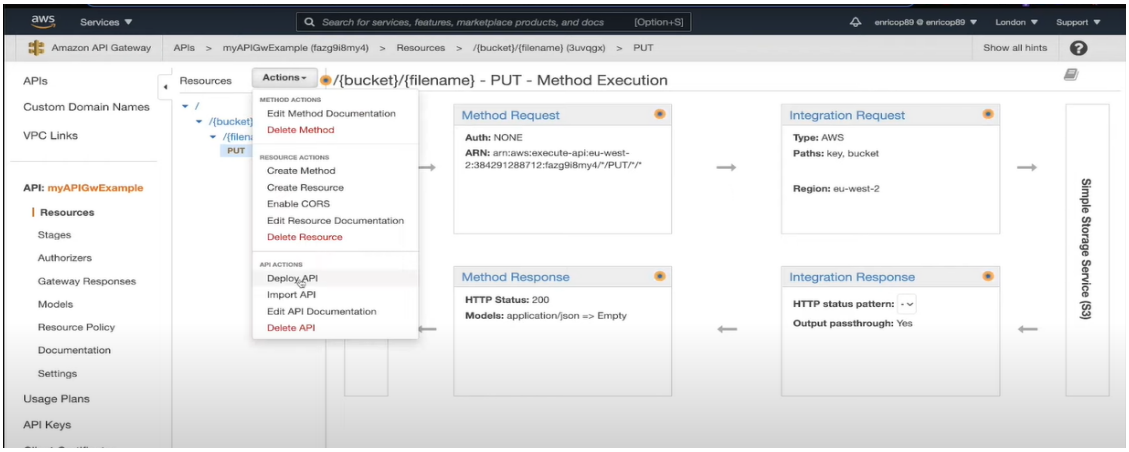
STEP 16: Under "Integration Request", configure how API Gateway should handle URL path parameters and set up binary media types.

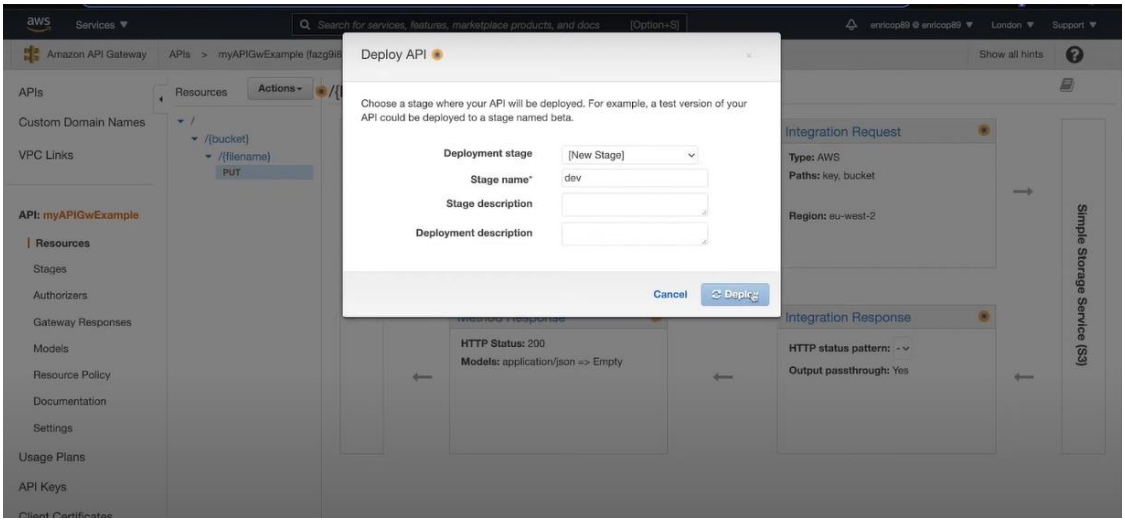


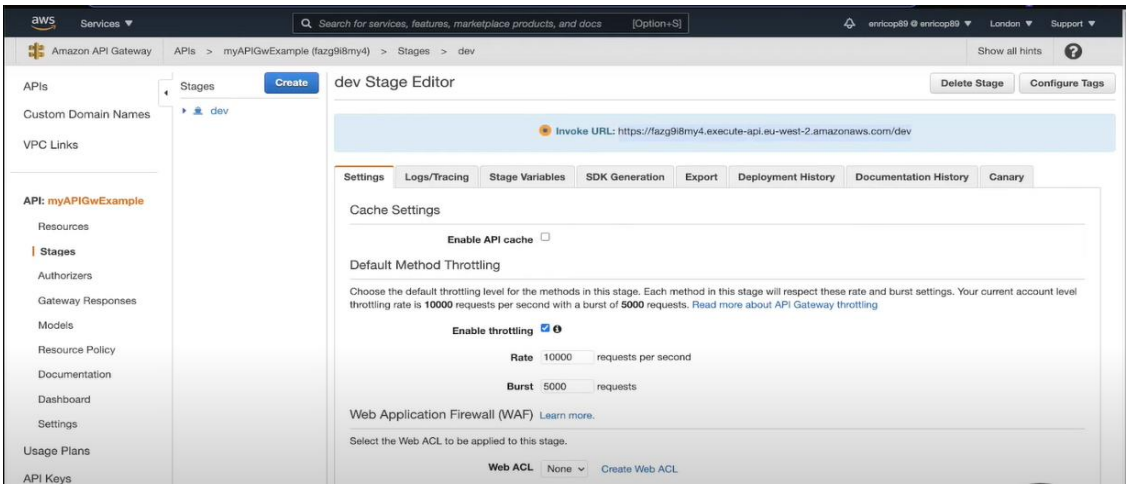
STEP 17: To enable binary media types, go to the API Gateway settings. Under "Binary Media Types," add the appropriate content types for binary data you want to support. In your case, for `image/jpeg`, you would add `image/jpeg`.



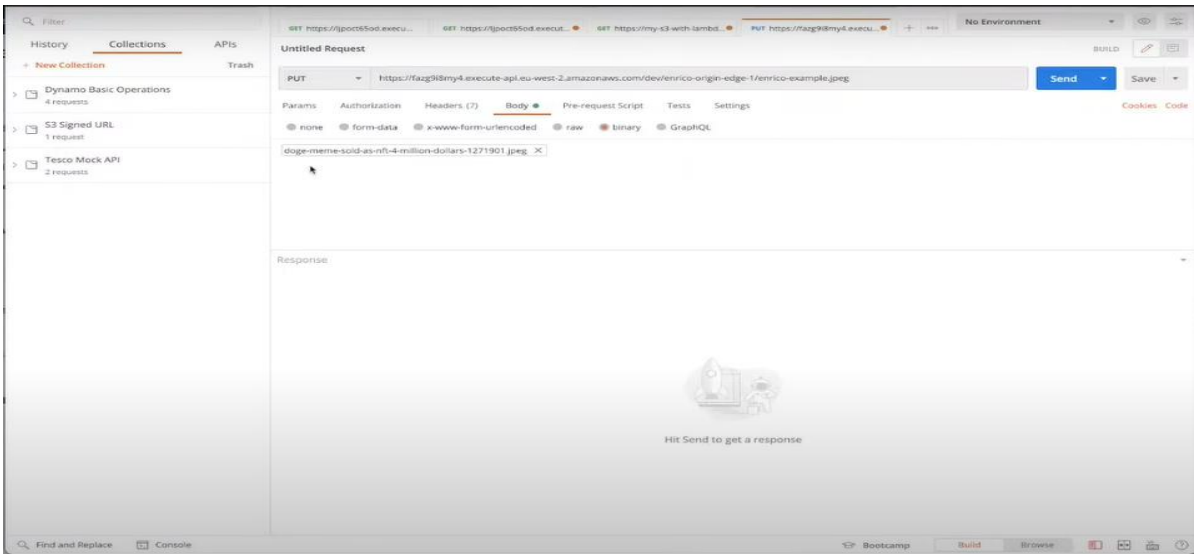
STEP 18: After configuring method and integration request settings, deploy your API so that it's accessible via a public URL.

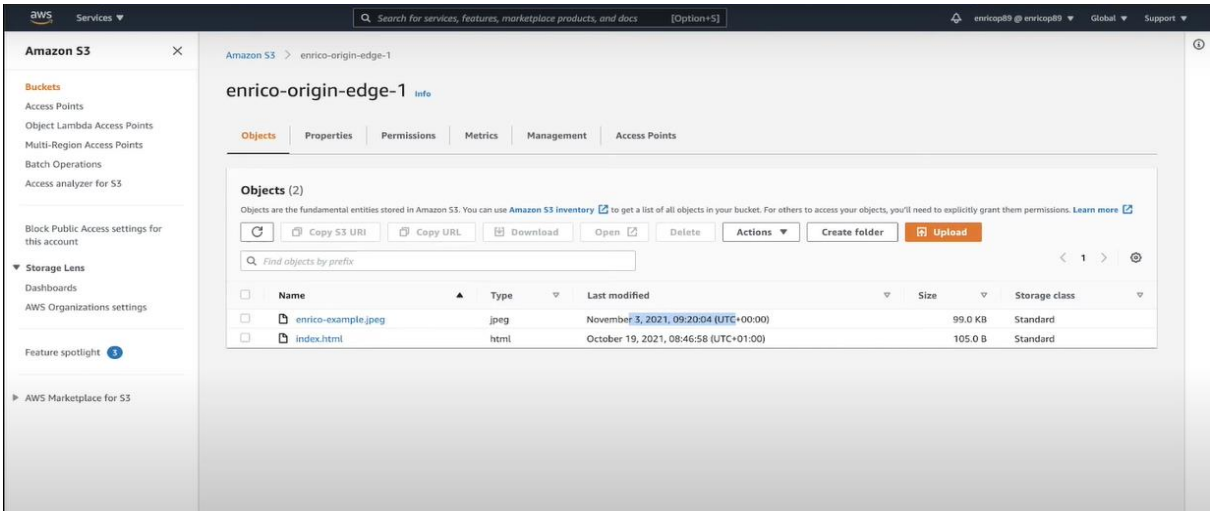






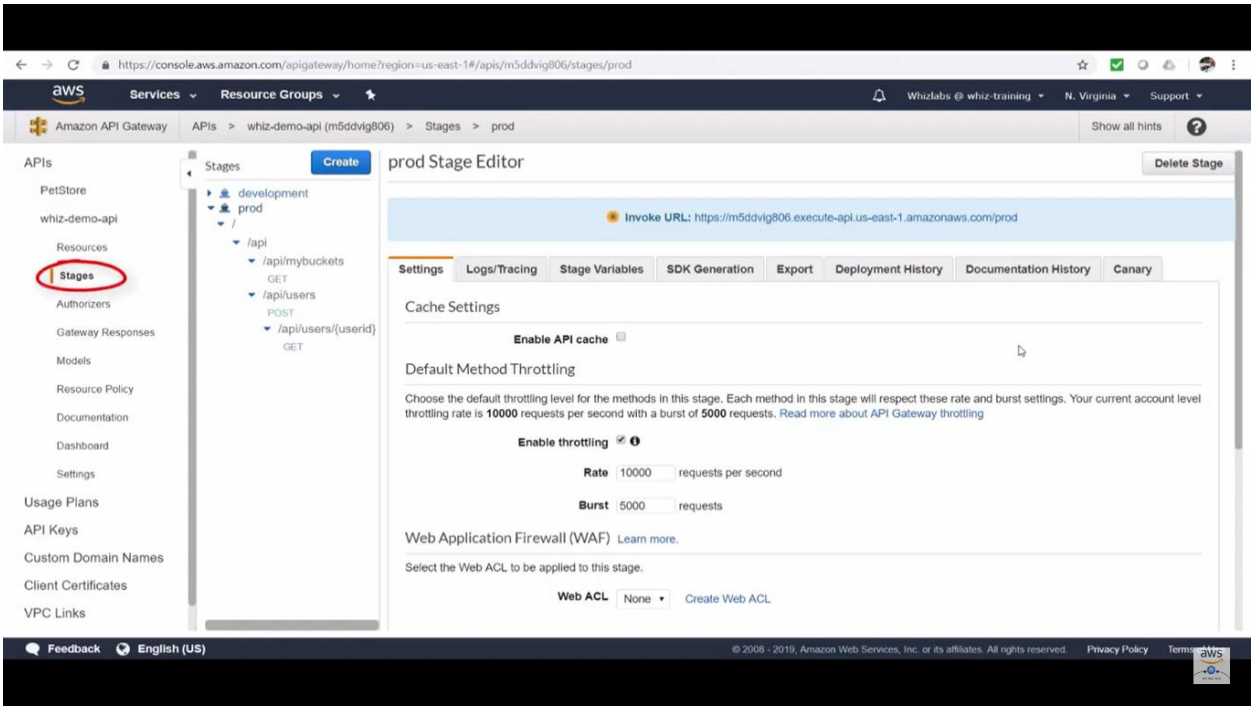
STEP 19: Test your API endpoint by sending a POST request with a file upload to ensure that the file is correctly forwarded to your S3 bucket.





UPDATE STAGE SETTINGS USING API GATEWAY CONSOLE

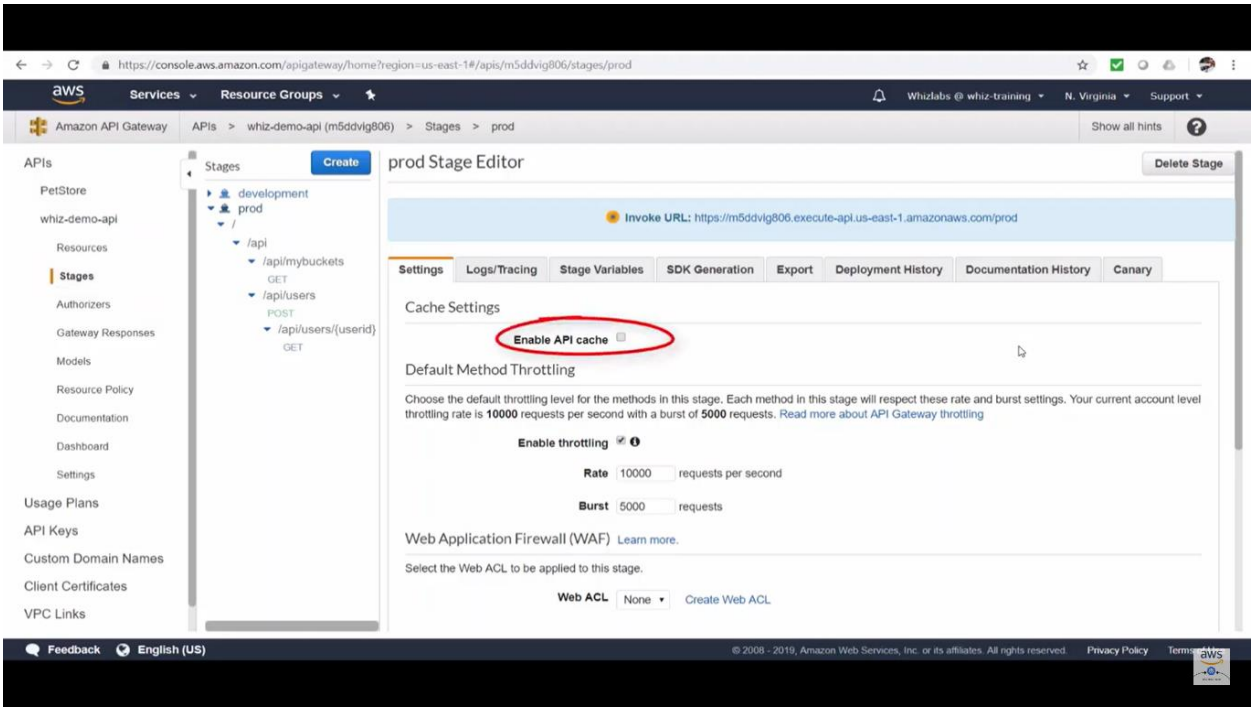
After a successful deployment of an API, the stage is populated with default settings. You can use the console or the API Gateway REST API to change the stage settings, including API caching and logging. The following steps show you how to do so using the **Stage Editor** of the API Gateway console.

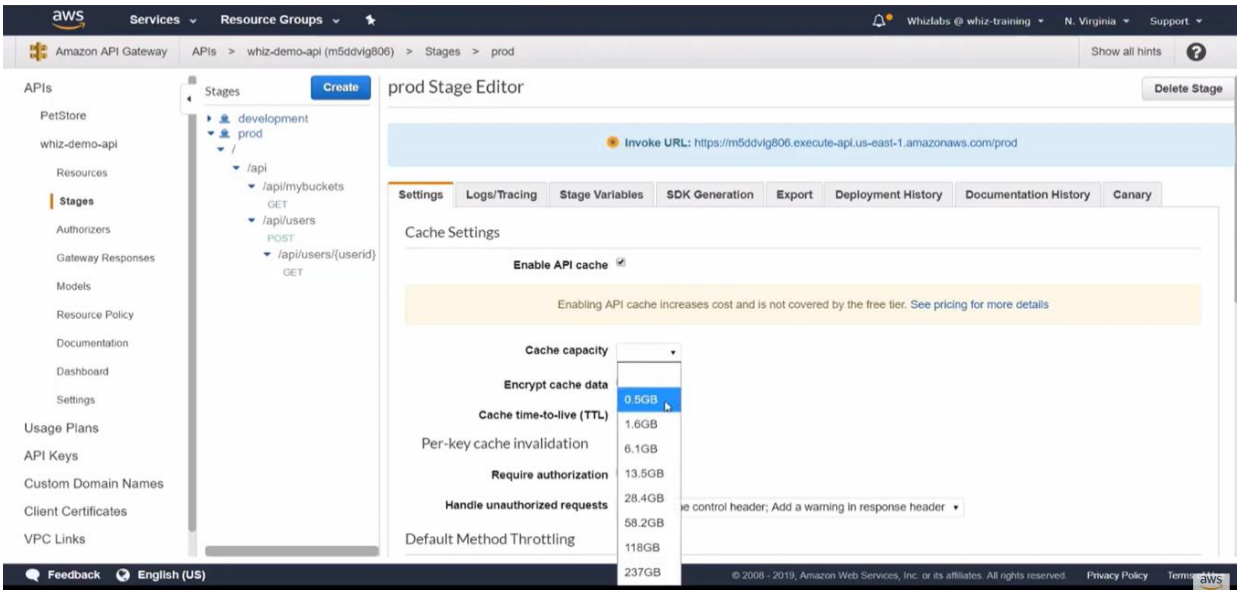


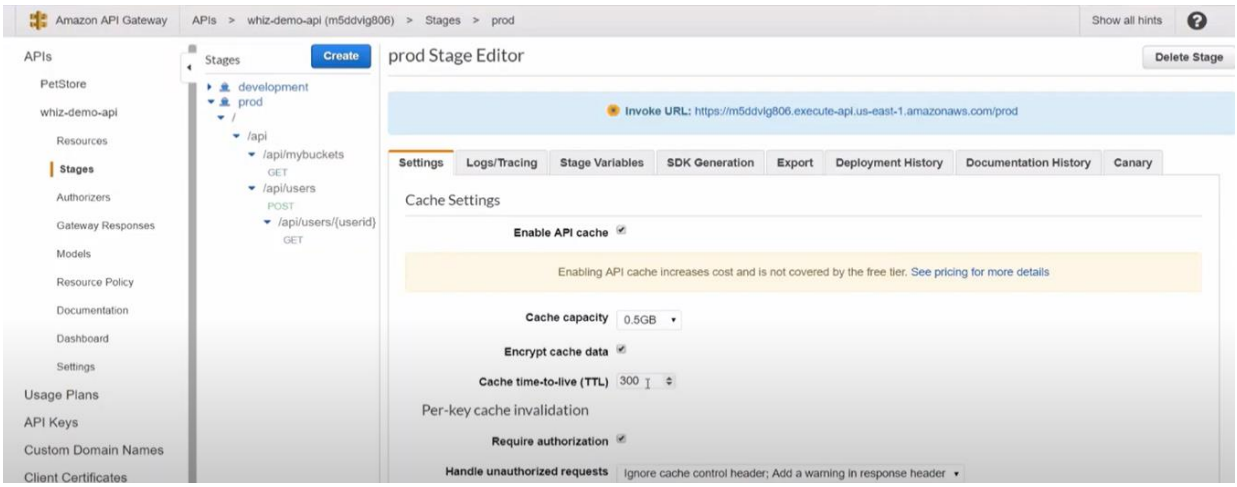
1.CACHE SETTINGS

NOTE: Caching is charged by the hour based on the cache size that you select. Caching is not eligible for the AWS Free Tier

Enable API Cache: This option allows you to enable or disable caching for the selected stage. When enabled, API Gateway will start caching responses for your API.


https://console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/m5ddvig806/stages/prod
Services
Resource Groups
Whizlabs @ whiz-training
N. Virginia
Support
Amazon API Gateway
APIs > whiz-demo-api (m5ddvig806) > Stages > prod
Show all hints
APIs
PetStore
whiz-demo-api
Resources
Stages
Authorizers
Gateway Responses
Models
Resource Policy
Documentation
Dashboard
Settings
Usage Plans
API Keys
Custom Domain Names
Client Certificates
VPC Links
Stages
Create
development
prod
/
/api
/api/mybuckets
GET
/api/users
POST
/api/users/{userid}
GET
prod Stage Editor
Delete Stage
Invoke URL: https://m5ddvig806.execute-api.us-east-1.amazonaws.com/prod
Settings
Logs/Tracing
Stage Variables
SDK Generation
Export
Deployment History
Documentation History
Canary
Cache Settings
Enable API cache
Default Method Throttling
Choose the default throttling level for the methods in this stage. Each method in this stage will respect these rate and burst settings. Your current account level throttling rate is 10000 requests per second with a burst of 5000 requests. Read more about API Gateway throttling
Enable throttling
Rate 10000 requests per second
Burst 5000 requests
Web Application Firewall (WAF) Learn more.
Select the Web ACL to be applied to this stage.
Web ACL None Create Web ACL
Feedback
English (US)
© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy


Services
Resource Groups
Whizlabs @ whiz-training
N. Virginia
Support
Amazon API Gateway
APIs > whiz-demo-api (m5ddvig806) > Stages > prod
Show all hints
APIs
PetStore
whiz-demo-api
Resources
Stages
Authorizers
Gateway Responses
Models
Resource Policy
Documentation
Dashboard
Settings
Usage Plans
API Keys
Custom Domain Names
Client Certificates
VPC Links
Stages
Create
development
prod
/
/api
/api/mybuckets
GET
/api/users
POST
/api/users/{userid}
GET
prod Stage Editor
Delete Stage
Invoke URL: https://m5ddvig806.execute-api.us-east-1.amazonaws.com/prod
Settings
Logs/Tracing
Stage Variables
SDK Generation
Export
Deployment History
Documentation History
Canary
Cache Settings
Enable API cache
Enabling API cache increases cost and is not covered by the free tier. See pricing for more details
Cache capacity
Encrypt cache data
Cache time-to-live (TTL)
Per-key cache invalidation
Require authorization
Handle unauthorized requests
0.5GB
1.6GB
6.1GB
13.5GB
28.4GB
58.2GB
118GB
237GB
control header; Add a warning in response header
Default Method Throttling
Feedback
English (US)
© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy



Use Case: You have an e-commerce API that provides product information. Product details change occasionally, but not very frequently. You want to reduce the load on your backend while ensuring that clients receive reasonably current information.

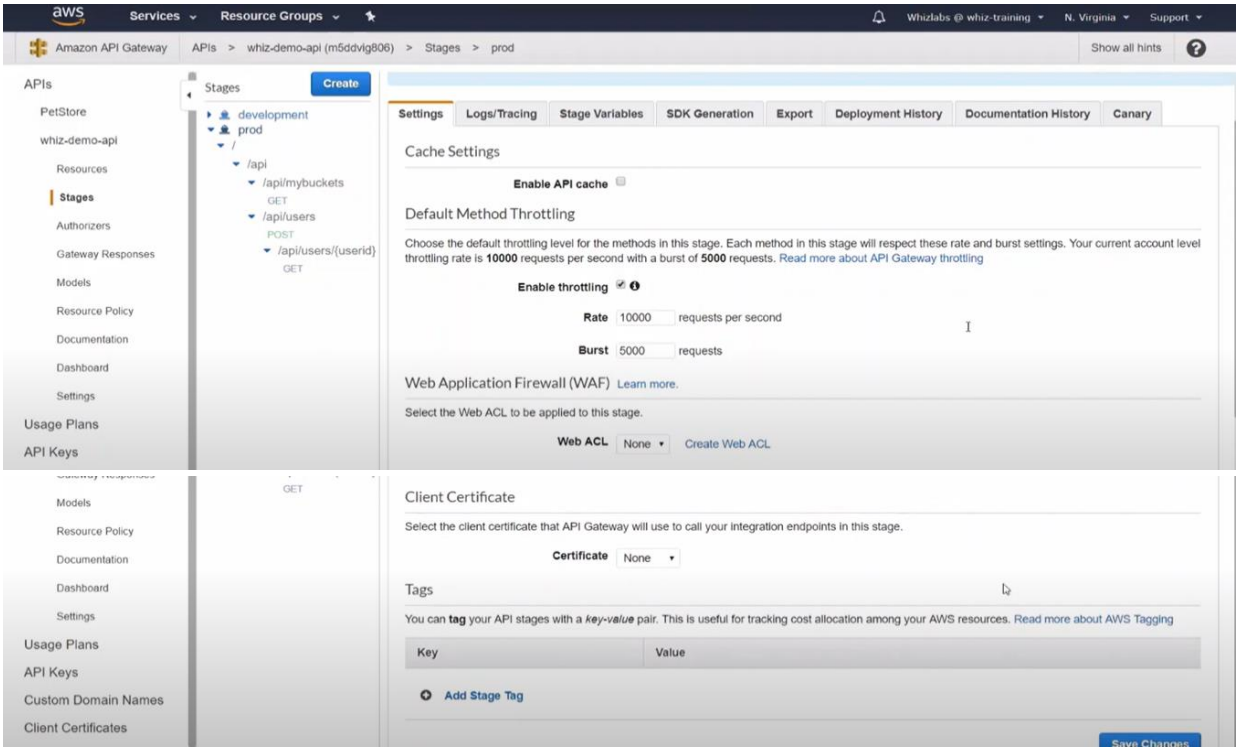
Cache Settings Choice:

Cache Capacity: Depending on the traffic and the size of your product details, you might allocate a cache capacity of, say, 100 MB.

Cache TTL: Since product details change relatively infrequently, you can set a TTL of 1 hour. This means that once a product's details are cached, they will be served from the cache for up to an hour before being refreshed from the backend.

Encrypt Cache Data: Enabling data encryption for the cache ensures that any data stored in the cache is protected with encryption.

2.DEFAULT METHOD THROTTLING



Use Case: Imagine you are operating an e-commerce platform with an API that allows users to check product availability and place orders. In this scenario, you want to use the "Default Throttle Settings" at the stage level to manage requests based on different user roles and protect your inventory.

Stage: Your API has a production stage where various users interact with your e-commerce system.

Default Throttle Settings Configuration:

Rate:

- Configure rate limits for different user roles:

- For Regular Users: Set a moderate rate limit (e.g., 60 requests per minute) for accessing product information and placing orders. This ensures that regular users can interact with your platform without overwhelming your backend systems.

- For Admin Users: Set a higher rate limit (e.g., 100 requests per minute) for administrators who need to perform tasks like inventory management and order processing.

Burst:

- Define a burst limit that allows temporary spikes in traffic without immediate throttling. For example, you could set a burst limit of 20 requests for both regular users and admin users.

3.LOGS/TRACING

Log Level: Choose the desired log level from the dropdown menu. The log level determines the verbosity of the logs. Options may include:

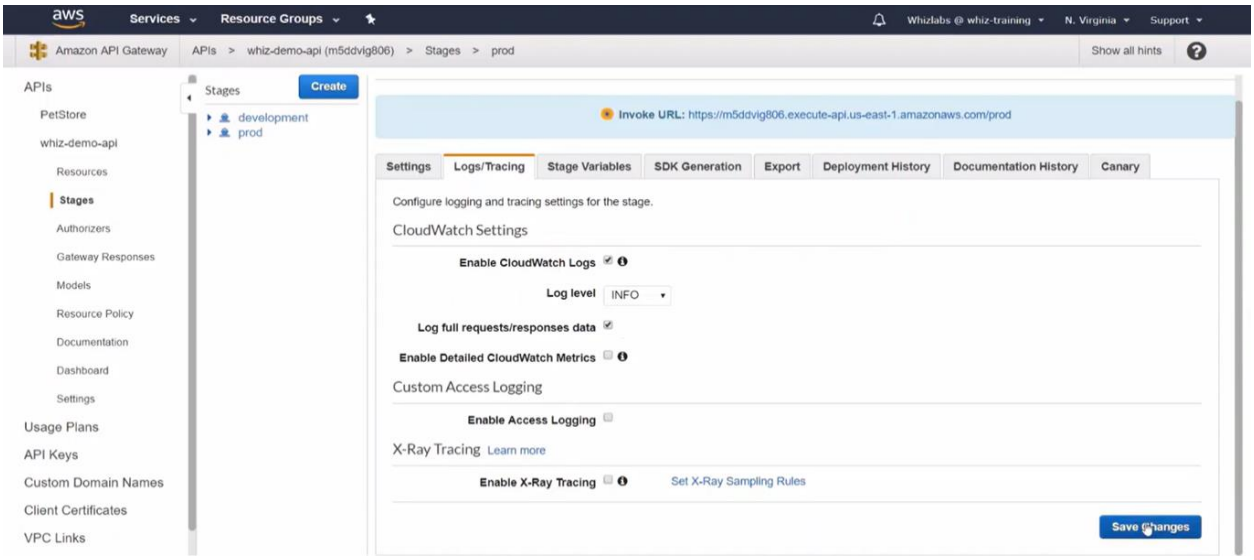
- OFF: No logs are generated.
- ERROR: Only error-related logs are generated.
- INFO: Informational logs are generated in addition to errors.
- DEBUG: Detailed debugging logs are generated.

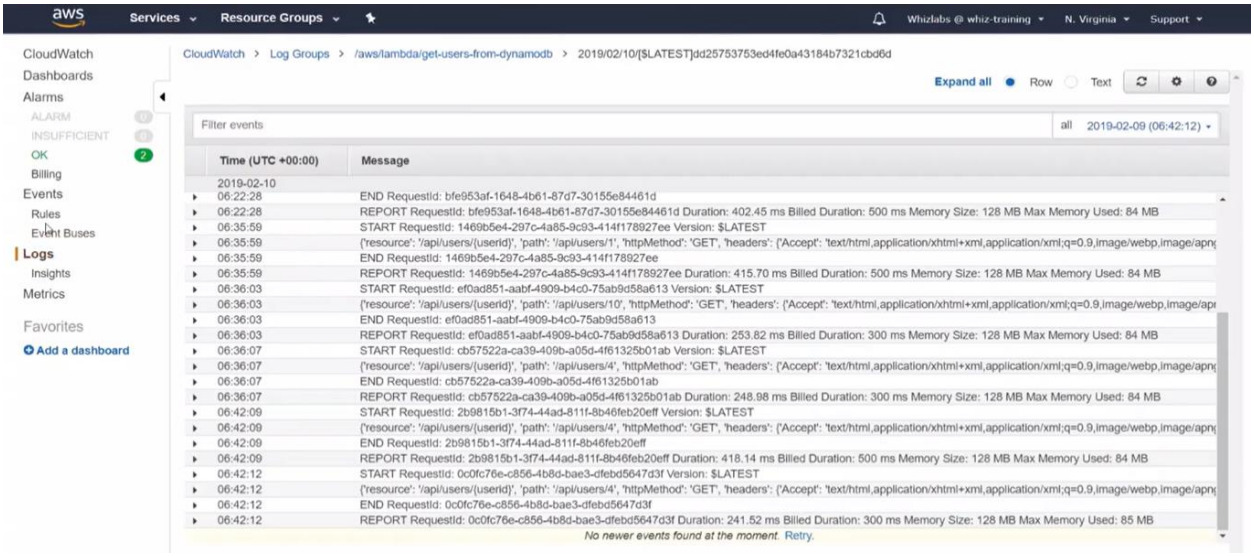
Select the log level that suits your debugging and monitoring needs.

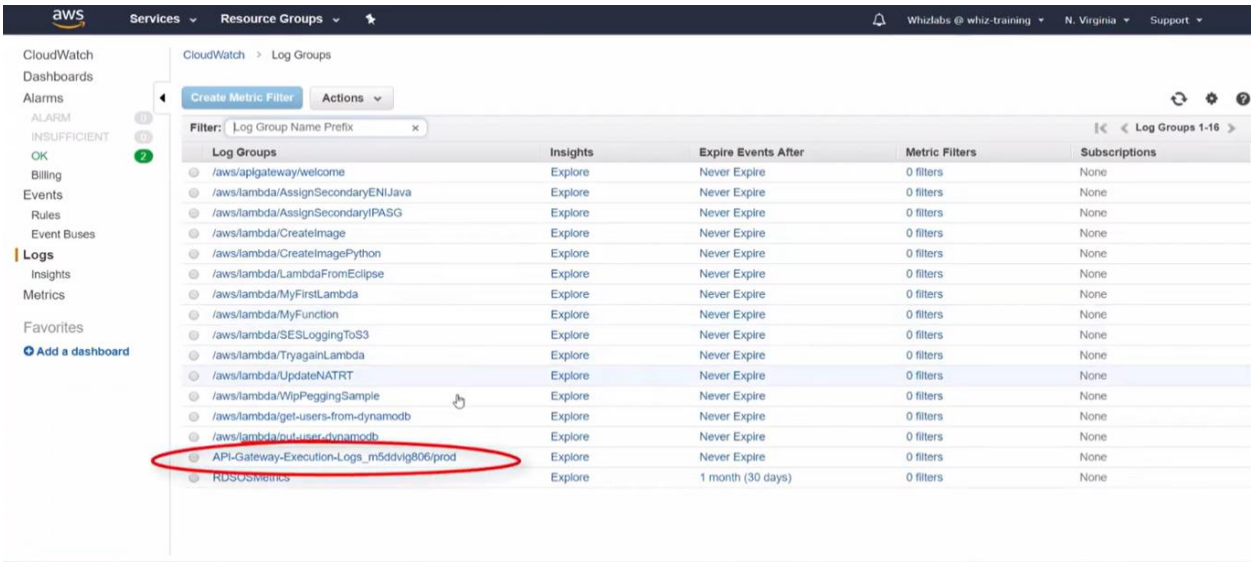
Log full request/response data: You can enable or disable this option based on your requirements. Enabling it will capture the entire request and response data in the logs, which can be useful for debugging but may generate a lot of data, so use it judiciously.

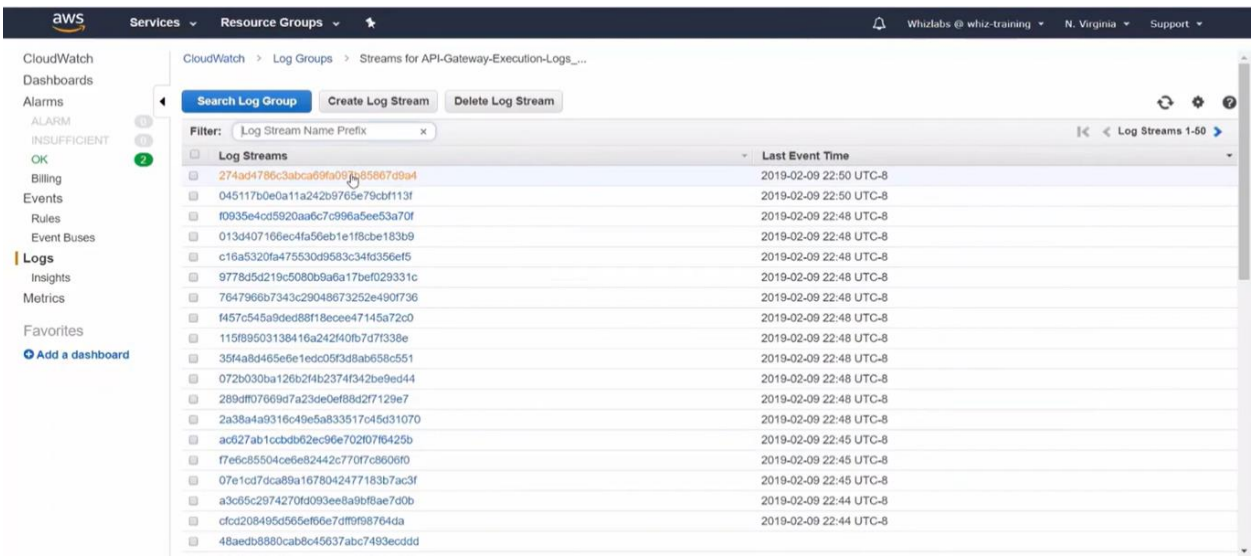
Enable Detailed CloudWatch Metrics: Optionally, you can enable detailed CloudWatch metrics for your API Gateway stage. Detailed metrics provide more granular information about the performance of your API.

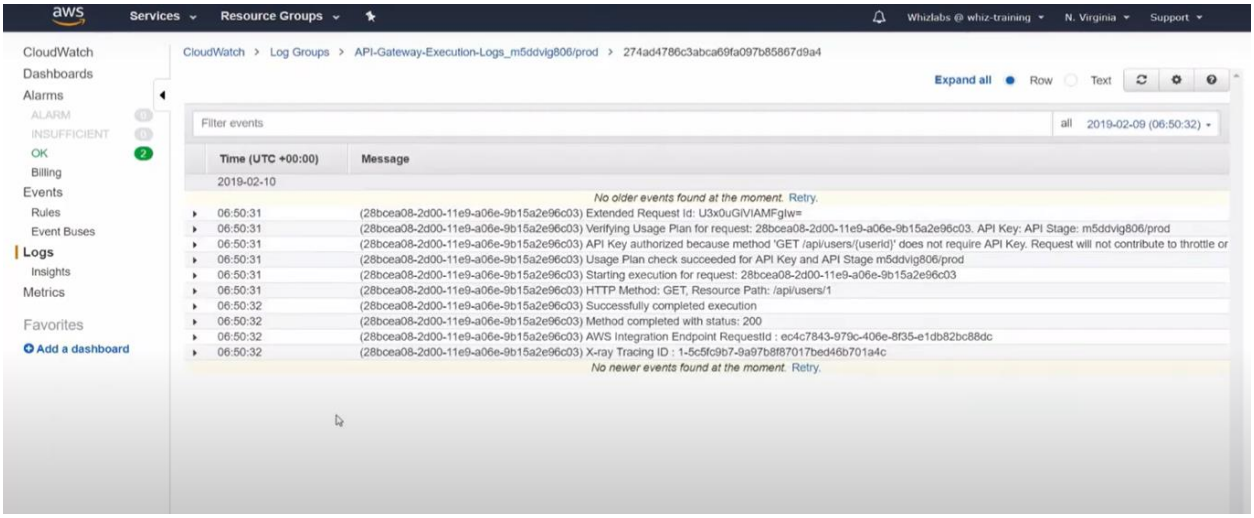
Access Logging: Enable access logging by selecting the "Access Logging" checkbox.



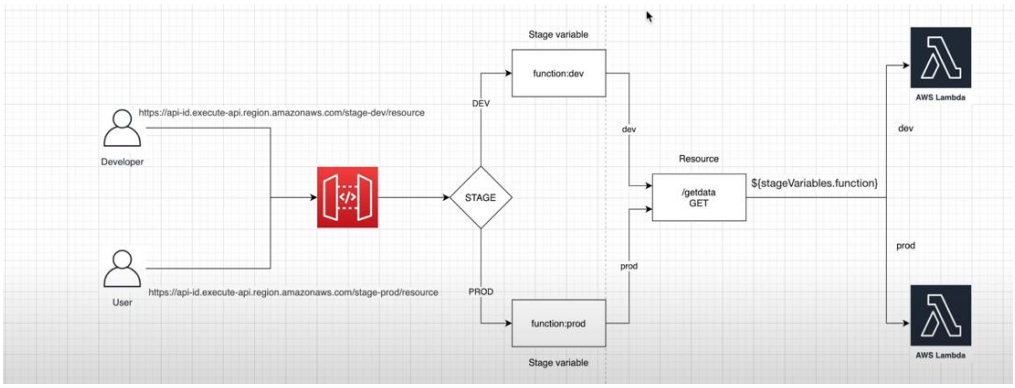

aws
Services
Resource Groups
Whizlabs @ whiz-training
N. Virginia
Support
CloudWatch
Dashboards
Alarms
ALARM
INSUFFICIENT
OK
Billing
Events
Rules
Event Buses
Logs
Insights
Metrics
Favorites
Add a dashboard
CloudWatch > Log Groups > /aws/lambda/get-users-from-dynamodb > 2019/02/10/[$LATEST]dd25753753ed4fe0a43184b7321cbd6d
Expand all
Row
Text
Filter events
all
2019-02-09 (06:42:12)
Time (UTC +00:00)
Message
2019-02-10
06:22:28 END RequestId: bfe953af-1648-4b61-87d7-30155e84461d
06:22:28 REPORT RequestId: bfe953af-1648-4b61-87d7-30155e84461d Duration: 402.45 ms Billed Duration: 500 ms Memory Size: 128 MB Max Memory Used: 84 MB
06:35:59 START RequestId: 1469b5e4-297c-4a85-9c93-414f178927ee Version: $LATEST
06:35:59 {'resource': '/api/users/{userid}', 'path': '/api/users/1', 'httpMethod': 'GET', 'headers': {'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng
06:35:59 END RequestId: 1469b5e4-297c-4a85-9c93-414f178927ee
06:35:59 REPORT RequestId: 1469b5e4-297c-4a85-9c93-414f178927ee Duration: 415.70 ms Billed Duration: 500 ms Memory Size: 128 MB Max Memory Used: 84 MB
06:36:03 START RequestId: ef0ad851-aabf-4909-b4c0-75ab9d58a613 Version: $LATEST
06:36:03 {'resource': '/api/users/{userid}', 'path': '/api/users/10', 'httpMethod': 'GET', 'headers': {'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng
06:36:03 END RequestId: ef0ad851-aabf-4909-b4c0-75ab9d58a613
06:36:03 REPORT RequestId: ef0ad851-aabf-4909-b4c0-75ab9d58a613 Duration: 253.82 ms Billed Duration: 300 ms Memory Size: 128 MB Max Memory Used: 84 MB
06:36:07 START RequestId: cb57522a-ca39-409b-a05d-4f61325b01ab Version: $LATEST
06:36:07 {'resource': '/api/users/{userid}', 'path': '/api/users/4', 'httpMethod': 'GET', 'headers': {'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng
06:36:07 END RequestId: cb57522a-ca39-409b-a05d-4f61325b01ab
06:36:07 REPORT RequestId: cb57522a-ca39-409b-a05d-4f61325b01ab Duration: 248.98 ms Billed Duration: 300 ms Memory Size: 128 MB Max Memory Used: 84 MB
06:42:09 START RequestId: 2b9815b1-3f74-44ad-811f-8b46feb20eff Version: $LATEST
06:42:09 {'resource': '/api/users/{userid}', 'path': '/api/users/4', 'httpMethod': 'GET', 'headers': {'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng
06:42:09 END RequestId: 2b9815b1-3f74-44ad-811f-8b46feb20eff
06:42:09 REPORT RequestId: 2b9815b1-3f74-44ad-811f-8b46feb20eff Duration: 418.14 ms Billed Duration: 500 ms Memory Size: 128 MB Max Memory Used: 84 MB
06:42:12 START RequestId: 0c0fc76e-c856-4b8d-bae3-dfebd5647d3f Version: $LATEST
06:42:12 {'resource': '/api/users/{userid}', 'path': '/api/users/4', 'httpMethod': 'GET', 'headers': {'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng
06:42:12 END RequestId: 0c0fc76e-c856-4b8d-bae3-dfebd5647d3f
06:42:12 REPORT RequestId: 0c0fc76e-c856-4b8d-bae3-dfebd5647d3f Duration: 241.52 ms Billed Duration: 300 ms Memory Size: 128 MB Max Memory Used: 85 MB
No newer events found at the moment. Retry.


aws
Services
Resource Groups
Whizlabs @ whiz-training
N. Virginia
Support
CloudWatch
Dashboards
Alarms
ALARM
INSUFFICIENT
OK
Billing
Events
Rules
Event Buses
Logs
Insights
Metrics
Favorites
Add a dashboard
CloudWatch > Log Groups
Create Metric Filter
Actions
Filter: Log Group Name Prefix
Log Groups 1-16
Log Groups
Insights
Expire Events After
Metric Filters
Subscriptions
/aws/apigateway/welcome Explore Never Expire 0 filters None
/aws/lambda/AssignSecondaryENIJava Explore Never Expire 0 filters None
/aws/lambda/AssignSecondaryIPASG Explore Never Expire 0 filters None
/aws/lambda/CreateImage Explore Never Expire 0 filters None
/aws/lambda/CreateImagePython Explore Never Expire 0 filters None
/aws/lambda/LambdaFromEclipse Explore Never Expire 0 filters None
/aws/lambda/MyFirstLambda Explore Never Expire 0 filters None
/aws/lambda/MyFunction Explore Never Expire 0 filters None
/aws/lambda/SESLoggingToS3 Explore Never Expire 0 filters None
/aws/lambda/TryagainLambda Explore Never Expire 0 filters None
/aws/lambda/UpdateNATRT Explore Never Expire 0 filters None
/aws/lambda/WipPeggingSample Explore Never Expire 0 filters None
/aws/lambda/get-users-from-dynamodb Explore Never Expire 0 filters None
/aws/lambda/put-user-dynamodb Explore Never Expire 0 filters None
API-Gateway-Execution-Logs_m5ddvig806/prod Explore Never Expire 0 filters None
RDSOSMetrics Explore 1 month (30 days) 0 filters None

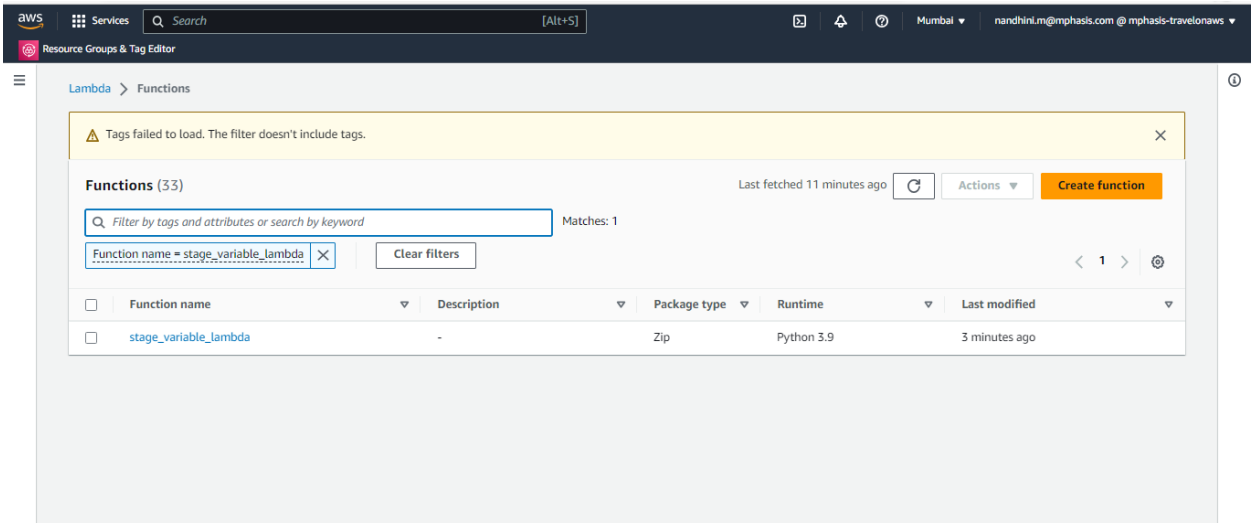

aws
Services
Resource Groups
Whizlabs @ whiz-training
N. Virginia
Support
CloudWatch
Dashboards
Alarms
ALARM
INSUFFICIENT
OK
Billing
Events
Rules
Event Buses
Logs
Insights
Metrics
Favorites
Add a dashboard
CloudWatch > Log Groups > Streams for API-Gateway-Execution-Logs_...
Search Log Group
Create Log Stream
Delete Log Stream
Filter: Log Stream Name Prefix
Log Streams 1-50
Log Streams
Last Event Time
274ad4786c3abca69fa097b85867d9a4 2019-02-09 22:50 UTC-8
045117b0e0a11a242b9765e79cbf113f 2019-02-09 22:50 UTC-8
f0935e4cd5920aa6c7c996a5ee53a70f 2019-02-09 22:48 UTC-8
013d407166ec4fa56eb1e1f8cbe183b9 2019-02-09 22:48 UTC-8
c16a5320fa475530d9583c34fd356ef5 2019-02-09 22:48 UTC-8
9778d5d219c5080b9a6a17bef029331c 2019-02-09 22:48 UTC-8
7647966b7343c29048673252e490f736 2019-02-09 22:48 UTC-8
f457c545a9ded88f18ecee47145a72c0 2019-02-09 22:48 UTC-8
115f89503138416a242f40fb7d7f338e 2019-02-09 22:48 UTC-8
35f4a8d465e6e1edc05f3d8ab658c551 2019-02-09 22:48 UTC-8
072b030ba126b2f4b2374f342be9ed44 2019-02-09 22:48 UTC-8
289dff07669d7a23de0ef88d2f7129e7 2019-02-09 22:48 UTC-8
2a38a4a9316c49e5a833517c45d31070 2019-02-09 22:48 UTC-8
ac627ab1ccbdb62ec96e702f07f6425b 2019-02-09 22:45 UTC-8
f7e6c85504ce6e82442c770f7c8606f0 2019-02-09 22:45 UTC-8
07e1cd7dca89a1678042477183b7ac3f 2019-02-09 22:45 UTC-8
a3c65c2974270fd093ee8a9bf8ae7d0b 2019-02-09 22:44 UTC-8
cfcd208495d565ef66e7dff9f98764da 2019-02-09 22:44 UTC-8
48aedb8880cab8c45637abc7493ecddd



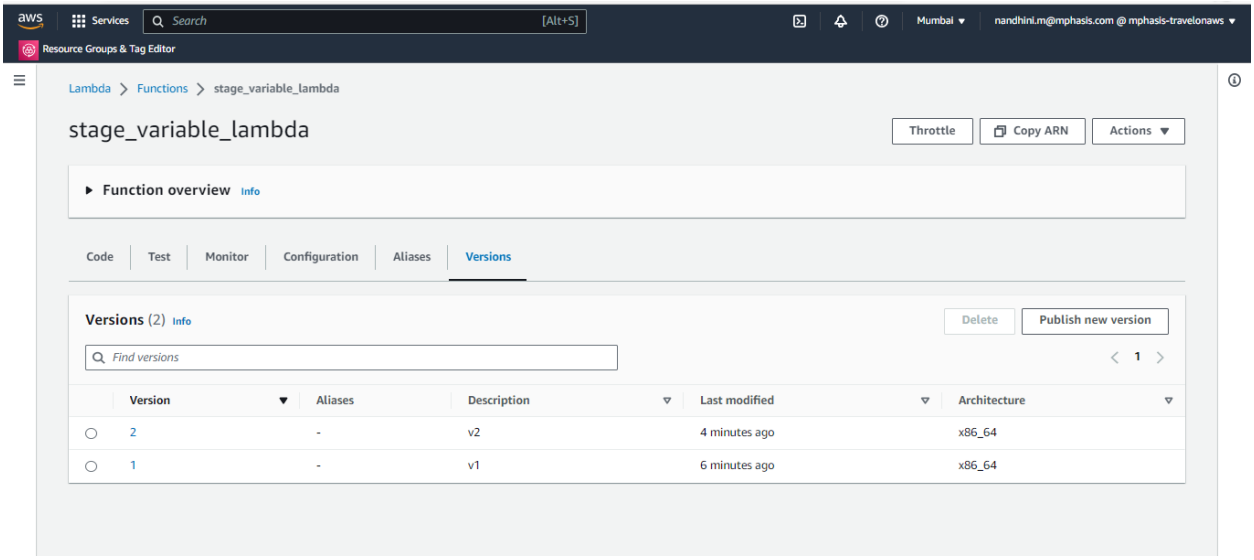
4.STAGE VARIABLE



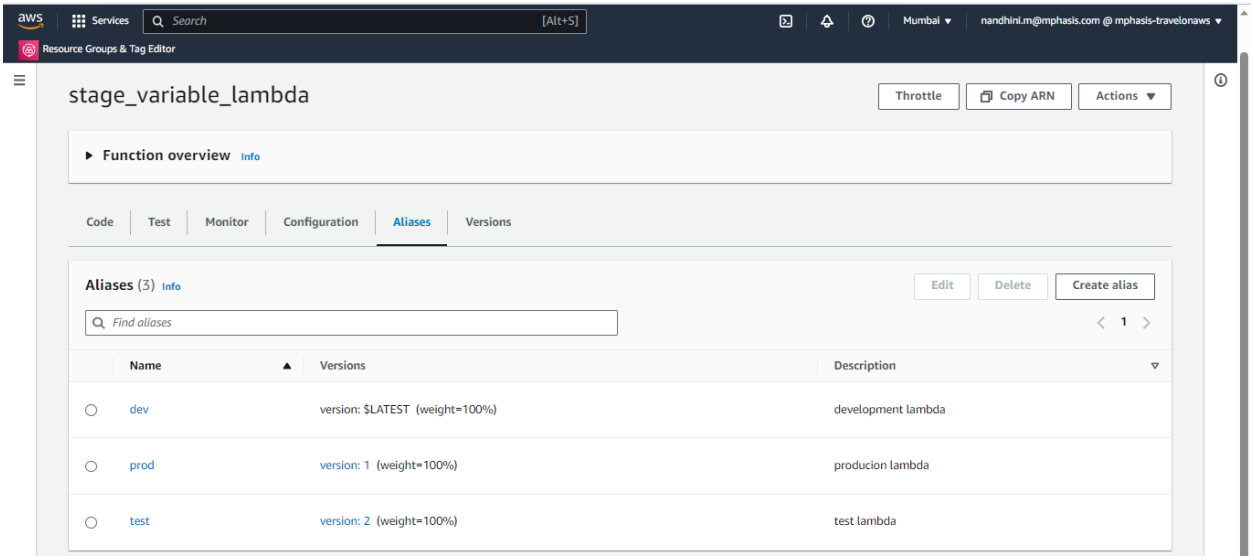
STEP 1: Create a Sample Lambda Function



Step 2: Publish Different Versions

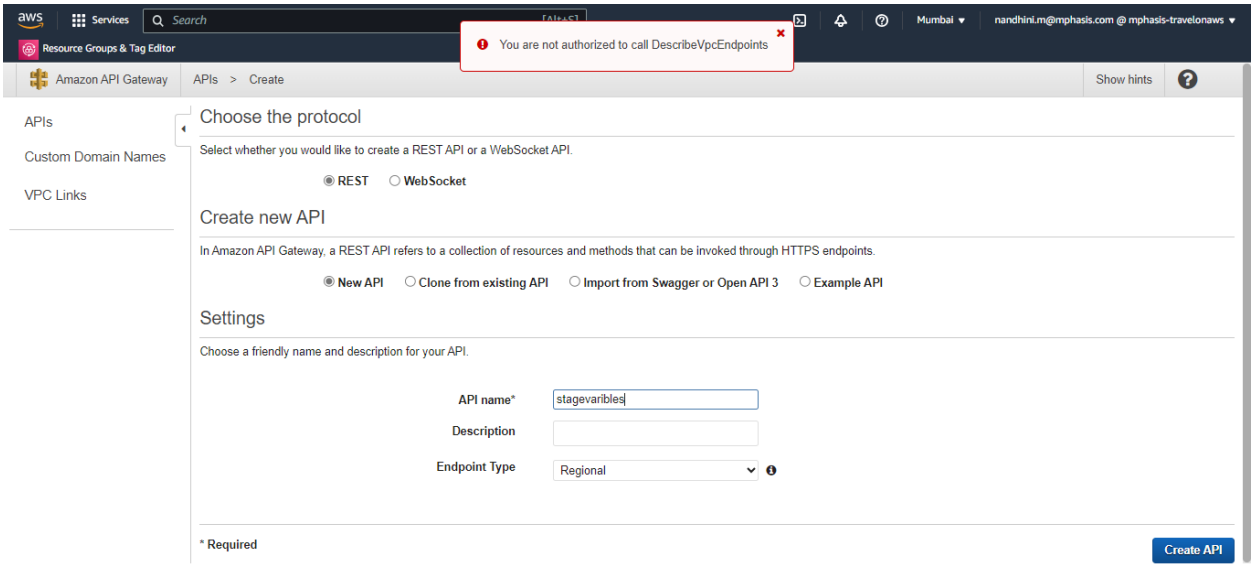


STEP 3: Create an Alias and Choose a version for the alias.



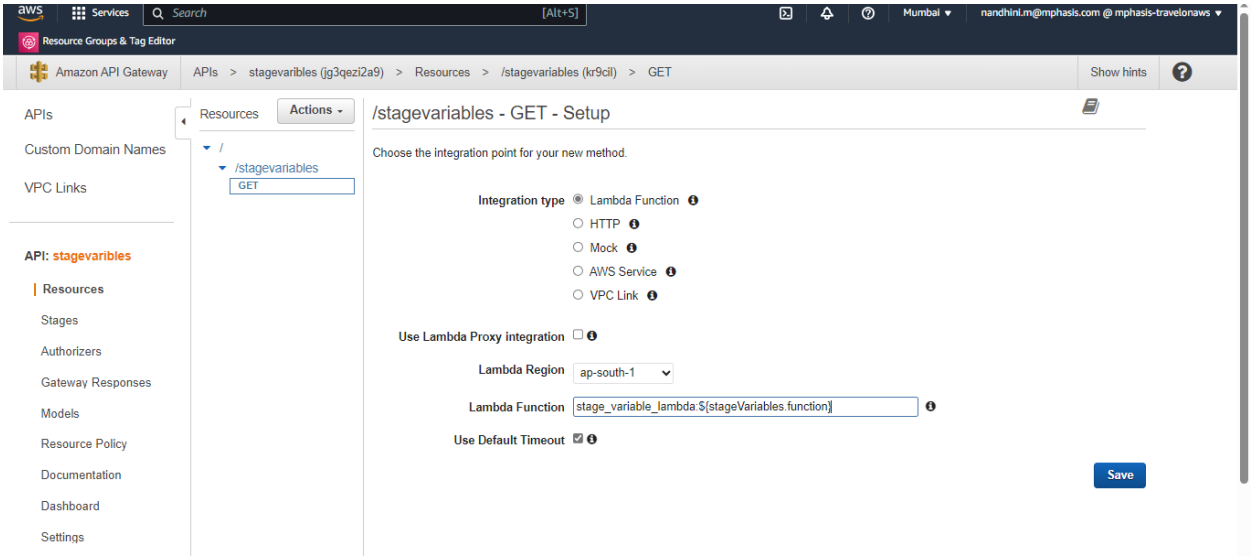
STEP 4: Set Up API Gateway with a Stage Variable.

- Go to the API Gateway service in the AWS Console.
- Select your API or create a new one if needed.



STEP 5: Create a resource and method (e.g., GET) for your API.

STEP 6: Configure the integration type to use Lambda and select the Lambda function alias you created.



STEP 7: Add permission to lambda so that API gateway can invoke

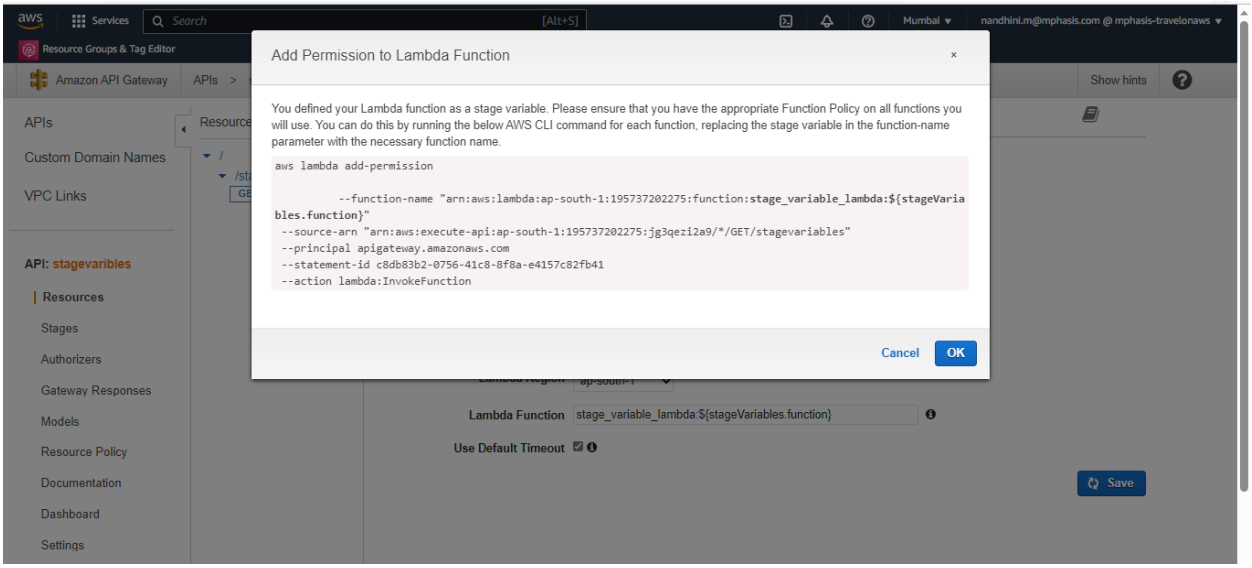
1. In the Lambda function integration settings, click on "Integration Request."
2. Under "Lambda Function," click on "Edit."
3. In the "Edit Integration" page, scroll down to the "Execution role" section.
4. Click on the "Edit" button next to the role name (e.g., "APIGatewayLambdaExecRole").

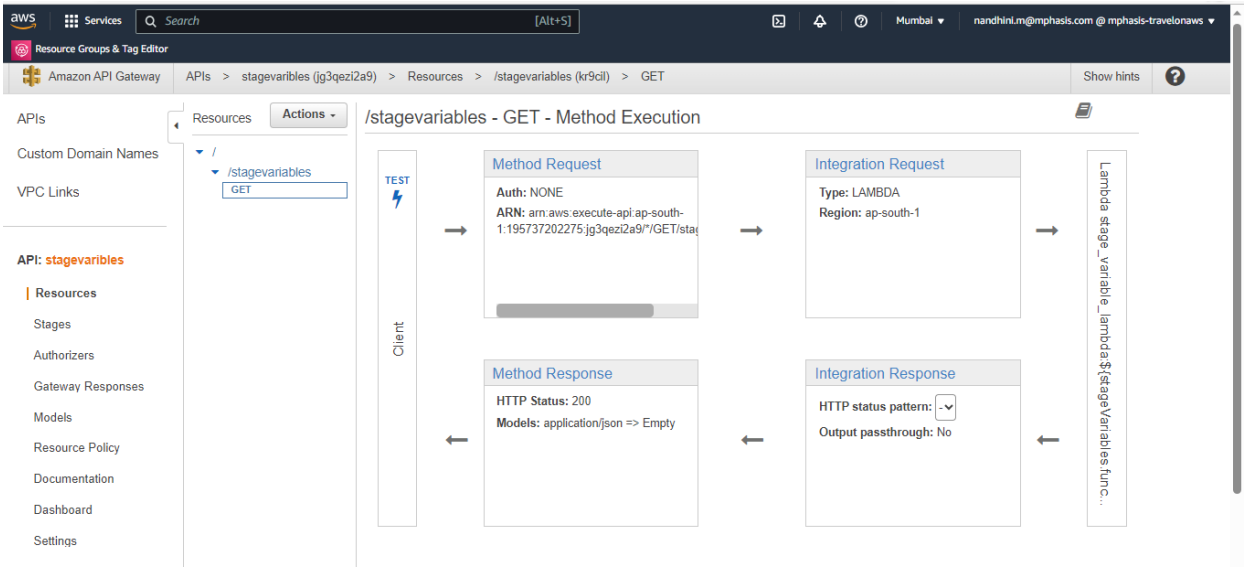
5. This will take you to the IAM Role configuration. To add permission, click on "Add inline policy."
6. In the "Create policy" wizard:
    - Choose the "JSON" tab for the policy document.
    - Enter a policy document that grants API Gateway permission to invoke the Lambda function. Here's a sample policy:

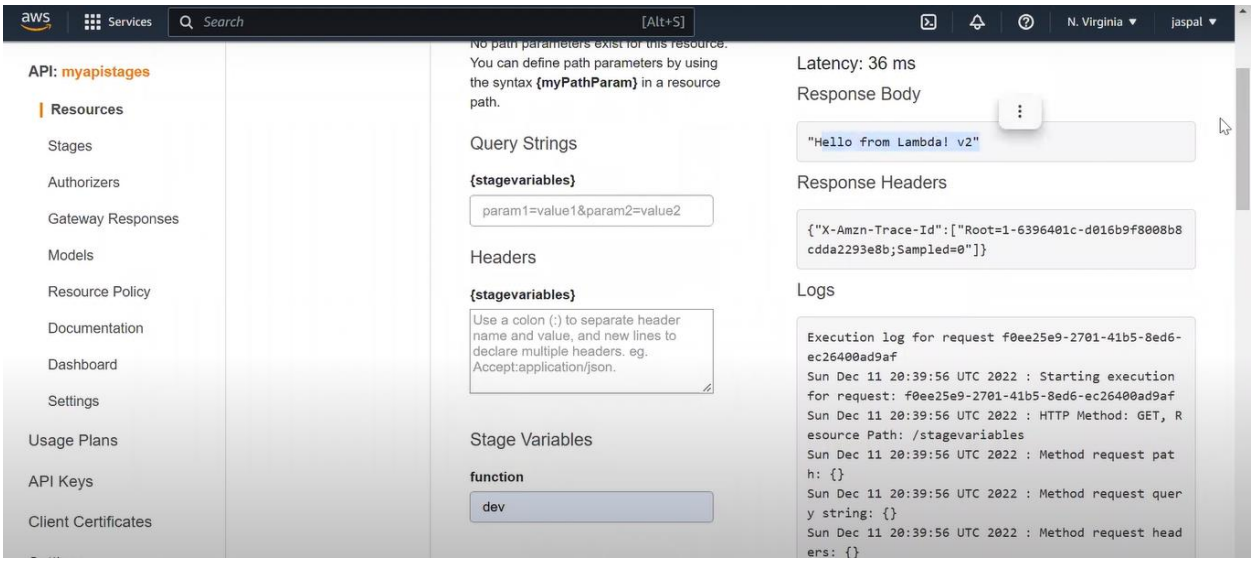
```
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": "lambda:InvokeFunction",
"Resource":
"arn:aws:lambda:YOUR_REGION:YOUR_ACCOUNT_ID:function/YOUR_LAMBDA_FUNCTION_NAME"
}
]
}
```

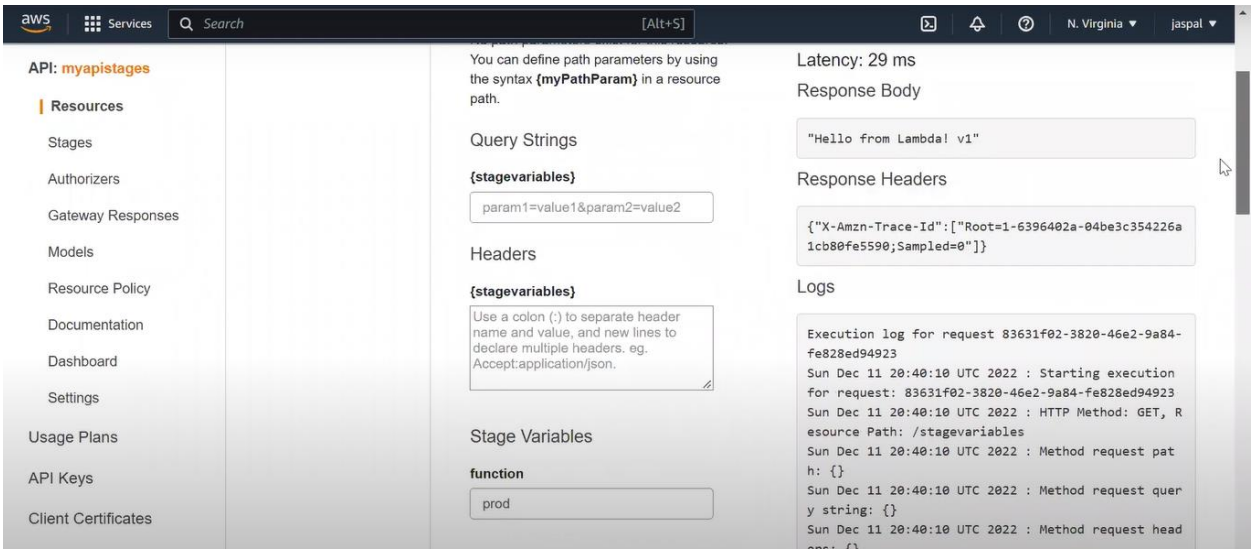
7. Replace `YOUR_REGION`, `YOUR_ACCOUNT_ID`, and `YOUR_LAMBDA_FUNCTION_NAME` with your specific values.
8. Click "Review policy."
9. On the "Review policy" page, give your policy a name (e.g., "APIGatewayInvokeLambdaPolicy").
10. Click "Create policy."
11. Go back to the Lambda integration configuration in the API Gateway console.
12. In the "Edit Integration" page for your Lambda function, click "Save."

Now, your API Gateway has the necessary permissions to invoke your Lambda function. When you deploy your API to a stage and test it, it should be able to execute the Lambda function without any permission issues.

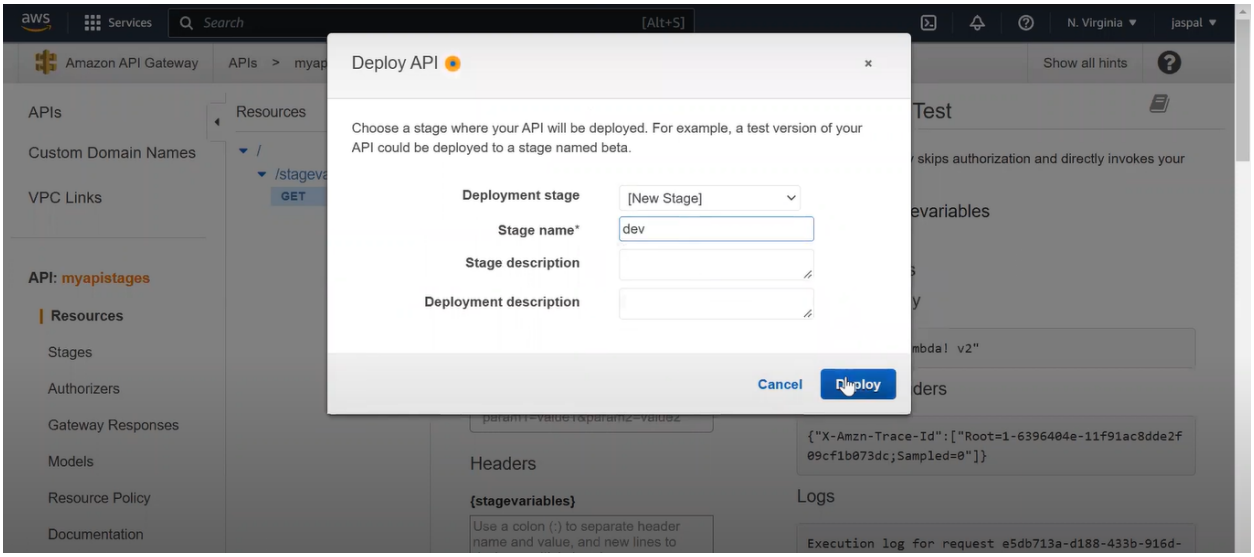

Add Permission to Lambda Function
You defined your Lambda function as a stage variable. Please ensure that you have the appropriate Function Policy on all functions you will use. You can do this by running the below AWS CLI command for each function, replacing the stage variable in the function-name parameter with the necessary function name.
aws lambda add-permission
--function-name "arn:aws:lambda:ap-south-1:195737202275:function:stage_variable_lambda:${stageVariables.function}"
--source-arn "arn:aws:execute-api:ap-south-1:195737202275:jg3qezi2a9/*/GET/stagevariables"
--principal apigateway.amazonaws.com
--statement-id c8db83b2-0756-41c8-8f8a-e4157c82fb41
--action lambda:InvokeFunction
Cancel
OK
Lambda Function
stage_variable_lambda:${stageVariables.function}
Use Default Timeout
Save


/stagevariables - GET - Method Execution
Method Request
Auth: NONE
Integration Request
Type: LAMBDA
Region: ap-south-1
Method Response
HTTP Status: 200
Models: application/json => Empty
Integration Response
HTTP status pattern:
Output passthrough: No
Client
TEST


API: myapistages
Resources
Stages
Authorizers
Gateway Responses
Models
Resource Policy
Documentation
Dashboard
Settings
Usage Plans
API Keys
Client Certificates
Query Strings
{stagevariables}
param1=value1&param2=value2
Headers
{stagevariables}
Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg. Accept:application/json.
Stage Variables
function
dev
Latency: 36 ms
Response Body
"Hello from Lambda! v2"
Response Headers
{"X-Amzn-Trace-Id":["Root=1-6396401c-d016b9f8008b8cdda2293e8b;Sampled=0"]}
Logs
Execution log for request f0ee25e9-2701-41b5-8ed6-ec26400ad9af
Sun Dec 11 20:39:56 UTC 2022 : Starting execution for request: f0ee25e9-2701-41b5-8ed6-ec26400ad9af
Sun Dec 11 20:39:56 UTC 2022 : HTTP Method: GET, Resource Path: /stagevariables
Sun Dec 11 20:39:56 UTC 2022 : Method request path: {}
Sun Dec 11 20:39:56 UTC 2022 : Method request query string: {}
Sun Dec 11 20:39:56 UTC 2022 : Method request headers: {}

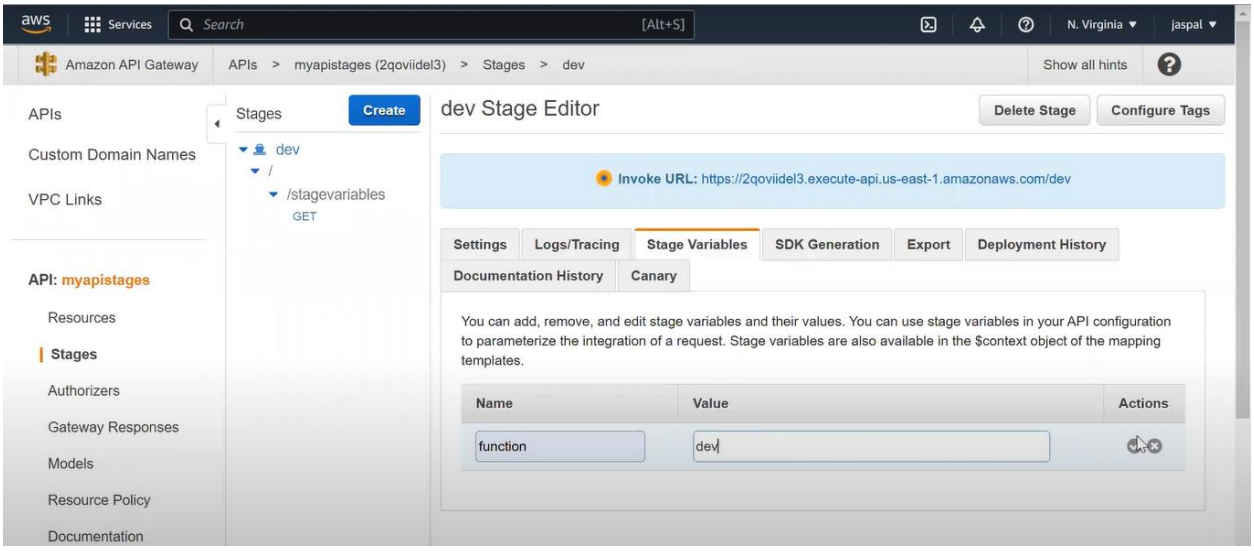


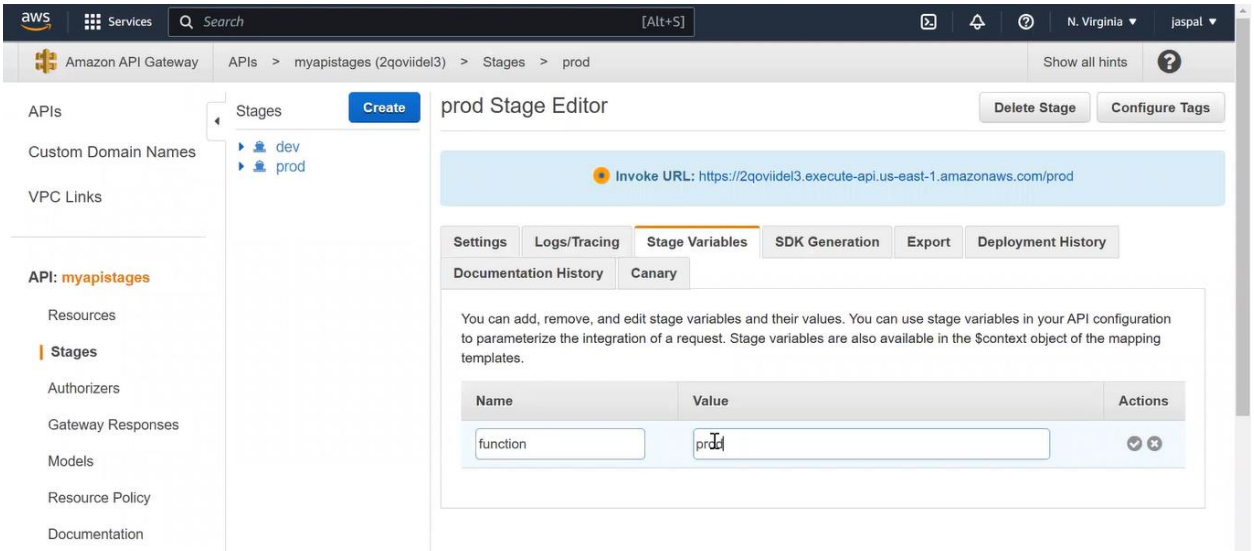
STEP 7: Deploy your API to a new stage (e.g., "prod" ,"dev", "test").



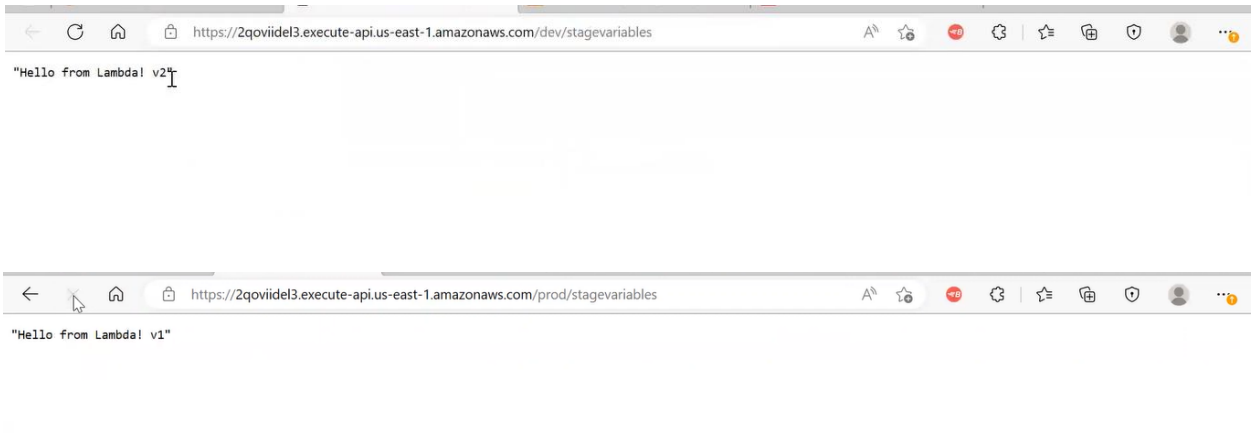
STEP 8: Create a Stage Variable

1. In the API Gateway console, navigate to your API and select the stage (e.g., "prod" or "test").
2. Scroll down to the "Stage variables" section.
3. Click "Add stage variable."
4. Enter a variable name (e.g., “function”) and specify the value as the alias name you created ("dev” or “prod”)
5. Click "Save changes."

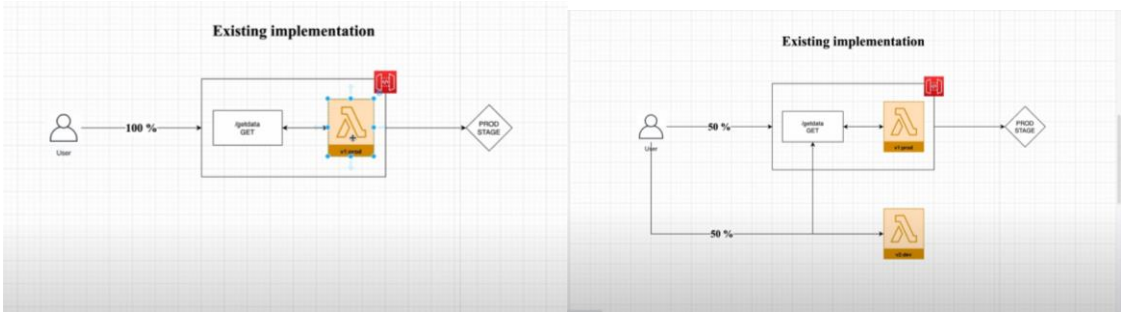




STEP 8: Test the invoke URL

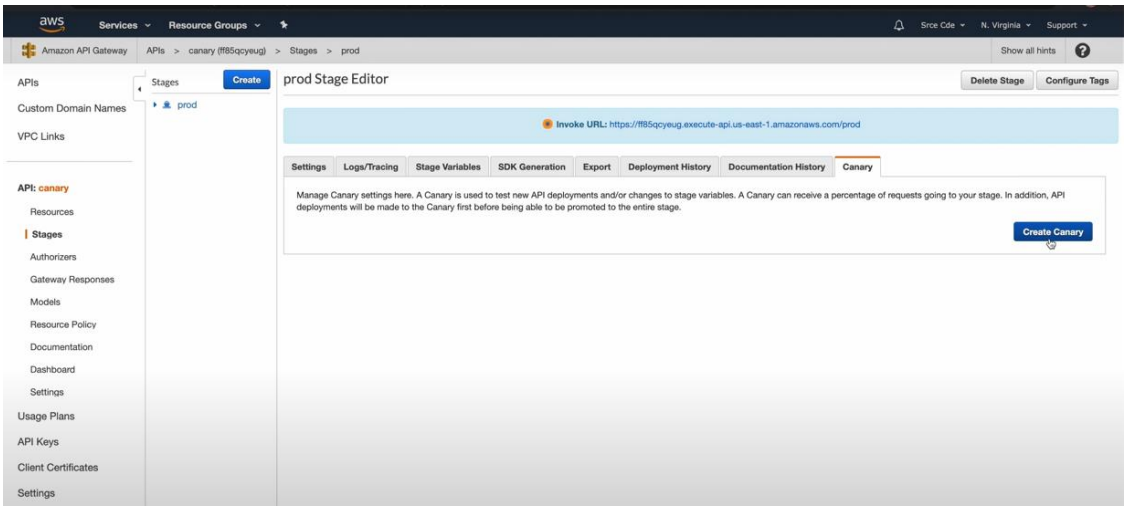


5.CANARY DEPLOYMENT

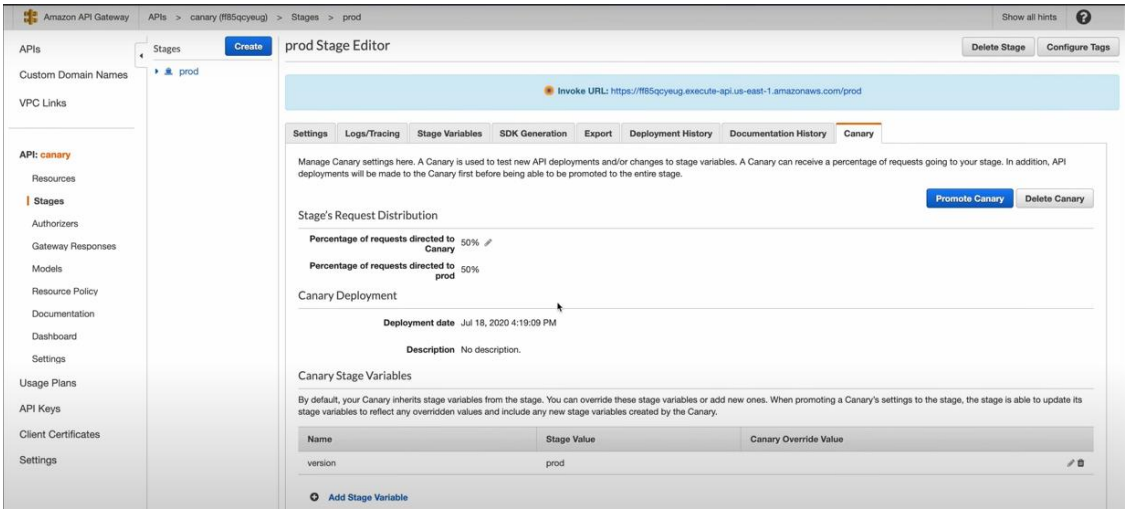


STEP 1: Under the "Stages" section, select the "production" stage.

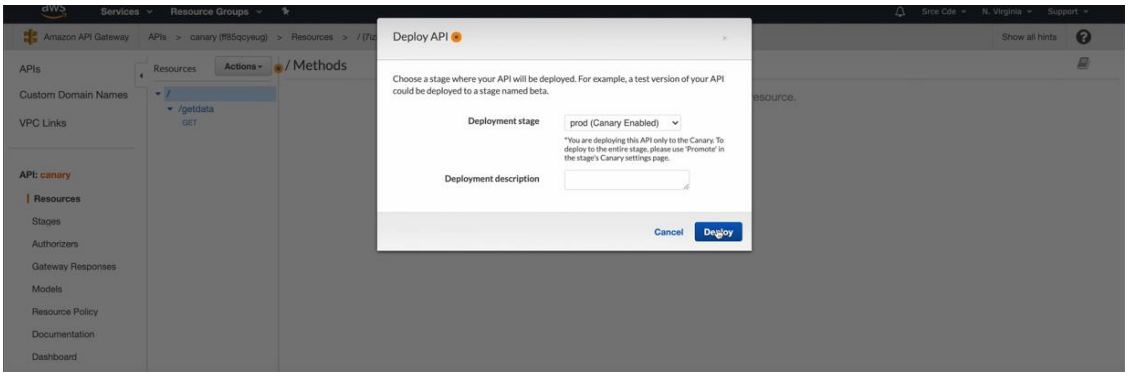
STEP 2: Click Create Canary



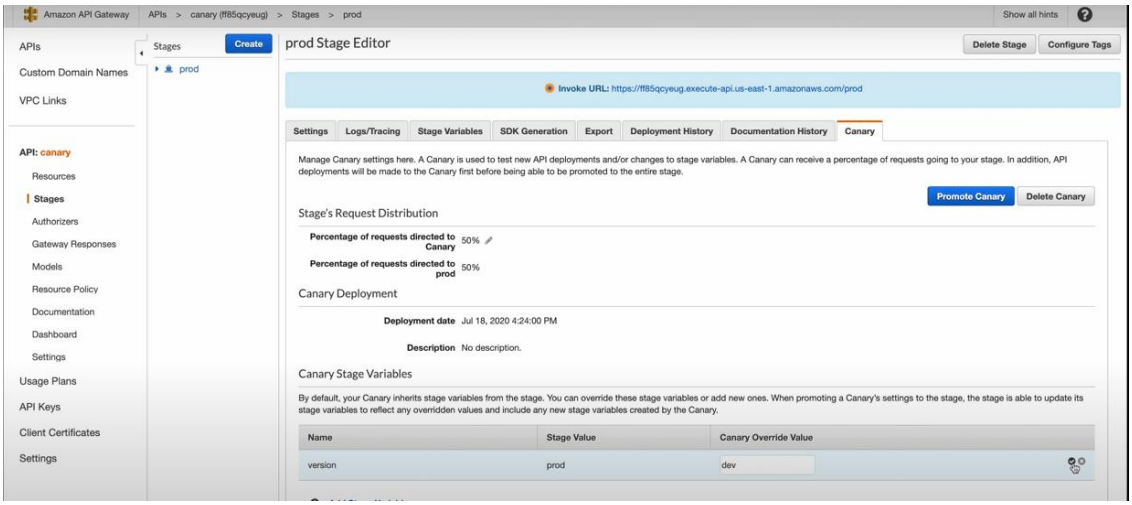
STEP 3: Configure the request distribution to allocate a percentage of traffic to the canary stage.



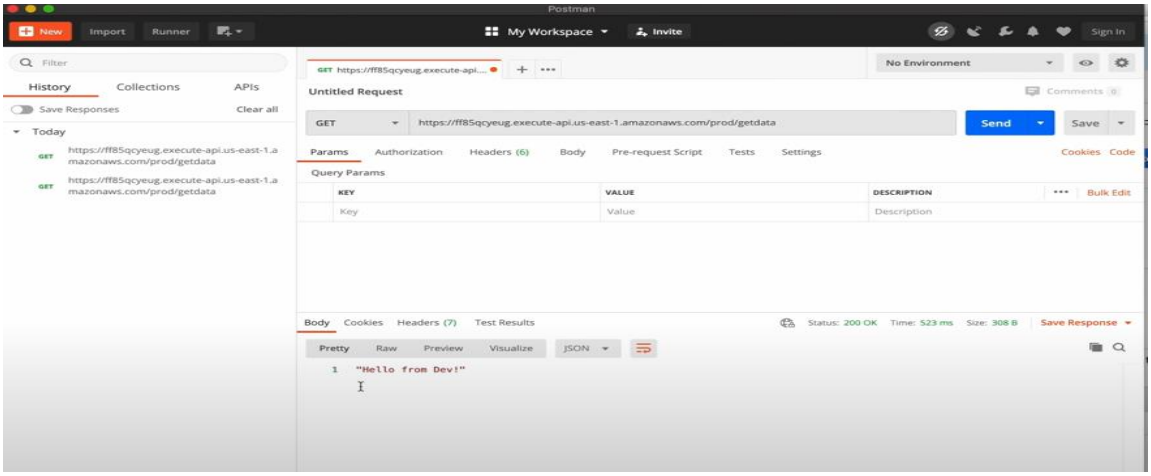
STEP 4: Deploy API Stage as prod(Canary Enabled)

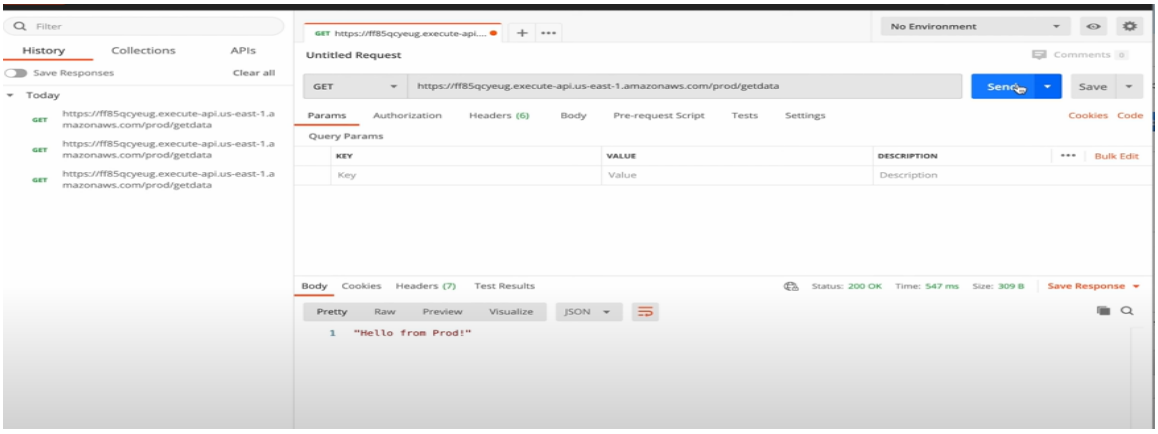


STEP 5: Configure the canary override value

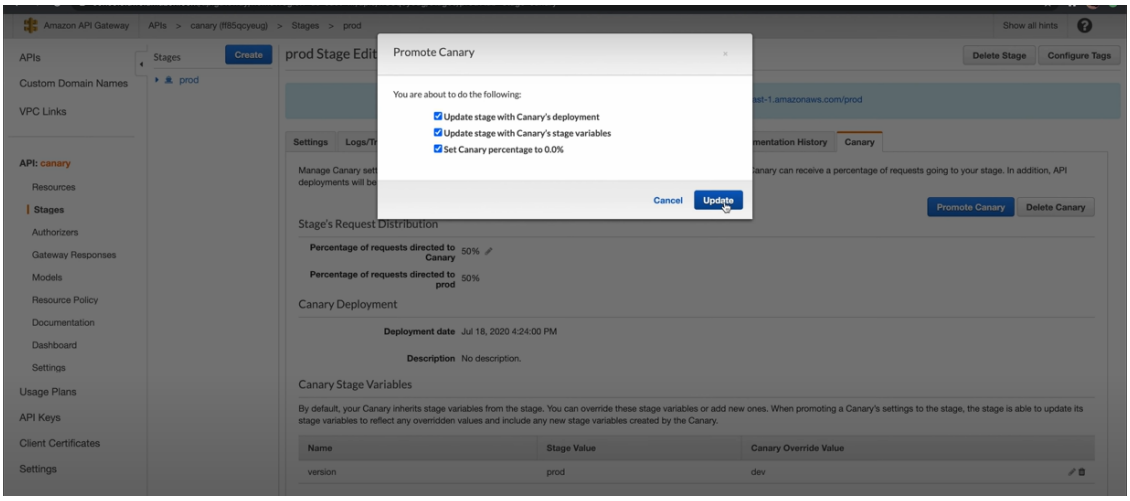


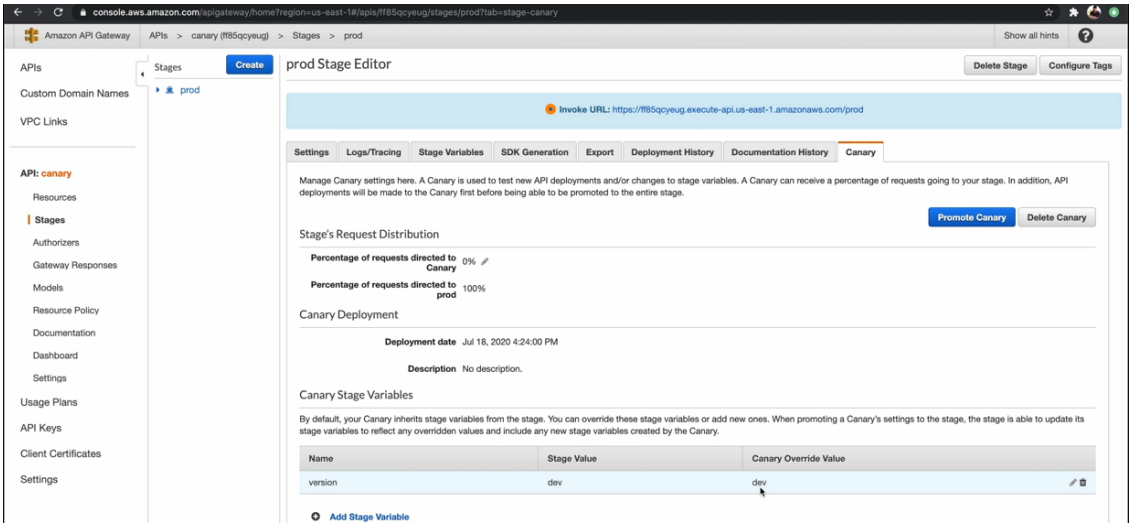
STEP 6: Test the invoke url

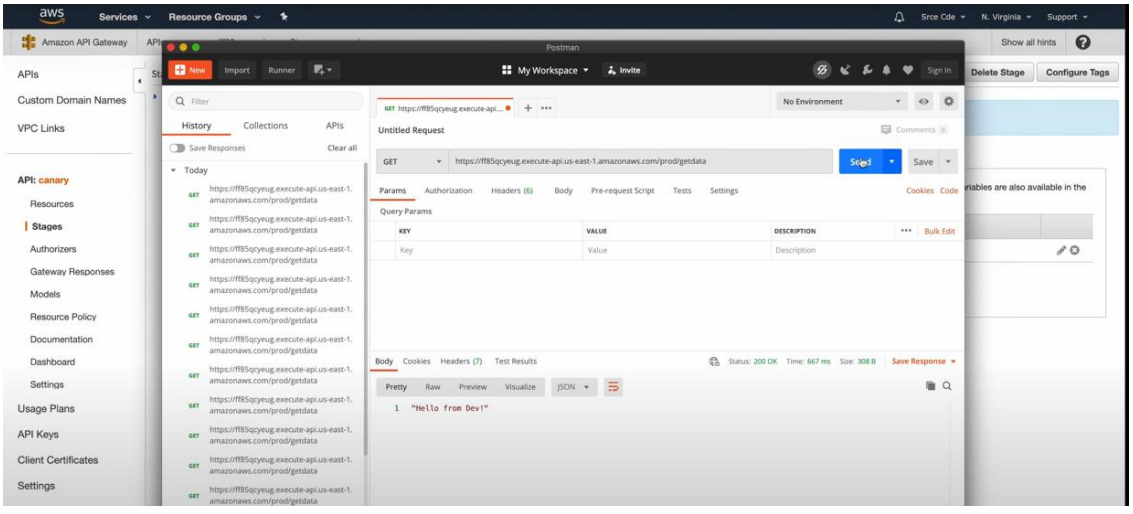




STEP 7: Promote the canary, then it will update all the changes







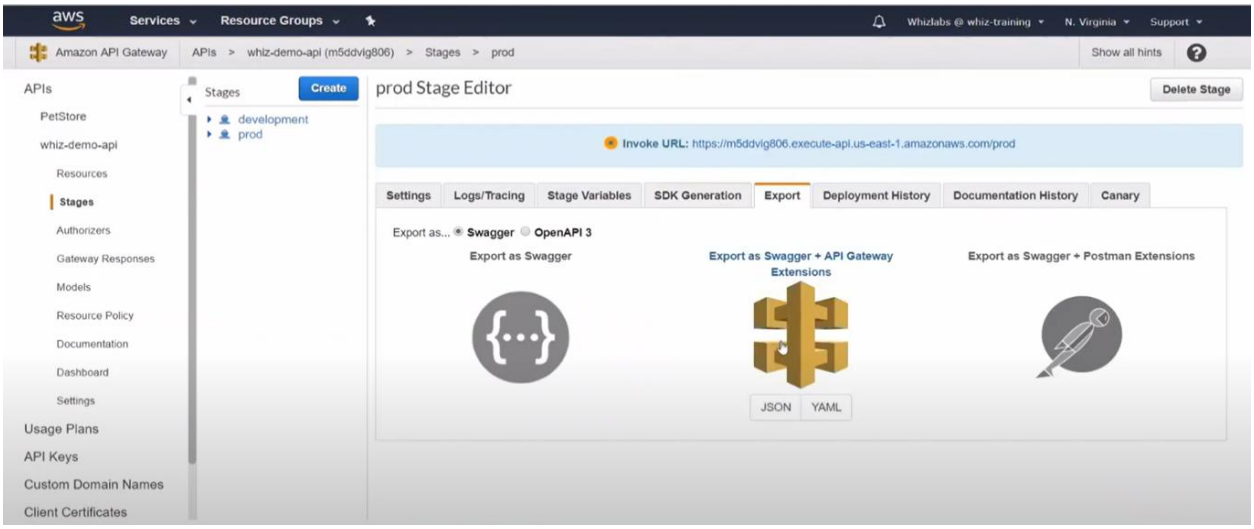
6.EXPORT:

Choose the Export Format

In the export dialog, you'll be prompted to choose the export format. AWS API Gateway supports both Swagger (Open API 2.0) and Open API (Open API 3.0). Select the format you prefer:

Swagger (Open API 2.0): If you choose this option, you'll get a Swagger 2.0-compliant JSON representation of your API definition.

Open API (Open API 3.0): If you choose this option, you'll get an Open API 3.0-compliant JSON representation of your API definition.

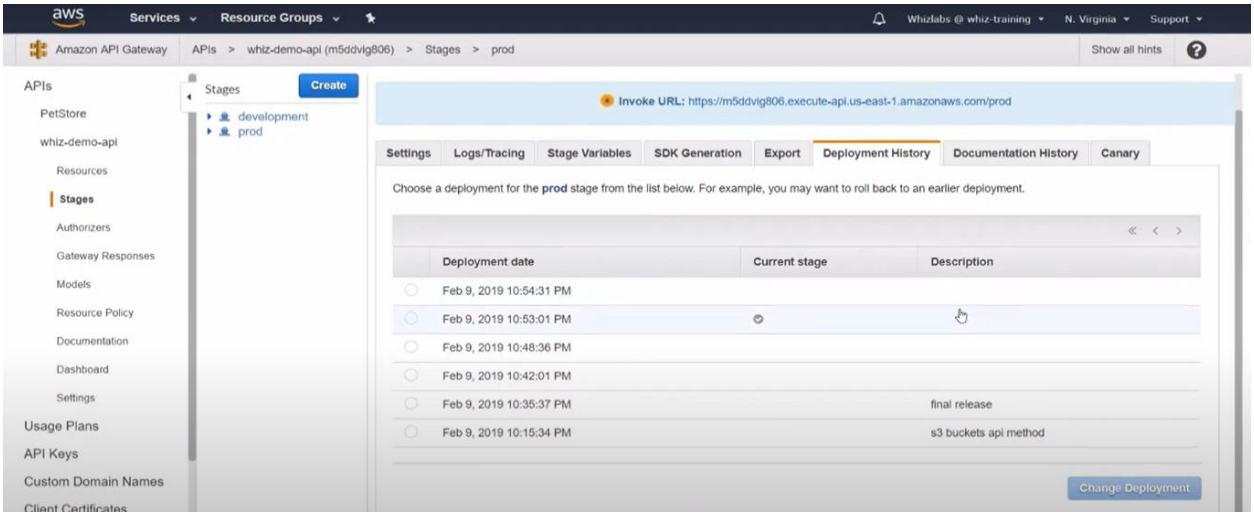


7.DEPLOYMENT HISTORY

In the "Deployment History" section, you will see a list of deployments for the selected stage. Each deployment entry typically includes the following information:

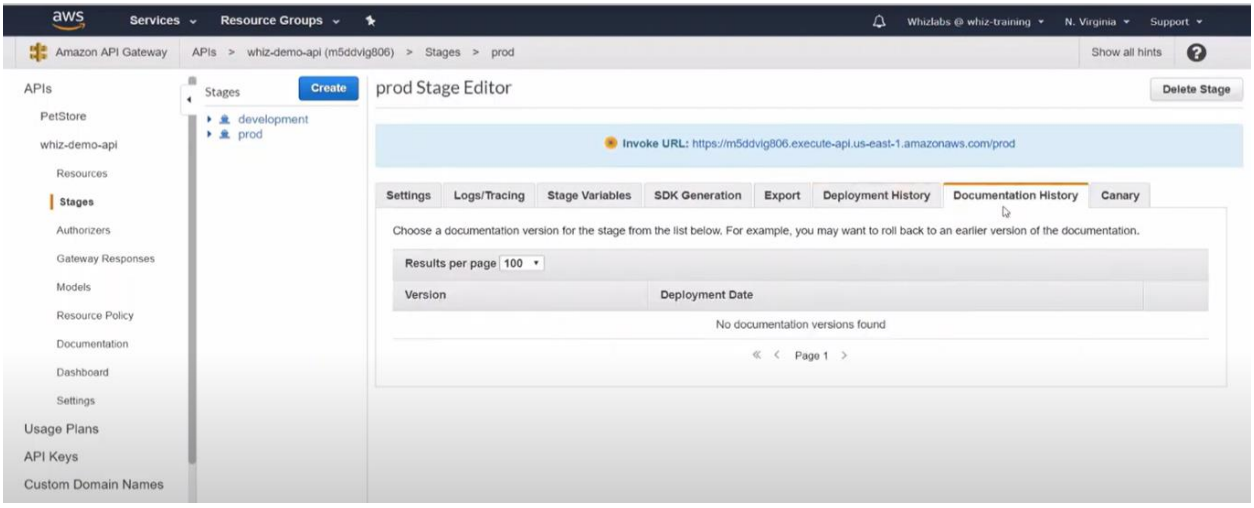
- **Deployment ID:** A unique identifier for the deployment.
- **Stage:** The name of the stage to which the deployment was made (e.g., "prod,""dev").
- **Deployment Date:** The date and time when the deployment was initiated.

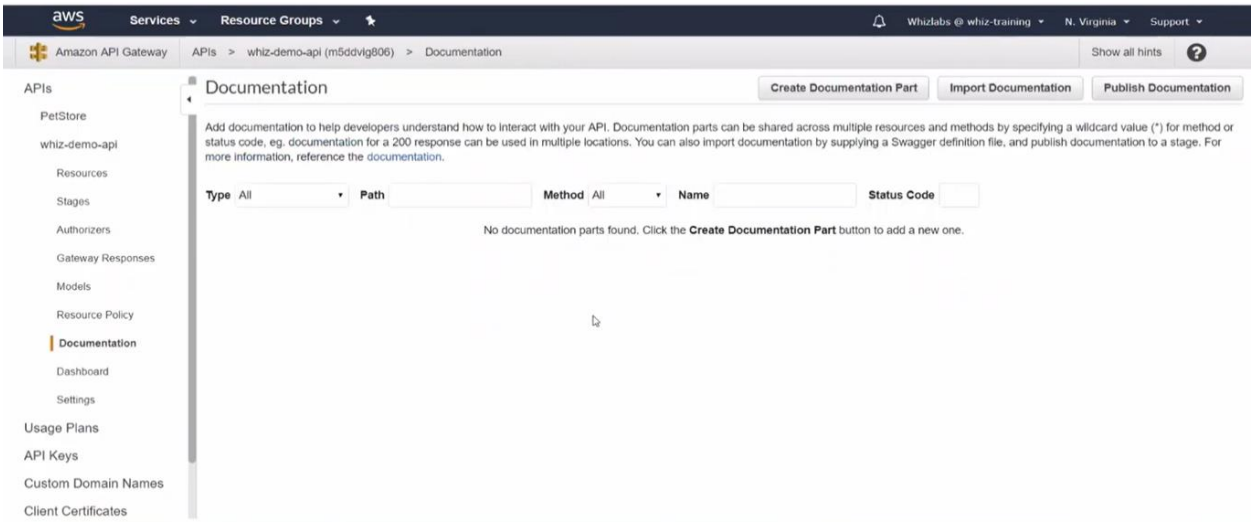
- **Description:** An optional description provided when the deployment was made. This can be useful for adding notes or context to deployments.



8.DOCUMENTATION HISTORY

In the "Documentation history" section, you can view the history of your API's documentation including:





CUSTOM DOMAIN NAME

A custom domain name in API Gateway refers to a user-defined domain name that you can associate with your API endpoints instead of using the default domain name provided by API Gateway. This allows you to have a more branded and user-friendly URL for your API.

Here's how it works:

1. Default Domain: When you create an API in API Gateway, it's assigned a default domain name like `randomstring.execute-api.region.amazonaws.com`.

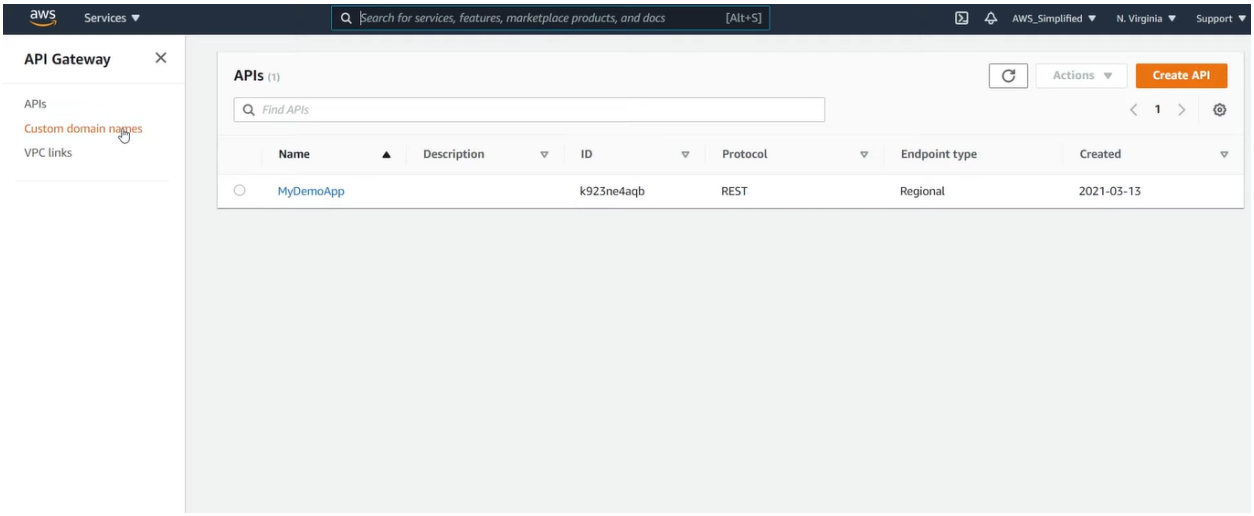
2. Custom Domain Name: With a custom domain name, you can map your API to a domain that you own, like `api.example.com`. This makes it easier for clients to access your API.

3. SSL/TLS: You can also set up SSL/TLS certificates for your custom domain to ensure secure communication between clients and your API.

4. Path Mapping: Custom domain names can also be used to map different paths to different APIs or stages within your API, allowing you to create a more organized API structure.

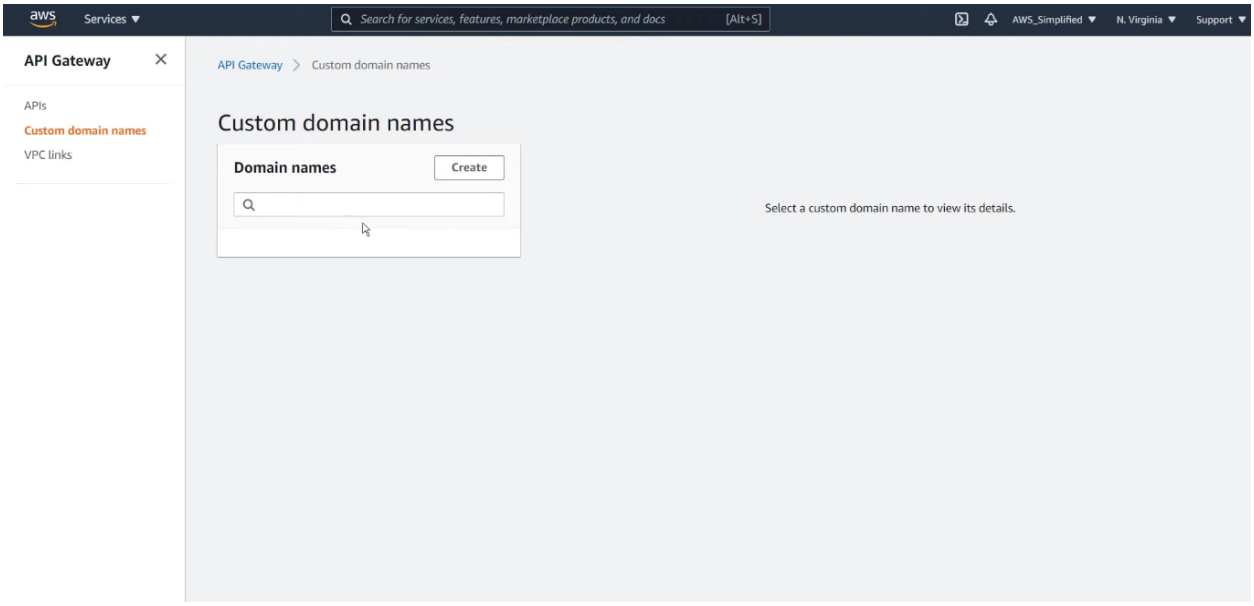
STEPS TO CREATE CUSTOM DOMAIN NAME

STEP 1: From the API Gateway dashboard, select the API for which you want to create a custom domain name.

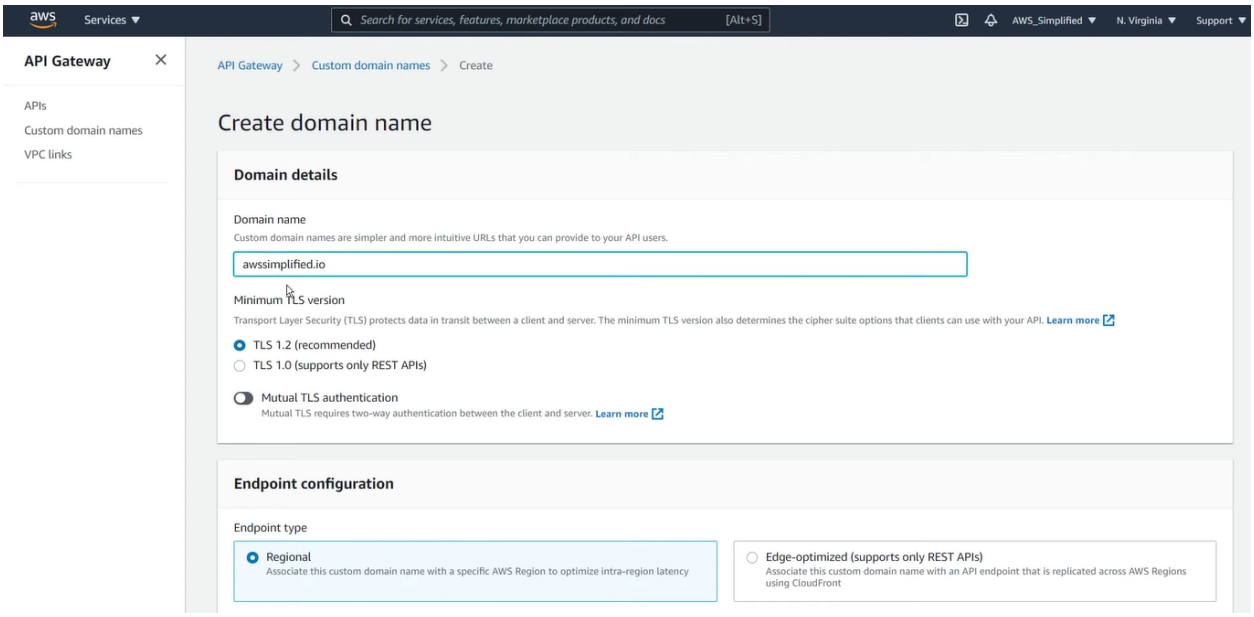


STEP 2: In the left navigation pane, under the selected API, click on "Custom domain names."

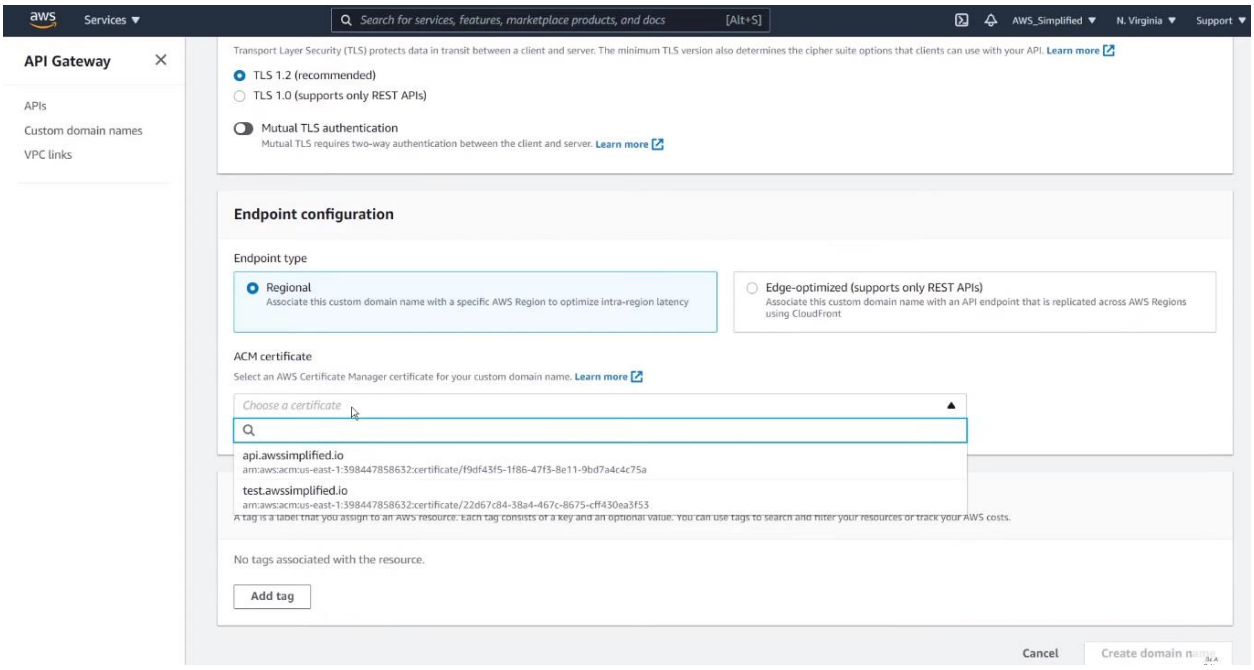
STEP 3: Click the "Create" button to start the process of creating a custom domain name.



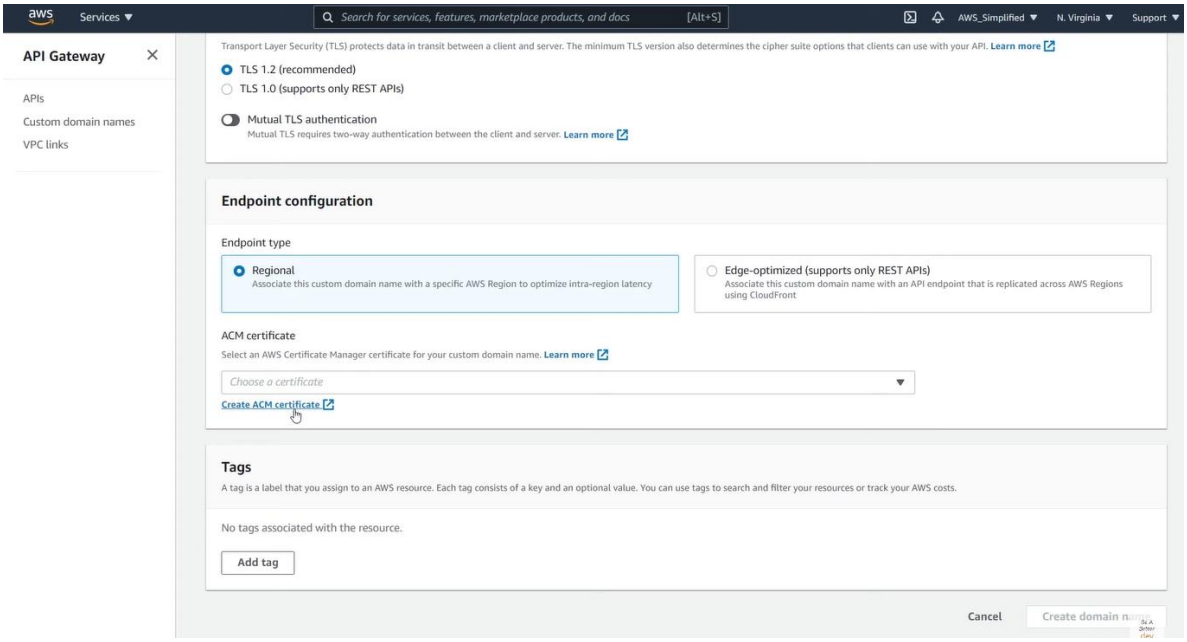
STEP 4: Enter the domain name you want to use (e.g., api.example.com).



STEP 5: Choose whether you want an edge-optimized or regional endpoint for your custom domain. The choice depends on your requirements and location.

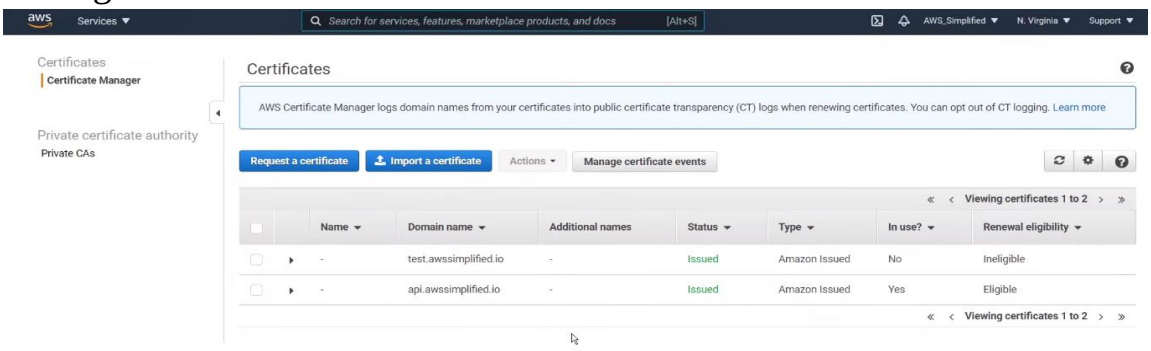


STEP 6: TLS(Transport Layer Security)/SSL(Secure Sockets Layer) certificate: Choose an existing ACM (AWS Certificate Manager) certificate or request a new one if you haven't already. Ensure that the certificate covers the domain name you're using.

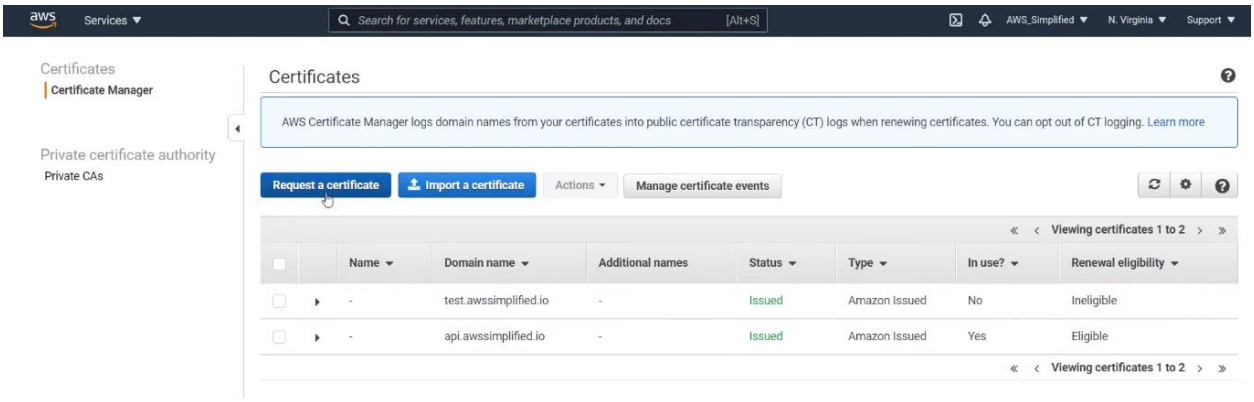


STEPS TO CREATE ACM CERTIFICATE

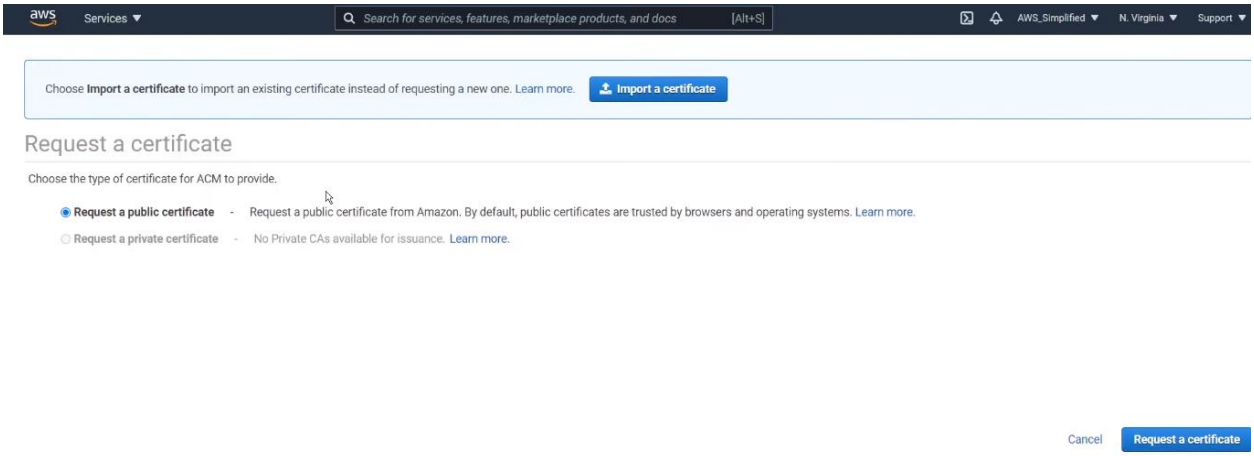
STEP 1: In the AWS Management Console, navigate to the "Services" menu. Under the "Security, Identity, & Compliance" section, select "Certificate Manager."



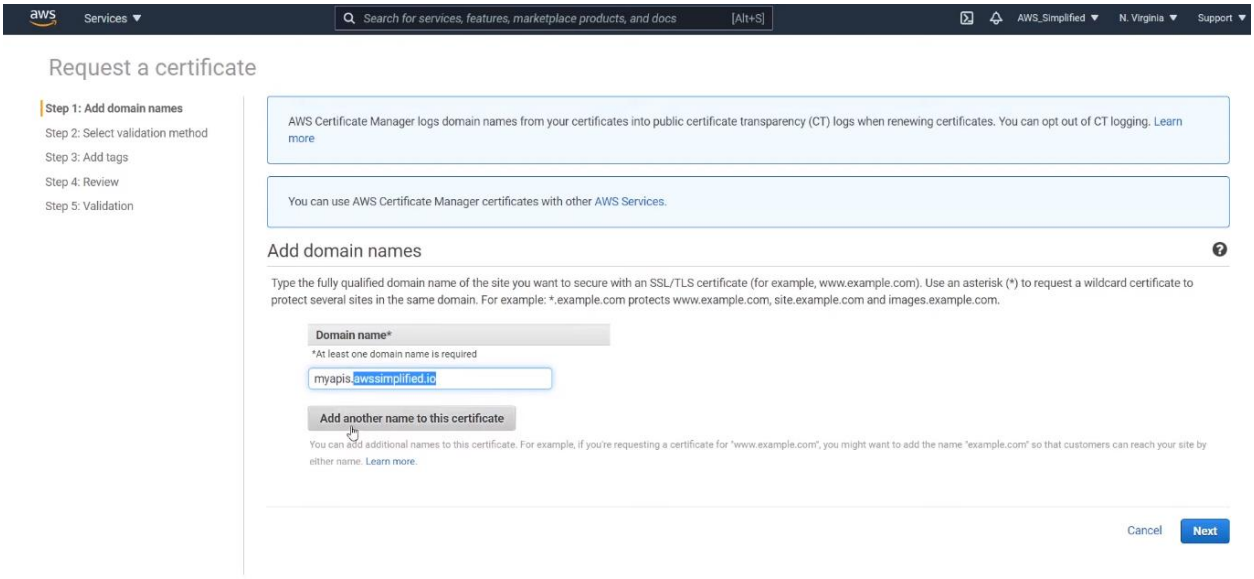
STEP 2: In the ACM dashboard, click the "Request a certificate" button.



STEP 3: Select "Request a public certificate" or "Request a private certificate" based on your use case.

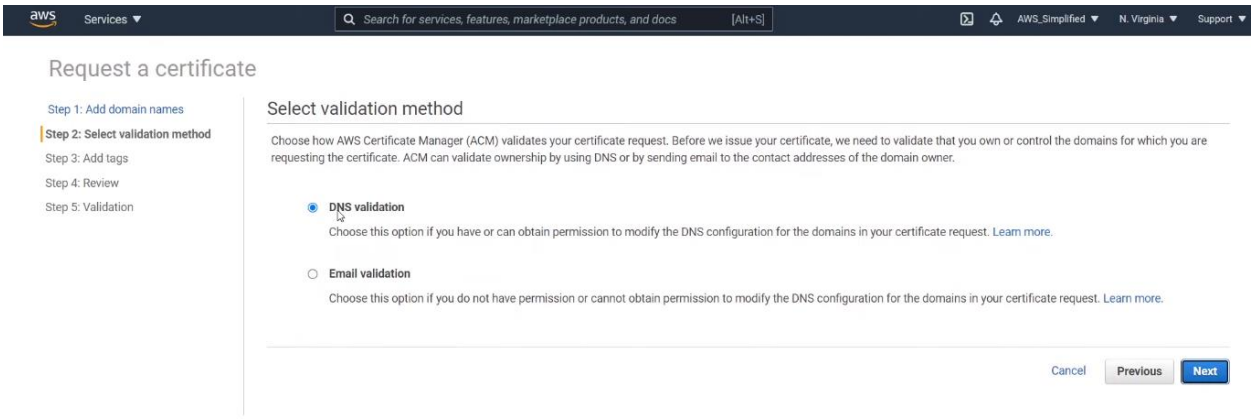


STEP 4: Enter the domain names you want to secure with the certificate (e.g., www.example.com, api.example.com). You can add multiple domain names, including wildcard certificates (e.g., *.example.com).

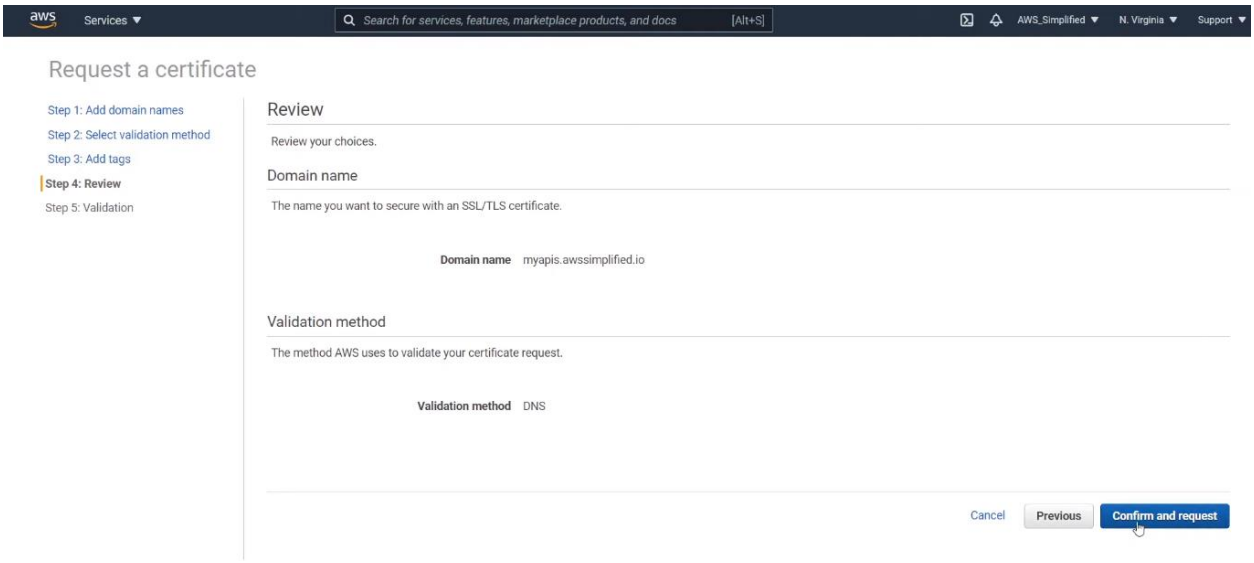


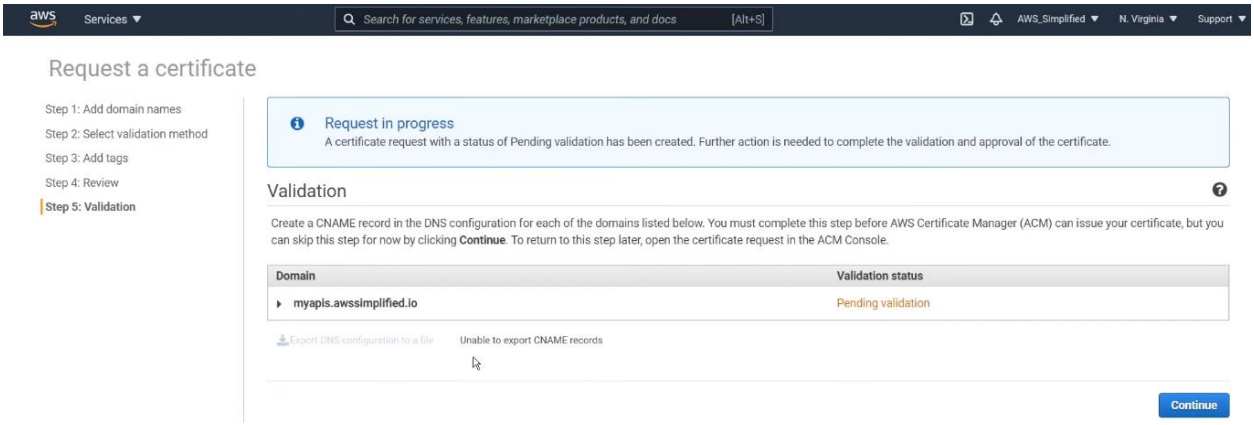
STEP 5: Choose how you want to validate ownership of the domain(s):

1. DNS validation: ACM provides you with CNAME(Canonical Name) records to add to your DNS configuration.
2. Email validation: ACM sends validation emails to the domain owner's registered email addresses.

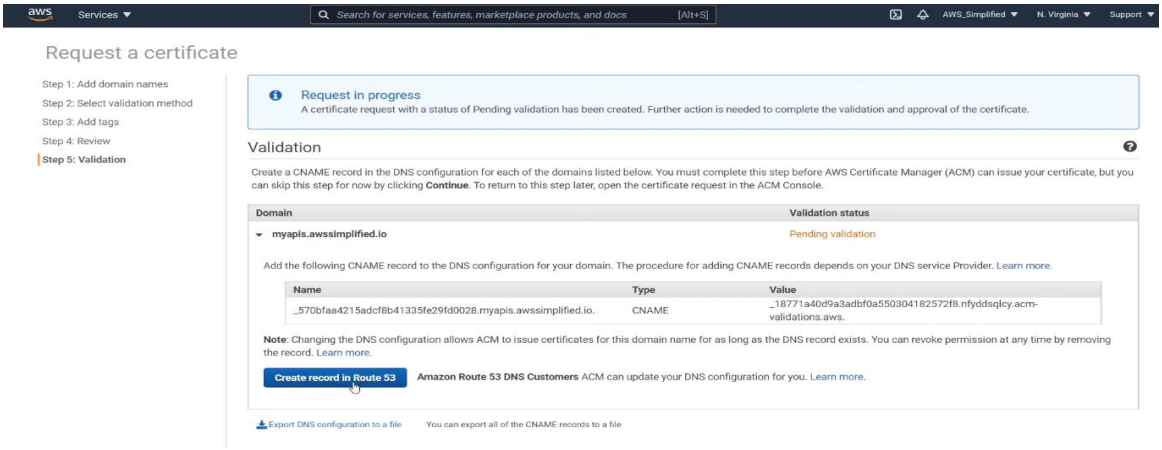


STEP 6: Review your certificate request details to ensure accuracy and Confirm the request.

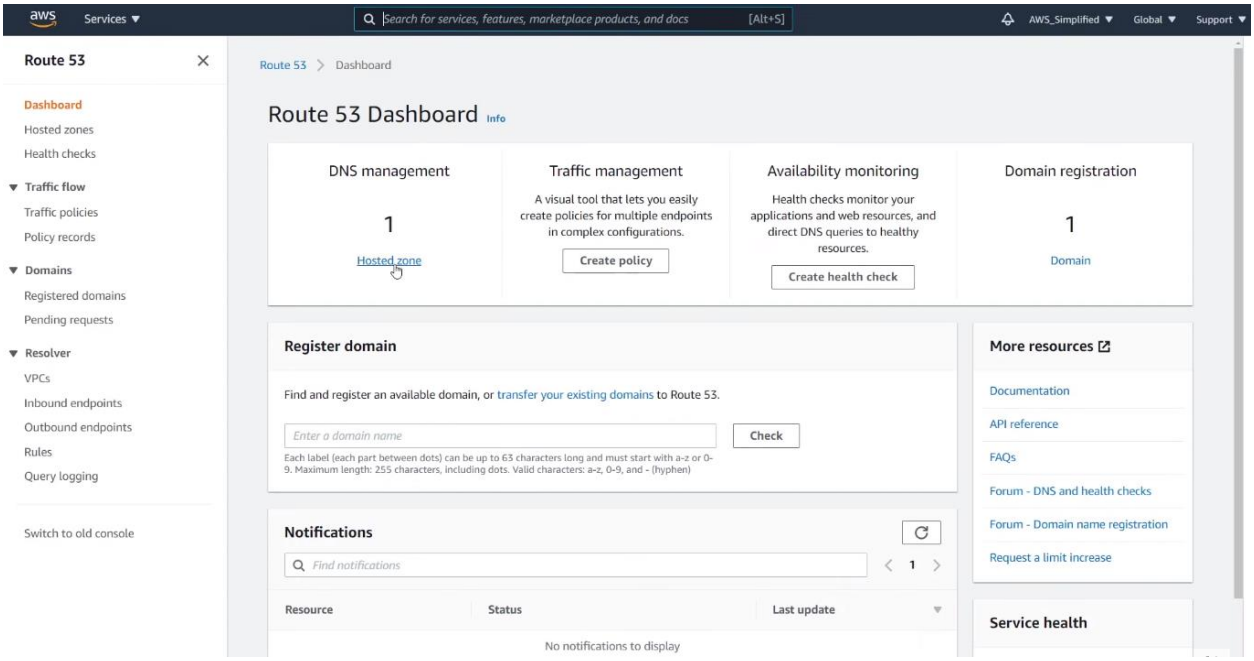




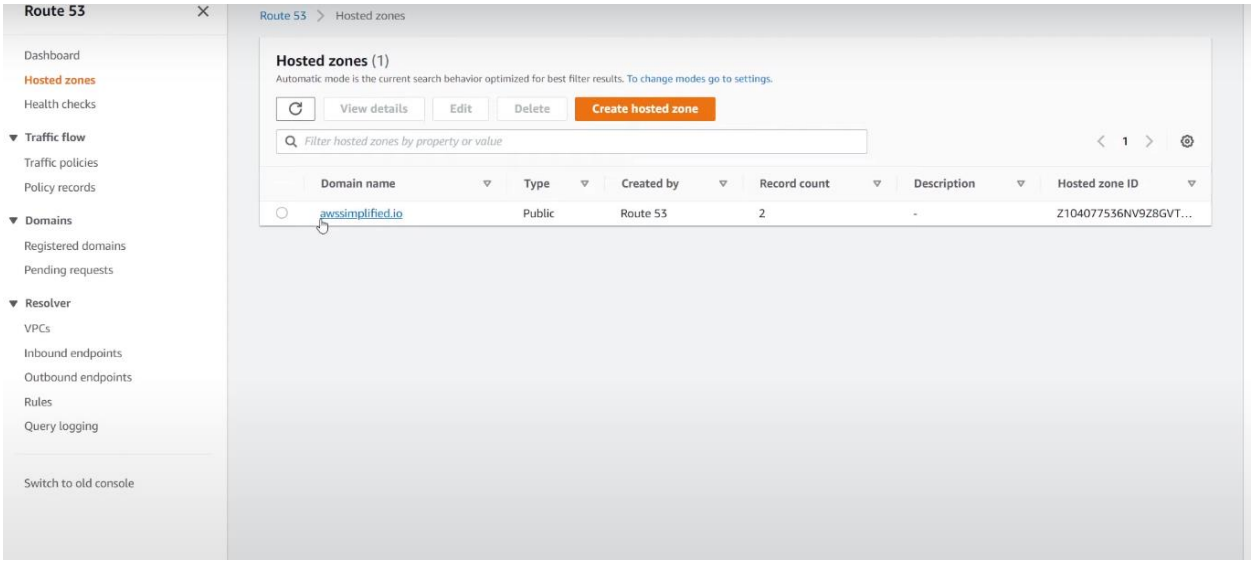
STEP 7: If you chose DNS validation, add the provided CNAME records to your DNS configuration.



STEP 8: In the Route 53 dashboard, choose the hosted zone where you want to create a DNS record.

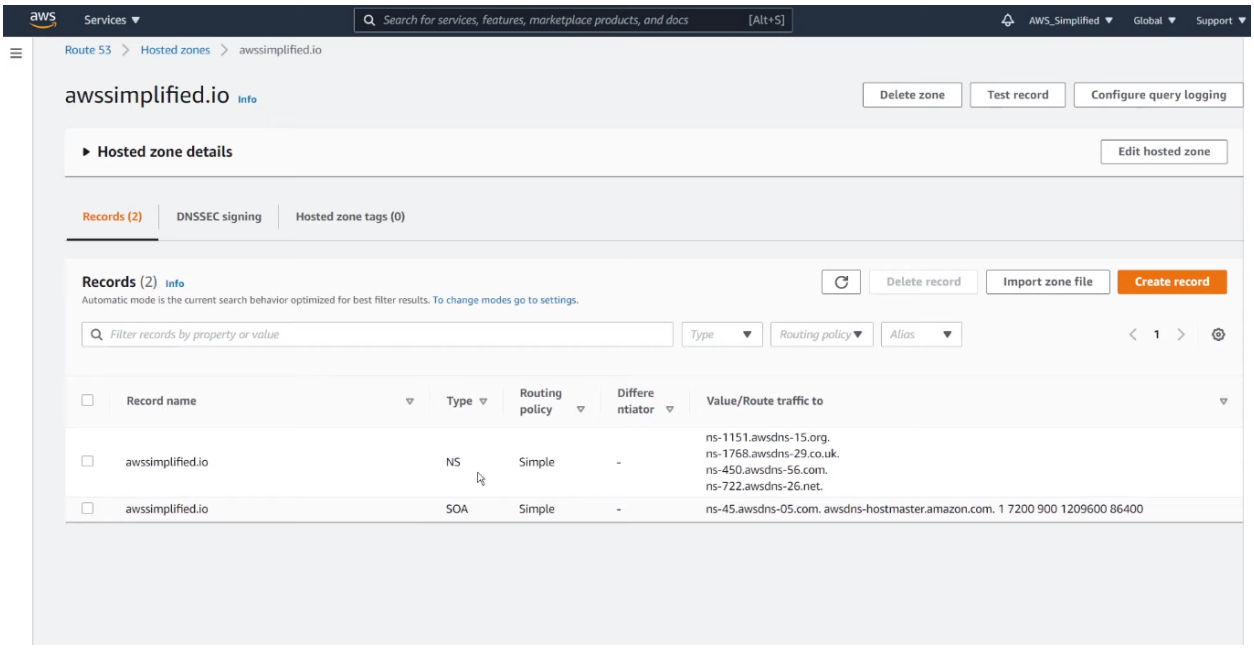


STEP 9: Click on the hosted zone's name to access its settings

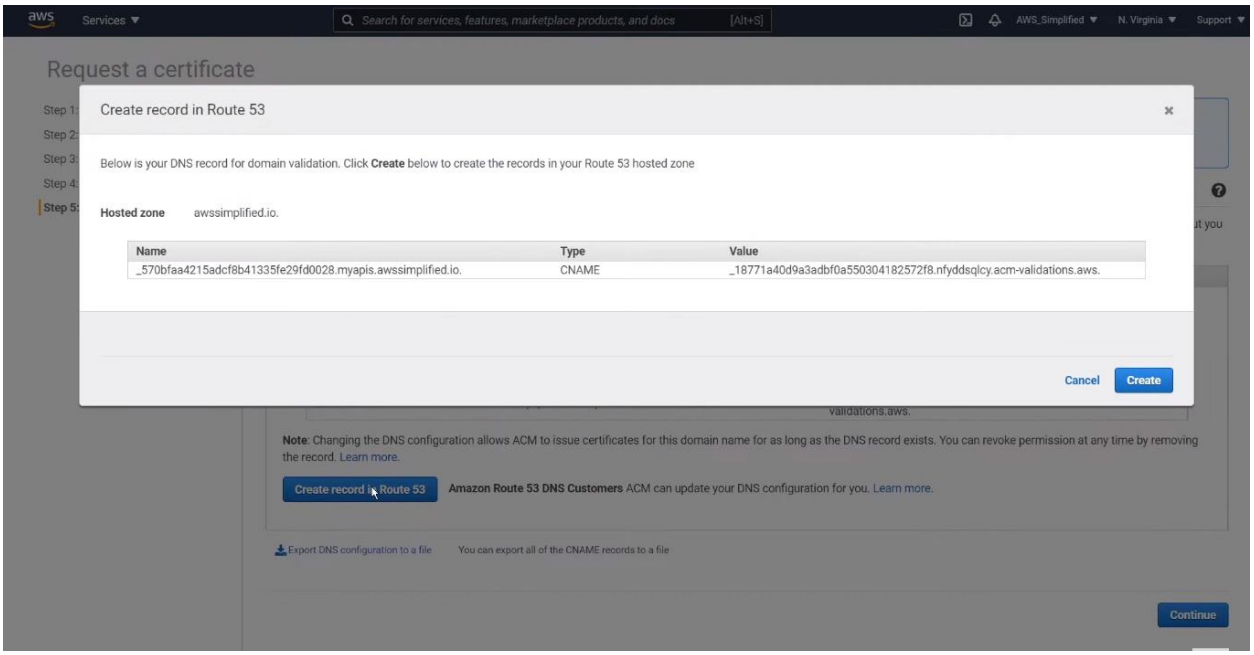


STEP 10: In the hosted zone details page, you'll see a list of existing DNS records (if any).

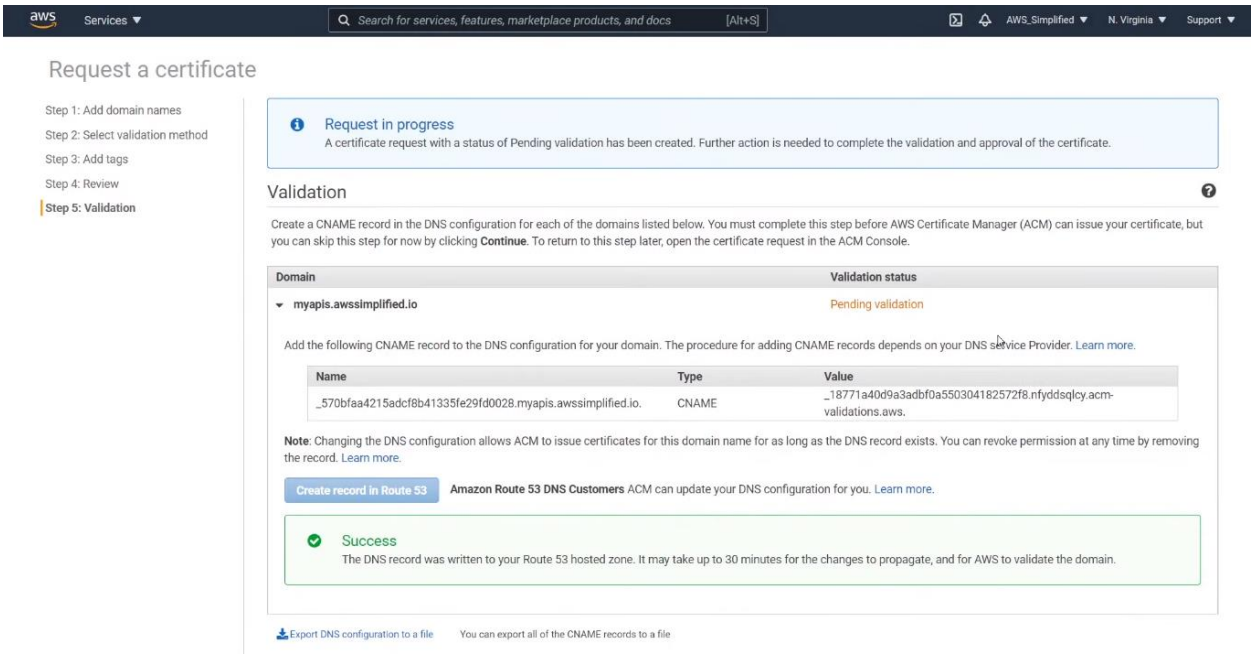
STEP 11: To create a new DNS record, click the "Create Record Set" button.



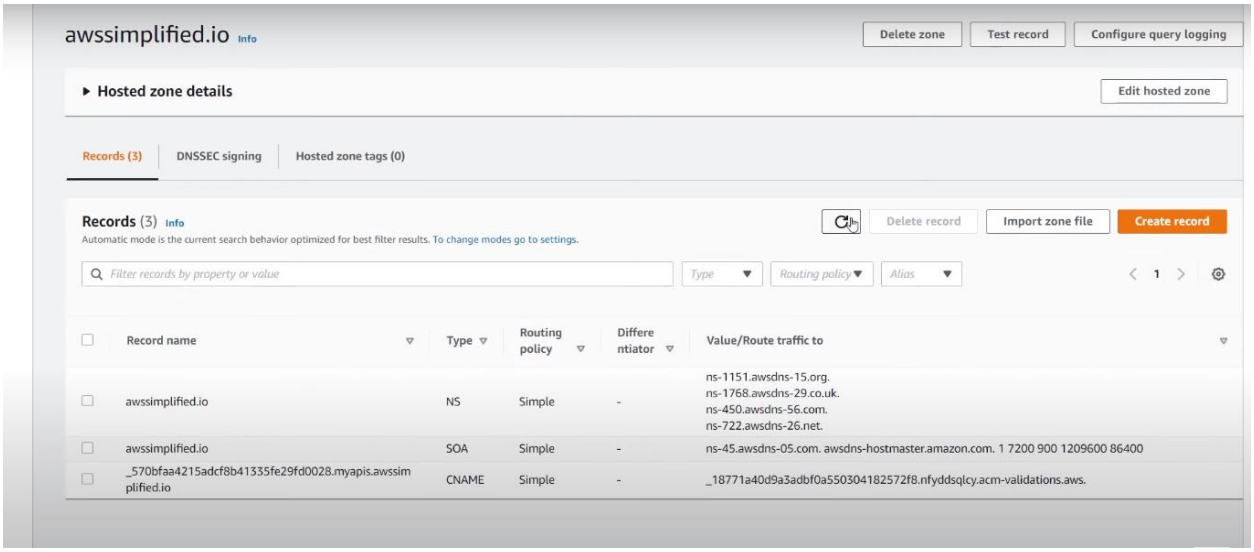
STEP 12: After configuring the record set, click the "Create" or "Save Record Set" button to create the DNS record.

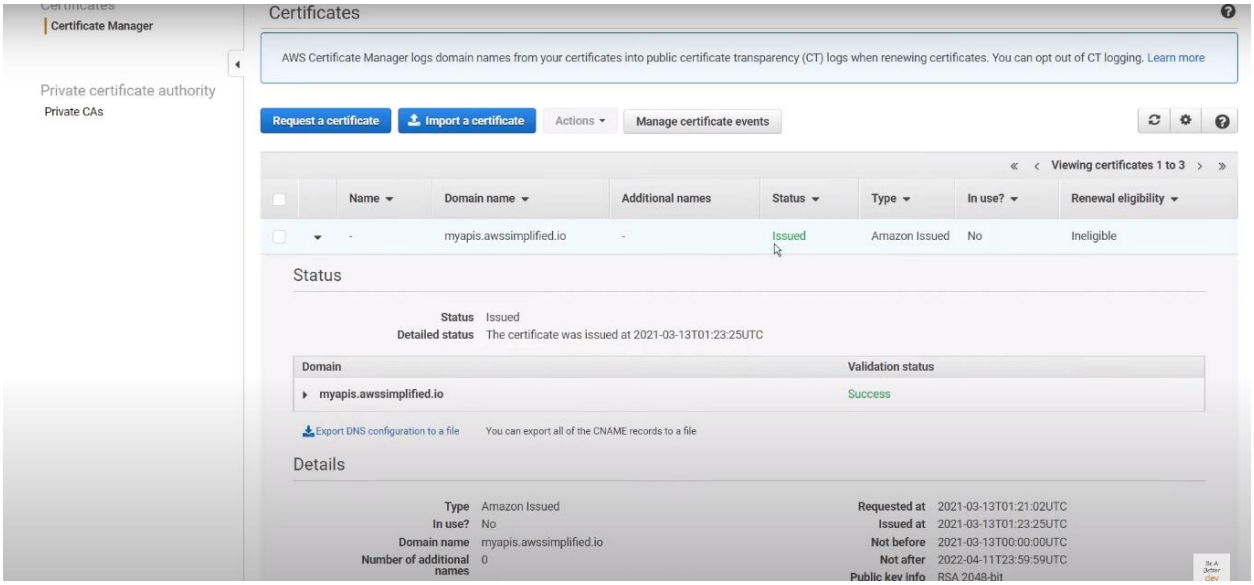


STEP 13: Once validation is successful, ACM will approve your certificate request.



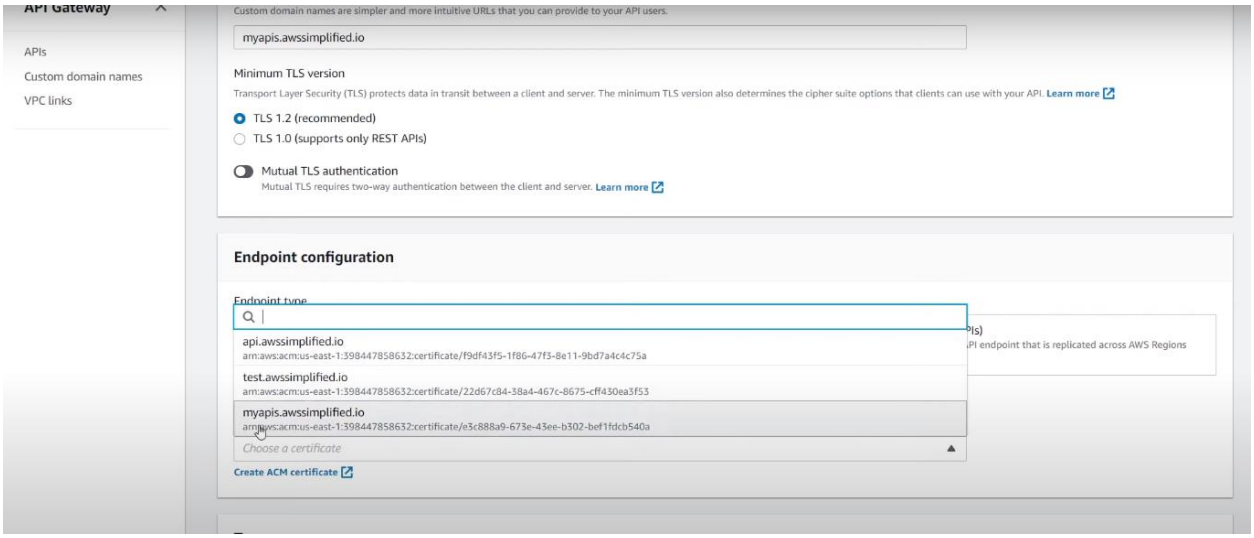
Now the record is added,



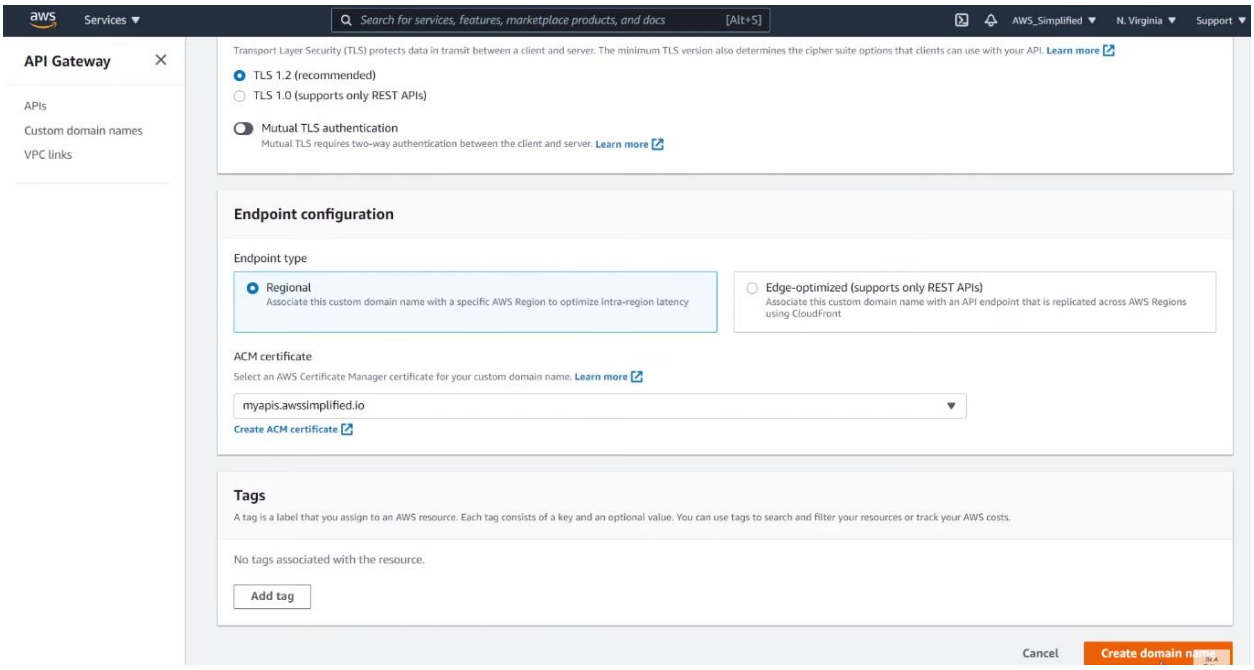


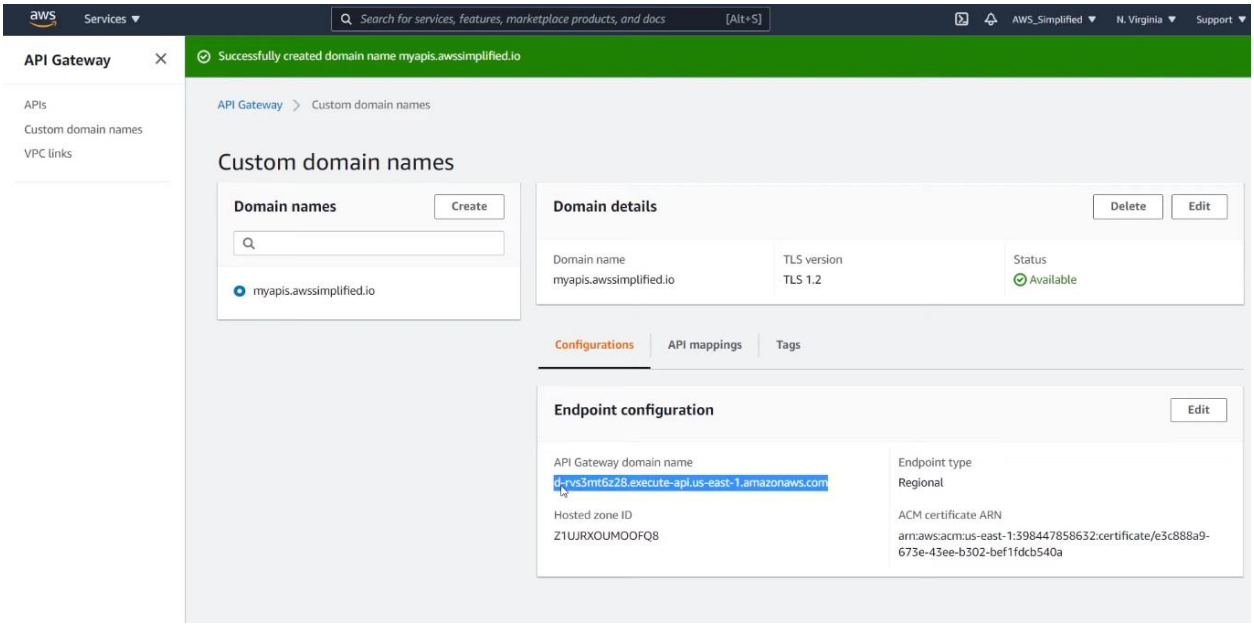
ADD ACM TO CUSTOM DOMAIN NAME

STEP 1: After approval, your certificate is ready to use.

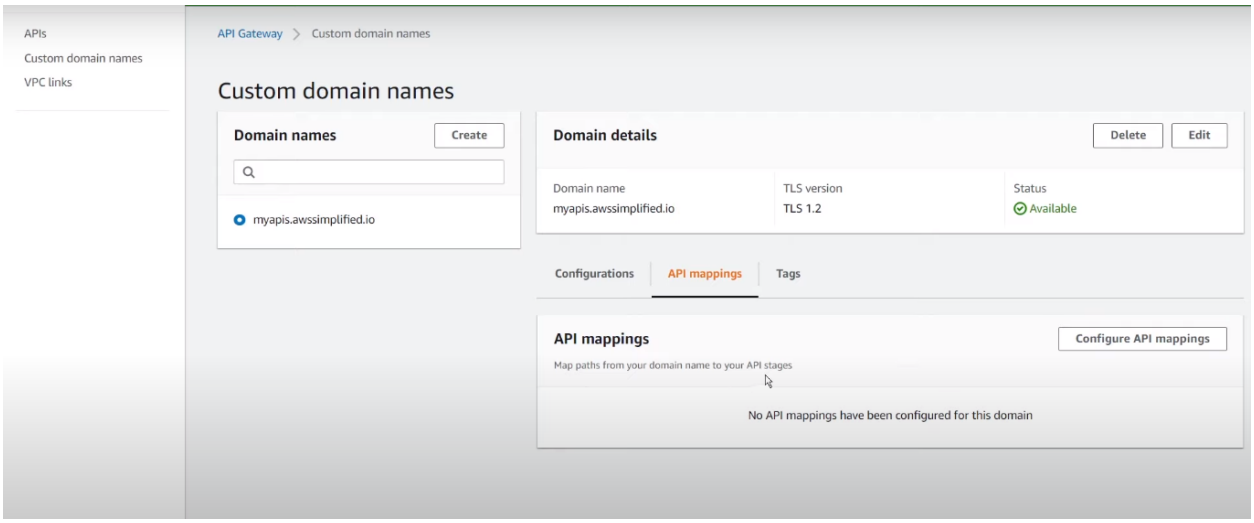


STEP 2: After configuring the settings, click the "Create Domain Name" button.

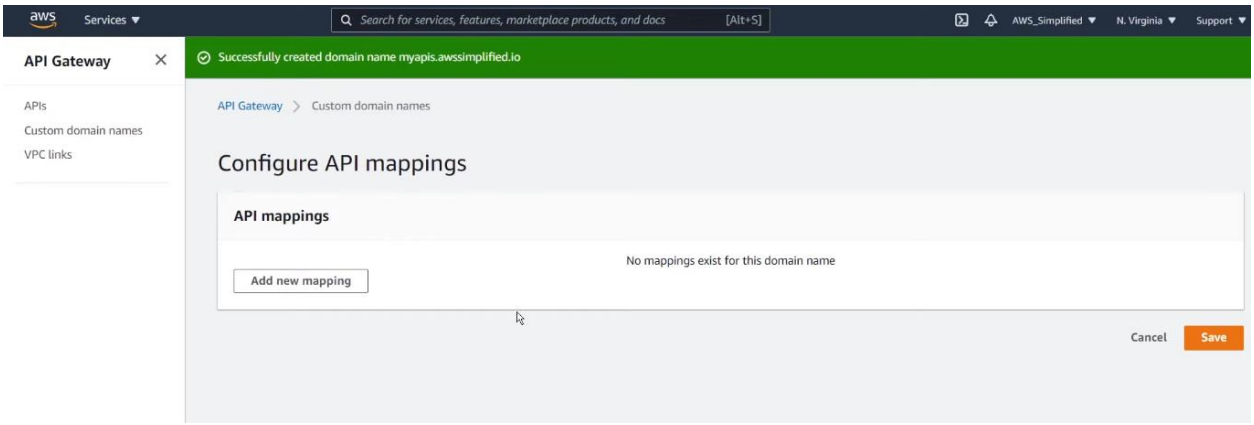




STEP 3: Click configure API Mapping.



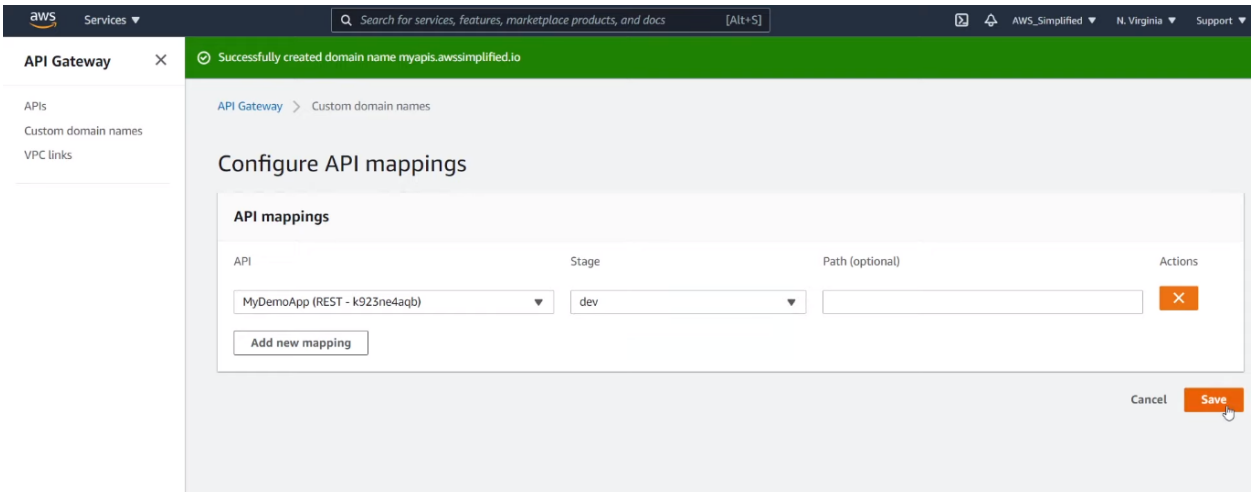
STEP 4: Click Add new mapping

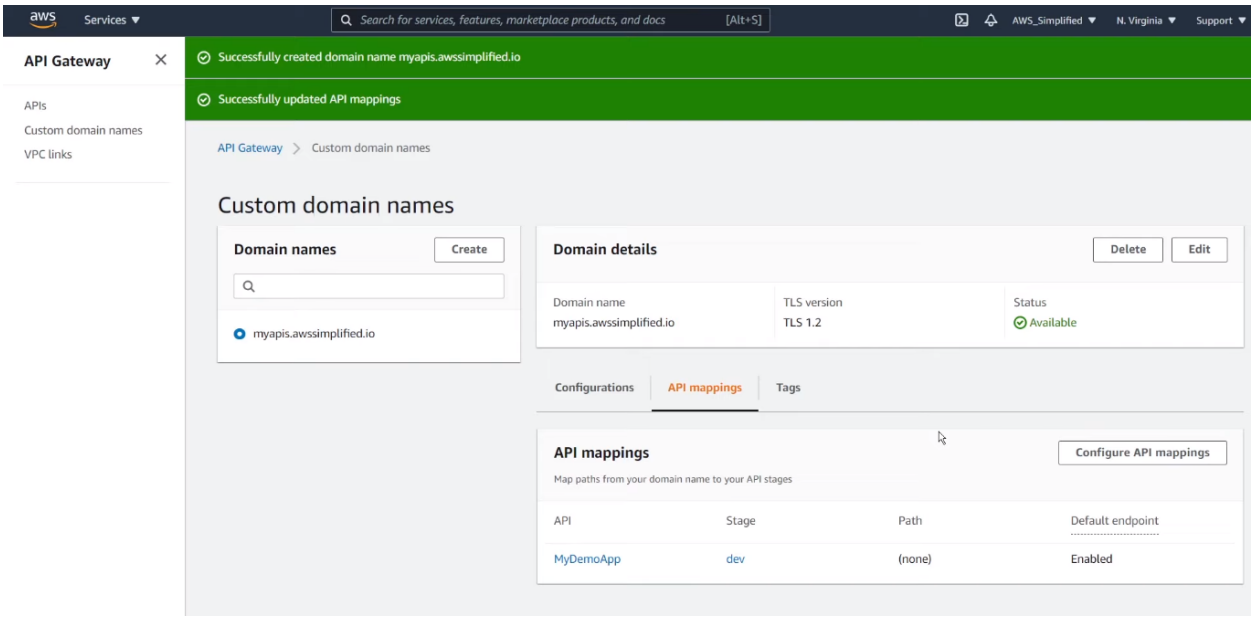


STEP 5:

- Select the API you want to map.
- Select the stage of the API (e.g., "prod," "dev," etc.) that you want to associate with this custom domain.
- Set the Base Path (e.g., "/" for the root domain or "/v1" for a specific version of your API).

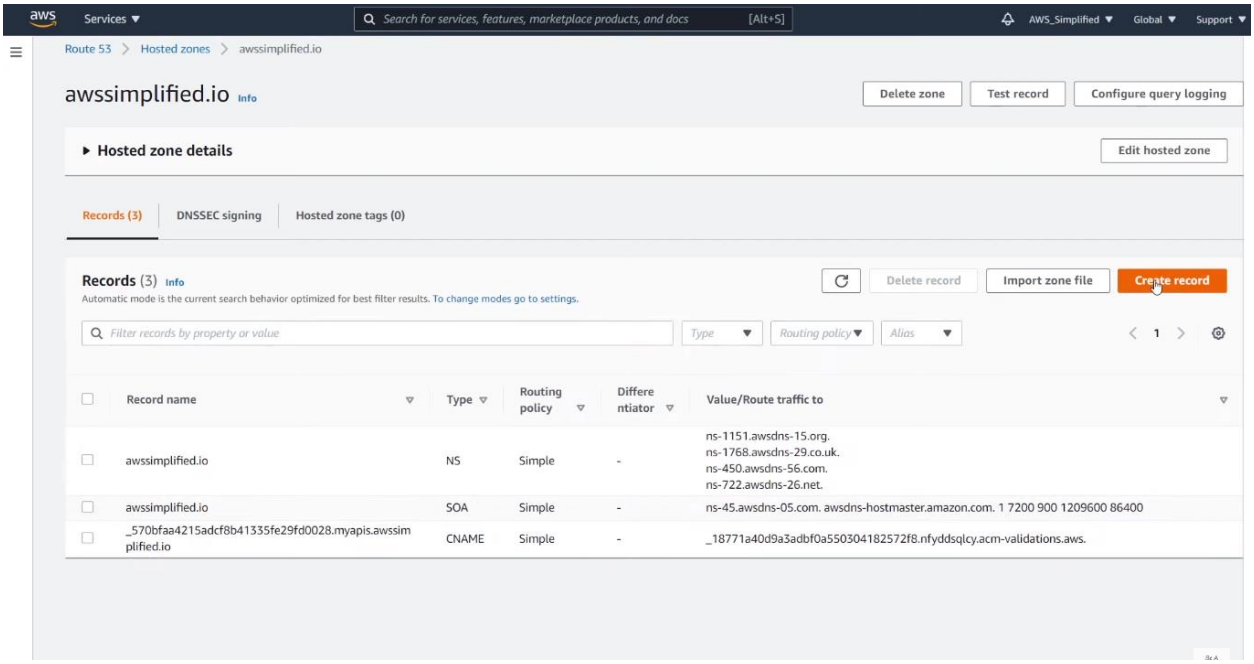
STEP 6: After configuring the mapping details, click the "Save"

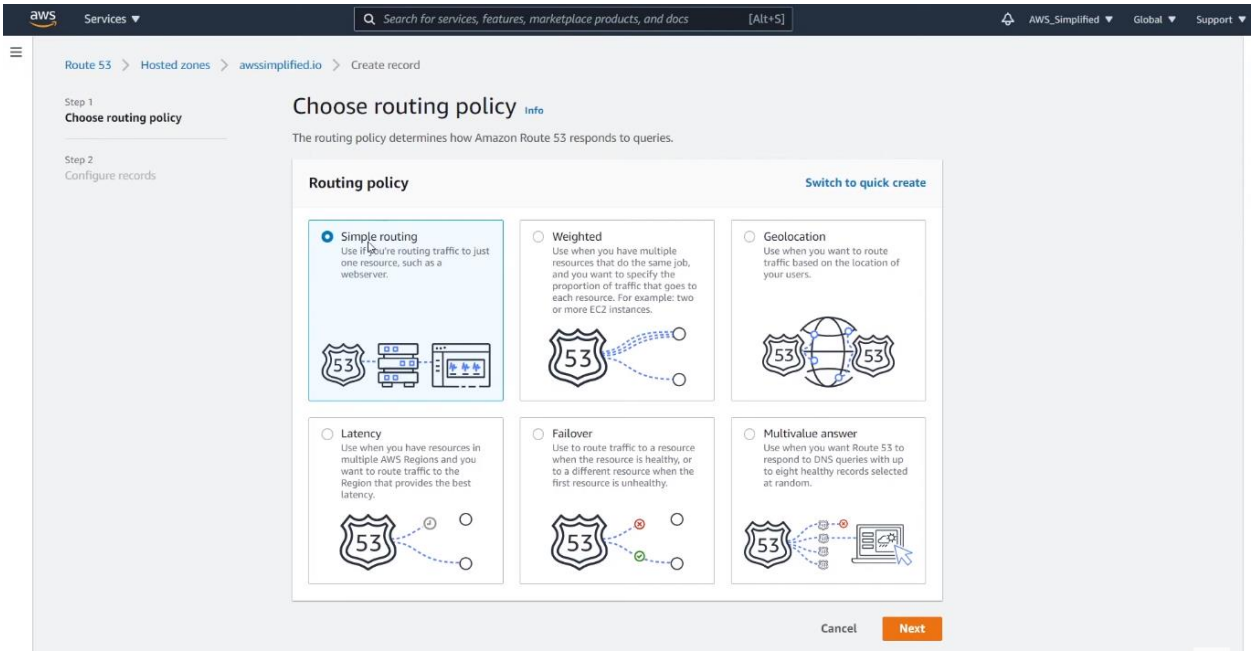




STEPS TO CONFIGURE API GATEWAY DOMAIN NAME WITH CUSTOM DOMAIN NAME

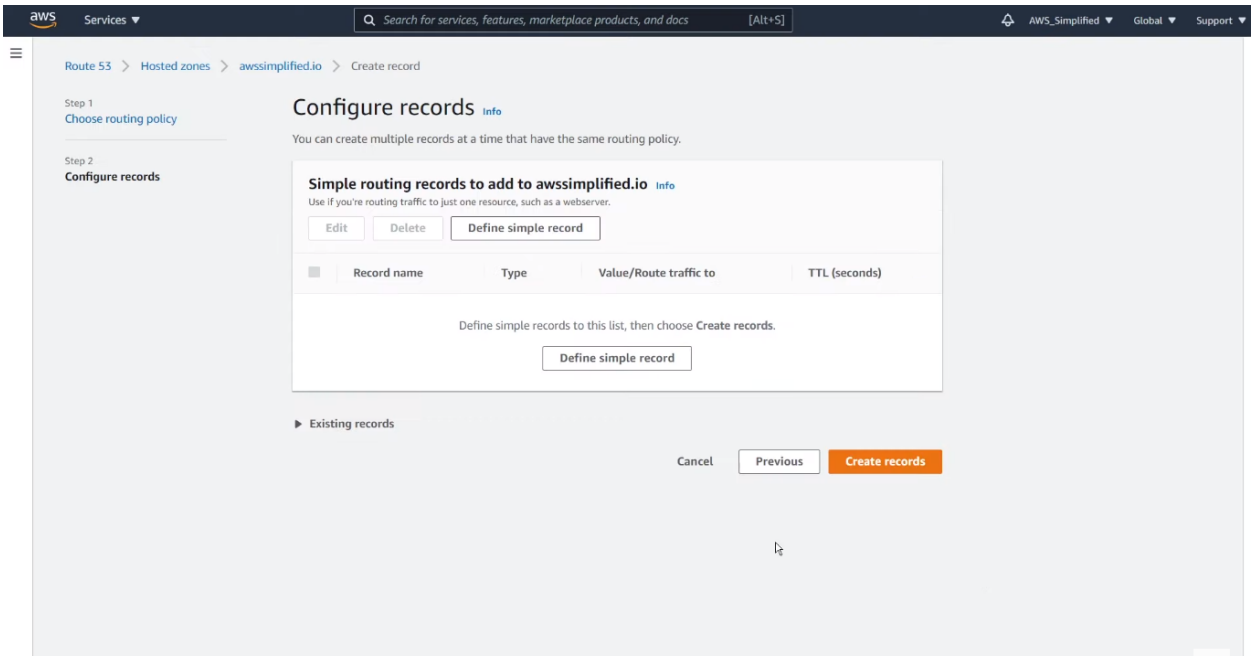
STEP 1: Create a new DNS record, click the "Create Record Set" button.





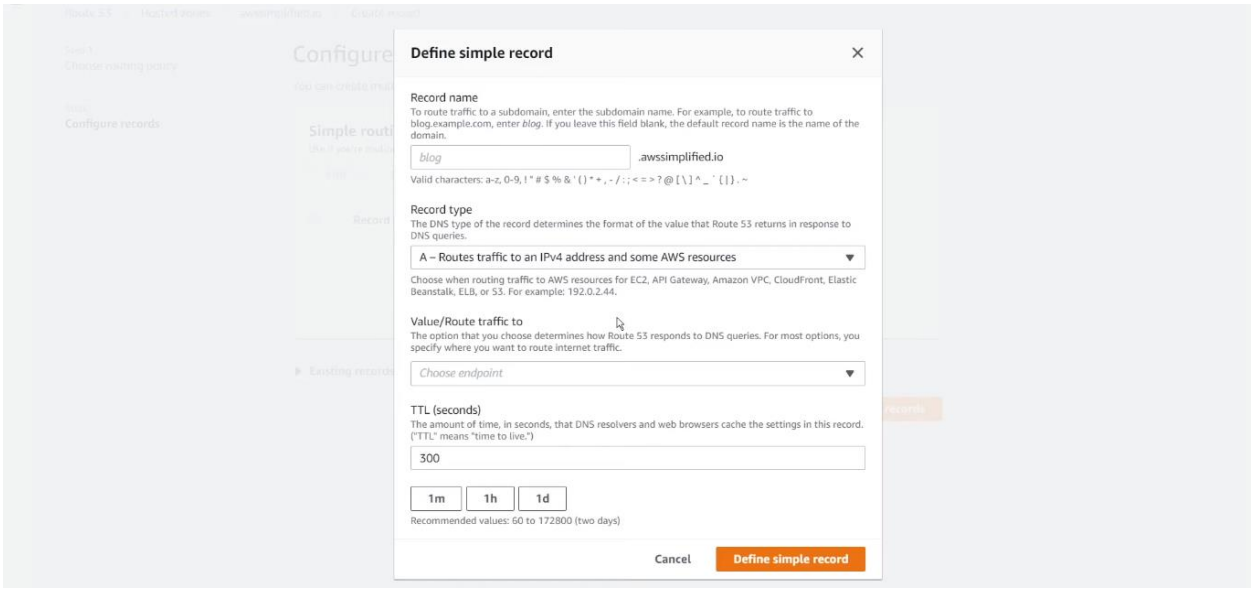
STEP 2: Add DNS Records

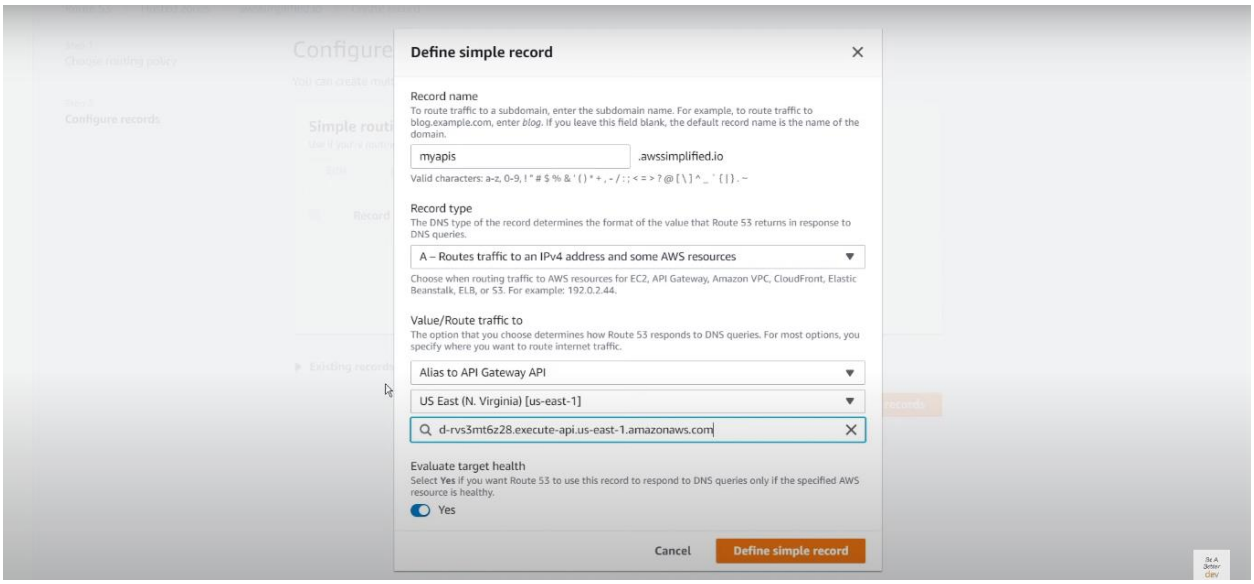
- In the newly created hosted zone, you can add various DNS record types (e.g., A, CNAME, MX, etc.) by clicking the "Create record set" button.
- Click the "Create records" or "Save record set" button to save the DNS record.



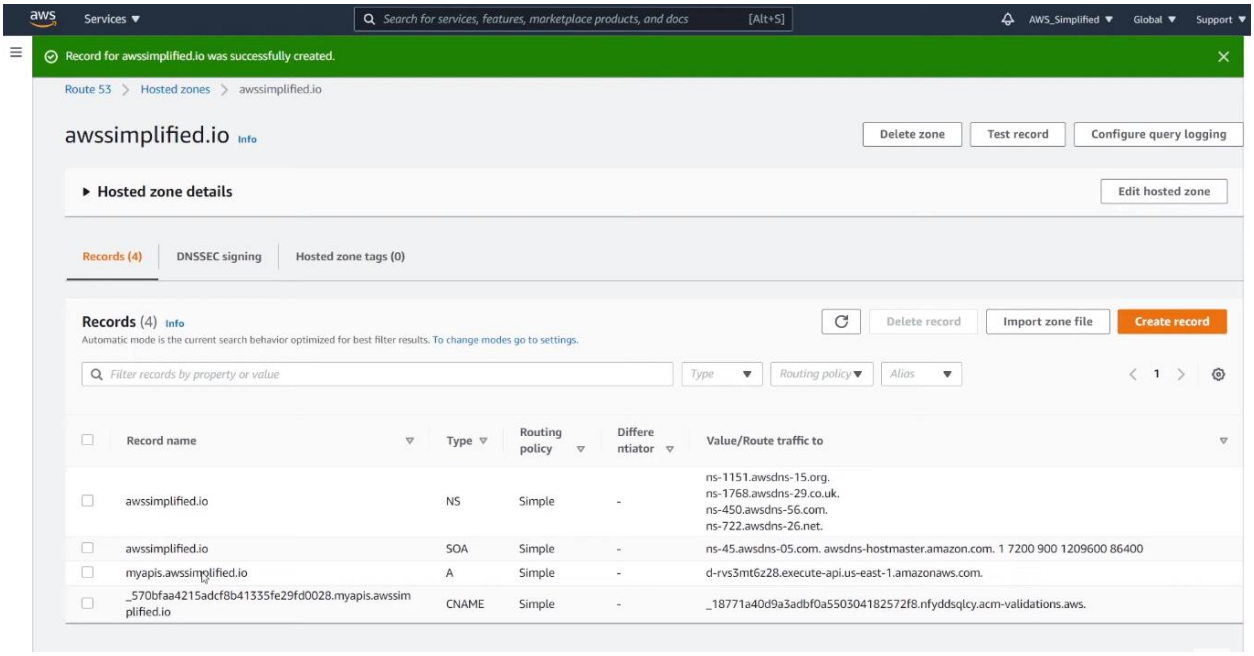
1. Name: Enter the subdomain or leave it blank for the root domain.
2. Type: Select the appropriate DNS record type.
3. Alias: For some record types like A or AAAA, you can configure an alias to point to AWS resources like an Elastic Load Balancer (ELB) or S3 bucket. If not using an alias, enter the IP address or target hostname.
4. Routing Policy: For more advanced use cases, like weighted or latency-based routing, select the appropriate routing policy.
5. Value: Enter the relevant data for the chosen record type
6. TTL (Time to Live): Set the time-to-live value for the record. This determines how long DNS resolvers should cache the record's information.

.





STEP 3: Now the record is added

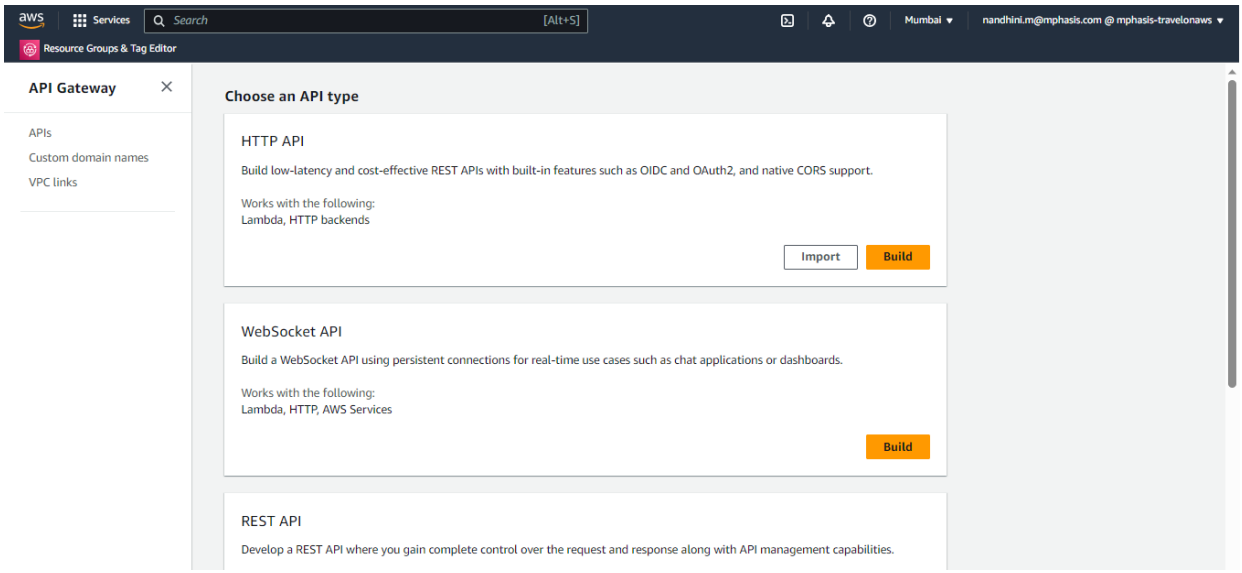


STEP 4: you can test the DNS record by entering the domain name in a web browser

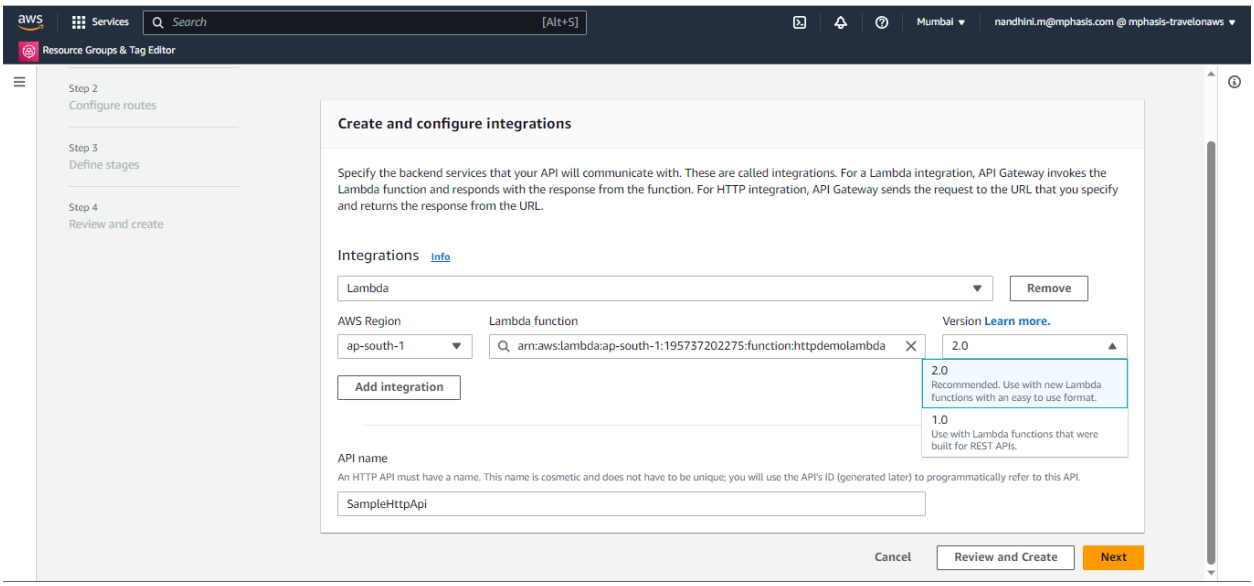
STEPS TO CREATE HTTP API

To create an HTTP API with Lambda integration in AWS, you can follow these steps:

STEP 1: Select "HTTP API" as the protocol for a simpler and cost-effective option.



STEP 2: Provide an API name and, optionally, a description and Click the "Create API" button.

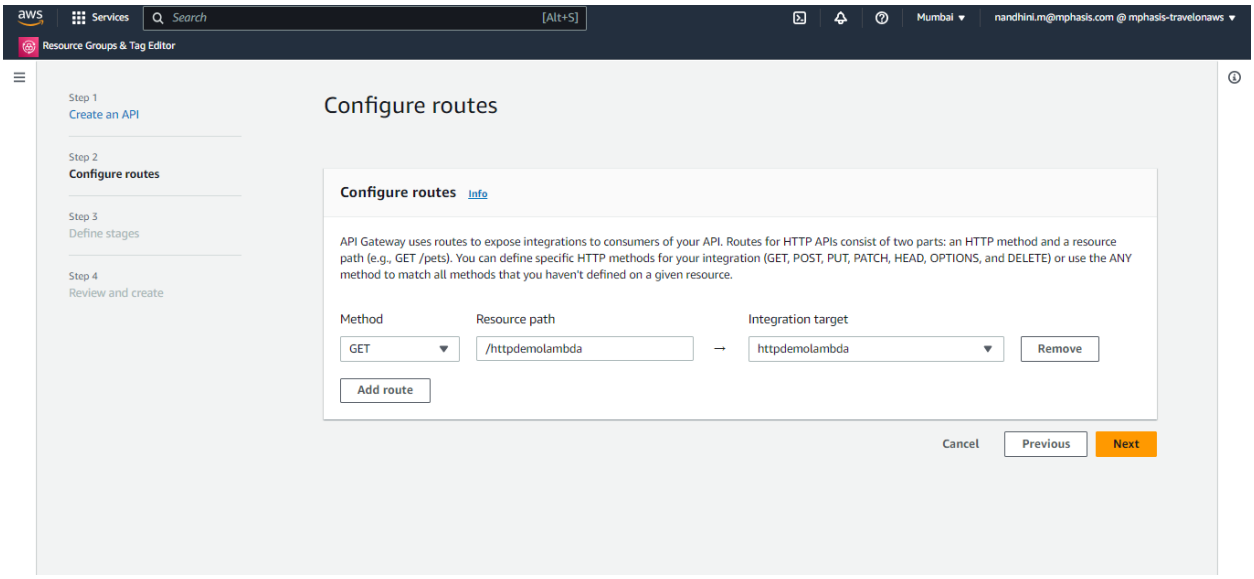


STEP 3:In your newly created API, navigate to the "Routes" section. Click the "Add Integration" button to create a new route.

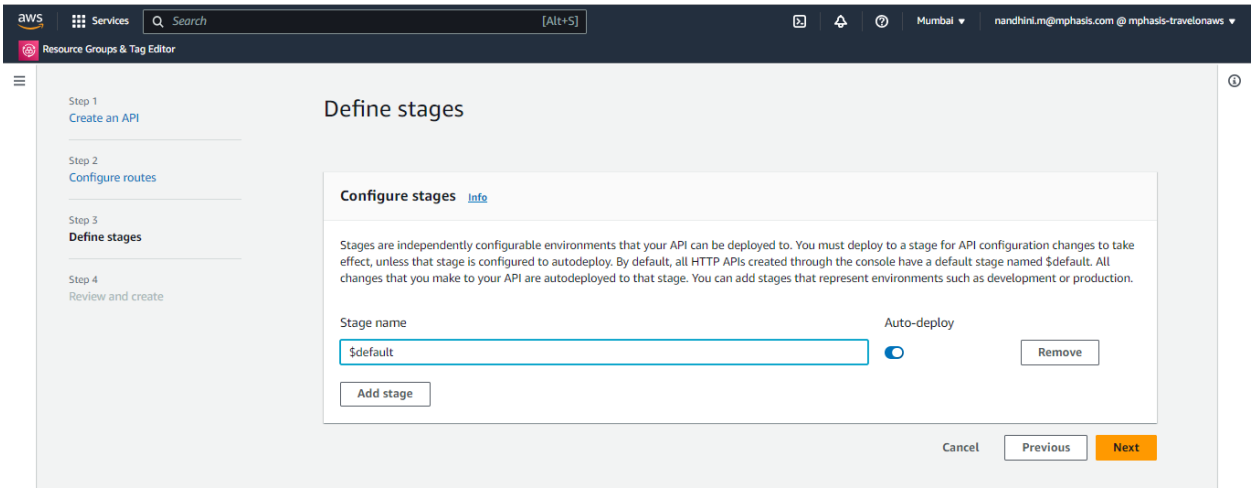
STEP 4:Choose "Lambda Function" as the integration type.

STEP 5: Select the Lambda function you want to integrate with from the dropdown list.

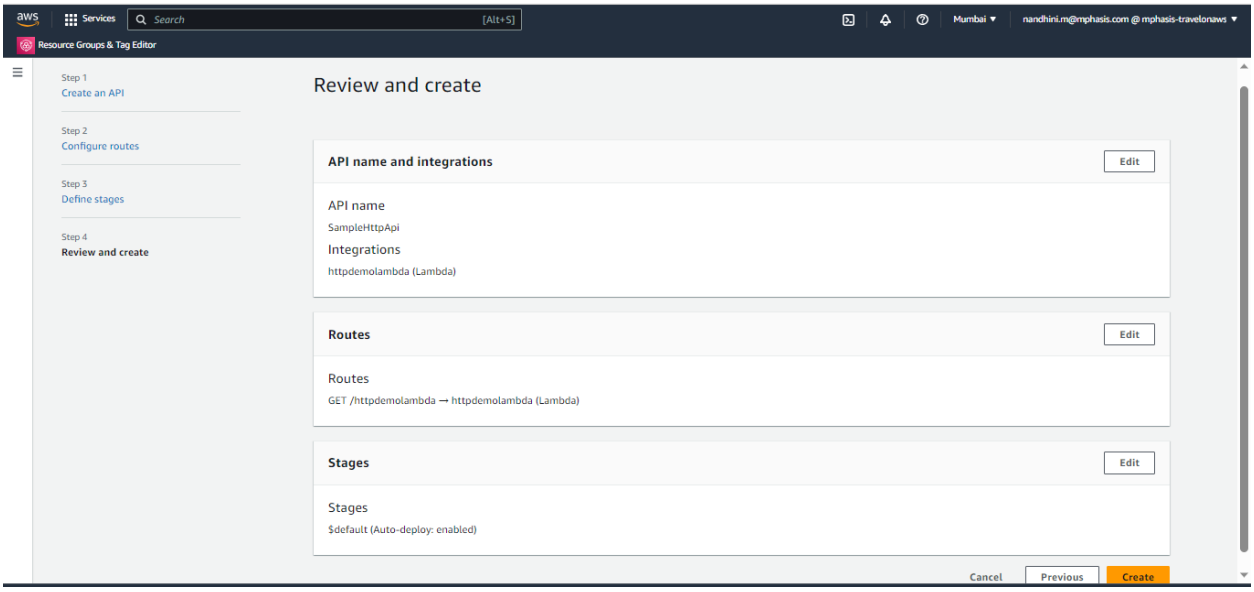
STEP 6:- Configure the HTTP method (e.g., GET, POST) and resource path for this route

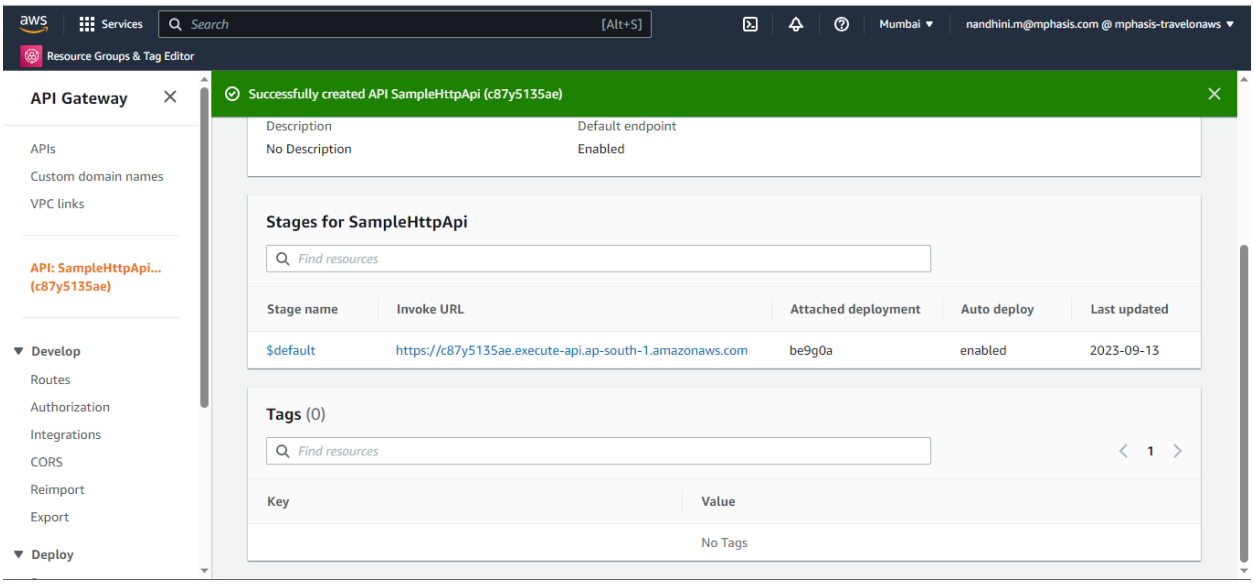


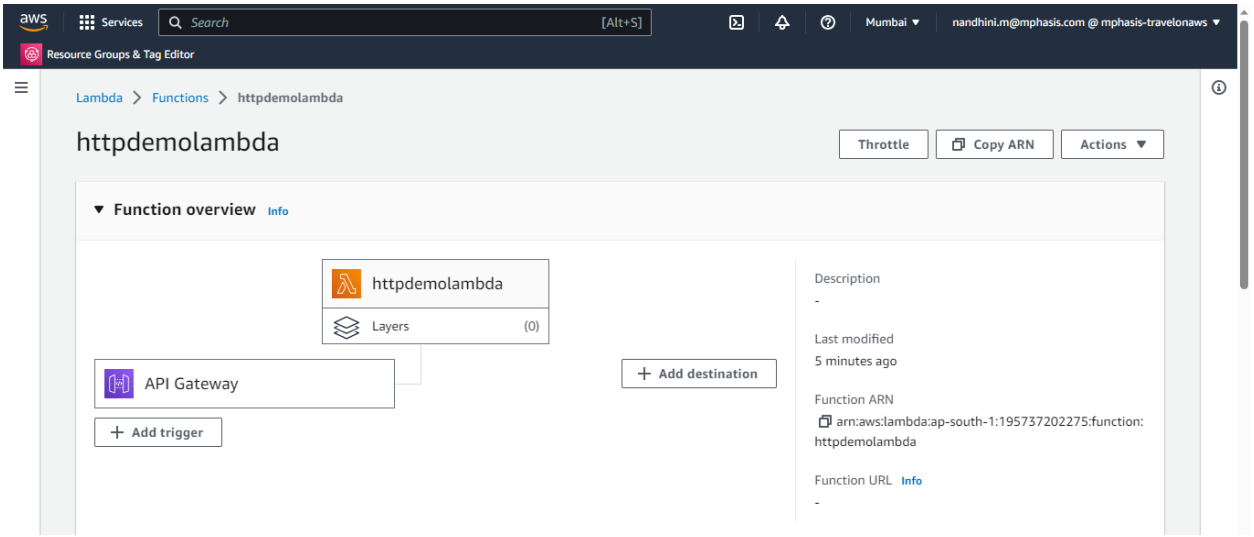
STEP 7: Go to the "Stages" section, click "Create," and set up a new stage (e.g., "prod" or "dev").Deploy your API to this stage.

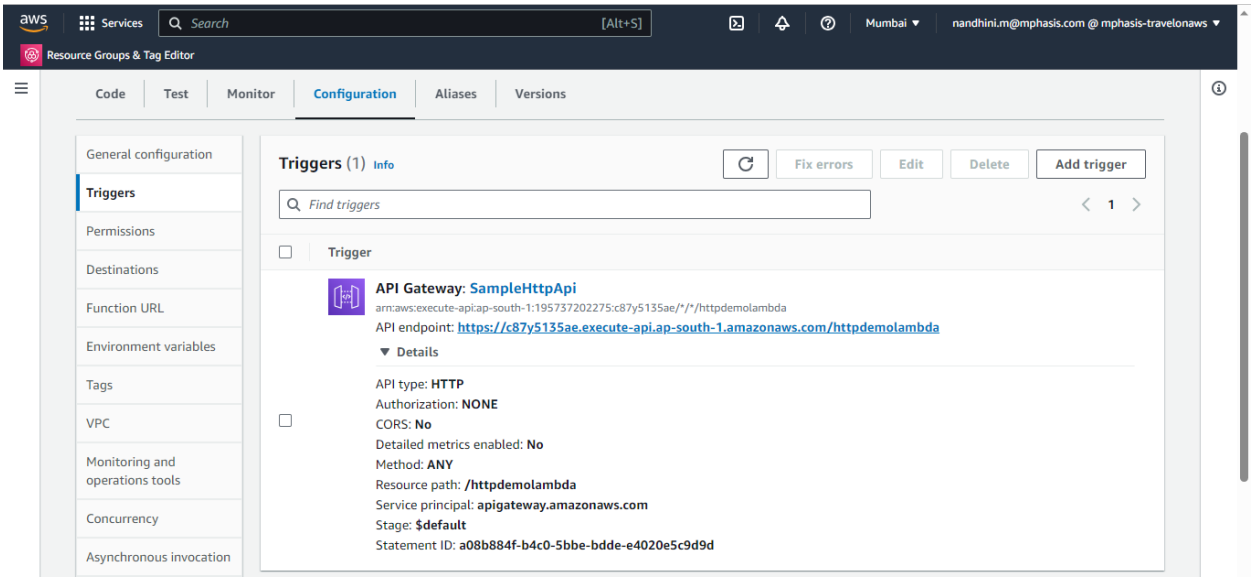


STEP 8: Click the "Create API" button.



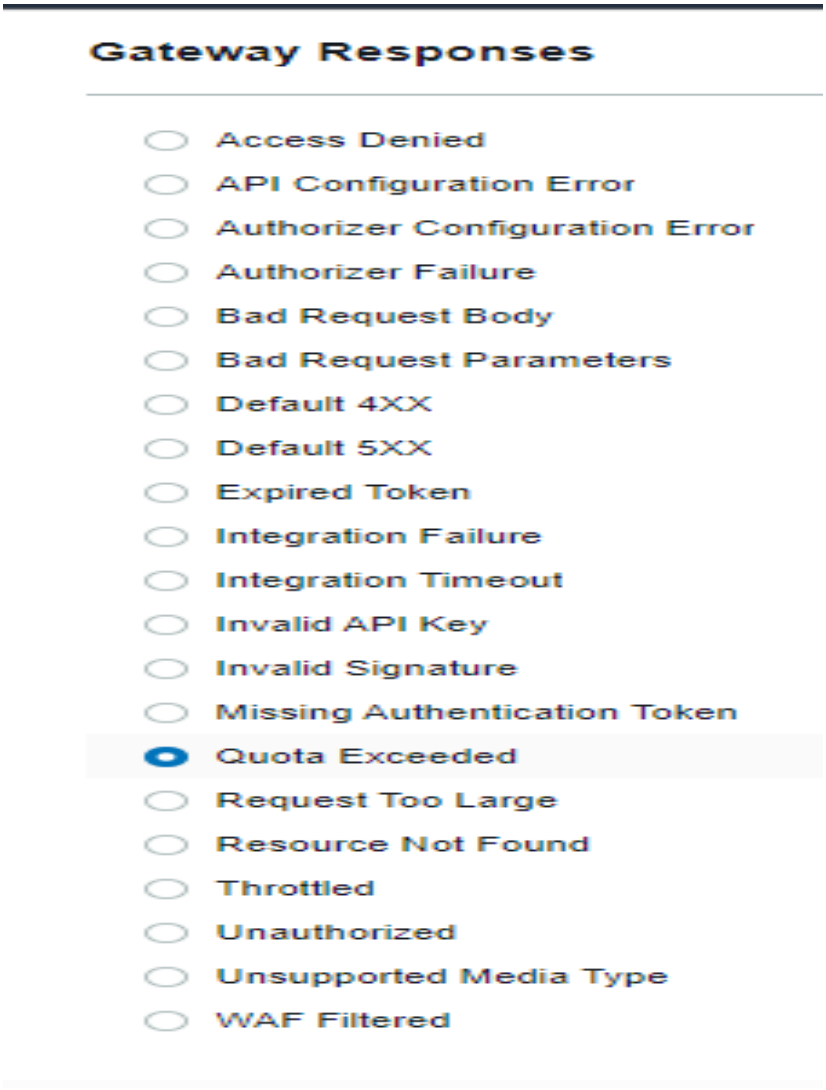



Services
Search
[Alt+S]
Mumbai
nandhini.m@mphasis.com @ mphasis-travelonaws
Resource Groups & Tag Editor
Lambda > Functions > httpdemolambda
httpdemolambda
Throttle
Copy ARN
Actions
Function overview Info
httpdemolambda
Layers (0)
API Gateway
Add trigger
Add destination
Description
-
Last modified
5 minutes ago
Function ARN
arn:aws:lambda:ap-south-1:195737202275:function:httpdemolambda
Function URL Info
-


Services
Search
[Alt+S]
Mumbai
nandhini.m@mphasis.com @ mphasis-travelonaws
Resource Groups & Tag Editor
Code
Test
Monitor
Configuration
Aliases
Versions
General configuration
Triggers
Permissions
Destinations
Function URL
Environment variables
Tags
VPC
Monitoring and operations tools
Concurrency
Asynchronous invocation
Triggers (1) Info
Fix errors
Edit
Delete
Add trigger
Find triggers
1
Trigger
API Gateway: SampleHttpApi
arn:aws:execute-api:ap-south-1:195737202275:c87y5135ae/*/*/httpdemolambda
API endpoint: https://c87y5135ae.execute-api.ap-south-1.amazonaws.com/httpdemolambda
Details
API type: HTTP
Authorization: NONE
CORS: No
Detailed metrics enabled: No
Method: ANY
Resource path: /httpdemolambda
Service principal: apigateway.amazonaws.com
Stage: $default
Statement ID: a08b884f-b4c0-5bbe-bdde-e4020e5c9d9d

GATEWAY RESPONSES

These are the gateway responses available in the API gateway.



1. **Gateway response type**: ACCESS DENIED

Default status code - Status code to be returned to the client -403

Description -

- The gateway responses for authorization failure for example, when access is denied by custom or Amazon Cognito authorizer.
- If the response type is unspecified, this response defaults to the DEFAULT_4XX type.
- In API Gateway, an "Access Denied" response typically indicates that the client making a request to an API endpoint does not have the necessary permissions or authorization to access that resource.

This can happen for various reasons, such as:

1. Missing or Invalid API Key
2. Incorrect IAM Permissions
3. Authentication failure
4. Resource-Based Policies
5. Rate Limiting

2. **Gateway response type:** API CONFIGURATION ERROR

Default status code - Status code to be returned to the client -500

Description -

- The gateway response for invalid API configuration, including invalid endpoint address submitted, or integration response mapping cannot match any template and no default template is configured.
- If the response type is unspecified, this response defaults to the DEFAULT_5XX type.
- The "API_CONFIGURATION_ERROR" response in API Gateway typically indicates that there is an issue with the configuration of your API.

This error can occur for various reasons, such as:

1. Missing or Invalid API Key
2. Misconfigured Routes or Endpoints

3.Integration Errors

4.Resource Permissions

5.Deployment Issues

3.Gateway response type: AUTHORIZER CONFIGURATION ERROR

Default status code - Status code to be returned to the client -500

Description -

- The gateway response for failing to connect to a custom or Amazon Cognito authorizer.
- If the response type is unspecified, this response defaults to the DEFAULT_5XX type.
- "Authorizer Configuration Error" in API Gateway typically indicates a problem with the configuration of an authorizer for your API. An authorizer is responsible for authentication and authorization of incoming requests.

Here are some common reasons you might encounter this error:

1.Invalid Authorizer Configuration

2.Authorizer Resource Permissions

3.Token Validation Issues

4.Authorizer Lambda Function

5.API Gateway Deployment

6.Logging and Monitoring

4.Gateway response type: AUTHORIZER_FAILURE

Default status code - Status code to be returned to the client -500

Description -

- The gateway response when a custom or Amazon Cognito authorizer failed to authenticate the caller.
- If the response type is unspecified, this response defaults to the DEFAULT_5XX type.

- An "Authorizer Failure" error in API Gateway indicates that the authorization process for an incoming API request has failed. This error typically occurs when the authorizer, which is responsible for authenticating and authorizing requests, encounters an issue.

Here are some common reasons for this error:

1. Invalid or Expired Tokens
2. Authorizer Misconfiguration
3. Authorizer Code or Lambda Function Issues
4. Resource Permissions
5. Rate Limiting or Quotas
6. Integration Issues
7. Logging and Monitoring

5.Gateway Response type - BAD REQUEST BODY

Default Status Code - Status code to be returned to the client "400".

Description -

- The gateway response when the request body cannot be validated according to an enabled request validator.
- If the response type is unspecified, this response defaults to the DEFAULT_4XX type.

Reason for Bad request body error in API Gateway:

1. Invalid data format
2. Missing required fields
3. Data type mismatch
4. Exceeding size limits
5. Authentication and Authorization issues

6.Gateway Response type - BAD REQUEST PARAMETERS

Default Status Code - Status code to be returned to the client "400".

Description –

- The gateway response when the request parameter cannot be validated according to an enabled request validator.
- If the response type is unspecified, this response defaults to the DEFAULT_4XX type.

Reason for Bad request parameter error in API Gateway:

1.Missing required parameter

2.Invalid parameter names

3.Data type mismatch

4.Exceeding parameter length limits

5.Incorrect parameter format

7.Gateway Response type - DEFAULT 4XX

Default Status Code - Status code to be returned to the client "Null".

Description -

- The default gateway response for an unspecified response type with the status code of 4XX.
- Changing the status code of this fallback gateway response changes the status codes of all other 4XX responses to the new value.
- Resetting this status code to null reverts the status codes of all other 4XX responses to their original values.

Reason for Default 4XX error in API Gateway:

1.400 Bad Request

2.401 Unauthorized

3.402 Payment Required

4.404 Not Found

8.Gateway Response type - DEFAULT 5XX

Default Status Code - Status code to be returned to the client "Null".

Description –

- The default gateway response for an unspecified response type with the status code of 5XX.
- Changing the status code of this fallback gateway response changes the status codes of all other 5XX responses to the new value.
- Resetting this status code to null reverts the status codes of all other 5XX responses to their original values.

Reason for Default 5XX error in API Gateway:

1.500 Internal server error

2.501 Not Implemented

3.502 Bad Gateway

4.503 Service Unavailable

9.Gateway Response type - Expired Token

Default Status Code: Status code to be returned to the client "403".

Description –

- The gateway response for an AWS authentication token expired error.
- If the response type is unspecified, this response defaults to the Default 4XX type.

Reason for Expired token error in API Gateway:

1.Security

2.Access control

3.Reducing Attack surface

4.User and Client Management

5.Token Rotation

10.Gateway Response type - Integration Failure

Default Status Code - Status code to be returned to the client “504”.

Description –

- The gateway response for an integration failed error.
- If the response type is unspecified, this response defaults to the Default 5XX type.

Reason for Integration failure error in API Gateway:

1. Backend Service Unreachable
2. Incorrect integration Configuration
3. Authentication and Authorization Issues
4. Resource Limitations
5. Lambda function errors