

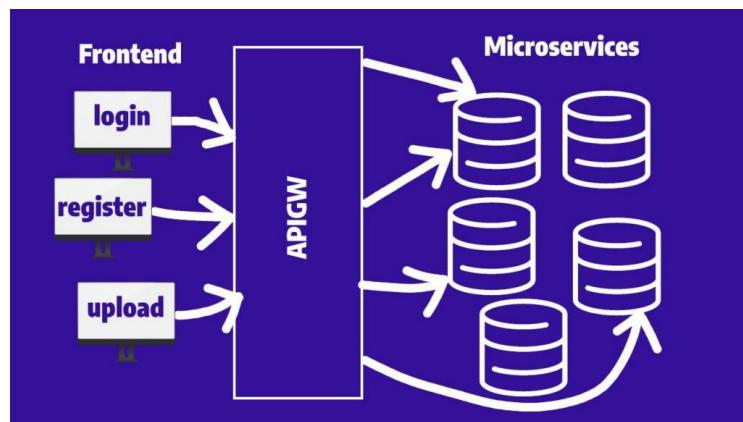
TABLE OF CONTENTS

WHAT IS API GATEWAY?.....	3
WHY DO WE NEED AN API GATEWAY?.....	4
WHERE DO WE USE AN API GATEWAY?.....	4
WHEN DO WE USE AN API GATEWAY?	5
INTRODUCTION.....	5
HISTORY OF AMAZON API GATEWAY	5
ARCHITECTURE OF API GATEWAY	6
WORKFLOW OF API GATEWAY	8
INTEGRATING OPTIONS	8
FEATURES OF API GATEWAY.....	9
COST OF API GATEWAY.....	10
ADVANTAGE OF USING API GATEWAY.....	11
DISADVANTAGE OF USING API GATEWAY.....	13
HOW TO CONNECT API GATEWAY.....	15
STEPS TO CREATE API GATEWAY.....	15
INTEGRATING TYPES.....	24
CONFIGURATION STEPS FOR DIFFERENT INTEGRATION TYPES WHEN SETTING UP AN API GATEWAY ENDPOINT.....	25
1. HTTP Integration:	25
2. Lambda Function Integration:	29
3. Mock Integration:.....	30
4. AWS Service Integration:.....	37
UPDATE STAGE SETTINGS USING API GATEWAY CONSOLE.....	65
1.CACHE SETTINGS.....	65
2.DEFAULT METHOD THROTTLING.....	67
3.LOGS/TRACING.....	69
4.STAGE VARIABLE.....	71

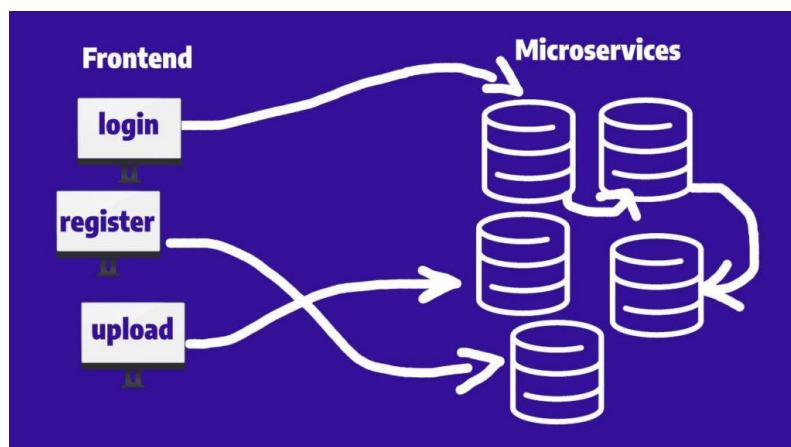
5.CANARY DEPLOYMENT	78
6.EXPORT:.....	82
7.DEPLOYMENT HISTORY.....	82
8.DOCUMENTATION HISTORY.....	83
CUSTOM DOMAIN NAME.....	84
STEPS TO CREATE CUSTOM DOMAIN NAME	85
STEPS TO CREATE ACM CERTIFICATE	87
ADD ACM TO CUSTOM DOMAIN NAME	94
STEPS TO CREATE HTTP API	102
GATEWAY RESPONSES	106

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.

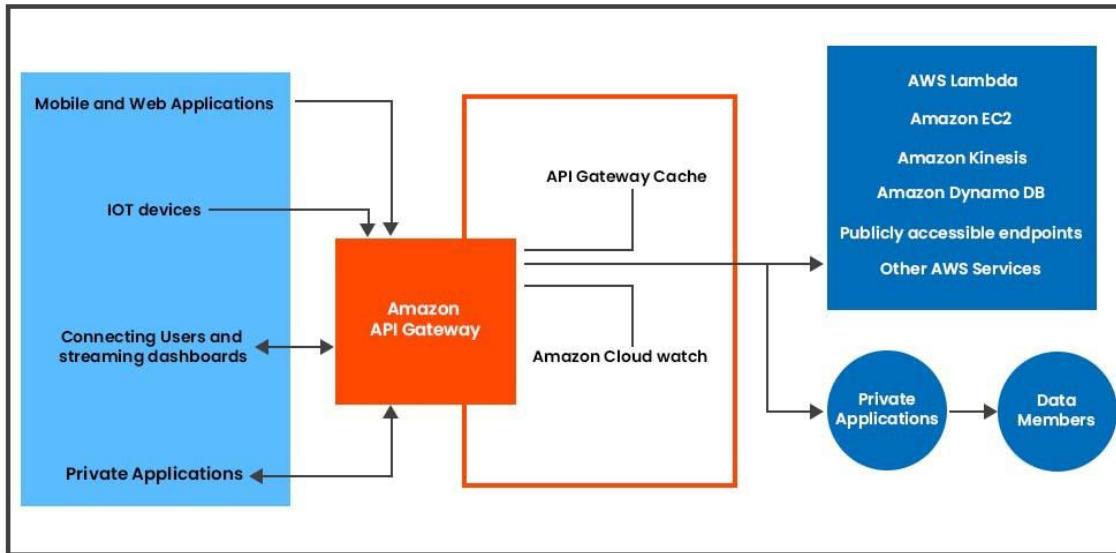
- **2016:** WebSocket API support was introduced. This feature allowed developers to create APIs that supported real-time, bidirectional communication between clients and servers over a single connection. AWS launched the "REST APIs with Private Integrations" feature, allowing users to integrate their 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

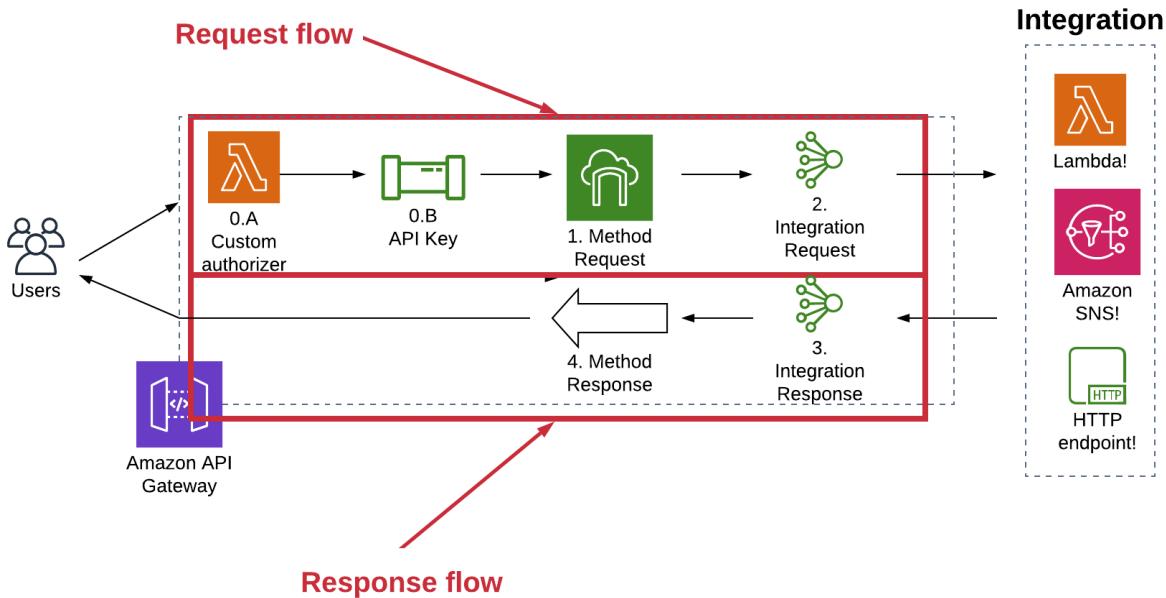
building AWS serverless applications. It acts as a gateway for applications to access data from your backend services such as code running on Amazon Elastic Compute Cloud (Amazon EC2), AWS Lambda, any web applications, etc.



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.

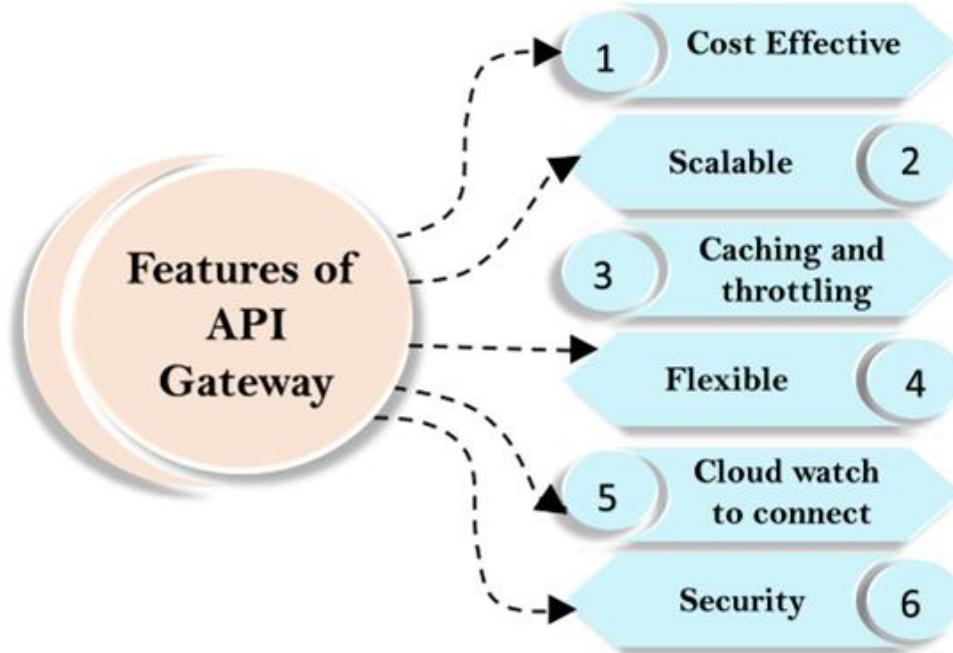
INTEGRATING OPTIONS

- **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
WebSocket	Over 20 billion	\$1.51
	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

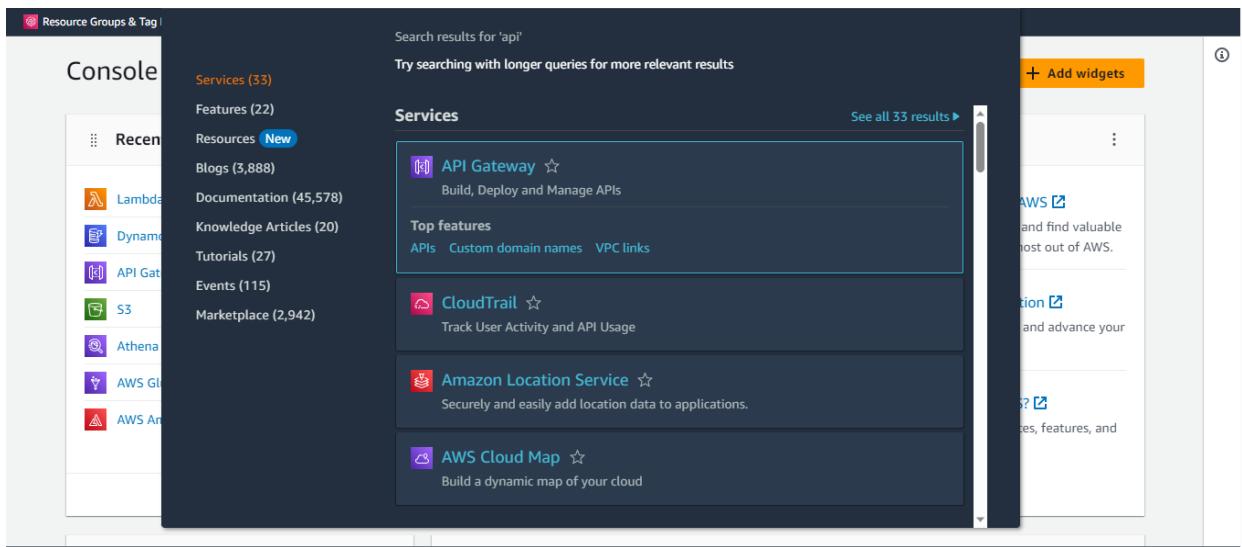
- **Single Point of Failure:** Since an API Gateway act as a central entry point for all API traffic, if the API Gateway goes down, all API communication may be affected, leading to potential service disruptions.
- **Performance Bottleneck:** If not properly designed or if the API Gateway becomes a performance 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

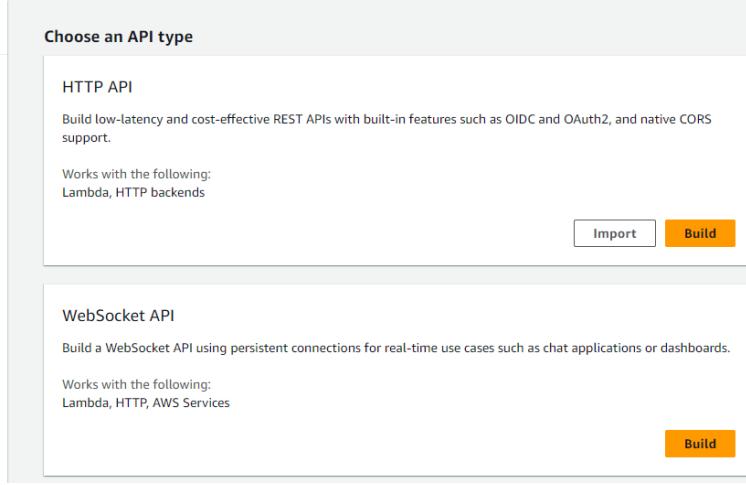
STEP 1: Log in to your AWS Management Console using your AWS account credentials. Once logged in, navigate to the "API Gateway" service. You can find this under the "Networking & Content Delivery" section.



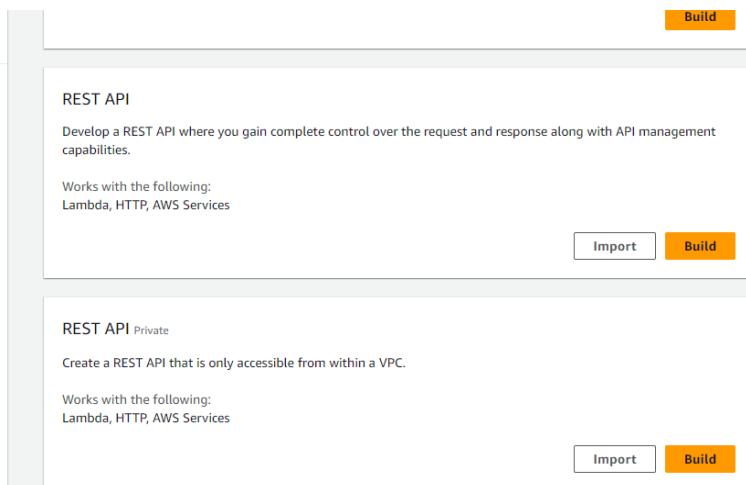
STEP 2: Click on the "Create API" button to start creating a new API.

Name	Description	ID	Protocol	Endpoint type	Created
alaskaforapi		9pbskko0ll	REST	Regional	2023-07-31
alaskaReportAthena api		eyj2hbza84	REST	Regional	2023-07-26
AlaskaReportForgot api		c75lyhtaol	REST	Regional	2023-07-31
alaskaReportLogin api		jdjdtdbf8jl	REST	Regional	2023-07-26
alaskaReportSignUp api		l574ws6n8k	REST	Regional	2023-07-26
AthenaAPI		d3ccg3d69b	REST	Regional	2023-07-21
Aviapi		plmhnkl1g9	REST	Regional	2023-08-09
Billy-reg-api		c3f1c4stc4	REST	Regional	2023-07-25
Black		gzjm8rjt30	REST	Regional	2023-08-09
estforont		vml71n6f3l	REST	Regional	2023-08-14

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.



The screenshot shows the 'API Gateway' interface with the 'APIs' tab selected. A modal window titled 'Choose an API type' is open, displaying two options: 'HTTP API' and 'WebSocket API'. The 'HTTP API' section describes it as a low-latency and cost-effective REST API with built-in features like OIDC and OAuth2, and native CORS support. It works with Lambda, HTTP backends, and has 'Import' and 'Build' buttons. The 'WebSocket API' section describes it as a persistent connection for real-time use cases like chat applications or dashboards, working with Lambda, HTTP, and AWS Services, with a single 'Build' button.



The screenshot shows the 'API Gateway' interface with the 'APIs' tab selected. A modal window titled 'Choose an API type' is open, displaying two options: 'REST API' and 'REST API Private'. The 'REST API' section describes it as a REST API where you gain complete control over requests and responses, working with Lambda, HTTP, and AWS Services, with 'Import' and 'Build' buttons. The 'REST API Private' section describes it as a REST API accessible only from within a VPC, working with Lambda, HTTP, and AWS Services, with 'Import' and 'Build' buttons.

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.

APIs > Create

Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

REST WebSocket

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

New API Clone from existing API Import from Swagger or Open API 3 Example API

Settings

Choose a friendly name and description for your API.

API name*

Description

Endpoint Type
 Regional
 Edge optimized
 Private

* Required

Create API

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).

APIs > SampleApi (eb5xbzobm7) > Resources > / (bp5ko4flok)

Actions | Methods

No methods defined for the resource.

API: SampleApi

- Resources (selected)
- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Dashboard
- Settings
- Usage Plans
- API Keys
- Client Certificates
- Settings

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

The screenshot shows the Amazon API Gateway interface. On the left, there's a navigation sidebar with options like APIs, Custom Domain Names, VPC Links, and a section for the current API named "SampleApi" which includes Resources, Stages, Authorizers, Gateway Responses, Models, Resource Policy, Documentation, and Dashboard. The main area is titled "Resources" and shows a single resource path "/ (bp5ko4ftok)". A context menu is open over this resource, with the "Actions" dropdown expanded. The "Actions" dropdown has two sections: "RESOURCE ACTIONS" containing Create Method, Create Resource, Enable CORS, and Edit Resource Documentation; and "API ACTIONS" containing Deploy API, Import API, Edit API Documentation, and Delete API. The "Delete API" option is highlighted in red.

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`](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.

The screenshot shows the "New Child Resource" configuration page. The left sidebar is identical to the previous screenshot. The main area is titled "New Child Resource" and contains a form for creating a new child resource. The form includes fields for "Resource Name*" (set to "Sampleresource") and "Resource Path*" (set to "/ sampleresource"). There's also a checkbox for "Configure as proxy resource" which is unchecked. Below the form is a note about using path parameters. At the bottom, there's a "Required" label, a "Cancel" button, and a prominent "Create Resource" button.

The screenshot shows the Amazon API Gateway console. The navigation bar at the top indicates the path: APIs > SampleApi (eb5xbzobm7) > Resources > /sampleresource (pm2vdi). On the left sidebar, under the API: SampleApi section, the Resources option is selected. The main content area displays the details for the /sampleresource resource, specifically the OPTIONS method. The method configuration shows 'Mock Endpoint' and 'Authorization None'. The sidebar also lists other options like Stages, Authorizers, and Models.

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.

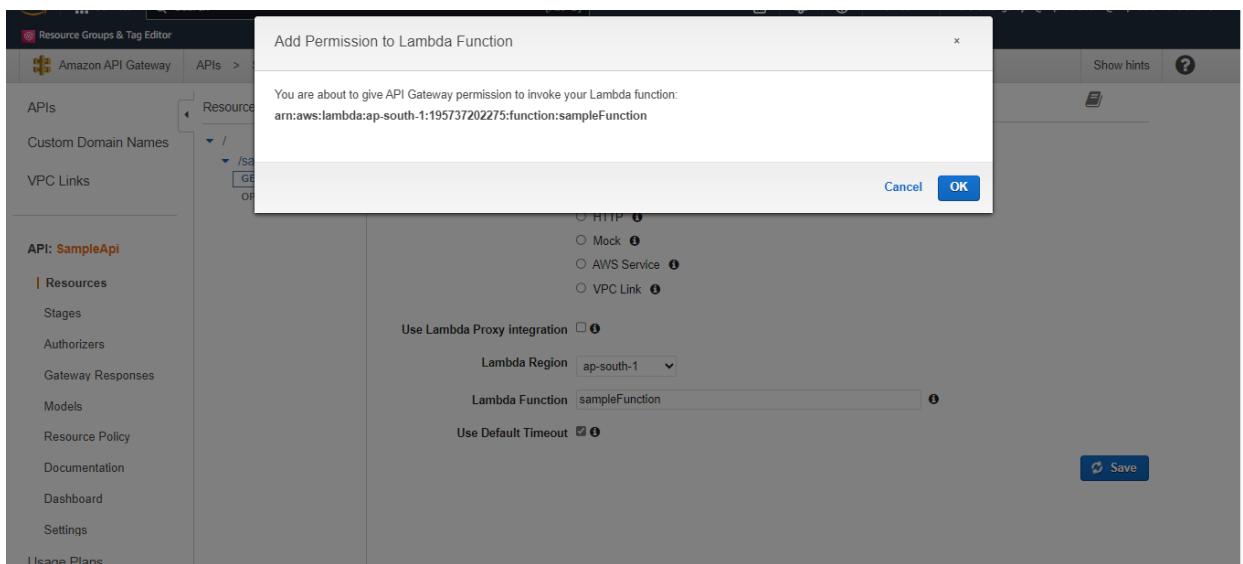
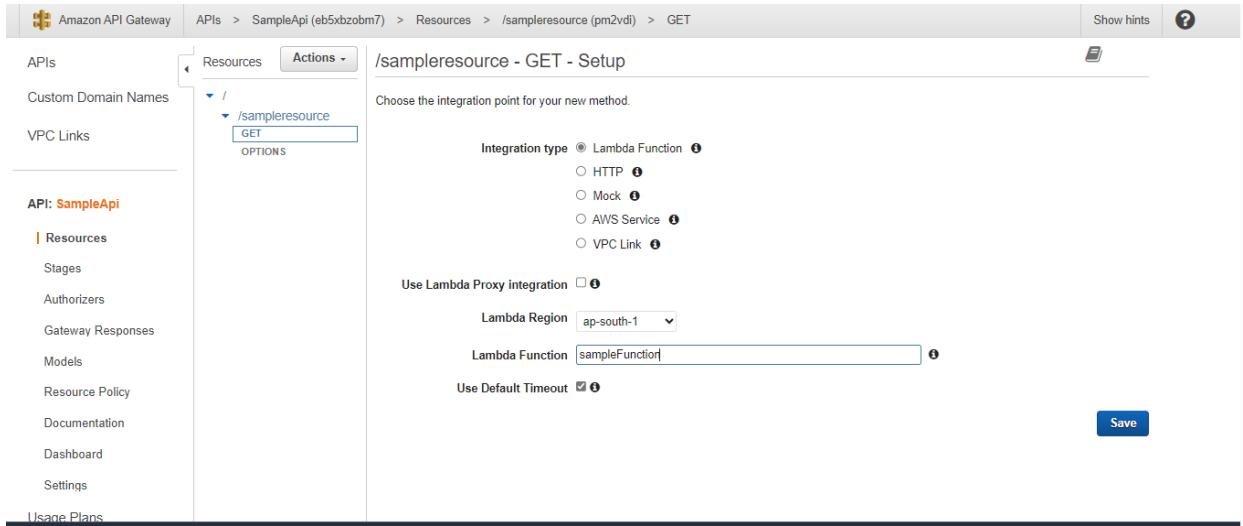
The screenshot shows the same Amazon API Gateway interface as the previous one, but the Actions dropdown menu is open over the /sampleresource resource entry. The menu is divided into two sections: RESOURCE ACTIONS and API ACTIONS. Under RESOURCE ACTIONS, the options are Create Method, Create Resource, Enable CORS, Edit Resource Documentation, and Delete Resource. Under API ACTIONS, the options are Deploy API, Import API, Edit API Documentation, and Delete API. The 'Create Method' option is highlighted.

The screenshot shows the AWS Lambda interface for managing API resources. On the left, a sidebar lists various API components like APIs, Custom Domain Names, VPC Links, and the current API named 'SampleApi'. Under 'Resources', the path '/sampleresource' is selected. In the main area, the 'Actions' dropdown is open, and 'OPTIONS' is selected from the list. A modal window titled 'OPTIONS' displays configuration details: 'Mock Endpoint' is set, 'Authorization' is 'None', and 'API Key' is marked as 'Not required'. A dropdown menu on the left lists other HTTP methods: ANY, DELETE, GET, HEAD, PATCH, POST, and PUT.

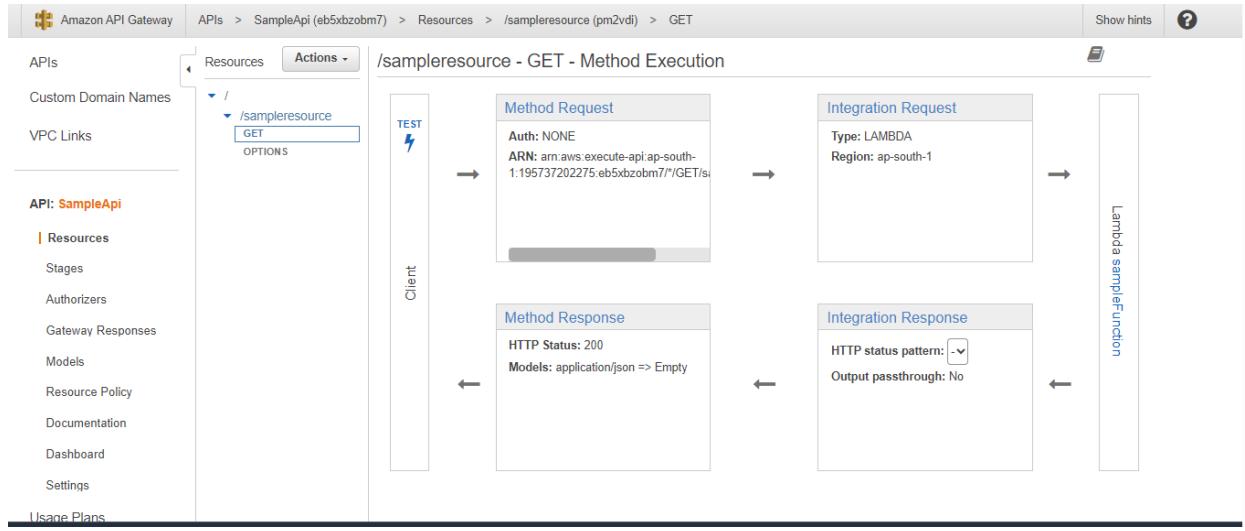
Other options for an API method include:

- **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.

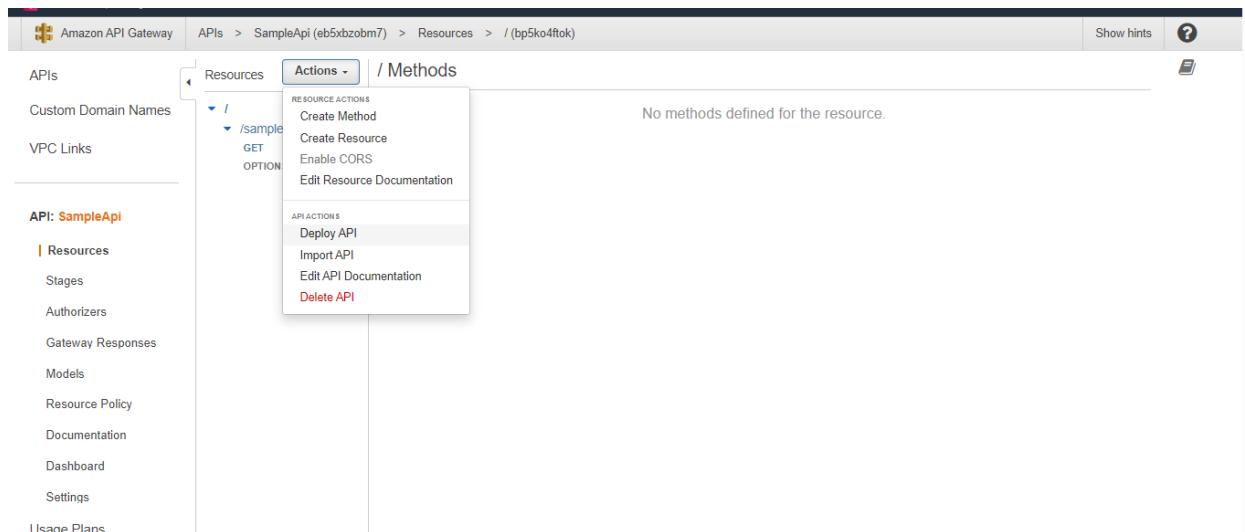
STEP 10: For each method, you'll need to configure how the method integrates with your backend service or Lambda function. You can choose from various integration types such as Lambda Function, HTTP, Mock, AWS Service, etc.

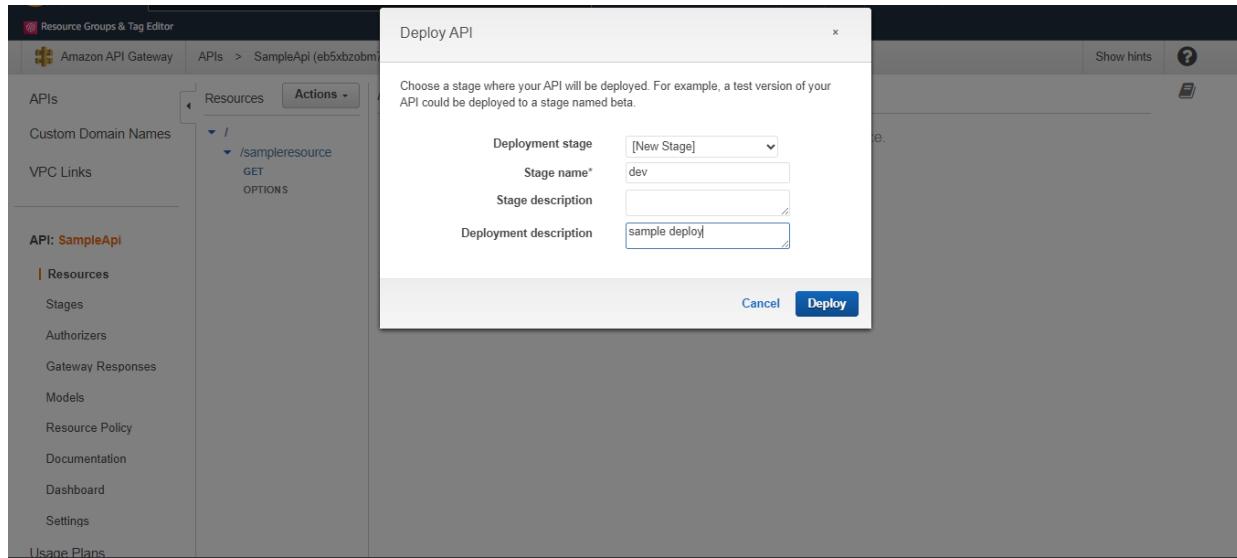


STEP 11: Define the method request and response settings. This includes configuring request parameters, headers, query strings, and mapping templates if needed.

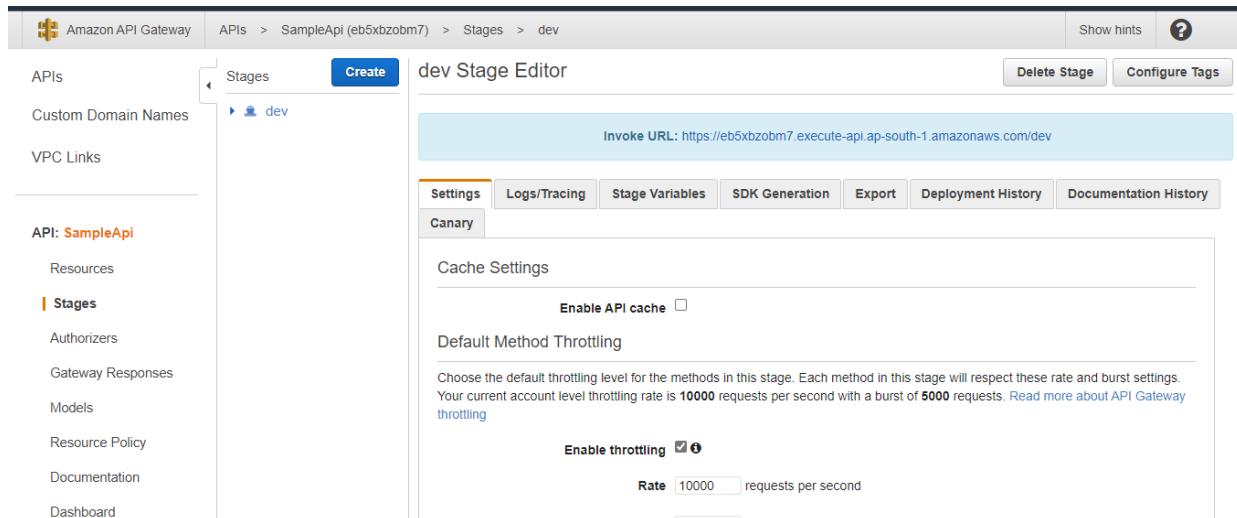


STEP 12: Once your resource and methods are configured, you need to deploy the API to make it accessible. Click on the "Deploy API" button and create a new stage (e.g., "prod" for production). This will provide you with a URL where clients can access your API.





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.



The screenshot shows the 'Stage Configuration' section of the AWS API Gateway console. On the left, a sidebar lists various API management features: Gateway Responses, Models, Resource Policy, Documentation, Dashboard, Settings, Usage Plans, API Keys, Client Certificates, and Settings. The main content area is titled 'Configure Stage' and contains the following fields:

- Throttling:** A note states: "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](#)". Below this are input fields for 'Rate' (10000) and 'Burst' (5000).
- Web Application Firewall (WAF):** A link to 'Learn more'.
- Web ACL:** A dropdown menu set to 'None' with a 'Create Web ACL' button.
- Client Certificate:** A note: "Select the client certificate that API Gateway will use to call your integration endpoints in this stage." Below is a dropdown menu set to 'None'.

A blue 'Save Changes' button is located at the bottom right of the configuration area.

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.

The screenshot shows the AWS API Gateway console with the following details:

- Left Sidebar:** APIs, Custom Domain Names, VPC Links, API: SampleApi (selected), Resources, Stages, Authorizers, Gateway Responses, Models, Resource Policy, Documentation, Dashboard.
- Current View:** /sampleresource - GET - Setup
- Resource Tree:** / (GET, ANY), /MyNewFunc (ANY), /sampleresource (selected, GET, OPTIONS).
- Integration Configuration:**
 - Integration type: HTTP (selected), Lambda Function, Mock, AWS Service, VPC Link.
 - Use HTTP Proxy Integration:
 - HTTP method:
 - Endpoint URL:
 - Content Handling:
 - Use Default Timeout:
- Buttons:** Save (blue button) at the bottom right.

The screenshot shows the AWS Lambda integration setup for a POST method. The 'Integration type' is set to 'HTTP'. The 'Content Handling' dropdown is open, showing 'Passthrough' as the selected option. Other options available are 'Convert to binary (if needed)' and 'Convert to text (if needed)'. The 'HTTP method' is set to 'POST' and the 'Endpoint URL' is 'https://api.endpoint.com/sampleresource'.

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.

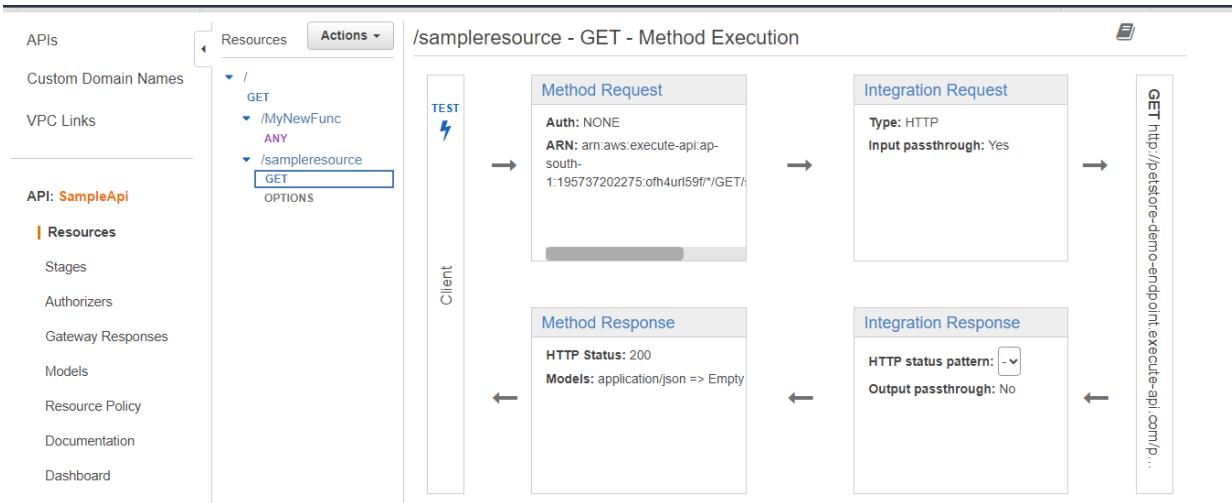
2. Convert to Binary:

- "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.

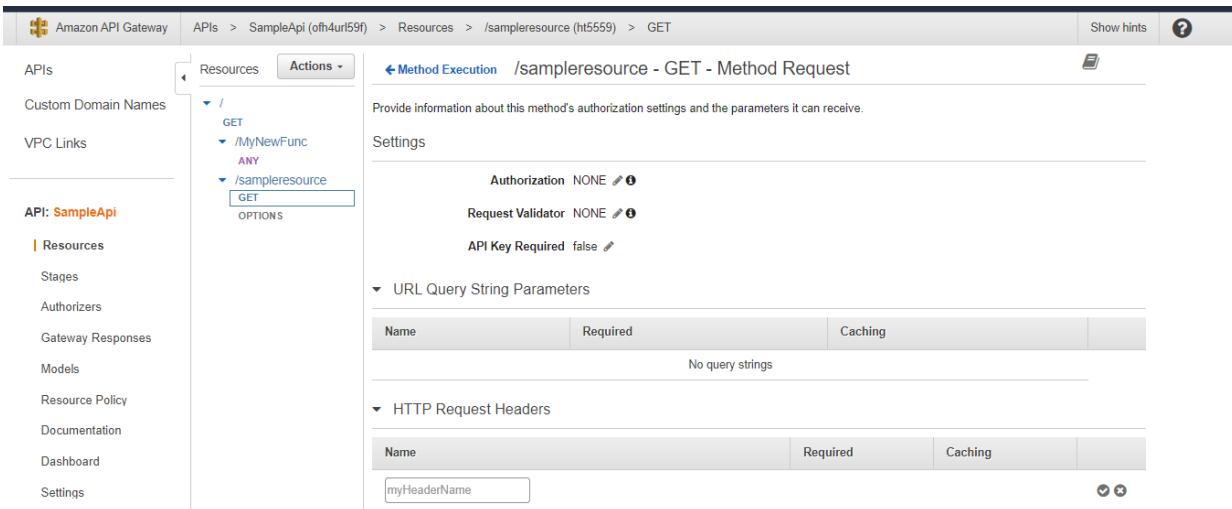
3. Convert to Text:

- "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.

Amazon API Gateway APIs > SampleApi (ofh4url59f) > Resources > /sampleresource (ht5559) > GET Show hints ?

API: SampleApi

- Resources
- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Dashboard
- Settings
- Usage Plans
- API Keys
- Client Certificates
- Settings

Resources Actions ▾

 / GET
 /MyNewFunc ANY
 /sampleresource GET
 OPTIONS
 POST

Method Execution /sampleresource - GET - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type Lambda Function i
 HTTP i
 Mock i
 AWS Service i
 VPC Link i

Use HTTP Proxy integration i

HTTP method GET e

Endpoint URL <http://petstore-demo-endpoint.execute-api.com/petstore/pets> e

Content Handling Passthrough e i

Use Default Timeout i

URL Path Parameters

Name	Mapped from <small>i</small>	Caching
No path parameters		

Add path

URL Query String Parameters

Name	Mapped from <small>i</small>	Caching
No query strings		

Add query string

HTTP Headers

Name	Mapped from <small>i</small>	Caching
No headers		

Add header

Mapping Templates

Request body passthrough When no template matches the request Content-Type header i
 When there are no templates defined (recommended) i
 Never i

Content-Type

No mapping templates defined. The request body will be passed through to the integration endpoint

Add mapping template

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.

The screenshot shows two screenshots of the Amazon API Gateway console. The top screenshot displays the 'Setup' page for a GET method on the '/sampleresource' resource of the 'SampleApi' API. It shows the selection of 'Lambda Function' as the integration type, with 'sampleFunction' selected from the dropdown. The bottom screenshot shows the 'Method Execution' page for the same GET method, illustrating the flow from a 'Client' sending a 'Method Request' to a 'Lambda sampleFunction' executing an 'Integration Request' and returning a 'Method Response'.

3. Mock Integration:

- Integration Type: Mock

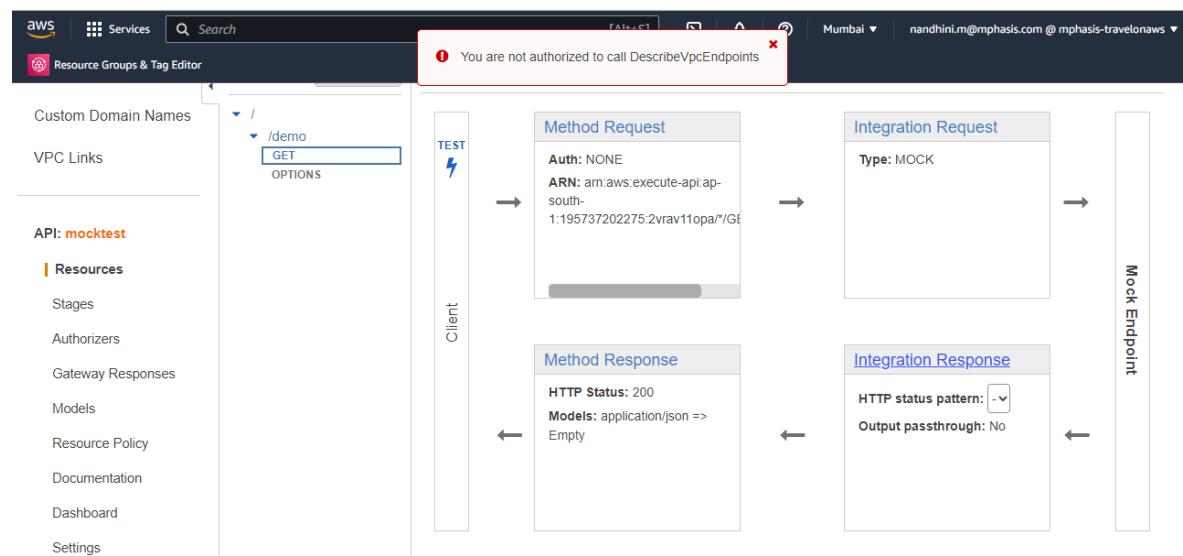
STEP 1: Select your API in the API Gateway console.

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.

The screenshot shows the AWS API Gateway console. On the left, the navigation pane is visible with options like APIs, Custom Domain Names, VPC Links, and Resources. Under Resources, there's a tree structure for the API 'mocktest' with a 'demo' stage and a 'GET' method selected. The main panel shows the path: APIs > mocktest (2vrav11opa) > Resources > /demo (7y2die) > GET. The title is '/demo - GET - Setup'. A red warning box at the top says 'You are not authorized to call DescribeVpcEndpoints'. Below it, the 'Integration type' section has 'Mock' selected. There are also options for Lambda Function, HTTP, AWS Service, and VPC Link. A blue 'Save' button is at the bottom right.

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.

The screenshot shows the AWS API Gateway interface. On the left, the navigation pane is visible with options like APIs, Custom Domain Names, VPC Links, and the current API named 'mocktest'. Under 'Resources', there's a tree structure with a root node '/' and a child node '/demo' which has 'GET' and 'OPTIONS' methods. The main content area is titled 'Method Execution /demo - GET - Integration Response'. It instructs the user to declare response types using 'Method Response' and map possible responses from the backend. A table is present with one row: 'HTTP status regex' set to '-' and 'Method response status' set to '200'. Below the table is a link to 'Add integration response'.

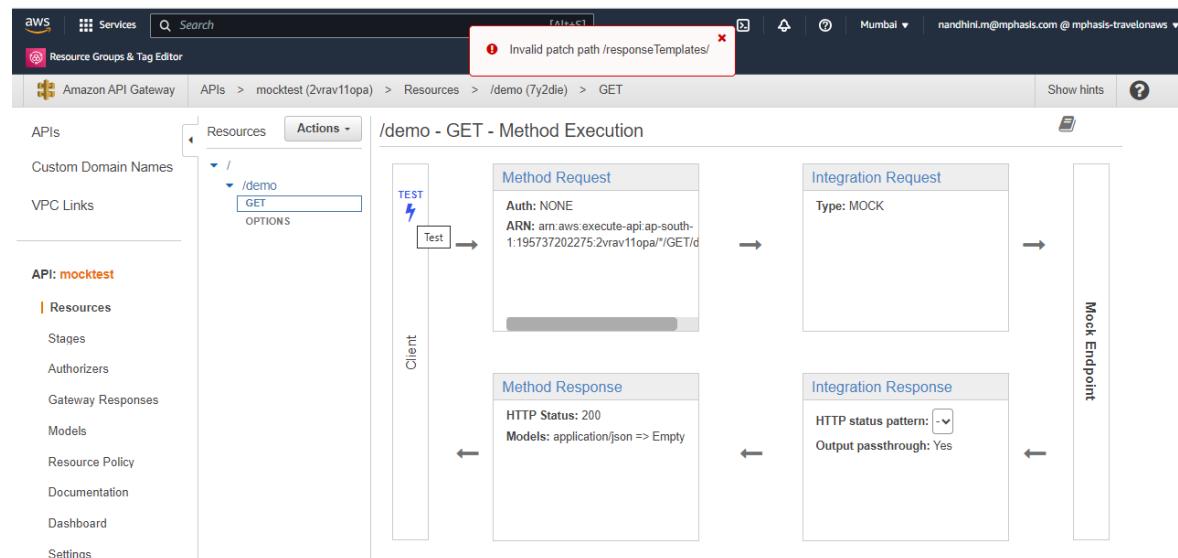
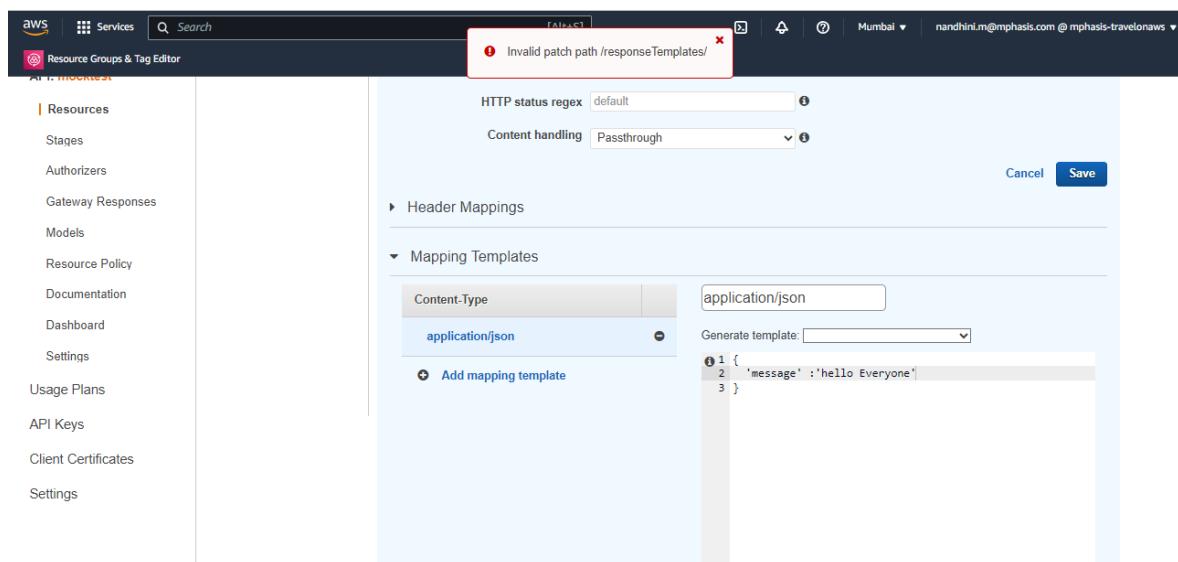
STEP 6: In the "Body Mapping Templates" section, click on "application/json" to configure a response in JSON format.

This screenshot shows the same AWS API Gateway interface as the previous one, but with a red box highlighting a validation error message: 'Invalid patch path /responseTemplates/'. The main content area shows the configuration for the '200' method response. Under 'Mapping Templates', there is a 'Content-Type' section where 'application/json' is selected. A 'Create' button is visible next to it. At the bottom right of the configuration area, there are 'Cancel' and 'Save' buttons.

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.

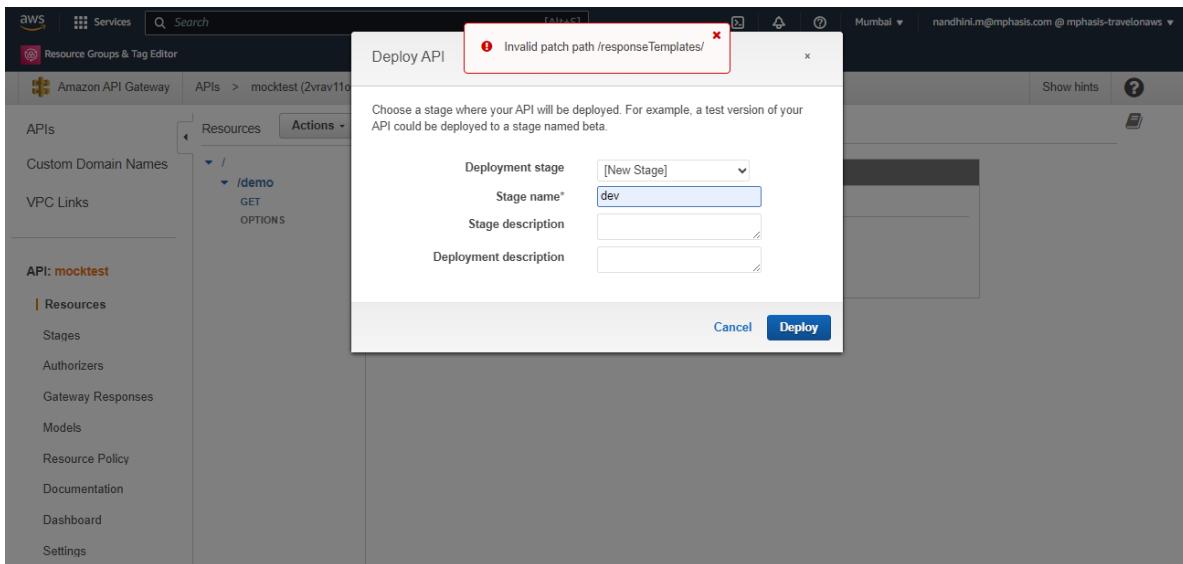


The screenshot shows the AWS API Gateway Resource Groups & Tag Editor interface. On the left, a sidebar lists various API management options like Stages, Authorizers, and Models. The main area is titled 'Resources' and contains sections for Query Strings, Headers, Stage Variables, Client Certificate, Request Body, and Logs. A red error box at the top right says 'Invalid patch path /responseTemplates/'. Below the error, there are input fields for Query Strings, Headers, and Logs, and a 'Test' button.

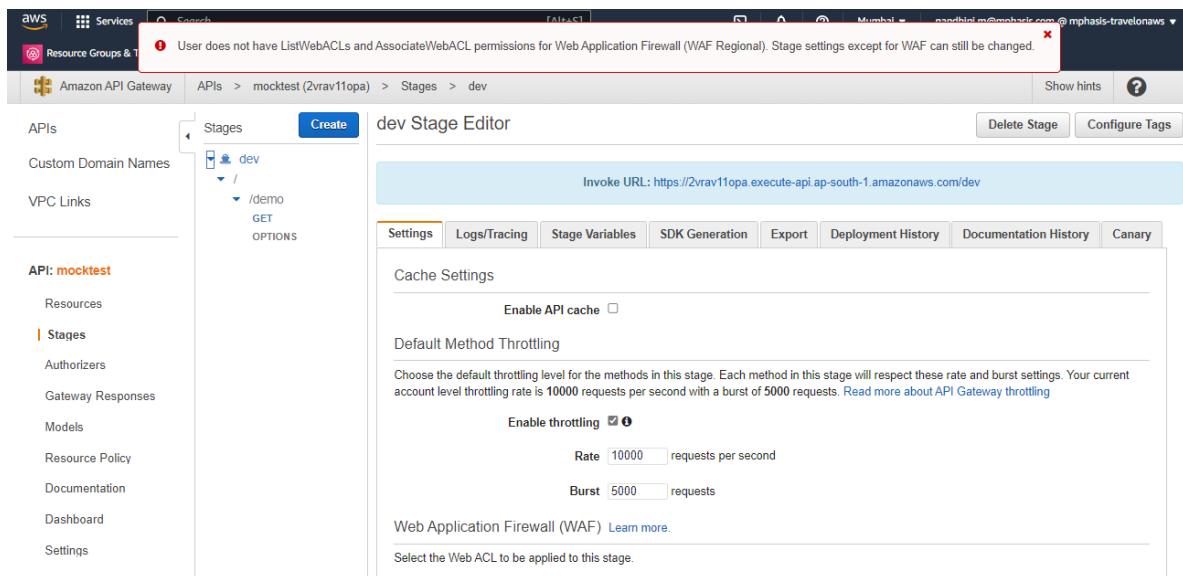
STEP 8: After configuring the mock integration response, deploy your API to make it accessible via a URL. Click on "Deploy API" under the "Actions" menu.

The screenshot shows the AWS API Gateway Actions menu for a specific resource. The 'Actions' dropdown is open, showing options like Create Method, Create Resource, Enable CORS, Edit Resource Documentation, Delete Resource, Deploy API, Import API, Edit API Documentation, and Delete API. The 'Deploy API' option is highlighted. The main panel shows the 'OPTIONS' configuration for the resource, including Mock Endpoint, Authorization (None), and API Key (Not required).

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.

The screenshot shows the AWS API Gateway console. The left sidebar is for the 'mocktest' API, with 'Resources' selected. The main pane shows the 'Enable CORS' configuration for the '/demo' resource. Under 'Gateway Responses for mocktest API', 'DEFAULT 4XX' and 'DEFAULT 5XX' are listed. Under 'Methods', 'GET' and 'OPTIONS' are checked. Under 'Access-Control-Allow-Methods', 'GET, OPTIONS' is listed. Under 'Access-Control-Allow-Headers', 'Content-Type,X-Amz-Date,Authorization' is listed. Under 'Access-Control-Allow-Origin', '*' is selected. A blue button at the bottom right says 'Enable CORS and replace existing CORS headers'.

The screenshot shows the AWS API Gateway console. The left sidebar is for the 'mocktest' API, with 'Stages' selected. The main pane shows the 'dev - GET - /demo' configuration. It includes an 'Invoke URL' field with the value 'https://2vrav11opa.execute-api.ap-south-1.amazonaws.com/dev/demo'. Below it, a note says 'Use this page to override the dev stage settings for the GET to /demo method.' There are two radio buttons for 'Settings': 'Inherit from stage' (selected) and 'Override for this method'. A blue 'Save Changes' button is at the bottom right.

The screenshot shows the Postman application interface. At the top, there is a browser-like header with the URL <https://2vrav11opa.execute-api.ap-south-1.amazonaws.com/dev/demo>. Below the header, the Postman interface displays a history of requests on the left and a detailed view of a specific request on the right.

History:

- Today:
 - GET https://2vrav11opa.execute-api.ap-south-1.amazonaws.com/dev/demo
 - GET https://2vrav11opa.execute-api.ap-south-1.amazonaws.com/dev/demo
- 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://reqres.in/api/users?page=1
 - GET https://uourgh6gj0.execute-api.ap-south-1.amazonaws.com/dev/demo
 - GET https://uourgh6gj0.execute-api.ap-south-1.amazonaws.com/dev/demo
- July 19:
 - GET https://6lmhy5xbpg.execute-api.ap-south-1.amazonaws.com/dev/demo
 - GET https://6lmhy5xbpg.execute-api.ap-south-1.amazonaws.com/dev/demo
 - GET https://6lmhy5xbpg.execute-api.ap-south-1.amazonaws.com/dev/demo
 - GET https://6lmhy5xbpg.execute-api.ap-south-1.amazonaws.com/dev/demo
 - GET https://6lmhy5xbpg.execute-api.ap-south-1.amazonaws.com/dev/demo
- July 4:
 - POST https://yy7dplk5k.execute-api.ap-south-1.amazonaws.com/dev/demo
 - GET https://yy7dplk5k.execute-api.ap-south-1.amazonaws.com/dev/demo
 - GET https://yy7dplk5k.execute-api.ap-south-1.amazonaws.com/dev/demo

Request Details:

Method: GET
URL: https://2vrav11opa.execute-api.ap-south-1.amazonaws.com/dev/demo

Headers:

- Content-Type: application/json
- User-Agent: PostmanRuntime/7.29.0
- Accept: */*

Body (Pretty):

```
1 {  
2   'message': 'hello Everyone'  
3 }
```

Response Headers:

- Status: 200 OK
- Time: 530 ms
- Size: 255 B

Response Body (Pretty):

```
1 {  
2   "message": "hello Everyone"  
3 }
```

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.

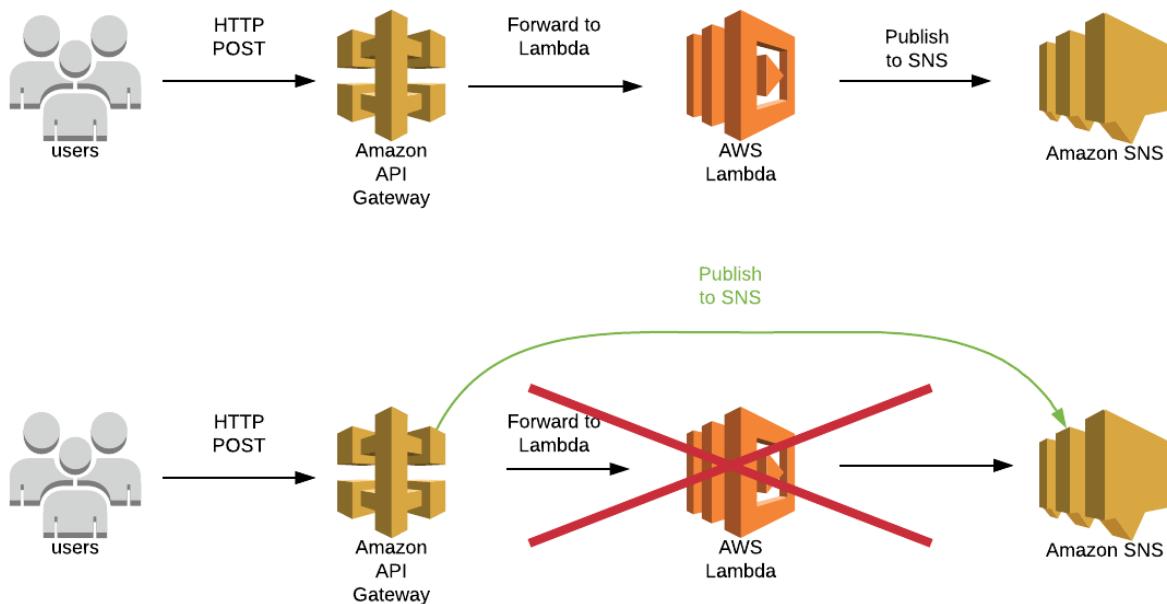
STEP 3: Under the "Integration" section, choose "AWS Service" as the integration type.

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.

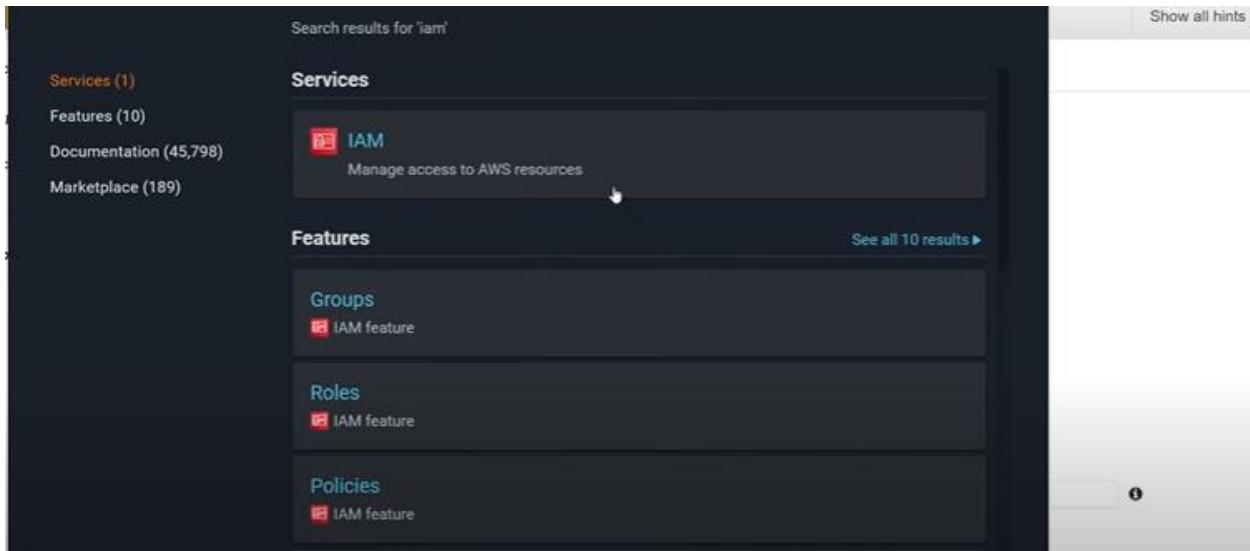
STEP 6: Test the integration using the "Test" button.

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



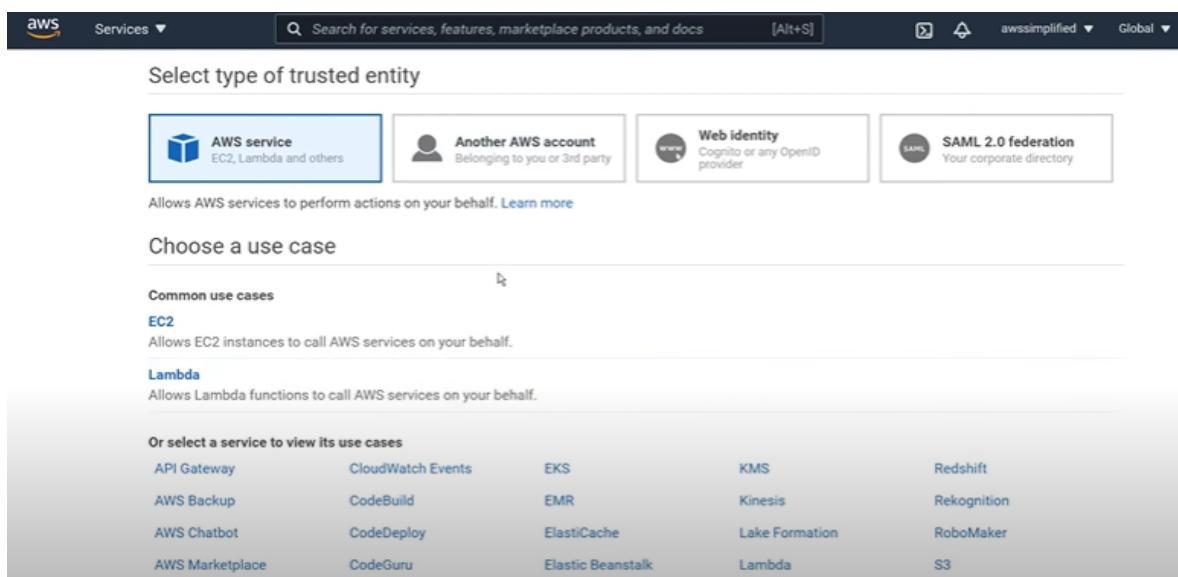
DIRECT INTEGRATION - API GATEWAY & DYNAMO DB

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).

The screenshot shows the AWS Services search interface. The search bar at the top contains the query 'Search for services, features, marketplace products, and docs'. Below the search bar is a grid of service names. The 'DynamoDB' service is highlighted with a blue background, indicating it is selected. Other services listed include DataSync, GuardDuty, Network Firewall, Systems Manager, DeepLens, Health Organizational View, OpsWorks, Texttract, Directory Service, Honeycode, Personalize, Transfer, Chime, IAM Access Analyzer, Purchase Orders, Trusted Advisor, CloudFormation, EC2, Inspector, QLDB, VPC, CloudHSM, EC2 - Fleet, IoT, RAM, WorkLink, CloudTrail, EC2 Auto Scaling, IoT SiteWise, RDS, WorkMail, CloudWatch Alarms, EC2 Image Builder, IoT Things Graph, CloudWatch Application Insights, and Insights.

Select your use case

Amazon DynamoDB Accelerator (DAX) - DynamoDB access
Allows DAX to call DynamoDB on your behalf.

DynamoDB - Global Tables
Allows DynamoDB to manage cross-region replication for Global Tables on your behalf.

DynamoDB Accelerator (DAX) - Cluster management
Allows DAX to manage resources within clusters on your behalf.

* Required Cancel Next: Permissions [A-Z] [a-z]

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.

The screenshot shows the AWS Policies search interface. The search bar at the top contains the query 'Search for services, features, marketplace products, and docs'. Below the search bar is a table titled 'Showing 1 result' showing a single policy entry. The table has columns for 'Policy name', 'Used as', and 'Description'. The policy listed is 'AmazonDynamoDBFullAccess', which is used as a 'Permissions policy (2)' and provides full access to Amazon DynamoDB via ...

The type of role that you selected requires the following policy.

Policy name	Used as	Description
AmazonDynamoDBFullAccess	Permissions policy (2)	Provides full access to Amazon DynamoDB via ...

Set permissions boundary

* Required Cancel Previous Next: Tags [A-Z] [a-z]

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

The type of role that you selected requires the following policy.

Filter policies ▾ Search Showing 1 result

Policy name	Used as	Description
DynamoDB	Full access	All resources

Allow (13 of 264 services) Show remaining 251

Service	Access level	Resource	Request condition
Application Auto Scaling	Limited: Read, Write	All resources	None
CloudWatch	Limited: List, Read, Write	Multiple	None
Data Pipeline	Limited: List, Read, Write	All resources	None
DynamoDB	Full access	All resources	None
DynamoDB Accelerator	Full access	All resources	None

Set permissions boundary

* Required Cancel Previous Next: Tags

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

Create role

Add tags (optional)

IAM tags are key-value pairs you can add to your role. Tags can include user information, such as an email address, or can be descriptive, such as a job title. You can use the tags to organize, track, or control access for this role. [Learn more](#)

Key	Value (optional)	Remove
Add new key		

You can add 50 more tags.

Cancel Previous Next: Review

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

Create role

Review

Provide the required information below and review this role before you create it.

Role name* Use alphanumeric and '+_@-' characters. Maximum 64 characters.

Role description

Maximum 1000 characters. Use alphanumeric and '+_@-' characters.

Trusted entities AWS service: dax.amazonaws.com

Policies AmazonDynamoDBFullAccess

Permissions boundary Permissions boundary is not set

IAM Services ▾ Q Search for services, features, marketplace products, and docs [Alt+S] awssimplified ▾ Global ▾ Support ▾

Create role Delete role

Role name	Trusted entities	Last activity
amplify-login-lambda-ccbdafe6	AWS service: lambda	None
AWSServiceRoleForApplicationAuto...	AWS service: dynamodb.application-autoscali...	Today
AWSServiceRoleForAutoScaling	AWS service: autoscaling (Service-Linked role)	114 days
AWSServiceRoleForDynamoDBRepli...	AWS service: replication.dynamodb (Service-Li...	82 days
AWSServiceRoleForECS	AWS service: ecs (Service-Linked role)	116 days
AWSServiceRoleForSupport	AWS service: support (Service-Linked role)	None
AWSServiceRoleForTrustedAdvisor	AWS service: trustedadvisor (Service-Linked r...	None
DynamoAPIGateway	AWS service: apigateway	Today
DynamoDBAPIGatewayRole	AWS service: dax	None
ecsInstanceRole	AWS service: ec2	114 days
nimbleEvents	AWS service: nimble	None

The screenshot shows the AWS Identity and Access Management (IAM) service. On the left, there's a navigation sidebar with options like Dashboard, Access management, Roles, Policies, and Access reports. The 'Roles' section is currently selected. The main area is titled 'Summary' for a role named 'DynamoDBAPIGatewayRole'. It displays various details such as Role ARN, Role description, Instance Profile ARNs, Path, Creation time, Last activity, and Maximum session duration. Below this, there are tabs for Permissions, Trust relationships, Tags, Access Advisor, and Revoke sessions. The 'Permissions' tab is active, showing a list of applied policies. One policy, 'AmazonDynamoDBFullAccess', is listed as an AWS managed policy. There are buttons for 'Attach policies' and 'Add inline policy'.

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:

STEP9: In the "Summary" tab of the role, click on the "Edit trust relationship" button.

This screenshot shows the same IAM role summary page as the previous one, but the 'Trust relationships' tab is now active. A message indicates that trusted entities can assume the role and provides a link to view the policy document. Below this, there are sections for 'Trusted entities' and 'Conditions'. The 'Trusted entities' section lists 'The identity provider(s) dax.amazonaws.com'. The 'Conditions' section states that there are no conditions associated with this role.

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

The screenshot shows the AWS IAM 'Edit Trust Relationship' page. At the top, there's a navigation bar with the AWS logo, 'Services ▾', a search bar ('Search for services, features, marketplace products, and docs'), and other account-related links like 'awssimplified ▾', 'Global ▾', and 'Support ▾'. Below the navigation, the title 'Edit Trust Relationship' is displayed. A sub-instruction says 'You can customize trust relationships by editing the following access control policy document.' The main area is titled 'Policy Document' and contains the following JSON code:

```
1+ {
2  "Version": "2012-10-17",
3  "Statement": [
4    {
5      "Sid": "",
6      "Effect": "Allow",
7      "Principal": {
8        "Service": "dax.amazonaws.com"
9      },
10     "Action": "sts:AssumeRole"
11   }
12 ]
13 }
```

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.

```
1 * {
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Sid": "",
6             "Effect": "Allow",
7             "Principal": {
8                 "Service": "apigateway.amazonaws.com"
9             },
10            "Action": "sts:AssumeRole"
11        }
12    ]
13 }
```

Cancel Update Trust Policy

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

Identity and Access Management (IAM)

Roles > DynamoDBAPIGatewayRole

Summary

Role ARN: arn:aws:iam::755314965794:role/DynamoDBAPIGatewayRole

Role description: Edit

Instance Profile ARNs: [Edit](#)

Path: /

Creation time: 2020-12-19 12:14 EST

Last activity: Not accessed in the tracking period

Maximum session duration: 1 hour [Edit](#)

Permissions Trust relationships Tags Access Advisor Revoke sessions

You can view the trusted entities that can assume the role and the access conditions for the role. [Show policy document](#)

[Edit trust relationship](#)

Trusted entities
The following trusted entities can assume this role.

Trusted entities
The identity provider(s) [apigateway.amazonaws.com](#)

Conditions
The following conditions define how and when trusted entities can assume the role.

There are no conditions associated with this role.

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

The screenshot shows the AWS DynamoDB Dashboard. On the left, there's a sidebar with 'Tables', 'Update settings', 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Reserved capacity', and 'Settings'. Below that is a 'DAX' section with 'Clusters' and 'Subnet groups'. The main area is titled 'Dashboard' and contains sections for 'Alarms (0)' and 'DAX clusters (0)'. To the right, there's a 'Create resources' section with a large orange 'Create table' button. Below it, there's a brief description of Amazon DynamoDB Accelerator (DAX) and a 'Create DAX cluster' button.

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.

Table details Info

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

Table name
This will be used to identify your table.

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

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

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

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.

DynamoDB

Tables

- Dashboard
- Tables
- Update settings
- Explore items
- PartiQL editor
- Backups
- Exports to S3
- Imports from S3
- Reserved capacity
- Settings

DynamoDB > Tables

Tables (34) Info

<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode
<input type="checkbox"/>	Customers	Active	customerId (\$)	date (\$)	0	Off	Provisioned (5)	Provisioned (5)

STEP 15: Create a REST API

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

Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

REST WebSocket

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

New API Clone from existing API Import from Swagger or Open API 3 Example API

Settings

Choose a friendly name and description for your API.

API name*

Description

Endpoint Type

* Required

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.

APIs > Customerapi (3eg836eawh) > Resources > / (mhwvqfe648) > Create

New Child Resource

Use this page to create a new child resource for your resource.

Configure as proxy resource

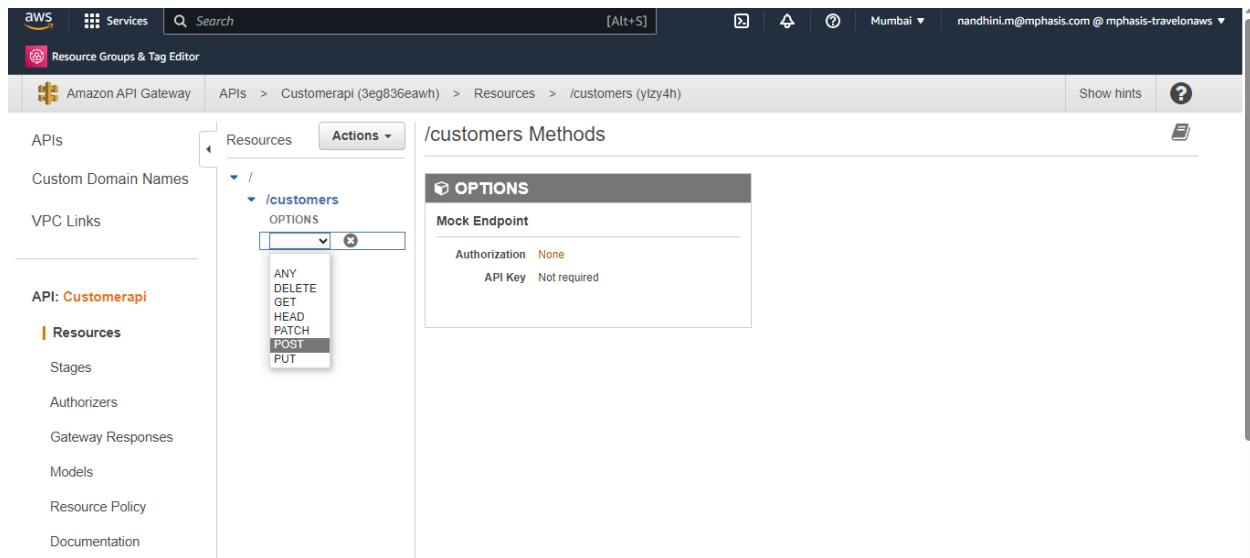
Resource Name*

Resource Path*

You can add path parameters using brackets. For example, the resource path {username} represents a path parameter called 'username'. Configuring /{proxy+} as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.

Enable API Gateway CORS

* Required



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.

Screenshot of the AWS API Gateway console showing the setup for a POST method on the '/customers' resource.

APIs > **CustomersApi (haatwbporl)** > **Resources** > **/customers (efu85o)** > **POST**

Actions > **/customers - POST - Setup**

Choose the integration point for your new method.

Integration type: AWS Service AWS Lambda

- Lambda Function
- HTTP
- Mock
- VPC Link

AWS Region: us-east-1

AWS Service: DynamoDB

AWS Subdomain:

HTTP method: POST

Action Type: Use action name

Use path override

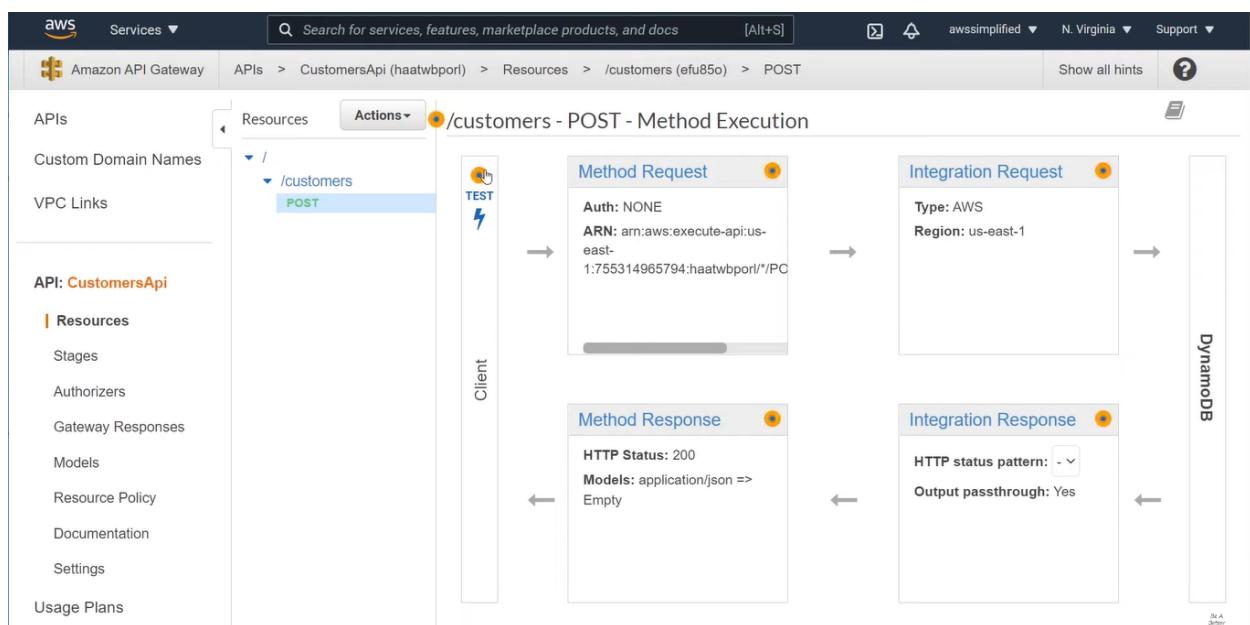
Action: PutItem

Execution role: arn:aws:iam::755314965794:role/DynamoDBAPIGatewayRole

Content Handling: Passthrough

Use Default Timeout:

Save



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.

The screenshot shows the AWS API Gateway console. On the left, the navigation pane is visible with sections like APIs, Custom Domain Names, VPC Links, and the current API named 'CustomersApi'. Under 'Resources', there's a tree structure for '/customers' with a 'POST' method selected. A context menu is open over this method, with the 'Actions' dropdown expanded. The 'API ACTIONS' section contains the 'Deploy API' option, which is highlighted. To the right of the tree, four boxes represent the flow: 'Method Request', 'Integration Request', 'Method Response', and 'Integration Response'. The 'Integration Request' box shows 'Type: AWS' and 'Region: us-east-1'. The 'Integration Response' box shows 'HTTP status pattern: -' and 'Output passthrough: Yes'. The 'Method Request' and 'Method Response' boxes show basic details like ARN and HTTP status.

This screenshot shows the 'Deploy API' dialog box overlaid on the API Gateway interface. The dialog has a title 'Deploy API' with a close button. It contains fields for 'Deployment stage' (set to '[New Stage]'), 'Stage name' (set to 'test'), and 'Stage description'. Below these is a 'Deployment description' field with a placeholder. At the bottom right of the dialog are 'Cancel' and 'Deploy' buttons, with 'Deploy' being the active button. The background shows the same API configuration as the previous screenshot, with the 'Integration Request' and 'Integration Response' sections partially visible.

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.

DynamoDB

Tables

Customers

Items

Item count: 0

Table size: 0 bytes

Average item size: 0 bytes

Items preview (1)

	customer...	date	name
1	2020	John Doe	

DIRECT INTEGRATION – API GATEWAY & S3

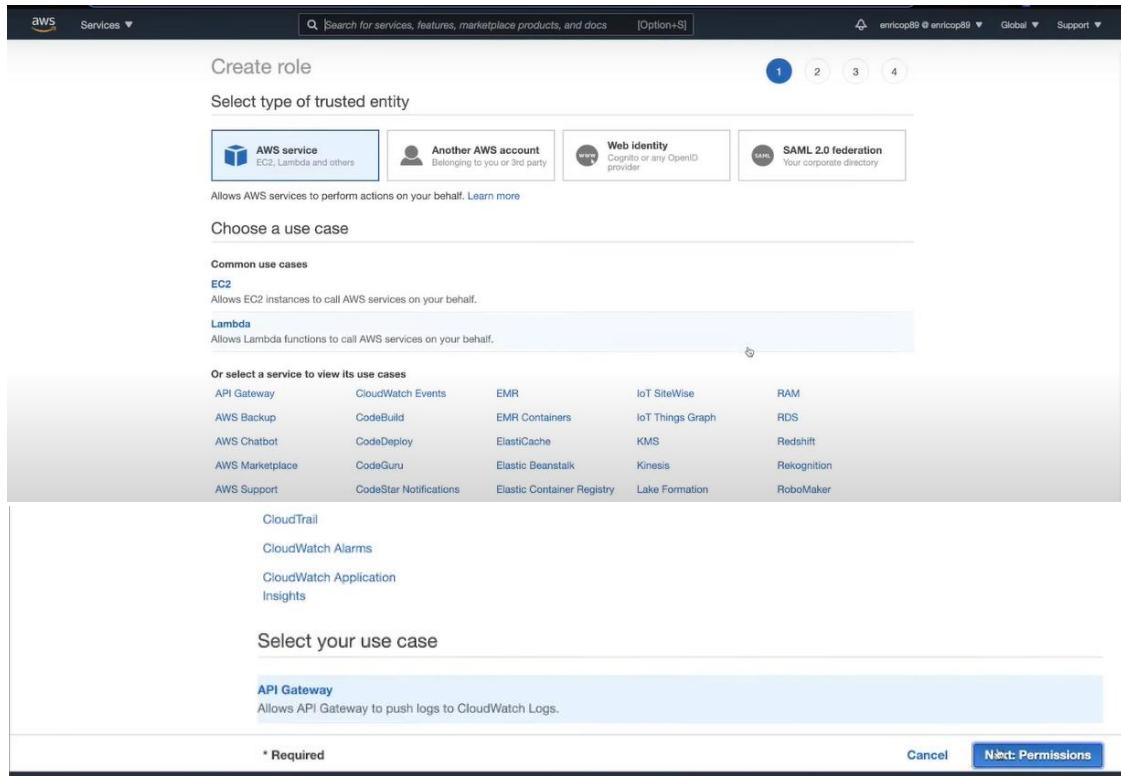
STEP 1: In the IAM dashboard, click on "Roles" in the left-hand navigation pane. Click the "Create role" button.

Identity and Access Management (IAM)

Roles (52) Info

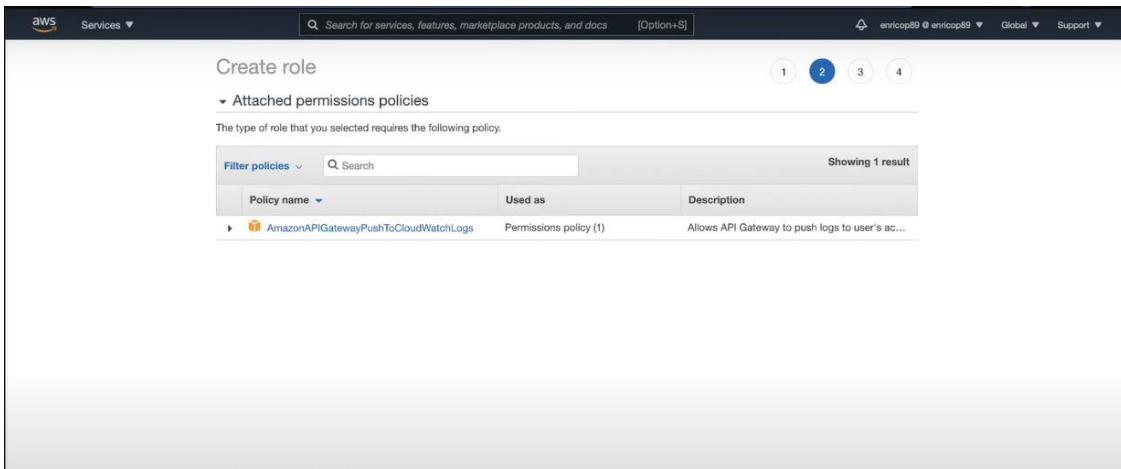
Role name	Trusted entities	Last act...
amplify-reactamplified-dev-71403-authRole	Identity Provider: cognito-identity.amazonaws.com	-
amplify-reactamplified-dev-71403-authRole-1dp	AWS Service: lambda	288 days ago
amplify-reactamplified-dev-71403-unauthRole	Identity Provider: cognito-identity.amazonaws.com	-
amplifyconsole-backend-role	AWS Service: amplify	288 days ago
apigateway-sample-1	AWS Service: apigateway	14 hours ago
aws-elasticbeanstalk-ec2-role	AWS Service: ec2	-
aws-elasticbeanstalk-service-role	AWS Service: elasticbeanstalk	-
AWSServiceRoleForAPIGateway	AWS Service: ops.apigateway (Service-Linked Role)	-

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"

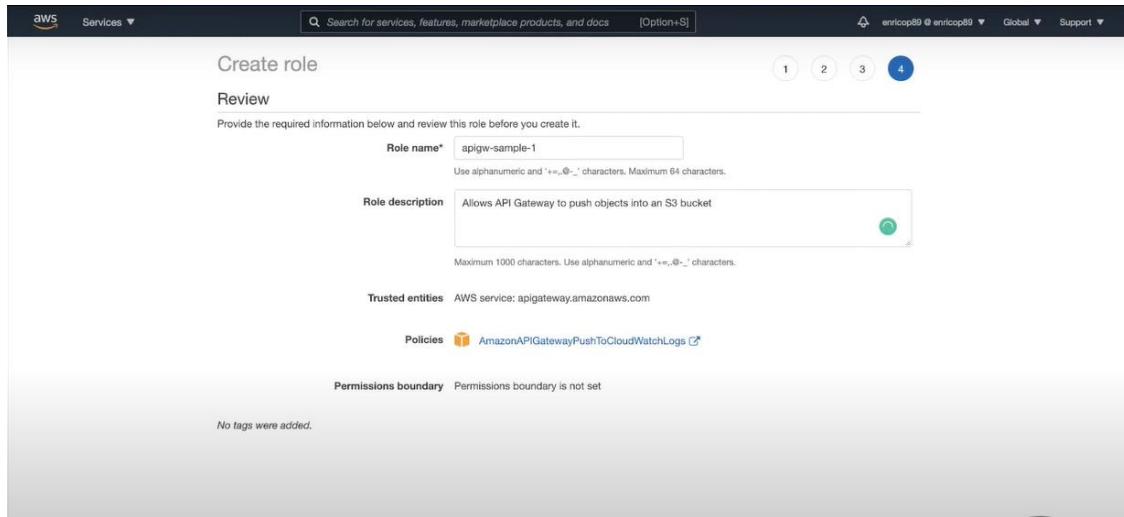


STEP 3: In the "Permissions" step, you can choose an existing policy that grants permissions to push logs to CloudWatch Logs.

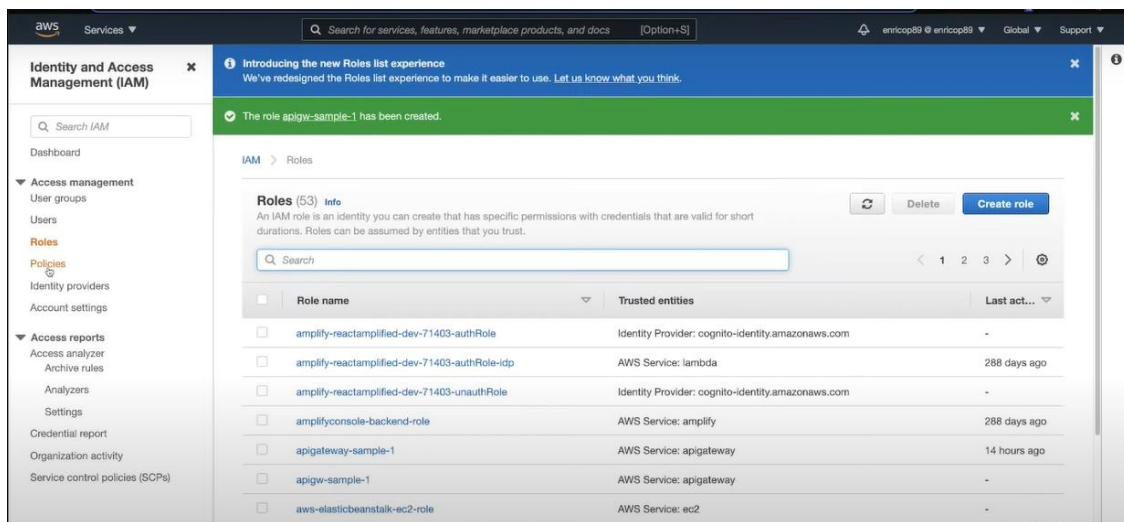
- Click the "Attach permissions policies" button.
- In the "Filter policies" search box, type "CloudWatchLogs" to find the built-in policy "CloudWatchLogs_FullAccess." Select this policy if it provides the necessary permissions



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.

The screenshot shows the AWS IAM Policies list page. The left sidebar is titled 'Identity and Access Management (IAM)' and includes sections for 'Access management' (User groups, Users, Roles, Policies), 'Access reports' (Access analyzer, Archive rules, Analyzers, Settings, Credential report, Organization activity, Service control policies (SCPs)), and a search bar for 'Search IAM'. The main content area is titled 'Policies (914) Info' and contains a table with columns: Policy name, Type, Used as, and Description. The table lists various AWS Lambda execution roles, all categorized as 'Customer managed' and used as 'Permissions policy (1)'. A search bar at the top of the table allows filtering by property or policy name.

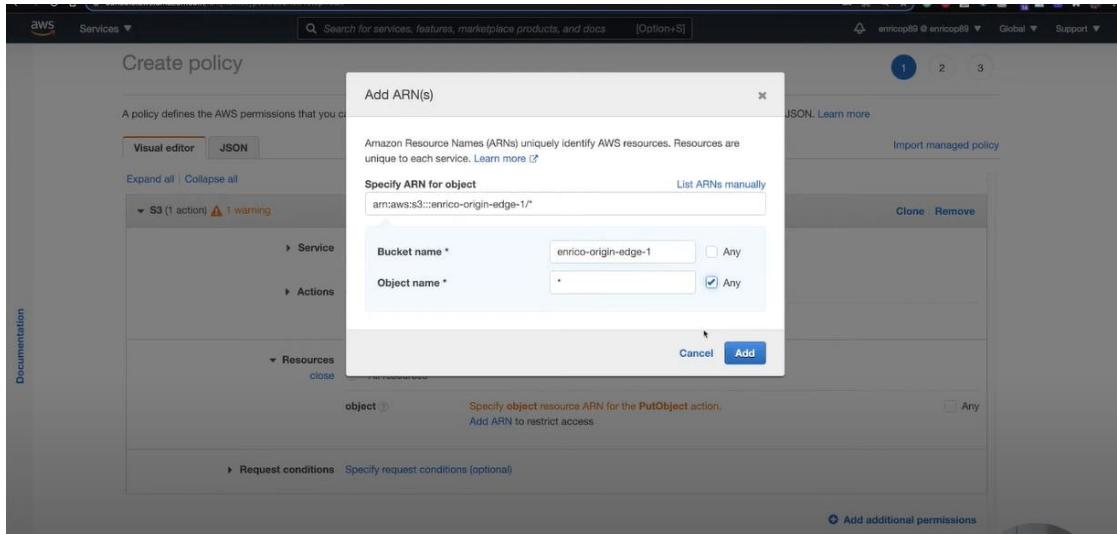
STEP 6: In the "Create Policy" wizard, select the "Visual editor" tab or choose the "JSON" tab to create a custom policy.

In Visual Editor:

- Select "S3" from the list of services.
- Select the "PutObject" as an action. Because you want to grant "Write" permissions for "PutObject."
- 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`.

The screenshot shows the 'Create policy' wizard in the AWS IAM console. The left sidebar has a 'Documentation' link. The main area is titled 'Create policy' and shows a progress bar with step 1 completed. Below it, a note states: 'A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. Learn more'. There are tabs for 'Visual editor' (which is selected) and 'JSON'. Under the 'Visual editor' tab, there's a 'Service' dropdown set to 'S3', an 'Actions' dropdown set to 'Write', and a 'PutObject' action listed. Below that, a 'Resources' section shows 'Specific' selected over 'All resources'. An 'object' field is present with placeholder text: 'Specify object resource ARN for the PutObject action.' and a note: 'Add ARN to restrict access'. At the bottom, there's a 'Request conditions' section and a 'Add additional permissions' link.

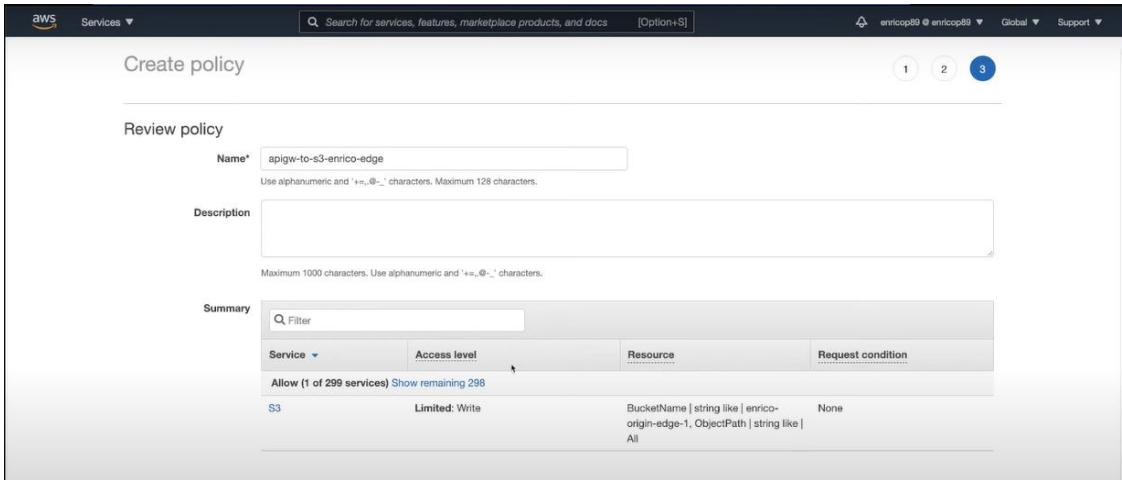


In the JSON editor, define the policy. Below is an example of a policy that allows Amazon API Gateway to perform the "s3: PutObject" action on a specific S3 bucket and object.

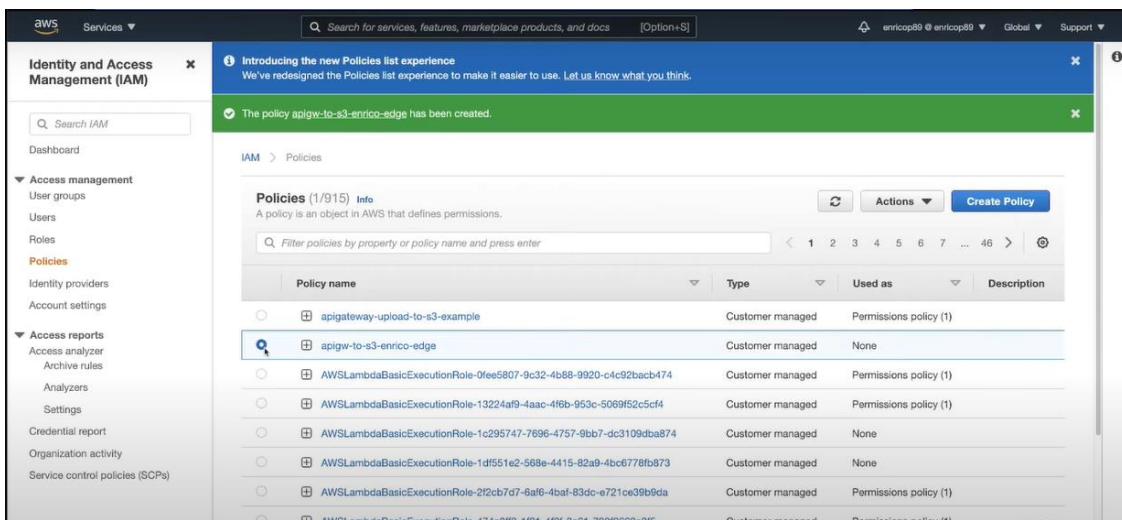
```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "s3:PutObject",  
      "Resource": "arn:aws:s3:::YOUR_BUCKET_NAME/YOUR_OBJECT_NAME"  
    }  
  ]  
}
```

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."

The screenshot shows the AWS Identity and Access Management (IAM) Policies list. A green notification bar at the top right says: "Introducing the new Policies list experience. We've redesigned the Policies list experience to make it easier to use. Let us know what you think." Below this, a message says: "The policy apigw-to-s3-enrico-edge has been created." The main table lists policies with columns for Policy name, Type, and Description. The policy 'apigw-to-s3-enrico-edge' is selected, indicated by a blue outline. The 'Actions' column for this row contains three buttons: 'Attach', 'Detach', and 'Delete'. Other policies listed include 'apigateway-upload-to-s3-example', 'AWSLambdaBasicExecutionRole-0fee5807-9c32-4b88-9920-c4c92bacb474', 'AWSLambdaBasicExecutionRole-13224af9-4aac-4fb6-953c-5069f52c5cf4', 'AWSLambdaBasicExecutionRole-1c295747-7696-4757-9bb7-dc3109dba874', 'AWSLambdaBasicExecutionRole-1df5f51e2-68ae-4415-82a9-4bc6778fb873', 'AWSLambdaBasicExecutionRole-2f2cb7d7-6af6-4ba1-83dc-e721ce39b9da', and 'AWSLambdaBasicExecutionRole-474a2ff3-1f01-4f2f-8c61-723f2662a3f5'. The 'Type' column shows most are 'Customer managed' except for the first one which is 'Permissions policy'.

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.

The screenshot shows the 'Attach policy' dialog box. At the top, it says 'Attach policy' and 'Attach the policy to users, groups, or roles in your account'. Below this is a search bar with 'apigw' typed in. A filter dropdown is set to 'Filter' and 'Name'. A table lists a single item: 'apigw-sample-1' under the 'Role' column. At the bottom right are 'Cancel' and 'Attach policy' buttons.

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

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

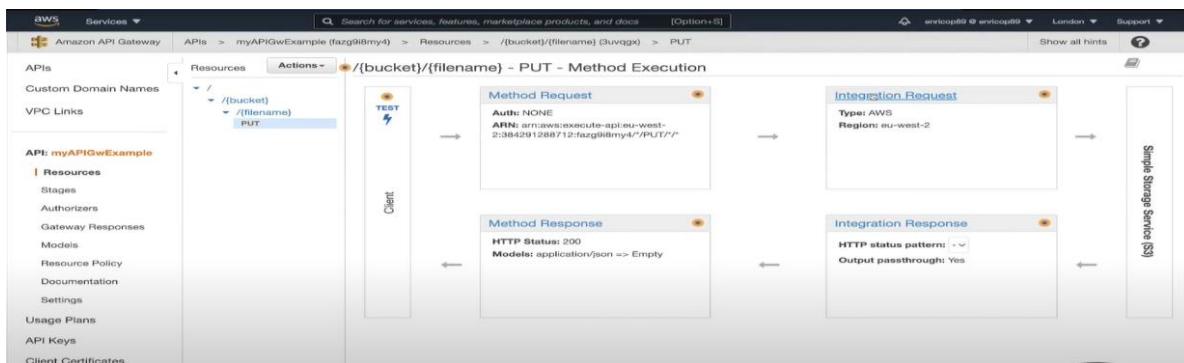
STEP 14: In the left sidebar, click on the "Resources" section. Under Actions menu, click "Create" to create a new resource. Enter a resource name, e.g., "/bucket" or any path you prefer. Click "Create Resource".

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:** In 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.

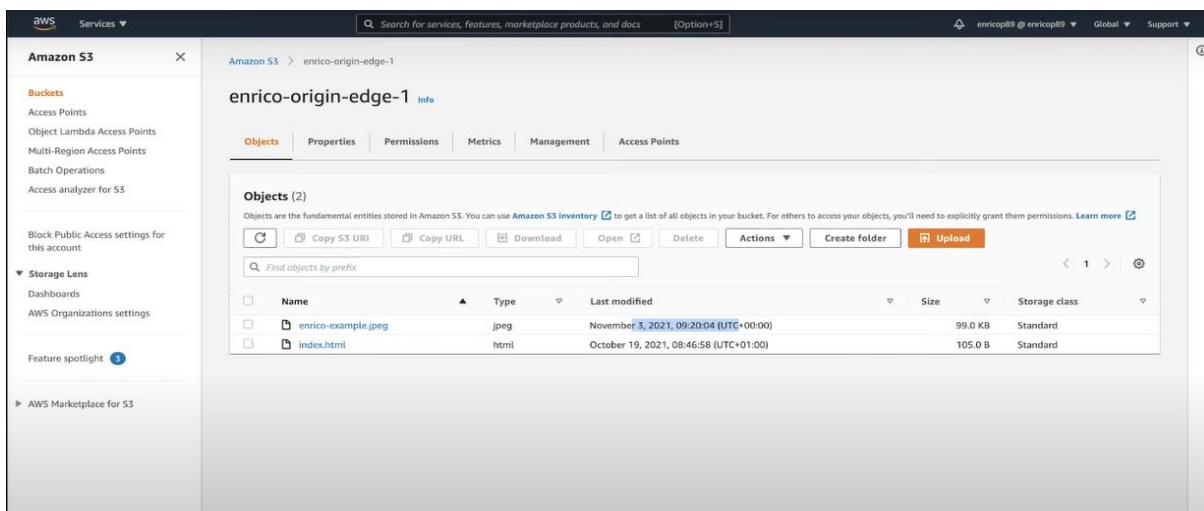
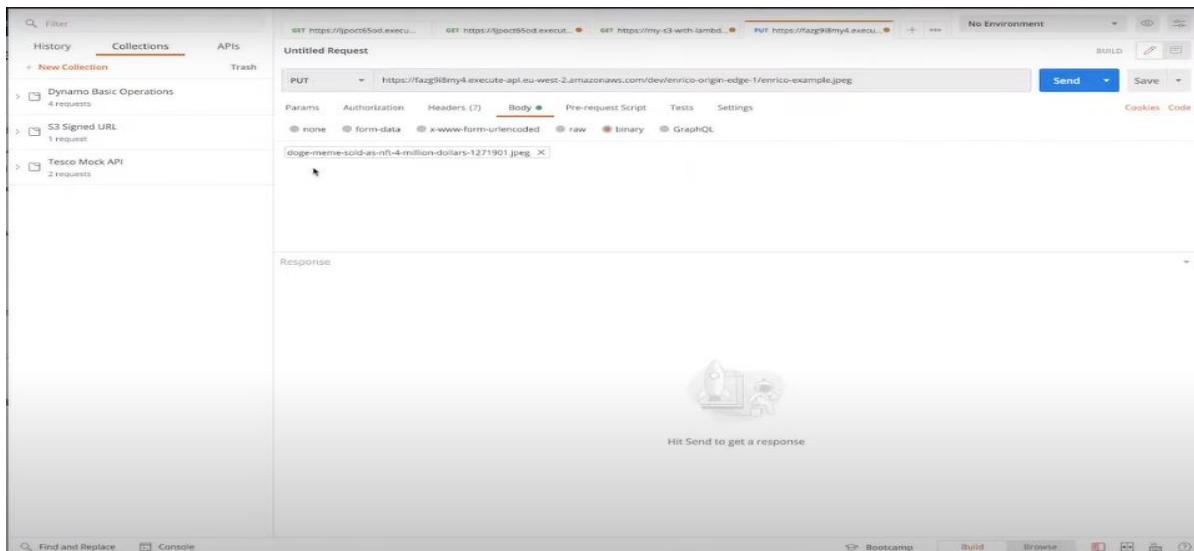
STEP 17: To enable binary media types, go to the API Gateway settings. Under "Binary Media Types," add the appropriate content types for binary data you want to support. In your case, for `image/jpeg`, you would add `image/jpeg`.

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

The image consists of three vertically stacked screenshots of the AWS API Gateway console, illustrating the deployment process for an API named "myAPIGwExample".

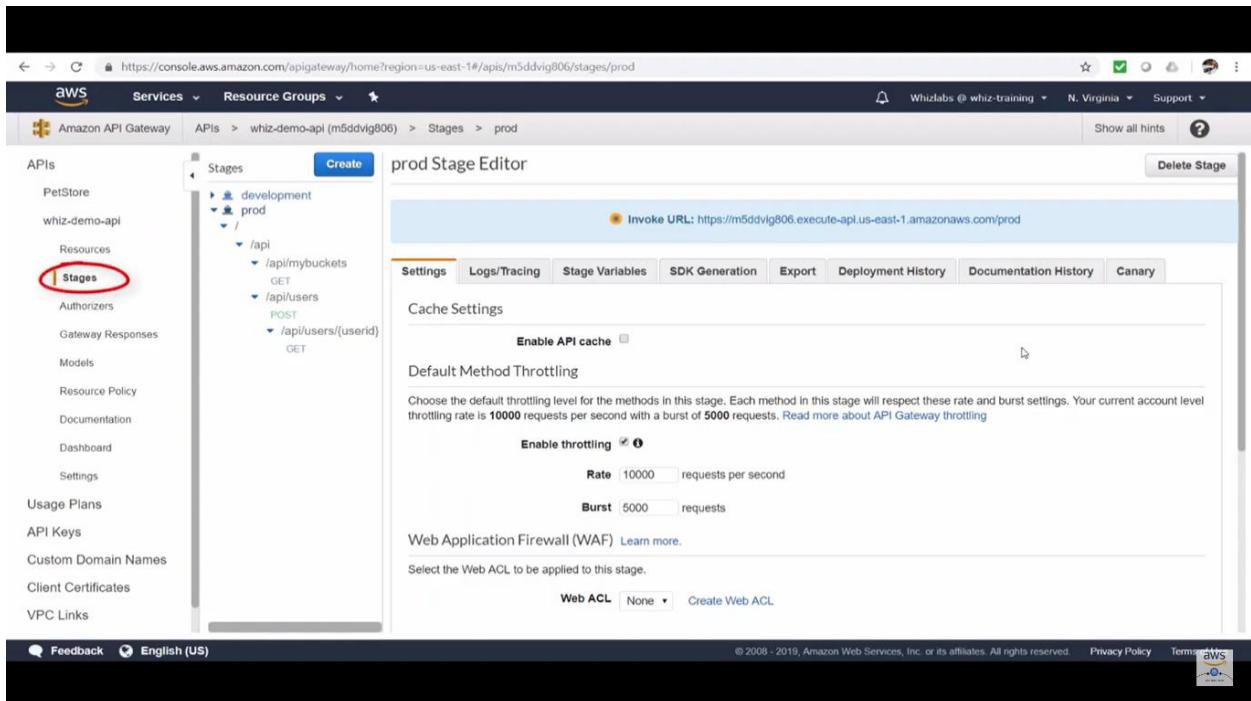
- Screenshot 1:** Shows the "Method Request" and "Integration Request" configurations for the PUT method of the "/{bucket}/{filename}" resource. The Method Request shows "Auth: NONE" and the Integration Request specifies "Type: AWS", "Paths: key, bucket", and "Region: eu-west-2".
- Screenshot 2:** A modal dialog titled "Deploy API" is open, prompting the user to choose a deployment stage. The "Deployment stage" dropdown is set to "New Stage" and the "Stage name" input field contains "dev".
- Screenshot 3:** The "dev Stage Editor" is shown, where the "Invoke URL" is displayed as <https://fazg9l8my4.execute-api.eu-west-2.amazonaws.com/dev>. The stage editor includes tabs for Settings, Logs/Tracing, Stage Variables, SDK Generation, Export, Deployment History, Documentation History, and Canary. Under the Settings tab, "Cache Settings" and "Default Method Throttling" are configured. Throttling settings show a rate of 10000 requests per second and a burst of 5000 requests. A "Web Application Firewall (WAF)" section is also present.

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.

The screenshot shows the AWS API Gateway Stage Editor for the 'prod' stage. On the left sidebar, under the 'Stages' section, the 'prod' stage is selected. In the main 'prod Stage Editor' area, the 'Cache Settings' tab is active. The 'Enable API cache' checkbox is checked and highlighted with a red circle. Below it, there are sections for 'Default Method Throttling' (Rate: 10000 requests per second, Burst: 5000 requests) and 'Web Application Firewall (WAF)' (Web ACL: None). At the bottom of the editor, there are links for 'Feedback', 'English (US)', and AWS terms and conditions.

This screenshot is identical to the one above, showing the AWS API Gateway Stage Editor for the 'prod' stage. The 'Cache Settings' tab is active, and the 'Enable API cache' checkbox is checked. A yellow callout box points to the note: 'Enabling API cache increases cost and is not covered by the free tier. See pricing for more details.' Below this, a dropdown menu for 'Cache capacity' is open, showing options: 0.5GB, 1.6GB, 6.1GB, 13.5GB, 28.4GB, 58.2GB, 118GB, and 237GB. The '0.5GB' option is selected and highlighted with a blue background. The rest of the interface is identical to the first screenshot.

The screenshot shows the 'prod Stage Editor' in the Amazon API Gateway console. On the left, a sidebar lists various API resources like PetStore, whiz-demo-api, and Stages. The 'Stages' section is selected. The main area displays the 'prod' stage's resources, which include a root resource '/' with two methods: 'GET /api/mybuckets' and 'POST /api/users/{userid}'. Below these, there are tabs for 'Settings', 'Logs/Tracing', 'Stage Variables', 'SDK Generation', 'Export', 'Deployment History', 'Documentation History', and 'Canary'. The 'Settings' tab is active, showing the 'Cache Settings' configuration. Key settings include enabling API cache, setting a cache capacity of 0.5GB, encrypting cache data, and defining a TTL of 300 seconds. There are also options for per-key cache invalidation and handling unauthorized requests.

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

The screenshot shows the AWS API Gateway interface. On the left, a sidebar lists various API resources like PetStore and whiz-demo-api. The main area shows the 'prod' stage under 'Stages'. The 'Settings' tab is selected. Under 'Cache Settings', 'Enable API cache' is checked. In 'Default Method Throttling', the rate is set to 10000 requests per second and the burst is set to 5000 requests. A note indicates the account level throttling rate is 10000 requests per second with a burst of 5000 requests. Below this, there's a 'Web Application Firewall (WAF)' section with a 'Learn more.' link and a 'Web ACL' dropdown set to 'None'. At the bottom, there are sections for 'Client Certificate' (with a dropdown set to 'None') and 'Tags' (with a table for adding key-value pairs). A 'Save Changes' button is at the bottom right.

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:

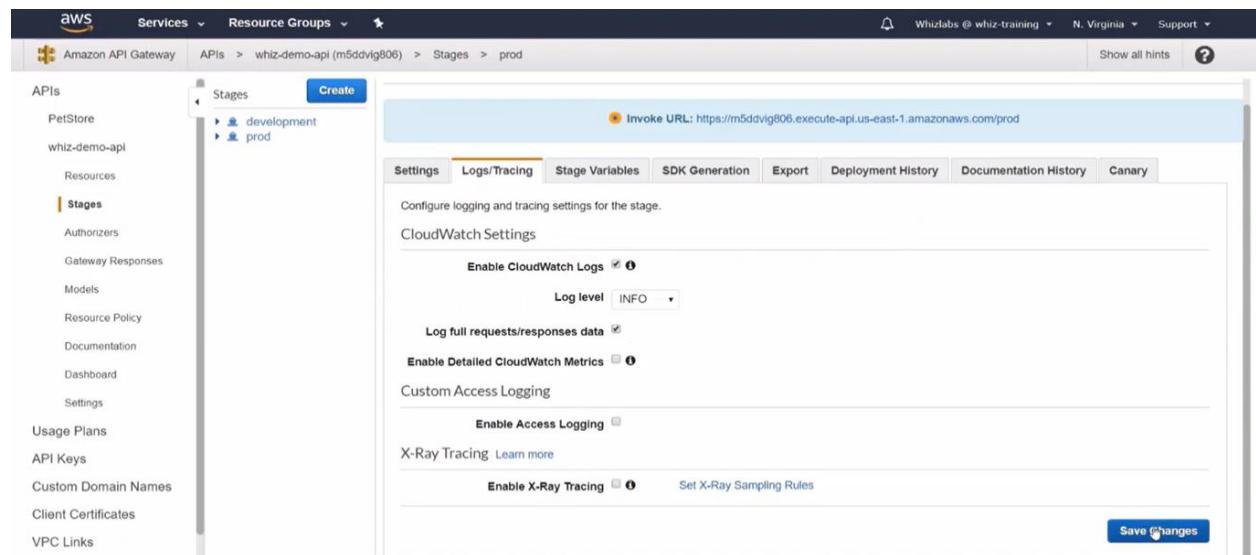
- OFF: No logs are generated.
- ERROR: Only error-related logs are generated.
- INFO: Informational logs are generated in addition to errors.
- DEBUG: Detailed debugging logs are generated.

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.

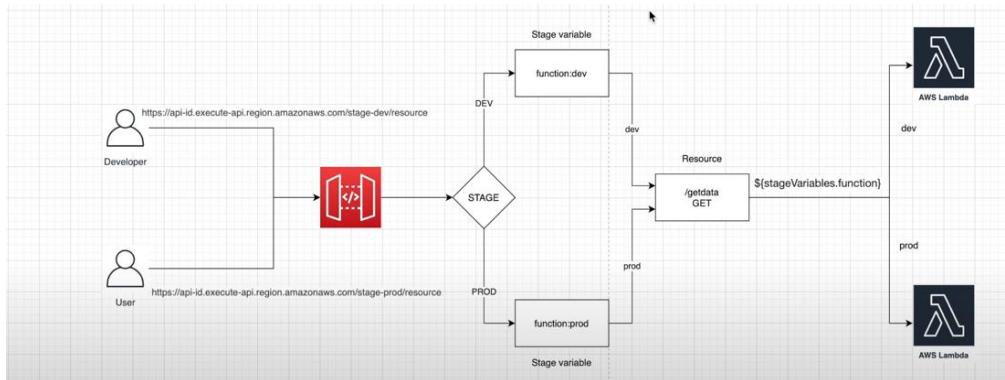


Screenshot of the AWS CloudWatch Log Groups interface for the 'aws/lambda/get-users-from-dynamodb' log group. The log entries show various API requests and responses, primarily in JSON format, with timestamps ranging from 2019-02-10 to 2019-02-09. The log entries include fields like 'Time (UTC +0:00)', 'Message', and detailed log data.

Screenshot of the AWS CloudWatch Metrics Filter interface. It shows a list of log groups under the prefix '/aws/lambda'. One specific log group, 'API-Gateway-Execution-Logs_m5ddvig806/prod', is highlighted with a red oval.

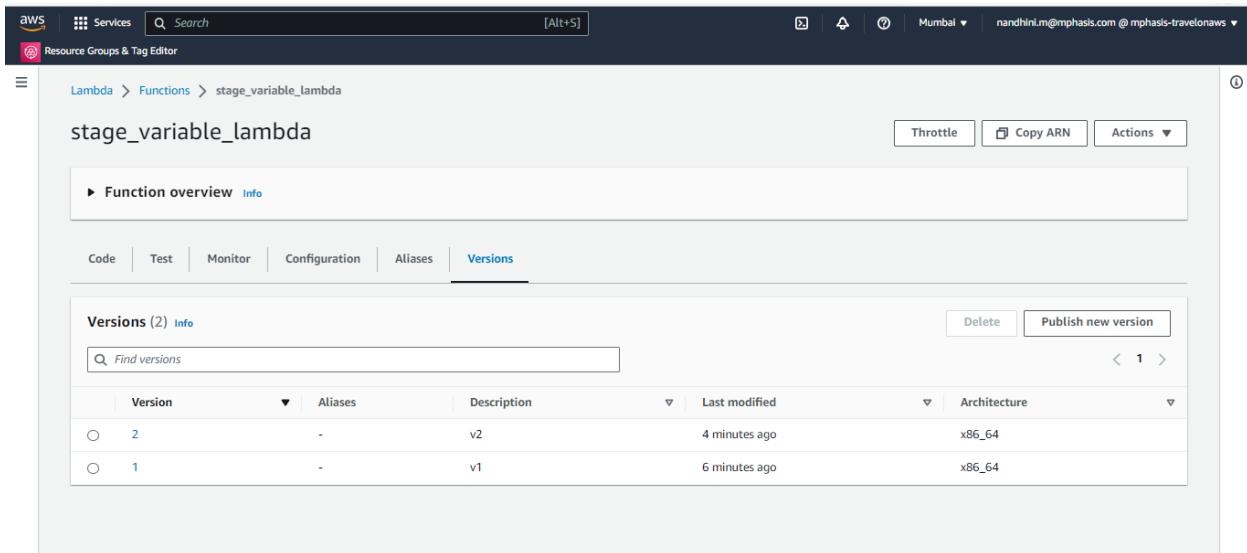
Screenshot of the AWS CloudWatch Streams interface for the 'Streams for API-Gateway-Execution-Logs...' stream. It lists multiple log streams, each with a unique ID and its last event time. One specific log stream, '274ad4786c3abc9a99a05e85867d9a4', is highlighted with a red oval.

4.STAGE VARIABLE



STEP 1: Create a Sample Lambda Function

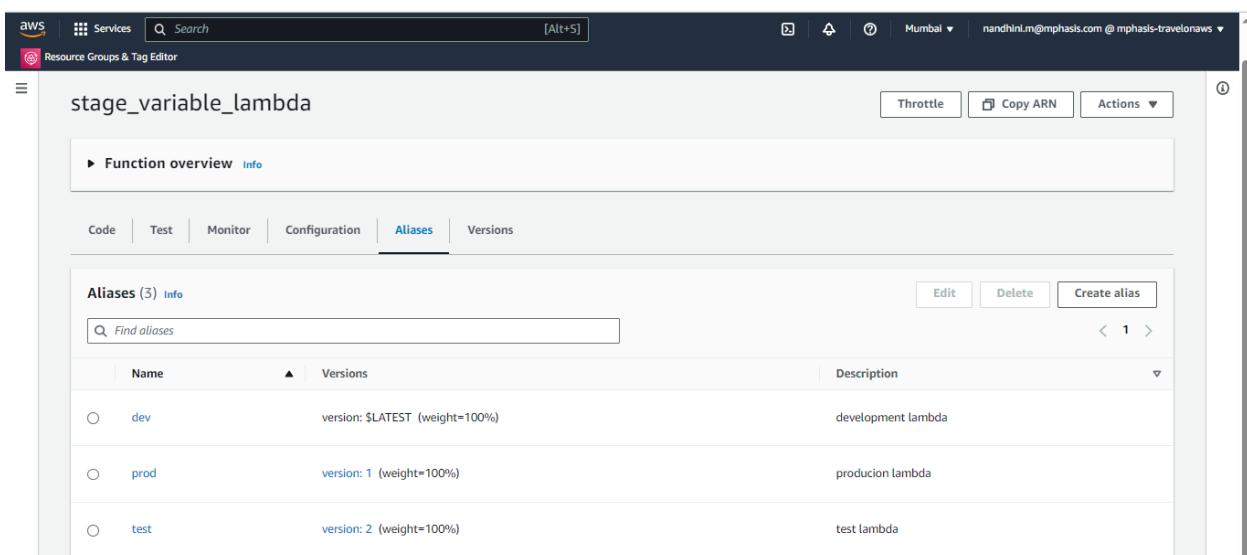
Step 2: Publish Different Versions



The screenshot shows the AWS Lambda console for the function 'stage_variable_lambda'. The 'Versions' tab is selected, displaying two versions:

Version	Aliases	Description	Last modified	Architecture
v2	-	v2	4 minutes ago	x86_64
v1	-	v1	6 minutes ago	x86_64

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

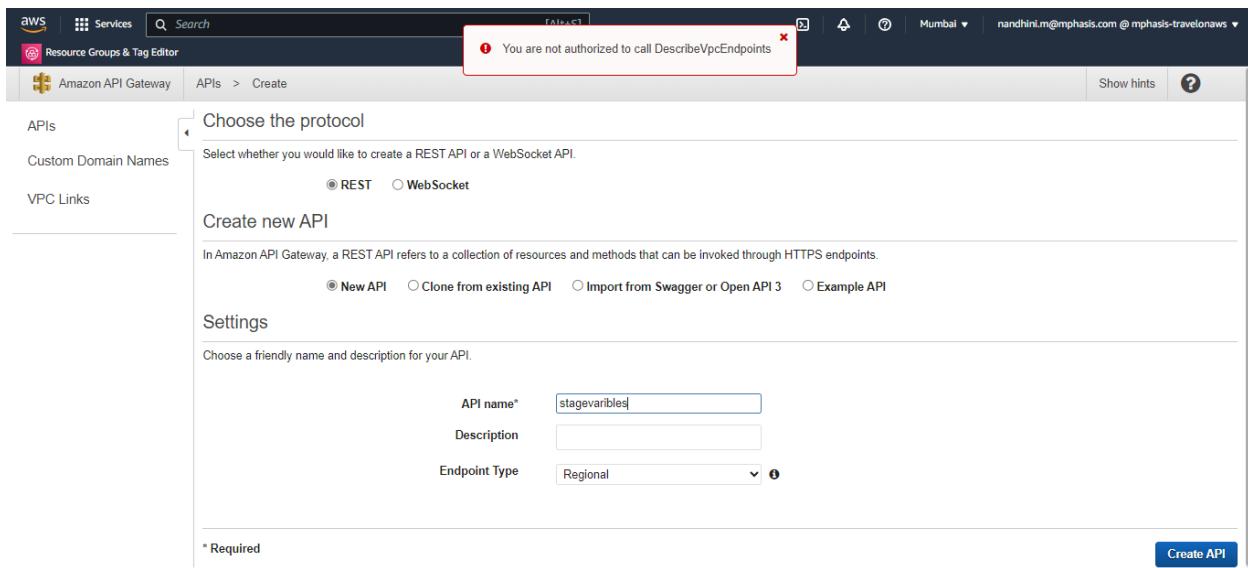


The screenshot shows the AWS Lambda console for the function 'stage_variable_lambda'. The 'Aliases' tab is selected, displaying three aliases:

Name	Versions	Description
dev	version: \$LATEST (weight=100%)	development lambda
prod	version: 1 (weight=100%)	producion lambda
test	version: 2 (weight=100%)	test lambda

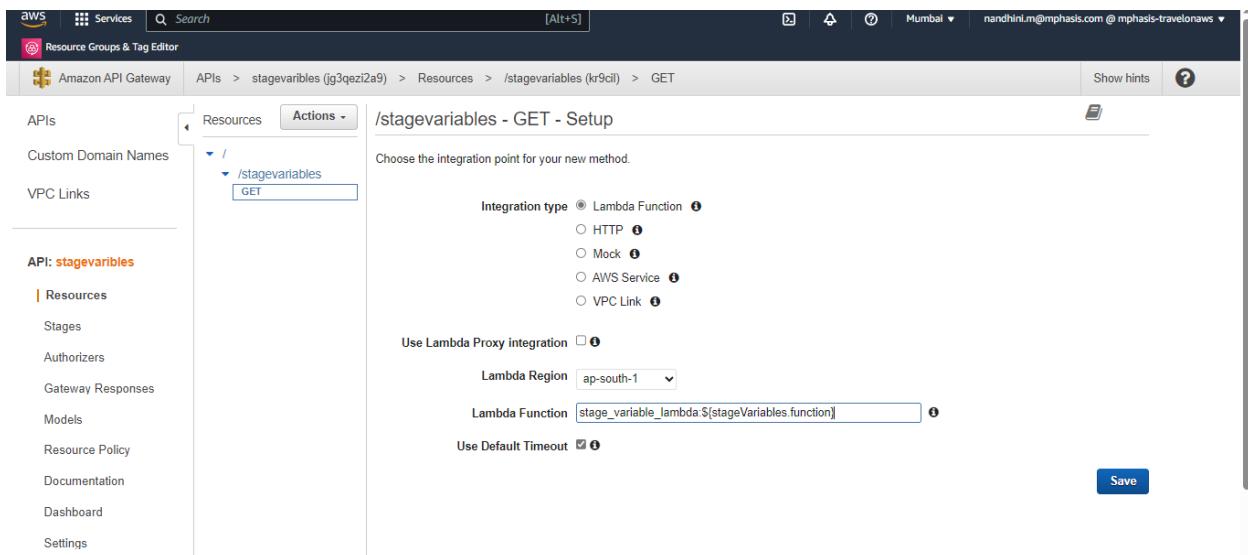
STEP 4: Set Up API Gateway with a Stage Variable.

- 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:jg3qeqz12a9/*:GET/stagevariables"
--principal apigateway.amazonaws.com
--statement-id c8db83b2-0756-41c8-8f8a-e4157c82fb41
--action lambda:InvokeFunction
```

Lambda Function: stage_variable_lambda \${stageVariables.function}

Use Default Timeout:

OK

Method Execution Flow

```

graph LR
    Client[Client] --> MethodRequest[Method Request]
    MethodRequest --> IntegrationRequest[Integration Request]
    IntegrationRequest --> MethodResponse[Method Response]
    MethodResponse --> IntegrationResponse[Integration Response]
    IntegrationResponse --> Lambda[Lambda stage_variable_lambda ${stageVariables.function}]
    
```

Method Request
Auth: NONE
ARN: arn:aws:execute-api:ap-south-1:195737202275:jg3qeqz12a9/*:GET/stagevariables

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

Logs

```

Execution log for request f0ee25e9-2701-41b5-8ed6-ec26480ad9af
Sun Dec 11 20:39:56 UTC 2022 : Starting execution for request: f0ee25e9-2701-41b5-8ed6-ec26480ad9af
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: {}
  
```

API: myapistages

Resources

- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Dashboard
- Settings

Query Strings
{stagevariables}
param1=value1¶m2=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

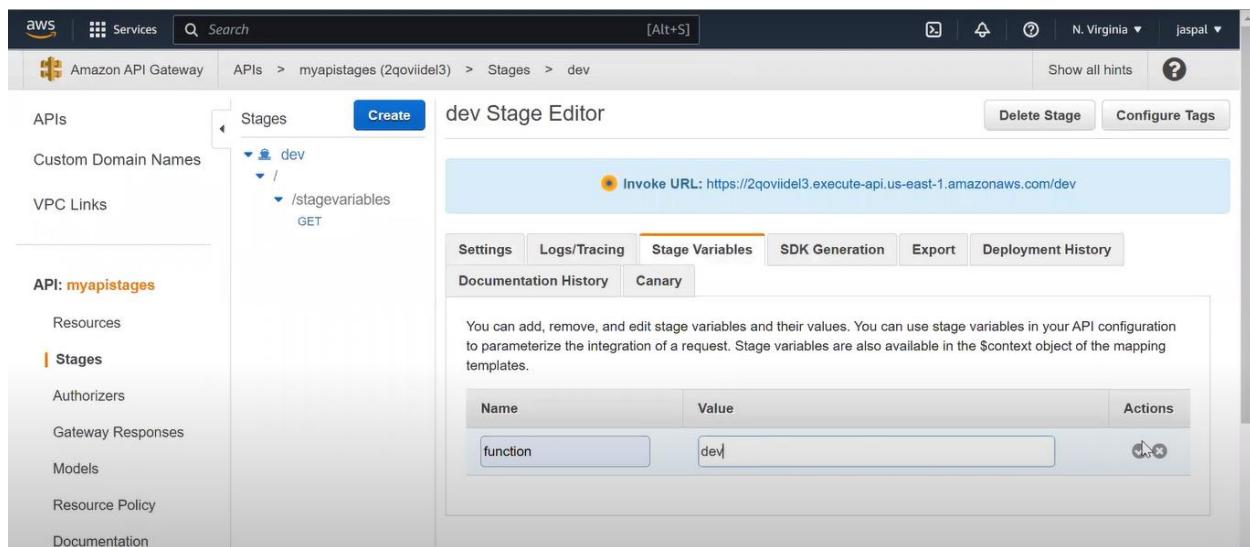
Response Headers
{ "X-Amzn-Trace-Id": ["Root=1-6396401c-d016b9f8008b8cdda2293e8b; Sampled=0"] }

Response Body
"Hello from Lambda! v2"

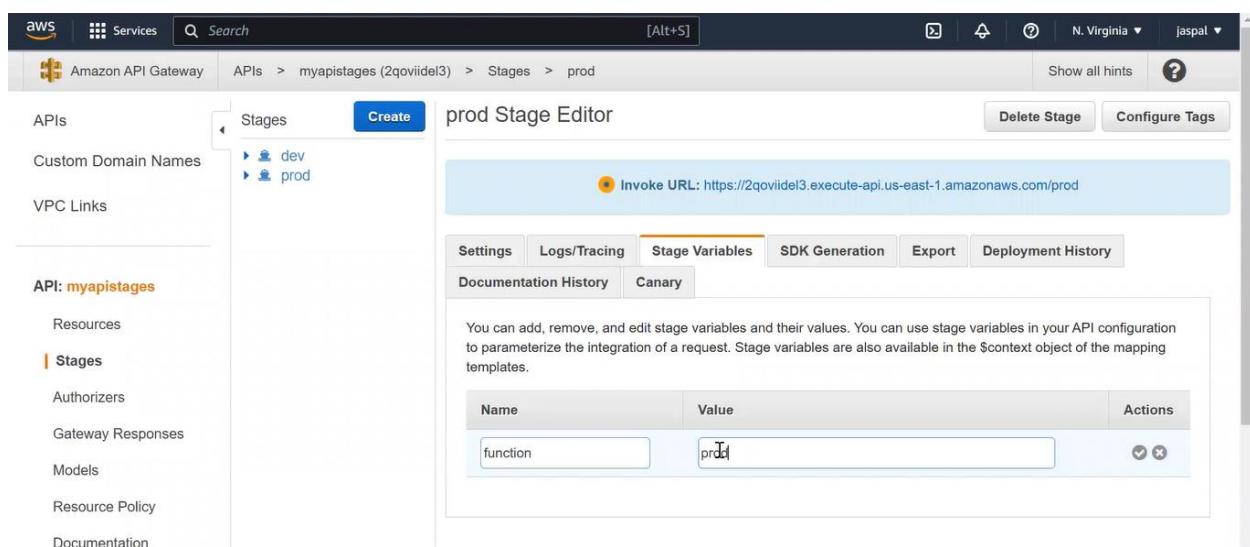
STEP 7: Deploy your API to a new stage (e.g., "prod", "dev", "test").

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."



The screenshot shows the AWS API Gateway interface. On the left, there's a sidebar with options like APIs, Custom Domain Names, VPC Links, and an API named 'myapistages'. Under 'myapistages', the 'Stages' option is selected. The main area shows a 'dev Stage Editor'. At the top, there are tabs for Settings, Logs/Tracing, Stage Variables, SDK Generation, Export, and Deployment History. The Stage Variables tab is active. Below it, there are tabs for Documentation History and Canary. A note says: 'You can add, remove, and edit stage variables and their values. You can use stage variables in your API configuration to parameterize the integration of a request. Stage variables are also available in the \$context object of the mapping templates.' A table lists a single stage variable: Name: function, Value: dev. There are 'Actions' buttons for each row.

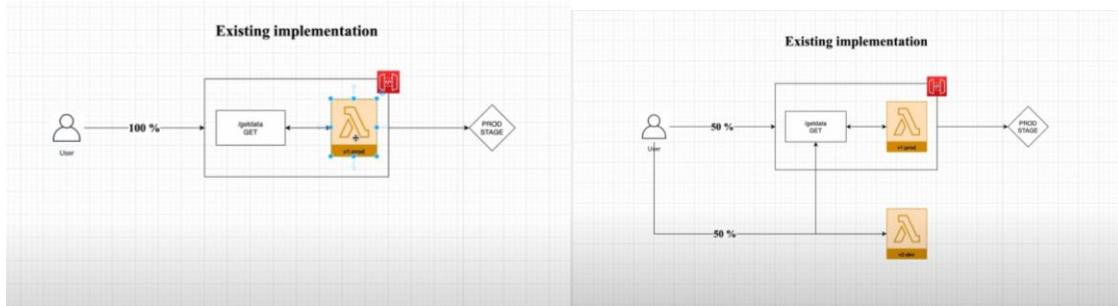


This screenshot shows the 'prod Stage Editor' within the same AWS API Gateway interface. The left sidebar and the main editor area are identical to the 'dev' stage editor, except the stage name 'prod' is highlighted in blue. The stage variables table shows a single entry: Name: function, Value: prod. The 'Actions' button next to the row has a checkmark icon.

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

The screenshot shows the 'prod Stage Editor' in the Amazon API Gateway console. Under the 'Canary' tab, it indicates that 50% of requests are directed to the 'Canary' and 50% to the 'prod' stage. In the 'Canary Stage Variables' section, there is a table:

Name	Stage Value	Canary Override Value
version	prod	dev

STEP 6: Test the invoke url

The screenshot shows a POSTMAN interface with an 'Untitled Request' for a GET request to `https://ff85qcyeg.execute-api.us-east-1.amazonaws.com/prod/getdata`. The response status is 200 OK, and the body contains the text "Hello from Dev!".

The screenshot shows a second POSTMAN interface with an 'Untitled Request' for a GET request to `https://ff85qcyeg.execute-api.us-east-1.amazonaws.com/prod/getdata`. The response status is 200 OK, and the body contains the text "Hello from Prod!".

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

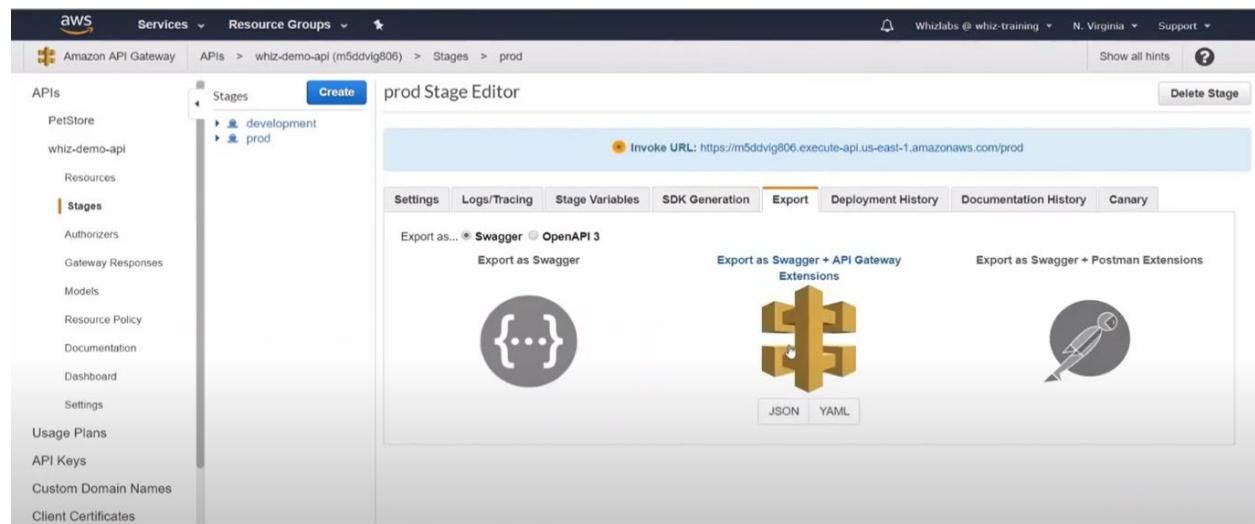
6.EXPORT:

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.

The screenshot shows the AWS API Gateway console. In the left sidebar, under 'APIs', 'whiz-demo-api' is selected. Under 'Stages', 'prod' is selected. The main area displays the 'Deployment History' tab. At the top, there's an 'Invoke URL' field: `https://m5ddvig806.execute-api.us-east-1.amazonaws.com/prod`. Below it, tabs include 'Settings', 'Logs/Tracing', 'Stage Variables', 'SDK Generation', 'Export', 'Deployment History' (which is highlighted in orange), 'Documentation History', and 'Canary'. A note says: 'Choose a deployment for the prod stage from the list below. For example, you may want to roll back to an earlier deployment.' A table lists six deployments with columns for 'Deployment date', 'Current stage', and 'Description'. The last two rows have descriptions: 'final release' and 's3 buckets api method'. A 'Change Deployment' button is at the bottom right.

Deployment date	Current stage	Description
Feb 9, 2019 10:54:31 PM		
Feb 9, 2019 10:53:01 PM		
Feb 9, 2019 10:48:36 PM		
Feb 9, 2019 10:42:01 PM		
Feb 9, 2019 10:35:37 PM		final release
Feb 9, 2019 10:15:34 PM		s3 buckets api method

8.DOCUMENTATION HISTORY

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

The screenshot shows the AWS API Gateway console. In the left sidebar, under 'APIs', 'whiz-demo-api' is selected. Under 'Stages', 'prod' is selected. The main area displays the 'Documentation History' tab. At the top, there's an 'Invoke URL' field: `https://m5ddvig806.execute-api.us-east-1.amazonaws.com/prod`. Below it, tabs include 'Settings', 'Logs/Tracing', 'Stage Variables', 'SDK Generation', 'Export', 'Deployment History' (which is highlighted in orange), 'Documentation History' (which is selected), and 'Canary'. A note says: 'Choose a documentation version for the stage from the list below. For example, you may want to roll back to an earlier version of the documentation.' A table lists documentation versions with columns for 'Version' and 'Deployment Date'. A message at the bottom says: 'No documentation versions found'. Navigation buttons include '<< < Page 1 > >>'.

Version	Deployment Date

The screenshot shows the AWS API Gateway interface. In the top navigation bar, the 'Documentation' section is selected. The main content area is titled 'Documentation' and contains a brief description of what documentation parts are for. Below this is a search/filter bar with dropdowns for 'Type' (set to 'All'), 'Path', 'Method' (set to 'All'), 'Name', and 'Status Code'. A message below the search bar says 'No documentation parts found. Click the **Create Documentation Part** button to add a new one.' On the left sidebar, under the 'APIs' section, the 'Documentation' option is highlighted.

CUSTOM DOMAIN NAME

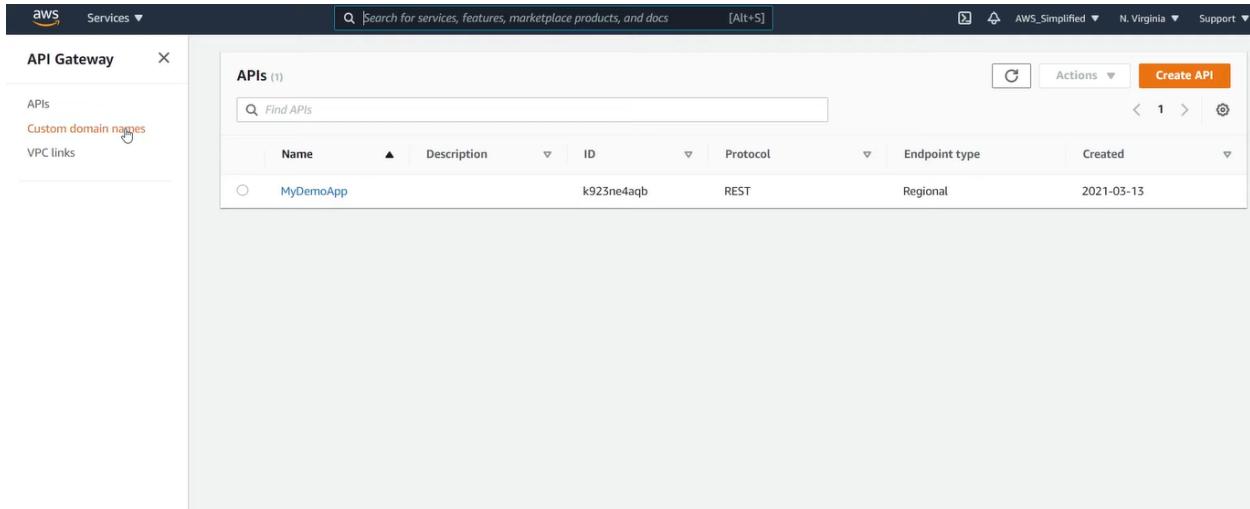
A custom domain name in API Gateway refers to a user-defined domain name that you can associate with your API endpoints instead of using the default domain name provided by API Gateway. This allows you to have a more branded and user-friendly URL for your API.

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

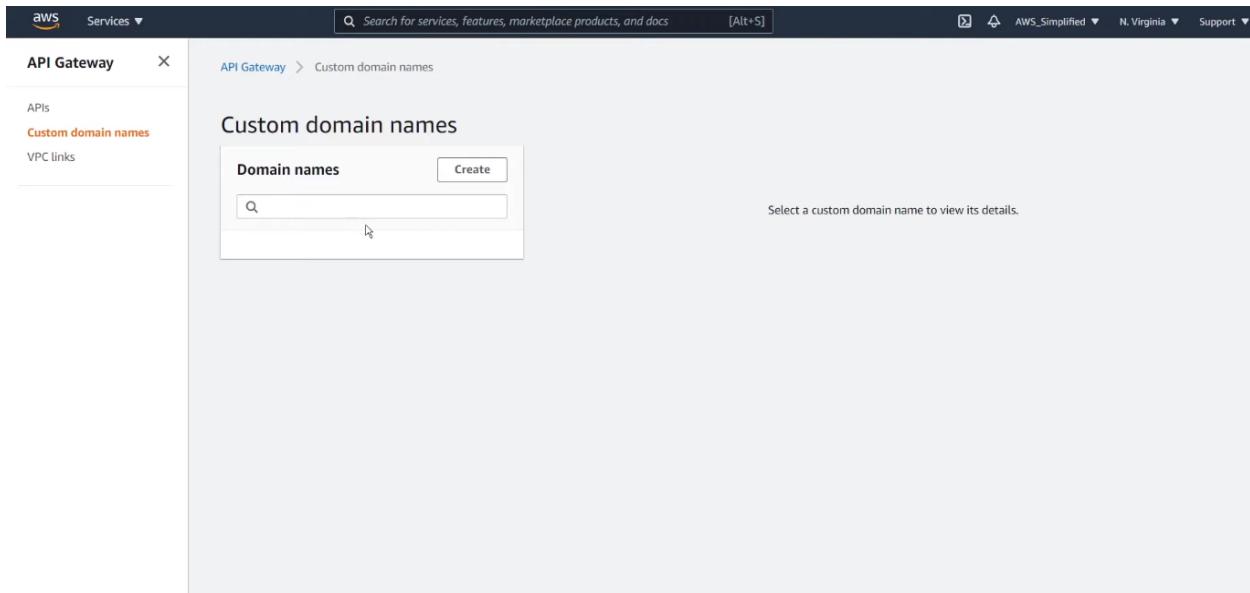
STEP 1: From the API Gateway dashboard, select the API for which you want to create a custom domain name.



The screenshot shows the AWS API Gateway dashboard. In the left navigation pane, under the 'APIs' section, 'Custom domain names' is highlighted. The main content area displays a table titled 'APIs (1)'. The table has columns: Name, Description, ID, Protocol, Endpoint type, and Created. There is one entry: 'MyDemoApp' with ID 'k923ne4aqb', Protocol 'REST', Endpoint type 'Regional', and Created '2021-03-13'. A 'Create API' button is located at the top right of the table.

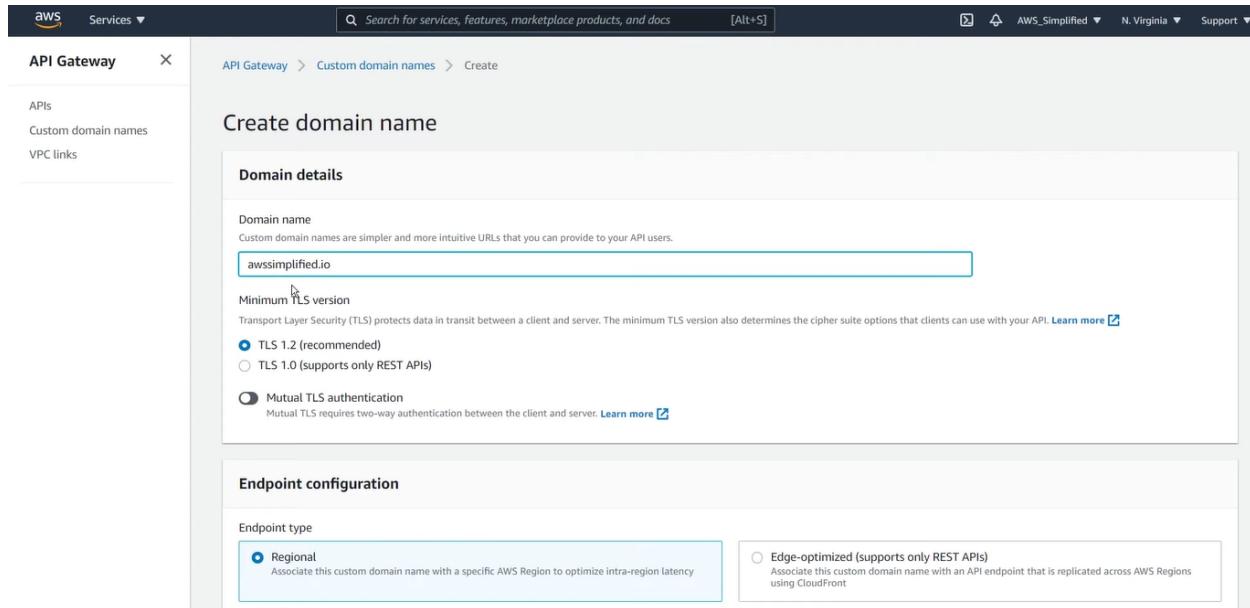
STEP 2: In the left navigation pane, under the selected API, click on "Custom domain names."

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



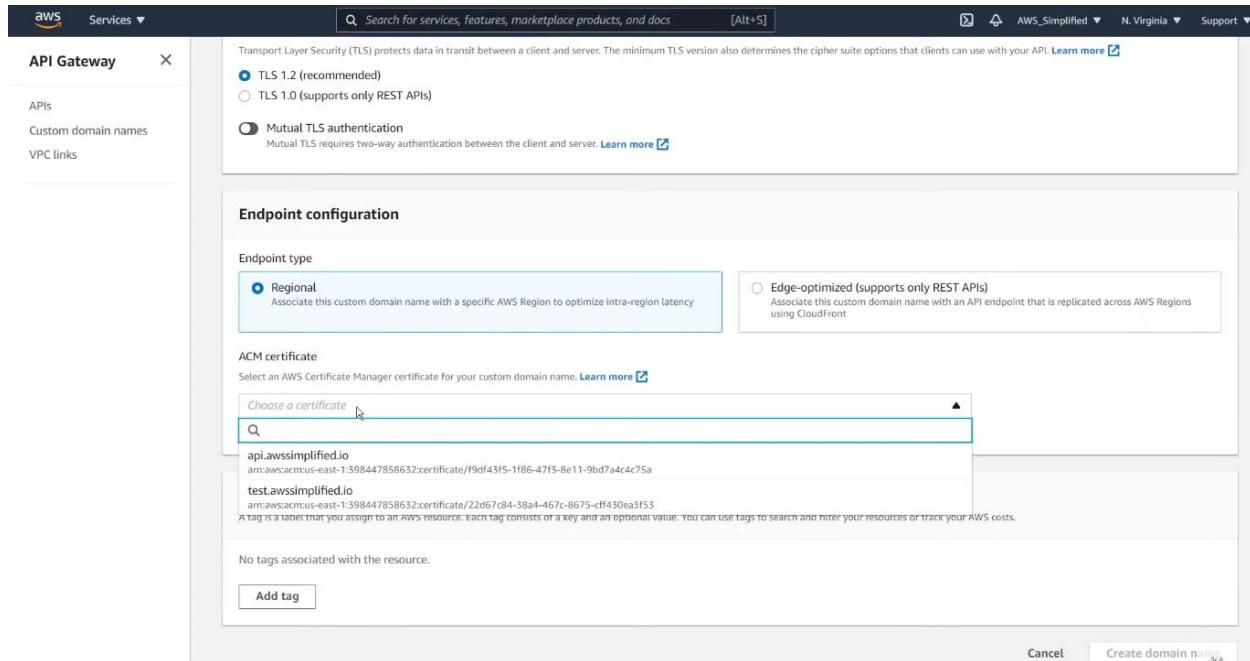
The screenshot shows the 'Custom domain names' creation page within the AWS API Gateway. The left navigation pane shows 'Custom domain names' is selected. The main content area has a heading 'Custom domain names' and a sub-section 'Domain names'. It features a 'Create' button and a search bar. Below the search bar, there is a message: 'Select a custom domain name to view its details.'.

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



The screenshot shows the 'Create domain name' page in the AWS API Gateway console. In the 'Domain details' section, the 'Domain name' field contains 'awssimplified.io'. Under 'Minimum TLS version', 'TLS 1.2 (recommended)' is selected. There are also options for 'TLS 1.0 (supports only REST APIs)' and 'Mutual TLS authentication'.

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



The screenshot shows the 'Create domain name' page in the AWS API Gateway console. In the 'Endpoint configuration' section, 'Regional' is selected as the endpoint type. Below it, there's a note about associating the domain with a specific AWS Region to optimize intra-region latency. The 'Edge-optimized (supports only REST APIs)' option is also present but not selected.

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.

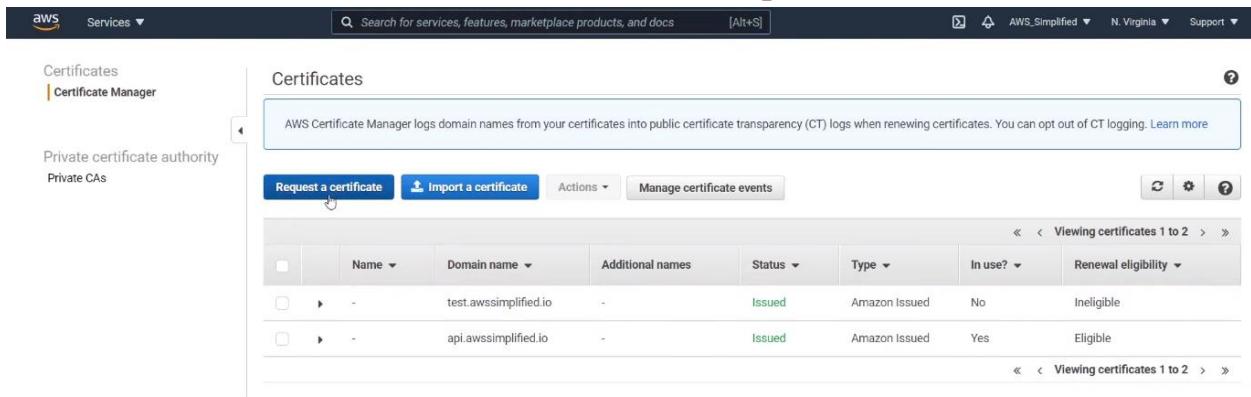
The screenshot shows the 'Endpoint configuration' section of the AWS API Gateway. Under 'TLS version', 'TLS 1.2 (recommended)' is selected. Under 'Endpoint type', 'Regional' is selected. In the 'ACM certificate' section, there is a dropdown menu labeled 'Choose a certificate' and a button labeled 'Create ACM certificate'. The 'Tags' section indicates 'No tags associated with the resource.' At the bottom right, there are 'Cancel' and 'Create domain name' buttons.

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."

The screenshot shows the 'Certificates' page in the AWS Certificate Manager. It displays two certificates: 'test.awssimplified.io' and 'api.awssimplified.io', both of which are 'Issued' by 'Amazon Issued' and marked as 'In use? No'. The page includes buttons for 'Request a certificate', 'Import a certificate', 'Actions', and 'Manage certificate events'.

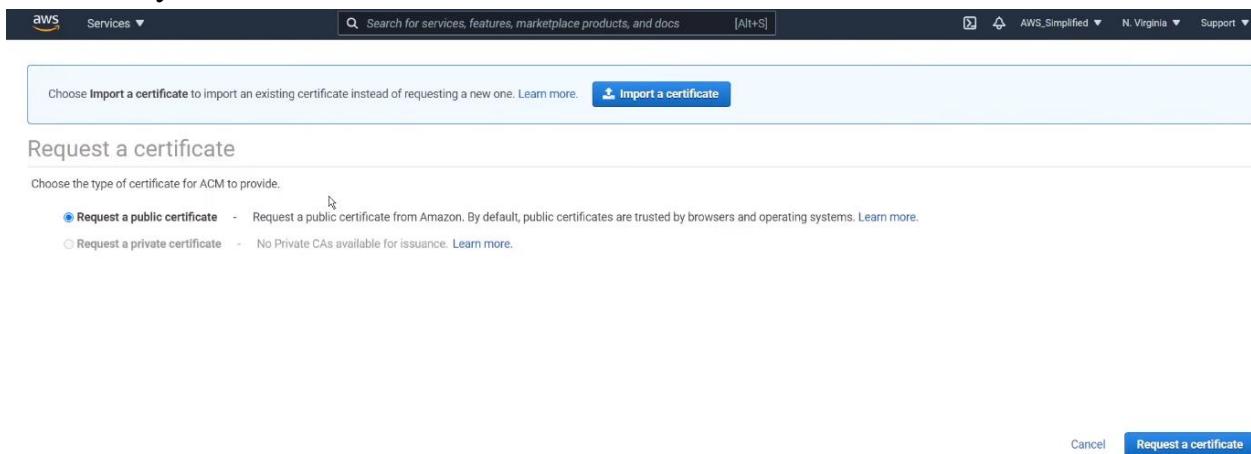
STEP 2: In the ACM dashboard, click the "Request a certificate" button.



The screenshot shows the AWS Certificate Manager (ACM) dashboard. On the left, there's a sidebar with 'Certificates' and 'Certificate Manager' under 'Services'. Below that, 'Private certificate authority' and 'Private CAs' are listed. The main area is titled 'Certificates' and contains a message about AWS Certificate Manager logs domain names from your certificates into public certificate transparency (CT) logs when renewing certificates. You can opt out of CT logging. Learn more. Below this is a table with the following data:

	Name	Domain name	Additional names	Status	Type	In use?	Renewal eligibility
<input type="checkbox"/>	-	test.awssimplified.io	-	Issued	Amazon Issued	No	Ineligible
<input type="checkbox"/>	-	api.awssimplified.io	-	Issued	Amazon Issued	Yes	Eligible

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



The screenshot shows the 'Request a certificate' wizard. At the top, it says 'Choose Import a certificate to import an existing certificate instead of requesting a new one. Learn more.' followed by a 'Import a certificate' button. Below this, the title 'Request a certificate' is displayed. A sub-instruction 'Choose the type of certificate for ACM to provide.' is present. There are two radio buttons: ' Request a public certificate' and ' Request a private certificate'. Each has a corresponding description: 'Request a public certificate from Amazon. By default, public certificates are trusted by browsers and operating systems. Learn more.' and 'No Private CAs available for issuance. Learn more.'. At the bottom right, there are 'Cancel' and 'Request a certificate' buttons.

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).

The screenshot shows the 'Request a certificate' wizard at Step 1: Add domain names. It includes a sidebar with navigation links: Step 1: Add domain names (highlighted), Step 2: Select validation method, Step 3: Add tags, Step 4: Review, and Step 5: Validation. The main content area has two sections: one about CT logging and another about using certificates with other AWS services. Below is a form to enter domain names, with 'myapis.aws.simplified.io' entered. A 'Add another name to this certificate' button is present, along with a note about adding additional names. At the bottom are 'Cancel' and 'Next' buttons.

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.

The screenshot shows the 'Select validation method' step. The sidebar shows Step 2: Select validation method is selected. The main content area explains the validation process and provides two options: 'DNS validation' (selected) and 'Email validation'. Both options have explanatory text and 'Learn more' links. At the bottom are 'Cancel', 'Previous', and 'Next' buttons.

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

Request a certificate

Review

Step 1: Add domain names
Step 2: Select validation method
Step 3: Add tags
Step 4: Review
Step 5: Validation

Domain name

The name you want to secure with an SSL/TLS certificate.

Domain name myapis.awssimplified.io

Validation method

The method AWS uses to validate your certificate request.

Validation method DNS

Cancel Previous **Confirm and request**

Request a certificate

Request in progress
A certificate request with a status of Pending validation has been created. Further action is needed to complete the validation and approval of the certificate.

Validation

Create a CNAME record in the DNS configuration for each of the domains listed below. You must complete this step before AWS Certificate Manager (ACM) can issue your certificate, but you can skip this step for now by clicking **Continue**. To return to this step later, open the certificate request in the ACM Console.

Domain	Validation status
myapis.awssimplified.io	Pending validation

Export DNS configuration to a file

Unable to export CNAME records

Continue

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

Request a certificate

Request in progress
A certificate request with a status of Pending validation has been created. Further action is needed to complete the validation and approval of the certificate.

Validation

Create a CNAME record in the DNS configuration for each of the domains listed below. You must complete this step before AWS Certificate Manager (ACM) can issue your certificate, but you can skip this step for now by clicking **Continue**. To return to this step later, open the certificate request in the ACM Console.

Domain	Validation status
myapis.awssimplified.io	Pending validation

Add the following CNAME record to the DNS configuration for your domain. The procedure for adding CNAME records depends on your DNS service Provider. Learn more.

Name	Type	Value
_570bfaa4215adcf8b41335fe29fd0028.myapis.awssimplified.io.	CNAME	_18771a40d9a3adbf0a550304182572f8.nfyddsqlcy.acm-validations.aws.

Note: Changing the DNS configuration allows ACM to issue certificates for this domain name for as long as the DNS record exists. You can revoke permission at any time by removing the record. Learn more.

Create record in Route 53 Amazon Route 53 DNS Customers ACM can update your DNS configuration for you. Learn more.

Export DNS configuration to a file You can export all of the CNAME records to a file

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

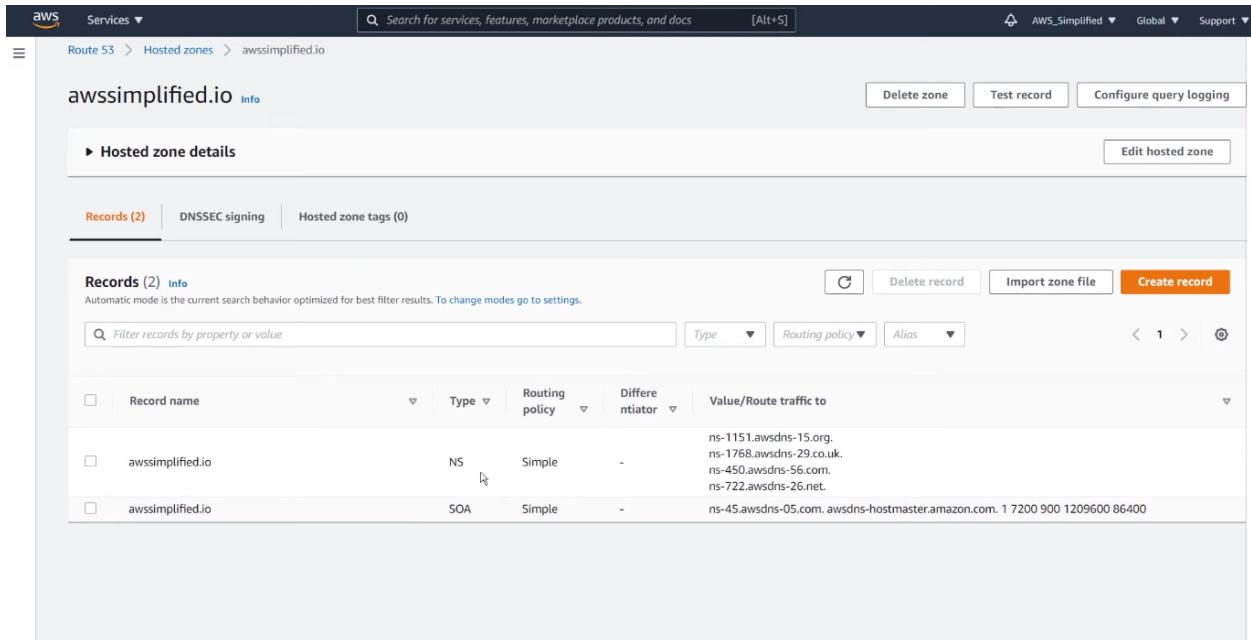
The screenshot shows the AWS Route 53 Dashboard. On the left, a sidebar menu includes options like 'Dashboard', 'Hosted zones', 'Health checks', 'Traffic flow', 'Domains', 'Resolver', and 'Switch to old console'. The main content area is titled 'Route 53 Dashboard' and contains four cards: 'DNS management' (1 Hosted zone), 'Traffic management' (A visual tool for creating policies), 'Availability monitoring' (Health checks monitor applications and web resources), and 'Domain registration' (1 Domain). Below these cards is a 'Register domain' section with a search bar and a 'Check' button. To the right, there's a 'More resources' sidebar with links to Documentation, API reference, FAQs, and forums. At the bottom, a 'Notifications' section shows no notifications.

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

The screenshot shows the 'Hosted zones' page in the AWS Route 53 console. The sidebar is identical to the previous dashboard screenshot. The main area is titled 'Hosted zones (1)' and shows a table with one row. The table columns are: Domain name, Type, Created by, Record count, Description, and Hosted zone ID. The single entry is 'awssimplified.io' (Type: Public, Created by: Route 53, Record count: 2, Hosted zone ID: Z104077536NV928GVT...). There are buttons for 'View details', 'Edit', 'Delete', and 'Create hosted zone'.

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

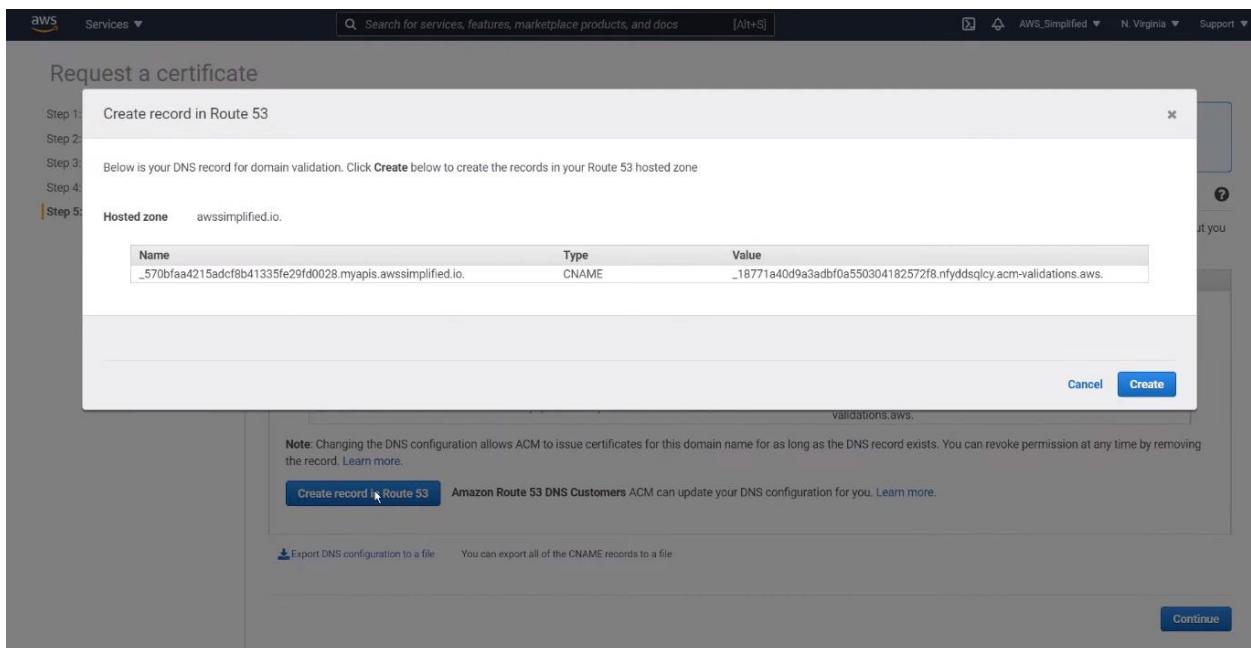
STEP 11: To create a new DNS record, click the "Create Record Set" button.



The screenshot shows the AWS Route 53 service interface. The URL in the address bar is `Route 53 > Hosted zones > awssimplified.io`. The main title is `awssimplified.io` with an `Info` link. Below it, there's a section titled `Hosted zone details` with an `Edit hosted zone` button. The `Records` tab is selected, showing a table with two entries:

Record name	Type	Routing policy	Differentiator	Value/Route traffic to
awssimplified.io	NS	Simple	-	ns-1151.awsdns-15.org. ns-1768.awsdns-29.co.uk. ns-450.awsdns-56.com. ns-722.awsdns-26.net.
awssimplified.io	SOA	Simple	-	ns-45.awsdns-05.com. awsdns-hostmaster.amazon.com. 1 7200 900 1209600 86400

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



The screenshot shows the 'Request a certificate' wizard, Step 5: Create record in Route 53. The main title is 'Request a certificate'. Step 5 is highlighted. The sub-step is 'Create record in Route 53'. The modal dialog contains the following information:

Name	Type	Value
_570bfaa4215adcf8b41335fe29fd0028.myapis.awssimplified.io.	CNAME	_18771a40d9a3adb0a550304182572f8.nfyddsqjcy.acm-validations.aws.

Below the table, there is a note: 'Note: Changing the DNS configuration allows ACM to issue certificates for this domain name for as long as the DNS record exists. You can revoke permission at any time by removing the record. Learn more.' There are 'Cancel' and 'Create' buttons at the bottom of the modal. At the bottom of the main page, there are buttons for 'Create record' (highlighted), 'Amazon Route 53 DNS Customers' (with a note about ACM updating DNS), 'Export DNS configuration to a file', and 'Continue'.

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

Request a certificate

Step 1: Add domain names
Step 2: Select validation method
Step 3: Add tags
Step 4: Review
Step 5: Validation

Request in progress
A certificate request with a status of Pending validation has been created. Further action is needed to complete the validation and approval of the certificate.

Validation

Create a CNAME record in the DNS configuration for each of the domains listed below. You must complete this step before AWS Certificate Manager (ACM) can issue your certificate, but you can skip this step for now by clicking **Continue**. To return to this step later, open the certificate request in the ACM Console.

Domain	Validation status
myapis.awssimplified.io	Pending validation

Add the following CNAME record to the DNS configuration for your domain. The procedure for adding CNAME records depends on your DNS service provider. Learn more.

Name	Type	Value
_570bfaa4215adcf8b41335fe29fd0028.myapis.awssimplified.io.	CNAME	_18771a40d9a3adbfoa550304182572f8.nfyddsqlcy.acm-validations.aws.

Note: Changing the DNS configuration allows ACM to issue certificates for this domain name for as long as the DNS record exists. You can revoke permission at any time by removing the record. Learn more.

Create record in Route 53 **Amazon Route 53 DNS Customers** ACM can update your DNS configuration for you. Learn more.

Success
The DNS record was written to your Route 53 hosted zone. It may take up to 30 minutes for the changes to propagate, and for AWS to validate the domain.

Export DNS configuration to a file You can export all of the CNAME records to a file

Now the record is added,

awssimplified.io [Info](#)

Hosted zone details

Records (3) [Info](#) Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.

Hosted zone tags (0)

Records (3) [Info](#)

Create record

Record name	Type	Routing policy	Differentiator	Value/Route traffic to
aws.simplified	NS	Simple	-	ns-1151.awsdns-15.org. ns-1768.awsdns-29.co.uk. ns-450.awsdns-56.com. ns-722.awsdns-26.net.
aws.simplified	SOA	Simple	-	ns-45.awsdns-05.com. awsdns-hostmaster.amazon.com. 1 7200 900 1209600 86400
_570bfaa4215adcf8b41335fe29fd0028.myapis.awssimplified.io	CNAME	Simple	-	_18771a40d9a3adbfoa550304182572f8.nfyddsqlcy.acm-validations.aws.

Certificates

Certificate Manager

Private certificate authority

Private CAs

Request a certificate Import a certificate Actions Manage certificate events

Viewing certificates 1 to 3

Name	Domain name	Additional names	Status	Type	In use?	Renewal eligibility
-	myapis.awssimplified.io	-	Issued	Amazon Issued	No	Ineligible

Status

Status: Issued
Detailed status: The certificate was issued at 2021-03-13T01:23:25UTC

Domain	Validation status
▶ myapis.awssimplified.io	Success

Export DNS configuration to a file You can export all of the CNAME records to a file

Details

Type	Amazon Issued	Requested at	2021-03-13T01:21:02UTC
In use?	No	Issued at	2021-03-13T01:23:25UTC
Domain name	myapis.awssimplified.io	Not before	2021-03-13T00:00:00UTC
Number of additional names	0	Not after	2022-04-11T23:59:59UTC
		Public key info	RSA 2048-bit

ADD ACM TO CUSTOM DOMAIN NAME

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

API Gateway

APIs

Custom domain names

VPC links

Custom domain names are simpler and more intuitive URLs that you can provide to your API users.

myapis.awssimplified.io

Minimum TLS version

Transport Layer Security (TLS) protects data in transit between a client and server. The minimum TLS version also determines the cipher suite options that clients can use with your API. Learn more

TLS 1.2 (recommended)

TLS 1.0 (supports only REST APIs)

Mutual TLS authentication

Mutual TLS requires two-way authentication between the client and server. Learn more

Endpoint configuration

Endpoint type

api.awssimplified.io
arn:aws:acm:us-east-1:398447858632:certificate/f9df43f5-1f86-47f3-8e11-9bd7a4c4c75a

testLawssimplified.io
arn:aws:acm:us-east-1:398447858632:certificate/22d67cb4-38a4-467c-8675-cff430ea3f53

myapis.awssimplified.io
arn:aws:acm:us-east-1:398447858632:certificate/e5c888a9-673e-45ee-b302-beff1fdch540a

Choose a certificate

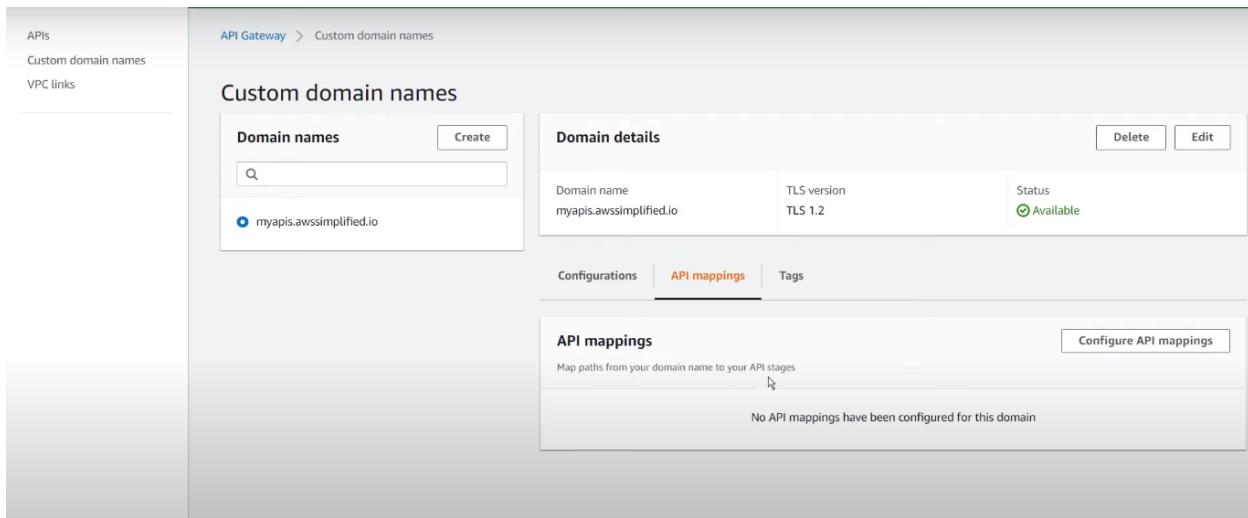
Create ACM certificate

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

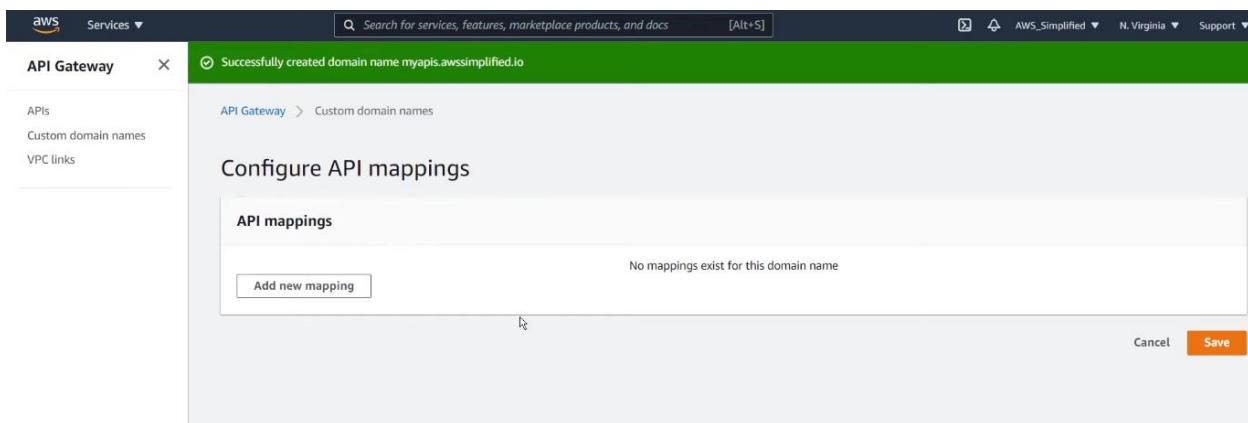
The screenshot shows the 'Endpoint configuration' section of the 'Create Custom Domain Name' wizard. Under 'Endpoint type', 'Regional' is selected. Under 'ACM certificate', 'myapis.awssimplified.io' is chosen from a dropdown menu. A 'Create ACM certificate' button is also present. The 'Tags' section indicates 'No tags associated with the resource.' A 'Create domain name' button is at the bottom right.

The screenshot shows the 'Custom domain names' list page. It displays a single domain entry: 'myapis.awssimplified.io'. The 'Domain details' table shows the domain name, TLS version (TLS 1.2), and status (Available). The 'Endpoint configuration' section shows the API Gateway domain name (d-rvs3mt6z28.execute-api.us-east-1.amazonaws.com), hosted zone ID (Z1UJRQUMOOOFQ8), endpoint type (Regional), and ACM certificate ARN (arn:aws:acm:us-east-1:398447858632:certificate/e3c888a9-673e-43ee-b302-befffdcb540a).

STEP 3: Click configure API Mapping.



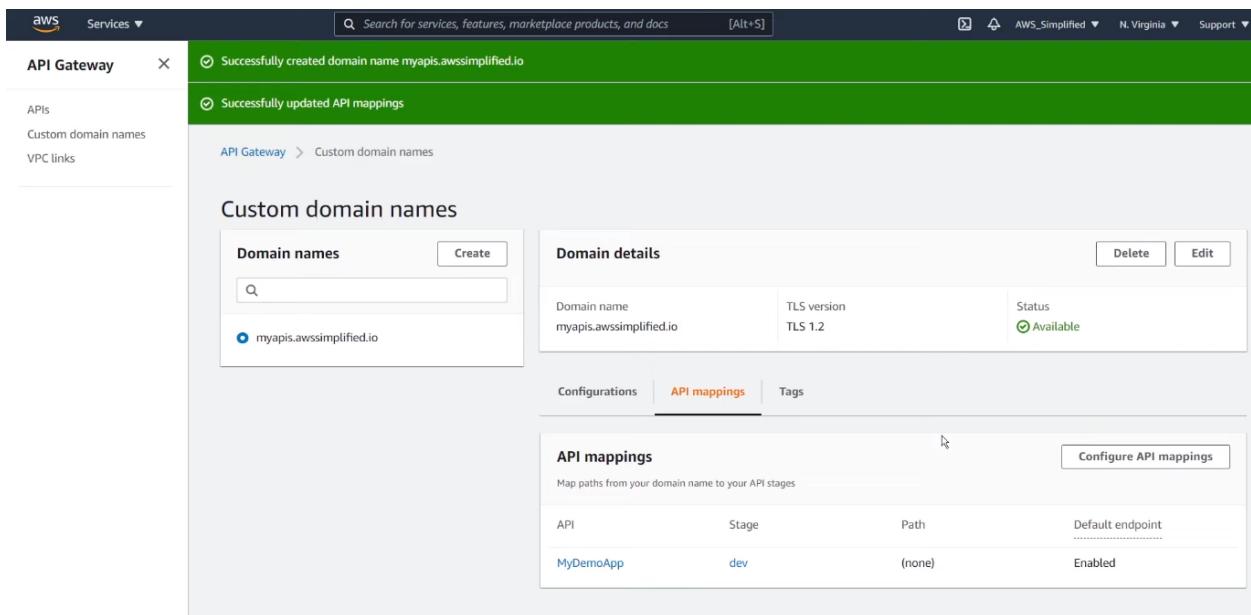
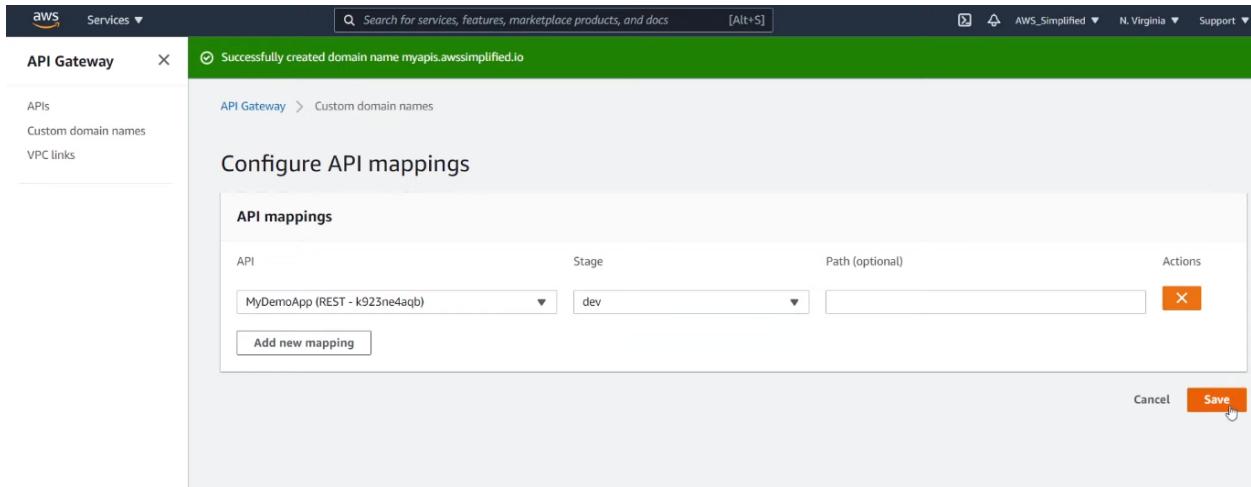
STEP 4: Click Add new mapping



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.

Screenshot of the AWS Route 53 Hosted Zone Details page for 'awssimplified.io'. The page shows three DNS records: an NS record for the root domain, an SOA record, and a CNAME record for a subdomain.

Record name	Type	Routing policy	Differentiator	Value/Route traffic to
awssimplified.io	NS	Simple	-	ns-1151.awsdns-15.org. ns-1768.awsdns-29.co.uk. ns-450.awsdns-56.com. ns-722.awsdns-26.net.
awssimplified.io	SOA	Simple	-	ns-45.awsdns-05.com. awsdns-hostmaster.amazon.com. 1 7200 900 1209600 86400
_570bfaa4215adcf8b41335fe29fd0028.myapis.awssimplified.io	CNAME	Simple	-	_18771a40d9a3adbf0a550304182572f8.nfyddsqlc.acm-validations.aws.

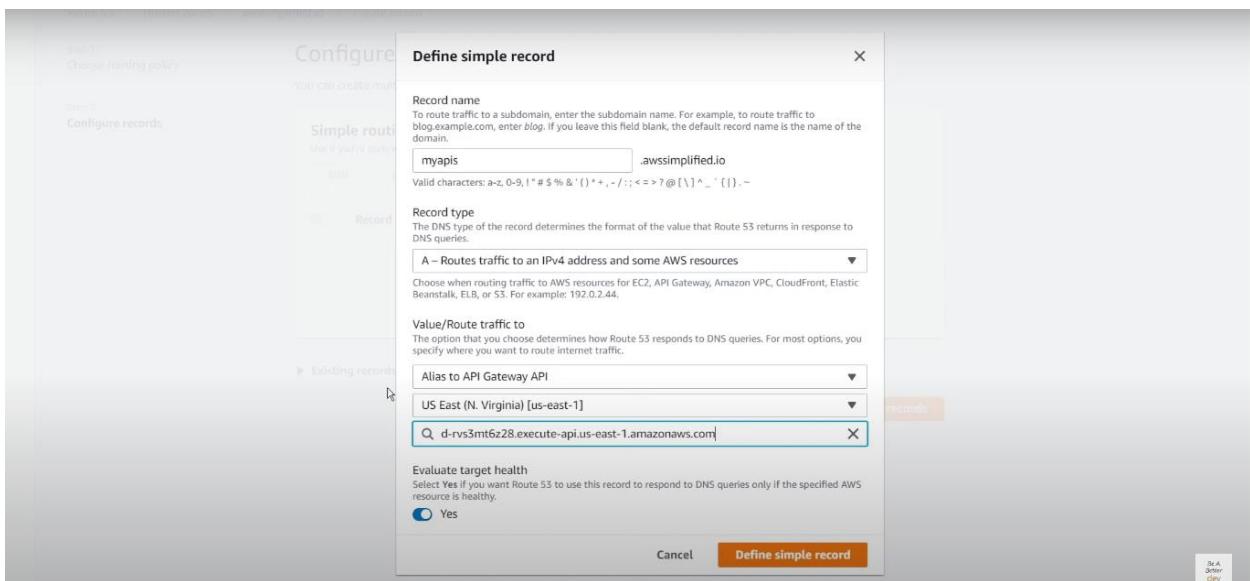
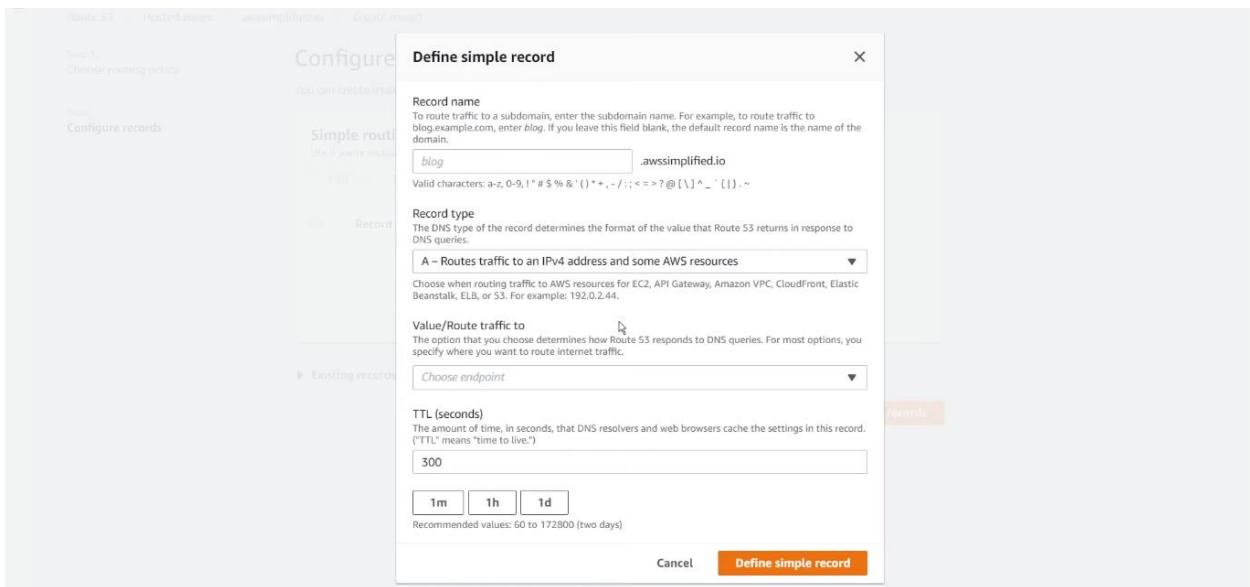
Screenshot of the 'Choose routing policy' step in the 'Create record' wizard. It lists six routing policies: Simple routing, Weighted, Geolocation, Latency, Failover, and Multivalue answer. The 'Simple routing' option is selected.

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.

The screenshot shows the AWS Route 53 'Create record' wizard. The current step is 'Step 2: Configure records'. The interface includes a breadcrumb navigation: Route 53 > Hosted zones > awssimplified.io > Create record. The main area is titled 'Configure records' with a 'Simple routing records to add to awssimplified.io' section. It explains that you can create multiple records at once. Below this is a table header with columns: Record name, Type, Value/Route traffic to, and TTL (seconds). There is a 'Define simple record' button. At the bottom of the table area, there is a note: 'Define simple records to this list, then choose Create records.' and another 'Define simple record' button. Navigation buttons at the bottom include 'Existing records', 'Cancel', 'Previous', and a prominent orange 'Create records' button.

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

The screenshot shows the AWS CloudFront console. At the top, there's a green banner indicating that a record for 'awssimplified.io' was successfully created. Below the banner, the navigation path is 'Route 53 > Hosted zones > awssimplified.io'. The main area displays the 'awssimplified.io' hosted zone details. A table lists four records:

Record name	Type	Routing policy	Differentiator	Value/Route traffic to
awssimplified.io	NS	Simple	-	ns-1151.awsdns-15.org. ns-1768.awsdns-29.co.uk. ns-450.awsdns-56.com. ns-722.awsdns-26.net.
awssimplified.io	SOA	Simple	-	ns-45.awsdns-05.com. awsdns-hostmaster.amazon.com. 1 7200 900 1209600 86400
myapis.awssimplified.io	A	Simple	-	d-rvs3mt6z28.execute-api.us-east-1.amazonaws.com.
_570bfaa4215adcf8b41335fe29fd0028.myapis.awssimplified.io	CNAME	Simple	-	_18771a40d9a3adbfoa550304182572f8.nfyddsqlc.yacm-validations.aws.

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

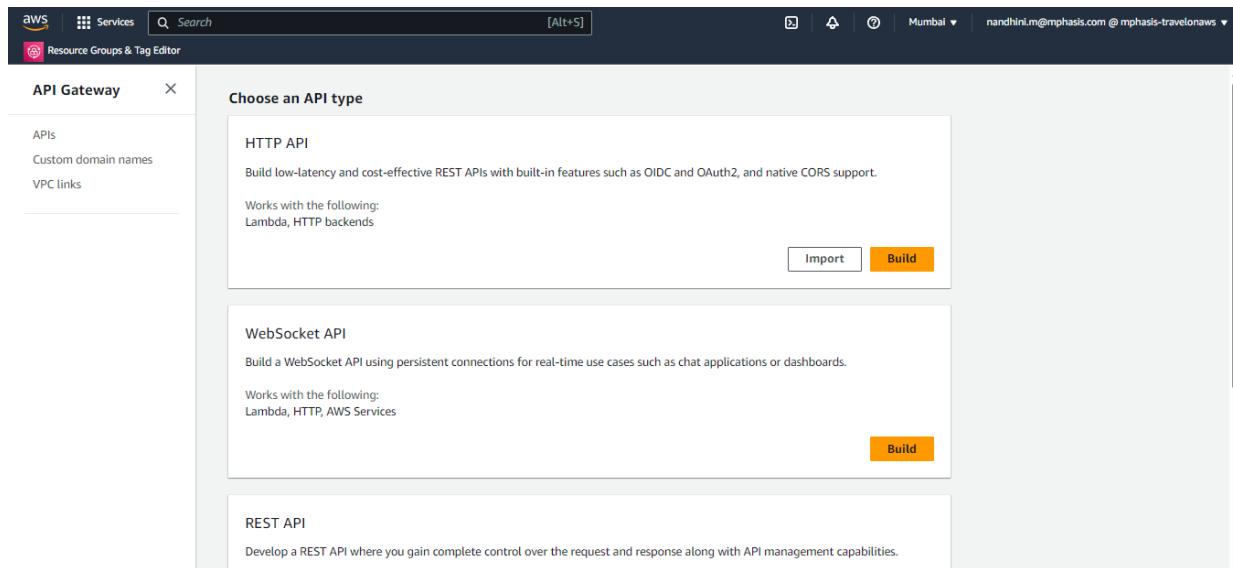
← → C myapis.awssimplified.io/customers/

{"statusCode":200,"body":"\"Hello from Lambda!\""}

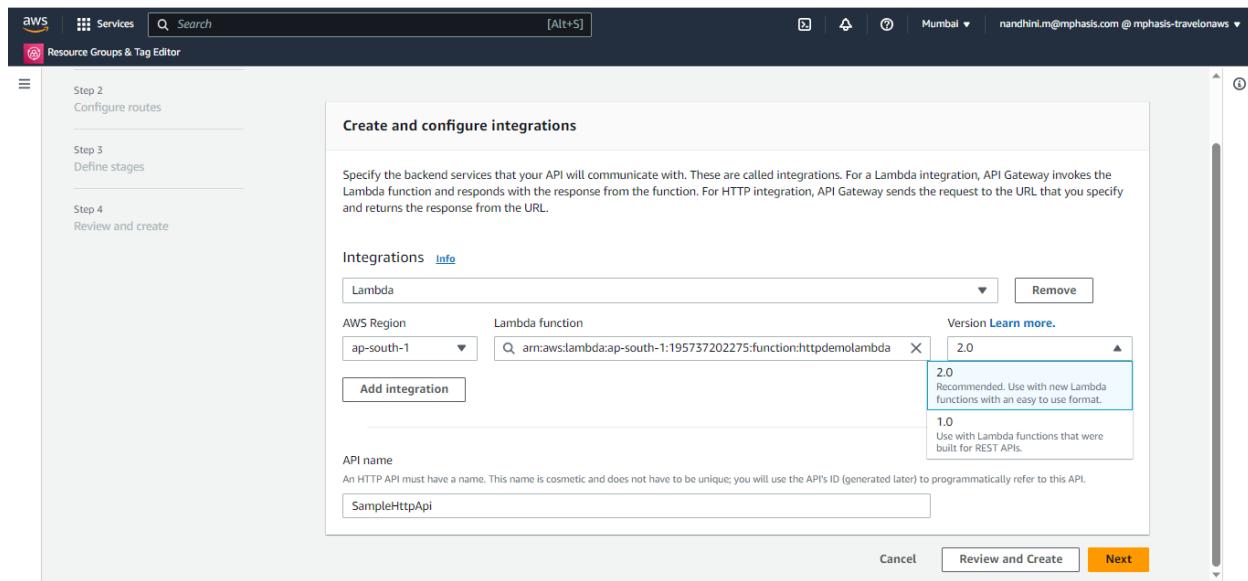

STEPS TO CREATE HTTP API

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.

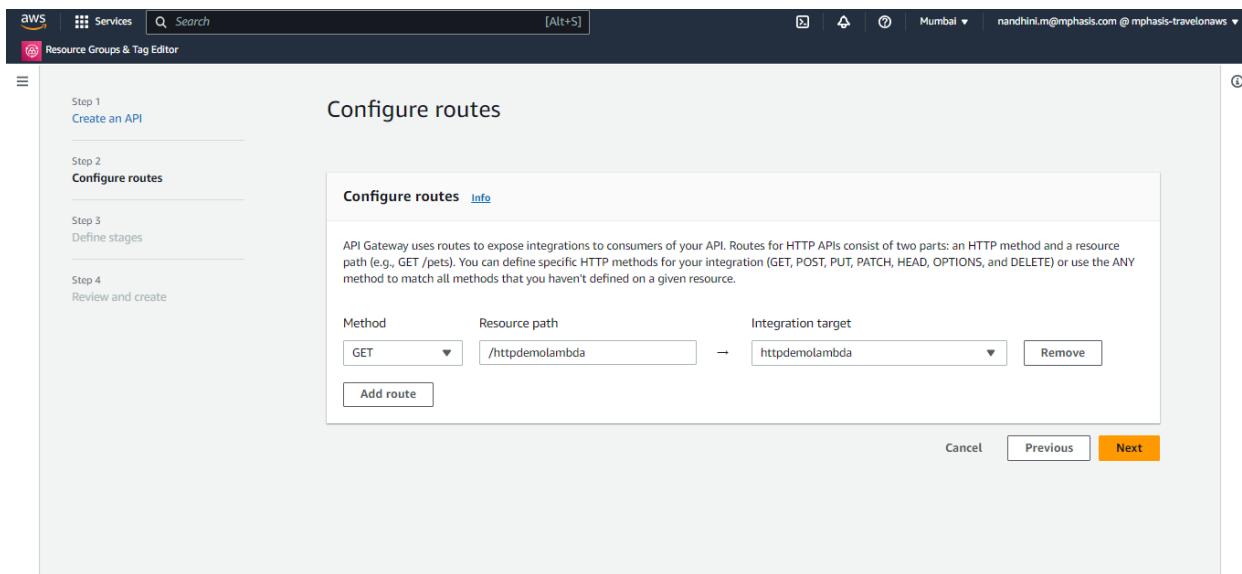


STEP 3: In your newly created API, navigate to the "Routes" section. Click the "Add Integration" button to create a new route.

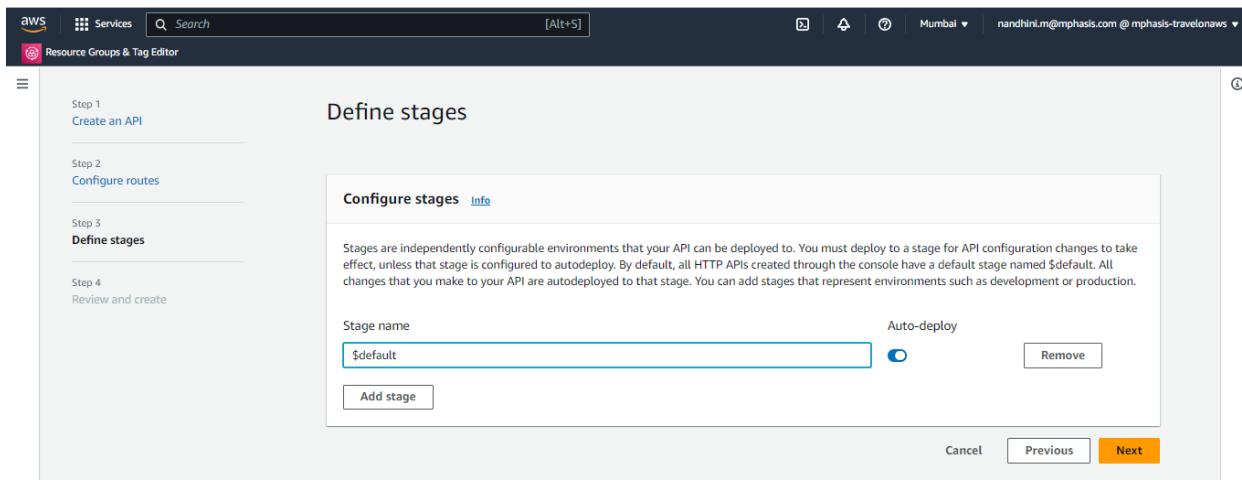
STEP 4: Choose "Lambda Function" as the integration type.

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

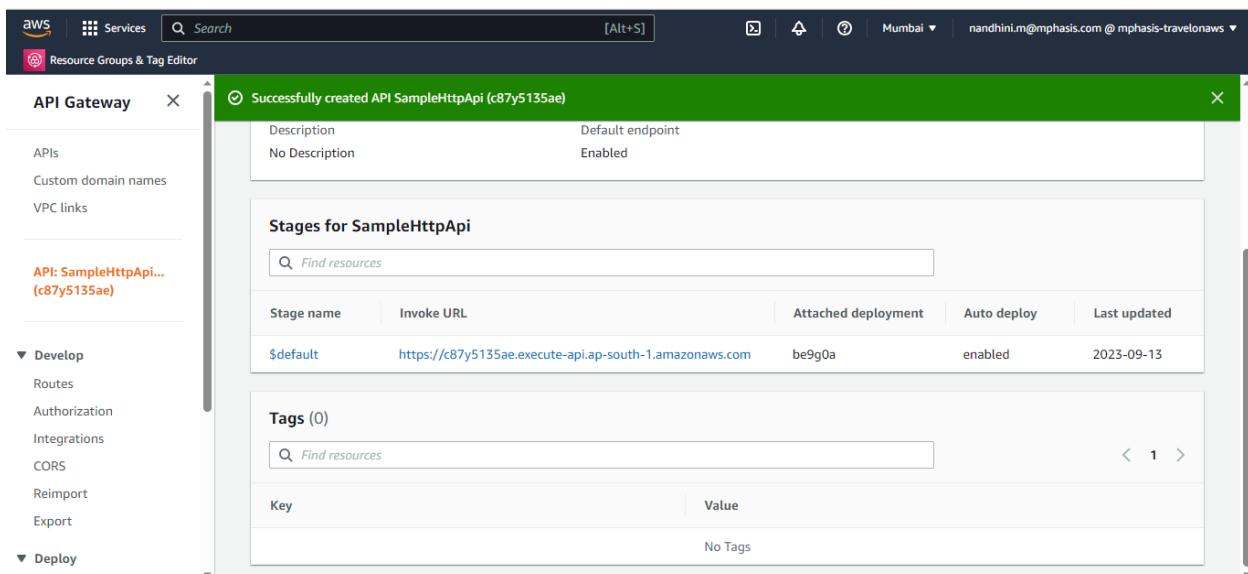
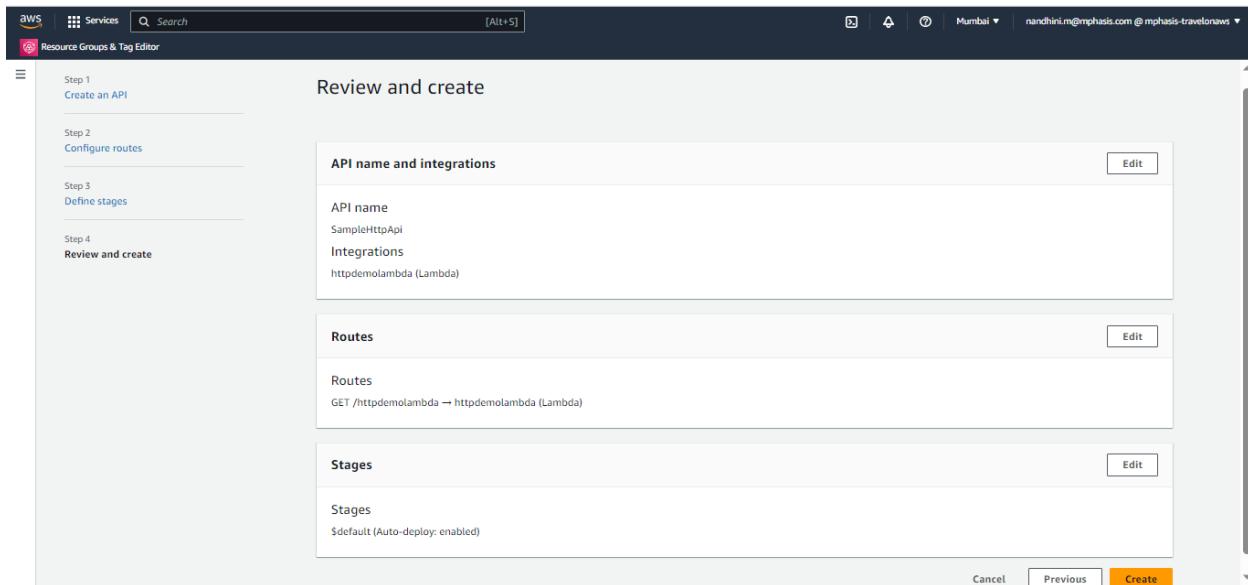
STEP 6: Configure the HTTP method (e.g., GET, POST) and resource path for this route



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.



The screenshot shows the AWS Lambda Function Overview page for a function named 'httpdemolambda'. The top navigation bar includes the AWS logo, Services, a search bar, and account information for Mumbai and nandhini.m@mphasis.com @ mphasis-travelonaws. The main content area displays the function's name, a thumbnail icon, and a 'Layers' section showing '(0)'. Below this is an 'API Gateway' section with a thumbnail icon, a '+ Add destination' button, and a '+ Add trigger' button. To the right, there are sections for 'Description' (empty), 'Last modified' (5 minutes ago), 'Function ARN' (arn:aws:lambda:ap-south-1:195737202275:function:httpdemolambda), and 'Function URL' (Info). Buttons for 'Throttle', 'Copy ARN', and 'Actions' are located at the top right.

The screenshot shows the AWS Lambda Configuration page for the 'httpdemolambda' function. The top navigation bar includes the AWS logo, Services, a search bar, and account information for Mumbai and nandhini.m@mphasis.com @ mphasis-travelonaws. The left sidebar has tabs for General configuration, Triggers (which is selected), Permissions, Destinations, Function URL, Environment variables, Tags, VPC, Monitoring and operations tools, Concurrency, and Asynchronous invocation. The main content area shows a 'Triggers (1) Info' section with a 'Find triggers' search bar and buttons for 'C' (Create), 'Fix errors', 'Edit', 'Delete', and 'Add trigger'. A single trigger is listed: 'Trigger' (API Gateway: SampleHttpApi). The details for this trigger include: API type: HTTP, Authorization: NONE, CORS: No, Detailed metrics enabled: No, Method: ANY, Resource path: /httpdemolambda, Service principal: apigateway.amazonaws.com, Stage: \$default, and Statement ID: a08b884f-b4c0-5bbe-bdde-e4020e5c9d9d. Navigation arrows for the trigger list are shown at the bottom right.

GATEWAY RESPONSES

These are the gateway responses available in the API gateway.

Gateway Responses

- Access Denied
- API Configuration Error
- Authorizer Configuration Error
- Authorizer Failure
- Bad Request Body
- Bad Request Parameters
- Default 4XX
- Default 5XX
- Expired Token
- Integration Failure
- Integration Timeout
- Invalid API Key
- Invalid Signature
- Missing Authentication Token
- Quota Exceeded
- Request Too Large
- Resource Not Found
- Throttled
- Unauthorized
- Unsupported Media Type
- WAF Filtered

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