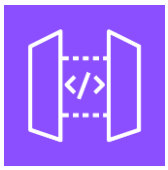


AWS API Gateway

Before we jump into the AWS API Gateway, we need to refresh our memory about two terms Gateway and API – what are those meant for us.

API (Application Programming Interface)

An API is a set of rules and protocols that allows different software applications to communicate with each other. It defines the methods and data formats that applications can use to request and exchange information.

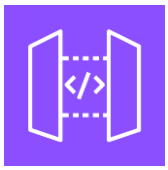
Gateway

A gateway is a server that acts as an intermediary, managing traffic between different networks or systems. In the context of an API Gateway, it handles incoming API requests, directing them to the appropriate backend services and returning the responses to the client.

API Gateway is needed to manage, secure, and optimize API requests and responses, providing a single-entry point to multiple backend services. It enhances security, scalability, and efficiency in application communication.

Now we know the API Gateway meaning, we will look into – How AWS API Gateway Operates and helps us in building our applications.

Now we know the API, what it is – we need to know who decides the structure of the API or Building Blocks of the API, those are the Resource and Method in the AWS API Gateway



AWS API Gateway

Resource

A resource represents an endpoint in your API. It is a URL path that corresponds to an object or a collection of objects.

Example: /users, /orders/{orderId}.

Resource Type	Purpose	Example
Path Resource	Main building blocks for API endpoints	`` `/users` ``, `` `/orders/{orderId}` ``
Proxy Resource	Captures all requests to a particular path and subpaths	`` `/proxy/{proxy+}` ``
Custom Domain Names	User-friendly URLs for API endpoints	`` `api.example.com` ``
Stage Variables	Different behaviors for different stages of API	Development vs. Production configurations
Lambda Authorizers	Custom authorization logic	Validate tokens, check user roles
Usage Plans and API Keys	Control and manage API usage	Limit requests per user per day

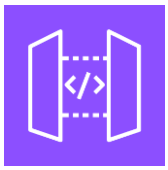
Method

A method represents an action that can be performed on a resource. It corresponds to an HTTP request method (e.g., GET, POST, PUT, DELETE).

Example:

- GET /users to retrieve a list of users.
- POST /orders to create a new order.

Method	Purpose	Use Case
GET	Retrieve information	Fetch data or resources
POST	Submit data	Create or submit data
PUT	Update or replace a resource	Modify or replace existing resources
DELETE	Remove a resource	Delete a specific resource
PATCH	Apply partial updates	Update specific fields of a resource
OPTIONS	Describe communication options for a resource	Find out allowed methods or other options
HEAD	Retrieve metadata about a resource	Check existence or metadata without data



Together, resources and methods define the structure and behaviour of your API endpoints.

One more fundamental block, what are REST & HTTP APIs

HTTP (Hypertext Transfer Protocol) is the underlying protocol used for transferring data over the web, facilitating communication between clients and servers through methods like GET, POST, PUT, and DELETE.

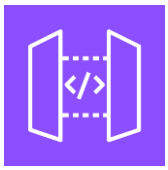
REST (Representational State Transfer) is an architectural style that leverages HTTP for creating scalable and stateless web services by organizing interactions around resources identified by URLs and using standard HTTP methods for operations.

Use HTTP:

- For general data transfer needs, such as serving web pages, images, and files.
- When implementing low-level communication between clients and servers without the constraints of a specific architectural style.

Use REST:

- When designing web services that require a clear, resource-oriented architecture.
- When you need stateless interactions, scalability, and the ability to leverage standard HTTP methods to perform CRUD operations on resources.



AWS API Gateway

How Many Types of APIs that AWS API Gateway supports

4 Types:

- HTTP API

HTTP APIs are designed for building APIs that are simpler and faster to deploy compared to REST APIs, with a focus on low latency and cost efficiency.

- WebSocket API

WebSocket APIs provide full-duplex communication channels over a single TCP connection.

- REST API

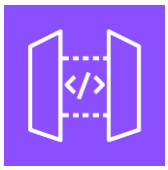
Representational State Transfer (REST) APIs are built around resources and HTTP methods.

- REST API Private

At a glance

API Type	Use Cases	Features
REST API	Web/mobile applications, microservices, third-party integrations	Caching, throttling, usage plans, AWS integrations
HTTP API	Lightweight microservices, event-driven architectures, webhooks	Lower latency, cost-efficient, simplified setup
WebSocket API	Real-time applications, streaming data, bidirectional communication	Persistent connections, real-time data transfer

PS: In REST API – AWS API Gateway supports Public and Private as well…!



AWS API Gateway

How Many Types of API End points that AWS API Gateway supports

Regional Endpoints

Regional endpoints are deployed in a specific AWS region, making them accessible within that region.

- For clients within the same region
- Could manually combine with your own CloudFront distribution for global deployment (this way you will have more control over the caching strategies and the distribution)

Edge-Optimized Endpoints

Edge-optimized endpoints use the Amazon CloudFront content delivery network (CDN) to cache API requests and responses closer to end users globally.

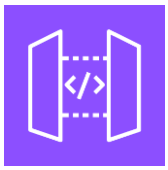
- For global clients
- Requests are routed through the CloudFront edge locations (improves latency)
- The API Gateway lives in only one region but it is accessible efficiently through edge locations

Private Endpoints

Private endpoints are accessible only within your Amazon Virtual Private Cloud (VPC) through a VPC endpoint, ensuring that the API is not exposed to the public internet.

- Can only be accessed within your VPC using an **Interface VPC endpoint** (ENI)
- Use resource policy to define access

Endpoint Type	Use Cases	Features
Regional Endpoint	Low latency for regional traffic, compliance, internal applications	Deployed in a specific region, low latency for regional users
Edge-Optimized Endpoint	Global applications, content delivery, e-commerce, media	Uses CloudFront CDN, cached responses at edge locations
Private Endpoint	Internal microservices, enterprise applications, data protection	Accessible only within VPC, ensures private network traffic



How Many Types of API Integrations *that AWS API Gateway supports*

Lambda Proxy Integration

Directly integrates API Gateway with AWS Lambda functions. The request data is passed as-is to the Lambda function and the response is returned directly from the Lambda function.

Use Cases:

- When you need serverless backends.
- Handling business logic without managing servers.
- Ideal for microservices architectures.

Lambda Non-Proxy Integration

Allows more control over the request and response format. API Gateway can transform the incoming request data before passing it to the Lambda function and can transform the Lambda function's response before sending it back to the client.

Use Cases:

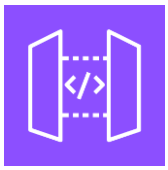
- When you need to map specific parts of the request or response to different formats.
- Useful when the input data needs significant preprocessing before being passed to Lambda.

HTTP Proxy Integration

Description: API Gateway forwards the entire request to a backend HTTP endpoint. The response from the backend is returned directly to the client.

Use Cases:

- When you have an existing HTTP backend service that you want to expose through an API.
- Quick and easy integration for existing applications.



HTTP Custom Integration

More customizable version of HTTP proxy integration. You can map the incoming request and outgoing response to different formats.

Use Cases:

- When you need to transform the request before sending it to the backend HTTP endpoint.
- When you need to transform the response from the backend before sending it to the client.

AWS Service Integration

API Gateway can directly interact with other AWS services such as S3, DynamoDB, SNS, SQS, etc. without needing a Lambda function as an intermediary.

Use Cases:

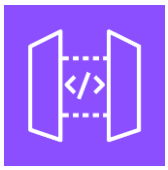
- Directly reading from or writing to DynamoDB.
- Uploading files to S3.
- Sending messages to SQS or SNS.
- Reducing the need for additional Lambda functions when direct service interactions suffice.

Mock Integration

API Gateway returns a specified response without sending the request to a backend service.

Use Cases:

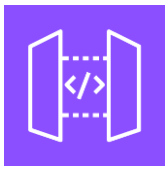
- Testing and prototyping APIs without a backend.
- Providing placeholder responses for not-yet-implemented services.



Enough Enough with theory …! 😉

Let's Create our First API Gateway program using the Lambda Integration to print Hello World …!, we will go to GitHub for that ..!

We will look at the URL Formats



REST API URL format

```
https://{api-id}.execute-api.{region}.amazonaws.com/{stage-name}/{resource-name}
```

- {api-id}: This is the unique identifier for your API, which is generated by API Gateway.
- {region}: The AWS region where your API Gateway is deployed.
- {stage-name}: The stage you deployed your API to (e.g., prod).
- {resource-name}: The resource path you defined in API Gateway (e.g., hello).

HTTP API URL format

```
https://{api-id}.execute-api.{region}.amazonaws.com/{stage}/{resource-path}
```

- {api-id}: The unique identifier for your API.
- {region}: The AWS region where your API is deployed (e.g., us-east-1).
- {stage}: The stage of your API (e.g., dev, test, prod).
- {resource-path}: The specific resource path defined in your API.

WebSocket API URL Format

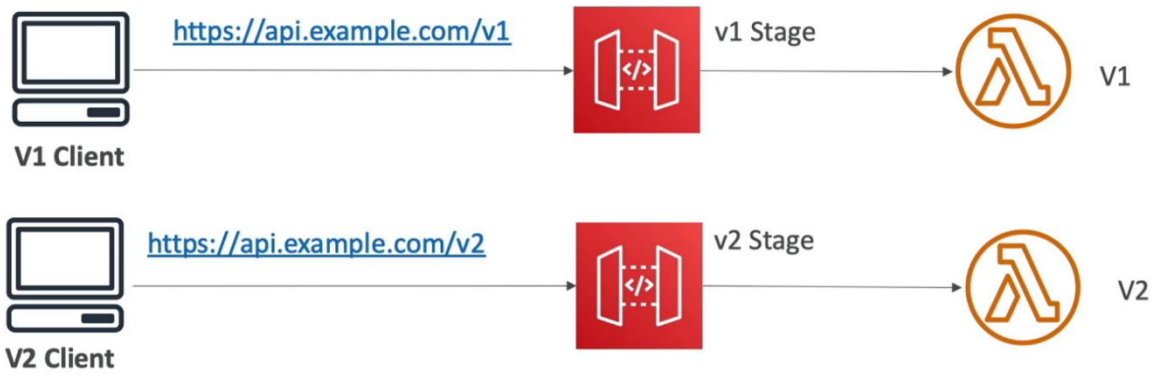
```
wss://{api-id}.execute-api.{region}.amazonaws.com/{stage}
```

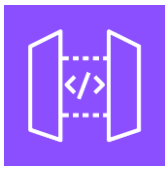
- {api-id}: The unique identifier for your WebSocket API.
- {region}: The AWS region where your WebSocket API is deployed (e.g., us-east-1).
- {stage}: The stage of your WebSocket API (e.g., dev, test, prod).

Deployments

- Deploy the API to get the API Gateway endpoint
- If you make changes to the API, a new version is internally created. You need to deploy the API for the changes to take effect.
- Versions (changes) are deployed to stages (no limit on the number of stages) Metrics and logs are separate for each stage
- Each stage has independent configuration and can be rolled back to any version (the whole history of deployments to a stage is kept)
- Handling breaking changes using multiple stages

We want to create a new version of the application which involves breaking changes at the API level. In this case, we can deploy a new stage (v2) and build our new application there. Since multiple stages can co-exist, we can have both the versions working and once all the users have migrated to v2, we can bring down v1.





Mapping Templates

Mapping templates in AWS API Gateway are used to transform request and response payloads between the client and backend systems. They allow you to manipulate the data format and structure to match the needs of your API and backend services. Mapping templates are written in Velocity Template Language (VTL).

In AWS API Gateway, there are primarily two types of mapping templates used for request and response transformations:

Request Mapping Templates:

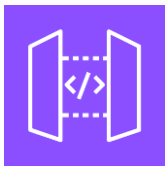
- Request mapping templates are used to transform the incoming request from the client before it reaches the backend integration.
- They allow you to modify headers, query parameters, and request body content.
- You can change the data format (e.g., JSON to XML) or structure to match the expected format of your backend service.
- Request mapping templates are applied in the "Integration Request" section of your API Gateway method configuration.

```
{
    "id": "$input.json('$.userId')",
    "name": "$input.json('$.userName')"
}
```

Response Mapping Templates:

- Response mapping templates are used to transform the response from the backend integration before it is sent back to the client.
- They allow you to modify headers, response body content, and status codes.
- You can change the response format or structure to match the expected format of your client application.
- Response mapping templates are applied in the "Integration Response" section of your API Gateway method configuration.

```
{
    "userId": "$input.json('$.id')",
    "userName": "$input.json('$.name')",
    "accountStatus": "$input.json('$.status')"
}
```



Stage Variables

- Environment variables for API gateway
- Can be changed without redeploying the API
- Stage variables are passed to **context** object in lambda functions
- Format to access stage variables in API gateway - `${stageVariables.variableName}`
- Example: Stage variables to point to Lambda Aliases

Stage variables can be used to point to different Lambda aliases. Each stage points to a different lambda alias depending on the value of a stage variable. To shift traffic, we can modify the alias weights without making changes to the API gateway.