API Gateways

An API Gateway is a server that acts as an entry point for clients to interact with a system that involves multiple services or microservices. It serves as a reverse proxy, routing requests from clients to the appropriate microservices.

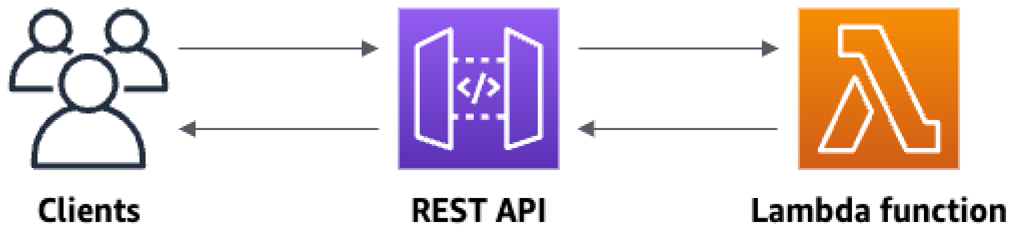
Key Functions of an API Gateway:

1. Routing: It directs incoming client requests to the correct microservice. For example, it can route requests to different services based on the URL, request type, or other parameters.
2. Aggregation: It can aggregate responses from multiple services into a single response, reducing the need for clients to make multiple requests.
3. Authentication and Authorization: The gateway can handle authentication and authorization, ensuring that only authorized clients can access certain services.
4. Rate Limiting and Throttling: It can limit the number of requests a client can make to prevent overloading the backend services.
5. Caching: Frequently requested data can be cached at the gateway level to improve performance and reduce load on backend services.
6. Logging and Monitoring: API Gateways often provide centralized logging and monitoring of requests, which is useful for tracking usage and diagnosing issues.
7. Request Transformation: It can modify incoming requests before sending them to the backend (e.g., change headers, parameters) and can also transform the response before sending it back to the client.
8. Load Balancing: It can distribute requests evenly across different instances of a microservice to ensure high availability and scalability.

Advantages:

- Single Entry Point: Simplifies the client-side architecture by providing one endpoint for all services.
- Decouples Clients from Services: Clients do not need to know the details of the backend services; they only interact with the API Gateway.

- Security: It acts as a barrier between the clients and microservices, centralizing security measures like authentication, authorization, and input validation.
- Scalability and Flexibility: You can scale individual services independently, and the API Gateway can manage how traffic is routed.
- Improved Client Experience: With features like caching and aggregation, the API Gateway can reduce the number of requests the client needs to make and improve response times.



Step 1: Create a Lambda function

You use a Lambda function for the backend of your API. Lambda runs your code only when needed and scales automatically, from a few requests per day to thousands per second.

For this exercise, you use a default Node.js function in the Lambda console.

To create a Lambda function

1. Sign in to the Lambda console at https://console.aws.amazon.com/lambda.
2. Choose Create function.
3. Under Basic information, for Function name, enter my-function.
4. For all other options, use the default setting.
5. Choose Create function.

The default Lambda function code should look similar to the following:

```
export const handler = async (event) => {

    const response = {

        statusCode: 200,

        body: JSON.stringify('The API Gateway REST API console is
great!'),
    };

    return response;

};
```

Step 2: Create a REST API

Next, you create a REST API with a root resource (/).

To create a REST API

1. Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
2. Do one of the following:
   - To create your first API, for REST API, choose Build.
   - If you've created an API before, choose Create API, and then choose Build for REST API.
3. For API name, enter my-rest-api.
4. (Optional) For Description, enter a description.
5. Keep API endpoint type set to Regional.
6. Choose Create API.

Step 3: Create a Lambda proxy integration

Next, you create an API method for your REST API on the root resource (/) and integrate the method with your Lambda function using a proxy integration. In a Lambda proxy integration, API Gateway passes the incoming request from the client directly to the Lambda function.

To create a Lambda proxy integration

1. Select the / resource, and then choose Create method.
2. For Method type, select ANY.
3. For Integration type, select Lambda.
4. Turn on Lambda proxy integration.
5. For Lambda function, enter my-function, and then select your Lambda function.
6. Choose Create method.

Step 4: Deploy your API

Next, you create an API deployment and associate it with a stage.

To deploy your API

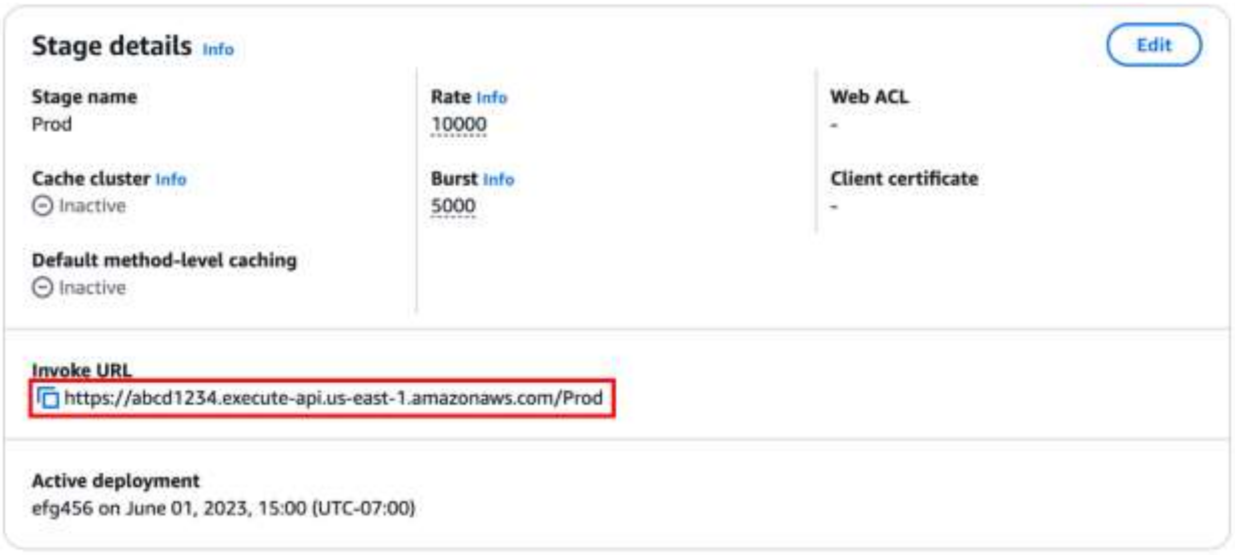
1. Choose Deploy API.
2. For Stage, select New stage.
3. For Stage name, enter Prod.
4. (Optional) For Description, enter a description.
5. Choose Deploy.

Now clients can call your API. To test your API before deploying it, you can optionally choose the ANY method, navigate to the Test tab, and then choose Test.

Step 5: Invoke your API

To invoke your API

1. From the main navigation pane, choose Stage.
2. Under Stage details, choose the copy icon to copy your API's invoke URL.



Stage name: Prod
Cache cluster: Inactive
Default method-level caching: Inactive
Rate: 10000
Burst: 5000
Web ACL: -
Client certificate: -
Invoke URL: https://abcd1234.execute-api.us-east-1.amazonaws.com/Prod
Active deployment: efg456 on June 01, 2023, 15:00 (UTC-07:00)

3. Enter the invoke URL in a web browser.

   The full URL should look like https://*abcd123*.execute-api.*us-east-2*.amazonaws.com/Prod.

Your browser sends a GET request to the API.

4. Verify your API's response. You should see the text "The API Gateway REST API console is great!" in your browser.

Links:

Choose between REST APIs and HTTP APIs - Amazon API Gateway

API Gateway HTTP APIs - Amazon API Gateway