**\_\_\_\_\_\_\_\_\_\_\_API GATEWAY (ESSENTIALSS)\_\_\_\_\_\_\_\_\_**

*--------------------------------------------------------------------------------------------*

***Purpose****: As part of the serverless architecture research here I will include interesting and relevant topics and features involving Lamba, API gateway, SQS, Dynamo, Ec2 and some other services that can provide an interesting feature within this type of architecture.*

*Note that all the following concepts are based on the following Udemy course and personal research, it is always a good idea to check for changes or updates from AWS to the lambda or other services used during this guide.*

<https://www.udemy.com/course/aws-lambda-serverless-architecture/>

The concept is simple, a full managed API within AWS in which you can define resources and methods like get, post etc. You can also link the API with different backends services or your own defined backend within Ec2, the case of interest for us is integrating API gateway with Lambda as the backend.

***Note: Usually this kind of infrastructure is not handled through the console of AWS, it is considered a good practice to use CloudFormation, SAM or other frameworks that allows you to deploy infrastructure using templates. I’m going to do a corresponding demo on different features using CloudFormation and SAM.***

***Keep in mind that SAM is just an abstraction of CloudFormation that is more oriented towards serverless design patterns but anything that can be done within SAM can be done within CFN and vice versa (there are some features within SAM like local testing and easy CLI deployments that can be useful so I will use SAM but inside of SAM I will also be using CFN syntax which is very similar)***

The API gateway service offers a lot of useful features like request validation and authorization as well as versioning and canary deployments which allow you divide your traffic within different stages of your API.

***Note: Aside from CFN and SAM there are third party tool providers like the serverless framework, some of the advantages of that framework is that it is designed to work with a multi-cloud architecture, and it has a community that develops plugins for it with extra functionality, one drawback is that the new features that AWS implement could be late for that framework.***

Concepts  **//**

There are different types of APIs we can create with this service:

**HTTP API:**

It offers less features than a rest API, but they are generally offered at a lower price and with a minimal technical overhead.

**REST API:**

Choose REST APIs if you need features such as API keys, per-client throttling, request validation, AWS WAF integration, or private API endpoints.

**Note: Find which one suits you better here** [**https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html**](https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html)

**Socket API:**

API Gateway WebSocket APIs are bidirectional. A client can send messages to a service, and services can independently send messages to clients. This bidirectional behavior enables richer client/service interactions because services can push data to clients without requiring clients to make an explicit request. WebSocket APIs are often used in real-time applications such as chat applications, collaboration platforms, multiplayer games, and financial trading platforms.

(Is just an open connection)

<https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-websocket-api.html4>

**Private Rest API:**

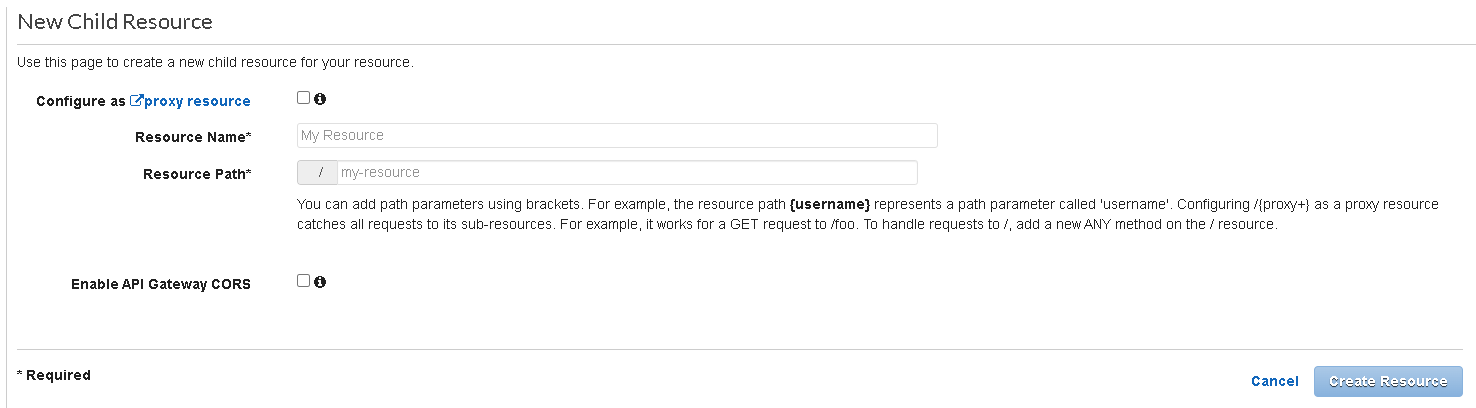
In general, is the same as a REST API but it is only accessible from within a VPC.

**Personal Note: To this research the Rest API seems to be the more suitable.**

In API gateway we have the concept of resources which is nothing more than the path for an endpoint, an example of a resource could be /users or /fetch-Data , each of those resources can be mapped to one or more http methods like GET,POST,DELETE and so on.

An API can be deployed to different stages, a stage is just like an environment, so with just one API we can run that same API within a test environment or a prod environment or custom environments which can al be versioned or integrated with different environment variables.

It offers fine grained access control as well (I will explain this later)



The resource path parameters are specified within curly braces like this /user/{id} = /user/1 this will then be accessible from our lambda function.

INTEGRATION TYPES //

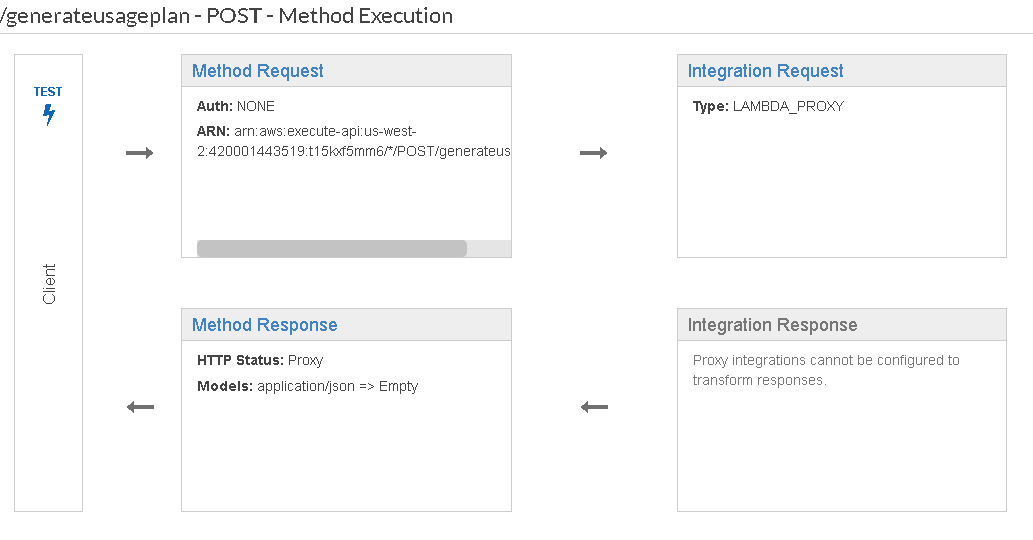
When creating a HTTP method for a resource we can specify different integration methods.

!!! THE FOLLOWING PART COULD BE REALLY IMPORTANT I SUGGEST TO PUT THE EFFOR TO DIVE DEEPER IN THE FOLLOWING CONCEPTS !!!!

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

===========================================================================



**Lambda function:**

It means that this method will be integrated/linked to a lambda function, in this case we must specify the region of the lambda function and the name, ARN or alias/version.

We can also select a timeout for the http method which is by default 29000 milliseconds which is 29 seconds.

**Note: Keep in mind that all this configuration is per method per resource, that means we can use lambda functions which are spread over different regions, and we can also use different integration types within our API.**

**HTTP:**

This allows you to connect your API with an HTTP endpoint, so like in VPC link now the API gateway API acts as a client and can call endpoint from the web or another API you have running in an EC2 instance for example with express.js.

**VPC link:**

This is an integration type that allows you to connect your API to different resources within a VPC in a secure way without exposing them to the public internet. It is like a two steps process in which a client calls API gateway and then API gateway acting now as a client calls the VPC endpoint.

**AWS service:**

Whit this type of integration the request is passed from API gateway towards a service you specify so a client could invoke a method from DynamoDB through our API gateway API for example.

**Mock:**

With this integration type there is no other service as a backend the response is purely handled by API gateway on itself using mappings and transformations (this features will be discussed later on).

**Note: This could be useful for testing, very simple workflows that doesn’t require a full lambda execution or request correction validation.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**I highly recommend reading this documentation since there are a lot of useful features within the different integration types:**

[**https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-api-integration-types.html**](https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-api-integration-types.html)

Non proxy integration vs proxy integration:

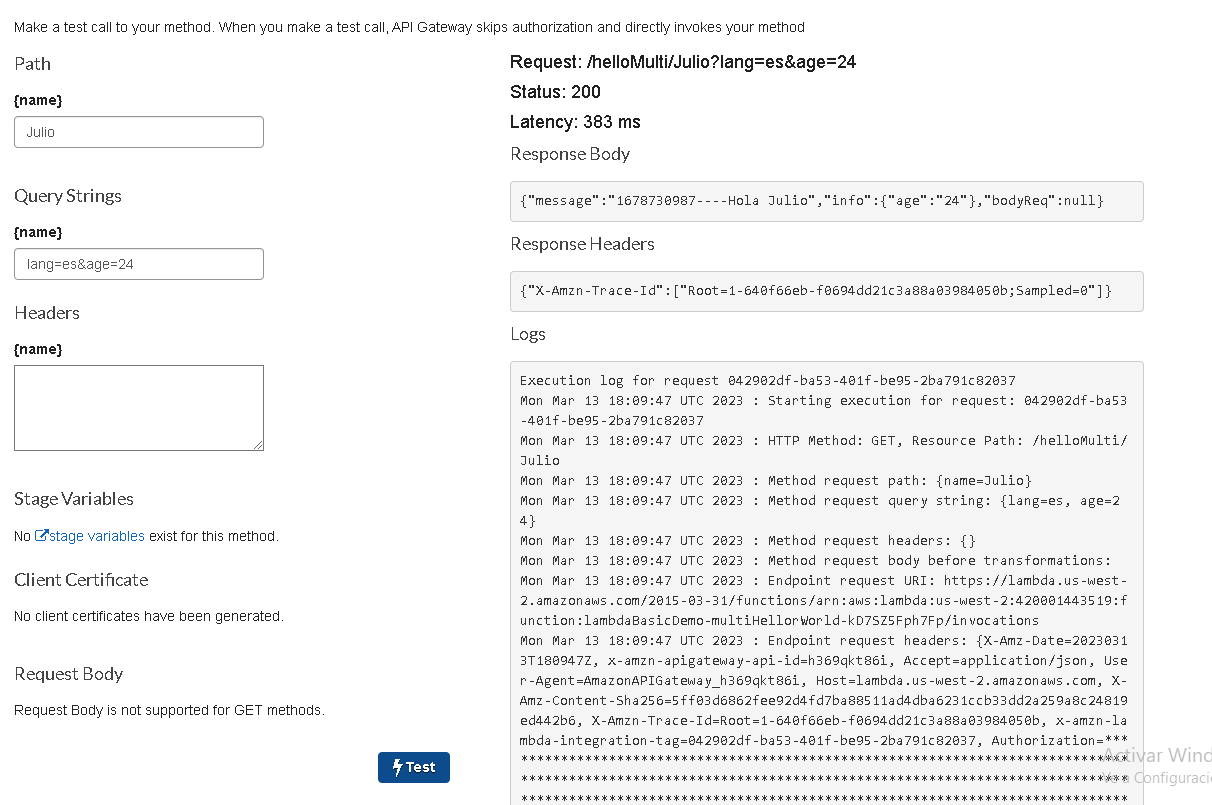
In a proxy integration API gateway receives all the HTTP request from the client and forwards it as it is to the backend service and similarly API gateway receives the response from the service and send it to the client as it is.

In a non-proxy integration API gateway allows you to receive and return only a specified part of the incoming request and the response, so here you can modify how the request reaches the backend and how the response reaches the client, to achieve this API gateway uses mapping templates and request/response transformations.

Is important to differentiate these two values since depending on the integration type you choose you can use one or the other, or both (in the link above you can see which proxy integration type does is integration type support)

**Note: API gateway needs to have permissions to invoke the lambda functions, more specifically the resource and method that are trying to invoke the lambda function.**

One of the useful features the API gateway console offers, is being able to test your endpoints/resources without having to use postman or other third-party tools, which are still useful to test that the deployment was successful and that it can be reached from anywhere but in the test overview you can test for debug purposes before deploying in a more faster and simpler way.



Before any changes can be accessible from the internet or the overall main workflow of the API you need to deploy the API to a stage, for example deploying your changes to a development stage, then a URL will be generated, and you can use it to call you API endpoint from postman for example. Also keep in mind that changes on backend services doesn’t require a redeployment from API gateway since they are isolated, only changes within API gateway requires a redeployment, CFN usually and practically always detects when a resource needs to be updated/re-deployed.

CORS  **//**