

EJERCICIO

API Gateway y AWS Lambda

Objetivo.

Desarrollar un pequeño API que su backend sea AWS Lambda.

Paso 1 – Crear funciones Lambda

Vamos a escribir dos funciones Lambdas para nuestra primera API. La primera función nos dara la lista de todos los Buckets de S3 que tenemos y la segunda nos presentara la lista de Objetos que hay dentro de un Bucket.

Nota: Recuerden que es necesario dar permisos al ROL de cada función.

Función 1 - GetBuckets

Entrada:

```
{  
  
}
```

Salida:

```
{  
  "Buckets": [ { "Name": "NOMBRE DEL BUCKET"}, ... ]  
}
```

```
1  import json  
2  import boto3  
3  
4  
5  def lambda_handler(event, context):  
6  
7      client = boto3.client('s3')  
8      response = client.list_buckets()  
9  
10     buckets = { "Buckets": [] }  
11     for bucket in response['Buckets']:  
12         buckets['Buckets'].append({"Name": bucket['Name']})  
13  
14     return buckets  
15
```

Función 2 – GetObjects

Entrada:

```
{
  "Bucket": "NOMBRE DEL BUCKET"
}
```

Salida:

```
{
  "Objects": [ {"Key": "NOMBRE DEL OBJETO", "Size" #####}, ... ]
}
```

```
import json
import boto3

def lambda_handler(event, context):

    client = boto3.client('s3')
    response = client.list_objects_v2(Bucket=event['Bucket'])

    objects = { "Objects": [] }
    for obj in response['Contents']:
        objects['Objects'].append({"Key": obj['Key'], "Size": obj['Size']})

    return objects
```

Paso 2 – Crear API Gateway

1. Ingresamos desde la consola al Servicio de **Amazon API Gateway**
2. Damos click en **Build** una **REST API**
3. La pantalla que se nos presenta la llenamos de la siguiente forma:

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 ☐ Import from Swagger or Open API 3 ☐ Example API

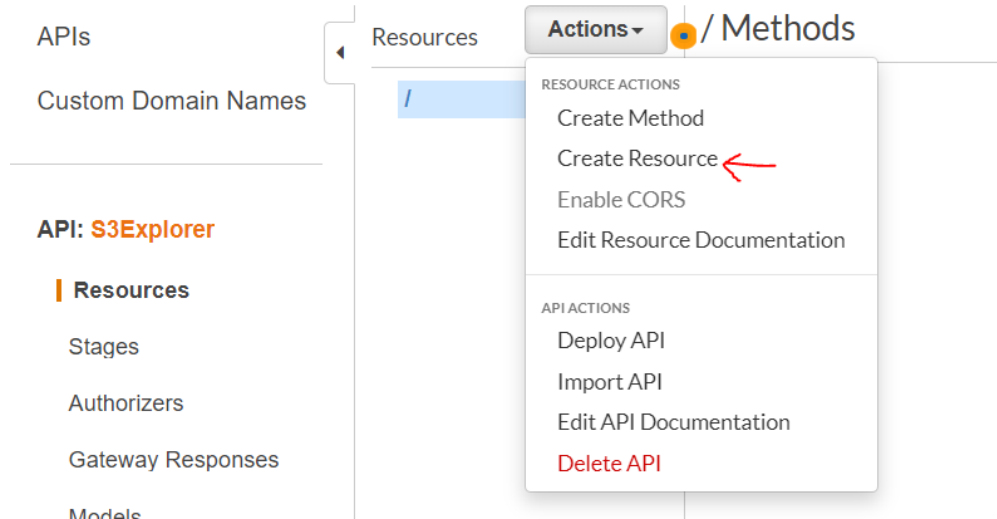
Settings

Choose a friendly name and description for your API.

API name*	<input type="text" value="S3Explorer"/>
Description	<input type="text"/>
Endpoint Type	<input type="text" value="Regional"/> ⓘ


* Required


4. Generamos un nuevo Recurso.



Y llenamos la pantalla de la siguiente manera:

New Child Resource


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

Configure as [proxy resource](#) ☐ 

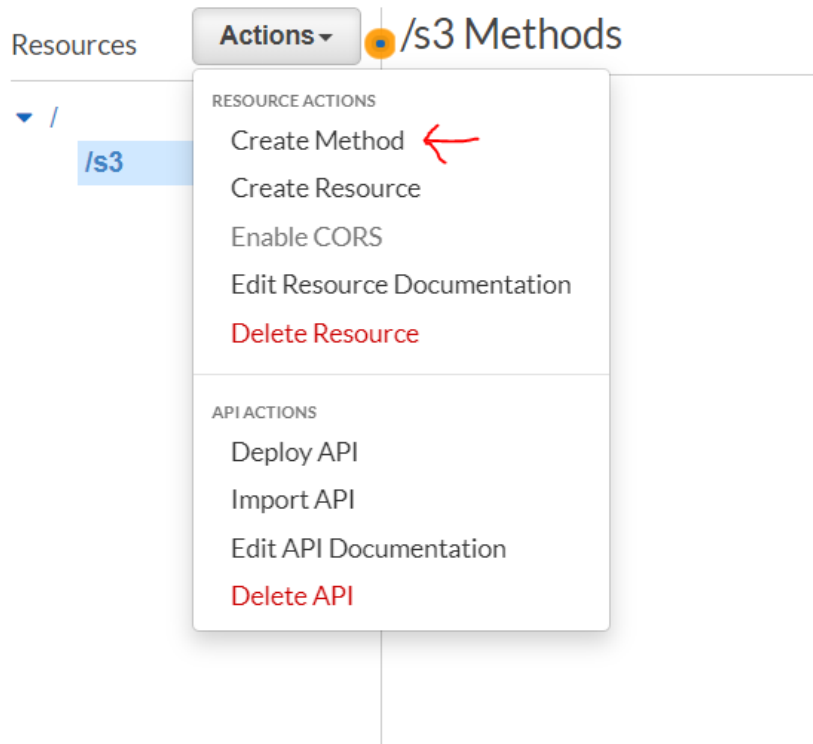
Resource Name*

Resource Path*

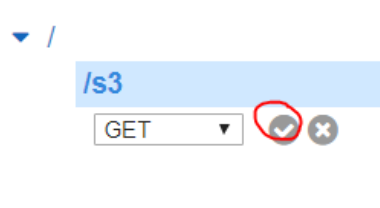
You can add path parameters using brackets. For example, the resource called 'username'. Configuring `/ {proxy+}` as a proxy resource catches all works for a GET request to `/foo`. To handle requests to `/`, add a new ANY

Enable API Gateway CORS ☐ 

5. Seleccionado nuestro recurso S3 damos click en **Create Method**



Ponemos que será GET y damos click en la palomita.



En la pantalla que se nos abre del lado derecho capturamos lo siguiente:

/s3 - GET - Setup

Choose the integration point for your new method.

Integration type ☒ Lambda Function ⓘ

☐ HTTP ⓘ

☐ Mock ⓘ

☐ AWS Service ⓘ

☐ VPC Link ⓘ

Use Lambda Proxy integration ☐ ⓘ **NO**

Lambda Region

Lambda Function

Use Default Timeout ☒ ⓘ

En la siguiente pantalla damos click en **TEST** y debemos tener la siguiente salida:

← Method Execution /s3 - GET - Method Test

Make a test call to your method with the provided input

Path

No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.

Query Strings

{s3}

Headers

{s3}

Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg.
Accept:application/json.

Stage Variables

Request: /s3

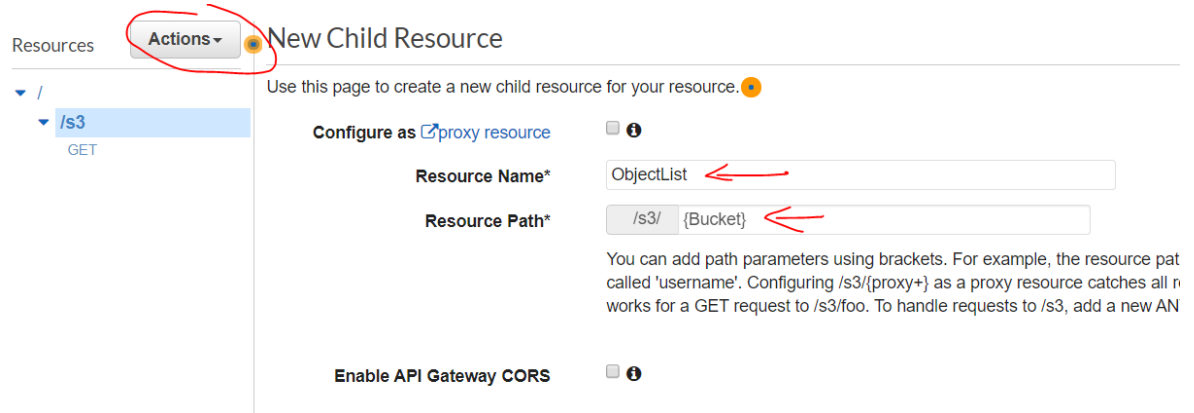
Status: 200

Latency: 1918 ms

Response Body

```
{
  "Buckets": [
    {
      "Name": "parres-bucket2"
    },
    {
      "Name": "parres-bucket3"
    },
    {
      "Name": "parres-bucket4"
    },
    {
      "Name": "parres-mibucketfeliz"
    }
  ]
}
```

6. Generamos un nuevo recurso para el listado de Objetos, por debajo de S3



Resources **Actions** New Child Resource

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

Configure as [proxy resource](#) ☒ ⓘ

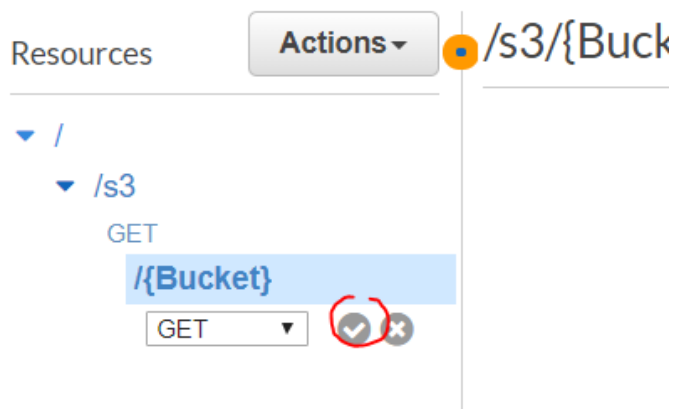
Resource Name* ObjectList ←

Resource Path* /s3/ {Bucket} ←

You can add path parameters using brackets. For example, the resource path called 'username'. Configuring /s3/(proxy+) as a proxy resource catches all requests for a GET request to /s3/foo. To handle requests to /s3, add a new AN

Enable API Gateway CORS ☐ ⓘ

Y sobre el nuevo recurso creamos un nuevo método GET



Resources **Actions** /s3/{Bucket}

/

/s3

GET

/ {Bucket}

GET ✓ ✕

Y lo configuramos de la siguiente manera:

/s3/{Bucket} - GET - Setup

Choose the integration point for your new method.

Integration type ☒ Lambda Function ⓘ

☐ HTTP ⓘ

☐ Mock ⓘ

☐ AWS Service ⓘ

☐ VPC Link ⓘ

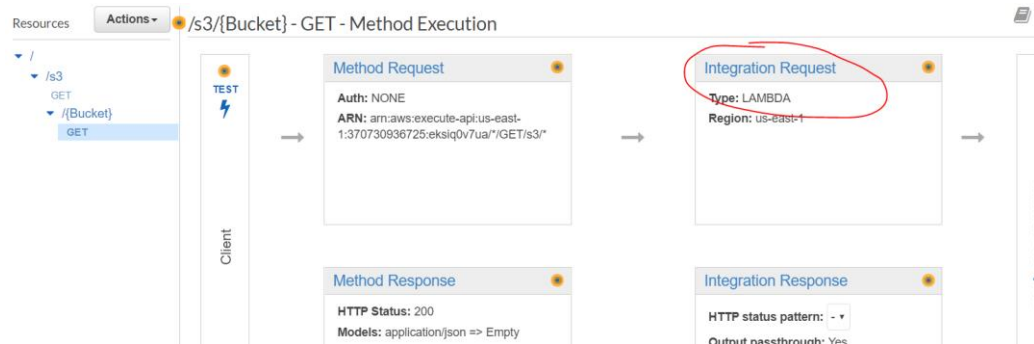
Use Lambda Proxy integration ☐ ⓘ ← NO

Lambda Region

Lambda Function

Use Default Timeout ☒ ⓘ

Con el objetivo de “preparar” la entrada a como lo espera nuestra función, modificamos la **Integration Request**



Despues de dar Click en la pagina siguiente nos vamos al final a **Mapping Templates** y damos click en **Add mapping template**

▼ Mapping Templates ⓘ

Request body passthrough ☒ When no template matches the request Content-Type header ⓘ

☐ When there are no templates defined (recommended) ⓘ

☐ Never ⓘ

Content-Type

⊕ Add mapping template

Despues de dar click en la palomita nos aparecerá un aviso al cual le daremos click en **No, use current settings.**

Change passthrough behavior

⚠ Your current passthrough behavior will pass all request payloads directly to the endpoint without transformation, unless there is a match for the incoming Content-Type. Do you want to secure this integration to only allow requests that match one of your defined Content-Types?

No, use current settings

Yes, secure this integration

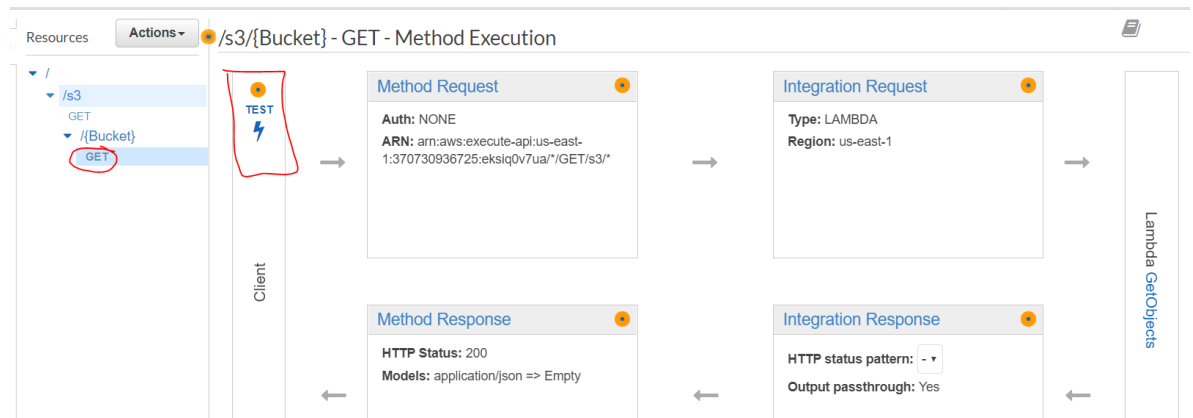
Nos aparecerá un cuadro de texto al final, donde seleccionamos de **Generate Template** EMPTY y completamos conforme la siguiente imagen:

application/json

Generate template: Empty

```
1 #set($inputRoot = $input.path('$'))
2 {
3   "Bucket": "$input.params('Bucket')|"
4 }
```

En nuestro árbol damos click de nueva cuenta con GET y damos click en TEST.



En la siguiente ventana completamos conforme lo siguiente y damos click en **TEST**

Make a test call to your method with the provided input

Path

{Bucket}

parres-mibucketfeliz

Query Strings

{Bucket}

param1=value1¶m2=value2

Headers

{Bucket}

Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg.
Accept:application/json.

Request: /s3/parres-mibucketfeliz

Status: 200

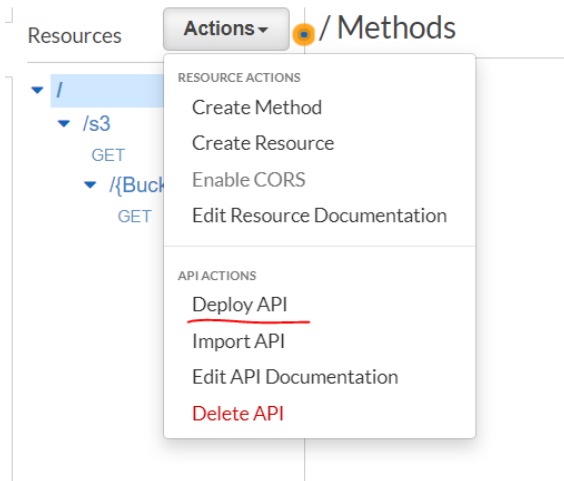
Latency: 437 ms

Response Body

```
{
  "objects": [
    {
      "Key": "Academy years 15.pdf",
      "Size": 522981
    },
    {
      "Key": "Cubierta_Albercas Guadalajara.pdf",
      "Size": 115908
    },
    {
      "Key": "Cubierta_medidascorrectas_cleanwater (1).pdf",
      "Size": 581848
    }
  ]
}
```

Paso 3 – Deploy del API

Una vez terminadas las pruebas damos click en la / de nuestro árbol y del menú Actions damos click en **Deploy API**



La venta que nos sale la completamos conforme lo siguiente:

Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage	<div>[New Stage]</div>
Stage name*	<div>prod</div>
Stage description	<div></div>
Deployment description	<div></div>

Cancel

Deploy

En la siguiente pantalla nos aparecerá una URL accesible desde Internet para probar nuestra API.

prod Stage Editor

Invoke URL: https://eksiq0v7ua.execute-api.us-east-1.amazonaws.com/

Settings

Logs/Tracing

Stage Variables

SDK Generation

Export

Deployment History

Cache Settings

Ejercicio.

Diseñar un AWS API que nos permita hacer las operaciones aritméticas básicas entre dos dígitos.

Esquema de acceso a la API

/calculator/{a}/{b}/{operator}

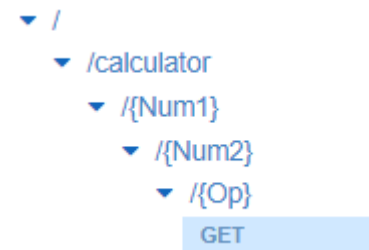
a,b = Números reales positivos.

operator = [+|-|*|div]

Salida esperada

JSON con la siguiente estructura

```
{
  "a": "Number",
  "b": "Number",
  "operator": "String",
  "c": "Number" | "String"
}
```



Deberá estar soportada por una única función Lambda, la cual debe validar que la operación sea posible, y en caso contrario regresa la palabra **NaN**