

# ***Documentation***

*Microservices 4<sup>th</sup> - 5<sup>th</sup> Week*



Microservices Architecture (Introduction)

Teacher: Mr. Zheng Li

Student: Juan Albornoz

## Index

1. Setting up the Lambda function. [\[AWS Lambda Service\]](#) *(Pages 3-4)*
2. Setting up the API Gateway service and connecting with Lambda. [\[AWS API Gateway\]](#) *(Pages 5-10)*
3. Working on the testMap.html file. *(Pages 11-13)*
4. Building my Lambda Function, clustering and testing it via API Gateway. *(Pages 14-17)*
5. Changing testMap.html to testMapv2.html file and implementing jQuery's AJAX() method in the script. [\[HTML, JavaScript\]](#) *(Pages 18-19)*
6. Testing locally. *(Pages 20-21)*
7. Deployment with S3 Bucket Service. [\[AWS S3 Service\]](#) *(Pages 22-26)*

## 1. Setting up the Lambda function.

1.1. Create a function with the **Author from scratch** option, as you see in the next screenshot:

**Create function** Info

Choose one of the following options to create your function.

**Author from scratch**  
Start with a simple Hello World example.

**Use a blueprint**  
Build a Lambda application from sample code and configuration presets for common use cases.

**Browse serverless app repository**  
Deploy a sample Lambda application from the AWS Serverless Application Repository.

1.2. My function's name is **lambda\_clustering**

1.3. The runtime language it's **Python 3.7**, as you see in the next screenshot:

**Basic information**

**Function name**  
Enter a name that describes the purpose of your function.  
  
Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime** Info  
Choose the language to use to write your function.

**Permissions** Info  
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

**Choose or create an execution role**

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☐ Create a new role with basic Lambda permissions  
☐ Use an existing role  
☒ Create a new role from AWS policy templates

Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

1.4. Create a new role from AWS templates, the parameters are the following:

1.4.1. The role name is **clusteringRole**

1.4.2. The policy template is **Simple microservice permissions**, as you see in the next screenshot:

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☐ Create a new role with basic Lambda permissions  
☐ Use an existing role  
☒ Create a new role from AWS policy templates

Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

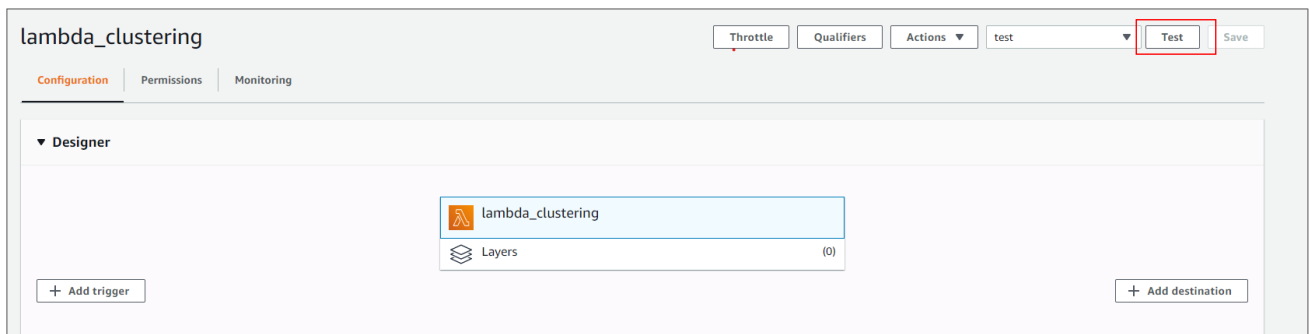
**Role name**  
Enter a name for your new role.  
  
Use only letters, numbers, hyphens, or underscores with no spaces.

**Policy templates - optional** Info  
Choose one or more policy templates.

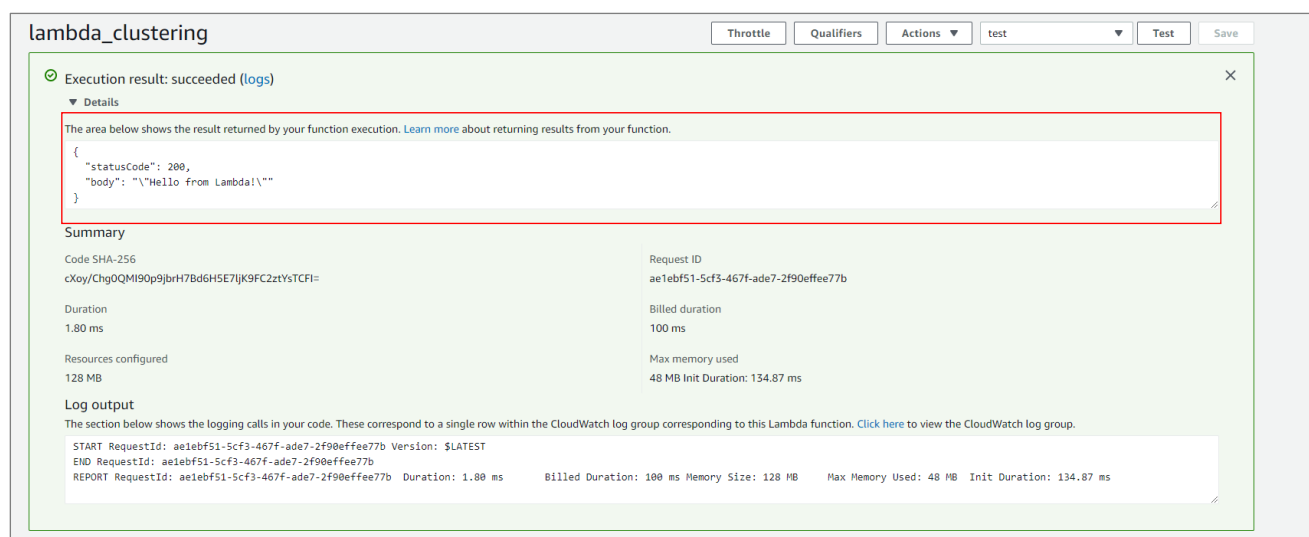
DynamoDB

Cancel **Create function**

1.5. Now I can test my function by using the **Test** button in the upper part of the Lambda module, as you see in the next screenshot:



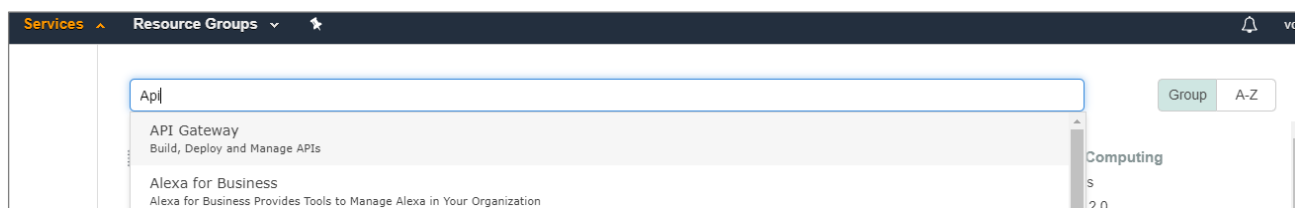
1.6. Then, the results of executing it successfully and his logs, as you see in the next screenshot:



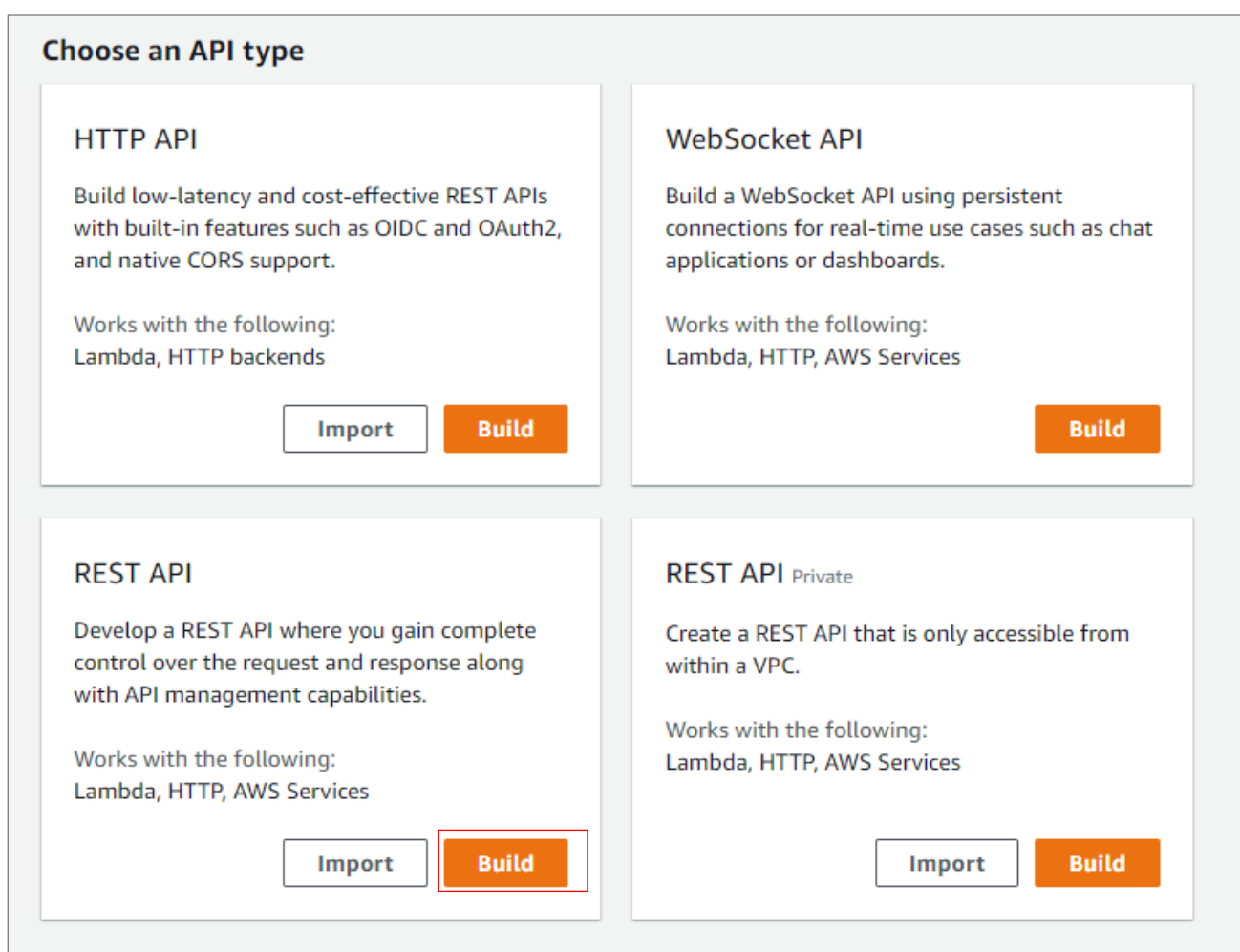
(\*) I leave it here for now setting up my Lambda function, so now it's time for the API Gateway service. In a later section I'm going to go back to my Lambda function more deeply.

## 2. Setting up the API Gateway service and connecting with Lambda.

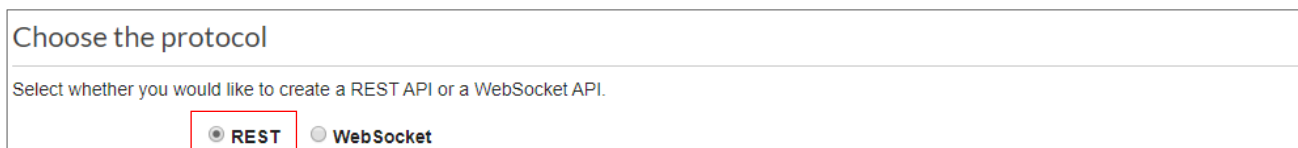
2.1. Search and open the API Gateway service, as you see in the next screenshot:



2.2. Choose REST API section and go to [Build](#) one, as you see in the next screenshot:



2.3. Choose protocol [REST](#), as you see in the next screenshot:



2.4. Select **New API** type, as you see in the next screenshot:

### 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

2.5. Give it a name to your API, I named it **mapLambdaAPI**, also a Description (optional) and leave the Endpoint Type the same. Then you must do the **Create API**, as you see in the next screenshot:

#### Settings

Choose a friendly name and description for your API.

API name\*   
Description   
Endpoint Type

\* Required

Create API

2.6. In the **Actions** button go to **Create Resource**, as you see in the next screenshot:

Resources **Actions** **New Child Resource**

create a new child resource for your resource.

**RESOURCE ACTIONS**

- Create Method
- Create Resource**
- Enable CORS
- Edit Resource Documentation

**API ACTIONS**

- Deploy API
- Import API
- Edit API Documentation
- Delete API

**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.

**API Gateway CORS**

\* Required

Cancel Create Resource

2.7. I named the resource **zoom**

2.8. The Resource path is called **/zoom**

2.9. Also I activated the option **"Enable API Gateway CORS"** for terms of calling my API in a later section. Then hit the **Create Resource** button, as you see in the next screenshot:

Resources **Actions** **New Child Resource**

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

**Configure as proxy resource**

**Resource Name\***

**Resource Path\***

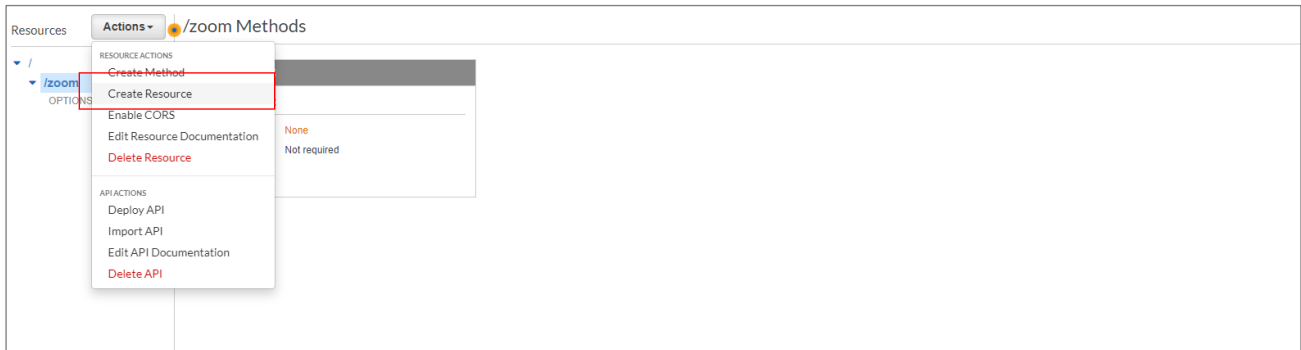
You can add path parameters using brackets. For example, the resource path **{username}** represents a path parameter called 'username'. Configuring **/ (proxy+)** as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to **/foo**. To handle requests to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

**Enable API Gateway CORS** ☒

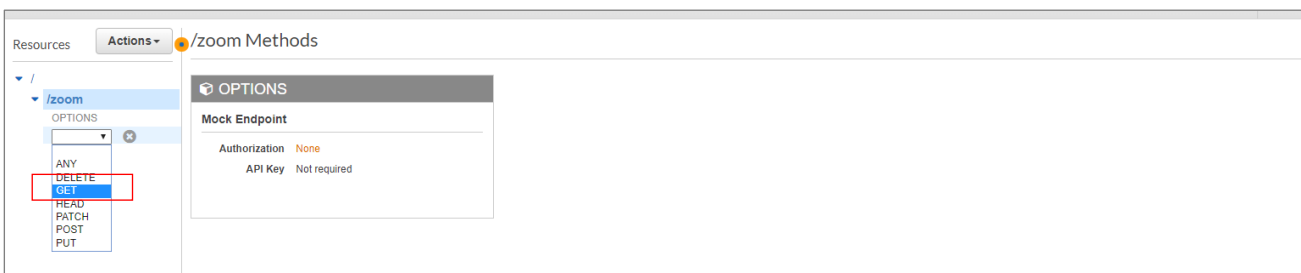
\* Required

Cancel **Create Resource**

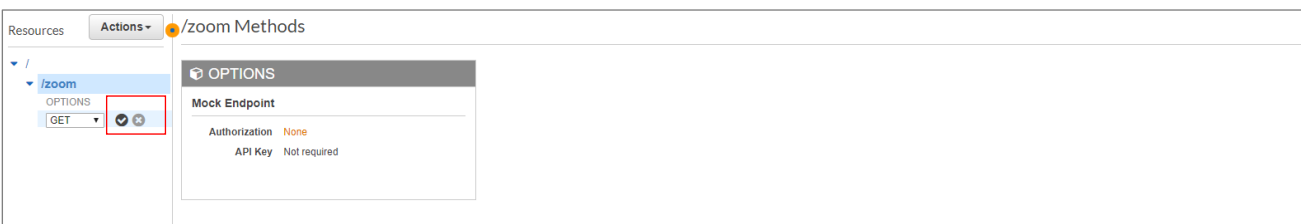
2.10. Inside you resource section, go to [Actions](#) again and then [Create Method](#), as you see in the next screenshot:



2.11. Choose a [GET](#) one from the listed options in the panel, as you see in the next screenshot:



2.12. And then press the [tick button](#), as you see in the next screenshot:



2.13. Choose [Lambda Function](#) as the Integration type

2.14. Put your [Lambda Region](#) as your region were are you working in, my region is [us-east-1](#).

2.15. Search your [Lambda Function](#), my function is [lambda\\_clustering](#).

2.16. Then press the [Save button](#), as you see in the next screenshot:

Resources Actions ▾ /zoom - GET - Setup

Choose the integration point for your new method.

Integration type **Lambda Function** ⓘ

- HTTP ⓘ
- Mock ⓘ
- AWS Service ⓘ
- VPC Link ⓘ

Use Lambda Proxy integration ⓘ

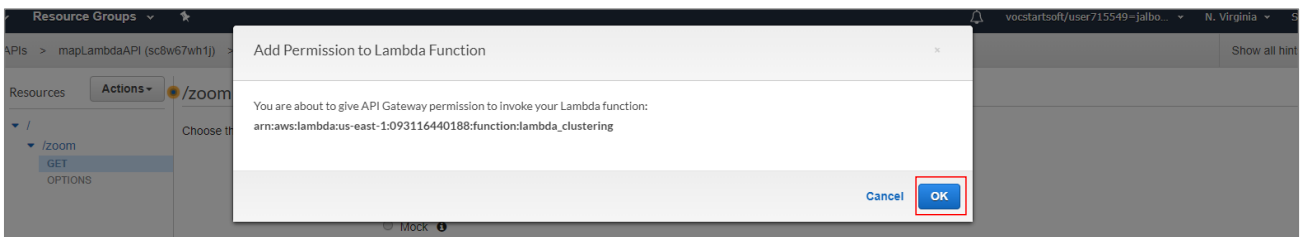
Lambda Region **us-east-1**

Lambda Function **lambda\_clustering** ⓘ

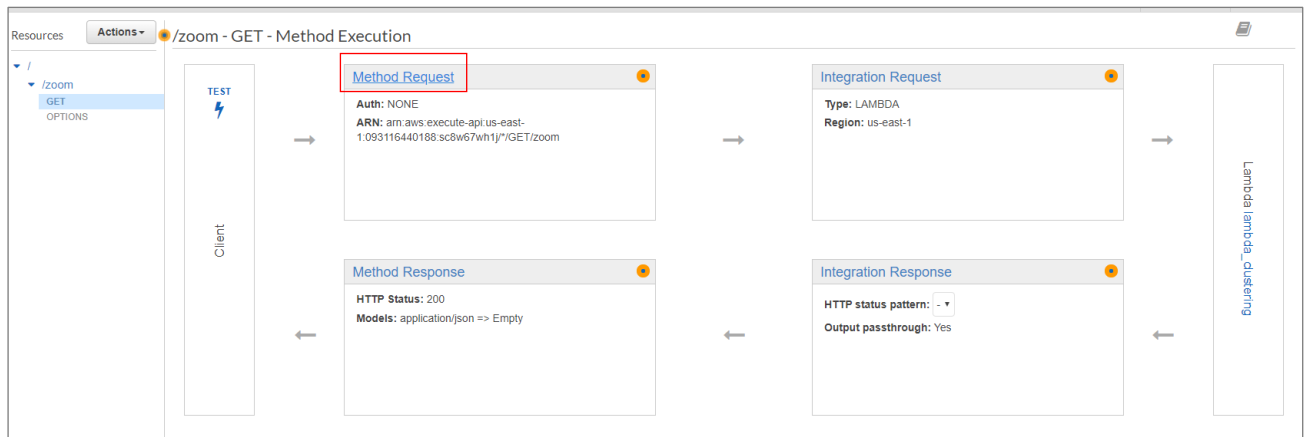
Use Default Timeout ☒ ⓘ

Save

2.17. Then you must Add permissions to your Lambda Function by pressing **OK** button, as you see in the next screenshot:



2.18. Go to **Method Request** section, as you see in the next screenshot:



2.19. Leave the **Authorization** in **NONE**, also the **Request Validator** in **NONE** and the **API Key Required** in **false**, as you see in the next screenshot:

Resources Actions ▾ /zoom - GET - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Settings ⓘ

Authorization **NONE** ⓘ

Request Validator **NONE** ⓘ

API Key Required **false** ⓘ



2.20. Go to [URL Query String Parameters](#) section and then [Add query string](#), as you see in the next screenshot:

Name	Required	Caching
No query strings		

[Add query string](#)

2.21. Then create it by giving it a [name](#), mine is [level](#) and press the [tick button](#) in the right side, as you see in the next screenshot:

Name	Required	Caching
level		

2.22. Now I have my API Gateway created and ready for testing it, but first I'm going to Deploy it by giving the [Deploy API](#) option in the [Actions](#) section, as you see in the next screenshot:

Resources

Actions - /zoom - GET - Method Execution

METHOD ACTIONS

- Edit Method Documentation
- Delete Method

RESOURCE ACTIONS

- Create Method
- Create Resource
- Enable CORS
- Edit Resource Documentation
- Delete Resource

APIS ACTIONS

- Deploy API
- Import API
- Edit API Documentation
- Delete API

Method Request

Auth: NONE

ARN: arn:aws:execute-api:us-east-1:093116440188:sc8w67wh1j/\*GET/zoom

Integration Request

Type: LAMBDA

Region: us-east-1

Method Response

HTTP Status: 200

Models: application/json => Empty

Integration Response

HTTP status pattern: -

Output passthrough: Yes

Lambda lambda\_clustering

2.23. Create a [New Stage](#) for deploying it. Give it a [Stage name](#), mine is [testingAPI](#). Then press the [Deploy button](#), as you see in the next screenshot:

Resource Groups

APIs > mapLambdaAPI (sc8w67wh1j) > Resources > /zoom

Resources

Actions - /zoom - GET - Method Execution

TEST

Client

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 [New Stage]

Stage name\* testingAPI

Stage description

Deployment description

Cancel Deploy

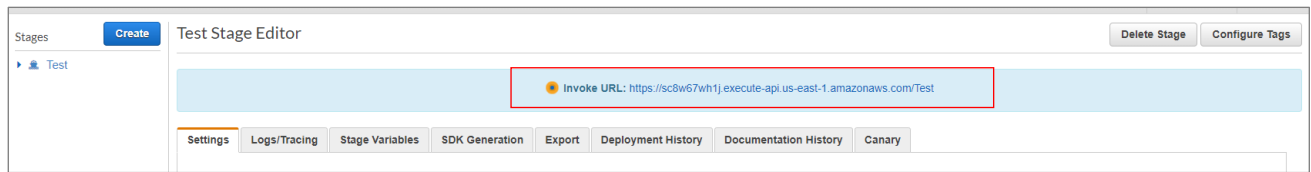
Integration Request

Type: LAMBDA

Region: us-east-1

Integration Response

2.24. Now I have an “Invoke URL” <https://sc8w67wh1j.execute-api.us-east-1.amazonaws.com/Test> that I must use later for calling my Lambda function from my script, also for doing the testing parts.



(\*) 2.9 Notice that activating the API Gateway CORS allows to execute the lambda function from the script in my html file and receive the data of my clusters in a later step.

### 3. Working on the testMap.html file.

- The first step I did it's to choose the sightseeing places from the web. My places I chose are the next:

**Valle de la Luna:**

Latitude: -22.924155, Longitude: -68.288103

**Humberstone y Santa Laura:**

Latitude: -20.21015, Longitude: -69.801569

**Chuquicamata:**

Latitude: -22.316964, Longitude: -68.933477

**Morro de Arica:**

Latitude: -18.479432, Longitude: -70.323676

**Playa Hornitos:**

Latitude: -22.915622, Longitude: -70.290275

**Laguna Cejar:**

Latitude: -23.063035, Longitude: -68.213378

**Laguna Parinacota:**

Latitude: -19.060498, Longitude: -69.250661

**Salar de Tara:**

Latitude: -23.060296, Longitude: -67.241466

**Pica:**

Latitude: -20.491570, Longitude: -69.329696

3.1. The container of my locations is a [List of Lists](#) type, with the [name](#), the [latitude](#) and the [longitude](#), as you can see in the next screenshot.

```
<!-- LOCATIONS CONTAINER -->
var locations = [
  ['Valle de la Luna', -22.924155, -68.288103, 1],
  ['Humberstone y Santa Laura', -20.210155, -69.801569, 2],
  ['Chuquicamata', -22.316964, -68.933477, 3],
  ['Morro de Arica', -18.479432, -70.323676, 4],
  ['Playa Hornitos', -22.915622, -70.290275, 5],
  ['Laguna Cejar', -23.063035, -68.213378, 6],
  ['Laguna Parinacota', -19.060498, -69.250661, 7],
  ['Salar de Tara', -23.060296, -67.241466, 8],
  ['Pica', -20.491570, -69.329696, 9],
];
```

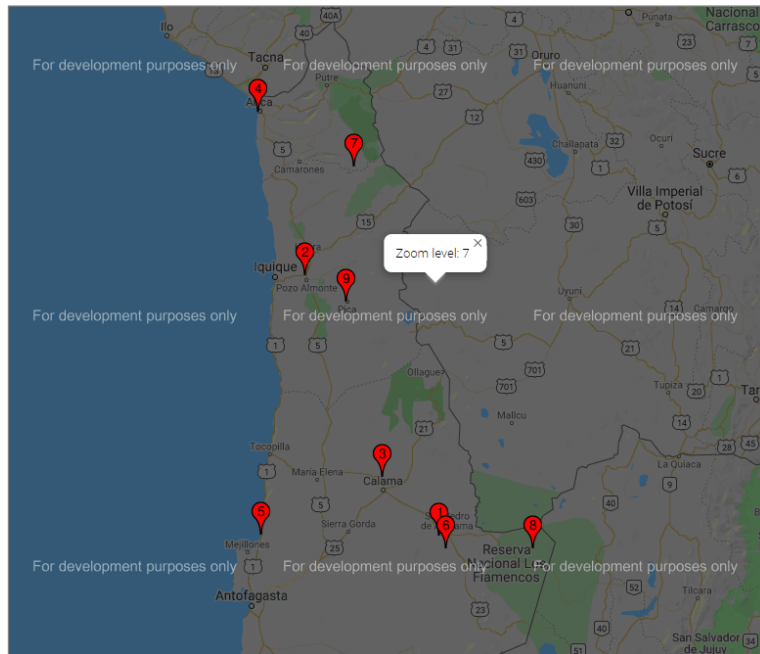
3.2. In the first part of the `InitMap()` function I changed the default `zoom` parameter and also the default `center` parameter of the screen to a new one calculated by hand. This following the next rule:

$$( (x1 + \dots + x9)/9, (y1 + \dots + y9)/9 )$$

3.3. Also I changed the way of marking the location's icons on the map by giving a formatted link into the loop. The icons I found are also from the Google Map API. The next screenshot shows the modified parameters.

```
...function InitMap(){
...    <!--DEFAULT ZOOM LEVEL CHANGED TO 4-->
...    var map = new google.maps.Map(document.getElementById('map'),{
...        zoom: 4,
...        center: new google.maps.LatLng(-21.391303, -69.185811),
...        mapTypeId: google.maps.MapTypeId.ROADMAP
...    });
...    <!--MARKER INITIALIZATION AND SETTINGS-->
...    var infowindow = new google.maps.InfoWindow();
...    var marker, i;
...    for (i = 0; i < locations.length; i++){
...        marker = new google.maps.Marker({
...            position: new google.maps.LatLng(
...                locations[i][1],+locations[i][2]
...            ),
...            map: map,
...            icon: 'https://chart.googleapis.com/chart?chst=d_map_pin_letter&chld=' + (i+1) + '|FF0000|000000'
...        });
...        google.maps.event.addListener(marker, 'click', (function (marker, i){
...            return function (){
...                infowindow.setContent(locations[i][0]);
...                infowindow.open(map, marker);
...            }
...        }))(marker, i);
...    }
...}
```

3.4. The zoom's data and his windows were not difficult to activate and I have a previewed sightseeing places numbered from 1 to 9 as it follows:



3.5. In the second part of the `InitMap()` function there is the `URL` for calling my deployed API. Notice that there are parameters for getting this in the right way. These parameters are `zoom` and `level` (resource and parameter) just established in the previous section. The next screenshot shows the AJAX script inside the listener:

```

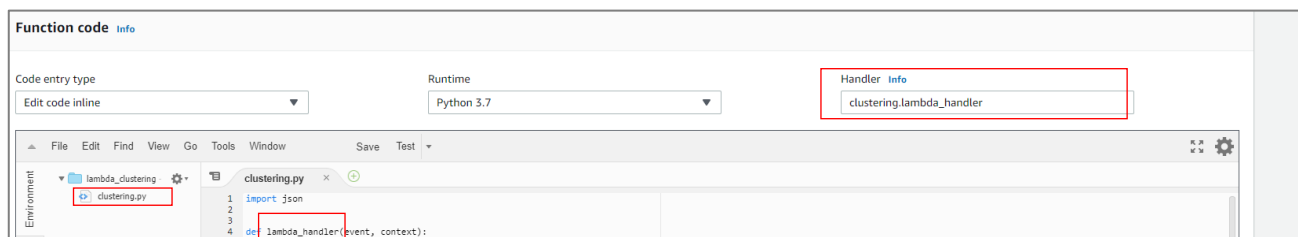
<!--ZOOM WINDOWS MESSAGE INITIALIZATION -->
var zoomwindow = new google.maps.InfoWindow({
  content: 'Zoom Level',
  position: new google.maps.LatLng(-21.391303, -69.185811)});
zoomwindow.open(map);
<!--ZOOMING EVENT AND MESSAGE-->
map.addListener('zoom_changed', function(){
  $.ajax({
    url: 'https://sc8w67wh1j.execute-api.us-east-1.amazonaws.com/Test/zoom?level=' + (map.getZoom()),
    headers: {'Content-Type': 'application/json'},
    type: 'GET',
    crossDomain: true,
    dataType: 'json',
    success: function(data) {
      window.alert(JSON.stringify(data));
    }
  });
  zoomwindow.setContent('Zoom level: ' + map.getZoom());
});

```

(\*) In a later section I'm going to approach more deeply into this script. First I must explain more closely the build of my Lambda Function, also by testing it with the API Gateway service.

## 4. Building my Lambda Function, clustering and testing it via API Gateway.

4.1. First I named `clustering.py` my python's file. The `def function` of the file must match the `handler` info in the right side, as you can see in the next screenshot:



4.2. Then I separate each of the zoom levels by `IF` and `ELIF` blocks. Inside of the conditionals I have my `location's clusters` in structures defined as `List of Dictionaries`. The next screenshot shows my lambda function structure:

```
def lambda_handler(event, context):
    zoom_level = event["queryStringParameters"]["level"]

    if 4 == int(zoom_level):
        clusters = {
            "cluster_size": 9,
            "latitude": -21.391303,
            "longitude": -69.185811
        }

    elif 5 == int(zoom_level):
        clusters = [
            {
                'cluster_size': 4,
                "latitude": -22.84111,
                "longitude": -68.169106
            },
            {
                'cluster_size': 3,
                "latitude": -21.205897,
                "longitude": -69.80718
            },
            {
                'cluster_size': 2,
                "latitude": -18.769965,
                "longitude": -69.787169
            }
        ]

    elif 6 == int(zoom_level):
        clusters = [
            {
                'cluster_size': 1,
                "latitude": -22.84111,
                "longitude": -68.169106
            },
            {
                'cluster_size': 3,
                "latitude": -22.768051,
                "longitude": -68.478319
            },
            {
                'cluster_size': 1,
                "latitude": -22.915622,
                "longitude": -70.290275
            },
            {
                'cluster_size': 1,
                "latitude": -20.491570,
                "longitude": -69.329696
            },
            {
                'cluster_size': 1,
                "latitude": -20.21015,
                "longitude": -69.801569
            }
        ]
```

(\*) Notice that the `zoom_level` parameter is defined as an event that receive the `"level"` parameter from the API.

```

else:
    clusters = [
        {
            'cluster_size': 1,
            "latitude": -22.84111,
            "longitude": -68.169106
        },
        {
            'cluster_size': 1,
            "latitude": -22.924155,
            "longitude": -68.288103
        },
        {
            'cluster_size': 1,
            "latitude": -22.316964,
            "longitude": -68.933477
        },
        {
            'cluster_size': 1,
            "latitude": -23.063035,
            "longitude": -68.213378
        },
        {
            'cluster_size': 1,
            "latitude": -22.915622,
            "longitude": -70.290275
        },
        {
            'cluster_size': 1,
            "latitude": -20.491570,
            "longitude": -69.329696
        },
        {
            'cluster_size': 1,
            "latitude": -20.21015,
            "longitude": -69.801569
        },
        {
            'cluster_size': 1,
            "latitude": -19.060498,
            "longitude": -69.250661
        },
        {
            'cluster_size': 1,
            "latitude": -18.479432,
            "longitude": -70.323676
        }
    ]

    return({
        'isBase64Encoded': True,
        'statusCode': 200,
        'headers': {"Access-Control-Allow-Origin": ":"},
        'body': json.dumps(clusters)
    })

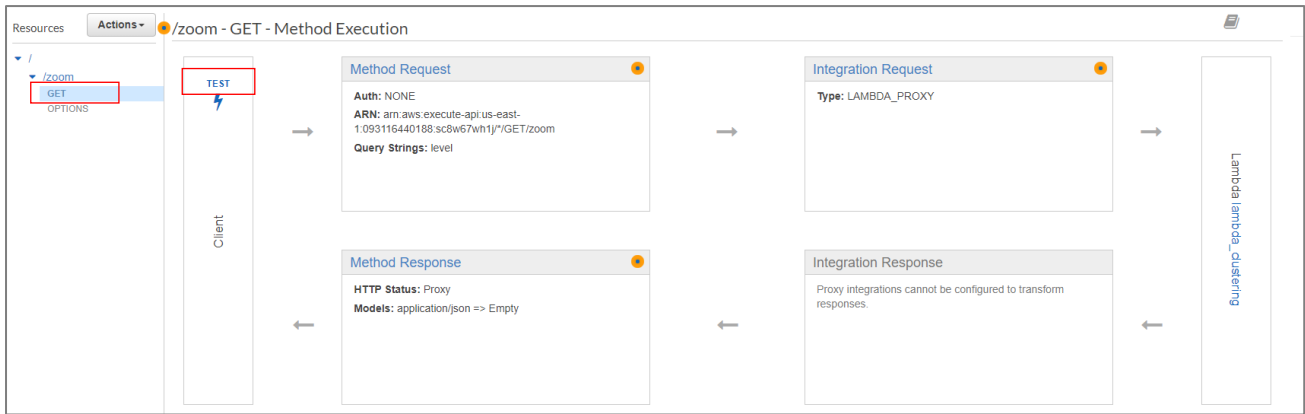
```

(\*) The `return` function above has the “headers” and “body” statements for working with the script in the html file, by giving some “Access permissions” and also the clusters in a JSON format.

(\*) Notice that function uses the `json.dumps()` method by taking the List of Dictionaries format and returning them into JSON format blocks.

(\*) Correction in the “headers” line: The right sentence for evading access control problems was this: 'Access-Control-Allow-Origin': '\*'. It's just a little mistake but I couldn't call the API from my file because of this.

4.3. But, also I have to test my lambda function via [API Gateway](#) to see if there's no problem calling it from there. Go to the API Gateway service and open [mapLambdaAPI](#). Press the [GET method](#) and then press [TEST action](#). In the next screenshot you can see it:



4.4. Give it a {zoom} parameter level = 5 and then press **Test** button, as you can see in the next screenshot:

(\*) Notice the right **response body and headers** from my lambda\_clustering, returning the size of the clusters and their locations and also the header's parameters.

The screenshot shows the 'Test' results for the /zoom GET method. The 'Query Strings' section has a red box around the {zoom} parameter with the value level=5. The 'Response Body' section has a red box around the JSON response:

```
[
  {
    "cluster_size": 4,
    "latitude": -22.84111,
    "longitude": -68.169106
  },
  {
    "cluster_size": 3,
    "latitude": -21.205897,
    "longitude": -69.80718
  },
  {
    "cluster_size": 2,
    "latitude": -18.769965,
    "longitude": -69.787169
  }
]
```

The 'Response Headers' section shows the following headers:

```
{
  "Access-Control-Allow-Origin": "/*",
  "X-Amzn-Trace-Id": "Root=1-5ec0f0c6-8dd5b975b1963ff6acaa19f9;Sampled=0"
}
```


The 'Logs' section shows the execution log for the request, indicating that the HTTP Method is GET and the Resource Path is /zoom.

4.5. Also, testing it through the browser Mozilla Firefox through the URL of the API service, shows the JSON format in a better way. The URL and the screenshot is shown next:

<https://sc8w67wh1j.execute-api.us-east-1.amazonaws.com/Test/zoom?level=6>

(\*) Notice that the parameters delivered in the PATH must be the right ones. Those are the zoom and level parameters.



JSON	Datos en bruto	Cabeceras
Guardar Copiar Contraer todo Expandir todo  Filtrar JSON		
▼ 0:		
cluster_size: 1		
latitude: -22.84111		
longitude: -68.169106		
▼ 1:		
cluster_size: 3		
latitude: -22.768051		
longitude: -68.478319		
▼ 2:		
cluster_size: 1		
latitude: -22.915622		
longitude: -70.290275		
▼ 3:		
cluster_size: 1		
latitude: -20.49157		
longitude: -69.329696		
▼ 4:		
cluster_size: 1		
latitude: -20.21015		
longitude: -69.801569		
▼ 5:		
cluster_size: 1		
latitude: -19.060498		
longitude: -69.250661		
▼ 6:		
cluster_size: 1		
latitude: -18.479432		
longitude: -70.323676		

## 5. Changing testMap.html to testMapv2.html file and implementing jQuery's AJAX() method in the script.

5.1. My `markers` container is an array, so I need a function for clear all the markers setted in a certain level of zoom, as you can see in the next screenshot:

```
var markers = [];  
  
function InitMap(){  
    <!--CLEARING MARKERS FUNCTION FOR USING ON EVERY ZOOM_CHANGED-->  
    function clearMarkers() {  
        for(var i = 0; i < markers.length; i++){  
            markers[i].setMap(null);  
        }  
        markers = [];  
    }  
}
```

(\*) I tried a lot for finding an implemented method in Google API to clear markers on the map, and these existed but now are [deprecated methods](#), so doesn't work on the actual version of the API.

5.2. I established the center of the map's view as the coordinates of my cluster 9 and also the first default marker at level 4 of zooming. The `markers.push()` method is for keeping the marker in the markers array, as you can see in the next screenshot:

```
<!--DEFAULT ZOOM LEVEL CHANGED TO 4 AND CENTER OF THE MAP WITH THE COORDINATES OF CLUSTER 9-->  
var map = new google.maps.Map(document.getElementById('map'), {  
    zoom: 4,  
    center: new google.maps.LatLng(-21.391303, -69.185811),  
    mapTypeId: google.maps.MapTypeId.ROADMAP  
});  
  
<!--DEFAULT CLUSTER 9 INITIALIZATION AND SETTING-->  
var marker;  
if(map.getZoom()===4){  
    clearMarkers();  
    marker = new google.maps.Marker({  
        position: new google.maps.LatLng(-21.391303, -69.185811),  
        map: map,  
        icon: 'https://chart.googleapis.com/chart?chst=d_map_pin_letter&chld=9|FF0000|000000'  
    });  
    markers.push(marker);  
}
```

5.3. Now, I did all the actualization of the markers in the ajax query. First I was working on the idea that I could set the default cluster 9 by doing another ajax query in the beginning of the `InitMap()` function (the idea was not to put the default cluster directly by hand on the script), but I couldn't find the right way for doing that. I think it's because the ajax query will call the API in every `InitMap()` adjustment. The markers are setted on the `zoom_changed` listener inside of his own conditional block, as you see in the next screenshot:

```

map.addListener('zoom_changed', function(){
$.ajax({
url: 'https://sc8w67wh1j.execute-api.us-east-1.amazonaws.com/Test/zoom?level='+ (map.getZoom()),
headers: {'Content-Type': 'application/json'},
type: 'GET',
crossDomain: true,
dataType: 'json',
success: function(data) {
clusters = JSON.parse(JSON.stringify(data));
<!-- CLUSTER DEFINED AS A DICT FOR LEVEL 4 AND A LIST OF DICTS FOR LEVELS 5-6-7+-->
if(map.getZoom()<=4){
clearMarkers();
marker = new google.maps.Marker({
position: new google.maps.LatLng(-21.391303, -69.185811),
map: map,
icon: 'https://chart.googleapis.com/chart?chst=d_map_pin_letter&chld=9|FF0000|000000'
});
markers.push(marker);
}
else if(map.getZoom()==5){
clearMarkers();
for(i=0; i < clusters.length; i++){
marker = new google.maps.Marker({
position: new google.maps.LatLng(clusters[i]["latitude"], clusters[i]["longitude"]),
map: map,
icon: 'https://chart.googleapis.com/chart?chst=d_map_pin_letter&chld=' + (clusters[i]["cluster_size"]) + '|FF0000|000000'
});
markers.push(marker);
}
}
else if(map.getZoom()==6){
clearMarkers();
for(i=0; i < clusters.length; i++){
marker = new google.maps.Marker({
position: new google.maps.LatLng(clusters[i]["latitude"], clusters[i]["longitude"]),
map: map,
icon: 'https://chart.googleapis.com/chart?chst=d_map_pin_letter&chld=' + (clusters[i]["cluster_size"]) + '|FF0000|000000'
});
markers.push(marker);
}
}
}
});

```

(\*) Here there is something important I have to mention. I was stucked for a moment doing some research to clarify the use of the `JSON.stringify()` and `JSON.parse()`. I didn't understand so well the right use of the both of them, but I realized that the cluster's data received from the lambda function in the JSON format is received as a string. I tried a lot of ways for get this done, to receive the data structure and pass it to my clusters structure. Then, my solution was to use both of them as a "nested" function as I show you here:

```
clusters = JSON.parse(JSON.stringify(data));
```

(\*) So, the data comes in a JSON object format (by the `json.dumps` in lambda), then the `JSON.stringify()` function convert this JSON object into an string representation, and finally the `JSON.parse()` does the final work of passing the formatted data to my structure. I'm not clear enough why it's not sufficient passing `JSON.stringify` directly to my cluster structure.

These are my sources that helped me to complete this part:

<https://stackoverflow.com/questions/17785592/difference-between-json-stringify-and-json-parse>

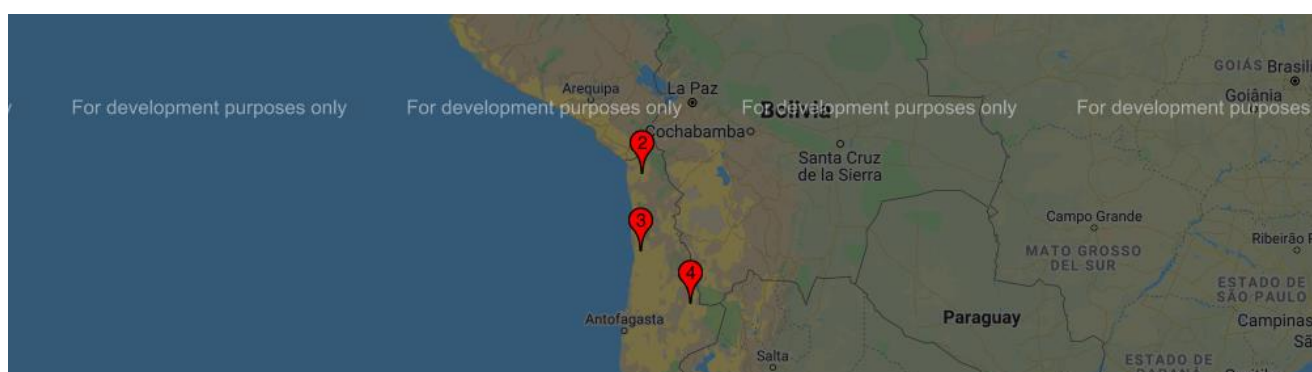
<https://pietschsoft.com/post/2015/09/05/javascript-basics-how-to-create-a-dictionary-with-keyvalue-pairs>

## 6. Testing locally.

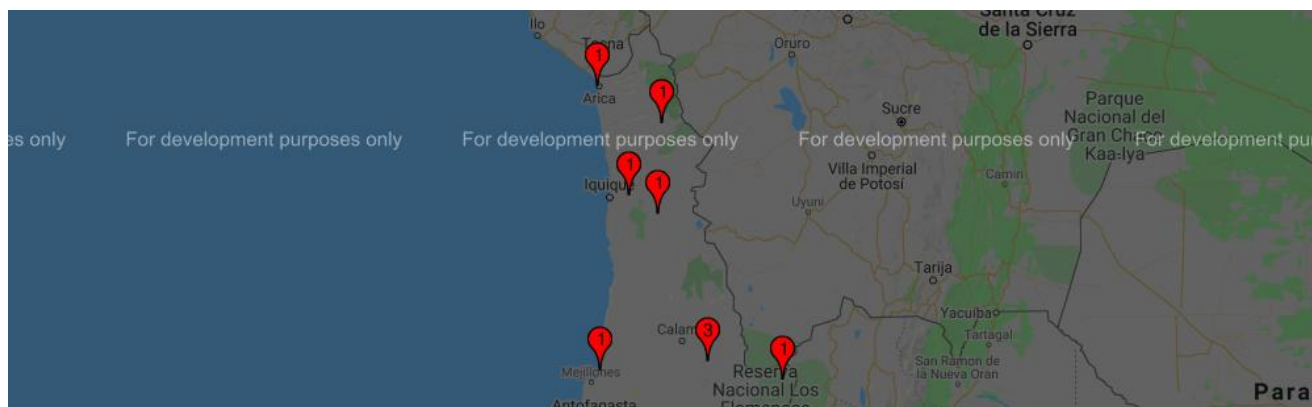
### 6.1. Zoom level: 4



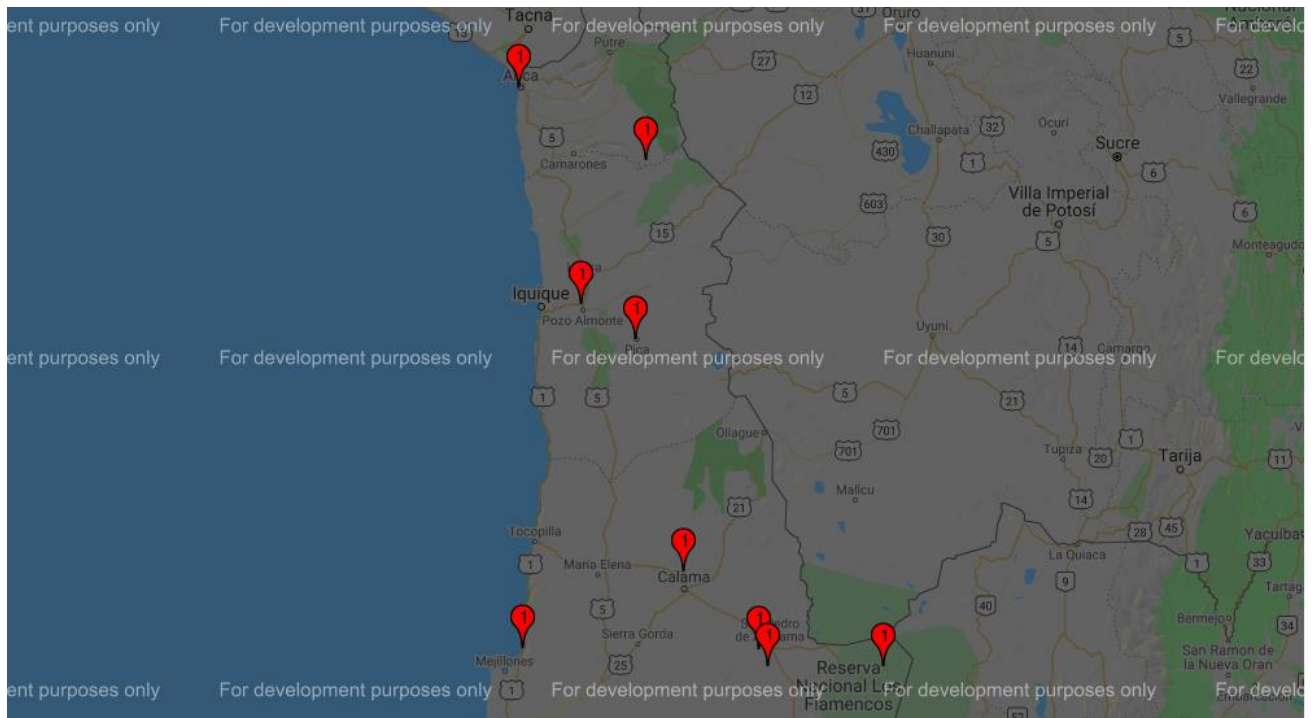
### 6.2. Zoom level 5:



### 6.3. Zoom level 6:

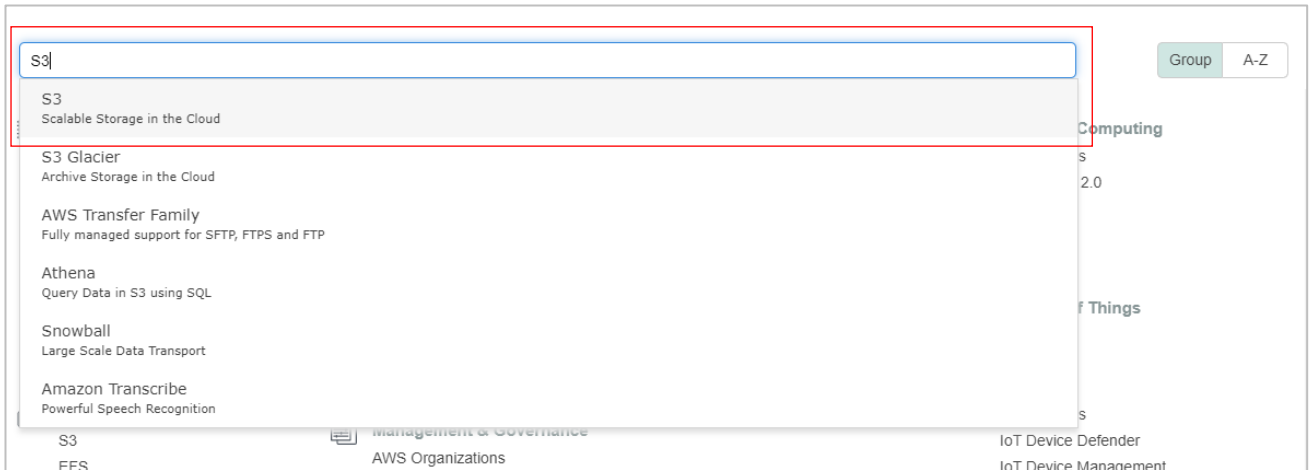


Zoom level 7:

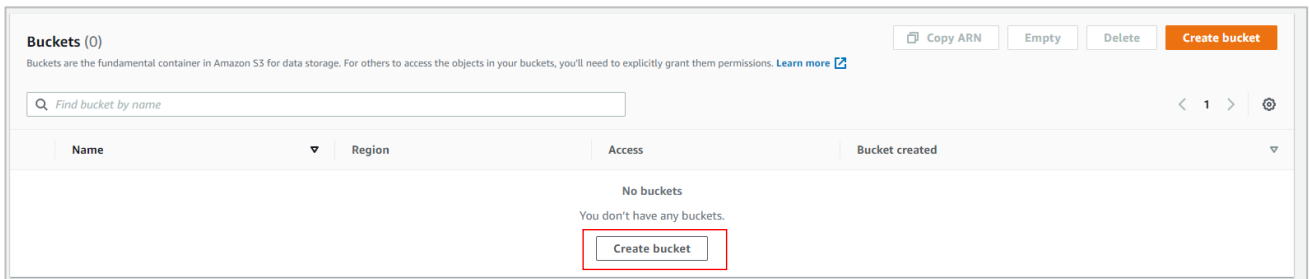


## 7. Deployment with S3 Bucket Service.

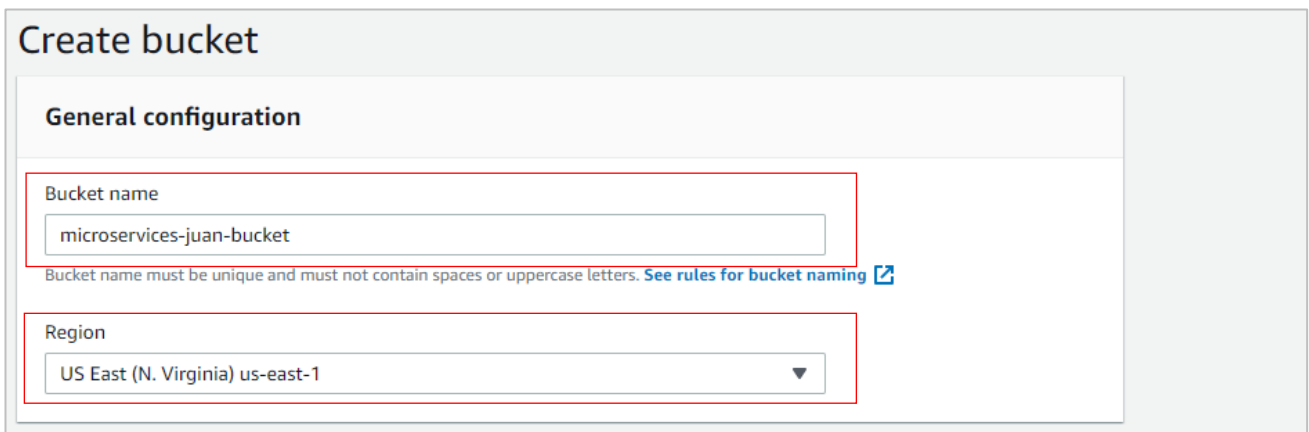
7.1. Search the S3 service in the AWS console, as you can see in the next screenshot:



7.2. Then I have to create a bucket pressing [Create bucket](#) button, as you can see in the next screenshot:




7.3. Give it a name (mine is [microservices-juan-bucket](#)) and keep the same region that I've been working on through all the services, as you can see in the next screenshot:



7.4. I have disabled all the “[block public access](#)” options in the advanced apart, as you can see in the next screenshot:

## Bucket settings for Block Public Access

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#) 

### ☐ Block *all* public access

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

#### ☐ Block public access to buckets and objects granted through *new* access control lists (ACLs)

S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

#### ☐ Block public access to buckets and objects granted through *any* access control lists (ACLs)

S3 will ignore all ACLs that grant public access to buckets and objects.

#### ☐ Block public access to buckets and objects granted through *new* public bucket or access point policies

S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

#### ☐ Block public and cross-account access to buckets and objects through *any* public bucket or access point policies

S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.



**Turning off block all public access might result in this bucket and the objects within becoming public**  
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

☒ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

7.5. Then just pressed **Create bucket** button, as you can see in the next screenshot:



**Turning off block all public access might result in this bucket and the objects within becoming public**  
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

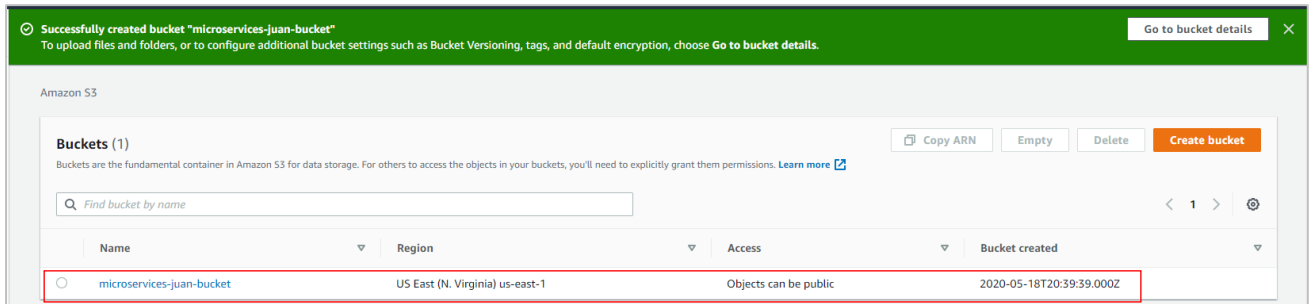
☒ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

► **Advanced settings**

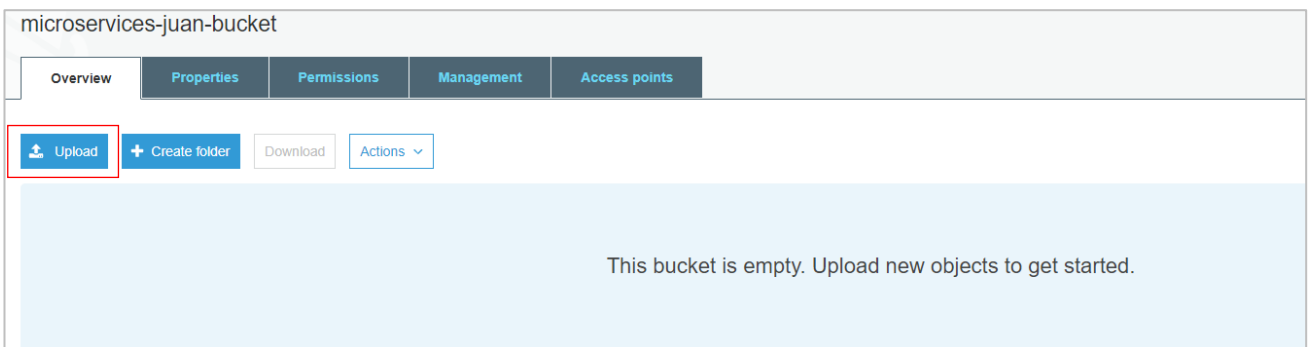
Cancel

**Create bucket**

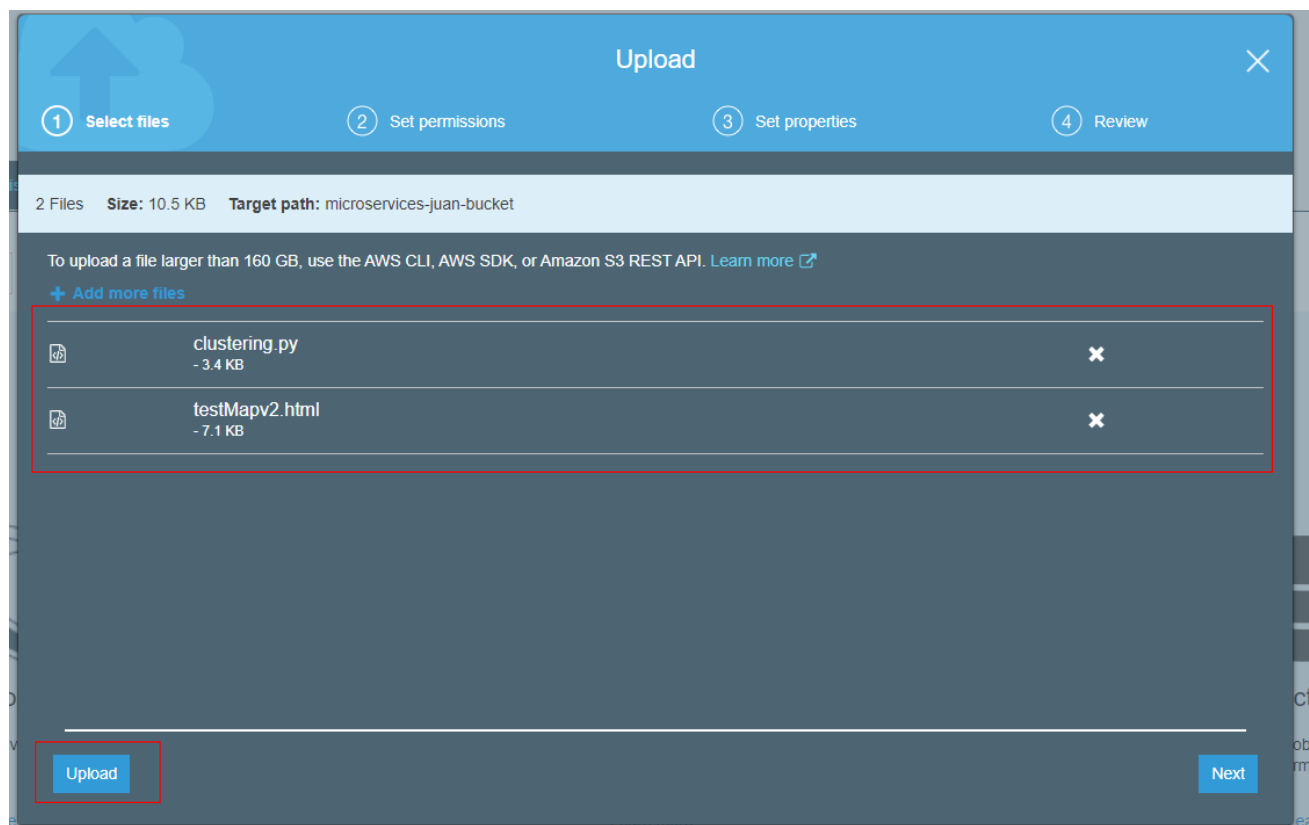
7.6. The bucket was created successfully, showing it in the buckets list, as you can see in the next screenshot:



7.7. I'm going to upload my files into the service for deploying it, so click in the bucket and press the **Upload** button, as you can see in the next screenshot:



7.8. Just add the files from your computer and then **Upload**, as you can see in the next screenshot:





7.9. Then, the files must be visible in the bucket files list, as you can see in the next screenshot:

microservices-juan-bucket

Overview Properties Permissions Management Access points

Q Type a prefix and press Enter to search. Press ESC to clear.

Upload Create folder Download Actions

US East (N. Virginia)

Name	Last modified	Size	Storage class
clustering.py	May 18, 2020 4:44:24 PM GMT-0400	3.4 KB	Standard
testMapv2.html	May 18, 2020 4:44:24 PM GMT-0400	7.1 KB	Standard

Viewing 1 to 2

7.10 Go to **properties** tab in the upper side of your bucket, and it shows you the properties of your bucket, as you can see in the next screenshot:

Versioning

Keep multiple versions of an object in the same bucket.

Learn more

Disabled

Server access logging

Set up access log records that provide details about access requests.

Learn more

Disabled

Static website hosting

Host a static website, which does not require server-side technologies.

Learn more

Disabled

Object-level logging

Record object-level API activity using the CloudTrail data events feature (additional cost).

Learn more

Disabled

Default encryption

Automatically encrypt objects when stored in Amazon S3.

Learn more

Disabled

7.11 Press the **Static website hosting** title card, and use your main.html file as the index document. Mine is **testMapv2.html**. Then, just press **Save** button, as you can see in the next screenshot:

Static website hosting

Endpoint : <http://microservices-juan-bucket.s3-website-us-east-1.amazonaws.com>

☒ Use this bucket to host a website [Learn more](#)

Index document [i](#)

testMapv2.html

Error document [i](#)

error.html

Redirection rules (optional) [i](#)

☐ Redirect requests [Learn more](#)

☐ Disable website hosting

Disabled

Cancel Save

7.12 Finally, I have my S3 bucket created and my [testMapv2.html](https://microservices-juan-bucket.s3.amazonaws.com/testMapv2.html) file deployed on this service. The URL for calling my html file it's the next: (The first is the clustered file, and the second just with the locations)

<https://microservices-juan-bucket.s3.amazonaws.com/testMapv2.html>  
<https://microservices-juan-bucket.s3.amazonaws.com/testMap.html>