Documentation

Microservices 4th - 5th Week

Microservices Architecture (Introduction)

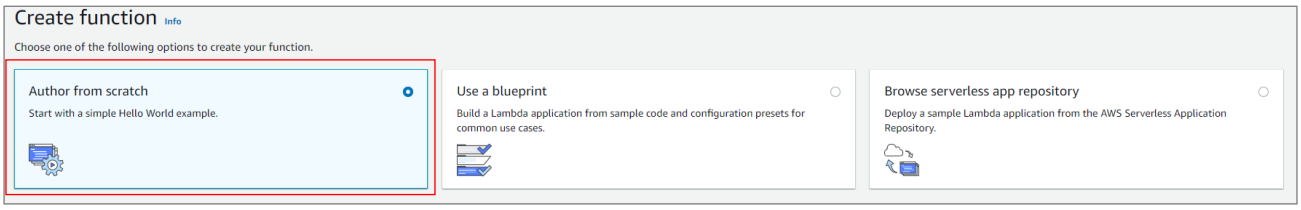
Teacher: Mr. Zheng Li

Student: Juan Albornoz

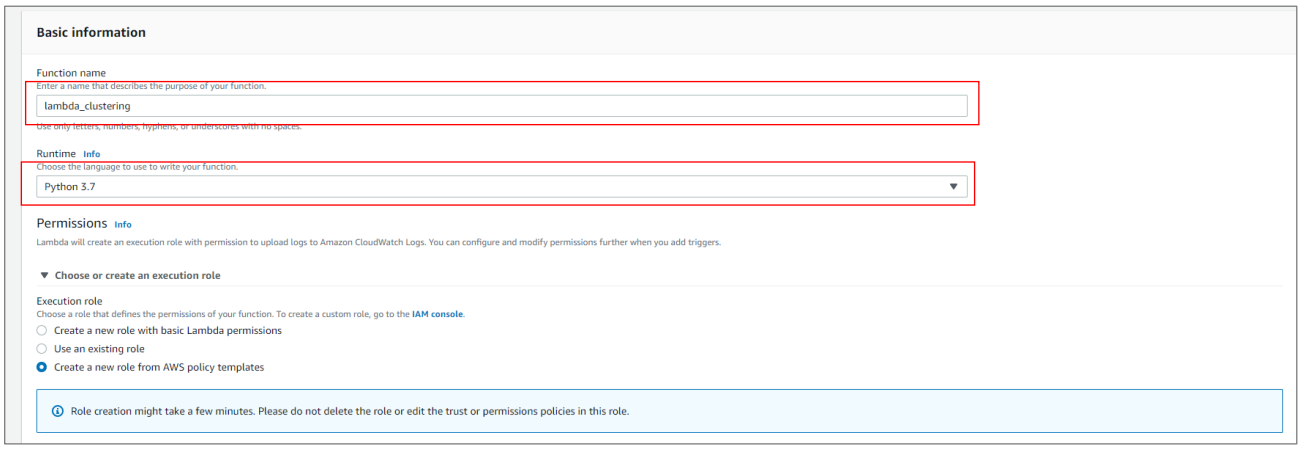
Index

1. Setting up the Lambda function.

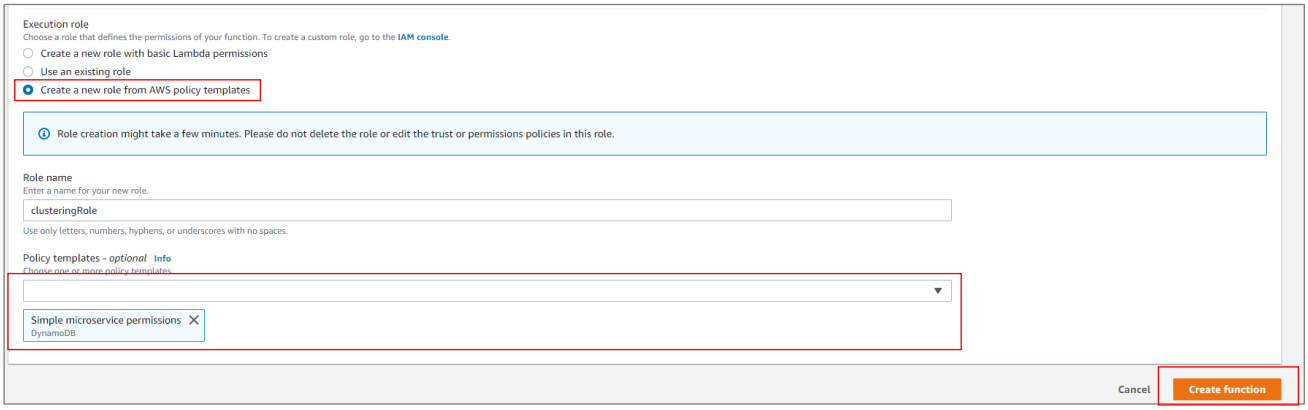
1.1. Create a function with the Autor from scratch option, as you see in the next screenshot:



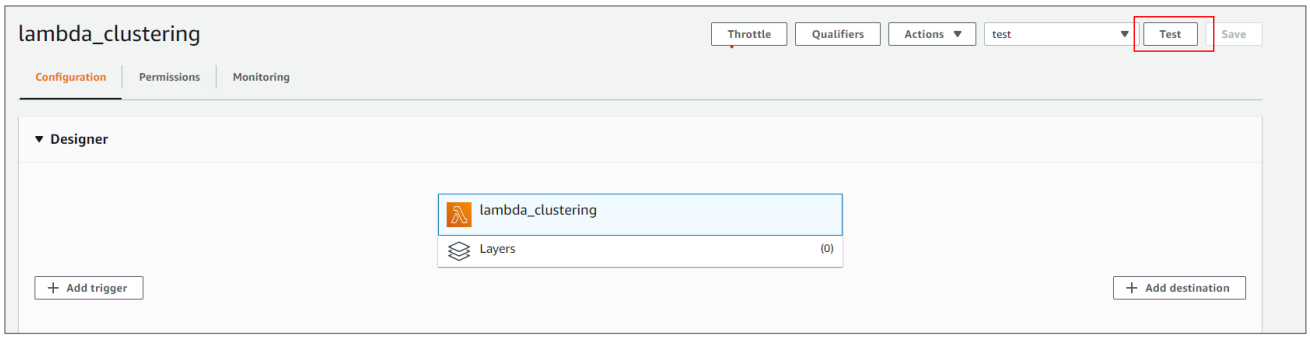
1.2. My function's name is lambda_clustering
1.3. The runtime language it´s Python 3.7, as you see in the nextscreenshot:



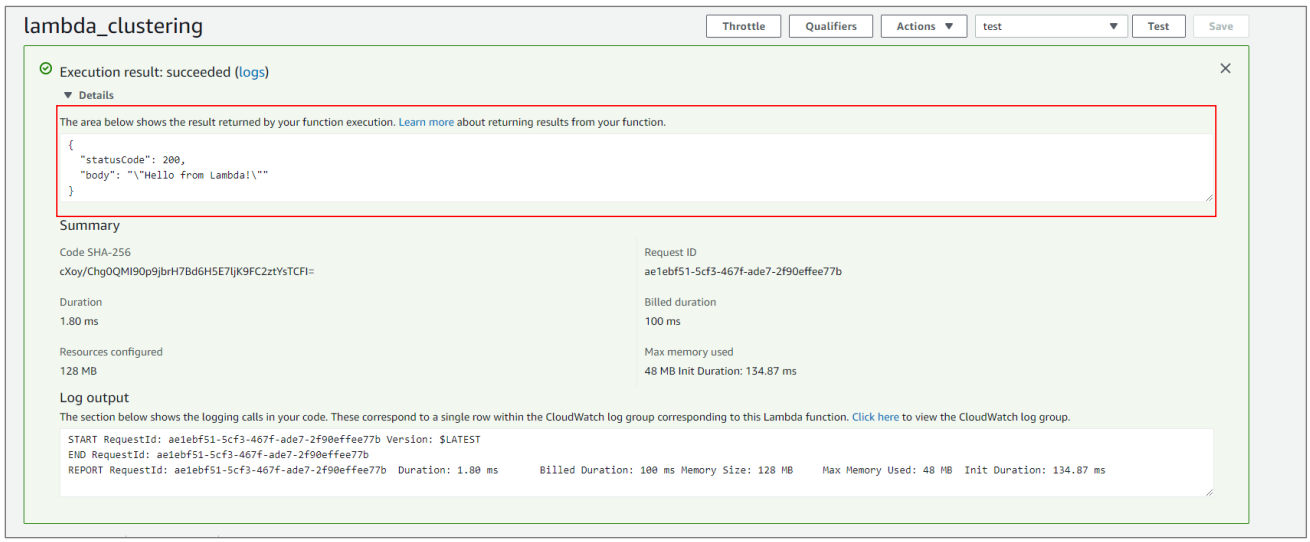
1.4. Create a new role from AWS templates, the parameters are the following:
1.4.1.The role name is clusteringRole
1.4.2.The policy template is Simple microservice permissions, as you see in the next screenshot:



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:



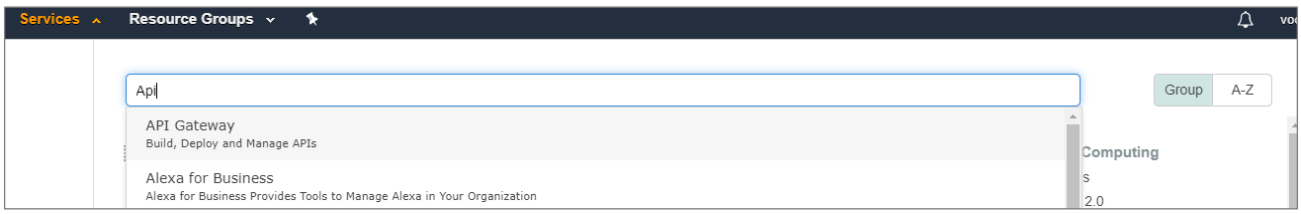
1.6. Then, the results of executing it successfully and his logs, as you see in the next screenshot:



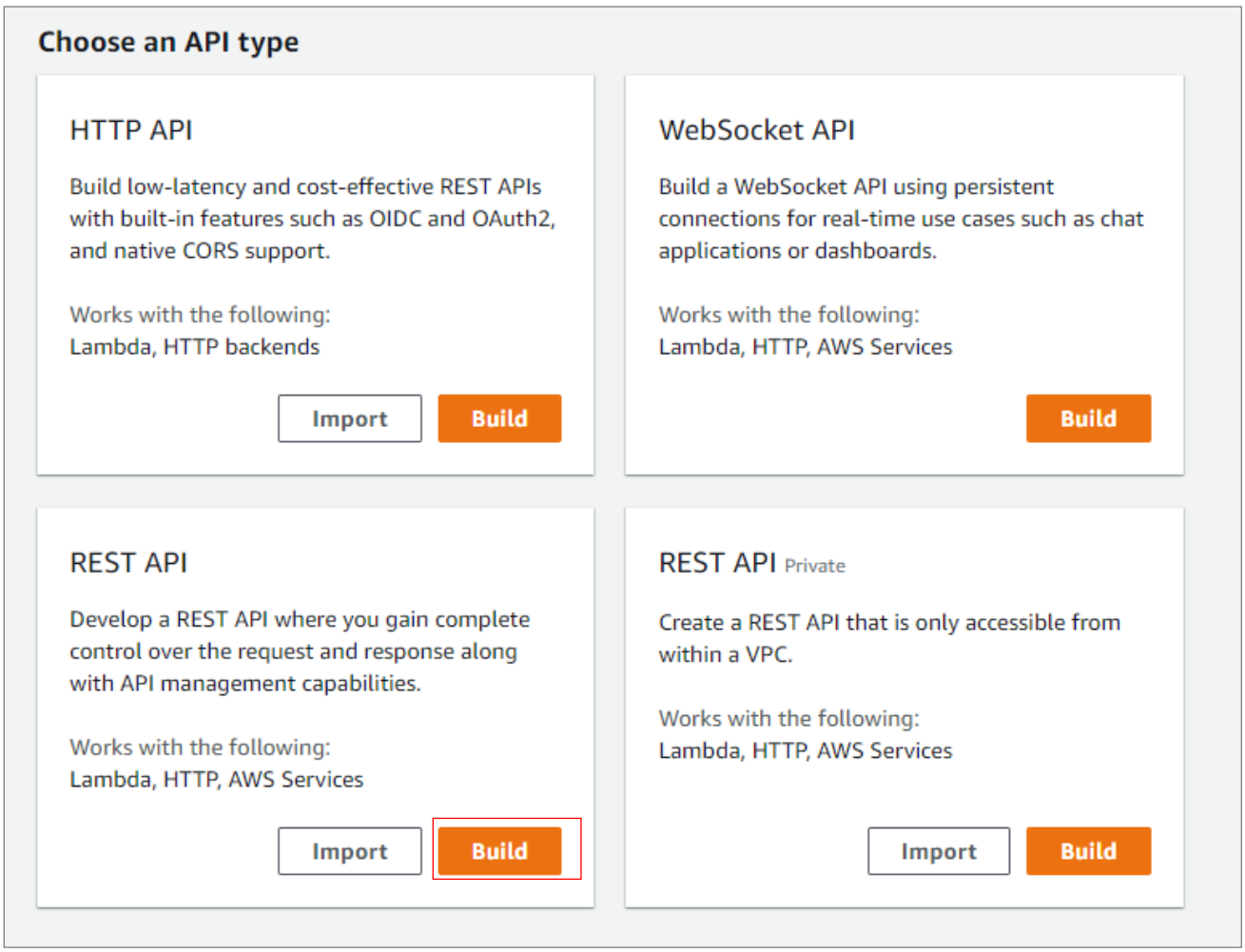
(*) I leave it here for now setting up my Lambda function, so now it's time for the API Gateway service. In a later section I'm going to go back to my Lambda function more deeply.

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

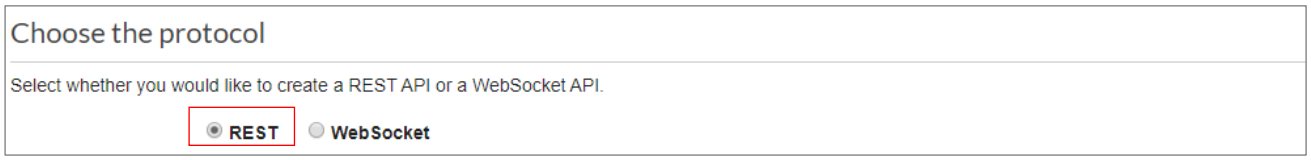
2.1. Search and open the API Gateway service, as you see in the next screenshot:



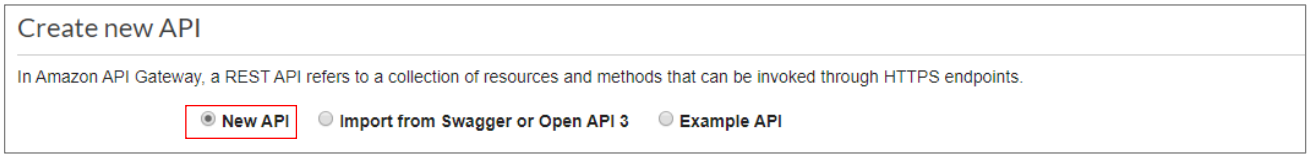
2.2. Choose REST API section and go to Build one, as you see in the next screenshot:



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:



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:

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

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:

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:

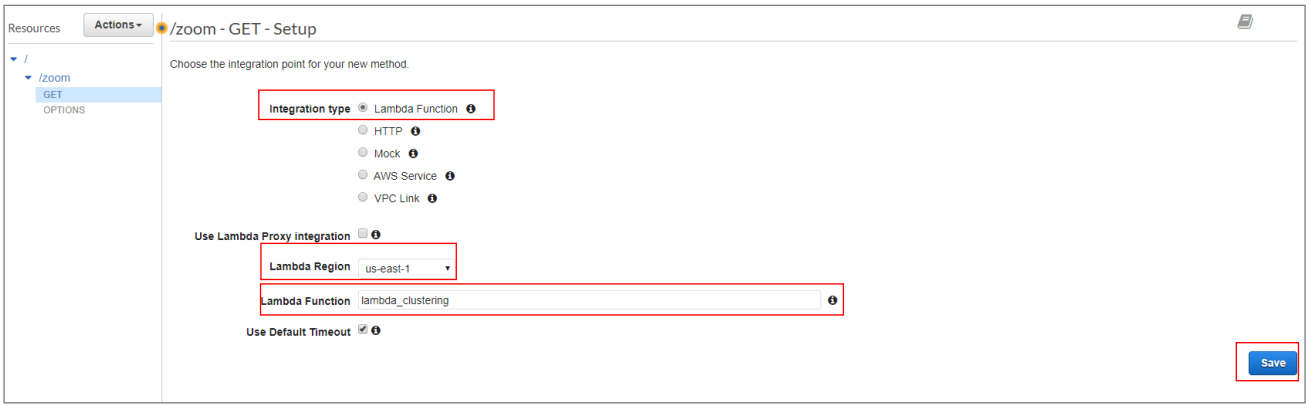
2.12. And then press the tick button, as you see in the next screenshot:

2.13. Choose Lambda Function as the Integration type

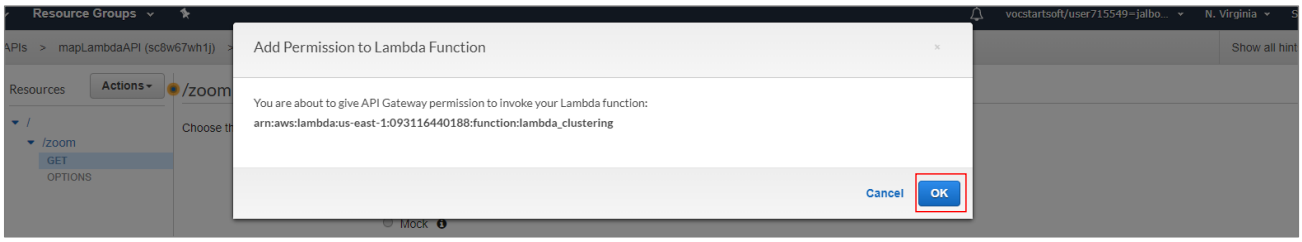
2.14. Put your Lambda Region as your region were are you working in, my region is us-east-1.

2.15. Search your Lambda Function, my function is lambda_clustering.

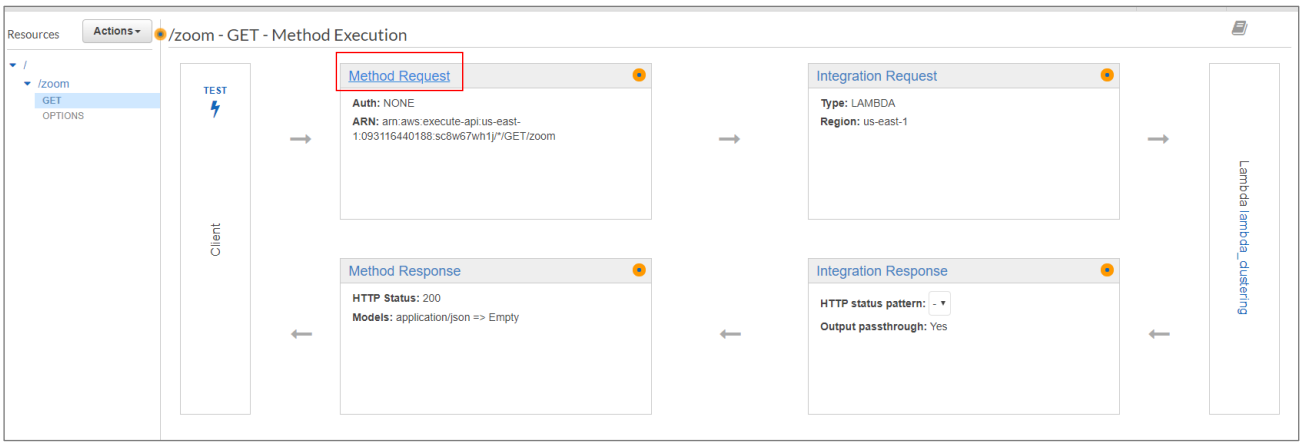
2.16. Then press the Save button, as you see in the next screenshot:



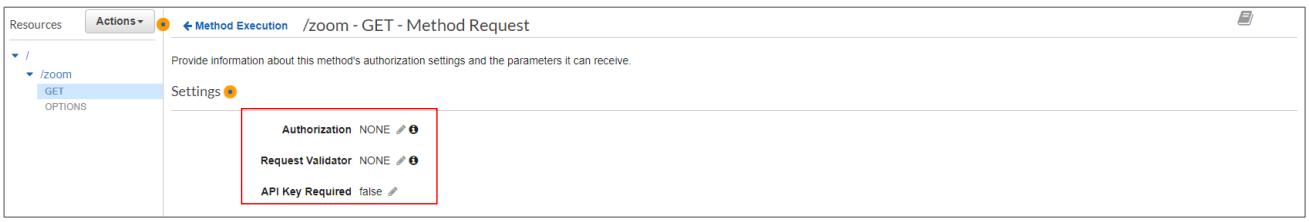
2.17. Then you must Add permissions to your Lambda Function by pressing OK button, as you see in the next screenshot:



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



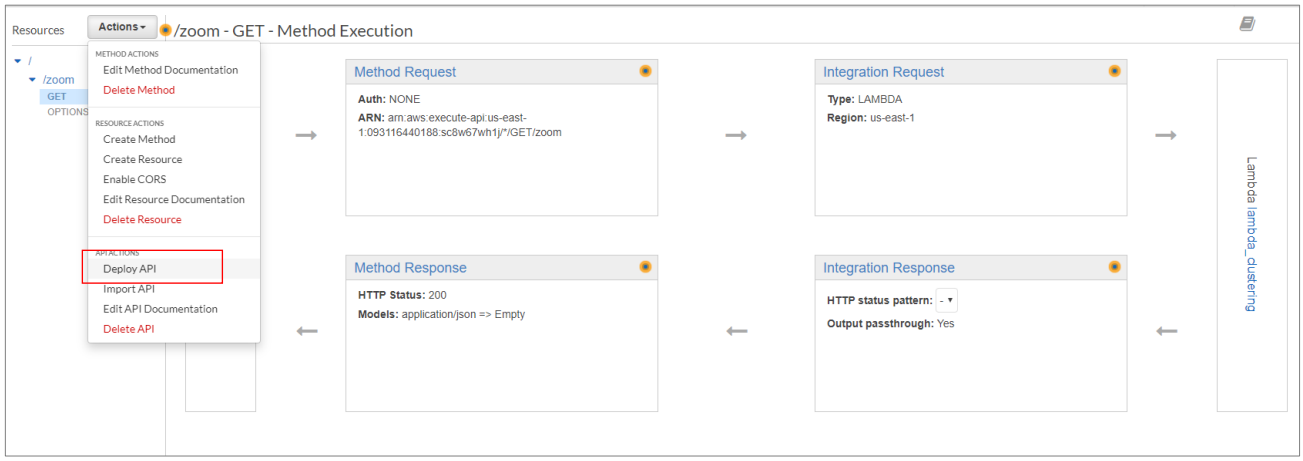
2.19. Leave the Authorization in NONE, also the Request Validator in NONE and the API Key Required in false, as you see in the next screenshot:



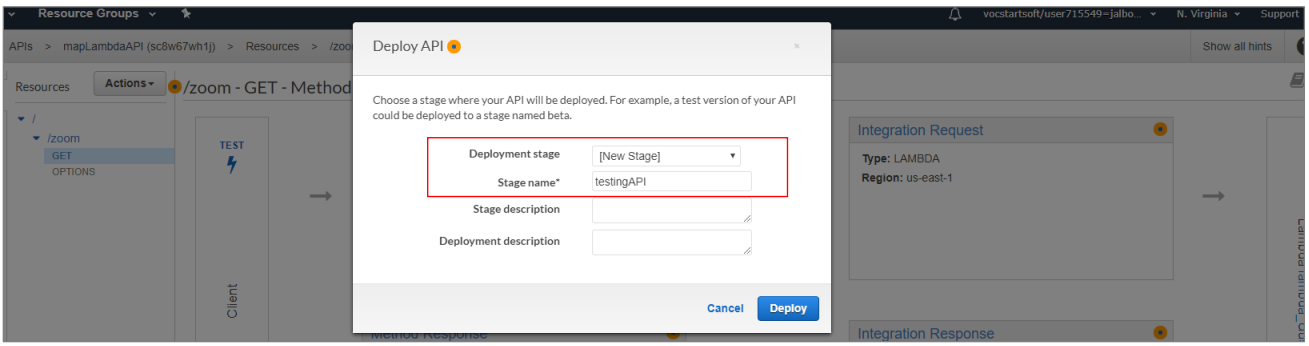
2.20. Go to URL Query String Parameters section and then Add query string, as you see in the next screenshot:

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:

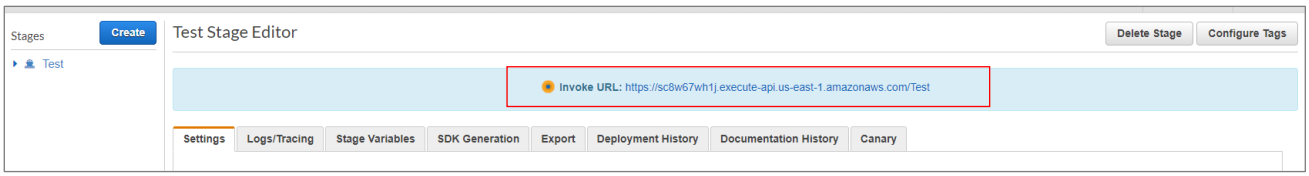
2.22. Now I have my API Gateway created and ready for testing it, but first I'm going to Deploy it by giving the Deploy API option in the Actions section, as you see in the next screenshot:



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:



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],
    ];
```

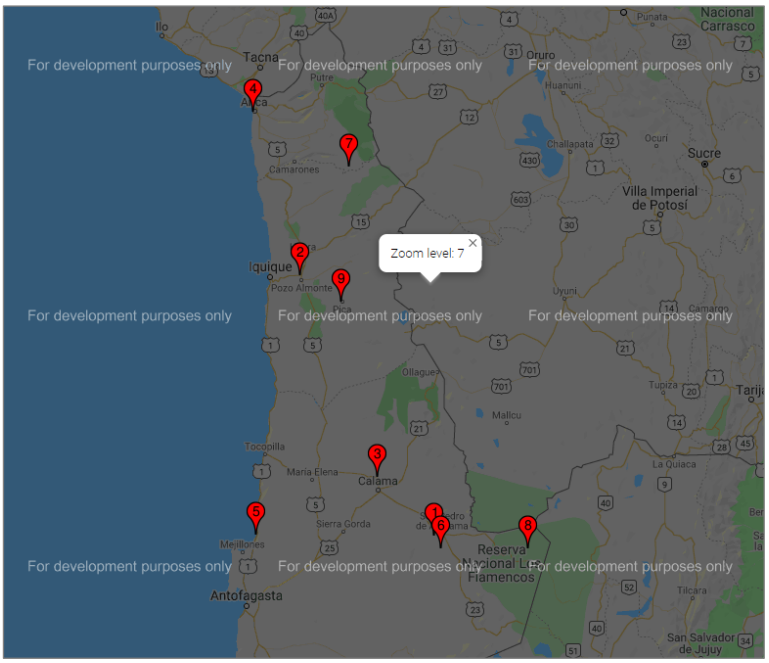
3.2. In the first part of the InitMap() function I changed the default zoom parameter and also the default center parameter of the screen to a new one calculated by hand. This following the next rule:

((x1 + …+ x9)/9, (y1+…+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:



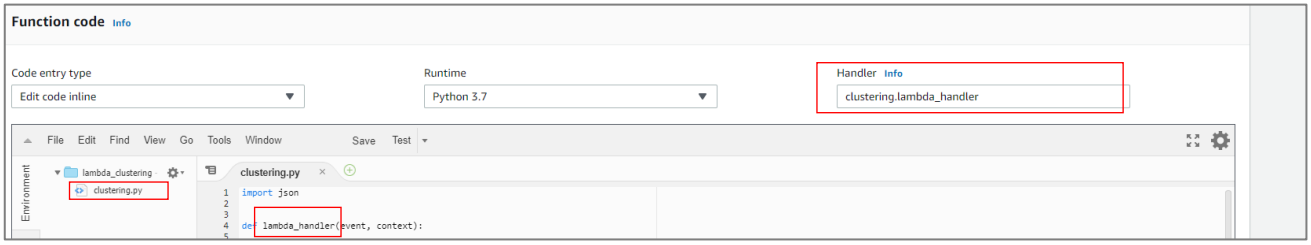
3.5. In the second part of the InitMap() function there is the URL for calling my deployed API. Notice that there are parameters for getting this in the right way. These parameters are zoom and level (resource and parameter) just established in the previous section. The next screenshot shows the AJAX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

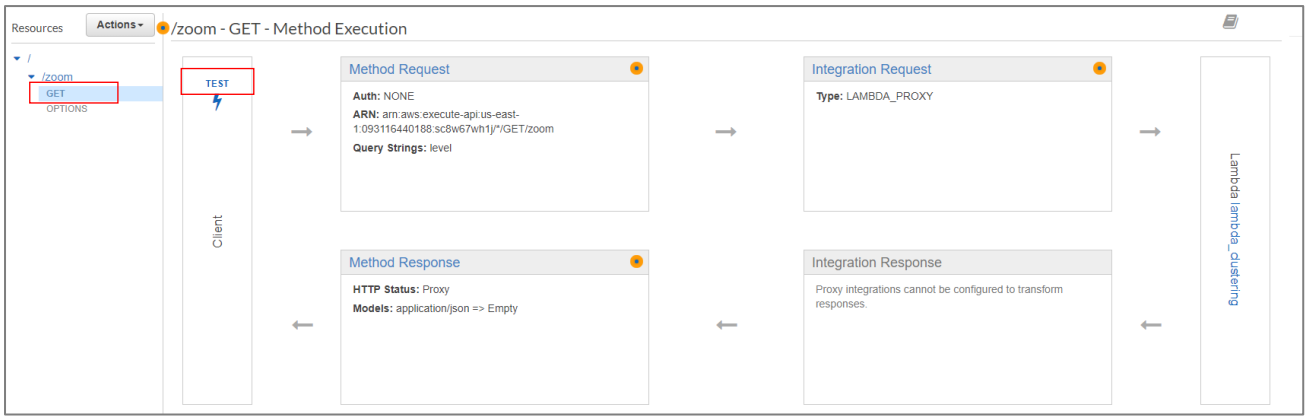
```
else:
    clusters = [
        {
            'cluster_size': 1,
            "latitude": -22.84111,
            "longitude": -68.169106
        },
        {
            'cluster_size': 1,
            "latitude": -22.924155,
            "longitude": -68.288103
        },
        {
            'cluster_size': 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.

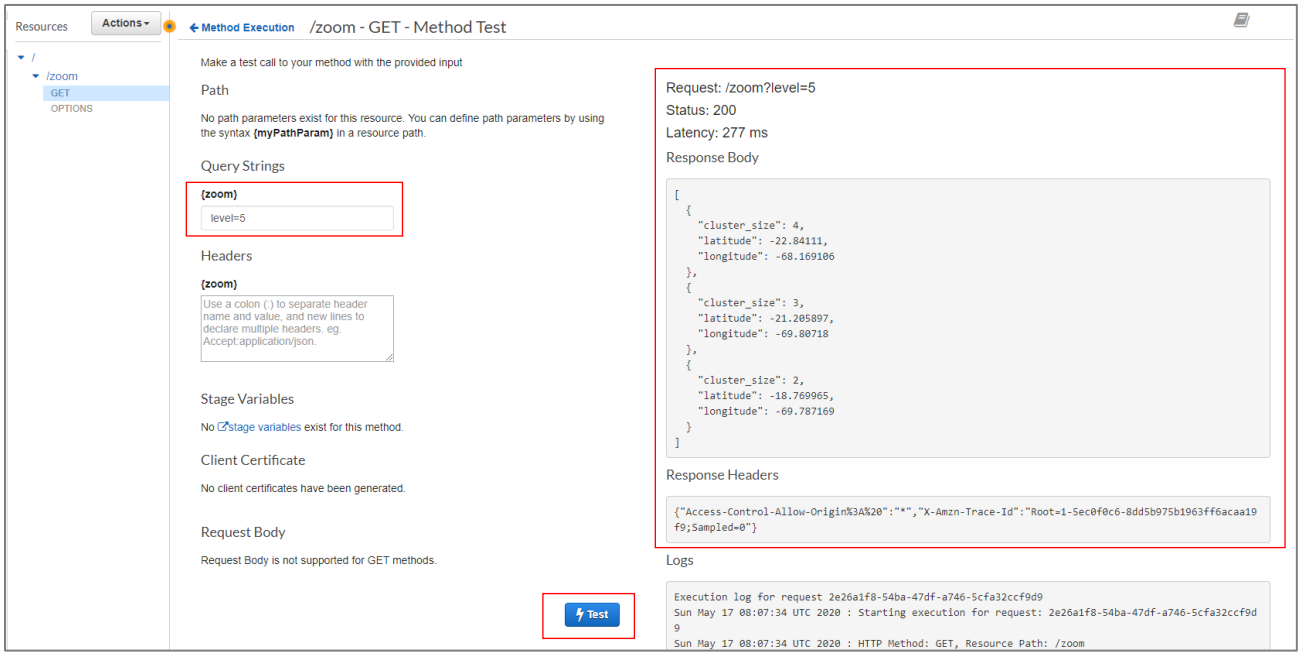
(*) Correction in the "headers" line: The right sentence for evading access control problems was this: 'Access-Control-Allow-Origin': '*'. It's just a little mistake but I couldn't call the API from my file because of this.

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.



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 if 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() adjustement. 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

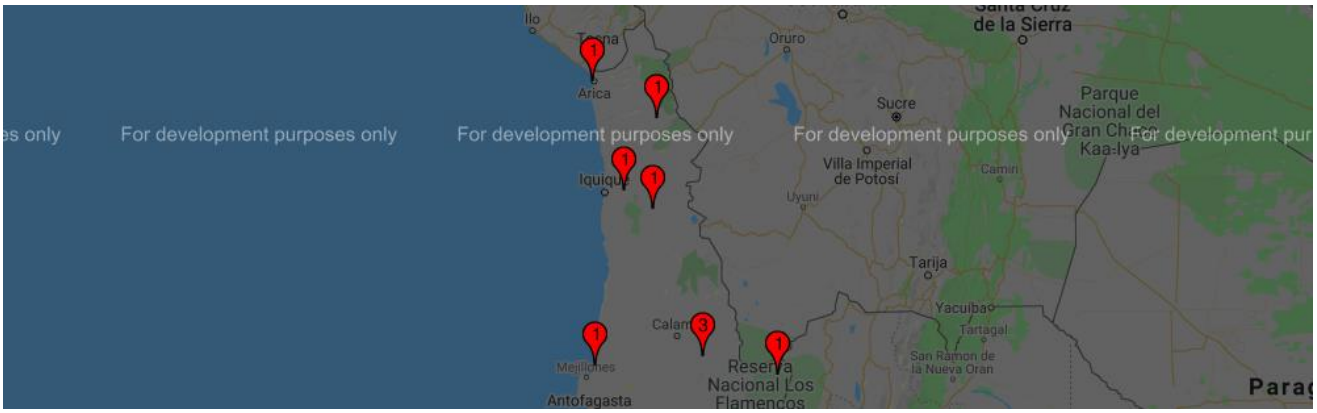
https://pietschsoft.com/post/2015/09/05/javascript-basics-how-to-create-a-dictionary-with-keyvalue-pairs

6. Testing locally.

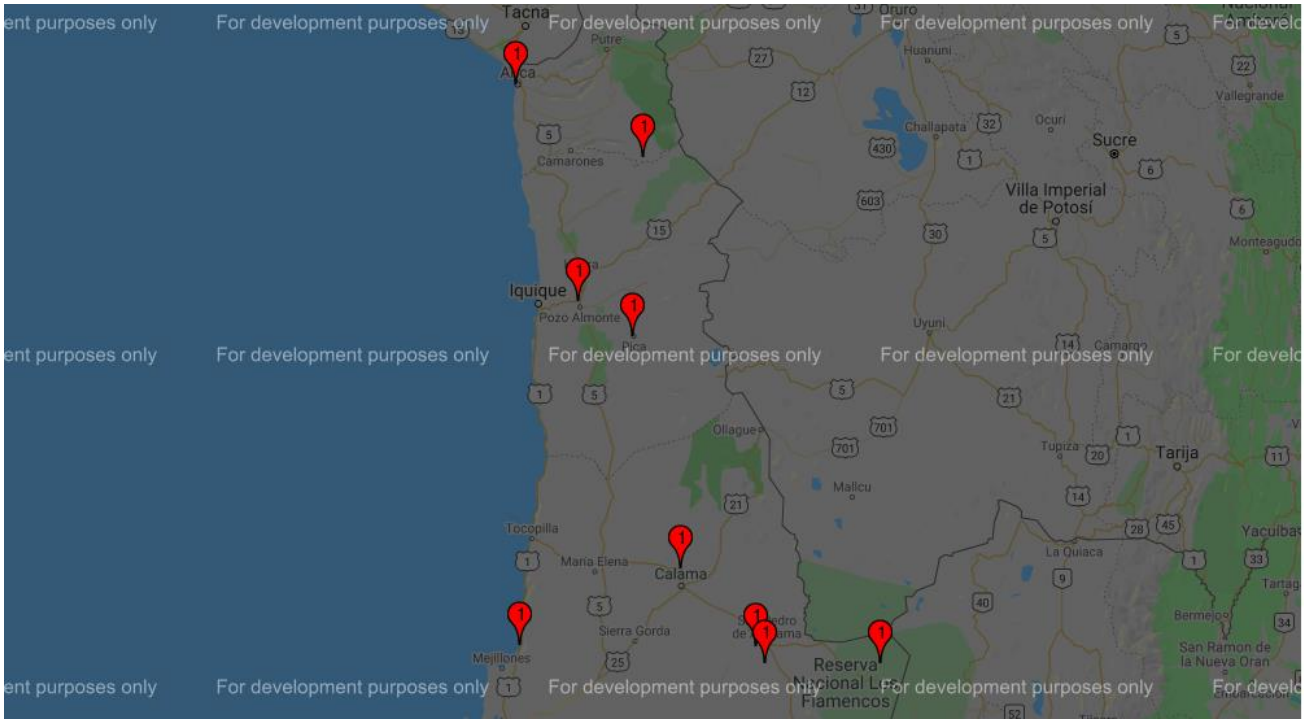
6.1. Zoom level: 4

6.2. Zoom level 5:

6.3. Zoom level 6:

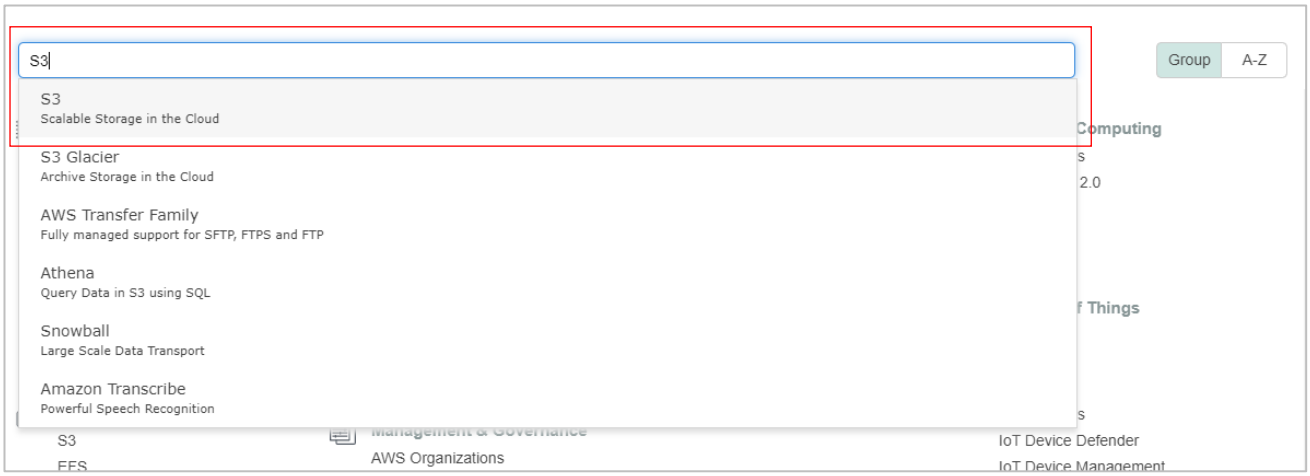


Zoom level 7:

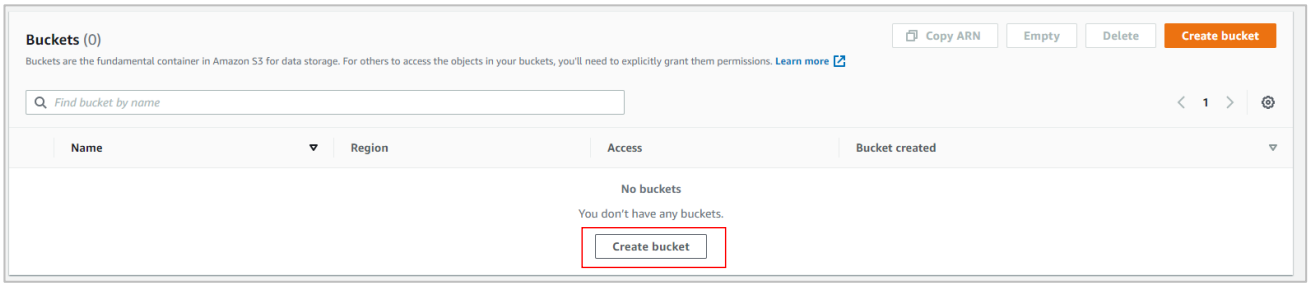

For development purposes only
Tacna
Arica
Putre
Camarones
Huanuni
Challapata
Ocuri
Sucre
Villa Imperial
de Potosí
Vallegrande
Iquique
Pozo Almonte
Uyuni
Ollague
Mallcu
Tupiza
Tarija
Tocopilla
María Elena
Calama
La Quiaca
Sierra Gorda
Mejillones
Bermejo
San Ramón de
la Nueva Orán
Reserva
Nacional Los
Flamencos

7. Deployment with S3 Bucket Service.

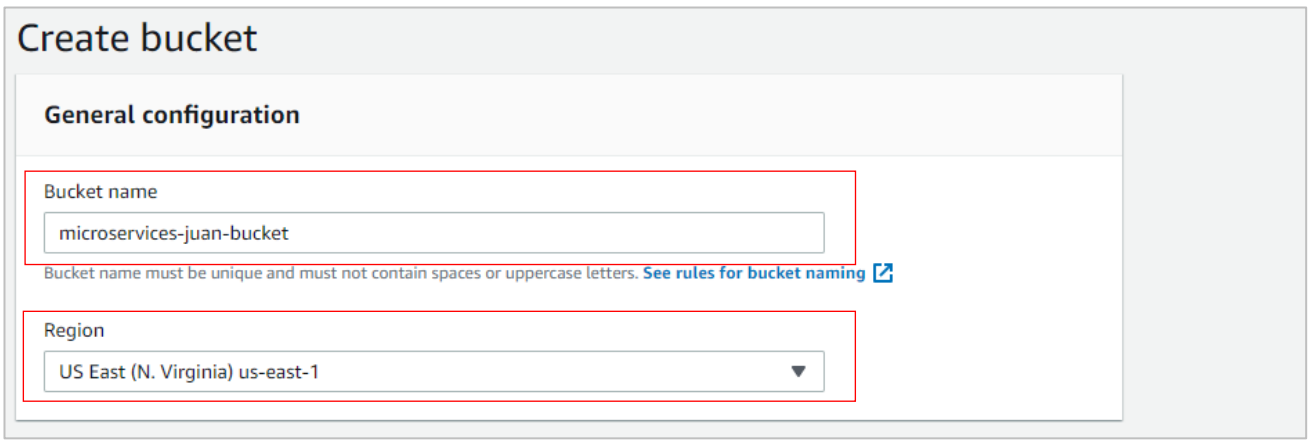
7.1. Search the S3 service in the AWS console, as you can see in the next screenshot:



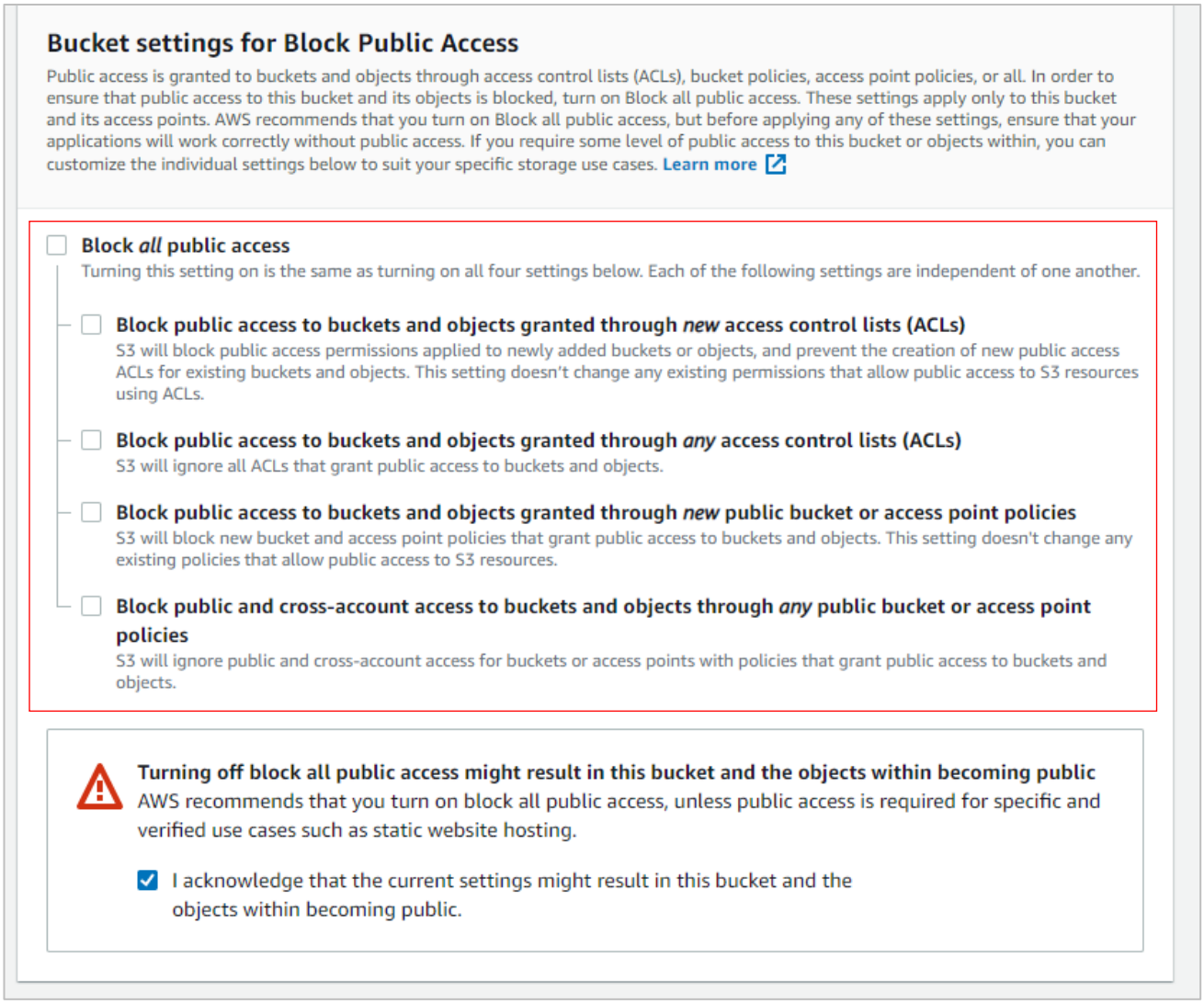
7.2. Then I have to create a bucket pressing Create bucket button, as you can see in the next screenshot:



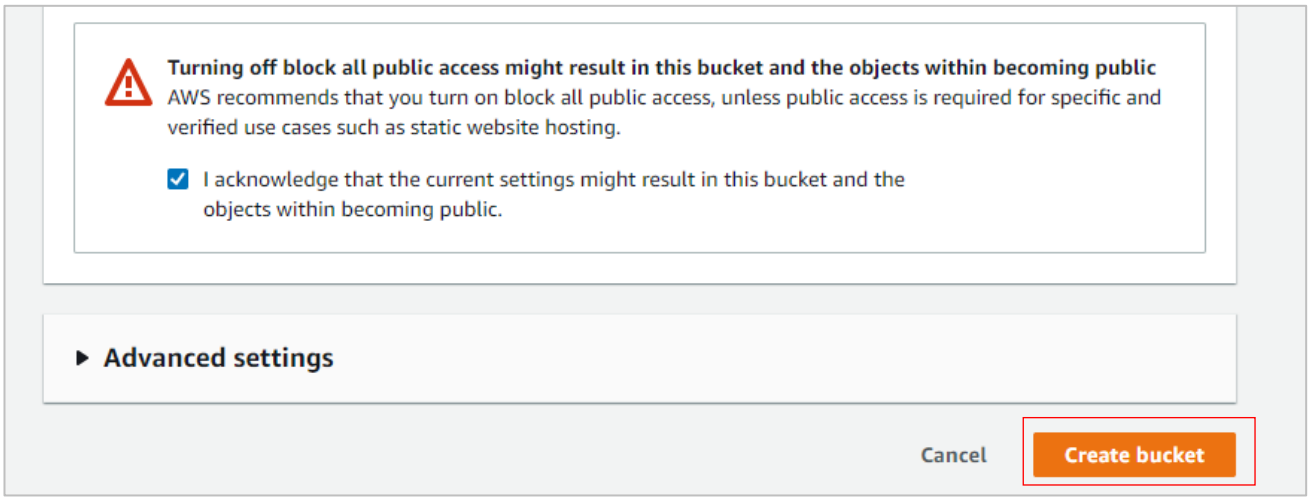
7.3. Give it a name (mine is microservices-juan-bucket) and keep the same region that I've been working on through all the services, as you can see in the next screenshot:



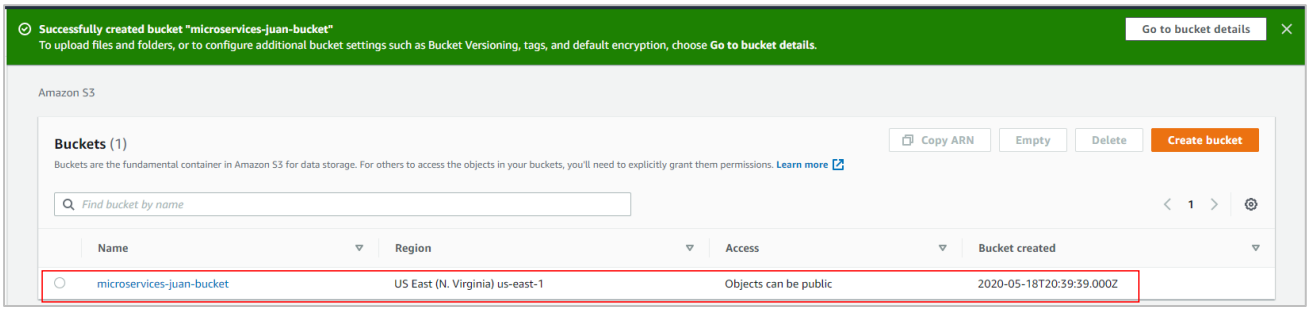
7.4. I have disabled all the "block public access" options in the advanced apart, as you can see in the next screenshot:



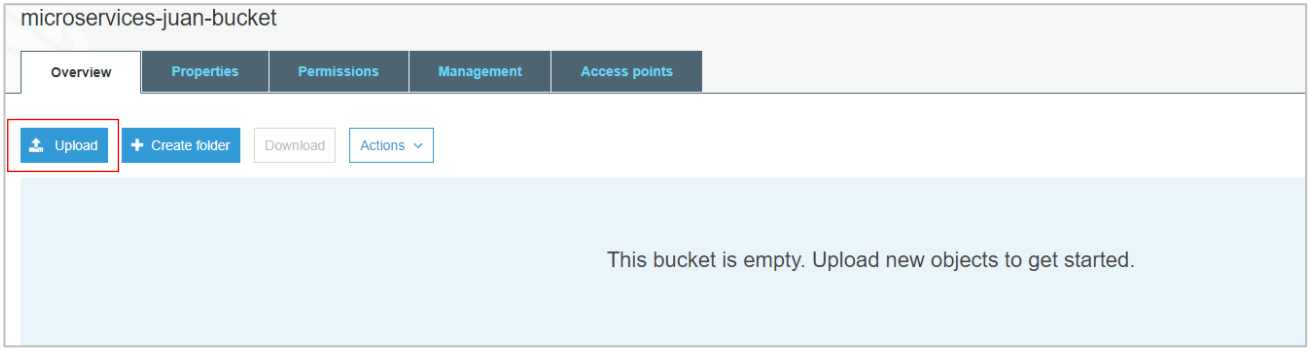
7.5. Then just pressed Create bucket button, as you can see in the next screenshot:



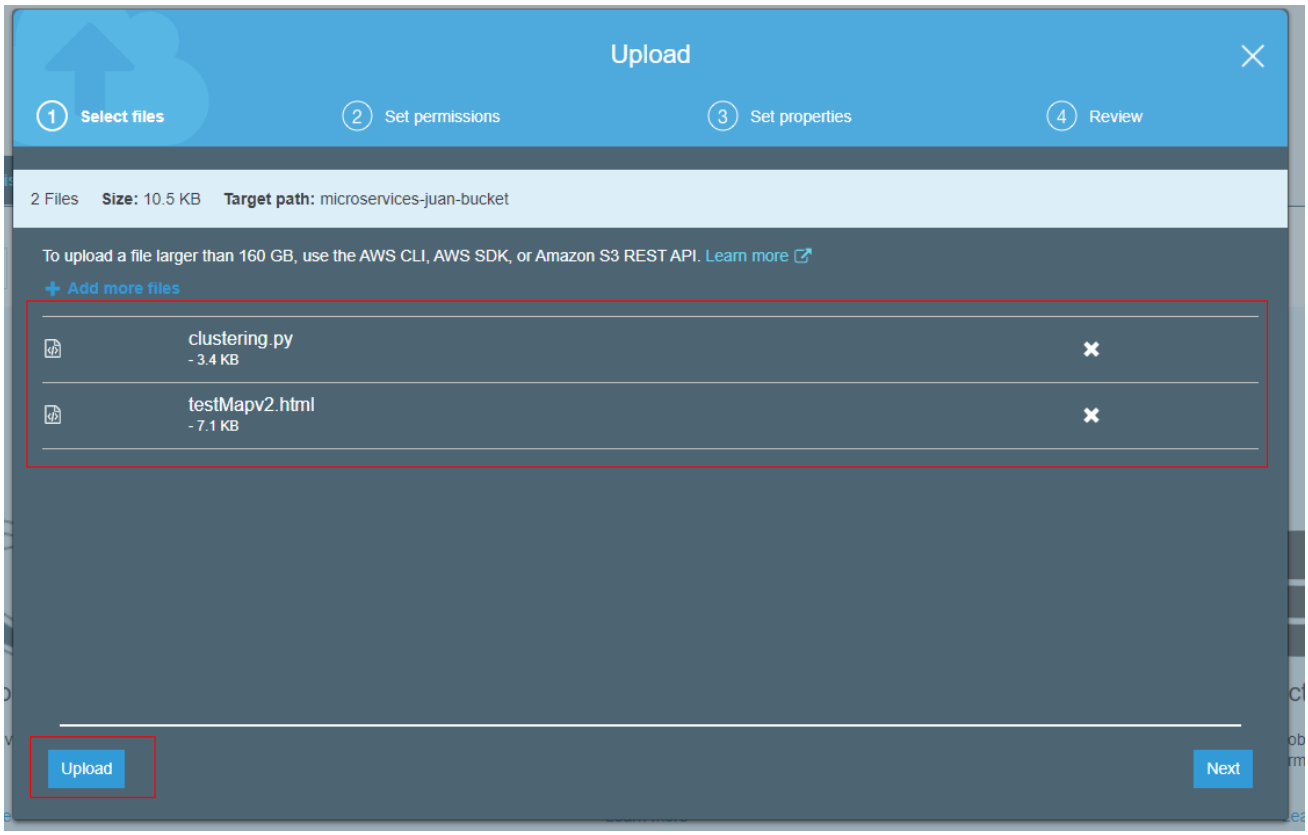
7.6. The bucket was created successfully, showing it in the buckets list, as you can see in the next screenshot:



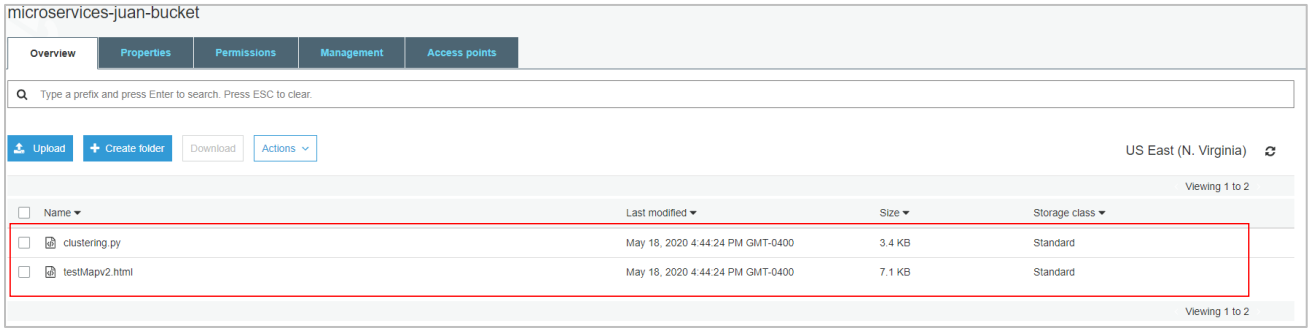
7.7. I'm going to upload my files into the service for deploying it, so click in the bucket and press the Upload button, as you can se in the next screenshot:



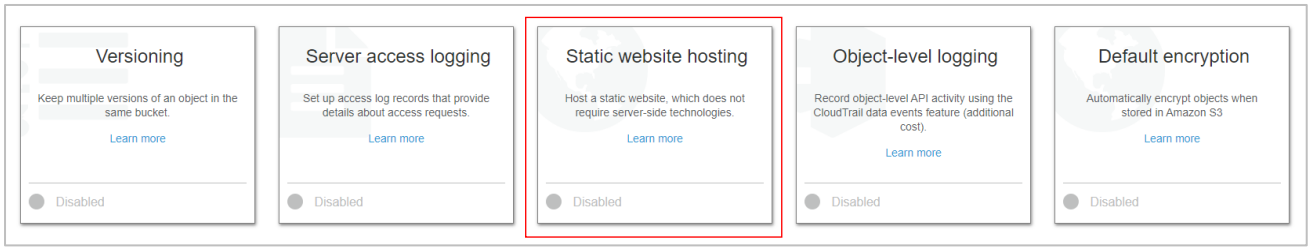
7.8. Just add the files from your computer and then Upload, as you can see in the next screenshot:



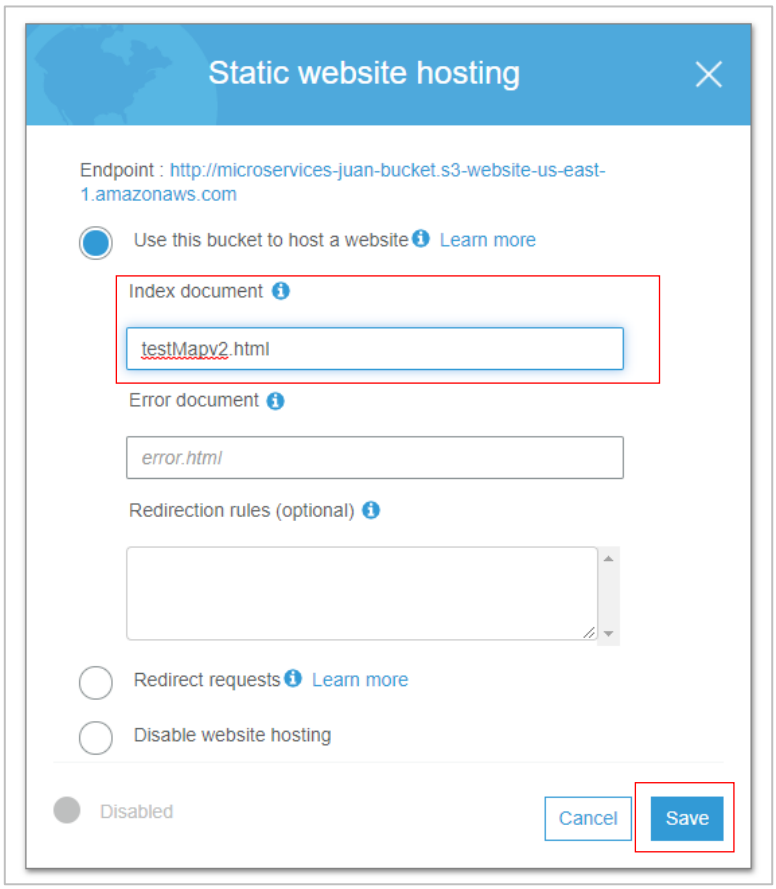
7.9. Then, the files must be visible in the bucket files list, as you can see in the next screenshot:



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:



7.11 Press the Static website hosting tittle 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:



7.12 Finally, I have my S3 bucket created and my 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