**AWS EC2 API GATEWAY PERSONAJES**

Vamos a crear una aplicación Mvc Net Core que será la encargada de consumir nuestro Api Gateway en producción.

Dentro de dicha aplicación, tendremos la posibilidad de subir las imágenes de nuestros Personajes a un Bucket S3.

Mostraremos las imágenes de los personajes a partir de dicho Bucket.

Comenzamos creando un nuevo bucket llamado **bucket-personajes-labs** y debemos configurarlo como **Public**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Texto, Escala de tiempo

Descripción generada automáticamente

Una vez creado, tenemos que generar una política de acceso al Bucket para poder acceder a los elementos mediante URL.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Y veremos nuestro Bucket público



Subimos nuestras imágenes a dicho Bucket y comprobamos que es funcional

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Una imagen de un hombre con traje y corbata

Descripción generada automáticamente

Como vamos a probar el proyecto en LOCAL, incluimos nuestras claves de IAM de administrador dentro del sistema mediante **aws configure**

Texto

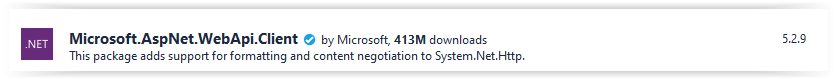
Descripción generada automáticamente

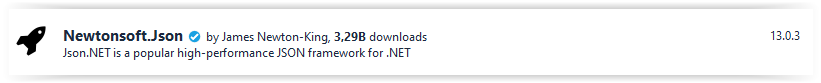
Creamos un nuevo proyecto llamado Mvc Net Core llamado **MvcAWSApiPersonajes**

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Agregamos los siguientes Nuget





Sobre **appsettings.json** vamos a incluir las claves necesarias para trabajar.

* ApiPersonajes
* S3BucketName
* S3BucketUrl

**APPSETTINGS.JSON**

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Sobre **Models**, creamos una nueva clase llamada **Personaje**

**PERSONAJE**

public class Personaje

{

public int IdPersonaje { get; set; }

public string Nombre { get; set; }

public string Imagen { get; set; }

public string Descripcion { get; set; }

}

Creamos una nueva carpeta llamada **Services** y una clase llamada **ServiceApiPersonajes**

**SERVICEAPIPERSONAJES**

public class ServiceApiPersonajes

{

private string UrlApi;

private MediaTypeWithQualityHeaderValue Header;

public ServiceApiPersonajes(IConfiguration configuration)

{

this.Header =

new MediaTypeWithQualityHeaderValue("application/json");

this.UrlApi =

configuration.GetValue<string>("AWS:ApiPersonajes");

}

private async Task<T> CallApiAsync<T>(string request)

{

using (HttpClient client = new HttpClient())

{

client.DefaultRequestHeaders.Clear();

client.DefaultRequestHeaders.Accept.Add(this.Header);

string urlRequest = this.UrlApi + request;

HttpResponseMessage response =

await client.GetAsync(urlRequest);

if (response.IsSuccessStatusCode)

{

T data = await response.Content.ReadAsAsync<T>();

return data;

}

else

{

return default(T);

}

}

}

public async Task CreatePersonajeAsync(Personaje personaje)

{

using (HttpClient client = new HttpClient())

{

string request = "api/personajes";

client.DefaultRequestHeaders.Clear();

client.DefaultRequestHeaders.Accept.Add(this.Header);

string jsonPersonaje = JsonConvert.SerializeObject(personaje);

StringContent content =

new StringContent(jsonPersonaje, Encoding.UTF8, "application/json");

HttpResponseMessage response =

await client.PostAsync(this.UrlApi + request, content);

}

}

public async Task<List<Personaje>> GetPersonajesAsync()

{

string request = "api/personajes";

List<Personaje> personajes = await this.CallApiAsync<List<Personaje>>(request);

return personajes;

}

public async Task<Personaje> FindPersonajeAsync(int id)

{

string request = "api/personajes/" + id;

Personaje personaje = await this.CallApiAsync<Personaje>(request);

return personaje;

}

}

Sobre **Controllers** creamos un nuevo controlador llamado **PersonajesController**

**PERSONAJESCONTROLLER**

public class PersonajesController : Controller

{

private ServiceApiPersonajes service;

public PersonajesController(ServiceApiPersonajes service)

{

this.service = service;

}

public async Task<IActionResult> Index()

{

List<Personaje> personajes = await this.service.GetPersonajesAsync();

return View(personajes);

}

public async Task<IActionResult> Details(int id)

{

Personaje personaje = await this.service.FindPersonajeAsync(id);

return View(personaje);

}

public IActionResult Create()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Create(Personaje personaje)

{

await this.service.CreatePersonajeAsync(personaje);

return RedirectToAction("Index");

}

}

Resolvemos las dependencias dentro de **Program**

**PROGRAM**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Creamos las vistas con **Scaffolding**

**INDEX.CSHTML**

@model List<Personaje>

<h1 style="color:blue">

Personajes AWS Api Gateway Labs

</h1>

<**a** **asp-controller**="Personajes"

**asp-action**="Create">

Create New

</**a**>

@foreach (Personaje personaje in Model){

<div class="card" style="width: 18rem;">

<img src="@personaje.Imagen" class="card-img-top" alt="...">

<div class="card-body">

<h5 class="card-title">@personaje.Nombre</h5>

<p class="card-text">

@personaje.Descripcion

</p>

</div>

</div>

}

**DETAILS.CSHTML**

@model Personaje

<h1 style="color:red">

Details

</h1>

<div class="card" style="width: 18rem;">

<img src="@Model.Imagen" class="card-img-top" alt="...">

<div class="card-body">

<h5 class="card-title">@Model.Nombre</h5>

<p class="card-text">

@Model.Descripcion

</p>

</div>

</div>

Ejecutamos y comprobamos su funcionamiento

A continuación, vamos a utilizar **IConfiguration** para mostrar las imágenes libres de nuestro Bucket

Incluimos las siguientes líneas dentro de **Index.cshtml** y **Details.cshtml**

@inject IConfiguration configuration

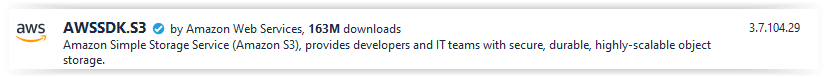
@{

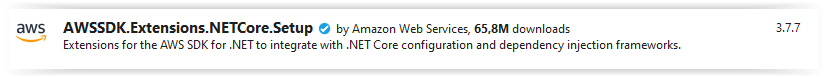
string UrlS3Bucket = configuration.GetValue<string>("AWS:S3BuckerUrl");

}

Ahora vamos a realizar la funcionalidad para poder subir imágenes a nuestro Bucket.

Para ello, instalamos los siguientes Nuget





Sobre **Services** creamos una nueva clase llamada **ServiceStorageAWS**

**SERVICESTORAGEAWS**

public class ServiceStorageAWS

{

private string BucketName;

private IAmazonS3 ClientS3;

public ServiceStorageAWS(IConfiguration configuration

, IAmazonS3 clientS3)

{

this.BucketName = configuration.GetValue<string>

("AWS:S3BucketName");

this.ClientS3 = clientS3;

}

//COMENZAMOS SUBIENDO FICHEROS AL BUCKET

//NECESITAMOS FileName, Stream y un Key/Value

public async Task<bool>

UploadFileAsync(string fileName, Stream stream)

{

PutObjectRequest request = new PutObjectRequest

{

InputStream = stream,

Key = fileName,

BucketName = this.BucketName

};

//DEBEMOS OBTENER UNA RESPUESTA CON EL MISMO TIPO

//DE REQUEST

PutObjectResponse response = await

this.ClientS3.PutObjectAsync(request);

if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)

{

return true;

}

else

{

return false;

}

}

}

Modificamos la vista **Create.cshtml**

**CREATE.CSHTML**

@model MvcAWSApiPersonajes.Models.Personaje

@{

ViewData["Title"] = "Create";

}

<h1>Create</h1>

<h4>Personaje</h4>

<hr />

<div class="row">

<div class="col-md-4">

<**form** **asp-action**="Create" enctype="multipart/form-data">

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

<div class="form-group">

<**label** **asp-for**="Nombre" class="control-label"></**label**>

<**input** **asp-for**="Nombre" class="form-control" />

<**span** **asp-validation-for**="Nombre" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Imagen" class="control-label"></**label**>

<**input** **asp-for**="Imagen" class="form-control" />

<**span** **asp-validation-for**="Imagen" class="text-danger"></**span**>

</div>

<div class="form-group">

<label>Descripción</label>

<textarea name="descripcion" class="form-control"></textarea>

<**span** **asp-validation-for**="Descripcion" class="text-danger"></**span**>

</div>

<label>Seleccione una Imagen para el personaje</label>

<input type="file" name="file" class="form-control" />

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

</**form**>

</div>

</div>

<div>

<**a** **asp-action**="Index">Back to List</**a**>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

Modificamos el método **Create** de **PersonajesController** y su constructor

**PERSONAJESCONTROLLER**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

[HttpPost]

public async Task<IActionResult> Create(Personaje personaje, IFormFile file)

{

personaje.Imagen = file.FileName;

using (Stream stream = file.OpenReadStream())

{

await this.serviceStorage.UploadFileAsync(personaje.Imagen, stream);

}

await this.service.CreatePersonajeAsync(personaje);

return RedirectToAction("Index");

}

Resolvemos las dependencias dentro de **Program**

**PROGRAM**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Y ya tendremos toda la aplicación funcional con S3 y Api Gateway

Es el momento de desplegar nuestra App dentro de una máquina **EC2**

Para ello, subimos nuestra App dentro de Github como **public** para no estar jugando con Tokens

Forma, Rectángulo

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Descargamos las Keys y las ponemos en una ubicación conocida

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Pondremos nuestra máquina EC2 dentro del grupo de RDS de MySql y abrimos los puertos para poder acceder a MySQL

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Escribimos el Bash

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

ghp\_y92fV4lYBpjjFSRgOJTagJx00XCfwQ2vKlDK

git clone https://ghp\_y92fV4lYBpjjFSRgOJTagJx00XCfwQ2vKlDK@github.com/serraguti/MvcAWSApiPersonajes.git

Bash de EC2

#!/bin/bash

sudo su

yum update -y

yum install httpd -y

yum install git -y

sudo rpm -Uvh https://packages.microsoft.com/config/centos/7/packages-microsoft-prod.rpm

sudo yum install dotnet-sdk-6.0 -y

sudo yum install dotnet-runtime-6.0 -y

sudo yum install libunwind -y

sudo yum install mod\_ssl -y

systemctl start httpd

systemctl enable httpd

sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 5000

sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 -j REDIRECT --to-port 5001

**Comandos extra**

cd /etc/pki/tls/certs

sudo ./make-dummy-cert localhost.crt

**sudo iptables -t nat -v -L -n --line-number**

dotnet run --urls <https://0.0.0.0:5001>

**nohup dotnet run --urls "**[**https://0.0.0.0:5001**](https://0.0.0.0:5001)**" > /dev/null 2>&1 &**

**sudo iptables -t nat --delete PREROUTING 1**

Abrimos los puertos necesarios en nuestro Grupo de seguridad

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Una vez iniciada nuestra máquina, nos conectamos

Texto

Descripción generada automáticamente

mkdir /var/mvcs

cd /var/mvcs

git clone https://ghp\_y92fV4lYBpjjFSRgOJTagJx00XCfwQ2vKlDK@github.com/serraguti/MvcAWSApiPersonajes.git

Entramos en nuestra máquina y todo funciona, excepto nuestra funcionalidad S3 al subir un fichero

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Debemos crear un Role de IAM con permisos sobre S3 y ponerlo en nuestra máquina

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Volvemos a EC2 y entramos en nuestra máquina

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico, Teams

Descripción generada automáticamente

Y podremos visualizar todo en correcto funcionamiento

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente