

# CERTIFICACION AZURE

miércoles, 12 de marzo de 2025 13:08

Tenemos actualmente 462 preguntas

Tenemos que generar nuestros VCE a partir del PDF

Si tocamos a 20 preguntas cada uno...

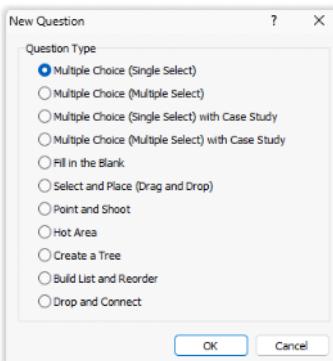
- 1) Buscar las soluciones posibles leyendo los foros (Exam topics) o documentación
- 2) Generar un fichero VCE individual, revisando que esté bien, con cariño
- 3) En las preguntas del VCE pondremos el número de Question y si tiene Topic
- 4) Intentemos copiar los textos, no pegar.
- 5) Nos aseguramos que el texto que hemos copiado del PDF es legible y nada De copiar cosas como fi o lo que sea. Si vemos que el texto NO está bien Lo corregimos (Gracias Manuel del Madrid)
- 6) Si es un caso de estudio, pondremos **CASE STUDY NOMBRE DEL CASO ARRIBA** En negrita
- 7) Los casos de estudio tendrán los Exhibits del caso de estudio que corresponda
- 8) Subir el fichero VCE al Temas en una carpeta llamada VCE Brutos
- 9) El nombre del fichero será el siguiente:

01\_Paco\_1\_20.vce

02\_Andrei\_21\_40.vce

La entrega del primer VCE será el **LUNES 17 de marzo**

Tendremos preguntas de Teoría



## Question

### Question 1, Topic 1

A company manages capital equipment for an electric utility company. The company has a SQL Server database that contains maintenance records for the equipment.

Technicians who service the equipment use the Dynamics 365 Field Service mobile app on tablet devices to view scheduled assignments.

Technicians use a canvas app to display the maintenance history for each piece of equipment and update the history.

Managers use a Power BI dashboard that displays Dynamics 365 Field Service and real-time maintenance data.

Due to increasing demand, managers must be able to work in the field as technicians.

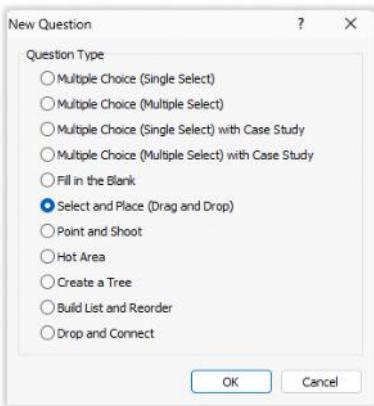
You need to design a solution that allows the managers to work from one single screen.

What should you do?

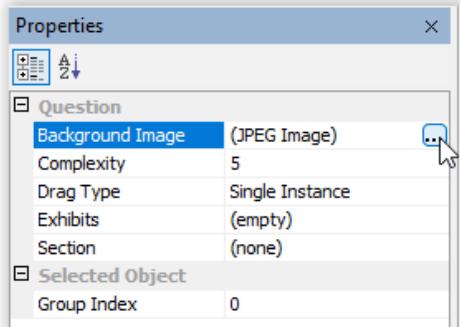
Available Choices (select all choices that are correct)

- A. Add the maintenance history app to the Field Service Mobile app.
- B. Add the manager Power BI dashboard to the Field Service mobile app.
- C. Create a new maintenance canvas app from within the Power BI management dashboard.
- D. Add the maintenance history app to the Power BI dashboard.

## Preguntas de Drag and Drop



Los cuadros los hacemos correctamente (Gracias Sofía del Atleti)



**Source Object:** El origen para arrastrar

**Target Place:** El lugar de destino de las posibles soluciones

Posteriormente, pulsamos en **Answer** y ponemos la respuesta que sea

**Question 11, Topic 1**

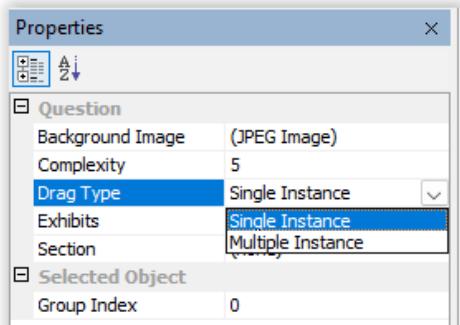
Teachers in a school district use Azure skill bots to teach specific classes. Students sign into an online portal to submit completed homework to their teacher for review. Students use a Power Virtual Agents chatbot to request help from teachers. You need to incorporate the skill bot for each class into the homework bot. Which three actions should you perform in sequence? To answer, move the appropriate actions from the list of actions to the answer area and arrange them in the correct order.

**Explanation/Reference**

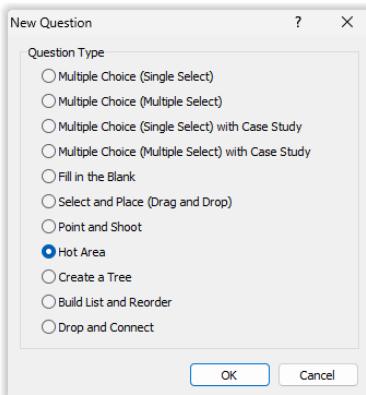
Action	Answer Box
Create a manifest for the skill bot.	Box 1
Register the skill bot in Azure Active Directory.	Box 2
Register the homework bot in Power Virtual Agents.	Box 3
Register the homework bot in Azure Active Directory.	Box 4
Create a manifest for the homework bot.	Box 5
Register the skill bot in Power Virtual Agents.	Box 6

#### DRAG AND DROP CON SELECCIÓN MULTIPLE

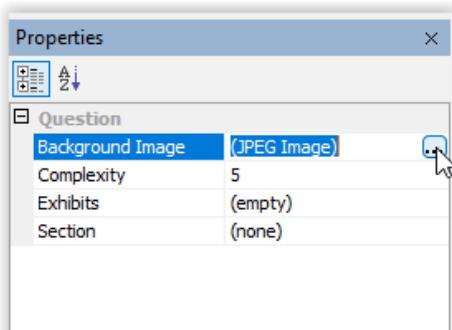
Son exactamente iguales al anterior, pero podemos arrastrar más de un mismo elemento.



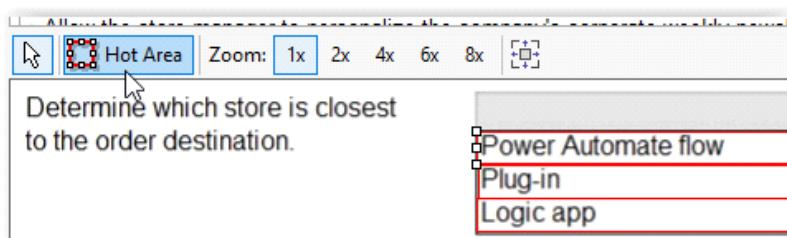
## HOTSPOT QUESTION



La imagen de fondo la ponemos mediante **Background Image**



Con **Hot Area** vamos marcando las zonas donde el usuario puede seleccionar  
De esquina a esquina completa de la caja. (Gracias Victor del Madrid)



En **Answer** ponemos la solución

## Question

### Question 3, Topic 1

You create a suite of Power Platform-based order management canvas apps for a bakery that has five retail stores. Each store uses a tablet device to manage inventory and process orders.

You need to make the following changes to the original order tracking app:

When an online order for delivery is received, send the order to the bakery that is located closest to the order destination.

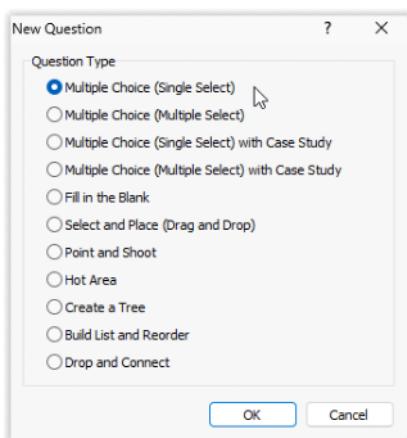
When an online order for pickup is received, require store staff to enter an estimated time in an app. Staff must prepare the order and then use the app to notify the customer when the order is ready.

The screenshot shows a Microsoft Power Platform canvas app interface. At the top, there are zoom controls (1x, 2x, 4x, 6x, 8x) and a 'Hot Area' tool icon. Below the toolbar, there are three dropdown menus:

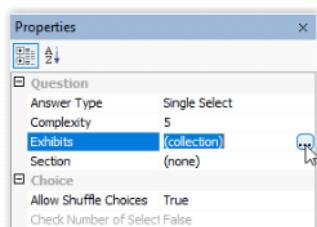
- Determine which store is closest to the order destination.** The menu contains three items: "Power Automate flow" (selected), "Plug-in", and "Logic app".
- Estimate the time required to prepare an order and notify the customer.** The menu contains three items: "New screen in an existing order canvas app" (selected), "New canvas app", and "New logic app".
- Send the newsletter by email to customers.** The menu contains three items: "Power Automate flow triggered from an email button" (selected), "Power Automate flow triggered manually", and "Power Automate UI flow triggered from an email button".

## CASOS DE ESTUDIO

Las preguntas, aunque sean casos de estudio, son las MISMAS que las anteriores, es decir,  
No marcamos las de **With Case Study**



Las imágenes estarán dentro de Exhibits



En GRANDE y que se vea bien (negrita) el caso de estudio

Question

**CASE STUDY - Northwind Traders**

Question #36 Topic 6

You need to determine the cause of the 404 error when connecting to the production instance of the Web API.  
What do you identify?

Available Choices (select all choices that are correct)

- A. The web service lacks data for the record.
- B. An authentication error occurred.
- C. The request timed-out.
- D. The host name in the URL is missing a valid value.

Explanation/Reference

Yo os voy a proporcionar algunas imágenes de casos de estudio.  
Si no están las imágenes del caso de estudio que os ha tocado, las generamos.

# AZURE DEVELOPING AZ-204

lunes, 17 de marzo de 2025 9:11

El portal de Azure es la siguiente dirección:

<https://portal.azure.com/#home>

Todo en Azure está organizado mediante **Resources**

Cada Resource podemos pensar que es como un sistema de carpetas.

Al tener servicios dentro de los Resources, podemos eliminar fácilmente o localizar los recursos de una forma organizada.

Tendremos recursos temporales que crearemos solamente para algunas prácticas y los eliminaremos posteriormente.

Azure es gigante, es un mundo de servicios donde se juntan desarrollo, sistemas, IA y cualquier idea que tengamos hacia los servicios.

En este módulo, no solo vamos a desarrollar, también tendremos que aprender cómo desplegar servicios y trabajar con ellos, por ejemplo, cómo montar una app Net Core dentro de una máquina virtual y asignar permisos.

Servicios de desarrollo:

- **Servicios Api/Rest:** Son servicios que devuelven información en formato JSON/XML
- **Api Management:** Es un administrador de APIs creado para ofrecer las APIs estructuradas como servicio a los clientes o usuarios.
- **Nuget:** Son clases/librerías para la comunidad de Net Core, crearemos nuestros propios Nuget y los mostraremos al mundo.
- **Docker:** Utilización de Docker y despliegue para montar de forma sencilla servicios aislados como, por ejemplo, montar una imagen de Postgres
- **Storage:** Es almacenamiento en la nube, nos permite tener nuestros recursos aislados o públicos mediante una URL.
- **Web Jobs:** Son lógicas que podemos crear (bucles) y asociar a servicios de Azure. Dichas lógicas podemos hacer que se ejecuten constantemente o mediante un CRON.
- **Azure Functions:** Esto se le llama código aislado, es un código que no está asociado a nada, un código en la nube y que podemos ejecutar cuando lo necesitemos.
- **Azure Key Vault:** Es un baúl para almacenar las claves de nuestras aplicaciones. En lugar de guardar las Keys dentro de **appsettings**, las almacenaremos en dicho Servicio.
- **Azure Cosmos Db:** Es la base de datos NoSQL de Microsoft.
- **Maquinas virtuales:** Creación de máquinas virtuales, despliegue de servicios en su interior y comunicaciones entre ellas.
- **Azure Resource Manager:** Son plantillas para generar recursos en Azure por código.
- **Azure Cloud Shell:** Líneas de comando para controlar recursos dentro de la Nube de Azure.
- **Active Directory:** Son los usuarios dados de alta dentro de nuestro dominio de Azure, es decir, en nuestro caso, los usuarios de @tajamar365.com.
- **EventHub y EventGrid:** Es un servicio que permite almacenar información y podemos

Dar formato a dicha información y enviarla a otros servicios.

- **Integración continua CI/CD:** Permite el despliegue de aplicaciones mediante una serie de Herramientas, es decir, por ejemplo, desde Github, al hacer cambios sobre una App, que se Publique directamente en algún Slot que yo le indique.
- **Net Core Aspire:** Es una nuevo servicio que nos permite desplegar múltiples aplicaciones desde Un mismo lugar sin necesidad de producción.
- **Azure IA:** Funcionalidades de IA
- **Logic Apps:** Utilizar Power Automate dentro de Azure.
- **Azure Cache Redis:** Es utilizar Session, pero dentro de Azure

Tenemos tres conceptos fundamentales dentro de Azure:

- **IaaS:** Infraestructure as Service. Contratamos o alquilamos Hardware en la nube de Azure. Yo me encargo de todo, VM, Servidor Apache, Net Core
- **SaaS:** Software as Service. Desplegamos aplicaciones dentro de Azure. No quiero controlar nada , solamente necesito que funcione mi app y desplegarla.
- **PaaS:** Platform as Service. Utilizamos servicios de Azure para nuestras aplicaciones como, por ejemplo Una base de datos SQL Server.

Lo primero que vamos a realizar, como prueba, es desplegar la aplicación de seguridad de Empleados dentro de Azure.

Abrimos Management Studio y copiamos los datos de Hospital dentro de la base de datos **AZURETAJAMAR**

**Importante:** NUNCA crearemos una base de datos dentro de Azure. Utilizaremos solamente la Base de datos que tenemos actual.

Usuario: **adminsql**

Password: **Admin123**

# NUGET

lunes, 17 de marzo de 2025 10:52

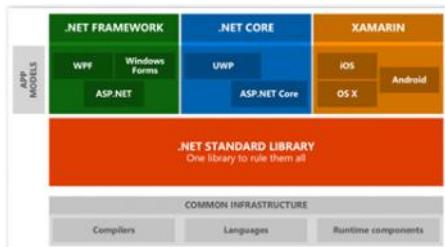
Los Nuget forman parte del desarrollo de Azure y entran dentro de la certificación pero, No se despliegan dentro del Portal de Azure.

Utilizan otro portal para almacenar todas las librerías asociadas a mi nombre.

Los Nuget son herramientas de librerías, no utilizan nada gráfico.

Tenemos tres tipos de librerías:

- Nuget Framework: Librerías solamente para proyectos de Windows
- Nuget Net Core: Se utilizan para aplicaciones Net Core y dependen de la versión
- Net Standard: Son librerías que no utilizan Net Core y se pueden utilizar en Cualquier tipo de proyectos



Un Nuget puede tener dependencias su vez, es decir, imaginemos que estamos creando Un Nuget para mejorar lo de Newton JSON, la librería que utilizamos es la de JSON y Las dependencias se agregan automáticamente aunque no lo veamos.

Un Nuget está compuesto por versiones y, una gran ventaja radica en que si realizamos Una nueva versión, los programadores que utilizan nuestro Nuget son automáticamente Informados que existe una nueva versión.

Necesitamos una serie de pasos para poder desplegar Nuget.

- 1) Instalar el cliente de Nuget para poder publicar las librerías que tengamos.

<https://dist.nuget.org/win-x86-commandline/latest/nuget.exe>

La publicación de Nuggets se realiza mediante líneas de comandos, por lo que vamos a Copiar el fichero a alguna ruta de Windows para acceder a la librería

C:\Windows\System32

- 2) Darnos de alta dentro del portal de desarrolladores de Nuget.org

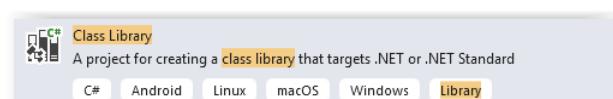
<https://www.nuget.org/>

Esta cuenta está asociada a nuestro Nuget, es decir, vamos a salir al mundo de Net Core.

Para poder publicar Nuget, necesitamos un Api Key que irá asociada a nuestra cuenta.

Vamos a realizar una librería estática, es decir, no vamos a leer nada, tendremos todo Copiar y pegar

Creamos un proyecto de tipo **Class Library** llamado **NugetCoches**



COCHE

```
public class Coche
{
    0 references
    public int IdCoche { get; set; }
    0 references
    public string Marca { get; set; }
    0 references
    public string Modelo { get; set; }
    0 references
    public string Imagen { get; set; }
}
```

Creamos una nueva clase llamada **Garaje**

Ejemplo de Coches

```

Coche c = new Coche
{
    IdCoche = 0,
    Marca = "DMG",
    Modelo = "Delorian",
    Imagen = "https://static.motor.es/fotos-noticias/2015/10/min652x435/curiosidades-delorean-regreso-al-futuro-201523728\_4.jpg"
};
coches.Add(c);
c = new Coche
{
    IdCoche = 1,
    Marca = "PONTIAC"
    ,
    Modelo = "FireBird"
    ,
    Imagen = "https://i.ytimg.com/vi/UJFwmjfTSJw/hqdefault.jpg"
};
coches.Add(c);
c = new Coche
{
    IdCoche = 2
    ,
    Marca = "Volkswagen"
    ,
    Modelo = "Escarabajo"
    ,
    Imagen = "https://i.ytimg.com/vi/AP-HLHi0HUw/maxresdefault.jpg"
};
coches.Add(c);
c = new Coche
{
    IdCoche = 3
    ,
    Marca = "Citroen"
    ,
    Modelo = "2 CV"
    ,
    Imagen = "http://iconroad.es/onewebmedia/iconos%20Citro%C3%ABn%202CV%20En%20el%20cine%204.png"
};

```

## GARAJE

```

public class Garaje
{
    List<Coche> coches;

    public Garaje()
    {
        this.coches = new List<Coche>();
        Coche c = new Coche
        {
            IdCoche = 0,
            Marca = "DMG",
            Modelo = "Delorian",
            Imagen = "https://static.motor.es/fotos-noticias/2015/10/min652x435/curiosidades-delorean-regreso-al-futuro-201523728\_4.jpg"
        };
        coches.Add(c);
        c = new Coche
        {
            IdCoche = 1,
            Marca = "PONTIAC"
            ,
            Modelo = "FireBird"
            ,
            Imagen = "https://i.ytimg.com/vi/UJFwmjfTSJw/hqdefault.jpg"
        };
        coches.Add(c);
        c = new Coche
        {
            IdCoche = 2
            ,
            Marca = "Volkswagen"
            ,
            Modelo = "Escarabajo"
            ,
            Imagen = "https://i.ytimg.com/vi/AP-HLHi0HUw/maxresdefault.jpg"
        };
        coches.Add(c);
        c = new Coche
        {
            IdCoche = 3
            ,
            Marca = "Citroen"
            ,
            Modelo = "2 CV"
            ,
            Imagen = "http://iconroad.es/onewebmedia/iconos%20Citro%C3%ABn%202CV%20En%20el%20cine%204.png"
        };
        coches.Add(c);
    }

    public List<Coche> GetCoches()
    {
        return this.coches;
    }

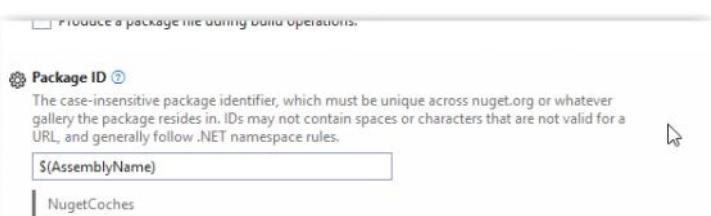
    public Coche FindCoche(int id)
    {
        return this.coches.FirstOrDefault(x => x.IdCoche == id);
    }
}

```

```
}
```

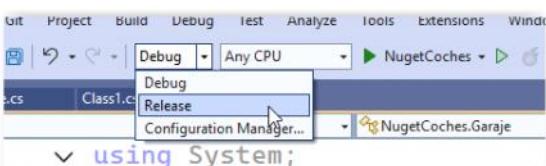
El siguiente paso es configurar nuestro proyecto para ser publicado como **Nuget**

**Nota:** El nombre del identificador debe ser único global



Nos generará un fichero con extensión **PKG** que será el que publicaremos mediante **Nuget.exe**

Debemos poner la publicación en versión **Release**

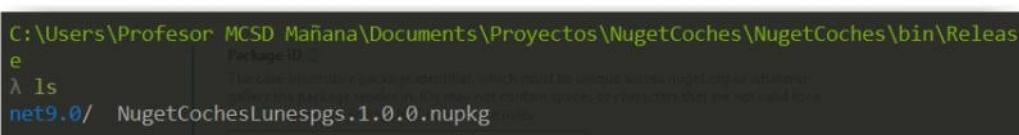


Una vez que hemos puesto títulos/iconos y demás, al realizar **Build** nos genera el package Del proyecto para publicar en Nuget

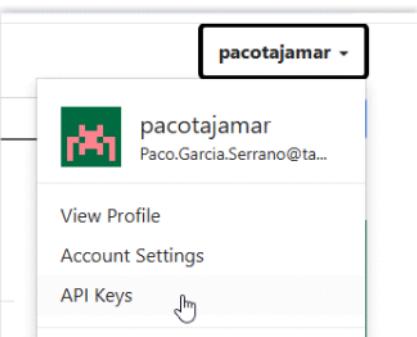
```
1>NugetCoches -> C:\Users\Profesor MCSD Mañana\Documents\Proyectos\NugetCoches\NugetCoches\bin\Release\net9.0\NugetCoches.dll
1>The package NugetCochesLunespgs.1.0.0 is missing a readme. Go to https://aka.ms/nuget/authoring-best-practices/readme to learn why package readmes are important
1>Successfully created package 'C:\Users\Profesor MCSD Mañana\Documents\Proyectos\NugetCoches\NugetCoches\NugetCoches\bin\Release\NugetCochesLunespgs.1.0.0.nupkg'.
1>Done building project "NugetCoches.csproj".
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ======
===== Build completed at 11:23 and took 04,372 seconds ======
```

Es el momento de publicar el package.

Abrimos CMDER y entramos en la ruta dónde tenemos PKG



Necesitamos generar un Api Key para la publicación.



## Create

<b>Key Name *</b>	<b>Expires In</b>
apikeylunes	365 days
<b>Package Owner *</b>	
pacotajamar	
<b>Select Scopes</b>	
<input checked="" type="checkbox"/> Push	
<input checked="" type="radio"/> Push new packages and package versions	
<input type="radio"/> Push only new package versions	
<input type="checkbox"/> Unlist or relist package versions	

## Select Packages

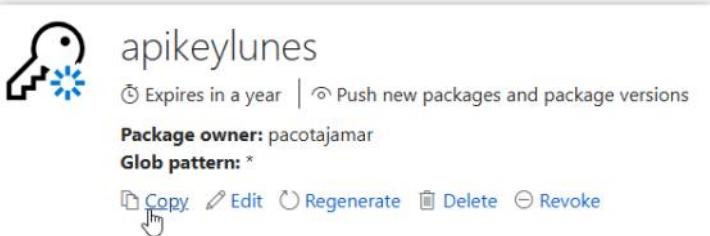
To select which packages to associate with a key, use a glob pattern, select individual packages, or both.

### Glob Pattern

\*

A glob patter

Copiamos la clave



Para poder subir nuestro nuget necesitamos la siguiente instrucción en línea de comandos

```
nuget push FICHERO.pkg API-KEY -source https://api.nuget.org/v3/index.json
```

Escribimos nuestros datos y ejecutamos dentro de CMDER

```
C:\Users\Profesor MCSD Mañana\Documents\Proyectos\NugetCoches\NugetCoches\bin\Releases\tdr37ed6evijmaxs1kz7sx4qaerkyy2rq4s2xahy
nuget push NugetCochesLunespgs.1.0.0.nupkg oy2kg6tdr37ed6evijmaxs1kz7sx4qaerkyy2rq
4s2xahy -source https://api.nuget.org/v3/index.json
ADVERTENCIA: No se puede cargar el archivo o ensamblado 'Windows.Devices.Printers.Ex
tensions.dll' ni una de sus dependencias. Se esperaba que el módulo tuviera un manif
iesto de ensamblado.
Pushing NugetCochesLunespgs.1.0.0.nupkg to 'https://www.nuget.org/api/v2/package'...
PUT https://www.nuget.org/api/v2/package/
ADVERTENCIA: No se puede cargar el archivo o ensamblado 'Windows.Devices.Printers.Ex
tensions.dll' ni una de sus dependencias. Se esperaba que el módulo tuviera un manif
iesto de ensamblado.
```

Tarda un tiempo en Indexar los datos de forma Global y, dentro de nuestra cuenta, podremos Ver el nuevo Package subido.

	NugetCochesLunespgs	pacotajamar	pacotajamar (0 certificates)	0	1.0.0
--	---------------------	-------------	------------------------------	---	-------

Para comprobar que es funcional, vamos a crear otro proyecto de tipo Console para Dibujar los datos de los coches de forma sencilla.

Creamos un nuevo proyecto llamado **TestingNugetCoches**

```
 NugetCochesLunespgs by pacotajamar 1.0.0
Esta es una prueba de paquetes Nuget de Coches de Lunes con lluvia
```

```

using NugetCoches;

Console.WriteLine("Mis coches de Nuget");
Garaje g = new Garaje();
List<Coche> cars = g.GetCoches();
foreach (Coche c in cars)
{
    Console.WriteLine(c.Marca + " " + c.Modelo);
}
Console.WriteLine("Fin de programa");

```

```

Mis coches de Nuget
DMG DeLorian
PONTIAC FireBird
Volkswagen Escarabajo
Citroen 2 CV
Fin de programa

```

A continuación, vamos a crear otro Nuget, pero esta vez, lo que haremos será algo más complejo.

Vamos a leer desde un Api y mapearemos los datos del Api y se los daremos directamente a los programadores con Clases.

<https://northwind-api.miloudi.dev/v1/customers>

```

{
  "metadata": {
    "page": 1,
    "pageSize": 10,
    "totalCount": 96,
    "totalPages": 10,
    "links": {
      "self": "https://northwind-api.miloudi.dev/v1/customers",
      "first": "https://northwind-api.miloudi.dev/v1/customers?page=1",
      "last": "https://northwind-api.miloudi.dev/v1/customers?page=10",
      "next": "https://northwind-api.miloudi.dev/v1/customers?page=2",
      "prev": null
    }
  },
  "data": [
    {
      "customerId": "ALFKI",
      "companyName": "Alfreds Futterkiste",
      "contactName": "Maria Anders",
      "contactTitle": "Sales Representative",
      "address": "Obere Str. 57",
      "city": "Berlin",
      "region": null,
      "postalCode": "12209",
      "country": "Germany",
      "phone": "030-0074321",
      "fax": "030-0076545"
    },
    {
      "customerId": "ANATR",
      "companyName": "Ana Trujillo Emparedados y helados",
      "contactName": "Ana Trujillo"
    }
  ]
}

```

Nos interesa leer DATA

Creamos un nuevo proyecto de tipo Class Library llamado **NugetCustomersPGS**

Vamos a mapear los datos que leamos desde el servicio Api mediante Newton JSON



Creamos una carpeta llamada **Models** y una clase llamada **Customer**

Debemos mapear cada Property del JSON con la decoración `[JsonProperty("PROPIEDAD")]`

**CUSTOMER**

```

public class Customer
{
    [JsonProperty("customerId")]
    0 references
    public int IdCustomer { get; set; }
    [JsonProperty("contactName")]
    0 references
    public string Contacto { get; set; }
    [JsonProperty("companyName")]
    0 references
    public string Empresa { get; set; }
    [JsonProperty("address")]
    0 references
    public string Direccion { get; set; }
}

```

Como los Customers están dentro de la palabra **data** debemos crear una clase a su vez  
Y mapear dicha palabra con una colección.

Creamos una nueva clase llamada **CustomersList**

**CUSTOMERSLIST**

```

public class CustomersList
{
    [JsonProperty("data")]
    0 references
    public List<Customer> Customers { get; set; }
}

```

Si consumimos una base de datos, se utilizan **Repositories**

Si consumimos recursos del Cloud, se utilizan **Services**

Creamos una nueva carpeta llamada **Services** y una clase llamada **ServiceCustomers**.

**SERVICECUSTOMERS**

```

public class ServiceCustomers
{
    public async Task<CustomersList> GetCustomersListAsync()
    {
        WebClient client = new WebClient();
        client.Headers["content-type"] = "application/json";
        //COPIAMOS LA URL COMPLETA PARA LOS CUSTOMERS
        string url =
            "https://northwind-api.miloudi.dev/v1/customers";
        string dataJson = await
            client.DownloadStringTaskAsync(url);
        CustomersList clients =
            JsonConvert.DeserializeObject<CustomersList>(dataJson);
        return clients;
    }
}

```

Sobre el proyecto **Properties** y configuramos nuestro **Package**

## Package

General

Generate NuGet package on build

Produce a package file during build operations.

### Package ID

The case-insensitive package identifier, which must be unique across nuget.org or whatever gallery the package resides in. IDs may not contain spaces or characters that are not valid for a URL, and generally follow .NET namespace rules.

\$AssemblyName

NugetCustomersPGS

### Title

A human-friendly title of the package, typically used in UI displays as on nuget.org and the Package Manager in Visual Studio.

Nuget Northwind Customers Service JSON

Generamos el Package después de ponerlo como Release

```
1>NugetCustomersPGS -> C:\Users\Profesor MCSD Mañana\Documents\Proyectos\NugetCustomersPGS\NugetCustomersPGS\bin\Release\net9.0\NugetCustomersPGS.dll
1>The package NugetCustomersPGS.1.0.0 is missing a readme. Go to https://aka.ms/nuget/authoring-best-practices/readme to learn why package readmes are important.
1>Successfully created package 'C:\Users\Profesor MCSD Mañana\Documents\Proyectos\NugetCustomersPGS\NugetCustomersPGS\bin\Release\NugetCustomersPGS.1.0.0.nupkg'.
1>Done building project "NugetCustomersPGS.csproj".
===== Build: 1 succeeded 0 failed 0 up-to-date 0 skipped =====
```

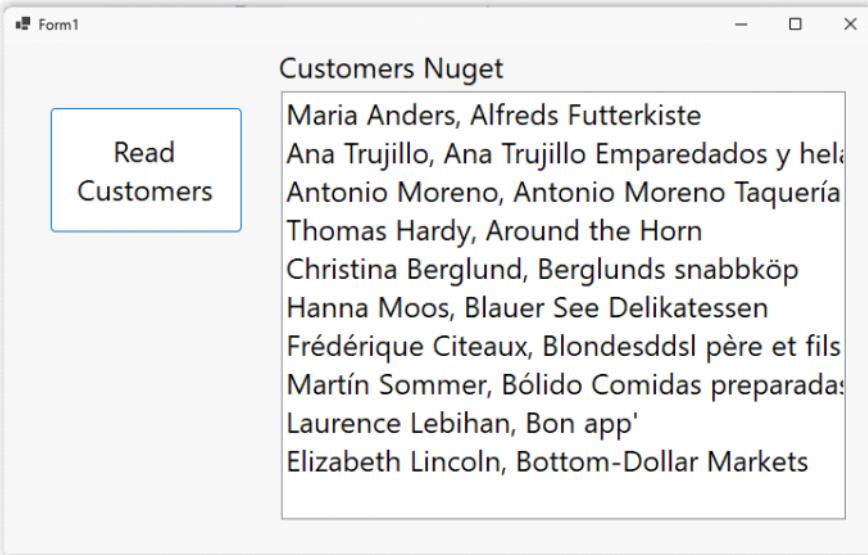
Publicamos nuestro Package dentro de Nuget.org

```
C:\Users\Profesor MCSD Mañana\Documents\Proyectos\NugetCustomersPGS\NugetCustomersPGS\bin\Release\NugetCustomersPGS.1.0.0.nupkg
1 nuget push NugetCustomersPGS.1.0.0.nupkg oy2kg6tdr37ed6evijmaxs1kz7sx4qaerkyy2rq4s
2xahy -source https://api.nuget.org/v3/index.json
ADVERTENCIA: No se puede cargar el archivo o ensamblado 'Windows.Devices.Printers.Extensions.dll' ni una de sus dependencias. Se esperaba que el módulo tuviera un manifiesto de ensamblado.
Pushing NugetCustomersPGS.1.0.0.nupkg to 'https://www.nuget.org/api/v2/package'...
PUT https://www.nuget.org/api/v2/package/
```

Y ya tendremos publicado nuestro nuevo Package

Package ID	Owners	Signing Owner	Downloads	Latest Version
 NugetCustomersPGS	pacotajamar	pacotajamar (0 certificates)	0	1.0.0

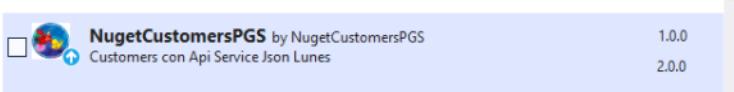
Mientras lo publica, creamos una aplicación de tipo Windows Forms llamada TestingNugetCustomers



```
private async void btnReadCustomers_Click(object sender, EventArgs e)
{
    ServiceCustomers service =
        new ServiceCustomers();
    CustomersList data = await service.GetCustomersList();
    List<Customer> clients = data.Customers;
    foreach (Customer c in clients)
    {
        this.lstCustomers.Items.Add
            (c.Contacto + ", " + c.Empresa);
    }
}
```



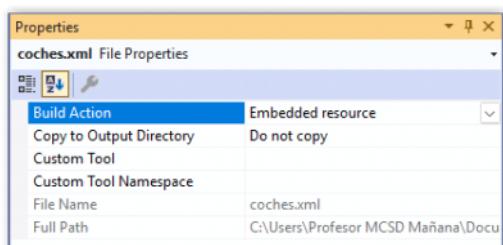
```
iments\Proyectos\NugetCustomersPGS\NugetCustomersPGS\bin\Release\net9.0\NugetCustomersPGS.dll
1. Go to https://aka.ms/nuget/authoring-best-practices/readme to learn why package readmes are important.
\mañana\Documents\Proyectos\NugetCustomersPGS\NugetCustomersPGS\bin\Release\NugetCustomersPGS.2.0.0.nupkg'.
skipped -----
```



El siguiente paso será realizar otro Nuget en el que aprenderemos a utilizar Linq to XML y además vamos a ver cómo podemos recuperar datos que tengamos Almacenados dentro del Assembly (Package)

Creamos un nuevo proyecto Class Library llamado NugetCarsPGS

Copiamos **coches.xml** dentro de nuestro proyecto y en **Properties** lo ponemos como **Recurso incrustado**



Sobre el proyecto, creamos una nueva carpeta llamada **Models** y una clase llamada **Coche**

**Coche**

```
public class Coche
{
    public int IdCoche { get; set; }
    public string Marca { get; set; }
    public string Modelo { get; set; }
    public string Imagen { get; set; }
}
```

Como leemos de un recurso físico, creamos una carpeta llamada **Repositories** y una clase llamada **RepositoryCoches**

**REPOSITORYCOCHES**

```
public class RepositoryCoches
{
    private XDocument document;
    public RepositoryCoches()
    {
        //PARA RECUPERAR UN RECURSO INCRUSTADO
        //NECESITAMOS EL namespace Y, SI LO TUVIERA
        //LA CARPETA DONDE ESTUVIERA EL RECURSO INCRUSTADO.
        string resourceName = "NugetCarsPGS.coches.xml";
        //LOS FICHEROS SE RECUPERAN MEDIANTE BYTES, ES DECIR
        //MEDIANTE Stream
        Stream stream =
            this.GetType().Assembly
                .GetManifestResourceStream(resourceName);
        //COMO ES UN XML FISICO, SE UTILIZA EL METODO LOAD
        this.document = XDocument.Load(stream);
    }

    public List<Coche> GetCoches()
    {
        var consulta =
            from datos in this.document.Descendants("coche")
            select datos;
        List<Coche> cars = new List<Coche>();
        foreach (var tag in consulta)
        {
            Coche car = new Coche();
            car.IdCoche = int.Parse(tag.Element("idcoche").Value);
            car.Marca = tag.Element("marca").Value;
            car.Modelo = tag.Element("modelo").Value;
            car.Imagen = tag.Element("imagen").Value;
            cars.Add(car);
        }
        return cars;
    }
}
```

Publicamos nuestro Package

```
C:\Users\Profesor MCSD Mañana\Documents\Proyectos\NugetCarsPGS\NugetCarsPGS\bin\Release
nuget push NugetCarsPGS.1.0.0.nupkg oy2kg6tdr37ed6evijmaxslkz7sx4qaerkyy2rq4s2xahy
-source https://api.nuget.org/v3/index.json
ADVERTENCIA: No se puede cargar el archivo o ensamblado 'Windows.Devices.Printers.Ex
tensions.dll' ni una de sus dependencias. Se esperaba que el módulo tuviera un manif
iesto de ensamblado.
Pushing NugetCarsPGS.1.0.0.nupkg to 'https://www.nuget.org/api/v2/package'...
PUT https://www.nuget.org/api/v2/package/
ADVERTENCIA: License missing. See how to include a license within the package: https
://aka.ms/nuget/authoring-best-practices#licensing., Readme missing. Go to https://ak
```

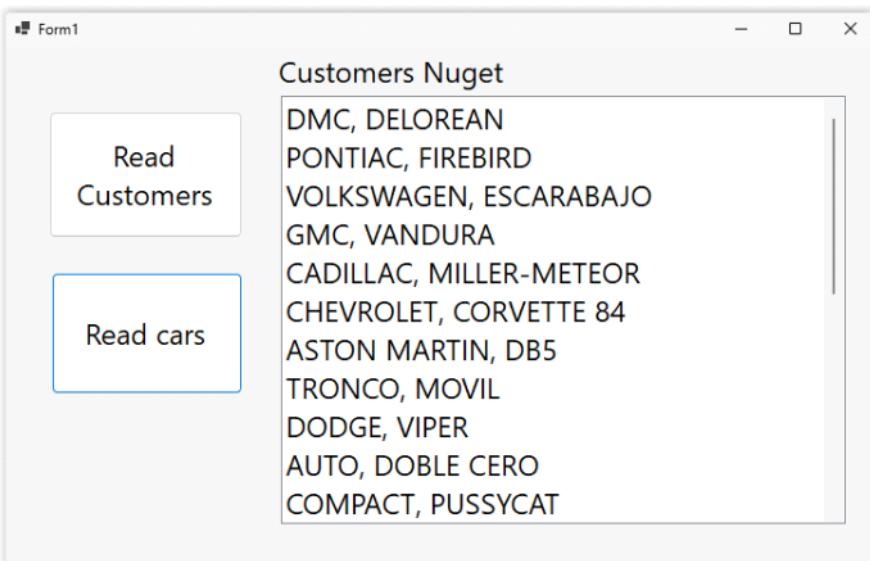
Para comprobar que es funcional, abrimos el último proyecto que tenemos de Windows Forms y lo probamos ahí mismo.



NugetCarsPGS by pacotajamar, 32 downloads  
Esto es la prueba de Assembly y Cars XML

1.0.0

```
private void btnReadCars_Click(object sender, EventArgs e)
{
    RepositoryCoches repo =
        new RepositoryCoches();
    List<Coche> coches =
        repo.GetCoches();
    this.lstCustomers.Items.Clear();
    foreach (Coche car in coches)
    {
        this.lstCustomers.Items.Add
            (car.Marca + ", " + car.Modelo);
    }
}
```



Vamos a realizar un nuevo Nuget para visualizar Aeropuertos

Debemos generar el Nuget y publicarlo y probarlo.

[https://services.odata.org/V4/\(S\(2esholowikwye30oggjzbvf\)\)/TripPinServiceRW/Airports](https://services.odata.org/V4/(S(2esholowikwye30oggjzbvf))/TripPinServiceRW/Airports)

NugetAeropuertosPGS

# WEB JOBS

miércoles, 19 de marzo de 2025 10:15

Esto es un trabajo Web, es decir, un código asociado a un App Service.  
Dicho código se puede ejecutar de forma continua o mediante un tiempo determinado  
Mediante CRON.

Está asociado a un App Service en Azure y depende completamente de dicho App Service.  
No podemos ejecutar el código desde otro lugar.

El código puede estar escrito en múltiples lenguajes lo que le ofrece una gran Versatilidad: C#, Python, Java, PowerShell o Bash

Este código se debe comprimir dentro de un **zip** y alojaremos dicho zip dentro Del App Service.

Vamos a realizar un proyecto de consola asociado a un App Service que haremos Posteriormente.

La aplicación de consola lo que hará será recuperar datos cada N tiempo  
La aplicación App Service mostrará dichos datos actualizados por el Web Job

<https://www.chollometro.com/>

Vamos a realizar una aplicación que leerá el servicio de noticias RSS de Chollometro

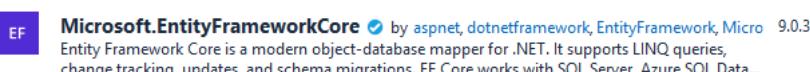
Un RSS es un estandar que se encarga de tener los datos actualizados para múltiples Web, es utilizado en periódicos y otras Web

Abrimos SQL Server de Azure.

Creamos una tabla para almacenar los chollos.

```
create table CHOLLOS
(
IDCHOLLO int primary key,
TITULO nvarchar(max),
LINK nvarchar(max),
DESCRIPCION nvarchar(max),
FECHA datetime)
```

Creamos un nuevo proyecto de consola llamado **WebJobChollometro**



Dentro de un proyecto de consola o de un proyecto de librerías, **no tenemos Inyección de dependencias**, es decir, no existe **Program/Container** para realizar Las inyecciones



Creamos una carpeta llamada **Models** y una clase llamada **Chollo**

**CHOLLO**

```

[Table("CHOLLOS")]
0 references
public class Chollo
{
    [Key]
    [Column("IDCHOLLO")]
    0 references
    public int IdChollo { get; set; }

    [Column("TITULO")]
    0 references
    public string Titulo { get; set; }

    [Column("LINK")]
    0 references
    public string Link { get; set; }

    [Column("DESCRIPCION")]
    0 references
    public string Descripcion { get; set; }

    [Column("FECHA")]
    0 references
    public DateTime Fecha { get; set; }
}

```

Creamos una carpeta llamada **Data** y una clase llamada **ChollometroContext**

**CHOLLOMETROCONTEXT**

```

public class ChollometroContext: DbContext
{
    0 references
    public ChollometroContext
        (DbContextOptions<ChollometroContext> options)
        : base(options){ }

    0 references
    public DbSet<Chollo> Chollos { get; set; }
}

```

Sobre el proyecto, creamos una nueva carpeta llamada **Repositories** y una clase  
Llamada **RepositoryChollometro**

Para poder leer de la Web, necesitamos "engaños" a la página que deseamos consumir  
Debemos pensar a la página que somos un Explorador Web, no un programa.

@"Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; Trident/6.0)"

**REPOSITORYCHOLLOMETRO**

```

public class RepositoryChollometro
{
    private ChollometroContext context;

    public RepositoryChollometro(ChollometroContext context)
    {
        this.context = context;
    }

    //LOS CHOLLOS LOS VAMOS A EXTRAER DE LA WEB
    //RECUPERAREMOS TODOS LO QUE TENGA DENTRO DEL RSS
    //IREMOS INCREMENTANDO CADA CHOLLO MEDIANTE UN MAX
    public async Task<int> GetMaxIdCholloAsync()
    {
        if (this.context.Chollos.Count() == 0)
        {
            return 1;
        }
        else
        {
            return await
                this.context.Chollos.MaxAsync(x => x.IdChollo) + 1;
        }
    }
}

```

```

//METODO PARA LEER LOS CHOLLOS DE LA WEB
public async Task<List<Chollo>> GetChollosWebAsync()
{
    //@"Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; Trident/6.0)"
    string url = "https://www.chollometro.com/rss";
    HttpWebRequest request = (HttpWebRequest)
        WebRequest.Create(url);
    request.Accept = @"text/html application/xhtml+xml, *.*";
    request.Host = "www.chollometro.com";
    request.Headers.Add("Accept-language", "es-ES");
    request.Referer = "https://www.chollometro.com";
    request.UserAgent = @"Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6
.2; Trident/6.0)";
    HttpWebResponse response = (HttpWebResponse)
        await request.GetResponseAsync();
    //ESTE TIPO DE PETICION SE PUEDE UTILIZAR PARA TODO,
    //PODEMOS LEER UN VIDEO, UNA IMAGEN O SIMPLE TEXTO
    //LA INFORMACION NOS LA DEVUELVE EN STREAM
    //DEBEMOS CONVERTIR EL STREAM EN TEXTO (xml)
    string xmlData = "";
    using (StreamReader reader =
        new StreamReader(response.GetResponseStream()))
    {
        xmlData = await reader.ReadToEndAsync();
    }
    //TENEMOS UN XML QUE PODEMOS LEER CON LINQ
    XDocument document = XDocument.Parse(xmlData);
    var consulta = from datos in document.Descendants("item")
                  select datos;
    List<Chollo> chollosWeb = new List<Chollo>();
    int idChollo = await this.GetMaxIdCholloAsync();
    foreach (var tag in consulta)
    {
        Chollo chollo = new Chollo();
        chollo.IdChollo = idChollo;
        chollo.Titulo = tag.Element("title").Value;
        chollo.Descripcion = tag.Element("description").Value;
        chollo.Link = tag.Element("link").Value;
        chollo.Fecha = DateTime.Now;
        idChollo += 1;
        chollosWeb.Add(chollo);
    }
    return chollosWeb;
}

//CREAMOS UN METODO PARA AGREGAR LOS CHOLLOS DE LA NUBE
//DENTRO DE LA BASE DE DATOS.
public async Task PopulateChollosAzureAsync()
{
    List<Chollo> chollos = await this.GetChollosWebAsync();
    foreach (Chollo c in chollos)
    {
        //LOS AGREGAMOS AL CONTEXT
        this.context.Chollos.Add(c);
    }
    await this.context.SaveChangesAsync();
}
}

```

## PROGRAM

```

using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using WebJobChollometro.Data;
using WebJobChollometro.Repositories;

Console.WriteLine("Bienvenido a nuestros Chollos!!!");
string connectionString = "Data Source=techriders.database.windows.net;Initial
Catalog=PRODUCCION;Persist Security Info=True;User ID=adminssql;Password=Admin
123;Encrypt=True;Trust Server Certificate=True";

//NECESITAMOS UTILIZAR INYECCION DE DEPENDENCIAS PARA PODER
//CREAR LOS OBJETOS.
var provider = new ServiceCollection()
    .AddTransient<RepositoryChollometro>()
    .AddDbContext<ChollometroContext>
    (options => options.UseSqlServer(connectionString))
    .BuildServiceProvider();
//DESDE ESTE CODIGO NECESITAMOS RECUPERAR EL REPO INYECTADO
RepositoryChollometro repo =
    provider.GetService<RepositoryChollometro>();
Console.WriteLine("Pulse Enter para Iniciar");
Console.ReadLine();
await repo.PopulateChollosAzureAsync();
Console.WriteLine("Proceso completado correctamente");
Console.WriteLine("Enhorabuena!!!");


```

Por ahora, dejamos aparcada la aplicación y es funcional.

Vamos a crear otra aplicación que será nuestro App Service y es dónde veremos los Chollo publicados y recuperados por nuestro Web Job

Sobre la solución, agregamos un nuevo proyecto de tipo Mvc Net Core llamado

**MvcNetCoreChollometro**

Agregamos los Nuget de Entity Framework

 EF	<b>Microsoft.EntityFrameworkCore</b> by Microsoft Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Data...	9.0.3
 EF	<b>Microsoft.EntityFrameworkCore.SqlServer</b> by Microsoft Microsoft SQL Server database provider for Entity Framework Core.	9.0.3

Publicamos nuestra Web de Chollometro en Azure.

Es el momento de asociar nuestro Web Job con App Service.

El Web Job debe ser autónomo, es decir, ejecutarse sin interrupciones.

Actualmente, tenemos interacción del usuario, es decir, tenemos que pulsar ENTER para que realice las acciones.

#### Quitamos los WriteLine y ReadLine

Para hacerlo, es necesario realizar un zip que contenga el Assembly del Web Job

Sobre el proyecto **Web Job** realizamos un ZIP con la carpeta **Debug** del **bin**

> ... Proyectos > WebJobChollometro > WebJobChollometro > bin >			
A	B	Ordenar	Ver
Nombre	Fecha de modificación	Tipo	Tamaño
Debug	19/03/2025 10:28	Carpeta de archivos	

Generamos un ZIP

> ... Proyectos > WebJobChollometro > WebJobChollometro > bin >			
A	B	Ordenar	Ver
Nombre	Fecha de modificación	Tipo	Tamaño
Debug	19/03/2025 10:28	Carpeta de archivos	
Release	19/03/2025 12:52	Carpeta de archivos	
Debug.zip	19/03/2025 13:01	Carpeta comprimi...	6.210 KB

Para crear un Web Job, necesitamos un App Service (lo tenemos) y un zip (lo tenemos)

Tenemos dos formas de utilizar el Web Job:

- 1) **Desencadenado (Trigger):** Se ejecuta cada N tiempo utilizando una tecnología Llamada CRON
- 2) **Continuo:** Se ejecuta constantemente.

La expresión CRON está compuesta por 6 elementos de tiempo:

{second}{minutes}{hours}{days}{months}{day of week}

Si queremos ejecutar cada 30 segundos:

0/30 \* \* \* \*

Si queremos cada 15 minutos

\* 0/15 \* \* \* \*

Abrimos Azure Portal y entramos en nuestro App Service de MVC

Home > rg-tajamar >

## MvcNetCoreChollometrops

Web App

Web

Browse Stop Swap Restart Delete Refresh Download public

Settings

- Environment variables
- Configuration
- WebJobs**
- Development Tools
- Advanced Tools
- Extensions
- Monitoring
- App Service logs

Essentials

Resource group (move)	: rg-tajamar	Default dc
Status	: Running	App Service
Location (move)	: Spain Central	Operating
Subscription (move)	: Azure for Students	Health Ch
Subscription ID	: 02ee10d4-2d35-43df-89fb-d7ac1e185646	
Tags (edit)	: Add tags	

Properties Monitoring Logs Capabilities Notifications (1) Recommendations

Web app

### Add WebJob

MvcNetCoreChollometrops

Name \*

File Upload \*

Type \*

Triggers \*

CRON Expression \*

**Info:** Scheduled WebJob will be executed based on provided CRON expression. Click here to learn more about CRON

**Create Webjob**

Y ya tendremos nuestro Web Job asociado al Service

+ Add

Basic authentication is disabled for your app. To configure settings and enable basic authentication [Click here](#)

WebJobs provide an easy way to run scripts or programs as background processes in the context of your app.

Name	Type	Status	Schedule	Logs	Run	D
WebJobChollometro	Triggered	Ready	* 0/15 * * * *			

## Web Job Chollometro

Número de chollos: 90

### Fibra 500MB gratis durante 1 año

<https://www.chollometro.com/ofertas/fibra-500mb-gratis-durante-1-ano-1492326>

Fecha: 3/19/2025 12:45:10 PM

Keio



Fibra gratis durante un año. Corre que vuelan. Registro sin tarjeta de crédito.

### Paleta 100% ibérica de bellota. Brida negra 5kg

<https://www.chollometro.com/ofertas/paleta-100-iberica-de-bellota-brida-negra-5kg-1489411>

Fecha: 3/19/2025 12:45:10 PM

## API SERVICES

viernes, 21 de marzo de 2025 9:12

Un servicio Api, también llamado **Restful** son servicios con formato texto (XML o JSON)

Una gran ventaja que tenemos con Net Core es que la serialización a los formatos Es automática, es decir, no tenemos que hacer nada, simplemente devolver las clases Y ya son transformadas a JSON, que es el formato por defecto

Los servicios Api también pueden tener seguridad y el tipo de seguridad puede ser Basic, OAuth2 o puede ser de fabricante (Third Party).

La documentación de las Apis está realizada con un standard llamado **OpenApi**, con dicho Estandar podemos documentar nuestro Api hacia otros servicios.

En Net Core, hasta ahora, teníamos un estándar de documentación llamado Swagger, pero Nos lo han quitado en la versión 9.

Lo que han hecho actualmente es que si yo quiero mostrar mi servicio documentado, ya no Viene por defecto, lo tengo que hacer, de esa forma, si alguien accede a mi Api, no verá nada.

Para consumir Api en Cliente debemos habilitar CORS y podemos hacerlo desde código C# o Desde un Api Service.

Tenemos 4 métodos dentro de cualquier Api:

- GET: Recuperar datos
- POST: Insertar o para seguridad
- PUT: Update
- DELETE: Borrar

Por defecto, en cualquier API, solamente podemos tener UNO de cada.

En Net Core, podemos utilizar actualmente solamente dos GET sin configurar nada.

Get  
Get(id)

Si quiero más métodos, necesito **Routing**

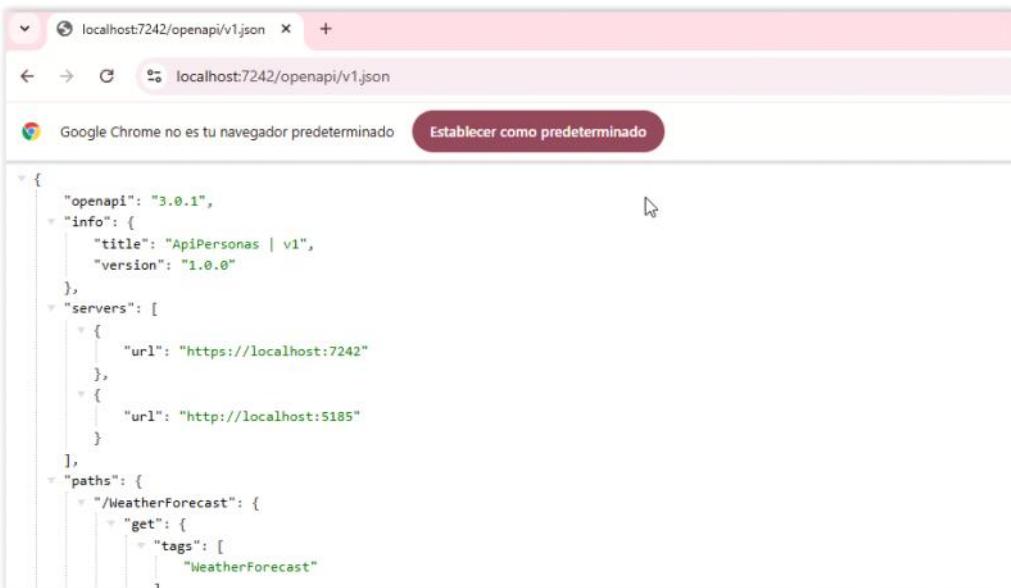


Vamos a crear un nuevo proyecto para mostrar datos de forma estática, mostraremos Datos de Personas.

Creamos un nuevo proyecto llamado **ApiPersonas**

Lo primero de todo será configurarlo con Swagger

Necesitamos averiguar la dirección de Open Api que es igual para todos los servicios



Con la URL de open api podemos incluir Swagger UI.

Para ello, necesitamos el siguiente Nuget



Con esto, ya podemos utilizar Swagger

```

app.UseHttpsRedirection();
app.UseSwaggerUI();
|
app.UseAuthorization();

```

Debemos incluir el EndPoint de OpenApi y ya tendremos Swagger documentado

```

app.UseSwaggerUI
    (options =>
        options.SwaggerEndpoint("/openapi/v1.json", "ApiPersonas"));

```

Creamos una nueva carpeta llamada **Models** y una clase llamada **Persona**

**PERSONA**

```

public class Persona
{
    public int IdPersona { get; set; }
    0 references
    public string Nombre { get; set; }
    0 references
    public string Email { get; set; }
    0 references
    public int Edad { get; set; }
}

```

Sobre **Controllers** creamos un nuevo controlador de tipo **Api Controller - Empty**  
Llamado **PersonasController**



Lo primero que veremos será la ruta para acceder a nuestro servicio API

```

[Route("api/[controller]")]
[ApiController]

public class PersonasController : ControllerBase
{
}

```

Para crear métodos necesitamos incluir la decoración **[HttpGet]** para indicar que será un método

## Get del Servicio

Se devuelve en los métodos `ActionResult<T>`

Creamos un método para devolver personas.

The screenshot shows the 'Server responses' section of a browser's developer tools. It displays a successful HTTP request (200) with the 'Response body' tab selected. The response content is a JSON array containing three objects, each representing a person with properties: idPersona, nombre, email, and edad. The JSON is as follows:

```
[{"idPersona": 1, "nombre": "Lucia", "email": "lucia@gmail.com", "edad": 22}, {"idPersona": 2, "nombre": "Adrian", "email": "adrian@gmail.com", "edad": 25}, {"idPersona": 3, "nombre": "Diana", "email": "diana@gmail.com", "edad": 42}]
```

The screenshot shows a browser window with the URL `localhost:7242/api/personas/1`. The page content is a single JSON object representing a person, identical to the one shown in the developer tools screenshot. The JSON is as follows:

```
{ "idPersona": 1, "nombre": "Lucia", "email": "lucia@gmail.com", "edad": 22 }
```

## PERSONASCONTROLLER

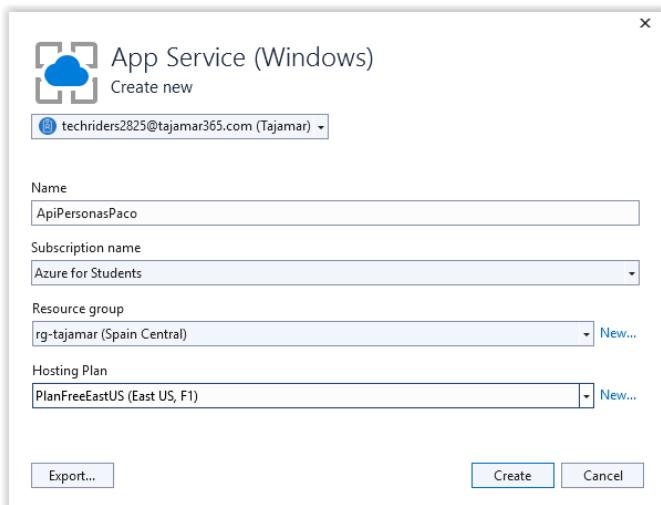
```
[Route("api/[controller]")]
[ApiController]
public class PersonasController : ControllerBase
{
    private List<Persona> personas;

    public PersonasController()
    {
        this.personas = new List<Persona>();
        Persona p = new Persona
        {
            IdPersona = 1,
            Nombre = "Lucia",
            Email = "lucia@gmail.com",
            Edad = 22
        };
        this.personas.Add(p);
        p = new Persona
        {
            IdPersona = 2,
            Nombre = "Adrian",
            Email = "adrian@gmail.com",
            Edad = 25
        };
        this.personas.Add(p);
        p = new Persona
        {
            IdPersona = 3,
            Nombre = "Diana",
            Email = "diana@gmail.com",
            Edad = 42
        };
        this.personas.Add(p);
        p = new Persona
        {
            IdPersona = 4,
            Nombre = "Carmen",
            Email = "carmen@gmail.com",
            Edad = 29
        };
        this.personas.Add(p);
    }

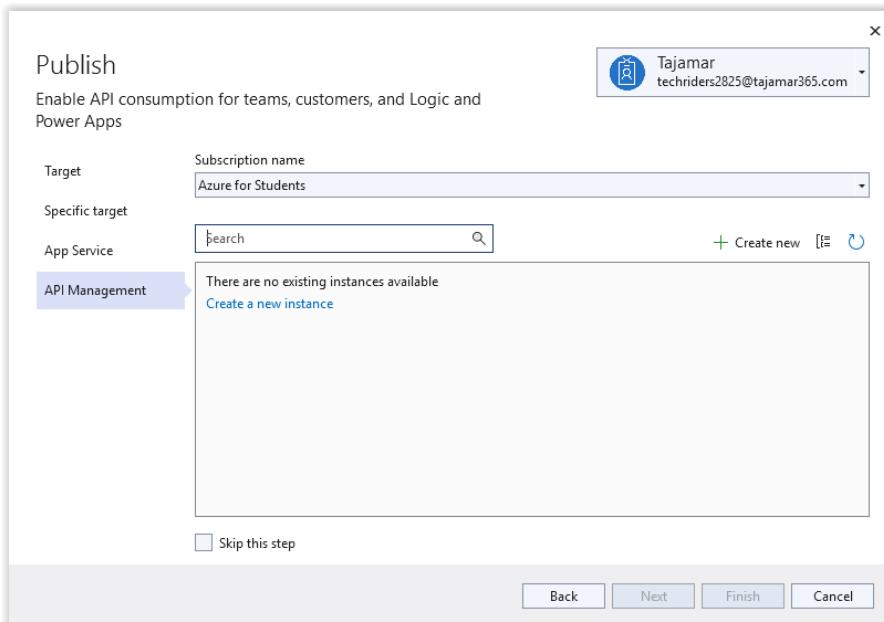
    [HttpGet]
    public ActionResult<List<Persona>> Get()
    {
        return this.personas;
    }

    [HttpGet("{id}")]
    public ActionResult<Persona> Get(int id)
    {
        return this.personas
            .FirstOrDefault(x => x.IdPersona == id);
    }
}
```

Vamos a publicar nuestro Api en Azure y ver si todo sigue siendo funcional

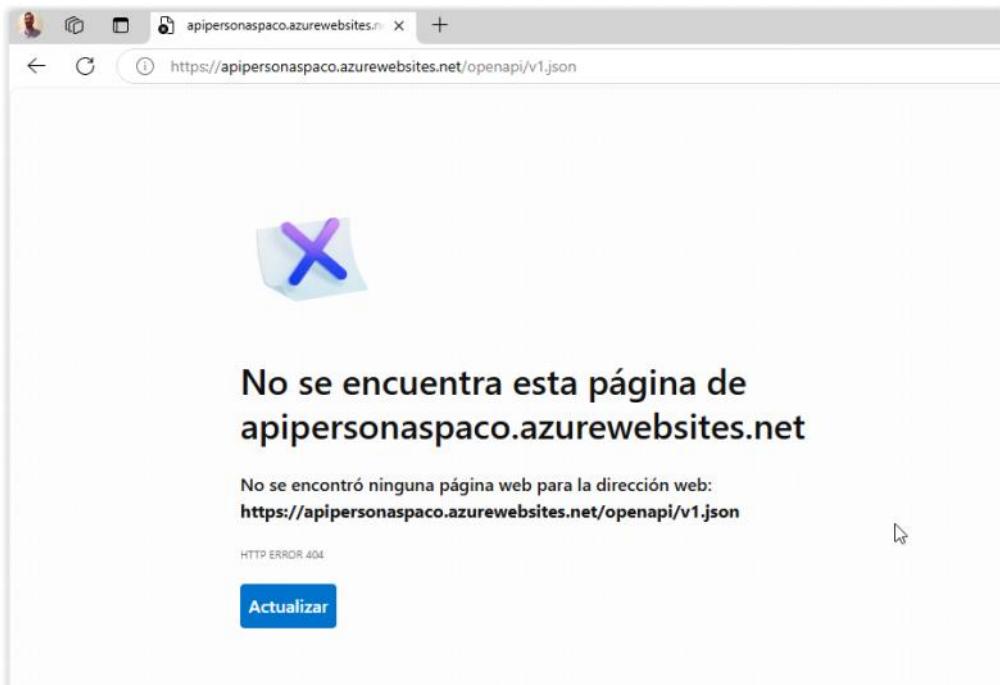


Los servicios API tienen un paso más que es API Management, pero como no sabemos nada  
De esto y, además, cuesta dinero y esfuerzo crearlo, ponemos Skip this step



Sólo están abiertos los servicios de documentación si estamos en Desarrollo

```
// Configure the environment properties
if (app.Environment.IsDevelopment())
{
    app.MapOpenApi();
}
```



Sacamos la línea del IF y ya tendremos swagger en Azure

Vamos a consumir nuestro servicio con un simple Jquery

Creamos un nuevo proyecto de tipo Mvc Net Core llamado **MvcCoreApiClient**

Trabajamos sobre **Home/Index**

**INDEX.CSHTML**

```

@{
    ViewData["Title"] = "Home Page";
}

@section Scripts {
    <script>
        $(document).ready(function() {
            $("#botonload").click(function(){
                var url = "https://apipersonaspaco.azurewebsites.net/api/personas";
                $.ajax({
                    url: url,
                    method: "GET",
                    success: function(data){
                        var html = "";
                        $.each(data, function(index, persona){
                            html += "<tr>";
                            html += "<td>" + persona.nombre + "</td>";
                            html += "<td>" + persona.email + "</td>";
                            html += "<td>" + persona.edad + "</td>";
                            html += "</tr>";
                        })
                        $("#tablaperonas tbody").html(html)
                    }
                })
            })
        })
    </script>
}

<div class="text-center">
    <h1 class="display-4">Personas</h1>
    <button id="botonload">
        Load Personas
    </button>
    <table class="table table-bordered" id="tablaperonas">
        <thead>

```

```

<tr>
  <th>Nombre</th>
  <th>Email</th>
  <th>Edad</th>
</tr>
</thead>
<tbody></tbody>
</table>
</div>

```

Y ya podremos probar el servicio y comprobar que no funciona, es lógico...

```

① Access to XMLHttpRequest at 'localhost:1
https://apipersonaspaco.azurewebsites.net/api/personas' from origin 'localhost:3001' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
② ► GET https://apipersonaspaco.azurewebsites.net/api/personas net::ERR_FAILED 200
(OK)

```

Debemos habilitar CORS dentro del servicio.

Vamos a utilizar la posibilidad de hacerlo dentro de Azure y su App Service

ApiPersonasPaco | CORS

CORS

Cross-Origin Resource Sharing (CORS) allows JavaScript code running in a browser on an external host to interact with your backend. Specify the origins that should be allowed to make cross-origin calls (for example: http://example.com:12345). To allow all, use "\*" and remove all other origins from the list. Slashes are not allowed as part of domain or after TLD. [Learn more](#)

Request Credentials

Enable Access-Control-Allow-Credentials

Allowed Origins

*	✓	✖	...
---	---	---	-----

Y ya podremos ver los datos

MvcCoreApiClient Home Privacy

## Personas

Load Personas

Nombre	Email	Edad
Lucia	lucia@gmail.com	22
Adrian	adrian@gmail.com	25
Diana	diana@gmail.com	42
Carmen	carmen@gmail.com	29

Vamos a realizar otro Api, pero esta vez, lo haremos con BBDD

Consumiremos datos de HOSPITAL de Azure.

Creamos un nuevo Api llamado **ApiCoreHospitales**

EF Microsoft.EntityFrameworkCore 9.0.3 by aspnet, dotnetframework, EntityFram 9.0.3  
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with...

EF Microsoft.EntityFrameworkCore.SqlServer 9.0.3 by aspnet, dotnetframewo 9.0.3  
Microsoft SQL Server database provider for Entity Framework Core.

Creamos una nueva carpeta llamada **Models** y una clase llamada **Hospital**

```

[HTable("HOSPITAL")]
public class Hospital
{
    [Key]
    [Column("HOSPITAL_COD")]
    public int IdHospital { get; set; }
    [Column("NOMBRE")]
    public string Nombre { get; set; }
}

```

```

    [Column("DIRECCION")]
    public string Direccion { get; set; }
    [Column("TELEFONO")]
    public string Telefono { get; set; }
    [Column("NUM_CAMA")]
    public int Camas { get; set; }
}

```

Creamos una carpeta llamada **Data** y una clase llamada **HospitalContext**

#### HOSPITALCONTEXT

```

public class HospitalContext : DbContext
{
    0 references
    public HospitalContext(DbContextOptions<HospitalContext>
        options): base(options) { }

    0 references
    public DbSet<Hospital> Hospitales { get; set; }
}

```

Creamos una carpeta llamada **Repositories** y una clase llamada **RepositoryHospital**

#### REPOSITORYHOSPITAL

```

public class RepositoryHospital
{
    private HospitalContext context;

    public RepositoryHospital(HospitalContext context)
    {
        this.context = context;
    }

    public async Task<List<Hospital>> GetHospitalesAsync()
    {
        return await this.context.Hospitales.ToListAsync();
    }

    public async Task<Hospital> FindHospitalAsync(int idHospital)
    {
        return await this.context.Hospitales
            .FirstOrDefaultAsync(x => x.IdHospital == idHospital);
    }
}

```

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



#### HOSPITALESCONTROLLER

```

[Route("api/[controller]")]
[ApiController]
public class HospitalesController : ControllerBase
{
    private RepositoryHospital repo;

    public HospitalesController(RepositoryHospital repo)
    {
        this.repo = repo;
    }

    [HttpGet]
    public async Task<ActionResult<List<Hospital>>>
        GetHospitales()
    {
        return await this.repo.GetHospitalesAsync();
    }

    [HttpGet("{id}")]
    public async Task<ActionResult<Hospital>> FindHospital(int id)
    {
        return await this.repo.FindHospitalAsync(id);
    }
}

```

Incluimos nuestra cadena de conexión a Azure dentro de **appsettings.json**.

#### APPSETTINGS.JSON

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "SqlAzure": "Data Source=techriders.database.windows.net;Initial Catalog=HospitalDB;User ID=sa;Password=12345678;Trusted_Connection=False;Encrypt=True;Connect Timeout=30"
  }
}
```

El siguiente paso es configurar Swagger y la inyección de dependencias.

 **Swashbuckle.AspNetCore** by domaindrivendev, 698M downloads      8.0.0  
Swagger tools for documenting APIs built on ASP.NET Core

Configuramos todo en **Program**

PROGRAM

```
// Add services to the container.
builder.Services.AddTransient<RepositoryHospital>();
string connectionString =
  builder.Configuration.GetConnectionString("SqlAzure");
builder.Services.AddDbContext<HospitalContext>
  (options => options.UseSqlServer(connectionString));

app.MapOpenApi();
app.UseHttpsRedirection();
app.UseSwaggerUI(options => options.SwaggerEndpoint("/openapi/v1.json"
  , "Api Hospitales 2025"));
```

También podemos hacer que nos aparezca la página de Swagger por defecto.

Necesitamos incluir algo más dentro de **SwaggerUI**

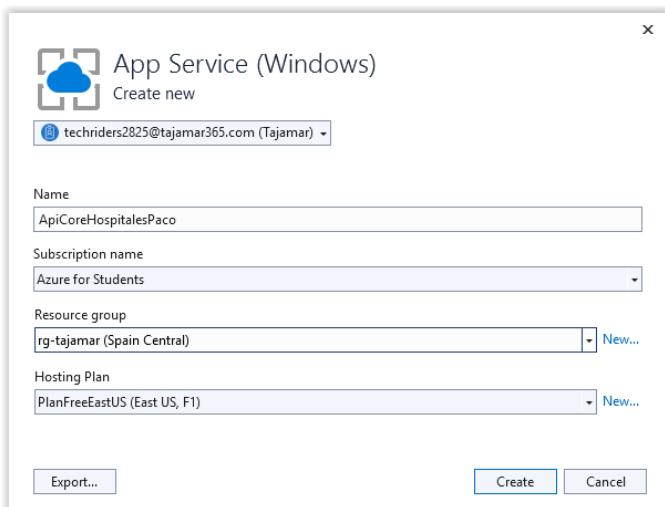
```
app.UseSwaggerUI(options =>
{
  options.SwaggerEndpoint("/openapi/v1.json", "Api Hospitales 2025");
  options.RoutePrefix = "";
});
```

Debemos abrir la carpeta **Properties** y el fichero **launchsettings.json**

Y ponemos **launchbrowser** a **true**

```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5164",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "https": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "applicationUrl": "https://localhost:5164"
    }
  }
}
```

Publicamos en Azure y habilitamos CORS



El siguiente paso es comprobar que es funcional todo nuestro servicio.  
Vamos a consumirlo de dos formas posibles: FRONT y BACK.

Abrimos el proyecto **MvcCoreApiClient**

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

**HOSPITALESCONTROLLER**

```
public class HospitalesController : Controller
{
  0 references | 0 changes | 0 authors, 0 changes
  public IActionResult Index()
  {
    return View();
  }

  0 references | 0 changes | 0 authors, 0 changes
  public IActionResult Cliente()
  {
    return View();
  }
}
```

Creamos la página **Cliente.cshtml**

**CLIENTE.CSHTML**

```
@section Scripts{
<script>
$(document).ready(function() { ... })
```

```

$( "#botonhospitales" ).click(function(){
    var url = "https://apicorehospitalespaco.azurewebsites.net/api/hospitales";
    $.ajax({
        url: url,
        method: "GET",
        success: function(data){
            let html = "";
            $.each(data, function(index, hospital){
                html += "<tr>";
                html += "<td>" + hospital.nombre + "</td>";
                html += "<td>" + hospital.direccion + "</td>";
                html += "<td>" + hospital.telefono + "</td>";
                html += "<td>" + hospital.camas + "</td>";
                html += "</tr>";
            })
            $("#tablahospitales tbody").html(html);
        }
    })
})
</script>

```

**Y podremos comprobar la funcionalidad**

Nombre	Dirección	Teléfono	Camas
Provincial	O' Donell 50	964-4256	502
General	Atocha s/n	595-3111	987
La Paz	Castellana 1000	923-5411	412
San Carlos	Ciudad Universitaria	597-1500	845
Ruber	Juan Bravo, 49	91-4027100	217

## API CONSUMO SERVIDOR

Necesitamos aprender a leer un Api con código C#

Elementos necesarios:

- Model/s** de los datos que vamos a mapear del Api  
Los models pueden estar decorados mediante **[JsonProperty]** si utilizamos Como serializador Newtonsoft JSON
- Service** para poder leer los datos del Servicio Api
- Nuggets**: Tenemos que agregar algunos Nuget nuevos para poder leer el servicio

Agregamos los siguientes Nugget

**Microsoft.AspNet.WebApi.Client** by aspnet, Microsoft, 692M downloads  
This package adds support for formatting and content negotiation to System.Net.Http.

**Newtonsoft.Json** by dotnetfoundation, jamesnk, newtonsoft, 5,97B downloads  
Json.NET is a popular high-performance JSON framework for .NET

Debemos crear un Model que tenga el mismo nombre que las propiedades del Json

```
[  
 {  
   "idHospital": 0,  
   "nombre": "string",  
   "direccion": "string",  
   "telefono": "string",  
   "camas": 0  
 }  
]
```

Decoramos el Model mediante [JsonProperty] si las propiedades no se llamasen igual.

Sobre Models creamos una nueva clase llamada Hospital

HOSPITAL

```
public class Hospital  
{  
  [JsonProperty("idHospital")]  
  0 references | 0 changes | 0 authors, 0 changes  
  public int IdHospital { get; set; }  
  [JsonProperty("nombre")]  
  0 references | 0 changes | 0 authors, 0 changes  
  public string Nombre { get; set; }  
  [JsonProperty("direccion")]  
  0 references | 0 changes | 0 authors, 0 changes  
  public string Direccion { get; set; }  
  [JsonProperty("telefono")]  
  0 references | 0 changes | 0 authors, 0 changes  
  public string Telefono { get; set; }  
  [JsonProperty("camas")]  
  0 references | 0 changes | 0 authors, 0 changes  
  public int Camas { get; set; }  
}
```

Creamos una nueva carpeta llamada Services y una clase llamada ServiceHospitales

SERVICEHOSPITALES

```
public class ServiceHospitales  
{  
  //NECESITAMOS LA URL DE ACCESO AL SERVICIO, SOLO LA URL  
  private string ApiUrl;  
  //NECESITAMOS INDICAR A NUESTRO SERVICE QUE LEEMOS  
  //CODIGO JSON  
  private MediaTypeWithQualityHeaderValue header;  
  
  public ServiceHospitales()  
  {  
    this.ApiUrl = "https://apicorehospitalespaco.azurewebsites.net/";  
    this.header = new MediaTypeWithQualityHeaderValue  
      ("application/json");  
  }  
  
  //CREAMOS UN METODO ASINCRONO PARA LEER LOS HOSPITALES  
  public async Task<List<Hospital>> GetHospitalesAsync()  
  {  
    //SE UTILIZA LA CLASE HttpClient PARA LAS PETICIONES  
    //AL SERVIDOR  
    using (HttpClient client = new HttpClient())  
    {  
      //NECESITAMOS UNA PETICION  
      string request = "api/hospitales";  
      //INDICAMOS LA URL BASE PARA ACCEDER AL SERVICIO API  
      client.BaseAddress = new Uri(this.ApiUrl);  
      //COMO ES POSIBLE QUE SE CRUCEN PETICIONES ENTRE  
      //METODOS CON DISTINTAS INFORMACIONES, DEBEMOS  
      //LIMPIAR LOS HEADER COMO NORMA  
      client.DefaultRequestHeaders.Clear();  
      //CREAMOS UN NUEVO Header PARA INDICAR QUE  
      //LEEREMOS JSON  
      client.DefaultRequestHeaders.Accept.Add(this.header);  
      //HACEMOS LA PETICION AL SERVICIO (GET) Y  
      //CAPTURAMOS LA RESPUESTA  
      HttpResponseMessage response =  
        await client.GetAsync(request);  
      //EN LA RESPUESTA, SE OFRECEN DISTINTOS STATUS CODE  
      if (response.IsSuccessStatusCode)  
      {  
        //DESCARGAMOS EL JSON COMO STRING  
        string json = await  
          response.Content.ReadAsStringAsync();  
        //UTILIZAMOS NEWTON PARA RECUPERAR LOS DATOS  
        //SERIALIZADOS DE JSON A List<Hospital>  
        List<Hospital> data =  
          JsonConvert.DeserializeObject<List<Hospital>>  
            (json);  
        return data;  
      }  
      else  
      {  
        return null;  
      }  
    }  
  }  
}
```

Resolvemos las dependencias en Program

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddTransient<ServiceHospitales>();
builder.Services.AddControllersWithViews();
```

Inyectamos el Service dentro de **HospitalesController** y creamos un nuevo método. Para acceder a los datos de Hospital del Api.

HOSPITALESCONTROLLER

```
public class HospitalesController : Controller
{
    private ServiceHospitales service;

    0 references | 0 changes | 0 authors, 0 changes
    public ...HospitalesController(ServiceHospitales service)
    {
        this.service = service;
    }

    0 references | 0 changes | 0 authors, 0 changes
    public async Task<IActionResult> Servidor()
    {
        List<Hospital> hospitales =
            await this.service.GetHospitalesAsync();
        return View(hospitales);
    }
}
```

Y ya tendremos la aplicación corriendo y funcional



Home Hospitales Privacy

## Api Servidor

IdHospital	Nombre	Direccion	Telefono	Cama
19	Provincial	O' Donell 50	964-4256	502
18	General	Atocha s/n	595-3111	987
22	La Paz	Castellana 1000	923-5411	412
45	San Carlos	Ciudad Univeritaria	597-1500	845
17	Ruber	Juan Bravo, 49	91-4027100	217

A continuación, vamos a leer el método en Servidor de detalles de un Hospital. No recuperaremos la Url del Api desde el propio constructor del Servicio, sino que Tendremos nuestra Url del Api inyectada dentro del constructor.

Incluimos la Url del Api dentro de **appsettings.json**

```

schemasstore.org/appsettings.json
{
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    },
    "AllowedHosts": "*",
    "ApiUrls": {
        "ApiHospitales": "https://apicorehospitalespaco.azurewebsites.net/"
    }
}

```

Modificamos el constructor del Service para recibir IConfiguration

#### SERVICEHOSPITAL

```

public ServiceHospitales(IConfiguration configuration)
{
    this.ApiUrl =
        configuration.GetValue<string>("ApiUrls:ApiHospitales");
    this.header = new MediaTypeWithQualityHeaderValue
        ("application/json");
}

```

Creamos un nuevo método dentro del servicio para leer los detalles

```

public async Task<Hospital> FindHospitalAsync(int idHospital)
{
    using (HttpClient client = new HttpClient())
    {
        string request = "api/hospitales/" + idHospital;
        client.BaseAddress = new Uri(this.ApiUrl);
        client.DefaultRequestHeaders.Clear();
        client.DefaultRequestHeaders.Accept.Add(this.header);
        HttpResponseMessage response =
            await client.GetAsync(request);
        if (response.IsSuccessStatusCode)
        {
            //SI LAS PROPIEDADES SE LLAMAN IGUAL A LA
            //LECTURA DE JSON, NO ES NECESARIO MAPEAR
            //CON LA DECORACION [JsonProperty] Y NO
            //LEEREMOS CON Newtonsoft.
            Hospital data =
                await response.Content.ReadAsAsync<Hospital>();
            return data;
        }
        else
        {
            return null;
        }
    }
}

```

Implementamos el método dentro del Controller y creamos una nueva vista.

#### HOSPITALESCONTROLLER

```

public async Task<IActionResult> Details(int id)
{
    Hospital hospital =
        await this.service.FindHospitalAsync(id);
    return View(hospital);
}

```

En el siguiente ejemplo lo que haremos será crear un Api con múltiples rutas de acceso para Get y veremos cómo podemos mapearlas.

Tendremos una clase llamada **Empleado** con algunos datos del empleado

- IdEmpleado, Apellido, Oficio, Salario, IdDepartamento

Comenzamos por Get() y Get{id} y cuando lo tengamos, seguimos probando.

Creamos un nuevo proyecto Api llamado **ApiEmpleadosMultiplesRutas**



 Microsoft.EntityFrameworkCore.SqlServer by aspnet, dotnetframework 9.0.3  
Microsoft SQL Server database provider for Entity Framework Core.

 Swashbuckle.AspNetCore by domaindrivendev, 699M downloads 8.0.0  
Swagger tools for documenting APIs built on ASP.NET Core

#### EMPLEADO

```
[Table("EMP")]
public class Empleado
{
    [Key]
    [Column("EMP_NO")]
    public int IdEmpleado { get; set; }
    [Column("APELLIDO")]
    public string Apellido { get; set; }
    [Column("OFICIO")]
    public string Oficio { get; set; }
    [Column("SALARIO")]
    public int Salario { get; set; }
    [Column("DEPT_NO")]
    public int Departamento { get; set; }
}
```

Creamos una carpeta llamada Data y una clase llamada EmpleadosContext

#### EMPLEADOSCONTEXT

```
public class EmpleadosContext : DbContext
{
    0 references
    public EmpleadosContext(DbContextOptions<EmpleadosContext>
        options) : base(options) { }

    0 references
    public DbSet<Empleado> Empleados { get; set; }
}
```

#### REPOSITORYEMPLEADOS

```
public class RepositoryEmpleados
{
    private EmpleadosContext context;

    public RepositoryEmpleados(EmpleadosContext context)
    {
        this.context = context;
    }

    public async Task<List<Empleado>> GetEmpleadosAsync()
    {
        return await this.context.Empleados.ToListAsync();
    }

    public async Task<Empleado> FindEmpleadoAsync
        (int idEmpleado)
    {
        return await this.context.Empleados
            .FirstOrDefaultAsync(z => z.IdEmpleado == idEmpleado);
    }
}
```

#### APPSETTINGS.JSON

```
store.org/appsettings.json
{
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    },
    "AllowedHosts": "*",
    "ConnectionStrings": {
        "SqlAzure": "Data Source=techriders.database.windows.net,1433;Initial Catalog=Techriders;User ID=sa;Password=Techriders@123;Encrypt=True;Trusted_Connection=False;MultipleActiveResultSets=True"
    }
}
```

#### PROGRAM

```
using ApiEmpleadosMultiplesRutas.Data;
using ApiEmpleadosMultiplesRutas.Repositories;
using Microsoft.EntityFrameworkCore;
```

```

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
string connectionString =
    builder.Configuration.GetConnectionString("SqlAzure");
builder.Services.AddTransient<RepositoryEmpleados>();
builder.Services.AddDbContext<EmpleadosContext>
    (options => options.UseSqlServer(connectionString));
builder.Services.AddControllers();
// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi
builder.Services.AddOpenApi();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
}

app.MapOpenApi();
app.UseHttpsRedirection();
app.UseSwaggerUI(options =>
{
    options.SwaggerEndpoint("/openapi/v1.json", "Api Empleados Routes");
    options.RoutePrefix = "";
});

app.UseAuthorization();
app.MapControllers();
app.Run();

```

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

#### EMPLEADOSCONTROLLER

```

[Route("api/[controller]")]
[ApiController]
public class EmpleadosController : ControllerBase
{
    private RepositoryEmpleados repo;

    public EmpleadosController(RepositoryEmpleados repo)
    {
        this.repo = repo;
    }

    [HttpGet]
    public async Task<ActionResult<List<Empleado>>> GetEmpleados()
    {
        return await this.repo.GetEmpleadosAsync();
    }

    [HttpGet("{id}")]
    public async Task<ActionResult<Empleado>> FindEmpleado
        (int id)
    {
        return await this.repo.FindEmpleadoAsync(id);
    }
}

```

Una vez que es funcional, es el momento de crear más métodos en el servicio para Aprender a Mapearlos con Routing.

Agregamos algunos métodos de búsqueda sobre **RepositoryEmpleados**

#### REPOSITORYEMPLEADOS

Para la acción de tener método extra NO importa el tipo de método del Api, es decir, se Realiza igual tanto si es un GET o un POST o un DELETE  
Para poder mapear utilizamos la decoración **[Route]**  
Dicha decoración contiene dos definiciones:

- **[action]:** Indica el nombre de la acción del método en la definición del Api  
Si incluimos esta palabra clave, el método del API se llamará igual al nombre Del método del Controller.
- **{parámetros}:** Indica el nombre del parámetro o parámetros que recibiremos Dentro de la acción. Los nombres de parámetro deben llamarse exactamente igual A como se llaman en la definición del método.

#### EMPLEADOSCONTROLLER

Empleados	
GET	/api/Empleados
GET	/api/Empleados/{id}
GET	/api/Empleados/Oficios
GET	/api/Empleados/EmpleadosOficio/{oficio}
GET	/api/Empleados/EmpleadosSalarioDepartamento/{salario}/{iddepartamento}

El siguiente paso es consumir nuestro servicio mediante un Client.

¿Qué necesitamos dentro del Client? Repetir los Models y Web API Client.

Lo que vamos a realizar es una réplica de los modelos dentro de un proyecto Nuget y

Agregaremos dicho Nuget a cada proyecto.

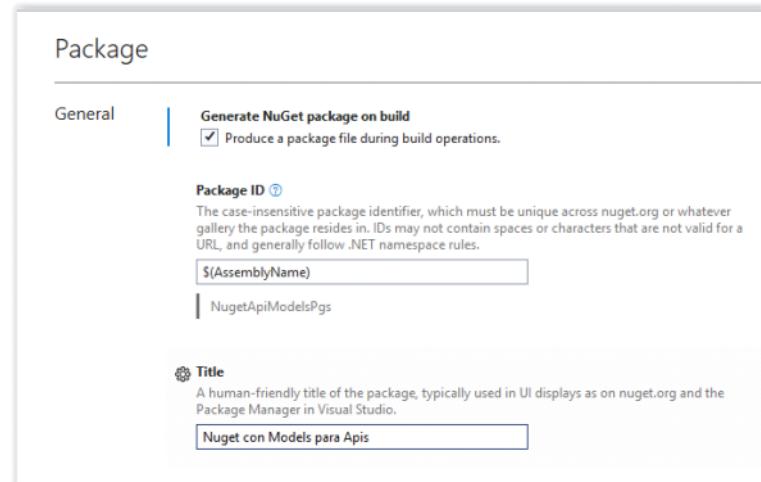
Creamos un nuevo proyecto llamado **NugetApiModels**

Agregamos el Nuget de Entity Core

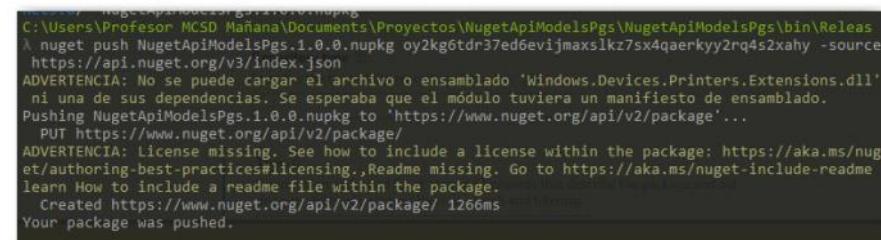


## EMPLEADO

```
[Table("EMP")]
public class Empleado
{
    [Key]
    [Column("EMP_NO")]
    public int IdEmpleado { get; set; }
    [Column("APELLIDO")]
    public string Apellido { get; set; }
    [Column("OFICIO")]
    public string Oficio { get; set; }
    [Column("SALARIO")]
    public int Salario { get; set; }
    [Column("DEPT_NO")]
    public int Departamento { get; set; }
}
```



Y subimos nuestro Nuget al servidor **nuget.org**



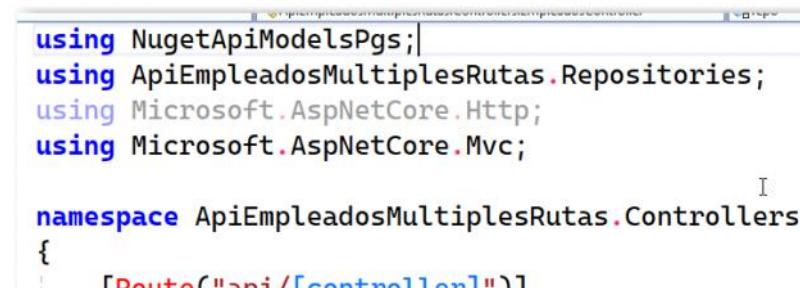
Y esperamos a que esté publicado



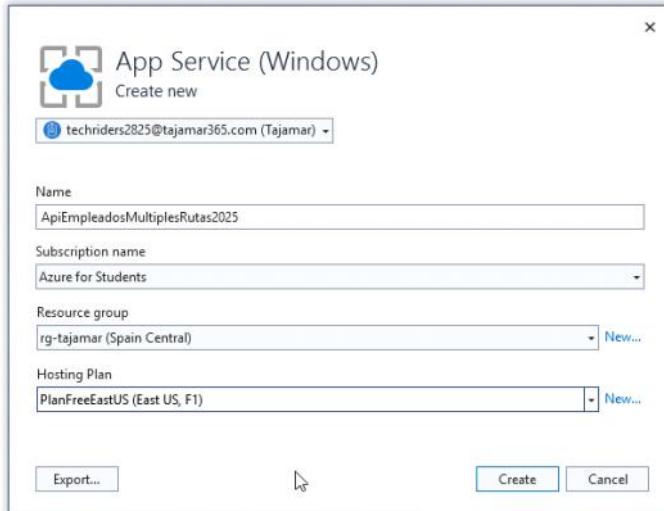
Abrimos el proyecto del Api y eliminamos la clase **Empleado**, agregando nuestro nuevo Nuget



Resolvemos los nuevos using



Y publicamos el servicio en Azure



Una vez que tenemos el API publicado, vamos a crear otro proyecto llamado **MvcCoreEmpleadosMultipleRutas**

Agregamos el Nuget de **Web Api Client** y nuestro nuevo Nuget

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

#### SERVICEEMPLEADOS

```
public class ServiceEmpleados
{
    private string ApiUrl;
    private MediaTypeWithQualityHeaderValue Header;

    public ServiceEmpleados(IConfiguration configuration)
    {
        this.ApiUrl =
            configuration.GetValue<string>("ApiUrls:ApiEmpleados");
        this.Header = new
            MediaTypeWithQualityHeaderValue("application/json");
    }

    public async Task<List<Empleado>> GetEmpleadosAsync()
    {
        using (HttpClient client = new HttpClient())
        {
            string request = "api/empleados";
            client.BaseAddress = new Uri(this.ApiUrl);
            client.DefaultRequestHeaders.Clear();
            client.DefaultRequestHeaders.Accept.Add(this.Header);
            HttpResponseMessage response = await
                client.GetAsync(request);
            if (response.IsSuccessStatusCode)
            {
                List<Empleado> data = await
                    response.Content.ReadAsAsync<List<Empleado>>();
                return data;
            }
            else
            {
                return null;
            }
        }
    }

    public async Task<List<string>> GetOficiosAsync()
    {
        using (HttpClient client = new HttpClient())
        {
            string request = "api/empleados/oficios";
            client.BaseAddress = new Uri(this.ApiUrl);
            client.DefaultRequestHeaders.Clear();
            client.DefaultRequestHeaders.Accept.Add(this.Header);
            HttpResponseMessage response = await
                client.GetAsync(request);
            if (response.IsSuccessStatusCode)
            {
                List<string> data = await response.Content
                    .ReadAsAsync<List<string>>();
                return data;
            }
            else
            {
                return null;
            }
        }
    }
}
```

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

#### EMPLEADOSCONTROLLER

Sobre `_ViewImports` agregamos el using a nuestro nuevo Nuget



```
1 @using MvcCoreEmpleadosMultipleRutas
2 @using MvcCoreEmpleadosMultipleRutas.Models
3 @using NuGetApiModelsPgs;
4 @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Y veremos que hemos perdido Scaffolding en cuanto utilizamos Nuget

#### INDEX.CSHTML

```
@model List<Empleado>
@{
    List<string> oficios =
        ViewData["OFICIOS"] as List<string>;
}
<h1>Api Empleados Nuget Routes</h1>
<form method="post">
    <label>Seleccione un oficio</label>
    <select name="oficio" class="form-control">
        @foreach (string ofi in oficios){
            <option value="@ofi">@ofi</option>
        }
    </select>
</form>
<table class="table table-bordered">
    <thead>
        <tr>
            <th>Apellido</th>
            <th>Oficio</th>
            <th>/</th>
        </tr>
    </thead>
    <tbody>
        @foreach (Empleado empleado in Model){
            <tr>
                <td>@empleado.Apellido</td>
                <td>@empleado.Oficio</td>
            </tr>
        }
    </tbody>
</table>
```

Sobre `appsettings.json` incluimos el acceso a nuestro Api

#### APPSETTINGS.JSON



```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ApiUrls": {
    "ApiEmpleados": "https://apiempleadosmultiplerutas2025.a"
  }
}
```

Resolvemos dependencias en `Program`

#### PROGRAM

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddTransient<ServiceEmpleados>();
builder.Services.AddControllersWithViews();
```

Ya tenemos montado el servicio para acceder a los datos.  
Si nos fijamos, ¿Qué diferencia tenemos entre estos dos métodos del Servicio?

```

public async Task<List<Empleado>>
    GetEmpleadosAsync()
{
    using (HttpClient client = new HttpClient())
    {
        string request = "api/empleados";
        client.BaseAddress = new Uri(this.ApiUrl);
        client.DefaultRequestHeaders.Clear();
        client.DefaultRequestHeaders.Accept.Add(this.Header);
        HttpResponseMessage response = await
            client.GetAsync(request);
        if (response.IsSuccessStatusCode)
        {
            List<Empleado> data = await
                response.Content.ReadAsAsync<List<Empleado>>();
            return data;
        }
        else
        {
            return null;
        }
    }
}

public async Task<List<string>> GetOficiosAsync()
{
    using (HttpClient client = new HttpClient())
    {
        string request = "api/empleados/oficios";
        client.BaseAddress = new Uri(this.ApiUrl);
        client.DefaultRequestHeaders.Clear();
        client.DefaultRequestHeaders.Accept.Add(this.Header);
        HttpResponseMessage response = await
            client.GetAsync(request);
        if (response.IsSuccessStatusCode)
        {
            List<string> data = await response.Content
                .ReadAsAsync<List<string>>();
            return data;
        }
        else
        {
            return null;
        }
    }
}

```

Las diferencias son:

- Request es distinto
- Los datos del return son distintos.

En lugar de crear 20 métodos con un código exacto, deberíamos tener un solo método Genérico que devuelva T y que reciba el request.

Vamos a crear un método genérico en el servicio y realizar las N llamadas a cada petición Del Api.

METODO GENERICO <T>

```

private async Task<T> CallApiAsync<T>(string request)
{
    using (HttpClient client = new HttpClient())
    {
        client.BaseAddress = new Uri(this.ApiUrl);
        client.DefaultRequestHeaders.Clear();
        Client.DefaultRequestHeaders.Accept.Add(this.Header);
        HttpResponseMessage response = await
            client.GetAsync(request);
        if (response.IsSuccessStatusCode)
        {
            T data = await response.Content.ReadAsAsync<T>();
            return data;
        }
        else
        {
            return default(T);
        }
    }
}

```

#### SERVICEEMPLEADOS

```

public class ServiceEmpleados
{
    private string ApiUrl;

    private MediaTypeWithQualityHeaderValue Header;

    public ServiceEmpleados(IConfiguration configuration)
    {
        this.ApiUrl =
            configuration.GetValue<string>("ApiUrls:ApiEmpleados");
        this.Header = new
            MediaTypeWithQualityHeaderValue("application/json");
    }

    private async Task<T> CallApiAsync<T>(string request)
    {
        using (HttpClient client = new HttpClient())
        {
            client.BaseAddress = new Uri(this.ApiUrl);
            client.DefaultRequestHeaders.Clear();
            client.DefaultRequestHeaders.Accept.Add(this.Header);

```

```

        HttpResponseMessage response = await
            client.GetAsync(request);
        if (response.IsSuccessStatusCode)
        {
            T data = await response.Content.ReadAsAsync<T>();
            return data;
        }
        else
        {
            return default(T);
        }
    }

    public async Task<List<Empleado>> GetEmpleadosAsync()
    {
        string request = "api/empleados";
        List<Empleado> data =
            await this.CallApiAsync<List<Empleado>>(request);
        return data;
    }

    public async Task<List<string>> GetOficiosAsync()
    {
        string request = "api/empleados/oficios";
        List<string> data =
            await this.CallApiAsync<List<string>>(request);
        return data;
    }

    public async Task<List<Empleado>>
        GetEmpleadosOficiosAsync(string oficio)
    {
        string request = "api/empleados/empleadosoficio/" + oficio;
        List<Empleado> data =
            await this.CallApiAsync<List<Empleado>>(request);
        return data;
    }
}

```

#### EMPLEADOSCONTROLLER

```

public class EmpleadosController : Controller
{
    private ServiceEmpleados service;

    public EmpleadosController(ServiceEmpleados service)
    {
        this.service = service;
    }

    public async Task<IActionResult> Index()
    {
        List<Empleado> empleados =
            await this.service.GetEmpleadosAsync();
        List<string> oficios = await
            this.service.GetOficiosAsync();
        ViewData["OFICIOS"] = oficios;
        return View(empleados);
    }

    [HttpPost]
    public async Task<IActionResult> Index(string oficio)
    {
        List<Empleado> empleados =
            await this.service.GetEmpleadosOficiosAsync(oficio);
        List<string> oficios = await
            this.service.GetOficiosAsync();
        ViewData["OFICIOS"] = oficios;
        return View(empleados);
    }
}

```

Y ya tendremos la aplicación funcional

Apellido	Oficio
GIL	ANALISTA
FERNANDEZ	ANALISTA
SANTIUSTE	ANALISTA
GUTIERREZ	ANALISTA

El siguiente paso es visualizar las consultas de acción en Apis.

Vamos a crear un CRUD de departamentos para comprobar su funcionalidad.

Creamos un nuevo proyecto llamado **ApiCoreCrudDepartamentos**



**Microsoft.EntityFrameworkCore** by aspnet, dotnetframework, EntityFramework 9.0.3  
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server,...

Sobre Models creamos una nueva clase llamada **Departamento**

#### DEPARTAMENTO

```
[Table("DEPT")]
0 references
public class Departamento
{
    [Key]
    [Column("DEPT_NO")]
    0 references
    public int IdDepartamento { get; set; }
    [Column("DNOMBRE")]
    0 references
    public string Nombre { get; set; }
    [Column("LOC")]
    0 references
    public string Localidad { get; set; }
}
```

Creamos una carpeta llamada **Data** y una clase llamada **DepartamentosContext**

#### DEPARTAMENTOSCONTEXT

```
public class DepartamentosContext : DbContext
{
    0 references
    public DepartamentosContext(
        DbContextOptions<DepartamentosContext> options)
        : base(options) { }

    0 references
    public DbSet<Departamento> Departamentos { get; set; }
}
```

Creamos una carpeta llamada **Repositories** y una clase llama **RepositoryDepartamentos**

#### REPOSITORYDEPARTAMENTOS

```
public class RepositoryDepartamentos
{
    private DepartamentosContext context;

    public RepositoryDepartamentos(DepartamentosContext context)
    {
        this.context = context;
    }

    public async Task<List<Departamento>> GetDepartamentosAsync()
    {
        return await this.context.Departamentos.ToListAsync();
    }

    public async Task<Departamento> FindDepartamentoAsync(
        int idDepartamento)
    {
        return await this.context.Departamentos
            .FirstOrDefaultAsync(
                x => x.IdDepartamento == idDepartamento);
    }

    public async Task InsertDepartamentoAsync(int id, string nombre,
        string localidad)
    {
        Departamento dept = new Departamento();
        dept.IdDepartamento = id;
        dept.Nombre = nombre;
        dept.Localidad = localidad;
        await this.context.Departamentos.AddAsync(dept);
        await this.context.SaveChangesAsync();
    }

    public async Task UpdateDepartamentoAsync(
        int id, string nombre, string localidad)
    {
        Departamento dept = await this.FindDepartamentoAsync(id);
        dept.Nombre = nombre;
        dept.Localidad = localidad;
        await this.context.SaveChangesAsync();
    }

    public async Task DeleteDepartamentoAsync(int id)
    {
```

```

        Departamento dept = await this.FindDepartamentoAsync(id);
        this.context.Departamentos.Remove(dept);
        await this.context.SaveChangesAsync();
    }
}

```

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

#### DEPARTAMENTOSCONTROLLER

```

[Route("api/[controller]")]
[ApiController]
public class DepartamentosController : ControllerBase
{
    private RepositoryDepartamentos repo;

    public DepartamentosController(RepositoryDepartamentos repo)
    {
        this.repo = repo;
    }

    [HttpGet]
    public async Task<ActionResult<List<Departamento>>> GetDepartamentos()
    {
        return await this.repo.GetDepartamentosAsync();
    }

    [HttpGet("{id}")]
    public async Task<ActionResult<Departamento>> FindDepartamento(int id)
    {
        return await this.repo.FindDepartamentoAsync(id);
    }

    //LOS METODOS POR DEFECTO DE POST O PUT RECIBEN UN OBJETO
    //SI QUEREMOS ENVIAR PARAMETROS, DEBEMOS MAPEARLOS CON
    //ROUTING
    [HttpPost]
    public async Task<ActionResult> InsertDepartamento
        (Departamento departamento)
    {
        await this.repo.InsertDepartamentoAsync
            (departamento.IdDepartamento
            , departamento.Nombre, departamento.Localidad);
        return Ok();
    }

    //EL METODO POR DEFECTO DE DELETE RECIBE UN ID
    [HttpDelete("{id}")]
    public async Task<ActionResult> DeleteDepartamento(int id)
    {
        await this.repo.DeleteDepartamentoAsync(id);
        return Ok();
    }

    [HttpPut]
    public async Task<ActionResult> UpdateDepartamento
        (Departamento departamento)
    {
        await this.repo.UpdateDepartamentoAsync
            (departamento.IdDepartamento, departamento.Nombre
            , departamento.Localidad);
        return Ok();
    }

    //PODEMOS PERSONALIZAR METODOS POST, PUT O DELETE
    //A CONVENIENCIA
    [HttpPost]
    [Route("{action}/{id}/{nombre}/{localidad}")]
    public async Task<ActionResult>
        PostDepartamento(int id, string nombre, string localidad)
    {
        await this.repo.InsertDepartamentoAsync
            (id, nombre, localidad);
        return Ok();
    }

    //SI LO NECESITAMOS, PODEMOS COMBINAR OBJETOS CON
    //PARAMETROS. EL OBJETO ES EL ULTIMO ELEMENTO QUE
    //SE INCLUYE EN LA PETICION DEL METODO
    [HttpPut]
    [Route("{action}/{id}")]
    public async Task<ActionResult>
        PutDepartamento(int id, Departamento departamento)
    {
        await this.repo.UpdateDepartamentoAsync(id
            , departamento.Nombre, departamento.Localidad);
        return Ok();
    }
}

```

Resolvemos las dependencias en Program e incluimos Swagger

```

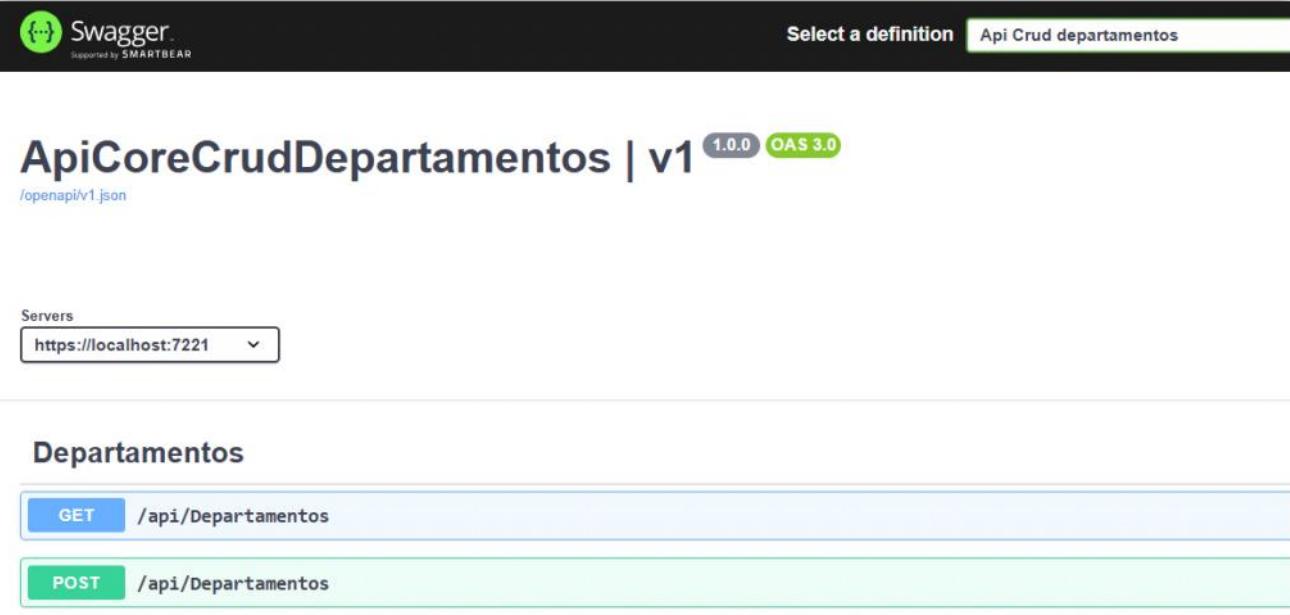
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
string connectionString =
    builder.Configuration.GetConnectionString("SqlAzure");
builder.Services.AddTransient<RepositoryDepartamentos>();
builder.Services.AddDbContext<DepartamentosContext>
    (options => options.UseSqlServer(connectionString));
builder.Services.AddControllers();
// Learn more about configuring OpenAPI at https://aka.ms/aspnet/

```

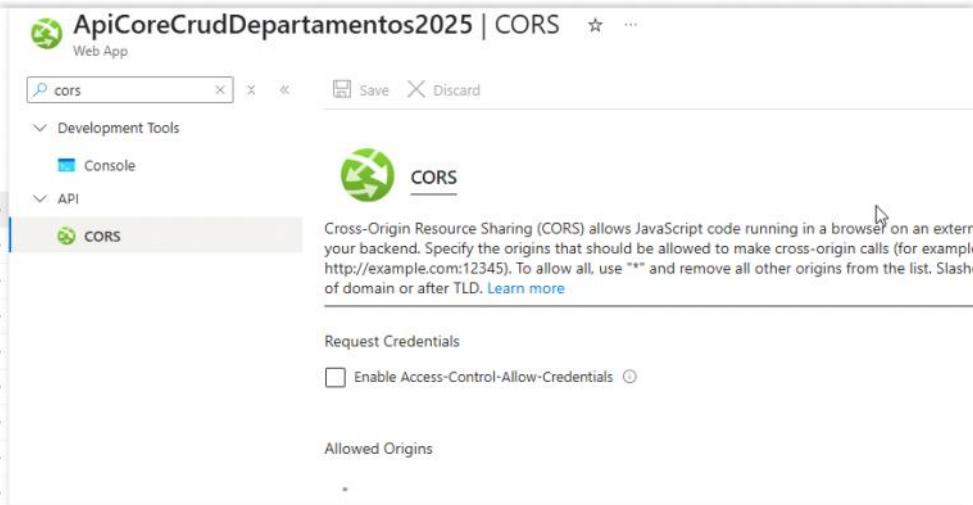
```
app.MapOpenApi();  
  
app.UseHttpsRedirection();  
app.UseSwaggerUI(options =>  
{  
    options.SwaggerEndpoint("/openapi/v1.json", "Api Crud departamentos")  
    options.RoutePrefix = "";  
});
```

Y ya tendremos funcional nuestro Api Crud



The screenshot shows the Swagger UI interface for the 'ApiCoreCrudDepartamentos | v1' API. At the top, there's a navigation bar with the API title, version (1.0.0), and OAS 3.0 badge. Below the title, it says '/openapi/v1.json'. On the left, there's a 'Servers' dropdown set to 'https://localhost:7221'. The main area is titled 'Departamentos' and lists two methods: 'GET /api/Departamentos' and 'POST /api/Departamentos'.

Lo subimos a la nube y abrimos CORS



The screenshot shows the Azure portal's CORS configuration for the 'ApiCoreCrudDepartamentos2025 | CORS' API. The 'cors' section is selected. It explains CORS and how to allow specific origins. Under 'Request Credentials', there's a checkbox for 'Enable Access-Control-Allow-Credentials'. Under 'Allowed Origins', there's a single entry: '\*'.

Vamos a consumir el proyecto CRUD con Cliente y con Servidor

# Cliente Api Crud departamentos

Id departamento

50

Nombre

Localidad

**Insertar** **Update** **Delete**

<b>Id</b>	<b>Nombre</b>	<b>Localidad</b>
10	CONTABILIDAD	ELCHE
20	INVESTIGACION	MADRID
30	VENTAS	BARCELONA
40	PRODUCCION	SALAMANCA

Creamos un nuevo proyecto llamado MvcCoreApiCrudDepartamentos

Creamos un nuevo controlador llamado DepartamentosController y un Action llamado Cliente con una página llamada Cliente.cshtml

CLIENTE.CSHTML

```
@section Scripts{
    <script>
        let url = "https://apicorecrudedepartamentos2025.azurewebsites.net/";
        $(document).ready(function(){
            loadDepartamentos();

            $("#botoninsert").click(function(){
                let id = parseInt($("#cajaid").val());
                let nombre = $("#cajanombre").val();
                let loc = $("#cajalocalidad").val();
                //CREAMOS UN OBJETO JS CON LAS MISMAS PROPIEDADES
                //QUE EL JSON DEL SERVICIO
                let dept = new Object();
                dept.idDepartamento = id;
                dept.nombre = nombre;
                dept.localidad = loc;
                var json = JSON.stringify(dept);
                var request = "api/departamentos";
                $.ajax({
                    url: url + request,
                    type: "POST",
                    data: json,
                    contentType: "application/json",
                    success: function(){
                        loadDepartamentos();
                    }
                })
            });

            $("#botonupdate").click(function(){
                let id = parseInt($("#cajaid").val());
                let nombre = $("#cajanombre").val();
                let localidad = $("#cajalocalidad").val();
                let dept = new Object();
                dept.idDepartamento = id;
                dept.nombre = nombre;
                dept.localidad = localidad;
                let json = JSON.stringify(dept);
                let request = "api/departamentos";
                $.ajax({
                    url: url + request,
                    type: "PUT",
                    data: json,
                    contentType: "application/json",
                    success: function(){
                        loadDepartamentos();
                    }
                })
            });

            $("#botonDelete").click(function() {
                let id = $("#cajaid").val();
                let request = "api/departamentos/" + id;
                $.ajax({
                    url: url + request,
                    type: "DELETE",
                    success: function(){
                        loadDepartamentos();
                    }
                })
            });
        });

        function loadDepartamentos() {
            let request = "api/departamentos";
            $.ajax({
                url: url + request,
                type: "GET",
                dataType: "json",
                success: function(data){
                    var html = "";
                    $.each(data, function(index, dept){
                        html += "<tr>";
                        html += "<td>" + dept.idDepartamento + "</td>";
                        html += "<td>" + dept.nombre + "</td>";
                        html += "<td>" + dept.localidad + "</td>";
                        html += "</tr>";
                    });
                    $("#tbody").html(html);
                }
            });
        }
    </script>
}
```

```

        })
        $("#tabladepartamentos tbody").html(html);
    }
}
</script>
}

<h1>Cliente Api Crud departamentos</h1>

<form>
    <label>Id departamento</label>
    <input type="text" id="cajaid" class="form-control" />
    <label>Nombre</label>
    <input type="text" id="cajanombre" class="form-control"/>
    <label>Localidad</label>
    <input type="text" id="cajalocalidad" class="form-control"/>
    <button type="button" class="btn btn-warning" id="botoninsert">
        Insertar
    </button>
    <button type="button" class="btn btn-info" id="botonupdate">
        Update
    </button>
    <button type="button" class="btn btn-danger" id="botondelete">
        Delete
    </button>
</form>
<table class="table table-bordered" id="tabladepartamentos">
    <thead>
        <tr>
            <th>Id</th>
            <th>Nombre</th>
            <th>Localidad</th>
        </tr>
    </thead>
    <tbody></tbody>
</table>

```

Imaginemos que tenemos que insertar o mostrar los departamentos utilizando código de Cliente, pero en distintas páginas. Solución?

Deberíamos crear un JS y eso es precisamente lo que vamos a realizar.

Creamos un nuevo JS dentro de `wwwroot/js` llamado `serviceapidepartamentos.js`

Necesitamos utilizar métodos asíncronos. AJAX es asíncrono y nuestro `loadDepartamentos`  
No lo es.

Cuando hace el return, ya ha pasado por el dibujo del LOAD.

Necesitamos crear una función `callBack()` que es una función que espera a que  
Finalice las acciones y devuelve los datos cuando decidamos nosotros dentro de la  
Función.

```

function myFunctionAsync(callBack) {
    //SE REALIZAN TODAS LAS ACCIONES QUE DESEEMOS
    //CUANDO HEMOS FINALIZADO Y DESEAMOS DEVOLVER,
    //UTILIZAMOS callBack
    callBack(parametros);
}

```

En la petición a la función:

```

function testFunction() {
    myFunctionAsync(function (data) {
        //AQUI TENEMOS EL CODIGO CUANDO HA FINALIZADO
    })
}

```

#### SERVICEAPIDEPARTAMENTOS.JS

```

let url = "https://apicorecruddepartamentos2025.azurewebsites.net/";
function getDepartamentosAsync(callBack) {
    let request = "api/departamentos";
    $.ajax({
        url: url + request,
        type: "GET",
        dataType: "json",
        success: function (data) {
            callBack(data);
        }
    })
}

function convertDeptToJson(id, nombre, localidad) {
    let dept = new Object();
    dept.idDepartamento = id;
    dept.nombre = nombre;
    dept.localidad = localidad;
    var json = JSON.stringify(dept);
    return json;
}

function insertDepartamentoAsync(id, nombre, localidad, callBack) {
    let json = convertDeptToJson(id, nombre, localidad);
    var request = "api/departamentos";
    $.ajax({
        url: url + request,
        type: "POST",
        data: json,
        contentType: "application/json",
        success: function () {
            callBack();
        }
    })
}

```

```

        })

    }

    function updateDepartamentoAsync(id, nombre, localidad, callBack) {
        let json = convertDeptToJson(id, nombre, localidad);
        var request = "api/departamentos";
        $.ajax({
            url: url + request,
            type: "PUT",
            data: json,
            contentType: "application/json",
            success: function () {
                callBack();
            }
        })
    }

    function deleteDepartamentoAsync(id, callBack) {
        let request = "api/departamentos/" + id;
        $.ajax({
            url: url + request,
            type: "DELETE",
            success: function () {
                callBack();
            }
        })
    }
}

CLIENTE.CSHTML

@section Scripts{
    <script src="~/js/serviceapidepartamentos.js"></script>
<script>
    //let url = "https://apicorecruddepartamentos2025.azurewebsites.net/";
    $(document).ready(function(){
        loadDepartamentos();

        $("#botoninsert").click(function(){
            let id = parseInt($("#cajaid").val());
            let nombre = $("#cajanombre").val();
            let loc = $("#cajalocalidad").val();
            insertDepartamentoAsync(id, nombre, loc, function(){
                loadDepartamentos();
            })
        });

        $("#botonupdate").click(function(){
            let id = parseInt($("#cajaid").val());
            let nombre = $("#cajanombre").val();
            let localidad = $("#cajalocalidad").val();
            updateDepartamentoAsync(id, nombre, localidad, function(){
                loadDepartamentos();
            })
        });

        $("#botondelete").click(function(){
            let id = $("#cajaid").val();
            deleteDepartamentoAsync(id, function(){
                loadDepartamentos();
            })
        });
    });

    function loadDepartamentos() {
        getDepartamentosAsync(function(data) {
            var html = "";
            $.each(data, function(index, dept){
                html += "<tr>";
                html += "<td>" + dept.idDepartamento + "</td>";
                html += "<td>" + dept.nombre + "</td>";
                html += "<td>" + dept.localidad + "</td>";
                html += "</tr>";
            })
            $("#tabladedepartamentos tbody").html(html);
        })
    }
</script>
}

<h1>Cliente Api Crud departamentos</h1>

<form>
    <label>Id departamento</label>
    <input type="text" id="cajaid" class="form-control" />
    <label>Nombre</label>
    <input type="text" id="cajanombre" class="form-control"/>
    <label>Localidad</label>
    <input type="text" id="cajalocalidad" class="form-control"/>
    <button type="button" class="btn btn-warning" id="botoninsert">
        Insertar
    </button>
    <button type="button" class="btn btn-info" id="botonupdate">
        Update
    </button>
    <button type="button" class="btn btn-danger" id="botondelete">
        Delete
    </button>
</form>
<table class="table table-bordered" id="tabladedepartamentos">
    <thead>
        <tr>
            <th>Id</th>
            <th>Nombre</th>
            <th>Localidad</th>
        </tr>
    </thead>
    <tbody></tbody>
</table>

```

A continuación, vamos a consumir el servicio utilizando código de servidor C#

Agregamos los siguientes Nuget al proyecto.



Sobre Models creamos una nueva clase llamada **Departamento**

#### DEPARTAMENTO

```
public class Departamento
{
    public int IdDepartamento { get; set; }
    public string Nombre { get; set; }
    public string Localidad { get; set; }
}
```

Creamos una carpeta llamada **Services** y una clase llamada **ServiceDepartamentos**

#### SERVICEDEPARTAMENTOS

```
public class ServiceDepartamentos
{
    private string UrlApi;
    private MediaTypeWithQualityHeaderValue header;

    public ServiceDepartamentos(IConfiguration configuration)
    {
        this.header = new
            MediaTypeWithQualityHeaderValue("application/json");
        this.UrlApi = configuration.GetValue<string>
            ("ApiUrls:ApiDepartamentos");
    }

    private async Task<T> CallApiAsync<T>(string request)
    {
        using (HttpClient client = new HttpClient())
        {
            client.BaseAddress = new Uri(this.UrlApi);
            client.DefaultRequestHeaders.Clear();
            client.DefaultRequestHeaders.Accept.Add(this.header);
            HttpResponseMessage response =
                await client.GetAsync(request);
            if (response.IsSuccessStatusCode)
            {
                T data = await
                    response.Content.ReadAsAsync<T>();
                return data;
            }
            else
            {
                return default(T);
            }
        }
    }

    public async Task<List<Departamento>>
        GetDepartamentosAsync()
    {
        string request = "api/departamentos";
        List<Departamento> data = await
            this.CallApiAsync<List<Departamento>>(request);
        return data;
    }

    public async Task<Departamento>
        FindDepartamentoAsync(int idDepartamento)
    {
        string request = "api/departamentos/" + idDepartamento;
        Departamento data = await
            this.CallApiAsync<Departamento>(request);
        return data;
    }

    //LOS METODOS DE ACCION QUE RECIBEN OBJETOS PUEDEN SER
    //GENERICOS Y RECIBIR T
    //SI LOS METODOS RECIBEN LA INFORMACION POR URL
    //SI QUE NO SUELEN SER GENERICOS
    public async Task InsertDepartamentoAsync
        (int id, string nombre
        , string localidad)
    {
        using (HttpClient client = new HttpClient())
        {
            string request = "api/departamentos";
            client.BaseAddress = new Uri(this.UrlApi);
            client.DefaultRequestHeaders.Clear();
            client.DefaultRequestHeaders.Accept.Add(this.header);
            //DEBEMOS CREAR NUESTRO MODEL A ENVIAR
            Departamento dept = new Departamento();
            dept.IdDepartamento = id;
            dept.Nombre = nombre;
            dept.Localidad = localidad;
            //CONVERTIMOS NUESTRO MODEL A JSON
            string json = JsonConvert.SerializeObject(dept);
            //PARA ENVIAR DATOS SE UTILIZA StringContent
            //DONDE DEBEMOS ENVIAR EL JSON, EL FORMATO Y EL TIPO
            StringContent content = new StringContent
                (json, Encoding.UTF8, "application/json");
            HttpResponseMessage response =
                await client.PostAsync(request, content);
        }
    }

    public async Task UpdateDepartamentoAsync
        (int id, string nombre, string localidad)
    {
        using (HttpClient client = new HttpClient())
        {
            string request = "api/departamentos";
            client.BaseAddress = new Uri(this.UrlApi);
            client.DefaultRequestHeaders.Clear();
            client.DefaultRequestHeaders.Accept.Add(this.header);
            Departamento departamento =
                new Departamento();
            departamento.IdDepartamento = id;
            departamento.Nombre = nombre;
            departamento.Localidad = localidad;
            string json = JsonConvert.SerializeObject(departamento);
            StringContent content = new StringContent
                (json, Encoding.UTF8, "application/json");
        }
    }
}
```

```

        await client.PutAsync(request, content);
    }

public async Task DeleteDepartamentoAsync(int idDepartamento)
{
    using (HttpClient client = new HttpClient())
    {
        string request = "api/departamentos/" + idDepartamento;
        client.BaseAddress = new Uri(this.UrlApi);
        client.DefaultRequestHeaders.Clear();
        client.DefaultRequestHeaders.Accept.Add(this.header);
        HttpResponseMessage response =
            await client.DeleteAsync(request);
    }
}

```

#### DEPARTAMENTOSCONTROLLER

```

public class DepartamentosController : Controller
{
    private ServiceDepartamentos service;
    public DepartamentosController(ServiceDepartamentos service)
    {
        this.service = service;
    }

    public async Task<IActionResult> Index()
    {
        List<Departamento> data = await
            this.service.GetDepartamentosAsync();
        return View(data);
    }

    public async Task<IActionResult> Details(int id)
    {
        Departamento data = await
            this.service.FindDepartamentoAsync(id);
        return View(data);
    }

    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Create(Departamento dept)
    {
        await this.service.InsertDepartamentoAsync(
            dept.IdDepartamento, dept.Nombre, dept.Localidad);
        return RedirectToAction("Index");
    }

    public async Task<IActionResult> Edit(int id)
    {
        Departamento dept = await
            this.service.FindDepartamentoAsync(id);
        return View(dept);
    }

    [HttpPost]
    public async Task<IActionResult> Edit(Departamento dept)
    {
        await this.service.UpdateDepartamentoAsync(
            dept.IdDepartamento, dept.Nombre, dept.Localidad);
        return RedirectToAction("Index");
    }

    public async Task<IActionResult> Delete(int id)
    {
        await this.service.DeleteDepartamentoAsync(id);
        return RedirectToAction("Index");
    }

    public IActionResult Cliente()
    {
        return View();
    }
}

```

#### APPSETTINGS.JSON

```

{
    "AllowedHosts": "*",
    "ApiUrls": {
        "ApiDepartamentos": "https://apicorecruddepartamentos2025.azu"
    }
}

```

#### PROGRAM

```

// Add services to the container.
builder.Services.AddTransient<ServiceDepartamentos>();
builder.Services.AddControllersWithViews();

var app = builder.Build();

```

Y tendremos lista nuestra App



# Index

[Create New](#)

<b>IdDepartamento</b>	<b>Nombre</b>	<b>Localidad</b>			
10	CONTABILIDAD	ELCHE	<a href="#">Details</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
20	INVESTIGACION	MADRID	<a href="#">Details</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
30	VENTAS	BARCELONA	<a href="#">Details</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
40	PRODUCCION	SALAMANCA	<a href="#">Details</a>	<a href="#">Edit</a>	<a href="#">Delete</a>

# SEGURIDAD API

jueves, 27 de marzo de 2025 9:18

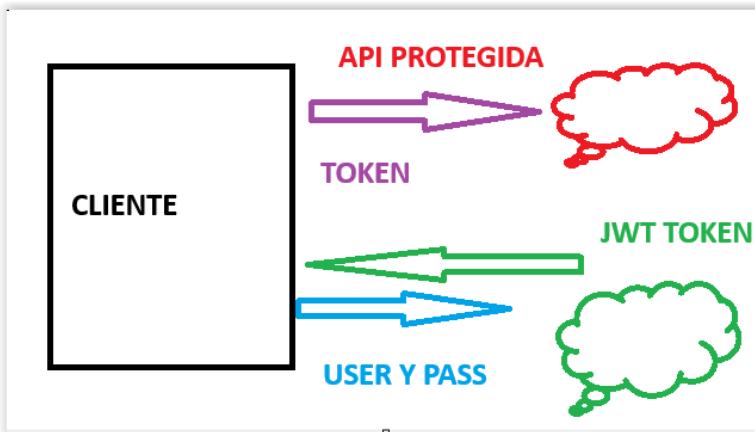
Dentro de nuestras Apis, tenemos varios tipos de seguridad

Algunos se pueden combinar y otros no:

- 1) **Basic Authentication:** Está basada en User y Password. Todo va cifrado y necesitamos de esos dos datos para la validación
- 2) **Third Party Authentication:** Autenticación basada en Providers. Se utilizan proveedores de acceso como, por ejemplo, Microsoft con Azure Active Directory. AD es una funcionalidad que tenemos dentro de Azure y que permitiría validar contra los usuarios. Que tengamos dentro de nuestro dominio (@tajamar365)
- 3) **Token Authentication (OAuth 2.0):** Se utiliza mediante un Token generado dentro del propio Api. Para poder acceder, los métodos deben estar protegidos al estilo de Net Core. Dicho Token puede llevar información del usuario, tiempo de acceso. También es llamado **JWT**

Con el tipo de seguridad **Oauth** tenemos que realizar dos peticiones:

- 1) Enviar el User y Password a un método abierto/cifrado en el Api para la validación y nos devolverá el Token
- 2) Con el Token, podemos hacer peticiones personalizadas a cualquier método.



Un token es generado en un servicio a partir de una serie de datos que nos inventamos. Dichos datos pueden ser opcionales algunos de ellos, a mayor número de datos, mayor seguridad

- 1) **Issuer:** Suele ser la URL de nuestro Servicio
- 2) **Secret Key:** Una clave que nos inventaremos, cuanto mayor longitud y complejidad, mayor seguridad.
- 3) **Audience:** Es otro conjunto de palabras que conformarán nuestro Token
- 4) **Actor:** El usuario que se ha validado en nuestro Api

Los Tokens van con tiempo siempre, aunque podemos poner 2 años.

Comenzamos creando un nuevo proyecto llamado **ApiOAuthEmpleados**

Microsoft.EntityFrameworkCore 9.0.3  
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ, queries, change tracking, updates, and schema migrations. EF Core works with SQL Server...

Microsoft.EntityFrameworkCore.SqlServer 9.0.3  
Microsoft SQL Server database provider for Entity Framework Core.

Swashbuckle.AspNetCore 702M downloads  
Swagger tools for documenting APIs built on ASP.NET Core

## EMPLEADO

```
[Table("EMP")]
public class Empleado
{
    [Key]
    [Column("EMP_NO")]
    public int IdEmpleado { get; set; }
    [Column("APELLIDO")]
    public string Apellido { get; set; }
    [Column("OFICIO")]
    public string Oficio { get; set; }
```

```

    [Column("SALARIO")]
    public int Salario { get; set; }
    [Column("DEPT_NO")]
    public int IdDepartamento { get; set; }
}

```

#### HOSPITALCONTEXT

```

public class HospitalContext : DbContext
{
    0 references
    public HospitalContext(DbContextOptions<HospitalContext>
        options) : base(options) { }

    0 references
    public DbSet<Empleado> Empleados { get; set; }
}

```

#### REPOSITORYHOSPITAL

```

public class RepositoryHospital
{
    private HospitalContext context;

    public RepositoryHospital(HospitalContext context)
    {
        this.context = context;
    }

    public async Task<List<Empleado>> GetEmpleadosAsync()
    {
        return await this.context.Empleados.ToListAsync();
    }

    public async Task<Empleado>
        FindEmpleadoAsync(int idEmpleado)
    {
        return await this.context.Empleados
            .FirstOrDefaultAsync(z => z.IdEmpleado == idEmpleado);
    }
}

```

#### APPSETTINGS.JSON

```

},
"AllowedHosts": "*",
"ConnectionStrings": {
    "SqlAzure": "Data Source=techriders.database.windows.net,1433;Initial Catalog=Techriders;User ID=sa;Password=123456;Trusted_Connection=False;Encrypt=True;Connect Timeout=30"
}


```

#### EMPLEADOSCONTROLLER

```

[Route("api/[controller]")]
[ApiController]
public class EmpleadosController : ControllerBase
{
    private RepositoryHospital repo;

    public EmpleadosController(RepositoryHospital repo)
    {
        this.repo = repo;
    }

    [HttpGet]
    public async Task<ActionResult<List<Empleado>>>
        GetEmpleados()
    {
        return await this.repo.GetEmpleadosAsync();
    }

    [HttpGet("{id}")]
    public async Task<ActionResult<Empleado>>
        FindEmpleado(int id)
    {
        return await this.repo.FindEmpleadoAsync(id);
    }
}

```

#### PROGRAM

```

using ApiOAuthEmpleados.Data;
using ApiOAuthEmpleados.Repositories;
using Microsoft.EntityFrameworkCore;
using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
string connectionString =
    builder.Configuration.GetConnectionString("SqlAzure");
builder.Services.AddTransient<RepositoryHospital>();
builder.Services.AddDbContext<HospitalContext>

```

```

    Options => options.UseSqlServer(connectionString));
builder.Services.AddControllers();
// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi
builder.Services.AddOpenApi();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
}

app.MapOpenApi();
app.UseHttpsRedirection();

app.UseSwaggerUI(options =>
{
    options.SwaggerEndpoint("/openapi/v1.json", "Api Seguridad Empleados")
    options.RoutePrefix = "";
});

app.UseAuthorization();

app.MapControllers();

app.Run();

```

Es el momento de implementar la seguridad

Lo primero que vamos a realizar es crear un método dentro del **RepositoryHospital** para validar a los empleados.  
Recibiremos el Apellido y el Id del Empleado

#### REPOSITORYEMPLEADOS

```

public async Task<Empleado>
    LogInEmpleadoAsync(string apellido, int idEmpleado)
{
    return await this.context.Empleados
        .Where(x => x.Apellido == apellido
            && x.IdEmpleado == idEmpleado)
        .FirstOrDefaultAsync();
}

```

Como os he comentado, necesitamos una serie de Keys para validar el Token  
Y poder generarlo correctamente.

#### APPSETTINGS.JSON

```

"ApiOAuthToken": {
    "Issuer": "URL DEL PROYECTO",
    "Audience": "NOMBRE DEL PROYECTO",
    "SecretKey": "KEY LARGA INVENTADA"
}

```

Necesitamos agregar el siguiente Nuget al proyecto

 Microsoft.AspNetCore.Authentication.JwtBearer by aspnet, dotnetfran 9.0.3  
ASP.NET Core middleware that enables an application to receive an OpenID Connect bearer token.

Para habilitar la seguridad necesitamos incluir una serie de Servicios dentro de **Program**

También debemos generar una Key para la seguridad

Todo esto, es bastante código para incluirlo dentro de **Program** lo que vamos a Realizar es una clase para habilitar los servicios de seguridad y, posteriormente, Lanzarlos desde **Program**  
Necesitamos crear un Servicio para builder y agregarlo en **Program**

```

0 references
public Action<TipoServicio> GetActionService()
{
    //TODO EL CODIGO
}

```

La llamada al servicio de **Program**

```
builder.Services.AddAuthentication(Clase.TipoServicio);
```

Creamos una carpeta llamada **Helpers** y una clase llamada **HelperActionServicesOAuth**

#### HELPERRACTIONSERVICESOAUTH

```
public class HelperActionServicesOAuth
{
    public string Issuer { get; set; }
    public string Audience { get; set; }
    public string SecretKey { get; set; }
    public HelperActionServicesOAuth(IConfiguration configuration)
    {
        this.Issuer =
            configuration.GetValue<string>("ApiOAuthToken:Issuer");
        this.Audience =
            configuration.GetValue<string>("ApiOAuthToken:Audience");
        this.SecretKey =
            configuration.GetValue<string>("ApiOAuthToken:SecretKey");
    }

    //NECESITAMOS UN METODO PARA GENERAR EL TOKEN
    //Dicho TOKEN SE BASA EN NUESTRO SECRET KEY
    public SymmetricSecurityKey GetKeyToken()
    {
        //CONVERTIMOS EL SECRET KEY A BYTES
        byte[] data =
            Encoding.UTF8.GetBytes(this.SecretKey);
        //DEVOLVEMOS LA KEY GENERADA A PARTIR DE LOS BYTES
        return new SymmetricSecurityKey(data);
    }

    //ESTA CLASE LA HEMOS CREADO TAMBIEN PARA QUITAR CODIGO
    //DEL PROGRAMA
    public Action<JwtBearerOptions> GetJwtBearerOptions()
    {
        Action<JwtBearerOptions> options =
            new Action<JwtBearerOptions>(options =>
        {
            //INDICAMOS QUE DEBEMOS VALIDAR PARA EL TOKEN
            options.TokenValidationParameters =
                new TokenValidationParameters
                {
                    ValidateIssuer = true,
                    ValidateAudience = true,
                    ValidateLifetime = true,
                    ValidateIssuerSigningKey = true,
                    ValidIssuer = this.Issuer,
                    ValidAudience = this.Audience,
                    IssuerSigningKey = this.GetKeyToken()
                };
        });
        return options;
    }

    //TODA SEGURIDAD SIEMPRE ESTA BASADA EN UN SCHEMA
    public Action<AuthenticationOptions>
        GetAuthenticateSchema()
    {
        Action<AuthenticationOptions> options =
            new Action<AuthenticationOptions>(options =>
        {
            options.DefaultScheme =
                JwtBearerDefaults.AuthenticationScheme;
            options.DefaultChallengeScheme =
                JwtBearerDefaults.AuthenticationScheme;
            options.DefaultAuthenticateScheme =
                JwtBearerDefaults.AuthenticationScheme;
        });
        return options;
    }
}
```

Habilitamos la seguridad dentro de **Program**

#### PROGRAM

```
var builder = WebApplication.CreateBuilder(args);

//CREAMOS UNA INSTANCIA DE NUESTRO HELPER
HelperActionServicesOAuth helper =
    new HelperActionServicesOAuth(builder.Configuration);
//ESTA INSTANCIA SOLAMENTE DEBEMOS CREARLA UNA VEZ
//PARA QUE NUESTRA APLICACION PUEDA VALIDAR CON TODO LO QUE HA CREADO
builder.Services.AddSingleton<HelperActionServicesOAuth>(helper);
//HABILITAMOS LA SEGURIDAD UTILIZANDO LA CLASE HELPER
builder.Services.AddAuthentication(helper.GetAuthenticateSchema())
    .AddJwtBearer(helper.GetJwtBearerOptions());
```

```

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app.Run();

```

Las validaciones se realizan mediante métodos POST, es decir, todo va cifrado.  
 Nunca se enviará el user y pass por Get o Url.  
 Como van por Post, necesitamos recibir información del User y Pass mediante  
 Una clase.

Sobre **Models** creamos una clase llamada **LoginModel**

**LOGINMODEL**

```

public class LoginModel
{
    0 references
    public string UserName { get; set; }

    0 references
    public string Password { get; set; }
}

```

Necesitamos un **EndPoint** para que el usuario pueda realizar la validación.  
 Sobre **Controllers** creamos un nuevo controlador llamado **AuthController**

**AUTHCONTROLLER**

```

[Route("api/[controller]")]
[ApiController]
public class AuthController : ControllerBase
{
    private RepositoryHospital repo;
    //CUANDO GENEREMOS EL TOKEN DEBEMOS INTEGRAR
    //ALGUNOS DATOS COMO ISSUER Y DEMAS
    private HelperActionServicesOAuth helper;

    public AuthController(RepositoryHospital repo
        , HelperActionServicesOAuth helper)
    {
        this.repo = repo;
        this.helper = helper;
    }

    [HttpPost]
    [Route("[action]")]
    public async Task<ActionResult>
        Login(LoginModel model)
    {
        Empleado empleado = await
            this.repo.LogInEmpleadoAsync(model.UserName
            , int.Parse(model.Password));
        if (empleado == null)
        {
            return Unauthorized();
        }
        else
        {
            //DEBEMOS CREAR UNAS CREDENCIALES PARA
            //INCLUIRLAS DENTRO DEL TOKEN Y QUE ESTARAN
            //COMPUESTAS POR EL SECRET KEY CIFRADO Y EL
            //TIPO DE CIFRADO QUE INCLUIREMOS EN EL TOKEN
            SigningCredentials credentials =
                new SigningCredentials
                    (this.helper.GetKeyToken(),
                     SecurityAlgorithms.HmacSha256);
            //EL TOKEN SE GENERA CON UNA CLASE
            //Y DEBEMOS INDICAR LOS DATOS QUE ALMACENARA EN SU
            //INTERIOR
            JwtSecurityToken token =
                new JwtSecurityToken(
                    issuer: this.helper.Issuer,
                    audience: this.helper.Audience,
                    signingCredentials: credentials,
                    expires: DateTime.UtcNow.AddMinutes(20),
                    notBefore: DateTime.UtcNow
                );
            //POR ULTIMO, DEVOLVEMOS LA RESPUESTA AFIRMATIVA
            //CON UN OBJETO QUE CONTENGA EL TOKEN (anónimo)
            return Ok(new
            {
                response =
                    new JwtSecurityTokenHandler()
                    .WriteToken(token)
            });
        }
    }
}

```

Hemos terminado. Cada método que deseemos proteger, debemos incluir  
 Una decoración **[Authorize]**

Vamos a probar el método protegido con **FindEmpleado** de nuestro **Controller**

**Nota:** No podemos probar la seguridad con Swagger, necesitamos un cliente

```
[Authorize]
[HttpGet("{id}")]
public async Task<ActionResult<Empleado>>
{
    FindEmpleado(int id)
{
    return await this.repo.FindEmpleadoAsync(id);
}
```

Al probar el método `Api/Empleados/{id}` nos devolverá un Error de autorización

The screenshot shows the Swagger UI interface. A search bar at the top has the text "Login Empleado". Below it, a table lists a single API endpoint:

Code	Details
401 Undocumented	Error: response status is 401 Response headers: <code>content-length: 0</code> <code>date: Thu,27 Mar 2025 10:19:57 GMT</code> <code>server: Kestrel</code> <code>www-authenticate: Bearer</code>

Below the table, a section titled "Responses" is visible.

Si realizamos una petición correcta, nos devolverá nuestro maravilloso Token

The screenshot shows the Postman application interface. A search bar at the top has the text "Login Empleado". Below it, a table lists a single API endpoint:

Method	URL	Status	Time	Last Run
POST	https://localhost:7109/api/auth/login	200 OK	265 ms	219 B Just Now

Below the table, tabs for "Params", "Body", "Auth", "Headers", "Scripts", and "Doc" are visible. The "Body" tab is selected and contains a JSON payload:

```
1: {
2:   "userName": "REY",
3:   "password": "7839"
4: }
```

The "Headers" tab shows the response headers:

```
1: {
2:   "response": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJumYiojE3NDMwNzA5OTksImV4cCI6Mtc0MzA3MjE5OSwiaXNzIjoiaHR0chHM6Ly9sb2NhbGhvc3Q6NzEwOSisImF1ZCI6IkFwaU9BdXRoRW1wbGVhZG9zIn0.vs0T_LUq9FDEsFAyQDUKGYK25hsHn88C8ryBzN1pXcA"
```

Debemos enviar el token mediante la palabra clave **bearer TOKEN**

La clave para enviarlo es mediante **Authorization**

The screenshot shows the Postman application interface. A search bar at the top has the text "Login Empleado". Below it, a table lists a single API endpoint:

Method	URL	Status	Time	Last Run
GET	https://localhost:7109/api/empleados/7	200 OK	883 ms	95 B

Below the table, tabs for "Params", "Body", "Auth", "Headers", "Scripts", and "Docs" are visible. The "Headers" tab is selected and contains an "Authorization" header:

```
Accept: /*  
Host: <calculated at runt  
User-Agent: insomnia/11.0.0  
Authorization: bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJumYiojE3NDMwNzA5OTksImV4cCI6Mtc0MzA3MjE5OSwiaXNzIjoiaHR0chHM6Ly9sb2NhbGhvc3Q6NzEwOSisImF1ZCI6IkFwaU9BdXRoRW1wbGVhZG9zIn0.vs0T_LUq9FDEsFAyQDUKGYK25hsHn88C8ryBzN1pXcA"
```

The "Preview" tab shows the response body:

```
1: {
2:   "idEmpleado": 7839,
3:   "apellido": "REY",
4:   "oficio": "PRESIDENTE",
5:   "salario": 650000,
6:   "idDepartamento": 10
7: }
```

Una vez que tenemos la seguridad habilitada vamos a visualizar cómo podemos

Consumir el servicio.  
Utilizaremos una aplicación de Consola para enviar User y Pass y recuperar  
El Token para llamar al Servicio Api.

Creamos una nueva aplicación de Consola llamada **ClienteApiOAuthEmpleados**

```
Client Api OAuth
Introduzca Apellido
REY
Password
7539
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJuYmYiOjE3NDMwNzc1OTYsImV4cCI6MTc0MzA30Dc5NiwiZXNzIjoiaHR0cHM6Ly9sb2Nhbgv3Q6NzEwOSIsImF1ZCI6IkFwaU9BdXR0RWlwbGVhZG9zIn0.ZcstBq6yhp4m7V2C8KzXpD6qzlf5Wtg2Yh06GB0NM0o
A continuación, Introduzca ID de empleado a buscar...
7566
{"idEmpleado":7566,"apellido":"JIMENEZ","oficio":"DIRECTOR","salario":386750,"idDepartamento":20}
```

Agregamos el Nuget de Web Api Client y Newtonsoft JSON



Creamos una clase llamada **LoginModel**

**LOGINMODEL**

```
public class LoginModel
{
    public string UserName { get; set; }
    public string Password { get; set; }
}
```

Vamos a escribir todo el código dentro de **Program**

Tendremos un método para recuperar el Token enviando el User y Pass y  
Otro método que devolverá un Empleado mediante dicho Token

**PROGRAM**

```
using ClienteApiOAuthEmpleados;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System.Net.Http.Headers;
using System.Runtime.CompilerServices;
using System.Text;

Console.WriteLine("Client Api OAuth");
Console.WriteLine("Introduzca Apellido");
string apellido = Console.ReadLine();
Console.WriteLine("Password");
string password = Console.ReadLine();
string respuesta = await GetTokenAsync(apellido, password);
Console.WriteLine(respuesta);

Console.WriteLine("A continuación, Introduzca ID de empleado a buscar...");
int id = int.Parse(Console.ReadLine());
string data = await FindEmpleadoAsync(id, respuesta);
Console.WriteLine(data);
Console.WriteLine("-----");
//PARA CREAR UN METODO EN PROGRAM, DEBEMOS HACERLO static
static async Task<string> GetTokenAsync(string user, string pass)
{
    string urlApi = "https://localhost:7109/";
    LoginModel model = new LoginModel
    {
        UserName = user,
        Password = pass
    };
    using (HttpClient client = new HttpClient())
    {
        string request = "api/auth/login";
        client.BaseAddress = new Uri(urlApi);
        client.DefaultRequestHeaders.Clear();
        client.DefaultRequestHeaders.Accept.Add(
            new MediaTypeWithQualityHeaderValue("application/json"));
        string json = JsonConvert.SerializeObject(model);
        StringContent content =
            new StringContent(json, Encoding.UTF8, "application/json");
        HttpResponseMessage response =
            await client.PostAsync(request, content);
        if (response.IsSuccessStatusCode)
    }
}
```

```

        string data = await
            response.Content.ReadAsStringAsync();
        JObject keys = JObject.Parse(data);
        string token = keys.GetValue("response").ToString();
        return token;
    }
}
//CREAMOS OTRO METODO PARA LEER UN EMPLEADO Y HACERLO CON
//EL TOKEN QUE HEMOS RECUPERADO
static async Task<string> FindEmpleadoAsync(int idEmpleado
    , string token)
{
    string urlApi = "https://localhost:7109/";
    using (HttpClient client = new HttpClient())
    {
        string request = "api/empleados/" + idEmpleado;
        client.BaseAddress = new Uri(urlApi);
        client.DefaultRequestHeaders.Clear();
        client.DefaultRequestHeaders.Accept.Add(
            new MediaTypeWithQualityHeaderValue("application/json"));
        //PARA LA PETICION, DEBEMOS HACER LO MISMO QUE HEMOS
        //HECHO EN Insomnia/Postman, ES DECIR, INCLUIR EN
        //EL HEADER Authorization y bearer token
        client.DefaultRequestHeaders.Add
            ("Authorization", "bearer " + token);
        HttpResponseMessage response =
            await client.GetAsync(request);
        if (response.IsSuccessStatusCode)
        {
            string data = await
                response.Content.ReadAsStringAsync();
            return data;
        }
        else
        {
            return "Error de algo: " + response.StatusCode;
        }
    }
}

```

#### SEGURIDAD EN PARALELO CON MVC NET CORE Y API

Cuando hablamos de seguridad, no solamente tiene que estar en el Api, sino que debe estar también en  
Cada aplicación que realicemos de acceso al Api.

Creamos un nuevo proyecto de tipo Mvc Net Core llamado **MvcOAuthEmpleados**



Sobre **Models** creamos una clase llamada **Empleado**

```

public class Empleado
{
    0 references
    public int IdEmpleado { get; set; }
    0 references
    public string Apellido { get; set; }
    0 references
    public string Oficio { get; set; }
    0 references
    public int Salario { get; set; }
    0 references
    public int IdDepartamento { get; set; }
}

```

Sobre **Models** creamos una clase llamada **LoginModel**

**LOGINMODEL**

```

public class LoginModel
{
    0 references
    public string UserName { get; set; }
    0 references
    public string Password { get; set; }
}

```

Creamos un nuevo servicio llamado **ServiceEmpleados**

#### SERVICEEMPLEADOS

```

public class ServiceEmpleados
{
    private string UrlApi;
    private MediaTypeWithQualityHeaderValue Header;

    public ServiceEmpleados(IConfiguration configuration)
    {
        this.UrlApi = configuration.GetValue<string>("ApiUrls:ApiEmpleados");
        this.Header = new MediaTypeWithQualityHeaderValue("application/json");
    }

    public async Task<string> GetTokenAsync
        (string userName, string password)
    {
        using (HttpClient client = new HttpClient())
        {
            string request = "api/auth/login";
            client.BaseAddress = new Uri(this.UrlApi);
            client.DefaultRequestHeaders.Clear();
            client.DefaultRequestHeaders.Accept.Add(this.Header);
            LoginModel model = new LoginModel
            {
                UserName = userName, Password=password
            };
            string json = JsonConvert.SerializeObject(model);
            StringContent content = new StringContent
                (json, Encoding.UTF8, "application/json");
            HttpResponseMessage response =
                await client.PostAsync(request, content);
            if (response.IsSuccessStatusCode)
            {
                string data = await response.Content
                    .ReadAsStringAsync();
                JObject keys = JObject.Parse(data);
                string token = keys.GetValue("response").ToString();
                return token;
            }
            else
            {
                return null;
            }
        }
    }

    private async Task<T> CallApiAsync<T>(string request)
    {
        using (HttpClient client = new HttpClient())
        {
            client.BaseAddress = new Uri(this.UrlApi);
            client.DefaultRequestHeaders.Clear();
            client.DefaultRequestHeaders.Accept.Add(this.Header);
            HttpResponseMessage response =
                await client.GetAsync(request);
            if (response.IsSuccessStatusCode)
            {
                T data = await response.Content.ReadAsAsync<T>();
                return data;
            }
            else
            {
                return default(T);
            }
        }
    }

    //VAMOS A REALIZAR UNA SOBRECARGA DEL METODO
    //RECIBIENDO EL TOKEN
    private async Task<T> CallApiAsync<T>
        (string request, string token)
    {
        using (HttpClient client = new HttpClient())
        {
            client.BaseAddress = new Uri(this.UrlApi);
            client.DefaultRequestHeaders.Clear();
            client.DefaultRequestHeaders.Accept.Add(this.Header);
            client.DefaultRequestHeaders.Add
                ("Authorization", "bearer " + token);
            HttpResponseMessage response =
                await client.GetAsync(request);
            if (response.IsSuccessStatusCode)
            {
                T data = await response.Content.ReadAsAsync<T>();
                return data;
            }
            else
            {
                return default(T);
            }
        }
    }
}

```

```

    }

    public async Task<List<Empleado>> GetEmpleadosAsync()
    {
        string request = "api/empleados";
        List<Empleado> empleados = await
            this.CallApiAsync<List<Empleado>>(request);
        return empleados;
    }

    //ALMACENAREMOS EL TOKEN EN SESSION
    //POR AHORA, RECIBIREMOS EL TOKEN EN EL METODO
    public async Task<Empleado> FindEmpleadoAsync
        (int idEmpleado, string token)
    {
        string request = "api/empleados/" + idEmpleado;
        Empleado empleado = await
            this.CallApiAsync<Empleado>(request, token);
        return empleado;
    }
}

```

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

#### EMPLEADOSCONTROLLER

```

public class EmpleadosController : Controller
{
    private ServiceEmpleados service;

    public EmpleadosController(ServiceEmpleados service)
    {
        this.service = service;
    }

    public async Task<IActionResult> Index()
    {
        List<Empleado> empleados =
            await this.service.GetEmpleadosAsync();
        return View(empleados);
    }

    public async Task<IActionResult> Details(int id)
    {
        //TENDREMOS EL TOKEN EN SESSION
        string token = HttpContext.Session.GetString("TOKEN");
        if (token == null)
        {
            ViewData["MENSAJE"] = "Debe validarse en Login";
            return View();
        }
        else
        {
            Empleado empleado = await
                this.service.FindEmpleadoAsync(id, token);
            return View(empleado);
        }
    }
}

```

#### APPSETTINGS.JSON

```

appsettings.json  EmpleadosController.cs  ServiceEmpleados.cs  LoginModel.cs  Empleado.cs  NuGet: MvcOAuthEmpleados  MvcOAuthEmpl...os: Overview
Schema: https://json.schemastore.org/appsettings.json

    },
    "AllowedHosts": "*",
    "ApiUrls": {
        "ApiEmpleados": "https://apicoreoauthempleados2025.azurewebsi"
    }
}

```

En **Program** inyectamos el servicio y habilitamos Session con Tiempo de 30 minutos que es lo mismo que pusimos ayer a nuestro Api.

#### PROGRAM

```

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddTransient<ServiceEmpleados>();
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromMinutes(30);
});
builder.Services.AddControllersWithViews();

```

```

app.UseAuthorization();

app.MapStaticAssets();
app.UseSession();
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}"
    .WithStaticAssets();

```

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

#### MANAGEDCONTROLLER

```

public class ManagedController : Controller
{
    private ServiceEmpleados service;

    public ManagedController(ServiceEmpleados service)
    {
        this.service = service;
    }

    public IActionResult Login()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Login(LoginModel model)
    {
        string token = await this.service
            .GetTokenAsync(model.UserName, model.Password);
        if (token == null)
        {
            ViewData["MENSAJE"] = "Usuario/Password incorrectos";
        }
        else
        {
            ViewData["MENSAJE"] = "Ya tienes tu Token!!!";
            HttpContext.Session.SetString("TOKEN", token);
        }
        return View();
    }
}

```

El siguiente paso es generar la seguridad correcta dentro de Mvc Net Core

Creamos una carpeta llamada **Filters** y una clase llamada **AuthorizeEmpleadosAttribute**

#### AUTHORIZEEMPLEADOSATTRIBUTE

```

public class AuthorizeEmpleadosAttribute : AuthorizeAttribute
    , IAuthorizationFilter
{
    public void OnAuthorization(AuthorizationFilterContext context)
    {
        var user = context.HttpContext.User;
        if (user.Identity.IsAuthenticated == false)
        {
            RouteValueDictionary routeLogin =
                new RouteValueDictionary(new
                {
                    controller = "Managed",
                    action = "Login"
                });
            context.Result = new
                RedirectToRouteResult(routeLogin);
        }
    }
}

```

Modificamos el código de ManagedController con funcionalidad de seguridad

#### MANAGEDCONTROLLER

```

public class ManagedController : Controller
{
    private ServiceEmpleados service;

    public ManagedController(ServiceEmpleados service)
    {
        this.service = service;
    }

    public IActionResult Login()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Login(LoginModel model)
    {
        string token = await this.service
            .GetTokenAsync(model.UserName, model.Password);
        if (token == null)
        {
            ViewData["MENSAJE"] = "Usuario/Password incorrectos";
            return View();
        }
    }
}

```

```

        }
    else
    {
        ViewData["MENSAJE"] = "Ya tienes tu Token!!!";
        HttpContext.Session.SetString("TOKEN", token);
        ClaimsIdentity identity =
            new ClaimsIdentity(
                CookieAuthenticationDefaults.AuthenticationScheme
                , ClaimTypes.Name, ClaimTypes.Role);
        //ALMACENAMOS EL NOMBRE DEL USUARIO (BONITO)
        identity.AddClaim(new Claim(
            ClaimTypes.Name, model.UserName));
        //EN ESTE EJEMPLO, ALMACENAMOS EL ID (PASSWORD)
        identity.AddClaim(new Claim(
            ClaimTypes.NameIdentifier, model.Password));
        ClaimsPrincipal principal =
            new ClaimsPrincipal(identity);
        //EL USUARIO ESTARA DADO DE ALTA 30 MINUTOS
        //, LO MISMO QUE SESSION
        await HttpContext.SignInAsync(
            CookieAuthenticationDefaults.AuthenticationScheme
            , principal, new AuthenticationProperties
            {
                ExpiresUtc = DateTime.UtcNow.AddMinutes(30)
            });
        return RedirectToAction("Index", "Home");
    }
}

public async Task<IActionResult> Logout()
{
    await HttpContext.SignOutAsync(
        CookieAuthenticationDefaults.AuthenticationScheme);
    return RedirectToAction("Index", "Home");
}
}

```

Sobre Shared vamos a crear una nueva vista llamada \_MenuEmpleado.cshtml

\_MENUEMPLEADO.CSHTML

```

@if (Context.User.Identity.IsAuthenticated){
    <li class="nav-item">
        <a class="nav-link text-danger"
            asp-controller="Empleados"
            asp-action="Index">
            Empleados
        </a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-danger"
            asp-controller="Managed"
            asp-action="Logout">
            Log Out <span style="color:blue">
                @Context.User.Identity.Name
            </span>
        </a>
    </li>
} else{
    <li class="nav-item">
        <a class="nav-link text-black"
            asp-controller="Empleados"
            asp-action="Index">
            Log In
        </a>
    </li>
}

```

Incluimos el Partial dentro de \_Layout y quitamos lo que había

\_LAYOUT.CSHTML

```

<ul class="navbar-nav flex-grow-1">
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area=>
    </li>
    <partial name="_MenuEmpleado" />
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area=>
    </li>
</ul>
iv>

```

Decoramos con seguridad el IActionResult de Index en EmpleadosController

EMPLEADOSCONTROLLER

[AuthorizeEmpleados]

0 references

```

public async Task<IActionResult> Index()
{

```

Por último, configuraremos la seguridad en Program

## PROGRAM

```
using Microsoft.AspNetCore.Authentication.Cookies;
using MvcOAuthEmpleados.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddTransient<ServiceEmpleados>();
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromMinutes(30);
});

builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme =
        CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme =
        CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultSignInScheme =
        CookieAuthenticationDefaults.AuthenticationScheme;
}).AddCookie();

builder.Services.AddControllersWithViews
    (options => options.EnableEndpointRouting = false);

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.UseSession();
app.UseMvc(routes =>
{
    routes.MapRoute(name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
//app.MapControllerRoute(
//    name: "default",
//    pattern: "{controller=Home}/{action=Index}/{id?}")
//    .WithStaticAssets();
}

app.Run();
```

Ya tenemos montada la seguridad, pero existe algo que chirria todavía.

Session y seguridad van por sitios distintos...

Tenemos que recuperar de forma explícita el Token

```
public async Task<IActionResult> Details(int id)
{
    //TENDREMOS EL TOKEN EN SESSION
    string token = HttpContext.Session.GetString("TOKEN");
```

¿Dónde deberíamos guardar el Token?

En los Claims deberíamos guardar el Token porque pertenece al usuario.

## MANAGEDCONTROLLER

```
identity.AddClaim(new Claim
    (ClaimTypes.NameIdentifier, model.Pass
//ALMACENAMOS EL TOKEN EN EL USUARIO
identity.AddClaim(new Claim
    ("TOKEN", token));
ClaimsPrincipal principal =
    new ClaimsPrincipal(identity);
```

Hasta ayer no sabíamos ni lo que era generar un Token.

Hemos mezclado elementos solamente por el Token:

Si miramos en nuestro Servicio:

¿Cuándo hemos enviado el token a un método de búsqueda? NUNCA

El método debería seguir el patrón de acceso, es decir, si buscamos un empleado por ID,

Solamente debería recibir el ID y ya recuperamos los datos de validación de donde sea.

```
public async Task<Empleado> FindEmpleadoAsync  
    (int idEmpleado, string token)  
{
```

Modificamos el método del Servicio:

```
public async Task<Empleado> FindEmpleadoAsync  
    (int idEmpleado)  
{
```

En el controlador, nada de Token ni nada de eso, simplemente llamamos a nuestro Service de Forma natural.

#### EMPLEADOSCONTROLLER

```
0 references  
public async Task<IActionResult> Details(int id)  
{  
    Empleado empleado = await  
        this.service.FindEmpleadoAsync(id);  
    return View(empleado);  
}
```

El siguiente paso es recuperar el User y su Claim con el Token dentro del Service.

Necesitamos esta línea (conocida) dentro del Service:

```
HttpContext.User.FindFirst(x => x.Type == "TOKEN");
```

Debemos inyectar, dentro del Service `IHttpContextAccessor`

#### SERVICEEMPLEADOS

```
public class ServiceEmpleados  
{  
    private string UrlApi;  
    private MediaTypeWithQualityHeaderValue Header;  
    private IHttpContextAccessor contextAccessor;  
  
    0 references  
    public ServiceEmpleados(IConfiguration configuration  
        , IHttpContextAccessor contextAccessor)  
    {  
        this.contextAccessor = contextAccessor;  
    }
```

Tenemos dos opciones, recuperar el token desde `CallApiAsync` protegido.  
Recuperar el token desde cada método Protegido.

```
public async Task<Empleado> FindEmpleadoAsync  
    (int idEmpleado)  
{  
    string token =  
        this.contextAccessor.HttpContext.User  
            .FindFirst(z => z.Type == "TOKEN").Value;  
    string request = "api/empleados/" + idEmpleado;  
    Empleado empleado = await  
        this.CallApiAsync<Empleado>(request, token);  
    return empleado;  
}
```

Debemos realizar la inyección de `HttpContextAccessor` desde `Program`

#### PROGRAM

```

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddHttpContextAccessor();

```

Lo siguiente que me gustaría es tener un Perfil del usuario validado.

Pongamos el siguiente ejemplo:

```

public async Task<IActionResult> PerfilEmpleado()
{
    //NECESITAMOS BUSCAR AL EMPLEADO QUE HA REALIZADO
    //LOGIN
    var data = HttpContext.User.FindFirst
        (x => x.Type == ClaimTypes.NameIdentifier).Value;
    int id = int.Parse(data);
    Empleado empleado = await
        this.service.FindEmpleadoAsync(id);
    return View(empleado);
}

```

Pongamos que quiero saber los compis de trabajo del empleado que se ha dado de alta.

```

public async Task<IActionResult> Compis()
{
    //NECESITAMOS BUSCAR AL EMPLEADO QUE HA REALIZADO
    //LOGIN
    var data = HttpContext.User.FindFirst
        (x => x.Type == "DEPARTAMENTO").Value;
    int idDept = int.Parse(data);
    List<Empleado> empleados = await
        this.service.GetCompisAsync(idDept);
    return View(empleados);
}

```

Para estos dos supuestos que acabo de comentar, tenemos que estar dando rodeos.

Actualmente, tenemos un Api que nos impide acceder a los datos del usuario.

¿Y si pudiera acceder a los datos del usuario con el Token?

En nuestro clientes (MVC Net Core) nunca podremos acceder a los datos del Token

El Api si que puede recuperar información del token, ya que se lo están enviando a los Métodos del Api.

Tenemos la posibilidad de almacenar información dentro del Token cifrado.

También podemos recuperar información dentro de dicho Token, pero en el Api

DENTRO DE NUESTRO API

```

//API
[Authorize]
0 references
public async Task<Empleado> Perfil()
{
    //ESTE METODO ESTARA PROTEGIDO DENTRO DEL API
    //AQUI PUEDO RECUPERAR EL EMPLEADO DEL TOKEN O
    //LO QUE YO QUIERA DEL TOKEN
    Empleado empleado = token.Empleado;
    return empleado;
}

```

Abrimos el Api de seguridad de empleados

Instalamos NewtonSoft

 **Newtonsoft.Json** by dotnetfoundation, jamesnk, newtonsoft, 6B downloads 13.0.3  
Json.NET is a popular high-performance JSON framework for .NET

Lo que vamos a realizar es almacenar "algo" dentro del Token.  
 Puedo almacenar lo que yo necesite mediante Claims  
 En un Claim podemos almacenar un NameIdentifier, el Id del departamento, el Apellido,  
 El oficio o, podemos almacenar TODO lo que necesitemos dentro del Claim en formato JSON

El almacenamiento de Claims se realiza en el Api de la siguiente forma al generar el Token

```
Claim[] informacion = new []
{
    new Claim("USUARIO", usuario), new Claim("DEPARTAMENTO", departamento)
}
```

```
JwtSecurityToken token =
    new JwtSecurityToken(
        claims: informacion,
```

AUTHCONTROLLER

```
string jsonEmpleado =
    JsonConvert.SerializeObject(empleado);
//CREAMOS UN ARRAY DE CLAIMS
Claim[] informacion = new []
{
    new Claim("UserData", jsonEmpleado)
};
//EL TOKEN SE GENERA CON UNA CLASE
//Y DEBEMOS INDICAR LOS DATOS QUE ALMACENARA EN SU
//INTERIOR
JwtSecurityToken token =
    new JwtSecurityToken(
        claims: informacion,
```

La gran ventaja está en que podemos acceder al Claim dentro de cualquier método del Api que  
 Esté decorado con **[Authorize]**

Vamos a crear un par de métodos para recuperar el Perfil del Empleado y sus Compis

REPOSITORYHOSPITAL

```
public async Task<Empleado>
    FindEmpleadoAsync(int idEmpleado)
{
    return await this.context.Empleados
        .FirstOrDefaultAsync(z => z.IdEmpleado == idEmpleado);
}

public async Task<List<Empleado>>
    GetCompisEmpleadoAsync(int idDepartamento)
{
    return await this.context.Empleados
        .Where(x => x.IdDepartamento == idDepartamento)
        .ToListAsync();
}
```

Implementamos y modificamos los métodos dentro de EmpleadosController

EMPLEADOSCONTROLLER

```
[Authorize]
[HttpGet]
[Route("[action]")]
public async Task<ActionResult<Empleado>>
    Perfil()
{
    Claim claim = HttpContext.User.FindFirst
        (z => z.Type == "UserData");
    string json = claim.Value;
    Empleado empleado = JsonConvert
        .DeserializeObject<Empleado>(json);
    return await
        this.repo.FindEmpleadoAsync(empleado.IdEmpleado);
}

[Authorize]
[HttpGet]
[Route("[action]")]
public async Task<ActionResult<List<Empleado>>>
    Compis()
{
    string json = HttpContext.User.FindFirst
        (x => x.Type == "UserData").Value;
    Empleado empleado = JsonConvert
        .DeserializeObject<Empleado>(json);
    return await this.repo.GetCompisEmpleadoAsync(
        empleado.IdDepartamento);
}
```

```
}
```

Publicamos nuestro Api en Azure y volvemos a Mvc Net Core

Escribimos los nuevos métodos dentro de **ServiceEmpleados**

#### SERVICEEMPLEADOS

```
public async Task<Empleado> GetPerfilAsync()
{
    string token =
        this.contextAccessor.HttpContext.User
            .FindFirst(x => x.Type == "TOKEN").Value;
    string request = "api/empleados/perfil";
    Empleado empleado = await
        this.CallApiAsync<Empleado>(request, token);
    return empleado;
}

public async Task<List<Empleado>> GetCompisAsync()
{
    string token =
        this.contextAccessor.HttpContext.User
            .FindFirst(x => x.Type == "TOKEN").Value;
    string request = "api/empleados/compis";
    List<Empleado> empleados = await
        this.CallApiAsync<List<Empleado>>(request, token);
    return empleados;
}
```

#### EMPLEADOSCONTROLLER

```
public async Task<IActionResult> Perfil()
{
    Empleado empleado = await
        this.service.GetPerfilAsync();
    return View(empleado);
}

[AuthorizeEmpleados]
public async Task<IActionResult> Compis()
{
    List<Empleado> empleados = await
        this.service.GetCompisAsync();
    return View(empleados);
}
```

Dentro de **\_MenuEmpleado.cshtml** creamos los nuevos Links

#### \_MENUEMPLEADO.CSHTML

```
@if (Context.User.Identity.IsAuthenticated){
    <li class="nav-item">
        <a class="nav-link text-danger"
            asp-controller="Empleados"
            asp-action="Index">
            Empleados
        </a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-danger"
            asp-controller="Empleados"
            asp-action="Perfil">
            Perfil
        </a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-danger"
            asp-controller="Empleados"
            asp-action="Compis">
            Compis
        </a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-danger"
            asp-controller="Managed"
            asp-action="Logout">
            Log Out <span style="color:blue">
                @Context.User.Identity.Name
            </span>
        </a>
    </li>
} else{
    <li class="nav-item">
        <a class="nav-link text-black"
            asp-controller="Empleados"
            asp-action="Index">
            Log In
        </a>
    </li>
}
```

Y ya tendremos nuestra aplicación funcional



Home Empleados Perfil Compis Log Out REY Privacy

## Perfil



### Empleado

<b>IdEmpleado</b>	7839
<b>Apellido</b>	REY
<b>Oficio</b>	PRESIDENTE
<b>Salario</b>	650000
<b>IdDepartamento</b>	10

Y los compis también



Home Empleados Perfil Compis Log Out REY Privacy

## Compis

Apellido	Oficio	Salario	IdDepartamento
CEREZO	DIRECTOR	318500	10
REY	PRESIDENTE	650000	10
MUÑOZ	EMPLEADO	169000	10
CASALES	EMPLEADO	179000	10
ALCALA	EMPLEADO	119000	10

El siguiente paso es abrir esta Web

<https://iwt.io/>

Debemos cifrar la información sensible a su vez mediante HelperCifrado de doble dirección para Cifrar y descifrar

#### EJEMPLO METODOS DE CIFRADO PARA EL HELPER

```
private string EncryptString(string key, string plainText)
{
    byte[] iv = new byte[16];
    byte[] array;

    using (Aes aes = Aes.Create())
    {
        aes.Key = Encoding.UTF8.GetBytes(key);
        aes.IV = iv;

        ICryptoTransform encryptor = aes.CreateEncryptor(aes.Key,
aes.IV);

        using (MemoryStream memoryStream = new MemoryStream())
        {
            using (CryptoStream cryptoStream = new CryptoStream((Stream)
memoryStream, encryptor, CryptoStreamMode.Write))
            {
                using (StreamWriter streamWriter = new
StreamWriter((Stream)cryptoStream))
                {
                    streamWriter.Write(plainText);
                }
                array = memoryStream.ToArray();
            }
        }
    }
}
```

```

        }
    }

    return Convert.ToBase64String(array);
}

private string DecryptString(string key, string cipherText)
{
    byte[] iv = new byte[16];
    byte[] buffer = Convert.FromBase64String(cipherText);

    using (Aes aes = Aes.Create())
    {
        aes.Key = Encoding.UTF8.GetBytes(key);
        aes.IV = iv;
        ICryptoTransform decryptor = aes.CreateDecryptor(aes.Key,
aes.IV);

        using (MemoryStream memoryStream = new MemoryStream(buffer))
        {
            using (CryptoStream cryptoStream = new CryptoStream((Stream)
memoryStream, decryptor, CryptoStreamMode.Read))
            {
                using (StreamReader streamReader = new
StreamReader((Stream)cryptoStream))
                {
                    return streamReader.ReadToEnd();
                }
            }
        }
    }
}

```

Por último, si necesitamos Roles en el Api, es muy simple, no tenemos que configurar nada

Solamente incluir los Roles de los usuarios dentro de sus Claims.

#### AUTHCONTROLLER

```
//CREAMOS UN ARRAY DE CLAIMS
Claim[] informacion = new[]
{
    new Claim("UserData", jsonEmpleado),
    new Claim(ClaimTypes.Role, empleado.Oficio)
};
```

Posteriormente, en cualquier controller, ya podremos filtrar por Role de usuario.

#### EMPLEADOSCONTROLLER

```
[Authorize(Roles = "PRESIDENTE")]
[HttpGet]
[Route("[action]")]
public async Task<ActionResult<List<Empleado>>>
    Compis()
{}
```

Si necesitamos que un usuario pueda entrar en varios sitios y solamente podemos poner Un Role en Authorize, simplemente, se le añaden los Roles.

Por ejemplo, si tuvieramos un usuario ADMINISTRADOR, actualmente no podría entrar En la zona del Presidente.

Lo único que tendríamos que hacer es incluir al Role de ADMINISTRADOR, el Role del PRESIDENTE.

#### AUTHCONTROLLER

```

if (empleado.Oficio == "ADMINISTRADOR")
{
    //CREAMOS UN ARRAY DE CLAIMS
    Claim[] informacion = new[]
    {
        new Claim("UserData", jsonEmpleado),
        new Claim(ClaimTypes.Role, "PRESIDENTE"), I
        new Claim(ClaimTypes.Role, "ADMINISTRADOR")
    };
}

```

Por último, aunque no tenga que ver con la seguridad, nos queda visualizar un último Aspecto dentro de los Apis.

Por ejemplo, algo al estilo del "Carrito de la compra", cuando necesitamos recibir Múltiples valores dentro de un método del Api.

#### POR EJEMPLO

```

public async Task<List<Empleado>>
    GetMultiplesEmpleadosAsync(List<int> ids)
{
    var consulta = from datos in this.context.Empleados
                  where ids.Contains(datos.IdEmpleado)
                  select datos;
    return await consulta.ToListAsync();
}

```

Y cómo podemos llamar a esto y enviar la información desde un método del Api?

Como lo hacemos con Route??

Necesitamos utilizar una decoración llamada **[FromQuery]**

Dicha decoración nos permite poder recibir múltiples datos en un método de nuestro Controller

```

[HttpGet]
[Route("[action]")]
public async Task<ActionResult>
    MultiplesValores
    ([FromQuery] List<int> datos)
{
    return Ok();
}

```

La información es enviada de forma "tradicional":

<https://localhost:7866/api/controller/multiplesvalores?datos=2&datos=77&datos=88>

Vamos a realizar un método con Empleados para probar esto.  
Tendremos un método que recibirá múltiples Oficios FromQuery y debemos devolver Los empleados que cumplan con los Oficios enviados.

#### REPOSITORYHOSPITAL

```

public async Task<List<string>> GetOficiosAsync()
{
    var consulta = (from datos in this.context.Empleados
                   select datos.Oficio).Distinct();
    return await consulta.ToListAsync();
}

public async Task<List<Empleado>>
    GetEmpleadosByOficiosAsync(List<string> oficios)
{
    var consulta = from datos in this.context.Empleados
                  where oficios.Contains(datos.Oficio)
                  select datos;
    return await consulta.ToListAsync();
}

```

```

public async Task IncrementarSalariosAsync
    (int incremento, List<string> oficios)
{
    List<Empleado> empleados = await
        this.GetEmpleadosByOficiosAsync(oficios);
    foreach (Empleado emp in empleados)
    {
        emp.Salario += incremento;
    }
}

EMPLEADOSCONTROLLER

[HttpGet]
[Route("[action]")]
public async Task<ActionResult<List<string>>>
    Oficios()
{
    return await this.repo.GetOficiosAsync();
}

//?oficio=ANALISTA&oficio=DIRECTOR
[HttpGet]
[Route("[action]")]
public async Task<ActionResult<List<Empleado>>>
    EmpleadosOficios([FromQuery] List<string> oficio)
{
    return await this.repo.GetEmpleadosByOficiosAsync
        (oficio);
}

[HttpPost]
[Route("{action}/{incremento}")]
public async Task<ActionResult> IncrementarSalarios
    (int incremento, [FromQuery] List<string> oficio)
{
    await this.repo.IncrementarSalariosAsync
        (incremento, oficio);
    return Ok();
}

```

/empleadosoficios?oficio=ANALISTA&oficio=DIRECTOR

Send		200 OK	2.22 s	Just Now
Body	Auth Headers (3) Scripts Docs	Preview Headers (3) Cookies		
pt	/* : : <calculated at runtime>	Preview		
Agent	insomnia/11.0.1	1 * [ 2 * { 3 "idEmpleado": 7566, 4 "apellido": "JIMENEZ", 5 "oficio": "DIRECTOR", 6 "salario": 386750, 7 "idDepartamento": 20 8 }, 9 * { 10 "idEmpleado": 7698, 11 "apellido": "NEGRO", 12 "oficio": "DIRECTOR", 13 "l " : " 270500		

/Incrementarsalarios/1?oficio=ANALISTA&oficio=DIRECTOR

Send		200 OK	610 ms	Just Now
Body	Auth Headers (3) Scripts Docs	Preview Headers (3) Cookies		
cept	/*	Preview		
ost	<calculated at runtime>	No body returned for response		
er-Agent	insomnia/11.0.1			

Publicamos los cambios y probamos dentro del proyecto MVC Core

#### SERVICEEMPLEADOS

```

public async Task<List<string>> GetOficiosAsync()
{
    string request = "api/empleados/oficios";

```

```

        List<string> oficios =
            await this.CallApiAsync<List<string>>(request);
        return oficios;
    }

//oficio=ANALISTA&oficio=DIRECTOR
private string TransformCollectionToQuery(List<string> collection)
{
    string result = "";
    foreach (string elem in collection)
    {
        result += "oficio=" + elem + "&";
    }
    result = result.TrimEnd('&');
    return result;
}

public async Task<List<Empleado>>
    GetEmpleadosOficiosAsync(List<string> oficios)
{
    string request = "api/empleados/empleadosoficios";
    string data = this.TransformCollectionToQuery(oficios);
    List<Empleado> empleados =
        await this.CallApiAsync<List<Empleado>>
            (request + "?" + data);
    return empleados;
}

public async Task UpdateEmpleadosOficioAsync
    (int incremento, List<string> oficios)
{
    string request = "api/empleados/incrementarsalarios/" +
        incremento;
    string data = this.TransformCollectionToQuery
        (oficios);
    using (HttpClient client = new HttpClient())
    {
        client.BaseAddress = new Uri(this.UrlApi);
        client.DefaultRequestHeaders.Clear();
        client.DefaultRequestHeaders.Accept.Add(this.Header);
        HttpResponseMessage response =
            await client.PutAsync(request + "?" + data, null);
    }
}

```

A continuación, creamos los métodos dentro de **EmpleadosController**

Tendremos una lista para seleccionar los oficios de forma múltiple.  
 Dos botones, uno para mostrar y otro para incrementar salarios.

## Empleados Oficios

Seleccione oficio/s

- ANALISTA
- DIRECTOR
- EMPLEADO
- PRESIDENTE
- VENDEDOR

Incremento salarial

[Mostrar empleados](#) [Incrementar salarios](#)

IdEmpleado	Apellido	Oficio	Salario	IdDepartamento
7566	JIMENEZ	DIRECTOR	386772	20
7698	NEGRO	DIRECTOR	370522	30
7782	CEREZO	DIRECTOR	318522	10
7788	GIL	ANALISTA	390022	20
7902	FFRNANDFZ	ANALISTA	390022	20

### EMPLEADOSCONTROLLER

```

public async Task<IActionResult>
    EmpleadosOficios()
{
    List<string> oficios = await
        this.service.GetOficiosAsync();
    ViewData["OFICIOS"] = oficios;
    return View();
}

[HttpPost]
public async Task<IActionResult>
    EmpleadosOficios(int? incremento
    , List<string> oficio, string accion)
{
    List<string> oficios = await this.service.GetOficiosAsync();
    ViewData["OFICIOS"] = oficios;
    if (accion.ToLower() == "update")
    {
        await this.service.UpdateEmpleadosOficioAsync
            (incremento.Value, oficio);
    }
    List<Empleado> empleados = await
        this.service.GetEmpleadosOficiosAsync(oficio);
    return View(empleados);
}

```

## EMPLEADOSOFICIOS.CSHTML

```
@model IEnumerable<MvcOAuthEmpleados.Models.Empleado>

@{
    ViewData["Title"] = "EmpleadosOficios";
}

@{
    List<string> oficios =
    ViewData["OFICIOS"] as List<string>;
}

<h1>Empleados Oficios</h1>

<form method="post">
    <label>Selecciona oficio/s</label>
    <select name="oficio" size="5" multiple class="form-control">
        @foreach (string ofi in oficios){
            <option value="@ofi">@ofi</option>
        }
    </select>
    <label>Incremento salarial</label>
    <input type="text" name="incremento" class="form-control"/>
    <button name="accion" value="read" class="btn btn-info">
        Mostrar empleados
    </button>
    <button name="accion" value="update" class="btn btn-warning">
        Incrementar salarios
    </button>
</form>

@if (Model != null){
    <table class="table table-info">
        <thead>
            <tr>
                <th>
                    @Html.DisplayNameFor(model => model.IdEmpleado)
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.Apellido)
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.Oficio)
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.Salario)
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.IdDepartamento)
                </th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td>
                        @Html.DisplayFor(modelItem => item.IdEmpleado)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.Apellido)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.Oficio)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.Salario)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.IdDepartamento)
                    </td>
                </tr>
            }
        </tbody>
    </table>
}
}
```

Vamos a cifrar los datos de los empleados en el Token.

## HELPERCRYPTOGRAPHY

```
public static class HelperCryptography
{
    private static IConfiguration configuration;
    private static string keyCifrado;
    public static void Initialize(IConfiguration config)
    {
        configuration = config;
        keyCifrado = configuration.GetValue<string>("ApiOAuth:cryptoKey");
    }

    public static string EncryptString(String dato)
    {
        var saltconf = configuration.GetValue<string>("Crypto:Salt");
        var bucleconf = configuration.GetValue<string>("Crypto:Iterate");
        string password = configuration.GetValue<string>("Crypto:Key");
        byte[] saltpassword = EncryptarPasswordSalt
            (password, saltconf, int.Parse(bucleconf));
        String res = EncryptString(saltpassword, dato);
        return res;
    }

    public static string DecryptString(String dato)
    {
        var saltconf = configuration.GetValue<string>("Crypto:Salt");
        var bucleconf = configuration.GetValue<string>("Crypto:Iterate");
        string password = configuration.GetValue<string>("Crypto:Key");
        byte[] saltpassword = EncryptarPasswordSalt
            (password, saltconf, int.Parse(bucleconf));
    }
}
```

```

        String res = DecryptString(saltpassword, dato);
        return res;
    }

private static string EncryptString(byte[] key, string plainText)
{
    byte[] iv = new byte[16];
    byte[] array;

    using (Aes aes = Aes.Create())
    {
        aes.Key = key;
        aes.IV = iv;

        ICryptoTransform encryptor = aes.CreateEncryptor(aes.Key, aes.IV);

        using (MemoryStream memoryStream = new MemoryStream())
        {
            using (CryptoStream cryptoStream = new CryptoStream((Stream)
memoryStream, encryptor, CryptoStreamMode.Write))
            {
                using (StreamWriter streamWriter = new StreamWriter((Strea
m)cryptoStream))
                {
                    streamWriter.WriteLine(plainText);
                }

                array = memoryStream.ToArray();
            }
        }
    }

    return Convert.ToBase64String(array);
}

private static byte[] EncriptarPasswordSalt(string contenido
, string salt, int numhash)
{
    //REALIZAMOS LA COMBINACION DE ENCRYPTADO
    //CON SU SALT
    string textocompleto = contenido + salt;
    //DECLARAMOS EL OBJETO SHA256
    //SHA256Managed objsha = new SHA256Managed();
    SHA256 objsha = SHA256.Create();
    byte[] bytesalida = null;

    try
    {
        //CONVERTIMOS EL TEXTO A BYTES
        bytesalida =
            Encoding.UTF8.GetBytes(textocompleto);
        //Convert.FromBase64String(textocompleto);
        //ENCRYPTAMOS EL TEXTO 1000 VECES
        for (int i = 0; i < numhash; i++)
            bytesalida = objsha.ComputeHash(bytesalida);
    }
    finally
    {
        objsha.Clear();
    }
    //DEVOLVEMOS LOS BYTES DE SALIDA
    return bytesalida;
}

private static string DecryptString(byte[] key, string cipherText)
{
    byte[] iv = new byte[16];
    byte[] buffer = Convert.FromBase64String(cipherText);

    using (Aes aes = Aes.Create())
    {
        aes.Key = key;
        aes.IV = iv;
        ICryptoTransform decryptor = aes.CreateDecryptor(aes.Key, aes.IV);

        using (MemoryStream memoryStream = new MemoryStream(buffer))
        {
            using (CryptoStream cryptoStream = new CryptoStream((Stream)
memoryStream, decryptor, CryptoStreamMode.Read))
            {
                using (StreamReader streamReader = new StreamReader((Strea
m)cryptoStream))
                {
                    return streamReader.ReadToEnd();
                }
            }
        }
    }
}

```

#### CONFIGURACIÓN

```

"Crypto": {
    "Salt": "ab$ed54",
    "Key": "P_Dff542312@==",
    "Iterate": 2
}

```

En Program inicializamos las variables para los cifrados

```

var builder = WebApplication.CreateBuilder(args);

HelperCryptography.Initialize(builder.Configuration);
//INYECTAMOS HttpContextAccessor
builder.Services.AddHttpContextAccessor();

```

A continuación, nunca deberíamos tener datos confidenciales dentro del Token, por ejemplo, El Password del usuario.

Nosotros, dentro del Empleado, tenemos el Salario porque es necesario para nuestra Aplicación en la base de datos, pero ese dato no debería ser expuesto.

Por un lado, la clase Empleado que será la del Repository para BBDD  
Por otro lado una clase **EmpleadoModel** con los datos a incluir en el Token (sin Salario).

#### EMPLEADOMODEL

```

public class EmpleadoModel
{
    0 references | 0 changes | 0 authors, 0 changes
    public int IdEmpleado { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Apellido { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Oficio { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public int IdDepartamento { get; set; }
}

```

Cuando vayamos a cifrar al empleado, lo convertimos a nuestro Model.

#### AUTHCONTROLLER en Login

```

//CREAMOS EL OBJETO MODEL PARA ALMACENARLO
//DENTRO DEL TOKEN
EmpleadoModel modelEmp = new EmpleadoModel();
modelEmp.IdEmpleado = empleado.IdEmpleado;
modelEmp.Apellido = empleado.Apellido;
modelEmp.Oficio = empleado.Oficio;
modelEmp.IdDepartamento = empleado.IdDepartamento;
//CONVERTIMOS A JSON LOS DATOS DEL EMPLEADO
string jsonEmpleado =
    JsonConvert.SerializeObject(modelEmp);
//CREAMOS UN ARRAY DE CLAIMS
Claim[] informacion = new[]
{
    new Claim("UserData", jsonEmpleado)
};

```

El siguiente paso es crear un método de ayuda que nos devolverá el usuario a partir Del Token almacenado. (EmpleadoModel)

En las clases que tengan seguridad, inyectamos el Helper.

#### HELPEREMPLEADOTOKEN

```

public class HelperEmpleadoToken
{
    private IHttpContextAccessor contextAccessor;
    public HelperEmpleadoToken(IHttpContextAccessor contextAccessor)
    {
        this.contextAccessor = contextAccessor;
    }
    public EmpleadoModel GetEmpleado()
    {
        Claim claim =
            this.contextAccessor.HttpContext
                .User.FindFirst(x => x.Type == "UserData");
        string json = claim.Value;
    }
}

```

```

        EmpleadoModel model = JsonConvert
            .DeserializeObject<EmpleadoModel>(json);
        return model;
    }
}

```

Simplemente, en las clases Controller que tengan seguridad, inyectamos nuestro Nuevo Helper.

#### EMPLEADOSCONTROLLER

```

public class EmpleadosController : ControllerBase
{
    private RepositoryHospital repo;
    private HelperEmpleadoToken helper;
    0 references | 0 changes | 0 authors, 0 changes
    public EmpleadosController
        (RepositoryHospital repo
        , HelperEmpleadoToken helper)
    {
        this.helper = helper;
        this.repo = repo;
    }
}

```

#### EMPLEADOSCONTROLLER

```

[Authorize]
[HttpGet]
[Route("[action]")]
public async Task<ActionResult<Empleado>>
    Perfil()
{
    EmpleadoModel model = this.helper.GetEmpleado();
    return await
        this.repo.FindEmpleadoAsync(model.IdEmpleado);
}

//#[Authorize(Roles = "PRESIDENTE")]
[Authorize]
[HttpGet]
[Route("[action]")]
public async Task<ActionResult<List<Empleado>>>
    Compis()
{
    EmpleadoModel model = this.helper.GetEmpleado();
    return await this.repo.GetCompisEmpleadoAsync(
        (model.IdDepartamento));
}

```

Una vez que hemos visto que es funcional, simplemente Ciframos los datos del UserData al generar el Token y los desciframos al recuperar el User.

#### AUTHCONTROLLER

```

        JsonConvert.DeserializeObject(Claims),
        string jsonCifrado =
            HelperCryptography.EncryptString(jsonEmpleado);
    //CREAMOS UN ARRAY DE CLAIMS
    Claim[] informacion = new[]
    {
        new Claim("UserData", jsonCifrado)
    };
}

```

#### HELPEREMPLEADOTOKEN

```
public EmpleadoModel GetEmpleado()
{
    Claim claim =
        this.contextAccessor.HttpContext
            .User.FindFirst(x => x.Type == "UserData");
    string json = claim.Value;
    string jsonEmpleado =
        HelperCryptography.DecryptString(json);
    EmpleadoModel model = JsonConvert
        .DeserializeObject<EmpleadoModel>(jsonEmpleado);
    return model;
}
```

## AZURE STORAGE

Junes, 31 de marzo de 2025 10:58

El servicio Azure Storage nos permite tener múltiples herramientas dentro de un mismo Servicio.

En realidad, sirve para centralizar ficheros y alguna característica más dentro del propio Servicio.

Los servicios que tenemos son los siguientes:

- **Azure Storage Files:** Se utiliza para almacenar ficheros de contenido texto. También podemos almacenar Files, pero para eso tenemos otro servicio
- **Azure Storage Blobs:** Se utiliza para almacenar ficheros en modo Byte, Está mucho mejor su rendimiento cuando quiero almacenar, facturas, videos, imágenes.
- **Azure Storage Tables:** Son tablas de información que almacenan contenido NoSQL como, por ejemplo, **Mongo Db**. Es la versión 1.0 de un servicio llamado **Cosmos Db**, que es la base de datos NoSQL oficial de Microsoft.
- **Azure Storage Queue:** Es el servicio de colas de mensajes. Es la versión 1.0. De otro servicio más potente llamado **Service Bus**

Comenzamos creando una cuenta de Azure Storage

En dicha cuenta nos van a cobrar por almacenamiento, no es gratis.

Para vuestro proyecto, os podéis crear una cuenta Storage sin problemas.

**Nota:** Existe un emulador de Azure que nos permite realizar nuestras pruebas en local Y control de código sobre Windows.

Actualmente, no la vamos a eliminar, porque seguiremos utilizando sus características  
Creamos la cuenta Storage sobre **rg-tajamar**

**Nota:** El nombre de la cuenta debe ser **UNICO**

**storagetajamarpgs**

The screenshot shows the 'Instance details' step of the Azure Storage account creation process. It includes fields for Storage account name (set to 'storagetajamarpgs'), Region (set to '(Europe) Spain Central'), Primary service (set to 'Azure Blob Storage or Azure Data Lake Storage Gen 2'), Performance (set to 'Standard: Recommended for most scenarios (general-purpose v2 account)'), and Redundancy (set to 'Locally-redundant storage (LRS)').

Vamos a comenzar con Azure Files

Los Files son almacenados dentro de recursos compartidos llamados **Shared**

Los recursos compartidos, a su vez, pueden tener subcarpetas dentro de un Shared

Tenemos que pensar en Shared como en una carpeta raíz.

Vamos a crear un Shared en Azure y subimos un fichero para verlo (TXT)

The screenshot shows the 'New file share' creation wizard on the 'Basics' tab. It includes fields for Name (set to 'ejemplofiles') and Access tier (set to 'Transaction optimized'). Below these, there's a 'Performance' section with Maximum IO/s (set to 20000) and Maximum capacity (set to 100 TiB).

<a href="#">Copy to clipboard</a>	
Storage account	: storagetajamarpgs
Resource group (...)	: rg-tajamar
Location	: spaincentral
Subscription (move)	: Azure for Students
Subscription ID	: 02ee10d4-2d35-43df-89fb-d7ac1e185646
Share URL	: https://storagetajamarpgs.file.core.windows.net/ejem...
Redundancy	: Locally-redundant storage (LRS)
Configuration modifi...	: 31/3/2025, 11:31:27

Los Files no son libres aunque tengamos una URL de Acceso.

Necesitamos **Access Keys** para acceder a los servicios Storage.

Creamos una nueva aplicación llamada **MvcCoreAzureStorage**

- Listar ficheros de Shared
- Subir ficheros al server
- Leer ficheros
- Eliminar ficheros

The screenshot shows the Azure Storage Files interface. At the top, there's a navigation bar with a profile icon, 'Home', 'Azure Files', and 'Privacy'. Below it, a large blue header says 'Azure storage Files'. Underneath, there's a section titled 'Upload Azure File'. Two files are listed: 'scriptseriesoracle.txt' and 'scriptTEST.sql'. Each file entry includes a 'Read File' button and a 'Delete file' button.

The screenshot shows the GitHub page for the 'Azure.Storage.FilesShares' package. It includes the package icon, name, download count (39,8M), version (12.22.0), and a brief description: 'This client library enables working with the Microsoft Azure Storage File Shares service for storing binary and text data.' A link to the release notes is also provided.

Necesitamos las **Access Keys** de Storage

The screenshot shows the 'Access keys' page for a storage account. It displays two keys: 'key1' and 'key2'. Key 1 was last rotated on 31/3/2025 (0 days ago). The key itself is shown as a series of dots. There are 'Show' and 'Hide' buttons next to the key fields. Below the keys, a 'Connection string' field contains the protocol and account information. A note at the bottom says 'Remember to update the keys with any Azure resources and apps that use this storage account. Learn more about managing storage account access keys'.

Vamos a almacenarlas dentro de **APPSETTINGS.JSON**

The screenshot shows the 'APPSETTINGS.JSON' file in a code editor. The file contains a 'AzureKeys' section with a 'StorageAccount' key pointing to a connection string. The JSON structure is as follows:

```

{
  "AllowedHosts": "*",
  "AzureKeys": {
    "StorageAccount": "DefaultEndpointsProtocol=https;AccountName=storagetajamarpgs;AccountKey=... "
  }
}
  
```

Creamos una carpeta llamada **Services** y una clase llamada **ServiceStorageFiles**

#### SERVICESTORAGEFILES

```
public class ServiceStorageFiles
{
    //TODO SERVICIO STORAGE SIEMPRE UTILIZA CLIENTS
    //PARA TRABAJAR
    //UN CLIENT PUEDE SER DIRECTAMENTE SOBRE UN SHARED O
    //PODRIA SER SOBRE TODO EL SERVICIO DE FILES
    private ShareDirectoryClient root;

    public ServiceStorageFiles(IConfiguration configuration)
    {
        string keys = configuration.GetValue<string>("AzureKeys:StorageAccount");
        //NUESTRO CLIENTE TRABAJARA SOBRE UN SHARED
        //DETERMINADO
        ShareClient client =
            new ShareClient(keys, "ejemplofiles");
        this.root = client.GetRootDirectoryClient();
    }

    //METODO PARA RECUPERAR TODOS LOS FICHEROS
    //DE LA RAIZ DE SHARED
    public async Task<List<string>>
        GetFilesAsync()
    {
        List<string> files =
            new List<string>();
        await foreach(ShareFileItem item in
            this.root.GetFilesAndDirectoriesAsync())
        {
            files.Add(item.Name);
        }
        return files;
    }

    public async Task<string> ReadFileAsync(string fileName)
    {
        ShareFileClient fileClient =
            this.root.GetFileClient(fileName);
        ShareFileDownloadInfo data =
            await fileClient.DownloadAsync();
        Stream stream = data.Content;
        string contenido = "";
        using (StreamReader reader = new StreamReader(stream))
        {
            contenido = await reader.ReadToEndAsync();
        }
        return contenido;
    }

    public async Task UploadFileAsync(
        string fileName, Stream stream)
    {
        ShareFileClient fileClient =
            this.root.GetFileClient(fileName);
        await fileClient.CreateAsync(stream.Length);
        await fileClient.UploadAsync(stream);
    }

    public async Task DeleteFileAsync(string fileName)
    {
        ShareFileClient fileClient =
            this.root.GetFileClient(fileName);
        await fileClient.DeleteAsync();
    }
}
```

Creamos un nuevo controlador llamado **AzureFilesController**

#### AZUREFILESCONTROLLER

```
public class AzureFilesController : Controller
{
    private ServiceStorageFiles service;

    public AzureFilesController(ServiceStorageFiles service)
    {
        this.service = service;
    }

    public async Task<IActionResult> Index()
    {
        List<string> files = await this.service.GetFilesAsync();
        return View(files);
    }

    public async Task<IActionResult>
        ReadFile(string filename)
    {
        string data =
            await this.service.ReadFileAsync(filename);
        ViewData["DATA"] = data;
        return View();
    }

    public async Task<IActionResult> DeleteFile(string filename)
    {
        await this.service.DeleteFileAsync(filename);
        return RedirectToAction("Index");
    }

    public IActionResult UploadFile()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult>
        UploadFile(IFormFile file)
    {
        string fileName = file.FileName;
        using (Stream stream = file.OpenReadStream())
        {
            await this.service.UploadFileAsync(fileName, stream);
        }
        ViewData["MENSAJE"] = "Fichero subido correctamente";
        return View();
    }
}
```

```
}
```

#### INDEX.CSHTML

```
@model List<string>

<h1 style="color:blue">
    Azure storage Files
</h1>
<p>
    <a asp-controller="AzureFiles"
        asp-action="UploadFile">
        Upload Azure File
    </a>
</p>

@if (Model != null){
<ul class="list-group">
    @foreach (string file in Model){
        <li class="list-group-item">
            @file
            <a asp-controller="AzureFiles"
                asp-action="ReadFile"
                asp-route-filename="@file"
                class="btn btn-info">
                Read File
            </a>
            <a asp-controller="AzureFiles"
                asp-action="DeleteFile"
                asp-route-filename="@file"
                class="btn btn-danger">
                Delete file
            </a>
        </li>
    }
</ul>
}
```

Sobre Program habilitamos los servicios

#### PROGRAM

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddTransient<ServiceStorageFiles>();
builder.Services.AddControllersWithViews();
```

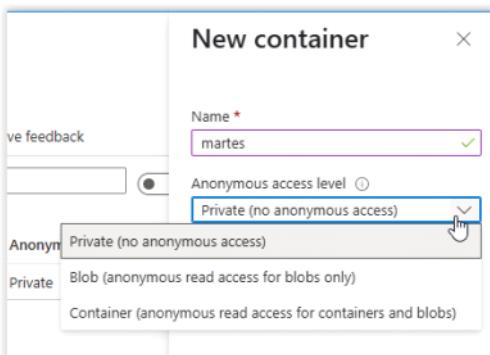
#### AZURE STORAGE BLOBS

Es un tipo de almacenamiento de ficheros creado para subir elementos que NO sean Textuales aunque, por supuesto, podemos subir ficheros de texto.

Algo muy importante y lo que diferencia respecto a Azure Files es que tiene seguridad dentro De sus elementos raíces, es decir, Containers.

Funciona mediante containers y dentro tendremos nuestros ficheros.

Si un container es Public, podremos acceder a sus ficheros mediante URL  
Si tenemos un container private, solamente podremos acceder mediante Keys



Necesitamos habilitar el acceso anónimo a los ficheros si lo necesitamos

**Nota:** Todos los nombres de los recursos de **Azure Storage** deben estar en minúscula.

Vamos a realizar las siguientes acciones:

- Listado de contenedores
- Crear los contenedores
- Eliminar contenedores
- Listado de ficheros de cada container
- Subir ficheros al container
- Eliminar ficheros del container

**Azure.Storage.Blobs** by azure-sdk, Microsoft, 376M downloads  
This client library enables working with the Microsoft Azure Storage Blob service for storing binary and text data.

Sobre **Models** creamos una nueva clase llamada **BlobModel**

## BLOBMODEL

```
public class BlobModel
{
    0 references | 0 changes | 0 authors, 0 changes
    public string Nombre { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Url { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Container { get; set; }
}
```

Para acceder al servicio de Blobs, necesitamos las Keys y una clase llamada **BlobServiceClient**

Vamos a inyectar dicho client desde **Program**

```
var builder = WebApplication.CreateBuilder(args);

string azureKeys =
    builder.Configuration.GetValue<string>
        ("AzureKeys:StorageAccount");
BlobServiceClient blobServiceClient =
    new BlobServiceClient(azureKeys);
builder.Services.AddTransient<BlobServiceClient>
    (x => blobServiceClient);
```

Creamos un nuevo servicio llamado **ServiceStorageBlobs**

## SERVICESTORAGEBLOBS

```
public class ServiceStorageBlobs
{
    private BlobServiceClient client;

    public ServiceStorageBlobs(BlobServiceClient client)
    {
        this.client = client;
    }

    //METODO PARA RECUPERAR TODOS LOS CONTAINERS
    public async Task<List<string>>
        GetContainersAsync()
    {
        List<string> containers = new List<string>();
        await foreach (BlobContainerItem item in
            this.client.GetBlobContainersAsync())
        {
            containers.Add(item.Name);
        }
        return containers;
    }

    //METODO PARA CREAR UN CONTAINER
    public async Task CreateContainerAsync(string containerName)
    {
        await this.client.CreateBlobContainerAsync(
            containerName, PublicAccessType.Blob);
    }

    public async Task DeleteContainerAsync(string containerName)
    {
        await this.client.DeleteBlobContainerAsync(containerName);
    }

    //METODO PARA RECUPERAR TODOS LOS BLOBS DE UN CONTAINER
    public async Task<List<BlobModel>>
        GetBlobsAsync(string containerName)
    {
        //NECESITAMOS UN CLIENTE DE CONTAINER
        BlobContainerClient containerClient =
            this.client.GetBlobContainerClient(containerName);
        List<BlobModel> models = new List<BlobModel>();
        await foreach (BlobItem item in
            containerClient.GetBlobsAsync())
        {
            BlobClient blobClient =
                containerClient.GetBlobClient(item.Name);
            BlobModel blob = new BlobModel();
            blob.Nombre = item.Name;
            blob.Url = blobClient.Uri.AbsoluteUri;
            blob.Container = containerName;
            models.Add(blob);
        }
        return models;
    }

    //METODO PARA ELIMINAR UN BLOB
    public async Task DeleteBlobAsync(string containerName
        , string blobName)
    {
        BlobContainerClient containerClient =
            this.client.GetBlobContainerClient(containerName);
        await containerClient.DeleteBlobAsync(blobName);
    }
}
```

```

//METODO PARA SUBIR UN BLOB A UN CONTAINER
public async Task UploadBlobAsync
    (string containerName, string blobName, Stream stream)
{
    BlobContainerClient containerClient =
        this.client.GetBlobContainerClient(containerName);
    await containerClient.UploadBlobAsync
        (blobName, stream);
}
}

```

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

#### AZUREBLOBSCONTROLLER

```

public class AzureBlobsController : Controller
{
    private ServiceStorageBlobs service;

    public AzureBlobsController(ServiceStorageBlobs service)
    {
        this.service = service;
    }
    public async Task<IActionResult> Index()
    {
        List<string> containers =
            await this.service.GetContainersAsync();
        return View(containers);
    }

    public IActionResult CreateContainer()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> CreateContainer
        (string containername)
    {
        await this.service.CreateContainerAsync(containername);
        return RedirectToAction("Index");
    }

    public async Task<IActionResult> DeleteContainer
        (string containername)
    {
        await this.service.DeleteContainerAsync(containername);
        return RedirectToAction("Index");
    }

    public async Task<IActionResult>
        ListBlobs(string containername)
    {
        List<BlobModel> blobs =
            await this.service.GetBlobsAsync(containername);
        return View(blobs);
    }

    public IActionResult UploadBlob(string containername)
    {
        ViewData["CONTAINER"] = containername;
        return View();
    }

    [HttpPost]
    public async Task<IActionResult>
        UploadBlob(string containername, IFormFile file)
    {
        string blobName = file.FileName;
        using (Stream stream = file.OpenReadStream())
        {
            await this.service.UploadBlobAsync
                (containername, blobName, stream);
        }
        return RedirectToAction("ListBlobs",
            new { containername = containername });
    }

    public async Task<IActionResult>
        DeleteBlob(string containername, string blobname)
    {
        await this.service.DeleteBlobAsync(containername, blobname);
        return RedirectToAction("ListBlobs",
            new { containername = containername });
    }
}

```

#### PROGRAM

---

```

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddTransient<ServiceStorageBlobs>();

```

#### FUNCIONAL??

Yo quiero ver también las fotos en privado si soy el jefe...

List Blobs			
Nombre	Url	Container	
Delete Blob: admin.jpeg	<a href="https://storagefajamarpos.blob.core.windows.net/martesprivate/admin.jpeg">https://storagefajamarpos.blob.core.windows.net/martesprivate/admin.jpeg</a>	martesprivate	
Delete Blob: air-jordan-6-retro-alternate-91.jpg	<a href="https://storagefajamarpos.blob.core.windows.net/martesprivate/air-jordan-6-retro-alternate-91.jpg">https://storagefajamarpos.blob.core.windows.net/martesprivate/air-jordan-6-retro-alternate-91.jpg</a>	martesprivate	
Delete Blob: fry.png	<a href="https://storagefajamarpos.blob.core.windows.net/martesprivate/fry.png">https://storagefajamarpos.blob.core.windows.net/martesprivate/fry.png</a>	martesprivate	

## AZURE STORAGE TABLES

Las tablas de Azure Storage contienen información de tipo NoSQL, lo que quiere decir que almacenan información no relacional en formato JSON.

En realidad, todas las bases de datos NoSQL almacenan información JSON

Si hablamos de información no relacional estamos hablando de elementos que son comunes entre sí, pero que podrían tener diferentes características en su información.

Por ejemplo, el tráfico, el clima.

No es lo mismo el clima de Madrid que el clima de un pueblo de Murcia o de Galicia

Pongamos un ejemplo de Familias:

- Familia Perez: Un matrimonio simple
- Familia Ruiz: Un matrimonio simple con un nene
- Familia Alcantara: Un matrimonio, mil hijos, la abuela, nietos, perro.

```
{  
    "nombrefamilia": "Perez": {  
        "Marido": "Luis", "Esposa": "Maria"  
    },  
    "nombrefamilia": "Alcantara": {  
        "Marido": "Andres", "Esposa": "Luisa",  
        "Hijos": [  
            "Pepe", "Laura", "Maite"  
        ],  
        "Mascotas": [  
            "Perro", "Gato"  
        ]  
    }  
}
```

Los contenidos a almacenar dentro de cualquier Storage de tables son los siguientes:

- **Tabla:** Debemos almacenar objetos del mismo tipo
- **Entidad:** Una entidad es un registro de la tabla que irá mapeado mediante Atributos Json.

Cada entidad, de forma obligatoria, tendrá que heredar de **ITableEntity**

A partir de dicha clase, se generan una serie de propiedades:

- **Partition Key:** Es una clave que almacena un "conjunto" de los datos. Por ejemplo, si estamos almacenando información de Vuelos, nuestro Partition Key podría ser el nombre de la compañía
- **Row Key:** Es la clave única de cada registro/entidad, por ejemplo, el código del vuelo

Vamos a trabajar con un ejemplo de Clientes.

Realizaremos las siguientes acciones sobre los registros/entidades

- Insertar registros
- Eliminar registros
- Mostrar registros
- Buscar registros por Row Key
- Buscar registros por Partition Key

Comenzaremos trabajando con un recurso ya creado, es decir, creamos una tabla llamada **clientes**  
Dentro de **Azure Storage**

The screenshot shows the Azure Storage Tables blade. On the left, there's a sidebar with links like 'Storage mover', 'Partner solutions', 'Resource visualizer', 'Data storage', 'Containers', and 'File shares'. The main area has a search bar and navigation buttons ('Table', 'Refresh', 'Delete', 'Give feedback'). Below that, it says 'Authentication method: Access key (Switch to Microsoft Entra user account)'. There's a 'Search tables by prefix' input field. A table lists a single row: 'Table' column has 'clientes', and 'Url' column has 'https://storagetajamarpgs.table.core.windows.net/clientes'. A success message 'Successfully created storage table 'clientes'' is displayed in a toast at the top right.

**Azure.Data.Tables** by [azure-sdk](#), [Microsoft](#), 63,9M downloads  
This client library enables working with the Microsoft Azure Table service

Sobre **Models** creamos una nueva clase llamada **Cliente**

Un Cliente tendrá un Nombre, Edad, Salario, IdCliente, Empresa

IDCLIENTE: **ROW KEY**

EMPRESA: **PARTITION KEY**

**CLIENTE**

```
public class Cliente: ITableEntity  
{  
    public string Nombre { get; set; }  
    public int Edad { get; set; }  
    public int Salario { get; set; }
```

```

private string _Empresa;
public string Empresa
{
    get { return this._Empresa; }
    set {
        this._Empresa = value;
        this.PartitionKey = value;
    }
}

private int _IdCliente;
public int IdCliente
{
    get { return this._IdCliente; }
    set {
        this._IdCliente = value;
        this.RowKey = value.ToString();
    }
}

public string PartitionKey { get; set; }
public string RowKey { get; set; }
public DateTimeOffset? Timestamp { get; set; }
public ETag ETag { get; set; }
}

```

Sobre Services, agregamos una nueva clase llamada ServiceStorageTables

#### SERVICESTORAGETABLES

```

public class ServiceStorageTables
{
    private TableClient tableClient;

    public ServiceStorageTables(TableServiceClient tableService)
    {
        this.tableClient =
            tableService.GetTableClient("clientes");
    }

    public async Task CreateClientAsync(int id, string nombre
        , string empresa, int salario, int edad)
    {
        Cliente cliente = new Cliente
        {
            IdCliente = id,
            Empresa = empresa,
            Nombre = nombre,
            Salario = salario,
            Edad = edad
        };
        await this.tableClient.AddEntityAsync<Cliente>(cliente);
    }

    //INTERNAMENTE, SE PUEDEN BUSCAR CLIENTES POR CUALQUIER CAMPO
    //SI VAMOS A REALIZAR UNA BUSQUEDA, POR EJEMPLO PARA DETAILS
    //NO SE PUEDE BUSCAR SOLAMENTE POR SU ROW KEY, SE GENERA
    //UNA COMBINACION DE ROW KEY Y PARTITION KEY PARA BUSCAR
    //POR ENTIDAD UNICA
    public async Task<Cliente> FindClientAsync
        (string partitionKey, string rowKey)
    {
        Cliente cliente = await
            this.tableClient.GetEntityAsync<Cliente>
            (partitionKey, rowKey);
        return cliente;
    }

    //PARA ELIMINAR UN ELEMENTO UNICO TAMBIEN NECESITAMOS
    //PK Y RK
    public async Task DeleteClientAsync
        (string partitionKey, string rowKey)
    {
        await this.tableClient.DeleteEntityAsync
            (partitionKey, rowKey);
    }

    public async Task<List<Cliente>> GetClientesAsync()
    {
        List<Cliente> clientes = new List<Cliente>();
        //PARA BUSCAR, NECESITAMOS UTILIZAR UN OBJETO QUERY
        //CON UN FILTER
        var query =
            this.tableClient.QueryAsync<Cliente>
            (filter: "");
        //DEBEMOS EXTRAER TODOS LOS DATOS DEL QUERY
        await foreach (var item in query)
        {
            clientes.Add(item);
        }
        return clientes;
    }

    public async Task<List<Cliente>> GetClientesEmpresaAsync
        (string empresa)
    {
        //TENEMOS DOS TIPOS DE FILTER, LOS DOS SE UTILIZAN
        //CON query
        //1) SI REALIZAMOS EL FILTER CON QueryAsync,
        //DEBEMOS UTILIZAR UNA
        //SINTAXIS Y EXTRAER LOS DATOS MANUALES
        //string filtro = "Campo eq valor";
        //string filtro = "Campo eq valor and Campo2 gt valor2";
        //string filtro = "Campo lt valor and Campo2 gt valor2";
        //string filtroSalario =
        //    "Salario gt 250000 and Salario lt 300000";
        //var query =
        //    this.tableClient.QueryAsync<Cliente>
        //    (filter: filtroSalario);

        //2) SI REALIZAMOS LA CONSULTA CON Query
        //PODEMOS UTILIZAR LAMBDA Y EXTRAER LOS DATOS
        //DIRECTAMENTE, PERO NO ES ASINCRONO
        var query =
            this.tableClient.Query<Cliente>
            (x => x.Empresa == empresa);
        return query.ToList();
    }
}

```

Realizamos la inyección de las clases sobre **Program**

#### PROGRAM

```
string azureKeys =
    builder.Configuration.GetValue<string>
    ("AzureKeys:StorageAccount");
TableServiceClient tableServiceClient =
    new TableServiceClient(azureKeys);
builder.Services.AddTransient<TableServiceClient>
    (x => tableServiceClient);
builder.Services.AddTransient<ServiceStorageTables>();
```

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

#### AZURETABLESCONTROLLER

```
public class AzureTablesController : Controller
{
    private ServiceStorageTables service;

    public AzureTablesController(ServiceStorageTables service)
    {
        this.service = service;
    }

    public async Task<IActionResult> Index()
    {
        List<Cliente> clientes =
            await this.service.GetClientesAsync();
        return View(clientes);
    }

    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Create(Cliente cliente)
    {
        await this.service.CreateClientAsync(cliente.IdCliente
            , cliente.Nombre, cliente.Empresa
            , cliente.Salario, cliente.Edad);
        return RedirectToAction("Index");
    }

    public async Task<IActionResult> Delete
        (string partitionkey, string rowkey)
    {
        await this.service.DeleteClientAsync
            (partitionkey, rowkey);
        return RedirectToAction("Index");
    }

    public async Task<IActionResult> Details
        (string partitionkey, string rowkey)
    {
        Cliente cliente = await
            this.service.FindClientAsync
            (partitionkey, rowkey);
        return View(cliente);
    }

    public IActionResult ClientesEmpresa()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> ClientesEmpresa(string empresa)
    {
        List<Cliente> clientes =
            await this.service.GetClientesEmpresaAsync(empresa);
        return View(clientes);
    }
}
```

Nos instalamos Azure Storage Explorer

<https://azure.microsoft.com/en-us/products/storage/storage-explorer>

Es un emulador que nos permite poder trabajar en Local si lo necesitamos para no Tener coste.

Documentación Azurite

<https://learn.microsoft.com/en-us/azure/storage/common/storage-use-azurite?toc=%2Fazure%2Fstorage%2Fblobs%2Ftoc.json&bc=%2Fazure%2Fstorage%2Fblobs%2Fbreadcrumb%2Ftoc.json&tabs=visual-studio%2Cblob-storage#install-azurite>

C:\Program Files\Microsoft Visual Studio\2022\Enterprise\Common7\IDE\Extensions\Microsoft\Azure Storage Emulator

#### SAS SHARED ACCESS SIGNATURE

Cuando estamos trabajando con Azure Storage, hemos visto que tenemos unas claves De acceso a los recursos de Storage.

Imaginemos que tenéis una cuenta compartida única de Tajamar entre todos los compañeros.

¿Cómo se puede controlar que uno de nosotros, no realice acciones por error en el trabajo De los otros con Storage?

Actualmente tenemos acceso a todos los recursos de Azure Storage. Si tenemos las claves Y estamos trabajando con Blobs, si nos apetece, podemos mirar Files o podemos entrar en Tables.

Es una funcionalidad que nos permite limitar el acceso a los recursos de Azure. Dicho límite depende de los recursos, por ejemplo, si son tablas, podemos limitar los Registros que visualizan los usuarios.

Si hablamos de Blobs, podemos limitar el acceso a un solo Container o incluir tiempo de acceso Si lo necesitamos.

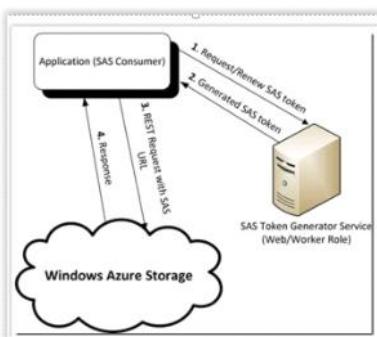
Dichas Keys las podemos generar desde el portal de Azure.

The screenshot shows the Azure portal interface for generating a Shared Access Signature (SAS). The top navigation bar includes 'Microsoft Azure', a search bar, and a 'Copilot' button. The main title is 'storagetajamarpgs | Shared access signature'. The left sidebar shows 'Data storage' and 'Security + networking' sections, with 'Shared access signature' selected. The main content area describes SAS as a URI that grants restricted access rights to Azure Storage resources. It allows specifying allowed services (Blob, File, Queue, Table), allowed resource types (Service, Container, Object), and allowed permissions (Read, Write, Delete, List, Add, Create, Update, Process, Immutable). A 'Learn more about creating an account SAS' link is also present.

Mediante el portal, podríamos generar un Token y utilizarlo para el acceso.

Nosotros vamos a darle una vuelta.

- 1) **Primer paso:**  
Vamos a realizar una aplicación en la que tendremos Alumnos en Storage Tables. Dichos alumnos tendrán un curso asignado que será su Partition Key. Esta aplicación se encargará de subir los alumnos en masa a Azure Storage.
- 2) **Segunda acción:**  
Crearemos un Minimal Api que se encargará de generar Tokens de acceso al servicio De Storage Tables, es decir, nos generará una clave SAS para acceder al Servicio.
- 3) **Tercer Paso:** Tener una aplicación de prueba para comprobar la funcionalidad del Token
- 4) **Cuarto paso:** Incluir Basic Authentication dentro de Minimal Api.



Comenzamos creando una aplicación dónde consumiremos un documento XML con los Alumnos Para subir a Azure Storage

Creamos una app de tipo Mvc que consumirá los datos y también subirá los datos a Azure Storage

Creamos una nueva App llamada **MvcCoreSasAzureStorage**

Agregamos Azure.Data.Tables y creamos una clase Alumno con la siguiente estructura.



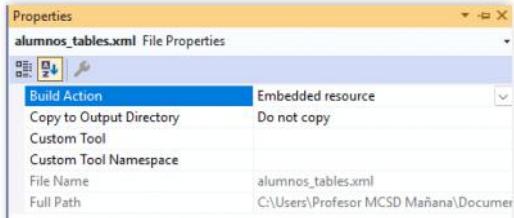
IdAlumno: RowKey  
Curso: Partition Key

```

<alumno>
    <idalumno>1</idalumno>
    <curso>NET</curso>
    <nombre>Nacho</nombre>
    <apellidos>Shum Llanos</apellidos>
    <nota>8</nota>
</alumno>

```

Descargamos de Teams un fichero llamado **alumnos\_tables.xml** y lo ponemos dentro del Proyecto en una carpeta llamada **Documents** (no dentro de root)



Creamos una carpeta llamada **Helpers** y una clase llamada **HelperXML**

Ahí recuperaremos el fichero XML que hemos incrustado en nuestro Assembly y Lo mapeamos para devolver todos los alumnos en una colección.

#### HELPERXML

```

public class HelperXML
{
    private XDocument document;

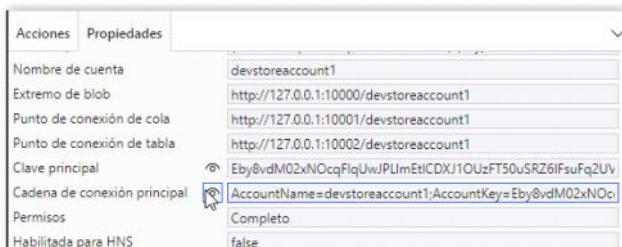
    public HelperXML()
    {
        string assemblyPath =
            "MvcCoreSasAzureStorage.Documents.alumnos_tables.xml";
        Stream stream =
            this.GetType().Assembly.GetManifestResourceStream(
                assemblyPath);
        this.document = XDocument.Load(stream);
    }

    public List<Alumno> GetAlumnos()
    {
        var consulta = from datos in
            this.document.Descendants("alumno")
        select new Alumno
        {
            IdAlumno =
                int.Parse(datos.Element("idalumno").Value),
            Nombre =
                datos.Element("nombre").Value,
            Apellidos =
                datos.Element("apellidos").Value,
            Curso =
                datos.Element("curso").Value,
            Nota =
                int.Parse(datos.Element("nota").Value)
        };
        return consulta.ToList();
    }
}

```

El siguiente paso que vamos a realizar será un Controller que almacenará los alumnos dentro Azure directamente con las Keys para probar el emulador.

Copiamos las claves del Emulador Azurite



Creamos un nuevo controlador llamado **MigracionController**

```

MIGRATIONCONTROLLER

public class MigracionController : Controller
{
    private HelperXML helper;

    public MigracionController(HelperXML helper)
    {
        this.helper = helper;
    }

    public IActionResult Index()
    {
        return View();
    }

    [HttpPost]

```

```

public async Task<IActionResult> Index(string accion)
{
    string azureKeys =
        "AccountName=devstoreaccount1;AccountKey=Eby8vdM02xNOcqFlqUwJPllmE
t1CDXJ10UzFT50uSRZ6IFsuq2UVErCz4I6tq/K1S2FPT0tr/KBHBeksoGMGw==;DefaultEndpo
intProtocol=http;BlobEndpoint=http://127.0.0.1:10000/devstoreaccount1;QueueEndp
oint=http://127.0.0.1:10001/devstoreaccount1;TableEndpoint=http://127.0.0.1:10
002/devstoreaccount1";
    TableServiceClient serviceClient =
        new TableServiceClient(azureKeys);
    TableClient tableClient =
        serviceClient.GetTableClient("alumnos");
    await tableClient.CreateIfNotExistsAsync();
    List<Alumno> alumnos = this.helper.GetAlumnos();
    foreach (Alumno alum in alumnos)
    {
        await tableClient.AddEntityAsync<Alumno>(alum);
    }
    ViewData["MENSAJE"] = "Migración completada";
    return View();
}

```

INDEX.CSHTML

```

<h1>Migración Alumnos To Azure</h1>

<form method="post">
    <button class="btn btn-danger" name="accion" value="accion">
        Migrar Alumnos
    </button>
</form>
<h2 style="color:blue">
    @ViewData["MENSAJE"]
</h2>

```

PROGRAM

```

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddTransient<HelperXML>();
builder.Services.AddControllersWithViews();

var app = builder.Build();

```

Y comprobamos que es funcional

The screenshot shows a web page with a header containing a logo of Spider-Man's mask, a 'Home' link, and 'Migración datos Azure' and 'Privacy' links. The main content area has a title 'Migración Alumnos To Azure'. Below the title is a red button labeled 'Migrar Alumnos'. Underneath the button, the text 'Migración completada' is displayed in blue.

IdAlumno	Curso	Nombre	Apellidos
182	EN PROCESO	Doryan	Mamani
183	EN PROCESO	Sofia	Martinez
184	EN PROCESO	Maki	Miron
185	EN PROCESO	Carolina	Penalba
186	EN PROCESO	Manuel	Perez
187	EN PROCESO	Andrei	Popa
188	EN PROCESO	Valentin	Preutesei
189	EN PROCESO	Alejandro	Robles
190	EN PROCESO	Daniel	Rodriguez
191	EN PROCESO	Jorge	Ruiz
192	EN PROCESO	Tomas	Santamarria
193	EN PROCESO	Sofia	Villarejo

El siguiente paso es generar una aplicación que nos ofrezca los Tokens de acceso  
A nuestro servicio de Azure.

Los desarrolladores NUNCA tendrán las Keys, tendrán unos accesos por seguridad.

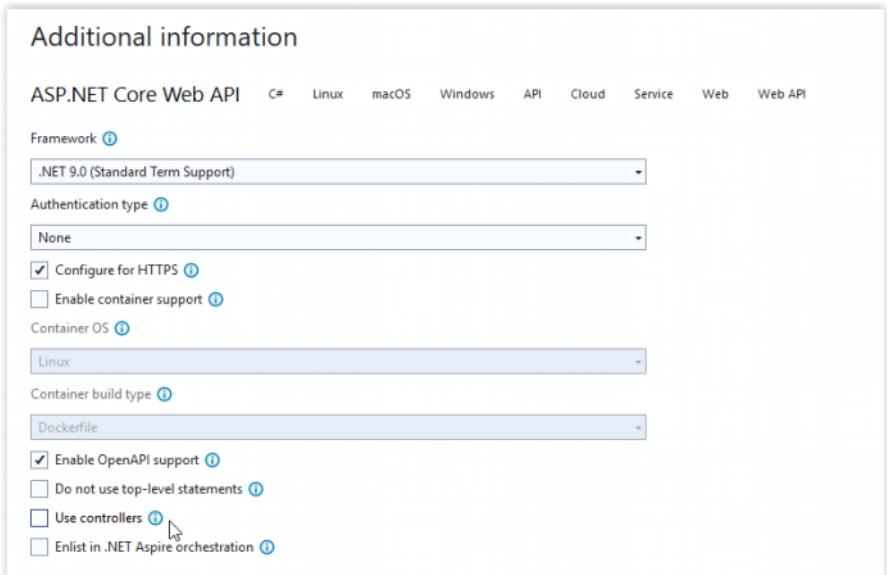
Vamos a utilizar un Api para generar dichos accesos.

Lo que haremos será un Minimal Api con un método que recibirá el CURSO que deseamos Que se permita el acceso.  
El Api nos devolverá un String con la cadena de conexión (SAS) de acceso solamente a dicho Recurso.

La única diferencia que tenemos con un Api convencional está en que NO tenemos **Controllers**, todo se realiza dentro de Program.

En la creación del Api, debemos desmarcar la opción de **Use Controllers**

Creamos un nuevo proyecto Api llamado **ApiAzureStorageTokens**



Este proyecto es el que tendrá las Keys de acceso a todo Azure Storage.

#### APPSETTINGS.JSON

```
ore.org/appsettings.json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "AzureKeys": {
    "StorageAccount": "AccountName=devstoreaccount1;AccountKey=Lip1Y...PQ=="
  }
}
```

Necesitamos **Azure Storage Tables**

 **Azure.Data.Tables** by azure-sdk, Microsoft, 64,1M downloads  
This client library enables working with the Microsoft Azure Table service

Creamos una carpeta llamada **Services** y una clase llamada **ServiceSaSToken**

**SERVICESASTOKEN**

```
public class ServiceSaSToken
{
  private TableClient tableAlumnos;

  public ServiceSaSToken(IConfiguration configuration)
  {
    string azureKeys =
      configuration.GetValue<string>(
        "AzureKeys:StorageAccount");
    TableServiceClient serviceClient =
      new TableServiceClient(azureKeys);
    this.tableAlumnos = serviceClient.GetTableClient("alumnos");
  }

  public string GenerateToken(string curso)
  {
    //NECESITAMOS LOS PERMISOS DE ACCESO, POR AHORA
    //SOLAMENTE PERMITIREMOS LECTURA
    TableSasPermissions permisos =
      new TableSasPermissions()
        .WithRead();
    TableSasToken token = new TableSasTokenBuilder()
      .WithPermissions(permisos)
      .WithTableName("alumnos")
      .WithValidDuration(TimeSpan.FromHours(1))
      .Build();
    return token.SasToken;
  }
}
```

```

    TableSasPermissions.Read;
    //EL ACCESO AL TOKEN ESTA DELIMITADO POR UN TIEMPO
    //NECESITAMOS INCLUIR EL TIEMPO QUE TENDRA ACCESO
    //A LEER LOS ELEMENTOS
    TableSasBuilder builder =
        this.tableAlumnos.GetSasBuilder(
            permisos, DateTime.UtcNow.AddMinutes(30));
    //COMO ROW KEY Y PARTITION KEY SON STRING
    //PODEMOS LIMITAR EL ACCESO DE FORMA ALFABETICA
    //A LOS DATOS, YA SEA POR ROW KEY, PARTITION KEY
    //O LOS DOS
    builder.PartitionKeyStart = curso;
    builder.PartitionKeyEnd = curso;
    //CON ESTO YA PODEMOS GENERAR EL TOKEN QUE
    //ES UN ACCESO POR URI
    Uri uriToken =
        this.tableAlumnos.GenerateSasUri(builder);
    string token = uriToken.AbsoluteUri;
    return token;
}
}

```

El siguiente paso es crear un nuevo EndPoint dentro de Program para generar los Token De SaS

#### PROGRAM

```

using ApiAzureStorageTokens.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddTransient<ServiceSaSToken>();
// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi
builder.Services.AddOpenApi();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
}

app.MapOpenApi();
app.UseSwaggerUI(options => {
    options.SwaggerEndpoint("/openapi/v1.json", "Azure Minimal API");
    options.RoutePrefix = "";
});
app.UseHttpsRedirection();

//app.MapGet("/testing", () =>
//{
//    return "sin nada";
//});

//app.MapGet("/parametros/{dato}", (string dato
//    , ObjetoInyectado service) =>
//{
//    return "con algo: " + dato;
//});

//NECESITAMOS UN METODO GET PARA ACCEDER AL TOKEN MEDIANTE EL CURSO
//EL METODO, AL DECLARARLO, LLEVA EL Routing DE ACCESO Y TAMBIEN
//PUEDE LLEVAR PARAMETROS
//NECESITAMOS RECUPERAR EL SERVICE DENTRO DE NUESTRO METODO
//DEL MINIMAL API
app.MapGet("/token/{curso}", (string curso
    , ServiceSaSToken service) =>
{
    string token = service.GenerateToken(curso);
    return new { token = token };
});
app.Run();

```

Y podremos visualizar cómo genera el Token de acceso

Server response	
Code	Details
200	<b>Response body</b> <pre>http://127.0.0.1:10002/devstoreaccount1/alumnos?tn=alumnos&amp;spk=EN%20PROCESO&amp;epk=EN%20PROCESO&amp;sv=2020-12-06&amp;se=2025-04-02T09%3A56%3A55Z&amp;sp=r&amp;sig=13jbwRwpulIL70i2nb89CxmNlR6a3hz47JmZsg04YI%3D</pre>
	<a href="#">Copy</a> <a href="#">Download</a>
	<b>Response headers</b>

El siguiente paso es utilizar el proyecto anterior de migración para acceder a los Datos de los alumnos mediante el Token.

En realidad, en el proyecto de acceso Mvc nunca estarían las claves, las hemos incluido Para jugar con la migración.

Abrimos una nueva ventana de Visual Studio con el proyecto MVC

Agregamos Newton Soft

**Newtonsoft.Json** by dotnetfoundation, jamesnk, newtonsoft, 6,02B downloads  
Json.NET is a popular high-performance JSON framework for .NET

13.0.3

Vamos a realizar un Service que devolverá los alumnos de un Curso utilizando el Token generado En nuestro Api.

Necesitamos dos llamadas:

- Generación de Token
- Recuperación de alumnos con dicho Token

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

#### SERVICEAZUREALUMNOS

```
public class ServiceAzureAlumnos
{
    private TableClient tableAlumnos;

    //NECESITAMOS LA URL DE ACCESO AL TOKEN
    private string UrlApi;

    public ServiceAzureAlumnos(IConfiguration configuration)
    {
        this.UrlApi = configuration.GetValue<string>
            ("ApiUrls:ApiAzureToken");
    }

    private async Task<string> GetTokenAsync(string curso)
    {
        using (HttpClient client = new HttpClient())
        {
            string request = "token/" + curso;
            client.BaseAddress = new Uri(this.UrlApi);
            client.DefaultRequestHeaders.Clear();
            client.DefaultRequestHeaders.Accept.Add(
                new MediaTypeWithQualityHeaderValue("application/json"));
            HttpResponseMessage response =
                await client.GetAsync(request);
            if (response.IsSuccessStatusCode)
            {
                string data = await
                    response.Content.ReadAsStringAsync();
                JObject keys = JObject.Parse(data);
                string token = keys.GetValue("token").ToString();
                return token;
            }
            else
            {
                return null;
            }
        }
    }

    public async Task<List<Alumno>>
        GetAlumnosAsync(string curso)
    {
        string token = await this.GetTokenAsync(curso);
        //LO QUE NOS DEVUELVE EL TOKEN ES UNA URL
        Uri uriToken = new Uri(token);
        //PARA ACCEDER AL RECURSO, SIMPLEMENTE CREAMOS
        //EL RECURSO MEDIANTE UN URI
        this.tableAlumnos = new TableClient(uriToken);
        List<Alumno> alumnos = new List<Alumno>();
        var query = this.tableAlumnos.QueryAsync<Alumno>
            (filter: "");
        await foreach(var item in query)
        {
            alumnos.Add(item);
        }
        return alumnos;
    }
}
```

Incluimos dentro de **appsettings.json** la Url del nuestro servicio Api

#### APPSETTINGS.JSON

```
},
"AllowedHosts": "*",
"ApiUrls": {
    "ApiAzureToken": "https://localhost:7187/"
}
```

Resolvemos las dependencias dentro de **Program**

#### PROGRAM

```
// Add services to the container.
builder.Services.AddTransient<ServiceAzureAlumnos>();
builder.Services.AddTransient<HelperXML>();
builder.Services.AddControllersWithViews();
```

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

#### ALUMNOSCONTROLLER

```
public class AlumnosController : Controller
{
    private ServiceAzureAlumnos service;

    public AlumnosController(ServiceAzureAlumnos service)
    {
        this.service = service;
    }

    public IActionResult Index()
```

```

    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Index(string curso)
    {
        List<Alumno> alumnos = await
            this.service.GetAlumnosAsync(curso);
        return View(alumnos);
    }
}

```

En Azurite no funciona SaS correctamente

IdAlumno	Curso	Nombre	Apellidos	Nota
172	EN PROCESO	Victor	Castrillo	10
173	EN PROCESO	Amanda	Crespo	10
174	EN PROCESO	Monica	Delgado	10
175	EN PROCESO	Aaron	Garcia	10

A continuación, vamos a comprobar si podemos crear alumnos dentro de Azure Storage

Creamos un nuevo método sobre el service **ServiceAzureAlumnos**

#### SERVICEAZUREALUMNOS

```

public async Task CreateAlumno
(
    int idAlumno
    , string nombre, string apellidos
    , int nota)
{
    string curso = "EN PROCESO";
    string token = await this.GetTokenAsync(curso);
    Alumno alumno = new Alumno();
    alumno.IdAlumno = idAlumno;
    alumno.Curso = curso;
    alumno.Nombre = nombre;
    alumno.Apellidos = apellidos;
    alumno.Nota = nota;
    Uri uriToken = new Uri(token);
    this.tableAlumnos = new TableClient(uriToken);
    await this.tableAlumnos.AddEntityAsync<Alumno>
        (alumno);
}

```

Implementamos la acción dentro de **AlumnosController**

#### ALUMNOSCONTROLLER

```

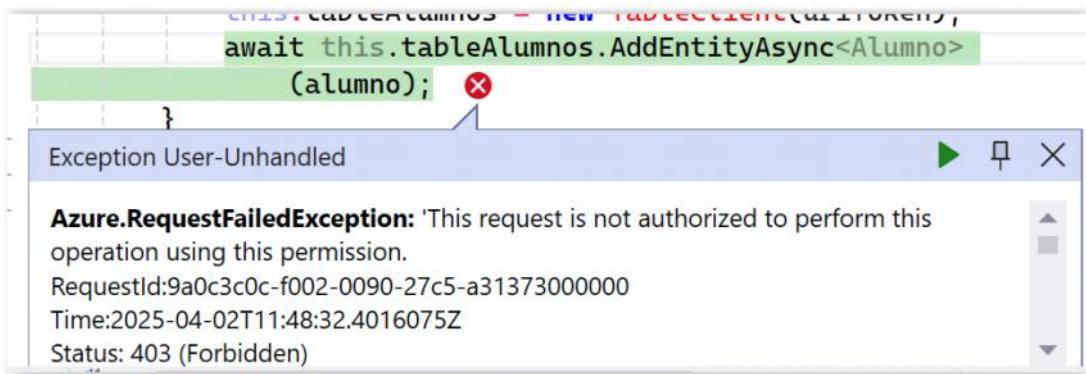
public IActionResult Create()
{
    return View();
}

[HttpPost]

public async Task<IActionResult> Create(Alumno alumno)
{
    await this.service.CreateAlumno
        (alumno.IdAlumno, alumno.Nombre
        , alumno.Apellidos, alumno.Nota);
    return RedirectToAction("Index");
}

```

Creamos la vista y ejecutamos



Debemos ofrecer más permisos dentro del Token en nuestro Api

```
public string GenerateToken(string curso)
{
    //NECESITAMOS LOS PERMISOS DE ACCESO, POR AHORA
    //SOLAMENTE PERMITIREMOS LECTURA
    TableSasPermissions permisos =
        TableSasPermissions.Read
        | TableSasPermissions.Add;
```

## AZURE CACHE REDIS

jueves, 3 de abril de 2025 9:09

The screenshot shows the Azure Cache for Redis dashboard. At the top, there's a search bar with the placeholder "Search all resources" and a dropdown menu for "Subscription". Below the search bar, there are buttons for "+ Create", "Manage view", "Refresh", and "Export to CSV". A sidebar on the left lists "Azure Managed Redis (Preview)", "Redis Enterprise", and "Azure Cache for Redis". The main area displays a table with columns for "Name", "Type", "Status", and "Actions". One row in the table is highlighted.

The screenshot shows the "New Redis Cache" creation wizard. It starts with a brief introduction about Azure Cache for Redis. The "Project details" section allows selecting a subscription ("Azure for Students") and a resource group ("rg-tajamar"). The "Instance Details" section includes fields for the cache name ("cacheredispaco"), region ("Spain Central"), cache SKU ("Basic (No SLA)"), and cache size ("C0 (250 MB Cache)").

The screenshot shows the "Networking" tab of the Redis configuration. It includes sections for "Network Connectivity" and "Advanced Networking". Under "Network Connectivity", it says "You can connect either publicly, via Public IP addresses or service endpoints, or privately, using a private endpoint." It shows the "Connectivity method" set to "Public Endpoint (Recommended)". A warning message at the bottom states: "⚠️ Enabling Public Endpoint exposes security vulnerabilities with uncontrolled public access. Unless public access is required, we recommend Private Endpoint. [Learn more](#)".

The screenshot shows the "Advanced" tab of the Redis configuration. It includes sections for "Redis version" (set to "Latest - 6"), "Non-TLS port" (with an unchecked "Enable" checkbox), "Authentication" (with checkboxes for "Microsoft Entra Authentication" and "Access Keys Authentication" both checked), and a warning message: "⚠️ Enable only Microsoft Entra ID to be secure by default. Enabling Access keys can lead to vulnerabilities from secrets leaked to servers, control systems, and connected to the public. [Learn more](#)".

Estamos creando un nuevo servicio de Cache Redis, como su nombre indica, es un servicio de Cache en la nube de Azure.  
Cuando nosotros creamos Cache (Favoritos), dicho cache es temporal a pesar de tener los datos en nuestro explorador.  
El Cache de favoritos es lo que seguiremos utilizando para favoritos, los elementos favoritos los hacemos mediante explorador o dispositivo, digamos que queremos que sea algo temporal.

Cuando tenemos Session, queremos almacenar información que suele ser más o menos importante para el usuario, por ejemplo, un carrito de la compra.

Un Cache Redis es un almacenamiento temporal en la nube. Si almacenamos Session tal y como lo hemos hecho hasta ahora, los datos no aparecen en múltiples dispositivos, es decir, el carrito no aparece en mi móvil si lo he hecho desde mi ordenador y viceversa.

Utiliza recursos mínimos, es decir, almacena JSON.

Funciona mediante una KEY única, es decir, **TODOS LOS USUARIOS ACCEDERAN A LA MISMA KEY**

Vamos a leer un documento XML como hicimos ayer, pero esta vez de Productos.

Tendremos productos y lo que haremos será mostrar los productos

Tendremos la posibilidad de almacenar productos favoritos y lo haremos mediante Cache Redis.

Creamos un nuevo proyecto llamado **MvcCoreCacheRedis**

Sobre **wwwroot** agregamos una nueva carpeta llamada **documents** y pegamos ahí **productos.xml**

Sobre **Models** creamos una nueva clase llamada **Producto**

#### PRODUCTO

```
public class Producto {  
    public int IdProducto {get;set;}  
    public string Nombre {get;set;}  
    public string Descripcion {get;set;}  
    public int Precio {get;set;}  
    public string Imagen {get;set;}  
}
```

Sobre el proyecto, creamos una carpeta llamada **Helpers** y una clase llamada **HelperPathProvider**

#### HELPERSPATHPROVIDER

```
public enum Folders { Images, Documents }  
public class HelperPathProvider {  
    private IWebHostEnvironment environment;  
  
    public HelperPathProvider( IWebHostEnvironment environment ) {  
        this.environment = environment;  
    }  
  
    public string MapPath(string fileName, Folders folder){  
        string carpeta = "";  
        if (folder == Folders.Images) {  
            carpeta = "images";  
        }else if (folder == Folders.Documents){  
            carpeta = "documents";  
        }  
        string path = Path.Combine(this.environment.WebRootPath, carpeta, fileName);  
        return path;  
    }  
}
```

Creamos una carpeta llamada **Repositories** y una clase llamada **RepositoryProductos**

#### REPOSITORYPRODUCTOS

```
public class RepositoryProductos {  
    private XDocument document;  
  
    public RepositoryProductos(HelperPathProvider helper){  
        string path = helper.MapPath("productos.xml", Folders.Documents);  
        this.document = XDocument.Load(path);  
    }  
  
    public List<Producto> GetProductos() {  
        var consulta = from datos in document.Descendants("producto")  
        select new Producto {  
            IdProducto = int.Parse(datos.Element("idproducto").Value),  
            Nombre = datos.Element("nombre").Value,  
            Descripcion = datos.Element("descripcion").Value,  
            Precio = int.Parse(datos.Element("precio").Value),  
            Imagen = datos.Element("imagen").Value  
        };  
        return consulta.ToList()  
    }  
  
    public Producto FindProducto(int idProducto) {  
        return this.GetProductos().FirstOrDefault(x => x.IdProducto == idProducto);  
    }  
}
```

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

#### PRODUCTOSCONTROLLER

```
public class ProductosController: Controller {  
    private RepositoryProductos repo;  
  
    public ProductosController(RepositoryProductos repo) {  
        this.repo = repo;  
    }  
  
    public IActionResult Index() {  
        List<Producto> productos = this.repo.GetProductos();  
        return View(productos);  
    }  
  
    public IActionResult Details(int id) {  
        Producto producto = this.repo.FindProducto(id);  
        return View(producto);  
    }  
}
```

Resolvemos las dependencias dentro de **Program**

#### PROGRAM

```
builder.Services.AddTransient<HelperPathProvider>();  
builder.Services.AddTransient<RepositoryProductos>();
```

Las vistas con Scaffolding

Una vez que todo es funcional, debemos copiar nuestras Keys de Redis dentro de appsettings.json

The screenshot shows the 'Access keys' tab in the Azure Cache for Redis settings. It displays two sets of connection strings: Primary and Secondary, both labeled as StackExchange.Redis. The connection strings are masked with asterisks. Below the connection strings are 'Show keys' and 'Save' buttons, and checkboxes for 'Disable Access Keys Authentication' and 'Regenerate Primary' or 'Regenerate Secondary'. A 'Copy to clipboard' button is also present.

#### APPSETTINGS.JSON

```
    },
    "AllowedHosts": "*",
    "AzureKeys": {
        "CacheRedis": "cacheredispaco.redis.cache.windows.net:6380,password=HJQyDkahEDAP0WZoTH5cJUFgbLqvvaAzCaQ7sG8
89,ssl=True,abortConnect=False",
    }
}
```

Agregamos los siguientes Nuget

The screenshot shows the NuGet package manager interface with three packages installed:

- StackExchange.Redis** by marcgravell, nickcraver, StackExchange, 604M downloads, version 2.8.1.
- Microsoft.Extensions.Caching.StackExchangeRedis** by aspnet, dotnetframework, Microsoft, 143M downloads, version 9.0.2.
- Newtonsoft.Json** by dotnetfoundation, jamesnk, newtonsoft, 6,038B downloads, version 13.0.3.

El siguiente paso es habilitar el servicio de Redis dentro de Program utilizando la cadena de conexión

#### PROGRAM

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
string cacheRedisKeys =
    builder.Configuration.GetValue<string>
    ("AzureKeys:CacheRedis");
builder.Services.AddStackExchangeRedisCache(options =>
{
    options.Configuration = cacheRedisKeys;
});
```

Necesitamos una clase que nos proporcione la conexión a la base de datos Cache Redis.  
Dicha clase debe ser static.

Si quiero recuperar algo de un objeto de inyección en static, no puedo directamente  
El truco está en crear un Helper con las claves que yo necesite.  
Dicho Helper tendrá propiedades static y podremos acceder a ellas directamente sin inyección.

Sobre **Helpers** creamos una nueva clase llamada **HelperCacheMultiplexer**

```
public static class HelperCacheMultiplexer
{
    private static Lazy<ConnectionMultiplexer> CreateConnection = new Lazy<ConnectionMultiplexer>(() =>
    {
        string cacheRedisKeys = "cacheredispaco.redis.cache.windows.net:6380
89,password=HJQyDkahEDAP0WZoTH5cJUFgbLqvvaAzCaQ7sG8
89,ssl=True,abortConnect=False";
        return ConnectionMultiplexer.Connect(cacheRedisKeys);
    });
    public static ConnectionMultiplexer Connection
    {
        get { return CreateConnection.Value; }
    }
}
```

Una vez que tenemos montadas las conexiones creamos un nuevo servicio.  
Sobre **Services** creamos una nueva clase llamada **ServiceCacheRedis**

```
SERVICECACHEREDIS

public class ServiceCacheRedis
{
    private IDatabase database;
    public ServiceCacheRedis()
    {
        this.database =
            HelperCacheMultiplexer.Connection.GetDatabase();
    }

    //METODO PARA ALMACENAR PRODUCTOS
    //LAS CLAVES DEBEN SER UNICAS POR USUARIO
    public async Task AddProductoFavoritoAsync(Producto producto)
    {
        //REDIS TRABAJA CON KEYS/VALUES PARA RECUPERAR LOS ELEMENTOS O
        //ALMACENAR LOS ELEMENTOS EN FORMATO JSON
        //ALMACENAREMOS EN REDIS UN CONJUNTO DE PRODUCTOS FAVORITOS
    }
}
```

```

        string jsonProductos = await
            this.database.StringGetAsync("favoritos");
        List<Producto> productosList;
        if (jsonProductos == null)
        {
            productosList = new List<Producto>();
        }
        else
        {
            //RECUPERAMOS LA COLECCION DE CACHE REDIS
            productosList = JsonConvert.DeserializeObject<List<Producto>>(jsonProductos);
        }

        //AGREGAMOS EL PRODUCTO NUEVO
        productosList.Add(producto);
        //EN EL PRODUCTO AGREGADO, VOLVEMOS A GENERAR EL JSON DE PRODUCTOS
        jsonProductos = JsonConvert.SerializeObject(productosList);
        //ALMACENAMOS LOS DATOS DE NUEVO EN CACHE REDIS
        await this.database.StringSetAsync("favoritos", jsonProductos);
    }

    //REALIZAMOS UN METODO PARA RECUPERAR LOS PRODUCTOS FAVORITOS
    public async Task<List<Producto>> GetProductosFavoritosAsync()
    {
        string jsonProductos = await this.database.StringGetAsync("favoritos");

        if (jsonProductos == null)
        {
            return null;
        }
        else
        {
            List<Producto> productos = JsonConvert.DeserializeObject<List<Producto>>(jsonProductos);
            return productos;
        }
    }

    public async Task DeleteProductoFavoritoAsync(int idProducto)
    {
        //ESTO NO ES UNA BASE DE DATOS, NO PODEMOS FILTRAR NI BUSCAR.
        //SOLAMENTE EXTRAER
        List<Producto> favoritos = await this.GetProductosFavoritosAsync();
        //SI EXISTEN FAVORITOS, ELIMINAMOS
        if (favoritos != null)
        {
            //BUSCAMOS EL PRODUCTO A ELIMINAR
            Producto productoDelete =
                favoritos.FirstOrDefault(x => x.IdProducto == idProducto);
            //ELIMINAMOS EL PRODUCTO DE LA COLECCION
            favoritos.Remove(productoDelete);
            //SI ELIMINAMOS TODOS LOS PRODUCTOS, DEBEMOS ELIMINAR CACHE REDIS
            if (favoritos.Count == 0)
            {
                await this.database.KeyDeleteAsync("favoritos");
            }
            else
            {
                //ALMACENAMOS LOS PRODUCTOS FAVORITOS DE NUEVO
                string jsonProductos = JsonConvert.SerializeObject(favoritos);
                //VOLVEMOS A GUARDAR NUESTRO CACHE REDIS
                //VAMOS A DARLE TIEMPO AL GUARDAR UN ELEMENTO EN REDIS
                //SI NO LE DECIMOS NADA, POR DEFECTO REDIS ALMACENA LOS DATOS
                24 horas
                    await this.database.StringSetAsync("favoritos", jsonProductos
                        , TimeSpan.FromMinutes(30));
                }
            }
        }
    }
}

```

Sobre Details incluimos un nuevo Link para seleccionar un Favorito

```

DETAILS.CSHTML
@model MvcCoreCacheRedis.Models.Producto

@{
    ViewData["Title"] = "Details";
}





### @Model.Nombre



Precio @Model.Precio



@Model.Descripcion


    Seleccionar favorito


```

Sobre \_Layout incluimos un Link para visualizar los favoritos

```

_LAYOUT.CSHTML
</li>
<li class="nav-item">
    <a class="nav-link text-dark" asp-controller="Productos" asp-action="Favoritos">
        Favoritos
    </a>
</li>

```

Sobre Program resolvemos las dependencias de nuestro Service

PROGRAM

```

// Add services to the container.
builder.Services.AddTransient<ServiceCacheRedis>();
string cacheRedisKeys =
    builder.Configuration.GetValue<string>(
        "AzureWave:CachRedis");

```

En ProductosController inyectamos el nuevo Servicio

PRODUCTOSCONTROLLER

```

public class ProductosController : Controller
{
    private RepositoryProductos repo;
    private ServiceCacheRedis service;

    0 references
    public ProductosController
        (RepositoryProductos repo,
        ServiceCacheRedis service)
    {
        this.service = service;
        this.repo = repo;
    }
}

```

```

public async Task<IActionResult> Favoritos()
{
    List<Producto> productos = await this.service.GetProductosFavoritosAsync();
    return View(productos);
}

0 references
public async Task<IActionResult> SeleccionarFavorito(int idproducto)
{
    //BUSCAMOS EL PRODUCTO DENTRO DEL REPO
    Producto producto = this.repo.FindProducto(idproducto);
    await this.service.AddProductoFavoritoAsync(producto);
    return RedirectToAction("Favoritos");
}

0 references
public async Task<IActionResult> DeleteFavorito(int idproducto)
{
    await this.service.DeleteProductoFavoritoAsync(idproducto);
    return RedirectToAction("Favoritos");
}

```

#### FAVORITOS.CSHTML

```

@model IEnumerable<MvcCoreCacheRedis.Models.Producto>
@{
    ViewData["Title"] = "Favoritos";
}
<h1>Favoritos Cache Redis</h1>
@if (Model != null)
{
    <table class="table table-bordered table-warning">
        <thead>
            <tr>
                <th>@Html.DisplayNameFor(model => model.Nombre)</th>
                <th>@Html.DisplayNameFor(model => model.Descripcion)</th>
                <th>@Html.DisplayNameFor(model => model.Precio)</th>
                <th>@Html.DisplayNameFor(model => model.Imagen)</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td>@Html.DisplayFor(modelItem => item.Nombre)</td>
                    <td>@Html.DisplayFor(modelItem => item.Descripcion)</td>
                    <td>@Html.DisplayFor(modelItem => item.Precio)</td>
                    <td>
                        
                        <a href="#" asp-controller="Productos"
                           asp-action="DeleteFavorito"
                           asp-route-idproducto="@item.IdProducto"
                           class="btn btn-danger">
                            Quitar favorito
                        </a>
                    </td>
                </tr>
            }
        </tbody>
    </table>
}

```

## Favoritos Cache Redis

Nombre	Descripción	Precio	Imagen	
Puma Drift	La Drift Cat 5 Core es una fantástica incorporación a nuestra gama de deportes de motor y estilo de vida y es la elección perfecta para cualquier aficionado a los deportes de motor. Cuenta con una silueta veloz de perfil bajo confeccionada con una moderna combinación de cuero plena flor y textil	70		<a href="#">Quitar favorito</a>
JHayber New Olimpo	Su diseño no era para tirar cohetes, pero todos queríamos aquellas zapatillas blancas con sus tres bandas azules, su puntera perforada y esa llama olímpica en su logo. A resistentes no le ganaba ninguna otra marca y con su pedazo suela podíamos estar seguros de no clavarnos nada	45		<a href="#">Quitar favorito</a>
Nike Air Max BW	El diseño original de los 90, se actualiza con un tejido Nike Tech Ultramesh para hacerlas más transpirables y una mediasuela ultra esculpida para ser más ligeras que nunca. Nike AirMax BW Ultra.	100		<a href="#">Quitar favorito</a>

El siguiente paso es crear una aplicación de consola que recupere los Favoritos y los muestre, es decir, una App distinta.

Vamos a realizar una aplicación de Consola en la misma solución llamada **ConsoleCacheRedis**

 **Newtonsoft.Json** by dotnetfoundation, jamesnk, newtonsoft, 6,038 downloads  
Json.NET is a popular high-performance JSON framework for .NET

 **StackExchange.Redis** by marcgravell, nick.clever, StackExchange, 604M downloads  
High performance Redis client, incorporating both synchronous and asynchronous usage.

Copiamos la clase **Producto**, **Service** y la clase **Multiplexer** a nuestro proyecto y les cambiamos el Name Space

PROGRAM

```
using ConsoleCacheRedis;

Console.WriteLine("Testing Cache Redis");

ServiceCacheRedis service = new ServiceCacheRedis();
List<Producto> favoritos =
    await service.GetProductosFavoritosAsync();
if (favoritos == null)
{
    Console.WriteLine("No tenemos favoritos");
}
else
{
    int i = 1;
    foreach (Producto p in favoritos)
    {
        Console.WriteLine(i + " - " + p.Nombre);
        i += 1;
    }
}
```

Y ya visualizaremos los productos en las dos aplicaciones

Nombre	Descripción	Precio	Imagen
Puma Drift	La Drift Cat 5 Core es una fantástica incorporación a nuestra gama de deportes de motor y estilo de vida y es la elección perfecta para cualquier aficionado a los deportes de motor. Cuenta con una silueta veloz de perfil bajo confeccionada con una moderna combinación de cuero plena flor y textil	70	
Nike Air Max BW	El diseño original de los 90, se actualiza con un tejido Nike Tech Ultramesh para hacerlas más transpirables y una mediasuela ultra esculpida para ser más	100	

Conclusiones:

- 1) Deberíamos tener un servicio llamado **ServiceSession**
- 2) Dentro del servicio, si el usuario NO se ha validado, utilizamos Session en nuestro código para devolver los favoritos o Añadir favoritos
- 3) Dentro del servicio, si el usuario se ha validado, utilizamos Cache Redis en nuestro código para devolver los favoritos o Añadir favoritos.
- 4) Si el usuario ya tenía datos en Session ANTES de validarse, cuando se valide, copiamos los datos de Session a Redis.

Necesito poder recuperar la cadena de conexión de Configuration dentro de la clase static **HelperMultiplexer**

## AZURE COSMOS DB

viernes, 4 de abril de 2025 9:11

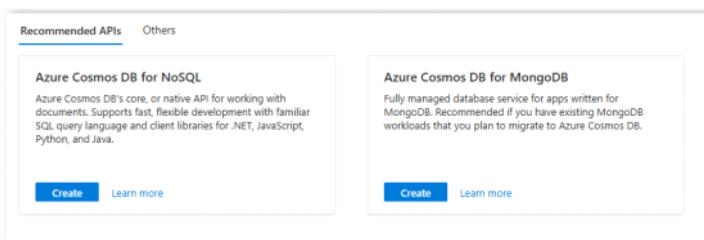
Creamos un nuevo grupo de recursos llamado rg-viernes

Buscamos el servicio de Cosmos

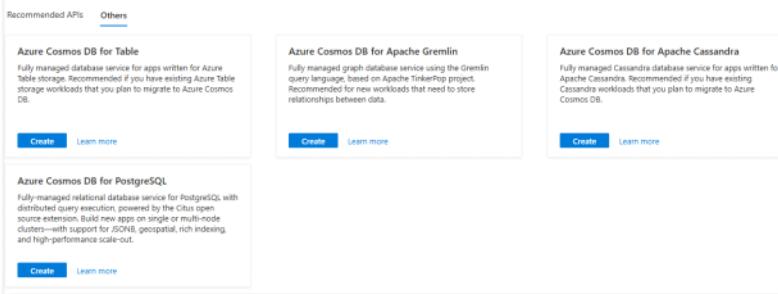


Cosmos Db es la base de datos oficial para Non SQL de Microsoft.

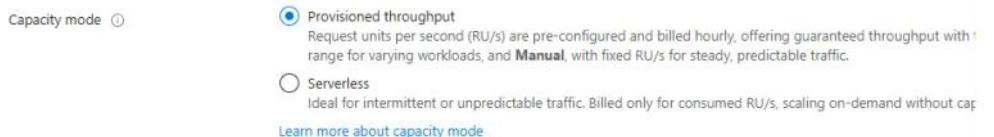
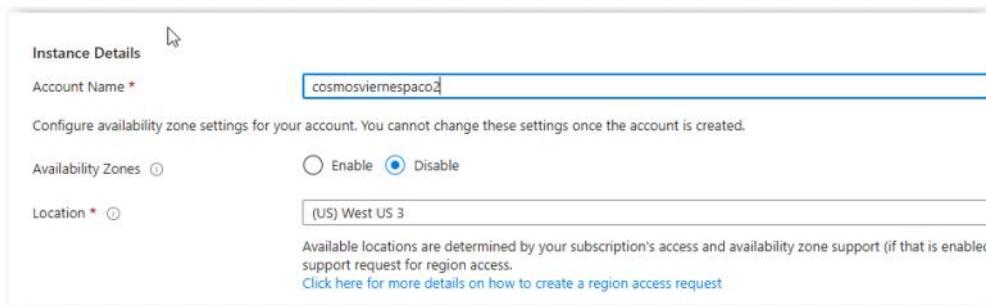
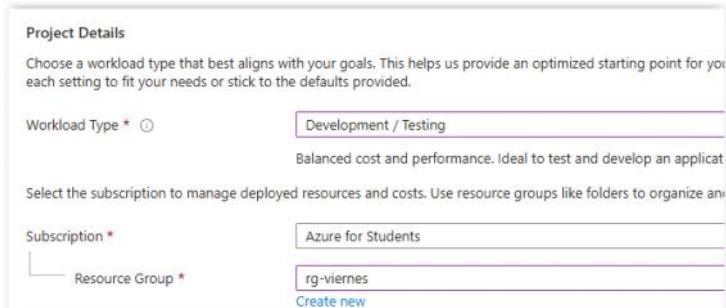
Seleccionamos For NO SQL



Create an Azure Cosmos DB account



Workload type: Development



With Azure Cosmos DB free tier, you will get the first 1000 RU/s and 25 GB of storage for free in an account. You can enable free tier on up to one account per month.

Create Azure Cosmos DB Account - Azure Cosmos DB for NoSQL ...

Basics Global distribution Networking Backup Policy Encryption Tags Review + create

Configure global distribution and regional settings for your account. You can also change these settings after the account is created.

Geo-Redundancy  Enable  Disable

Multi-region Writes  Enable  Disable

Basics Global distribution **Networking** Backup Policy Encryption Tags Review + create

**Network connectivity**

You can connect to your Azure Cosmos DB account either publically, via public IP addresses or service endpoints, or privately, using a private endpoint.

Connectivity method \*  All networks  
 Public endpoint (selected networks)  
 Private endpoint

All networks will be able to access this CosmosDB account. <http://aka.ms/network-security>

**Connection Security Settings**

Minimum Transport Layer Security Protocol

Basics Global distribution Networking **Backup Policy** Encryption Tags Review + create

Azure Cosmos DB provides three different backup policies. You will not be able to switch to Periodic mode once you adopt Continuous mode. [Learn more](#) about the differences of

Backup policy  Periodic  
Backup is taken at periodic interval based on your configuration  
 Continuous (7 days)  
Provides backup window of 7 days / 168 hours and you can restore to any point of time within the window. This mode is available f  
 Continuous (30 days)  
Provides backup window of 30 days / 720 hours and you can restore to any point of time within the window. This mode has cost im

Backup interval  Minute(s)

Backup retention  Hours(s)

Basics Global distribution Networking Backup Policy **Encryption** Tags Review + create

**Data Encryption**

Azure Cosmos DB encryption protects your data at rest by seamlessly encrypting your data as it's written in our datacenters, and automatically

By default your Azure Cosmos DB account is encrypted at rest using service-managed keys. At the moment, you will not be able to switch between these key types for your account. [Learn More](#)

Data Encryption \*  Service-managed key  
 Customer-managed key (CMK)

También tenemos un emulador de Cosmos Db para jugar.  
Se necesita de Docker para trabajar.

Funciona mediante **Containers** que almacenan los datos

Dentro de cada Container tenemos los Items, que son los registros en formato JSON

**Quickstart Guide: Set Up and Run a Sample Application with Azure Cosmos DB**

Follow these steps to quickly set up and explore the capabilities of Azure Cosmos DB. The sample app includes pre-configured functions to test and interact with your database, so you can focus on exploring features without writing code from scratch. Each step includes detailed explanations and tips to guide you along the way.

**Getting Started Tip**

As you follow these steps, it can be helpful to open the Azure Portal in a second browser tab. This way, you can reference the quickstart in one tab while working on your account in the other.

**Choose a platform**

- .NET
- Java
- TypeScript
- Python
- Go

**1 Setup your Database and Container to store data**

In Azure Cosmos DB, data is stored in containers. Click the button to create the following resources within this account, which are required for use with the sample application:

- A database named '**cosmicworks**'.
- A container named '**products**'. If your account uses provisioned throughput, the container is created with the minimum throughput capacity (RU/s) for the account.

[Create sample applic...](#)

Desde nuestra propia aplicación vamos a realizar todo, crearemos los containers y También los datos.

En nuestra aplicación almacenaremos Vehículos. Algunos vehículos tendrán motor y otros no.

Creamos una nueva aplicación llamada **MvcCoreAzureCosmos**

**Newtonsoft.Json** by dotnetfoundation, jamesnk, newtonsoft, 6,04B downloads 13.0.3  
Json.NET is a popular high-performance JSON framework for .NET

**Microsoft.Azure.Cosmos** by azure-sdk, Microsoft, 122M downloads 3.48.0  
This client library enables client applications to connect to Azure Cosmos DB via the NoSQL API. Azure Cosmos DB is a globally distributed, multi-model database service. F...

Sobre **Models** creamos una nueva clase llamada **Motor**

**MOTOR**

```
public class Motor
{
    public string Tipo { get; set; }
    public int Caballos { get; set; }
    public int Cilindrada { get; set; }
    public bool Turbo { get; set; }
}
```

Creamos una clase llamada **Coche**

Toda clase dentro de Cosmos Db debe contener un **Partition Key**

Si no queremos utilizar **Partition Key** en nuestros objetos, debemos indicar que el Partition Key será el **Item ID**, llamado **id**

**COCHE**

```

public class Coche
{
    [JsonProperty(PropertyName = "id")]
    0 references
    public int Id { get; set; }
    0 references
    public string Marca { get; set; }
    0 references
    public string Modelo { get; set; }
    0 references
    public string Imagen { get; set; }
    //NUESTRAS CLASES DINAMICAS
    0 references
    public Motor Motor { get; set; }
}

```

Creamos una carpeta llamada **Services** y una clase llamada **ServiceCosmosDb**

#### SERVICECOSMOSDB

```

public class ServiceCosmosDb
{
    //DENTRO DE COSMOS TRABAJAMOS CON CLIENT Y CONTAINERS
    //DENTRO DE LOS CONTAINERS ESTAN LOS ITEMS
    //DESDE EL CODIGO VAMOS A CREAR UN CONTAINER TAMBIEEN.

    //RECIBIREMOS DOS CLASES, UNA EL CosmosClient PARA
    //TRABAJAR CON CONTAINERS
    //RECIBIREMOS UN CONTAINER QUE SERA NUESTRA TABLA
    private CosmosClient clientCosmos;
    private Container containerCosmos;

    public ServiceCosmosDb(CosmosClient client,
                           Container container)
    {
        this.containerCosmos = container;
        this.clientCosmos = client;
    }

    //VAMOS A CREAR UN METODO PARA CREAR NUESTRA BASE DE DATOS
    //Y DENTRO NUESTRO CONTAINER PARA LOS ITEMS.
    public async Task CreateDatabaseAsync()
    {
        //CREAMOS PRIMERO LA BASE DE DATOS
        await this.clientCosmos.CreateDatabaseIfNotExistsAsync
            ("vehiculoscosmos");
        //DENTRO DE ESTA BASE DE DATOS, CREAMOS NUESTROS CONTAINER
        ContainerProperties properties =
            new ContainerProperties("containercoches", "/id");
        //CREAMOS EL CONTAINER DENTRO DE NUESTRA BBDD
        await this.clientCosmos.GetDatabase("vehiculoscosmos")
            .CreateContainerIfNotExistsAsync(properties);
    }

    //METODO PARA INSERTAR ELEMENTOS DENTRO DE COSMOS
    public async Task InsertCocheAsync(Coche car)
    {
        //EN EL MOMENTO DE INSERTAR, COSMOS NO SABE
        //ASIGNAR AUTOMATICAMENTE SU PARTITION KEY
        //DEBEMOS DECIRSELO DE FORMA EXPLICITA
        await this.containerCosmos
            .CreateItemAsync<Coche>
            (car, new PartitionKey(car.Id));
    }

    public async Task<List<Coche>> GetCochesAsync()
    {
        //UNA BASE DE DATOS COSMOS NO SABE EL NUMERO DE REGISTROS
        //REALES.
        //DEBEMOS LEER UTILIZANDO UN BUCLE WHILE MIENTRAS
        //QUE EXISTAN REGISTROS
        var query =
            this.containerCosmos.GetItemQueryIterator<Coche>();
        List<Coche> coches = new List<Coche>();
        while (query.HasMoreResults)
        {
            var results = await query.ReadNextAsync();
            //SON MULTIPLES COCHES LO QUE DEVUELVE, SE ALMACENAN
            //DENTRO DE NUESTRA COLECCION A LA VEZ
            coches.AddRange(results);
        }
        return coches;
    }

    public async Task UpdateCocheAsync(Coche car)
    {
        //VOY A UTILIZAR UN METODO LLAMADO upsert
        //Dicho metodo, si encuentra el coche lo modifica
        //y si no lo encuentra, lo inserta
        await this.containerCosmos.UpsertItemAsync<Coche>
            (car, new PartitionKey(car.Id));
    }

    public async Task DeleteCocheAsync(string id)
    {
        await this.containerCosmos.DeleteItemAsync<Coche>
            (id, new PartitionKey(id));
    }

    //METODO PARA BUSCAR UN COCHE POR SU ID
    public async Task<Coche> FindCocheAsync(string id)
    {
        ItemResponse<Coche> response = await
            this.containerCosmos.ReadItemAsync<Coche>
            (id, new PartitionKey(id));
        return response.Resource;
    }
}

```

Para poder acceder a los recursos de Cosmos, necesitamos sus Keys.

The screenshot shows the 'Keys' section of the Azure Cosmos DB management interface. It displays four types of keys: Primary Key, Secondary Key, Primary Connection String, and Secondary Connection String. Each key is represented by a text input field with a redacted value. Below each field is a status message indicating it was last regenerated on 4/4/2025 (0 days ago). A 'Show Primary Connection String' button is located at the bottom right.

Sobre **Program** resolvemos las dependencias.

Necesitamos el nombre de la base de datos: **vehiculoscosmos**  
El nombre del Container **containercoches**

#### PROGRAM

```
// Add services to the container.
string connectionStringCosmos =
    builder.Configuration.GetConnectionString("CosmosDb");
CosmosClient client = new CosmosClient(connectionStringCosmos);
Container container =
    client.GetContainer("vehiculoscosmos", "containercoches");
builder.Services.AddSingleton<CosmosClient>(x => client);
builder.Services.AddTransient<Container>(x => container);
builder.Services.AddTransient<ServiceCosmosDb>();
```

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

```
COCHESCONTROLLER
public class CochesController : Controller
{
    private ServiceCosmosDb service;
    public CochesController(ServiceCosmosDb service)
    {
        this.service = service;
    }
    public IActionResult Index()
    {
        return View();
    }
    [HttpPost]
    public async Task<IActionResult> Index(string accion)
    {
        await this.service.CreateDatabaseAsync();
        ViewData["MENSAJE"] = "Database Cosmos Creada!!!";
        return View();
    }
    public async Task<IActionResult> MisCoches()
    {
        List<Coche> cars =
            await this.service.GetCochesAsync();
        return View(cars);
    }
    public IActionResult Create()
    {
        return View();
    }
    [HttpPost]
    public async Task<IActionResult> Create(Coche car)
    {
        await this.service.InsertCocheAsync(car);
        return RedirectToAction("MisCoches");
    }
    public async Task<IActionResult> Delete(string id)
    {
        await this.service.DeleteCocheAsync(id);
        return RedirectToAction("MisCoches");
    }
    public async Task<IActionResult> Details(string id)
```

```

        Coche car = await
            this.service.FindCocheAsync(id);
        return View(car);
    }

    public async Task<IActionResult> Edit(string id)
    {
        Coche car = await this.service.FindCocheAsync(id);
        return View(car);
    }

    [HttpPost]
    public async Task<IActionResult> Edit(Coche car)
    {
        await this.service.UpdateCocheAsync(car);
        return RedirectToAction("MisCoches");
    }
}

```

#### INDEX.CSHTML

```

<h1 style="color:blue">
    Cosmos Db Objetos
</h1>
<p style="color:red">
    Aquí creamos nuestros recursos en Cosmos, solamente UNA VEZ
</p>
<form method="post">
    <button class="btn btn-outline-danger" name="accion" value="">
        Crear Database y Container
    </button>
</form>
<h3 style="color:blue">@ViewData["MENSAJE"]</h3>
El resto de vista las haremos mediante Scaffolding

```

A continuación, vamos a implementar la funcionalidad para incluir Motor en la Vista **Create.cshtml**

#### CREATE.CSHTML

```

@model MvcCoreAzureCosmos.Models.Coche

 @{
    ViewData["Title"] = "Create";
}

@section Scripts {
    <script>
        $(document).ready(function() {
            $("#existemotor").click(function(){
                $("#capamotor").toggle();
            })
        })
    </script>
}

<h1>Create</h1>
<h4>Coche</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Id" class="control-label"></label>
                <input asp-for="Id" class="form-control" />
                <span asp-validation-for="Id" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Marca" class="control-label"></label>
                <input asp-for="Marca" class="form-control" />
                <span asp-validation-for="Marca" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Modelo" class="control-label"></label>
                <input asp-for="Modelo" class="form-control" />
                <span asp-validation-for="Modelo" 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>
            <input type="checkbox" id="existemotor"
                   name="existemotor" value="true"/>
            Coche con motor
            <div id="capamotor" style="display: none">
                <label>Tipo:</label>
                <input type="text" name="Motor.Tipo"
                       class="form-control"/>
                <label>Cilindrado:</label>
                <input type="text" name="Motor.Cilindrada"
                       class="form-control"/>
                <label>Caballos:</label>
                <input type="text" name="Motor.Caballos"
                       class="form-control"/>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>
<a asp-controller="Coches" asp-action="MisCoches">Back to List</a>

```

#### COCHESCONTROLLER

```
[HttpPost]
0 references | 0 changes | 0 authors, 0 changes
public async Task<IActionResult> Create
    (Coche car, string existemotor)
{
    if (existemotor == null)
    {
        car.Motor = null;
    }
    await this.service.InsertCocheAsync(car);
    return RedirectToAction("MisCoches");
}
```

En Details dibujamos el Motor si tiene Motor el objeto

DETAILS.CSHTML

```
@if (Model.Motor != null){
    <dd class="col-sm-10">
        <p>Tipo: @Model.Motor.Tipo</p>
        <p>Cilindrada: @Model.Motor.Cilindrada</p>
        <p>Caballos: @Model.Motor.Caballos</p>
    </dd>
}
```



Home Creación Cosmos Coches Cosmos Privacy

## Mis Coches de Cosmos

[Create New](#)

Id	Marca	Modelo	Imagen			
1	DMG	DELOREAN		<a href="#">Details</a>	<a href="#">Edit</a>	<a href="#">Details</a>
2	POCO	YO		<a href="#">Details</a>	<a href="#">Edit</a>	<a href="#">Details</a>
3	DMG	DELOREAN MOTOR		<a href="#">Details</a>	<a href="#">Edit</a>	<a href="#">Details</a>

Y podremos visualizar los coches creados con Motor

Home conta...items

SELECT \* FROM c Type a query predicate (e.g., WHERE c.id='1'), or choose one from the drop down list, or leave empty to query all d

<input type="checkbox"/> /id	...
<input type="checkbox"/> 1	
<input type="checkbox"/> 2	
<input checked="" type="checkbox"/> 3	

```

1  {
2      "id": "3",
3      "Marca": "DMG",
4      "Modelo": "DELOREAN MOTOR",
5      "Imagen": "https://upload.wikimedia.org/wikipedia/commons/
6      "Motor": [
7          {
8              "Tipo": "GASOLINA",
9              "Caballos": 150,
10             "Cilindrada": 150
11         },
12         "_rid": "LIdiAOi9HxYDAAAAAAA==",
13         "_self": "dbs/LIdiAOi9HxY/colls/LIdiAOi9HxY/docs/LIdiAOi9HxY
14         "_etag": "\\"0000bdb7-0000-0000-67efc8590000\"",
15         "_attachments": "attachments/",
16         "_ts": 1743767641
17     }
18 }
```

Las búsquedas dentro de Cosmos Db son muy simples, de hecho, utilizan la sintaxis de SQL.

Si nos fijamos en el portal:

The screenshot shows the Azure Cosmos DB portal's query editor. The top bar has tabs for 'Home' and 'conta\_items'. The main area contains the SQL query: 'SELECT \* FROM c WHERE c.Motor.Caballos > 100'. Below the query is a text input field with placeholder text: 'Type a query predicate (e.g., WHERE c.id='1'), or choose one from...'. There are also some dropdown and search icons.

Los objetos dentro de cosmos siempre se llamarán c

Todos los que tengan determinados Caballos

SELECT \* FROM c WHERE c.Motor.Caballos > 100

Queremos filtrar por una Marca

SELECT \* FROM c WHERE c.Marca = "DMG"

The screenshot shows the Azure Cosmos DB portal's results view. The top bar has tabs for 'Home' and 'conta\_items'. The results table shows three items. Item 1 is selected and highlighted in blue. The table columns are '/id' and '1'. To the right of the table, the JSON document for item 1 is displayed with line numbers 1 through 11. The JSON content includes fields like id, Marca, Modelo, Imagen, Motor, \_rid, \_self, \_etag, \_attachments, and \_ts.

/id	1
	{ "id": "1", "Marca": "DMG", "Modelo": "DELOREAN", "Imagen": "https://upload.wikimedia.org/wikipedia/commons/ "Motor": null, "_rid": "LIdiAOw9hxYBAAAAAAA==", "_self": "dbs/LIdiAA=/colls/LIdiAOw9hxY=/docs/LIdiAOw9hxY "_etag": "\\"0000a2b7-0000-4d00-0000-67efbfbe20000\\", "_attachments": "attachments/", "_ts": 1743765474 }
	3

En Cosmos Db no existe la inyección SQL a pesar de que la sintaxis utiliza SQL.

Eso decir, los filtros NO se realizan con parámetros, sino que se concatena

#### METODO PARA BUSCAR COCHES

##### SERVICECOSMOSDB

```
public async Task<List<Coche>> GetCochesMarcaAsync  
(string marca)  
{  
    string sql =  
        "select * from c where c.Marca='" + marca + "'";  
    //PARA APLICAR LOS FILTROS SE UTILIZA UNA CLASE  
    //LLAMADA QueryDefinition  
    QueryDefinition definition =  
        new QueryDefinition(sql);  
    var query =  
        this.containerCosmos.GetItemQueryIterator<Coche>  
        (definition);  
    List<Coche> coches = new List<Coche>();  
    while (query.HasMoreResults)  
    {  
        var results = await query.ReadNextAsync();  
        coches.AddRange(results);  
    }  
    return coches;  
}
```

# DOCKER

lunes, 7 de abril de 2025 9:07

Comenzamos instalando desde la siguiente URL

<https://docs.docker.com/desktop/setup/install/windows-install/>

Un docker es una imagen que contiene una serie de herramientas listas para trabajar con ellas directamente sin instalación.  
Es una forma de utilizar accesos de Producción en Development.

Son imágenes de sistemas operativos con distintas arquitecturas en su interior.  
En realidad, cuando nosotros necesitamos realizar "algo" de infraestructura necesitamos un entorno dónde probarlo.

Gracias a Docker, **en mi casa funciona** se ha terminado.

Pensemos en lo siguiente:

Tenemos un servicio API que utiliza Postgres para la base de datos.  
Tenemos un MVC que consume dicho servicio API.

Nuestro entorno de producción donde el cliente va a tener su servidor es  
Una máquina Linux Ubuntu con un Apache.

¿Qué pasos tenemos que hacer para comprobar el funcionamiento correcto de  
Nuestra App en producción?

- 1) Instalar un VirtualBox o VM Ware
- 2) Instalar nuestro sistema operativo Ubuntu
- 3) Instalar Postgres en Ubuntu
- 4) Instalar un servidor Apache
- 5) Poner nuestra App dentro de la VM y visualizar posibles errores.

¿Qué haríamos con Docker?

- 1) Buscamos una imagen de Ubuntu con Apache
- 2) Instalamos Postgres y probamos App.

Un Docker no es un S.O, es una funcionalidad de una imagen de un S.O que no requiere  
Instalación y nos permite utilizar cualquier característica

Un Docker es una imagen y tendremos un **Container** para desplegar la imagen

Una vez instalado, nos damos de alta en la siguiente URL

<https://hub.docker.com/>

Esto es solamente para S.O Windows o MacOS

Esta URL es un repositorio global al estilo de GitHub.

Aquí podemos encontrar todas las imágenes que deseemos.

Para trabajar con Docker se realiza mediante **línea de comandos**.

Tenemos imágenes para descargar y tenemos Containers con las imágenes desplegadas

```
C:\Users\Profesor MCSD Mañana
λ docker --version
Docker version 27.5.1, build 9f9e405

C:\Users\Profesor MCSD Mañana
λ docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED      STAT
US            PORTS          NAMES
06a0aa8e4b69   gvenzl/oracle-xe   "container-entrypoin..."   7 weeks ago   Exit
ed (143) 6 weeks ago           fervent_visvesvaraya
```

Para descargar una imagen y ejecutarla en un container, buscamos la imagen y la  
Desplegamos mediante **docker run NOMBREIMAGEN**

```
C:\Users\Profesor MCSD Mañana
λ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
e6590344b1a5: Download complete
Digest: sha256:7e1a4e2d11e2ac7a8c3f768d4166c2defeb09d2a750b
19
Status: Downloaded newer image for hello-world:latest
```

Las imágenes de Docker no tienen parte gráfica

Si deseamos compilar una App en un Docker, tenemos dos opciones:

- 1) Azure Container Registry
- 2) Locally Docker

Los pasos a realizar siempre son los mismos:

- 1) Indicar la imagen a utilizar
- 2) Compilar el proyecto
- 3) Descargar los Nuget
- 4) Publicar App dentro de Docker

Todo esto se realiza mediante un fichero de despliegue de Apps Net Core llamado **Dockerfile**

El comando interno que utiliza Net Core para trabajar es **dotnet**

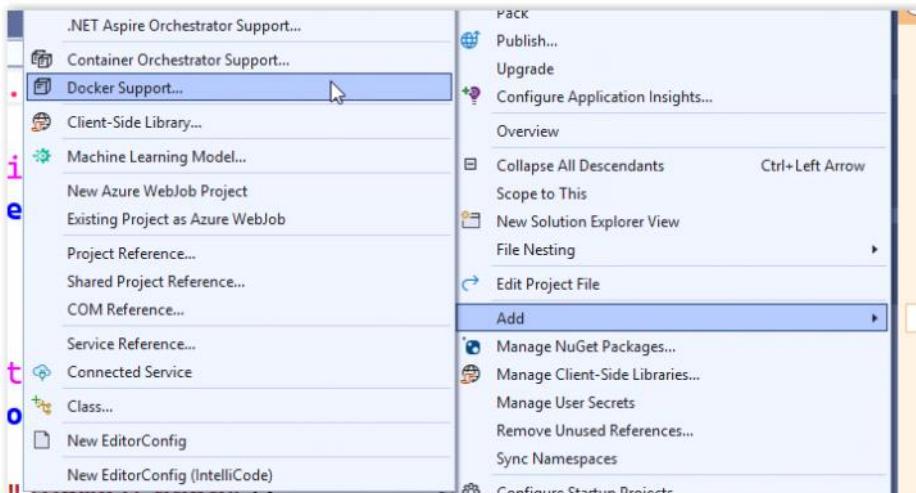
Para probarlo, vamos a crear una nueva App Mvc llamada **AppDockers**

Sobre **Program** incluimos la siguiente línea

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.WebHost.UseUrls("http://0.0.0.0:80");
var app = builder.Build();
```

Sobre nuestra nueva aplicación, creamos un nuevo fichero llamado **Dockerfile**



#### DOCKERFILE

```
#LA IMAGEN A DESCARGAR
FROM mcr.microsoft.com/dotnet/aspnet:9.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

#ESTA ES LA IMAGEN QUE NECESITA EL COMPILADOR DE NET
FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
```

```

COPY *.csproj ./
#DESCARGAR TODOS LOS NUGET DEL PROYECTO
RUN dotnet restore
COPY .
WORKDIR "/src/AppDockers"
#COMPILA EL PROYECTO EN LA IMAGEN DE DOCKER
RUN dotnet build "/src/AppDockers.csproj" -c Release -o /app/build

#CREAMOS UN PUNTO PARA PUBLICAR LA IMAGEN EN EL DOCKER
FROM build AS publish
RUN dotnet publish "/src/AppDockers.csproj" -c Release -o /app/publish

#COPIA EL PROYECTO PUBLICADO DENTRO DE LA APP PARA SU EJECUCION
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "AppDockers.dll"]

```

Cuando tenemos una imagen creada, podemos utilizar F5 o generar imágenes a partir de nuestro Container

Si necesitamos generar imágenes de Producción para desplegarlas o para distribuirlas, Se realiza mediante línea de comandos.

Abrimos la carpeta de nuestro proyecto hasta el fichero **Dockerfile**

```

C:\Users\Profesor MCSD Mañana\Documents\Proyectos\AppDockers\AppDockers
\ls
AppDockers.csproj.mandos.      bin/          obj/          wwwroot/
AppDockers.csproj.user        Controllers/   Program.cs
appsettings.Development.json  Dockerfile    Properties/
appsettings.json              proyecto hasModels/hero Dockerfile

```

Para crear imágenes, necesitamos el siguiente comando:

```

docker build -t miapp:v1
docker build -t miapp:v2

```

Si es una imagen de producto final

```
docker build -t miapp:latest
```

Creamos una imagen con nuestro proyecto y nuestra imagen de Docker

```

C:\Users\Profesor MCSD Mañana\Documents\Proyectos\AppDockers\AppDockers
\ docker build -t appdocker:latest .

```

Veremos que ya tenemos una imagen

CONTAINER ID	IMAGE	COMMAND	CREATED
6f459bbef4cc	appdockers:dev	"dotnet --roll-forward=if-reloadable"	8 minutes ago

Esta imagen lleva en su interior una App de tipo MVC, como también podríamos instalar Aplicaciones como un Servidor Web o lo que deseemos.

Una imagen es solo un conjunto de elementos, si deseamos visualizar o desplegar la Imagen necesitamos **docker run**

Vamos a lanzar nuestra imagen en un puerto determinado

```
docker run -p 8001:80 --rm -d appdockers:latest
```

CONTAINER ID	IMAGE	COMMAND	CREATED
c60a6a55dc41	appdockers:latest	"dotnet AppDockers.d..."	39 seconds ago

Esto nos genera una imagen funcional.

Dicha imagen, con todo lo que tuvieramos dentro podemos perfectamente publicarla Para que otros la utilicen y les de error.

En Docker Hub podemos publicar nuestras propias imágenes.

The screenshot shows the Docker Hub interface for creating a new repository. At the top, there's a search bar labeled 'Search by repository name' with 'dockerlunes2025' typed in, a dropdown menu set to 'All content', and a prominent blue 'Create a repository' button with a hand cursor icon. Below this, the URL 'Repositories / Create' is visible, along with a message indicating 'Using 0 of 1 private repositories.' and a link to 'Get more'. The main form for creating a repository includes fields for 'Repository Name \*' (set to 'dockerlunes2025'), 'Short description' (containing the text 'Esto es un docker que no funciona ahora mismo...'), and a note explaining the short description's purpose. To the right, a section titled 'Pushing images' provides instructions on how to push a new image using the CLI, with a command example: 'docker tag local-image:tagname | docker push new-repo:tagname'. It also includes a note to replace 'tagname' with the desired image repository tag.

```
λ docker login
Authenticating with existing credentials...
Login Succeeded
```

```
C:\Users\Profesor MCSD Mañana\Documents\Proyectos\AppDockers\AppDockers
λ docker tag appdocker:latest serraguti/dockerlunes2025

C:\Users\Profesor MCSD Mañana\Documents\Proyectos\AppDockers\AppDockers
λ docker push serraguti/dockerlunes2025
```

Ya somos parte del mundo con nuestra imagen vamos a realizar una práctica para Desplegar una aplicación utilizando un docker y comunicando con él.

Vamos a tener dos docker en paralelo, uno con nuestra app y otro con MySql.

Lo que haremos es modo real, imaginando que, por un lado, el cliente tiene un MySql y En otra máquina de Linux tiene desplegado nuestro MVC.



Creamos una base de datos llamada **dockers** y ejecutamos el script

```

use dockers;

CREATE TABLE COMICS
(
IDCOMIC INT PRIMARY KEY
, NOMBRE NVARCHAR(150)
, IMAGEN NVARCHAR(150));

```

```

INSERT INTO COMICS VALUES (1, 'Spiderman MYSQL', 'https://3.bp.blogspot.com/-')
INSERT INTO COMICS VALUES (2, 'Batman MYSQL', 'https://dam.smashmexico.com.mx')
INSERT INTO COMICS VALUES (3, 'Wolverine MYSQL', 'https://vignette.wikia.noco')
INSERT INTO COMICS VALUES (4, 'Spawn MYSQL', 'https://cdn.imagecomics.com/ass')
INSERT INTO COMICS VALUES (5, 'Mortadelo y Filemon MYSQL', 'https://images-na')

```

Vamos a realizar una aplicación para mostrar y crear comics.

Creamos una nueva App llamada **MvcComicsDocker**

 Microsoft.EntityFrameworkCore by aspnet, dotnetframework, EntityFramework, Micr  9.0.3  
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Dat...

 MySQL.EntityFrameworkCore by MySQL, 8,62M downloads 9.0.0  
MySQL.EntityFrameworkCore adds support for Microsoft Entity Framework Core.

Sobre **Models** creamos una nueva clase llamada **Comic**

COMIC

```

[Table("COMICS")]
0 references
public class Comic
{
    [Key]
    [Column("IDCOMIC")]
    0 references
    public int IdComic { get; set; }
    [Column("NOMBRE")]
    0 references
    public string Nombre { get; set; }
    [Column("IMAGEN")]
    0 references
    public string Imagen { get; set; }
}

```

Creamos una carpeta llamada **Data** y una clase llamada **ComicsContext**

COMICSCONTEXT

```

public class ComicsContext: DbContext
{
    0 references
    public ComicsContext(DbContextOptions<ComicsContext> options)
        : base(options) { }

    0 references
    public DbSet<Comic> Comics { get; set; }
}

```

Creamos una carpeta llamada **Repositories** y una clase llamada **RepositoryComics**

#### REPOSITORYCOMICS

```
public class RepositoryComics
{
    private ComicsContext context;

    public RepositoryComics(ComicsContext context)
    {
        this.context = context;
    }

    public async Task<List<Comic>> GetComicsAsync()
    {
        return await this.context.Comics.ToListAsync();
    }

    private async Task<int> GetMaxIdComicAsync()
    {
        return await this.context.Comics
            .MaxAsync(z => z.IdComic) + 1;
    }

    public async Task CreateComic (string nombre, string imagen)
    {
        Comic c = new Comic();
        c.IdComic = await this.GetMaxIdComicAsync();
        c.Nombre = nombre;
        c.Imagen = imagen;
        await this.context.Comics.AddAsync(c);
        await this.context.SaveChangesAsync();
    }
}
```

Incluimos la cadena de conexión en **appsettings.json**

#### APPSETTINGS.JSON

```
{
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    },
    "AllowedHosts": "*",
    "ConnectionStrings": {
        "MySQL": "server=localhost;port=3306;user id=root;password=mysql;database=dockers;SSL mode=None"
    }
}
```

Resolvemos las dependencias dentro de Program

#### PROGRAM

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
string connectionString =
    builder.Configuration.GetConnectionString("MySQL");
builder.Services.AddTransient<RepositoryComics>();
builder.Services.AddDbContextPool<ComicsContext>
    (options => options.UseMySQL(connectionString));
builder.Services.AddControllersWithViews();
```

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

#### COMICSCONTROLLER

```
public class ComicsController : Controller
{
    private RepositoryComics repo;

    public ComicsController(RepositoryComics repo)
    {
        this.repo = repo;
    }

    public async Task<IActionResult> Index()
    {
        List<Comic> comics = await this.repo.GetComicsAsync();
        return View(comics);
    }

    public IActionResult Create()
    {
```

```

        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Create(Comic comic)
    {
        await this.repo.CreateComic(comic.Nombre
            , comic.Imagen);
        return RedirectToAction("Index");
    }
}

```

Creamos las vistas con Scaffolding

El siguiente paso es utilizar un MySql de Docker

Pongamos que tenemos el siguiente laboratorio:

- 1) Linux MySql dedicado
- 2) Servidor Linux con MVC

Necesitamos comunicar los dos servicios en Docker.

Necesitamos primero probar si nuestra App se comunica con **MySql Docker**

Descargamos una imagen Docker de Mysql

Lanzamos el docker en Windows

```
docker run --name mysql -e MYSQL_ROOT_PASSWORD=mysql -d mysql:8.0.0
```

```

λ docker run --name mysql -e MYSQL_ROOT_PASSWORD=mysql -d mysql:8.0.0
Unable to find image 'mysql:8.0.0' locally
8.0.0: Pulling from library/mysql
cc158c0bcdef: Download complete

```

Con esta instrucción, podemos visualizar las características de nuestro Docker

```
docker inspect 99fd5091c916
```

```

"82e59bfadd3cf4", 
"Gateway": "172.17.0.1",
"IPAddress": "172.17.0.2",
"IPPrefixLen": 16,
"IPv6Gateway": "",
"GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0,
"DNSNames": null
}

```

Si queremos conectar a nuestra base de datos, tendremos que hacerlo desde el propio Docker.

```
docker exec -it mysql mysql -uroot -p
```

Necesitamos que los Docker se comuniquen entre sí.

Para ello, tenemos dos posibilidades:

- 1) **Docker-compose:** Docker compose utiliza fichero YAML para configurar múltiples servicios de Docker a la vez. Se ejecuta para agrupar Docks como una sola Red/instancia

```

version: '3.7'

services:
  db:
    image: mysql:latest
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: your_root_password
      MYSQL_DATABASE: your_database_name
      MYSQL_USER: your_username
      MYSQL_PASSWORD: your_password
    volumes:
      - ./mysql_data:/var/lib/mysql
    ports:
      - "3306:3306"

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    restart: always
    depends_on:
      - db
    environment:
      PMA_HOST: db
      MYSQL_ROOT_PASSWORD: your_root_password
    ports:
      - "8080:80"

```

- 2) Utilizar un network para todos los servicios. Cuando creamos un Network, podemos incluir ahí cada uno de los elementos Docker que necesitemos y automáticamente se le asigna una IP dentro de la red

Creamos una nueva red en docker

`docker network create my-network`

```

C:\Users\Profesor MCSD Mañana\Documents\Proyectos\AppDockers\AppDockers> docker network create my-network
dd122db2f6a5d66fa85e981930f87708e9d46645adab5e769b6332db27c92736

```

Debemos desplegar nuestro MySql en la red que hemos creado.

`docker run --name mysqlDb --network=my-network -e MYSQL_ROOT_PASSWORD=mysql -d mysql:8.0.0`

```

\ docker run --name mysqlDb --network=my-network -e MYSQL_ROOT_PASSWORD=mysql -d mysql:8.0.0
919e4e3ad320685f17a051ef91b4ba4e524e4ad7f2eda6e0afb494f60933bb96

```

El siguiente paso es conectar con MySql dentro de Docker y crear nuestra bbdd

`docker exec -it mysqlDb mysql -uroot -p`

Creamos la base de datos `dockers`

```

mysql> create database dockers;
Query OK, 1 row affected (0.01 sec)

mysql> use dockers;
Database changed

```

Copiamos los registros con el Script y veremos que ya tenemos montado nuestro servidor MySql

```

mysql> select * from COMICS; | Images-
+----+-----+
| IDCOMIC | NOMBRE | IMAGEN
+----+-----+
| 1 | Spiderman MYSQL | https://3.bp.blogspot.com/
0xxduu-I/AAAAAAALq8/8bXDrdvW50o/s1600/spiderman1.jpg
| 2 | Batman MYSQL | https://dam.smashmexico.co
uploads/2019/11/dc-batman-detective-comics-desfigurando-el-rostro-
| 3 | Wolverine MYSQL | https://vignette.wikia.noc
database/images/d/da/Wolverine_Vol_2_305_Variant_McNiven.jpg
| 4 | Capitan MYSQL | https://cdn.imagescomic.co

```

Podríamos tener de todo dentro de la red, podemos averiguar container a container o inspeccionar la propia red para visualizar los Docker que tenemos dentro con sus IP.

`docker network inspect my-network`

```

    "ConfigOnly": false,
    "Containers": {
        "919e4e3ad320685f17a051ef91b4ba4e524e4ad7f2eda
            "Name": "mysqldb",
            "EndpointID": "89d86cdd6c16b78f7a2b068632a
18f407921", para visualizar los Docker que tenemos dent
            "MacAddress": "02:42:ac:12:00:02",
            "IPv4Address": "172.18.0.2/16",
            "IPv6Address": ""
        },
        "Options": {},
        "Labels": {}
    }
}

```

El siguiente paso es poner en paralelo nuestra App dentro del Docker.

Tengo dos opciones:

- 1) Desplegar mi App dentro del Docker de MySql
- 2) Tener una máquina con servicios dedicados

Debemos comunicar nuestra App con MySql de Docker. Modificamos la cadena de conexión hacia nuestro Docker.

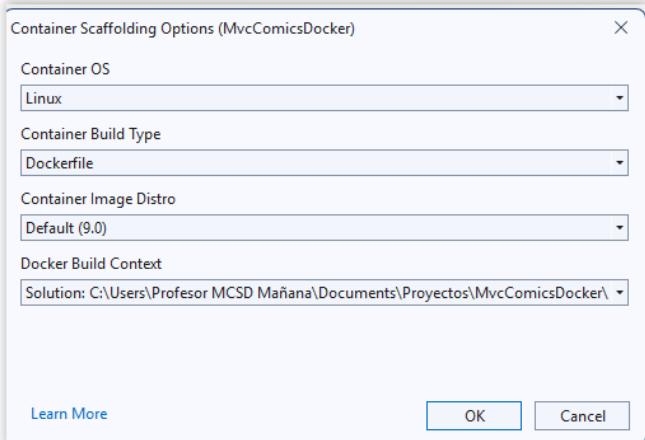
`APPSETTINGS.JSON`

```

{
    "Logging": {
        "LogLevel": {
            "Default": "Warning"
        }
    },
    "AllowedHosts": "*",
    "ConnectionStrings": {
        "MySql": "server=172.18.0.2;port=3306;user id=root;
    }
}

```

Sobre nuestro proyecto, agregamos un nuevo File de Dockerfile



## DOCKERFILE

```
#LA IMAGEN A DESCARGAR
FROM mcr.microsoft.com/dotnet/aspnet:9.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

#ESTA ES LA IMAGEN QUE NECESITA EL COMPILADOR DE NET
FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY *.csproj ./
#DESCARGAR TODOS LOS NUGET DEL PROYECTO
RUN dotnet restore
COPY . .
WORKDIR "/src/MvcComicsDocker"
#COMPILA EL PROYECTO EN LA IMAGEN DE DOCKER
RUN dotnet build "/src/MvcComicsDocker.csproj" -c Release -o /app/build

#CREAMOS UN PUNTO PARA PUBLICAR LA IMAGEN EN EL DOCKER
FROM build AS publish
RUN dotnet publish "/src/MvcComicsDocker.csproj" -c Release -o /app/publish

#COPIA EL PROYECTO PUBLICADO DENTRO DE LA APP PARA SU EJECUCION
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "MvcComicsDocker.dll"]
```

Sobre **Program** incluimos la línea de Victor y de Dani

## PROGRAM

---

```
(options => options.UseMySQL(connectionString));
builder.Services.AddControllersWithViews();
builder.WebHost.UseUrls("http://0.0.0.0:80");
```

---

El siguiente paso es realizar el build para generar una imagen de Docker.

Entramos en la carpeta de nuestro proyecto y escribimos el comando para generar la imagen

`docker build -t mvccomics:latest .`

```
C:\Users\Profesor MCSD Mañana\Documents\Proyectos\MvcComicsDocker> docker build -t mvccomics:latest .
[+] Building 5.6s (6/17)
=> [internal] load build definition from Dockerfile
=> [internal] transferring dockerfile: 883B una imagen de Docker.
=> [internal] load metadata for mcr.microsoft.com/dotnet/
=> [internal] load metadata for mcr.microsoft.com/dotnet/
=> [internal] load c.dockerignore escribimos el comando pa
=> [internal] transfer context: 3B
```

El siguiente paso es desplegar nuestro Docker en un Container a partir de la imagen

**Nota:** El Container debe estar en la misma RED.

`docker run -p 8014:80 --network my-network --rm -d mvccomics:latest`

```
C:\Users\Profesor MCSD Mañana\Documents\Proyectos\MvcComicsDocker> docker run -p 8014:80 --network my-network --rm -d mvccomics:latest
```

Podemos verificar que los dos contenedores están en la misma Red

```
docker network inspect my-network
```

```
"919e4e3ad320685f17a051ef91b4ba4e524e4ad7f2eda6e0afb494f60933bb96": {
    "Name": "mysqlDb",
    "EndpointID": "89d86cdd6c16b78f7a2b068632ab891e51bba626db066d2b628b77018f407921",
    "MacAddress": "02:42:ac:12:00:02",
    "IPv4Address": "172.18.0.2/16",
    "IPv6Address": ""
},
"daef28488b1ff83a160028a11e3c5b5fc63d8c880d461f187842692e6bf23128": {
    "Name": "modest_kowalevski",
    "EndpointID": "37c2d46ee94b1cf55a6d9fdc9de159d84445490674351d134bf10f6888559996",
    "MacAddress": "02:42:ac:12:00:03",
    "IPv4Address": "172.18.0.3/16",
    "IPv6Address": ""
}
```

Y podremos visualizar cómo se comunica con Docker

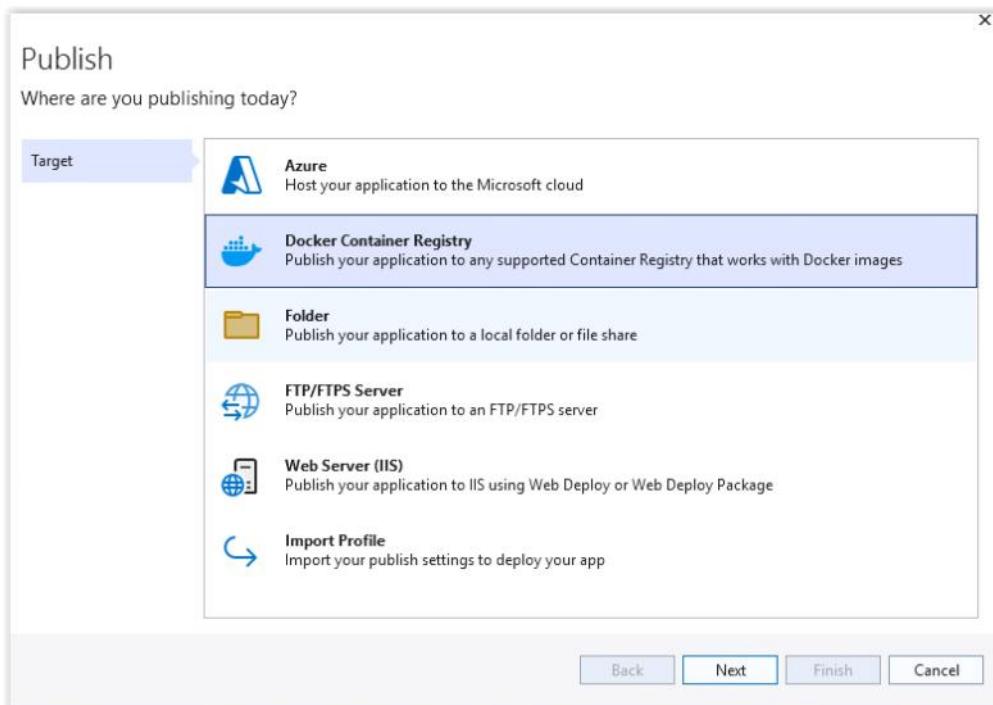
IdComic	Nombre	Imagen
1	Spiderman MYSQL	
2	Batman MYSQL	

Si necesitamos visualizar los errores en producción:

```
// Configure the HTTP request pipeline.
//if (!app.Environment.IsDevelopment())
//{
//    app.UseExceptionHandler("/Home/Error");
//    // The default HSTS value is 30 days. You may want to change this for production scenarios
//    app.UseHsts();
//}
app.UseDeveloperExceptionPage();
```

Como habéis podido comprobar, esto nos permite saber si en un entorno de Producción Real esto funciona o NO.

También podemos desplegar las imágenes de Docker en Azure



# IAAS VIRTUAL MACHINE

martes, 8 de abril de 2025 9:09

Esta unidad tiene que ver con Sistemas, es decir, IaaS (Infrastructure as Service)

Creación de una máquina virtual y despliegue de aplicaciones.

Podemos tener todas las máquinas virtuales que deseemos, excepto MacOS.

**Nota:** Las máquinas virtuales tienen coste, solamente si están encendidas.

Tenemos dos formas de conectar con una máquina virtual:

1) **SSH Keys:** Son claves que almacena Azure, tendremos una clave pública y otra privada y debemos utilizar la privada para conectarnos desde cualquier ubicación remota a la máquina.  
Una de las ventajas es que podemos tener múltiples keys para una misma máquina

2) **User y Password:** Usuario y password para la máquina

Para comenzar, vamos a crear dos máquinas en las que generaremos una SSH compartida.

Instalaremos un servidor Apache y comprobaremos cómo conectar.

Utilizamos una herramienta llamada **Putty**

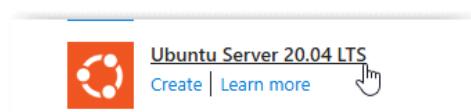
<https://putty.org/>

Comenzamos creando un grupo de recursos llamado **rg-sistemas**

Vamos a realizar el siguiente laboratorio:

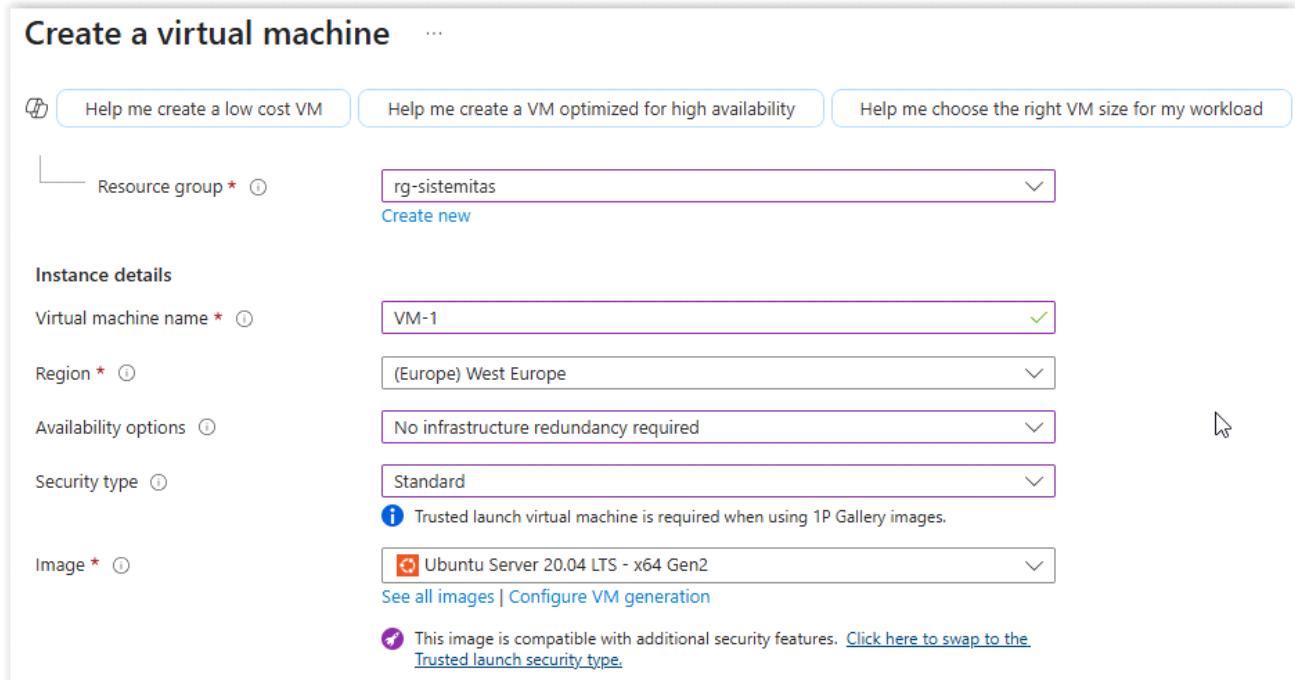
Crearemos una máquina de Ubuntu llamada **VM-1** con User y Password

Posteriormente, generaremos unas Keys para dicha máquina y nos conectaremos con esas Keys a otra máquina llamada **VM-2**



User: **usuario**

Password: **Tajamar12345**



**Create a virtual machine**

Help me create a low cost VM | Help me create a VM optimized for high availability | Help me choose the right VM size for my workload

Resource group \* rg-sistemas

Virtual machine name \* VM-1

Region \* (Europe) West Europe

Availability options No infrastructure redundancy required

Security type Standard

Image \* Ubuntu Server 20.04 LTS - x64 Gen2

This image is compatible with additional security features. [Click here to swap to the Trusted launch security type.](#)

VM architecture  Arm64  x64

Run with Azure Spot discount

Size \*  Standard\_B1s - 1 vcpu, 1 GiB memory (8.76 US\$/month) (free services eligible)  See all sizes

Enable Hibernation   
i Hibernate is not supported by the size that you have selected. Choose a size that is compatible with Hibernate to enable this feature. [Learn more](#)

Authentication type  SSH public key  Password

Username \*  ✓

Password \*  ✓

Confirm password \*  ✓

**Inbound port rules**

Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports \*  None  Allow selected ports

Select inbound ports \*  ✓

## Create a virtual machine

Help me create a low cost VM Help me create a VM optimized for high availability Help me choose the right VM size for my workload

**Basics** **Disks** **Networking** **Management** **Monitoring** **Advanced** **Tags** **Review + Create**

Configure monitoring options for your VM.

**Alerts**

Enable recommended alert rules

**Diagnostics**

Boot diagnostics  Enable with managed storage account (recommended)  Enable with custom storage account  Disable

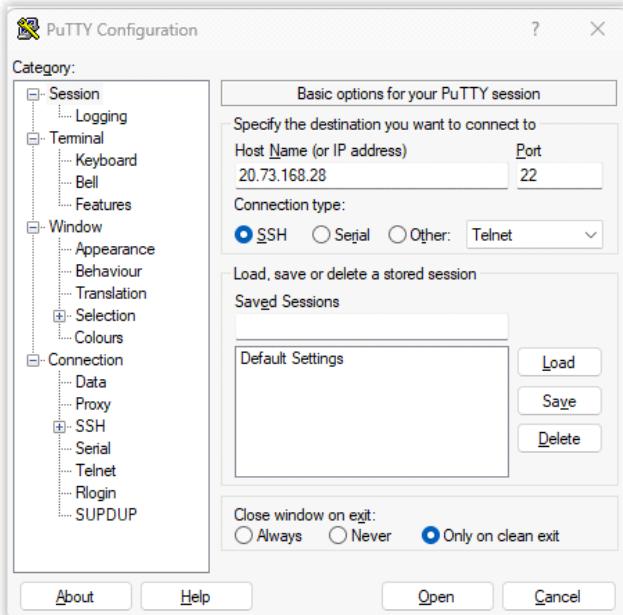
Enable OS guest diagnostics

**Health**

Al crear la máquina, tendremos una dirección IP pública por defecto por la que podremos conectar

Operating system	:	Linux (ubuntu 20.04)
Size	:	Standard B1s (1 vcpu, 1 GiB memory)
Public IP address	:	<a href="#">20.73.168.28</a>
Virtual network/subnet	:	<a href="#">VM-1-vnet/default</a>
DNS name	:	<a href="#">Not configured</a>
Health state	:	-

Abrimos Putty



Y estaremos conectados a nuestra máquina

```
usuario@VM-1: ~
Usage of /: 5.3% of 28.89GB Users logged in: 0
Memory usage: 30%           IPv4 address for eth0: 10.1.0.4
Swap usage: 0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

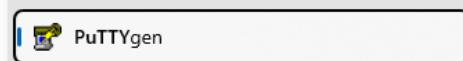
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

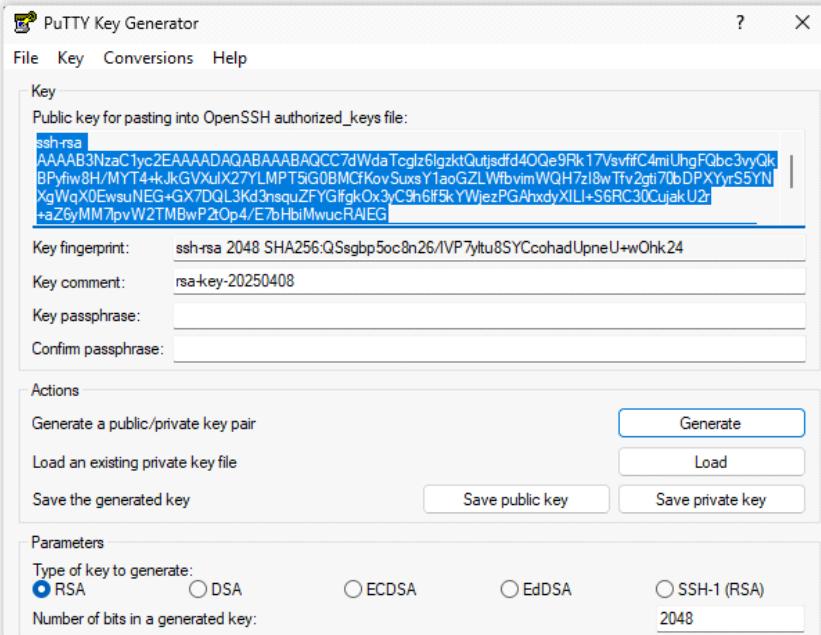
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

usuario@VM-1:~$
```

El siguiente paso es generar unas claves SSH desde la máquina VM-1 para poder conectar con las mismas claves para el resto de máquinas.

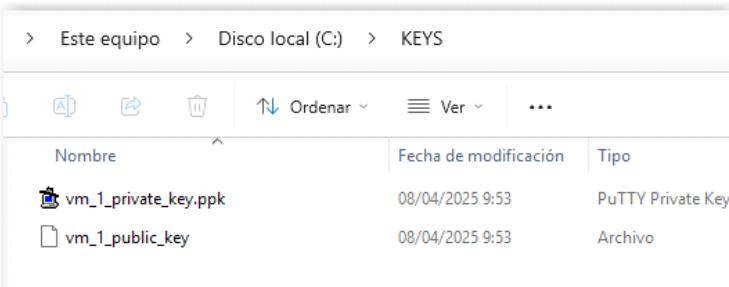
Abrimos PuttiGen y pulsamos en Generate





Nos ha generado dos Keys, clave privada y pública.

Vamos a crear una carpeta en C:\ llamada **KEYS** y almacenamos las claves



Nos interesa **Public Key** para generar las claves para las siguientes máquinas virtuales

```
----- BEGIN SSH2 PUBLIC KEY -----
Comment: "rsa-key-20250408"
AAAAB3NzaC1yc2EAAAQABAAQCC7dWdaTcg1z6IgzkQtjsfd4OQe9Rk17VsvfifC4miUhgFQbc3vyQkBPYfiw8H/MYT4+kJkGVXu1X27YLMPT5iG0BMCfSuxsY1aoGZLWfbvimWQH7zI8wTfv2gti70bDPXYyrS5YNXgWqX0EwsuNEG+GXL3Kd3nsquZFYGIfgkOx3yC9h6If5kYWjezPGAhxdyXILI+S6RC30CujakU2r+yMM7lpvW2TMBwP2tOp4/E7bHbiMwucRALEG+3F6+9tAxYW6UMqKbaR1rGBdeIukvKvePQ1RmI55qxWWsYXYrDovgu3dLP1IdJGJyRSDH7gnBsfpWT
----- END SSH2 PUBLIC KEY -----
```

La clave privada será para conectarnos desde cualquier máquina

Vamos a crear otra máquina virtual desde Azure llamada **VM-2**

**Instance details**

Virtual machine name *	VM-2
Region *	(Europe) West Europe
Availability options	No infrastructure redundancy required
Security type	Standard
<small> ⓘ Trusted launch virtual machine is required when using 1P Gallery images.</small>	
Image *	Ubuntu Server 20.04 LTS - x64 Gen2
<a href="#">See all images</a>   <a href="#">Configure VM generation</a>	
<small> ⓘ This image is compatible with additional security features. <a href="#">Click here to swap to the Trusted launch security type.</a></small>	

VM architecture

<input type="radio"/> Arm64
<input checked="" type="radio"/> x64

Run with Azure Spot discount

Size \*

Standard\_B1s - 1 vcpu, 1 GiB memory (8.76 US\$/month) (free services eligible)

[See all sizes](#)

Enable Hibernation

ⓘ Hibernate is not supported by the size that you have selected. Choose a size that is compatible with Hibernate to enable this feature. [Learn more](#)

Authentication type

<input checked="" type="radio"/> SSH public key
<input type="radio"/> Password

ⓘ Azure now automatically generates an SSH key pair for you and allows you to store it for future use. It is a fast, simple, and secure way to connect to your virtual machine.

Username \*

usuario

SSH public key source

Use existing public key

ⓘ Ed25519 and RSA SSH formats are supported for the selected VM image. Ed25519 provides a fixed security level of no more than 128 bits for 256-bit key, while RSA could offer better security with keys longer than 3072 bits.

SSH public key \*

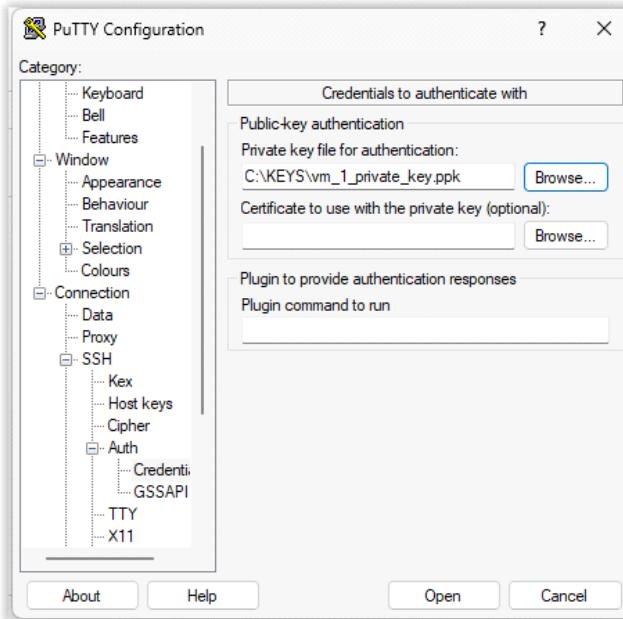
```
deIQFv
ukvKvepQ1Rml55qxWWsYXYrDovgu3dLP1IdJGJyRSDH7gnBsfzWT
---- END SSH2 PUBLIC KEY ----
```

ⓘ Learn more about creating and using SSH keys in Azure

Si nos equivocamos al copiar la clave o necesitamos generar otra, tenemos una Opción dentro de las máquinas llamada **Reset Password**

Una vez creada la máquina, para acceder a ella debemos hacerlo con **private key**

Para comprobarlo, abrimos **Putty**, escribimos la dirección IP de la nueva máquina y En **Connection, SSH, Auth Credentials** incluimos nuestra clave privada PPK



## INSTALAR UN SERVIDOR WEB APACHE EN LINUX

Para poder desplegar una aplicación Net Core en nuestra máquina necesitamos un Servidor Web que se encargue de administrar las peticiones.

Actualizamos la máquina

**sudo apt-get update**

Instalamos el servidor Apache

**sudo apt-get install apache2 -y**

Comprobamos que el servidor está activo y corriendo

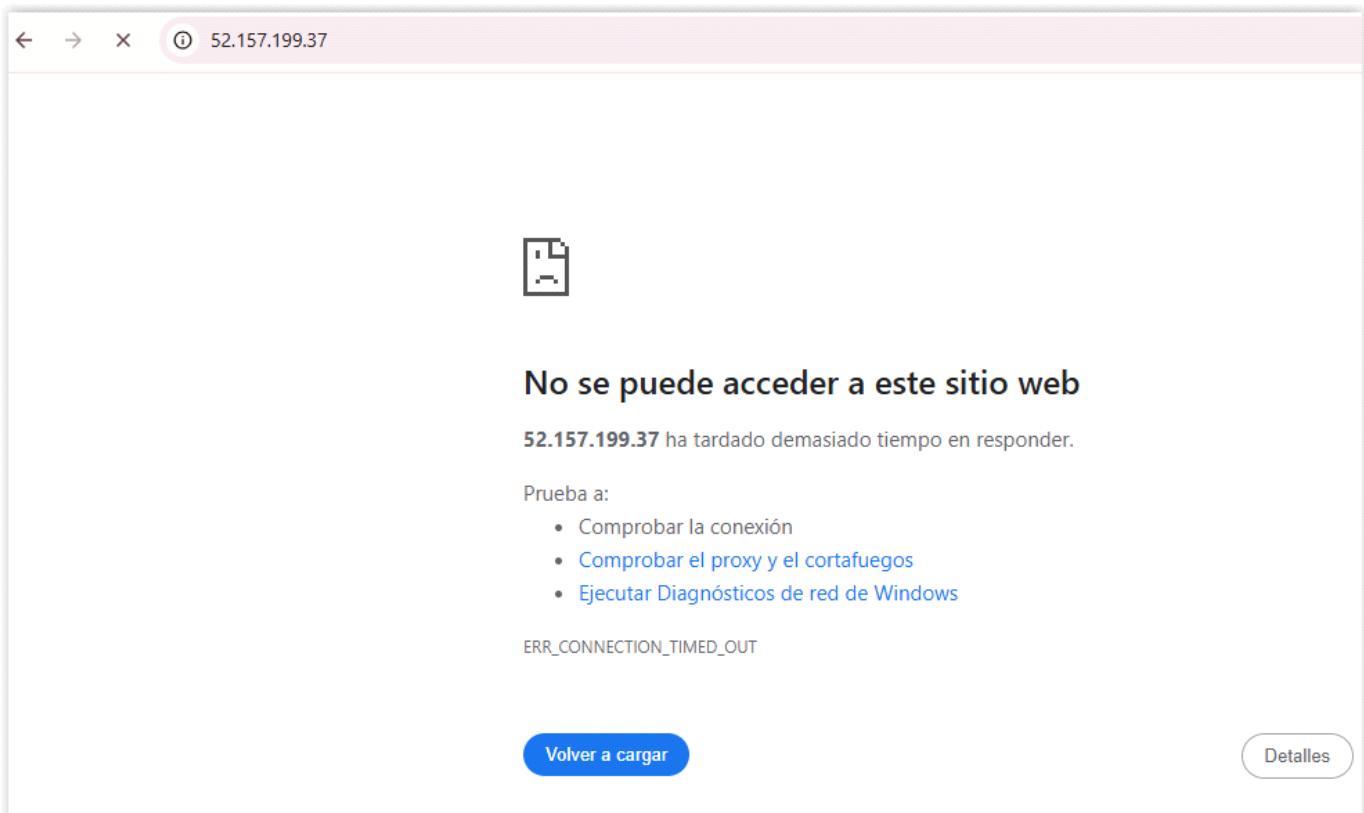
**sudo systemctl status apache2**

```
usuario@VM-2:~$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor pres>
   Active: active (running) since Tue 2025-04-08 08:21:02 UTC; 24s ago
     Docs: https://httpd.apache.org/docs/2.4/
 Main PID: 2687 (apache2)
    Tasks: 55 (limit: 1062)
   Memory: 9.3M
      CGroup: /system.slice/apache2.service
              ├─2687 /usr/sbin/apache2 -k start
              ├─2689 /usr/sbin/apache2 -k start
              └─2690 /usr/sbin/apache2 -k start

Apr 08 08:21:02 VM-2 systemd[1]: Starting The Apache HTTP Server...
Apr 08 08:21:02 VM-2 systemd[1]: Started The Apache HTTP Server.
lines 1-14/14 (END)
```

Con el servidor activo, simplemente bastaría con conectar a nuestra página para acceder

**<http://IP PUBLICA>**

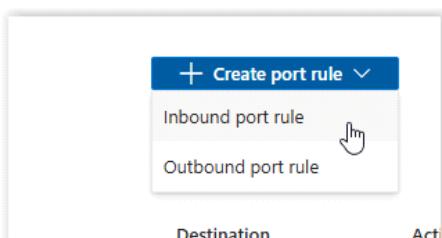


Podemos comprobar que no tenemos acceso, pero hemos montado un Web Server y  
Está activo, por lo que podríamos conectar a él sin problema.

Al crear la máquina, hemos abierto únicamente el puerto 22 que es el SSH, pero no  
Tenemos abiertas más conexiones de acceso.

Por ejemplo, si deseamos acceder por HTTP debemos abrir el puerto 80  
Si deseamos acceder por HTTPS debemos abrir el puerto 443

Entramos en **Network settings** y pulsamos sobre **Port Rule**  
Necesitamos **Inbound Rules**



**Add inbound security rule**

VM-2-nsg

Source ①  
Any

Source port ranges \* ①  
\*

Destination ①  
Any

Service ①  
HTTP

Destination port ranges ①  
80

Protocol  
 Any  
 TCP  
 UDP  
 ICMPv4

Action  
 Allow  
 Deny

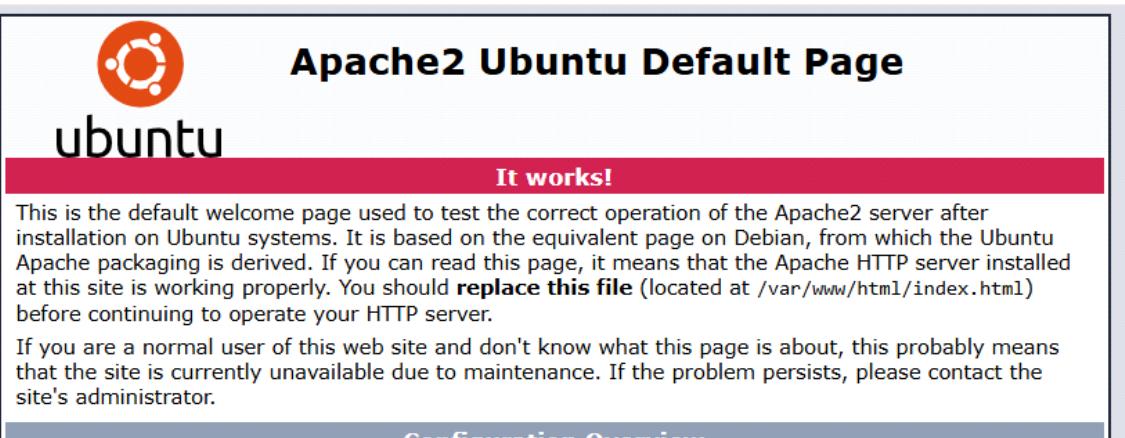
Priority \* ①  
310

Name \*  
AllowAnyHTTPInbound

Description

Y ya tendremos un servidor activo en nuestra máquina

No es seguro 52.157.199.37



This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at /var/www/html/index.html) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Eliminamos el grupo de recursos

## Delete a resource group

The following resource group and all its dependent resources will be permanently de

### Resource group to be deleted

 rg-sistemas 

### Dependent resources to be deleted (11)

All dependent resources, including hidden types, are shown

Name	Resource type
 VM-1	Virtual machine
 VM-1-ip	Public IP address
 VM-1-nsg	Network security group
 VM-1-vnet	Virtual network

Apply force delete for selected Virtual machines and Virtual machine scale sets

Enter resource group name to confirm deletion \*

rg-sistemas

**Delete**

**Cancel**

## AZURE CLOUD SHELL

martes, 8 de abril de 2025 10:31

Azure nos permite crear recursos de diferentes formas dentro de su Cloud.  
Podemos crear recursos desde el portal de Azure tal y como hemos visto.

También tenemos la posibilidad de crear recursos mediante Cloud Shell o ARM

- 1) **ARM:** Son plantillas Azure Resource Manager que podemos tenerlas en un fichero Y utilizarlas para la creación de recursos sin necesidad de utilizar el portal de Azure.  
Se pueden crear recursos desde Power Shell, C# o desde Bash

```
1 {  
2   "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",  
3   "contentVersion": "1.0.0.0",  
4   "parameters": {  
5     "subscriptionId": {  
6       "type": "String"  
7     },  
8     "resourceGroupName": {  
9       "type": "String"  
10    },  
11    "name": {  
12      "type": "String"  
13    }  
14  }  
15}
```

- 2) **Azure Portal:** si vamos a crear pocos recursos con siguiente siguiente
- 3) **Bash o PowerShell:** Mediante comandos de sistemas, podemos crear recursos

Para trabajar podemos hacerlo de dos formas:

- a. **Azure Cloud Shell:** Es una línea de comandos desde el propio portal de Azure
- b. **Azure Client:** Mediante línea de comandos con un programa llamado **az cliente** y  
Con instrucciones desde nuestro propio equipo

Debemos descargar **az client**

<https://learn.microsoft.com/es-es/cli/azure/install-azure-cli-windows?tabs=azure-cli>

Vamos a trabajar desde **CMDER**

Tenemos un montón de comandos, lo primero de todo será hacer Login con nuestra Cuenta para poder crear recursos

**az login**

Tenemos un montón de comandos, por ejemplo, si queremos averiguar comandos sobre Nuestros grupos de recursos

**az group --help**

Vamos a intentar crear un nuevo grupo

**az group create --location eastus --name rg-shell**

```
C:\Users\Profesor MCSD Mañana  
λ az group create --location eastus --name rg-shell  
{  
  "id": "/subscriptions/02ee10d4-2d35-43df-89fb-d7ac1e185646/resourceGroups/rg-shell",  
  "location": "eastus",  
  "managedBy": null,  
  "name": "rg-shell",  
  "properties": {  
    "provisioningState": "Succeeded"  
  },  
  "tags": null,  
  "type": "Microsoft.Resources/resourceGroups"  
}
```

```
C:\Users\Profesor MCSD Mañana  
λ az group delete --resource-group rg-shell  
λ az group delete --resource-group rg-shell  
λ Provisioning state: "Succeeded"
```

Vamos a ir combinando Cloud Shell con Bash.

Utilizaremos comandos desde el propio portal de Azure.

**Nota:** Necesitaremos una cuenta de almacenamiento para este Lab

```
az group create --location francecentral --name rg-gabachos
{
  "id": "/subscriptions/02ee10d4-2d35-43df-89fb-d7ac1e185646/res
```

Create a storage account

Subscription \* Azure for Students

Resource group \* rg-gabachos Create new

Instance details

Storage account name \* storageshellpaco

Region \* (Europe) France Central Deploy to an Azure Extended Zone

Primary service \* Azure Blob Storage or Azure Data Lake Storage Gen 2

Performance \* Standard: Recommended for most scenarios (general-purpose v2 account)

Redundancy \* Locally-redundant storage (LRS)

El siguiente paso es conectar con el Cloud Shell de Azure directamente

Microsoft Azure Search resources, services, and docs (G+)

Copilot Cloud Shell

Azure services

Getting started

Select a subscription to get started. You can optionally mount a storage account to persist files between sessions. [Learn more](#)

No storage account required

Mount storage account

Subscription \* Azure for Students

Use an existing private virtual network [Learn more](#)

**Apply** **Previous**

Vamos a crear una máquina virtual con una bbdd Mongo Db y un Servidor Web

Trabajamos desde Cloud Shell

```
az vm create --resource-group "rg-gabachos" --name MeanStack --image Canonical:UbuntuServer:16.04-LTS:latest --admin-username usuario --generate-ssh-keys
```

```
tech [ ~ ]$ az vm create --resource-group "rg-gabachos" --name MeanStack --image Canonical:UbuntuServer:16.04-LTS:latest --admin-username usuario --generate-ssh-keys
Selecting "northeurope" may reduce your costs. The region you've selected may cost more for the same services. You can disable this message in the future with the command "az config set core.display_region_identified=false". Learn more at https://go.microsoft.com/fwlink/?linkid=222571
SSH key files '/home/tech/.ssh/id_rsa' and '/home/tech/.ssh/id_rsa.pub' have been generated under ~/.ssh to allow SSH access to the VM. If using machines without permanent storage, back up your keys to a safe location.
Consider upgrading security for your workloads using Azure Trusted Launch VMs. To know more about Trusted Launch, please visit https://aka.ms/TrustedLaunch.
{
  "fqdns": "",
  "id": "/subscriptions/02ee10d4-2d35-43df-89fb-d7ac1e185646/resourceGroups/rg-gabachos/providers/Microsoft.Compute/virtualMachines/MeanStack",
  "location": "francecentral",
  "macAddress": "60-45-BD-1A-AD-C1",
  "powerState": "VM running",
  "privateIpAddress": "10.0.0.4",
  "publicIpAddress": "4.211.155.55",
  "resourceGroup": "rg-gabachos",
  "zones": ""
}
```

Abrimos el puerto 80 para nuestro servidor

```
az vm open-port --port 80 --name MeanStack --resource-group rg-gabachos
```

Tenemos que tener en cuenta que actualmente no tenemos acceso al portal.

Para poder conectar a esta máquina necesitamos averiguar nuestra ip pública

```
az vm show --name MeanStack --resource-group rg-gabachos --show-details --query [publicips]
```

```
tech [ ~ ]$ az vm show --name MeanStack --resource-group rg-gabachos --show-details --query [publicIps]
[
  "4.211.155.55"
]
tech [ ~ ]$ []
```

4.211.155.55

¿Cómo nos podemos conectar a nuestra máquina si tenemos en Azure los SSH?

Yo quiero conectar on premise, es decir, desde donde estoy sentado

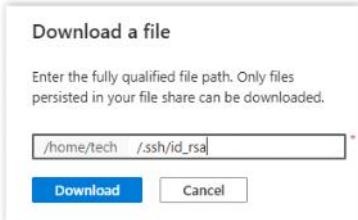
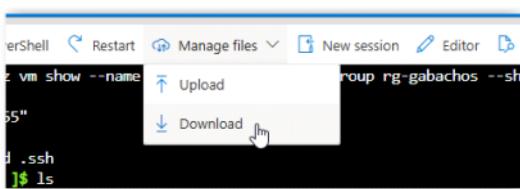
Cuando hemos creado la máquina, ha almacenado las claves dentro de /Home/User  
Y una carpeta oculta llamada .ssh

Trabajamos en Cloud Shell

```
] tech [ ~ ]$ cd .ssh
tech [ ~/ssh ]$ ls
id_rsa  id_rsa.pub
tech [ ~/ssh ]$ []
```

Necesitamos la clave privada: id\_rsa

Descargamos la clave privada en nuestro equipo.



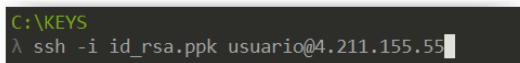
Copiamos el fichero dentro de KEYS y cambiamos su extensión por .ppk

Nos vamos a conectar a nuestra máquina MeanStack desde Windows/Bash utilizando CMDER



Para conectar, utilizamos lo siguiente:

```
ssh -i id_rsa.ppk usuario@IP
```



Nos está dando un error de permisos por nuestra Key, pero está comunicando con la máquina

```
PK.
@@@@@@@WARNING: UNPROTECTED PRIVATE KEY FILE! @@@@_
Permissions for 'id_rsa.ppk' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "id_rsa.ppk": bad permissions
usuario@4.211.155.55: Permission denied (publickey).
```

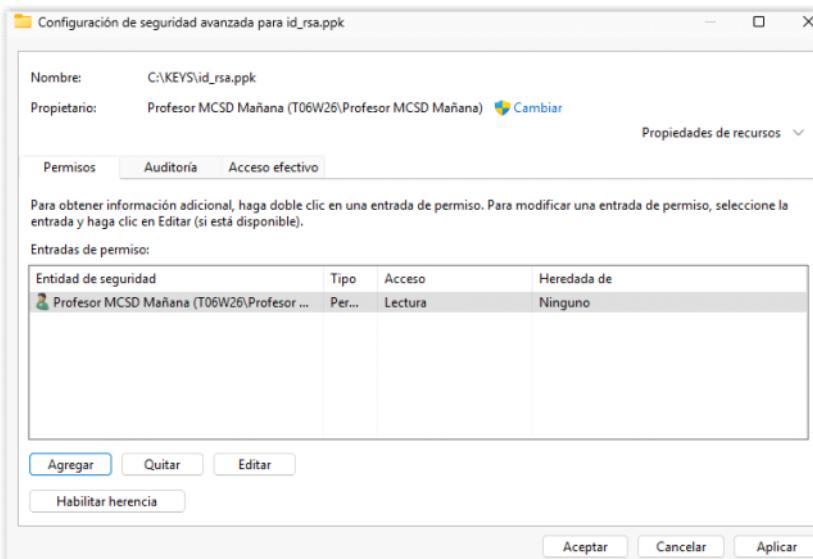
Tenemos que quitar permisos sobre el fichero **Private Key** para poder conectar

**Nota:** Necesitamos saber nuestro usuario de Windows.

Sobre PPK, Propiedades, Pestaña Seguridad y Opciones Avanzadas

Pulsamos sobre Deshabilitar Herencia

Agregamos nuestro usuario de Windows con solo Lectura



Y ya estaremos en nuestra máquina virtual

```
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
```

```
usuario@MeanStack:~$
```

Trabajamos dentro de nuestra máquina

Actualizamos el sistema

```
sudo apt update && sudo apt upgrade -y
```

Instalamos Mongo

```
sudo apt-get install mongodb -y
```

Instalamos el repositorio para descargar Node

```
curl -sL https://deb.nodesource.com/setup\_8.x | sudo -E bash
```

Instalamos Node

```
sudo apt-get install nodejs -y
```

A continuación, trabajamos con Cloud Shell

Debemos crear un sistema de carpetas y unos ficheros para nuestra app

```
cd ~
mkdir Books
touch Books/server.js
touch Books/package.json
mkdir Books/app
touch Books/app/model.js
touch Books/app/routes.js
mkdir Books/public
touch Books/public/script.js
touch Books/public/index.html
```

El siguiente paso es utilizar Visual Studio Code de Azure.

Para guardar los ficheros se utiliza control + s

```
code Books
```

Una vez copiados todos los ficheros, para salir control + q

El siguiente paso es copiar los ficheros que tenemos en Azure Cloud Shell dentro de Nuestra máquina virtual.

```
scp -r ~/Books usuario@IP:~/Books
```

```

tech [ ~ ]$ scp -r ~/Books usuario@4.211.155.55:~/Books
The authenticity of host '4.211.155.55 (4.211.155.55)' can't be established.
ED25519 key fingerprint is SHA256:7p0H+pUSTN/5dkMKMKA6sbyWTAONDCYZ6NDyYqzcc14.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '4.211.155.55' (ED25519) to the list of known hosts.
routes.js
model.js
package.json
script.js
index.html
server.js
tech [ ~ ]$ []

```

Hemos copiado los ficheros.  
A continuación es probar la conexión a nuestro Servidor de **MeanStack**.

Vamos a CMDER

Ejecutamos los siguientes comandos en este orden:

```

sudo apt install npm -y
sudo apt install express -y
sudo apt install body-parser -y
sudo apt install mongoose -y
sudo npm install -g express-generator
cd . && sudo npm install

```

Arrancamos nuestro Server

```
sudo nodejs server.js
```

Y tendremos nuestra App funcional

Name:	Fray Perico y su Borrico			
Isbn:	231455			
Author:	Juan Muñoz			
Pages:	198			
<input type="button" value="Add"/>				
<b>Name</b>	<b>Isbn</b>	<b>Author</b>	<b>Pages</b>	
<input type="button" value="Delete"/>	Fray Perico y su Borrico	231455	Juan Muñoz	198

La última acción es eliminar el grupo de recursos

```
C:\KEYS
az group delete --resource-group rg-gabachos
```

# IAAS NET CORE

martes, 8 de abril de 2025 12:54

El siguiente paso que vamos a intentar es crear/desplegar una aplicación Net Core dentro De una VM.

Lo primero será crear la máquina, instalar las librerías, algún Server y utilizar algo llamado **Reserve engineering** para apuntar nuestra App de Net Core hacia el Server de Apache

Vamos a crear un nuevo grupo de recursos llamado **rg-core**

Creamos una nueva máquina virtual llamada **vm-core** con Ubuntu 20

The screenshot shows the 'Instance details' section of the Azure VM creation wizard. The 'Virtual machine name' is set to 'vm-core'. The 'Region' is '(Europe) West Europe'. Under 'Availability options', it says 'No infrastructure redundancy required'. The 'Security type' is 'Standard'. A note indicates that a Trusted launch virtual machine is required when using 1P Gallery images. The 'Image' selected is 'Ubuntu Server 20.04 LTS - x64 Gen2'. Below this, there's a note about compatibility with additional security features and a link to swap to the Trusted launch security type.

**VM architecture:** x64

**Run with Azure Spot discount:**

**Size:** Standard\_B1s - 1 vcpu, 1 GiB memory (8.76 US\$/month) (free services eligible)

**Enable Hibernation:**  (Note: Hibernate is not supported by the size that you have selected. Choose a size that is compatible with Hibernate to enable this feature. [Learn more](#))

**Authentication type:** SSH public key (selected)

**Username:** azureuser

**SSH public key source:** Generate new key pair

**SSH Key Type:** RSA SSH Format (selected)

**Key pair name:** vm-core\_key

**Inbound port rules**

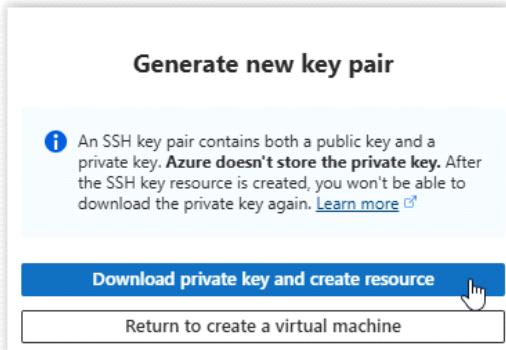
Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports \*  None  Allow selected ports

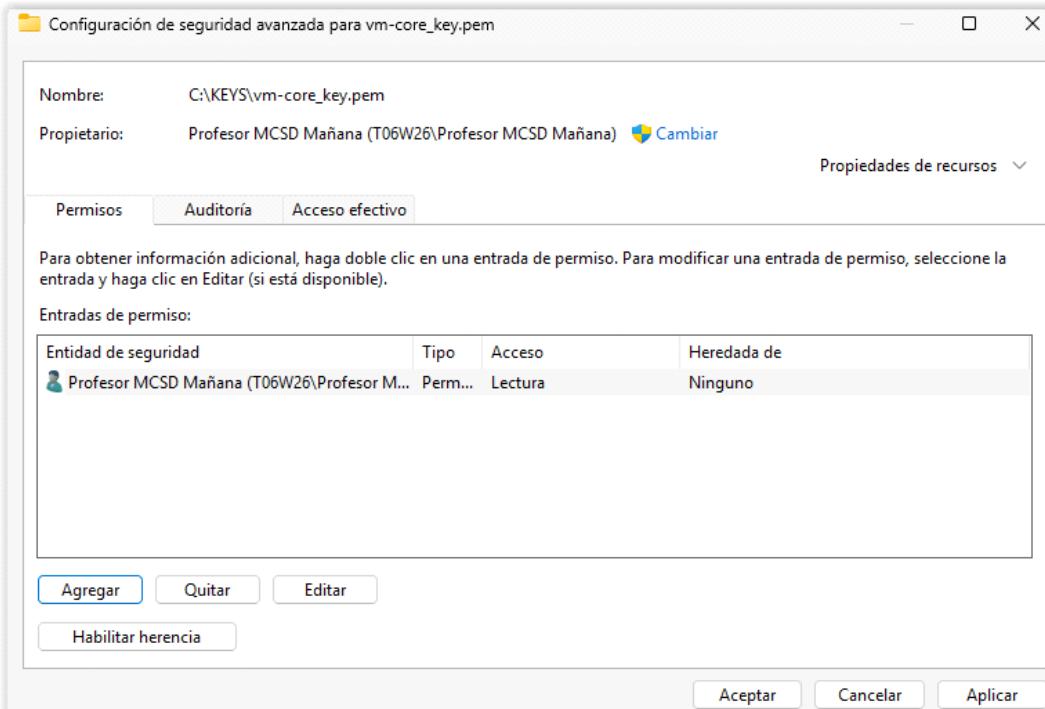
Select inbound ports \*  HTTP (80), HTTPS (443), SSH (22)

- HTTP (80)
- HTTPS (443)
- SSH (22)

Avanzamos a Create y nos dirá de descargar la clave privada



Mientras creamos la máquina, reasignamos los permisos a la clave privada



Como hemos utilizado Azure Portal, copiamos la IP y entramos dentro de CMDER a nuestra Máquina

```
ssh -i FICHERO.pem azureuser@IP
```

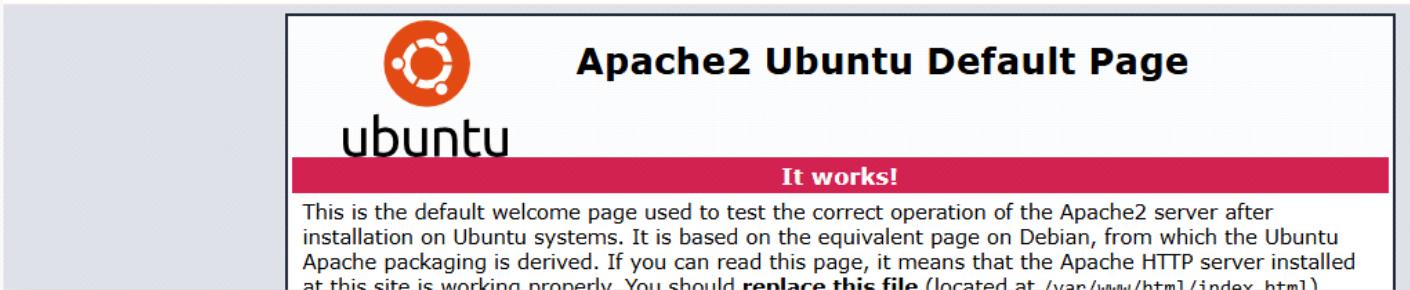
Comenzamos actualizando el sistema

```
sudo apt-get update
```

Necesitamos un Web Server

```
sudo apt-get install apache2 -y
```

Comprobamos que ya es funcional todo



Manual:

<https://learn.microsoft.com/es-es/dotnet/core/install/linux-scripted-manual#scripted-install>

#### Todos desde la máquina virtual

Descargamos los repositorios de SDK y Net Core

```
wget https://packages.microsoft.com/config/ubuntu/21.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb
```

Instalamos los paquetes

```
sudo dpkg -i packages-microsoft-prod.deb
```

Volvemos a actualizar el sistema

```
sudo apt-get update
```

Debemos instalar las librerías de Net Core y su versión actual o la que deseemos utilizar.

```
wget https://packages.microsoft.com/config/ubuntu/$(lsb_release -rs)/packages-microsoft-prod.deb -O packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
sudo apt-get update
sudo apt-get install -y apt-transport-https
sudo apt-get update
sudo apt-get install -y dotnet-sdk-9.0
```

Con **dotnet --info** comprobamos la instalación de Net Core 9.0

```
apt-get install -y apt-transport-https
azureuser@vm-core:~$ dotnet --info
.NET SDK: install -y dotnet-sdk-9.0
  Version:          9.0.202
  Commit:          3a53853c30
  Workload version: 9.0.200-manifests.21502d11
  MSBuild version:  17.13.13+1c2026462

Runtime Environment:
  OS Name:        ubuntu
  OS Version:     20.04
  OS Platform:   Linux
  RID:            linux-x64
  Base Path:      /usr/share/dotnet/sdk/9.0.202/
```

Cuando tenemos aplicaciones Net Core, dichas Apps utilizan dos puertos

HTTP: 5000  
HTTPS: 5001

Necesitamos indicar al Web Server que cuando reciba una petición por puerto 5000 nos lleve a su puerto 80

Necesitamos indicar al Web Server que cuando reciba una petición por puerto 5001 nos lleve a su puerto 443

Para ello se utiliza **Table Routing**

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

El siguiente paso es comprobar si funciona esto realmente.

```
mkdir netcore
```

```
cd netcore
```

Creamos un nuevo proyecto MVC Net Core

```
dotnet new mvc
```

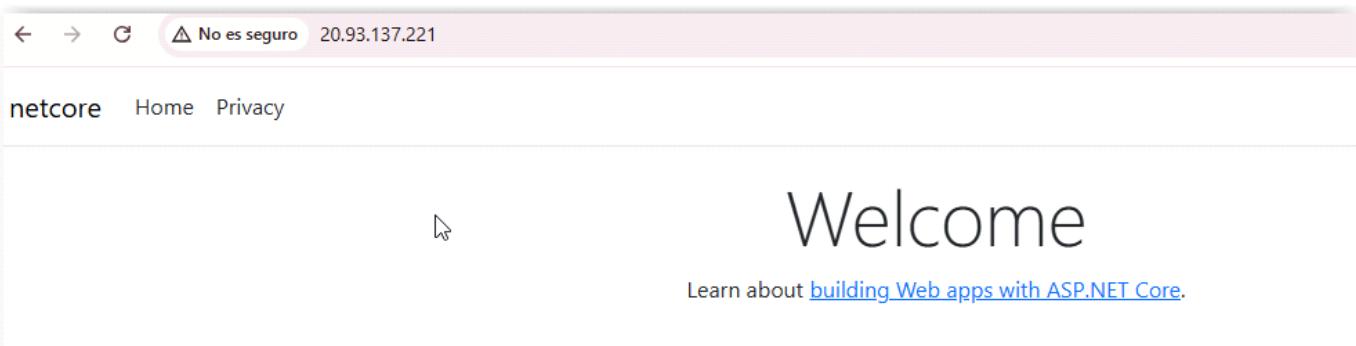
```
azureuser@vm-core:~/netcore$ ls
Controllers Program.cs Views appsettings.json obj
Models Properties appsettings.Development.json netcore.csproj wwwroot
azureuser@vm-core:~/netcore$
```

Lanzamos nuestra App para comunicar con el puerto **5000**

```
dotnet run --urls "http://0.0.0.0:5000"
```

```
azureuser@vm-core:~/netcore$ dotnet run --urls "http://0.0.0.0:5000"
Using launch settings from /home/azureuser/netcore/Properties/launchSettings.json...
Building...
warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
      No XML encryptor configured. Key {cb97987a-9b99-4b2b-aaed-a5c5e0a07181} may be persisted to storage in unencrypted form.
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://0.0.0.0:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /home/azureuser/netcore
```

Y ya tendremos nuestra App corriendo en Apache



Este ha sido un paso para crear una App vacía y comprobar que es funcional.

El siguiente paso es tener una App nuestra y desplegarla en la máquina

- 1) **FTPS:** Nos conectamos con credenciales de FTP a la máquina y copiamos los Ficheros que hemos compilado en nuestro equipo
- 2) **GitHub:** Utilizando el repositorio de GitHub y bajando los ficheros a nuestra máquina

Instalamos GitHub en nuestra máquina

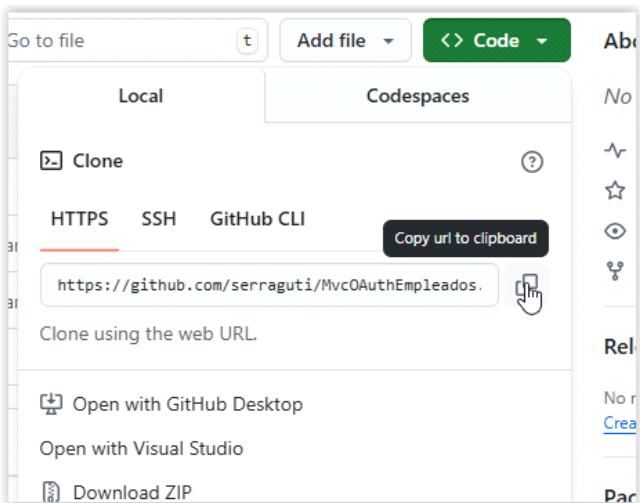
```
sudo apt-get install git -y
```

Configuramos nuestro usuario de Git y su Email.

```
git config --global user.name "USUARIO"
git config --global user.email "mail@github.com"
```

Vamos a clonar una aplicación **pública**, si es una privada, actualmente debemos hacerlo  
Generando una Key dentro de GitHub

Copiamos el GIT



Creamos una nueva carpeta para nuestra App

```
sudo mkdir /var/appcore
```

```
cd /var/appcore
```

Clonamos nuestro repositorio

```
sudo git clone URL.git
```

Como hemos puesto nuestra app dentro del directorio /var de sistema, debemos Asignar permisos a nuestro usuario **azureuser** para compilar el proyecto

```
sudo chown -R azureuser /var/appcore
```

Nos hemos descargado un proyecto sin nada. Debemos compilar el proyecto con Un Build, un restore para descargar las librerías (Nuget) y, por último, hacer un RUN

Entramos en nuestro directorio del proyecto

```
azureuser@vm-core:/var/appcore/MvcOAuthEmpleados/MvcOAuthEmpleados$ ls
Controllers  MvcOAuthEmpleados.csproj  Services  wwwroot
Filters     Program.cs               Views      appsettings.json
Models      Properties             appsettings.Development.json
azureuser@vm-core:/var/appcore/MvcOAuthEmpleados/MvcOAuthEmpleados$
```

Restauramos los Nuget

```
dotnet restore
```

Realizamos una compilación del proyecto

```
dotnet build
```

Lanzamos nuestra App

```
dotnet run --urls "http://0.0.0.0:5000"
```

El último paso es intentar abrir el puerto 443 para HTTPS

Necesitamos instalar un Certificado para HTTPS

```
sudo apt-get install apt-transport-https -y
```

Lo siguiente es direccionar el Web Server a nuestro puerto 443 con 5001

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

Probamos nuestra App con el protocolo https

```
dotnet run --urls "https://0.0.0.0:5001"
```

Y lo tenemos



```
C:\KEYS
λ az group delete -resource-group rg-core
```

## API MANAGEMENT

miércoles, 9 de abril de 2025 9:08

Debemos crear un recurso cuyo nombre es único.

Creamos un nuevo grupo de trabajo llamado rg-arsenal

Create API Management service

Subscription \* Azure for Students

Resource group \* rg-arsenal

Instance details

Region \* (Europe) West Europe

Resource name \* apimanagementpgs

Organization name \* Tajamar

Administrator email \* paco.garcia.serrano@tajamar365.com

En el SKU ponemos **Developer**

Pricing tier

The Developer tier of API Management does not include SLA and should not be used for production purposes. Your service may experience intermittent outages, for example during upgrades. [Learn more](#)

Pricing tier \* Developer (no SLA)

Como su nombre indica, es un servicio para administrar Apis.

Para entender realmente que es esto, tenemos que visualizar ahora mismo en los Recursos que tenemos disponibles.

Por ejemplo, si os digo ahora mismo quiero utilizar un CRUD de departamentos, que Es lo que tenemos que hacer? Ni nos acordamos de la URL, tendríamos que estar buscando en qué grupo de recursos Está y que URL nos da acceso.

Este servicio nos permite administrar todos los APIs publicados desde un único punto de URL. Las URLs de cada API no van a cambiar, pero las podemos unificar desde un solo lugar y Realizar más "cosas". A esto se le llama **Gateway**

Una de las ventajas es que también tendremos un Portal para probar las APIs, Administrarlas y venderlas como Productos si lo deseamos.

Nuestros productos pueden ser un conjunto de APIs o individual. Dichos productos pueden contener una **Subscripción**. Con dicha Subscripción puedo vender a múltiples clientes

Imaginamos que tenemos el API CRUD de departamentos publicado.

Se le hemos vendido a tres clientes para su uso y disfrute.

- 1) Cliente 1 y 2 pueden utilizar el API sin restricciones
- 2) Cliente 3 estará limitado a 100 peticiones al día

Y si el cliente 2 deja de pagarme?

Vamos a localizar dos APIs que tenemos:

- 1) ApiCrudDepartamentos
- 2) ApiEmpleadosRoutes

Lo que vamos a realizar es poner estas APIs como **Products** dentro de API Management

Crearemos una suscripción de Pago y otra Free

Mediante Tokens dejaremos acceso al cliente que deseemos

Para publicar nuestro API dentro de API Management tenemos dos formas.

Desde el propio Portal o mediante Visual Studio



- 1) Developer Portal: Es el portal donde estarán publicadas nuestras APIs y donde los clientes podrán visualizar nuestros Productos
- 2) Gateway URL: Es la URL de acceso a nuestros API para el cliente

Vamos a crear uno desde el Portal para Empleados

**Define a new API**

- HTTP**  
Manually define an HTTP API
- WebSocket**  
Streaming, full-duplex communication with a WebSocket server
- GraphQL**  
Access the full data from a single endpoint

**Create from definition**

### Create an HTTP API

**Basic** Full

* Display name	Empleados
* Name	empleados
Web service URL	e.g. http://httpbin.org
API URL suffix	empleados
Base URL	<a href="https://apimanagementpgs.azure-api.net/empleados">https://apimanagementpgs.azure-api.net/empleados</a>

**Create** **Cancel**

Otro para departamentos

### Create an HTTP API

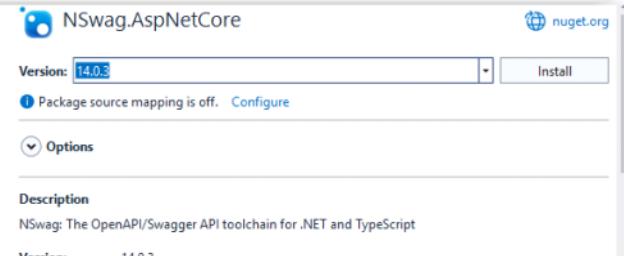
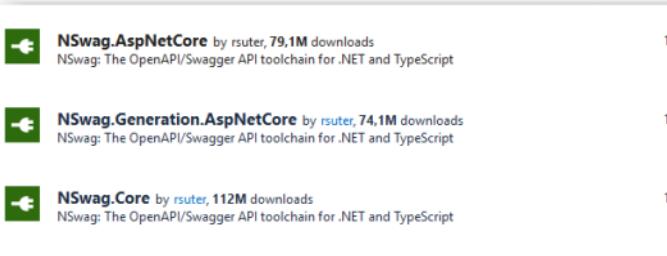
**Basic** Full

* Display name	departamentos
* Name	departamentos
Web service URL	e.g. http://httpbin.org
API URL suffix	departamentos
Base URL	<a href="https://apimanagementpgs.azure-api.net/departamentos">https://apimanagementpgs.azure-api.net/departamentos</a>

**Create** **Cancel**

Localizamos el proyecto en VS para ApiCrudDepartamentos Azure

Debemos agregar los siguientes Nuget



Muchas gracias Aaron!!!

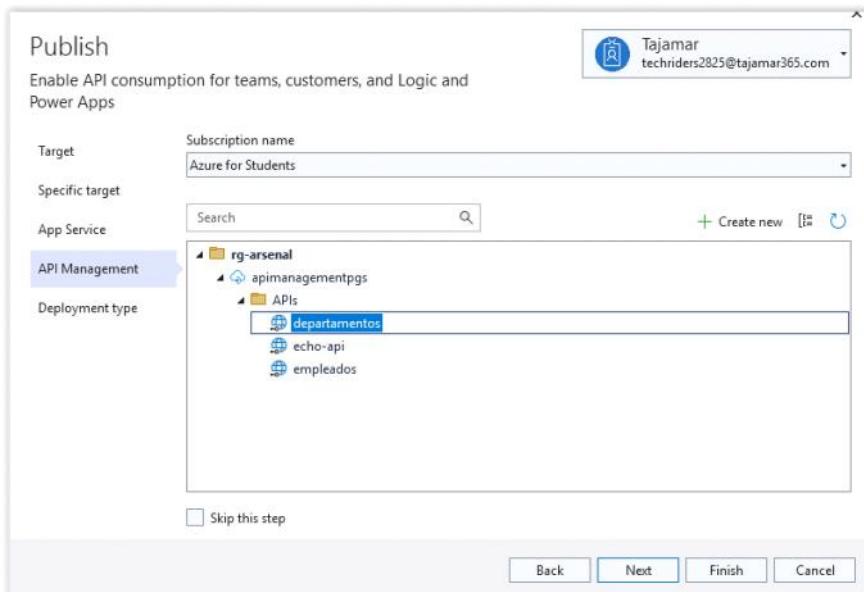
```
using ApiCoreCrudDepartamentos.Data;
using ApiCoreCrudDepartamentos.Repositories;
using Microsoft.EntityFrameworkCore;
using NSwag.AspNetCore;
using NSwag;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
string connectionString =
    builder.Configuration.GetConnectionString("SqlAzure");
builder.Services.AddTransient<RepositoryDepartamentos>();
builder.Services.AddDbContext<DepartamentosContext>(
    options => options.UseSqlServer(connectionString));
builder.Services.AddControllers();
// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi
//builder.Services.AddOpenApi();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerDocument(config =>
{
    config.DocumentName = "v1";
    config.Title = "Api Crud departamentos";
    config.Version = "v1";
});
var app = builder.Build();

// Configure the HTTP request pipeline.
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment() || app.Environment.IsProduction()) // Both environments need Swagger
{
    app.UseOpenApi(); // Serves the OpenAPI document
    app.UseSwaggerUi(); // Optional: Serves Swagger UI
}
app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

Lo vamos a publicar utilizando nuestro Visual Studio y el nuevo Endpoint que acabamos de crear en Api Management



**Nota:** NO FUNCIONA ACTUALMENTE YA CON SWAGGER, HAN RETIRADO LA DOCUMENTACION

Lo que necesitamos son los Endpoint que tenemos publicados.

Vamos a trabajar con ApiCrudDepartamentos y los empleados con Routes y Almacenaremos los datos dentro de nuestro Api Management mediante OpenApi.

## Create from definition



**OpenAPI**  
Standard, language-agnostic interface to REST APIs



**WADL**  
Standard XML representation of your RESTful API



**WSDL**  
Standard XML representation of your SOAP API

## Create from OpenAPI specification

Basic  Full

\* OpenAPI specification:  or   
(maximum size 4 MB)

Include required query parameters in operation templates

\* Display name:

\* Name:

API URL suffix:

Base URL:

## Create from OpenAPI specification

Basic  Full

\* OpenAPI specification:  or   
(maximum size 4 MB)

Include required query parameters in operation templates

\* Display name:

\* Name:

API URL suffix:

Base URL:

Podemos realizar un Test sobre los métodos y comprobar si es funcional

Developer portal

**REVISION 1** CREATED Apr 10, 2025, 9:27:07 AM

Design Settings **Test** Revisions (1) Change log

Search operations  Trace  Bypass CORS proxy

Filter by tags  Group by tag

Add API

All APIs

- ApiCoreCrudDepartame...
- ApiEmpleadosMultiples...
- Echo API

ApiCoreCrudDepartamento | v1 > /api/Departamentos - GET > Console

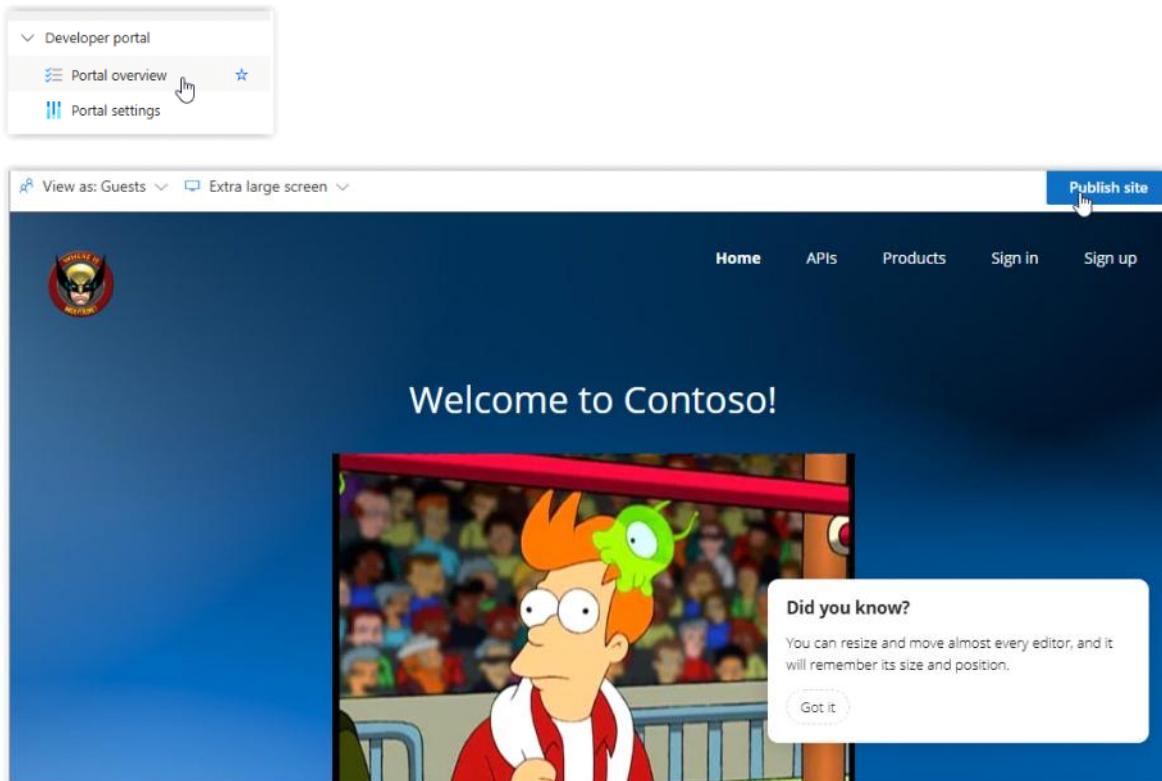
```
x-powered-by: ASP.NET

[{
    "idDepartamento": 10,
    "nombre": "CONTABILIDAD",
    "localidad": "ELCHE"
}, {
    "idDepartamento": 20,
    "nombre": "INVESTIGACION",
    "localidad": "MADRID"
}, {
    "idDepartamento": 30,
    "nombre": "VENTAS"
}]
```

Con nuestras Apis ya publicadas es el momento de Administrar el portal  
De desarrollador.

Este Portal es de estilo Power Platform, es decir, Low Code.

La primera vez debemos configurarlo y publicarlo y es el portal donde nuestros Clientes visualizarán todo.



Tened en cuenta que ahora vamos a ser dos Roles distintos

- 1) Administrador: Nosotros con nuestra cuenta de Azure en el portal de Developer
- 2) Cliente: Como clientes no tenemos acceso a nada, solamente visualizar lo que El administrador indique.

Entramos como Cliente en otro Explorador:

Developer portal URL : <https://apimangementpgs.developer.azure-api.net>

Gateway URL : <https://apimangementpgs.azure-api.net>

Tier : Developer (No SLA)

Virtual IP (VIP) addresses : public: 132.220.144.251

Platform version : stv2.1

Como clientes solamente vemos lo que el desarrollador tenga expuesto.

El administrador de Apis debe incluir los Endpoints como Productos

Nos damos de alta como cliente en la página con un correo distinto al de Administrador

Password: Alumn@12345

Email \*  
pacoserranox@gmail.com

Password \*  
.....

Confirm password \*  
.....

First name \*  
Cliente

Last name \*  
Test

Enter the characters you see.  
New | Audio

Nos llegará un correo para confirmar

Please confirm your new Tajamar API account Recibidos x

apimgmt-noreply@mail.windowsazure.com <apimgmt-noreply@mail.windowsazure.com>  
para mí ▾

Traducir al español ×

Dear Cliente Test,

Thank you for joining the Tajamar API program! We host a growing number of cool APIs and strive to provide an awesome experience for API developers.

First order of business is to activate your account and get you going. To that end, please click on the following link:

<https://apimanagementpgs.developer.azure-api.net/confirm-v2/identities/basic/signup?userid=67f7767346346119943fa557&identity=pacoserrano%40gmail.com&ticketid=67f7767346346119943fa559&ticket=7b6ebf16f2394f429d2057b94aeda634>

If clicking the link does not work, please copy-and-paste or re-type it into your browser's address bar and hit "Enter".

El siguiente paso es, como ADMINISTRADOR, creamos Productos para publicar nuestras Apis

Vamos a crear dos productos: Free y Premium

**Add product** ...

API Management service

Display name \*

 Edit Delete

Id \* Edit

 Delete

Description \*

 Edit Delete

Published

Requires subscription

Requires approval

Legal terms

ApiCoreCrudDepartamentos | v1

ApiEmpleadosMultiplesRutas | v1 Edit

Echo API

Type to search

AF  Edit Delete

+

Create

**Add product** ...

API Management service

Display name \*

 Edit Delete

Id \* Edit

 Delete

Description \*

 Edit Delete

Published

Requires subscription

Requires approval

Legal terms

- ApiCoreCrudDepartamentos | v1
- ApiEmpleadosMultipleRutas | v1
- Echo API

Type to search

Search

Back Forward History

El siguiente paso es indicar quién puede acceder a nuestros Productos

Existen tres tipos de usuario: Administrador, Invitado y desarrolladores

Entramos primero en Free

Home > apimanagementpgs | Products > Free

## Free | Access control

Product

Search Add group Columns

- Overview
- Product
- Settings
- APIs
- Policies
- Access control**
- Subscriptions

Groups

Product

Search to filter items...

Display name
<input type="checkbox"/> Administrators
<input checked="" type="checkbox"/> Developers
<input checked="" type="checkbox"/> Guests

Select

Home > apimanagementpgs | Products > Premium

## Premium | Access control

Product

Search Add group Columns

- Overview
- Product
- Settings
- APIs
- Policies
- Access control**
- Subscriptions

Groups

Product

Search to filter items...

Display name
<input type="checkbox"/> Administrators
<input checked="" type="checkbox"/> Developers
<input checked="" type="checkbox"/> Guests

Select

Con los cambios realizados, volvemos a Publicar el portal

Home > apimanagementpgs

## apimanagementpgs | Portal overview

API Management service

Search Developer portal Send us your feedback

Tags Diagnose and solve problems Resource visualizer Events Settings APIs Developer portal

Portal overview Portal settings Users Groups Identities

Overview Revisions

Developer portal is an automatically generated, fully customizable website with the documentation use them, and request access. Learn more about the developer portal or follow the tutorial to custo

Publish the portal Publishing the portal makes your changes and customizations available to visitors.

Last published Revision 20250410073604 was created on 10/4/2025, 9:36:03 and published on 10/4/2025, 9:45:50.

**Publish**

Enable CORS Cross-origin resource sharing is a mechanism that allows resources on a web page to be requested served. CORS is required to let portal visitors use the interactive console in the API reference pages custom domains, go to the domains view in the Azure portal. Learn more

Tenemos un producto Free y otro producto de Pago

Los productos de pago, hemos puesto que irán con una suscripción y, dicha suscripción Debe ir con un Token de acceso.

Dicho Token será el que nuestros clientes tendrán y que podremos revocar en cualquier Momento si lo deseamos.

Vamos a generar un nuevo Token, para ello, necesitamos una Suscripción con Cliente/Empresa imaginario.

## apimanagementpgs | Subscriptions

API Management service

Search Add subscription Columns Refresh

Tags Diagnose and solve problems Resource visualizer Events Settings APIs Workspaces APIs Products Subscriptions Named values Backends

Add subscription

API consul

Products to start using your APIs. Learn more

Display name	Primary key	Secondary key	Scope
ACME	*****	*****	Product: Starter
Built-in all-access sub...	*****	*****	Product: Unlimit
Cliente Acme	*****	*****	Service
Cliente Test (pacoserrano...	*****	*****	Product: Free
Customer	*****	*****	Product: Premiu

### New subscription

Subscription

Name \* ACME

Display name Cliente Acme

Allow tracing

Scope Product

API

Product Premium

User Cliente Test (pacoserrano@gmail.c...)

**Create**

Display name	Primary key	Secondary key	Scope	State	Owner	Allow tra
.....	.....	.....	Product: Starter	Active	Administrator	X
.....	.....	.....	Product: Unlimited	Active	Administrator	X
Built-in all-access sub...	.....	.....	Service	Active		X
.....	.....	.....	Product: Free	Active	Administrator	X
.....	Copy to clipboard	.....	Product: Premium	Active	Administrator	X
Cliente Acme	4413c129516e4...	d912e91807e64...	Product: Premium	Active	Cliente Test	X

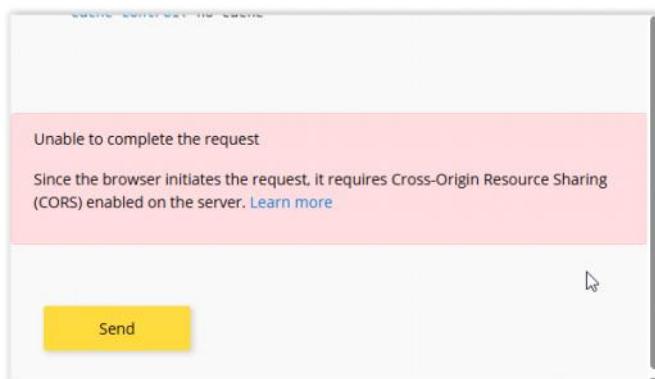
A pesar de tener dos productos separados como Premium y como Free, al generar las Apis, todas son Premium para el portal y yo debo eliminar la subscripción o no...

The screenshot shows the 'Subscription' section of the Azure API Management developer portal. It displays a list of APIs on the left and a configuration form on the right. The configuration includes fields for 'Subscription required' (checkbox checked), 'Header name' (Ocp-Apim-Subscription-Key), and 'Query parameter name' (subscription-key). At the bottom are 'Save' and 'Discard' buttons.

Ya podemos visualizar nuestras apis en el portal como CLIENTE.

Pero no podemos hacer peticiones, solamente visualizar los métodos.

Esto es normal, a lo mejor, quiero que visualicen las Apis, pero no queremos que hagan Test



Si queremos que se puedan visualizar las Apis desde el Portal, debemos habilitar CORS en Cada Api que deseemos.

En nuestro ejemplo, habilitamos CORS para todas las Apis

The screenshot shows the 'Settings' tab for an API named 'Frontend'. It displays two sections: 'Frontend' and 'Outbound processing'. The 'Frontend' section has a note about enabling CORS for the backend service. The 'Outbound processing' section has a note about modifying the response before sending it to the client. Both sections have 'Policies' buttons with '+ Add policy' options.



All APIs > Policies

[Basic](#) | Full

\* Allowed origins: \*

+ Add allowed origin

Allowed methods:

- GET
- POST
- PUT
- DELETE
- HEAD
- OPTIONS
- PATCH
- TRACE

Allowed headers: \*

+ Add allowed header

Exposed headers: + Add exposed header

Allow credentials:  No  Yes

[Save](#) [Discard](#)

En teoría, ya podríamos probar las Apis.

ApiEmpleadosMultiplesRutas | v1

Search operations

Group by tag

## ApiEmpleadosMultiples

API definition Changelog

### /api/Empleados - GET

Empleados

Request

GET <https://apimangementpgs.azure-api.net/empleados>

Response: 200 OK

#### HTTP response

HTTP/1.1 200 OK

content-type: application/json; charset=utf-8

```
[{"idEmpleado": 7369, "apellido": "SANCH", "oficio": "EMPLEADO", "salario": 104000, "departamento": 20}, {"idEmpleado": 7499, "apellido": "ARROYO", "oficio": "VENDEDOR", "salario": 208000, "departamento": 30}, {"idEmpleado": 7521,}
```

También, podríamos probar los productos Premium

ApiCoreCrudDepartamentos | v1 / /api/Departamentos - GET

GET /api/Departamentos

Authorization ↴

Subscription key Primary: Cliente Acme

Parameters ↴

+ Add parameter

**Parameters**

+ Add parameter

**Headers**

Cache-Control	no-cache
Ocp-Apim-Subscri	4413c129516e41c6890cf165631c

+ Add header

Por último, vamos a crear una aplicación simple de Consola para ver qué código necesitamos para llamar a estas APIs.

Creamos un nuevo proyecto llamado **ClienteApiManagement**

**.NET Microsoft.AspNet.WebApi.Client** by aspnet, Microsoft, 701M downloads  
This package adds support for formatting and content negotiation to System.Net.Http.

**PROGRAM**

Y podremos comprobar su funcionalidad

```
C:\Users\Profesor MCSD Mañ X + ▾
Test Api Management
[{"idDepartamento":10,"nombre":"CONTABILIDAD","localidad":"ELCHE"}, {"idDepartamento":20,"nombre":"INVESTIGACION","localidad":"MADRID"}, {"idDepartamento":30,"nombre":"VENTAS","localidad":"BARCELONA"}, {"idDepartamento":40,"nombre":"PRODUCCION","localidad":"SALAMANCA"}]
ENTER para finalizar
```

## AZURE KEY VAULT

miércoles, 9 de abril de 2025 9:16

Key Vault es un almacén de claves dentro de Azure.

Nos permite almacenar todas las Keys dentro de Azure y poder recuperarlas en nuestras Aplicaciones sin necesidad de que estén expuestas en cada App.

Podemos generar certificados SSL que son pccx para nuestros servicios HTTPS.

Nos va a permitir almacenar Keys de múltiples recursos que tengamos, por ejemplo, Cadenas de conexión, URLs de APIs, el acceso a los Storage.

A partir de ahora, con este recurso, ya no será necesario tener **appsettings.json** expuesto.

**Key Vault** funciona mediante secretos. Dichos secretos son accesibles desde una URL Única y cada secreto tiene un **ID UNICO**.

Vamos a crear el recurso desde la línea de comandos.

**az login**

Ya tenemos nuestro grupo de recursos **rg-arsenal**

The screenshot shows the Azure for Students | Resource providers interface. In the search bar, 'Keyv' is typed. A table lists the provider 'Microsoft.KeyVault' with a status of 'NotRegistered'. A tooltip over the 'Unregister' button indicates it is being held down. The left sidebar shows various navigation options like Settings, Programmatic deployment, Billing properties, etc., with 'Resource providers' currently selected.

Registrar por Command Line (Gracias Carolina!!!)

**az provider register --namespace Microsoft.KeyVault**

Mostrar los Registros (Gracias Dani)

**az provider show --namespace Microsoft.KeyVault --query "registrationState"**

El nombre del Key Vault debe ser único, cada uno con nuestras iniciales

**keyvaultpgs**

The terminal window shows the command: **az keyvault create --name keyvaultpgs --resource-group rg-arsenal --location westeurope**. The output indicates the vault is running.

Debemos crear un secreto en nuestro nuevo Key Vault

Debemos tener permisos de Administrador sobre el KeyVault

Con estas líneas de código podemos asignar el Role de Administrador

```
az role assignment create --assignee-object-id 13c26524-ebab-4b3a-99d4-fdc01a15535b --role "Key Vault Administrator" --scope /subscriptions/7037a691-01bf-4781-9f49-0de018b6a04c/resourceGroups/rg-arsenal/providers/Microsoft.KeyVault/vaults/keyvaultcpc1
```

keyvaultpgs | Access control (IAM)

Add role assignment

Number of role assignments for this subscription: 3 / 4000

Name	Type	Role	Scope	Condition
Tech Riders 2825 techriders2825...	User	Owner	Subscription (Inhe...	None
Tech Riders 2825 techriders2825...	User	Owner	Subscription (Inhe...	None

Showing 1 - 2 of 2 results.

**Role** Members Conditions Review + assign

A role definition is a collection of permissions. You can use the built-in roles or you can create your own custom roles. [Learn more](#)

**Job function roles** Privileged administrator roles

Grant access to Azure resources based on job function, such as the ability to create virtual machines.

Name	Description	Type	Category
Key Vault Administrator	Perform all data plane operations on a key vault and all objects in it, including certificates, keys, and se...	BuiltinRole	All

Deben ser secretos con Nombre UNICO

az keyvault secret set --vault-name keyvaultpgs --name secretopgs --value "Hoy es miércoles"

El secreto está protegido, pero como somos los "dueños" de la cuenta, podemos visualizarlo

az keyvault secret show --name secretopgs --vault-name keyvaultpgs

```
"recoverableDays": 90,
"recoveryLevel": "Recoverable+Purgeable",
"updated": "2025-04-09T07:45:48+00:00"
},
Deleted content type: null, con Nombre UNICO
"id": "https://keyvaultpgs.vault.azure.net/secrets/secretopgs/85027c24ac564dd0b2de3fcf507aa3b2",
"kid": null,
"managed": null,
"name": "secretopgs",
"tags": {
    "file-encoding": "utf-8"
},
"value": "Carolina Crack!!!"
}
```

El secreto está protegido, pero como somos los "dueños" de la cuenta, podemos visualizarlo

az keyvault secret show --name secretopgs --vault-name keyvaultpgs

C:\Users\Profesor

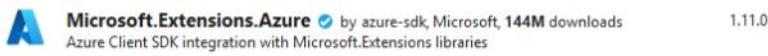
Los secretos son accesibles de dos formas diferentes:

- 1) **Azure Identity:** Tenemos que correr dentro de nuestras aplicaciones unos permisos para la cuenta que quiere utilizar Azure Key Vault
- 2) **Managed Identity:** Este tipo de validación se utiliza para el propio entorno de Azure entre los recursos y permite indicar qué recursos pueden acceder a

## Otros recursos

Vamos a crear una aplicación para leer un secreto desde nuestra App

Creamos una nueva aplicación llamada **MvcCoreKeyVault**



Vamos a trabajar sobre **Home/Index**

### INDEX.CSHTML

```
@{  
    ViewData["Title"] = "Home Page";  
}  
  
<div class="text-center">  
    <h1 class="display-4">Azure Key Vault</h1>  
    <form method="post">  
        <label>Key Vault Secret</label>  
        <input type="text" name="secretname" class="form-control"/>  
        <button class="btn btn-info">  
            Mostrar Secreto  
        </button>  
    </form>  
</div>  
<h2 style="color:blue">@ViewData["SECRETO"]</h2>
```

HOMECONTROLLER

```
public class HomeController : Controller  
{  
    //NECESITAMOS INYECTAR SecretClient  
    private SecretClient secretClient;  
  
    public HomeController(SecretClient client)  
    {  
        this.secretClient = client;  
    }  
  
    [HttpPost]  
    public async Task<IActionResult> Index(string secretname)  
    {  
        KeyVaultSecret secret =  
            await this.secretClient.GetSecretAsync(secretname);  
        ViewData["SECRETO"] = secret.Value;  
        return View();  
    }  
  
    public IActionResult Index()  
    {  
        return View();  
    }  
}
```

Para acceder a nuestro servicios necesitamos la URI de nuestro Key Vault

```
    "public IActionResult Index()  
    "contentType": null,  
    "id": "https://keyvaultpgs.vault.azure.net/secrets/secret",  
    "return View();  
    "kid": null,
```

### APPSETTINGS.JSON

```
,  
    "AllowedHosts": "*",  
    "KeyVault": {  
        "VaultUri": "https://keyvaultpgs.vault.azure.net/"  
    }
```

Activamos nuestro Key Vault dentro de **Program**

### PROGRAM

```
builder.Services.AddAzureClients(factory =>  
{  
    factory.AddSecretClient  
        (builder.Configuration.GetSection("KeyVault"));  
});
```

Tendremos un error de permisos

An unhandled exception occurred while processing the request.

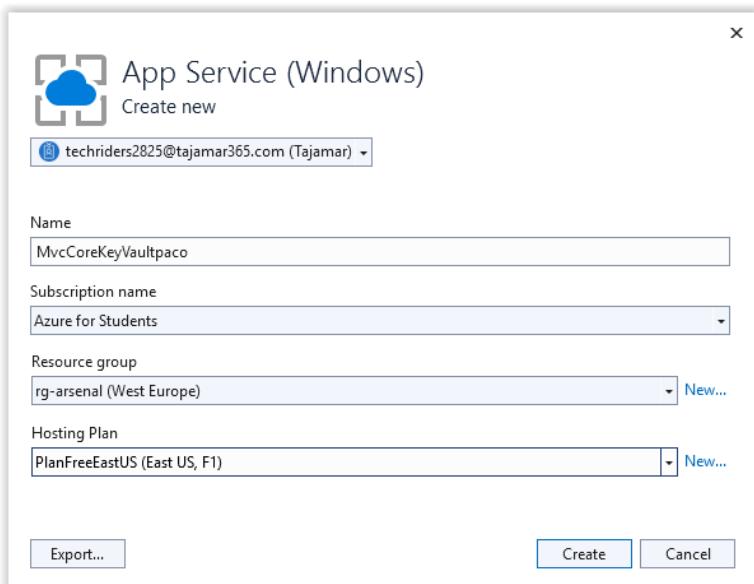
RequestFailedException: Caller is not authorized to perform action on resource.  
If role assignments, deny assignments or role definitions were changed recently, please observe propagation time.  
Caller: appid=04f0c124-f2bc-4f59-8241-bf6df9866bbd;oid=ad53dde6-75a3-4449-a3df-97d55bb94055;iss=https://sts.windows.net/435f-a38a-1a7aa77ba987/  
Action: 'Microsoft.KeyVault/vaults/secrets/getSecret/action'

A algunos nos está funcionando porque estamos utilizando el mismo explorador Web donde estamos logueados con Azure.

Nuestra aplicación está administrada para comunicarse con Azure directamente, es decir, Nuestra cuenta.

¿Qué sucede si publicamos nuestra app en un Docker, App Service o VM?

Vamos a publicar nuestro servicio en un App Service y debemos asignar permisos Administrados a nuestro App Service para que tenga acceso a Azure Key Vault.



Una vez publicado, ahora mismo NINGUNO vemos nuestro Secret.

Necesitamos utilizar **Managed Identity** e indicar que nuestra App de MVC tendrá acceso Mediante Access Policy a nuestro Azure Key Vault

Cada App que tengamos tiene un ID único dentro de Azure y dicho ID nos permite Asignar permisos para Azure Key Vault

Abrimos nuestro MVC App Service en Azure y navegamos hasta **Identity**

MvcCoreKeyVaultpaco | Identity

Web App

identi

Deployment Center

Identity

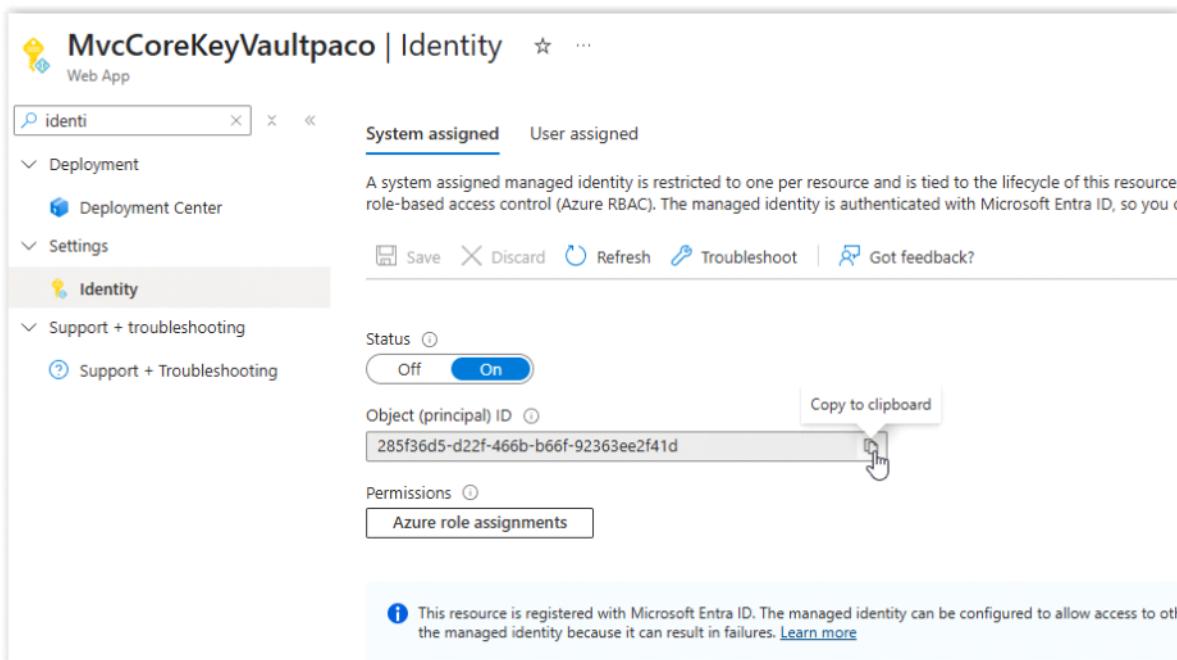
Support + troubleshooting

Off On

Object (principal) ID: 285f36d5-d22f-466b-b66f-92363ee2f41d

Azure role assignments

This resource is registered with Microsoft Entra ID. The managed identity can be configured to allow access to other resources because it can result in failures. [Learn more](#)



Copiamos el ID y navegamos a nuestro Azure Key Vault

keyvaultpgs | Access configuration

Key vault

Search

Refresh

Please click the 'Apply' button to commit your changes.

Configure your options on access policy for this key vault

To access a key vault in data plane, all callers (users or applications) must have proper authentication and authorization. Authorization determines which operations the caller can execute. [Learn more](#)

Permission model

Grant data plane access by using a [Azure RBAC](#) or [Key Vault access policy](#)

Azure role-based access control (recommended) [?](#)

Vault access policy [?](#)

[Go to access policies](#)

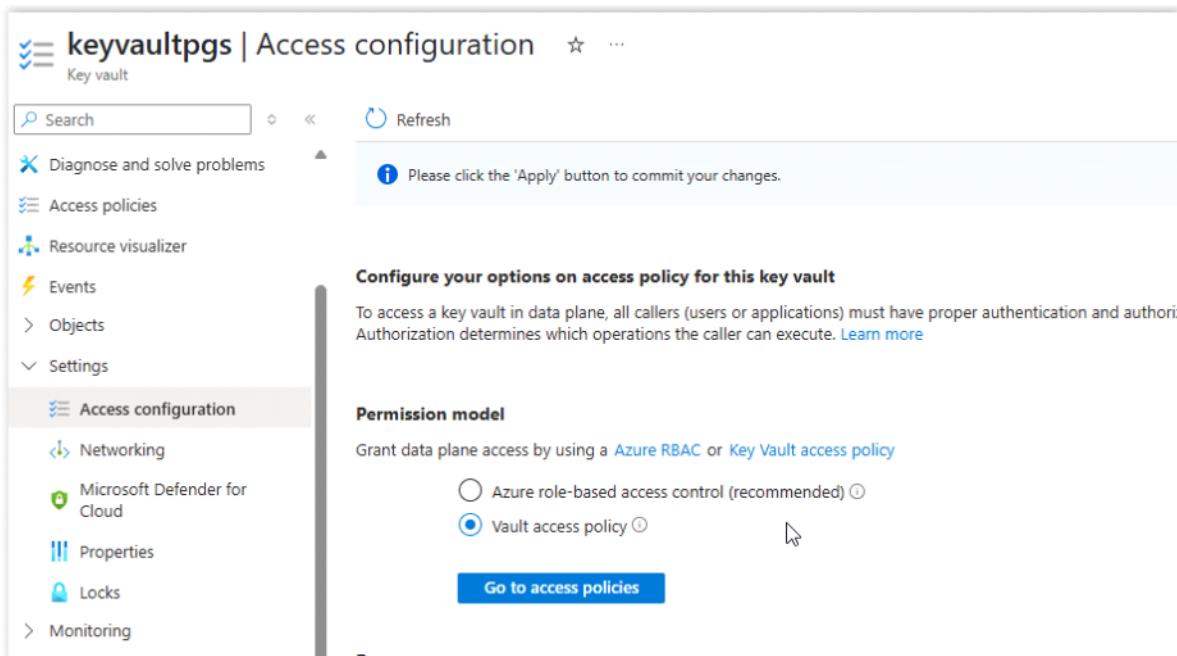
Networking

Microsoft Defender for Cloud

Properties

Locks

Monitoring



## Create an access policy ...

keyvaultpgs

**1** Permissions   **2** Principal   **3** Application (optional)   **4** Review + create

Configure from a template

Select a template

### Key permissions

- Key Management Operations
  - Select all
  - Get
  - List
  - Update
  - Create
  - Import
  - Delete

### Secret permissions

- Secret Management Operations
  - Select all
  - Get
  - List
  - Set
  - Delete
  - Recover
  - Backup

### Certificate permissions

- Certificate Management Operations
  - Select all
  - Get
  - List
  - Update
  - Create
  - Import
  - Delete

## Create an access policy ...

keyvaultpgs

**1** Permissions   **2** Principal   **3** Application (optional)   **4** Review + create

Only 1 principal can be assigned per access policy.

Use the new embedded experience to select a principal. The previous popup experience can be accessed here. [Select a principal](#)

285f36d5-d22f-466f-b66f-92363ee2f41d

MvcCoreKeyVaultpaco  
6fc67071-3396-4af2-a1a2-9dbb4974e094

### Selected item

MvcCoreKeyVaultpaco  
6fc67071-3396-4af2-a1a2-9dbb4974e094

Y ya podremos ir a nuestra App y comprobar si es funcional.

MvcCoreKeyVault Home Privacy

# Azure Key Vault

Key Vault Secret

Mostrar Secreto

Carolina Crack!!!

Por último, vamos a probar a utilizar un proyecto con ConnectionString para visualizar  
Cómo podemos recuperar dicho ConnectionString en el proyecto

```

},
"AllowedHosts": "*",
"KeyVault": {
  "VaultUri": "https://keyvaultpgs.vault.azure.net/"
},
"ConnectionStrings": {

```

Creamos un nuevo Secreto con nuestra cadena de conexión llamado **SqlAzure**

Home > keyvaultpgs | Secrets >

## Create a secret

Upload options Manual

Name \* SqlAzure

Secret value \*  (Masked)

Content type (optional)

Set activation date

Set expiration date

Enabled Yes

Tags 0 tags

**Create** **Cancel**

A **Azure.Security.KeyVault.Secrets** by azure-sdk, Microsoft, 355M downloads 4.7.0  
This is the Microsoft Azure Key Vault Secrets client library

A **Microsoft.Extensions.Azure** by azure-sdk, Microsoft, 144M downloads 1.11.0  
Azure Client SDK integration with Microsoft.Extensions libraries

## PROGRAM

```

builder.Services.AddAzureClients(factory =>
{
  factory.AddSecretClient
    (builder.Configuration.GetSection("KeyVault"));
});

//YA NO UTILIZAREMOS CONFIGURATION NUNCA MAS
//NECESITAMOS RECUPERAR EL OBJETO INYECTADO
SecretClient secretClient =
  builder.Services.BuildServiceProvider()
    .GetService<SecretClient>();
KeyVaultSecret secret =
  await secretClient.GetSecretAsync("SqlAzure");
string connectionString =
  secret.Value;

```

Publicamos nuestro proyecto en Azure y damos permisos en Access Policy de KeyVault  
Con nuestro ID de aplicación.

The screenshot shows the Azure portal interface for an API application named "ApiOAuthEmpleadosKeyVault". The user is navigating through the "Identity" settings. The "System assigned" tab is selected, showing that a managed identity is active ("On"). The "Object (principal) ID" is displayed as "d97082cc-8981-4ec1-98dd-78b9c62e34c2", with a "Copy to clipboard" button next to it. Below this, under "Permissions", there is a section for "Azure role assignments". A note at the bottom states: "This resource is registered with Microsoft Entra ID. The managed identity can be configured to allow access to other resources. Be careful when making changes; it can result in failures." There are also "Save", "Discard", "Refresh", "Troubleshoot", and "Got feedback?" buttons at the top of the page.

The screenshot shows the "Create an access policy" wizard. The current step is "Permissions". The user has selected the "Secret permissions" tab. Under "Secret permissions", the "Get" and "List" checkboxes are checked. Other options like "Set", "Delete", "Recover", "Backup", and "Restore" are available but not selected. The wizard also includes tabs for "Principal", "Application (optional)", and "Review + create".

## Create an access policy ...

keyvaultpgs

Permissions     Principal     Application (optional)     Review + create

Only 1 principal can be assigned per access policy.

Use the new embedded experience to select a principal. The previous popup experience can be accessed here. [Select a principal](#)

d97082cc-8981-4ec1-98dd-78b9c62e34c2

 ApiOAuthEmpleadosKeyVault  
a1485ebd-af8b-41e5-aba0-ea4edb469b42



### Selected item

 ApiOAuthEmpleadosKeyVault  
a1485ebd-af8b-41e5-aba0-ea4edb469b42

## DEPLOYMENT

jueves, 10 de abril de 2025 10:46

En el momento de hablar de despliegues de aplicaciones, tenemos multitud de formas:

- 1) App Service con Visual Studio Enterprise
- 2) Máquina virtual
- 3) Docker
- 4) DevOps
- 5) Implementación continua CI/CD

Estamos hablando de producción.

No siempre seremos los jefes, lo más seguro es que, a veces, no tengamos permisos para publicar nuestra aplicación directamente.

Existe un Super usuario dentro de Azure que podría desplegar todas las aplicaciones y ese User es independiente a el **Usuario de Azure**.

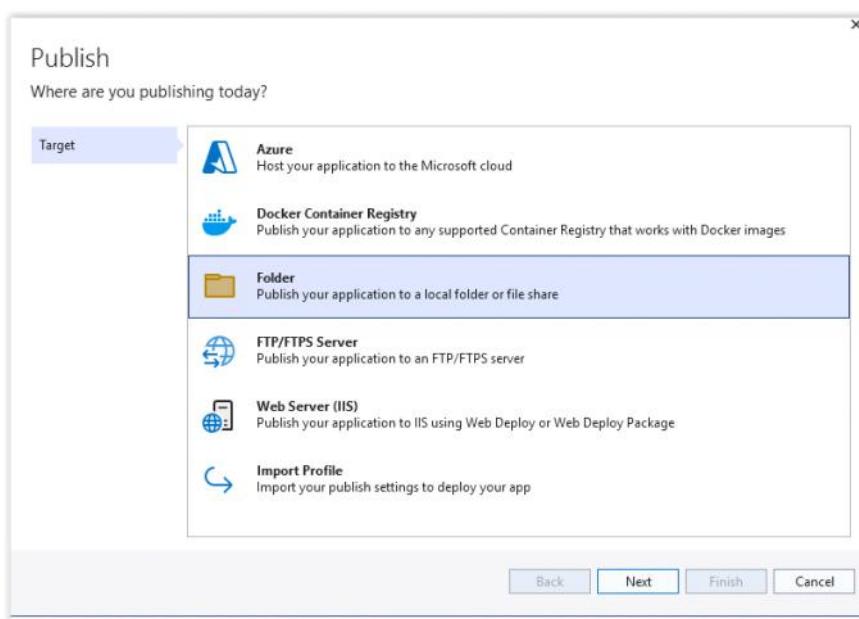
También podemos desplegar aplicaciones directamente con un App Service.

Dependiendo del App service, tenemos que configurar nuestros servicios.

Vamos a publicar una aplicación utilizando credenciales **FTP**

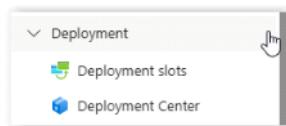
Descargamos **FileZilla**

Vamos a publicar un proyecto cualquiera MVC de forma manual.



Nombre	Fecha de modificación	Tipo	Tamaño
wwwroot	10/04/2025 11:01	Carpeta de archivos	
appsettings.Development.json	10/04/2025 10:58	Archivo JSON	1 KB
appsettings.json	10/04/2025 10:58	Archivo JSON	1 KB
MvcTestingFilezilla.deps.json	10/04/2025 11:01	Archivo JSON	1 KB
MvcTestingFilezilla.dll	10/04/2025 11:01	Extensión de la apl...	44 KB
MvcTestingFilezilla.exe	10/04/2025 11:01	Aplicación	142 KB
MvcTestingFilezilla.pdb	10/04/2025 11:01	Program Debug D...	33 KB
MvcTestingFilezilla.runtimeconfig.json	10/04/2025 11:01	Archivo JSON	1 KB
MvcTestingFilezilla.staticwebassets.endp...	10/04/2025 11:01	Archivo JSON	823 KB
web.config	10/04/2025 11:01	Archivo CONFIG	1 KB

En Azure debemos crear un nuevo App Service para alojar nuestra App



Activamos FTP en Settings

## gs | Configuration

refresh save discard leaveFeedback

appSettingsUpsellBannerMessage

platform: 32 Bit

managedPipelineVersion: integrated

scmBasicAuthPublishing...: on (radio button)

ftpBasicAuthPublishing...: on (radio button)

basicAuthPublishingCredInfoBubbleMessage: learnMore

ftpState: ftpsOnly

ftpsInfoMessage: learnMore

## Logs | Deployment Center

Save Discard Browse Manage publish profile Sync Leave Feedback

Settings Logs **FTPS credentials**

App Service supports multiple technologies to access, publish and modify the content of your app. FTPS credentials can be scoped to the application or the user.

FTPS endpoint: <https://waws-prod-blu-419.ftp.azurewebsites.windows.net/site/wwwroot>

**Application scope**

Application scope credentials are auto-generated and provide access only to this specific app or deployment slot. These credentials can be used with FTPS, Local Git and WebDeploy. They cannot be configured manually, but can be reset anytime. [Learn more](#)

FTPS Username: mvctestingfilezillapgs\\$mvctestingfilezillapgs

Password: [REDACTED]

Para poder publicar, necesitamos Detener el servidor

Estado:	Transferencia correcta, transferidos 14.093 bytes en 1 segundo
Estado:	Comenzando la subida de C:\Users\Profesor MCSD Mañana\Documents\Proyectos\MvcTestingFilezilla\MvcTestingFilezilla\bin\Release\net9.0\publish\wwwroot\lib\bootstrap\dist\css\bootstrap-reboot.css.br
Estado:	Transferencia correcta, transferidos 12.065 bytes en 1 segundo
Estado:	Comenzando la subida de C:\Users\Profesor MCSD Mañana\Documents\Proyectos\MvcTestingFilezilla\MvcTestingFilezilla\bin\Release\net9.0\publish\wwwroot\lib\bootstrap\dist\css\bootstrap-reboot.css.gz
Estado:	Transferencia correcta, transferidos 2.862 bytes en 1 segundo
Estado:	Comenzando la subida de C:\Users\Profesor MCSD Mañana\Documents\Proyectos\MvcTestingFilezilla\MvcTestingFilezilla\bin\Release\net9.0\publish\wwwroot\lib\bootstrap\dist\css\bootstrap-reboot.css.map

Sitio local: C:\Users\Profesor MCSD Mañana\Documents\Proyectos\MvcTestingFilezilla\MvcTestingFilezilla\bin\Release\net9.0\publish

Sitio remoto: /site/wwwroot

Nombre de archivo	Tamaño de...	Tipo de archivo	Última modificación
..			
wwwroot		Carpeta de archivos	10/04/2025 11:01:19
appsettings.Development...	127	Archivo JSON	10/04/2025 10:58:51
appsettings.json	151	Archivo JSON	10/04/2025 10:58:51
MvcTestingFilezilla.deps.j...	449	Archivo JSON	10/04/2025 11:01:13
MvcTestingFilezilla.dll	44.544	Extensión de la apl...	10/04/2025 11:01:12
MvcTestingFilezilla.exe	145.408	Aplicación	10/04/2025 11:01:12
MvcTestingFilezilla.pdb	33.552	Program Debug D...	10/04/2025 11:01:12
MvcTestingFilezilla.runti...	488	Archivo JSON	10/04/2025 11:01:13
MvcTestingFilezilla.static...	841.980	Archivo JSON	10/04/2025 11:01:18
web.config	563	Archivo CONFIG	10/04/2025 11:01:19

Nombre de archivo	Tamaño d...	Tipo de arc...	Última modific...	Permisos	Propietario/...
..					
wwwroot		Carpeta de...			
appsettings.Development.json	127	Archivo JS...			
appsettings.json	151	Archivo JS...			
MvcTestingFilezilla.deps.json	449	Archivo JS...			
MvcTestingFilezilla.dll	44.544	Extensión ...			
MvcTestingFilezilla.exe	145.408	Aplicación			
MvcTestingFilezilla.pdb	33.552	Program D...			
MvcTestingFilezilla.runtimeconfig.json	488	Archivo JS...			
MvcTestingFilezilla.staticwebassets.en...	841.980	Archivo JS...			
web.config	563	Archivo C...			

## LOGIC APPS

martes, 22 de abril de 2025 9:38

Este módulo nos permite poder utilizar Power Automate dentro de aplicaciones de Azure.  
Las llamadas podemos implementarlas desde el código C#.

Por ejemplo, una utilidad es poder enviar correos sin coste utilizando Logic Apps.

Vamos a crear una funcionalidad para enviar un Mail.  
Debemos enviar al Logic Apps información para su ejecución.

- Asunto, Mensaje, Email

Creamos un nuevo grupo llamado rg-martes

The screenshot shows the 'Create Logic App' wizard. In the top navigation bar, there is a 'Logic apps' icon and three dots for more options. Below the title, it says 'Select a hosting option' and provides a link to 'Learn more about Logic App hosting options'. A table compares four hosting plans: Consumption, Standard, Hybrid, and Hybrid (Preview). The 'Multi-tenant' plan under Consumption is selected. The table details the following:

	Consumption	Standard	Hybrid
Hosting plans	<b>Multi-tenant</b> Fully managed and easy to get started.	<b>Workflow Service Plan</b> Single tenant runtime with in-app connectors and scaling features.	<b>App Service Environment V3</b> Single tenant runtime with full isolation and scale out feature across App Service plans.
Compute	Shared	Dedicated	Customer managed
Networking	Public cloud	VNET Integration	Local network access
Pricing	Pay-per-operation	Per workflow service plan instance	Per vCPU hour of Kubernetes cluster

At the bottom left is a blue 'Select' button.

### Create Logic App (Multi-tenant) ...

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \*  Azure for Students

Resource Group \*  rg-martes [Create new](#)

#### Instance Details

Logic App name \*

Region \*

Enable log analytics \*  Yes  No

**⚠️** There are no log analytics workspace resources in the selected subscription. In order to enable log analytics, either create a new log analytics workspace resource or switch to a subscription which already has one.

Al crear el recurso, veremos que tenemos un editor para Logic Apps

The screenshot shows the 'logicappmailpgs' Logic App overview page. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, and Resource visualizer. The main area displays the following details:

Resource group (..)	: rg-martes	Definition
Location ( <a href="#">move</a> )	: West Europe	Status
Subscription ( <a href="#">move</a> )	: Azure for Students	Runs last
Subscription ID	: 12b04939-1235-4fcc-96b1-af57ef9a9745	Integrations
Workflow URL	: --	

Screenshot of the Azure Logic App Overview page for 'logicappmailpgs'.

**Essentials:**

- Resource group (...): rg-martes
- Location (move): West Europe
- Subscription (move): Azure for Students
- Subscription ID: 12b04939-1235-4fcc-96b1-af57ef9a9745
- Workflow URL: --
- Tags (edit): Add tags

**Run history:** Selected tab. Sub-options: Get started, Run history, Trigger history, Metrics.

**Run history filters:** Resubmit, Add filter, Specify the run identifier to open monitor view directly.

**Versions:**

Screenshot of the 'Triggers' section in the Logic App designer.

**Add a trigger:** A button with a plus icon and the text 'Add a trigger'.

**Triggers:** Triggers tell your app when to start running. Each workflow needs at least one trigger.

Screenshot of the 'Add a trigger' dialog.

**Request:** Operations to handle inbound request to workflow and send workflow response.

**When a HTTP request is received:** Selected trigger type. In-app, Trigger, ⓘ buttons.

Screenshot of the 'Enter or paste a sample JSON payload' section.

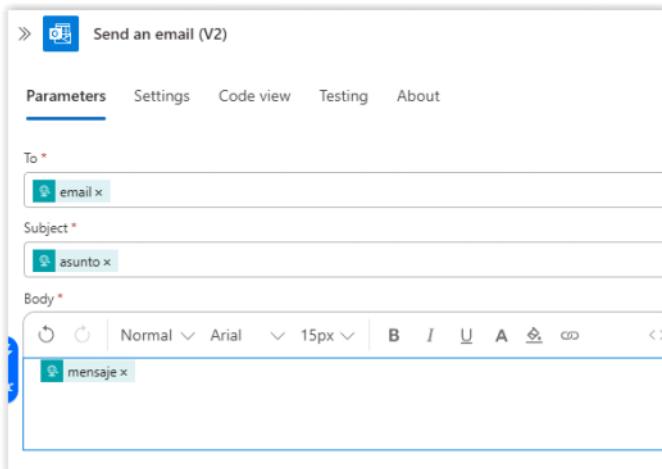
```
1 "email": "email",
2 "asunto": "asunto",
3 "mensaje": "mensaje"
```

Screenshot of the 'Office 365 Outlook' trigger configuration.

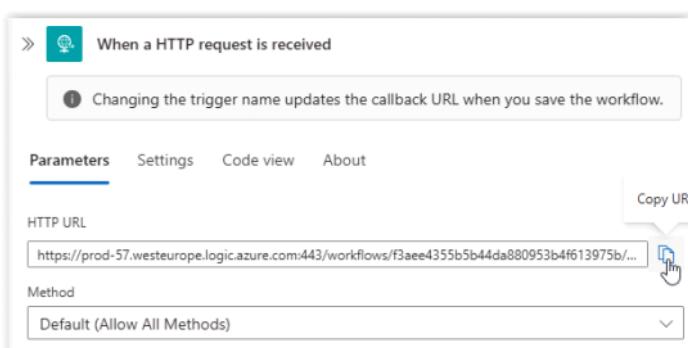
**Office 365 Outlook:** Microsoft Office 365 is a cloud-based service that is designed to help meet your organization's needs for robust security, reliability, and user productivity.

Screenshot of the 'Send an email (V2)' action configuration.

**Send an email (V2):** Action type. ⓘ button.



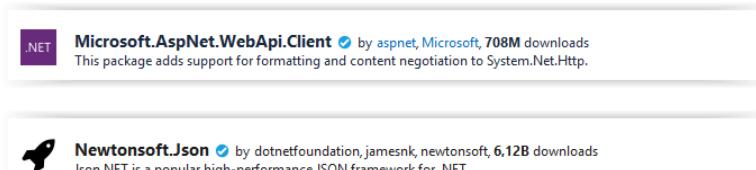
Una vez que hemos guardado, nos generará una URL para ser llamada con el Trigger



Creamos una aplicación sencilla para poder enviar un Email

Nueva aplicación de Consola para realizar la llamada. **ConsoleLogicAppsAzure**

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



#### SERVICELOGICAPPS

```
public class ServiceLogicApps
{
    private MediaTypeWithQualityHeaderValue Header;

    public ServiceLogicApps()
    {
        this.Header = new
            MediaTypeWithQualityHeaderValue("application/json");
    }

    public async Task<string> SendEmailAsync(string email, string asunto, string mensaje)
    {
        string urlLogicApps = "https://prod-57.westeurope.logic.azure.com:443/
workflows/f3ae4355b5b44da880953b4f613975b/triggers/When_a_HTTP_request_is_rec
eived/paths/invoke?api-version=2016-10-01&sp=%2Ftriggers%
2FWhen_a_HTTP_request_is_received%2FRunAsv=1_0&sig=
770G7Nrkj89MhBiyu7Rks39GyAOtpL60FF4Y2h8pk";
        var model = new
        {
            email = email,
            asunto = asunto,
            mensaje = mensaje
        };
        using (HttpClient client = new HttpClient())
        {
            client.DefaultRequestHeaders.Clear();
            client.DefaultRequestHeaders.Accept.Add(this.Header);
            string json = JsonConvert.SerializeObject(model);
            StringContent content = new StringContent
                (json, Encoding.UTF8, "application/json");
            HttpResponseMessage response = await
            client.PostAsync(urlLogicApps, content);
        }
    }
}
```

```
}
```

Realizamos las acciones dentro de **Program**

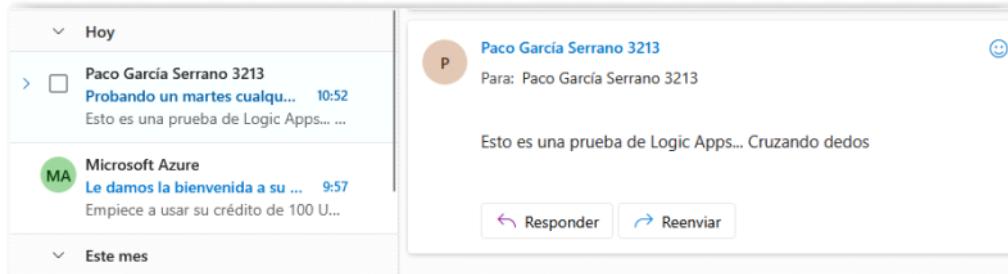
#### PROGRAM

```
using ConsoleLogicAppsAzure.Services;

Console.WriteLine("Probando Logic Apps");
Console.WriteLine("Introduzca un email");
string email = Console.ReadLine();
Console.WriteLine("Introduzca un asunto");
string asunto = Console.ReadLine();
Console.WriteLine("Escriba su mensaje");
string mensaje = Console.ReadLine();
ServiceLogicApps service = new ServiceLogicApps();
await service.SendEmailAsync(email, asunto, mensaje);
Console.WriteLine("Email enviado correctamente. Pulse ENTER para fin");
Console.ReadLine();
```

```
Probando Logic Apps
Introduzca un email
paco.garcia.serrano.3213@tajamar365.com
Introduzca un asunto
Probando un martes cualquiera
Escriba su mensaje
Esto es una prueba de Logic Apps... Cruzando dedos
```

Y podremos comprobar su funcionalidad



# AZURE FUNCTIONS

martes, 22 de abril de 2025 10:57

Las funciones de Azure son códigos **serverless**, es decir, no dependen de un servidor. Son códigos alojados en una máquina compartida con el motor del lenguaje que tengamos Creado.

Se pueden escribir en múltiples lenguajes, C#, Java, Python

No dependen de ningún servicio, se pueden utilizar como disparadores.

Podemos realizar cualquier acción que deseemos en su código dentro de Azure.

Son **Logic Apps** pero con código nuestro.

Podemos realizar llamadas de múltiples formas:

- 1) **Trigger:** Permite llamar a una función mediante un desencadenador, por ejemplo,  
Cuando creamos un nuevo registro dentro de **Cosmos Db**.  
Cuando subimos un nuevo **Blob** podemos activar una función y que realice Verificaciones sobre dicho Blob o genere algo partir de él.

- 2) **Request:** Podemos realizar una petición mediante una acción HTTP. (url)
- 3) **Timer:** Son funciones que se ejecutan cada X tiempo.

Algunas funciones necesitan de Azure Storage para trabajar.

El código podemos visualizarlo dentro del portal, pero NO siempre, depende del Tipo de código seleccionado.

Vamos a visualizar, dentro del Portal de Azure, la creación de una nueva función.

Creamos un nuevo Azure Storage para trabajar

### Create a storage account

Subscription \*

Resource group \*  [Create new](#)

Instance details

Storage account name \*

Region \*  [Deploy to an Azure Extended Zone](#)

Primary service

Performance \*  Standard: Recommended for most scenarios (general-purpose v2 account)  
 Premium: Recommended for scenarios that require low latency.

Redundancy \*

Dentro del Portal, buscamos **Functions App**



Este recurso admite un grupo de funciones, podríamos tener múltiples funciones en su Interior.

## Create Function App

### Select a hosting option

These options determine how your app scales, resources available per instance, and pricing. [Learn more about Functions hosting options](#)

Hosting plans	Flex Consumption	Consumption	Functions Premium	App Service
	Get high scalability with compute choices, virtual networking, and pay-as-you-go billing.	Pay for compute resources when your functions are running (pay-as-you-go).	Deploy multiple function apps on the same plan with event-driven scaling.	Run web apps and function apps on the same plan with more compute choices across the instances of the plan.
Scale to zero	✓	✓	-	-
Scale behavior	Fast event-driven	Event-driven	Event-driven	Metrics based
Virtual networking	✓	-	✓	✓

Select

## Create Function App (Consumption)

Subscription \* ⓘ

Azure for Students

Resource Group \* ⓘ

rg-martes

[Create new](#)

### Instance Details

Function App name \*

azurefunctionsmartespgs

.azurewebsites.net

Try a secure unique default hostname (preview). [More about this update](#)

Operating System \*

Linux (legacy)  Windows

Runtime stack \*

Node.js

Version \*

20 LTS

Region \*

West Europe

[Review + create](#)

[< Previous](#)

[Next : Storage >](#)

## Create Function App (Consumption) ...

Basics Storage Networking Monitoring Deployment Tags Review + create

### Storage

When creating a function app, you must create or link to a general-purpose Azure Storage account that supports Blobs, Queue, and Table storage. [Learn more ↗](#)

Storage account \*

storagetajamarpgs2 (v2)

[Create new](#)

i If you don't see a storage account, it may not be supported. [Storage account requirements](#)

Add an Azure Files connection



i Azure Files is used to enable certain features but you can create an app without one if you don't want to add another connection to your storage account. [What is Azure Files used for?](#)

### Diagnostic Settings

The storage account associated with the function app stores important app data. You may wish to enable monitoring for the account. You can quickly configure basic diagnostic settings as you create the function app, or you can fully

[Review + create](#)

[< Previous](#)

[Next : Networking >](#)

Las funciones utilizan **Application Insights** para mostrar sus mensajes en el portal de Azure

**Application Insights** nos permite tener telemetría dentro de nuestras aplicaciones.

En tiempo real, podemos saber qué es lo que sucede con nuestras Apps.

## Create Function App (Consumption) ...

Basics Storage Networking **Monitoring** Deployment Tags Review + create

### Application Insights

Azure Monitor application insights is an Application Performance Management (APM) service for developers and DevOps professionals. Enable it below to automatically monitor your application. It will detect performance anomalies, and includes powerful analytics tools to help you diagnose issues and to understand what users actually do with your app. Your bill is based on amount of data used by Application Insights and your data retention settings. [Learn more ↗](#)

[App Insights pricing ↗](#)

Enable Application Insights \*

No  Yes

Application Insights \*

(New) azurefunctionsmaartespgs (West Europe)

[Create new](#)

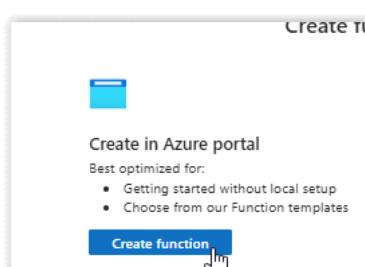
Region

West Europe

[Review + create](#)

[< Previous](#)

[Next : Deployment >](#)



HTTP trigger



A function that will be run whenever it receives an HTTP request, responding based on data in the body or query string

**1 Select a template** **2 Template details**

We need more information to create the function. [Learn more](#)

Function template	HTTP trigger
Function name *	HttpTrigger1
Authorization level *	Anonymous

## HttpTrigger1 | Code + Test

azurefunctionsmarespgs

**Code + Test** Integration Function Keys Invocations Logs Metrics

Save Discard Refresh Test/Run Get function URL Disable Delete Upload { } Re

azurefunctionsmarespgs / HttpTrigger1 / index.js

```

1 module.exports = async function (context, req) {
2   context.log('JavaScript HTTP trigger function processed a request.');
3
4   const name = (req.query.name || (req.body && req.body.name));
5   const responseMessage = name;

```

Logs App Insights Logs Log Level St

Connected! You are now viewing logs of Function runs in the current Code + Test panel. To see all the Function menu.

Tendremos una URL para poder realizar la petición a nuestro código

← → ⌂ [azurefunctionsmarespgs.azurewebsites.net/api/HttpTrigger1?name=Alumno](https://azurefunctionsmarespgs.azurewebsites.net/api/HttpTrigger1?name=Alumno)

Hello, Alumno. This HTTP triggered function executed successfully.

Hemos creado un grupo de funciones de Node.js.  
Dicho código es perfecto y podemos programar tanto en VS Code como en el editor Web.

Dependiendo del lenguaje, no podré utilizar el editor Web.

Vamos a crear un proyecto con un grupo de funciones, probarlo y publicarlo en Azure.

Creamos un nuevo proyecto de tipo **Azure Functions** llamado **grupofunctionsmartes**

 Azure Functions  
A template to create an Azure Function project.

C# Azure Cloud

Functions worker [\(i\)](#)

.NET 8.0 Isolated (Long Term Support) [\(i\)](#)

Function [\(i\)](#)

Http trigger [\(i\)](#)

Enlist in .NET Aspire orchestration (Preview) [\(i\)](#)

Use Azurite for runtime storage account (AzureWebJobsStorage) [\(i\)](#)

Enable container support [\(i\)](#)

Authorization level [\(i\)](#)

Function [\(i\)](#)

Las funciones son autónomas, es decir, no tenemos código extra, no existe Context, Ni Repos ni nada.

Tampoco Services, simplemente nuestro código irá directamente dentro de Function.

Debemos programar sin Inyección. Todo el código estará dentro de Function.

Lo que sí podemos utilizar son Nuget.

Vamos a realizar una sencilla funcionalidad en la que enviaremos un código de Empleado E incrementaremos su Salario.

 **Microsoft.Data.SqlClient** [\(i\)](#) by Microsoft, nugetsqltools, 850M downloads  
The current data provider for SQL Server and Azure SQL databases. This has replaced System.Data.SqlClient. These classes provide access to SQL and encapsulate database-specif

 **System.Data.Common** [\(i\)](#) by dotnetframework, Microsoft, 388M downloads  
Provides the base abstract classes, including System.Data.DbConnection and System.Data.DbCommand, for all data providers.

## FUNCTION

```
public class Function1
{
    private readonly ILogger<Function1> _logger;

    public Function1(ILogger<Function1> logger)
    {
        _logger = logger;
    }

    [Function("Function1")]
    public IActionResult Run
        ([HttpTrigger(AuthorizationLevel.Anonymous, "get", "post")]
        HttpRequest req)
    {
        string idempleado = req.Query["idempleado"];
        if (idempleado == null)
        {
            return new BadRequestObjectResult
                ("Debe incluir un Id de empleado");
        }
        else
        {
            string connectionString = @"Data Source=sqltajamarpgs.database.windows.net;Initial Catalog=AZURETAJAMAR;Persist Security Info=True;User ID=admin;Password=Admin123;Encrypt=True;Trust Server Certificate=True";
            SqlConnection cn = new SqlConnection(connectionString);
            SqlCommand com = new SqlCommand();
            com.Connection = cn;
            com.CommandType = System.Data.CommandType.Text;
            com.CommandText = "update EMP set SALARIO=SALARIO+1 "
                + " where EMP_NO=" + idempleado;
            cn.Open();
            com.ExecuteNonQuery();
            com.CommandText = "select * from EMP where EMP_NO="
                + idempleado;
            SqlDataReader reader = com.ExecuteReader();
            reader.Read();
            string mensaje = "El empleado "
                + reader["APELLEIDO"] + " ha incrementado su salario "
                + " a " + reader["SALARIO"];
            cn.Close();
            _logger.LogInformation("C# HTTP trigger function processed a request.");
            return new OkObjectResult(mensaje);
        }
    }
}
```

## El empleado REY ha incrementado su salario a 650003

El siguiente paso es publicar nuestra función en Azure.

Si es código C#, no tengo acceso a visualizar el código desde el Portal, solamente Desde un Editor.

¿Qué sucede si cambiamos de servidor en nuestro SQL Server con la función publicada?

- 1) Implementación continua en GitHub.
- 2) Tener un Azure Key Vault y utilizarlo dentro de la función.
- 3) Utilizar **Application Settings**. Esta funcionalidad nos permite almacenar elementos Dentro de los Settings de un Servicio.

**Nota:** Esto solamente funcionará en Producción (Azure)

Modificamos la línea de acceso a nuestra base de datos.

```
string connectionString =  
    Environment.GetEnvironmentVariable("SqlAzure");
```

### Publish

Which Azure service would you like to use to host your application?

The screenshot shows the 'Publish' dialog with the 'Target' section selected. It lists four options: 'Azure Function App (Windows)', 'Azure Function App (Linux)', 'Azure Function App Container', and 'Azure Container Registry'. The 'Azure Function App (Windows)' option is highlighted with a blue background and white text. A mouse cursor is visible at the bottom right of the dialog.

### Publish

Select existing or create a new Azure Function

The screenshot shows the 'Publish' dialog with the 'Specific target' section selected. It displays a list of existing Azure Functions. At the top right, there is a user profile icon for 'Tajamar' and the email 'paco.garcia.serrano.3213@taja...'. The list includes a folder 'rg-martes' containing a function 'azurefunctionsmartespgo (Consumption)'. A search bar and a 'Create new' button are also present. A mouse cursor is visible at the bottom right of the dialog.

Debemos administrar los Settings de nuestro servicio para Azure.

Show all settings

Hosting

Account paco.garcia.serrano.3213@tajamar365.com (Tajai)

Subscription 12b04939-1235-4fcc-96b1-af57ef9a9745

Resource group rg-martes

Resource name azurefunctionsmartespgo

Site: <https://azurefunctionsmartespgo.azurewebsites.net>

Manage Azure App Service settings

- Open in Azure portal
- Associate instance
- Attach Debugger
- View streaming logs
- Start application
- Stop application
- Download publish profile

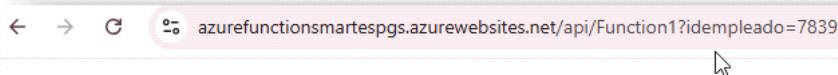
### Application settings

WEBSITE_CONTENTAZUREFILECONNECTIONSTRING	<input type="text"/>	X
Local	<input type="text"/>	X
Remote	<input type="text"/> New App Setting name	X
Def	<input type="text"/> SqlAzure	X
Insert	<input type="text"/> ey=jqgng\	X
WEBSITE_CO	<input type="text"/>	X
Local	<input type="text"/>	X
Remote	<input type="text"/> azurefunctionsmartespgo998	X
Insert value from Local		
<a href="#">+ Add setting</a>		
<a href="#">OK</a> <a href="#">Cancel</a>		

### Application settings

WEBSITE_CONTENTSHARE	<input type="text"/>	X
Local	<input type="text"/>	X
Remote	<input type="text"/> azurefunctionsmartespgo998	X
Insert value from Local		
SqlAzure	<input type="text"/>	X
Local*	<input type="text"/> Data Source=sqltajamarpgs.database.windows.net;Initial Catalog=AZURETAJAMAR;Persi:	X
Remote*	<input type="text"/> Data Source=sqltajamarpgs.database.windows.net;Initial Catalog=AZURETAJAMAR;Persi:	X
Insert value from Local		
<a href="#">+ Add setting</a>		
<a href="#">OK</a> <a href="#">Cancel</a>		

Podremos visualizar la funcionalidad en Azure



El empleado REY ha incrementado su salario a 650004

Para poder visualizar o modificar la cadena de conexión desde el Portal Azure, debemos entrar  
En **Settings** y **Environment Variables**

[x] **azurefunctionsmarespgs** | Environment variables ⭐ ...

Function App

Search ◇ < > App settings Connection strings

Events (preview)

Recommended services (preview)

Resource visualizer

> Functions

> Deployment

Settings

{x} **Environment variables**

Configuration

Authentication

Identity

Environment Variable	Action
FUNCTIONS_EXTENSION_VERSION	Show value
FUNCTIONS_WORKER_RUNTIME	Show value
SqlAzure	Show value
WEBSITE_CONTENTAZUREFILEC...	Show value
WEBSITE_CONTENTSHARE	Show value
WEBSITE_NODE_DEFAULT_VERS...	Show value
WEBSITE_RUN_FROM_PACKAGE	Show value

# IMPLEMENTACION CI/CD

lunes, 28 de abril de 2025 9:04

Todo esto que vamos a ver aquí tiene que ver con Producción y DevOps.

DevOps tiene multitud de herramientas, de hecho existe una llamada **DevOps**  
Para hacer un seguimiento continuo de Apps y Servicios en su entorno de  
Publicación hacia producción.

Cuando hablamos de publicación y seguimiento continuo estamos hablando  
De algún tipo de Repo, del estilo **Github** o del estilo **Azure Repos**

Si utilizamos Github podemos utilizar **Workflows** que son seguimientos  
De ejecuciones la publicación de nuestra App.

Un Workflow son los pasos que tiene que dar una app para ser publicada.

El **Deployment** se realiza mediante **Slots**

Un slot es una ranura que permite tener la aplicación en Producción, pero  
En distintas fases.

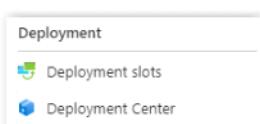
Cada Slot tendrá un espacio de producción y una Url distinta.

Podríamos tener un Slot llamado **Testing** y otro llamado **Clients** y los dos  
Son de producción, pero cada uno irá enfocado a un público.

Por ejemplo, una vez visualizado que la aplicación en el Slot **Testing** es funcional,  
Podemos indicar, que de forma automática lleve la aplicación al Slot de **Clients**.

Y al revés también, si nos da un error en Clients, podemos hacer un Rollback  
E indicar que devuelva a su versión anterior.

Esto está dentro de la pestaña **Deployments** y no podemos hacerlo con  
SKU Free (F1).



**Deployment Center** nos permitirá poder publicar una App de forma continua  
(cuando tenga cambios) mediante un Repo (GitHub, Azure Repos)

Tenemos varias formas:

- 1) **Visual Studio**: Realizar un deployment que generará un archivo YAML, que  
Es el encargado de realizar el despliegue continuo.
- 2) **Azure Portal**: Debemos indicar unos pasos, como por ejemplo, el compilador  
A utilizar y el portal generará el fichero YAML.

Vamos a realizar la publicación continua de una App, tanto en código de  
Cliente (HTML) como en código de servidor (C#).

Vamos a utilizar GitHub para ello de forma pública

Creamos una nueva App Mvc Core para su publicación continua llamada  
**MvcCoreDeployment**

Tendremos un par de imágenes dentro **wwwroot/images** y hacemos un  
Bucle o lo que sea dentro de **HomeController/Index**



```
public IActionResult Index()
{
    ViewData["SALUDO"] = "Nuestra App Deployment de lunes";
    Random random = new Random();
    List<int> numeros = new List<int>();
    for (int i = 1; i <= 10; i++)
    {
        int num = random.Next(1, 50);
        numeros.Add(num);
    }
    return View(numeros);
}
```

Modificamos el código de **Index.cshtml**

```

@model List<int>

{@
    ViewData["Title"] = "Home Page";
}



<h1 class="display-4">App Deployment Github</h1>
    <h3 style="color:red">@ViewData["SALUDO"]</h3>
    <img src "~/images/cerebros.png"
        style="width: 100px; height: 100px"/>
    <ul class="list-group">
        @foreach (int num in Model)
        {
            <li class="list-group-item">@num</li>
        }
    </ul>


```

Probamos nuestra primera versión de la App y ahora vamos a publicarla

# App Deployment Github

## Nuestra App Deployment de lunes



21

6

20

48

Creamos un nuevo grupo de recursos llamado rg-lunes

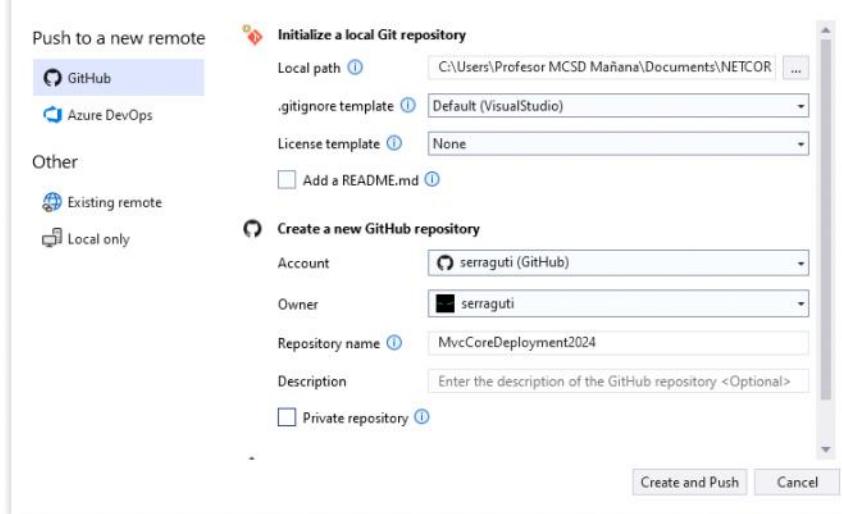
Vamos a crear en Azure Portal un nuevo App Service llamado  
MvcCoreDeploymentPaco

Create Web App ...

Subscription *	Azure for Students
Resource Group *	rg-lunes
Create new	
Instance Details	
Name *	mvccoredeploymentpaco.azurewebsites.net
Publish *	<input checked="" type="radio"/> Code <input type="radio"/> Container <input type="radio"/> Static Web App
Runtime stack *	.NET 7 (STS)
Operating System *	<input type="radio"/> Linux <input checked="" type="radio"/> Windows
Region *	West Europe
<small>Not finding your App Service Plan? Try a different region or select your App Service Environment.</small>	

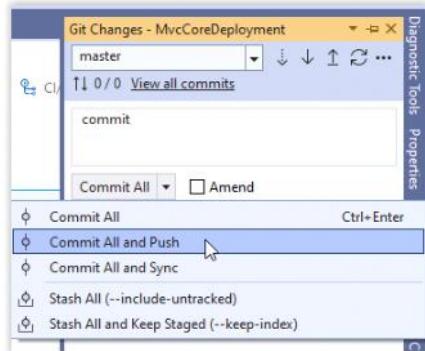
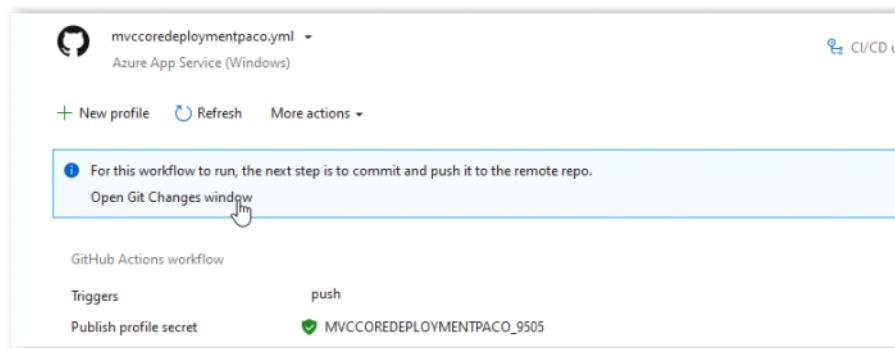
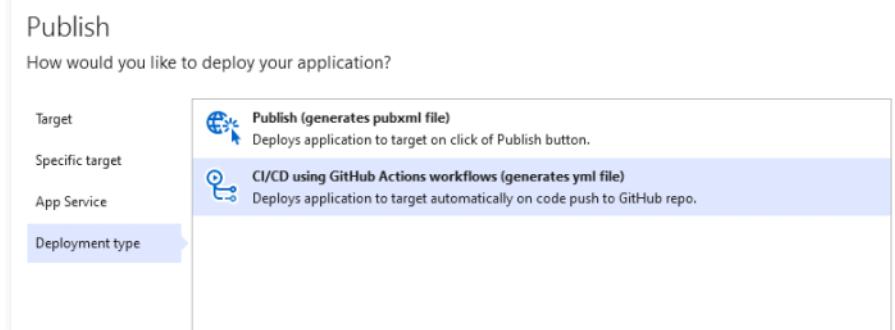
Publicamos nuestra nueva App dentro de **Github** de forma **Public**

## Create a Git repository



Ya tenemos los dos recursos necesarios.

Lo que vamos a realizar es publicar nuestra App de forma tradicional  
Dentro del App Service y cambiaremos el **Deployment Type** para generar  
Un fichero YAML para el workflow de Github



En el momento de realizar un Commit, automáticamente está  
Ejecutando el Workflow del fichero YAML

Showing 1 changed file with 48 additions and 0 deletions.

```

v 48 ████ .github/workflows/mvccoredeploymentpaco.yml □
...
@@ -0,0 +1,48 @@
 1 + name: Build and deploy .NET Core application to Web App mvccoredeploymentpaco
 2 + on:
 3 +   push:
 4 +     branches:
 5 +       - master
 6 +     env:
 7 +       AZURE_WEBAPP_NAME: mvccoredeploymentpaco
 8 +       AZURE_WEBAPP_PACKAGE_PATH: MvcCoreDeployment\published
 9 +       CONFIGURATION: Release
10 +      DOTNET_CORE_VERSION: 7.0.x
11 +      WORKING_DIRECTORY: MvcCoreDeployment
12 +    jobs:

```

Ponemos a ON la autentificación básica

Platform: 32 Bit

Managed pipeline version: Integrated

SCM Basic Auth Publish...: On

FTP Basic Auth Publish...: On

Disable basic authentication for FTP and SCM access. [Learn more](#)

Y ya veremos que publica nuestra App cada vez que realizamos un Commit.

La gran ventaja radica en que no necesitamos ningún compilador, simplemente Nuestro fichero YAML ya indica a Github el compilador que utilizará.

Cerramos Visual Studio y en Github vamos a modificar cualquier código, no importa Si es Back o Front

#### DEPLOYMENT CENTER

Dentro del Deployment Center tenemos la posibilidad de poder publicar Aplicaciones de forma manual con lenguajes con no son compatibles con Su publicación dentro del IDE (Java)

Por ejemplo, si estoy en una App de Java, debo compilar la aplicación y Posteriormente crear un fichero comprimido con todo llamado **root.jar**

Este es el fichero que tendríamos que publicar en el server.

Si estamos con un **React**, debemos utilizar un comando de **production/build** y Coger todos los elementos que ha generado y subirlos directamente a nuestro Server

Tenemos dos formas de acceder a la publicación:

- 1) **User y Pass de App Service:** Tendremos un usuario y un Password Únicos para cada App Service. Esos datos nos permitirán conectarnos Mediante FPTs a nuestro Server y publicar manualmente.

Settings   Log   **FTPS credentials**

App Service supports multiple technologies to access, publish and modify the content of your app. FTPS credentials can be scoped to the application or the user.

FTPS endpoint: https://wawi-prod-am2-475.ftp.azurewebsites.windows.net/site/wwwroot

Application scope: Application scope credentials are auto-generated and provide access only to this specific app or deployment slot. These credentials can be used with FTPS, Local Git and WebDeploy. They cannot be configured manually, but can be reset anytime. [Learn more](#)

FTPS Username: mvccftazurepaco\\$mvccftazurepaco

Password:

- 1) **User y Pass general (User scope):** Podemos crear para todo el portal de Azure un UNICO usuario que será quien se encargará de publicar. Dicho usuario Será quien tendrá que validarse en el equipo de publicación mediante **Az login**

### User scope

User scope credentials are defined by you, the user, and can be used with all the apps to which you have access. These credentials can be used with FTPS, Local Git and WebDeploy. Authenticating to an FTPS endpoint using user-level credentials requires a username in the following format: 'mvcpptazurepaco\{your username}'. Authenticating with Git requires only the username '{your username}' defined below. [Learn more](#)

Username

Password

Confirm Password

Creamos un nuevo App Service llamado **MvcFtpAzure**  
Vamos a publicar nuestra App anterior pero utilizando este nuevo método

En el servicio, cambiamos en configuración

### Platform settings

Platform

Managed pipeline version

SCM Basic Auth Publish...  On  Off

FTP Basic Auth Publish...  On  Off

Disable basic authentication for FTP and SCM access. [Learn more](#)

Borramos el perfil de publicación y volvemos a pulsar en Publish

### Publish

Where are you publishing today?

Target  Host your application to the Microsoft cloud

Publish your application to any supported Container Registry that works with Docker images

Publish your application to a local folder or file share

Publish your application to an FTP/FTPS server

Nos habrá creado los ensamblados de Production/Build

MvcCoreDeployment: Publish

FolderProfile.pubxml

New profile More actions

Publish succeeded on 29/04/2024 at 13:04.

Abrimos **Filezilla** para conectar a nuestro FTP

Debemos copiar el Build de Production a **wwwroot**

The screenshot shows two windows side-by-side. On the left, the 'Sitio local' (Local Site) window displays the directory structure of a published .NET Core application. It includes a 'bin' folder containing 'Debug' and 'Release' subfolders, and a 'Controllers' folder. Below this is a table listing files by name, size, type, and last modified date. On the right, the 'Sitio remoto' (Remote Site) window shows the contents of the '/site/wwwroot' directory, which mirrors the local structure.

Nombre de archivo	Tamaño de...	Tipo de archivo	Última modificación
..			
wwwroot		Carpetas de archivos	29/04/2024 13:04:42
appsettings.Development...	127	Archivo JSON	29/04/2024 11:33:25
appsettings.json	151	Archivo JSON	29/04/2024 11:33:25
MvcCoreDeployment.dep...	443	Archivo JSON	29/04/2024 13:04:41
MvcCoreDeployment.dll	46.080	Extensión de la apl...	29/04/2024 13:04:40
MvcCoreDeployment.exe	154.624	Aplicación	29/04/2024 13:04:40
MvcCoreDeployment.pdb	35.856	Program Debug D...	29/04/2024 13:04:40
MvcCoreDeployment.runtimeconfig.json	488	Archivo JSON	29/04/2024 13:04:41

Nombre de archivo	Tamaño d...	Tipo de arc...	Última modific...	Permisos
..		Carpeta de...	29/04/2024 13:...	
wwwroot		Archivo JS...	29/04/2024 13:...	
appsettings.Development.json	127	Archivo JS...	29/04/2024 13:...	
appsettings.json	151	Archivo JS...	29/04/2024 13:...	
MvcCoreDeployment.deps.json	443	Archivo JS...	29/04/2024 13:...	
MvcCoreDeployment.dll	46.080	Extensión ...	29/04/2024 13:...	
MvcCoreDeployment.exe	154.624	Aplicación	29/04/2024 13:...	
MvcCoreDeployment.pdb	35.856	Program D...	29/04/2024 13:...	
MvcCoreDeployment.runtimeconfig.json	488	Archivo JS...	29/04/2024 13:...	

Por último, abrimos la URL de nuestro Server y veremos que ya está publicado. Si no lo vemos, **Restart al App Service**

Por último, si necesitamos "más potencia" en nuestra App, podemos escalar Nuestra API y nuestro Mvc manteniendo todo.

The screenshot shows the Azure portal's 'App Service' blade. On the left, a sidebar lists various configuration options like Identity, Backups, Custom domains, Certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), WebJobs, and MySQL In App. The main area displays a table of App Service plans under the 'Dev/Test (For less demanding workloads)' section. The 'Free F1' plan is selected, indicated by a checked checkbox. Other plans listed include Shared D1, Basic B1, Basic B2, and Basic B3. The table includes columns for Name, ACU/vCPU, vCPU, Memory (GB), Remote Storage (GB), Cost per hour (instance), and Cost per month (instance).

Name	ACU/vCPU	vCPU	Memory (GB)	Remote Storage (GB)	Cost per hour (instance)	Cost per month (instance)
Free F1	60 minutes/day...	N/A	1	1	<b>Free</b>	<b>Free</b>
Shared D1	240 minutes/da...	N/A	1	1	0,013 USD	9,49 USD
Basic B1	100	1	1.75	10	0,075 USD	54,75 USD
Basic B2	100	2	3.5	10	0,15 USD	109,50 USD
Basic B3	100	4	7	10	0,30 USD	219,00 USD