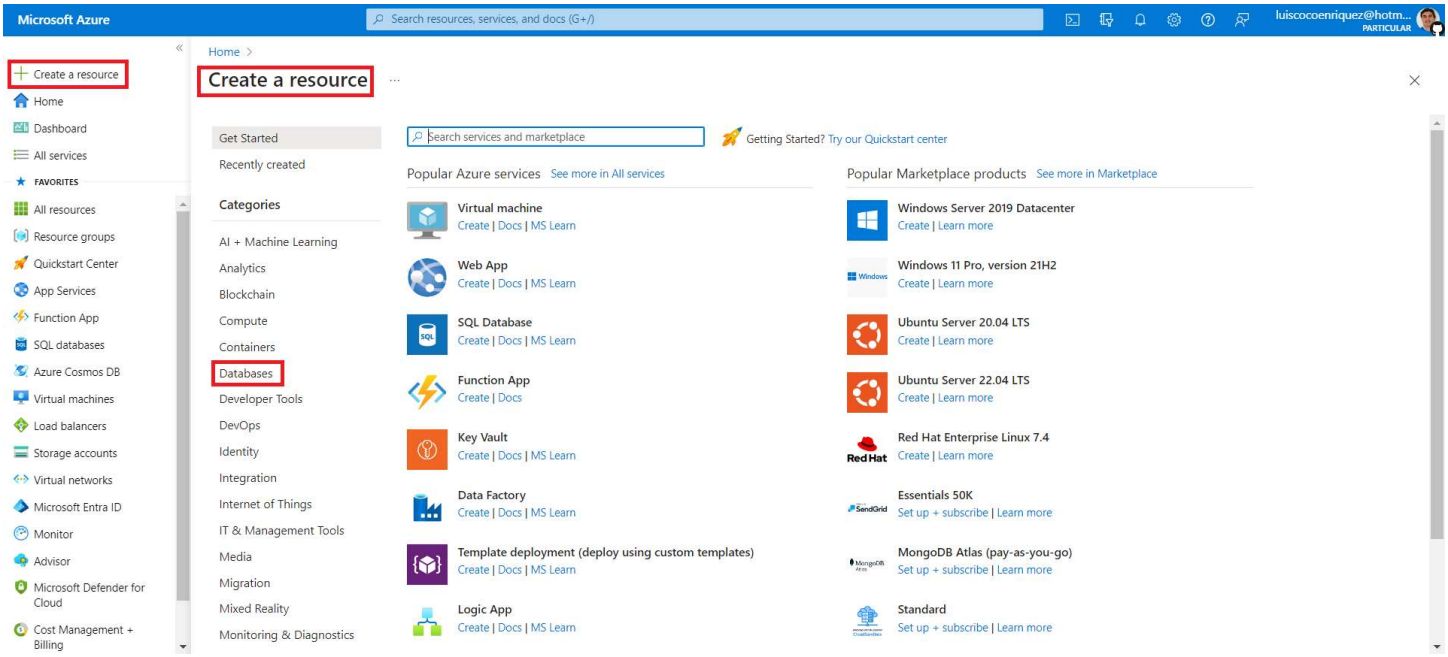
How to create .NET8 CRUD WebAPI Azure MySQL Microservice and deploy to Docker Desktop and Kubernetes (in your local laptop)

The source code is available in this github:
https://github.com/luiscoco/MicroServices_dotNET8_CRUD_WebAPI-Azure-MySQL

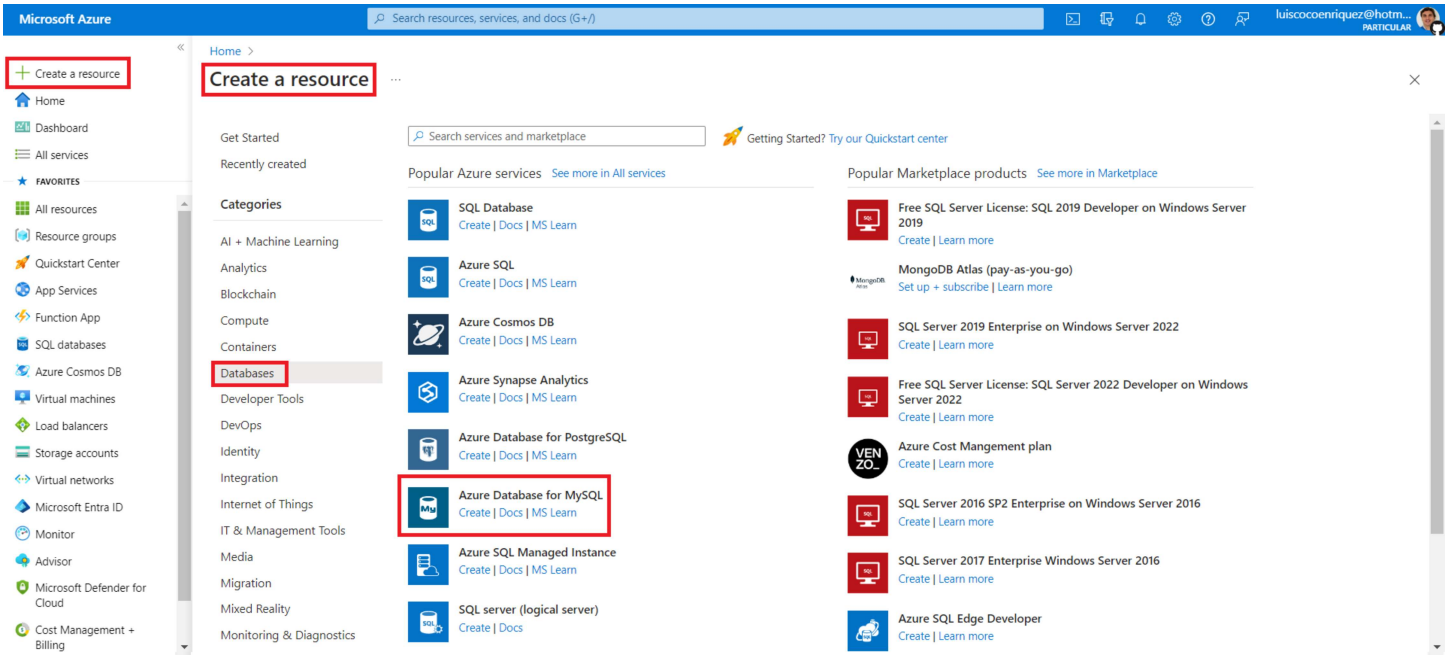
1. Prerequisite

1.1. Create Azure MySQL instance

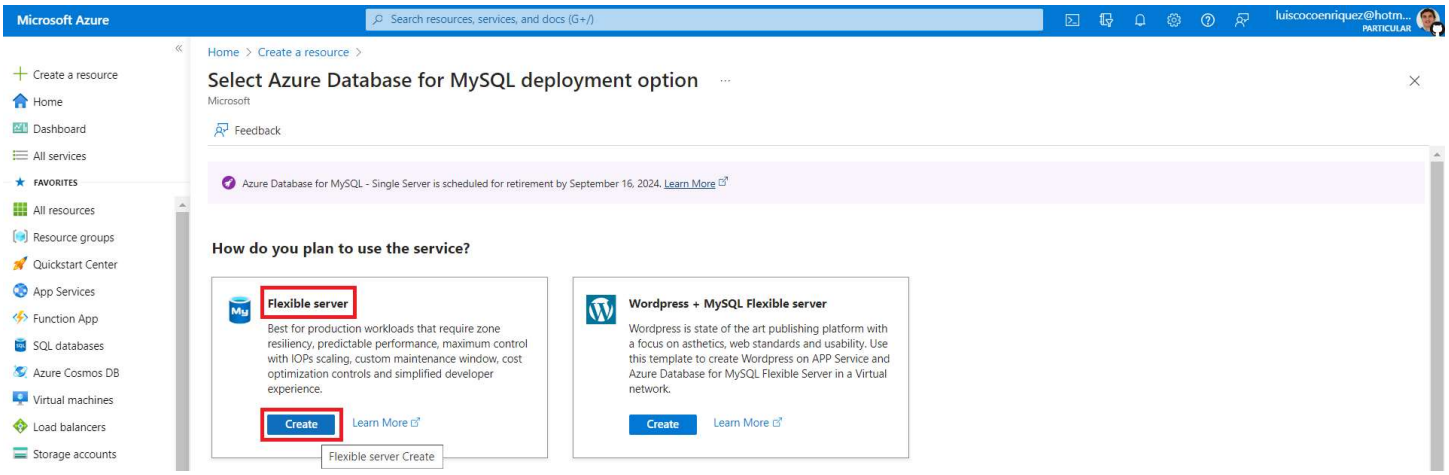
We navigate to **Create a resource** and select **Databases**



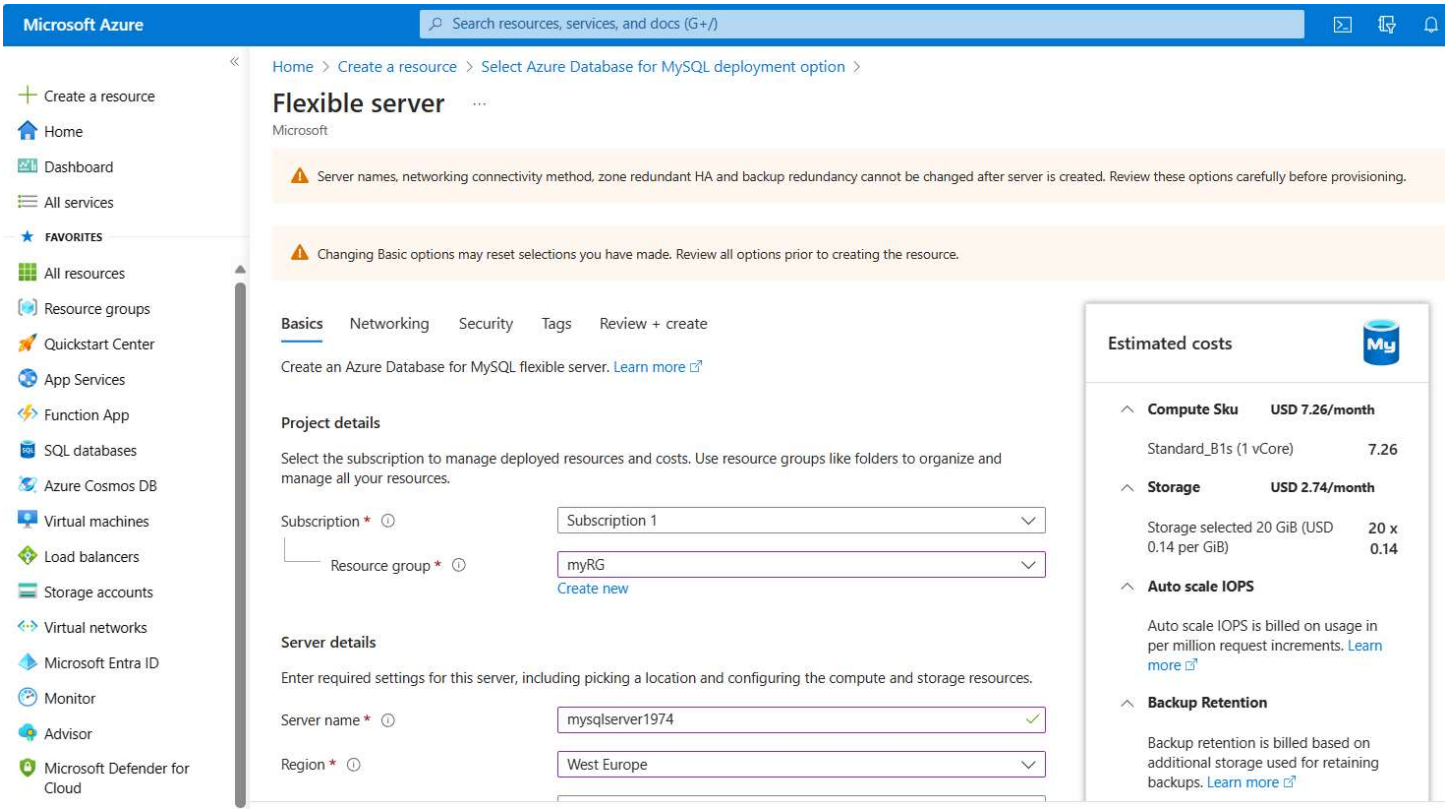
We select **Azure Database for MySQL**

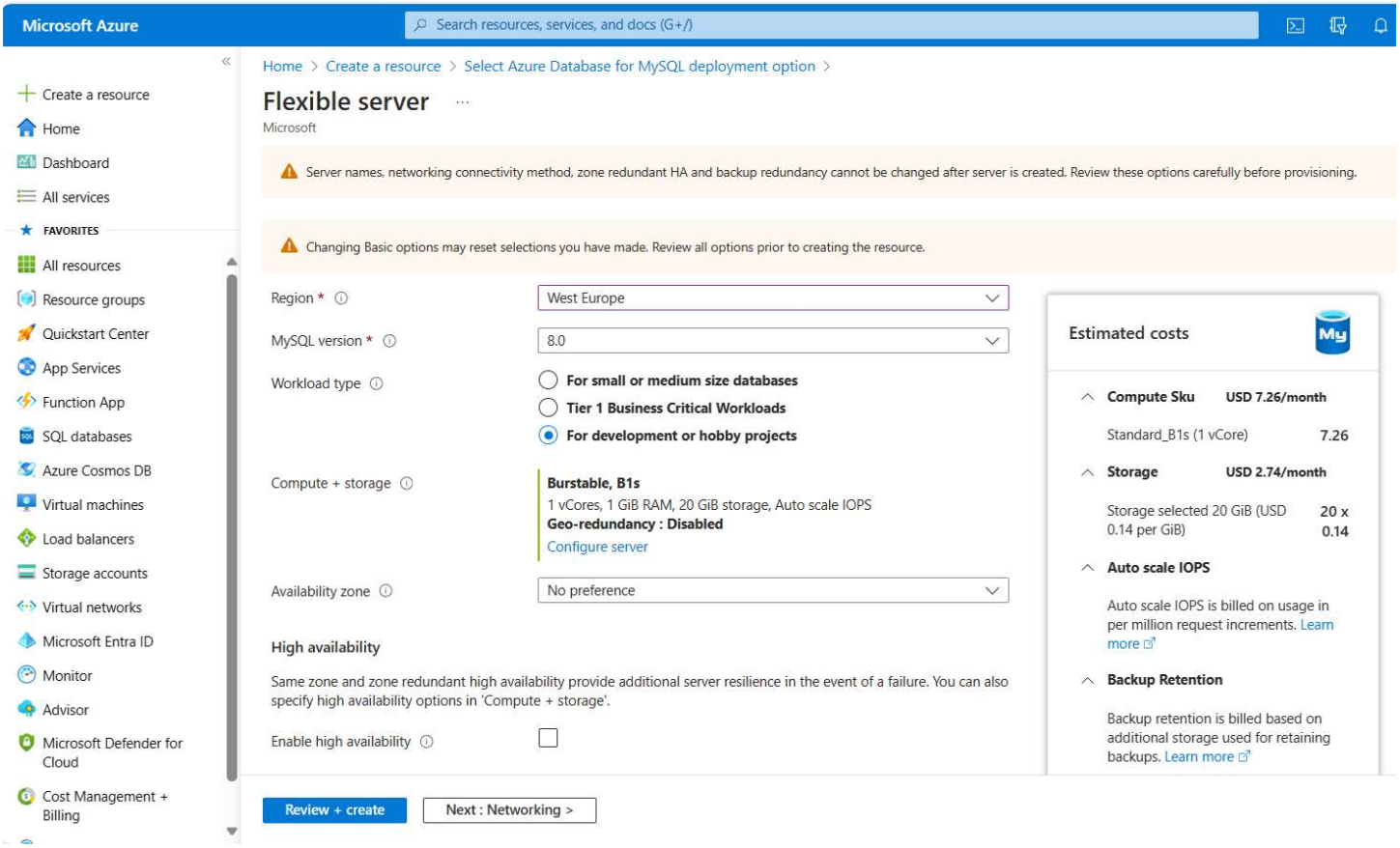


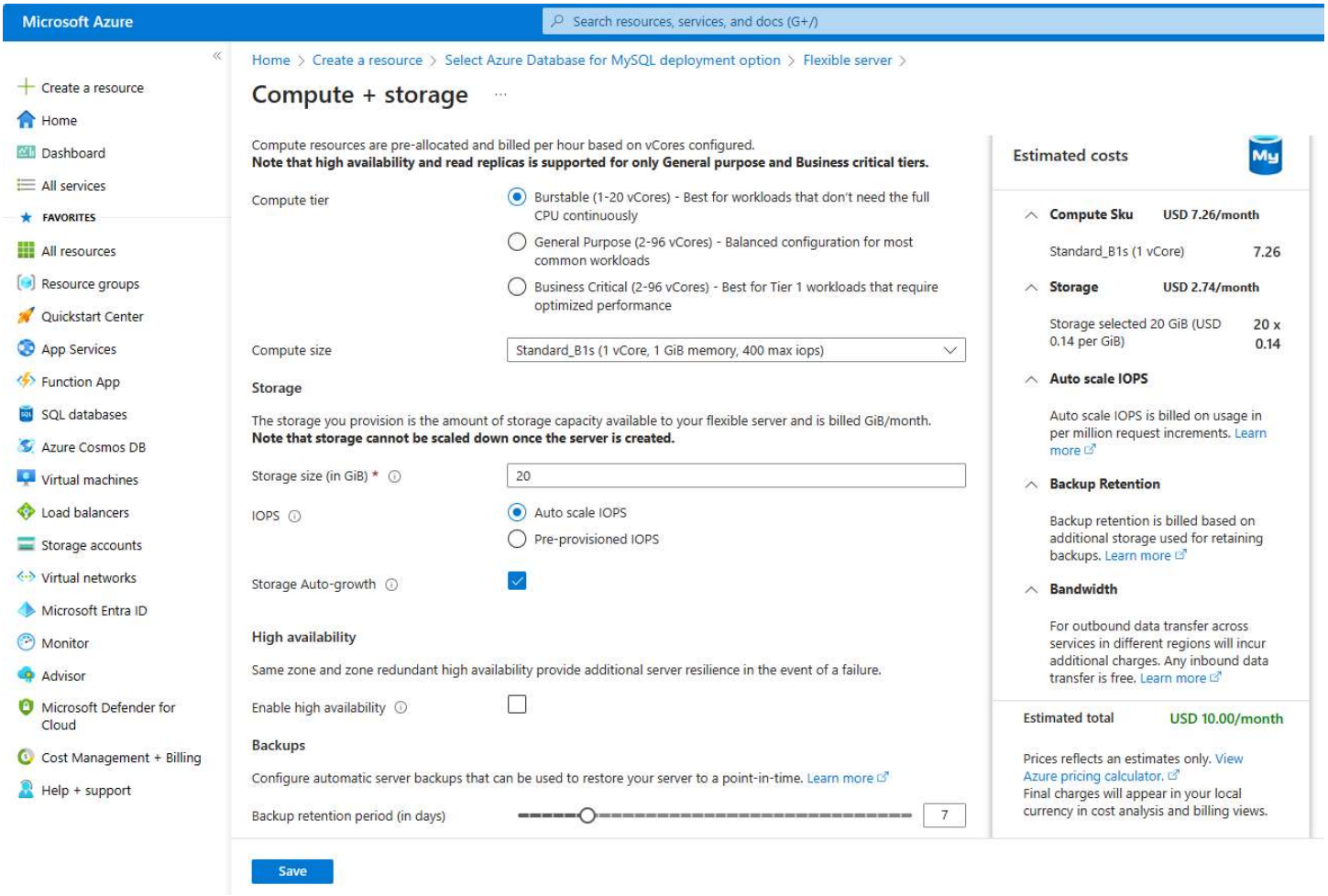
We select create **Flexible server**



Server Name: mysqlserver1974

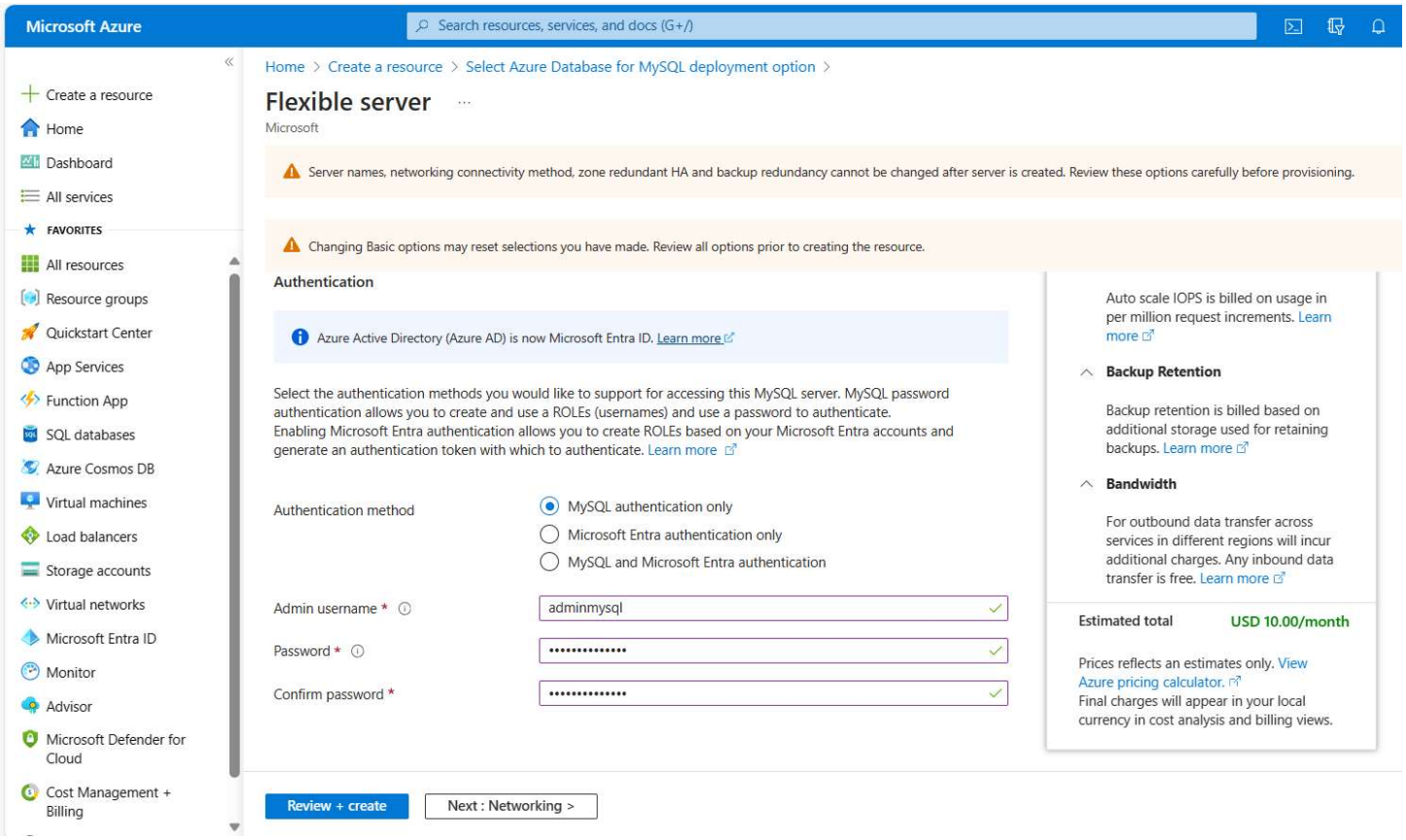

Microsoft Azure
Search resources, services, and docs (G+/)
Create a resource
Home
Dashboard
All services
FAVORITES
All resources
Resource groups
Quickstart Center
App Services
Function App
SQL databases
Azure Cosmos DB
Virtual machines
Load balancers
Storage accounts
Virtual networks
Microsoft Entra ID
Monitor
Advisor
Microsoft Defender for Cloud
Home > Create a resource > Select Azure Database for MySQL deployment option >
Flexible server
Microsoft
Server names, networking connectivity method, zone redundant HA and backup redundancy cannot be changed after server is created. Review these options carefully before provisioning.
Changing Basic options may reset selections you have made. Review all options prior to creating the resource.
Basics
Networking
Security
Tags
Review + create
Create an Azure Database for MySQL flexible server. Learn more
Project details
Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.
Subscription *
Subscription 1
Resource group *
myRG
Create new
Server details
Enter required settings for this server, including picking a location and configuring the compute and storage resources.
Server name *
mysqlserver1974
Region *
West Europe
Estimated costs
Compute Sku
USD 7.26/month
Standard_B1s (1 vCore)
7.26
Storage
USD 2.74/month
Storage selected 20 GiB (USD 0.14 per GiB)
20 x 0.14
Auto scale IOPS
Auto scale IOPS is billed on usage in per million request increments. Learn more
Backup Retention
Backup retention is billed based on additional storage used for retaining backups. Learn more


Microsoft Azure
Search resources, services, and docs (G+/)
Home > Create a resource > Select Azure Database for MySQL deployment option >
Flexible server
Microsoft
Server names, networking connectivity method, zone redundant HA and backup redundancy cannot be changed after server is created. Review these options carefully before provisioning.
Changing Basic options may reset selections you have made. Review all options prior to creating the resource.
Region *
West Europe
MySQL version *
8.0
Workload type
For small or medium size databases
Tier 1 Business Critical Workloads
For development or hobby projects
Compute + storage
Burstable, B1s
1 vCores, 1 GiB RAM, 20 GiB storage, Auto scale IOPS
Geo-redundancy : Disabled
Configure server
Availability zone
No preference
High availability
Same zone and zone redundant high availability provide additional server resilience in the event of a failure. You can also specify high availability options in 'Compute + storage'.
Enable high availability
Review + create
Next : Networking >
Estimated costs
Compute Sku
USD 7.26/month
Standard_B1s (1 vCore)
7.26
Storage
USD 2.74/month
Storage selected 20 GiB (USD 0.14 per GiB)
20 x 0.14
Auto scale IOPS
Auto scale IOPS is billed on usage in per million request increments. Learn more
Backup Retention
Backup retention is billed based on additional storage used for retaining backups. Learn more

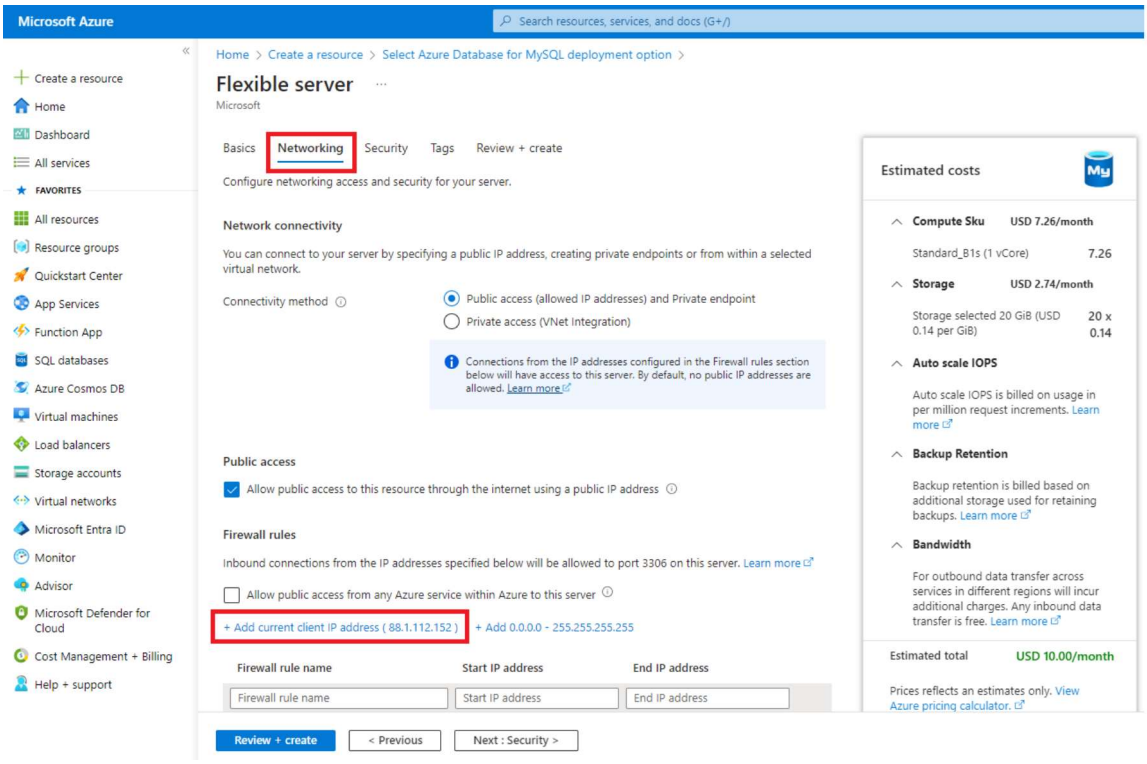


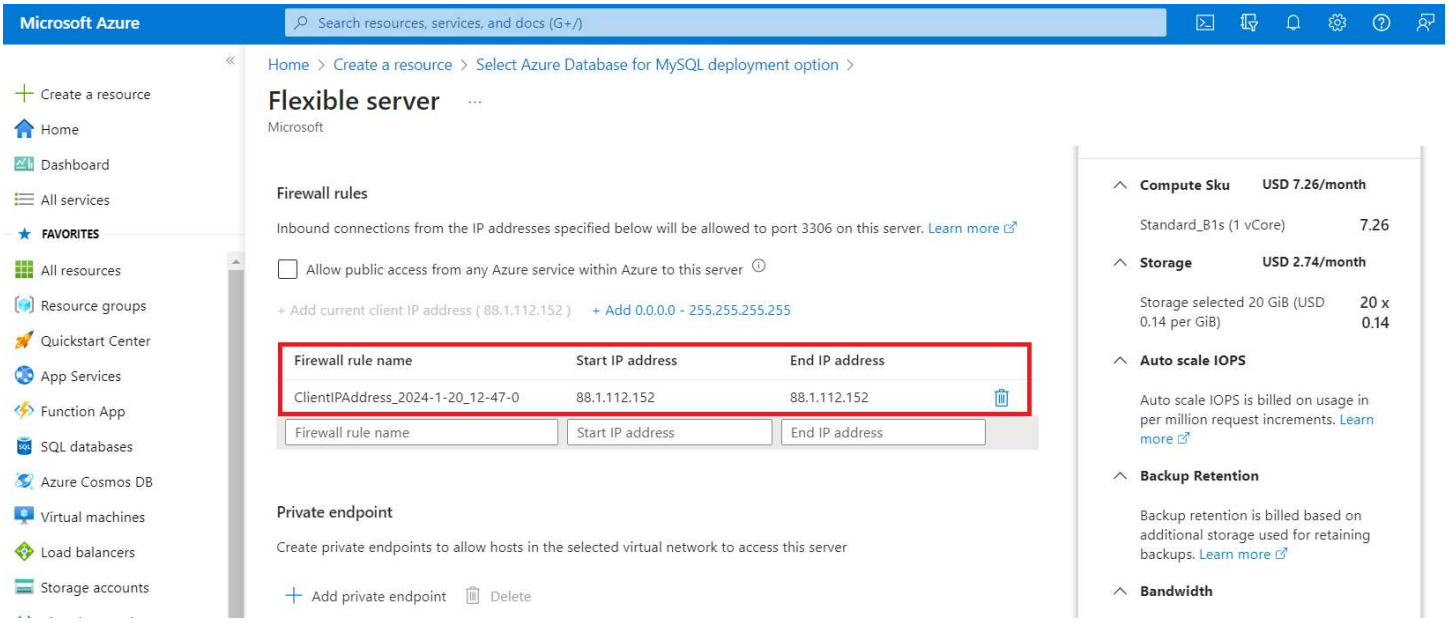
Admin username: adminmysql

Password: LuiscocoXXXXXXXXXXX

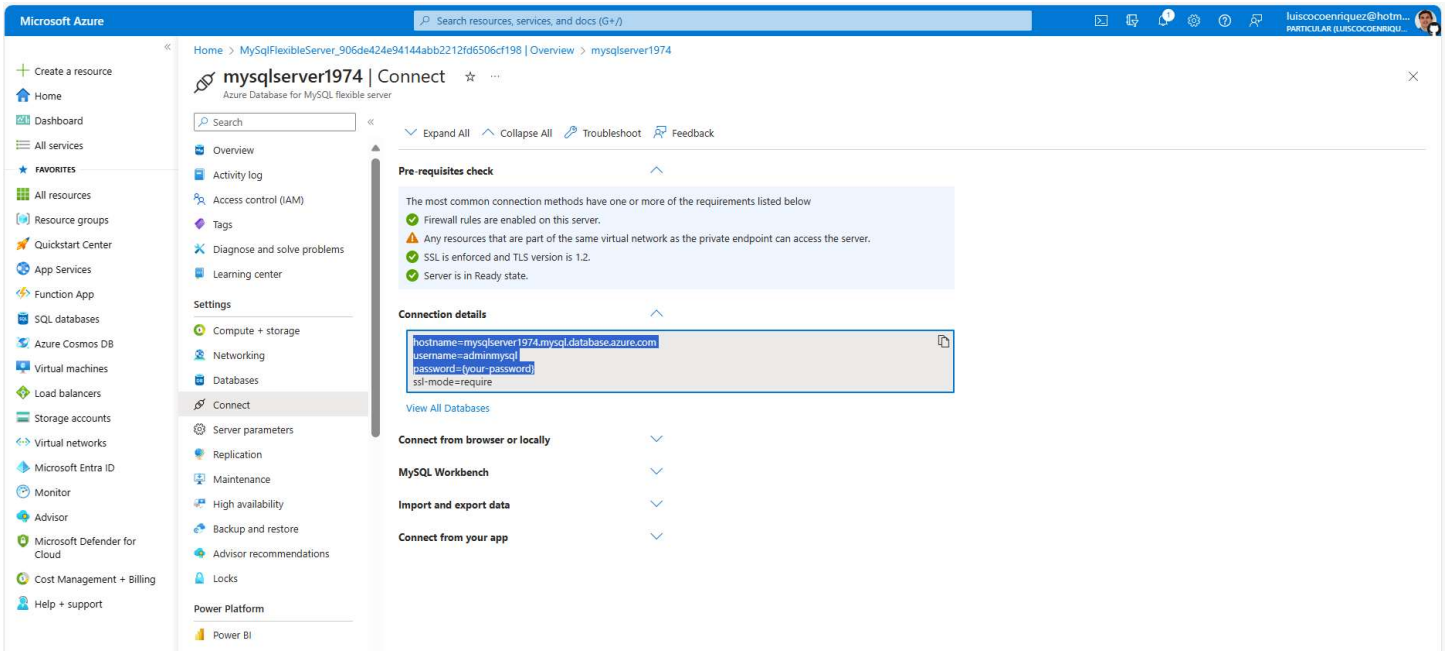


We navigate to the **Networking** tab and we add our laptop IP address as a FireWall rule



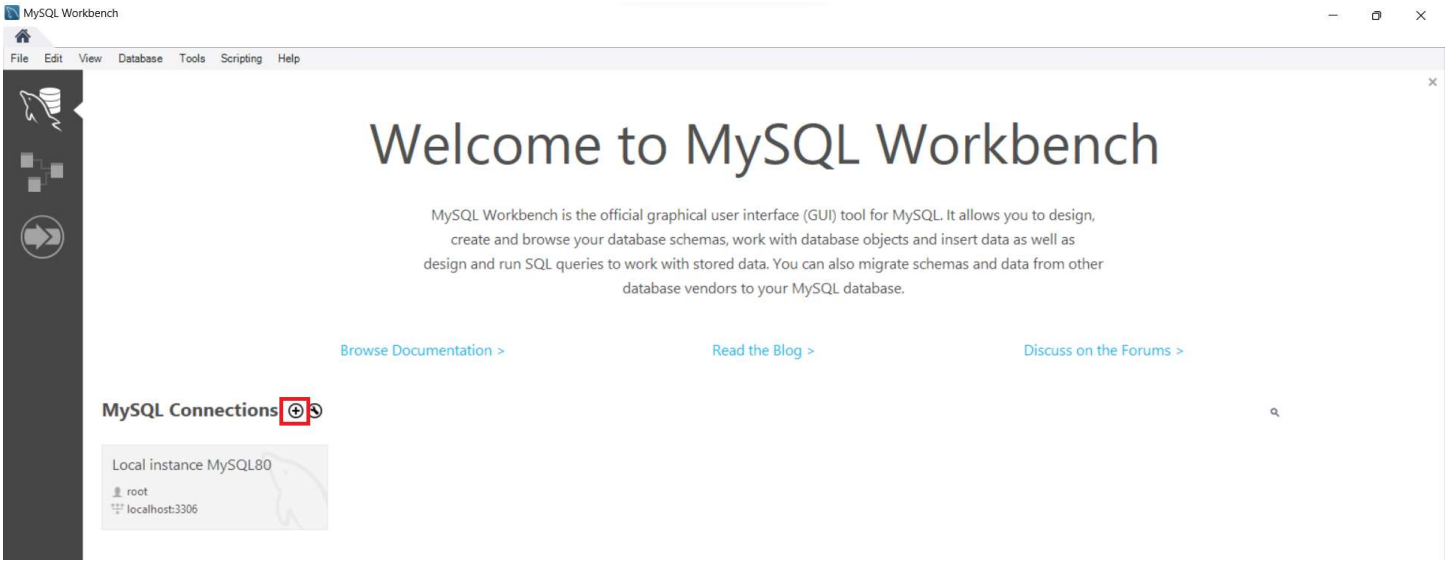


We can now access Azure MySQL from MySQL Workbench setting the hostname, username and password

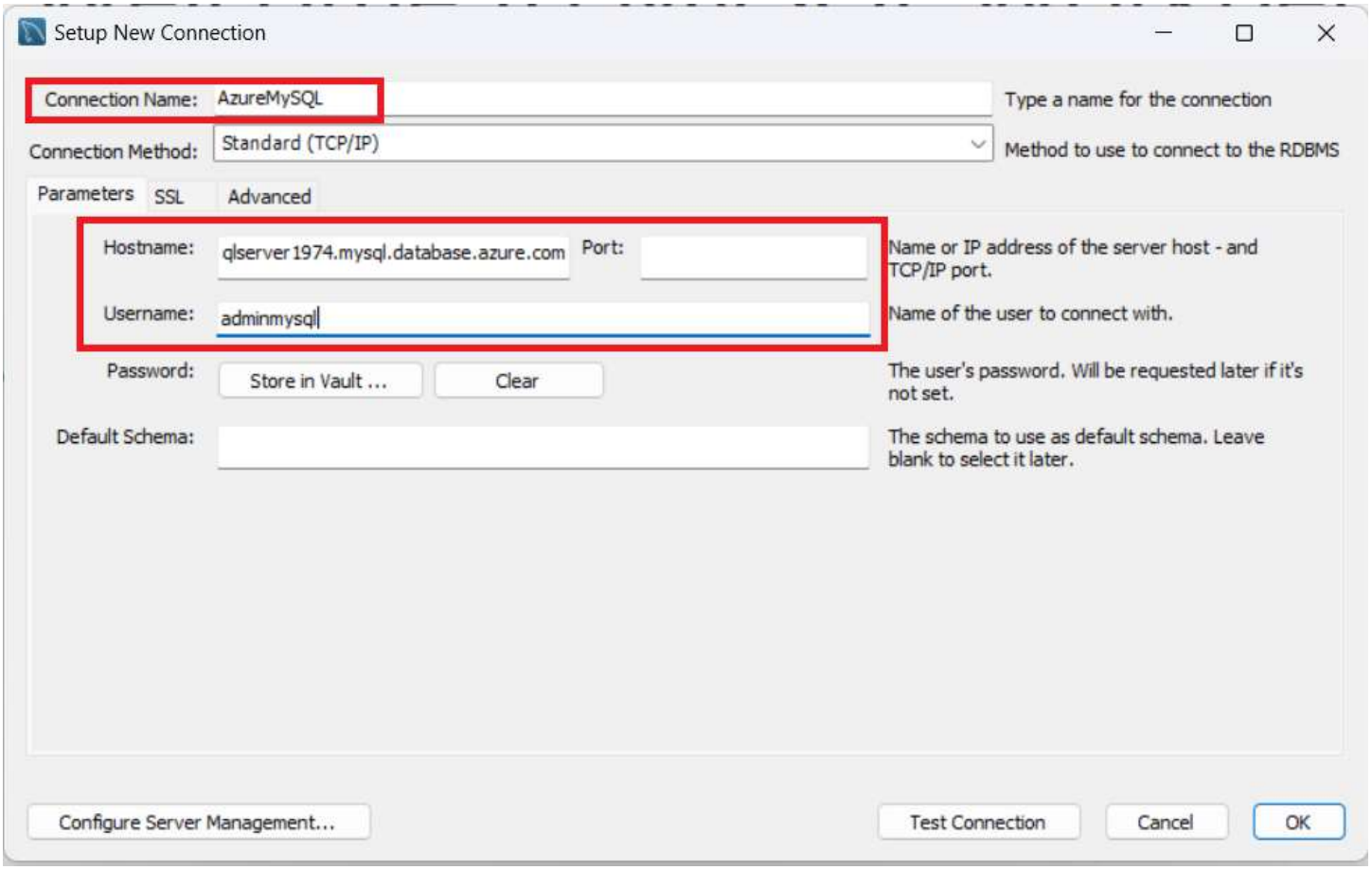


1.2. Run MySQL Workbench and create new database

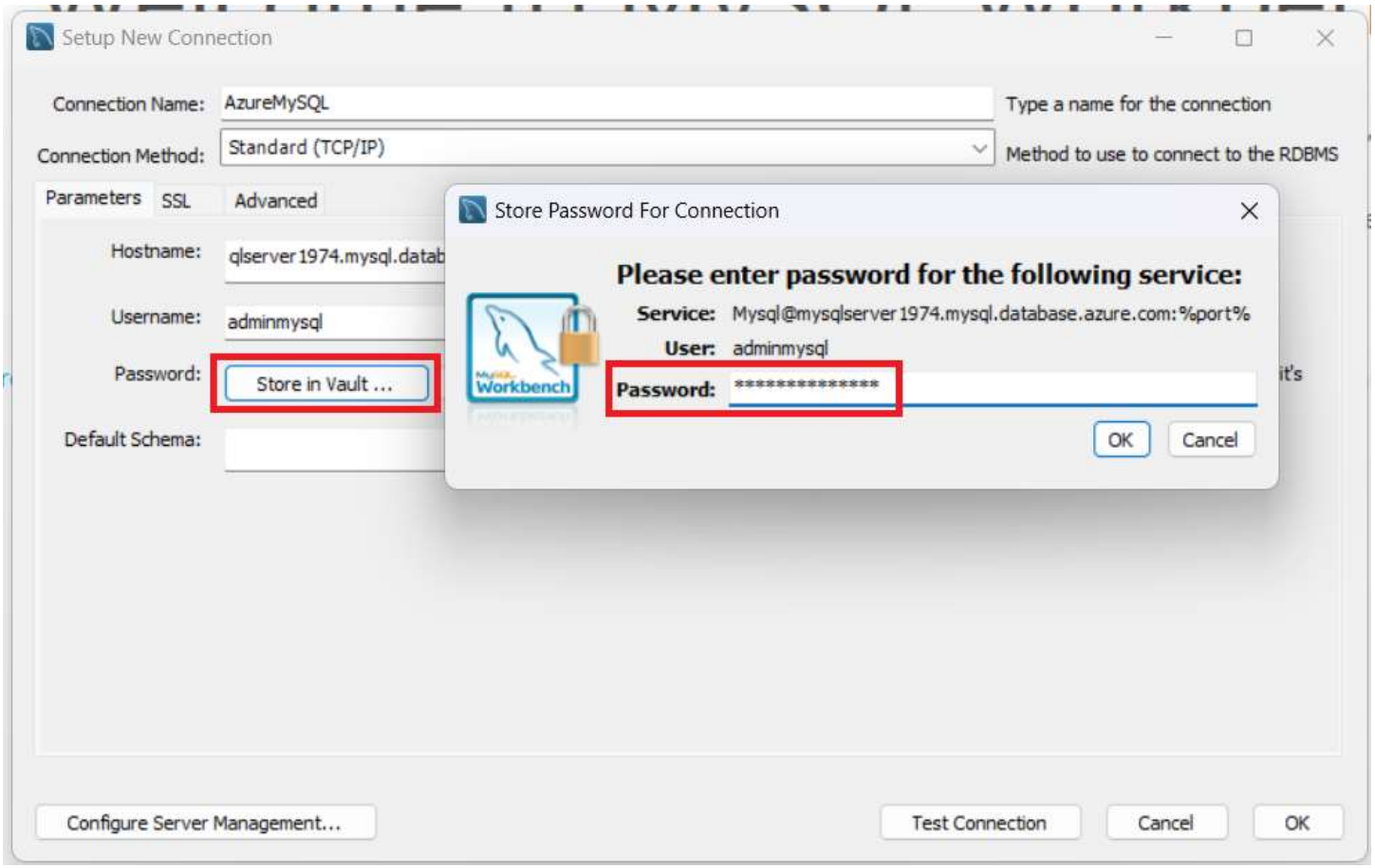
We run **MySQL Workbench** and we create a new connection



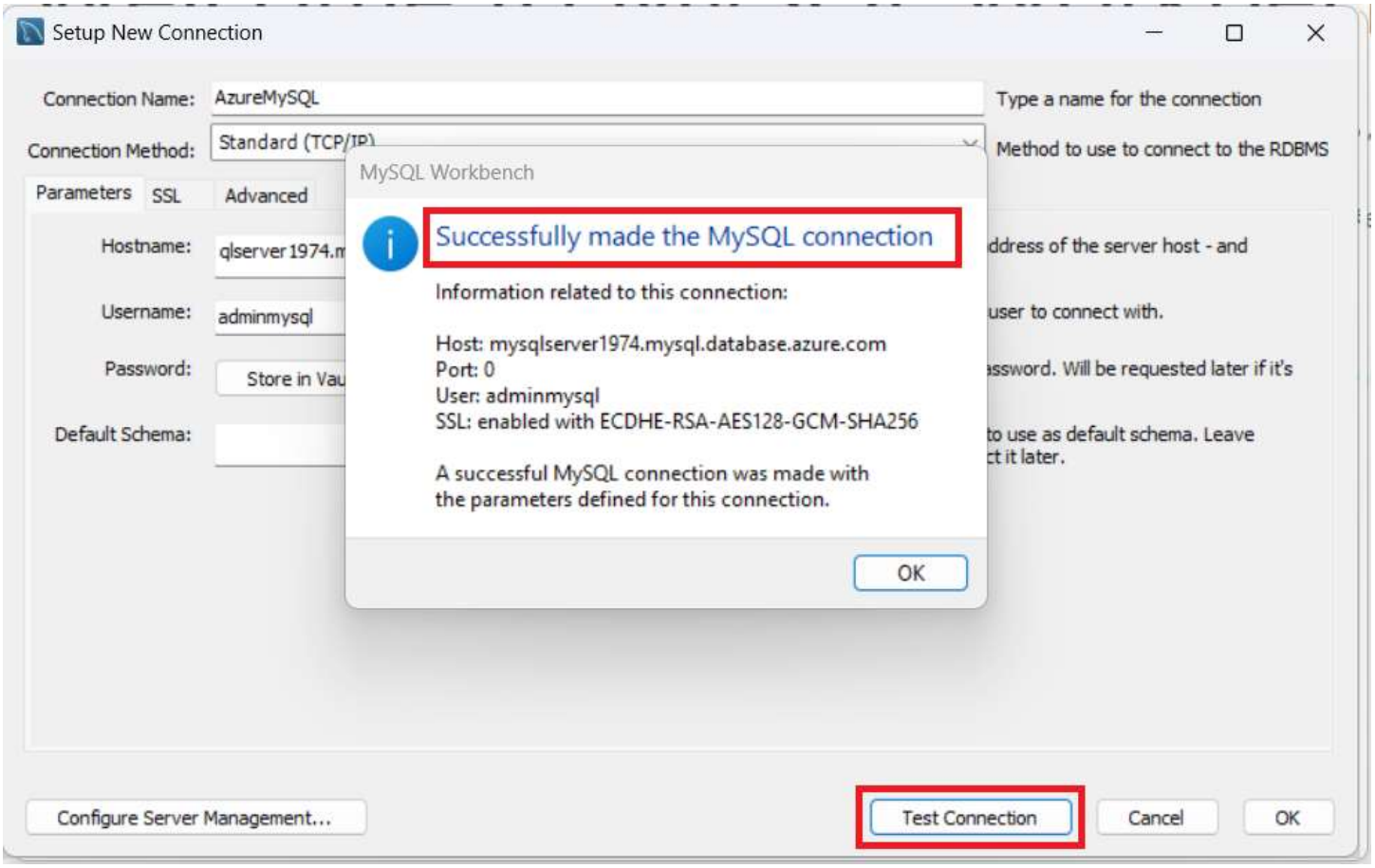
We input the new connection data

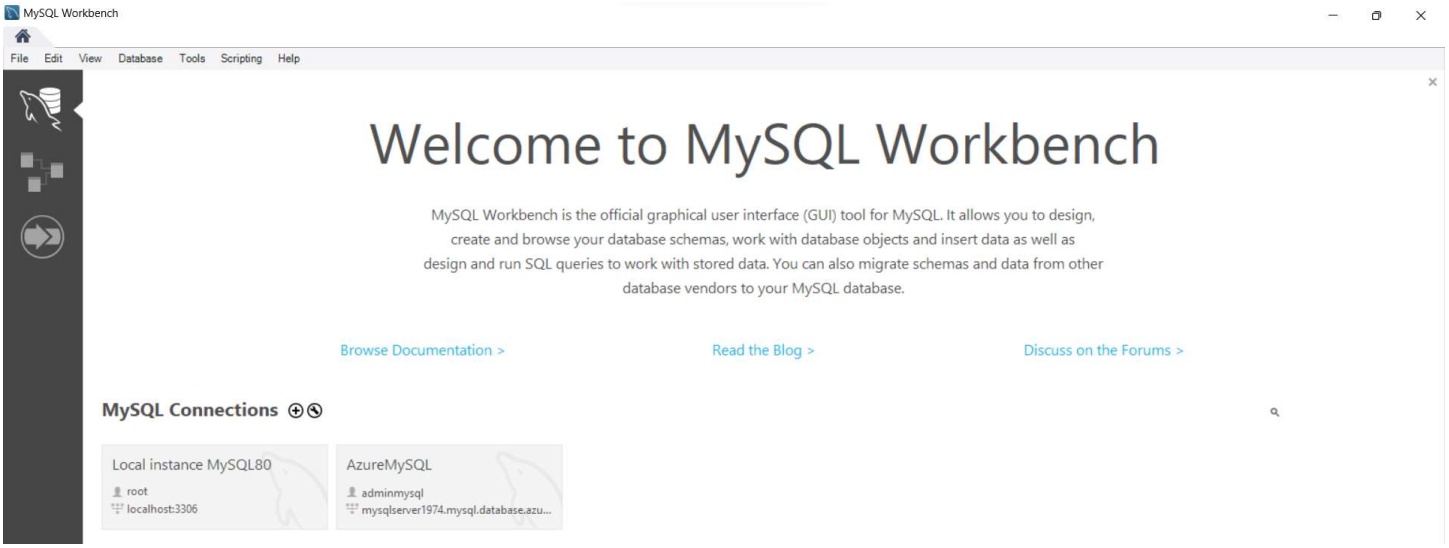


For input the password press the **Store in Vault** button



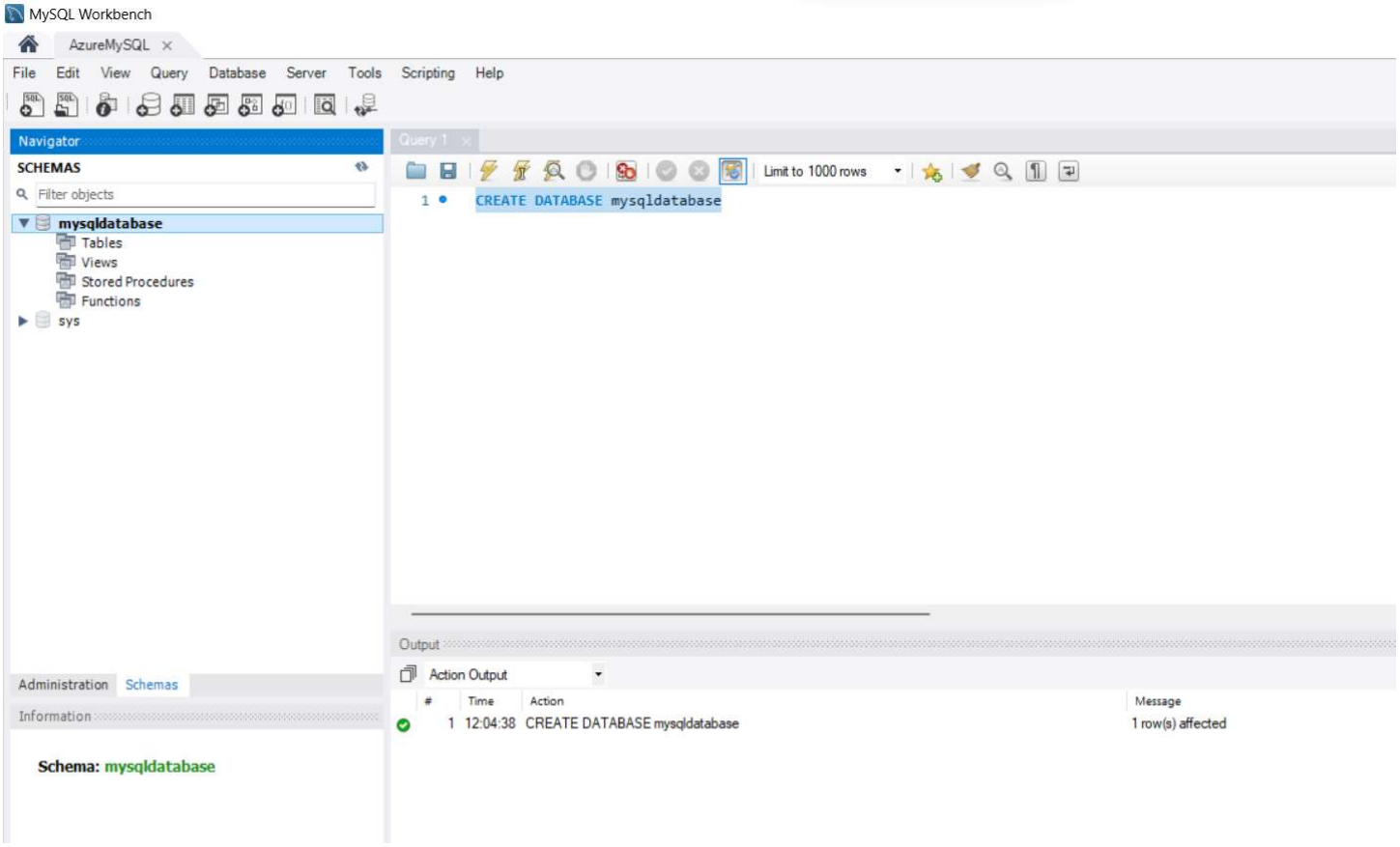
Now we can test the connection pressing on the **Test connection** button





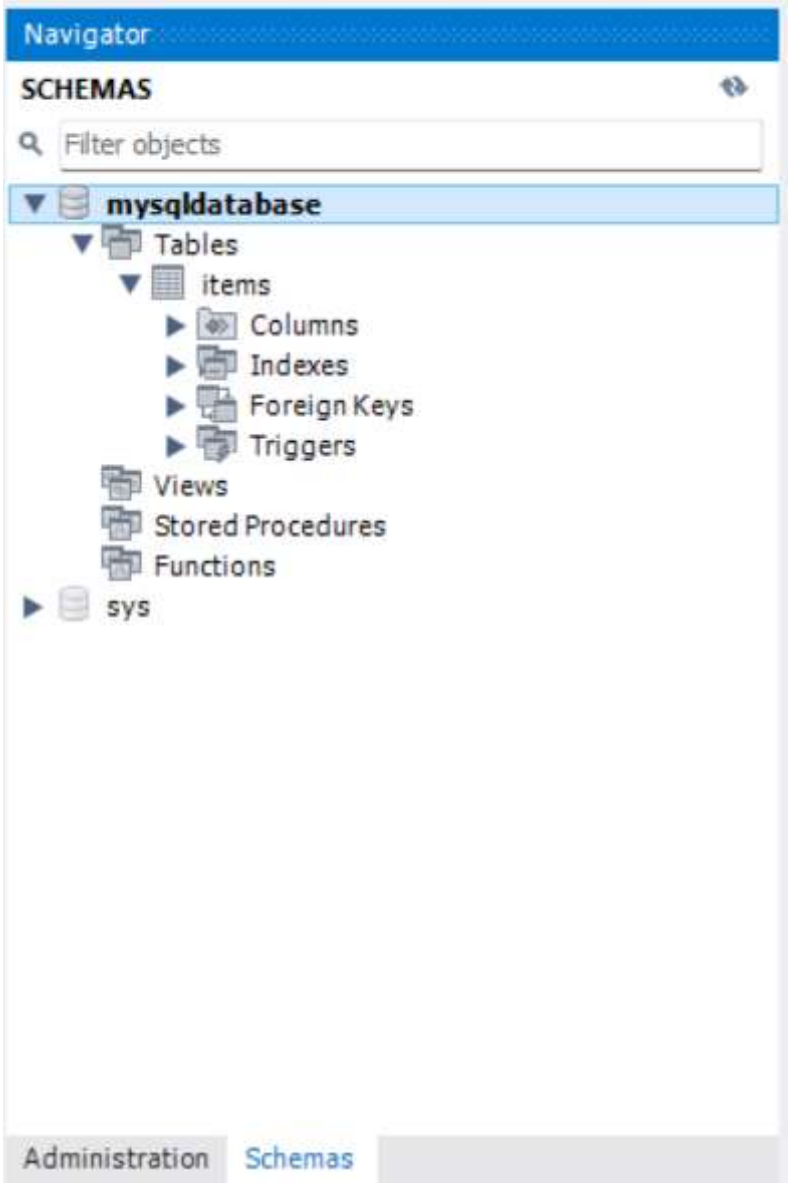
We create a database running this command

´´´sql CREATE DATABASE mysqldatabase ´´´

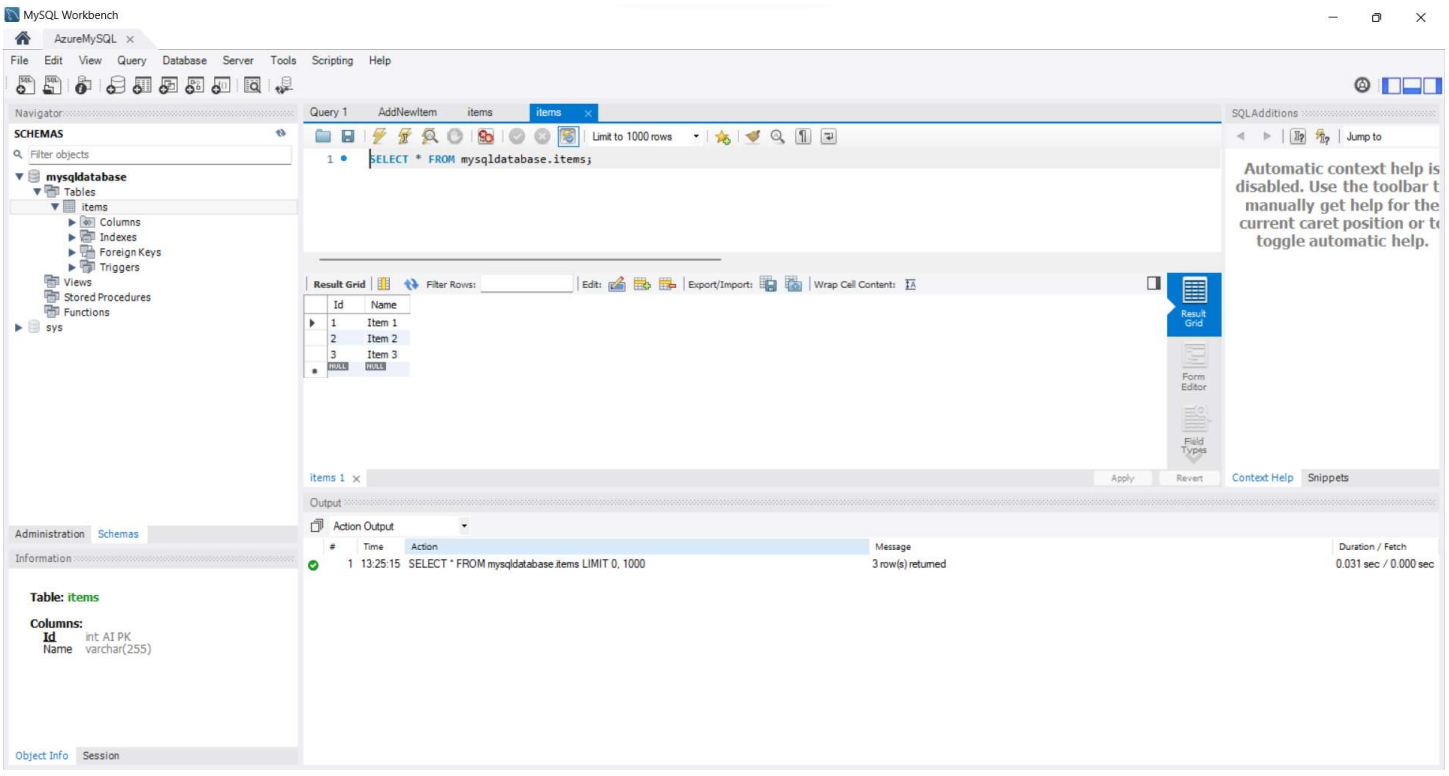


We create a new Table an insert some rows

```
CREATE TABLE Items (
    Id INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL
    -- Add other fields here as per your model
);

INSERT INTO Items (Name) VALUES ('Item 1');
INSERT INTO Items (Name) VALUES ('Item 2');
INSERT INTO Items (Name) VALUES ('Item 3');
```

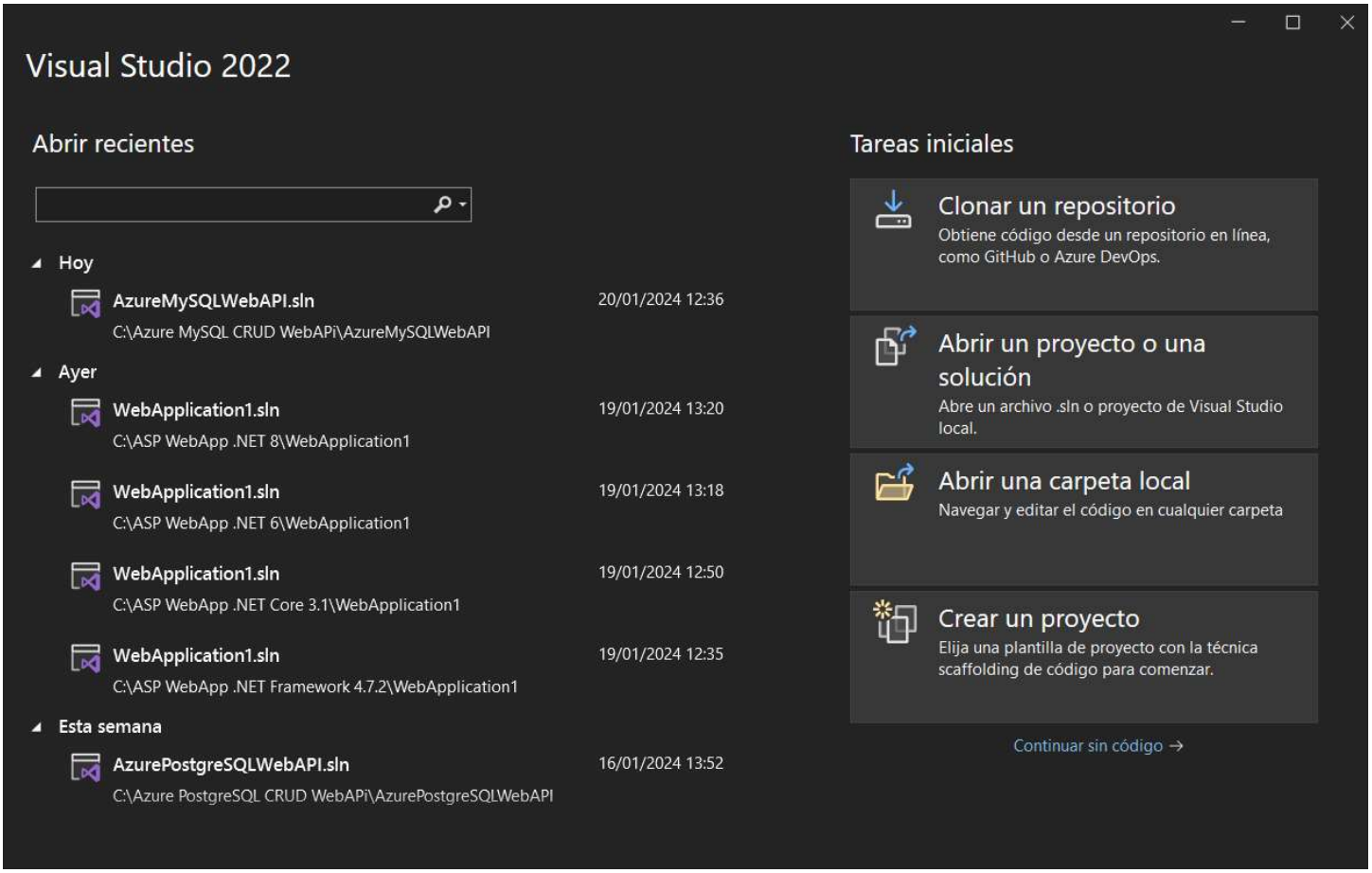


We can verify the inserted items

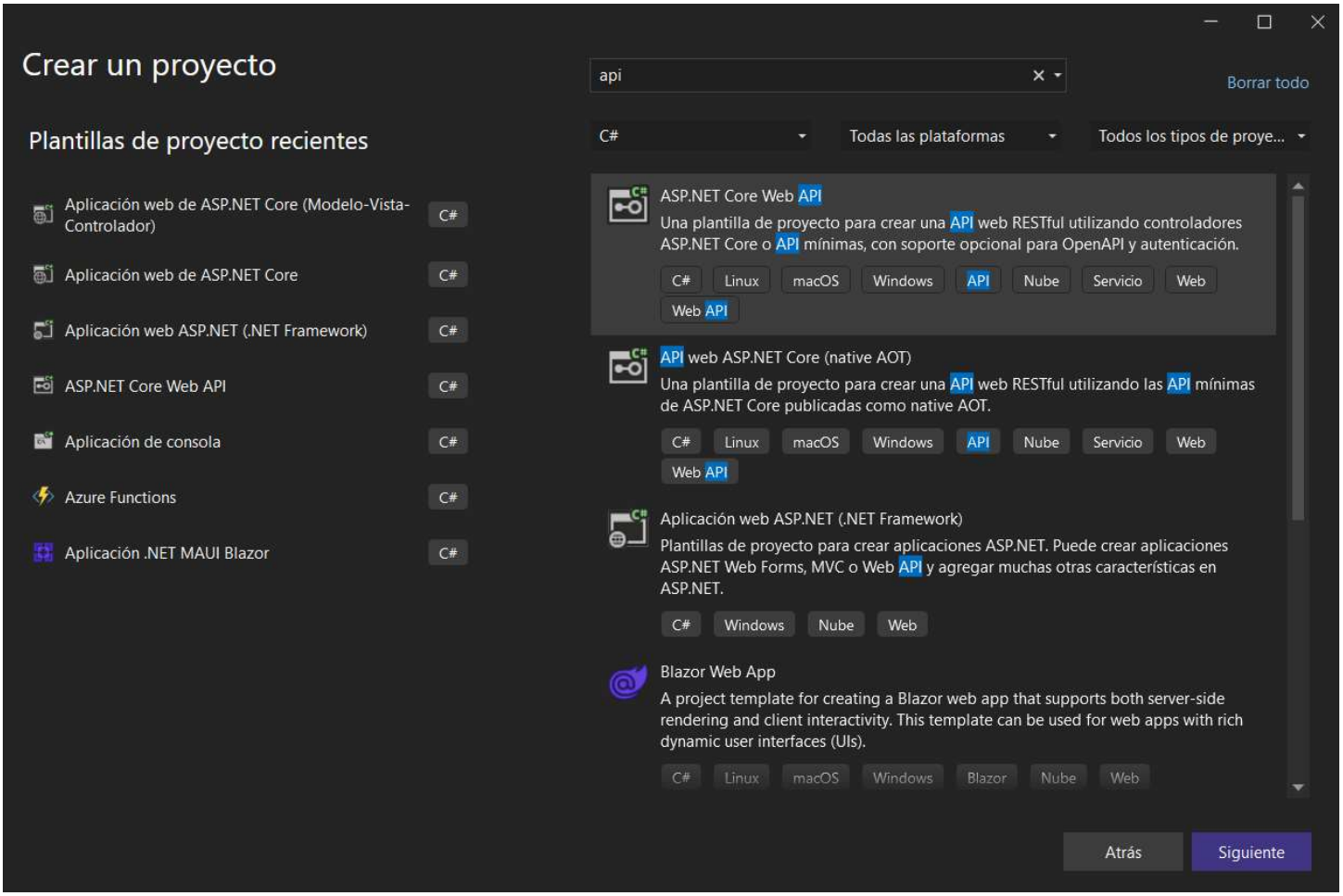


2. How to Create .NET 8 WebAPI CRUD Microservice with Dapper

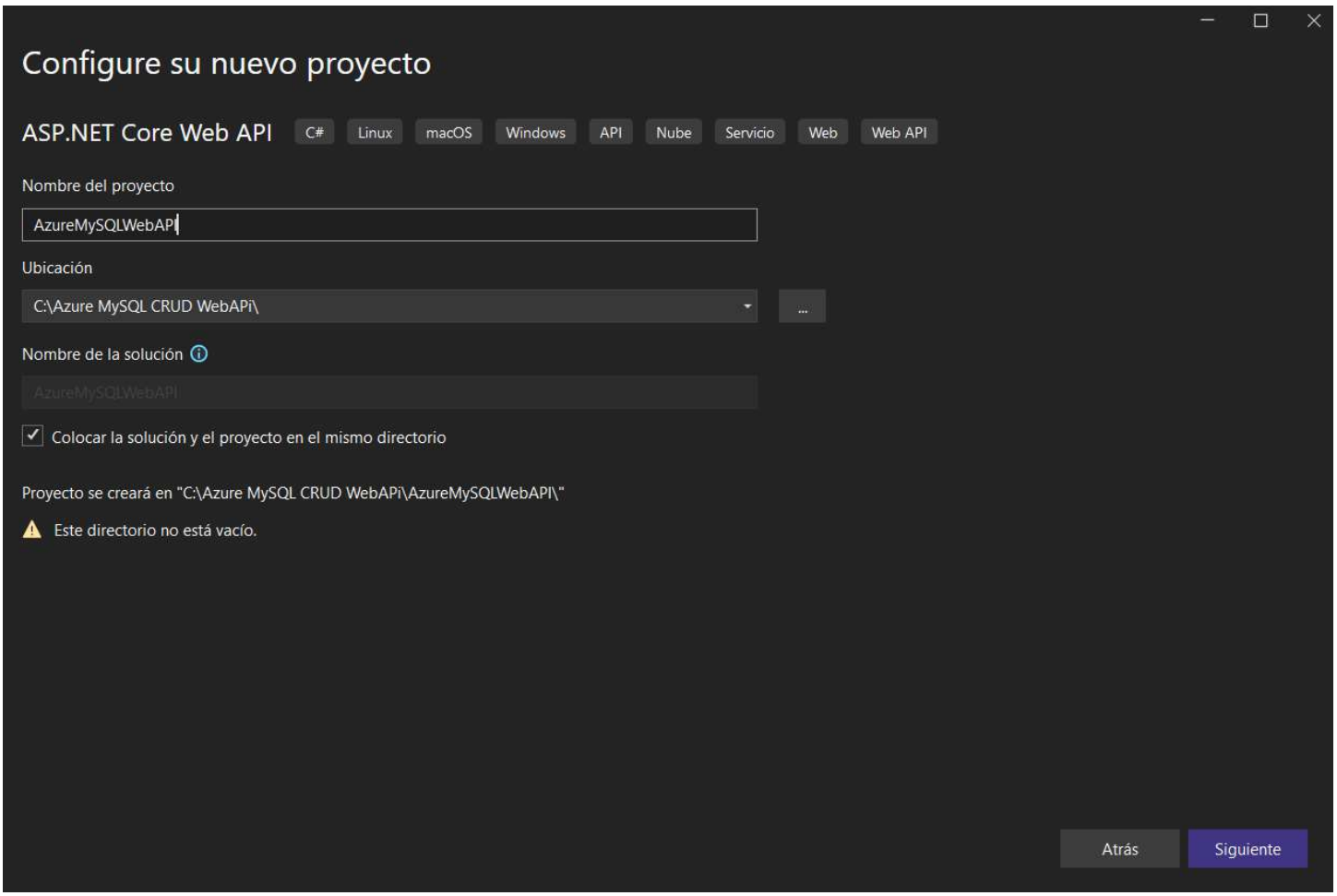
We run Visual Studio 2022 Community Edition and we create a new project



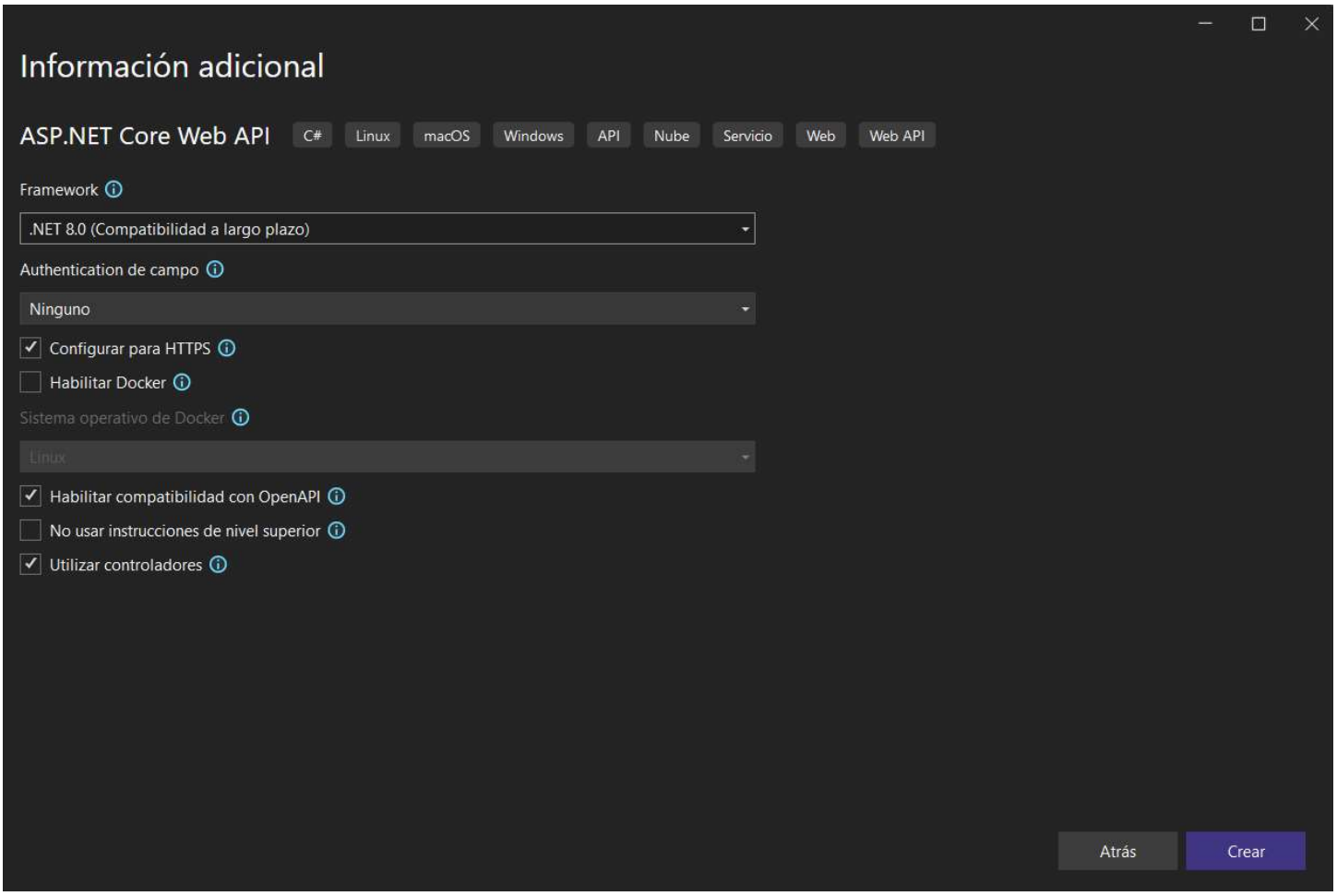
We select the **ASP Net Core Web API** project template



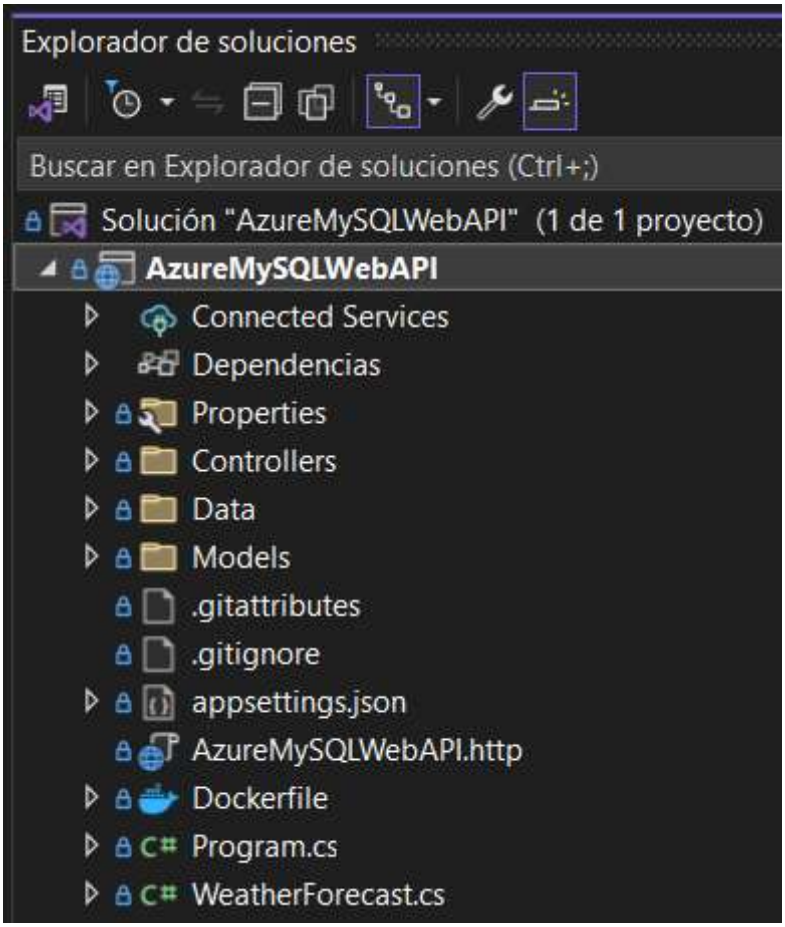
We set the project name and location



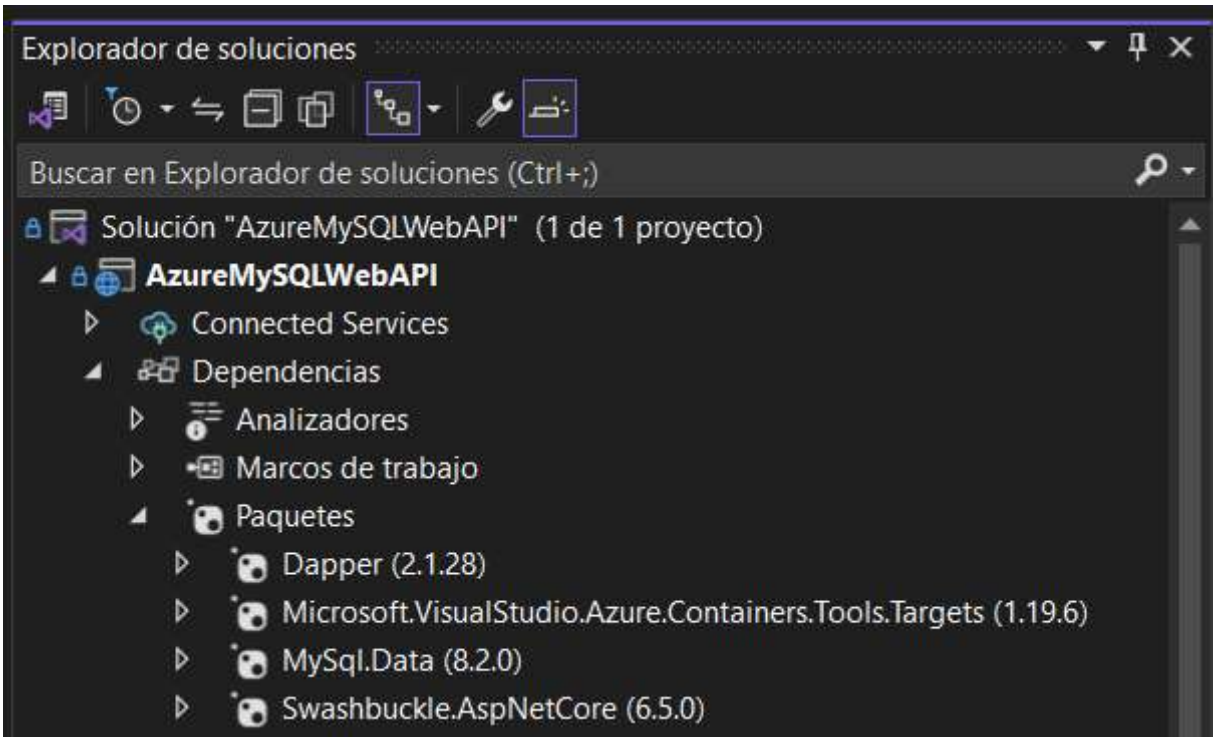
We select the project main features



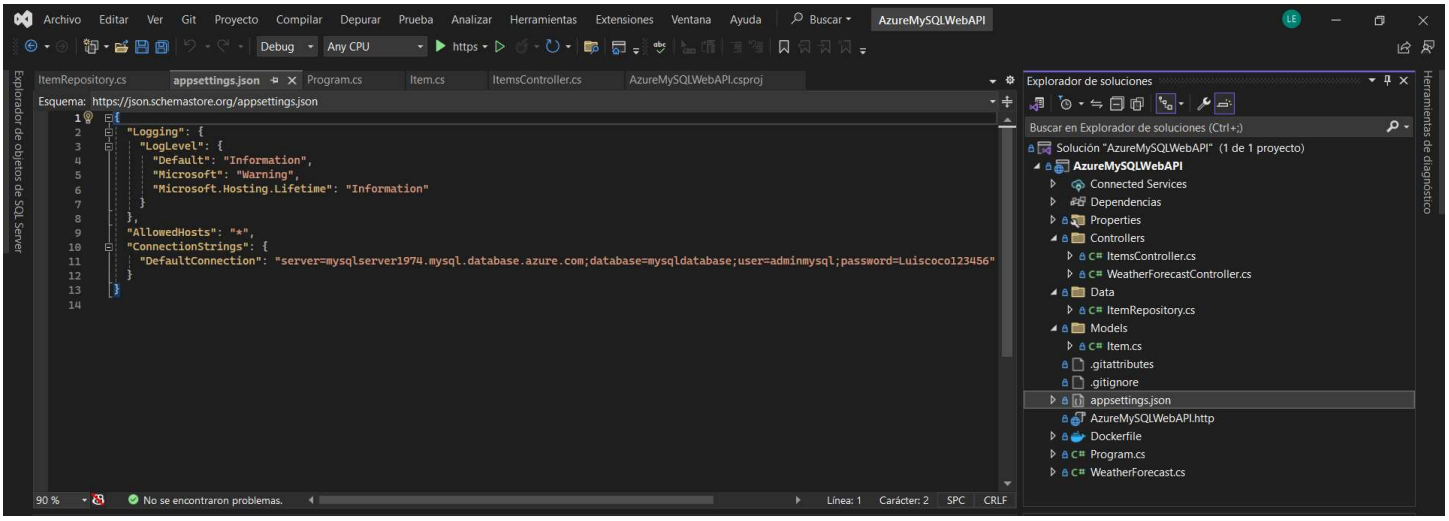
We create the following project folders structure, with the **Data** and **Models** new folders



We load the dependencies: **Dapper**, Microsoft.VisualStudio.Azure.Containers.Tools.Targets, **MySql.Data** and Swashbuckle.AspNetCore



We define in the **appsettings.json** file the database connection string




```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "server=mysqlserver1974.mysql.database.azure.com;database=mysqldataba
```

```
    }
}
```

We register the database service in the **pogram.cs** file

progam.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using AzureMySQLWebAPI.Data;
using AzureMySQLWebAPI.Controllers;

var builder = WebApplication.CreateBuilder(args);

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

// Register database repository
builder.Services.AddScoped<ItemRepository>(serviceProvider =>
    new ItemRepository(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

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

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

We create the database **Model** in the Item.cs file

Item.cs

```
namespace AzureMySQLWebAPI.Models
{
    public class Item
    {
        public int Id { get; set; }
        public string Name { get; set; }
        // Add other properties as needed
```

```
    }
}
```

We create the database **Repository** in the ItemRepository.cs file

ItemRepository.cs

```
using Dapper;
using MySql.Data.MySqlClient;
using System.Collections.Generic;
using System.Data;
using System.Threading.Tasks;
using AzureMySQLWebAPI.Models;

namespace AzureMySQLWebAPI.Data
{
    public class ItemRepository
    {
        private readonly string _connectionString;

        public ItemRepository(string connectionString)
        {
            _connectionString = connectionString;
        }

        public async Task<IEnumerable<Item>> GetAllItemsAsync()
        {
            using (IDbConnection db = new MySqlConnection(_connectionString))
            {
                return await db.QueryAsync<Item>("SELECT * FROM Items");
            }
        }

        // Add method to retrieve a single item by id
        public async Task<Item> GetItemByIdAsync(int id)
        {
            using (IDbConnection db = new MySqlConnection(_connectionString))
            {
                return await db.QueryFirstOrDefaultAsync<Item>("SELECT * FROM Items WHERE Id =
            }
        }

        // Add method to insert a new item
        public async Task<int> AddItemAsync(Item item)
        {
            using (IDbConnection db = new MySqlConnection(_connectionString))
            {
                var sql = "INSERT INTO Items (Name) VALUES (@Name); SELECT LAST_INSERT_ID();";
                return await db.ExecuteScalarAsync<int>(sql, item);
            }
        }
```

```
        // Add method to update an existing item
        public async Task UpdateItemAsync(Item item)
        {
            using (IDbConnection db = new MySqlConnection(_connectionString))
            {
                var sql = "UPDATE Items SET Name = @Name WHERE Id = @Id";
                await db.ExecuteAsync(sql, item);
            }
        }

        // Add method to delete an item
        public async Task DeleteItemAsync(int id)
        {
            using (IDbConnection db = new MySqlConnection(_connectionString))
            {
                await db.ExecuteAsync("DELETE FROM Items WHERE Id = @Id", new { Id = id });
            }
        }

        public async Task AddItemUsingStoredProcedureAsync(Item item)
        {
            using (IDbConnection db = new MySqlConnection(_connectionString))
            {
                var parameters = new DynamicParameters();
                parameters.Add("itemName", item.Name, DbType.String);

                await db.ExecuteAsync("AddNewItem", parameters, commandType: CommandType.Store
            }
        }
    }
}
```

We create the **Controller** for defining the database CRUD actions

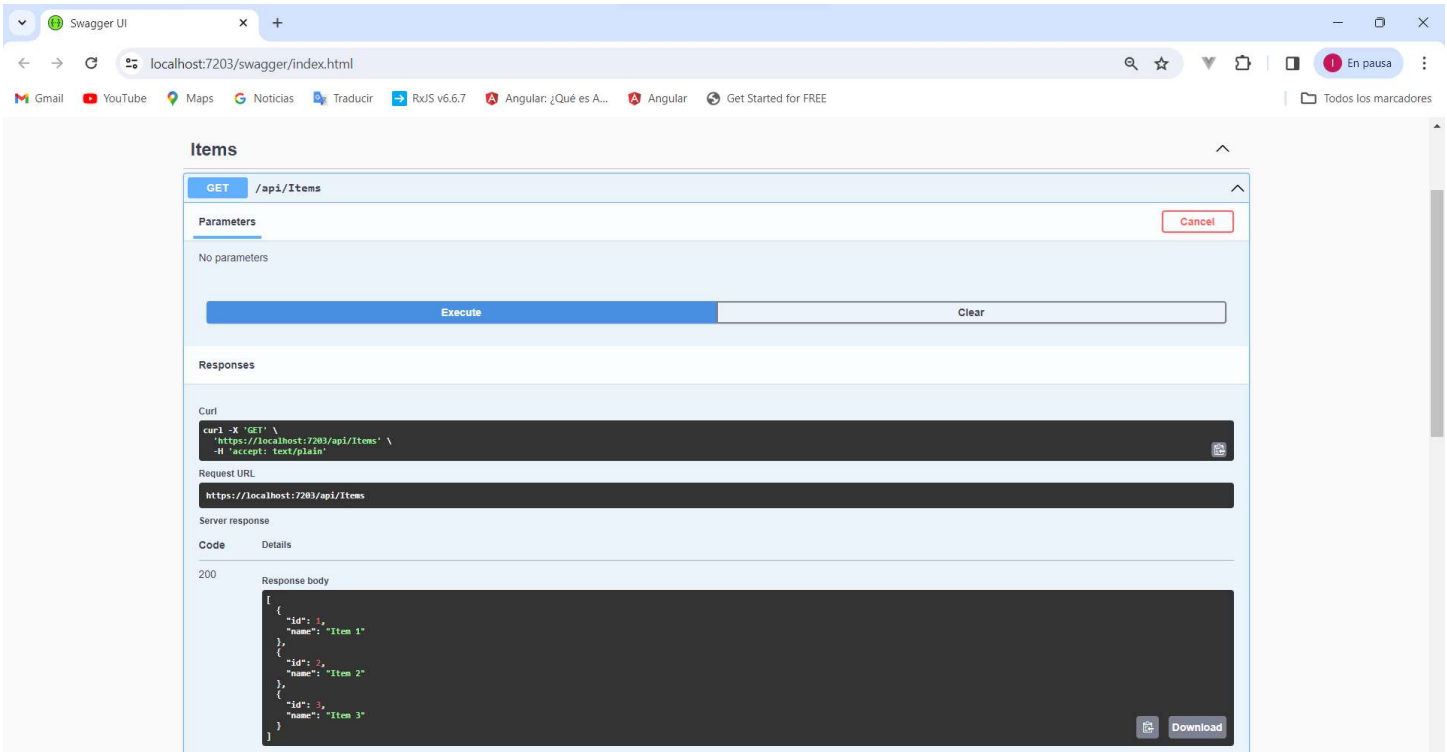
ItemsController.cs

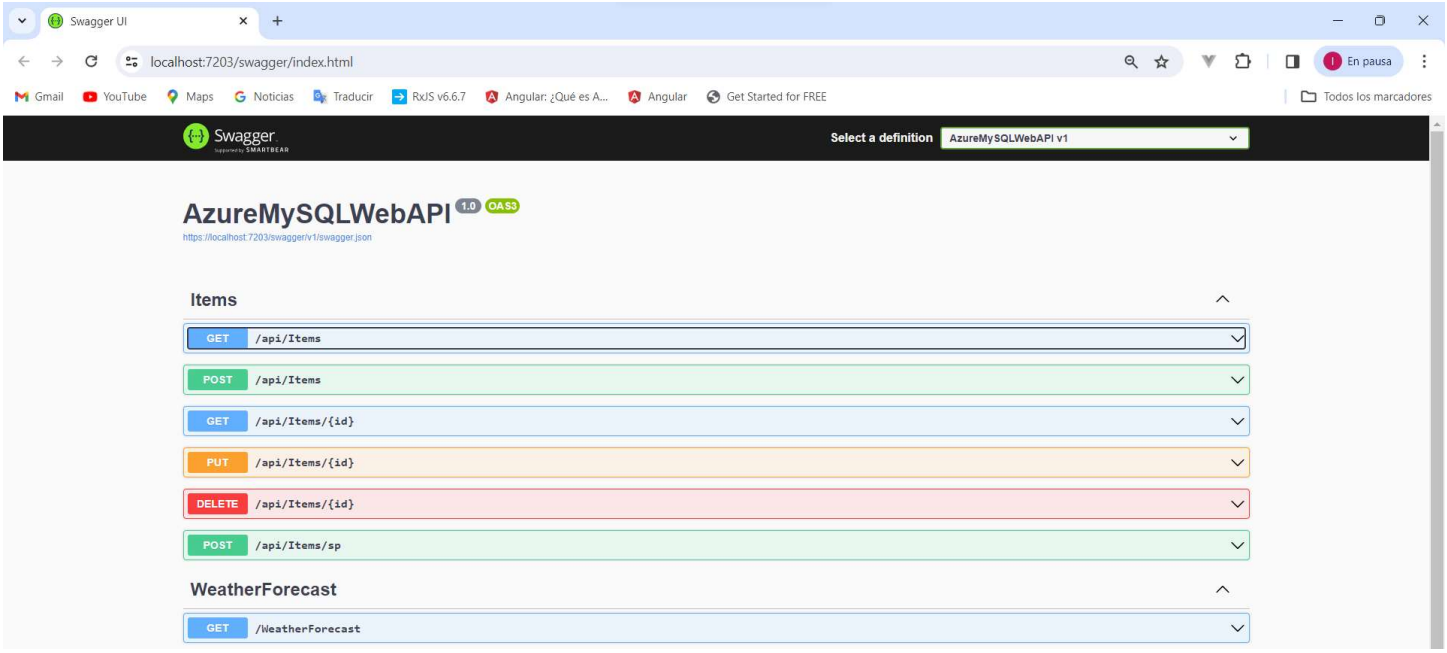
```
using Microsoft.AspNetCore.Mvc;
using AzureMySQLWebAPI.Data;
using AzureMySQLWebAPI.Models;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace AzureMySQLWebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ItemsController : ControllerBase
    {
        private readonly ItemRepository _repository;
```

```
public ItemsController(ItemRepository repository)
{
    _repository = repository;
}

// GET: api/items
[HttpGet]
public async Task<ActionResult<IEnumerable<Item>>> GetItems()
{
    var items = await _repository.GetAllItemsAsync();
    return Ok(items);
}

// GET: api/items/5
[HttpGet("{id}")]
public async Task<ActionResult<Item>> GetItem(int id)
{
    var item = await _repository.GetItemByIdAsync(id);
    if (item == null)
    {
        return NotFound();
    }
    return item;
}

// POST: api/items
[HttpPost]
public async Task<ActionResult<Item>> PostItem(Item item)
{
    int id = await _repository.AddItemAsync(item);
    item.Id = id;
    return CreatedAtAction(nameof(GetItem), new { id = item.Id }, item);
}

// PUT: api/items/5
[HttpPut("{id}")]
public async Task<IActionResult> PutItem(int id, Item item)
{
    if (id != item.Id)
    {
        return BadRequest();
    }

    await _repository.UpdateItemAsync(item);

    return NoContent();
}

// DELETE: api/items/5
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteItem(int id)
{
```

```
            await _repository.DeleteItemAsync(id);
            return NoContent();
        }

        // POST: api/items/sp
        [HttpPost("sp")]
        public async Task<IActionResult> PostItemUsingStoredProcedure([FromBody] Item item)
        {
            await _repository.AddItemUsingStoredProcedureAsync(item);
            return CreatedAtAction("GetItem", new { id = item.Id }, item);
        }
    }
}
```

We **run the application** and we verify the enpoints with swagger





3. How to deploy the WebAPI Microservice to Docker Desktop

We right click on the project and we add Docker support...

Automatically Visual Studio will create the Dockerfile

Dockerfile

```
#See https://aka.ms/customizecontainer to learn how to customize your debug container and how

FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER app
WORKDIR /app
EXPOSE 8080
EXPOSE 8081

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["AzureMySQLWebAPI.csproj", "."]
RUN dotnet restore "./././AzureMySQLWebAPI.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "./AzureMySQLWebAPI.csproj" -c $BUILD_CONFIGURATION -o /app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./AzureMySQLWebAPI.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseA

FROM base AS final
WORKDIR /app
```

```
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "AzureMySQLWebAPI.dll"]
```

We right click on the project and select **Open in Terminal** and the we run the following command to create the Docker image

```
docker build -t myapp:latest .
```

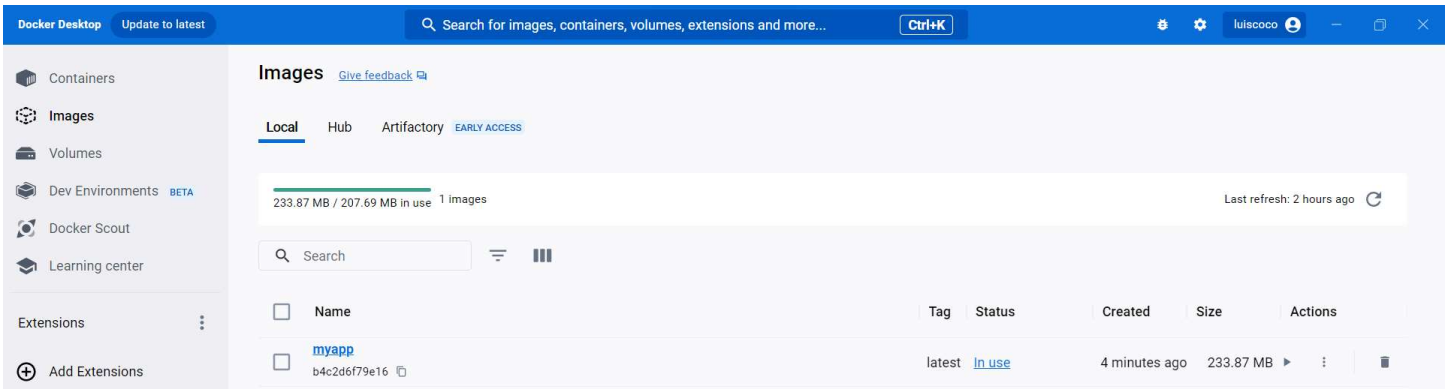
We verify the **Docker image** was created with the command

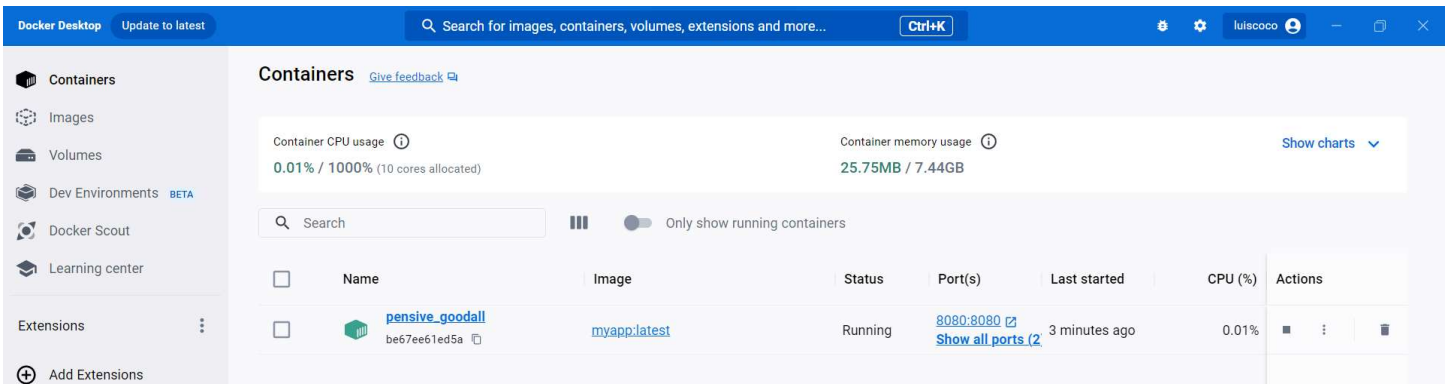
```
docker images
```

We run the **Docker container** with the following command

```
docker run -d -p 8080:8080 myapp:latest
```

Also in **Docker Desktop** we can see the Docker image and the running container



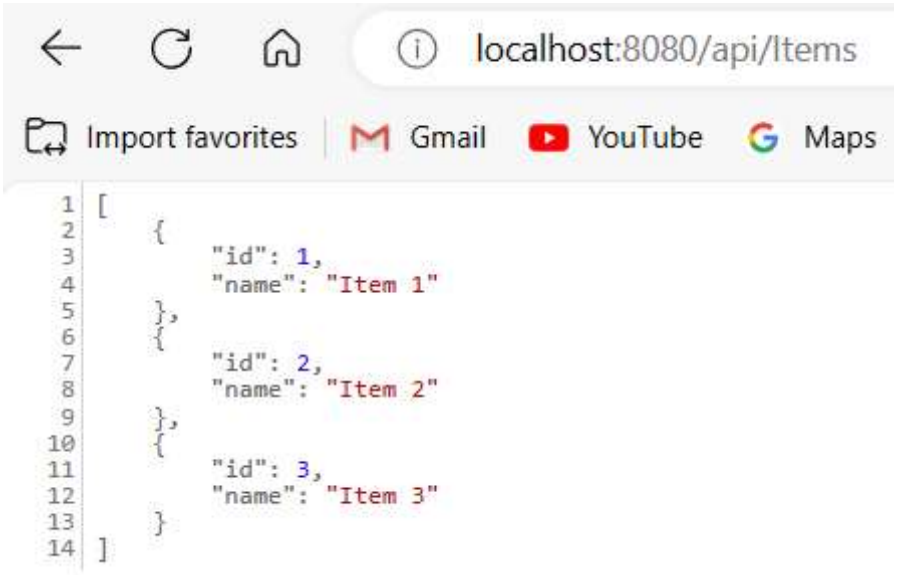


We verify the running docker container with the command

```
docker ps
```

We can access with **HTTP** protocol to the **application endpoints**

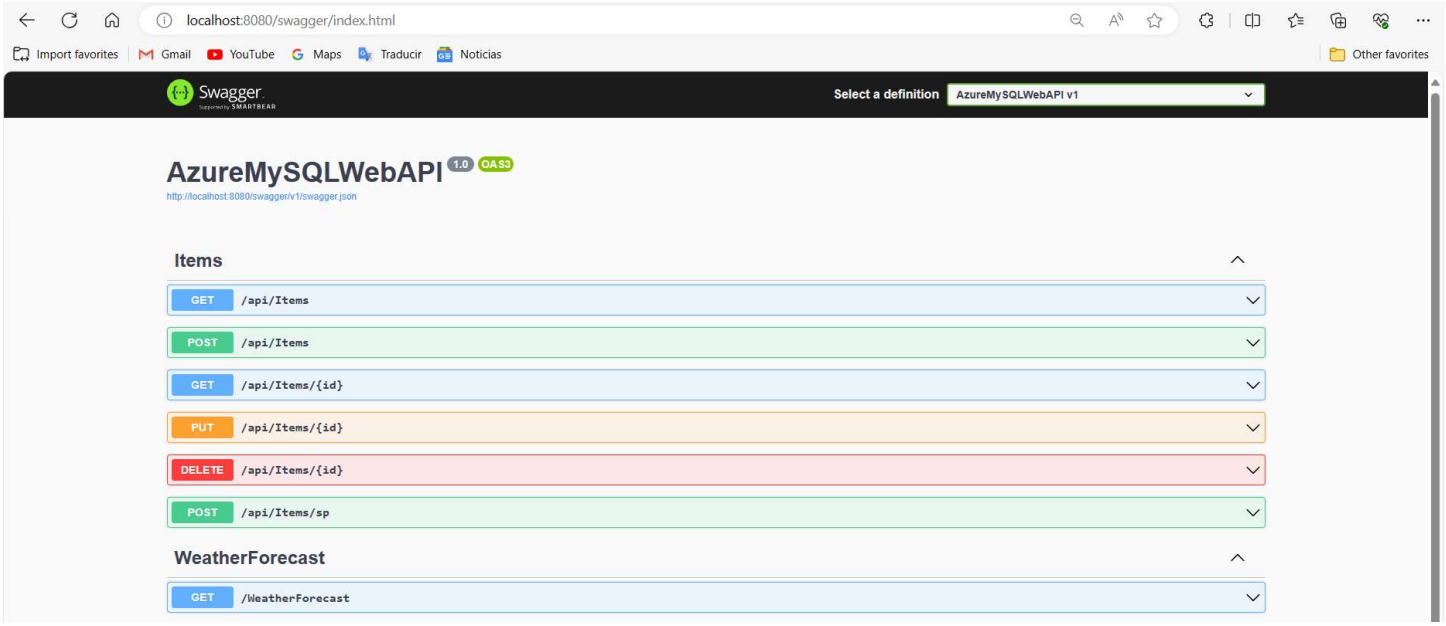
http://localhost:8080/api/Items



If we would like to access with Swagger the API documentation, we first should comment in program.cs the following lines:

```
...
// Configure the HTTP request pipeline.
//if (app.Environment.IsDevelopment())
//{
    app.UseSwagger();
    app.UseSwaggerUI();
//}
...
```

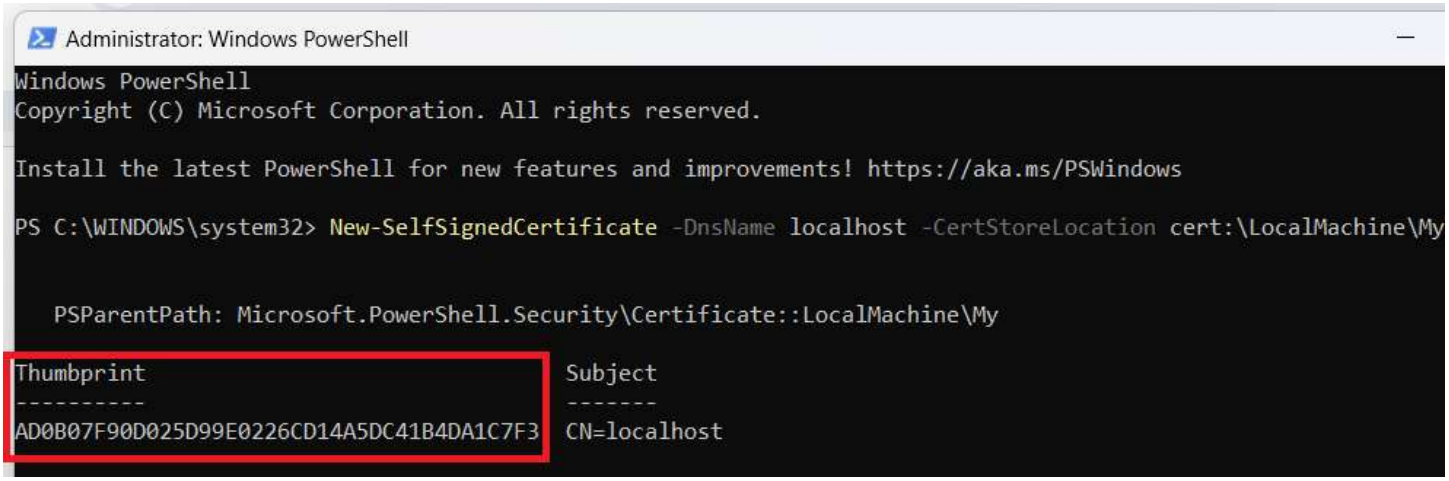
Now we can run the docker container and access to API documentation (Swagger)



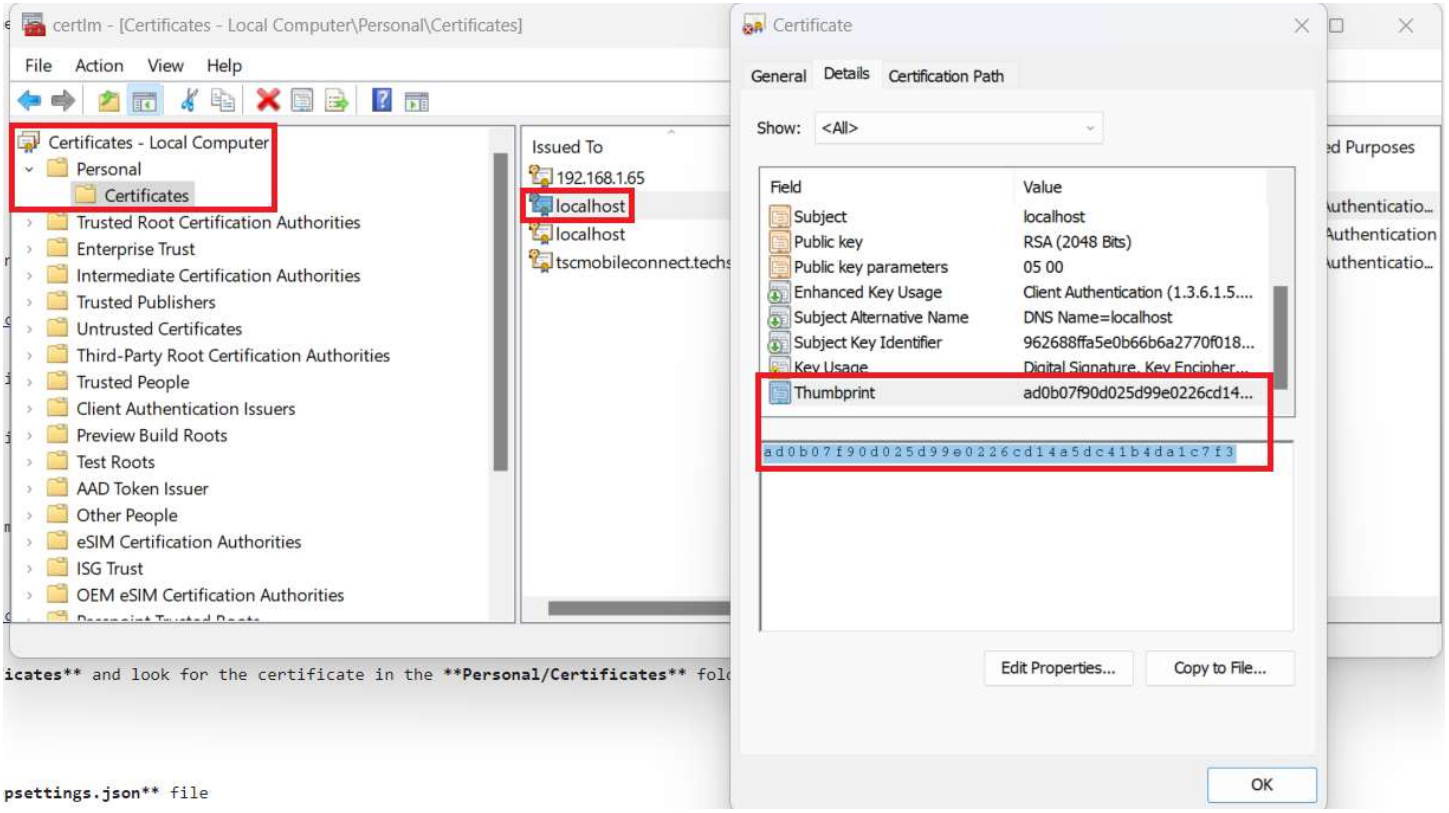
If we would like to run the application with **HTTPS** protocol we have to create a certificate

We create a certificate running in PowerShell this command

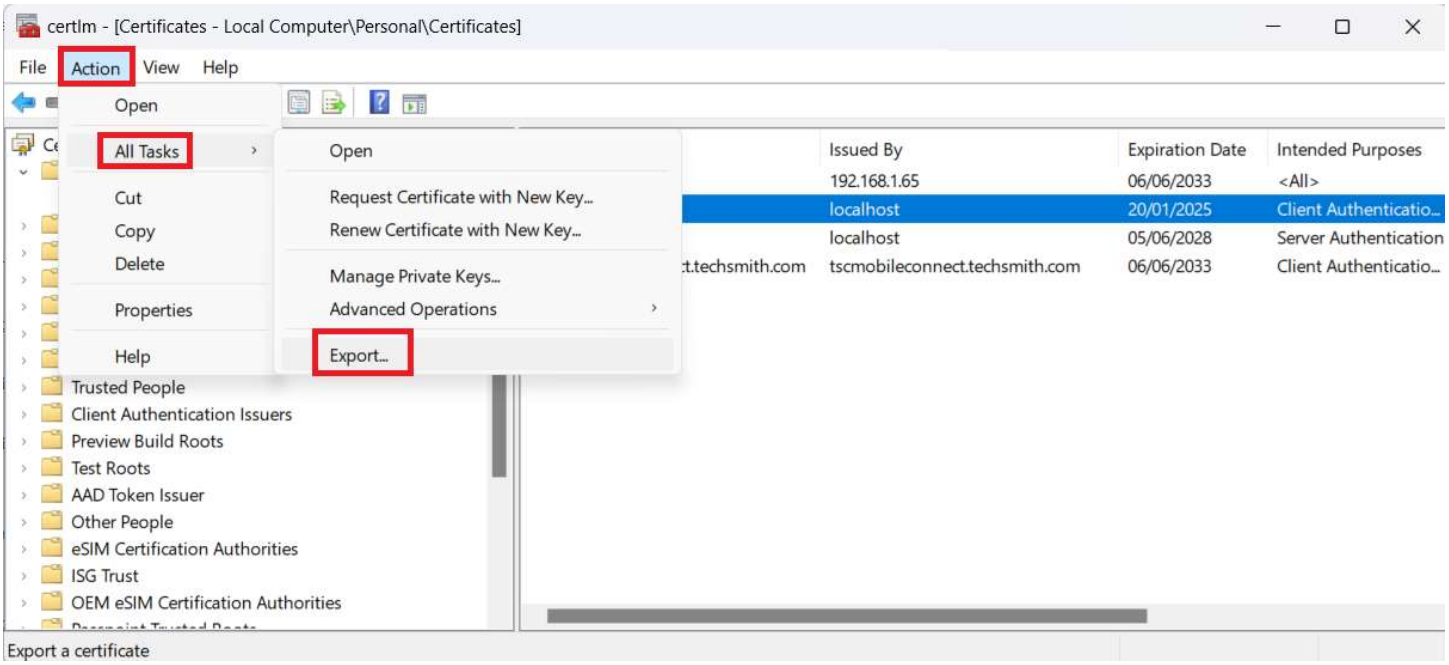
```
New-SelfSignedCertificate -DnsName localhost -CertStoreLocation cert:\LocalMachine\My
```

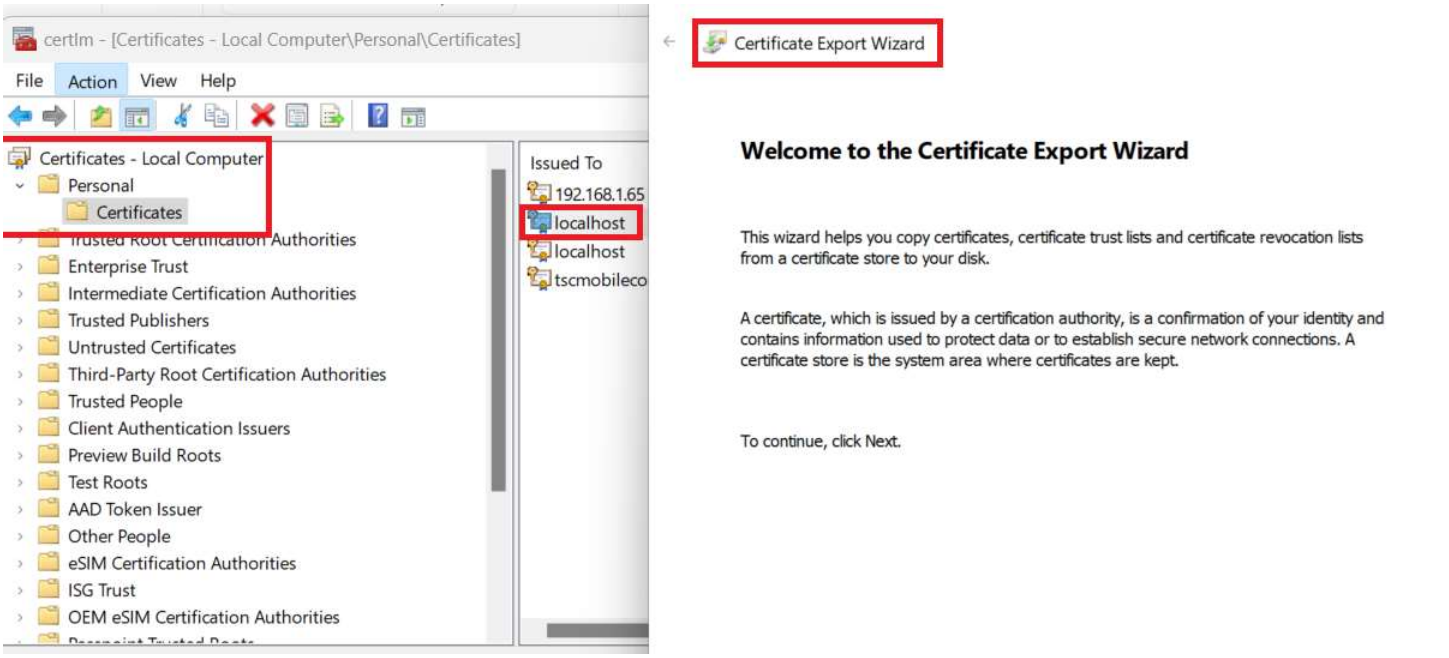


We open **Manage Computer Certificates** and look for the certificate in the **Personal/Certificates** folder

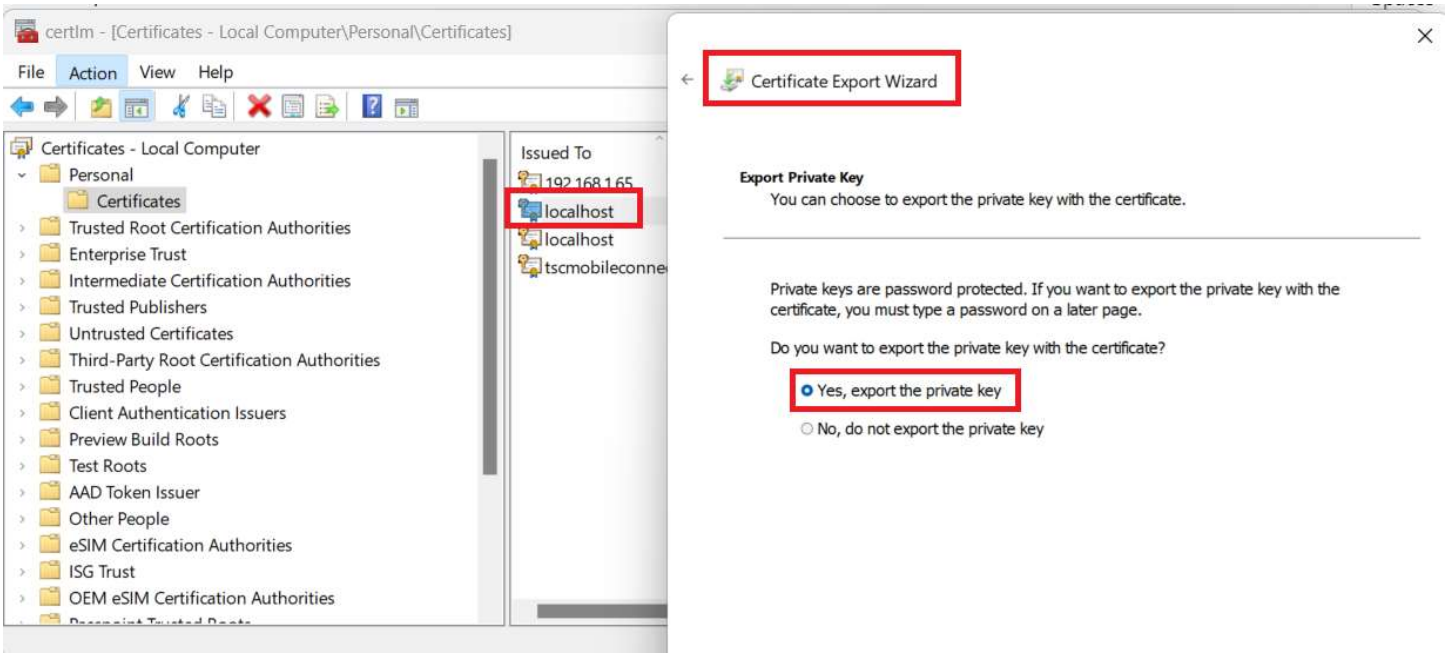


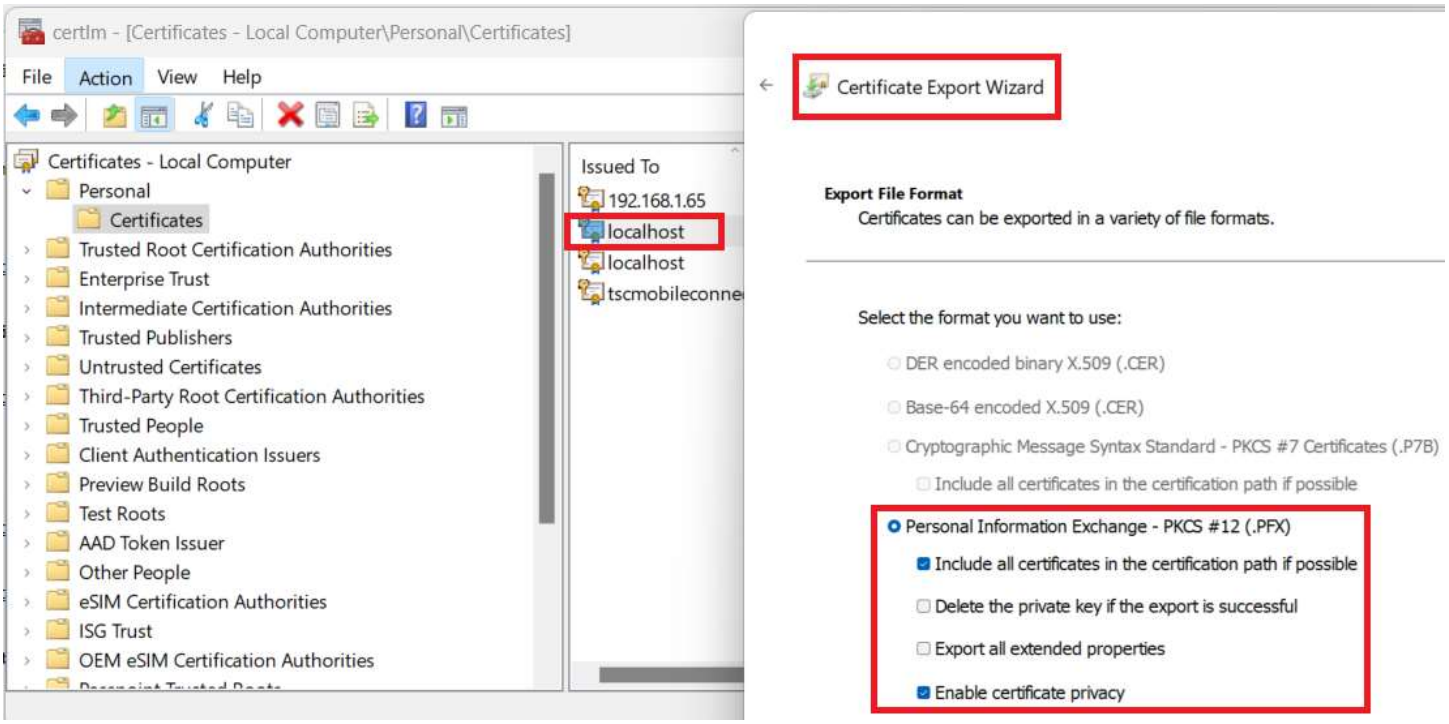
We select the certificate and we **export to PFX** file

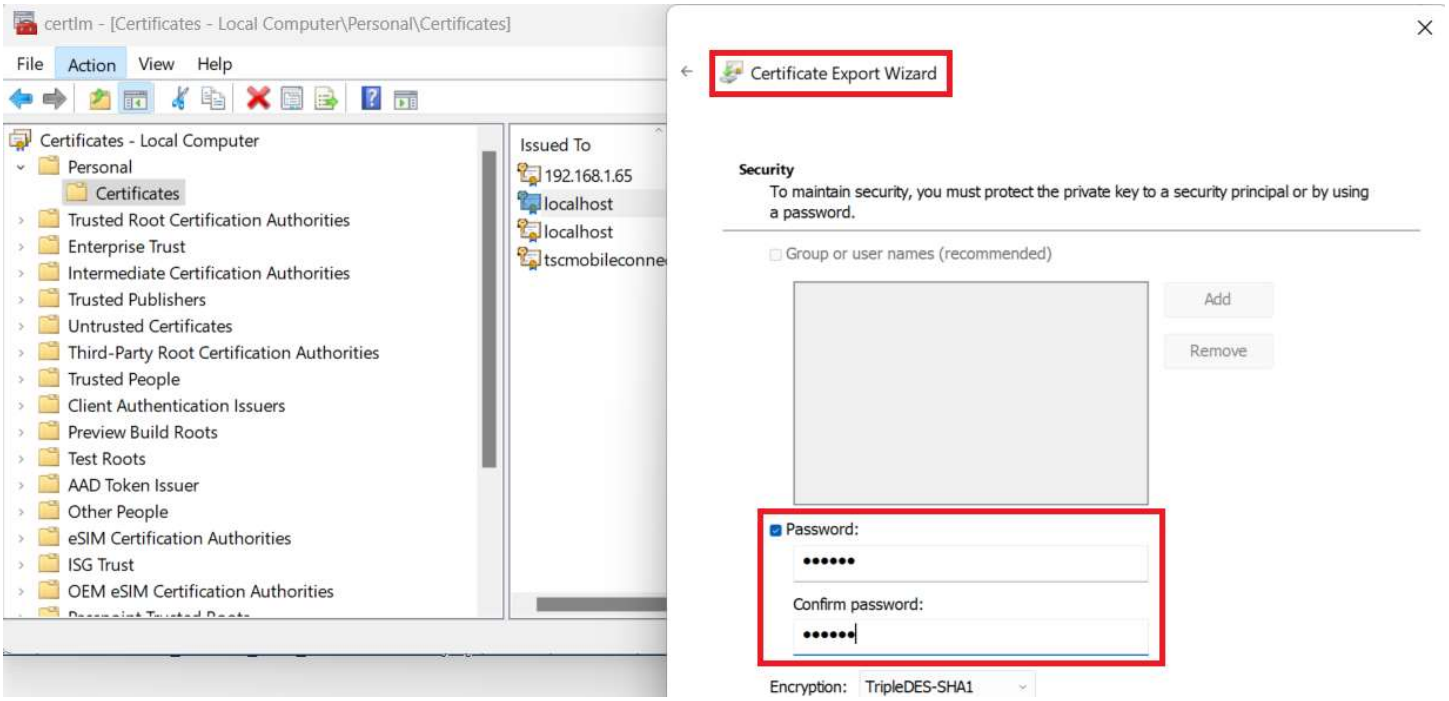




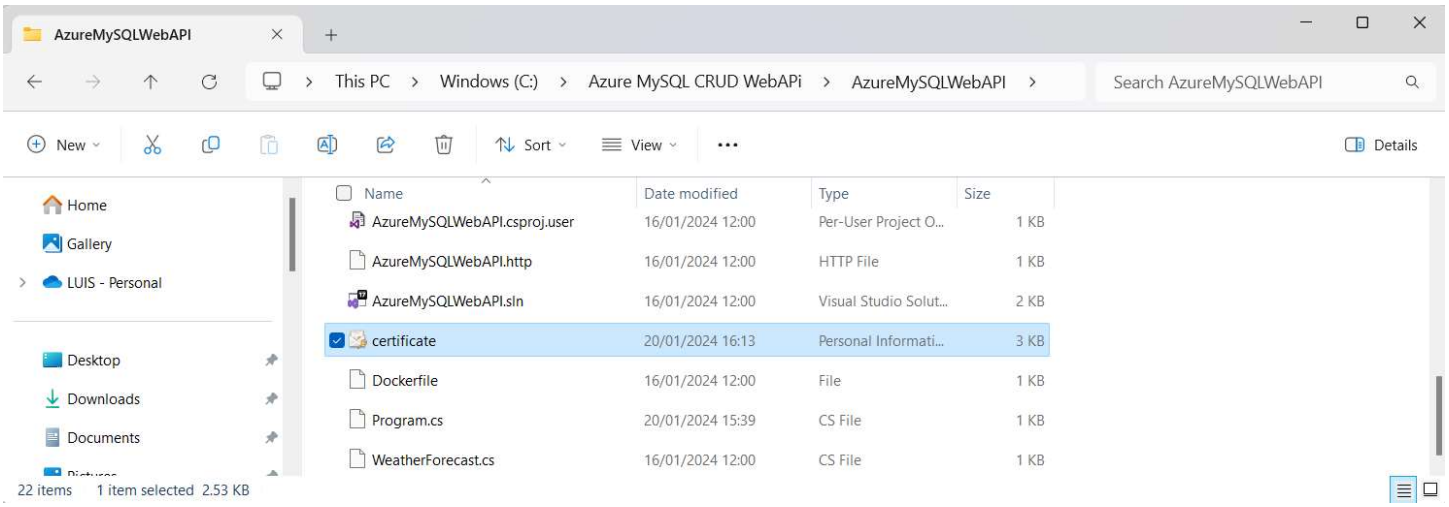
We select the option **Yes export the private key**


certlm - [Certificates - Local Computer\Personal\Certificates]
File Action View Help
Certificates - Local Computer
Personal
Certificates
Trusted Root Certification Authorities
Enterprise Trust
Intermediate Certification Authorities
Trusted Publishers
Untrusted Certificates
Third-Party Root Certification Authorities
Trusted People
Client Authentication Issuers
Preview Build Roots
Test Roots
AAD Token Issuer
Other People
eSIM Certification Authorities
ISG Trust
OEM eSIM Certification Authorities
Issued To
192.168.1.65
localhost
localhost
tscmobileconne
Certificate Export Wizard
Export Private Key
You can choose to export the private key with the certificate.
Private keys are password protected. If you want to export the private key with the certificate, you must type a password on a later page.
Do you want to export the private key with the certificate?
Yes, export the private key
No, do not export the private key

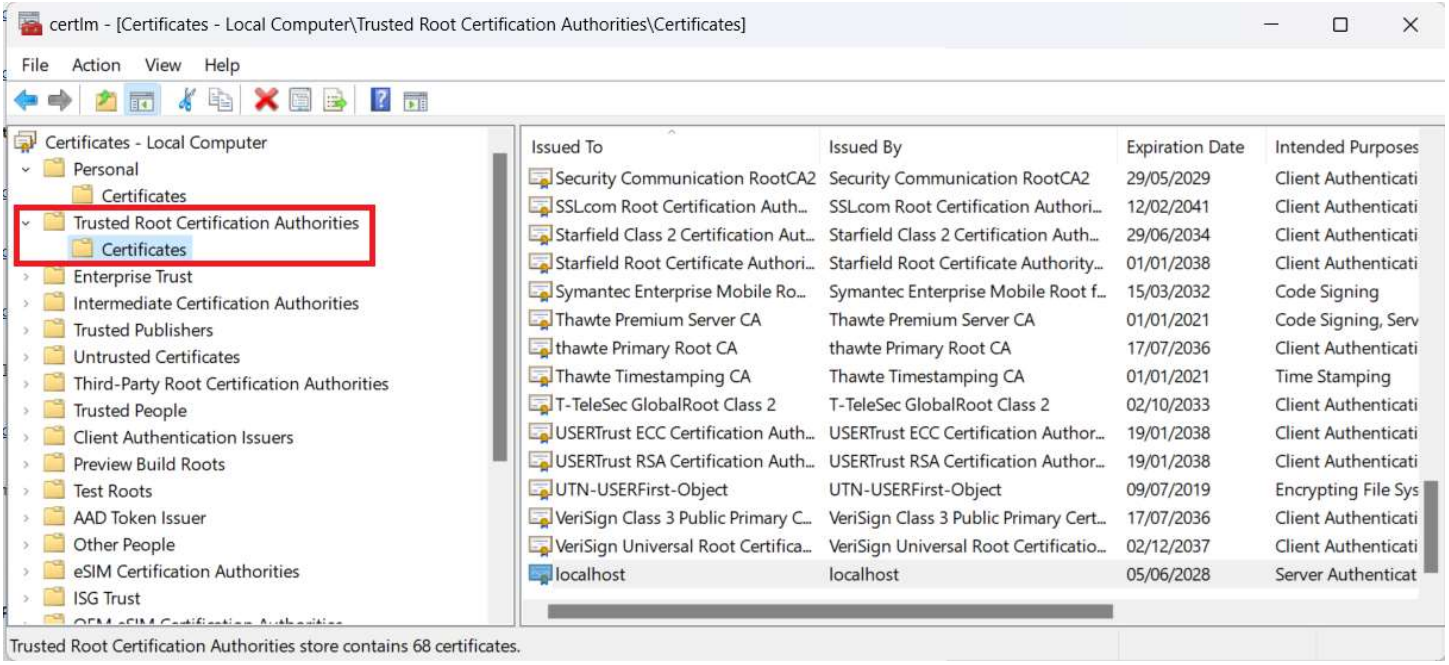

certlm - [Certificates - Local Computer\Personal\Certificates]
File Action View Help
Certificates - Local Computer
Personal
Certificates
Trusted Root Certification Authorities
Enterprise Trust
Intermediate Certification Authorities
Trusted Publishers
Untrusted Certificates
Third-Party Root Certification Authorities
Trusted People
Client Authentication Issuers
Preview Build Roots
Test Roots
AAD Token Issuer
Other People
eSIM Certification Authorities
ISG Trust
OEM eSIM Certification Authorities
Issued To
192.168.1.65
localhost
localhost
tscmobileconne
Certificate Export Wizard
Export File Format
Certificates can be exported in a variety of file formats.
Select the format you want to use:
DER encoded binary X.509 (.CER)
Base-64 encoded X.509 (.CER)
Cryptographic Message Syntax Standard - PKCS #7 Certificates (.P7B)
Include all certificates in the certification path if possible
Personal Information Exchange - PKCS #12 (.PFX)
Include all certificates in the certification path if possible
Delete the private key if the export is successful
Export all extended properties
Enable certificate privacy



We save the **PFX** file in our local application source code folder



We also copy the certificate from **Personal/Certificates** folder to **Trusted Root Certification Authorities/Certificates** folder



We first have to modify the **appsettings.json** file

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "server=mysqlserver1974.mysql.database.azure.com;database=mysqldataba
  },
  "Kestrel": {
    "Endpoints": {
      "Http": {
        "Url": "http://*:8080"
      },
      "Https": {
        "Url": "https://*:8081",
        "Certificate": {
          "Path": "certificate.pfx",
          "Password": "123456"
        }
      }
    }
  }
}
```

We also need to modify the **Program.cs** file commenting this line: //app.UseHttpsRedirection();

And also to modify the **Dockerfile** to copy the certificate adding this line:

```
# Copy the certificate file into the Docker image
COPY ["certificate.pfx", "."]
```

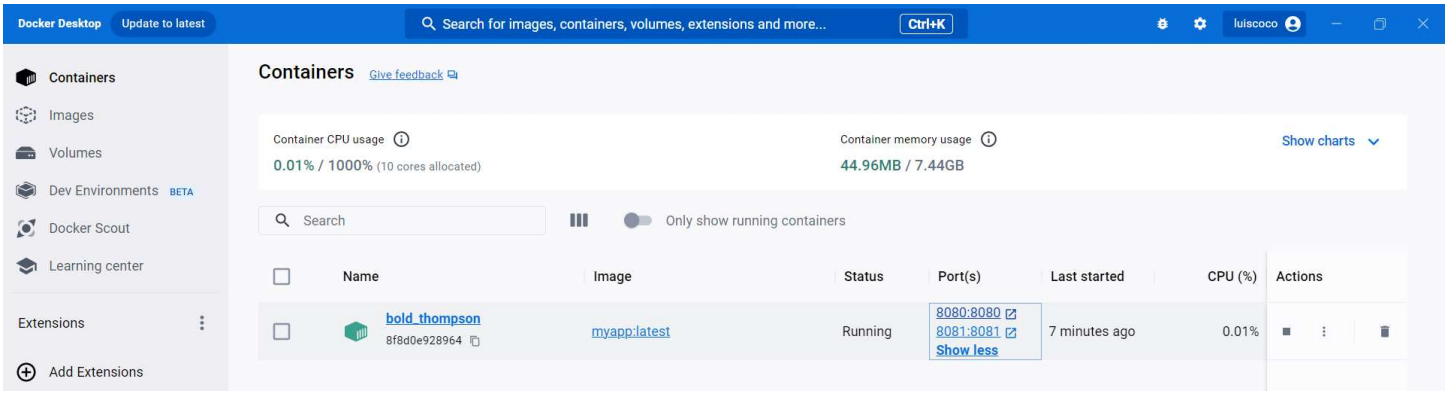
See the new Dockerfile

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER app
WORKDIR /app
EXPOSE 8080
EXPOSE 8081

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["AzureMySQLWebAPI.csproj", "."]
RUN dotnet restore "./././AzureMySQLWebAPI.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "./AzureMySQLWebAPI.csproj" -c $BUILD_CONFIGURATION -o /app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./AzureMySQLWebAPI.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseA

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
# Copy the certificate file into the Docker image
COPY ["certificate.pfx", "."]
ENTRYPOINT ["dotnet", "AzureMySQLWebAPI.dll"]
```

If we would like to access our application via **HTTP** or **HTTPS** we run the docker container with this command

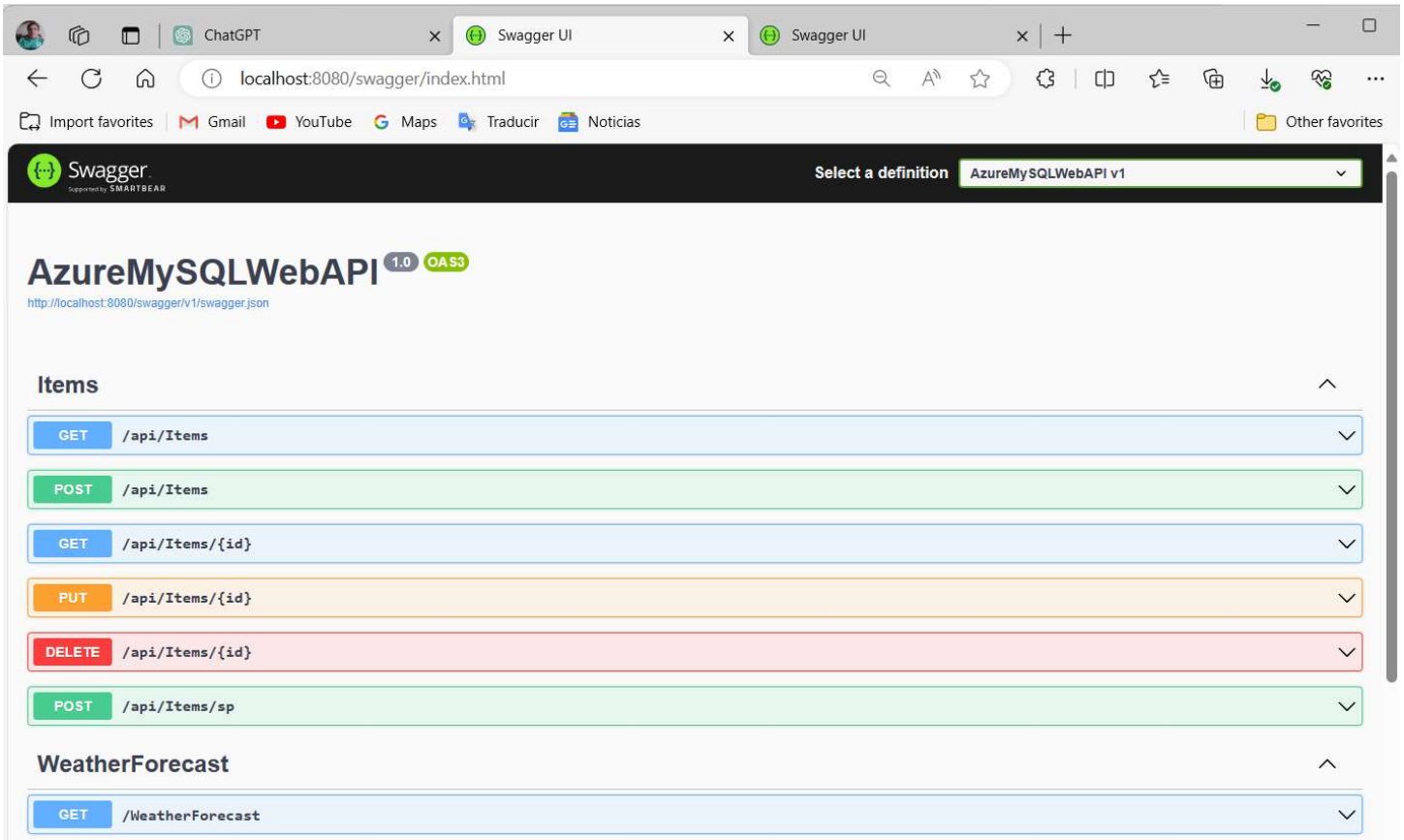
```
docker run -d -p 8080:8080 -p 8081:8081 myapp:latest
```

We verify in **Docker Desktop**

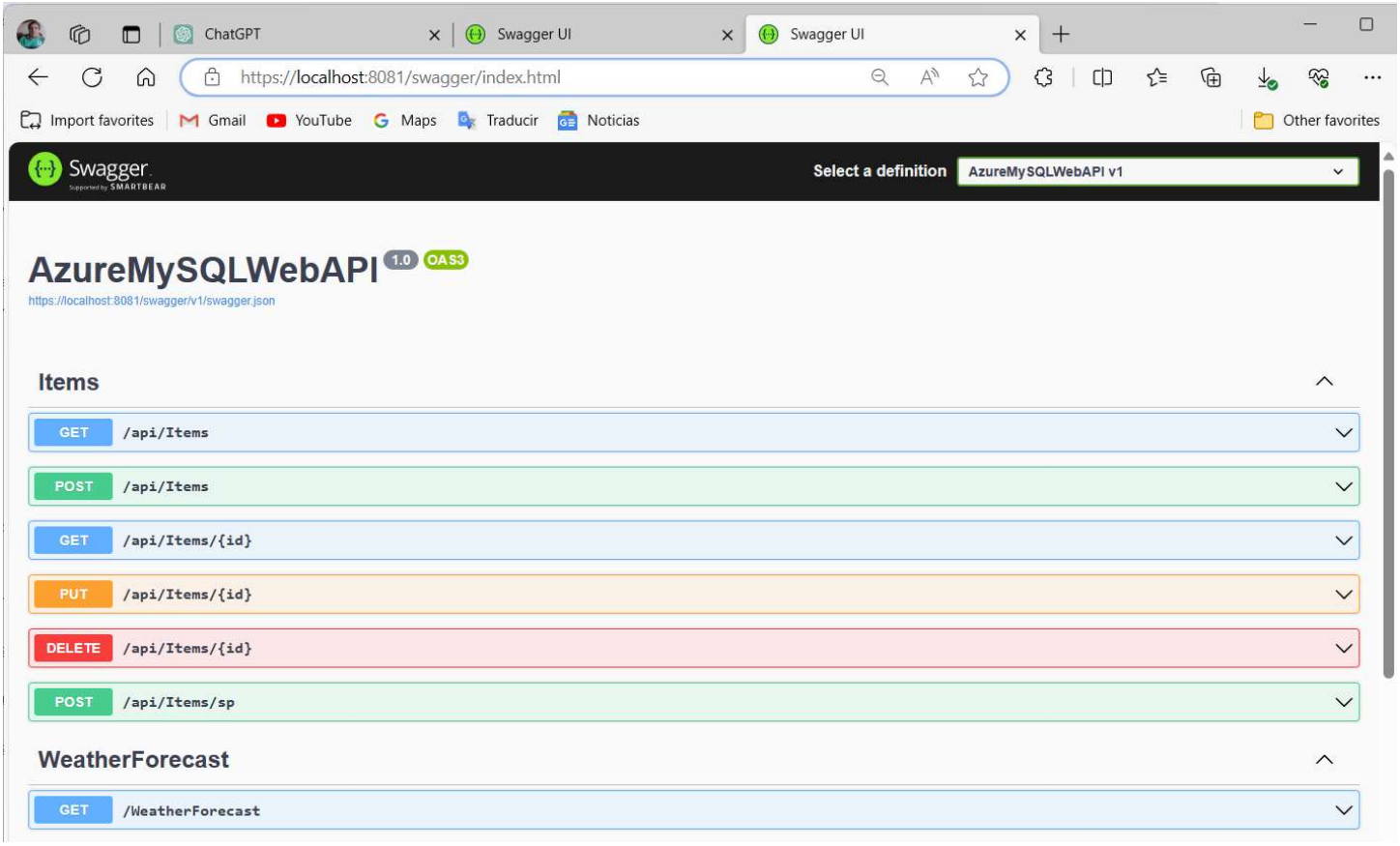


We verify the **application endpoints**

http://localhost:8080/swagger/index.html



https://localhost:8081/swagger/index.html



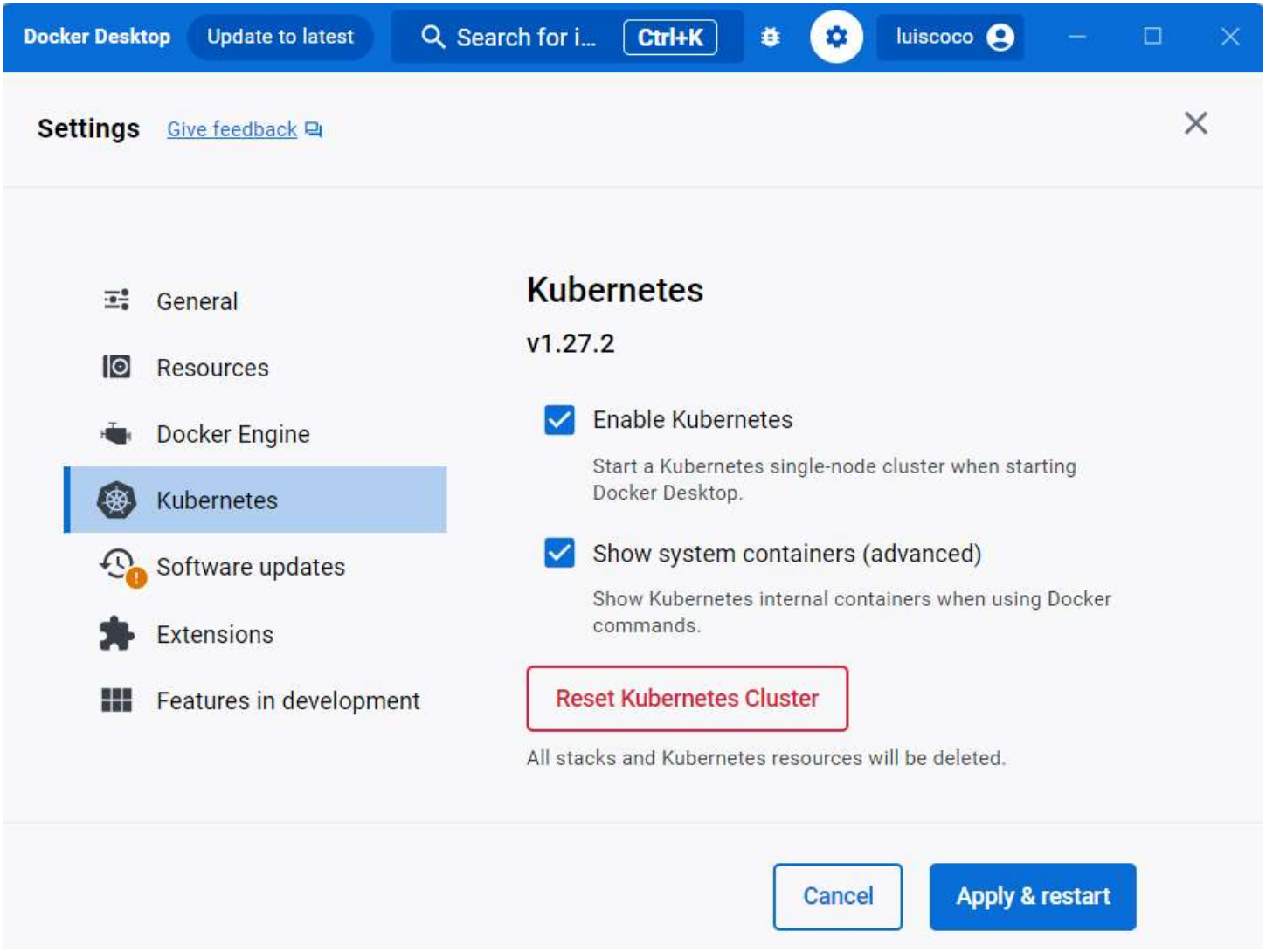
4. How to deploy the WebAPI Microservice to Kubernetes (in Docker Desktop)

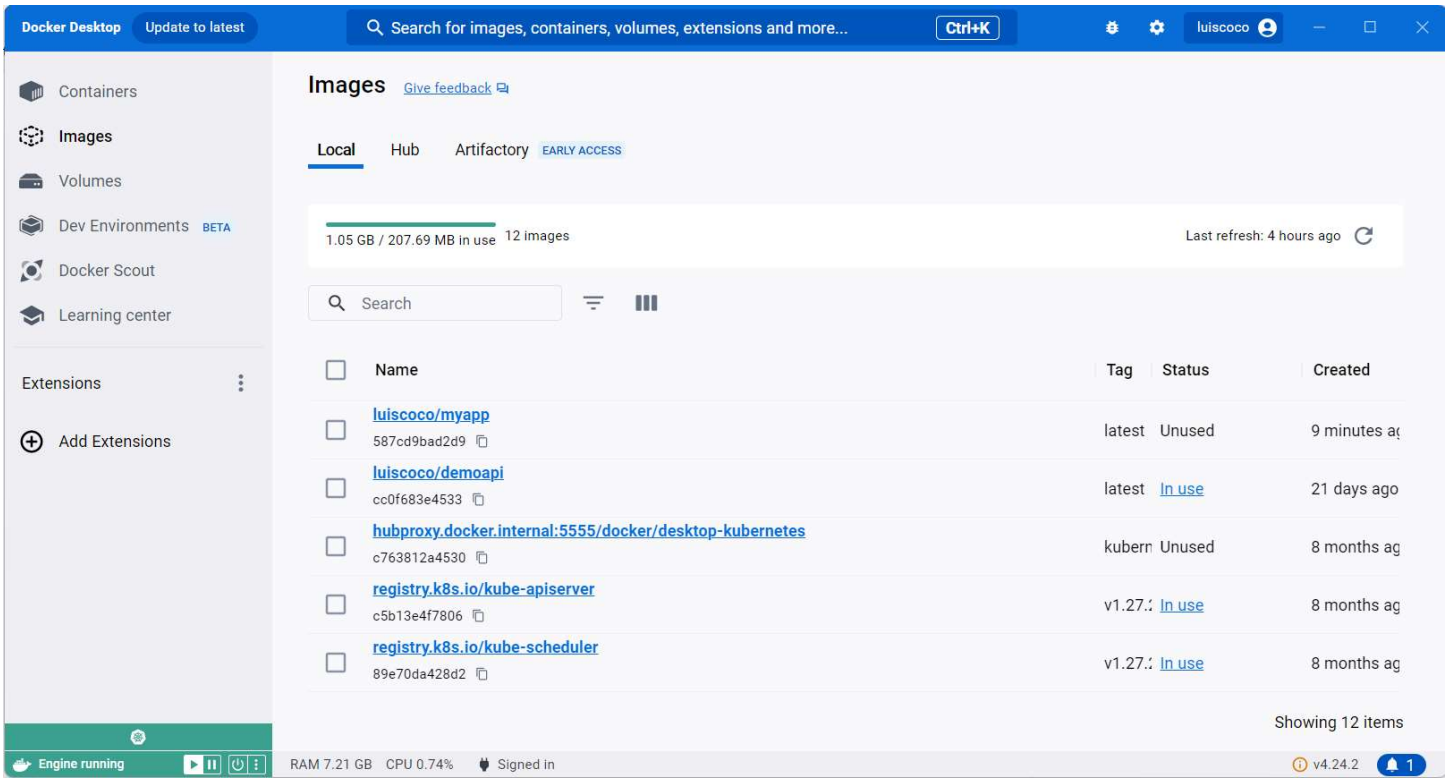
For more details about this section see the repo:
https://github.com/luiscoco/Kubernetes_Deploy_dotNET_8_Web_API

We enable **Kubernetes** in **Docker Desktop**

We run **Docker Desktop** and press on **Settings** button

We select **Enable Kubernetes** in the left hand side menu an press **Apply & Restart** button





Here are the general steps to deploy your .NET 8 Web API to Kubernetes:

- **Build** and **Push** the Docker image to the **Docker Hub registry/repo**

- Create Kubernetes **Deployment YAML file**. This file defines how your application is deployed in Kubernetes.
- Create Kubernetes **Service YAML file**. This file defines how your application is exposed, either within Kubernetes cluster or to the outside world.
- Apply the **YAML** files to your Kubernetes Cluster: use the command "kubectl apply" to create the resource defined in your YAML file in your Kubernetes cluster.

We start building and pushing the application Docker image to the Docker Hub registry/repo

```
docker build -t luiscoco/myapp:latest .
```

To verify we created the docker image run the command:

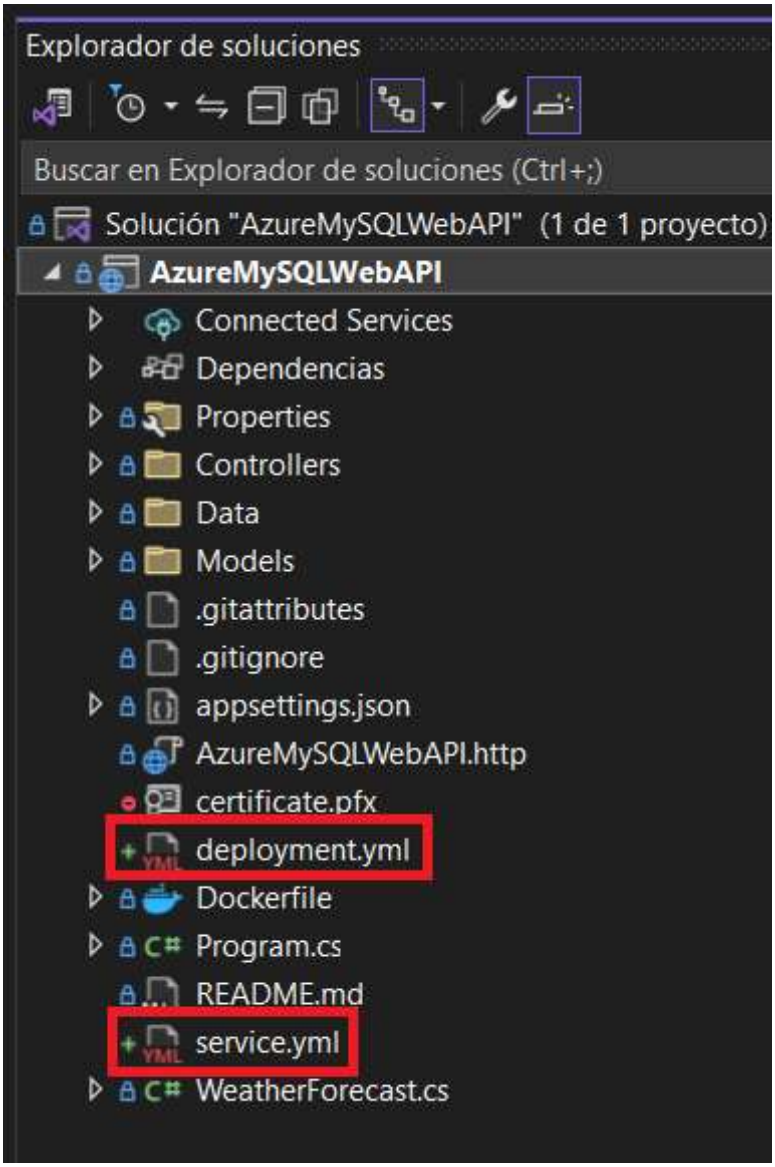
```
docker images
```

Then we use the docker push command to upload the image to the Docker Hub repository:

```
docker push luiscoco/myapp:latest
```

Note: run the "**docker login**" command if you have no access to Docker Hub repo

We create the deployment.yml and the service.yml files in our project



deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: luiscoco/myapp:latest
        ports:
        - containerPort: 8080
```

```
        - containerPort: 8081
        env:
        - name: ConnectionStrings__DefaultConnection
          value: server=mysqlserver1974.mysql.database.azure.com;database=mysqldatabase;user=a
        volumeMounts:
        - mountPath: /app/certificate.pfx
          name: cert-volume
          subPath: certificate.pfx
      volumes:
      - name: cert-volume
        secret:
          secretName: myapp-cert
```

service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: LoadBalancer
  selector:
    app: myapp
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 8080
    - name: https
      protocol: TCP
      port: 443
      targetPort: 8081
```

We set the current Kubernetes context to Docker Desktop Kubernetes with this command:

```
kubectl config use-context docker-desktop
```

In a typical Kubernetes deployment, the files used for configuration (like **certificates**) need to be accessible to the Kubernetes cluster.

The most secure and common approach is to use a **Kubernetes Secret**.

Since your certificate is a sensitive file, it should be managed as a Secret.

Here's how you can modify your deployment to use a Kubernetes Secret for your certificate:

We create a Kubernetes Secret that contains your certificate.

You can do this by running the following command in your terminal (make sure you're in the directory where **certificate.pfx** is located):

```
kubectl create secret generic myapp-cert --from-file=certificate.pfx
```

This command creates a new Secret named myapp-cert with the contents of certificate.pfx.

Now we can apply both kubernetes manifest files with these commands

```
kubectl apply -f deployment.yml
```

and

```
kubectl apply -f service.yml
```

We can use the command "**kubectl get services**" to check the **external IP** and port your application is accessible on, if using a LoadBalancer.

Verify the Deployment with the command:

```
kubectl get deployments
```

Verify the service status with the command:

```
kubectl get services
```

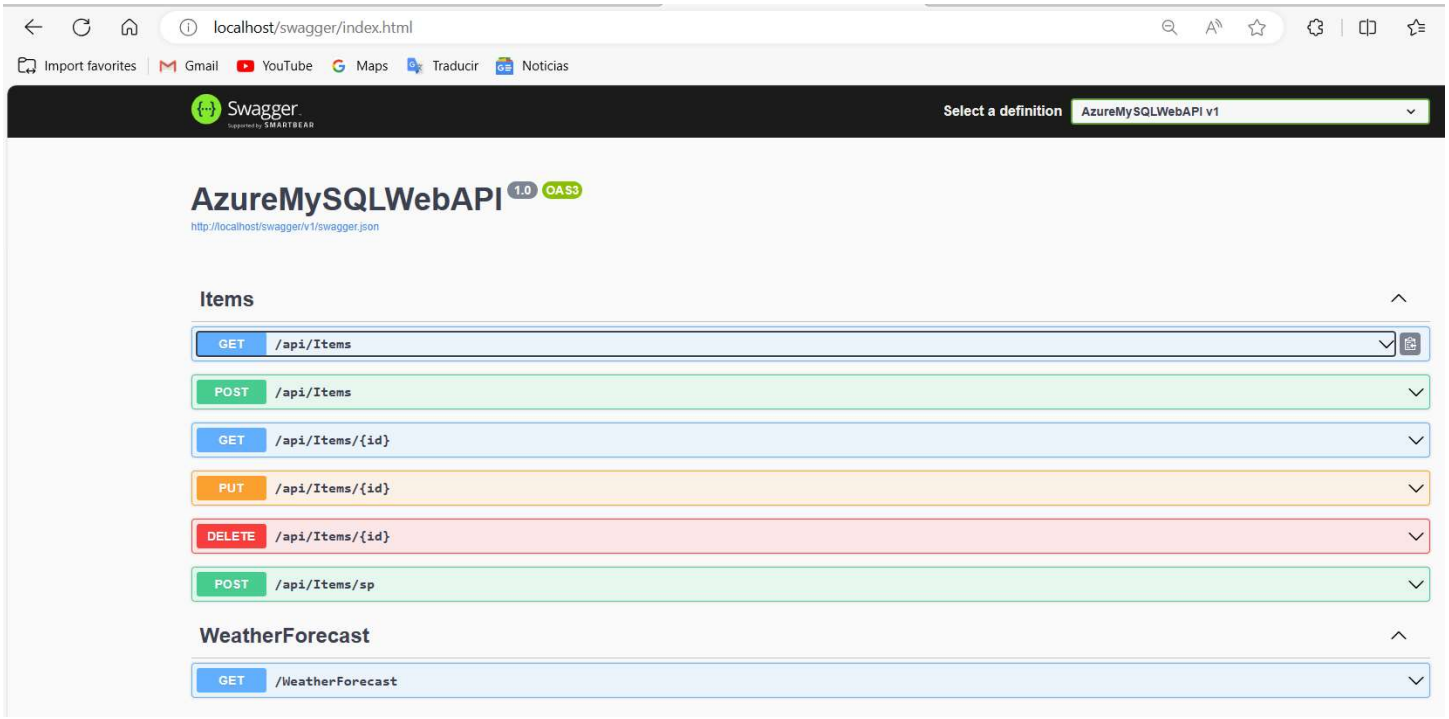
We can also verify the deployment with this command

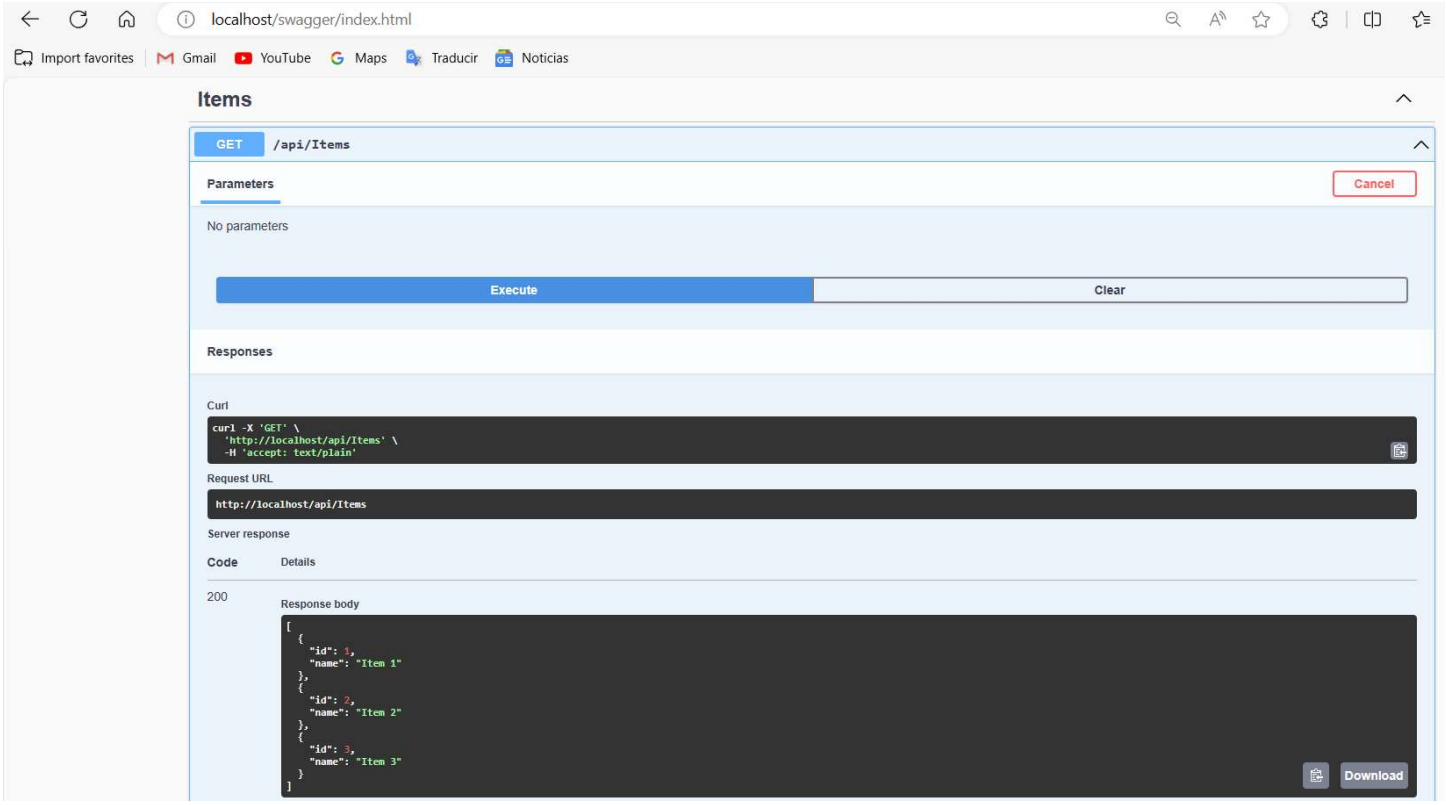
```
kubectl get all
```

```
PowerShell para desarrolladores
PowerShell para desarrolladores
PS C:\Azure MySQL CRUD WebAPi\AzureMySQLWebAPI> kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/myapp-deployment-797c7556b9-6mh7f    1/1     Running   0          79s
pod/myapp-deployment-797c7556b9-fnxq5    1/1     Running   0          79s

NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
service/kubernetes      ClusterIP      10.96.0.1       <none>        443/TCP                      35m
service/myapp-service   LoadBalancer   10.98.197.212   localhost     80:32210/TCP,443:32099/TCP   6s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp-deployment    2/2     2            2           80s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-deployment-797c7556b9    2         2         2       79s
```

We verify the application access endpoint

http://localhost/swagger/index.html




localhost/swagger/index.html
Import favorites
Gmail
YouTube
Maps
Traducir
Noticias
Items
GET
/api/Items
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost/api/Items' \
-H 'accept: text/plain'
Request URL
http://localhost/api/Items
Server response
Code
Details
200
Response body
[
{
"id": 1,
"name": "Item 1"
},
{
"id": 2,
"name": "Item 2"
},
{
"id": 3,
"name": "Item 3"
}
]
Download