

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/liscoco/MicroServices_dotNET8_CRUD_WebAPI-Azure-MySQL

1. Prerequisite

1.1. Create Azure MySQL instance

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

The screenshot shows the Microsoft Azure portal's 'Create a resource' interface. On the left, a sidebar lists various service categories like Home, Dashboard, All services, and Favorites. A red box highlights the '+ Create a resource' button. The main area has a search bar and a 'Get Started' section. Under 'Categories', 'Databases' is highlighted with a red box. Other categories shown include AI + Machine Learning, Analytics, Blockchain, Compute, Containers, Function App, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks, Microsoft Entra ID, Monitor, Advisor, Microsoft Defender for Cloud, and Cost Management + Billing. To the right, there are sections for Popular Azure services (Virtual machine, Web App, SQL Database, Function App, Key Vault, Data Factory, Template deployment, Logic App) and Popular Marketplace products (Windows Server 2019 Datacenter, Windows 11 Pro, version 21H2, Ubuntu Server 20.04 LTS, Ubuntu Server 22.04 LTS, Red Hat Enterprise Linux 7.4, Essentials 50K, MongoDB Atlas (pay-as-you-go), Standard).

We select Azure Database for MySQL

The screenshot shows the Microsoft Azure portal's 'Create a resource' interface. On the left, a sidebar lists various service categories like Home, Dashboard, All services, etc. A red box highlights the 'Create a resource' button at the top. The main area shows a search bar and a 'Popular Azure services' section. Under 'Categories', 'Databases' is selected and highlighted with a red box. Within the 'Databases' category, 'Azure Database for MySQL' is also highlighted with a red box. Other options listed include SQL Database, Azure SQL, Azure Cosmos DB, Azure Synapse Analytics, Azure Database for PostgreSQL, Azure SQL Managed Instance, and SQL server (logical server). To the right, there's a 'Popular Marketplace products' section.

We select create Flexible server

The screenshot shows the 'Select Azure Database for MySQL deployment option' page. The left sidebar is identical to the previous one. The main content area has a heading 'How do you plan to use the service?'. It shows two options: 'Flexible server' and 'Wordpress + MySQL Flexible server'. The 'Flexible server' option is highlighted with a red box. Both options have a 'Create' button, which is also highlighted with a red box. Below each option is a 'Learn More' link.

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

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+/-)



- [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](#)
- [Cost Management + Billing](#)

Home > Create a resource > Select Azure Database for MySQL deployment option >

Flexible server

Microsoft

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](#)

Compute + storage

Region * [West Europe](#)

MySQL version * [8.0](#)

Workload type [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

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	

[Review + create](#) [Next : Networking >](#)

Microsoft Azure

Home > Create a resource > Select Azure Database for MySQL deployment option > Flexible server >

Compute + storage

Compute resources are pre-allocated and billed per hour based on vCores configured.
Note that high availability and read replicas is supported for only General purpose and Business critical tiers.

Compute tier

- Burstable (1-20 vCores)** - Best for workloads that don't need the full CPU continuously
- General Purpose (2-96 vCores)** - Balanced configuration for most common workloads
- Business Critical (2-96 vCores)** - Best for Tier 1 workloads that require optimized performance

Compute size

Standard_B1s (1 vCore, 1 GiB memory, 400 max iops)

Storage

The storage you provision is the amount of storage capacity available to your flexible server and is billed GiB/month.
Note that storage cannot be scaled down once the server is created.

Storage size (in GiB) *

IOPS **Auto scale IOPS** Pre-provisioned IOPS

Storage Auto-growth

High availability

Same zone and zone redundant high availability provide additional server resilience in the event of a failure.

Enable high availability

Backups

Configure automatic server backups that can be used to restore your server to a point-in-time. [Learn more](#)

Backup retention period (in days)

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 = 2.74

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](#)

Bandwidth

For outbound data transfer across services in different regions will incur additional charges. Any inbound data transfer is free. [Learn more](#)

Estimated total USD 10.00/month

Prices reflects an estimates only. [View Azure pricing calculator](#). Final charges will appear in your local currency in cost analysis and billing views.

Save

Admin username: adminmysql

Password: LuiscocoXXXXXXXXXXXX

Microsoft Azure

Home > Create a resource > Select Azure Database for MySQL deployment option >

Flexible server

Microsoft

Important Server names, networking connectivity method, zone redundant HA and backup redundancy cannot be changed after server is created. Review these options carefully before provisioning.

Important Changing Basic options may reset selections you have made. Review all options prior to creating the resource.

Authentication

Azure Active Directory (Azure AD) is now Microsoft Entra ID. [Learn more](#)

Select the authentication methods you would like to support for accessing this MySQL server. MySQL password authentication allows you to create and use a ROLES (usernames) and use a password to authenticate. Enabling Microsoft Entra authentication allows you to create ROLES based on your Microsoft Entra accounts and generate an authentication token with which to authenticate. [Learn more](#)

Authentication method

- MySQL authentication only
- Microsoft Entra authentication only
- MySQL and Microsoft Entra authentication

Admin username *

Password *

Confirm password *

Estimated total USD 10.00/month

Prices reflects an estimates only. [View Azure pricing calculator](#). Final charges will appear in your local currency in cost analysis and billing views.

Review + create **Next : Networking >**

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

Networking

Configure networking access and security for your server.

Network connectivity

Connectivity method: Public access (allowed IP addresses) and Private endpoint Private access (VNet Integration)

Public access: Allow public access to this resource through the internet using a public IP address

Firewall rules

Inbound connections from the IP addresses specified below will be allowed to port 3306 on this server. [Learn more](#)

Allow public access from any Azure service within Azure to this server

+ Add current client IP address (88.1.112.152) + Add 0.0.0 - 255.255.255.255

Firewall rule name	Start IP address	End IP address
ClientIPAddress_2024-1-20_12-47-0	88.1.112.152	88.1.112.152
Firewall rule name	Start IP address	End IP address

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](#)
- Bandwidth
For outbound data transfer across services in different regions will incur additional charges. Any inbound data transfer is free. [Learn more](#)

Estimated total USD 10.00/month

Prices reflect an estimates only. [View Azure pricing calculator](#).

Firewall rules

Inbound connections from the IP addresses specified below will be allowed to port 3306 on this server. [Learn more](#)

Allow public access from any Azure service within Azure to this server

+ Add current client IP address (88.1.112.152) + Add 0.0.0 - 255.255.255.255

Firewall rule name	Start IP address	End IP address
ClientIPAddress_2024-1-20_12-47-0	88.1.112.152	88.1.112.152
Firewall rule name	Start IP address	End IP address

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](#)
- Bandwidth

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

The screenshot shows the Azure portal interface for managing a MySQL flexible server. The left sidebar includes options like Home, Dashboard, All services, Favorites, and various Azure services. The main content area is titled 'mysqlserver1974 | Connect' and displays 'Azure Database for MySQL flexible server'. Under 'Connection details', the connection string is shown as:

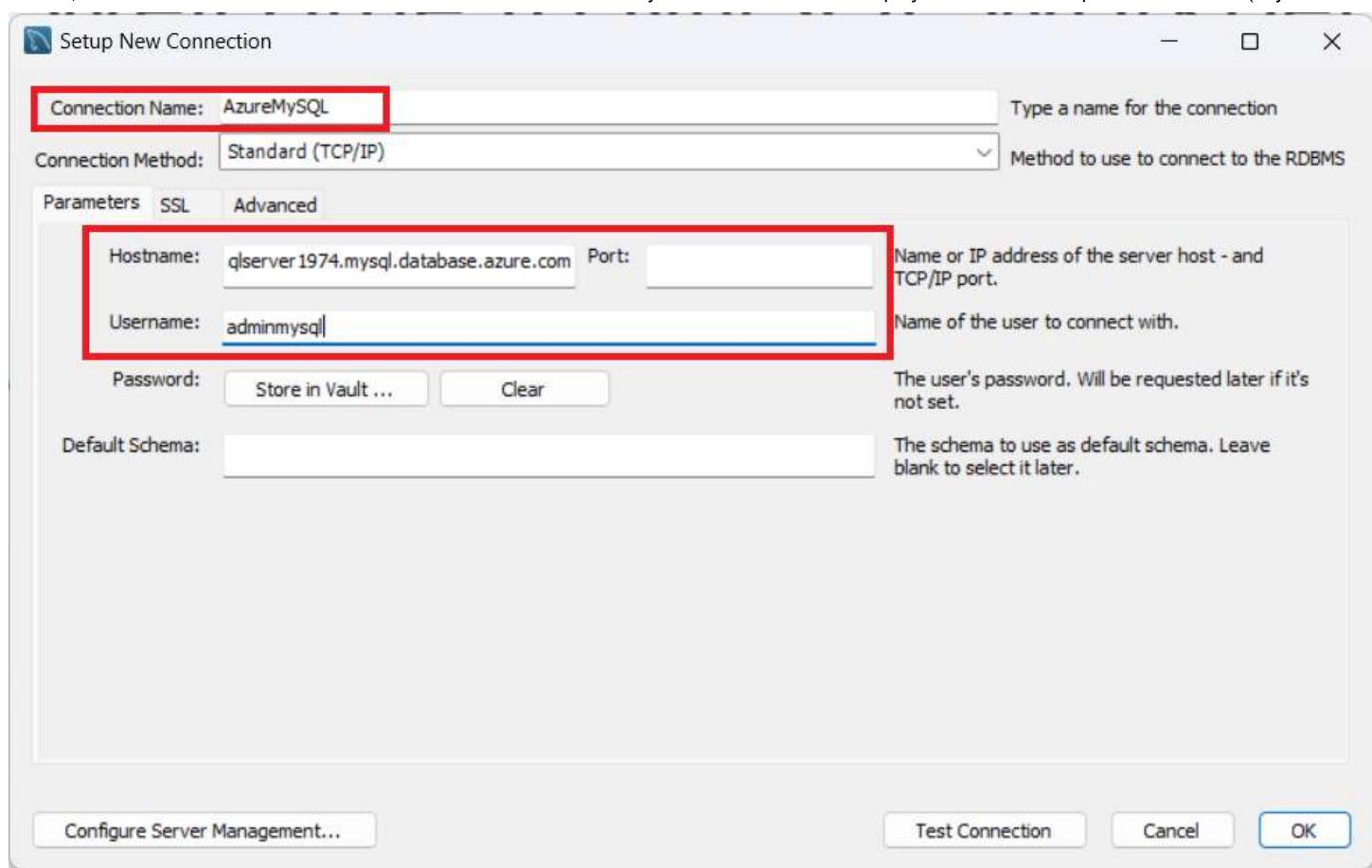
```
hostname=mysqlserver1974.mysql.database.azure.com
username=admin@mysql
password=(your-password)
ssl-mode=require
```

1.2. Run MySQL Workbench and create new database

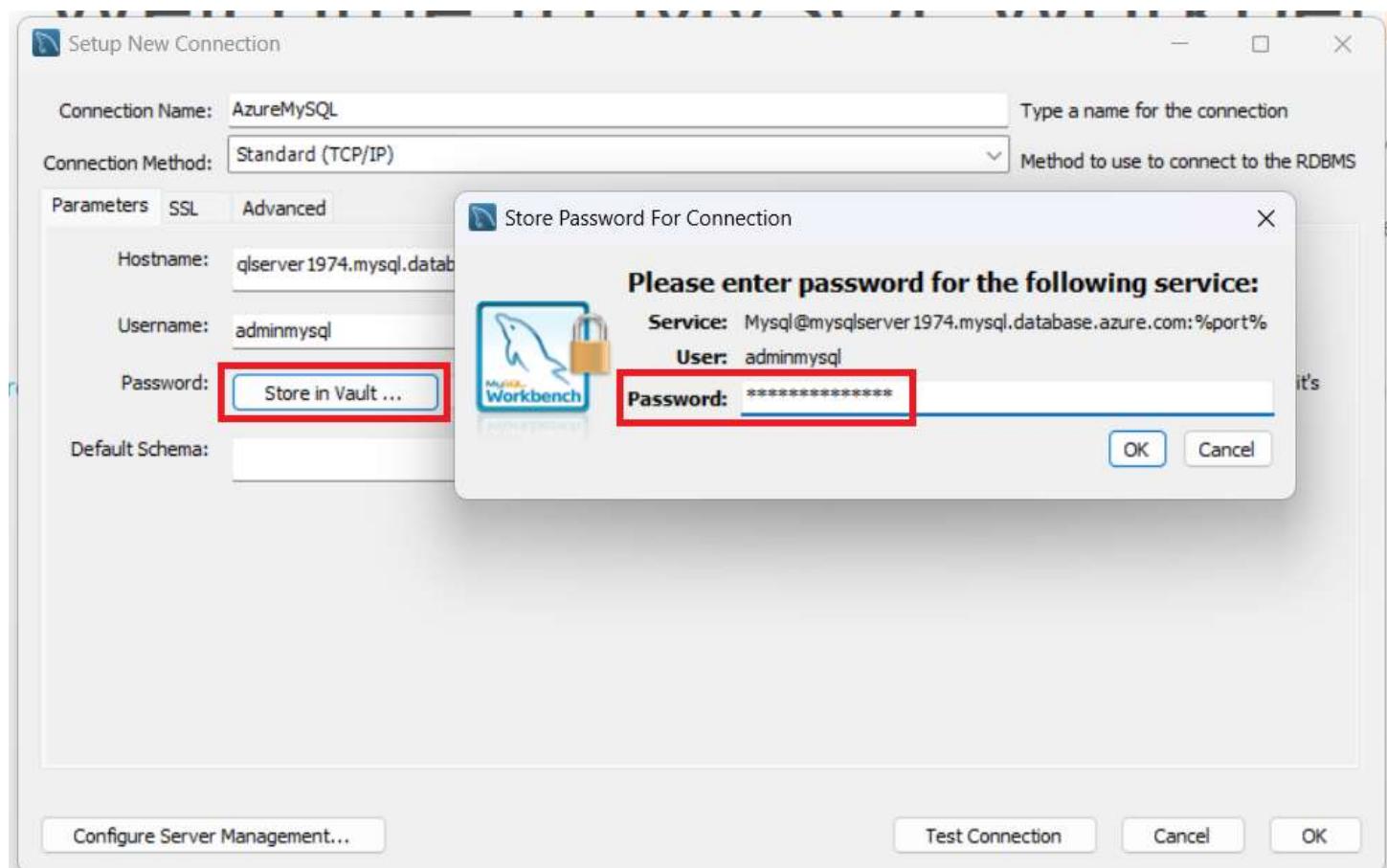
We run MySQL Workbench and we create a new connection

The screenshot shows the MySQL Workbench application window. The top menu bar includes File, Edit, View, Database, Tools, Scripting, and Help. The main area is titled 'Welcome to MySQL Workbench'. Below it, there's a brief description of what MySQL Workbench is. At the bottom, there are links to 'Browse Documentation >', 'Read the Blog >', and 'Discuss on the Forums >'. The 'MySQL Connections' section is highlighted, showing a single connection entry: 'Local instance MySQL80' with a user 'root' and host 'localhost:3306'.

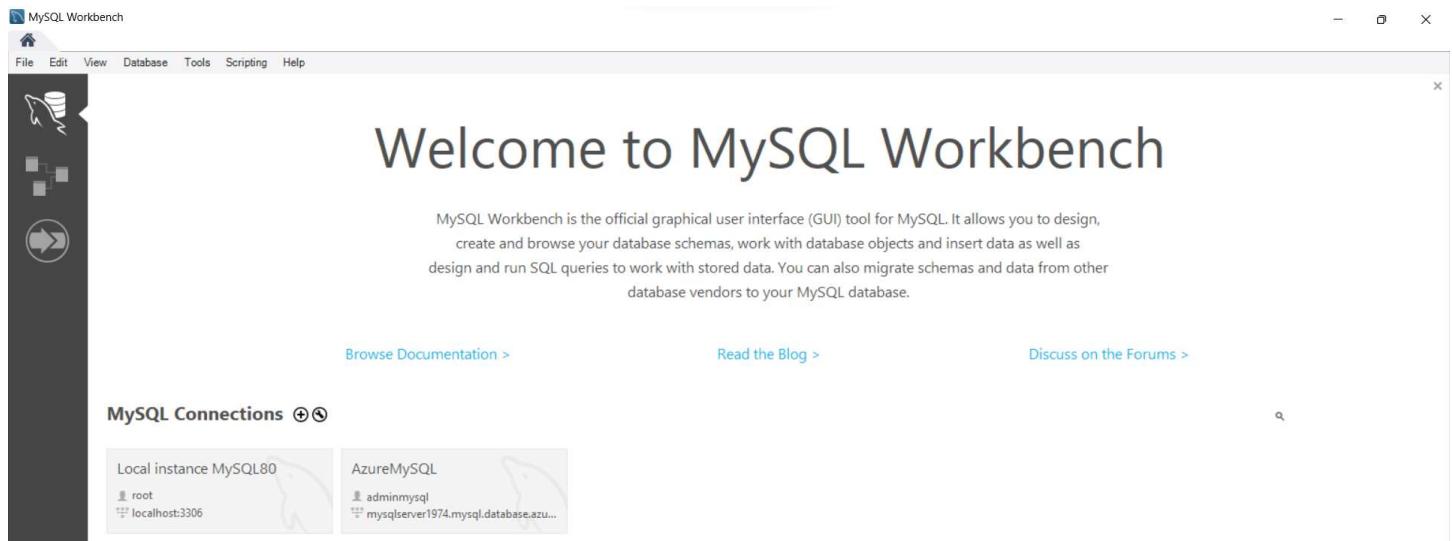
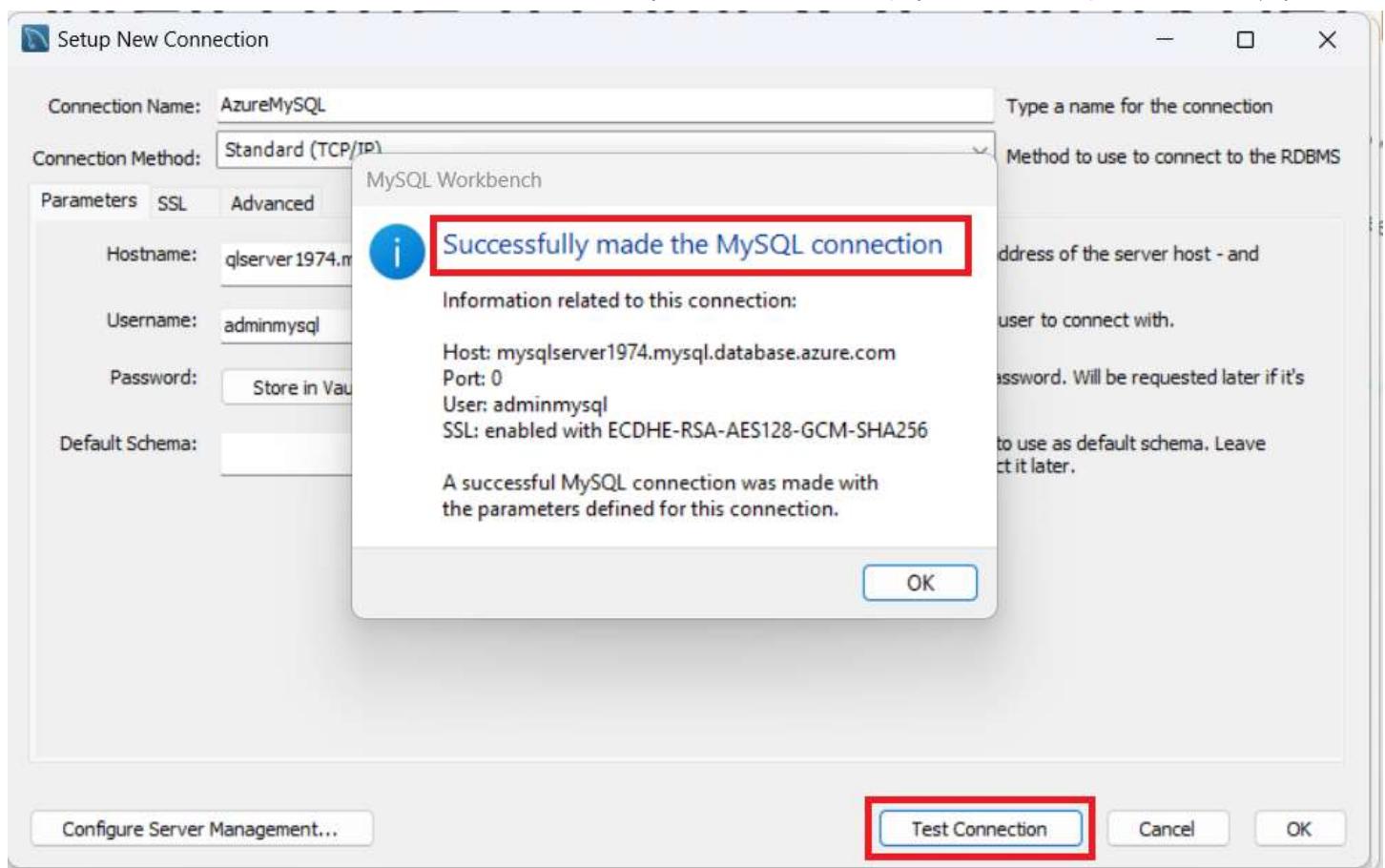
We input the new connection data



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

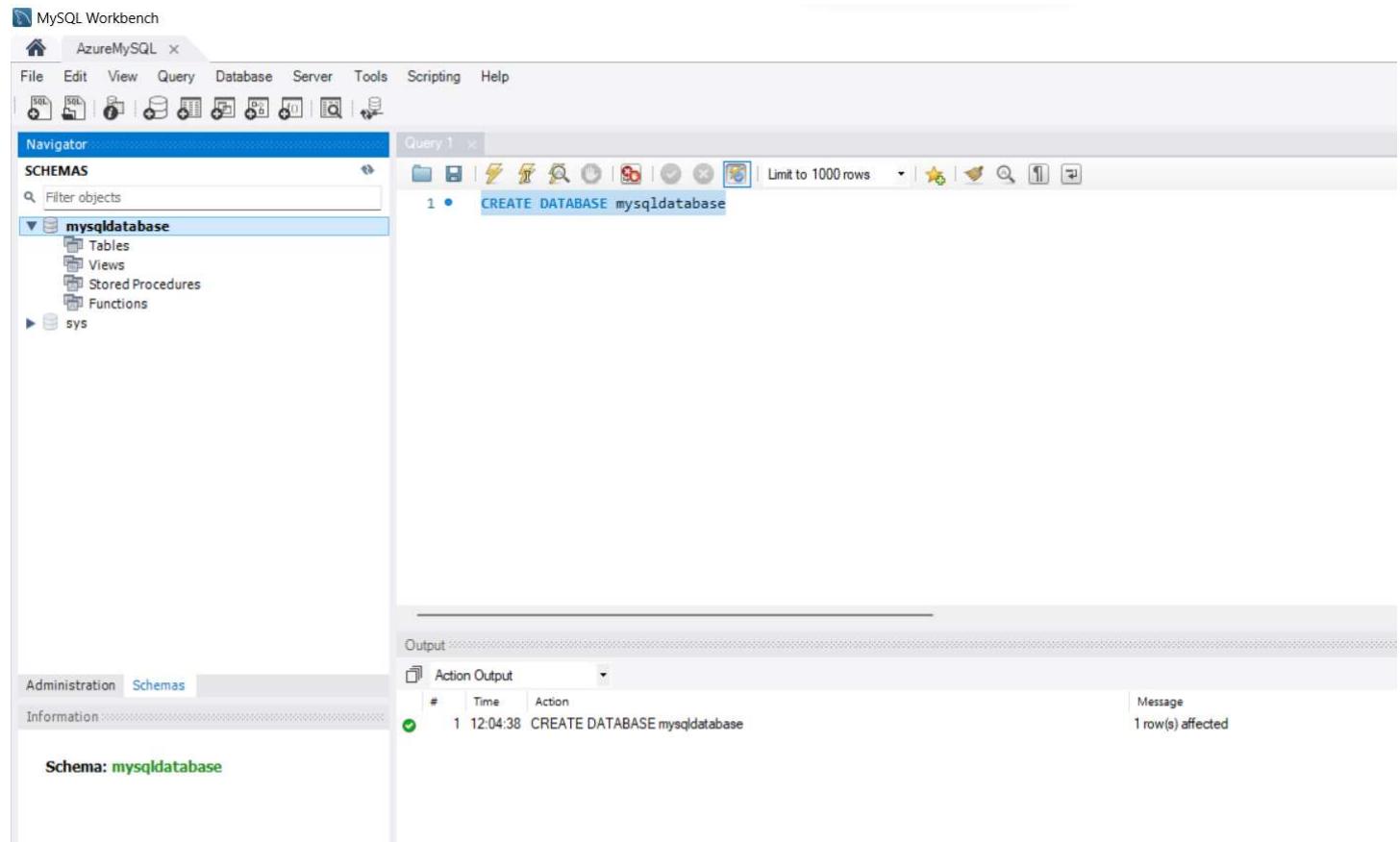


Now we can test the connection pressing on the **Test connection** button



We create a database running this command

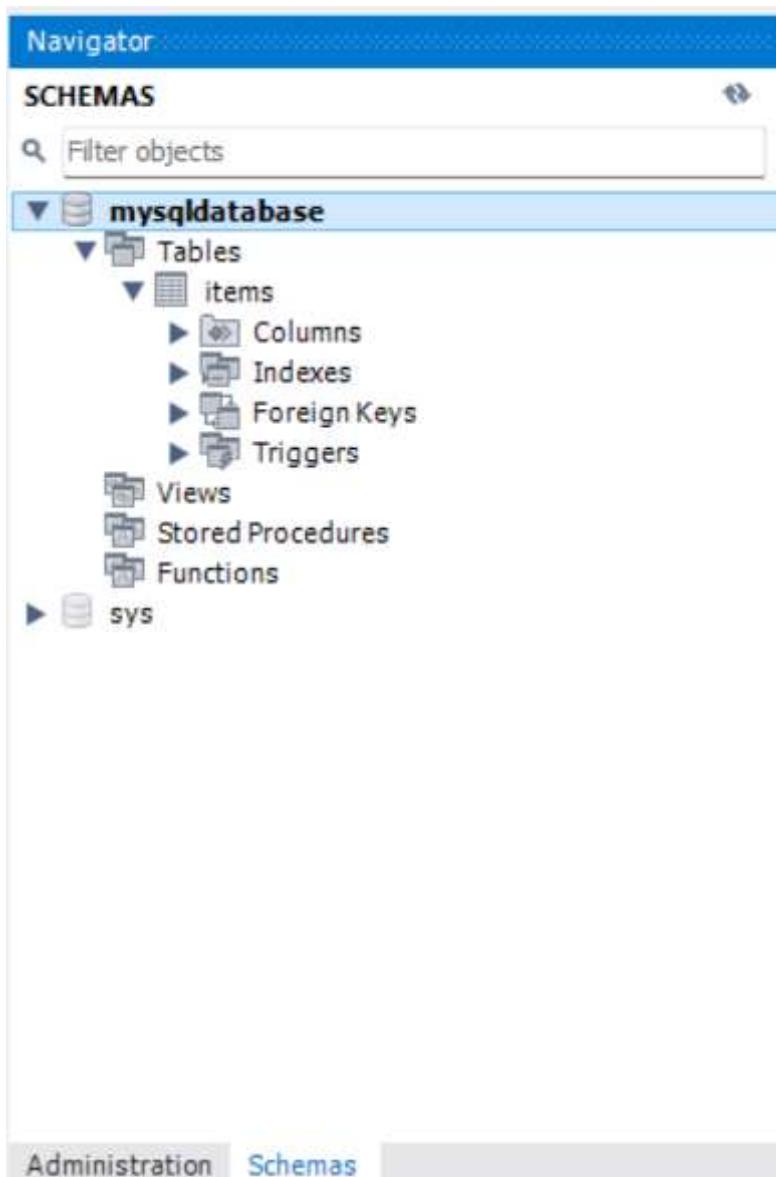
```
'''sql CREATE DATABASE mysqldatabase'''
```



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

The screenshot shows the MySQL Workbench interface with the 'Administration' tab selected. In the center, the results of a query are displayed in a grid:

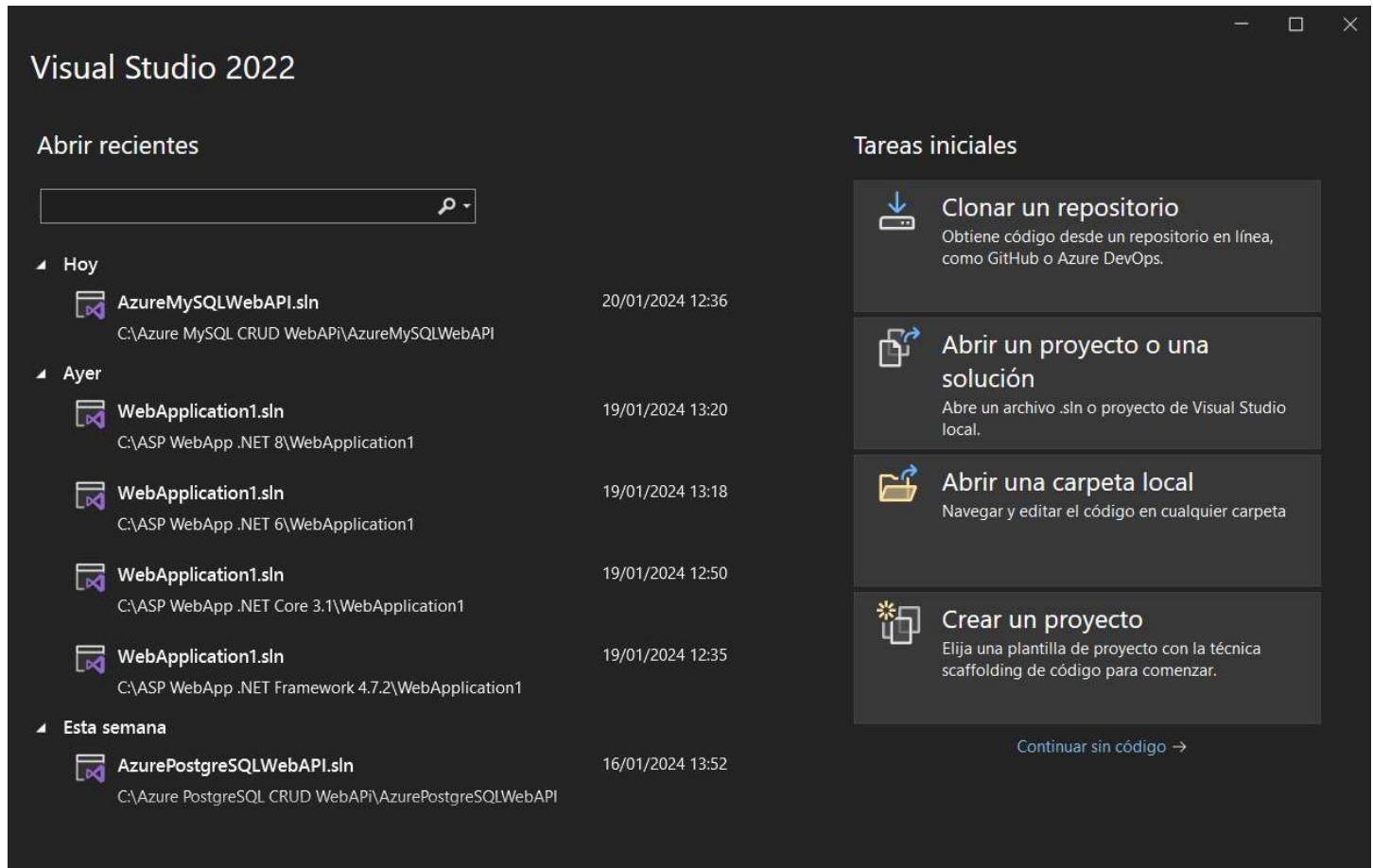
```
1 • |SELECT * FROM mysqldatabase.items;
```

1	Item 1	
2	Item 2	
3	Item 3	
*	NULL	NULL

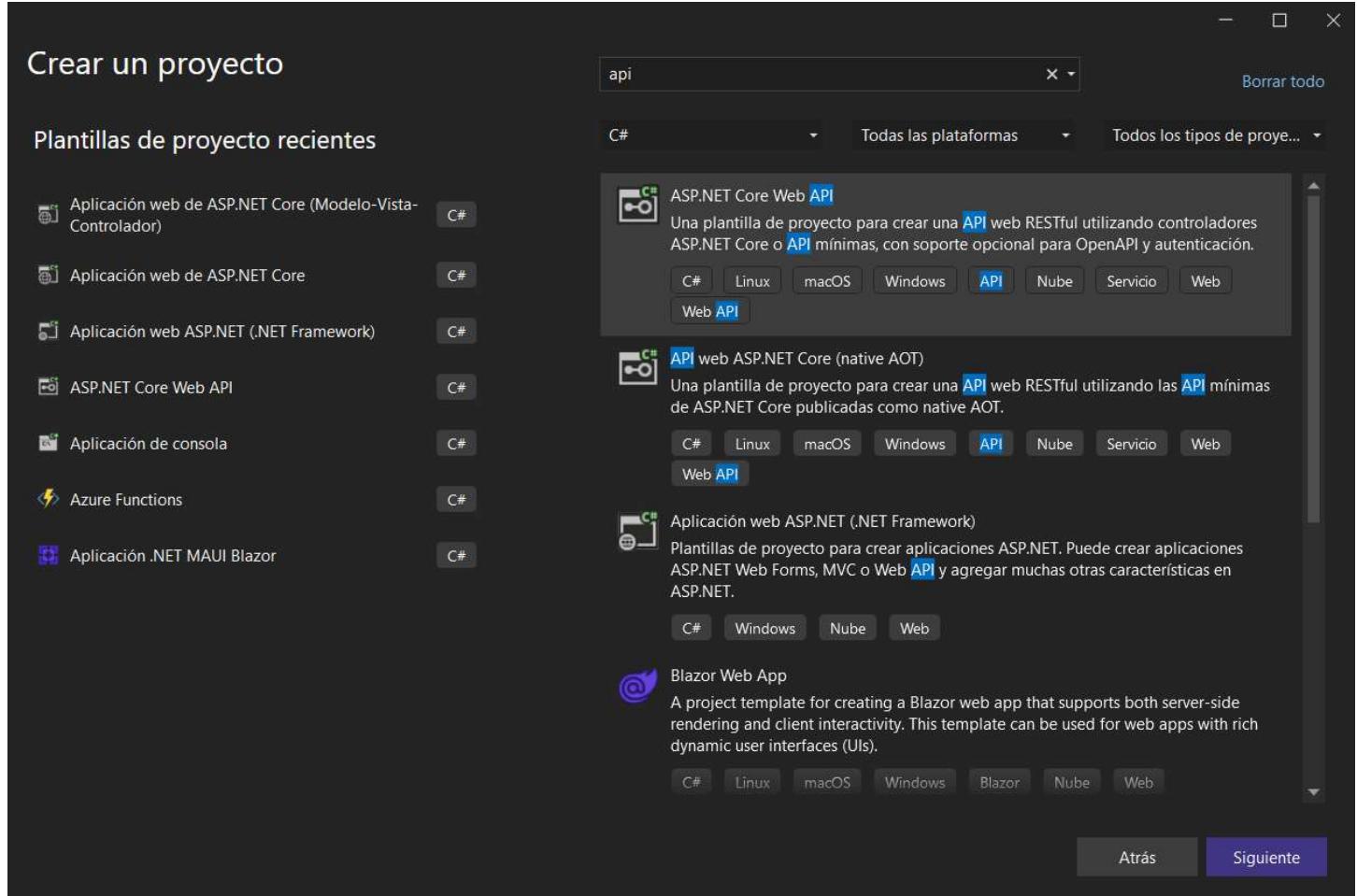
The 'Information' pane on the left provides details about the 'items' table, showing it has two columns: 'Id' (int AI PK) and 'Name' (varchar(255)). The 'Object Info' pane at the bottom shows the column definitions.

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

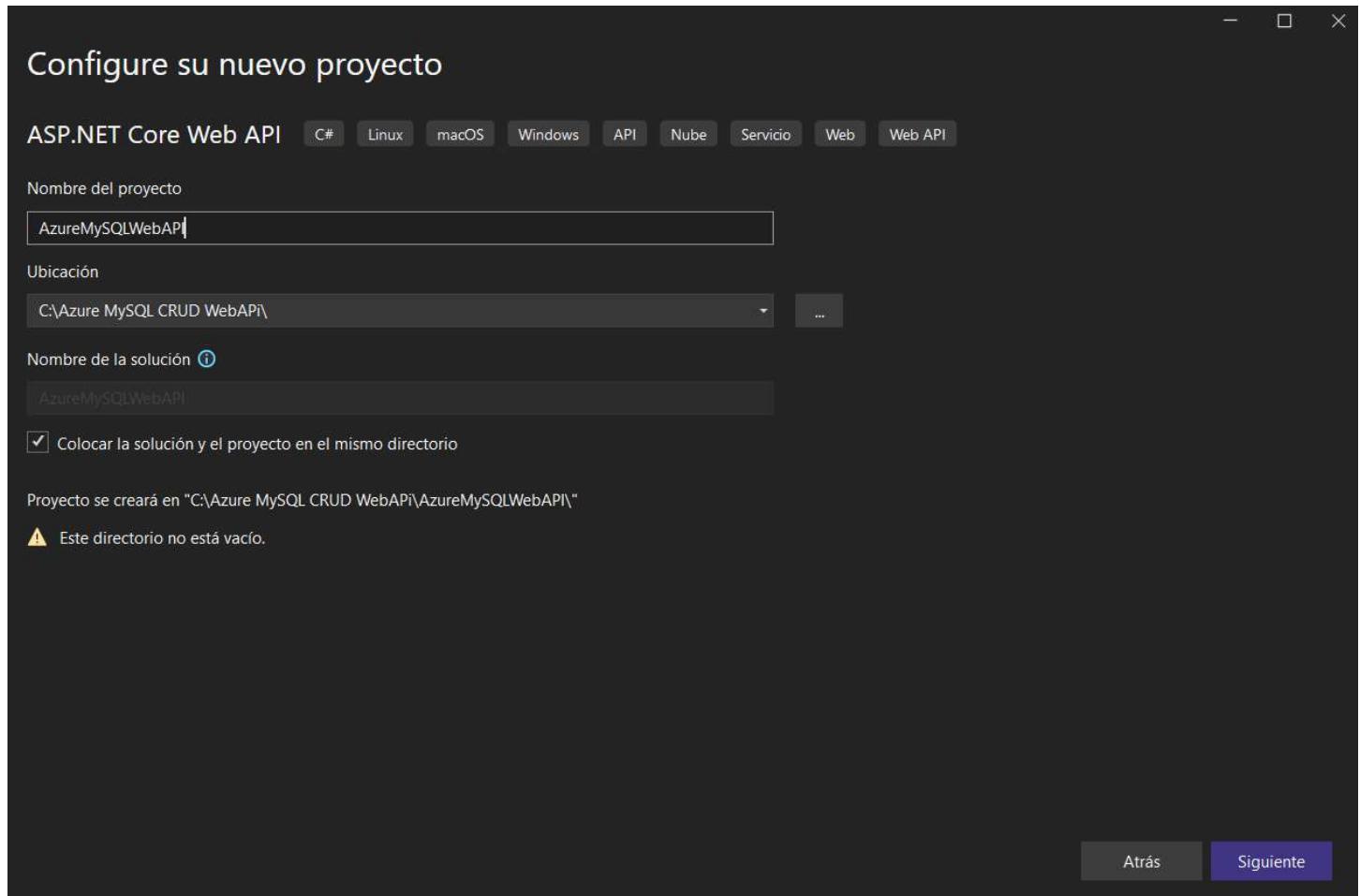
We run Visual Studio 2022 Community Edition and we create a new project



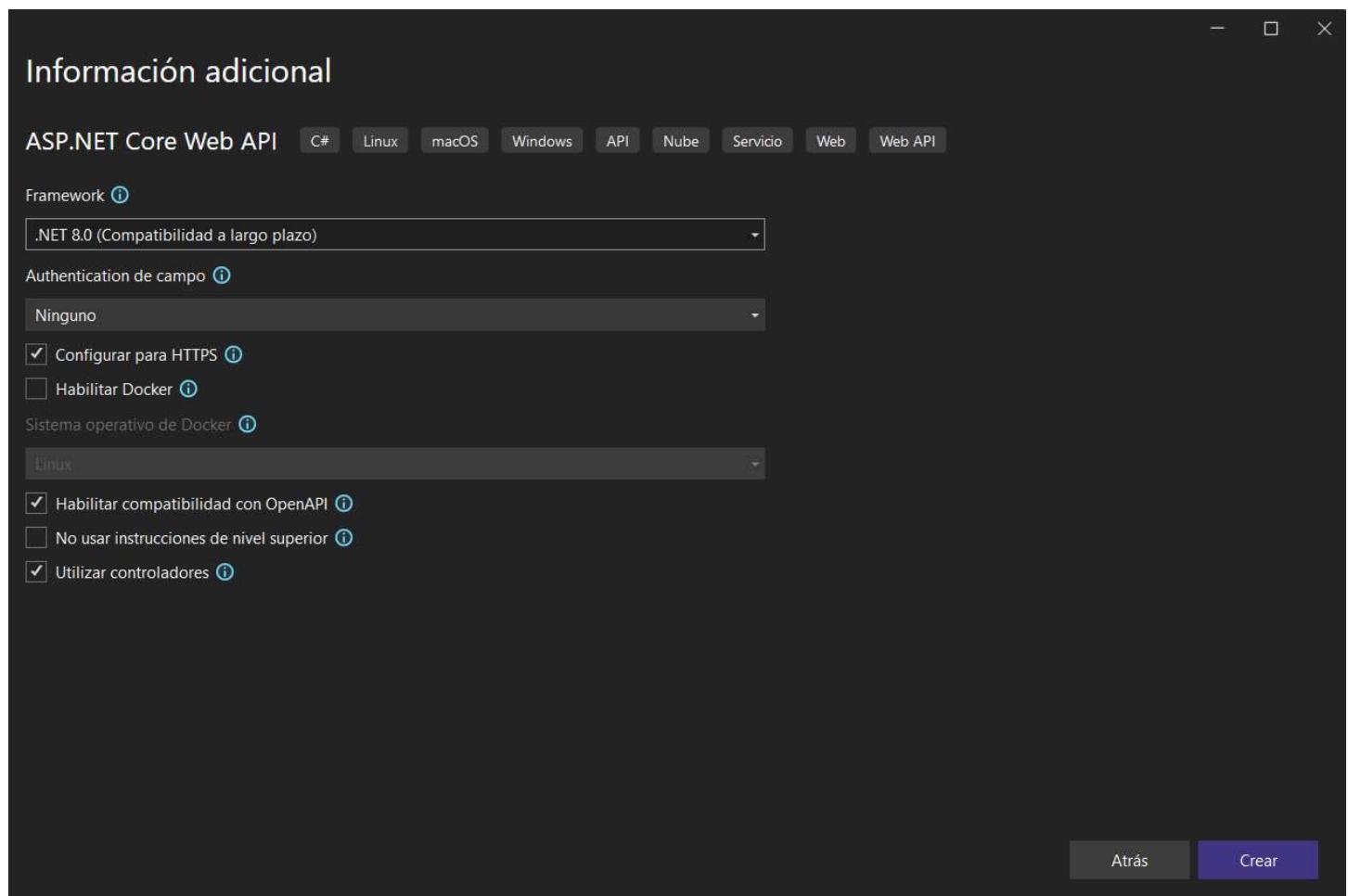
We select the **ASP Net Core Web API** project template



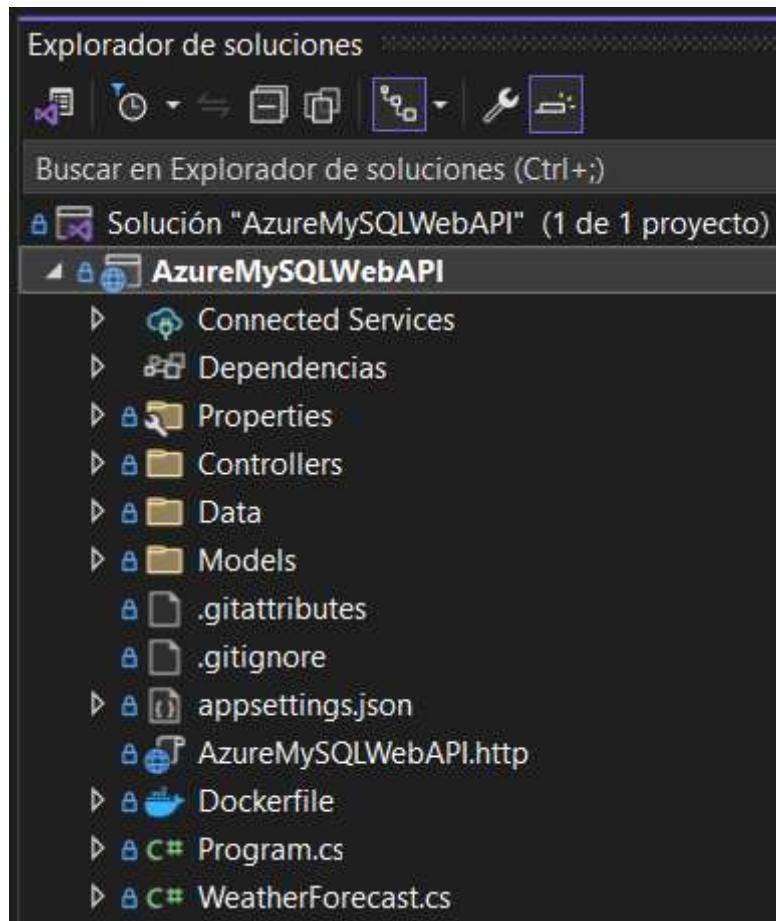
We set the project name and location



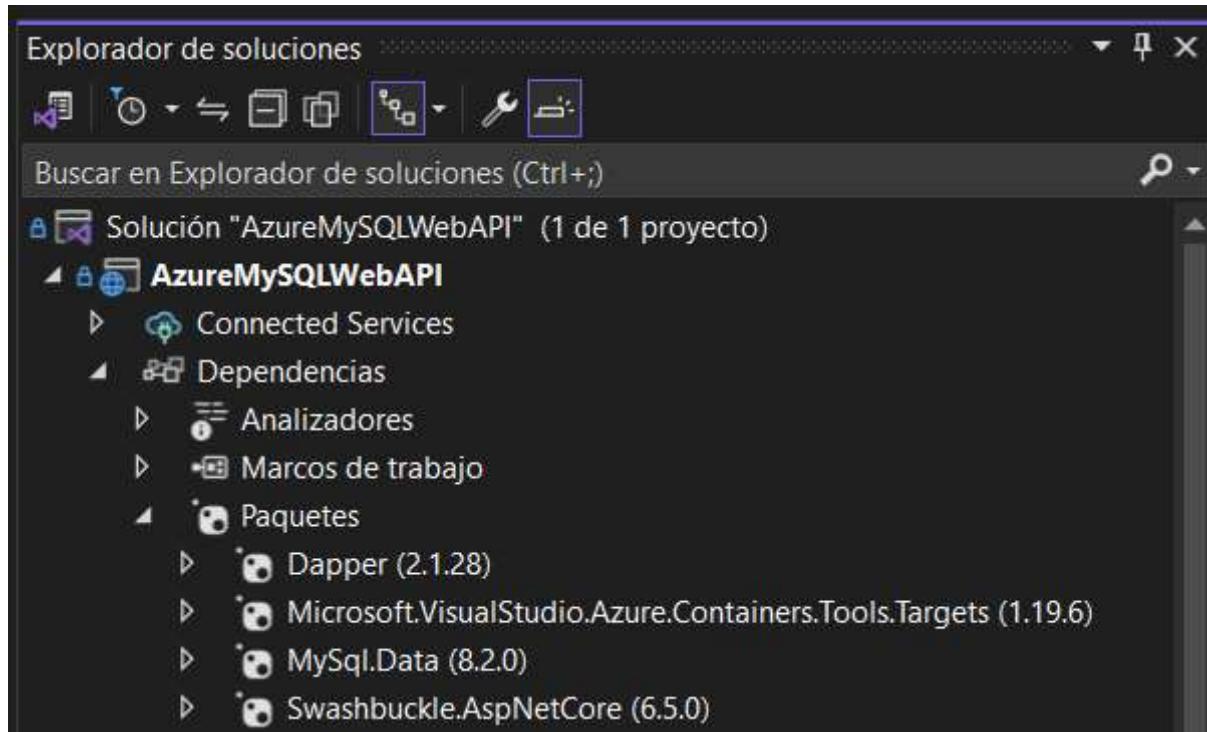
We select the project main features



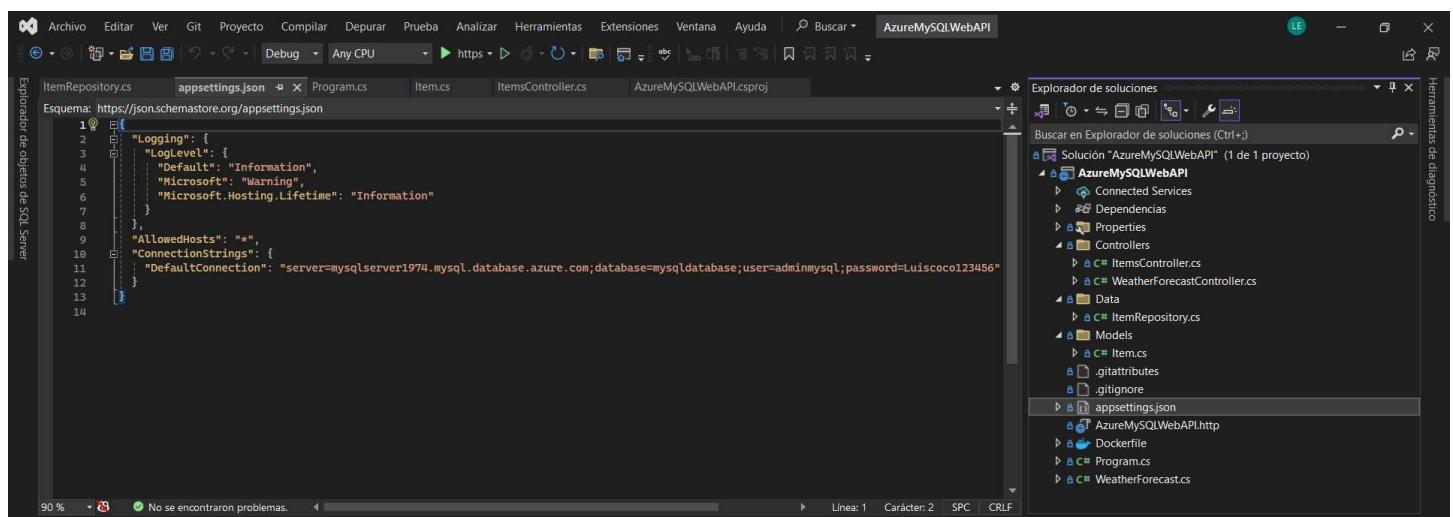
We create the following project folders structure, with the **Data** and **Models** new folders



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=mysqldatabase;user=adminmysql;password=Luiscocol123456"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "server=mysqlserver1974.mysql.database.azure.com;database=mysqldatabase;user=adminmysql;password=Luiscocol123456"
  }
}
```

```

    }
}

```

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

program.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 = @Id", new { Id = 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.StoredProcedure);
    }
}
}

```

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 endpoints with swagger

The screenshot shows the Swagger UI interface for a .NET8 CRUD WebAPI. The main title is "Items". Under the "GET /api/Items" section, there are tabs for "Parameters" (which shows "No parameters"), "Responses", and "Code". The "Responses" tab shows a 200 status code with a "Response body" containing JSON data:

```
[{"id": 1, "name": "Item 1"}, {"id": 2, "name": "Item 2"}, {"id": 3, "name": "Item 3"}]
```

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 to run it with Docker Desktop

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:UseAnsiEncoding=true

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 then 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

Name	Tag	Status	Created	Size	Actions
myapp	latest	In use	4 minutes ago	233.87 MB	Details

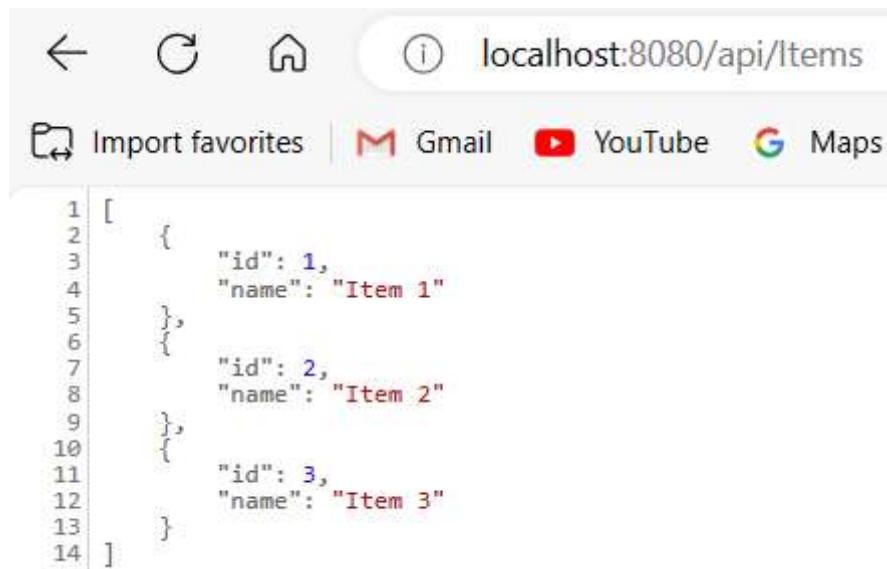
Name	Image	Status	Port(s)	Last started	CPU (%)	Actions
pensive_goodall	myapp:latest	Running	8080:8080	3 minutes ago	0.01%	Details

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>



```

1  [
2    {
3      "id": 1,
4      "name": "Item 1"
5    },
6    {
7      "id": 2,
8      "name": "Item 2"
9    },
10   {
11     "id": 3,
12     "name": "Item 3"
13   }
14 ]

```

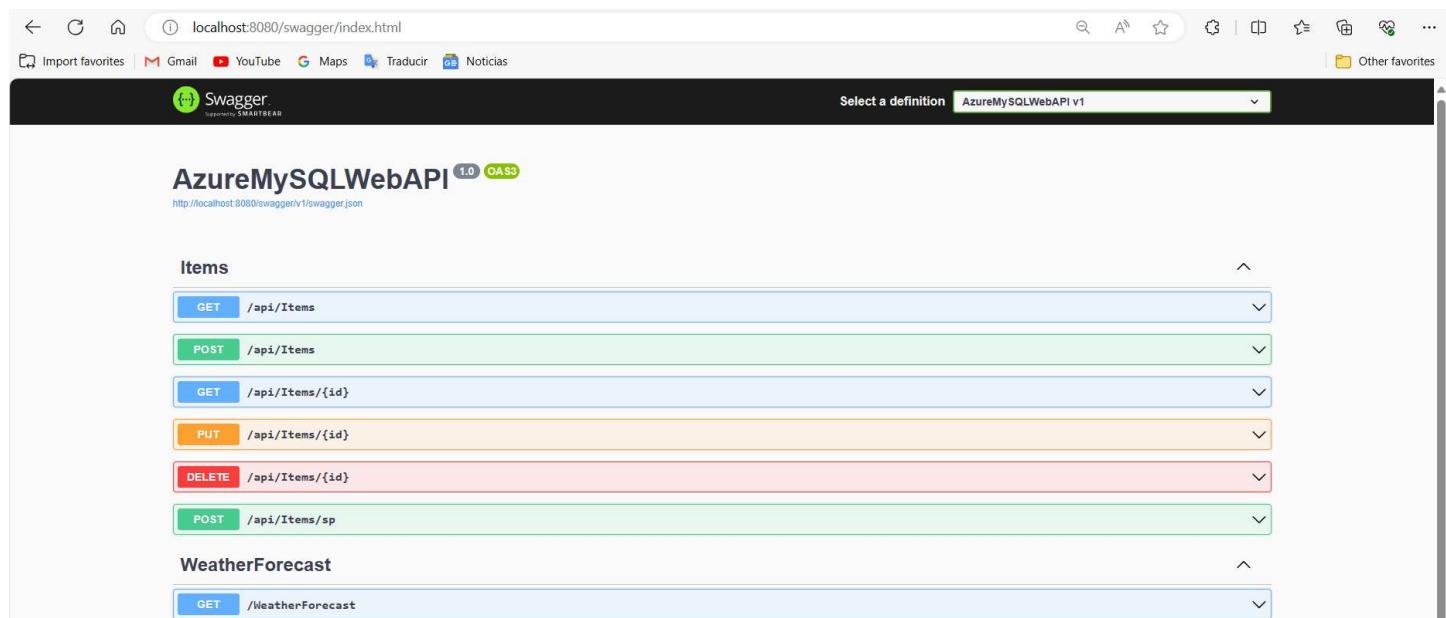
If we would like to access with Swagger the API documentation, we first should comment in program.cs the following lines:

```

...
// 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)



The screenshot shows the Swagger UI interface for the AzureMySQLWebAPI v1. At the top, it displays the title "AzureMySQLWebAPI 1.0 OAS3" and the URL "http://localhost:8080/swagger/v1/swagger.json". The main content area is titled "Items" and lists several API endpoints with their methods and URLs: GET /api/Items, POST /api/Items, GET /api/Items/{id}, PUT /api/Items/{id}, DELETE /api/Items/{id}, and POST /api/Items/sp. Below the "Items" section, there is another section titled "WeatherForecast" with a single endpoint: GET /WeatherForecast.

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
```

```

Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

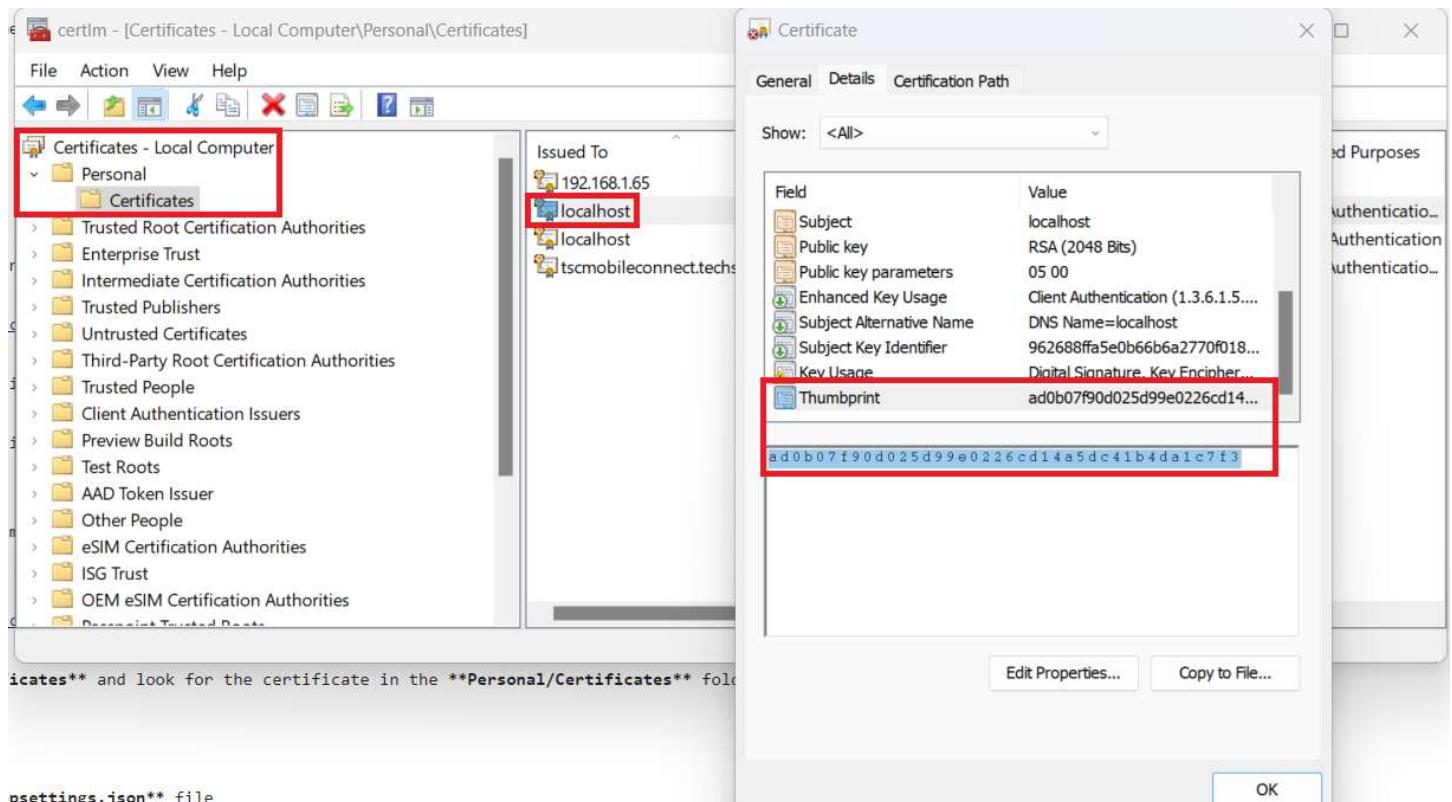
PS C:\WINDOWS\system32> New-SelfSignedCertificate -DnsName localhost -CertStoreLocation cert:\LocalMachine\My

PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\My

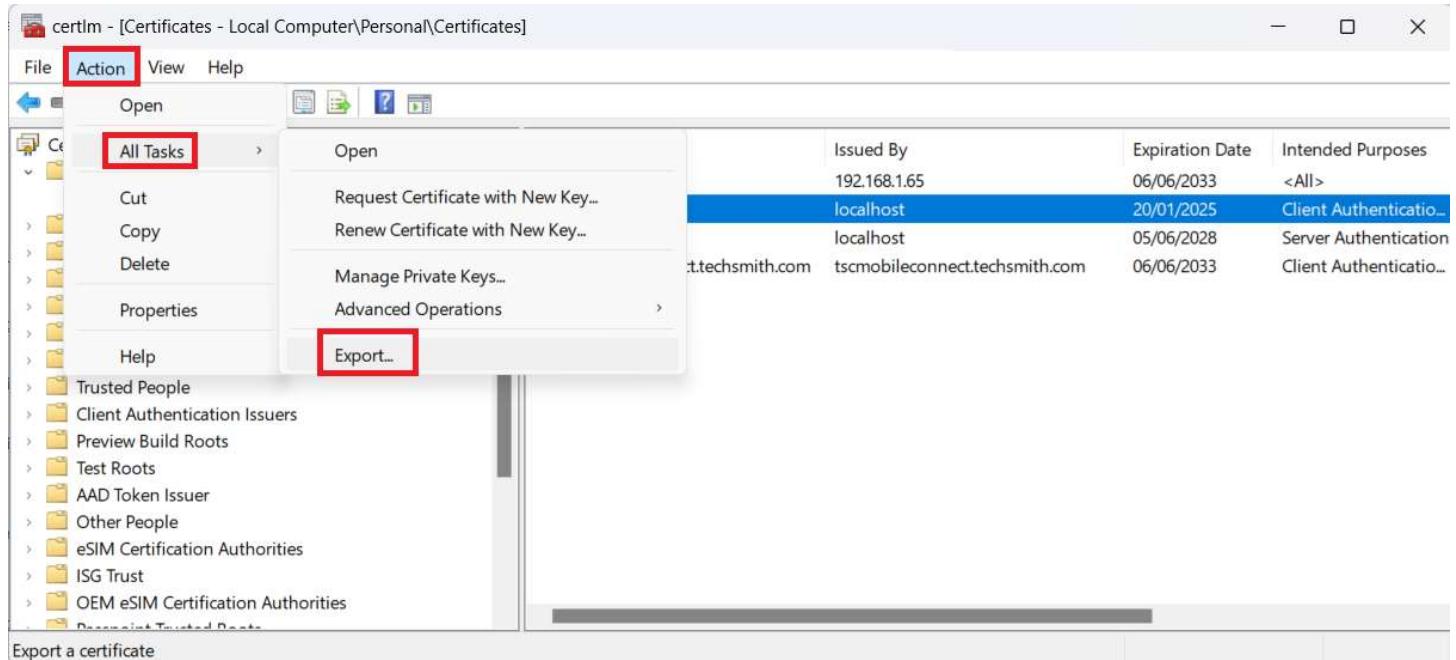
Thumbprint Subject
----- -----
AD0B07F90D025D99E0226CD14A5DC41B4DA1C7F3 CN=localhost

```

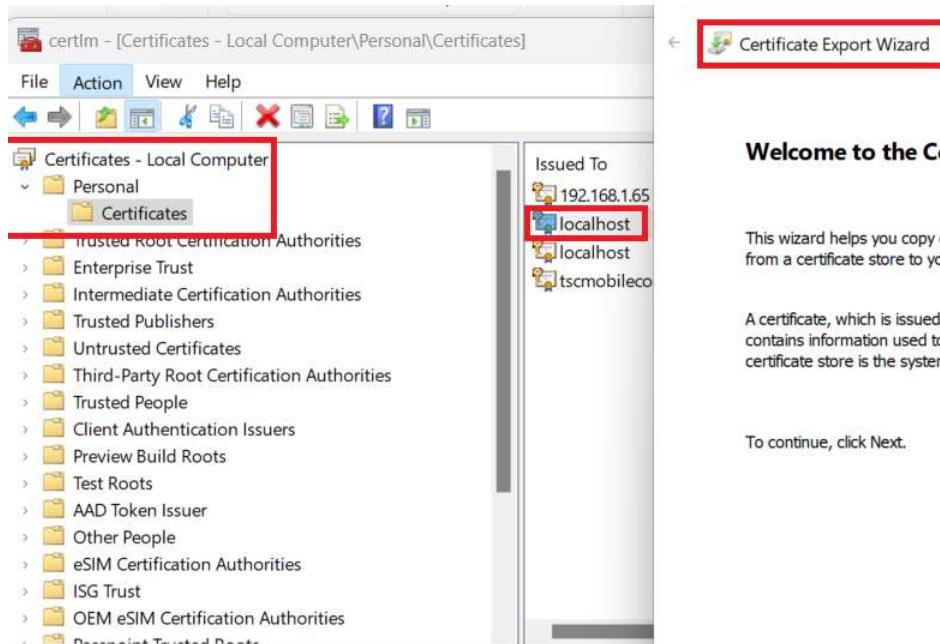
We open **Manage Computer Certificates** and look for the certificate in the **Personal/Certificates** folder



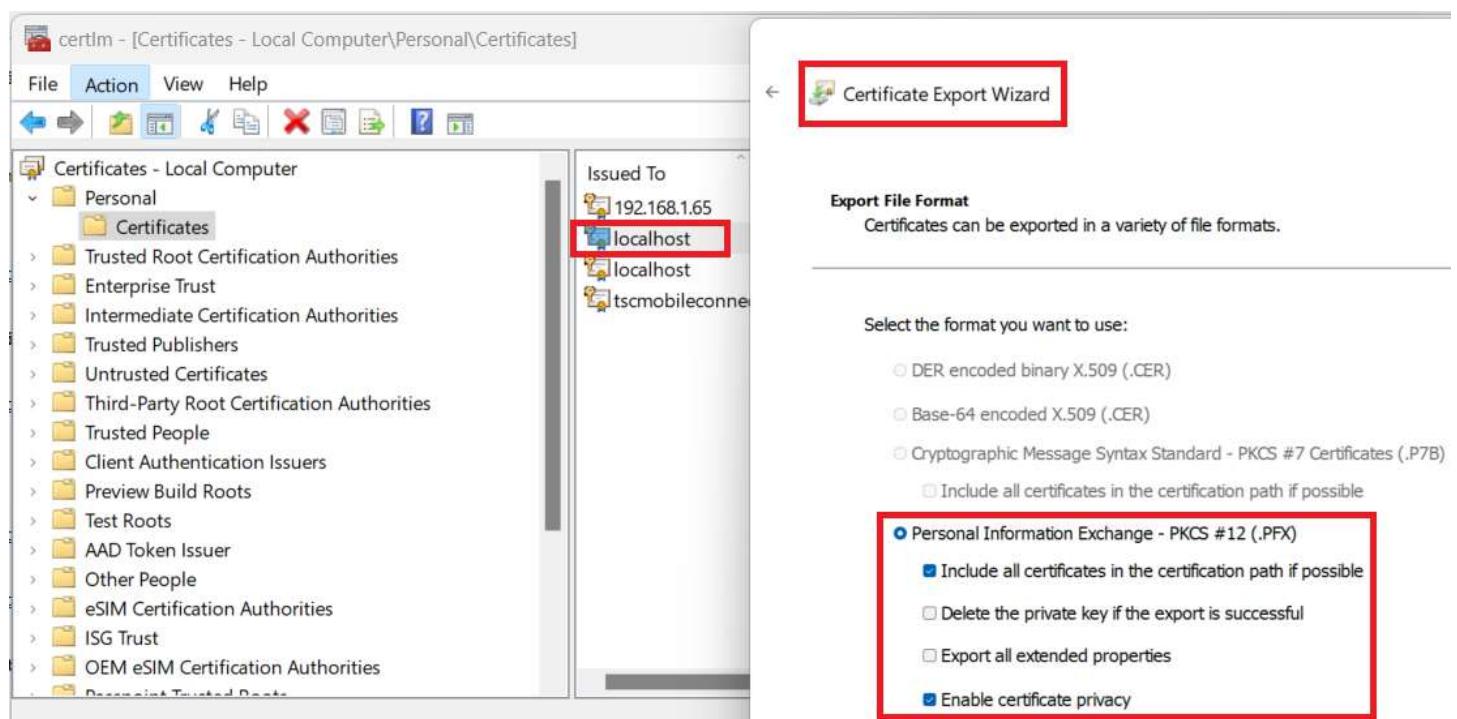
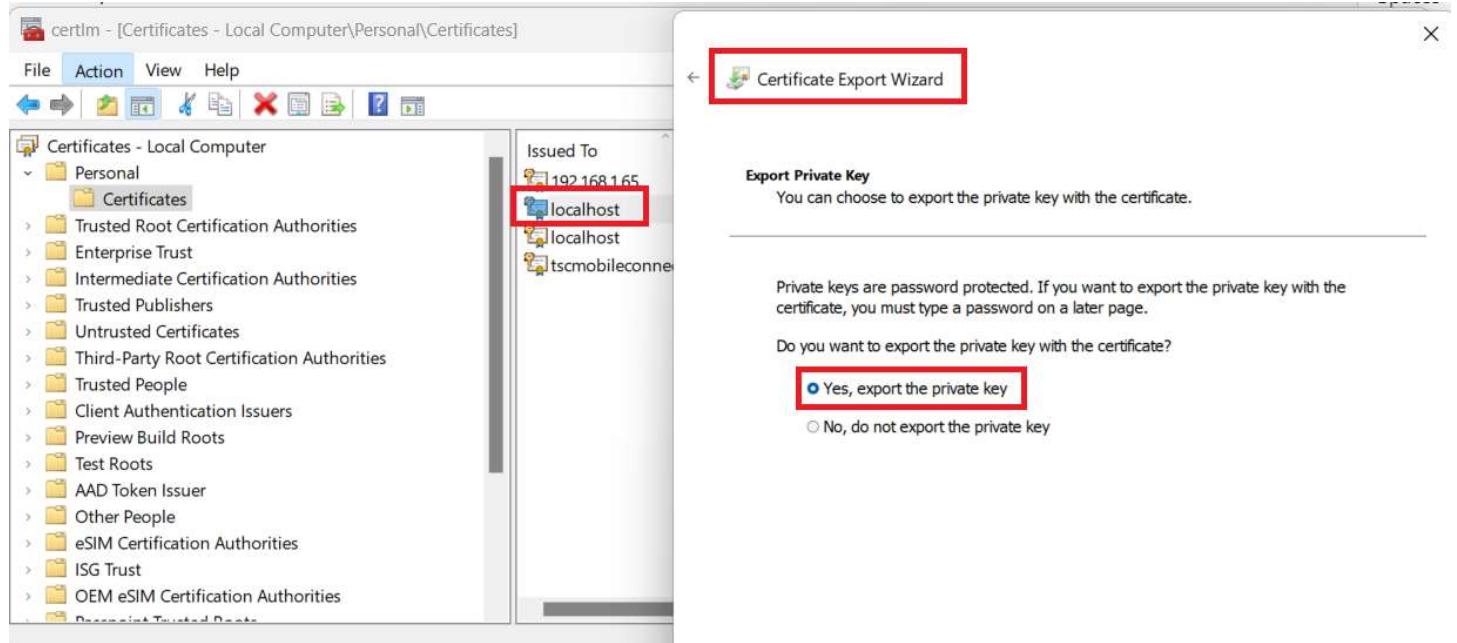
We select the certificate and we **export to PFX** file

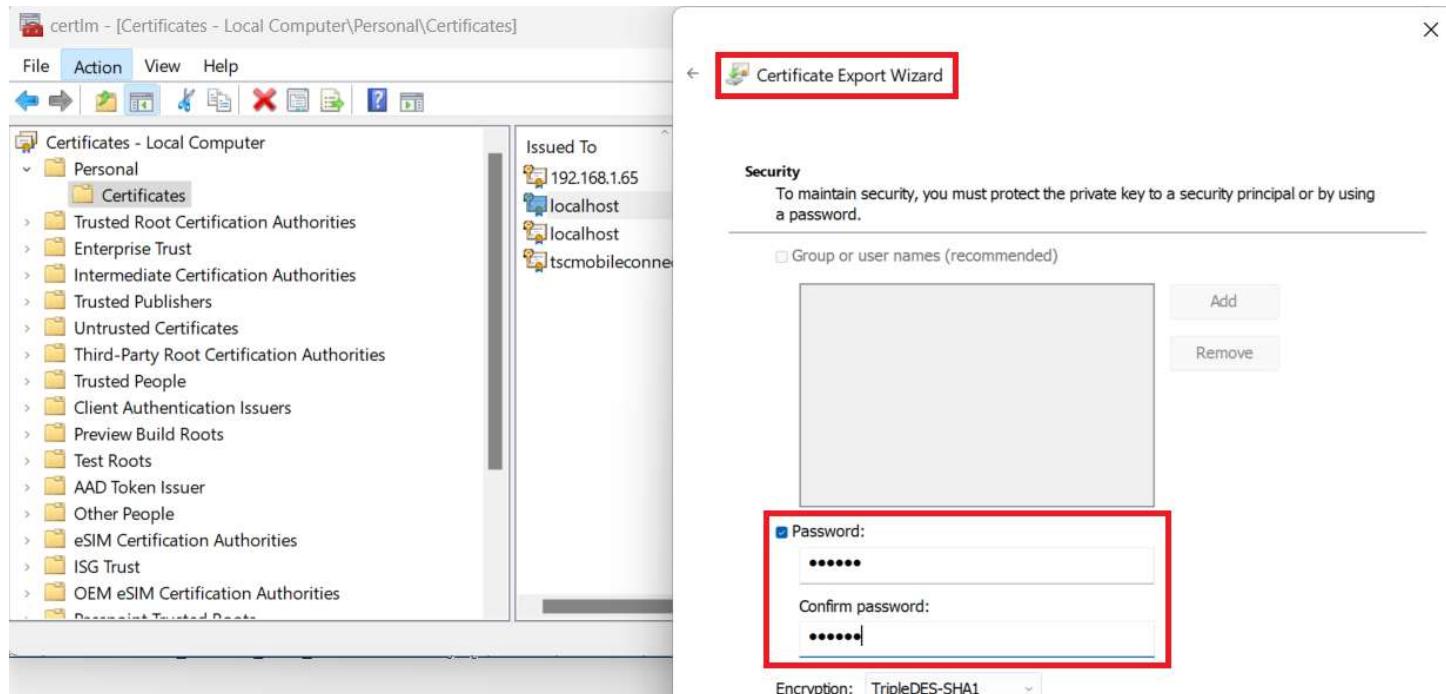


Export a certificate

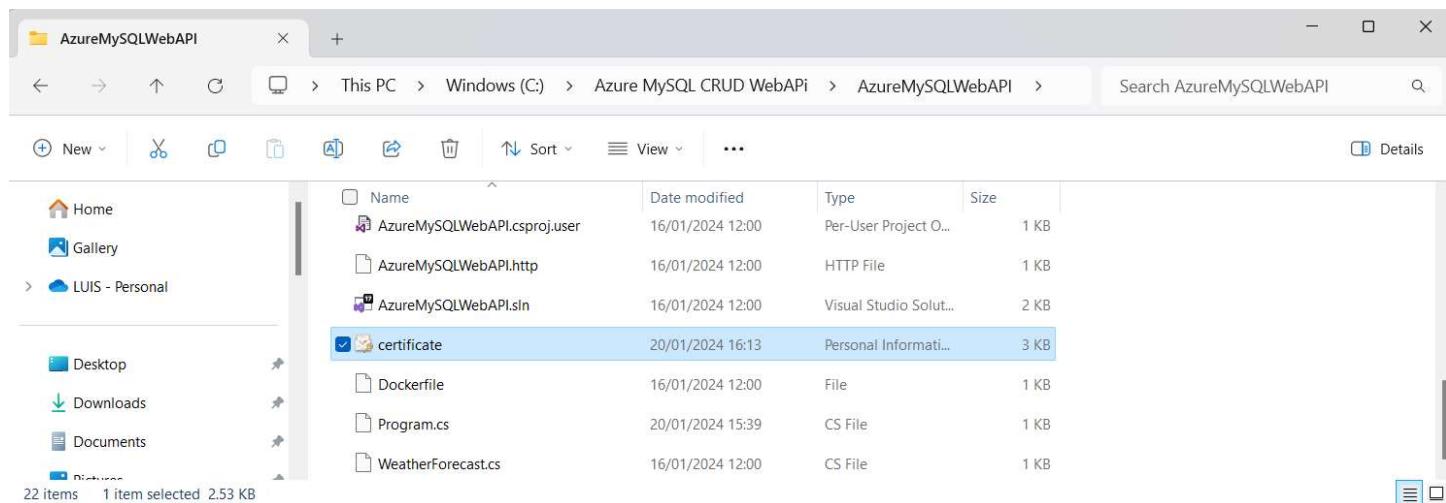


We select the option Yes export the private key





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

Trusted Root Certification Authorities store contains 68 certificates.

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=mysqldata"
  },
  "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**

Docker Desktop interface showing a single container named "bold_thompson" running the "myapp:latest" image. The container is assigned ports 8080:8080 and 8081:8081. It has been running for 7 minutes and is using 0.01% CPU.

We verify the application endpoints

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

The screenshot shows the Swagger UI for the AzureMySQLWebAPI v1. The main navigation bar includes links for ChatGPT, Swagger UI, and another Swagger UI instance. The main content area displays the API documentation for the 'Items' endpoint, which includes several methods: GET /api/Items, POST /api/Items, GET /api/Items/{id}, PUT /api/Items/{id}, DELETE /api/Items/{id}, and POST /api/Items/seed. The 'DELETE /api/Items/{id}' method is highlighted with a red background. Below this, the 'WeatherForecast' endpoint is shown with a single GET method for /WeatherForecast.

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

The screenshot shows the Swagger UI for the AzureMySQLWebAPI v1. The top navigation bar includes tabs for ChatGPT, Swagger UI, and another Swagger UI tab. The address bar shows the URL https://localhost:8081/swagger/index.html. The main content area displays the API documentation with sections for 'Items' and 'WeatherForecast'. Under 'Items', there are seven API endpoints listed: GET /api/Items, POST /api/Items, GET /api/Items/{id}, PUT /api/Items/{id}, DELETE /api/Items/{id}, POST /api/Items/seed, and a summary for WeatherForecast with a single GET /WeatherForecast endpoint.

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

The screenshot shows the Docker Desktop application window. The top bar includes tabs for 'Docker Desktop' and 'Update to latest', a search bar, and a 'Ctrl+K' keyboard shortcut. On the far right, there is a user profile icon with the name 'luiscoco' and a 'Settings' button, which is also highlighted with a red box. The main pane shows the 'Images' section with tabs for 'Local', 'Hub', and 'Artifactory (EARLY ACCESS)'. It displays 1 image, 0 Bytes / 207.69 MB in use, and a 'Last refresh: 4 hours ago' message. The left sidebar contains links for 'Containers', 'Images', 'Volumes', 'Dev Environments (BETA)', 'Docker Scout', and 'Learning center'.

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

The screenshot shows the Docker Desktop settings interface. On the left, a sidebar lists categories: General, Resources, Docker Engine, **Kubernetes**, Software updates, Extensions, and Features in development. The Kubernetes section is highlighted with a blue background. The title "Kubernetes" and version "v1.27.2" are displayed. Two configuration options are shown with checked checkboxes: "Enable Kubernetes" and "Show system containers (advanced)". A red-bordered button labeled "Reset Kubernetes Cluster" is present. Below it, a warning message states "All stacks and Kubernetes resources will be deleted." At the bottom right are "Cancel" and "Apply & restart" buttons.

The screenshot shows the Docker Desktop Images tab. The sidebar includes links for Containers, Images, Volumes, Dev Environments (BETA), Docker Scout, and Learning center. The main area displays a table of images with columns for Name, Tag, Status, and Created. The table shows 12 items, including local images like "luiscoco/myapp" and "luiscoco/demoapi", and system images from the Docker Hub internal registry such as "hubproxy.docker.internal:5555/docker/desktop-kubernetes". The status bar at the bottom indicates "Showing 12 items".

Name	Tag	Status	Created
luiscoco/myapp	latest	Unused	9 minutes ago
luiscoco/demoapi	latest	In use	21 days ago
hubproxy.docker.internal:5555/docker/desktop-kubernetes	kubern	Unused	8 months ago
registry.k8s.io/kube-apiserver	v1.27.0	In use	8 months ago
registry.k8s.io/kube-scheduler	v1.27.0	In use	8 months ago

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:

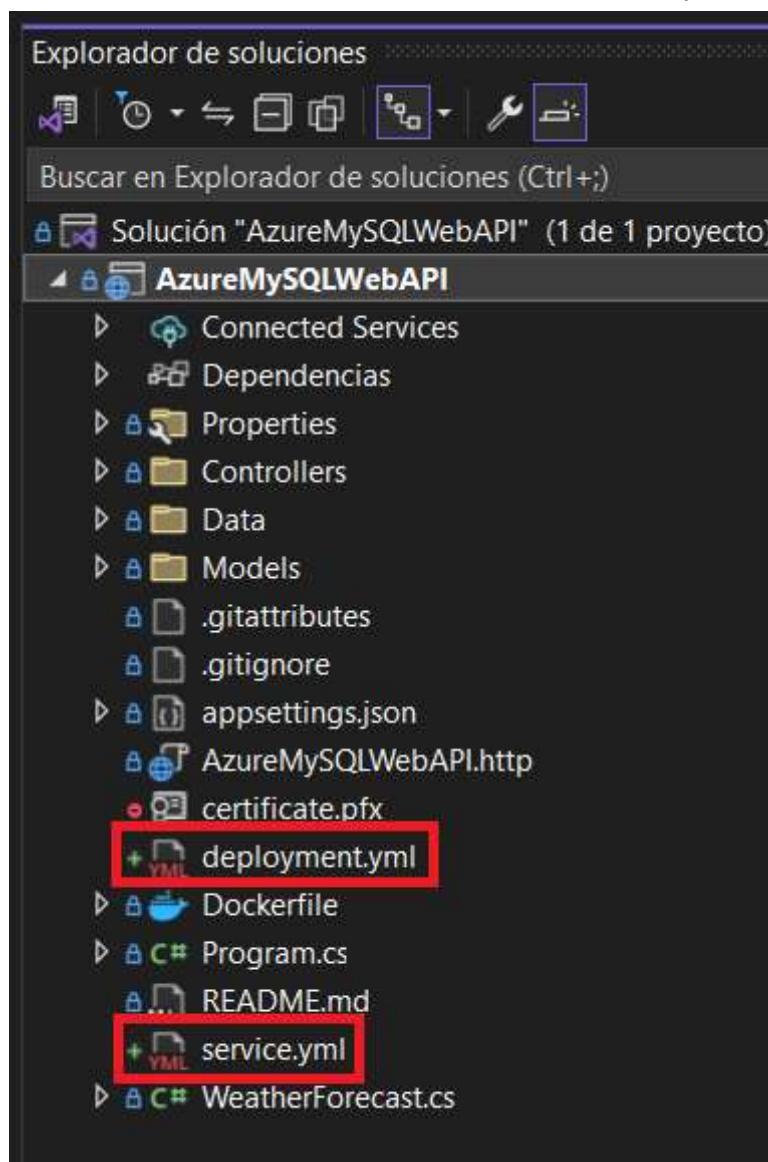
```
docker images
```

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

```
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>

The screenshot shows the Swagger UI interface for the AzureMySQLWebAPI v1. At the top, the URL is `localhost/swagger/index.html`. The main title is "AzureMySQLWebAPI 1.0 OAS3". Below it, the URL `http://localhost/swagger/v1/swagger.json` is shown. The interface is organized into sections: "Items" and "WeatherForecast". The "Items" section contains several API endpoints: a GET method for `/api/Items` (blue button), a POST method for `/api/Items` (green button), a GET method for `/api/Items/{id}` (light blue button), a PUT method for `/api/Items/{id}` (orange button), a DELETE method for `/api/Items/{id}` (red button), and two additional POST methods for `/api/Items/sp` (light green buttons). The "WeatherForecast" section contains a single GET method for `/WeatherForecast` (blue button).

Items

GET /api/Items

Parameters

No parameters

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"}]
```

Cancel

Execute Clear

Download