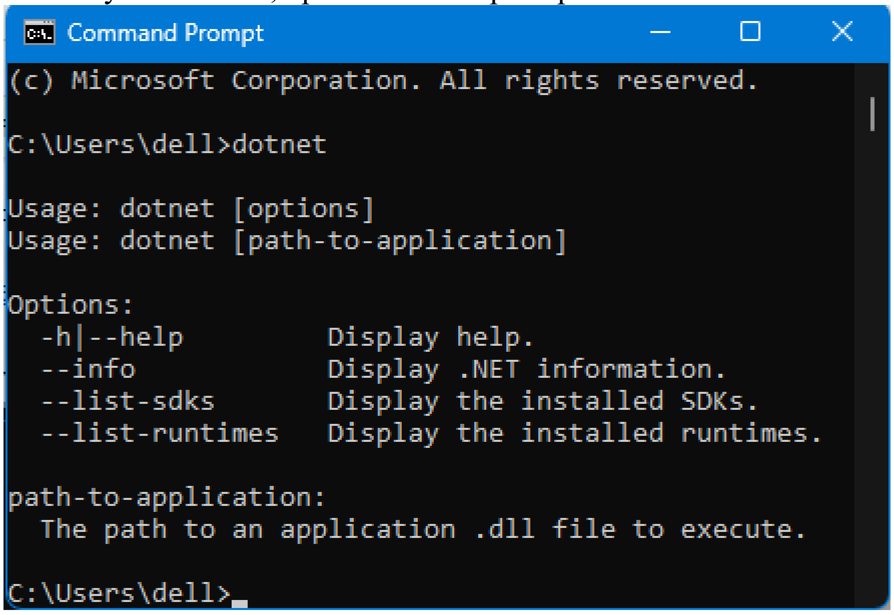
Practical No : 1

Aim: Create a console based ASP.NET Core application

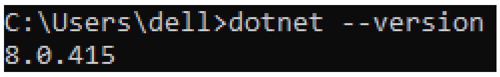
Step 1:

- Download asp.net core from https://dotnet.microsoft.com/en-us/download
- Install the downloaded file
- To verify installation, open command prompt and enter 'dotnet'.



```
(c) Microsoft Corporation. All rights reserved.

C:\Users\dell>dotnet

Usage: dotnet [options]
Usage: dotnet [path-to-application]

Options:
  -h|--help         Display help.
  --info            Display .NET information.
  --list-sdks       Display the installed SDKs.
  --list-runtimes   Display the installed runtimes.

path-to-application:
  The path to an application .dll file to execute.

C:\Users\dell>
```

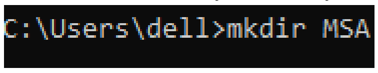
- To check dotnet version enter command : dotnet -–version



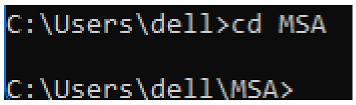
```
C:\Users\dell>dotnet --version
8.0.415
```

Step 2:

- Create a directory where you want to store all the files

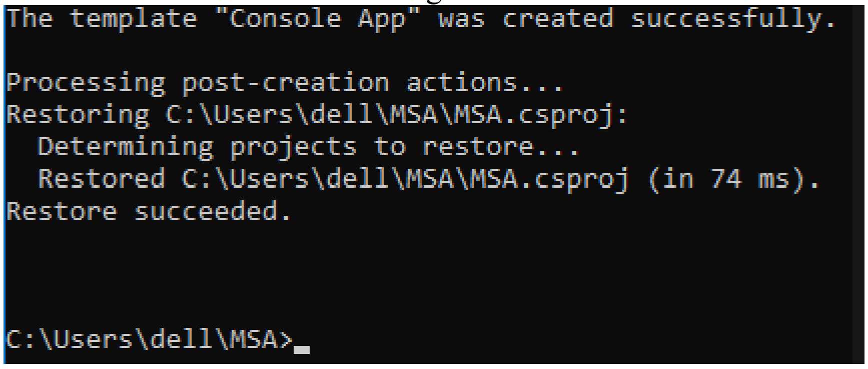


```
C:\Users\dell>mkdir MSA
```



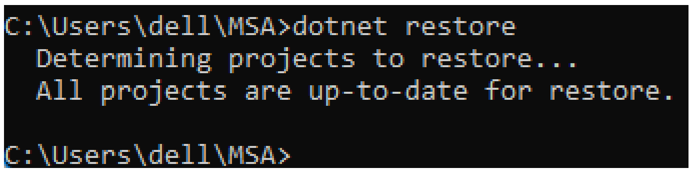
```
C:\Users\dell>cd MSA

C:\Users\dell\MSA>
```

- Create a new dotnet console using command : dotnet new console



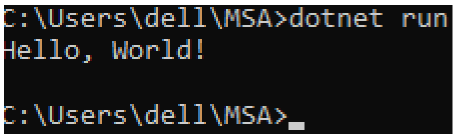
```
The template "Console App" was created successfully.

Processing post-creation actions...
Restoring C:\Users\dell\MSA\MSA.csproj:
  Determining projects to restore...
  Restored C:\Users\dell\MSA\MSA.csproj (in 74 ms).
Restore succeeded.



C:\Users\dell\MSA>
```

- To restore dotnet enter command : dotnet restore



```
C:\Users\dell\MSA>dotnet restore
  Determining projects to restore...
  All projects are up-to-date for restore.

C:\Users\dell\MSA>
```

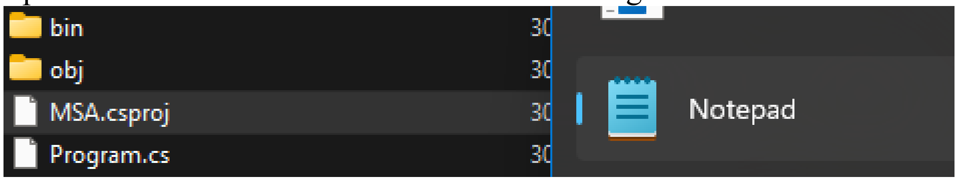
- To run dotnet enter command : dotnet run



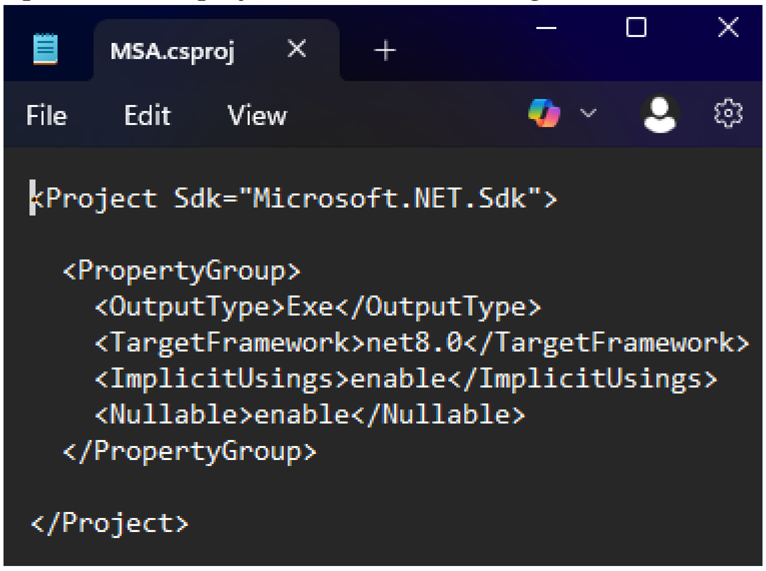
```
C:\Users\dell\MSA>dotnet run
Hello, World!

C:\Users\dell\MSA>
```

Step 3:

- Open the file location. You will find the following files.

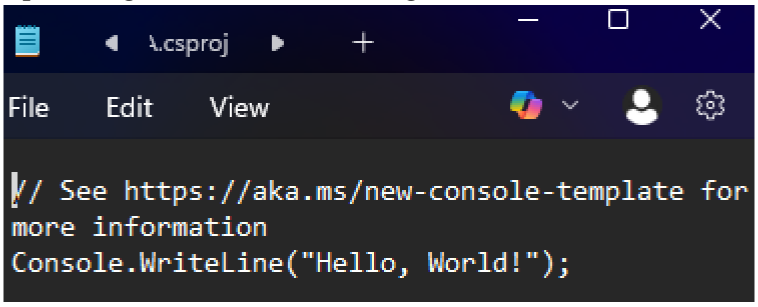


- Open MSA C# project file . The following code will be available there by default.



```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

</Project>
```

- Open Program.cs. The following code will be available there by default.

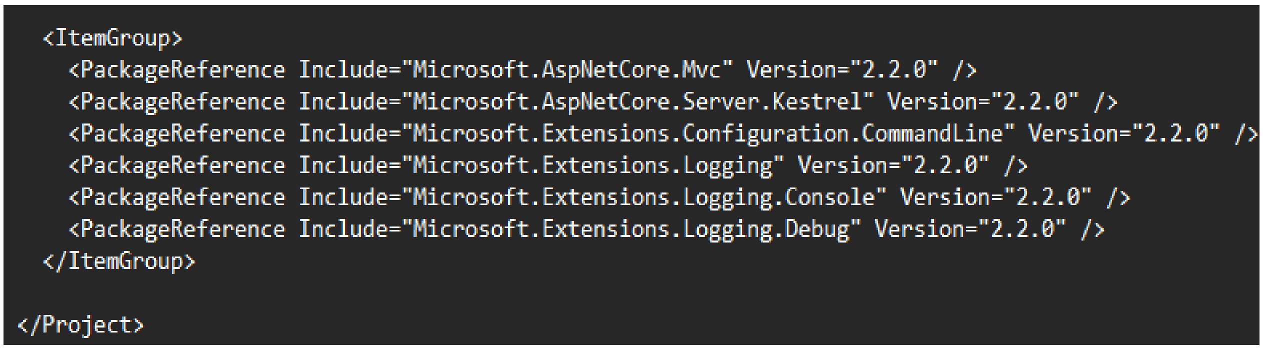


```
// See https://aka.ms/new-console-template for
more information
Console.WriteLine("Hello, World!");
```

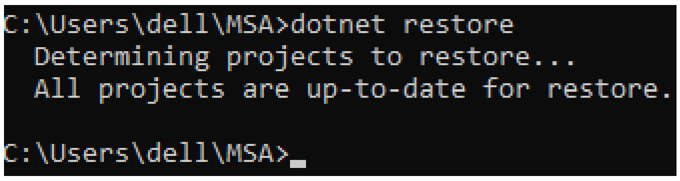
Step 4:

- Install the following packages using command : dotnet add package package_name -v version_number
  1. Microsoft.AspNetCore.Mvc
  2. Microsoft.AspNetCore.Server.Kestrel
  3. Microsoft.Extensions.Logging
  4. Microsoft.Extensions.Logging.Console
  5. Microsoft.Extensions.Logging.Debug
  6. Microsoft.Extensions.Configuration.CommandLine
- The installed packages will be reflected automatically in the MSA.csproj file.

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>
```


```
  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="2.2.0" />
    <PackageReference Include="Microsoft.AspNetCore.Server.Kestrel" Version="2.2.0" />
    <PackageReference Include="Microsoft.Extensions.Configuration.CommandLine" Version="2.2.0" />
    <PackageReference Include="Microsoft.Extensions.Logging" Version="2.2.0" />
    <PackageReference Include="Microsoft.Extensions.Logging.Console" Version="2.2.0" />
    <PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="2.2.0" />
  </ItemGroup>

</Project>
```

- Use ‘dotnet restore’ to restore


```
C:\Users\dell\MSA>dotnet restore
  Determining projects to restore...
  All projects are up-to-date for restore.

C:\Users\dell\MSA>_
```

Step 5:

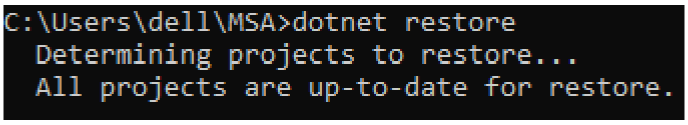
- Open Program.cs and enter the following code.

```
using System;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Configuration;
using Microsoft.AspNetCore.Http;

namespace helloworld
{
   internal class Program
   {
      static void Main(string[] args)
      {
         var config = new ConfigurationBuilder()
            .AddCommandLine(args)
            .Build();

         var host = new WebHostBuilder()
            .UseKestrel()
            .UseStartup<Startup>()
            .UseUrls("http://localhost:5200/")
            .UseConfiguration(config)
            .Build();

         host.Run();
      }
   }

   public class Startup
   {
      public Startup(IHostingEnvironment env) { }

      public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory
loggerFactory)
      {
         app.Run(async (context) =>
         {
            await context.Response.WriteAsync("Hello, world!");
```

```
                });
            }
        }
    }
```

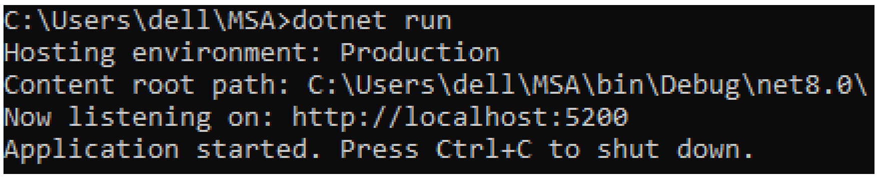
OUTPUT :

- Use ‘dotnet restore’ to restore



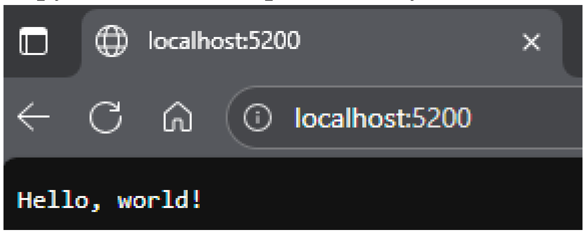
```
C:\Users\dell\MSA>dotnet restore
  Determining projects to restore...
  All projects are up-to-date for restore.
```

- Use ‘dotnet run’ to run



```
C:\Users\dell\MSA>dotnet run
Hosting environment: Production
Content root path: C:\Users\dell\MSA\bin\Debug\net8.0\
Now listening on: http://localhost:5200
Application started. Press Ctrl+C to shut down.
```

- Copy the URL and open it in any browser



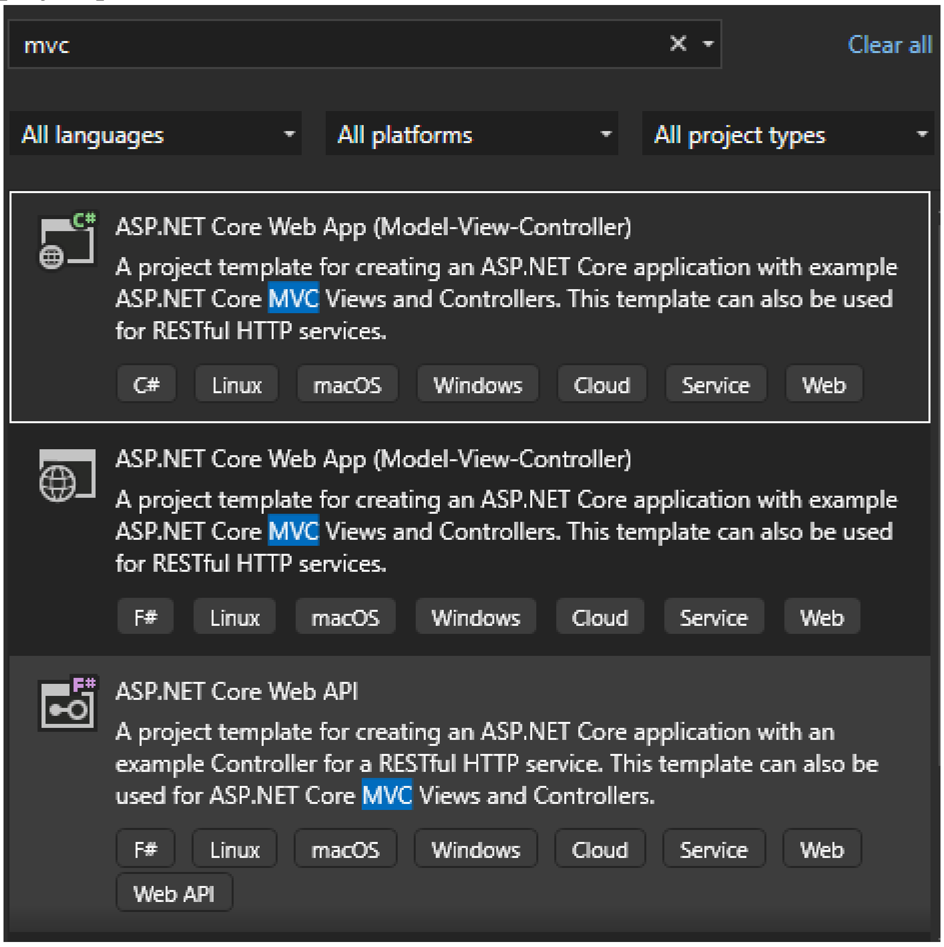
```
Hello, world!
```

Practical No : 2

Aim: Create a MVC Project in ASP.net core

Step 1:

- Go to the command prompt and navigate to the folder to where you want to store application files. Enter the following command: mkdir practical2 Then go to visual studio and create a project practical 2



mvc

Clear all

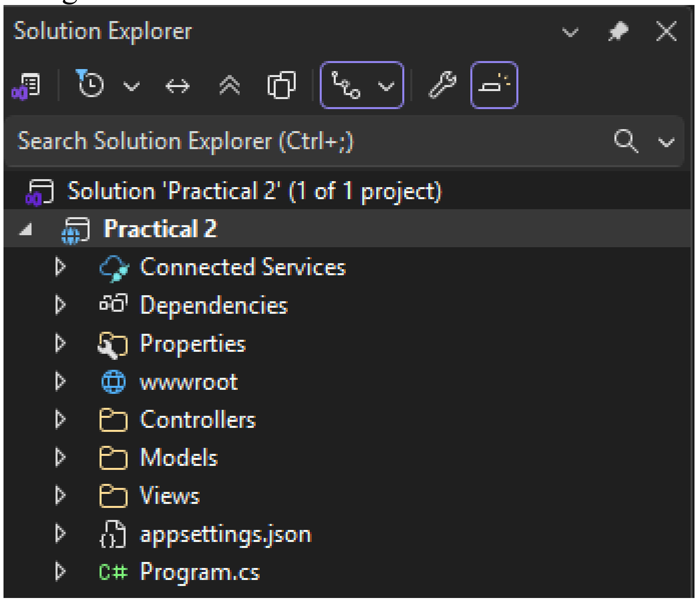
All languages | All platforms | All project types

ASP.NET Core Web App (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.
C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web App (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.
F# Linux macOS Windows Cloud Service Web

ASP.NET Core Web API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.
F# Linux macOS Windows Cloud Service Web Web API

- Navigate to the location and files will be created in the specified location.



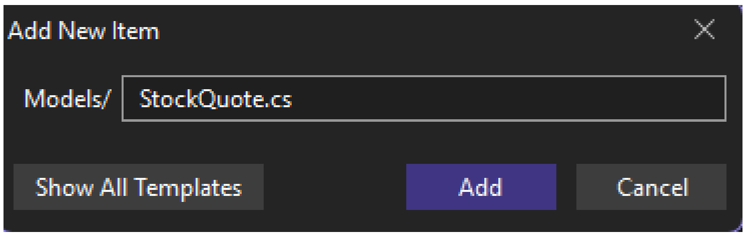
Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'Practical 2' (1 of 1 project)
Practical 2
Connected Services
Dependencies
Properties
wwwroot
Controllers
Models
Views
appsettings.json
Program.cs

Step 2:

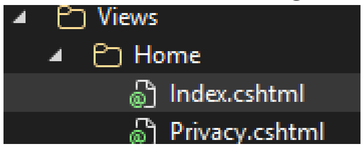
- Right on model and add a new item and enter the following code. Save the file as ‘StockQuote.cs’ in the Models folder of the project.



```
namespace Practical_2.Models
{
   public class StockQuote
   {
      public string Symbol { get; set; } = default!;
      public int Price { get; set; } = default!;
   }

}
```

Step 3:

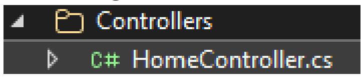
- Go to Views folder → Open Home folder → Open Index.cshtml file and replace the existing code with the following:



```
@{
   ViewData["Title"] = "Home Page";
}
<html>
<head>
   <title>Hello World</title>
</head>
<body>
   <h1>Hello World</h1>
   <div>
      <h2>StockQuote</h2>
      <div>
         Symbol: @Model.Symbol<br />
         Price: $@Model.Price<br />
      </div>
   </div>
</body>
</html>
```

Step 4:

- Go to Controllers folder → Open HomeController.cs file and replace the existing code with the following:

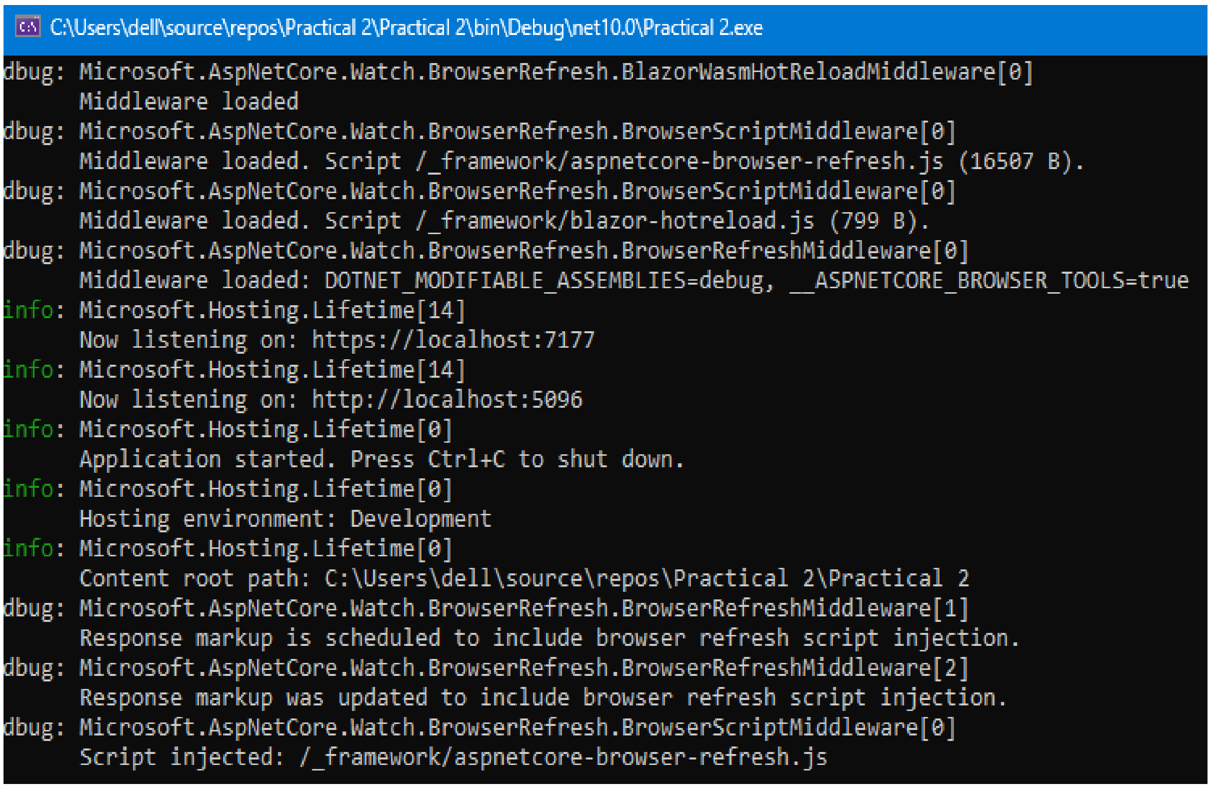


```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Practical_2.Models;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
namespace Practical_2.Controllers;
```

```
public class HomeController : Controller
{
   public async Task<IActionResult> Index()
   {
      var model = new StockQuote
      {
         Symbol = "Nike",
         Price = 3200
      }; await Task.Delay(0); return View(model);
   }
}
```

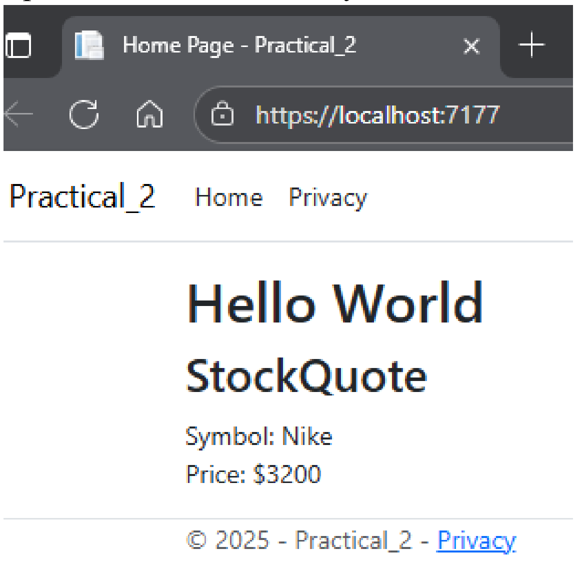
Step 5:

- Run the code



```
C:\Users\dell\source\repos\Practical 2\Practical 2\bin\Debug\net10.0\Practical 2.exe
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BlazorWasmHotReloadMiddleware[0]
      Middleware loaded
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserScriptMiddleware[0]
      Middleware loaded. Script /_framework/aspnetcore-browser-refresh.js (16507 B).
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserScriptMiddleware[0]
      Middleware loaded. Script /_framework/blazor-hotreload.js (799 B).
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserRefreshMiddleware[0]
      Middleware loaded: DOTNET_MODIFIABLE_ASSEMBLIES=debug, __ASPNETCORE_BROWSER_TOOLS=true
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7177
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5096
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: C:\Users\dell\source\repos\Practical 2\Practical 2
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserRefreshMiddleware[1]
      Response markup is scheduled to include browser refresh script injection.
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserRefreshMiddleware[2]
      Response markup was updated to include browser refresh script injection.
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserScriptMiddleware[0]
      Script injected: /_framework/aspnetcore-browser-refresh.js
```

- Open the above URL in any browser A



Home Page - Practical_2

https://localhost:7177

Practical_2 Home Privacy

Hello World

StockQuote

Symbol: Nike

Price: $3200

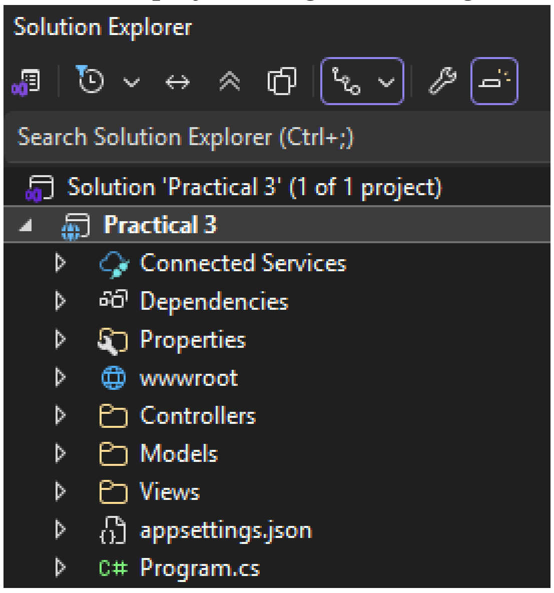
© 2025 - Practical_2 - Privacy

Practical No : 3

Aim: Invoking REST APIs from JavaScript

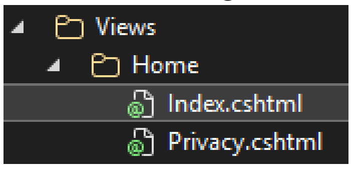
Step 1:

- Create new project using the below given template.



Step 2:

- Go to Views folder → Open Home folder→ Open Index.cshtml file and replace the existing code with the following:

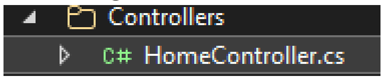


```
<!DOCTYPE html>
<html>
<head>
    <title>Invoke API</title>
    <script>
        async function fetchData() {
            try {
                const response = await fetch('/api/data');
                if (!response.ok) {
                    throw new Error('Network response was not ok');
                }

                const data = await response.json();
                document.getElementById('output').innerText = JSON.stringify(data, null, 2);
            } catch (error) {
                console.error('Error fetching data:', error);
                document.getElementById('output').innerText = 'Error fetching data.';
            }
        }
    </script>
</head>
<body>
    <h1>Fetch Data from API</h1>
    <button onclick="fetchData()">Fetch API Data</button>
    <pre id="output"></pre>
</body>
```

```
</html>
```

Step 3:

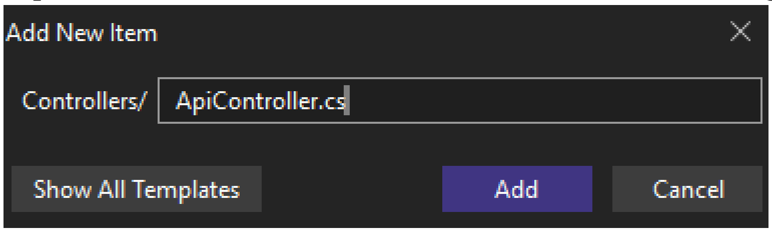
- Go to Controllers folder→ Open HomeController.cs file and replace the existing code with the following:



```
using Microsoft.AspNetCore.Mvc;

namespace MvcApp.Controllers
{
   public class HomeController : Controller
   {
      public IactionResult Index()
      {
         return View();
      }
   }
}
```

Step 4:

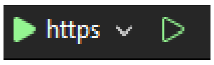
- Creating an API: Right click on project→ Add→New Items. Create a new file named ‘ApiController.cs’ in the Controller folder with the following code



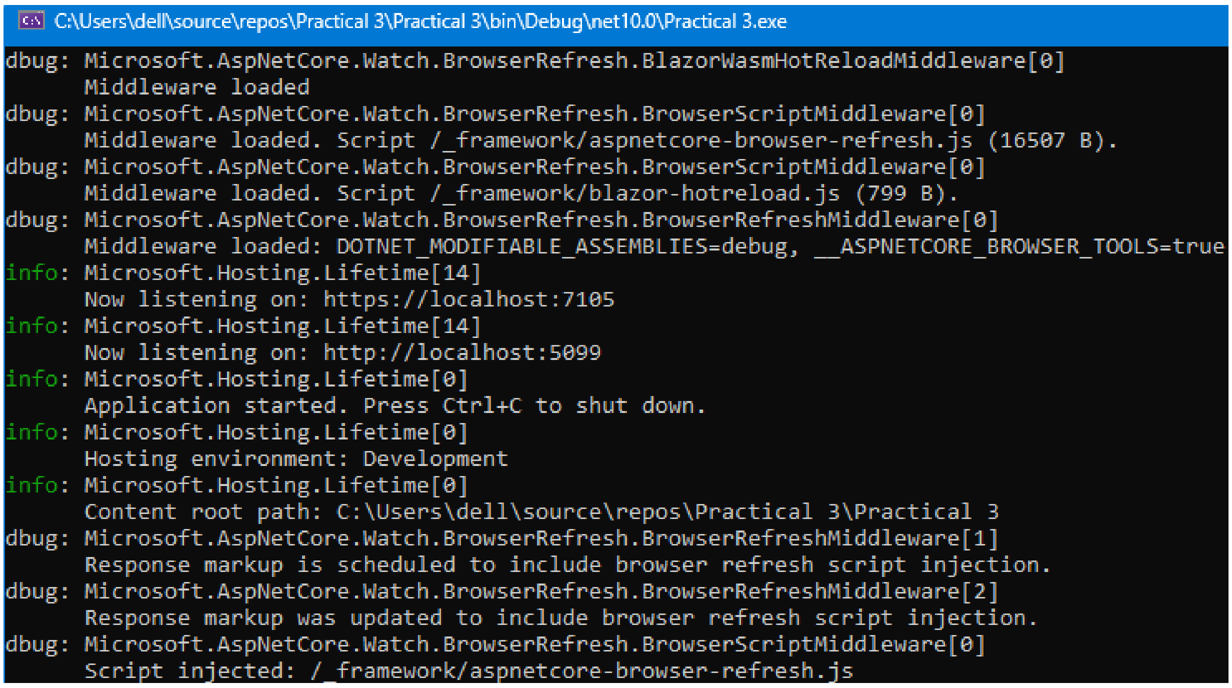
```
using Microsoft.AspNetCore.Mvc;

namespace MvcApp.Controllers
{
   [Route(“api/[controller]”)]
   [ApiController]
   public class ApiController : ControllerBase
   {
      [HttpGet(“data”)]
      public IactionResult GetData()
      {
         var response = new
         {
            Message = “Hello from the API!”,
            Timestamp = System.DateTime.UtcNow
         };

         return Ok(response);
      }
   }
}
```

Step 4:

- Save and Run the code

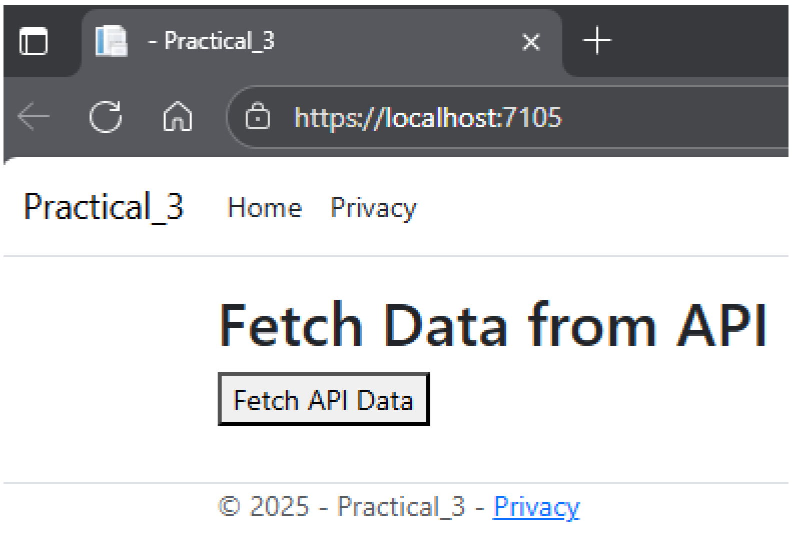


- OR Use the url of the command prompt in the browser and see the output



C:\Users\dell\source\repos\Practical 3\Practical 3\bin\Debug\net10.0\Practical 3.exe

```
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BlazorWasmHotReloadMiddleware[0]
      Middleware loaded
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserScriptMiddleware[0]
      Middleware loaded. Script /_framework/aspnetcore-browser-refresh.js (16507 B).
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserScriptMiddleware[0]
      Middleware loaded. Script /_framework/blazor-hotreload.js (799 B).
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserRefreshMiddleware[0]
      Middleware loaded: DOTNET_MODIFIABLE_ASSEMBLIES=debug, __ASPNETCORE_BROWSER_TOOLS=true
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7105
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5099
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: C:\Users\dell\source\repos\Practical 3\Practical 3
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserRefreshMiddleware[1]
      Response markup is scheduled to include browser refresh script injection.
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserRefreshMiddleware[2]
      Response markup was updated to include browser refresh script injection.
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserScriptMiddleware[0]
      Script injected: /_framework/aspnetcore-browser-refresh.js
```

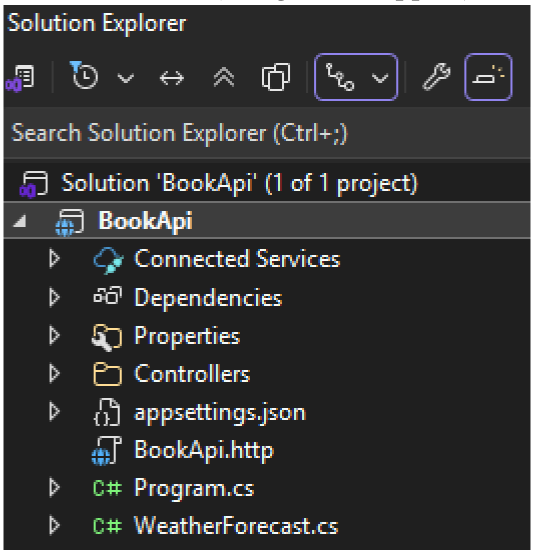


Practical No : 4

Aim: Building ASP.Net core REST API

Step 1:

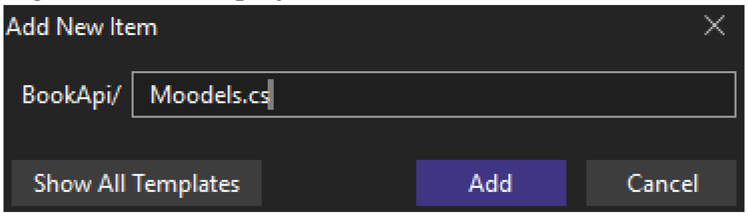
- Open Visual Studio → Create a new project→ Select ASP.NET Core Web API → Click Next.
- Name the project: `BookApi` → Click Next.
- Select .NET 8.0 (Long-term support) → Click Create.



Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'BookApi' (1 of 1 project)
BookApi
Connected Services
Dependencies
Properties
Controllers
appsettings.json
BookApi.http
Program.cs
WeatherForecast.cs

Step 2:

- Add Book Model (Models/Book.cs):
- Right click on the project→add → new item→enter

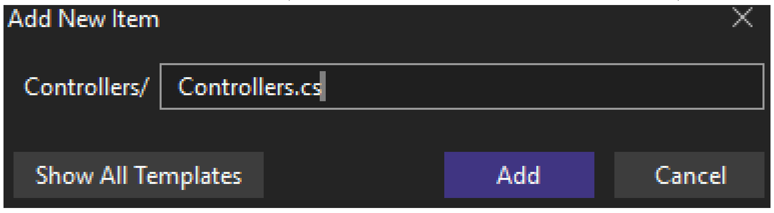


Add New Item
BookApi/ Moodels.cs
Show All Templates Add Cancel

```
namespace BookApi.Models
{
    public class Book
    {
        public int Id { get; set; }
        public string Title { get; set; } = string.Empty;
        public string Author { get; set; } = string.Empty;
        public int YearPublished { get; set; }
    }
}
```

Step 3:

- Add Book Controller (Controllers/BookController.cs):



Add New Item
Controllers/ Controllers.cs
Show All Templates Add Cancel

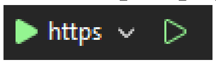
```
using Microsoft.AspNetCore.Mvc;
using BookApi.Models;
using System.Collections.Generic;
using System.Linq;

namespace BookApi.Controllers
```

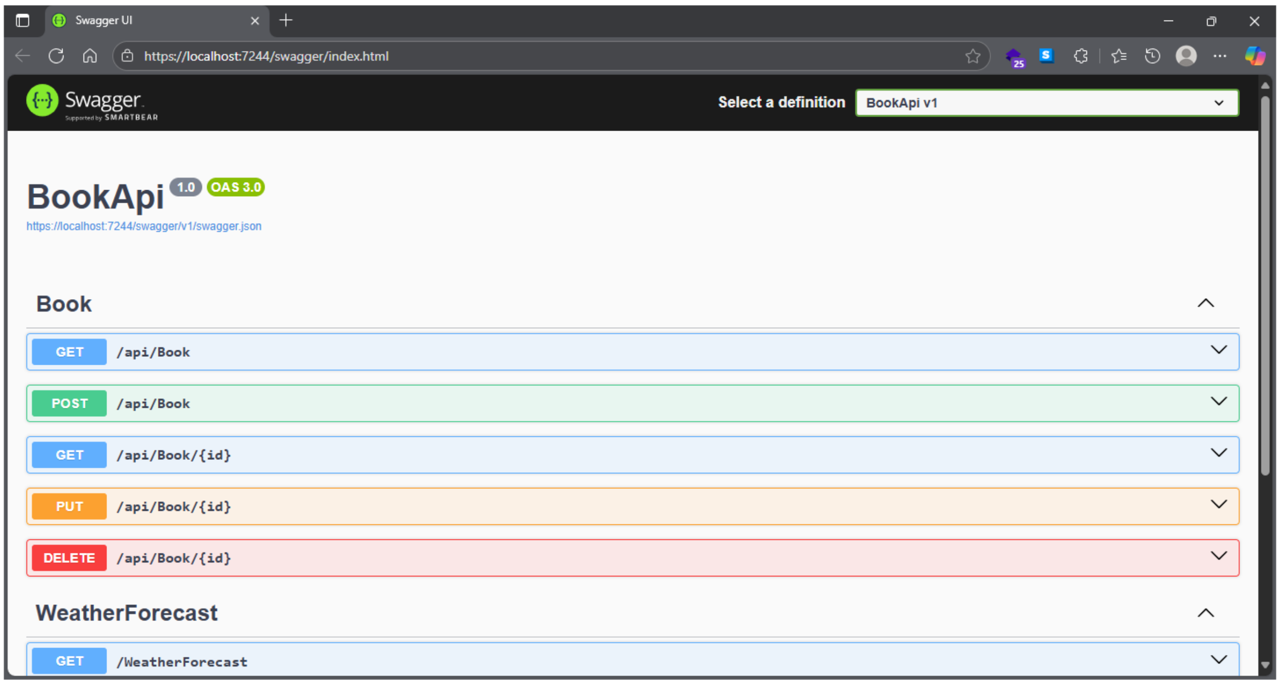
```
{
   [ApiController]
   [Route("api/[controller]")]
   public class BookController : ControllerBase
   {
      private static readonly List<Book> Books = new()
      {
         new Book { Id = 1, Title = "1984", Author = "George Orwell", YearPublished = 1949 },
         new Book { Id = 2, Title = "To Kill a Mockingbird", Author = "Harper Lee", YearPublished
= 1960 }
      };

      // GET: api/book
      [HttpGet]
      public ActionResult<IEnumerable<Book>> GetAllBooks() => Books;

      // GET: api/book/{id}
      [HttpGet("{id}")]
      public ActionResult<Book> GetBookById(int id)
      {
         var book = Books.FirstOrDefault(b => b.Id == id);
         return book is not null ? Ok(book) : NotFound();
      }

      // POST: api/book
      [HttpPost]
      public ActionResult<Book> CreateBook(Book book)
      {
         book.Id = Books.Max(b => b.Id) + 1;
         Books.Add(book);
         return CreatedAtAction(nameof(GetBookById), new { id = book.Id }, book);
      }

      // PUT: api/book/{id}
      [HttpPut("{id}")]
      public IActionResult UpdateBook(int id, Book updatedBook)
      {
         var book = Books.FirstOrDefault(b => b.Id == id);
         if (book is null) return NotFound();

         book.Title = updatedBook.Title;
         book.Author = updatedBook.Author;
         book.YearPublished = updatedBook.YearPublished;
         return NoContent();
      }

      // DELETE: api/book/{id}
      [HttpDelete("{id}")]
      public IActionResult DeleteBook(int id)
      {
         var book = Books.FirstOrDefault(b => b.Id == id);
         if (book is null) return NotFound();

         Books.Remove(book);
         return NoContent();
```

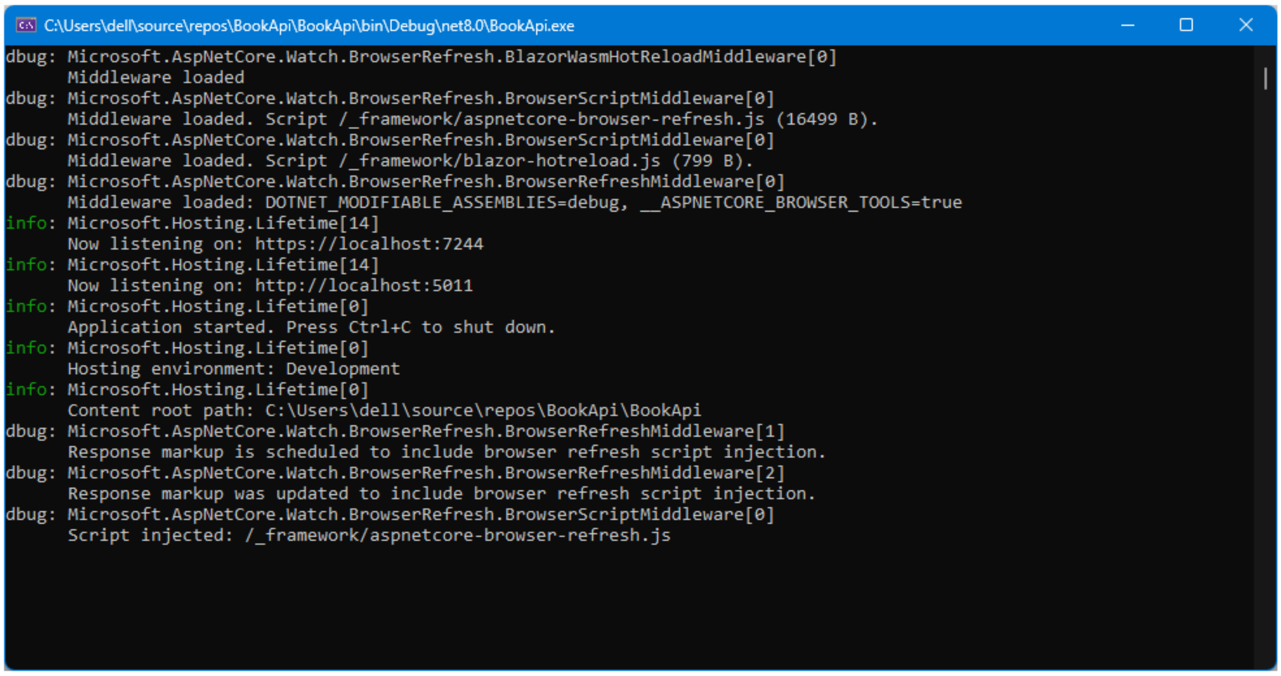
```
            }
        }
    }
```

Step 4:

- Select http/https port to run the project

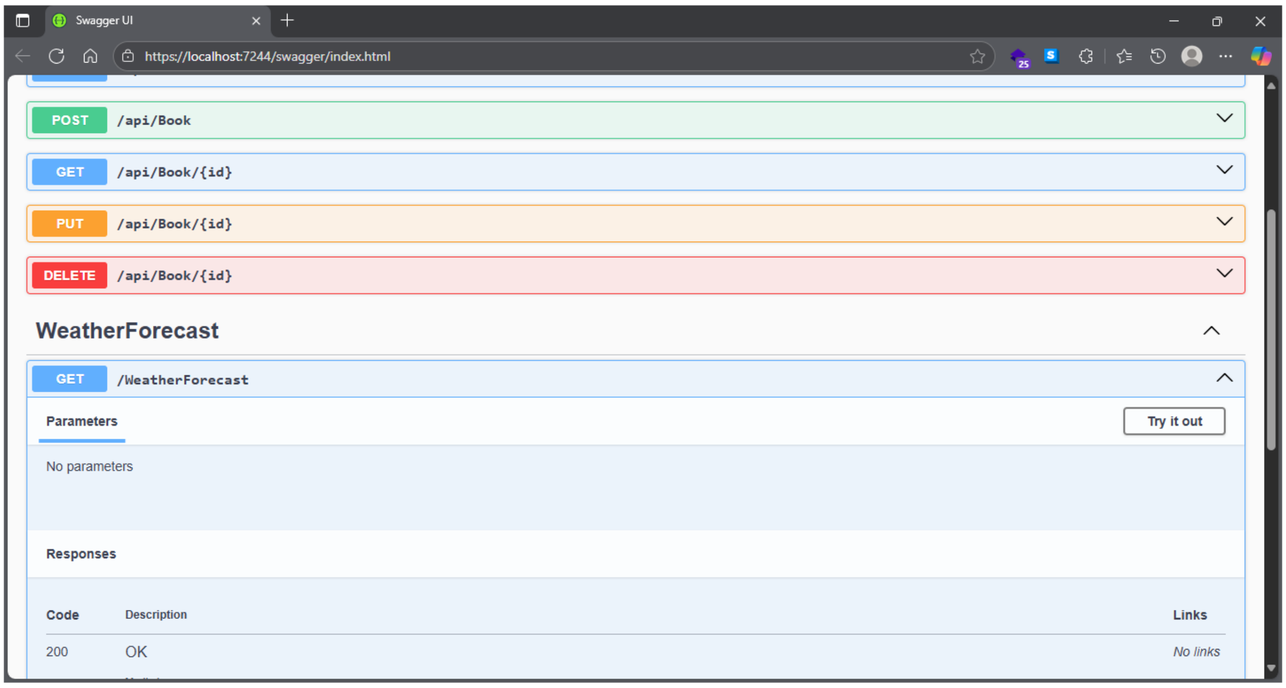





C:\Users\dell\source\repos\BookApi\BookApi\bin\Debug\net8.0\BookApi.exe

```
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BlazorWasmHotReloadMiddleware[0]
      Middleware loaded
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserScriptMiddleware[0]
      Middleware loaded. Script /_framework/aspnetcore-browser-refresh.js (16499 B).
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserScriptMiddleware[0]
      Middleware loaded. Script /_framework/blazor-hotreload.js (799 B).
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserRefreshMiddleware[0]
      Middleware loaded: DOTNET_MODIFIABLE_ASSEMBLIES=debug, __ASPNETCORE_BROWSER_TOOLS=true
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7244
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5011
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: C:\Users\dell\source\repos\BookApi\BookApi
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserRefreshMiddleware[1]
      Response markup is scheduled to include browser refresh script injection.
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserRefreshMiddleware[2]
      Response markup was updated to include browser refresh script injection.
dbug: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserScriptMiddleware[0]
      Script injected: /_framework/aspnetcore-browser-refresh.js
```

- Click on get→try it out→execute



POST /api/Book

GET /api/Book/{id}

PUT /api/Book/{id}

DELETE /api/Book/{id}

WeatherForecast

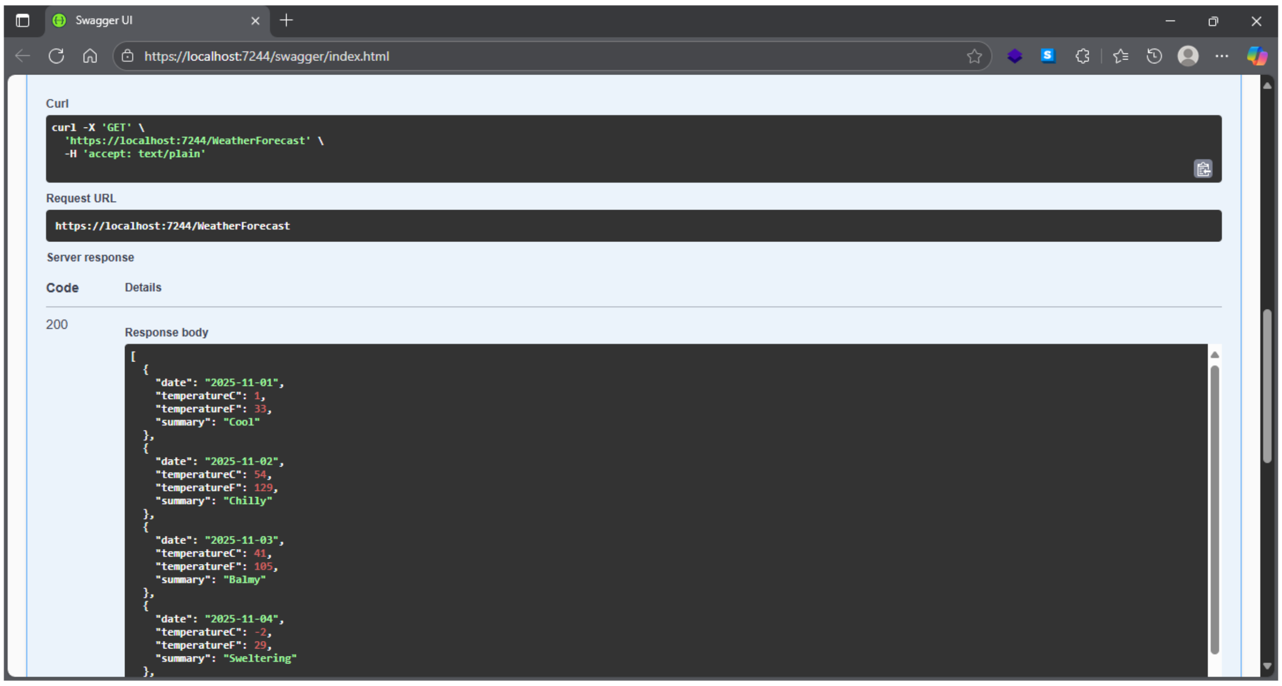
GET /WeatherForecast

Parameters

No parameters

Responses

Code	Description	Links
200	OK	No links



Curl

```
curl -X 'GET' \
  'https://localhost:7244/WeatherForecast' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:7244/WeatherForecast
```

Server response

Code Details

200 Response body

```
[
  {
    "date": "2025-11-01",
    "temperatureC": 1,
    "temperatureF": 33,
    "summary": "Cool"
  },
  {
    "date": "2025-11-02",
    "temperatureC": 54,
    "temperatureF": 129,
    "summary": "Chilly"
  },
  {
    "date": "2025-11-03",
    "temperatureC": 41,
    "temperatureF": 105,
    "summary": "Balmy"
  },
  {
    "date": "2025-11-04",
    "temperatureC": -2,
    "temperatureF": 29,
    "summary": "Sweltering"
  },
```

Practical No : 5

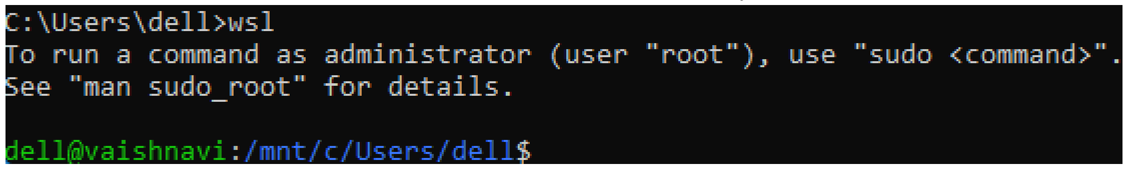
Aim : Working with Docker Desktop – Docker installation and commands

Step 1: Open The Windows Features On Or Tick the

- Virtual machine Platform and
- Windows Subsystem For Linux
- Windows Hypervisor Platform

Step 2: Install WSL

- Open PowerShell or Windows Command Prompt in administrator mode.
- Enter the 'wsl' command. WSL stands for Windows Subsystem for Linux.

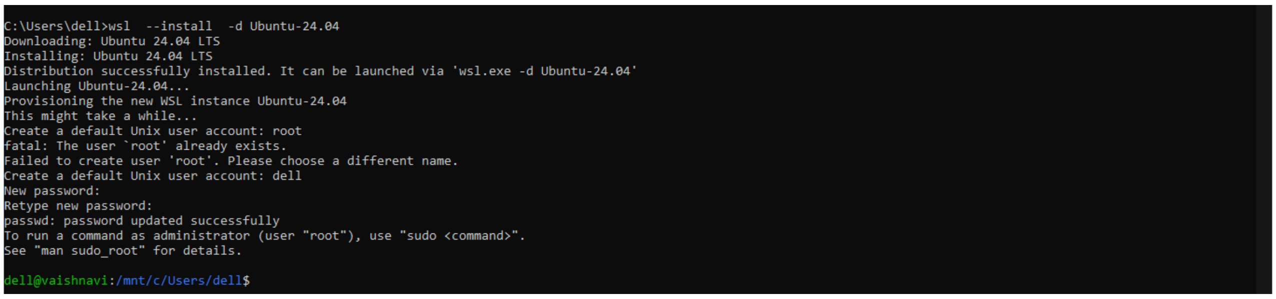


```
C:\Users\dell>wsl
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

dell@vaishnavi:/mnt/c/Users/dell$
```

- And Then Close the Power Shall Open new Power shell

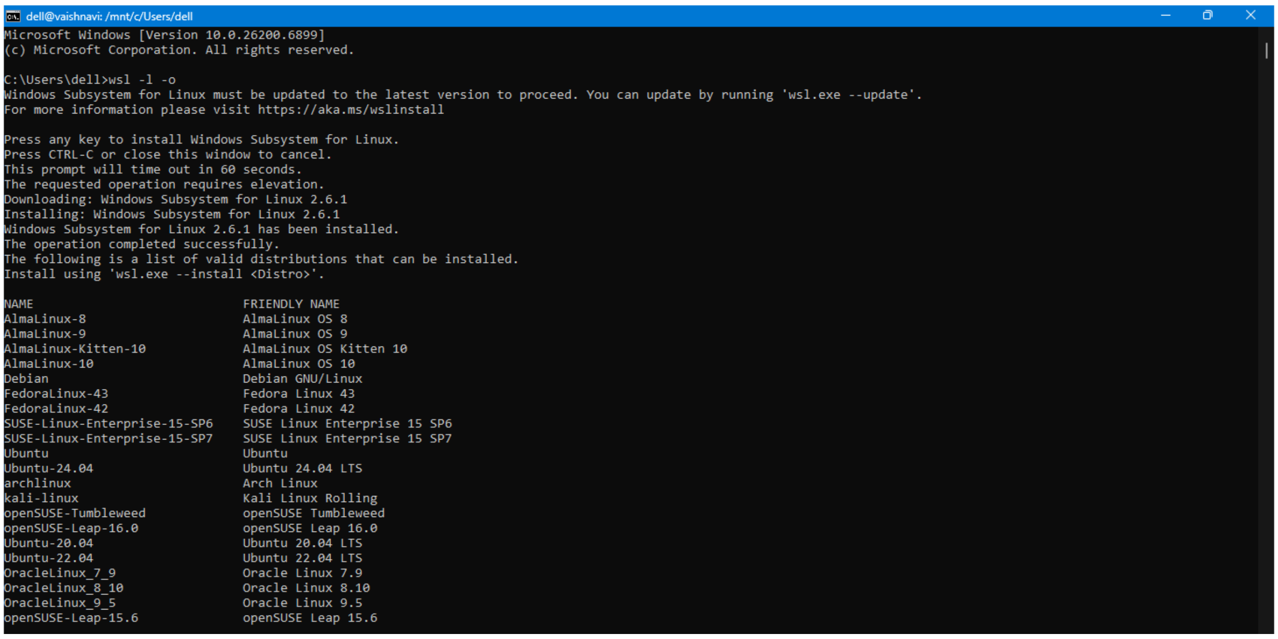
Step 3: To install Linux distributions, use the command: 'wsl --install -d '.

- The command is – 'wsl --install -d Ubuntu-24.04'



```
C:\Users\dell>wsl  --install  -d Ubuntu-24.04
Downloading: Ubuntu 24.04 LTS
Installing: Ubuntu 24.04 LTS
Distribution successfully installed. It can be launched via 'wsl.exe -d Ubuntu-24.04'
Launching Ubuntu-24.04...
Provisioning the new WSL instance Ubuntu-24.04
This might take a while...
Create a default Unix user account: root
fatal: The user `root' already exists.
Failed to create user 'root'. Please choose a different name.
Create a default Unix user account: dell
New password:
Retype new password:
passwd: password updated successfully
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

dell@vaishnavi:/mnt/c/Users/dell$
```

- To see a list of available Linux distributions available for download through the online store, enter: 'wsl -l -o'



```
dell@vaishnavi: /mnt/c/Users/dell
Microsoft Windows [Version 10.0.26200.6899]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dell>wsl -l -o
Windows Subsystem for Linux must be updated to the latest version to proceed. You can update by running 'wsl.exe --update'.
For more information please visit https://aka.ms/wslinstall

Press any key to install Windows Subsystem for Linux.
Press CTRL-C or close this window to cancel.
This prompt will time out in 60 seconds.
The requested operation requires elevation.
Downloading: Windows Subsystem for Linux 2.6.1
Installing: Windows Subsystem for Linux 2.6.1
Windows Subsystem for Linux 2.6.1 has been installed.
The operation completed successfully.
The following is a list of valid distributions that can be installed.
Install using 'wsl.exe --install <Distro>'.

NAME                            FRIENDLY NAME
AlmaLinux-8                     AlmaLinux OS 8
AlmaLinux-9                     AlmaLinux OS 9
AlmaLinux-Kitten-10             AlmaLinux OS Kitten 10
AlmaLinux-10                    AlmaLinux OS 10
Debian                          Debian GNU/Linux
FedoraLinux-43                  Fedora Linux 43
FedoraLinux-42                  Fedora Linux 42
SUSE-Linux-Enterprise-15-SP6    SUSE Linux Enterprise 15 SP6
SUSE-Linux-Enterprise-15-SP7    SUSE Linux Enterprise 15 SP7
Ubuntu                          Ubuntu
Ubuntu-24.04                    Ubuntu 24.04 LTS
archlinux                       Arch Linux
kali-linux                      Kali Linux Rolling
openSUSE-Tumbleweed             openSUSE Tumbleweed
openSUSE-Leap-16.0              openSUSE Leap 16.0
Ubuntu-20.04                    Ubuntu 20.04 LTS
Ubuntu-22.04                    Ubuntu 22.04 LTS
OracleLinux_7_9                 Oracle Linux 7.9
OracleLinux_8_10                Oracle Linux 8.10
OracleLinux_9_5                 Oracle Linux 9.5
openSUSE-Leap-15.6              openSUSE Leap 15.6
```

Step 4: After successful installation of the Linux distribution, the following window will appear. Set username and password for the Windows Subsystem for Linux.
Note: The password will not be visible when you enter it.

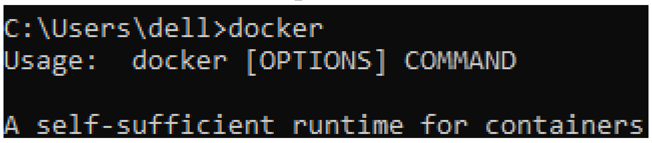
Step 5: Installation of Docker Desktop

- Download Docker Desktop using the following link:
  https://www.docker.com/products/docker-desktop/
- Run the docker installation exe as administrator.

Step 6: Select the checkbox on the configuration window.
After installation, run Docker Desktop. Accept the terms and conditions.

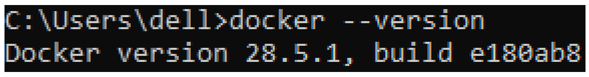
Step 7: Verify whether Docker installation is successful.

- Go to Command Prompt and enter: docker



```
C:\Users\dell>docker
Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers
```

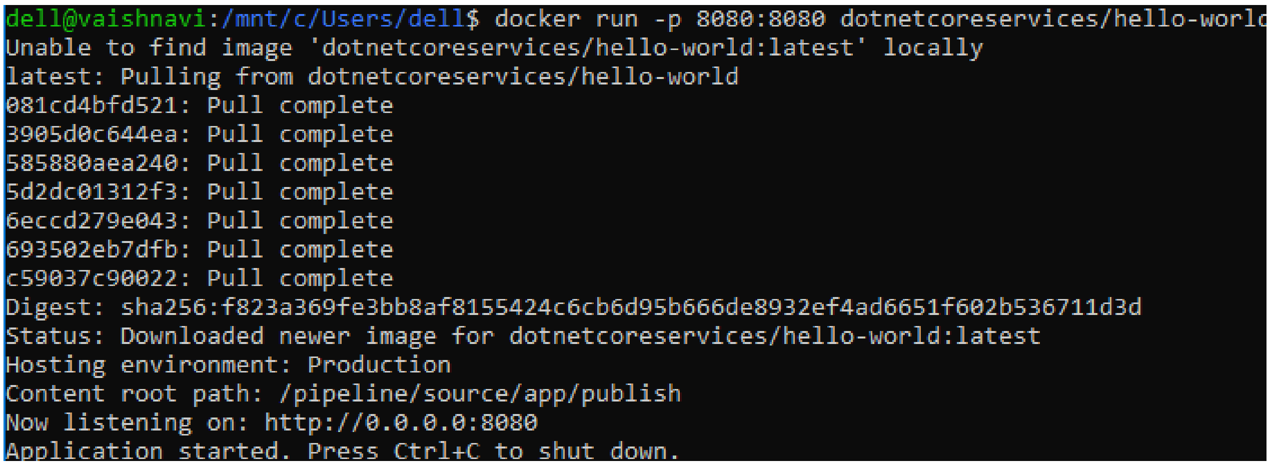
- To check version of Docker, enter the following command: docker –version



```
C:\Users\dell>docker --version
Docker version 28.5.1, build e180ab8
```

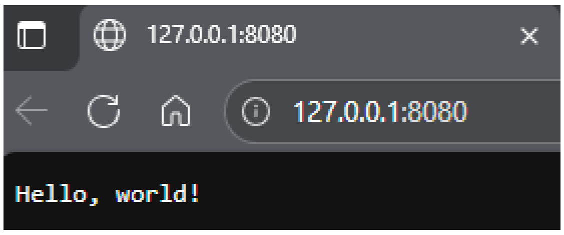
Step 8: Publishing an existing image into Docker container

- To publish an existing docker 'hello world' image into a docker container, use: docker run -p 8080:8080 dotnetcoreservices/hello-world



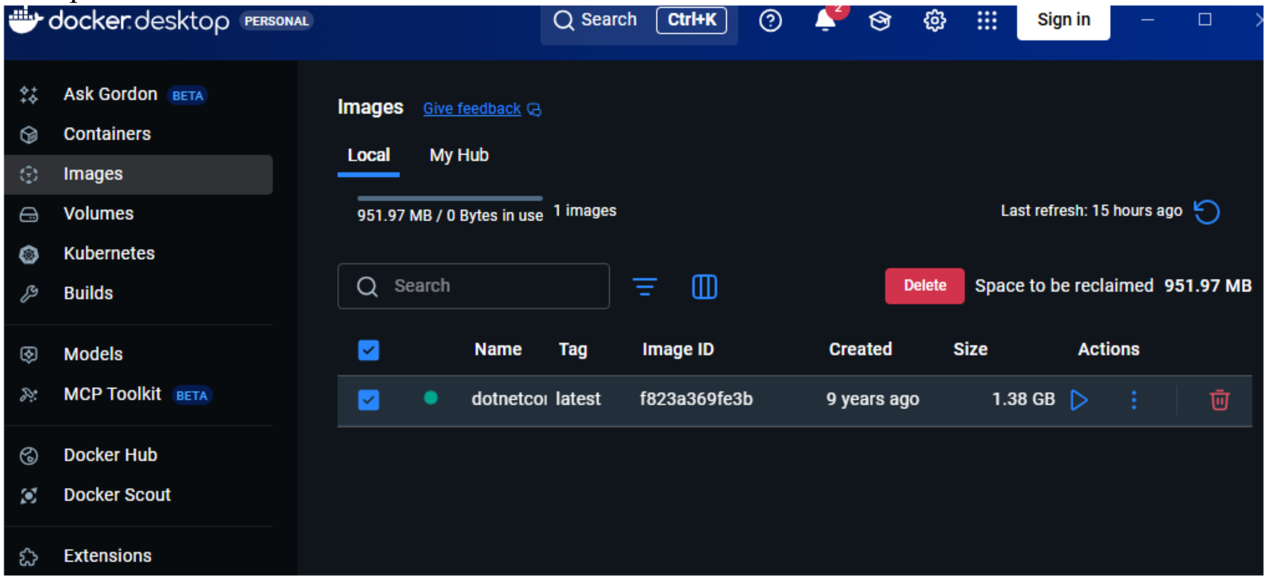
```
dell@vaishnavi:/mnt/c/Users/dell$ docker run -p 8080:8080 dotnetcoreservices/hello-world
Unable to find image 'dotnetcoreservices/hello-world:latest' locally
latest: Pulling from dotnetcoreservices/hello-world
081cd4bfd521: Pull complete
3905d0c644ea: Pull complete
585880aea240: Pull complete
5d2dc01312f3: Pull complete
6eccd279e043: Pull complete
693502eb7dfb: Pull complete
c59037c90022: Pull complete
Digest: sha256:f823a369fe3bb8af8155424c6cb6d95b666de8932ef4ad6651f602b536711d3d
Status: Downloaded newer image for dotnetcoreservices/hello-world:latest
Hosting environment: Production
Content root path: /pipeline/source/app/publish
Now listening on: http://0.0.0.0:8080
Application started. Press Ctrl+C to shut down.
```

- Copy the URL from the above output and paste it in any browser. Change the IP address in the URL to '127.0.0.1'.



```
Hello, world!
```

- Also if you go to Docker Desktop ,in the Images tab, you will see that the hello-world image has been published in the container.



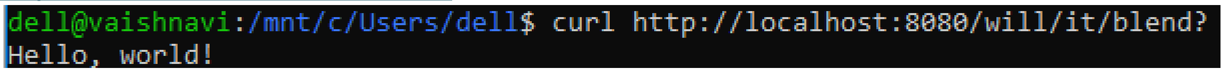
951.97 MB / 0 Bytes in use 1 images

Last refresh: 15 hours ago

Space to be reclaimed 951.97 MB

Name	Tag	Image ID	Created	Size	Actions
dotnetcoı	latest	f823a369fe3b	9 years ago	1.38 GB	

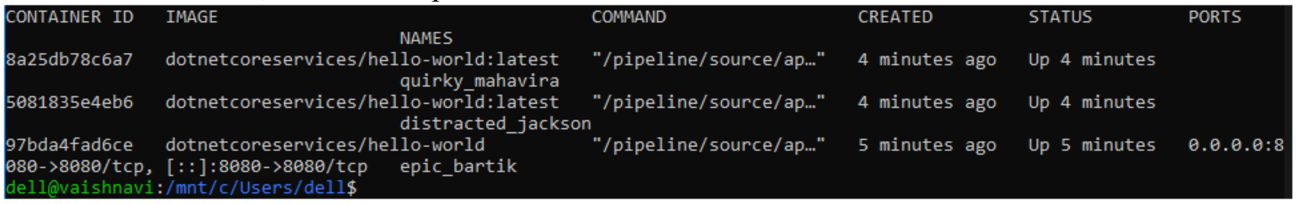
- curl or ClientURL is a command line tool that enables data exchange between a device and a server through a terminal. To run the image in the command prompt enter: curl http://localhost:8080/will/it/blend?



```
dell@vaishnavi:/mnt/c/Users/dell$ curl http://localhost:8080/will/it/blend?
Hello, world!
```

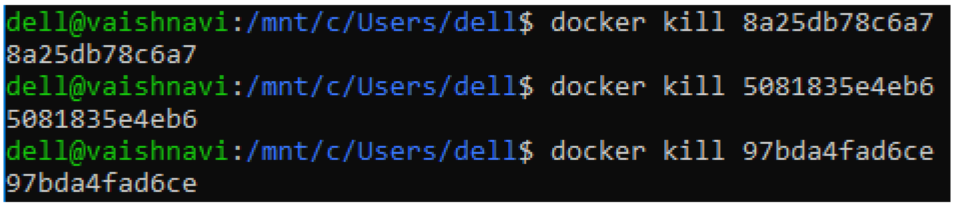
Step 9: Basic Docker commands

- To list all containers, use: docker ps



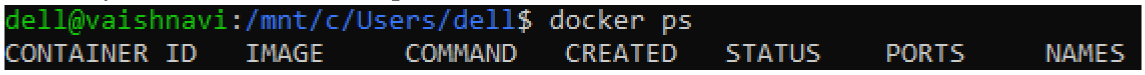
```
CONTAINER ID   IMAGE                            COMMAND                  CREATED         STATUS         PORTS
                                                NAMES
8a25db78c6a7   dotnetcoreservices/hello-world:latest   "/pipeline/source/ap…"   4 minutes ago   Up 4 minutes
                                                quirky_mahavira
5081835e4eb6   dotnetcoreservices/hello-world:latest   "/pipeline/source/ap…"   4 minutes ago   Up 4 minutes
                                                distracted_jackson
97bda4fad6ce   dotnetcoreservices/hello-world          "/pipeline/source/ap…"   5 minutes ago   Up 5 minutes   0.0.0.0:8
080->8080/tcp, [::]:8080->8080/tcp   epic_bartik
dell@vaishnavi:/mnt/c/Users/dell$
```

- To delete or kill a Docker container use: docker kill



```
dell@vaishnavi:/mnt/c/Users/dell$ docker kill 8a25db78c6a7
8a25db78c6a7
dell@vaishnavi:/mnt/c/Users/dell$ docker kill 5081835e4eb6
5081835e4eb6
dell@vaishnavi:/mnt/c/Users/dell$ docker kill 97bda4fad6ce
97bda4fad6ce
```

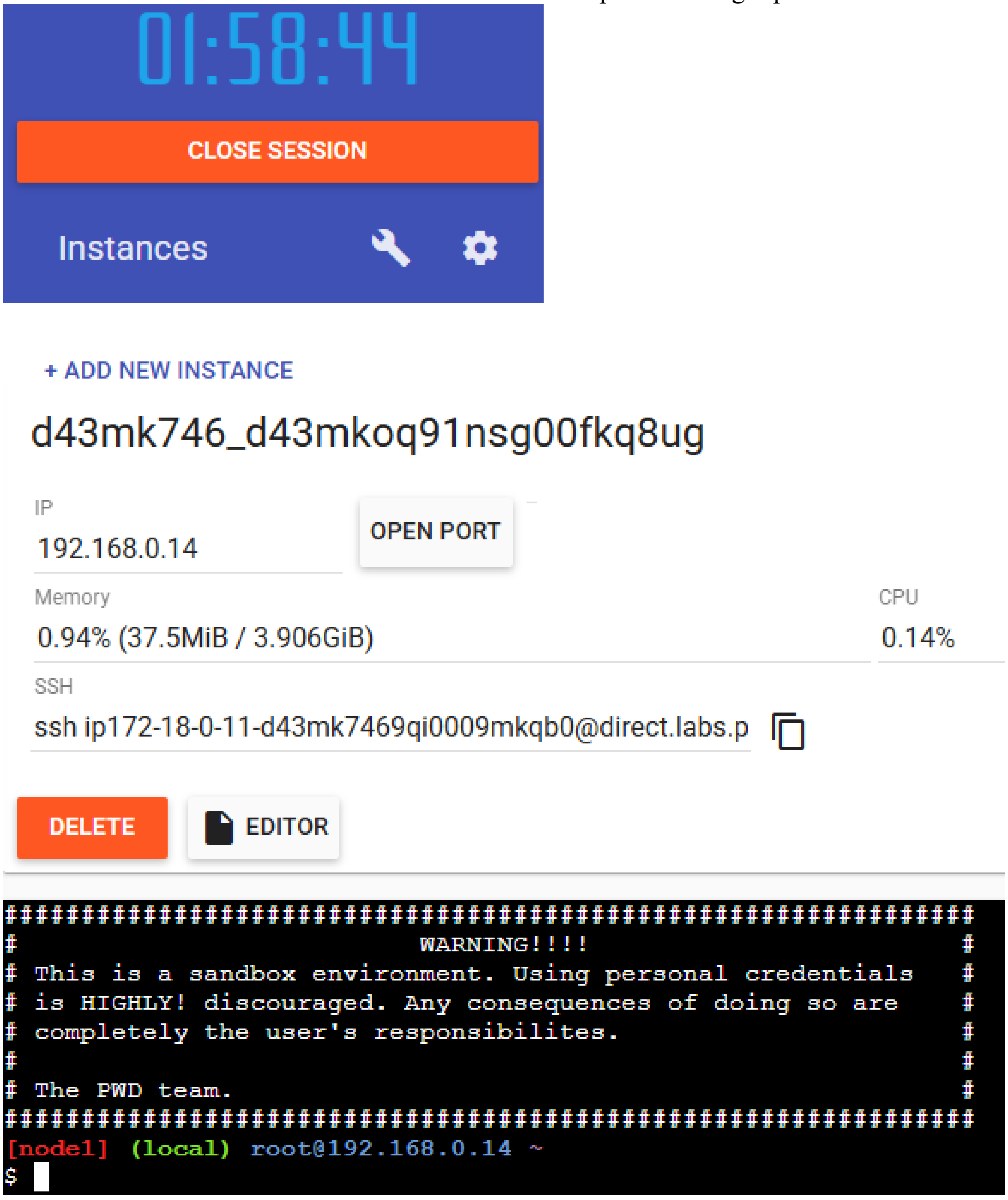
- To verify deletion use: docker ps



```
dell@vaishnavi:/mnt/c/Users/dell$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Practical No : 6

Aim : Usage of Docker - Docker Hub commands and its output
Important Note : Remember we will be using the Account credentials of Docker Desktop only while logging in for both Docker Playground and Docker Hub**.**
Note : Docker Playground is a cloud-based tool that allows you to experiment with Docker containers in a web browser without the need to rely on Docker Desktop. It behaves just like a Docker installation on your local machine.
Step 2: In Your Docker Desktop go to Account Settings , it will take you to webpage, sign in with the same credentials of your Docker Desktop.
Step 3: Go to Account Settings. Reset your password (or Set it if it's for first time)
Note: Remember this password, it will be used later in Docker playground
Step 4: Now to use Docker Playground go to https://labs.play-with-docker.com/, login with your Docker Desktop Credentials. Click on Start button.
Step 5:
• Click on Add New Instance. You will see the editor open in the right pane.



d43mk746_d43mkoq91nsg00fkq8ug

IP
192.168.0.14

Memory
0.94% (37.5MiB / 3.906GiB)

CPU
0.14%

SSH
ssh ip172-18-0-11-d43mk7469qi0009mkqb0@direct.labs.p

```
###############################################################
#                          WARNING!!!!                         #
# This is a sandbox environment. Using personal credentials    #
# is HIGHLY! discouraged. Any consequences of doing so are     #
# completely the user's responsibilites.                       #
#                                                              #
# The PWD team.                                                #
###############################################################
[node1] (local) root@192.168.0.14 ~
$ 
```

• Commands will be given in the black terminal or console.

• To check the version of the docker type the command: docker –version

```
$ docker --version
Docker version 27.3.1, build ce12230
[node1] (local) root@192.168.0.14 ~
```

• To pull the ready made image type the command: docker pull hello-world

```
$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
17eec7bbc9d7: Pull complete
Digest: sha256:56433a6be3fda188089fb548eae3d91df3ed0d6589f7c2656121b911198df065
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
[node1] (local) root@192.168.0.14 ~
```

• To check the images in docker type the command : docker images

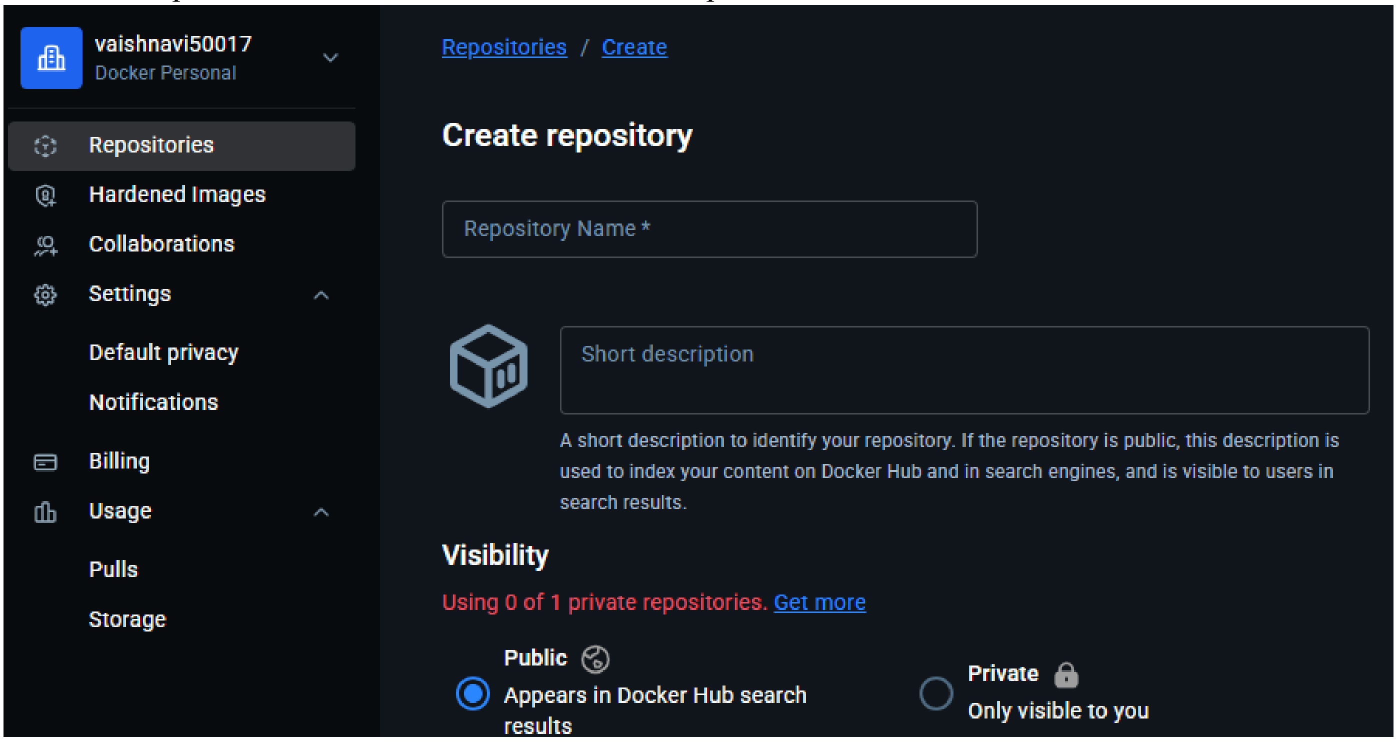
```
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    1b44b5a3e06a   2 months ago   10.1kB
[node1] (local) root@192.168.0.14 ~
```

Step 6 :

Part 1: To pull and Push images in docker:

• Open the new tab in the browser and go to https://hub.docker.com/ , sign in with your Docker Desktop account.

• Click on Repositories tab and then click on Create Repositories



Create repository

Repository Name *

Short description

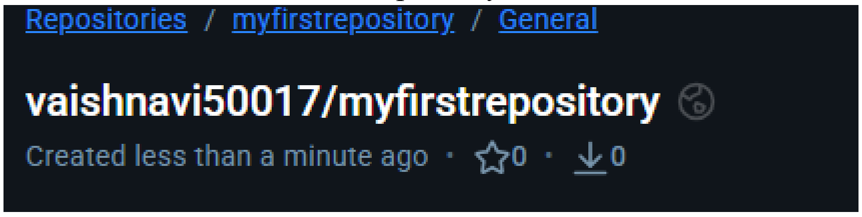
A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.

Visibility

Using 0 of 1 private repositories. Get more

[✓ Public Appears in Docker Hub search results] Private Only visible to you

• Give any name to your repository and in description add “My first repository”. Also Make visibility as Private , click on create , the repository will be created.


Repositories / myfirstrepository / General

vaishnavi50017/myfirstrepository

Created less than a minute ago · ☆0 · ↓0

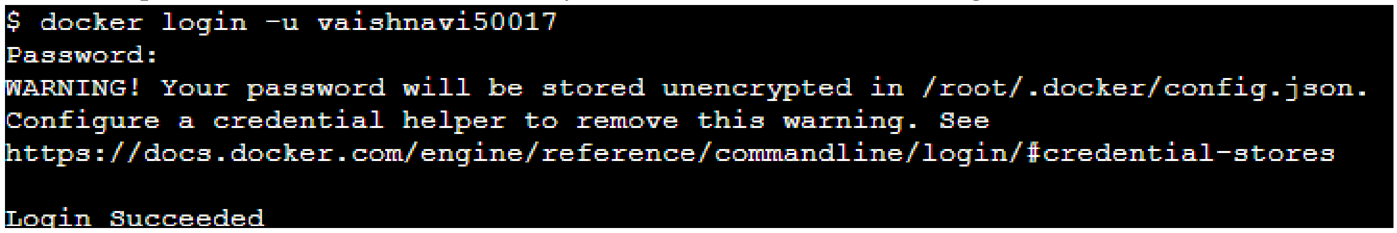
• Now come back to Docker Playground and give following command to login to your Docker Account :

```
docker login -u <username>
```

• Put your username above , you will find username in your docker desktop’s Account Panel

• You will be prompted to enter a password , now enter the password which you created in Account Settings at the beginning

Note: The password won't be visible while you enter it, so continue entering it.

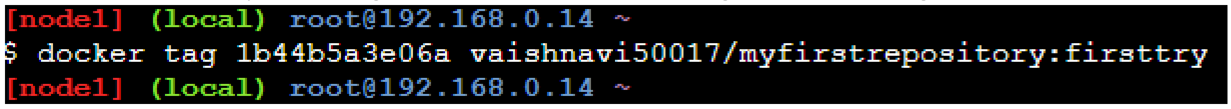


```
$ docker login -u vaishnavi50017
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

Login Succeeded
```

• To see earlier pulled image and its image id enter the command : docker images
• To tag that image in docker type command:

```
docker tag  <image_id>  <username>/<name_of_your_repository>:firsttry
```

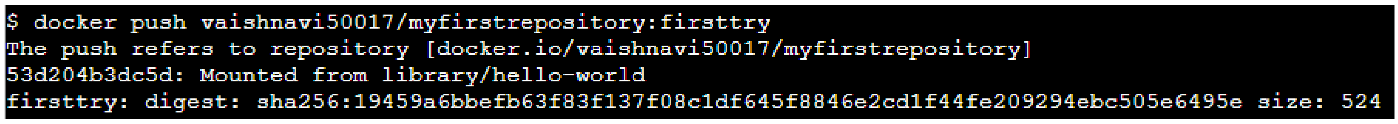
• Put the image id of the image , your username and the name of your repository which you created in Docker Hub, firsttry is the tag name which we have assigned to the image.



```
[node1] (local) root@192.168.0.14 ~
$ docker tag 1b44b5a3e06a vaishnavi50017/myfirstrepository:firsttry
[node1] (local) root@192.168.0.14 ~
```

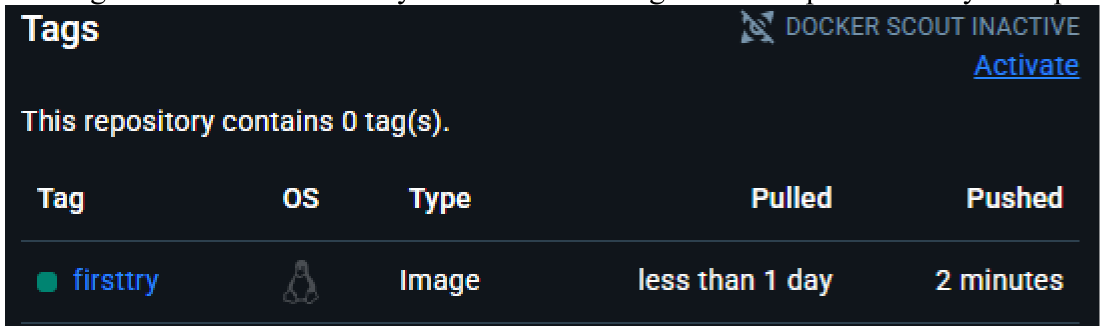
• To push the image to your docker hub use command (put your username):

```
docker push <username>/<name_of_your_repository>:firsttry
```



```
$ docker push vaishnavi50017/myfirstrepository:firsttry
The push refers to repository [docker.io/vaishnavi50017/myfirstrepository]
53d204b3dc5d: Mounted from library/hello-world
firsttry: digest: sha256:19459a6bbefb63f83f137f08c1df645f8846e2cd1f44fe209294ebc505e6495e size: 524
```

• Now go back to Docker Hub you can see the image has been pushed into your repository



Tags

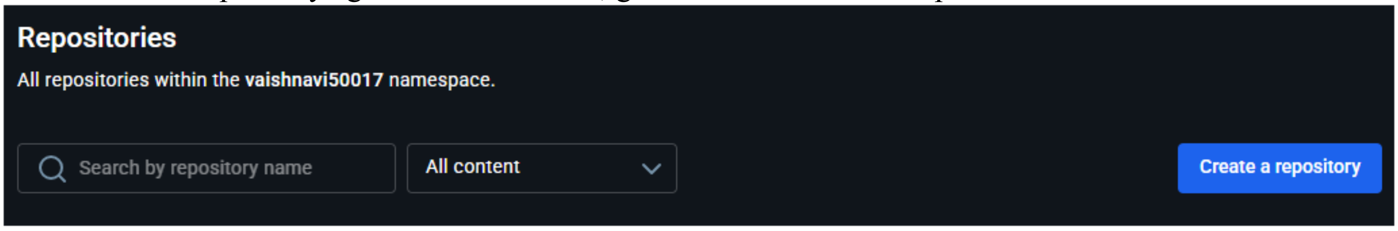
This repository contains 0 tag(s).

Tag	OS	Type	Pulled	Pushed
🟩 firsttry		Image	less than 1 day	2 minutes

Step 7 :

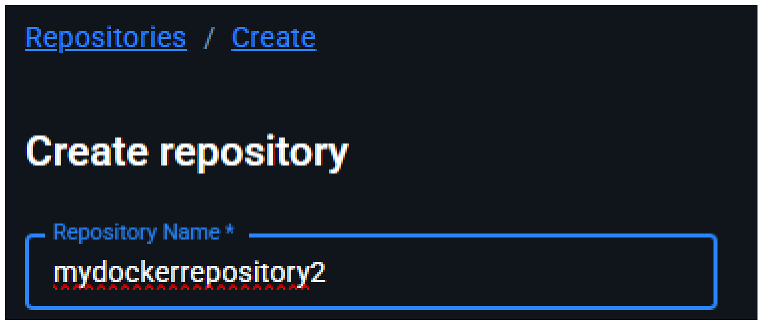
Part 2: Build an image and then push and run in the docker :

• Create a new repository again in Docker Hub, give it a name and description.



Repositories

All repositories within the **vaishnavi50017** namespace.

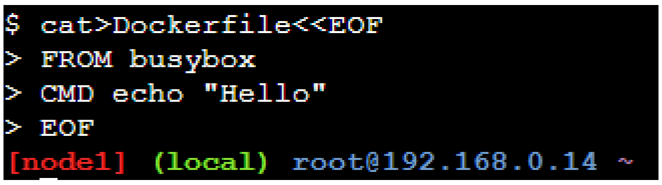


Create repository

Repository Name *: mydockerrepository2

• In Docker Playground type command :

```
cat >Dockerfile<<EOF
FROM busybox
CMD echo "Hello world! This is my first Docker image."
EOF
```

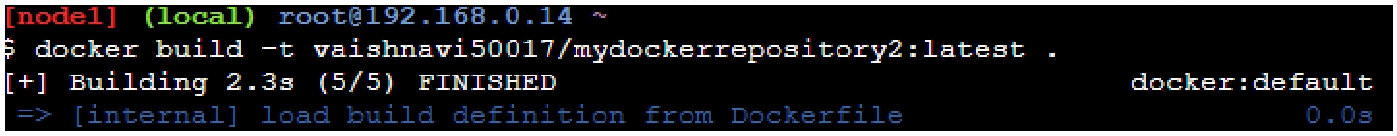


```
$ cat>Dockerfile<<EOF
> FROM busybox
> CMD echo "Hello"
> EOF
[node1] (local) root@192.168.0.14 ~
```

• To build the image from docker file use command:

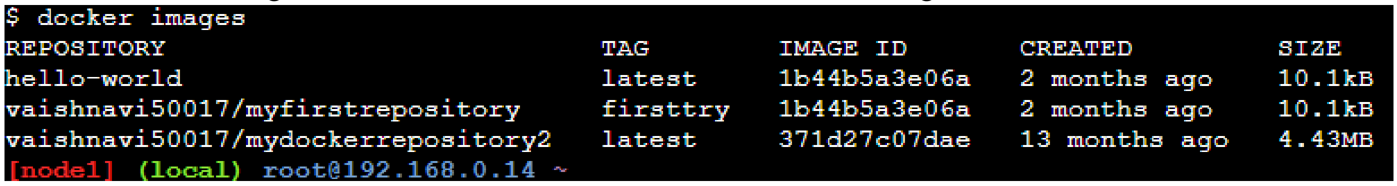
docker build –t <username>/<repository_name>:latest .

• Give your username and the repository name which you just created above , latest is the tag name here



```
[node1] (local) root@192.168.0.14 ~
$ docker build -t vaishnavi50017/mydockerrepository2:latest .
[+] Building 2.3s (5/5) FINISHED                                    docker:default
 => [internal] load build definition from Dockerfile                          0.0s
```

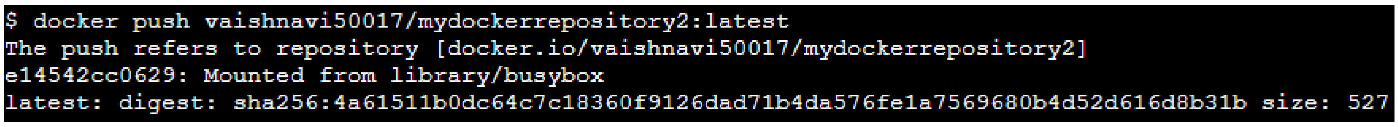
• To check if the image has been built or not use command : docker images



```
$ docker images
REPOSITORY                           TAG        IMAGE ID       CREATED         SIZE
hello-world                          latest     1b44b5a3e06a   2 months ago    10.1kB
vaishnavi50017/myfirstrepository     firsttry   1b44b5a3e06a   2 months ago    10.1kB
vaishnavi50017/mydockerrepository2   latest     371d27c07dae   13 months ago   4.43MB
[node1] (local) root@192.168.0.14 ~
```

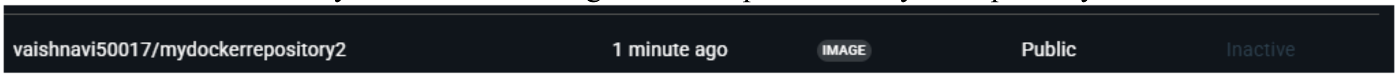
• To push this newly created image to your docker hub use command (put your username):

docker push <username>/<name_of_your_repository>:latest



```
$ docker push vaishnavi50017/mydockerrepository2:latest
The push refers to repository [docker.io/vaishnavi50017/mydockerrepository2]
e14542cc0629: Mounted from library/busybox
latest: digest: sha256:4a61511b0dc64c7c18360f9126dad71b4da576fe1a7569680b4d52d616d8b31b size: 527
```

• Go back to Docker Hub you can see the image has been pushed into your repository

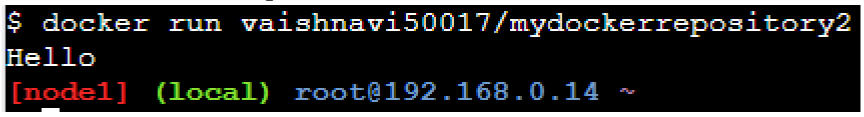


vaishnavi50017/mydockerrepository2 1 minute ago IMAGE Public Inactive

• Now to run the docker image run the command in Docker playground :

docker run <username>/<repository_name>

• You can see the output. Close the session

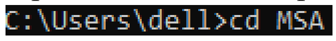


```
$ docker run vaishnavi50017/mydockerrepository2
Hello
[node1] (local) root@192.168.0.14 ~
```

Practical No : 7

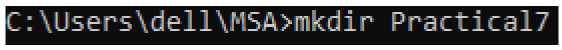
Aim : Running a Asp.net Core Console Application in Docker Container

Step 1: Open Command Prompt and navigate to the path where you would like to create your Application



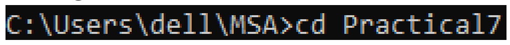
```
C:\Users\dell>cd MSA
```

Step 2: Create a new folder named "Practical7" using the command: mkdir Practical7



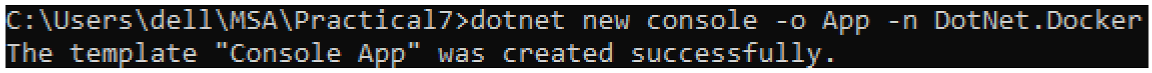
```
C:\Users\dell\MSA>mkdir Practical7
```

Step 3: Navigate to this folder



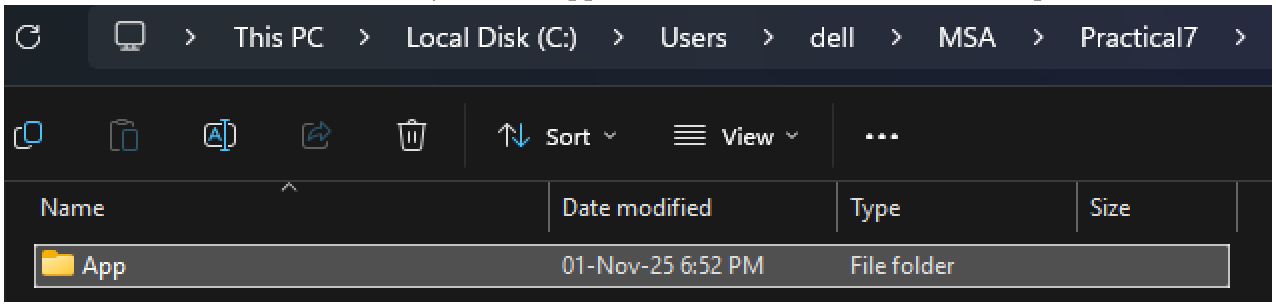
```
C:\Users\dell\MSA>cd Practical7
```

Step 4: In this path Enter the following command to create the dotnet application : dotnet new console -o App -n DotNet.Docker



```
C:\Users\dell\MSA\Practical7>dotnet new console -o App -n DotNet.Docker
The template "Console App" was created successfully.
```

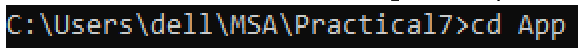
Step 5: You can see all the files necessary for the Application have been created in the specified location



This PC > Local Disk (C:) > Users > dell > MSA > Practical7 >

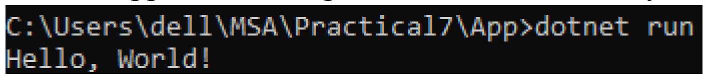
Name	Date modified	Type	Size
App	01-Nov-25 6:52 PM	File folder	

Step 6: Now navigate to the "App" folder where all these files have been created ,remember we will be running all the commands within this path only.



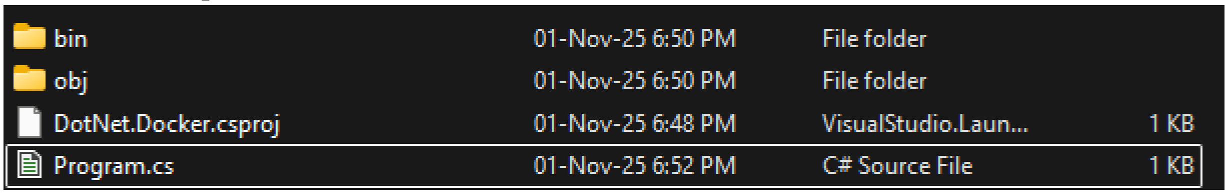
```
C:\Users\dell\MSA\Practical7>cd App
```

Step 7: Run the application using command: 'dotnet run' you can see the output :



```
C:\Users\dell\MSA\Practical7\App>dotnet run
Hello, World!
```

Step 8: Now go to the path where our dotnet application was created & get inside the "App" folder and open Program.cs file in Notepad

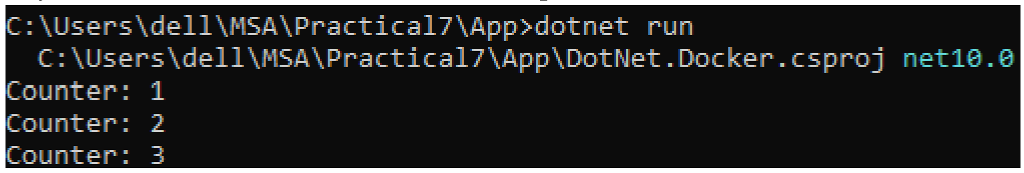


bin	01-Nov-25 6:50 PM	File folder	
obj	01-Nov-25 6:50 PM	File folder	
DotNet.Docker.csproj	01-Nov-25 6:48 PM	VisualStudio.Laun...	1 KB
Program.cs	01-Nov-25 6:52 PM	C# Source File	1 KB

Step 9: Replace any existing code in this file with the following code, This code counts numbers every second :

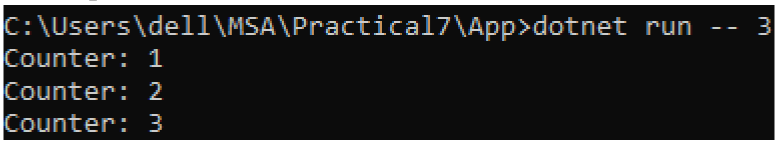
```
var counter = 0;
var max = args.Length is not 0 ? Convert.ToInt32(args[0]) : -1;
while (max is -1 || counter < max)
{
Console.WriteLine($"Counter: {++counter}");
await Task.Delay(TimeSpan.FromMilliseconds(1_000));
}
```

Step 10: Save the file and run the application in command prompt using 'dotnet run'. This app runs indefinitely so use cancel command Ctrl+C to stop it.



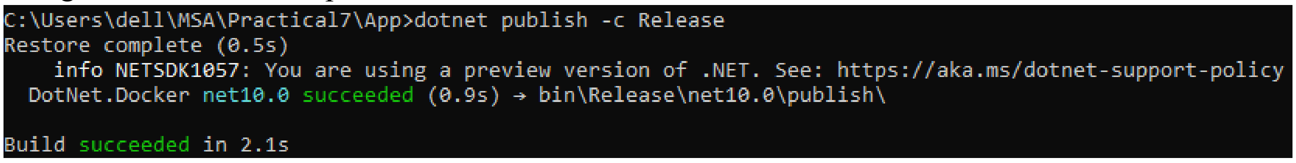
```
C:\Users\dell\MSA\Practical7\App>dotnet run
  C:\Users\dell\MSA\Practical7\App\DotNet.Docker.csproj net10.0
Counter: 1
Counter: 2
Counter: 3
```

Step 11: If you pass a number on the command line to the app, it will only count up to that amount and then exit. For example: 'dotnet run -- 3'



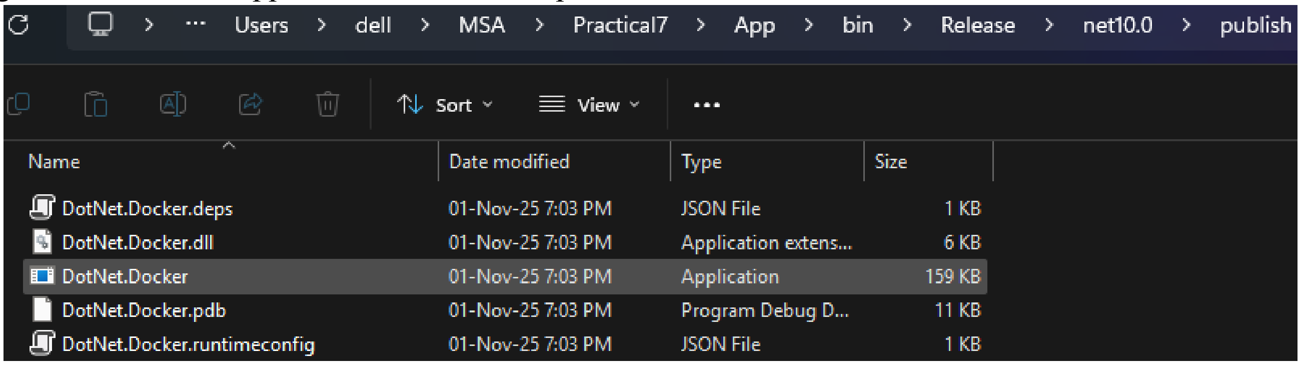
```
C:\Users\dell\MSA\Practical7\App>dotnet run -- 3
Counter: 1
Counter: 2
Counter: 3
```

Step 12: Before adding the .NET app to the Docker image, first it must be published. To publish the app, run the following command: 'dotnet publish -c Release'



```
C:\Users\dell\MSA\Practical7\App>dotnet publish -c Release
Restore complete (0.5s)
    info NETSDK1057: You are using a preview version of .NET. See: https://aka.ms/dotnet-support-policy
  DotNet.Docker net10.0 succeeded (0.9s) → bin\Release\net10.0\publish\

Build succeeded in 2.1s
```

Step 13: This command compiles your app to the publish folder. The path to the publish folder from the working folder will be ".\App\bin\Release\net10.0\publish\".



Users > dell > MSA > Practical7 > App > bin > Release > net10.0 > publish

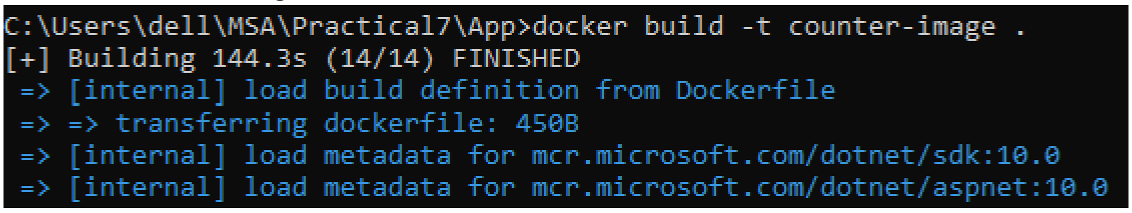
Name	Date modified	Type	Size
DotNet.Docker.deps	01-Nov-25 7:03 PM	JSON File	1 KB
DotNet.Docker.dll	01-Nov-25 7:03 PM	Application extens...	6 KB
DotNet.Docker	01-Nov-25 7:03 PM	Application	159 KB
DotNet.Docker.pdb	01-Nov-25 7:03 PM	Program Debug D...	11 KB
DotNet.Docker.runtimeconfig	01-Nov-25 7:03 PM	JSON File	1 KB

Step 14: Open Notepad → Create a New file and write the following code in it :

```
FROM mcr.microsoft.com/dotnet/sdk:10.0 AS build-env
WORKDIR /App
# Copy everything
COPY . ./
# Restore as distinct layers
RUN dotnet restore
# Build and publish a release
RUN dotnet publish -c Release -o out
# Build runtime image
FROM mcr.microsoft.com/dotnet/aspnet:10.0
WORKDIR /App
COPY --from=build-env /App/out .
ENTRYPOINT ["dotnet", "DotNet.Docker.dll"]
```

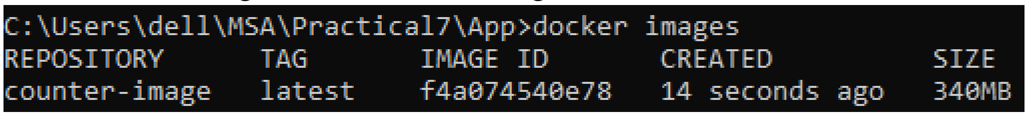
Step 15: Save this file by naming it "Dockerfile". This file should be saved in the same directory where the DotNet.Docker.csproj file is saved i.e., in the App folder.

Step 16: Start Docker Desktop , then go back to command prompt and run the following command in our Application Root Folder to build the 'counter-image' (make sure you add dot at the end of this command): ' docker build -t counter-image -f Dockerfile . '



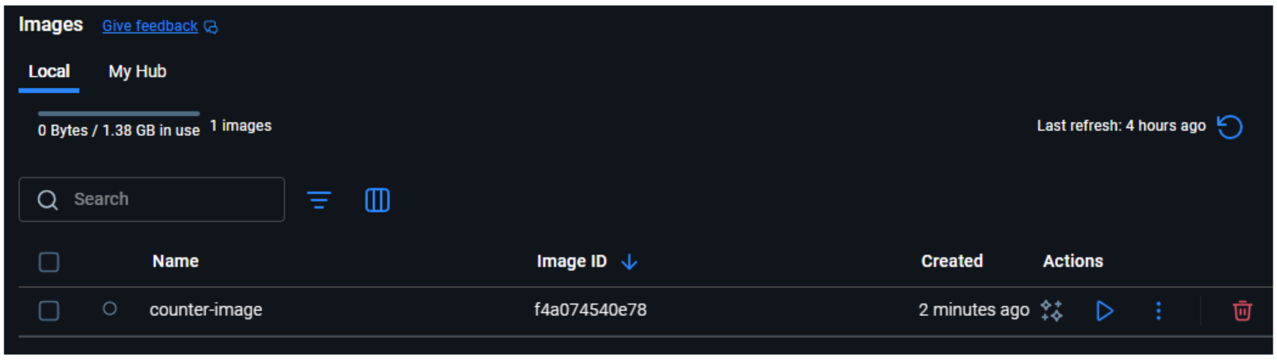
```
C:\Users\dell\MSA\Practical7\App>docker build -t counter-image .
[+] Building 144.3s (14/14) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 450B
 => [internal] load metadata for mcr.microsoft.com/dotnet/sdk:10.0
 => [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:10.0
```

Step 17: Run 'docker images' to see a list of images installed.



```
C:\Users\dell\MSA\Practical7\App>docker images
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
counter-image   latest    f4a074540e78   14 seconds ago   340MB
```

Step 18: Also, if you check Images in Docker Desktop you will see the newly created image.



Name	Image ID	Created	Actions
counter-image	f4a074540e78	2 minutes ago	

Step 19: Create a container named ‘core-counter’ using the following command: docker create --name core-counter counter-image

```
C:\Users\dell\MSA\Practical7\App>docker create --name core-counter counter-image
63dcd453c27417242b5a0b1f0d8c2a3626f5911faa74cc8cc367b0bb8aa935dc
```

Step 20: To see a list of all containers, use the ‘docker ps -a’ command.

```
C:\Users\dell\MSA\Practical7\App>docker ps -a
CONTAINER ID   IMAGE           COMMAND                  CREATED          STATUS    PORTS     NAMES
63dcd453c274   counter-image   "dotnet DotNet.Docke…"   38 seconds ago   Created             core-counter
```

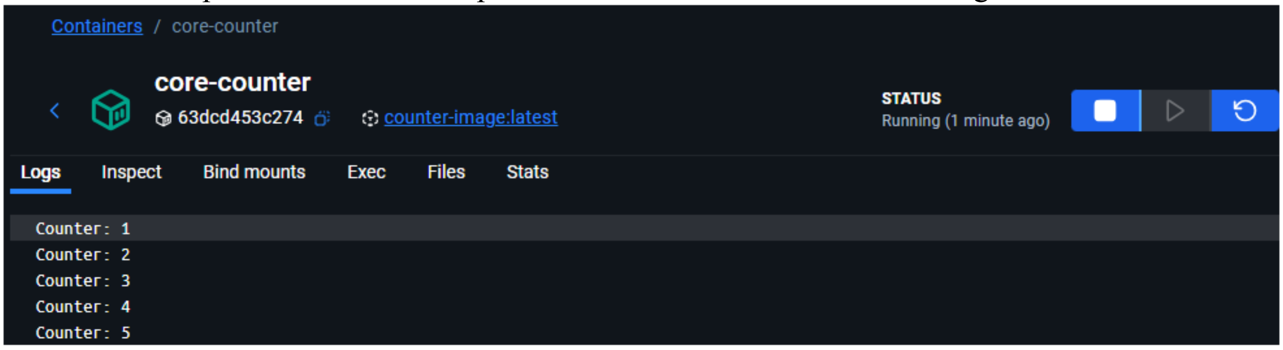
Step 21: Start the ‘core-counter’ container using following command: ‘docker start core-counter’.

```
C:\Users\dell\MSA\Practical7\App>docker start core-counter
core-counter
```

Step 22: You can check using ‘docker ps’ command that the container is running

```
C:\Users\dell\MSA\Practical7\App>docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED         STATUS          PORTS     NAMES
63dcd453c274   counter-image   "dotnet DotNet.Docke…"   2 minutes ago   Up 36 seconds             core-counter
```

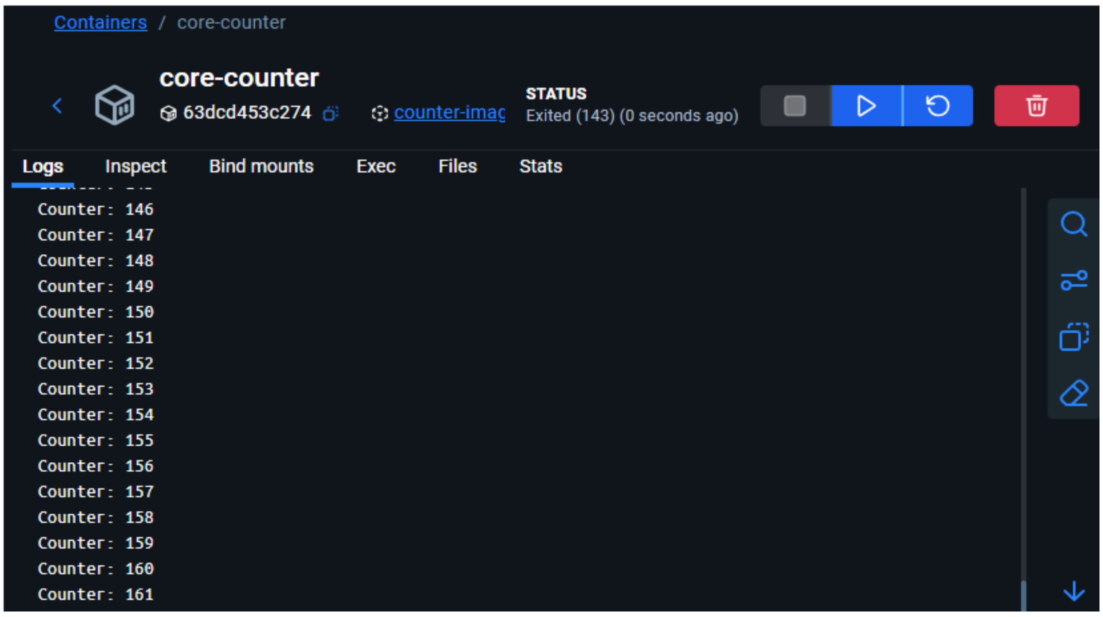
Step 23: Check the output in Docker Desktop → Containers → core-counter → Logs.



```
Counter: 1
Counter: 2
Counter: 3
Counter: 4
Counter: 5
```

Step 24: To stop the code, enter ‘docker stop core-counter’ in the command prompt.

```
C:\Users\dell\MSA\Practical7\App>docker stop core-counter
core-counter
```



```
Counter: 146
Counter: 147
Counter: 148
Counter: 149
Counter: 150
Counter: 151
Counter: 152
Counter: 153
Counter: 154
Counter: 155
Counter: 156
Counter: 157
Counter: 158
Counter: 159
Counter: 160
Counter: 161
```

Practical No : 8

Aim : Demonstrate docker swarm working with asp.net console application

Step 1: Open Command Prompt and navigate to the path where you would like to create your Application

```
C:\Users\dell>cd MSA
```

Step 2: Create a new folder named “Practical8” using the command: mkdir Practical8

```
C:\Users\dell\MSA>mkdir Practical8
```

Step 3: Navigate to this folder

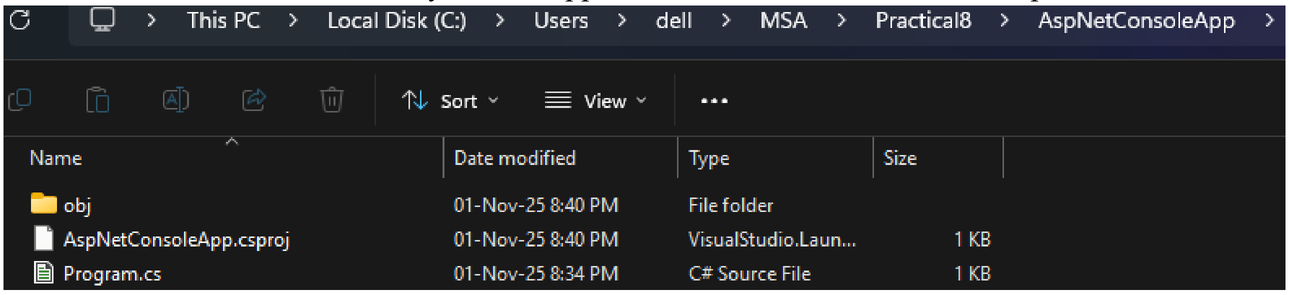
```
C:\Users\dell\MSA>cd Practical8
```

Step 4: Now navigate to the “AspNetConsoleApp” folder where all these files have been created ,remember we will be running all the commands in this path only.

Step 5: In this path Enter the following command to create the dotnet application : dotnet new console -o AspNetConsoleApp

```
C:\Users\dell\MSA\Practical8\AspNetConsoleApp>dotnet add package Microsoft.AspNetCore.App --version 2.2.8
```

Step 6: You can see all the files necessary for the Application have been created in the specified location



This PC > Local Disk (C:) > Users > dell > MSA > Practical8 > AspNetConsoleApp >
Sort
View

```
Name                      Date modified        Type               Size
obj                       01-Nov-25 8:40 PM    File folder
AspNetConsoleApp.csproj   01-Nov-25 8:40 PM    VisualStudio.Laun...   1 KB
Program.cs                01-Nov-25 8:34 PM    C# Source File         1 KB
```

Step 7: Run the following commands to install the necessary ASP.NET Core NuGet packages:

- Install Microsoft.AspNetCore.App (if it’s not already included):
  dotnet add package Microsoft.AspNetCore.App --version 2.2.8

```
C:\Users\dell\MSA\Practical8\AspNetConsoleApp>dotnet add package Microsoft.AspNetCore.App --version 2.2.8
```

- Install Microsoft.AspNetCore.Hosting (if not already included):
  dotnet add package Microsoft.AspNetCore.Hosting

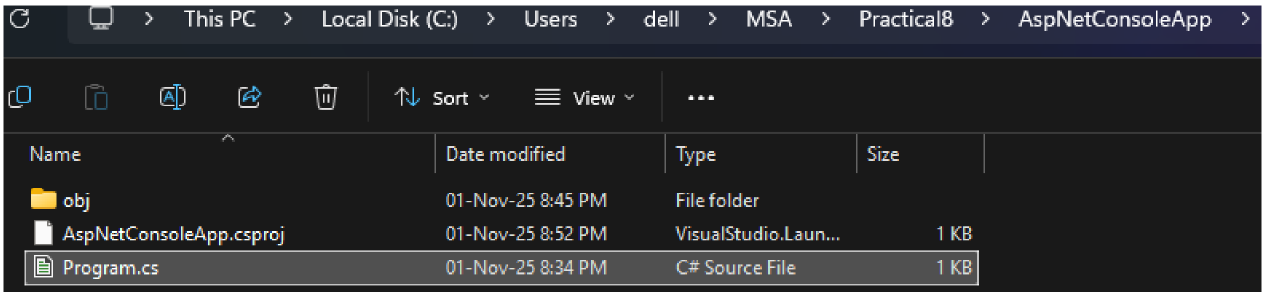
```
C:\Users\dell\MSA\Practical8\AspNetConsoleApp>dotnet add package Microsoft.AspNetCore.Hosting
```

Step 8: Go to the path where our dotnet application was created & get inside the “AspNetConsoleApp” folder and open AspNetConsoleApp.csproj in Notepad

Step 9: Replace any existing code in this file with the following code and save the file:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net9.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.App" Version="2.2.8" />
    <PackageReference Include="Microsoft.AspNetCore.Hosting" Version="2.2.0" />
  </ItemGroup>

</Project>
```

Step 10: Now go to the path where our dotnet application was created & get inside the “AspNetConsoleApp” folder and open Program.cs file in Notepad


This PC > Local Disk (C:) > Users > dell > MSA > Practical8 > AspNetConsoleApp >
Sort
View

Name	Date modified	Type	Size
obj	01-Nov-25 8:45 PM	File folder	
AspNetConsoleApp.csproj	01-Nov-25 8:52 PM	VisualStudio.Laun...	1 KB
Program.cs	01-Nov-25 8:34 PM	C# Source File	1 KB

Step 11: Replace any existing code in this file with the following code and save the file . This code starts a minimal web server that responds with a simple message.:

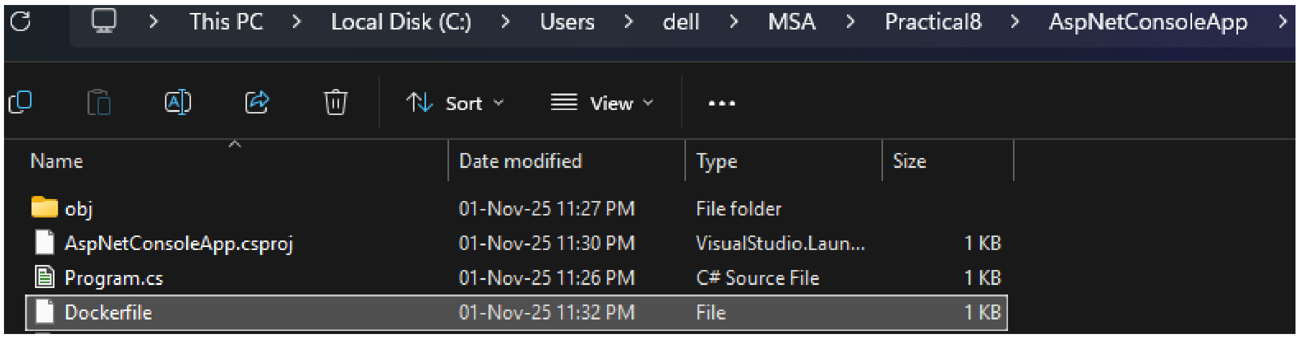
```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

// Route: responds with a simple message
app.MapGet("/", () => "Hello from Docker Swarm with ASP.NET Core!");

// Listen on all network interfaces and port 5000
app.Run("http://0.0.0.0:5000");
```

Step 12: Open Notepad → Create a New file and write the following code in it :

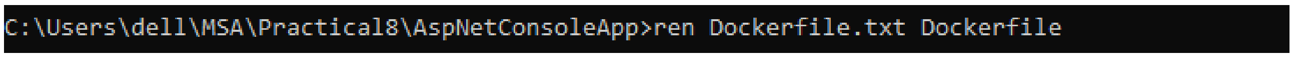
```
FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build
WORKDIR /app
COPY . ./
RUN dotnet publish -c Release -o out

# Use the runtime image for the final stage
FROM mcr.microsoft.com/dotnet/aspnet:9.0 AS runtime
WORKDIR /app
COPY --from=build /app/out ./
EXPOSE 5000
ENTRYPOINT ["dotnet", "AspNetConsoleApp.dll"]
```

Step 13: Save this file by naming it “Dockerfile”. This file should be saved in the AspNetConsoleApp folder.

Note: Dockerfile is a text file that doesn't have any extension. While saving the file, use double inverted quotes - “Dockerfile” - and it will be saved as a file as shown below. If not saved in file format refer below screenshot.

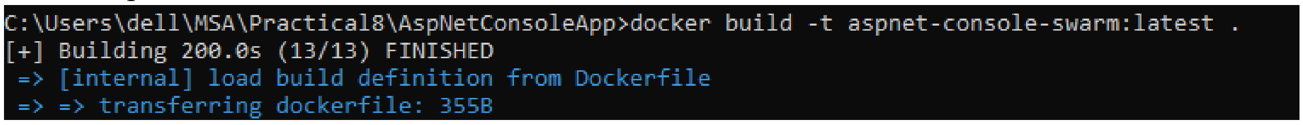

This PC > Local Disk (C:) > Users > dell > MSA > Practical8 > AspNetConsoleApp >
Sort
View

Name	Date modified	Type	Size
obj	01-Nov-25 11:27 PM	File folder	
AspNetConsoleApp.csproj	01-Nov-25 11:30 PM	VisualStudio.Laun...	1 KB
Program.cs	01-Nov-25 11:26 PM	C# Source File	1 KB
Dockerfile	01-Nov-25 11:32 PM	File	1 KB



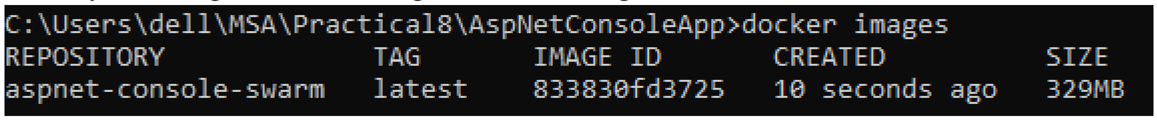
```
C:\Users\dell\MSA\Practical8\AspNetConsoleApp>ren Dockerfile.txt Dockerfile
```

Step 14: Start Docker Desktop , then go back to command prompt and run the following command in our Application Root Folder to build the Docker image (make sure you add dot at the end of this command) : docker build -t aspnet-console-swarm:latest .



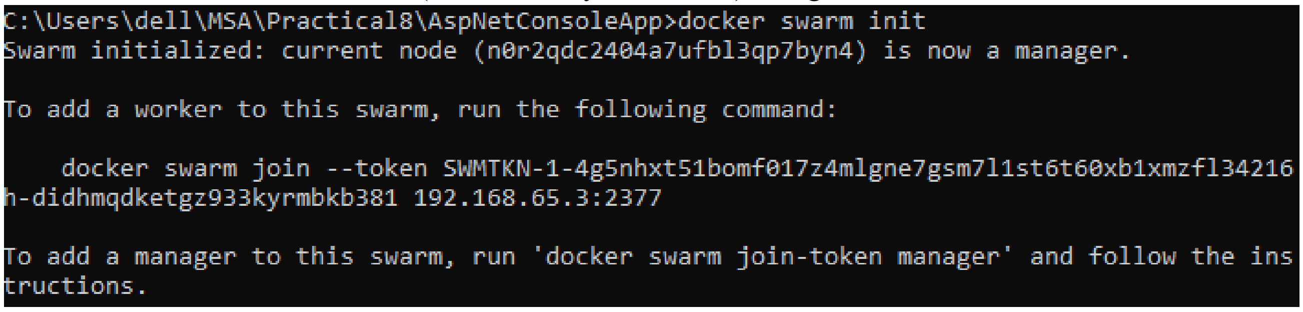
```
C:\Users\dell\MSA\Practical8\AspNetConsoleApp>docker build -t aspnet-console-swarm:latest .
[+] Building 200.0s (13/13) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 355B
```

Step 15: Verify the image creation using : " docker images "



```
C:\Users\dell\MSA\Practical8\AspNetConsoleApp>docker images
REPOSITORY             TAG       IMAGE ID       CREATED          SIZE
aspnet-console-swarm   latest    833830fd3725   10 seconds ago   329MB
```

Step 16: Now Initialize Docker Swarm (if not already initialized) using command : " docker swarm init "

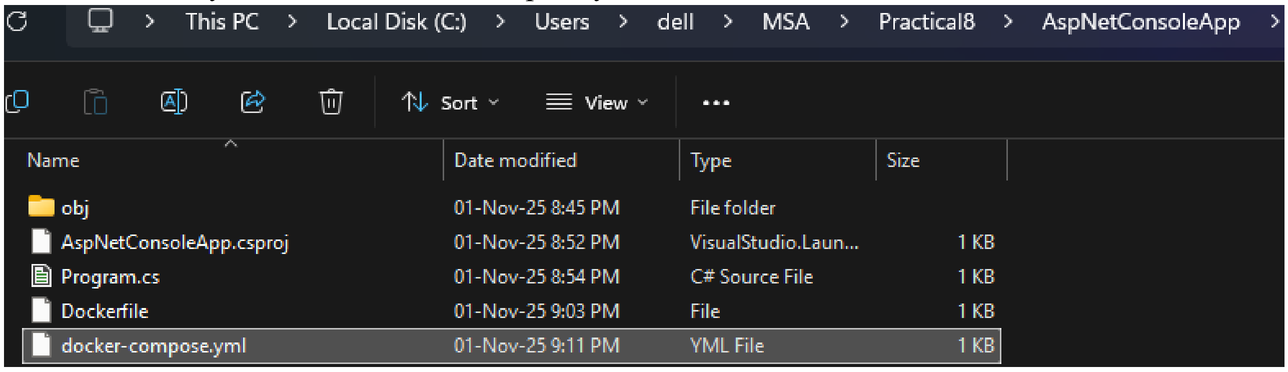


```
C:\Users\dell\MSA\Practical8\AspNetConsoleApp>docker swarm init
Swarm initialized: current node (n0r2qdc2404a7ufbl3qp7byn4) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-4g5nhxt51bomf017z4mlgne7gsm7l1st6t60xb1xmzfl34216
h-didhmqdketgz933kyrmbkb381 192.168.65.3:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the ins
tructions.
```

Step 17: Create a Docker Compose File (Optional but Recommended) :
In the root directory open notepad and write the following code :

```
version: "3.8"

services:
  aspnetapp:
    image: aspnet-console-swarm:latest
    ports:
      - "5000:5000"
    deploy:
      replicas: 3
      resources:
        limits:
          cpus: "0.50"
          memory: "256M"
      restart_policy:
        condition: on-failure
```

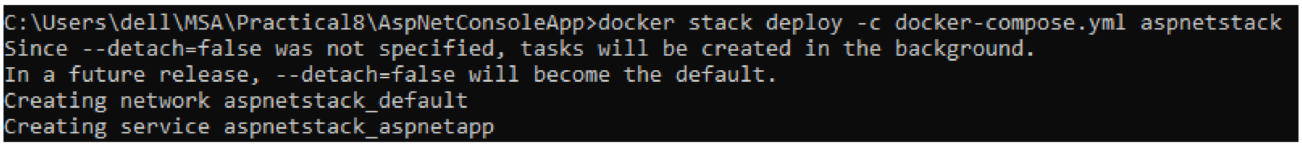
Step 18: Save the file by the name : docker-compose.yml.



This PC > Local Disk (C:) > Users > dell > MSA > Practical8 > AspNetConsoleApp >
Sort
View

Name	Date modified	Type	Size
obj	01-Nov-25 8:45 PM	File folder	
AspNetConsoleApp.csproj	01-Nov-25 8:52 PM	VisualStudio.Laun...	1 KB
Program.cs	01-Nov-25 8:54 PM	C# Source File	1 KB
Dockerfile	01-Nov-25 9:03 PM	File	1 KB
docker-compose.yml	01-Nov-25 9:11 PM	YML File	1 KB

Step 19: Deploy the application to swarm using command : docker stack deploy -c docker-compose.yml aspnetstack



```
C:\Users\dell\MSA\Practical8\AspNetConsoleApp>docker stack deploy -c docker-compose.yml aspnetstack
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating network aspnetstack_default
Creating service aspnetstack_aspnetapp
```

Step 20: Verify that the services are running using: docker service ls

```
C:\Users\dell\MSA\Practical8\AspNetConsoleApp>docker service ls
ID             NAME                  MODE         REPLICAS   IMAGE                         PORTS
cfpeq6tus0pn   aspnetstack_aspnetapp   replicated   3/3        aspnet-console-swarm:latest   *:5000->5000/tcp
```

And If you are getting error or can't see all 3 replicas try the following command : docker stack deploy -c docker-compose.yml swarmstack

Step 21: Check the running containers using : docker ps

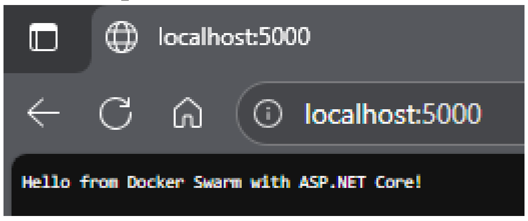
```
C:\Users\dell\MSA\Practical8\AspNetConsoleApp>docker ps
CONTAINER ID   IMAGE                         COMMAND                  CREATED          STATUS          PORTS      NAMES
628a2385d15d   aspnet-console-swarm:latest   "dotnet AspNetConsol…"   17 seconds ago   Up 17 seconds   5000/tcp   aspnetstack_aspnetapp.3.ltenkvzdromckmb1kjxueakcy
c6b68ec12d96   aspnet-console-swarm:latest   "dotnet AspNetConsol…"   17 seconds ago   Up 17 seconds   5000/tcp   aspnetstack_aspnetapp.2.xexnylw4fxm115r0m22hzn7xq
09362dcac57c   aspnet-console-swarm:latest   "dotnet AspNetConsol…"   18 seconds ago   Up 17 seconds   5000/tcp   aspnetstack_aspnetapp.1.jupdqmatbxfozg9m692eddk4y
```

Step 22: Now, test the app using : curl http://localhost:5000

```
C:\Users\dell\MSA\Practical8\AspNetConsoleApp>  curl http://localhost:5000
Hello from Docker Swarm with ASP.NET Core!
```

Step 23: Open in Browser Go to : http://localhost:5000

Final Output:



```
Hello from Docker Swarm with ASP.NET Core!
```

Step 24: Clean Up : Remove the deployed stack using: docker stack rm aspnetstack

```
C:\Users\dell\MSA\Practical8\AspNetConsoleApp>docker stack rm aspnetstack
Removing service aspnetstack_aspnetapp
Removing network aspnetstack_default
```

Step 25: Leave the Swarm (if no longer needed) using : docker swarm leave –f

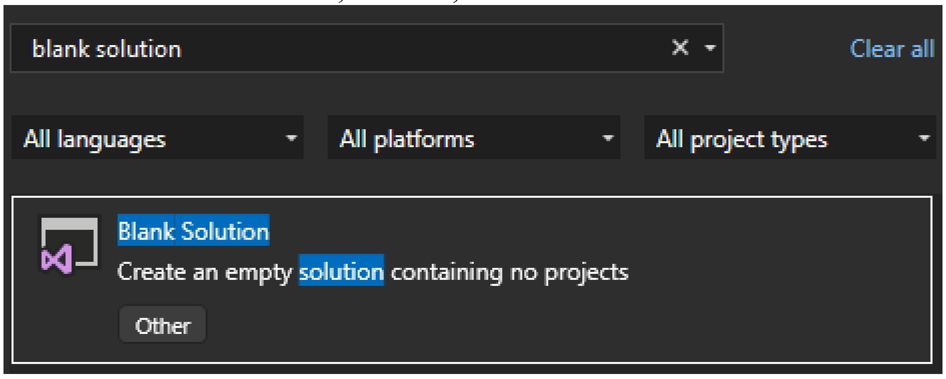
```
C:\Users\dell\MSA\Practical8\AspNetConsoleApp>docker swarm leave -f
Node left the swarm.
```

Practical No : 9

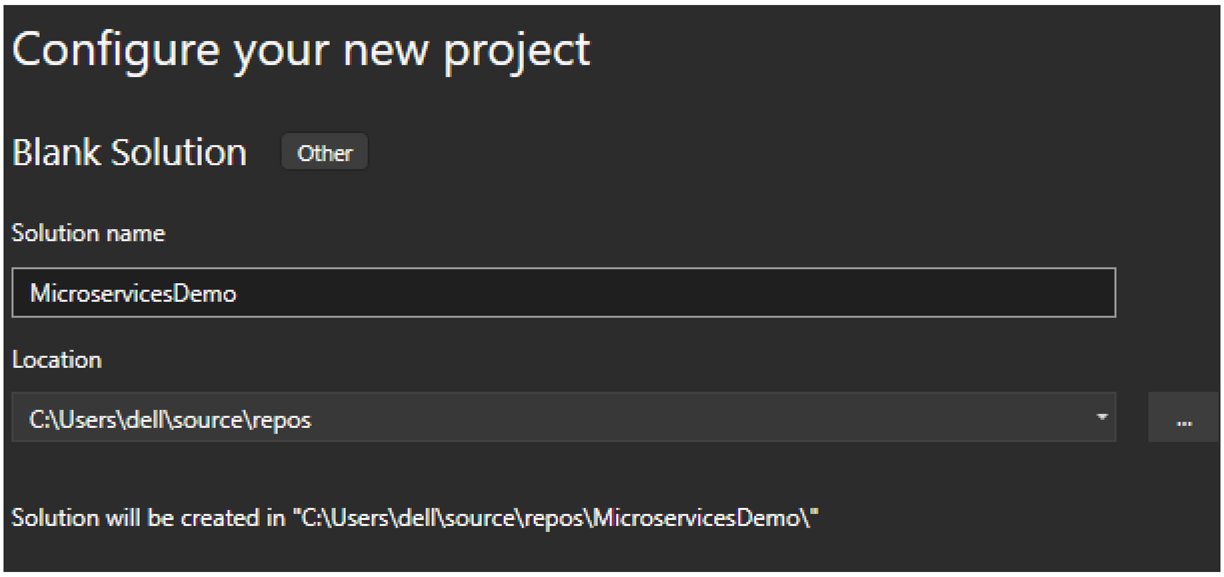
Aim: Creating microservices in ASP.net core

Steps:

- Open Visual Studio Community 2026
- Click on Create a new Project
- Search for Blank Solution, select it, and click Next



- Set the name as MicroservicesDemo and click Create


Configure your new project

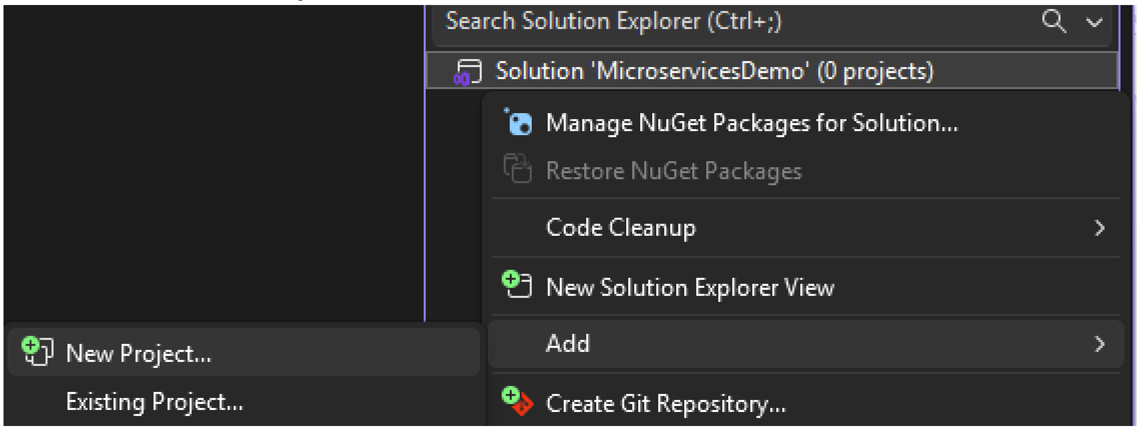
Blank Solution Other

Solution name: MicroservicesDemo

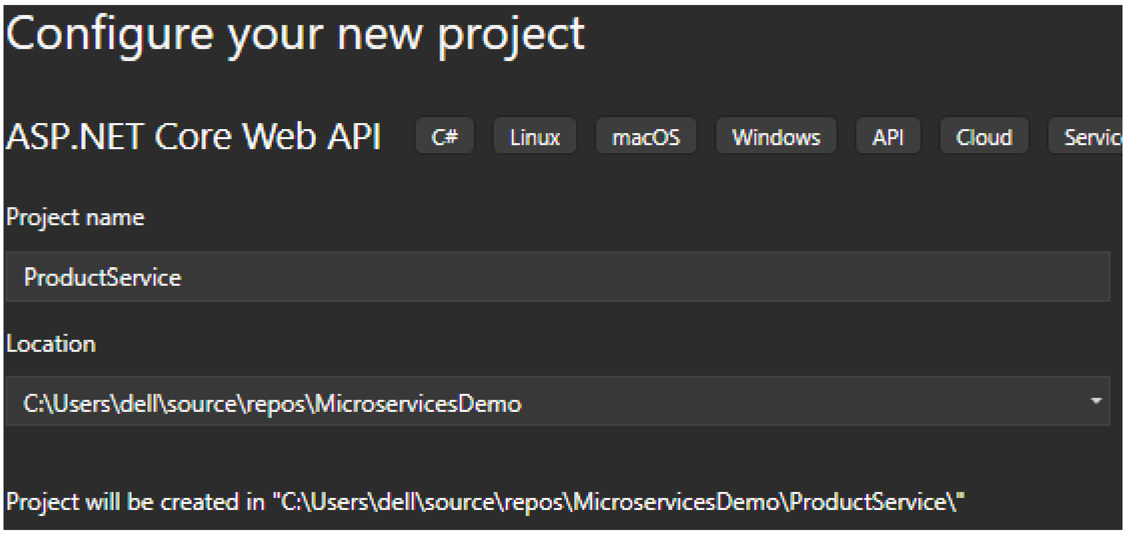
Location: C:\Users\dell\source\repos

Solution will be created in "C:\Users\dell\source\repos\MicroservicesDemo\"

- Right-click the **MicroservicesDemo** solution in Solution Explorer
- Select Add → New Project.


Search Solution Explorer (Ctrl+;)
Solution 'MicroservicesDemo' (0 projects)
Manage NuGet Packages for Solution...
Restore NuGet Packages
Code Cleanup
New Solution Explorer View
Add
Create Git Repository...
New Project...
Existing Project...

- Search for ASP.NET Core Web API, select it, and click Next.
- Set the name as **ProductService**, then click Next.
- Choose .NET 8.0 (Long Term Support), and click Create.


Configure your new project

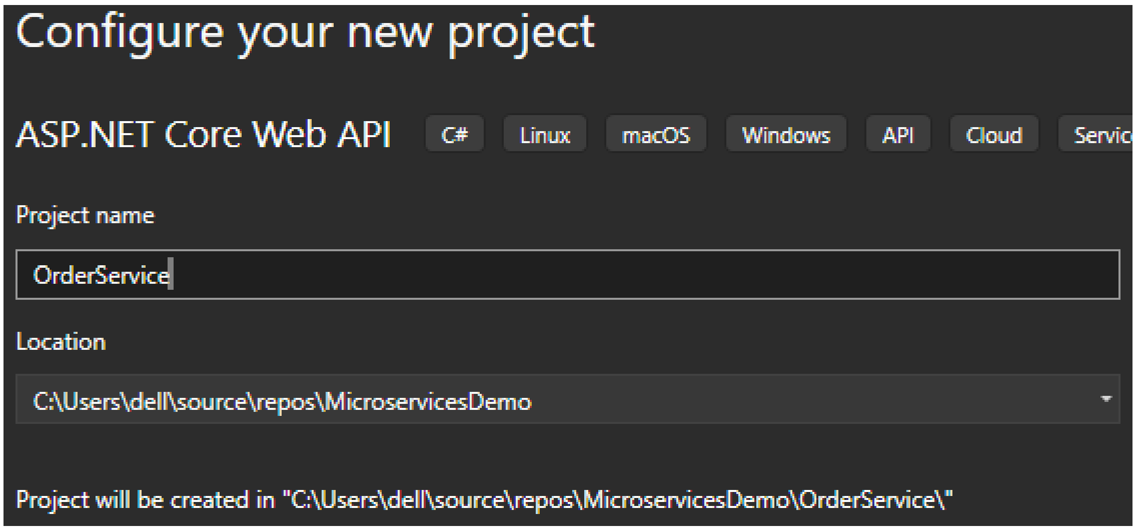
ASP.NET Core Web API | C# | Linux | macOS | Windows | API | Cloud | Servic

Project name: ProductService

Location: C:\Users\dell\source\repos\MicroservicesDemo

Project will be created in "C:\Users\dell\source\repos\MicroservicesDemo\ProductService\"

- Right-click the MicroservicesDemo solution again.
- Select Add → New Project.
- Search for ASP.NET Core Web API, select it, and click Next.
- Name it **OrderService**, then click Next.


Configure your new project

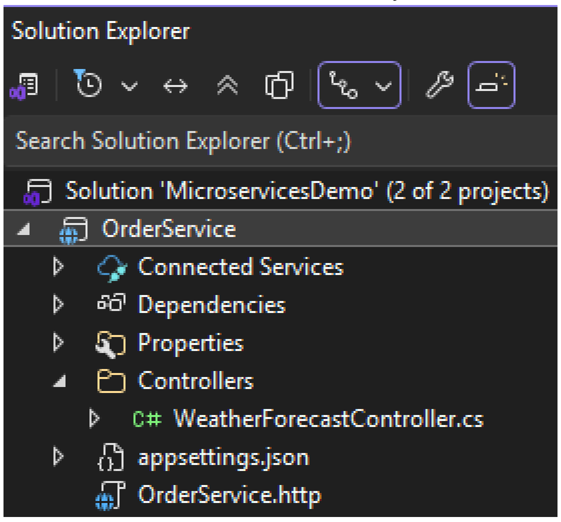
ASP.NET Core Web API | C# | Linux | macOS | Windows | API | Cloud | Servic

Project name: OrderService

Location: C:\Users\dell\source\repos\MicroservicesDemo

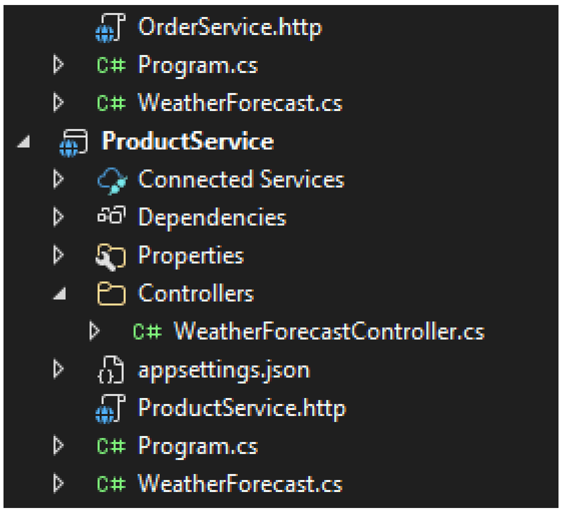
Project will be created in "C:\Users\dell\source\repos\MicroservicesDemo\OrderService\"

- Choose .NET 8.0 (Long Term Support), and click Create.
- Your Solution Explorer should now have:
  ➢ MicroservicesDemo (Solution)
  1) ProductService (ASP.NET Core Web API)
  2) OrderService (ASP.NET Core Web API)
- You can delete the WeatherForecast.cs file from the OrderService and ProductService project folders, as well as the WeatherForecastController.cs file from the Controllers folder of both projects, since these files are included by default.

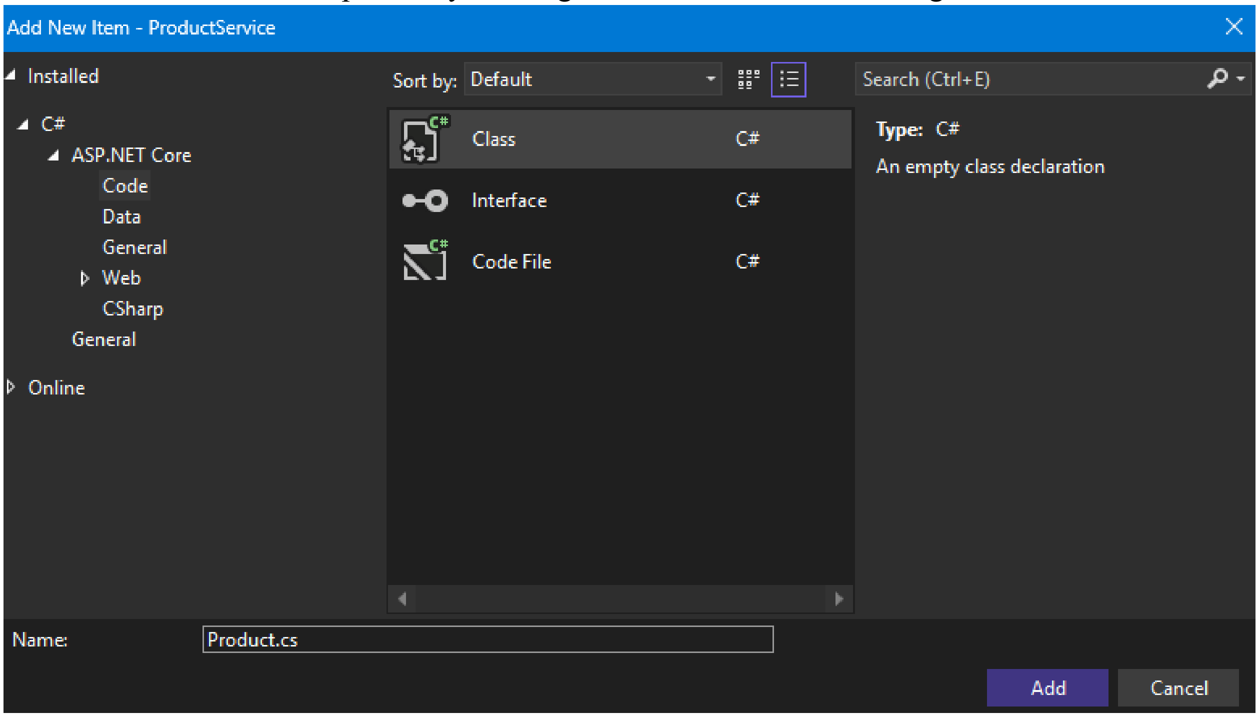

Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'MicroservicesDemo' (2 of 2 projects)
◢ OrderService
    ▷ Connected Services
    ▷ Dependencies
    ▷ Properties
    ◢ Controllers
        ▷ C# WeatherForecastController.cs
    ▷ appsettings.json
      OrderService.http
```



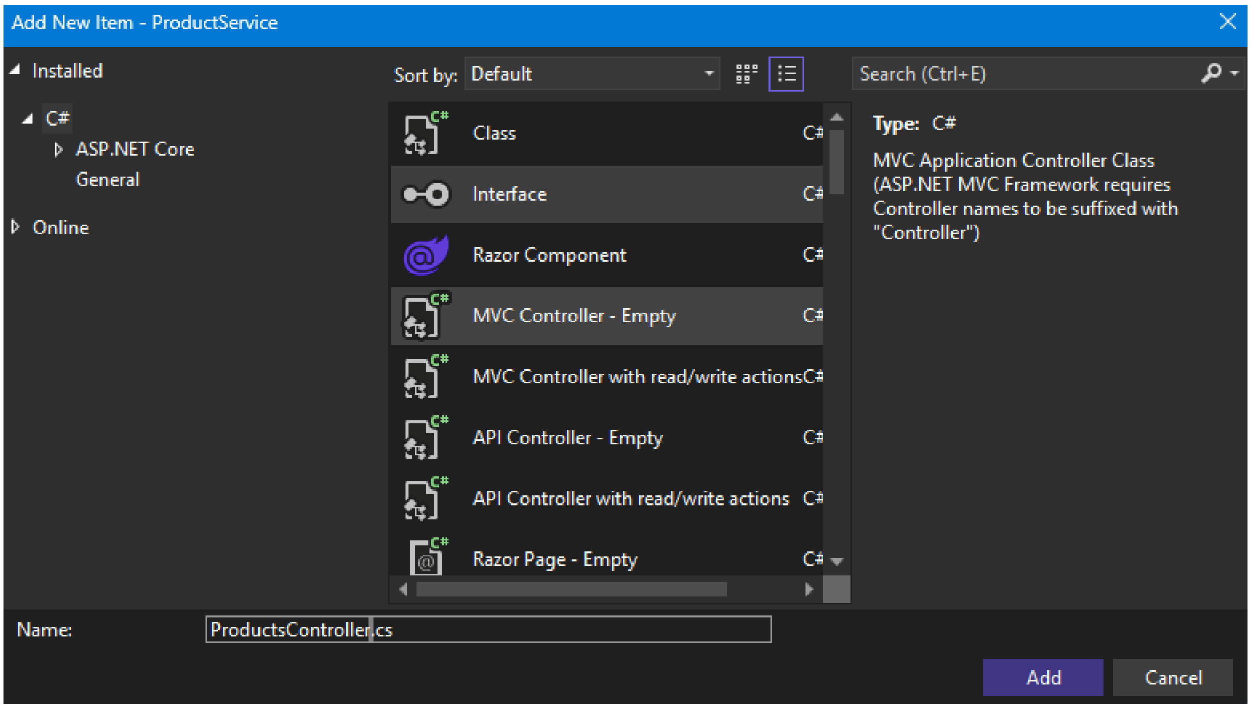
- In ProductService, right-click the Models folder (create one if needed).
- Click Add → Class.
- Name it **Product.cs** and replace any existing code in it with the following code & save it:



Name: Product.cs

```
namespace ProductService.Models
{
   public class Product
   {
      public int Id { get; set; }
      public string Name { get; set; } = string.Empty;
      public decimal Price { get; set; }
   }
}
```

- In ProductService, go to the Controllers folder.
- Right-click and select Add → Controller.
- Select API Controller - Empty, name it **ProductsController**.
- Replace the existing code in it with the below code and save it :


Add New Item - ProductService
Installed
C#
ASP.NET Core
General
Online
Sort by: Default
Search (Ctrl+E)
Class C#
Interface C#
Razor Component C#
MVC Controller - Empty C#
MVC Controller with read/write actions C#
API Controller - Empty C#
API Controller with read/write actions C#
Razor Page - Empty C#
Add
Cancel

Type: C#

MVC Application Controller Class (ASP.NET MVC Framework requires Controller names to be suffixed with "Controller")

Name: ProductsController.cs

```
using Microsoft.AspNetCore.Mvc;
using ProductService.Models;

namespace ProductService.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ProductsController : ControllerBase
    {
        private static readonly List<Product> Products = new()
        {
            new() { Id = 1, Name = "Laptop", Price = 50000 },
            new() { Id = 2, Name = "Phone", Price = 20000 },
            new() { Id = 3, Name = "Tablet", Price = 15000 }
        };

        [HttpGet]
        public IActionResult GetAllProducts() => Ok(Products);

        [HttpGet("{id}")]
        public IActionResult GetProductById(int id)
        {
            var product = Products.FirstOrDefault(p => p.Id == id);
            return product == null ? NotFound() : Ok(product);
        }
    }
}
```

- Open Program.cs in ProductService.
- Replace the code in it with the following code and save it:

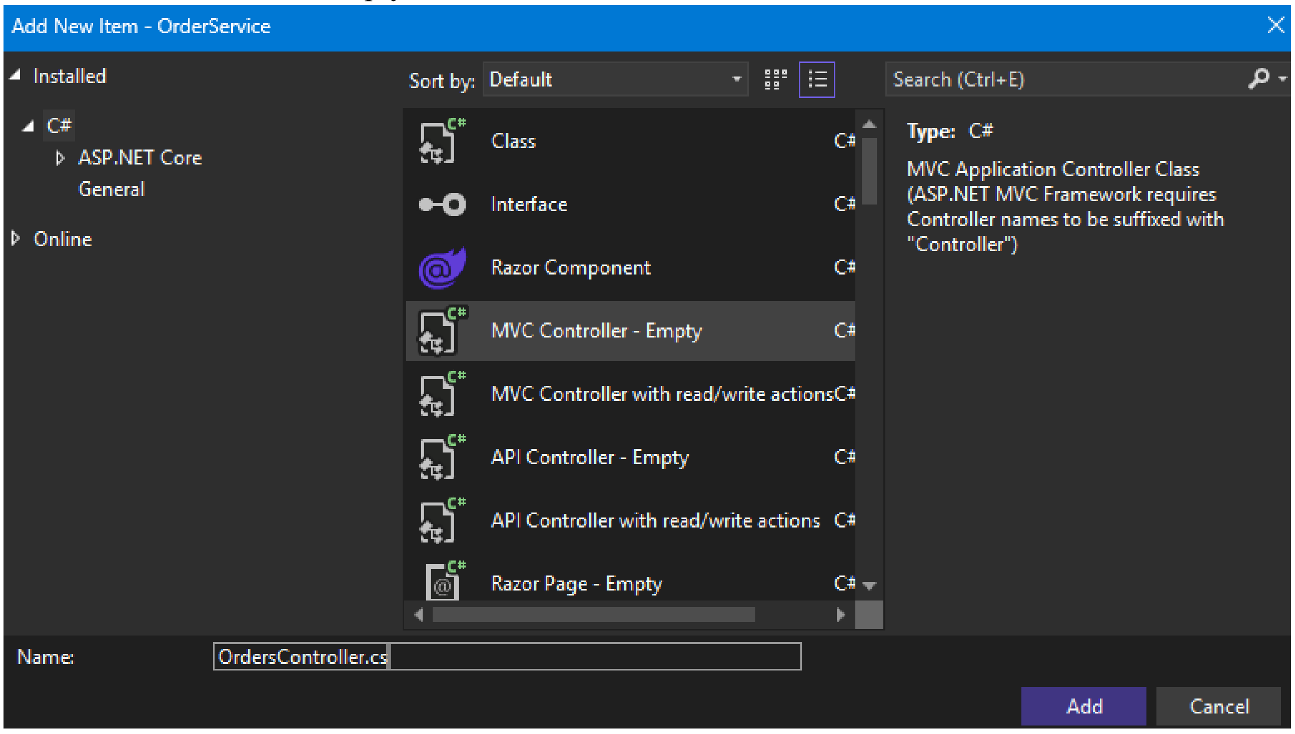
```
var builder = WebApplication.CreateBuilder(args);

// Add controller and Swagger support
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
```

```
var app = builder.Build();

// Show Swagger only in development mode
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

// Map controller routes and run the app
app.MapControllers();
app.Run();
```

- In OrderService, right-click the Models folder (create one if needed).
- Click Add → Class.
- Name it **Order.cs** and replace any existing code in it with the following code & save it:

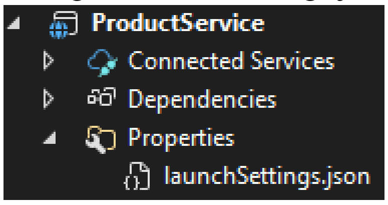
```
namespace OrderService.Models
{
    public class Order
    {
        public int OrderId { get; set; }          // Unique ID for the order
        public int ProductId { get; set; }        // ID of the product being ordered
        public string ProductName { get; set; } = string.Empty;  // Name of the product
        public decimal Price { get; set; }        // Price of the product
    }
}
```

- In OrderService, go to the Controllers folder.
- Right-click and select Add → Controller.
- Choose API Controller - Empty, name it **OrdersController**.

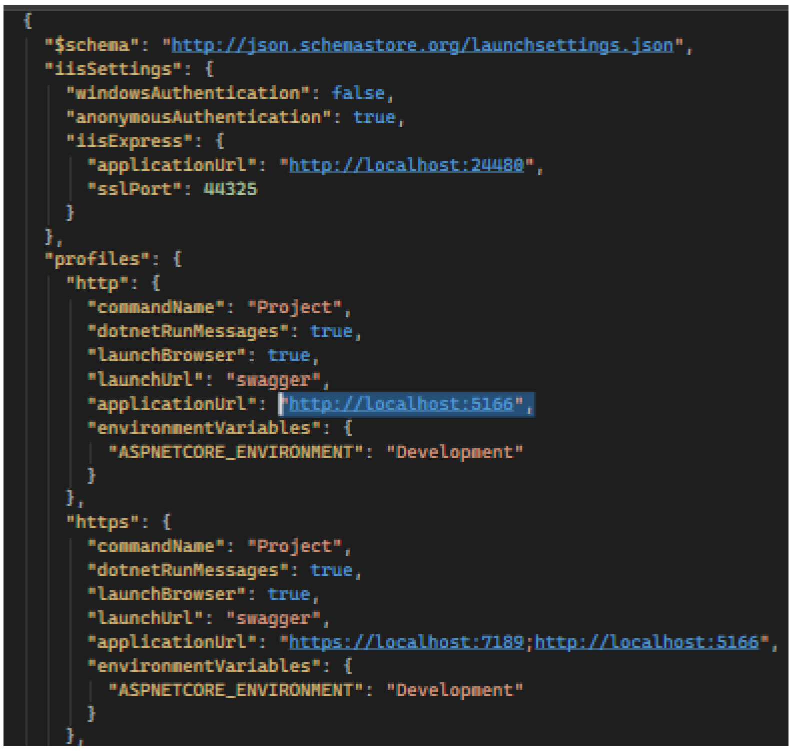

Add New Item - OrderService
Installed
C#
ASP.NET Core
General
Online
Sort by: Default
Search (Ctrl+E)
Class C#
Interface C#
Razor Component C#
MVC Controller - Empty C#
MVC Controller with read/write actions C#
API Controller - Empty C#
API Controller with read/write actions C#
Razor Page - Empty C#
Type: C#
MVC Application Controller Class (ASP.NET MVC Framework requires Controller names to be suffixed with "Controller")
Name: OrdersController.cs
Add
Cancel

```
using Microsoft.AspNetCore.Mvc;
using OrderService.Models;
using System.Net.Http.Json;

namespace OrderService.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class OrdersController : ControllerBase
    {
        private readonly HttpClient _httpClient;

        // Constructor: Gets an HttpClient from the factory
        public OrdersController(IHttpClientFactory httpClientFactory)
        {
            _httpClient = httpClientFactory.CreateClient("ProductService");
        }

        // POST: api/orders/{productId}
        [HttpPost("{productId}")]
        public async Task<IActionResult> CreateOrder(int productId)
        {
            // Get product details from another microservice (ProductService)
            var product = await
_httpClient.GetFromJsonAsync<ProductDto>($"/api/products/{productId}");
            if (product == null)
                return NotFound("Product not found");

            // Create a new order using the product info
            var order = new Order
            {
                OrderId = new Random().Next(1000, 9999),
                ProductId = product.Id,
                ProductName = product.Name,
                Price = product.Price
            };

            return Ok(order); // Return the created order
        }
    }

    // Product model used to receive data from ProductService
    public class ProductDto
    {
        public int Id { get; set; }
        public string Name { get; set; } = string.Empty;
        public decimal Price { get; set; }
    }
}
```

- Now go to launchSettings.json file in project folder ProductService



- Note down the URL and Port under the object http of profiles list

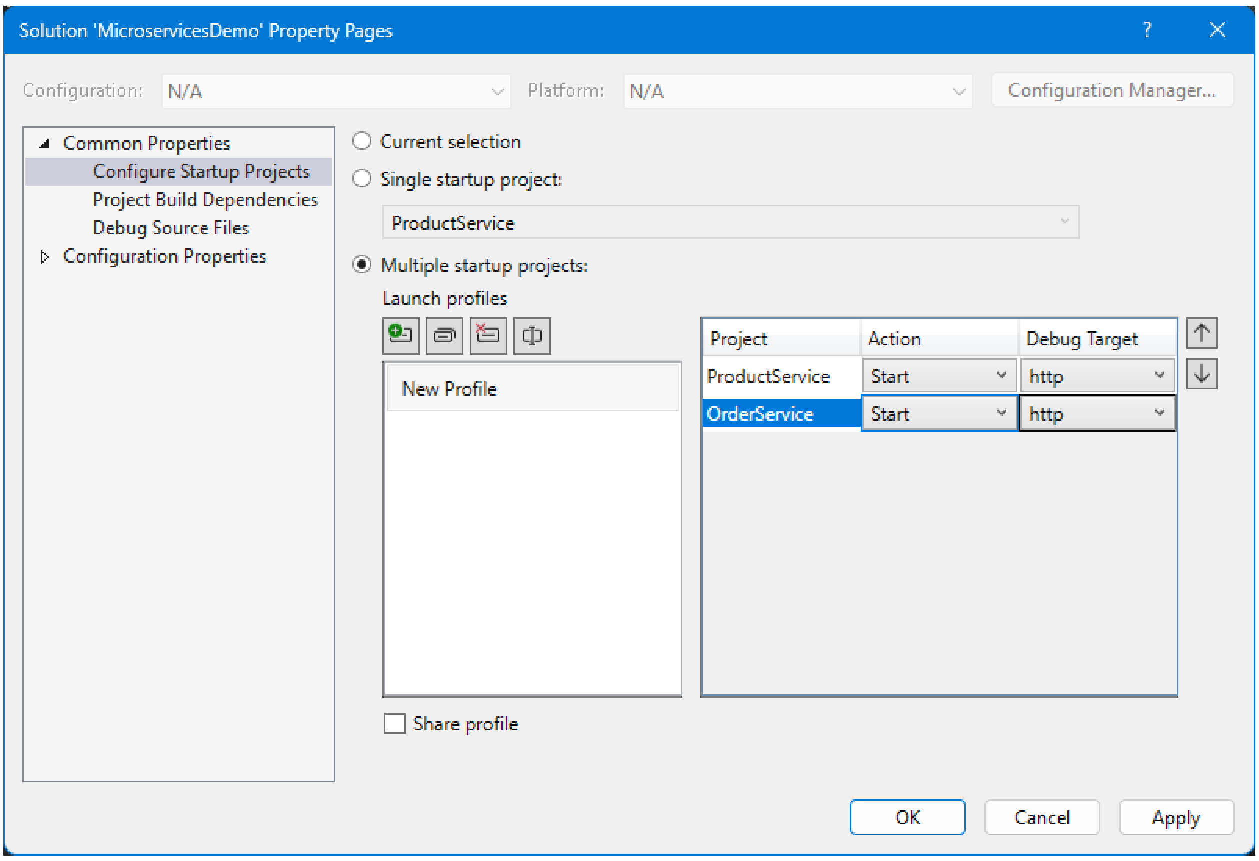


```
{
  "$schema": "http://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:24480",
      "sslPort": 44325
    }
  },
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "http://localhost:5166",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "https": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "https://localhost:7189;http://localhost:5166",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
```

- Open Program.cs in OrderService.
- Replace the code in it with the following code , and most importantly put the above noted URL in client.BaseAddress = new Uri(“ ”); part of the code and save it:

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

// Register an HttpClient named "ProductService"
builder.Services.AddHttpClient("ProductService", client =>
{
    client.BaseAddress = new Uri("http://localhost:5166"); // ProductService API base URL
});

// Add controller and Swagger support
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Enable Swagger only in Development
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

// Map controllers and start the app
app.MapControllers();
app.Run();
```

- Now right click on your Solution MicroservicesDemounder Solution Explorer and go to “Configure Startup Projects”
- Select Multiple Startup Projects and select the options as below, apply the settings and click OK :


Solution 'MicroservicesDemo' Property Pages

Configuration: N/A Platform: N/A Configuration Manager...

- Common Properties
  - Configure Startup Projects
  - Project Build Dependencies
  - Debug Source Files
- Configuration Properties

( ) Current selection
( ) Single startup project: ProductService
(●) Multiple startup projects:

Launch profiles

New Profile

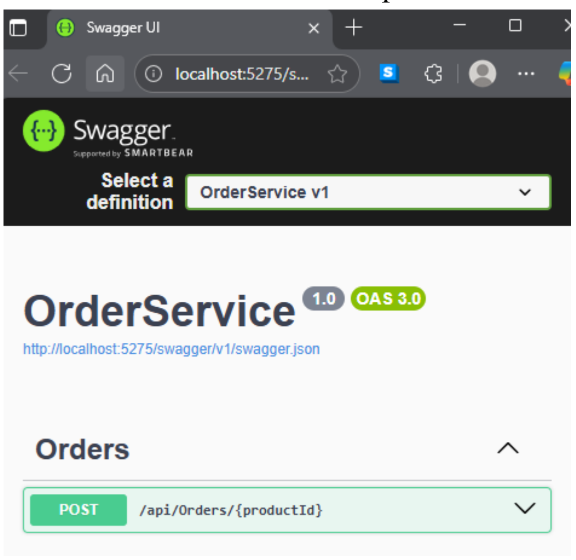
Project	Action	Debug Target
ProductService	Start	http
OrderService	Start	http

☐ Share profile

OK Cancel Apply

- Now start running the application

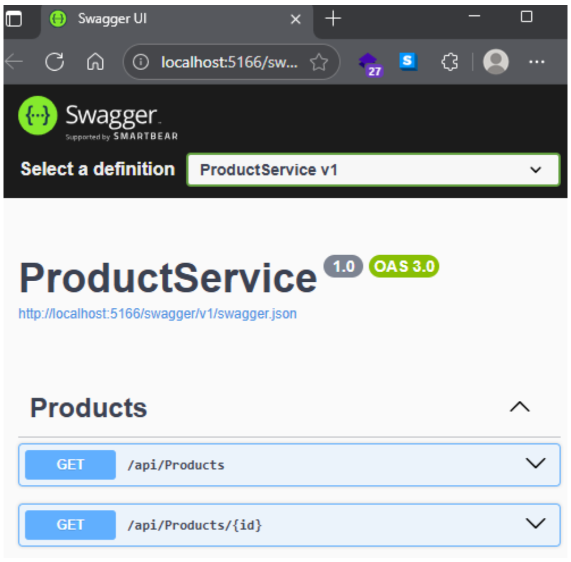
- Two Browser windows will open one for the OrderService and other for ProductService


Swagger UI — localhost:5275/s...

Select a definition: OrderService v1

OrderService 1.0 OAS 3.0
http://localhost:5275/swagger/v1/swagger.json

Orders
POST /api/Orders/{productId}

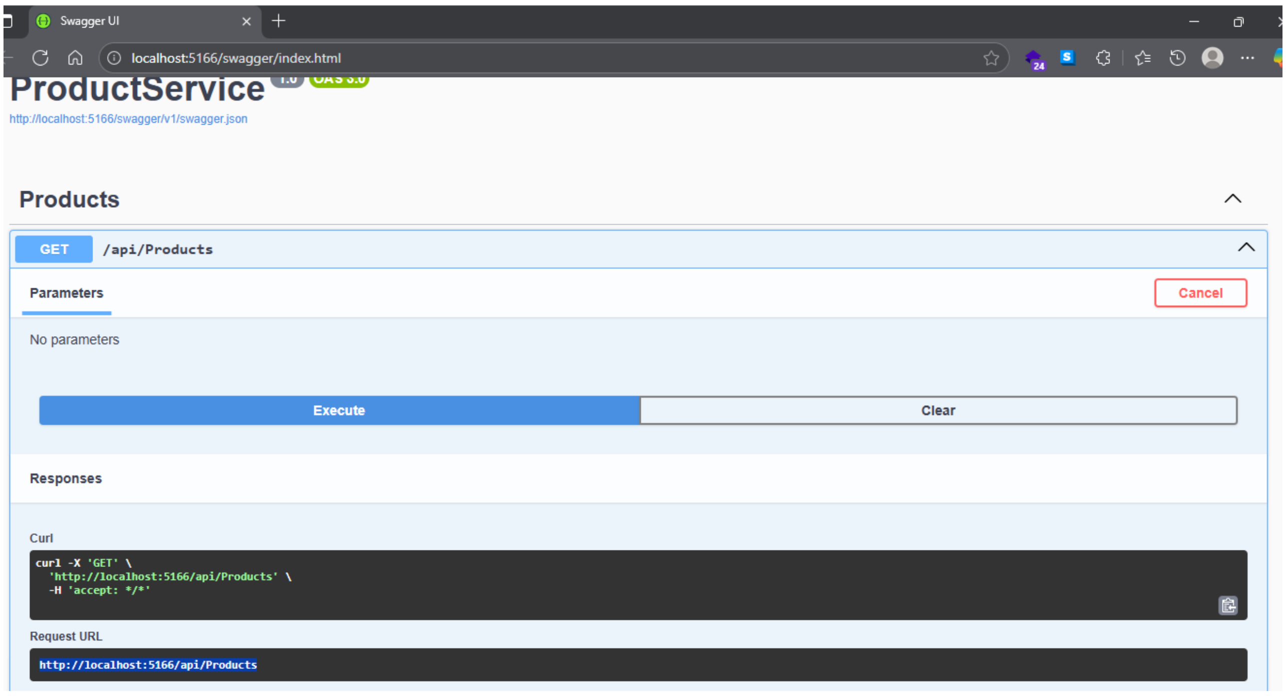

Swagger UI — localhost:5166/sw...

Select a definition: ProductService v1

ProductService 1.0 OAS 3.0
http://localhost:5166/swagger/v1/swagger.json

Products
GET /api/Products
GET /api/Products/{id}

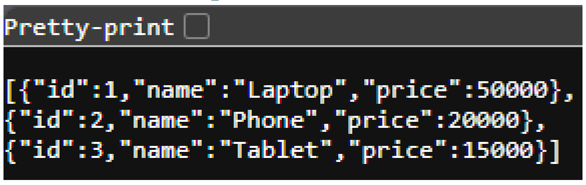
- For ProductService :
  The API should be accessible at:
  ➢ GET /api/Products → Retrieve all Products.
  ➢ GET /api/Products/{id} → Retrieve a Product by ID.
- For example click on GET /api/Products dropdown ,click on try it out and then execute , a request URL will be displayed.


Swagger UI
localhost:5166/swagger/index.html
ProductService 1.0 OAS 3.0
http://localhost:5166/swagger/v1/swagger.json
Products
GET /api/Products
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
Request URL

```
curl -X 'GET' \
  'http://localhost:5166/api/Products' \
  -H 'accept: */*'
```

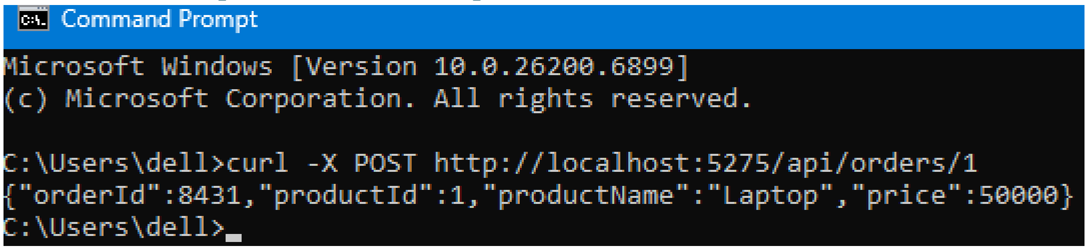
```
http://localhost:5166/api/Products
```

- Search this URL in browser you can see the output ,similarly you can check for GET /api/Products/{id} also.
  localhost:5166/api/Products/



```
[{"id":1,"name":"Laptop","price":50000},
{"id":2,"name":"Phone","price":20000},
{"id":3,"name":"Tablet","price":15000}]
```

- For OrderSevice : The API should be accessible at:
  ➢ GET /api/Orders/{productId} → Retrieve an order by Product ID.
- For this you can check the output in terminal using the command (for Product ID -1):
  curl -X POST http://localhost:5275/api/orders/1



```
Microsoft Windows [Version 10.0.26200.6899]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dell>curl -X POST http://localhost:5275/api/orders/1
{"orderId":8431,"productId":1,"productName":"Laptop","price":50000}
C:\Users\dell>
```