Index

|  |  |  |
| --- | --- | --- |
| **SR.**  **NO.** | **TITLE** | **PAGE NO** |
| 1 | Building .Net Core MVC (BookStore) | 1 |
| 2(A) | Building ASP.Net Core REST API (using command prompt) | 7 |
| 2(B) | Building ASP.Net Core REST API (using Visual Studio) | 17 |
| 3 | Building .Net Core MVC (Hello World & StockQuote) | 25 |
| 4 | Working with Docker Commands, Images and Container | 30 |
| 5 | Working with Docker Volumes and Networks (Play With Docker) | 33 |
| 6 | Working with Docker Swarm | 42 |
| 7 | Working with CircleCI | 48 |
| 8 | Working with Kubernets | 55 |

## PRACTICAL NO 1 – BUILDING ASP .NET CORE MVC (BOOKSTORE)

Steps:

1. Install .Net Core Sdk

(Link: <https://dotnet.microsoft.com/en-us/download> )

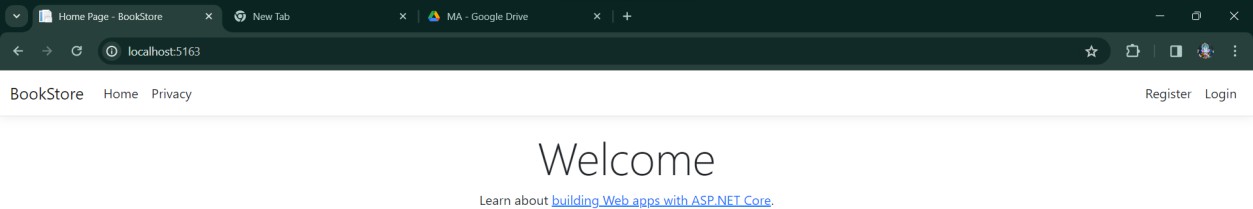
1. Create folder BookStore folder in D: drive or any other drive.
2. Open Command prompt and perform following operations

## Command:

dotnet new mvc --auth Individual -o BookStore cd BookStore

dotnet run

# Output:



# 

1. Add the following **Book.cs** file in the Models Folder.

## Code:

using System;

using System.Collections.Generic; namespace BookStore.Models

{

public class Book

{

public int Id {get;set;} public string Title {get;set;}

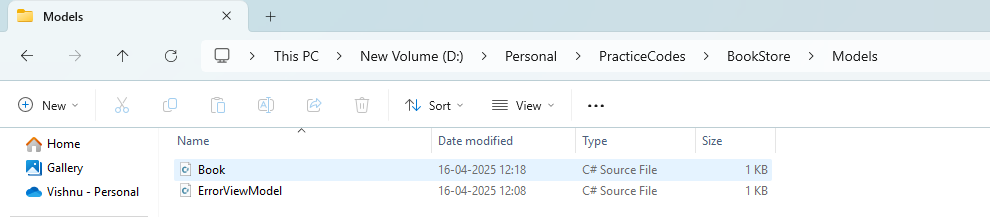
public string Genre {get;set;}

public List<string>Authors{get;set;} public decimal Price {get;set;}

public DateTime PublishDate{get;set;}

}

}



1. Go to the Controllers Folder and add **BooksController.cs** file in it.

## Code:

using System.Diagnostics;

using System.Linq;

using System.Threading.Tasks; using Microsoft.AspNetCore.Mvc; using Microsoft.Extensions.Logging; using BookStore.Models;

using System.Collections.Generic; namespace BookStore.controllers

{

public class BooksController:Controller

{

public IActionResult Index()

{

return View();

}

public IActionResult Details()

{

Book book=new Book()

{

Id=1,

Title="ASP.NET Core 3.1", Genre="Programming for Web Development", Price=300,

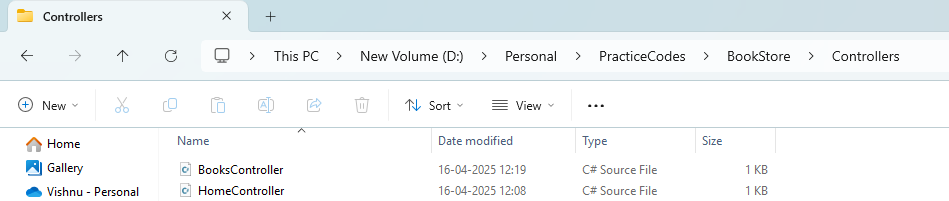
PublishDate=new System.DateTime(2019,01,28), Authors=new List<string>{"Ajay","Atul"}

};

return View(book);

}

}



1. Go to Views folder and add the folder “**Books**” and add the “**Details.cshtml**” file in Views Folder

## Code:

@model BookStore.Models.Book @{

ViewData["Title"]="Book Details";

}

<h1 class="display-4">Book Details</h1>

<table>

<tr>

<td>@Html.DisplayNameFor(model=>model.Id)</td>

<td>@Html.DisplayFor(model=>model.Id)</td>

</tr>

<tr>

<td>@Html.DisplayNameFor(model=>model.Title)</td>

<td>@Html.DisplayFor(model=>model.Title)</td>

</tr>

<tr>

<td>@Html.DisplayNameFor(model=>model.Genre)</td>

<td>@Html.DisplayFor(model=>model.Genre)</td>

</tr>

<tr>

<td>@Html.DisplayNameFor(model=>model.Price)</td>

<td>@Html.DisplayFor(model=>model.Price)</td>

</tr>

<tr>

<td>@Html.DisplayNameFor(model=>model.PublishDate)</td>

<td>@Html.DisplayFor(model=>model.PublishDate)</td>

</tr>

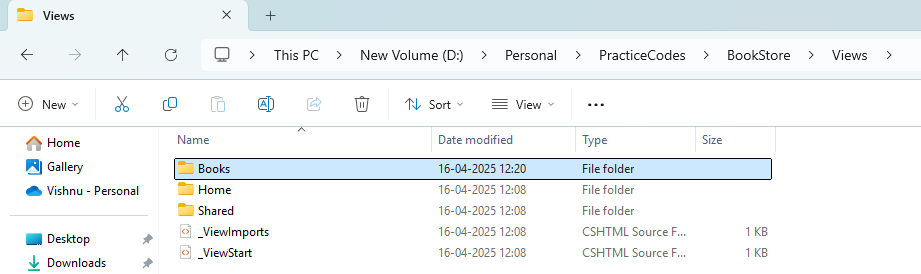
<tr>

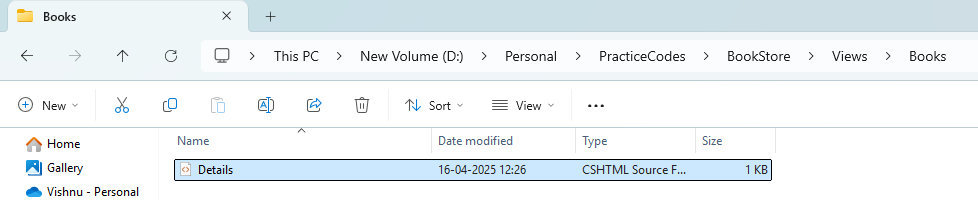
<td>@Html.DisplayNameFor(model=>model.Authors)</td>

<td>@Html.DisplayFor(model=>model.Authors)</td>

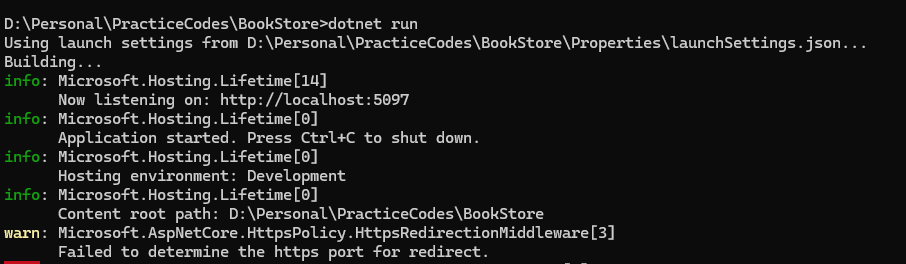
</tr>

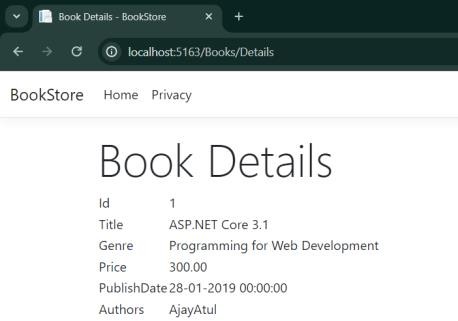
</table>





1. Open command prompt and enter “Ctrl+C” and dotnet run





## PRACTICAL NO 2(A) – BUILDING ASP .NET CORE REST API SOFTWARE REQUIREMENT:

1. Download and Install

To start building .NET apps you just need to download and install the .NET SDK (Software Development Kit version 3.0 above).

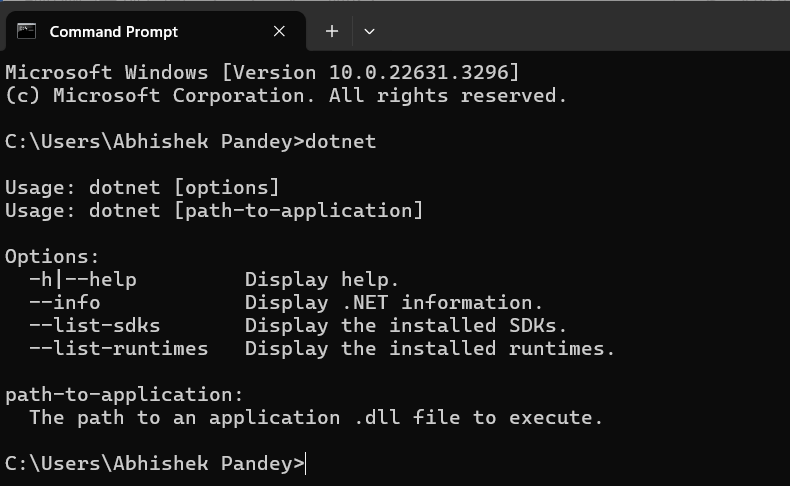
Link: <https://dotnet.microsoft.com/en-us/download/visual-studio-sdks>

1. Download the curl (Link: <https://curl.se/windows/> )
2. Check everything installed correctly Once you've installed, open a new command prompt and run the following command:

## Command:

Command to check whether .net sdk have been downloaded properly. dotnet

## Output:

****

Steps: Create your web API

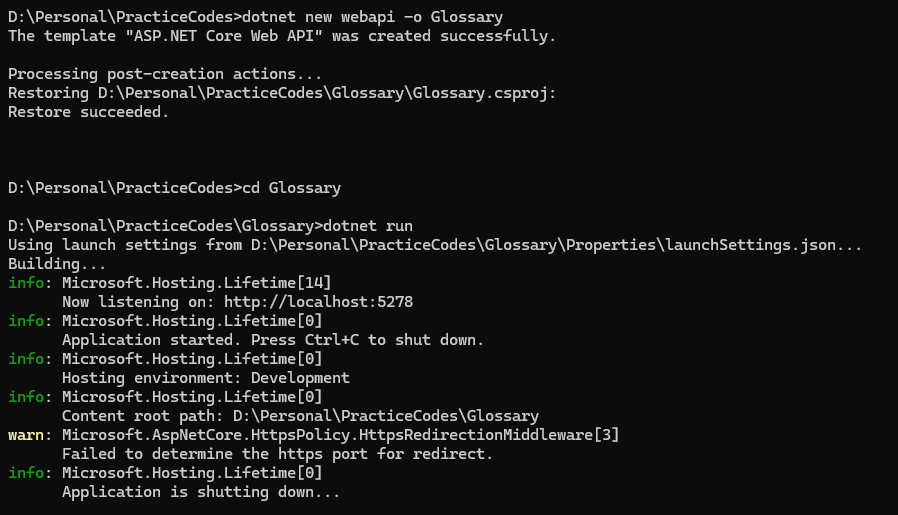
1. Open two command prompts

## Command prompt 1:

dotnet new webapi -o Glossary cd Glossary

dotnet run

## Output:

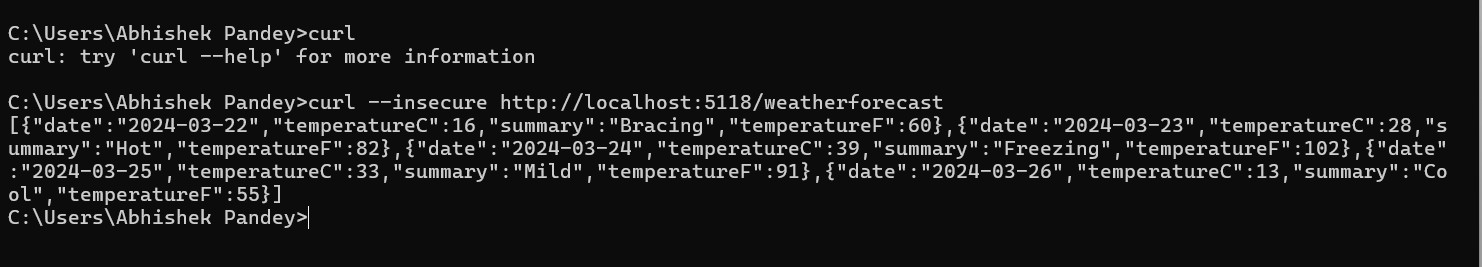
****

**Open Command prompt 2**

Command:

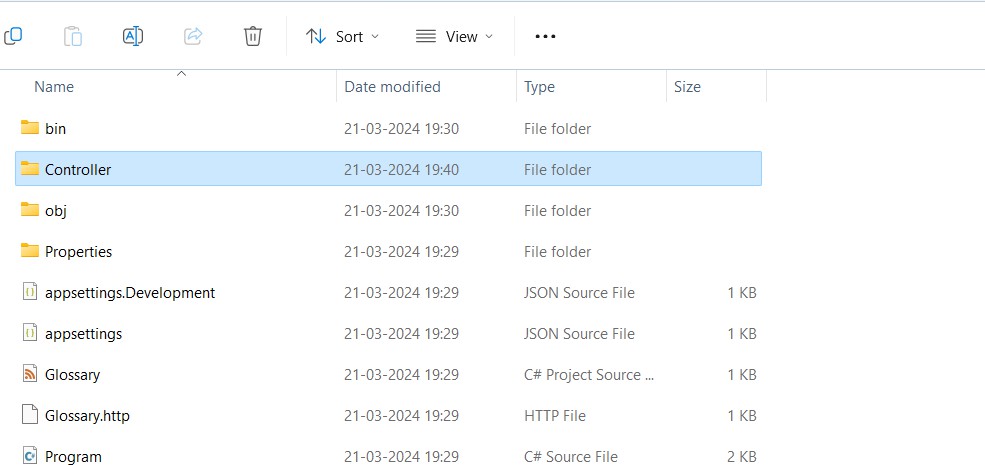
curl

curl --insecure <http://localhost:5118/weatherforecast> Output:



1. Now Change the content:

To get started, create a Controller Folder in Glossary Folder



1. In Controller Folder create a “**GlossaryController.cs**” file in it.

## Code:

using System;

using System.Collections.Generic; using Microsoft.AspNetCore.Mvc; using System.IO;

namespace Glossary.Controllers

{

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

public class GlossaryController: ControllerBase

{

private static List <GlossaryItem>Glossary = new List<GlossaryItem>

{

new GlossaryItem

{

Term= "HTML",

Definition = "Hypertext Markup Language"

},

new GlossaryItem

{

Term= "MVC",

Definition = "Model View Controller"

},

new GlossaryItem

{

Term= "OpenID",

Definition = "An open standard for authentication"

}

};

[HttpGet]

public ActionResult<List<GlossaryItem>> Get()

{

return Ok(Glossary);

}

[HttpGet] [Route("{term}")]

public ActionResult<GlossaryItem> Get(string term)

{var glossaryItem = Glossary.Find(item => item.Term.Equals(term, StringComparison.InvariantCultureIgnoreCase));

if (glossaryItem == null)

{

return NotFound();

} else

{ return Ok(glossaryItem);

}

}

[HttpPost]

public ActionResult Post(GlossaryItem glossaryItem)

{

var existingGlossaryItem = Glossary.Find(item => item.Term.Equals(glossaryItem.Term,StringComparison.InvariantCultureIgnoreCase)

);

if (existingGlossaryItem != null)

{

return Conflict("Cannot create the term because it already exists.");

}

else

{ Glossary.Add(glossaryItem);

var resourceUrl = Path.Combine(Request.Path.ToString(), Uri.EscapeUriString(glossaryItem.Term));

return Created(resourceUrl, glossaryItem);

}

}

[HttpPut]

public ActionResult Put(GlossaryItem glossaryItem)

{

var existingGlossaryItem = Glossary.Find(item => item.Term.Equals(glossaryItem.Term,

StringComparison.InvariantCultureIgnoreCase)); if (existingGlossaryItem == null)

{

return BadRequest("Cannot update a nont existing term.");

}

else

{existingGlossaryItem.Definition = glossaryItem.Definition; return Ok();

}

}

[HttpDelete] [Route("{term}")]

public ActionResult Delete(string term)

{

var glossaryItem = Glossary.Find(item => item.Term.Equals(term, StringComparison.InvariantCultureIgnoreCase));

if (glossaryItem == null)

{ return NotFound();

}

else

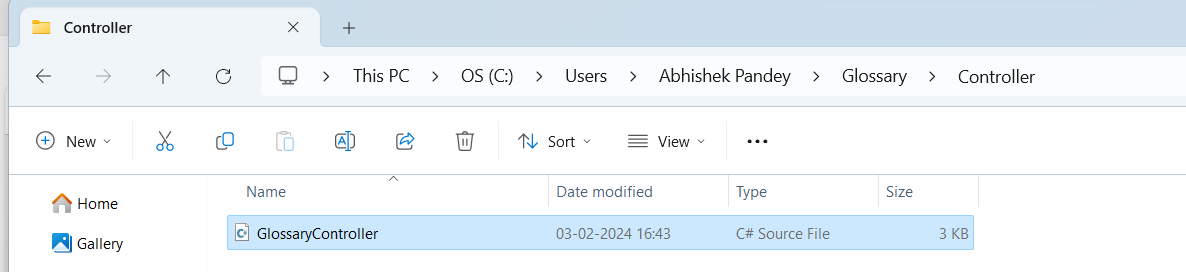
{ Glossary.Remove(glossaryItem); return NoContent();

}

}

}

}



1. In Glossary Folder create a file “**GlossaryItem.cs” Code:**

namespace Glossary

{

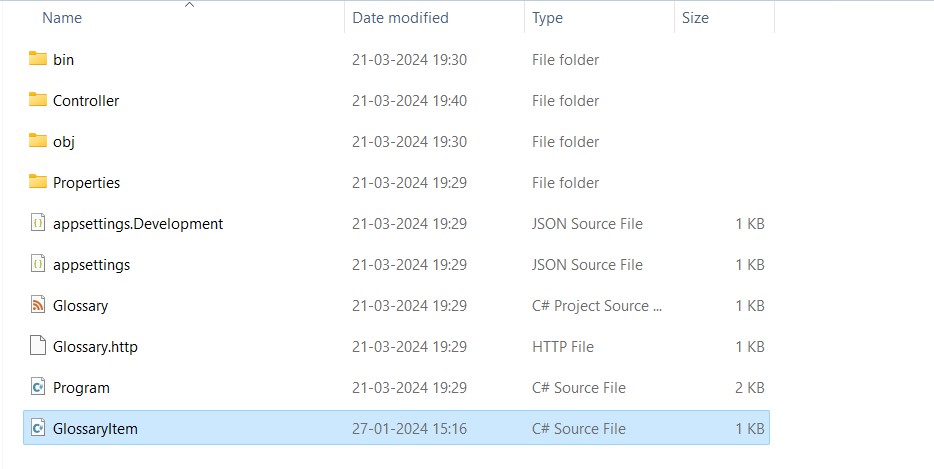
public class GlossaryItem

{

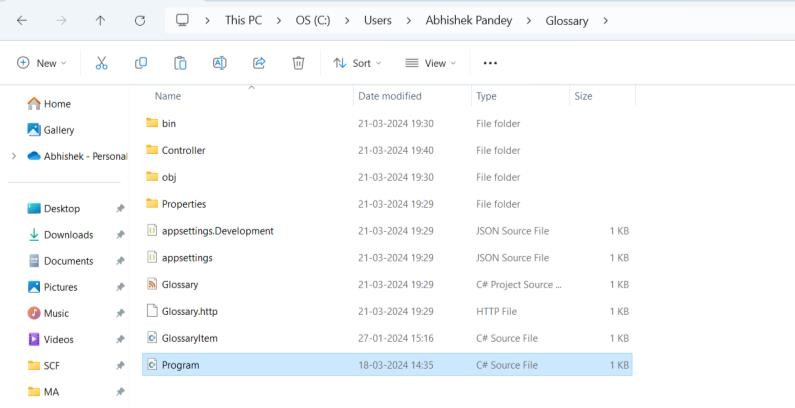
public string Term {get;set;} public string Definition{get;set;}

}

}



1. Now Change the “Program.cs” code



Code:

var builder = WebApplication.CreateBuilder(args);

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

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

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();

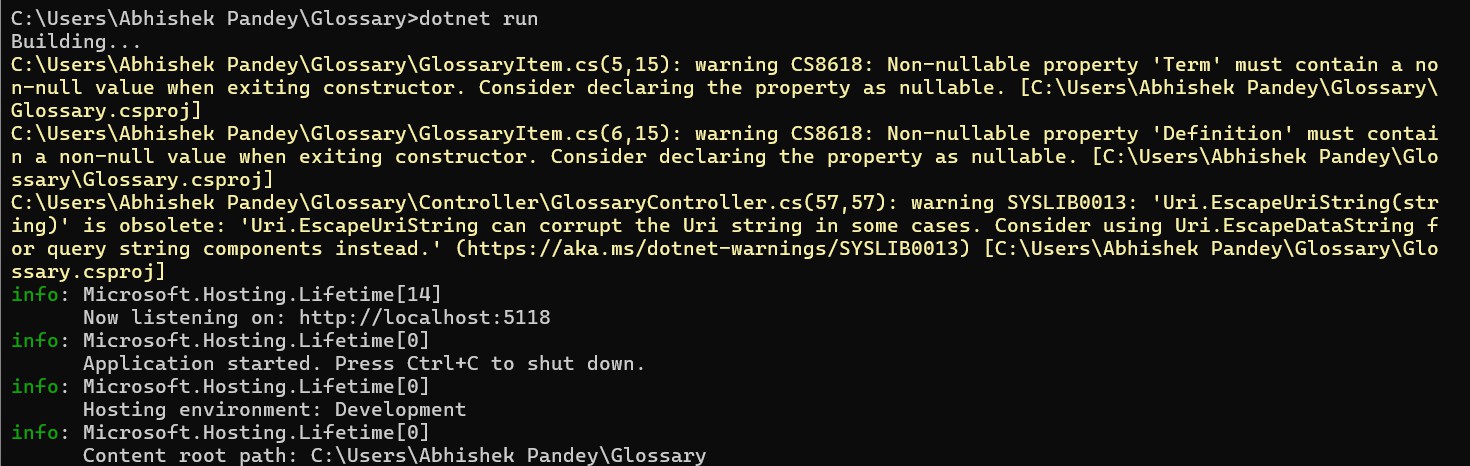
app.Run();

1. Now stop running previous dotnet run on command prompt 1 using Ctrl+C and run it again for new code on Command Prompt 1

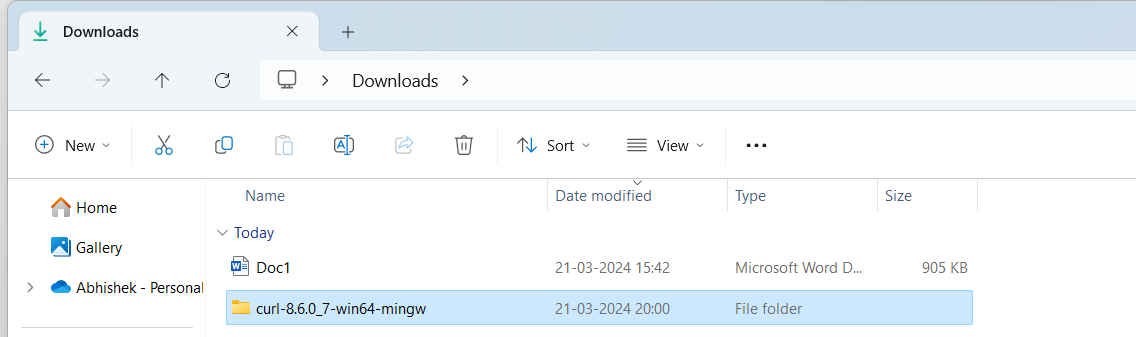
## Command:

dotnet run

## Output:

****

1. Now Extract the curl in downloads folder

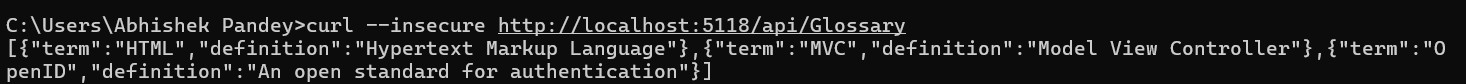


## On Command Prompt 2:

1. **Getting a list of Item Command:**

curl --insecure <http://localhost:5118/api/Glossary>

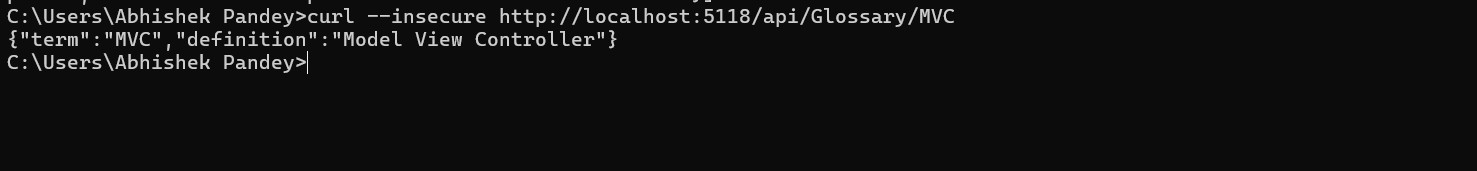
## Output:

****

1. **Getting a single Item Command:**

curl --insecure <http://localhost:5118/api/Glossary/MVC>

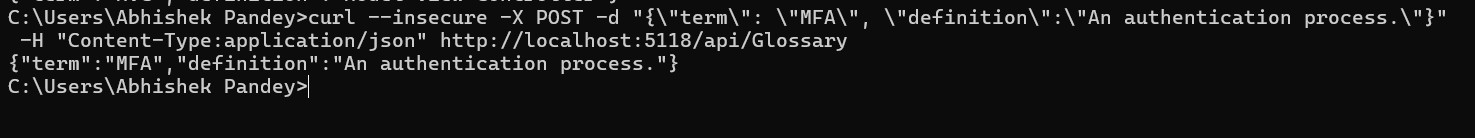
## Output:

****

1. **Creating an Item Command:**

curl --insecure -X POST -d "{\"term\": \"MFA\", \"definition\":\"An authentication process.\"}" -H "Content-Type:application/json" <http://localhost:5118/api/Glossary>

## Output:

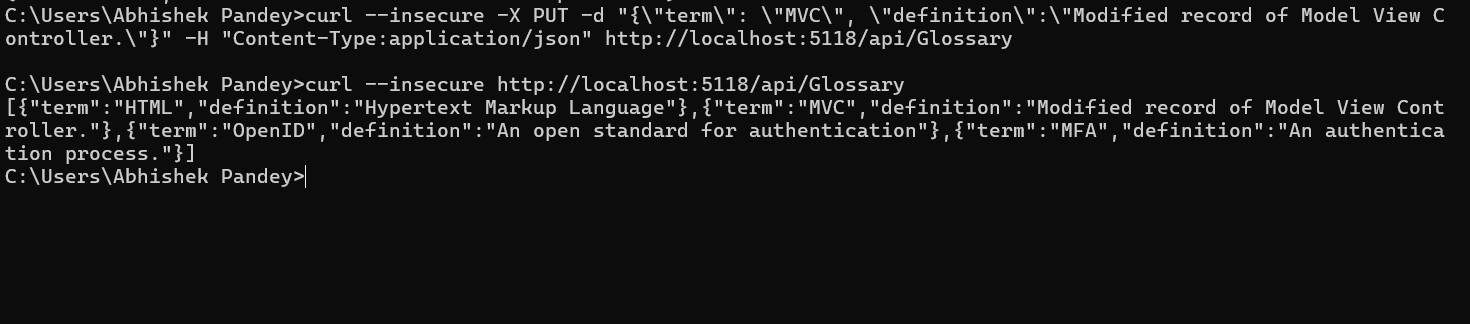
****

1. **Update Item Command:**

curl --insecure -X PUT -d "{\"term\": \"MVC\", \"definition\":\"Modified record of Model View Controller.\"}" -H "Content-Type:application/json" <http://localhost:5118/api/Glossary>

curl --insecure <http://localhost:5118/api/Glossary>

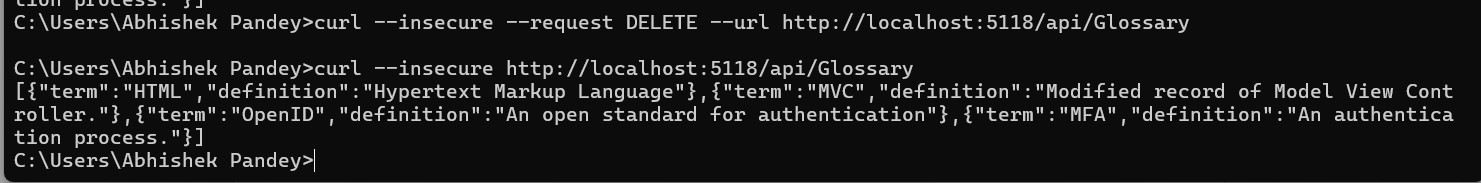
## Output:

****

1. **Delete Item Command:**

curl --insecure --request DELETE –url [http://localhost:5118/api/Glossary](http://localhost:5118/api/Glossary/d) curl --insecure <http://localhost:5118/api/Glossary>

## Output:

****

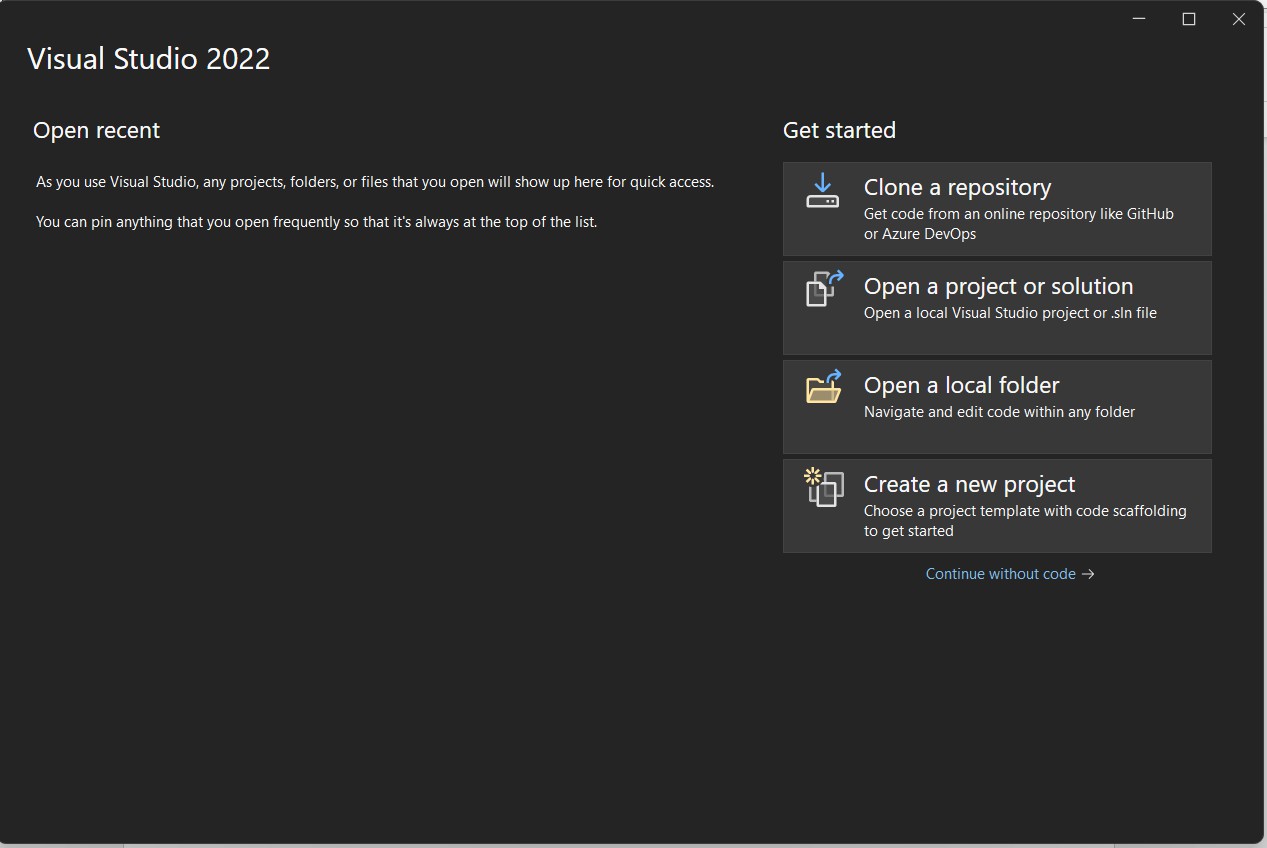
**PRACTICAL NO 2(B) – BUILDING ASP.NET CORE REST API**

Steps:

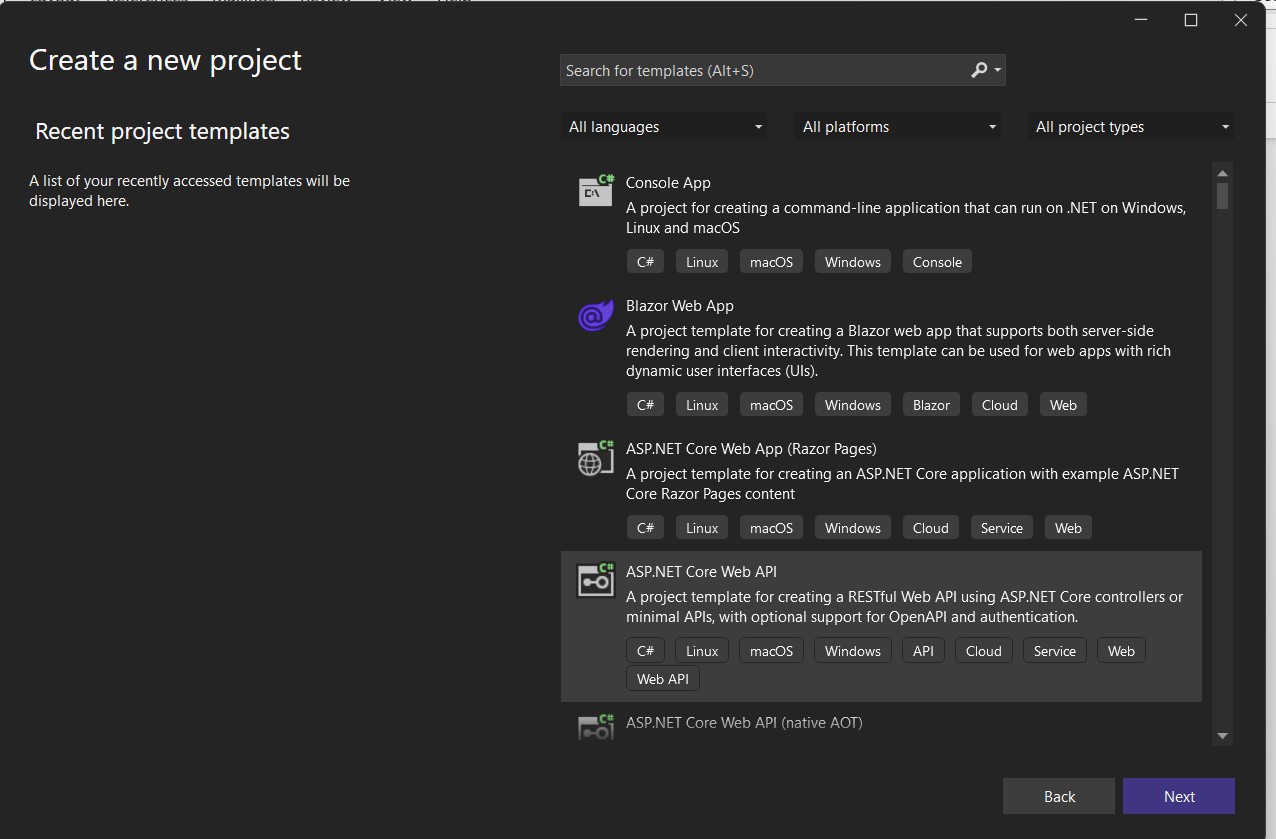
1. Open Visual Studio 2022.



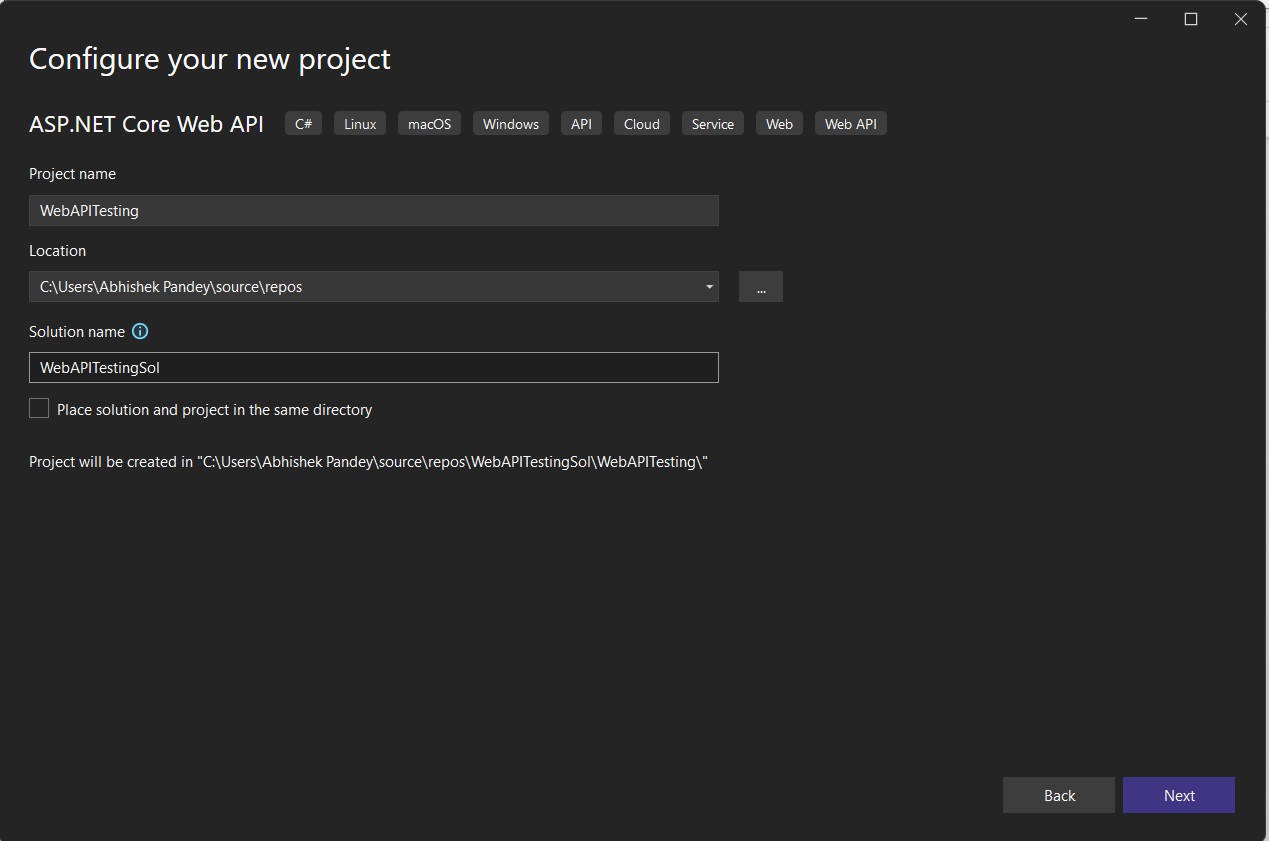
1. Create a Project Window opens



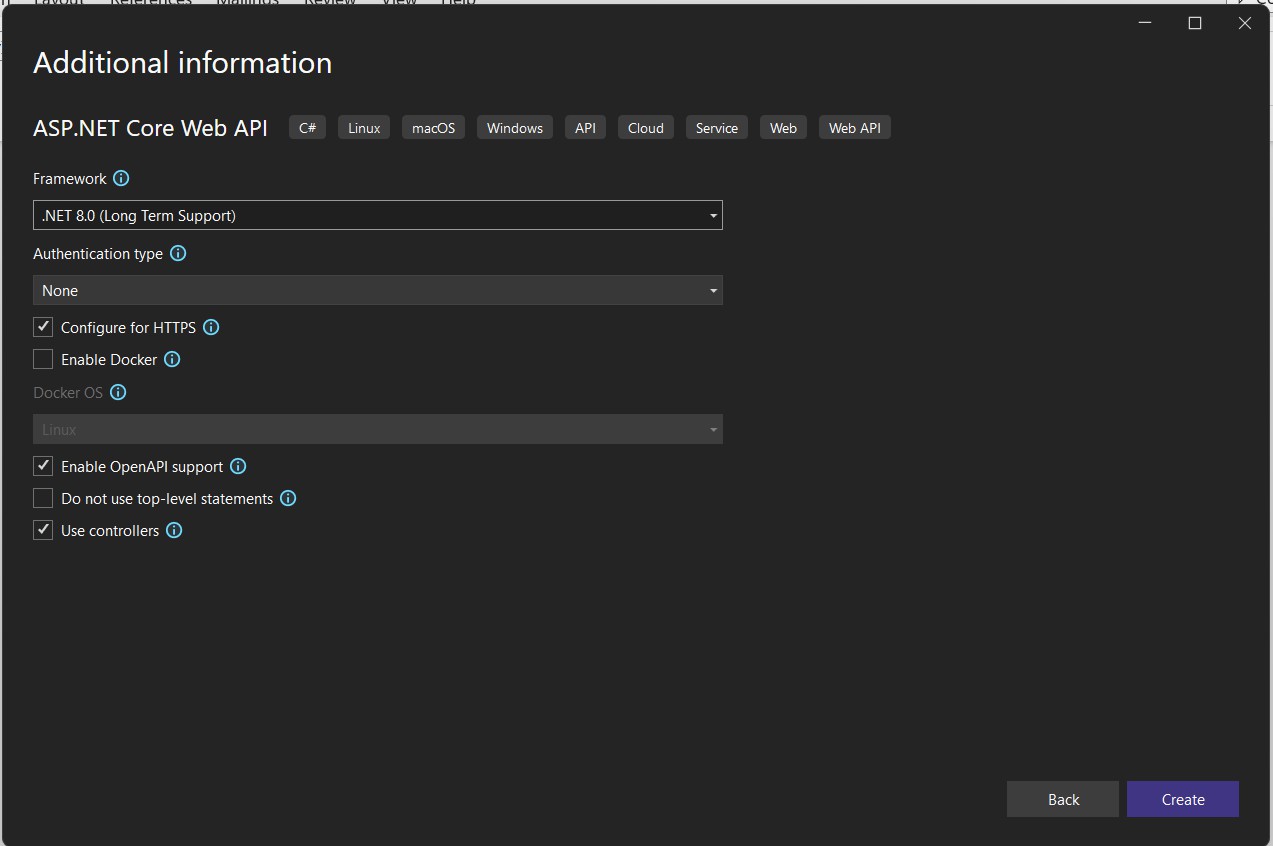
1. Select the project type (ASP.NET Core Web Api)



1. Name the Project (WebApi)



1. Select the Target Framework. ( .NET 7.0)

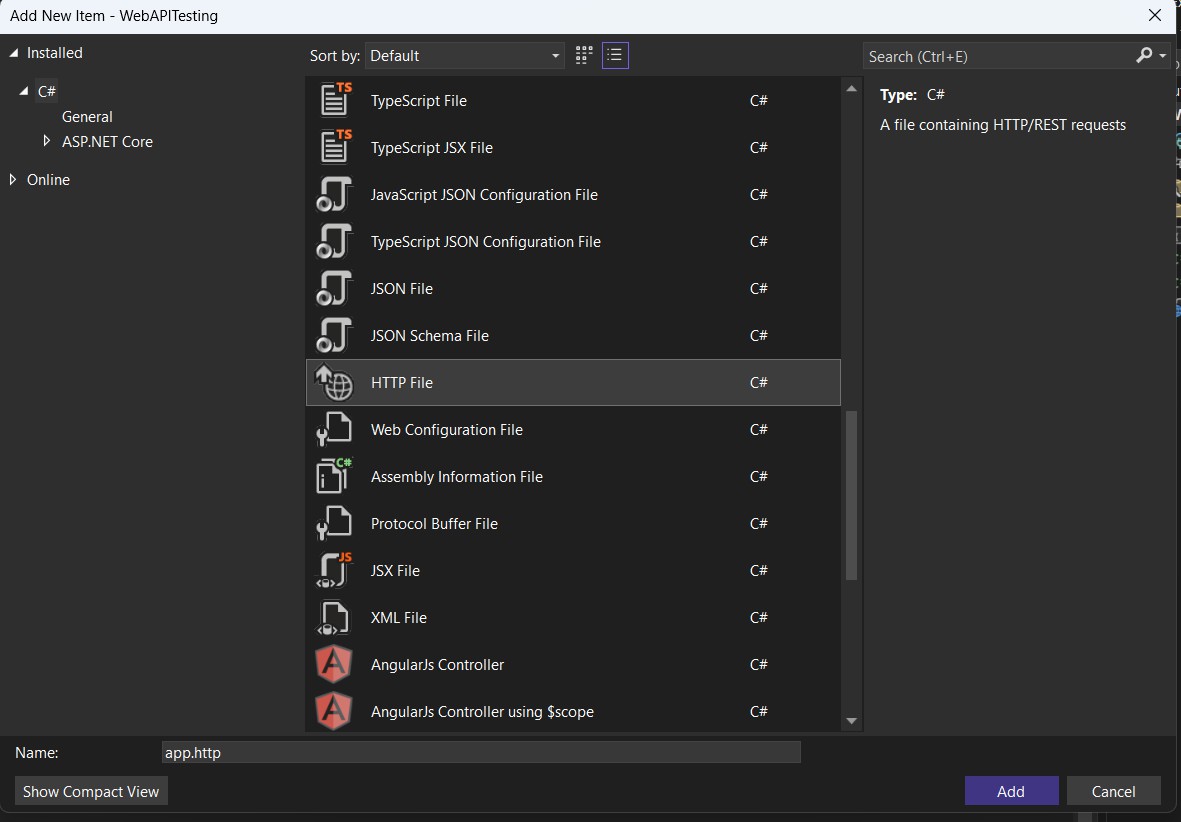
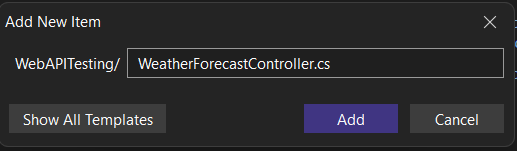


1. This will create the standard API project with the Weather Forecast controller. Next, add a new http file as below:

(Right Click on WebAPITesting to create http file)



1. Give name “**WeatherForecastController**” Show all templates and select “**HTTP File**”



1. Go to Controller and update **“WeatherForecastController.cs”**

using Microsoft.AspNetCore.Mvc;

namespace WebAPITesting.Controllers

{

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

public class WeatherForecastController : ControllerBase

{

private static readonly string[] Summaries = new[]

{

"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot",

"Sweltering", "Scorching"

};

private readonly ILogger<WeatherForecastController> \_logger;

public WeatherForecastController(ILogger<WeatherForecastController> logger)

{

\_logger = logger;

}

[HttpGet(Name = "GetWeatherForecast")] public IEnumerable<WeatherForecast> Get()

{

return Enumerable.Range(1, 5).Select(index => new WeatherForecast

{

Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)), TemperatureC = Random.Shared.Next(-20, 55),

Summary = Summaries[Random.Shared.Next(Summaries.Length)]

})

.ToArray();

}

[HttpPost("{id}", Name = "GetLocationDetails")]

public IActionResult GetLocationDetails(int id)

{

return Ok(new LocationDetails

{

Id = id,

Name = "Test Location", Address = "Test Address", City = "Test City",

State = "Test State", ZipCode = "Test ZipCode"

});

}

private class LocationDetails

{

public int Id { get; set; }

public string? Name { get; set; } public string? Address { get; set; }

public string? City { get; set; } public string? State { get; set; } public string? ZipCode { get; set; }

}

}

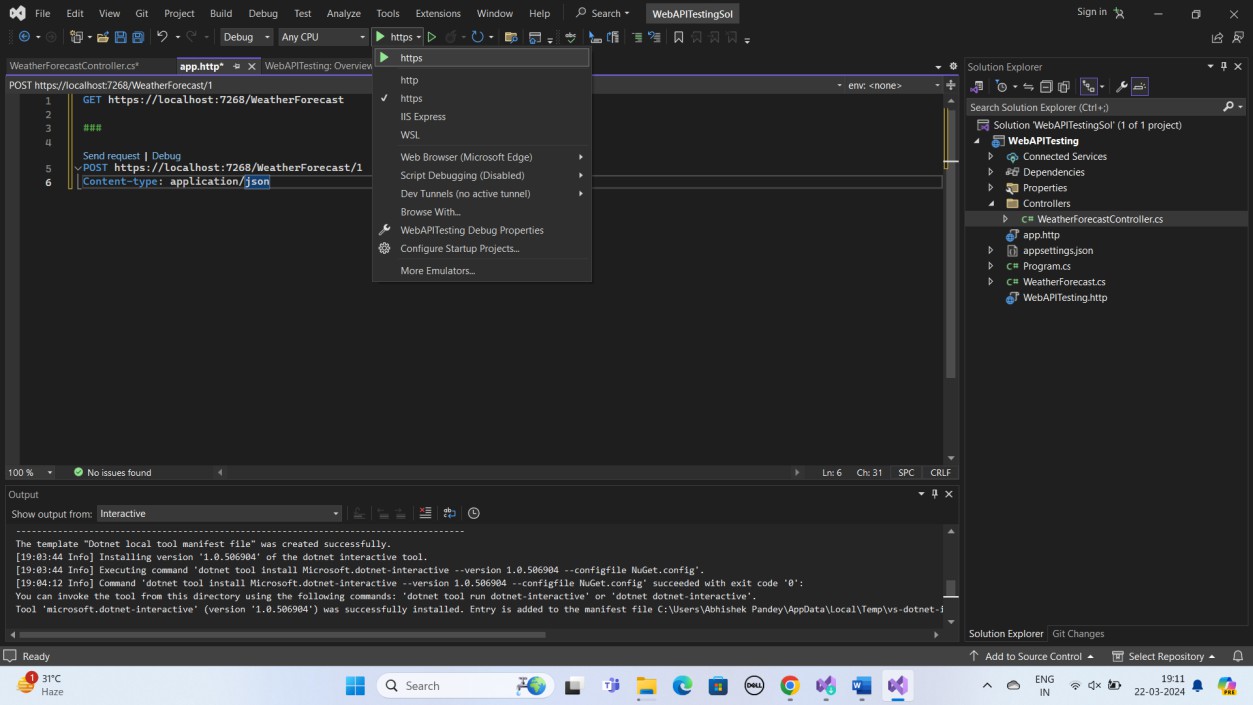
}

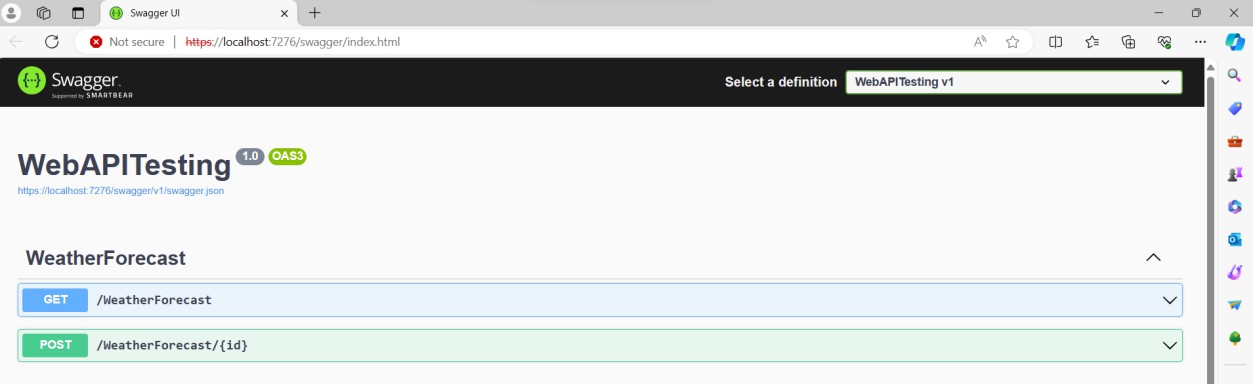
## Finally, update the code in the “app.http” file as below:

GET https://localhost:7268/WeatherForecast ###

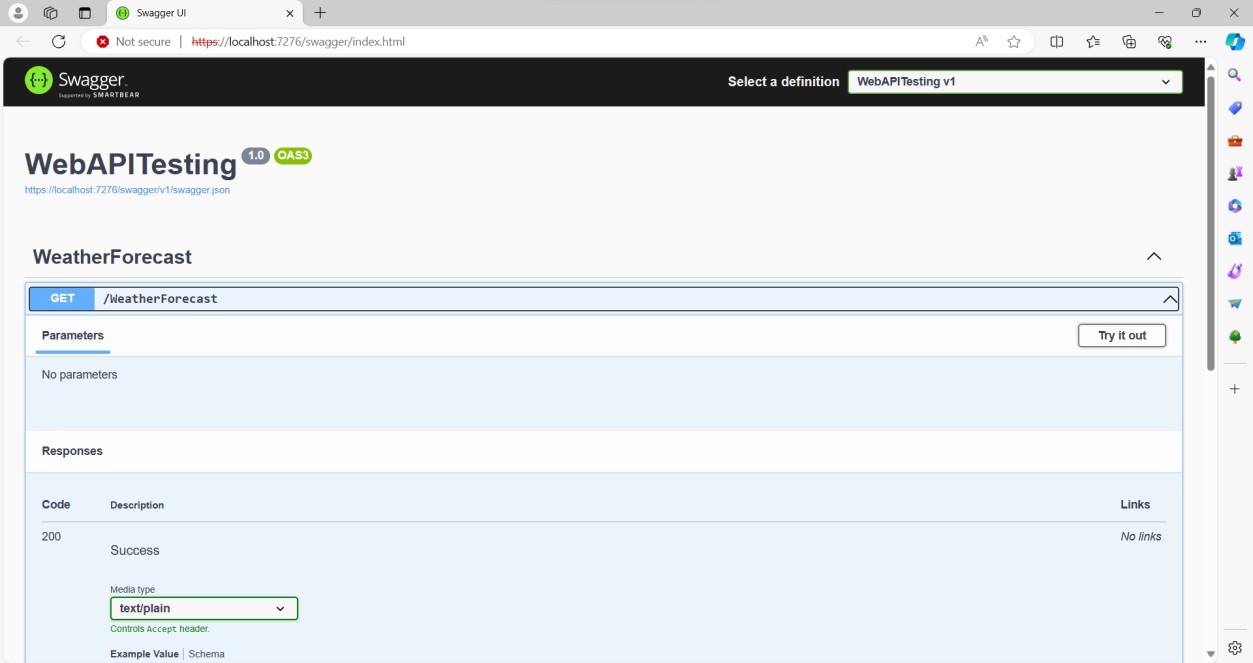
POST https://localhost:7268/WeatherForecast/1 Content-type: application/json

1. **Now run the file**

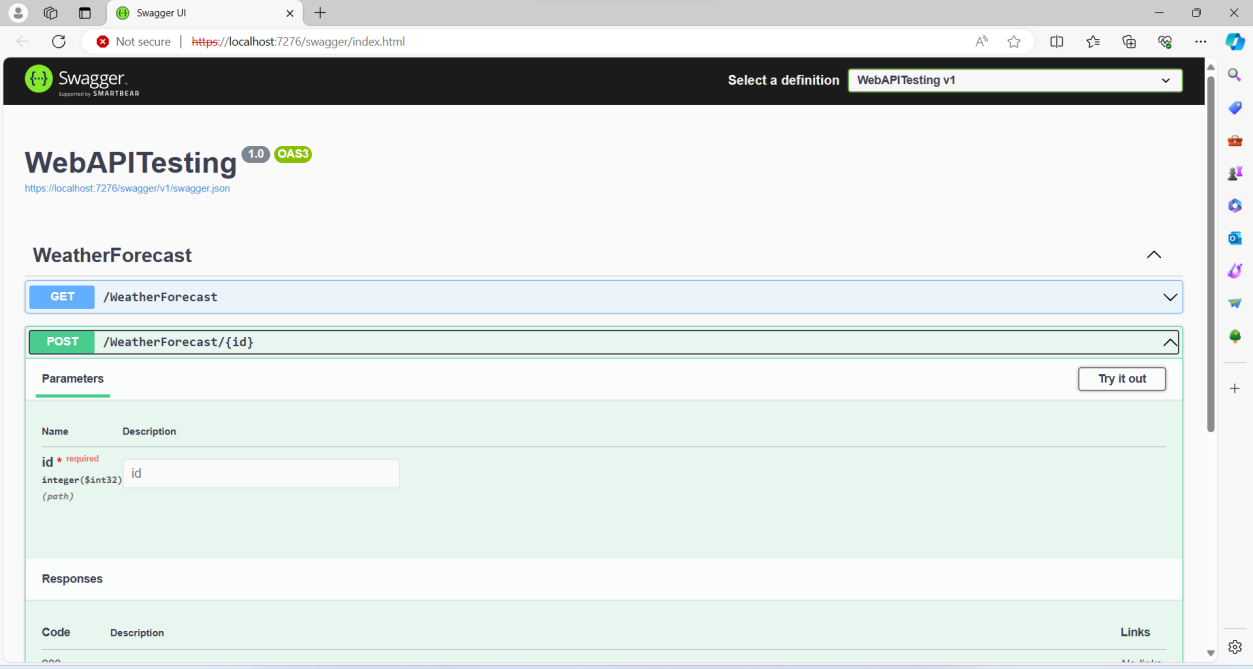
****

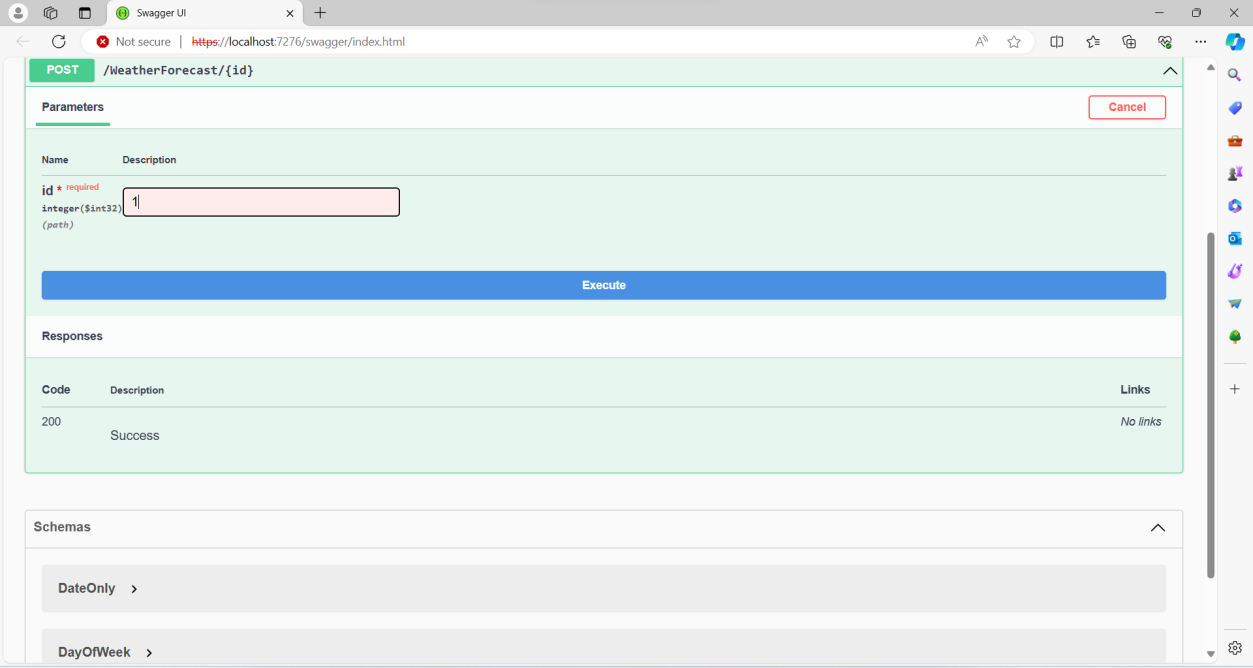
****

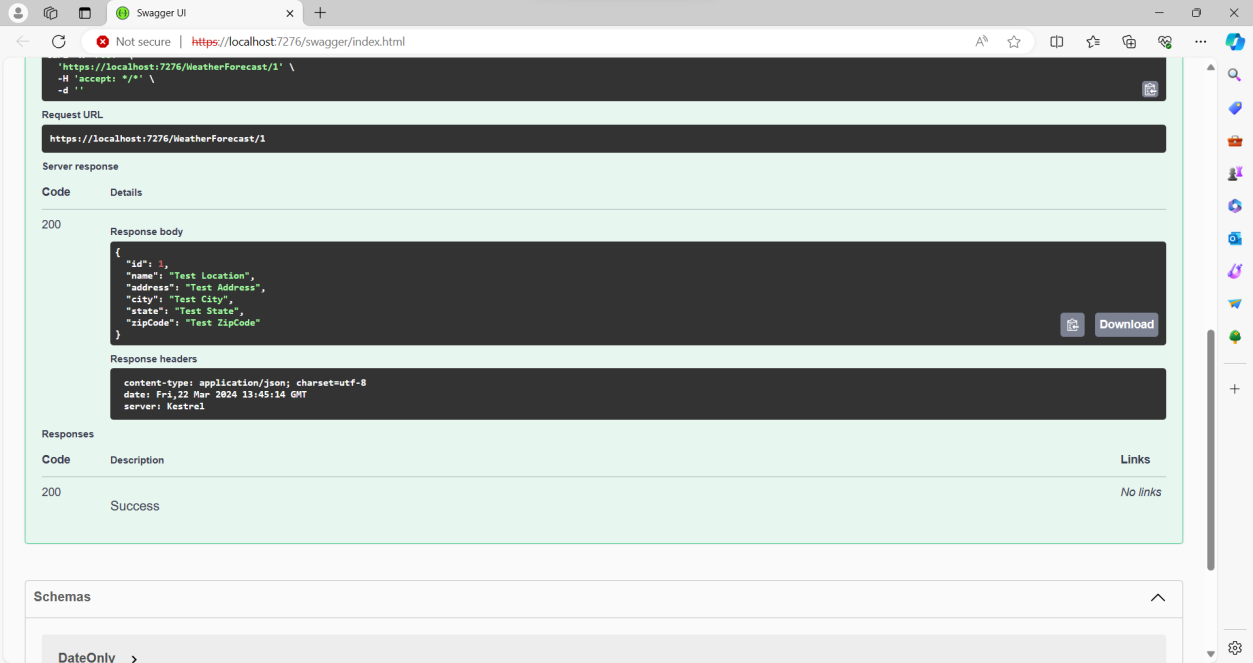
1. **Click on try out**

****

1. **Click on Try out**

****

****



**PRACTICAL NO 3 – BUILDING .NET CORE MVC (Hello World &**

**StockQuote)**

STEPS:

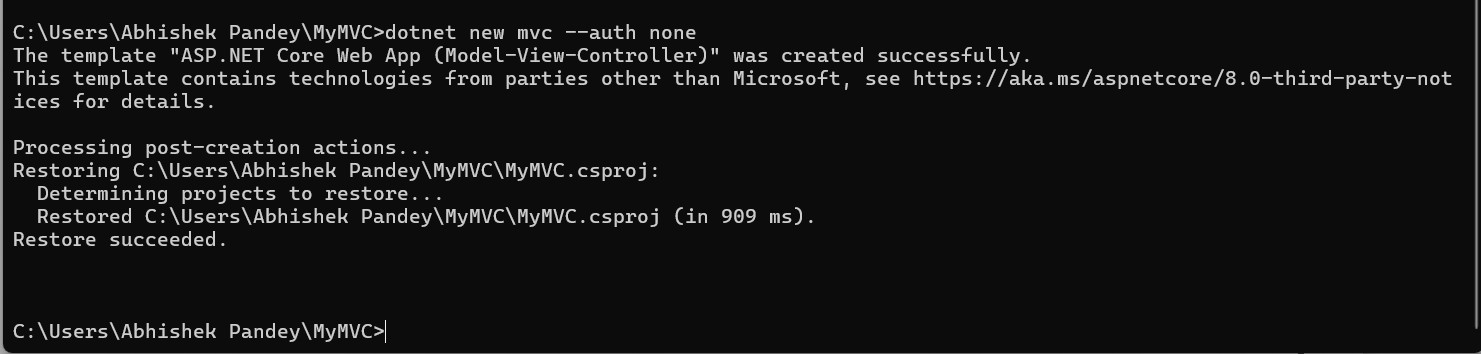
1. Install .Net Core Sdk (Link: <https://dotnet.microsoft.com/en-us/download> )
2. Create folder MyMVC folder in D: drive or any other drive
3. Open command prompt and perform following operations

## Command:

cd MyMVC

dotnet new mvc --auth none

# Output:

****

1. Go to Controllers folder and modify **HomeController.cs** file to match following: Code:

using System;

using System.Collections.Generic; using System.Diagnostics;

using System.Linq;

using System.Threading.Tasks; using Microsoft.AspNetCore.Mvc; using Microsoft.Extensions.Logging; using MyMVC.Models;

namespace MyMVC.Controllers

{ public class HomeController : Controller

{

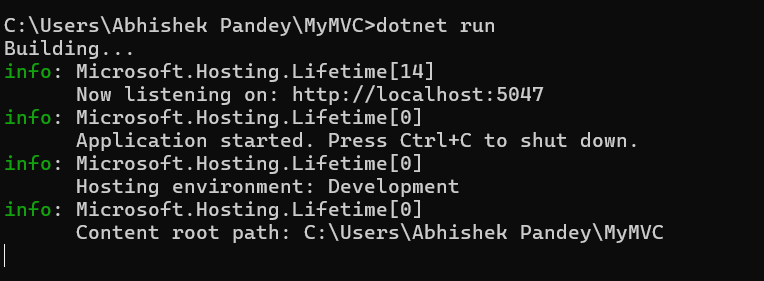
public String Index()

{ return "Hello World"; }

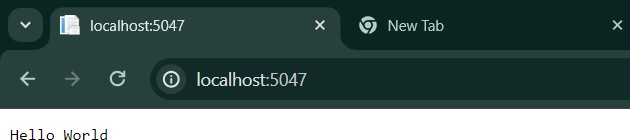
}

}

## Output:

****

1. Go to website ( [http://localhost:5047](http://localhost:5047/) ) and see the ouput
2. Now go back to command prompt and stop running project using CTRL+C



1. Go to Models folder and add new file “StockQuote.cs” to it with following content

## Code:

using System;

namespace MYMVC.Models

{

public class StockQuote

{

public string Symbol{get;set;} public int Price{get;set;}

}

}

1. Go to View folder then home folder in it and modify index.cshtml file to match following

## Code:

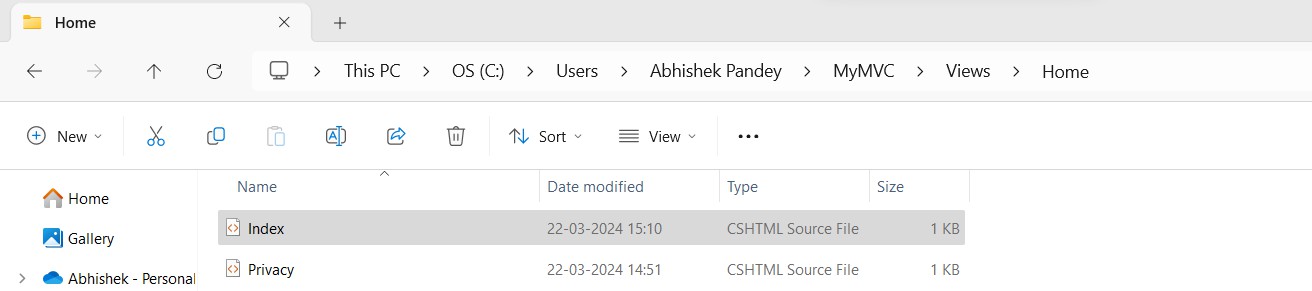
@{

ViewData["Title"] = "Home Page";

}

<div> Symbol:@Model.Symbol</br> Price:$@Model.Price</br>

</div>



1. Again go back to Controllers Folder and make the changes in “**HomeController.cs**”

## Code:

using System;

using System.Collections.Generic; using System.Diagnostics;

using System.Linq;

using System.Threading.Tasks; using Microsoft.AspNetCore.Mvc; using Microsoft.Extensions.Logging; using MyMVC.Models;

namespace MyMVC.Controllers

{

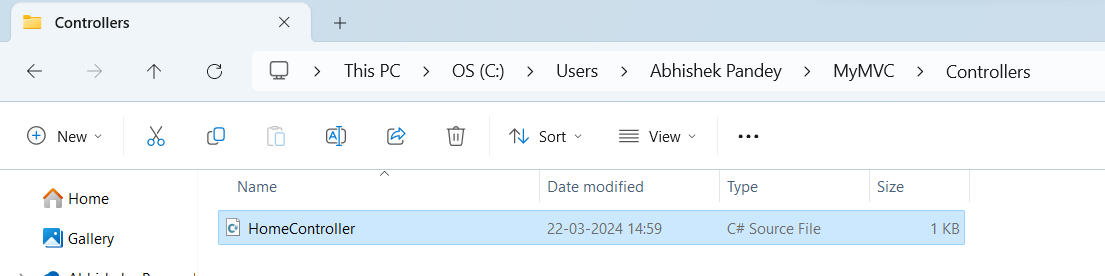
public class HomeController:Controller

{

public async Task<IActionResult>Index()

{

var model=new StockQuote{Symbol="Hello",Price=3200}; return View(model);

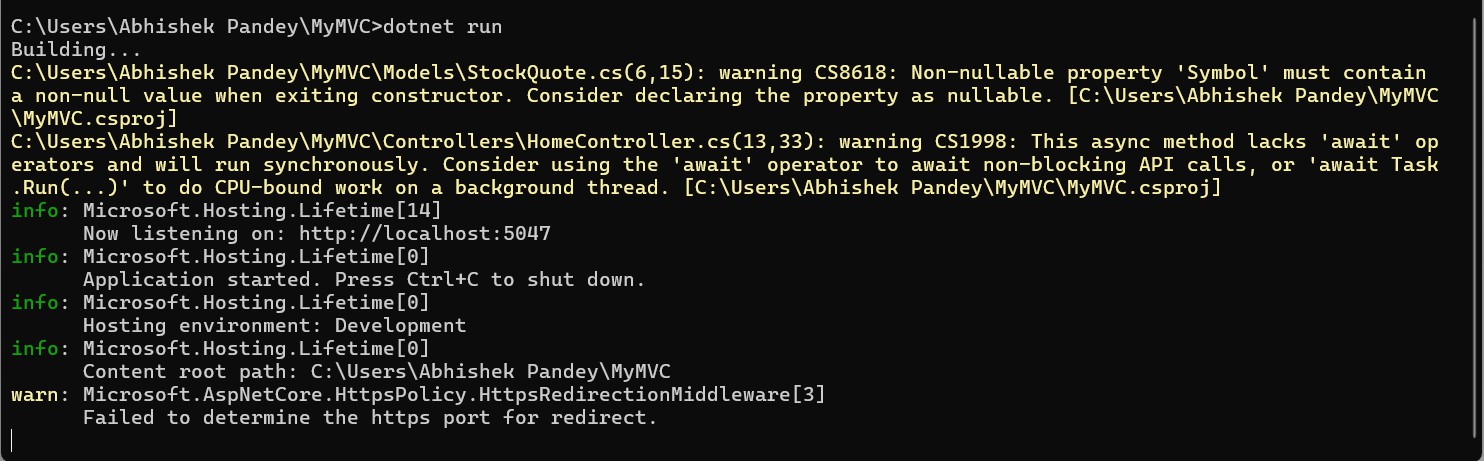


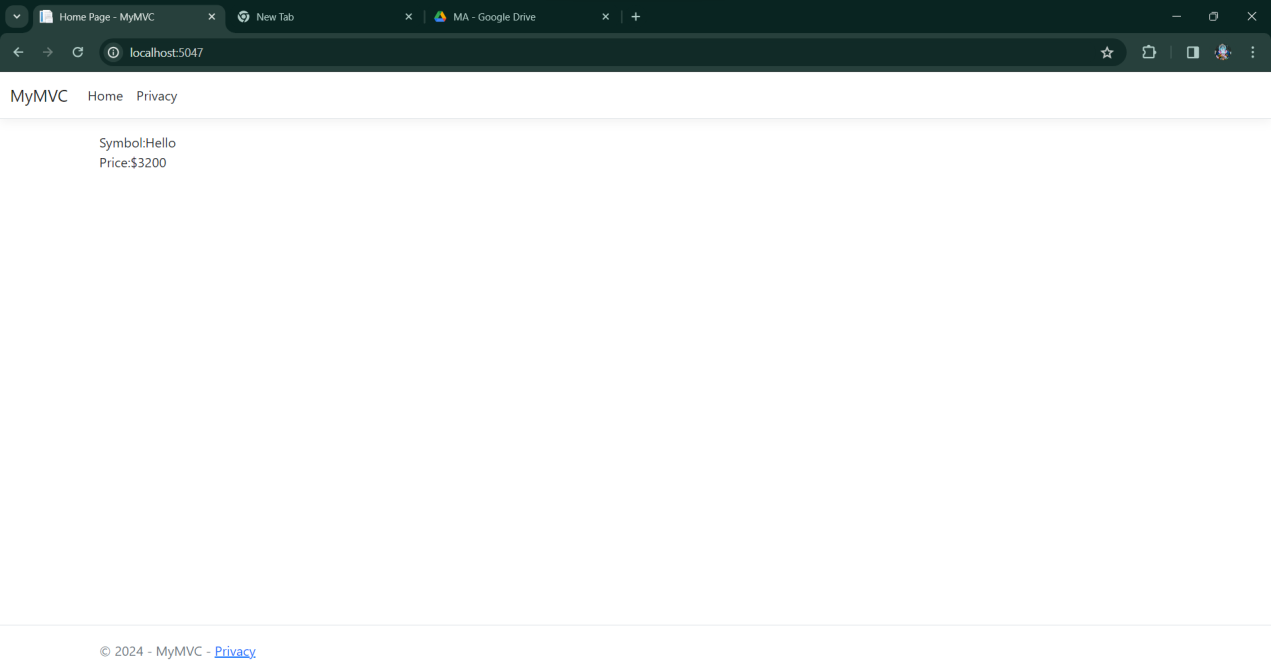
1. Now run the project.

## Code:

dotnet ru

**Output:**



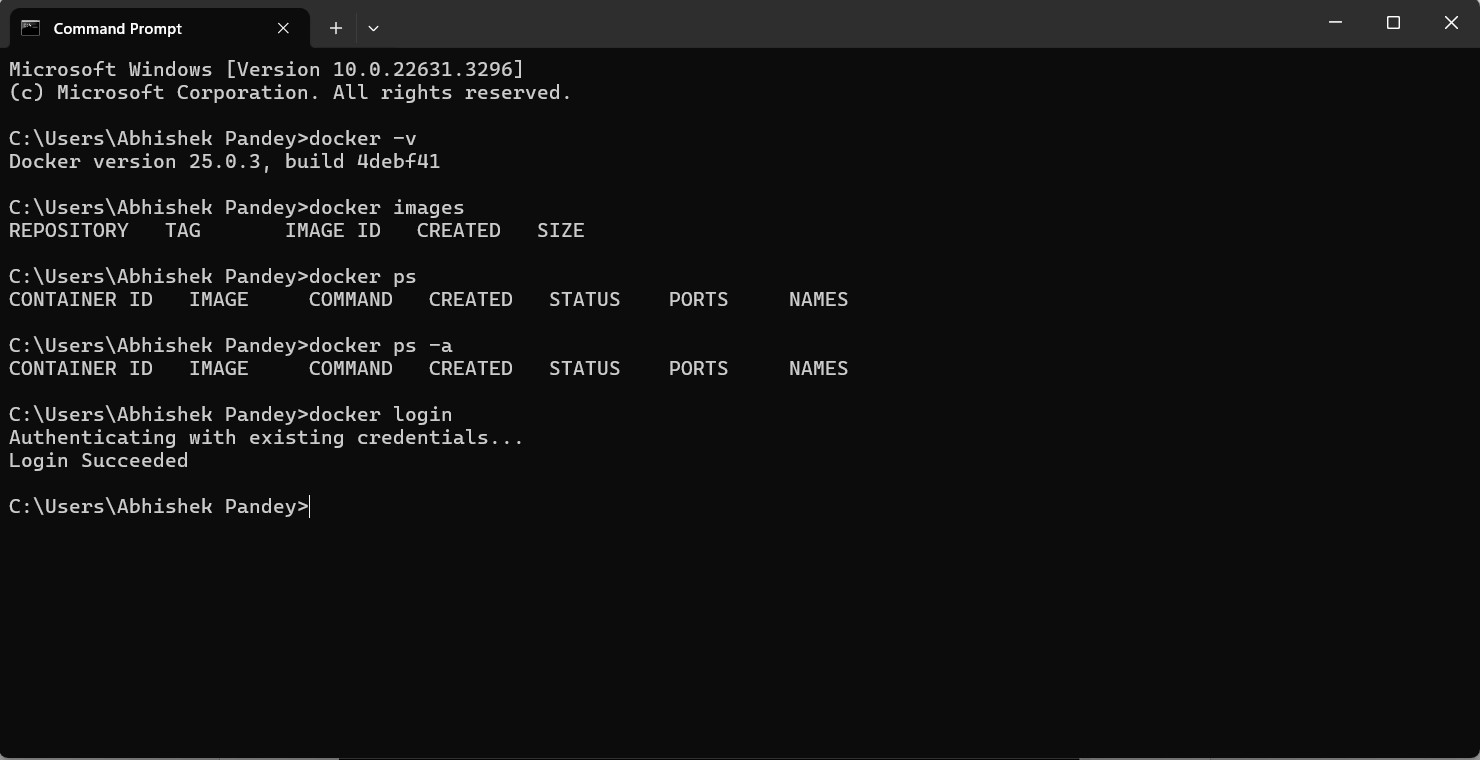


**PRACTICAL NO 4 – WORKING WITH DOCKER COMMANDS, IMAGES AND CONTAINER**

**Command:**

docker -v docker images docker ps docker ps -a docker login

## Output:

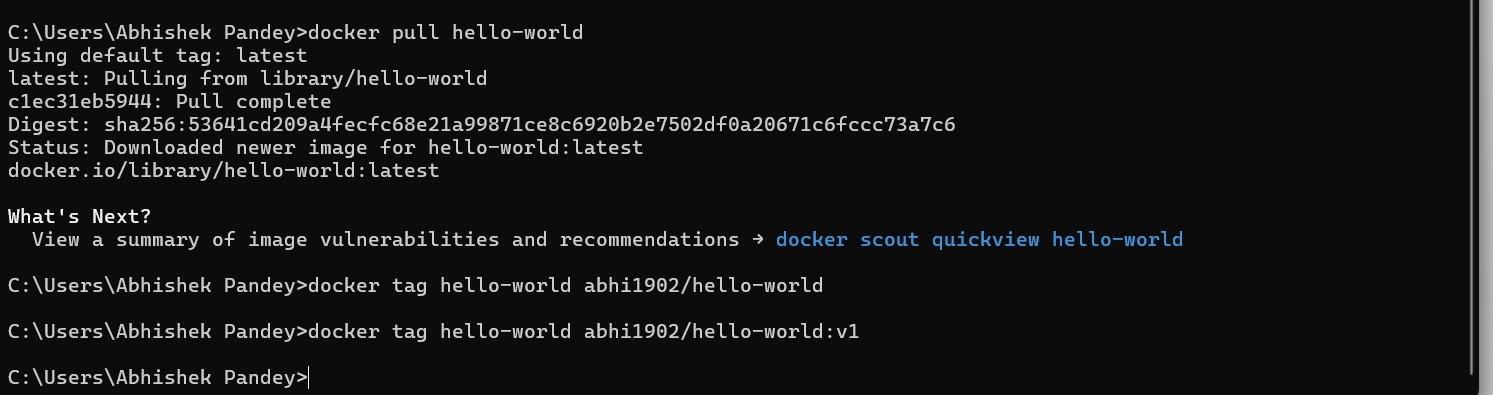
****

**Command:**

docker pull hello-world

docker tag hello-world abhi1902/hello-world docker tag hello-world abhi1902/hello-world:v1

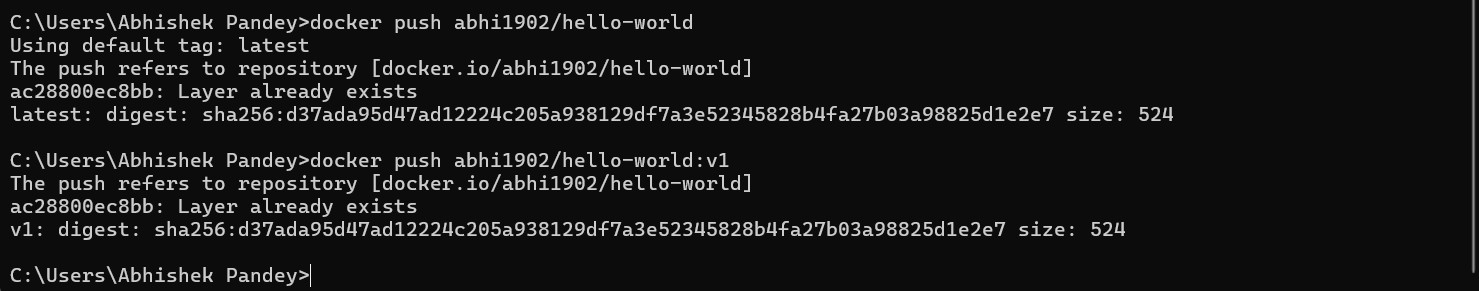
# Output:

****

## Command:

docker push abhi1902/hello-world docker push abhi1902/hello-world:v1

**Output:**

****

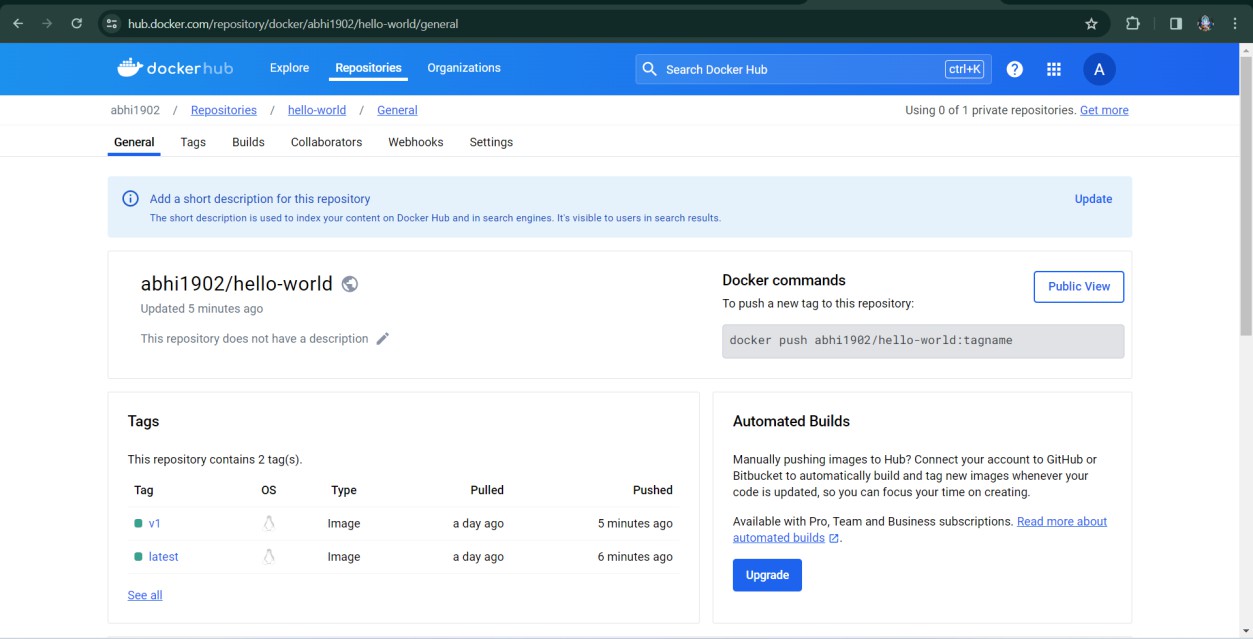
**Go to Docker hub and login and see the repository**

**Command:**

docker images docker ps docker ps -a

## Output:

****

**Command:**

docker rmi -f d2c94e258dcb (Image ID)

## Output:

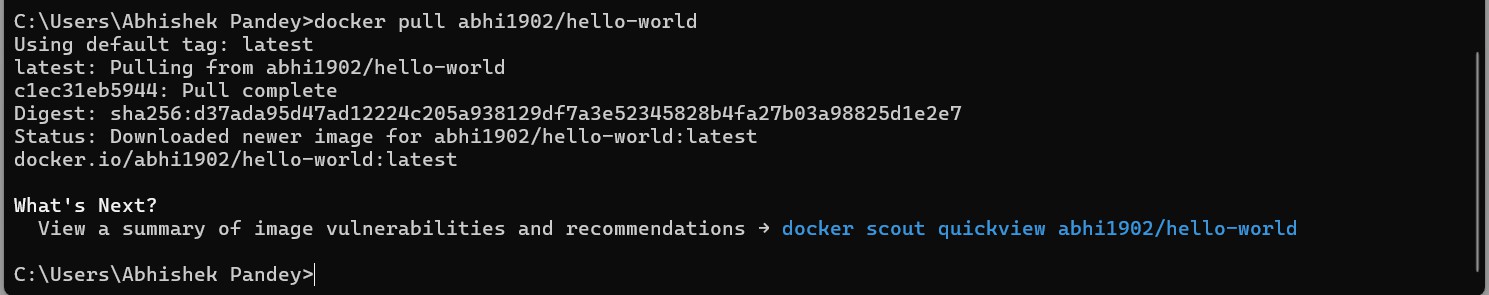
****

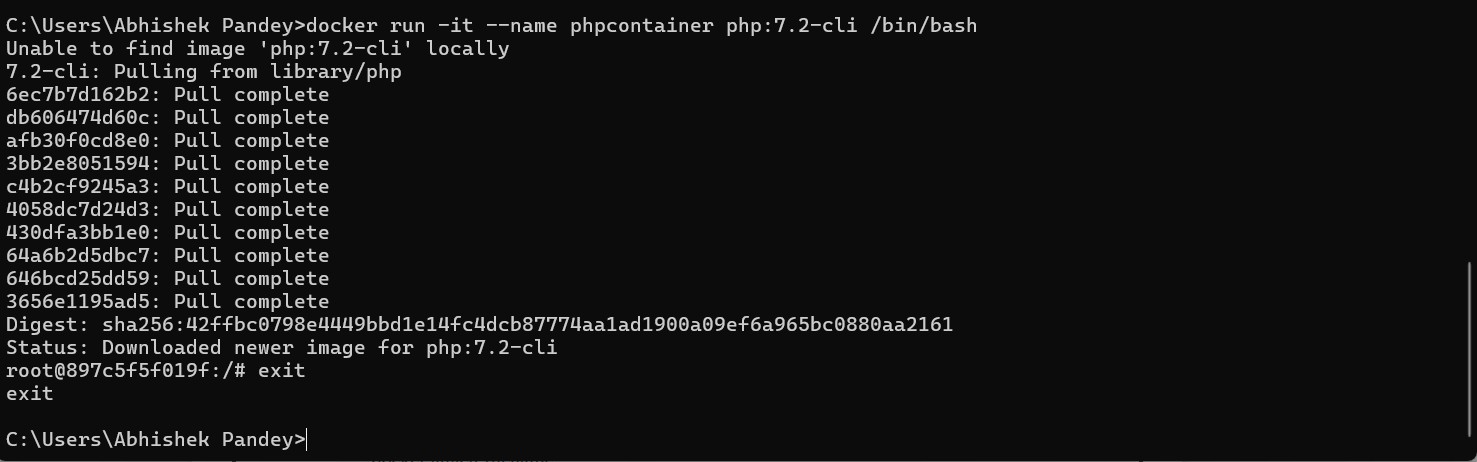
**Command:**

docker pull abhi1902/hello-world

docker run -it --name phpcontainer php:7.2-cli /bin/bash #exit

## Output:

****

****

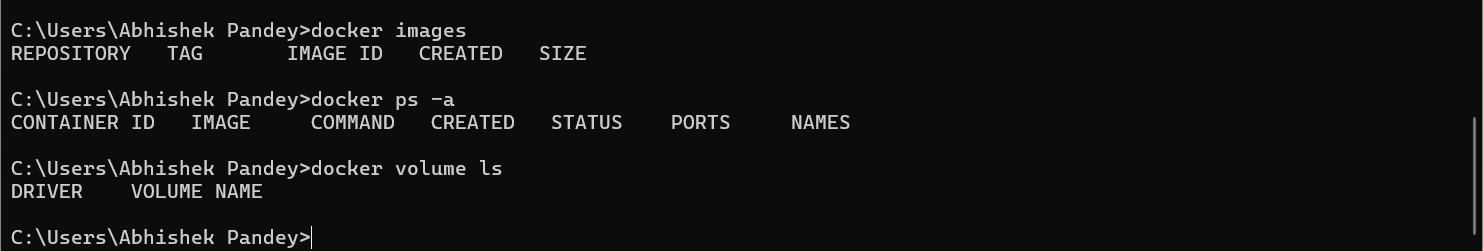
**Command:**

docker ps -a

docker system prune -a docker images

docker ps -a docker volume ls

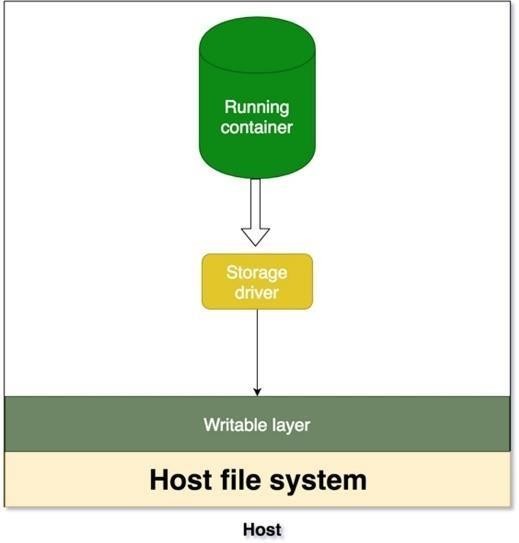
**Output:**

****

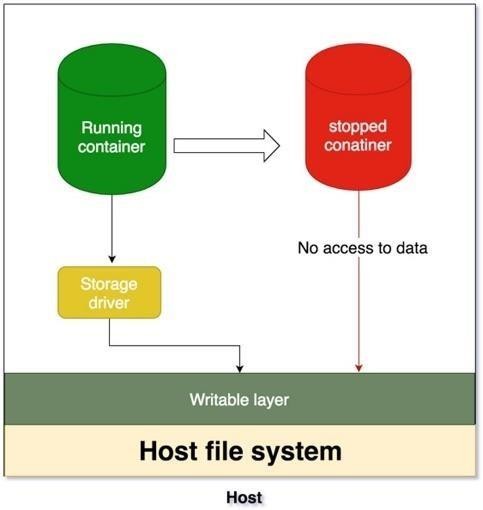
**PRACTICAL NO 5 – WORKING WITH DOCKER VOLUMES AND**

**NETWORKS (Play with Docker)**

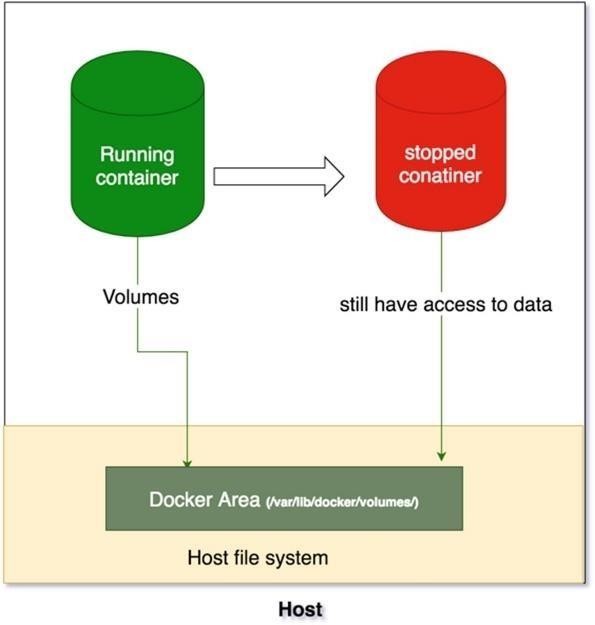
Before going deep into volumes, Let’s understand how containers persist data in the host filesystem



* If we look at the above diagram, whenever running container wants to persist data, it actually put that data into the writable layer through storage driver.
* What are the problems:
* Data is no longer persisted and difficult to access if container stops as shown in the following diagram
* As we can see writable layer is tightly coupled with host filesystem and difficult to move the data.
* We have an extra layer of abstraction with a storage driver which reduces the performance.



* How Volumes can solve above issues
* Volumes are saved in the host filesystem (/var/lib/docker/volumes/) which is owned and maintained by docker.
* Any other nondocker process can’t access it.
* But, As depicted in the below other docker processes/containers can still access the data even container is stopped since it is isolated from the container file system.



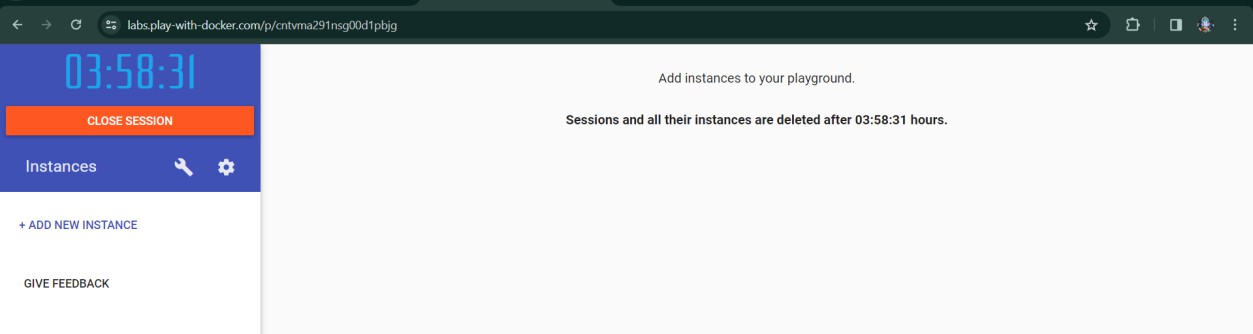
## Steps:

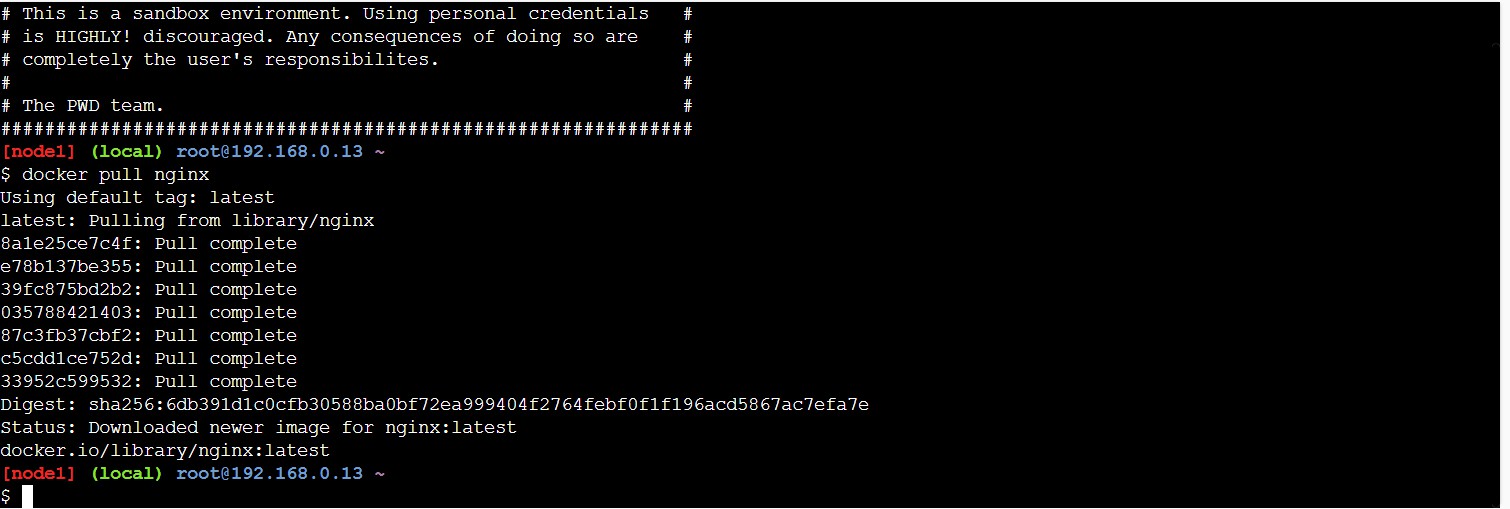
1. Go to website Play with Docker and login (Link: Play with Docker (play-with-docker.com))
2. Add new instance
3. Perform Following Inside Play-With-Docker
4. Pull nginx image into docker

## Command:

docker pull ngnix

## Output:



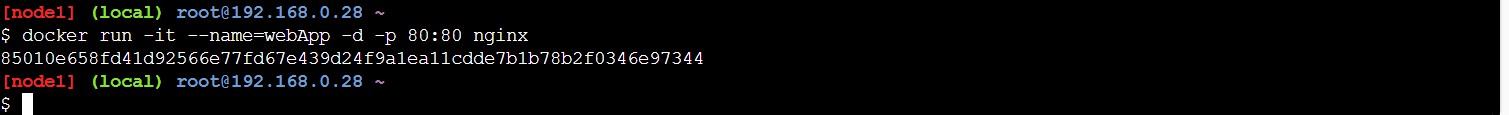
****

1. Now run the pulled image in Conatiner named “webApp”

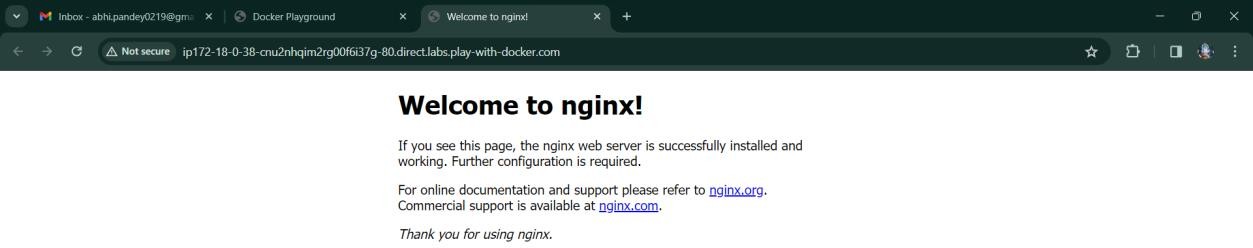
## Command:

docker run -it --name=webApp -d -p 80:80 nginx

## Output:

****

1. Click on port 80 to check output (it shows welcome page)



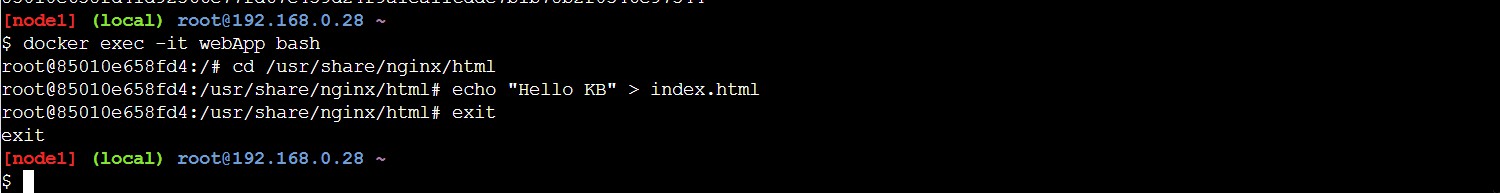
1. We make changes into “index.html” file inside /usr/share/nginx/html folder

## Command:

docker exec -it webApp bash cd /usr/share/nginx/html

echo “Hello KB” > index.html exit

## Output:

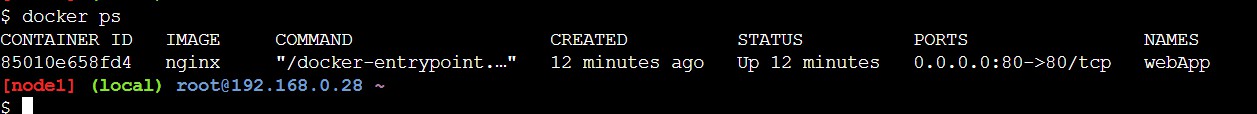
****

1. Type docker prompt and check process status using ps option

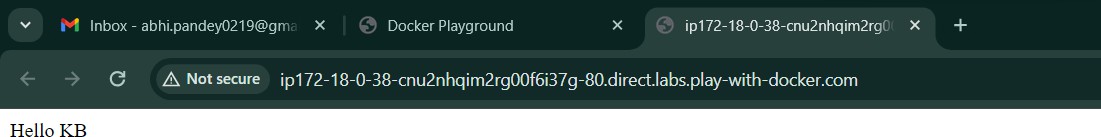
## Command:

docker ps

## Output:

****

1. Open the port 80



1. Now stop running container named “webApp”.

## Command:

docker stop webApp

## Output:



1. Create new instance and Start nginx in new container named as “webApp1”.

## Command:

docker run -it --name=webApp1 -d -p 80:80 nginx

## Output:

****

1. Click on port 80 to check output (it shows welcome page)



1. Go back to node1 and create new volume

## Command:

docker volume create MyVolume docker volume ls

docker volume inspect MyVolume

## Output:

****

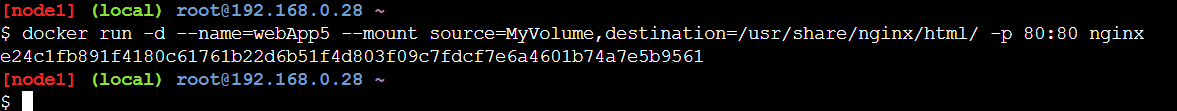
1. Mount this volume to nginx new container named “webApp5”

## Command:

docker run -d –name=webApp5 –mount source=MyVolume,destination=

/usr/share/nginx/html -p 80:80 nginx

## Output:

****

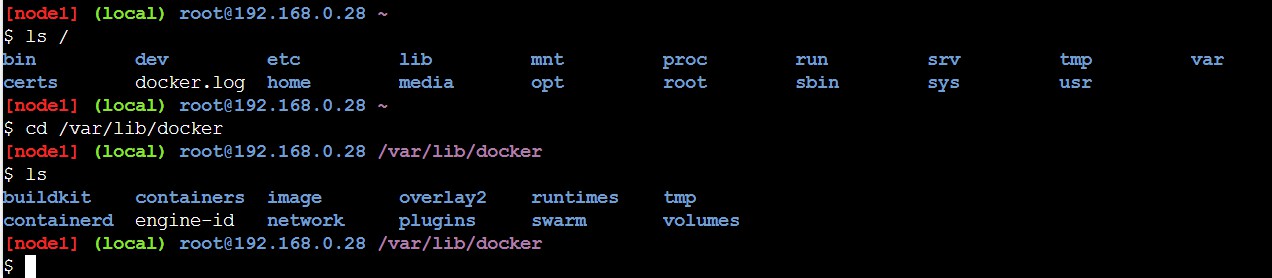
1. Now keep on doing “ls” and “cd “ to go inside \_data folder of our volume “MyVolume”

## Command:

ls /

cd /var/lib/docker ls

## Output:

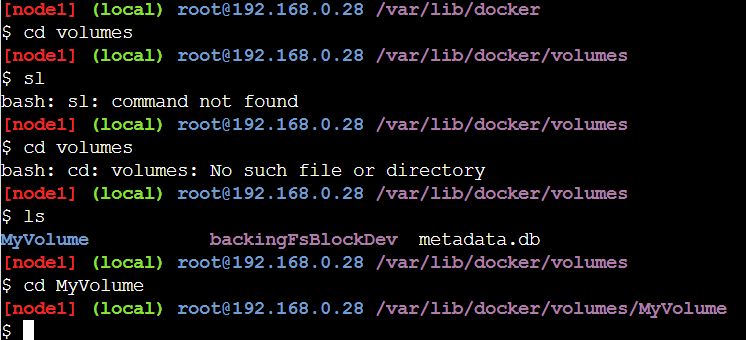
****

**Command:**

cd volumes ls

cd MyVolume ls

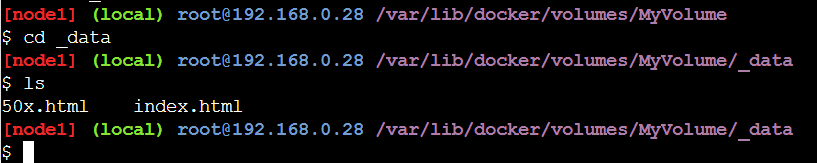
## Output:

****

**Command:**

cd \_data ls

## Output:

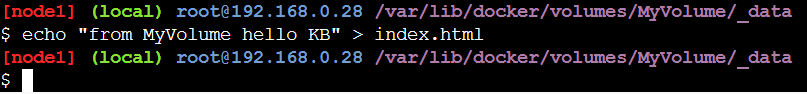


1. Modify contents of index.html file with “from MyVolume hello KB”

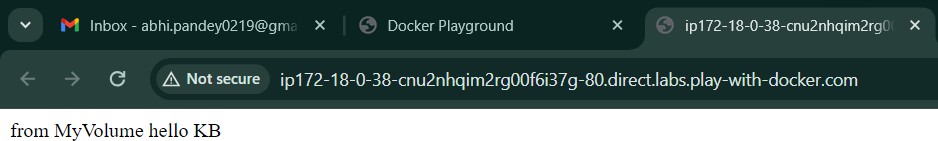
## Command:

echo “from MyVolume hello KB” > index.html

## Output:

****

1. Open the port 80 (to get the modified output)

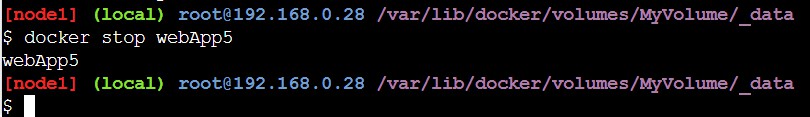


1. Now stop this running container named “webApp5”

## Command:

docker stop webApp5

## Output:

****

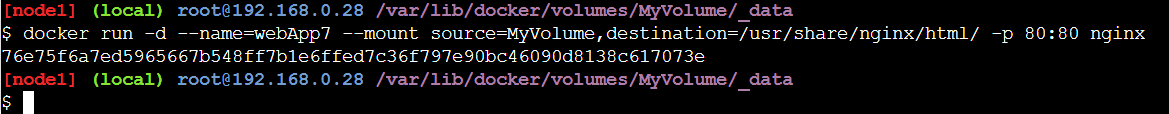
1. Now run nginx in new container named “webApp7”

## Command:

docker run -d –name=webApp7 --mount source=MyVolume,destination=

/usr/share/nginx/html -p 80:80 nginx

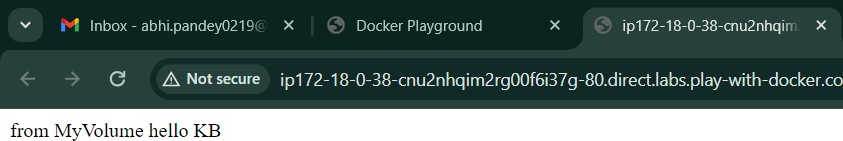
## Output:

****

1. Click on port 80 and refresh the page you should get edited file as output.

We can load the page again localhost:80 and still see the html file that we edited in the volume.

So, with the help of volumes, we can easily access the data even we stop the container and it’s very easy to access data and import the data to anywhere.



## PRACTICAL NO 6 – WORKING WITH DOCKER SWARM

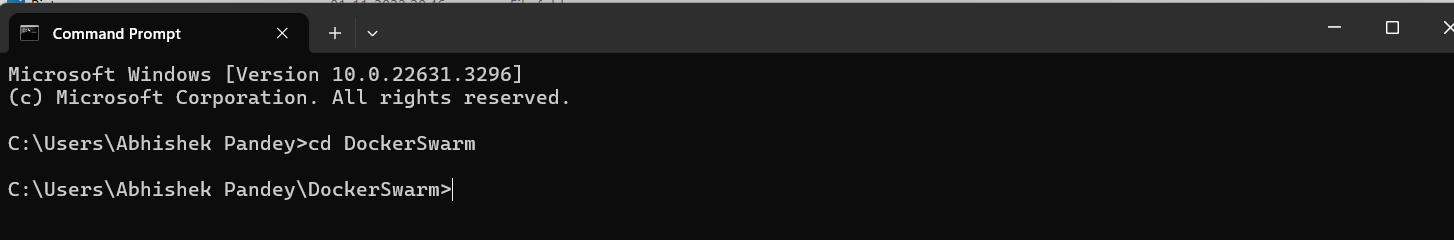
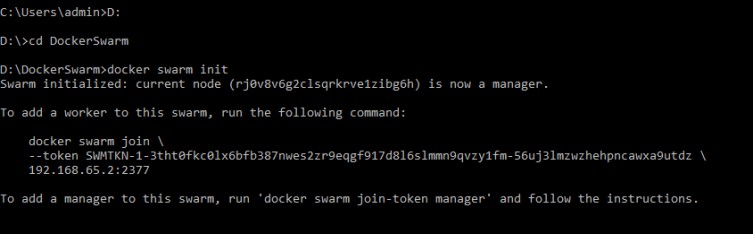
(P.S. This practical is implemented on a single node which acts both as manager and worker)

## Steps:

1. Login the Docker with github
2. Create a Folder in local drive with name “**DockerSwarm”**
3. Open the cmd and enter the commands

## Commands:

cd DockerSwarm docker swarm init



1. Open a folder Docker Swarm in D drive add “**docker-compose.yml”** file in it and add the following contents in it

## Code:

version : '3'

services : database: image : redis

deploy :

replicas : 2 web:

image : nginx deploy :

replicas : 2 ports :

- 8500:80

visualizer:

image : dockersamples/visualizer deploy:

placement:

constraints: [node.role == manager] ports :

- 8080:8080

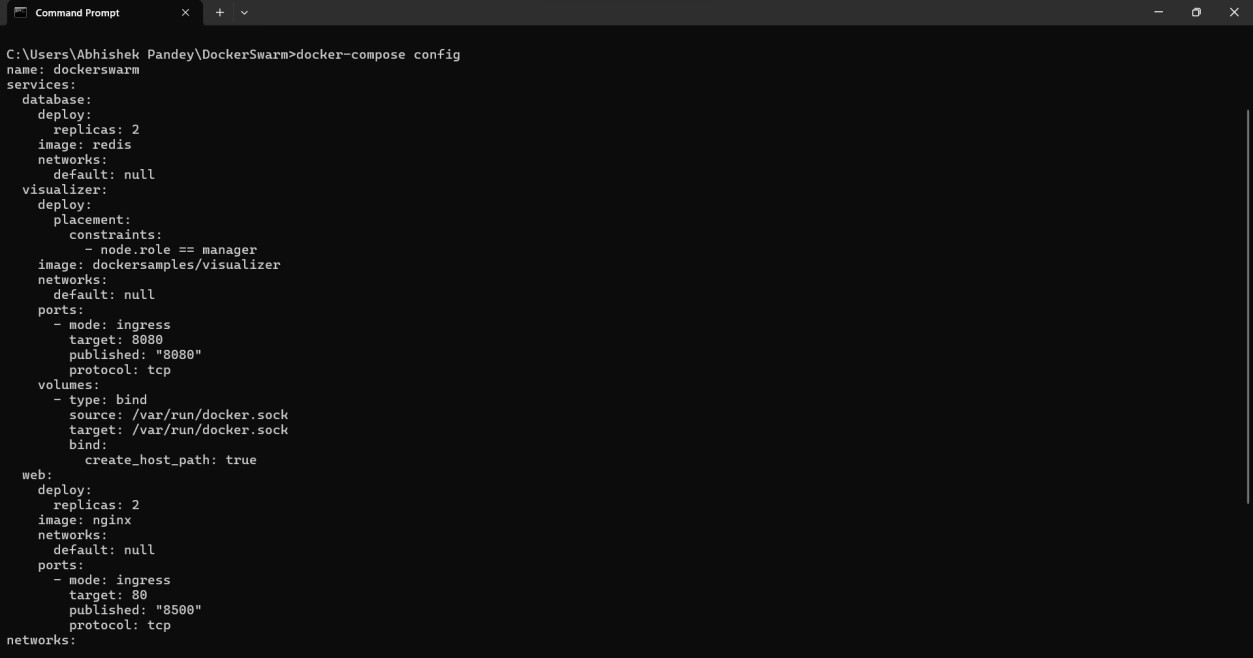
volumes :

- /var/run/docker.sock:/var/run/docker.sock

## Command:

docker-compose config

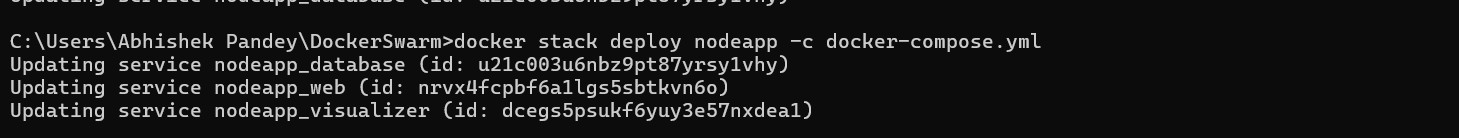
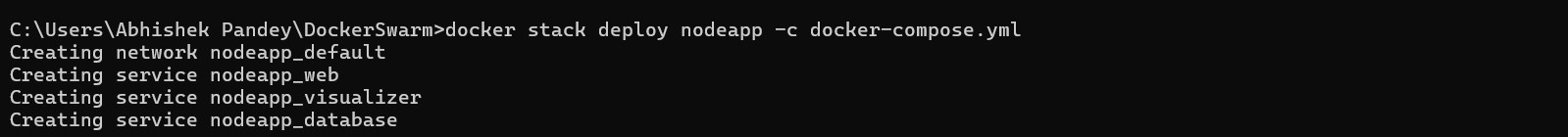
## Output:



**Command:**

docker stack deploy nodeapp -c docker-compose.yml

## Output:

****

1. Go to browser and check the “**localhost:8500”** and “**localhost:8080”**





## Command:

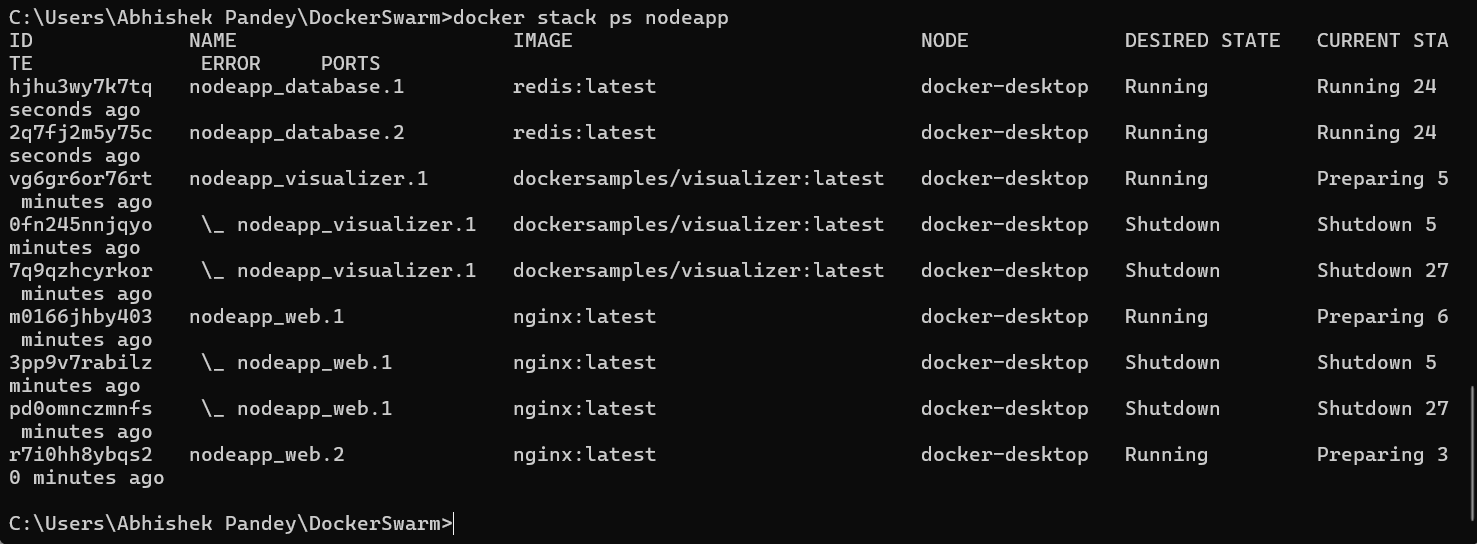
docker stack ls

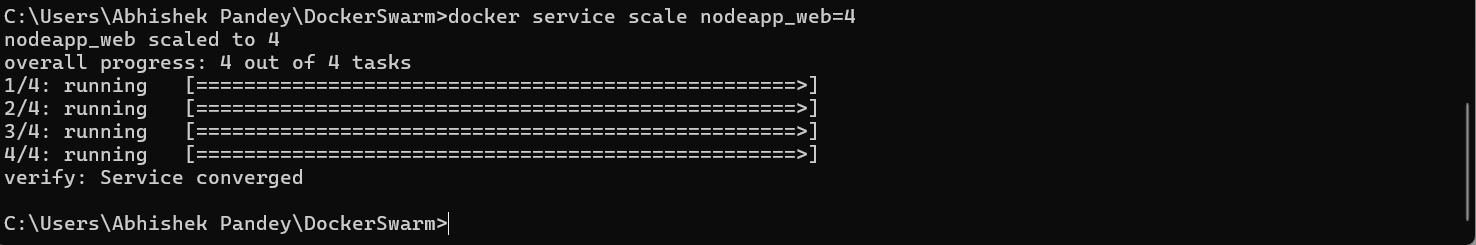
docker stack services nodeapp docker stack ps nodeapp

docker service scale nodeapp\_web=4

## Output:

****



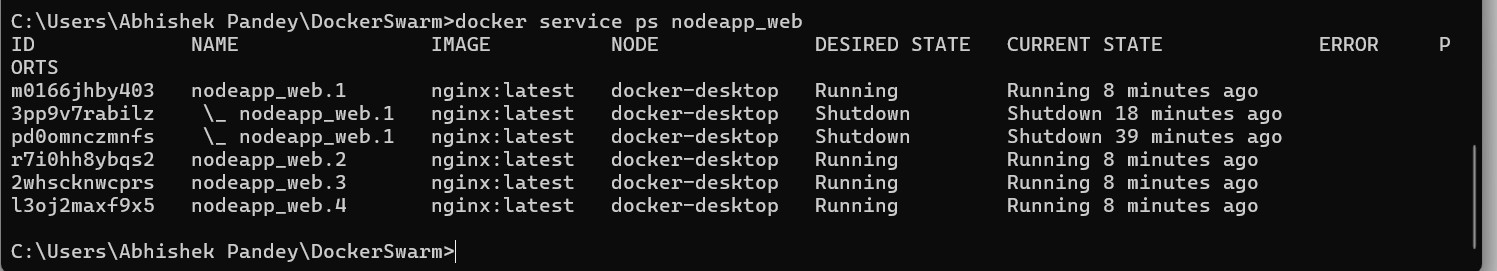
****

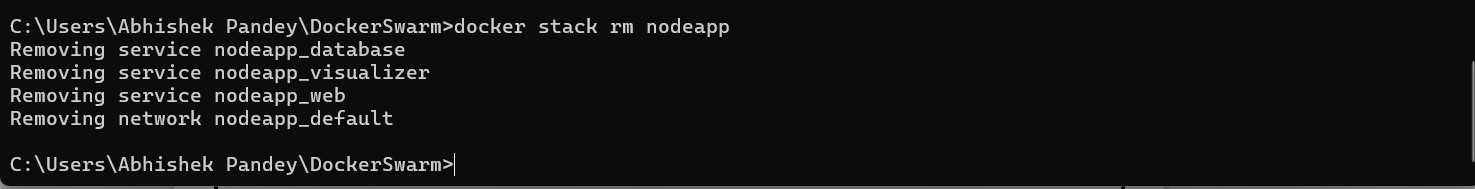
**Command:**

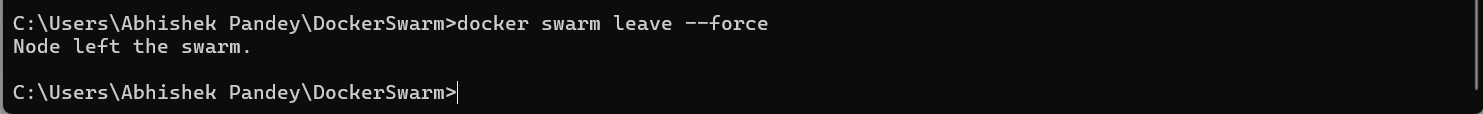
docker stack ls docker service ls

docker service ps nodeapp\_web docker stack rm nodeapp docker swarm leave --force

**Output:**

****



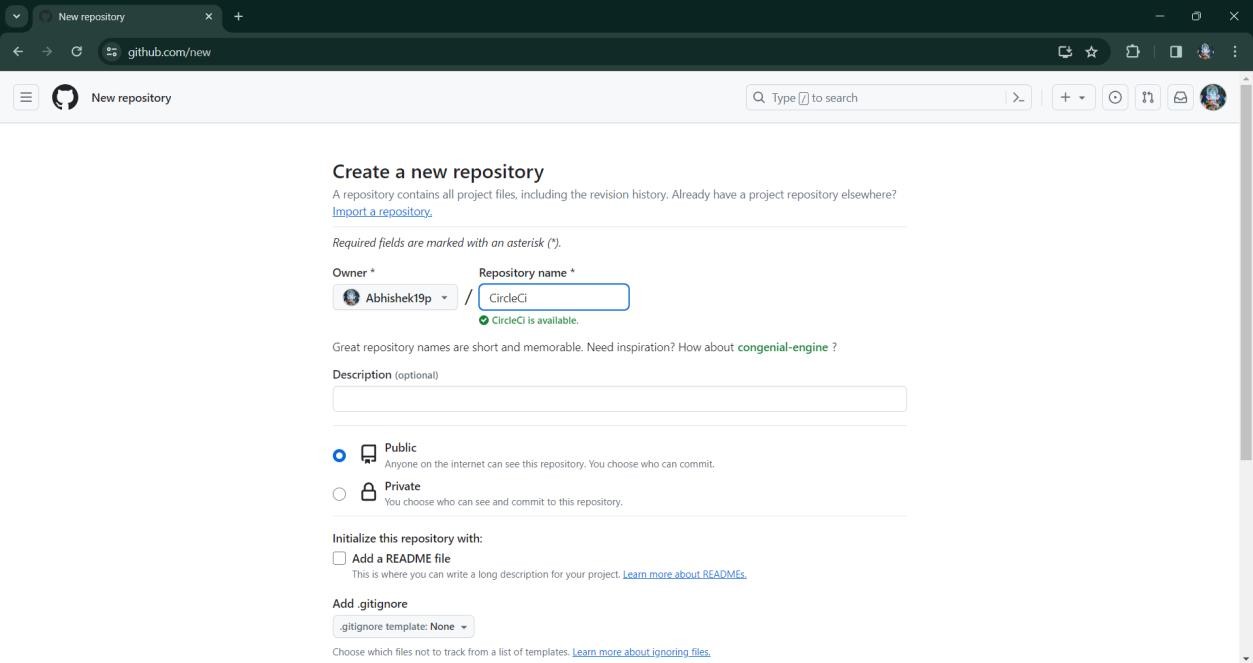
****

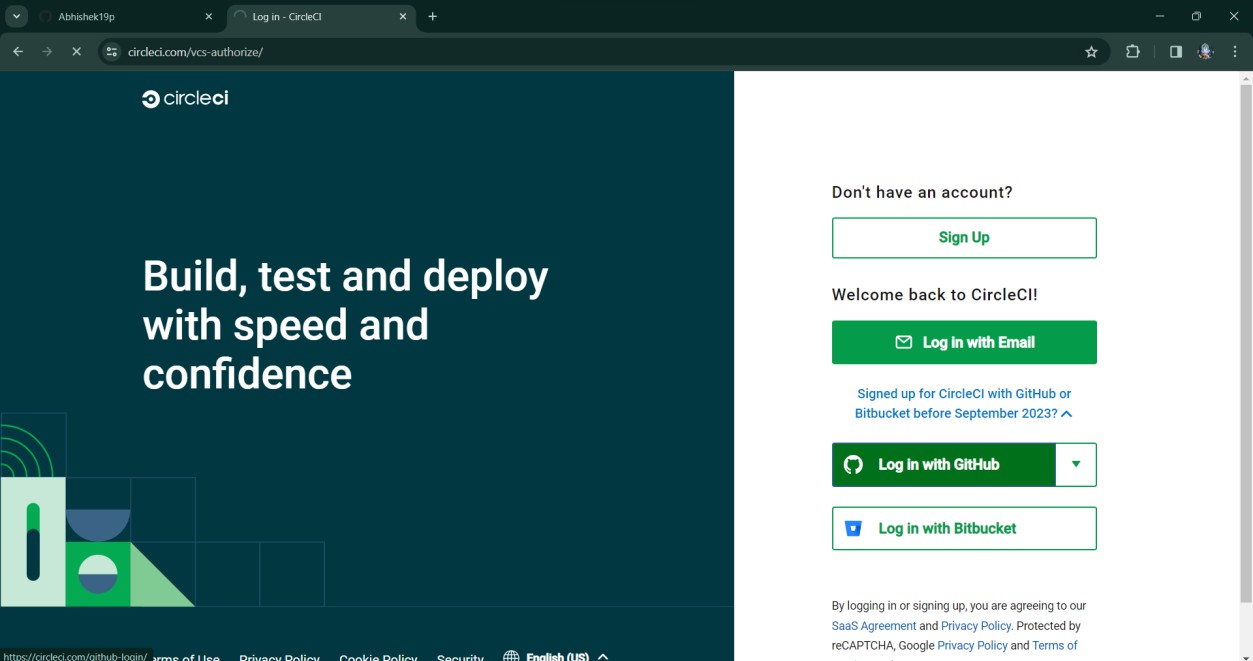
**PRACTICAL NO 7 – WORKING WITH CIRCLECI**

**Steps:**

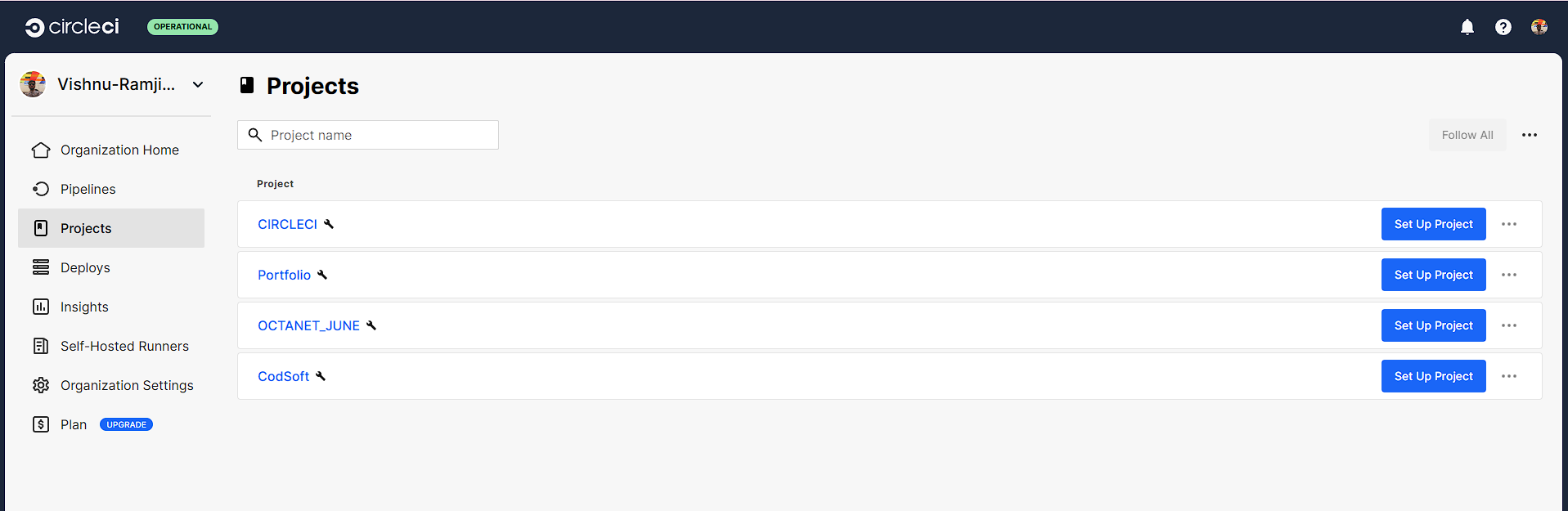
1. Create an account in Github or Login in.



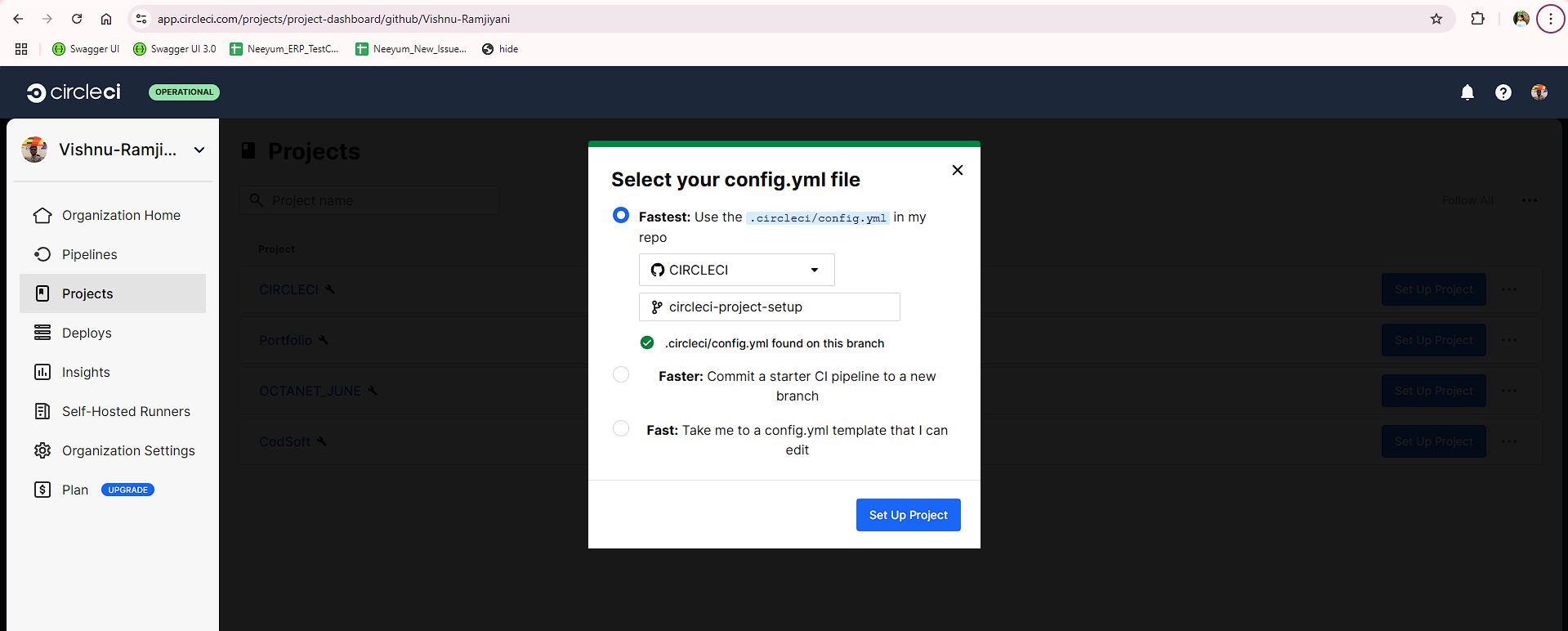
1. Navigate to GitHub and create a new repository. Input the name of your repository and finally, click New repository
2. Create an account on CircleCI by navigating to the singup page and clicking on sign up with GitHub



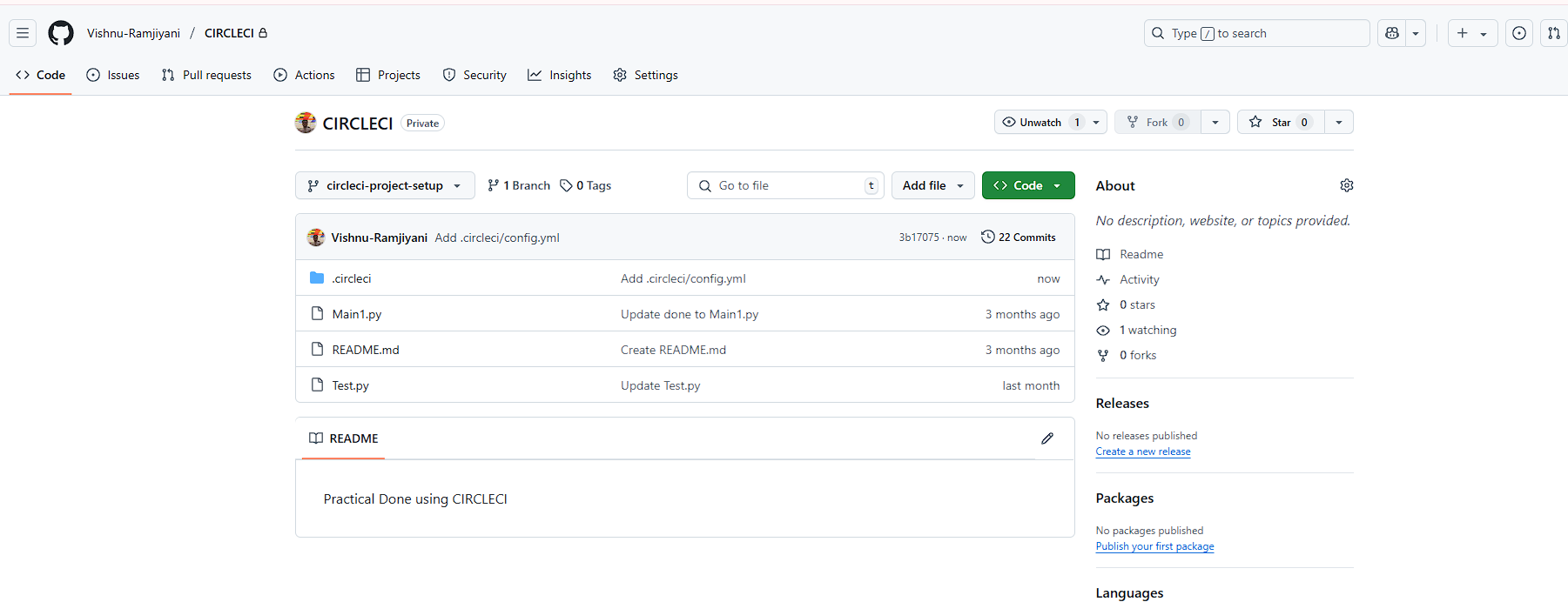
1. Authorize GitHub with CircleCI.
2. Set up the Project “CircleCi”

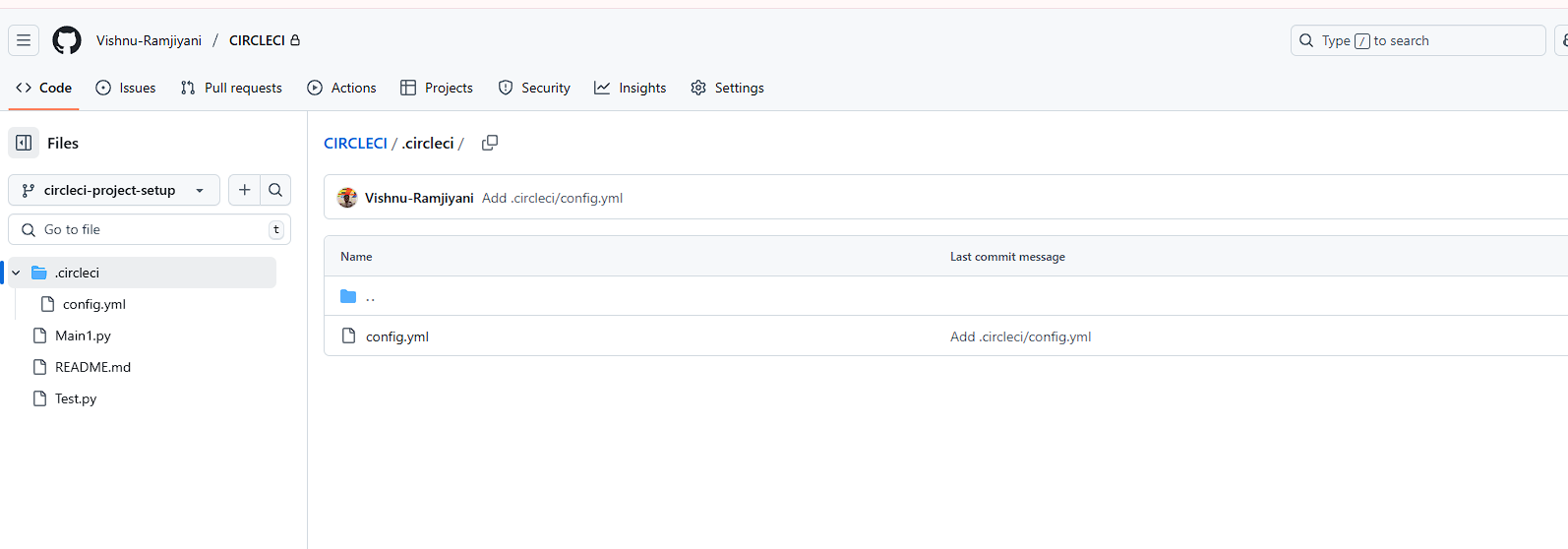


1. Select the Faster and set up project.



Go to GitHub. Click on the Repositories and refresh it and it already created a “.circleci”





Add two python files in “CircleCi” repository “**Main1.py**” – filename

## Code:

def Add(a,b):

return a+b def Display(): print("Hello")

if name ==' main ': Display()



“**Test.py**” - filename

## Code:

from Main1 import Add

def TestAdd():

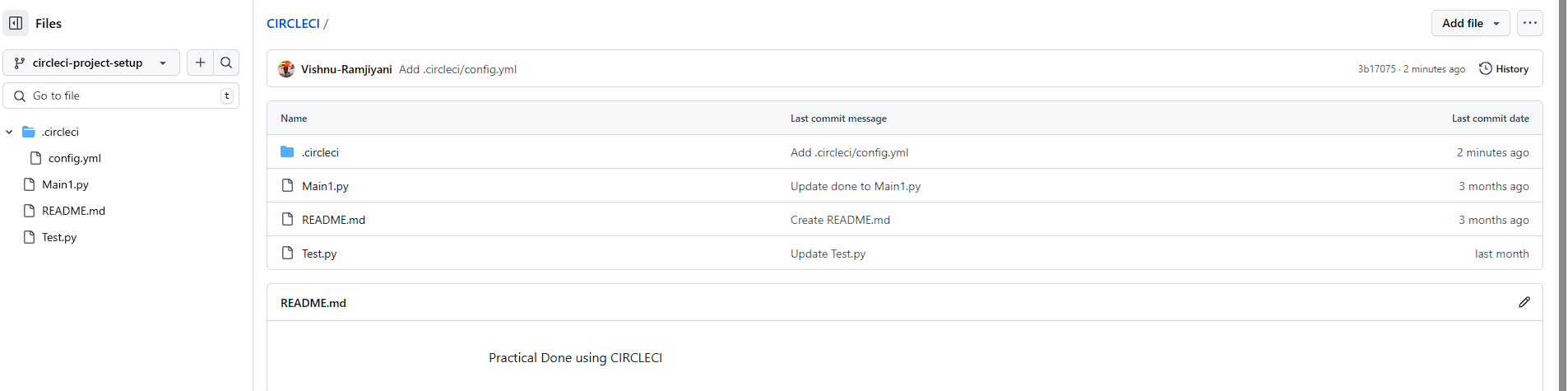
assert Add(3,4)==9

print("Add Funtion works correctly")

if name ==' main ':

TestAdd()

1. Upload the both file in “CircleCi” repositories (Main1.py and Test.py) and commit changes.



1. Write the code as shown below in the config.yml file in the GitHub repositories in CircleCI.

## Code:

version: 2.1

#Define the jobs we want to run for this projectjobs: jobs:

build:

docker:

* image: python steps:
* checkout
* run: python Main1.py test:

docker:

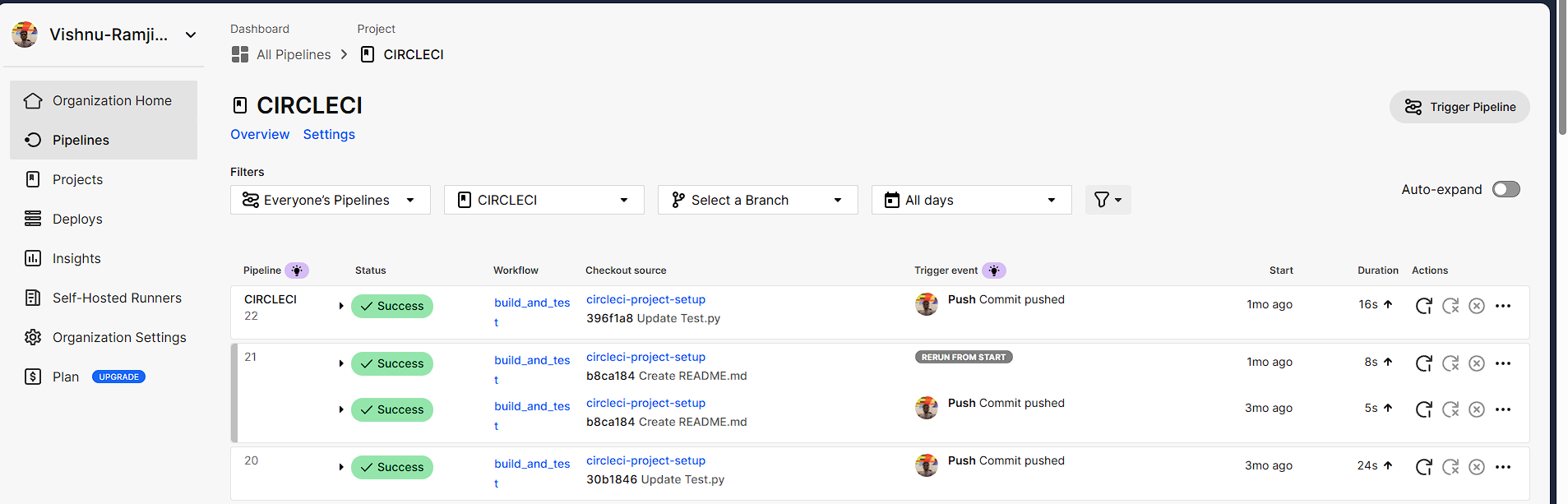
* image: python steps:
* checkout
* run: python Test.py

# Orchestrate our job run sequence workflows:

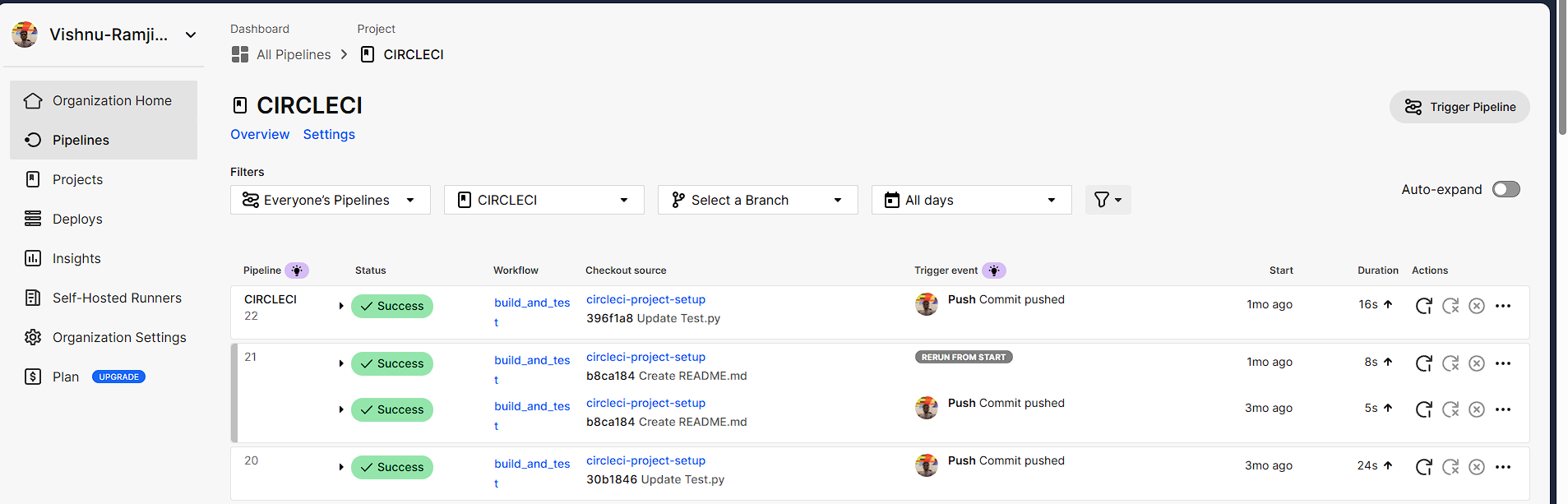
build\_and\_test: jobs:

* build
* test

1. After making all changes. Check the CircleCi website



1. This time we have done (3,4)==9 It must show failed.
2. Change the code in “**Test.py**” (3,4)==7 and commit the changes.
3. Go to CircleCi Website and refresh it and it will show successfully done.



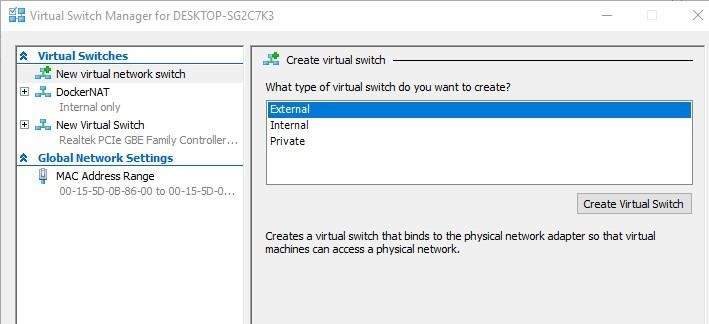
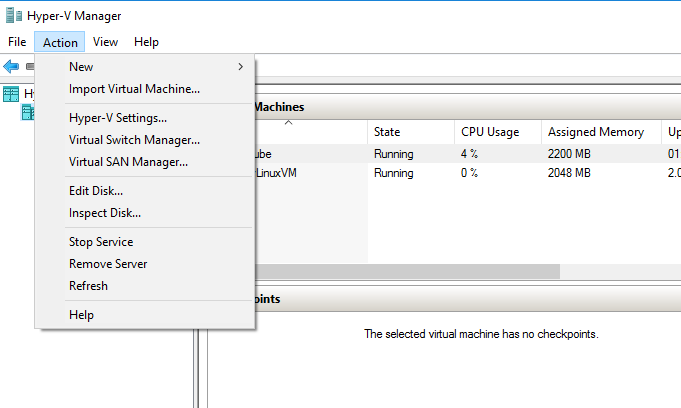
## PRACTICAL NO 8 – WORKING WITH KUBERNETES

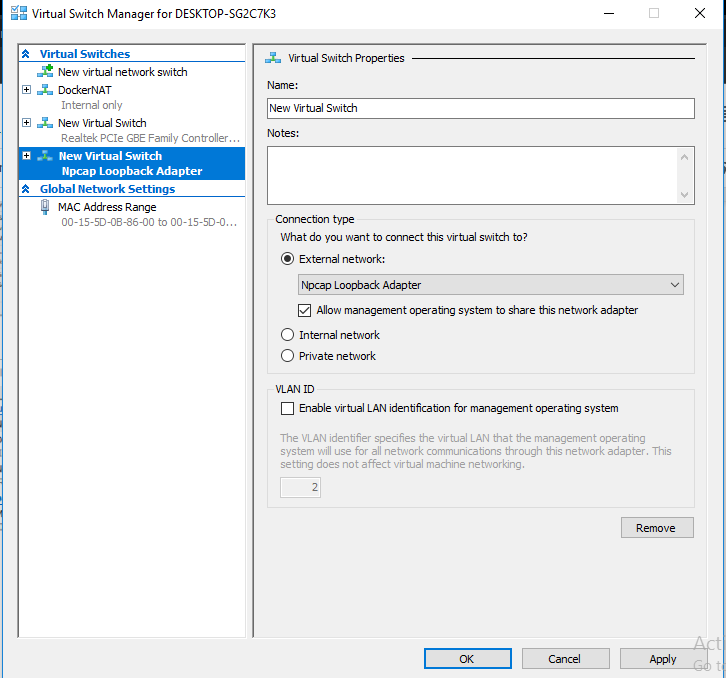
Steps:

Download kubectl and minikube

Refer the following link for the complete installation process:- <https://iteritory.com/install-minikube-in-windows-10-laptop-step-by-step-tutorial/>

Enter the path in environment variable

Open Hyper-V Manager



5) Make the changes in kubectl-edit-iofgz.yaml

//Nginx.yml file

apiVersion: apps/v1 kind: Deployment metadata :

name : nginx- deployment labels :

app :

nginx spec : replicas : 1 selector :

matchLabels : app : nginx

template :

metadata :

labels :

app :

nginx spec :

containers :

* name : nginx image :

nginx:1.16 ports :

* containerPort : 80

Open Powershell run it as administrator and type the following commands.

minikube start --driver=hyperv

minikube status

minikube stop

minikube start

kubectl get nodes

kubectl get pods

kubectl get services

kubectl create deployement nginxdep --image=nginx

kubectl create deployement

kubectl get deployement

kubectl get pod

kubectl get deployement

kubectl get replicaset

kubectl edit deployment ngnixdep

kubectl get pod

kubectl get pod

kubectl get replicaset

kubectl log nginxdep(the value generated in pod)

kubectl create deployment mongodep --image=mongo

kubectl get deployement

kubectl get pod

kubectl logs mongodep(pod value)

kubectl get deployement

kubectl get pod

kubectl describe pod mongodep(pod value)

kubectl get pod

kubectl exec -it mongodep-9ccfdff7-4n8j4 -- /bin/bash

once the terminal opens

ls

df -h

exit

kubectl delete deployment mongodep

kubectl get deployment

kubectl get pod

kubectl get replicaset

kubectl delete deployment nginxdep

kubectl get deployment

kubectl get pod

kubectl get replicaset

to go into drive d -> D:

kubectl

minikube

kubectl get deployment

kubectl get deployment

kubectl get deployment

kubectl get pod

kubectl get replicaset

kubectl get replicaset

kubectl get replicaset

kubectl get all

kubectl delete deployment nginxdep

kubectl get all

kubectl apply -f nginx\_depl\_config.yml

