

MICROSERVICES

ARCHITECTURE

A PRACTICAL REPORT
ON
MICROSERVICES ARCHITECTURE

SUBMITTED BY
Mr. RAHUL KEWAT
Roll No: 22006

UNDER THE GUIDANCE OF
PROF. MEHDI REZAEI

Submitted in fulfillment of the requirements for qualifying
MSc. IT Part I Semester - II Examination 2022-2023

University of Mumbai
Department of Information Technology

R.D. & S.H National College of Arts, Commerce &
S.W.A. Science College Bandra (West), Mumbai – 400 050



R. D. & S. H. National & S. W. A. Science College

Bandra (W), Mumbai – 400050.

**Department of Information Technology
M.Sc. (IT – SEMESTER II)**

Certificate

This is to certify that Microservices Architecture Practicals performed at R.D & S.H National & S.W.A. Science College by Mr. Rahul Kewat holding Seat No. _____ studying Master of Science in Information Technology Semester – II has been satisfactorily completed as prescribed by the University of Mumbai, during the year 2022– 2023.

Subject In-Charge Coordinator In-Charge External Examiner

College Stamp

INDEX

Sr. No	Date	Practical	Page	Sign
1	4/3/23	Building ASP.NET Core MVC Application	1-7	
2	11/3/23	Building ASP.NET Core REST API	8-16	
3	18/3/23	Working with Docker, Docker Commands, Docker Images and Containers	17-25	
4	25/3/23	Installing software packages on Docker, Working with Docker Volumes and Networks	26-29	
5	1/4/23	Working with Circle CI for continuous integration	30-42	
6	8/4/23	Creating Microservice with ASP.NET Core	43-55	
7	29/4/23	Creating Backing Service with ASP.NET Core	56-62	
8	20/5/23	Building real-time Microservice with ASP.NET Core	63-74	

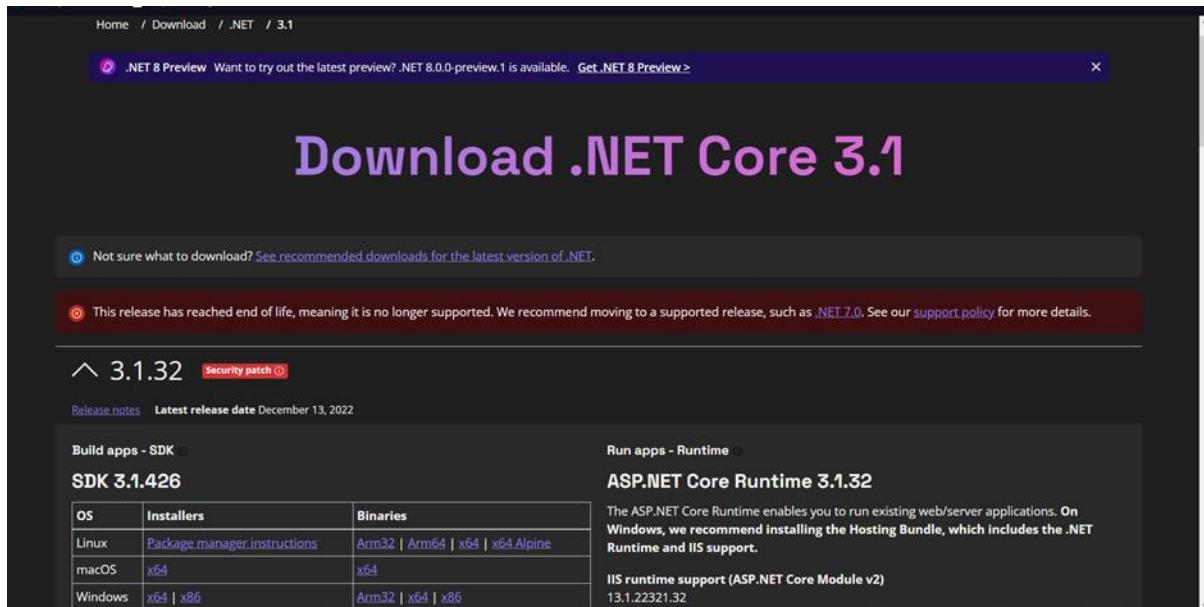
Practical 1

Aim: Creating a basic MVC application using .NET CORE

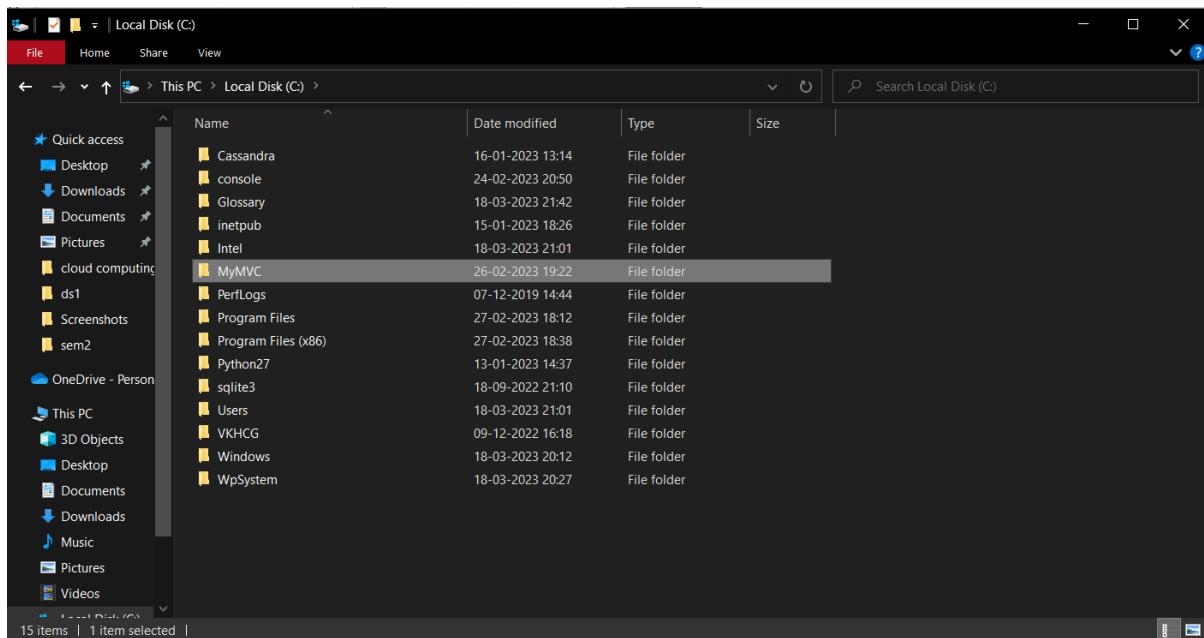
Writeup:

Step 1:

- Install .Net Core SDK.

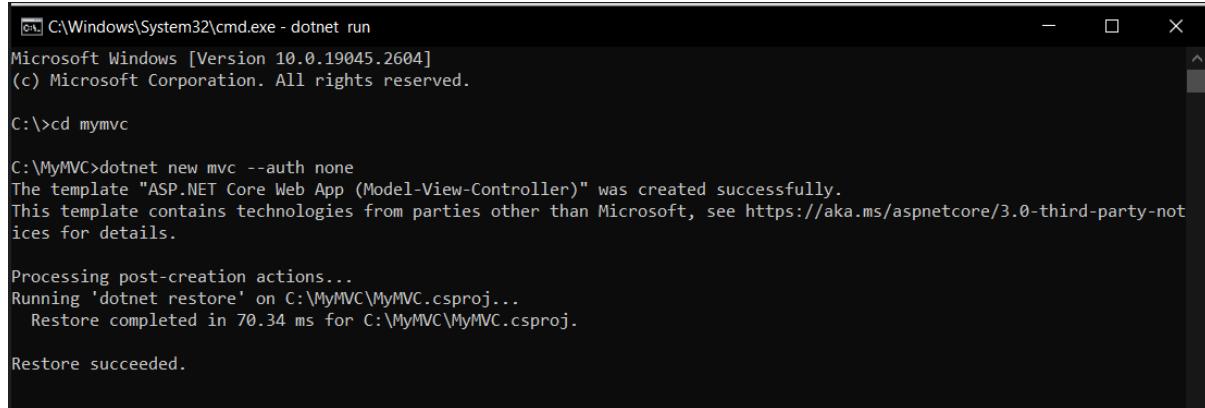
**Step 2:**

- Create a Folder MyMVC in C: drive.



Step 3:

- Open Command Prompt and type the following commands:
dotnet new mvc --auth none



```
C:\Windows\System32\cmd.exe - dotnet run
Microsoft Windows [Version 10.0.19045.2604]
(c) Microsoft Corporation. All rights reserved.

C:\>cd mymvc

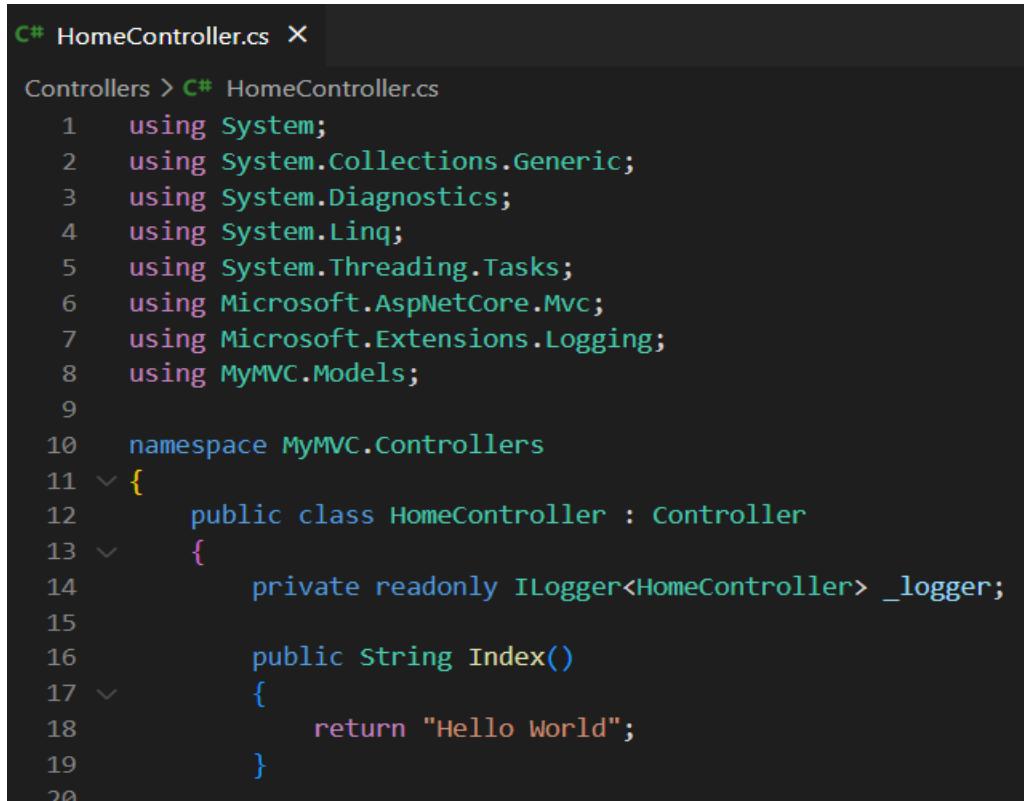
C:\MyMVC>dotnet new mvc --auth none
The template "ASP.NET Core Web App (Model-View-Controller)" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/aspnetcore/3.0-third-party-notices for details.

Processing post-creation actions...
Running 'dotnet restore' on C:\MyMVC\MyMVC.csproj...
  Restore completed in 70.34 ms for C:\MyMVC\MyMVC.csproj.

Restore succeeded.
```

Step 4:

- Go to the controllers folder and modify HomeController.cs file to match the following code:



```
C# HomeController.cs X

Controllers > C# HomeController.cs

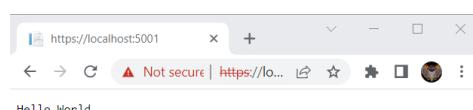
1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Linq;
5  using System.Threading.Tasks;
6  using Microsoft.AspNetCore.Mvc;
7  using Microsoft.Extensions.Logging;
8  using MyMVC.Models;
9
10 namespace MyMVC.Controllers
11 {
12     public class HomeController : Controller
13     {
14         private readonly ILogger<HomeController> _logger;
15
16         public string Index()
17         {
18             return "Hello World";
19         }
20     }
}
```

Step 5:

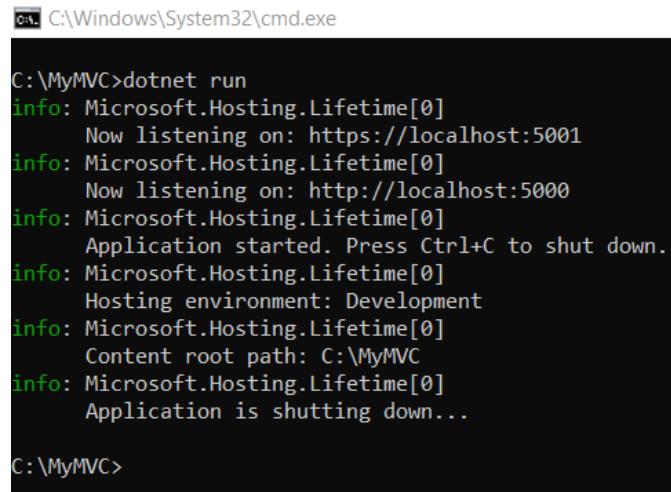
- Run the code

```
C:\MyMVC>dotnet run
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\MyMVC
```

- Paste the localhost:5001 link in your browser(In the case we are using Chrome).

**Step 6:**

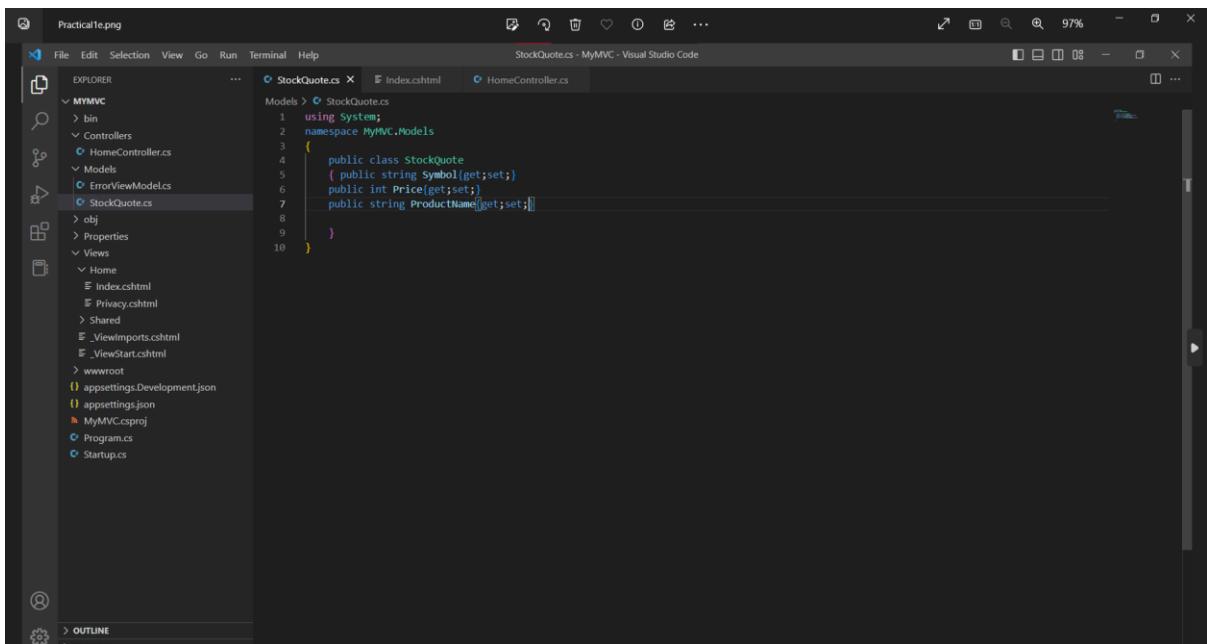
- Now go back to command prompt and stop running project using CTRL+C.



```
C:\Windows\System32\cmd.exe
C:\MyMVC>dotnet run
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\MyMVC
info: Microsoft.Hosting.Lifetime[0]
      Application is shutting down...
C:\MyMVC>
```

Step 7:

- Go to models folder and add new file StockQuote.cs to it with following content:

**Step 8:**

- Now Add View to folder then home folder in it and modify index.cshtml file to match following

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Practical14.png, File, Edit, Selection, View, Go, Run, Terminal, Help, Index.cshtml - MyMVC - Visual Studio Code.
- Explorer:** Shows the project structure for 'MyMVC' with files like StockQuote.cs, HomeController.cs, Index.cshtml, Privacy.cshtml, etc.
- Editor:** Displays the content of Index.cshtml:


```

1 @{
2     ViewData["Title"] = "Home Page";
3 }
4
5 <div class="text-center">
6     <h1 class="display-4">Welcome Rahul Kewat</h1>
7     Symbol: @Model.Symbol <br/>
8     Price: @Model.Price<br/>
9     Product name: @Model.ProductName<br/>
10    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building web apps with ASP.NET Core</a>.</p>
11 </div>
12
      
```
- Bottom:** OUTLINE and TRACE LOG sections.

Step 9:

- Now modify HomeController.cs file to match following:

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Practical14.png, File, Edit, Selection, View, Go, Run, Terminal, Help, HomeController.cs - MyMVC - Visual Studio Code.
- Explorer:** Shows the project structure for 'MyMVC' with files like StockQuote.cs, HomeController.cs, Index.cshtml, Privacy.cshtml, etc.
- Editor:** Displays the content of HomeController.cs:


```

1 using System.Collections.Generic;
2 using System.Diagnostics;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using Microsoft.AspNetCore.Mvc;
6 using Microsoft.Extensions.Logging;
7 using MyMVC.Models;
8
9 namespace MyMVC.Controllers
10 {
11     public class HomeController : Controller
12     {
13         public IActionResult Index()
14         {
15             var model=new StockQuote{Symbol="INR",Price=58000,ProductName="Hp"};
16             return View(model);
17         }
18         private readonly ILogger<HomeController> _logger;
19
20         public HomeController(ILogger<HomeController> logger)
21         {
22             _logger = logger;
23         }
24
25         //public IActionResult Index()
26         //{
27         //    return View();
28         //}
29
30         public IActionResult Privacy()
31         {
32             return View();
33         }
34
35         [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
36         public IActionResult Error()
37         {
38             return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
39         }
40     }
41 }
      
```
- Bottom:** OUTLINE and TRACE LOG sections.

Step 10:

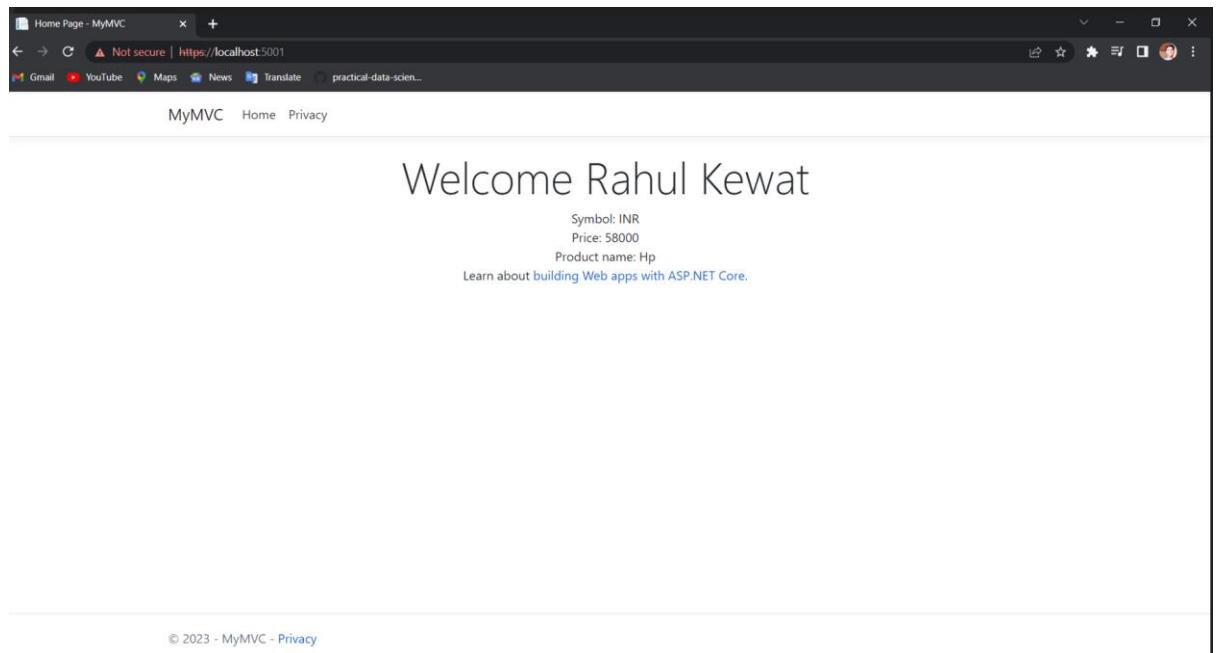
- Now run the project using **dotnet run**

```
C:\Windows\System32\cmd.exe - dotnet run

C:\MyMVC>dotnet run
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\MyMVC
```

Step 11:

- Now go back to browser and refresh to get modified view response.



Practical 2

Aim: Building an ASP.NET CORE REST API.

Writeup:

Step 1: Create your Web API.

Open two command prompts

CMD 1:

Code:

```
dotnet new webapi -o Glossary
```

```
C:\>dotnet new webapi -o Glossary --force
The template "ASP.NET Core Web API" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on Glossary\Glossary.csproj...
  Restore completed in 18.87 ms for C:\Glossary\Glossary.csproj.

Restore succeeded.
```

CMD 1:

```
cd Glossary
```

```
dotnet run
```

```
C:\Glossary>dotnet run
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Glossary
```

Step 2: In Command Prompt 2: (try running ready made weatherforecast class for testing)

```
curl --insecure https://localhost:5001/weatherforecast
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.2728]
(c) Microsoft Corporation. All rights reserved.

C:\Program Files>cd..

C:\>curl --insecure https://localhost:5001/weatherforecast
[{"date": "2023-03-30T18:39:50.870404+05:30", "temperatureC": 42, "temperatureF": 107, "summary": "Sweltering"}, {"date": "2023-03-31T18:39:50.8708603+05:30", "temperatureC": 20, "temperatureF": 67, "summary": "Chilly"}, {"date": "2023-04-01T18:39:50.8708752+05:30", "temperatureC": -15, "temperatureF": 6, "summary": "Bracing"}, {"date": "2023-04-02T18:39:50.8708761+05:30", "temperatureC": -9, "temperatureF": 16, "summary": "Scorching"}, {"date": "2023-04-03T18:39:50.8708766+05:30", "temperatureC": 16, "temperatureF": 60, "summary": "Warm"}]
```

Step 3: Now Change the content:-

To get started, remove the **WeatherForecast.cs** file from the root of the project and the **WeatherForecastController.cs** file from the Controllers folder. Add Following two files.

[A]: GlossaryItem.cs

```
C# GlossaryItem.cs X
C: > Users > Dell > OneDrive > Pictures > MA2 > C# GlossaryItem.cs
1 namespace Glossary
2 {
3     public class GlossaryItem
4     {
5         public string Term
6         {
7             get;
8             set;
9         }
10
11        public string Definition
12        {
13            get;
14            set;
15        }
16    }
17 }
18 }
```

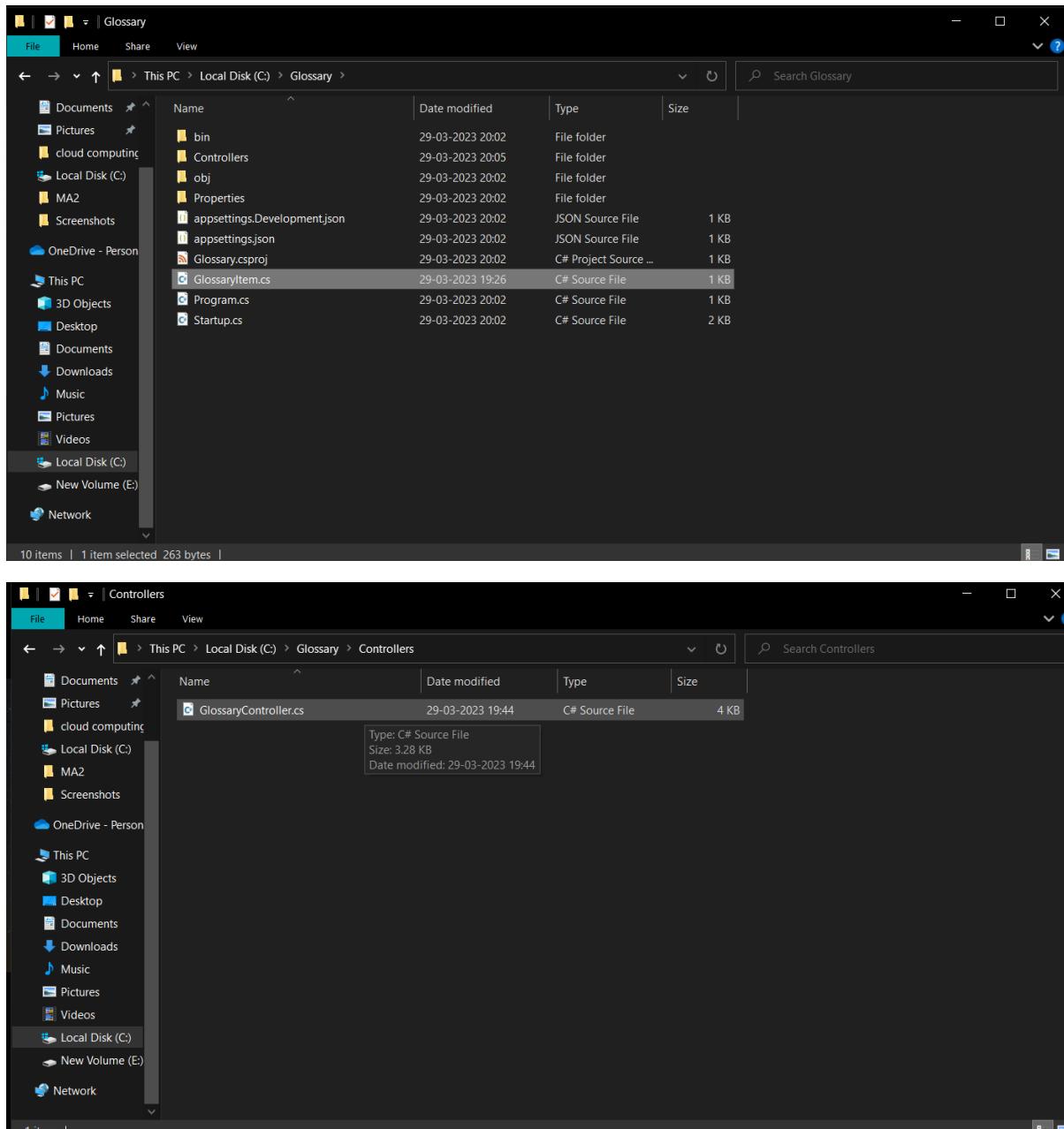
[B]: GlossaryController.cs

```
C: > Users > Dell > OneDrive > Pictures > MA2 > C# GlossaryController.cs
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Threading.Tasks;
 5  using Microsoft.AspNetCore.Mvc;
 6  using Microsoft.Extensions.Logging;
 7
 8  namespace Glossary.Controllers
 9  {
10      [ApiController]
11      [Route("api/[controller]")]
12
13      public class GlossaryController: ControllerBase
14      {
15          private static List<GlossaryItem> Glossary = new List<GlossaryItem>
16          {
17              new GlossaryItem
18              {
19                  Term= "HTML",
20                  Definition = "Hypertext Markup Language"
21              },
22
23              new GlossaryItem
24              {
25                  Term= "MVC",
26                  Definition = "Model View Controller"
27              },
28
29              new GlossaryItem
30              {
31                  Term= "OpenID",
32                  Definition = "An open standard for authentication"
33              }
34          };
35      }
```

```
C:\> Users > Dell > OneDrive > Pictures > MA2 > C# GlossaryController.cs
-- 36  [HttpGet]
37  public ActionResult<List<GlossaryItem>> Get()
38  {
39      return Ok(Glossary);
40  }
41
42  [HttpGet]
43  [Route("{term}")]
44  public ActionResult<GlossaryItem> Get(string term)
45  {
46      var glossaryItem = Glossary.Find(item =>
47          item.Term.Equals(term, StringComparison.InvariantCultureIgnoreCase));
48      if (glossaryItem == null)
49      {
50          return NotFound();
51      }
52
53      else
54      {
55          return Ok(glossaryItem);
56      }
57  }
58
59  [HttpPost]
60  public ActionResult Post(GlossaryItem glossaryItem)
61  {
62      var existingGlossaryItem = Glossary.Find(item =>
63          item.Term.Equals(glossaryItem.Term, StringComparison.InvariantCultureIgnoreCase));
64      if (existingGlossaryItem != null)
65      {
66          return Conflict("Cannot create the term because it already exists.");
67      }
68
69      else
```

```
C:\> Users > Dell > OneDrive > Pictures > MA2 > GlossaryController.cs
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
```

Output:



Step 4: Now stop running previous **dotnet run** on command prompt 1 using **Ctrl+C**. and Run it again for new code.

```
cmd: Command Prompt - dotnet run
```

```
C:\Glossary>dotnet run
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Glossary
```

Step 5: Run commands to perform certain operations on the **GlossaryItem** dataset.

[A]: Getting a list of items.

curl --insecure <https://localhost:5001/api/glossary>

```
C:\Glossary>curl --insecure https://localhost:5001/api/glossary
[{"term": "HTML", "definition": "Hypertext Markup Language"}, {"term": "MVC", "definition": "Model View Controller"}, {"term": "OpenID", "definition": "An open standard for authentication"}]
```

[B]: Getting a single item.

curl --insecure <https://localhost:5001/api/glossary/MVC>

```
C:\Glossary>curl --insecure https://localhost:5001/api/glossary/MVC
{"term": "MVC", "definition": "Model View Controller"}
```

[C]: Creating an item.

curl --insecure -X POST -d "{\"term\": \"MFA\", \"definition\": \"An authentication process.\"} -H \"ContentType:application/json\" <https://localhost:5001/api/glossary>

```
C:\Glossary>curl --insecure -X POST -d "{\"term\": \"MFA\", \"definition\": \"An authentication process.\"} -H \"Content-Type:application/json\" https://localhost:5001/api/glossary
{"term": "MFA", "definition": "An authentication process."}
```

[D]: Update an item.

curl --insecure -X PUT -d "{\"term\": \"MVC\", \"definition\": \"Modified record of Model View Controller.\"} -H \"Content-Type:application/json\" <https://localhost:5001/api/glossary>

```
C:\Glossary>curl --insecure -X PUT -d "{\"term\": \"MVC\", \"definition\":\"Modified record of Model View Controller.\"}" -H "Content-Type:application/json" https://localhost:5001/api/glossary
C:\Glossary>curl --insecure https://localhost:5001/api/glossary/MVC
{"term":"MVC","definition":"Modified record of Model View Controller."}
```

[E]: Delete an item.

curl --insecure --request DELETE --url <https://localhost:5001/api/glossary/openid>

```
C:\Glossary>curl --insecure --request DELETE --url https://localhost:5001/api/glossary/openid
C:\Glossary>curl --insecure https://localhost:5001/api/glossary
[{"term":"HTML","definition":"Hypertext Markup Language"}, {"term":"MVC","definition":"Modified record of Model View Controller."}, {"term":"MFA","definition":"An authentication process."}]
```

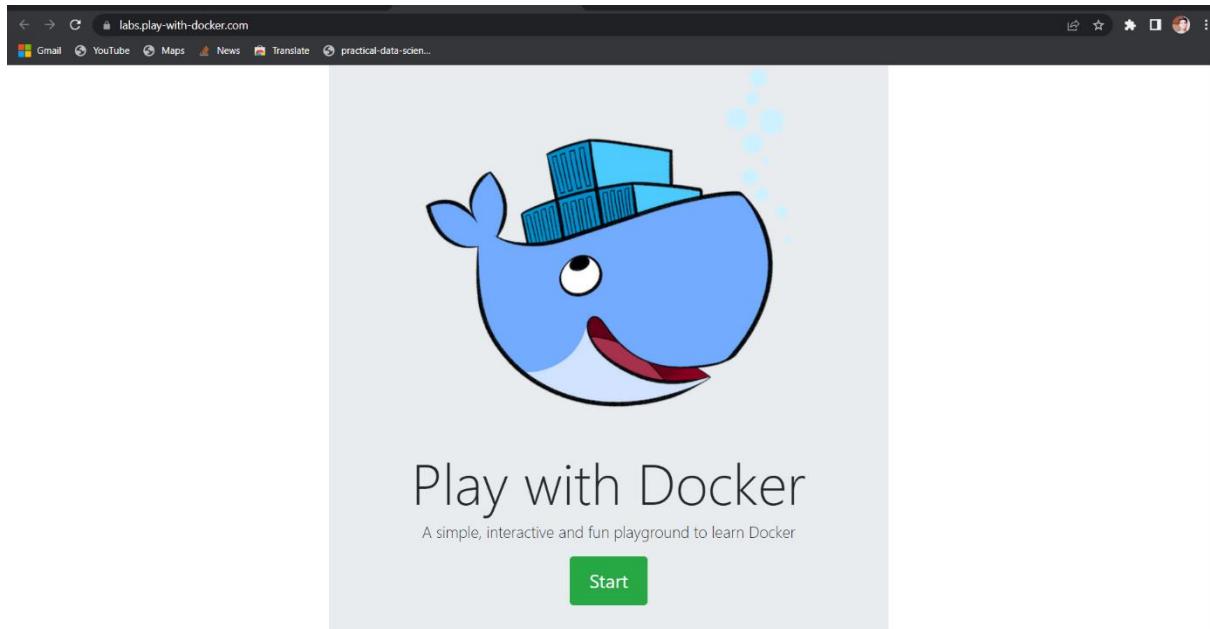
Practical 3

Aim: Working with Docker

Writeup:

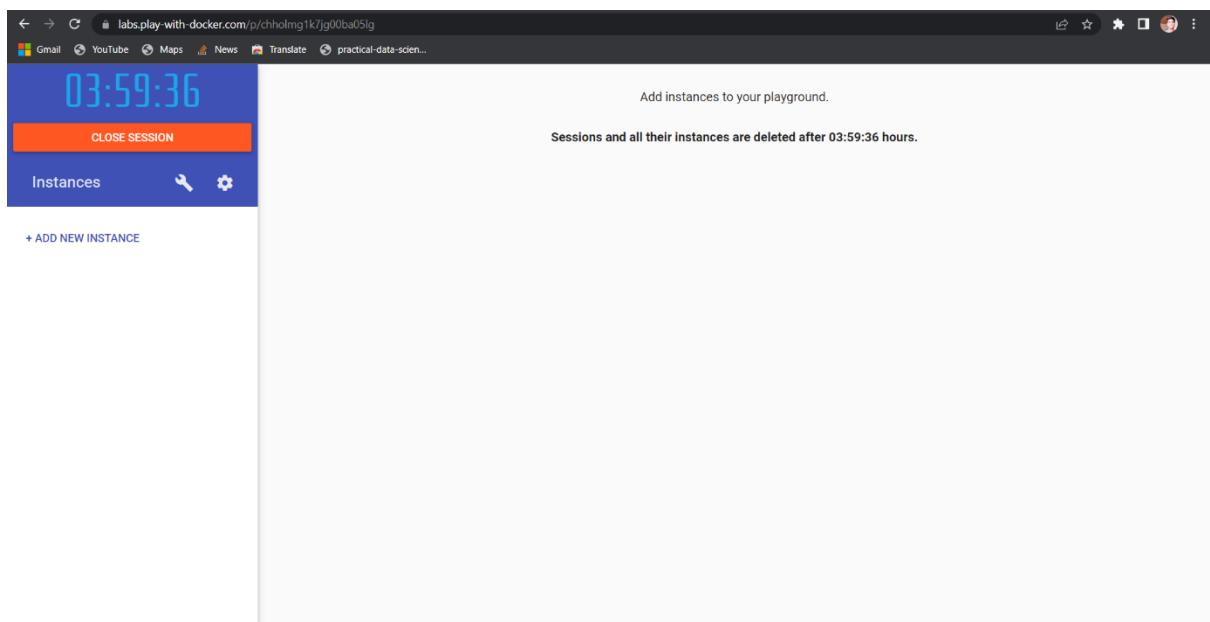
Step 1: create Docker Hub account (sign up)

Step 2: login to <https://labs.play-with-docker.com/>



Click on start

Step 3: add new instance



Step 4: perform following:

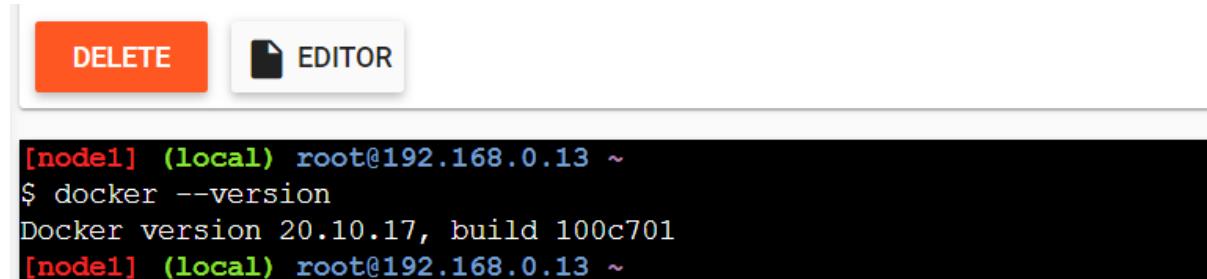
Method1:

To pull and push images using docker

Command: to check docker version

docker –version

Output:

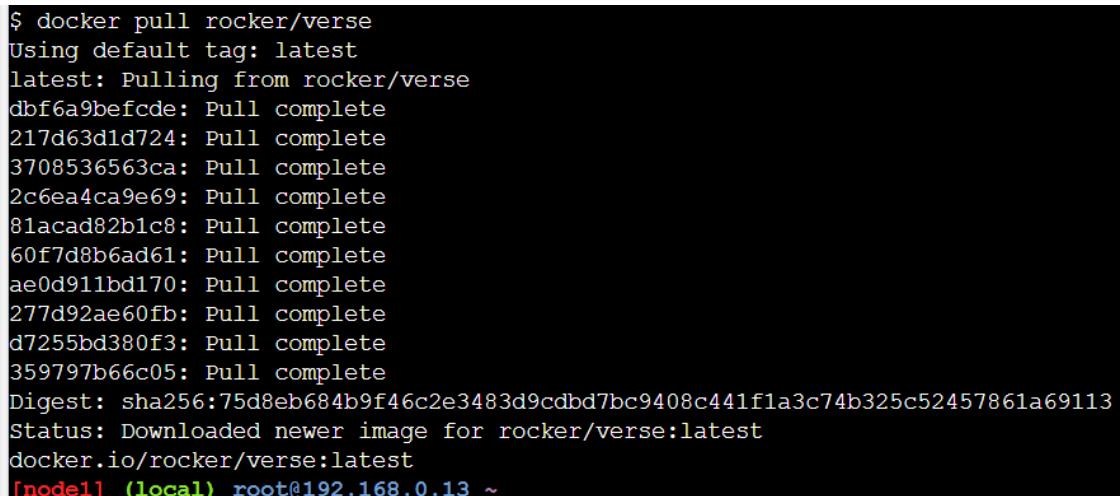


```
[node1] (local) root@192.168.0.13 ~
$ docker --version
Docker version 20.10.17, build 100c701
[node1] (local) root@192.168.0.13 ~
```

Command: to pull readymade image

docker pull rocker/verse

Output:



```
$ docker pull rocker/verse
Using default tag: latest
latest: Pulling from rocker/verse
dbf6a9befcde: Pull complete
217d63d1d724: Pull complete
3708536563ca: Pull complete
2c6ea4ca9e69: Pull complete
81acad82b1c8: Pull complete
60f7d8b6ad61: Pull complete
ae0d911bd170: Pull complete
277d92ae60fb: Pull complete
d7255bd380f3: Pull complete
359797b66c05: Pull complete
Digest: sha256:75d8eb684b9f46c2e3483d9cdbd7bc9408c441f1a3c74b325c52457861a69113
Status: Downloaded newer image for rocker/verse:latest
docker.io/rocker/verse:latest
[node1] (local) root@192.168.0.13 ~
```

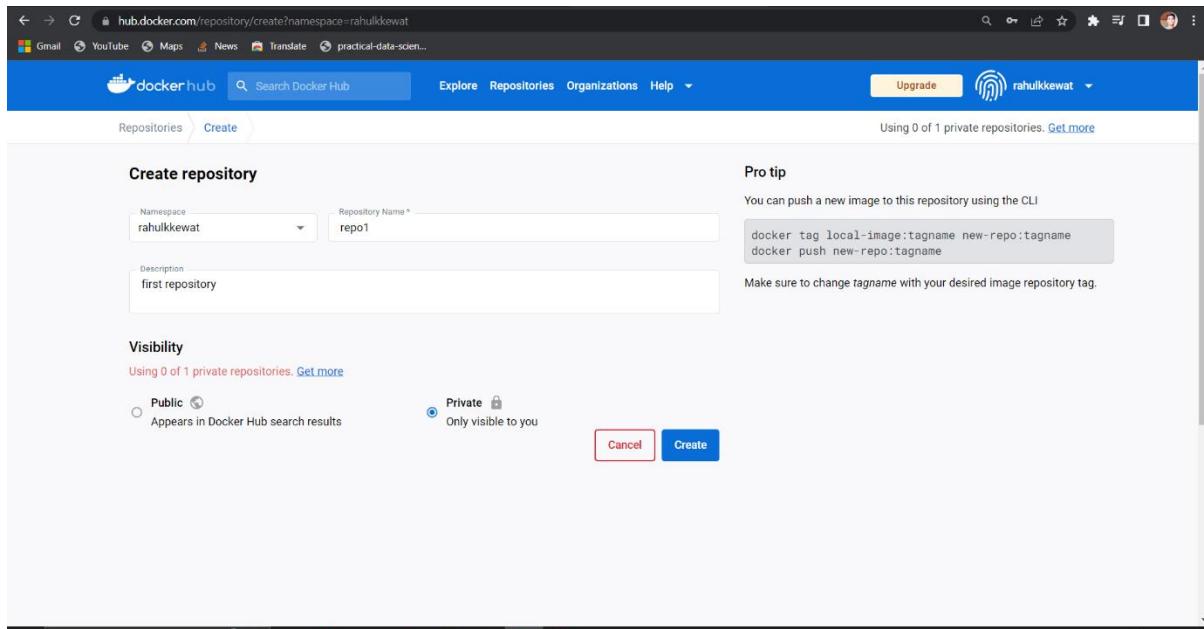
Command: to check images in docker

docker images

Output:

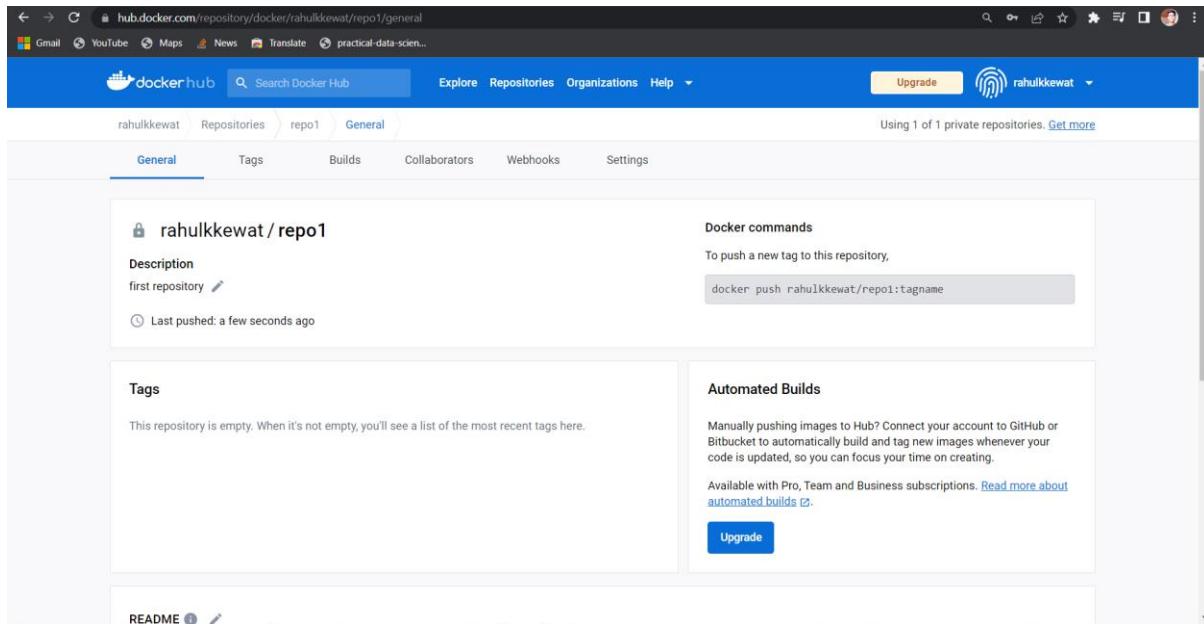
```
[node1] (local) root@192.168.0.13 ~
$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
rocker/verse    latest   9d0311b60ec6  3 days ago  3.43GB
[node1] (local) root@192.168.0.13 ~
```

Now Login to docker hub and create repository

Output:

Click on Create button

Now check repository created



Command: to login to your docker account

docker login –username=rahulkewat

password:

Output:

```
[node1] (local) root@192.168.0.13 ~
$ docker login --username=rahulkewat
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/#credentials-store

Login Succeeded
[node1] (local) root@192.168.0.13 ~
```

Command : to tag image

docker tag 9d0311b60ec6 rahulkewat/repo1:firsttry

note: here 9d0311b60ec6 this is image id which you can get from docker images command.

Output:

```
[node1] (local) root@192.168.0.13 ~
$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
rocker/verse    latest   9d0311b60ec6  3 days ago   3.43GB
[node1] (local) root@192.168.0.13 ~
$ docker tag 9d0311b60ec6 rahulkewat/repo1:firsttry
[node1] (local) root@192.168.0.13 ~
```

Command: to push image to docker hub account

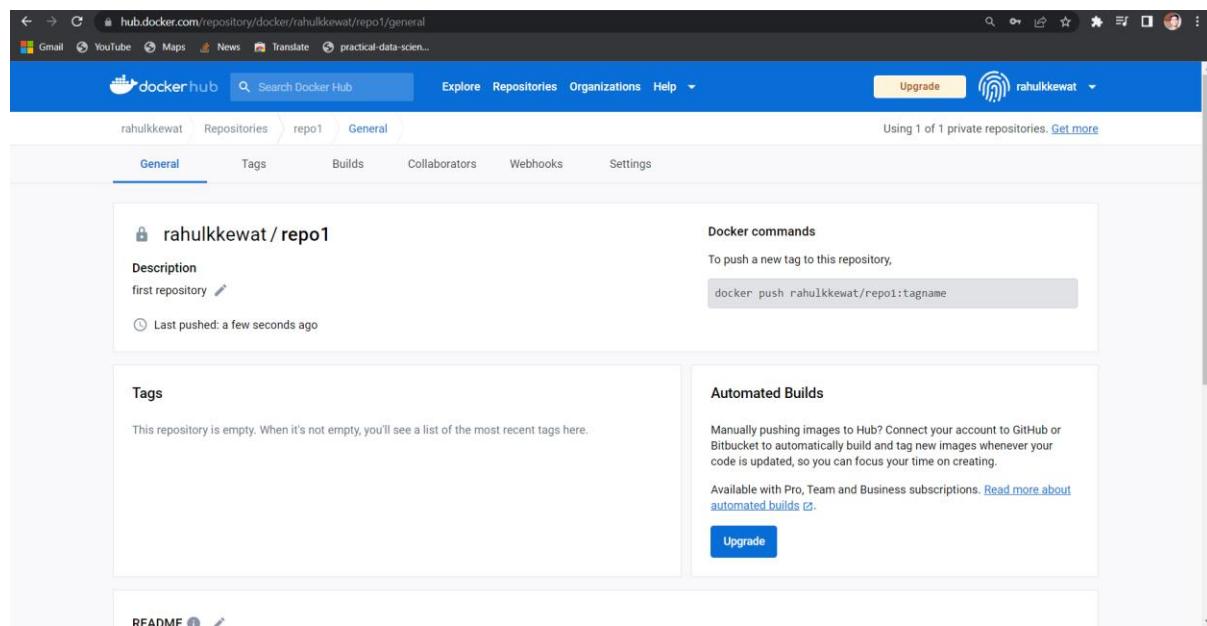
docker push rahulkewat/repo1:firsttry

note: firsttry is tag name created above.

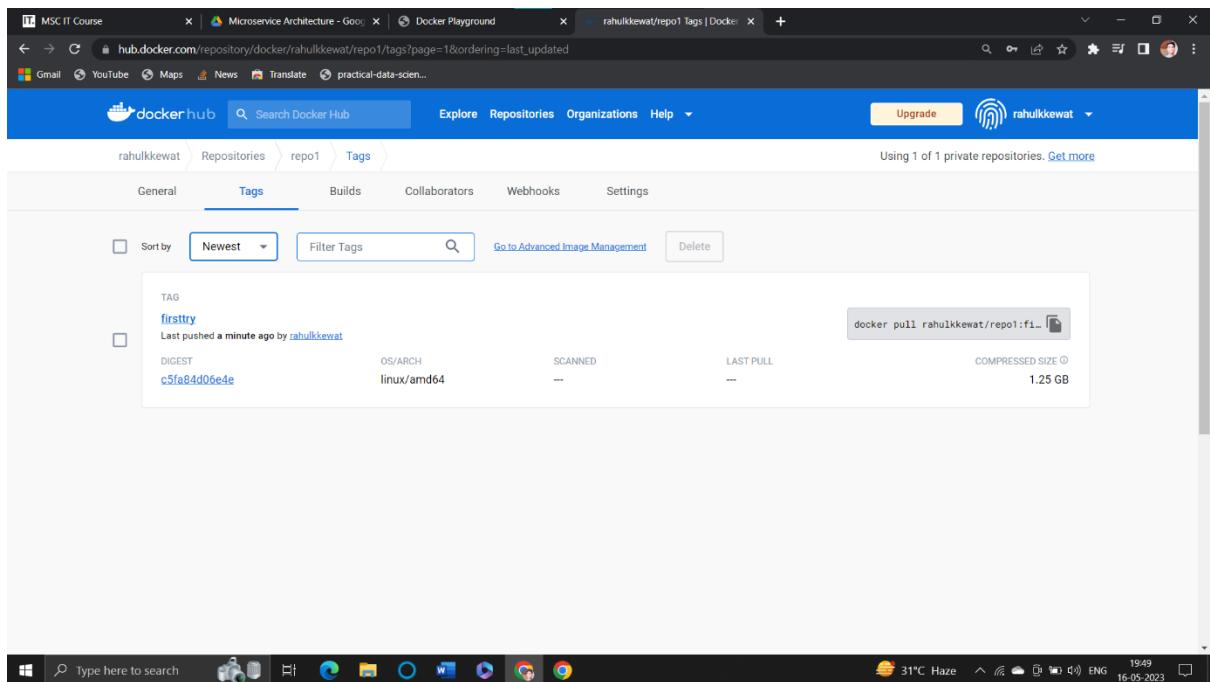
Output:

```
[node1] (local) root@192.168.0.13 ~
$ docker tag 9d0311b60ec6 rahulkewat/repo1:firsttry
[node1] (local) root@192.168.0.13 ~
$ docker push rahulkewat/repo1:firsttry
The push refers to repository [docker.io/rahulkewat/repo1]
d48c1f84033f: Mounted from rocker/verse
19104f1ab534: Mounted from rocker/verse
213229d6f055: Mounted from rocker/verse
f943d8a59e83: Mounted from rocker/verse
037ce214b08c: Mounted from rocker/verse
24a8e79e576d: Mounted from rocker/verse
e95c9f6015cb: Mounted from rocker/verse
a0157309e33c: Mounted from rocker/verse
de44b0398654: Mounted from rocker/verse
b8a36d10656a: Mounted from rocker/verse
firsttry: digest: sha256:c5fa84d06e4e536bfde730e90e0b54f7714c222d572d18e60c047c5839b76f57 size: 2428
[node1] (local) root@192.168.0.13 ~
```

Check it in docker hub now



Click on tags and check



Method 2:

Build an image then push it to docker and run it

Command : to create docker file

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

Output:

```
[node1] (local) root@192.168.0.13 ~
$ cat > Dockerfile <<EOF
> FROM busybox
> CMD echo "Hello world! This is my first Docker image."
> EOF
[node1] (local) root@192.168.0.13 ~
$ docker build -t rahulkewat/repo2
"docker build" requires exactly 1 argument.
See 'docker build --help'.

Usage: docker build [OPTIONS] PATH | URL | -
      Build an image from a Dockerfile
[node1] (local) root@192.168.0.13 ~
```

Command : to build image from docker file

dokcer build -t rahulkewat/repo2 .

Output:

```
[node1] (local) root@192.168.0.13 ~
$ docker build -t rahulkewat/repo2 .
Sending build context to Docker daemon 12.8kB
Step 1/2 : FROM busybox
latest: Pulling from library/busybox
a58ecd4f0c86: Pull complete
Digest: sha256:9e2bbca079387d7965c3a9cee6d0c53f4f4e63ff7637877a83c4c05f2a666112
Status: Downloaded newer image for busybox:latest
--> af2c3e96bcf1
Step 2/2 : CMD echo "Hello world! This is my first Docker image."
--> Running in 1518d05eeeeee
Removing intermediate container 1518d05eeeeee
--> 773f1f708953
Successfully built 773f1f708953
Successfully tagged rahulkewat/repo2:latest
[node1] (local) root@192.168.0.13 ~
```

Command: to check docker images

docker images

```
[node1] (local) root@192.168.0.13 ~
$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
rahulkewat/repo2  latest   773f1f708953  25 seconds ago  4.86MB
rahulkewat/repol  firsttry  9d0311b60ec6  3 days ago   3.43GB
rocker/verse      latest   9d0311b60ec6  3 days ago   3.43GB
busybox           latest   af2c3e96bcf1  4 days ago   4.86MB
[node1] (local) root@192.168.0.13 ~
```

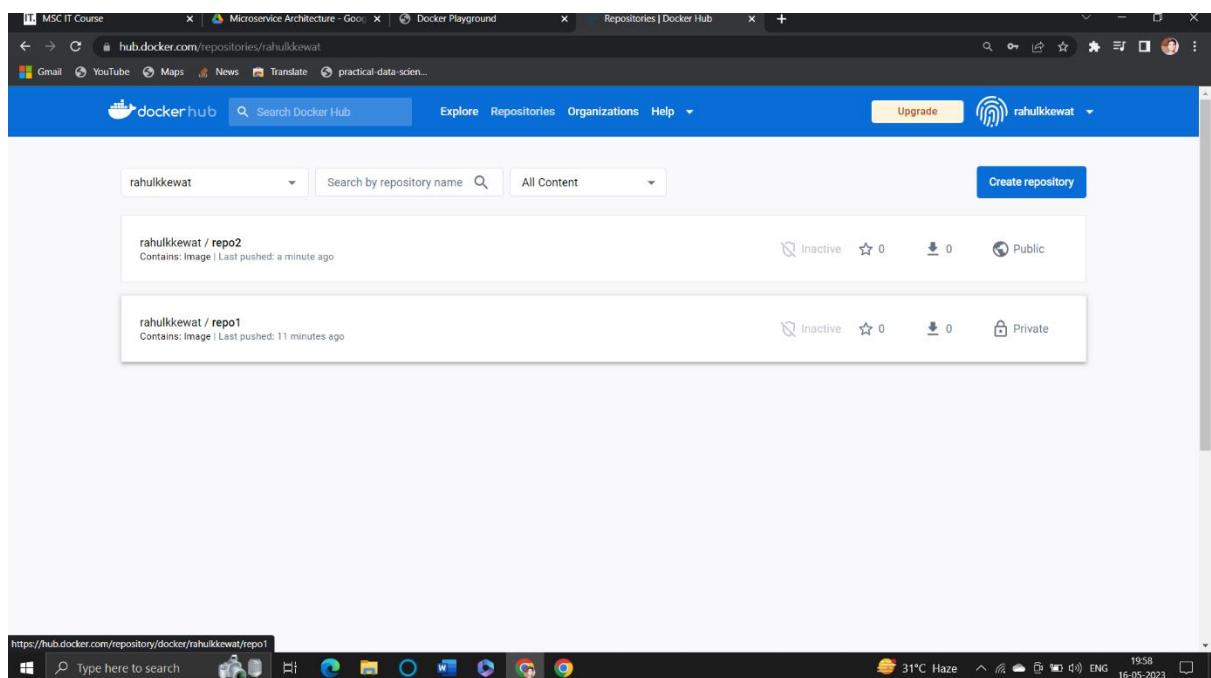
Command: to push image to docker hub

docker push rahulkewat/repo2

Output:

```
[node1] (local) root@192.168.0.13 ~
$ docker push rahulkewat/repo2
Using default tag: latest
The push refers to repository [docker.io/rahulkewat/repo2]
1f1d08b81bbe: Mounted from library/busybox
latest: digest: sha256:91a3413f9966bb9ccd39199c79d14c886bf799fa8e671681a7daf6f37c561ca7 size: 528
```

Now check it on docker hub



command: to run docker image:

docker run rahulkewat/repo2

output:

```
[node1] (local) root@192.168.0.13 ~
$ docker run rahulkewat/repo2
Hello world! This is my first Docker image.
[node1] (local) root@192.168.0.13 ~
$ ]
```

Practical 4

Aim: Installing software packages on Docker, Working with Docker Volumes and Networks.

Writeup:

Step 1: Working with Basic Functionalities Docker**[A]:** Creating a volume using the **docker volume** command.

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

C:\Users\Dell>docker volume

Usage: docker volume COMMAND

Manage volumes

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused local volumes
  rm          Remove one or more volumes

Run 'docker volume COMMAND --help' for more information on a command.
```

[B]: Creating the Actual Volume using command **docker volume create myvol1**

```
C:\Users\Dell>docker volume create myvol1
myvol1
```

[C]: To list the volume we will write the command **docker volume ls**

```
C:\Users\Dell>docker volume ls
DRIVER      VOLUME NAME
local      myvol1
local      myvol1
```

[D]: To get the details of our volume we have to write the command **docker volume inspect myvol1**

```
C:\Users\Dell>docker volume inspect myvol1
[
  {
    "CreatedAt": "2023-05-18T14:05:03Z",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/myvol1/_data",
    "Name": "myvol1",
    "Options": null,
    "Scope": "local"
  }
]
```

[E]: To remove your volume you can use the command **docker volume rm myvol1** Also using **docker volume ls** to confirm that the volume has been removed.

```
C:\Users\Del1>docker volume rm myvol1  
myvol1  
  
C:\Users\Del1>docker volume ls  
DRIVER      VOLUME NAME  
local        myvol1
```

Step 2: Working with Docker Network

[A]: To Connect a container to a network using command **docker network create Vol**

```
C:\Users\Del1>docker network create Vol  
c6513820991a23026f6edd67f102ef76616b1f6ca9f2fd3f7f1d8e71ee1a78e4
```

[B]: To get details of a container from a network using command **docker network inspect Vol**

```
C:\Users\Del1>docker network inspect Vol  
[  
  {  
    "Name": "Vol",  
    "Id": "c6513820991a23026f6edd67f102ef76616b1f6ca9f2fd3f7f1d8e71ee1a78e4",  
    "Created": "2023-05-18T14:08:01.255257083Z",  
    "Scope": "local",  
    "Driver": "bridge",  
    "EnableIPv6": false,  
    "IPAM": {  
      "Driver": "default",  
      "Options": {},  
      "Config": [  
        {  
          "Subnet": "172.19.0.0/16",  
          "Gateway": "172.19.0.1"  
        }  
      ]  
    },  
    "Internal": false,  
    "Attachable": false,  
    "Ingress": false,  
    "ConfigFrom": {  
      "Network": ""  
    },  
    "ConfigOnly": false,  
    "Containers": {},  
    "Options": {},  
    "Labels": {}  
  }  
]
```

[C]: To see the list of networks use command **docker network ls**

```
C:\Users\Del\>docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
c6513820991a    Vol       bridge      local
ad8b6fd8eac0    bridge     bridge      local
f8fd230d380f    host       host       local
ed6a373a9a1c    none      null       local
```

[D]: To remove all unused networks using the command **docker network prune** Also using **docker network ls** to confirm the removal of the network.

```
C:\Users\Del\>docker network prune
WARNING! This will remove all custom networks not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Networks:
Vol

C:\Users\Del\>docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
ad8b6fd8eac0    bridge     bridge      local
f8fd230d380f    host       host       local
ed6a373a9a1c    none      null       local
```

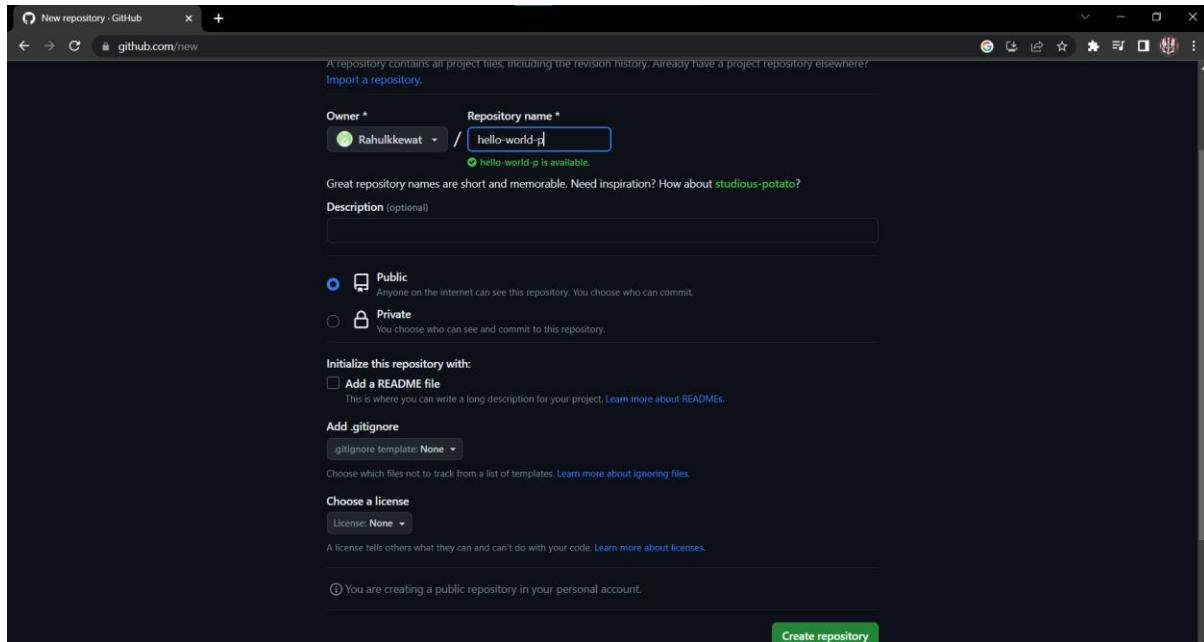
Practical 5

Aim: Working with Circle CI for continuous integration

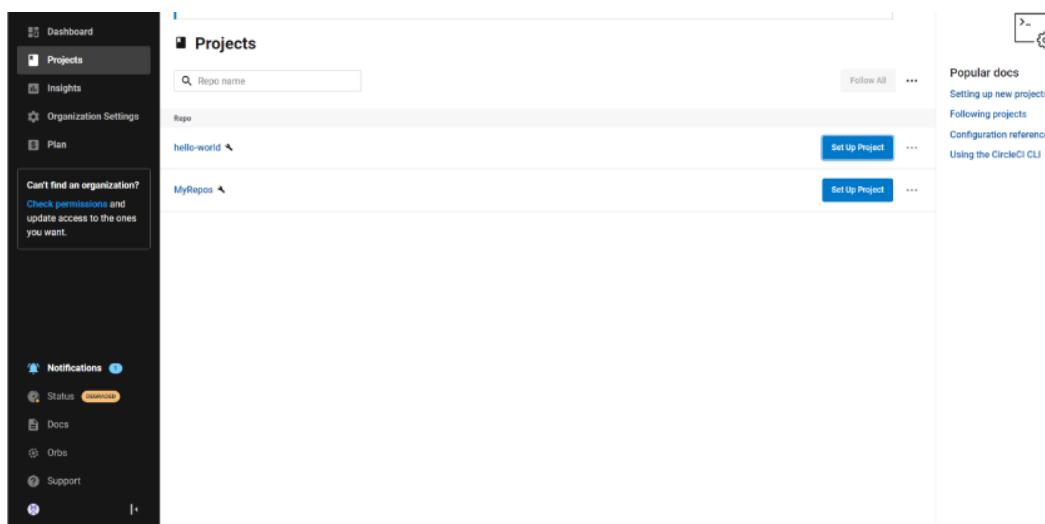
Writeup:

Step 1: Create a repository

1. Log in to GitHub and begin the process to create a new repository.
2. Enter a name for your repository (for example, hello-world).
3. Select the option to initialize the repository with a README file.
4. Finally, click Create repository.
5. There is no need to add any source code for now.

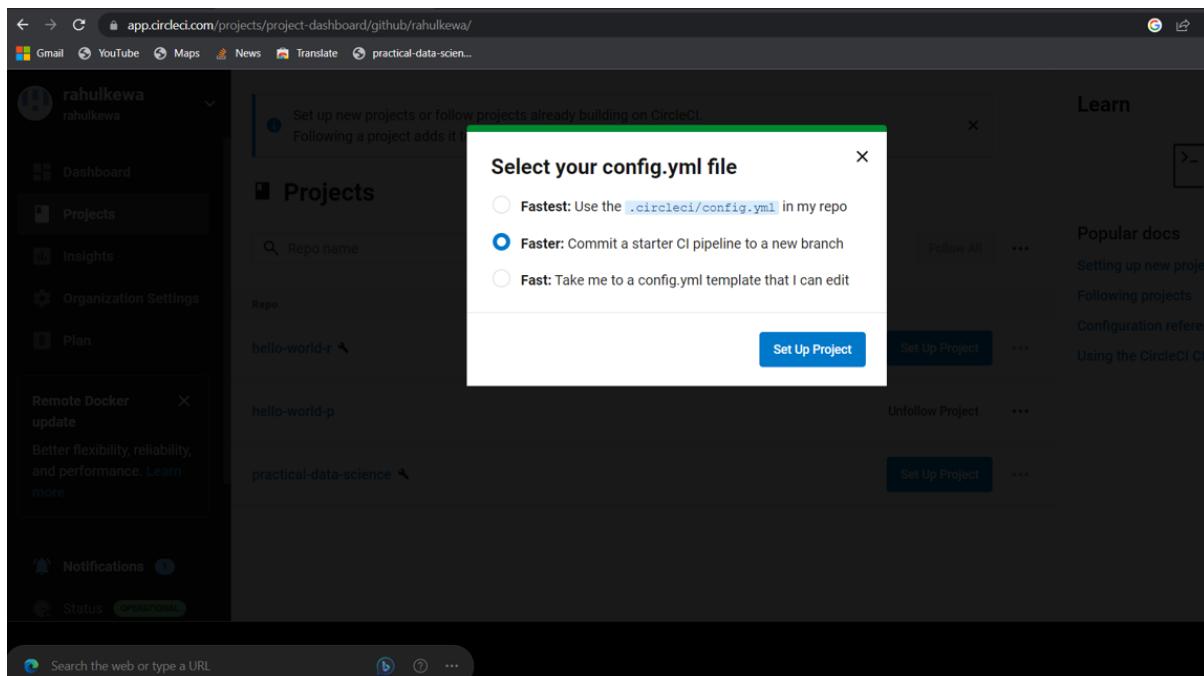


Login to Circle CI <https://app.circleci.com/> Using GitHub Login, Once logged in navigate to Projects.



Step 2: Set up CircleCI

1. Navigate to the CircleCI Projects page. If you created your new repository under an organization, you will need to select the organization name.
2. You will be taken to the Projects dashboard. On the dashboard, select the project you want to set up (hello-world).
3. Select the option to commit a starter CI pipeline to a new branch, and click Set Up Project. This will create a file `.circleci/config.yml` at the root of your repository on a new branch called `circleci-project-setup`.



Step 3: Your first pipeline

On your project's pipeline page, click the green Success button, which brings you to the workflow that ran (say-hello-workflow).

Within this workflow, the pipeline ran one job, called say-hello. Click say-hello to see the steps in this job:

- a. Spin up environment
- b. Preparing environment variables
- c. Checkout code
- d. Say hello

Now select the “say-hello-workflow” to the right of Success status column

Pipeline	Status	Workflow	Branch / Commit	Start	Duration	Actions
hello-world-p_2	Success	say-hello-workflow	circleci-project-setup 860626e Add .circleci/config.yml	57s ago	6s ↑	↻ ⟳ ✖ ...
hello-world-p_1	Success	say-hello-workflow	circleci-project-setup 860626e	58s ago	7s	↻ ⟳ ✖ ...

Select “say-hello” Job with a green tick

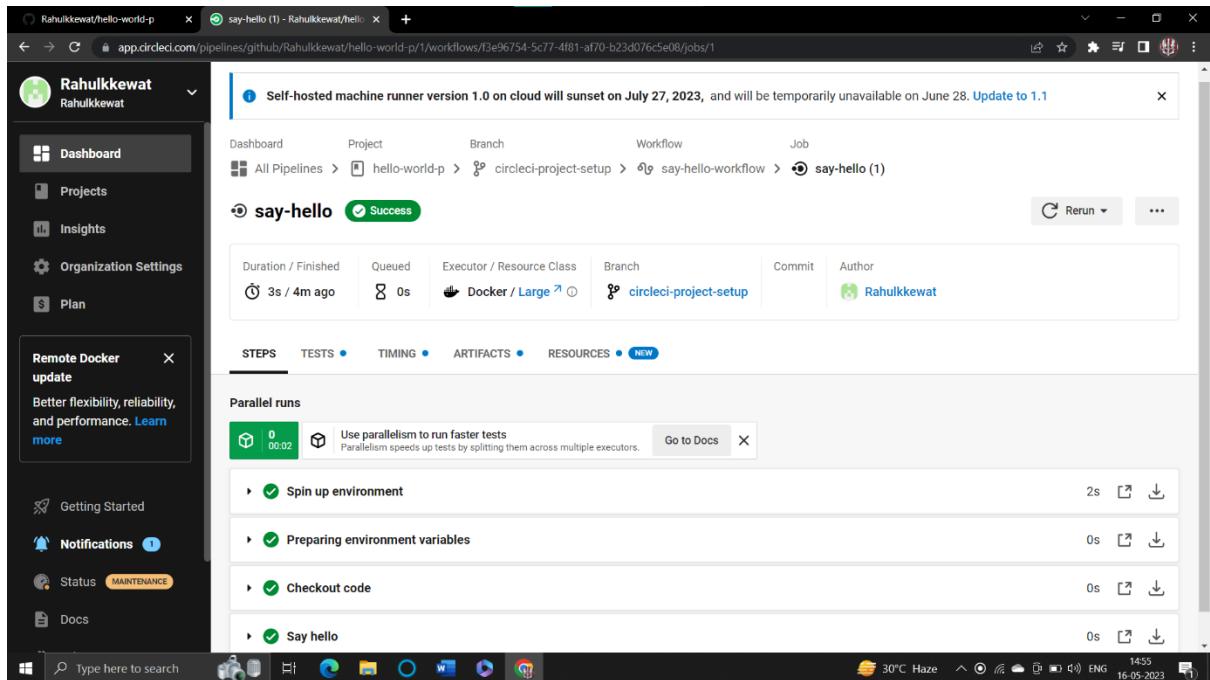
Duration / Finished	Branch	Commit	Author
7s / 1m ago	circleci-project-setup	860626e	Rahulkewat

✓ say-hello 4s

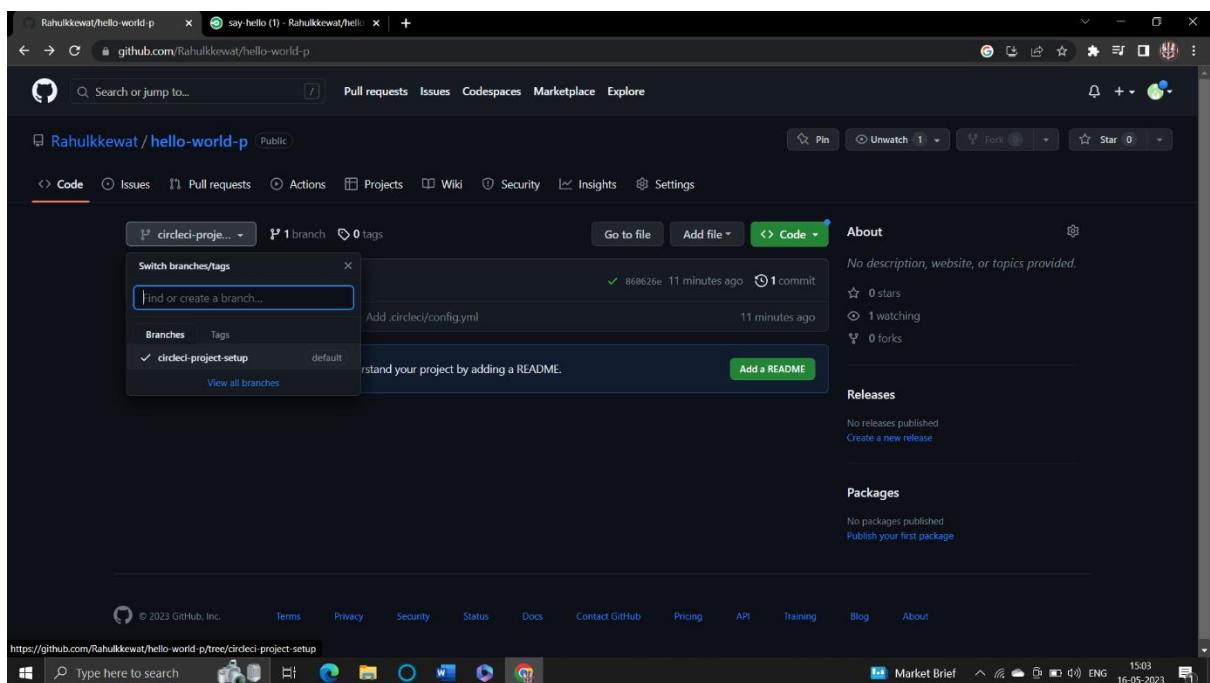
Did you know? CircleCI teams that commit 4x as often fix failed builds 2x faster.

Learn more ✖

20 pipelines/day 70 min to recovery
5 pipelines/day 142 min



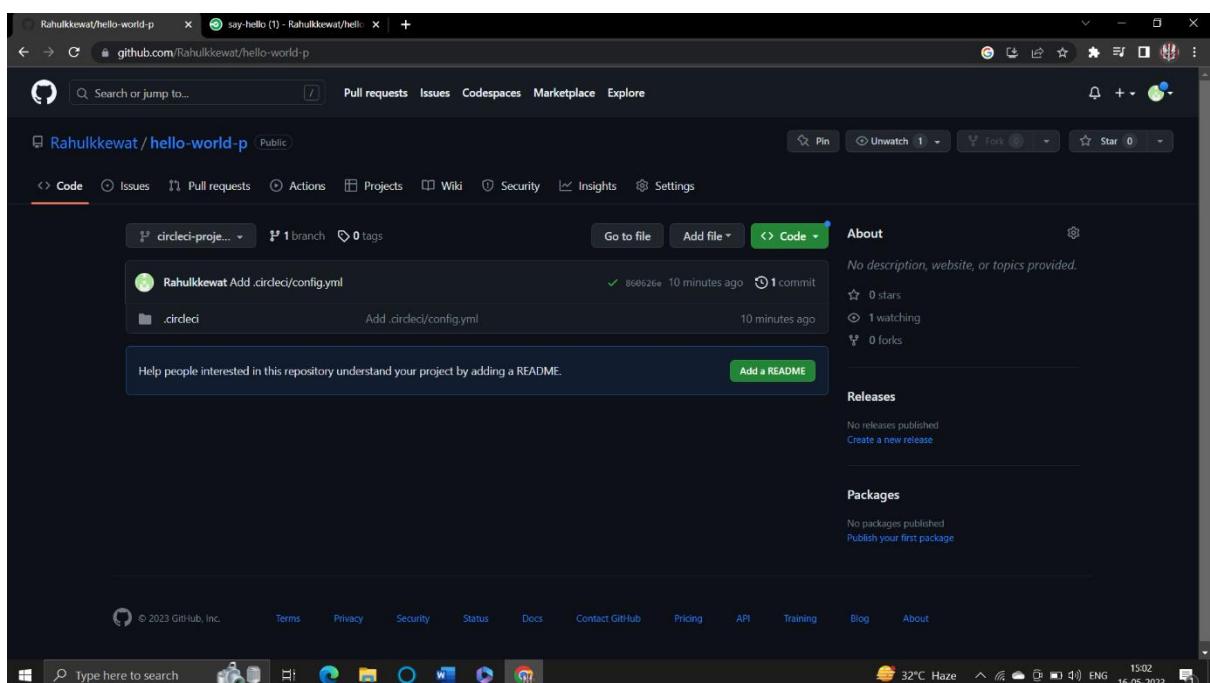
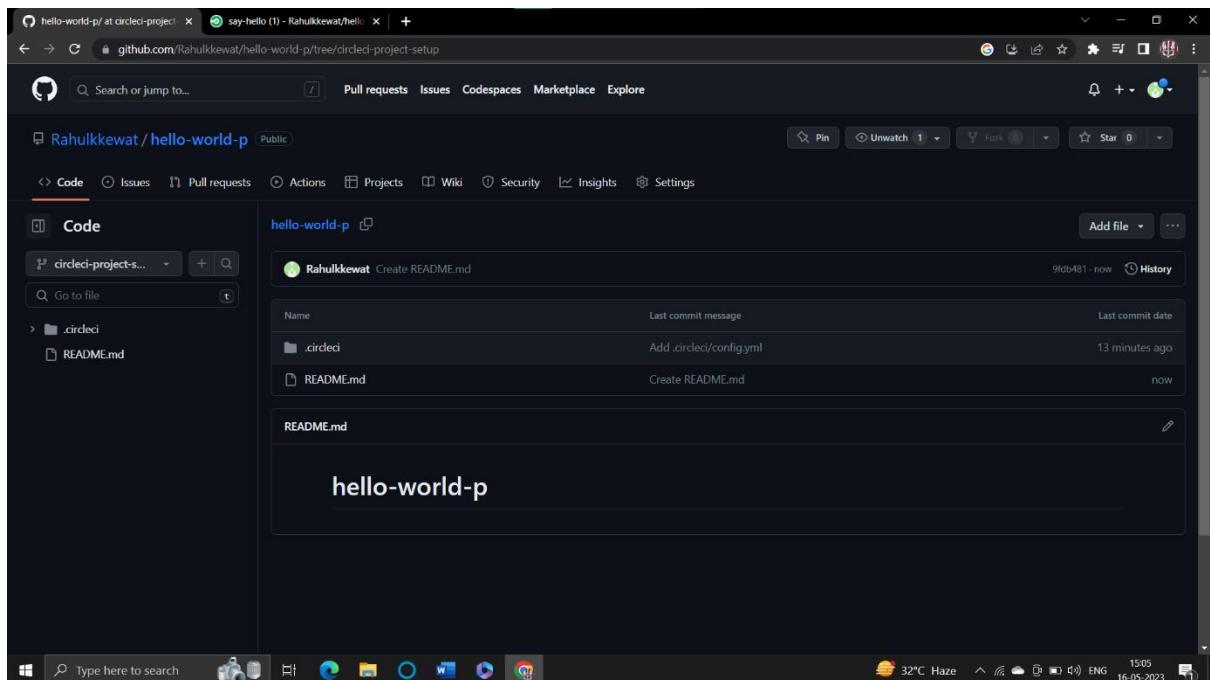
Select Branch and option circleci-project-setup



Step 4: Break your build

In this section, you will edit the `.circleci/config.yml` file and see what happens if a build does not complete successfully.

It is possible to edit files directly on GitHub.



```

1  # Use the latest 2.1 version of CircleCI pipeline process engine.
2  # See: https://circleci.com/docs/configuration-reference
3  version: 2.1
4
5  # Define a job to be invoked later in a workflow.
6  # See: https://circleci.com/docs/configuration-reference/#jobs
7  jobs:
8    say-hello:
9      # Specify the execution environment. You can specify an image from Docker Hub or use one of our convenience images from CircleCI's Developer Hub.
10     # See: https://circleci.com/docs/configuration-reference/#executor-job
11     docker:
12       - image: circleci/base:stable
13       # Add steps to the job
14       # See: https://circleci.com/docs/configuration-reference/#steps
15       steps:
16         - checkout
17         - run:
18           name: "Say hello"
19           command: "echo Hello, World!"
20
21   # Orchestrate jobs using workflows
22   # See: https://circleci.com/docs/configuration-reference/#workflows
23   workflows:
24     say-hello-workflow:
25       jobs:
26         - say-hello

```

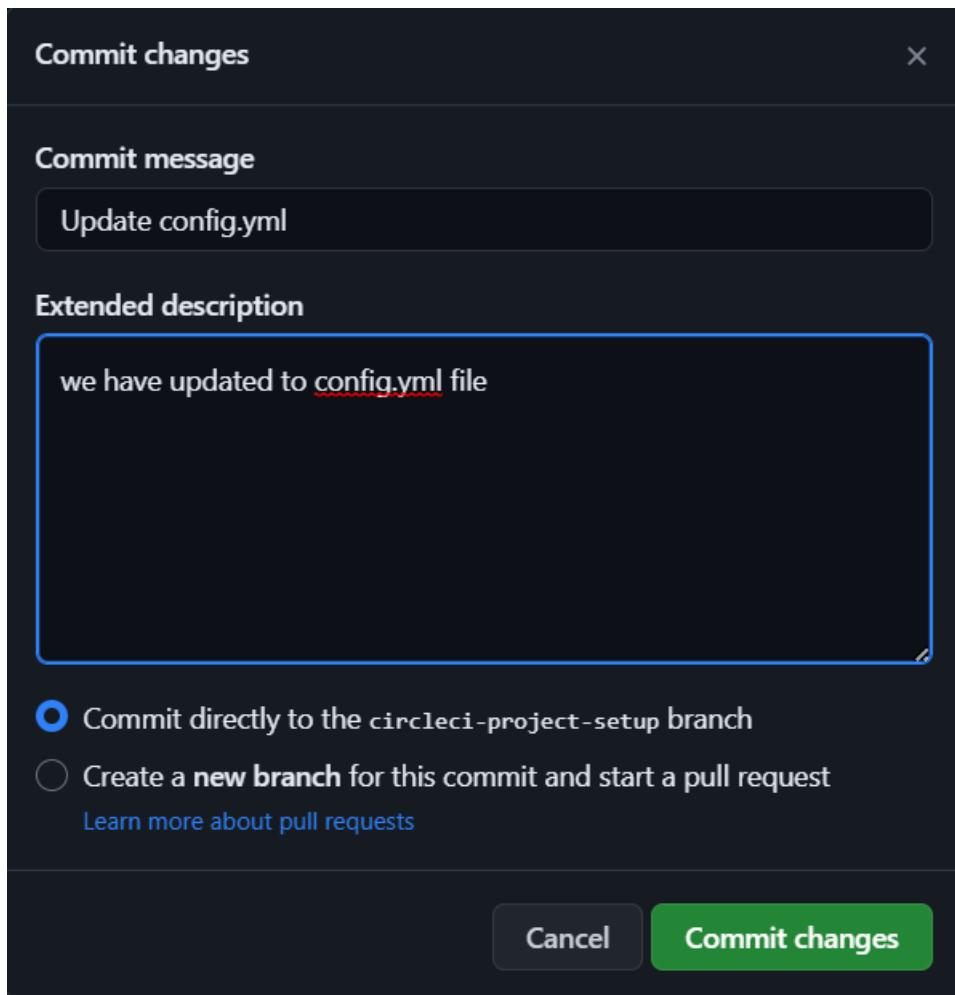
The GitHub file editor should look like this

```

1  version: 2.1
2  orbs:
3    node: circleci/node@4.7.0
4  jobs:
5    build:
6      executor:
7        name: node/default
8        tag: '10.4'
9      steps:
10        - checkout
11        - node/with-cache:
12          steps:
13            - run: npm install
14            - run: npm run test

```

Scroll down and Commit your changes on GitHub



After committing your changes, then return to the Projects page in CircleCI. You should see a new pipeline running... and it will fail! What's going on? The Node orb runs some common Node tasks. Because you are working with an empty repository, running `npm run test`, a Node script, causes the configuration to fail. To fix this, you need to set up a Node project in your repository.

Pipeline	Status	Workflow	Branch / Commit	Start	Duration	Actions
hello-world-p_4	⚠️ Error	say-hello-workflow	circleci-project-setup 9fdb481 Create README.md	17m ago	7s	View Logs Details ...
hello-world-p_3	Success	say-hello-workflow	circleci-project-setup 860e26e Add .circleci/config.yml	30m ago	6s	View Logs Details ...
hello-world-p_2	Success	say-hello-workflow	circleci-project-setup 860e26e Add .circleci/config.yml	30m ago	7s	View Logs Details ...
hello-world-p_1	Success	say-hello-workflow	circleci-project-setup 860e26e	30m ago	7s	View Logs Details ...

Step 5: Use Workflows

You do not have to use orbs to use CircleCI. The following example details how to create a custom configuration that also uses the workflow feature of CircleCI.

- 1) Take a moment and read the comments in the code block below. Then, to see workflows in action, edit your .circleci/config.yml file and copy and paste the following text into it.

```
1  version: 2
2  jobs: # we now have TWO jobs, so that a workflow can coordinate them!
3    one: # This is our first job.
4      docker: # it uses the docker executor
5        - image: cimg/ruby:2.6.8 # specifically, a docker image with ruby 2.6.8
6        auth:
7          username: mydockerhub-user
8          password: $DOCKERHUB_PASSWORD # context / project UI env-var reference
9      # Steps are a list of commands to run inside the docker container above.
10     steps:
11       - checkout # this pulls code down from GitHub
12       - run: echo "A first hello" # This prints "A first hello" to stdout.
13       - run: sleep 25 # a command telling the job to "sleep" for 25 seconds.
14   two: # This is our second job.
15     docker: # it runs inside a docker image, the same as above.
16       - image: cimg/ruby:3.0.2
17       auth:
18         username: mydockerhub-user
19         password: $DOCKERHUB_PASSWORD # context / project UI env-var reference
20     steps:
21       - checkout
22       - run: echo "A more familiar hi" # We run a similar echo command to above.
23       - run: sleep 15 # and then sleep for 15 seconds.
24   # Under the workflows: map, we can coordinate our two jobs, defined above.
25 workflows:
26   version: 2
27   one_and_two: # this is the name of our workflow
28     jobs: # and here we list the jobs we are going to run.
29       - one
30       - two
```

- 2) Commit these changes to your repository and navigate back to the CircleCI Pipelines page. You should see your pipeline running.

A screenshot of a Windows desktop environment. In the center, a Microsoft Edge browser window displays a GitHub pull request for a file named 'config.yml' in a repository named 'hello-world-p'. The code editor shows the YAML configuration for a CircleCI pipeline. Below the browser, a terminal window is open, showing the command 'git status' which indicates there are no changes staged or committed.

```

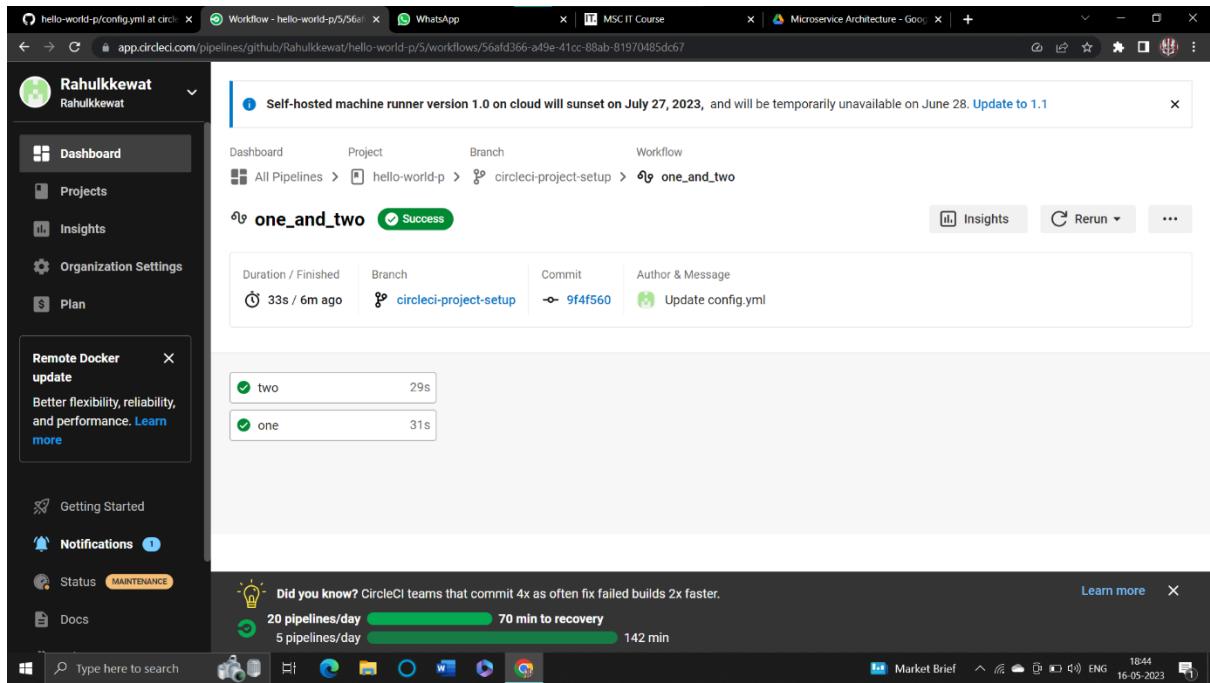
version: 2
jobs:
  one:
    docker:
      - image: cimg/ruby:2.6.8
      auth:
        username: kaif3120
        password: $00C_PASS #env
    steps:
      - checkout
      - run: echo "A first hello"
      - run: sleep 25
  two:
    docker:
      - image: cimg/ruby:3.0.2
      auth:
        username: kaif3120
        password: $00C_PASS
    steps:
      - checkout
      - run: echo "A more familiar HI"
      - run: sleep 15
workflows:
  version: 2
  one_and_two:
    jobs:
      - one
      - two

```

- 3) Click on the running pipeline to view the workflow you have created. You should see that two jobs ran (or are currently running!) concurrently.

A screenshot of the CircleCI web interface. On the left, a sidebar shows the user's organization 'Rahulkewat' with options like Dashboard, Projects, Insights, Organization Settings, and Plan. A prominent 'Remote Docker update' message is displayed. The main area is titled 'All Pipelines' and lists several pipelines. Pipeline 'hello-world-p' is shown as 'Running' with the workflow 'one_and_two'. Other pipelines listed include 'hello-world-p' (status 'Error calling workflow'), 'hello-world-p' (status 'Success'), 'hello-world-p' (status 'Success'), and 'hello-world-p' (status 'Success'). Each pipeline row includes a 'Actions' column with icons for viewing logs, cloning, and deleting.

Pipeline	Status	Workflow	Branch / Commit	Start	Duration	Actions
hello-world-p 5	Running	one_and_two	circleci-project-setup 9f4f560 Update config.yml	14s ago	27s	  
hello-world-p 4	Error	'workflow'	Error calling job: 'build' Cannot find a definition for command named node/with-cache			
hello-world-p 3	Success	say-hello-workflow	circleci-project-setup 9fdb481 Create README.md	4h ago	7s	  
hello-world-p 2	Success	say-hello-workflow	circleci-project-setup 860626e Add .circleci/config.yml	4h ago	6s	  
hello-world-p 1	Success	say-hello-workflow	circleci-project-setup 860626e	4h ago	7s	  



Step 6: Add some changes to use workspaces

Each workflow has an associated workspace which can be used to transfer files to downstream jobs as the workflow progresses. You can use workspaces to pass along data that is unique to this run and which is needed for downstream jobs. Try updating config.yml to the following:

```

1  version: 2
2  jobs:
3    one:
4      docker:
5        - image: cimg/ruby:3.0.2
6        auth:
7          username: mydockerhub-user
8          password: $DOCKERHUB_PASSWORD # context / project UI env-var reference
9      steps:
10        - checkout
11        - run: echo "A first hello"
12        - run: mkdir -p my_workspace
13        - run: echo "Trying out workspaces" > my_workspace/echo-output
14        - persist_to_workspace:
15          # Must be an absolute path, or relative path from working_directory
16          root: my_workspace
17          # Must be relative path from root
18        paths:
19          - echo-output
20    two:
21      docker:
22        - image: cimg/ruby:3.0.2
23        auth:
24          username: mydockerhub-user
25          password: $DOCKERHUB_PASSWORD # context / project UI env-var reference
26      steps:
27        - checkout
28        - run: echo "A more familiar hi"
29        - attach_workspace:
30          # Must be absolute path or relative path from working_directory
31          at: my_workspace
32
33        - run: |
34          if [[ $(cat my_workspace/echo-output) == "Trying out workspaces" ]]; then
35            echo "It worked!";
36          else
37            echo "Nope!";
38          fi
39    workflows:
40      version: 2
41      one_and_two:
42        jobs:
43          - one
44          - two:
45            requires:
46              - one

```

Updated config.yml in GitHub file editor should be updated like this

```

version: 2
jobs:
  one:
    docker:
      - image: cimg/ruby:2.6.8
      auth:
        username: kaiF3120
        password: $00C_PASS #env
    steps:
      - checkout
      - run: echo "A first hello"
      - run: mkdir -p my_workspace
      - run: echo "Trying out workspaces" > my_workspace/echo-output
      - persist_to_workspace:
          root: my_workspace
          paths:
            - echo-output
  two:
    docker:
      - image: cimg/ruby:3.0.2
      auth:
        username: kaiF3120
        password: $00C_PASS
    steps:
      - checkout
      - run: echo "A more familiar HI"
      - attach_workspace:
          at: my_workspace
      - run: |
          if [[ $(cat my_workspace/echo-output) == "Trying out workspaces" ]]; then
            echo "It worked!"
          else
            echo "Nope!";
            exit 1
          fi
  workflows:
    version: 2
    one_and_two:
      jobs:
        - one
        - two:
          requires:
            - one

```

Finally your workflow with the jobs running should look like this

The screenshot shows the CircleCI dashboard for the 'hello-world-p' repository. A success message for the 'one_and_two' workflow is displayed. The workflow details show two steps: 'one' and 'two', both completed successfully. The total duration was 15s / 11s ago. The commit hash is abf3209. A 'Did you know?' statistic indicates that teams committing 4x as often fix failed builds 2x faster.

Practical 6

Aim: Creating Microservice with ASP.NET Core.

Writeup:

Step 1: Create new project.

Command: **dotnet new webapi -o TeamService**

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

E:\RAHUL>dotnet new webapi -o TeamService
The template "ASP.NET Core Web API" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on TeamService\TeamService.csproj...
  Restore completed in 78.28 ms for E:\RAHUL\TeamService\TeamService.csproj.

Restore succeeded.
```

Step 2: Remove existing weatherforecast files both model and controller files.

Step 3: Add new files as follows:

[A]: Add Member.cs to “C:\Users\RAHUL\TeamService\Models” folder.

```
using System;
namespace TeamService.Models
{
    public class Member
    {
        public Guid ID
        {
            get;
            set;
        }

        public string FirstName
        {
            get;
            set;
        }

        public string LastName
        {
            get;
            set;
        }

        public Member()
        {
```

```
    }

    public Member(Guid id) : this()
    {
        this.ID = id;
    }

    public Member(string firstName, string lastName, Guid id) : this(id)
    {
        this.FirstName = firstName;
        this.LastName = lastName;
    }

    public override string ToString()
    {
        return this.LastName;
    }
}
```

[B]: Add Team.cs to “C:\Users\RAHUL\TeamService\Models” folder.

```
using System;
using System.Collections.Generic;

namespace TeamService.Models

{
    public class Team
    {
        public string Name
        {
            get;
            set;
        }

        public Guid ID
        {
            get;
            set;
        }

        public ICollection<Member> Members
        {
            get;
            set;
        }
}
```

```
public Team()
{
    this.Members = new List<Member>();
}

public Team(string name) : this()
{
    this.Name = name;
}

public Team (string name, Guid id) : this(name)
{
    this.ID = id;
}

public override string ToString()
{
    return this.Name;
}
}
```

[C]: Add TeamsController.cs file to “C:\Users\RAHUL\TeamService\Controllers” folder.

```
using System;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Linq;
using TeamService.Models;
using System.Threading.Tasks;
using TeamService.Persistence;

namespace TeamService
{
    [Route("[controller]")]
    public class TeamsController : Controller
    {
        ITeamRepository repository;
        public TeamsController(ITeamRepository repo)
        {
            repository = repo;
        }

        [HttpGet]
        public virtual IActionResult GetAllTeams()
```

```
{  
    return this.Ok(repository.List());  
}  
  
[HttpGet("{id}")]  
public IActionResult GetTeam(Guid id)  
{  
    Team team = repository.Get(id);  
    if (team != null)  
    {  
        return this.Ok(team);  
    }  
  
    else  
    {  
        return this.NotFound();  
    }  
}  
  
[HttpPost]  
public virtual IActionResult CreateTeam ([FromBody]Team newTeam)  
{  
    repository.Add(newTeam);  
    return this.Created($"/teams/{newTeam.ID}", newTeam);  
}  
  
[HttpPut("{id}")]  
public virtual IActionResult UpdateTeam([FromBody]Team team, Guid id)  
{  
    team.ID = id;  
    if(repository.Update(team) == null)  
    {  
        return this.NotFound();  
    }  
  
    else  
    {  
        return this.Ok(team);  
    }  
}  
  
[HttpDelete("{id}")]  
public virtual IActionResult DeleteTeam(Guid id)  
{  
    Team team = repository.Delete(id);  
    if(team == null)  
    {  
        return this.NotFound();  
    }  
}
```

```
        }

        else
        {
            return this.Ok(team.ID);
        }
    }
}
```

[D]: Add MembersController.cs file to “C:\Users\RAHUL\TeamService\Controllers” folder.

```
using System;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Linq;
using TeamService.Models;
using System.Threading.Tasks;
using TeamService.Persistence;

namespace TeamService

{
    [Route("/teams/{teamId}/[controller]")]
    public class MembersController : Controller
    {
        ITeamRepository repository;
        public MembersController(ITeamRepository repo)
        {
            repository = repo;
        }

        [HttpGet]
        public virtual IActionResult GetMembers(Guid teamID)
        {
            Team team = repository.Get(teamID);
            if(team == null)
            {
                return this.NotFound();
            }

            else
            {
```

```
        return this.Ok(team.Members);
    }
}

[HttpGet]
[Route("/teams/{teamId}/[controller]/{memberId}")]
public virtual IActionResult GetMember(Guid teamID, Guid memberId)
{
    Team team = repository.Get(teamID);

    if(team == null)
    {
        return this.NotFound();
    }

    else
    {
        var q = team.Members.Where(m => m.ID == memberId);
        if(q.Count() < 1)
        {
            return this.NotFound();
        }

        else
        {
            return this.Ok(q.First());
        }
    }
}

[HttpPut]
[Route("/teams/{teamId}/[controller]/{memberId}")]
public virtual IActionResult UpdateMember ([FromBody]Member
updatedMember, Guid teamID, Guid memberId)
{
    Team team = repository.Get(teamID);
    if(team == null)
    {
        return this.NotFound();
    }

    else
    {
        var q = team.Members.Where(m => m.ID == memberId);
        if(q.Count() < 1)
        {
            return this.NotFound();
        }
    }
}
```

```

        else
    {
        team.Members.Remove(q.First());
        team.Members.Add(updatedMember);
        return this.Ok();
    }
}

[HttpPost]
public virtual IActionResult CreateMember([FromBody]Member newMember,
Guid teamID)
{
    Team team = repository.Get(teamID);
    if(team == null)
    {
        return this.NotFound();
    }

    else
    {
        team.Members.Add(newMember);
        var teamMember = new {TeamID = team.ID, MemberID =
newMember.ID};
        return
this.Created($"~/teams/{teamMember.TeamID}/{[controller]}/{teamMember.MemberID}",

teamMember);
    }
}

[HttpGet]
[Route("/members/{memberId}/team")]
public IActionResult GetTeamForMember(Guid memberId)
{
    var teamId = GetTeamIdForMember(memberId);
    if (teamId != Guid.Empty)
    {
        return this.Ok(new {TeamID = teamId});
    }

    else
    {
        return this.NotFound();
    }
}

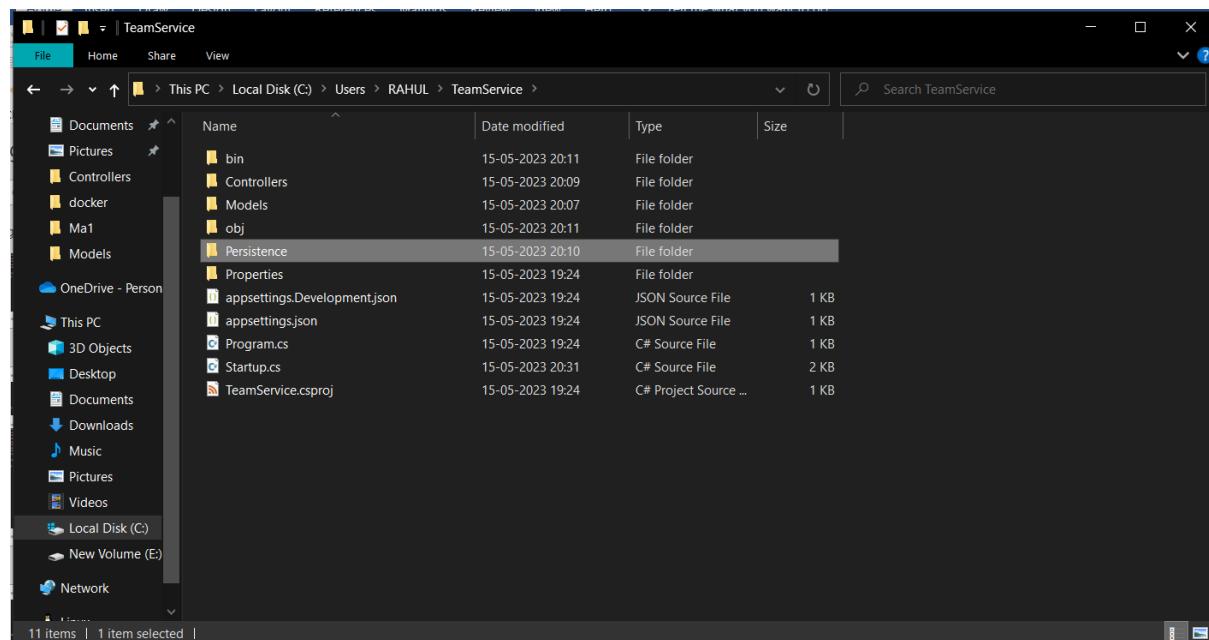
private Guid GetTeamIdForMember(Guid memberId)
{
}

```

```
foreach (var team in repository.List())
{
    var member = team.Members.FirstOrDefault(m => m.ID ==
memberId);
    if(member != null)
    {
        return team.ID;
    }
}

return Guid.Empty;
}
}
```

Step 4: Create folder “C:\Users\RAHUL\TeamService\Persistence”.



Step 5: Add file ITeamRepository.cs in “C:\Users\RAHUL\TeamService\Persistence’ folder.

```
using System;
using System.Collections.Generic;
using TeamService.Models;

namespace TeamService.Persistence
{
    public interface ITeamRepository
```

```
{  
    IEnumerable<Team> List();  
    Team Get(Guid id);  
    Team Add(Team team);  
    Team Update(Team team);  
    Team Delete(Guid id);  
}  
}
```

Step 6: Add MemoryTeamRepository.cs in “C:\Users\RAHUL\TeamService\Persistence” folder.

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using TeamService;  
using TeamService.Models;  
  
namespace TeamService.Persistence  
{  
    public class MemoryTeamRepository : ITeamRepository  
    {  
        protected static ICollection<Team> teams;  
        public MemoryTeamRepository()  
        {  
            if(teams == null)  
            {  
                teams = new List<Team>();  
            }  
        }  
  
        public MemoryTeamRepository(ICollection<Team> teams)  
        {  
            MemoryTeamRepository.teams = teams;  
        }  
  
        public IEnumerable<Team> List()  
        {  
            return teams;  
        }  
        public Team Get(Guid id)  
        {  
            return teams.FirstOrDefault(t => t.ID == id);  
        }  
  
        public Team Update(Team t)
```

```
{  
    Team team = this.Delete(t.ID);  
    if(team != null)  
    {  
        team = this.Add(t);  
    }  
  
    return team;  
}  
  
public Team Add(Team team)  
{  
    teams.Add(team);  
    return team;  
}  
  
public Team Delete(Guid id)  
{  
    var q = teams.Where(t => t.ID == id);  
    Team team = null;  
    if(q.Count() > 0)  
    {  
        team = q.First();  
        teams.Remove(team);  
    }  
  
    return team;  
}  
}  
}
```

Step 7: Add following line to Startup.cs in public void ConfigureServices(IServiceCollection services) method.

```
services.AddScoped<ITeamRepository, MemoryTeamRepository>();
```

Step 8: Now open two command prompts to run this project.

Step 9: On Command prompt 1 (go inside folder teamservice first)

Commands:

Dotnet run:

```
C:\Windows\System32\cmd.exe - dotnet run
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\RAHUL\TeamService>dotnet run
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\RAHUL\TeamService
```

Step 10: On Command prompt 2

Command: To get all teams

`curl --insecure https://localhost:5001/teams`

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\RAHUL\TeamService>curl --insecure https://localhost:5001/teams
[]
```

Step 11: On Command prompt 2

Command: To create new team

`curl --insecure -H "Content-Type:application/json" -X POST -d "{\"id\":\"e52baa63-d511-417e-9e54-7aab04286281\", \"name\":\"KC\"}" https://localhost:5001/teams`

```
C:\Users\RAHUL\TeamService>curl --insecure -H "Content-Type:application/json" -X POST -d "{\"id\":\"e52baa63-d511-417e-9e54-7aab04286281\", \"name\":\"KC\"}" https://localhost:5001/teams
[]
```

Step 12: On Command prompt 2

Command: To create one more new team

`curl --insecure -H "Content-Type:application/json" -X POST -d "{\"id\":\"e12baa63-d511-417e-9e54-7aab04286281\", \"name\":\"MSC Part1\"}" https://localhost:5001/teams`

```
C:\Users\RAHUL\TeamService>curl --insecure -H "Content-Type:application/json" -X POST -d "{\"id\":\"e12baa63-d511-417e-9e54-7aab04286281\", \"name\":\"MSC Part1\"}" https://localhost:5001/teams
[]
```

Step 13: On Command prompt 2

Command: To get all teams

```
curl --insecure https://localhost:5001/teams
```

```
C:\Users\RAHUL\TeamService>curl --insecure https://localhost:5001/teams
[{"name": "KC", "id": "e52baa63-d511-417e-9e54-7aab04286281", "members": []}, {"name": "MSC Part1", "id": "e12baa63-d511-417e-9e54-7aab04286281", "members": []}]
```

Step 14: On Command prompt 2

Command: To get single team with team-id as parameter

```
curl --insecure https://localhost:5001/teams/e52baa63-d511-417e-9e54-7aab04286281
```

```
C:\Users\RAHUL\TeamService>curl --insecure https://localhost:5001/teams/e52baa63-d511-417e-9e54-7aab04286281
{"name": "KC", "id": "e52baa63-d511-417e-9e54-7aab04286281", "members": []}
C:\Users\RAHUL\TeamService>
```

Step 15: On Command prompt 2

Command: To update team details (change name of first team from “KC” to “KC IT DEPT”)

```
curl --insecure -H "Content-Type:application/json" -X PUT -d "{\"id\":\"e52baa63-d511-417e-9e54-7aab04286281\", \"name\":\"KC IT DEPT\"}"
```

```
https://localhost:5001/teams/e52baa63-d511-417e-9e54-7aab04286281
```

```
C:\Users\RAHUL\TeamService>curl --insecure -H "Content-Type:application/json" -X PUT -d "{\"id\": \"e52baa63-d511-417e-9e54-7aab04286281\", \"name\": \"KC IT DEPT\"}" https://localhost:5001/teams/e52baa63-d511-417e-9e54-7aab04286281
{"name": "KC IT DEPT", "id": "e52baa63-d511-417e-9e54-7aab04286281", "members": []}
```

Step 16: On command prompt 2

Command: To delete team

```
curl --insecure -H "Content-Type:application/json" -X DELETE
```

```
https://localhost:5001/teams/e52baa63-d511-417e-9e54-7aab04286281
```

```
C:\Users\RAHUL\TeamService>curl --insecure -H "Content-Type:application/json" -X DELETE https://localhost:5001/teams/e52baa63-d511-417e-9e54-7aab04286281
C:\Users\RAHUL\TeamService>
```

Step 17: On Command prompt 2

Command: Confirm with get all teams now it shows only one team (first one is deleted)

```
curl -insecure https://localhost:5001/teams
```

```
C:\Users\RAHUL\TeamService>curl --insecure https://localhost:5001/teams
[{"name": "MSC Part1", "id": "e12baa63-d511-417e-9e54-7aab04286281", "members": []}]
C:\Users\RAHUL\TeamService>
```

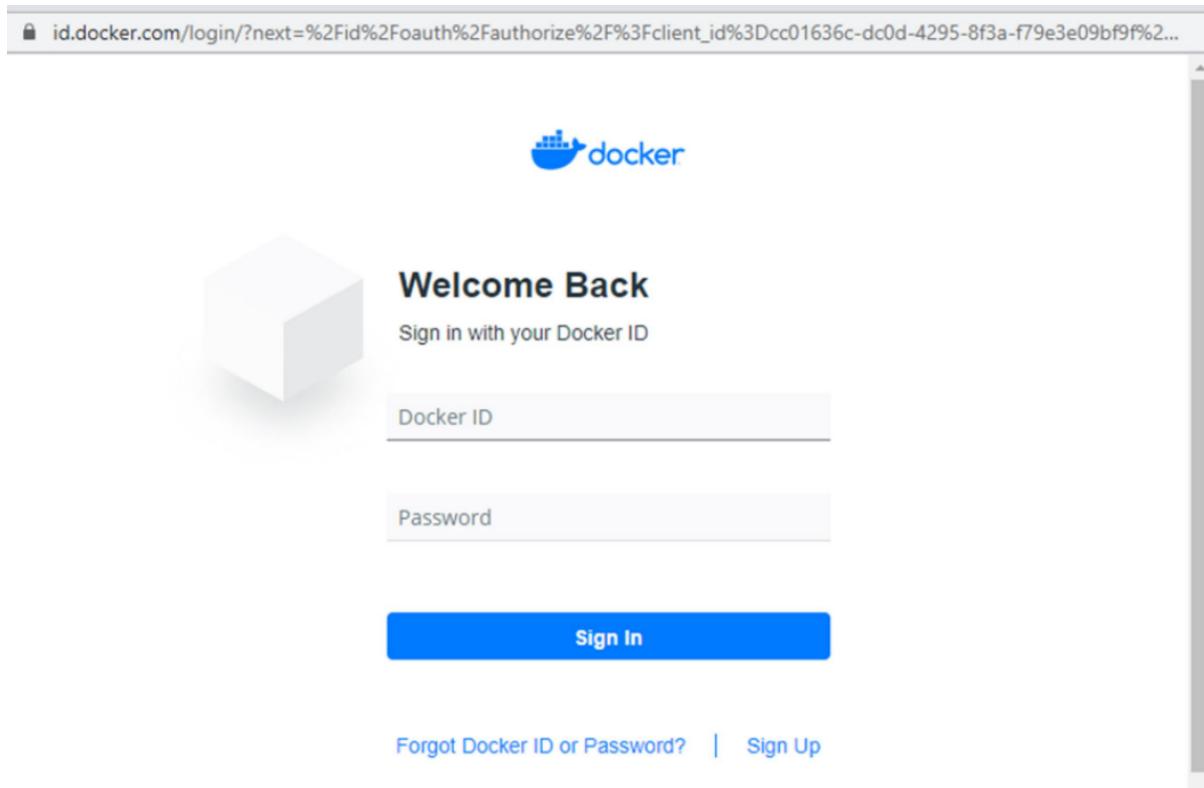
Practical 7

Aim: Running Location Service in Docker

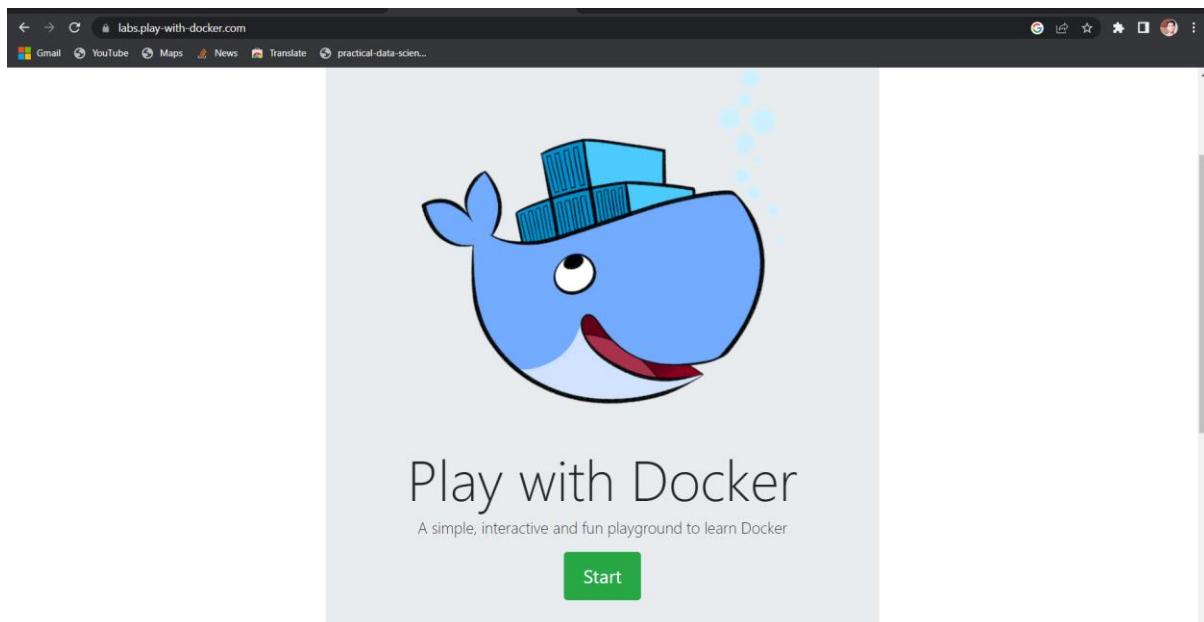
Writeup:

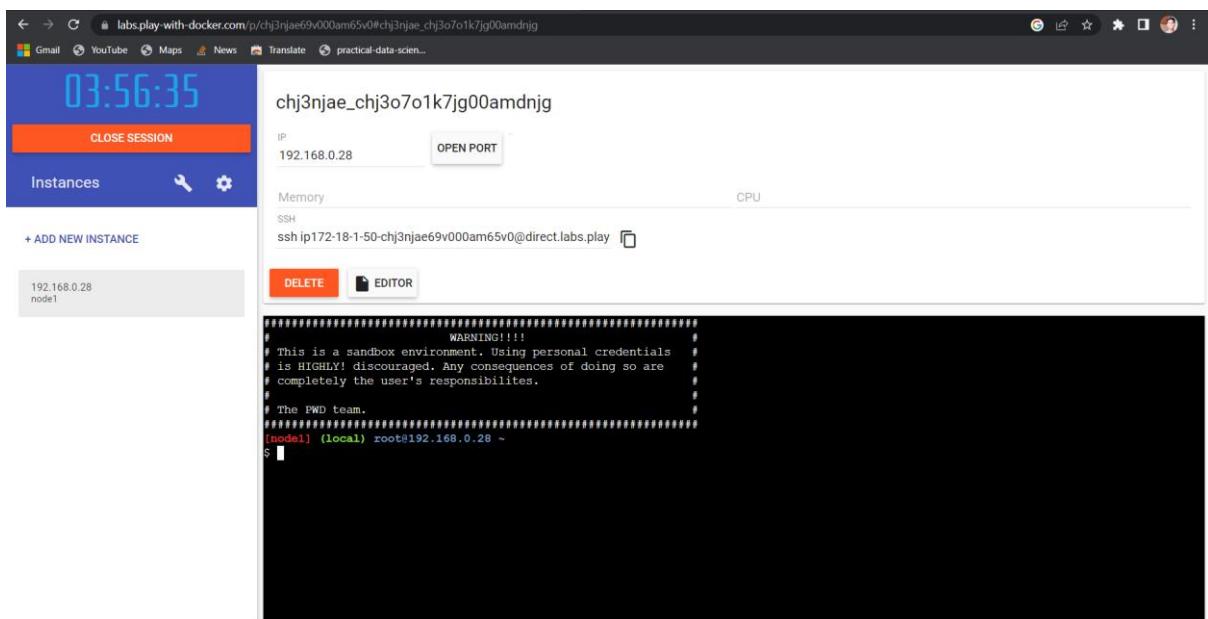
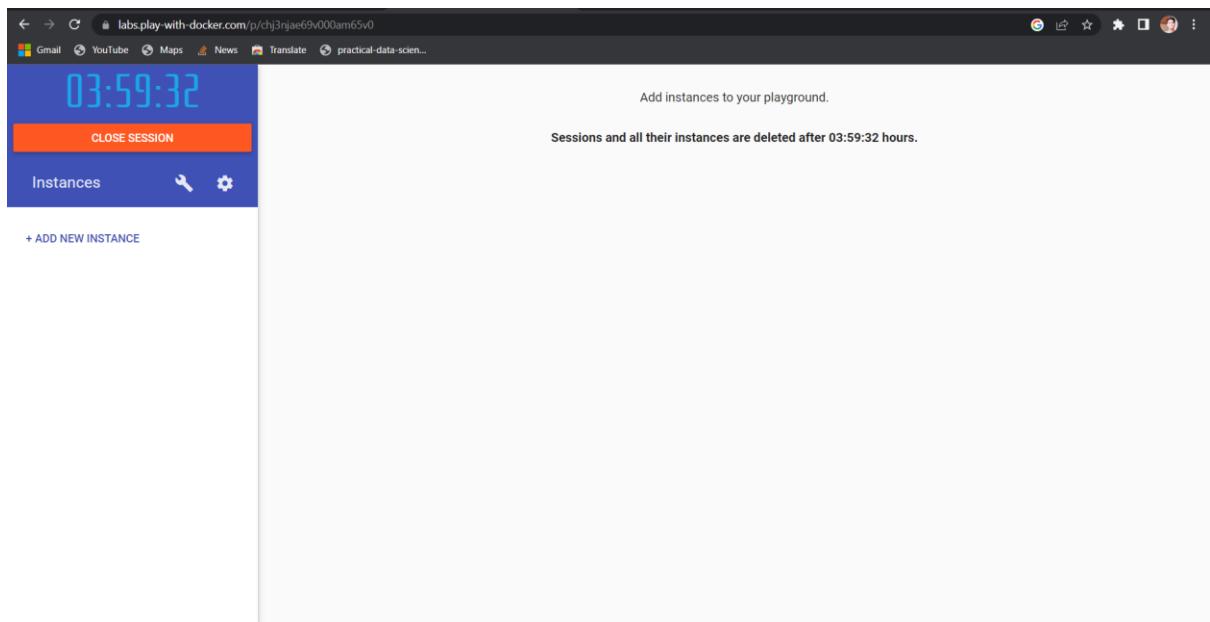
Step 1: create docker hub login first to use it in play with docker

Now login in to Play-With-Docker



Step 2: Click on Start



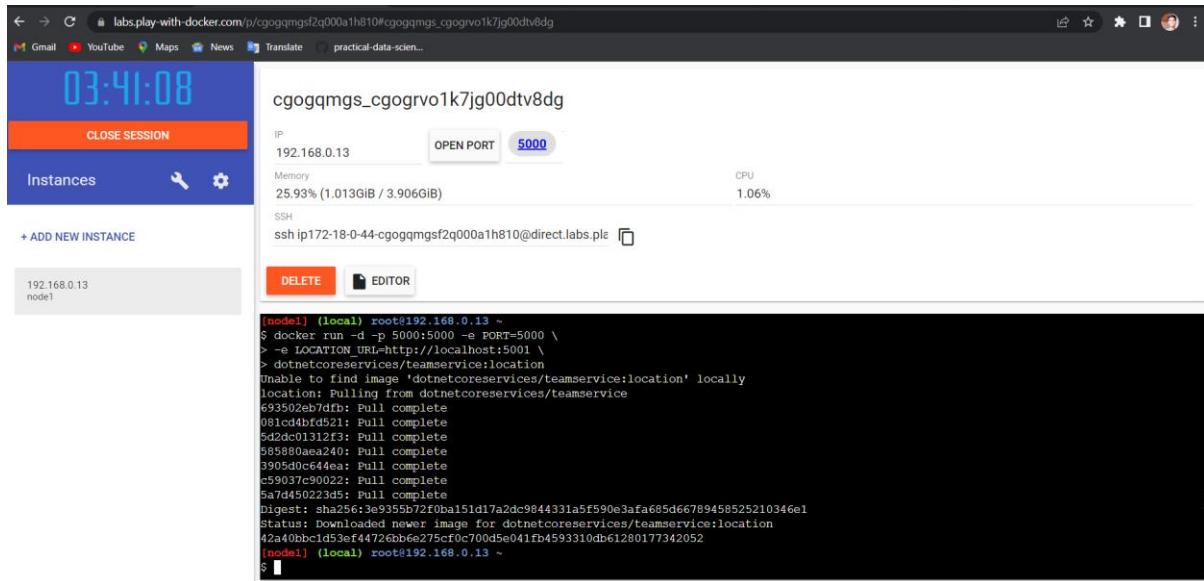
Step 3: Click on Add New Instance

Step 3: Start typing following commands

Command : To run teamservice

```
docker run -d -p 5000:5000 -e PORT=5000 \
-e LOCATION_URL=http://localhost:5001 \
dotnetcoreservices/teamservice:location
```

Output:



The screenshot shows a browser window with the URL https://labs.play-with-docker.com/p/cgogqmgsf2q000a1h810#cgogqmgs_cgogrvo1k7jg00dtv8dg. The page displays a terminal session with the following content:

```
[node1] (local) root@192.168.0.13 ~
$ docker run -d -p 5000:5000 -e PORT=5000 \
> -e LOCATION_URL=http://localhost:5001 \
> dotnetcoreservices/teamservice:location
Unable to find image 'dotnetcoreservices/teamservice:location' locally
location: Pulling from dotnetcoreservices/teamservice
693502eb7dfb: Pull complete
081cd4bfbd521: Pull complete
5d2dc01312f3: Pull complete
585880aea240: Pull complete
3905d0c644ea: Pull complete
c59037c90022: Pull complete
5a7d450223d5: Pull complete
Digest: sha256:3e935b72f0ba151d17a2dc9844331a5f590e3afa685d66789458525210346e1
Status: Downloaded newer image for dotnetcoreservices/teamservice:location
42a40bbc1d53ef44726b6e275cf0c700d5e041fb4593310db61280177342052
[node1] (local) root@192.168.0.13 ~
$
```

Step 4:

Command: to run location service

```
docker run -d -p 5001:5001 -e PORT=5001 \
dotnetcoreservices/locationservice:nodb
```

output: (now it has started one more port that is 5001 for location service)

```

42a40bbc1d53ef44726bb6e275cf0c700d5e041fb4593310db61280177342052
[node1] (local) root@192.168.0.13 ~
$ docker run -d -p 5001:5001 -e PORT=5001 \
> dotnetcoreservices/locationservice:nodb
Unable to find image 'dotnetcoreservices/locationservice:nodb' locally
nodb: Pulling from dotnetcoreservices/locationservice
693502eb7dfb: Already exists
081cd4bfdf521: Already exists
5d2dc01312f3: Already exists
585880aea240: Already exists
3905d0c644ea: Already exists
c59037c90022: Already exists
dbc03883a4ca: Pull complete
Digest: sha256:15f7aca33c5e2117e04f58a59e0cf96fd20d5cbf2cf66c3cd708118d573255168
Status: Downloaded newer image for dotnetcoreservices/locationservice:nodb
ab1837840015fafecc00ced52971daa8b14ea9a6d29c87b6a669b05e94aeb24
[node1] (local) root@192.168.0.13 ~

```

Step 4:**Command : to check running images in docker**

Docker images

Output:

```

42a40bbc1d53ef44726bb6e275cf0c700d5e041fb4593310db61280177342052
[node1] (local) root@192.168.0.13 ~
$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
dotnetcoreservices/teamservice   location  b27d0de8f2de  6 years ago  880MB
dotnetcoreservices/locationservice   nodb     03339f0ea9dd  6 years ago  883MB
[node1] (local) root@192.168.0.13 ~

```

Step 5:**Command: to create new team**

```
curl -H "Content-Type:application/json" -X POST -d \'{"id":"e52baa63-d511-417e-9e54-7aab04286281", "name":"KC"}' http://localhost:5000/teams
```

Output:

```

$ curl -H "Content-Type:application/json" -X POST -d \
> '{"id":"e52baa63-d511-417e-9e54-7aab04286281","name":"KC"}' http://localhost:5000/teams
{"name":"KC","id":"e52baa63-d511-417e-9e54-7aab04286281","members":[]}[node1] (local) root@192.168.0.28 ~
$ curl http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281
{"name":"KC","id":"e52baa63-d511-417e-9e54-7aab04286281","members":[]}[node1] (local) root@192.168.0.28 ~

```

Step 6: To confirm that team is added

Command : curl http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281

Output:

```
$ curl -H "Content-Type:application/json" -X POST -d \
> '{"id":"e52baa63-d511-417e-9e54-7aab04286281","name":"KC"}' http://localhost:5000/teams
{"name":"KC","id":"e52baa63-d511-417e-9e54-7aab04286281","members":[]}[node1] (local) root@192.168.0.28 ~
$ curl http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281
{"name":"KC","id":"e52baa63-d511-417e-9e54-7aab04286281","members":[]}[node1] (local) root@192.168.0.28 ~
```

Step 7:

Command : to add new member to team

```
curl -H "Content-Type:application/json" -X POST -d \ '{"id":"63e7acf8-8fae-42ce-9349-3c8593ac8292", "firstName":"Rahul", "lastName":"Kewat"}'
```

<http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281/members>

Output:

```
$ curl -H "Content-Type:application/json" -X POST -d \
> '{"id":"63e7acf8-8fae-42ce-9349-3c8593ac8292", "firstName":"Rahul", "lastName":"Kewat"}' http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281/members
{"teamID":"e52baa63-d511-417e-9e54-7aab04286281", "memberID":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}[node1] (local) root@192.168.0.28 ~
```

Step 8:

Command: to confirm member added

<http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281>

Output:

```
$ curl http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281
{"name":"KC","id":"e52baa63-d511-417e-9e54-7aab04286281","members": [{"id":"63e7acf8-8fae-42ce-9349-3c8593ac8292", "firstName":"Rahul", "lastName":"Kewat"}]}[node1] (local) root@192.168.0.28 ~
```

Step 9:

Command: To add location for member

```
curl -H "Content-Type:application/json" -X POST -d \ {"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f", "latitude":12.0,"longitude":12.0,"altitude":10.0,"timestamp":0,"memberId":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}' http://localhost:5001/locations/63e7acf8-8fae42ce9349-3c8593ac8292
```

Output:

```
$ curl -H "Content-Type:application/json" -X POST -d \
> '{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f", "latitude":12.0,"longitude":12.0,"altitude":10.0,"timestamp":0,"memberId":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}' http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292
{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f", "latitude":12.0,"longitude":12.0,"altitude":10.0,"timestamp":0,"memberID":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}[node1] (local) root@192.168.0.28 ~
```

Step 10:**Command : To confirm location is added in member****curl http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292****output:**

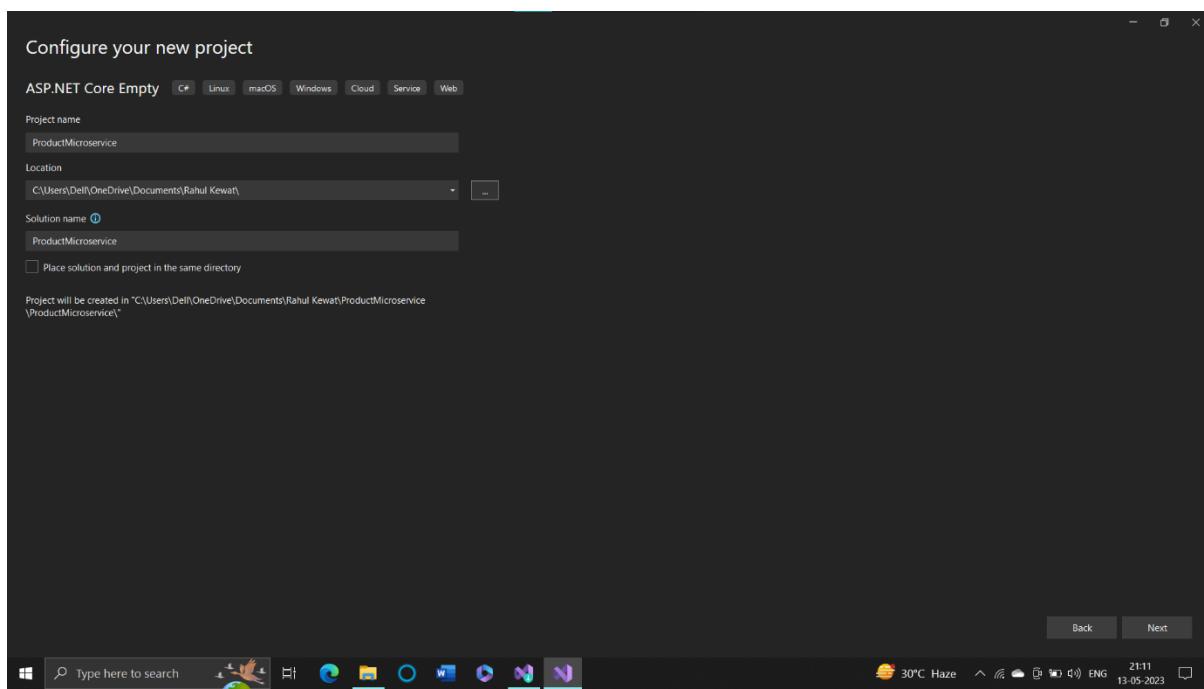
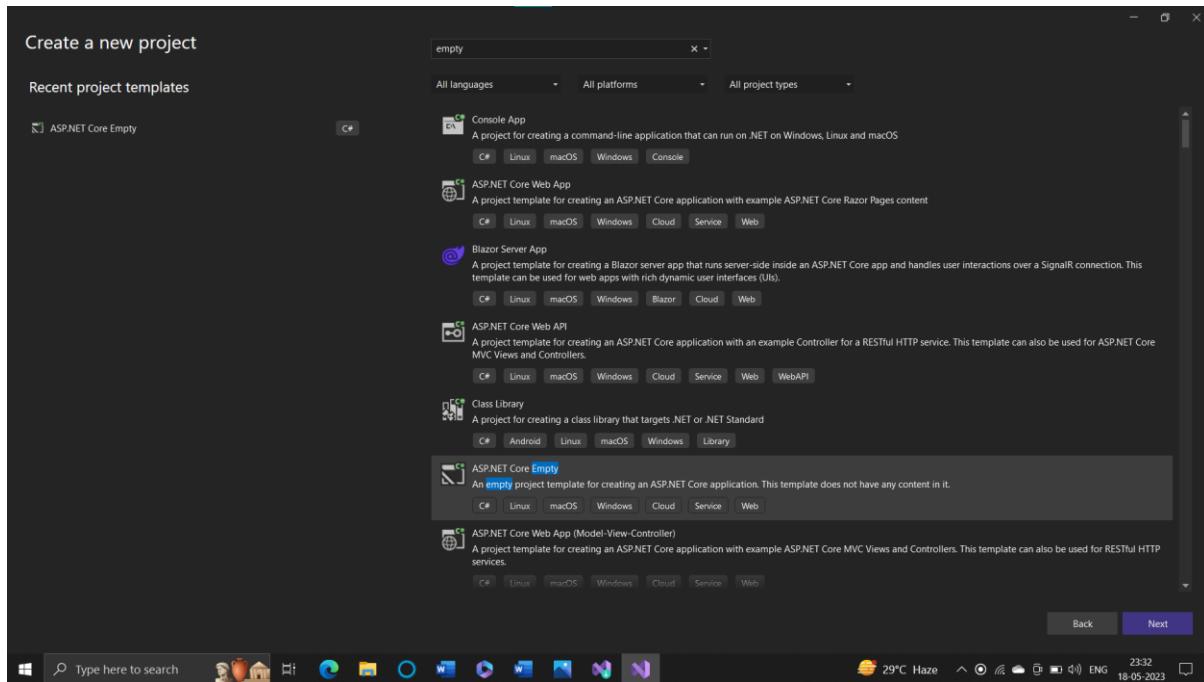
```
$ curl http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292
[{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0elf","latitude":12.0,"longitude":12.0,"timestamp":0,"memberID":"63e7acf8-8fae-
42ce-9349-3c8593ac8292"}] [node1] (local) root@192.168.0.13 ~
$ █
```

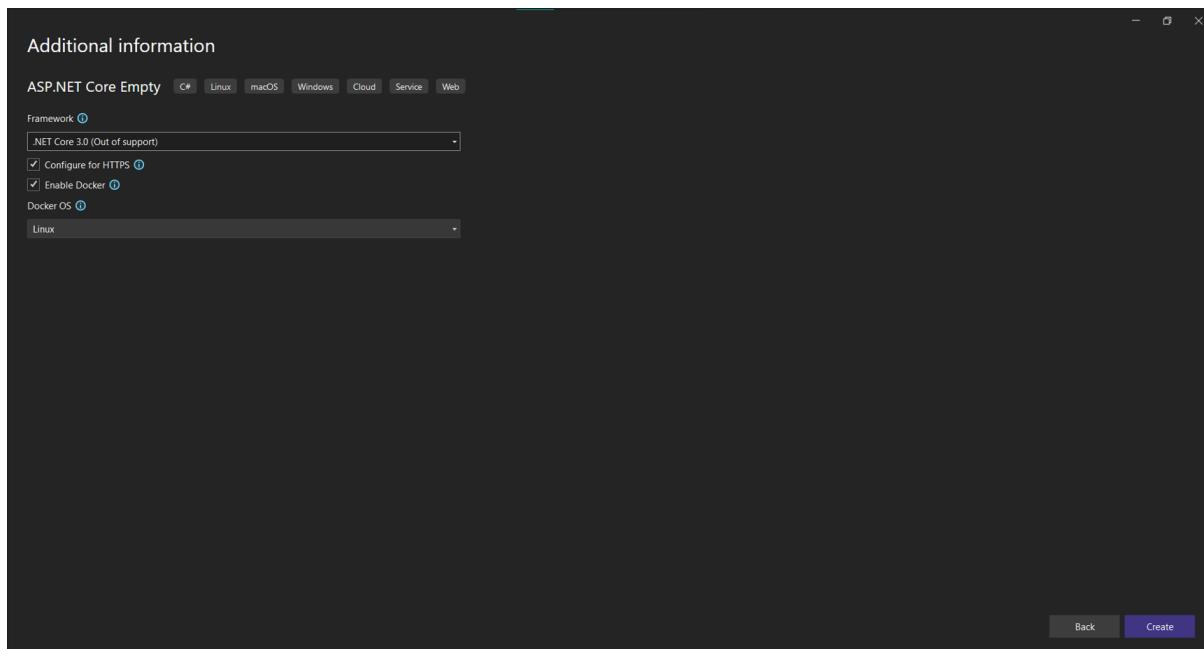
Practical 8

Aim: Building real-time Microservice with ASP.NET Core.

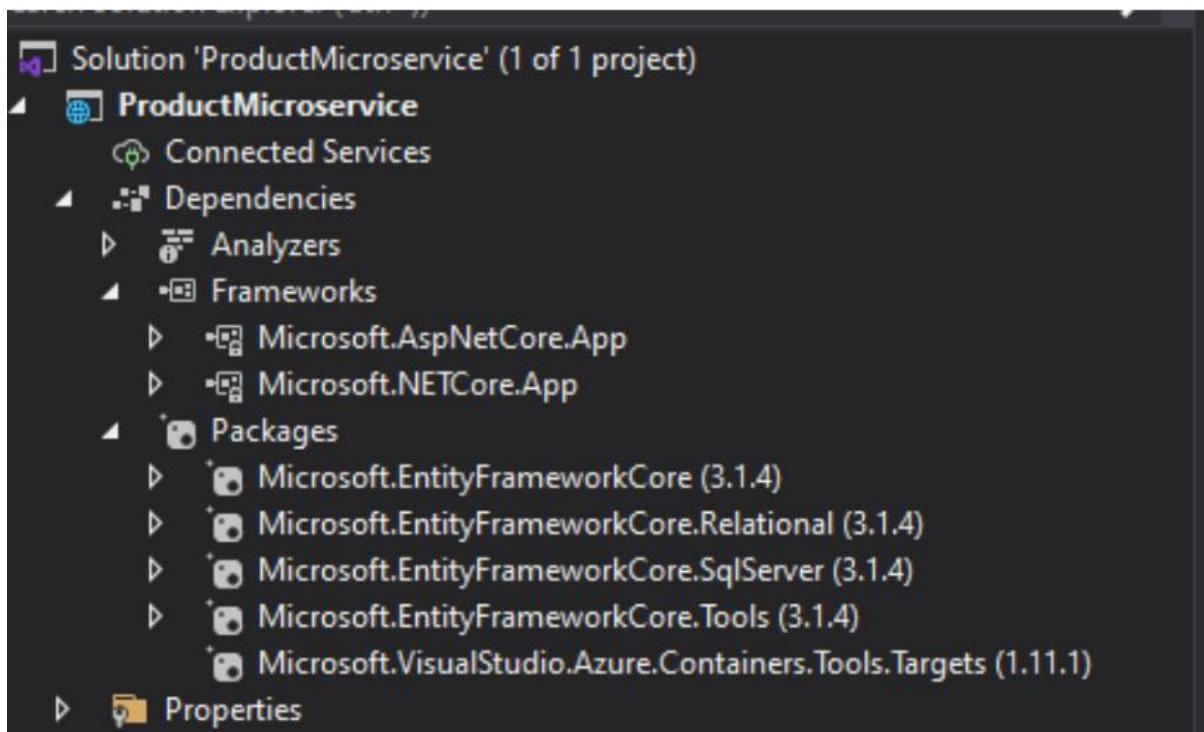
Writeup:

Step 1: Create and configure a new empty project

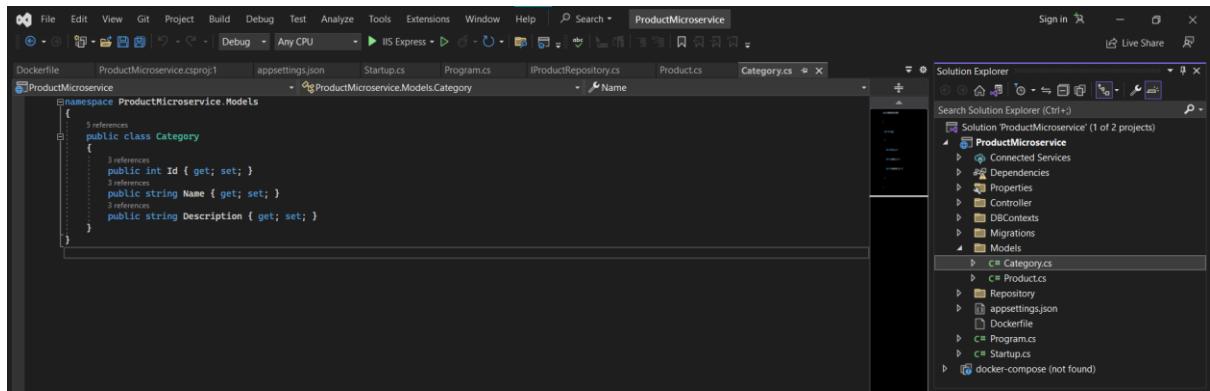




Step 2: Make sure to add all these packages via nugget package manager



Step 3: Create the Category Model, which will eventually be Category Table

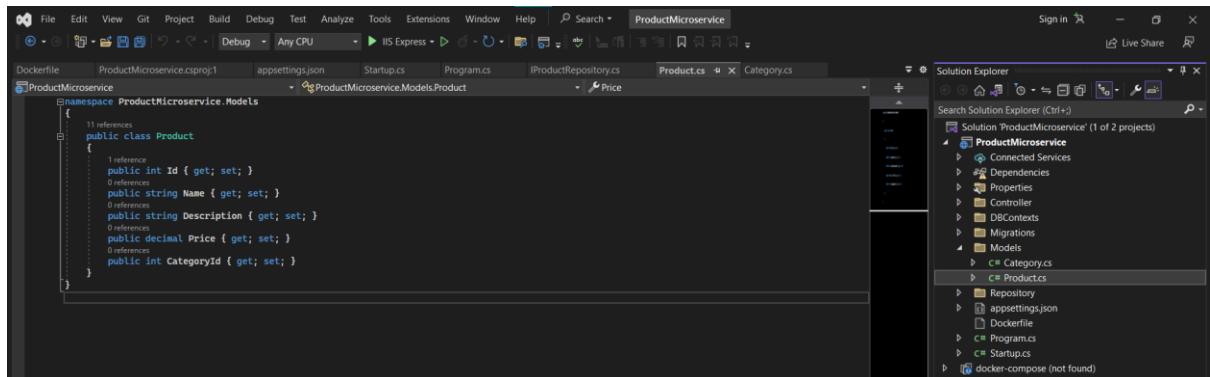


```

namespace ProductMicroservice.Models
{
    public class Category
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
    }
}

```

Step 4: Create the Product Model which will eventually be the Product Table

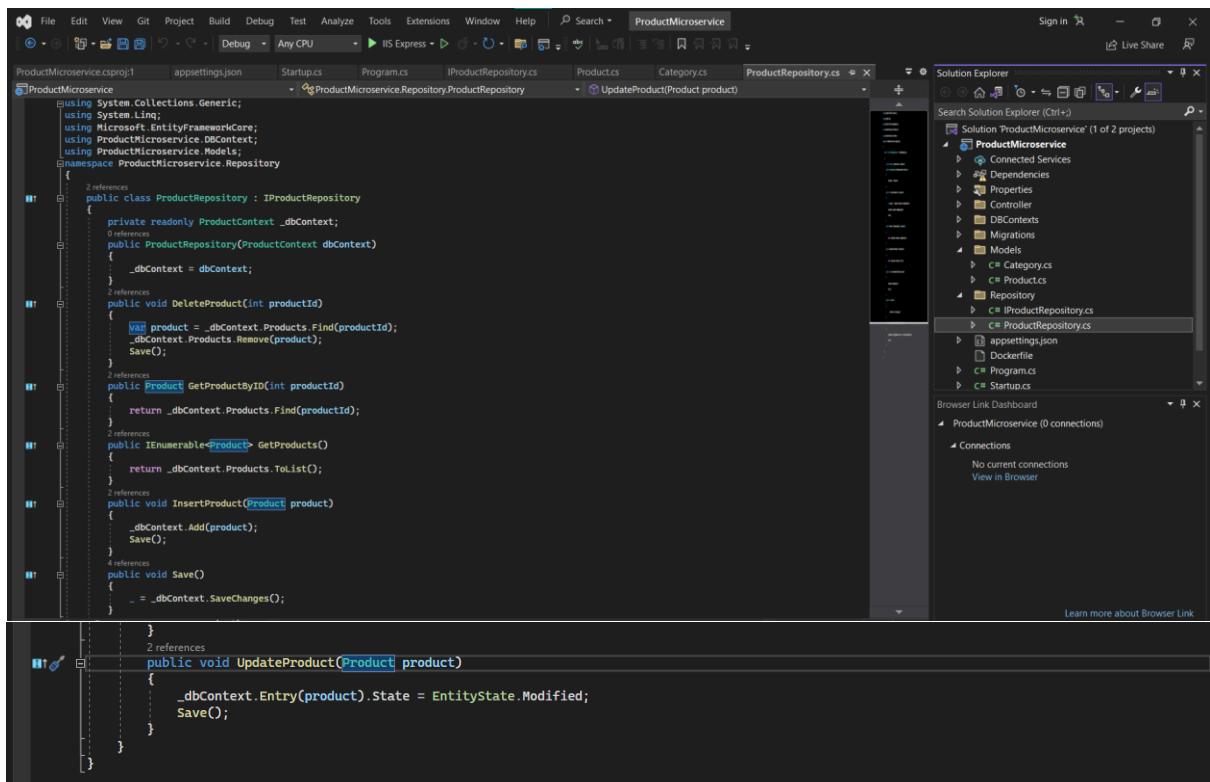


```

namespace ProductMicroservice.Models
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public decimal Price { get; set; }
        public int CategoryId { get; set; }
    }
}

```

Step 5: Create a ProductRepository for all our operations



```

using System;
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;
using ProductMicroservice.DBContext;
using ProductMicroservice.Models;
using ProductMicroservice.Repository;

public class ProductRepository : IProductRepository
{
    private readonly ProductContext _dbContext;
    public ProductRepository(ProductContext dbContext)
    {
        _dbContext = dbContext;
    }

    public void DeleteProduct(int productId)
    {
        var product = _dbContext.Products.Find(productId);
        _dbContext.Products.Remove(product);
        Save();
    }

    public Product GetProductByID(int productId)
    {
        return _dbContext.Products.Find(productId);
    }

    public IEnumerable<Product> GetProducts()
    {
        return _dbContext.Products.ToList();
    }

    public void InsertProduct(Product product)
    {
        _dbContext.Add(product);
        Save();
    }

    public void Save()
    {
        _dbContext.SaveChanges();
    }

    public void UpdateProduct(Product product)
    {
        _dbContext.Entry(product).State = EntityState.Modified;
        Save();
    }
}

```

Step 6: Create an interface IProductRepository to access the ProductRepository

```

using System.Collections.Generic;
using ProductMicroservice.Models;

namespace ProductMicroservice.Repository
{
    public interface IProductRepository
    {
        IEnumerable<Product> GetProducts();
        Product GetProductByID(int ProductId);
        void InsertProduct(Product Product);
        void DeleteProduct(int ProductId);
        void UpdateProduct(Product Product);
        void Save();
    }
}

```

Step 7: Create a ProductContext to build the model for the database; this will be a Code First Database Approach

```

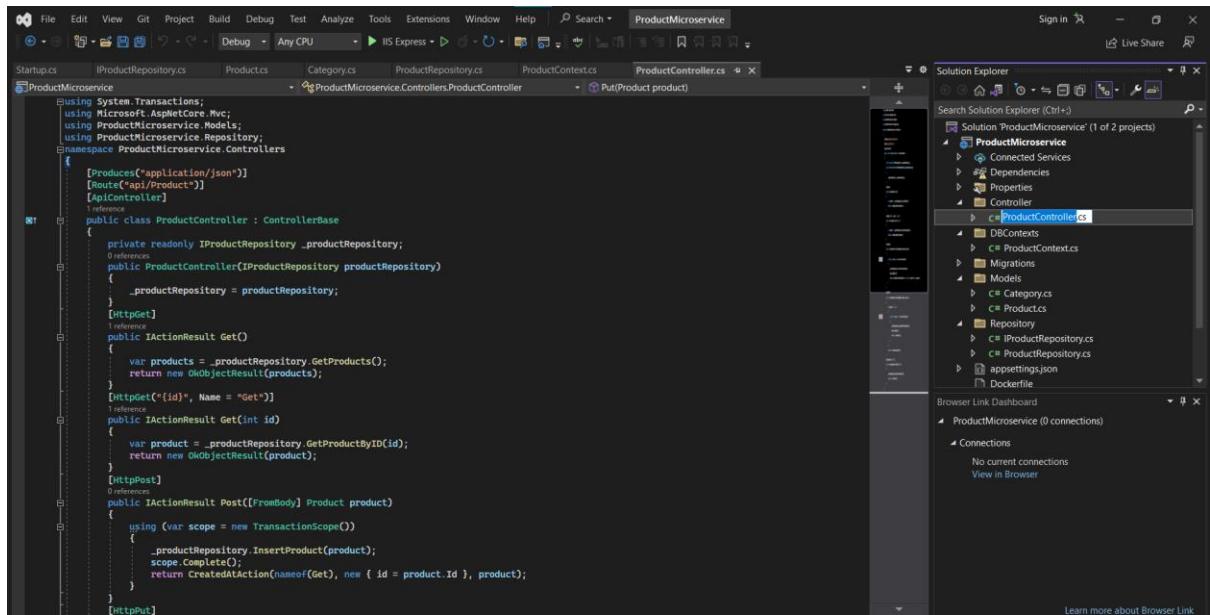
using Microsoft.EntityFrameworkCore;
using ProductMicroservice.Models;

namespace ProductMicroservice.DBContext
{
    public class ProductContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Category> Categories { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Category>().HasData(
                new Category
                {
                    Id = 1,
                    Name = "Electronics",
                    Description = "Electronic Items",
                },
                new Category
                {
                    Id = 2,
                    Name = "Clothes",
                    Description = "Dresses",
                },
                new Category
                {
                    Id = 3,
                    Name = "Grocery",
                    Description = "Grocery Items",
                }
            );
        }
    }
}

```

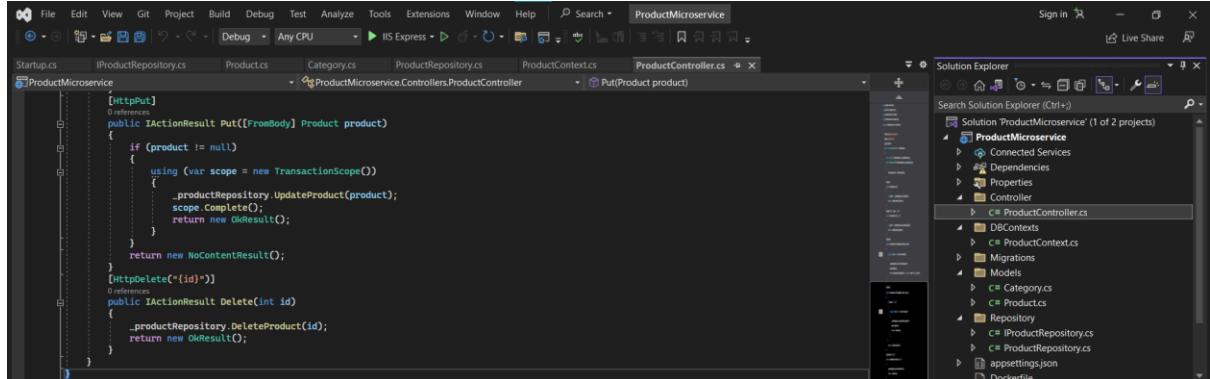
Step 8: Add a ProductController to get an endpoint for the APIs



```

using System;
using Microsoft.AspNetCore.Mvc;
using ProductMicroservice.Models;
using ProductMicroservice.Repository;
namespace ProductMicroservice.Controllers
{
    [Produces("application/json")]
    [Route("api/Products")]
    [ApiController]
    public class ProductController : ControllerBase
    {
        private readonly IProductRepository _productRepository;
        public ProductController(IProductRepository productRepository)
        {
            _productRepository = productRepository;
        }
        [HttpGet]
        [Route("")]
        public IActionResult Get()
        {
            var products = _productRepository.GetProducts();
            return new OkObjectResult(products);
        }
        [HttpGet("{id}")]
        public IActionResult Get(int id)
        {
            var product = _productRepository.GetProductByID(id);
            return new OkObjectResult(product);
        }
        [HttpPost]
        public IActionResult Post([FromBody] Product product)
        {
            using (var scope = new TransactionScope())
            {
                _productRepository.InsertProduct(product);
                scope.Complete();
                return CreatedAtAction(nameof(Get), new { id = product.Id }, product);
            }
        }
        [HttpPut]
    }
}

```

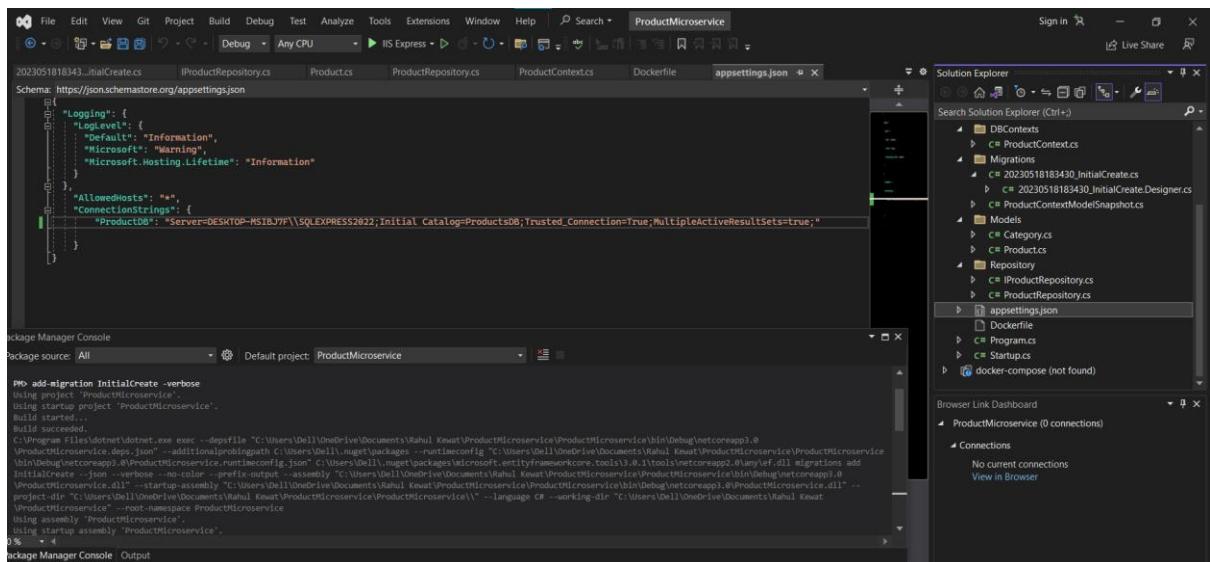



```

[HttpPut]
public IActionResult Put([FromBody] Product product)
{
    if (product != null)
    {
        using (var scope = new TransactionScope())
        {
            _productRepository.UpdateProduct(product);
            scope.Complete();
            return new OkResult();
        }
    }
    return new NoContentResult();
}
[HttpDelete("{id}")]
public IActionResult Delete(int id)
{
    _productRepository.DeleteProduct(id);
    return new OkResult();
}

```

Step 9: Add the connection string settings to connect the project to the SQLSERVER database. Once added run add-migration command to run the migration



appsettings.json:

```

{
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft": "Warning",
            "Microsoft.Hosting.Lifetime": "Information"
        }
    },
    "AllowedHosts": "*",
    "ConnectionStrings": {
        "ProductDB": "Server=DESKTOP-HSIBJ7F\\SQLEXPRESS2022;Initial Catalog=Products0;Trusted_Connection=True;MultipleActiveResultSets=true;"
    }
}

```

Package Manager Console Output:

```

PM> add-migration InitialCreate -verbose
Using project 'ProductMicroservice'.
Using startup project 'ProductMicroservice'.
Build started.
Build succeeded.
C:\Program Files\dotnet\dotnet.exe exec --depsfile "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice\ProductMicroservice\bin\Debug\netcoreapp3.0\ProductMicroservice.deps.json" --additionalprobingpath "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice\ProductMicroservice\bin\Debug\netcoreapp3.0\ProductMicroservice.runtimeconfig.json" "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice\bin\Debug\netcoreapp3.0\tools\netcoreapp2.0\anyef.dll" migrations add
InitialCreate -jms -non-interactive -v -f -o "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice\ProductMicroservice\bin\Debug\netcoreapp3.0\ProductMicroservice.dll" -startUpAssembly "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice\ProductMicroservice\bin\Debug\netcoreapp3.0\ProductMicroservice.dll" --projectDir "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice\ProductMicroservice" --language C# --working-dir "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice" --root-namespace ProductMicroservice
Using assembly 'ProductMicroservice'.
Using startup assembly 'ProductMicroservice'.
%> 4

```

Step 10: Run the update-database command to reflect the model changes to the database

The screenshot shows the Visual Studio interface. In the center, the `appsettings.json` file is open, displaying configuration settings for logging, allowed hosts, and connection strings. Below it, the `Package Manager Console` shows the command `PM> update-database -verbose` being run, followed by its execution output. To the right, the `Solution Explorer` pane lists project files like `ProductContext.cs`, `Migrations`, and `Models`. At the bottom right, the `Browser Link Dashboard` is visible.

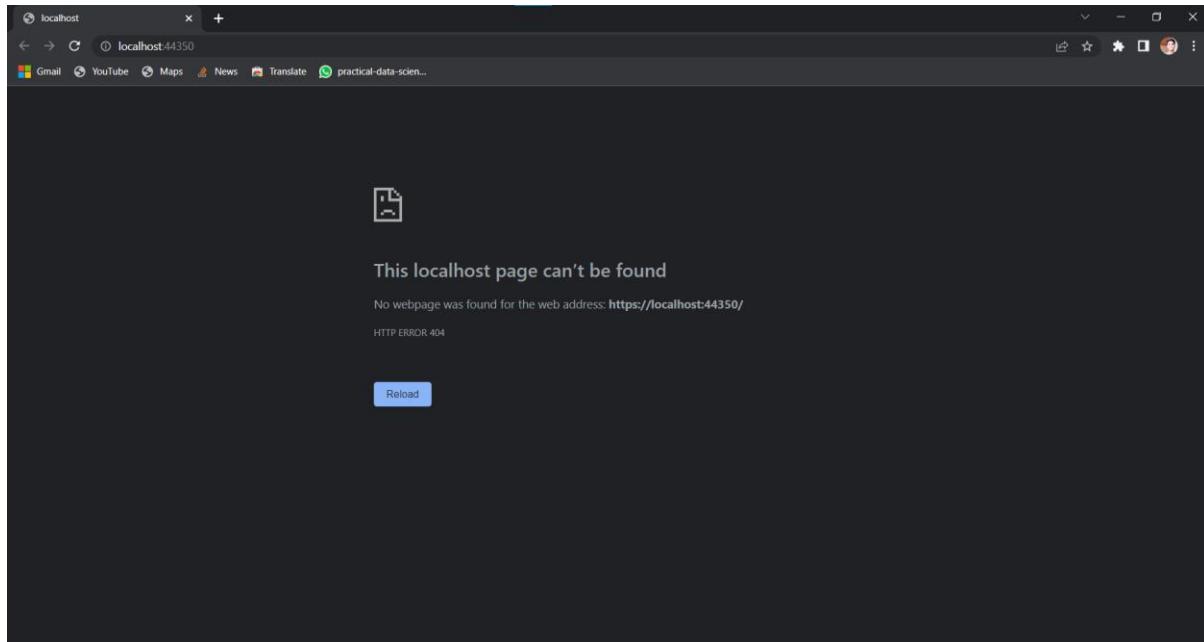
```

{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "ProductDB": "Server=DESKTOP-M5I8JYJF\\SQLEXPRESS022;Initial Catalog=ProductsDB;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}

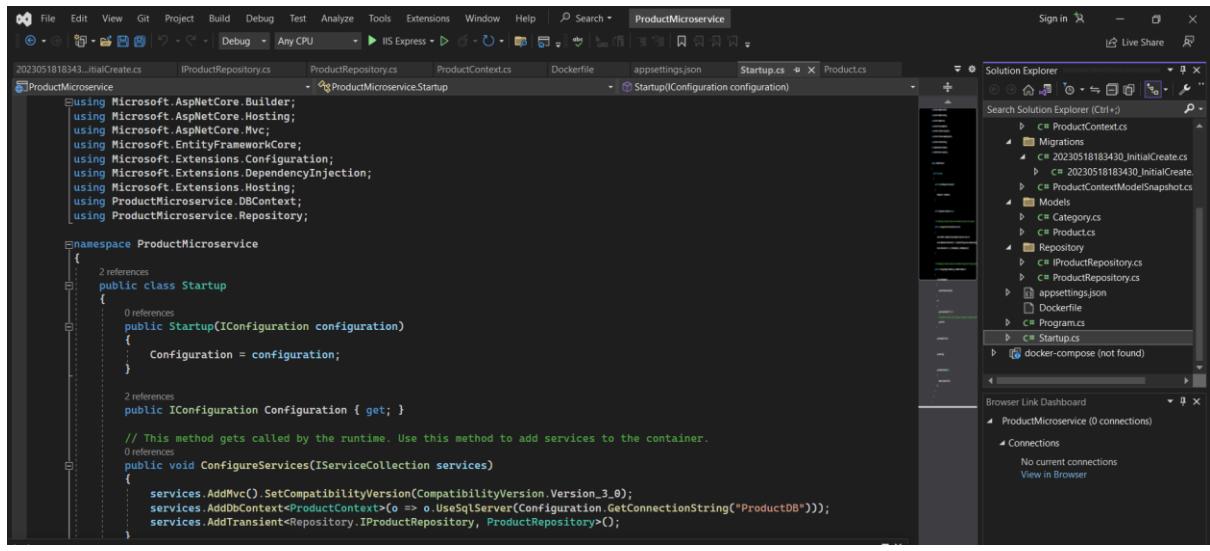
PM> update-database -verbose
Using project 'ProductMicroservice'.
Using startup project 'ProductMicroservice'.
Build started...
Build succeeded.
C:\Windows\Program Files\dotnet\dotnet.exe exec --depsfile "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice\ProductMicroservice\bin\Debug\netcoreapp3.0\productmicroservice.deps.json" --additionalprobingpath "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice" --runtimeconfig "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice\productmicroservice.runtimeconfig.json" "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice\bin\Debug\netcoreapp3.0\productmicroservice.dll" database update --
  -language C# --working-dir "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice" --assembly "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice\bin\Debug\netcoreapp3.0\productmicroservice.dll" --start-up-assembly "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice\bin\Debug\netcoreapp3.0\productmicroservice.dll" --project-dir "C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice" --root-namespace ProductMicroservice
Using assembly 'ProductMicroservice'.
Using startup assembly 'ProductMicroservice'.
Using application base 'C:\Users\DeLL\OneDrive\Documents\Rahul Kewat\ProductMicroservice\ProductMicroservice\bin\Debug\netcoreapp3.0'.
0% > 

```

Step 11: Try to run the project, you will see that a browser window opens with an error



Step 12: Make some modifications to the Startup.cs file accordingly for the Routing mechanism of the API to work



```

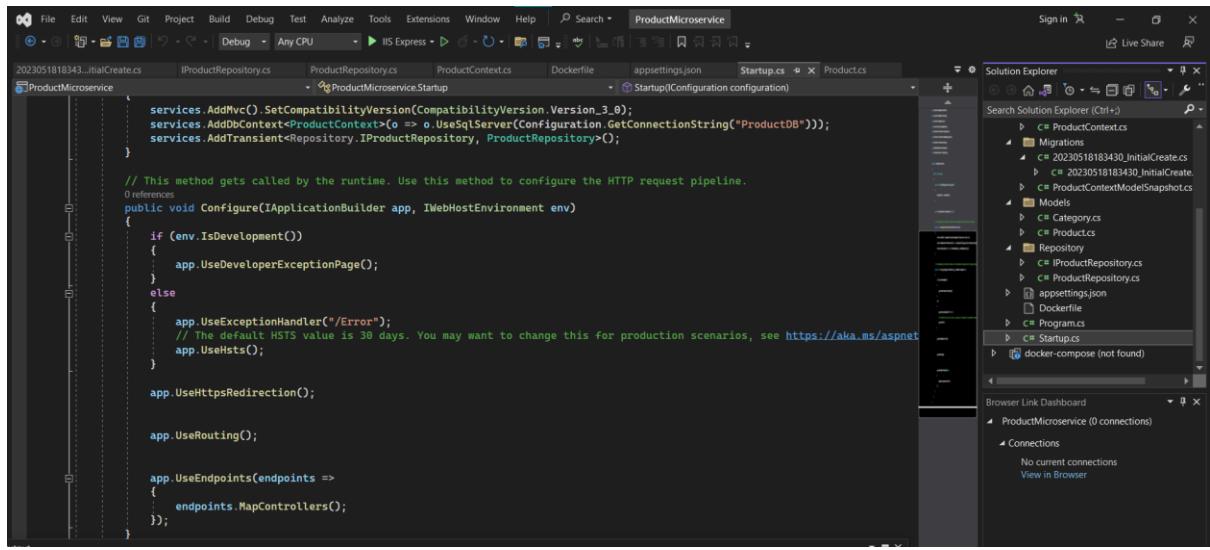
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using ProductMicroservice.DataContext;
using ProductMicroservice.Repository;

namespace ProductMicroservice
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_3_0);
            services.AddDbContext<ProductContext>(o => o.UseSqlServer(Configuration.GetConnectionString("ProductDB")));
            services.AddTransient<Repository.IProductRepository, ProductRepository>();
        }
    }
}

```



```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using ProductMicroservice.DataContext;
using ProductMicroservice.Repository;

namespace ProductMicroservice
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

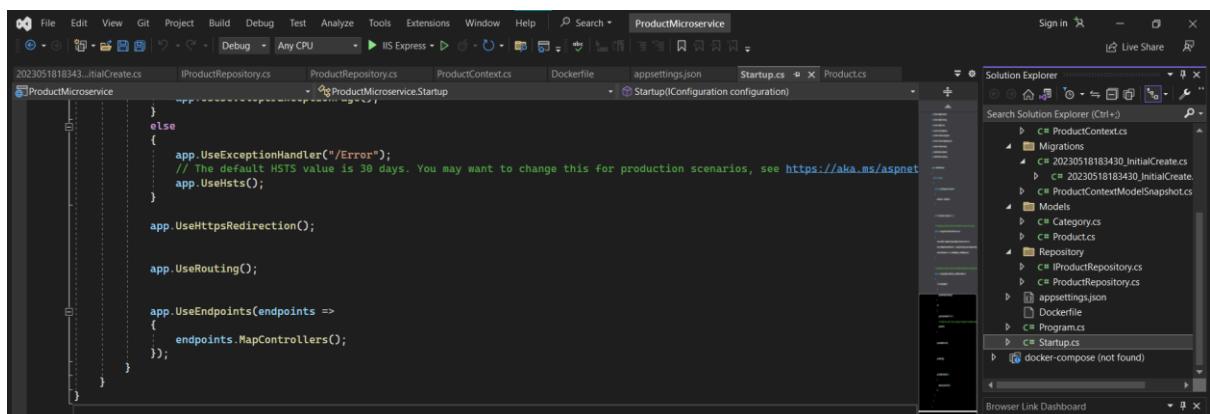
        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnet
                app.UseHsts();
            }

            app.UseHttpsRedirection();

            app.UseRouting();
        }

        public void ConfigureEndpoints(IEndpointRouteBuilder endpoints)
        {
            endpoints.MapControllers();
        }
    }
}

```



```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using ProductMicroservice.DataContext;
using ProductMicroservice.Repository;

namespace ProductMicroservice
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnet
                app.UseHsts();
            }

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
}

```

Step 13: Using Postman try to add a product into the product table via the api call

The screenshot shows the Postman interface. The URL is `https://localhost:44350/api/Product/`. The Body tab contains the following JSON payload:

```

1 "name": "Iphone",
2 ... "description": "Apple Mobile Phone",
3 ... "price": 6500.00,
4 ... "categoryId": 1
5
6
7
  
```

The response tab shows a successful `201 Created` status with the following JSON data:

```

1 {
2   "id": 1,
3   "name": "Iphone",
4   "description": "Apple Mobile Phone",
5   "price": 6500.00,
6   "categoryId": 1
7
  
```

Step 14: You will notice the SQL table to populate with that product

The screenshot shows the SSMS interface. The query window displays the following SQL code and result:

```

SELECT TOP 1000 [id]
      ,[Name]
      ,[Description]
      ,[Price]
      ,[CategoryId]
  FROM[ProductsDB].[dbo].[Products]
  
```

The results pane shows the following data:

	Id	Name	Description	Price	CategoryId
1	1	Iphone	Apple Mobile Phone	6500.00	1

A message at the bottom of the results pane says "Query executed successfully."

Step 15: Add another Product to test

The screenshot shows the Postman interface with a POST request to `https://localhost:44350/api/Product/`. The request body contains the following JSON:

```

1
2   ...
3     "name": "Tata Tiago",
4     "description": "Tata Motors",
5     "price": 500000,
6     "categoryId": 12
7

```

The response status is 201 Created, with a time of 179 ms and a size of 323 B. The response body is:

```

1
2   {
3     "id": 2,
4     "name": "Tata Tiago",
5     "description": "Tata Motors",
6     "price": 500000,
7     "categoryId": 12
8

```

Step 16: Get all the added products using GET

The screenshot shows the Postman interface with a GET request to `https://localhost:44350/api/Product/`. The request body is identical to the one in Step 15. The response status is 200 OK, with a time of 333 ms and a size of 365 B. The response body is:

```

1
2   [
3     {
4       "id": 1,
5       "name": "iPhone",
6       "description": "Apple Mobile Phone",
7       "price": 6500.00,
8       "categoryId": 1
9     },
10    {
11      "id": 2,
12      "name": "Tata Tiago",
13      "description": "Tata Motors",
14      "price": 500000.00,
15      "categoryId": 12
16    }
17

```

Step 17: Delete a product using DELETE

The screenshot shows the Postman interface with a DELETE request to `https://localhost:44350/api/Product`. The request body is set to `JSON` and contains the following JSON data:

```

1   {
2     "id":1,
3     "name":"iPhone",
4     "description":"Apple Mobile Phone",
5     "price":6599.99,
6     "categoryId":1
7   }
  
```

The response status is `405 Method Not Allowed`.

Step 18: Update a product using the PUT

The screenshot shows the Postman interface with a PUT request to `https://localhost:44350/api/Product`. The request body is set to `JSON` and contains the following JSON data:

```

1   {
2     "id":3,
3     "name":"Tata neon",
4     "description":"Tata Motors",
5     "price":150999.99,
6     "categoryId":1
7   }
  
```

The response status is `200 OK`.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'DESKTOP-MSIB7F\SQLEXPRESS2022'. The 'ProductsDB' database is selected, and the 'Tables' node is expanded, showing 'Products'. The 'Products' table is selected in the 'Tables' list. The 'SQLQuery2.sql' query window on the right contains the following SQL code:

```
SELECT TOP 1000 [Id]
    ,[Name]
    ,[Description]
    ,[Price]
    ,[CategoryId]
FROM[ProductsDB].[dbo].[Products]
```

The results pane below the query window displays the following data:

	Id	Description	Price	CategoryId
1	1	iPhone	6500.00	1
2	2	Tata Tiago	50000.00	12
3	3	Tata neon	150000.00	1
4	4	iPhone	7500.00	1

A status bar at the bottom indicates 'Query executed successfully.' and shows the system date and time as '14-05-2023 21:21'.

PRESENTATION

API Design for Microservices

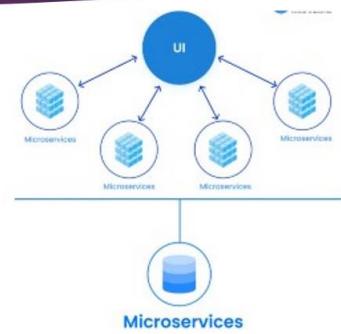
Prepared By: Rahul Kewat

Msc-IT Part-1

Roll No-22006

what are Microservices?

- Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs.
- Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features.



What Are API?

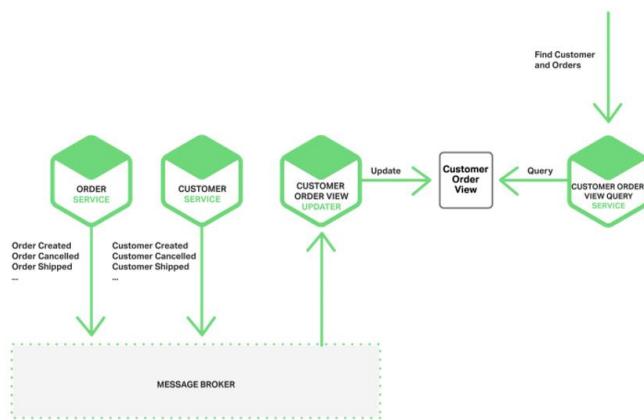
- An API is a system that allows two or more applications to communicate. Think of them as a set of procedures and functions, which allow the consumer to use the underlying service of an application.
- APIs are a part of microservices and help these services to communicate with each other.
- while communicating with other services, each service can have its own CRUD (create, read, Updated, delete) operations to store the relevant data in its database.

Introduction

- When considering microservice component boundaries, the source code itself is only part of our concern.
- Microservice components only become valuable when they can communicate with other components in the system.
- They each have an interface or API.
- We see two practices in crafting APIs for microservices worth mentioning here:
 - Message-oriented
 - Hypermedia-driven

Messsage-Oriented:

- Just as we work to write component code that can be safely refactored over time, we need to apply the same efforts to the shared interfaces between components.
- The most effective way to do this is to adopt a message -oriented implementation for microservice APIs.
- The notion of messaging as a way to share information between components dates back to the initial ideas about how object -oriented programming would work.



- All of the companies we talked with about microservice component design mentioned the notion of messaging as a key design practice.
- For example, Netflix relies on message formats like Avro, Protobuf, and Thrift over TCP/IP for communicating internally and JSON over HTTP for communicating to external consumers (e.g., mobile phones, browsers, etc.).
- By adopting a message-oriented approach, developers can expose general entry points into a component (e.g., an IP address and port number) and receive task-specific messages at the same time.

Hypermedia-Driven :

- Hypermedia API design is based on the way as that HTML works in a web browser and HTTP message are transmitted over the internet, it contains data and actions encoded in HTML format.
- Hypermedia provides links to navigate a workflow and template input to request information. In the same ways as we use the links to navigate the web and forms to provide the input.
- Hypermedia API message contains data and actions which provides necessary elements for it to work dynamically with client services and application.

- Example: When we use google chrome or other application on a laptop, our system processor makes a queue and it's works on the fundamentals of FIFO(First in first out). To run the application on the system, CPU allocate its process id and port number.
- HTTP message are sent to an Ip Address and a port number i.e. 80/ 443/88 and message contains the data and action which includes data in a HTML format.

Thank you