

MICROSERVICES ARCHITECTURE

A PRACTICAL REPORT
ON
MICROSERVICES ARCHITECTURE

SUBMITTED BY
Mr. MOHD KAIF
Roll No: 22001

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. MOHD KAIF 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	

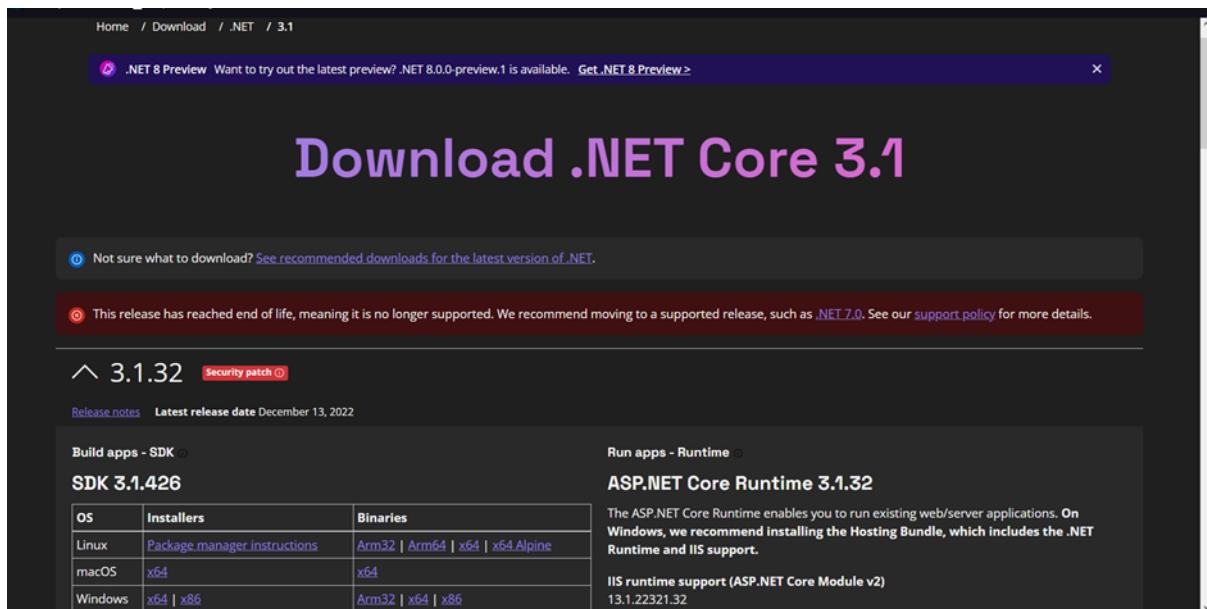
Practical 1

Aim: Creating a basic MVC application using .NET CORE

Writeup:

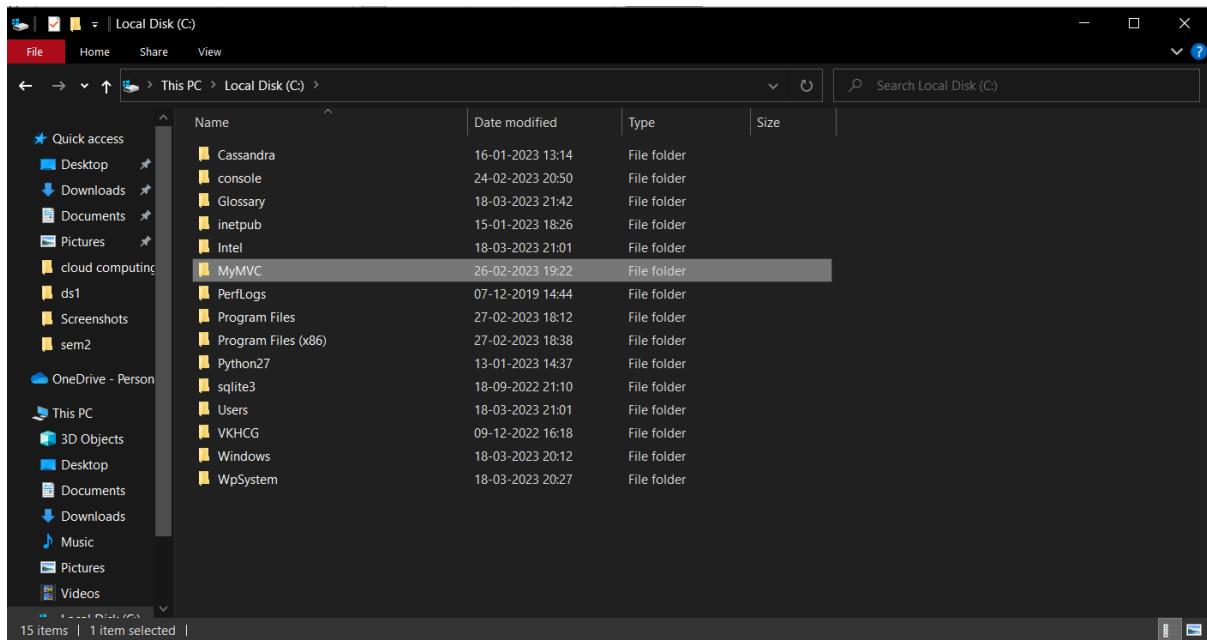
Step 1:

- Install .Net Core SDK.



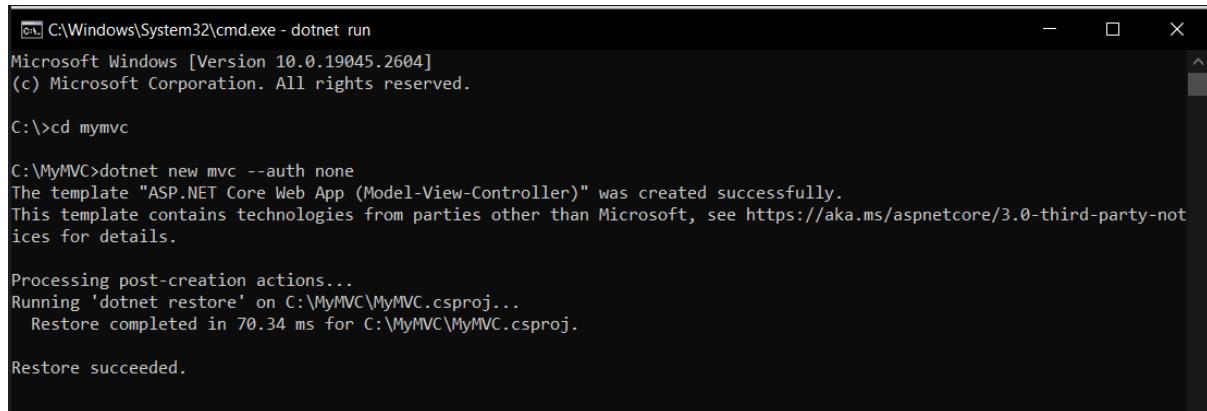
Step 2:

- 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

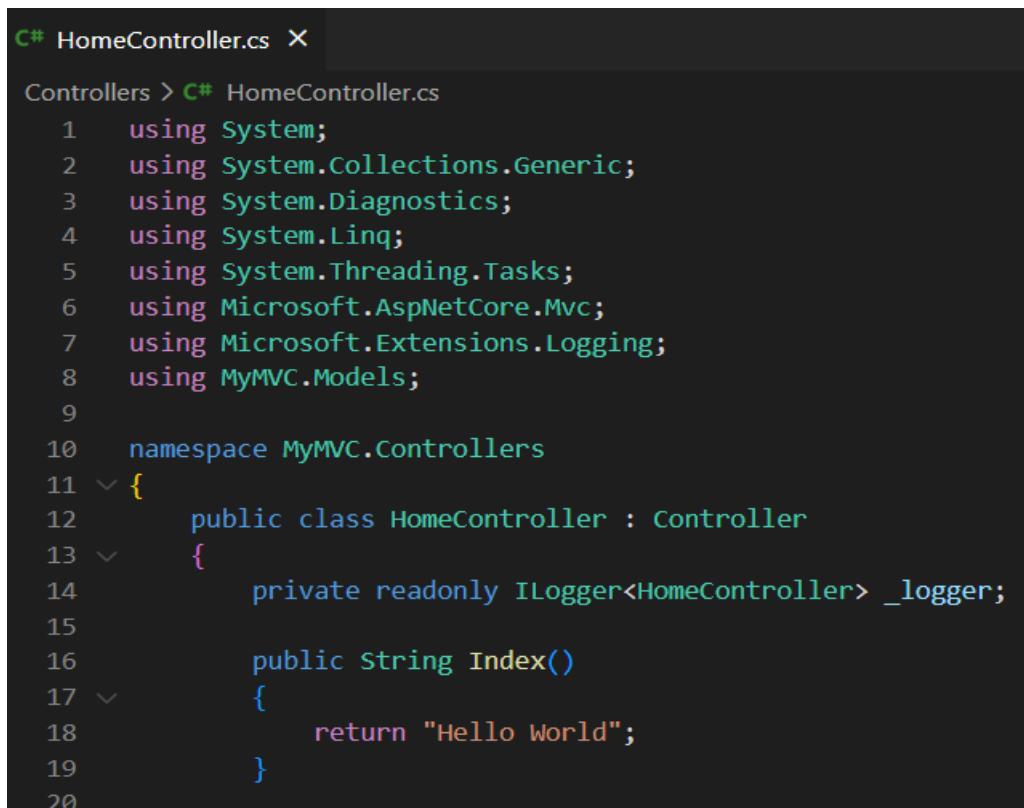
C:\MyMVC>dotnet new mvc --auth none
The template "ASP.NET Core Web App (Model-View-Controller)" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/aspnetcore/3.0-third-party-notices for details.

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

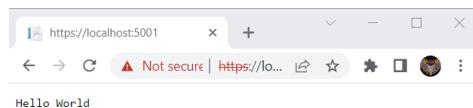
Controllers > C# HomeController.cs
1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Linq;
5  using System.Threading.Tasks;
6  using Microsoft.AspNetCore.Mvc;
7  using Microsoft.Extensions.Logging;
8  using MyMVC.Models;
9
10 namespace MyMVC.Controllers
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:

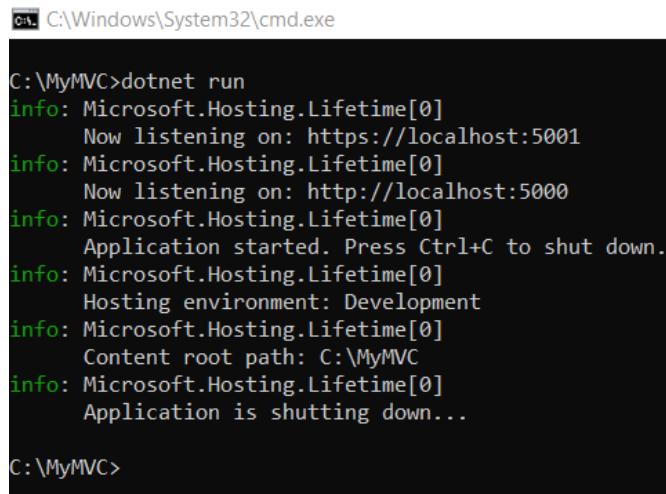
- Run the code

```
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.

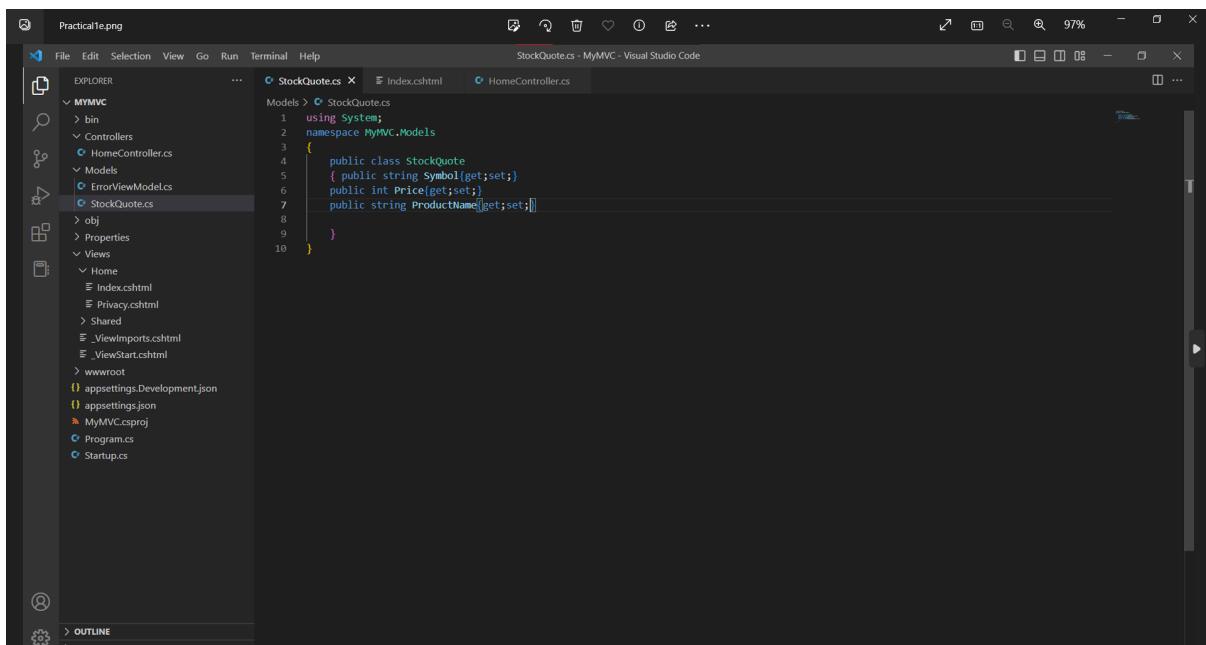


```
C:\MyMVC>dotnet run
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]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

```

<!DOCTYPE html>
<html>
<head>
    <title>Home Page</title>
</head>
<body>
    <div class="text-center">
        <h1>Welcome Rahul Kewat</h1>
        Symbol: @Model.Symbol <br/>
        Price: @Model.Price<br/>
        Product name: @Model.ProductName<br/>
        <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
    </div>
</body>
</html>

```

Step 9:

- Now modify HomeController.cs file to match following:

```

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 IActionResult Index()
        {
            var model=new StockQuote{Symbol="INR",Price=58000,ProductName="IP"};
            return View(model);
        }

        public HomeController	ILogger<HomeController> _logger;
        {
            _logger = logger;
        }

        //public IActionResult Index()
        //{
        //    return View();
        //}

        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }
    }
}

```

Step 10:

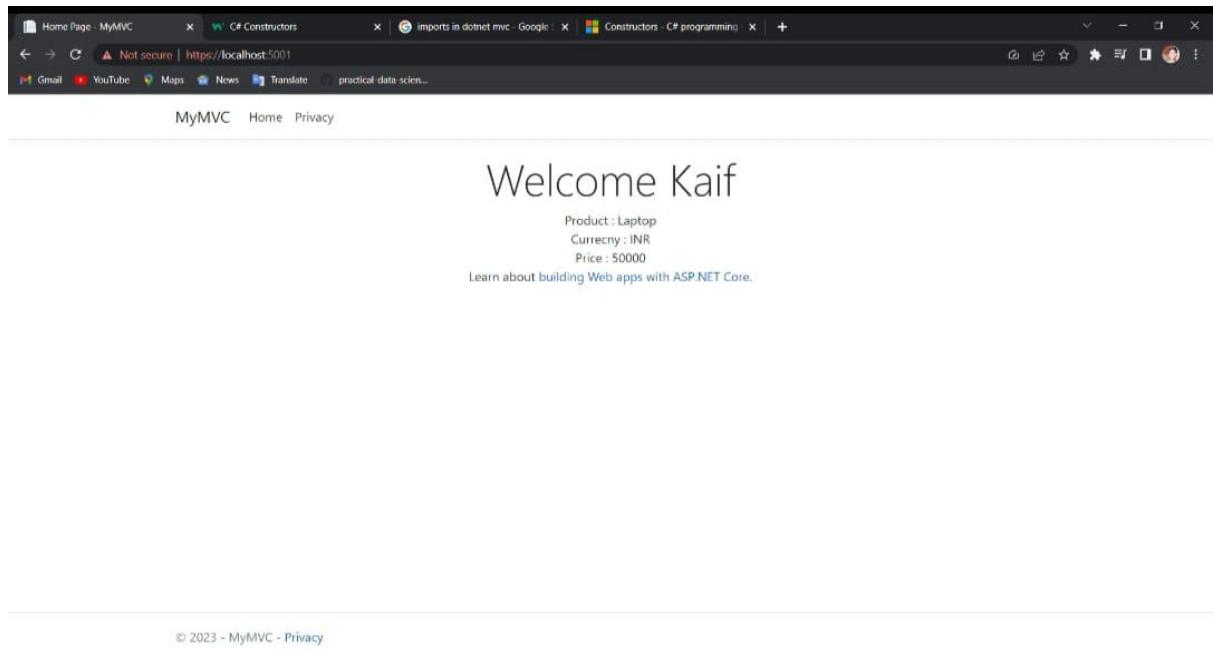
- Now run the project using **dotnet run**

```
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"}]
C:\>
```

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;
```

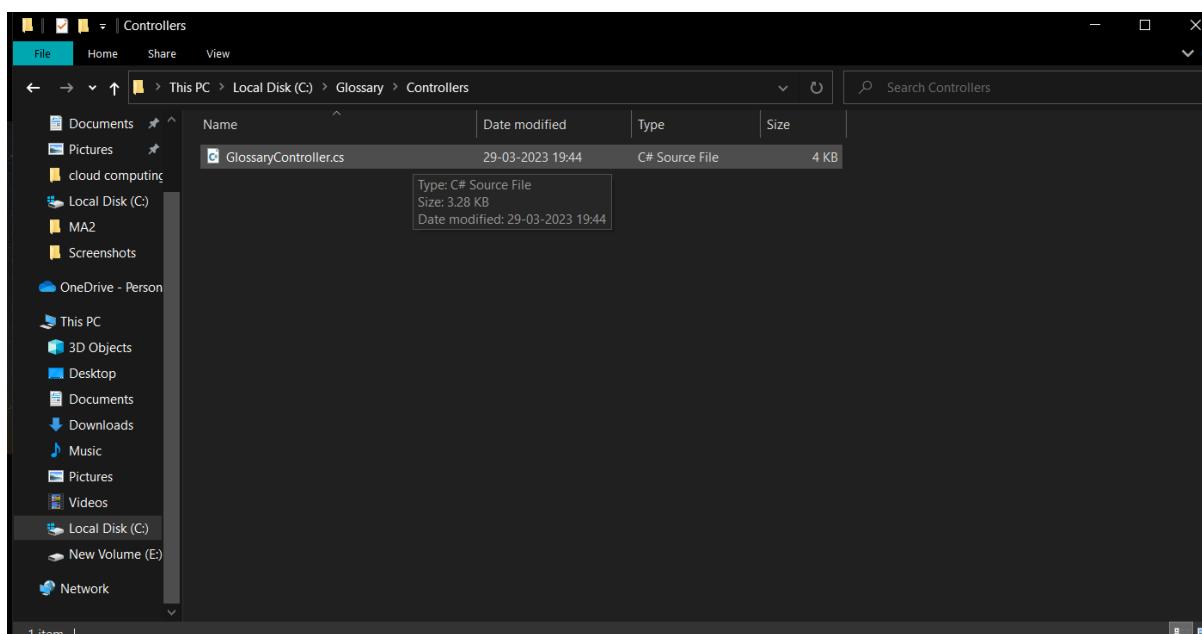
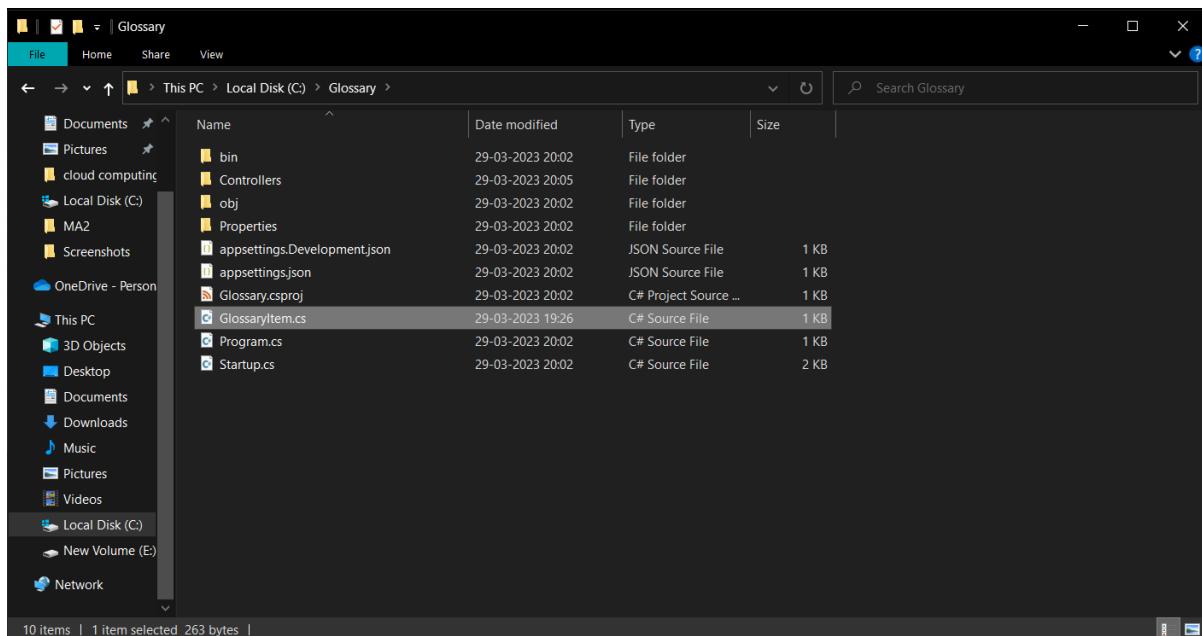
```
C: > Users > Dell > OneDrive > Pictures > MA2 > C# GlossaryController.cs
  36     [HttpGet]
  37     public ActionResult<List<GlossaryItem>> Get()
  38     {
  39         return Ok(Glossary);
  40     }
  41
  42     [HttpGet]
  43     [Route("{term}")]
  44     public ActionResult<GlossaryItem> 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
 68
 69     else
 70     {
 71         Glossary.Add(glossaryItem);
 72         var resourceUrl = Path.Combine(Request.Path.ToString(), Uri.EscapeUriString(glossaryItem.
 73             return Created(resourceUrl, glossaryItem);
 74     }
 75 }
 76
 77 [HttpPost]
 78 public ActionResult Put(GlossaryItem glossaryItem)
 79 {
 80     var existingGlossaryItem = Glossary.Find(item =>
 81     item.Term.Equals(glossaryItem.Term, StringComparison.InvariantCultureIgnoreCase));
 82     if (existingGlossaryItem == null)
 83     {
 84         return BadRequest("Cannot update a non-existing term.");
 85     }
 86
 87     else
 88     {
 89         existingGlossaryItem.Definition = glossaryItem.Definition;
 90         return Ok();
 91     }
 92 }
 93 [HttpDelete]
 94 [Route("{term}")]
 95 public ActionResult Delete(string term)
 96 {
 97     var glossaryItem = Glossary.Find(item =>
 98     item.Term.Equals(term, StringComparison.InvariantCultureIgnoreCase));
 99     if (glossaryItem == null)
100     {
101         return NotFound();
102     }
103
104     else
105     {
106         Glossary.Remove(glossaryItem);
107         return NoContent();
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.

 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 "Content-Type: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."}]
C:\Glossary>
```

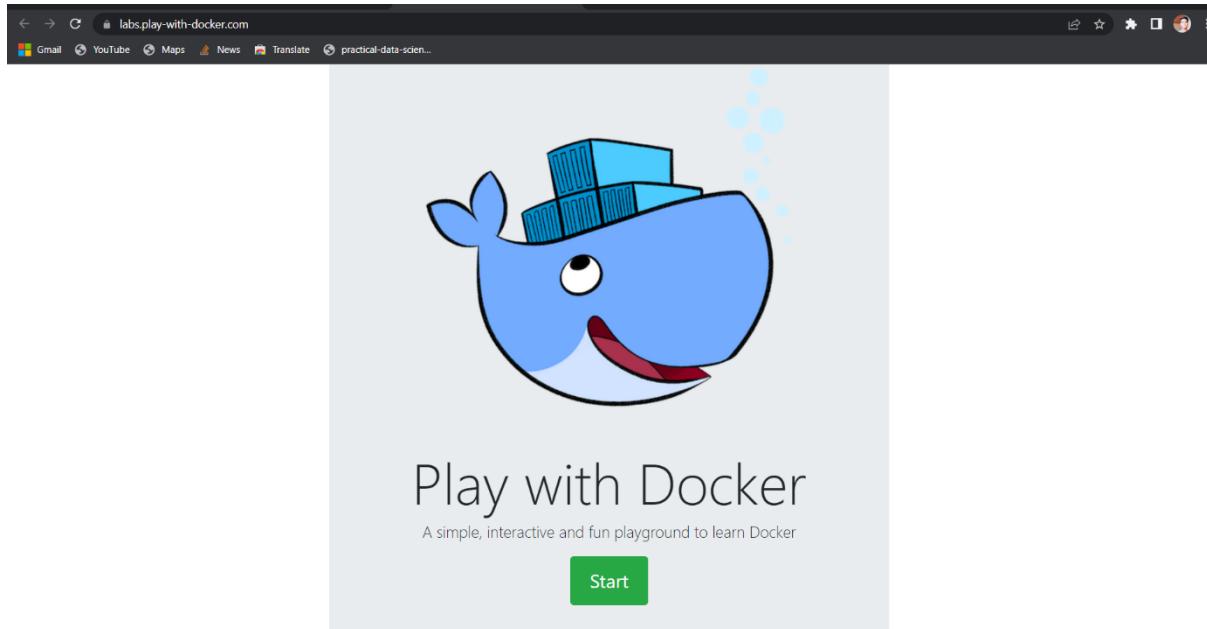
Practical 3

Aim: Working with Docker

Writeup:

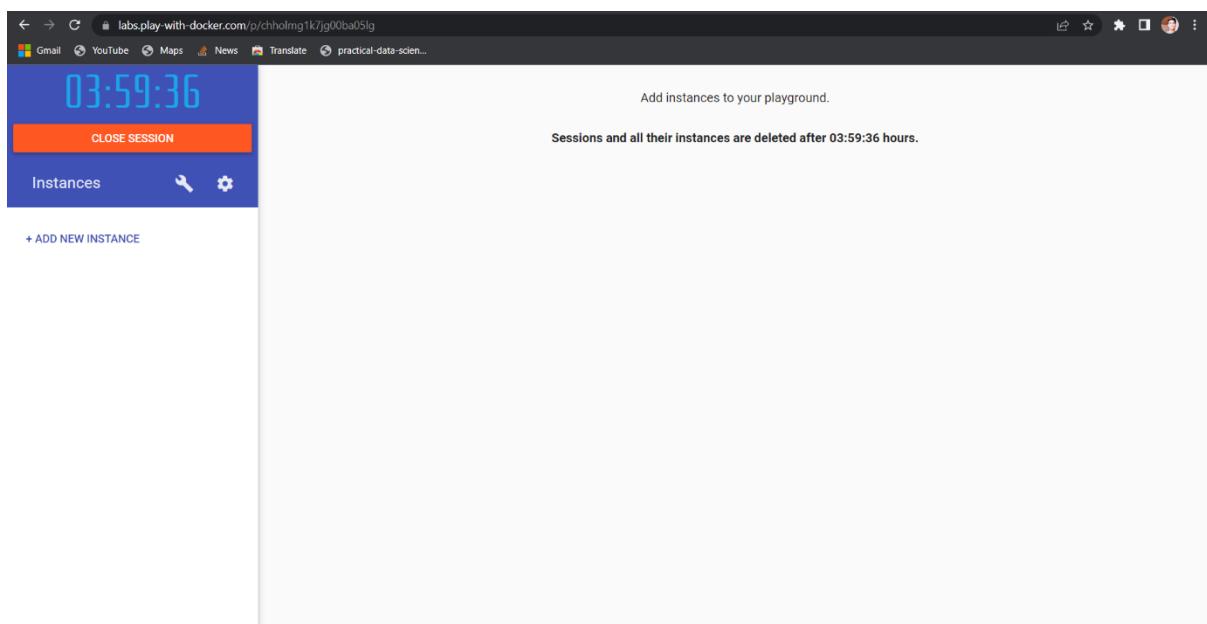
Step 1: create Docker Hub account (sign up)

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



Click on start

Step 3: add new instance



Step 4: perform following:

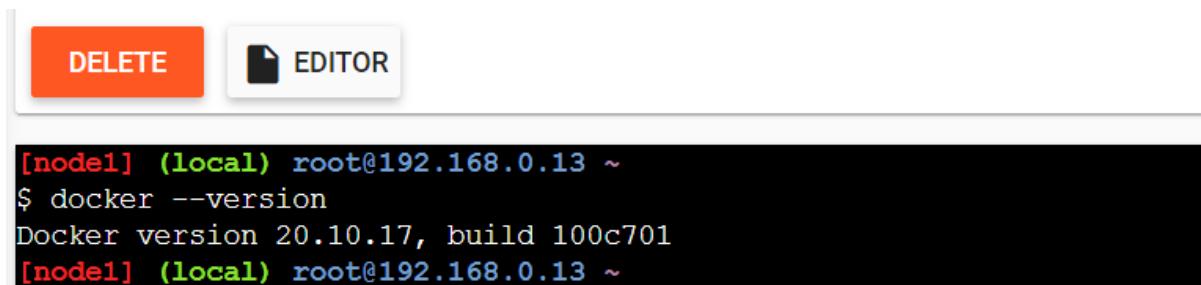
Method1:

To pull and push images using docker

Command: to check docker version

docker –version

Output:



The screenshot shows a terminal window with two buttons at the top: "DELETE" (orange) and "EDITOR" (grey). The terminal output is as follows:

```
[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:



The screenshot shows a terminal window with the following 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:

Want faster and simpler Kubernetes development? Test out [Telepresence for Docker](#) today.

Create repository

Namespace: kaif3120 Repository Name*: repo1

Pro tip

You can push a new image to this repository using the CLI

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Description

Make sure to change tagname with your desired image repository tag.

Visibility

Using 0 of 1 private repositories. [Get more](#)

Public Appears in Docker Hub search results

Private Only visible to you

[Cancel](#) [Create](#)

Click on Create button

Now check repository created

Want faster and simpler Kubernetes development? Test out [Telepresence for Docker](#) today.

General

kaif3120 / Repository / repo1 / General

Using 0 of 1 private repositories. [Get more](#)

Add a short description for this repository

The short description is used to index your content on Docker Hub and in search engines. It's visible to users in search results.

[Update](#)

kaif3120 / repo1

Description

This repository does not have a description [edit](#)

Last pushed: a few seconds ago

Docker commands

To push a new tag to this repository,

```
docker push kaif3120/repo1:tagname
```

[Public View](#)

Tags

This repository is empty. When it's not empty, you'll see a list of the most recent tags here.

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. [Read more about automated builds](#).

Command: to login to your docker account

```
docker login --username=kaif3120
```

password:

Output:

```
[response from daemon: See https://registry.docker.io/v2/ for unauthorized information]
$1] (local) root@192.168.0.28 ~
$ docker login --username=kaif3120
$ 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
$ Succeeded
$1] (local) root@192.168.0.28 ~
```

Command : to tag image

```
docker tag 9d0311b60ec6 kaif3120/repo1:firsttry
```

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

Output:

```
[node1] (local) root@192.168.0.28 ~
$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
docker/verse    latest   9d0311b60ec6  13 days ago  3.43GB
[node1] (local) root@192.168.0.28 ~
$ docker tag ^C
[node1] (local) root@192.168.0.28 ~
$ docker tag 9d0311b60ec6 kaif3120/repo1:firsttry
[node1] (local) root@192.168.0.28 ~
```

Command: to push image to docker hub account

```
docker push kaif3120/repo1:firsttry
```

note: firsttry is tag name created above.

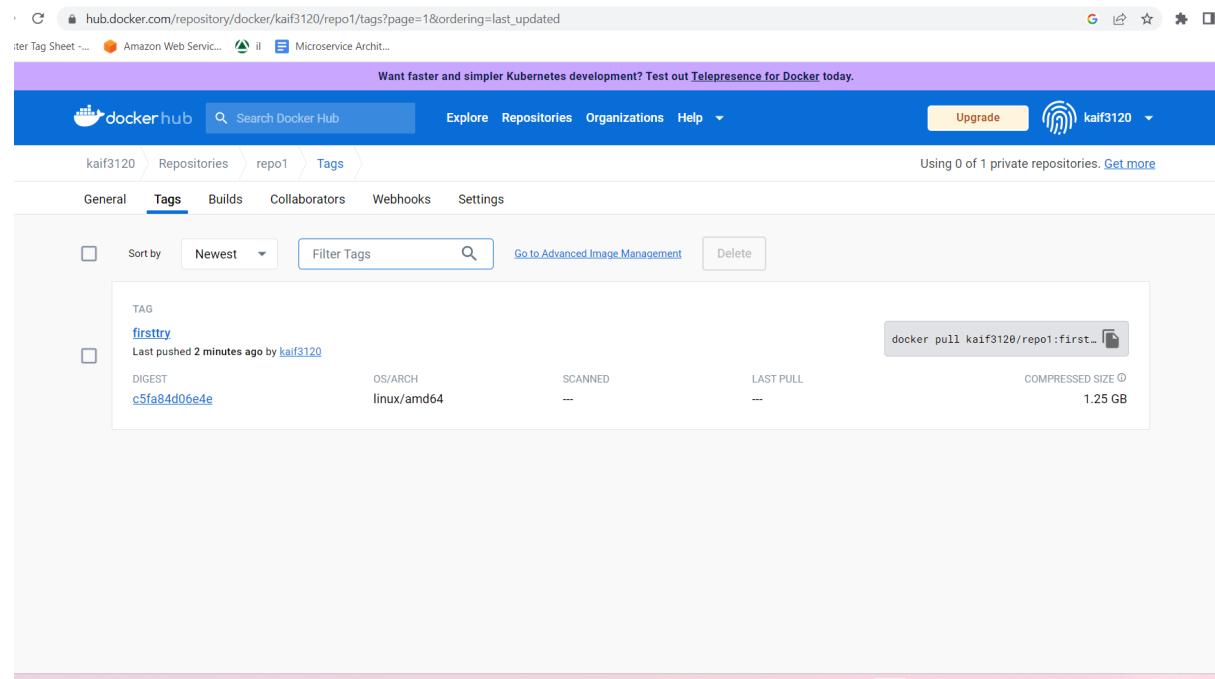
Output:

```
(local) root@192.168.0.28 ~
$ docker push kaif3120/repo1:firsttry
The push refers to repository [docker.io/kaif3120/repo1]
34033f: Mounted from rocker/verse
ab534: Mounted from rocker/verse
16f055: Mounted from rocker/verse
a59e83: Mounted from rocker/verse
14b08c: Mounted from rocker/verse
9e576d: Mounted from rocker/verse
5015cb: Mounted from rocker/verse
09e33c: Mounted from rocker/verse
398654: Mounted from rocker/verse
10656a: Mounted from rocker/verse
digest: sha256:c5fa84d06e4e536bfde730e90e0b54f7714c222d572d18e60c047c5839b76f57 size: 2428
(local) root@192.168.0.28 ~
```

Check it in docker hub now

The screenshot shows the Docker Hub interface for the repository 'kaif3120/repo1'. The top navigation bar includes links for Explore, Repositories, Organizations, Help, and an Upgrade button. The user profile 'kaif3120' is visible on the right. The main content area displays the repository details: 'General' tab selected, a note to add a short description, the repository name 'kaif3120 / repo1', and a 'Docker commands' section with a placeholder for pushing a new tag. Below this are sections for 'Tags' (empty) and 'Automated Builds' (disabled). A sidebar on the left shows the user's repositories.

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:

```
[root@192.168.0.28 ~]# cat > Dockerfile <<EOF
FROM busybox
CMD echo "hello world, this is my first docker image"
EOF
[root@192.168.0.28 ~]# docker build -t kaif3120/repo2 .
[root@192.168.0.28 ~]# docker build --help .
Usage: docker build [OPTIONS] PATH | URL | -
      build an image from a Dockerfile
[root@192.168.0.28 ~]
```

Command : to build image from docker file

dokcer build -t kaif3120/repo2 .

Output:

```
d an image from a Dockerfile
e1] (local) root@192.168.0.28 ~
cker build -t kaif3120/repo2 .
ing build context to Docker daemon 12.8kB
 1/2 : FROM busybox
st: Pulling from library/busybox
69979d33: Pull complete
st: sha256:560af6915bfc8d7630e50e212e08242d37b63bd5c1ccf9bd4acccf116e262d5b
us: Downloaded newer image for busybox:latest
> 8135583d97fe
 2/2 : CMD echo "hello world, this is my first docker image"
> Running in 3c4e9f6b2fb6
ving intermediate container 3c4e9f6b2fb6
> de71d6431c78
essfully built de71d6431c78
essfully tagged kaif3120/repo2:latest
e1] (local) root@192.168.0.28 ~
```

Command: to check docker images

docker images

```
(local) root@192.168.0.28 ~
r images
DRY      TAG      IMAGE ID      CREATED      SIZE
0/repo2  latest   de71d6431c78  55 seconds ago  4.86MB
          latest   8135583d97fe  7 days ago    4.86MB
0/repo1  firsttry 9d0311b60ec6  13 days ago   3.43GB
verse    latest   9d0311b60ec6  13 days ago   3.43GB
(local) root@192.168.0.28 ~
```

Command: to push image to docker hub

docker push kaif3120/repo2

Output:

```
docker push kaif3120/repo2
sing default tag: latest
he push refers to repository [docker.io/kaif3120/repo2]
547b4c33213: Mounted from library/busybox
atest: digest: sha256:53b4cde66f50f04446ee2deb19c621812758d9c2c0eba86616c0c63b085a5172 size: 528
```

Now check it on docker hub

The screenshot shows the Docker Hub interface. At the top, there is a purple banner with the text "Want faster and simpler Kubernetes development? Test out Telepresence for Docker today." Below the banner, the Docker Hub header includes the logo, a search bar with the placeholder "Search Docker Hub", navigation links for "Explore", "Repositories", "Organizations", and "Help", and a user profile section for "kaif3120". A yellow "Upgrade" button is visible in the top right. The main content area displays five repository cards for the user "kaif3120":

- kaif3120 / repo2**
Contains: No content | Last pushed: a few seconds ago
Inactive, 0 stars, 0 downloads, Public
- kaif3120 / repo1**
Contains: No content | Last pushed: 2 minutes ago
Inactive, 0 stars, 0 downloads, Public
- kaif3120 / sl-project**
Contains: Image | Last pushed: a year ago
Inactive, 0 stars, 21 downloads, Public
- kaif3120 / sp-b-demo**
Contains: Image | Last pushed: a year ago
Inactive, 0 stars, 18 downloads, Public
- kaif3120 / apache-ubuntu**
Contains: Image | Last pushed: 2 years ago
Inactive, 0 stars, 9 downloads, Public

command: to run docker image:

docker run kaif3120/repo2

output:

```
[node1] (local) root@192.168.0.28 ~
$ docker run kaif3120/repo2
hello world, this is my first docker image
[node1] (local) root@192.168.0.28 ~
```

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\Dell>docker volume rm myvol1
myvol1

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

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

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

Practical 5

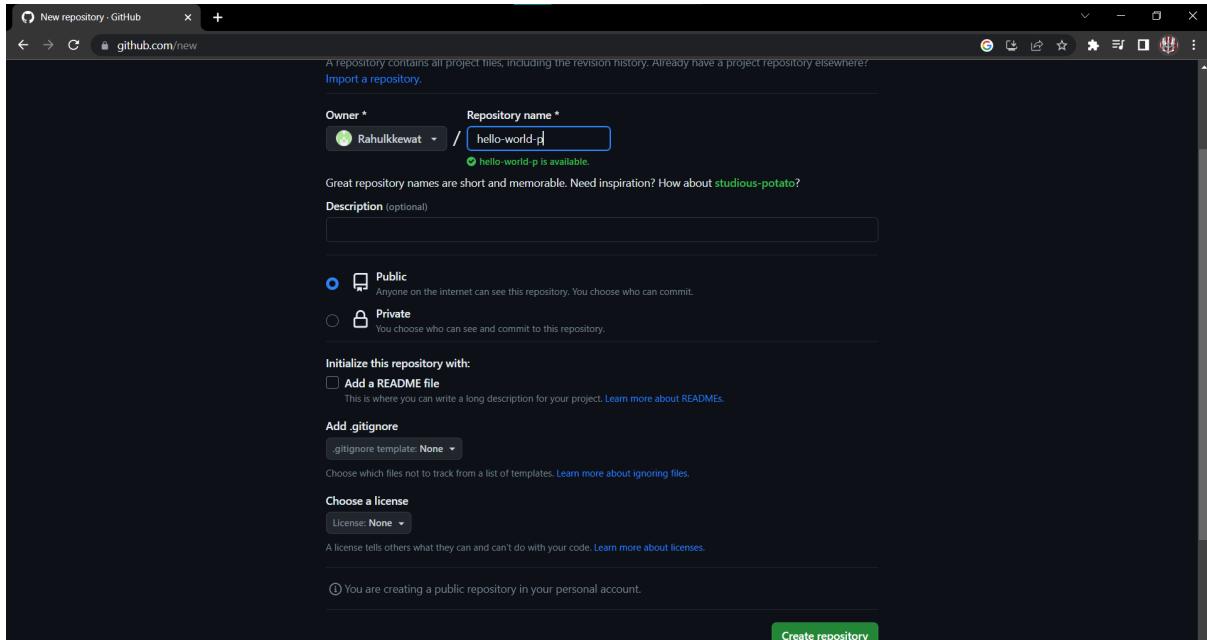
Aim: Working with Circle CI for continuous integration

Writeup:

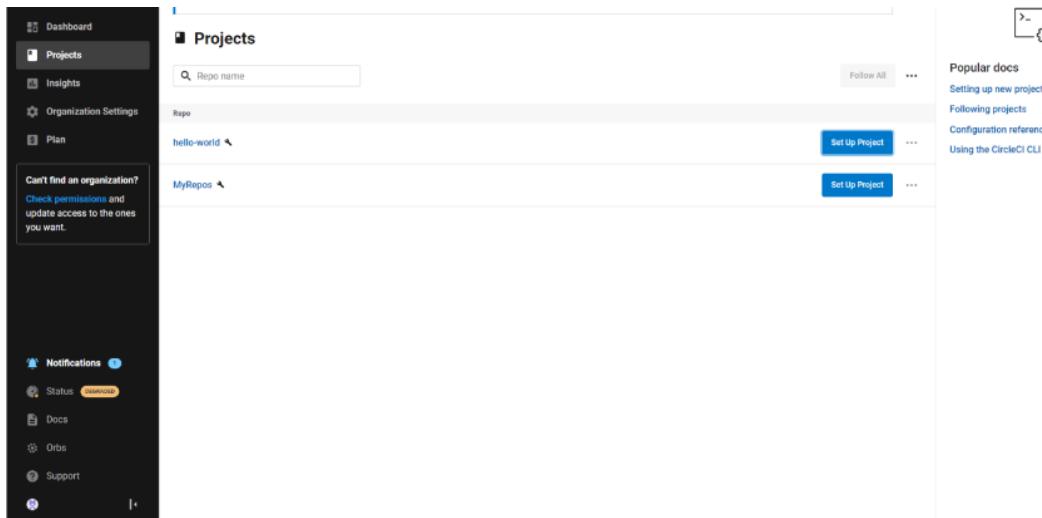
Step 1: Create a repository

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.

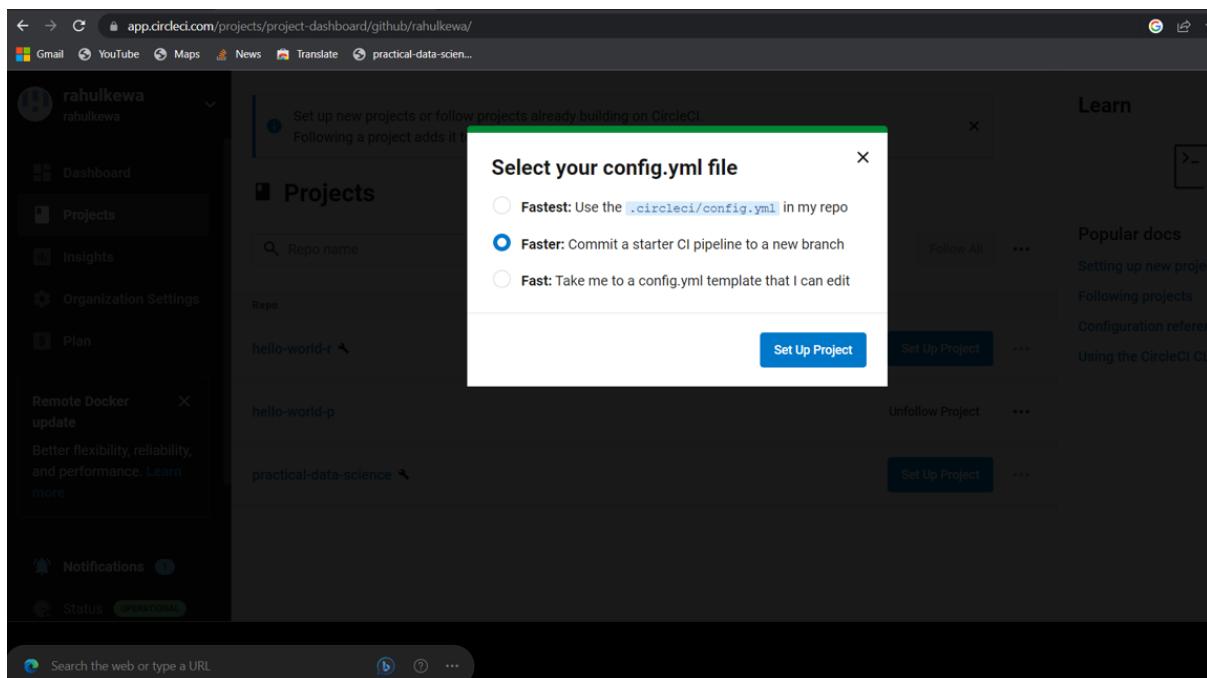


Login to Circle CI <https://app.circleci.com/> Using GitHub Login, Once logged in navigate to Projects.



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

- hello-world 6 Success one_and_two cicleci-project-setup 6f4c152 indentation errors 2mo ago
- hello-world 5 Unable to parse YAML while parsing a block collection in 'string', line 5, column 7: - image: cimg/ruby:2.6.8
- hello-world 4 Unable to parse YAML while parsing a block collection in 'string', line 5, column 7: - image: cimg/ruby:2.6.8
- hello-world 3 Error calling workflow: 'workflow' Error calling job: 'build' Cannot find a definition for command named node/with-cache
- hello-world 2 Success say-hello-workflow cicleci-project-setup 99a6b4d Add .circleci/config.yml 2mo ago
- hello-world 1 Success say-hello-workflow cicleci-project-setup 99a6b4d 2mo ago

No more pipelines to load

Select “say-hello” Job with a green tick

Self-hosted machine runner version 1.0 on cloud will sunset on July 27, 2023, and will be temporarily unavailable on June 28. [Update](#)

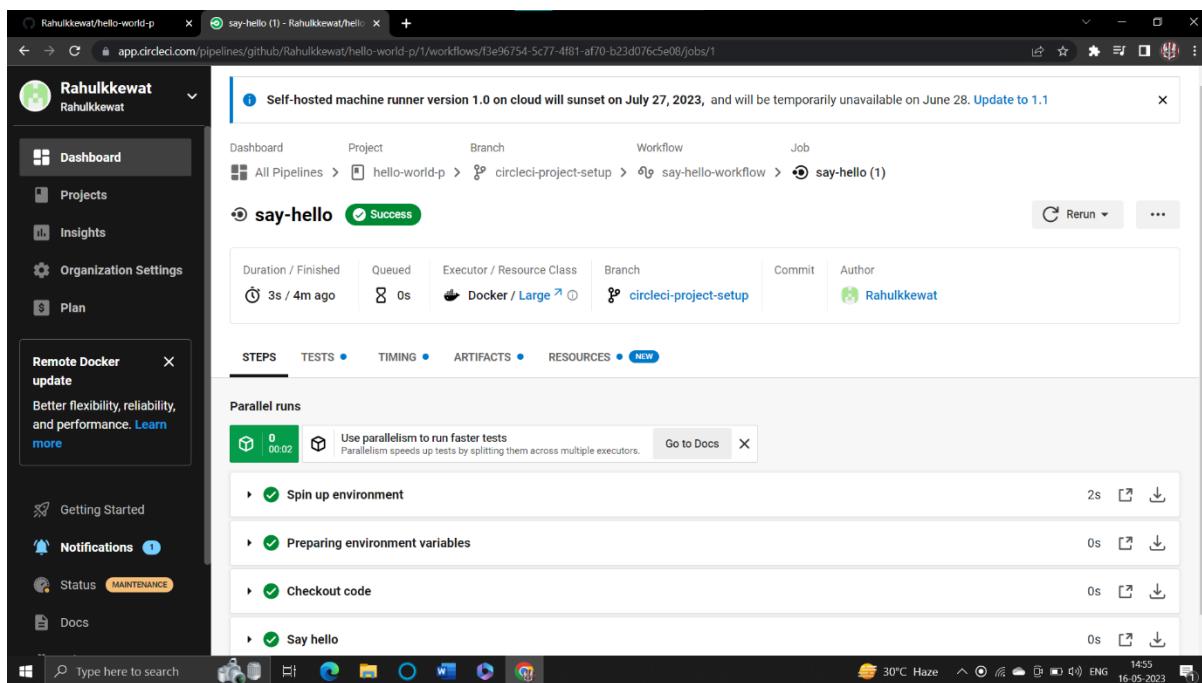
Dashboard Project Branch Workflow

All Pipelines > hello-world > cicleci-project-setup > say-hello-workflow

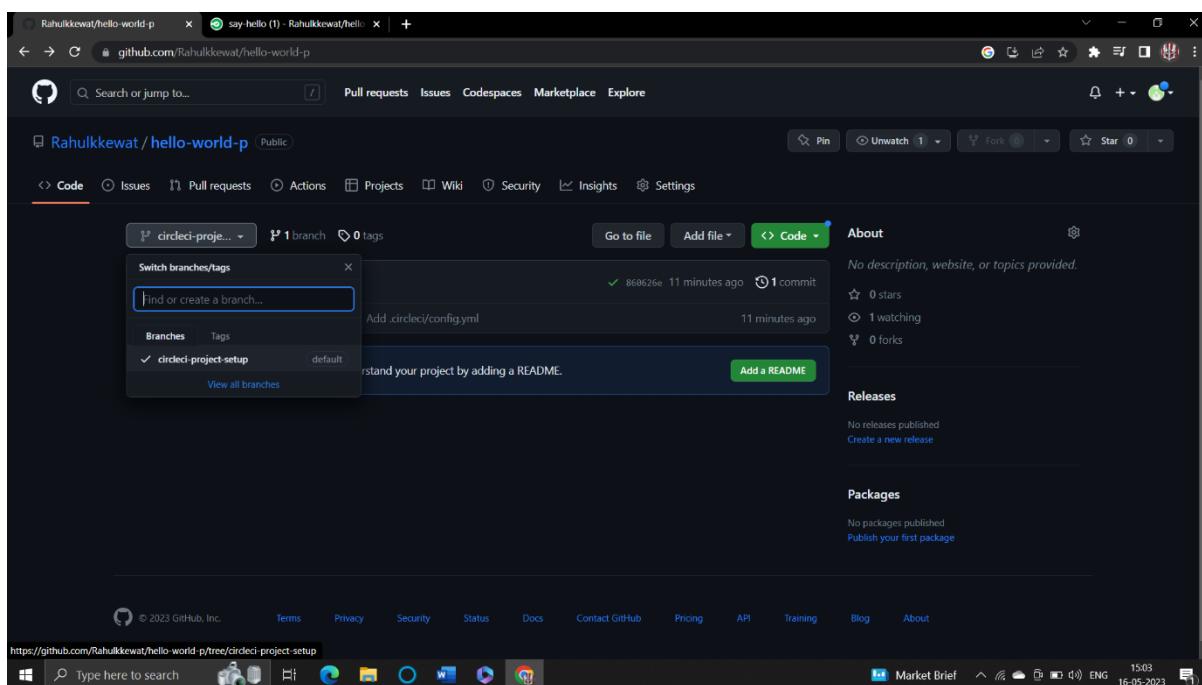
say-hello-workflow Success

Duration / Finished	Branch	Commit	Author
4s / 2mo ago	cicleci-project-setup	99a6b4d	kaif3120

say-hello 3s



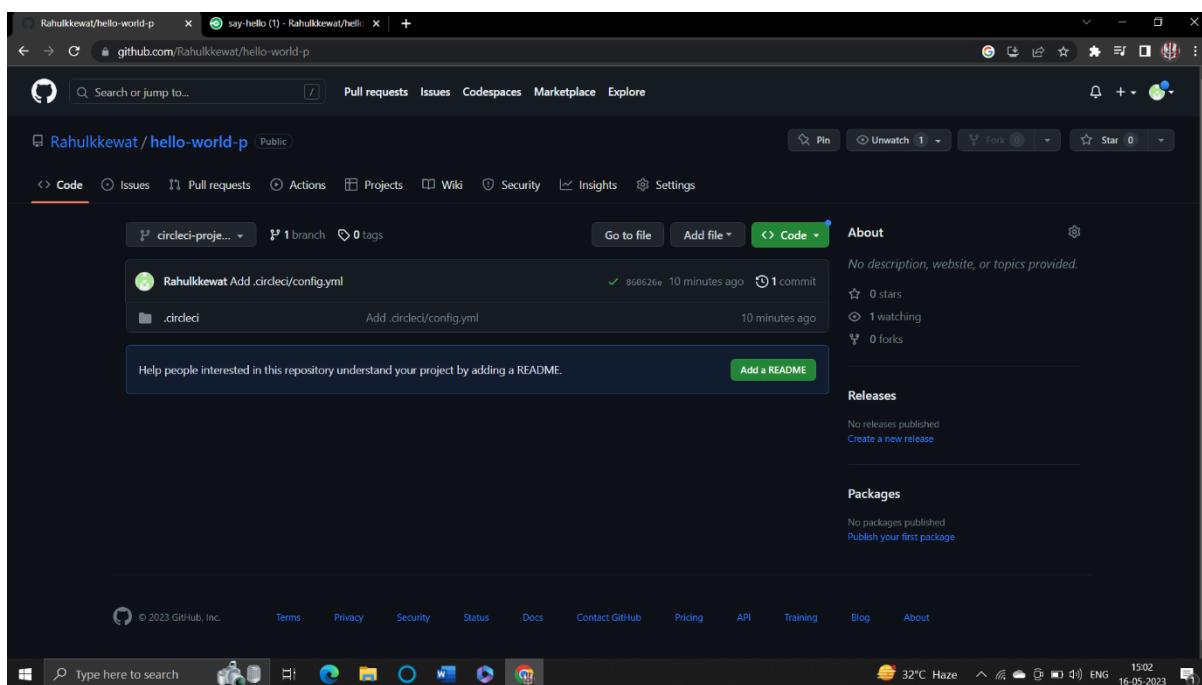
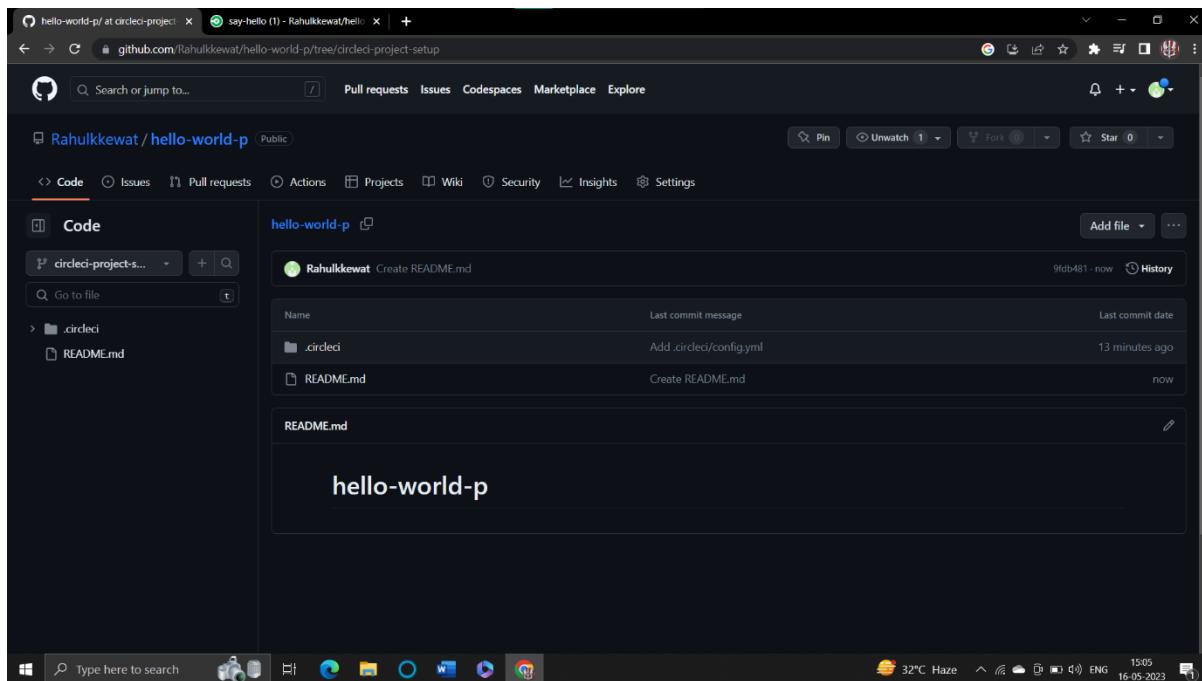
Select Branch and option circleci-project-setup



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     docker:
11       image: cimg/base:stable
12       # Add steps to the job
13       # See: https://circleci.com/docs/configuration-reference/#steps
14       steps:
15         - checkout
16         - run:
17           name: "Say hello"
18           command: "echo Hello, World!"
19
20   # Orchestrate jobs using workflows
21   # See: https://circleci.com/docs/configuration-reference/#workflows
22   workflows:
23     say-hello-workflow:
24       jobs:
25         - say-hello
26

```

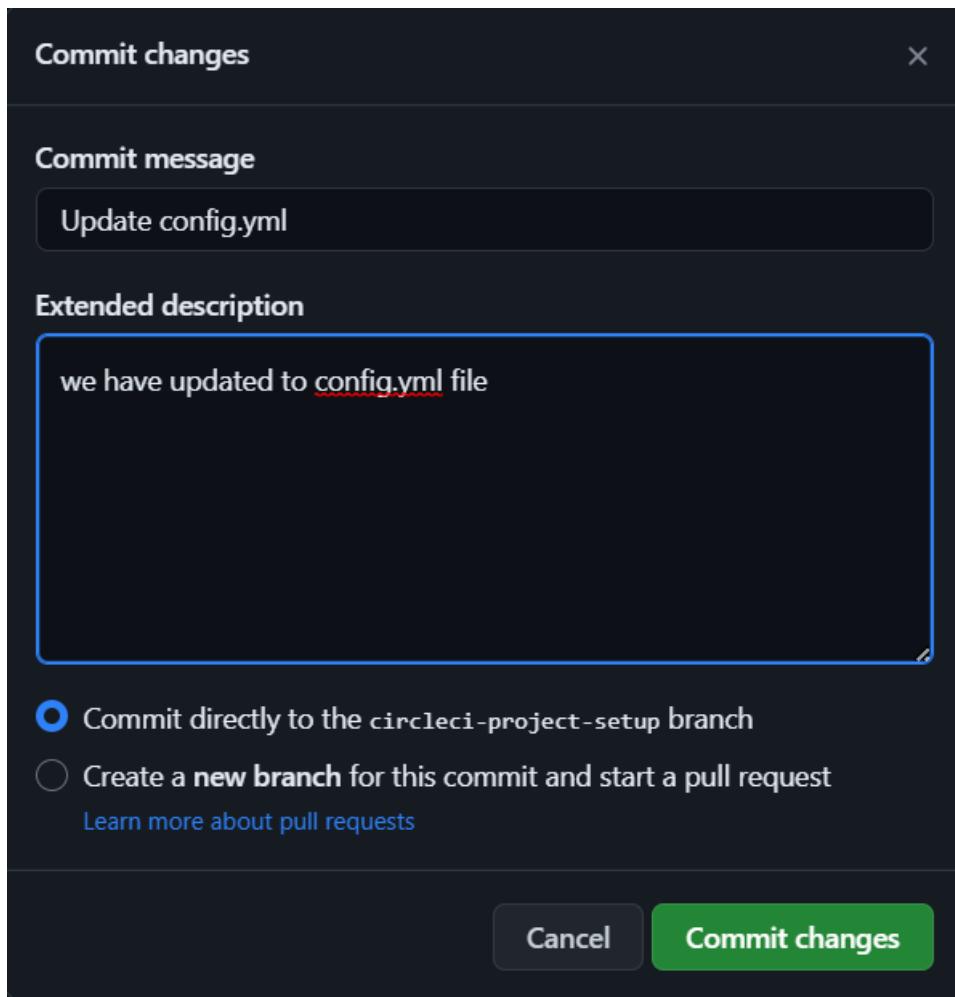
The GitHub file editor should look like this

```

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
hello-world 6	Success	one_and_two	circleci-project-setup 6f4c152 indentation errors	2mo ago
hello-world 5	⚠️ Failed	one_and_two	circleci-project-setup 6f4c152 indentation errors	2mo ago
hello-world 4	⚠️ Failed	one_and_two	circleci-project-setup 6f4c152 indentation errors	2mo ago
hello-world 3	⚠️ Failed	one_and_two	circleci-project-setup 6f4c152 indentation errors	2mo ago
hello-world 2	Success	say-hello-workflow	circleci-project-setup 99a6b4d Add .circleci/config.yml	2mo ago
hello-world 1	Success	say-hello-workflow	circleci-project-setup 99a6b4d	2mo ago

No more pipelines to load

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.

- 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:
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

```

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

```

version: 2
jobs:
  one:
    docker:
      - image: cimg/ruby:2.6.8
      auth:
        username: kaif13120
        password: $DOC_PASS #env
    steps:
      - checkout
      - run: echo "A first hello"
      - run: sleep 25
  two:
    docker:
      - image: cimg/ruby:3.0.2
      auth:
        username: kaif13120
        password: $DOC_PASS
    steps:
      - checkout
      - run: echo "A more familiar hi"
      - run: sleep 15
workflows:
  version: 2
  one_and_two:
    jobs:
      - one
      - two

```

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

Pipeline Status Workflow Branch / Commit Start

- hello-world 6 Success one_and_two circleci-project-setup 6f4c152 Indentation errors 2mo ago
- hello-world 5 Unable to parse YAML while parsing a block collection in 'string', line 5, column 7: - image: cimg/ruby:2.6.8
- hello-world 4 Unable to parse YAML while parsing a block collection in 'string', line 5, column 7: - image: cimg/ruby:2.6.8
- hello-world 3 Error calling workflow: 'workflow' Error calling job: 'build' Cannot find a definition for command named node/with-cache
- hello-world 2 Success say-hello-workflow circleci-project-setup 99a6b4d Add .circleci/config.yml 2mo ago
- hello-world 1 Success say-hello-workflow circleci-project-setup 99a6b4d 2mo ago

No more pipelines to load

Self-hosted machine runner version 1.0 on cloud will sunset on July 27, 2023, and will be temporarily unavailable on June 28. [Up](#)

Dashboard Project Branch Workflow

All Pipelines > hello-world > circleci-project-setup > one_and_two

one_and_two Success

Duration / Finished Branch Commit Author & Message

36s / 2mo ago circleci-project-setup 6f4c152 indentation errors

two	33s
one	31s

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

```

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:

```

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
21 two:
22   docker:
23     - image: cimg/ruby:3.0.2
24     auth:
25       username: mydockerhub-user
26       password: $DOCKERHUB_PASSWORD # context / project UI env-var reference
27   steps:
28     - checkout
29     - run: echo "A more familiar hi"
30     - attach_workspace:
31       # Must be absolute path or relative path from working_directory
32       at: my_workspace

```

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

```

1  version: 2
2  jobs:
3    one:
4      docker:
5        - image: cimg/ruby:2.6.8
6        auth:
7          username: kaif3120
8          password: $OOC_PASS #env
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
21 two:
22   docker:
23     - image: cimg/ruby:3.0.2
24     auth:
25       username: kaif3120
26       password: $OOC_PASS
27   steps:
28     - checkout
29     - run: echo "A more familiar HI"
30     - attach_workspace:
31       at: my_workspace
32     - run: |
33       if [[ $(cat my_workspace/echo-output) == "trying out workspaces" ]]; then

```

```

31         echo "It worked!"
32       else
33         echo "Nope!"; exit 1
34       fi
35   workflows:
36     version: 2
37     one_and_two:
38       jobs:
39         - one
40         - two:
41           requires:
42             - one

```

Finally your workflow with the jobs running should look like this

The screenshot shows the CircleCI web interface for a user named 'kaif3120'. The main navigation bar includes 'Dashboard', 'Project', 'Branch', and 'Workflow'. Below this, the project structure is shown: 'All Pipelines' > 'hello-world' > 'circleci-project-setup' > 'one_and_two'. A specific workflow run titled 'one_and_two' is selected, indicated by a green checkmark icon and the word 'Success'. The run details include: Duration / Finished (19s / 2mo ago), Branch (circleci-project-setup), Commit (a108b0e), and Author & Message (Updated config.yml). The workflow itself consists of two sequential steps: 'one' and 'two', connected by a horizontal arrow labeled '12s'. Both steps are marked with green checkmarks, indicating they have completed successfully. A small delay of '3s' is shown between them. On the left sidebar, there is a 'Remote Docker update' notification with a message about better flexibility, reliability, and performance, and a 'Status' indicator showing 'OPERATIONAL'. At the bottom of the page, a status bar shows 'Waiting for rum-http-intake.logs.datadoghq.com...'.

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\KAIF\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\KAIF\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\KAIF\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\KAIF\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;
        }

        public IActionResult Index()
        {
            var teams = repository.Teams;
            return View(teams);
        }
    }
}

```

```
{  
    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());  
        }  
    }  
}  
  
[HttpPost]  
[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();
    }
}

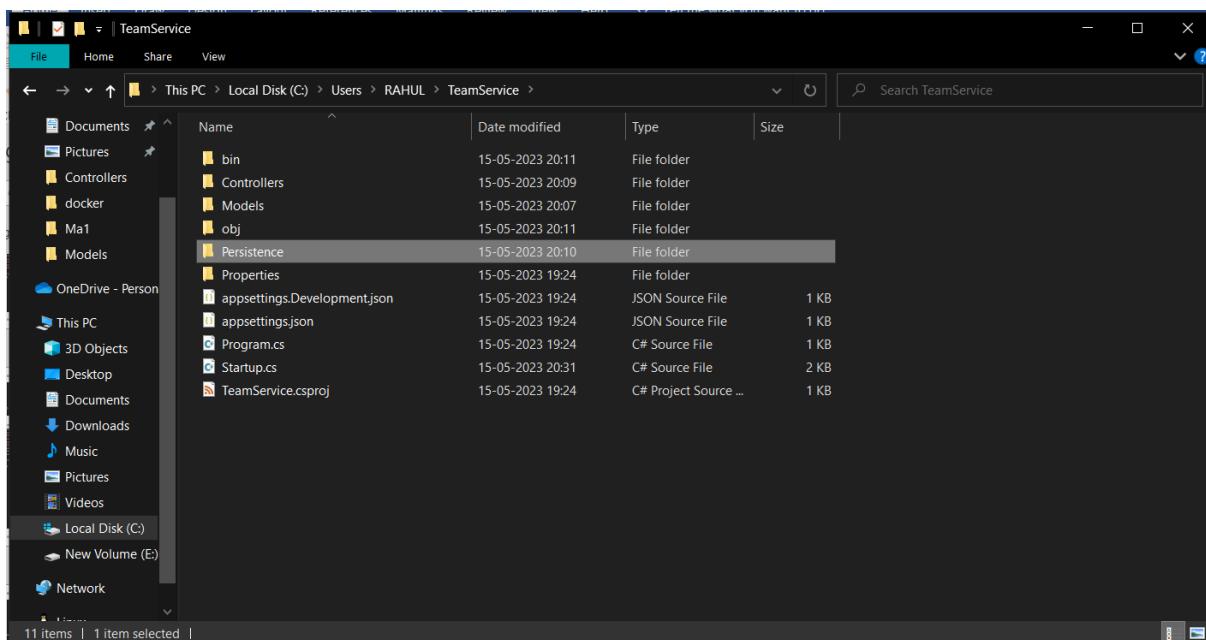
private Guid GetTeamIdForMember(Guid memberId)
{
    foreach (var team in repository.List())
    {
        var member = team.Members.FirstOrDefault(m => m.ID == memberId);
        if(member != null)
        {
            return team.ID;
        }
    }
}

```

```
        }
    }

    return Guid.Empty;
}
```

Step 4: Create folder "C:\Users\KAIF\TeamService\Persistence".



Step 5: Add file ITeamRepository.cs in “C:\Users\KAIF\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\KAIF\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
{"name": "KC", "id": "e52baa63-d511-417e-9e54-7aab04286281", "members": []}
C:\Users\RAHUL\TeamService>
```

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

Step 13: On Command prompt 2**Command:** To get all teams

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

```
C:\Users\RAHUL\TeamService>
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":[]}]
```

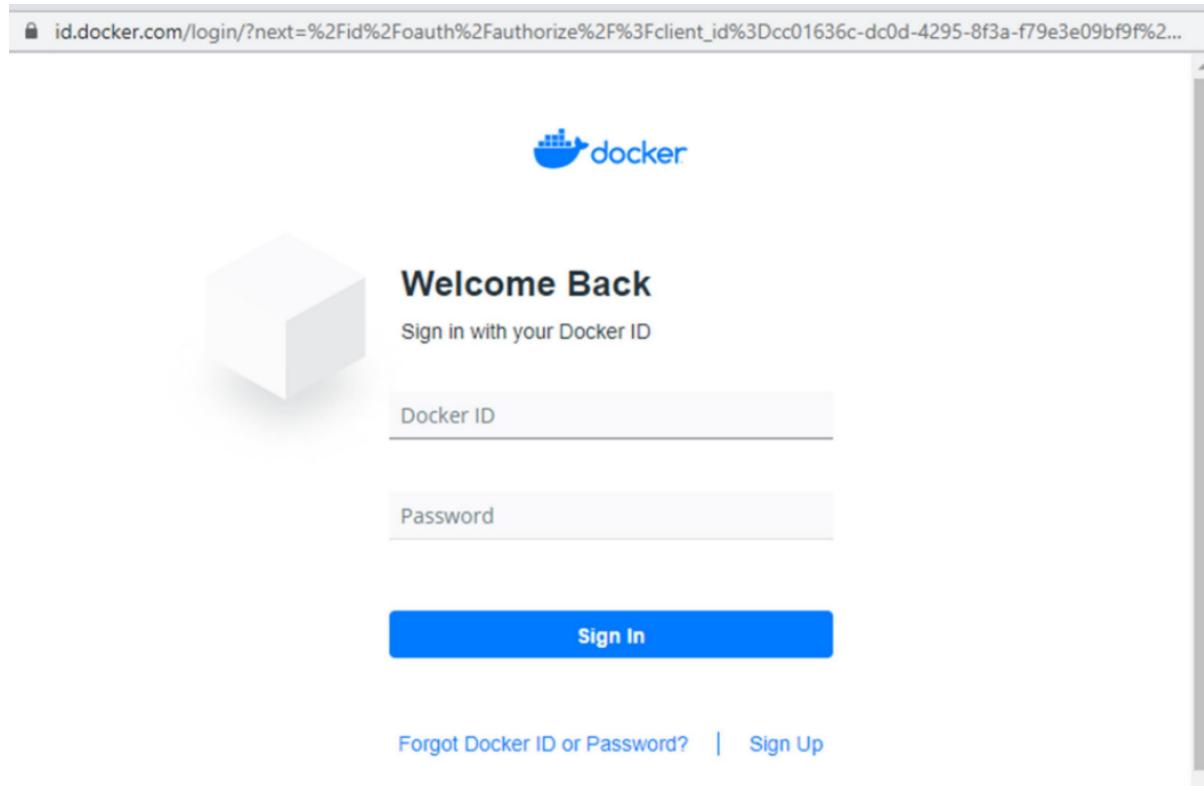
Practical 7

Aim: Running Location Service in Docker

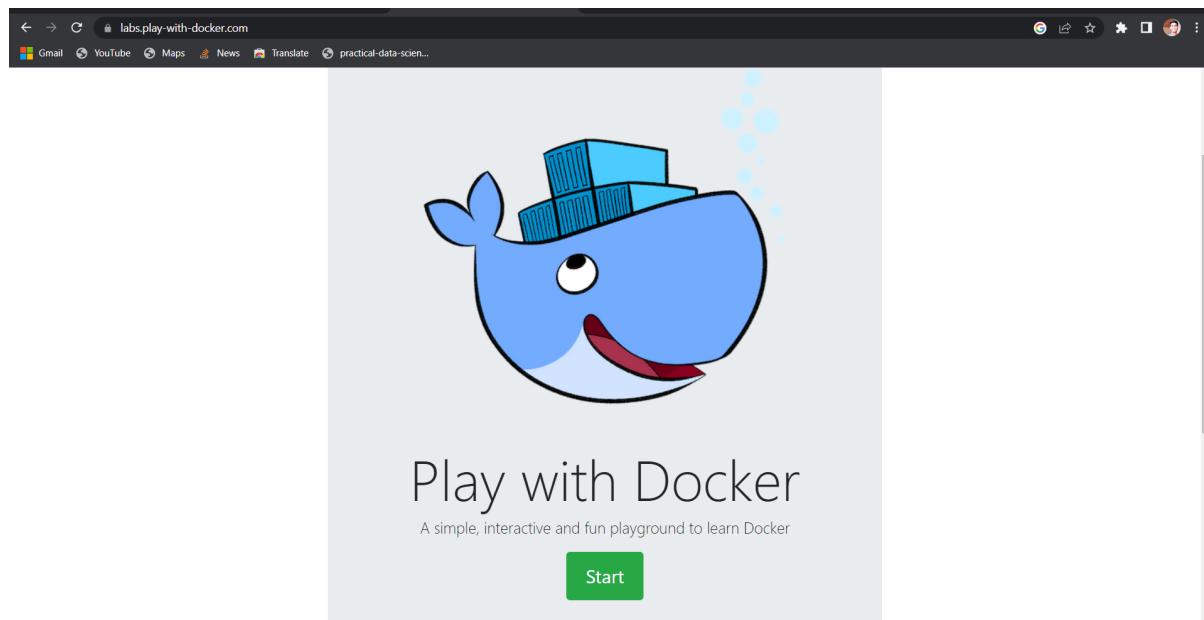
Writeup:

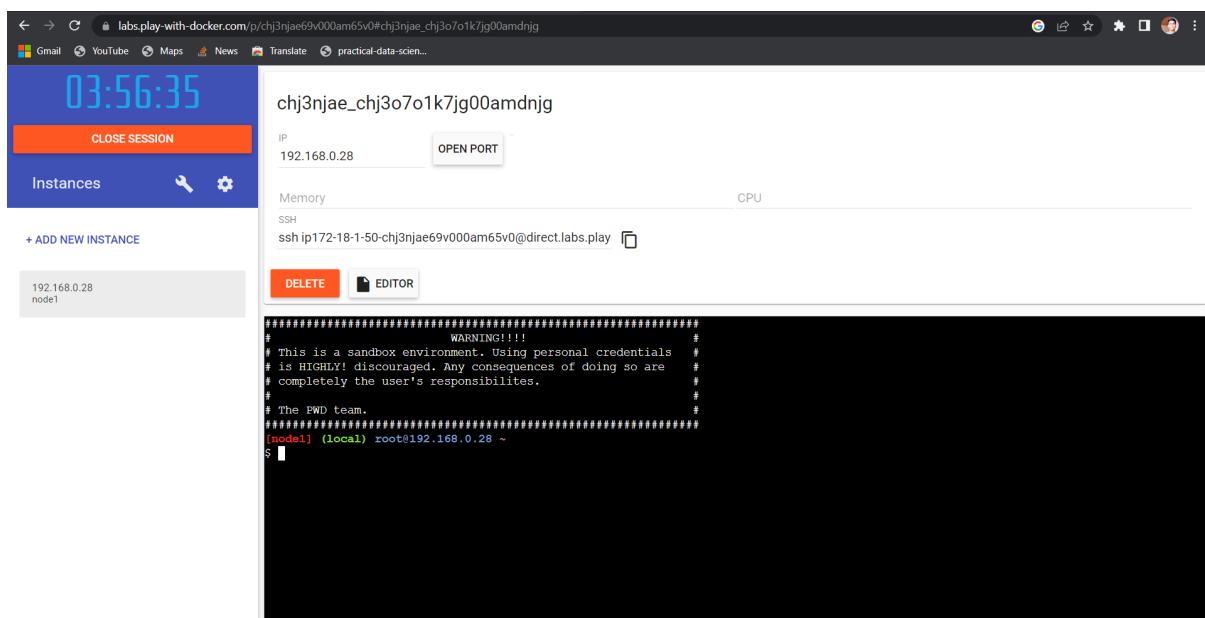
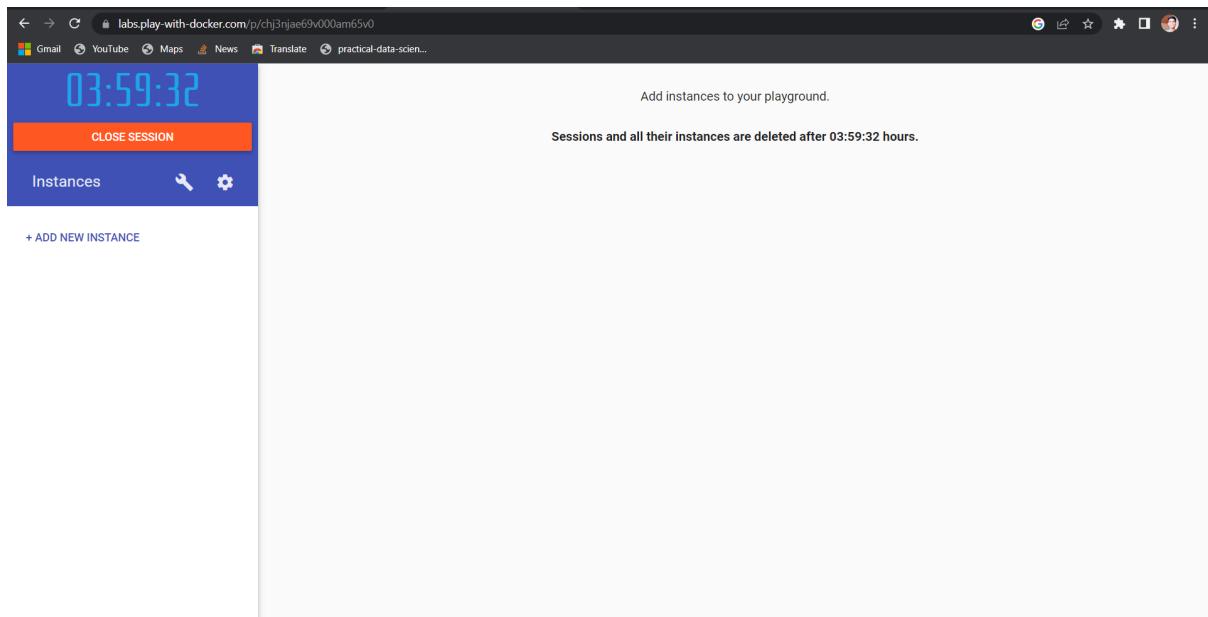
Step 1: create docker hub login first to use it in play with docker

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:

```
03:41:08
CLOSE SESSION
IP 192.168.0.13 OPEN PORT 5000
Instances SSH
+ ADD NEW INSTANCE
192.168.0.13 node1
DELE EDITOR

[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
081cd4bf4d521: 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
42a40bbc1d53ef44726bb6e275cf0c700d5e041fb4593310db61280177342052
(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)

```
02:41:00
CLOSE SESSION
IP 192.168.0.13 OPEN PORT 5001 5000
Instances SSH
+ ADD NEW INSTANCE
192.168.0.13 node1
DELE EDITOR

[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
081cd4bf4d521: Already exists
5d2dc01312f3: Already exists
585880aea240: Already exists
3905d0c644ea: Already exists
c59037c90022: Already exists
d8c03883a4ca: Pull complete
Digest: sha256:5f7aca33c5e2117e04f58a59e0cf96fd20d5cbf2cf66c3cd708118d573255168
Status: Downloaded newer image for dotnetcoreservices/locationservice:nodb
ab1837840015fafec00ced52971d2aa8b14ea9a6d29c87b6a669b05e94aeb24
(node1) (local) root@192.168.0.13 ~
```

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

Docker images

Output:

The screenshot shows a session for a Docker container named 'cgogqmgs_cgogrvo1k7jg00dtv8dg' with IP 192.168.0.13. The terminal shows the following output:

```
[node1] (local) root@192.168.0.13 ~
$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
dotnetcoreservices/teamservice   location   b27d0de8f2de  6 years ago  886MB
dotnetcoreservices/locationservice nodb      03339f0ea9dd  6 years ago  883MB
```

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**[curl http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281](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@192.168.0.13 ~
```

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

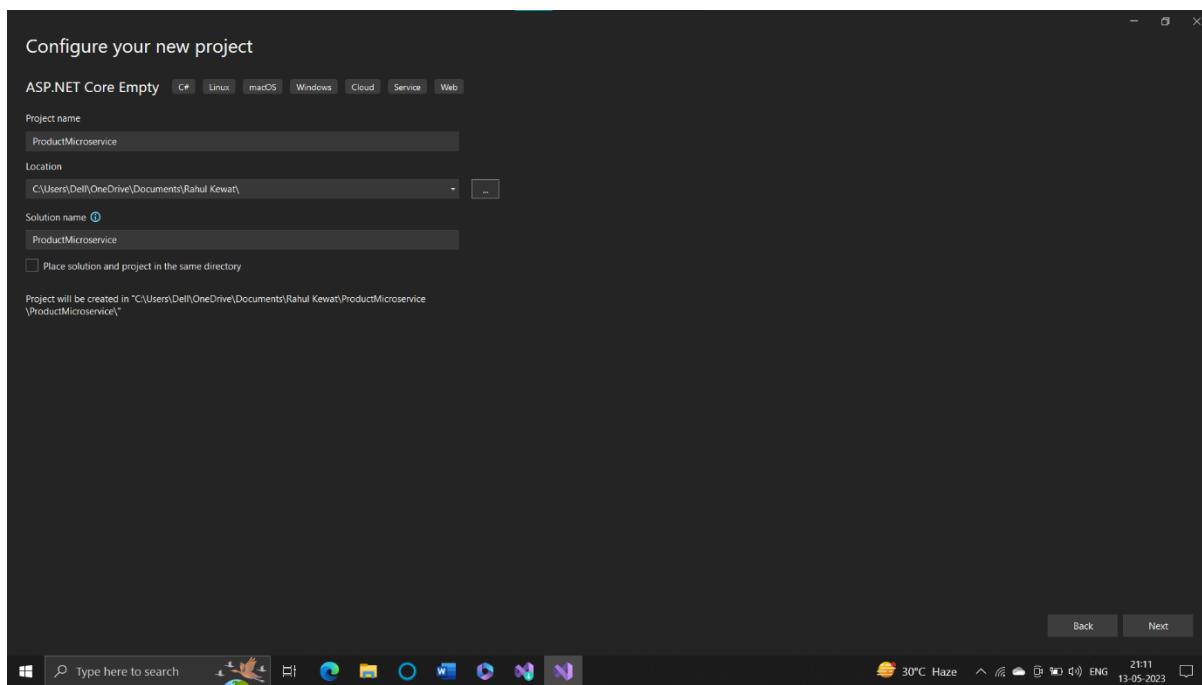
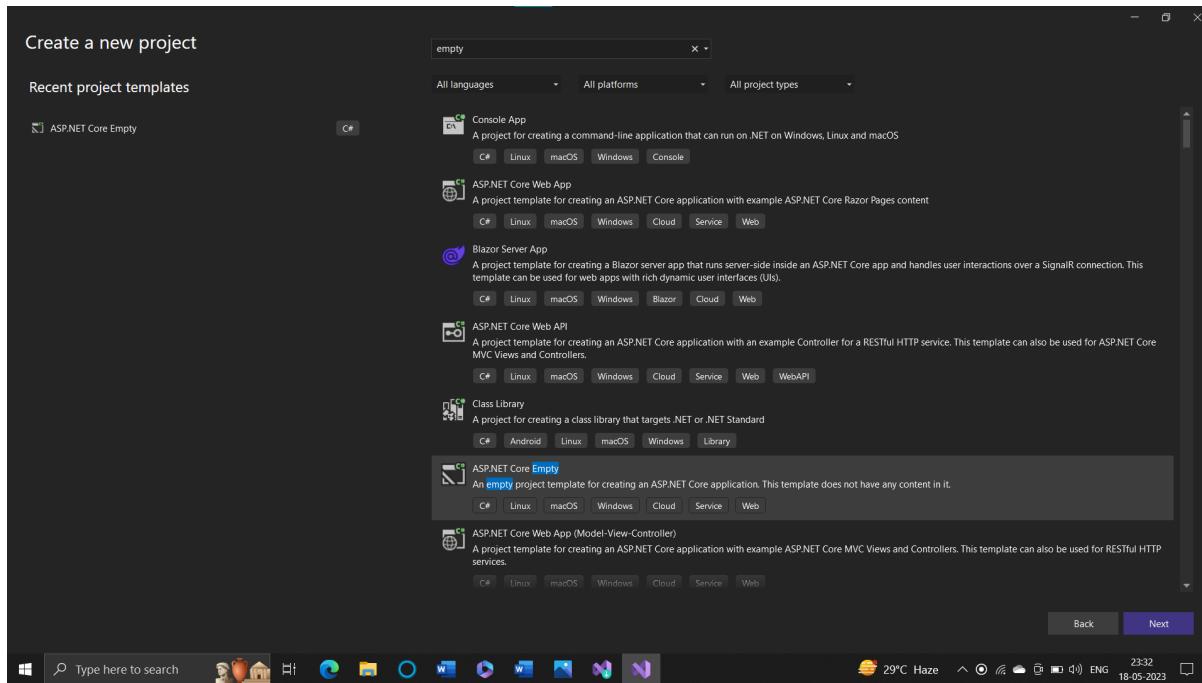
```
$ curl http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292
[{"id": "64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f", "latitude": 12.0, "longitude": 12.0, "altitude": 10.0, "timestamp": 0, "memberID": "63e7acf8-8fae-42ce-9349-3c8593ac8292"}] [node1] (local) root@192.168.0.13 ~
$
```

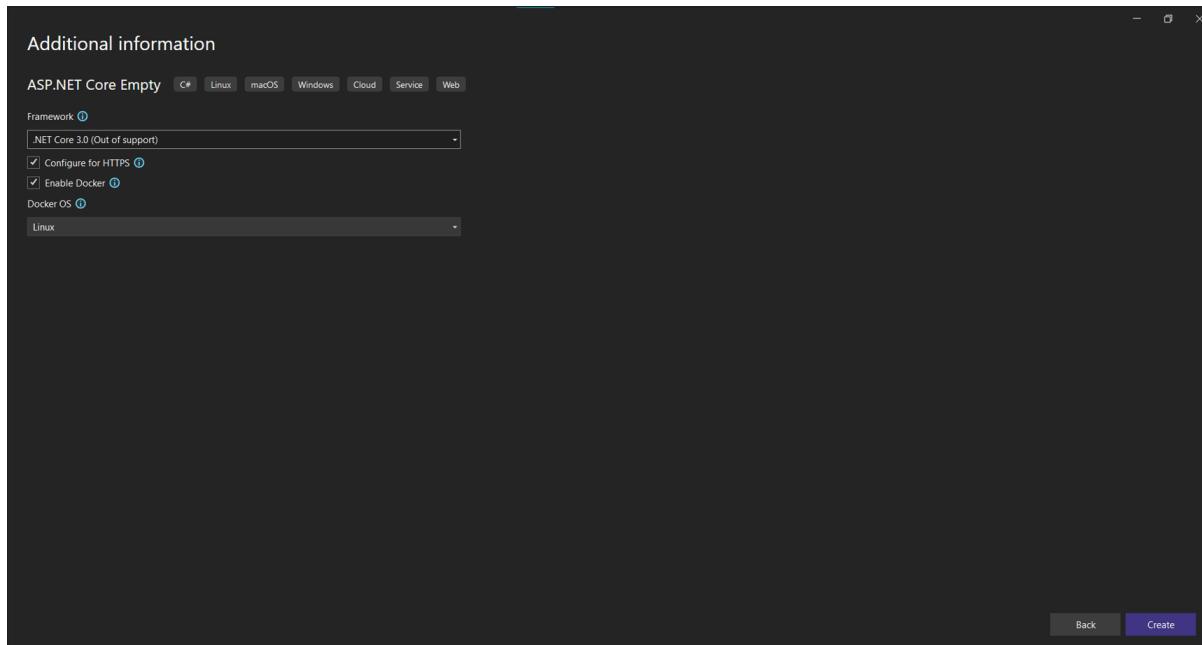
Practical 8

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

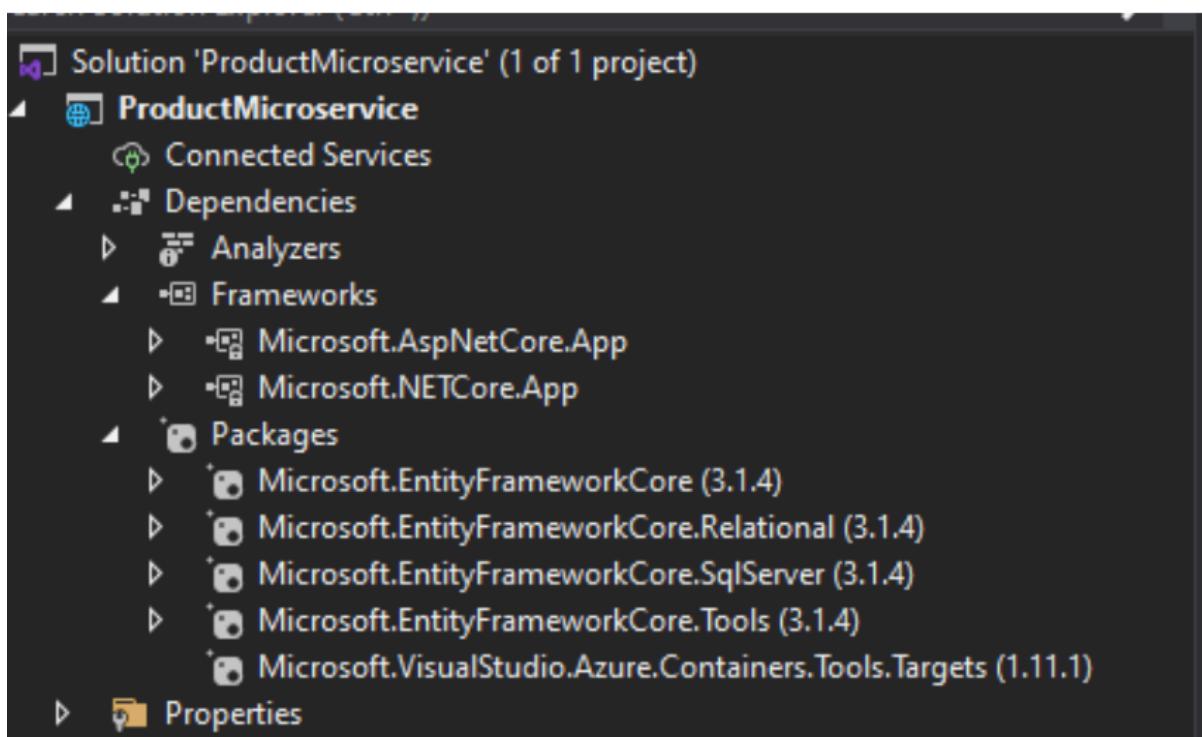
Writeup:

Step 1: Create and configure a new empty project

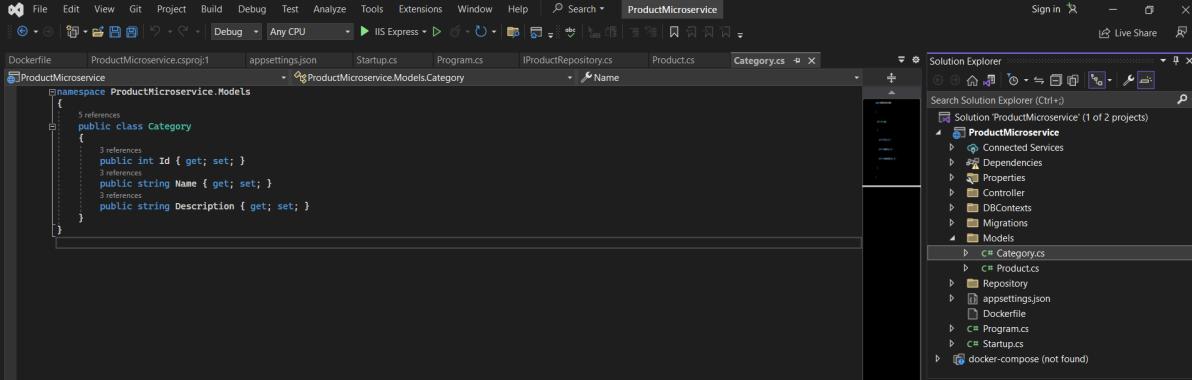




Step 2: Make sure to add all these packages via nugget package manager



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



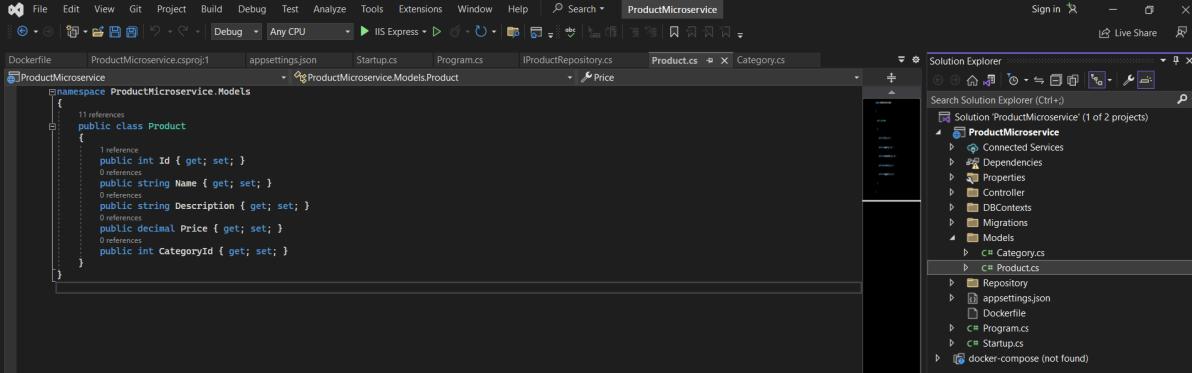
```

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

```

The screenshot shows the Visual Studio IDE with the code editor open to the Category.cs file. The code defines a simple class named Category with properties for Id, Name, and Description. The Solution Explorer on the right shows the project structure, including the Models folder where Category.cs is located.

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



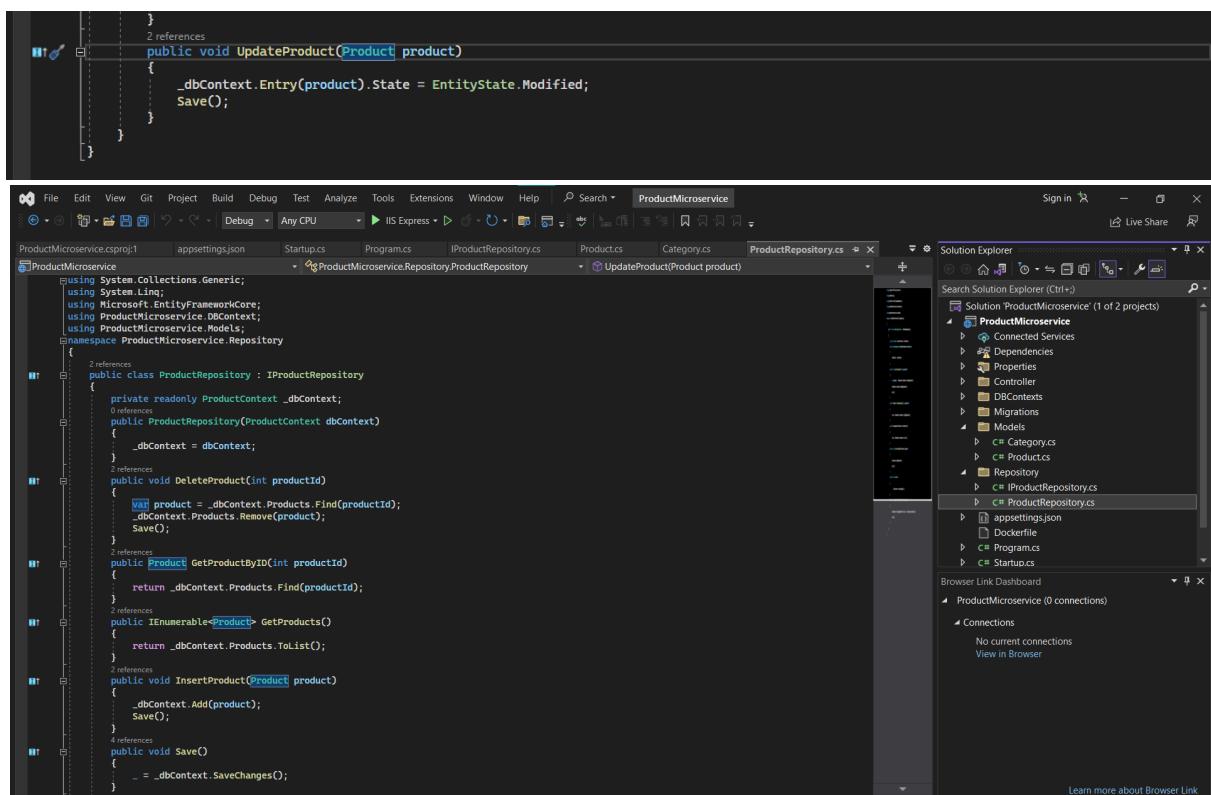
```

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; }
    }
}

```

The screenshot shows the Visual Studio IDE with the code editor open to the Product.cs file. The code defines a Product class with properties for Id, Name, Description, Price, and CategoryId. The Solution Explorer on the right shows the project structure, including the Models folder where Product.cs is located.

Step 5: Create a ProductRepository for all our operations



The screenshot shows the Visual Studio IDE interface. The main window displays the `ProductRepository.cs` file, which contains the implementation of the `IProductRepository` interface. The code includes methods for updating, deleting, getting products by ID, and inserting products. The `Solution Explorer` pane on the right shows the project structure for `ProductMicroservice`, including files like `Startup.cs`, `Program.cs`, and `Category.cs`.

```

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

    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();
    }
}

```

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;

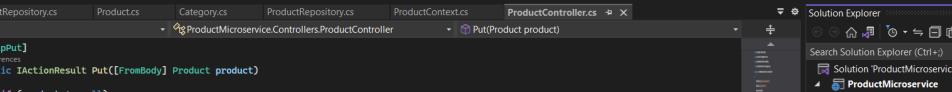
namespace ProductMicroservice.DBContext
{
    public class ProductContext : DbContext
    {
        public ProductContext(DbContextOptions<ProductContext> options) : base(options)
        {
        }

        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



The screenshot shows the Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search. The title bar displays "ProductMicroservice". The main code editor window shows the "ProductController.cs" file with the following code:

```
[HttpPost]
0 references
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}")]
0 references
public IActionResult Delete(int id)
{
    _productRepository.DeleteProduct(id);
    return new OkResult();
}
```

The Solution Explorer on the right side shows the solution structure for "ProductMicroservice" (1 of 2 projects):

- ProductMicroservice
 - Connected Services
 - Dependencies
 - Properties
 - Controller
 - ProductController.cs
 - DBContexts
 - Models
 - Category.cs
 - Products.cs
 - Migrations
 - Repository
 - ProductRepository.cs
 - ProductRepository.cs
 - appsettings.json
 - DotNetCoreClient

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

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "ProductDB": "Server=DESKTOP-MSTB7F\SQLEXPRESS2022;Initial Catalog=ProductsDB;Trusted Connection=True;MultipleActiveResultSets=true"
  }
}
```

Package Manager Console

Package source: All Default project: ProductMicroservice

```
PM> add-migration InitialCreate -verbose
Using project 'ProductMicroservice'.
Using startup project 'ProductMicroservice'.
Build started...
Build succeeded.
C:\Program Files\dotnet\sdk\3.1.401\NuGet.targets(114,5): warning NU1603: Additional assembly 'C:\Users\DeLL\OneDrive\Documents\Rahul Keat\ProjectMicroservice\bin\Debug\netcoreapp3.1\ProductsDB.dll' was found in the output directory. Consider using the 'nuget.g.targets' target file to handle this assembly.
```

```
PM> dotnet ef migrations add InitialCreate -j json -v -o prefix-output --assembly "C:\Users\DeLL\OneDrive\Documents\Rahul Keat\ProjectMicroservice\bin\Debug\netcoreapp3.1\ProductsDB.dll" --start-up-assembly "C:\Users\DeLL\OneDrive\Documents\Rahul Keat\ProjectMicroservice\bin\Debug\netcoreapp3.1\ProductsDB.dll" --language C# --working-dir "C:\Users\DeLL\OneDrive\Documents\Rahul Keat\ProjectMicroservice\"
```

Browser Link Dashboard

- ProductMicroservice (0 connections)

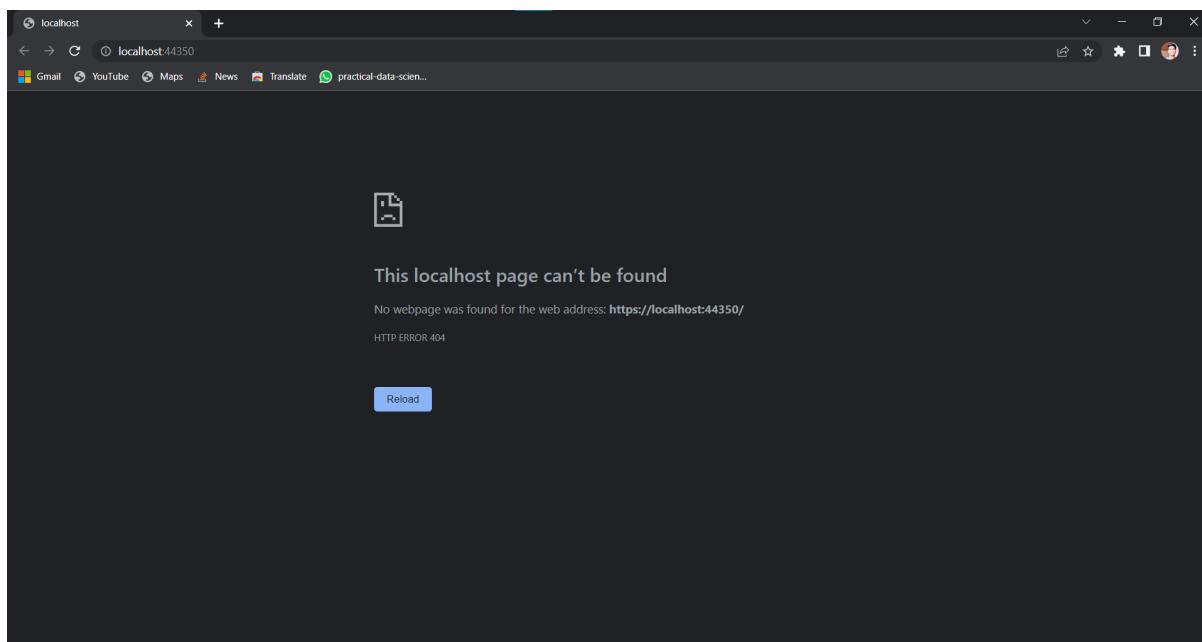
Sign in | Live Share

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

The screenshot shows the Visual Studio interface with the following components:

- Solution Explorer:** Shows the project structure with files like DBContexts.cs, Migrations, Models, Repository, and appsettings.json.
- Code Editor:** Displays the appsettings.json file containing configuration settings for logging and databases.
- Package Manager Console:** Shows the command "PM> update-database -verbose" being run, followed by the output of the database update process.
- Status Bar:** Shows the status "Ready".

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 Microsoft.Extensions.Logging;
using ProductMicroservice.Data;
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>();
        }
    }
}

```

```

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_3_0);
        services.AddDbContext<ProductContext>(o => o.UseSqlServer(Configuration.GetConnectionString("ProductDB")));
        services.AddTransient<Repository.IProductRepository, ProductRepository>();

        // 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();
            });
        }
    }
}

```

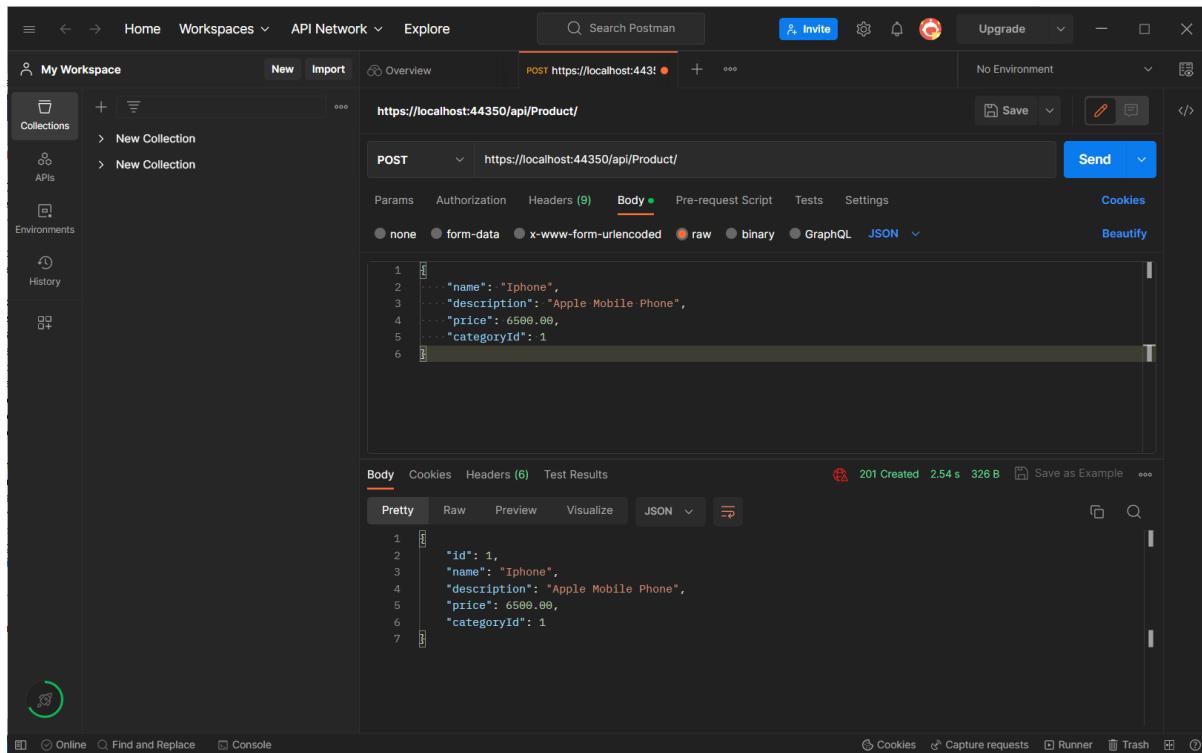
```

            endpoints.MapControllers();
        }

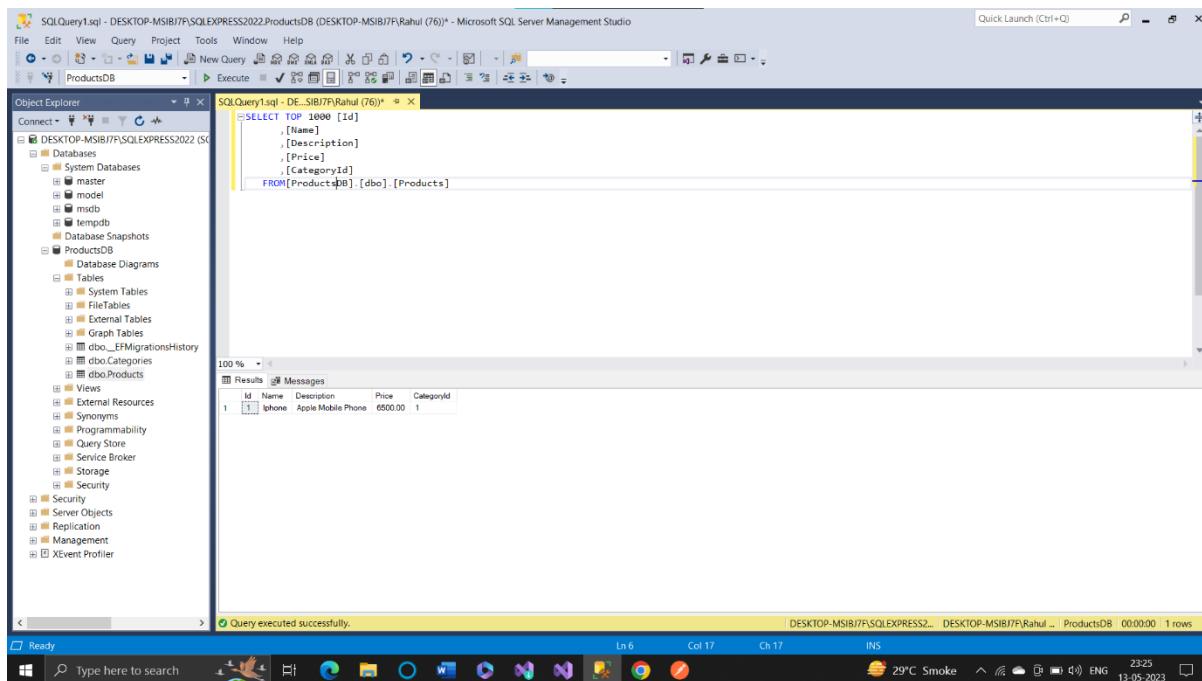
    }
}

```

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



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



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   "name": "Tata Tiago",
3   "description": "Tata Motors",
4   "price": 50000,
5   "categoryId": 12
6 }

```

The response status is 201 Created, and the response body is:

```

1 {
2   "id": 2,
3   "name": "Tata Tiago",
4   "description": "Tata Motors",
5   "price": 50000,
6   "categoryId": 12
7 }

```

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 empty. The response status is 200 OK, and the response body is:

```

1 [
2   {
3     "id": 1,
4     "name": "iPhone",
5     "description": "Apple Mobile Phone",
6     "price": 65000.00,
7     "categoryId": 1
8   },
9   {
10    "id": 2,
11    "name": "Tata Tiago",
12    "description": "Tata Motors",
13    "price": 50000.00,
14    "categoryId": 12
15  }
16 ]

```

Step 17: Delete a product using DELETE

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

```

1   {
2     "id":1,
3     "name":"iPhone",
4     "description":"Apple Mobile Phone",
5     "price":6590.00,
6     "categoryId":1
7   }
  
```

The response status is 405 Method Not Allowed, Time: 116 ms, Size: 174 B.

Step 18: Update a product using the PUT

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

```

1   {
2     "id":3,
3     "name":"Tata nexon",
4     "description":"Tata Motors",
5     "price":1590000.00,
6     "categoryId":1
7   }
  
```

The response status is 200 OK, Time: 109 ms, Size: 135 B.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer on the left, a database named 'ProductsDB' is selected under 'DESKTOP-MSIBJ7F\SQLEXPRESS2022'. A query window titled 'SQLQuery2.sql' is open, displaying the following T-SQL code:

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

The results pane shows a table with four rows of data:

	Id	Name	Description	Price	CategoryId
1	1	Iphone	Apple Mobile Phone	6500.00	1
2	2	Tata Tago	Tata Motors	500000.00	12
3	3	TaTa nexon	TaTa Motors	1500000.00	1
4	4	Iphone	Apple Mobile Phone	7500.00	1

At the bottom of the screen, the taskbar shows various application icons and the system tray displays weather information (29°C Haze) and system status.

Microservices using spring cloud config server.

Microservices using spring cloud config server.

-Mohd Kaif
M.Sc. I.T. Part 1
22001

Introduction to Spring Cloud & Config Server

Spring Cloud is a set of libraries and tools that help developers build and manage microservices-based applications on top of the Spring Framework. It provides a comprehensive set of tools and libraries that make it easy to build and manage cloud-native applications.

Spring Cloud Config Server is a tool in the Spring Cloud ecosystem that provides centralized configuration management for microservices-based applications. It allows for the externalization and management of application configuration in a distributed environment, and enables microservices to be configured dynamically without the need for redeployment. Spring Cloud Config Server is designed to work seamlessly with other Spring Cloud tools and technologies, such as Eureka for service discovery and Ribbon for load balancing. Spring Cloud Config Server is designed to work seamlessly with other Spring Cloud tools and technologies, such as Eureka for service discovery and Ribbon for load balancing. It can also be integrated with other third-party tools, such as HashiCorp Vault for secure configuration management

Introduction to Microservices

Definition of microservices:

Microservices is an architectural style for building software applications as a collection of small, independent services that communicate with each other over a network. Each microservice is self-contained and focused on performing a specific business function, and can be developed, deployed, and scaled independently from the rest of the system.

Advantages of microservices:

- Greater agility and flexibility: Microservices allow for faster and more flexible development and deployment, as each service can be developed, tested, and deployed independently.
- Improved scalability: Microservices can be scaled independently based on the specific demand for each service, leading to more efficient resource utilization and cost savings.
- Improved fault tolerance: Microservices are more resilient to failures, as a failure in one service does not necessarily affect the entire system.
- Improved technology diversity: Microservices allow for the use of different technologies for each service, which can improve performance, scalability, and functionality.

Benefits of using Spring Cloud Config Server:

- Dynamic configuration: Spring Cloud Config Server allows for dynamic configuration of microservices, which means that changes can be made without the need for redeployment.
- Centralized management: Spring Cloud Config Server provides a centralized location for storing and managing configuration information, which can improve collaboration and version control.
- Hierarchical configuration: Spring Cloud Config Server supports hierarchical configuration, which allows developers to define different configuration files for different environments and profiles.
- Integration with other Spring Cloud tools: Spring Cloud Config Server is designed to work seamlessly with other Spring Cloud tools, such as Eureka for service discovery and Ribbon for load balancing.

How Spring Cloud Config Server works:

- Spring Cloud Config Server pulls configuration files from a Git repository.
- The configuration files are stored in different formats, such as YAML, JSON, and properties files.
- Microservices pull their configuration information from Spring Cloud Config Server.
- Spring Cloud Config Server can be configured to refresh the configuration information automatically, or on-demand.

Spring Cloud Config Server architecture

Spring Cloud Config Server follows a client-server architecture, where the Config Server component serves as the server and the Config Client component serves as the client.

The Config Server is responsible for serving configuration files to the Config Client. It pulls the configuration files from a Git repository, which allows for versioning, auditing, and collaboration. The Config Server can be configured to use different Git repository providers, such as GitHub or Bitbucket. When a microservice requests its configuration information from the Config Server, the Config Server first checks its cache to see if the requested configuration information is already available. If the configuration information is not available in the cache, the Config Server pulls it from the Git repository and stores it in its cache. The Config Server can be configured to refresh its cache automatically or on-demand.

Spring Cloud Config Server supports different modes of operation, depending on the needs of the application. In the default mode, the Config Server serves as a standalone service that runs on its own. In the embedded mode, the Config Server is embedded into a Spring Boot application, which can simplify deployment and configuration. Spring Cloud Config Server can also be clustered for high availability and scalability, by running multiple instances of the Config Server and using a load balancer.

Setting up Spring Cloud Config Server

Prerequisites for setting up Spring Cloud Config Server:

- Spring Boot 2.x or higher
- Java 8 or higher
- A Git repository for storing configuration files
- Knowledge of Git and version control

Installing and configuring Spring Cloud Config Server:

Add the Spring Cloud Config Server dependency to your project's build file, such as Gradle or Maven. Create a Spring Boot application and annotate it with the `@EnableConfigServer` annotation to enable Spring Cloud Config Server.

Configure the application to use the Git repository for storing configuration files, by specifying the Git repository URL, username, password, and other settings in the `application.yml` or `application.properties` file.

Run the Spring Boot application to start the Spring Cloud Config Server.

Creating a Git repository for storing configuration files:

- Create a Git repository on your preferred Git repository provider, such as GitHub or Bitbucket.
- Create a folder in the repository for storing configuration files. The folder should be named after the application name and should have a suffix of -config, such as my-application-config.
- Create configuration files for each environment and profile of the application, using the appropriate file format, such as YAML or properties files.
- Commit the configuration files to the Git repository and push the changes to the remote repository.

In summary, setting up Spring Cloud Config Server involves installing and configuring the Spring Boot application, creating a Git repository for storing configuration files, and creating configuration files for each environment and profile of the application. With these steps in place, Spring Cloud Config Server can serve as a centralized configuration management tool for microservices-based applications.

Configuring microservices using Spring Cloud Config Server

Introduction to Spring Cloud Config Client:

- Spring Cloud Config Client is a component that enables microservices to retrieve their configuration information from Spring Cloud Config Server.
- Spring Cloud Config Client is a library that can be added to microservices as a dependency, just like any other library.

Configuring microservices to use Spring Cloud Config Server:

- Add the Spring Cloud Config Client dependency to your microservice's build file.
- Configure the microservice to use Spring Cloud Config Server by specifying the Config Server URL and other settings in the bootstrap.yml or bootstrap.properties file.
- Access the configuration information in your microservice by using Spring's @Value annotation or the Environment object.

Spring Cloud Config Client architecture:

- The Spring Cloud Config Client is responsible for communicating with the Spring Cloud Config Server and retrieving the configuration information.
- The Spring Cloud Config Client caches the configuration information locally, which can improve performance and reduce the load on the Config Server.
- The Spring Cloud Config Client can be configured to refresh its configuration information on-demand or automatically, based on a schedule or other triggers.

Loading configuration files from Spring Cloud Config Server:

- Spring Cloud Config Server serves configuration files to microservices in a format that is compatible with Spring's Environment object, such as YAML or properties files.
- Microservices can load configuration files from Spring Cloud Config Server using the @Value annotation or the Environment object, just like they would load configuration files from the file system or environment variables.
- Spring Cloud Config Server supports different ways of organizing configuration files, such as by application name, profile, and label, which can enable fine-grained control over configuration information.

Service discovery with Spring Cloud Config Server

Introduction to service discovery:

- Service discovery is a mechanism for automatically locating and accessing services in a distributed system.
- In a microservices-based application, service discovery can be used to simplify communication between microservices and enable dynamic scaling and deployment.

Integrating Spring Cloud Config Server with Eureka:

- Spring Cloud Config Server can be integrated with Eureka to enable centralized configuration management for services.
- By registering with Eureka and using Spring Cloud Config Server as a configuration source, microservices can dynamically retrieve their configuration information and adapt to changes in the environment.
- Spring Cloud Config Server can also be configured to refresh the configuration information for a service based on changes in the Eureka registry, which can enable dynamic and reactive behavior in the application.

Service registration and discovery with Eureka:

- Eureka is a service registry and discovery server that is part of the Spring Cloud ecosystem.
- Eureka enables microservices to register themselves with the registry server, which can then be used by other microservices to discover and communicate with them.
- Eureka supports different modes of registration and discovery, such as by IP address or hostname, and can handle scenarios such as service failures and load balancing.

Load balancing and fault tolerance with Eureka:

- Eureka can be used as a load balancer for microservices, by distributing incoming requests across multiple instances of the same service.
- Eureka also supports fault tolerance and resiliency features, such as by monitoring the health of services and removing failed instances from the registry.
- By integrating with Eureka, microservices can benefit from these features and achieve high availability and scalability in a distributed environment.

Conclusion

Recap of the benefits of using Spring Cloud Config Server for microservices-based applications:

- Spring Cloud Config Server enables centralized and dynamic configuration management for microservices, which can simplify deployment and reduce downtime.
- By separating configuration from code, Spring Cloud Config Server can enable faster iterations and experimentation in the development process.
- Spring Cloud Config Server integrates with other tools in the Spring Cloud ecosystem, such as Eureka, to enable features such as service discovery and load balancing.

Future trends and directions for Spring Cloud Config Server:

- Spring Cloud Config Server is constantly evolving and improving, with new features and integrations being added regularly.
- Some of the emerging trends in the Spring Cloud Config Server ecosystem include support for Kubernetes and other container orchestration platforms, and integration with other configuration management tools such as HashiCorp Vault.

- Additional resources for learning more about Spring Cloud Config Server:
 - The Spring Cloud website (<https://spring.io/projects/spring-cloud>) is a great resource for getting started with Spring Cloud and learning about the different tools and features available.
 - The Spring Cloud Config Server documentation (<https://docs.spring.io/spring-cloud-config/docs/current/reference/html/>) provides detailed information on how to install, configure, and use Spring Cloud Config Server.
 - The Spring Cloud Config Server GitHub repository (<https://github.com/spring-cloud/spring-cloud-config>) contains the source code for the project and provides a platform for contributing to the development of the tool.

In summary, Spring Cloud Config Server is a powerful tool for managing configuration in microservices-based applications. By enabling centralized and dynamic configuration management, Spring Cloud Config Server can simplify deployment and improve application resiliency. With ongoing development and support from the Spring community, Spring Cloud Config Server is poised to remain a key tool for building modern, distributed applications.

Thank you.