MICROSERVICES ARCHITECTURE

A PRACTICAL REPORT

ON

MICROSERVICES ARCHITECTURE

SUBMITTED BY
Mr. RAHUL KEWAT
Roll No: 22006

UNDER THE GUIDANCE OF
PROF. MEHDI REZAEI

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

University of Mumbai
Department of Information Technology

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

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

Bandra (W), Mumbai – 400050.

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

Certificate

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

Subject In-Charge **Coordinator In-Charge** **External Examiner**

College Stamp

INDEX

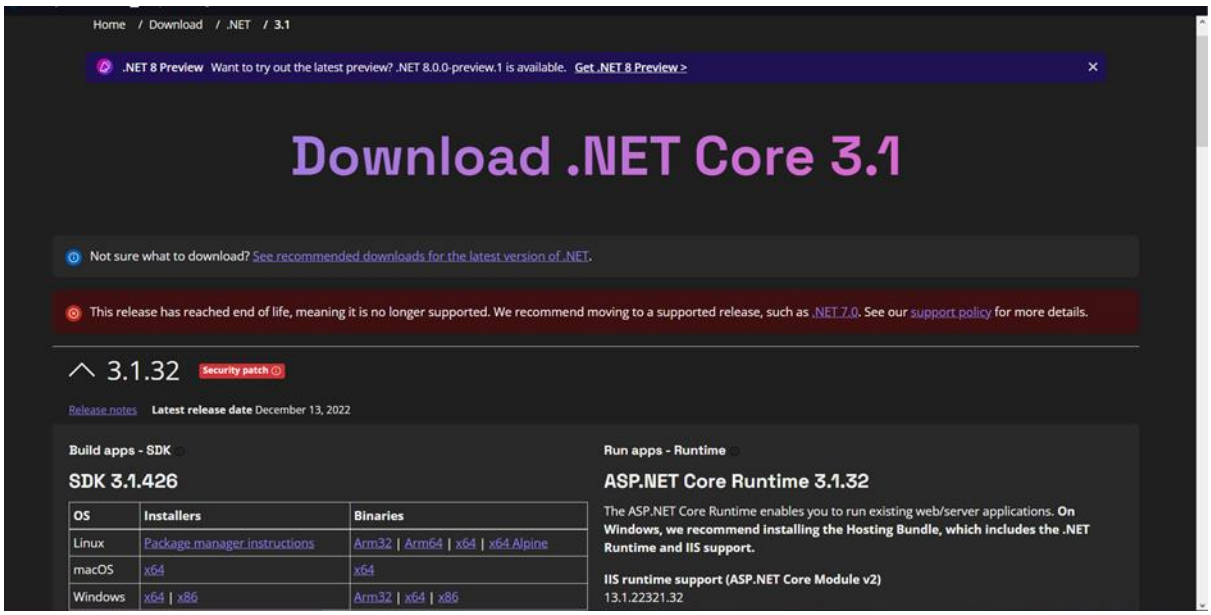
Practical 1

Aim: Creating a basic MVC application using .NET CORE

Writeup:

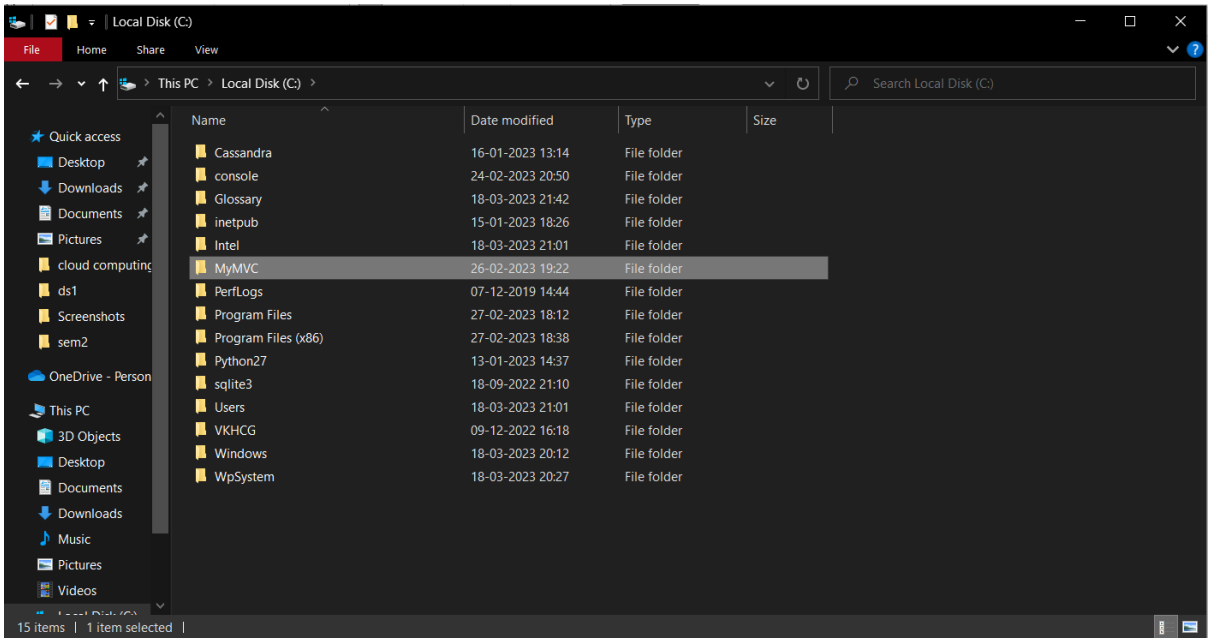
Step 1:

- Install .Net Core SDK.



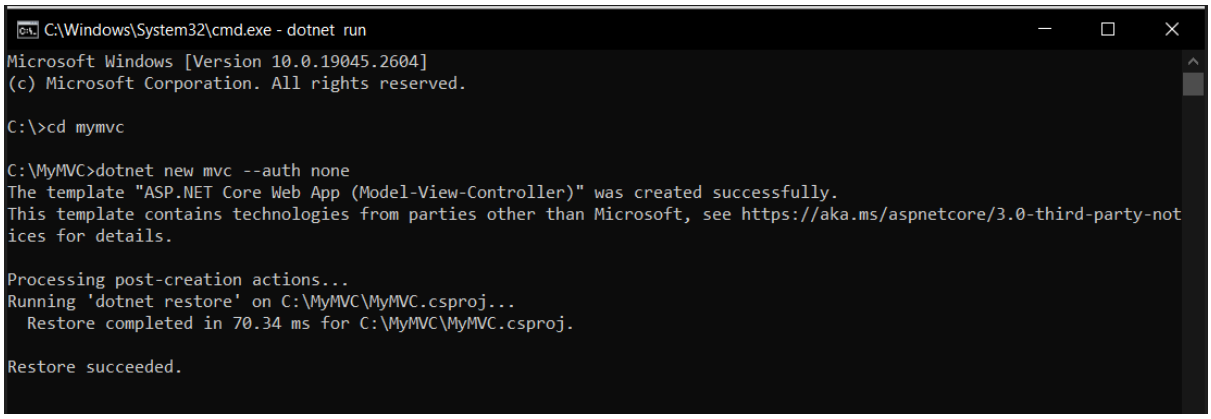
Step 2:

- Create a Folder MyMVC in C: drive.



Step 3:

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



Step 4:

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

HomeController.cs

Controllers > HomeController.cs

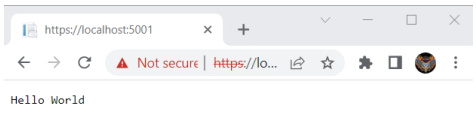
```
1   using System;
2   using System.Collections.Generic;
3   using System.Diagnostics;
4   using System.Linq;
5   using System.Threading.Tasks;
6   using Microsoft.AspNetCore.Mvc;
7   using Microsoft.Extensions.Logging;
8   using MyMVC.Models;
9
10  namespace MyMVC.Controllers
11  {
12      public 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
```

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



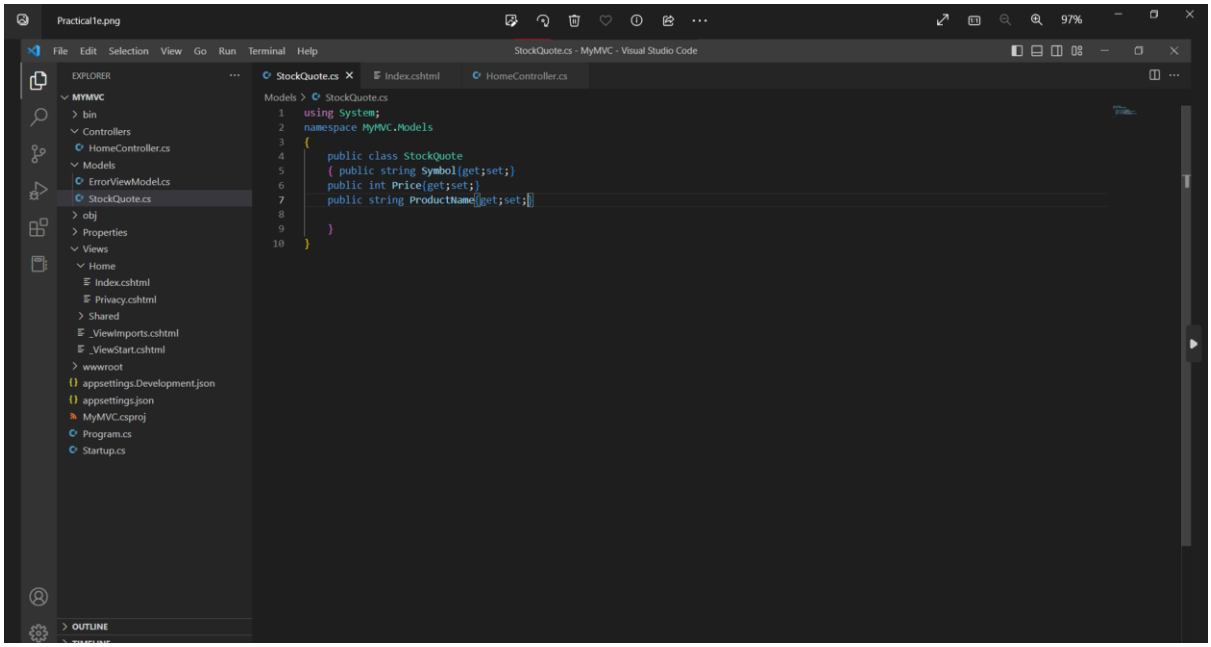
Step 6:

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

```
C:\Windows\System32\cmd.exe

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

C:\MyMVC>
```

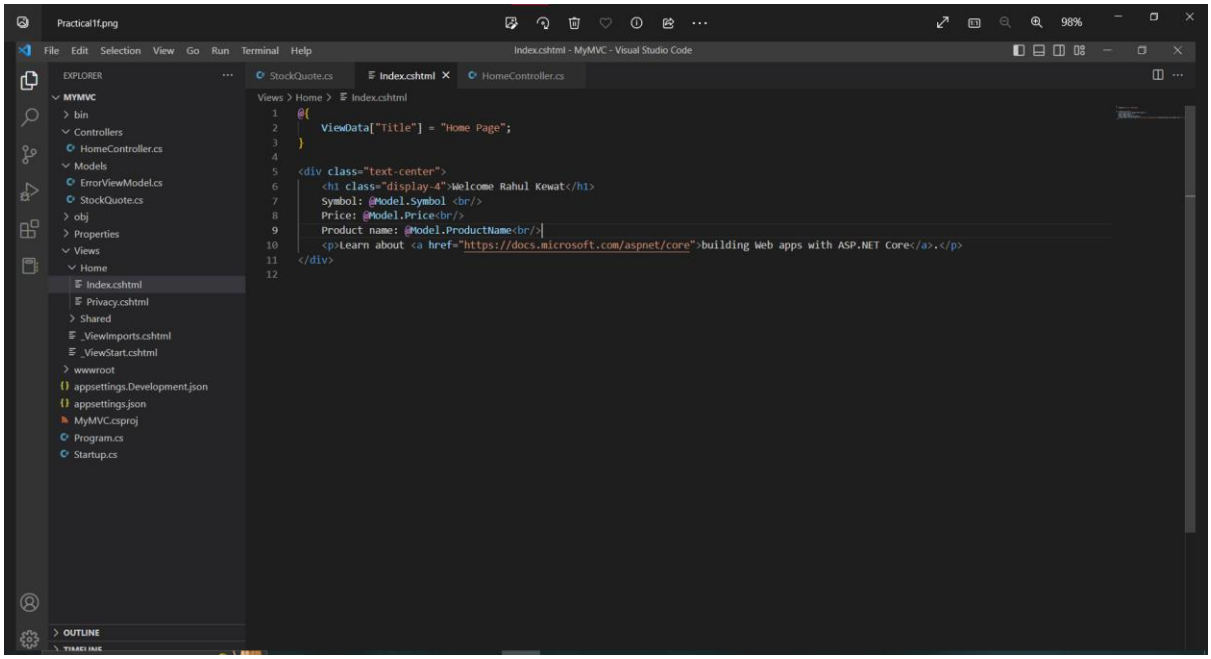
Step 7:

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



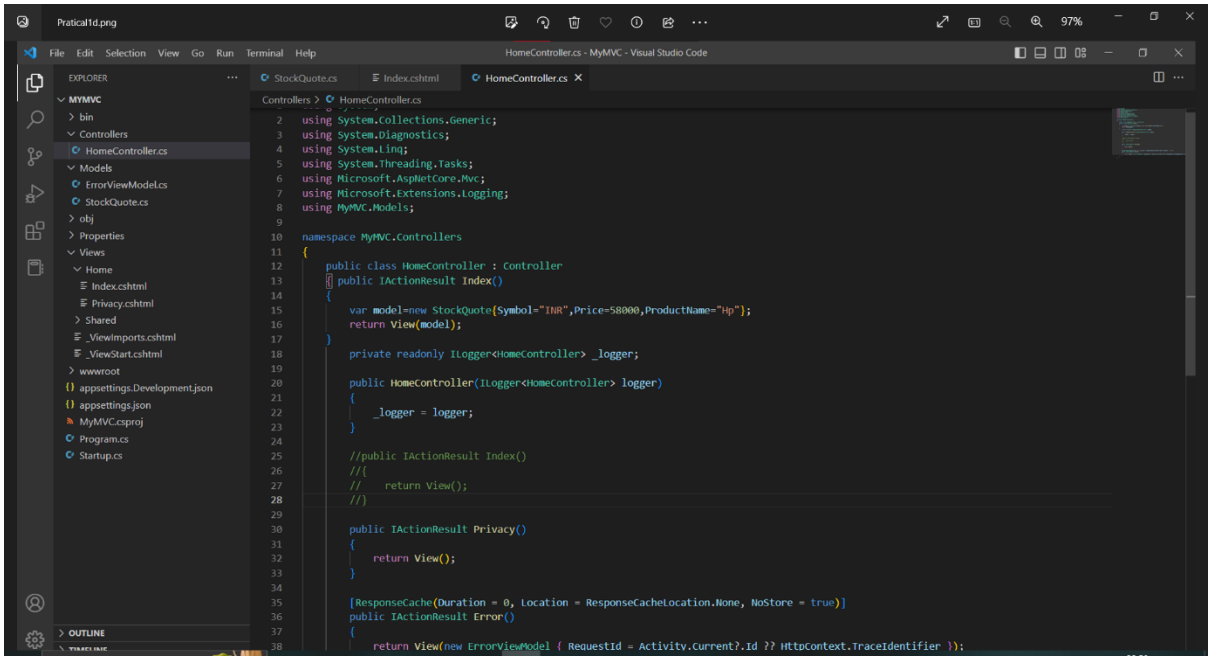
Step 8:

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



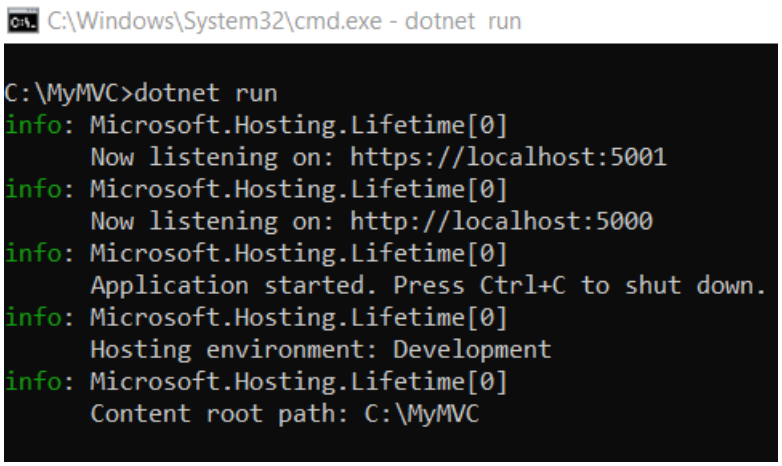
Step 9:

- Now modify HomeController.cs file to match following:



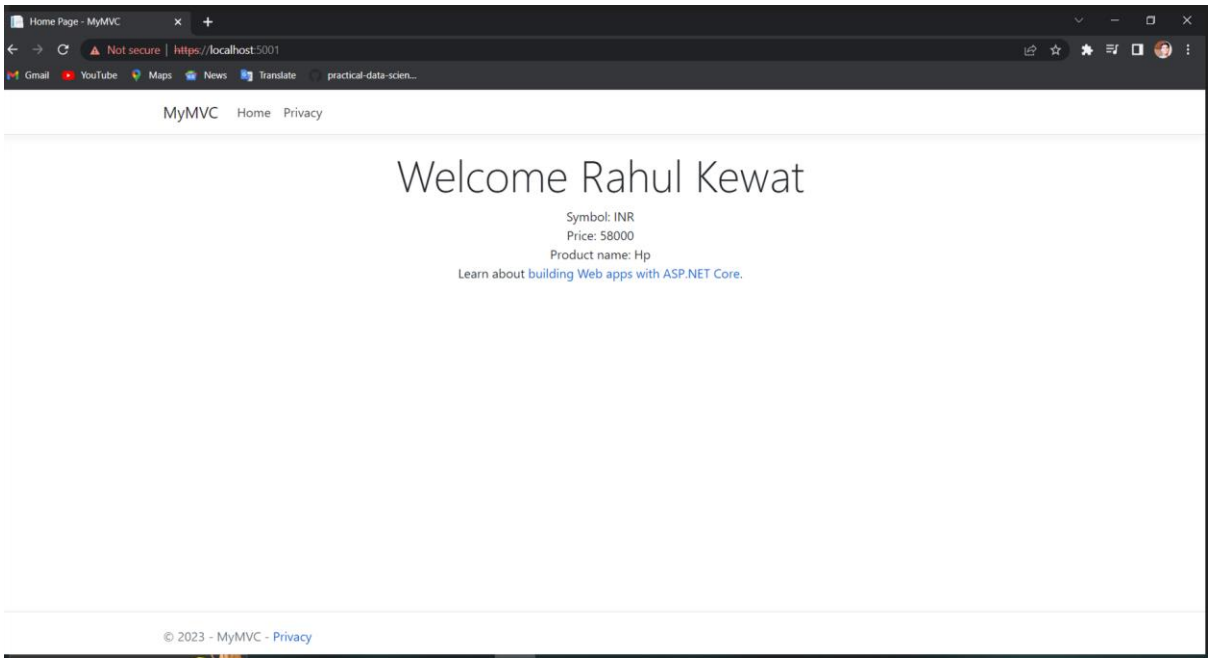
Step 10:

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



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:

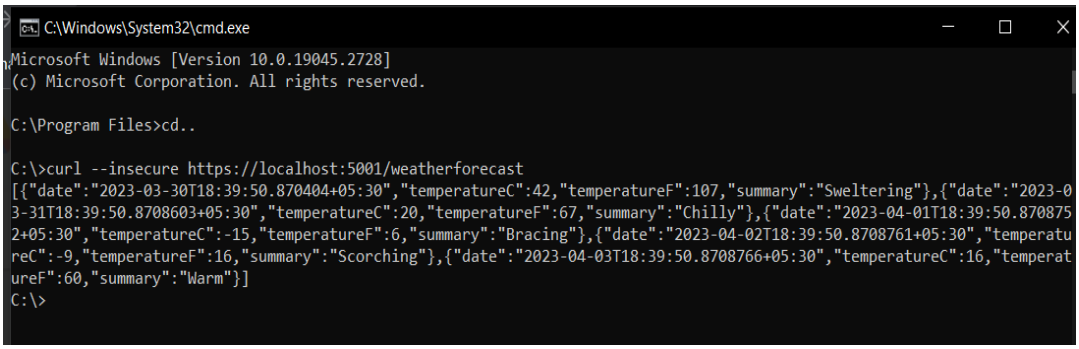
cd Glossary

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)

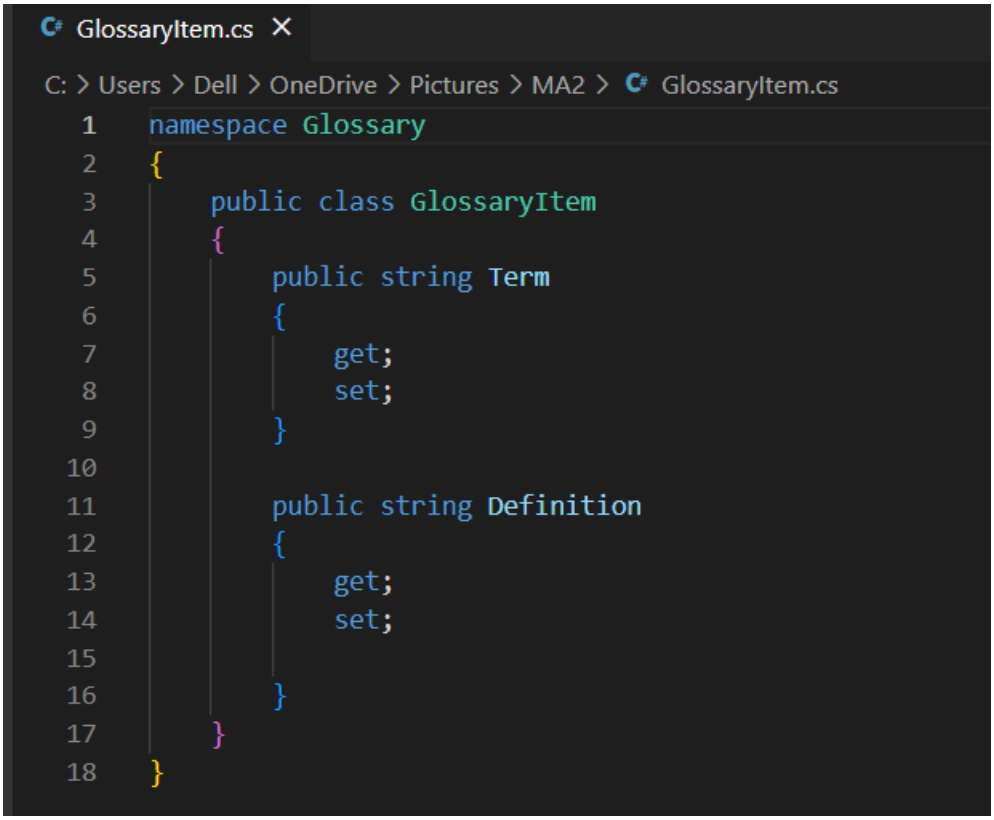
curl --insecure https://localhost:5001/weatherforecast



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



[B]: GlossaryController.cs

C: > Users > Dell > OneDrive > Pictures > MA2 > GlossaryController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;

namespace Glossary.Controllers
{
    [ApiController]
    [Route("api/[controller]")]

     public class GlossaryController: ControllerBase
     {
        private static List<GlossaryItem> Glossary = new List<GlossaryItem>
        {
            new GlossaryItem
            {
                Term= "HTML",
                Definition = "Hypertext Markup Language"
            },

            new GlossaryItem
            {
                Term= "MVC",
                Definition = "Model View Controller"
            },

            new GlossaryItem
            {
                Term= "OpenID",
                Definition = "An open standard for authentication"
            }
        };

```

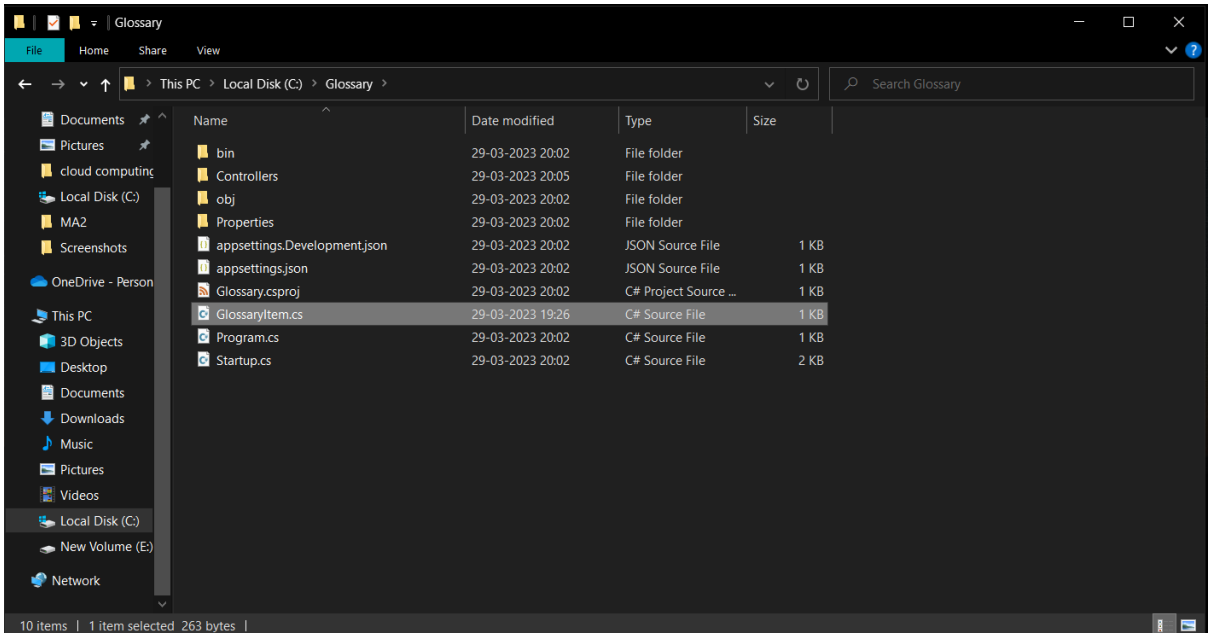
C: > Users > Dell > OneDrive > Pictures > MA2 > GlossaryController.cs

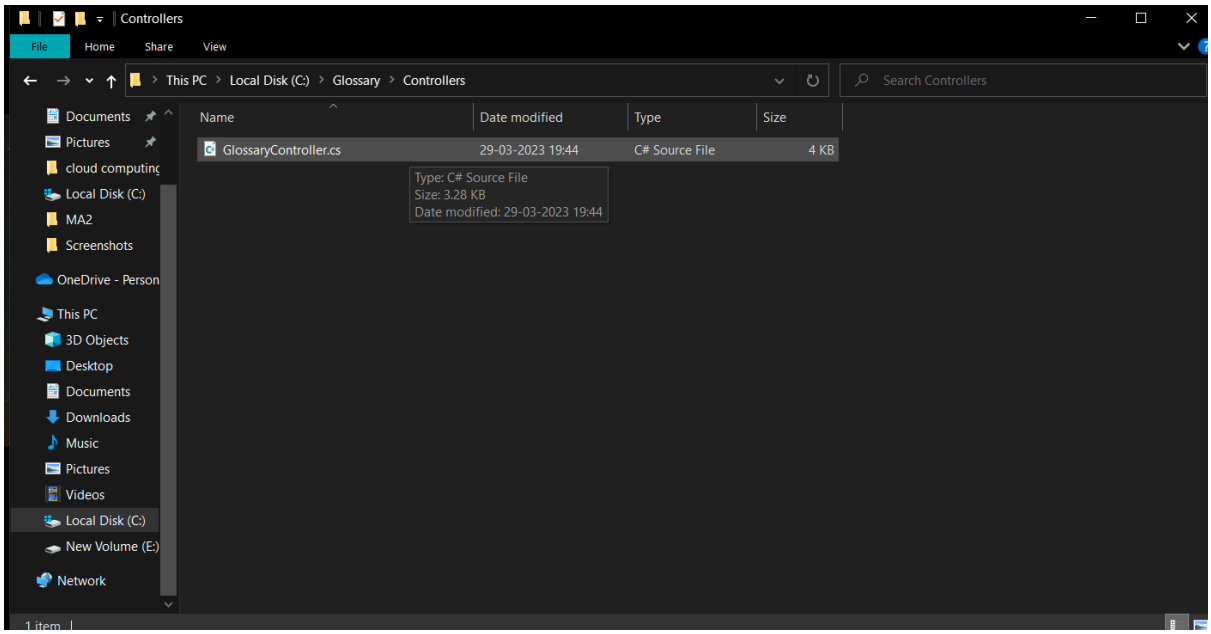
```
36          [HttpGet]
37          public ActionResult<List<GlossaryItem>> Get()
38          {
39              return Ok(Glossary);
40          }
41
42          [HttpGet]
43          [Route("{term}")]
44          public ActionResult<GlossaryItem> Get(string term)
45          {
46              var 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

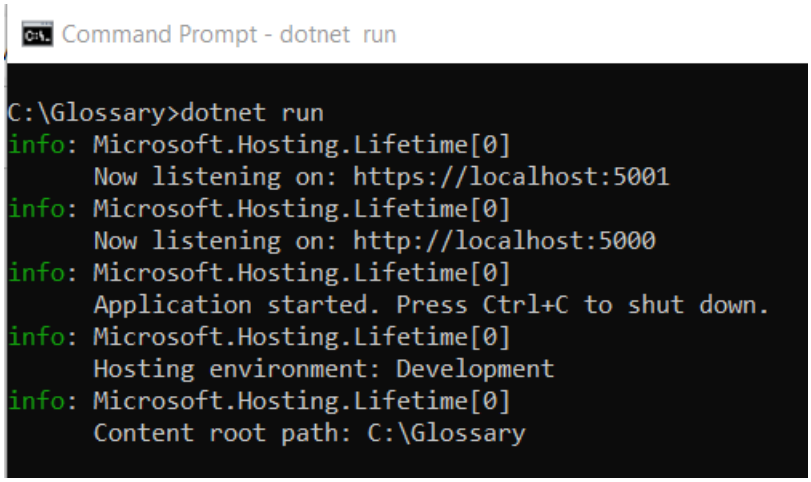
```
            else
            {
                Glossary.Add(glossaryItem);
                var resourceUrl = Path.Combine(Request.Path.ToString(), Uri.EscapeUriString(glossaryItem.
                return Created(resourceUrl, glossaryItem);
            }
        }

        [HttpPut]
        public ActionResult Put(GlossaryItem glossaryItem)
        {
            var existingGlossaryItem = Glossary.Find(item =>
            item.Term.Equals(glossaryItem.Term, StringComparison.InvariantCultureIgnoreCase));
            if (existingGlossaryItem == null)
            {
                return BadRequest("Cannot update a nont existing term.");
            }

            else
            {
                existingGlossaryItem.Definition = glossaryItem.Definition;
                return Ok();
            }
        }
        [HttpDelete]
        [Route("{term}")]
        public ActionResult Delete(string term)
        {
            var glossaryItem = Glossary.Find(item =>
            item.Term.Equals(term, StringComparison.InvariantCultureIgnoreCase));
            if (glossaryItem == null)
            {
                return NotFound();
            }

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

```

Output:





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



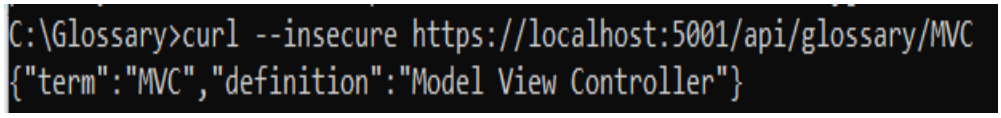
Step 5: Run commands to perform certain operations on the **GlossaryItem** dataset.

[A]: Getting a list of items.

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

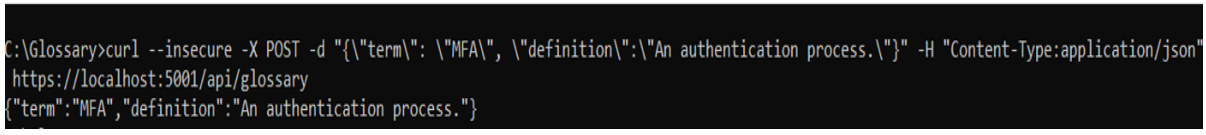
[B]: Getting a single item.

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



[C]: Creating an item.

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



[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:app
lication/json" https://localhost:5001/api/glossary

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

[E]: Delete an item.

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

```
C:\Glossary>curl --insecure --request DELETE --url https://localhost:5001/api/glossary/openid

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

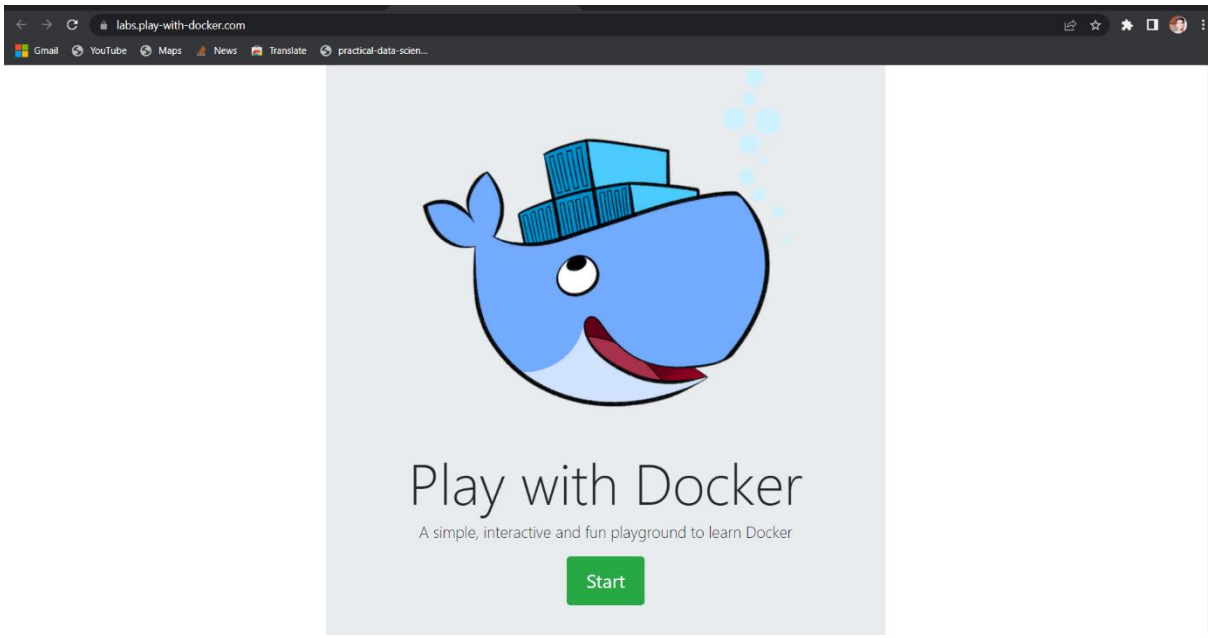
Practical 3

Aim: Working with Docker

Writeup:

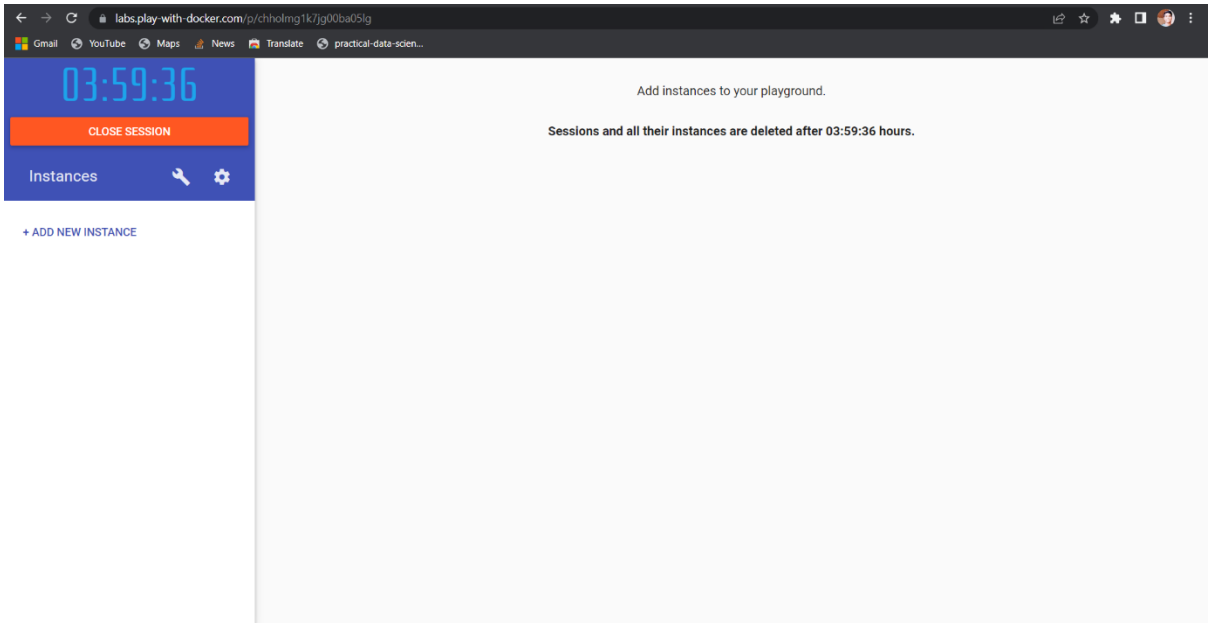
Step 1: create Docker Hub account (sign up)

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



Click on start

Step 3: add new instance



Step 4: perform following:

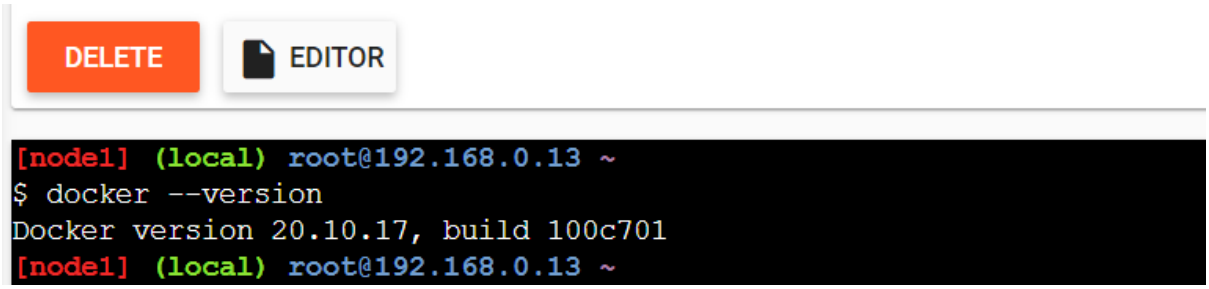
Method1:

To pull and push images using docker

Command: to check docker version

docker –version

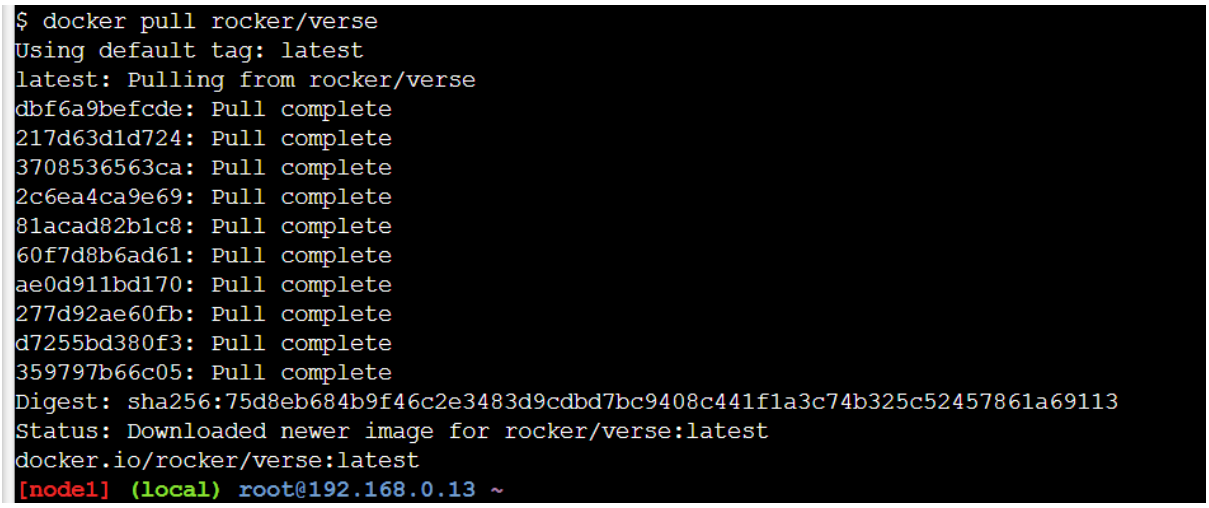
Output:



Command: to pull readymade image

docker pull rocker/verse

Output:



Command: to check images in docker

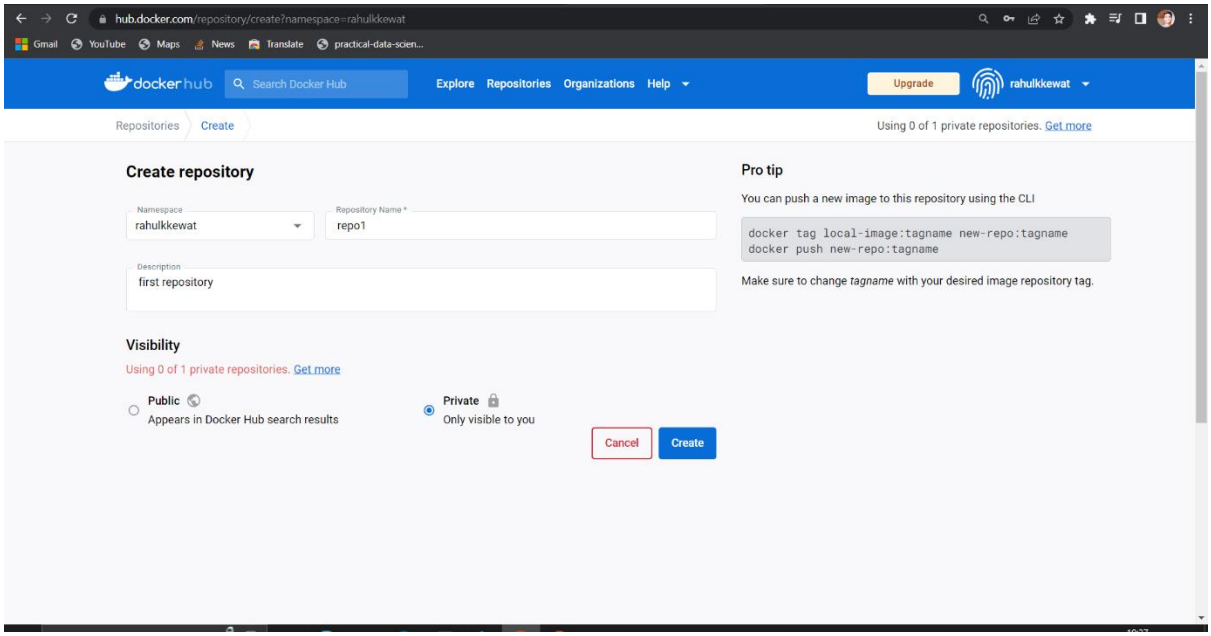
docker images

Output:

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

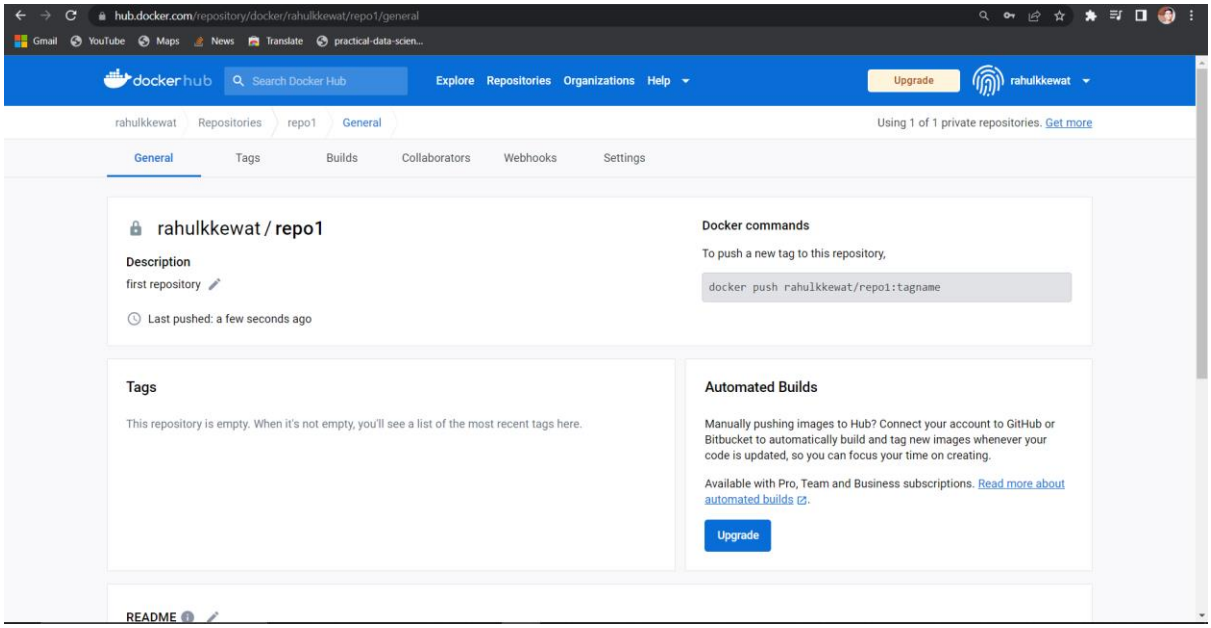
Now Login to docker hub and create repository

Output:



Click on Create button

Now check repository created



Command: to login to your docker account

docker login –username=rahulkkewat

password:

Output:

```
[node1] (local) root@192.168.0.13 ~
$ docker login --username=rahulkkewat
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

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

Command : to tag image

docker tag 9d0311b60ec6 rahulkkewat/repo1:firsttry

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

Output:

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

Command: to push image to docker hub account

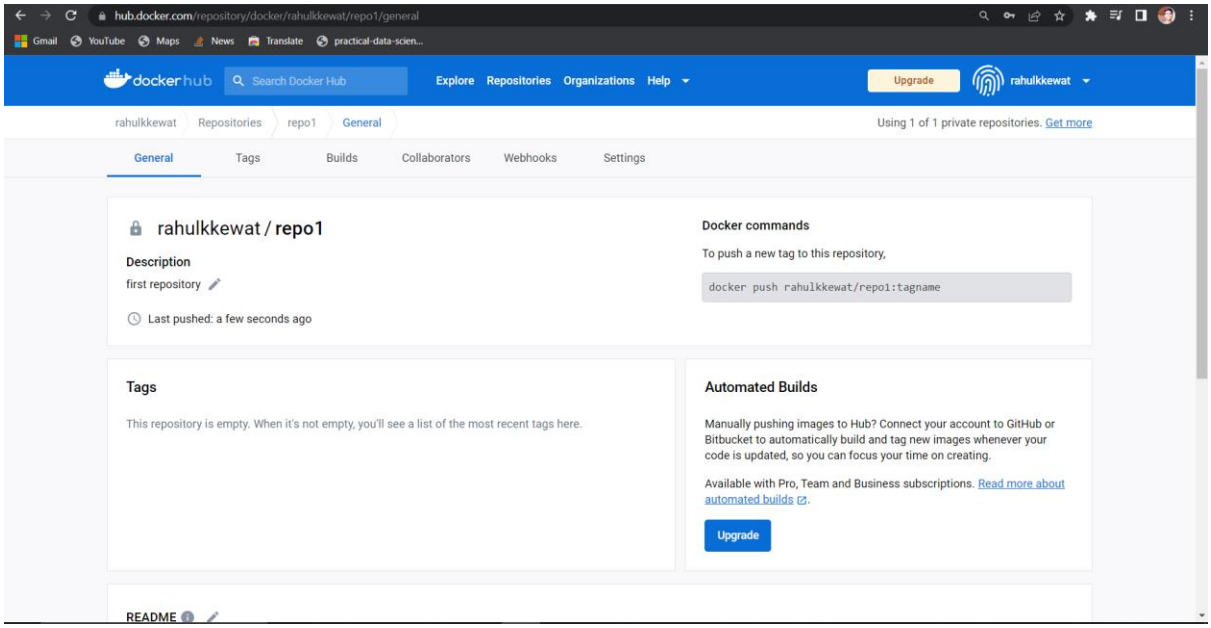
docker push rahulkkewat/repo1:firsttry

note: firsttry is tag name created above.

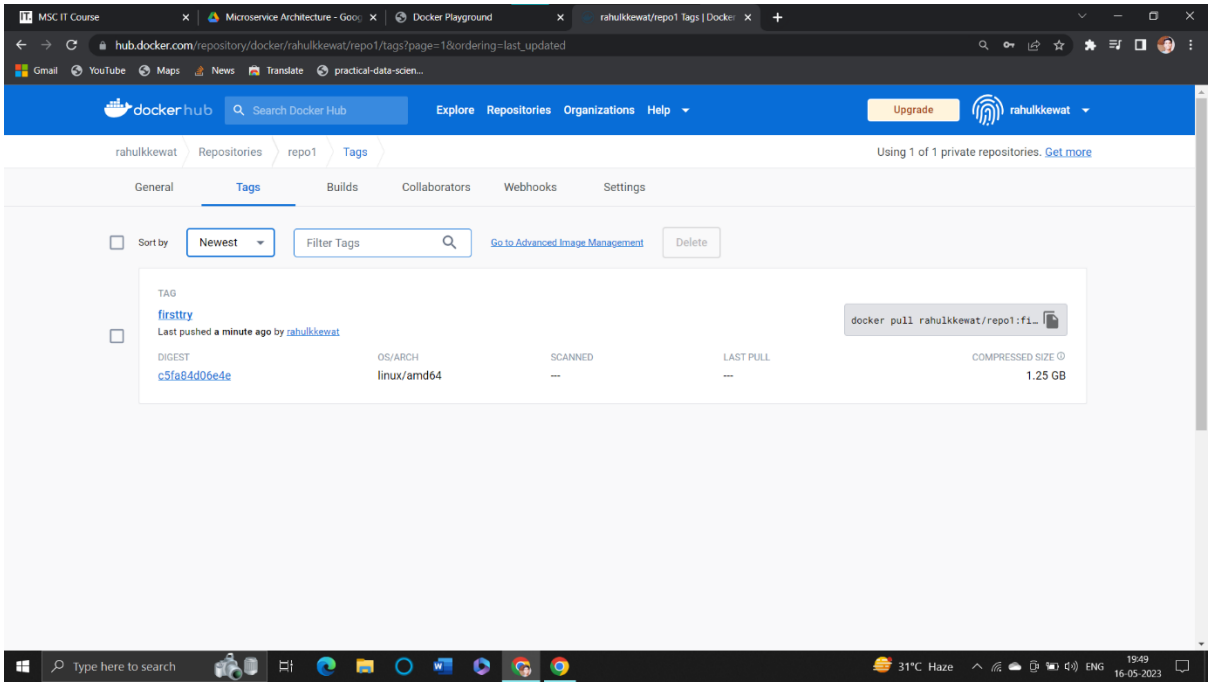
Output:

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

Check it in docker hub now



Click on tags and check



Method 2:

Build an image then push it to docker and run it

Command : to create docker file

1. cat > Dockerfile <<EOF

2. FROM busybox

3. CMD echo "Hello world! This is my first Docker image."

4. EOF

Output:

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

Usage:  docker build [OPTIONS] PATH | URL | -

Build an image from a Dockerfile
[node1] (local) root@192.168.0.13 ~
```

Command : to build image from docker file

dokcer build –t rahulkkewat/repo2 .

Output:

```
[node1] (local) root@192.168.0.13 ~
$ docker build -t rahulkkewat/repo2 .
Sending build context to Docker daemon   12.8kB
Step 1/2 : FROM busybox
latest: Pulling from library/busybox
a58ecd4f0c86: Pull complete
Digest: sha256:9e2bbca079387d7965c3a9cee6d0c53f4f4e63ff7637877a83c4c05f2a666112
Status: Downloaded newer image for busybox:latest
 ---> af2c3e96bcf1
Step 2/2 : CMD echo "Hello world! This is my first Docker image."
 ---> Running in 1518d05eeeee
Removing intermediate container 1518d05eeeee
 ---> 773f1f708953
Successfully built 773f1f708953
Successfully tagged rahulkkewat/repo2:latest
[node1] (local) root@192.168.0.13 ~
```

Command: to check docker images

docker images

```
[node1] (local) root@192.168.0.13 ~
$ docker images
REPOSITORY          TAG         IMAGE ID       CREATED          SIZE
rahulkkewat/repo2   latest      773f1f708953   25 seconds ago   4.86MB
rahulkkewat/repo1   firsttry    9d0311b60ec6   3 days ago       3.43GB
rocker/verse        latest      9d0311b60ec6   3 days ago       3.43GB
busybox             latest      af2c3e96bcf1   4 days ago       4.86MB
[node1] (local) root@192.168.0.13 ~
```

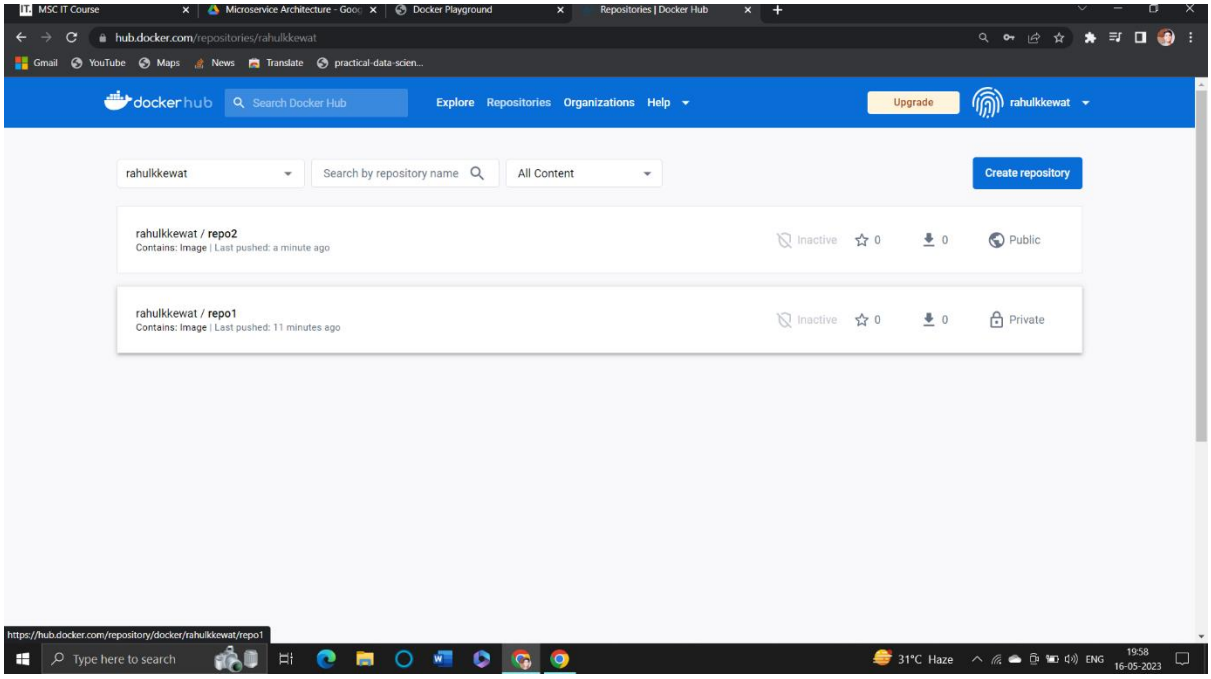
Command: to push image to docker hub

docker push rahulkkewat/repo2

Output:

```
$ docker push rahulkkewat/repo2
Using default tag: latest
The push refers to repository [docker.io/rahulkkewat/repo2]
1f1d08b81bbe: Mounted from library/busybox
latest: digest: sha256:91a3413f9966bb9ccd39199c79d14c886bf799fa8e671681a7daf6f37c561ca7 size: 528
```

Now check it on docker hub



command: to run docker image:

docker run rahulkkewat/repo2

output:

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

Practical 4

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

Writeup:

Step 1: Working with Basic Functionalities Docker

[A]: Creating a volume using the **docker volume** command.

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

C:\Users\Dell>docker volume

Usage:  docker volume COMMAND

Manage volumes

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

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

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

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

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

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

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

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

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

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

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

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

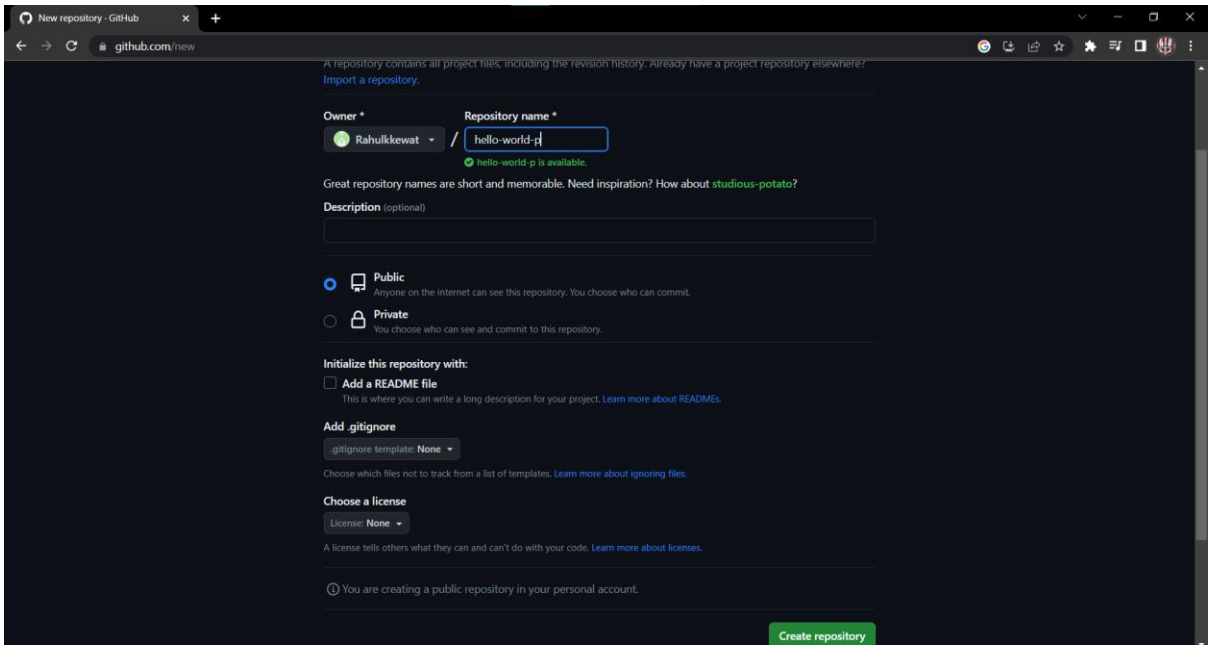
1. Log in to GitHub and begin the process to create a new repository.

2. Enter a name for your repository (for example, hello-world).

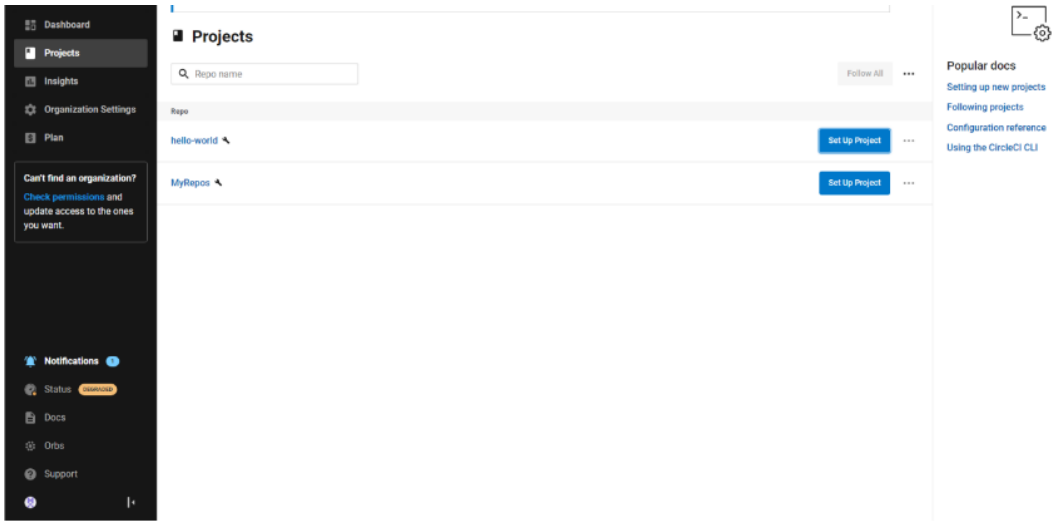
3. Select the option to initialize the repository with a README file.

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.

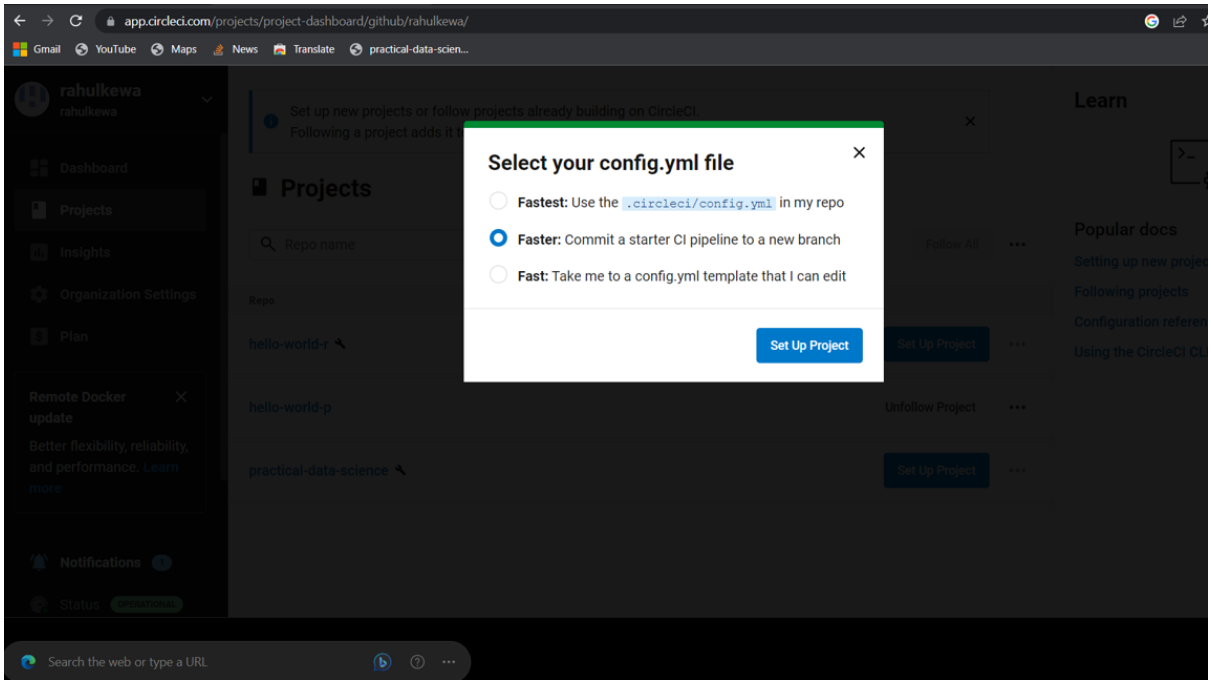


Step 2: Set up CircleCI

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:

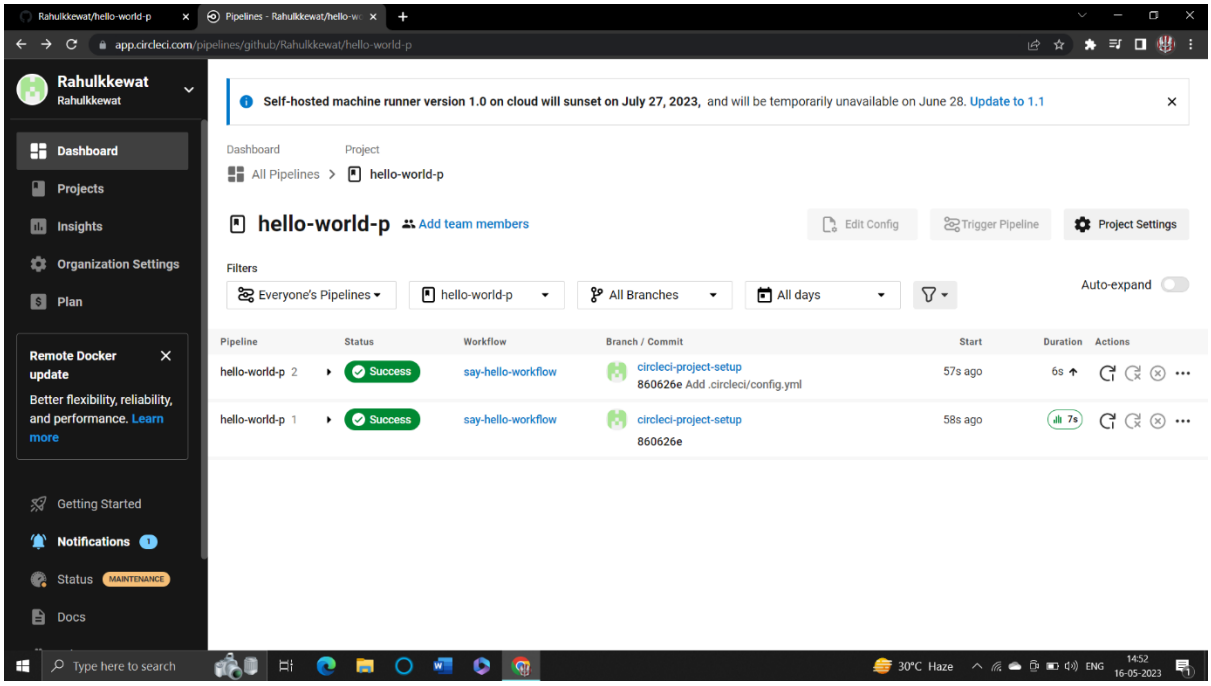
a. Spin up environment

b. Preparing environment variables

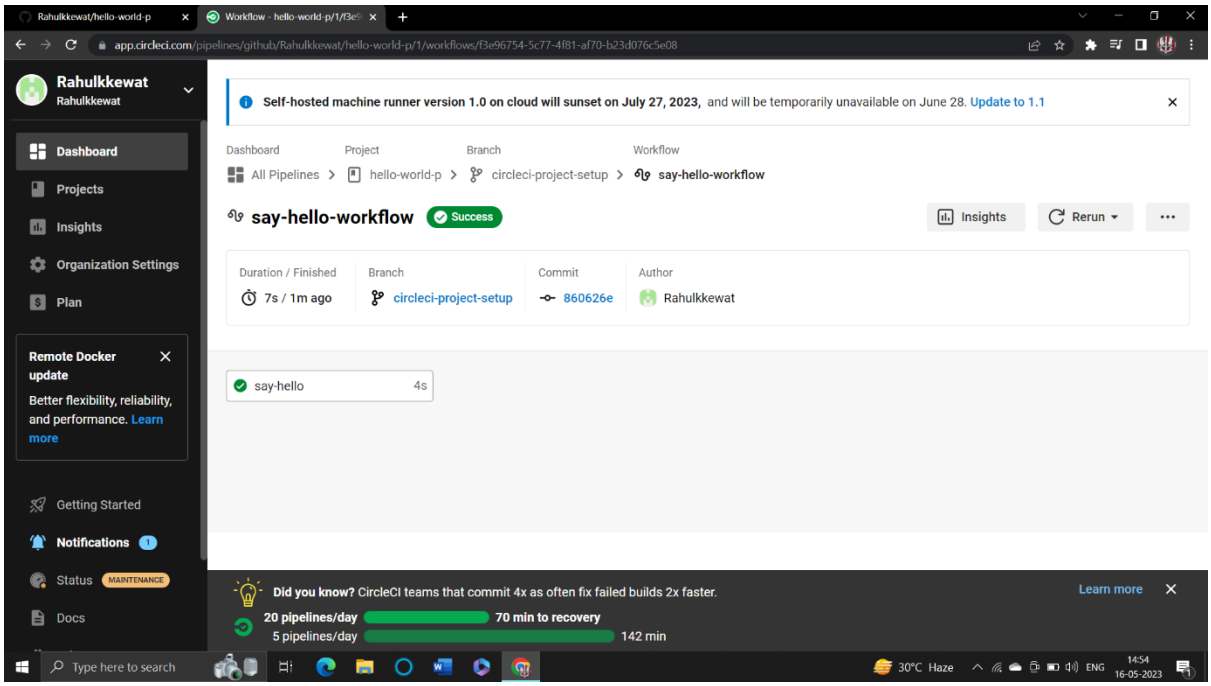
c. Checkout code

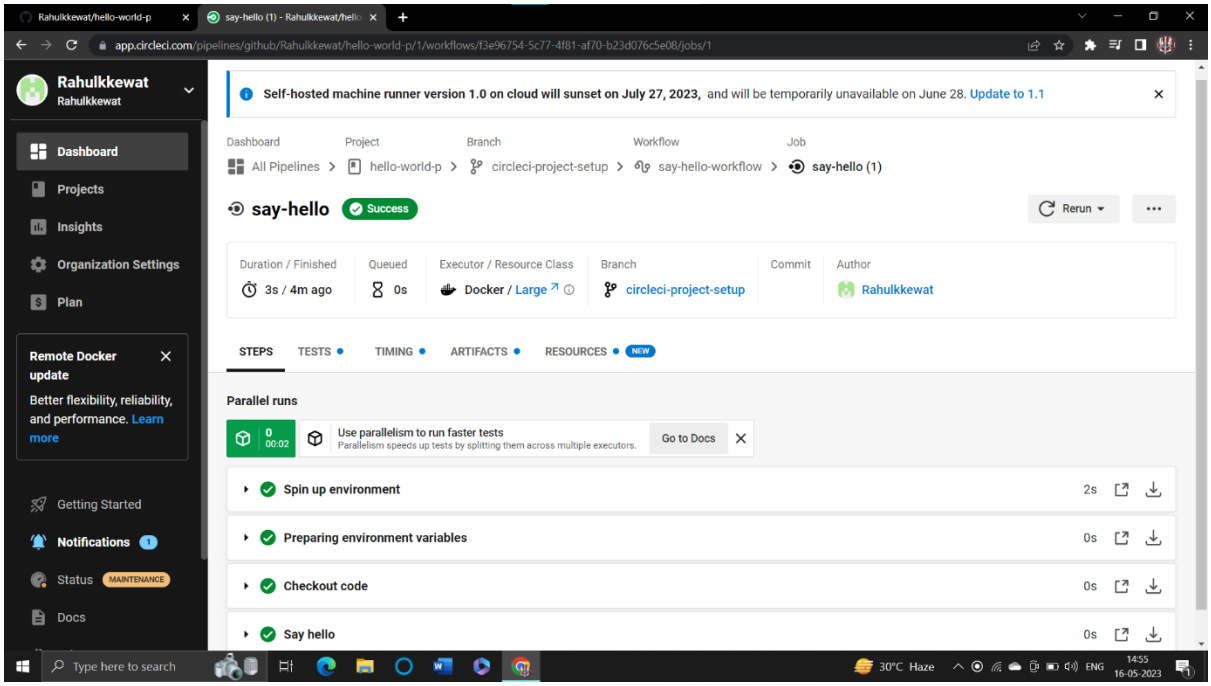
d. Say hello

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

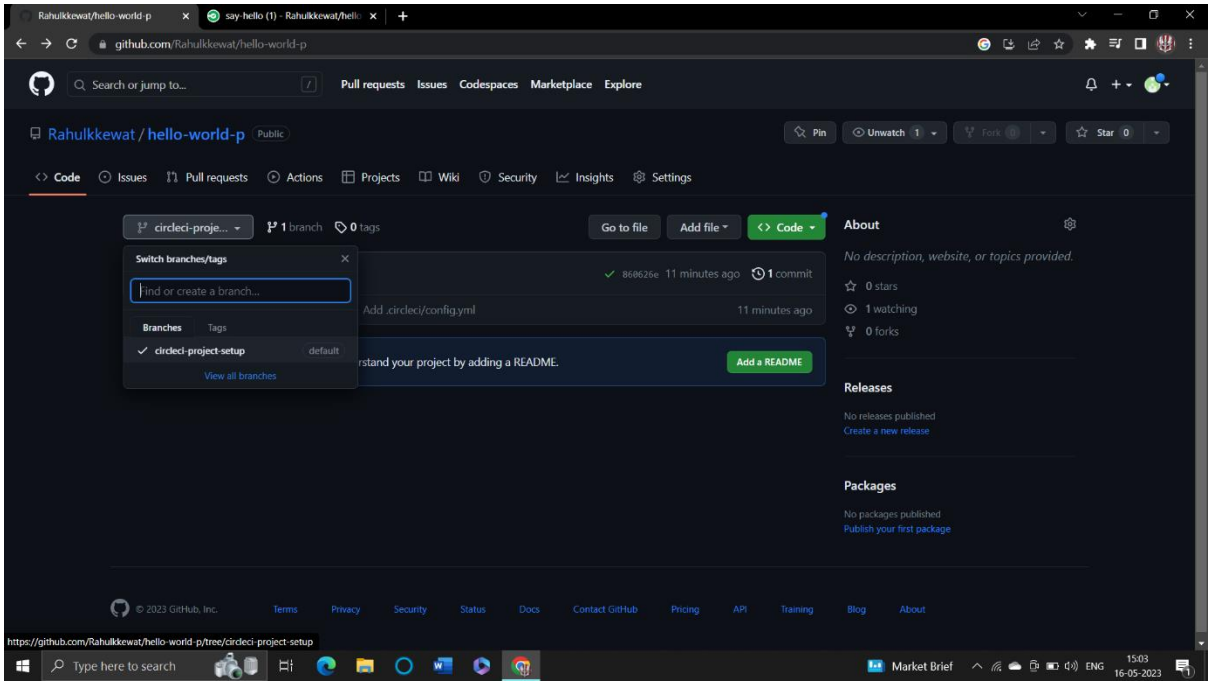


Select “say-hello” Job with a green tick





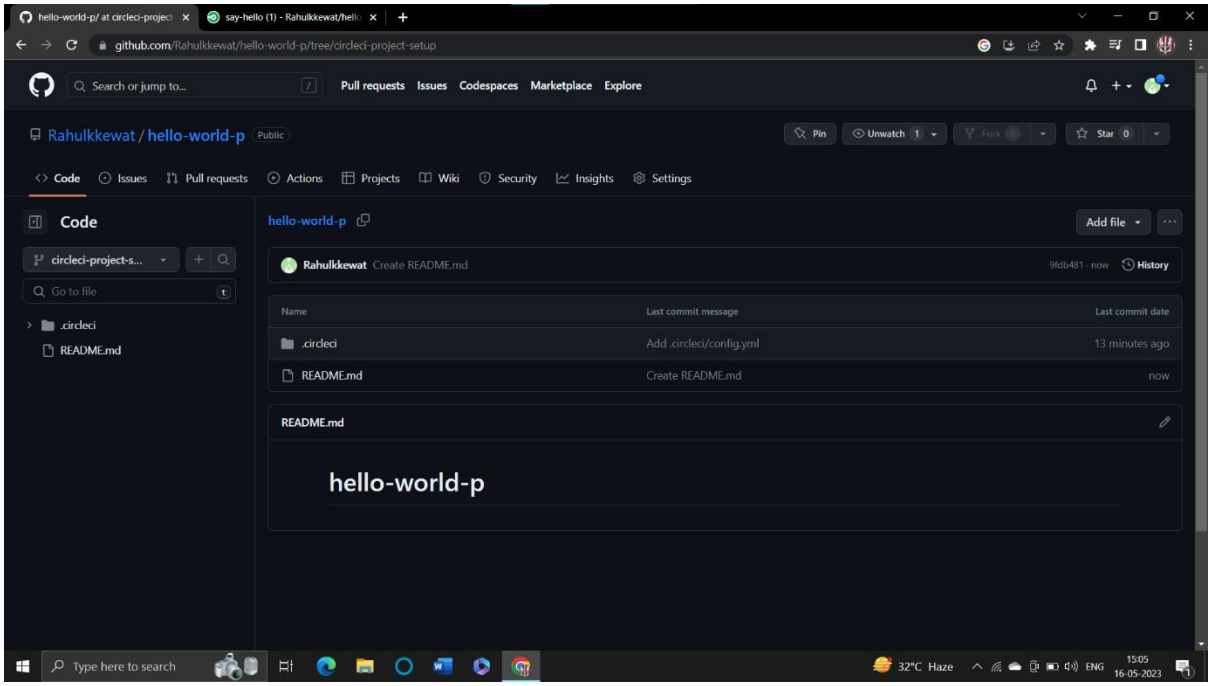
Select Branch and option circleci-project-setup

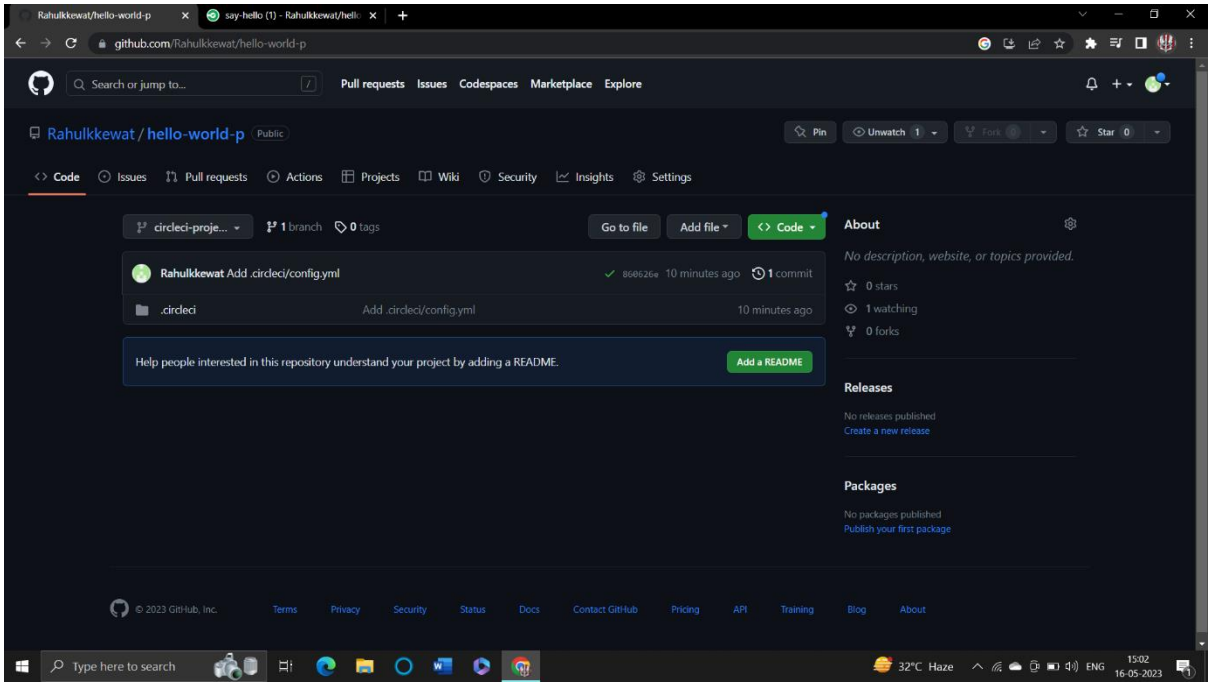


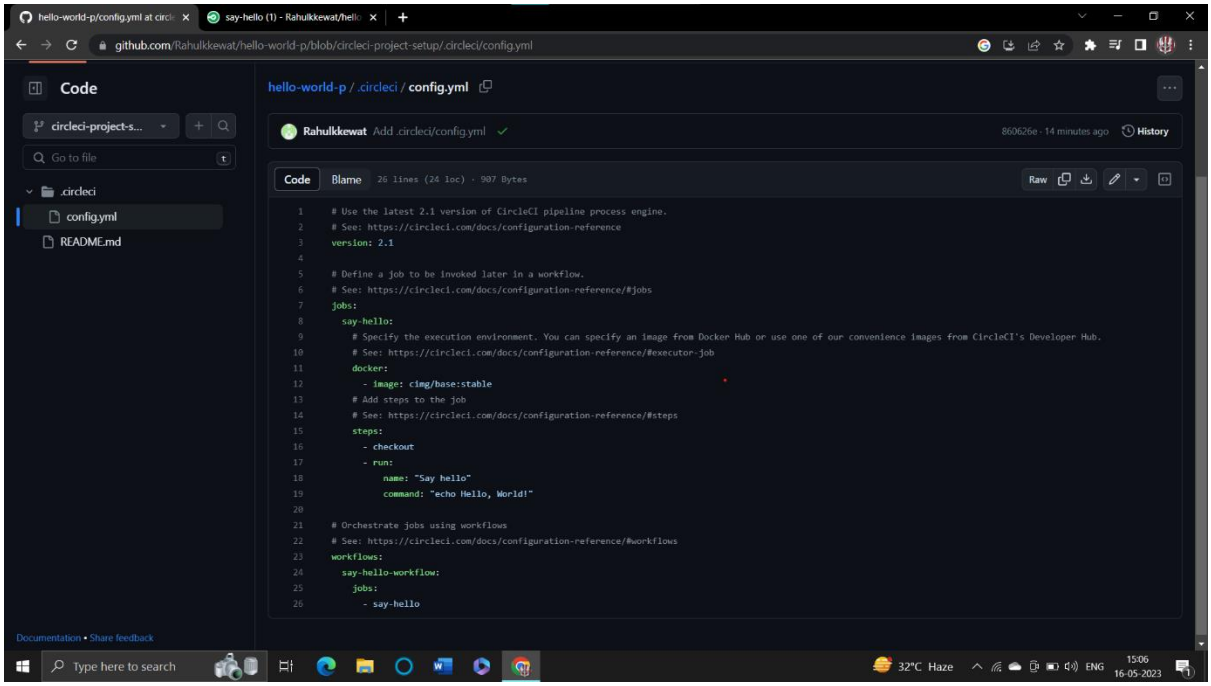
Step 4: Break your build

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

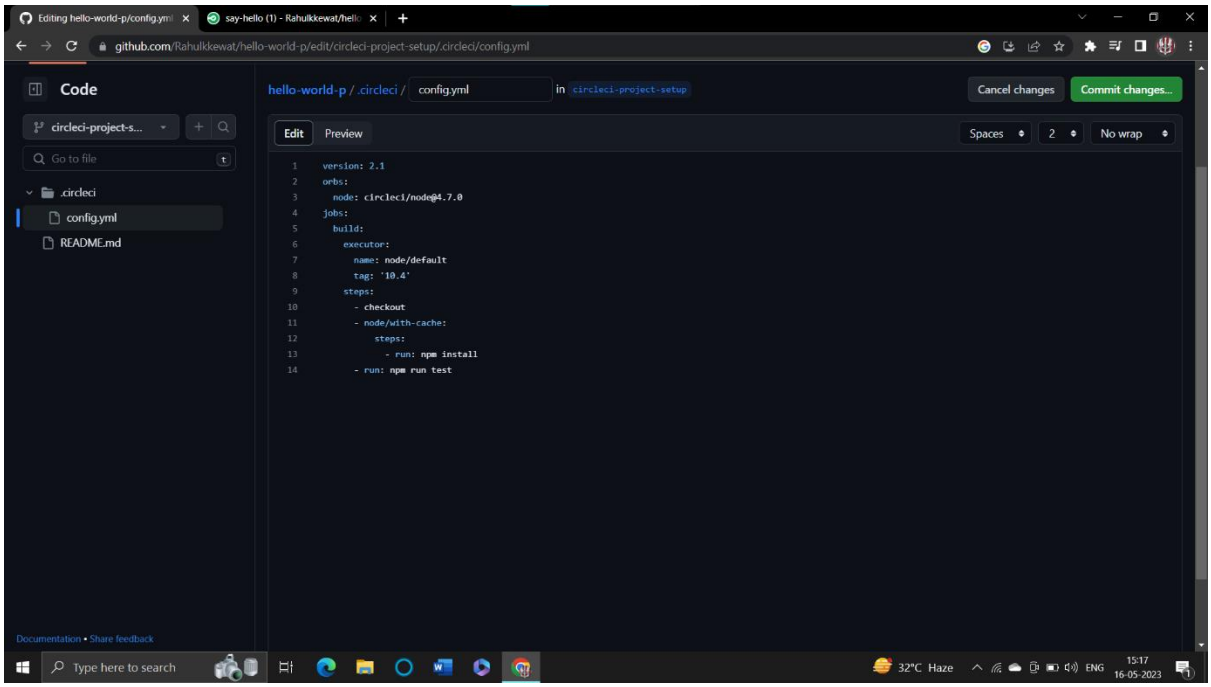
It is possible to edit files directly on GitHub.


github.com/Rahulkkewat/hello-world-p/tree/circleci-project-setup
Rahulkkewat / hello-world-p
Public
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
hello-world-p
Add file
Rahulkkewat Create README.md
.circleci
Add .circleci/config.yml
13 minutes ago
README.md
Create README.md
now
README.md
hello-world-p
Type here to search
32°C Haze
ENG
15:05
16-05-2023

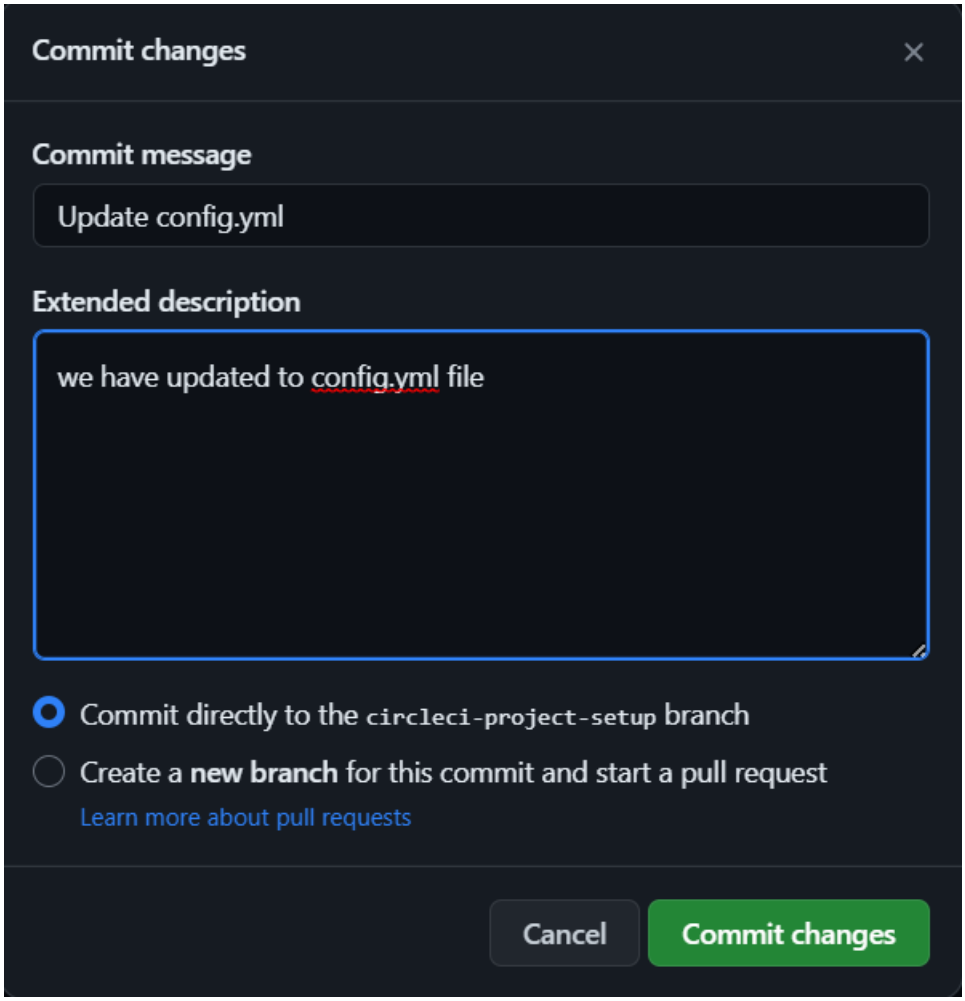

github.com/Rahulkkewat/hello-world-p
Rahulkkewat / hello-world-p
Public
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
1 branch
0 tags
Go to file
Add file
Code
Rahulkkewat Add .circleci/config.yml
1 commit
10 minutes ago
.circleci
Add .circleci/config.yml
Help people interested in this repository understand your project by adding a README.
Add a README
About
No description, website, or topics provided.
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
© 2023 GitHub, Inc.
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About
Type here to search
32°C Haze
ENG
15:02
16-05-2023



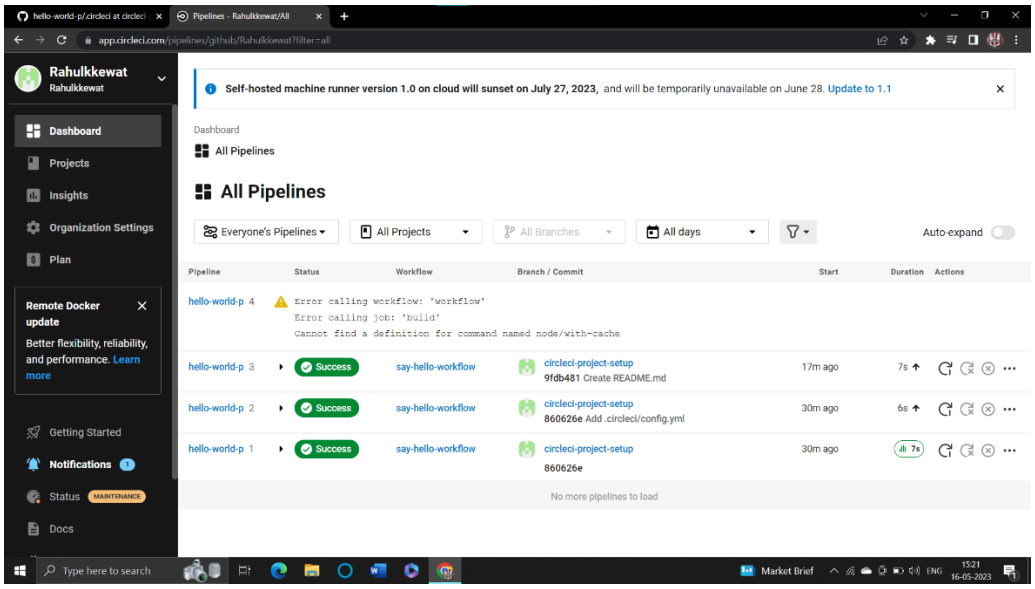
The GitHub file editor should look like this



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 Nodeproject in your repository.



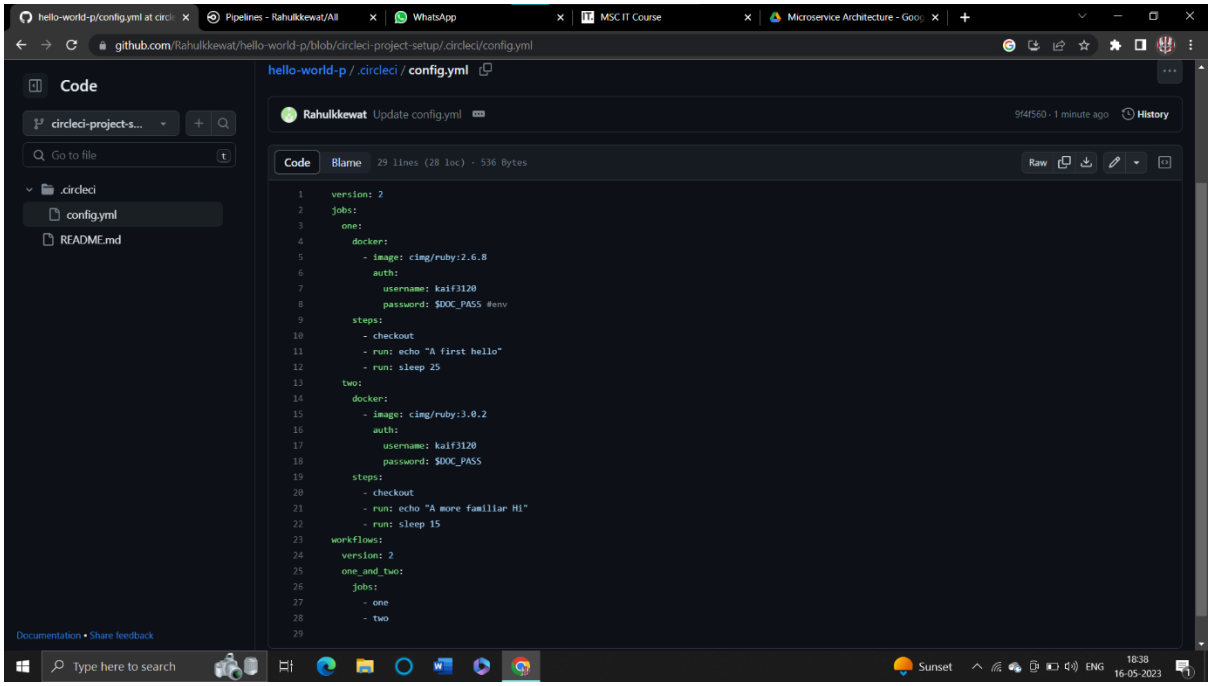
Step 5: Use Workflows

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

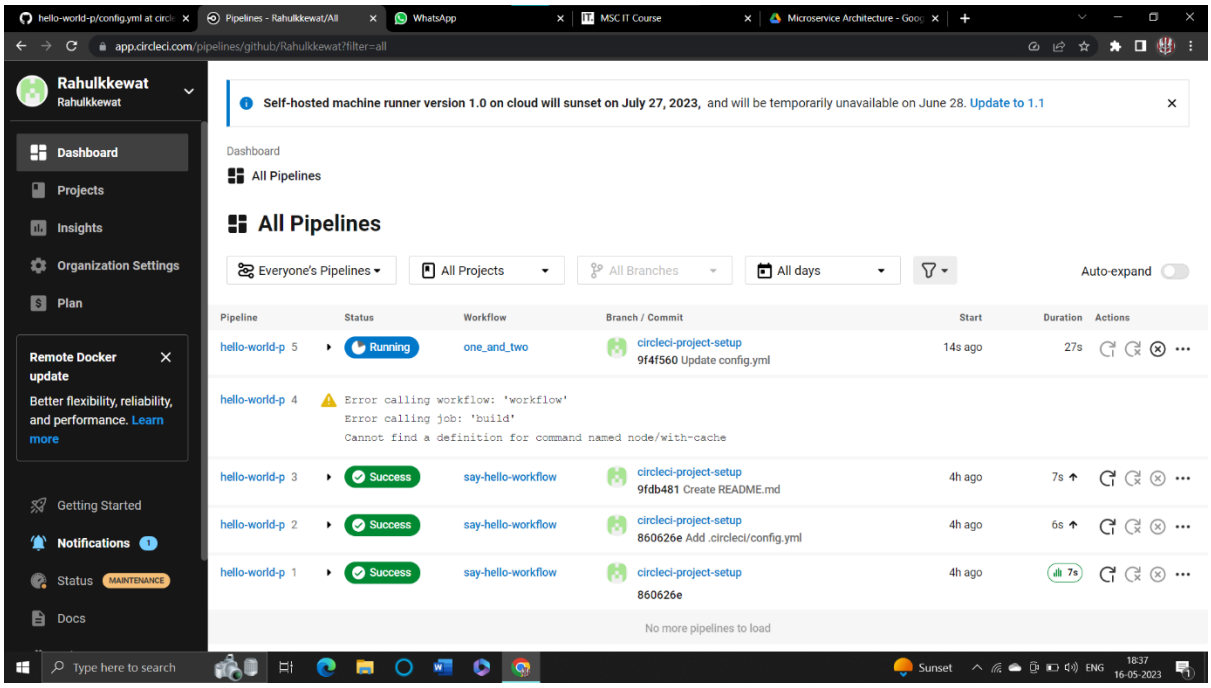
1) **Take a moment and read the comments in the code block below. Then, to see workflows in action, edit your .circleci/config.yml file and copy and paste the following text into it.**

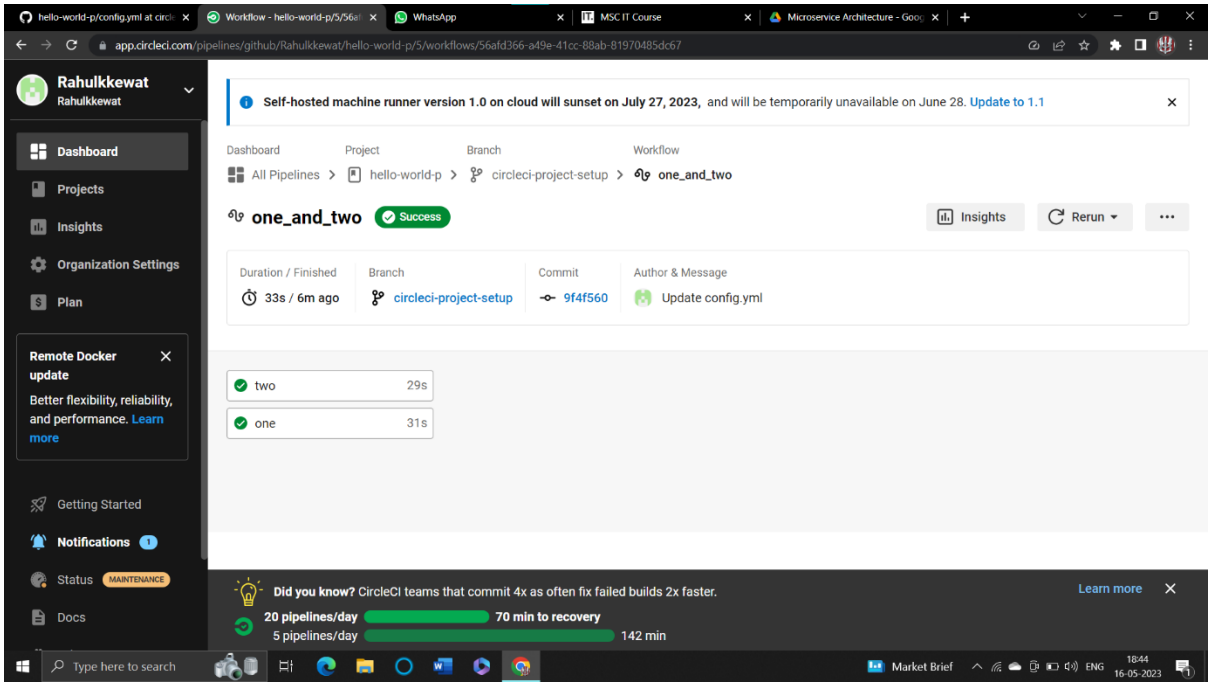
```yaml
version: 2
jobs: # we now have TWO jobs, so that a workflow can coordinate them!
  one: # This is our first job.
    docker: # it uses the docker executor
      - image: cimg/ruby:2.6.8 # specifically, a docker image with ruby 2.6.8
        auth:
          username: mydockerhub-user
          password: $DOCKERHUB_PASSWORD  # context / project UI env-var reference
    # Steps are a list of commands to run inside the docker container above.
    steps:
      - checkout # this pulls code down from GitHub
      - run: echo "A first hello" # This prints "A first hello" to stdout.
      - run: sleep 25 # a command telling the job to "sleep" for 25 seconds.
  two: # This is our second job.
    docker: # it runs inside a docker image, the same as above.
      - image: cimg/ruby:3.0.2
        auth:
          username: mydockerhub-user
          password: $DOCKERHUB_PASSWORD  # context / project UI env-var reference
    steps:
      - checkout
      - run: echo "A more familiar hi" # We run a similar echo command to above.
      - run: sleep 15 # and then sleep for 15 seconds.
# Under the workflows: map, we can coordinate our two jobs, defined above.
workflows:
  version: 2
  one_and_two: # this is the name of our workflow
    jobs: # and here we list the jobs we are going to run.
      - one
      - two
```

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



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



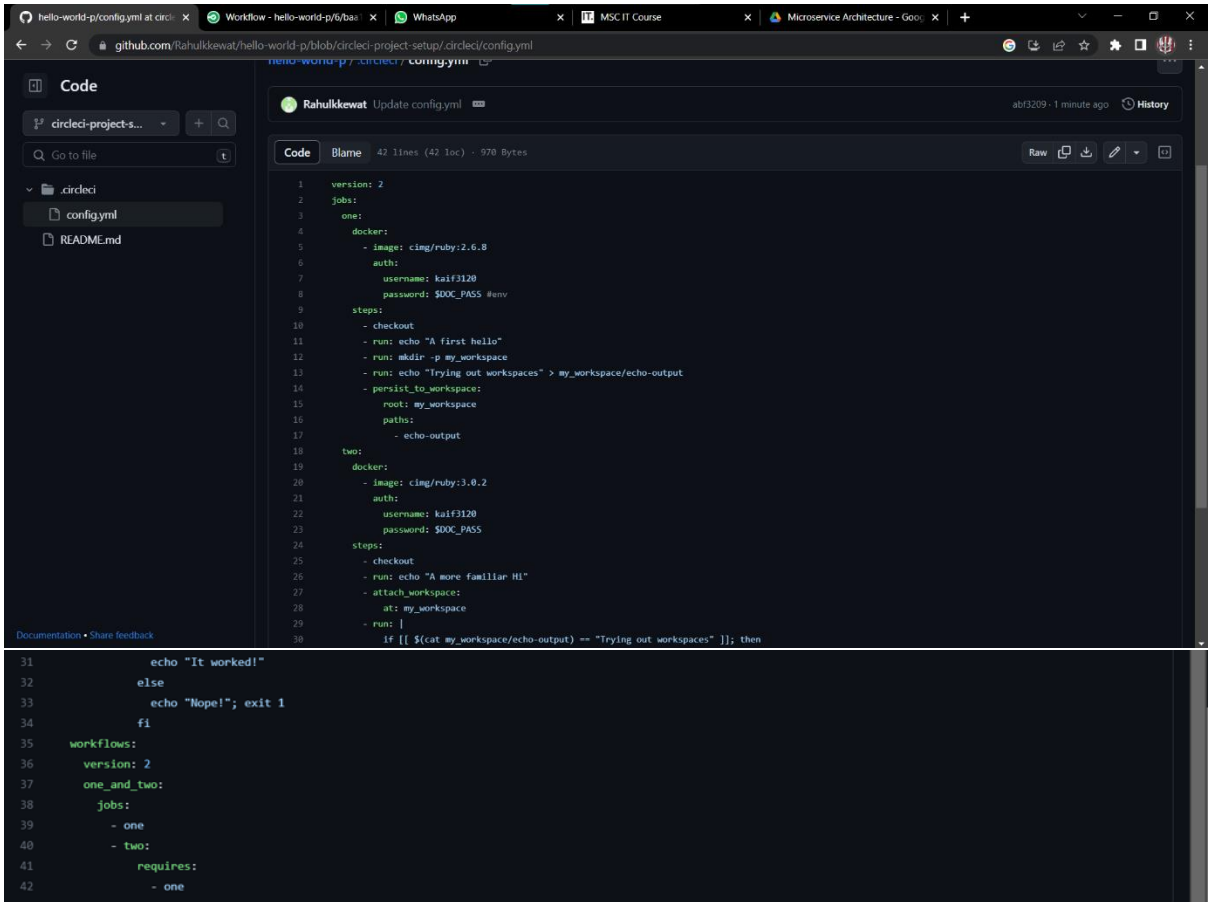


Step 6: Add some changes to use workspaces

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

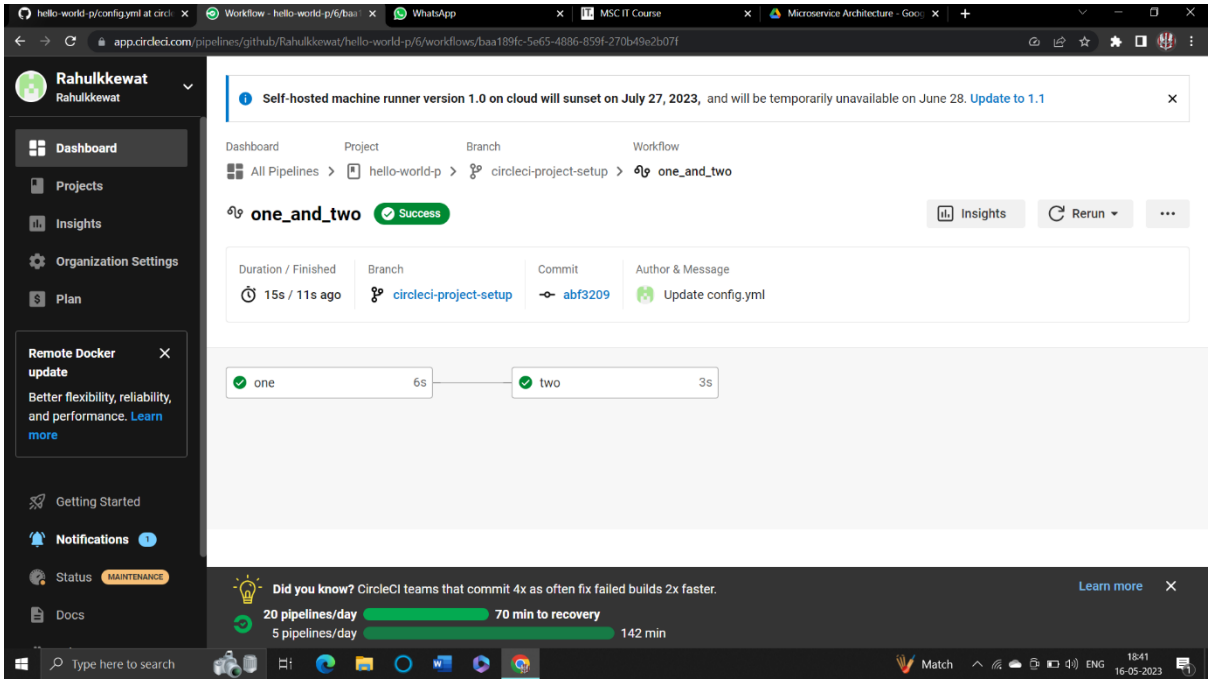
```yaml
version: 2
jobs:
  one:
    docker:
      - image: cimg/ruby:3.0.2
        auth:
          username: mydockerhub-user
          password: $DOCKERHUB_PASSWORD  # context / project UI env-var reference
    steps:
      - checkout
      - run: echo "A first hello"
      - run: mkdir -p my_workspace
      - run: echo "Trying out workspaces" > my_workspace/echo-output
      - persist_to_workspace:
          # Must be an absolute path, or relative path from working_directory
          root: my_workspace
          # Must be relative path from root
          paths:
            - echo-output
  two:
    docker:
      - image: cimg/ruby:3.0.2
        auth:
          username: mydockerhub-user
          password: $DOCKERHUB_PASSWORD  # context / project UI env-var reference
    steps:
      - checkout
      - run: echo "A more familiar hi"
      - attach_workspace:
          # Must be absolute path or relative path from working_directory
          at: my_workspace

      - run: |
          if [[ $(cat my_workspace/echo-output) == "Trying out workspaces" ]]; then
            echo "It worked!";
          else
            echo "Nope!"; exit 1
          fi
workflows:
  version: 2
  one_and_two:
    jobs:
      - one
      - two:
          requires:
            - one
```

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



```
version: 2
jobs:
  one:
    docker:
      - image: cimg/ruby:2.6.8
        auth:
          username: kaif3120
          password: $DOC_PASS #env
    steps:
      - checkout
      - run: echo "A first hello"
      - run: mkdir -p my_workspace
      - run: echo "Trying out workspaces" > my_workspace/echo-output
      - persist_to_workspace:
          root: my_workspace
          paths:
            - echo-output
  two:
    docker:
      - image: cimg/ruby:3.0.2
        auth:
          username: kaif3120
          password: $DOC_PASS
    steps:
      - checkout
      - run: echo "A more familiar Hi"
      - attach_workspace:
          at: my_workspace
      - run: |
          if [[ $(cat my_workspace/echo-output) == "Trying out workspaces" ]]; then
            echo "It worked!"
          else
            echo "Nope!"; exit 1
          fi
workflows:
  version: 2
  one_and_two:
    jobs:
      - one
      - two:
          requires:
            - one
```

Finally your workflow with the jobs running should look like this



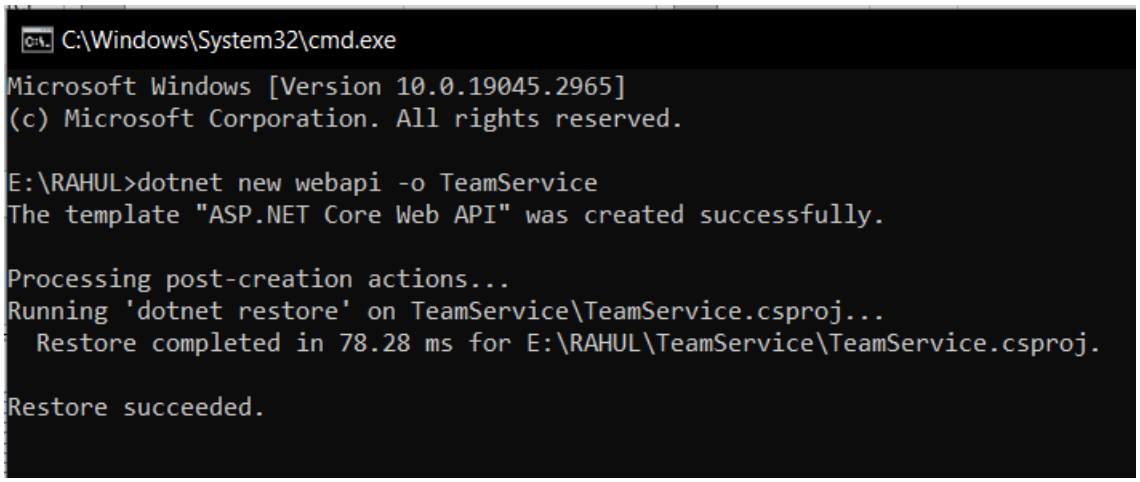
Practical 6

Aim: Creating Microservice with ASP.NET Core.

Writeup:

Step 1: Create new project.

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



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

Step 3: Add new files as follows:

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

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

        public string FirstName
        {
            get;
            set;
        }

        public string LastName
        {
            get;
            set;
        }

        public Member()
        {
```

```
        }

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

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

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

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

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

namespace TeamService.Models

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

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

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

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

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

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

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

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

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

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

```
        {
            return this.Ok(repository.List());
        }

        [HttpGet("{id}")]
        public IActionResult GetTeam(Guid id)
        {
            Team team = repository.Get(id);
            if (team != null)
            {
                return this.Ok(team);
            }

            else
            {
                return this.NotFound();
            }
        }

        [HttpPost]
        public virtual IActionResult CreateTeam ([FromBody]Team newTeam)
        {
             repository.Add(newTeam);
             return this.Created($"/teams/{newTeam.ID}", newTeam);
        }

        [HttpPut("{id}")]
        public virtual IActionResult UpdateTeam([FromBody]Team team, Guid id)
        {
            team.ID = id;
            if(repository.Update(team) == null)
            {
                return this.NotFound();
            }

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

        [HttpDelete("{id}")]
        public virtual IActionResult DeleteTeam(Guid id)
        {
                        Team team = repository.Delete(id);
            if(team == null)
            {
                return this.NotFound();
```

```
            }

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

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

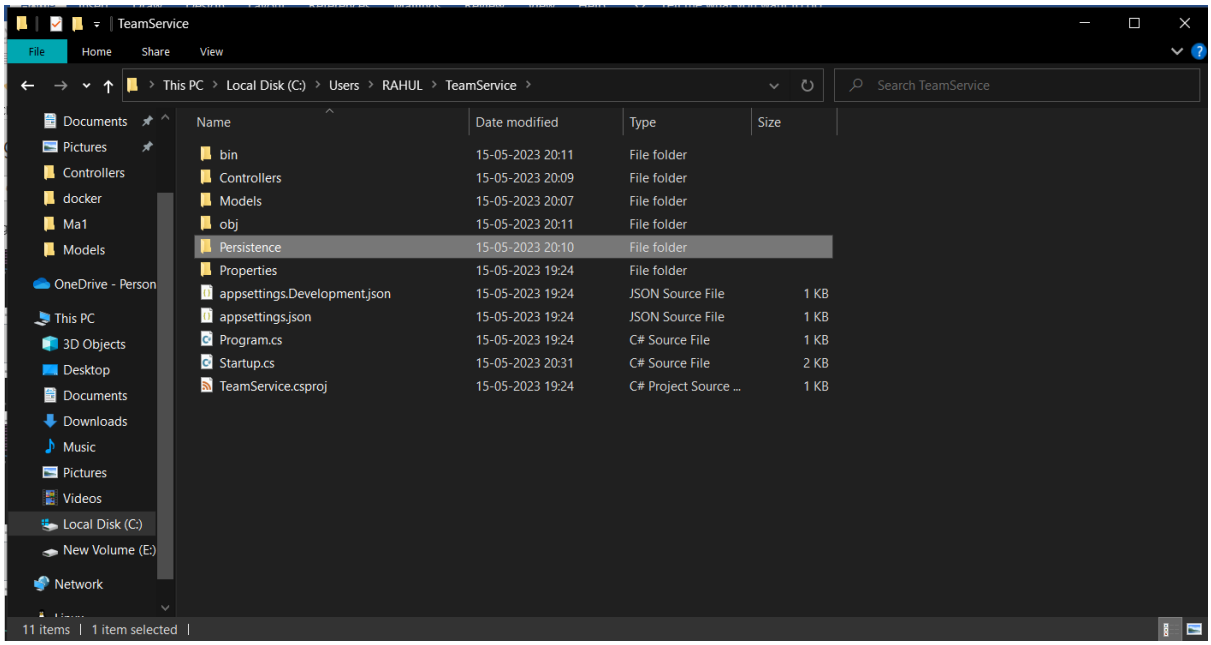
```
using System;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Linq;
using TeamService.Models;
using System.Threading.Tasks;
using TeamService.Persistence;

namespace TeamService

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

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

            else
            {
```

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

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

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

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

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

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

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

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

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

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

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

            else
            {
                return this.NotFound();
            }
        }

        private Guid GetTeamIdForMember(Guid memberId)
        {
```

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

            return Guid.Empty;
        }
    }
}
```

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



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

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

namespace TeamService.Persistence
{
    public interface ITeamRepository
```

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

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

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

namespace TeamService.Persistence
{
    public class MemoryTeamRepository : ITeamRepository
    {
        protected static ICollection<Team> teams;
        public MemoryTeamRepository()
        {
            if(teams == null)
            {
                teams = new List<Team>();
            }
        }


        public MemoryTeamRepository(ICollection<Team> teams)
        {
            MemoryTeamRepository.teams = teams;
        }

        public IEnumerable<Team> List()
        {
            return teams;
        }
        public Team Get(Guid id)
        {
            return teams.FirstOrDefault(t => t.ID == id);
        }

        public Team Update(Team t)
```

```
        {
            Team team = this.Delete(t.ID);
            if(team != null)
            {
                team = this.Add(t);
            }

            return team;
        }

        public Team Add(Team team)
        {
            teams.Add(team);
            return team;
        }

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

            return team;
        }
    }
}
```

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

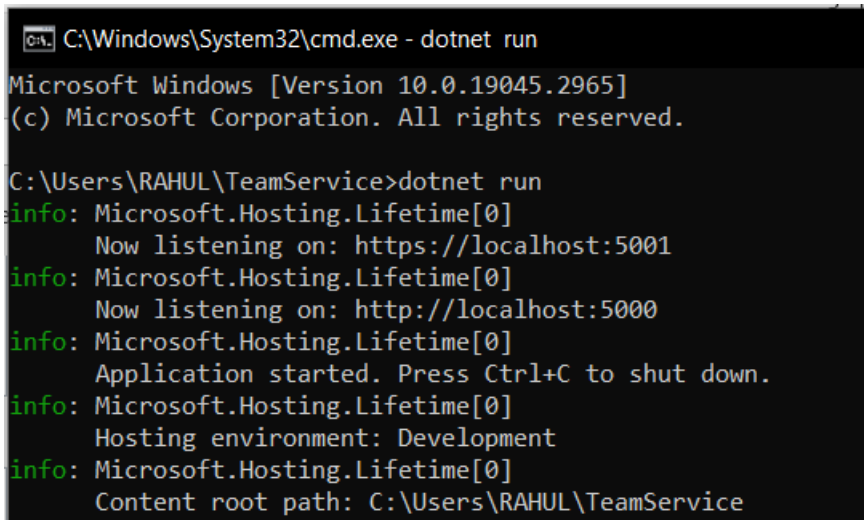
```
services.AddScoped<ITeamRepository, MemoryTeamRepository>();
```

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

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

Commands:

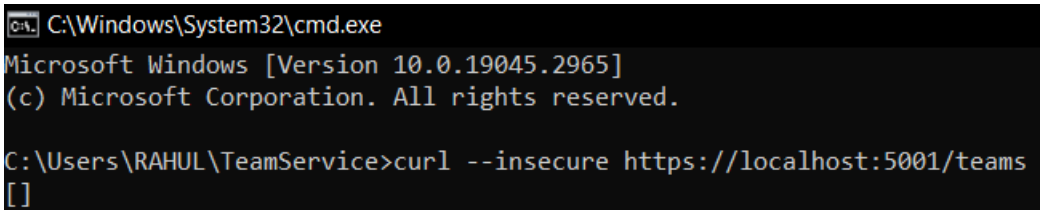
Dotnet run:



Step 10: On Command prompt 2

Command: To get all teams

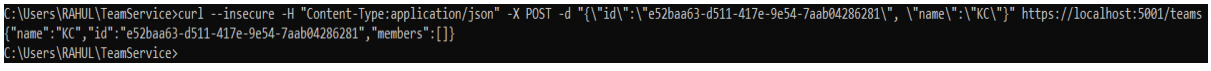
curl --insecure https://localhost:5001/teams



Step 11: On Command prompt 2

Command: To create new team

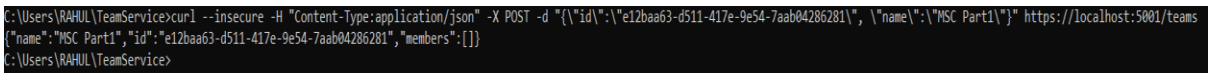
curl --insecure -H "Content-Type:application/json" –X POST –d "{\"id\":\"e52baa63-d511-417e-9e54-7aab04286281\", \"name\":\"KC\"}" https://localhost:5001/teams



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



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 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-d511-417
e-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
"e52baa63-d511-417e-9e54-7aab04286281"
C:\Users\RAHUL\TeamService>
```

Step 17: On Command prompt 2

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

curl –insecure https://localhost:5001//teams

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

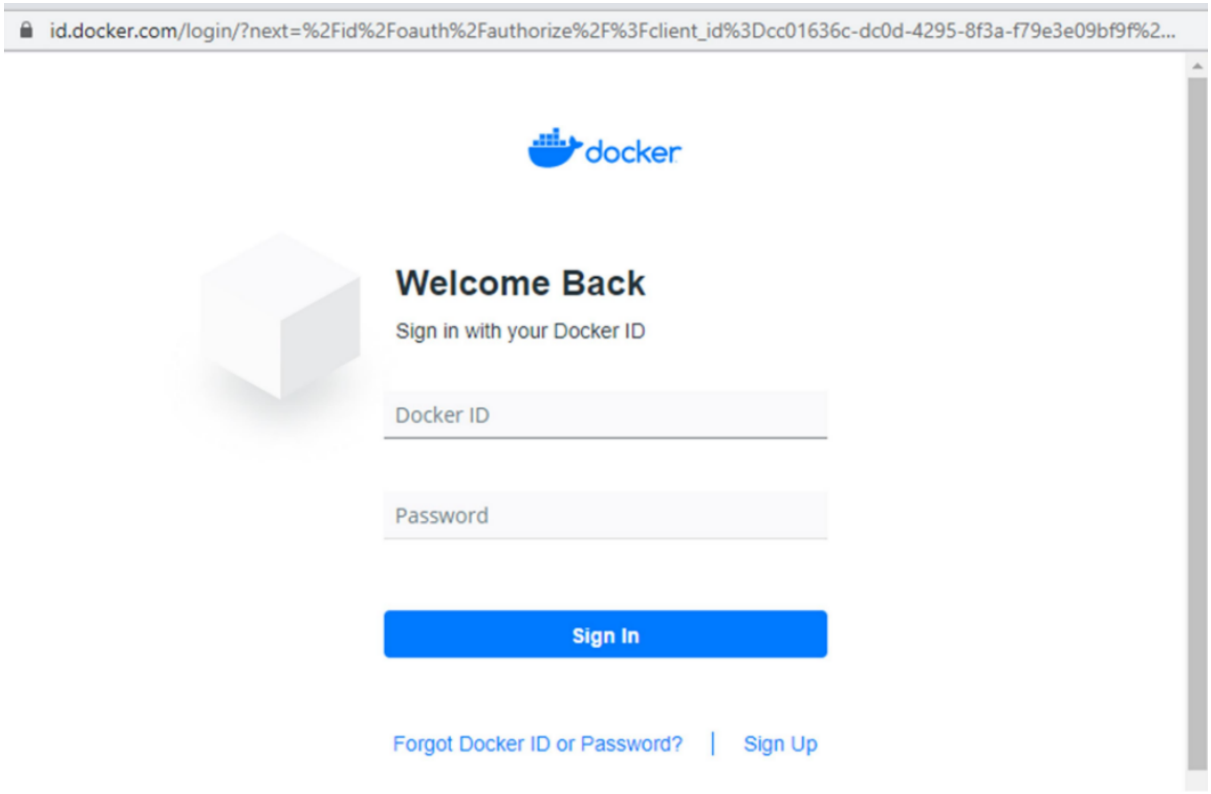
Practical 7

Aim: Running Location Service in Docker

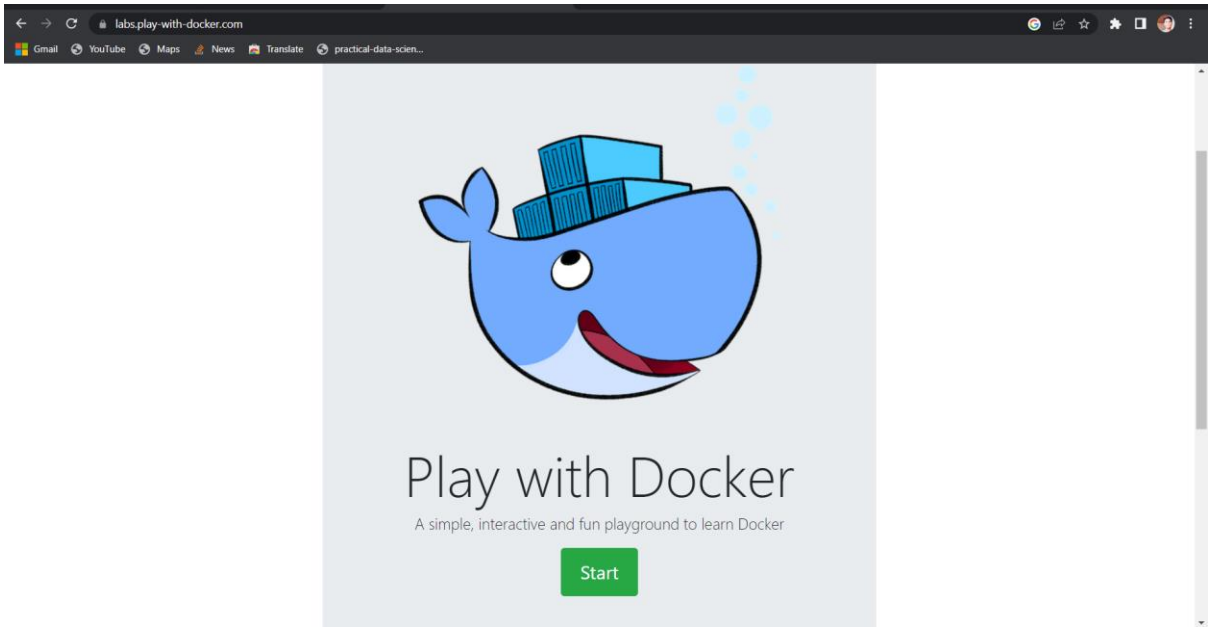
Writeup:

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

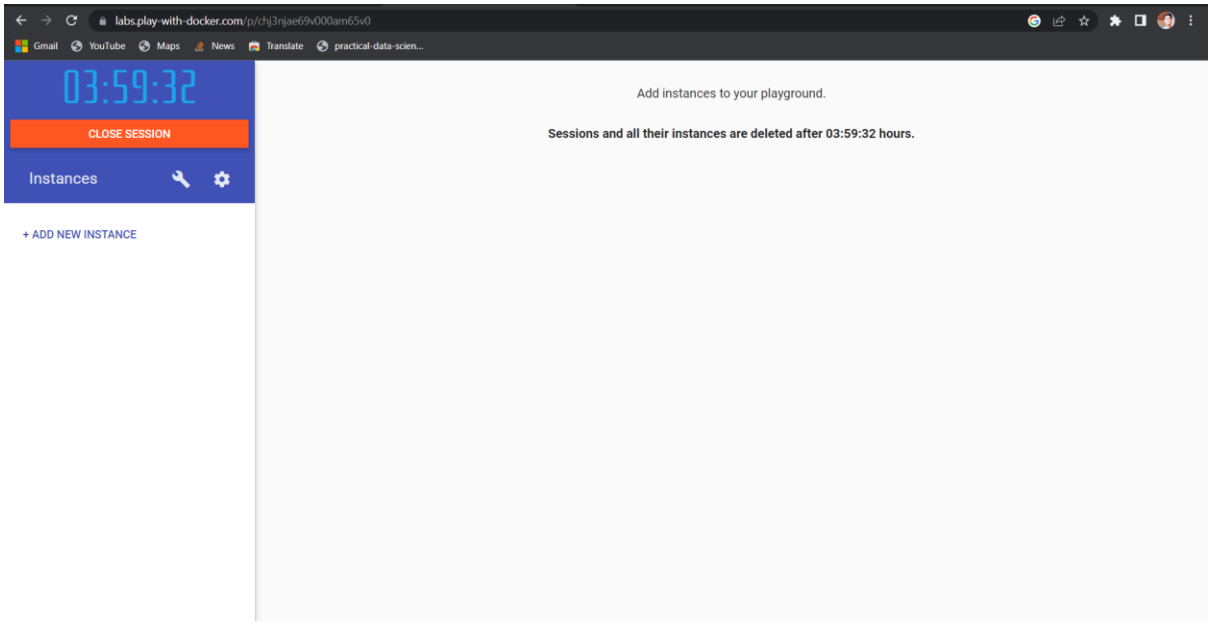
Now login in to Play-With-Docker

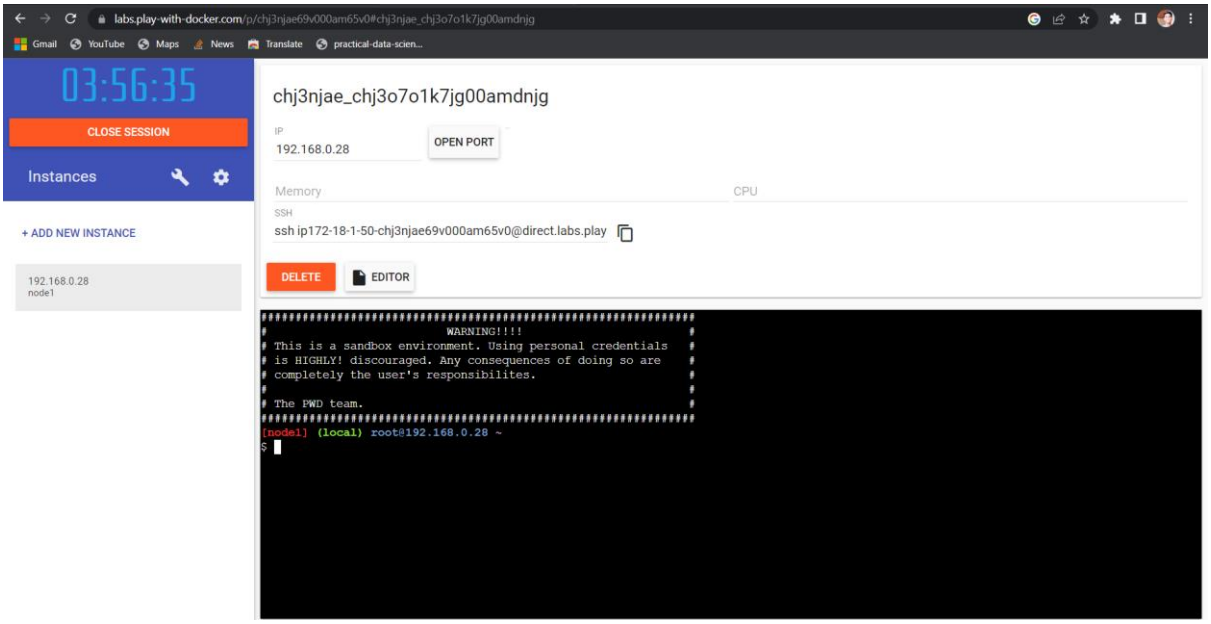


Step 2: Click on Start



Step 3: Click on Add New Instance





Step 3: Start typing following commands

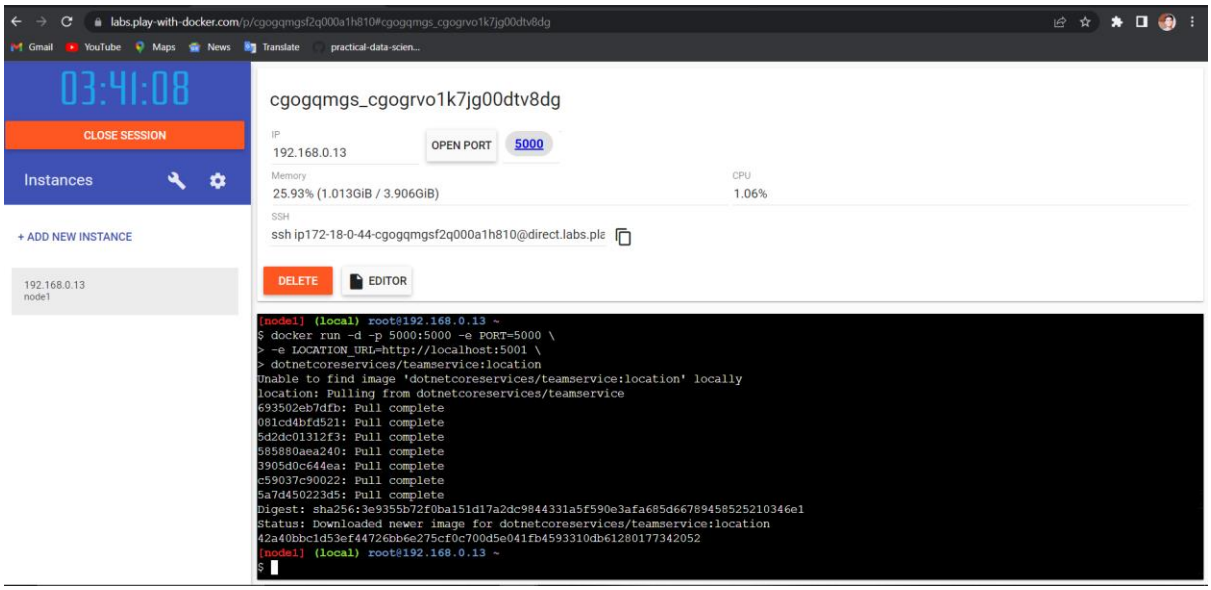
Command : To run teamservice

**docker run -d -p 5000:5000 -e PORT=5000 **

**-e LOCATION__URL=http://localhost:5001 **

dotnetcoreservices/teamservice:location

Output:



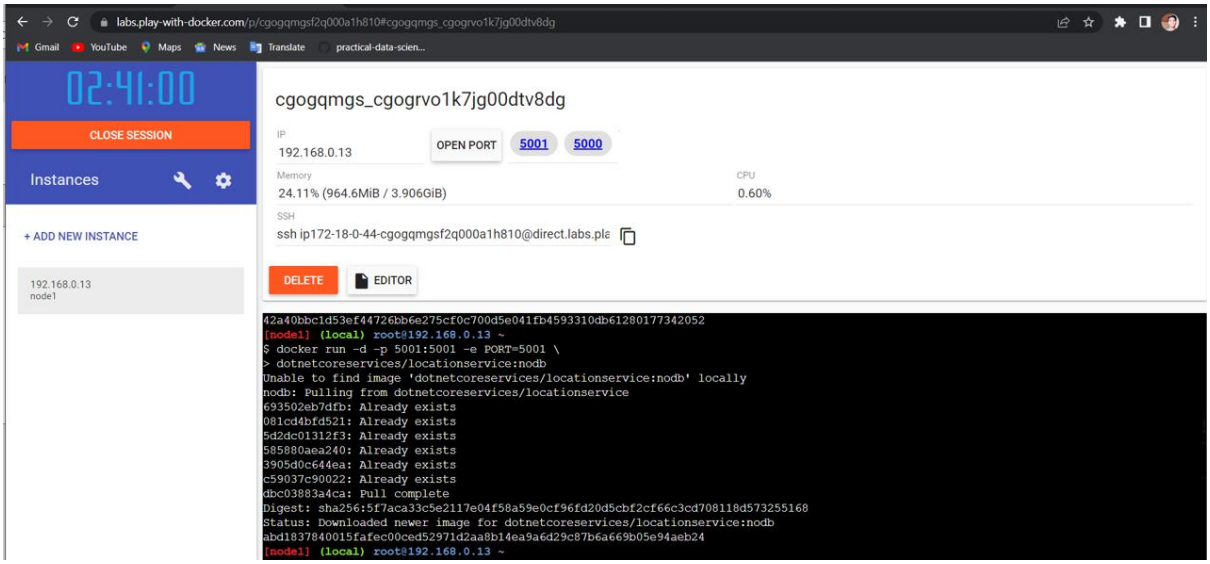
Step 4:

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)

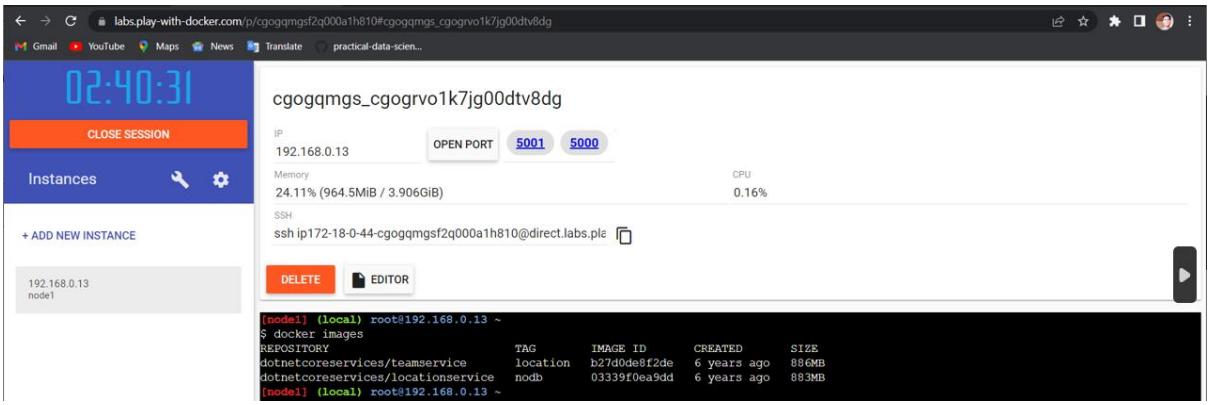


Step 4:

Command : to check running images in docker

Docker images

Output:



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-d511-4
17e-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

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","l
astName":"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:

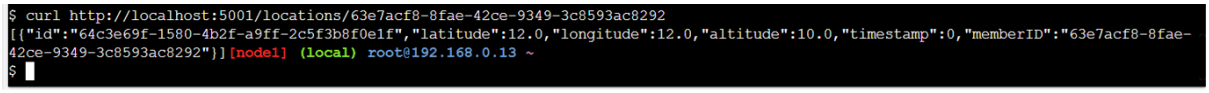
```
$ curl -H "Content-Type:application/json" -X POST -d \
> '{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f", "latitude":12.0,"longitude":12.0,"altitude":10.0,
"timestamp":0,"memberId":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}' http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292
{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f","latitude":12.0,"longitude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

Step 10:

Command : To confirm location is added in member

curl http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292

output:

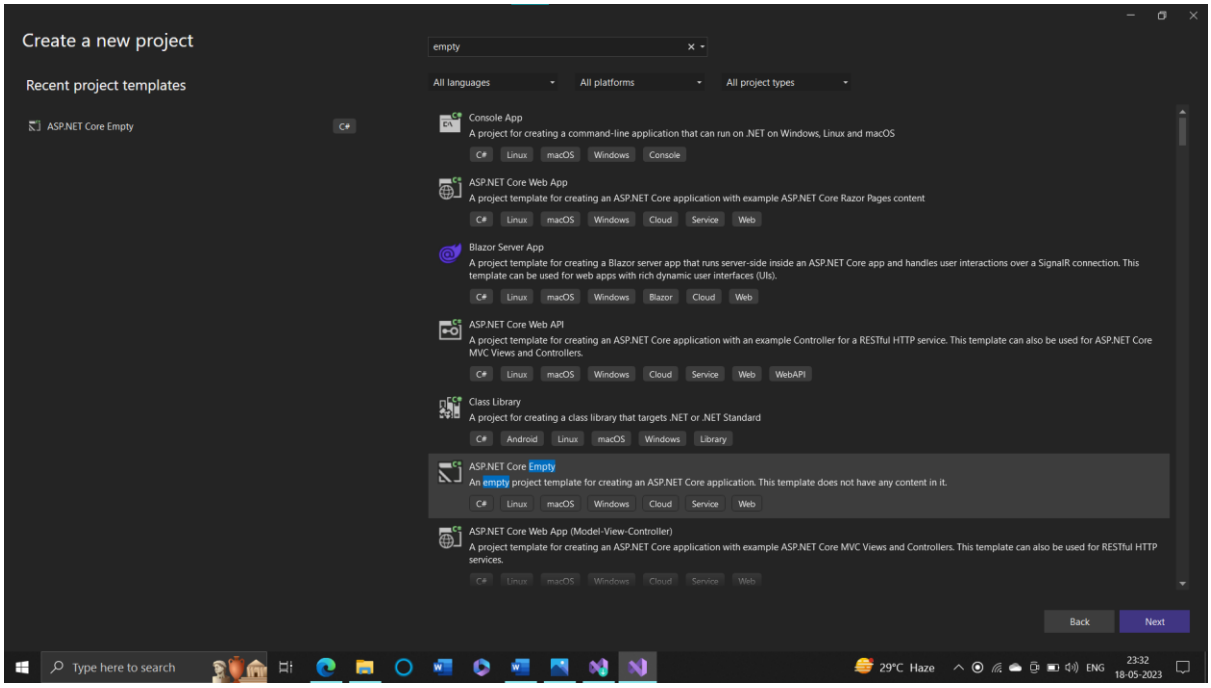


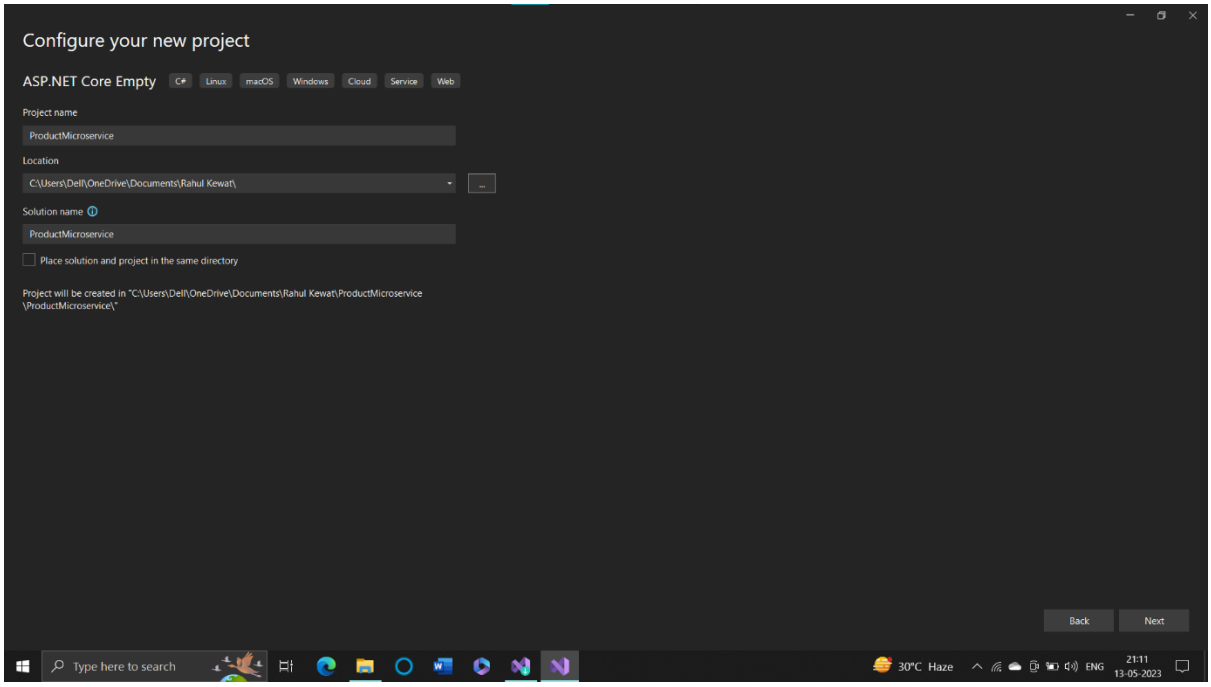
Practical 8

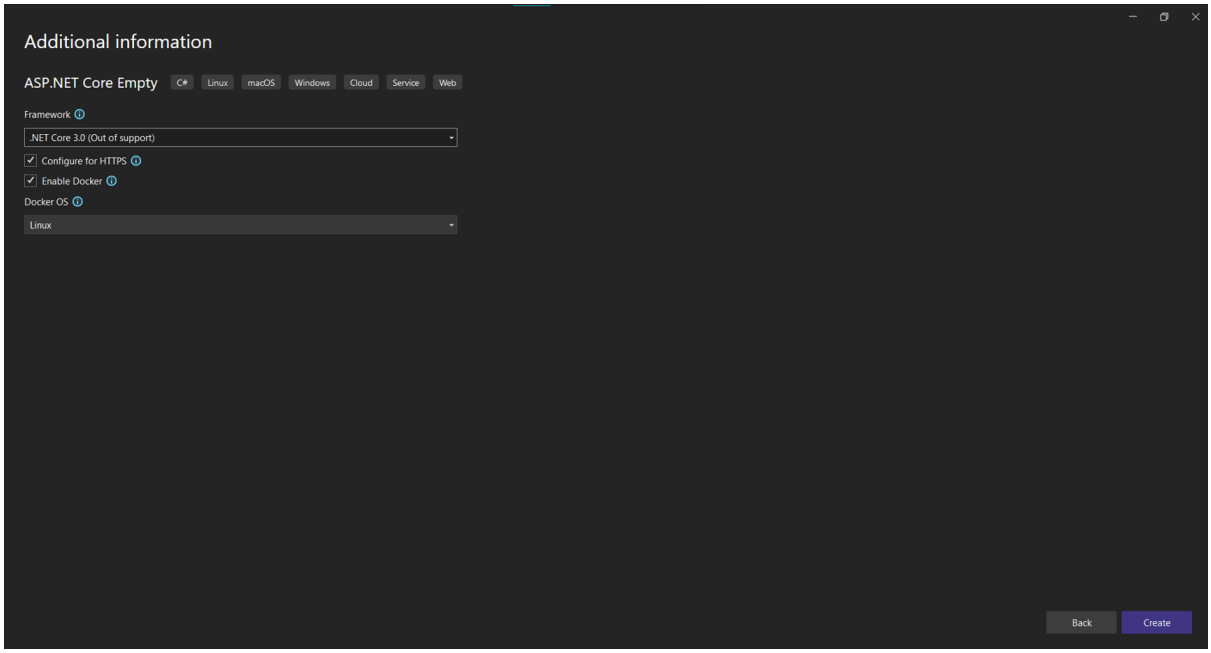
Aim: Building real-time Microservice with ASP.NET Core.

Writeup:

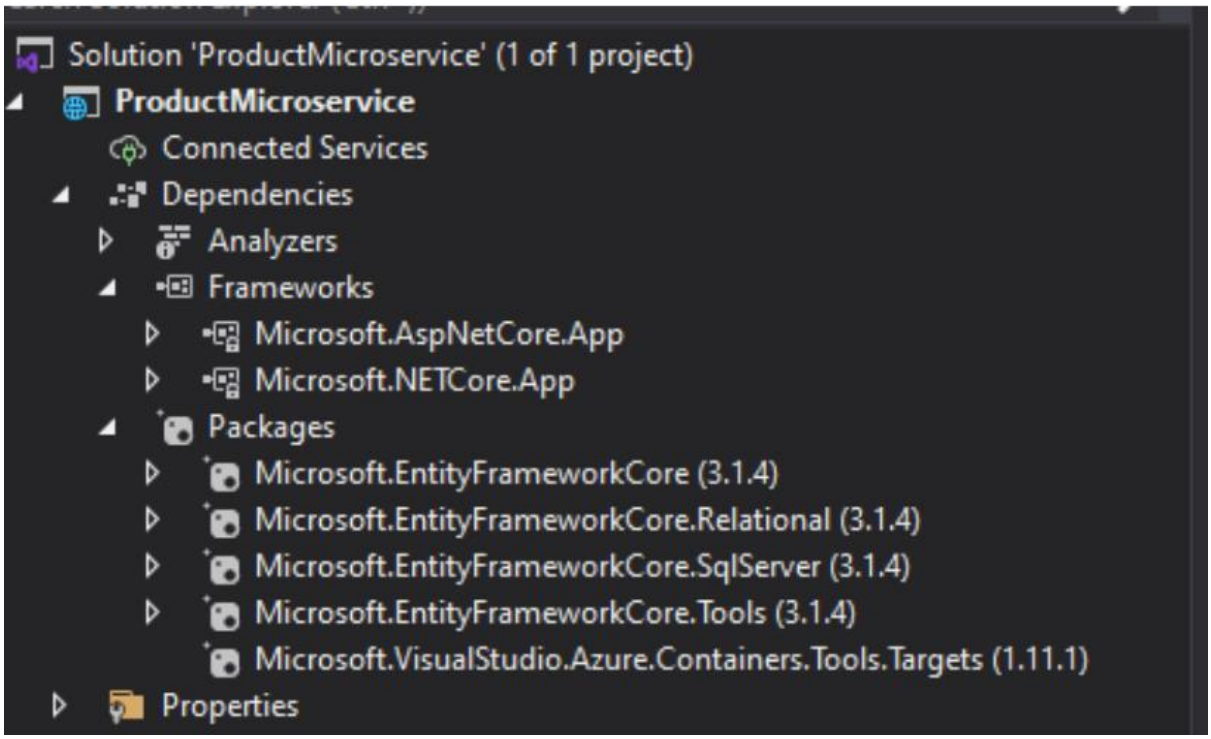
Step 1: Create and configure a new empty project



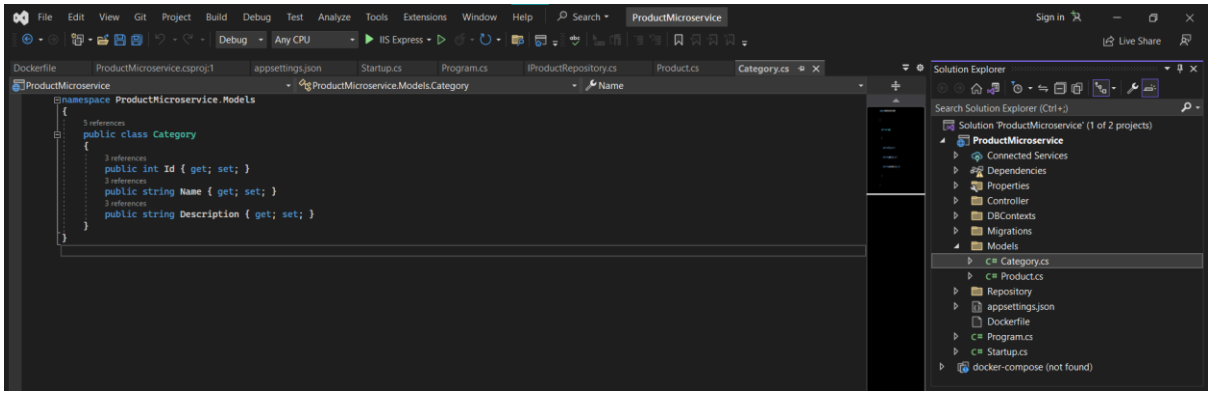




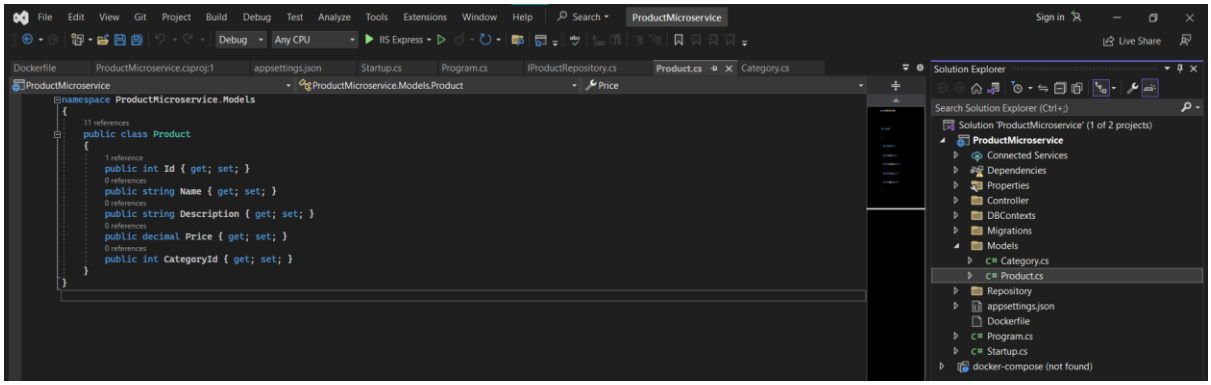
Step 2: Make sure to add all these packages via nugget package manager



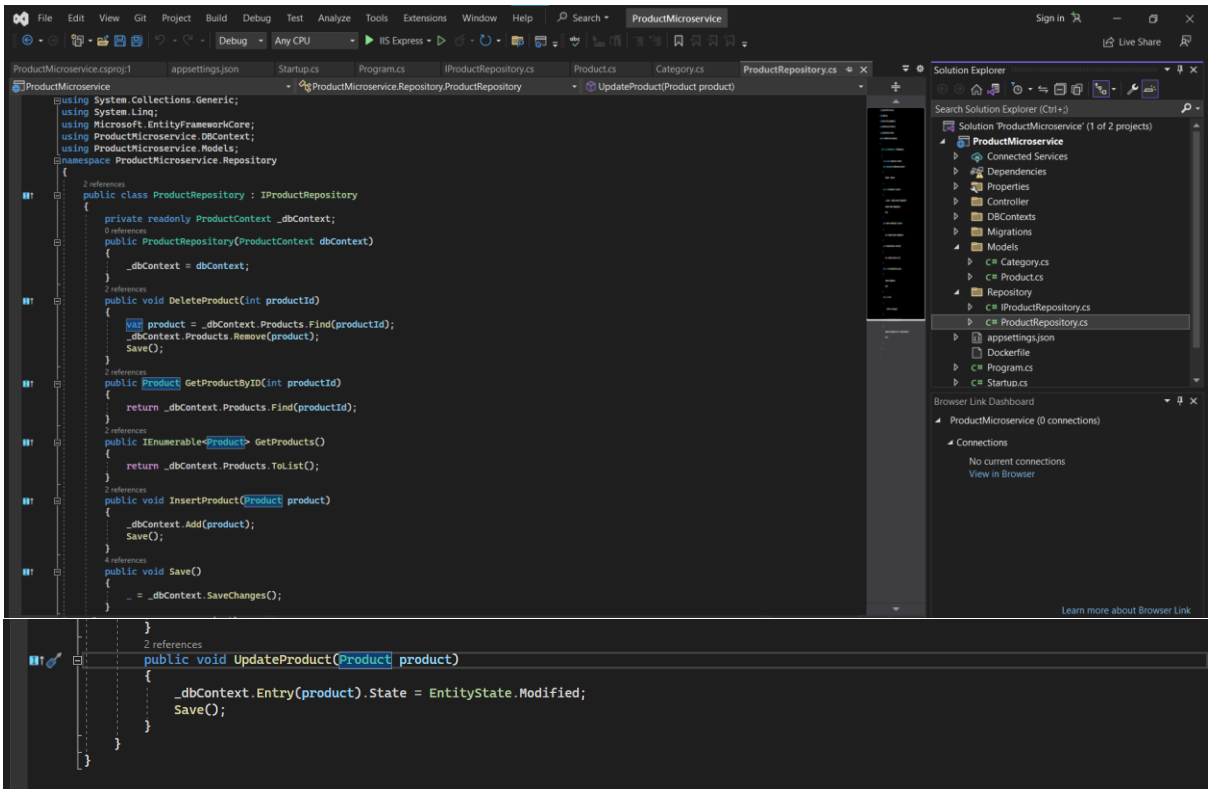
Step 3: Create the Category Model, which will eventually be Category Table



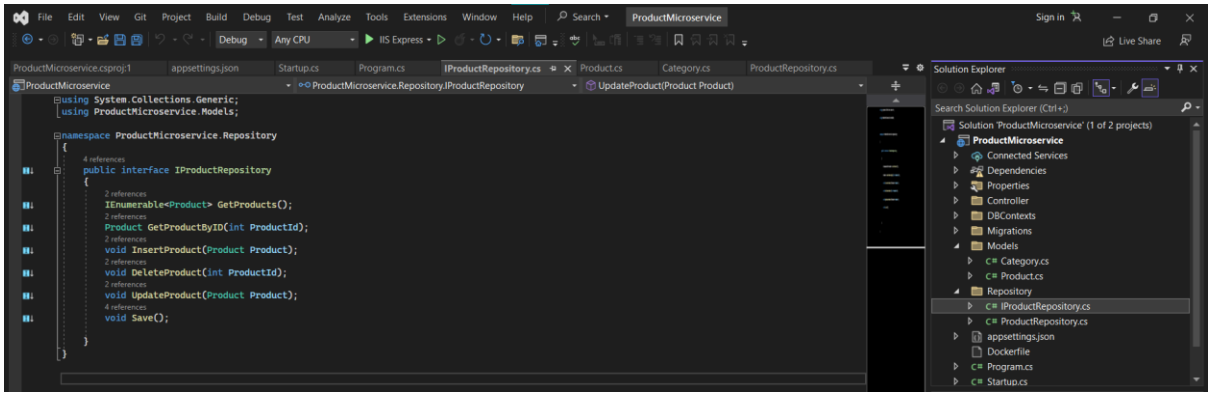
Step 4: Create the Product Model which will eventually be the Product Table



Step 5: Create a ProductRepository for all our operations

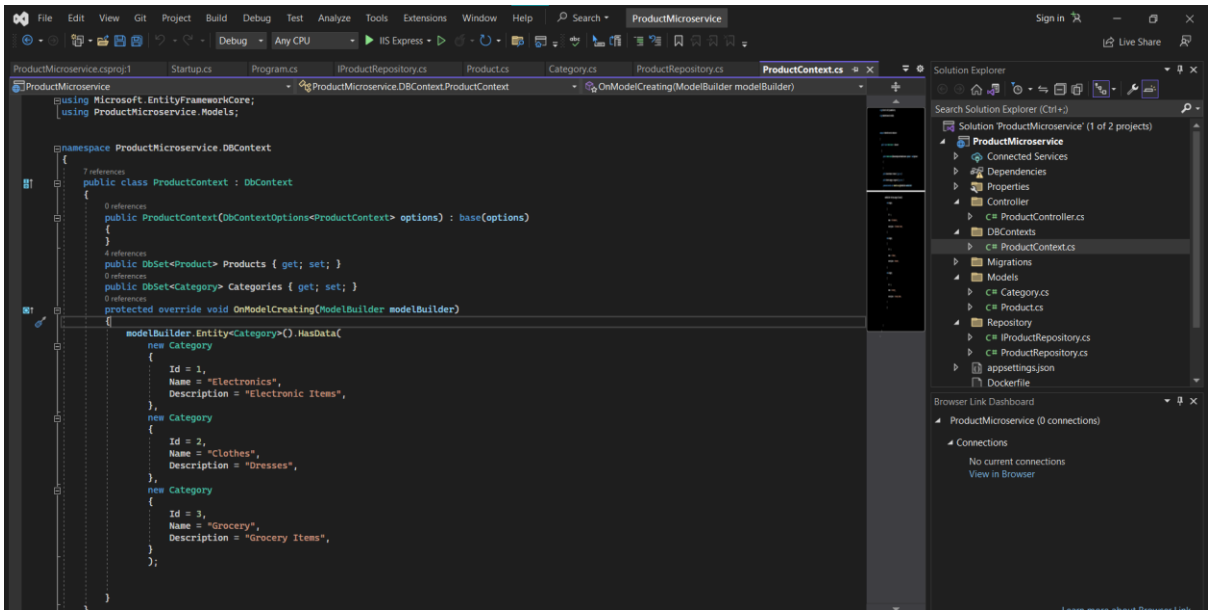


Step 6: Create an interface IProductRepository to access the ProductRepository



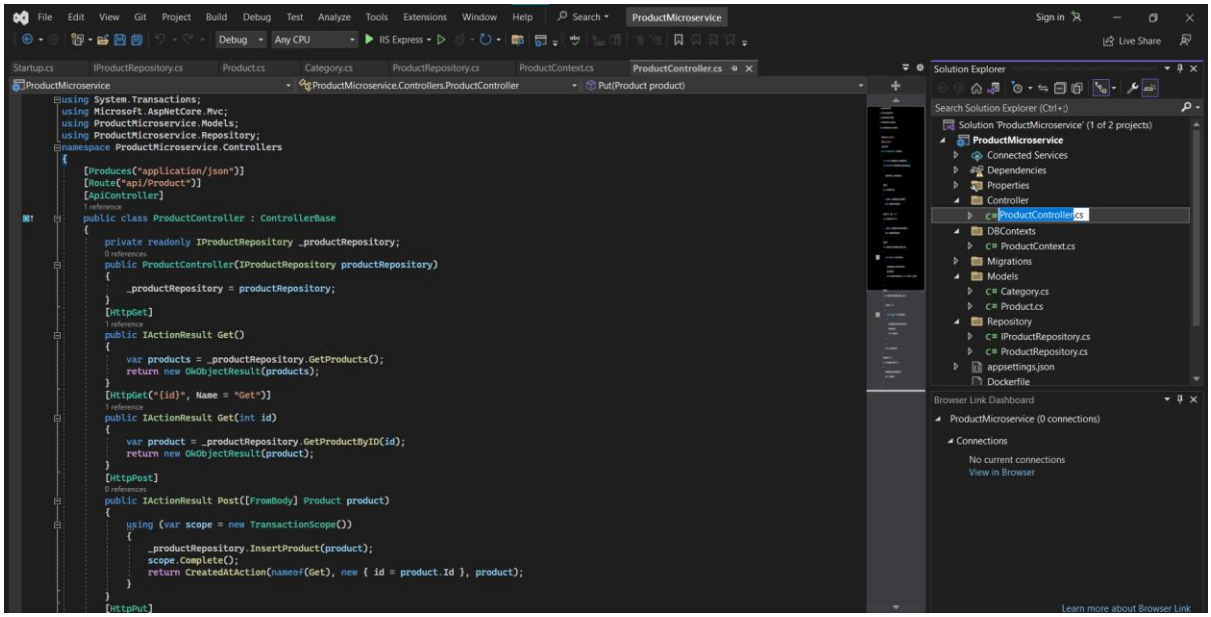
```
using System.Collections.Generic;
using ProductMicroservice.Models;

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();
    }
}
```

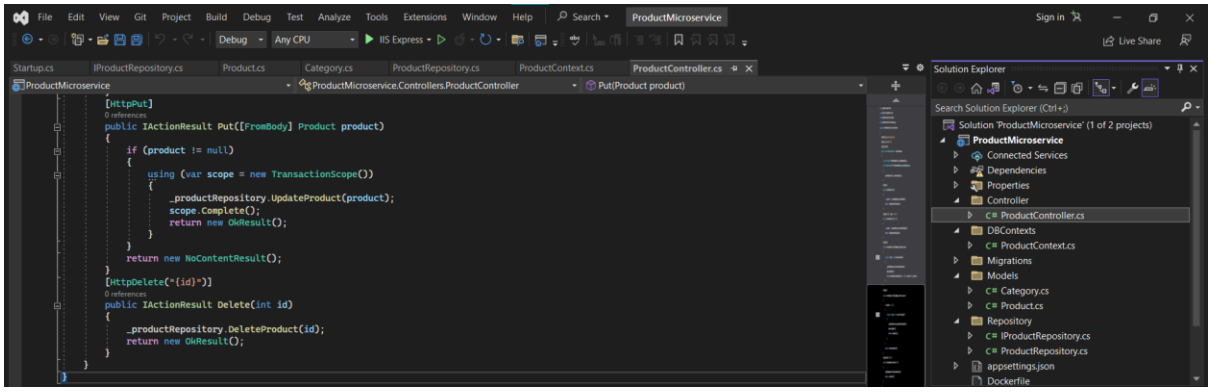
Step 7: Create a ProductContext to build the model for the database; this will be a Code First Database Approach



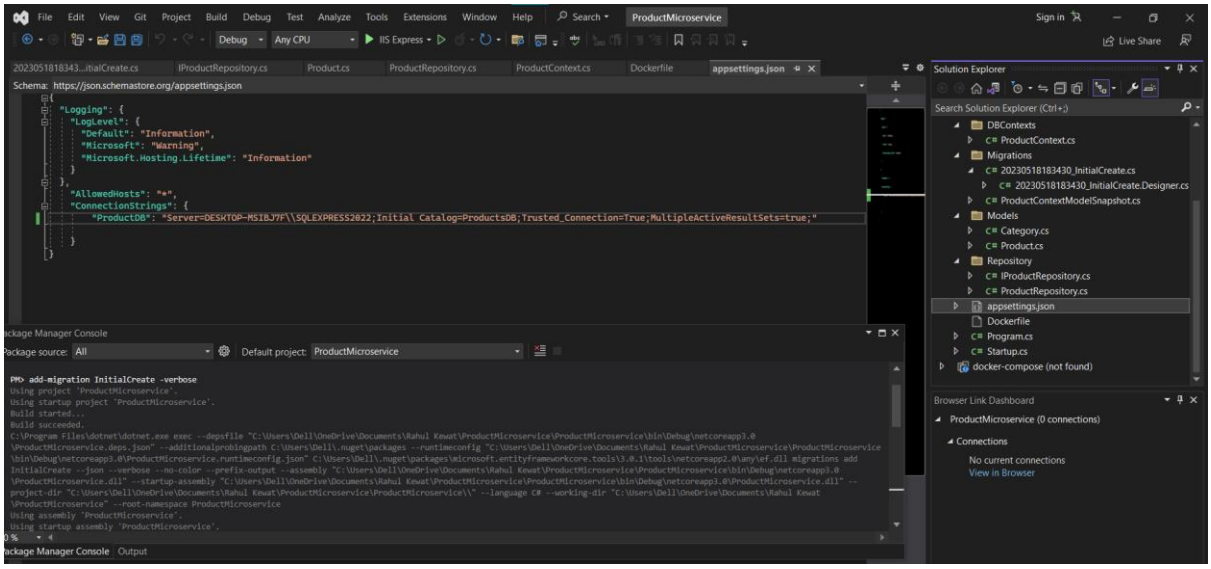
```
using Microsoft.EntityFrameworkCore;
using ProductMicroservice.Models;


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

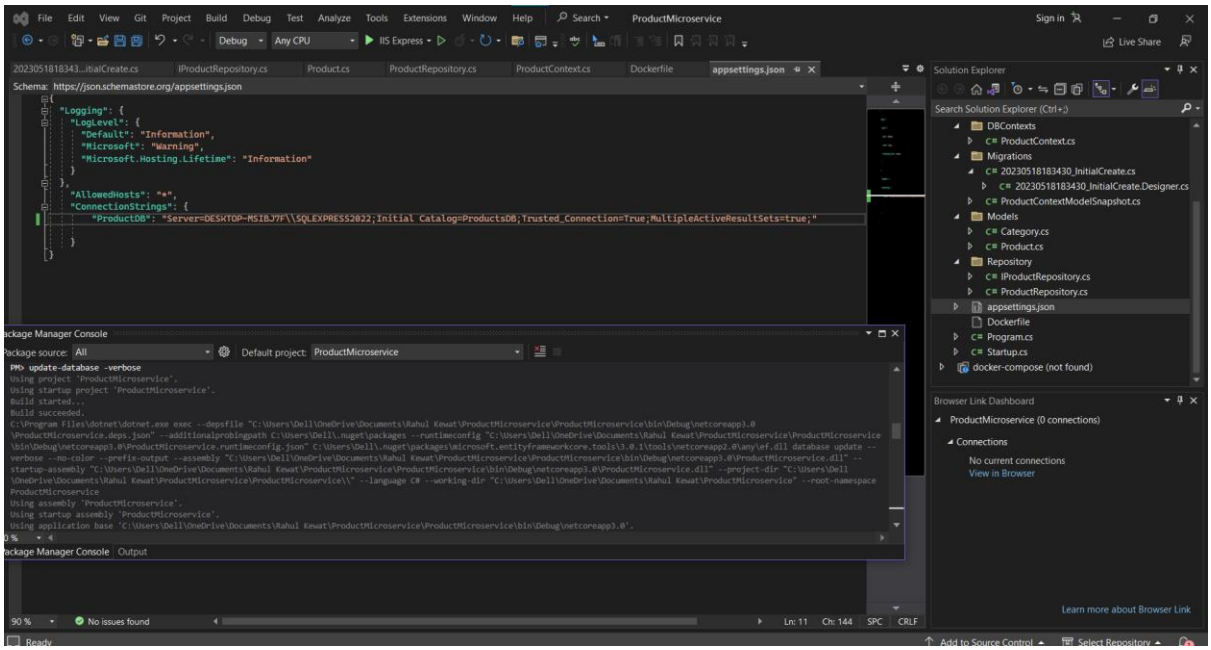




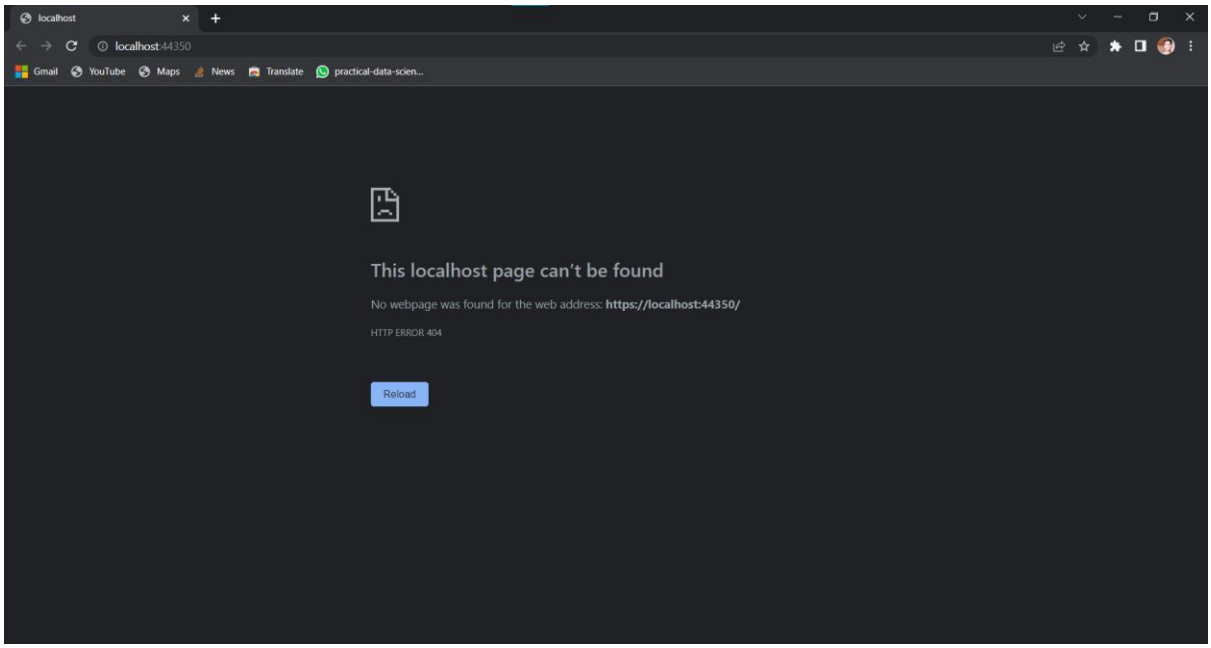
Step 9: Add the connection string settings to connect the project to the SQLSERVER database. Once added run add-migration command to run the migration



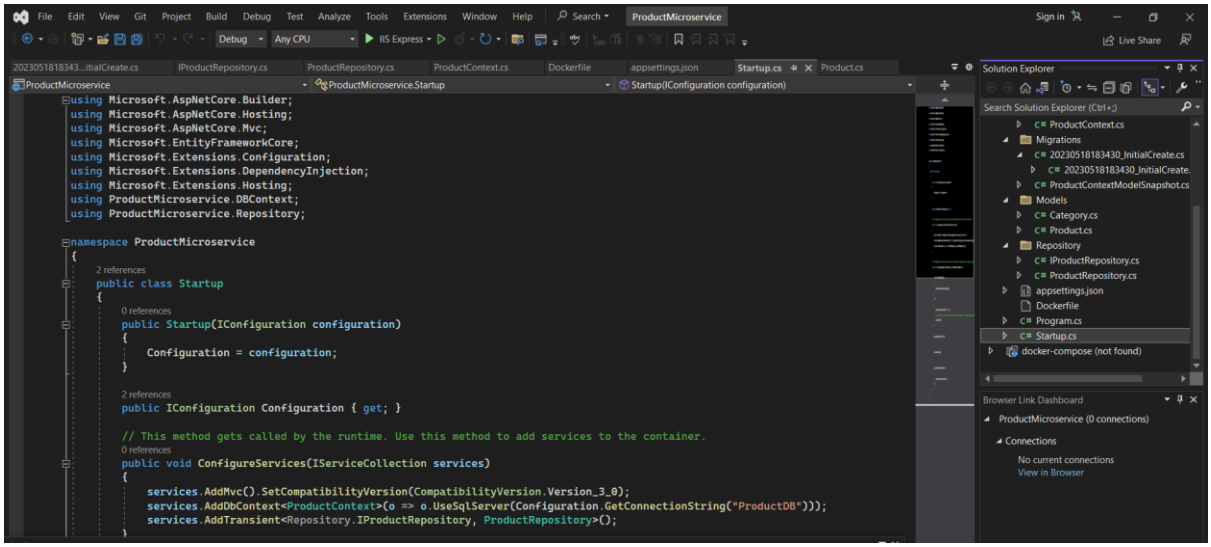
Step 10: Run the update-database command to reflect the model changes to the database



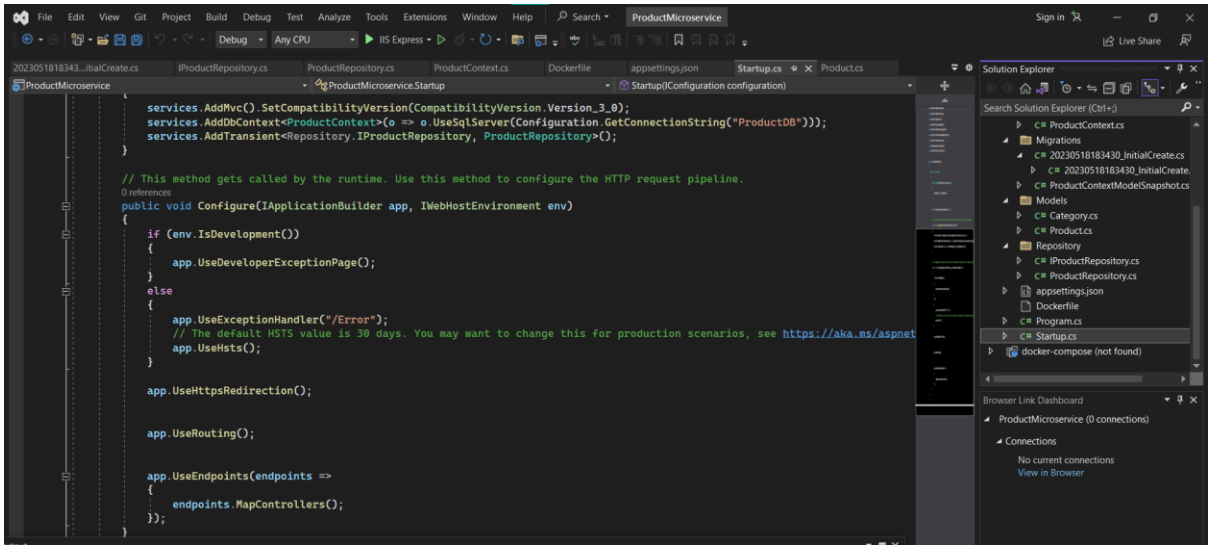
Step 11: Try to run the project, you will see that a browser window opens with an error



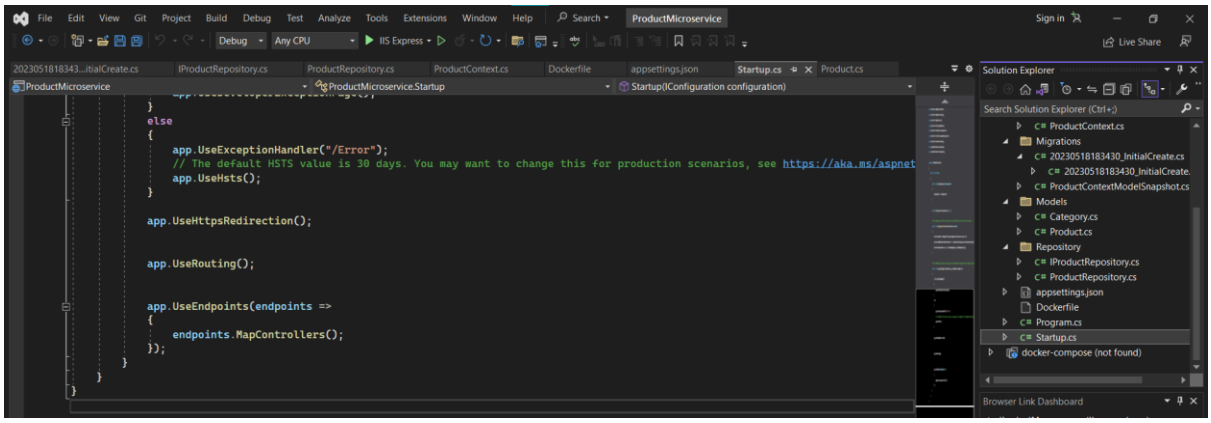
Step 12: Make some modifications to the Startup.cs file accordingly for the Routing mechanism of the API to work



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

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

        2 references
        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        0 references
        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>();
        }
```

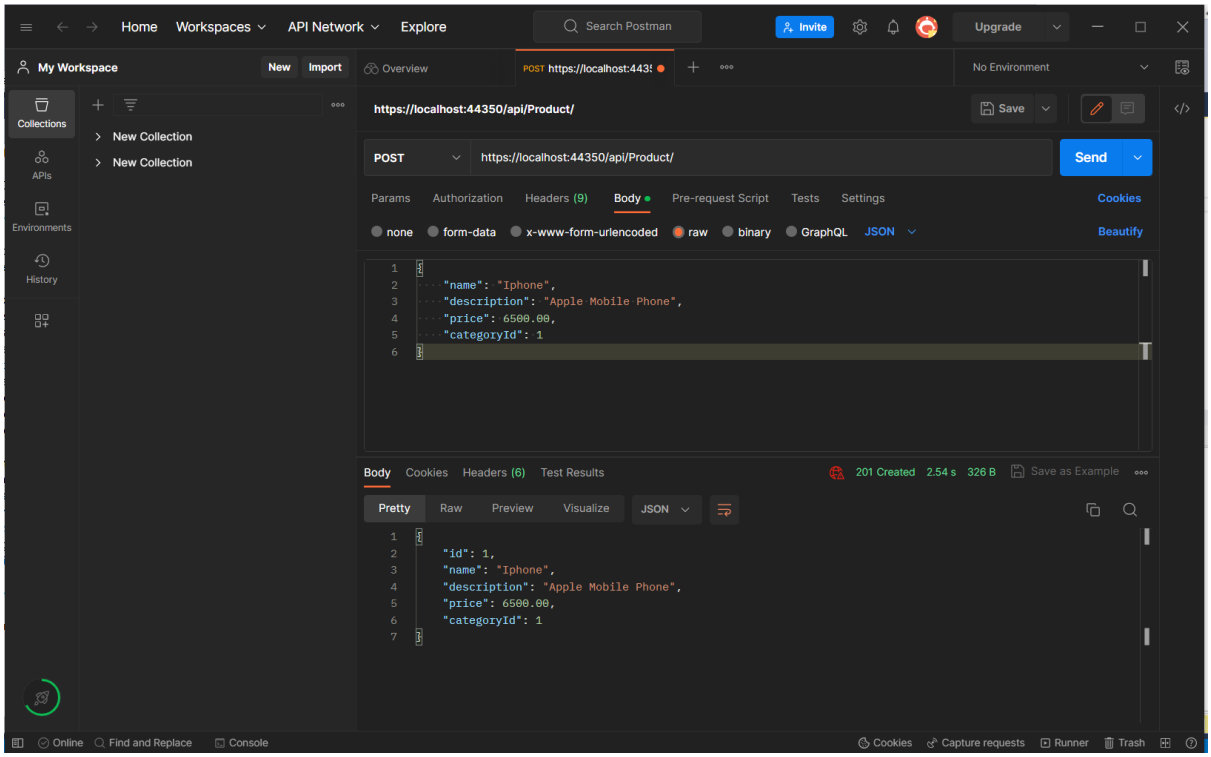


```
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        0 references
        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();
            });
        }
```

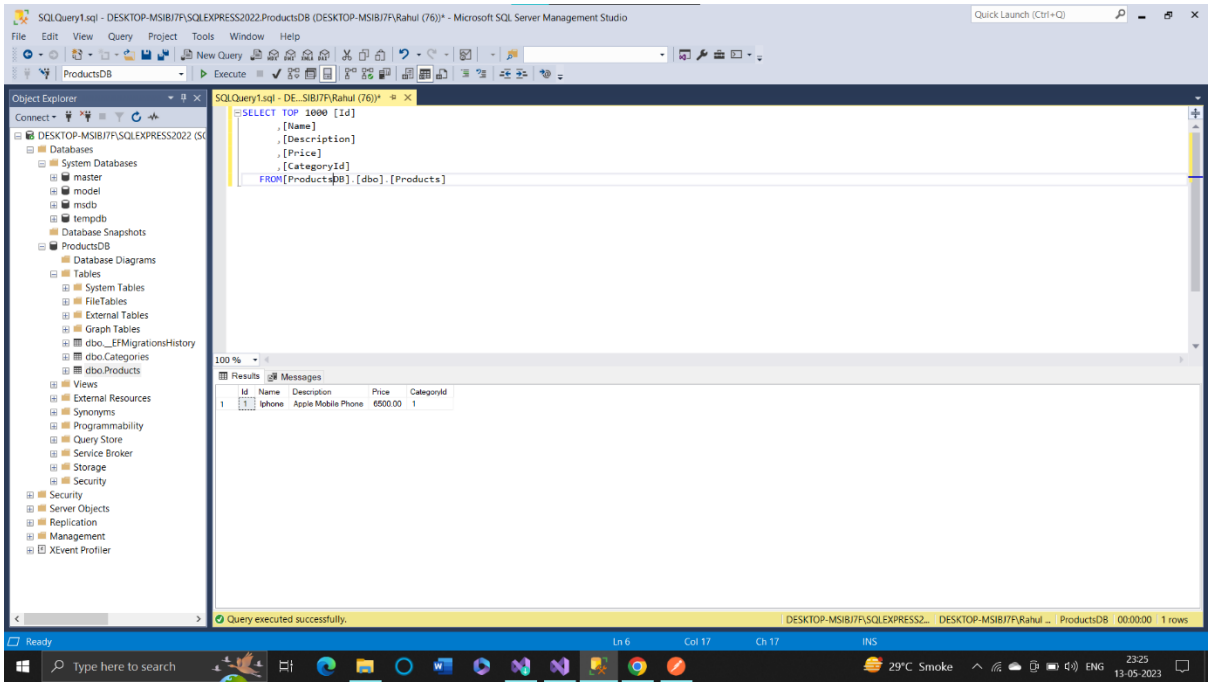


```
            }
            else
            {
                app.UseExceptionHandler("/Error");
                // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnet
                app.UseHsts();
            }

            app.UseHttpsRedirection();


            app.UseRouting();


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

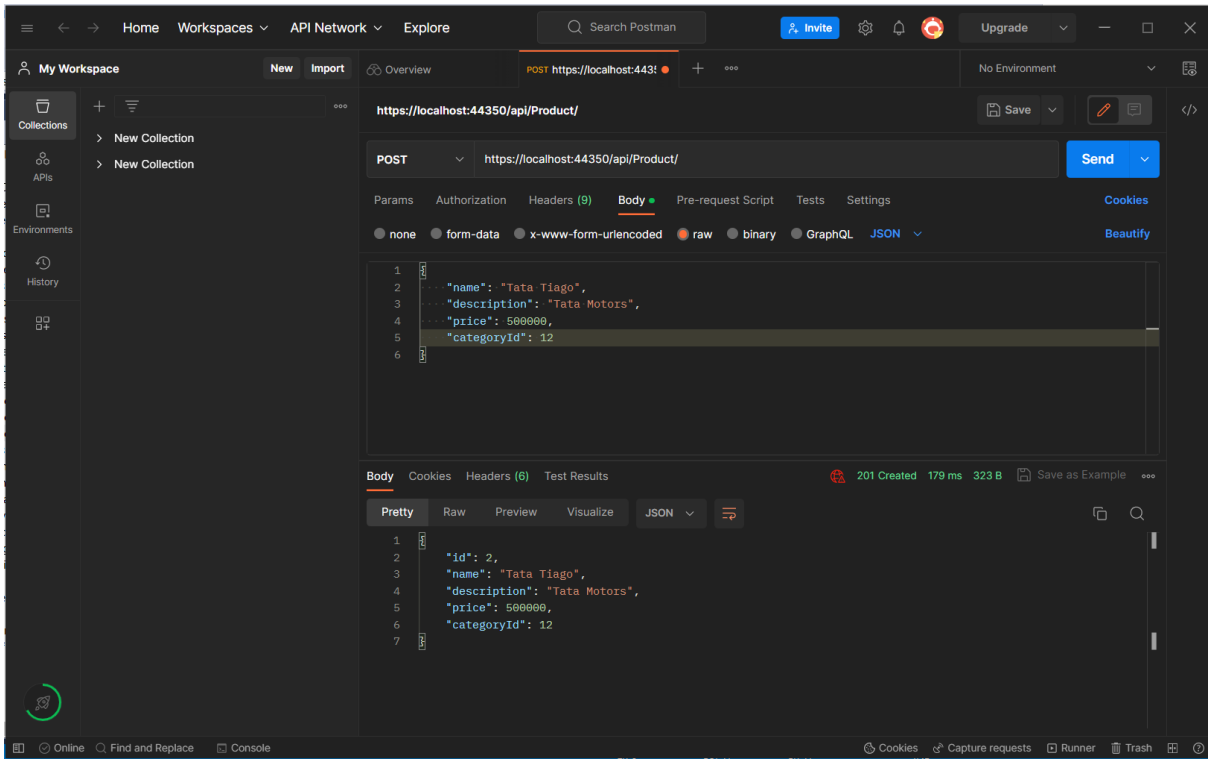
Step 13: Using Postman try to add a product into the product table via the api call



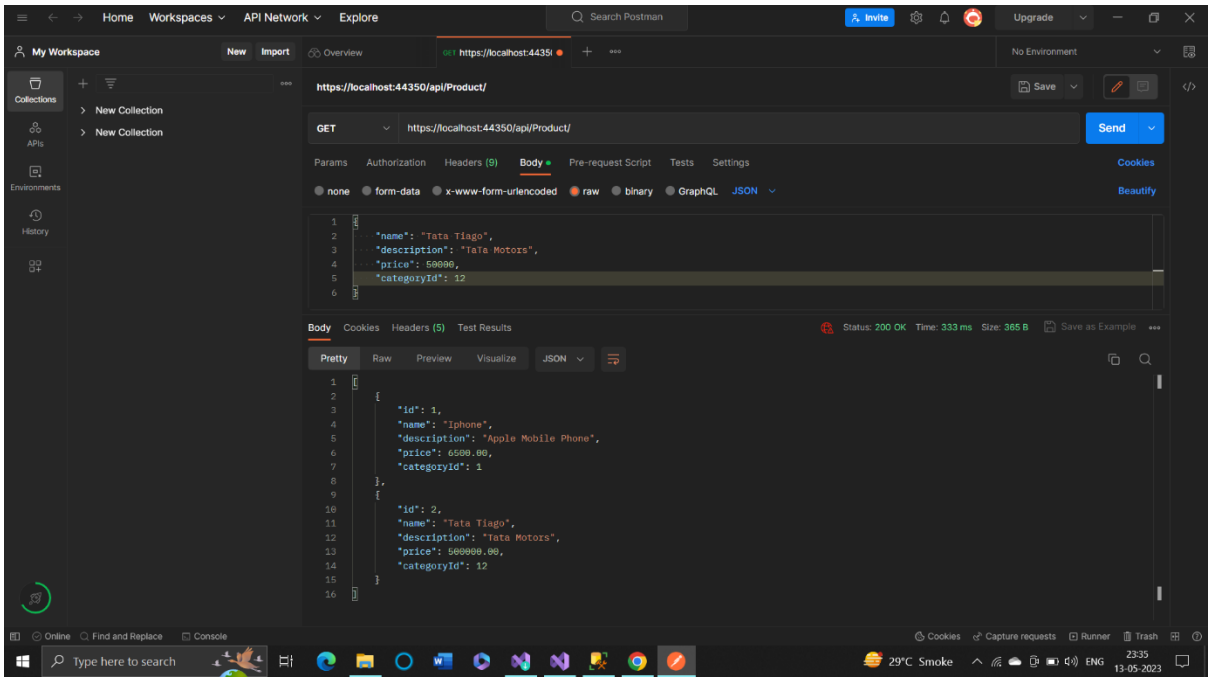
Step 14: You will notice the SQL table to populate with that product



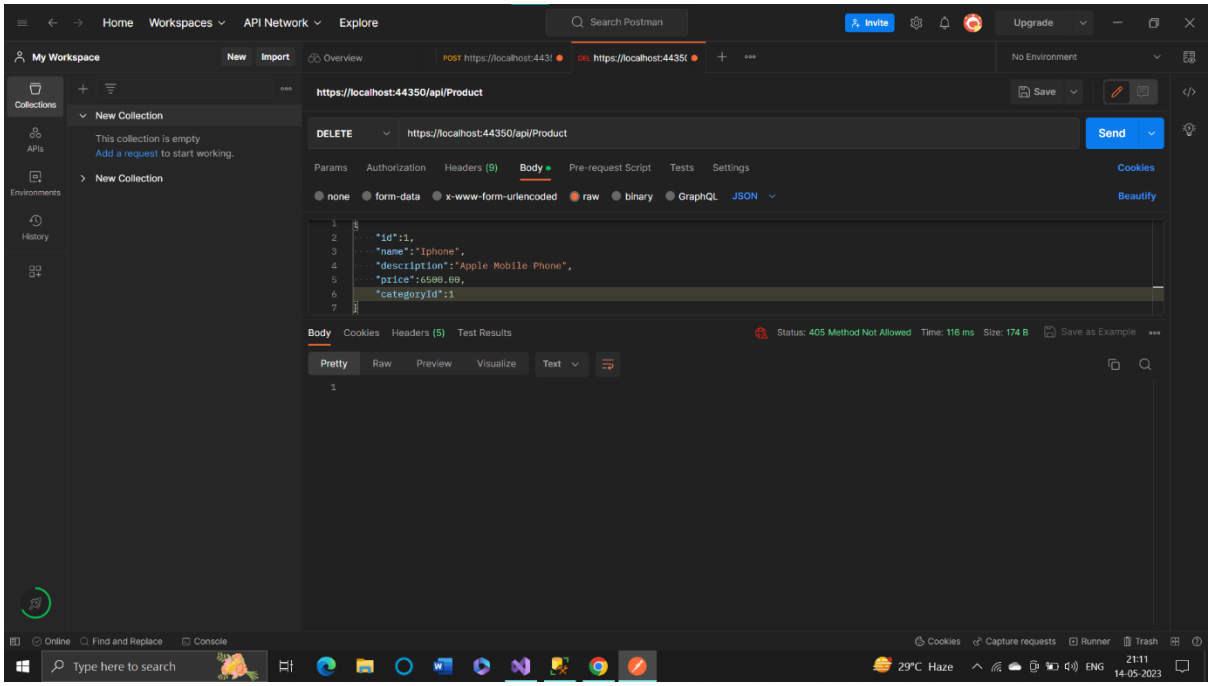
Step 15: Add another Product to test



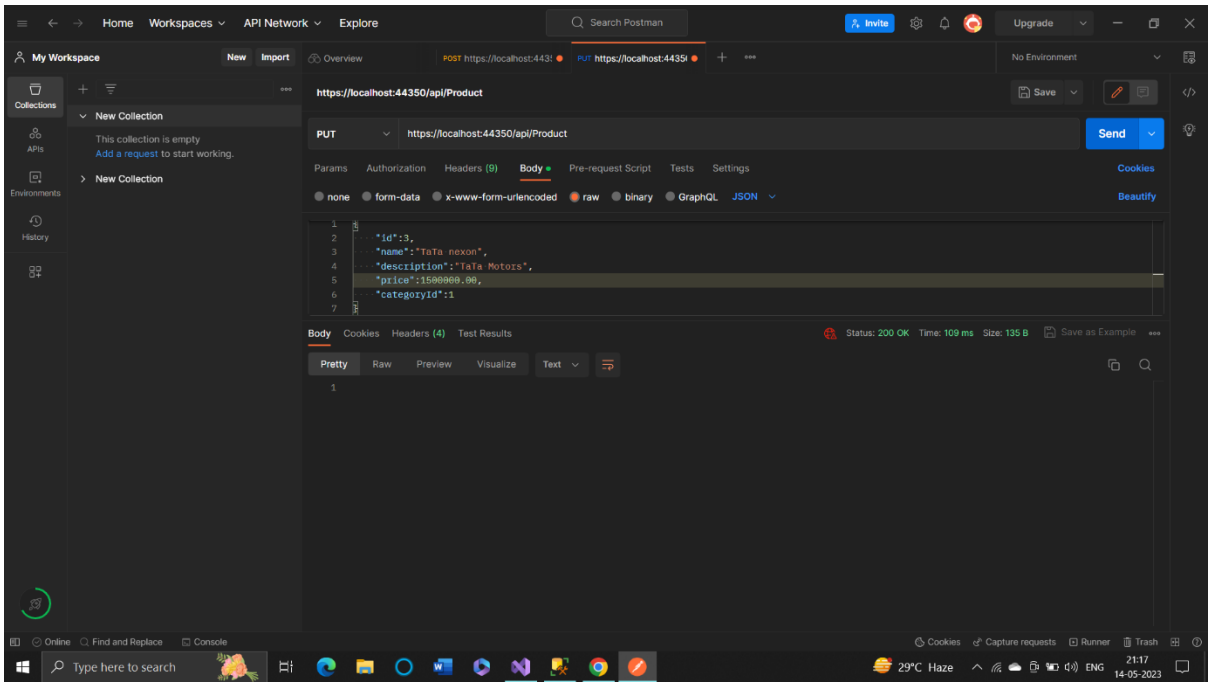
Step 16: Get all the added products using GET

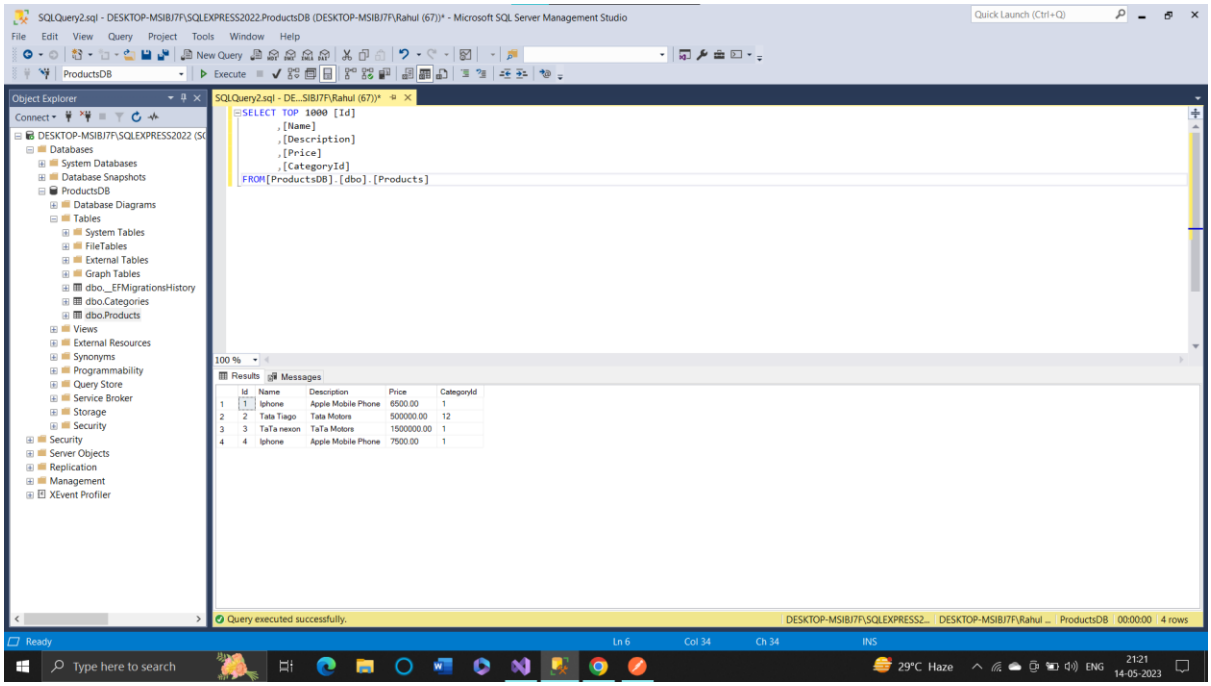


Step 17: Delete a product using DELETE



Step 18: Update a product using the PUT




SQLQuery2.sql - DESKTOP-MSIBJ7F\SQLEXPRESS2022.ProductsDB (DESKTOP-MSIBJ7F\Rahul (67))* - Microsoft SQL Server Management Studio
SELECT TOP 1000 [Id]
,[Name]
,[Description]
,[Price]
,[CategoryId]
FROM[ProductsDB].[dbo].[Products]
Query executed successfully.
Ready

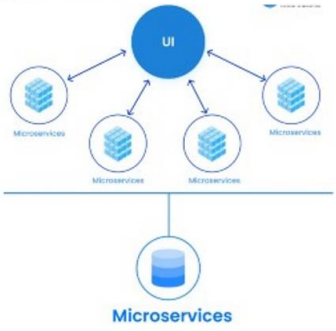
PRESENTATION

API Design for Microservices

Prepared By: Rahul Kewat
Msc-IT Part-1
Roll No-22006

what are Microservices?

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



What Are API?

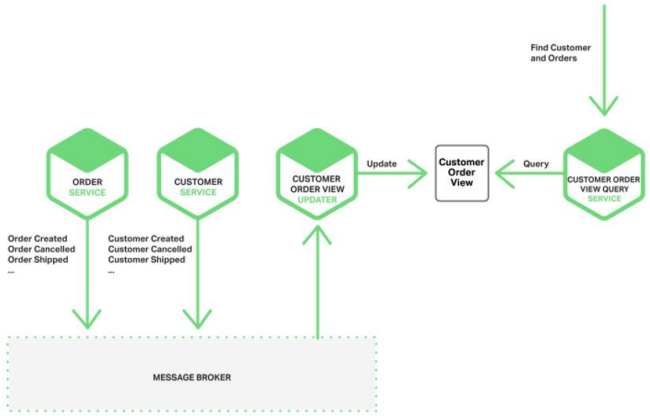
- An API is a system that allows two or more applications to communicate. Think of them as a set of procedures and functions, which allow the consumer to use the underlying service of an application.
- APIs are a part of microservices and help these services to communicate with each other.
- while communicating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

Introduction

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

Messsage-Oriented:

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



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

Hypermedia-Driven :

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

- Example: When we use google chrome or other application on a laptop, our system processor makes a queue and it's works on the fundamentals of FIFO(First in first out). To run the application on the system, CPU allocate its process id and port number.

- HTTP message are sent to an Ip Address and a port number i.e. 80/ 443/88 and message contains the data and action which includes data in a HTML format.

Thank you