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

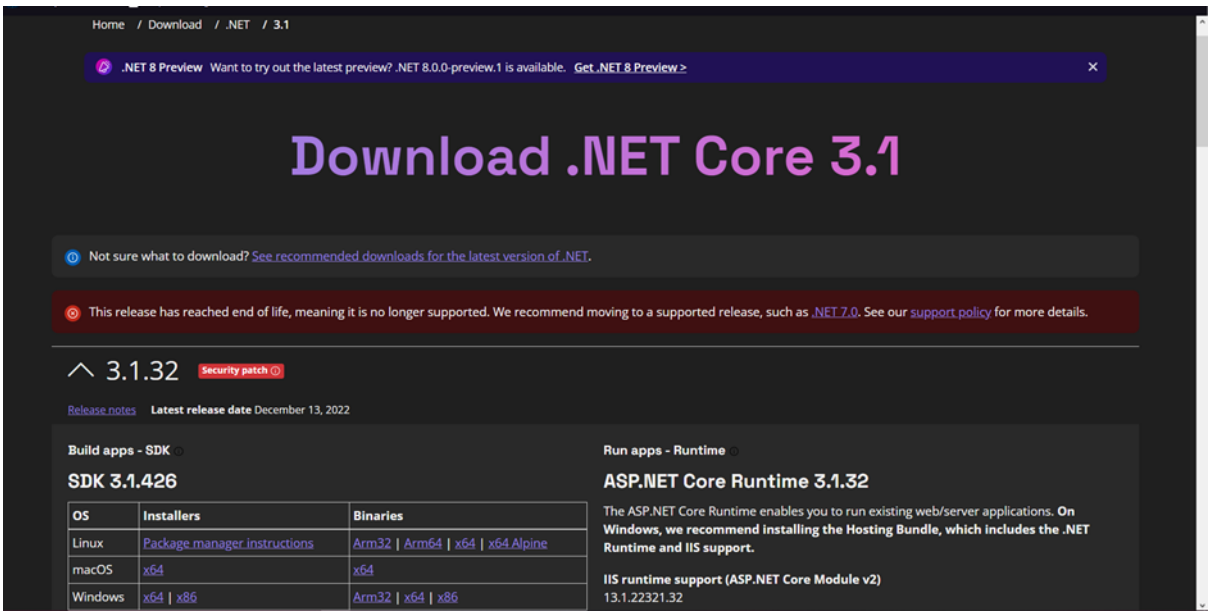
Practical 1

Aim: Creating a basic MVC application using .NET CORE

Writeup:

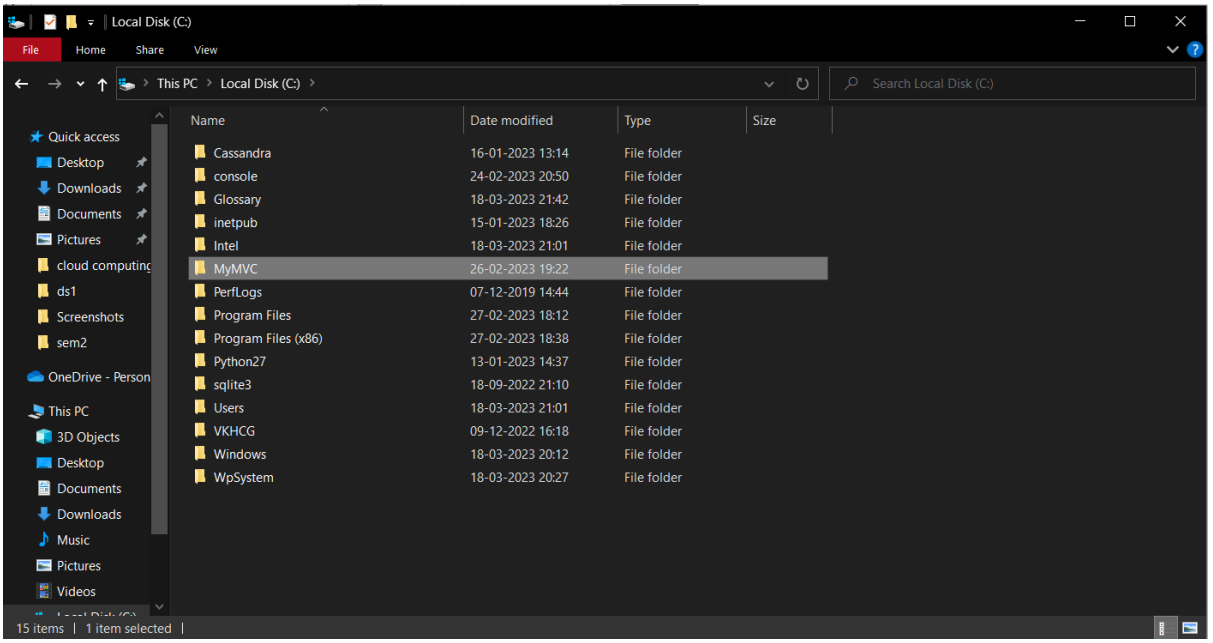
Step 1:

- Install .Net Core SDK.



Step 2:

- Create a Folder MyMVC in C: drive.

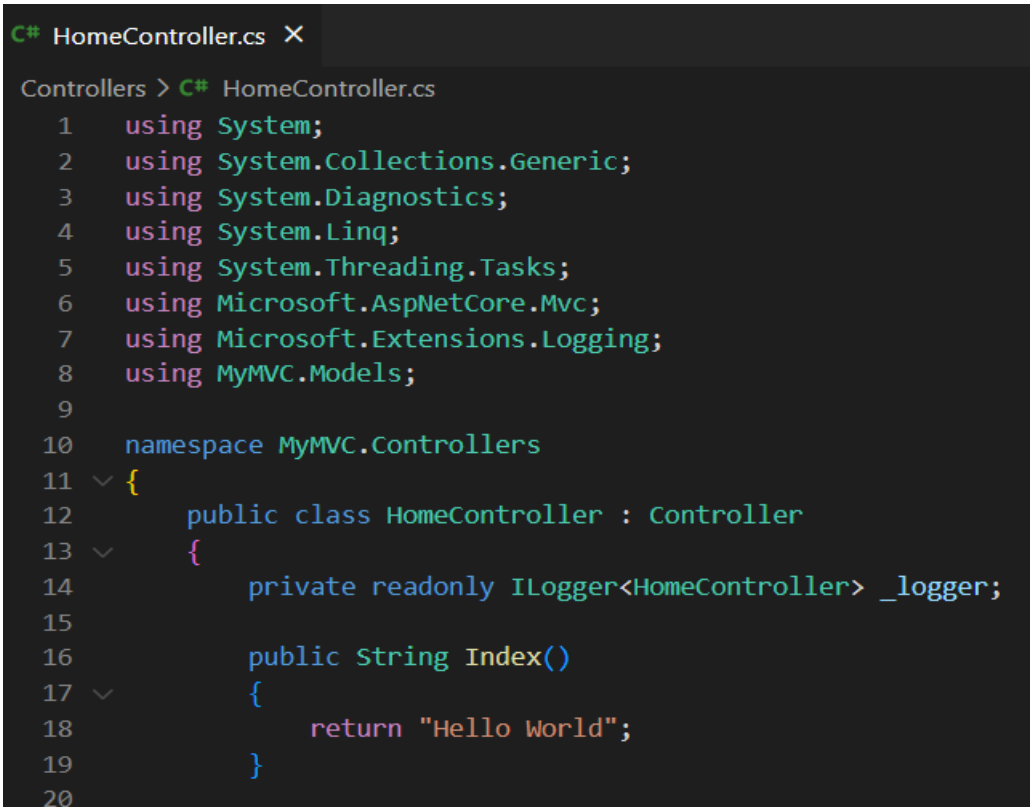


Step 3:

- 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:

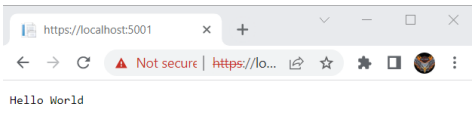


Step 5:

- Run the code

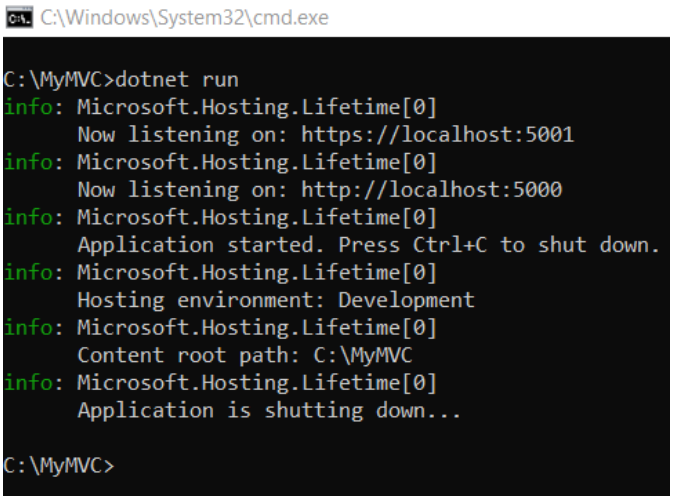
```
C:\MyMVC>dotnet run
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]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



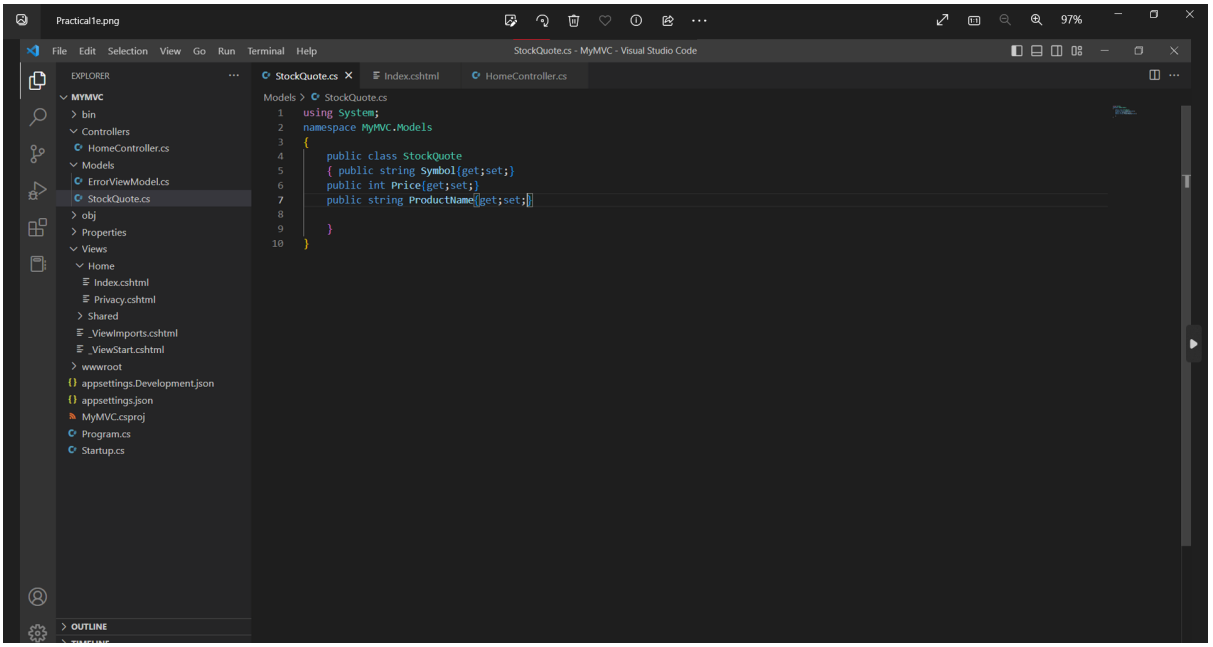
Step 6:

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



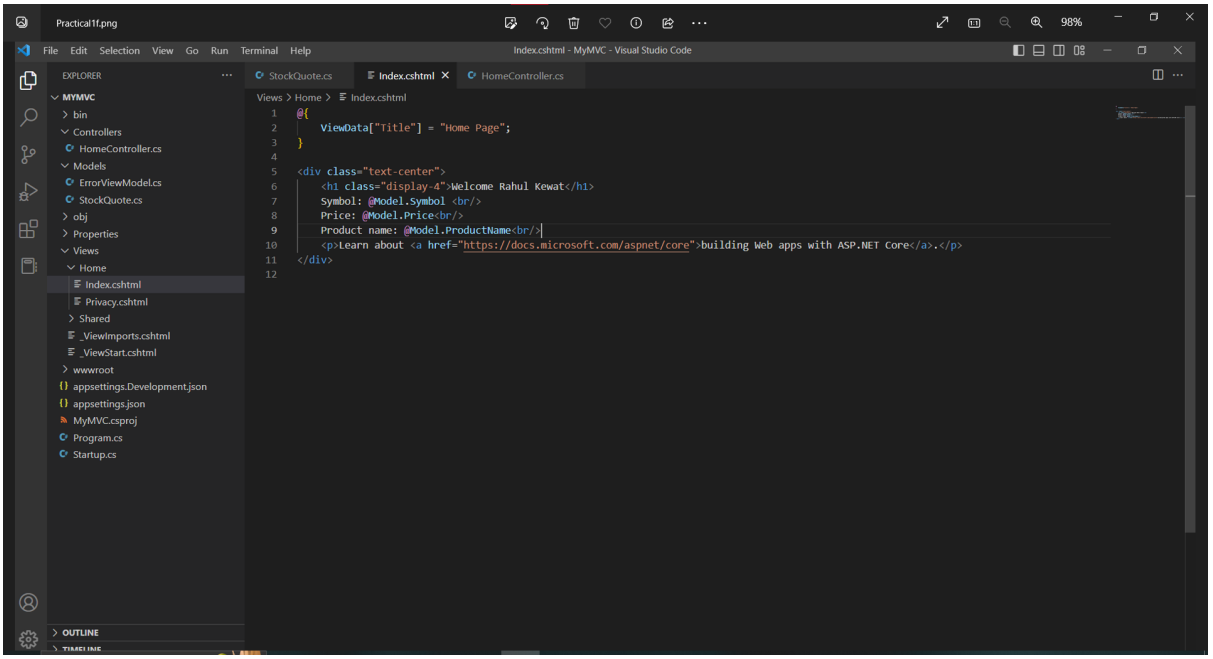
Step 7:

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



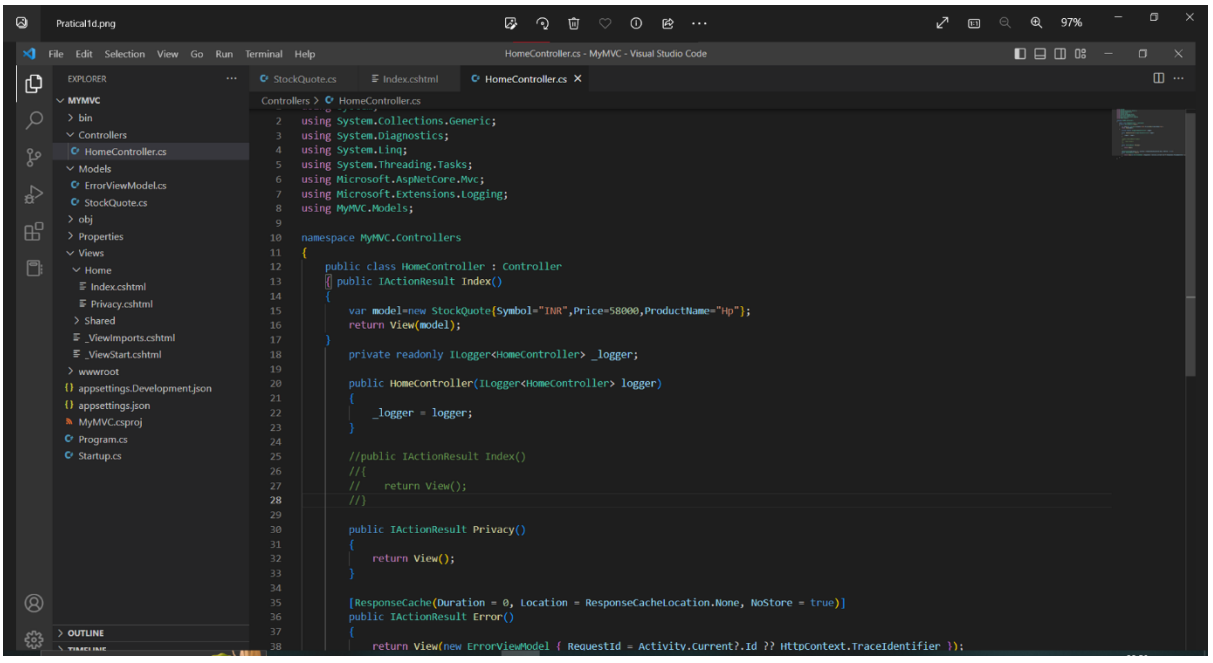
Step 8:

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



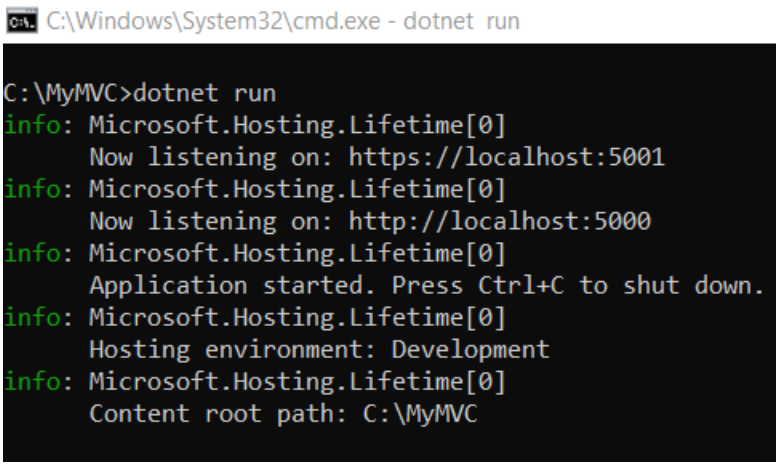
Step 9:

- Now modify HomeController.cs file to match following:



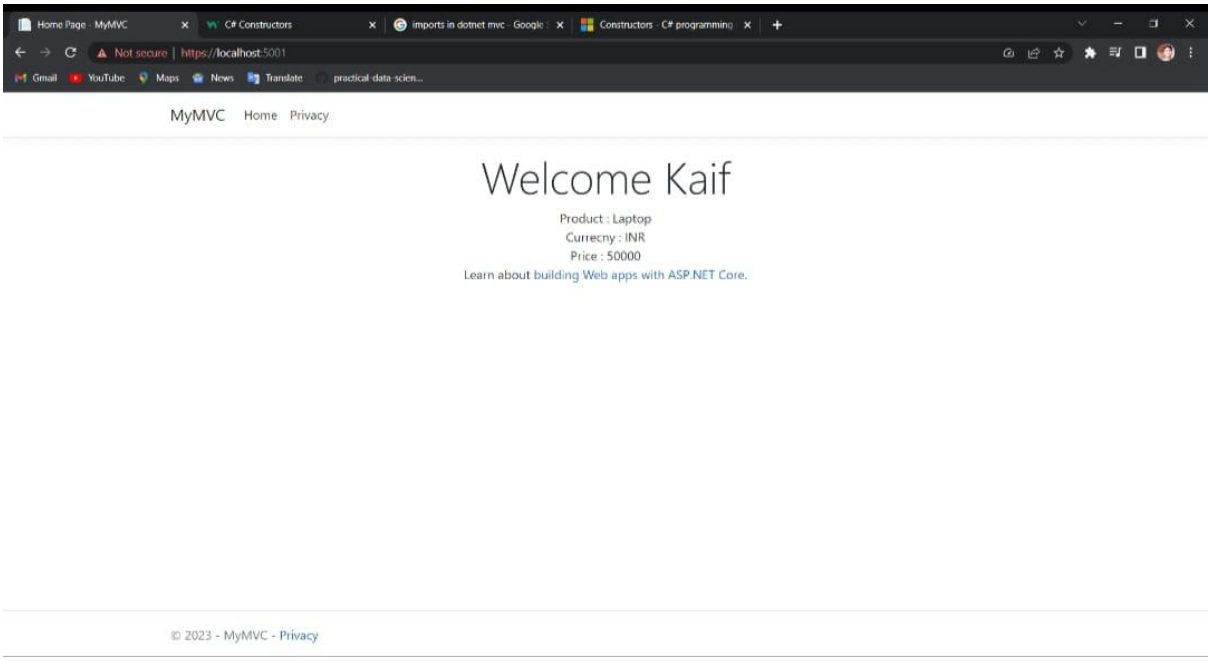
Step 10:

- 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:

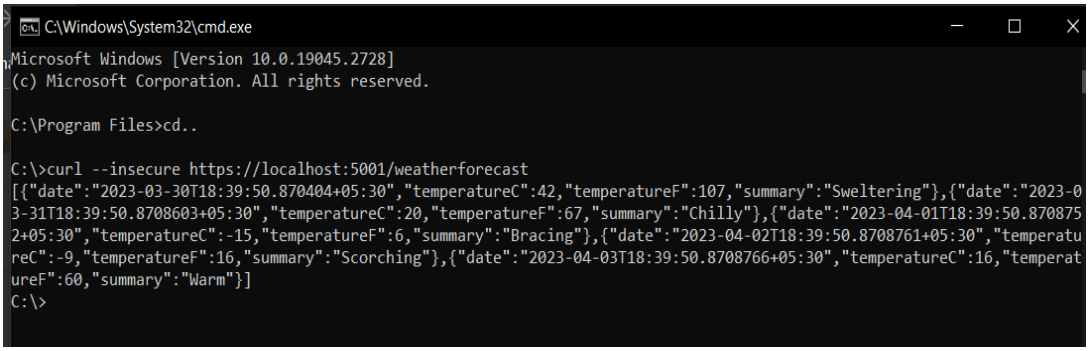
cd Glossary

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)

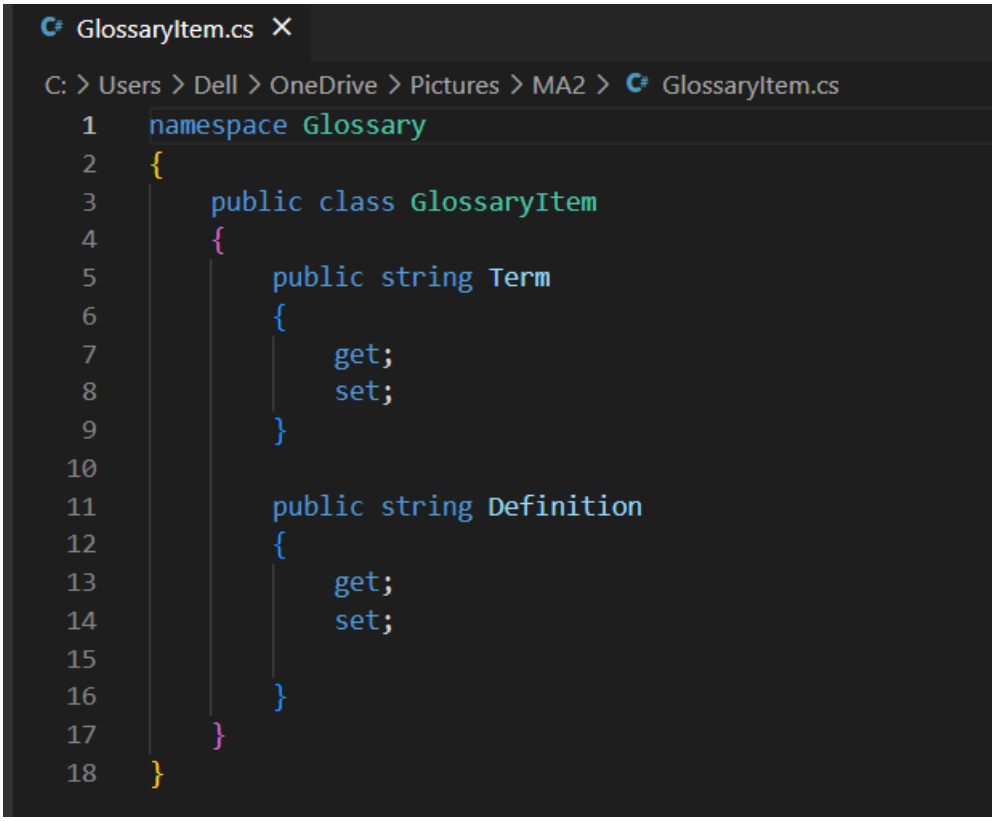
curl --insecure https://localhost:5001/weatherforecast



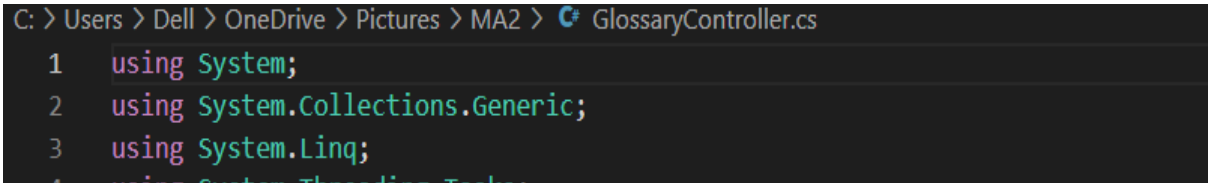
Step 3: Now Change the content:-

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

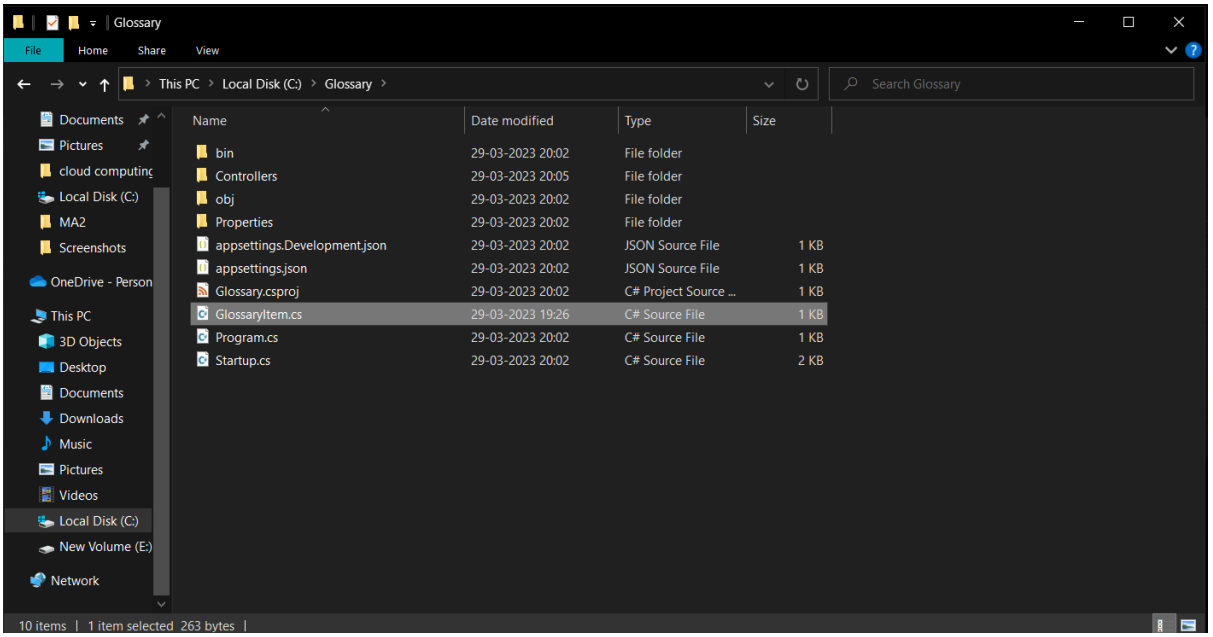
```
36          [HttpGet]
37          public ActionResult<List<GlossaryItem>> Get()
38          {
39              return Ok(Glossary);
40          }
41
42          [HttpGet]
43          [Route("{term}")]
44          public ActionResult<GlossaryItem> Get(string term)
45          {
46              var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

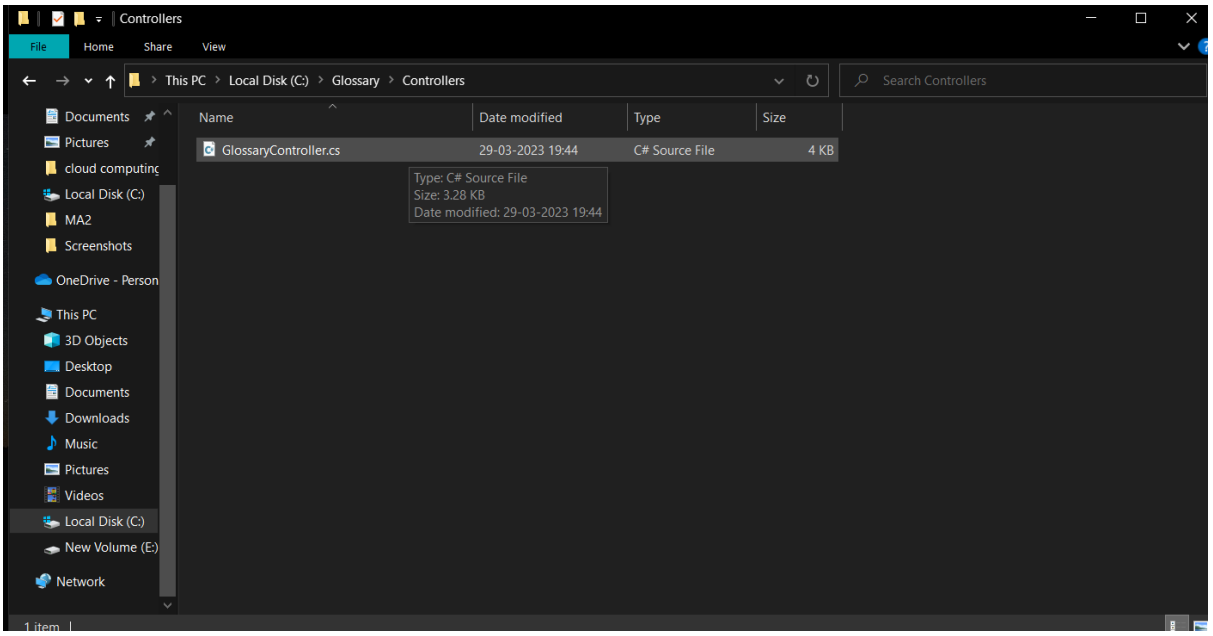
C: > Users > Dell > OneDrive > Pictures > MA2 > GlossaryController.cs

```
            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();
            }
```

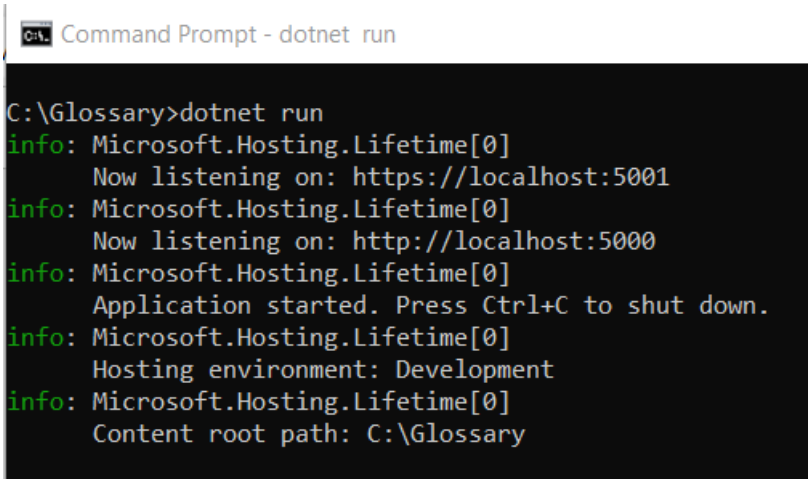
```
            else
            {
                Glossary.Remove(glossaryItem);
                return NoContent();
            }
        }
    }
}
```

Output:





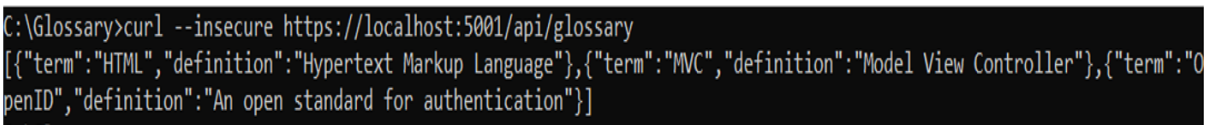
Step 4: Now stop running previous **dotnet run** on command prompt 1 using **Ctrl+C**. and Run it again for new code.



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

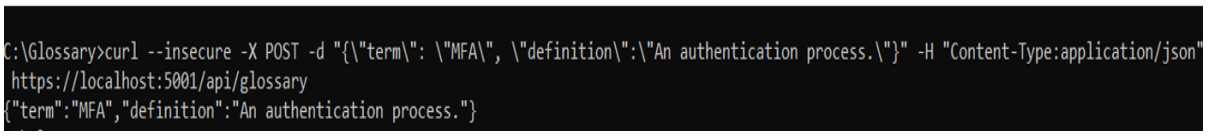


[B]: Getting a single item.

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."}]
```

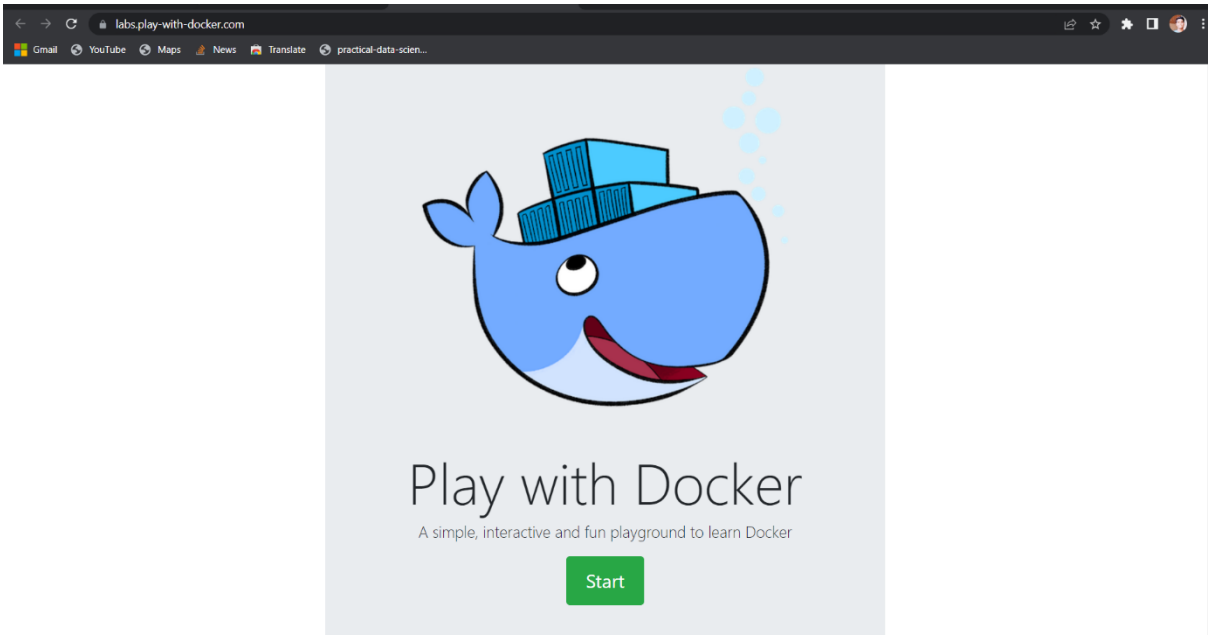
Practical 3

Aim: Working with Docker

Writeup:

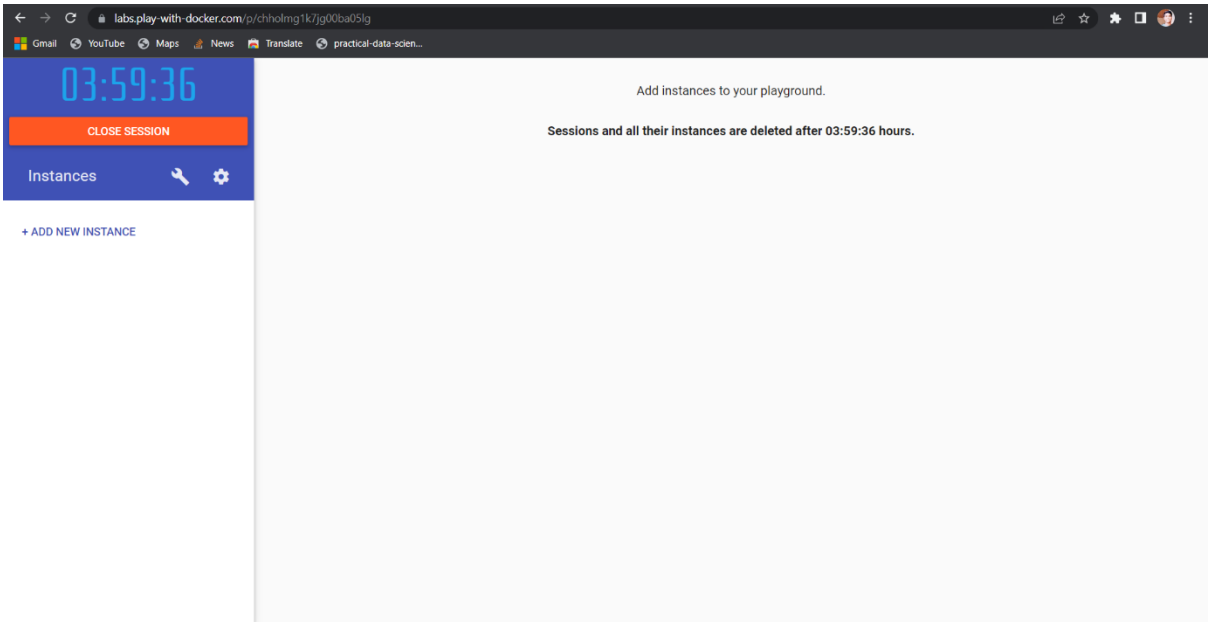
Step 1: create Docker Hub account (sign up)

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



Click on start

Step 3: add new instance



Step 4: perform following:

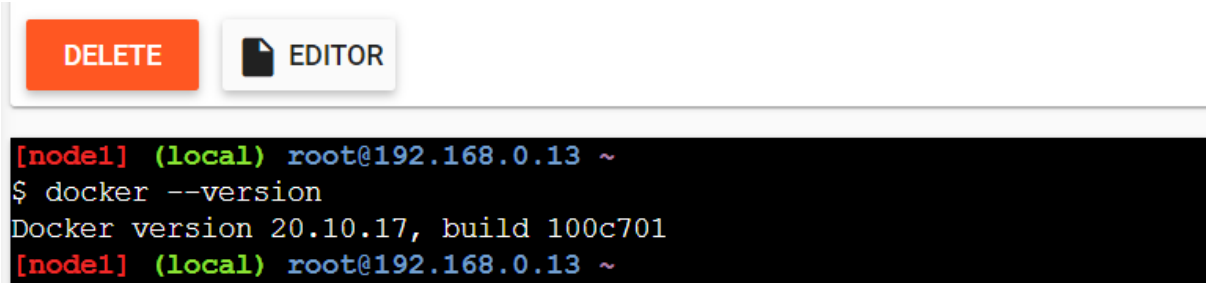
Method1:

To pull and push images using docker

Command: to check docker version

docker –version

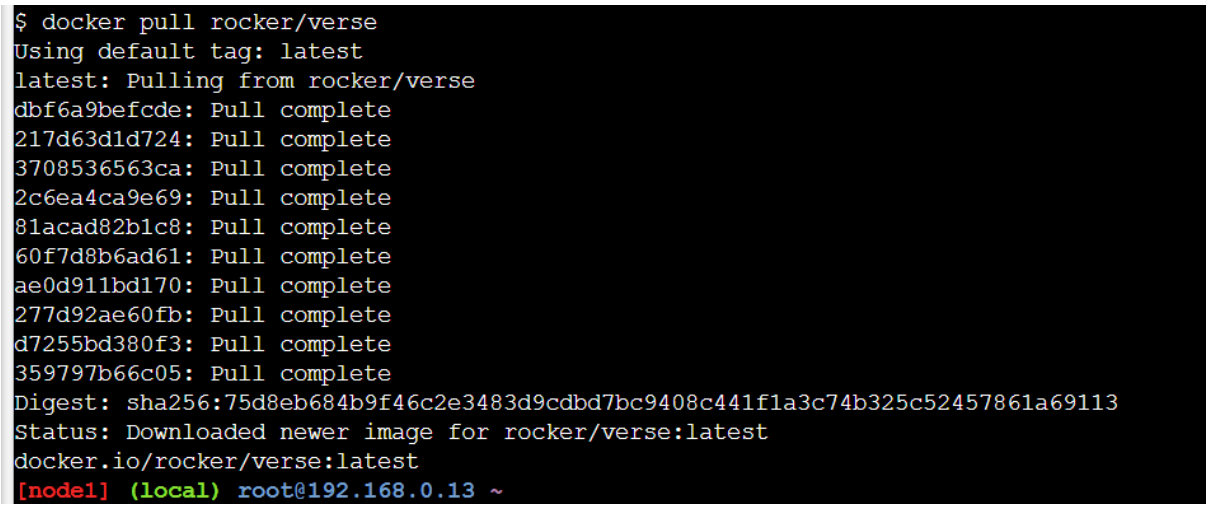
Output:



Command: to pull readymade image

docker pull rocker/verse

Output:



Command: to check images in docker

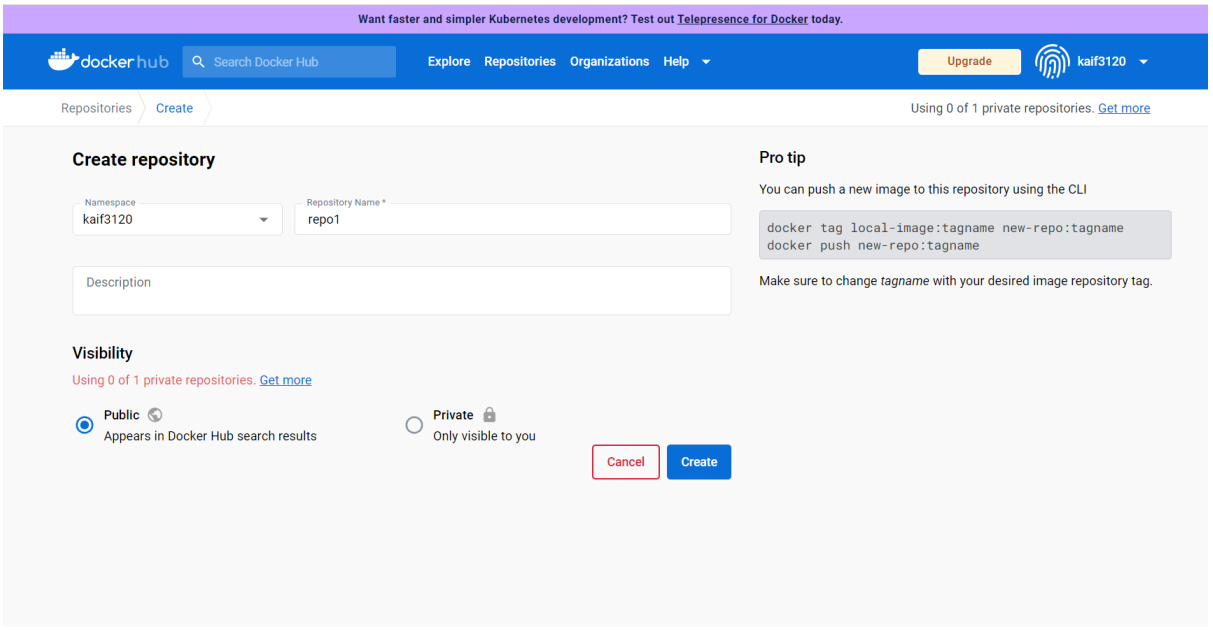
docker images

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 ~
```

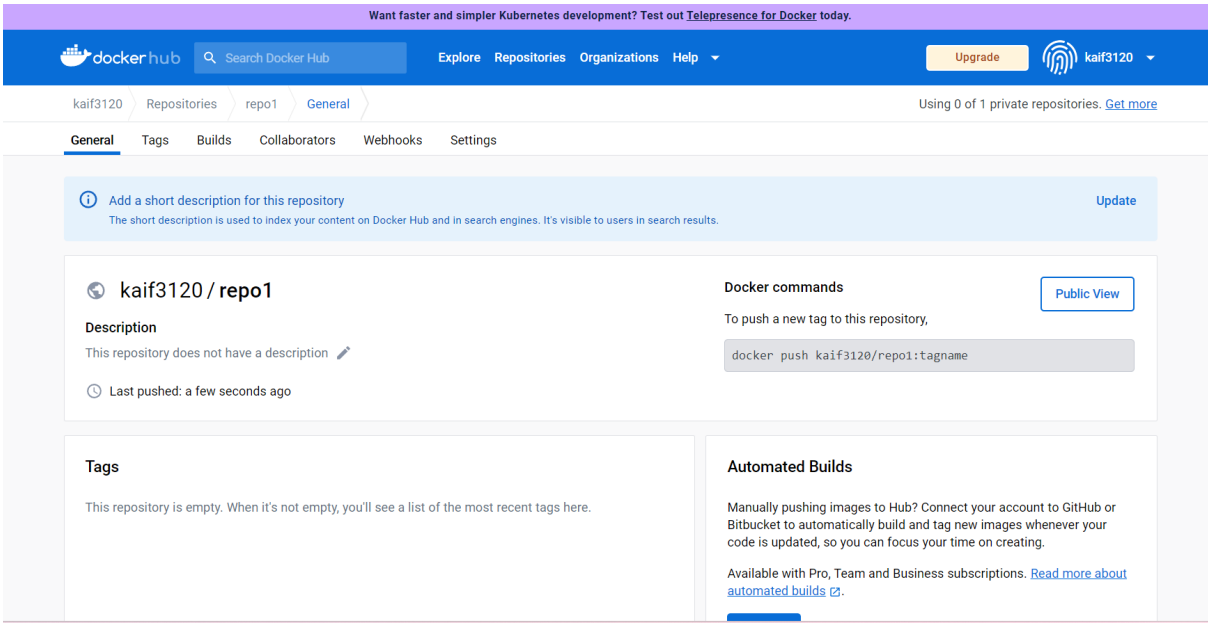
Now Login to docker hub and create repository

Output:



Click on Create button

Now check repository created

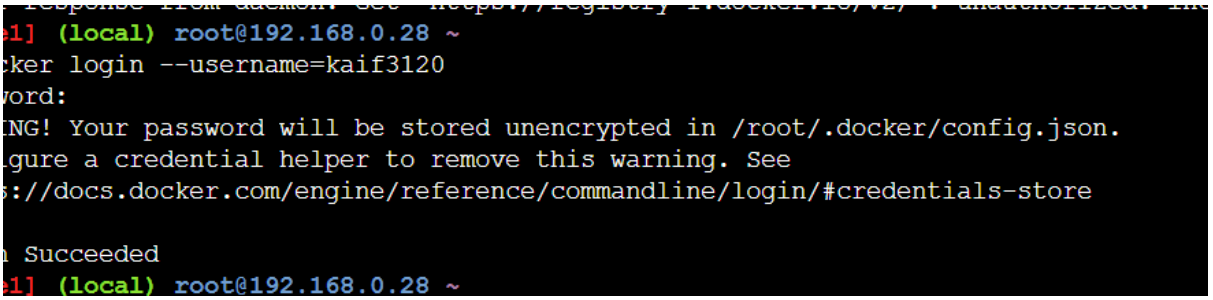


Command: to login to your docker account

docker login –username=kaif3120

password:

Output:



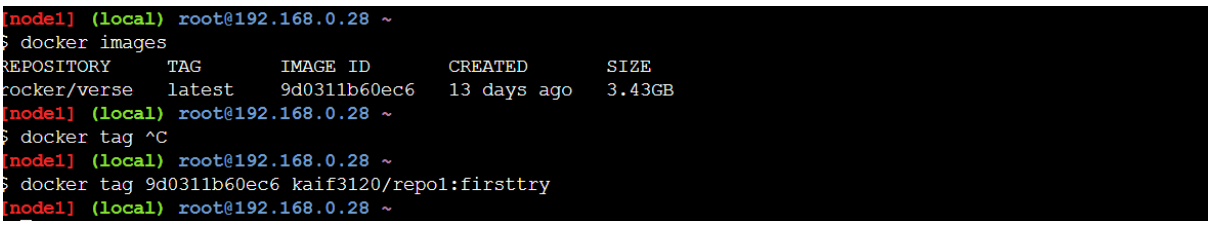
Command : to tag image

docker tag 9d0311b60ec6 kaif3120/repo1:firsttry

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

command.

Output:



Command: to push image to docker hub account

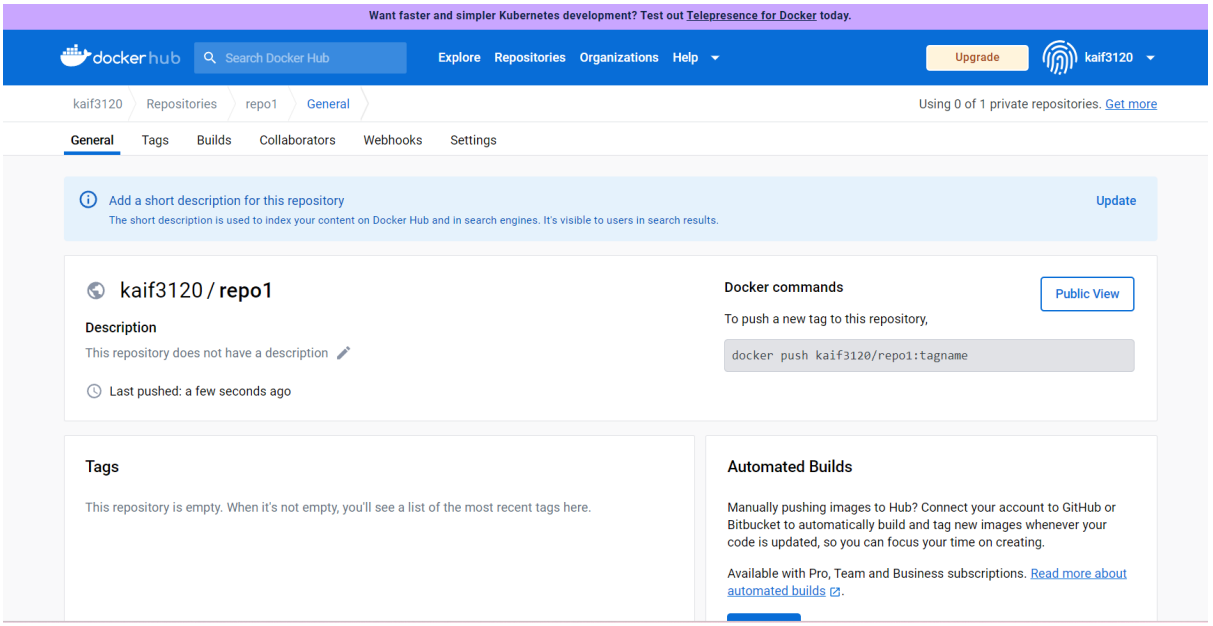
docker push kaif3120/repo1:firsttry

note: firsttry is tag name created above.

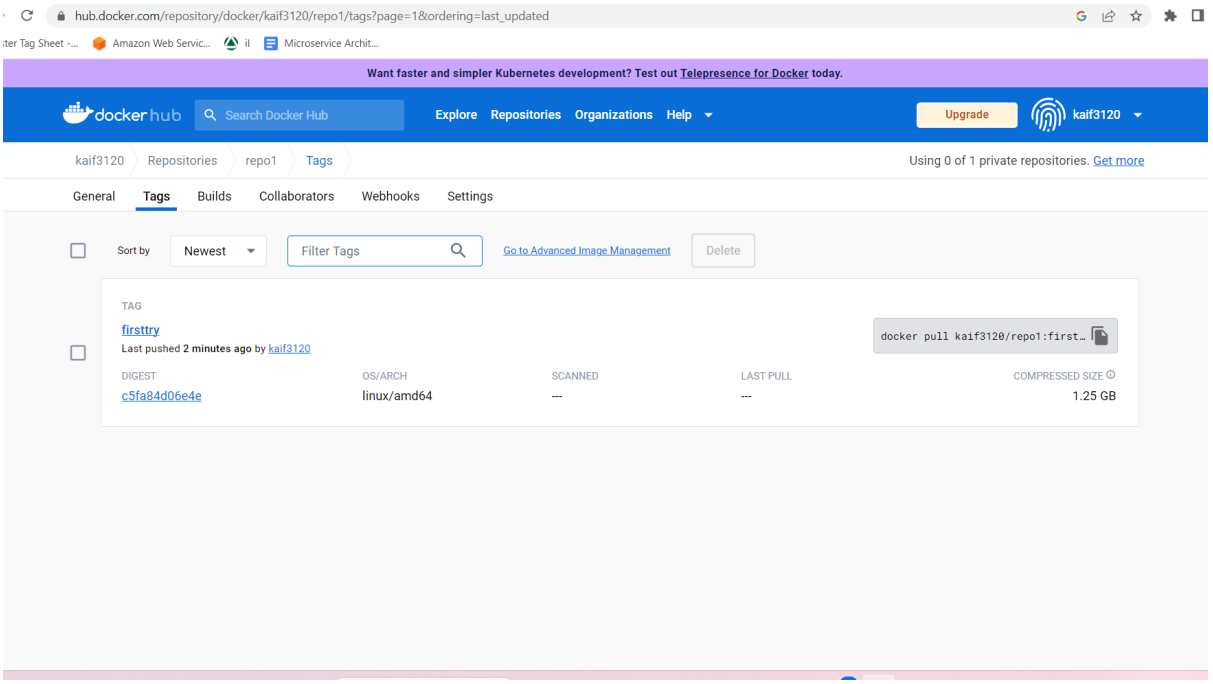
Output:

```
(local) root@192.168.0.28 ~
er push kaif3120/repo1:firsttry
sh refers to repository [docker.io/kaif3120/repo1]
84033f: Mounted from rocker/verse
lab534: Mounted from rocker/verse
d6f055: 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
ry: digest: sha256:c5fa84d06e4e536bfde730e90e0b54f7714c222d572d18e60c047c5839b76f57 size: 2428
(local) root@192.168.0.28 ~
```

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:

```
e1] (local) root@192.168.0.28 ~
t > Dockerfile <<EOF
OM busybox
D echo "hello world, this is my first docker image"
F
e1] (local) root@192.168.0.28 ~
cker build -t kaif3120/repo2
ker build" requires exactly 1 argument.
'docker build --help'.

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

d an image from a Dockerfile
e1] (local) 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

```
r images
ORY        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
```

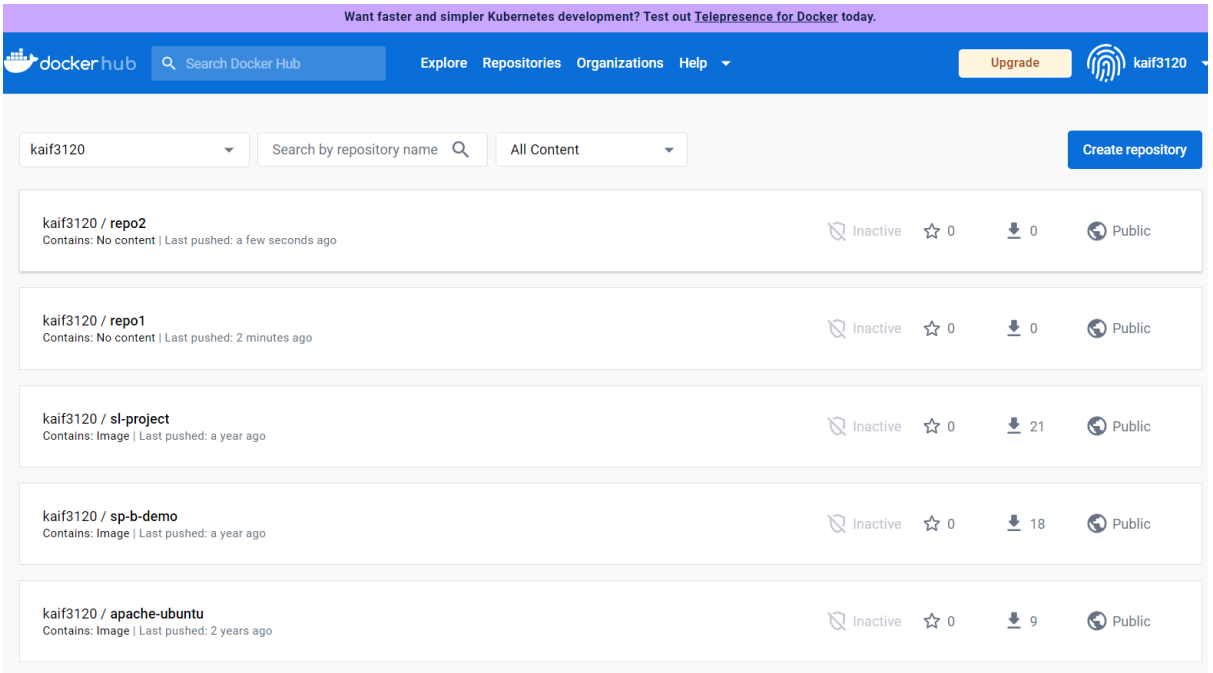
Command: to push image to docker hub

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



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

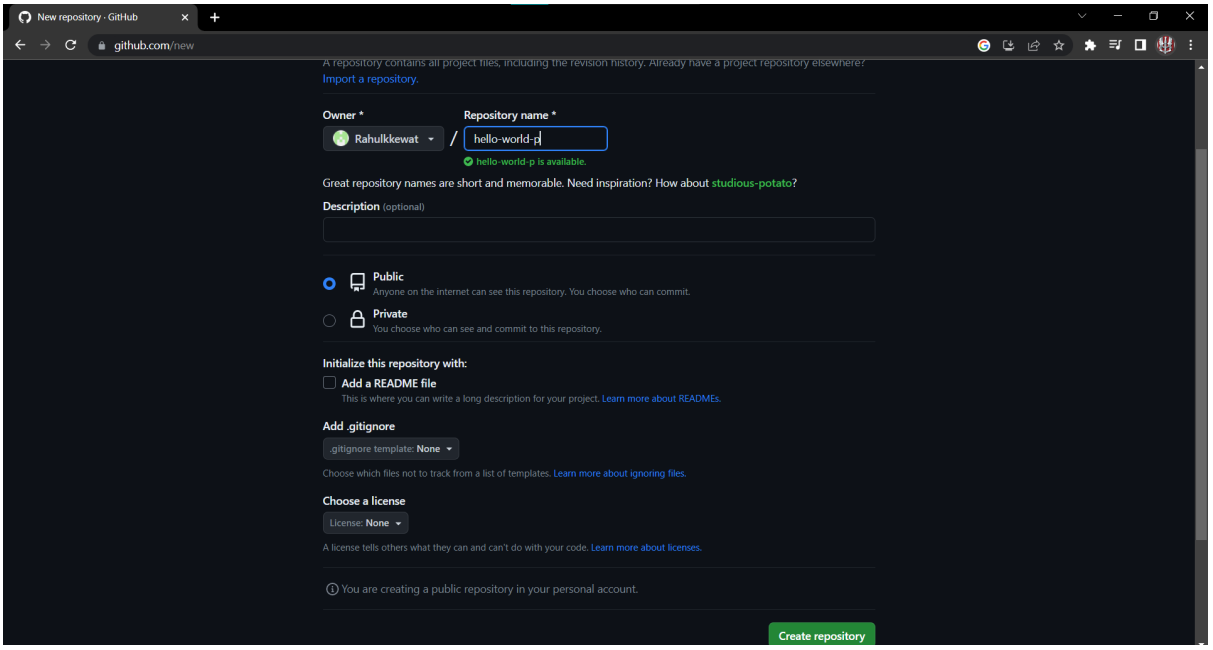
1. Log in to GitHub and begin the process to create a new repository.

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

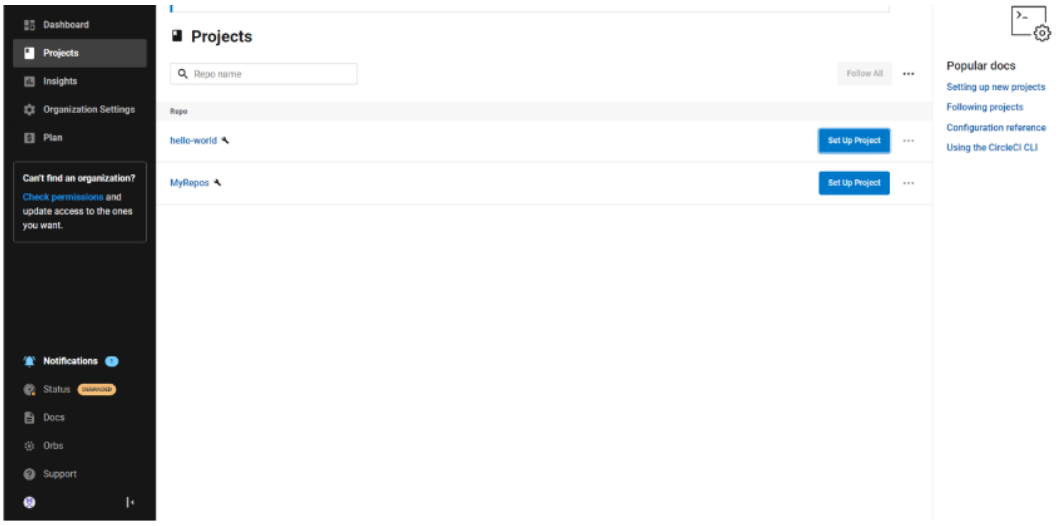
3. Select the option to initialize the repository with a README file.

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.

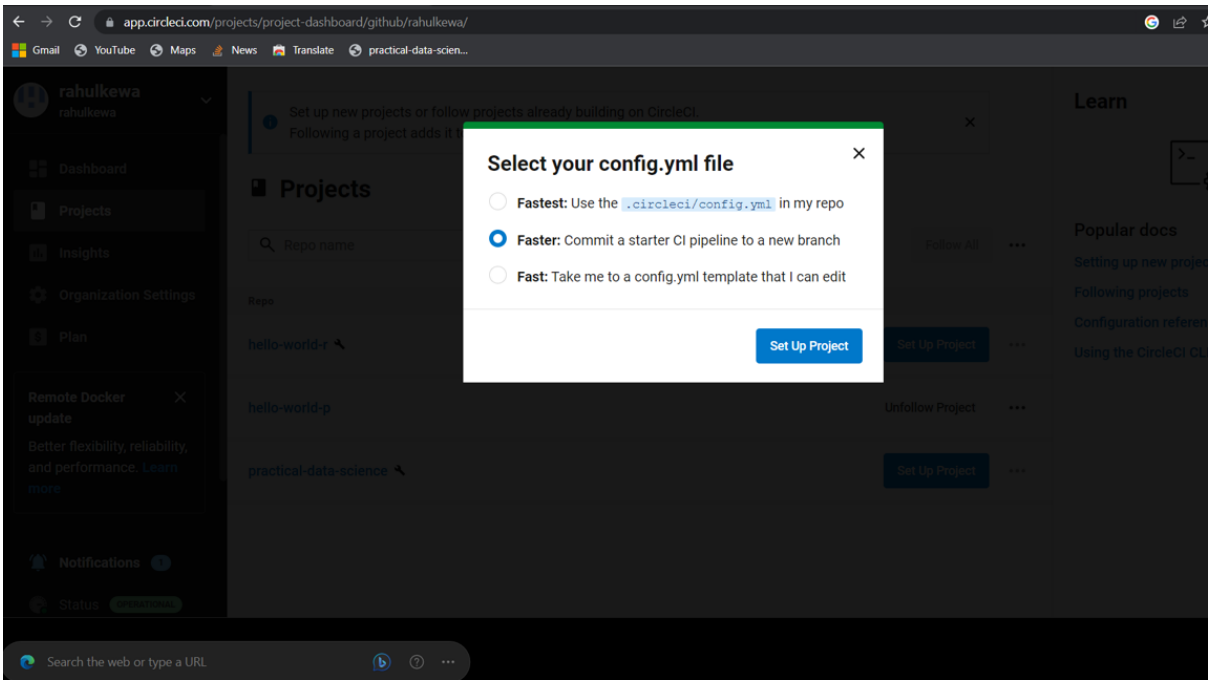


Step 2: Set up CircleCI

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:

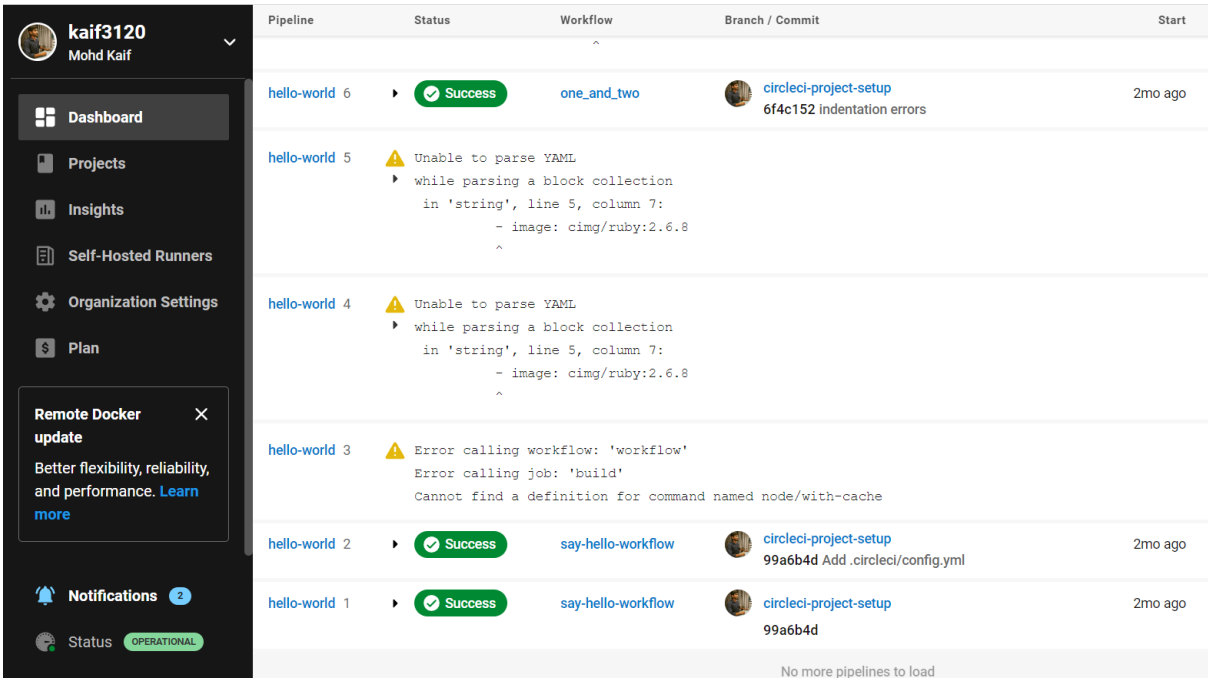
a. Spin up environment

b. Preparing environment variables

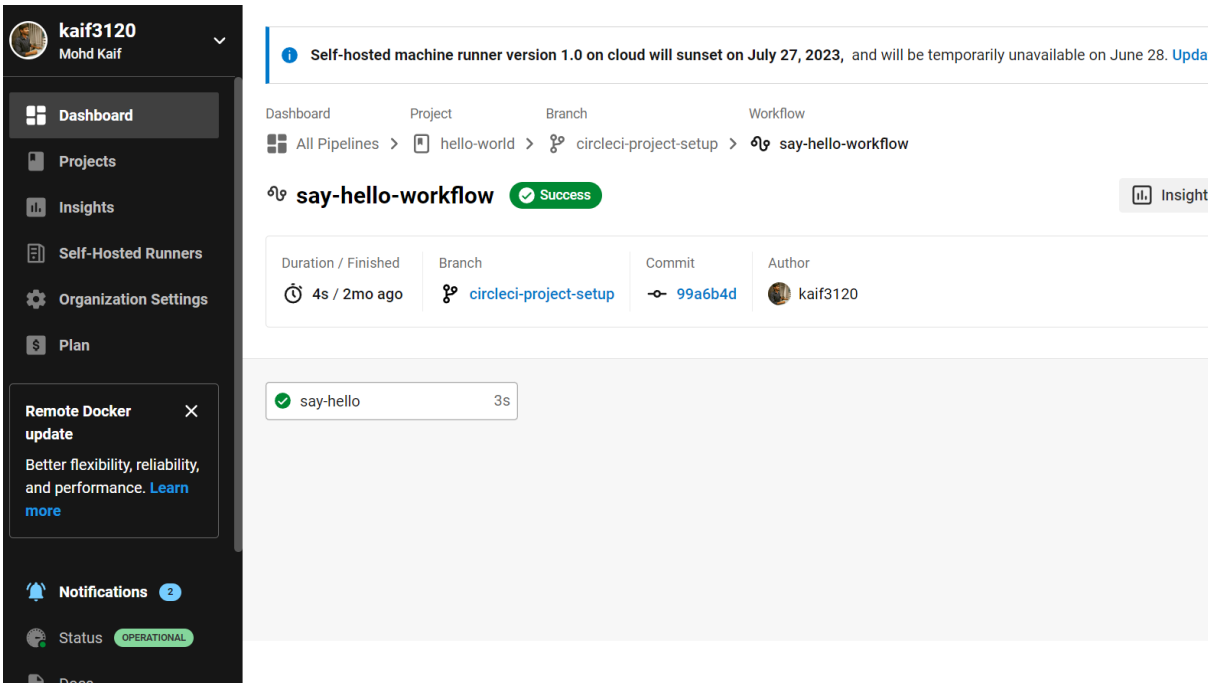
c. Checkout code

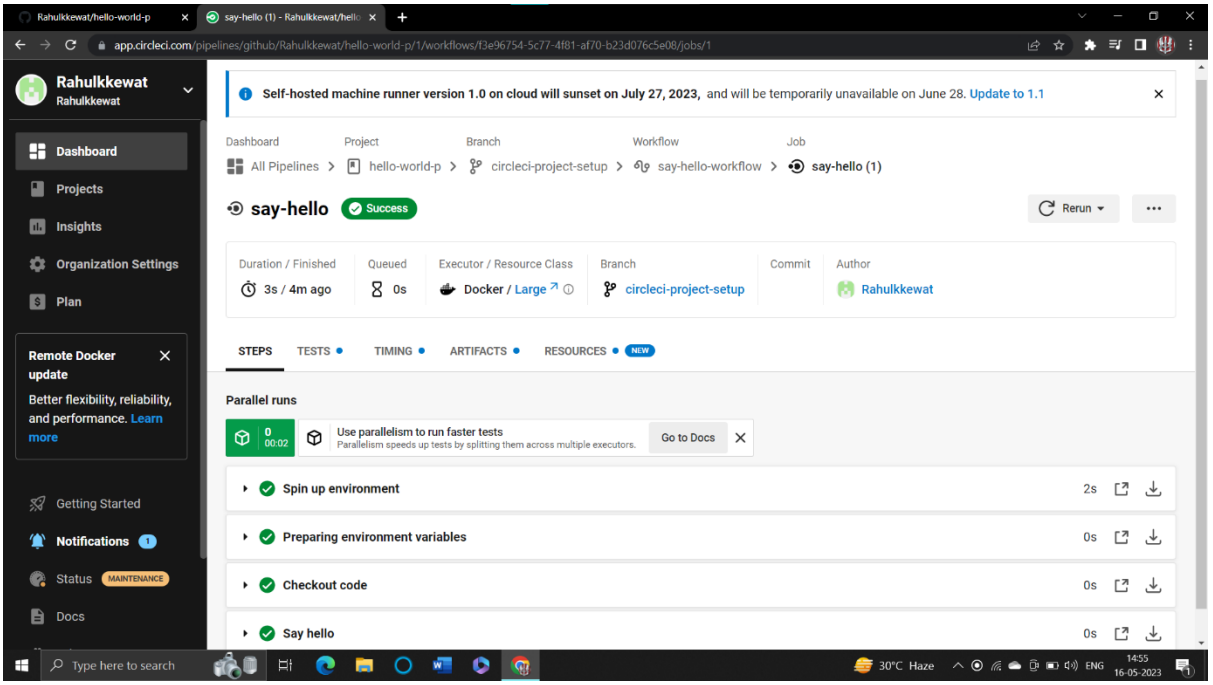
d. Say hello

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

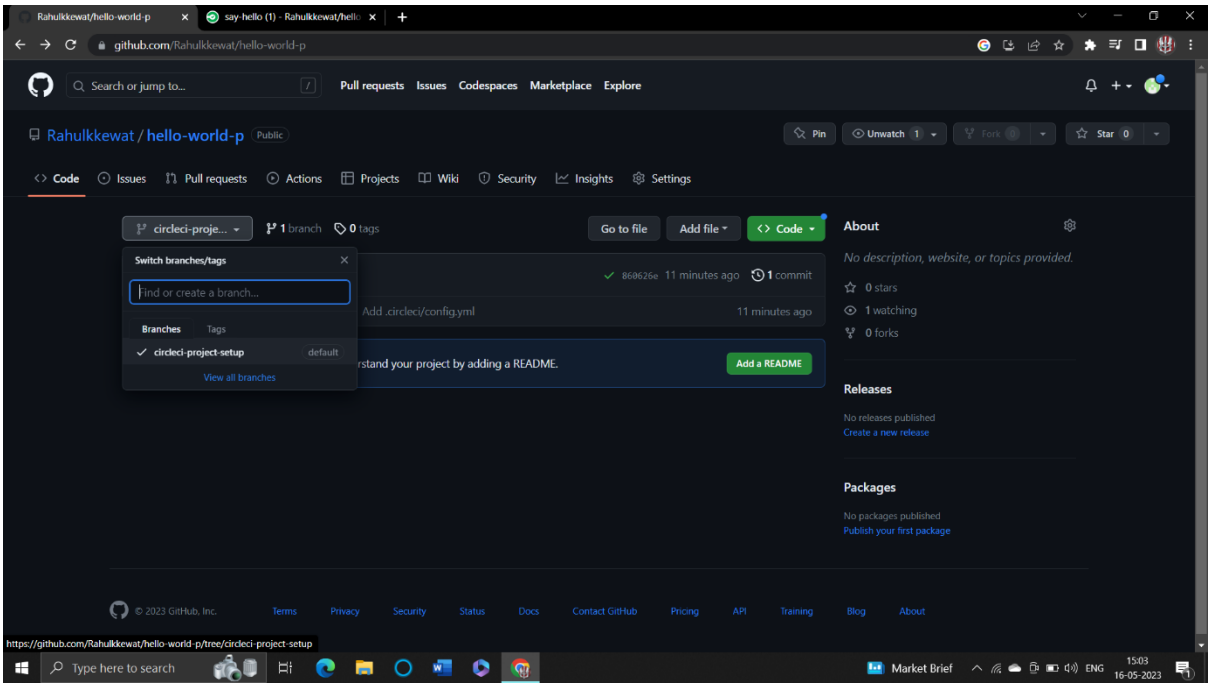


Select “say-hello” Job with a green tick





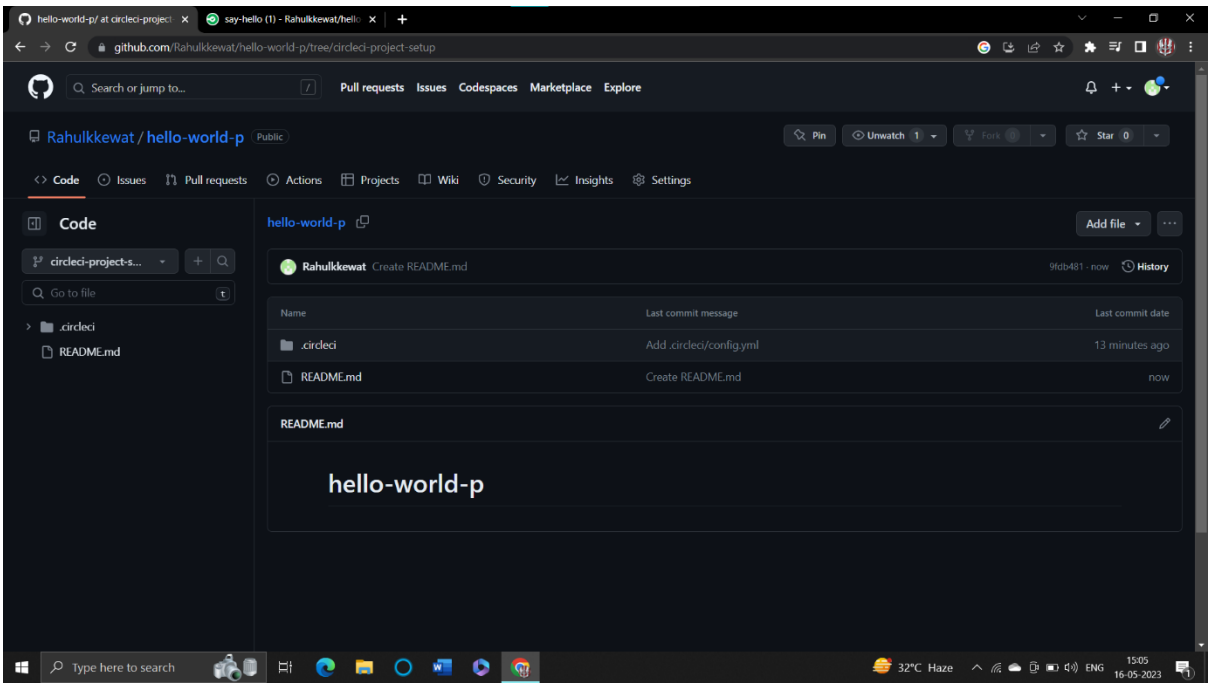
Select Branch and option circleci-project-setup

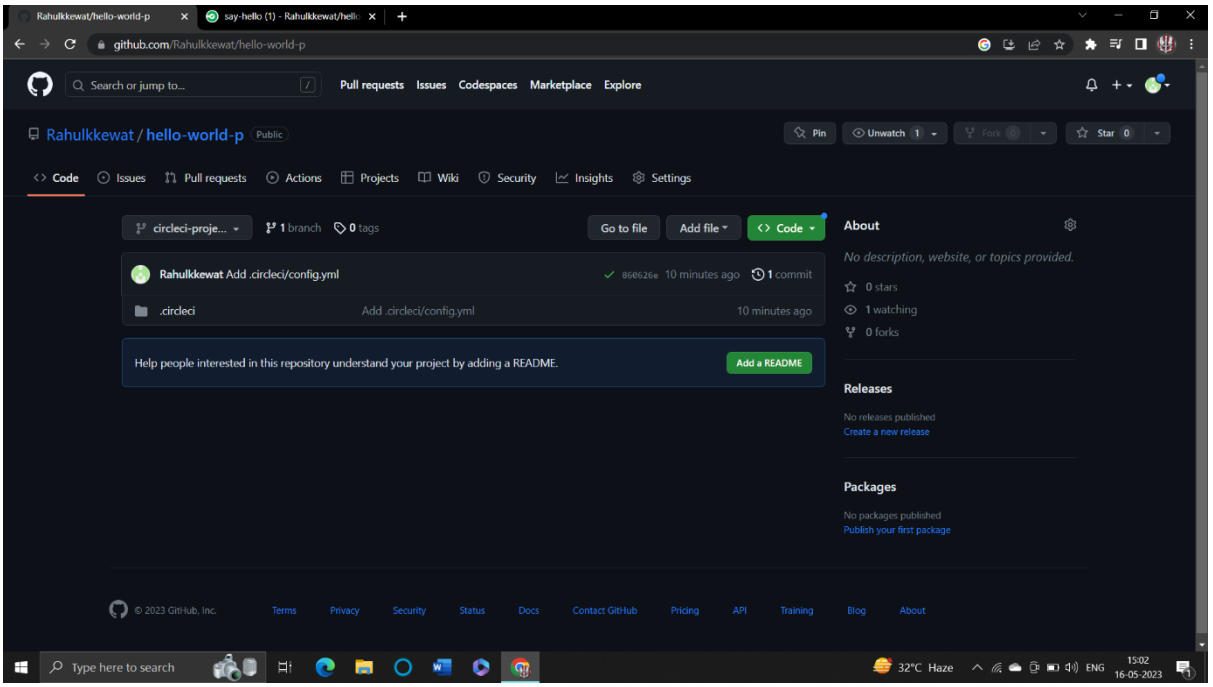


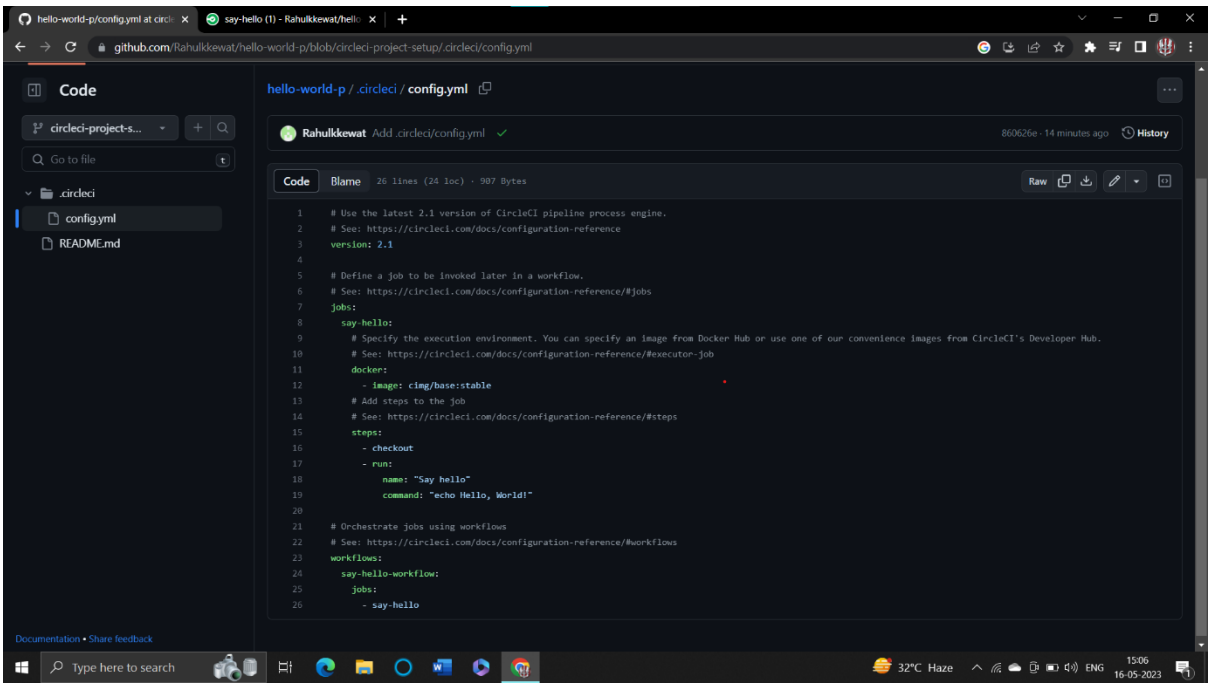
Step 4: Break your build

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

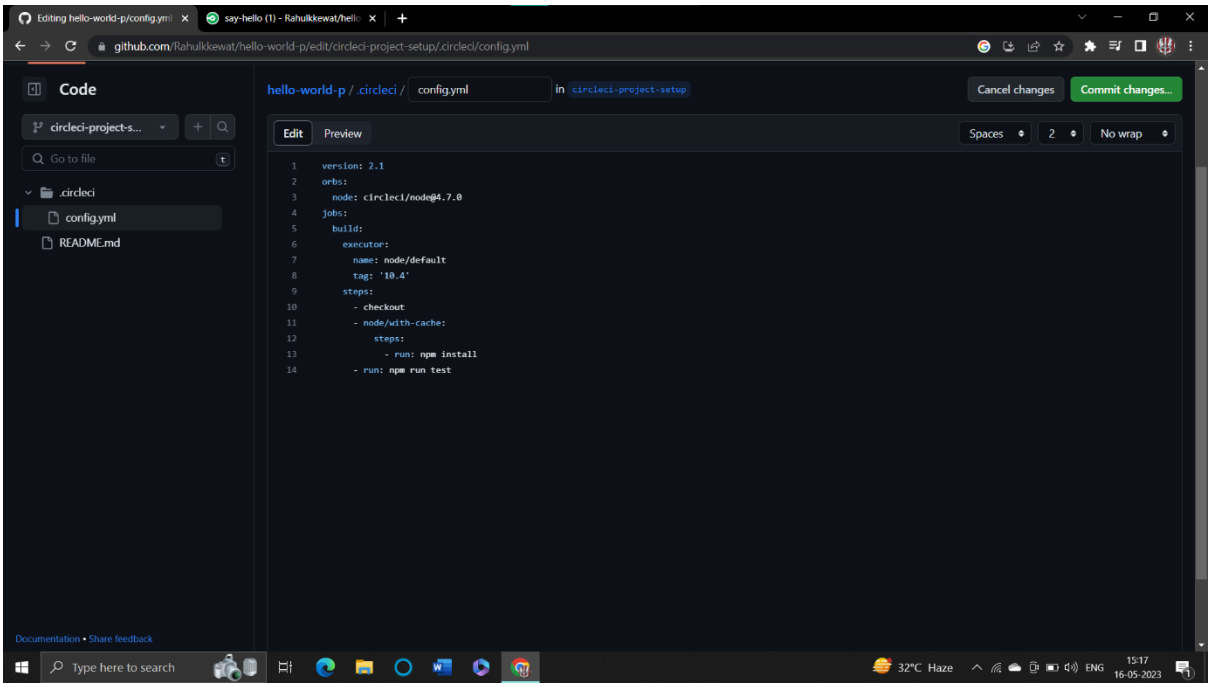
It is possible to edit files directly on GitHub.


hello-world-p/ at circleci-project-setup
say-hello (1) - Rahulkkewat/hello-world-p
github.com/Rahulkkewat/hello-world-p/tree/circleci-project-setup
Search or jump to...
Pull requests
Issues
Codespaces
Marketplace
Explore
Rahulkkewat / hello-world-p
Public
Pin
Unwatch 1
Fork 0
Star 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Code
circleci-project-s...
Go to file
.circleci
README.md
hello-world-p
Add file
Rahulkkewat Create README.md
9fdb481 · now
History
Name
Last commit message
Last commit date
.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

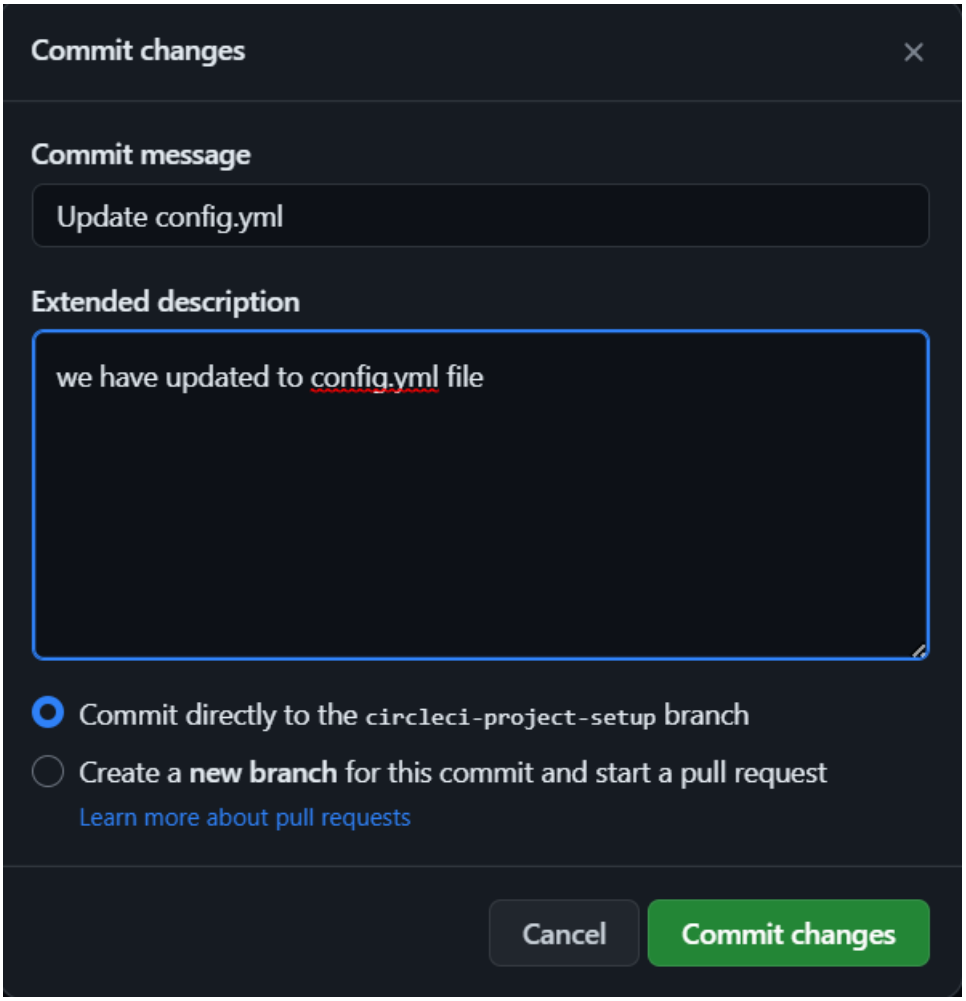

Rahulkkewat/hello-world-p
say-hello (1) - Rahulkkewat/hello-world-p
github.com/Rahulkkewat/hello-world-p
Search or jump to...
Pull requests
Issues
Codespaces
Marketplace
Explore
Rahulkkewat / hello-world-p
Public
Pin
Unwatch 1
Fork 0
Star 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
circleci-proje...
1 branch
0 tags
Go to file
Add file
Code
Rahulkkewat Add .circleci/config.yml
860626e 10 minutes ago
1 commit
.circleci
Add .circleci/config.yml
10 minutes ago
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



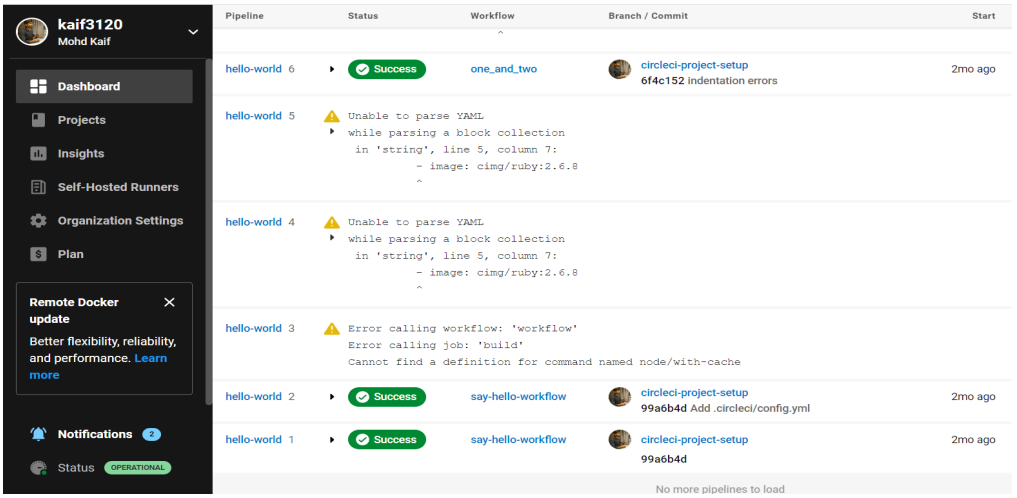
The GitHub file editor should look like this



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.



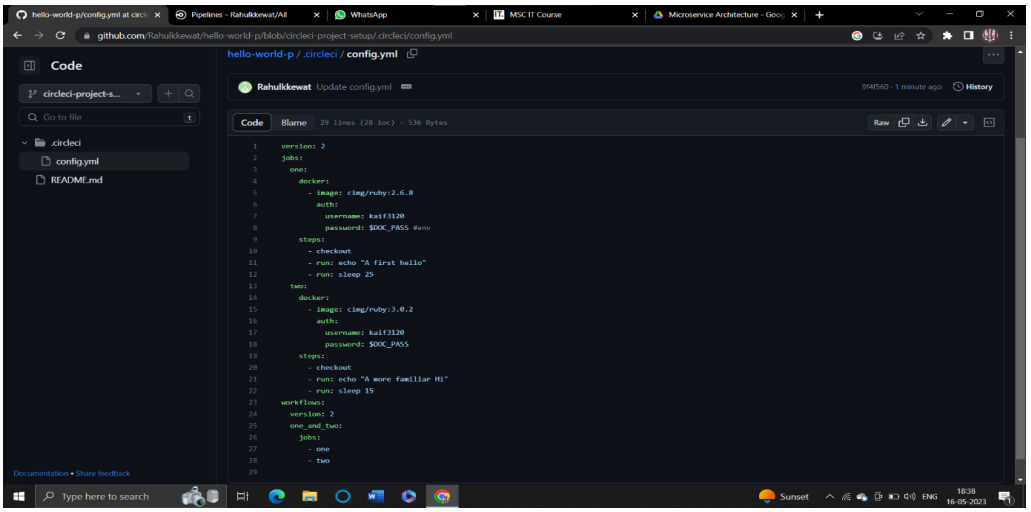
Step 5: Use Workflows

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.

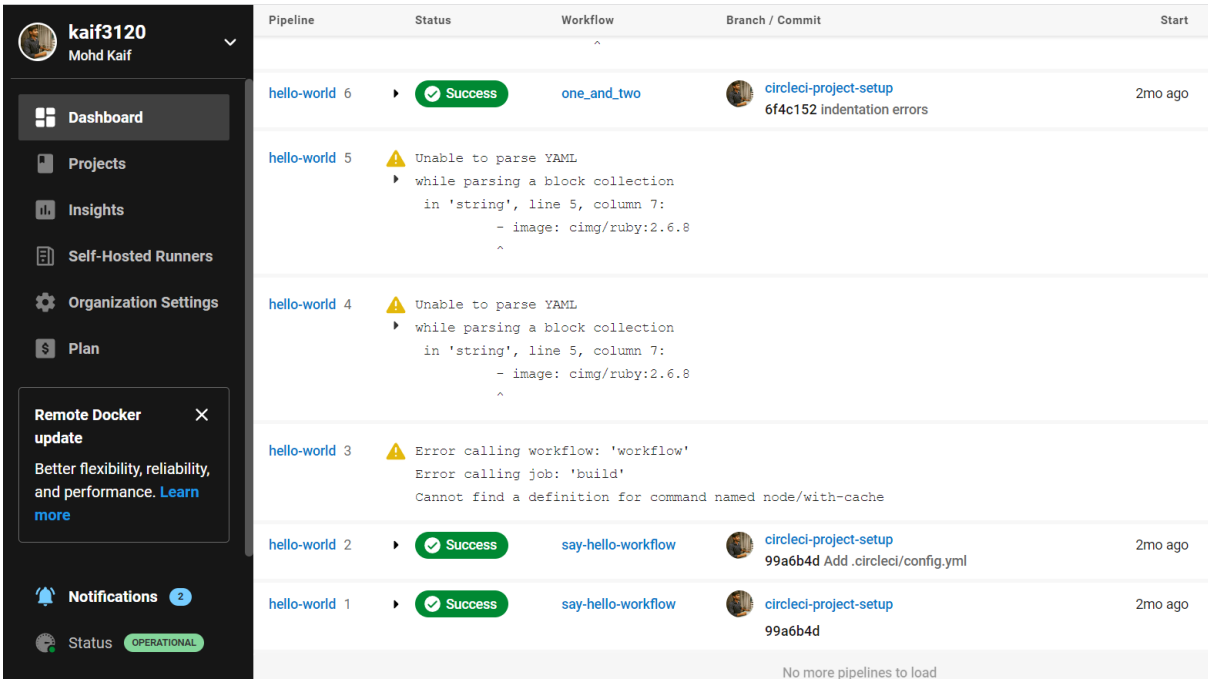
1) **Take a moment and read the comments in the code block below. Then, to see workflows in action, edit your .circleci/config.yml file and copy and paste the following text into it.**

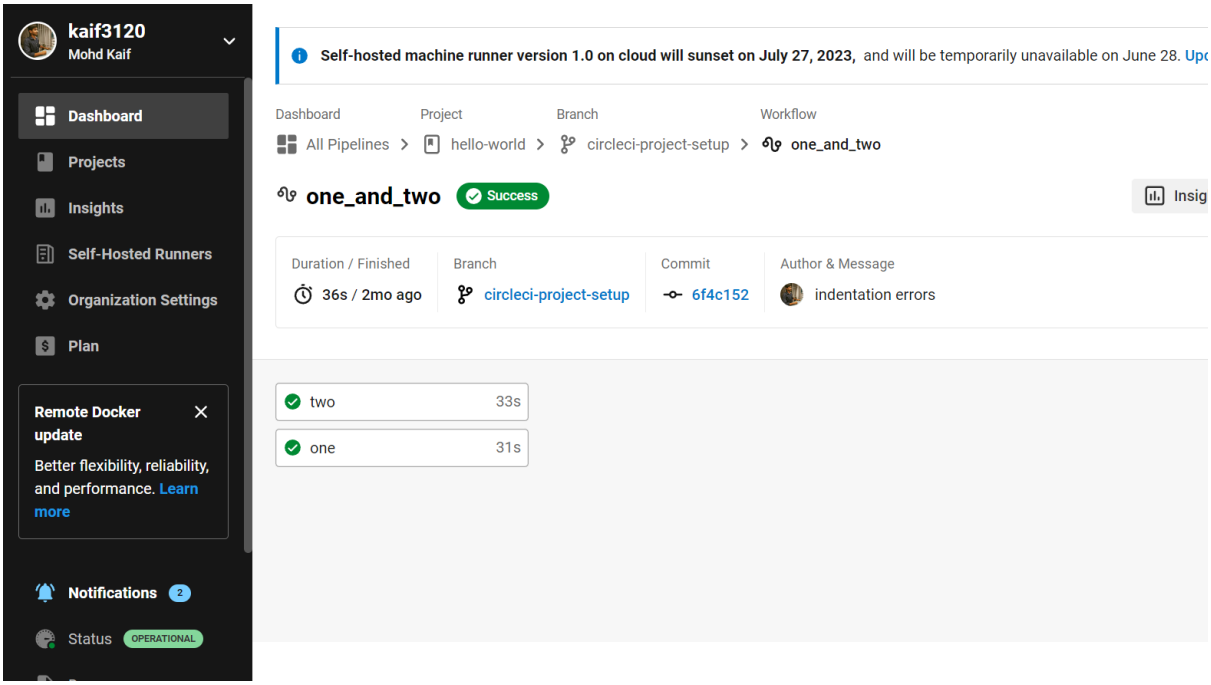
```
version: 2
jobs: # we now have TWO jobs, so that a workflow can coordinate them!
  one: # This is our first job.
    docker: # it uses the docker executor
      - image: cimg/ruby:2.6.8 # specifically, a docker image with ruby 2.6.8
        auth:
          username: mydockerhub-user
          password: $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

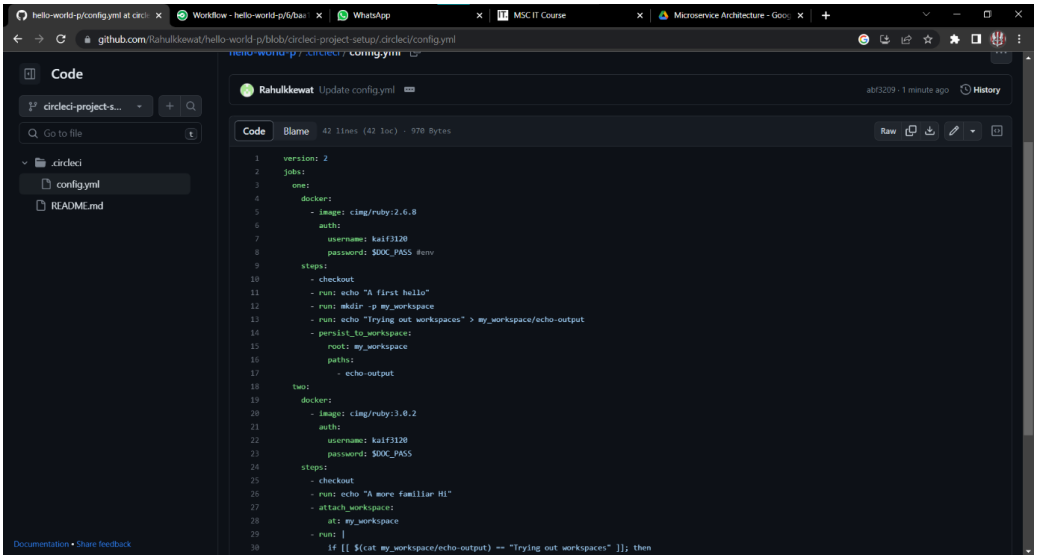
Each workflow has an associated workspace which can be used to transfer files to downstream jobs as the workflow progresses. You can use workspaces to pass along data that is unique to this

```
32
33      - run: |
34          if [[ $(cat my_workspace/echo-output) == "Trying out workspaces" ]]; then
35            echo "It worked!";
36          else
37            echo "Nope!"; exit 1
38          fi
39  workflows:
```

run and which is needed for downstream jobs. Try updating config.yml to the following:

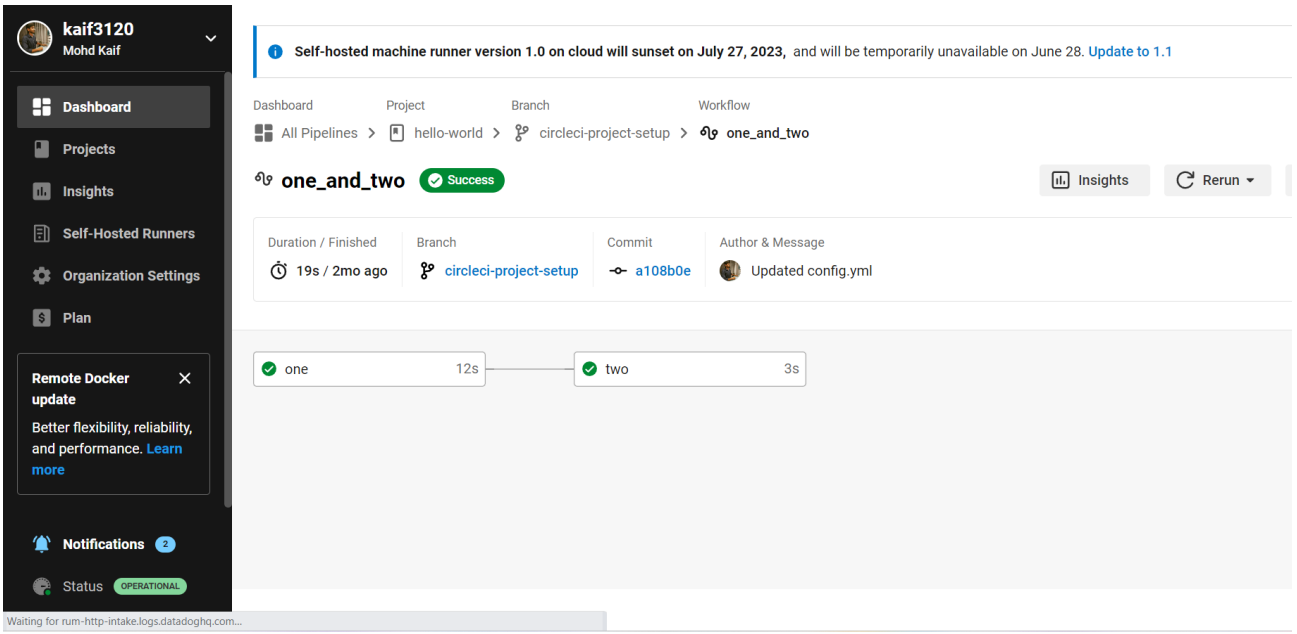
```yaml
version: 2
jobs:
  one:
    docker:
      - image: cimg/ruby:3.0.2
        auth:
          username: mydockerhub-user
          password: $DOCKERHUB_PASSWORD  # context / project UI env-var reference
    steps:
      - checkout
      - run: echo "A first hello"
      - run: mkdir -p my_workspace
      - 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
```

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




```yaml
            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



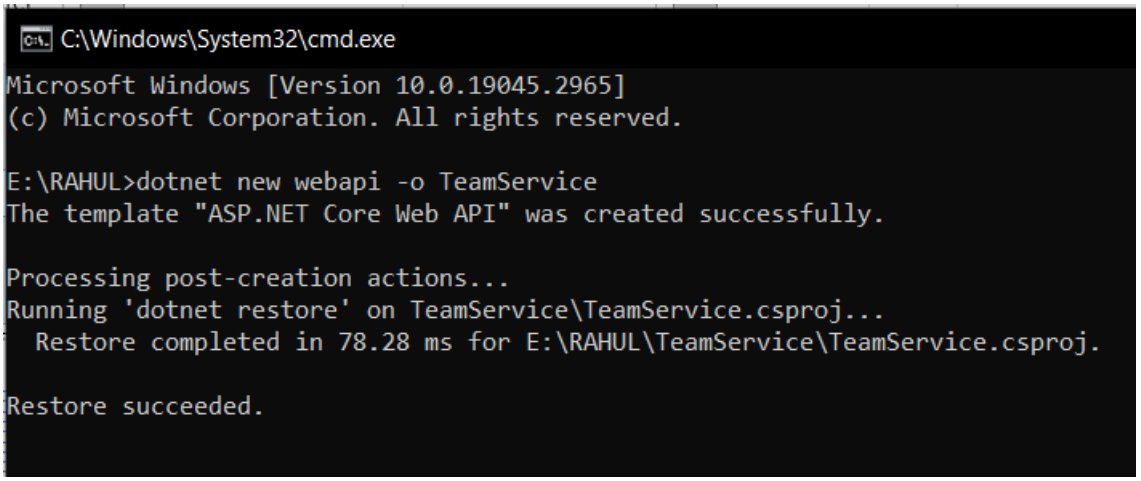
Practical 6

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\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.

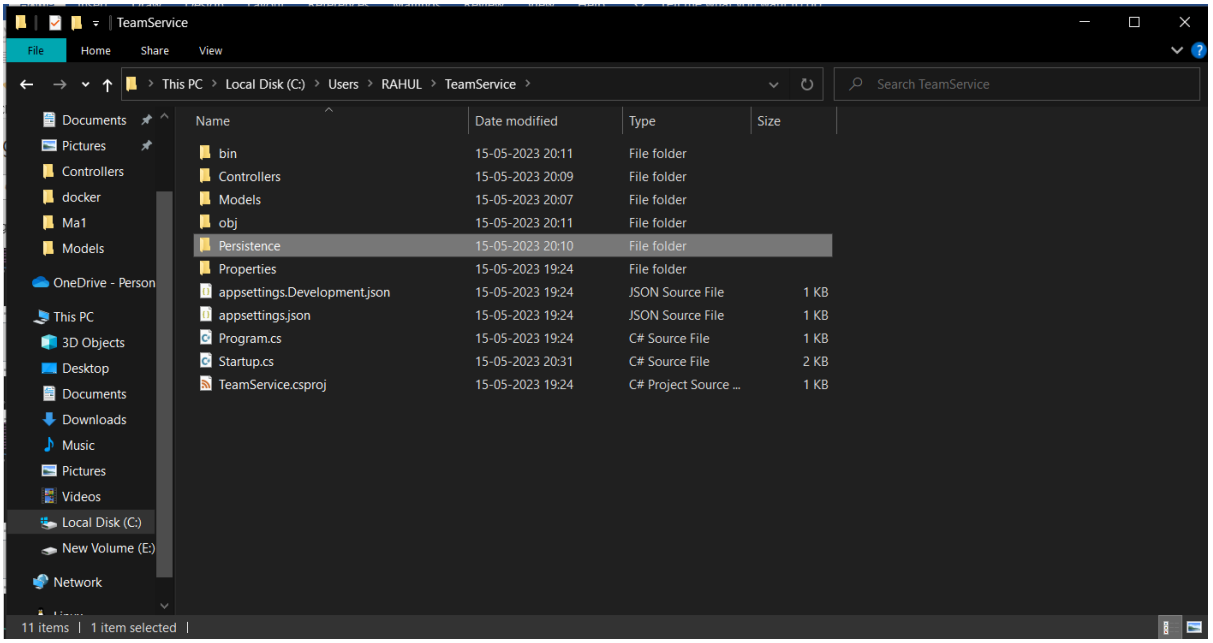
```
using System;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Linq;
using TeamService.Models;
using System.Threading.Tasks;
using TeamService.Persistence;

namespace TeamService

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

```
        {
            repository = repo;
        }

        [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\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

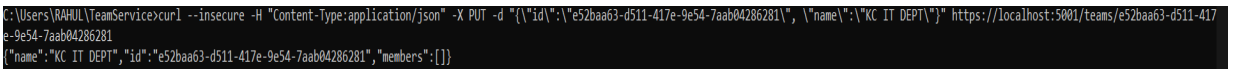
curl --insecure https://localhost:5001/teams/e52baa63-d511-417e-9e54-7aab04286281

```
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")

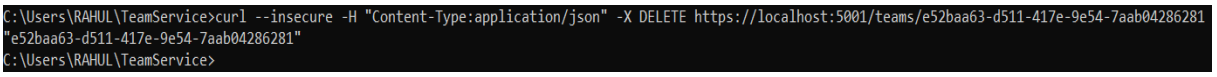
curl --insecure -H "Content-Type:application/json" –X PUT –d "{\"id\":\"e52baa63-d511-417e-9e54-7aab04286281\", \"name\":\"KC IT DEPT\"}" https://localhost:5001/teams/e52baa63-d511-417e-9e54-7aab04286281



Step 16: On command prompt 2

Command: To delete team

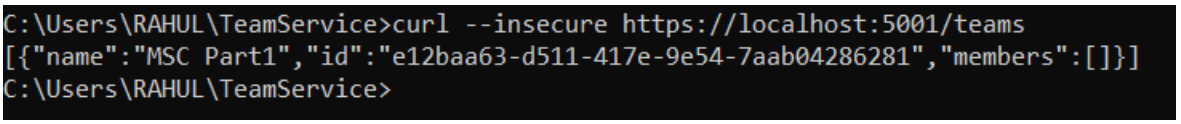
curl --insecure -H "Content-Type:application/json" –X DELETE https://localhost:5001/teams/e52baa63-d511-417e-9e54-7aab04286281



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



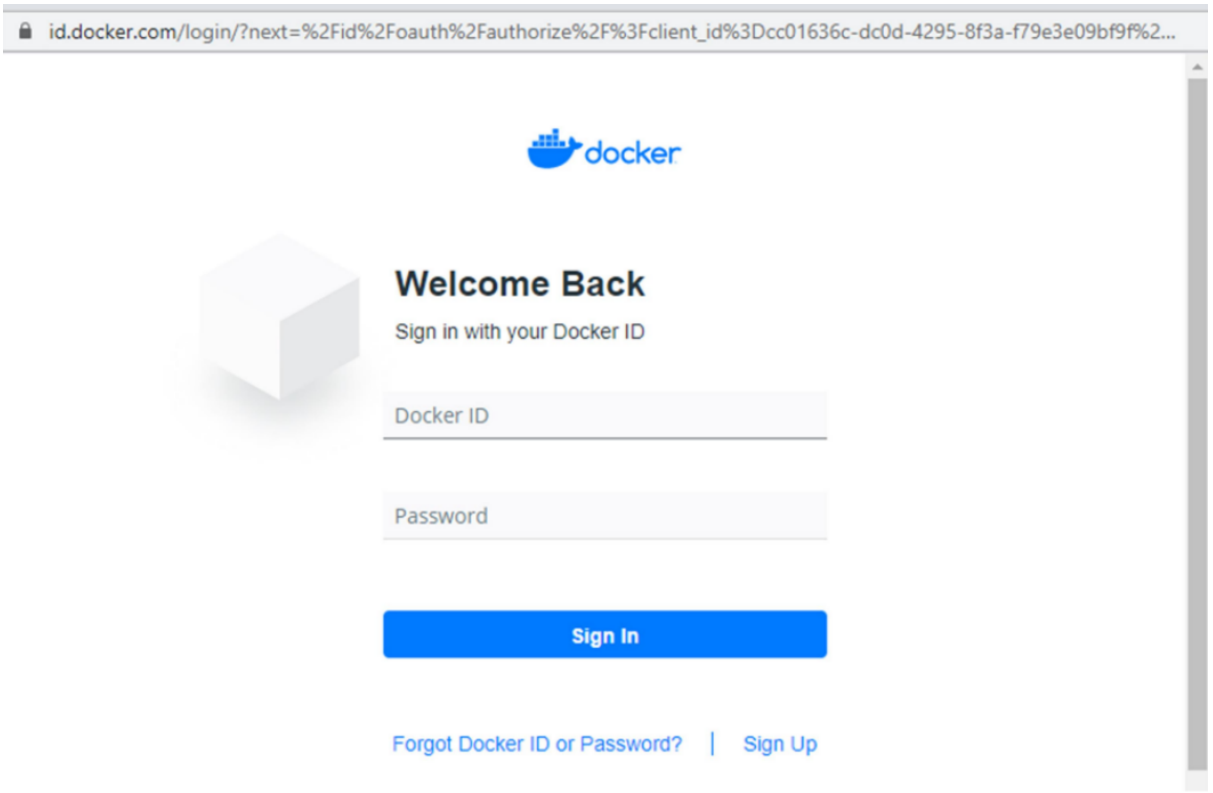
Practical 7

Aim: Running Location Service in Docker

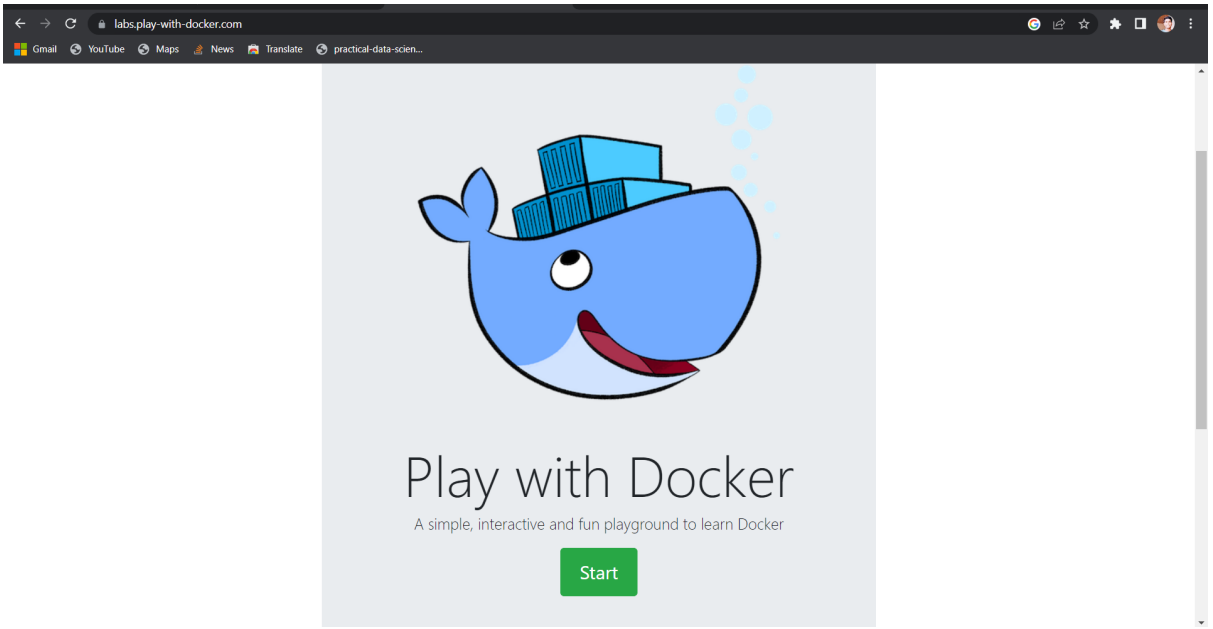
Writeup:

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

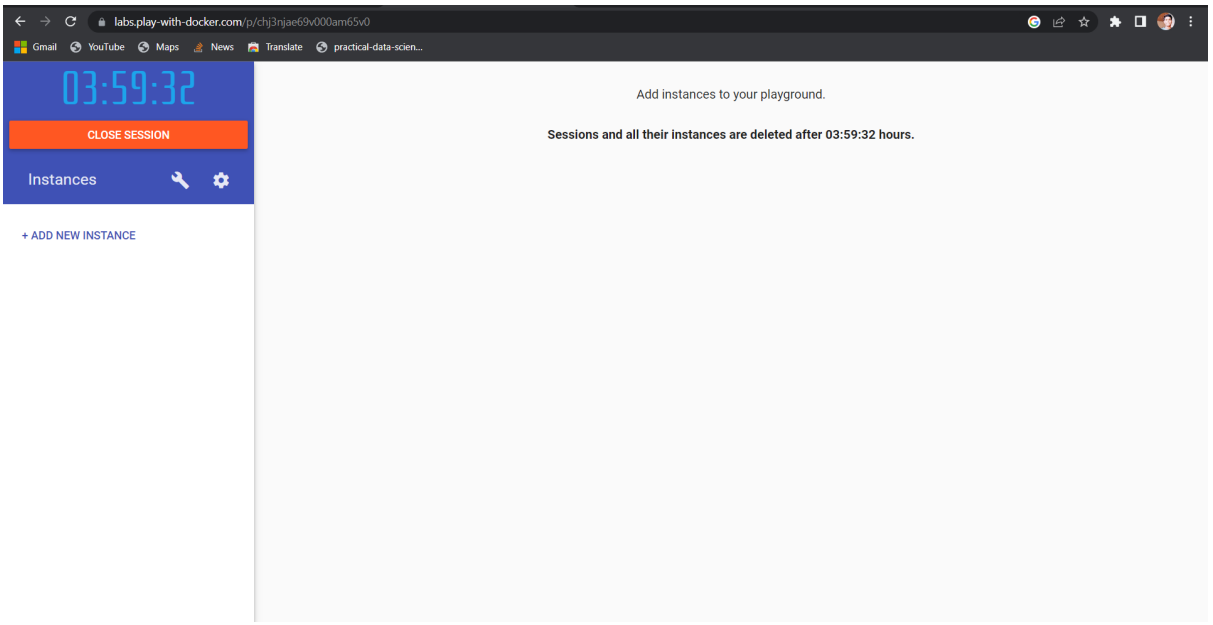
Now login in to Play-With-Docker

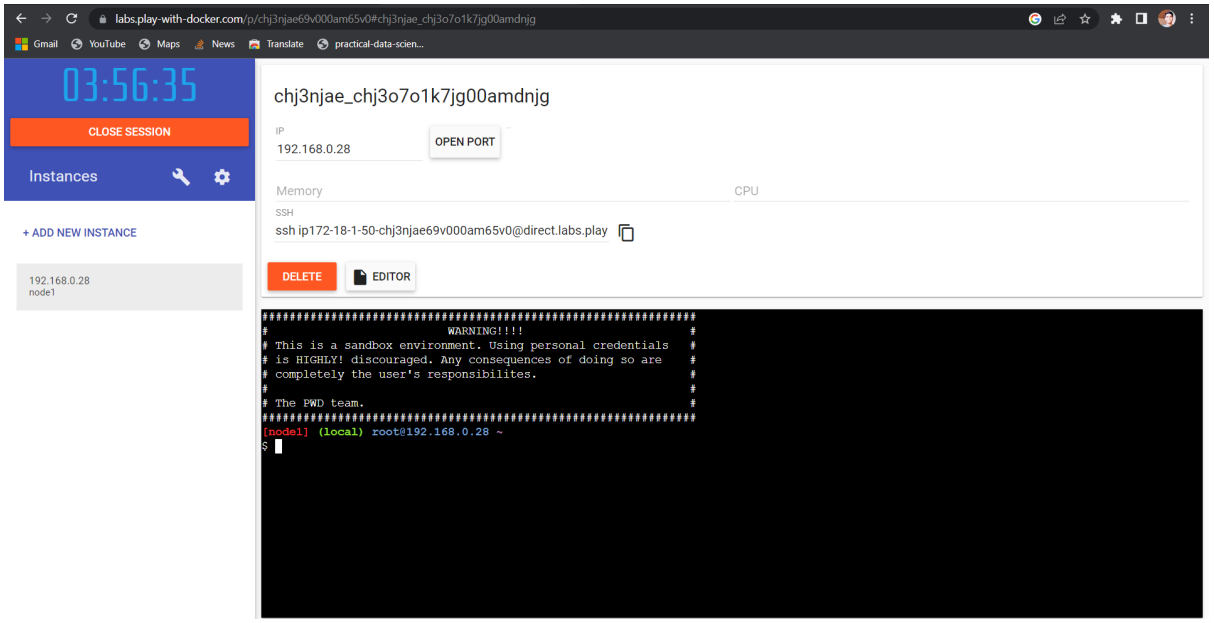


Step 2: Click on Start



Step 3: Click on Add New Instance





Step 3: Start typing following commands

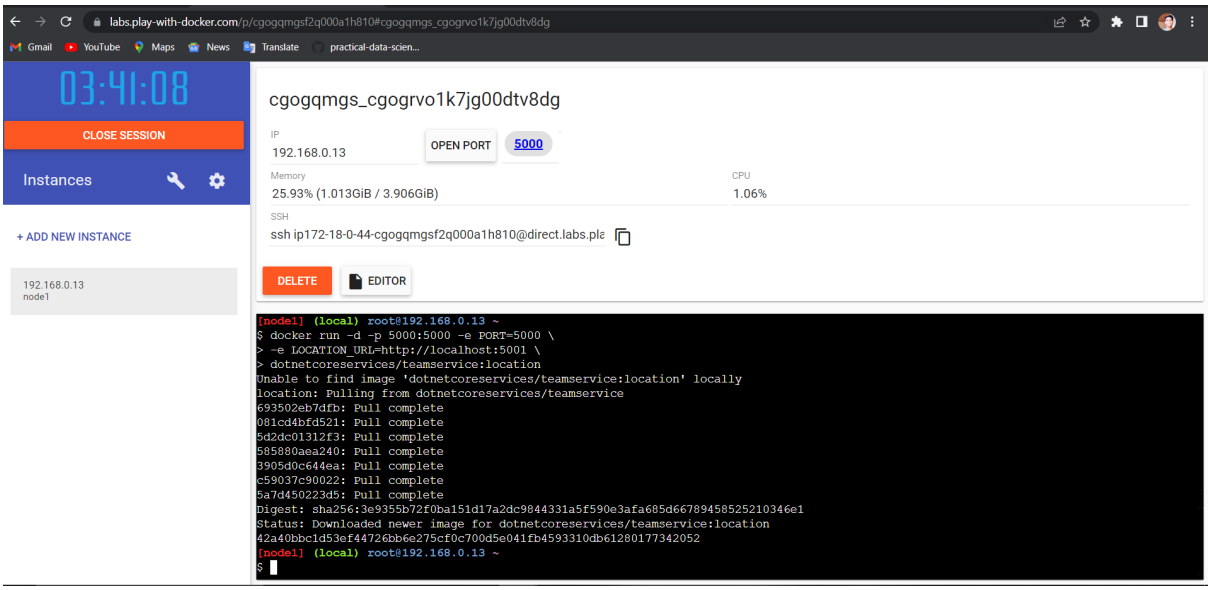
Command : To run teamservice

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

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

dotnetcoreservices/teamservice:location

Output:



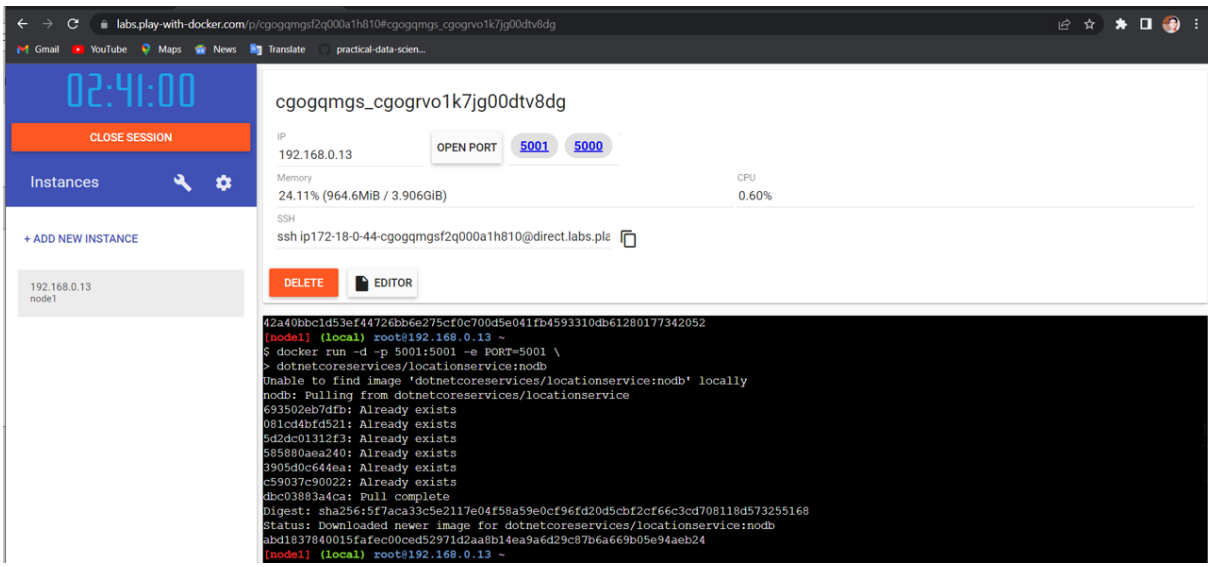
Step 4:

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)

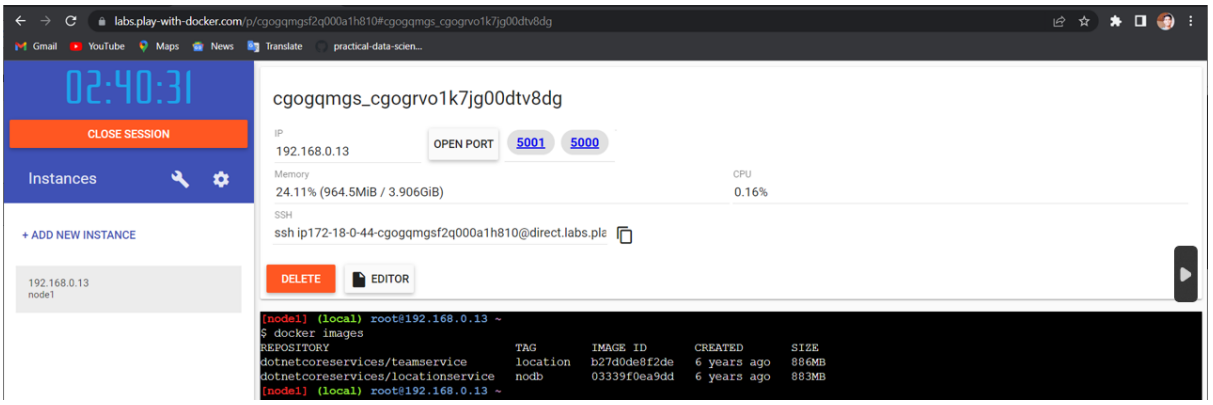


Step 4:

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:

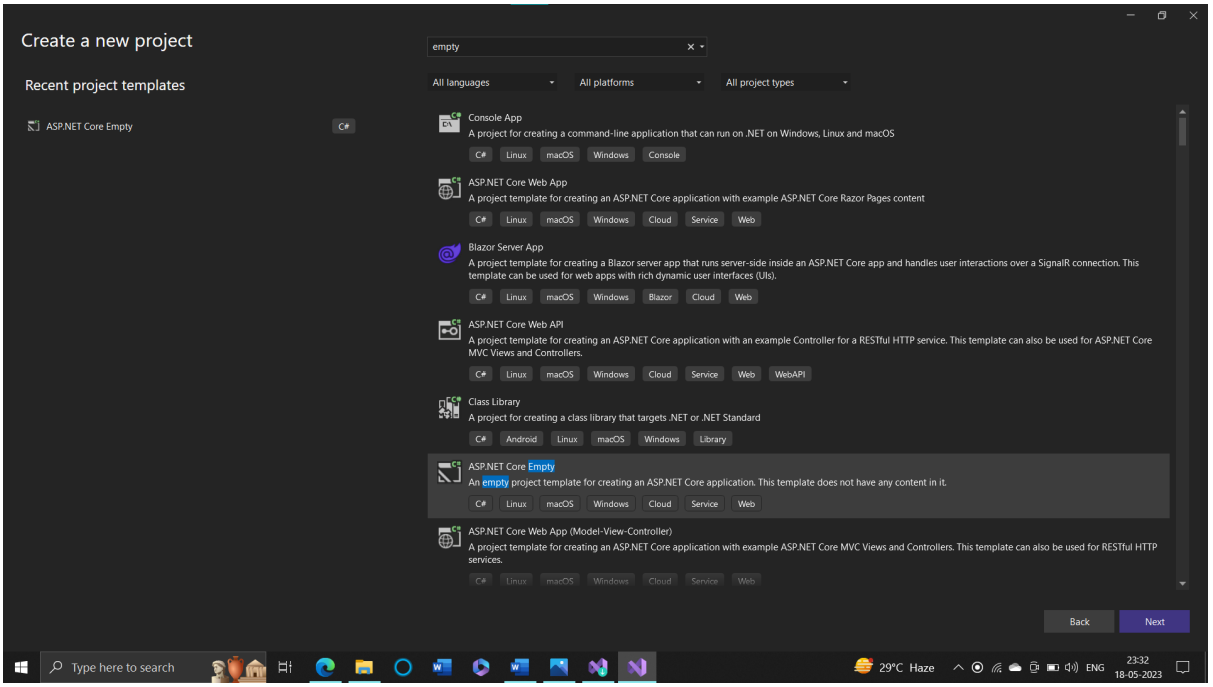
```
$ curl http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292
[{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f","latitude":12.0,"longitude":12.0,"altitude":10.0,"timestamp":0,"memberID":"63e7acf8-8fae-
42ce-9349-3c8593ac8292"}][node1] (local) root@192.168.0.13 ~
$
```

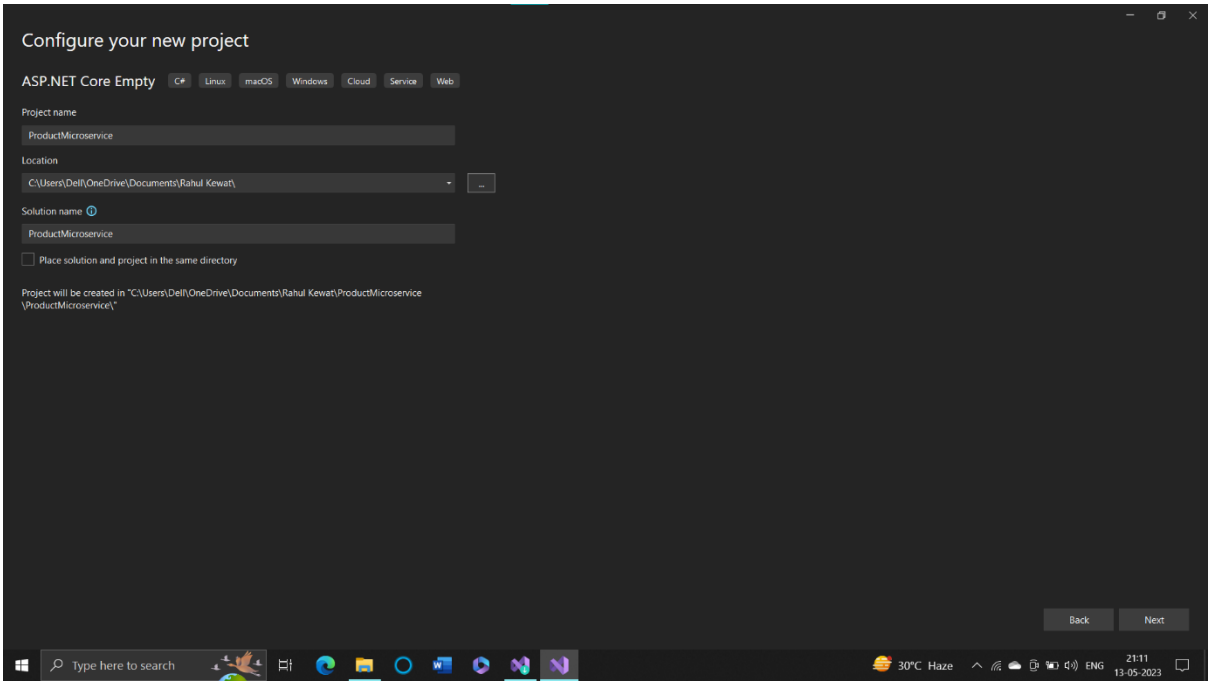
Practical 8

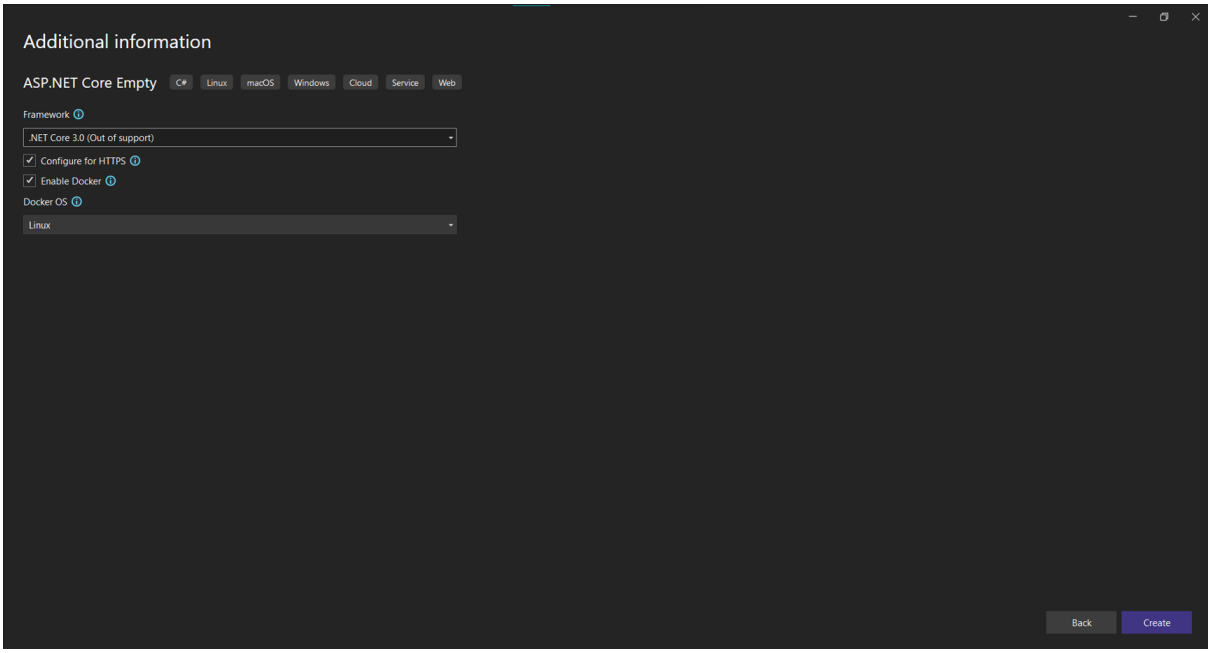
Aim: Building real-time Microservice with ASP.NET Core.

Writeup:

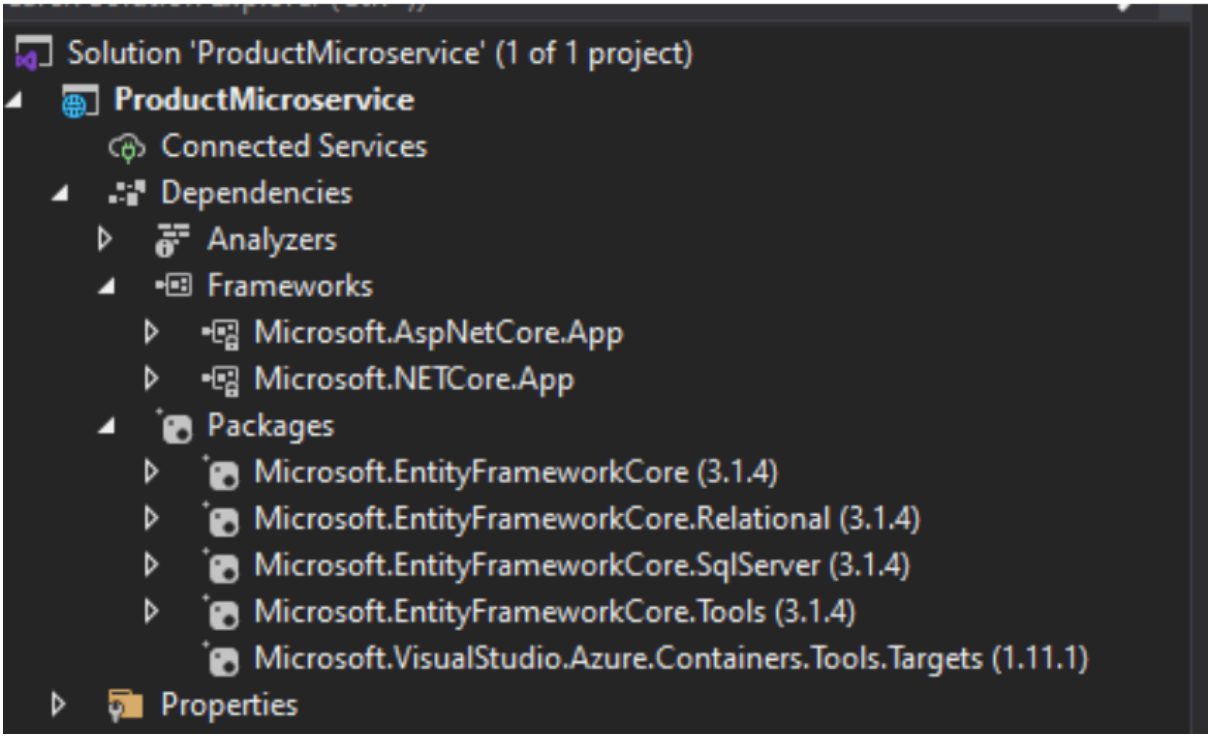
Step 1: Create and configure a new empty project


Create a new project
Recent project templates
ASP.NET Core Empty
C#
empty
All languages
All platforms
All project types
Console App
A project for creating a command-line application that can run on .NET on Windows, Linux and macOS
C# Linux macOS Windows Console
ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Core Razor Pages content
C# Linux macOS Windows Cloud Service Web
Blazor Server App
A project template for creating a Blazor server app that runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Blazor Cloud Web
ASP.NET Core Web API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.
C# Linux macOS Windows Cloud Service Web WebAPI
Class Library
A project for creating a class library that targets .NET or .NET Standard
C# Android Linux macOS Windows Library
ASP.NET Core Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.
C# Linux macOS Windows Cloud Service Web
ASP.NET Core Web App (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.
Back
Next
Type here to search
29°C Haze
ENG
23:32
18-05-2023

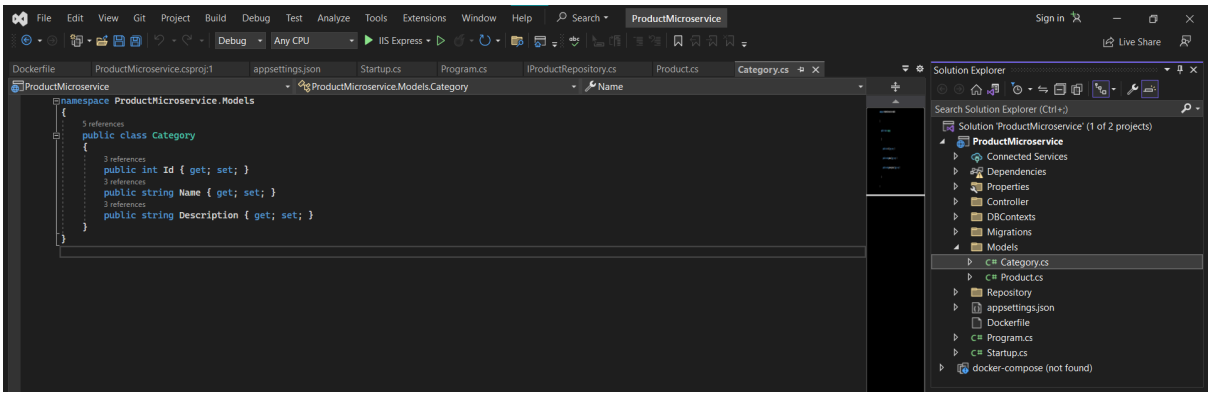

Configure your new project
ASP.NET Core Empty C# Linux macOS Windows Cloud Service Web
Project name
ProductMicroservice
Location
C:\Users\Dell\OneDrive\Documents\Rahul Kewat\
Solution name
ProductMicroservice
Place solution and project in the same directory
Project will be created in "C:\Users\Dell\OneDrive\Documents\Rahul Kewat\ProductMicroservice\ProductMicroservice\"
Back
Next
Type here to search
30°C Haze
ENG
21:11
13-05-2023



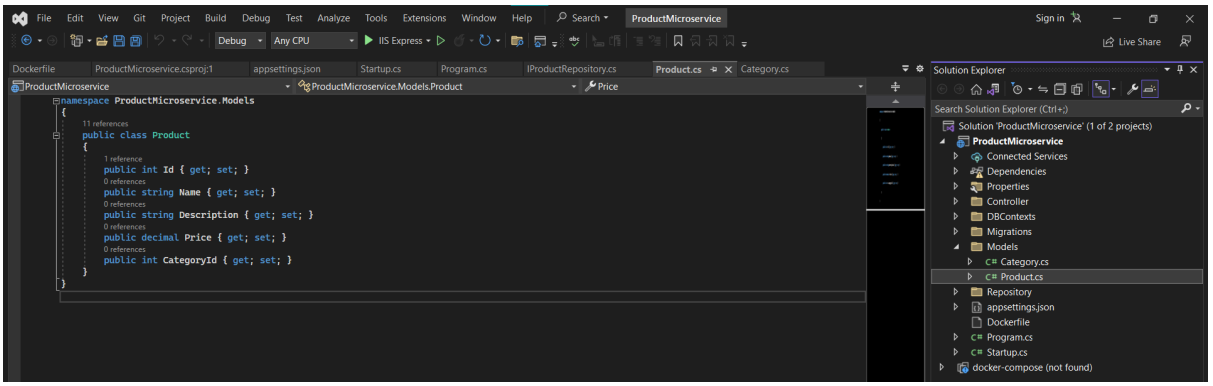
Step 2: Make sure to add all these packages via nugget package manager



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

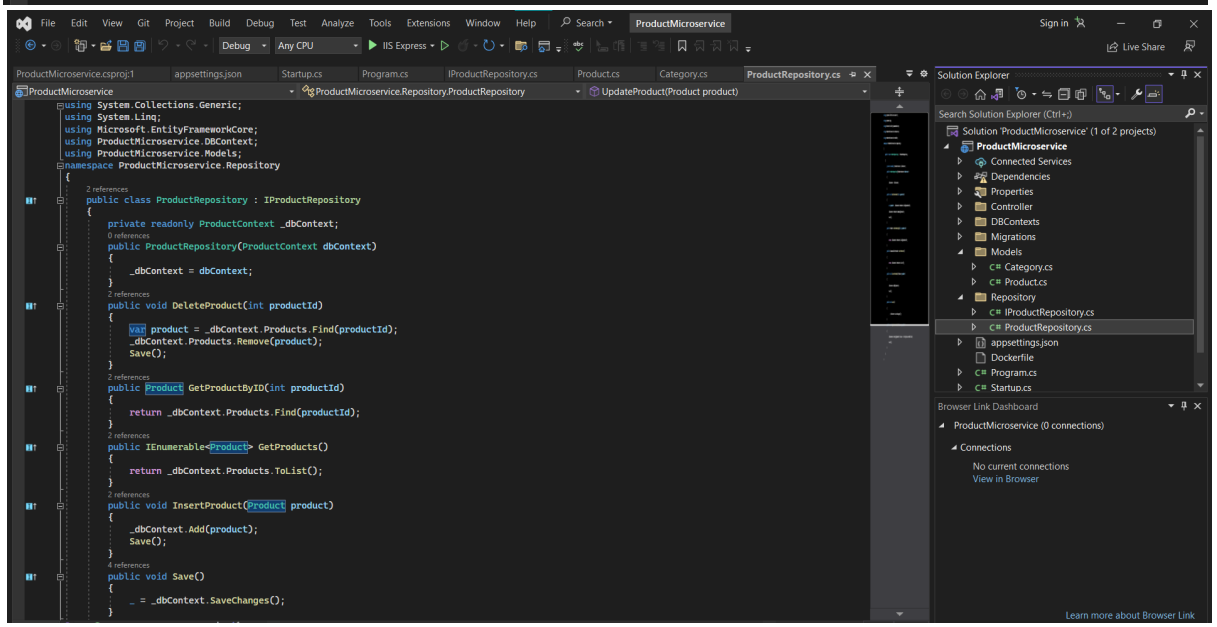


Step 4: Create the Product Model which will eventually be the Product Table

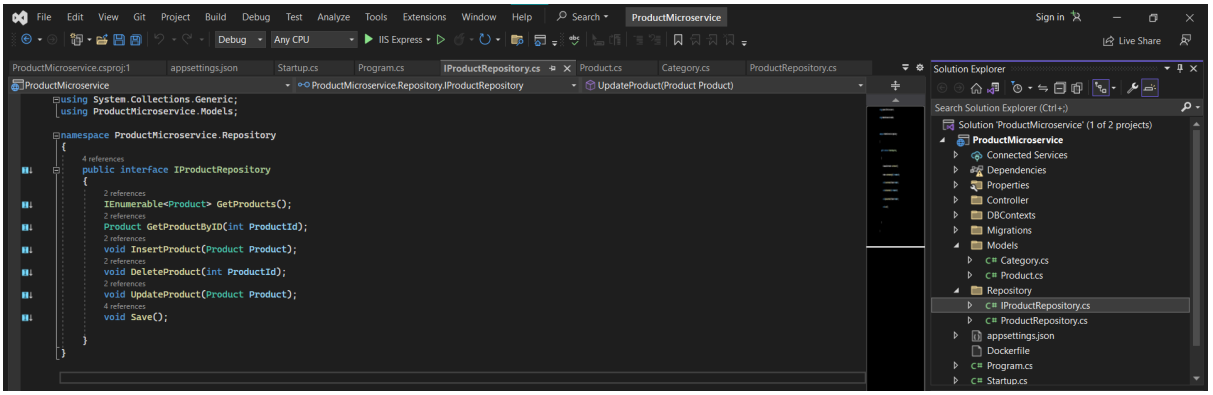


Step 5: Create a ProductRepository for all our operations

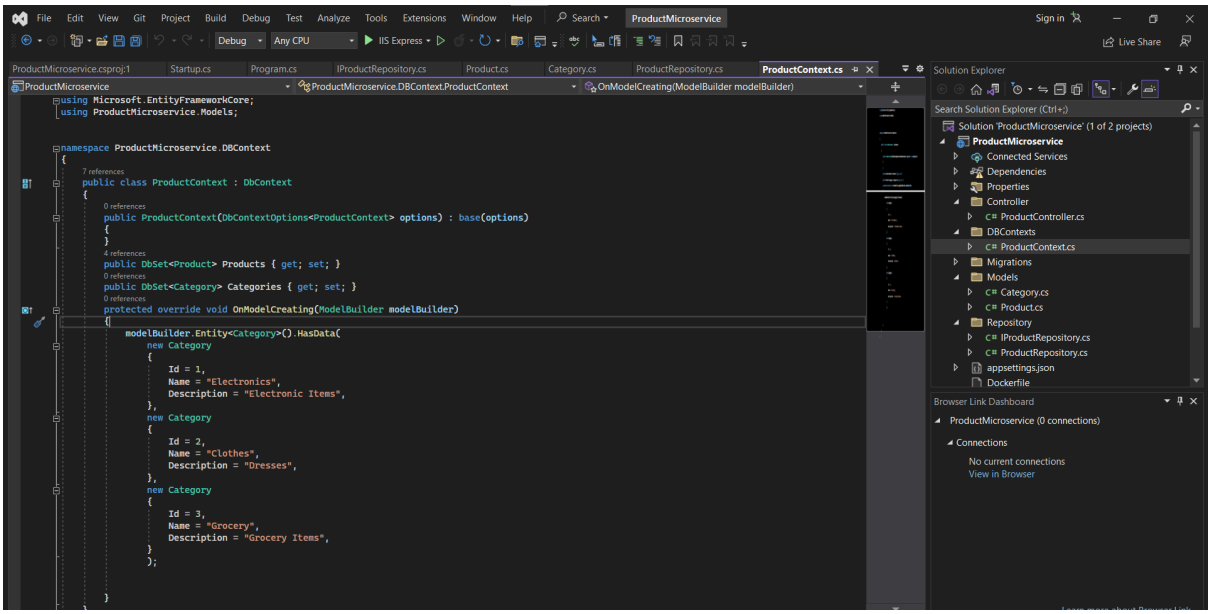
```
        }
        2 references
        public void UpdateProduct(Product product)
        {
            _dbContext.Entry(product).State = EntityState.Modified;
            Save();
        }
    }
}
```



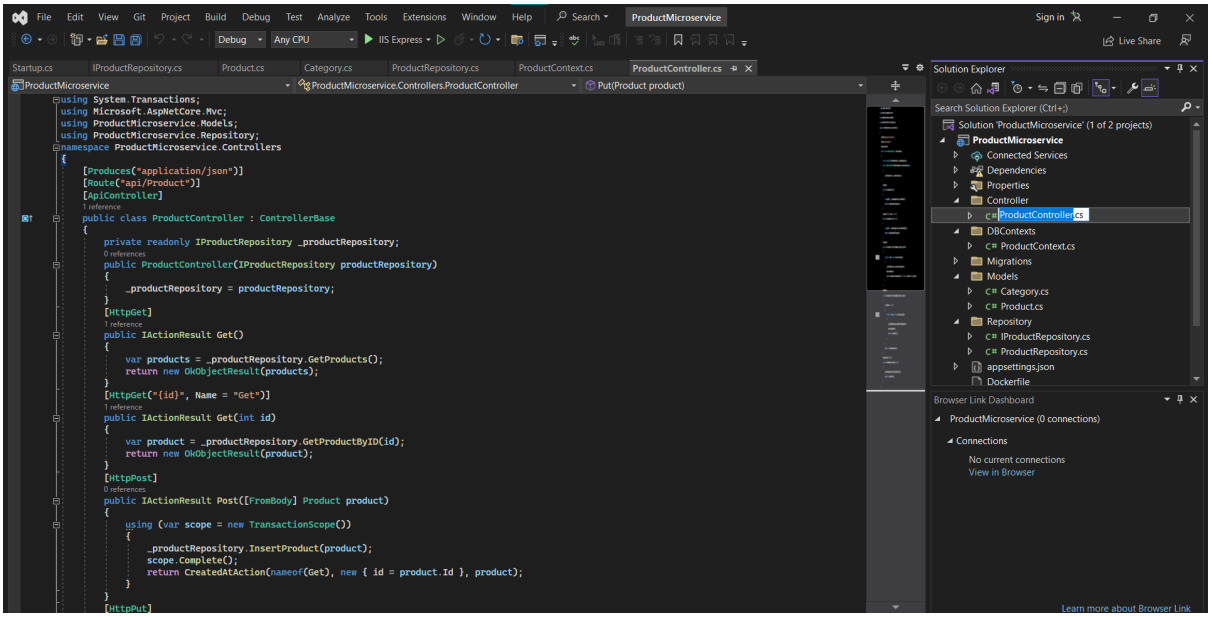
Step 6: Create an interface IProductRepository to access the ProductRepository

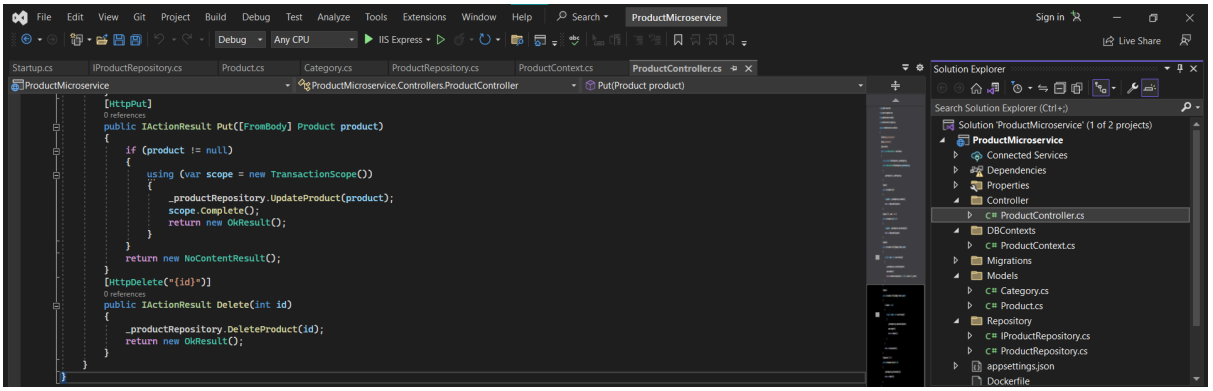


Step 7: Create a ProductContext to build the model for the database; this will be a Code First Database Approach

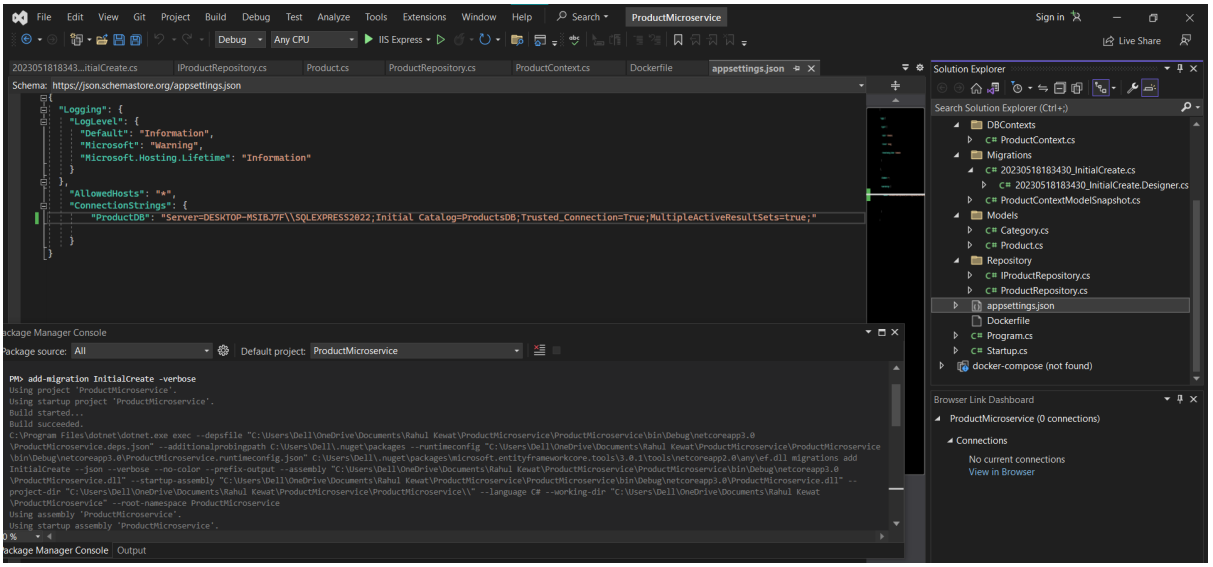


Step 8: Add a ProductController to get an endpoint for the APIs

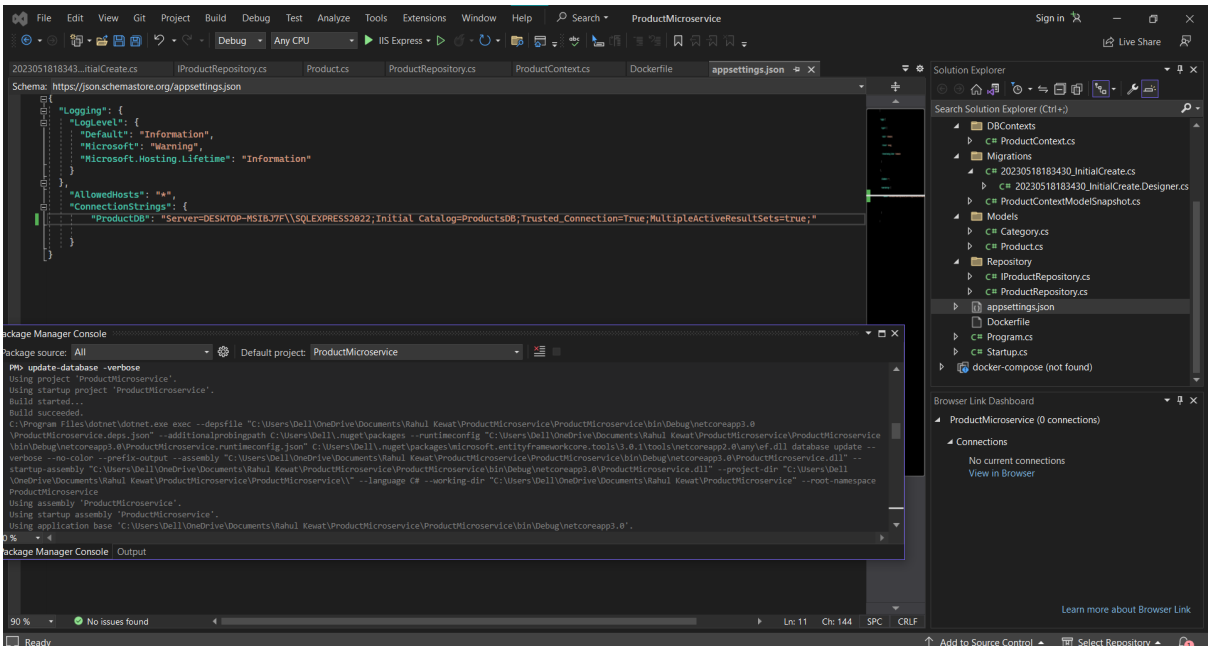




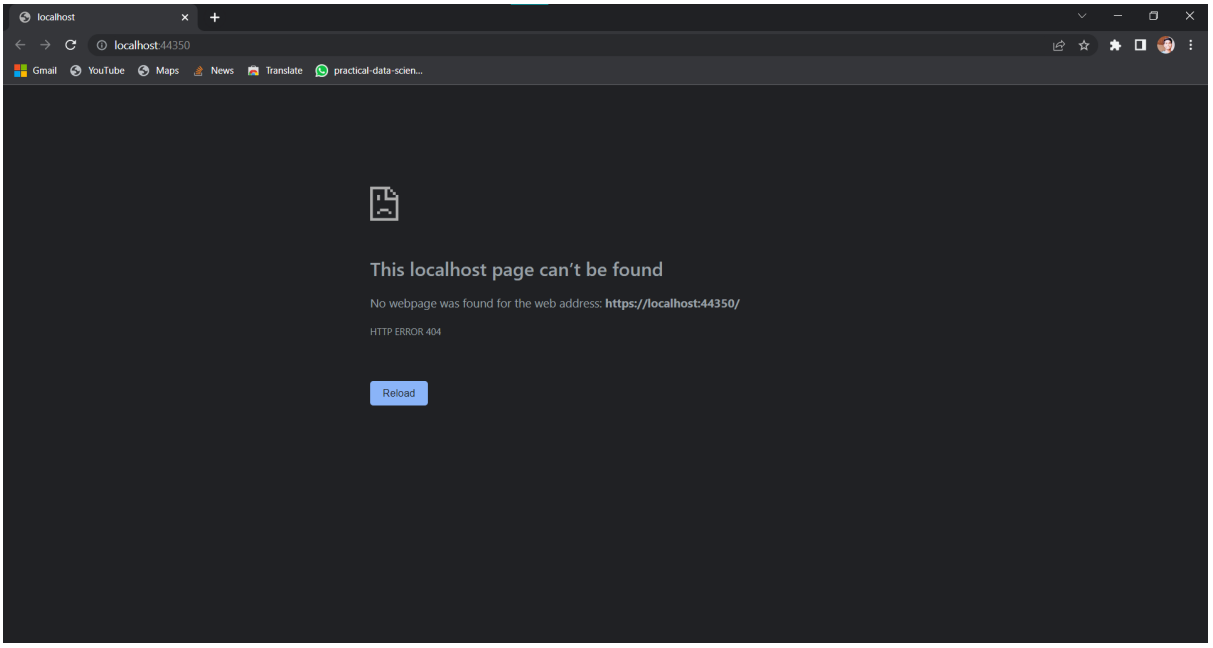
Step 9: Add the connection string settings to connect the project to the SQLSERVER database. Once added run add-migration command to run the migration



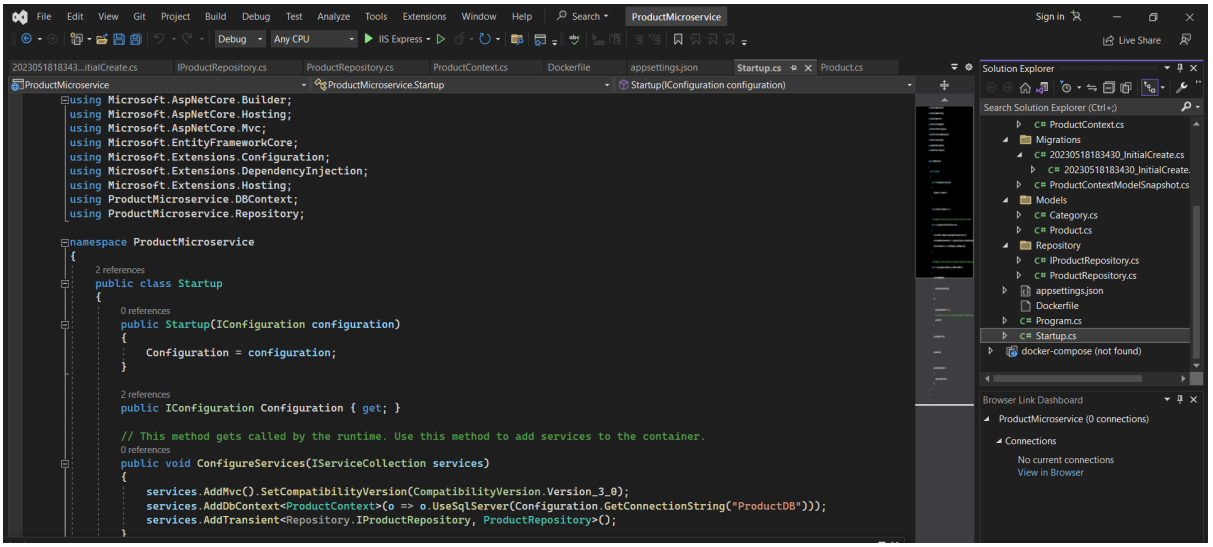
Step 10: Run the update-database command to reflect the model changes to the database



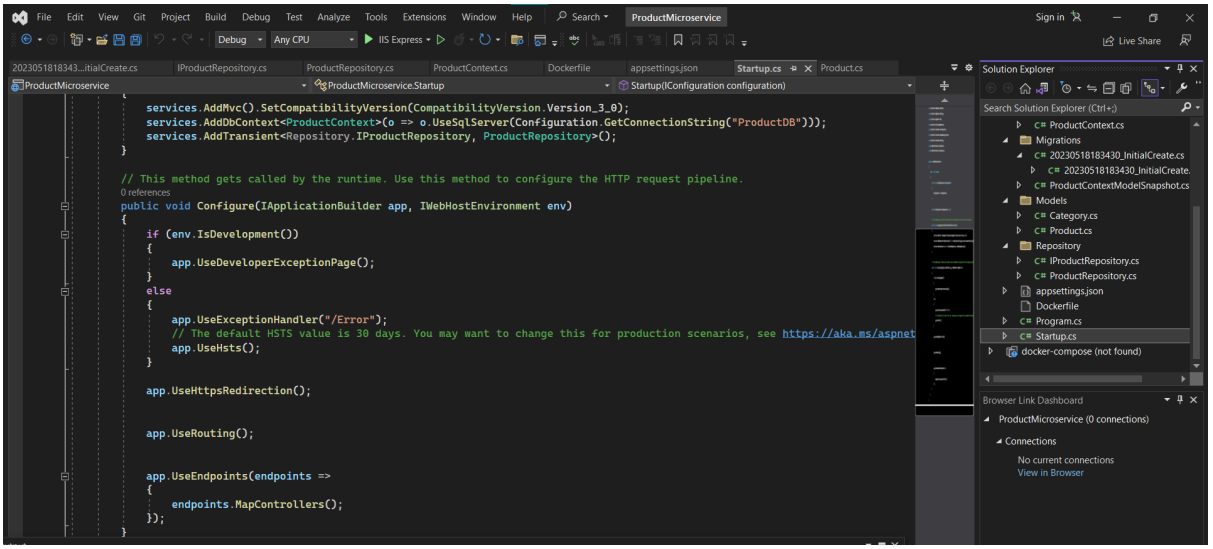
Step 11: Try to run the project, you will see that a browser window opens with an error



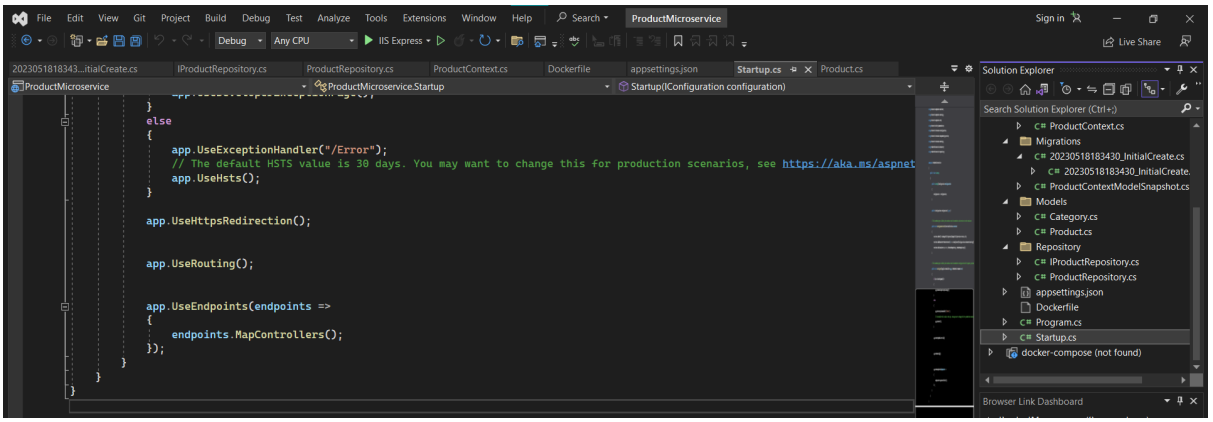
Step 12: Make some modifications to the Startup.cs file accordingly for the Routing mechanism of the API to work



```
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>();
        }
```

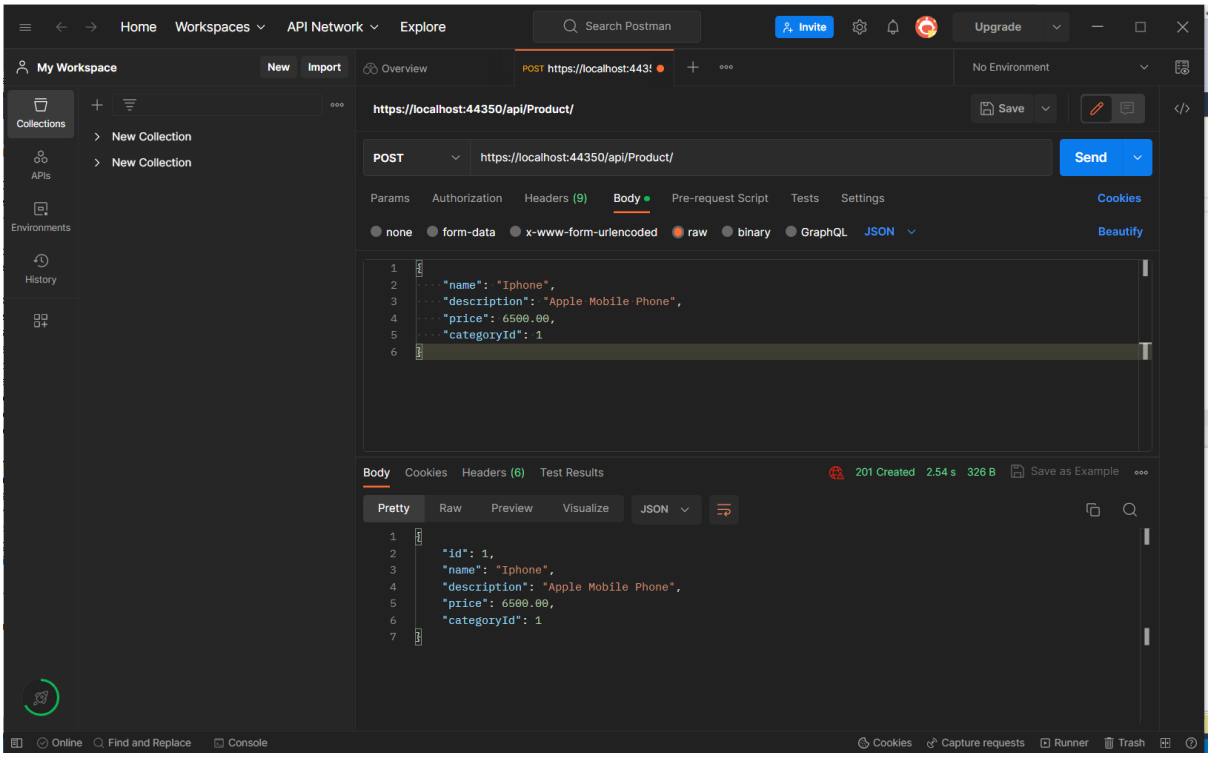


```
        }

        // 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();
            });
        }
```

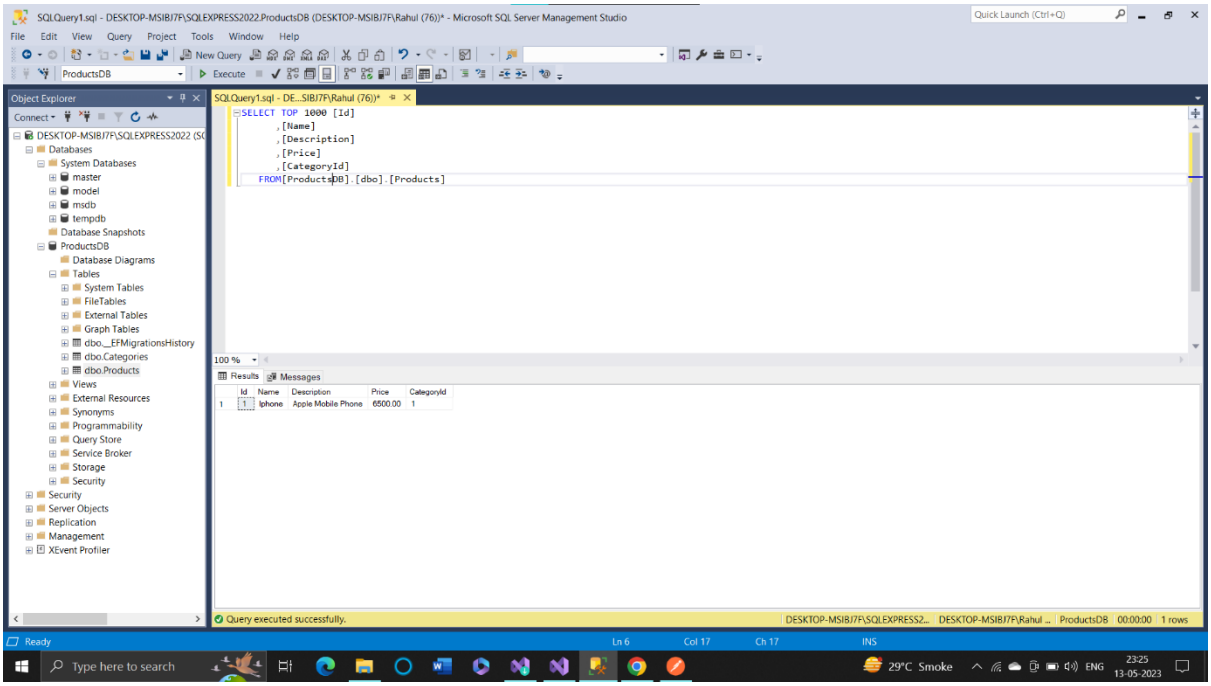


```
            }
            else
            {
                app.UseExceptionHandler("/Error");
                // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnet
                app.UseHsts();
            }

            app.UseHttpsRedirection();


            app.UseRouting();


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

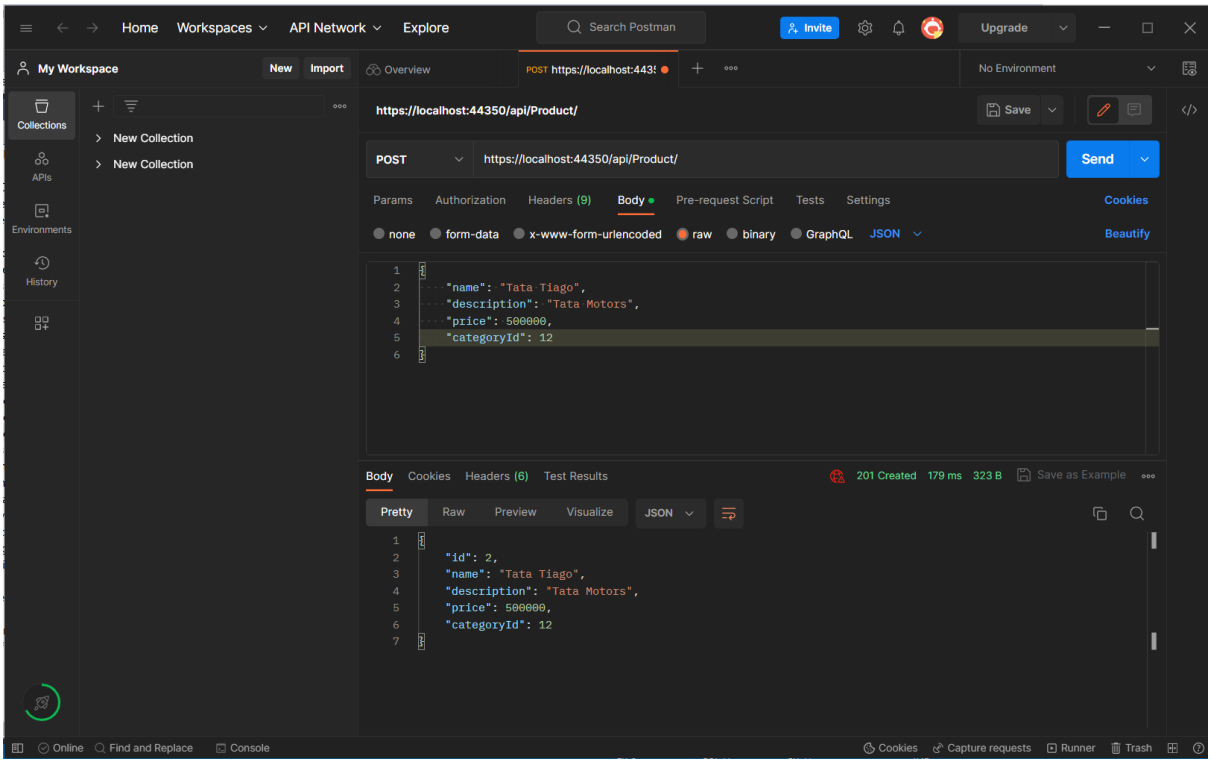
Step 13: Using Postman try to add a product into the product table via the api call



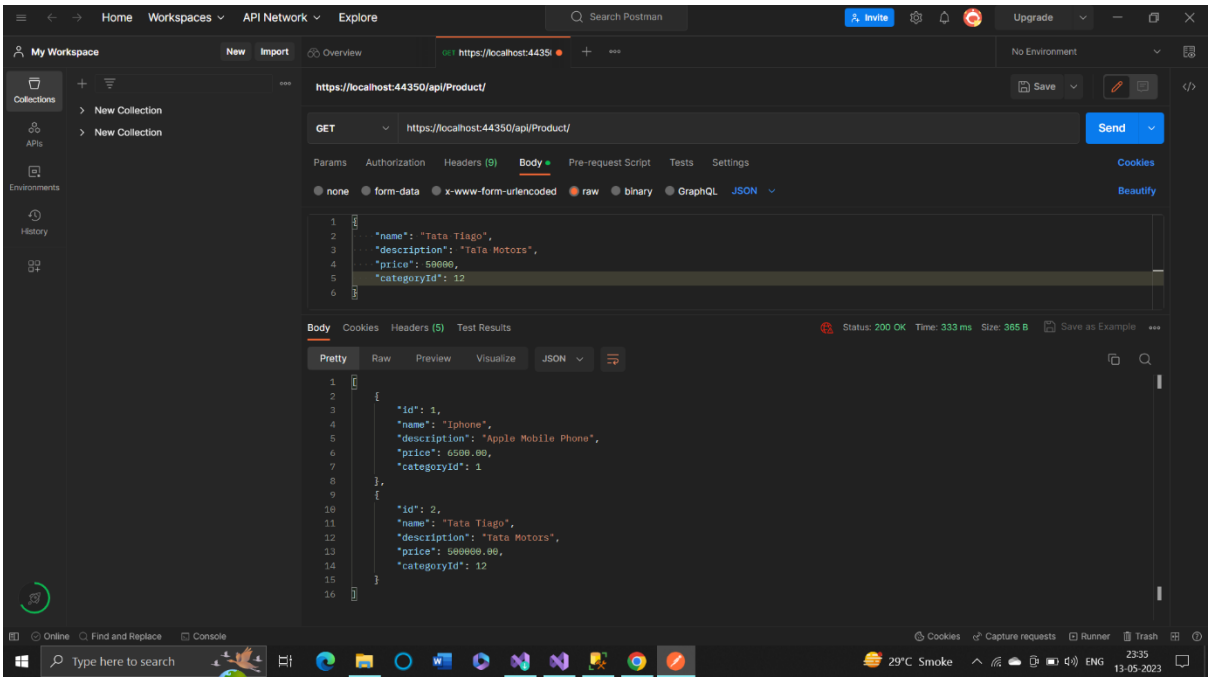
Step 14: You will notice the SQL table to populate with that product



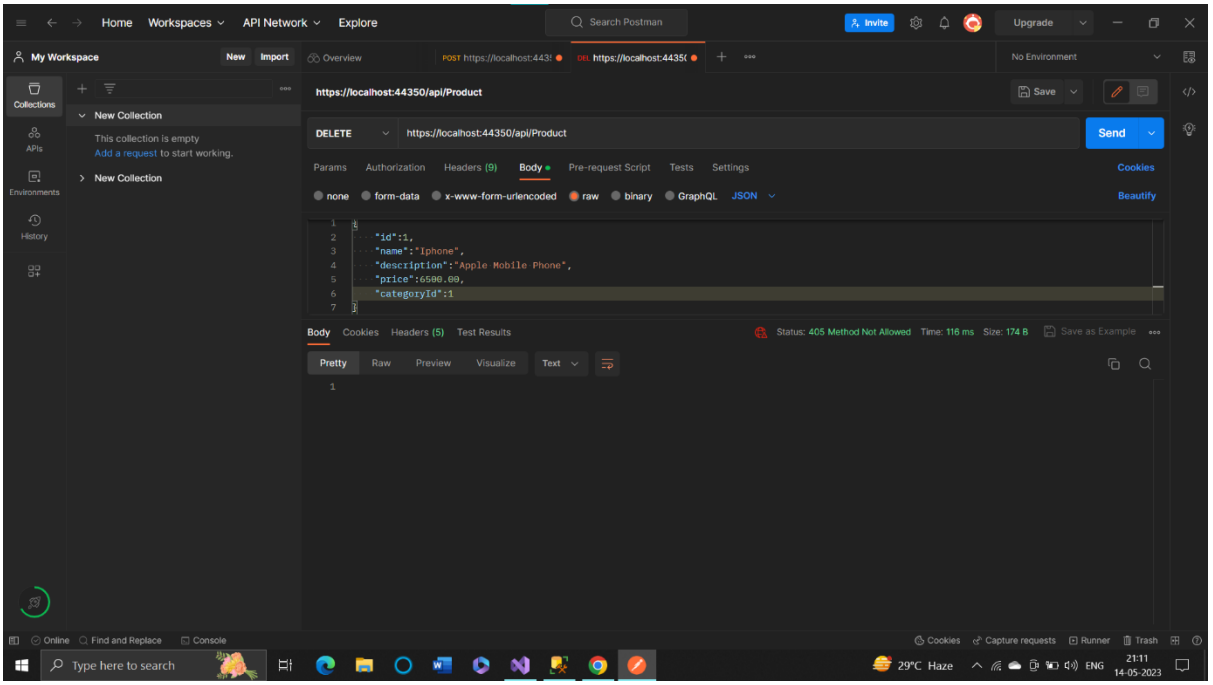
Step 15: Add another Product to test



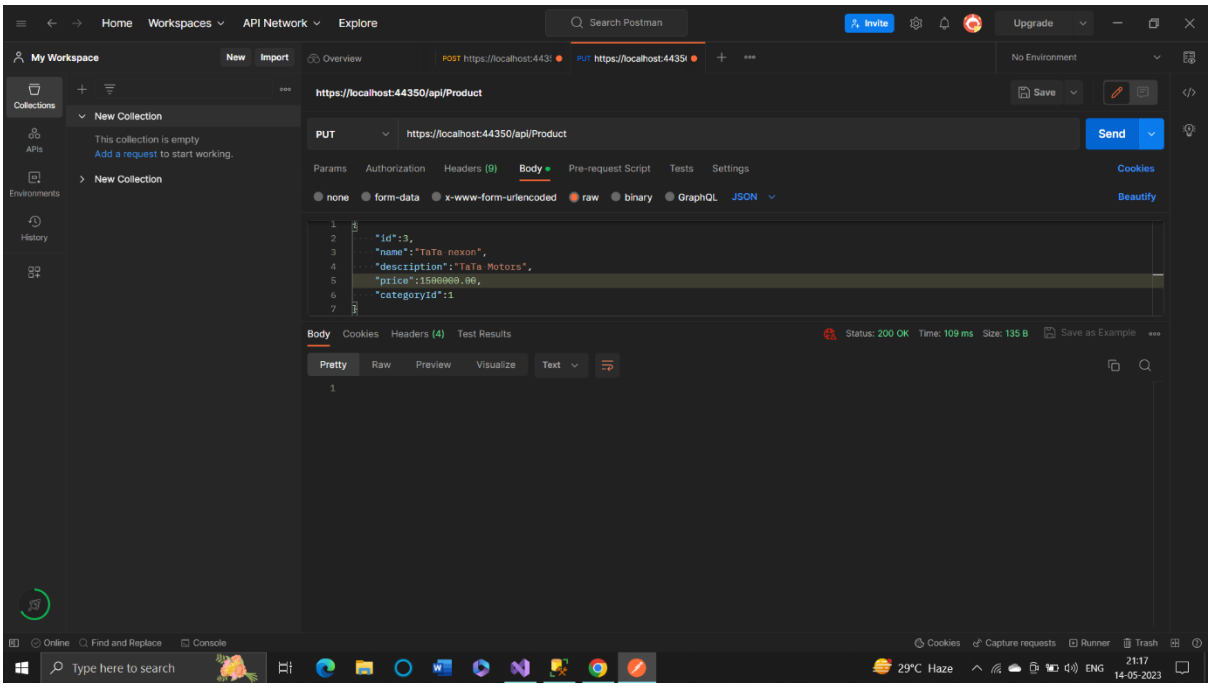
Step 16: Get all the added products using GET

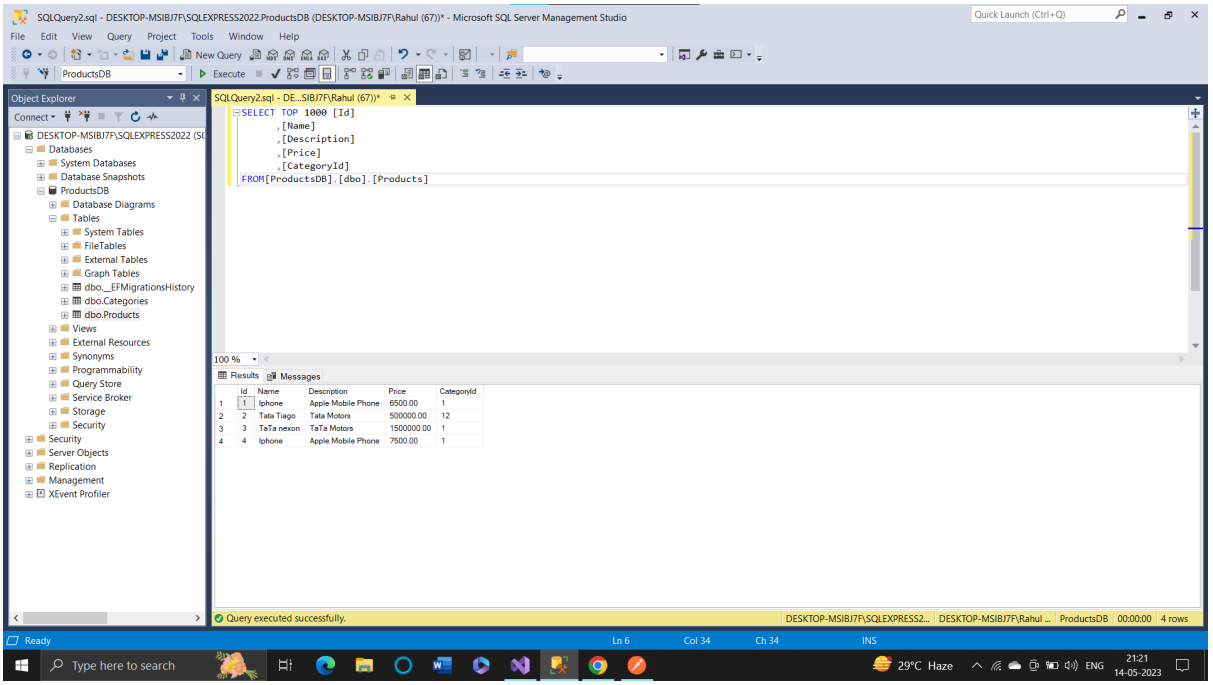


Step 17: Delete a product using DELETE



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]
Id Name Description Price CategoryId
1 Iphone Apple Mobile Phone 6500.00 1
2 Tata Tiago Tata Motors 500000.00 12
3 TaTa nexon TaTa Motors 1500000.00 1
4 Iphone Apple Mobile Phone 7500.00 1
Query executed successfully.
Ready
29°C Haze
21:21
14-05-2023

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.