

MICROSERVICE ARCHITECTURE

INDEX

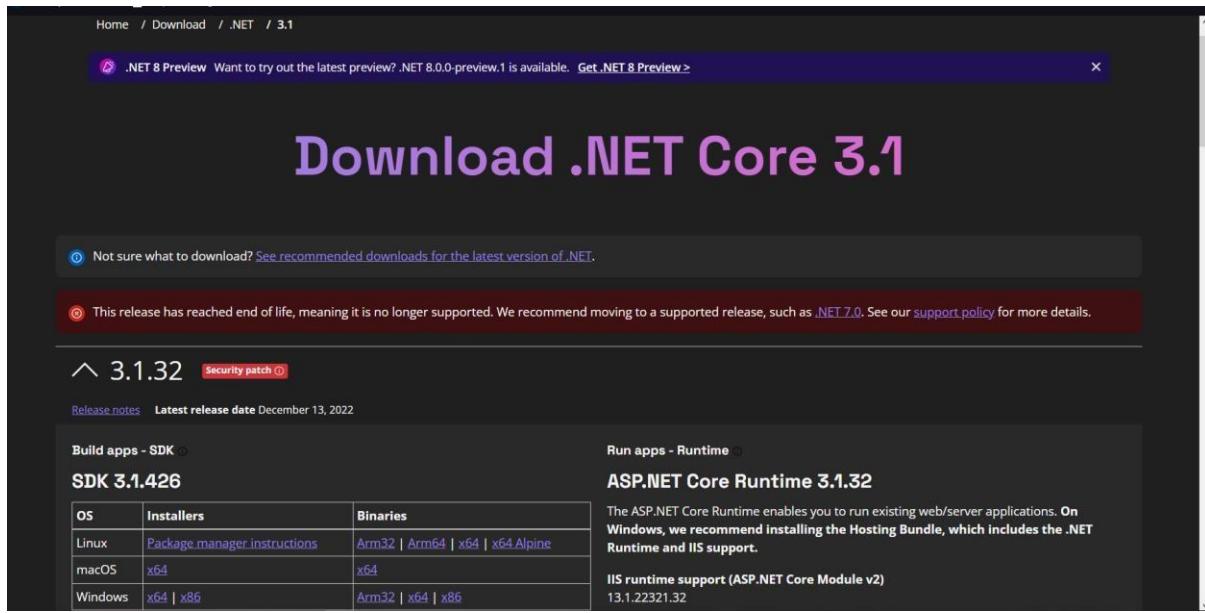
Sr. No	Date	Practical	Page No.	Sign
1	25-02-2023	Building ASP.NET Core MVC Application	1	
2	04-03-2023	Building ASP.NET Core REST API	7	
3	11-03-2023	Working with Docker, Docker Commands, Docker Images and Containers	20	
4	18-03-2023	Installing software packages on Docker, Working with Docker Volumes and Networks	28	
5	29-03-2023	Working with Circle CI for continuous integration	32	
6	01-04-2023	Creating Microservice with ASP.NET Core	43	
7	08-04-2023	Creating Backing Service with ASP.NET Core	58	
8	29-04-2023	Building real-time Microservice with ASP.NET Core	64	

Practical 1

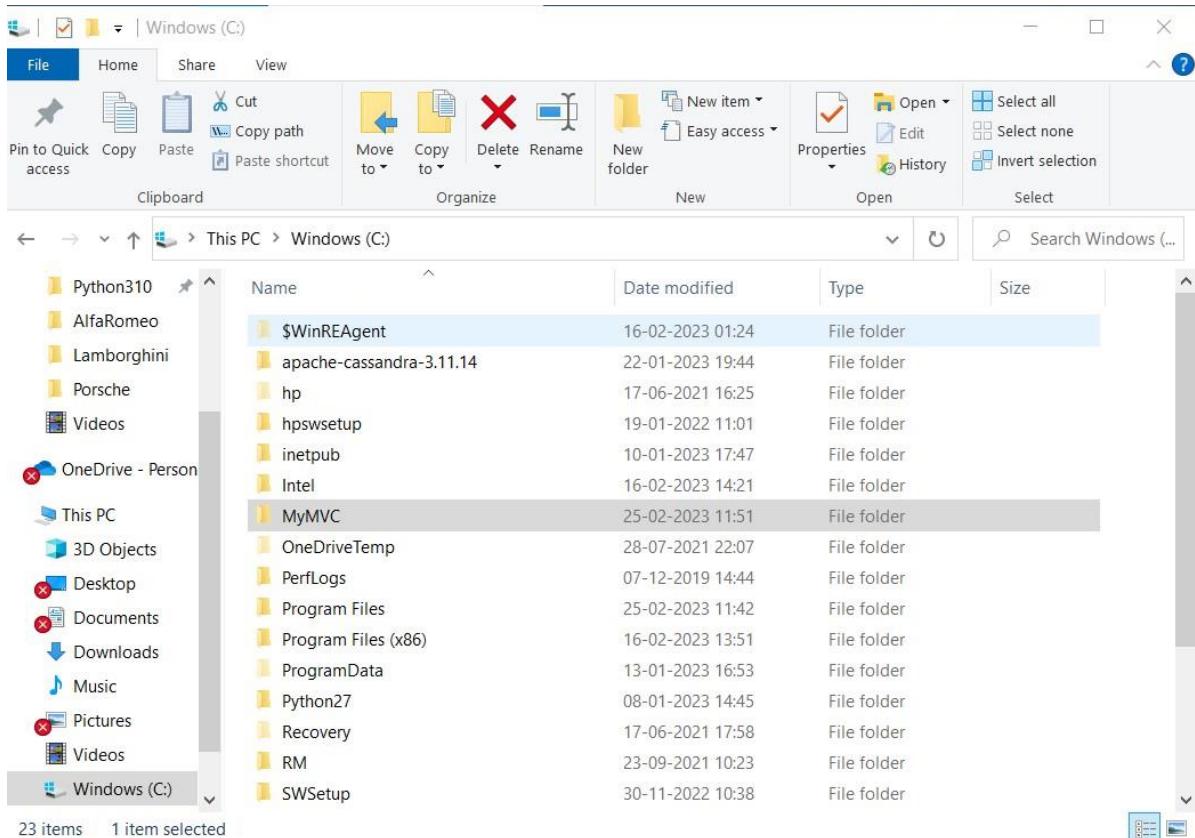
Aim: Building ASP.NET Core MVC Application.

Writeup:

Step 1:
Install .Net Core SDK.



Step 2:
Create a Folder MyMVC in C: drive.



Step 3:

Open Command Prompt and type the following commands:

dotnet new mvc --auth none

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.2604]
(c) Microsoft Corporation. All rights reserved.

C:\MyMVC\dotnet new mvc --auth none
The template "ASP.NET Core Web App (Model-View-Controller)" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/aspnetcore/3.1-third-party-notices for details.

Processing post-creation actions...
Running 'dotnet restore' on C:\MyMVC\MyMVC.csproj...
Determining projects to restore...
Restored C:\MyMVC\MyMVC.csproj (in 66 ms).

Restore succeeded.
```

Step 4:

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

```
C# HomeController.cs X

Controllers > C# HomeController.cs

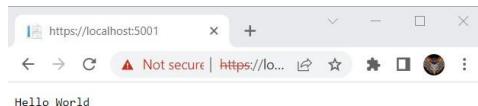
1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Linq;
5  using System.Threading.Tasks;
6  using Microsoft.AspNetCore.Mvc;
7  using Microsoft.Extensions.Logging;
8  using MyMVC.Models;
9
10 namespace MyMVC.Controllers
11 {
12     public class HomeController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

Step 5:

- Run the code

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

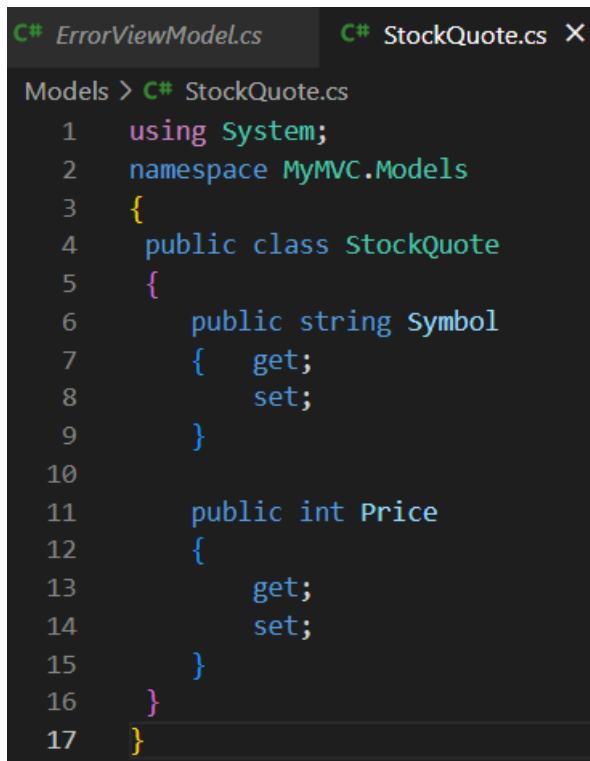
- Paste the localhost:5001 link in your browser (In the case we are using Chrome).

**Step 6:** Now go back to command prompt and stop running project using CTRL+C.

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

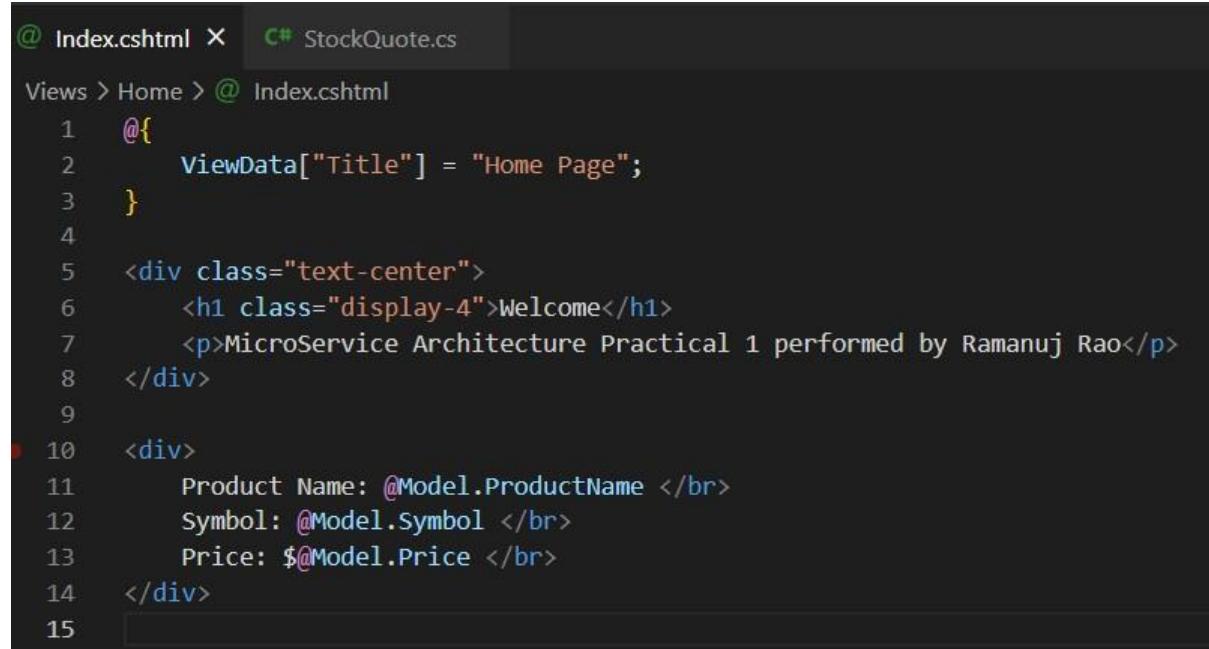
C:\MyMVC>
```

Step 7: Go to models folder and add new file StockQuote.cs to it with following content:



```
C# ErrorViewModel.cs      C# StockQuote.cs X
Models > C# StockQuote.cs
1  using System;
2  namespace MyMVC.Models
3  {
4      public class StockQuote
5      {
6          public string Symbol
7          {
8              get;
9              set;
10         }
11         public int Price
12         {
13             get;
14             set;
15         }
16     }
17 }
```

Step 8: Now Add View to folder then home folder in it and modify index.cshtml file to match following



```
@ Index.cshtml X  C# StockQuote.cs
Views > Home > @ Index.cshtml
1  @{
2      ViewData["Title"] = "Home Page";
3  }
4
5  <div class="text-center">
6      <h1 class="display-4">Welcome</h1>
7      <p>MicroService Architecture Practical 1 performed by Ramanuj Rao</p>
8  </div>
9
10 <div>
11     Product Name: @Model.ProductName <br>
12     Symbol: @Model.Symbol <br>
13     Price: $@Model.Price <br>
14 </div>
15
```

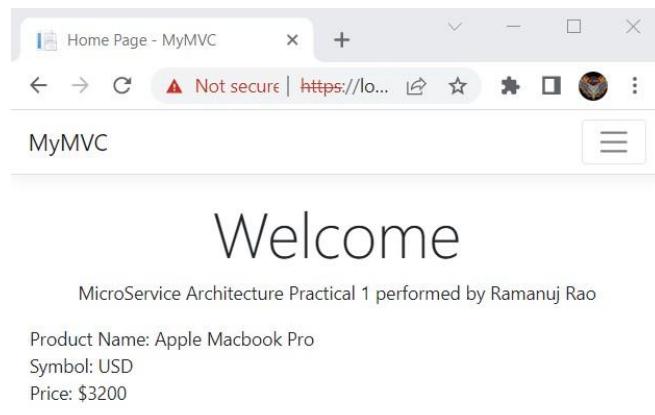
Step 9: Now modify HomeController.cs file to match following:

```
C# StockQuote.cs    C# HomeController.cs X
Controllers > C# HomeController.cs
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 HomeController(ILogger<HomeController> logger)
17         {
18             _logger = logger;
19         }
20
21         public IActionResult Index()
22         {
23             var model= new StockQuote{ ProductName='Apple Macbook Pro', Symbol='USD', Price='3200'};
24             return View(model);
25         }
26     }
}
```

Step 10: Now run the project using **dotnet run**

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

Step 11: Now go back to browser and refresh to get modified view response.



Practical 2

Aim: Building an ASP.NET CORE REST API.

Writeup:

Step 1: Create your Web API.

- Open two command prompts.

CMD-1:**dotnet new webapi -o Glossary**

```
Windows Command Prompt
Microsoft Windows [Version 10.0.19044.2604]
(c) Microsoft Corporation. All rights reserved.

C:\Users\raman>cd..

C:\Users>cd..

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

Processing post-creation actions...
Running 'dotnet restore' on Glossary\Glossary.csproj...
  Determining projects to restore...
    Restored C:\Glossary\Glossary.csproj (in 108 ms).

Restore succeeded.
```

CMD-2:**cd Glossary**
dotnet run

```
Windows Command Prompt - dotnet run
Microsoft Windows [Version 10.0.19044.2604]
(c) Microsoft Corporation. All rights reserved.

C:\Users\raman>cd..

C:\Users>cd..

C:\>cd Glossary

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

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

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

```
C:\>curl --insecure https://localhost:5001/weatherforecast
[{"date": "2023-03-11T17:28:21.0483959+05:30", "temperatureC": 50, "temperatureF": 121, "summary": "Sweltering"}, {"date": "2023-03-12T17:28:21.0487982+05:30", "temperatureC": -13, "temperatureF": 9, "summary": "Sweltering"}, {"date": "2023-03-13T17:28:21.0488162+05:30", "temperatureC": 7, "temperatureF": 44, "summary": "Cool"}, {"date": "2023-03-14T17:28:21.0488168+05:30", "temperatureC": 47, "temperatureF": 116, "summary": "Warm"}, {"date": "2023-03-15T17:28:21.0488171+05:30", "temperatureC": 37, "temperatureF": 98, "summary": "Balmy"}]
```

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

[i]: **GlossaryItem.cs**

```
C# GlossaryItem.cs
1  namespace Glossary
2  {
3      public class GlossaryItem
4      {
5          public string Term
6          {
7              get;
8              set;
9          }
10
11         public string Definition
12         {
13             get;
14             set;
15         }
16     }
17 }
```

[ii]: **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"  
    }  
};  
  
[HttpGet]  
public ActionResult<List<GlossaryItem>> Get()  
{  
    return Ok(Glossary);  
}  
  
[HttpGet]  
[Route("{term}")]  
public ActionResult<GlossaryItem> Get(string term)  
{  
    var glossaryItem = Glossary.Find(item =>  
        item.Term.Equals(term,  
StringComparison.InvariantCultureIgnoreCase));  
    if (glossaryItem == null)  
    {  
        return NotFound();  
    }  
  
    else  
    {  
        return Ok(glossaryItem);  
    }  
}  
  
[HttpPost]  
public ActionResult Post(GlossaryItem glossaryItem)  
{  
    var existingGlossaryItem = Glossary.Find(item =>
```

```
        item.Term.Equals(glossaryItem.Term,
StringComparison.InvariantCultureIgnoreCase));
        if (existingGlossaryItem != null)
        {
            return Conflict("Cannot create the term because it already
exists.");
        }

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

    [HttpPost]
    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.");
        }

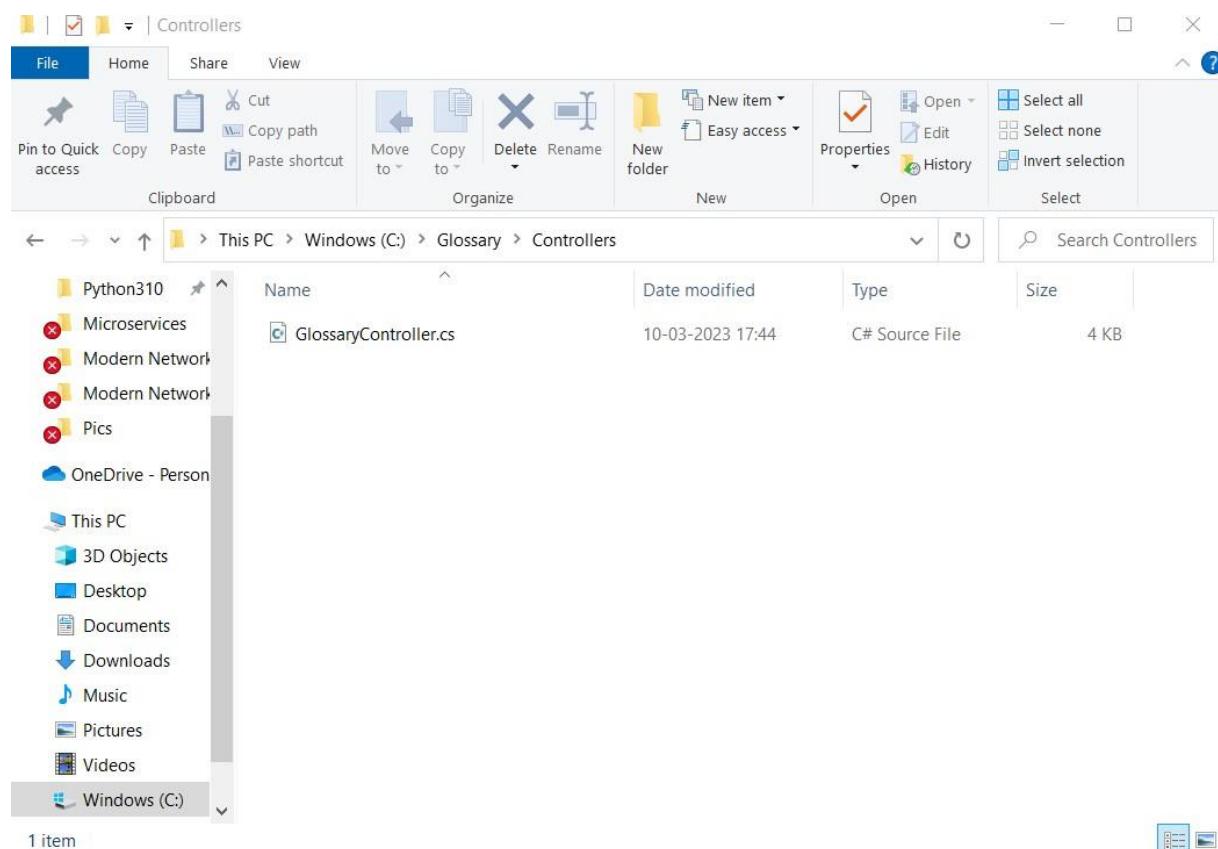
        else
        {
            existingGlossaryItem.Definition = glossaryItem.Definition;
            return Ok();
        }
    }

    [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.

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

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

[i] : Getting a list of items.

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

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

[ii] : Getting a single item.

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

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

[iii] : Creating an item.

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

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

[iv] : Update an item.

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

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

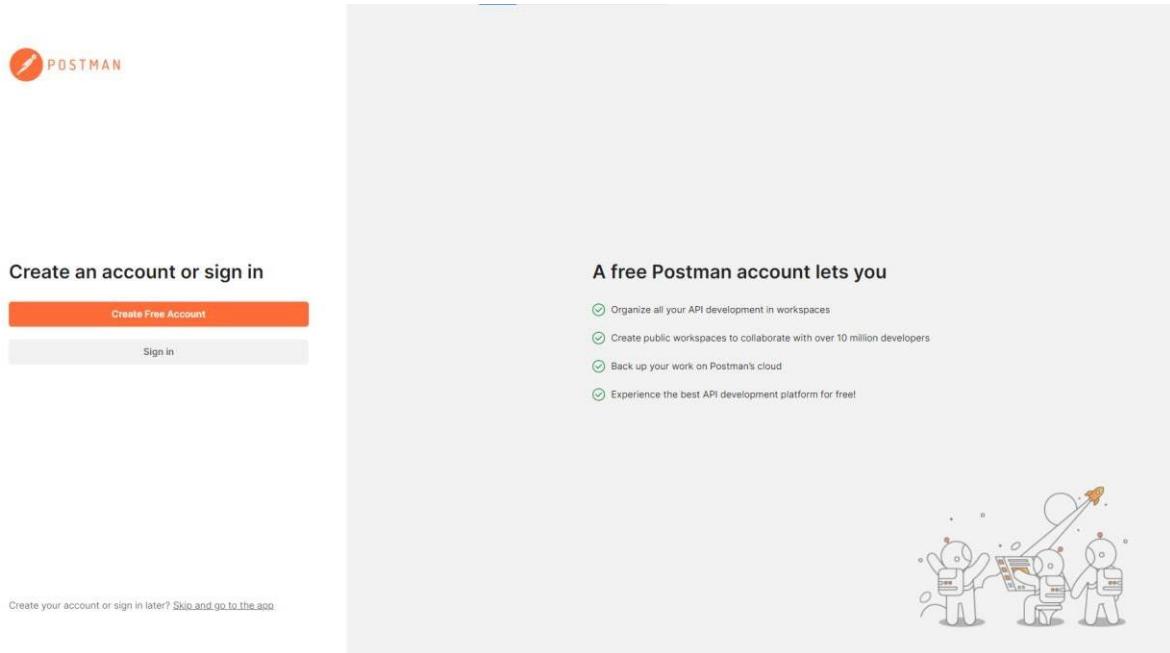
[v] : Delete an item.

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."}]
```

Extra (Step 6): Running commands using Postman.

- First you have you download and install Postman.
- After installing you will see this screen



- Click on “Skip and go to the app”
- Now Click on New Collection and give it a name or your choice

Home Workspaces Explore

Scratch Pad

New Import Overview GET New Request POST New Request PUT New Request DEL New Request Practical-2

Sign In Create Account

Practical-2

Authorization Pre-request Script Tests Variables

Type No Auth

This collection does not use any authorization. Learn more about authorization ↗

Documentation

Add collection description...

View complete collection documentation →

Find and Replace Console Runner Trash

- Now within the new collection Click on “Add new Request” and within the choices choose the “GET” option.

The screenshot shows the Postman interface with the 'Practical-2 / New Request' collection selected. A context menu is open over the 'GET' method, listing various HTTP methods: GET, POST, PUT, PATCH, DELETE, COPY, HEAD, OPTIONS, LINK, UNLINK, PURGE, LOCK, UNLOCK, PROPFIND, and VIEW. The 'GET' method is highlighted. The main request configuration area shows the method set to 'GET', the URL field empty, and several tabs: Headers (6), Body, Pre-request Script, Tests, and Settings. Below the tabs is a table for 'Headers' and 'Body'. A cartoon character icon is visible in the center of the interface, and a note at the bottom says 'Enter the URL and click Send to get a response'.

- Now type in **localhost:5000/api/glossary** in the field and Click **Send**.

The screenshot shows the Postman interface after sending a GET request to 'localhost:5000/api/glossary'. The URL field now contains 'localhost:5000/api/glossary'. The 'Params' tab is selected, showing a single parameter 'Key' with value 'Value'. The 'Body' tab is active, displaying the JSON response. The response body is a JSON object with three items:

```

1  [
2   {
3     "term": "HTML",
4     "definition": "Hypertext Markup Language"
5   },
6   {
7     "term": "MVC",
8     "definition": "Model View Controller"
9   },
10  {
11    "term": "OpenID",
12    "definition": "An open standard for authentication"
13  }
14 ]

```

The status bar at the bottom indicates 'Status: 200 OK Time: 6 ms Size: 327 B Save Response'.

- Now within the new collection Click on “Add new Request” and within the choices choose the “POST” option.

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Collections (mfas), APIs, Environments, Mock Servers, Monitors, History.
- Request Type:** POST (selected).
- URL:** Enter request URL (empty).
- Method Options:** GET, POST, PUT, PATCH, DELETE, COPY, HEAD, OPTIONS, LINK, UNLINK, PURGE, LOCK, UNLOCK, PROPFIND, VIEW.
- Body Tab:** Headers (7), Body, Pre-request Script, Tests, Settings.
- Body Content:** Value (empty) and Description (empty).
- Send Button:** Send (blue button).
- Bottom Status:** Enter the URL and click Send to get a response.

- Now type in **localhost:5000/api/glossary?=** in the field.
- Then choose the “Body” option below and select “Raw” and fill out the following code:

Code:

```
{
  "term": "MVC3",
  "definition": "Model View Controller 3"
}
```

- Now Click **Send**.
- Tip: Make sure to select **JSON** instead of **Text**.

The screenshot shows the Postman interface after sending the request:

- Left Sidebar:** Collections (mfas), APIs, Environments, Mock Servers, Monitors, History.
- Request Type:** POST (selected).
- URL:** localhost:5000/api/glossary?=
- Body Tab:** Params (green dot), Authorization, Headers (8), Body (green dot), Pre-request Script, Tests, Settings.
- Body Content:** JSON tab selected, showing the JSON code from the previous step.
- Response:** Status: 201 Created, Time: 4 ms, Size: 237 B, Save Response.
- Body Content (Pretty):**

```

1   {
2     "term": "MVC3",
3     "definition": "Model View Controller 3"
4 }
```

- To confirm that the **POST** Request worked send another **GET** Request and it would have updated with a new value, if not then check your code.

The screenshot shows the Postman interface with a collection named "Practical-2". A GET request is made to "localhost:5000/api/glossary". The response status is 200 OK, and the response body is a JSON array of terms:

```

1 [
2   {
3     "term": "HTML",
4     "definition": "Hypertext Markup Language"
5   },
6   {
7     "term": "MVC",
8     "definition": "Model View Controller"
9   },
10  {
11    "term": "OpenID",
12    "definition": "An open standard for authentication"
13  },
14  {
15    "term": "MVC3",
16    "definition": "Model View Controller 3"
17  }
18 ]

```

- Now within the new collection Click on “Add new Request” and within the choices choose the “PUT” option.

The screenshot shows the Postman interface with a collection named "Practical-2". A PUT request is being configured to "localhost:5000/api/glossary". The method dropdown is set to PUT. The response status is 200 OK, and the response body is a JSON object:

```

{
  "Value": "Updated Definition for Model View Controller 3"
}

```

- Now type in **localhost:5000/api/glossary** in the field.
- Then choose the “Body” option below and select “Raw” and fill out the following code:

Code:

```
{
  "term": "MVC3",
  "definition": "Updated Definition for Model View Controller 3"
}
```

- Click **Send**.
- To confirm the changes made by the PUT Request, send another GET Request to update the values.

The screenshot shows the Postman interface with a successful GET request to `localhost:5000/api/glossary`. The response body is a JSON array of terms:

```

1  [
2   {
3     "term": "HTML",
4     "definition": "HyperText Markup Language"
5   },
6   {
7     "term": "MVC",
8     "definition": "Model View Controller"
9   },
10  {
11    "term": "OpenID",
12    "definition": "An open standard for authentication"
13  },
14  {
15    "term": "MVC3",
16    "definition": "Updated Definition for Model View Controller 3"
17 }
18

```

- Now within the new collection Click on “Add new Request” and within the choices choose the “DELETE” option.

The screenshot shows the Postman interface with a new collection named "Practical-2". A context menu is open over a "GET New Request" item, with "DELETE" selected. The response body is a JSON object:

```

{
  "method": "DELETE",
  "url": "localhost:5000/api/glossary/OpenID"
}

```

- Now type in “`localhost:5000/api/glossary`” in the field and after it choose the term you would like to remove in this case I have chosen “**OpenID**”.
- After this Click **Send**.

The screenshot shows the Postman interface with a collection named "Practical-2". A DELETE request is made to the endpoint `localhost:5000/api/glossary/OpenID`. The response status is 204 No Content, indicating that the term was successfully deleted.

- To Confirm the DELETE Request, send another GET Request to update the table.

The screenshot shows the Postman interface with the same collection "Practical-2". A GET request is made to the endpoint `localhost:5000/api/glossary`. The response status is 200 OK, and the JSON response body contains the following data:

```

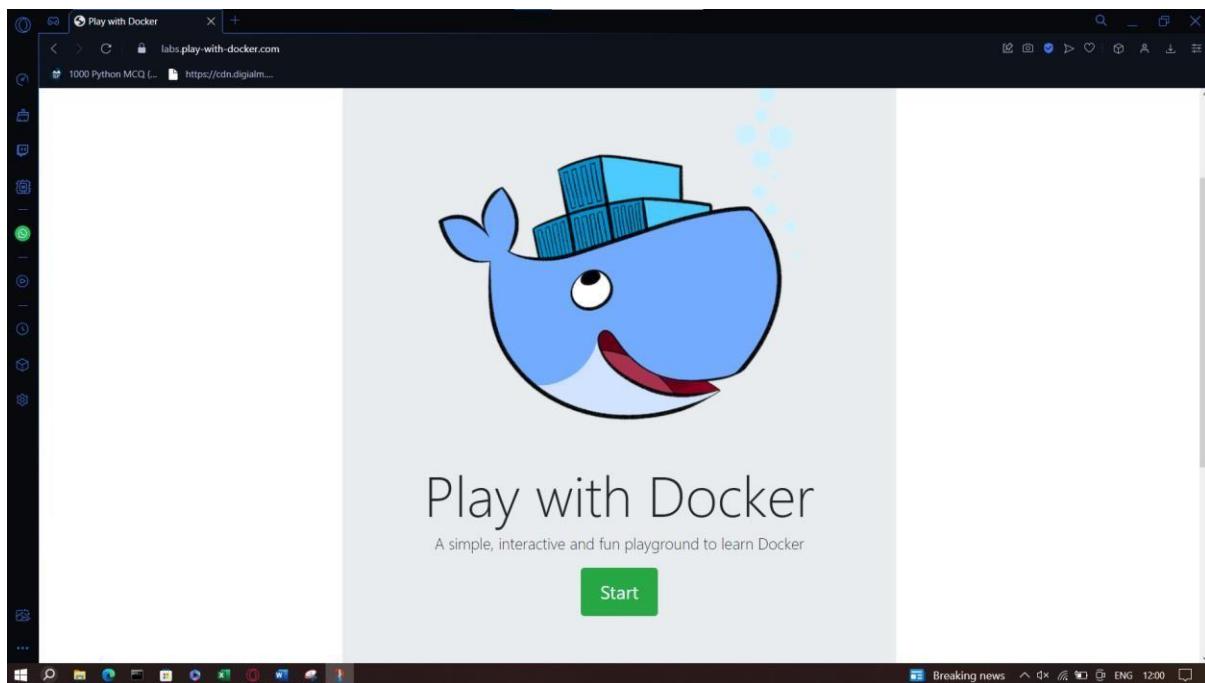
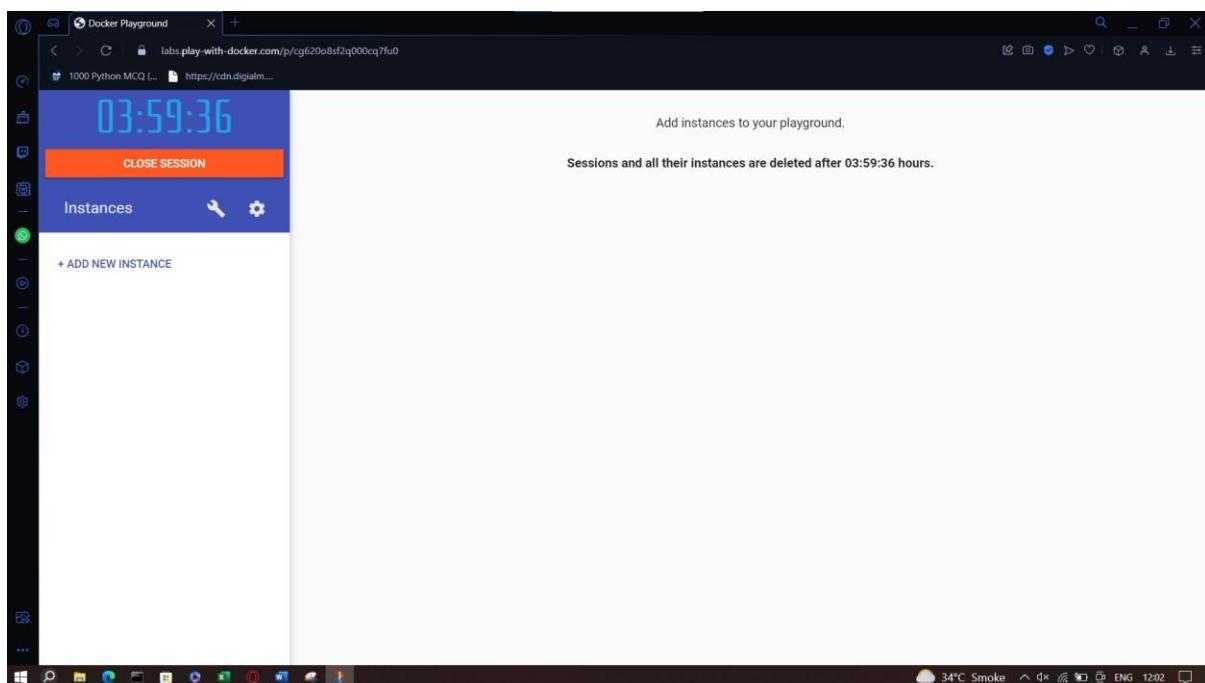
1   [
2     {
3       "term": "HTML",
4       "definition": "Hypertext Markup Language"
5     },
6     {
7       "term": "MVC",
8       "definition": "Model View Controller"
9     },
10    {
11      "term": "MVC3",
12      "definition": "Updated Definition for Model View Controller 3"
13    }
14 ]

```

Practical 3

Aim: Working with Docker, Docker Commands, Docker Images and Containers.

Writeup:

Step 1: Create a Docker Hub Account (Sign Up)Login to this website <https://labs.play-with-docker.com/>**Step 2:** Click on Start and Add a new Instance.

Step 3: Perform the following

Method 1: To pull and push images using docker

Command: To check docker version

docker –version

Output:

```
cg620o8s_cg6213osf2q000cq7fug
IP: 192.168.0.18 OPEN PORT
Memory: 1.15% (45.98MiB / 3.906GiB) CPU: 0.00%
SSH: ssh ip172-19-0-15-cg620o8sf2q000cq7fug@direct.labs.play [ ]
DELETE EDITOR

#####
# WARNING!!!!#
# This is a sandbox environment. Using personal credentials#
# is HIGHLY! discouraged. Any consequences of doing so are#
# completely the user's responsibilites.#
##
# The PWD team.
#####
(node1) (local) root@192.168.0.18 ~
$ docker --version
Docker version 20.10.17, build 100c701
(node1) (local) root@192.168.0.18 ~
$ [ ]
```

Command: To pull ready-made images.

docker pull rocker/verse

Output:

```
cg620o8s_cg6213osf2q000cq7fug
IP: 192.168.0.18 OPEN PORT
Memory: 74.59% (2.914GiB / 3.906GiB) CPU: 0.00%
SSH: ssh ip172-19-0-15-cg620o8sf2q000cq7fug@direct.labs.play [ ]
DELETE EDITOR

(node1) (local) root@192.168.0.18 ~
$ docker --version
Docker version 20.10.17, build 100c701
(node1) (local) root@192.168.0.18 ~
$ docker pull rocker/verse
Using default tag: latest
latest: Pulling from rocker/verse
75769433fd8a: Pull complete
04aaad001c33: Pull complete
b593023d0f01: Pull complete
90f9547708a: Pull complete
de634d6eeac2: Pull complete
66480f89de4f: Pull complete
38b8432a63be: Pull complete
4fed43313400: Pull complete
8279c3963ce9: Pull complete
dac6f07e377b: Pull complete
Digest: sha256:5f4ecbeb19f915954645c5f23e52fad04cbbef1c8a6ccb6c39081050285c645
Status: Downloaded newer image for rocker/verse:latest
docker.io/rocker/verse:latest
(node1) (local) root@192.168.0.18 ~
$ [ ]
```

Command: To check images in docker.

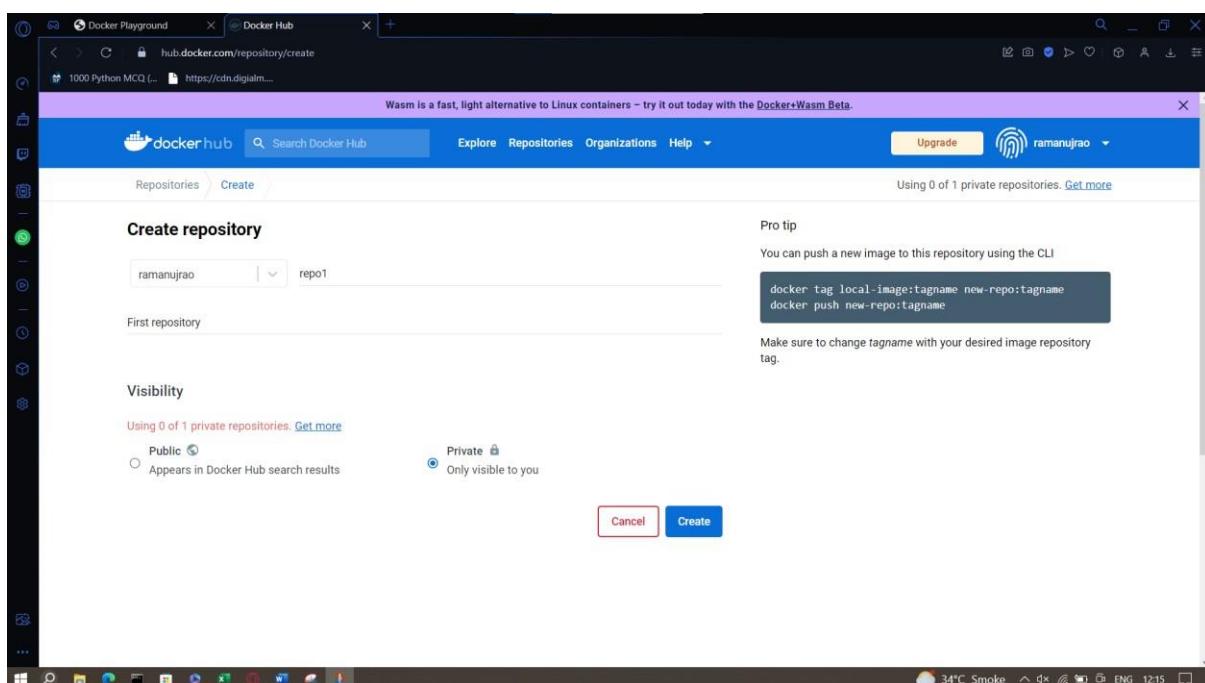
```
docker images
```

Output:

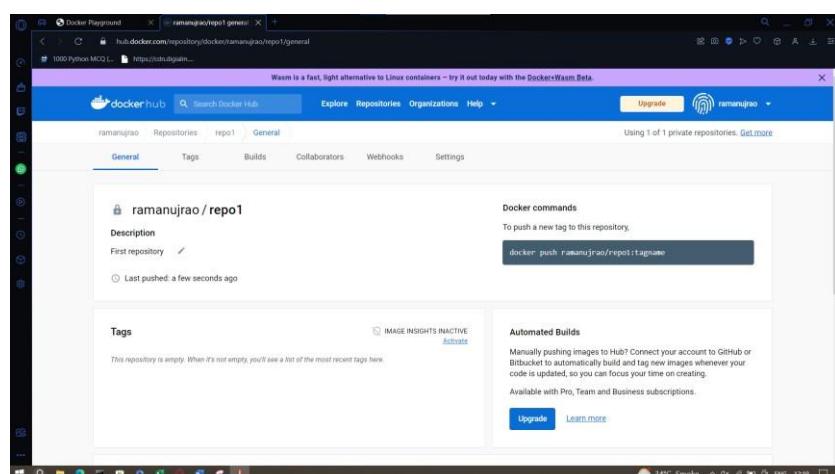
```
$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
rocker/verse    latest   551e1a37de34  8 days ago  3.43GB
[nodel] (local) root@192.168.0.18 ~
```

- Now login to Docker Hub and create a repository.

Output:



- Click on Create Button
- Check to see if the repository has been created



Command: to login to your docker account.

```
docker login --username=ramanujrao
password:
```

```
[node1] (local) root@192.168.0.18 ~
$ docker login --username=ramanujrao
Password:
Error response from daemon: Get "https://registry-1.docker.io/v2/": unauthorized: incorrect username or password
[node1] (local) root@192.168.0.18 ~
$ docker login --username=ramanujrao
Password:
Error: Password Required
[node1] (local) root@192.168.0.18 ~
$ docker login --username=ramanujrao
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@192.168.0.18 ~
$ []
```

Command: To tag an image.

```
docker tag 551e1a37de34 ramanujrao/repo1:firsttry
```

Output:

```
[node1] (local) root@192.168.0.18 ~
$ docker images
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE
rocker/verse    latest       551e1a37de34   8 days ago   3.43GB
[node1] (local) root@192.168.0.18 ~
$ docker tag 551e1a37de34 ramanujrao/repo1:firsttry
[node1] (local) root@192.168.0.18 ~
$ []
```

Command: To push image to docker hub account.

```
docker push ramanujrao/repo1:firsttry
```

Output:

```
$ docker push ramanujrao/repo1:firsttry
The push refers to repository [docker.io/ramanujrao/repo1]
6c1711f305ff: Mounted from rocker/verse
54cc7e366446: Mounted from rocker/verse
1e82ee1f79d4: Mounted from rocker/verse
e4f6f141a475: Mounted from rocker/verse
94644a51ea10: Mounted from rocker/verse
99e44ef3e8e9: Mounted from rocker/verse
fa35739b43d8: Mounted from rocker/verse
a0f5608ee4a8: Mounted from rocker/verse
e7484d5519b7: Mounted from rocker/verse
202fe64c3ce3: Mounted from rocker/verse
firsttry: digest: sha256:66e87a013127faaf3065f9c9544c47f5fd61484321fec6058f411a0687de4a6f size: 2428
[node1] (local) root@192.168.0.18 ~
$ []
```

- Check it in Docker Hub now.

The screenshot shows the Docker Hub interface for the repository `ramanujrao/repo1`. The General tab is active. The repository details include:

- Description:** First repository
- Last pushed:** a minute ago

Docker commands:

```
docker push ramanujrao/repo1:tagname
```

Tags:

Tag	OS	Type	Pulled	Pushed
firsttry	Ubuntu	Image	---	a minute ago

[See all](#) [Go to Advanced Image Management](#)

Automated Builds:

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

Available with Pro, Team and Business subscriptions.

[Upgrade](#) [Learn more](#)

- Click on tags and check

The screenshot shows the Docker Hub interface for the repository `ramanujrao/repo1`, specifically the Tags page. The Tags tab is active. The tag details are:

TAG	DIGEST	OS/ARCH	SCANNED	LAST PULL	COMPRESSED SIZE
firsttry	66e87a013127	linux/amd64	---	---	1.25 GB

[docker pull ramanujrao/repo1...](#)

Method 2: Build an image then push it to docker and run it.

Command: To create a docker file.

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

Output:

```
[node1] (local) root@192.168.0.18 ~
$ cat > Dockerfile << EOF
> FROM busybox
> CMD echo "Hello world! This is Ramanuj Rao and this is my Docker Image."
> EOF
[node1] (local) root@192.168.0.18 ~
$ []
```

Command: to build image for a docker file.

```
docker build -t ramanujrao/repo2
```

Output:

```
$ docker build -t ramanujrao/repo2 .
Sending build context to Docker daemon 12.8kB
Step 1/2 : FROM busybox
latest: Pulling from library/busybox
1487bff95222: Pull complete
Digest: sha256:c118f538365369207c12e5794c3cbfb7b042d950af590ae6c287ede74f29b7d4
Status: Downloaded newer image for busybox:latest
--> bab98d58e29e
Step 2/2 : CMD echo "Hello world! This is Ramanuj Rao and this is my Docker Image."
--> Running in 304e5f79022a
Removing intermediate container 304e5f79022a
--> 45baae10ed17
Successfully built 45baae10ed17
Successfully tagged ramanujrao/repo2:latest
[node1] (local) root@192.168.0.18 ~
```

Command: To check docker images.

```
docker images
```

```
[node1] (local) root@192.168.0.18 ~
$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
ramanujrao/repo2  latest   45baae10ed17  About a minute ago  4.86MB
busybox          latest   bab98d58e29e   4 days ago    4.86MB
rocker/verse     latest   551e1a37de34  8 days ago    3.43GB
ramanujrao/repol  firsttry 551e1a37de34  8 days ago    3.43GB
[node1] (local) root@192.168.0.18 ~
$ []
```

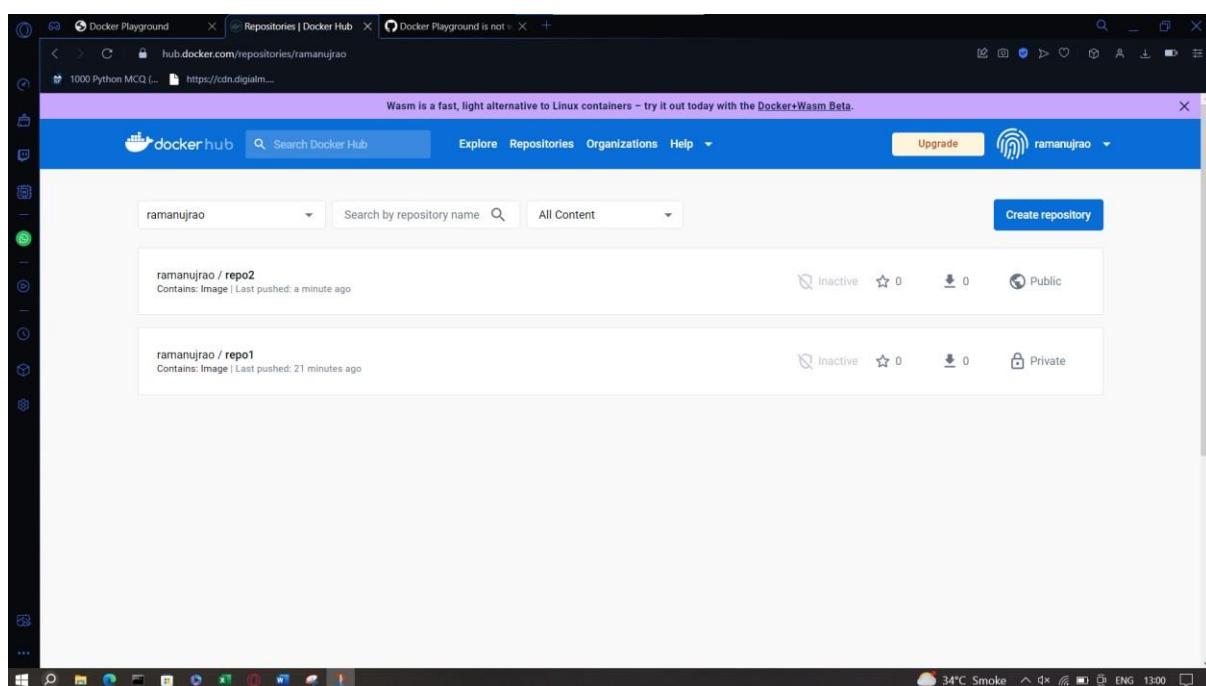
Command: to push image to docker hub.

```
docker push ramanujrao/repo2 .
```

Output:

```
[node1] (local) root@192.168.0.18 ~
$ docker push ramanujrao/repo2
Using default tag: latest
The push refers to repository [docker.io/ramanujrao/repo2]
427701cb9c96: Mounted from library/busybox
latest: digest: sha256:bb7a35449124b00ee196485e5020f5d7bc646c49576b7de608d4b9b81cf099bc size: 528
[node1] (local) root@192.168.0.18 ~
$ 
```

- Now check in Docker Hub.



Command: to run docker image:

```
docker run ramanujrao/repo2
```

Output:

```
$ docker run ramanujrao/repo2
Hello world! This is Ramanuj Rao and this is my Docker Image.
[node1] (local) root@192.168.0.18 ~
$ 
```

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.

```
Microsoft Windows [Version 10.0.19044.2728]
(c) Microsoft Corporation. All rights reserved.

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

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

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

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

```
C:\Users\raman>docker volume inspect myvol1
[
  {
    "CreatedAt": "2023-03-18T08:28:07Z",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/myvol1/_data",
    "Name": "myvol1",
    "Options": {},
    "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\raman>docker volume rm myvol1
myvol1

C:\Users\raman>docker volume ls
DRIVER      VOLUME NAME

C:\Users\raman>
```

Step 2: Working with Docker Network

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

```
C:\Users\raman>docker network create Vol
28deade85cb4918dcccfc3dab6905c56d63c27bb9c9c1ca2638c829336f851503
```

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

```
C:\Users\raman>docker network inspect Vol
[
  {
    "Name": "Vol",
    "Id": "28deade85cb4918dcccfc3dab6905c56d63c27bb9c9c1ca2638c829336f851503",
    "Created": "2023-04-07T10:55:14.586981516Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1"
        }
      ]
    },
    "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\raman>docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
28deade85cb4   Vol       bridge      local
008c69b55069   bridge     bridge      local
26198ed8c76c   host       host       local
aed51ccdff48   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\raman>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\raman>docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
008c69b55069   bridge     bridge      local
26198ed8c76c   host       host       local
aed51ccdff48   none      null       local
```

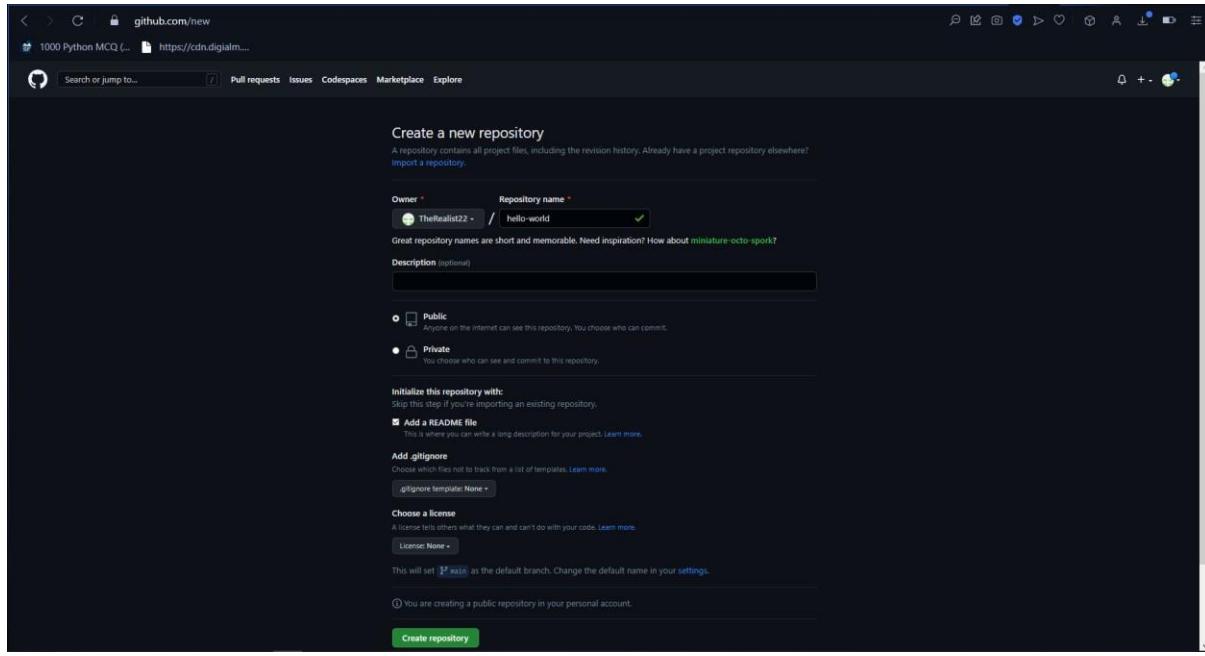
Practical 5

Aim: Working with Circle CI for continuous integration.

Writeup:

Step 1: Create a repository

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

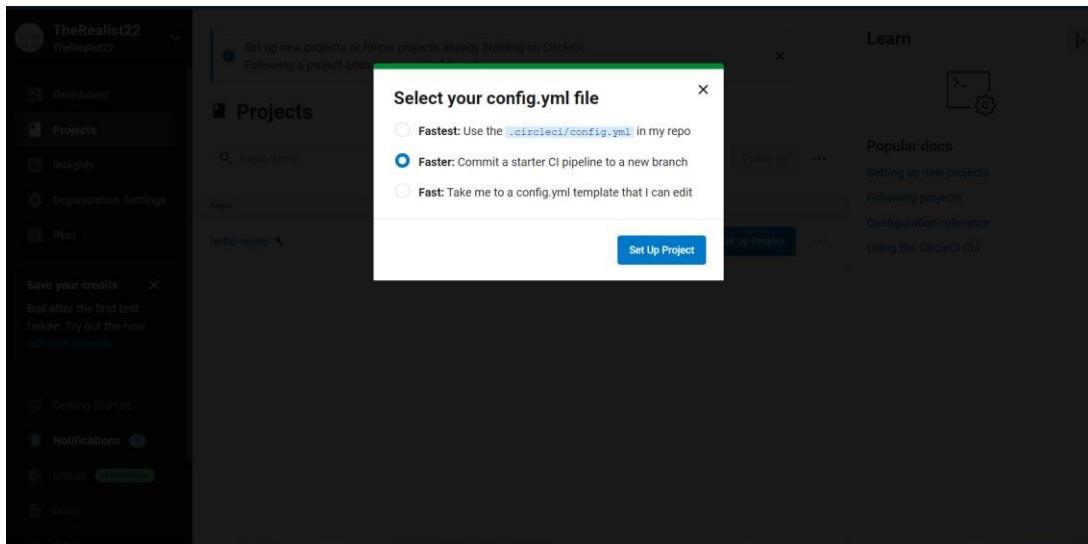


- Login to Circle CI <https://app.circleci.com/> Using GitHub Login, Once logged in navigate to Projects.

The screenshot shows the CircleCI dashboard under the 'Projects' section. A project named 'hello-world' is listed. To the right, there is a 'Learn' sidebar with links to 'Popular docs' such as 'Setting up new projects', 'Following projects', 'Configuration reference', and 'Using the CircleCI CLI'.

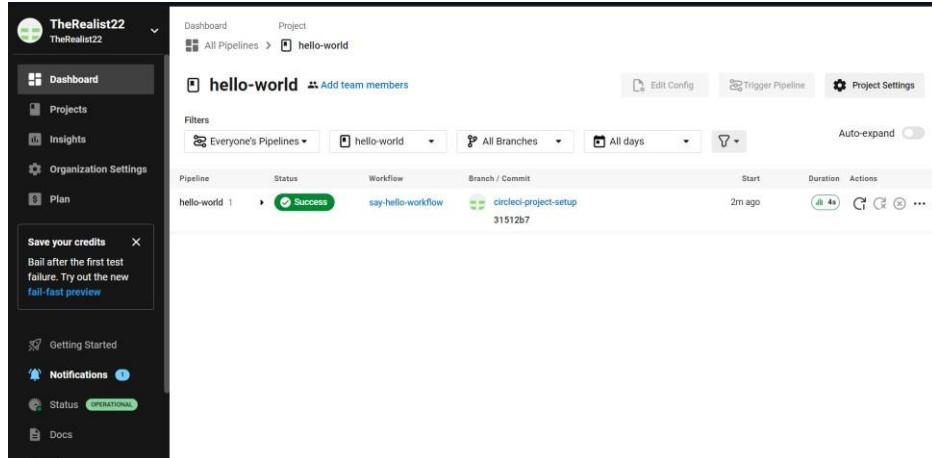
Step 2: Setup CircleCI

- Navigate to the CircleCI Projects page. If you created your new repository under an organization, you will need to select the organization name.
- You will be taken to the Projects dashboard. On the dashboard, select the project you want to set up (hello-world).
- 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.

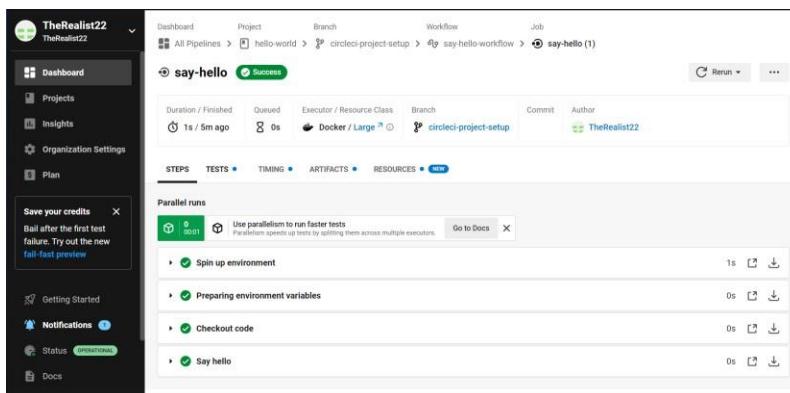
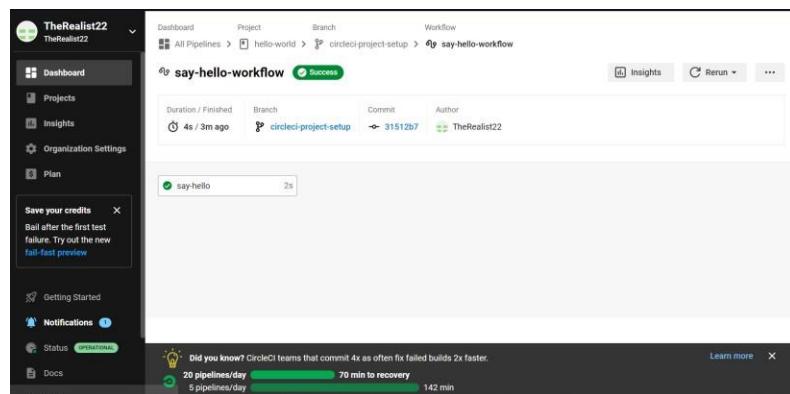


Step 3: First Pipeline

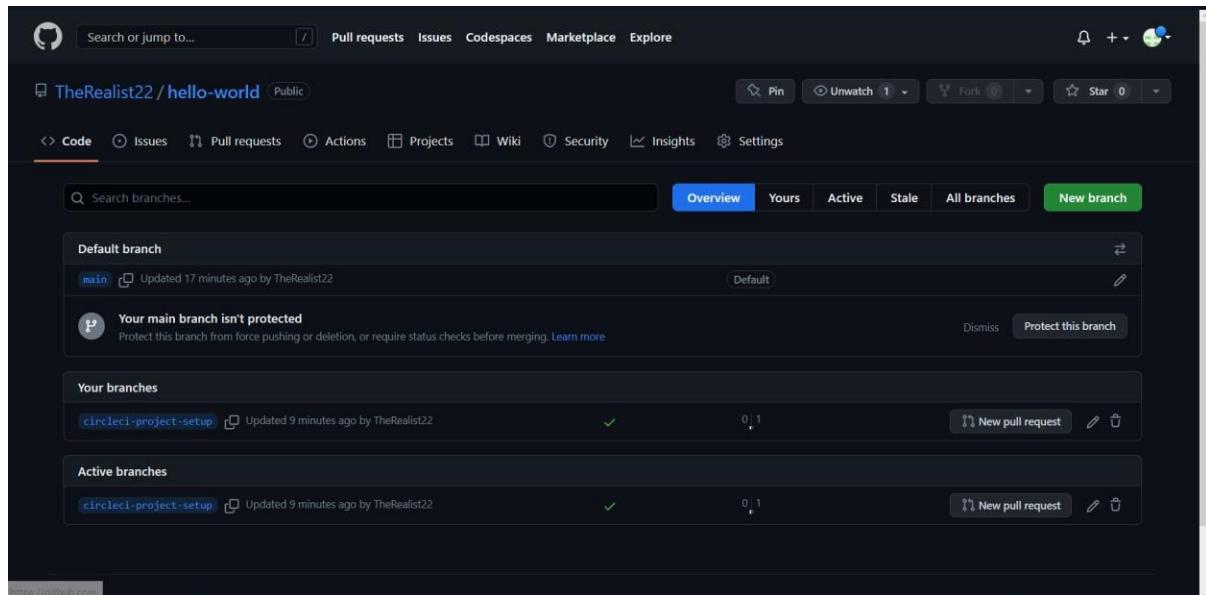
- On your project's pipeline page, click the green Success button, which brings you to the workflow that ran (say-hello-workflow).
- Within this workflow, the pipeline ran one job, called say-hello. Click say-hello to see the steps in this job:
 - a. Spin up environment
 - b. Preparing environment variables
 - 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.

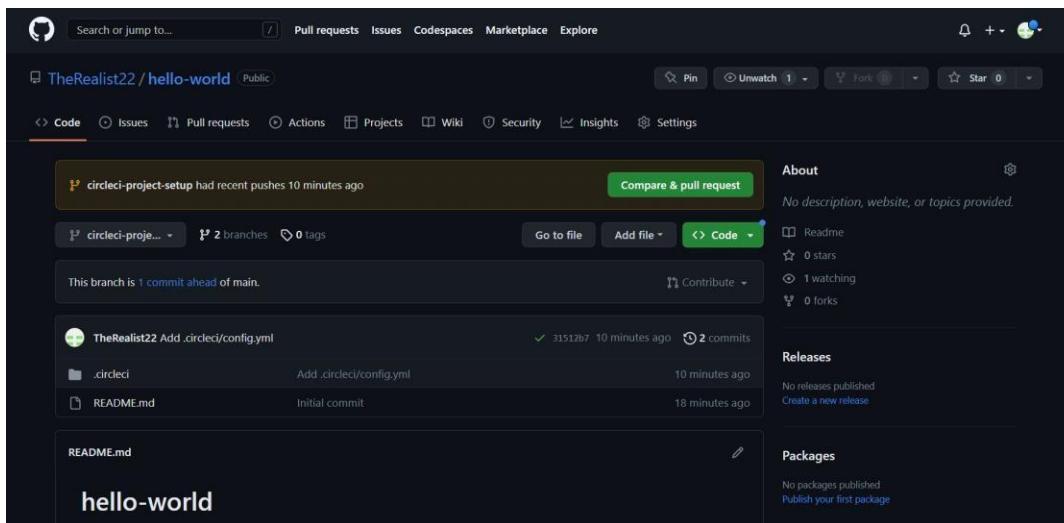


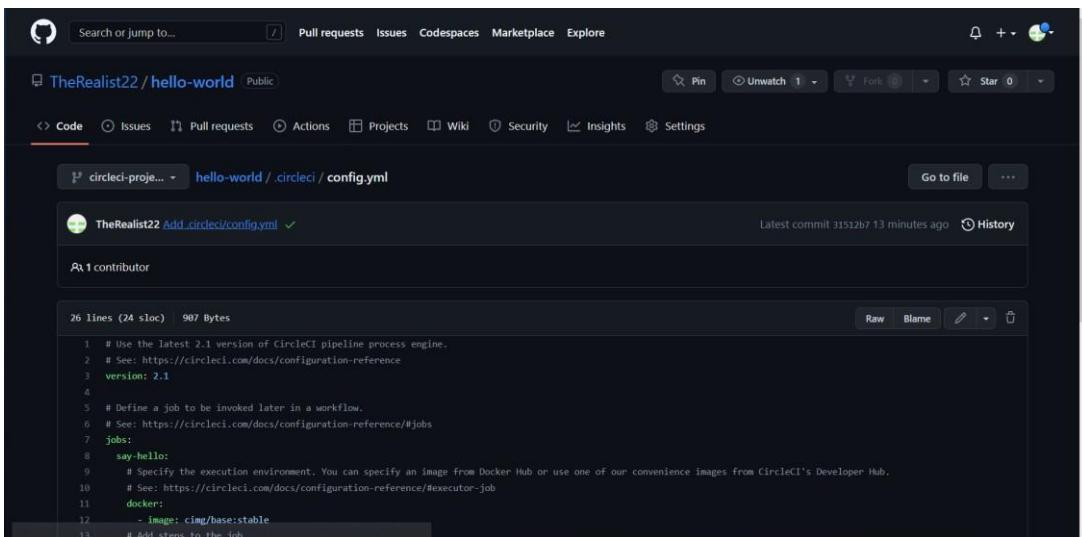
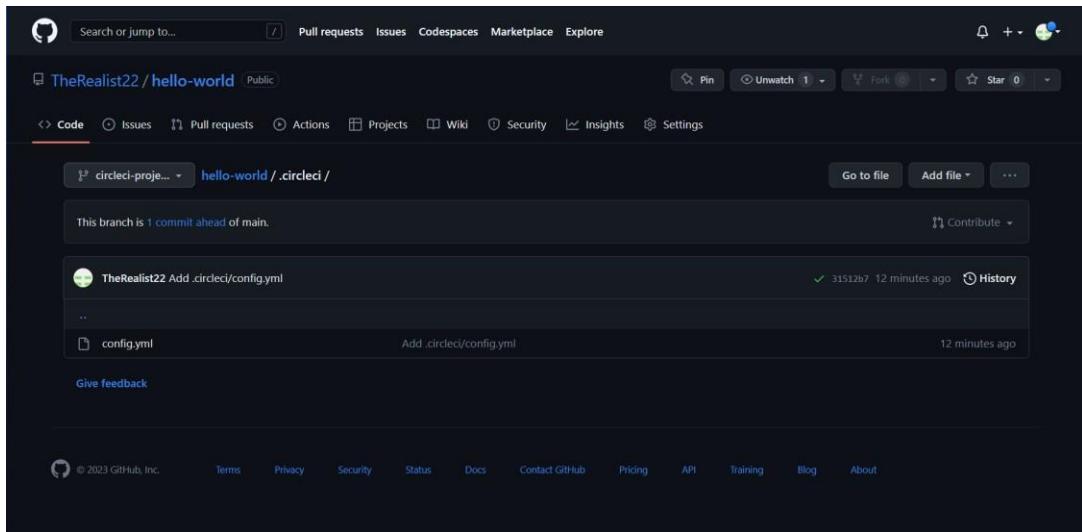
- Select Branch and option circleci-project-setup



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.

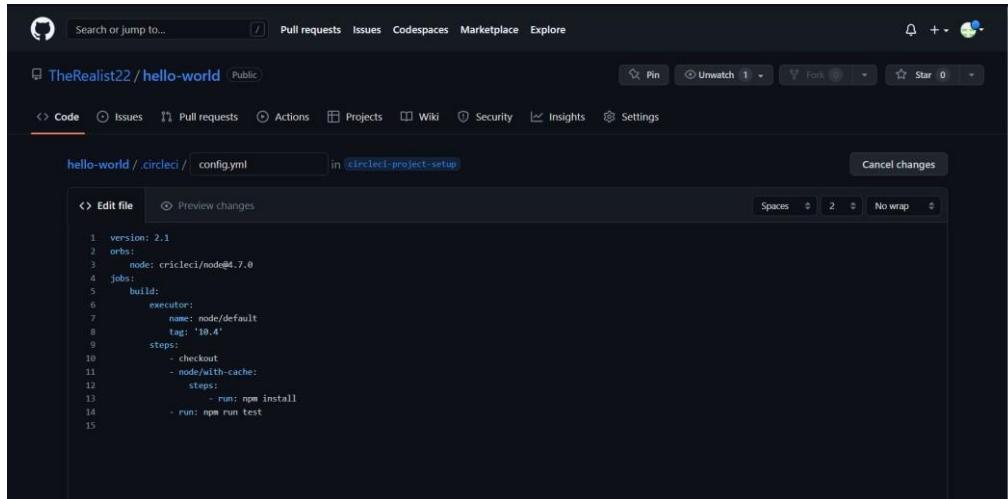




- Replace the existing code with new code:

```
version: 2.1
orbs:
  node: circleci/node@4.7.0
jobs:
  build:
    executor:
      name: node/default
      tag: '10.4'
    steps:
      - checkout
      - node/with-cache:
          steps:
            - run: npm install
            - run: npm run test
```

- The Github File Editor should look like this

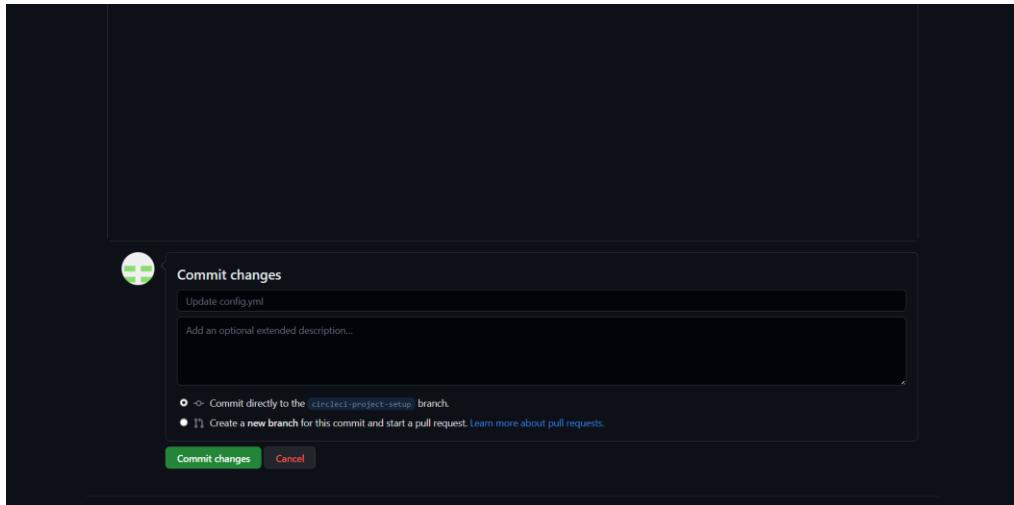


```

version: 2.1
orbs:
  node: circleci/node@1.7.0
jobs:
  build:
    executor:
      name: node/default
      tag: '10.4'
    steps:
      - checkout
      - node/with-cache:
          steps:
            - run: npm install
            - run: npm run test

```

- Scroll down and Commit your changes on GitHub



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

The screenshot shows the CircleCI web interface for a project named 'hello-world'. The pipeline 'circleci-project-setup' is currently active. Job 3 failed with errors related to workflow and build commands. Job 2 failed with an orb registry error. Job 1 succeeded. A sidebar on the left shows project navigation and status.

Step 5: Use Workflows

- You do not have to use orbs to use CircleCI. The following example details how to create a custom configuration that also uses the workflow feature of CircleCI.
- Take a moment and read the comments in the code block below. Then, to see workflows in action, edit your `.circleci/config.yml` file and copy and paste the following text into it.

Code:

```

version: 2
jobs:
  one:
    docker:
      - image: cimg/ruby:2.6.8

      steps:
        - checkout
        - run: echo "A first hello"
        - run: sleep 25

  two:
    docker:
      - image: cimg/ruby:3.0.2

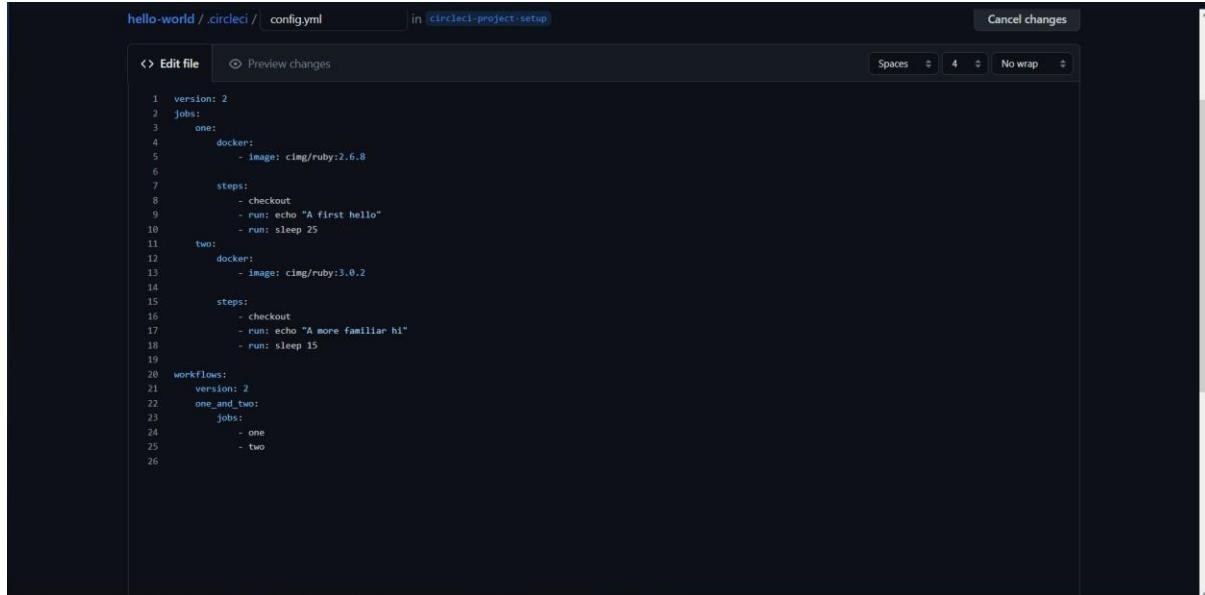
      steps:
        - checkout
        - run: echo "A more familiar hi"
        - run: sleep 15

workflows:
  version: 2
  one_and_two:
    jobs:

```

- one
- two

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



```

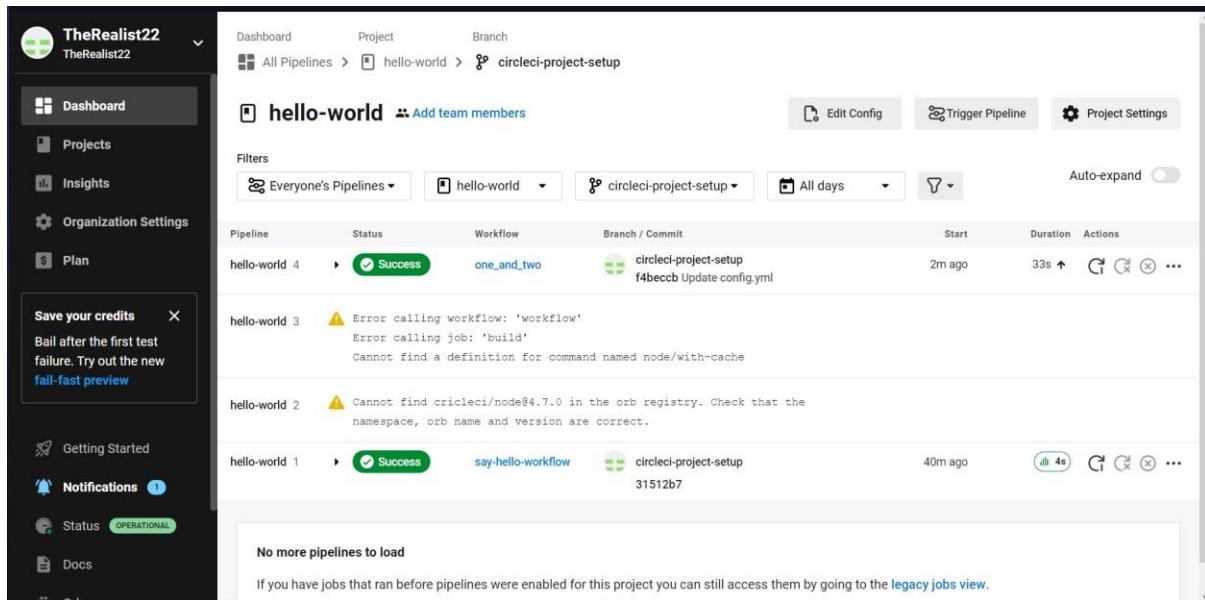
hello-world / .circleci / config.yml      in circleci-project-setup
Cancel changes

< > Edit file  Preview changes  Spaces  4  No wrap

1 version: 2
2 jobs:
3   one:
4     docker:
5       - image: cimg/ruby:2.6.8
6     steps:
7       - checkout
8       - run: echo "A first hello"
9       - run: sleep 25
10  two:
11    docker:
12      - image: cimg/ruby:3.0.2
13    steps:
14      - checkout
15      - run: echo "A more familiar hi"
16      - run: sleep 15
17  workflows:
18    version: 2
19    one_and_two:
20      jobs:
21        - one
22        - two
23
24
25
26

```

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



TheRealist22

Dashboard Project Branch

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

hello-world Add team members

Edit Config Trigger Pipeline Project Settings

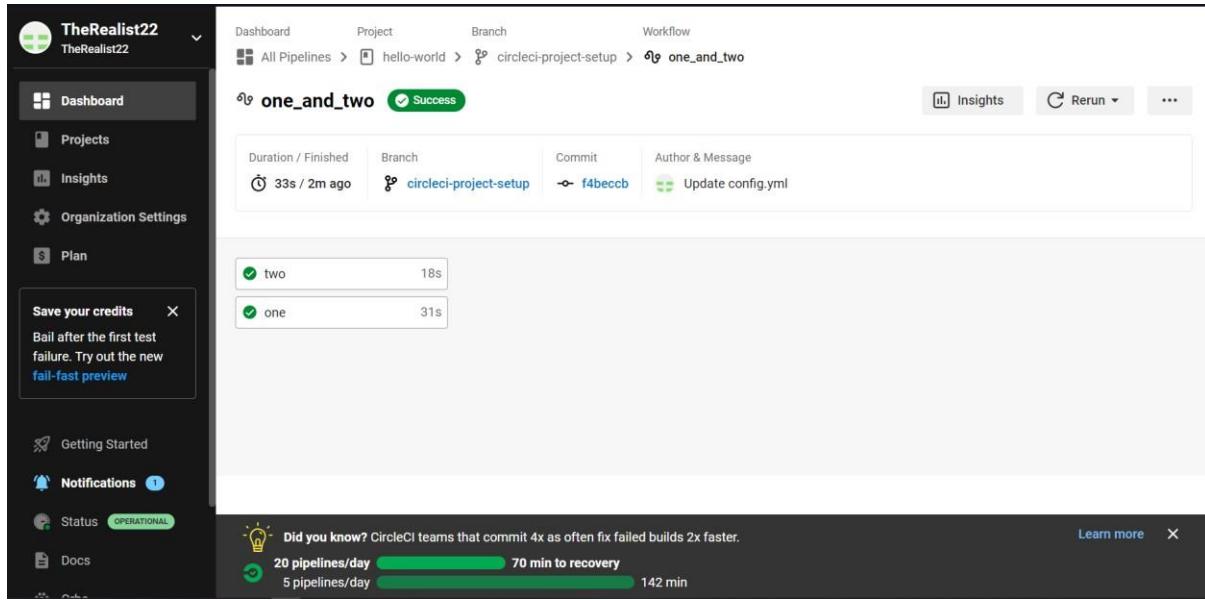
Filters Everyone's Pipelines hello-world circleci-project-setup All days

Pipeline	Status	Workflow	Branch / Commit	Start	Duration	Actions
hello-world 4	Success	one_and_two	circleci-project-setup f4beccb Update config.yml	2m ago	33s	
hello-world 3	Error		Error calling workflow: 'workflow' Error calling job: 'build' Cannot find a definition for command named node/with-cache			
hello-world 2	Error		Cannot find circleci/node@4.7.0 in the orb registry. Check that the namespace, orb name and version are correct.			
hello-world 1	Success	say-hello-workflow	circleci-project-setup 31512b7	40m ago	4s	

No more pipelines to load

If you have jobs that ran before pipelines were enabled for this project you can still access them by going to the [legacy jobs view](#).

- Click on **one_and_two** workflow.



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:

Code:

```
version: 2
jobs:
  one:
    docker:
      - image: cimg/ruby:3.0.2

      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

      steps:
        - checkout
        - run: echo "A more familiar hi"
```

```

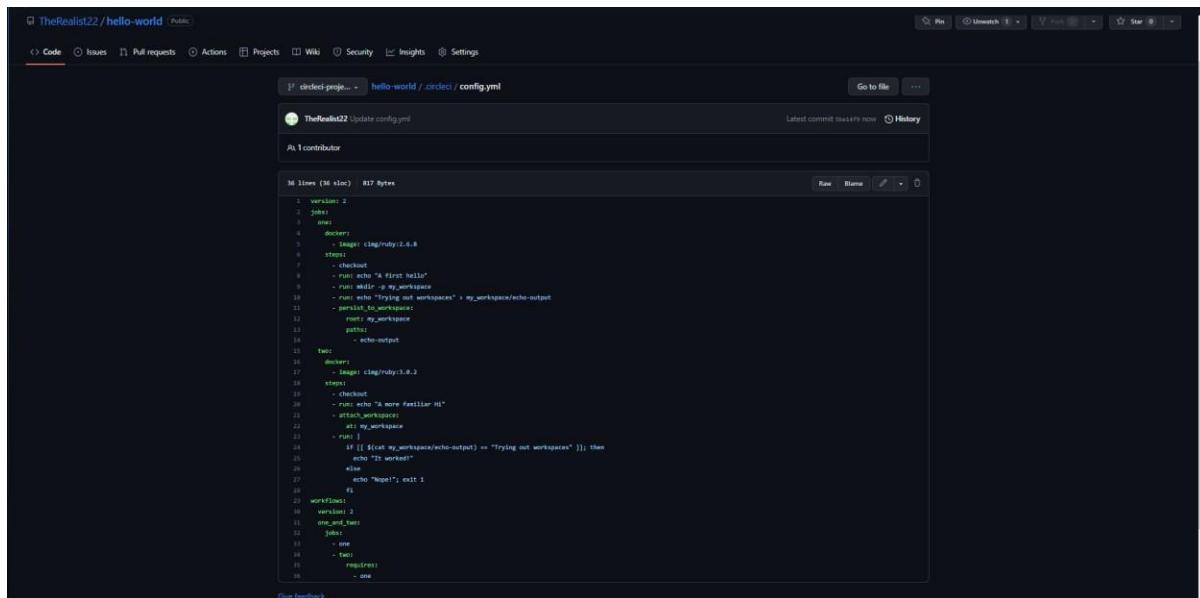
- attach_workspace:
  at: my_workspace

- run: |
    if [[ $(cat my_workspace/echo-output) == "Trying out
workspaces" ]]; then
      echo "It worked!"

workflows:
  version: 2
  one_and_two:
    jobs:
      - one
      - two:
        requires:
          - one

```

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

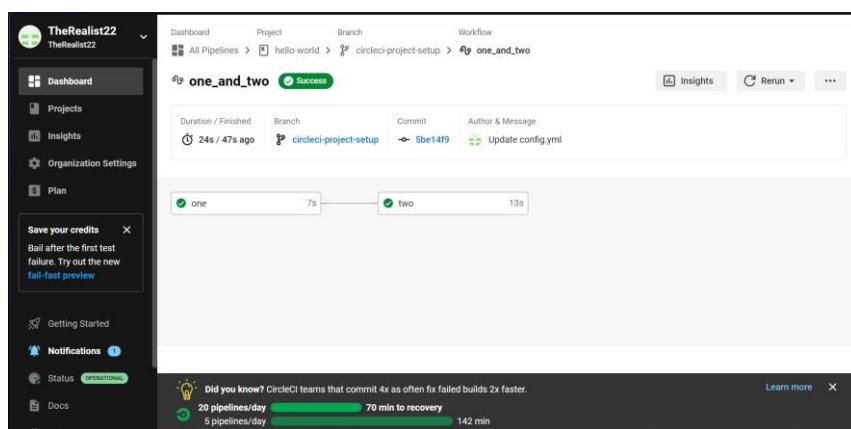


```

version: 2
jobs:
  one:
  two:
    steps:
      - checkout
      - run: echo "Hello"
      - run: echo "Trying out workspaces" > my_workspace/echo-output
      - parallel_to: workspace
        run: my_workspace/parallel.sh
      - echo-output
  workflows:
    version: 2
    one_and_two:
      jobs:
        - one
        - two:
          requires:
            - one

```

- Finally, your workflow with the jobs running should look like this



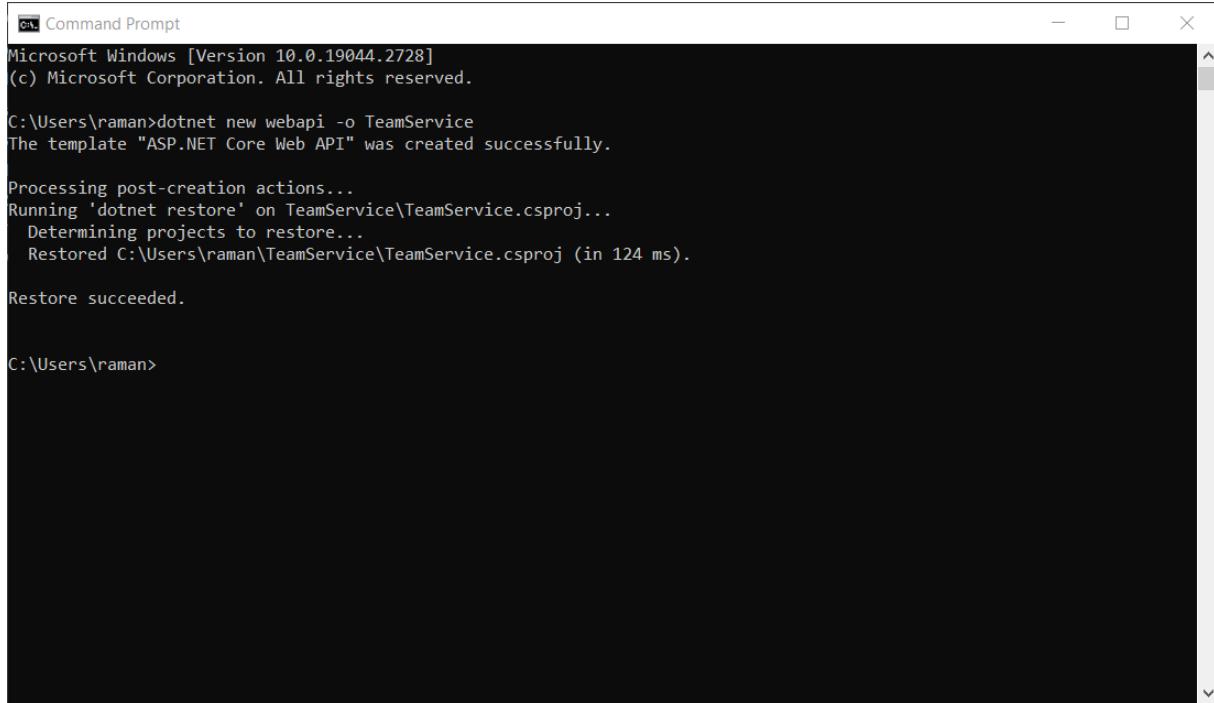
Practical 6

Aim: Creating Microservice with ASP.NET Core.

Writeup:

Step 1: Create new project.

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



```
Microsoft Windows [Version 10.0.19044.2728]
(c) Microsoft Corporation. All rights reserved.

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

Processing post-creation actions...
Running 'dotnet restore' on TeamService\TeamService.csproj...
Determining projects to restore...
Restored C:\Users\raman\TeamService\TeamService.csproj (in 124 ms).

Restore succeeded.

C:\Users\raman>
```

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

Step 3: Add new files as follows:

[A]: Add Member.cs to “D:\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 “D:\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 “D:\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 “D:\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)
```

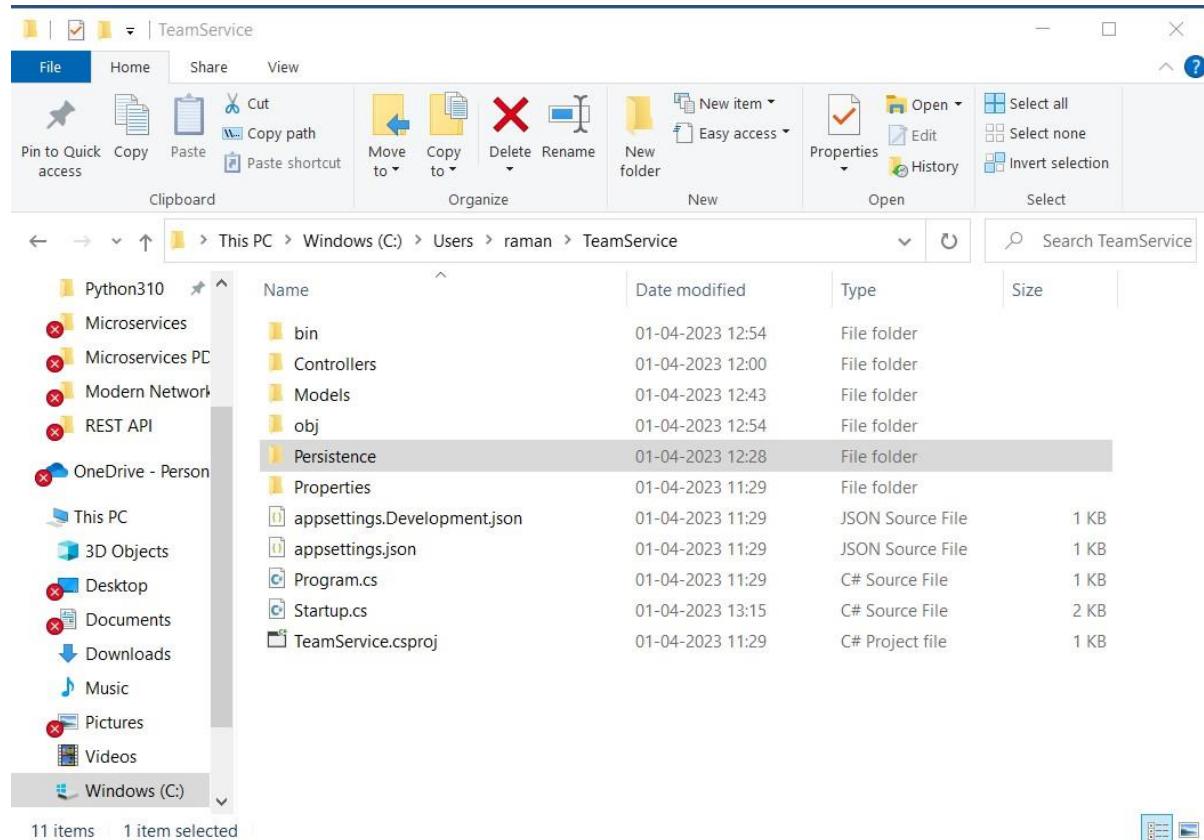
```

    {
        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 “D:\TeamService\Persistence”.



Step 5: Add file ITeamReposiroty.cs in “D:\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 “D:\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.

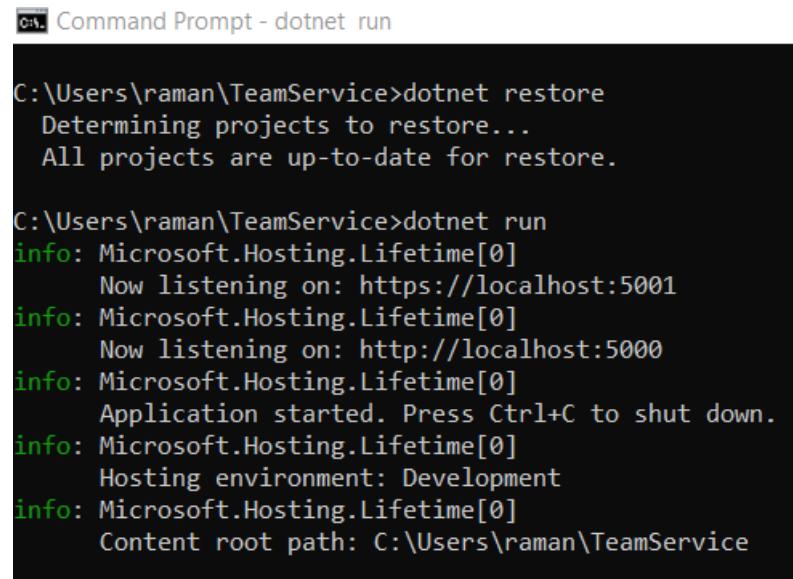
```
services.AddScoped<ITeamRepository, MemoryTeamRepository>();
```

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



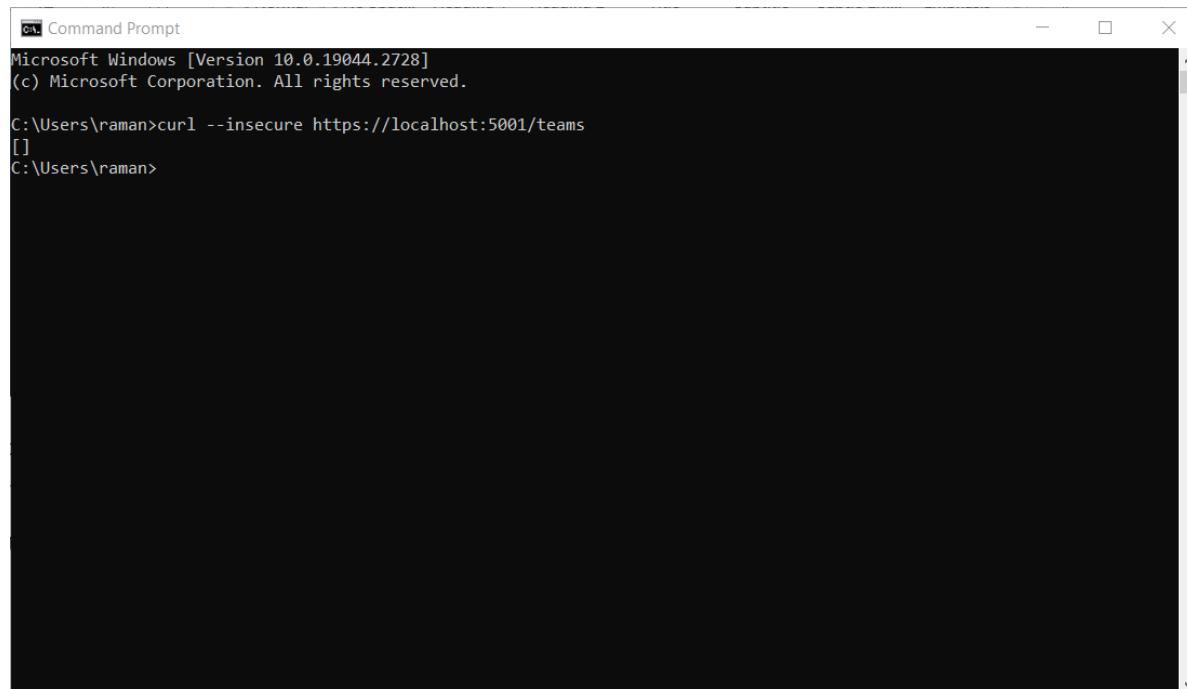
```
C:\Users\raman\TeamService>dotnet restore
Determining projects to restore...
All projects are up-to-date for restore.

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

Step 10: On command prompt 2

Command: To get all teams

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



```
Microsoft Windows [Version 10.0.19044.2728]
(c) Microsoft Corporation. All rights reserved.

C:\Users\raman>curl --insecure https://localhost:5001/teams
[]

C:\Users\raman>
```

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>

```
cmd Command Prompt
C:\Users\raman>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\raman>
```

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>

```
cmd Command Prompt
C:\Users\raman>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\raman>
```

Step 13: On command prompt 2

Command : To get all teams

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

```
cmd Command Prompt
C:\Users\raman>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": []}]
C:\Users\raman>
```

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>

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

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>

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

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

Step 17: Confirm with get all teams now it shows only one team (first one is deleted)Command: curl –insecure <https://localhost:5001/teams>

```
C:\ Command Prompt
C:\Users\raman>curl -insecure https://localhost:5001/teams
HTTP/1.1 200 OK
Date: Sat, 01 Apr 2023 08:43:18 GMT
Content-Type: application/json; charset=utf-8
Server: Kestrel
Transfer-Encoding: chunked

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

Step 18: Adding Members to the team.

Firstly, recreate the team which has been deleted by running the previous command for create in team.

Within the created team we will add new properties for “members” within the team with a specific name and id. (Done using Postman)

The screenshot shows the Postman application interface. The left sidebar displays collections like 'mfas' and 'Practical 6'. Under 'Practical 6', there are several requests: 'Get Values', 'Create a New Team', 'Get a Particular Value Through...', 'PUT + +*', 'New Request', and 'Create a New Member'. The 'Create a New Member' request is selected. The main panel shows a POST request to 'localhost:5001/teams/e12baa63-d511-417e-9e54-7aab04212345/members/'. The 'Body' tab is selected, showing a JSON payload:

```

1   {
2     "firstName": "Ram",
3     "lastName": "Rao",
4     "id": "e12baa63-d511-417e-9e54-7aab04211111"
5 }
```

The response status is '201 Created' with a size of 357 B. The response body is:

```

1   {
2     "teamID": "e12baa63-d511-417e-9e54-7aab04212345",
3     "memberID": "e12baa63-d511-417e-9e54-7aab04211111"
4 }
```

- Confirm the creation of the member.

The screenshot shows the Postman application interface. On the left, the 'Scratch Pad' sidebar lists several collections: 'Practical 6' (selected), 'Practical 5', 'Practical 4', 'Practical 3', 'Practical 2', and 'Practical 1'. The main workspace displays a 'GET / Get Values' request. The URL is set to 'localhost:5000/teams'. The 'Body' tab shows the response in pretty-printed JSON format:

```

1  {
2   "name": "Ramanuj",
3   "id": "e12baa63-d511-417e-9e54-7aab04212345",
4   "members": [
5     {
6       "id": "e12baa63-d511-417e-9e54-7aab04211111",
7       "firstname": "Ram",
8       "lastName": "Rao"
9     }
10   ]
11 }
12
13
  
```

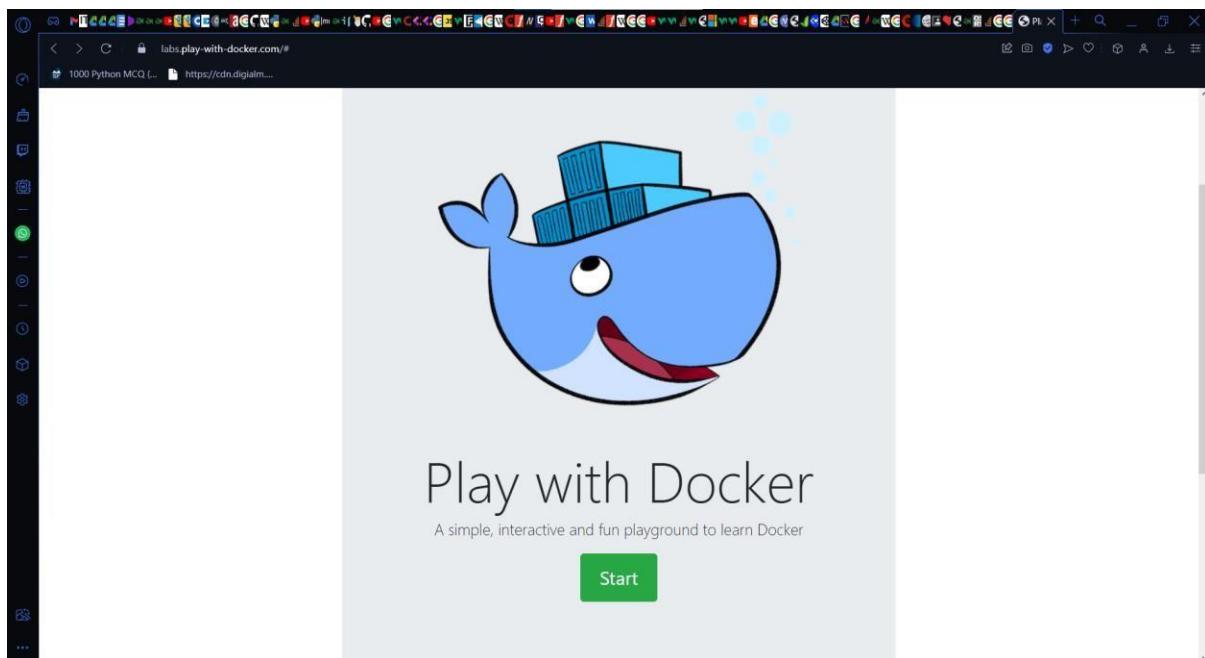
The status bar at the bottom indicates 'Status: 200 OK'.

Practical 7

Aim: Creating Backing Service with Docker.

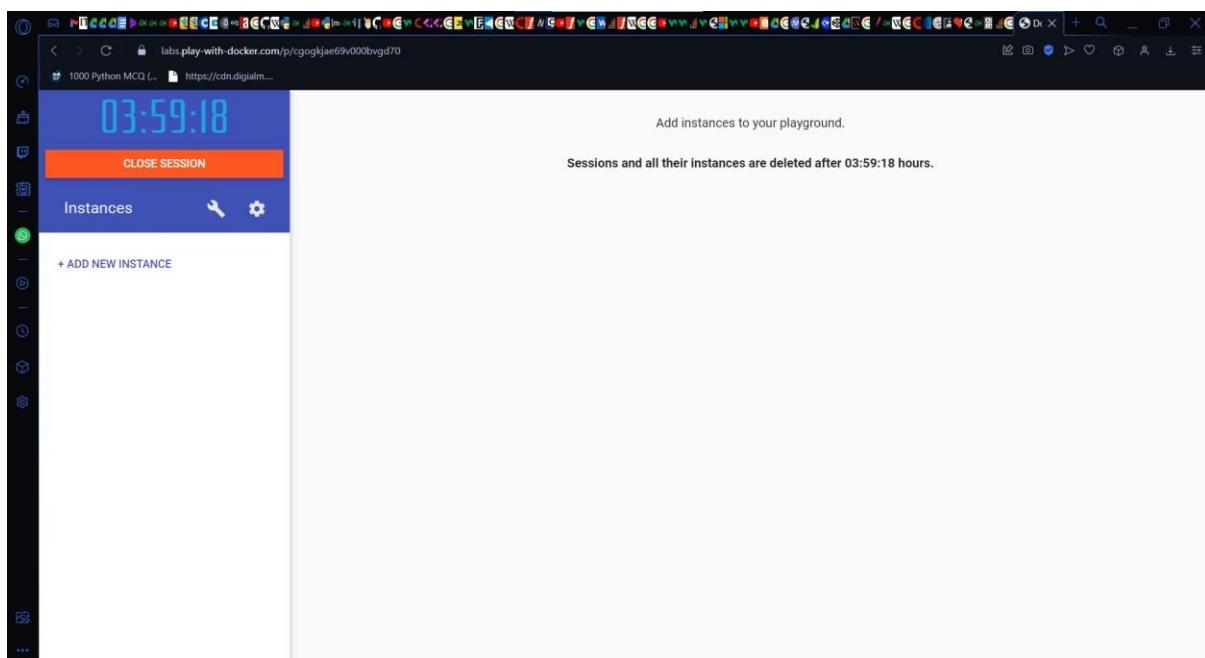
Writeup:

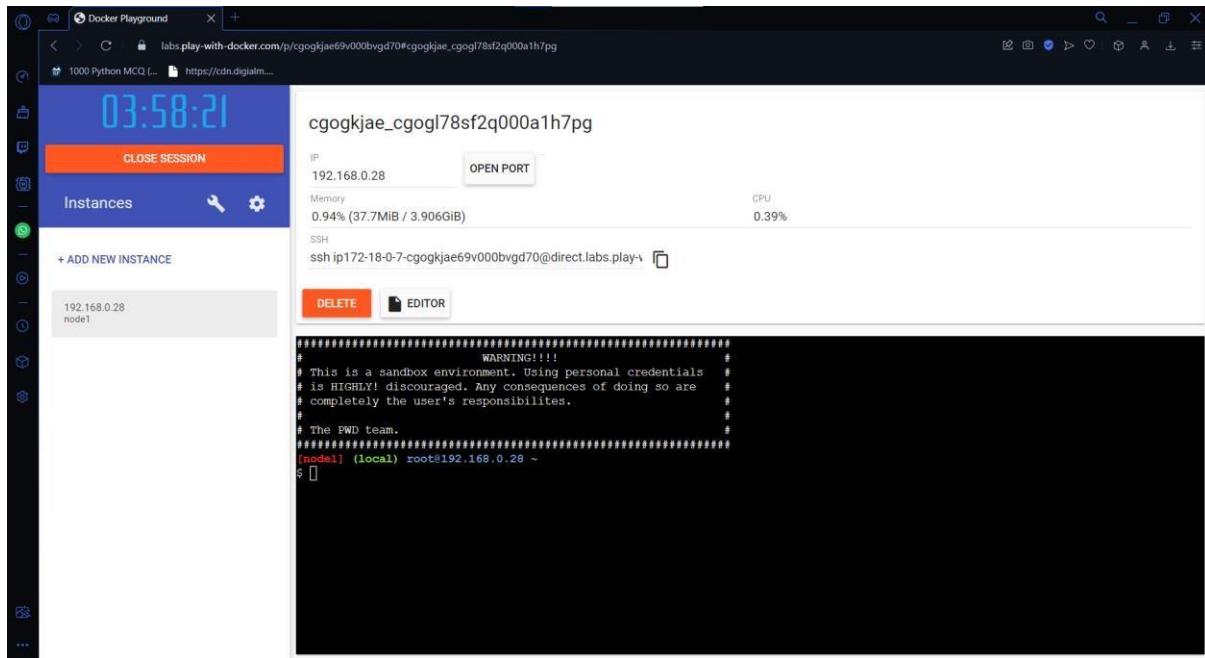
Step 1: Create docker hub login first to use it in play with docker.



- Click on Start

Step 2: Click on Add new instance



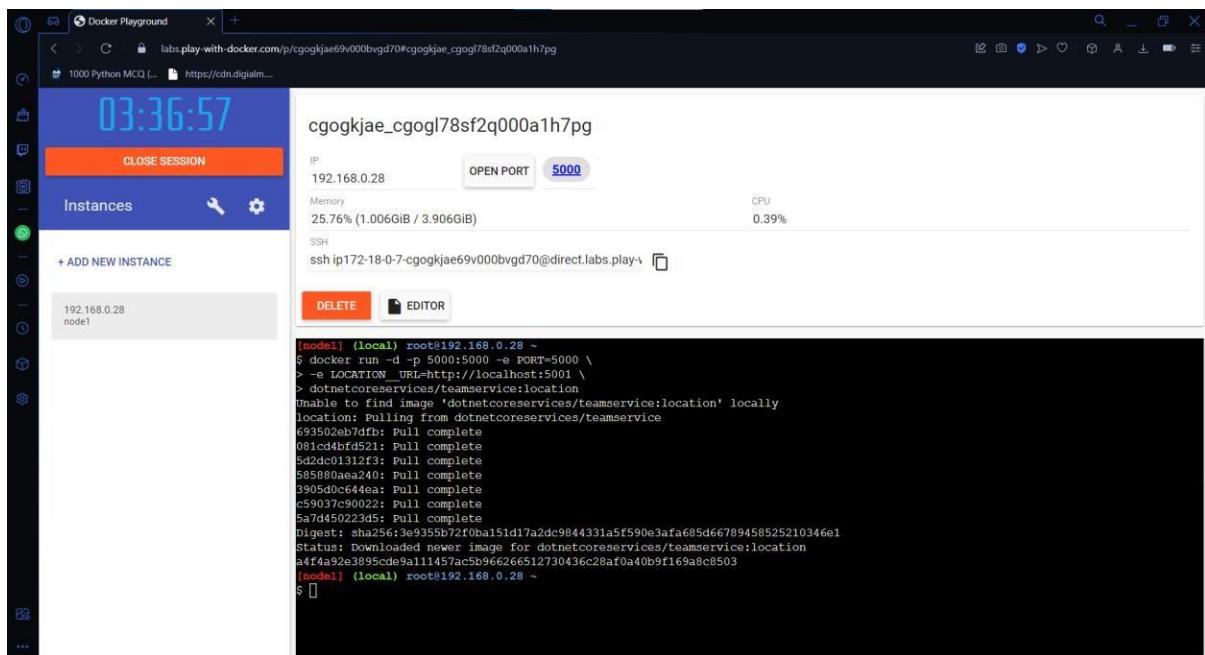


Step 3: Start typing the following command

Command: To run TeamService

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

Output:



Command: To run Location Service

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

Output:

```
(node1) (local) root@192.168.0.28 ~
$ docker run -d -p 5001:5001 -e PORT=5001 \
> dotnetcoreservices/locationservice:nodb
Unable to find image 'dotnetcoreservices/locationservice:nodb' locally
node: Pulling from dotnetcoreservices/locationservice
693502eb7dfb: Already exists
081cd4b4fd52: Already exists
5d2dc01312f3: Already exists
585880ea240: Already exists
3905d0c64e4a: Already exists
c59037c90022: Already exists
dbc03883a4ca: Pull complete
Digest: sha256:5f7aca33c5e2117e04f58a59e0cf96fd20d5cbf2cf66c3cd708118d573255168
Status: Downloaded newer image for dotnetcoreservices/locationservice:nodb
56e34fa06c74136fa84170b901b7d9dbd80930fba14adf348be7276e100b644
[node1] (local) root@192.168.0.28 ~
$ [ ]
```

Command: To check running images in docker

docker images

Output:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
dotnetcoreservices/teamservice	location	b27d0de8f2de	6 years ago	886MB
dotnetcoreservices/locationservice	nodb	03339f0ea9dd	6 years ago	883MB

```
(node1) (local) root@192.168.0.28 ~
$ docker images
REPOSITORY          TAG      IMAGE ID   CREATED        SIZE
dotnetcoreservices/teamservice   location  b27d0de8f2de  6 years ago  886MB
dotnetcoreservices/locationservice   nodb    03339f0ea9dd  6 years ago  883MB
[node1] (local) root@192.168.0.28 ~
$ [ ]
```

Command: To create a new team

Output:

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

```
03:15:20
CLOSE SESSION
cgogkjae_cgogl78sf2q000a1h7pg
IP: 192.168.0.28 OPEN PORT 5001 5000
Memory: 21.74% (869.4MiB / 3.906GiB) CPU: 0.48%
SSH: ssh ip172-18-0-7-cgogkjae69v000bvgd70@direct.labs.play-with-docker.com
DELETE EDITOR

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

Command: To confirm if member is added

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

Output:

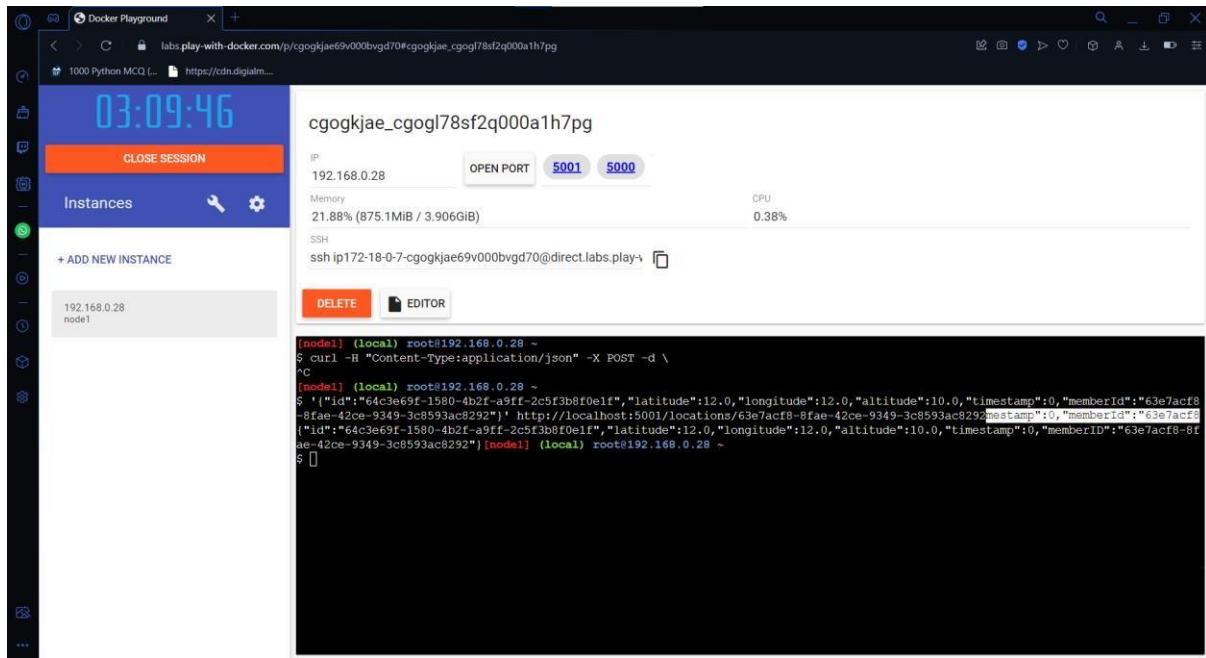
```
03:14:00
CLOSE SESSION
cgogkjae_cgogl78sf2q000a1h7pg
IP: 192.168.0.28 OPEN PORT 5001 5000
Memory: 21.76% (870.4MiB / 3.906GiB) CPU: 0.70%
SSH: ssh ip172-18-0-7-cgogkjae69v000bvgd70@direct.labs.play-with-docker.com
DELETE EDITOR

(node1) 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) root@192.168.0.28 ~
$
```

Command: To add location for member

```
curl -H "Content-Type:application/json" -X POST -d \
'{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0elf","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
```

Output:

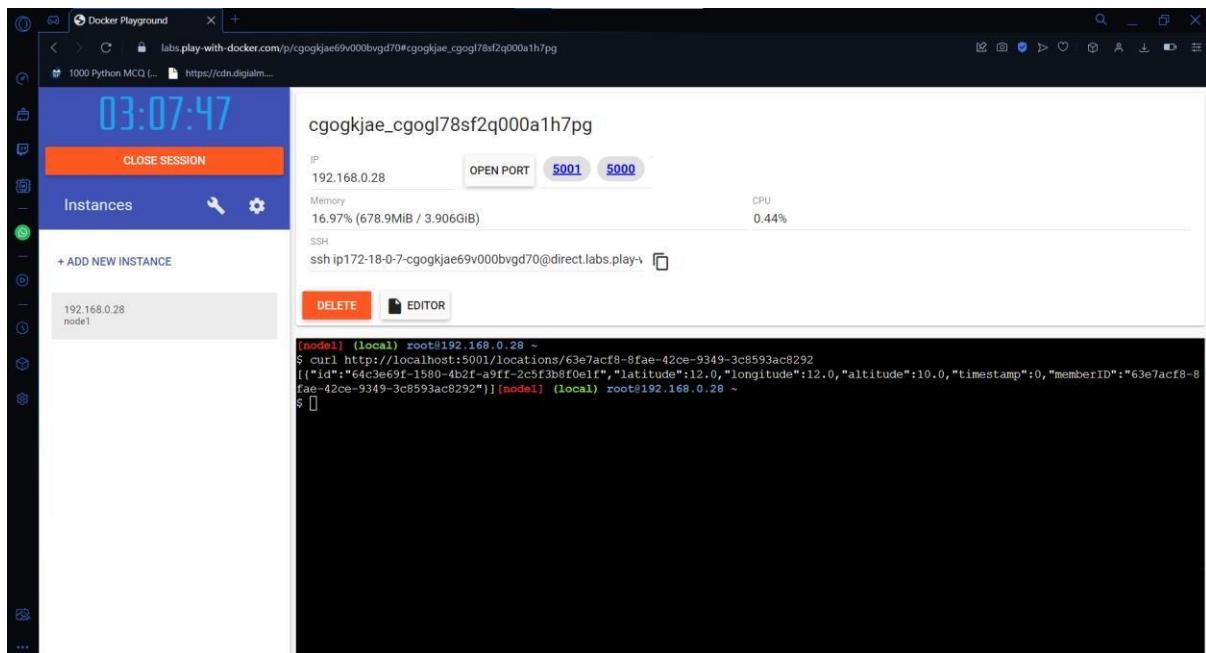


The screenshot shows a Docker Playground interface. On the left, there's a sidebar with various icons. In the center, there's a terminal window titled 'cgogkjae_cgogl78sf2q000a1h7pg'. The terminal shows the command being run and its output:

```
[node1] (local) root@192.168.0.28 ~
$ curl -H "Content-Type:application/json" -X POST -d \
'{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0elf","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-2c5f3b8f0elf","latitude":12.0,"longitude":12.0,"altitude":10.0,"timestamp":0,"memberId":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}]
```

Command: To confirm location is added in member

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



The screenshot shows a Docker Playground interface. The terminal window shows the result of the previous command:

```
[node1] (local) root@192.168.0.28 ~
$ curl http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292
[{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0elf","latitude":12.0,"longitude":12.0,"altitude":10.0,"timestamp":0,"memberId":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}]
```

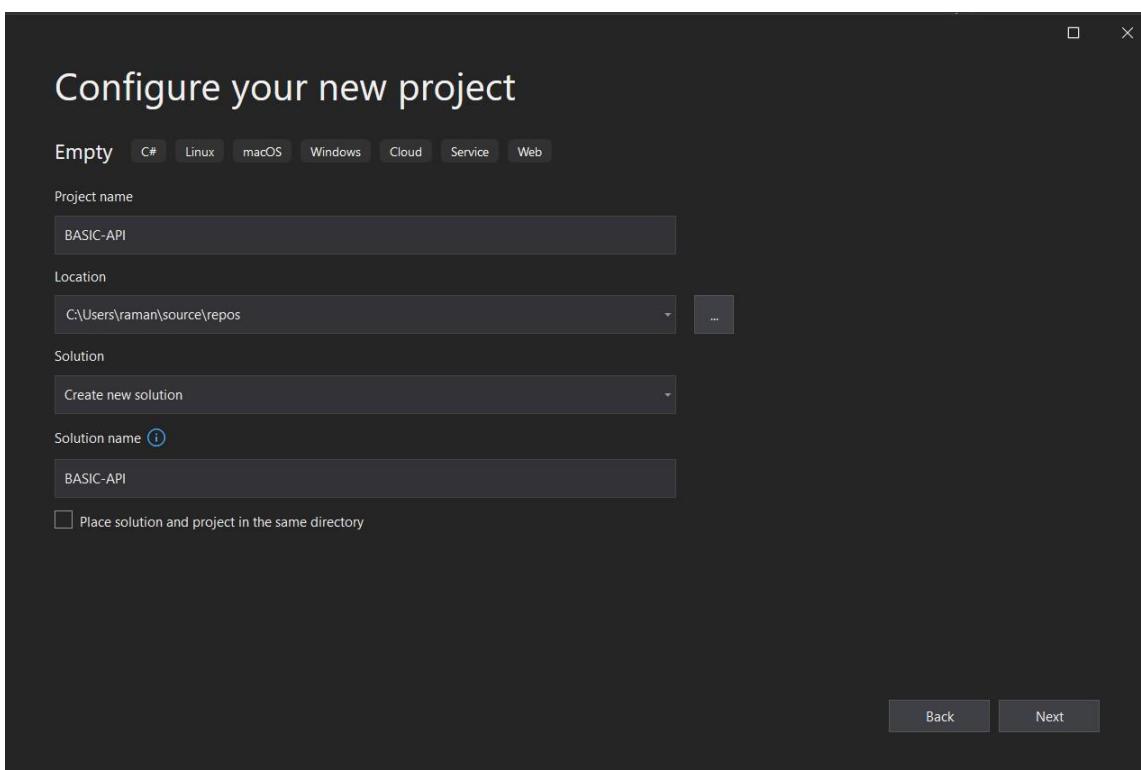
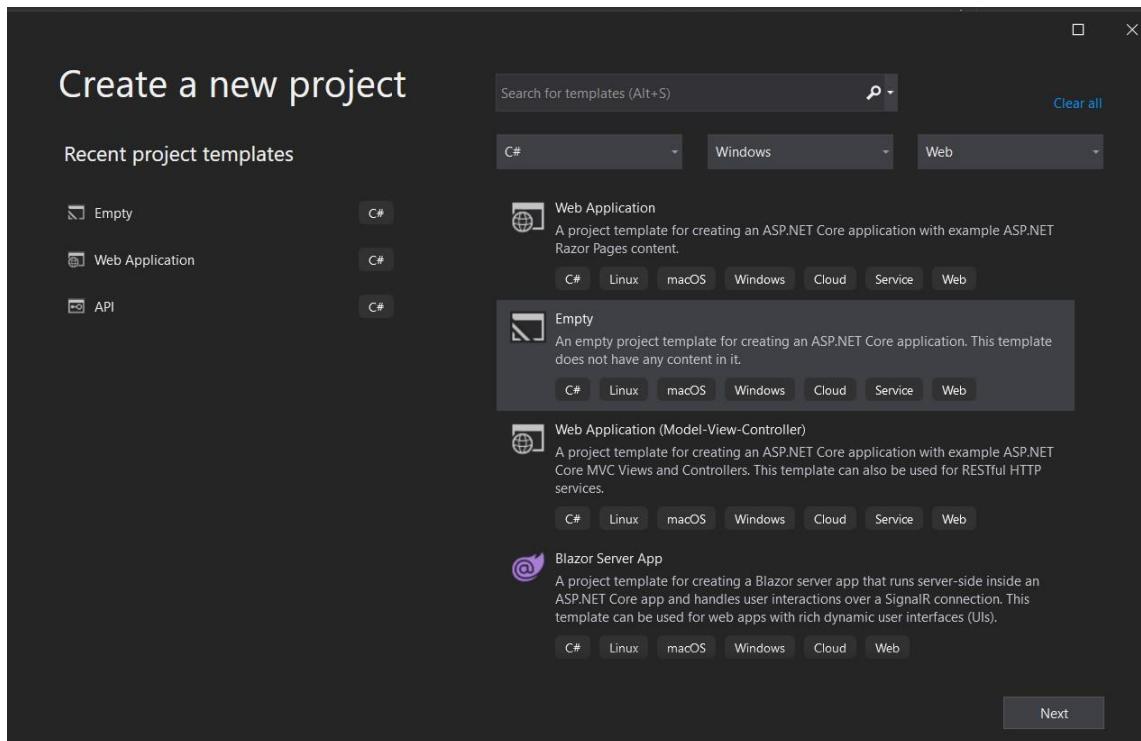
Practical 8

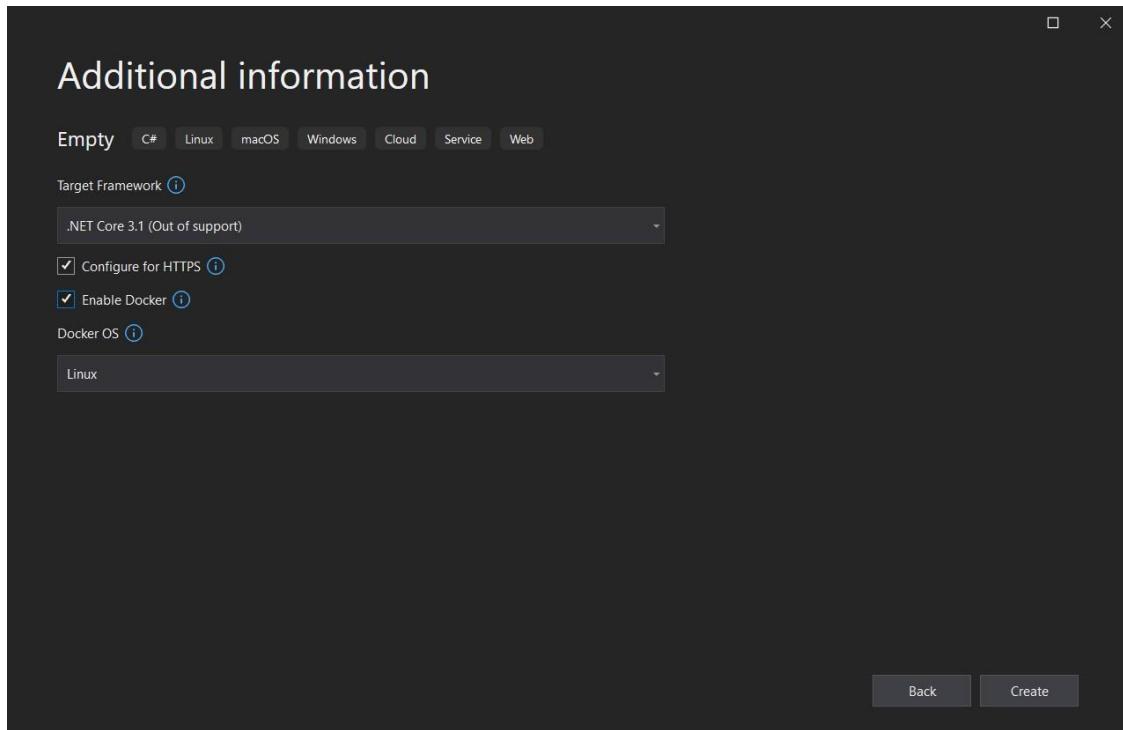
Aim: Building real-time Microservice with ASP.NET Core.

Writeup:

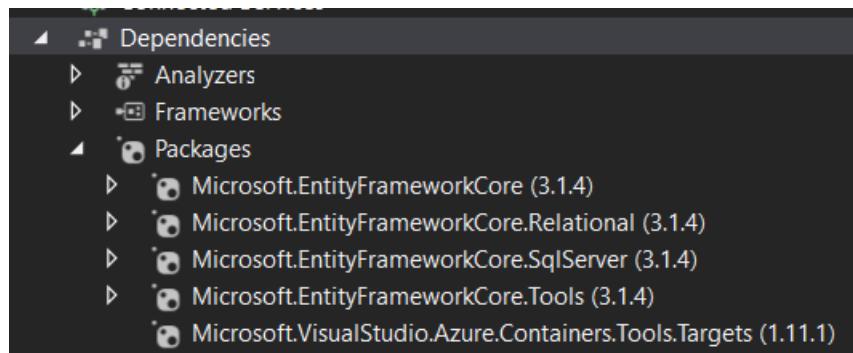
Requirement:

- Microsoft Visual Studio 2019 or higher
- ASP.Net Core 3.1

Step 1: Create and Configure a new empty project



Step 2: Make sure to add these packages via NuGet Package Manager.



Step 3: Configure the Startup.cs file

```
namespace BASIC_API
{
    2 references
    public class Startup
    {
        4 references
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        2 references
        public IConfiguration Configuration { get; }

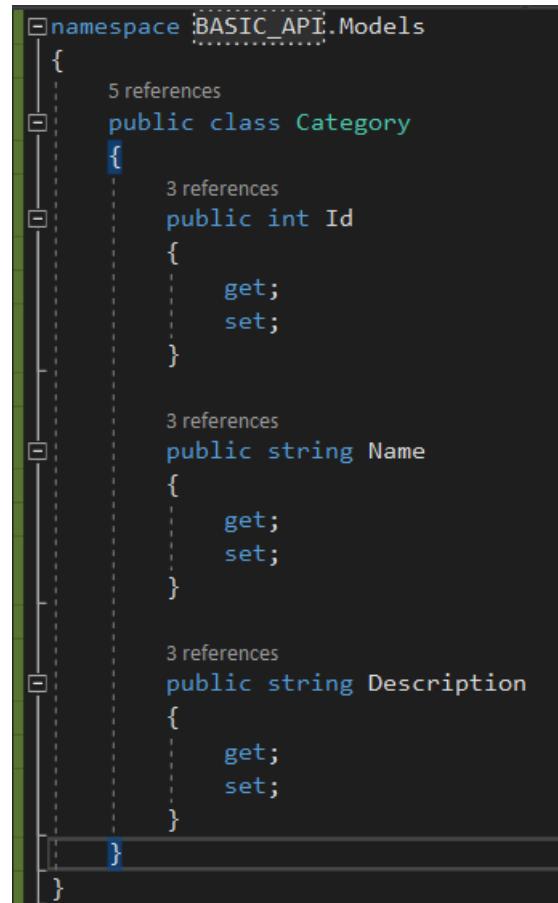
        // This method gets called by the runtime. Use this method to add services to the container.
        0 references
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_3_0);
            services.AddDbContext<ProductContext>(o => o.UseSqlServer(Configuration.GetConnectionString("ProductDB")));
            services.AddTransient<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
        {
            // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
            app.UseHsts();
        }

        app.UseRouting();
        app.UseHttpsRedirection();

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

Step 4: Create the Category Model, which will eventually be the Category Table.

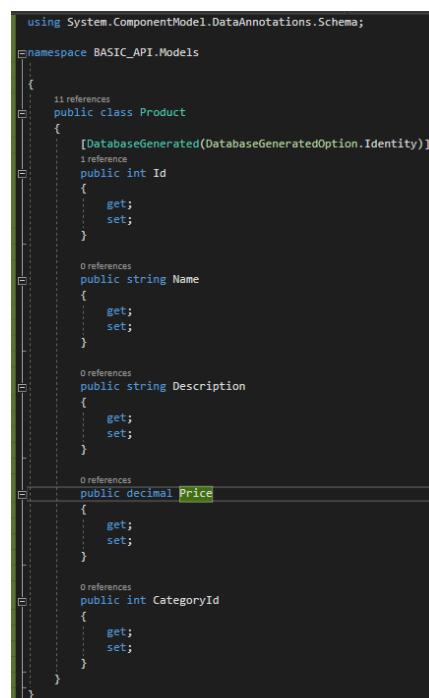


```
namespace BASIC_API.Models
{
    public class Category
    {
        public int Id
        {
            get;
            set;
        }

        public string Name
        {
            get;
            set;
        }

        public string Description
        {
            get;
            set;
        }
    }
}
```

Step 5: Create the Product Model which will eventually be the Product Table.



```
using System.ComponentModel.DataAnnotations.Schema;

namespace BASIC_API.Models
{
    public class Product
    {
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id
        {
            get;
            set;
        }

        public string Name
        {
            get;
            set;
        }

        public string Description
        {
            get;
            set;
        }

        public decimal Price
        {
            get;
            set;
        }

        public int CategoryId
        {
            get;
            set;
        }
    }
}
```

Step 6: Create a ProductRepository for all our operations.

```

using BASIC_API.DBContexts;
using BASIC_API.Models;
using BASIC_API.Repository;
using Microsoft.EntityFrameworkCore;

namespace BASIC_API.Repository
{
    2 references
    public class ProductRepository : IProductRepository
    {
        private readonly ProductContext _dbContext;

        0 references
        public ProductRepository(ProductContext dbContext)
        {
            _dbContext = dbContext;
        }

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

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

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

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

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

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

```

Step 7: Create an interface IProductRepository to access the ProductRepository.

```

using BASIC_API.Models;
using System.Collections.Generic;

namespace BASIC_API.Repository
{
    4 references
    public interface IProductRepository
    {
        2 references
        IEnumerable<Product> GetProducts();
        2 references
        Product GetProductByID(int product);
        2 references
        void InsertProduct(Product product);
        2 references
        void DeleteProduct(int productId);
        2 references
        void UpdateProduct(Product product);
        4 references
        void Save();
    }
}

```

Step 8: Create a ProductContext to build the model for the database; this will be a Code First Database Approach.

```
using Microsoft.EntityFrameworkCore;
using BASIC_API.Models;

namespace BASIC_API.DBContexts
{
    public class ProductContext : DbContext
    {
        public DbSet<Product> Products { get; set; }

        public DbSet<Category> Categories { get; set; }

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

Step 9: Add a ProductController to get an endpoint for the APIs.

```
using Microsoft.AspNetCore.Mvc;
using System.Transactions;
using BASIC_API.Models;
using BASIC_API.Repository;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=399860

namespace BASIC_API.Controllers
{
    [Produces("application/json")]
    [Route("api/Product")]
    [ApiController]
    public class ProductController : ControllerBase
    {
        private readonly IProductRepository _productRepository;

        public ProductController(IProductRepository productRepository)
        {
            _productRepository = productRepository;
        }

        //GET: api/Product
        [HttpGet]
        public IActionResult Get()
        {
            var products = _productRepository.GetProducts();
            return new OkObjectResult(products);
        }

        //GET: api/Product/{id}
        [HttpGet("{id}", Name = "Get")]
        public IActionResult Get(int id)
        {
            var product = _productRepository.GetProductByID(id);
            return new OkObjectResult(product);
        }

        //POST: api/Product
        [HttpPost]
        public IActionResult Post([FromBody] Product product)
        {
            using (var scope = new TransactionScope())
            {
                _productRepository.InsertProduct(product);
                scope.Complete();
                return CreatedAtAction(nameof(Get), new { id = product.Id }, product);
            }
        }

        //PUT: api/Product/{id}
        [HttpPut]
        public IActionResult Put([FromBody] Product product)
        {
            if (product == null)
            {
                using (var scope = new TransactionScope())
                {
                    _productRepository.UpdateProduct(product);
                    scope.Complete();
                    return new OkResult();
                }
            }
            return new NoContentResult();
        }

        //DELETE: api/Product/{id}
        [HttpDelete("{id}")]
        public IActionResult Delete(int id)
        {
            _productRepository.DeleteProduct(id);
            return new OkResult();
        }
    }
}
```

Step 10: Add the connection string settings to connect the project to the SQLSERVER database. Once added run **add-migration** command to run the migration.



```
1 {
2   "Logging": {
3     "LogLevel": {
4       "Default": "Information",
5       "Microsoft": "Warning",
6       "Microsoft.Hosting.Lifetime": "Information"
7     }
8   },
9   "AllowedHosts": "*",
10  "ConnectionStrings": {
11    "ProductDB": "Server=LAPTOP-5A4DVJ0D\\MYSQLSERVER2022;Database=ProductDB;Trusted_Connection=True;MultipleActiveResultSets=true;"
12  }
13 }
```

```
PM> add-migration InitialCreate -verbose
Using project 'BASIC-API'.
Using startup project 'BASIC-API'.
Build started...
Build succeeded.
```

Step 11: Run the **update-database** command to reflect the model changes to the database.

```
PM> update-database -verbose
Using project 'BASIC-API'.
Using startup project 'BASIC-API'.
Build started...
Build succeeded.
```

Step 12: Try to run the project, you will see that a browser window opens with an empty list.



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

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** <https://localhost:44317/api/Product/>
- Body (JSON):**

```

1  {
2   ... "Name" : "HP Pavillion Gaming 15",
3   ... "Description" : "15.2 inch IPS Display with 16GB DDR4 RAM and 256GB SSD with Nvidia GTX 1650",
4   ... "Price" : 770.00,
5   ... "CategoryId" : 1
6 }
```
- Response Headers:** 201 Created, 153 ms, 398 B
- Response Body (Pretty JSON):**

```

1  {
2   "id": 9,
3   "name": "HP Pavillion Gaming 15",
4   "description": "15.2 inch IPS Display with 16GB DDR4 RAM and 256GB SSD with Nvidia GTX 1650",
5   "price": 770.00,
6   "categoryId": 1
7 }
```

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

The screenshot shows the SSMS interface with the following details:

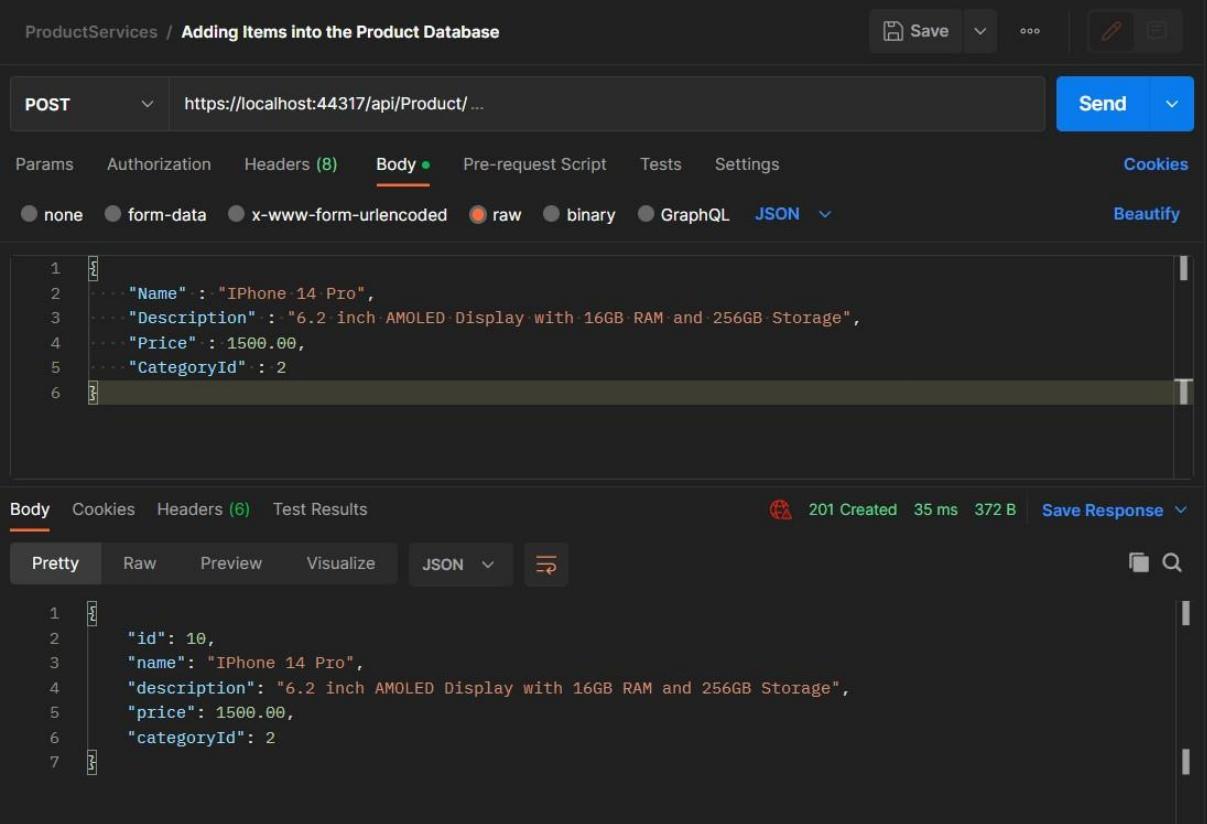
- Query:**

```

***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP (1000) [Id]
    ,[Name]
    ,[Description]
    ,[Price]
    ,[CategoryId]
FROM [ProductDB].[dbo].[Products]
```
- Results:**

	Id	Name	Description	Price	CategoryId
1	9	HP Pavillion Gaming 15	15.2 inch IPS Display with 16GB DDR4 RAM and 256G...	770.00	1

Step 15: Add another Product to test.



The screenshot shows the Postman interface for a POST request to `https://localhost:44317/api/Product/...`. The request body is a JSON object:

```

1
2   "Name" : "iPhone 14 Pro",
3   "Description" : "6.2 inch AMOLED Display with 16GB RAM and 256GB Storage",
4   "Price" : 1500.00,
5   "CategoryId" : 2
6

```

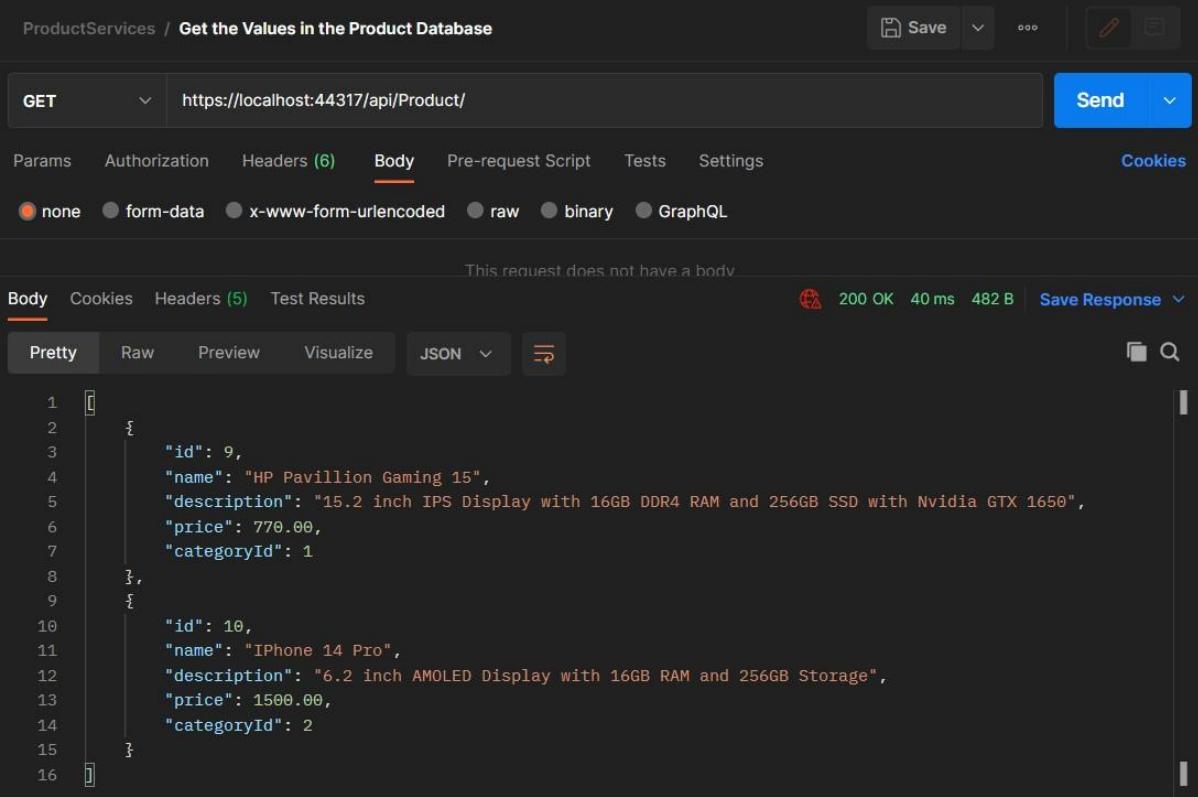
The response status is 201 Created, with a response time of 35 ms and a size of 372 B. The response body is:

```

1
2   "id": 10,
3   "name": "iPhone 14 Pro",
4   "description": "6.2 inch AMOLED Display with 16GB RAM and 256GB Storage",
5   "price": 1500.00,
6   "categoryId": 2
7

```

Step 16: Get all the added products using GET.



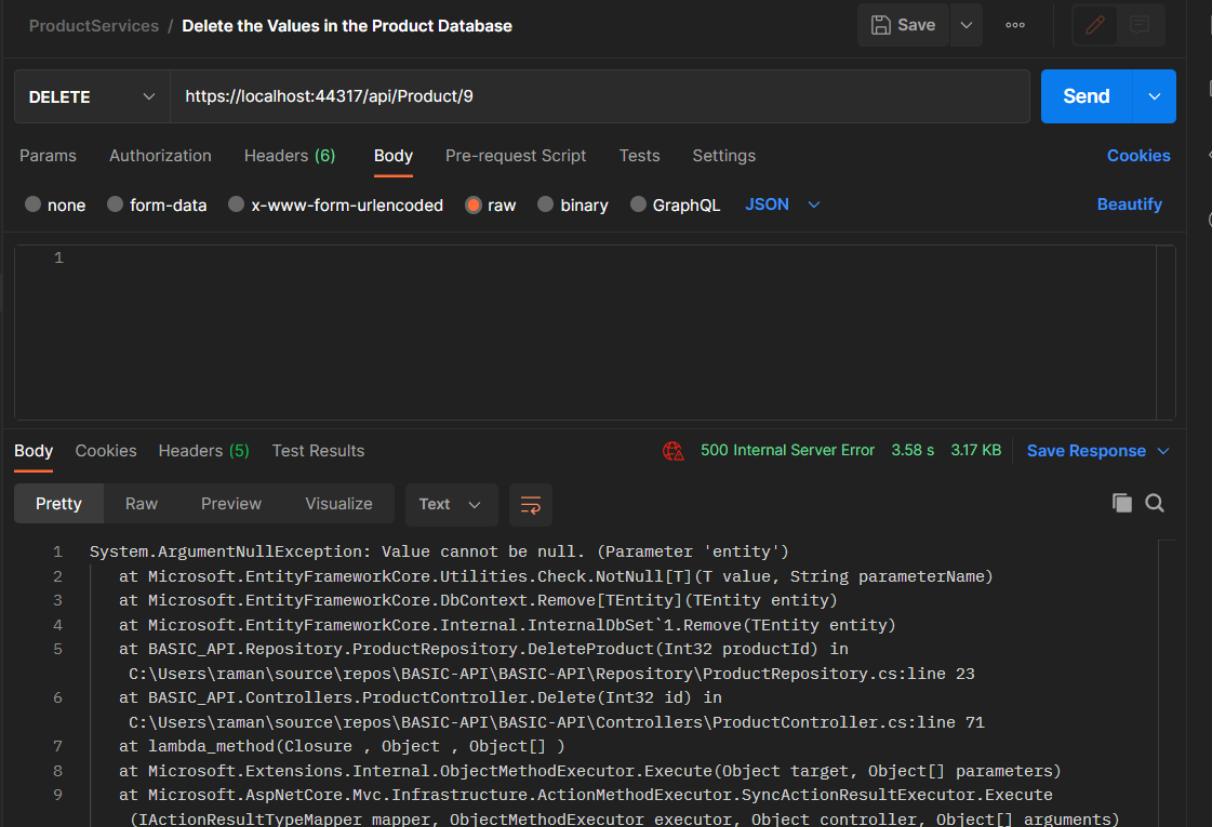
The screenshot shows the Postman interface for a GET request to `https://localhost:44317/api/Product/`. The response status is 200 OK, with a response time of 40 ms and a size of 482 B. The response body is:

```

1
2   {
3     "id": 9,
4     "name": "HP Pavillion Gaming 15",
5     "description": "15.2 inch IPS Display with 16GB DDR4 RAM and 256GB SSD with Nvidia GTX 1650",
6     "price": 770.00,
7     "categoryId": 1
8   },
9   {
10    "id": 10,
11    "name": "iPhone 14 Pro",
12    "description": "6.2 inch AMOLED Display with 16GB RAM and 256GB Storage",
13    "price": 1500.00,
14    "categoryId": 2
15  }
16

```

Step 17: Delete a product using DELETE.



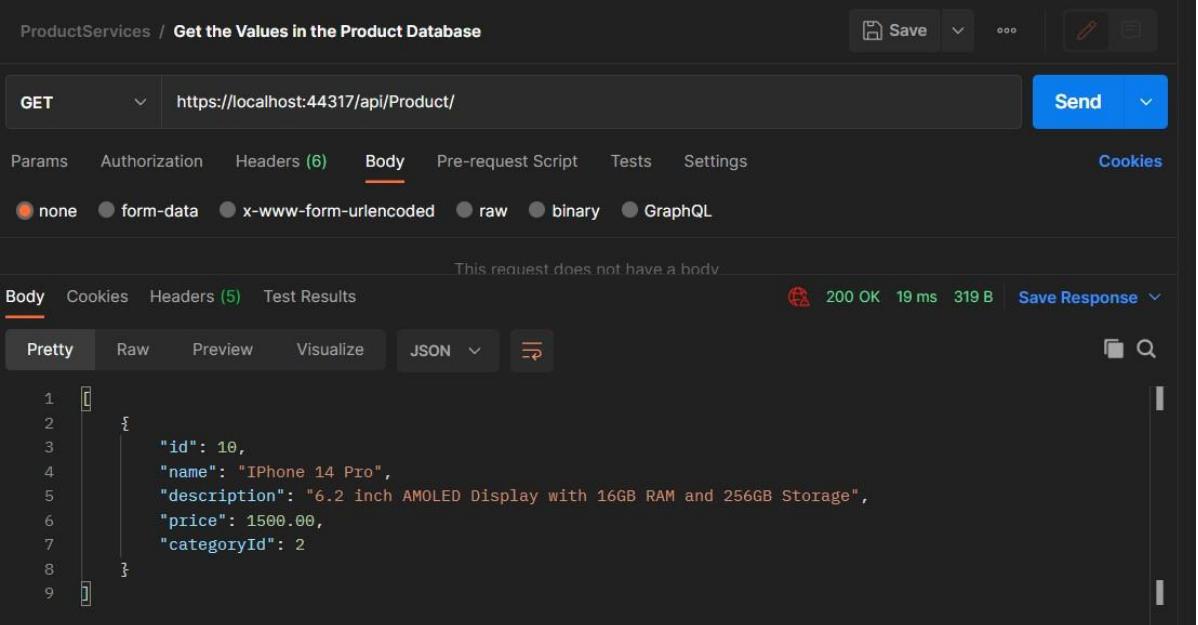
The screenshot shows a Postman request to `https://localhost:44317/api/Product/9` using the `DELETE` method. The response code is `500 Internal Server Error` with a timestamp of `3.58 s` and a size of `3.17 KB`. The error message in the body is:

```

1 System.ArgumentNullException: Value cannot be null. (Parameter 'entity')
2   at Microsoft.EntityFrameworkCore.Utilities.Check.NotNull[T](T value, String parameterName)
3   at Microsoft.EntityFrameworkCore.DbContext.Remove[TEntity](TEntity entity)
4   at Microsoft.EntityFrameworkCore.Internal.InternalDbSet`1.Remove(TEntity entity)
5   at BASIC_API.Repository.ProductRepository.DeleteProduct(Int32 productId) in
6     C:\Users\raman\source\repos\BASIC-API\BASIC-API\Repository\ProductRepository.cs:line 23
7   at BASIC_API.Controllers.ProductController.Delete(Int32 id) in
8     C:\Users\raman\source\repos\BASIC-API\BASIC-API\Controllers\ProductController.cs:line 71
9   at lambda_method(Closure , Object , Object[] )
  at Microsoft.Extensions.Internal.ObjectMethodExecutor.Execute(Object target, Object[] parameters)
  at Microsoft.AspNetCore.Mvc.Infrastructure.ActionMethodExecutor.SyncActionResultExecutor.Execute
    (IActionResultTypeMapper mapper, ObjectMethodExecutor executor, Object controller, Object[] arguments)

```

Step 18: Check if product is deleted using GET again.



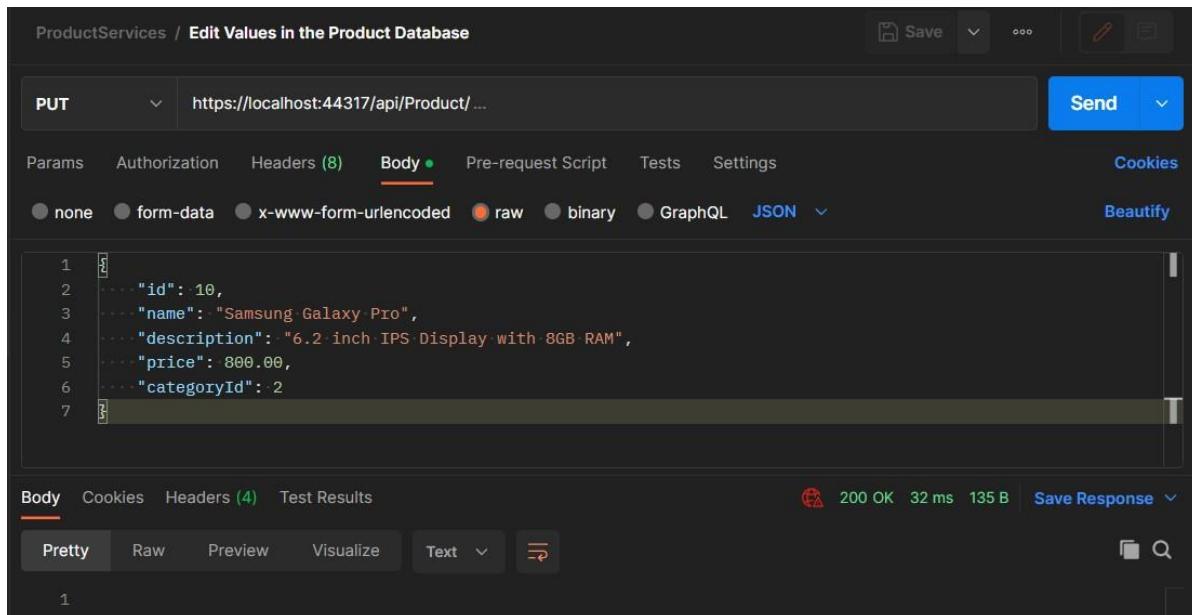
The screenshot shows a Postman request to `https://localhost:44317/api/Product/` using the `GET` method. The response code is `200 OK` with a timestamp of `19 ms` and a size of `319 B`. The response body is a JSON array:

```

1 [
2   {
3     "id": 10,
4     "name": "iPhone 14 Pro",
5     "description": "6.2 inch AMOLED Display with 16GB RAM and 256GB Storage",
6     "price": 1500.00,
7     "categoryId": 2
8   }
9 ]

```

Step 19: Update a product using the PUT.



```

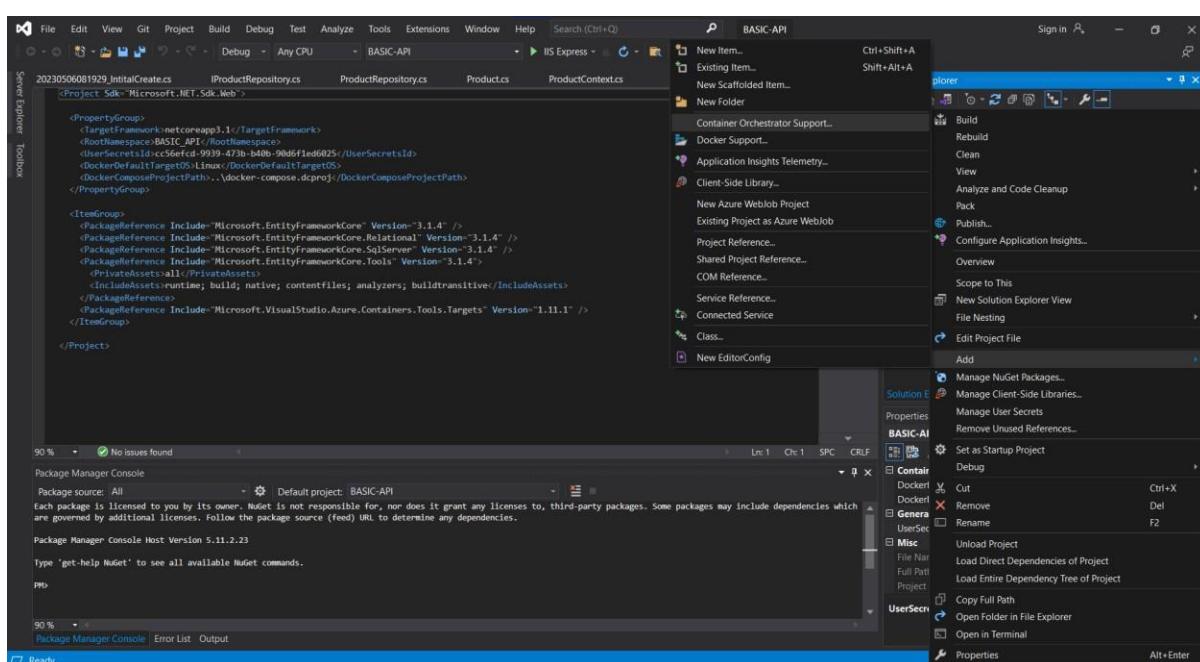
SQLQuery5.sql - L...4DV0D\yraman (55)  SQLQuery4.sql - L...4DV0D\yraman
***** Script for SelectTopNRows command from SSMS ****
SELECT TOP (1000) [Id]
    ,[Name]
    ,[Description]
    ,[Price]
    ,[CategoryId]
FROM [ProductDB].[dbo].[Products]

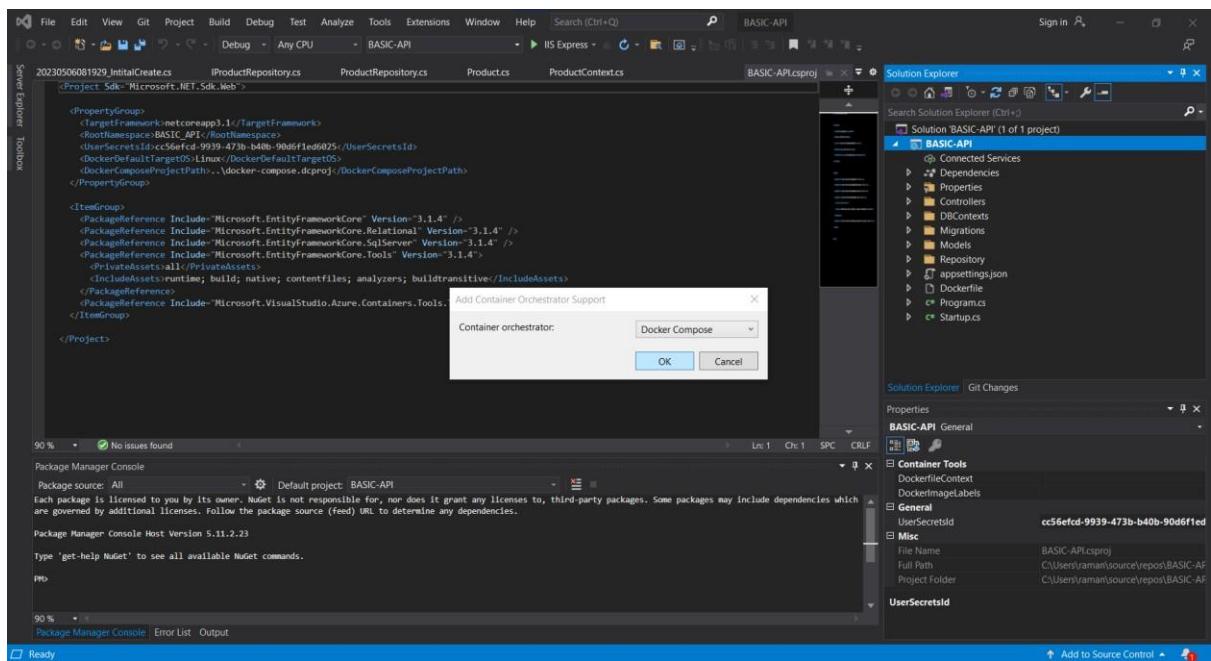
```

100 %

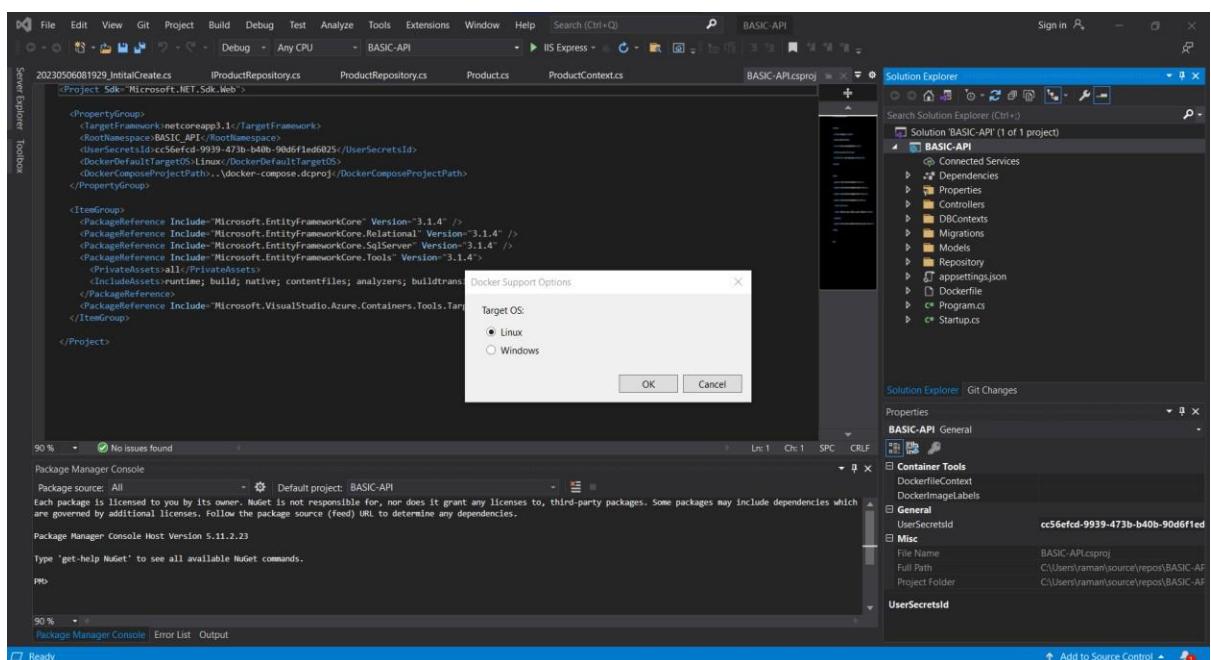
	Results	Messages
1	10 Samsung Galaxy Pro 6.2 inch IPS Display with 8GB RAM 800.00 2	

Step 20: Now create a docker container and image of the product. Right click the project and select Container Orchestrator Support and select Ok.

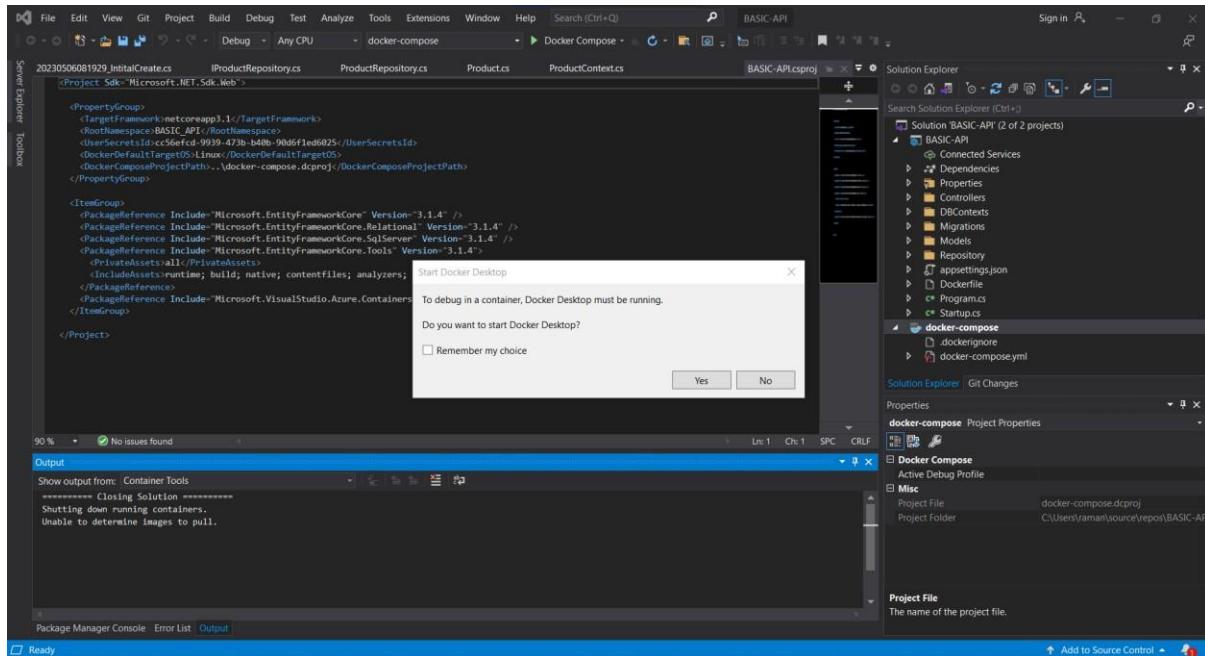




Step 21: Select the target docker OS support which will be LINUX.



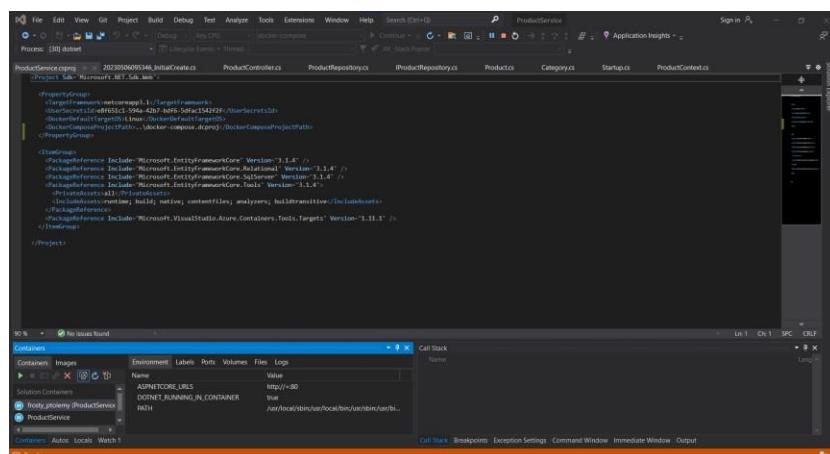
Step 22: You can select Yes to run the Docker desktop too.



Step 23: Run the docker commands ‘docker images’ to see the created image of your project.

```
C:\Users\raman>docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
webapi1            dev      30aee5f1f9c9  7 weeks ago   208MB
webapplication1    dev      872e14f1554c  7 weeks ago   208MB
<none>              <none>  bae4e11c4bb1  7 weeks ago   208MB
productservice     dev      2a9886bd1404  7 weeks ago   208MB
basicapi           dev      591cee1a5e44  7 weeks ago   208MB
<none>              <none>  7a5deb8e122b  7 weeks ago   208MB
webapplication2    dev      5a5a5c6434b9  7 weeks ago   208MB
<none>              <none>  ba2df3366bff  7 weeks ago   208MB
mcr.microsoft.com/dotnet/aspnet      3.1      454a55eab920  4 months ago  208MB
```

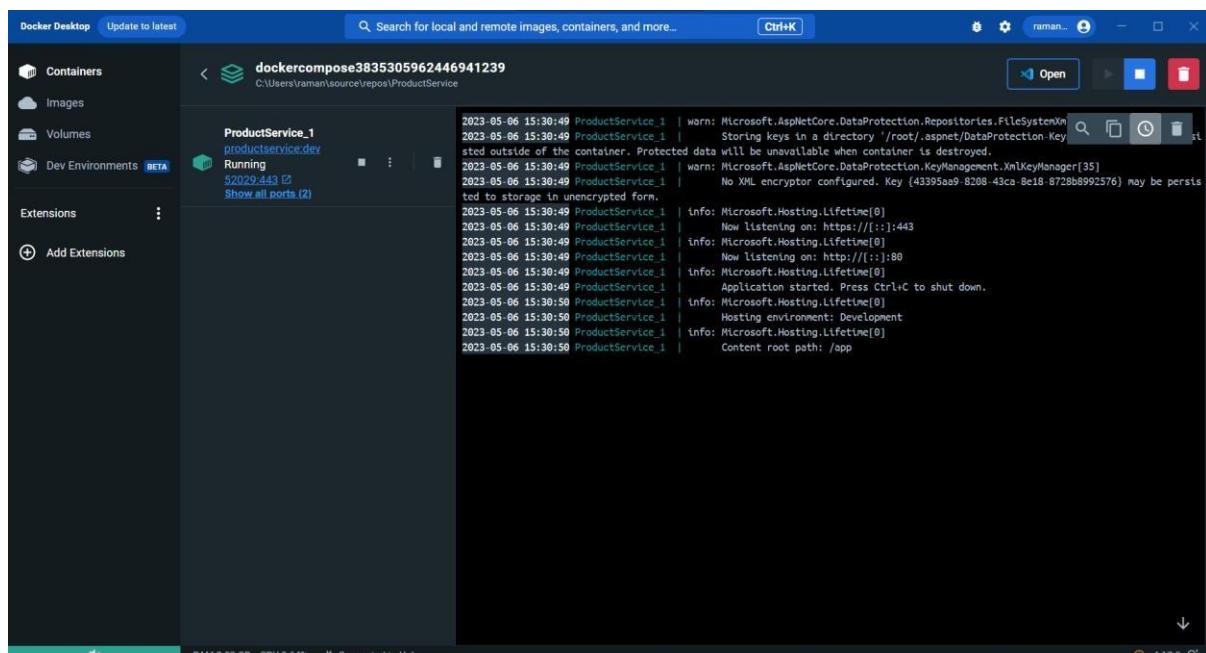
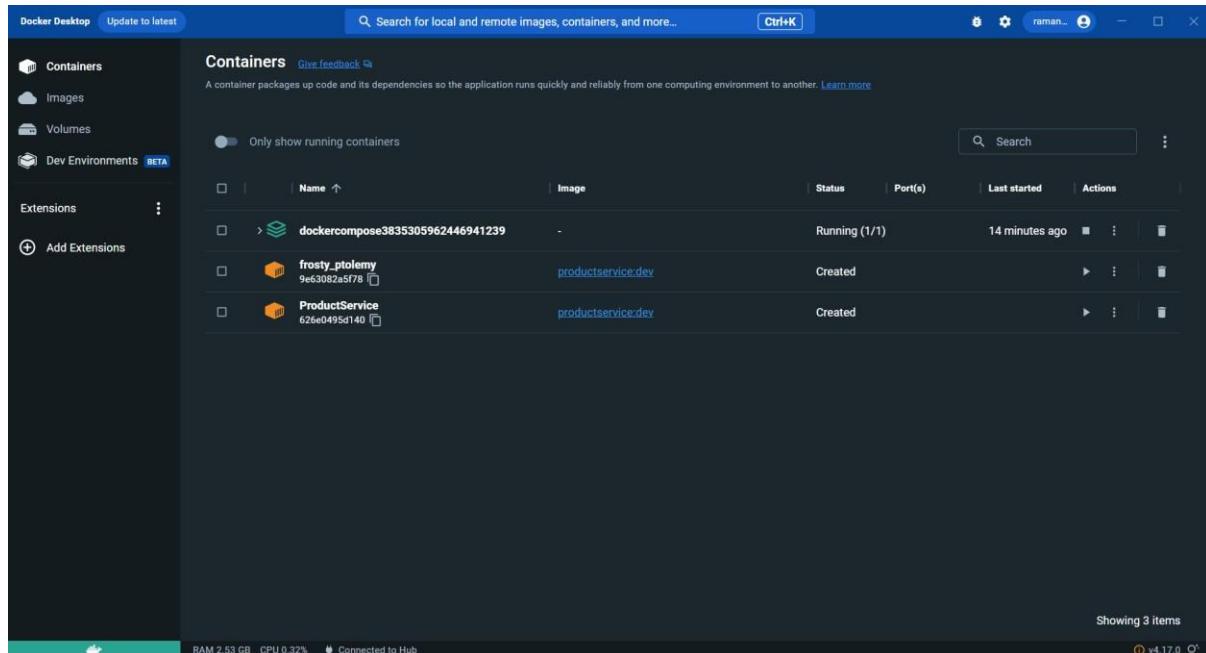
Step 24: Run the Project with the option Docker Compose on the Play.



Step 25: Run the **docker ps** command to check the running docker container.

```
C:\Users\raman\source\repos\ProductService>docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
6d946df292ed      productservice:dev   "tail -f /dev/null"   11 minutes ago    Up 11 minutes     0.0.0.0:52030->80/tcp, 0.0.0.0:52029->443/tcp   ProductService_1
```

Step 26: Check docker desktop for the running image.



Step 27: The docker image and containers will be running in the docker desktop app.