**Practical No.1**

**Aim: Building APT.NET Core MVC Application.**

1)Install .Net Core Sdk (Link: <https://dotnet.microsoft.com/learn/dotnet/hello-world-tutorial/install>)

2)create folder MyMVC folder in C: drive or any other drive

3)open command prompt and perform following operations

Command: to create mvc project

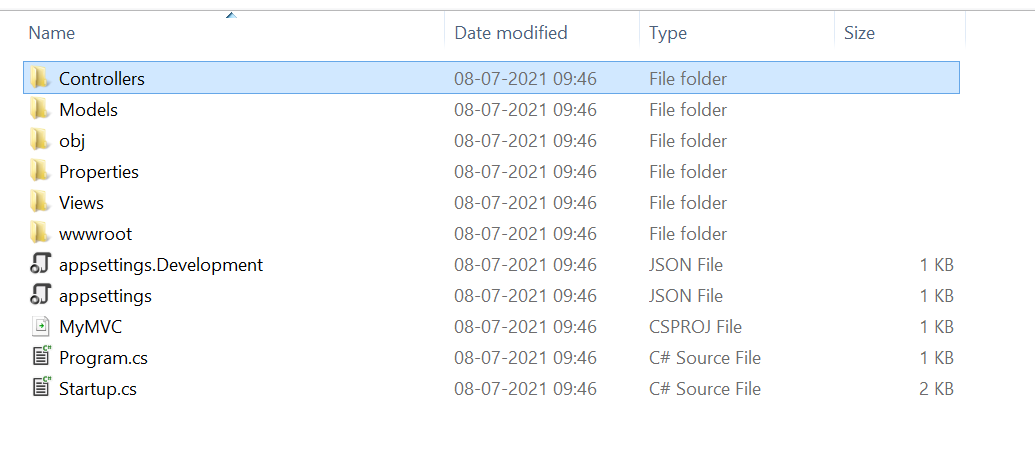
dotnet new mvc --auth none

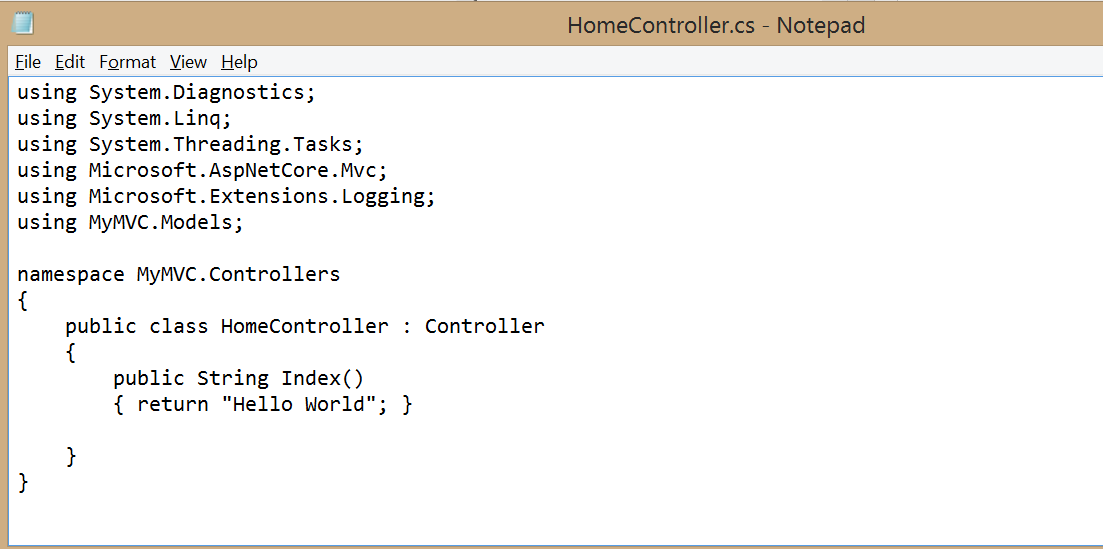
**Output:**

Graphical user interface, application, Word

Description automatically generated

4) Go to controllers folder and modify HomeController.cs file to match following:

****

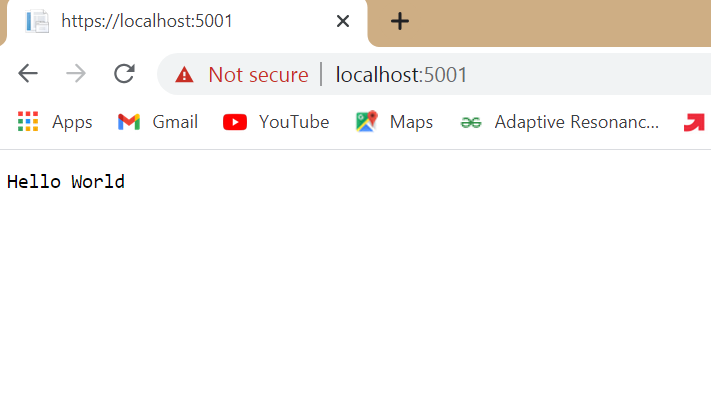
****

Run the Project

Text

Description automatically generated

Now open browser and and type URL: localhost:5000

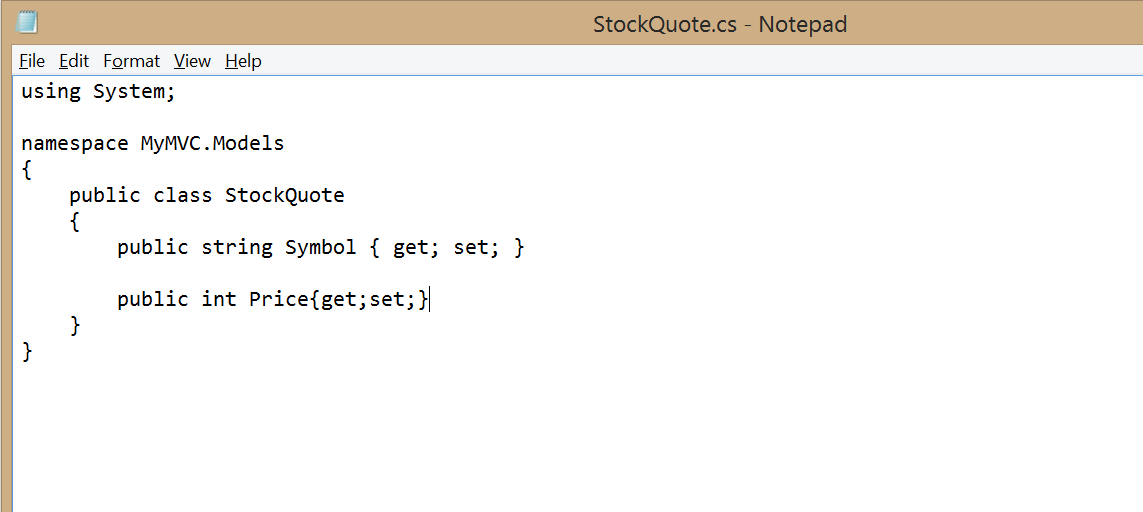


Now go back to command prompt and stop running project using CTRL+C

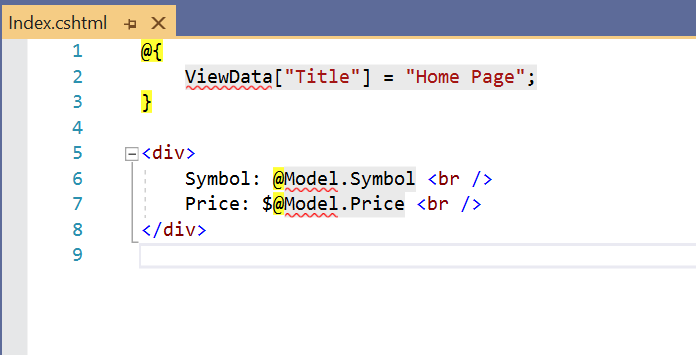
Text

Description automatically generated

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



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



Now modify HomeController.cs file to match following:

Graphical user interface, text, application

Description automatically generated

Now run the project using

Text

Description automatically generated

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



**Practical NO. 2**

**Aim:Building ASP.NET Core REST API.**

Software requirement:

1. Download and install

To start building .NET apps you just need to download and install the .NET SDK (Software Development Kit version

3.0 above).

Link:

https://dotnet.microsoft.com/learn/dotnet/hello-world-tutorial/install

2. Check everything installed correctly

Once you've installed, open a new command prompt and run the following command:

Command prompt

> dotnet

Text

Description automatically generated

Create your web API

1. Open two command prompts

Command prompt 1:

Command:

dotnet new webapi -o Glossary

**output:**

Text

Description automatically generated

Command:

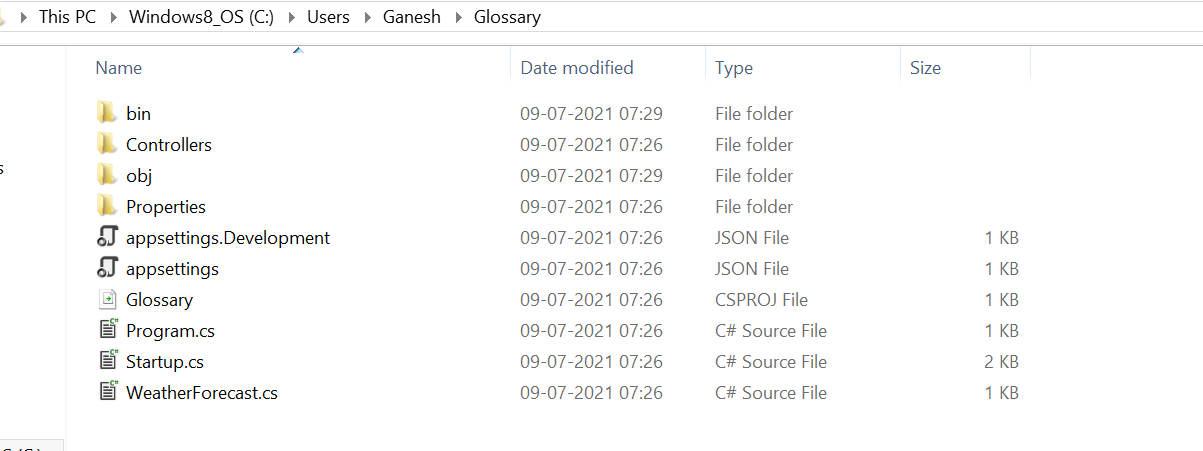
cd Glossary

dotnet run

**Output:**

Text

Description automatically generated

****

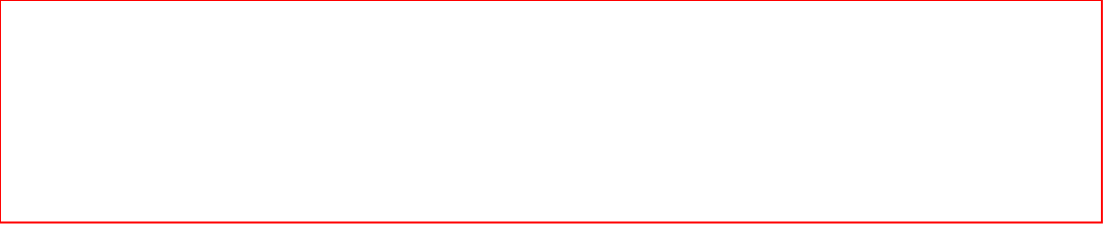
Command Prompt 2: (try running readymade weatherforecast class for testing)

Command:

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

output:

Text

Description automatically generated

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

1) D:\Glossary\GlossaryItem.cs (type it in notepad and save as all files)

//GlossaryItem.cs

namespace Glossary

{

public class GlossaryItem

{

public string Term { get; set; }

public string Definition { get; set; }

}

}

****

D:\Glossary\Controllers\ GlossaryController.cs (type it in notepad and save as all files)

//Controllers/GlossaryController.cs

using System;

using System.Collections.Generic;

using Microsoft.AspNetCore.Mvc;

using System.IO;

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);

}

}

[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();

}

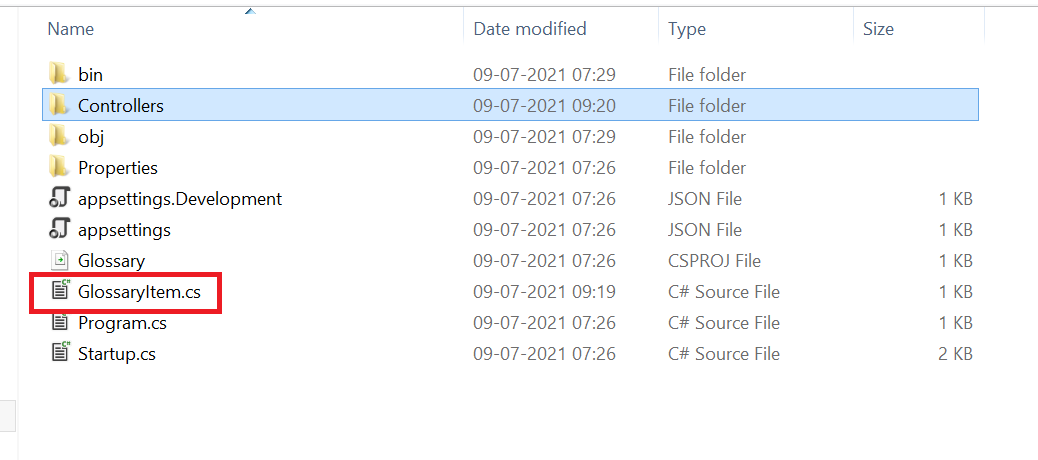
}

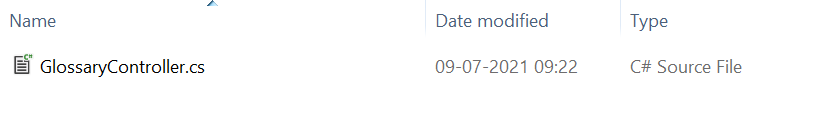
}

}

****

**Output:**

****

****

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

On Command prompt1:

Command:

dotnet run

**output:**

Text

Description automatically generated

On Command prompt2:

1) Getting a list of items:

Command:

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

Text

Description automatically generated

Getting a single item

Command:

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

**Output:**

Text

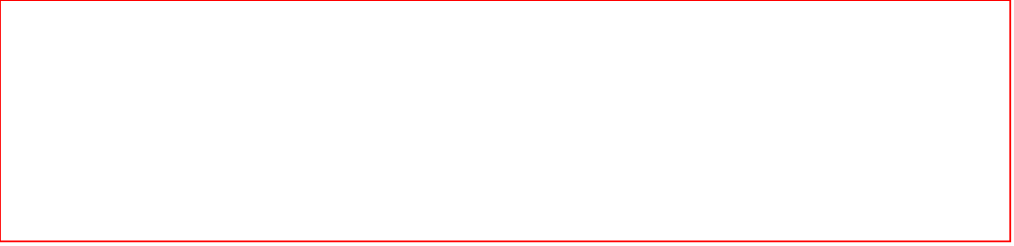
Description automatically generated

Creating an item

Command:

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

Text

Description automatically generated

Update Item

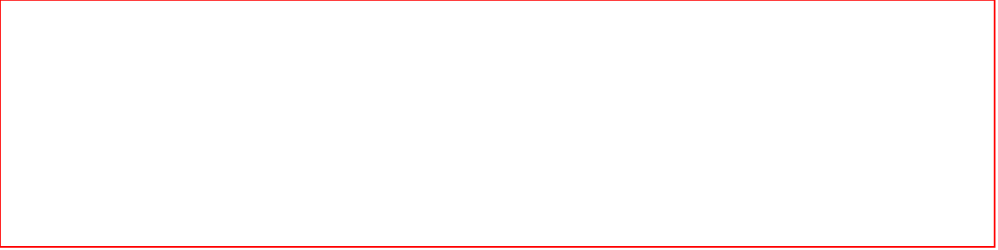
Command:

curl --insecure -X PUT -d "{\"term\": \"MVC\", \"definition\":\"Modified record of Model View

Controller.\"}" -H "Content-Type:application/json" https://localhost:5001/api/glossary

**Output:**

Text

Description automatically generated

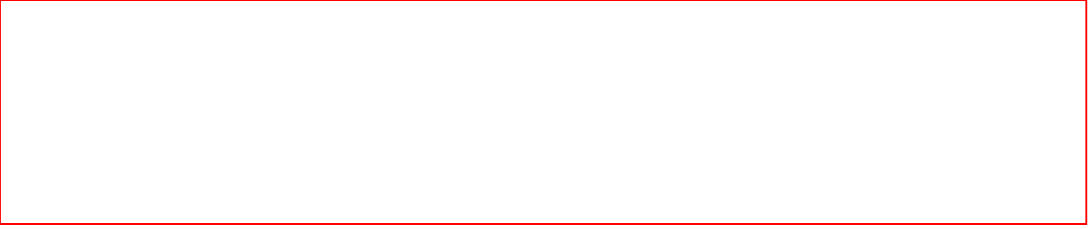
Delete Item

Command:

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

**Output:**

Text

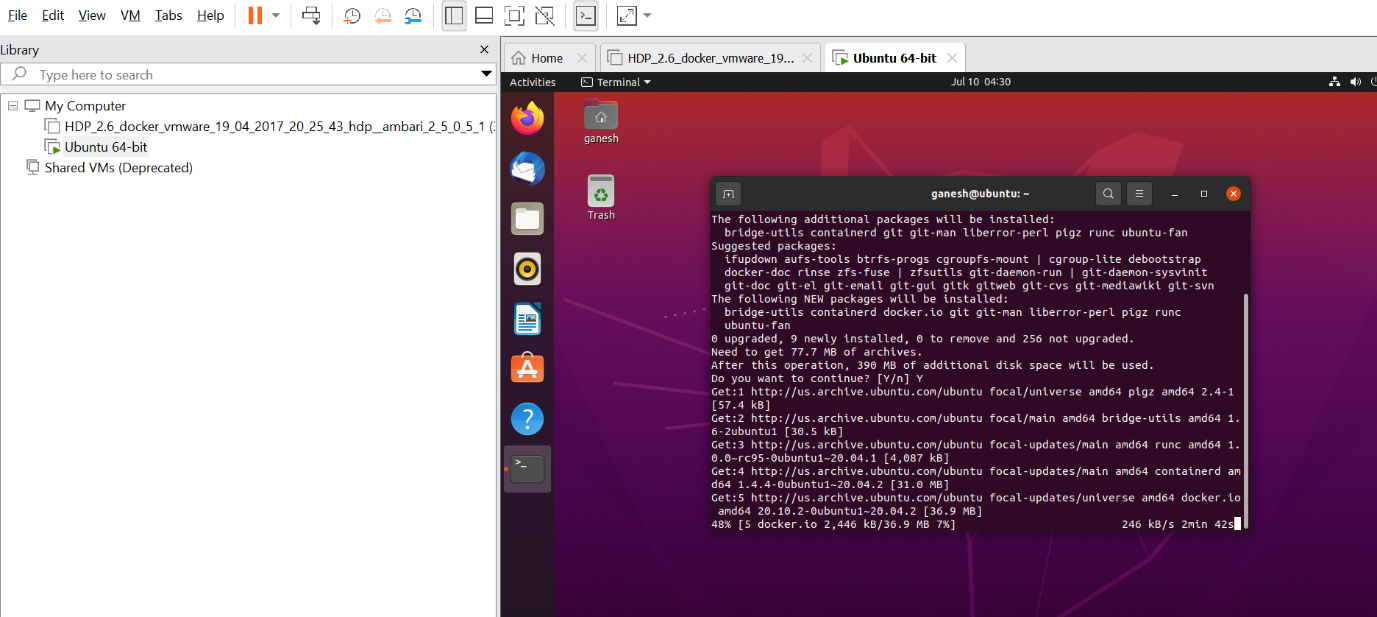
Description automatically generated

**Practical No. 3**

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

**After install ubuntu in vmware. Install docker**

**Command: sudo apt-get install docker.io**

****

Install using the repository

Before you install Docker Engine for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

Set up the repository

Update the apt package index and install packages to allow apt to use a repository over HTTPS:

1. $ sudo apt-get update
2. $ sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

gnupg \

lsb-release

****

1. Add Docker’s official GPG key:

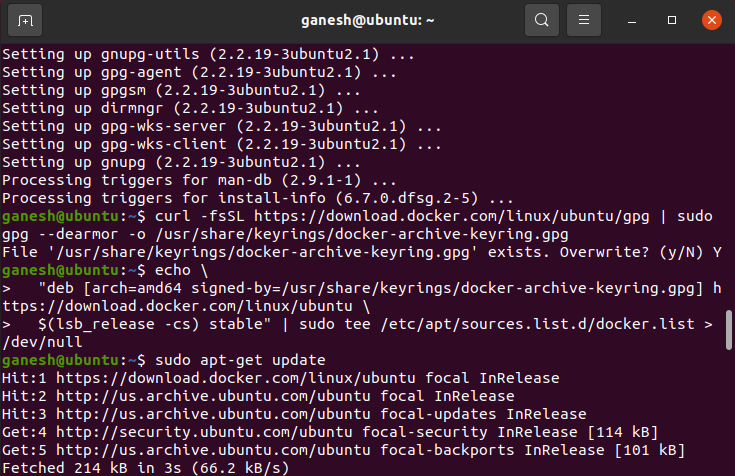
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

Use the following command to set up the **stable** repository

$ echo \

"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

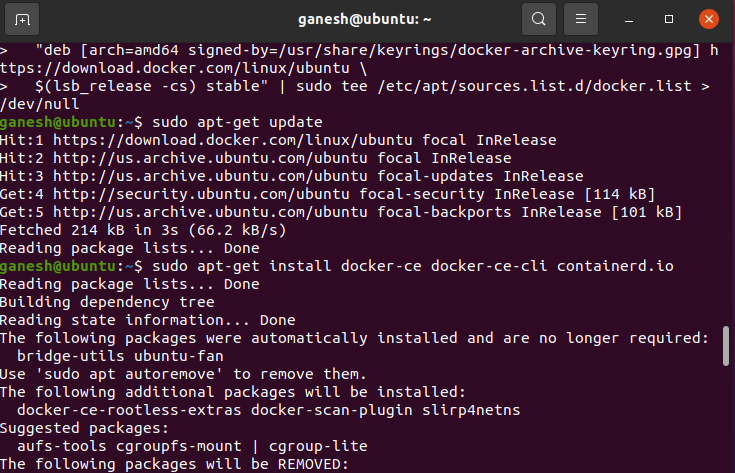


Install Docker Engine

Update the apt package index, and install the *latest version* of Docker Engine and containerd, or go to the next step to install a specific version:

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io

****

To install a *specific version* of Docker Engine, list the available versions in the repo, then select and install:

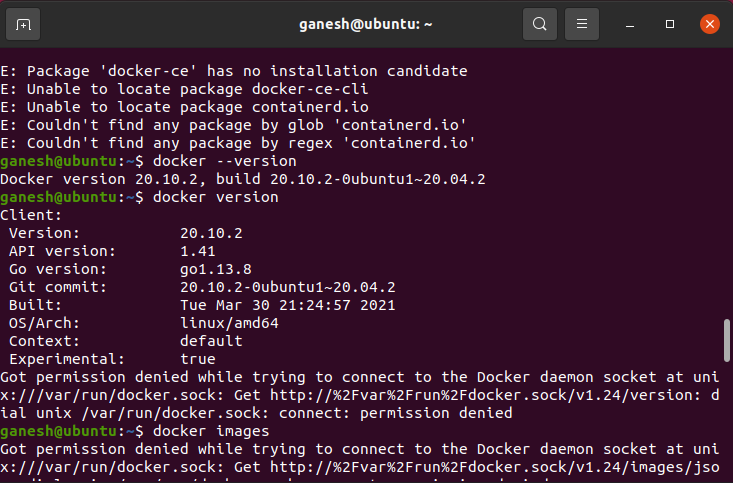
apt-cache madison docker-ce

****

Docker Commands:

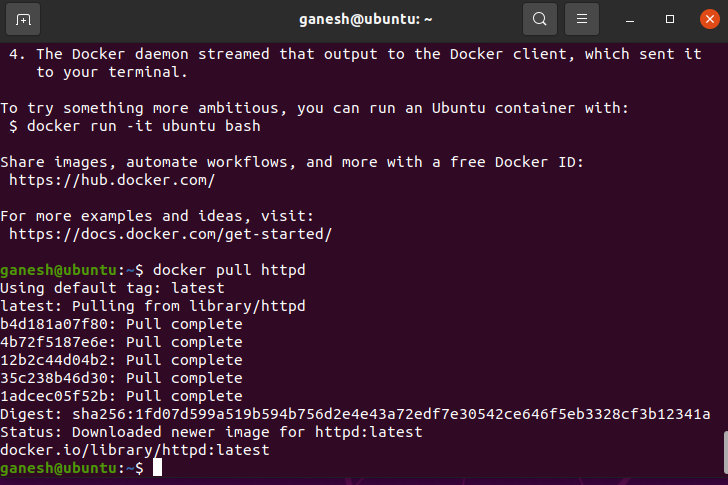
Docker –version

Docker version

****

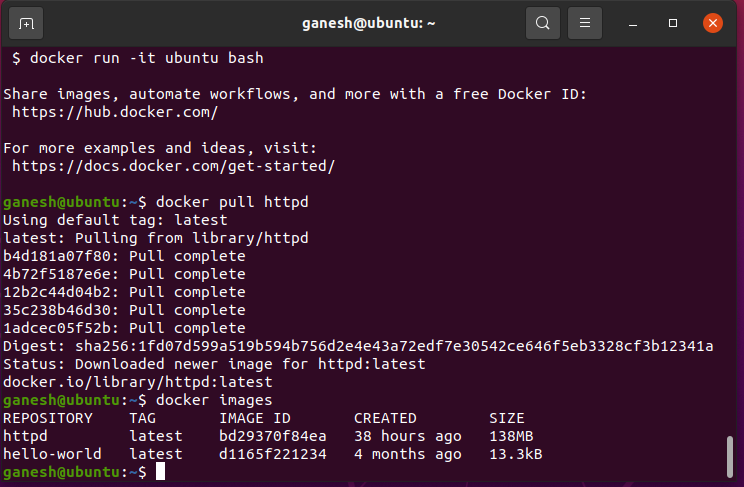
**Docker pull httpd**

Pull an image or a repository from a registry



**Docker images**

It lists all the images



**#nano Dockerfile**

**FROM busybox**

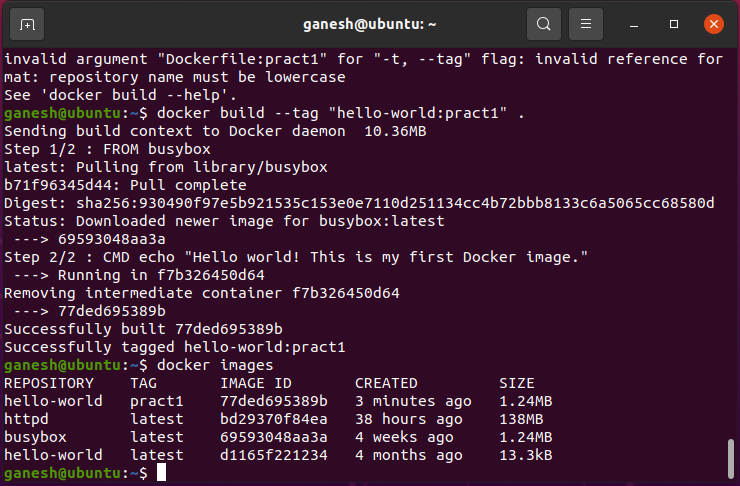
**CMD echo "Hello world! This is my first Docker image."**

**//above two line we have to add into dockerfile**

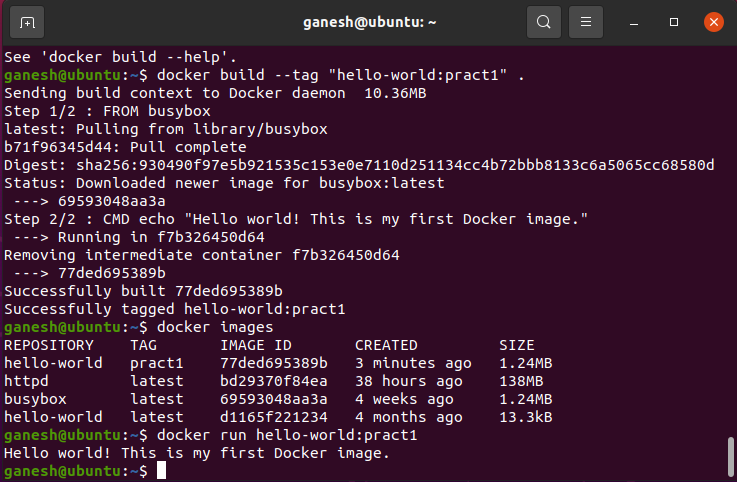
**to save press ctrl+o(to write) then enter then ctrl+x (to exit)**

**docker build --tag "hello-world:pract1" .**

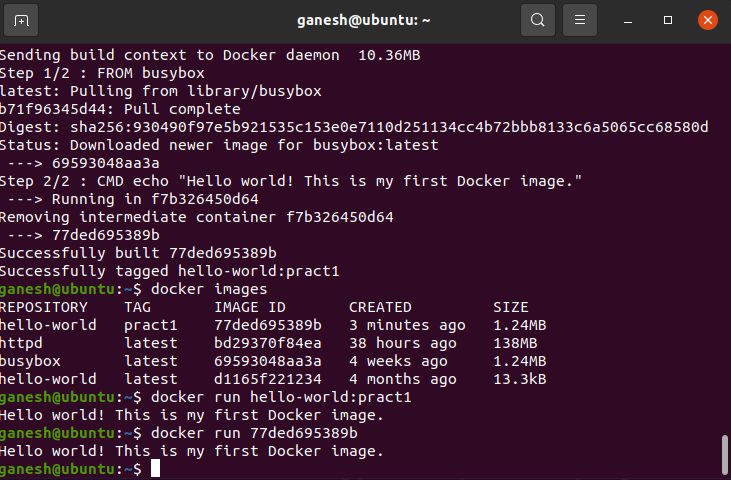
**docker images**

****

**docker run hello-world:pract1**

****

**docker run 77ded695389b**

****

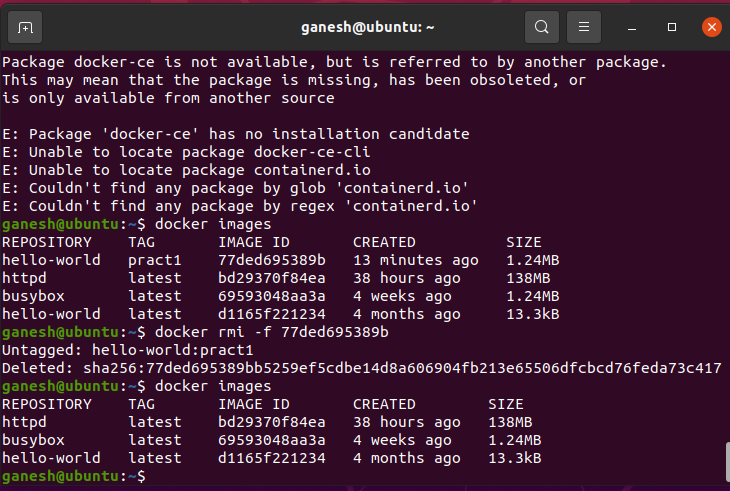
**Docker rmi**

**Remove one or more images**

**docker rmi -f images-id**

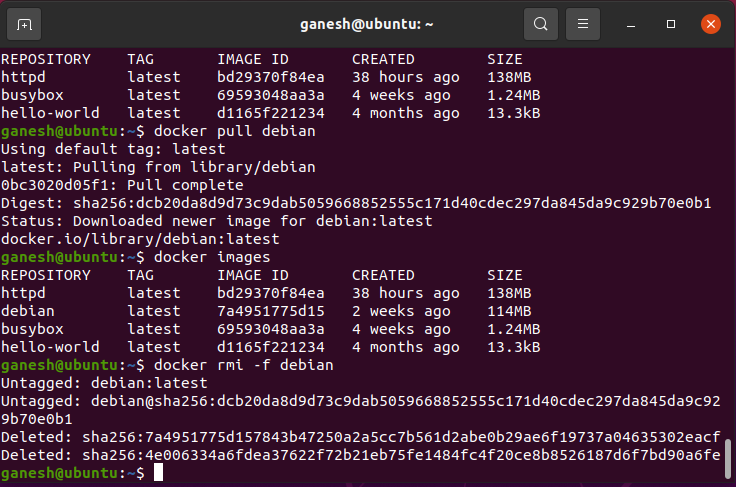
**docker rmi -f 77ded695389b**

**After running docker images we can see that 77ded695389b is deleted.**

****

**docker rmi -f Respository-name**

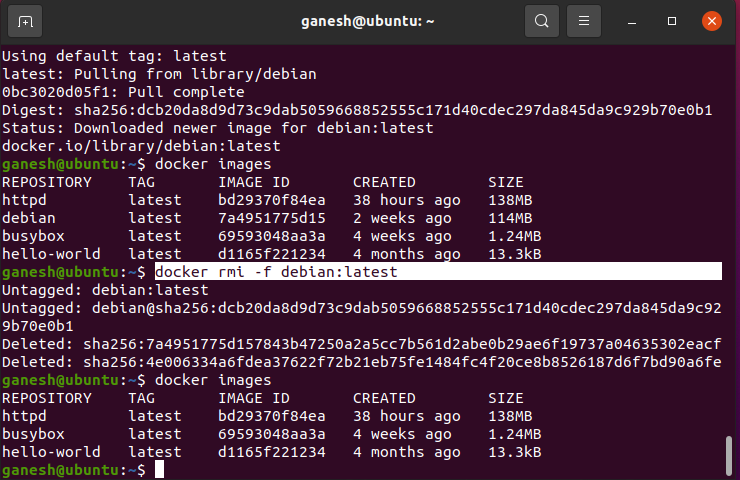
**docker rmi -f Debian**



**docker rmi -f Respository-name:tag**

**docker rmi -f debian:latest**

**After this debain image will be deleted**

****

**Practical No. 4**

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

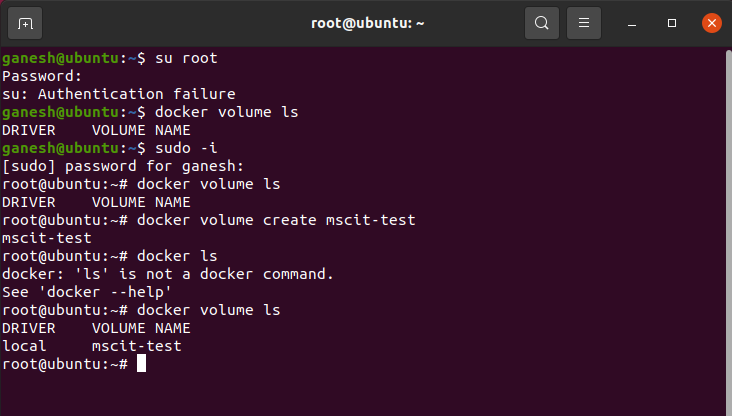
Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While [bind mounts](https://docs.docker.com/storage/bind-mounts/) are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker.

List volumes created

Command: docker volume ls

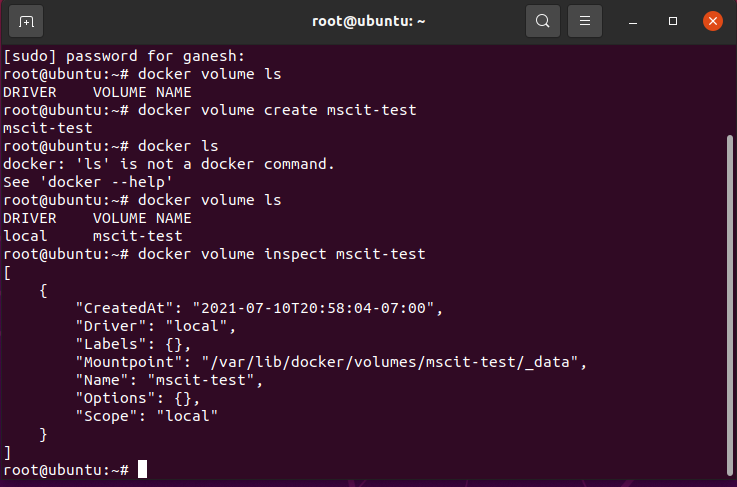
To create volume.

Command: docker volume create mscit-test



Return low-level information on Docker objects

Command: docker volume inspect mscit-test

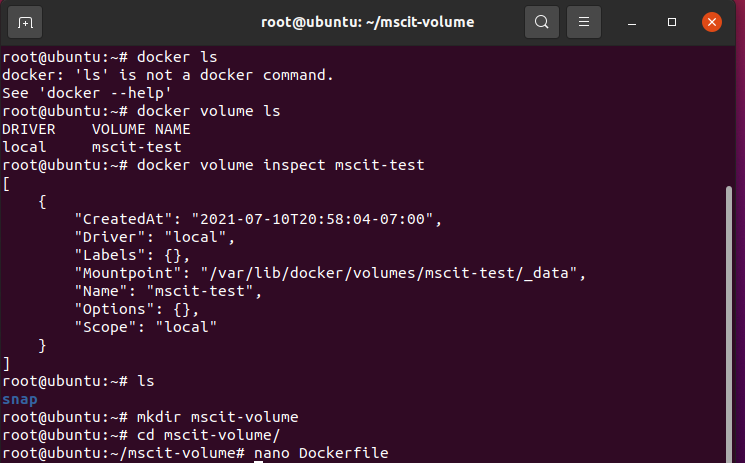


Create a directory

mkdir mscit-volume

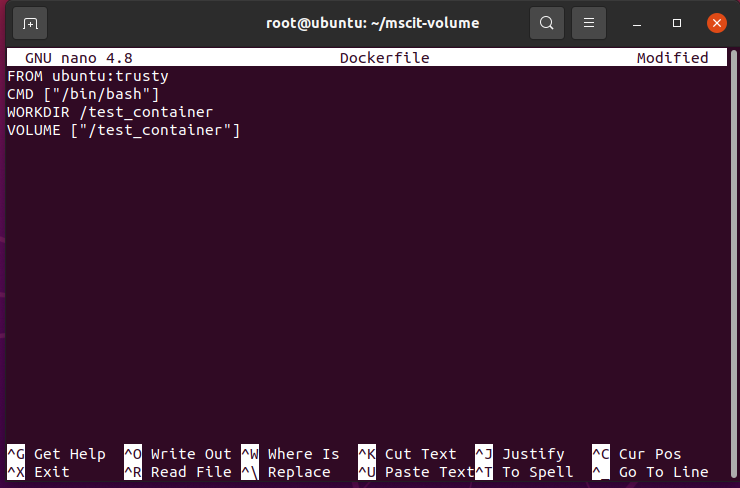
Now, change directory to mscit-volume

cd mscit-volume/



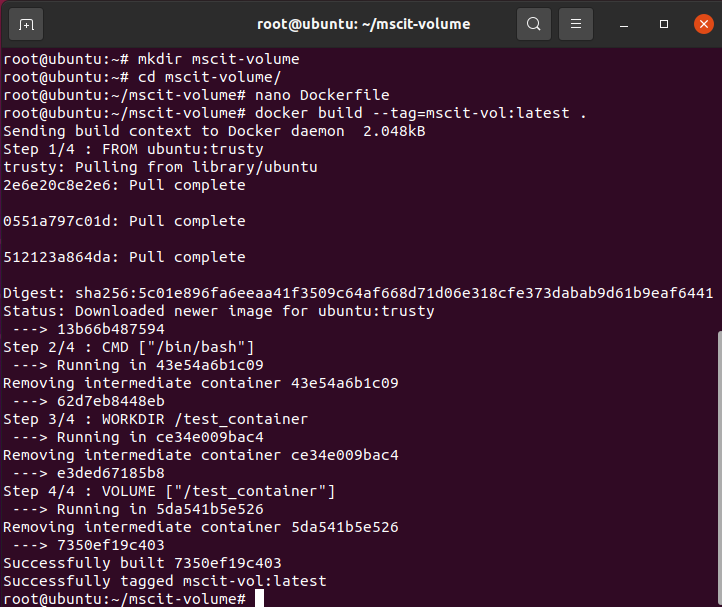
Create a file

Nano Dockerfile



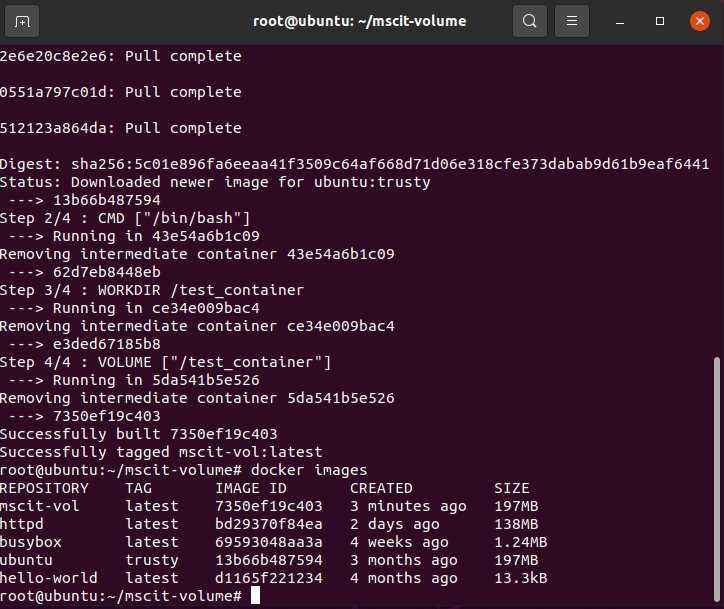
To create an image file

docker build --tag=mscit-vol:latest .



Check the image create

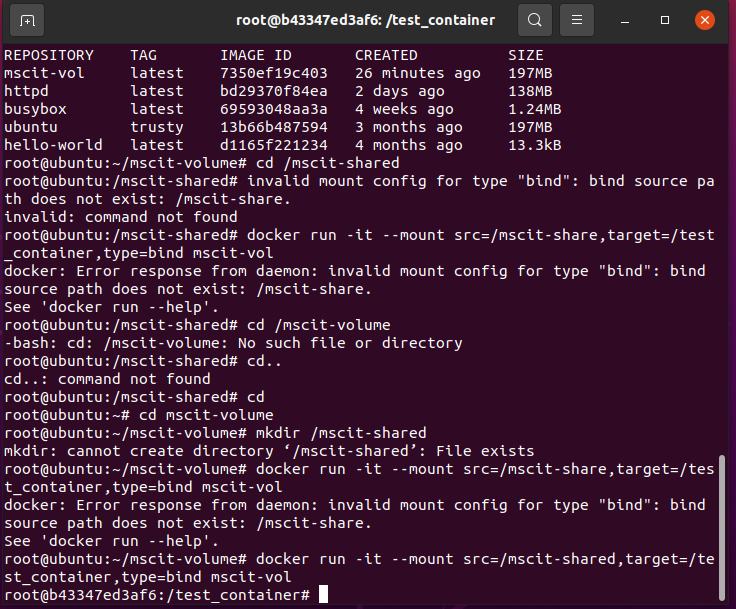
Command: docker images



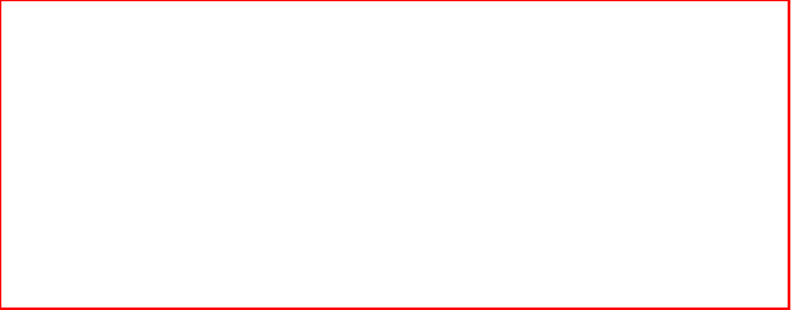
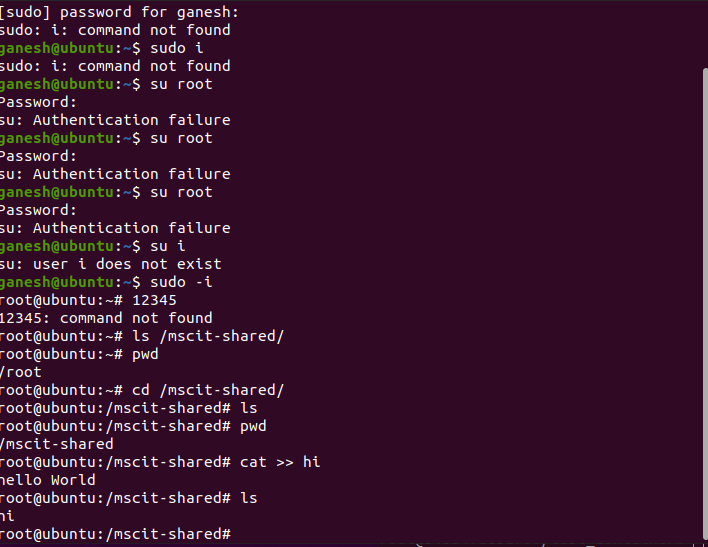
Mounting the container

docker run -it --mount src=/mscit-shared,target=/test\_container,type=bind mscit-vol

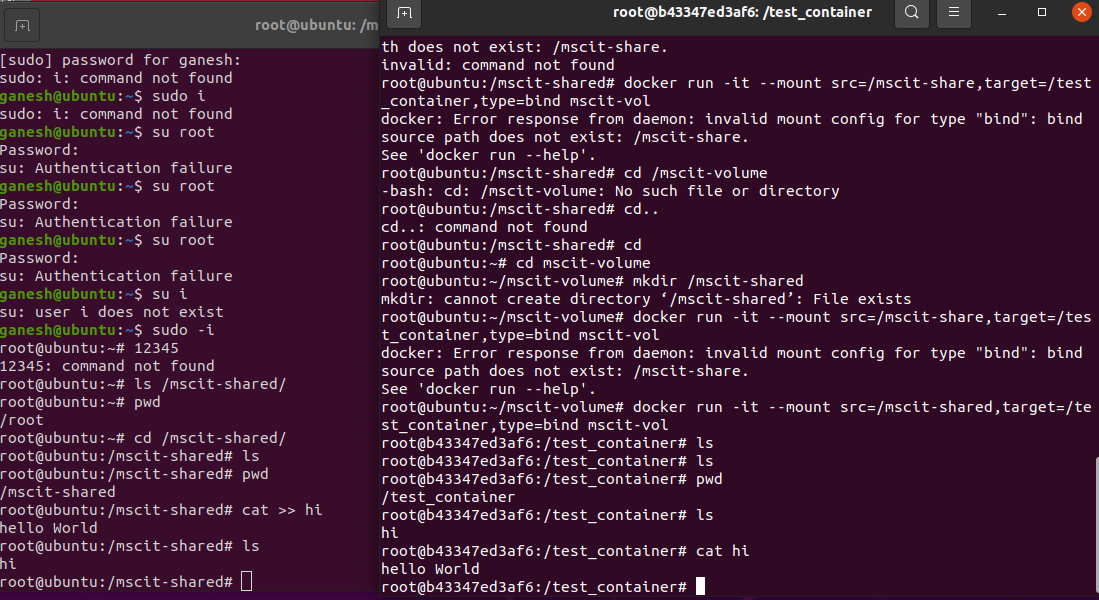
It will change in root and show test\_Container

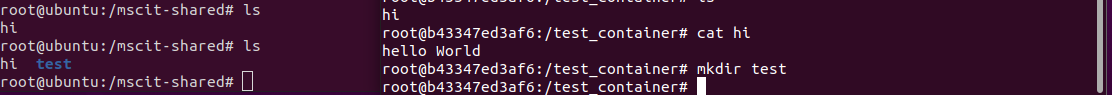


Now open other terminal and get into mscit-shared directory and create a file called hi



Now check the file created in root is listed in test\_Container and vice-versa.





We can see that file location are mapped.

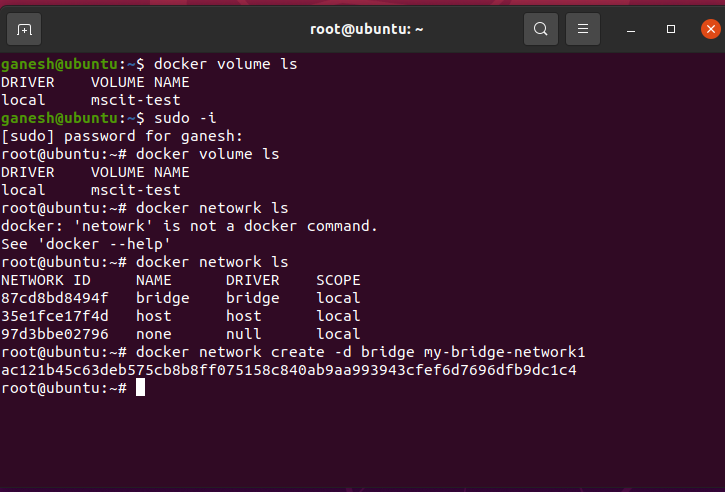
When below command is executed, it will delete the volume.

docker volume rm mscit-test

**Network:**

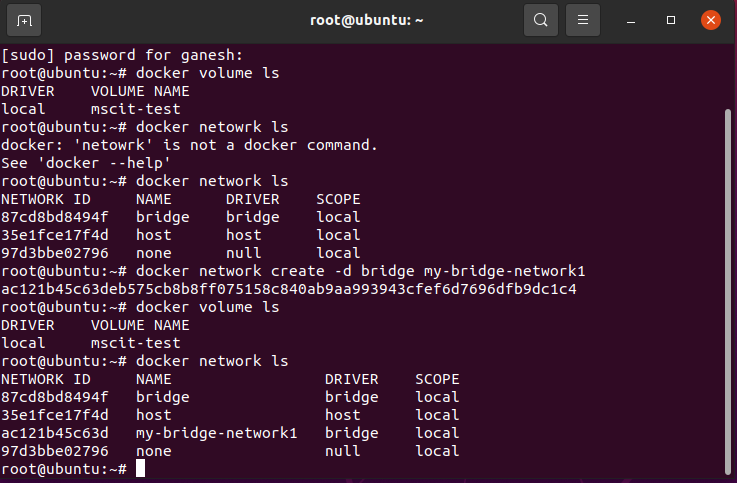
Create network with following command

docker network create -d bridge my-bridge-network1



Check network is created with below command

Command: docker network ls



We can inspect the created network with below command

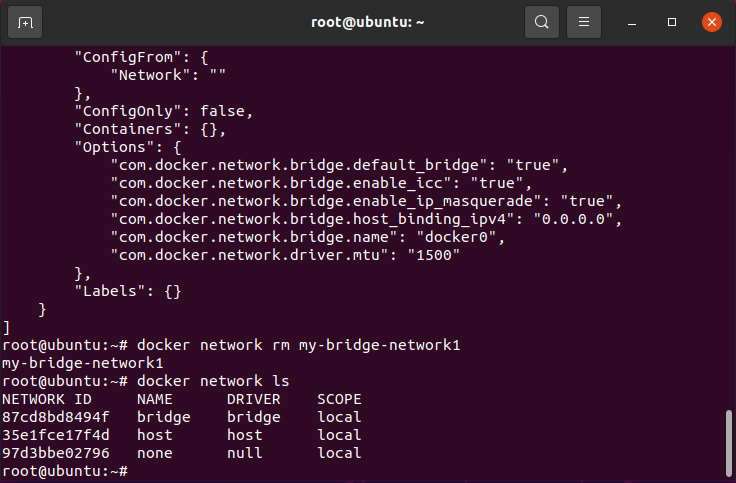
docker network inspect bridge (network name)



Now, lets remove the create network using below command.

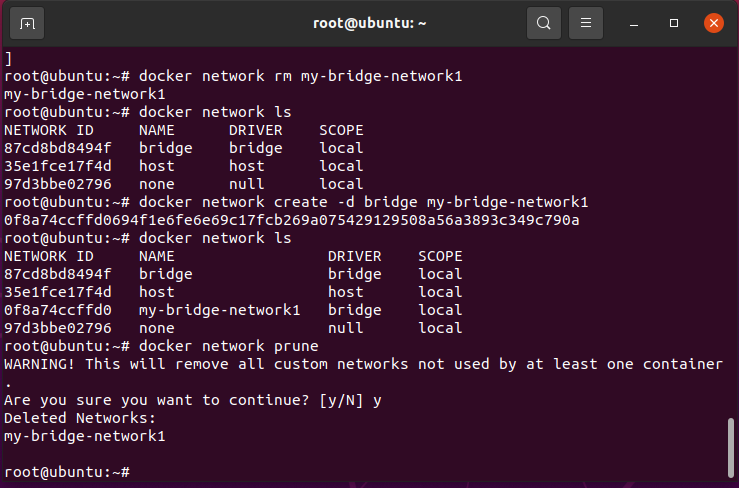
docker network rm network-name

With docker network ls we can see the my-bridge-network1 is delected.



With below command we can delete unused networks

docker network prune



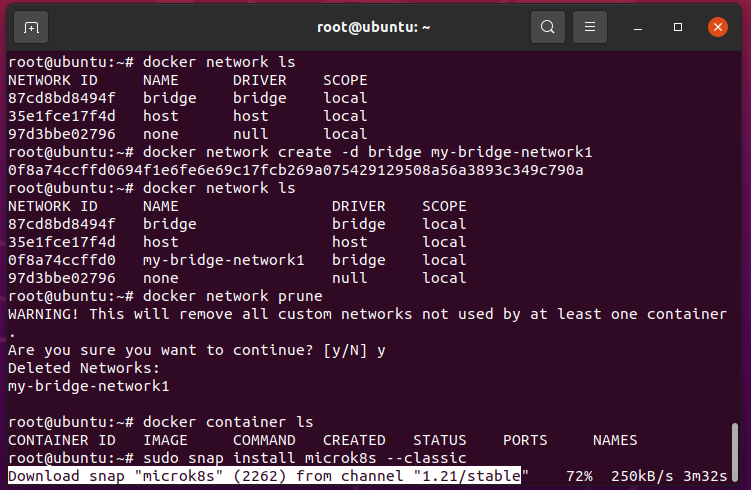
**Practical No. 5**

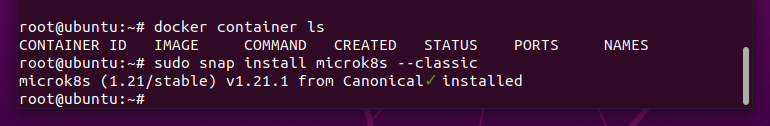
**Aim: Working with Kubernetes.**

Kubernetes, or k8s, is an open-source platform that automates Linux container operations. It eliminates many of the manual processes involved in deploying and scaling containerized applications. “In other words, you can cluster together groups of hosts running Linux containers, and Kubernetes helps you easily and efficiently manage those clusters.”

Install MicroK8s on Linux

sudo snap install microk8s --classic





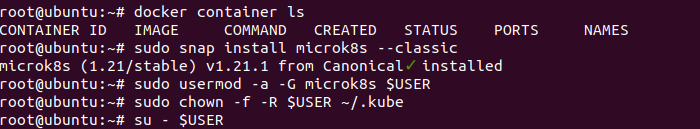
Add your user to the microk8s admin group

MicroK8s creates a group to enable seamless usage of commands which require admin privilege. Use the following commands to join the group:

sudo usermod -a -G microk8s $USER

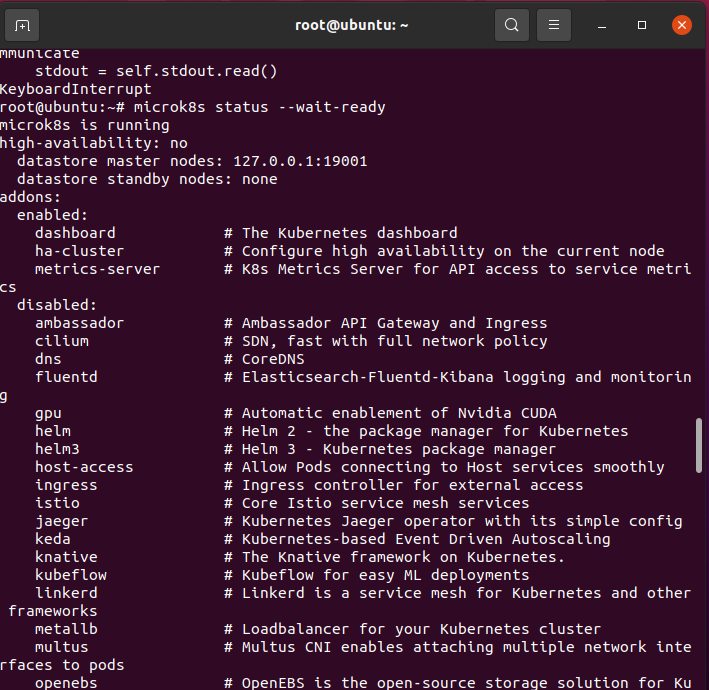
sudo chown -f -R $USER ~/.kube

su - $USER



Check the status while Kubernetes starts

microk8s status --wait-ready



Turn on the services you want

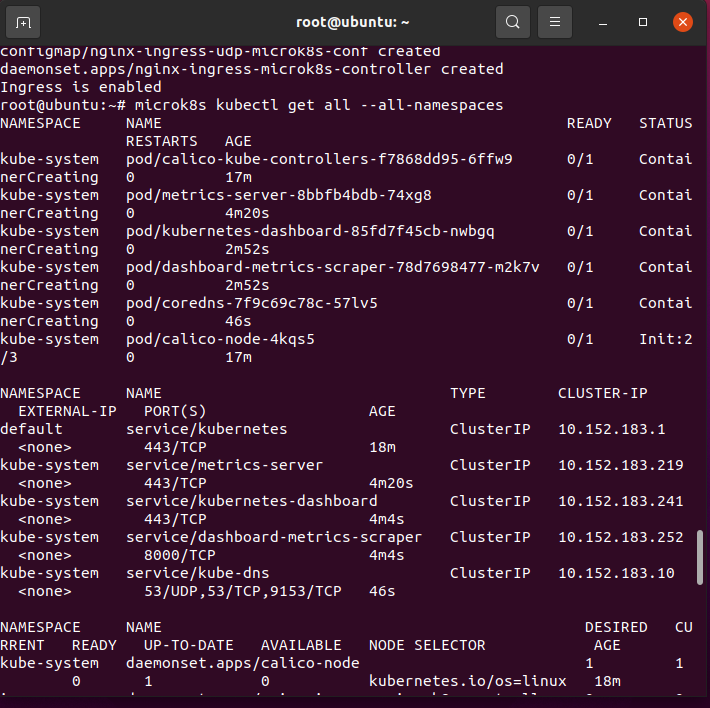
microk8s enable dashboard dns ingress

Text

Description automatically generated

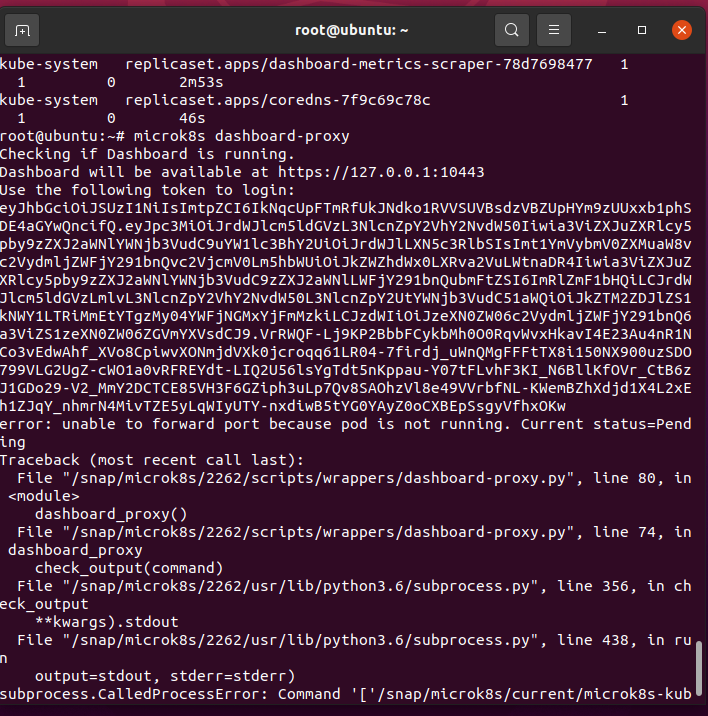
Start using Kubernetes

microk8s kubectl get all --all-namespaces



Access the Kubernetes dashboard

microk8s dashboard-proxy



**Token for login:**

eyJhbGciOiJSUzI1NiIsImtpZCI6IkNqcUpFTmRfUkJNdko1RVVSUVBsdzVBZUpHYm9zUUxxb1phSDE4aGYwQncifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJkZWZhdWx0LXRva2VuLWtnaDR4Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImRlZmF1bHQiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiJkZTM2ZDJlZS1kNWY1LTRiMmEtYTgzMy04YWFjNGMxYjFmMzkiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6a3ViZS1zeXN0ZW06ZGVmYXVsdCJ9.VrRWQF-Lj9KP2BbbFCykbMh0O0RqvWvxHkavI4E23Au4nR1NCo3vEdwAhf\_XVo8CpiwvXONmjdVXk0jcroqq61LR04-7firdj\_uWnQMgFFFtTX8i150NX900uzSDO799VLG2UgZ-cWO1a0vRFREYdt-LIQ2U56lsYgTdt5nKppau-Y07tFLvhF3KI\_N6BllKfOVr\_CtB6zJ1GDo29-V2\_MmY2DCTCE85VH3F6GZiph3uLp7Qv8SAOhzVl8e49VVrbfNL-KWemBZhXdjd1X4L2xEh1ZJqY\_nhmrN4MivTZE5yLqWIyUTY-nxdiwB5tYG0YAyZ0oCXBEpSsgyVfhxOKw

Sign in with token:

