MICROSERVICE ARCHITECTURE

**SUBMITTED BY SHUBHAM P. KALE**

**2022-2023**



## HSNC UNIVERSITY

**MASTERS OF SCIENCE IN INFORMATION TECHNOLOGY KISHINCHAND CHELLARAM COLLEGE**

## D.W.ROAD, CHURCHGATE, MUMBAI – 400 020.

Subject code – MS-FIT-2P1

# MICROSERVICE ARCHITECTURE

### Hyderabad (Sind) National Collegiate Board - WikipediaKishinchand Chellaram College - WikipediaKISHINCHAND CHELLARAM COLLEGE

CHURCHGATE, MUMBAI – 400 020.

### DEPARTMENT OF INFORMATION TECHNOLOGY M.SC. PART- I

**CERTIFICATE**

This is to certify that the practical done at **K.C. College** by

### MR/MS. SHUBHAM P. KALE

**(**Seat No :\_**KFMSCI**\_**T01**\_**5** ) in partial fulfilment for M.SC. (I.T.) Degree Examination has been found satisfactory. This Practical journal had not been submitted for any other examination and does not form part of any other course undergone by the candidate.

**Signature Signature Signature Lecturer-In-Charge External Examiner Course Coordination**

**Guided By Examined By Certified By**

**College Stamp**

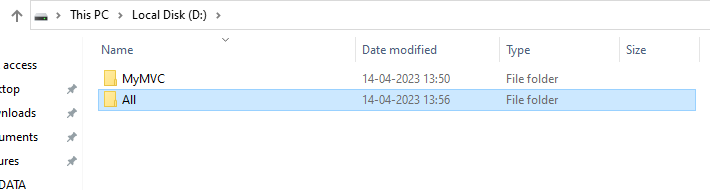
# INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Practical** | **Date** | **Sign.** |
| **1.** | **Building ASP.NET Core MVC Application** |  |  |
| **2.** | **Building ASP. NET Core REST API** |  |  |
| **3.** | **Working with docker, docker commands, docker images & Containers.** |  |  |
| **4.** | **Installing s/w packages on docker.** |  |  |
| **5.** | **working with docker volumes & networks.** |  |  |
| **6.** | **working with CircleCI for Continuous integration** |  |  |
| **7.** | **Creating Backing service with ASP.NET 2.0 core.** |  |  |

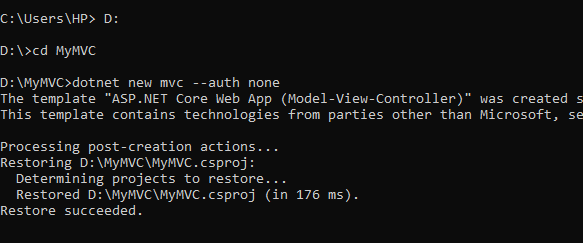
## PRACTICAL NO. : 01

**Aim:- Building ASP.NET Core MVC Application**

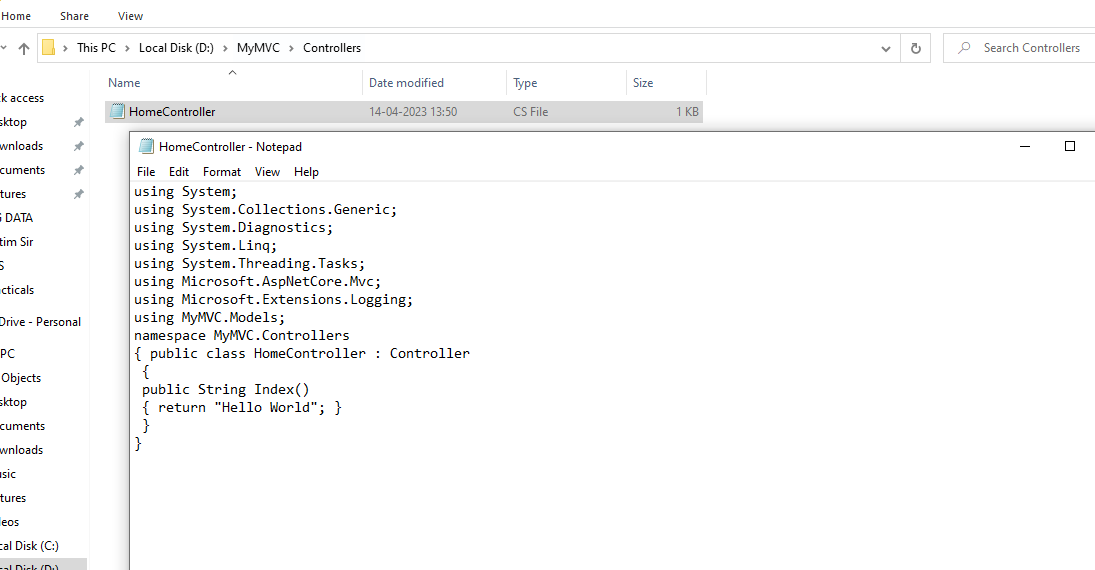
1. Install .Net Core Sdk
2. create folder MyMVC folder in D: drive or any other drive

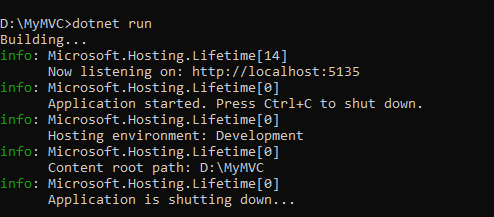


1. open command prompt and perform following operations Command: to create mvc project dotnet new mvc --auth none

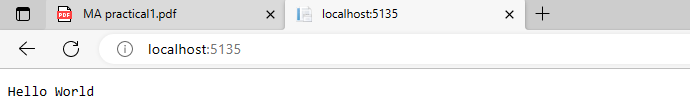


1. Go to controllers folder and modify HomeController.cs file to match following:

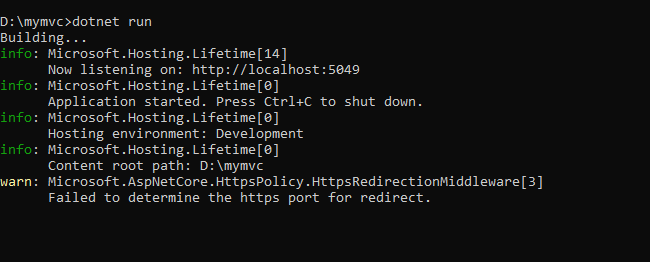


1. Run the project

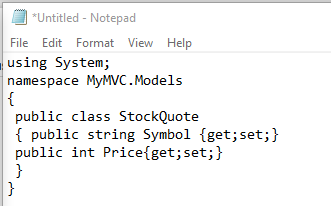
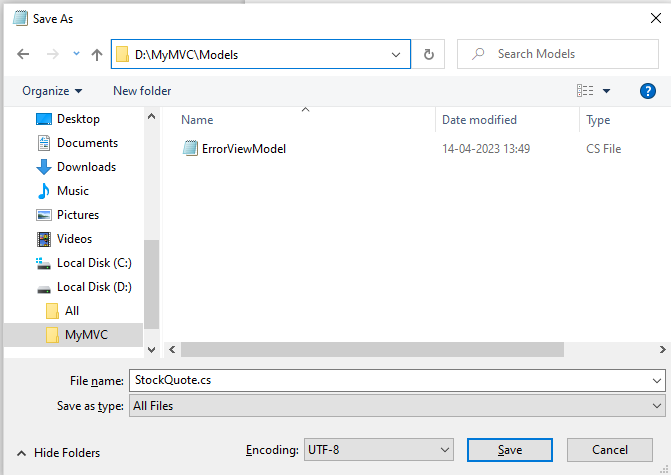
Now open browser and and type URL: localhost:5135

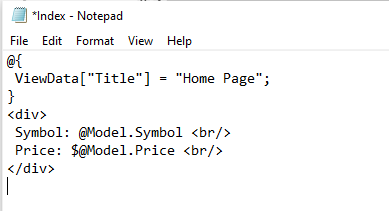
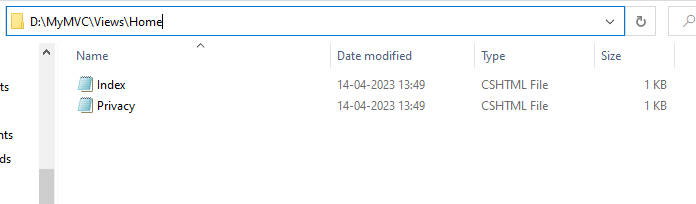


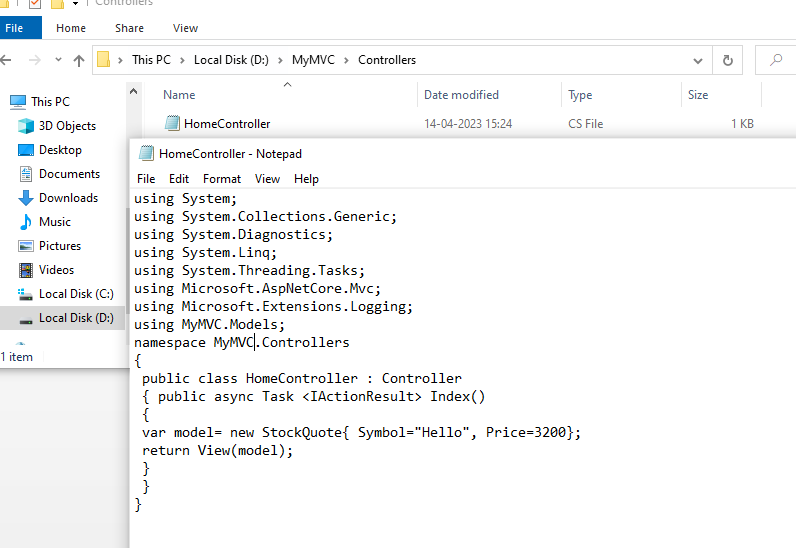
1. Now go back to command prompt and stop running project using CTRL+C



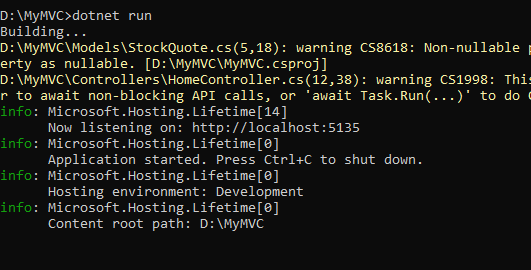
1. Go to models folder and add new file StockQuote.cs to it with following content

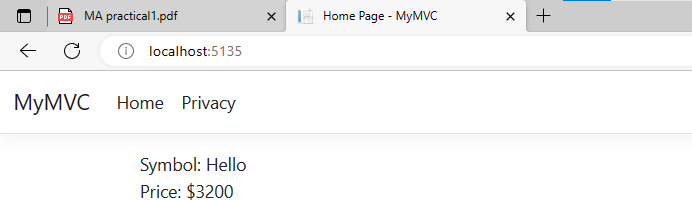
1. Now Add View to folder then home folder in it and modify index.cshtml file to match following
2. Now modify HomeController.cs file to match following:



1. Now run the project using

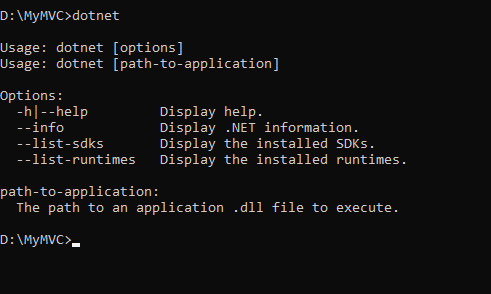


1. Now go back to browser and refresh to get modified view response



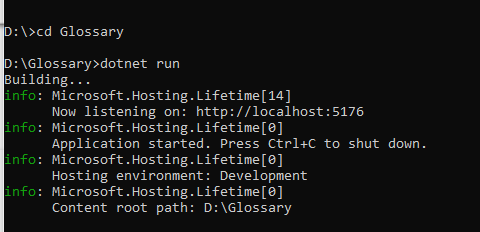
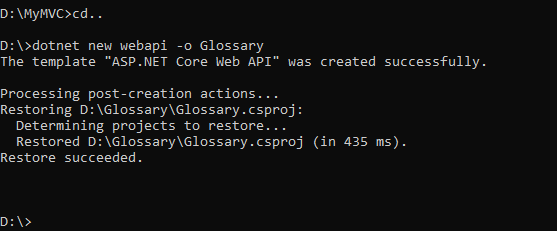
## PRACTICAL NO. : 02

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

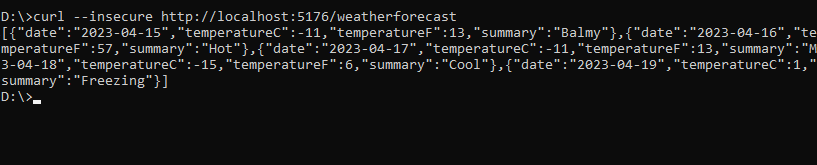
* 1. Download and install dotnet
  2. Check everything installed correctly

Create your web API

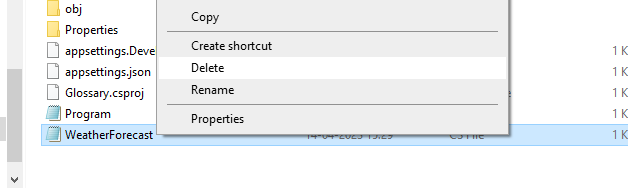
1. Open two command prompts Command prompt 1: Command: dotnet new webapi -o Glossary

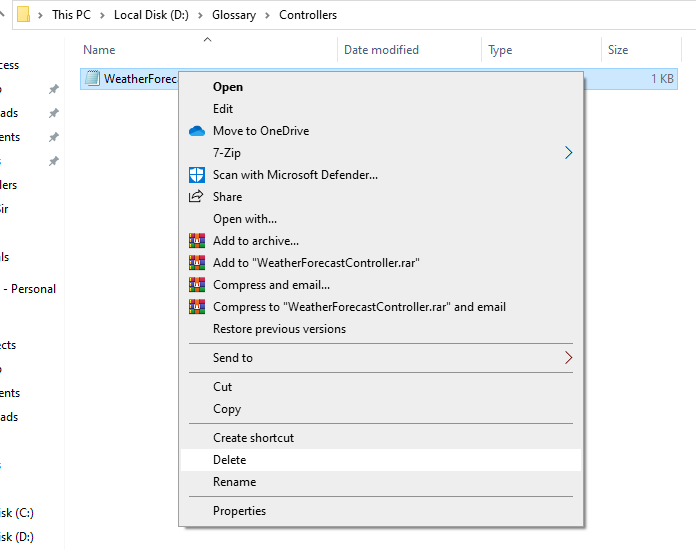


1. Command Prompt 2: (try running ready made weatherforecast class for testing) Command: curl --insecure https://localhost:5001/weatherforecast

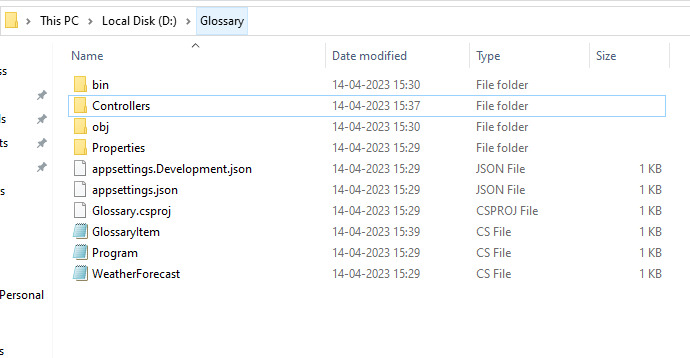
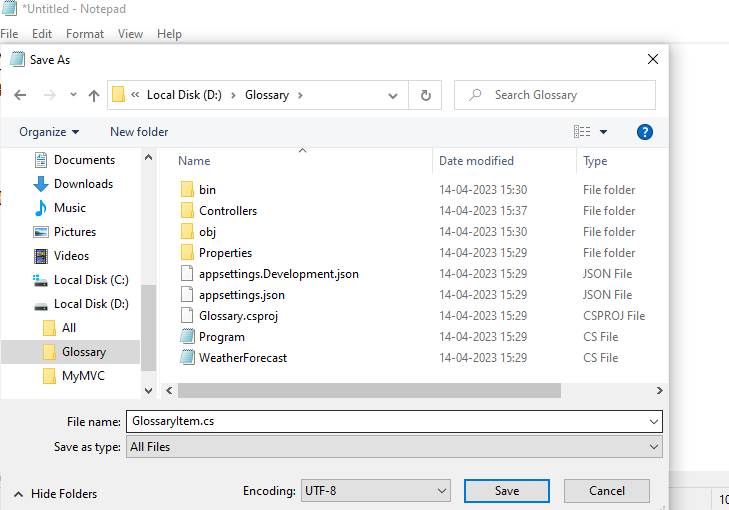


1. Now Change the content: To get started, remove the WeatherForecast.cs file from the root of the project and the WeatherForecastController.cs file from the Controllers folder. Add Following two files

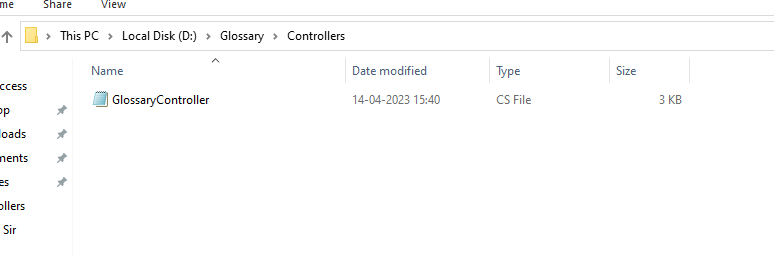




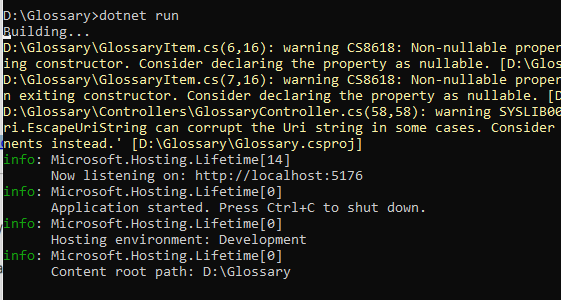
* 1. D:\Glossary\GlossaryItem.cs (type it in notepad and save as all files)



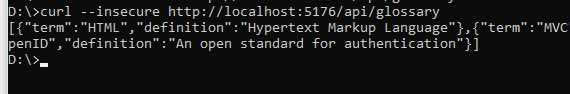
* 1. D:\Glossary\Controllers\ GlossaryController.cs (type it in notepad and save as all files)

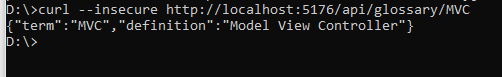


3.Now stop running previous dotnet run on command prompt 1 using Ctrl+C. and Run it again for new code. On Command prompt1:

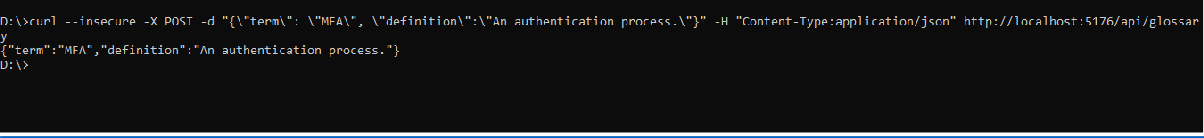


On Command prompt2:

1. Getting a list of items: Command: curl --insecure http://localhost:5176/api/glossary
2. Getting a single item Command: curl --insecure ttp://localhost:5176/api/glossary/MVC

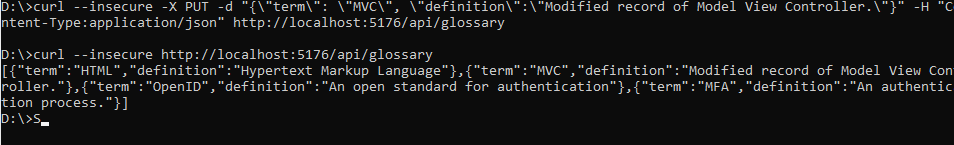


1. Creating an item Command: curl --insecure -X POST -d "{\"term\": \"MFA\", "definition\":\"An uthentication process.\"}" -H "ContentType:application/json" http://localhost:5176/api/glossary



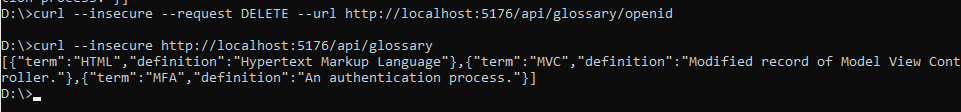
1. Update Item Command:

curl --insecure -X PUT -d "{\"term\": \"MVC\", \"definition\":\"Modified record of Model View Controller.\"}" -H "Content-Type:application/json" http://localhost:5176/api/glossary



1. Delete Item:

curl --insecure --request DELETE --url http://localhost:5176/api/glossary/openid

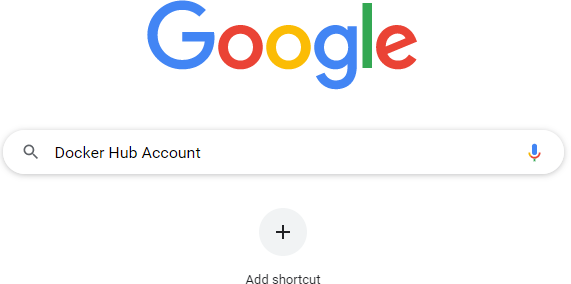


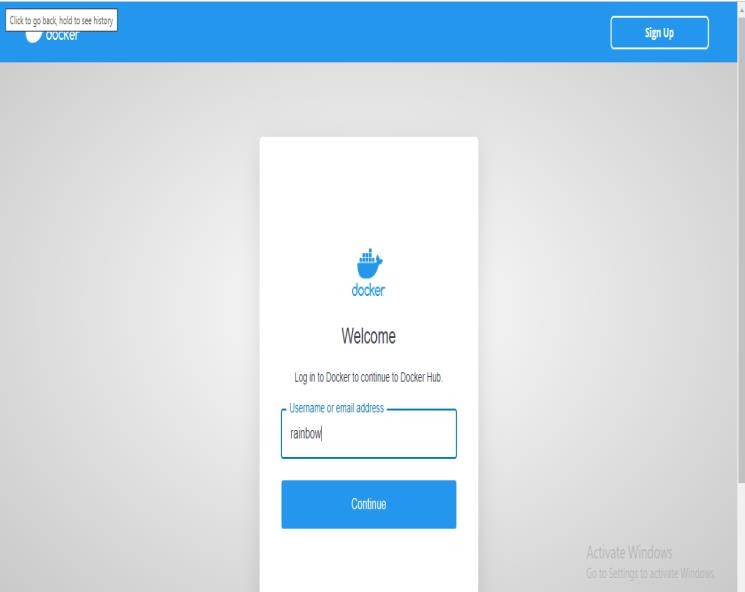
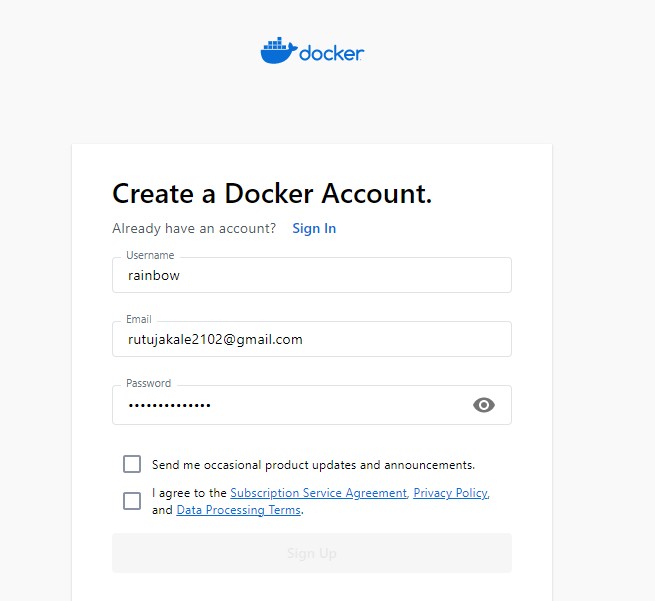
## Practical No.:03

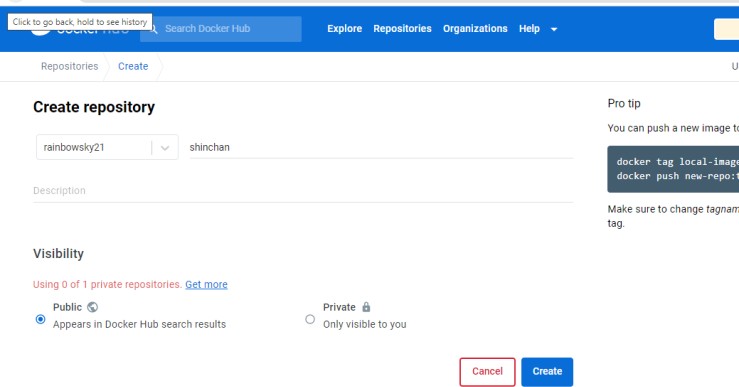
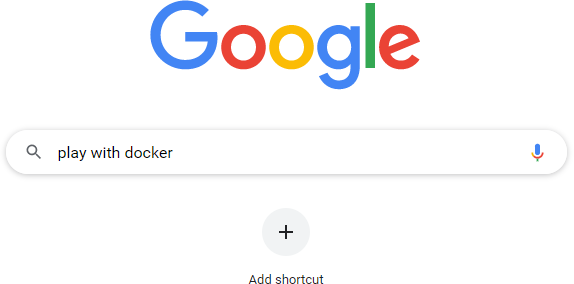
**Aim:- Working with docker, docker commands, docker images & Containers.**

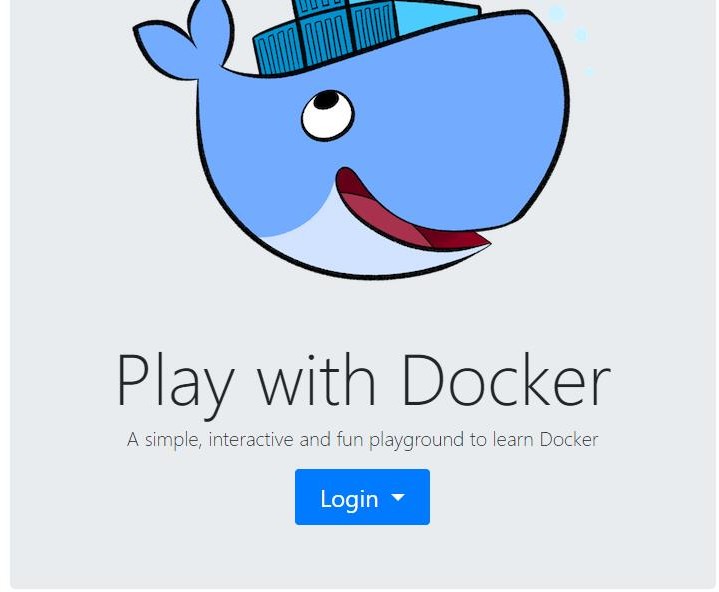
Step 1: Create Docker Hub Account from <https://hub.docker.com/>

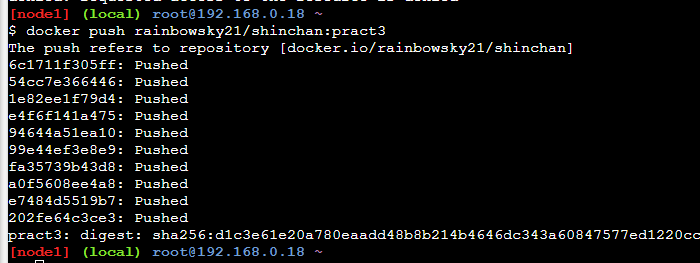
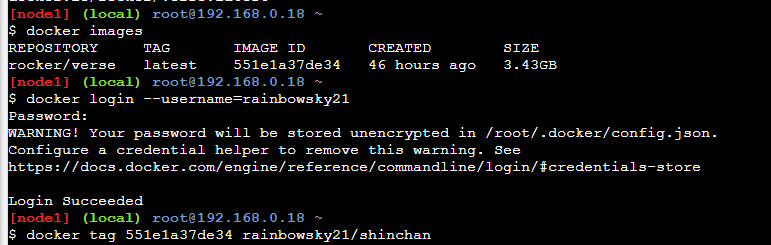
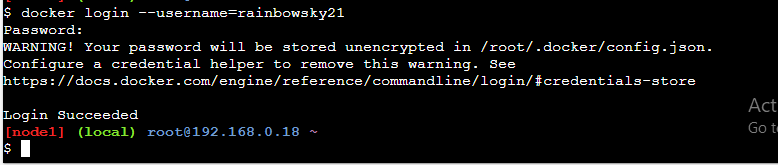
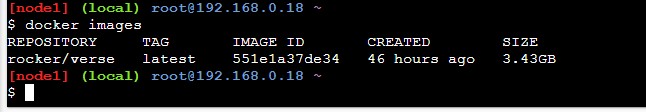
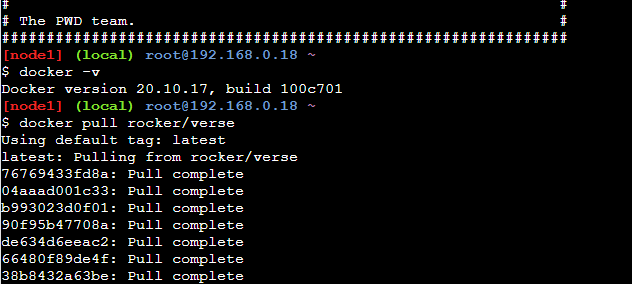
Step 2: Navigate to Play with Docker from <https://labs.play-with-docker.com/> Step 3: Click on create Instance and enter the following command.

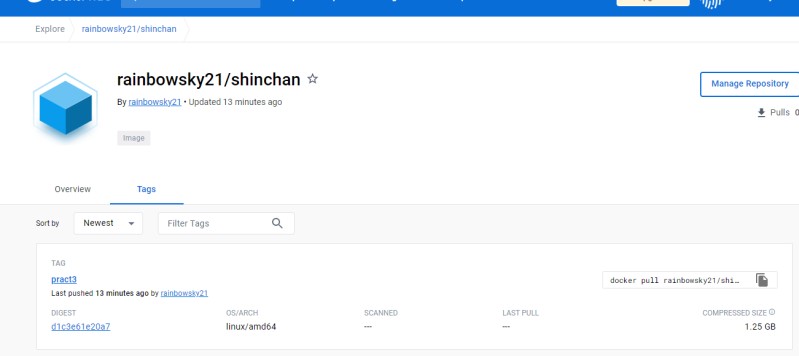
1. docker -v
2. docker –version



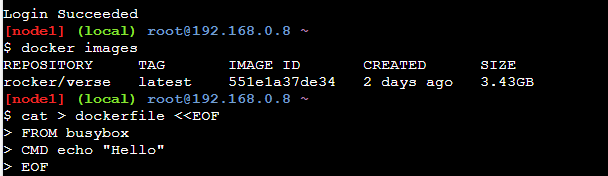
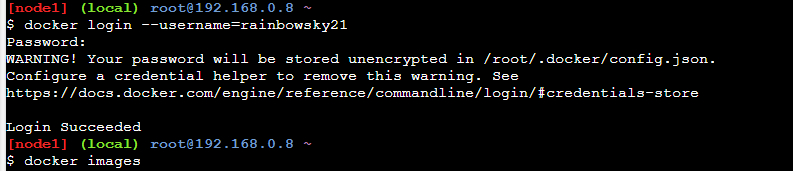




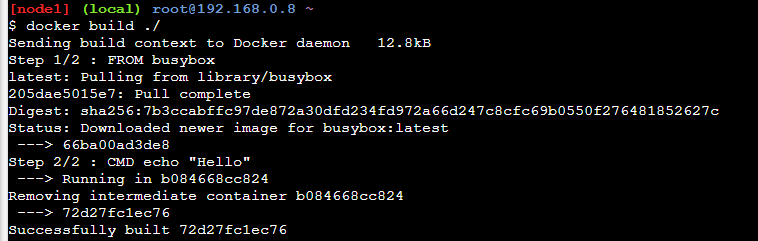


### 3B) Build your own image file from docker file & push & pull & run the file.

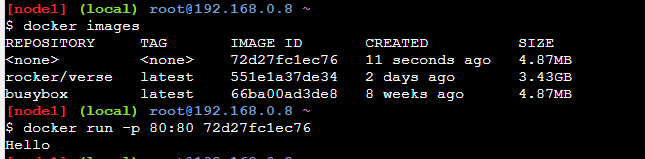
Step 1: Create Docker file.



Step 2: Enter command to build the ‘dockerfile’ created in step 1: docker build .\

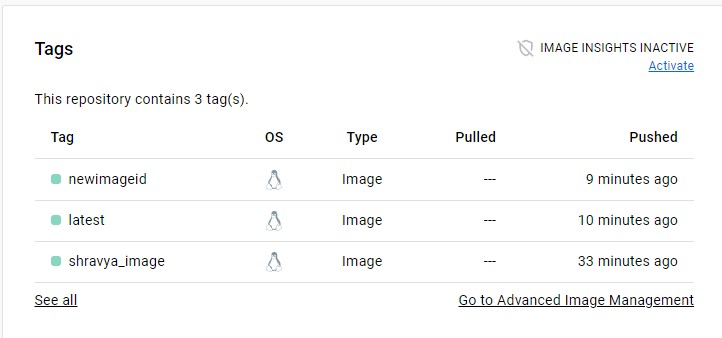


Step 3: List all the images from the docker: docker images Step 4: docker run -p 80:80 imageid



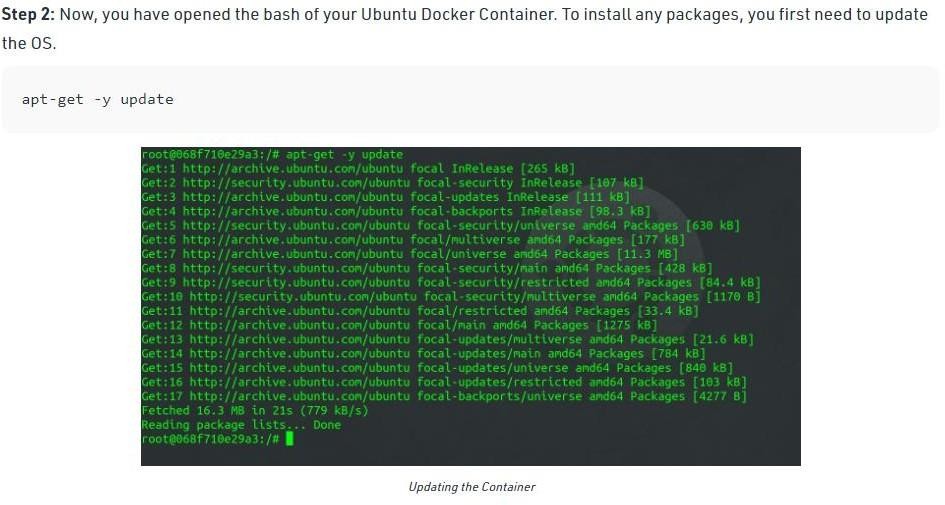
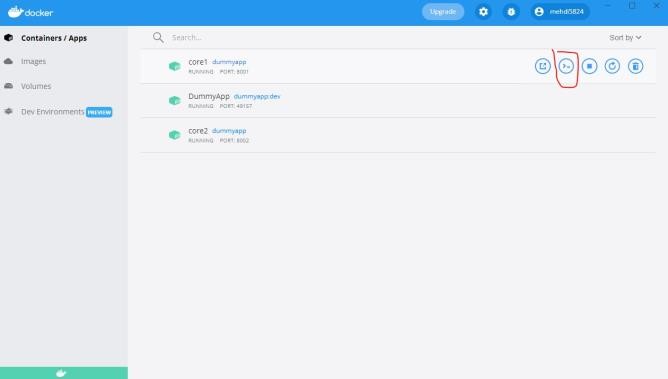
Step 5: Push the image to docker.

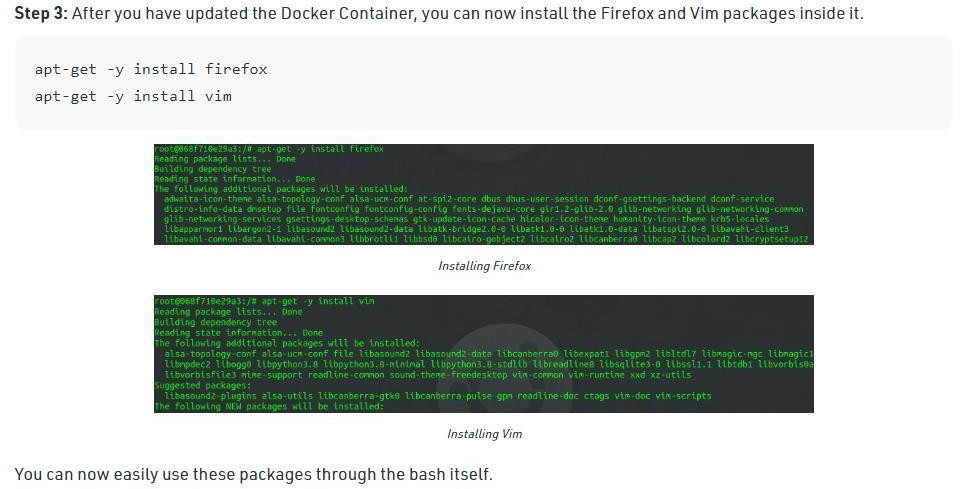
Step 6: Navigate to Docker repository and check the Image in Tags section



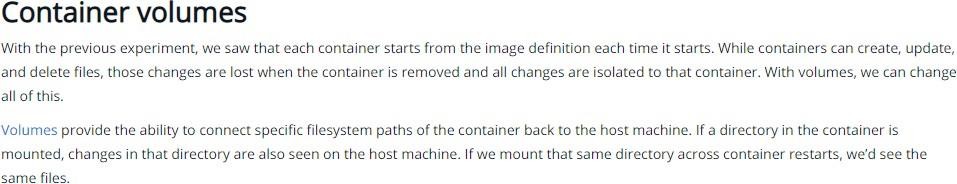
## Practical No.:04 Aim:- Installing s/w packages on docker.

Step 1 – Go to CLI Option on the container in Docker Desktop

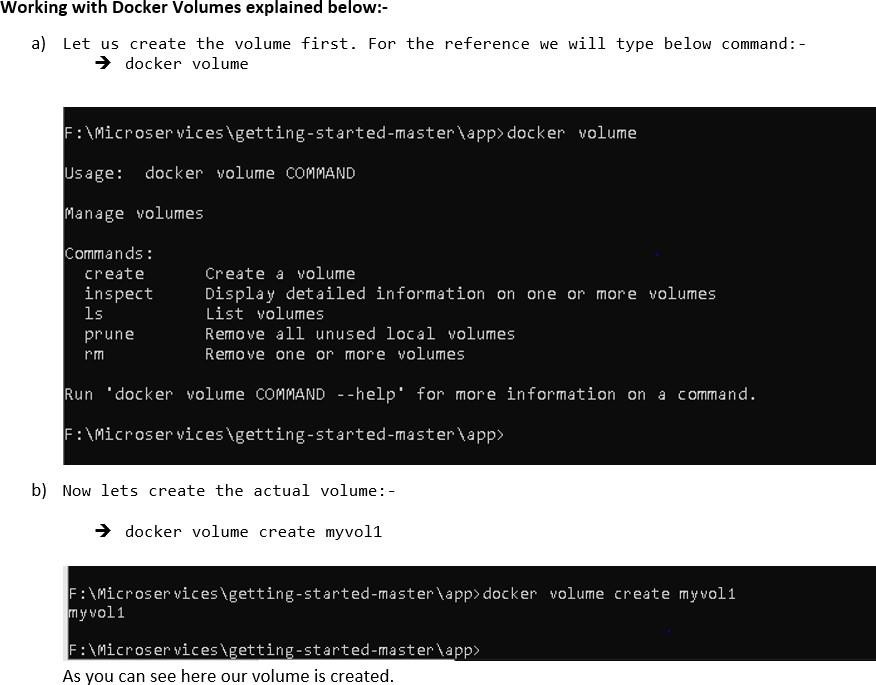
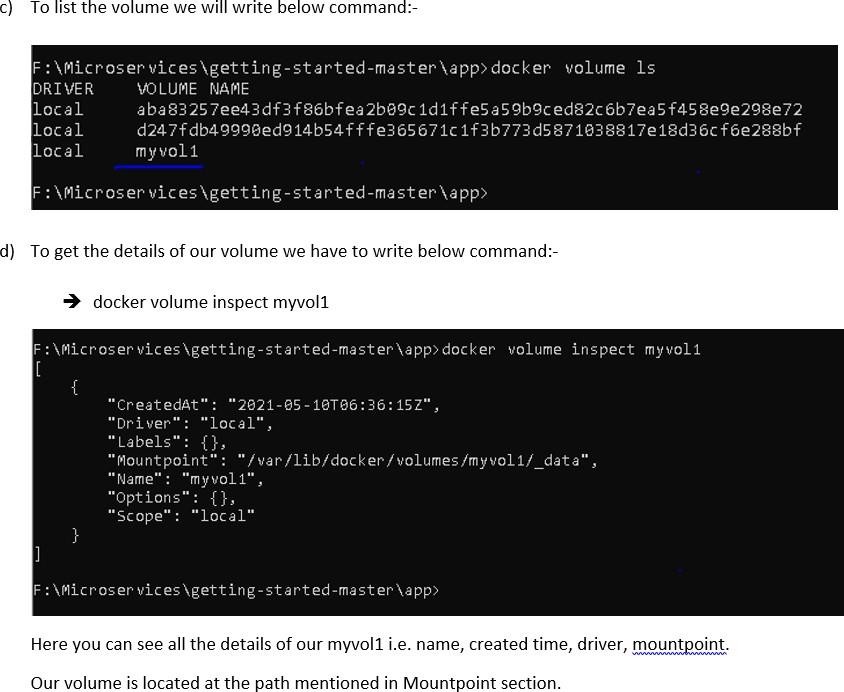


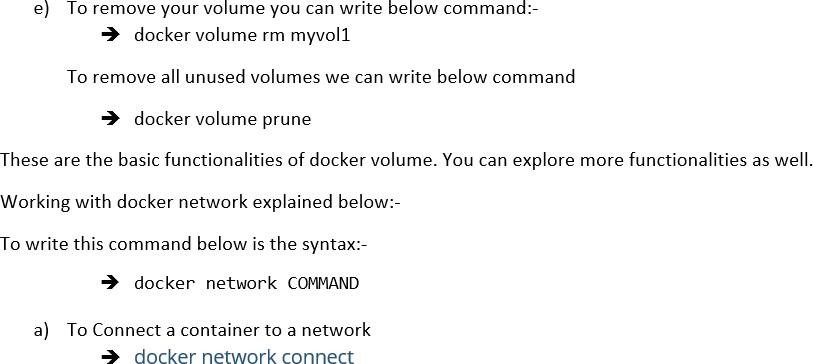


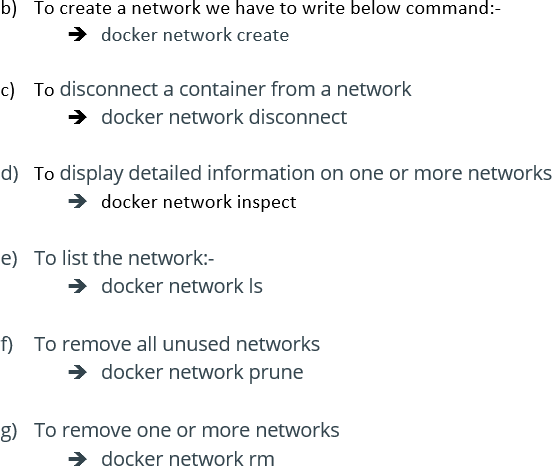
**Step 4:** Run vim to verify if the software package has been installed



**Step 1**







## Practical 05

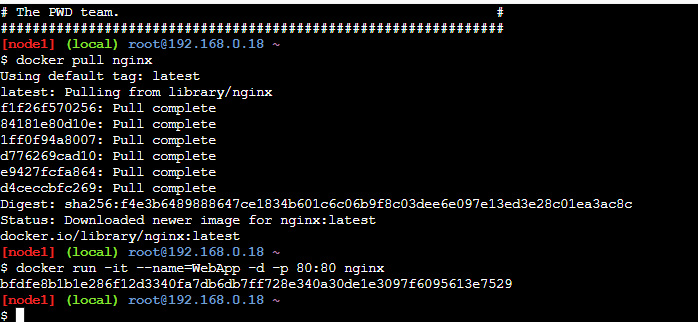
**Aim: Working with Docker Volumes and Networks.**

Pre-Requisites:

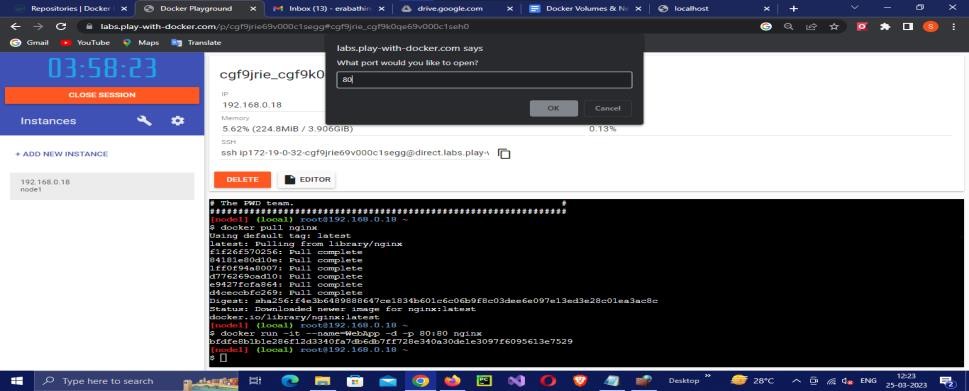
* 1. Open Windows Firewall
  2. Click on Advanced Security
  3. Click on Inbound Rules
  4. Create a New Rule
     1. Which type of rule would you like to create → port
     2. Does this rule apply the local ports or specific local ports
     3. Select Specific local ports - 80
     4. What action should be taken when a connection matches the specified conditions? - Allow the connection
     5. When does this apply? – Domain, Private, Public
     6. Name: ReportServer
     7. Description: ReportServer

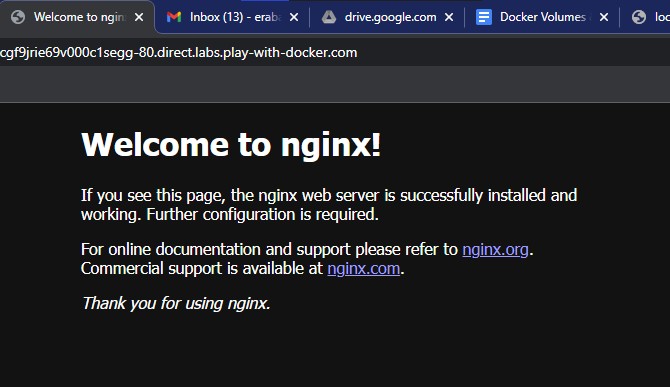
Step 1: Enter the following Commands

1. docker pull nginx – nginx : nginx is the image which is already available in docker
2. docker run -it –name=webApp -d -p 80:80 nginx: Create a webapp and run it with nginx image on port 80



Step 2: Click on Port and enter 80 in the dropdown window and click OK.

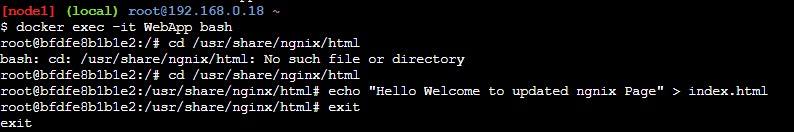


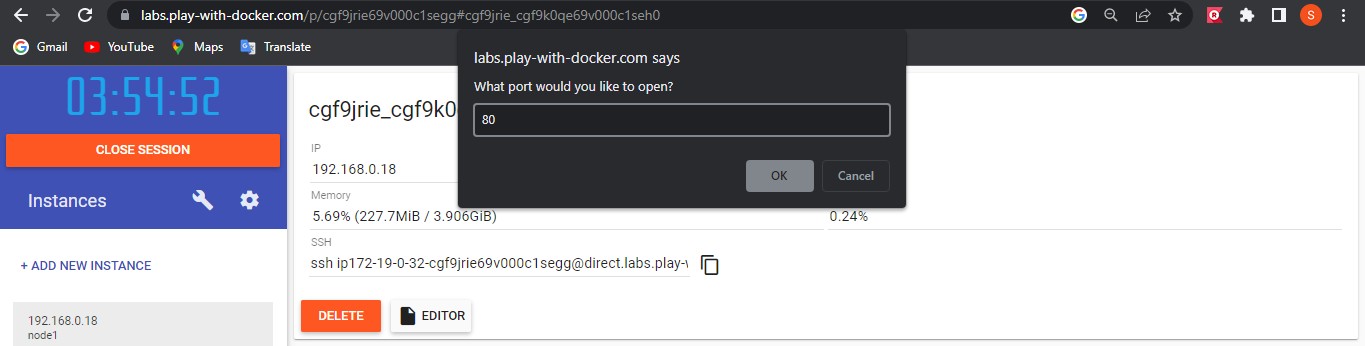
Output: The below webpage will be visible

Step 3: Enter the below command in order to enter into bash shell and then open port 80.

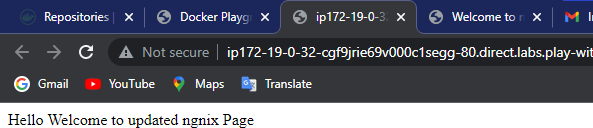
docker exec -it WebApp bash Cd /usr/share/nginx/html

Echo “Hello Welcome to updated nginx Page.”> index.html exit

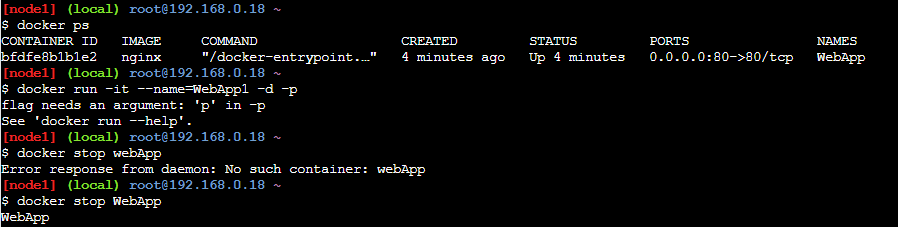


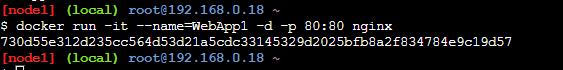


Output: The below webpage will be visible

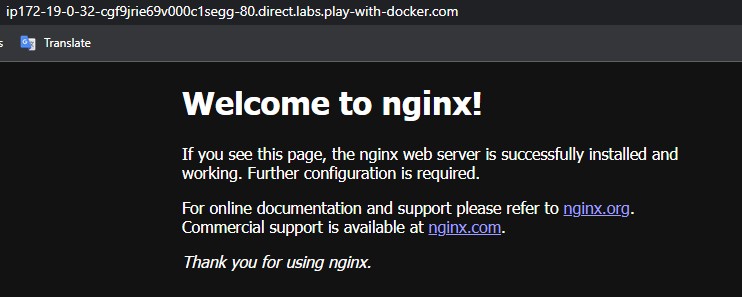


Step 4: List all the running containers: docker ps



Step 5: Create another container in Docker: WebApp1 Docker run -it –name=WebApp1 -d -p 80:80 nginx:

Step 6: Click on port and enter 80 in the dropdown and click ok Output: the welcome page of nginx should be visible

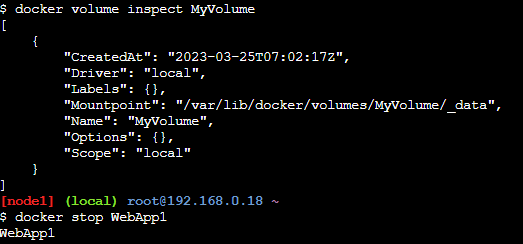
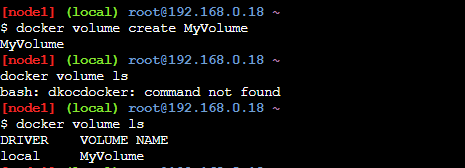


Problem: Updates made in one container is not reflected into another container. Solution: - Volume

Update made in one container within the volume will be reflected in all the containers of that volume.

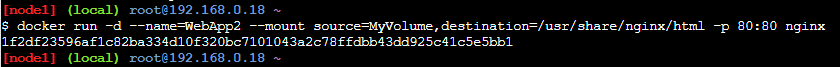
Step 7: Creation of Volume(MyVolume) Command:

1. docker volume create MyVolume
2. docker volume ls
3. docker volume inspect MyVolume
4. docker stop WebApp1

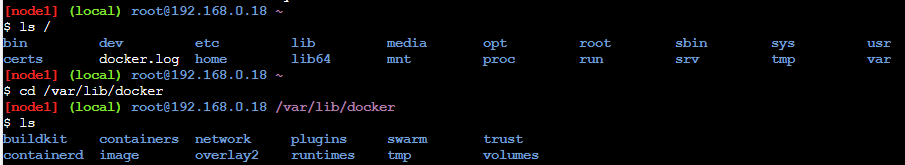


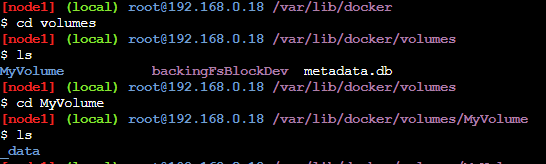
Step 8: Create a container (WebApp2) inside the container MyVolume

docker run -d –name = WebApp2 –mount source=MyVolume,destination=/usr/share/nginx/html –p 80:80 nginx



Step 9: Enter the below commands:

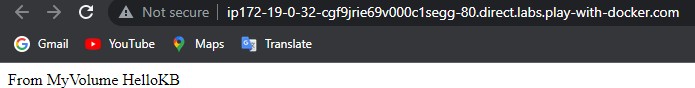
1. ls /
2. cd /var/lib/docker
3. ls
4. cd volumes
5. ls
6. cd MyVolume
7. ls
8. cd \_data



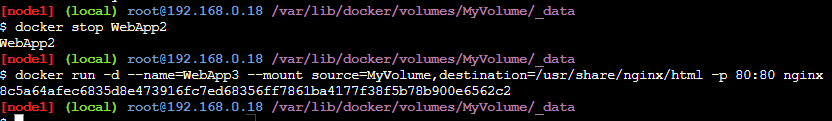
Step 10:Edit the index file with the below content to “Display the content on the Webpage”

Text  Description automatically generated

Open Port 80

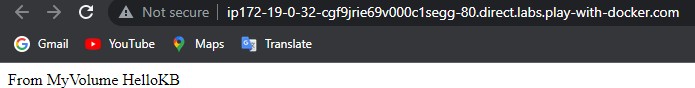


Step 11: Stop the above container (WebApp2) and Create another container within the volume (MyVolume)



Open port 80

Output: The edits made in one container of the volume will be reflected in all the containers of that volume

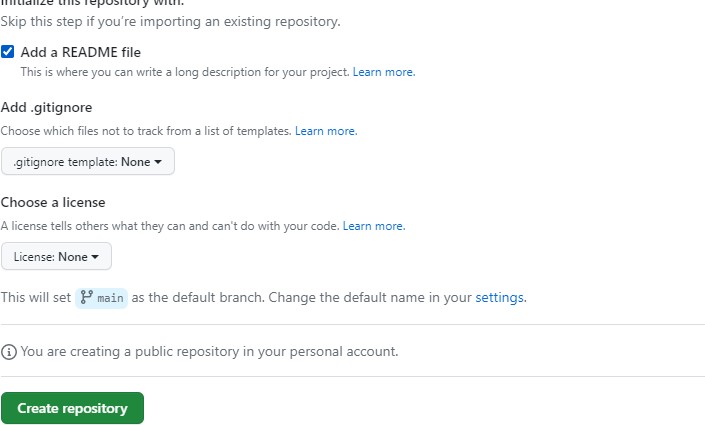
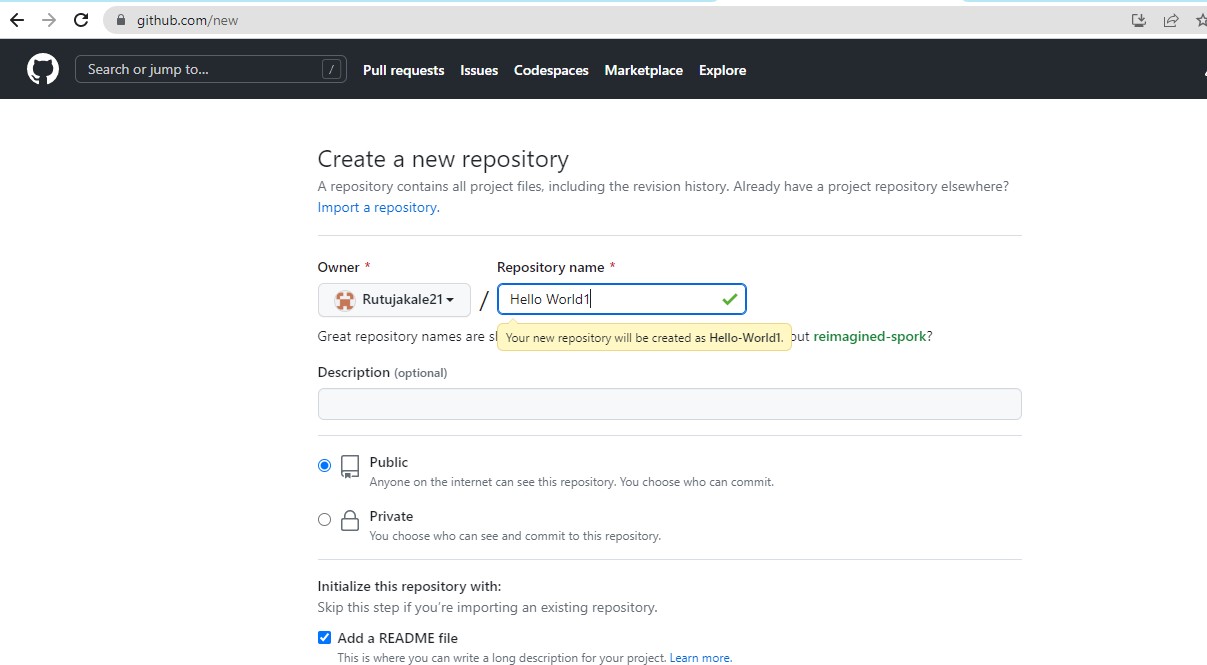


## Practical 6

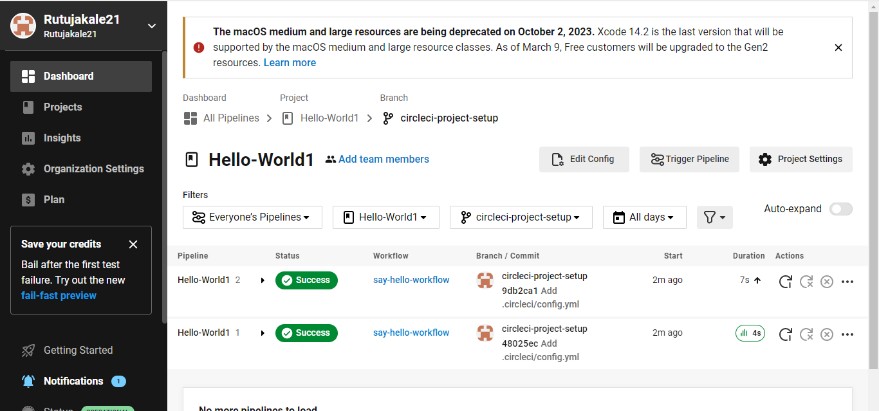
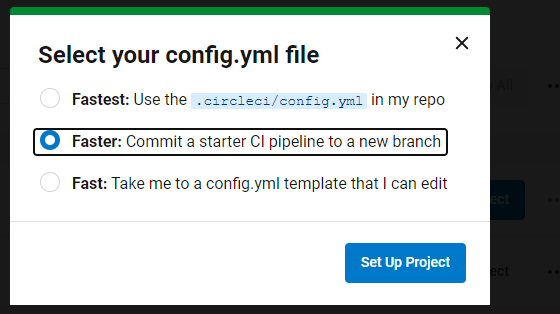
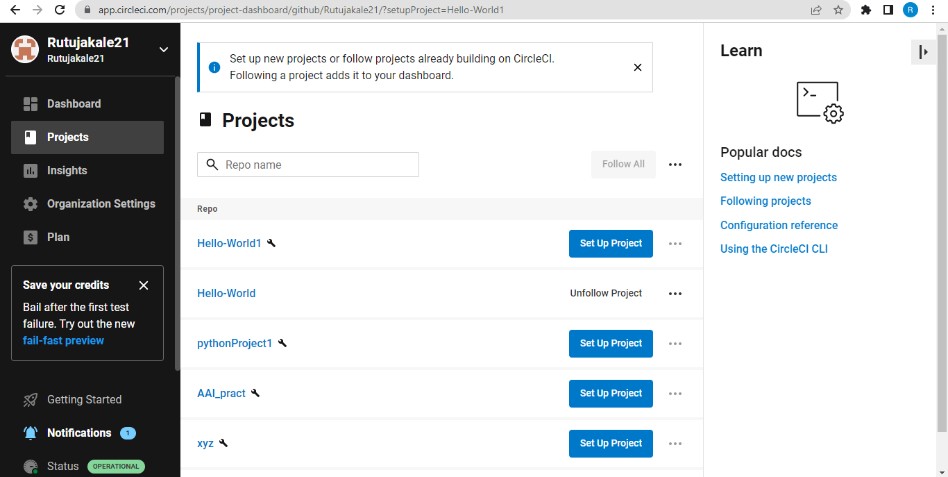
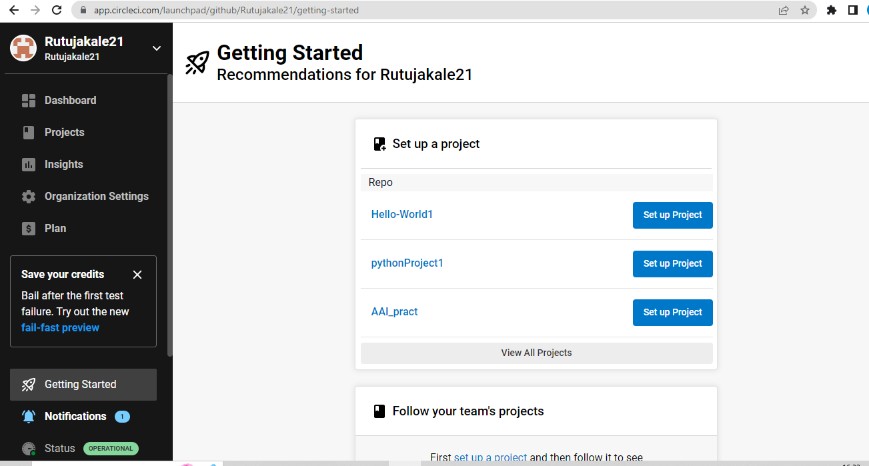
**Aim: Working with Circle CI for continuous integration**

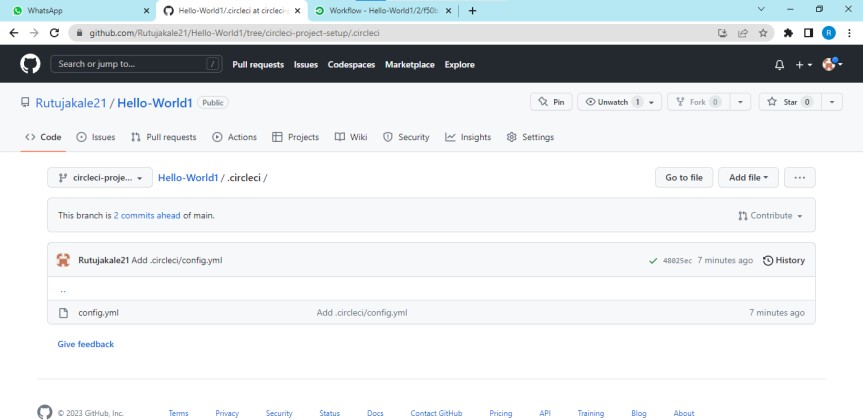
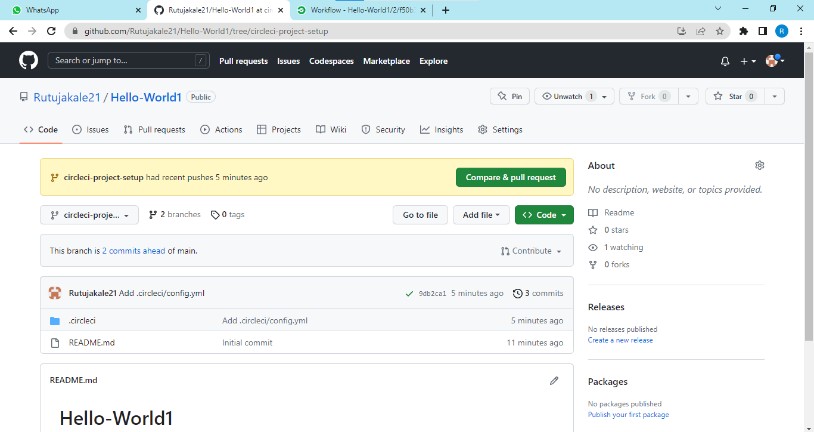
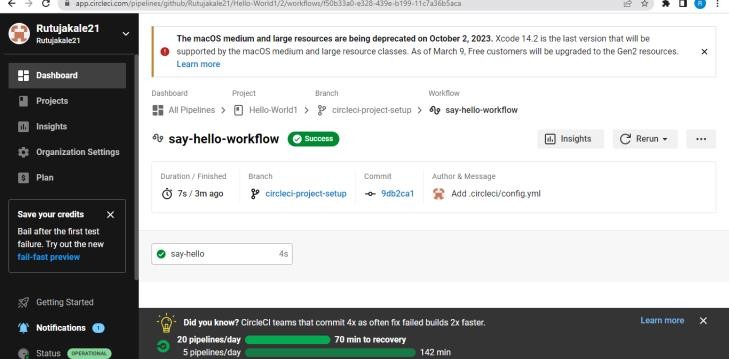
Step 1 - Create a repository

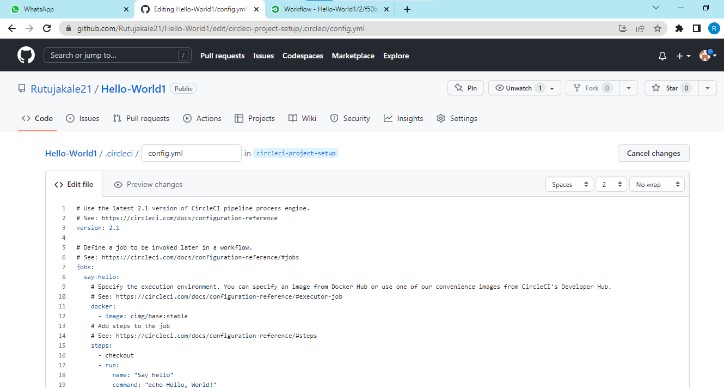
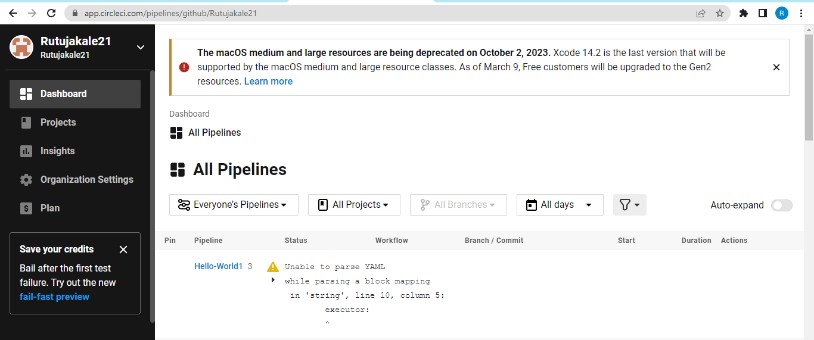
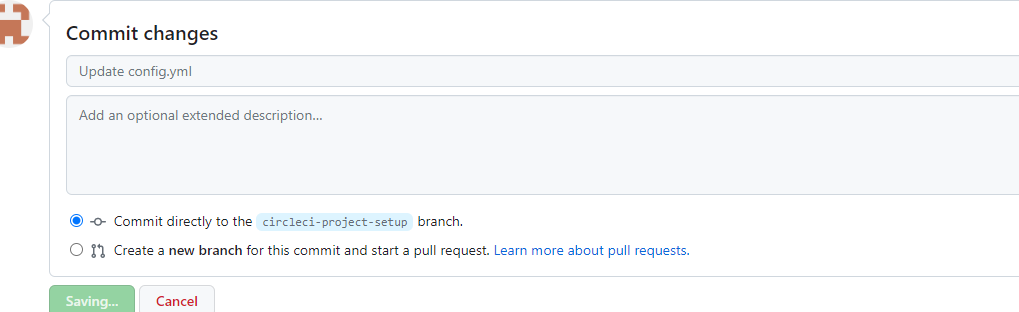
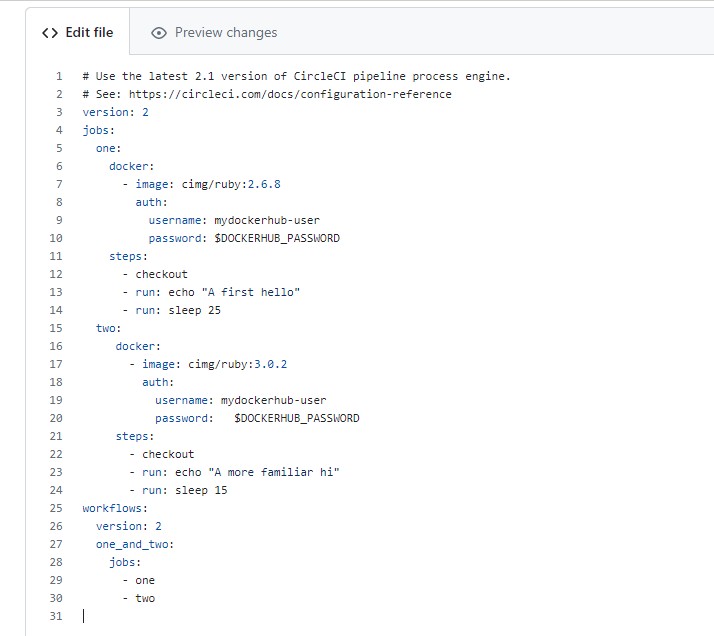
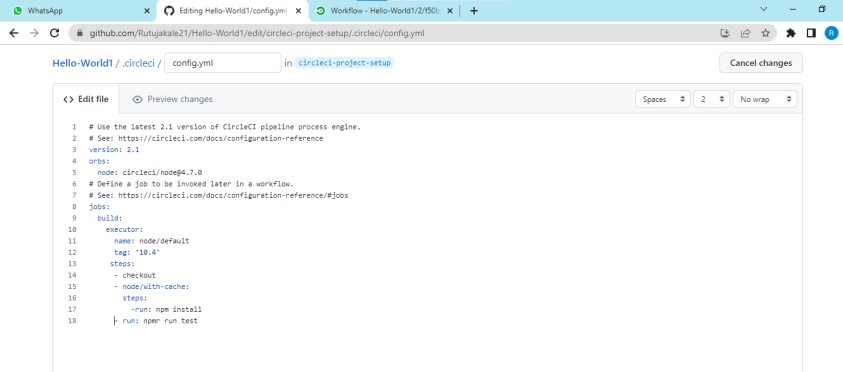
1. Log in to GitHub and begin the process to create a new repository.
2. Enter a name for your repository (for example, hello-world).
3. Select the option to initialize the repository with a README file.
4. Finally, click Create repository.
5. There is no need to add any source code for now

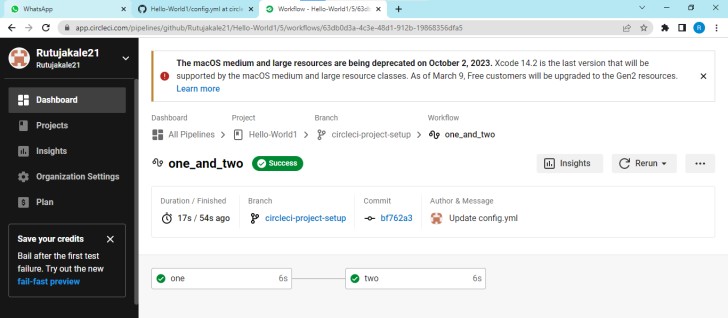
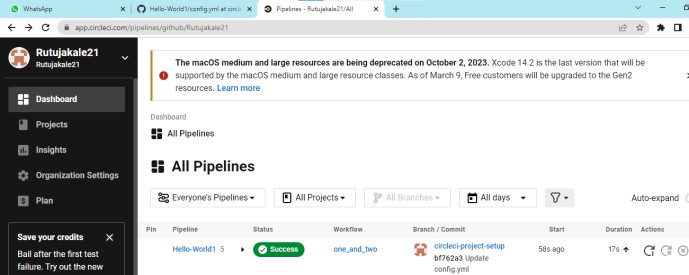
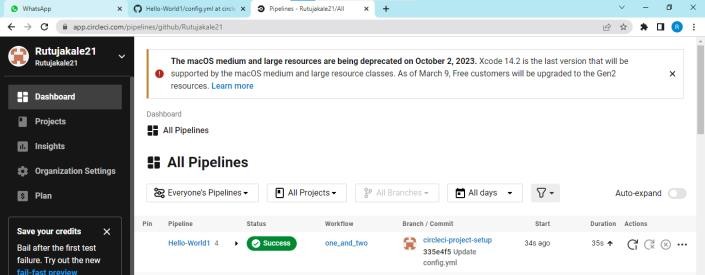
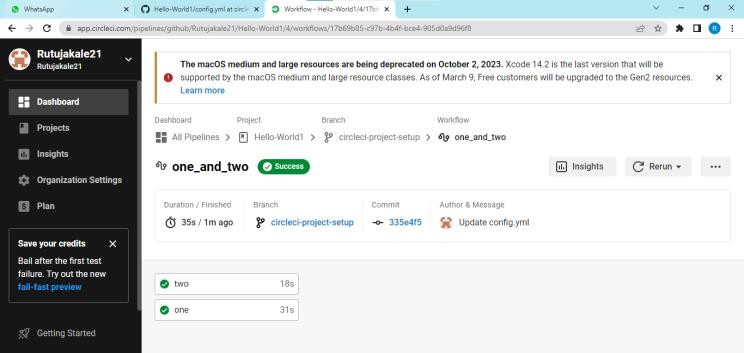


1. Login to Circle CI https://app.circleci.com/ Using GitHub Login, Once logged in navigate to Projects.





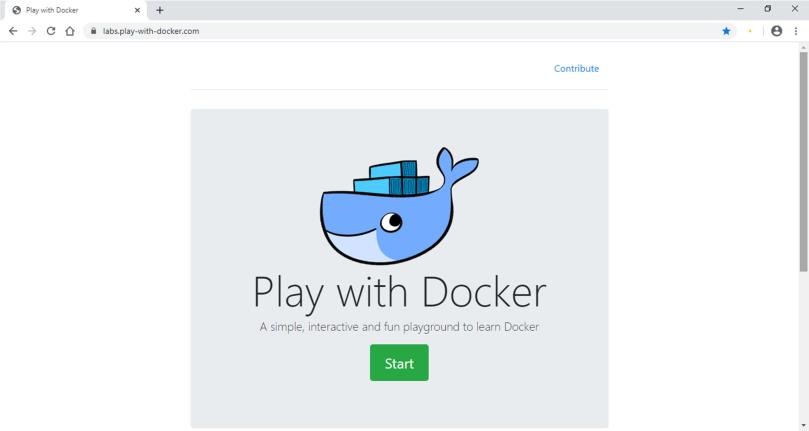
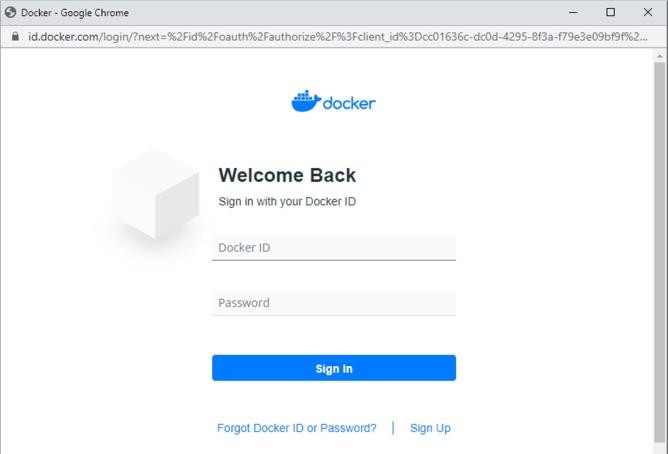
 



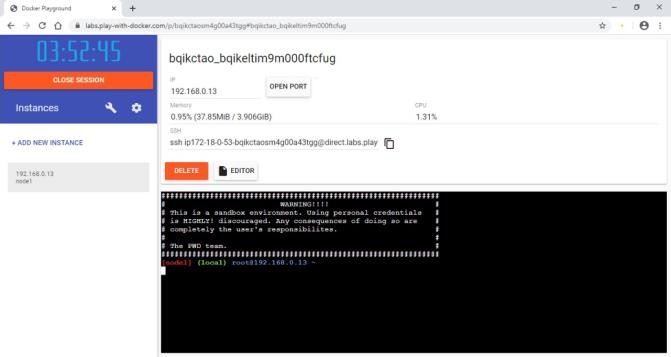
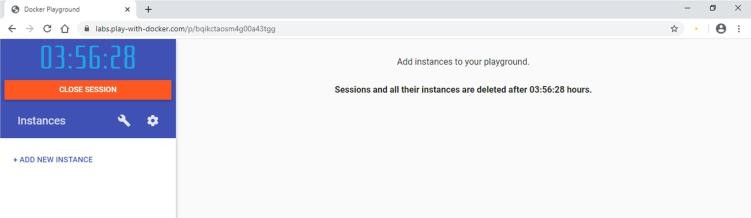
## Practical No.:07

### Aim:- Creating Backing service with ASP.NET 2.0 core.

Now login in to Play-With-Docker

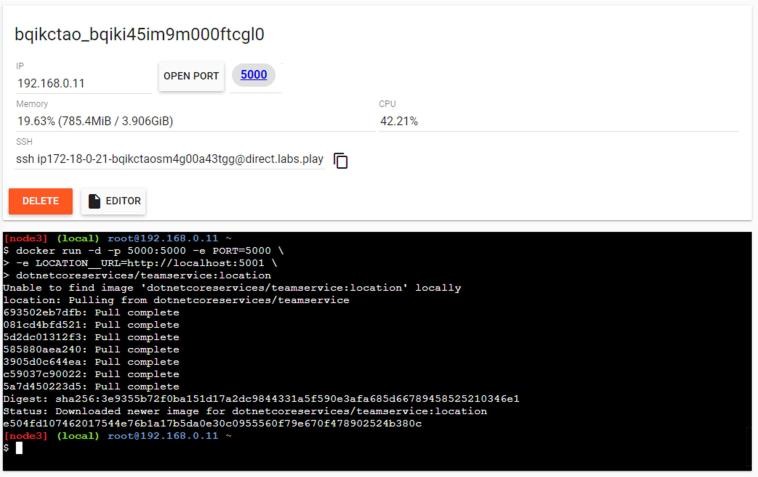


Click on Add New Instance



Start typing following commands

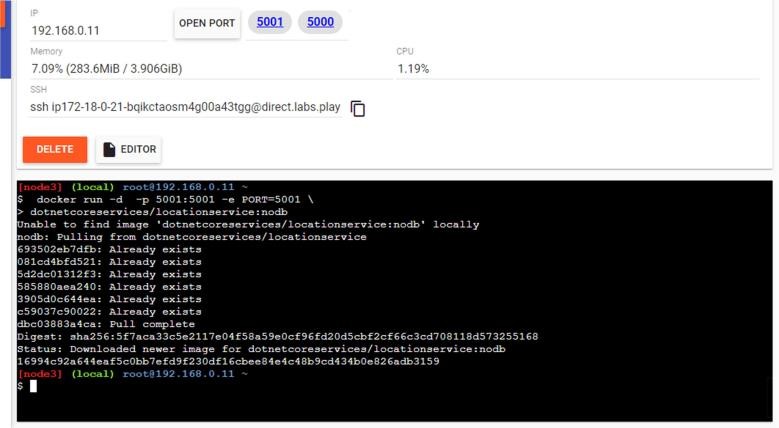
Command :docker run -d -p 5000:5000 -e PORT=5000 \-e LOCATION URL=http://localhost:5001 \dotnetcoreservices/teamservice:location

output: (you can observe that it has started port 5000 on top)

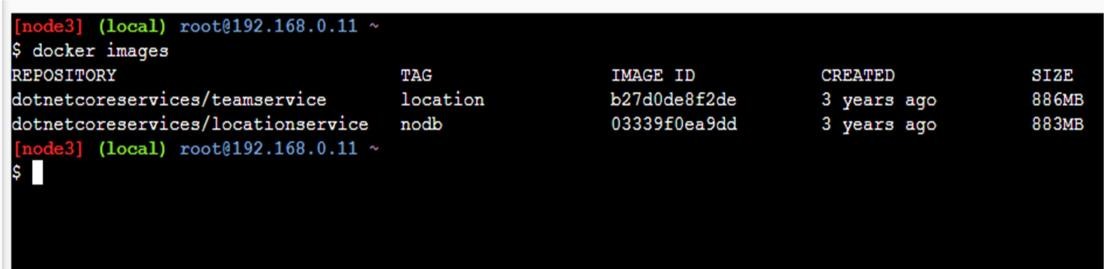
Command: to run location service

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

output: (now it has started one more port that is 5001 for location service)

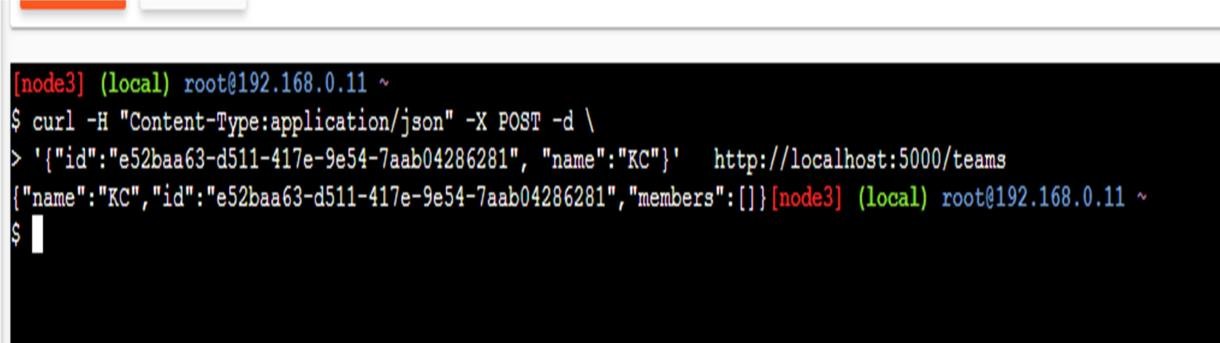


Command : to check running images in docker $docker images output:

Command: to create new team

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

Output:Command :To confirm that team is added

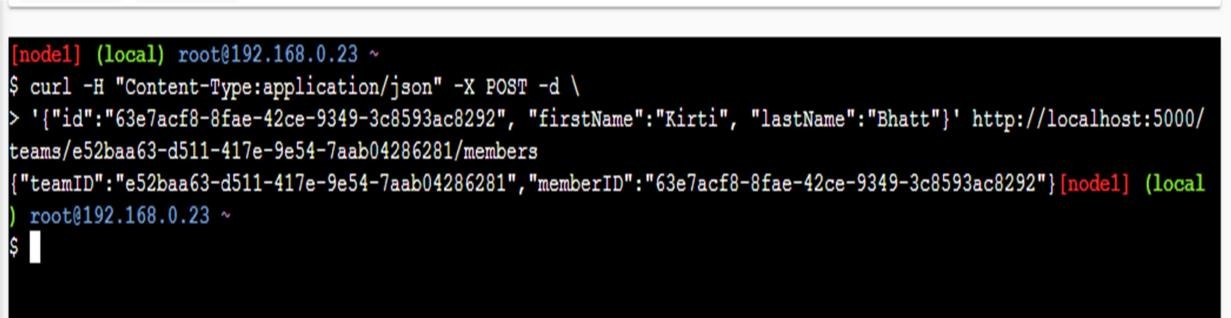


curl http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281 Output

Command : to add new member to team

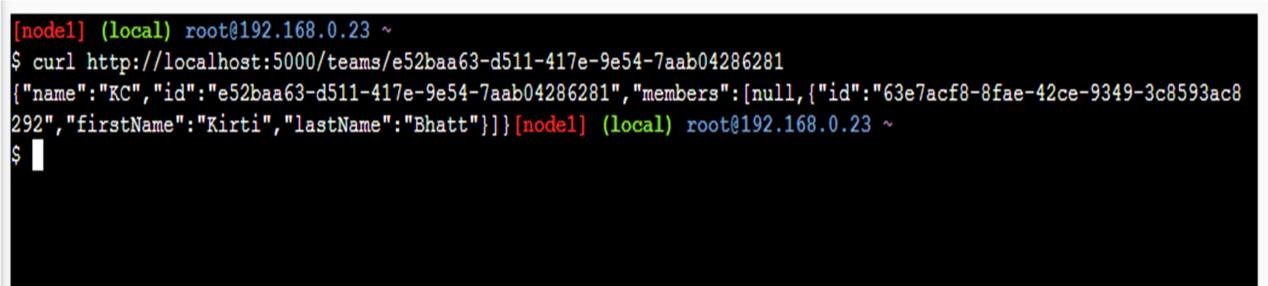
curl -H "Content-Type:application/json" -X POST -d \'{"id":"63e7acf8-8fae- 42ce-9349-3c8593ac8292", "firstName":"Kirti", "lastName":"Bhatt"}' http://localhost:5000/teams/e52baa63-d511-417e-9e54- 7aab04286281/members

Output:



Command :To confirm member added

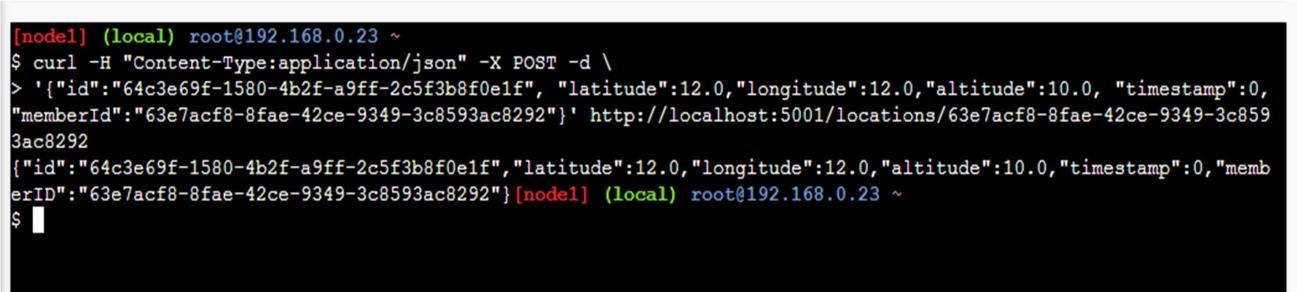
curl http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281

output:

Command : To add location for member

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

output:



Command : To confirm location is added in member

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