

Practical 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 D: drive or any other drive

3) open command prompt and perform following operations

Command: to create mvc project

```
dotnet new mvc --auth none
```

output:

```
Command Prompt
D:\>cd mymvc

D:\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 D:\MyMVC\MyMVC.csproj...
  Restore completed in 1.93 sec for D:\MyMVC\MyMVC.csproj.

Restore succeeded.
```

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

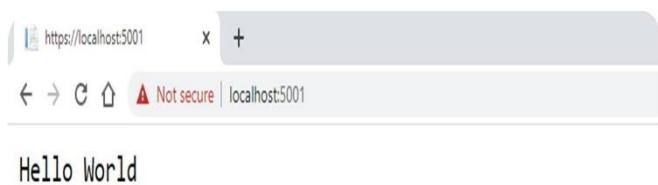
```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using MyMVC.Models;
```

```
namespace MyMVC.Controllers
{
    public class HomeController : Controller
    {
        public String Index()
        {
            return "Hello World";
        }
    }
}
```

5) Run the project

```
Command Prompt - dotnet run
D:\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: D:\MyMVC
```

Now open browser and type URL: localhost:5000



5. Now go back to command prompt and stop running project using CTR+C

```
D:\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: D:\MyMVC
info: Microsoft.Hosting.Lifetime[0]
      Application is shutting down...

D:\MyMVC>
```

A screenshot of a Windows Command Prompt window titled 'Command Prompt'. The window contains the output of a 'dotnet run' command. It shows the application starting up, listening on both https://localhost:5001 and http://localhost:5000, and then shutting down. The command prompt prompt 'D:\MyMVC>' is visible at the bottom.

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

```
using System;
namespace MyMVC.Models
{
    public class StockQuote
    {
        public string Symbol {get;set;}
        public int Price{get;set;}
    }
}
```

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

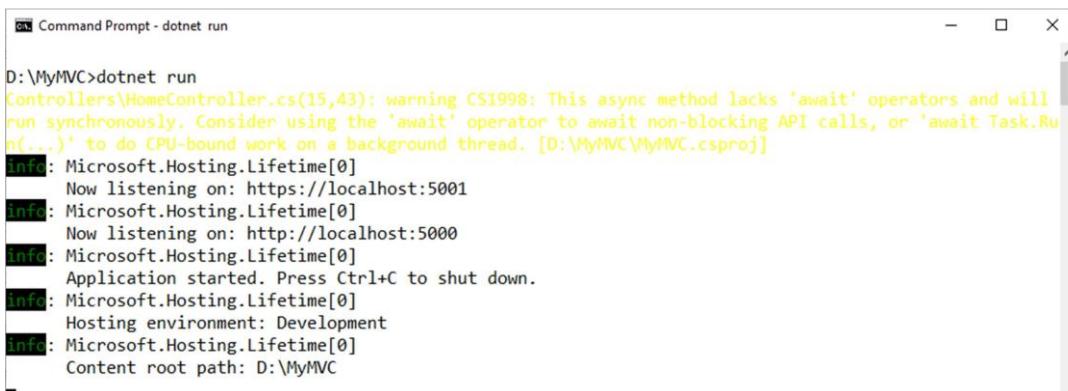
```
@{
    ViewData["Title"] = "Home Page";
}
<div>
    Symbol: @Model.Symbol <br/>
    Price: $@Model.Price <br/>
</div>
```

7) Now modify HomeController.cs file to match following:

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

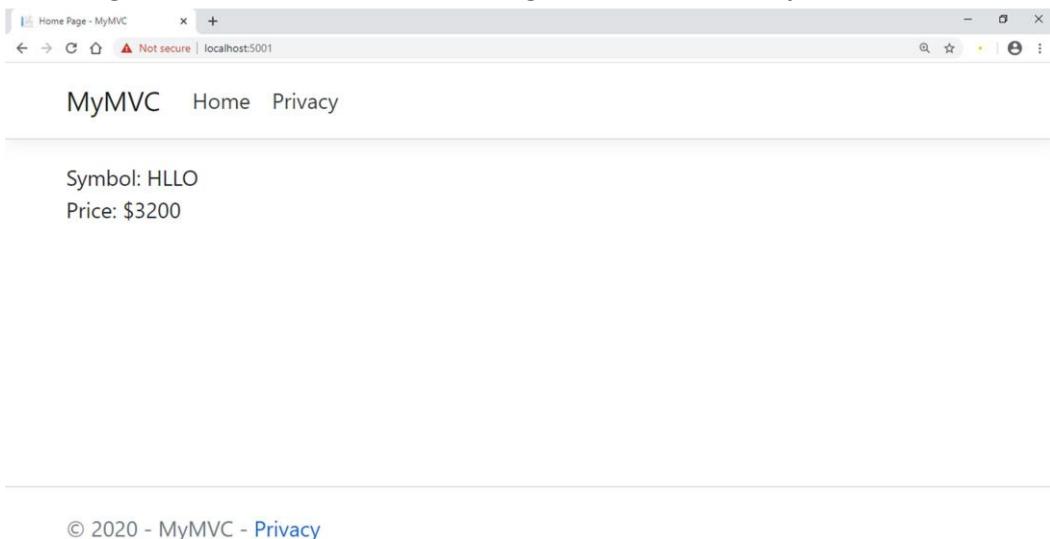
namespace MyMVC.Controllers
{
    public class HomeController : Controller
    {
        public async Task <ActionResult> Index()
        {
            var model= new StockQuote{ Symbol='HLLO', Price=3200};
            return View(model);
        }
    }
}
```

**8. Now run the project using
dotnet run**



```
D:\MyMVC>dotnet run
Controllers\HomeController.cs(15,43): warning CS1998: This async method lacks 'await' operators and will
run synchronously. Consider using the 'await' operator to await non-blocking API calls, or 'await Task.Ru
nt...)' to do CPU-bound work on a background thread. [D:\MyMVC\MyMVC.csproj]
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: D:\MyMVC
```

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



Practical 2:

Aim: Building ASP.Net core REST API

Software requirement:

a. 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>

b. Check everything installed correctly

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

Command prompt

> dotnet

Create your web API

1. Open two command prompts

Command prompt 1:

Command:

dotnet new webapi -o Glossary

output:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\HP\OneDrive\Desktop> 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...
  Restore completed in 1.13 sec for D:\Glossary\Glossary.csproj.

Restore succeeded.
```

Command:

cd Glossary

dotnet run

Output:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\HP\OneDrive\Desktop> cd Glossary

D:\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: D:\Glossary
```

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

Command:

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

output:

```
Command Prompt
Microsoft Windows [Version 10.0.18362.175]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Admin>d:

D:\>curl --insecure https://localhost:5001/weatherforecast
[{"date": "2020-04-17T21:07:12.4769001+05:30", "temperatureC": 10, "temperatureF": 49, "summary": "Hot"}, {"date": "2020-04-18T21:07:12.478667+05:30", "temperatureC": -2, "temperatureF": 29, "summary": "Hot"}, {"date": "2020-04-19T21:07:12.4787101+05:30", "temperatureC": 29, "temperatureF": 84, "summary": "Warm"}, {"date": "2020-04-20T21:07:12.4787134+05:30", "temperatureC": 29, "temperatureF": 84, "summary": "Balmy"}, {"date": "2020-04-21T21:07:12.4787152+05:30", "temperatureC": 13, "temperatureF": 55, "summary": "Chilly"}]
D:\>
```

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

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; }
    }
}
```

2) 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);
    }
}
```

[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 non-existing term.");
    }
    else
    {
        existingGlossaryItem.Definition = glossaryItem.Definition;
        return Ok();
    }
}
```

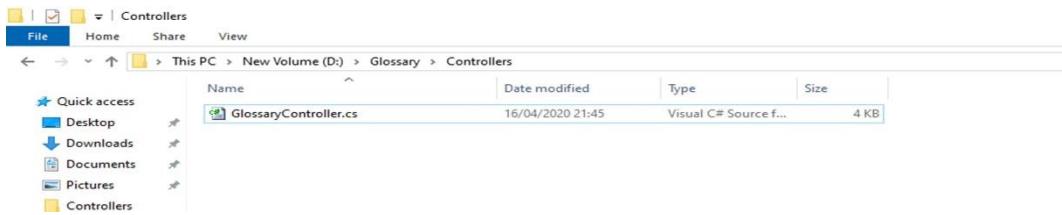
[HttpDelete]

```
[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:

Name	Date modified	Type	Size
bin	16/04/2020 21:04	File folder	
Controllers	16/04/2020 21:22	File folder	
obj	16/04/2020 21:04	File folder	
Properties	16/04/2020 20:57	File folder	
appsettings.Development	16/04/2020 20:57	JSON File	1 KB
appsettings	16/04/2020 20:57	JSON File	1 KB
Glossary	16/04/2020 20:57	Visual C# Project f...	1 KB
GlossaryItem.cs	16/04/2020 21:12	Visual C# Source f...	1 KB
Program.cs	16/04/2020 20:57	Visual C# Source f...	1 KB
Startup.cs	16/04/2020 20:57	Visual C# Source f...	2 KB



3. 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:

```
D:\Glossary>dotnet run
C:\Program Files\dotnet\sdk\3.1.201\Microsoft.Common.CurrentVersion.targets(4643,5): warning MSB3026: Cou
ld not copy "D:\Glossary\obj\Debug\netcoreapp3.1\Glossary.exe" to "bin\Debug\netcoreapp3.1\Glossary.exe".
Beginning retry 1 in 1000ms. The process cannot access the file 'D:\Glossary\obj\Debug\netcoreapp3.1\Glo
ssary.exe' because it is being used by another process. [D:\Glossary\Glossary.csproj]
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: D:\Glossary
```

On Command prompt2:

1) Getting a list of items:

Command:

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

Output:

```
D:\>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"}]
D:\>
```

2) Getting a single item

Command:

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

Output:

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

2) 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
```

Output:


```
D:\>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."}
D:\>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"}, {"term":"MFA","definition":"An authentication process."}]
D:\>
```

4) 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:


```
D:\>curl --insecure -X PUT -d "{\"term\": \"MVC\", \"definition\":\"Modified record of Model View Controller.\"}" -H "Content-Type:application/json" https://localhost:5001/api/glossary
D:\>curl --insecure https://localhost:5001/api/glossary
[{"term":"HTML","definition":"Hypertext Markup Language"}, {"term":"MVC","definition":"Modified record of Model View Controller."}, {"term":"OpenID","definition":"An open standard for authentication"}, {"term":"MFA","definition":"An authentication process."}]
D:\>
```

5) Delete Item**Command:**

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

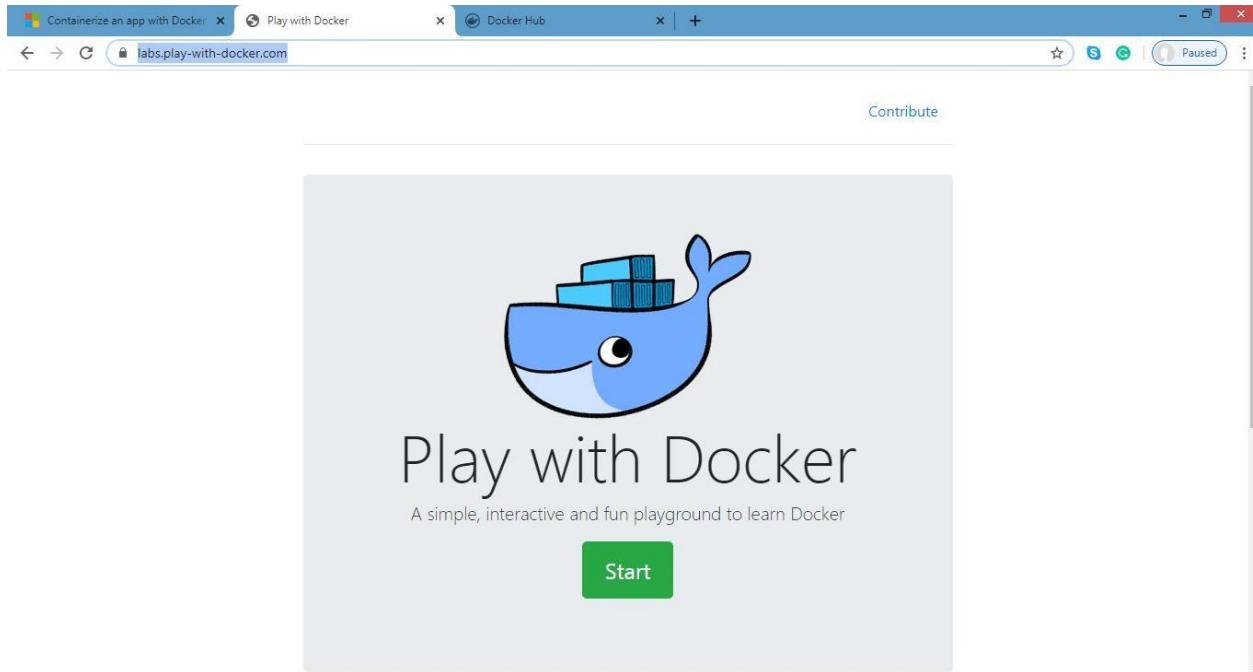
Output:


```
D:\>curl --insecure --request DELETE --url https://localhost:5001/api/glossary/openid
D:\>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."}]
D:\>
```

Practical No: 3

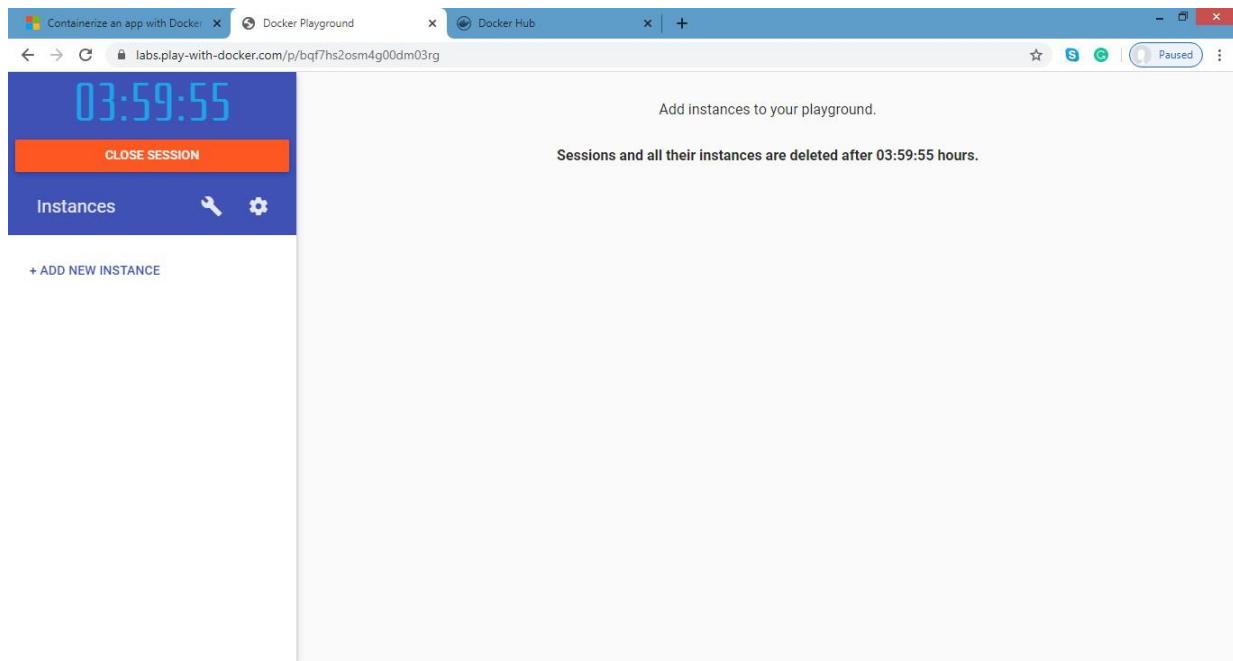
Aim: Working with Docker, Docker Commands, Docker Images&Containers.

- 1) create Docker Hub account (sign up)
- 2) login to <https://labs.play-with-docker.com/>



Click on start

- 3) add new instance



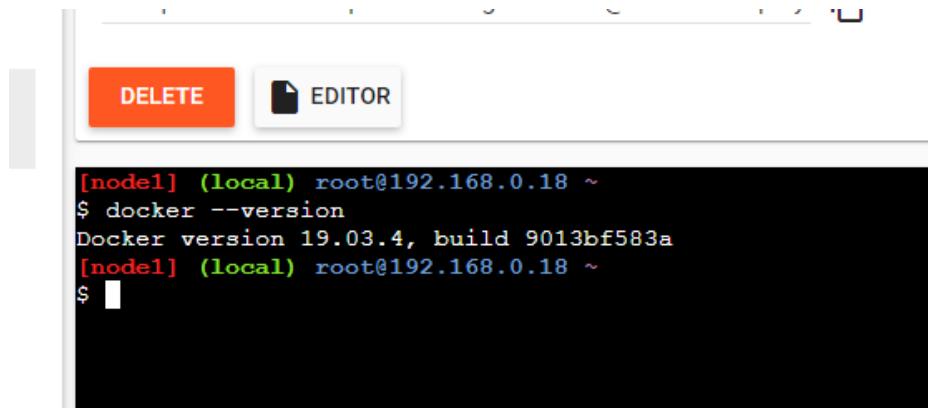
4. perform following:

Method1:**To pull and push images using docker**

Command: to check docker version

```
docker --version
```

output:



The screenshot shows a terminal window with two buttons at the top: "DELETE" and "EDITOR". The terminal output is as follows:

```
[node1] (local) root@192.168.0.18 ~
$ docker --version
Docker version 19.03.4, build 9013bf583a
[node1] (local) root@192.168.0.18 ~
$
```

Command: to pull readymade image

```
docker pull rocker/verse
```

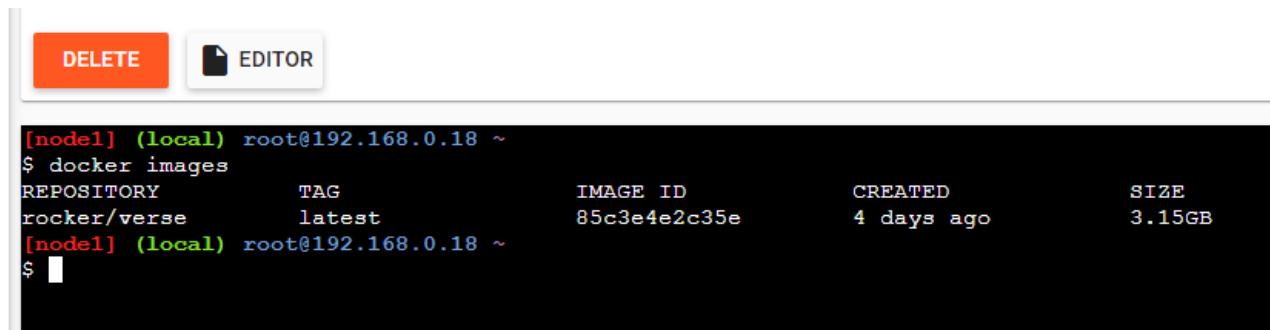
output:



The screenshot shows a terminal window with two buttons at the top: "DELETE" and "EDITOR". The terminal output is as follows:

```
[node1] (local) root@192.168.0.18 ~
$ docker pull rocker/verse
Using default tag: latest
latest: Pulling from rocker/verse
7e2b2a5af8f6: Pull complete
59c89b5f9b0c: Pull complete
4017849f9f85: Pull complete
c8b29d62979a: Pull complete
12004028a6a7: Pull complete
3f09b9a53dfb: Pull complete
03ed58116b0c: Pull complete
7844554d9ef7: Pull complete
6a9d719663d2: Pull complete
Digest: sha256:89b1c8faa7b6b6bb1beb2f2eba41e27a79e6eaeb4d08af28c39b3c3902b04b7d
Status: Downloaded newer image for rocker/verse:latest
docker.io/rocker/verse:latest
[node1] (local) root@192.168.0.18 ~
$ clear
```

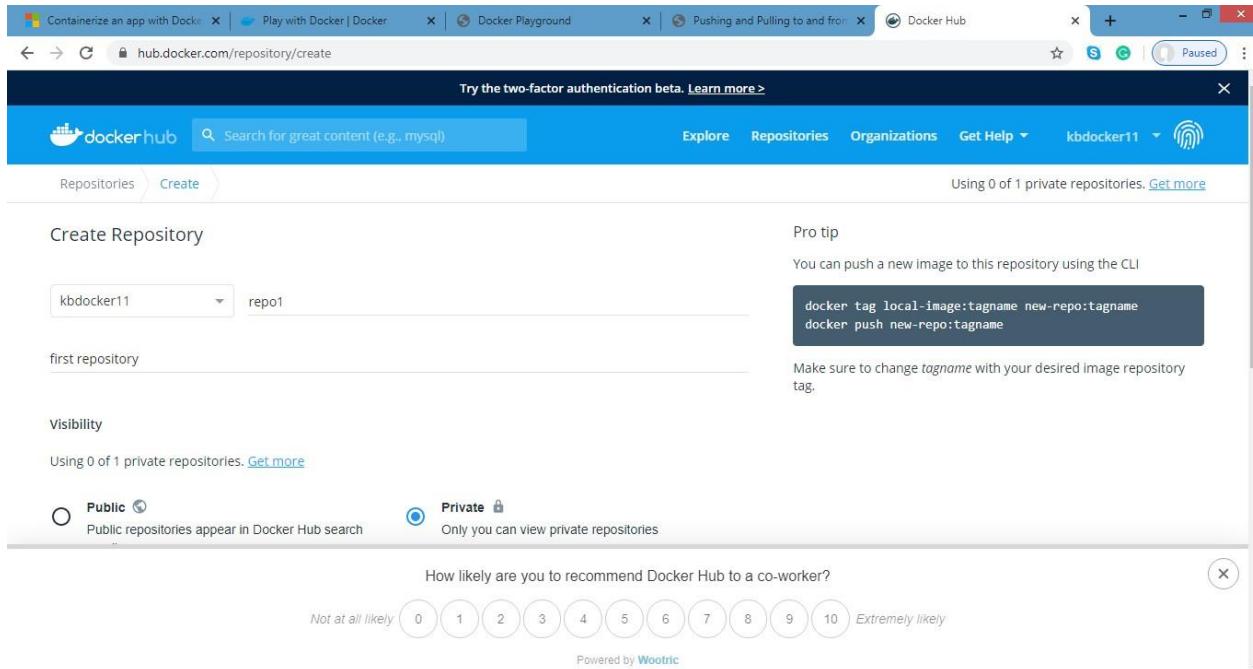
Command: to check images in docker docker images

output:


```
[node1] (local) root@192.168.0.18 ~
$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
rocker/verse        latest   85c3e4e2c35e   4 days ago   3.15GB
[node1] (local) root@192.168.0.18 ~
$
```

Now Login to docker hub and create repository

Output:



The screenshot shows the Docker Hub interface for creating a new repository. The URL is hub.docker.com/repository/create. The page has a blue header with tabs for 'Explore', 'Repositories', 'Organizations', 'Get Help', and a user icon. Below the header, there's a search bar and a 'Create' button. The main form is titled 'Create Repository' and contains fields for 'Repository Name' (set to 'kbdocker11') and 'Repository Tag' (set to 'repo1'). A 'Pro tip' section provides CLI commands: `docker tag local-image:tagname new-repo:tagname` and `docker push new-repo:tagname`. A note says to change 'tagname' to a desired image repository tag. Under 'Visibility', the 'Private' option is selected, with a note that only the user can view private repositories. At the bottom, a satisfaction survey asks 'How likely are you to recommend Docker Hub to a co-worker?' with a scale from 0 (Not at all likely) to 10 (Extremely likely). The survey is powered by Wootric.

Click on Create button

Now check repository created

Command: to login to your docker account docker login –

username=kbdocker11 password:

note: kbdocker11 is my docker ID . You will use your docker ID here. And enter your password .

Output:

The screenshot shows a terminal window with two buttons at the top: 'DELETE' (orange) and 'EDITOR'. The terminal output is as follows:

```
[node1] (local) root@192.168.0.18 ~
$ docker login --username=kbdocker11
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 image

docker tag 8c3e4e2c3e kbdocker11/repo1:firsttry

note: here 8c3e4e2c3e this is image id which you can get from docker images command.

Output:

The screenshot shows a terminal window with two buttons at the top: 'DELETE' (orange) and 'EDITOR'. The terminal output is as follows:

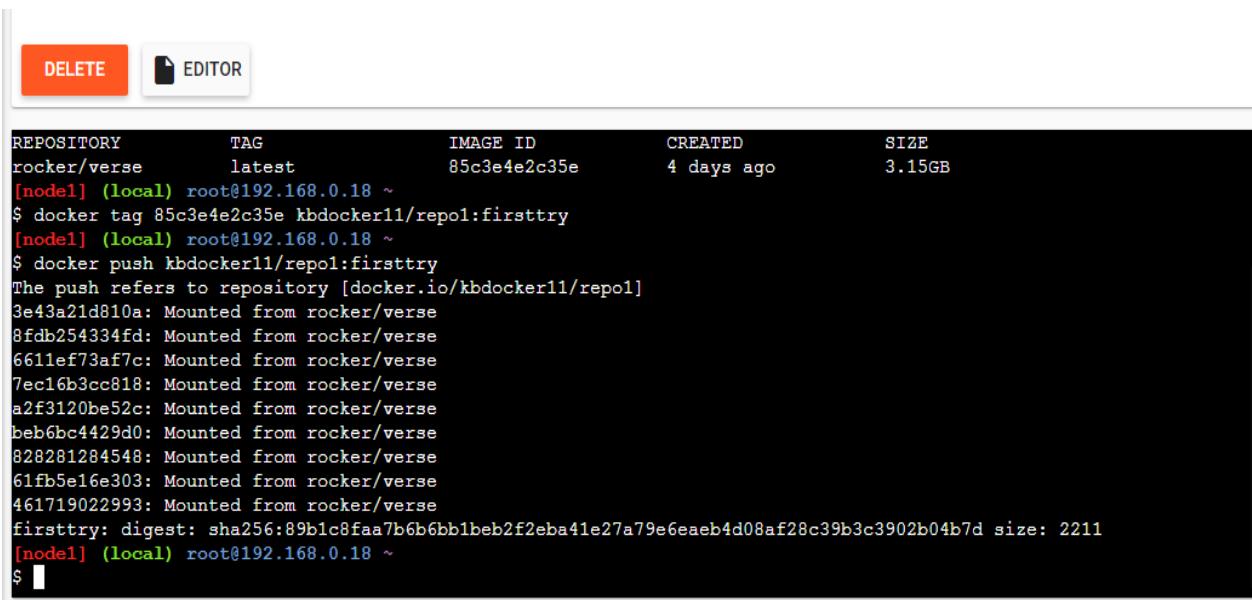
```
[node1] (local) root@192.168.0.18 ~
$ docker images
REPOSITORY          TAG           IMAGE ID      CREATED        SIZE
rocker/verse         latest        85c3e4e2c35e   4 days ago    3.15GB
[node1] (local) root@192.168.0.18 ~
$ docker tag 85c3e4e2c35e kbdocker11/repo1:firsttry
[node1] (local) root@192.168.0.18 ~
$
```

Command: to push image to docker hub account

docker push kbdocker11/repo1:firsttry

note: firsttry is tag name created above.

Output



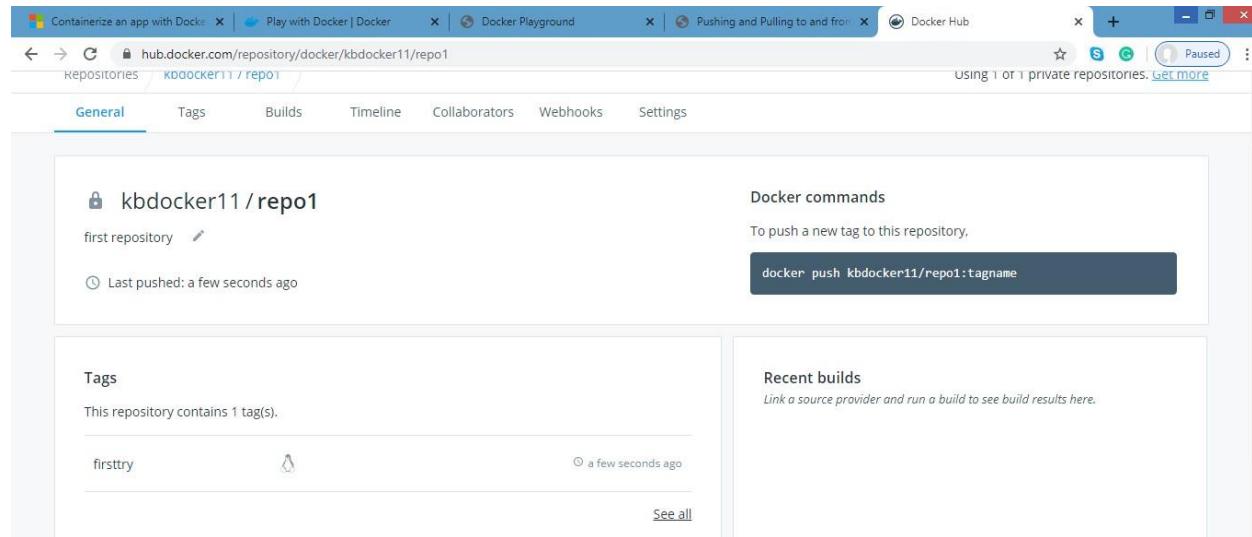
The screenshot shows a terminal window with two tabs: 'DELETE' and 'EDITOR'. The 'EDITOR' tab is active, displaying the following Docker command and its output:

```

REPOSITORY      TAG        IMAGE ID      CREATED       SIZE
rocker/verse    latest     85c3e4e2c35e   4 days ago   3.15GB
[node1] (local) root@192.168.0.18 ~
$ docker tag 85c3e4e2c35e kbdocker11/repo1:firsttry
[node1] (local) root@192.168.0.18 ~
$ docker push kbdocker11/repo1:firsttry
The push refers to repository [docker.io/kbdocker11/repo1]
3e43a21d810a: Mounted from rocker/verse
8fdb254334fd: Mounted from rocker/verse
6611ef73af7c: Mounted from rocker/verse
7ec16b3cc818: Mounted from rocker/verse
a2f3120be52c: Mounted from rocker/verse
beb6bc4429d0: Mounted from rocker/verse
828281284548: Mounted from rocker/verse
61fb5e16e303: Mounted from rocker/verse
461719022993: Mounted from rocker/verse
firsttry: digest: sha256:89b1c8faa7b6b6bb1beb2f2eba41e27a79e6eaeb4d08af28c39b3c3902b04b7d size: 2211
[node1] (local) root@192.168.0.18 ~
$ 

```

Check it in docker hub now



The screenshot shows the Docker Hub repository page for `kbdocker11/repo1`. The page includes sections for General, Tags, Builds, Timeline, Collaborators, Webhooks, and Settings. The General section shows the repository name, a note that it's a first repository, and a message indicating it was last pushed a few seconds ago. The Docker commands section provides a command to push a new tag: `docker push kbdocker11/repo1:tagname`. The Tags section lists one tag: `firsttry`, which was pushed a few seconds ago. The Recent builds section indicates there are no recent builds.

Click on tags and check

Repositories kbdocker11 / repo1 Using 1 of 1 private repositories. [Get more](#)

General Tags Builds Timeline Collaborators Webhooks Settings

Action Filter Tags Sort by Latest

IMAGE
firsttry
Last updated 2 minutes ago by kbdocker11

DIGEST
[89b1c8faa7b6](#)

OS/ARCH
linux/amd64

COMPRESSED SIZE
1.19 GB

docker pull kbdocker11/repo1:firsttry

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

Command : to create docker file

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

Output:

The terminal shows the creation of a Dockerfile and a failed docker build attempt.

```
[node1] (local) root@192.168.0.18 ~
$ cat > Dockerfile <<EOF
> FROM busybox
> CMD echo "Hello world! This is my first Docker image."
> EOF
[node1] (local) root@192.168.0.18 ~
$ docker build -t kbdocker11/repo2
"docker build" requires exactly 1 argument.
See 'docker build --help'.
```

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

Build an image from a Dockerfile

```
[node1] (local) root@192.168.0.18 ~
$
```

Command: to build image from docker file dokcer build -t

kbdocker11/repo2 .

Output:

The terminal shows a successful docker build command.

```
[node1] (local) root@192.168.0.18 ~
$ docker build -t kbdocker11/repo2 .
Sending build context to Docker daemon 38.15MB
Step 1/2 : FROM busybox
latest: Pulling from library/busybox
e2334dd5fe0d: Full complete
Digest: sha256:a8c7ff6367c2fa2a90acd081b484cbded349a7076e7bdf37a05279f276bc12
Status: Downloaded newer image for busybox:latest
--> 6e588be67be6
Step 2/2 : CMD echo "Hello world! This is my first Docker image."
--> fa7c9a33b9bc
Removing intermediate container fa7c9a33b9bc
--> 32be029659d1
Successfully built 32be029659d1
Successfully tagged kbdocker11/repo2:latest
[node1] (local) root@192.168.0.18 ~
$
```

Command: to check docker images docker images

output:

```
$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
kbdocker11/repo2  latest   32be029659d1  About a minute ago  1.22MB
kbdocker11/repo1  firsttry 85c3e4e2c35e  4 days ago    3.15GB
rocker/verse     latest   85c3e4e2c35e  4 days ago    3.15GB
busybox          latest   be5888e67be6  6 days ago    1.22MB
```

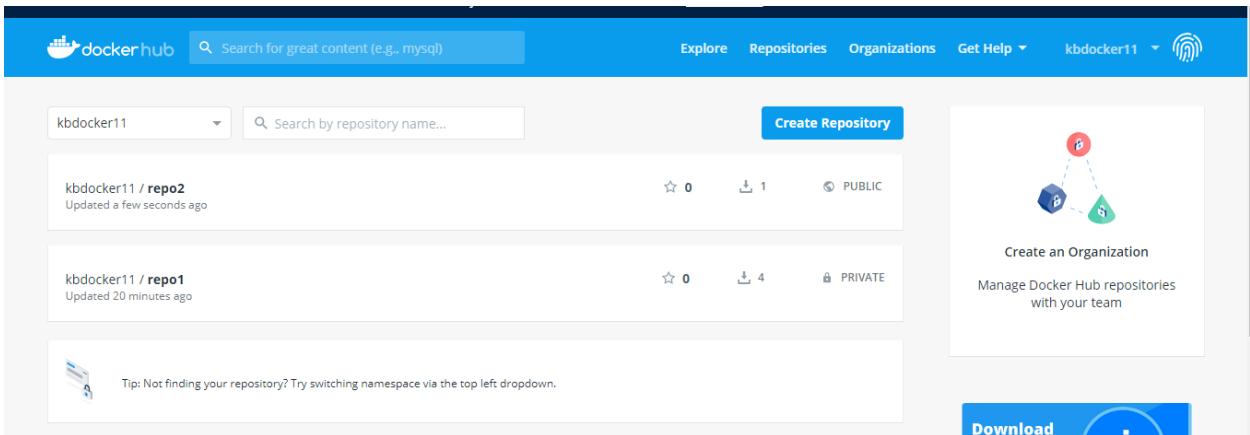
Command: to push image to docker hub

```
docker push kbdocker11/repo2 .
```

Output:

```
[node1] (local) root@192.168.0.18 ~
$ docker push kbdocker11/repo2
The push refers to repository [docker.io/kbdocker11/repo2]
5b0d2d635df8: Mounted from library/busybox
latest: digest: sha256:afa7a4103608d128764a15889501141a10eb9e733f19e4f57645a5ac01c85407 size: 527
[node1] (local) root@192.168.0.18 ~
$
```

Now check it on docker hub



command: to run docker image: docker run

```
kbdocker11/repo2
```

output:

```
[node1] (local) root@192.168.0.18 ~
$ docker run kbdocker11/repo2
Hello world! This is my first Docker image.
[node1] (local) root@192.168.0.18 ~
$
```

Now close session.

Shankar Narayan College

Page:20

PRACTICAL 4

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

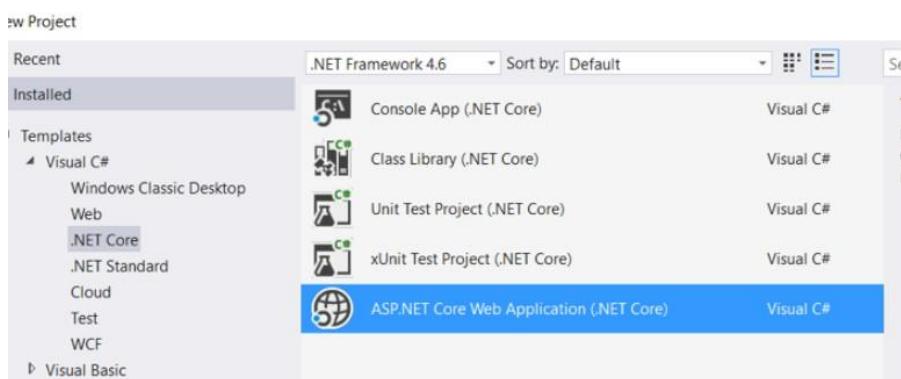
What is the software required for Windows?

- Windows 10 is required for Docker installation.
- Visual Studio 2017 has built-in support for Docker, so this is highly recommended.
- .NET Core SDK
- Docker for Windows
- Docker Tools

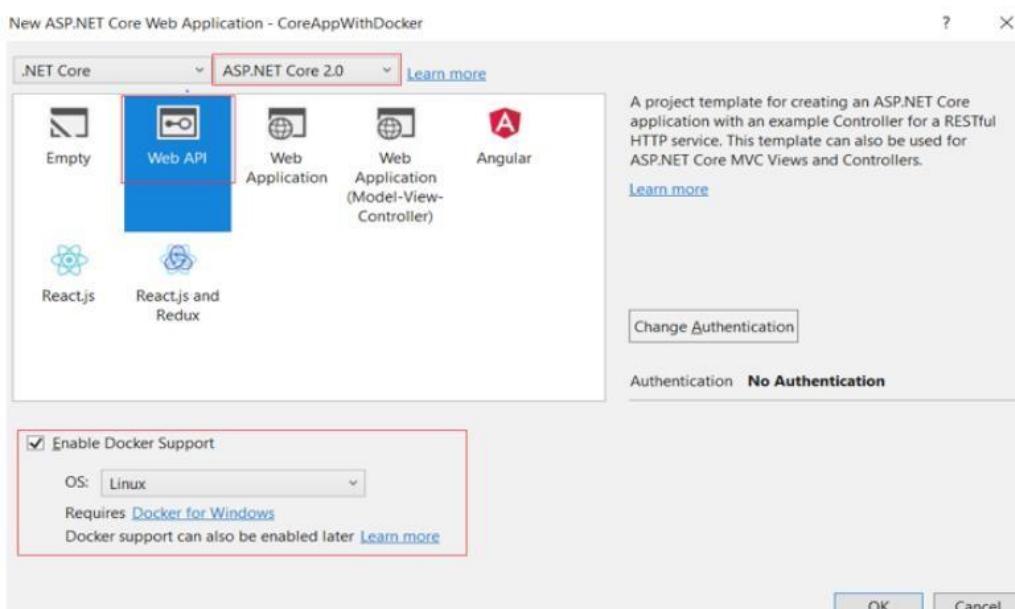
How to create a new microservice using .NET Core and then build and run it using Docker

Step 1: Create a microservice (.NET Core WebAPI) with Docker support as shown below:

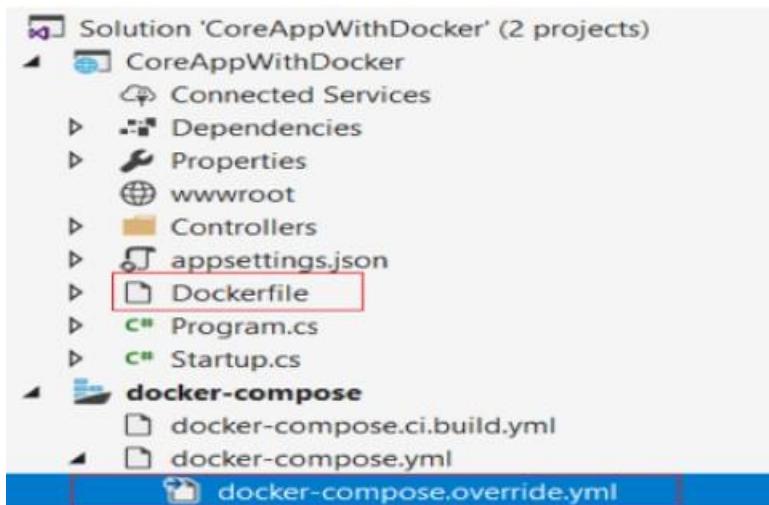
Select “ASP.NET Core Web Application (.NET Core)” from the drop-down menu.



Select the “Enable Docker Support” option.



The following Application Structure will be created along with "Docker File."



Dockerfile

This file is the entry point for running any Docker application.

It is used to build an image of the application's published code in "obj/Docker/publish."

<code>compose.override.yml</code>	<code>Dockerfile</code>	<code>CoreAppWithDocker</code>
- FROM microsoft/aspnetcore:2.0		
ARG source		
WORKDIR /app		
EXPOSE 80		
COPY \${source:-obj/Docker/publish} .		
ENTRYPOINT ["dotnet", "CoreAppWithDocker.dll"]		

<code>docker-compose.override.yml</code>	<code>Dockerfile</code>	<code>CoreAppWithDocker</code>
1 version: '3'		
2		
3 services:		
4 coreappwithdocker:		
5 environment:		
6 - ASPNETCORE_ENVIRONMENT=Development		
7 ports:		
8 - "80"		

Step 2: Update Dockerfile and docker-compose.override.yml as shown below and build the application. 80 is the default Docker container port, so you should update it to a different port number, like 83.

<code>Dockerfile</code>	<code>docker-compose.override.yml</code>
<code>compose.override.yml</code>	<code>docker-compose.override.yml</code>
- FROM microsoft/aspnetcore:2.0	1 version: '3'
ARG source	2
WORKDIR /app	3 services:
ENV ASPNETCORE_URLS http://+:83	4 coreappwithdocker:
EXPOSE 83	5 environment:
COPY \${source:-obj/Docker/publish} .	6 - ASPNETCORE_ENVIRONMENT=Development
ENTRYPOINT ["dotnet", "CoreAppWithDocker.dll"]	7 ports:
	8 - "83"

Note: You can run the application using both Visual Studio and the Docker command line.

First Line FROM Microsoft/aspnetcore:2.0 is the base image for this application in order to run the dot net core application.

ARG source is the argument which helps to pass data to the image.

WORKDIR /app is the working directory of the image; it will store all DLLs inside the app folder.

COPY will copy the DLLs of the application to the root directory (here dot represents root) of the image.

ENTRYPOINT is responsible to run the main application with the help of ASPNETCOREAPP.dll.

In the case of .NET Core 3.1 the Docker file will look like this

```

Dockerfile  DummyApp: Overview
1  #See https://aka.ms/containerfastmode to understand how Visual Studio uses this Dockerfile
2
3  FROM mcr.microsoft.com/dotnet/aspnet:3.1 AS base
4  WORKDIR /app
5  ENV ASPNETCORE_URLS http://+:80
6  EXPOSE 80
7
8
9  FROM mcr.microsoft.com/dotnet/sdk:3.1 AS build
10 WORKDIR /src
11 COPY ["DummyApp.csproj", "."]
12 RUN dotnet restore "./DummyApp.csproj"
13 COPY .
14 WORKDIR "/src/."
15 RUN dotnet build "DummyApp.csproj" -c Release -o /app/build
16
17 FROM build AS publish
18 RUN dotnet publish "DummyApp.csproj" -c Release -o /app/publish
19
20 FROM base AS final
21 WORKDIR /app
22 ENV ASPNETCORE_URLS http://+:80
23 EXPOSE 80
24 COPY --from=publish /app/publish .
25 ENTRYPOINT ["dotnet", "DummyApp.dll"]

```

Step 3: Run the application using Visual Studio.



Step 4: Run the application using Docker Command. Open Application folder from command prompt and check the existing images using Docker images and running containers using Docker PS.

```

D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker images
REPOSITORY          TAG        IMAGE ID      CREATED       SIZE
microsoft/aspnetcore   2.0        f0548d7fbbb7    11 days ago   280MB
microsoft/aspnetcore   1.1.2      4ee30989ab6e    6 weeks ago   305MB
microsoft/aspnetcore   1.1        e7f71ca6bdb2    7 weeks ago   305MB
d4w/nsenter           latest     9e4f13a0901e    13 months ago  83.8kB

D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED          STATUS          PORTS
NAMES

D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>

```

As you can see, there is no running container. So, run the following commands to create build:

To restore packages: dotnet restore

```
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>dotnet restore
Restoring packages for D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker\CoreAppWithDocker.csproj...
Restore completed in 41.91 ms for D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker\CoreAppWithDocker.csproj.
Restore completed in 959.77 ms for D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker\CoreAppWithDocker.csproj.

D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>
```

To publish the application code: dotnet publish -o obj/Docker/publish

```
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>dotnet publish -o obj/Docker/publish
Microsoft (R) Build Engine version 15.4.8.50001 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

CoreAppWithDocker -> D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker\bin\Debug\netcoreapp2.0\CoreAppWithDocker.dll
CoreAppWithDocker -> D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker\obj\Docker\publish\

D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>
```

To build the image: docker build -t imagename

```
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker build -t coreappimage .
Sending build context to Docker daemon 324.6kB
Step 1/6 : FROM microsoft/aspnetcore:2.0
--> f0548d7fbbb7
Step 2/6 : ARG source
--> Running in 592bc21b76ab
--> 89c174bc6537
Removing intermediate container 592bc21b76ab
Step 3/6 : WORKDIR /app
--> 7e8e26f37175
Removing intermediate container 06c848058834
Step 4/6 : EXPOSE 80
--> Running in ed13b8447417
--> a951d3046049
Removing intermediate container ed13b8447417
Step 5/6 : COPY ${source:-obj/Docker/publish} .
--> 615b829e6cbf
Step 6/6 : ENTRYPOINT dotnet CoreAppWithDocker.dll
--> Running in f6540ca62ca4
--> 500f5a37043b
Removing intermediate container f6540ca62ca4
Successfully built 500f5a37043b
Successfully tagged coreappimage:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context will have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions for sensitive files and directories.

D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>
```

Now, check the newly created image “coreappimage” in Docker Images.

D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker images				
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
coreappimage	latest	0db1b2442d44	4 seconds ago	280MB
microsoft/aspnetcore	2.0	f0548d7fbbb7	11 days ago	280MB
microsoft/aspnetcore	1.1.2	4ee30989ab6e	6 weeks ago	305MB
microsoft/aspnetcore	1.1	e7f71ca6bdb2	7 weeks ago	305MB
d4w/nsenter	latest	9e4f13a0901e	13 months ago	83.8kB

Run the image in a container: docker run -d -p 8001:83 –name core1 coreappimage

```
D:\Rakhi\Blogs\CoreAppWithDocker>docker run -d -p 8001:83 --name core1 coreappimage  
2c35a96602889c1f93213a4a0598b7903dddb404fde64c06881d8db3bbef663a  
D:\Rakhi\Blogs\CoreAppWithDocker>
```

Check the running container: Docker PS

```
D:\Rakhi\Blog\Code\CoreAppWithDocker>docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES              coreappimage       "dotnet CoreApp Wit..."   About a minute ago   Up About a minute   0.0.0.0:8001->
2c35a9660288      core1              coreappimage:latest
83/tcp
```

Now the application is running in the Core1 container with the URL <http://localhost:8001>.



Case 1: Run the same image in multiple containers

We can run the same image in multiple containers at the same time by using:

```
docker run -d -p 8002:83 --name core2 coreappimage
```

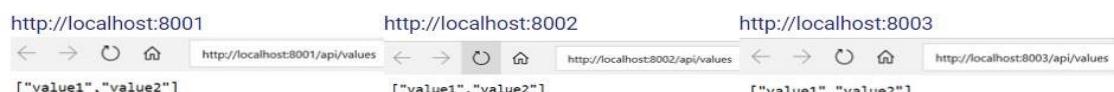
```
docker run -d -p 8003:83 --name core3 coreappimage
```

```
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker run -d -p 8002:83 --name core2 coreappimage  
3f26f83e36f1bda6aa40d67111a2be71ab66c1b775cc080064062cd9b26b7d40  
  
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker run -d -p 8003:83 --name core3 coreappimage  
647d658fa98b989ef58982e7a580ded5278896171dd60bbc88fdf14c486c77b  
  
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>
```

Check the running containers by using Docker PS.

D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker ps						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	
647d658fa98b	coreappimage	"dotnet CoreApp Wit..."	51 seconds ago	Up 51 seconds	0.0.0.0:8003->	
83/tcp	core3	"dotnet CoreApp Wit..."	About a minute ago	Up About a minute	0.0.0.0:8002->	
3f26f83e36f1	coreappimage	"dotnet CoreApp Wit..."				
83/tcp	core2	"dotnet CoreApp Wit..."				
2c35a9660288	coreappimage	"dotnet CoreApp Wit..."	38 minutes ago	Up 38 minutes	0.0.0.0:8001->	
83/tcp	core1	"dotnet CoreApp Wit..."				

We can see that there are 3 containers running for the same image at 8001, 8002, and 8003.



Case 2: Manage Containers: Stop/Start/Remove Containers

Stop container:

We can stop any running containers using "docker stop containerid/containername"
 docker stop core1.

```
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker stop core1
core1
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>
```

Note: Now this container core1 will be listed in the "All Containers" list but not in the running containers list.

Check running containers: docker ps:

```
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
 NAMES
647d658fa98b      coreappimage       "dotnet CoreAppWit..."   23 minutes ago    Up 23 minutes     0.0.0.0:8003->8
8/tcp              core3
3f26f83e36f1      coreappimage       "dotnet CoreAppWit..."   23 minutes ago    Up 23 minutes     0.0.0.0:8002->8
3/tcp              core2
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>
```

Check all containers: docker ps -a

```
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
 NAMES
647d658fa98b      coreappimage       "dotnet CoreAppWit..."   16 minutes ago    Up 16 minutes     0.0.0.0:
8003->83/tcp      core3
3f26f83e36f1      coreappimage       "dotnet CoreAppWit..."   16 minutes ago    Up 16 minutes     0.0.0.0:
8002->83/tcp      core2
2c35a9660288      coreappimage       "dotnet CoreAppWit..."   About an hour ago   Exited (0) 4 minutes ago
core1
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>
```

Note: Now the container running on <http://localhost:8001> will not work.

localhost:8001/api/values



Hmmm...can't reach this page

Try this

- Make sure you've got the right web address:
<http://localhost:8001>
- Search for "<http://localhost:8001>" on Bing

Start Container:

We can start a stopped container using “docker start containerId/containername”

```
=> docker start core1
```

Note: Now it will be listed in the running containers list and <http://localhost:8001> will start working.

**Remove Container:**

We can remove any stopped container, but then we will not be able to start it again.

So first we should stop the container before removing it:

```
=> docker stop core1
```

```
=> docker rm core1
```

```
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker rm core1
core1

D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>
```

Now this container will not be listed on the containers list:

```
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
 NAMES
647d658fa98b      coreappimage       "dotnet CoreAppWit..."   25 minutes ago    Up 25 minutes     0.0.0.0:8003->8
3/tcp   core3
3f26f83e36f1      coreappimage       "dotnet CoreAppWit..."   26 minutes ago    Up 26 minutes     0.0.0.0:8002->8
3/tcp   core2

D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>
```

Case 3 – Share the Application on Docker Hub Repository

Share the application

Estimated reading time: 4 minutes

Now that we've built an image, let's share it! To share Docker images, you have to use a Docker registry. The default registry is Docker Hub and is where all of the images we've used have come from.

[Docker ID](#)

A Docker ID allows you to access Docker Hub which is the world's largest library and community for container images. Create a [Docker ID](#) for free if you don't have one.

Create a repo

To push an image, we first need to create a repository on Docker Hub.

1. Sign up or Sign in to Docker Hub.
2. Click the [Create Repository](#) button.
3. For the repo name, use `getting-started`. Make sure the Visibility is `Public`.

[Private repositories](#)

Did you know that Docker offers private repositories which allows you to restrict content to specific users or teams? Check out the details on the [Docker pricing](#) page.

4. Click the [Create](#) button!

If you look at the image below an example [Docker command](#) can be seen. This command will push to this repo.

Docker commands

[Public View](#)

To push a new tag to this repository,

```
docker push docker/getting-started:tagname
```

In your case use your docker image name instead of getting-started

Push the image

1. In the command line, try running the push command you see on Docker Hub. Note that your command will be using your namespace, not "docker".

```
$ docker push docker/getting-started
The push refers to repository [docker.io/docker/getting-started]
An image does not exist locally with the tag: docker/getting-started
```

Why did it fail? The push command was looking for an image named docker/getting-started, but didn't find one. If you run `docker image ls`, you won't see one either.

To fix this, we need to "tag" our existing image we've built to give it another name.

2. Login to the Docker Hub using the command `docker login -u YOUR-USER-NAME`.
3. Use the `docker tag` command to give the `getting-started` image a new name. Be sure to swap out `YOUR-USER-NAME` with your Docker ID.

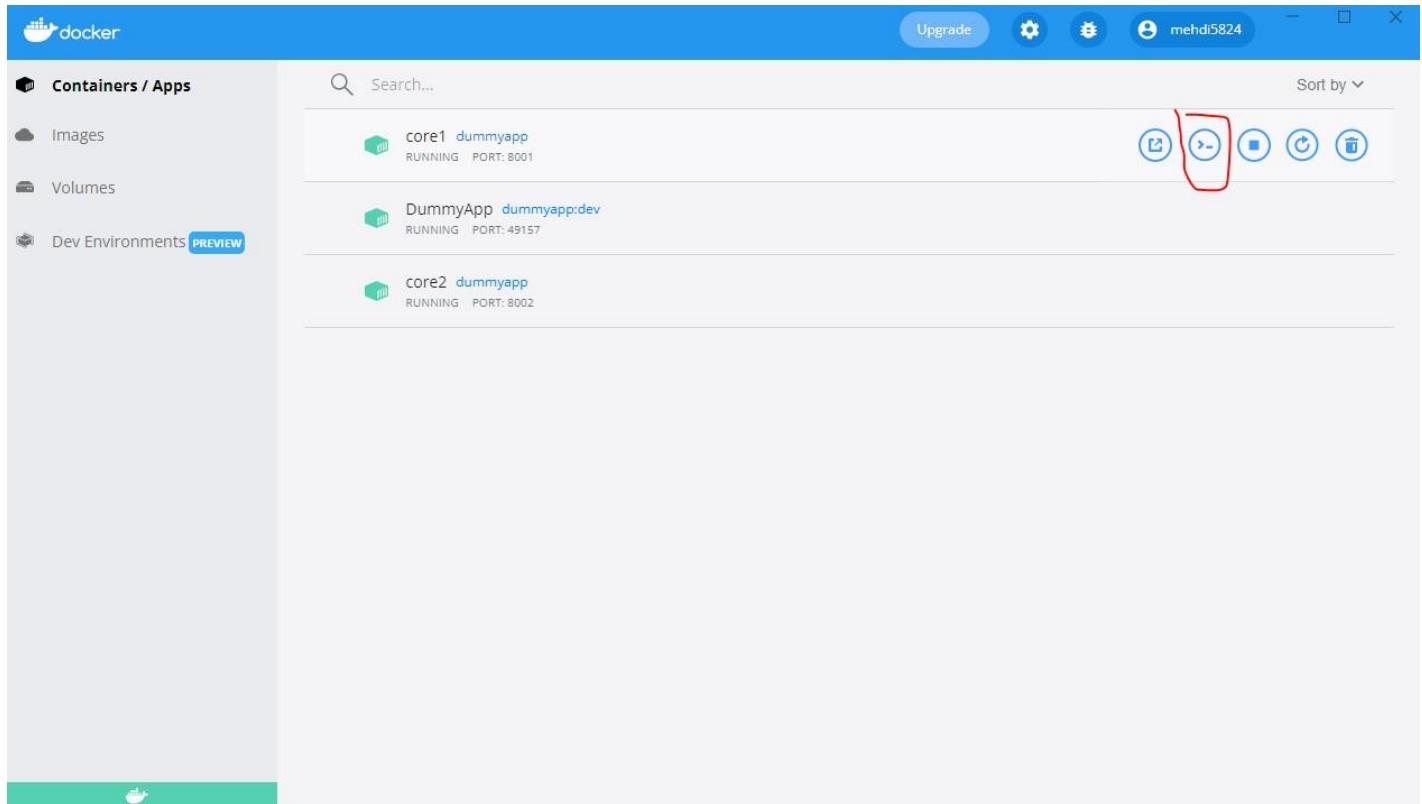
```
$ docker tag getting-started YOUR-USER-NAME/getting-started
```

4. Now try your push command again. If you're copying the value from Docker Hub, you can drop the `tagname` portion, as we didn't add a tag to the image name. If you don't specify a tag, Docker will use a tag called `latest`.

```
$ docker push YOUR-USER-NAME/getting-started
```

Installing Software Packages in Docker

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



Step 2: Now, you have opened the bash of your Ubuntu Docker Container. To install any packages, you first need to update the OS.

```
apt-get -y update
```

```
root@068f710e29a3:/# apt-get -y update
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [107 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [111 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [98.3 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [630 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [428 kB]
Get:9 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [84.4 kB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [1170 kB]
Get:11 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:12 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:13 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [21.6 kB]
Get:14 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [784 kB]
Get:15 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [840 kB]
Get:16 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [103 kB]
Get:17 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages [4277 kB]
Fetched 16.3 MB in 21s (779 kB/s)
Reading package lists... Done
root@068f710e29a3:/#
```

Updating the Container

Step 3: After you have updated the Docker Container, you can now install the Firefox and Vim packages inside it.

```
apt-get -y install firefox  
apt-get -y install vim
```

```
root@068f710e29a3:/# apt-get -y install firefox  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
adwaita-icon-theme alsa-ucm-conf at-spi2-core dbus dbus-user-session dconf-gsettings-backend dconf-service  
distro-info-data dnsetup file fontconfig fontconfig-config fonts-dejavu-core gir1.2-glib-2.0 glib-networking glib-networking-common  
glib-networking-services gsettings-desktop-schemas gtk-update-icon-cache hicolor-icon-theme humanity-icon-theme krb5-locales  
libapparmor1 libargon2-1 libasound2 libasound2-data libatk-bridge2.0-0 libatk1.0-0 libatk1.0-data libatspi2.0-0 libavahi-client3  
libavahi-common-data libavahi-common3 libbrotli libbsdsd libcroffice libcro2 libcanberra0 libcap2 libcolord2 libcryptsetup12
```

Installing Firefox

```
root@068f710e29a3:/# apt-get -y install vim  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
alsa topology conf alsa-ucm-conf file libasound2 libasound2-data libcanberra0 libexpat1 libgpm2 libltdl7 libmagic-noc libmagic1  
libmpdec2 libogg0 libpython3.8 libpython3.8-minimal libpython3.8-stdlib libreadline8 libsqlite3-0 libssl1.1 libtdb1 libvorbis0a  
libvorbisfiles mine-support readline-common sound-theme-freedesktop vim-common vim-runtime xxd xz-utils  
Suggested packages:  
libasound2-plugins alsa-utils libcanberra-gtk0 libcanberra-pulse gpm readline-doc ctags vim-doc vim-scripts  
The following NEW packages will be installed:
```

Installing Vim

You can now easily use these packages through the bash itself.

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

Container volumes

With the previous experiment, we saw that each container starts from the image definition each time it starts. While containers can create, update, and delete files, those changes are lost when the container is removed and all changes are isolated to that container. With volumes, we can change all of this.

Volumes provide the ability to connect specific filesystem paths of the container back to the host machine. If a directory in the container is mounted, changes in that directory are also seen on the host machine. If we mount that same directory across container restarts, we'd see the same files.

Step 1

Working with Docker Volumes explained below:-

- a) Let us create the volume first. For the reference we will type below command:-
→ docker volume

```
F:\Microservices\getting-started-master\app>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.

F:\Microservices\getting-started-master\app>
```

- b) Now lets create the actual volume:-

→ docker volume create myvol1

```
F:\Microservices\getting-started-master\app>docker volume create myvol1

F:\Microservices\getting-started-master\app>
```

As you can see here our volume is created.

- c) To list the volume we will write below command:-

```
F:\Microservices\getting-started-master\app>docker volume ls
DRIVER      VOLUME NAME
local      ab83257ee43df3f86bfea2b09c1d1ffe5a59b9ced82c6b7ea5f458e9e298e72
local      d247fdb49990ed914b54ffffe365671c1f3b773d5871038817e18d36cf6e288bf
local      myvol1
.
.
.
F:\Microservices\getting-started-master\app>
```

- d) To get the details of our volume we have to write below command:-

→ docker volume inspect myvol1

```
F:\Microservices\getting-started-master\app>docker volume inspect myvol1
[
  {
    "CreatedAt": "2021-05-10T06:36:15Z",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/myvol1/_data",
    "Name": "myvol1",
    "Options": {},
    "Scope": "local"
  }
]
F:\Microservices\getting-started-master\app>
```

Here you can see all the details of our myvol1 i.e. name, created time, driver, mountpoint.

Our volume is located at the path mentioned in Mountpoint section.

- e) To remove your volume you can write below command:-

→ docker volume rm myvol1

To remove all unused volumes we can write below command

→ docker volume prune

These are the basic functionalities of docker volume. You can explore more functionalities as well.

Working with docker network explained below:-

To write this command below is the syntax:-

→ docker network COMMAND

- a) To Connect a container to a network

→ docker network connect

- b) To create a network we have to write below command:-
→ docker network create
- c) To disconnect a container from a network
→ docker network disconnect
- d) To display detailed information on one or more networks
→ docker network inspect
- e) To list the network:-
→ docker network ls
- f) To remove all unused networks
→ docker network prune
- g) To remove one or more networks
→ docker network rm

Practical 5

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.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * Repository name *

Mehdi5824 / hello-world-p ✓

Great repository names are short and memorable. Need inspiration? How about [redesigned-goggles](#)?

Description (optional)

Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file
This is where you can write a long description for your project. [Learn more](#).

Add .gitignore
Choose which files not to track from a list of templates. [Learn more](#).

gitignore template: None

Choose a license
A license tells others what they can and can't do with your code. [Learn more](#).

License: None

This will set main as the default branch. Change the default name in your settings.

You are creating a public repository in your personal account.

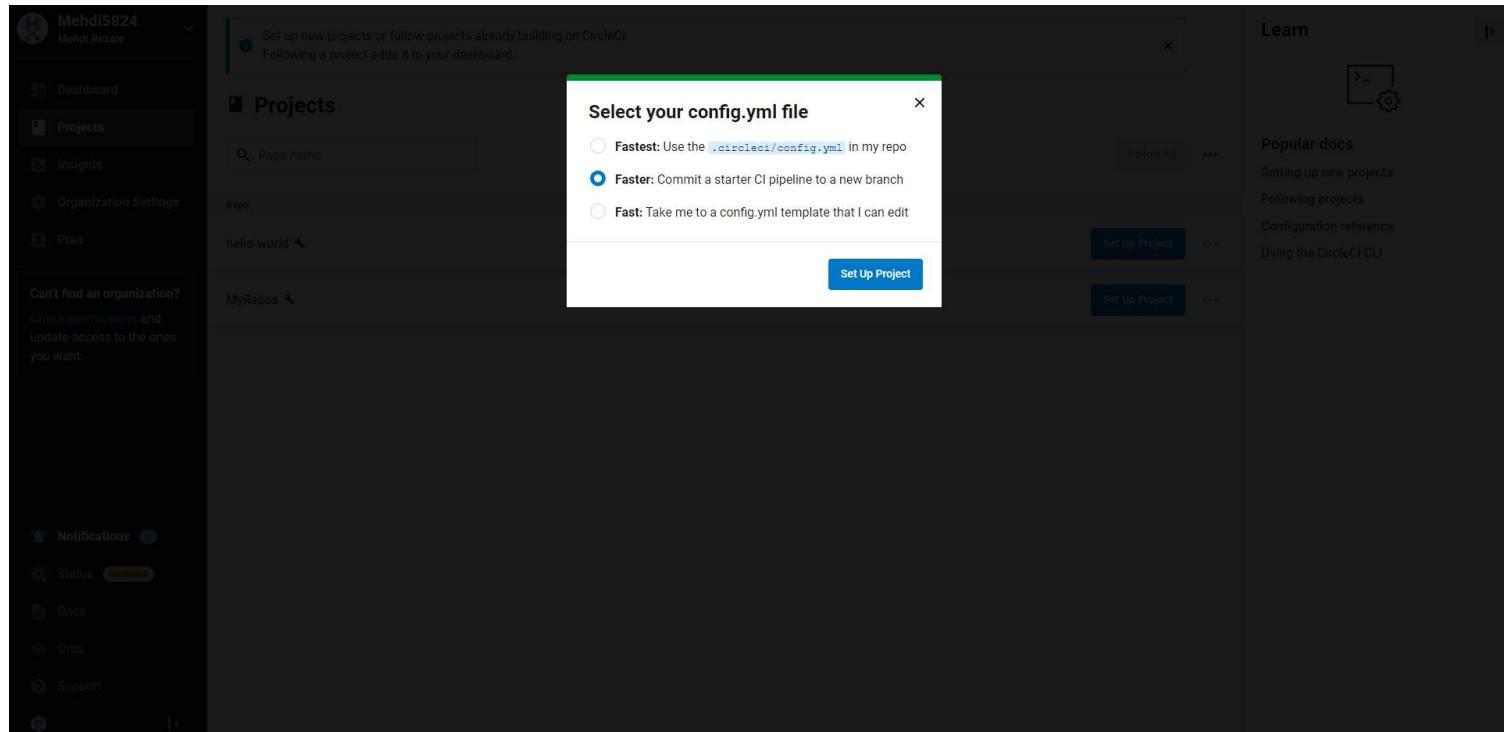
Create repository

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

The screenshot shows the CircleCI project dashboard for the user 'MehdiRezaie'. The left sidebar includes links for Dashboard, Projects (which is selected), Insights, Organization Settings, and Plan. A message box at the top right says 'Set up new projects or follow projects already building on CircleCI. Following a project adds it to your dashboard.' Below this, the 'Projects' section lists two repositories: 'Repo' containing 'hello-world' and 'MyRepos' containing 'MyRepos'. Each repository has a 'Set Up Project' button and a three-dot menu icon. The right sidebar features a 'Learn' section with a gear icon and links to 'Popular docs' such as 'Setting up new projects', 'Following projects', 'Configuration reference', and 'Using the CircleCI CLI'.

Step 2 - Set up CircleCI

1. Navigate to the CircleCI Projects page. If you created your new repository under an organization, you will need to select the organization name.
2. You will be taken to the Projects dashboard. On the dashboard, select the project you want to set up (hello-world).
3. Select the option to commit a starter CI pipeline to a new branch, and click Set Up Project. This will create a file .circleci/config.yml at the root of your repository on a new branch called circleci-project-setup.



Step 3 - Your first pipeline

On your project's pipeline page, click the green Success button, which brings you to the workflow that ran (say-hello-workflow).

Within this workflow, the pipeline ran one job, called say-hello. Click say-hello to see the steps in this job:

- a. Spin up environment
- b. Preparing environment variables
- c. Checkout code
- d. Say hello

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

The screenshot shows the CircleCI web interface for a project named 'hello-world'. The pipeline 'say-hello-workflow' has run successfully, indicated by a green success icon. The pipeline details are as follows:

Pipeline	Status	Workflow	Branch / Commit	Start	Duration	Actions
hello-world 1	Success	say-hello-workflow	circleci-project-setup eb308f0	3m ago	5s	↻ ⟳ X ...

The sidebar on the left includes sections for Dashboard, Projects, Insights, Organization Settings, Plan (with an Upgrade button), Notifications (1 notification), Status (DEGRADED), Docs, Orbs, and Support.

Select “say-hello” Job with a green tick

The screenshot shows the CircleCI web interface for the 'say-hello-workflow' job within the 'hello-world' pipeline. The job status is 'Success'. The job details are as follows:

Duration / Finished	Branch	Commit	Author
5s / 8m ago	circleci-project-setup	6b308f0	Mehdi5824

The job itself is labeled 'say-hello' and has a duration of 3s. A green checkmark icon indicates it was successful.

The sidebar on the left is identical to the first screenshot, showing the same navigation and status indicators.

The screenshot shows the CircleCI web interface for a project named 'say-hello'. The build status is 'Success'. Key details include:

- Duration:** 2s / 9m ago
- Executor:** Docker / Large
- Branch:** circleci-project-setup
- Job:** say-hello (1)
- Author:** Mehdi Rezaie
- Steps:** Spin up environment, Preparing environment variables, Checkout code, Say hello.

Select Branch and option circleci-project-setup

The screenshot shows the GitHub repository page for 'Mehdi5824/hello-world'. The repository has 2 branches and 0 tags. The 'Code' tab is selected. A dropdown menu for 'Switch branches/tags' is open, showing 'main' (selected), '2 branches', and '0 tags'. The repository has 2 branches and 0 tags.

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.

This screenshot shows the GitHub repository page for `Mehdi5824/hello-world`. The repository is public and contains two branches: `circleci-project-setup` (the active branch) and `main`. The `circleci-project-setup` branch is 1 commit ahead of `main`. The repository has 2 commits, both made by `Mehdi5824` on `14 minutes ago`. The first commit is `Add .circleci/config.yml`, and the second is `Initial commit`. The `README.md` file contains the text `hello-world`.

This screenshot shows the same GitHub repository page after a configuration file has been added. The `circleci-project-setup` branch now has one additional commit: `Add .circleci/config.yml` by `Mehdi5824` on `15 minutes ago`. The commit message is `Add .circleci/config.yml`. The repository structure shows a new folder named `.circleci` containing the `config.yml` file.

The screenshot shows a GitHub repository page for `Mehdi5824/hello-world`. The file `.circleci/config.yml` is displayed. The code defines a CircleCI pipeline with one job named `say-hello` that runs a Docker container with the image `circleci/circleci:stable`, checks out the repository, and runs a command to echo "Hello, World!". There is one commit by `Mehdi5824` with the message "Add .circleci/config.yml".

```
1 # Use the latest 2.1 version of CircleCI pipeline process engine.
2 # See: https://circleci.com/docs/2.0/configuration-reference
3 version: 2.1
4
5 # Define a job to be invoked later in a workflow.
6 # See: https://circleci.com/docs/2.0/configuration-reference/#jobs
7 jobs:
8   say-hello:
9     # Specify the execution environment. You can specify an image from Dockerhub or use one of our Convenience Images from CircleCI's Developer Hub.
10    # See: https://circleci.com/docs/2.0/configuration-reference/#docker-machine-macos-windows-executor
11    docker:
12      - image: circleci/circleci:stable
13      # Add steps to the job
14      # See: https://circleci.com/docs/2.0/configuration-reference/#steps
15      steps:
16        - checkout
17        - run:
18          name: "Say hello"
19          command: "echo Hello, World!"
20
21 # Invoke jobs via workflows
22 # See: https://circleci.com/docs/2.0/configuration-reference/#workflows
23 workflows:
24   say-hello-workflow:
25     jobs:
26       - say-hello
```

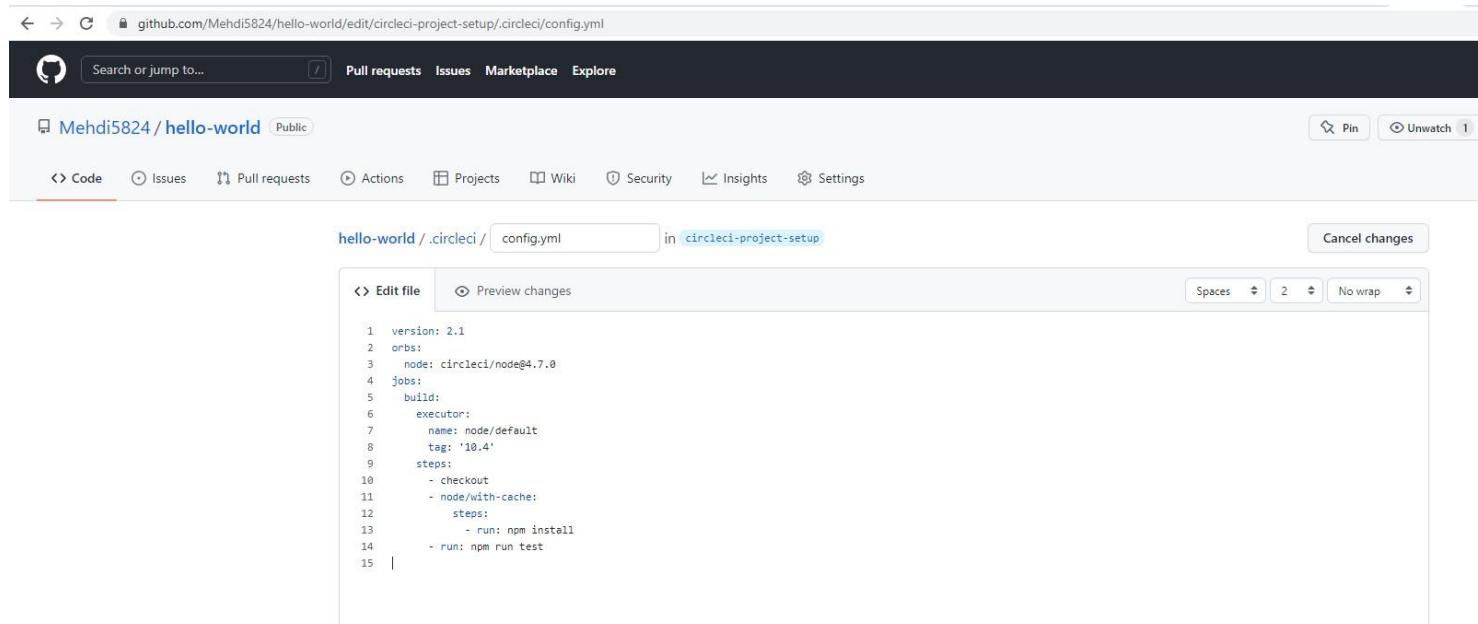
Let's use the **Node orb**. Replace the existing config by pasting the following code:

```

1  version: 2.1
2  orbs:
3    node: circleci/node@4.7.0
4  jobs:
5    build:
6      executor:
7        name: node/default
8        tag: '10.4'
9      steps:
10     - checkout
11     - node/with-cache:
12       steps:
13         - run: npm install
14         - run: npm run test

```

The GitHub file editor should look like this



The screenshot shows the GitHub interface for editing a file named `config.yml` located in the `.circleci` directory of a repository named `hello-world`. The file contains the YAML configuration for a CircleCI build job named `build` using the `node` orb. The configuration includes steps for checkout, running `npm install`, and running `npm run test`.

```

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

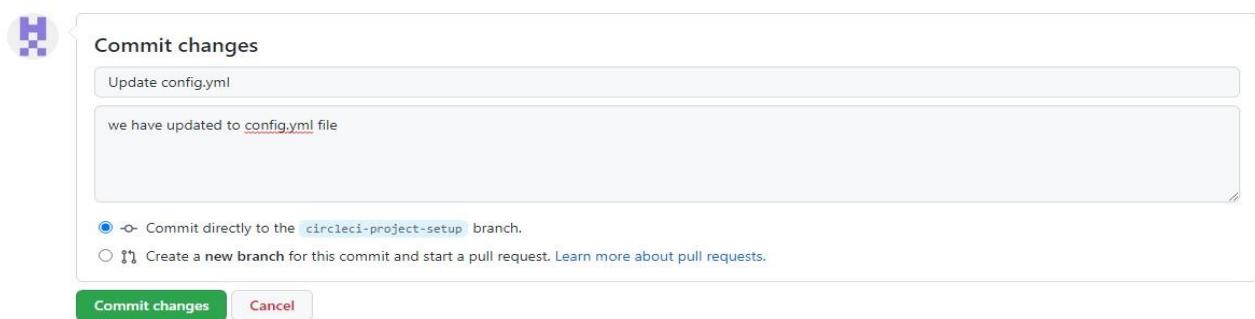
```

Scroll down and Commit your changes on GitHub

```

10      - checkout
11      - node/with-cache:
12          steps:
13              - run: npm install
14      - run: npm run test
15

```



© 2022 GitHub, Inc.

[Terms](#)

[Privacy](#)

[Security](#)

[Status](#)

[Docs](#)

[Contact GitHub](#)

[Pricing](#)

[API](#)

[Training](#)

[Blog](#)

[About](#)

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.

Pipeline	Status	Workflow	Branch / Commit	Start	Duration	Actions
hello-world 2	Failed	Error calling workflow: 'Workflow'Error calling job: 'build'Cannot find a definition for command named node/with-cache	circleci-project-setup ca5ab74 Update config.yml	3m ago	6s	
hello-world 1	Success	say-hello-workflow	circleci-project-setup 6b308f0	22m ago	5s	

Step 5 – Use Workflows

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

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

```
1  version: 2
2  jobs: # we now have TWO jobs, so that a workflow can coordinate them!
3    one: # This is our first job.
4      docker: # it uses the docker executor
5        - image: cimg/ruby:2.6.8 # specifically, a docker image with ruby 2.6.8
6        auth:
7          username: mydockerhub-user
8          password: $DOCKERHUB_PASSWORD # context / project UI env-var reference
9      # Steps are a list of commands to run inside the docker container above.
10     steps:
11       - checkout # this pulls code down from GitHub
12       - run: echo "A first hello" # This prints "A first hello" to stdout.
13       - run: sleep 25 # a command telling the job to "sleep" for 25 seconds.
14   two: # This is our second job.
15     docker: # it runs inside a docker image, the same as above.
16       - image: cimg/ruby:3.0.2
17       auth:
18         username: mydockerhub-user
19         password: $DOCKERHUB_PASSWORD # context / project UI env-var reference
20     steps:
21       - checkout
22       - run: echo "A more familiar hi" # We run a similar echo command to above.
23       - run: sleep 15 # and then sleep for 15 seconds.
24   # Under the workflows: map, we can coordinate our two jobs, defined above.
25 workflows:
26   version: 2
27   one_and_two: # this is the name of our workflow
28     jobs: # and here we list the jobs we are going to run.
29       - one
30       - two
```

You don't need to write the comments which are the text after #

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

```

1 version: 2
2 jobs: # we now have TWO jobs, so that a workflow can coordinate them!
3 one: # This is our first job.
4   docker: # it uses the docker executor
5     - image: cimg/ruby:2.6.8 # specifically, a docker image with ruby 2.6.8
6     auth:
7       username: mydockerhub-user
8       password: $DOCKERHUB_PASSWORD # context / project UI env-var reference
9   # Steps are a list of commands to run inside the docker container above.
10  steps:
11    - checkout # this pulls code down from GitHub
12    - run: echo "A first hello" # This prints "A first hello" to stdout.
13    - run: sleep 25 # a command telling the job to "sleep" for 25 seconds.
14 two: # This is our second job.
15   docker: # it runs inside a docker image, the same as above.
16     - image: cimg/ruby:3.0.2
17     auth:
18       username: mydockerhub-user
19       password: $DOCKERHUB_PASSWORD # context / project UI env-var reference
20   steps:
21    - checkout
22    - run: echo "A more familiar hi" # We run a similar echo command to above.
23    - run: sleep 15 # and then sleep for 15 seconds.
24 # Under the workflows: map, we can coordinate our two jobs, defined above.
25 workflows:
26   version: 2
27   one_and_two: # this is the name of our workflow
28     jobs: # and here we list the jobs we are going to run.
29       - one
30       - two
31

```

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

Pipeline	Status	Workflow	Branch / Commit	Start	Duration	Actions
hello-world 3	Running	one_and_two	circleci-project-setup e11ad95 Update config.yml	14s ago	13s	...
hello-world 2	Failed	Build Error Workflow error: Cannot find a definition for command named node/with-cache	circleci-project-setup ca5ab74 Update config.yml	9m ago	6s	...
hello-world 1	Success	say-hello-workflow	circleci-project-setup 6b308f0	29m ago	5s	...

The screenshot shows the CircleCI dashboard for a user named 'Mehdi5824'. The main navigation bar includes 'Dashboard', 'Project', 'Branch', and 'Workflow'. Under 'Project', the path 'All Pipelines > hello-world > circleci-project-setup > one_and_two' is selected. A success message 'one_and_two Success' is displayed. Below this, detailed information is shown: Duration / Finished (34s / 3s ago), Branch (circleci-project-setup), Commit (e11ad95), and Author & Message (Update config.yml). Two build steps are listed: 'two' (19s) and 'one' (31s), both marked as successful. A sidebar on the left provides links for 'Dashboard', 'Projects', 'Insights', 'Organization Settings', 'Plan', and 'Notifications' (with 1 notification). A message box suggests checking permissions if no organizations are found.

Step 5 – 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:

```
1  version: 2
2  jobs:
3    one:
4      docker:
5        - image: cimg/ruby:3.0.2
6        auth:
7          username: mydockerhub-user
8          password: $DOCKERHUB_PASSWORD # context / project UI env-var reference
9      steps:
10     - checkout
11     - run: echo "A first hello"
12     - run: mkdir -p my_workspace
13     - run: echo "Trying out workspaces" > my_workspace/echo-output
14     - persist_to_workspace:
15       # Must be an absolute path, or relative path from working_directory
16       root: my_workspace
17       # Must be relative path from root
18     paths:
19       - echo-output
20 two:
21   docker:
22     - image: cimg/ruby:3.0.2
23     auth:
24       username: mydockerhub-user
25       password: $DOCKERHUB_PASSWORD # context / project UI env-var reference
26   steps:
27     - checkout
28     - run: echo "A more familiar hi"
29     - attach_workspace:
30       # Must be absolute path or relative path from working_directory
31       at: my_workspace
```

```

32      - run: |
33          if [[ $(cat my_workspace/echo-output) == "Trying out workspaces" ]]; then
34              echo "It worked!";
35          else
36              echo "Nope!"; exit 1
37          fi
38
39 workflows:
40     version: 2
41     one_and_two:
42         jobs:
43             - one
44             - two:
45                 requires:
46                     - one

```

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

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

[Cancel changes](#)

<> Edit file [Preview changes](#)

Spaces 2 No wrap

```

1  version: 2
2  jobs:
3    one:
4      docker:
5        - image: cimg/ruby:3.0.2
6        auth:
7          username: mydockerhub-user
8          password: $DOCKERHUB_PASSWORD # context / project UI env-var reference
9      steps:
10        - checkout
11        - run: echo "A first hello"
12        - run: mkdir -p my_workspace
13        - run: echo "Trying out workspaces" > my_workspace/echo-output
14        - persist_to_workspace:
15            # Must be an absolute path, or relative path from working_directory
16            root: my_workspace
17            # Must be relative path from root
18            paths:
19                - echo-output
20    two:
21      docker:
22        - image: cimg/ruby:3.0.2
23        auth:
24          username: mydockerhub-user
25          password: $DOCKERHUB_PASSWORD # context / project UI env-var reference
26      steps:
27        - checkout
28        - run: echo "A more familiar hi"
29        - attach_workspace:
30            # Must be absolute path or relative path from working_directory
31            at: my_workspace
32
33      - run: |
34          if [[ $(cat my_workspace/echo-output) == "Trying out workspaces" ]]; then
35              echo "It worked!";

```

 Commit changes

 Commit changes

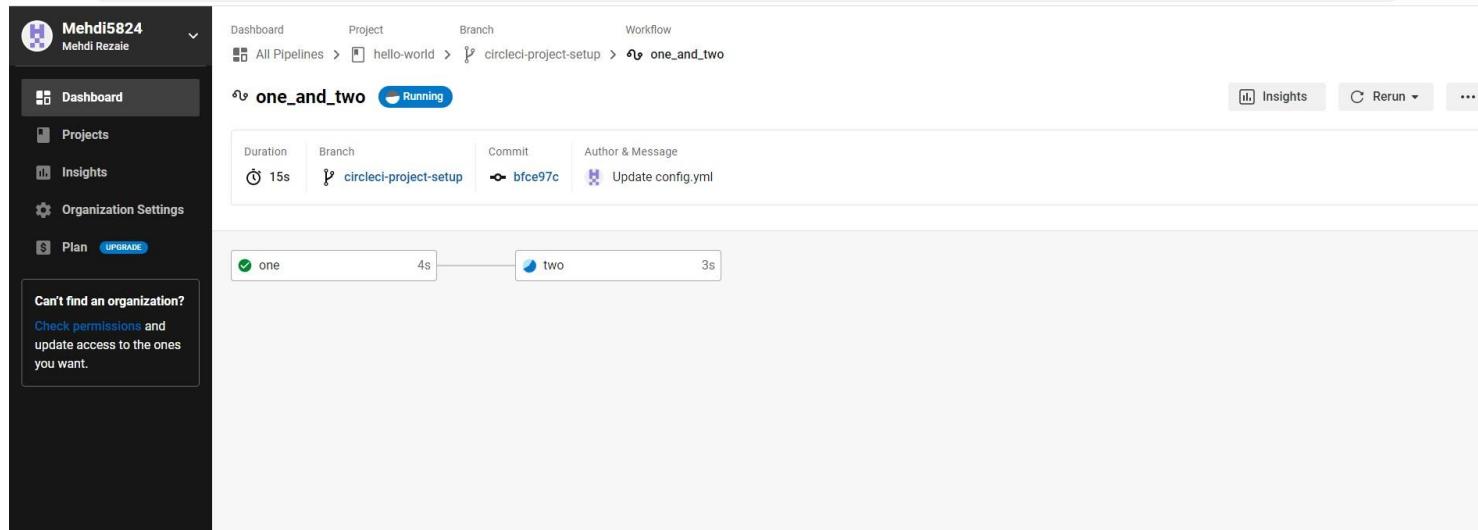
Update config.yml

3rd Update

- o Commit directly to the `circleci-project-setup` branch.
- ↗ Create a new branch for this commit and start a pull request. Learn more about pull requests.

[Commit changes](#) [Cancel](#)

Finally your workflow with the jobs running should look like this



Practical No 6

Aim: Creating Microservice with ASP.NET Core.

(Install .Net core sdk first)

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

1) Create new project:

Command :

```
dotnet new webapi -o TeamService
```

output:

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

D:\>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...
  Restore completed in 5.9 sec for D:\TeamService\TeamService.csproj.

Restore succeeded.
```

2) Remove existing weatherforecast files both model and controller files.

3) Add new files as follows:

4) 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;
    }
}
```

5) Add Team.cs to “D:\TeamService\Models” folder

using System;
using System.Collections.Generic;

```
{  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;
    }
}
```

3) 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) // I HATE NULLS, MUST FIXERATE THIS.
            {
                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);
        }
    }
}

```

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;
    }
}

```

4) create folder “D:\TeamService\Persistence”:**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);
    }
}

```

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;
}
}
}

```

- 7) add following line to Startup.cs in public void ConfigureServices(IServiceCollection services) method**

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

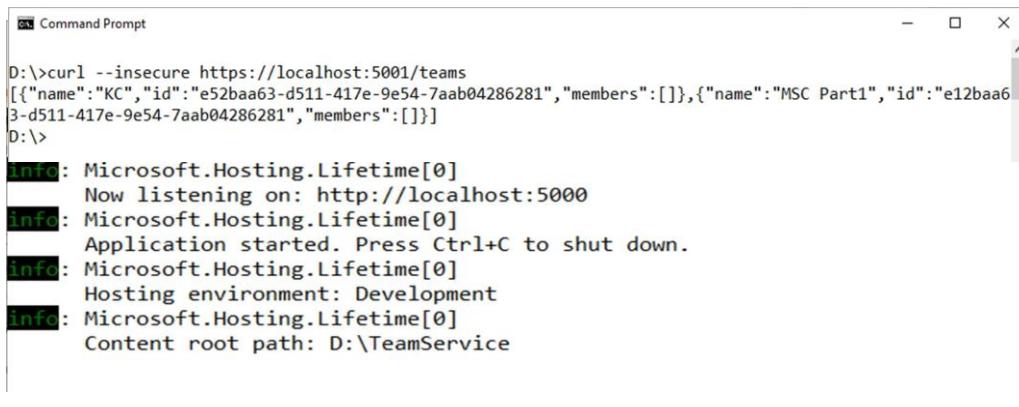
- 8) Now open two command prompts to run this project**

- 9) On Command prompt 1: (go inside folder teamservice first)**
Commands:

```
dotnet run
```

Shankar Narayan College

Page:58

Output:


```
D:\>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": []}]
D:\>
[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: D:\TeamService
```

10) On command prompt 2**Command: To get all teams**

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

output:


```
D:\>curl --insecure https://localhost:5001/teams
[]
D:\>
```

Command : To create new teamcurl --insecure -H "Content-Type:application/json" -X POST -d "{\"id\":\"e52baa63-d511-417e-9e54-7aab04286281\", \"name\":\"KC\"}" <https://localhost:5001/teams>**output:**


```
D:\>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": []}
D:\>
```

Command : To create one more new teamcurl --insecure -H "Content-Type:application/json" -X POST -d "{\"id\":\"e12baa63-d511-417e-9e54-7aab04286281\", \"name\":\"MSC Part1\"}" <https://localhost:5001/teams>**output:**


```
D:\>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": []}
D:\>
```

Command : To get all teamscurl --insecure <https://localhost:5001/teams>**Output:**

Command : to get single team with team-id as parameter

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

output:

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

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
```

output:

```
D:\>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":[]}
D:\>
```

Command: to delete team

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

Output:

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

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

Command:

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

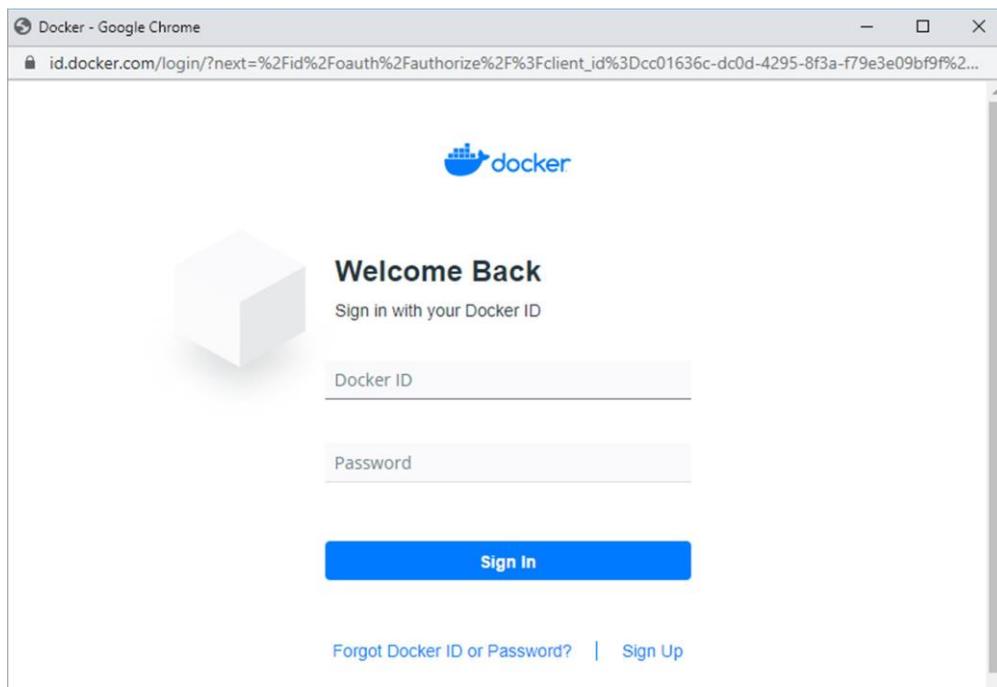
Output:

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

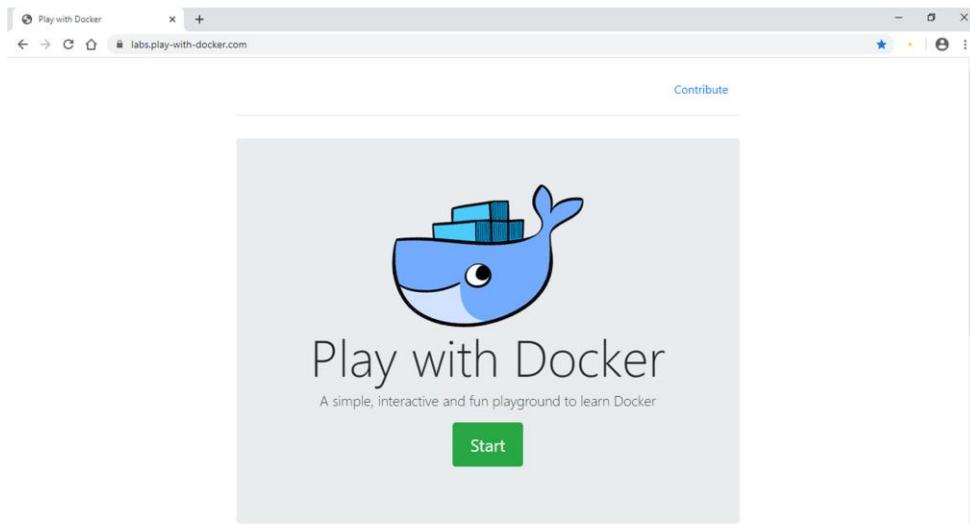
Practical 7

Aim: Creating Backing Service with ASP.NET Core.

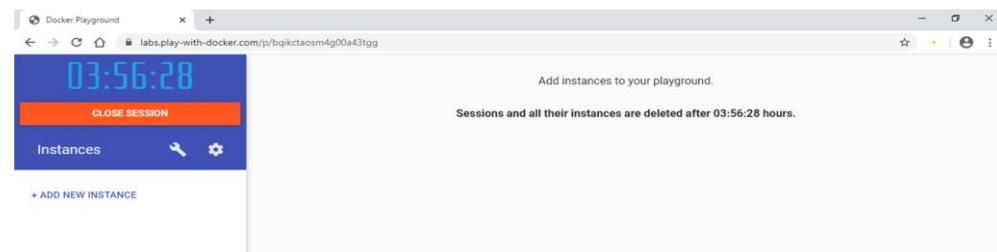
Running Location Service in Docker (create docker hub login first to use it in play with docker)
Now login in to Play-With-Docker

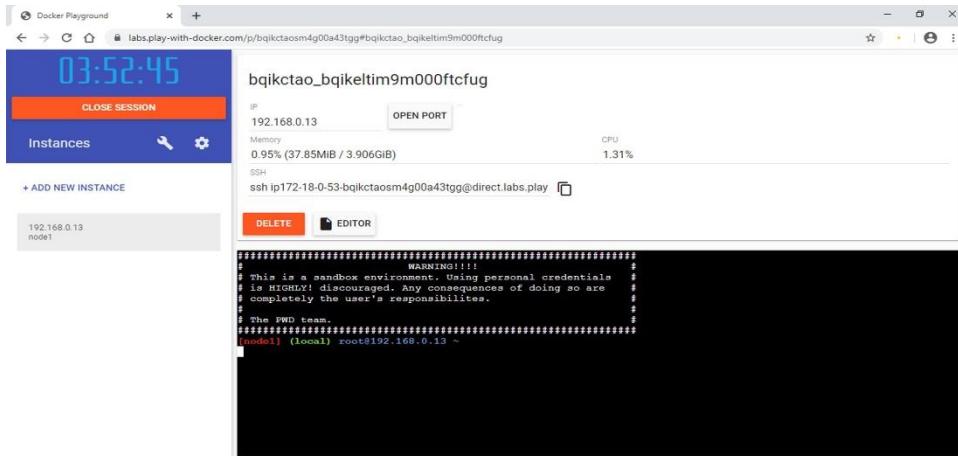


Click on Start



Click on Add New Instance



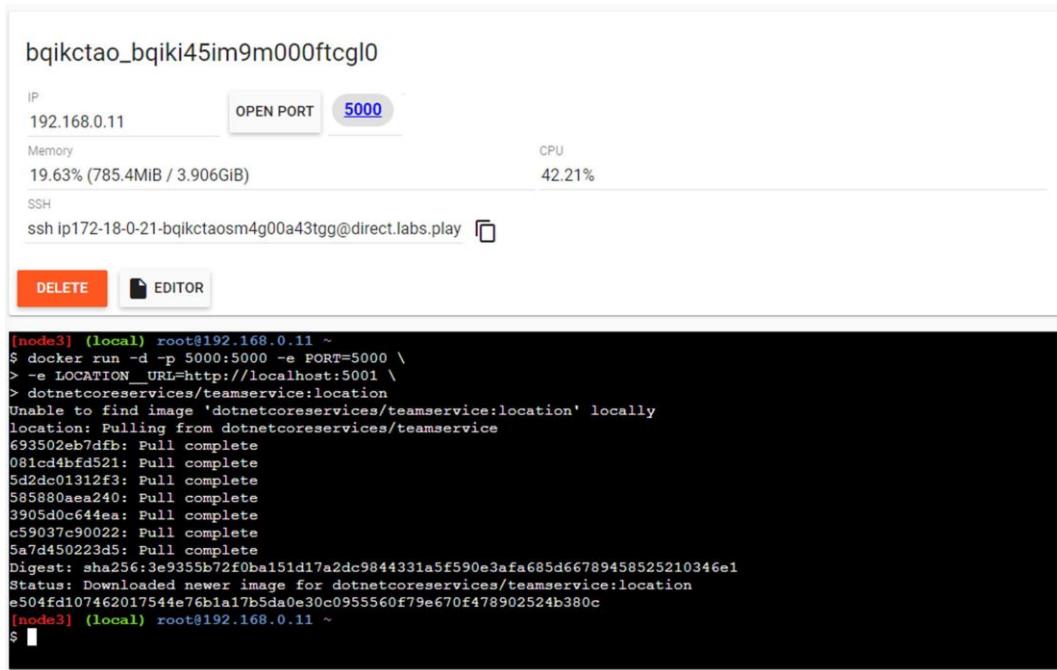


Start typing following commands

Command : To run teamservice

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

output: (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)

```
[node3] (local) root@192.168.0.11 ~
$ docker run -d -p 5001:5001 -e PORT=5001 \
> dotnetcoreservices/locationservice:nodb
Unable to find image 'dotnetcoreservices/locationservice:nodb' locally
nodb: Pulling from dotnetcoreservices/locationservice
693502eb7dfb: Already exists
081cd4b7fd521: Already exists
5d2dc01312f3: Already exists
585880aea240: Already exists
3905d0cc644ea: Already exists
c59037c90022: Already exists
dbc03883a4ca: Pull complete
Digest: sha256:5f7aca33c5e2117e04f58a59e0cf96fd20d5cbf2cf66c3cd708118d573255168
Status: Downloaded newer image for dotnetcoreservices/locationservice:nodb
16994c92a644ea5c0bb7ef9f230df16cbee84e4c48b9cd434b0e826adb3159
[node3] (local) root@192.168.0.11 ~
$ 
```

Command : to check running images in docker

docker images

output:

```
[node3] (local) root@192.168.0.11 ~
$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
dotnetcoreservices/teamservice   location  b27d0de8f2de  3 years ago  886MB
dotnetcoreservices/locationservice   nodb    03339f0ea9dd  3 years ago  883MB
[node3] (local) root@192.168.0.11 ~
$ 
```

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:

```
[node3] (local) root@192.168.0.11 ~
$ curl -H "Content-Type:application/json" -X POST -d \
> '{"id":"e52baa63-d511-417e-9e54-7aab04286281", "name":"KC"}' http://localhost:5000/teams
{"name":"KC","id":"e52baa63-d511-417e-9e54-7aab04286281","members":[]}[node3] (local) root@192.168.0.11 ~
$ 
```

Command :To confirm that team is added

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

Output

```
[node3] (local) root@192.168.0.11 ~
$ curl -H "Content-Type:application/json" -X POST -d \
> '{"id":"e52baa63-d511-417e-9e54-7aab04286281", "name":"KC"}' http://localhost:5000/teams
{"name":"KC", "id":"e52baa63-d511-417e-9e54-7aab04286281", "members":[]} [node3] (local) root@192.168.0.11 ~
$ curl http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281
{"name":"KC", "id":"e52baa63-d511-417e-9e54-7aab04286281", "members":[]} [node3] (local) root@192.168.0.11 ~
$
```

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:

```
[node1] (local) root@192.168.0.23 ~
$ 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
{"teamID":"e52baa63-d511-417e-9e54-7aab04286281", "memberID":"63e7acf8-8fae-42ce-9349-3c8593ac8292"} [node1] (local)
) root@192.168.0.23 ~
$
```

Command :To confirm member added

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

output:

```
[node1] (local) root@192.168.0.23 ~
$ curl http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281
{"name":"KC", "id":"e52baa63-d511-417e-9e54-7aab04286281", "members": [null, {"id":"63e7acf8-8fae-42ce-9349-3c8593ac8
292", "firstName":"Kirti", "lastName":"Bhatt"}]} [node1] (local) root@192.168.0.23 ~
$
```

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

```
[node1] (local) root@192.168.0.23 ~
$ curl -H "Content-Type:application/json" -X POST -d \
> '{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f", "latitude":12.0,"longitude":12.0,"altitude":10.0, "timestamp":0,
"memberId":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}' http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292
{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f", "latitude":12.0,"longitude":12.0,"altitude":10.0,"timestamp":0,"membe
rID":"63e7acf8-8fae-42ce-9349-3c8593ac8292"} [node1] (local) root@192.168.0.23 ~
$ █
```

Command : To confirm location is added in member

```
curl http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593a
```

```
[node1] (local) root@192.168.0.23 ~
$ curl http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292
[{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f", "latitude":12.0,"longitude":12.0,"altitude":10.0,"timestamp":0,"mem
berID":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}] [node1] (local) root@192.168.0.23 ~
$ █
```

