MA Practical 1

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

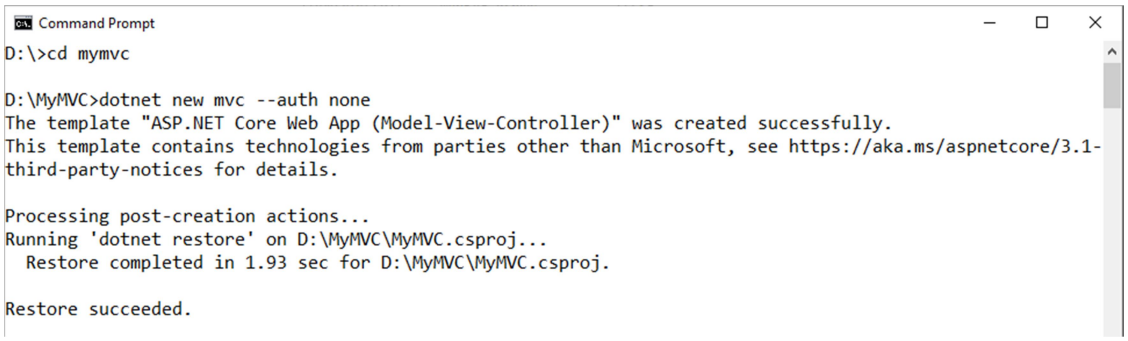
2)create folder MyMVC folder in D: drive or any other drive

3)open command prompt and perform following operations

Command: to create mvc project

dotnet new mvc --auth none

output:



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

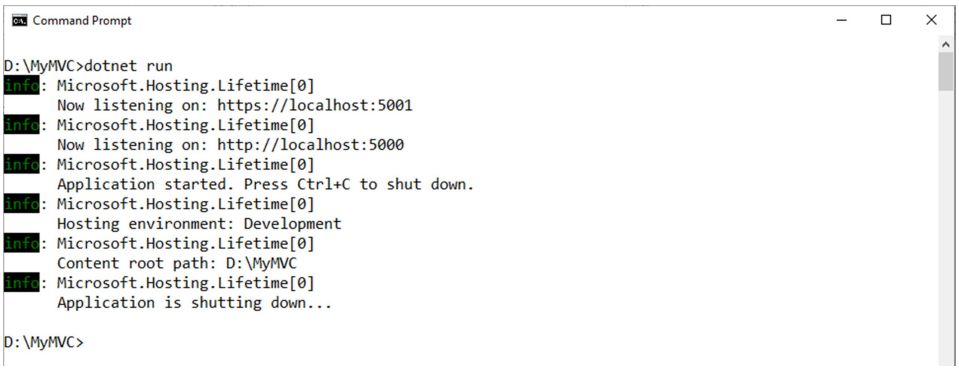
```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using MyMVC.Models;

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

5) Run the project

Now open browser and and type URL: localhost:5000

Hello World

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



7)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;}
    }
}
```

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

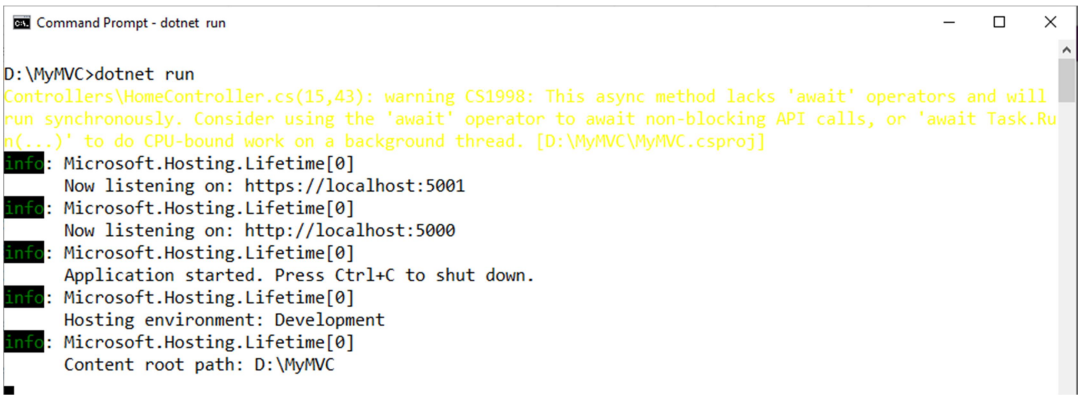
```
@{
   ViewData["Title"] = "Home Page";
}
<div>
   Symbol: @Model.Symbol <br/>
   Price: $@Model.Price <br/>
</div>
```

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

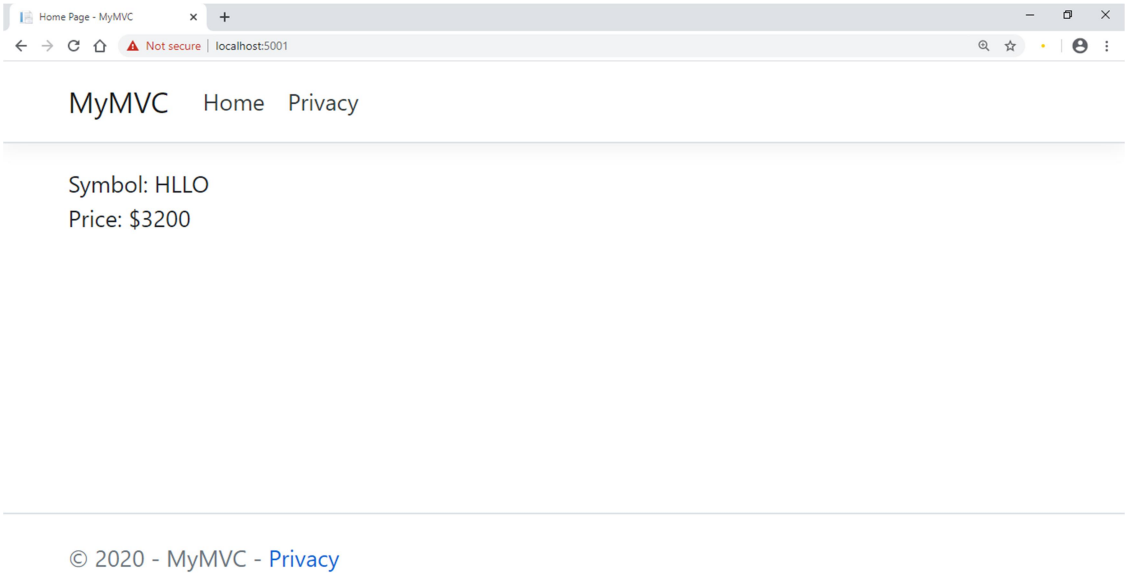
```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using MyMVC.Models;

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

10)Now run the project using

dotnet run



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



Microservices Architecture
Practical 2:
Building ASP.Net core REST API

Software requirement:

1. **Download and install**

To start building .NET apps you just need to download and install the .NET SDK (Software Development Kit version 3.0 above).
Link:
https://dotnet.microsoft.com/learn/dotnet/hello-world-tutorial/install

2. **Check everything installed correctly**

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

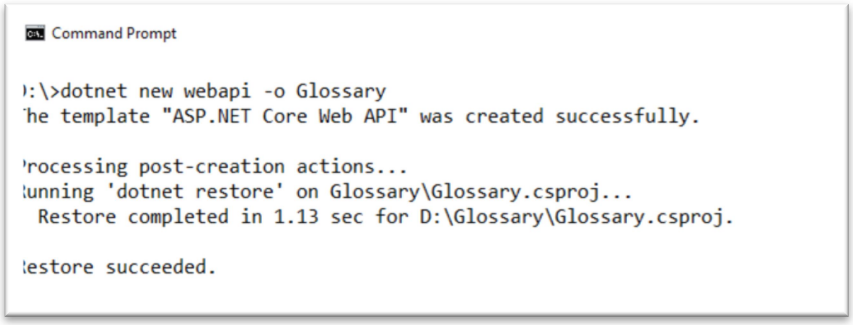
Create your web API

1. **Open two command prompts**

Command prompt 1:
Command:

dotnet new webapi -o Glossary

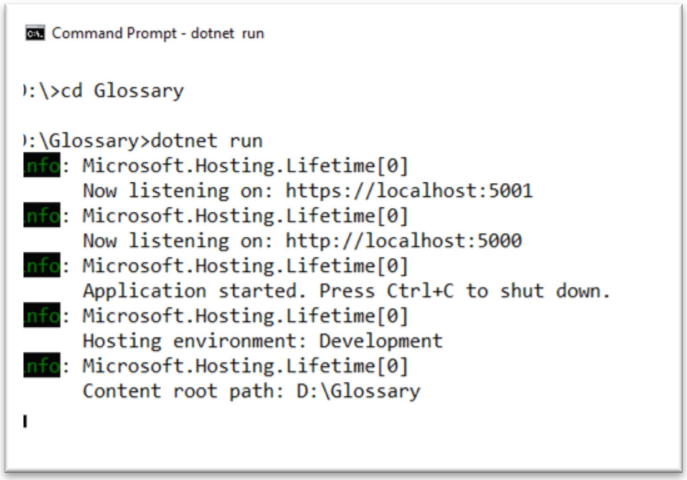
output:



Command:

cd Glossary
dotnet run

Output:

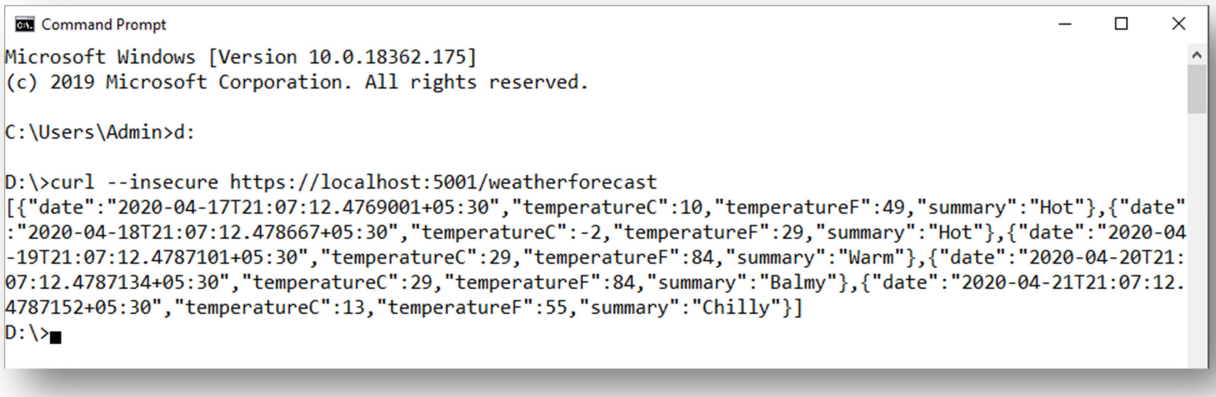


2. **Command Prompt 2: (try running ready made weatherforecast class for testing)**

Command:

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

output:



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; }
   }
}
```

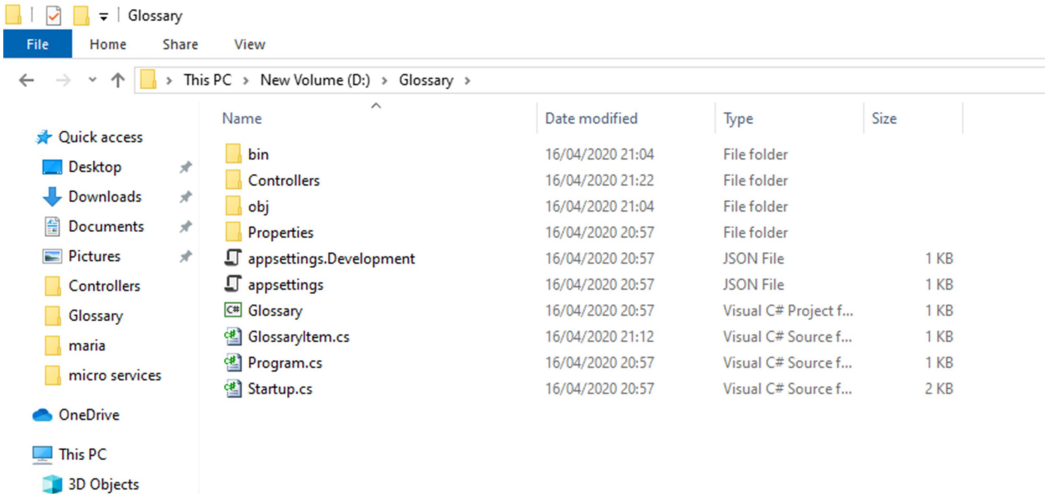
2) D:\Glossary\Controllers\ GlossaryController.cs (type it in notepad and save as all files)

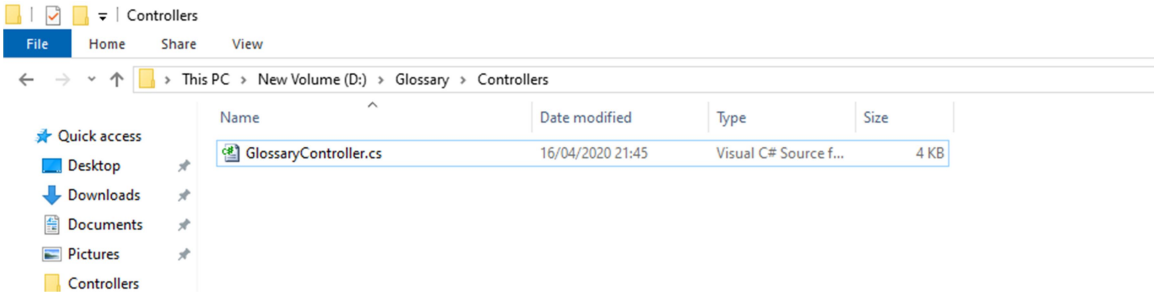
```
//Controllers/GlossaryController.cs
using System;
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using System.IO;
namespace Glossary.Controllers
{
   [ApiController]
   [Route("api/[controller]")]
   public class GlossaryController: ControllerBase
   {
      private static List<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);
    }
}
```

```
  [HttpPut]
        public ActionResult Put(GlossaryItem glossaryItem)
        {
           var existingGlossaryItem = Glossary.Find(item =>
           item.Term.Equals(glossaryItem.Term, StringComparison.InvariantCultureIgnoreCase));

           if (existingGlossaryItem == null)
           {
              return BadRequest("Cannot update a nont existing term.");
           } else
           {
              existingGlossaryItem.Definition = glossaryItem.Definition;
              return Ok();
           }
        }

      [HttpDelete]
      [Route("{term}")]
      public ActionResult Delete(string term)
      {
         var glossaryItem = Glossary.Find(item =>
              item.Term.Equals(term, StringComparison.InvariantCultureIgnoreCase));

         if (glossaryItem == null)
         {          return NotFound();
         }
         else
         {         Glossary.Remove(glossaryItem);
            return NoContent();
         }
      }
   }
}
```

Output:





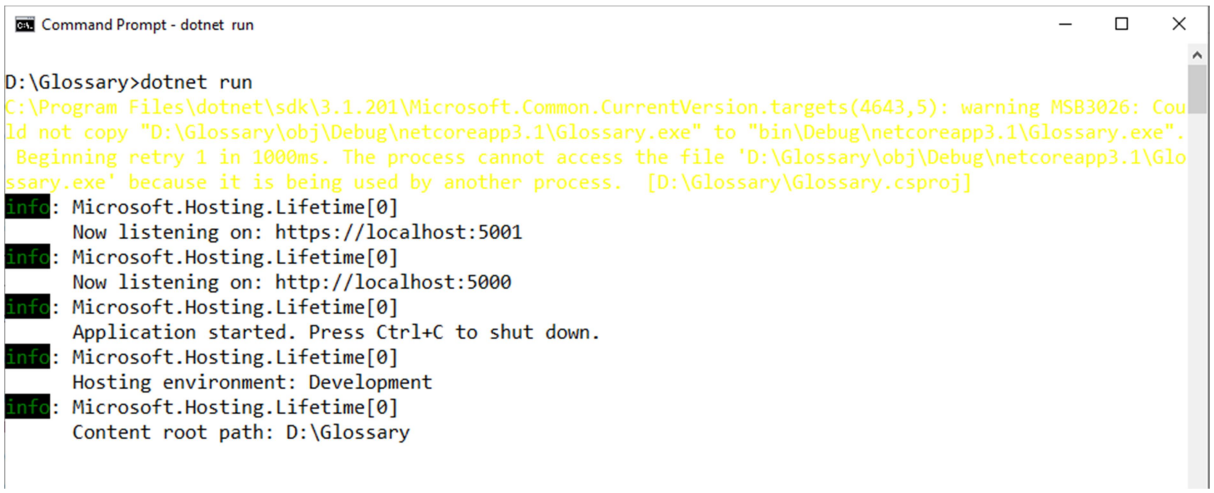
3.Now stop running previous dotnet run on command prompt 1 using Ctrl+C. and Run it again for new code.

On Command prompt1:

Command:

dotnet run

output:



On Command prompt2:

1) Getting a list of items:

Command:

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

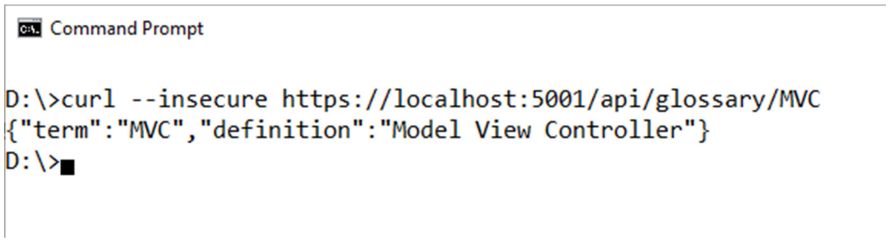
Output:

2) Getting a single item

Command:

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

Output:



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:

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

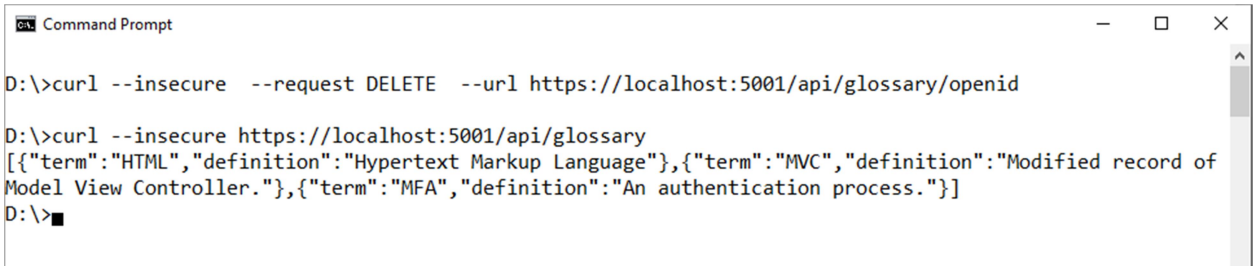
Output:

5) Delete Item

Command:

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

Output:

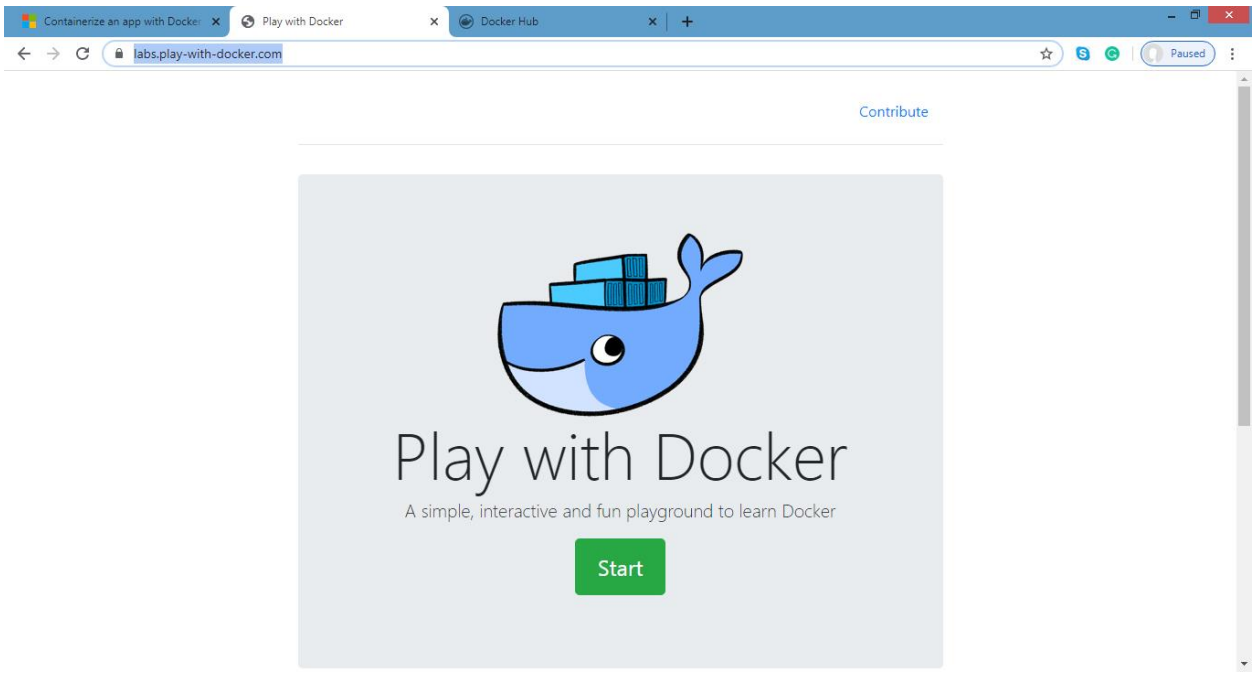


MA practical 3:

Working with Docker

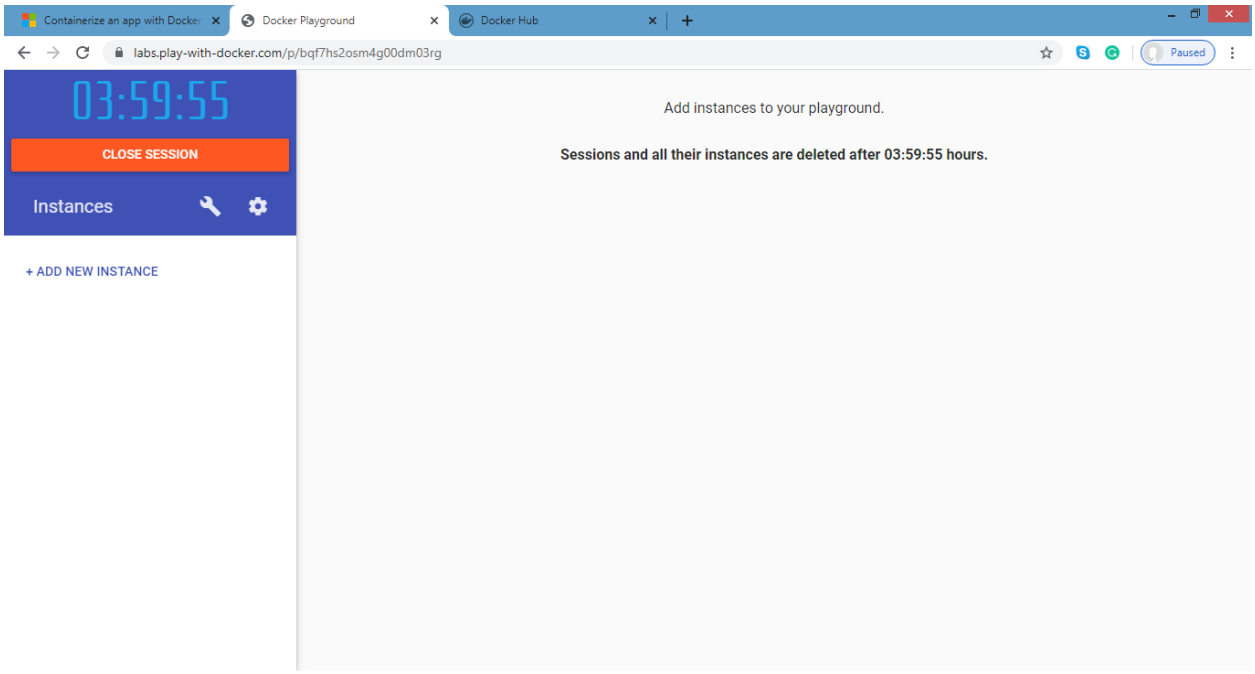
1) create Docker Hub account (sign up)

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



Click on start

3)add new instance



4)perform following:

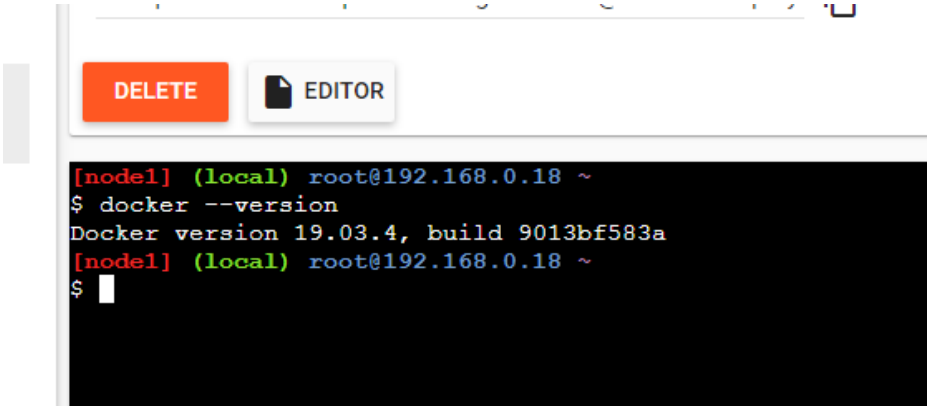
Method1:

To pull and push images using docker

Command: to check docker version

docker –version

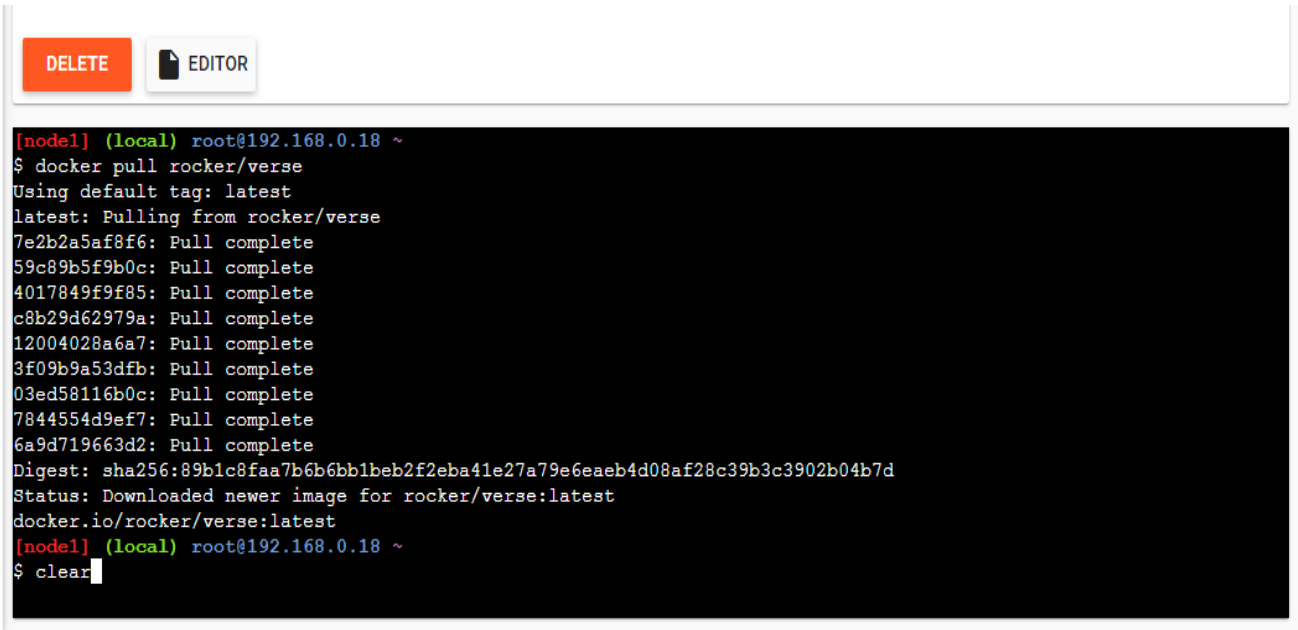
output:



Command: to pull readymade image

docker pull rocker/verse

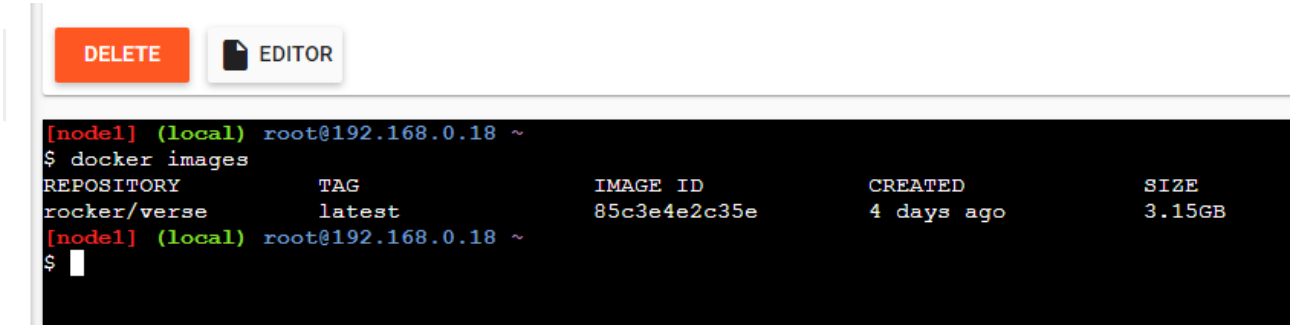
output:



Command: to check images in docker

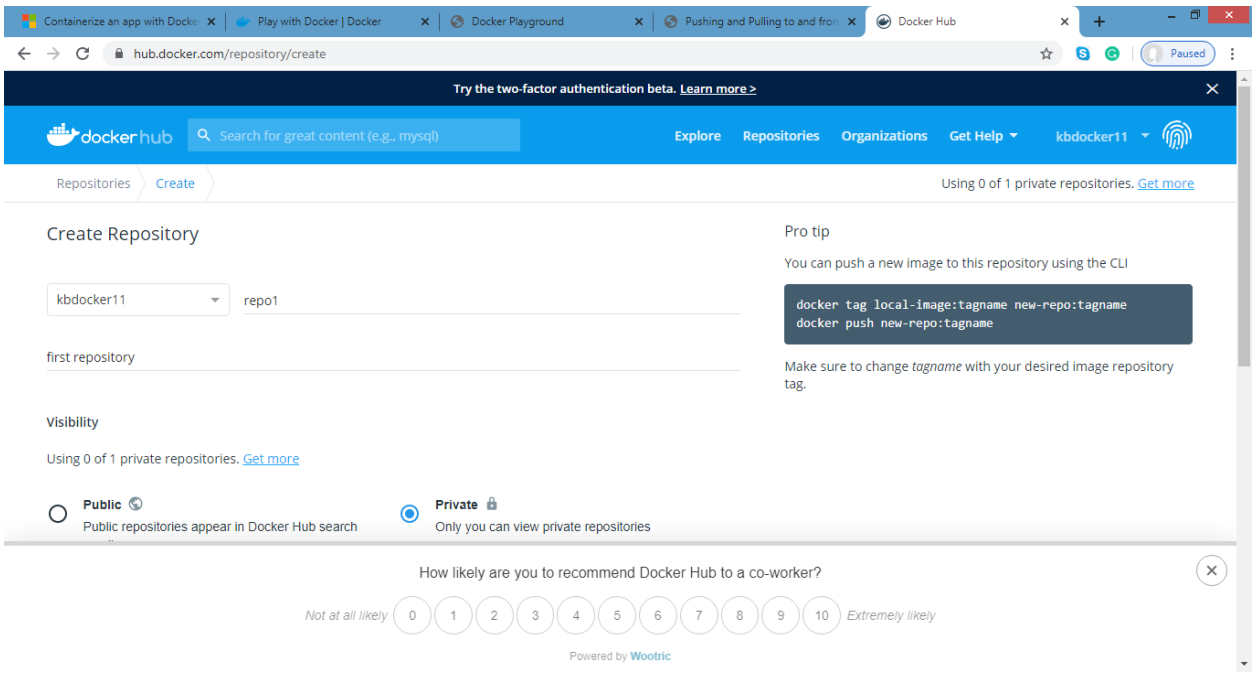
docker images

output:



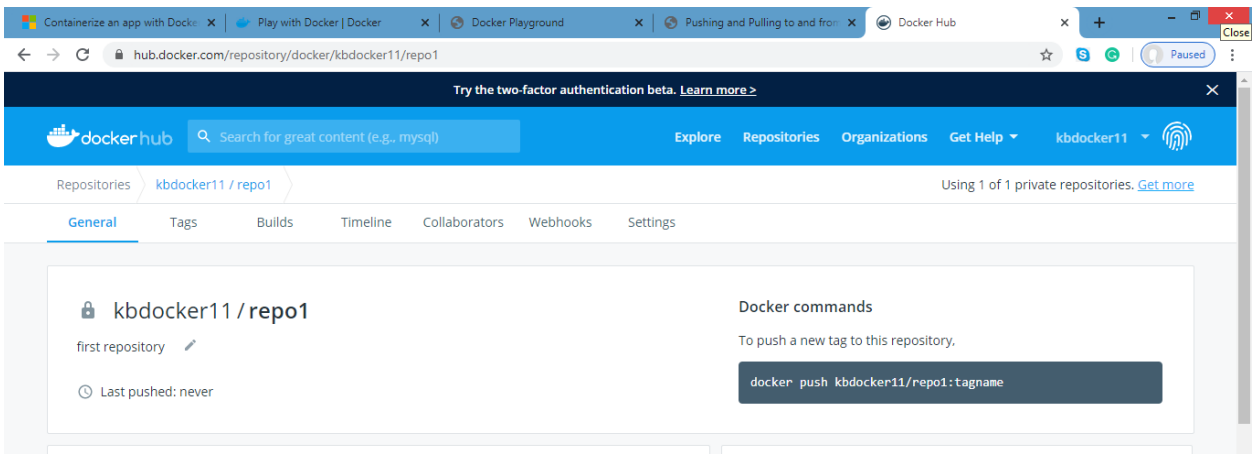
Now Login to docker hub and create repository

Output:



Click on Create button

Now check repository created



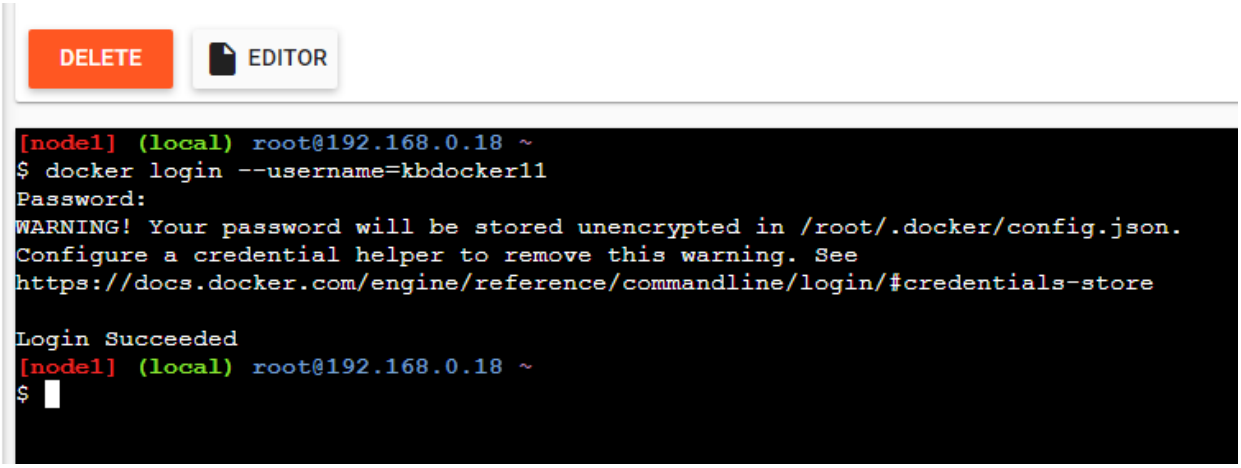
Command: to login to your docker account

docker login –username=kbdocker11

password:

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

Output:

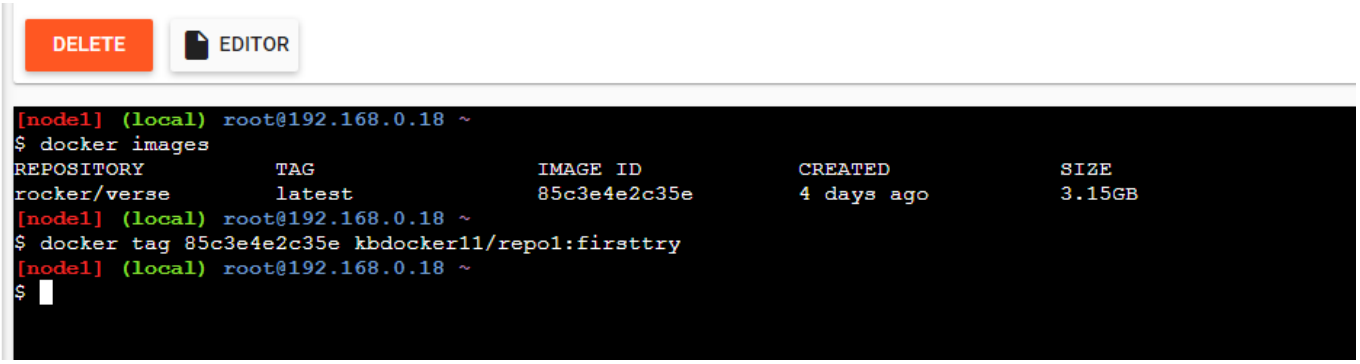


Command : to tag image

docker tag 8c3e4e2c3e kbdocker11/repo1:firsttry

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

Output:

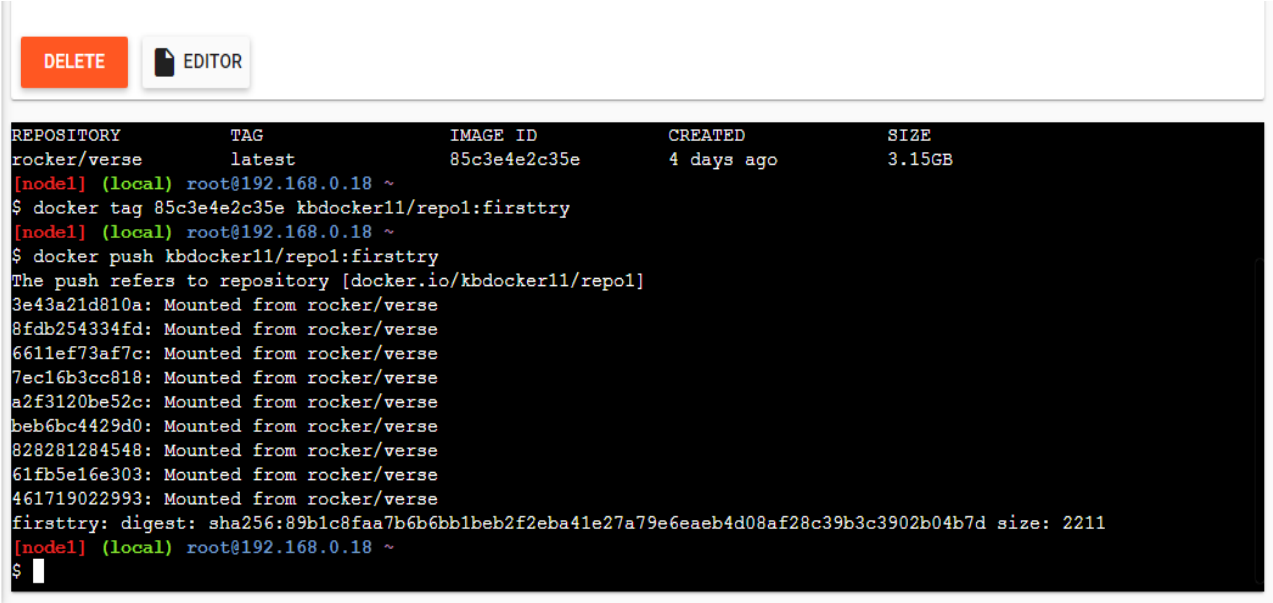


Command: to push image to docker hub account

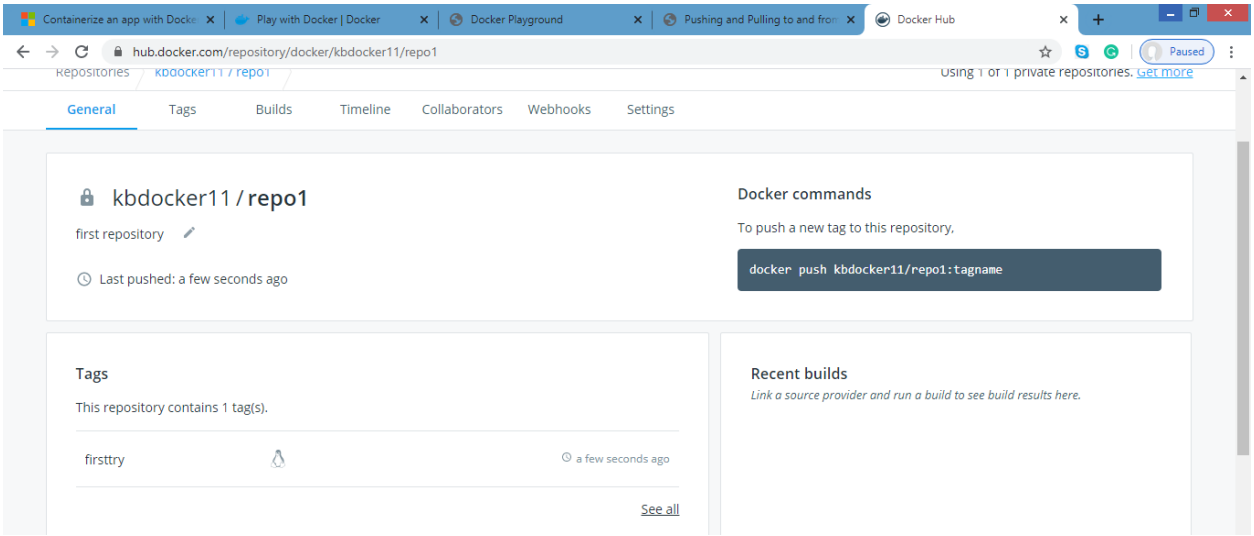
docker push kbdocker11/repo1:firsttry

note: firsttry is tag name created above.

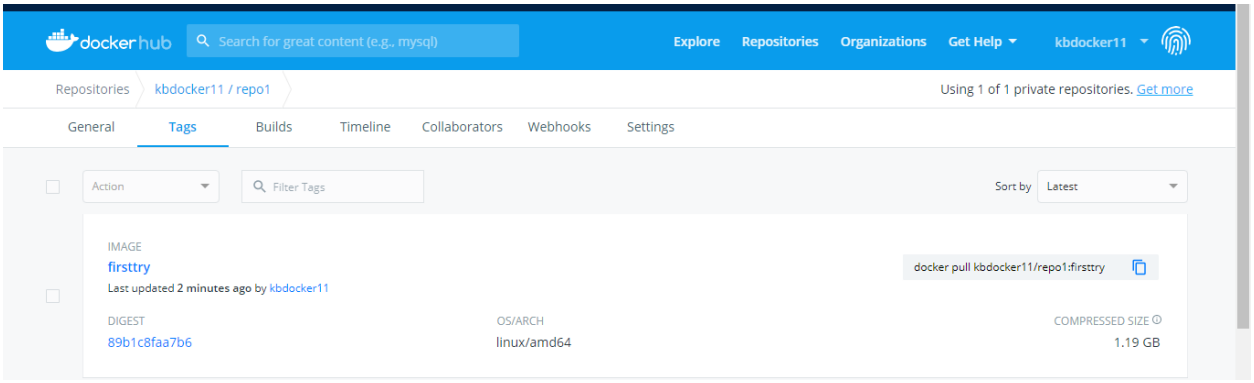
Output



Check it in docker hub now



Click on tags and check



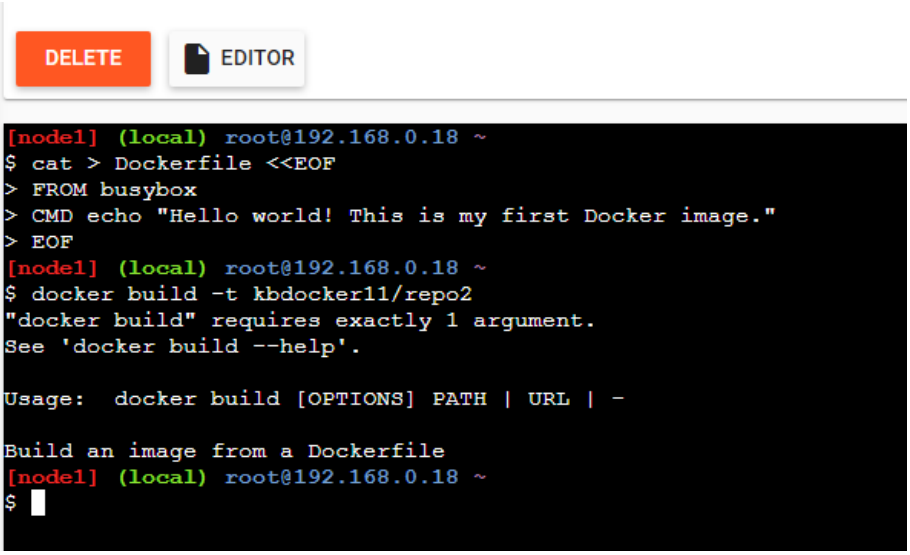
Method 2:

Build an image then push it to docker and run it

Command : to create docker file

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

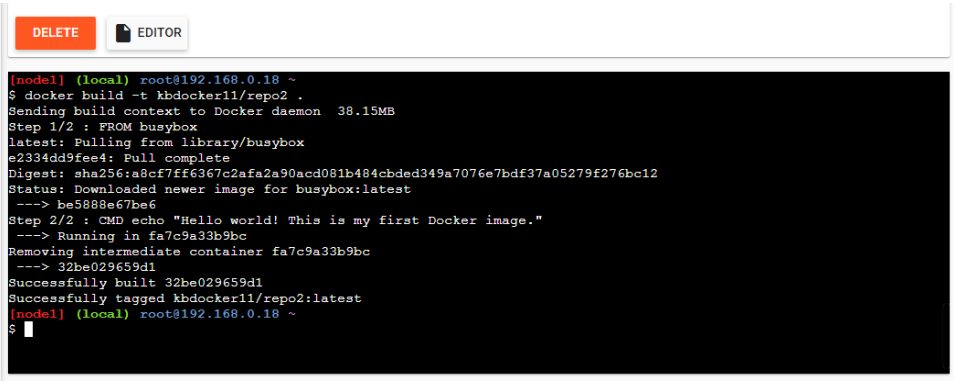
Output:



Command : to build image from docker file

dokcer build –t kbdocker11/repo2 .

Output:



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
```

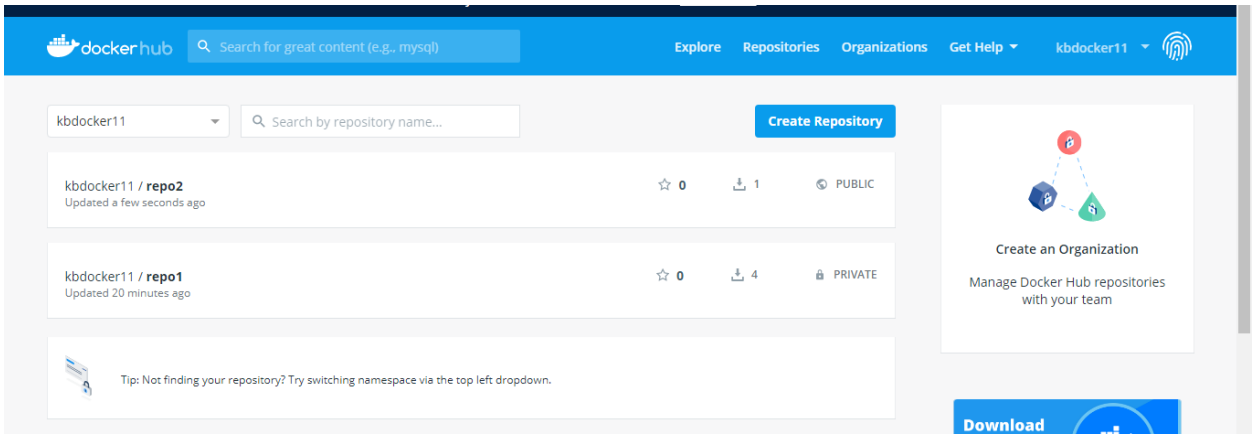
Command: to push image to docker hub

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.

MA PRACTICAL 4

(Installing software packages on Docker, Working with Docker Volumes and Networks)

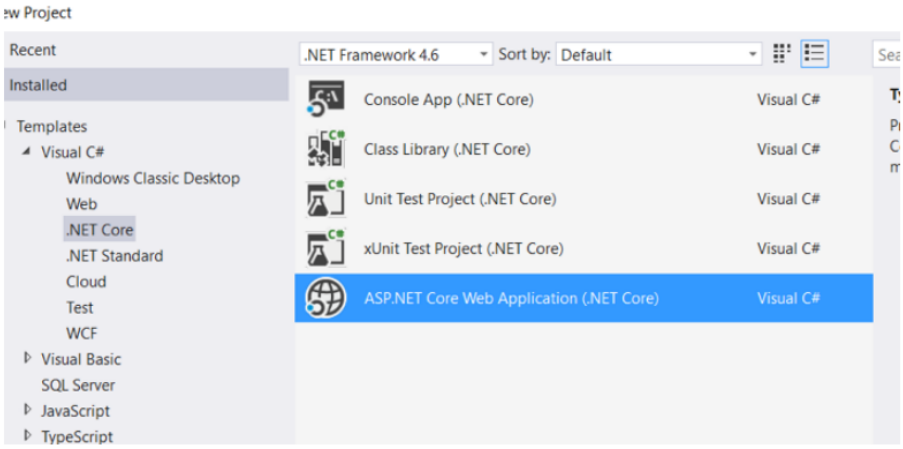
What is the software required for Windows?

- 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

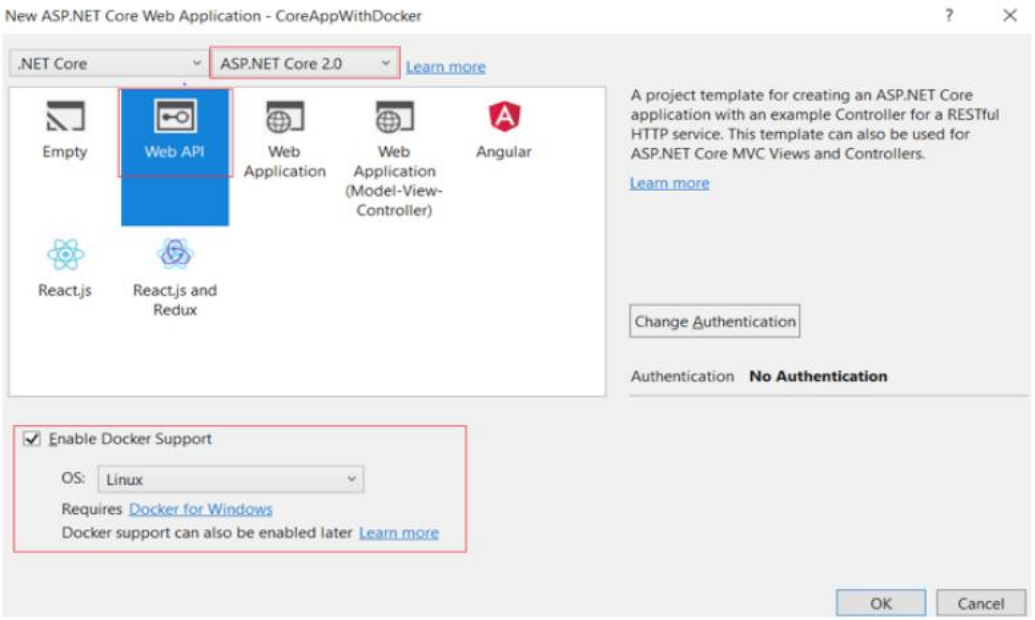
How to create a new microservice using .NET Core and then build and run it using Docker

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

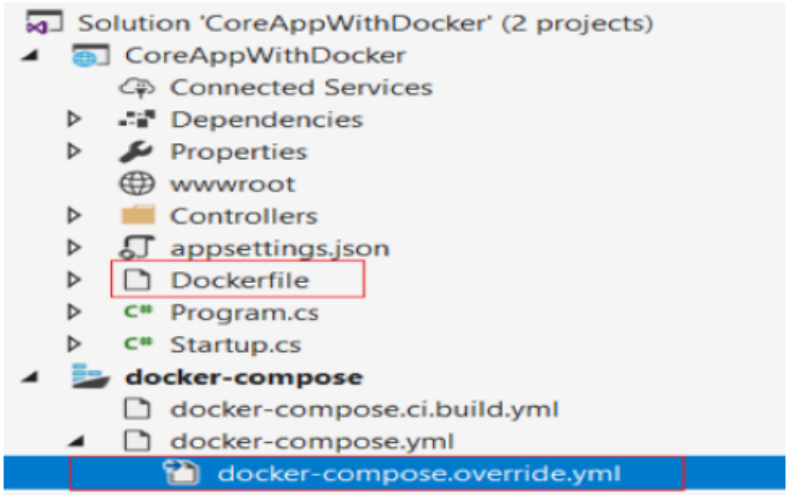
Select "ASP.NET Core Web Application (.NET Core)" from the drop-down menu.



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.”

docker-compose.override.yml
This file is used when running an application using Visual Studio.

```
FROM microsoft/aspnetcore:2.0
ARG source
WORKDIR /app
EXPOSE 80
COPY ${source:-obj/Docker/publish} .
ENTRYPOINT ["dotnet", "CoreAppWithDocker.dll"]
```

```
version: '3'

services:
  coreappwithdocker:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
    ports:
      - "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.

Dockerfile

```
FROM microsoft/aspnetcore:2.0
ARG source
WORKDIR /app
ENV ASPNETCORE_URLS http://+:83
EXPOSE 83
COPY ${source:-obj/Docker/publish} .
ENTRYPOINT ["dotnet", "CoreAppWithDocker.dll"]
```

docker-compose.override.yml

```
version: '3'

services:
  coreappwithdocker:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
    ports:
      - "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.

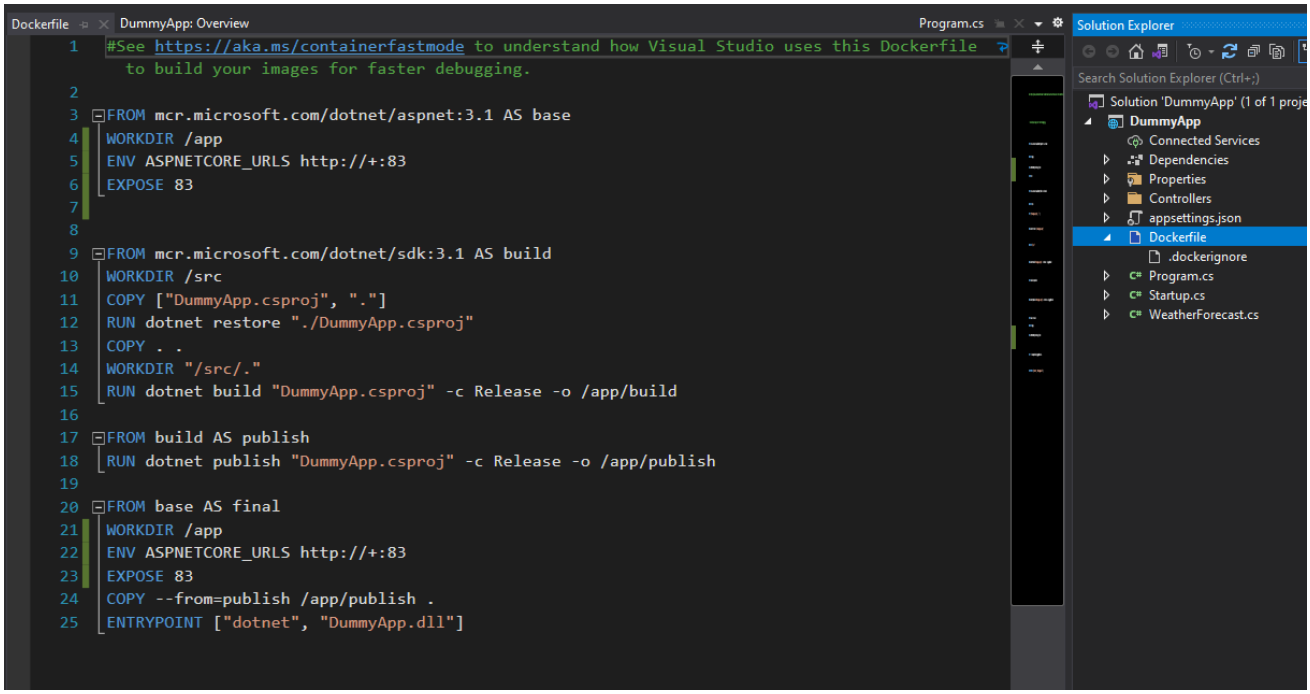
ARG source is the argument which helps to pass data to the image.

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

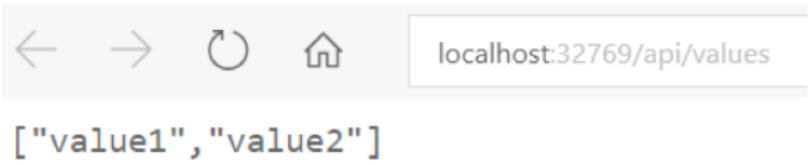
COPY will copy the DLLs of the application to the root directory (here dot represents root) of the image.

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



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 director
ies added to build context will have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions f
or 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

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

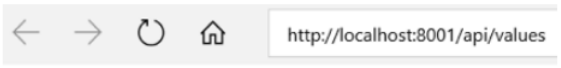
Run the image in a container: docker run -d -p 8001:83 –name core1 coreappimage

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

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

Check the running container: Docker PS

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

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

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




```
["value1","value2"]
```

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

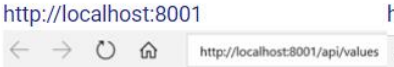
```
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker run -d -p 8002:83 --name core2 coreappimage
3f26f83e36f1bda6aa40d67111a2be71ab66c1b775cc080064062cd9b26b7d40

D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker run -d -p 8003:83 --name core3 coreappimage
647d658fa98b989ef58982e7a580ded5278896171dd60bbcbc8fdf14c486c77b

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
        NAMES
647d658fa98b        coreappimage        "dotnet CoreAppWit..."   51 seconds ago      Up 51 seconds       0.0.0.0:8003->
83/tcp   core3
3f26f83e36f1        coreappimage        "dotnet CoreAppWit..."   About a minute ago  Up About a minute   0.0.0.0:8002->
83/tcp   core2
2c35a9660288        coreappimage        "dotnet CoreAppWit..."   38 minutes ago      Up 38 minutes       0.0.0.0:8001->
83/tcp   core1

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

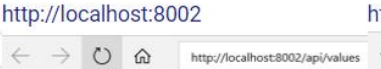
We can see that there are 3 containers running for the same image at 8001, 8002, and 8003.

http://localhost:8001



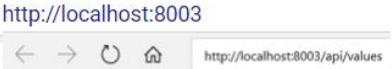

```
["value1","value2"]
```

http://localhost:8002




```
["value1","value2"]
```

http://localhost:8003




```
["value1","value2"]
```

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.

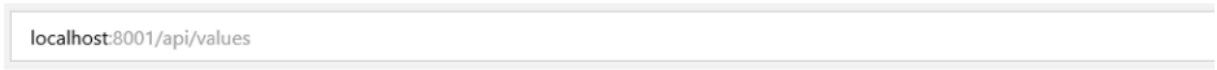
Check running containers: docker ps:

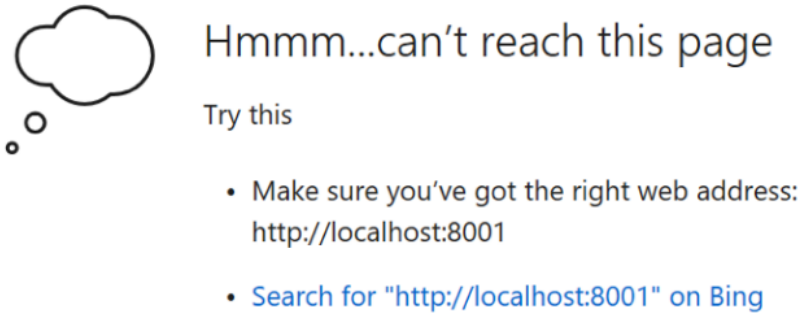
```
D:\Rakhi\Blogs\CoreAppWithDocker\CoreAppWithDocker>docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
       NAMES
647d658fa98b        coreappimage        "dotnet CoreAppWit..."   23 minutes ago      Up 23 minutes       0.0.0.0:8003->8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



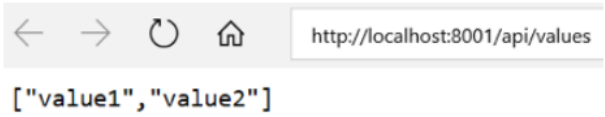


Start Container:

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.

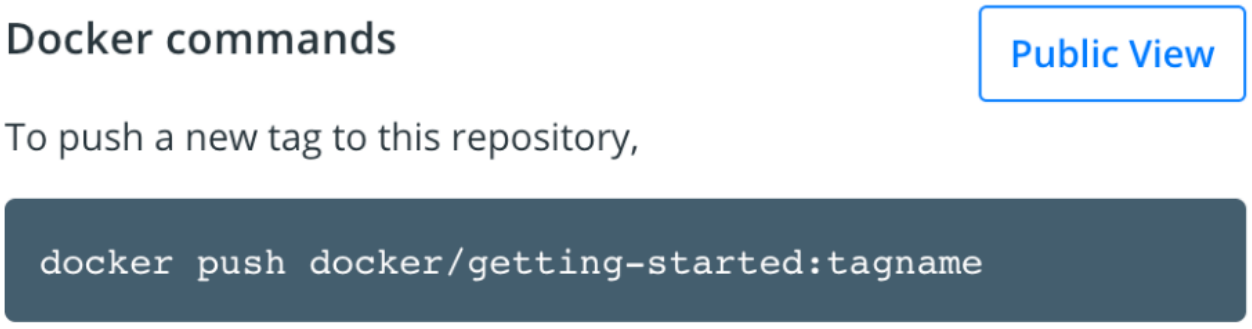
1. Sign up or Sign in to Docker Hub.
2. Click the **Create Repository** button.
3. For the repo name, use `getting-started`. Make sure the Visibility is `Public`.

   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.



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
   ```

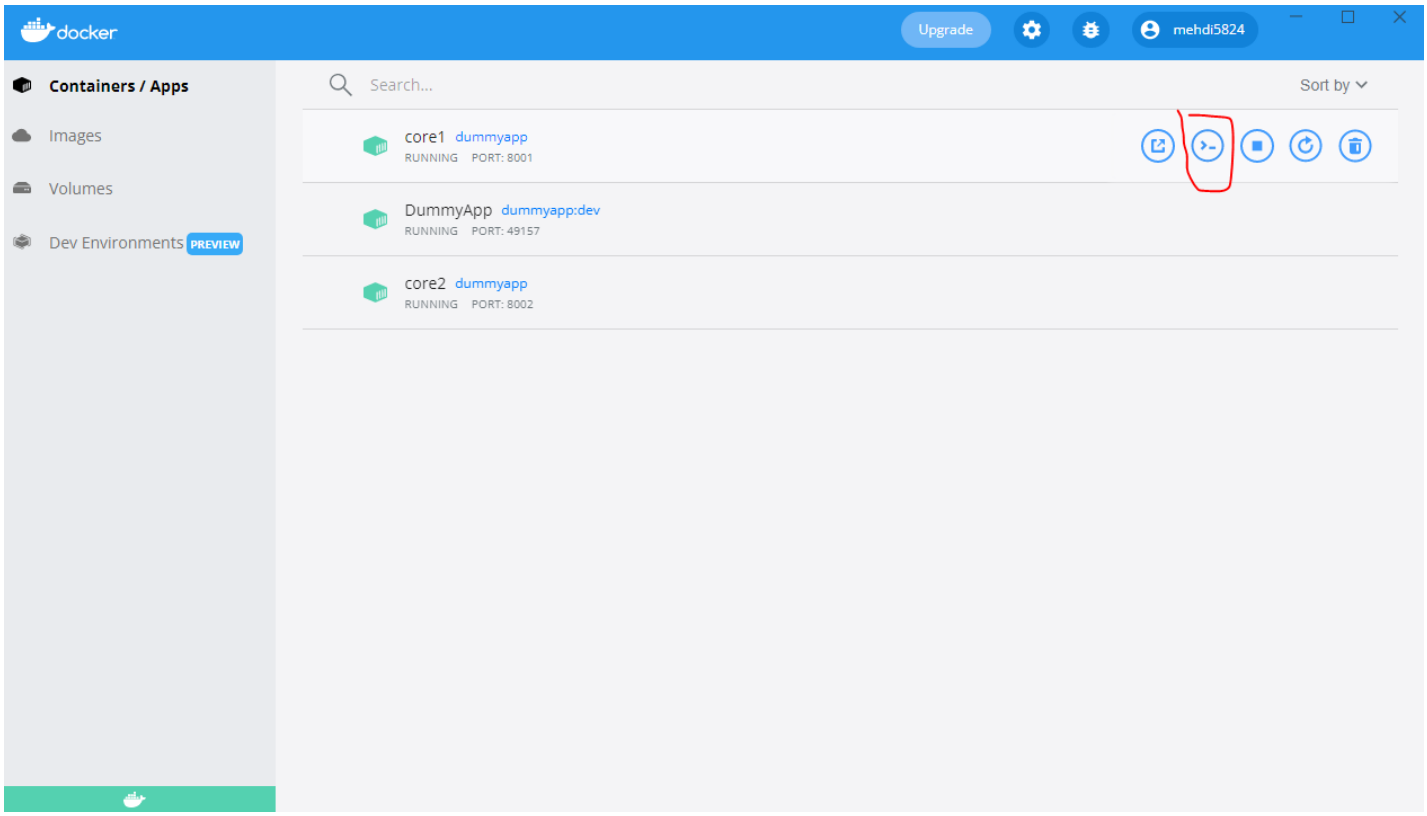
4. Now try your push command again. If you're copying the value from Docker Hub, you can drop the `tagname` portion, as we didn't add a tag to the image name. If you don't specify a tag, Docker will use a tag called `latest`.

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

Part 2

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
```

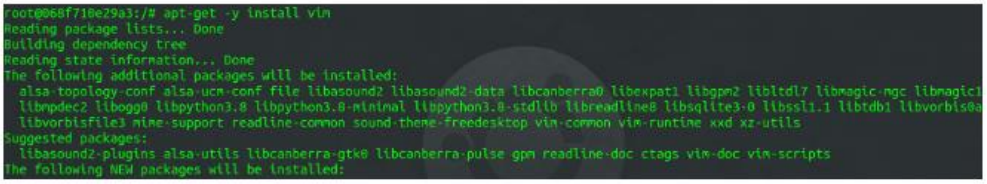
```
root@068f710e29a3:/# apt-get -y update
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [107 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [111 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports 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 B]
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 B]
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
```

Installing Firefox



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
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     aba83257ee43df3f86bfea2b09c1d1ffe5a59b9ced82c6b7ea5f458e9e298e72
local     d247fdb49990ed914b54fffe365671c1f3b773d5871038817e18d36cf6e288bf
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

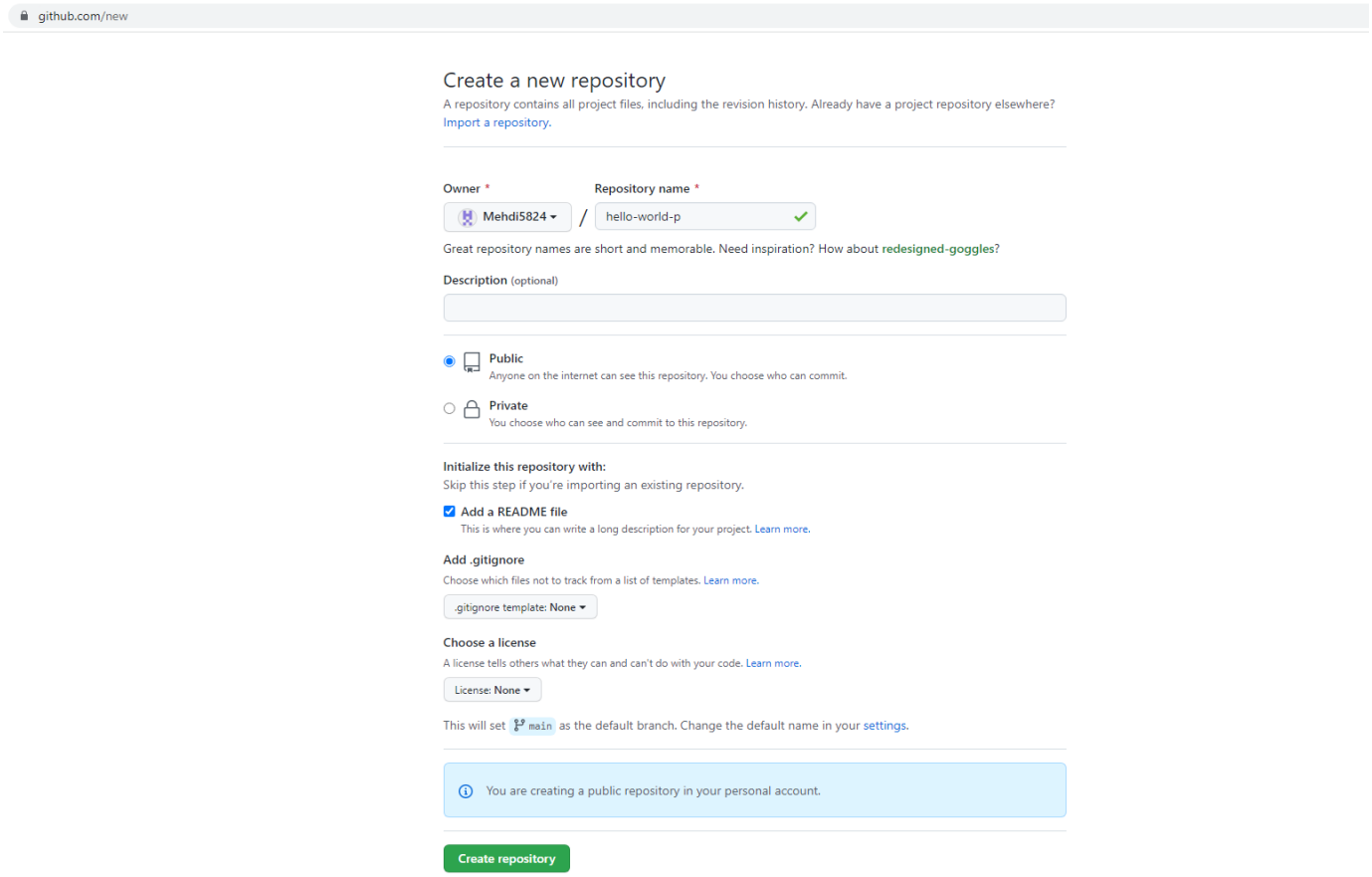
f) To remove all unused networks
   ➔ docker network prune

g) To remove one or more networks
   ➔ docker network rm

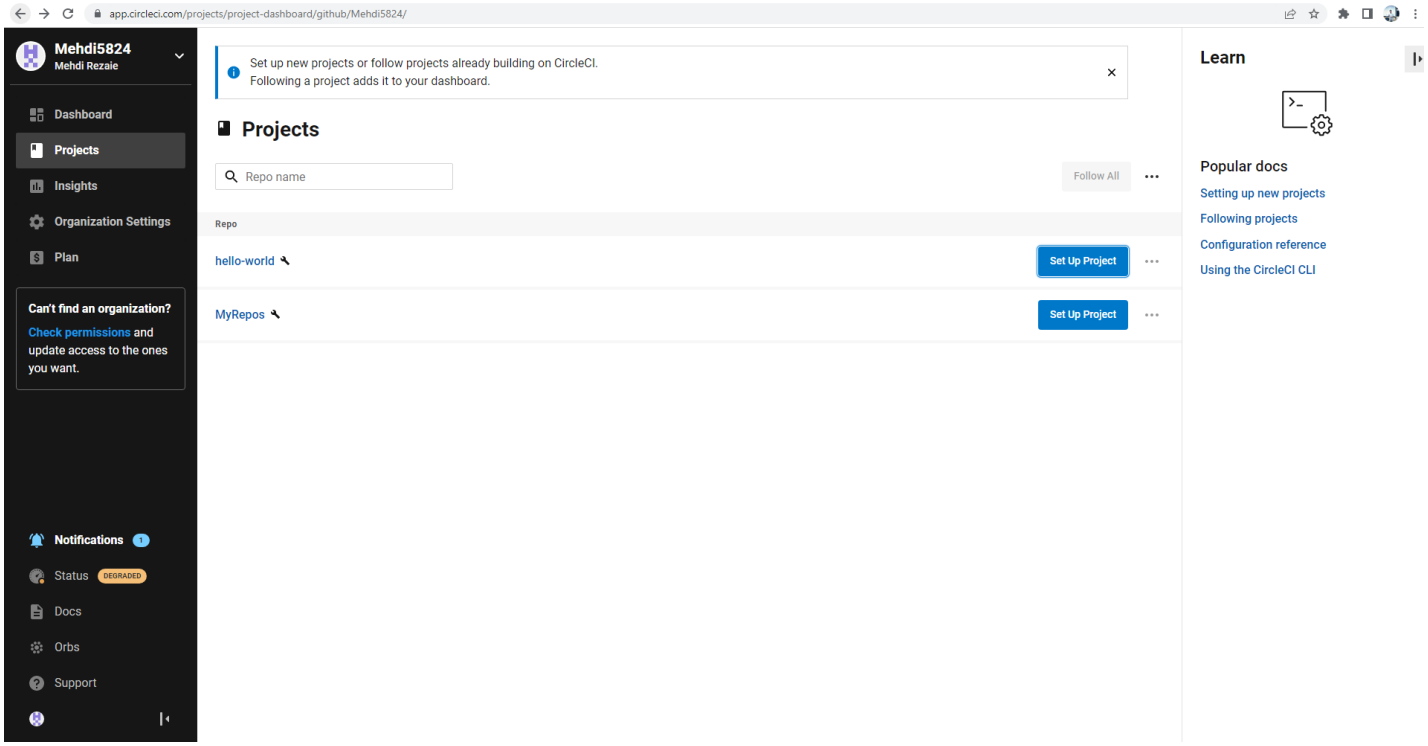
Practical 6 (Working with Circle CI for continuous integration)

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.

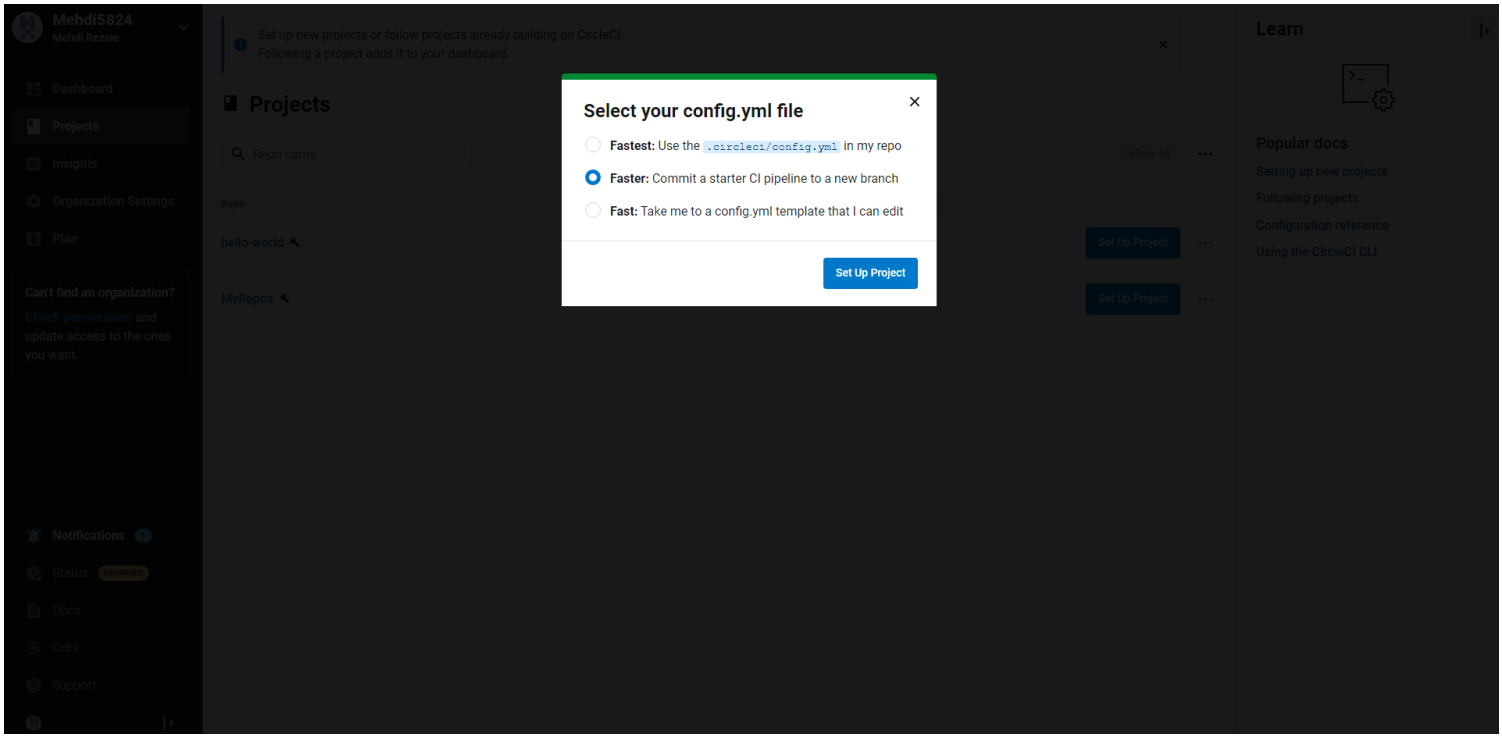


Login to Circle CI https://app.circleci.com/ Using GitHub Login, Once logged in navigate to Projects.



Step 2 - Set up CircleCI

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



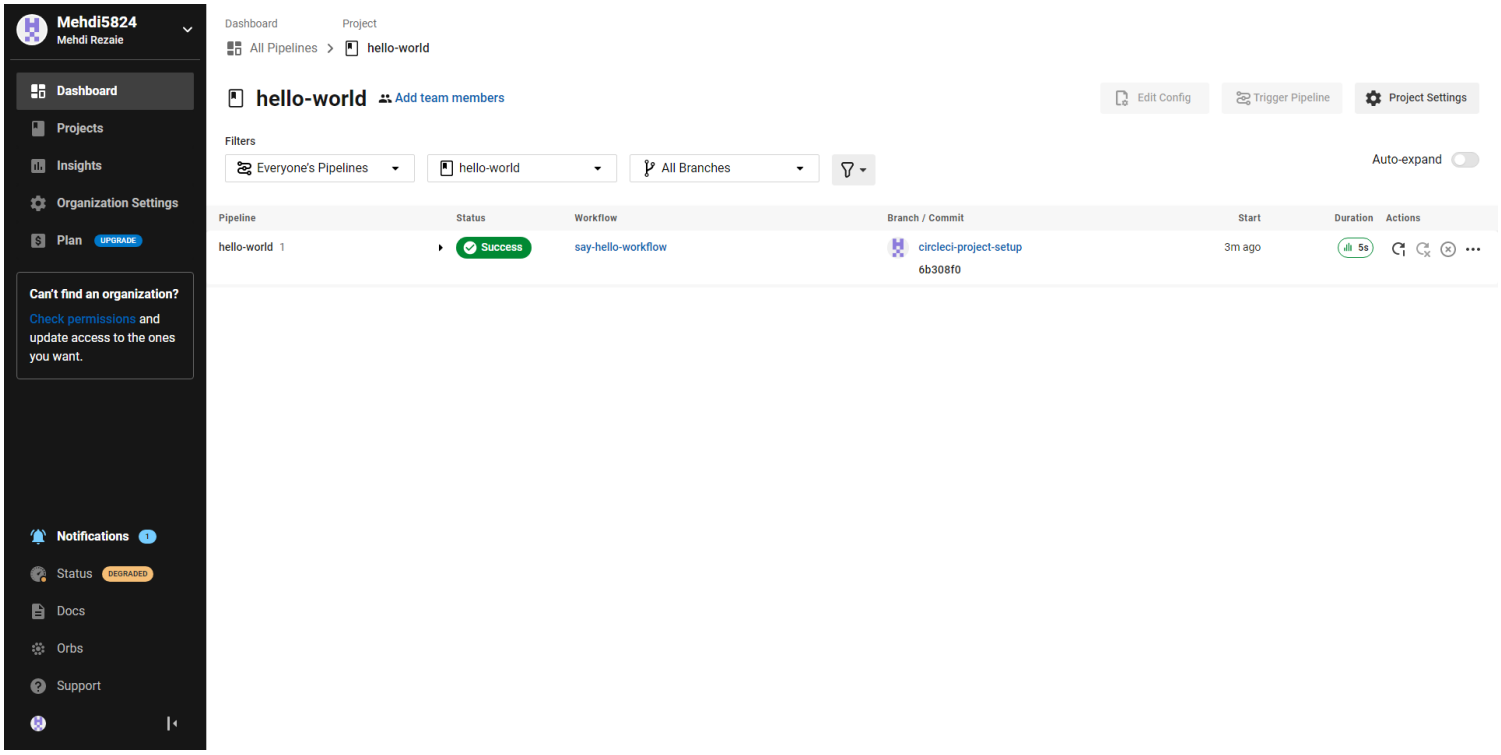
Step 3 - Your first pipeline

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

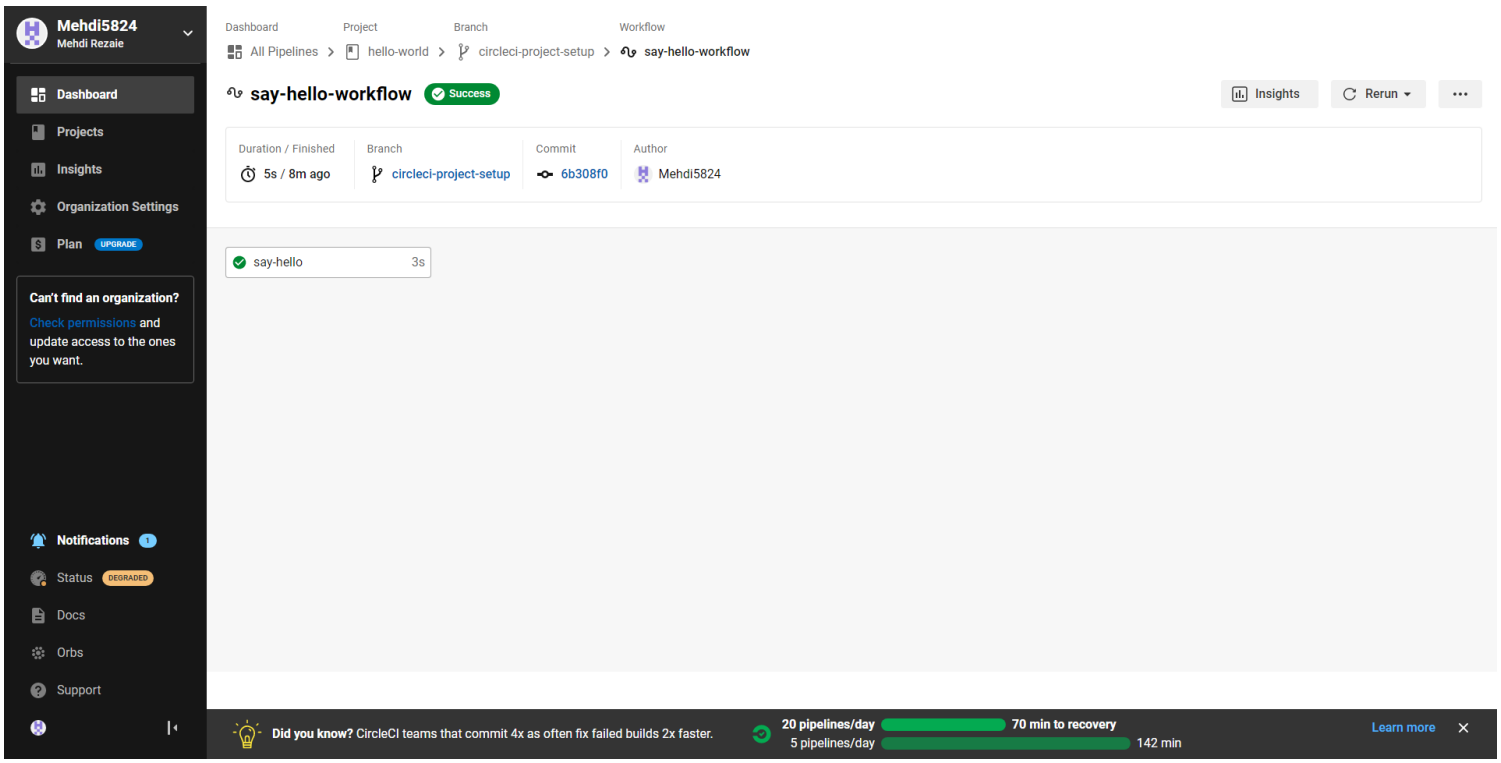
Within this workflow, the pipeline ran one job, called say-hello. Click say-hello to see the steps in this job:

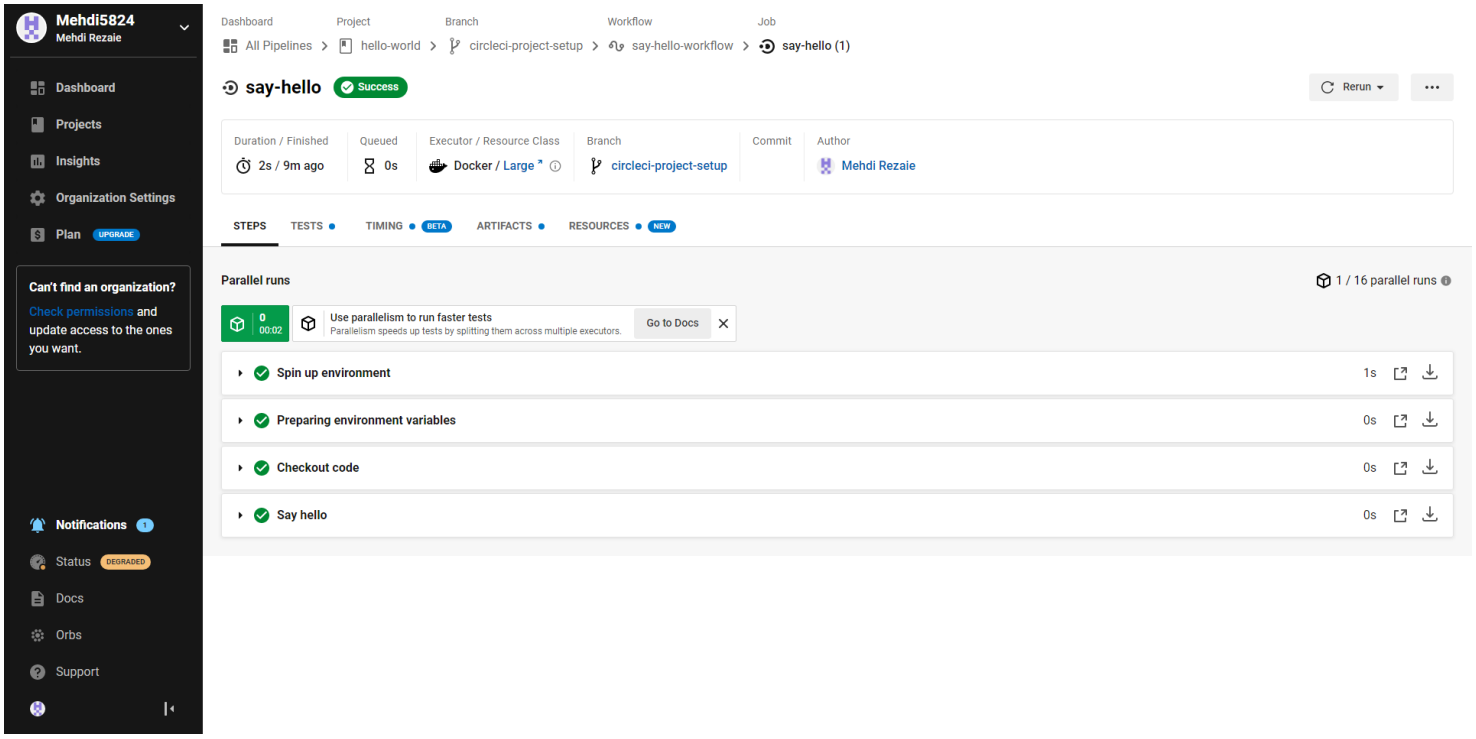
a. Spin up environment
b. Preparing environment variables
c. Checkout code
d. Say hello

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

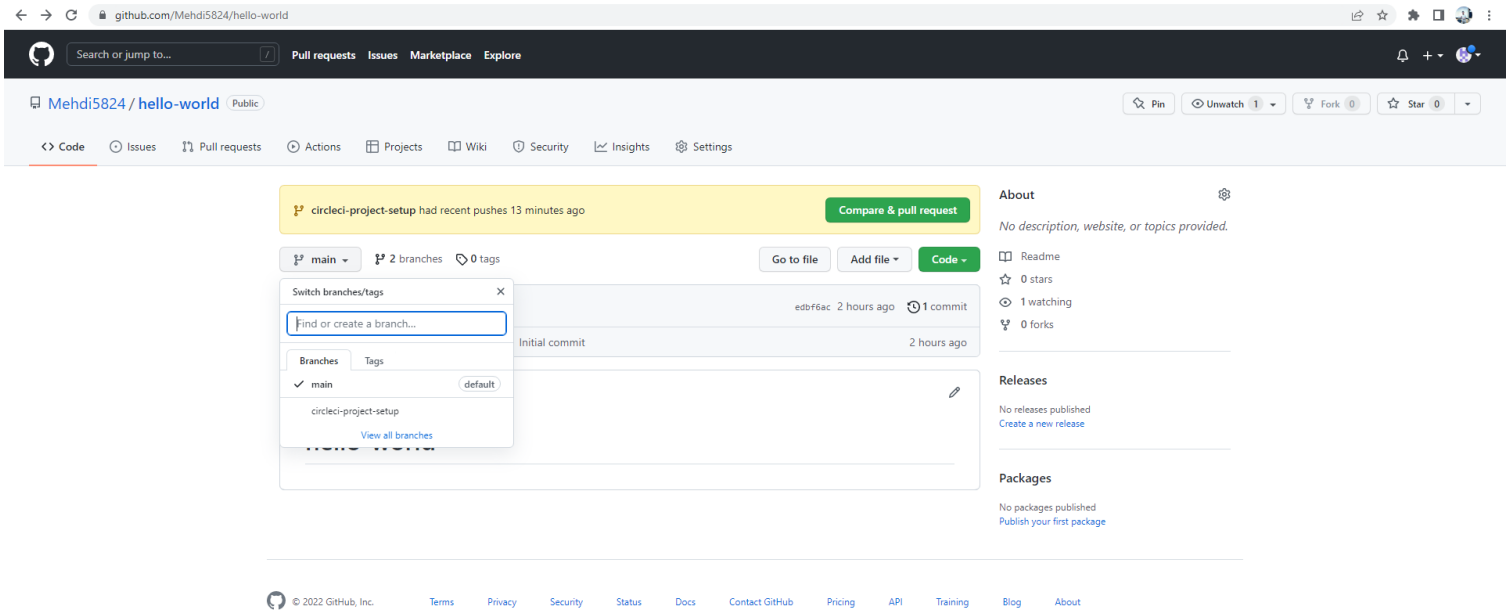


Select “say-hello” Job with a green tick





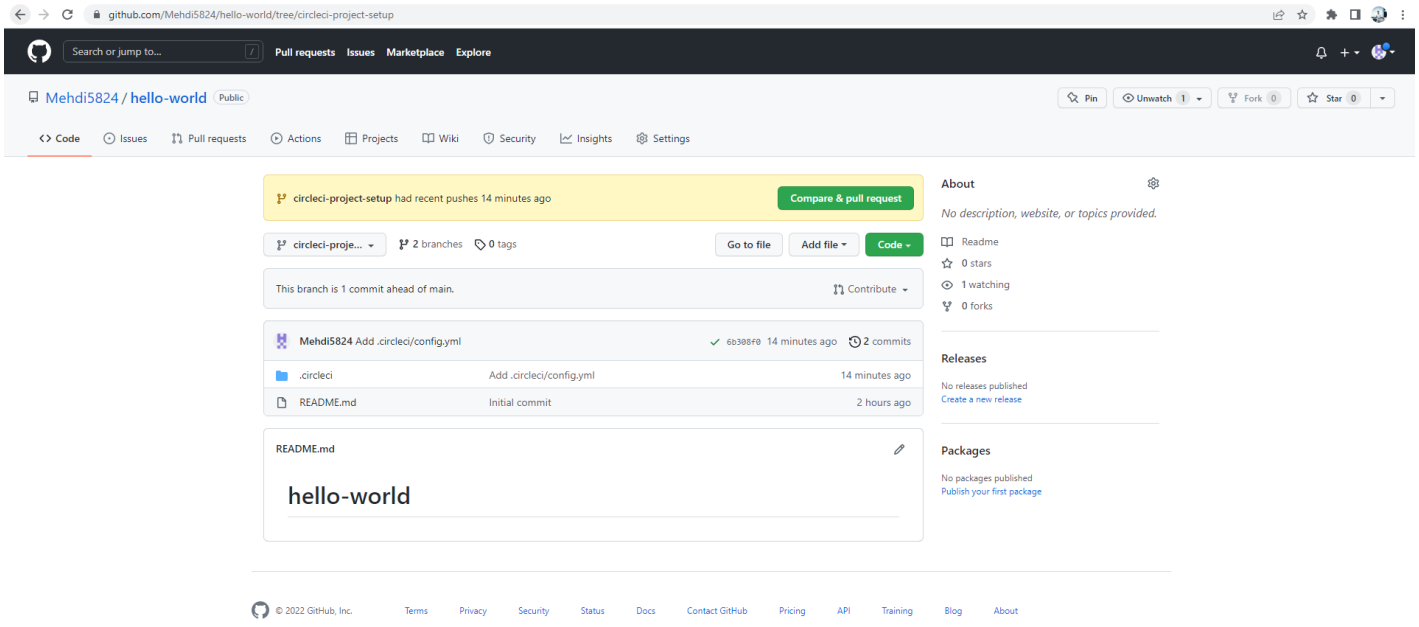
Select Branch and option circleci-project-setup

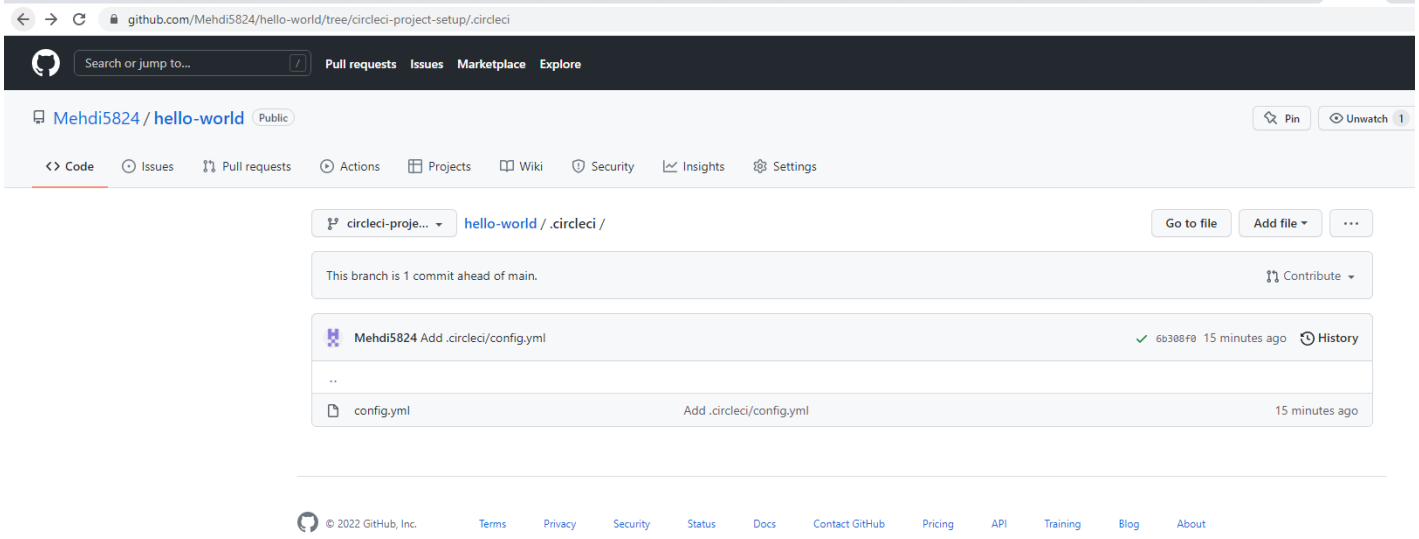


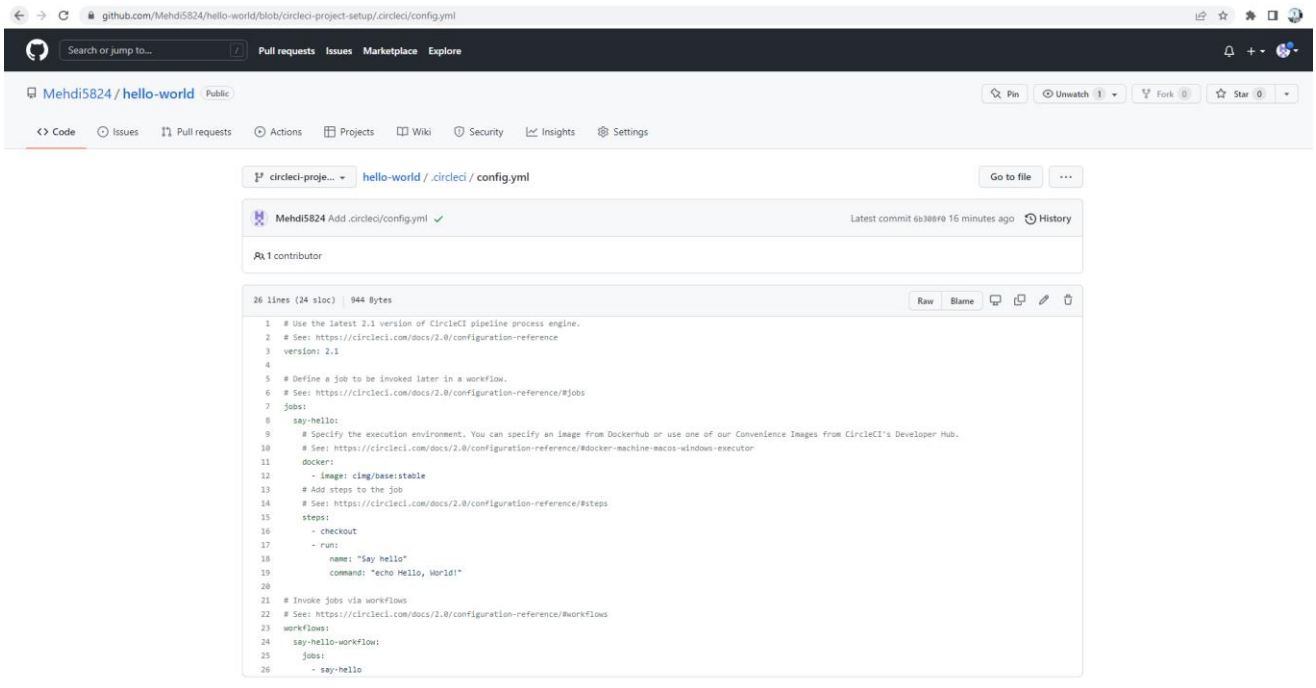
Step 4 - Break your build

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

It is possible to edit files directly on GitHub.


github.com/Mehdi5824/hello-world/tree/circleci-project-setup
Search or jump to...
Pull requests
Issues
Marketplace
Explore
Mehdi5824 / hello-world
Public
Pin
Unwatch 1
Fork 0
Star 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
circleci-project-setup had recent pushes 14 minutes ago
Compare & pull request
circleci-proje...
2 branches
0 tags
Go to file
Add file
Code
This branch is 1 commit ahead of main.
Contribute
Mehdi5824 Add .circleci/config.yml
6b308f0 14 minutes ago
2 commits
.circleci
Add .circleci/config.yml
14 minutes ago
README.md
Initial commit
2 hours ago
README.md
hello-world
About
No description, website, or topics provided.
Readme
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
© 2022 GitHub, Inc.
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About

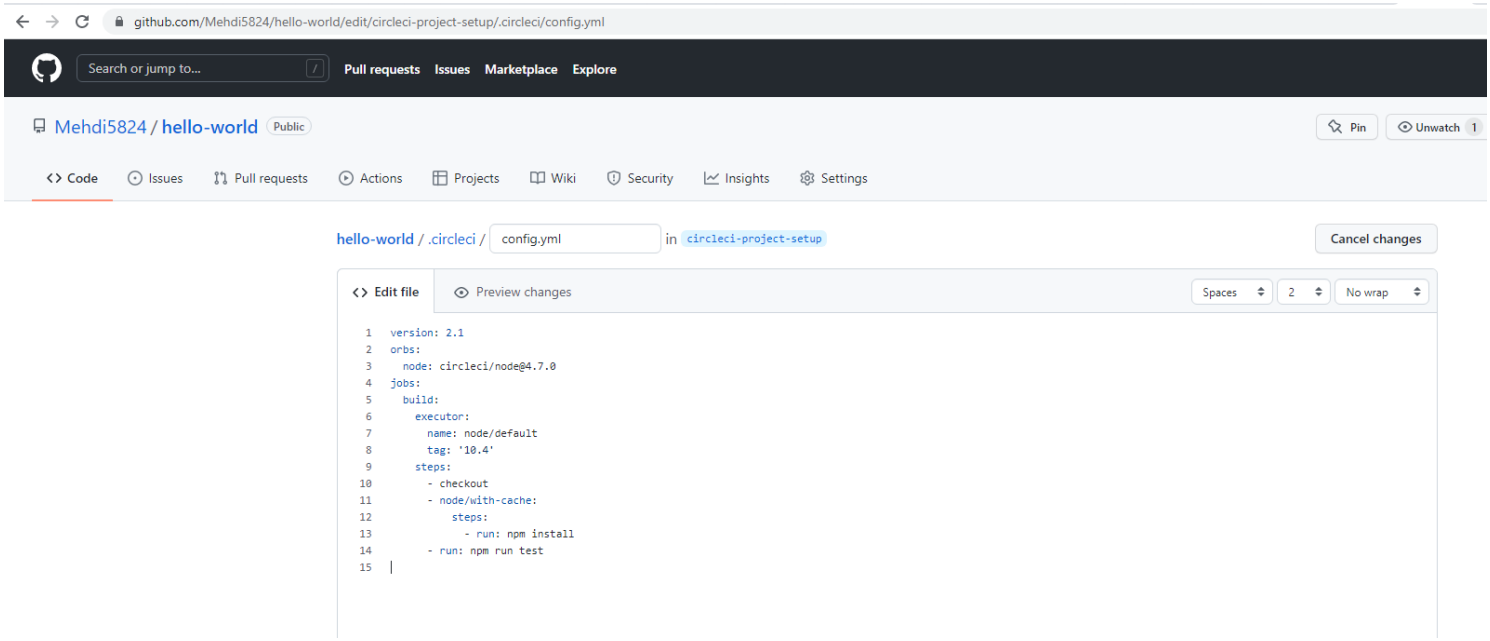

github.com/Mehdi5824/hello-world/tree/circleci-project-setup/.circleci
Search or jump to...
Pull requests
Issues
Marketplace
Explore
Mehdi5824 / hello-world
Public
Pin
Unwatch 1
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
circleci-proje...
hello-world / .circleci /
Go to file
Add file
This branch is 1 commit ahead of main.
Contribute
Mehdi5824 Add .circleci/config.yml
6b308f0 15 minutes ago
History
..
config.yml
Add .circleci/config.yml
15 minutes ago
© 2022 GitHub, Inc.
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About


github.com/Mehdi5824/hello-world/blob/circleci-project-setup/.circleci/config.yml
Search or jump to...
Pull requests
Issues
Marketplace
Explore
Mehdi5824 / hello-world
Public
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
circleci-proje...
hello-world / .circleci / config.yml
Go to file
Mehdi5824 Add .circleci/config.yml
Latest commit 6b308f0 16 minutes ago
History
1 contributor
26 lines (24 sloc)
944 Bytes
Raw
Blame
Use the latest 2.1 version of CircleCI pipeline process engine.
See: https://circleci.com/docs/2.0/configuration-reference
version: 2.1
Define a job to be invoked later in a workflow.
See: https://circleci.com/docs/2.0/configuration-reference/#jobs
jobs:
say-hello:
Specify the execution environment. You can specify an image from Dockerhub or use one of our Convenience Images from CircleCI's Developer Hub.
See: https://circleci.com/docs/2.0/configuration-reference/#docker-machine-macos-windows-executor
docker:
- image: cimg/base:stable
Add steps to the job
See: https://circleci.com/docs/2.0/configuration-reference/#steps
steps:
- checkout
- run:
name: "Say hello"
command: "echo Hello, World!"
Invoke jobs via workflows
See: https://circleci.com/docs/2.0/configuration-reference/#workflows
workflows:
say-hello-workflow:
jobs:
- say-hello

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

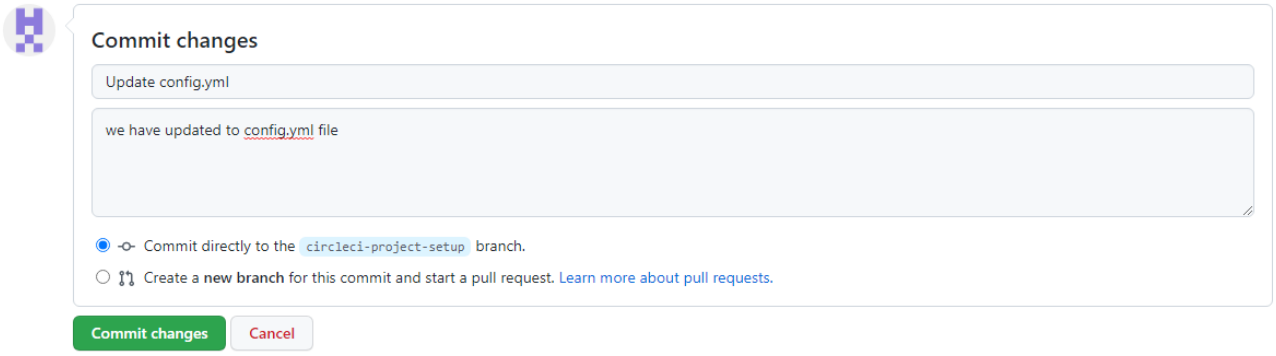
```yaml
version: 2.1
orbs:
  node: circleci/node@4.7.0
jobs:
  build:
    executor:
      name: node/default
      tag: '10.4'
    steps:
      - checkout
      - node/with-cache:
          steps:
            - run: npm install
      - run: npm run test
```

The GitHub file editor should look like this

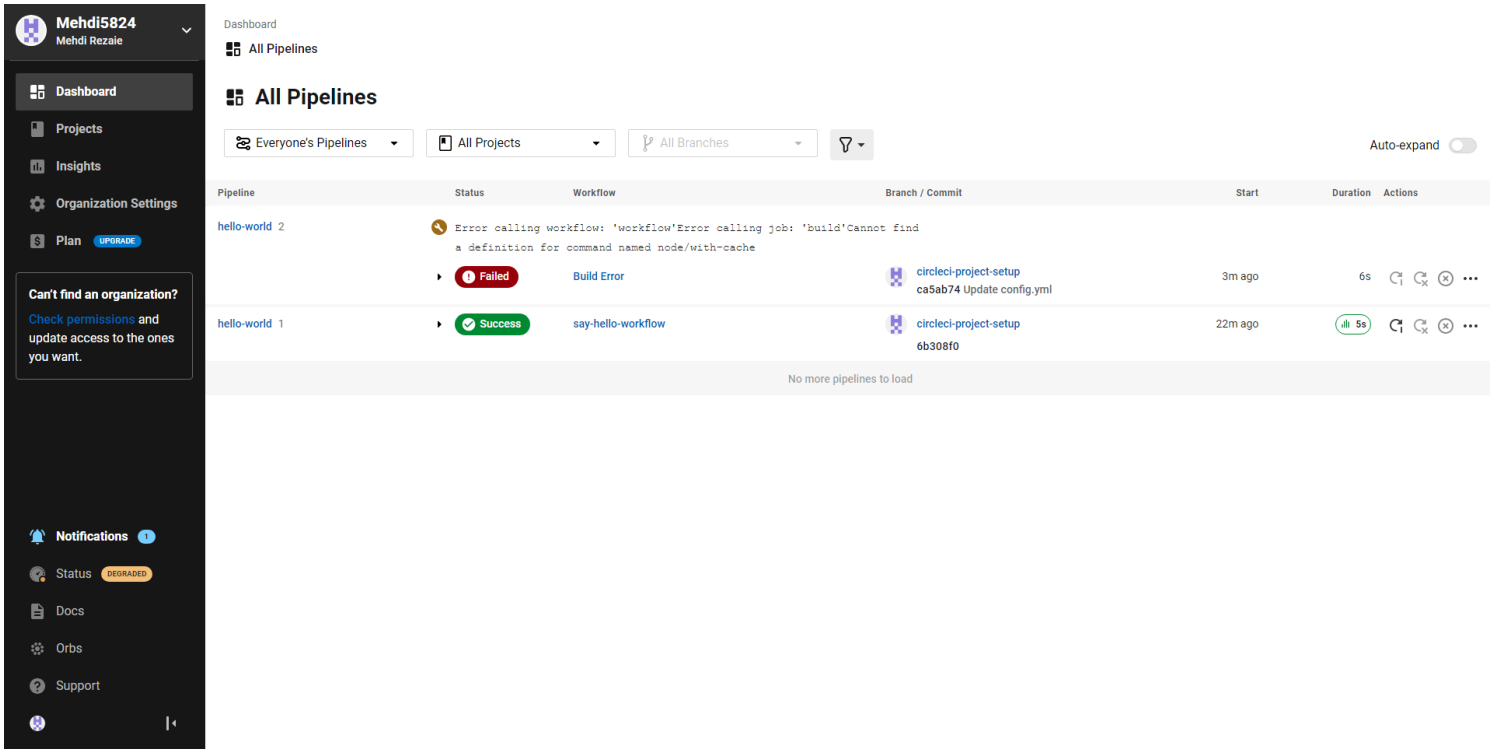


Scroll down and Commit your changes on GitHub

```
      - checkout
      - node/with-cache:
          steps:
            - run: npm install
      - run: npm run test

```



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.



Step 5 – Use Workflows

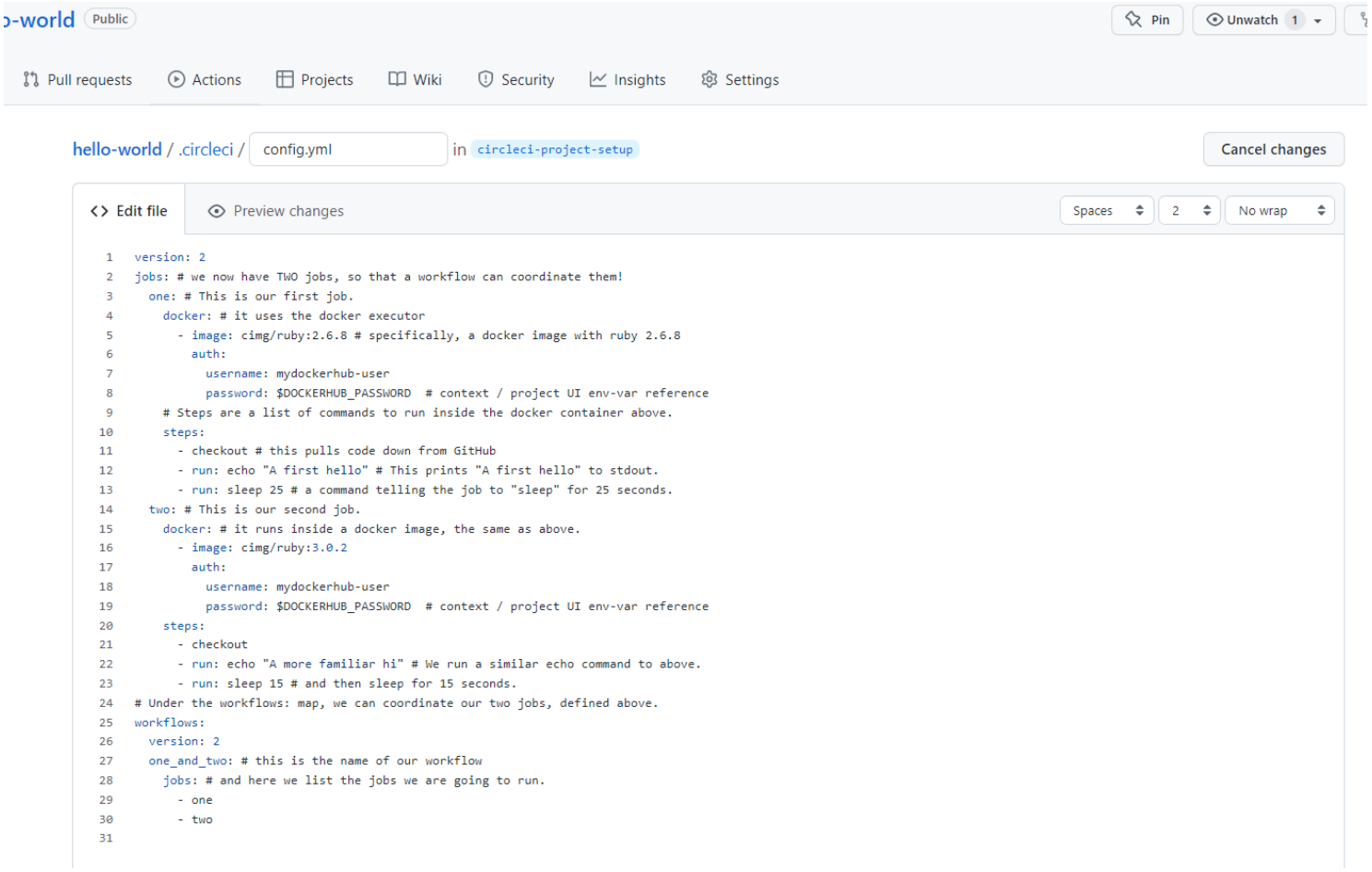
You do not have to use orbs to use CircleCI. The following example details how to create a custom configuration that also uses the workflow feature of CircleCI.

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

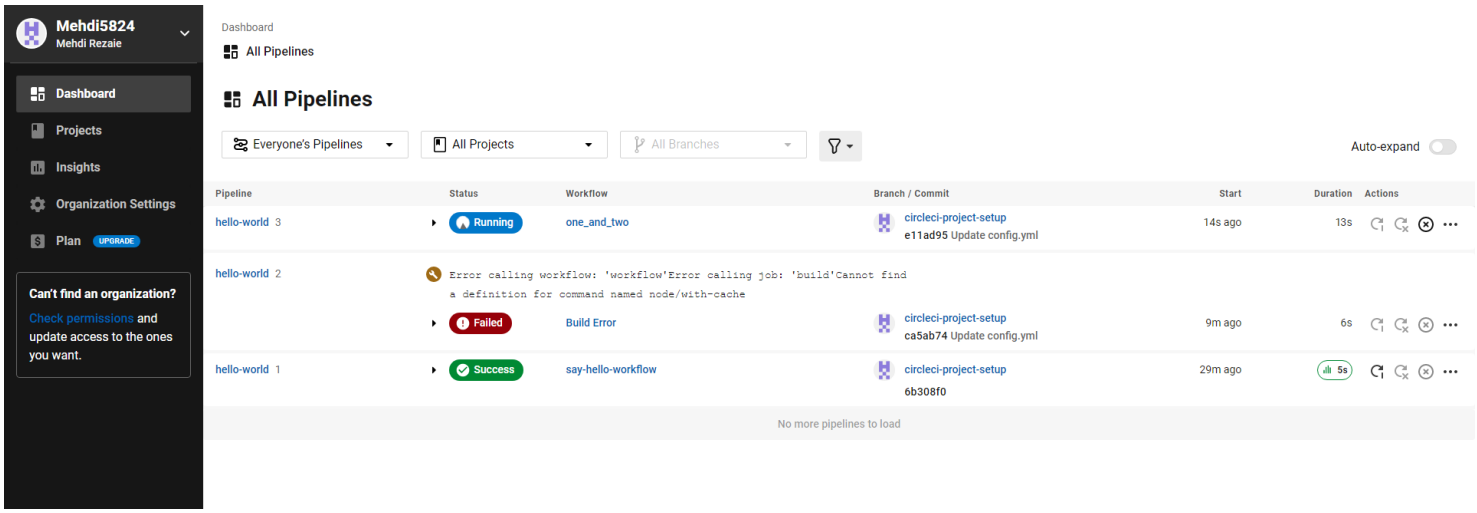
```
version: 2
jobs: # we now have TWO jobs, so that a workflow can coordinate them!
  one: # This is our first job.
    docker: # it uses the docker executor
      - image: cimg/ruby:2.6.8 # specifically, a docker image with ruby 2.6.8
        auth:
          username: mydockerhub-user
          password: $DOCKERHUB_PASSWORD  # context / project UI env-var reference
    # Steps are a list of commands to run inside the docker container above.
    steps:
      - checkout # this pulls code down from GitHub
      - run: echo "A first hello" # This prints "A first hello" to stdout.
      - run: sleep 25 # a command telling the job to "sleep" for 25 seconds.
  two: # This is our second job.
    docker: # it runs inside a docker image, the same as above.
      - image: cimg/ruby:3.0.2
        auth:
          username: mydockerhub-user
          password: $DOCKERHUB_PASSWORD  # context / project UI env-var reference
    steps:
      - checkout
      - run: echo "A more familiar hi" # We run a similar echo command to above.
      - run: sleep 15 # and then sleep for 15 seconds.
# Under the workflows: map, we can coordinate our two jobs, defined above.
workflows:
  version: 2
  one_and_two: # this is the name of our workflow
    jobs: # and here we list the jobs we are going to run.
      - one
      - two
```

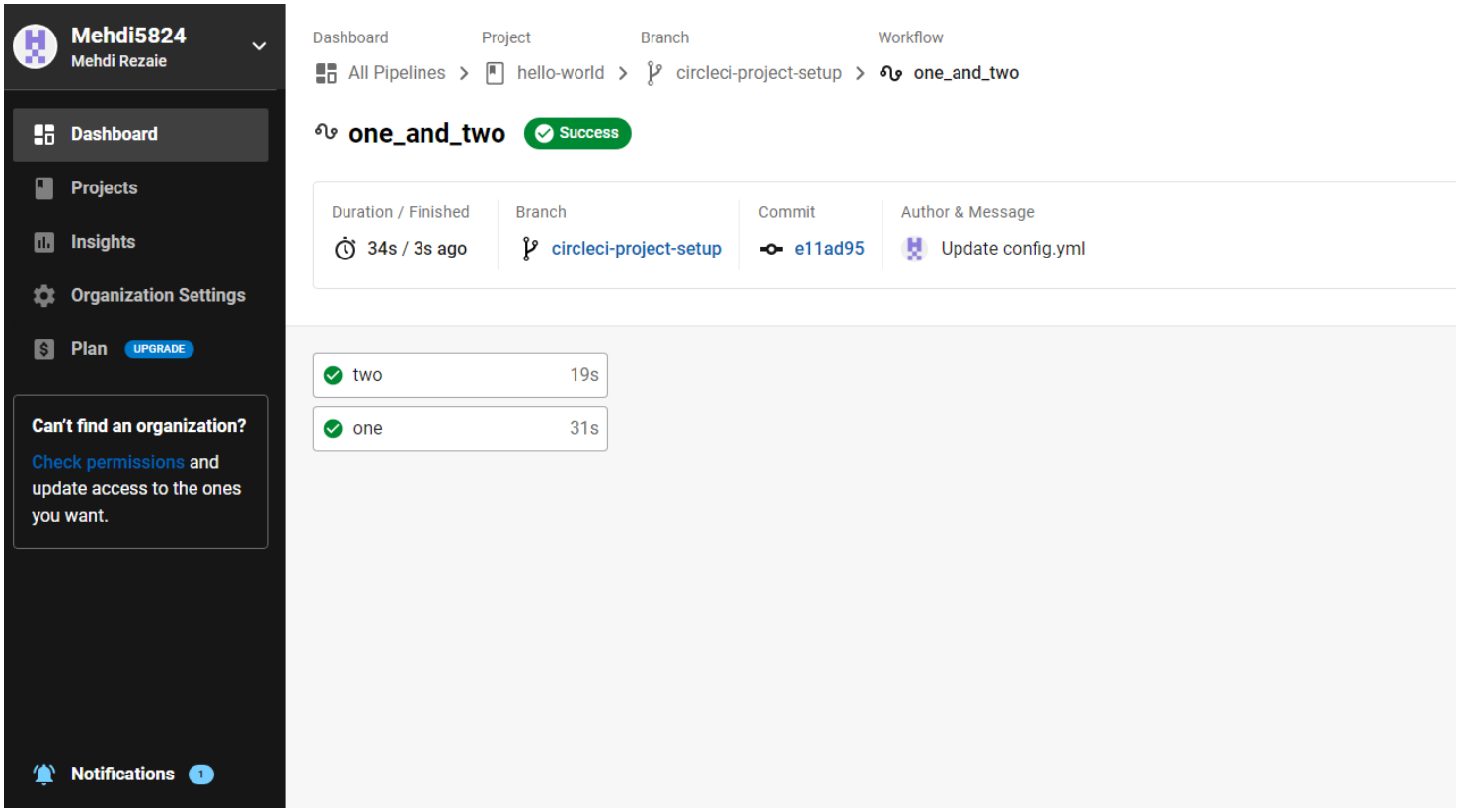
You don't need to write the comments which are the text after #

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



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





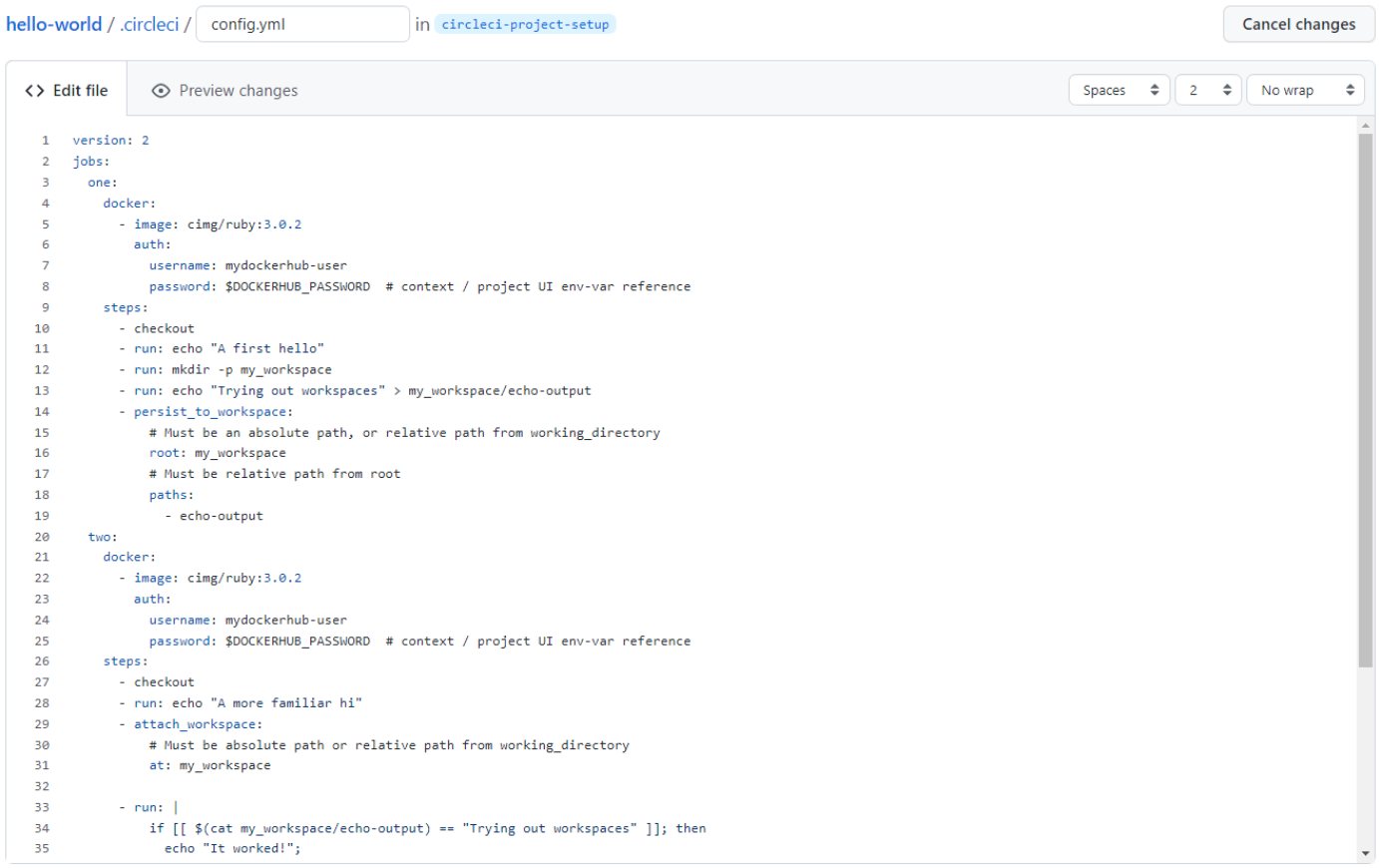
Step 5 – Add some changes to use workspaces

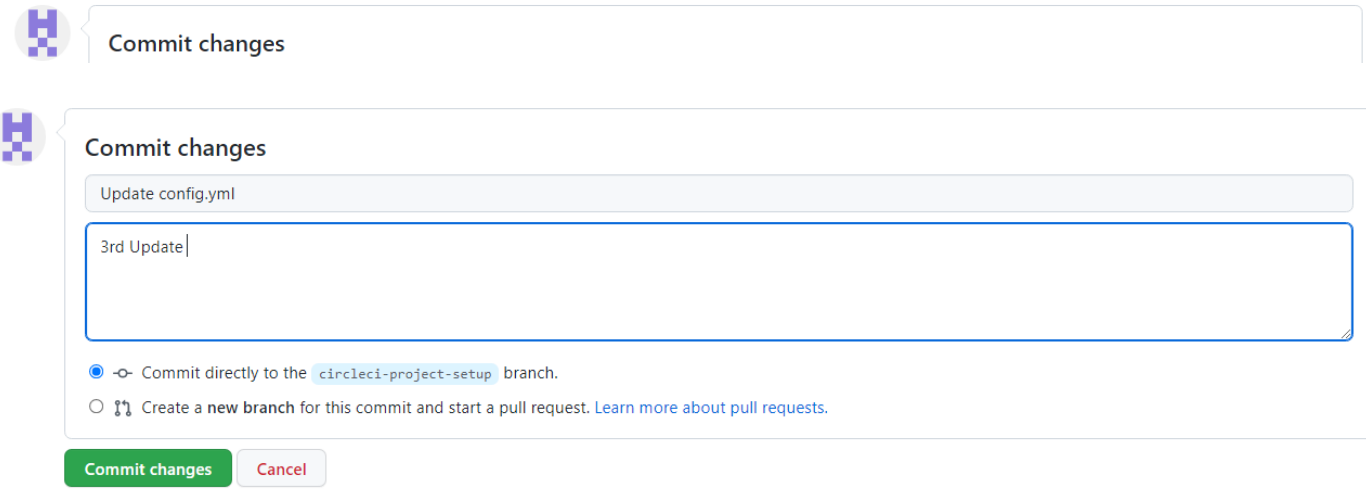
Each workflow has an associated workspace which can be used to transfer files to downstream jobs as the workflow progresses. You can use workspaces to pass along data that is unique to this run and which is needed for downstream jobs. Try updating config.yml to the following:

```
version: 2
jobs:
  one:
    docker:
      - image: cimg/ruby:3.0.2
        auth:
          username: mydockerhub-user
          password: $DOCKERHUB_PASSWORD  # context / project UI env-var reference
    steps:
      - checkout
      - run: echo "A first hello"
      - run: mkdir -p my_workspace
      - run: echo "Trying out workspaces" > my_workspace/echo-output
      - persist_to_workspace:
          # Must be an absolute path, or relative path from working_directory
          root: my_workspace
          # Must be relative path from root
          paths:
            - echo-output
  two:
    docker:
      - image: cimg/ruby:3.0.2
        auth:
          username: mydockerhub-user
          password: $DOCKERHUB_PASSWORD  # context / project UI env-var reference
    steps:
      - checkout
      - run: echo "A more familiar hi"
      - attach_workspace:
          # Must be absolute path or relative path from working_directory
          at: my_workspace
```

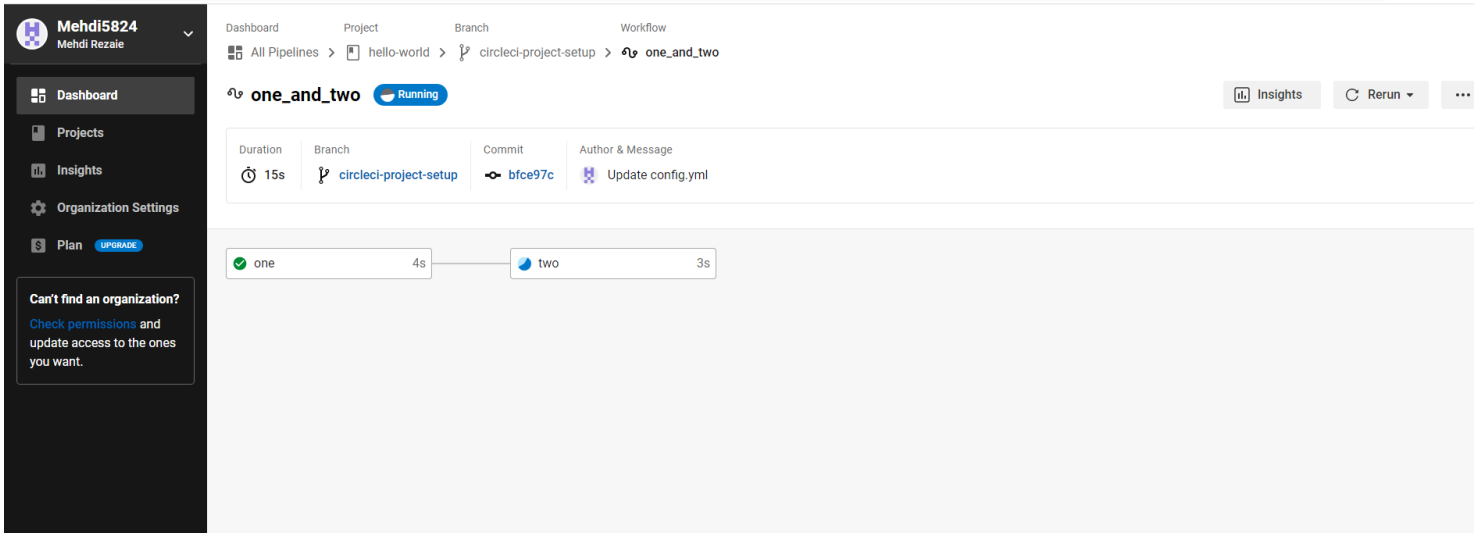
```
32
33        - run: |
34            if [[ $(cat my_workspace/echo-output) == "Trying out workspaces" ]]; then
35              echo "It worked!";
36            else
37              echo "Nope!"; exit 1
38            fi
39  workflows:
40    version: 2
41    one_and_two:
42      jobs:
43        - one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

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





Finally your workflow with the jobs running should look like this



Working with TeamService

(Install .Net core sdk first)

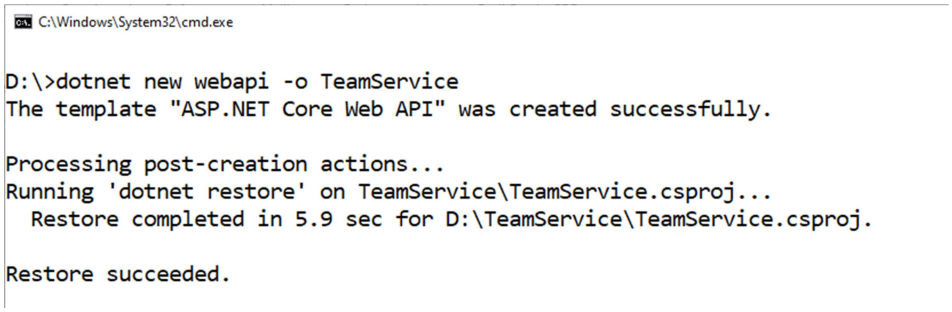
Link: https://dotnet.microsoft.com/learn/dotnet/hello-world-tutorial/install

1) **Create new project:**

Command :

dotnet new webapi -o TeamService

output:



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;
namespace TeamService.Models
{   public class Team
  {
      public string Name { get; set; }
      public Guid ID { get; set; }
      public ICollection<Member> Members { get; set; }
```

```
        public Team()
        {
           this.Members = new List<Member>();
        }
        public Team(string name) : this()
        {
           this.Name = name;
        }
        public Team(string name, Guid id)  : this(name)
        {
           this.ID = id;
        }
        public override string ToString() {
           return this.Name;
        }
     }
  }
```

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);
                }
            }
        }
}
```

4) 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;
            }
        }
}
```

5) create folder "D:\TeamService\Persistence":

6)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);
        }
}
```

7)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;
        }
    }
}
```

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

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

9) Now open two command prompts to run this project

10) On Command prompt 1: (go inside folder teamservice first)

Commands:

```
dotnet run
```

Output:

```
Command Prompt - dotnet run

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

11)On command prompt 2

Command: To get all teams

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

output:

```
Command Prompt

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

Command : To create new team

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

output:

```
Command Prompt

D:\>curl --insecure -H "Content-Type:application/json" -X POST -d "{\"id\":\"e52baa63-d511-417e-9e54-7aab
04286281\",\"name\":\"KC\"}" https://localhost:5001/teams
{"name":"KC","id":"e52baa63-d511-417e-9e54-7aab04286281","members":[]}
D:\>
```

Command : To create one more new team

curl --insecure -H “Content-Type:application/json” –X POST –d “{\“id\”:\”e12baa63-d511-417e-9e54-7aab04286281\”, \”name\”:\”MSC Part1\”}” https://localhost:5001/teams

output:

```
Command Prompt

D:\>curl --insecure -H "Content-Type:application/json" -X POST -d "{\"id\":\"e12baa63-d511-417e-9e54-7aab
04286281\", \"name\":\"MSC Part1\"}"  https://localhost:5001/teams
{"name":"MSC Part1","id":"e12baa63-d511-417e-9e54-7aab04286281","members":[]}
D:\>
```

Command : To get all teams

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

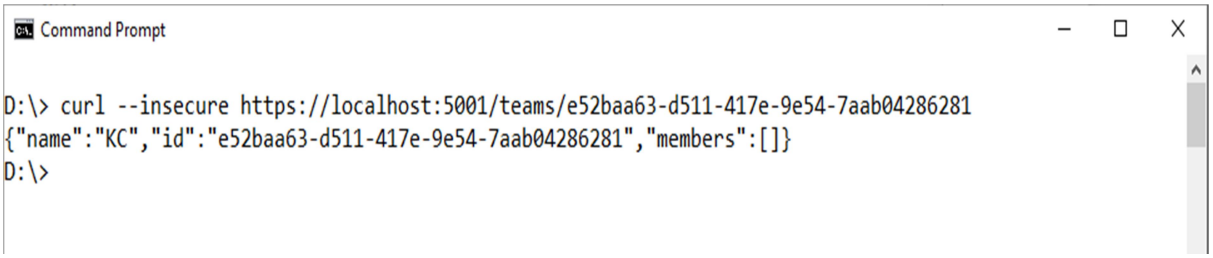
Output:

```
Command Prompt

D:\>curl --insecure https://localhost:5001/teams
[{"name":"KC","id":"e52baa63-d511-417e-9e54-7aab04286281","members":[]},{"name":"MSC Part1","id":"e12baa6
3-d511-417e-9e54-7aab04286281","members":[]}]
D:\>
```

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

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

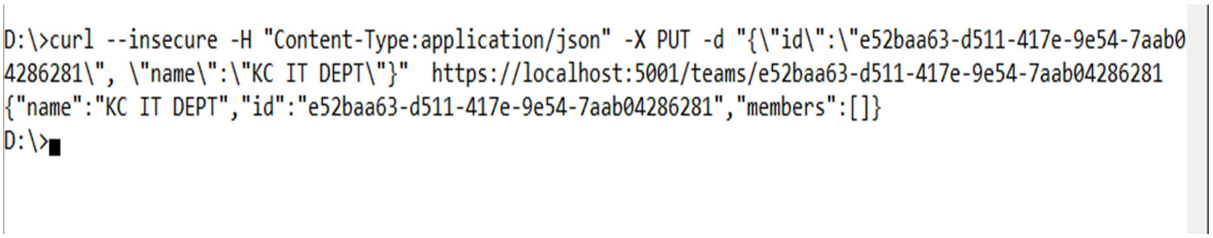
output:



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:



Command: to delete team

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

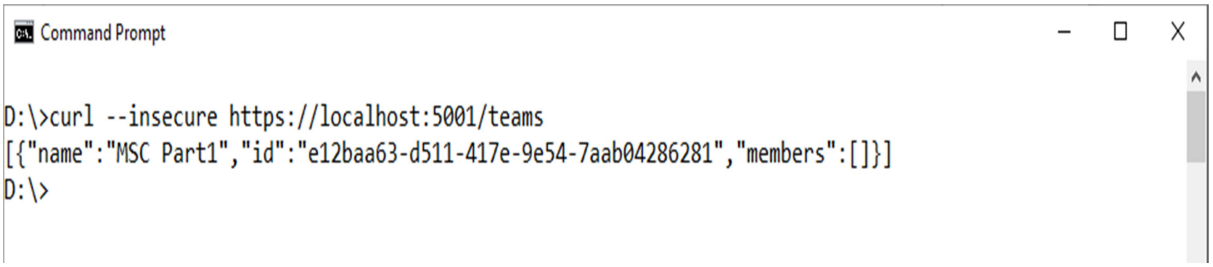
output:

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

Command:

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

Output:

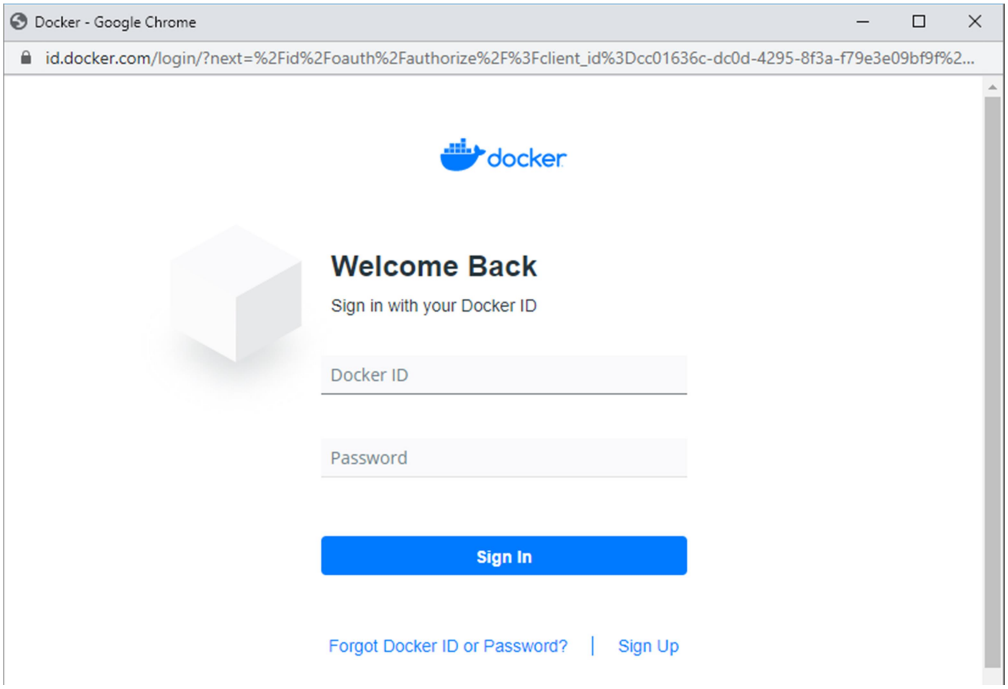


Practical 9 (Backing Services)

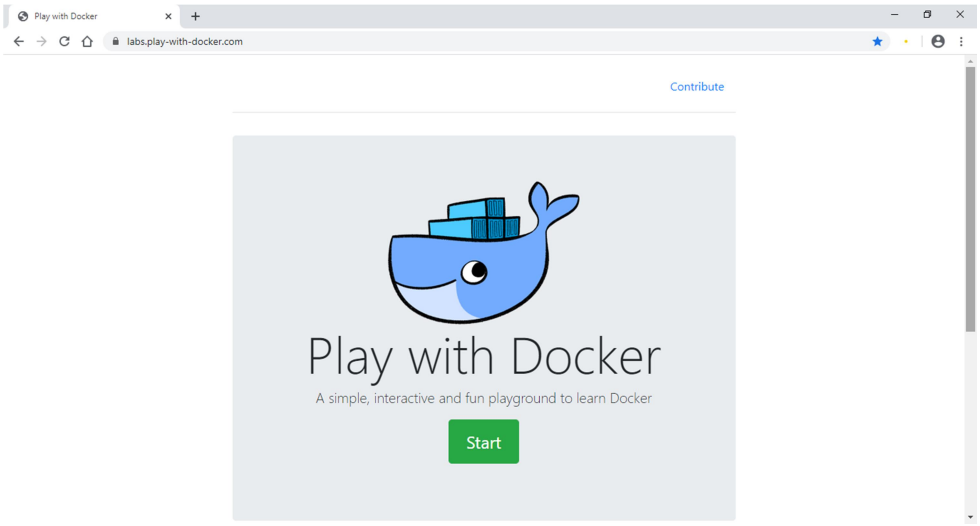
Running Location Service in Docker

(create docker hub login first to use it in play with docker)

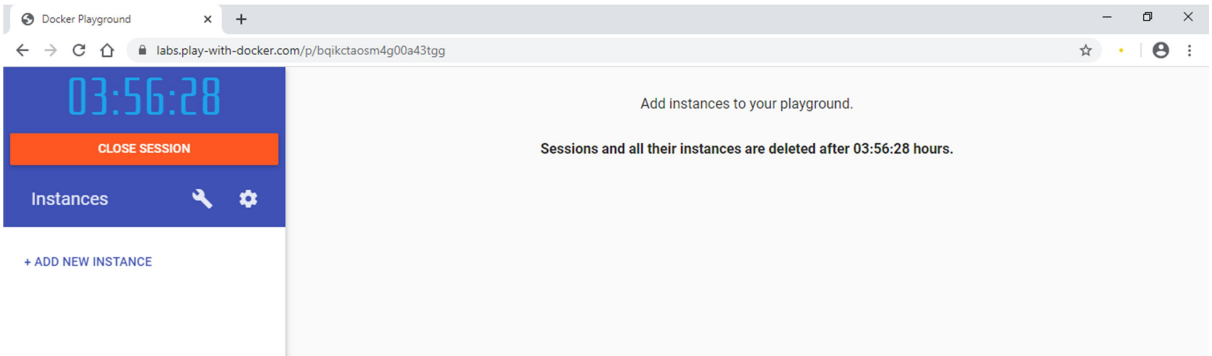
Now login in to Play-With-Docker

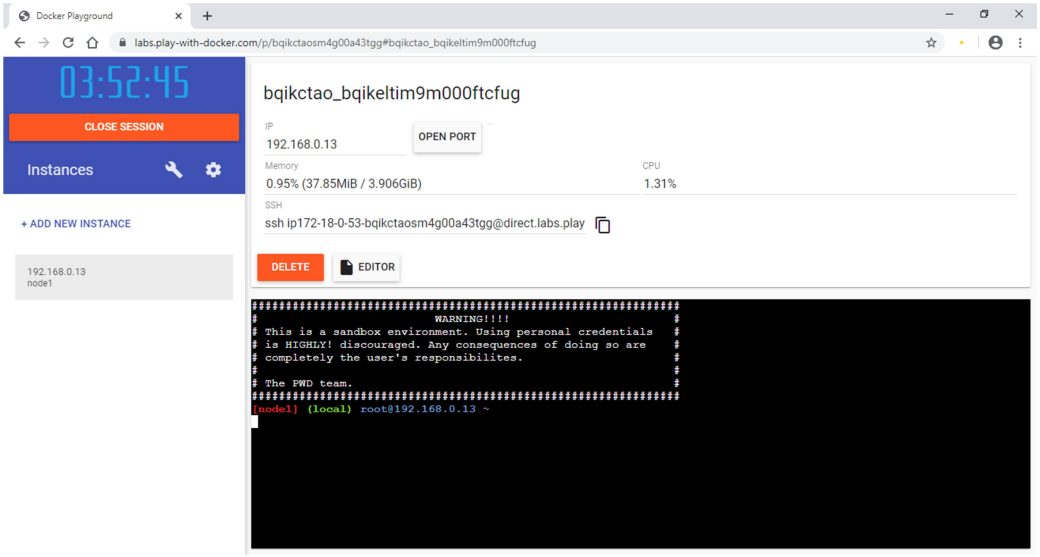


Click on Start



Click on Add New Instance





Start typing following commands

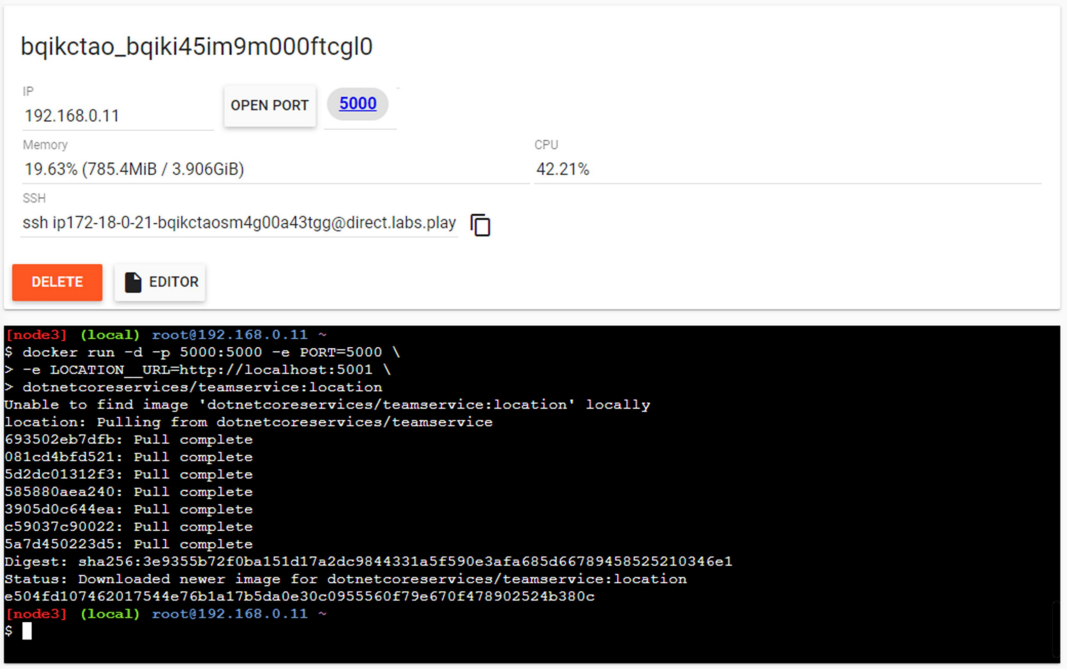
Command : To run teamservice

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)

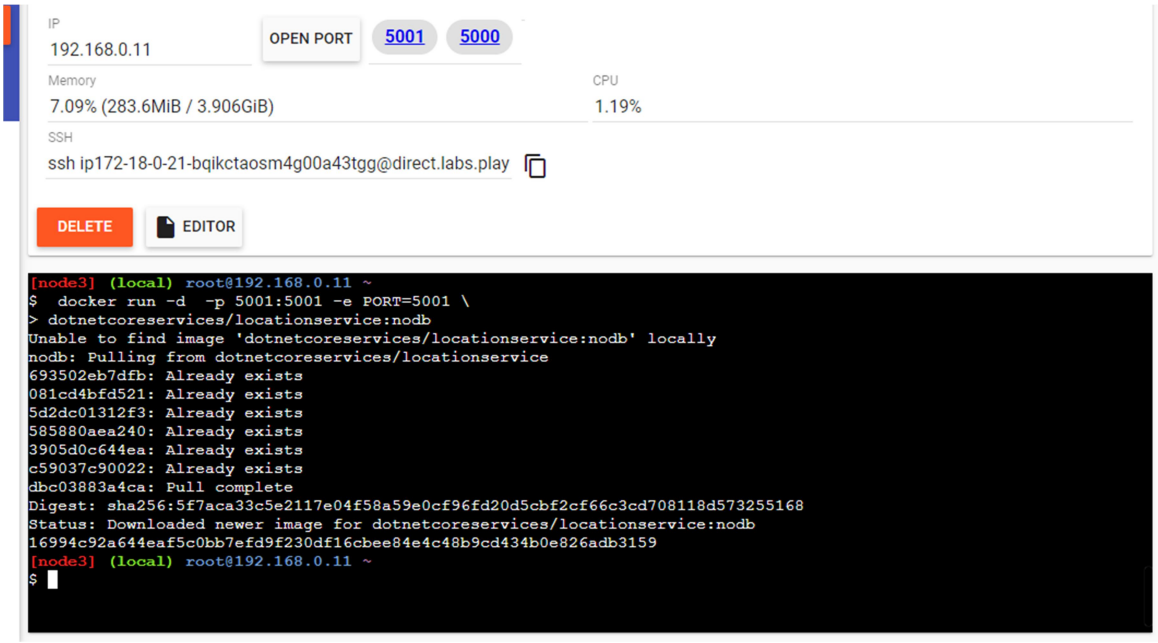


Command: to run location service

docker run -d -p 5001:5001 -e PORT=5001 \

dotnetcoreservices/locationservice:nodb

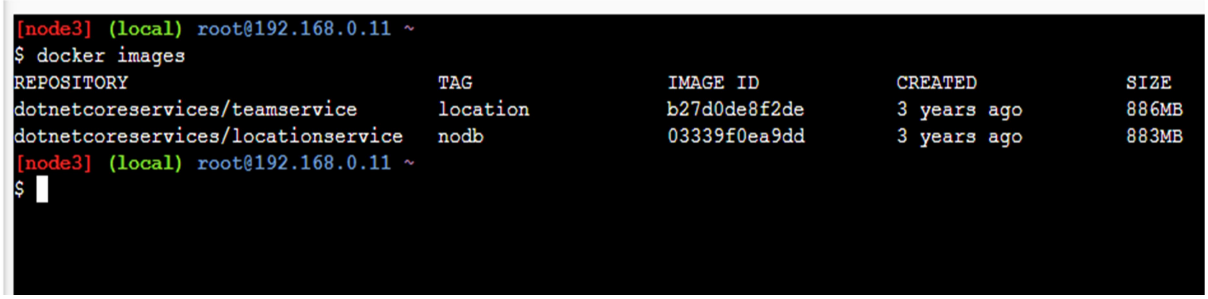
output: (now it has started one more port that is 5001 for location service)



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

```
[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:

Command : To confirm location is added in member

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

output: