Microservice Architecture

# INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.**  **No.** | **Name of Practical** | **Date** | **Signature** |
| 01. | Building APT.NET Core MVC Application. | 29/03/2023 |  |
| 02. | Building ASP.NET Core REST API. | 29/03/2023 |  |
| 03. | Working with Docker, Docker Commands, Docker Images and Containers | 20/04/2023 |  |
| 04. | Installing software packages on Docker, Working with Docker Volumes and Networks | 20/04/2023 |  |
| 05. | Working with Circle CI for continuous integration. | 19/06/2023 |  |
| 06. | Creating Microservice with ASP.NET Core. | 19/06/2023 |  |
| 07. | Creating Backing Service with ASP.NET Core. | 20/06/2023 |  |

**Practical No.: 01**

## Building APT.NET Core MVC Application.

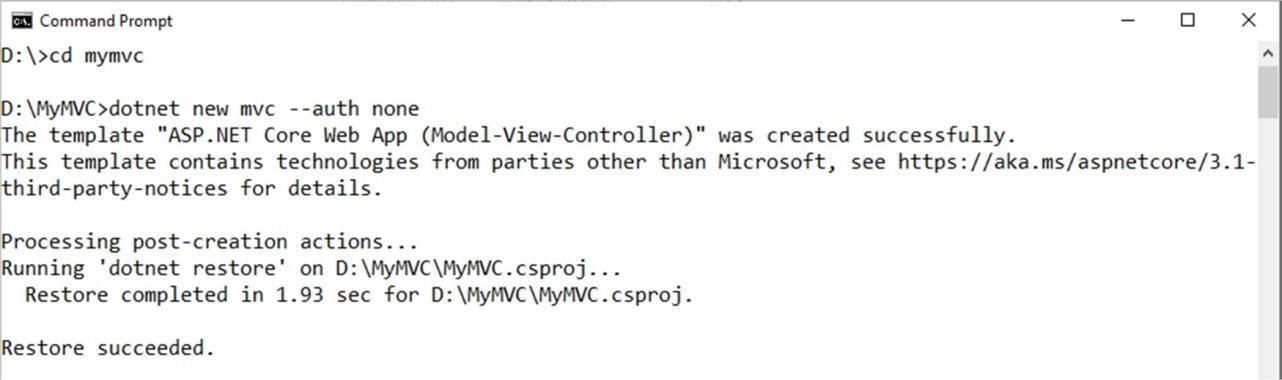
##### Install .Net Core Sdk (Link: https://dotnet.microsoft.com/learn/dotnet/hello-world-tutorial/install)

1. **create folder MyMVC folder in D: drive or any other drive**

##### open command prompt and perform following operationsCommand: to create mvc project

dotnet new mvc --auth none

##### output:



1. **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"; }

}

}

##### Run the project



1. **Now open browser and and type URL: localhost:5000**



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

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

using System;

namespace MyMVC.Models

{

public class StockQuote

{ public string Symbol {get;set;} public int Price{get;set;}

}

}

##### 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>

##### 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()

{

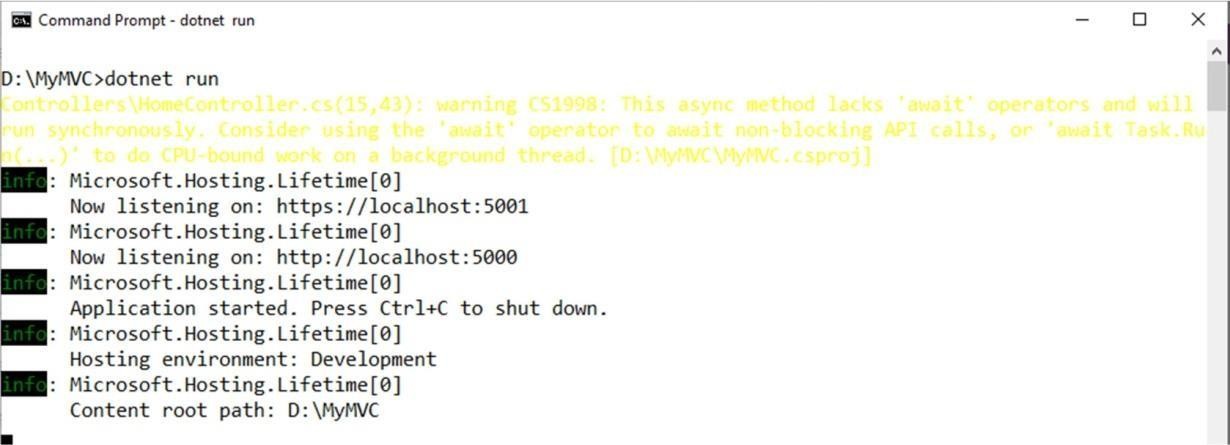
var model= new StockQuote{ Symbol='HLLO', Price=3200}; return View(model);

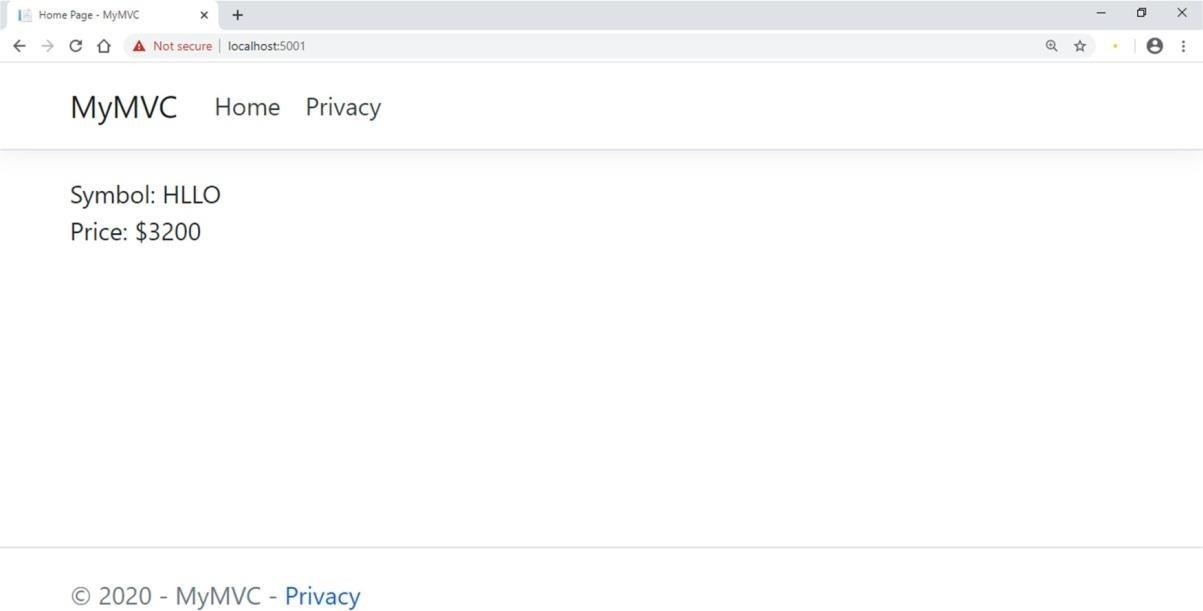
}

}

}

##### Now run the project using

dotnet run

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

# Practical No.: 02

## Building ASP.Net core REST API

#### Software requirement:

##### 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

##### Check everything installed correctly

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

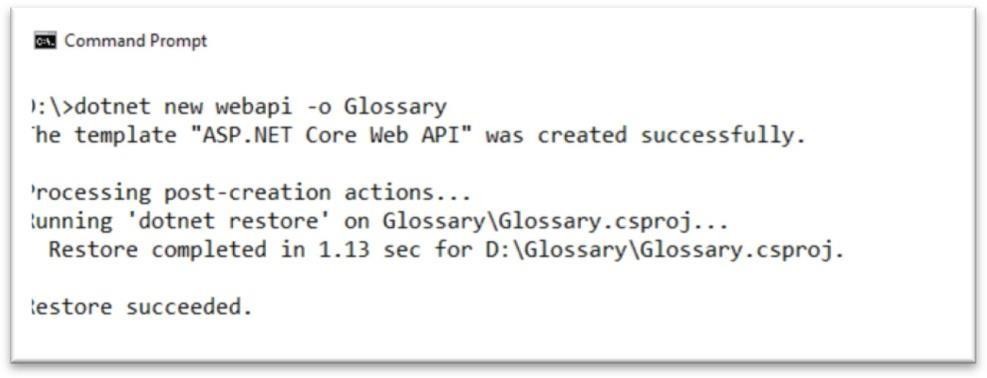
> dotnet

#### Create your web API

##### Open two command prompts

Command prompt 1: Command:

dotnet new webapi -o Glossary

output:

Command:

cd Glossary dotnet run

Output:

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

Command:

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

output:

##### Now Change the content:

**To get started, remove the WeatherForecast.cs file from the root of the project and theWeatherForecastController.cs file from the Controllers folder.**

##### Add Following two files

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

//GlossaryItem.cs namespace Glossary

{

public class GlossaryItem

{

public string Term { get; set; } public string Definition { get; set;

}

}

}

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

//Controllers/GlossaryController.cs using System;

using System.Collections.Generic; using Microsoft.AspNetCore.Mvc; using System.IO;

namespace Glossary.Controllers

{

[ApiController] [Route("api/[controller]")]

public class GlossaryController: ControllerBase

{

private static List<GlossaryItem> Glossary = new List<GlossaryItem> {new GlossaryItem

{

Term= "HTML",

Definition = "Hypertext Markup Language"

},

new GlossaryItem

{

Term= "MVC",

Definition = "Model View Controller"

},

new GlossaryItem

{

Term= "OpenID",

Definition = "An open standard for authentication"

}

};

[HttpGet]

public ActionResult<List<GlossaryItem>> Get()

{ return Ok(Glossary);

}

[HttpGet] [Route("{term}")

]

public ActionResult<GlossaryItem> Get(string term)

{

var glossaryItem = Glossary.Find(item =>

item.Term.Equals(term, StringComparison.InvariantCultureIgnoreCase));

if (glossaryItem == null)

{ return NotFound();

} else

{

return Ok(glossaryItem);

}

}

[HttpPost]

public ActionResult Post(GlossaryItem glossaryItem)

{

var existingGlossaryItem = Glossary.Find(item =>

item.Term.Equals(glossaryItem.Term, StringComparison.InvariantCultureIgnoreCase));

if (existingGlossaryItem != null)

{

return Conflict("Cannot create the term because it already exists.");

}

else

{

Glossary.Add(glossaryItem);

var resourceUrl = Path.Combine(Request.Path.ToString(), Uri.EscapeUriString(glossaryItem.Term));return Created(resourceUrl, glossaryItem);

}

}

[HttpPut]

public ActionResult Put(GlossaryItem glossaryItem)

{

var existingGlossaryItem = Glossary.Find(item => item.Term.Equals(glossaryItem.Term, StringComparison.InvariantCultureIgnoreCase));

if (existingGlossaryItem == null)

{

return BadRequest("Cannot update a nont existing term.");

} else

{

existingGlossaryItem.Definition = glossaryItem.Definition;return Ok();

}

}

[HttpDelete] [Route("{term}")

]

public ActionResult Delete(string term)

{

var glossaryItem = Glossary.Find(item =>

item.Term.Equals(term, StringComparison.InvariantCultureIgnoreCase));

if (glossaryItem == null)

{ return NotFound();

}

else

{

Glossary.Remove(glossaryItem)

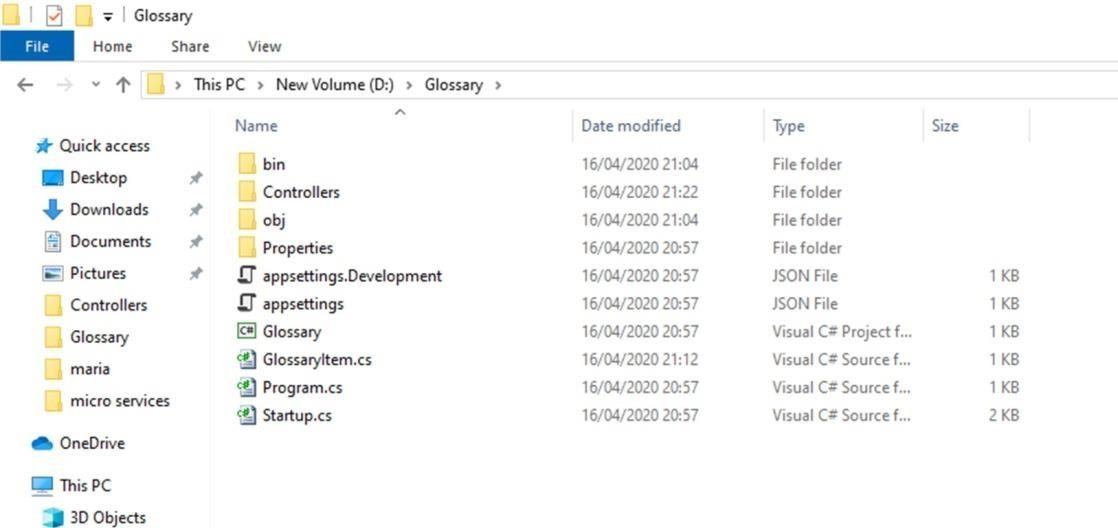
;return NoContent();

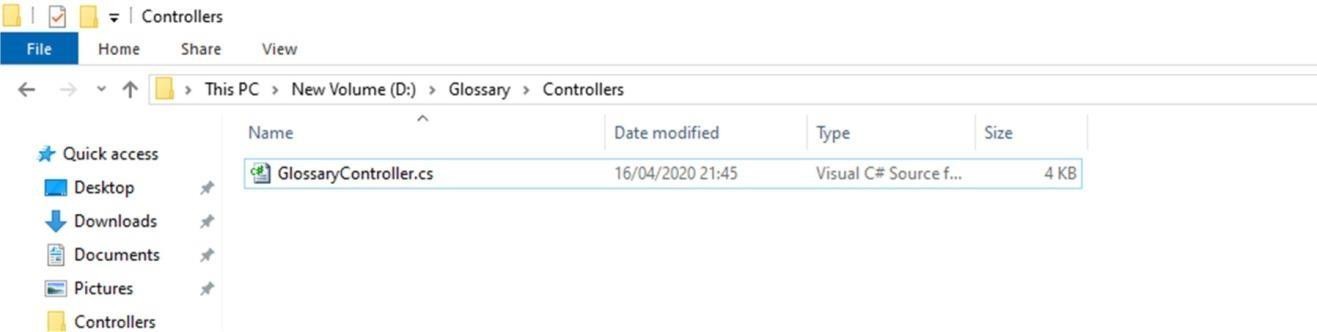
}

}

}

}

Output:



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

Command:

dotnet run

output:

##### On Command prompt2:

1. **Getting a list of items:**

Command:

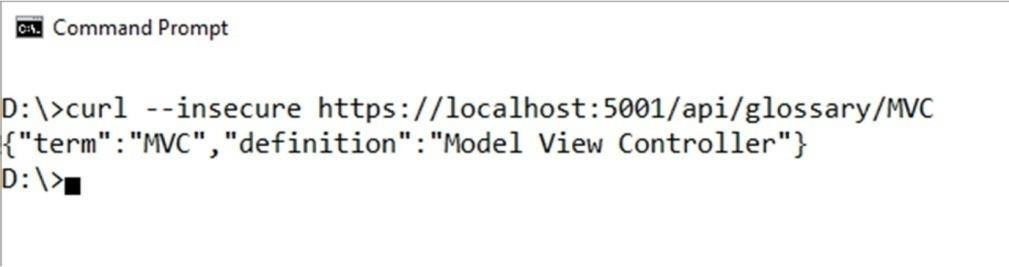
curl --insecure https://localhost:5001/api/glossary

Output:

##### 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:

##### 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:



##### Delete Item

Command:

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

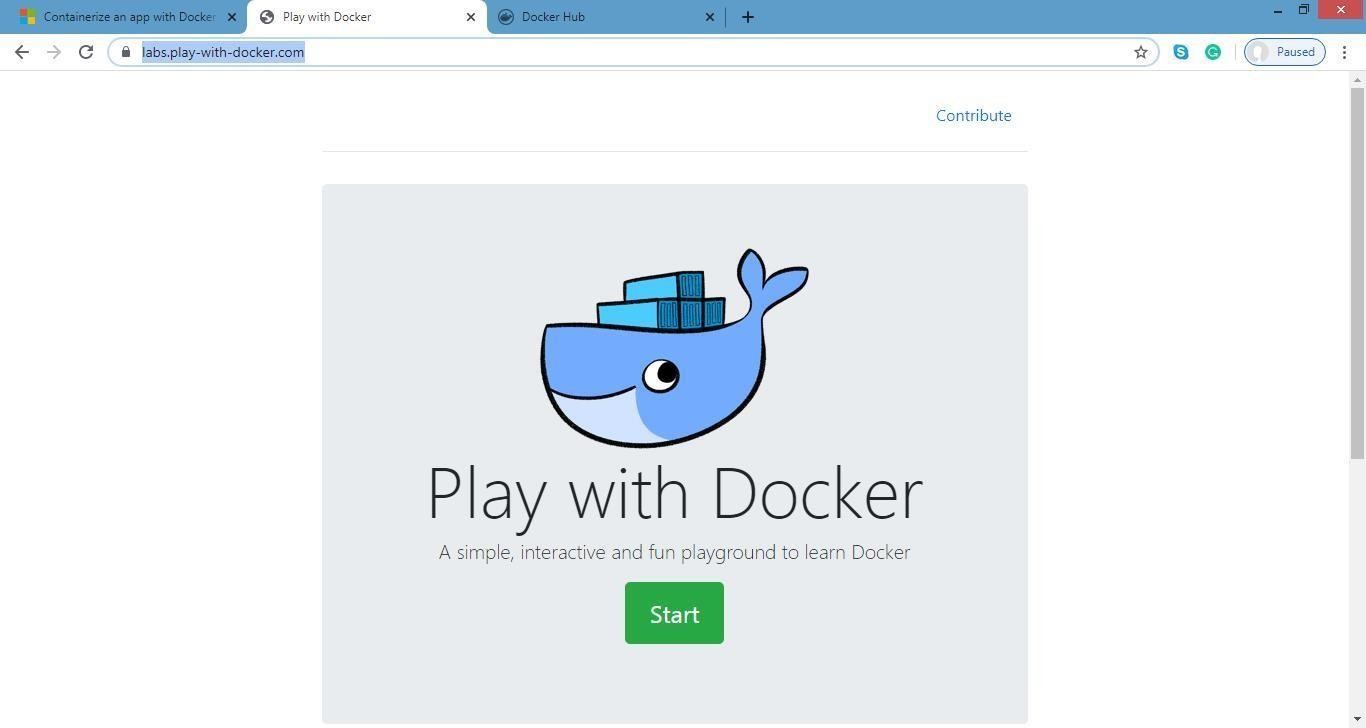
Output:



# Practical No.: 03

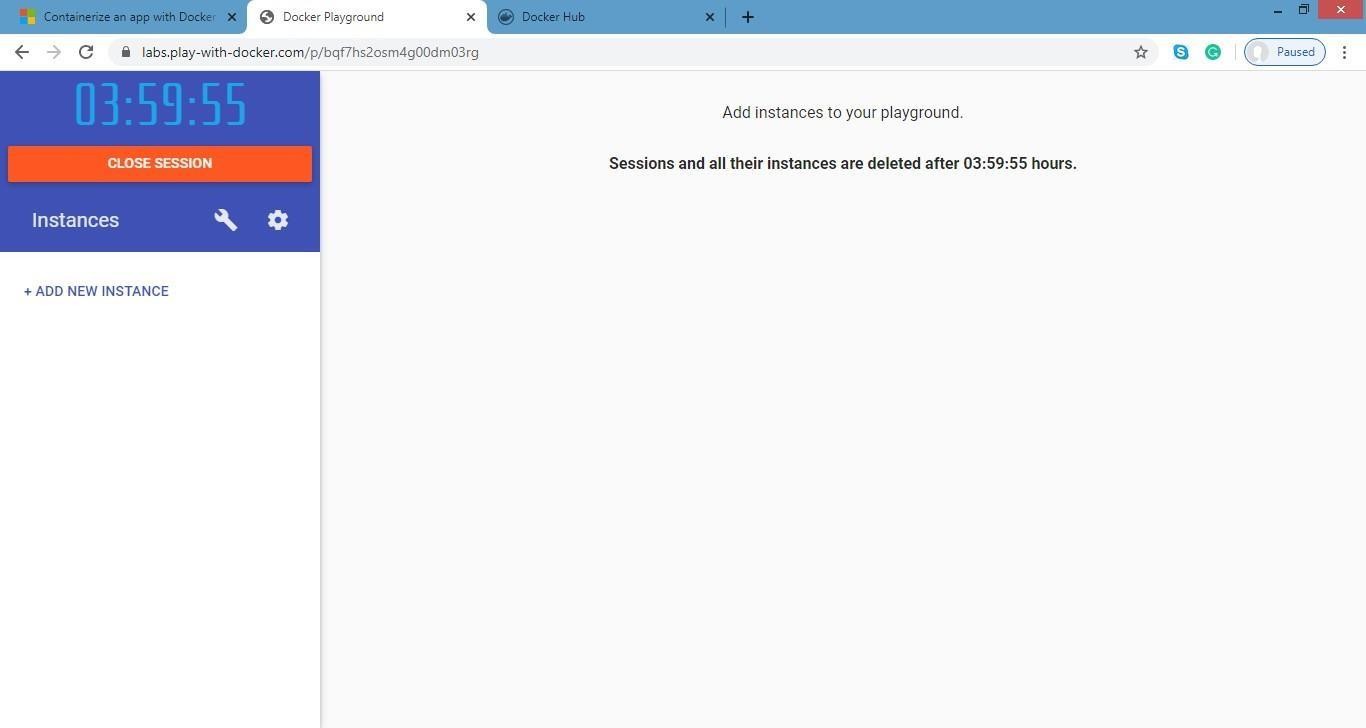
## Working with Docker

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



Click on start

1. add new instance

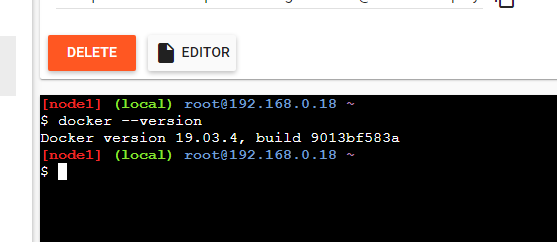


1. perform following:

**Method1:**

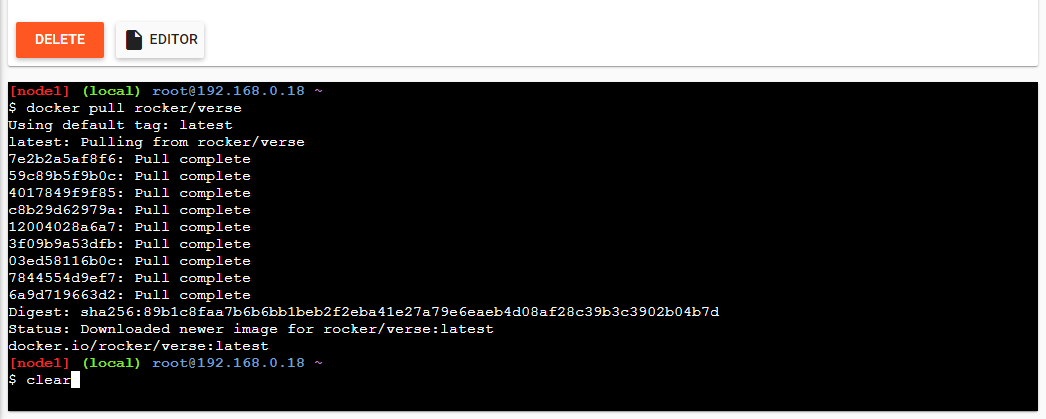
**To pull and push images using docker**

Command: to check docker version docker –version

output:

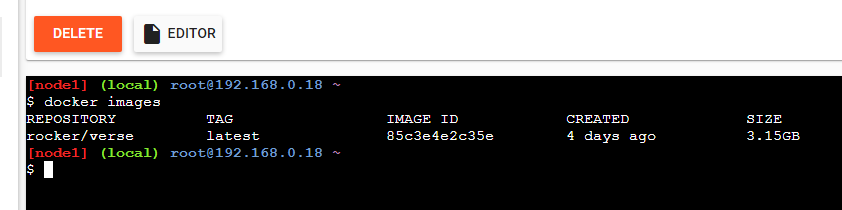
Command: to pull readymade image docker pull rocker/verse

output:

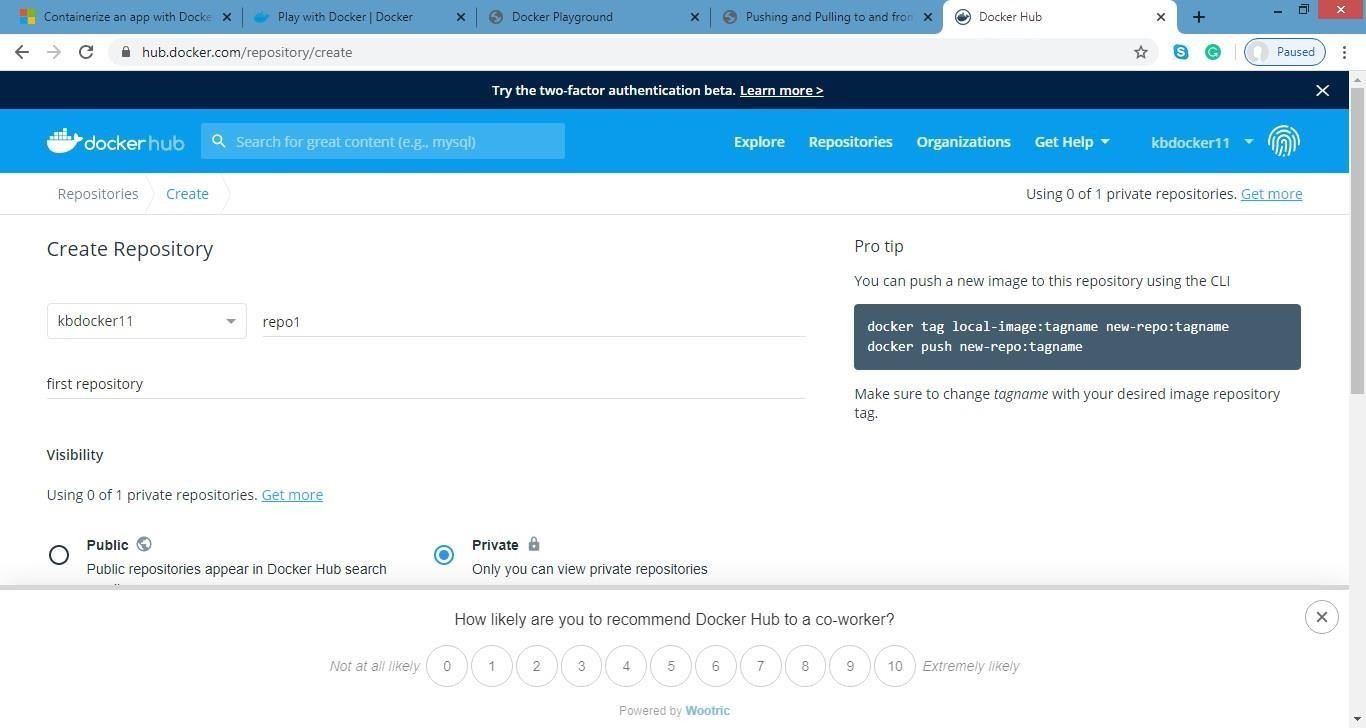


Command: to check images in docker docker images

output:

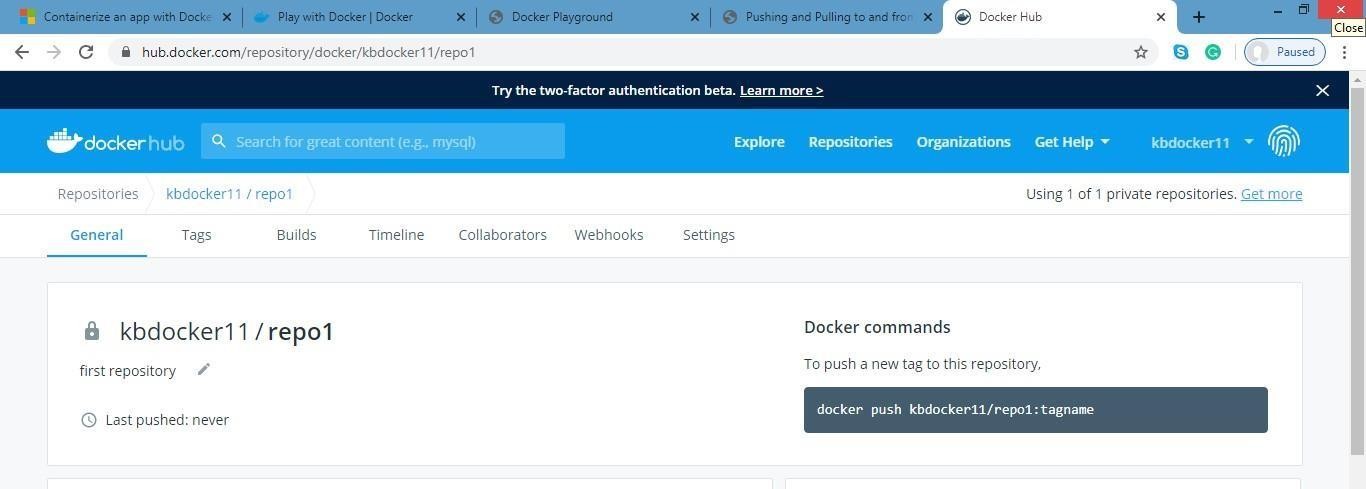


Now Login to docker hub and create repositoryOutput:



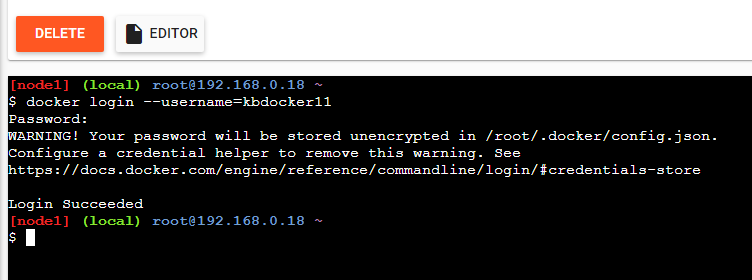
Click on Create button

Now check repository created



Command: to login to your docker account docker login –username=kbdocker11password:

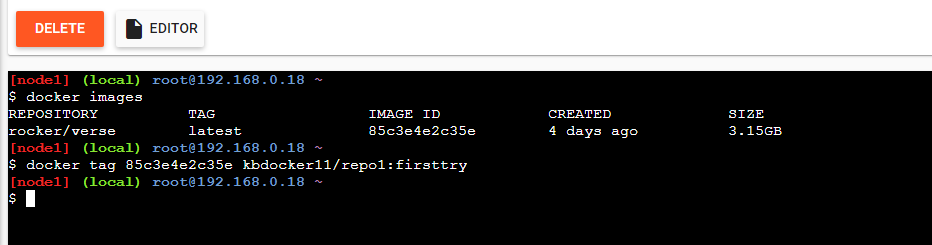
note: kbdocker11 is my docker ID . You will use your docker ID here. And enter yourpassword . Output:



Command : to tag image

docker tag 8c3e4e2c3e kbdocker11/repo1:firsttry

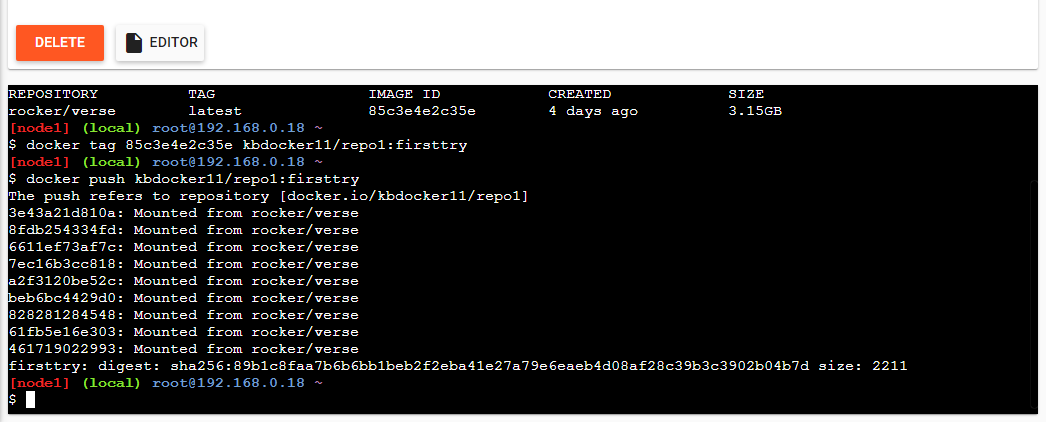
note: here 8c3e4e2c3e this is image id which you can get from docker imagescommand. Output:



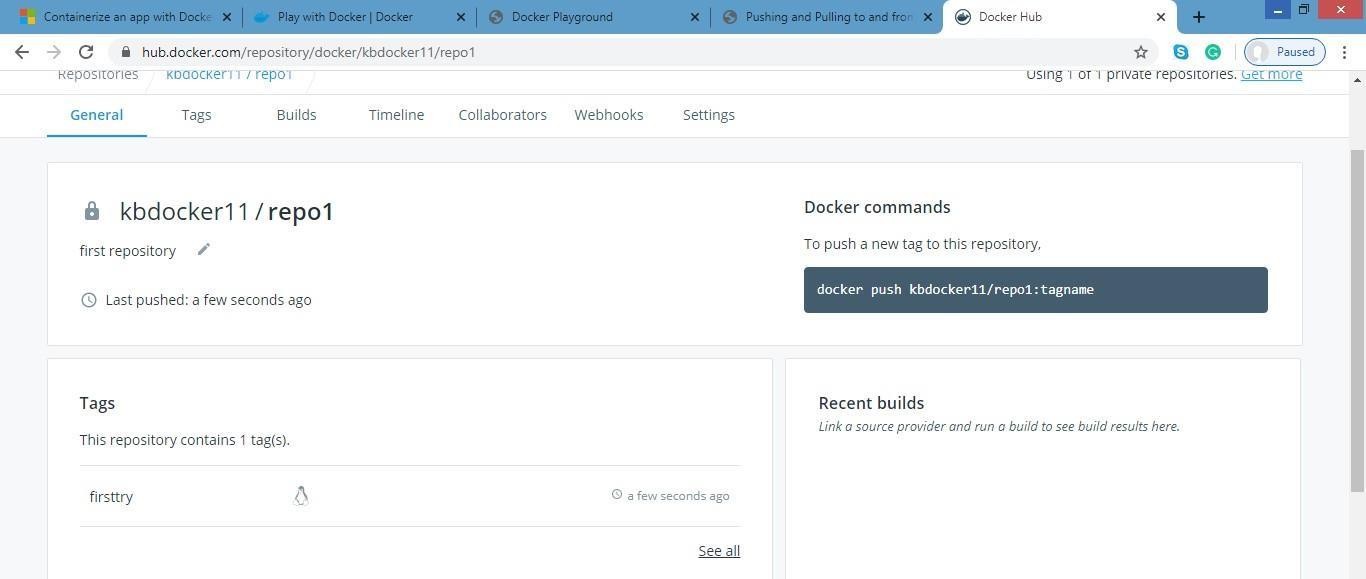
Command: to push image to docker hub accountdocker push kbdocker11/repo1:firsttry

note: firsttry is tag name created above.

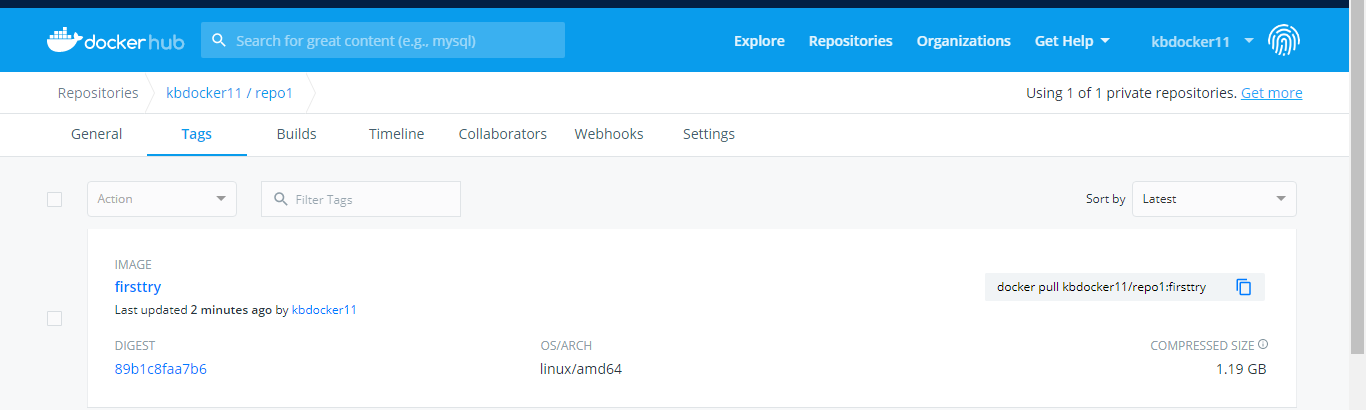
Output



Check it in docker hub now



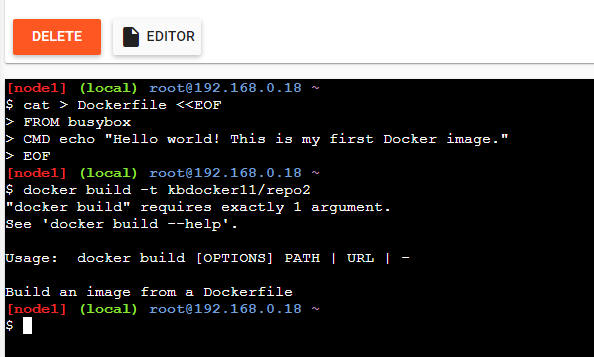
Click on tags and check



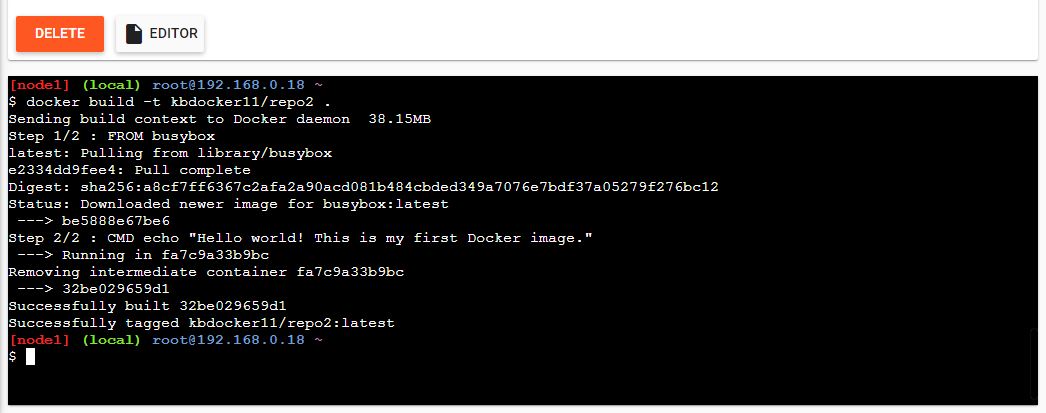
#### Method 2:

**Build an image then push it to docker and run it**

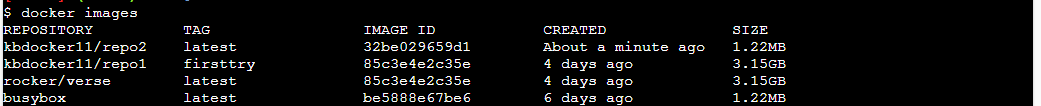
Command : to create docker file

* 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 .

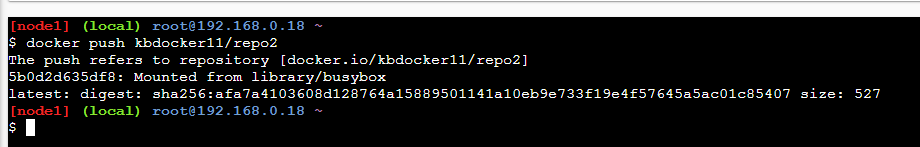
Output:

Command: to check docker images docker images

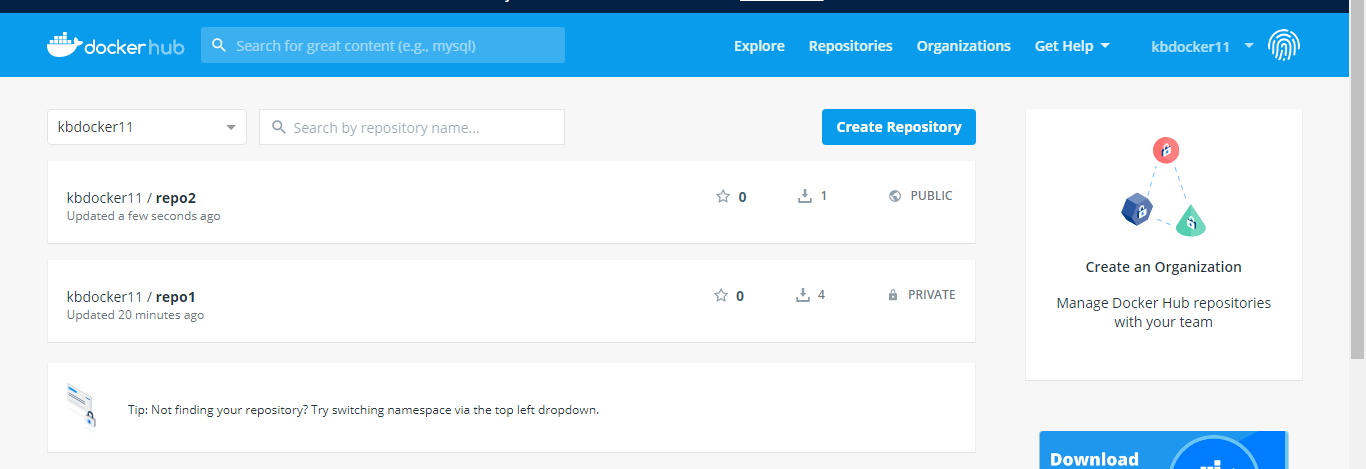
output:

Command: to push image to docker hub docker push kbdocker11/repo2 .

Output:

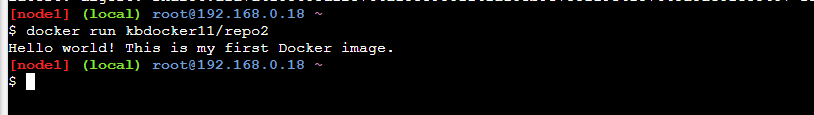


Now check it on docker hub



command: to run docker image:

docker run kbdocker11/repo2 output:

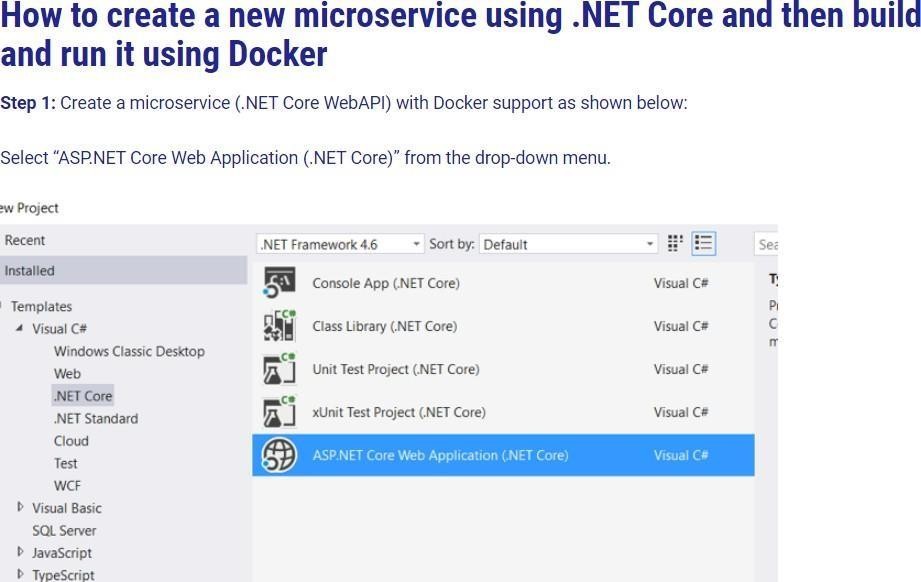
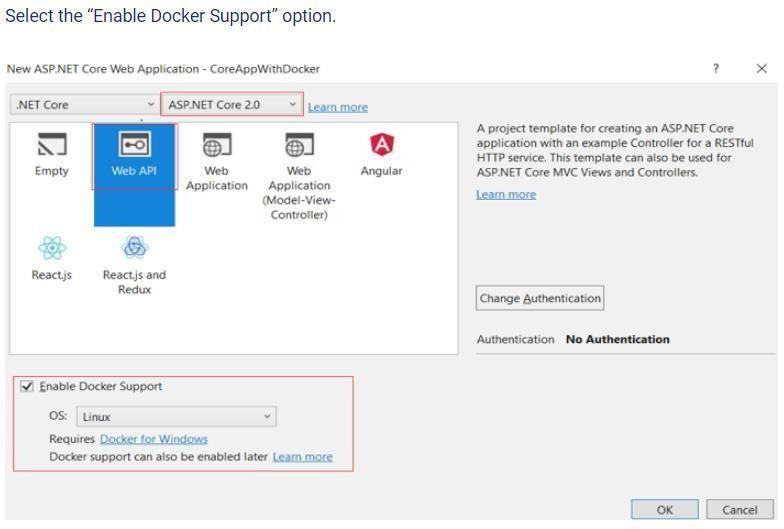


Now close session.

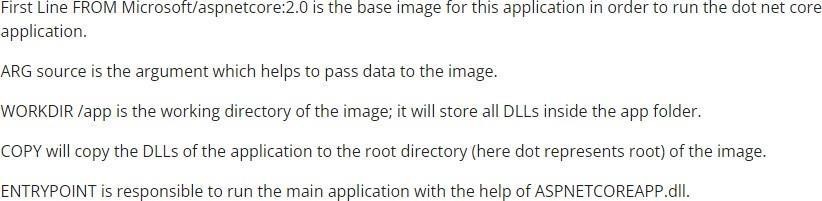
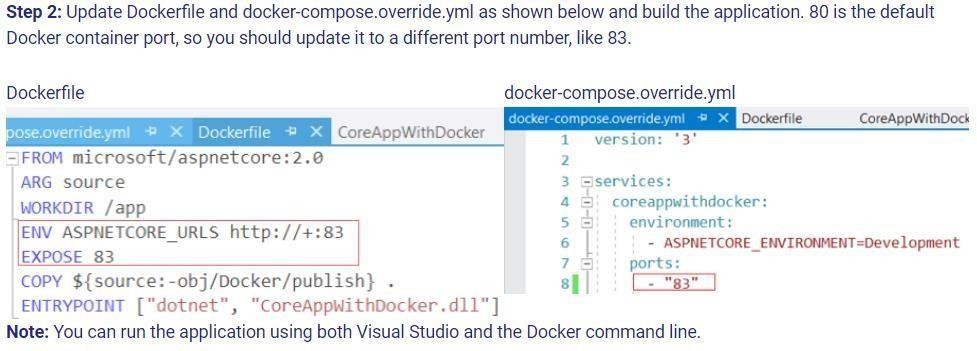
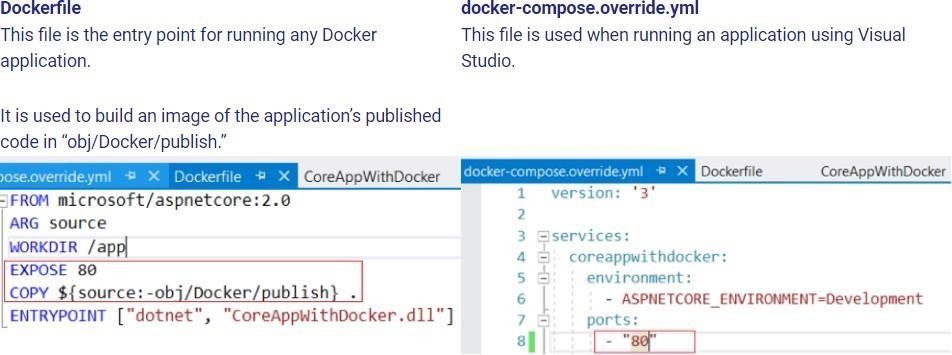
# PRACTICAL No.: 04

## Installing software packages on Docker, Working with Docker Volumes and Networks

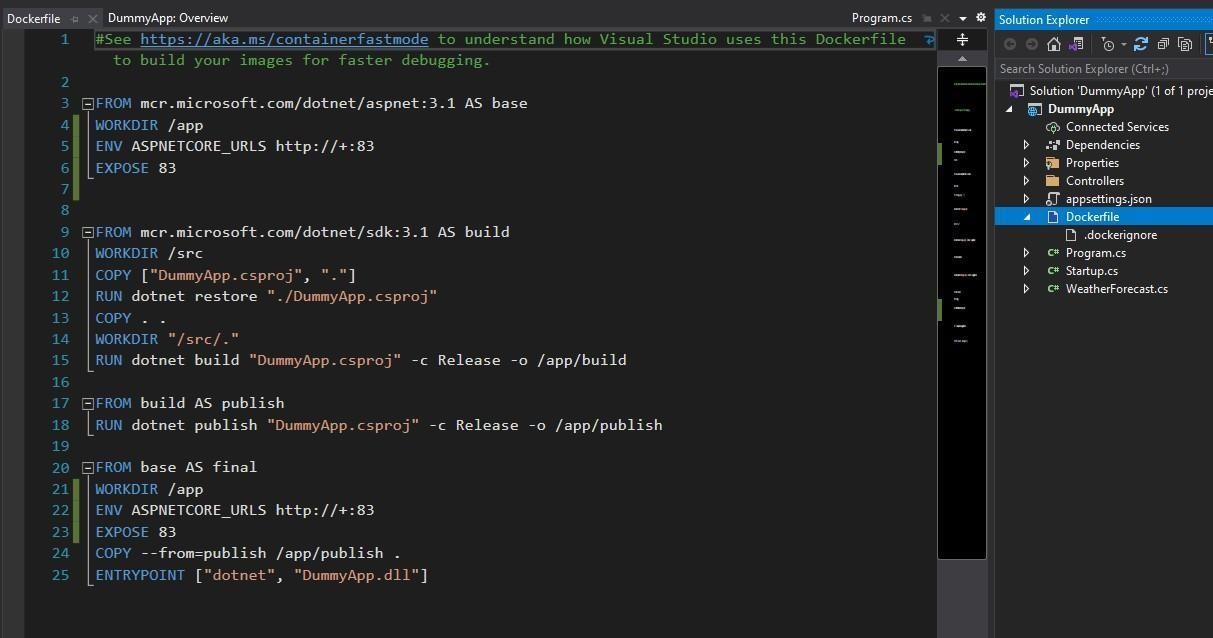


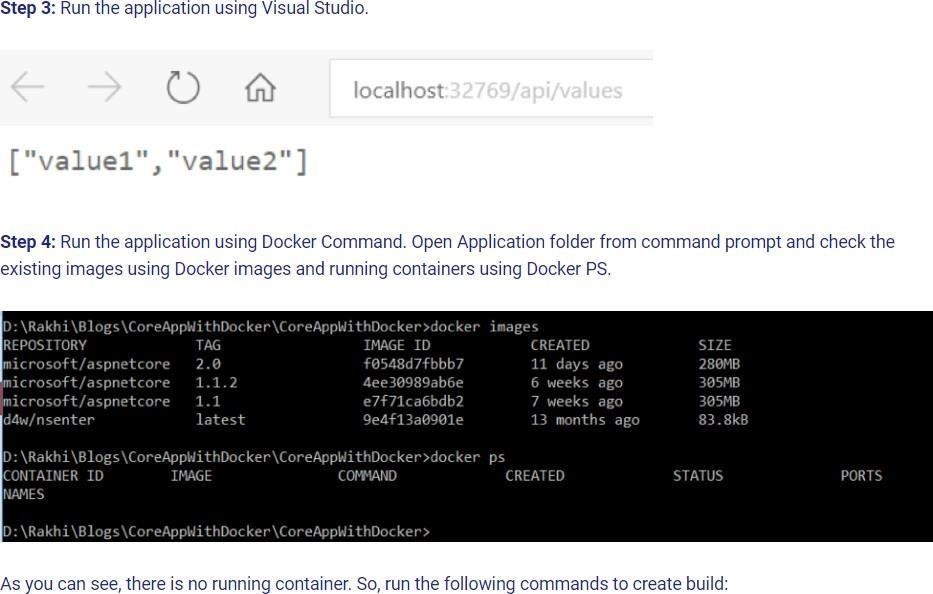




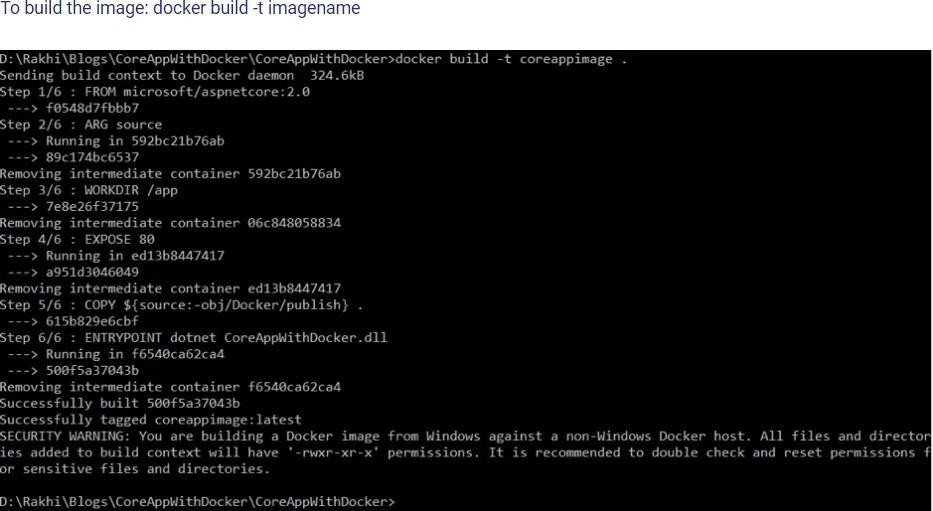


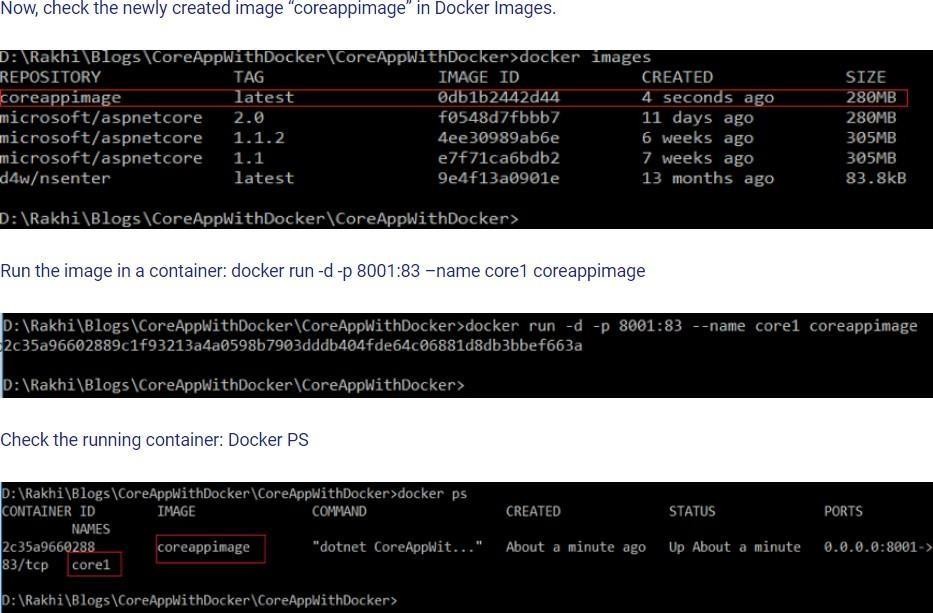
**In the case of .NET Core 3.1 the Docker file will look like this**



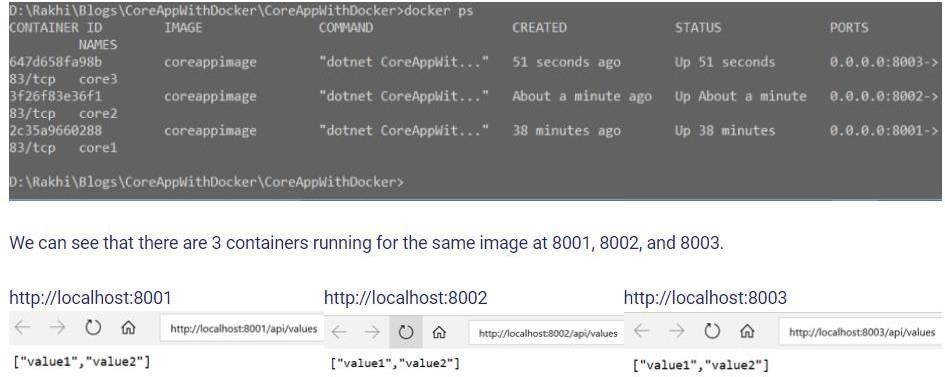


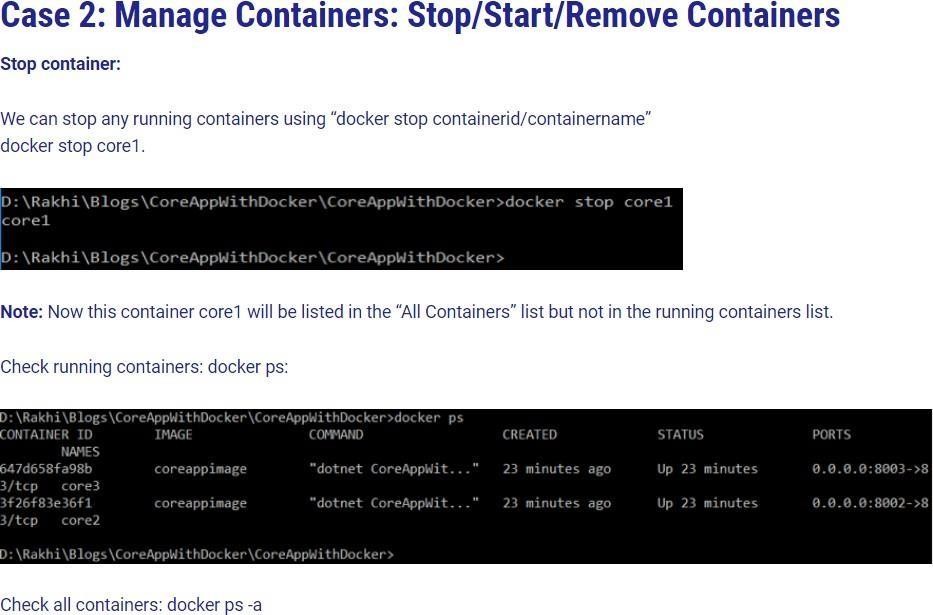


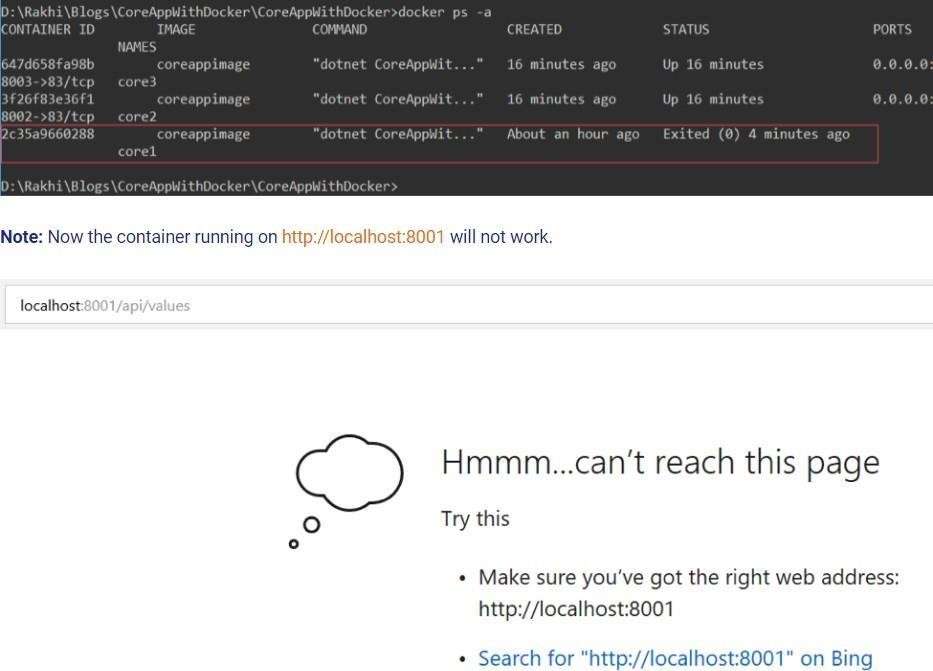


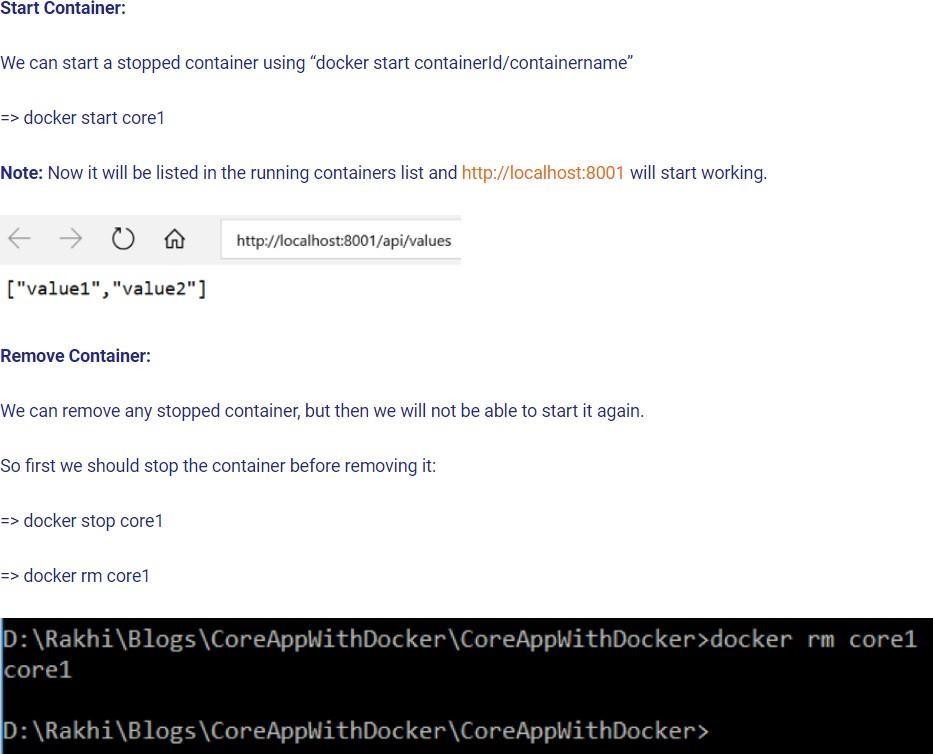






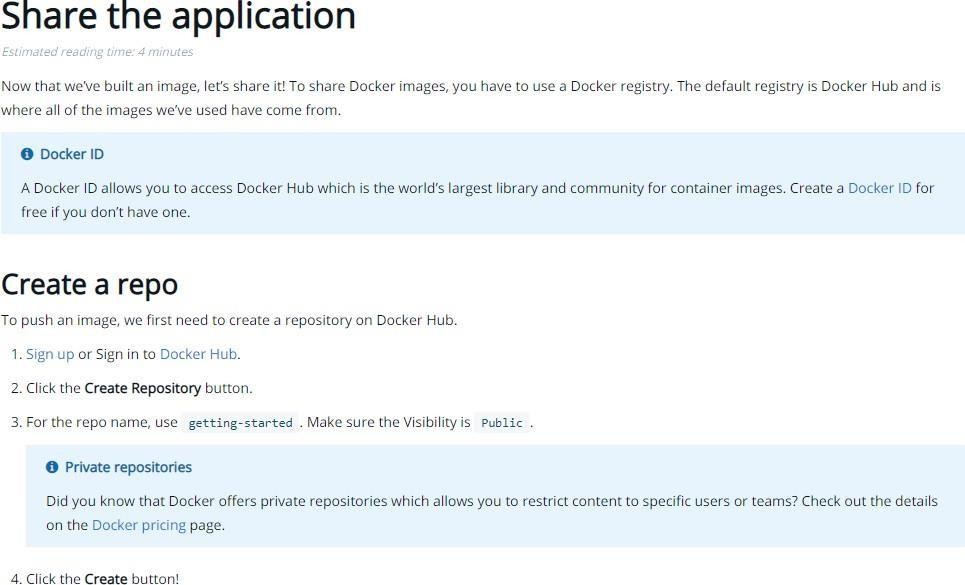


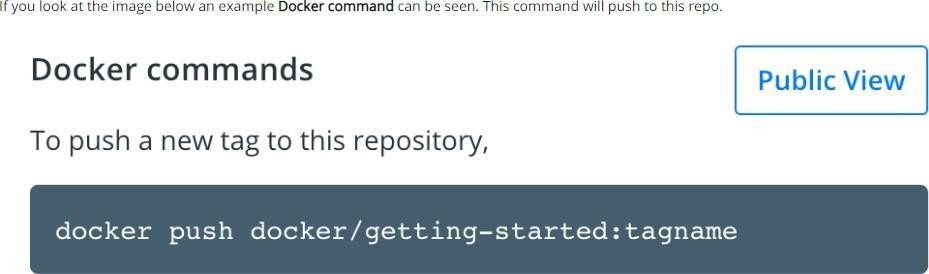




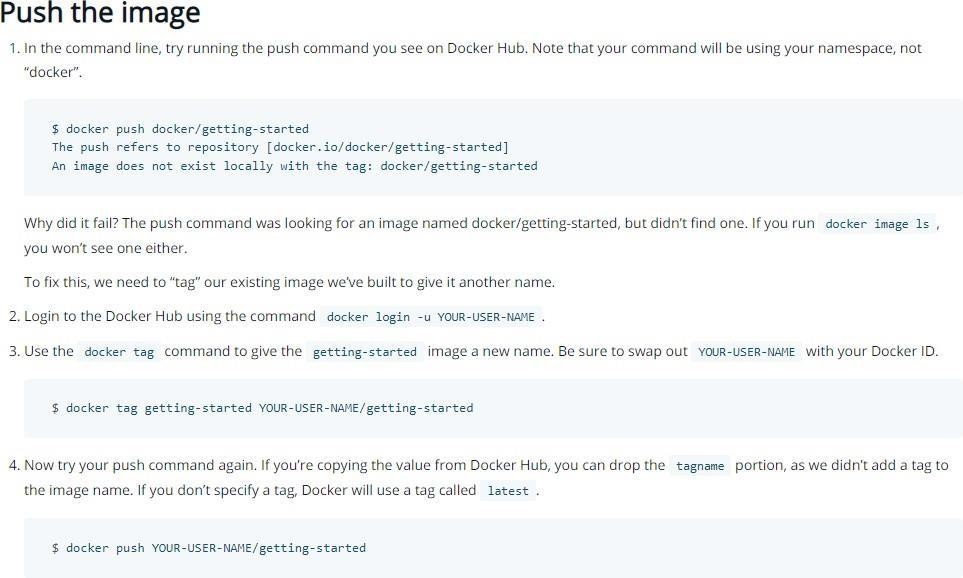


**Case 3 – Share the Application on Docker Hub Repository**





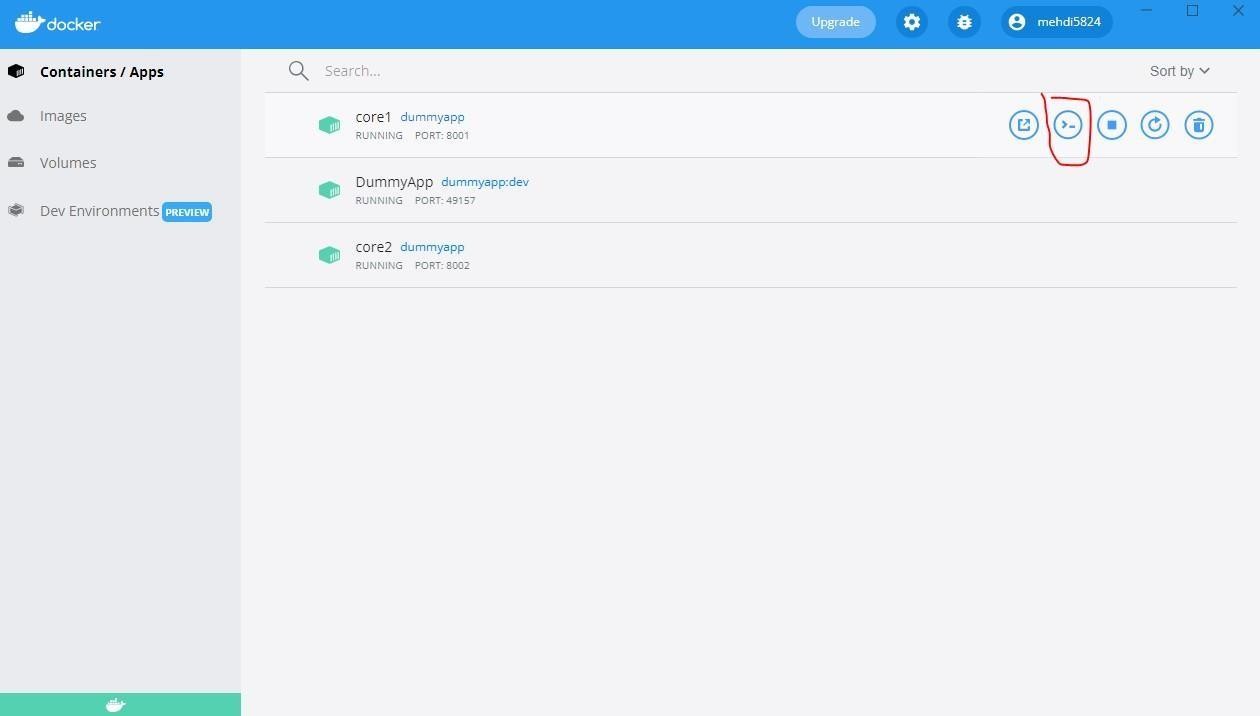
**In your case use your docker image name instead of getting-started**

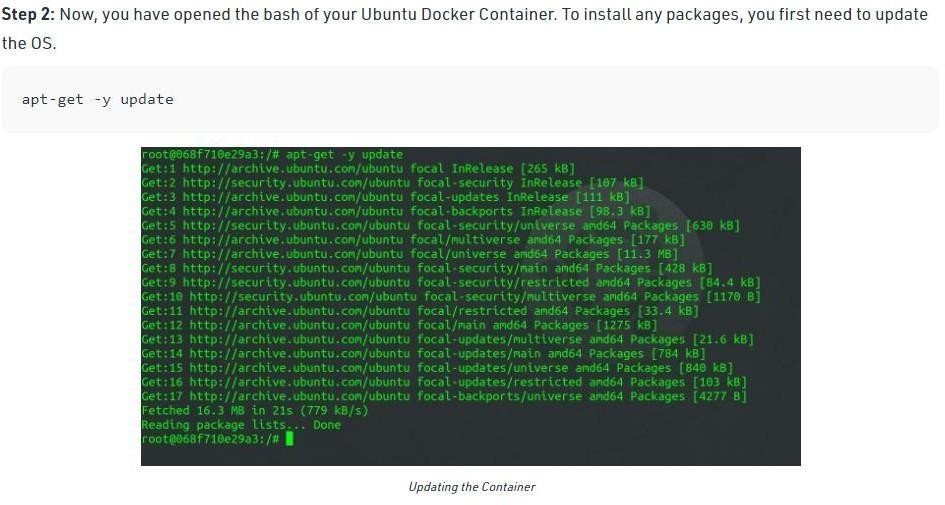


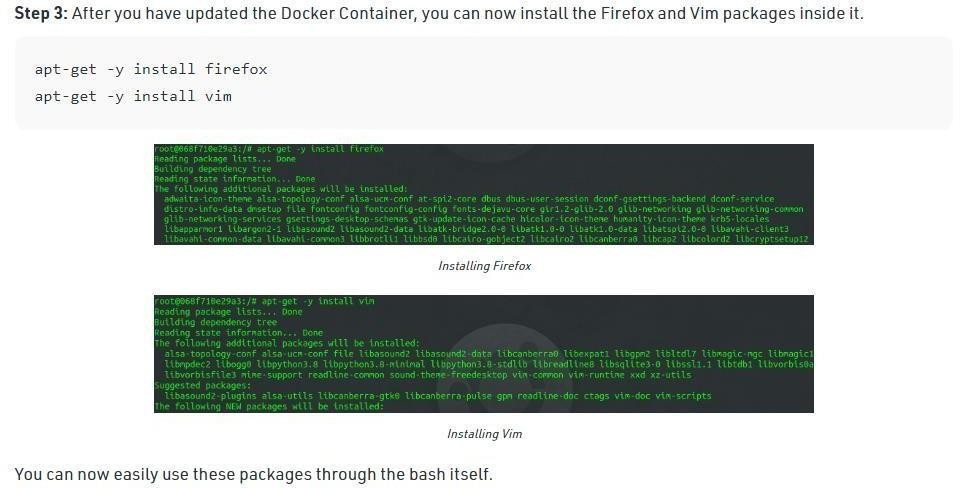
### Part 2

**Installing Software Packages in Docker**

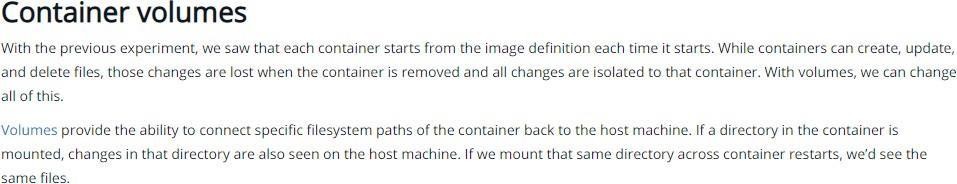
### Step 1 – Go to CLI Option on the container in Docker Desktop



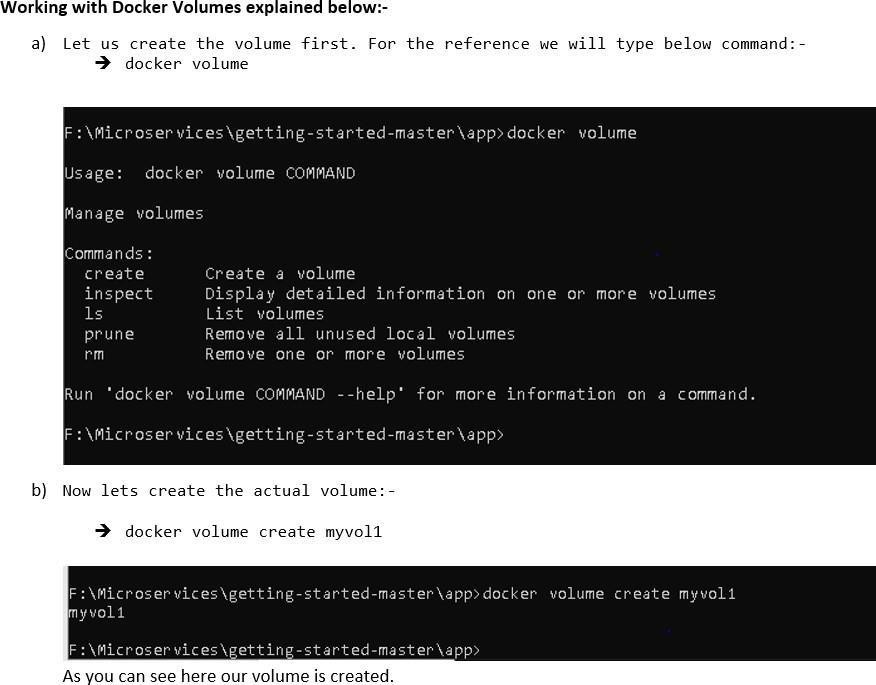


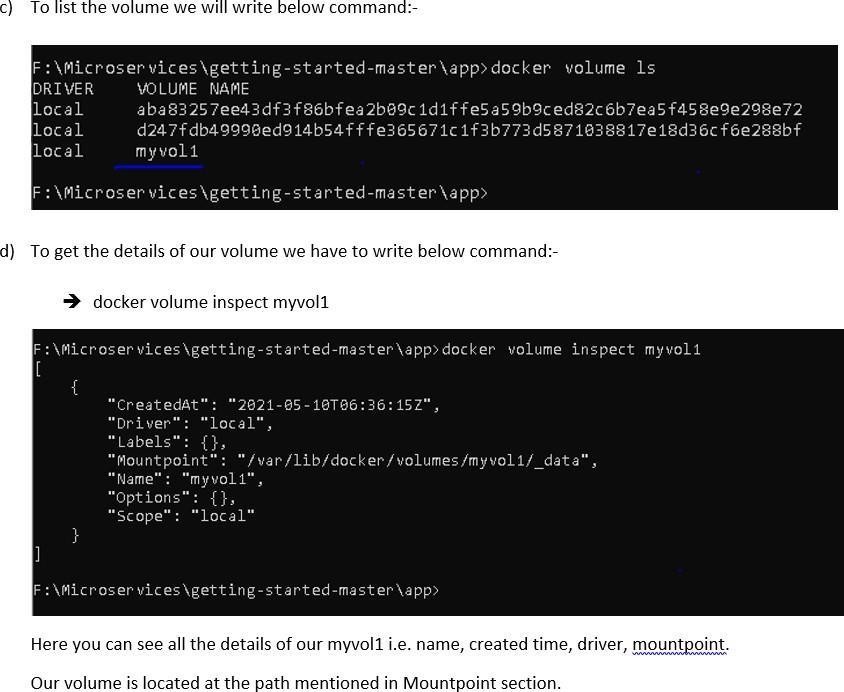


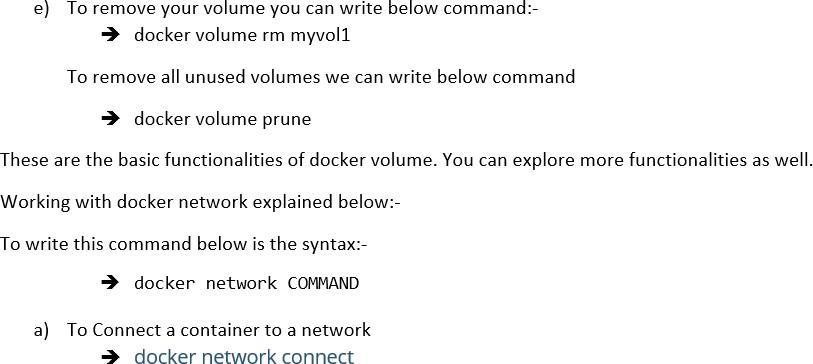
**Step 4:** Run vim to verify if the software package has been installed

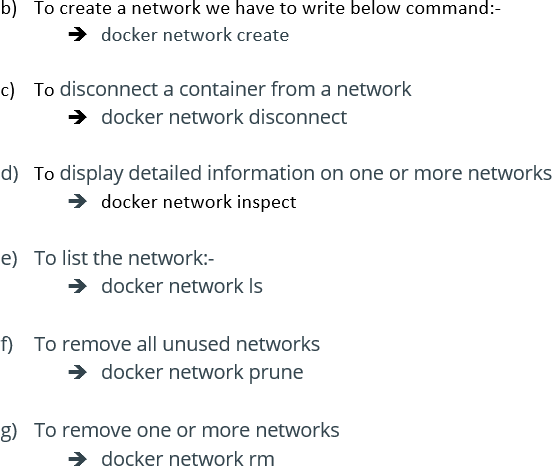


##### Step 1









G. M. Momin Women’s College 31

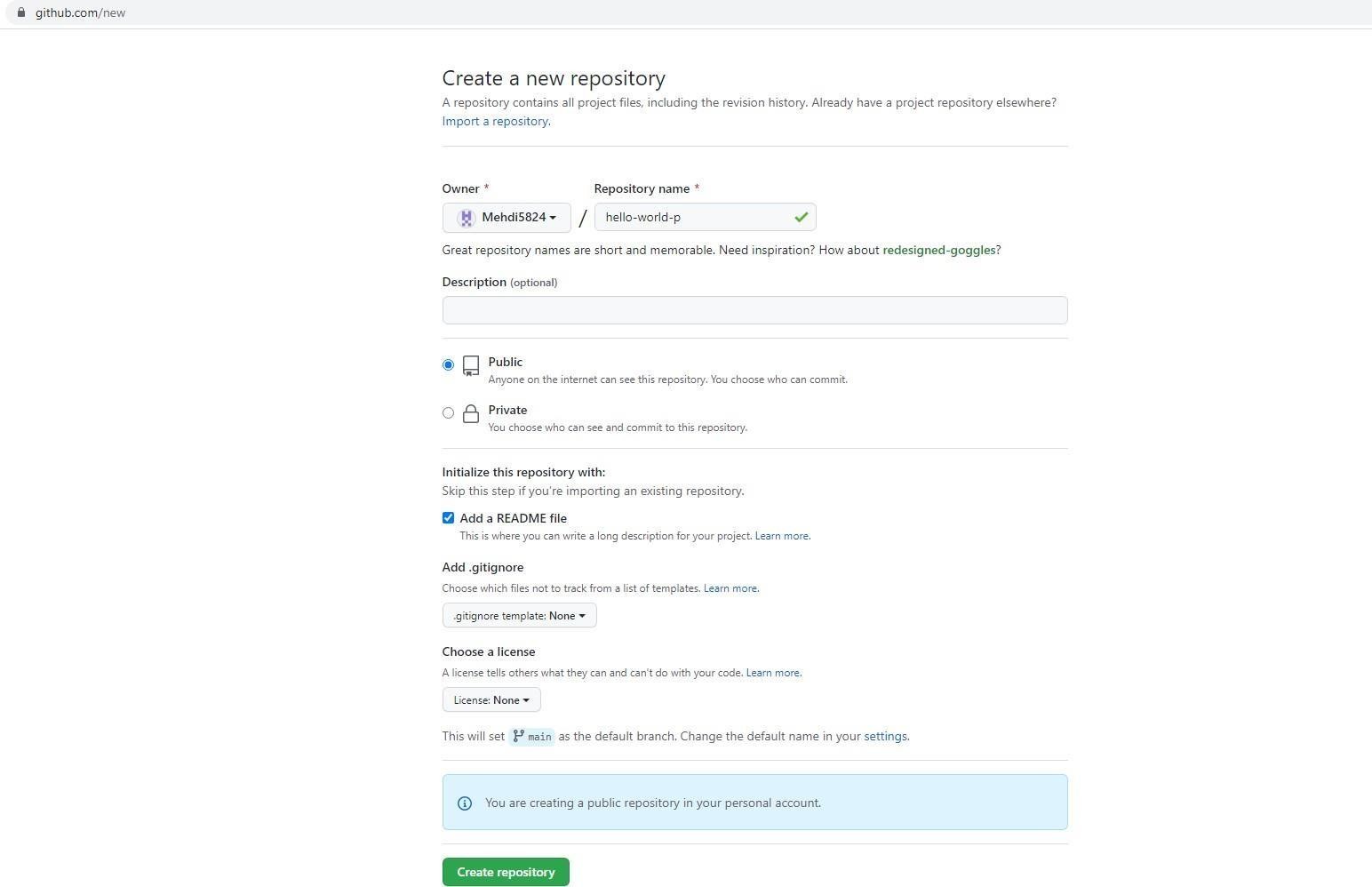
Shaikh Nasera MD Shahid Microservice

# Practical No.: 05

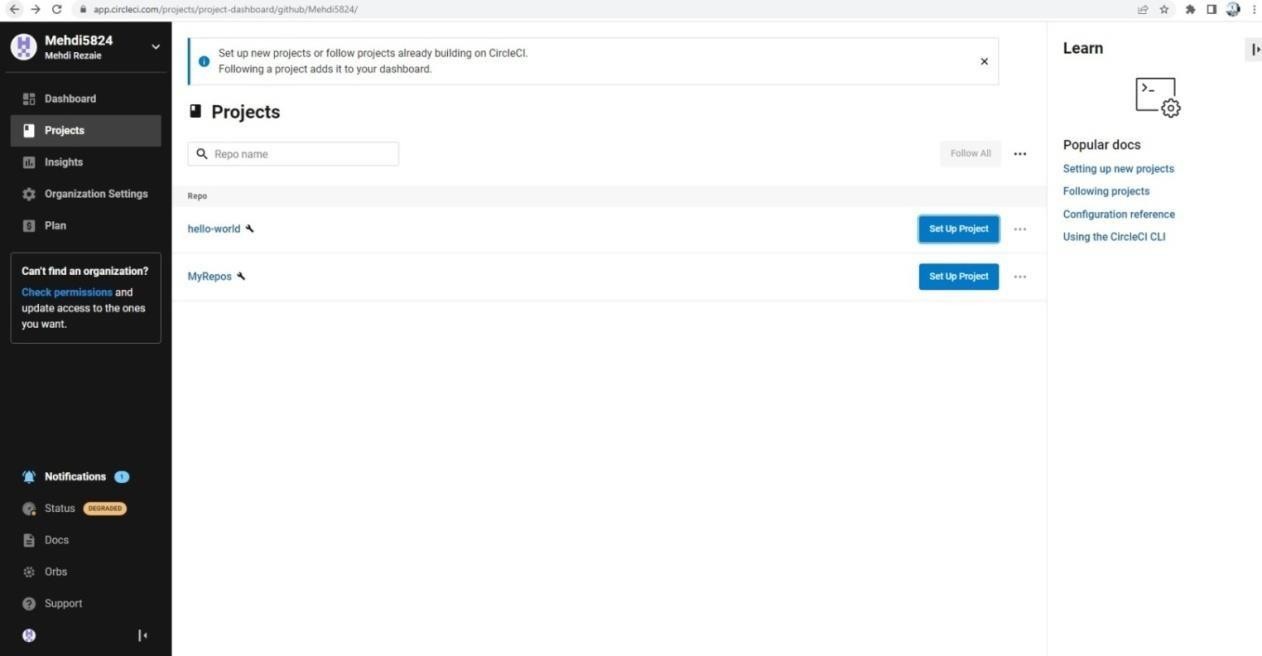
## Working with Circle CI for continuous integration

### 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/**](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 toselect 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.



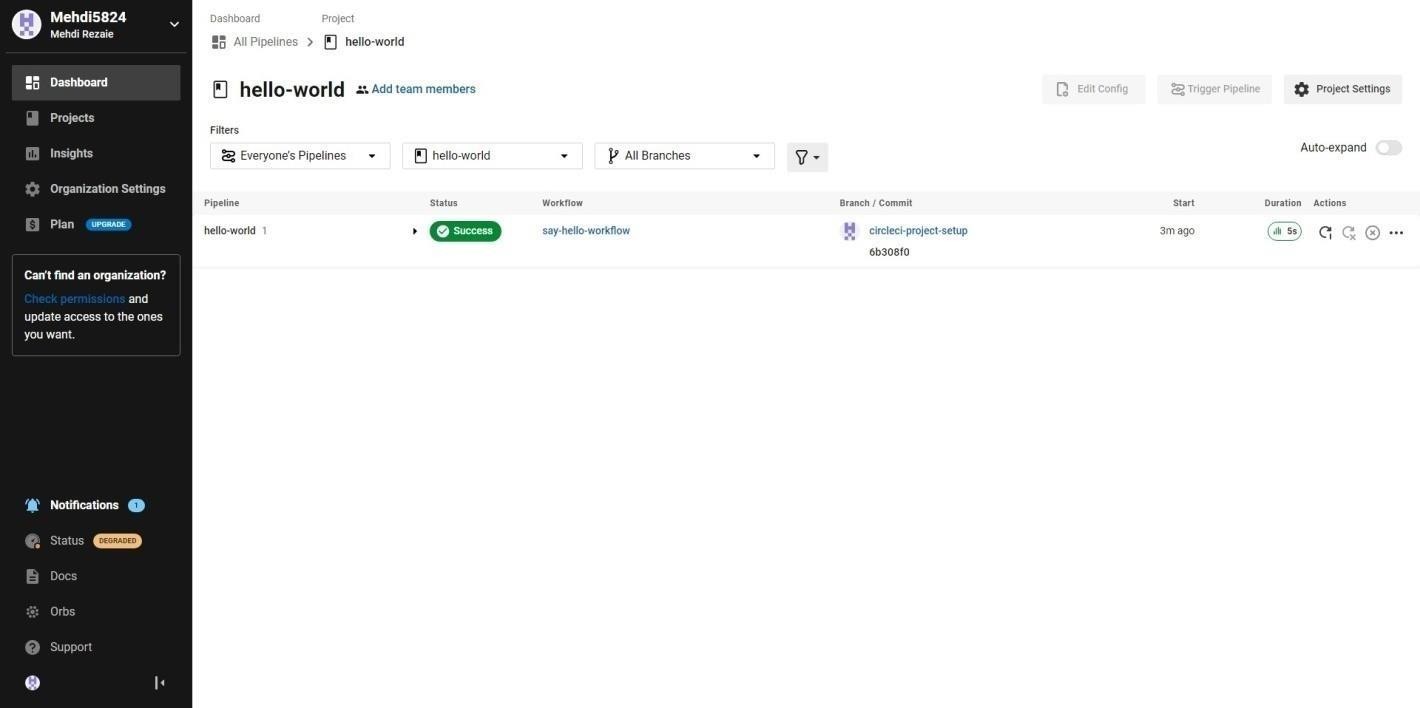
### Step 3 - Your first pipeline

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:

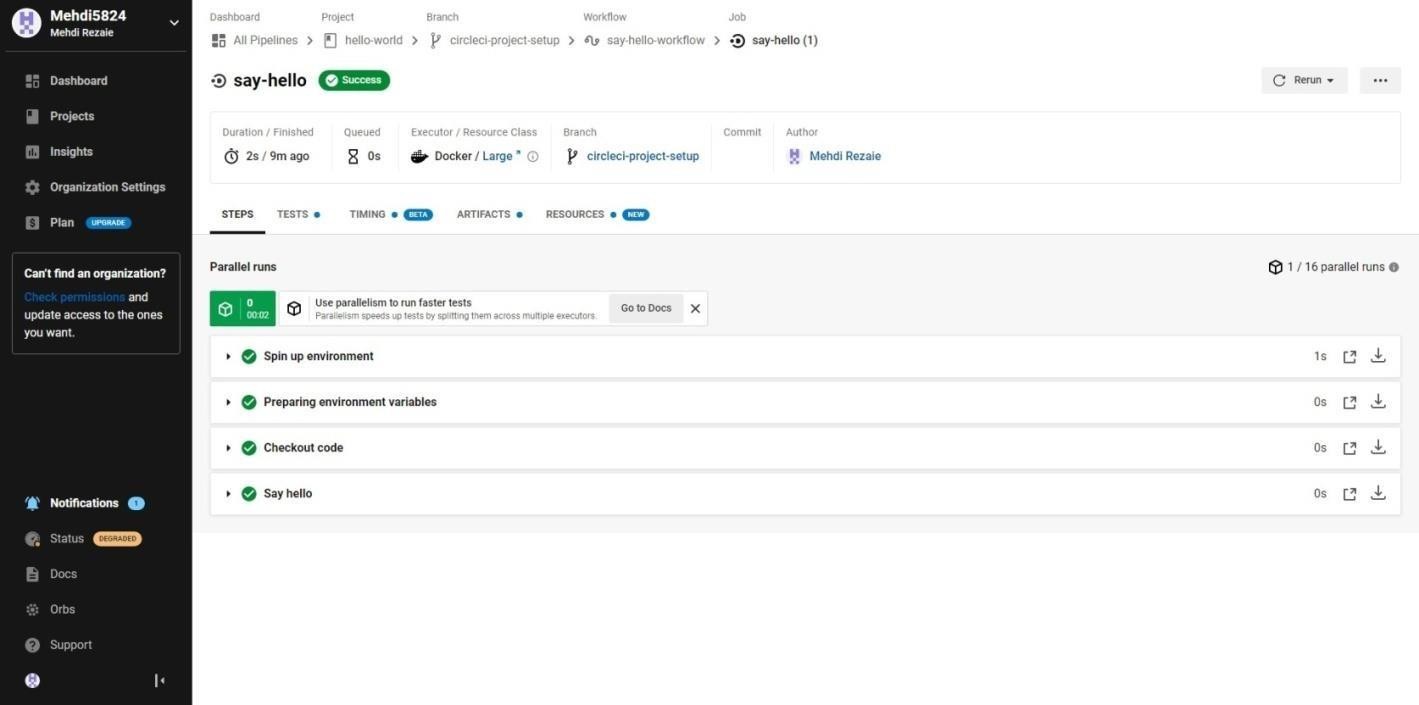
* + 1. Spin up environment
    2. Preparing environment variables
    3. Checkout code
    4. Say hello

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

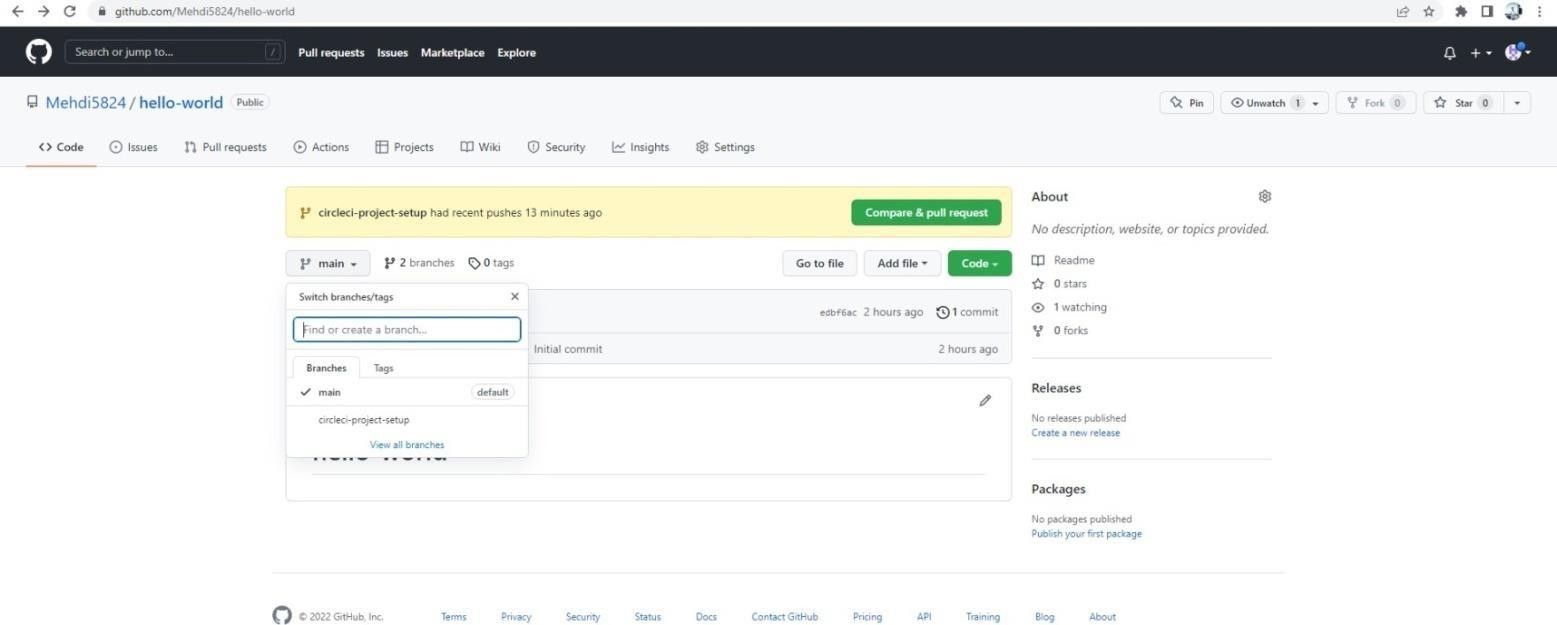


**Select “say-hello” Job with a green tick**



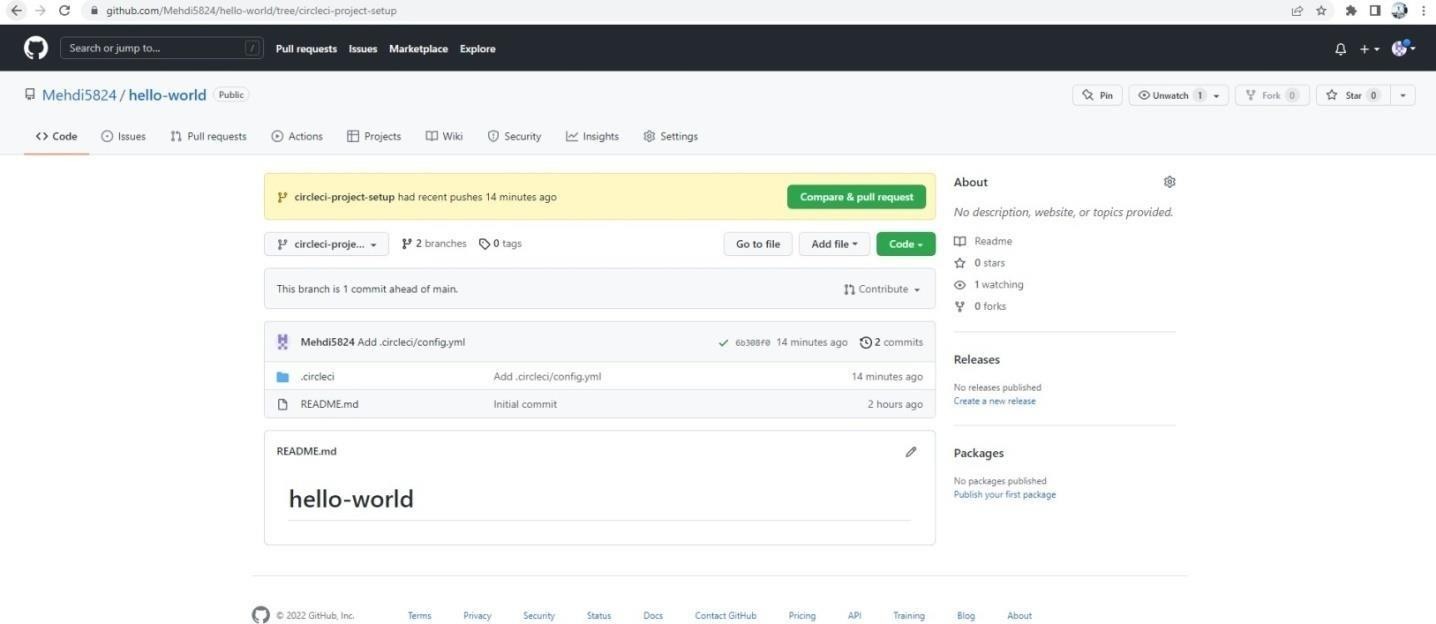


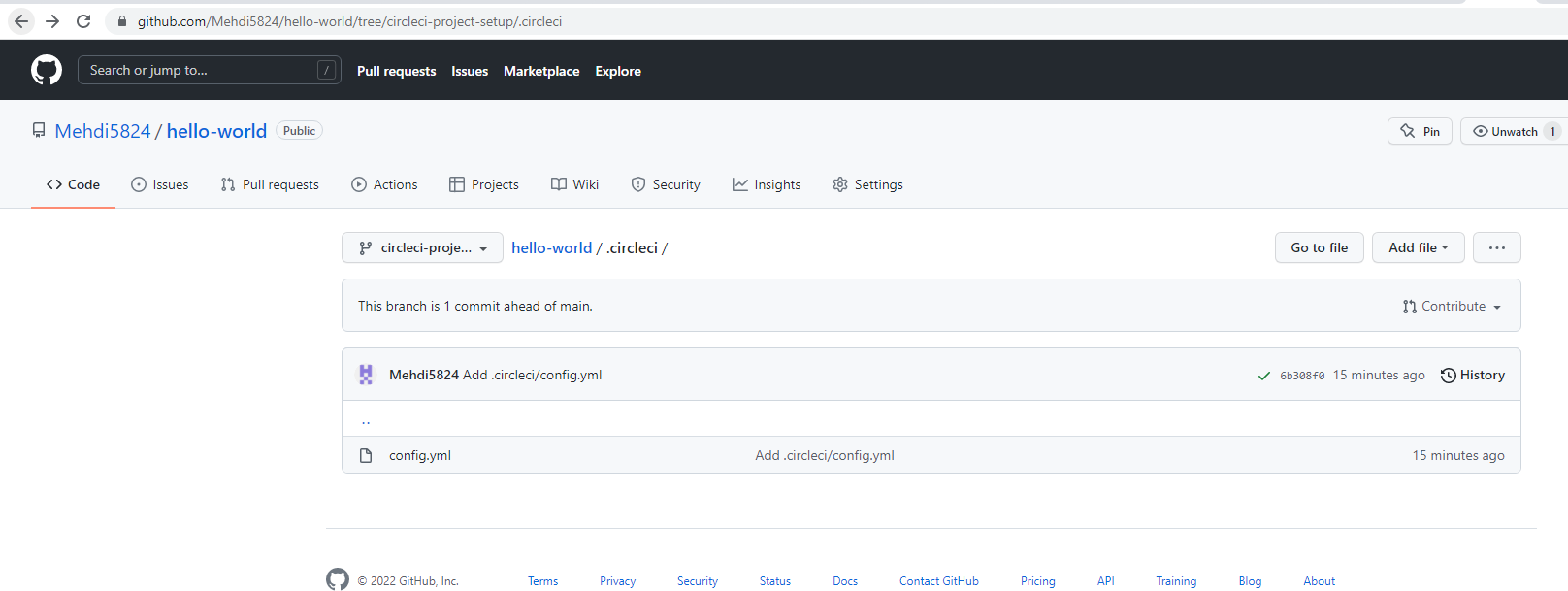
**Select Branch and option circleci-project-setup**



### 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.

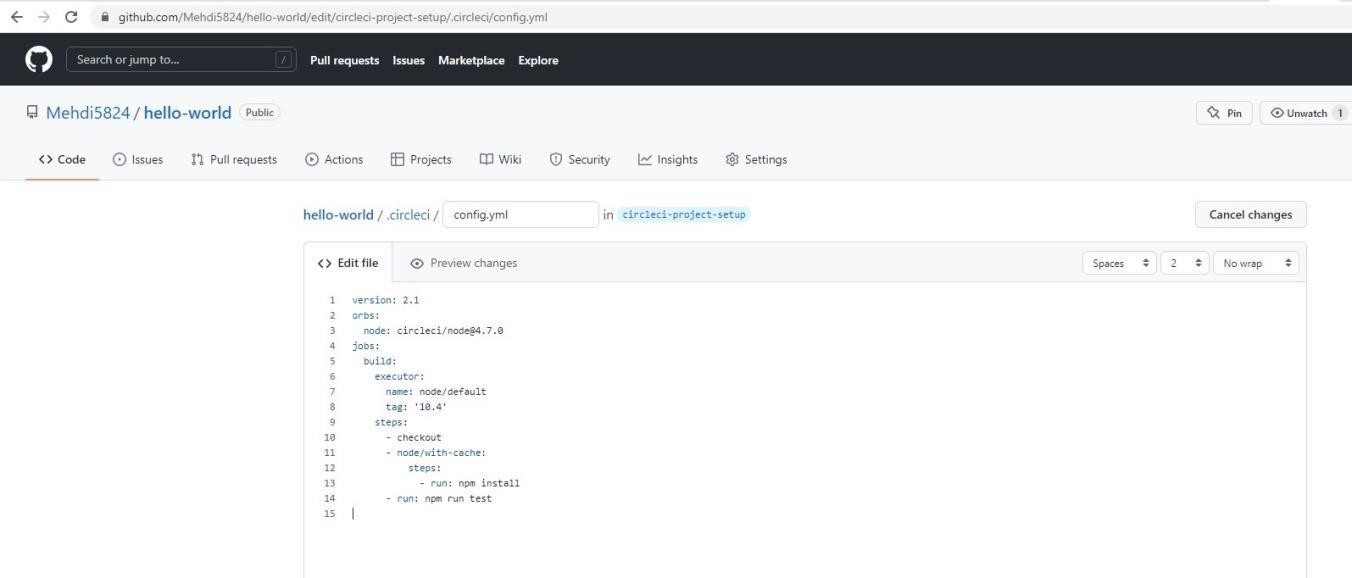




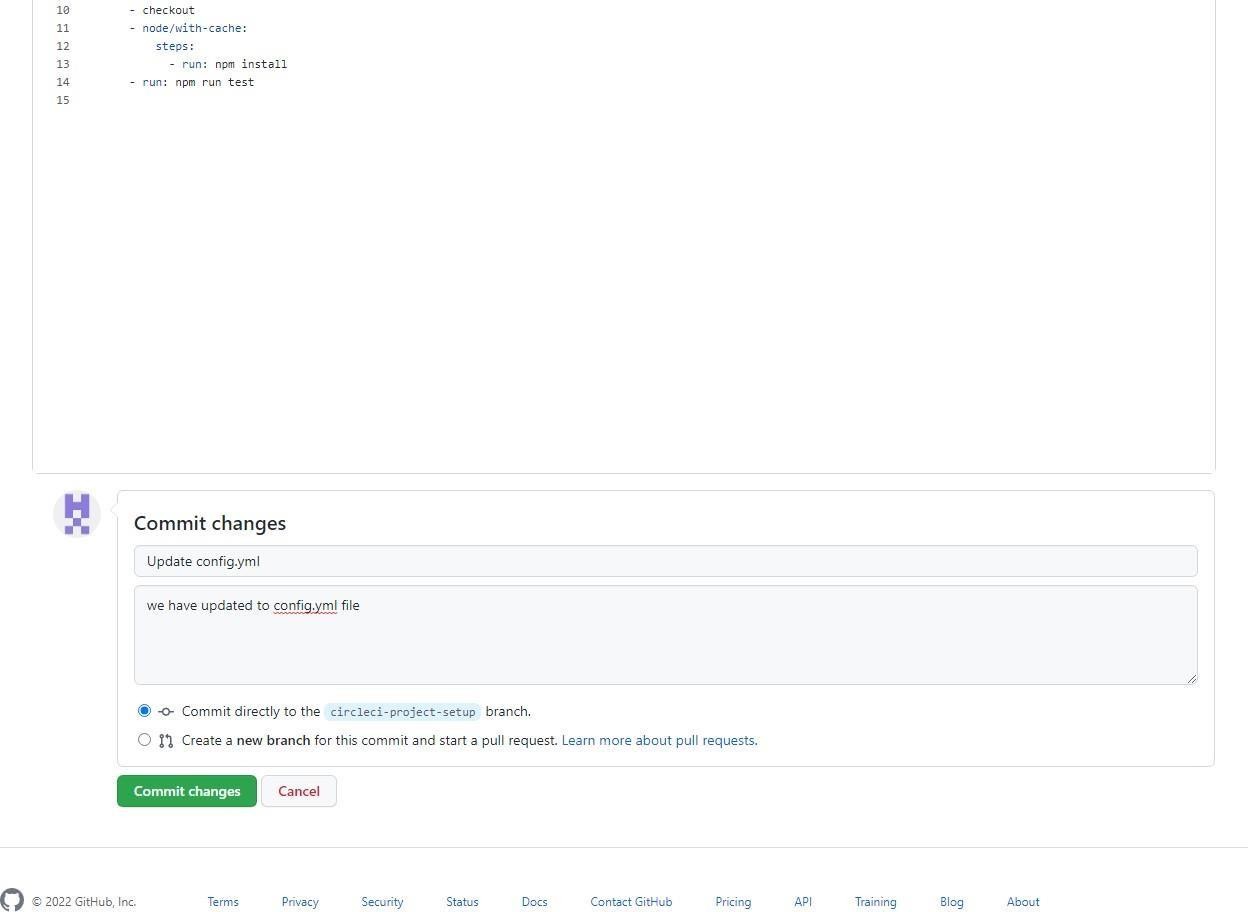




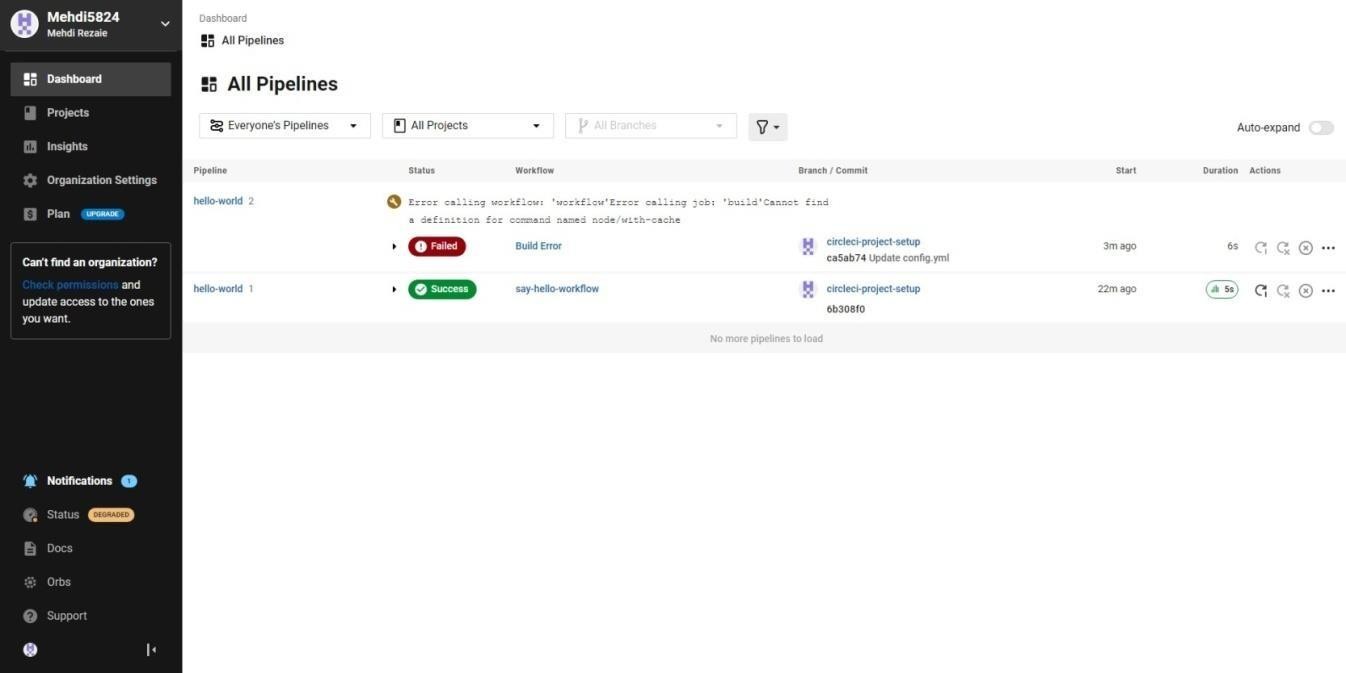
**The GitHub file editor should look like this**



##### Scroll down and Commit your changes on GitHub



**After committing your changes, then return to the Projects page in CircleCI. You should see a new pipeline running… and itwill 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

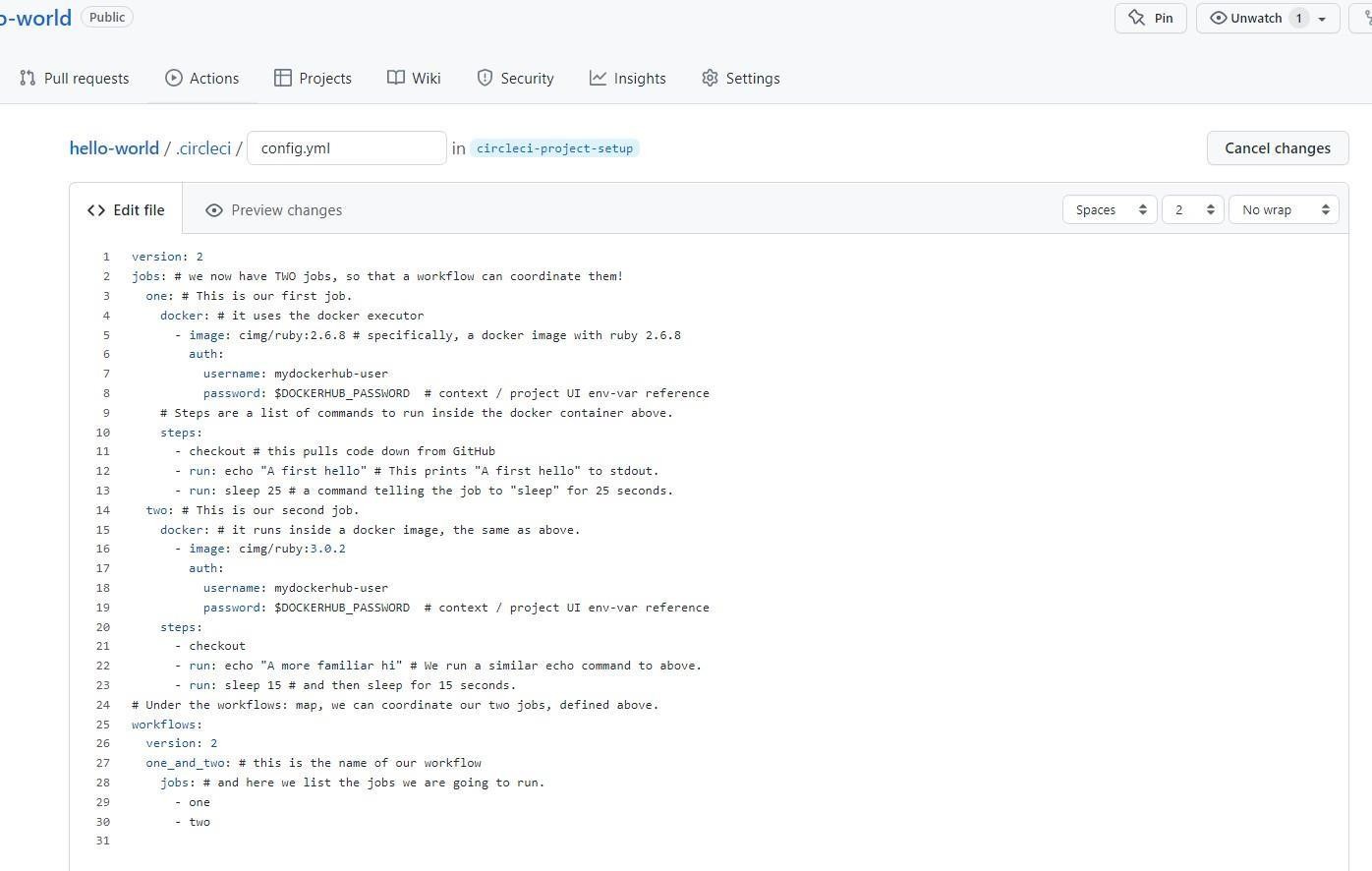
##### You do not have to use orbs to use CircleCI. The following example details how to create a custom configuration that alsouses the [workflow feature](https://circleci.com/docs/2.0/workflows) of CircleCI.

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

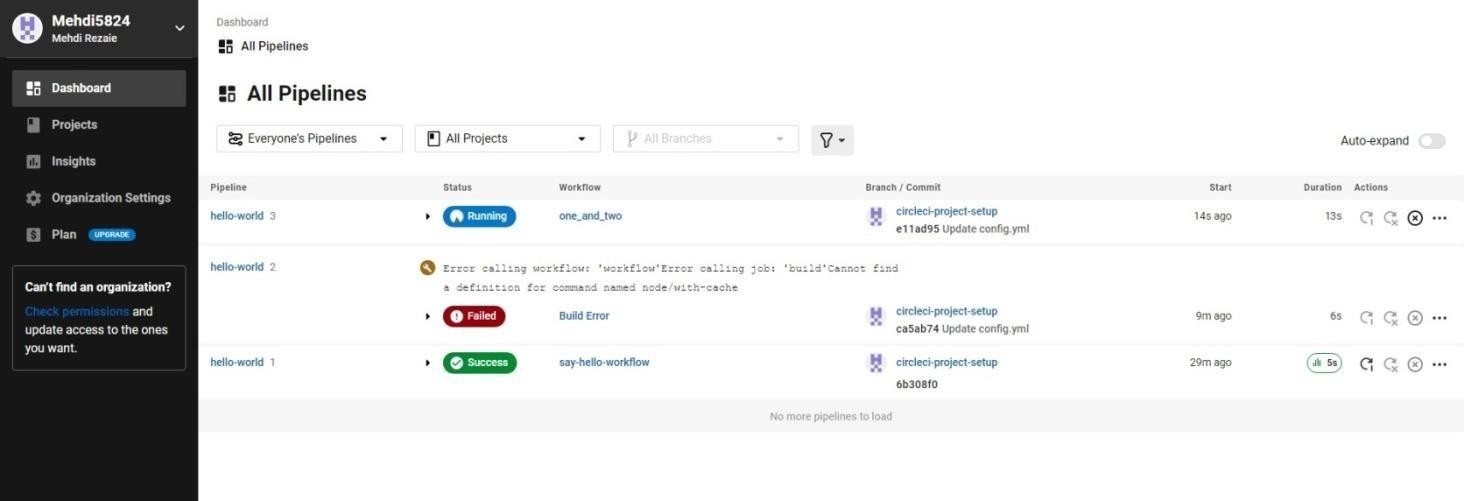


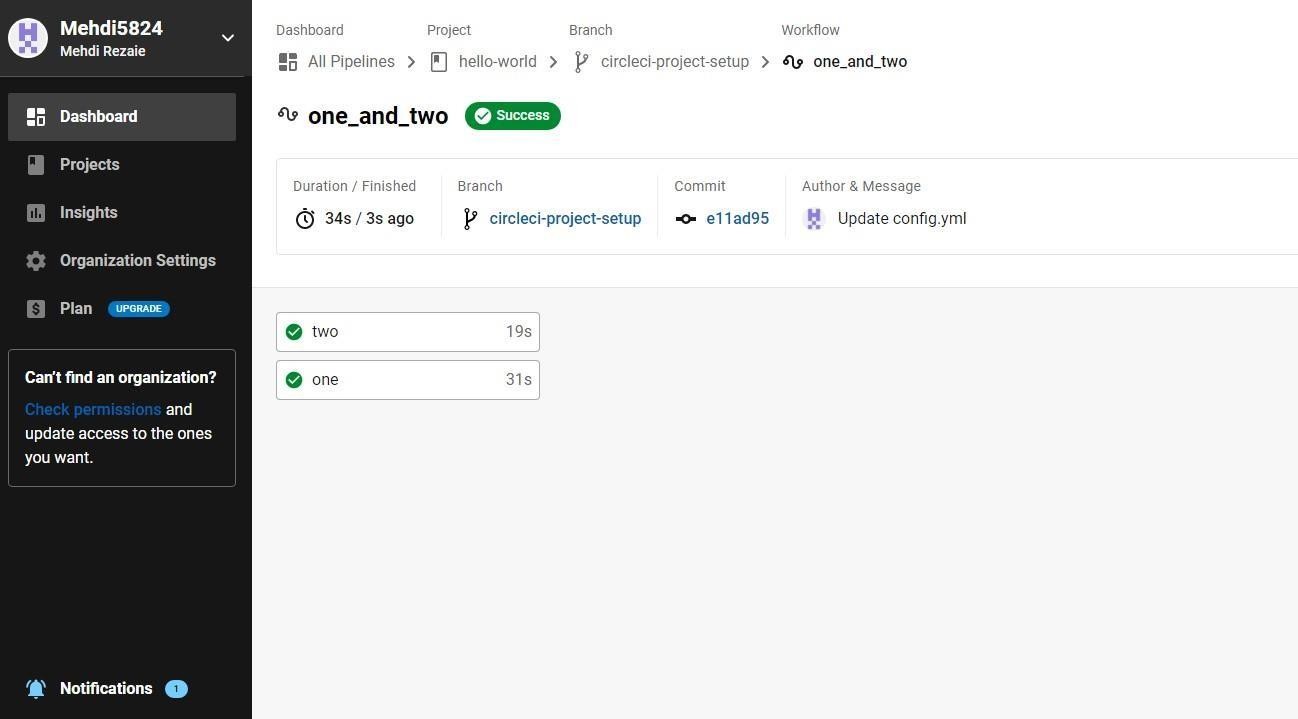
##### You don’t need to write the comments which are the text after #

1. Commit these changes to your repository and navigate back to the CircleCI Pipelines page. You should see your pipelinerunning.



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

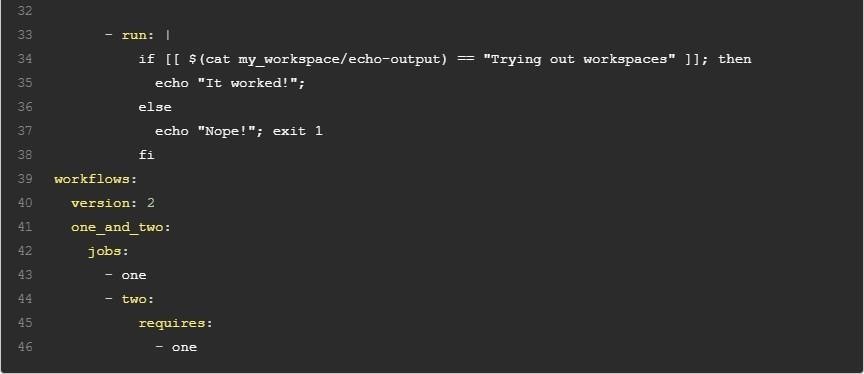




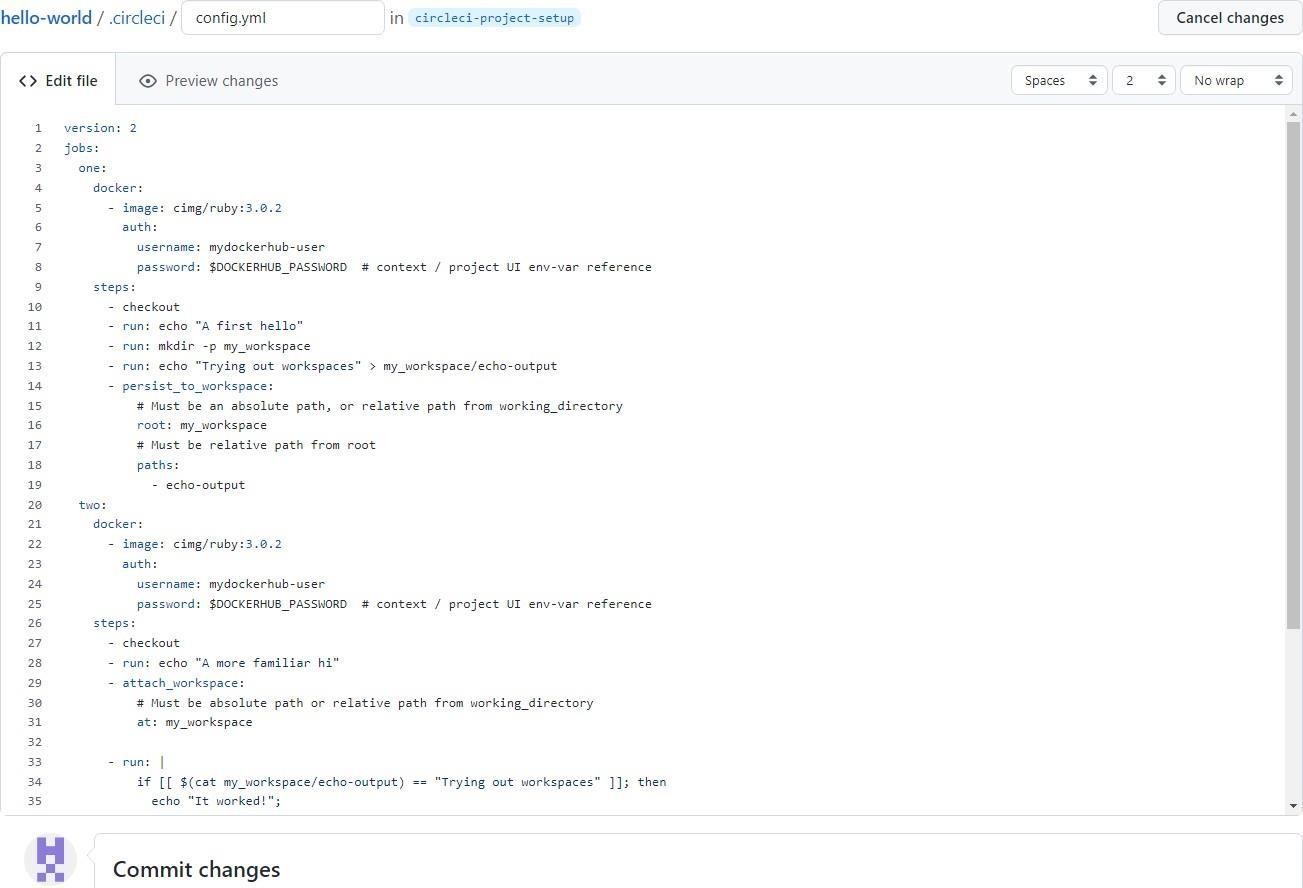
### Step 5 – Add some changes to use workspaces

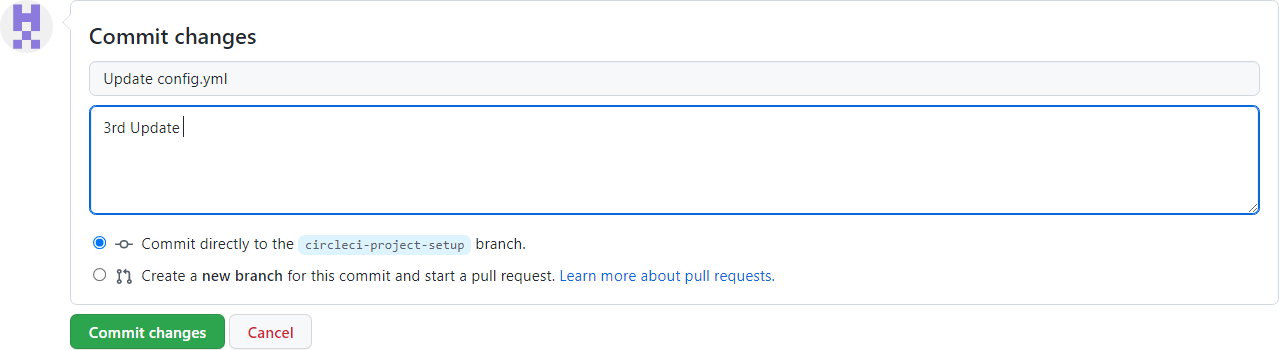
##### Each workflow has an associated workspace which can be used to transfer files to downstream jobs as the workflow progresses. You can use workspaces to pass along data that is unique to this run and which is needed for downstreamjobs. Try updating config.yml to the following:





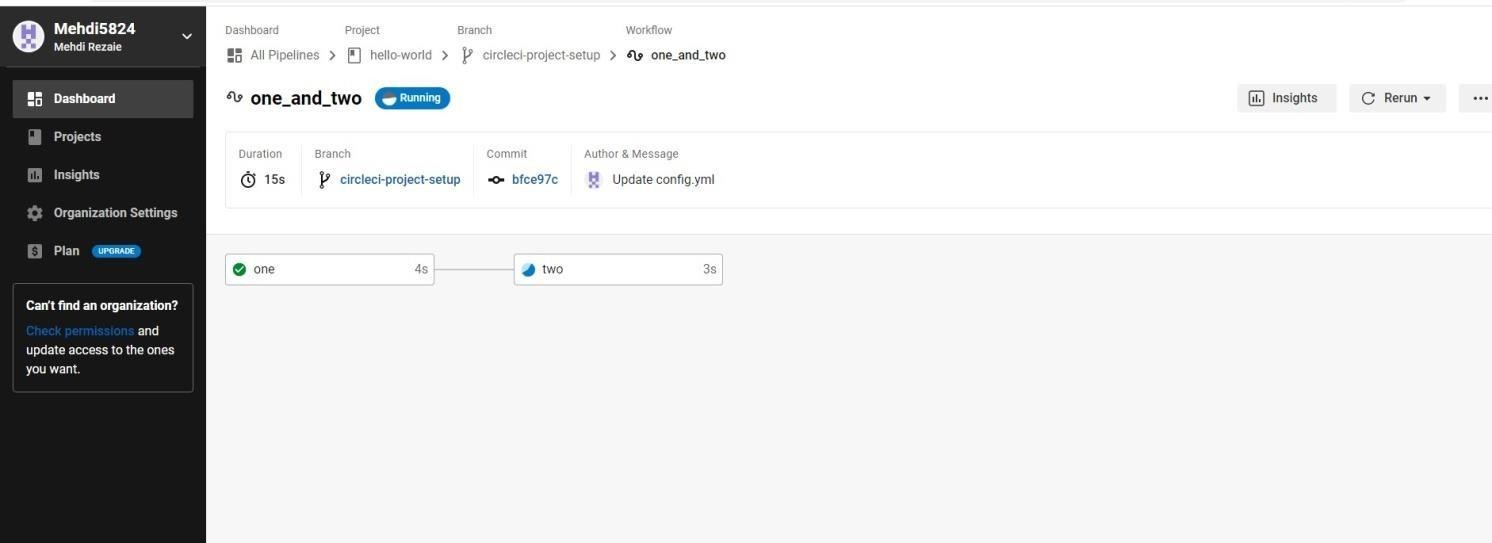
**Updated config.yml in GitHub file editor should be updated like this**





G. M. Momin Women’s College 42

**Finally your workflow with the jobs running should look like this**



# Practical No.: 06

## Working with TeamService

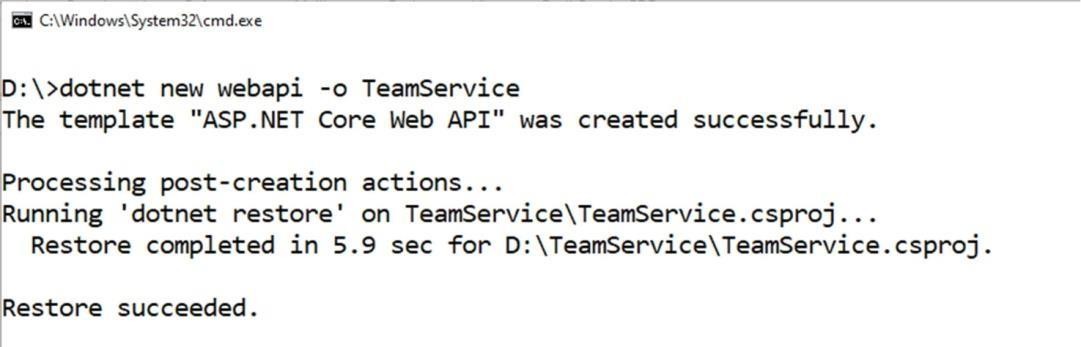
### (Install .Net core sdk first)

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

1. **Create new project:**

Command :

dotnet new webapi -o TeamService

output:

##### Remove existing weatherforecast files both model and controller files.

1. **Add new files as follows:**

##### 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;

}

}

}

##### 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;

}

}

}

##### 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)

{

}

else

{

}

}

return this.NotFound();

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)

{

}

else

{

}

}

return this.NotFound();

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)

{

}

else

{

return this.NotFound();

var q = team.Members.Where(m => m.ID == memberId);if(q.Count() < 1)

{

}

else

{

}

}

}

return this.NotFound();

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)

{

}

else

{

return this.NotFound();

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)

{

}

else

{

return this.NotFound();

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)

{

}

else

{

}

}

return this.Ok(new {TeamID = teamId });

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;

}

}

}

##### create folder “D:\TeamService\Persistence”:

1. **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);

}

}

##### 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;

}

}

}

##### add following line to Startup.cs in public void ConfigureServices(IServiceCollection services)method services.AddScoped<ITeamRepository, MemoryTeamRepository>();

1. **Now open two command prompts to run this project**

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

Commands:

dotnet run

Output:

##### On command prompt 2 Command: To get all teams

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

##### output:

**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 : 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 : To get all teams**

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

##### Output:

G. M. Momin Women’s College 51

##### 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:

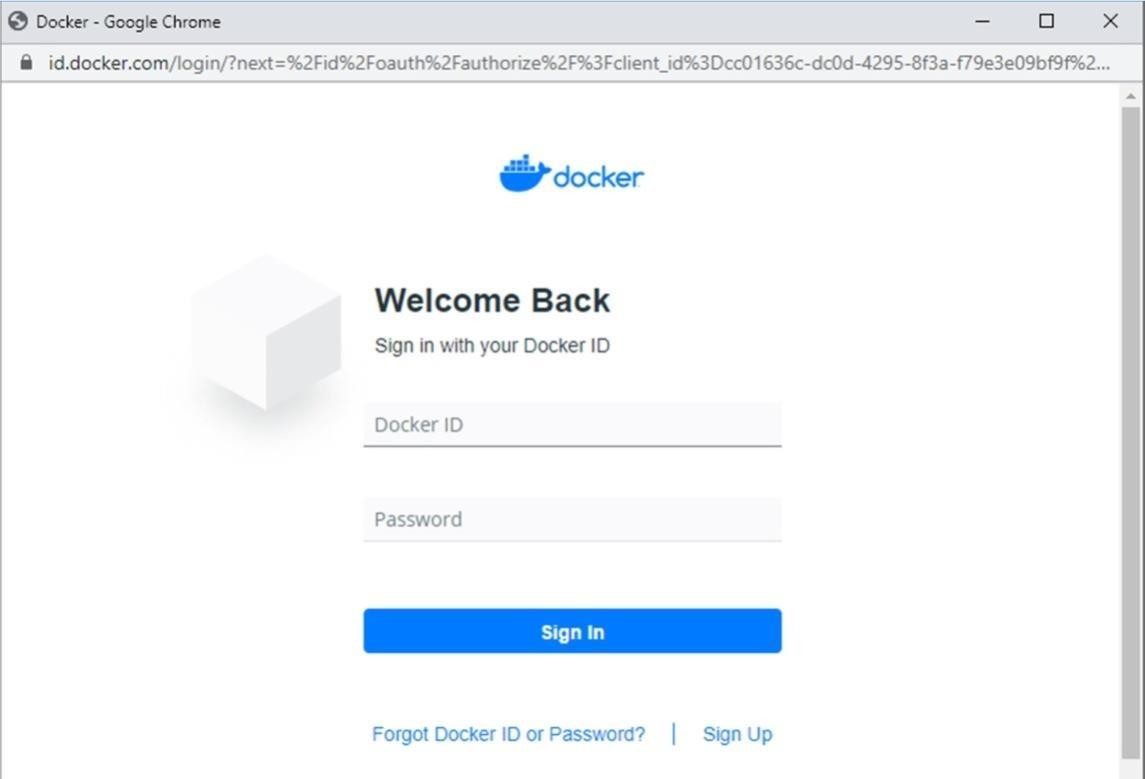
G. M. Momin Women’s College 52

# Practical No.: 07

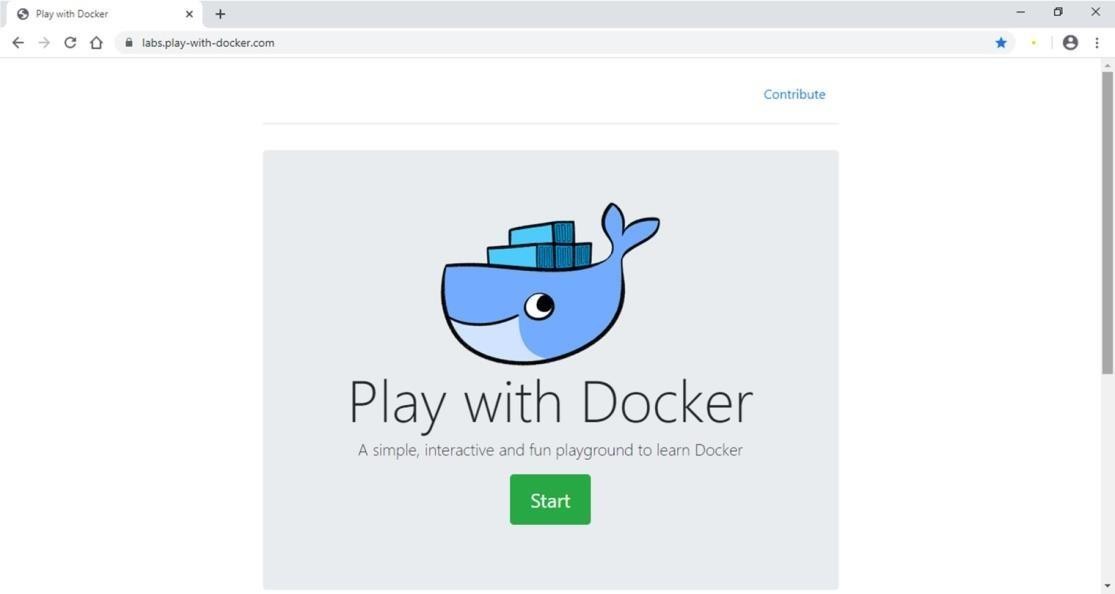
## (Backing Services)

**Running Location Service in Docker**

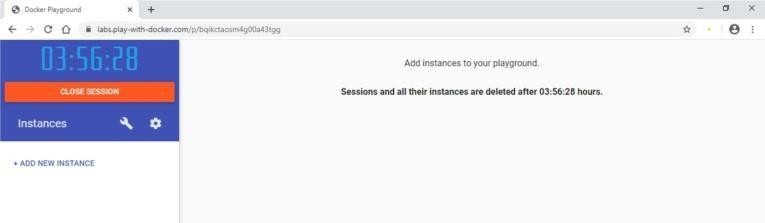
##### (create docker hub login first to use it in play with docker) Now login in to Play-With-Docker

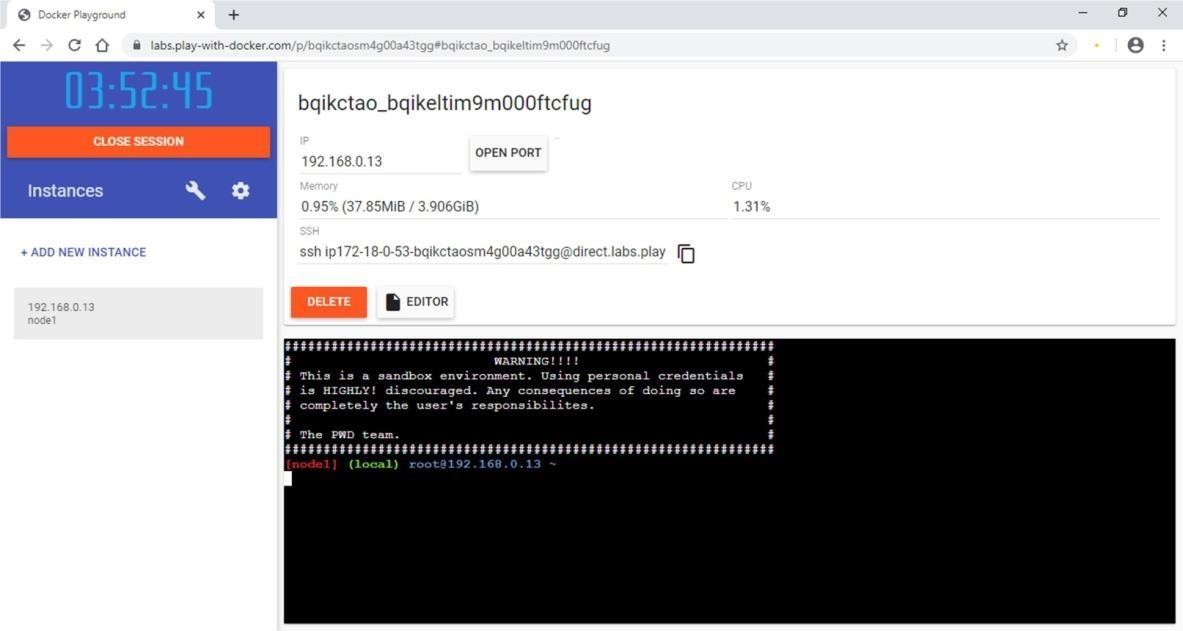


**Click on Start**



##### Click on Add New Instance





Start typing following commands

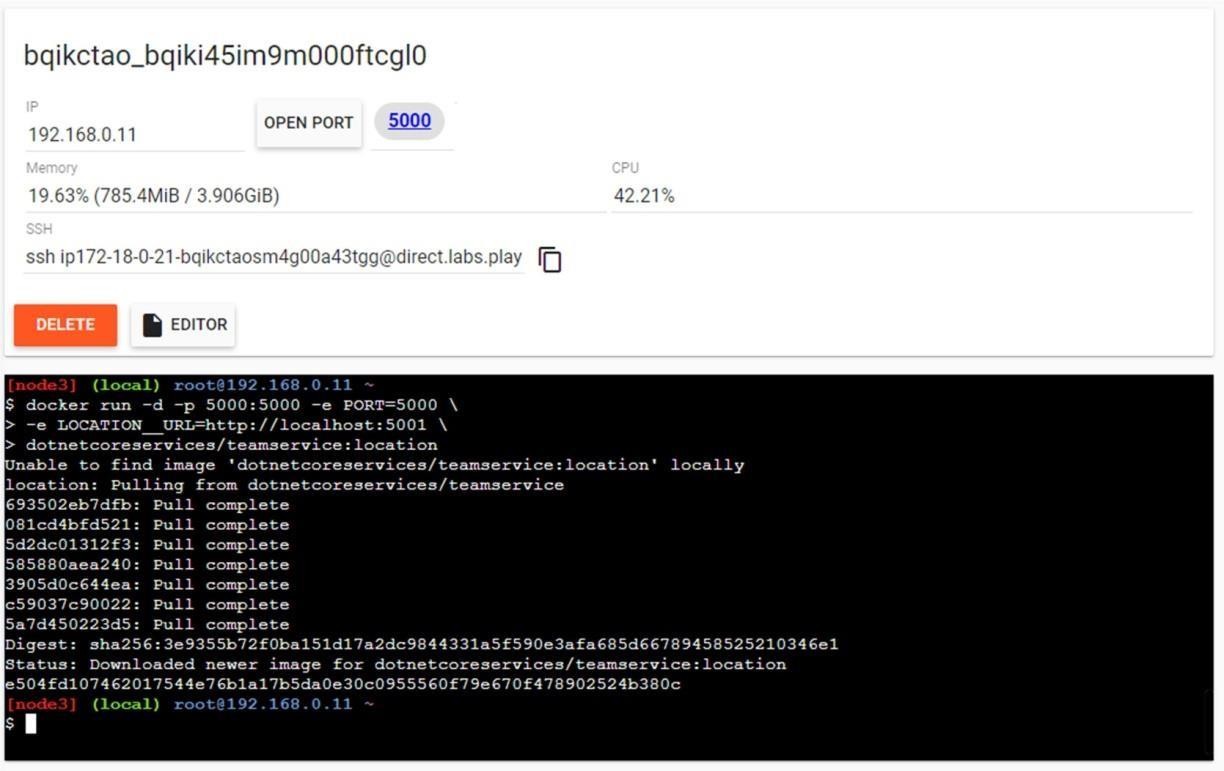
##### Command : To run teamservice

docker run -d -p 5000:5000 -e PORT=5000 \

-e LOCATION

URL=http://localhost:5001 \ dotnetcoreservices/teamservice:location

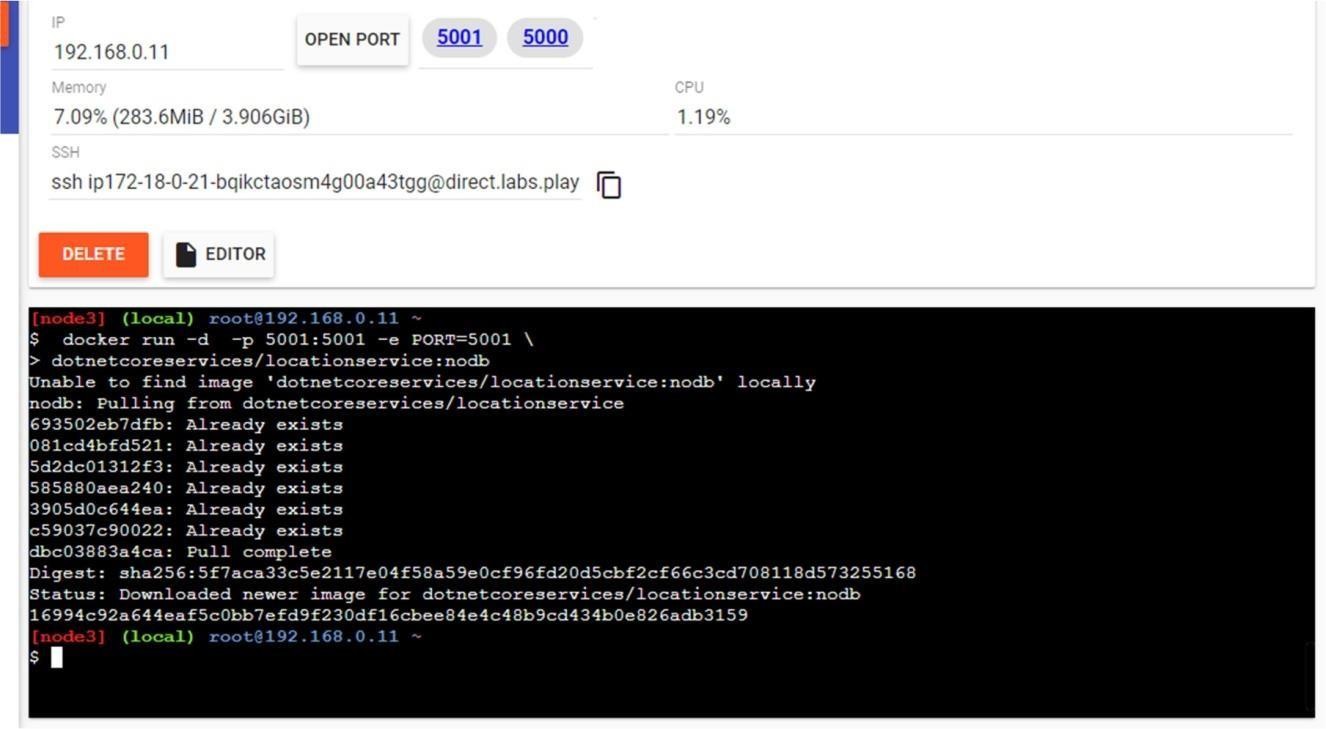
##### output: (you can observe that it has started port 5000 on top)



**Command: to run location service**

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

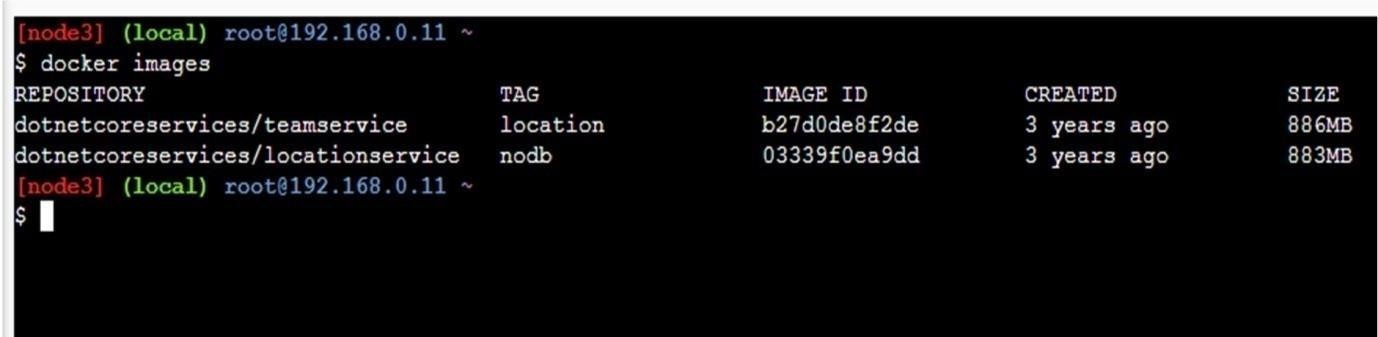
##### output: (now it has started one more port that is 5001 for location service)



**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

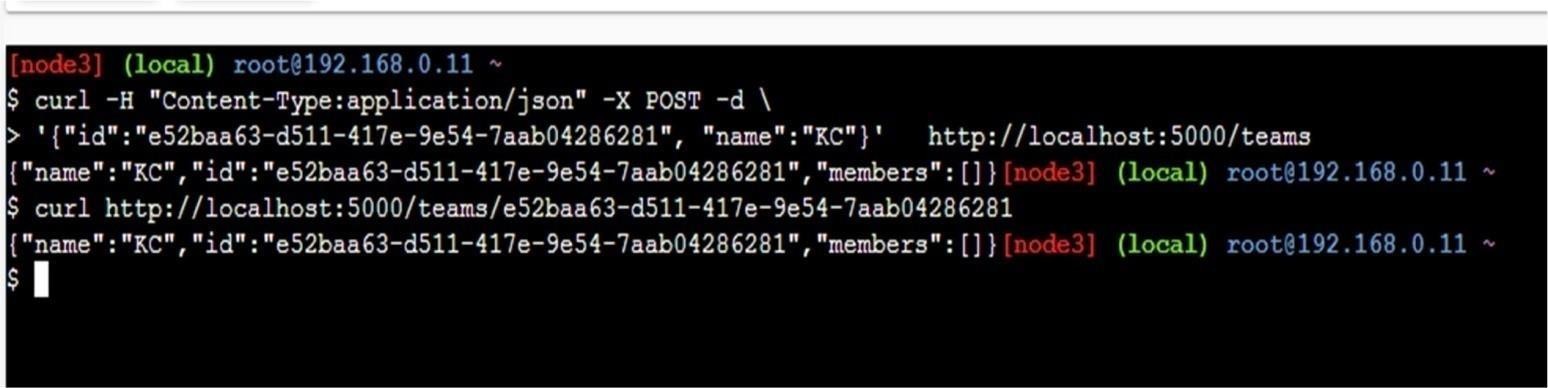
##### Output:



**Command :To confirm that team is added**

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

##### Output

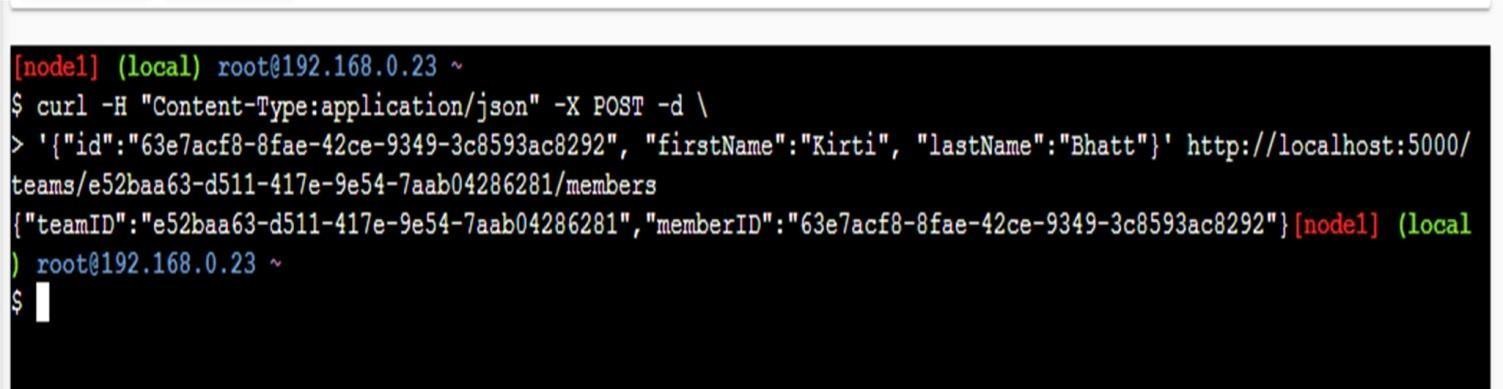


**Command : to add new member to team**

curl -H "Content-Type:application/json" -X POST -d \ '{"id":"63e7acf8-8fae-42ce-9349-3c8593ac8292", "firstName":"Kirti",

"lastName":"Bhatt"}'http://localhost:5000/teams/e52baa63-d511-417e-9e54- 7aab04286281/members

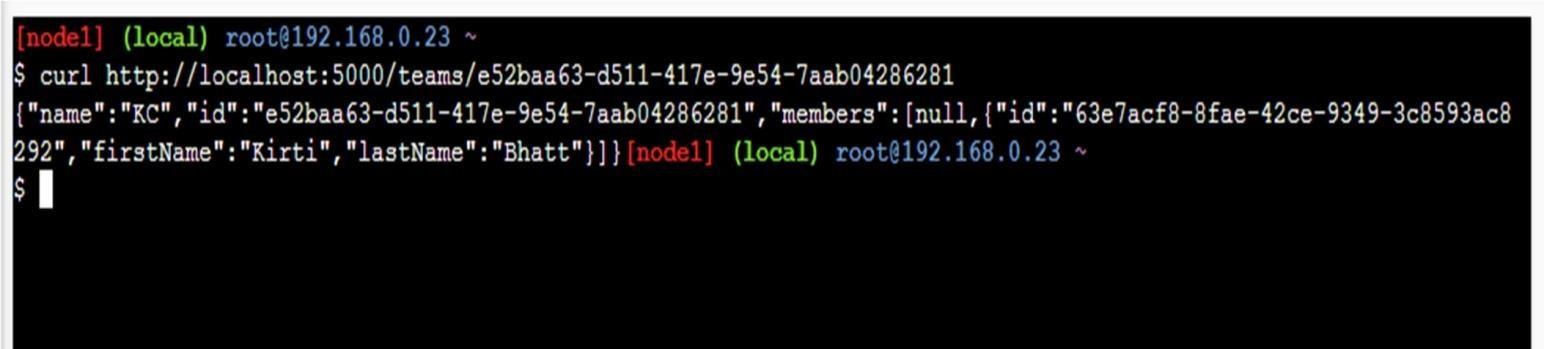
##### Output:



**Command :To confirm member added**

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

##### output:



Command : To add location for member

curl -H "Content-Type:application/json" -X POST -d \

'{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f", "latitude":12.0,"longitude":12.0,"altitude":10.0, "timestamp":0,"memberId":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}' http://localhost:5001/locations/63e7acf8- 8fae-42ce-9349-3c8593ac8292

G. M. Momin Women’s College 56

##### Output:



**Command : To confirm location is added in member**

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

##### output:

