**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Date** | **Title** | **Page No.** | **Signature** |
|  | 19/03/2022 | Create a console based ASP.net core application | 1 |  |
|  | 24/03/2022 | Create a MVC Project in ASP.net core | 4 |  |
|  | 31/03/2022 | Usage of Docker Desktop | 7 |  |
|  | 07/04/2022 | Working with Docker | 10 |  |
|  | 16/04/2022 | Building ASP.Net core REST API | 14 |  |
|  | 21/04/2022 | Working with Circle CI for continuous integration | 20 |  |
|  | 22/04/2022 | Working with Team Service | 32 |  |

**Practical No 1**

**Aim:** Create a console based ASP.net core application.

**Source Code:**

**Step 1**:

* Download the asp.net core sdk from <https://dotnet.microsoft.com/learn/dotnet/hello-worldtutorial/install>
* Install the asp.net core sdk.
* To check whether the asp.net sdk is successful install, open command prompt and type command: **dotnet**

Graphical user interface, application

Description automatically generated

* To check the version of the dotnet

Text

Description automatically generated with medium confidence

**Step 2**:

* Go to the drive where you want to create the console application. Create a folder in the drive and go to that folder. Type the following command in the command prompt to create the application.

Graphical user interface

Description automatically generated

* Restore the project and run the application

Graphical user interface, application

Description automatically generated

**Step 3**:

* Now open **HelloWorld.csproj** file, edit the code

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net6.0</TargetFramework>

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Microsoft.AspNetCore.Mvc" Version="1.1.1"/>

<PackageReference Include="Microsoft.AspNetCore.Server.Kestrel" Version="1.1.1"/>

<PackageReference Include="Microsoft.Extensions.Logging" Version="1.1.1"/>

<PackageReference Include="Microsoft.Extensions.Logging.Console" Version="1.1.1"/>

<PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="1.1.1"/>

<PackageReference Include="Microsoft.Extensions.Configuration.CommandLine" Version="1.1.1"/>

</ItemGroup>

</Project>

* Open Program.cs file and edit the code

using System;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Logging;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Configuration;

namespace HelloWorld // Note: actual namespace depends on the project name.

{

internal class Program

{

static void Main(string[] args)

{

var config = new ConfigurationBuilder()

.AddCommandLine(args)

.Build();

var host = new WebHostBuilder()

.UseKestrel()

.UseStartup<Startup>()

.UseConfiguration(config)

.Build();

host.Run();

}

}

public class Startup

{

public Startup(IHostingEnvironment env) { }

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)

{

app.Run(async (context) => { await context.Response.WriteAsync("Hello, world!");});

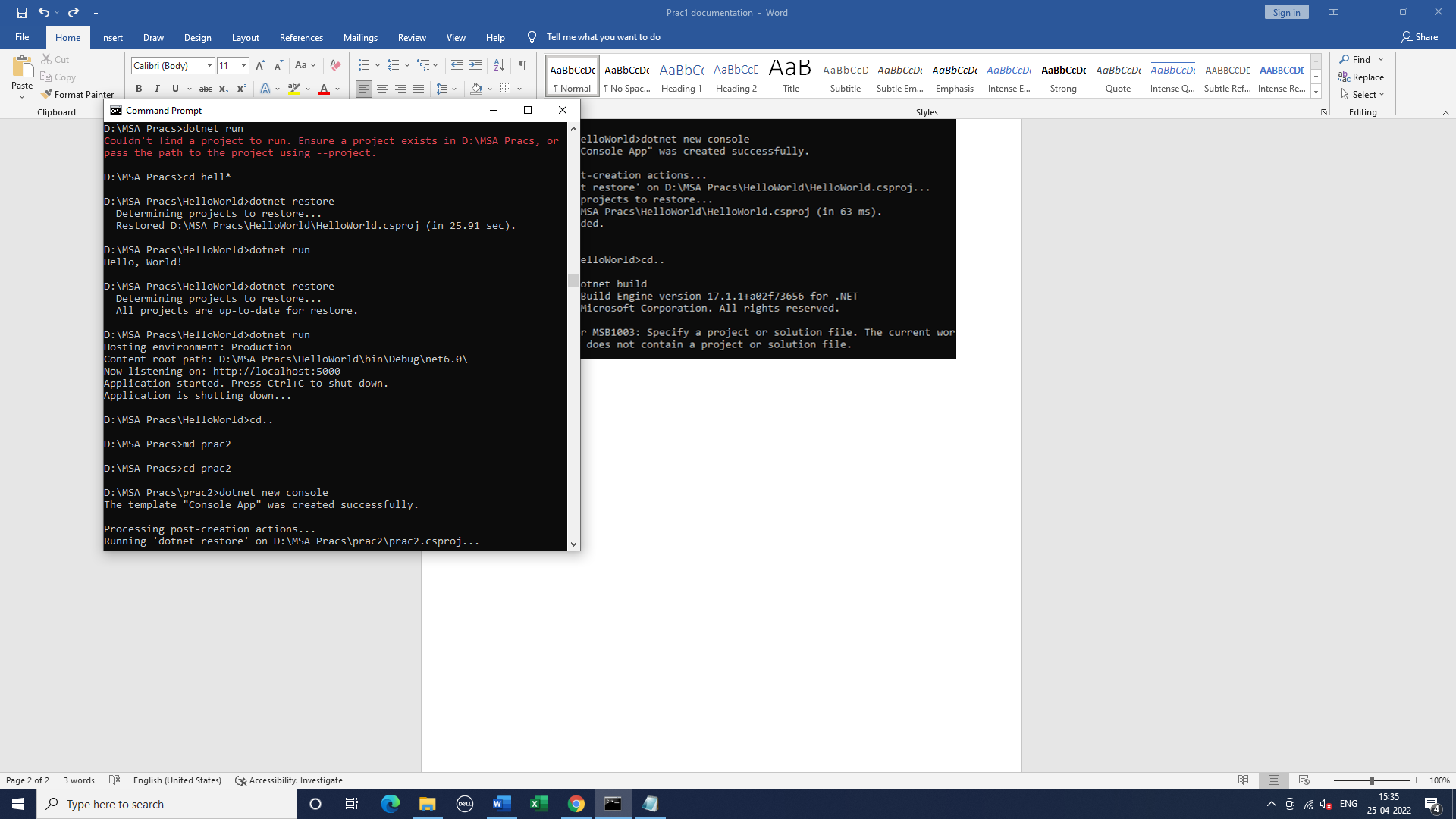
}

}

}

**Step 4**:

Restore the project.



**Output:**

Run the application

Graphical user interface, application

Description automatically generated

Now open the browser open the url: <http://localhost:5000>

Graphical user interface, application, Word

Description automatically generated

**A screenshot of a computer

Description automatically generated**

**Practical No 2**

**Aim:** Create a MVC Project in ASP.net core

**Source Code:**

**Step 1**:

Create a mvc project

dotnet new mvc –auth none

**Text

Description automatically generated**

**Step 2**:

Restore, build and run the program.

Use the first url of the command prompt in the browser and see theoutput

**Text

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

**Step 3**:

Go to Models Folder and create StockQuote.cs file in it.

using System;

namespace pracs.Models

{

public class StockQuote

{

public string Symbol {get;set;}

public int Price{get;set;}

}

}

**Step 4**:

Now go to views folder and then in home folder. Edit the index.cshtml file

@{

ViewData["Title"] = "Home Page";

}

<div class="text-center">

<h1 class="display-4">Welcome</h1>

Symbol: @Model.Symbol <br/>

Price: $@Model.Price <br/>

</div>

**Step 5**:

Now go to controller folder and edit HomeController.cs

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 pracs2.Models;

namespace pracs2.Controllers;

public class HomeController : Controller

{

public async Task <IActionResult> Index()

{

var model= new StockQuote{ Symbol="Nike", Price=3200};

return View(model);

}

}

**Step 6**:

**Text

Description automatically generated**

**Output:**

Open the first url in the browser and see the output

**Graphical user interface, text, application

Description automatically generated**

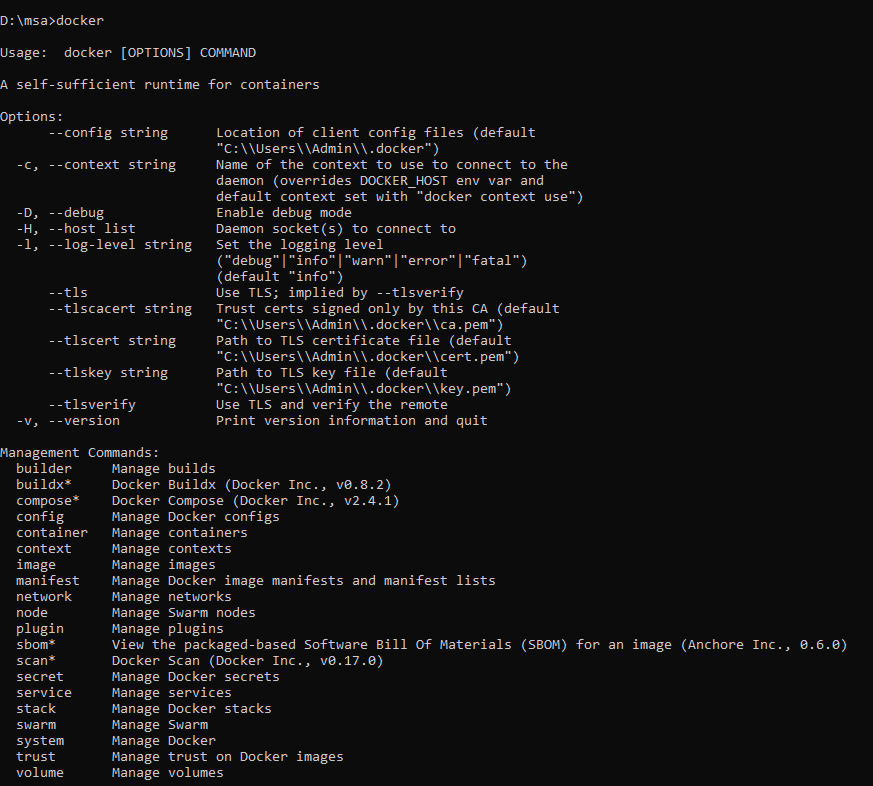
**Practical No 3**

**Aim:** Usage of Docker Desktop

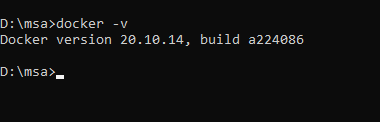
**Commands & its output:**

Open command prompt

* To check whether docker is installed properly   
  $ docker

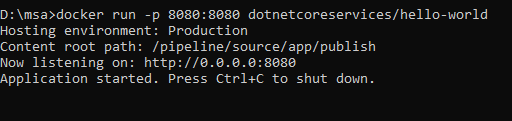


* To see the version of the docker  
  $ docker -v



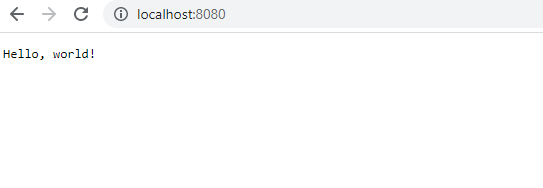
* To run hello-world image

$ docker run -p 8080:8080 dotnetcoreservices/hello-world



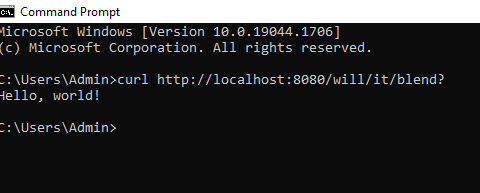
* Run localhost in the browser

http://localhost:8080

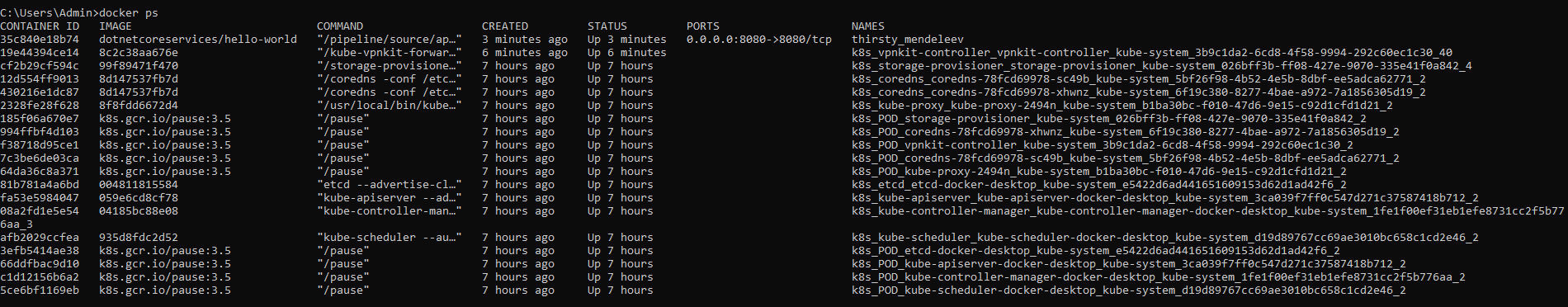


* To see the output in the command prompt

$ curl http://localhost:8080/will/it/blend?



* To see the images in the docker  
  $ docker ps



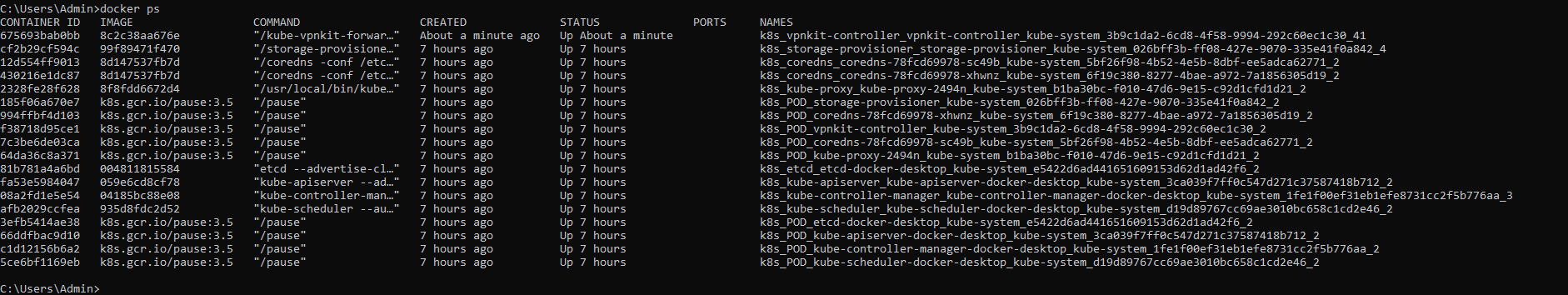
* To terminate the image in the docker.   
  note the container id of the docker that you want to terminal and replace the <Containerid> in the below command

$ docker kill <containerid>



To check whether the docker is terminated or not

$ docker ps

****

**Practical No 4**

**Aim:** Working with Docker

**Commands and its output:**

**Step 1:**

Create a account in the docker hub. Remember the username and password of the account

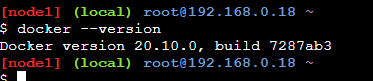
**Step 2:**

* Now to go <https://labs.play-with-docker.com/> and click on **Start** button.
* Click on **Add New Instance**. You will see the editor open in the right pane. Give the commands in the editor

**Step 3:**

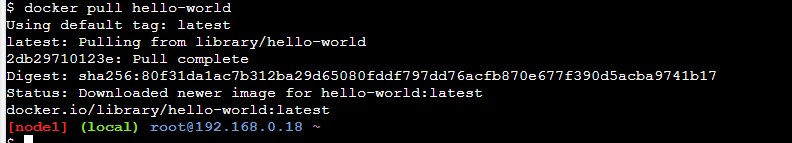
* To check the version of the docker

$ docker – version



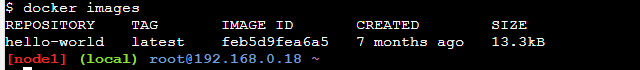
* To pull the readymade image

$ docker pull hello-world



* To check the images in docker

$ docker images



**Part 1: To pull and Push images in docker**

**Step 4:**

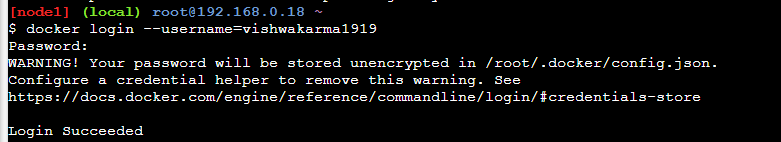
* Open the new tab in the browser and login to <hub.docker.com>
* Click on **Repositories** and then click on **Create Repositories**
* Give the name of the repository as “**repo1**” and in description add “**My first repository**”
* Make visibility as **Private**
* And now click on **Create** button and check whether the repository is created or not.

**Step 5:**

* Now come to the <https://labs.play-with-docker.com/> and give the following command
* Login into docker account

$ docker login -username= your\_user\_name

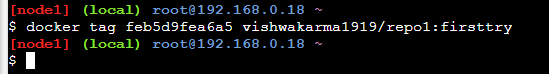
password:



Note: Give your username and password that you have used to login to <hub.docker.com>

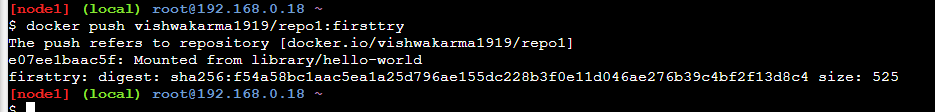
* To tag an image in docker

$ docker tag <image id> <username>/repo1: firsttry



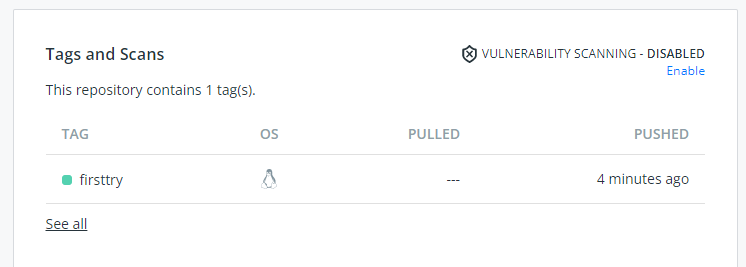
* To push the image to docker account

$ docker push <username>/repo1:firsttry



Note: firsttry is tag name created above.

* Check it in [hub.docker.com](file:///D:\Microservices%20Architecture\Practice%20Practical\hub.docker.com) now in tags tab

****

**Part 2: Build and image and then push and run in the docker**

**Step 6:**

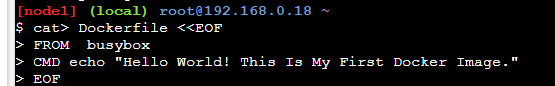
* In <https://labs.play-with-docker.com/> give the following command

cat > Dockerfile <<EOF

FROM busybox

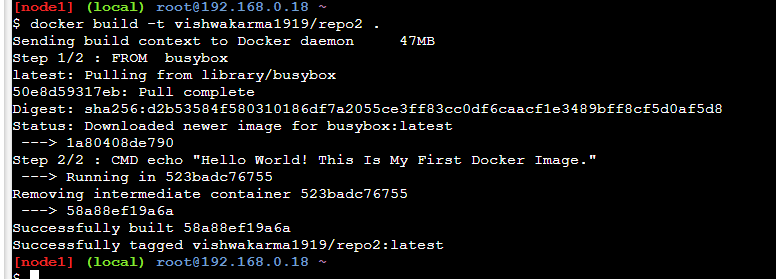
CMD echo "Hello world! This is my first Docker image."

EOF



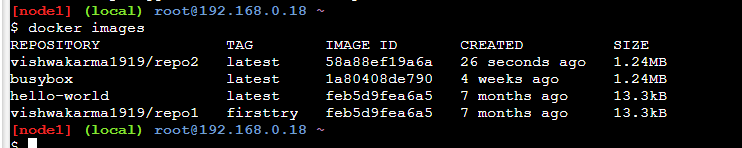
* To build the image from docker file

$ docker build –t <username>/repo2 .



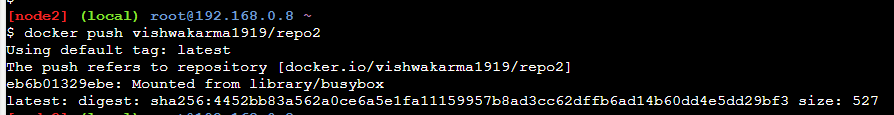
* Check images in docker

$ docker images

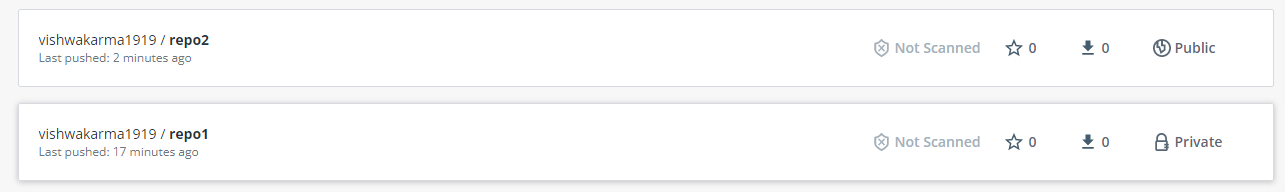
****

* To push the image on the docker hub

$ docker push <username>/repo2.

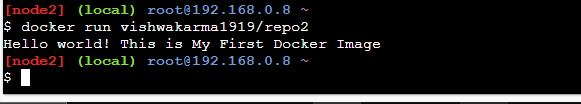


* Check it in [hub.docker.com](file:///D:\Microservices%20Architecture\Practice%20Practical\hub.docker.com) now in tags tab



* Come back to the <https://labs.play-with-docker.com/> and give the below command to run the docker image

$ docker run <username>/repo2



* Close the session

**Practical No 5**

**Aim:** Building ASP.Net core REST API

**Source Code:**

**Step 1**: Create a webAPI

Open command prompt and give the command

dotnet new webapi -o Glossary

Text

Description automatically generated

Now enter into the glossary folder and then run the project

Text

Description automatically generated

**Step 2**: Open another command prompt & give curl command to view the output

Text

Description automatically generated

**Step 3**: Delete the weatherforecast.cs from the Glossary Folder i.e root folder and also from the Controller Folder.

**Step 4**: Create a class file in the Glossary folder named “GlossaryItem.cs”

namespace Glossary

{

public class GlossaryItem

{

public string Term { get; set;}

public string Definition { get; set; }

}

}

**Step 5**: Create a class file in the Controller folder named “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();

}

}

}

**Step 6**: To stop the application running on command prompt do Ctrl+c

**Step 7:** Now restore, build and then run the program

**Text

Description automatically generated**

**Output:**

Open the other command prompt and give the following command.

Kindly note the port number that you will get in the previous command prompt and change the port number in the curl

1. Getting the List of Items

curl --insecure <https://localhost:7010/api/glossary>

1. Getting Single Item 
   1. curl --insecure <https://localhost:7010/api/glossary/MVC>
   2. curl --insecure <https://localhost:7010/api/glossary/HTML>
   3. curl --insecure <https://localhost:7010/api/glossary/OpenID>
2. Creating an item

curl --insecure -X POST -d "{\"term\": \"MFA\", \"definition\":\"An authentication process.\"}" -H "Content-Type:application/json" <https://localhost:7010/api/glossary>

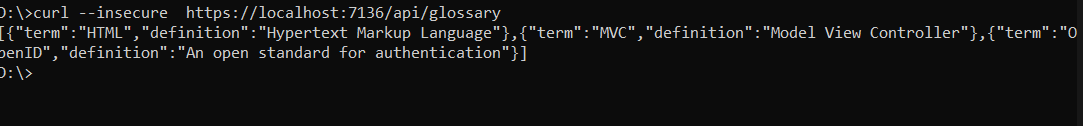
1. Updating an Item  
   curl --insecure -X PUT -d "{\"term\": \"MVC\", \"definition\":\"Modified record of Model View Controller.\"}" -H "Content-Type:application/json" <https://localhost:7010/api/glossary>
2. Delete an Item  
   curl --insecure --request DELETE --url <https://localhost:7010/api/glossary/openid>

**Output:-**

Open the other command prompt and give the following command.

Kindly note the port number that you will get in the previous command prompt and change the port number in the curl

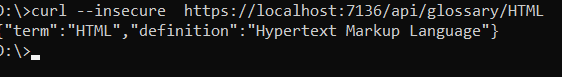
1. Getting the List of Items

curl --insecure <https://localhost:7010/api/glossary>  


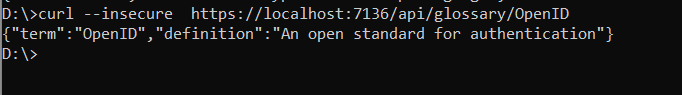
1. Getting Single Item
   1. curl --insecure <https://localhost:7010/api/glossary/MVC>

****

* 1. curl --insecure <https://localhost:7010/api/glossary/HTML>

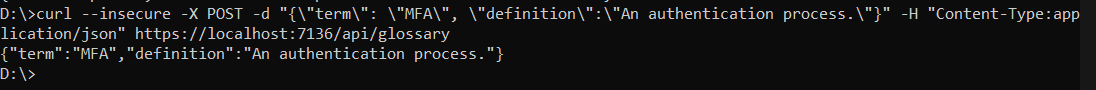
****

* 1. curl --insecure <https://localhost:7010/api/glossary/OpenID>

****

1. Creating an item

curl --insecure -X POST -d "{\"term\": \"MFA\", \"definition\":\"An authentication process.\"}" -H "Content-Type:application/json" <https://localhost:7010/api/glossary>



1. Updating an Item  
   curl --insecure -X PUT -d "{\"term\": \"MVC\", \"definition\":\"Modified record of Model View Controller.\"}" -H "Content-Type:application/json" <https://localhost:7010/api/glossary>

5. Delete an Item  
curl --insecure --request DELETE --url <https://localhost:7010/api/glossary/openid>

**Practical No 6**

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

**Steps and its output:**

**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.

Graphical user interface, text, application, email

Description automatically generated

**Step 2**: **Set up CircleCI**

* Login to Circle CI <https://app.circleci.com/> using GitHub Login
* Navigate to the CircleCI Projects page. If you created your new repository under an organization, you will need to select the organization name.

Graphical user interface, text, application, email

Description automatically generated

* You will be taken to the Projects dashboard. On the dashboard, select the project you want to set up (hello-world).
* 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.

Graphical user interface, application

Description automatically generated

**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-helloworkflow).
* Within this workflow, the pipeline ran one job, called say-hello. Click say-hello to see the steps in this job:
  + Spin up environment
  + Preparing environment variables
  + Checkout code
  + Say hello
* Now select the “say-hello-workflow”

A screenshot of a computer

Description automatically generated

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

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

* Select Branch and option circleci-project-setup

A screenshot of a computer

Description automatically generated

**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.

Graphical user interface, text, application, email, website

Description automatically generated

A screenshot of a computer

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

The GitHub file editor should look like this

Graphical user interface, text, email, website

Description automatically generated

Scroll down and Commit your changes on GitHub

Graphical user interface, text, application

Description automatically generated

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

A screenshot of a computer

Description automatically generated

**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.

* 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.

Text

Description automatically generated

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

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

Graphical user interface, text, application, email

Description automatically generated

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

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Step 6**: **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:

Text

Description automatically generatedText

Description automatically generated

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

Graphical user interface, text, application, email

Description automatically generated

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

Graphical user interface, text, application, email

Description automatically generated

**Practical No 7**

**Aim:** Working with TeamService

**Source Code:**

**Step 1:**

* Open command prompt and create a web api

Text

Description automatically generated

* Remove existing weatherforecast files both model and controller files.

**Step 2:**

* 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)

{

return this.Ok(team);

}else

{

return this.NotFound();

}

}

[HttpPost]

public virtual IActionResult CreateTeam([FromBody]Team newTeam)

{

repository.Add(newTeam);

return this.Created($"/teams/{newTeam.ID}", newTeam);

}

[HttpPut("{id}")]

public virtual IActionResult UpdateTeam([FromBody]Team team, Guid id)

{

team.ID = id;

if(repository.Update(team) == null)

{

return this.NotFound();

}

else

{

return this.Ok(team);

}

}

[HttpDelete("{id}")]

public virtual IActionResult DeleteTeam(Guid id)

{

Team team = repository.Delete(id);

if (team == null)

{

return this.NotFound();

}

else

{

return this.Ok(team.ID);

}

}

}

}

* Add MembersController.cs file to “D:\TeamService\Controllers” folder

using System;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Linq;

using TeamService.Models;

using System.Threading.Tasks;

using TeamService.Persistence;namespace TeamService

{

[Route("/teams/{teamId}/[controller]")]

public class MembersController : Controller

{

ITeamRepository repository;

public MembersController(ITeamRepository repo)

{

repository = repo;

}

[HttpGet]

public virtual IActionResult GetMembers(Guid teamID)

{

Team team = repository.Get(teamID);

if(team == null)

{

return this.NotFound();

}

else

{

return this.Ok(team.Members);

}

}

[HttpGet]

[Route("/teams/{teamId}/[controller]/{memberId}")]

public virtual IActionResult GetMember(Guid teamID, Guid memberId)

{

Team team = repository.Get(teamID);

if(team == null)

{

return this.NotFound();

}

else

{

var q = team.Members.Where(m => m.ID == memberId);

if(q.Count() < 1)

{

return this.NotFound();

}

else

{

return this.Ok(q.First());

}

}

}

[HttpPut]

[Route("/teams/{teamId}/[controller]/{memberId}")]

public virtual IActionResult UpdateMember([FromBody]Member updatedMember, Guid teamID, Guid memberId)

{

Team team = repository.Get(teamID);

if(team == null)

{

return this.NotFound();

}

else

{

var q = team.Members.Where(m => m.ID == memberId);

if(q.Count() < 1)

{

return this.NotFound();

}

else

{

team.Members.Remove(q.First());

team.Members.Add(updatedMember);

return this.Ok();

}

}

}

[HttpPost]

public virtual IActionResult CreateMember([FromBody]Member newMember, Guid teamID)

{

Team team = repository.Get(teamID);

if(team == null)

{

return this.NotFound();

}

else

{

team.Members.Add(newMember);

var teamMember = new {TeamID = team.ID, MemberID = newMember.ID};

return this.Created($"/teams/{teamMember.TeamID}/[controller]/{teamMember.MemberID}", teamMember);

}

}

[HttpGet]

[Route("/members/{memberId}/team")]

public IActionResult GetTeamForMember(Guid memberId)

{

var teamId = GetTeamIdForMember(memberId);

if (teamId != Guid.Empty)

{

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

}

else

{

return this.NotFound();

}

}

private Guid GetTeamIdForMember(Guid memberId)

{

foreach (var team in repository.List())

{

var member = team.Members.FirstOrDefault( m => m.ID == memberId);

if (member != null)

{

return team.ID;

}

}

return Guid.Empty;

}

}

}

**Step 3:**

* Create folder “D:\TeamService\Persistence”
* 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;

}

}

}

**Step 4:**

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

services.AddScoped<ITeamRepository, MemoryTeamRepository>();

**Output:**

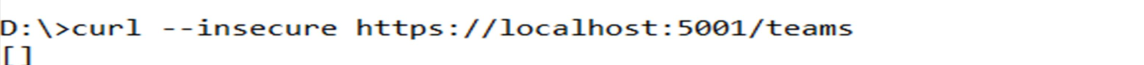
* Open two command prompt
* Command Prompt 1: go inside folder teamservice first

A screenshot of a computer

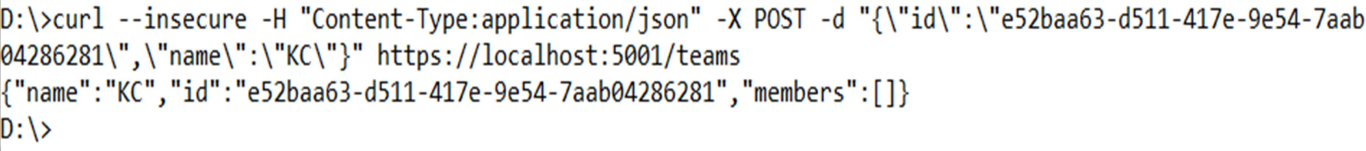
Description automatically generated with medium confidence

* On Command Prompt 2:

**To get all teams**curl --insecure <https://localhost:5001/teams>

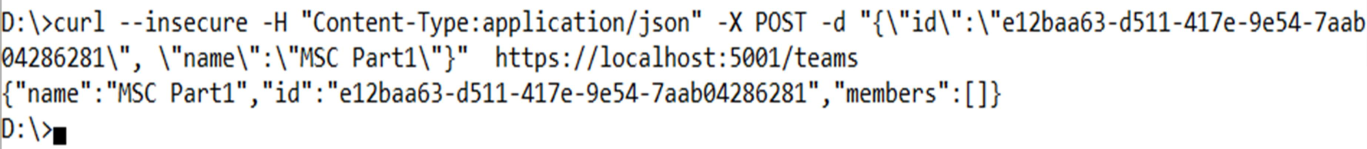


**To create new team**curl --insecure -H “Content-Type:application/json” –X POST –d “{\“id\”:\”e52baa63-d511-417e-9e54-7aab04286281\”, \”name\”:\”KC\”}” <https://localhost:5001/teams>



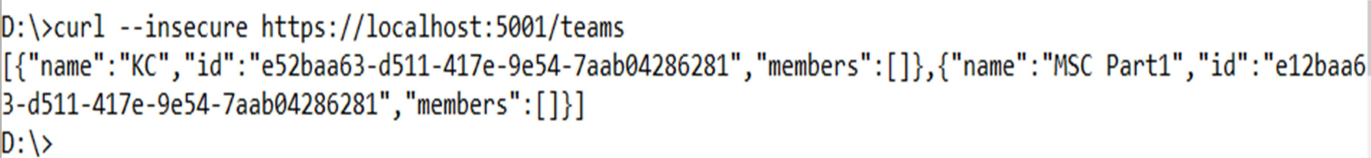
**To create one more new team**

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



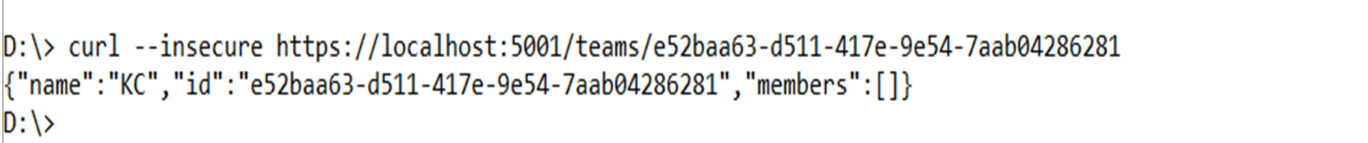
**To get all teams**

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



**To get single team with team-id as parameter**

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



**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>

Text

Description automatically generated

**To delete team**

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

A picture containing text

Description automatically generated

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

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

Text, letter

Description automatically generated