Q1. Install .NET Core and create real time microservice with ASP.Net

**SUM of 2 number**

using System;

namespace applicationname

{

Class program

{

**static** **void** Main(**string**[] args)

{

Console.WriteLine("Enter num1:");

int a=Convert.ToInt16(Console.ReadLine());

Console.WriteLine("Enter num2:");

int b=Convert.ToInt16(Console.ReadLine());

int sum =a+b;

Console.WriteLine($"The sum of {a} and {b} is {sum}");

}

}

}

**Calculate 2 no**

Using System;

namespace applicationname

{

Class program

{

**static** **void** Main(**string**[] args)

{

Console.WriteLine("Enter num1:");

int a=Convert.ToInt16(Console.ReadLine());

Console.WriteLine("Enter num2:");

int b=Convert.ToInt16(Console.ReadLine());

Console.WriteLine("Operation to perform + - / \*");

string opr=Console.ReadLine();

if(opr=="+"){

Console.WriteLine($"The multiplication of{a} and {b} is {a+b}");

}else if(opr=="-"){

    Console.WriteLine($"The subtraction of {a} and {b} is {a-b}");

}else if(opr=="/"){

    Console.WriteLine($"The division of {a} and {b} is {a/b}");

}else if(opr=="\*"){

    Console.WriteLine($"The multiplication of {a} and {b} is {a\*b}");

}else{

    Console.WriteLine("Enter a valid Number");

}

}

}

}

**Fibonacci**

int n1=0,n2=1,n3,i,number;

         Console.Write("Enter the number of elements: ");

         number = int.Parse(Console.ReadLine());

         Console.Write(n1+" "+n2+" "); //printing 0 and 1

         for(i=2;i<number;++i) //loop starts from 2 because 0 and 1 are already printed

         {

          n3=n1+n2;

          Console.Write(n3+" ");

          n1=n2;

          n2=n3;

}

**Q2.Building ASP.NET Core App**

**1. continue practical 1, Go to the folder myFirstApp and open the file Program.cs in notepad**

**and make the changes in the file as shown**

using System;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Logging;

using Microsoft.Extensions.Configuration;

using Microsoft.AspNetCore.Http;

using System.Configuration;

namespace myFirstConsoleApp

{

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();

}

}

}

**2. Go to the folder myFirstApp and create the file Startup.cs in notepad and type the code**

**as shown**

using System;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Logging;

using Microsoft.Extensions.Configuration;

using Microsoft.AspNetCore.Http;

using System.Configuration;

namespace myFirstConsoleApp

{

public class Startup

{

public Startup(IHostingEnvironment env)

{

}

public void Configure(IApplicationBuilder app,IHostingEnvironmentenv,ILoggerFactory lf)

{

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

}

}

}

**3. Go to the command prompt and execute the following commands.**

dotnet add package Microsoft.AspNetCore.Mvc

dotnet add package Microsoft.AspNetCore.Server.Kestrel

dotnet add package Microsoft.Extensions.Logging

dotnet add package Microsoft.Extensions.Logging.Console

dotnet add package Microsoft.Extensions.Logging.Debug

dotnet add package Microsoft.Extensions.Configuration.CommandLine

**4. rename myfirstconsolapp to myFirstApp.csproj.**

**5. Go to the command prompt and execute the following commands.**

**1. dotnet restore**

**2. dotnet build**

**3. dotnet run.**

Q6. Install .NET Core and create real time microservice with ASP.Net

🡪login to github 🡪click on + icon then click create repo🡪select private🡪repo name as hello –word🡪add description--.then click on add readme file🡪then click on create repo

To create a brance

Click on the main in left hand side->then enter readme-edits🡪click on create brances

Q7. Working with Circle CI for continuous integration

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 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 circleciproject-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.

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 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 # 2) Commit these changes to your repository and navigate back to the CircleCI Pipelines page. You should see your pipelinerunning 3) 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 Finally your workflow with the jobs running should look like this

**Q5. Install Docker Desktop on Windows, verify installation create your first repository.**

**1.** If your admin account is different to your user account, you must add the user to the dockerusers group. Run Computer Management as an administrator and navigate to Local Users and Groups > Groups > docker-users. Right-click to add the user to the group. Log out and log back in for the changes to take effect.

Step 1: Sign up for a Docker account.

Step 2: Create your first repository To create a repository:

1. Sign in to Docker Hub. 2. Click Create a Repository on the Docker Hub welcome page. 3. Name it /my-private-repo. 4. Set the visibility to Private.

**Q5.Built a Docker container image on your computer and pushed it to Docker Hub.**

**1.** Download and install Docker Desktop.

**Step 2: Build and push a container image to Docker Hub from your computer**

1. Start by creating a Dockerfile to specify your application as shown below: 2. # syntax=docker/dockerfile:1 3. FROM busybox 4. CMD echo "Hello world! This is my first Docker image." 5. Run docker build -t /my-private-repo . to build your Docker image. 6. Run docker run /my-private-repo to test your Docker image locally. 7. Run docker push /my-private-repo to push your Docker image to Docker Hub.

Signed up for a Docker account • Created your first repository • Built a Docker container image on your computer • Pushed it successfully to Docker Hub