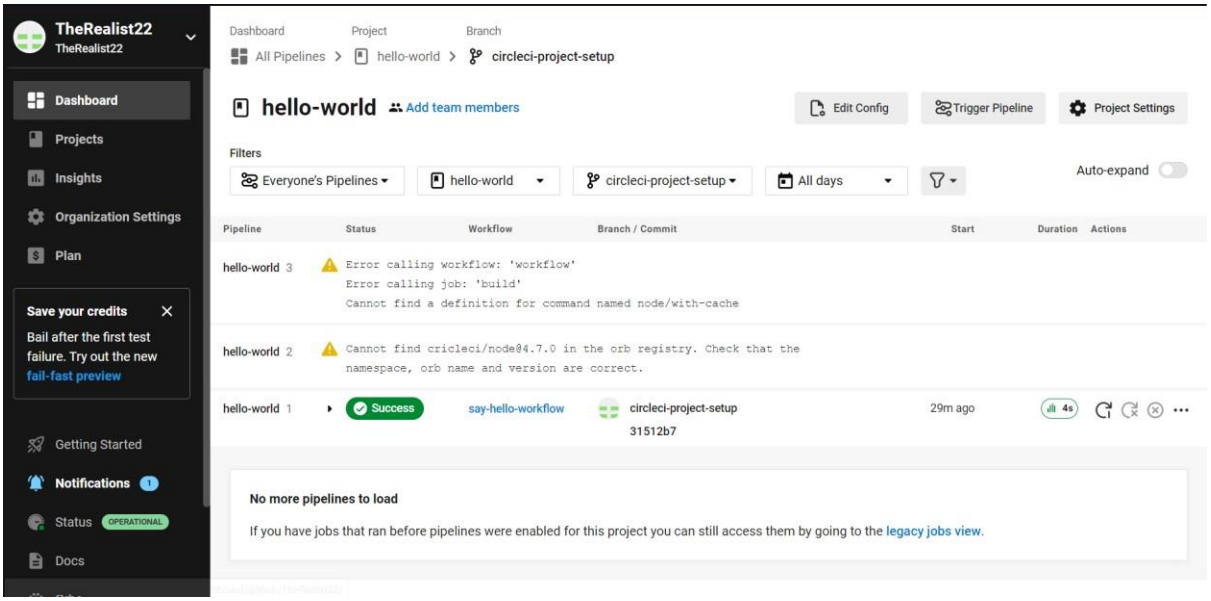


Step 5: Use Workflows

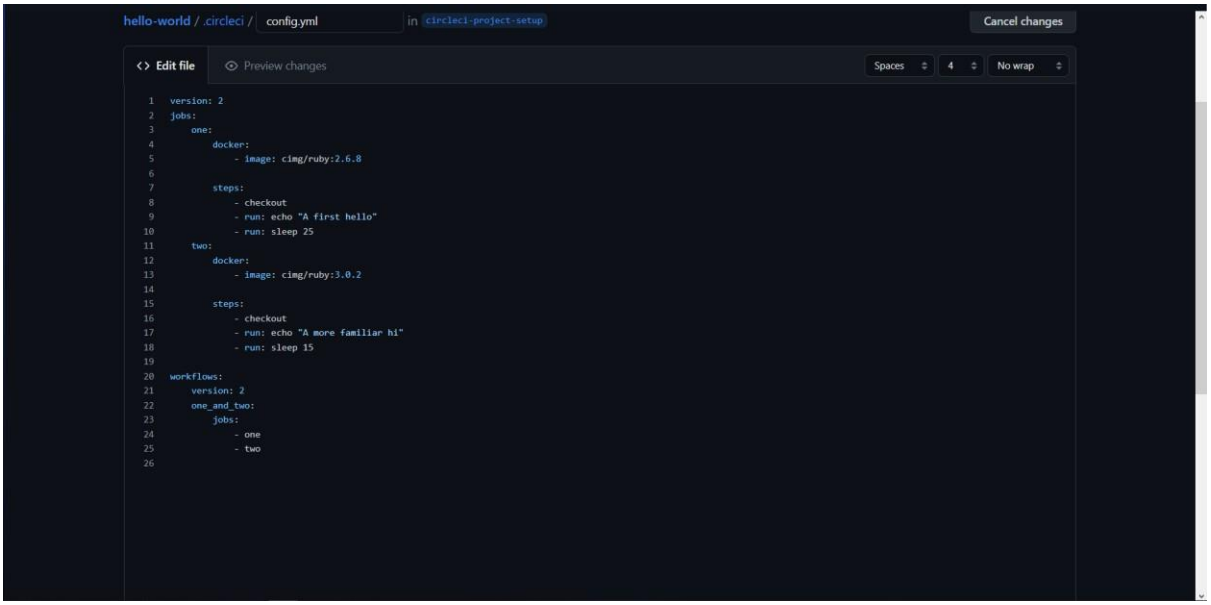
- You do not have to use orbs to use CircleCI. The following example details how to create a custom configuration that also uses the workflow feature of CircleCI.
- Take a moment and read the comments in the code block below. Then, to see workflows in action, edit your .circleci/config.yml file and copy and paste the following text into it.

Code:

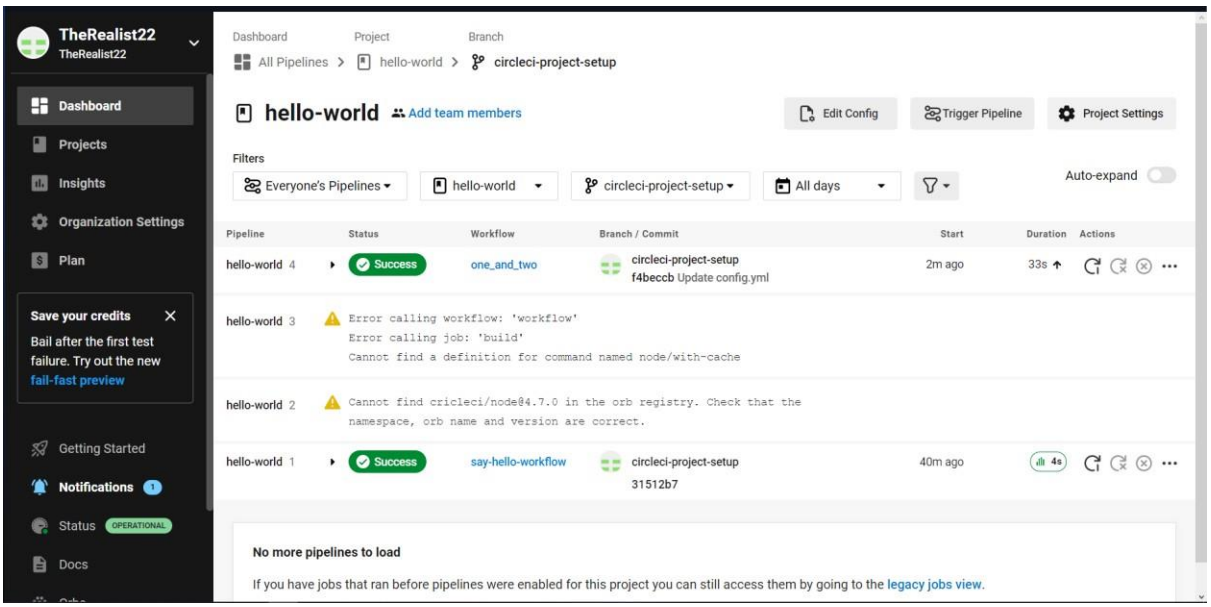
```
version: 2
jobs:
    one:
        docker:
            - image: cimg/ruby:2.6.8

        steps:
            - checkout
            - run: echo "A first hello"
            - run: sleep 25
    two:
        docker:
            - image: cimg/ruby:3.0.2

        steps:
            - checkout
            - run: echo "A more familiar hi"
            - run: sleep 15

workflows:
    version: 2
    one_and_two:
        jobs:
```

```
            - one
            - two
```

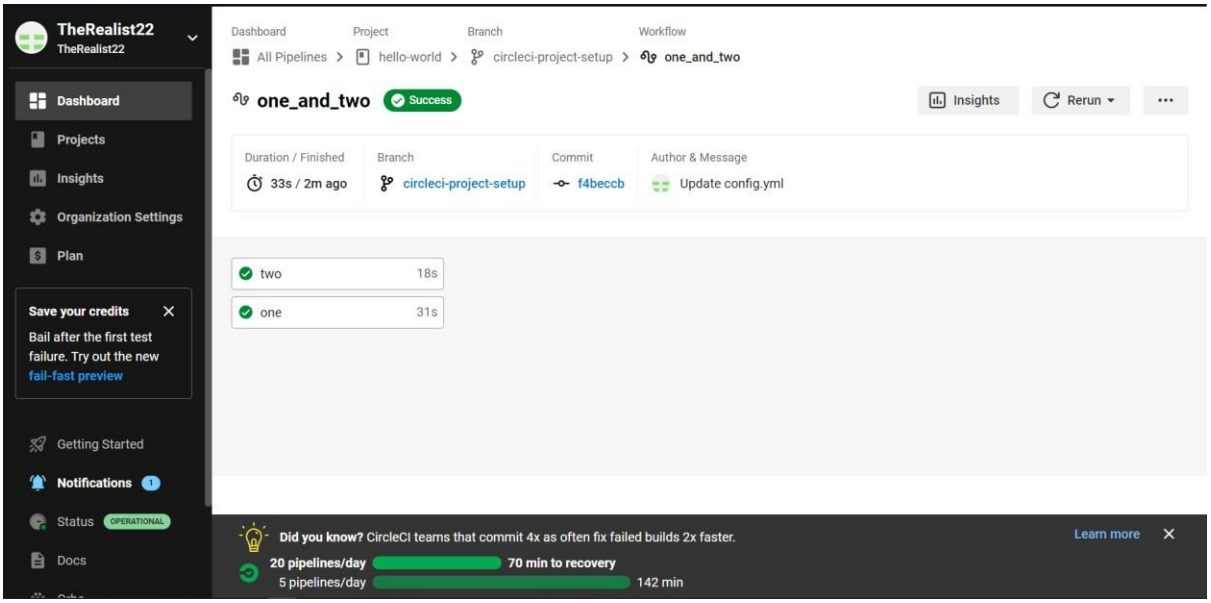
- Commit these changes to your repository and navigate back to the CircleCI Pipelines page. You should see your pipeline running.



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



- Click on **one_and_two** workflow.



Step 6: Add some changes to use workspaces.

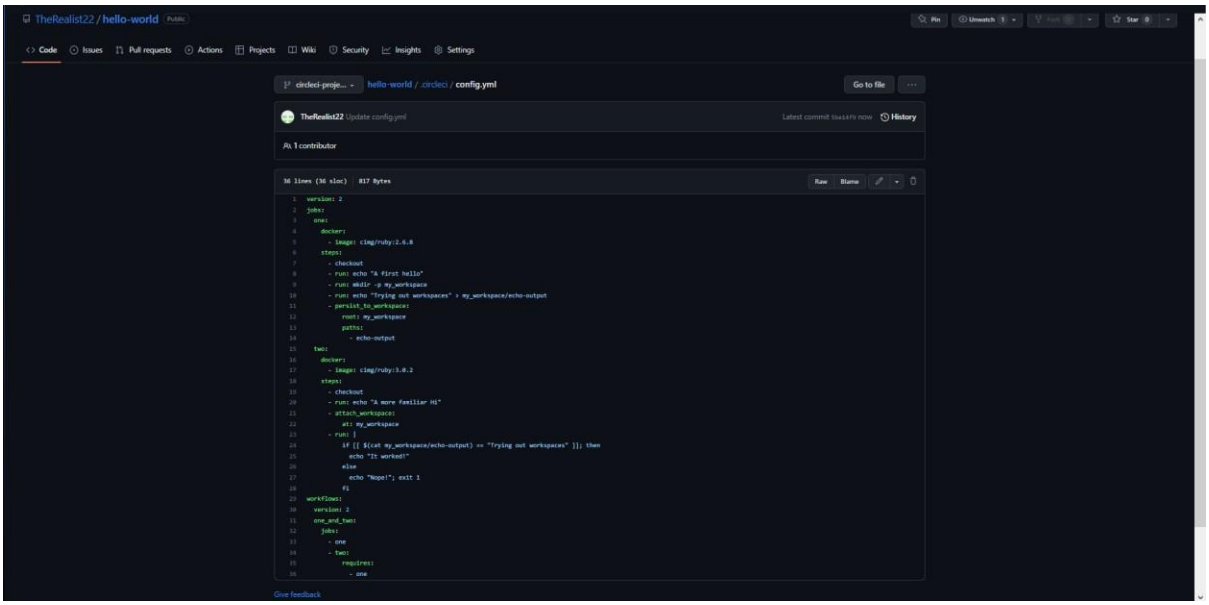
- Each workflow has an associated workspace which can be used to transfer files to downstream jobs as the workflow progresses.
- You can use workspaces to pass along data that is unique to this run and which is needed for downstream jobs.
- Try updating config.yml to the following:

Code:

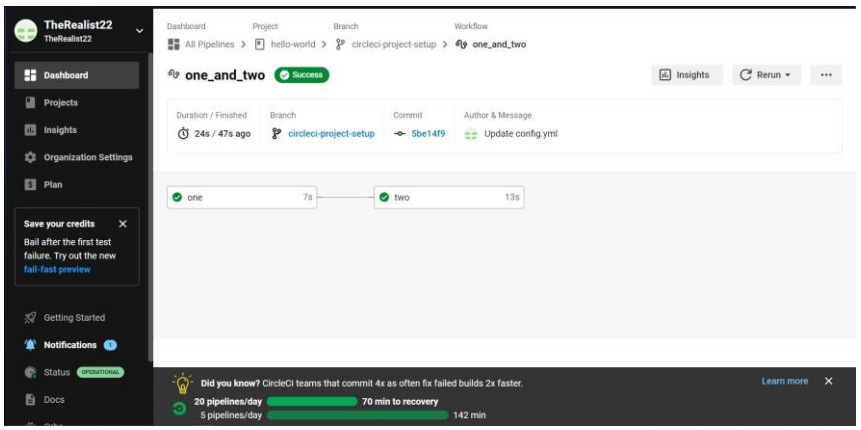
```
version: 2
jobs:
    one:
        docker:
            - image: cimg/ruby:3.0.2

        steps:
            - checkout
            - run: echo "A first hello"
            - run: mkdir -p my_workspace
            - run: echo "Trying out workspaces" > my_workspace/echo-output
            - persist_to_workspace:
                root: my_workspace

                paths:
                    - echo-output

    two:
        docker:
            - image: cimg/ruby:3.0.2

        steps:
            - checkout
            - run: echo "A more familiar hi"
```

```
            - attach_workspace:
                at: my_workspace

            - run: |
                if [[ $(cat my_workspace/echo-output) == "Trying out
workspaces"]]; then
                    echo "It worked!";

workflows:
    version: 2
    one_and_two:
        jobs:
            - one
            - two:
                requires:
                    - one
```

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



- Finally, your workflow with the jobs running should look like this



Practical 6

Aim: Creating Microservice with ASP.NET Core.

Writeup:

Step 1: Create new project.

Command: **dotnet new webapi -o TeamService**

```
Command Prompt
Microsoft Windows [Version 10.0.19044.2728]
(c) Microsoft Corporation. All rights reserved.

C:\Users\raman>dotnet new webapi -o TeamService
The template "ASP.NET Core Web API" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on TeamService\TeamService.csproj...
  Determining projects to restore...
  Restored C:\Users\raman\TeamService\TeamService.csproj (in 124 ms).

Restore succeeded.


C:\Users\raman>
```

Step 2: Remove existing weatherforecast files both model and controller files.

Step 3: Add new files as follows:

[A] : 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;
        }
    }
}
```

[B] : 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;
        }
    }
}
```

[C] : 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);
            }
        }
    }
}
```

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

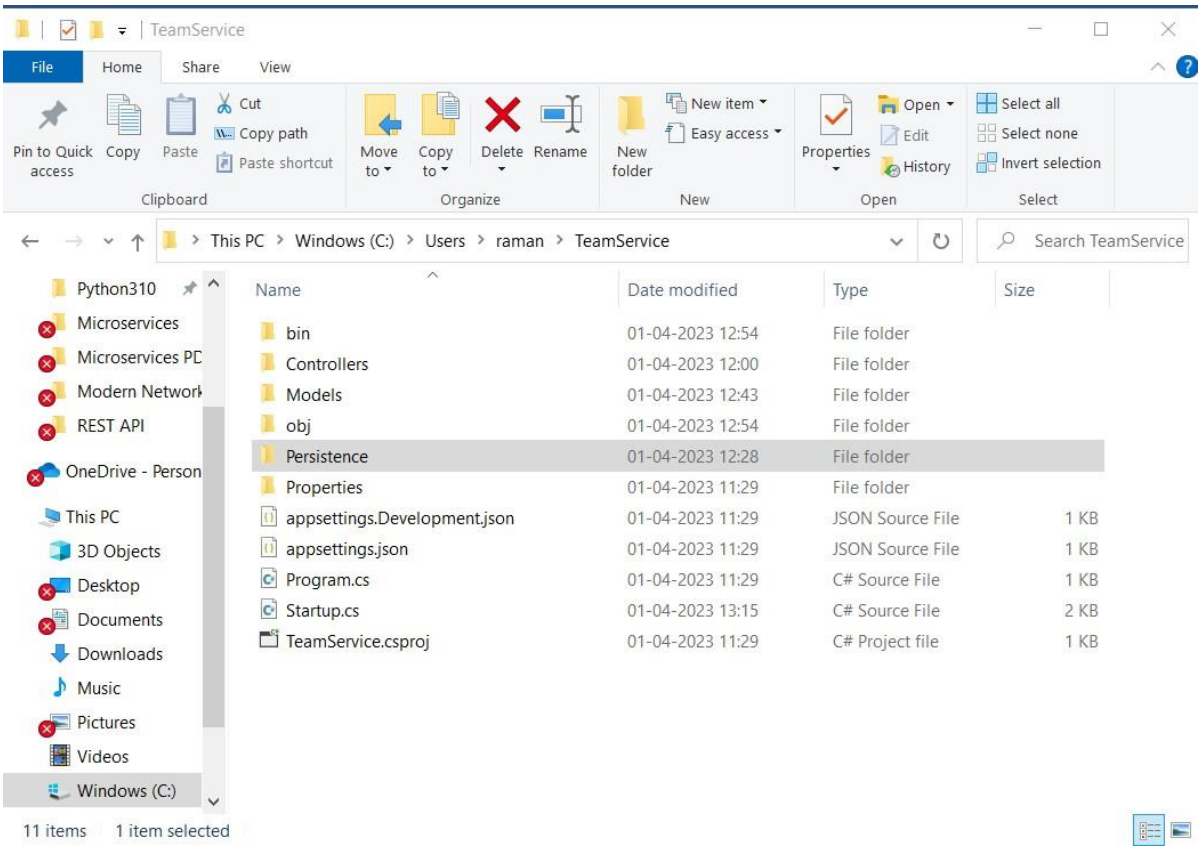
```
using System;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Linq;
using TeamService.Models;
using System.Threading.Tasks;
using TeamService.Persistence;

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 4: Create folder “D:\TeamService\Persistence”.



Step 5: Add file ITeamReposiroty.cs in “D:\TeamService\Persistence” folder.

```
using System;
using System.Collections.Generic;
using TeamService.Models;

namespace TeamService.Persistence
{
    public interface ITeamRepository
    {
        IEnumerable <Team> List();
        Team Get(Guid id);
        Team Add(Team team);
        Team Update(Team team);
        Team Delete(Guid id);
    }
}
```

Step 6: Add MemoryTeamRepository.cs in “D:\TeamService\Persistence” folder

```
using System;
using System.Collections.Generic;
using System.Linq;
using TeamService;
using TeamService.Models;

namespace TeamService.Persistence
{
    public class MemoryTeamRepository : ITeamRepository
    {
        protected static ICollection<Team> teams;
        public MemoryTeamRepository()
        {
            if(teams == null)
            {
                teams = new List<Team>();
            }
        }


        public MemoryTeamRepository(ICollection<Team> teams)
        {
            MemoryTeamRepository.teams = teams;
        }

        public IEnumerable<Team> List()
        {
            return teams;
        }
```

```
        public Team Get(Guid id)
        {
            return teams.FirstOrDefault(t => t.ID == id);
        }

        public Team Update(Team t)
        {
            Team team = this.Delete(t.ID);
            if(team != null)
            {
                team = this.Add(t);
            }

            return team;
        }

        public Team Add(Team team)
        {
            teams.Add(team);
            return team;
        }

        public Team Delete(Guid id)
        {
            var q = teams.Where(t => t.ID == id);
            Team team = null;
            if(q.Count() > 0)
            {
                team = q.First();
                teams.Remove(team);
            }

            return team;
        }
    }
}
```

Step 7: Add following line to Startup.cs in public void ConfigureServices(IServiceCollection services) method.

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

Step 8: Now open two command prompts to run this project.

Step 9: On Command prompt 1 (go inside folder teamservice first)

Commands:

dotnet run

Command Prompt - dotnet run

```
C:\Users\raman\TeamService>dotnet restore
  Determining projects to restore...
  All projects are up-to-date for restore.

C:\Users\raman\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: C:\Users\raman\TeamService
```

Step 10: On command prompt 2
Command: To get all teams

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

Command Prompt

```
Microsoft Windows [Version 10.0.19044.2728]
(c) Microsoft Corporation. All rights reserved.

C:\Users\raman>curl --insecure https://localhost:5001/teams
[]
C:\Users\raman>
```

Step 11: On command prompt 2
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

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

Step 12: On command prompt 2
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

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

Step 13: On command prompt 2
Command : To get all teams

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

```
Command Prompt
C:\Users\raman>curl --insecure https://localhost:5001/teams
[{"name":"KC","id":"e52baa63-d511-417e-9e54-7aab04286281","members":[]},{"name":"MSC Part1","id":"e12baa63-d511-417e-9e54-7aab04286281","members":[]}]
C:\Users\raman>
```

Step 14: On command prompt 2
Command : To get single team with team-id as parameter

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

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

Step 15: On command prompt 2
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

```
Command Prompt
C:\Users\raman>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-9e54-7aab0
4286281
{"name":"KC IT DEPT","id":"e52baa63-d511-417e-9e54-7aab04286281","members":[]}
C:\Users\raman>
```

Step 16: On command prompt 2
Command : To delete team

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

Command Prompt

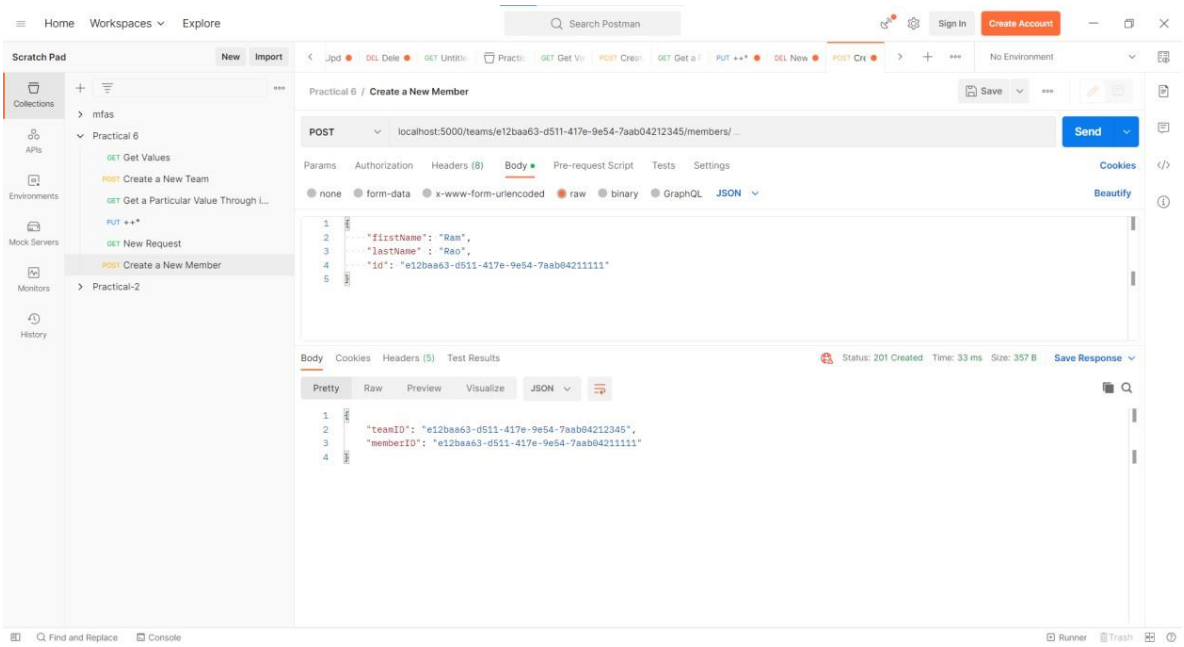
```
C:\Users\raman>curl --insecure -H "Content-Type:application/json" -X DELETE https://localhost:5001/teams/e52baa63-d511-417e-9e54-7aab04286281
"e52baa63-d511-417e-9e54-7aab04286281"
C:\Users\raman>
```

Step 17: Confirm with get all teams now it shows only one team (first one is deleted)
Command: **curl –insecure https://localhost:5001/teams**

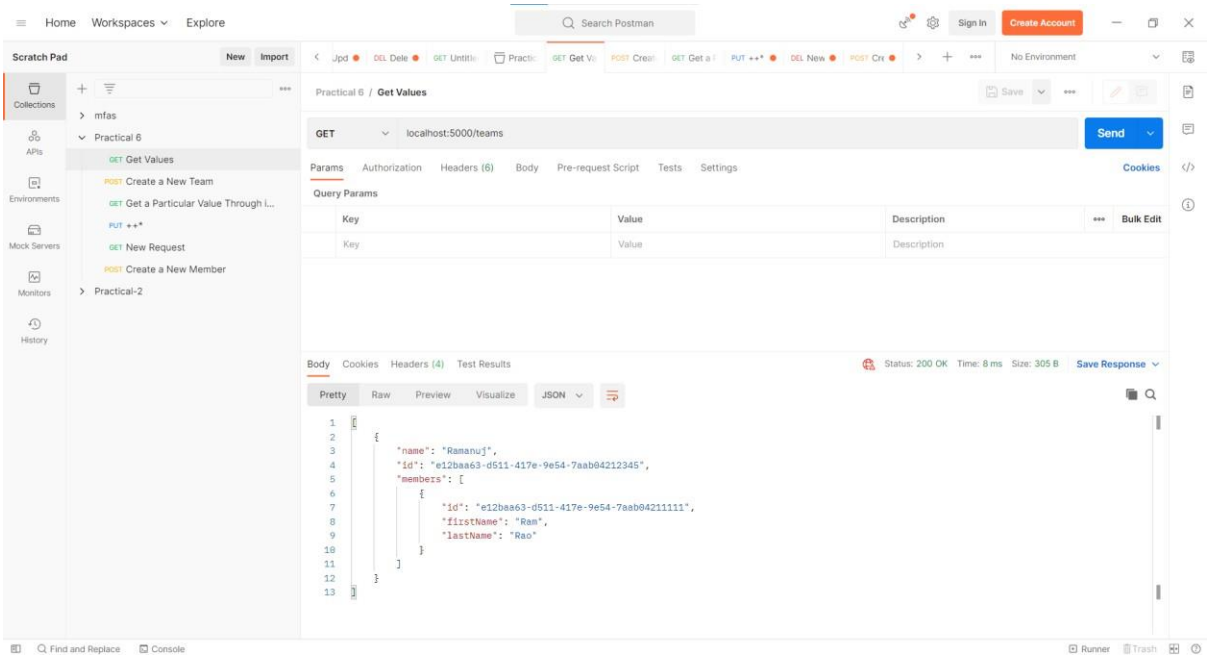
Command Prompt

```
C:\Users\raman>curl -insecure https://localhost:5001/teams
HTTP/1.1 200 OK
Date: Sat, 01 Apr 2023 08:43:18 GMT
Content-Type: application/json; charset=utf-8
Server: Kestrel
Transfer-Encoding: chunked

[{"name":"MSC Part1","id":"e12baa63-d511-417e-9e54-7aab04286281","members":[]}]
C:\Users\raman>
```

Step 18: Adding Members to the team.
Firstly, recreate the team which has been deleted by running the previous command for create in team. Within the created team we will add new properties for "members" within the team with a specific name and id. (Done using Postman)



- Confirm the creation of the member.

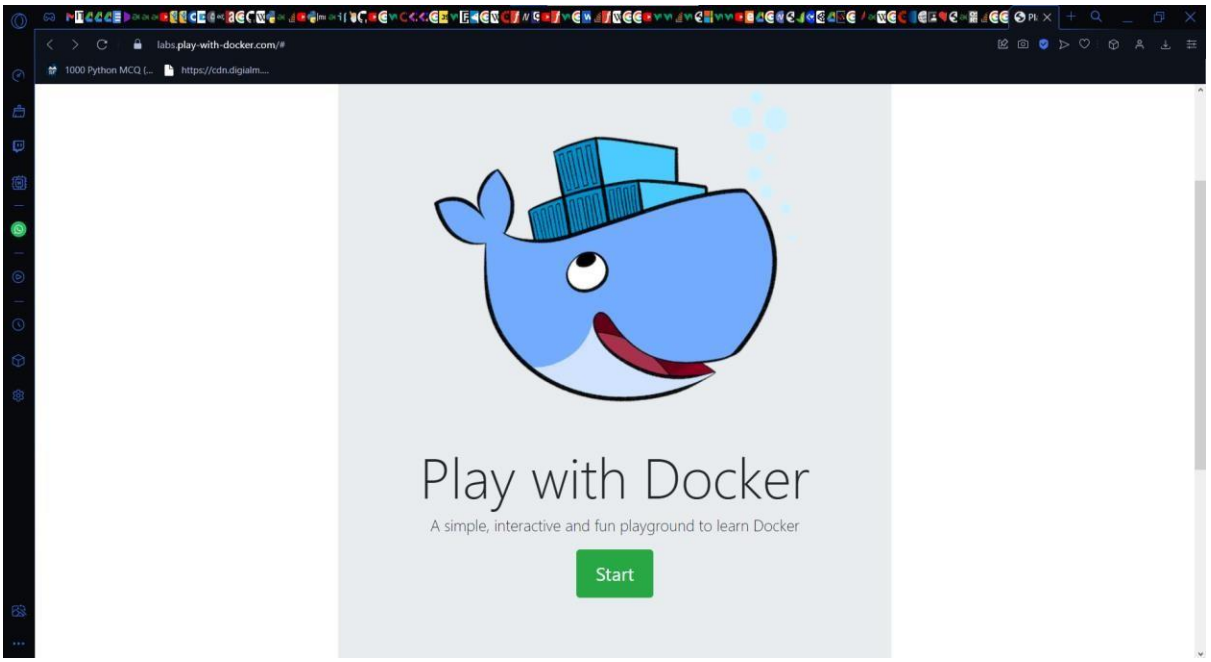


Practical 7

Aim: Creating Backing Service with Docker.

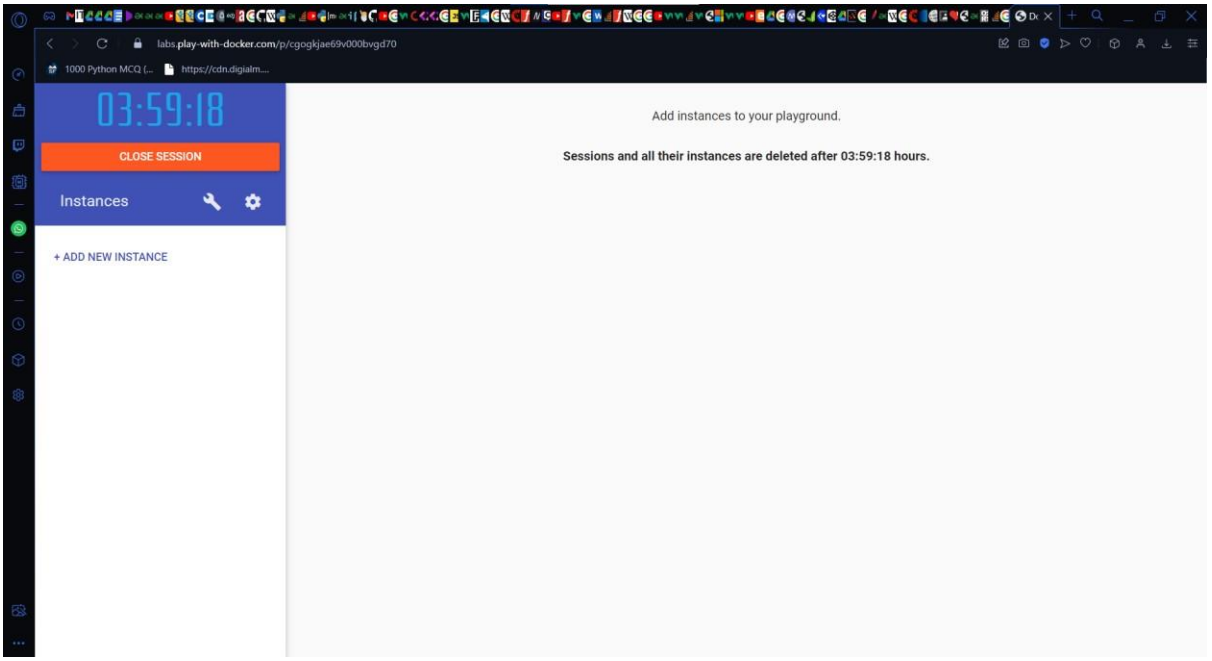
Writeup:

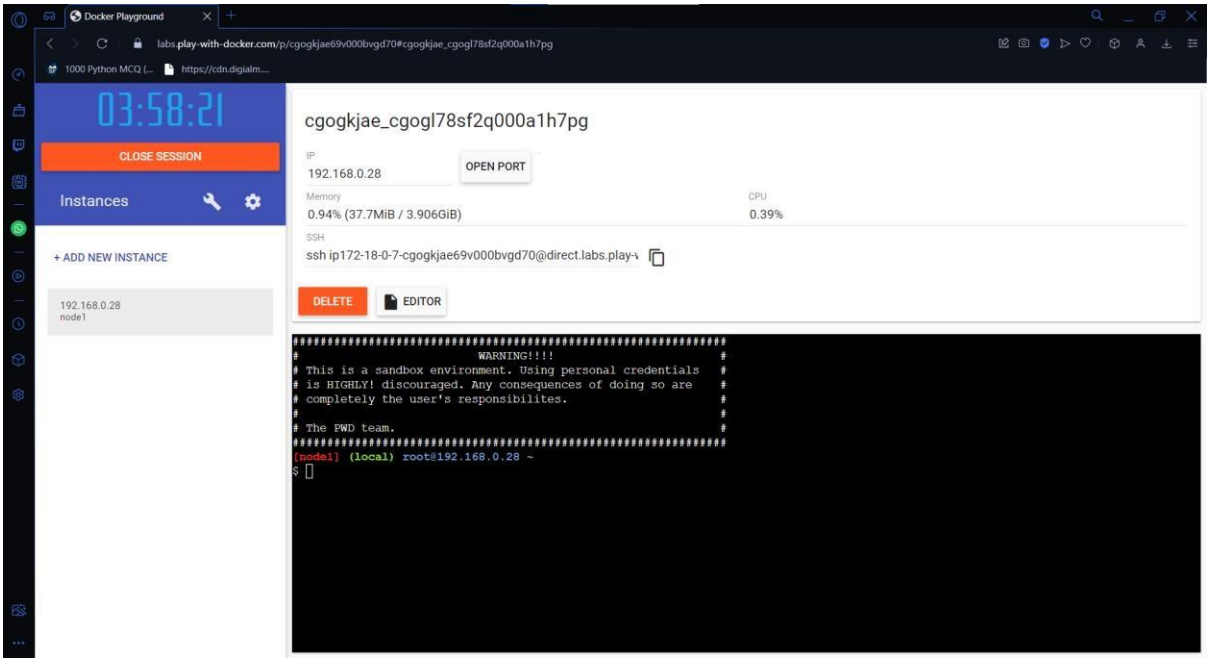
Step 1: Create docker hub login first to use it in play with docker.



- Click on Start

Step 2: Click on Add new instance



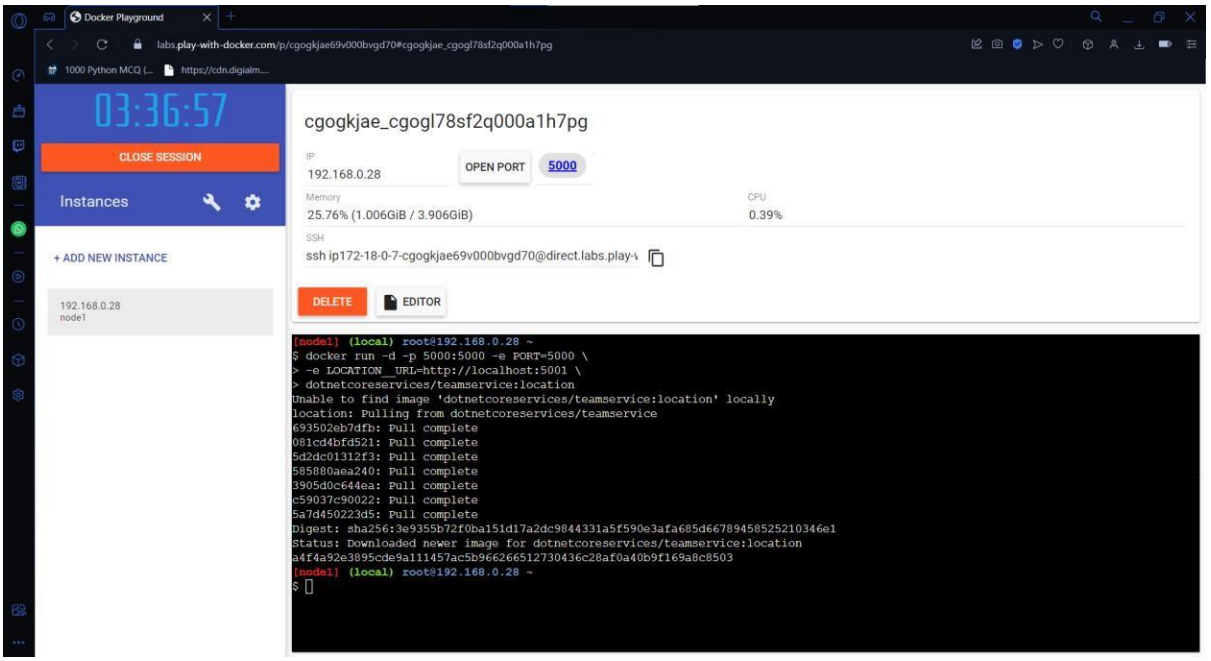


Step 3: Start typing the following command

Command: To run TeamService

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

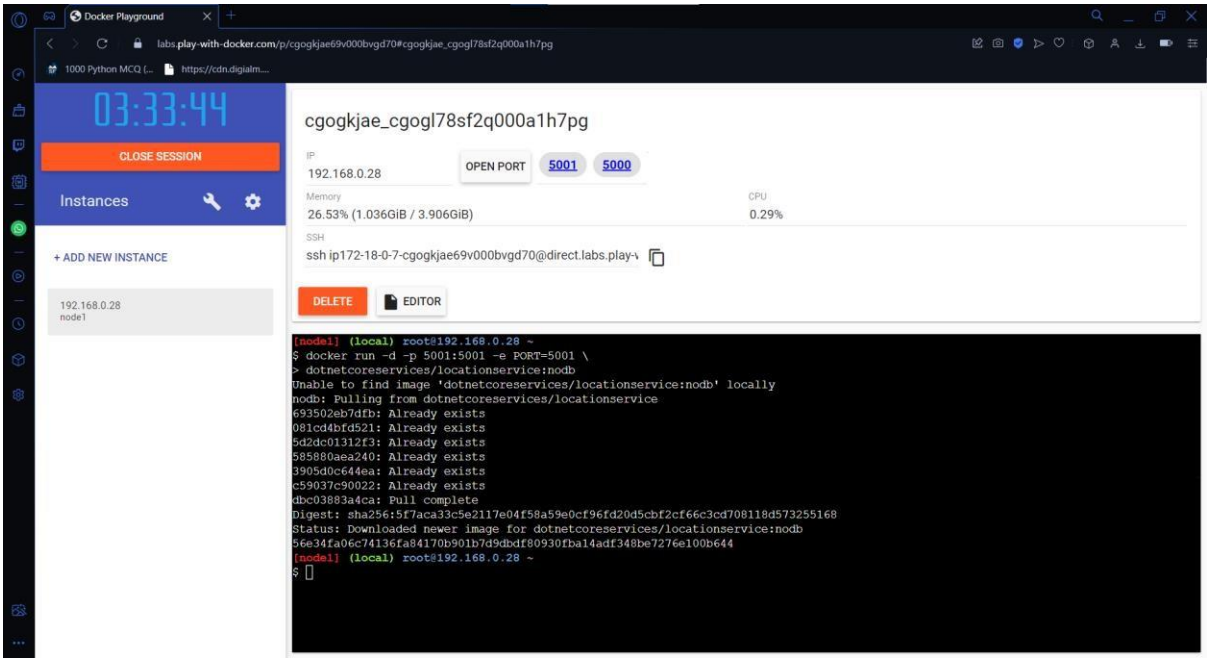
Output:



Command: To run Location Service

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

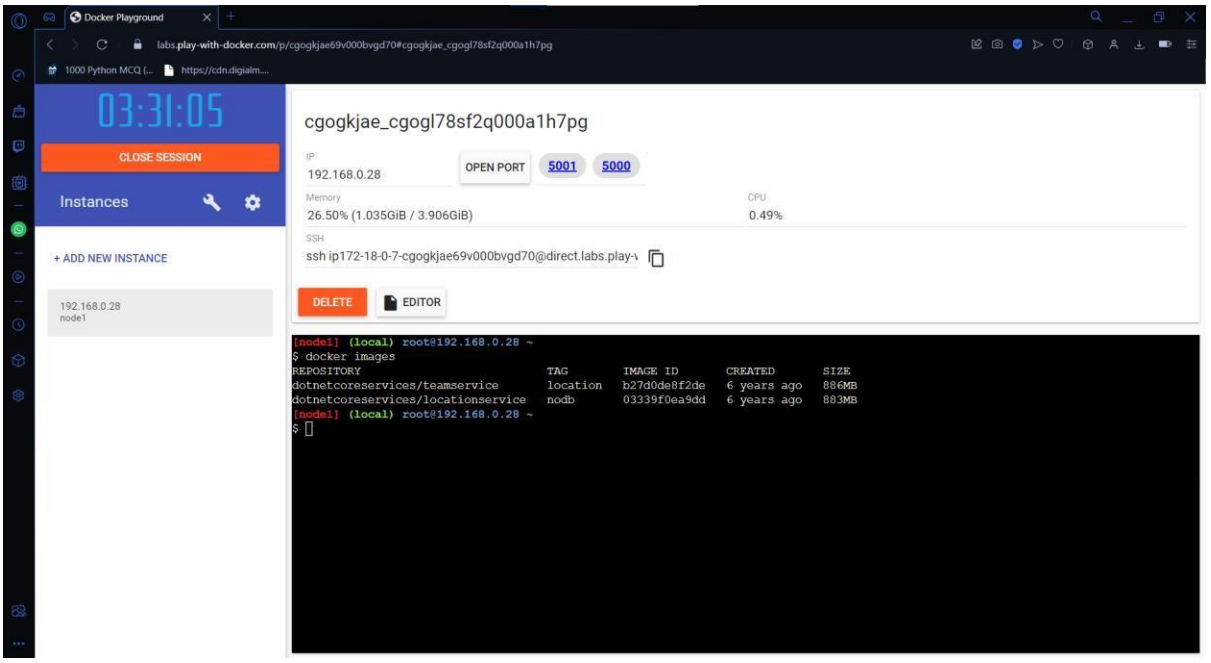
Output:



Command: To check running images in docker

docker images

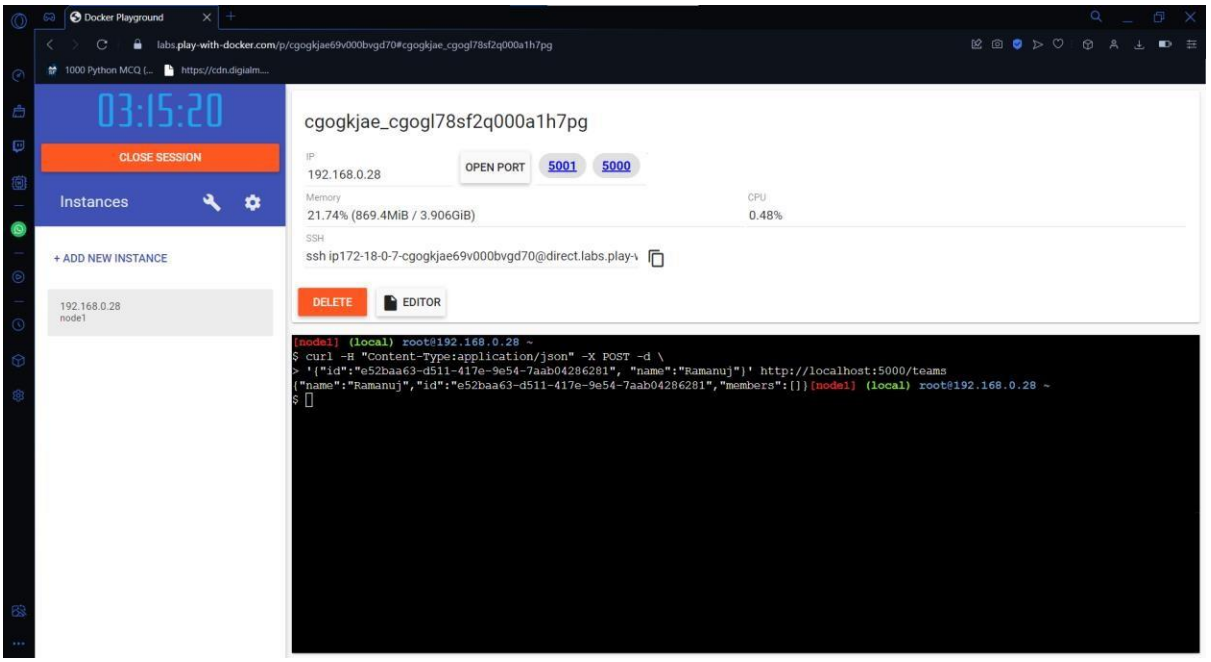
Output:



Command: To create a new team

Output:

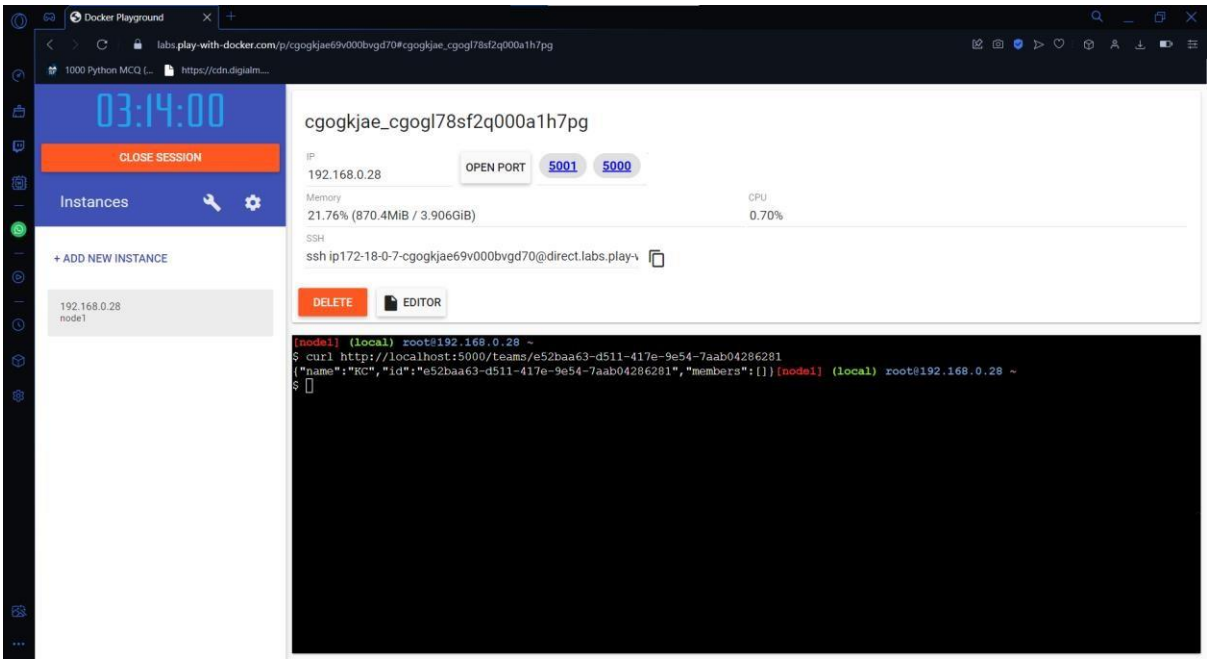
**curl -H "Content-Type:application/json" -X POST -d \
'{"id":"e52baa63-d511-417e-9e54-7aab04286281", "name":"Ramanuj"}'
http://localhost:5000/teams**



Command: To confirm if member is added

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

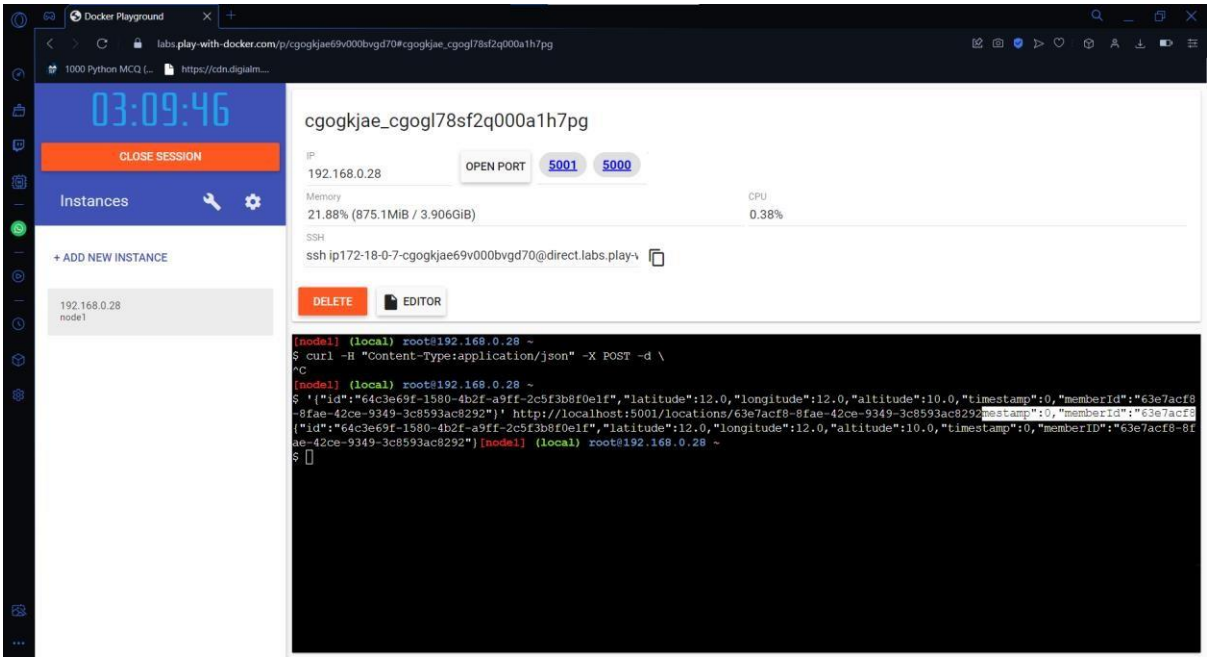
Output:



Command: To add location for member

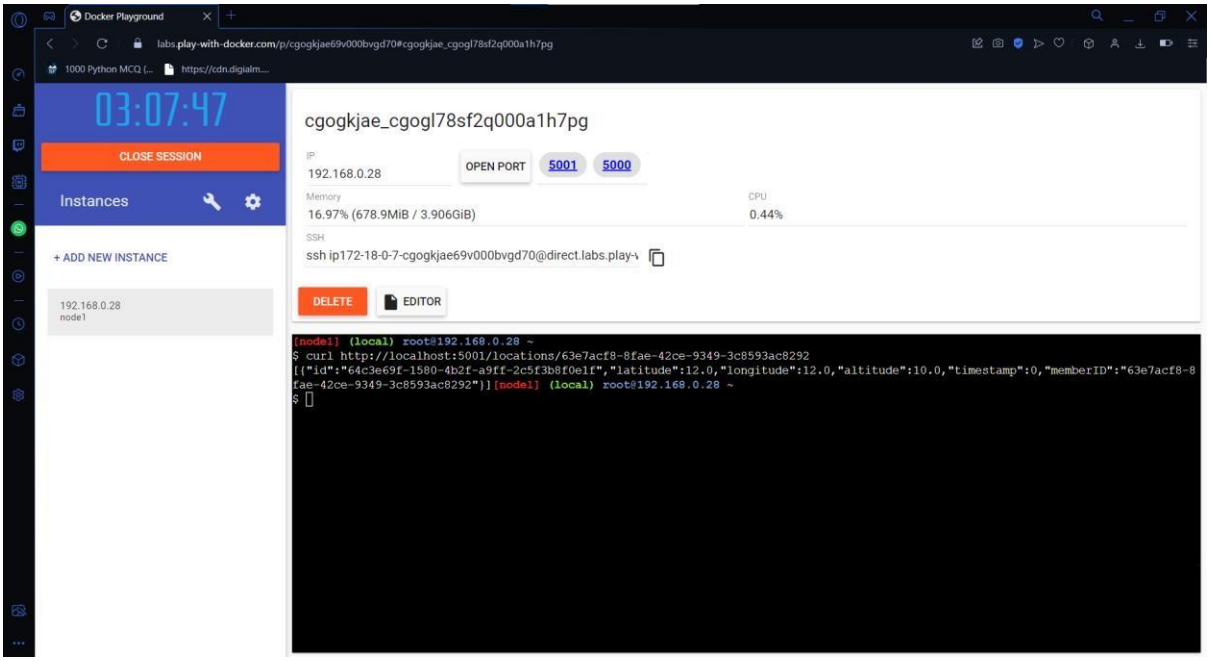
curl -H "Content-Type:application/json" -X POST -d \ '{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f","latitude":12.0,"longitude":12.0,"altitude":10.0,"timestamp":0,"memberId":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}' http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292

Output:



Command: To confirm location is added in member

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



Practical 8

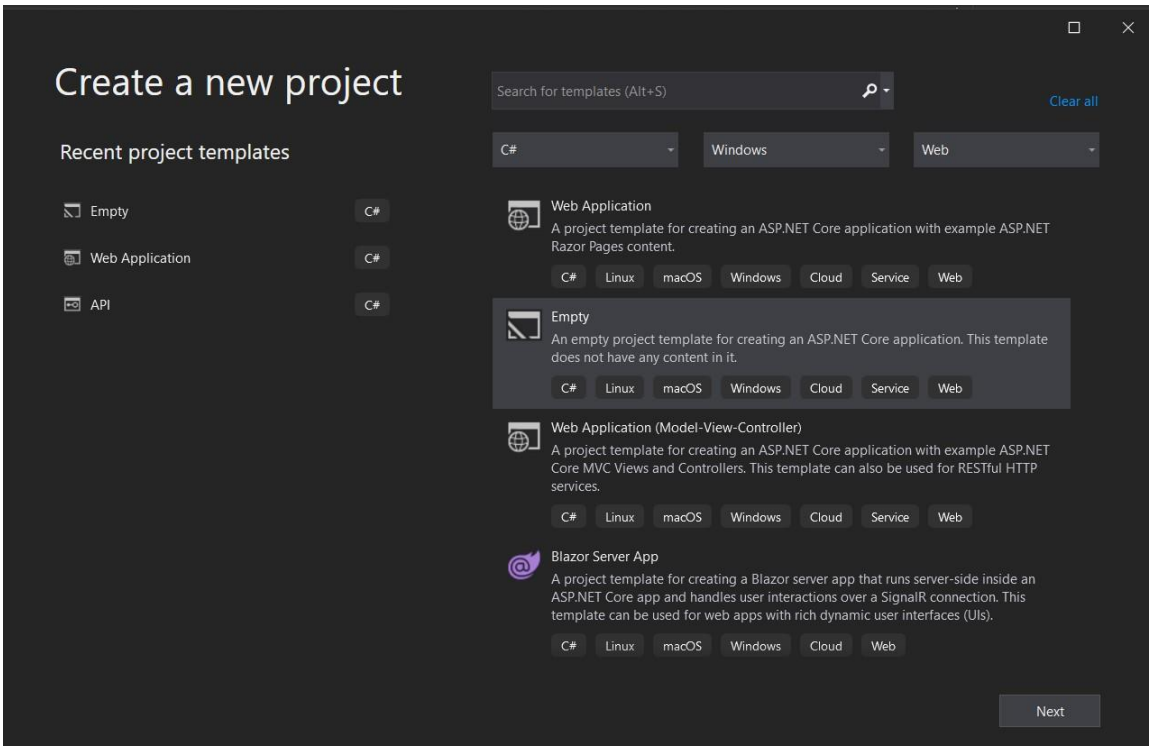
Aim: Building real-time Microservice with ASP.NET Core.

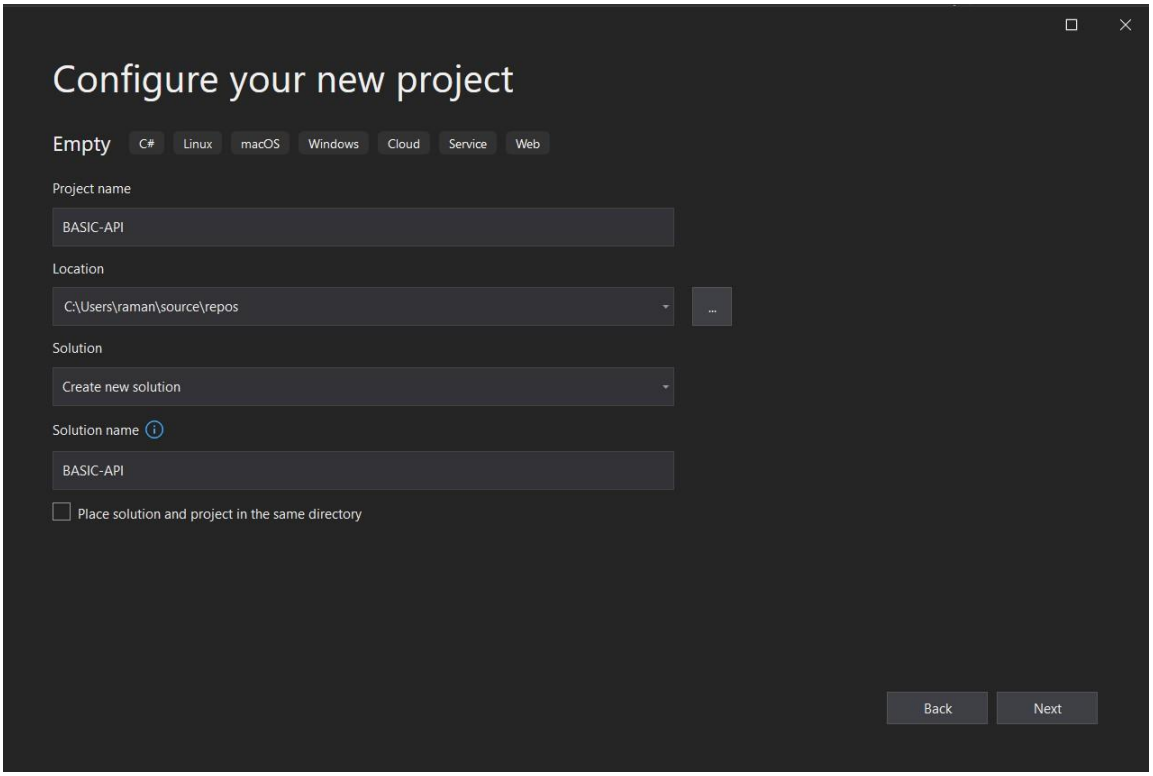
Writeup:

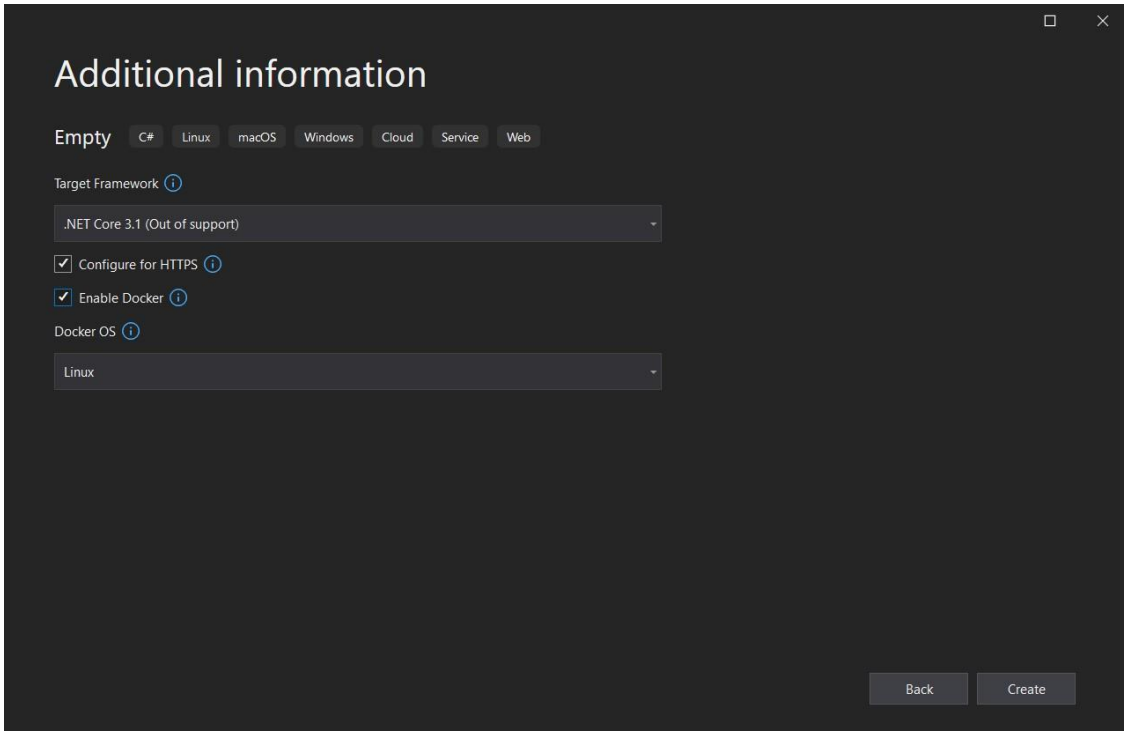
Requirement:

- Microsoft Visual Studio 2019 or higher
- ASP.Net Core 3.1

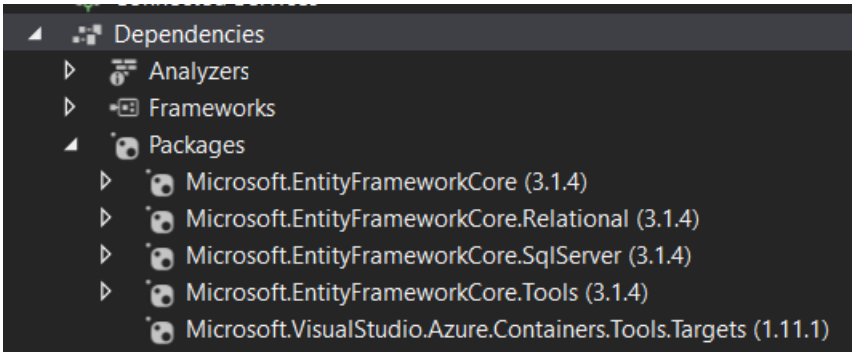
Step 1: Create and Configure a new empty project



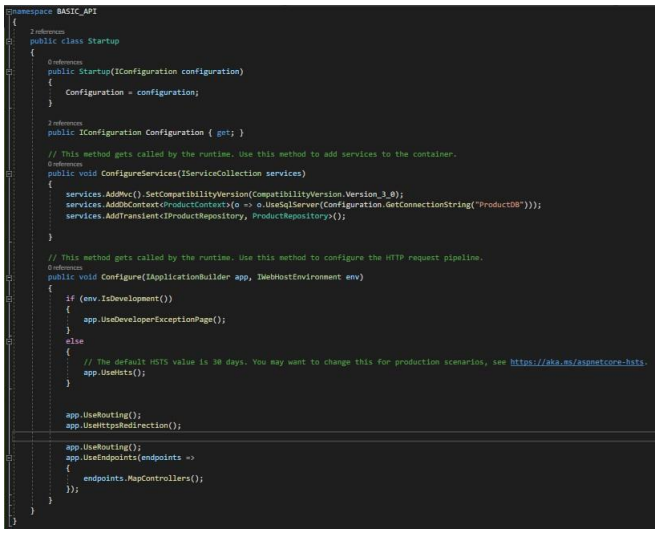




Step 2: Make sure to add these packages via NuGet Package Manager.



Step 3: Configure the Startup.cs file



```
namespace BASIC_API
{
    2 references
    public class Startup
    {
        0 references
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        2 references
        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        0 references
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_3_0);
            services.AddDbContext<ProductContext>(o => o.UseSqlServer(Configuration.GetConnectionString("ProductDB")));
            services.AddTransient<IProductRepository, ProductRepository>();

        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        0 references
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
                app.UseHsts();
            }


            app.UseRouting();
            app.UseHttpsRedirection();

            app.UseRouting();
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
}
```

Step 4: Create the Category Model, which will eventually be the Category Table.

```
namespace BASIC_API.Models
{
    5 references
    public class Category
    {
        3 references
        public int Id
        {
            get;
            set;
        }

        3 references
        public string Name
        {
            get;
            set;
        }

        3 references
        public string Description
        {
            get;
            set;
        }
    }
}
```

Step 5: Create the Product Model which will eventually be the Product Table.

```
using System.ComponentModel.DataAnnotations.Schema;

namespace BASIC_API.Models

{
    11 references
    public class Product
    {
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        1 reference
        public int Id
        {
            get;
            set;
        }

        0 references
        public string Name
        {
            get;
            set;
        }

        0 references
        public string Description
        {
            get;
            set;
        }

        0 references
        public decimal Price
        {
            get;
            set;
        }

        0 references
        public int CategoryId
        {
            get;
            set;
        }
    }
}
```

Step 6: Create a ProductRepository for all our operations.

```
using BASIC_API.DBContexts;
using BASIC_API.Models;
using BASIC_API.Repository;
using Microsoft.EntityFrameworkCore;

namespace BASIC_API.Repository
{
    2 references
    public class ProductRepository : IProductRepository
    {
        private readonly ProductContext _dbContext;

        0 references
        public ProductRepository(ProductContext dbContext)
        {
            _dbContext = dbContext;
        }

        2 references
        public void DeleteProduct(int productId)
        {
            var product = _dbContext.Products.Find(productId);
            _dbContext.Products.Remove(product);
            Save();
        }

        2 references
        public Product GetProductByID(int productId)
        {
            return _dbContext.Products.Find(productId);
        }

        2 references
        public IEnumerable<Product> GetProducts()
        {
            return _dbContext.Products.ToList();
        }

        2 references
        public void InsertProduct(Product product)
        {
            _dbContext.Add(product);
            Save();
        }

        4 references
        public void Save()
        {
            _dbContext.SaveChanges();
        }

        2 references
        public void UpdateProduct(Product product)
        {
            _dbContext.Entry(product).State = EntityState.Modified;
            Save();
        }
    }
}
```

Step 7: Create an interface IProductRepository to access the ProductRepository.

```
using BASIC_API.Models;
using System.Collections.Generic;

namespace BASIC_API.Repository
{
    4 references
    public interface IProductRepository
    {
        2 references
        IEnumerable<Product> GetProducts();
        2 references
        Product GetProductByID(int product);
        2 references
        void InsertProduct(Product product);
        2 references
        void DeleteProduct(int productId);
        2 references
        void UpdateProduct(Product product);
        4 references
        void Save();

    }
}
```

Step 8: Create a ProductContext to build the model for the database; this will be a Code First Database Approach.

```
using Microsoft.EntityFrameworkCore;
using BASIC_API.Models;

namespace BASIC_API.DBContexts
{
    7 references
    public class ProductContext : DbContext
    {
        0 references
        public ProductContext(DbContextOptions<ProductContext> options) : base(options)
        {
        }

        4 references
        public DbSet<Product> Products
        {
            get;
            set;
        }

        0 references
        public DbSet<Category> Categories
        {
            get;
            set;
        }

        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Category>().HasData(
                new Category
                {
                    Id = 1,
                    Name = "Electronics",
                    Description = "Electronic Items",
                },
                new Category
                {
                    Id = 2,
                    Name = "Clothes",
                    Description = "Dresses",
                },
                new Category
                {
                    Id = 3,
                    Name = "Grocery",
                    Description = "Grocery Items",
                }
            );
        }
    }
}
```

Step 9: Add a ProductController to get an endpoint for the APIs.

```
using Microsoft.AspNetCore.Mvc;
using System.Transactions;
using BASIC_API.Models;
using BASIC_API.Repository;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace BASIC_API.Controllers
{
    [Produces("application/json")]
    [Route("api/Product")]
    [ApiController]
    public class ProductController : ControllerBase
    {
        private readonly IProductRepository _productRepository;

        public ProductController(IProductRepository productRepository)
        {
            _productRepository = productRepository;
        }

        //GET: api/Product
        [HttpGet]
        public IActionResult Get()
        {
            var products = _productRepository.GetProducts();
            return new OkObjectResult(products);
        }

        //GET: api/Product/5
        [HttpGet("{id}", Name = "Get")]
        public IActionResult Get(int id)
        {
            var product = _productRepository.GetProductByID(id);
            return new OkObjectResult(product);
        }

        //POST: api/Product
        [HttpPost]
        public IActionResult Post([FromBody] Product product)
        {
            using (var scope = new TransactionScope())
            {
                _productRepository.InsertProduct(product);
                scope.Complete();
                return CreatedAtAction(nameof(Get), new { id = product.Id }, product);
            }
        }

        //PUT: api/Product/5
        [HttpPut]
        public IActionResult Put([FromBody] Product product)
        {
            if (product != null)
            {
                using (var scope = new TransactionScope())
                {
                    _productRepository.UpdateProduct(product);
                    scope.Complete();
                    return new OkResult();
                }

            }
            return new NoContentResult();
        }

        //DELETE: api/ApiWithAction/5
        [HttpDelete("{id}")]
        public IActionResult Delete(int id)
        {
            _productRepository.DeleteProduct(id);
            return new OkResult();
        }
    }
}
```

Step 10: Add the connection string settings to connect the project to the SQLSERVER database. Once added run **add-migration** command to run the migration.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "ProductDB": "Server=LAPTOP-5A4DVJ0D\\MYSQLSERVER2022;Database=ProductDB;Trusted_Connection=True;MultipleActiveResultSets=true;"
  }
}
```

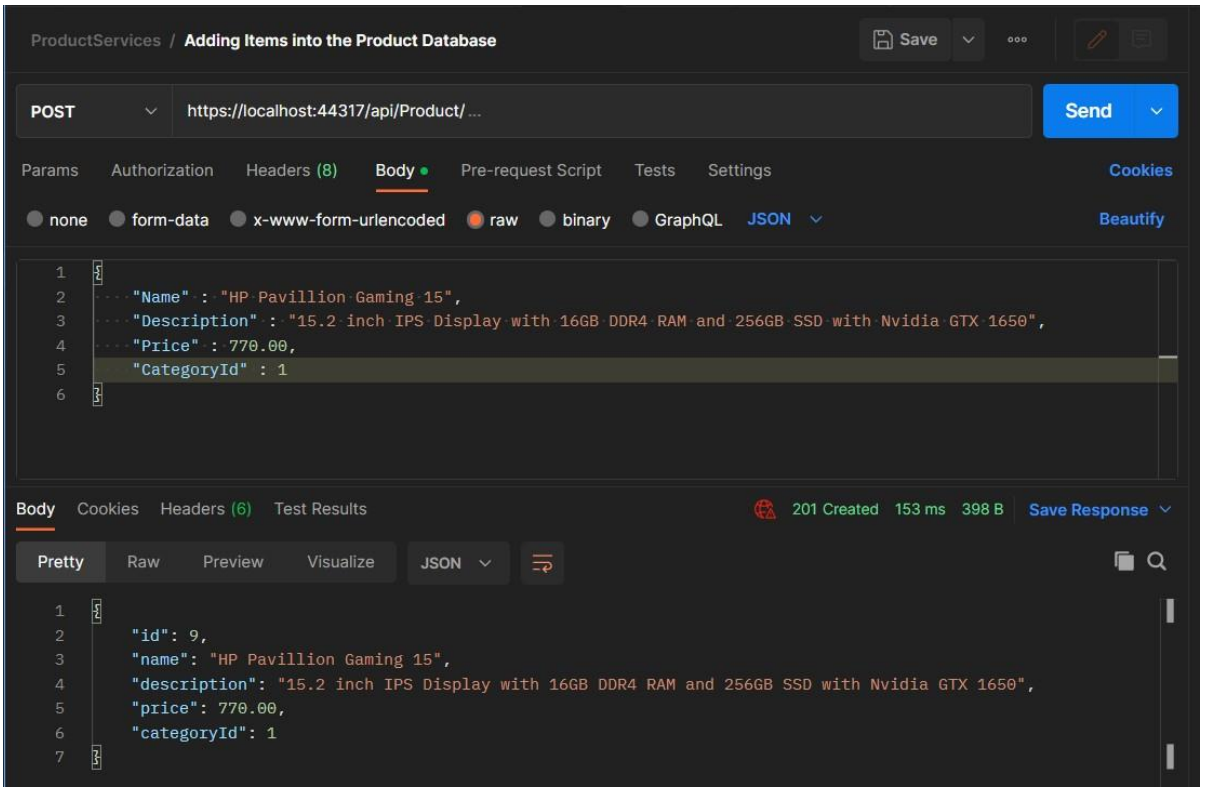
```
PM> add-migration IntitalCreate -verbose
Using project 'BASIC-API'.
Using startup project 'BASIC-API'.
Build started...
Build succeeded.
```

Step 11: Run the **update-database** command to reflect the model changes to the database.

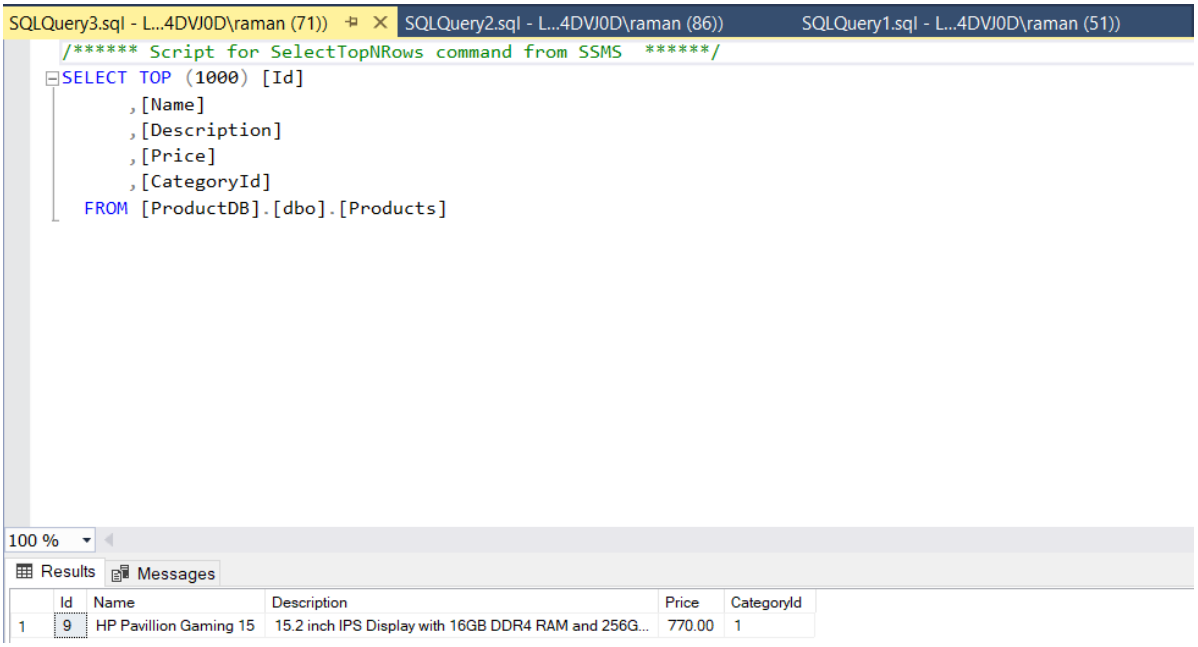
```
PM> update-database -verbose
Using project 'BASIC-API'.
Using startup project 'BASIC-API'.
Build started...
Build succeeded.
```

Step 12: Try to run the project, you will see that a browser window opens with an empty list.

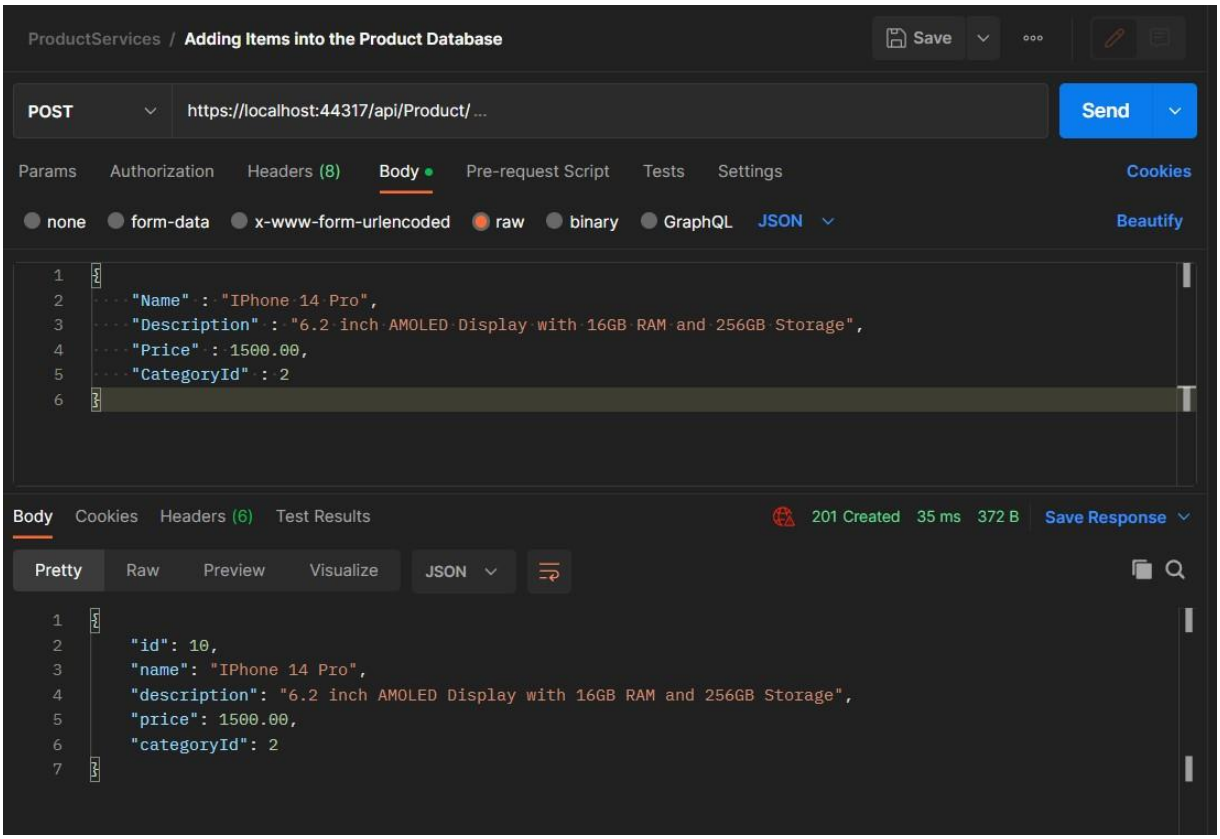
Step 13: Using Postman try to add a product into the product table via the API call.



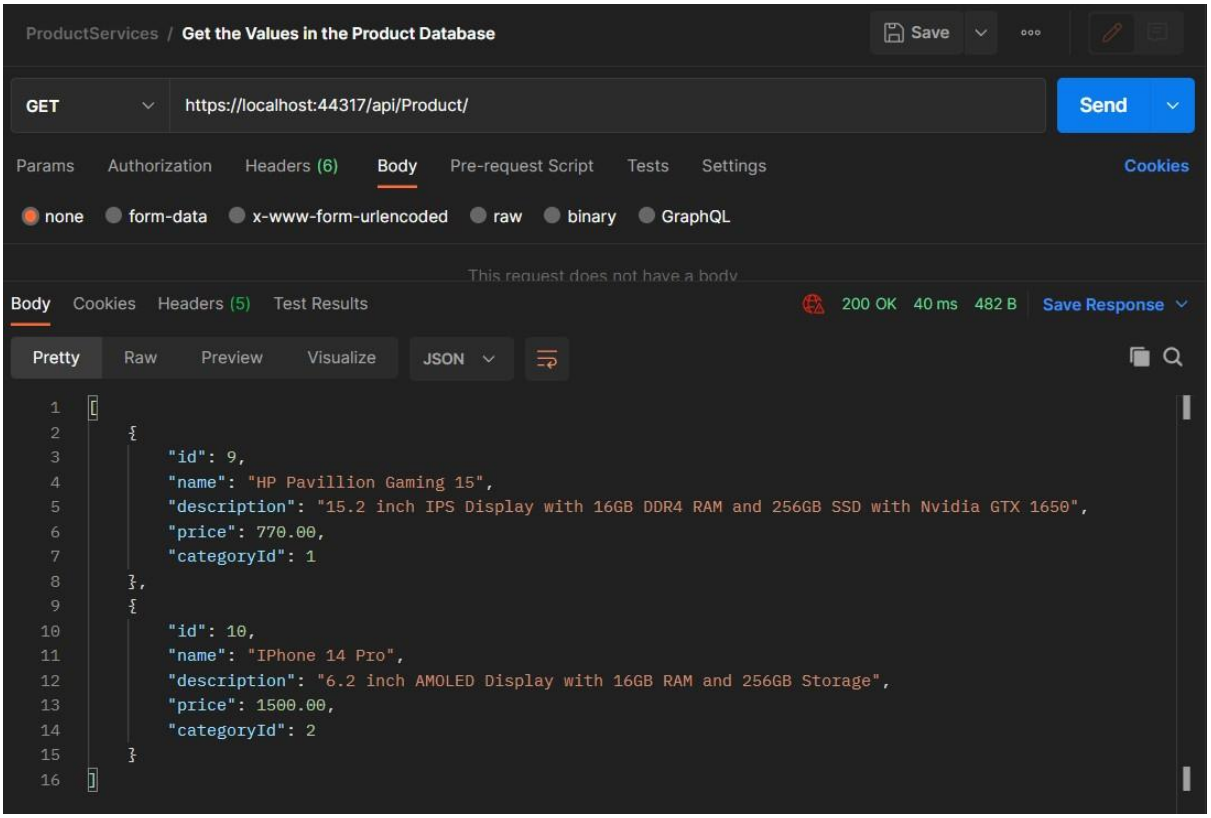
Step 14: You will notice the SQL table to populate with that product.



Step 15: Add another Product to test.



Step 16: Get all the added products using GET.



Step 17: Delete a product using DELETE.

ProductServices / **Delete the Values in the Product Database**

DELETE https://localhost:44317/api/Product/9 Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

Body Cookies Headers (5) Test Results 500 Internal Server Error 3.58 s 3.17 KB Save Response

Pretty Raw Preview Visualize Text

```
System.ArgumentNullException: Value cannot be null. (Parameter 'entity')
   at Microsoft.EntityFrameworkCore.Utilities.Check.NotNull[T](T value, String parameterName)
   at Microsoft.EntityFrameworkCore.DbContext.Remove[TEntity](TEntity entity)
   at Microsoft.EntityFrameworkCore.Internal.InternalDbSet`1.Remove(TEntity entity)
   at BASIC_API.Repository.ProductRepository.DeleteProduct(Int32 productId) in
    C:\Users\raman\source\repos\BASIC-API\BASIC-API\Repository\ProductRepository.cs:line 23
   at BASIC_API.Controllers.ProductController.Delete(Int32 id) in
    C:\Users\raman\source\repos\BASIC-API\BASIC-API\Controllers\ProductController.cs:line 71
   at lambda_method(Closure , Object , Object[] )
   at Microsoft.Extensions.Internal.ObjectMethodExecutor.Execute(Object target, Object[] parameters)
   at Microsoft.AspNetCore.Mvc.Infrastructure.ActionMethodExecutor.SyncActionResultExecutor.Execute
    (IActionResultTypeMapper mapper, ObjectMethodExecutor executor, Object controller, Object[] arguments)
```

Step 18: Check is product is deleted using GET again.

ProductServices / **Get the Values in the Product Database**

GET https://localhost:44317/api/Product/ Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

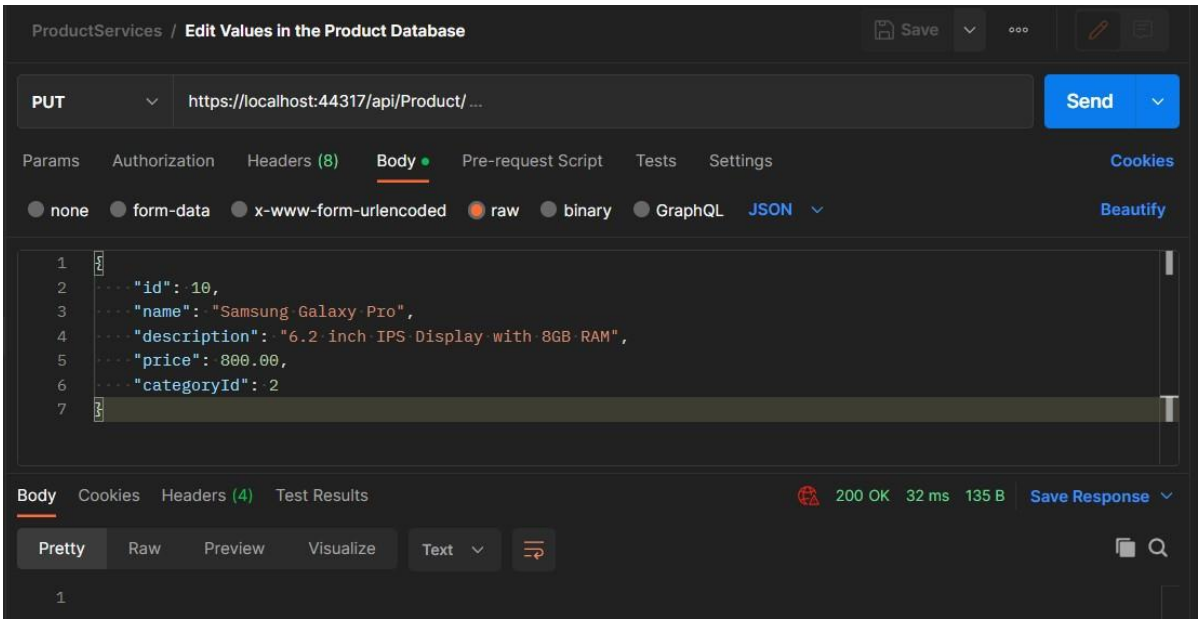
This request does not have a body

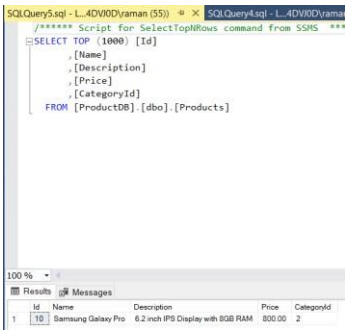
Body Cookies Headers (5) Test Results 200 OK 19 ms 319 B Save Response

Pretty Raw Preview Visualize JSON

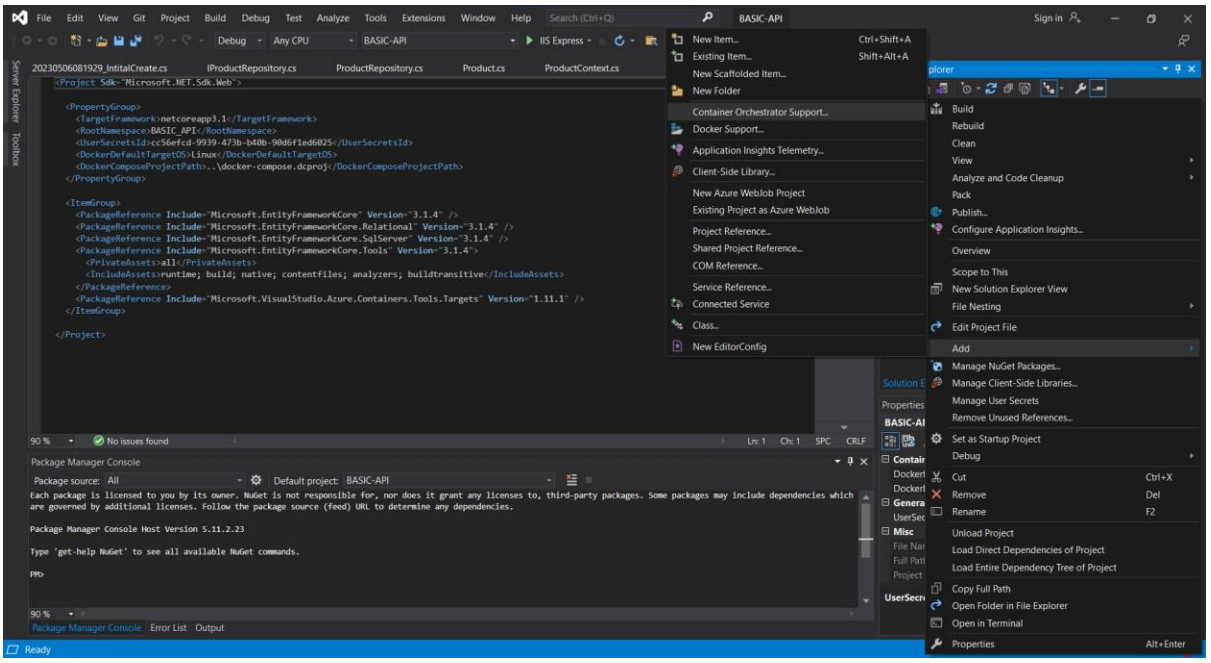
```
[
    {
        "id": 10,
        "name": "IPhone 14 Pro",
        "description": "6.2 inch AMOLED Display with 16GB RAM and 256GB Storage",
        "price": 1500.00,
        "categoryId": 2
    }
]
```

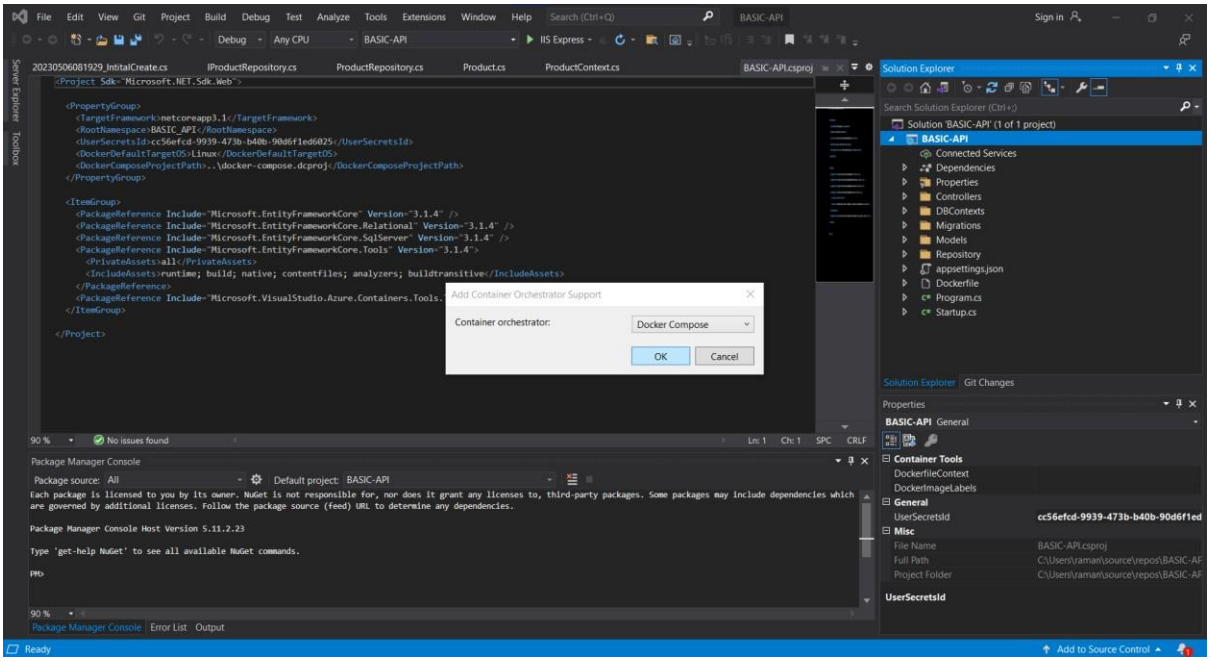
Step 19: Update a product using the PUT.



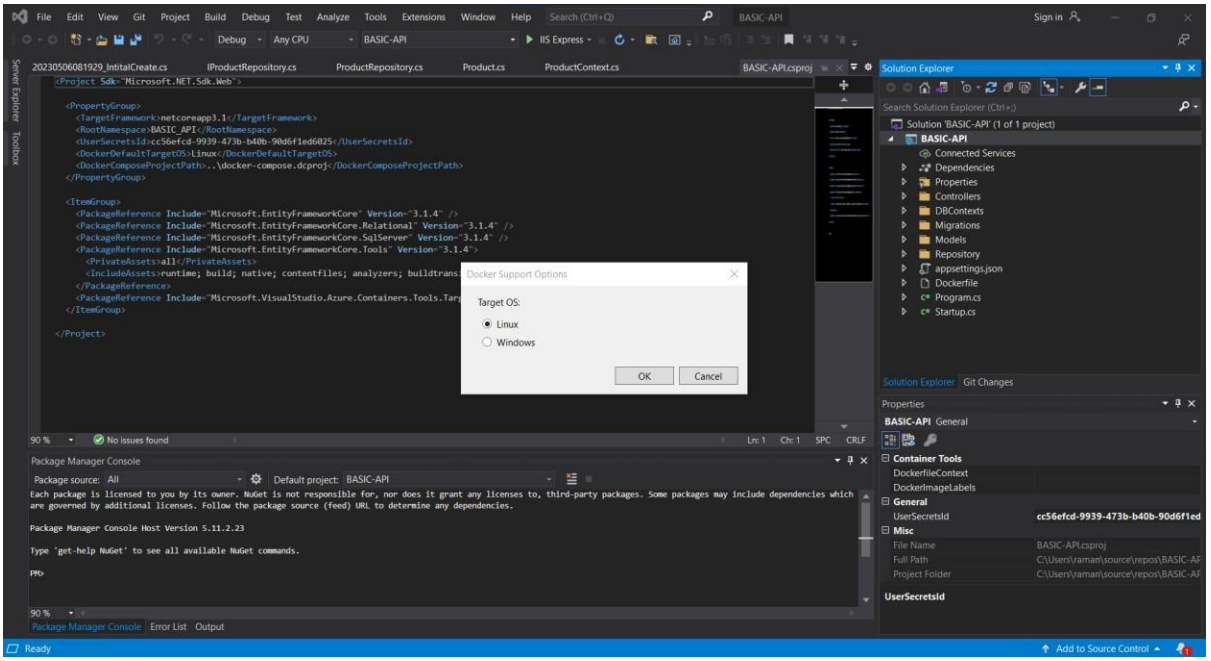


Step 20: Now create a docker container and image of the product. Right click the project and select Container Orchestrator Support and select Ok.

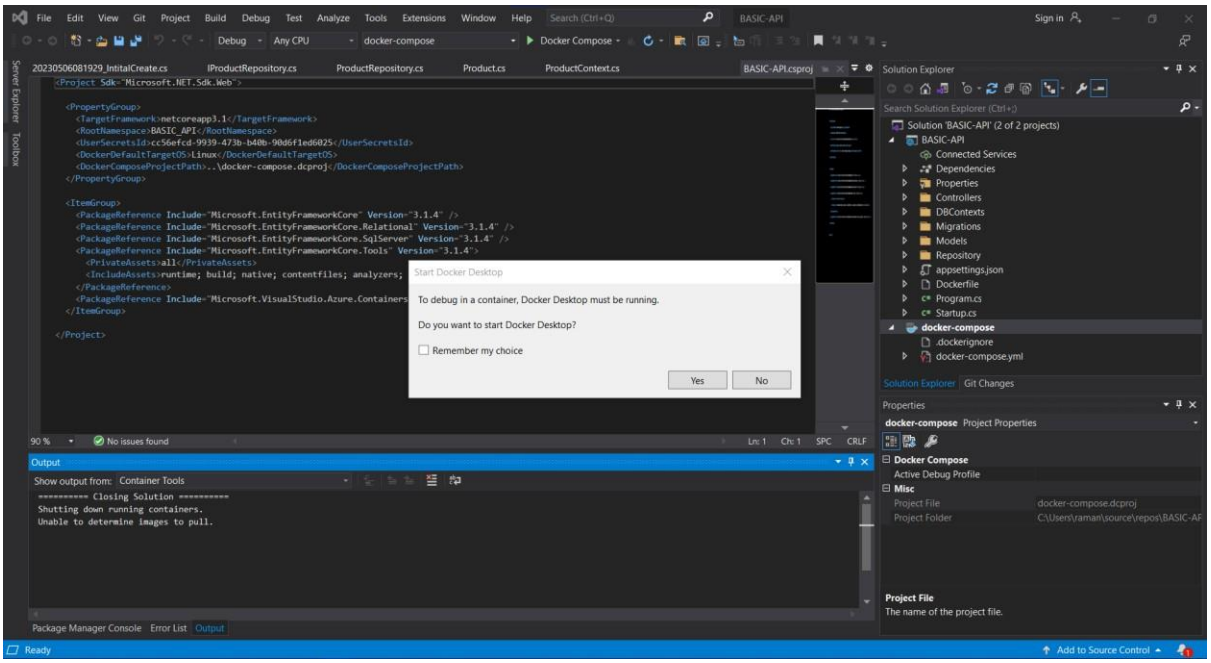




Step 21: Select the target docker OS support which will be LINUX.



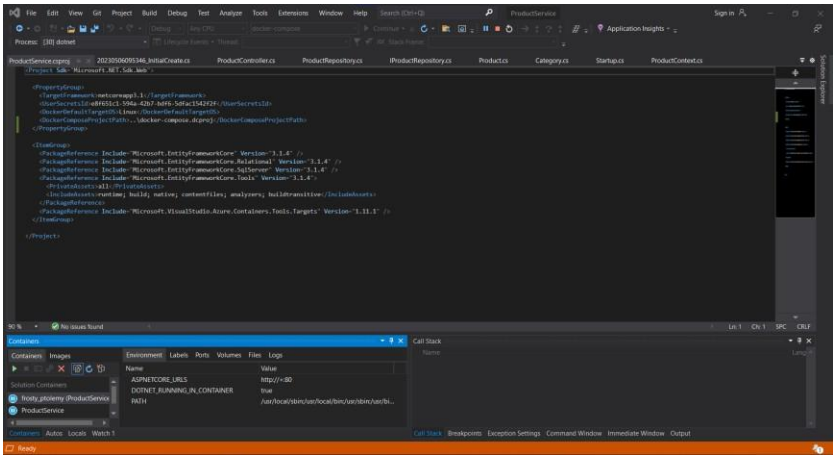
Step 22: You can select Yes to run the Docker desktop too.



Step 23: Run the docker commands ‘docker images’ to see the created image of your project.

```
C:\Users\raman>docker images
REPOSITORY                    TAG       IMAGE ID       CREATED        SIZE
webapi1                       dev       30aee5f1f9c9   7 weeks ago    208MB
webapplication1               dev       872e14f1554c   7 weeks ago    208MB
<none>                        <none>    bae4e11c4bb1   7 weeks ago    208MB
productservice                dev       2a9886bd1404   7 weeks ago    208MB
basicapi                      dev       591cee1a5e44   7 weeks ago    208MB
<none>                        <none>    7a5deb8e122b   7 weeks ago    208MB
webapplication2               dev       5a5a5c6434b9   7 weeks ago    208MB
<none>                        <none>    ba2df3366bff   7 weeks ago    208MB
mcr.microsoft.com/dotnet/aspnet  3.1    454a55eab920   4 months ago   208MB
```

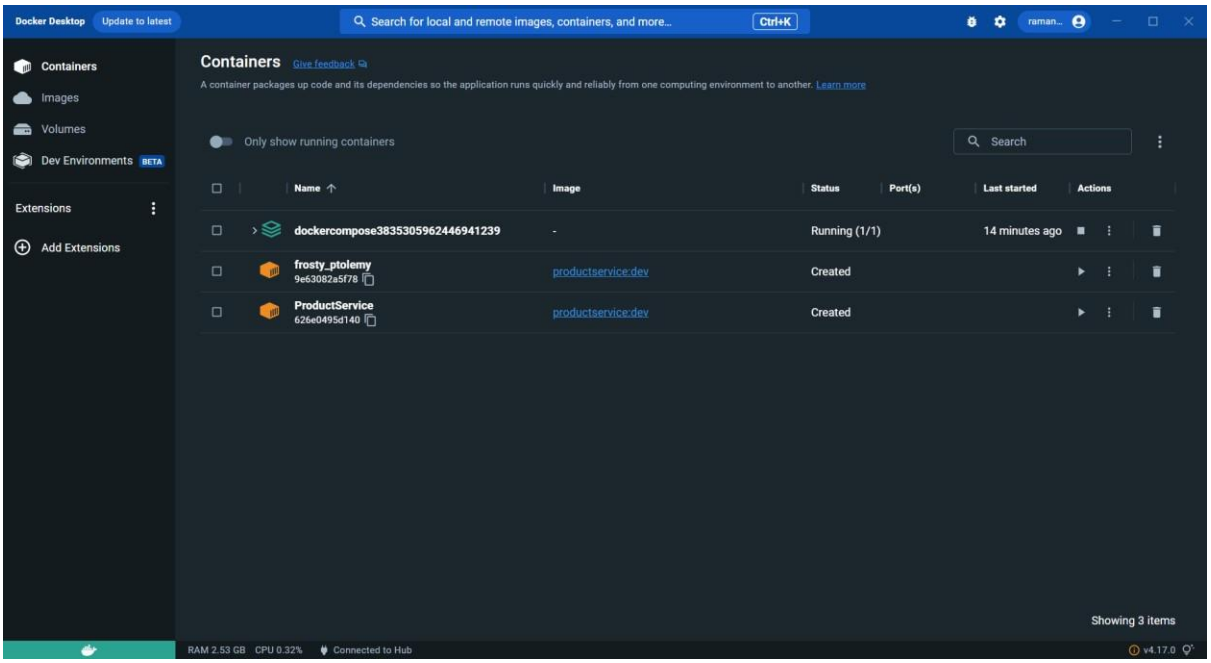
Step 24: Run the Project with the option Docker Compose on the Play.

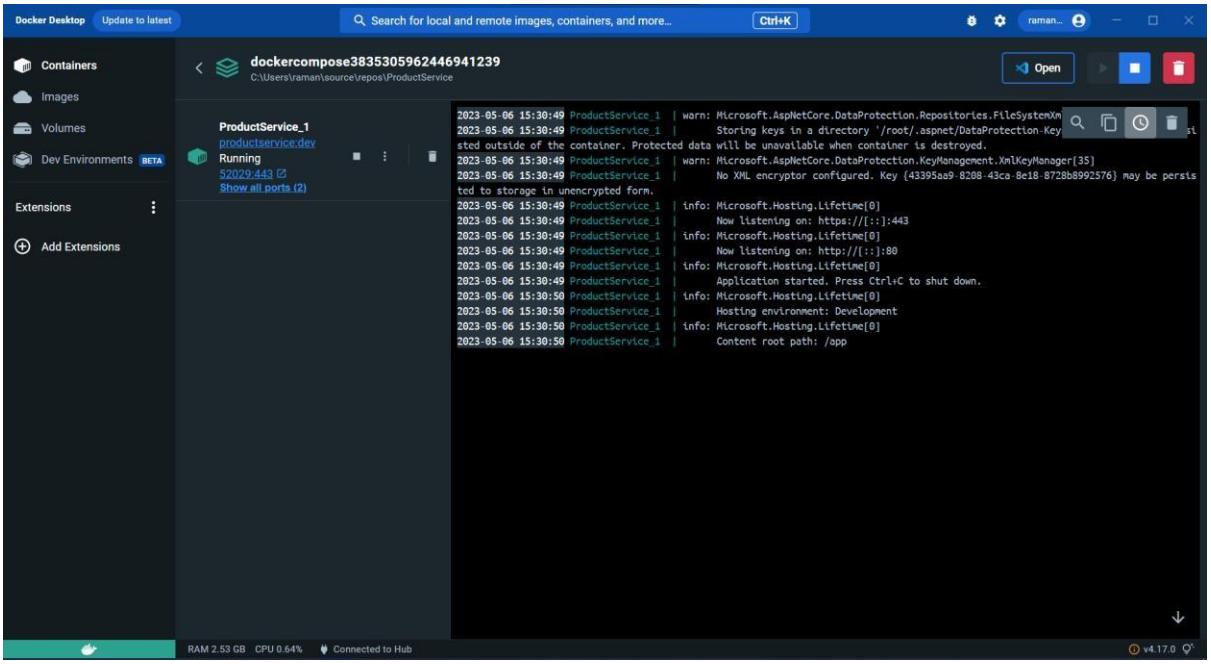


Step 25: Run the **docker ps** command to check the running docker container.

```
C:\Users\raman\source\repos\ProductService>docker ps
CONTAINER ID   IMAGE                 COMMAND               CREATED          STATUS          PORTS                                          NAMES
6d946df292ed   productservice:dev    "tail -f /dev/null"   11 minutes ago   Up 11 minutes   0.0.0.0:52030->80/tcp, 0.0.0.0:52029->443/tcp   ProductService_1
```

Step 26: Check docker desktop for the running image.





Step 27: The docker image and containers will be running in the docker desktop app.