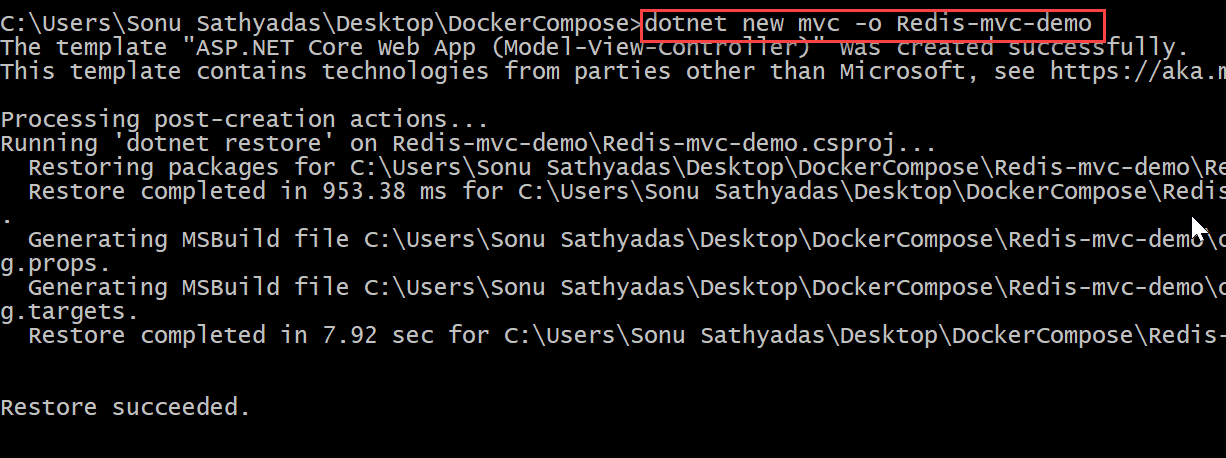
**.NET Core with Redis cache**

Steps to connect .NET Core MVC application with Redis cache.

1. Create a .NET Core MVC project by running the following command.



1. Open the project folder in VS code. To do so, move to the project folder in command prompt and use the command “code .”.
2. We will need to use Nuget to add 2 addtional packages to our project.

**Install-Package Microsoft.Extensions.Caching.Redis.Core**

**Install-Package Microsoft.AspNetCore.Session**

1. We can now start add some settings to our appsettings.Development.json , these are custom configuration settings we have set them explicitly for our development machine. These settings may change for Production Environment and you will have a different appsettings file for that environment.

**"redis": {**

**"host": "127.0.0.1",**

**"port": 6379,**

**"name": "localhost"**

**}**

1. Now, we need to configure the distributed cache service for our application. To configure it, open the Startup.cs file and add the following lines of code to the ConfigureServices method.

**services.AddDistributedRedisCache(options =>**

**{**

**options.InstanceName = Configuration.GetValue<string>("redis:name");**

**options.Configuration = Configuration.GetValue<string>("redis:host");**

**});**

**services.AddSession();**

1. Update the Configure method to use the session for the application.

**app.UseSession();**

**app.UseStaticFiles();**

**app.UseMvc(routes =>**

**{**

**routes.MapRoute(**

**name: "default",**

**template: "{controller=Home}/{action=Index}/{id?}"**

**);**

**});**

1. Adding the AddDistributedCache option enables an Interface IDistributedCache that we can use to add and retrieve values. This also enables controller dependency injection by default, so we can make use of IDistributedCache throughout our web application. We have now completed most of our configuration required for using Redis. You can now add some code to the HomeController.cs

**public IActionResult Index()**

**{**

**var helloRedis = Encoding.UTF8.GetBytes("Hello Redis");**

**HttpContext.Session.Set("hellokey", helloRedis);**

**var getHello = default(byte[]);**

**HttpContext.Session.TryGetValue("hellokey", out getHello);**

**ViewData["Hello"] = Encoding.UTF8.GetString(getHello);**

**return View();**

**}**

1. Update the index.cshtml view file with the following code.

**@{**

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

**}**

**<div class="row">**

**<div class="col-md-12">**

**<h2>Redis - MVC application</h2>**

**<p>Message from Redis: @ViewData["message"]</p>**

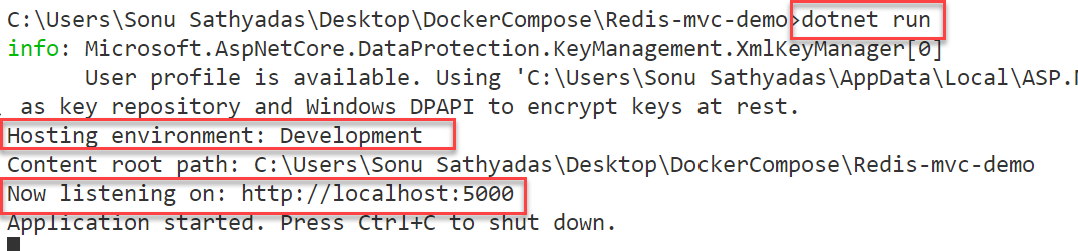
**</div>**

**</div>**

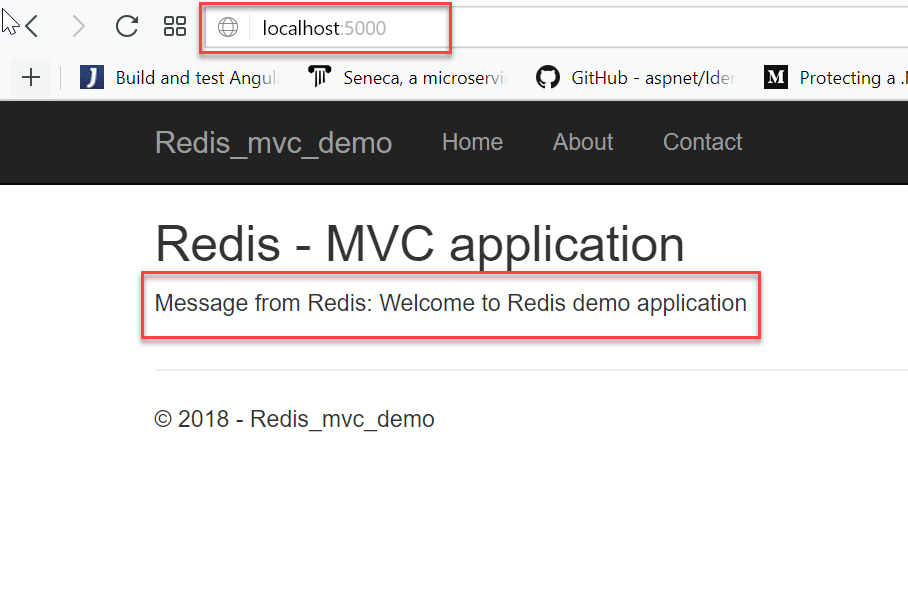
1. We have configured the redis cache settings in appsettings.Development.json file, so we need to run the application in Development environment. To run the application in development environment set the ASPNETCORE\_ENVIRONMENT environment variable to “Development” by running the following command.

**Set ASPNETCORE\_ENVIRONMENT = Development**

1. Now, run the application using “dotnet run” command.



1. Open the browser and navigate to the url <http://localhost:5000>



**IDistributedCache Interface**

Redis stores data in [Key, Value] pairs, enabling the developer to use and store values in efficient ways using a number of different data structures:

* Strings
* Lists
* Hashes
* Sets
* Sorted Sets
* HyperLog

A typical web application needs to be able to store a lot more information in a session or in memory store for users. At the same time applications need to be fast and responsive.

**IDistributedCache**

All distributed cache implementations should adhere to the IDistributedCache interface, which contains contracts for both Synchronous and Asynchronous methods. It is up to the developer to implement these methods.

**public interface IDistributedCache**

**{**

**byte[] Get(string key);**

**Task<byte[]> GetAsync(string key);**

**void Refresh(string key);**

**Task RefreshAsync(string key);**

**void Remove(string key);**

**Task RemoveAsync(string key);**

**void Set(string key, byte[] value, DistributedCacheEntryOptions options);**

**Task SetAsync(string key, byte[] value, DistributedCacheEntryOptions options);**

**}**

1. We have already configured the IDistributedCache interface in Startup.cs. We go ahead and Dependency Inject that into our HomeController and start using it right away.

Add the Following line code to your HomeController

**public class HomeController : Controller**

**{**

**private readonly IDistributedCache \_cache;**

**public HomeController( IDistributedCache cache)**

**{**

**\_cache = cache;**

**}**

**public IActionResult About()**

**{**

**// Add a Value to Redis Cache**

**\_cache.SetString("about", "MVC redis integration demo");**

**//Read that value from the cache**

**ViewData["about"] = \_cache.GetString("about");**

**return View();**

**}**

**}**

1. Update the about.cshtml view with the following code

**@{**

**ViewData["Title"] = "About";**

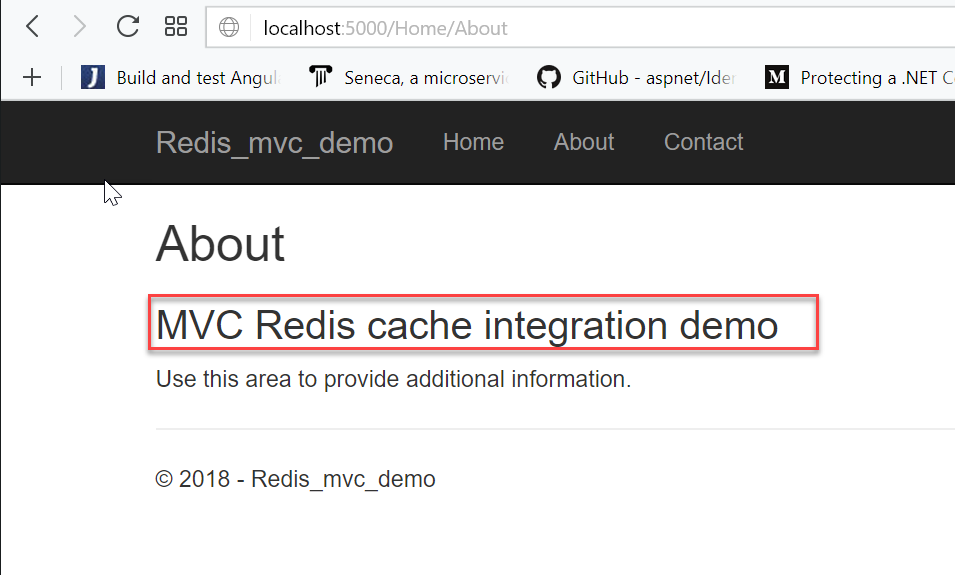
**}**

**<h2>@ViewData["Title"]</h2>**

**<h3>@ViewData["about"]</h3>**

**<p>Use this area to provide additional information.</p>**

1. Run the application and navigate to about page.



**Redis client package from StackExchange**

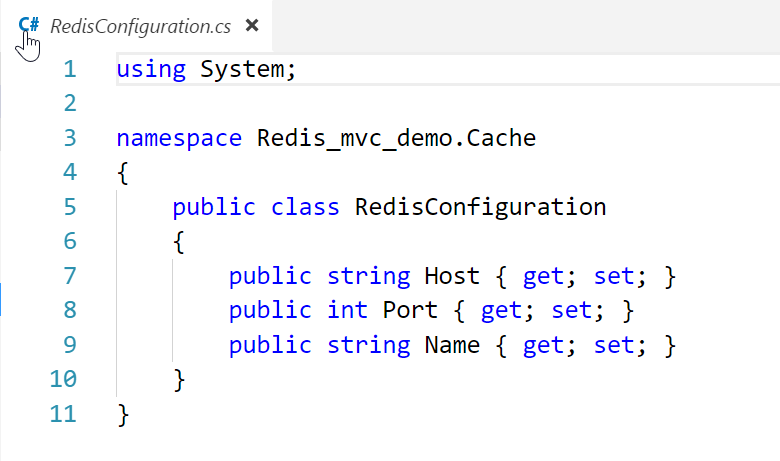
1. Create a .NET Core MVC application using “dotnet new mvc -o redis-demo” command.
2. Navigate to the project folder and open it in VS Code.
3. Install the Redis Client library using the following command.

Dotnet add package StackExchange.Redis

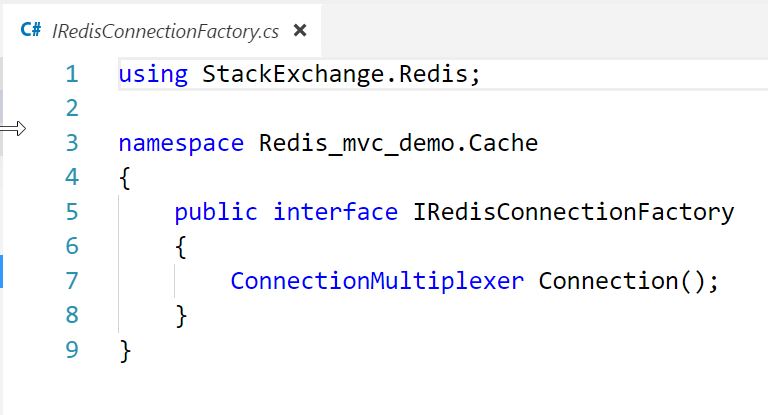
1. Open “appsettings.Development.json” file and add the following configuration settings.



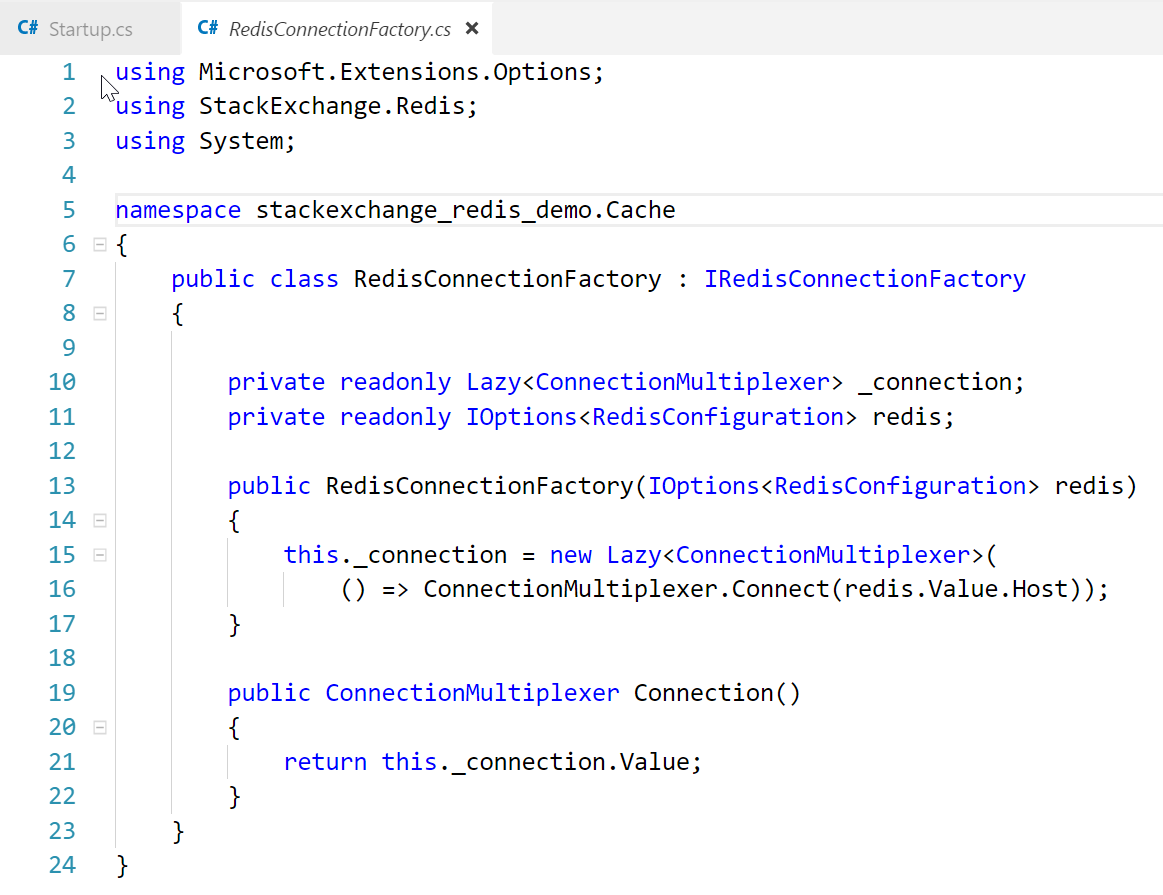
1. Create a new folder named “Cache” in the project root folder and add a new class file named “RedisConfiguration” with the following content.



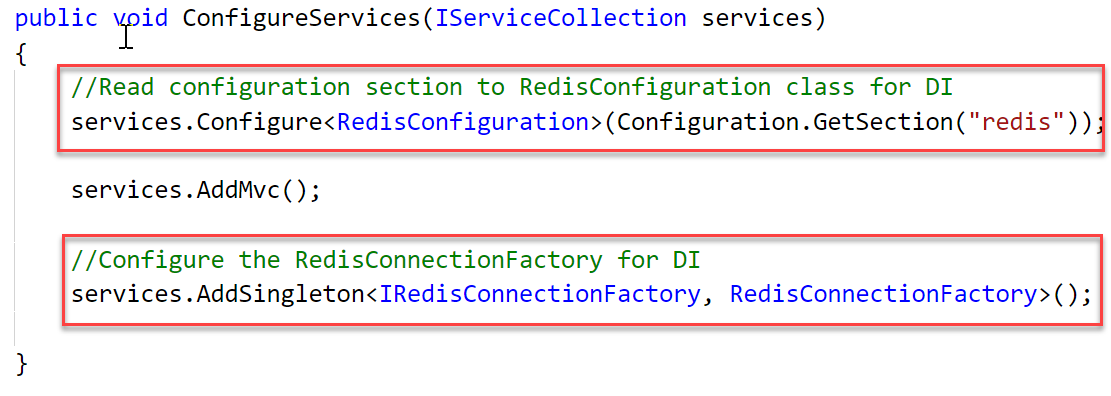
1. Create a new interface file named “IRedisConnectionFactory” in the “Cache” folder.



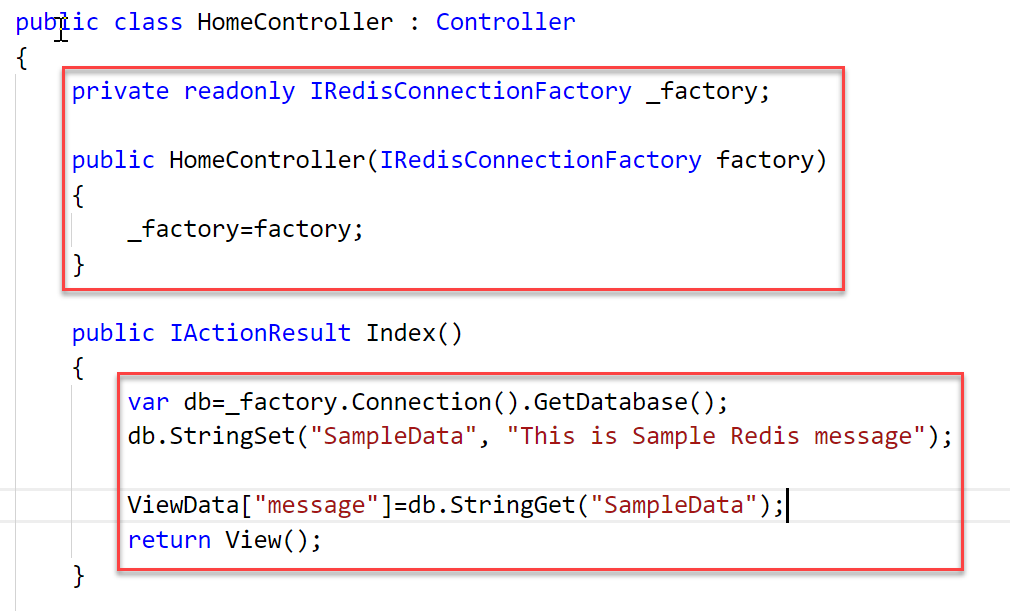
1. Add a new class by implementing the IRedisConnectionFactory interface with the following code.



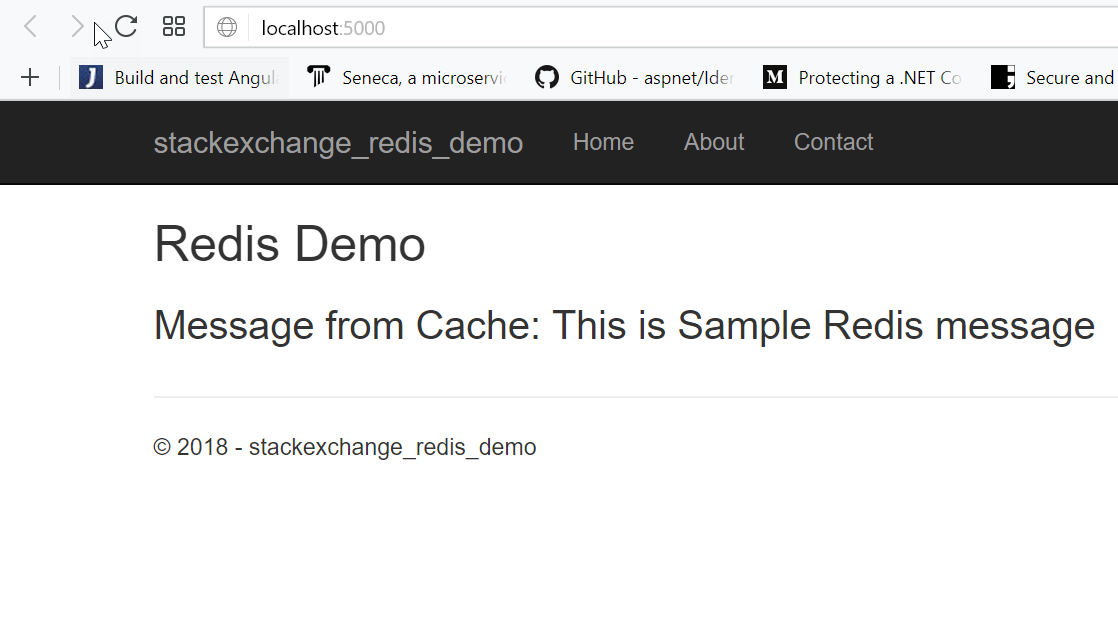
1. Open the startup.cs file and update the ConfigureServices method to configure DistributedCache service.



1. Open the HomeController class and add the following code to the Index page.



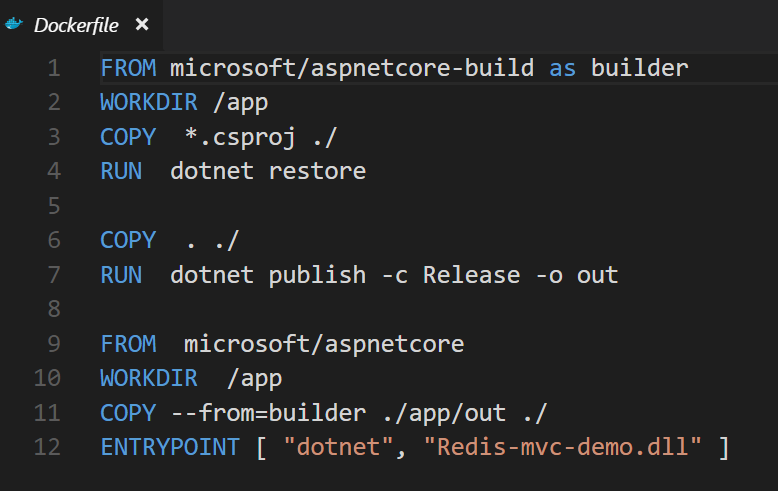
1. Run the project using “dotnet run” command and navigate to the url <http://localhost:5000>



**Deploying multi-container microservices using Docker compose**

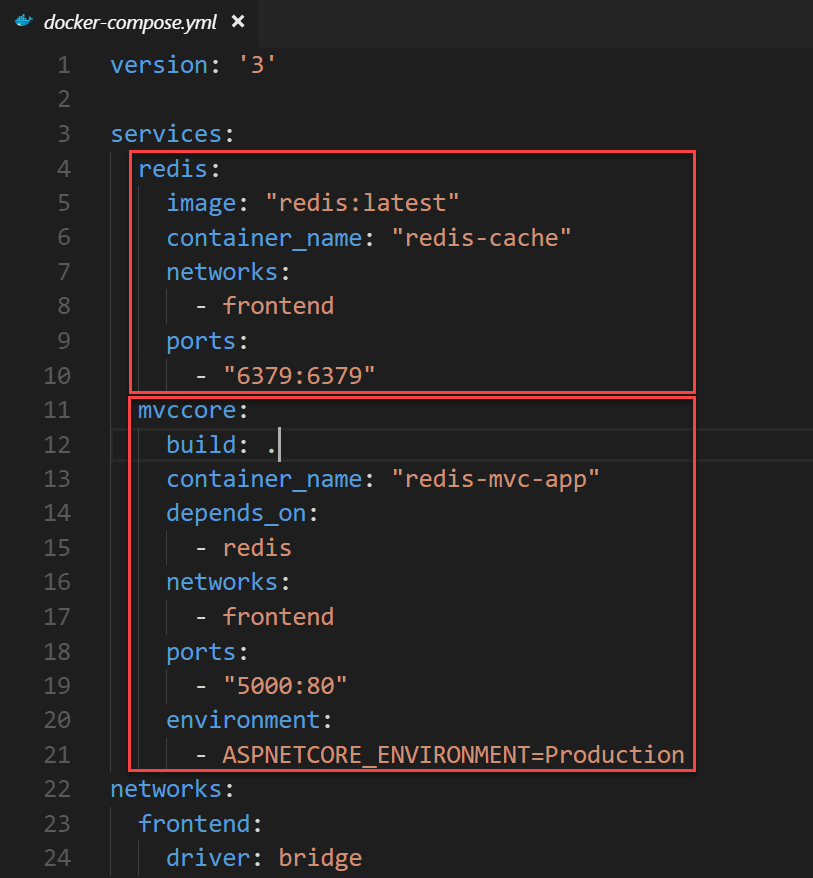
You can deploy the .NET core application along with redis cache using a docker compose file. To deploy multiple containerized applications using docker compose tool, you need to create a docker-compose.yml file in the .net core project root folder, which contains a dockerfile to build and image of the application.

**Docker file sample**



**Docker-compose file sample**

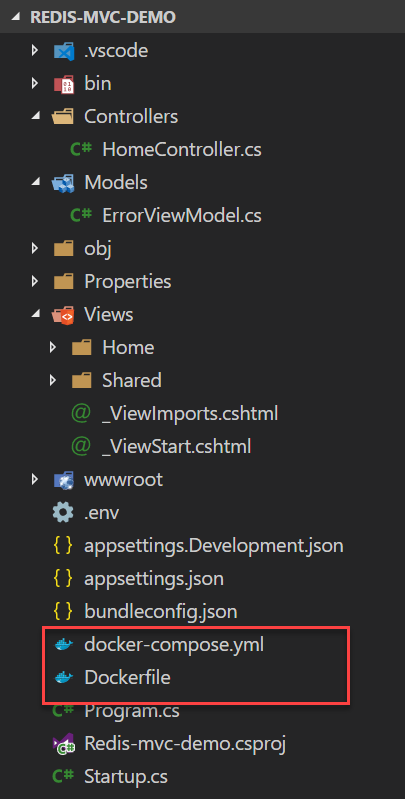
Make sure bot the containers are sharing the same network. Define a network as frontend in the networks section and specify the driver type as bridge. Update the network name of the containers as frontend.



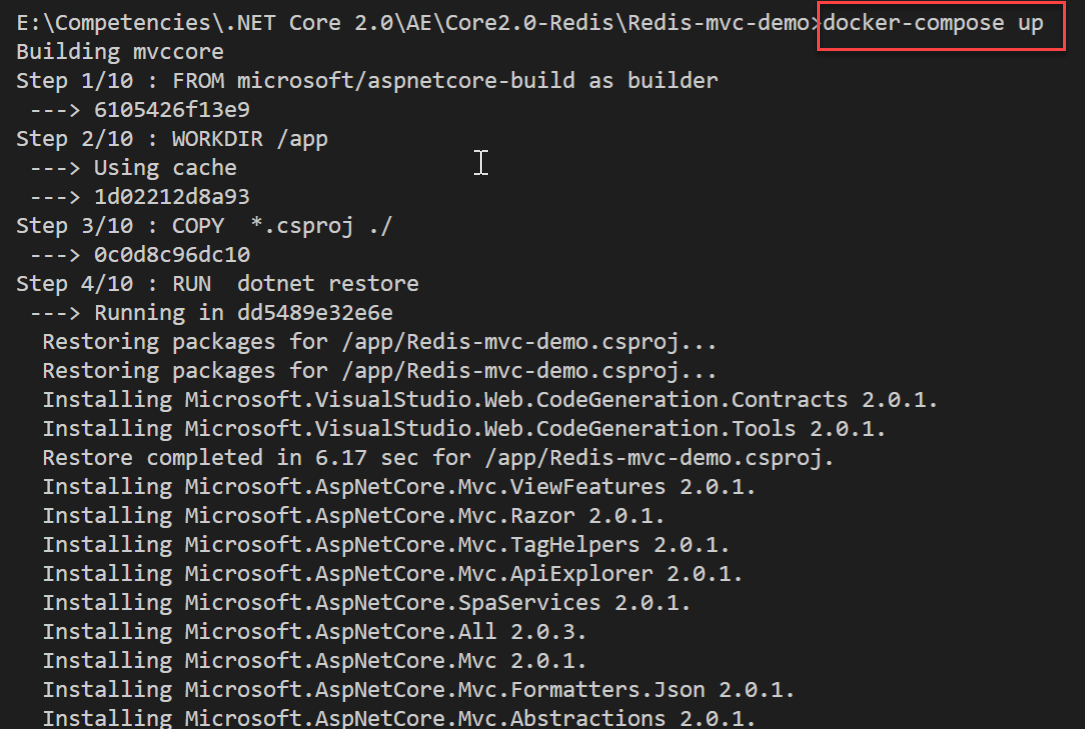
Update the appsettings.json file to include the redis cache configuration. Host name will be the name of the redis container name in the docker compose file.



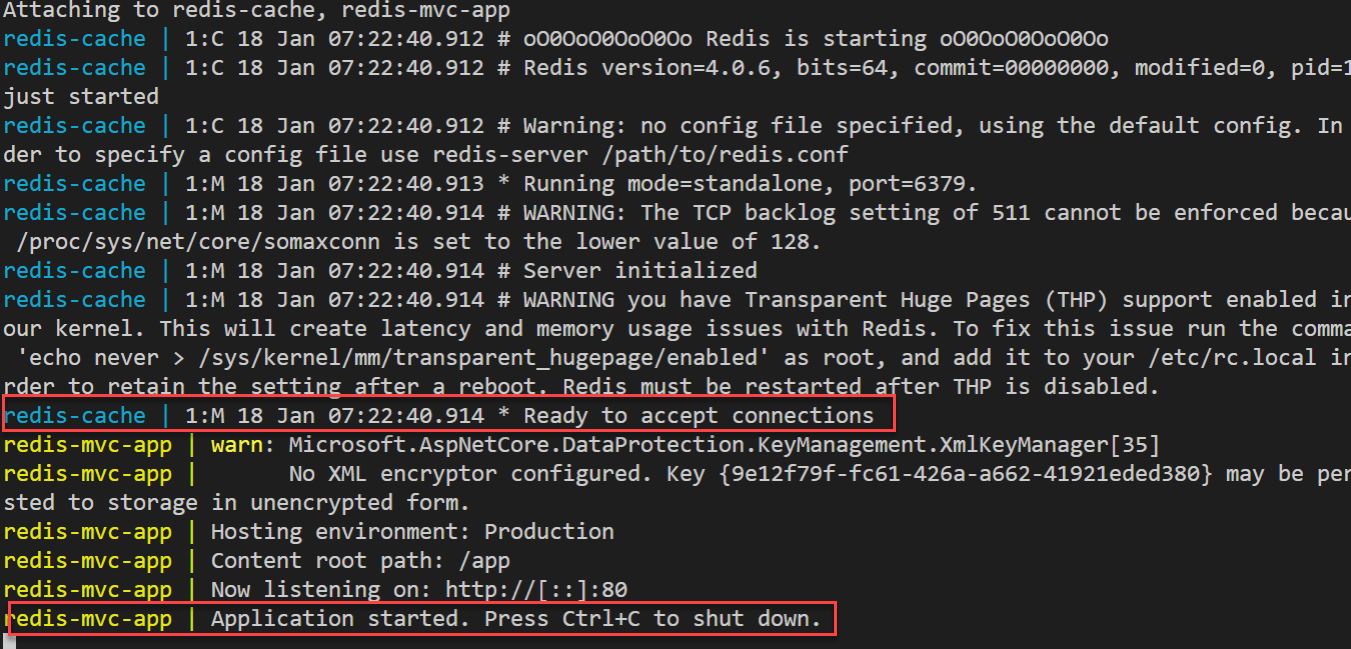
The project structure looks like the following



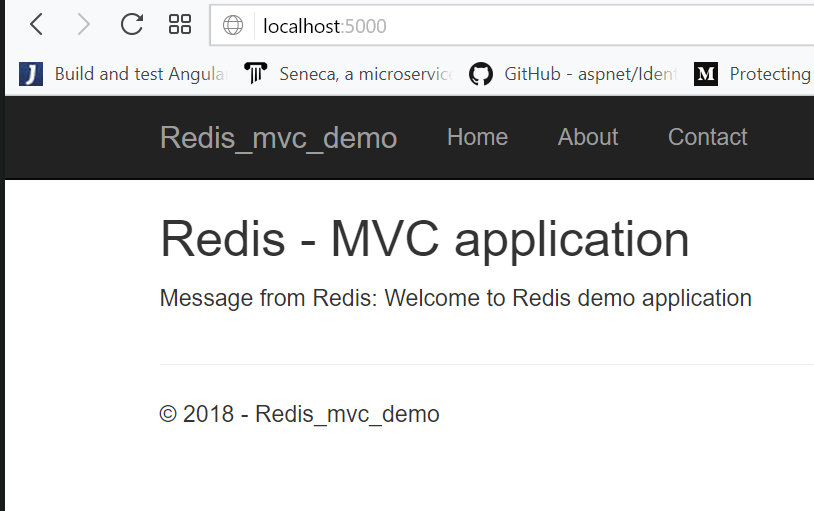
Run the “**docker-compose up**” command to run the multi-container applications.



Once the deployment is completed it shows the services are up and running



Open your browser and navigate to <http://localhost:5000>. It shows the MVC application running with Redis cache.



Execute the “docker ps“ command to list the running containers

