

**Lab Manual- Build and Deploy DotNet Application with Docker**

**Prepared for**:

**Date:** 18th Dec 2023

**Prepared by:**

Document Name: Lab Manual **Document Number** AZLabn991

**Contributor:**

Contents

[1. Objective 2](#_Toc155374966)

[2. Prerequisites 3](#_Toc155374967)

[3. Create .NET app 8](#_Toc155374968)

[4. Publish .NET app 12](#_Toc155374969)

[5. Create the Dockerfile 13](#_Toc155374970)

[6. Create a container 16](#_Toc155374971)

[1. Single run 16](#_Toc155374972)

[2. Create a Stopped container 17](#_Toc155374973)

# Objective

In this Lab, you'll learn how to containerize a .NET application with Docker. Containers have many features and benefits, such as being an immutable infrastructure, providing a portable architecture, and enabling scalability. The image can be used to create containers for your local development environment, private cloud, or public cloud.

In this Lab, you:

* Create and publish a simple .NET app
* Create and configure a Dockerfile for .NET
* Build a Docker image
* Create and run a Docker container

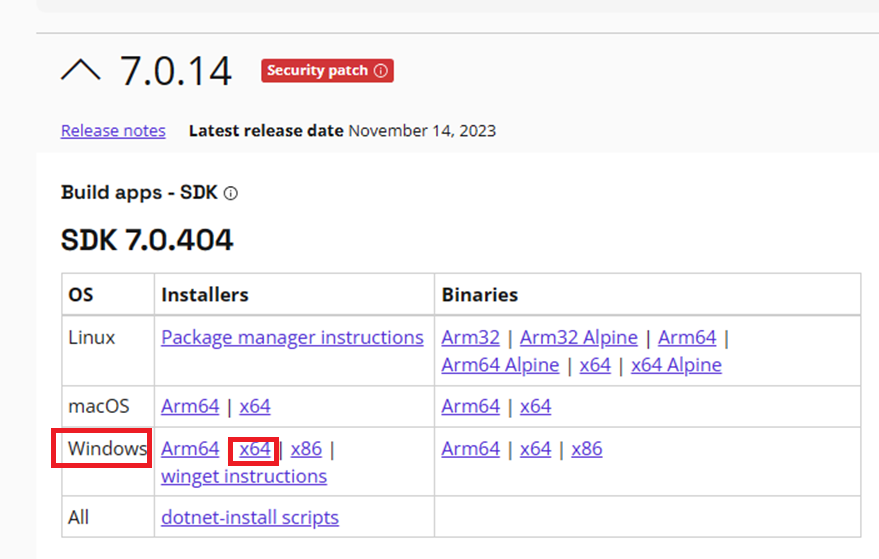
You'll understand the Docker container build and deploy tasks for a .NET application. The *Docker platform* uses the *Docker engine* to quickly build and package apps as *Docker images*. These images are written in the *Dockerfile* format to be deployed and run in a layered container.

# Prerequisites

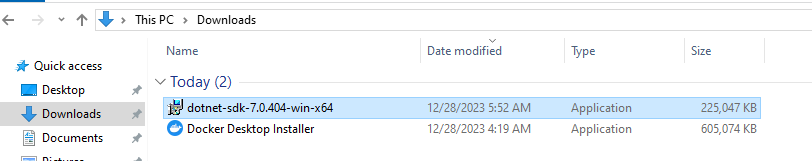
Install the following prerequisites:

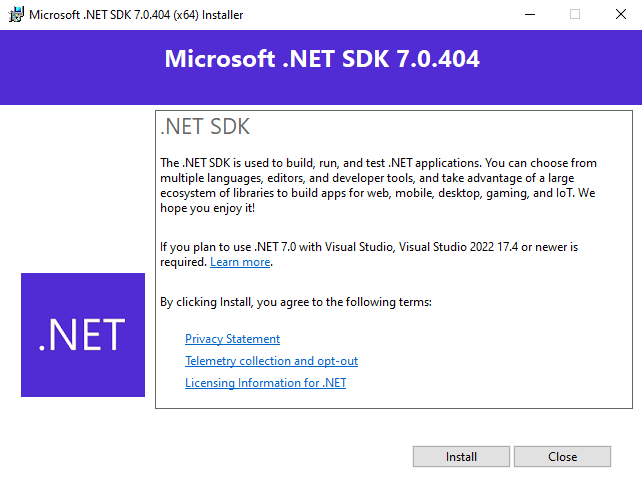
* 1. **Install Dotnet 7.0 SDK**

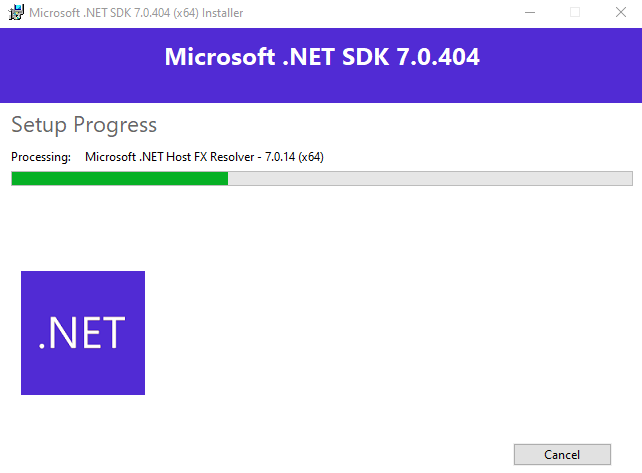
<https://dotnet.microsoft.com/en-us/download/dotnet/7.0>

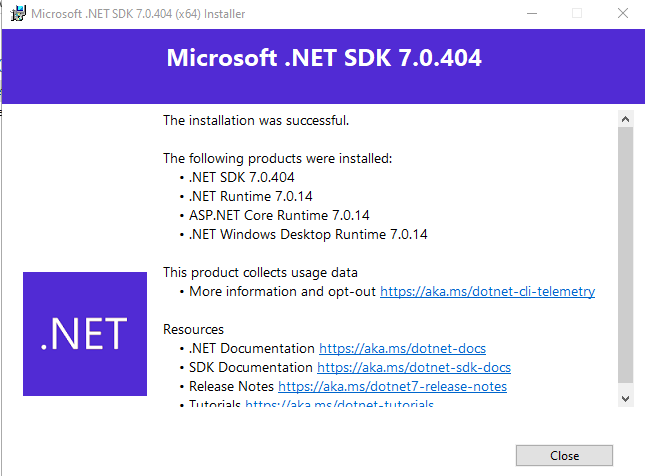


<https://download.visualstudio.microsoft.com/download/pr/03a5170a-a4cd-458c-b5d0-e5149ee4fdcf/e9026f6fe3c3fec4a774e034d4f98ead/dotnet-sdk-7.0.404-win-x64.exe>





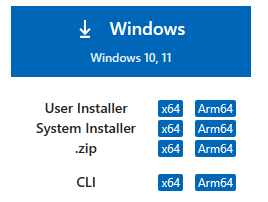


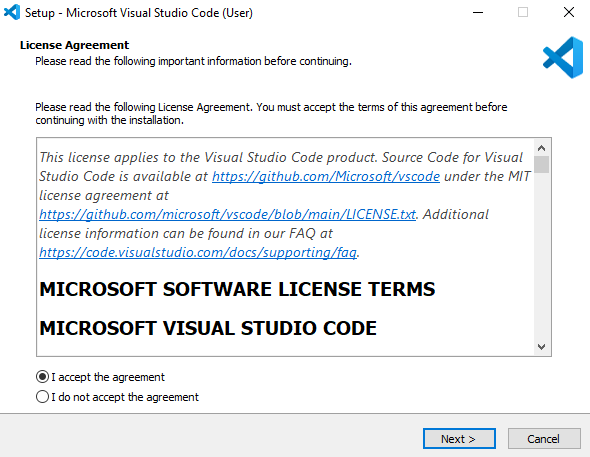


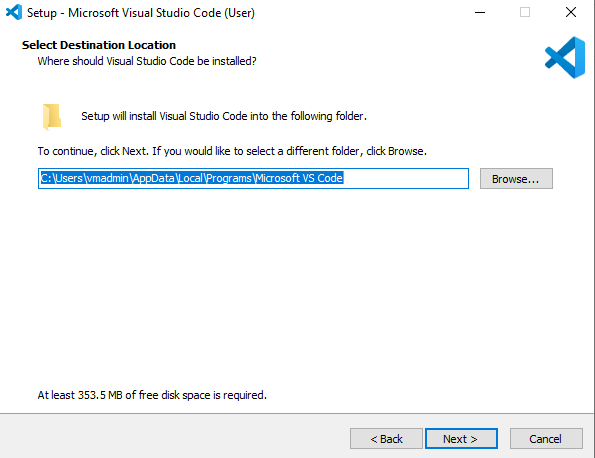
* 1. **Install VSCode**

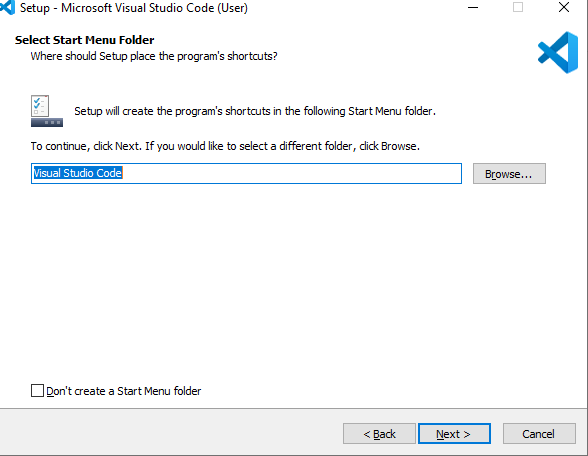
<https://code.visualstudio.com/download>

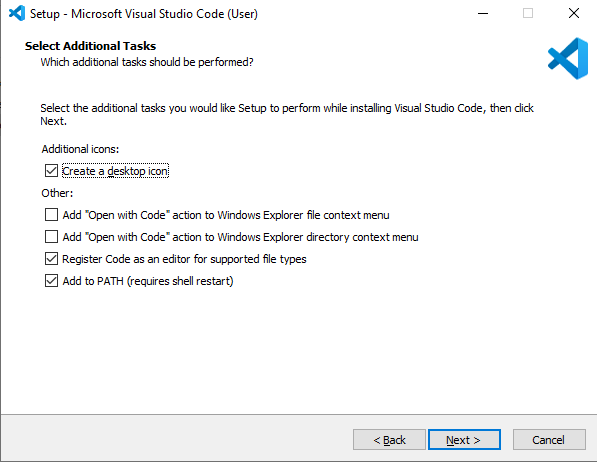
<https://code.visualstudio.com/sha/download?build=stable&os=win32-x64-user>

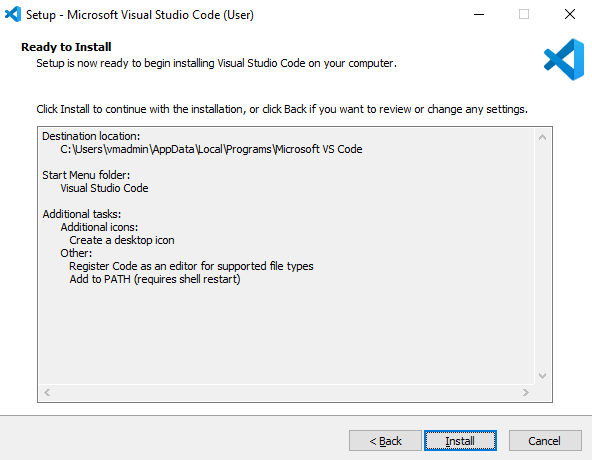


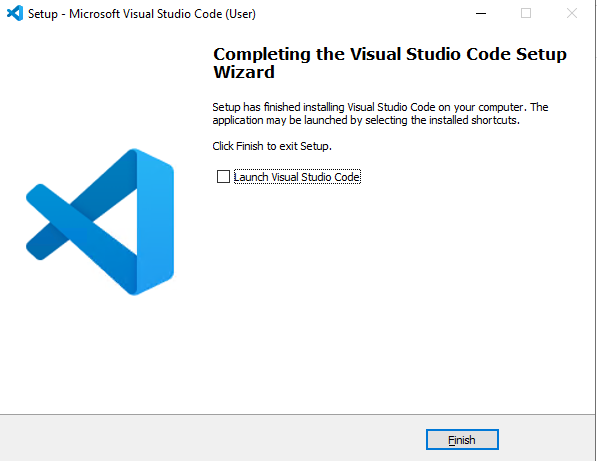








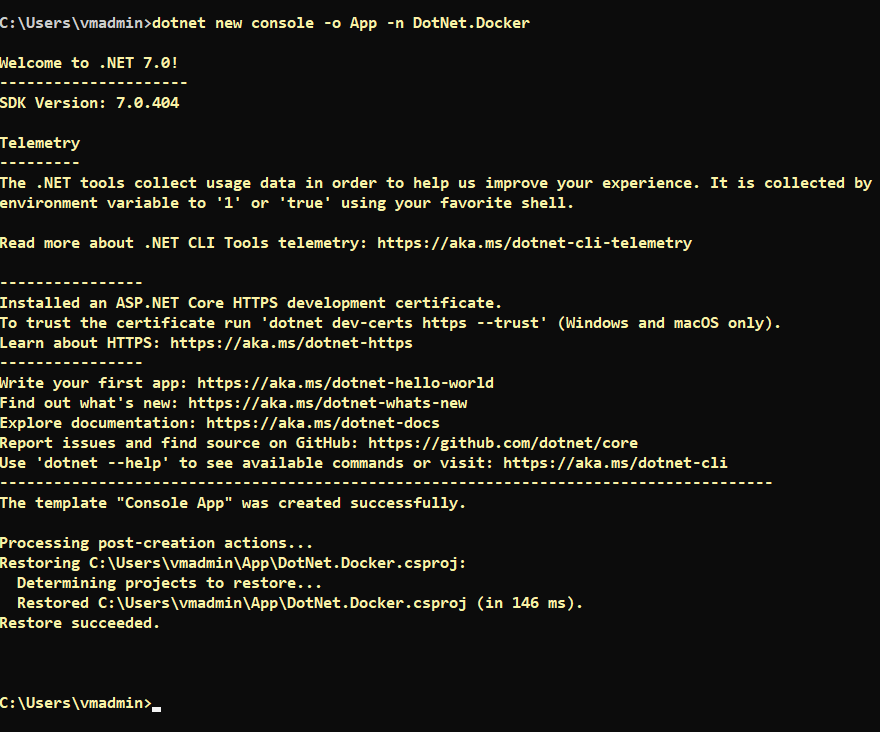




# Create .NET app

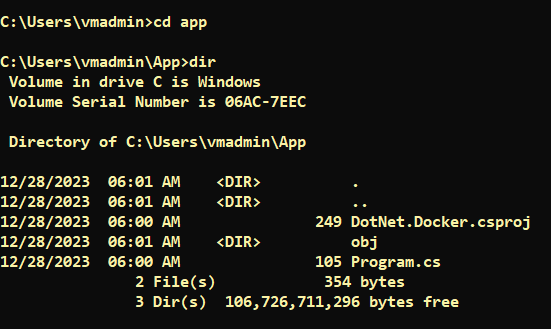
* **Step1:** You need a .NET app that the Docker container runs. Open your terminal, create a working folder if you haven't already, and enter it. In the working folder, run the following command to create a new project in a subdirectory named App:

dotnet new console -o App -n DotNet.Docker



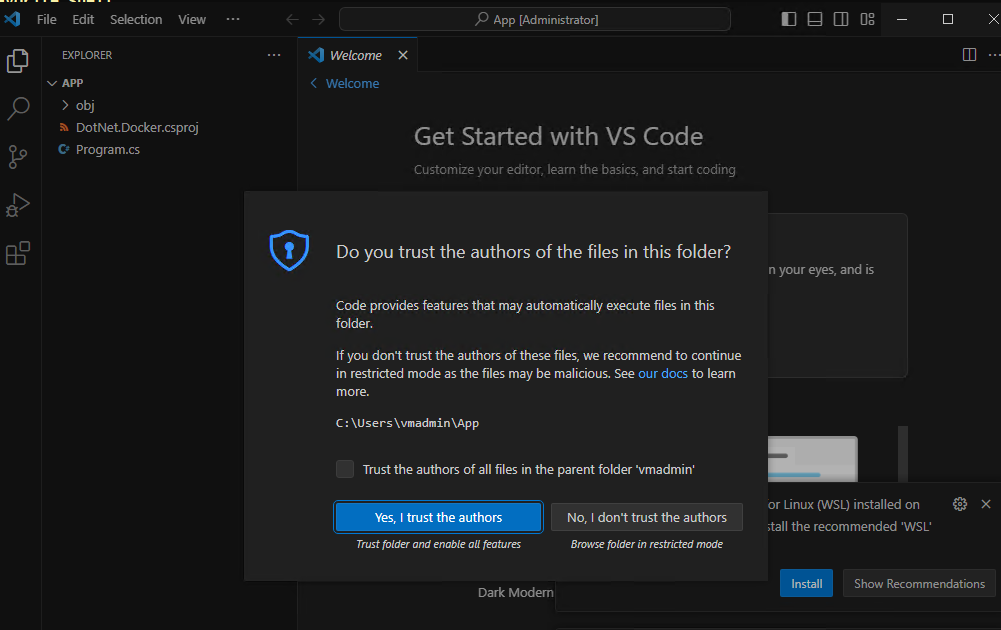
* **Step2:** The dotnet new command creates a new folder named *App* and generates a "Hello World" console application. Change directories and navigate into the *App* folder, from your terminal session

Cd app



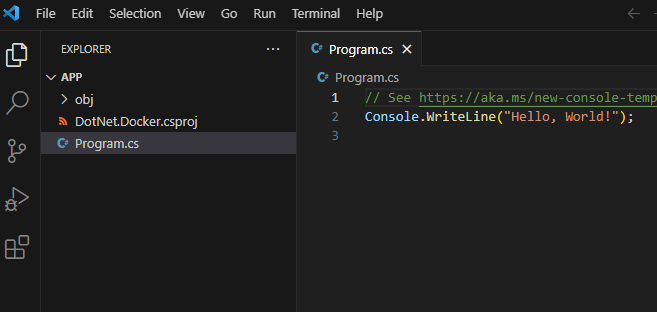
* **Step3:** Launch VSCode from current App Directory. Once Open Click “**Yes , I trust the Author**”

Code .



* **Step4:** The Program.cs should look like the following C# code:

Console.WriteLine("Hello World!");



* **Step5:** Replace the file with the following code that counts numbers every second:

var counter = 0;

var max = args.Length is not 0 ? Convert.ToInt32(args[0]) : -1;

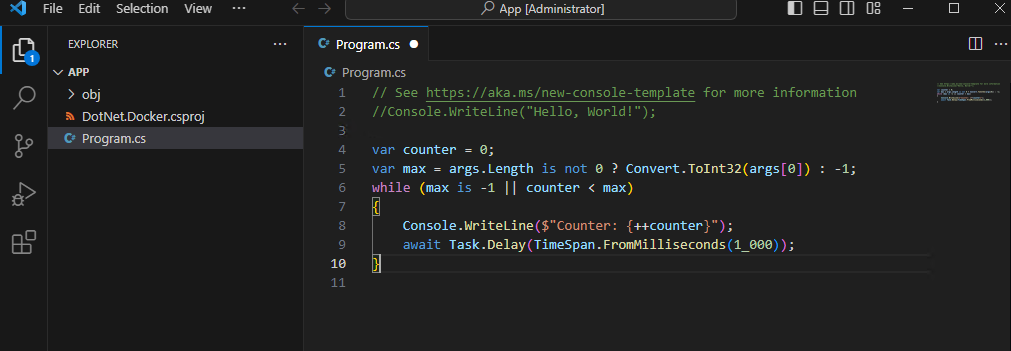
while (max is -1 || counter < max)

{

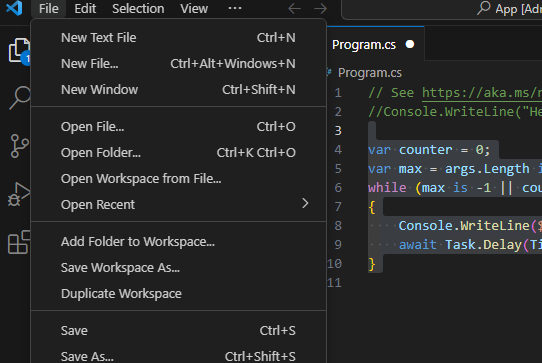
Console.WriteLine($"Counter: {++counter}");

await Task.Delay(TimeSpan.FromMilliseconds(1\_000));

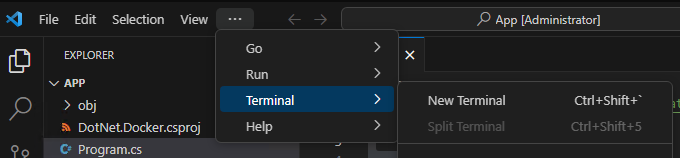
}



* **Step6:** Save the file

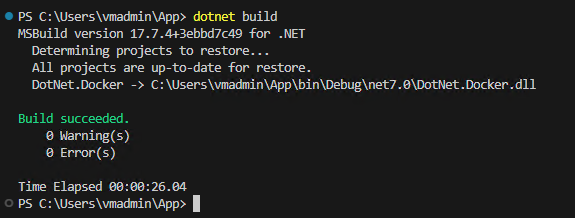


* **Step7:** Launch new Terminal



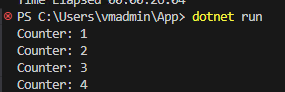
* **Step8:** Run Build

dotnet build



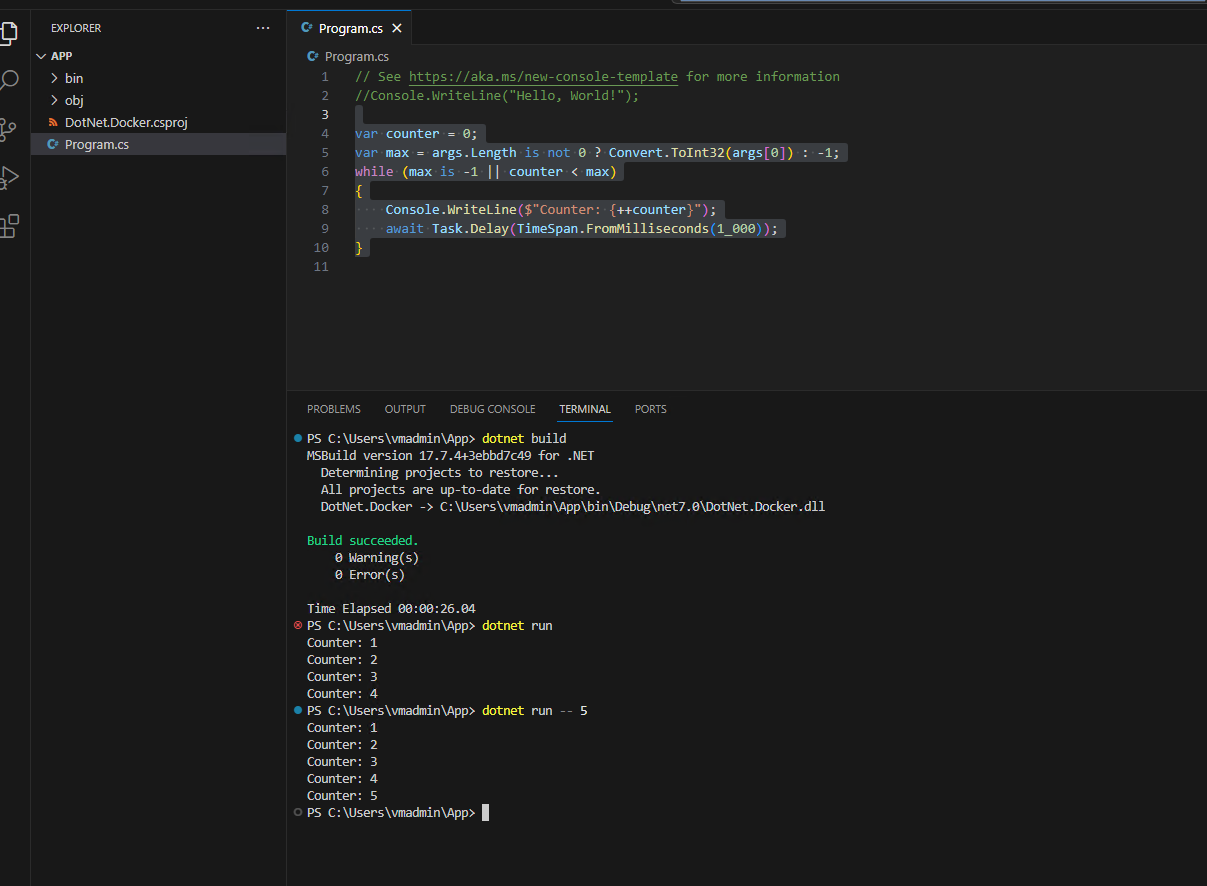
* **Step9:** Test the program with **dotnet run.** Remember that this app runs indefinitely. Use the cancel command Ctrl+C to stop it. The following is an example output:

dotnet run



* **Step10:** Test the If you pass a number on the command line to the app, it will only count up to that amount and then exit. Try it with dotnet run -- 5 to count to five.

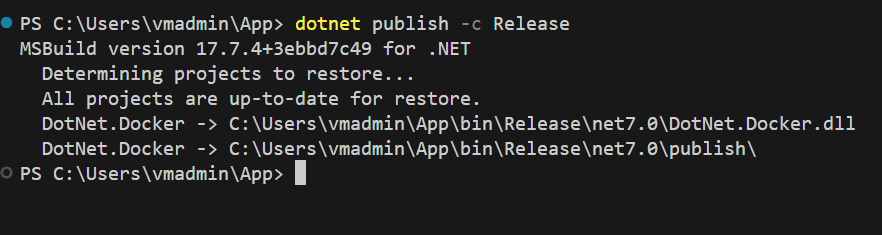
dotnet run -- 5



# Publish .NET app

Before adding the .NET app to the Docker image, first it must be published. It's best to have the container run the published version of the app. To publish the app, run the following command:

dotnet publish -c Release

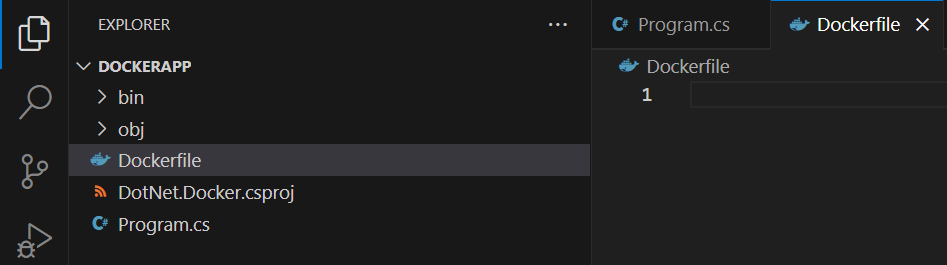


This command compiles your app to the publish folder. The path to the publish folder from the working folder should be .\App\bin\Release\net7.0\publish\.

# Create the Dockerfile

The **Dockerfile** file is used by the docker build command to create a container image. This file is a text file named Dockerfile that doesn't have an extension.

* **Step1:** Create a file named Dockerfile in the directory containing the **.csproj** and open it in a text editor. This tutorial uses the ASP.NET Core runtime image (which contains the .NET runtime image) and corresponds with the .NET console application.



* **Step2:** Write below code

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build-env

WORKDIR /App

# Copy everything

COPY . ./

# Restore as distinct layers

RUN dotnet restore

# Build and publish a release

RUN dotnet publish -c Release -o out

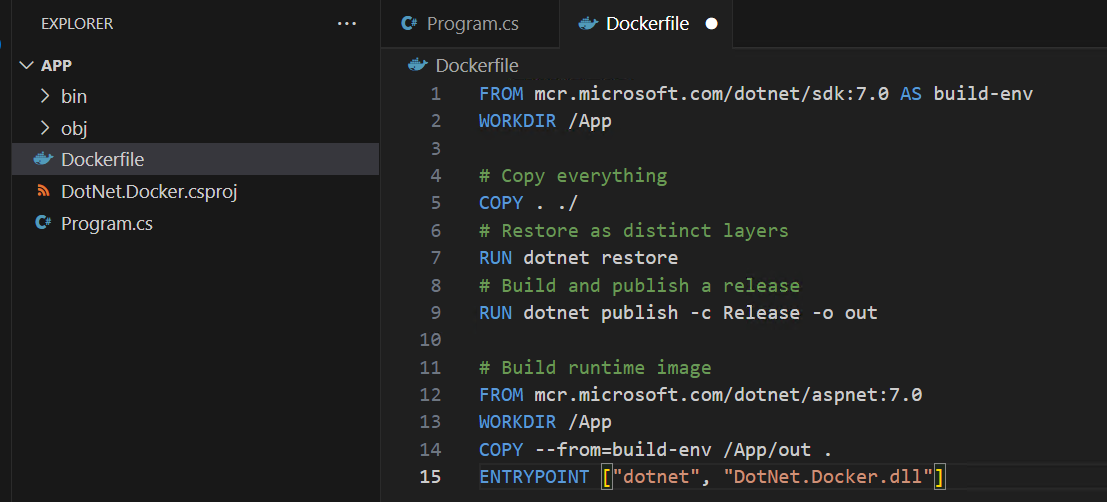
# Build runtime image

FROM mcr.microsoft.com/dotnet/aspnet:7.0

WORKDIR /App

COPY --from=build-env /App/out .

ENTRYPOINT ["dotnet", "DotNet.Docker.dll"]



The **FROM** keyword requires a **fully qualified Docker container image name**. The **Microsoft Container Registry** (**MCR, mcr.microsoft.com**) is a syndicate of Docker Hub, which hosts publicly accessible containers. The dotnet segment is the container repository, whereas the **sdk or aspnet** segment is the container image name. The image is tagged with 7.0, which is used for versioning.

Thus, mcr.microsoft.com/dotnet/aspnet:7.0 is the .NET 7.0 runtime. Make sure that you pull the runtime version that matches the runtime targeted by your SDK. For example, the app created in the previous section used the .NET 7.0 SDK, and the base image referred to in the Dockerfile is tagged with **7.0**.

* **Step2:** From your terminal, run the following command:

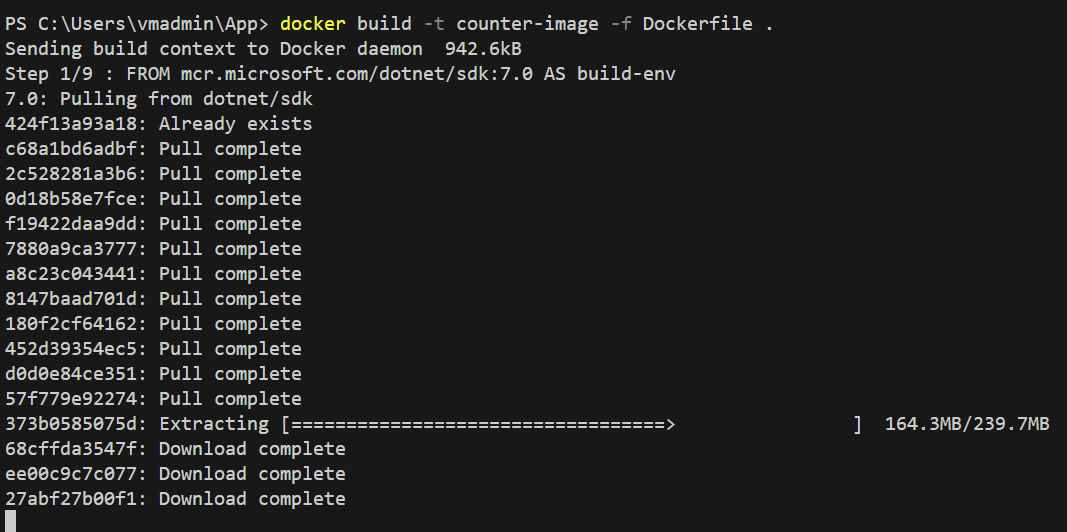
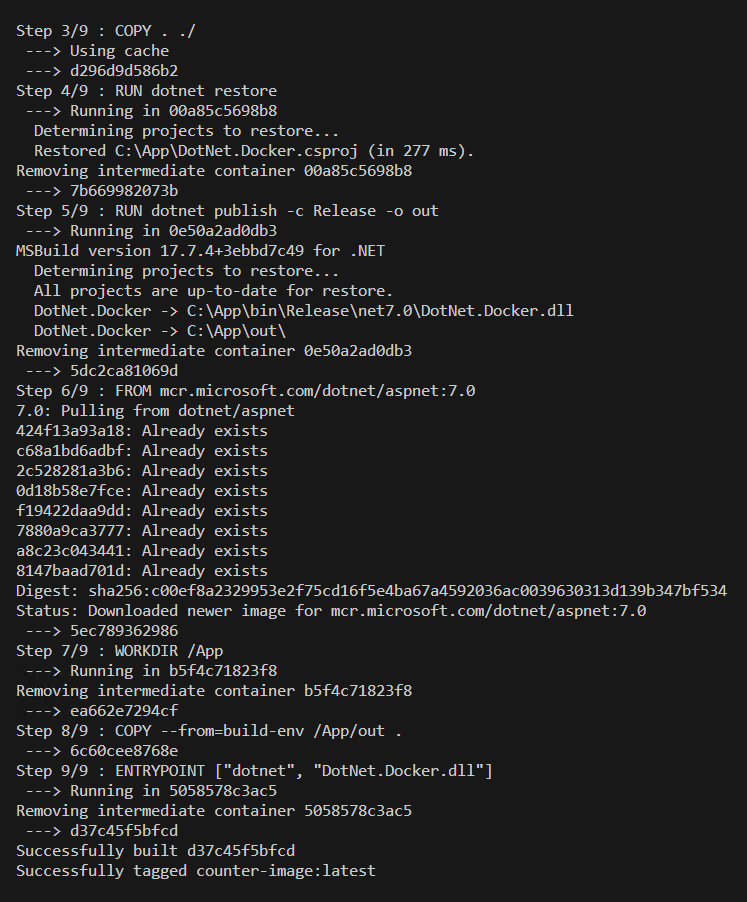
docker build -t counter-image -f Dockerfile .

Docker will process each line in the Dockerfile. The . in the docker build command sets the build context of the image. The **-f** switch is the **path** to the Dockerfile. This command builds the image and creates a local repository named **counter-image** that points to that image.

The COPY command tells Docker to copy the specified folder on your computer to a folder in the container. In this example, the publish folder is copied to a folder named App/out in the container.

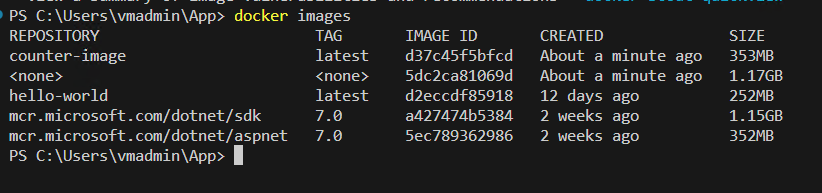
The WORKDIR command changes the **current directory** inside of the container to App.

The next command, ENTRYPOINT, tells Docker to configure the container to run as an executable. When the container starts, the ENTRYPOINT command runs. When this command ends, the container will automatically stop.

* **Step3:** After this command finishes, run docker images to see a list of images installed:

docker images



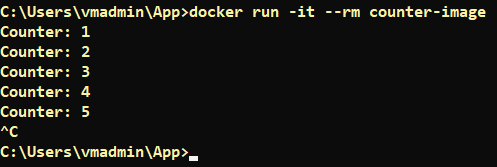
# Create a container

Now that you have an image that contains your app, you can create a container. You can create a container in two ways. First, create a new container that is stopped.

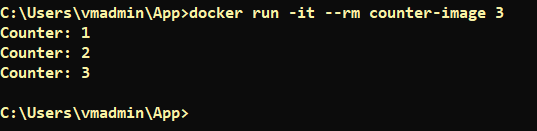
### Single run

Docker provides the **docker run** command to create and run the container as a single command. This command eliminates the need to run **docker create** and then **docker start**. You can also set this command to automatically delete the container when the container stops. For example, use docker run **-it --rm** to do two things, first, automatically use the current terminal to connect to the container, and then when the container finishes, remove it:

docker run -it --rm counter-image



The container also passes parameters into the execution of the .NET app. To instruct the .NET app to count only to three, pass in 3.



## Create a Stopped container

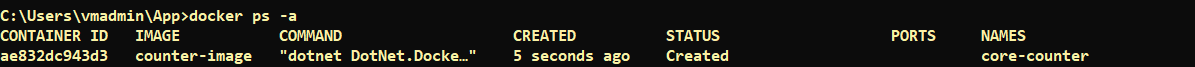
Use docker create command to creates a container based on the **counter-image** image.

docker create --name core-counter counter-image



To see a list of all containers, use the docker ps -a command:

docker ps -a

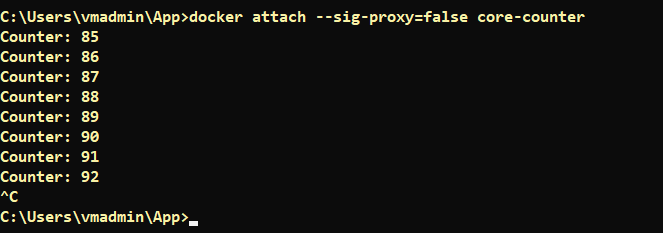


docker start core-counter



docker attach --sig-proxy=false core-counter

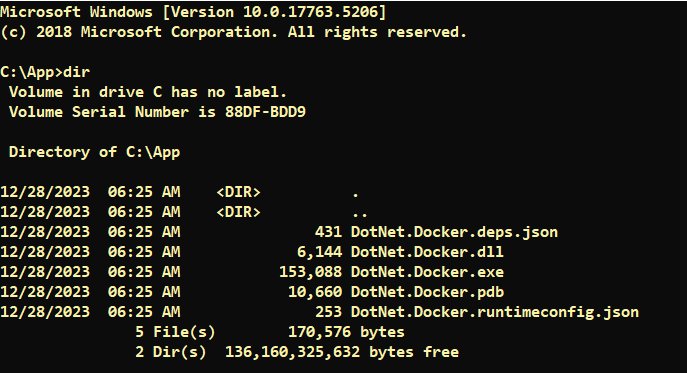
The --sig-proxy=false parameter ensures that Ctrl+C won't stop the process in the container.



docker exec -it core-counter cmd.exe

To access container command prompt

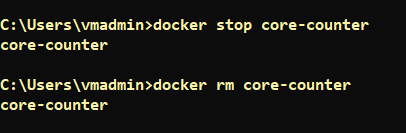




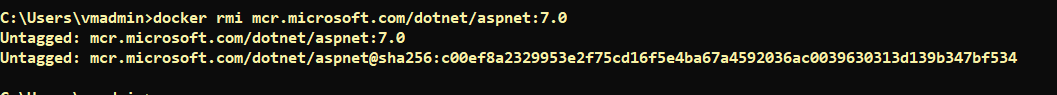
T

docker stop core-counter

docker rm core-counter



docker rmi mcr.microsoft.com/dotnet/aspnet:7.0



# Access File Share in Windows Container

