<https://www.c-sharpcorner.com/article/create-docker-image-and-hosting-for-simple-web-application-using-visual-studio/>

<https://codefresh.io/blog/docker-images-net-core/>

https://www.edureka.co/blog/docker-commands/

Docker is the platform for deploying and building the applications which delivers the application in the packages over the operating system level virtualization. It is a Linux-based open-source platform. It is a way of hiding or protecting your code which you need to run applications, ASP.NET core is one of them.

Docker requires the following components to run the application.

* Image
* Container

docker image

Docker image is the set of configuration and instructions to create the docker container, an image is created during building the application based on steps defined in the docker file of your application. The docker image is the read-only file which cannot be modified once it's created, but we can delete the image from docker.

Container

The Container is an independent isolated process of an operating system that has its own networking and file system to run the image or application. The container is created on the docker engine based on the image configuration.

A Docker container image is a standalone, lightweight package that can be executed and contains all the requirements you need to run an application, such as: code, runtime, libraries, and settings. The image can then be pushed to a container registry and pulled to your server to run as a container.

The container images were created in the following stages.

* Build
* Publish
* Final Image

Docker can be used for the following:

* Build
* Host
* Scale .NET applications with ease

**Create a Dockerfile in your project folder**  
The [Dockerfile](https://docs.docker.com/samples/dotnetcore/) is essentially a recipe for creating your Docker image and is added to the project’s root. It includes the necessary commands for a user to build an image when executing

* docker build

Dockerfile

* FROM mcr.microsoft.com/dotnet/core/aspnet:3.1-buster-slim AS base
* WORKDIR /app
* EXPOSE 80
* FROM mcr.microsoft.com/dotnet/core/sdk:3.1-buster AS build
* WORKDIR /src
* COPY ["DockerDemo.csproj", ""]
* RUN dotnet restore "./DockerDemo.csproj"
* COPY . .
* WORKDIR "/src/."
* RUN dotnet build "DockerDemo.csproj" -c Release -o /app/build
* FROM build AS publish
* RUN dotnet publish "DockerDemo.csproj" -c Release -o /app/publish
* FROM base AS final
* WORKDIR /app
* COPY --from=publish /app/publish .
* ENTRYPOINT ["dotnet", "DockerDemo.dll"]

**Create a .dockerignore file**

You can also see .dockerignore file in project solution explorer. It is acting like a git ignore file, ignoring a few files while creating a docker image.

Similar to a .gitignore file a [.dockerignore](https://docs.docker.com/engine/reference/builder/#dockerignore-file) file allows you to mention a list of files or directories that you want to ignore while building the Docker image. This is essential because it helps reduce the size of an image and speeds up the docker building process.

Note: We also have an excellent article reviewing the [.dockerignore](https://codefresh.io/docker-tutorial/not-ignore-dockerignore-2/) file that you might enjoy

.dockerignore

class=""> \*\*/.classpath

\*\*/.dockerignore

\*\*/.env

\*\*/.git

\*\*/.gitignore

\*\*/.project

\*\*/.settings

\*\*/.toolstarget

\*\*/.vs

\*\*/.vscode

\*\*/\*.\*proj.user

\*\*/\*.dbmdl

\*\*/\*.jfm

\*\*/azds.yaml

\*\*/bin

\*\*/charts

\*\*/docker-compose\*

\*\*/Dockerfile\*

\*\*/node\_modules

\*\*/npm-debug.log

\*\*/obj

\*\*/secrets.dev.yaml

\*\*/values.dev.yaml

LICENSE

README.md

**Build Docker image and start container**  
The [docker build](https://docs.docker.com/engine/reference/commandline/build/) builds a Docker image from the Dockerfile and a [“context”](https://docs.docker.com/engine/reference/commandline/build/#examples). The context is a set of files located in a specified PATH or URL.

Open the terminal or command prompt and navigate to your project folder. Use the following command to build your Docker image:

docker build -t dockerdemo .

**Command: docker build -t dockerhubid/dockerdemowebapp .**

* -t --> means tagging this image with the following name.
* dockerhubid --> https://hub.docker.com/ // Here you can create docker hubid.
* dockerhubid/dockerdemowebapp --> You can give any name in that place but when you try to host this docker container into Azure or any other cloud service means you should move this docker container image into docker hub and then only you can deploy.

* By projectname, it means the full name that we have given.

* Finally we have . (dot) symbol which is important with space. We have to follow the docker principle.

Once the docker image has been created successfully, you can enter the below command to see the images.

**command: docker images**

There you can see your recently created images.

**Create and run container**

So, we have created the docker image. Now we need to run this image locally. To do that we need a container to run this image. This process can be achieved by running the below command.

The [docker run](https://docs.docker.com/engine/reference/commandline/run/)command creates a new container and runs the Docker image.

Open the terminal or command prompt and use the following command to run your Docker image:

docker run -d -p 8080:80 --name myapp dockerdemo

**command: docker run -p 9090:80 dockerhubid/projectname**

* -p --> It means port mapping.
* 9090 --> It is our random port number to run our application (We can give our own).
* 80 --> This one is we exposed 80 port while creating an image. So that port number we have to map with container that we created now.

Check that the container was created and is running with the command:

docker ps

This returns a lot of helpful information, including your container ID, the registry location, and even the command used to run it.

docker image list

The above command show a list of all the installed images, with the latest at the top.

### FROM

The FROM instruction sets the container image that will be used during the new image creation process.

### WORKDIR

The WORKDIR instruction sets a working directory for other Dockerfile instructions, such as RUN, CMD, and also the working directory for running instances of the container image.

### COPY

The COPY instruction copies files and directories to the container's file system. The files and directories must be in a path relative to the Dockerfile.

### RUN

The RUN instruction specifies commands to be run, and captured into the new container image. These commands can include items such as installing software, creating files and directories, and creating environment configuration.

### ADD

The ADD instruction is like the COPY instruction, but with even more capabilities. In addition to copying files from the host into the container image, the ADD instruction can also copy files from a remote location with a URL specification.

### CMD

The CMD instruction sets the default command to be run when deploying an instance of the container image.

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build

In the preceding Dockerfile, the \*.csproj files are copied and restored as distinct layers. When the docker build command builds an image, it uses a built-in cache. If the \*.csproj files haven't changed since the docker build command last ran, the dotnet restore command doesn't need to run again.

**A few commands in Docker**

* docker version - Version
* docker info - Provides information about installed docker
* docker ps - List of containers
* docker images - List of images
* docker start containerid - Starts the given container
* docker stop containerid - Stop the given container