Docker images for ASP.NET Core

05/12/2020 • 5 minutes to read • +5

In this article

This tutorial shows how to run an ASP.NET Core app in Docker containers.

In this tutorial, you:

- ✓ Learn about Microsoft .NET Core Docker images
- ✓ Download an ASP.NET Core sample app
- ✓ Run the sample app locally
- ✓ Run the sample app in Linux containers
- ✓ Run the sample app in Windows containers
- ✓ Build and deploy manually

ASP.NET Core Docker images

For this tutorial, you download an ASP.NET Core sample app and run it in Docker containers. The sample works with both Linux and Windows containers.

The sample Dockerfile uses the Docker multi-stage build feature to build and run in different containers. The build and run containers are created from images that are provided in Docker Hub by Microsoft:

- `dotnet/core/sdk`

  The sample uses this image for building the app. The image contains the .NET

Core SDK, which includes the Command Line Tools (CLI). The image is optimized for local development, debugging, and unit testing. The tools installed for development and compilation make this a relatively large image.

- `dotnet/core/aspnet`

  The sample uses this image for running the app. The image contains the ASP.NET Core runtime and libraries and is optimized for running apps in production. Designed for speed of deployment and app startup, the image is relatively small, so network performance from Docker Registry to Docker host is optimized. Only the binaries and content needed to run an app are copied to the container. The contents are ready to run, enabling the fastest time from `Docker run` to app startup. Dynamic code compilation isn't needed in the Docker model.

Prerequisites

- .NET Core SDK 3.0
- Docker client 18.03 or later
  - Linux distributions
    - CentOS
    - Debian
    - Fedora
    - Ubuntu
  - macOS
  - Windows
- Git

Download the sample app

- Download the sample by cloning the .NET Core Docker repository:

  Console Copy

  ```
  git clone https://github.com/dotnet/dotnet-docker
  ```

Run the app locally

- Navigate to the project folder at *dotnet-docker/samples/aspnetapp/aspnetapp*.
- Run the following command to build and run the app locally:

.NET Core CLI Copy

```
dotnet run
```

- Go to `http://localhost:5000` in a browser to test the app.
- Press Ctrl+C at the command prompt to stop the app.

Run in a Linux container

- In the Docker client, switch to Linux containers.
- Navigate to the Dockerfile folder at *dotnet-docker/samples/aspnetapp*.
- Run the following commands to build and run the sample in Docker:

  Console Copy

  ```
  docker build -t aspnetapp .
  docker run -it --rm -p 5000:80 --name aspnetcore_sample aspnetapp
  ```

  The `build` command arguments:
  - Name the image aspnetapp.
  - Look for the Dockerfile in the current folder (the period at the end).

  The run command arguments:
  - Allocate a pseudo-TTY and keep it open even if not attached. (Same effect as `--interactive --tty`.)
  - Automatically remove the container when it exits.
  - Map port 5000 on the local machine to port 80 in the container.
  - Name the container aspnetcore_sample.
  - Specify the aspnetapp image.
- Go to `http://localhost:5000` in a browser to test the app.

Run in a Windows container

- In the Docker client, switch to Windows containers.

Navigate to the docker file folder at `dotnet-docker/samples/aspnetapp`.

- Run the following commands to build and run the sample in Docker:

  Console Copy

```
docker build -t aspnetapp .
docker run -it --rm --name aspnetcore_sample aspnetapp
```

- For Windows containers, you need the IP address of the container (browsing to `http://localhost:5000` won't work):
  - Open up another command prompt.
  - Run `docker ps` to see the running containers. Verify that the "aspnetcore_sample" container is there.
  - Run `docker exec aspnetcore_sample ipconfig` to display the IP address of the container. The output from the command looks like this example:

Console Copy

```
Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : contoso.com
   Link-local IPv6 Address . . . . . : fe80::1967:6598:124:cfa3%4
   IPv4 Address. . . . . . . . . . . : 172.29.245.43
   Subnet Mask . . . . . . . . . . . : 255.255.240.0
   Default Gateway . . . . . . . . . : 172.29.240.1
```

- Copy the container IPv4 address (for example, 172.29.245.43) and paste into the browser address bar to test the app.

Build and deploy manually

In some scenarios, you might want to deploy an app to a container by copying to it the application files that are needed at run time. This section shows how to deploy manually.

- Navigate to the project folder at *dotnet-docker/samples/aspnetapp/aspnetapp*.
- Run the dotnet publish command:

.NET Core CLI Copy

```
dotnet publish -c Release -o published
```

The command arguments:
  - Build the application in release mode (the default is debug mode).
  - Create the files in the *published* folder.

- Run the application.
  - Windows:

    .NET Core CLI Copy

    ```
    dotnet published\aspnetapp.dll
    ```

  - Linux:

    .NET Core CLI Copy

    ```
    dotnet published/aspnetapp.dll
    ```

- Browse to `http://localhost:5000` to see the home page.

To use the manually published application within a Docker container, create a new Dockerfile and use the `docker build .` command to build the container.

Dockerfile Copy

```
FROM mcr.microsoft.com/dotnet/core/aspnet:3.0 AS runtime
WORKDIR /app
COPY published/aspnetapp.dll ./
ENTRYPOINT ["dotnet", "aspnetapp.dll"]
```

The Dockerfile

Here's the *Dockerfile* used by the `docker build` command you ran earlier. It uses `dotnet publish` the same way you did in this section to build and deploy.

Dockerfile Copy

```
FROM mcr.microsoft.com/dotnet/core/sdk:3.0 AS build
WORKDIR /app

# copy csproj and restore as distinct layers
COPY *.sln .
COPY aspnetapp/*.csproj ./aspnetapp/
RUN dotnet restore

# copy everything else and build app
COPY aspnetapp/. ./aspnetapp/
WORKDIR /app/aspnetapp
RUN dotnet publish -c Release -o out

FROM mcr.microsoft.com/dotnet/core/aspnet:3.0 AS runtime
```

```
WORKDIR /app
COPY --from=build /app/aspnetapp/out ./
ENTRYPOINT ["dotnet", "aspnetapp.dll"]
```

As noted 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. Instead, the built-in cache for the corresponding `dotnet restore` layer is reused. For more information, see Best practices for writing Dockerfiles.

Additional resources

- Docker build command
- Docker run command
- ASP.NET Core Docker sample (The one used in this tutorial.)
- Configure ASP.NET Core to work with proxy servers and load balancers
- Working with Visual Studio Docker Tools
- Debugging with Visual Studio Code
- GC using Docker and small containers

Next steps

The Git repository that contains the sample app also includes documentation. For an overview of the resources available in the repository, see the README file. In particular, learn how to implement HTTPS:

Developing ASP.NET Core Applications with Docker over HTTPS