

# Visual Studio Container Tools with ASP.NET Core

Article • 10/30/2023

Visual Studio 2017 and later versions support building, debugging, and running containerized ASP.NET Core apps targeting .NET Core. Both Windows and Linux containers are supported.

[View or download sample code](#) (how to download)

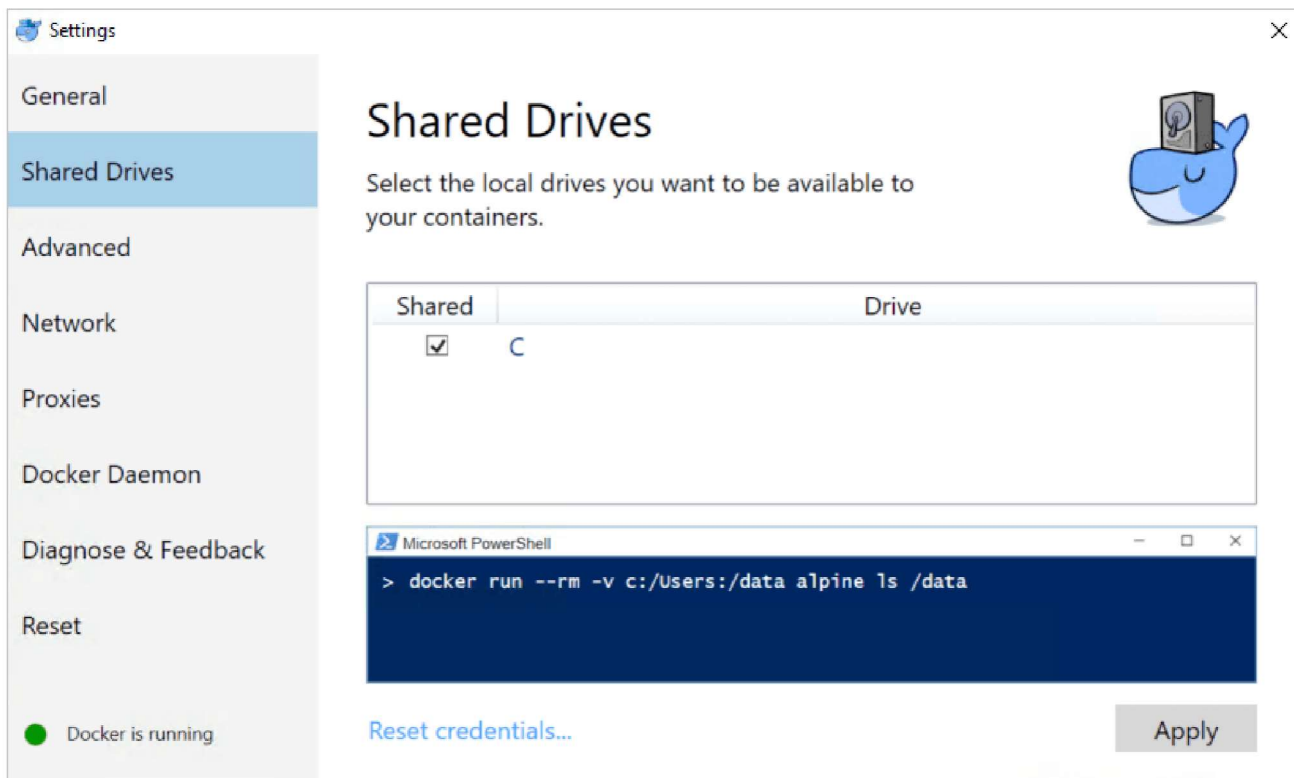
## Prerequisites

- [Docker for Windows](#)
- [Visual Studio 2019](#) with the .NET Core cross-platform development workload

## Installation and setup

For Docker installation, first review the information at [Docker for Windows: What to know before you install](#) . Next, install [Docker For Windows](#) .

**Shared Drives** in Docker for Windows must be configured to support volume mapping and debugging. Right-click the System Tray's Docker icon, select **Settings**, and select **Shared Drives**. Select the drive where Docker stores files. Click **Apply**.



### 💡 Tip

Visual Studio 2017 versions 15.6 and later prompt when **Shared Drives** aren't configured.

## Add a project to a Docker container

To containerize an ASP.NET Core project, the project must target .NET Core. Both Linux and Windows containers are supported.

When adding Docker support to a project, choose either a Windows or a Linux container. The Docker host must be running the same container type. To change the container type in the running Docker instance, right-click the System Tray's Docker icon and choose **Switch to Windows containers...** or **Switch to Linux containers....**

## New app

When creating a new app with the **ASP.NET Core Web Application** project templates, select the **Enable Docker Support** checkbox:

☒ **Enable Docker Support**

OS: Windows

Requires [Docker for Windows](#)

Docker support can also be enabled later [Learn more](#)

If the target framework is .NET Core, the **OS** drop-down allows for the selection of a container type.

## Existing app

For ASP.NET Core projects targeting .NET Core, there are two options for adding Docker support via the tooling. Open the project in Visual Studio, and choose one of the following options:

- Select **Docker Support** from the **Project** menu.
- Right-click the project in **Solution Explorer** and select **Add > Docker Support**.

The Visual Studio Container Tools don't support adding Docker to an existing ASP.NET Core project targeting .NET Framework.

## Dockerfile overview

A *Dockerfile*, the recipe for creating a final Docker image, is added to the project root. Refer to [Dockerfile reference](#) for an understanding of the commands within it. This particular *Dockerfile* uses a [multi-stage build](#) with four distinct, named build stages:

### Dockerfile

```
FROM mcr.microsoft.com/dotnet/core/aspnet:2.1 AS base
WORKDIR /app
EXPOSE 59518
EXPOSE 44364

FROM mcr.microsoft.com/dotnet/core/sdk:2.1 AS build
WORKDIR /src
COPY HelloDockerTools/HelloDockerTools.csproj HelloDockerTools/
RUN dotnet restore HelloDockerTools/HelloDockerTools.csproj
COPY . .
WORKDIR /src/HelloDockerTools
RUN dotnet build HelloDockerTools.csproj -c Release -o /app

FROM build AS publish
```

```
RUN dotnet publish HelloDockerTools.csproj -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "HelloDockerTools.dll"]
```

The preceding *Dockerfile* image includes the ASP.NET Core runtime and NuGet packages. The packages are just-in-time (JIT) compiled to improve startup performance.

When the new project dialog's **Configure for HTTPS** checkbox is checked, the *Dockerfile* exposes two ports. One port is used for HTTP traffic; the other port is used for HTTPS. If the checkbox isn't checked, a single port (80) is exposed for HTTP traffic.

## Add container orchestrator support to an app

Visual Studio 2017 versions 15.7 or earlier support [Docker Compose](#) as the sole container orchestration solution. The Docker Compose artifacts are added via **Add > Docker Support**.

Visual Studio 2017 versions 15.8 or later add an orchestration solution only when instructed. Right-click the project in **Solution Explorer** and select **Add > Container Orchestrator Support**. The following choices are available:

- [Docker Compose](#)
- [Service Fabric](#)
- [Kubernetes/Helm](#)

## Docker Compose

The Visual Studio Container Tools add a *docker-compose* project to the solution with the following files:

- *docker-compose.dcproj*: The file representing the project. Includes a `<DockerTargetOS>` element specifying the OS to be used.
- *.dockerignore*: Lists the file and directory patterns to exclude when generating a build context.
- *docker-compose.yml*: The base [Docker Compose](#) file used to define the collection of images built and run with `docker-compose build` and `docker-compose run`, respectively.

- *docker-compose.override.yml*: An optional file, read by Docker Compose, with configuration overrides for services. Visual Studio executes `docker-compose -f "docker-compose.yml" -f "docker-compose.override.yml"` to merge these files.

The *docker-compose.yml* file references the name of the image that's created when the project runs:

YAML

```
version: '3.4'

services:
  heliodockertools:
    image: ${DOCKER_REGISTRY}heliodockertools
    build:
      context: .
      dockerfile: HelloDockerTools/Dockerfile
```

In the preceding example, `image: heliodockertools` generates the image `heliodockertools:dev` when the app runs in **Debug** mode. The `heliodockertools:latest` image is generated when the app runs in **Release** mode.

Prefix the image name with the [Docker Hub](#) username (for example, `dockerhubusername/heliodockertools`) if the image is pushed to the registry. Alternatively, change the image name to include the private registry URL (for example, `privateregistry.domain.com/heliodockertools`) depending on the configuration.

If you want different behavior based on the build configuration (for example, Debug or Release), add configuration-specific *docker-compose* files. The files should be named according to the build configuration (for example, *docker-compose.vs.debug.yml* and *docker-compose.vs.release.yml*) and placed in the same location as the *docker-compose.override.yml* file.

Using the configuration-specific override files, you can specify different configuration settings (such as environment variables or entry points) for Debug and Release build configurations.

For Docker Compose to display an option to run in Visual Studio, the docker project must be the startup project.

## Service Fabric



In addition to the base [Prerequisites](#), the [Service Fabric](#) orchestration solution demands the following prerequisites:

- [Microsoft Azure Service Fabric SDK](#) version 2.6 or later
- Visual Studio's **Azure Development** workload

Service Fabric doesn't support running Linux containers in the local development cluster on Windows. If the project is already using a Linux container, Visual Studio prompts to switch to Windows containers.

The Visual Studio Container Tools do the following tasks:

- Adds a `<project_name>Application` **Service Fabric Application** project to the solution.
- Adds a *Dockerfile* and a *.dockerignore* file to the ASP.NET Core project. If a *Dockerfile* already exists in the ASP.NET Core project, it's renamed to *Dockerfile.original*. A new *Dockerfile*, similar to the following, is created:

Dockerfile

```
# See https://aka.ms/containerimagehelp for information on how to use
Windows Server 1709 containers with Service Fabric.
# FROM microsoft/aspnetcore:2.0-nanoserver-1709
FROM microsoft/aspnetcore:2.0-nanoserver-sac2016
ARG source
WORKDIR /app
COPY ${source:-obj/Docker/publish} .
ENTRYPOINT ["dotnet", "HelloDockerTools.dll"]
```

- Adds an `<IsServiceFabricServiceProject>` element to the ASP.NET Core project's *.csproj* file:

XML

```
<IsServiceFabricServiceProject>True</IsServiceFabricServiceProject>
```

- Adds a *PackageRoot* folder to the ASP.NET Core project. The folder includes the service manifest and settings for the new service.

For more information, see [Deploy a .NET app in a Windows container to Azure Service Fabric](#).

# Debug

Select **Docker** from the debug drop-down in the toolbar, and start debugging the app. The **Docker** view of the **Output** window shows the following actions taking place:

- The *2.1-aspnetcore-runtime* tag of the *microsoft/dotnet* runtime image is acquired (if not already in the cache). The image installs the ASP.NET Core and .NET Core runtimes and associated libraries. It's optimized for running ASP.NET Core apps in production.
- The `ASPNETCORE_ENVIRONMENT` environment variable is set to `Development` within the container.
- Two dynamically assigned ports are exposed: one for HTTP and one for HTTPS. The port assigned to localhost can be queried with the `docker ps` command.
- The app is copied to the container.
- The default browser is launched with the debugger attached to the container using the dynamically assigned port.

The resulting Docker image of the app is tagged as *dev*. The image is based on the *2.1-aspnetcore-runtime* tag of the *microsoft/dotnet* base image. Run the `docker images` command in the **Package Manager Console** (PMC) window. The images on the machine are displayed:

Console				
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hellodockertools	dev	d72ce0f1dfe7	30 seconds ago	255MB
microsoft/dotnet	2.1-aspnetcore-runtime	fcc3887985bb	6 days ago	255MB

## ❗ Note

The *dev* image lacks the app contents, as **Debug** configurations use volume mounting to provide the iterative experience. To push an image, use the **Release** configuration.

Run the `docker ps` command in PMC. Notice the app is running using the container:

Console			
CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
baf9a678c88d	hellodockertools:dev	"C:\\\\remote_debugge..."	21 seconds

```
ago      Up 19 seconds      0.0.0.0:37630->80/tcp
dockercompose4642749010770307127_hellodockertools_1
```

## Edit and continue

Changes to static files and Razor views are automatically updated without the need for a compilation step. Make the change, save, and refresh the browser to view the update.

Code file modifications require compilation and a restart of Kestrel within the container. After making the change, use `CTRL+F5` to perform the process and start the app within the container. The Docker container isn't rebuilt or stopped. Run the `docker ps` command in PMC. Notice the original container is still running as of 10 minutes ago:

Console

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
ba9a678c88d	hellodockertools:dev	"C:\\remote_debugge..."	10 minutes ago
Up 10 minutes	0.0.0.0:37630->80/tcp		
dockercompose4642749010770307127_hellodockertools_1			

## Publish Docker images

Once the develop and debug cycle of the app is completed, the Visual Studio Container Tools assist in creating the production image of the app. Change the configuration drop-down to **Release** and build the app. The tooling acquires the compile/publish image from Docker Hub (if not already in the cache). An image is produced with the *latest* tag, which can be pushed to the private registry or Docker Hub.

Run the `docker images` command in PMC to see the list of images. Output similar to the following is displayed:

Console

REPOSITORY	TAG	IMAGE ID	CREATED
hellodockertools	latest	e3984a64230c	About a minute ago
hellodockertools	dev	d72ce0f1dfe7	4 minutes ago
microsoft/dotnet	2.1-sdk	9e243db15f91	6 days ago



1.7GB

microsoft/dotnet 2.1-aspnetcore-runtime fcc3887985bb 6 days ago  
255MB


### ⓘ Note

The `docker images` command returns intermediary images with repository names and tags identified as `<none>` (not listed above). These unnamed images are produced by the [multi-stage build](#) *Dockerfile*. They improve the efficiency of building the final image—only the necessary layers are rebuilt when changes occur. When the intermediary images are no longer needed, delete them using the [docker rmi](#) command.

There may be an expectation for the production or release image to be smaller in size by comparison to the *dev* image. Because of the volume mapping, the debugger and app were running from the local machine and not within the container. The *latest* image has packaged the necessary app code to run the app on a host machine. Therefore, the delta is the size of the app code.

## Additional resources

- [Container development with Visual Studio](#)
- [Azure Service Fabric: Prepare your development environment](#)
- [Deploy a .NET app in a Windows container to Azure Service Fabric](#)
- [Troubleshoot Visual Studio development with Docker](#)
- [Visual Studio Container Tools GitHub repository](#)
- [GC using Docker and small containers](#)
- [System.IO.IOException: The configured user limit \(128\) on the number of inotify instances has been reached](#)
- [Updates to Docker images](#)

 Collaborate with us on  
GitHub

The source for this content can be found on GitHub, where you can also create and review issues



### ASP.NET Core feedback

ASP.NET Core is an open source project. Select a link to provide feedback:

and pull requests. For more information, see [our contributor guide](#).

 [Open a documentation issue](#)

 [Provide product feedback](#)