# Introduction to .NET and Docker

Article • 01/04/2024

Containers are one of the most popular ways for deploying and hosting cloud applications, with tools like Docker , Kubernetes , and Podman . Many developers choose containers because it's straightforward to package an app with its dependencies and get that app to reliably run on any container host. There's extensive support for using .NET with containers .

Docker provides a great overview of containers. Docker Desktop: Community Edition is a good tool to use for using containers on developer desktop machine.

## .NET images

Official .NET container images are published to the Microsoft Artifact Registry and are discoverable on the Docker Hub . There are runtime images for production and SDK images for building your code, for Linux (Alpine, Debian, Ubuntu, Mariner) and Windows. For more information, see .NET container images.

.NET images are regularly updated whenever a new .NET patch is published or when an operating system base image is updated.

Chiseled container images are Ubuntu container images with a minimal set of components required by the .NET runtime. These images are ~100 MB smaller than the regular Ubuntu images and have fewer CVEs since they have fewer components. In particular, they don't contain a shell or package manager, which significantly improves their security profile. They also include a non-root user and are configured with that user enabled.

## Building container images

You can build a container image with a **Dockerfile** or rely on the .NET SDK to produce an image. For samples on building images, see dotnet/dotnet-docker and dotnet/sdk-container-builds .

The following example demonstrates building and running a container image in a few quick steps (supported with .NET 8 and .NET 7.0.300).

Bash

```
$ dotnet new webapp -o webapp
$ cd webapp/
$ dotnet publish -t:PublishContainer
MSBuild version 17.8.3+195e7f5a3 for .NET
  Determining projects to restore...
  All projects are up-to-date for restore.
  webapp -> /home/rich/webapp/bin/Release/net8.0/webapp.dll
  webapp -> /home/rich/webapp/bin/Release/net8.0/publish/
  Building image 'webapp' with tags 'latest' on top of base image
'mcr.microsoft.com/dotnet/aspnet:8.0'.
  Pushed image 'webapp:latest' to local registry via 'docker'.
$ docker run --rm -d -p 8000:8080 webapp
7c7ad33409e52ddd3a9d330902acdd49845ca4575e39a6494952b642e584016e
$ curl -s http://localhost:8000 | grep ASP.NET
    <p>Learn about <a href="https://learn.microsoft.com/aspnet/core">building
Web apps with ASP.NET Core</a>.</p>
$ docker ps
CONTAINER ID   IMAGE     COMMAND               CREATED              STATUS
PORTS                                       NAMES
7c7ad33409e5   webapp    "dotnet webapp.dll"   About a minute ago   Up About a
minute   0.0.0.0:8000->8080/tcp, :::8000->8080/tcp   jovial_shtern
$ docker kill 7c7ad33409e5
```

docker init is a new option for developers wanting to use Dockerfiles.

# Ports

Port mapping is a key part of using containers. Ports must be published outside the container in order to respond to external web requests. ASP.NET Core container images changed in .NET 8 to listen on port `8080`, by default. .NET 6 and 7 listen on port `80`.

In the prior example with `docker run`, the host port `8000` is mapped to the container port `8080`. Kubernetes works in a similar way.

The `ASPNETCORE_HTTP_PORTS`, `ASPNETCORE_HTTPS_PORTS`, and `ASPNETCORE_URLS` environment variables can be used to configure this behavior.

# Users

Starting with .NET 8, all images include a non-root user called `app`. By default, chiseled images are configured with this user enabled. The publish app as .NET container feature

(demonstrated in the Building container images section) also configures images with this user enabled by default. In all other scenarios, the `app` user can be set manually, for example with the `USER` *Dockerfile* instruction. If an image has been configured with `app` and commands need to run as `root`, then the `USER` instruction can be used to set to the user to `root`.

# Staying informed

Container-related news is posted to dotnet/dotnet-docker discussions and to the .NET Blog "containers" category.

# Azure services

Various Azure services support containers. You create a Docker image for your application and deploy it to one of the following services:

- Azure Kubernetes Service (AKS)
  Scale and orchestrate Windows & Linux containers using Kubernetes.
- Azure App Service
  Deploy web apps or APIs using containers in a PaaS environment.
- Azure Container Apps
  Run your container workloads without managing servers, orchestration, or infrastructure and leverage native support for Dapr and KEDA for observability and scaling to zero.
- Azure Container Instances
  Create individual containers in the cloud without any higher-level management services.
- Azure Batch
  Run repetitive compute jobs using containers.
- Azure Service Fabric
  Lift, shift, and modernize .NET applications to microservices using Windows & Linux containers.

- Azure Container Registry
  Store and manage container images across all types of Azure deployments.

# Next steps

- Learn how to containerize a .NET Core application.
- Learn how to containerize an ASP.NET Core application.
- Try the Learn ASP.NET Core Microservice tutorial.
- Learn about Container Tools in Visual Studio

**Collaborate with us on GitHub**

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

## .NET feedback

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

Open a documentation issue

Provide product feedback