Choosing Between .NET Core and .NET Framework for Docker Containers

Contents

[General Guidance 2](#_Toc59290174)

[When to choose .NET Core for Docker Containers 2](#_Toc59290175)

[Developing and deploying cross platform 2](#_Toc59290176)

[Using containers for new projects 2](#_Toc59290177)

[Create and deploy microservices on containers 2](#_Toc59290178)

[Deploying high density in scalable systems 2](#_Toc59290179)

[When to choose .NET Framework for Docker containers 2](#_Toc59290180)

[Migrating existing applications directly to a Windows Server container 2](#_Toc59290181)

[Using third-party .NET libraries or NuGet packages no available for .NET Core 2](#_Toc59290182)

[Using .NET technologies, no available for .NET Core 2](#_Toc59290183)

[Using a platform or API that does not support .NET Core 2](#_Toc59290184)

[Decision table: .NET frameworks to use Docker 2](#_Toc59290185)

[What OS to target with .NET containers 2](#_Toc59290186)

[Official .NET Docker images 2](#_Toc59290187)

[.NET Core and Docker image optimizations for development versus production 2](#_Toc59290188)

[During development and build 2](#_Toc59290189)

[In Production 2](#_Toc59290190)

# General Guidance

Use .NET Core with Linux or Windows Containers, for your containerized Docker server application when:

1. You have cross-platform needs.
2. Your application architecture is based on microservices.
3. You need to start containers fast and want a small footprint per container to achieve better density or more containers per hardware unit in order to lower your costs.

Use .NET Framework for your containerized Docker server application when:

1. Your application currently uses .NET Framework and has strong dependencies on Windows.
2. You need to use Windows APIs that are not supported by .NET Core.
3. You need to use third-party .NET libraries or NuGet packages that are not available for .NET Core.

# When to choose .NET Core for Docker Containers

***Modularity and Lightweight nature***

When you deploy and start a container, its image is far smaller with .NET Core than with .NET Framework. In contrary, to use .NET Framework for a container, you must base your image on the Windows Server Core Image, which is a lot heavier than the Windows Nano Server or Linux images that you use for .NET Core.

Also, .NET Core is cross-platform, so you can deploy server apps with Linux or Windows container images. However, if you are using the traditional .NET Framework, you can only deploy images based on Windows Server Core.

## Developing and deploying cross platform

If your goal is to have an application that can run on multiple platforms supported by Docker, the right choice is .NET Core, because .NET Framework only supports Windows.

## Using containers for new projects

When you create and deploy a container, its image is far smaller with .Net Core than with .NET Framework.

## Create and deploy microservices on containers

## Deploying high density in scalable systems

# When to choose .NET Framework for Docker containers

## Migrating existing applications directly to a Windows Server container

## Using third-party .NET libraries or NuGet packages no available for .NET Core

## Using .NET technologies, no available for .NET Core

## Using a platform or API that does not support .NET Core

# Decision table: .NET frameworks to use Docker

# What OS to target with .NET containers

# Official .NET Docker images

## .NET Core and Docker image optimizations for development versus production

### During development and build

### In Production