.NET Core vs. .NET Framework for server apps

04/28/2020 • 6 minutes to read • +18

In this article

There are two supported .NET implementations for building server-side apps: .NET Framework and .NET Core. Both share many of the same components and you can share code across the two. However, there are fundamental differences between the two and your choice depends on what you want to accomplish. This article provides guidance on when to use each.

Use .NET Core for your server application when:

- You have cross-platform needs.
- You're targeting microservices.
- You're using Docker containers.
- You need high-performance and scalable systems.
- You need side-by-side .NET versions per application.

Use .NET Framework for your server application when:

- Your app currently uses .NET Framework (recommendation is to extend instead of migrating).
- Your app uses third-party .NET libraries or NuGet packages not available for .NET Core.
- Your app uses .NET technologies that aren't available for .NET Core.
- Your app uses a platform that doesn’t support .NET Core. Windows, macOS, and Linux support .NET Core.

When to choose .NET Core

The following sections give a more detailed explanation of the previously stated reasons for picking .NET Core.

Cross-platform needs

If your application (web/service) needs to run on multiple platforms (Windows, Linux, and macOS), use .NET Core.

.NET Core supports the previously mentioned operating systems as your development workstation. Visual Studio provides an Integrated Development Environment (IDE) for Windows and macOS. You can also use Visual Studio Code, which runs on macOS, Linux, and Windows. Visual Studio Code supports .NET Core, including IntelliSense and debugging. Most third-party editors, such as Sublime, Emacs, and VI, work with .NET Core. These third-party editors get editor IntelliSense using Omnisharp. You can also avoid any code editor and directly use the .NET Core CLI, available for all supported platforms.

Microservices architecture

A microservices architecture allows a mix of technologies across a service boundary. This technology mix enables a gradual embrace of .NET Core for new microservices that work with other microservices or services. For example, you can mix microservices or services developed with .NET Framework, Java, Ruby, or other monolithic technologies.

There are many infrastructure platforms available. Azure Service Fabric is designed for large and complex microservice systems. Azure App Service is a good choice for stateless microservices. Microservices alternatives based on Docker fit any kind of microservices approach, as explained in the Containers section. All these platforms support .NET Core and make them ideal for hosting your microservices.

For more information about microservices architecture, see .NET Microservices. Architecture for Containerized .NET Applications.

Containers

Containers are commonly used in conjunction with a microservices architecture. Containers can also be used to containerize web apps or services that follow any architectural pattern. .NET Framework can be used on Windows containers, but the modularity and lightweight nature of .NET Core makes it a better choice for containers. When creating and deploying a container, the size of its image is much smaller with .NET Core than with .NET Framework. Because it's cross-platform, you can deploy server apps to Linux Docker containers, for example.

Docker containers can be hosted in your own Linux or Windows infrastructure, or in a cloud service such as Azure Kubernetes Service. Azure Kubernetes Service can manage, orchestrate, and scale container-based applications in the cloud.

High-performance and scalable systems

When your system needs the best possible performance and scalability, .NET Core and ASP.NET Core are your best options. High-performance server runtime for Windows Server and Linux makes .NET a top performing web framework on TechEmpower benchmarks.

Performance and scalability are especially relevant for microservices architectures, where hundreds of microservices may be running. With ASP.NET Core, systems run with a much lower number of servers/Virtual Machines (VM). The reduced servers/VMs save costs in infrastructure and hosting.

Side by side .NET versions per application level

To install applications with dependencies on different versions of .NET, we recommend .NET Core. .NET Core supports side-by-side installation of different versions of the .NET Core runtime on the same machine. This side-by-side installation allows multiple services on the same server, each of them on its own version of .NET Core. It also lowers risks and saves money in application upgrades and IT operations.

Side-by-side installation isn't possible with .NET Framework. It's a Windows component, and only one version can exist on a machine at a time. Each version of .NET Framework replaces the previous version. If you install a new app that targets a later version of .NET Framework, you might break existing apps that run on the machine, because the previous version was replaced.

When to choose .NET Framework

.NET Core offers significant benefits for new applications and application patterns. However, .NET Framework continues to be the natural choice for many existing scenarios, and as such, .NET Framework isn't replaced by .NET Core for all server applications.

Current .NET Framework applications

In most cases, you don’t need to migrate your existing applications to .NET Core. Instead, a recommended approach is to use .NET Core as you extend an existing application, such as writing a new web service in ASP.NET Core.

ASP.NET Core on .NET Framework

For information about support for ASP.NET Core on .NET Framework, see .NET Core Support Policy.

Third-party libraries or NuGet packages not available for .NET Core

Libraries are quickly embracing .NET Standard. .NET Standard enables sharing code across all .NET implementations, including .NET Core. With .NET Standard 2.0, this is even easier:

- The API surface became much larger.
- It introduced a .NET Framework compatibility mode.

  This compatibility mode allows .NET Standard and .NET Core projects to reference .NET Framework libraries. To learn more about the compatibility mode, see Announcing .NET Standard 2.0.

You need to use .NET Framework only in cases where the libraries or NuGet packages use technologies that aren't available in .NET Standard or .NET Core.

.NET technologies not available for .NET Core

Some .NET Framework technologies aren't available in .NET Core. Some of them might be available in later .NET Core releases. Others don't apply to the new application patterns targeted by .NET Core and may never be available. The following list shows the most common technologies not found in .NET Core:

- ASP.NET Web Forms applications: ASP.NET Web Forms are only available in .NET Framework. ASP.NET Core cannot be used for ASP.NET Web Forms. There are no plans to bring ASP.NET Web Forms to .NET Core.
- ASP.NET Web Pages applications: ASP.NET Web Pages aren't included in ASP.NET Core.
- WCF services implementation. Even when there's a WCF client library to consume WCF services from .NET Core, WCF server implementation is currently only available in .NET Framework. This scenario is not part of the current plan for .NET Core, but it's being considered for the future.
- Workflow-related services: Windows Workflow Foundation (WF), Workflow Services (WCF + WF in a single service), and WCF Data Services (formerly known as "ADO.NET Data Services") are only available in .NET Framework. There are no plans to bring these technologies to .NET Core.
- Language support: Visual Basic and F# are currently supported in .NET Core, but not for all project types. For a list of supported project templates, see Template options for dotnet new.

Platform doesn't support .NET Core

Some Microsoft or third-party platforms don't support .NET Core. Some Azure services provide an SDK not yet available for consumption on .NET Core. This is a transitional circumstance, as all of Azure services use .NET Core. In the meantime, you can use the equivalent REST API instead of the client SDK.

See also

- Choose between ASP.NET and ASP.NET Core
- ASP.NET Core targeting .NET Framework
- Target frameworks
- .NET Core introduction
- Porting from .NET Framework to .NET Core
- Introduction to .NET and Docker
- .NET Components Overview
- .NET Microservices. Architecture for Containerized .NET Applications