

Web server implementations in ASP.NET Core

Article • 05/12/2023

By [Tom Dykstra](#) , [Steve Smith](#) , [Stephen Halter](#) , and [Chris Ross](#)

An ASP.NET Core app runs with an in-process HTTP server implementation. The server implementation listens for HTTP requests and surfaces them to the app as a set of [request features](#) composed into an [HttpContext](#).

Windows

ASP.NET Core ships with the following:

- [Kestrel server](#) is the default, cross-platform HTTP server implementation. Kestrel provides the best performance and memory utilization, but it doesn't have some of the advanced features in HTTP.sys. For more information, see [Kestrel vs. HTTP.sys](#) in the Windows tab.
- IIS HTTP Server is an [in-process server](#) for IIS.
- [HTTP.sys server](#) is a Windows-only HTTP server based on the [HTTP.sys kernel driver](#) and [HTTP Server API](#).

When using [IIS](#) or [IIS Express](#), the app either runs:

- In the same process as the IIS worker process (the [in-process hosting model](#)) with the IIS HTTP Server. *In-process* is the recommended configuration.
- In a process separate from the IIS worker process (the [out-of-process hosting model](#)) with the [Kestrel server](#).

The [ASP.NET Core Module](#) is a native IIS module that handles native IIS requests between IIS and the in-process IIS HTTP Server or Kestrel. For more information, see [ASP.NET Core Module \(ANCM\) for IIS](#).

Kestrel vs. HTTP.sys

Kestrel has the following advantages over HTTP.sys:

- Better performance and memory utilization.
- Cross platform
- Agility, it's developed and patched independent of the OS.
- Programmatic port and TLS configuration

- Extensibility that allows for protocols like [PPv2](#) and alternate transports.

Http.Sys operates as a shared kernel mode component with the following features that kestrel does not have:

- Port sharing
- Kernel mode windows authentication. [Kestrel supports only user-mode authentication.](#)
- Fast proxying via queue transfers
- Direct file transmission
- Response caching

Hosting models

Several hosting models are available:

- Kestrel self-hosting: The Kestrel web server runs without requiring any other external web server such as IIS or HTTP.sys.
- HTTP.sys self-hosting is an alternative to Kestrel. Kestrel is recommended over HTTP.sys unless the app requires features not available in Kestrel.
- IIS in-process hosting: An ASP.NET Core app runs in the same process as its IIS worker process. IIS in-process hosting provides improved performance over IIS out-of-process hosting because requests aren't proxied over the loopback adapter, a network interface that returns outgoing network traffic back to the same machine. IIS handles process management with the [Windows Process Activation Service \(WAS\)](#).
- IIS out-of-process hosting: ASP.NET Core apps run in a process separate from the IIS worker process, and the module handles process management. The module starts the process for the ASP.NET Core app when the first request arrives and restarts the app if it shuts down or crashes. This is essentially the same behavior as seen with apps that run in-process that are managed by the [Windows Process Activation Service \(WAS\)](#). Using a separate process also enables hosting more than one app from the same app pool.

For more information, see the following:

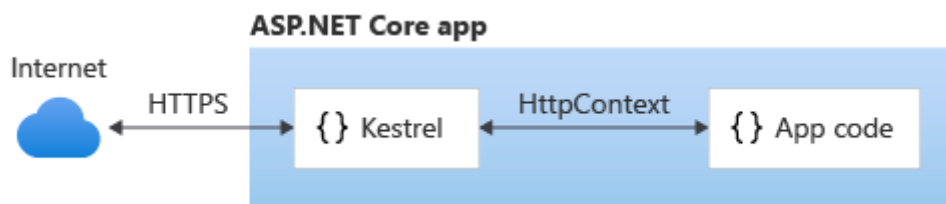
- [Kestrel vs. HTTP.sys](#)
- [Host ASP.NET Core on Windows with IIS](#)
- [ASP.NET Core Module \(ANCM\) for IIS](#)

Kestrel

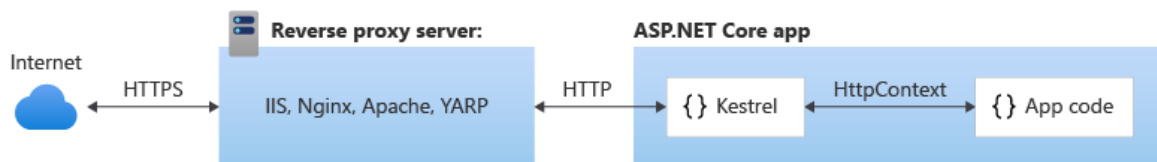
[Kestrel server](#) is the default, cross-platform HTTP server implementation. Kestrel provides the best performance and memory utilization, but it doesn't have some of the advanced features in HTTP.sys. For more information, see [Kestrel vs. HTTP.sys](#) in this document.

Use Kestrel:

- By itself as an edge server processing requests directly from a network, including the Internet.



- With a *reverse proxy server*, such as [Internet Information Services \(IIS\)](#) , [Nginx](#) , or [Apache](#) . A reverse proxy server receives HTTP requests from the Internet and forwards them to Kestrel.



Either hosting configuration—with or without a reverse proxy server—is supported.

For Kestrel configuration guidance and information on when to use Kestrel in a reverse proxy configuration, see [Kestrel web server in ASP.NET Core](#).

Nginx with Kestrel

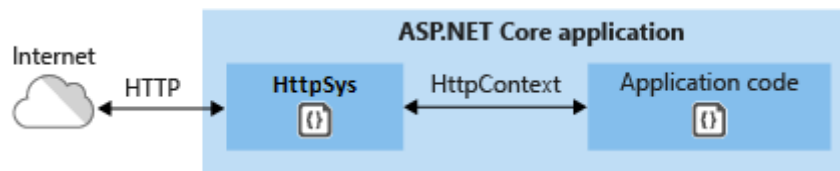
For information on how to use Nginx on Linux as a reverse proxy server for Kestrel, see [Host ASP.NET Core on Linux with Nginx](#).

Apache with Kestrel

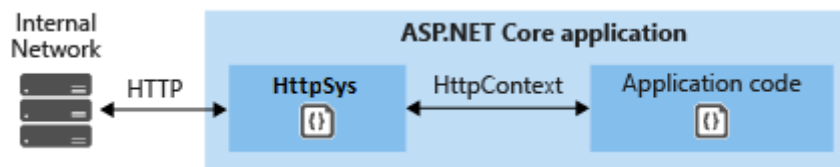
For information on how to use Apache on Linux as a reverse proxy server for Kestrel, see [Host ASP.NET Core on Linux with Apache](#).

HTTP.sys

If ASP.NET Core apps are run on Windows, HTTP.sys is an alternative to Kestrel. Kestrel is recommended over HTTP.sys unless the app requires features not available in Kestrel. For more information, see [HTTP.sys web server implementation in ASP.NET Core](#).



HTTP.sys can also be used for apps that are only exposed to an internal network.



For HTTP.sys configuration guidance, see [HTTP.sys web server implementation in ASP.NET Core](#).

ASP.NET Core server infrastructure

The [ApplicationBuilder](#) available in the `Startup.Configure` method exposes the [ServerFeatures](#) property of type [IFeatureCollection](#). Kestrel and HTTP.sys only expose a single feature each, [IServerAddressesFeature](#), but different server implementations may expose additional functionality.

`IServerAddressesFeature` can be used to find out which port the server implementation has bound at runtime.

Custom servers

If the built-in servers don't meet the app's requirements, a custom server implementation can be created. The [Open Web Interface for .NET \(OWIN\) guide](#) demonstrates how to write a [Nowin](#) -based [IServer](#) implementation. Only the feature interfaces that the app uses require implementation, though at a minimum [IHttpRequestFeature](#) and [IHttpResponseFeature](#) must be supported.

Server startup

The server is launched when the Integrated Development Environment (IDE) or editor starts the app:

- [Visual Studio](#) : Launch profiles can be used to start the app and server with either [IIS Express/ASP.NET Core Module](#) or the console.
- [Visual Studio Code](#) : The app and server are started by [Omnisharp](#) , which activates the CoreCLR debugger.
- [Visual Studio for Mac](#) : The app and server are started by the [Mono Soft-Mode Debugger](#) .

When launching the app from a command prompt in the project's folder, [dotnet run](#) launches the app and server (Kestrel and HTTP.sys only). The configuration is specified by the `-c|--configuration` option, which is set to either `Debug` (default) or `Release`.

A `launchSettings.json` file provides configuration when launching an app with `dotnet run` or with a debugger built into tooling, such as Visual Studio. If launch profiles are present in a `launchSettings.json` file, use the `--launch-profile {PROFILE NAME}` option with the `dotnet run` command or select the profile in Visual Studio. For more information, see [dotnet run](#) and [.NET Core distribution packaging](#).

HTTP/2 support

[HTTP/2](#) is supported with ASP.NET Core in the following deployment scenarios:

- [Kestrel](#)
 - Operating system
 - Windows Server 2016/Windows 10 or later†
 - Linux with OpenSSL 1.0.2 or later (for example, Ubuntu 16.04 or later)
 - HTTP/2 will be supported on macOS in a future release.
 - Target framework: .NET Core 2.2 or later
- [HTTP.sys](#)
 - Windows Server 2016/Windows 10 or later
 - Target framework: Not applicable to HTTP.sys deployments.
- [IIS \(in-process\)](#)
 - Windows Server 2016/Windows 10 or later; IIS 10 or later
 - Target framework: .NET Core 2.2 or later
- [IIS \(out-of-process\)](#)
 - Windows Server 2016/Windows 10 or later; IIS 10 or later
 - Public-facing edge server connections use HTTP/2, but the reverse proxy connection to Kestrel uses HTTP/1.1.
 - Target framework: Not applicable to IIS out-of-process deployments.

†Kestrel has limited support for HTTP/2 on Windows Server 2012 R2 and Windows 8.1. Support is limited because the list of supported TLS cipher suites available on these

operating systems is limited. A certificate generated using an Elliptic Curve Digital Signature Algorithm (ECDSA) may be required to secure TLS connections.

An HTTP/2 connection must use [Application-Layer Protocol Negotiation \(ALPN\)](#) and TLS 1.2 or later. For more information, see the topics that pertain to your server deployment scenarios.

Additional resources

- [Kestrel web server in ASP.NET Core](#)
- [ASP.NET Core Module \(ANCM\) for IIS](#)
- [Host ASP.NET Core on Windows with IIS](#)
- [Deploy ASP.NET Core apps to Azure App Service](#)
- [Host ASP.NET Core on Linux with Nginx](#)
- [Host ASP.NET Core on Linux with Apache](#)
- [HTTP.sys web server implementation in ASP.NET Core](#)