Kestrel web server implementation in ASP.NET Core

By Tom Dykstra, Chris Ross, and Stephen Halter

Kestrel is a cross-platform web server for ASP.NET Core. Kestrel is the web server that's included by default in ASP.NET Core project templates.

Kestrel supports the following features:

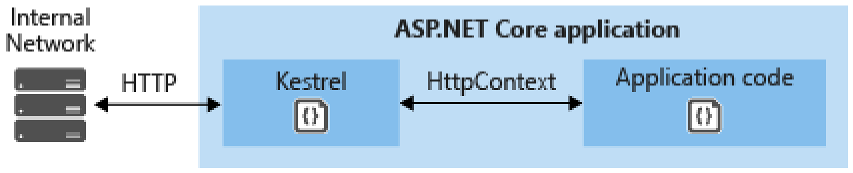
- HTTPS
- Opaque upgrade used to enable WebSockets
- Unix sockets for high performance behind Nginx

Kestrel is supported on all platforms and versions that .NET Core supports.

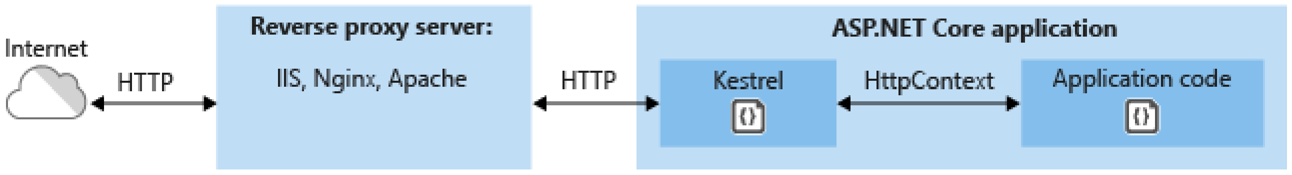
View or download sample code (how to download)

When to use Kestrel with a reverse proxy

If an app accepts requests only from an internal network, Kestrel can be used directly as the app's server.



If you expose your app to the Internet, use IIS, Nginx, or Apache as a *reverse proxy server*. A reverse proxy server receives HTTP requests from the Internet and forwards them to Kestrel after some preliminary handling.



A reverse proxy is required for public-facing edge server deployments (exposed to traffic from the Internet) for security reasons. The 1.x versions of Kestrel don't have a full complement of defenses against attacks, such as appropriate timeouts, size limits, and concurrent connection limits.

A reverse proxy scenario exists when there are multiple apps that share the same IP and port running on a single server. Kestrel doesn't support this scenario because Kestrel doesn't support sharing the same IP and port among multiple processes. When Kestrel is configured to listen on a port, Kestrel handles all of the traffic for that port regardless of requests' host header. A reverse proxy that can share ports has the ability to forward requests to Kestrel on a unique IP and port.

Even if a reverse proxy server isn't required, using a reverse proxy server might be a good choice:

- It can limit the exposed public surface area of the apps that it hosts.
- It provides an additional layer of configuration and defense.
- It might integrate better with existing infrastructure.
- It simplifies load balancing and SSL configuration. Only the reverse proxy server requires an SSL certificate, and that server can communicate with your app servers on the internal network using plain HTTP.

Warning

If not using a reverse proxy with host filtering enabled, host filtering must be enabled.

How to use Kestrel in ASP.NET Core apps

Install the Microsoft.AspNetCore.Server.Kestrel NuGet package.

Call the UseKestrel extension method on WebHostBuilder in the `Main` method, specifying any Kestrel options required, as shown in the next section.

C#
```
public static void Main(string[] args)
{
    Console.WriteLine("Running demo with Kestrel.");

    var config = new ConfigurationBuilder()
        .AddCommandLine(args)
        .Build();

    var builder = new WebHostBuilder()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .UseConfiguration(config)
        .UseStartup<Startup>()
        .UseKestrel(options =>
        {
            if (config["threadCount"] != null)
            {
                options.ThreadCount = int.Parse(config["threadCount"]);
            }
        })
        .UseUrls("http://localhost:5000");

    var host = builder.Build();
    host.Run();
}
```

Kestrel options

The maximum number of concurrent open TCP connections can be set for the entire app with the following code:

There's a separate limit for connections that have been upgraded from HTTP or HTTPS to another protocol (for example, on a WebSockets request). After a connection is upgraded, it isn't counted against the `MaxConcurrentConnections` limit.

Here's an example that shows how to configure the constraint for the app on every request:

For information about Kestrel options and limits, see:

- KestrelServerOptions class
- KestrelServerLimits

Endpoint configuration

Bind to a TCP socket

The Listen method binds to a TCP socket, and an options lambda permits SSL certificate configuration:

By default, ASP.NET Core binds to `http://localhost:5000`. Configure URL prefixes and ports for Kestrel using:

- UseUrls extension method
- `--urls` command-line argument
- `urls` host configuration key
- ASP.NET Core configuration system, including `ASPNETCORE_URLS` environment variable

For more information on these methods, see Hosting.

IIS endpoint configuration

When using IIS, the URL bindings for IIS override bindings set by `UseUrls`. For more information, see the ASP.NET Core Module topic.

URL prefixes

When using `UseUrls`, `--urls` command-line argument, `urls` host configuration key, or `ASPNETCORE_URLS` environment variable, the URL prefixes can be in any of the following formats.

- IPv4 address with port number

-
  ```
  http://65.55.39.10:80/
  https://65.55.39.10:443/
  ```

`0.0.0.0` is a special case that binds to all IPv4 addresses.

- IPv6 address with port number

-
  ```
  http://[0:0:0:0:0:ffff:4137:270a]:80/
  https://[0:0:0:0:0:ffff:4137:270a]:443/
  ```

`[::]` is the IPv6 equivalent of IPv4 `0.0.0.0`.

- Host name with port number

-
  ```
  http://contoso.com:80/
  http://*:80/
  https://contoso.com:443/
  https://*:443/
  ```

Host names, `*`, and `+` aren't special. Anything that isn't a recognized IP address or `localhost` binds to all IPv4 and IPv6 IPs. To bind different host names to different ASP.NET Core apps on the same port, use WebListener or a reverse proxy server, such as IIS, Nginx, or Apache.

- Host `localhost` name with port number or loopback IP with port number

-
  ```
  http://localhost:5000/
  http://127.0.0.1:5000/
  http://[::1]:5000/
  ```

When `localhost` is specified, Kestrel attempts to bind to both IPv4 and IPv6 loopback interfaces. If the requested port is in use by another service on either loopback interface, Kestrel fails to start. If either loopback interface is unavailable for any other reason (most commonly because IPv6 isn't supported), Kestrel logs a warning.

- Unix socket

  - `http://unix:/run/dan-live.sock`
  -

Port 0

When the port number is `0` is specified, Kestrel dynamically binds to an available port. Binding to port `0` is allowed for any host name or IP except for `localhost`.

When the app is run, the console window output indicates the dynamic port where the app can be reached:

console
```
Now listening on: http://127.0.0.1:48508
```

URL prefixes for SSL

If calling the `UseHttps` extension method, be sure to include URL prefixes with `https:`:

C#
```
var host = new WebHostBuilder()
    .UseKestrel(options =>
    {
        options.UseHttps("testCert.pfx", "testPassword");
    })
   .UseUrls("http://localhost:5000", "https://localhost:5001")
   .UseContentRoot(Directory.GetCurrentDirectory())
   .UseStartup<Startup>()
   .Build();
```

Note

HTTPS and HTTP can't be hosted on the same port.

On Windows, self-signed certificates can be created using the New-SelfSignedCertificate PowerShell cmdlet. For an unsupported example, see UpdateIISExpressSSLForChrome.ps1.

On macOS, Linux, and Windows, certificates can be created using OpenSSL.

Host filtering

While Kestrel supports configuration based on prefixes such as `http://example.com:5000`, Kestrel largely ignores the host name. Host `localhost` is a special case used for binding to loopback addresses. Any host other than an explicit IP address binds to all public IP addresses. None of this information is used to validate request `Host` headers.

As a workaround, host behind a reverse proxy with host header filtering. This is the only supported scenario for Kestrel in ASP.NET Core 1.x.

Additional resources

- Enforce HTTPS
- Kestrel source code

- RFC 7230: Message Syntax and Routing (Section 5.4: Host)
- Configure ASP.NET Core to work with proxy servers and load balancers