How-To: Learn Kestrel Webserver in 10 Minutes

What is Kestrel?

Kestrel is an open source, cross platform, light weight and a default webserver used for Asp.Net Core applications. Asp.Net Core applications run Kestrel webserver as in-process server to handle web request. Kestrel webserver is based on async I/O library called libuv primarily developed for Node.js.

By default, all Asp.Net core project templates include Kestrel webserver when we create new Asp.Net Core Projects. As said before, Kestrel is a very light weight webserver that does not have every advanced features of webservers like IIS, Nginx, Apache, etc has. Due to its lightweight nature, Kestrel provides better request processing performance to Asp.Net Core applications. Some features of Kestrel:

1. Kestrel does not support multiple applications sharing same port similar to IIS where applications are differentiated by host header value.
2. Kestrel is cross platform, runs in Windows, LINUX and Mac.
3. Kestrel webserver supports SSL.

Let's understand why there is a new webserver called Kestrel when we already have a feature-rich and most used webserver IIS.

Why Kestrel Webserver when we have IIS/Nginx/Apache?

The primary requirement of Asp.Net Core is to make Asp.Net Core applications to run across multiple platforms (or cross platform). Though, IIS is feature rich and highly used webserver it is a windows-only webserver. Without Kestrel, to run Asp.Net Core application on other cross platform webservers like Nginx, Apache, the Asp.Net Core application need to satisfy the Startup criteria of each of these webservers. In other words, each webserver has a different Startup configurations and this will make Asp.Net Core applications have different Startup mechanisms. This is why Asp.Net Core applications use Kestrel webserver as an in-process server where the application will have same and consistent Startup (Main() and Startup.ConfigireServices() & Startup.Configure()) process even when offering cross platform support.

How to use Kestrel?

Kestrel server is by default included in the Asp.Net Core project templates. It is included in the project as a Nuget package Microsoft.AspNetCore.Server.Kestrel in Project.json(in Visual Studio 2015) and as an implicitly included package when you using Visual Studio 2017 project template.

Note – All Asp.Net Core project should use Visual Studio 2017 as it is the latest and default project template for Asp.Net Core projects henceforth. Read here to know more.

Let's see how kestrel server is configured in an Asp.Net Core project. Below is the Program.Main() method in a Asp.Net Core application.

```
public class Program
{
public static void Main(string[] args)
{
    var host = new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .UseIISIntegration()
        .UseStartup<Startup>()
        .UseApplicationInsights()
        .Build();

    host.Run();
}
}
```

In the above Main() method, calling UseKestrel() on WebHostBuilder object will configure the Kestrel webserver for the application.

Note – Every Asp.Net Core application requires a Host process to start Kestrel webserver and initialize application for request processing. Calling UseIISIntegration() integrates Kestrel with IIS webserver, let's see this in detail in next section.

The UseKestrel() also takes KestrelServerOptions class as argument to customize Kestrel server.

Asp.Net Core Deployment Options

We generally deploy Asp.Net Core (also Asp.Net applications) for 2 scenarios listed below.

1. Internal Hosted applications
2. Externally Hosted applications

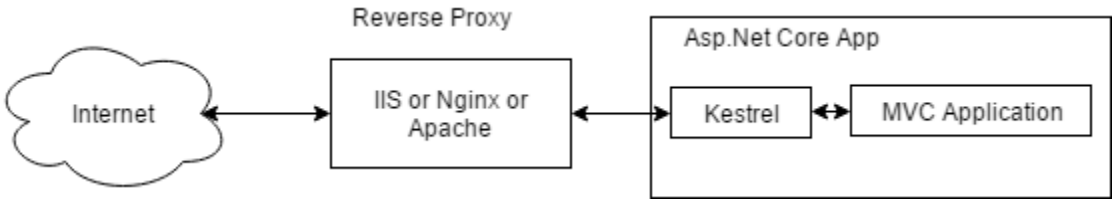
Let's see how Kestrel webserver can be used in both of these scenarios.

1. Internally Hosted applications

For internal applications, the applications can just be hosted with Kestrel webserver alone. The features provided by Kestrel and with some additional middlewares (for response compressions, authentication) is sufficient for serving internally hosted applications.

2. Externally Hosted applications

As it is said before, Kestrel is a very light webserver and it does not support all features of a full-fledged webserver like IIS provide. So, if your application is external facing public website then the application should be deployed behind a full-fledged webserver like IIS or Nginx or Apache for increased security and to get feature rich support. These webservers act as proxy (commonly called reverse proxy in this scenario) and forward the request to Kestrel for request processing. This is why the Program.Main() code calls UseIISIntegration() to integrate IIS as reverse proxy to forward request to Kestrel. This is the default mode the project template code uses. The request flow goes like below:



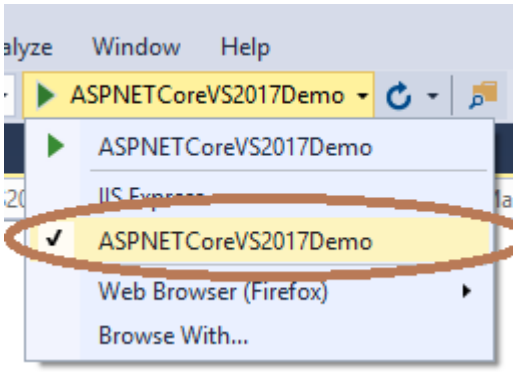
Note – This requires IIS integration module commonly called Asp.Net Core module that comes with Nuget package [Microsoft.AspNetCore.Server.IISIntegration]

Executing the Asp.Net Core Application with Kestrel

We can start the Asp.Net application that uses Kestrel webserver in two ways. Assuming, you have created a default Asp.Net Core project in Visual Studio 2017.

1. Visual Studio

By default, press F5 to start the application with IISExpress as reverse proxy. To start the application from Kestrel webserver click project name menu item in run dropdown like below. ASPNETCoreVS2017Demo is my project name.



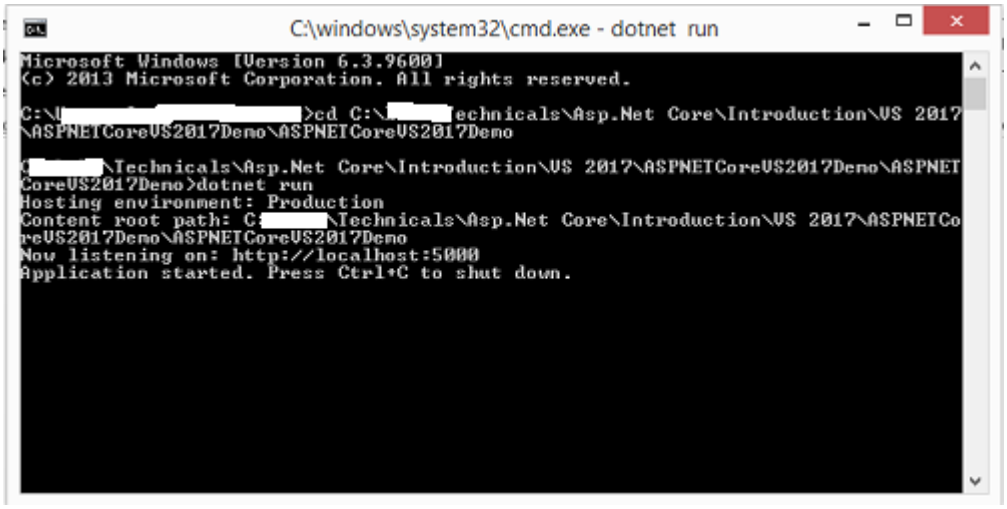
This will start the app directly form Kestrel webserver.

2. Dotnet CLI

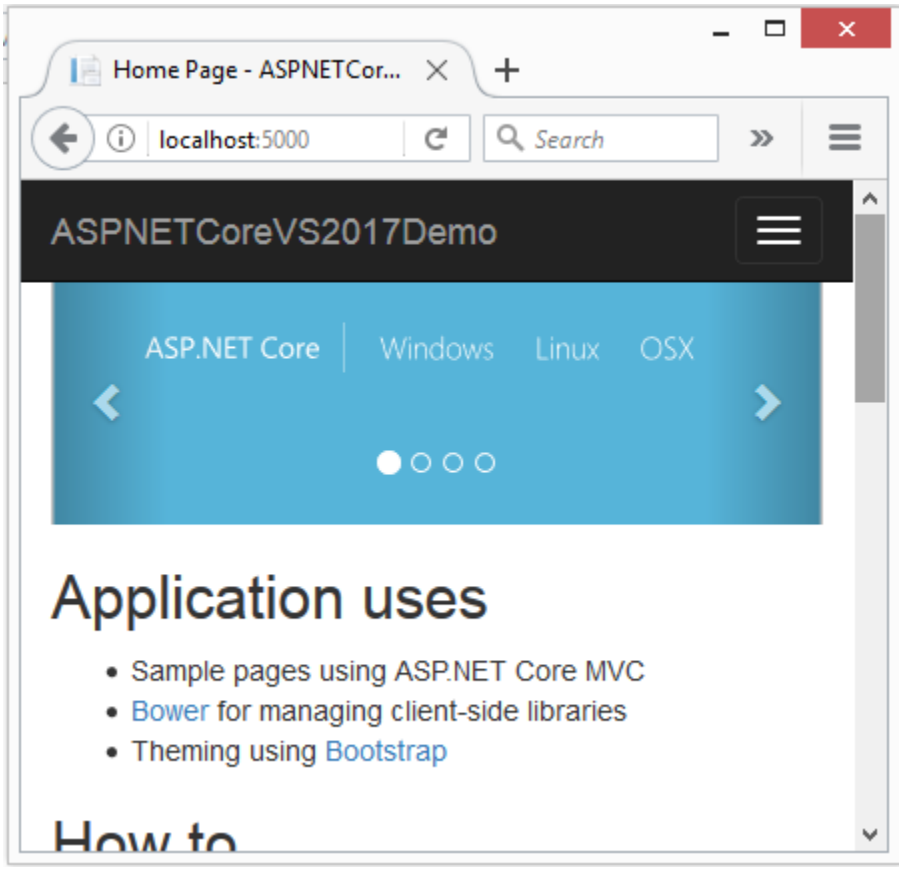
You can also start the app from command prompt using dotnet CLI(Command Line Interface) provide by Asp.Net Core. To do this, open Command prompt and change directory to project root. Type the below command

```
dotnet run
```

This will start the app with Kestrel webserver and server will start listening to port 5000 by default. Refer the below image.



Visiting http://localhost:5000 will bring the application like below,



Courtesy: http://www.codedigest.com/quick-start/5/learn-kestrel-webserver-in-10-minutes

Modified: 2021.10.04.7.21.AM
Dököll Solutions, Inc.