ASP.Net Core

ASP.Net Core is a cross-platform, high-performance, open-source framework for building modern, cloud-enabled web applications and services.

Features

- Cross-platform
  Asp.Net core apps can be hosted on Windows, LINUX and Mac.

- Can be hosted on different serves
  Supports Kestrel, IIS, Nginx, Docker, Apache

- Open-source

- Cloud-enabled
  Out-of-box support for Microsoft Azure

Modules

- ASP.Net core MVC
  For creating medium to complex web applications

- ASP.Net core WebAPI
  For creating RESTful services for all types of client applications

- ASP.Net core Razor Pages
  For creating simple & page-focused web applications

- ASP.Net core Blazor
  For creating web applications with C# code both on client-side and server-side

ASP.Net Web Forms vs ASP.Net MVC vs ASP.Net Core

ASP.Net Web Forms:

- 2002
- Performance issues due to server events and view-state.
- Windows-only
- Not cloud-friendly
- Not open-source
- Event-driven development model

ASP.Net MVC:

- 2009
- Performance issues due to dependencies with ASP.Net (.net framework)
- Windows-only
- Slightly cloud-friendly
- Open source
- Model-View-Controller (MVC) pattern

ASP.Net Core:

- 2016
- Faster performance

- Cross-platform
- Open source
- Cloud-friendly
- Model-View-Controller (MVC) pattern

What is ASP.Net Core?

ASP.Net Core is a free, open-source, and cross-platform framework for building cloud based applications, such as web apps, IoT apps, and mobile backends. It is designed to run on the cloud as well as on-premises.

ASP.Net Core is not an upgraded version of ASP.Net. ASP.Net Core is completely rewriting that work with the .Net Core framework. It is much faster, configurable, modular, scalable, extensible, and has cross-platform support. It is best suitable for developing cloud-based such as web applications, mobile applications and IoT applications.

What are the features of ASP.Net Core?

Following are the core features that are provided by the ASP.Net Core

- Built-in support for Dependency Injection.
- Built-in support for logging framework and it can be extensible.
- Introduced a new, fast and cross-platform web server – Kestrel. So, a web application can run without IIS, Apache, and Nginx.
- Multiple hosting platforms are supported.
- It supports modularity, so the developer need to include the module required by the application.
- Command-line support to creating, building and running of the application.
- There is no web.config file. We can store the custom configuration into an appsettings.json file.
- It has good support for asynchronous programming

What are the advantages of ASP.Net Core over ASP.Net (.Net Framework)?

These are the following advantages of ASP.Net Core over ASP.Net:

- It offers faster performance due to its minimalistic design.
- It is cross-platform, so it can be run on Windows, Linux and Mac.
- It is open-source.
- There is no dependency on framework installation because all the required dependencies are shipped with our application to the production server.
- Multiple deployment platform available with ASP.Net Core.

What is ASP.Net Core meta package?

The ASP.Net Core shared framework (Microsoft.AspNetCore.App) contains assemblies that are developed and supported by Microsoft.

Microsoft.AspNetCore.App is installed when the .Net Core 3.0 or later SDK is installed. The shared framework is set of assemblies (.dll files) that are installed on the machine and includes a runtime component and a targeting pack.

When do you choose classic ASP.Net MVC over ASP.Net Core?

Though ASP.Net Core is a better choice in almost all the aspects, you don't have to switch to ASP.Net Core if you are maintaining a legacy ASP.Net application that you are happy with and that is no longer actively developed.

ASP.NET MVC is a better choice if you:

- Don’t need cross-platform support for your Web app.
- The existing team is already working on an existing app and extending its functionality.
- The existing developers need a learning curve to upgrade themselves to ASP.Net Core.

What is a web application framework, and what are its benefits?

Learning to build modern web application can be daunting. Most of the applications have a standard set of functionalities such as:

- Build a dynamic response that corresponds to an HTTP request.
- Allow users to login to the application and manage their data.
- Store the data in the database.
- Handle database connections and transactions.
- Route URLs to appropriate methods.
- Supporting sessions, cookies, and user authorization.
- Format output (e.g: HTML, JSON, XML)
- Improve security.

Frameworks help developers to write, maintain, and scale applications. They provide tools and libraries that simplify the above recurring tasks, eliminating a lot of unnecessary complexity.

Kestrel Server and Reverse Proxy Servers

Kestrel Server

Overview:

- Kestrel is the cross-platform web server for ASP.NET Core.
- It is lightweight and suitable for serving dynamic content.

Responsibilities:

- **HTTP Requests Handling:** Handles incoming HTTP requests and responses.
- **Hosting:** Hosts the ASP.NET Core application.
- **Configuration:** Supports various configurations such as HTTP/2, HTTPS, etc.,

Use Case:

- Ideal for development and internal networks.
- Typically used in conjunction with a reverse proxy for production environment.

Reverse Proxy Servers

Overview:

- A reverse proxy server forwards client requests to backend servers and returns the responses to the clients.
- Common reverse proxy servers include Nginx, Apache, and IIS.

Responsibilities:

- **Load Balancing:** Distributes incoming requests across multiple servers.
- **SSL Termination:** Handles SSL/TLS encryption and decryption.

- **Caching:** Catches responses to improve performance.
- **Security:** Provides additional security features like request filtering, IP whitelisting, and rate limiting.

Use Case:

- Used in front of Kestrel to enhance security, load balancing, and other enterprise-level requirements.

Responsibilities of Kestrel and Reverse Proxy Servers

Kestrel:

- Serves HTTP requests directly.
- Provides efficient request processing.
- Should be used behind a reverse proxy for additional security and stability.

Reverse proxy:

- Acts as an intermediary between clients and Kestrel.
- Provides SSL termination, load balancing and security features.
- Enhances the overall performance and security of the application.

Summary of launchsettings.json:

- launchsettings.json configures how an ASP.NET Core application is launched during development.
- It can define multiple profiles, each with its own settings for URLs, environment variables, and launch options.
- The iisSettings section configures IIS Express settings, while the profiles section defines different launch profiles for the application.

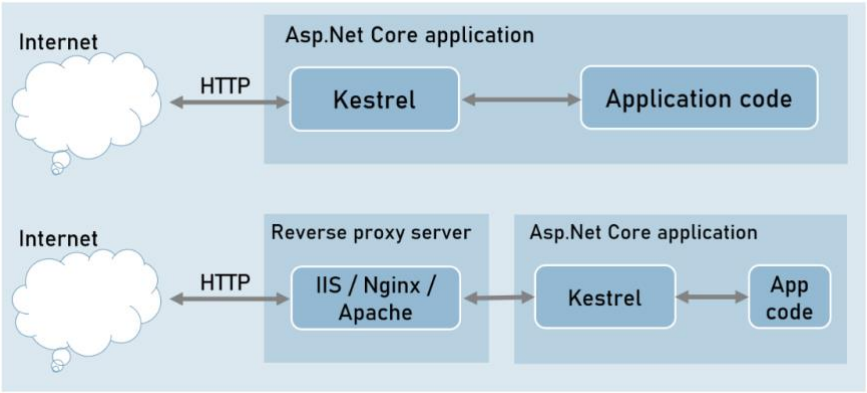
Kestrel and Other Servers

Application Servers

- Kestrel

Reverse Proxy Servers

- IIS
- Nginx
- Apache



Benefits of Reverse proxy Servers

- Load Balancing
- Caching
- URL Rewriting
- Decompressing the requests
- Authentication
- Decryption of SSL certificates

IIS Express

- HTTP access logs
- Post sharing
- Windows authentication
- Management console
- Process activation
- Configuration API
- Request filters
- HTTP redirect rules

Code

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```

- The code creates and configures a minimal ASP.NET Core web application.
- WebApplication.CreateBuilder(args) sets up the application with default settings.
- builder.Build() finalizes the configuration and prepares the application.
- App.MapGet("/", () => "Hello World!"); maps a get request to the root URL and returns "Hello World!" as a response.
- App.Run() starts the web server and runs the application, ready to handle incoming requests.

What is Kestrel and what are advantages of Kestrel in ASP.NET Core?

ASP.Net Core application uses Kestrel by default.

Kestrel is an event-driven, I/O-based, open-source, cross-platform and asynchronous server which hosts ASP.Net applications. It is provided as a default server for .NET Core; therefore, it is compatible with all the platforms and their versions which .NET Core supports.

It is a listening server with a command-line interface.

It can be used to reverse proxy servers such as IIS, Nginx etc.,

Features of Kestrel are:

- Lightweight and fast.
- Cross-platform and supports all versions of .NET Core.
- Supports HTTP and HTTPS.
- Easy configuration.
- Multiple apps on same port is not supported.

- Windows authentication is not supported.

What is the difference between IIS and Kestrel? Why do we need two web servers?

The main difference between IIS and Kestrel is that Kestrel is a cross-platform server. It runs on Windows, Linux and Mac, whereas IIS runs only on Windows.

Another essential difference between the two is that Kestrel is fully open-source, whereas IIS is closed-source and deployed and maintained only be Microsoft.

IIS is very old software and comes with a considerable legacy and bloat. With Kestrel, Microsoft started with high-performance in mind. They developed it from scratch, which allowed them to ignore the legacy/compatibility issues and focus on speed and efficiency.

However, Kestrel doesn't provide all the rich functionality of a full-fledged web server such as IIS, Nginx, or Apache. Hence, we typically use it as an application server, with one of the above servers acting as a reverse proxy.

What is the purpose of launchSettings.json in asp.net core?

The launchSettings.json file is used to store the configuration information, which describes how to the ASP.NET Core application, in Visual Studio.

It mainly contains the run time profiles to configure application urls and environment.

The file is used only during the development of the application using Visual Studio. It contains only these settings that are required to run the application.

launchSettings.json is only used by Visual Studio.

You don't need launchSettings.json for publishing an app (on production server).

What is generic host or host builder in .NET Core?

.NET generic host called 'HostBuilder' helps us to manage all the below tasks:

- Dependency Injection
- Service lifetime management
- Configuration
- Logging

The generic host was previously present as 'Web Host', in .NET Core for web applications. Later, the 'Web Host' was deprecated and a generic host was introduced to cater to the web, Windows, Linux, and console applications.

What is the purpose of the .csproj file?

The project file is one of the most important files in our application. It tells .NET how to build (compile) the project.

The csproj file stores list of package dependencies of the current project, target .net version and other compilation settings.

The .csproj file also contains all the information that .NET tooling needs to build the project. It includes the type of project you are building (console, web, desktop etc.), the platform this project targets, and any dependencies on other projects or 3rd party libraries.

Here is an example of a .csproj file that lists the NuGet packages and their specific versions.

```
<Project Sdk=”Microsoft.NET.Sdk.Web”>
<PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
</PropertyGroup>

<ItemGroup>
<PackageReference Include=”PackageName” Version=”1.0.0.0”>
</ItemGroup>
</Project>
```

What is IIS?

IIS stans for Internet Information Services. It is a powerful web server developed by Microsoft. IIS can also act as a load balancer to distribute incoming HTTP requests to different application servers to allow high reliability and scalability.

It can also act as a reverse proxy, i.e. accept a client’s request, forward it to an application server, and return the client’s response. A reverse proxy improves the security, reliability, and performance of your application.

A limitation of IIS is that it only runs on Windows. However, it is very configurable. You can configure it to suit your application’s specific needs.

What is the “Startup” class in ASP.NET core prior to to ASP.NET Core 6?

The startup class is the entry point of the ASP.NET Core application. Every ASP.Net Core (ASP.NET Core 5 and earlier) application must have this class. It contains the necessary code to bootstrap the application. This class contains two methods Configure() and ConfigureServices().

- Configure(): The Configure() method is used to essential middleware(s) to the application request pipeline.
- ConfigurationServices(): The ConfigurationServices() method is used to add services to the IoC container.

In ASP.NET Core 6 (.NET 6), Microsoft unifies startup.cs and Program.cs into a single Program.cs.

In asp.net core 6 (and up), we need to add all such as registering middleware to the application pipeline, adding services to the IoC container, configuring the ‘application configuration’, configuring the logger, authentication and adding DbContext in Program.cs file.

What does WebApplication.CreateBuilder() do?

This method does the following things:

- Configure the app to use Kestrel as web server.
- Specify to use the current project directory as root directory of the application.
- Setup the configuration sub-system to read settings from appsettings.json and appsetting.{env}.json to environment specific configuration.
- Set Local user secrets storage only for the development environment.
- Configure environment variables to allow for server-specific settings.
- Configure command line arguments (if any).
- Configure logging to read from the logging section of the appsettings.json file and log to the Console and Debug window.
- Configure integration with IIS.

- Configure the default service provider.

HTTP Protocol

Overview:

- HTTP (Hypertext Transfer Protocol) is a protocol used for transmitting hypertext (e.g., HTML) over the internet.
- It operates on a client-server model, where the client (usually a web browser) makes requests to a server, which then responds with the requested resources or error messages.
- **Stateless Protocol:** Each HTTP request is independent of other; the server does not retain information from previous requests.

Request/Response Model:

- **Client Request:** The client sends an HTTP request to the server.
- **Server Response:** The server processes the request and sends back an HTTP response.

HTTP Server

Definition:

- An HTTP server is software that handles HTTP requests from clients and serves back responses. It processes incoming requests, executes the necessary logic (e.g., accessing a database, generating HTML), and returns the appropriate response.

Examples:

- Apache HTTP Server, Nginx, Microsoft IIS, Kestrel (used with ASP.NET Core)

Kestrel:

- Kestrel is a cross-platform web server included with ASP.NET Core.
- It is lightweight, high-performance, and suitable for running both internal and public-facing web applications.

Request and Response Flow with Kestrel

1. **Client Sends Request:**
    1. The client (e.g., web browser) sends an HTTP request to the server.
2. **Kestrel Receives Request:**
    2. Kestrel receives the request and passes it through the ASP.NET Core middleware pipeline.
3. **Request Processing:**
    3. Middleware components process the request and eventually pass it to the application's request handling logic.
4. **Generate Response:**
    4. The application generates an HTTP response and sends it back through the middleware pipeline.

5. **Kestrel Sends Response:**
    5. Kestrel sends the HTTP response back to the client.

How Browsers Use HTTP

- Browsers use HTTP to request resources such as HTML documents, images, CSS files, and JavaScript files from servers.
- When a user enters a URL or clicks a link, the browser sends an HTTP request to the server, which then responds with the requested resource.

Observing HTTP Requests and Responses in Chrome Dev Tools

- **Open Chrome Dev Tools:**
    - Press F12 or Ctrl + Shift + I (or Cmd+Option+I on Mac) to open Chrome Dev Tools.
- **Navigate to the Network Tab:**
    - Click on the Network tab to view HTTP requests and responses.
- **Inspect a Request:**
    - Click on any request in the list to see detailed information:
        - **Headers:** View request and response headers.
        - **Preview/Response:** View the response body.
        - **Timing:** See the timing details of the request.

HTTP Response Message Format

Response Message Format:

- **Start Line:** Contains the HTTP version, status code, and status message.
- **Headers:** Key-value pairs providing information about the response.
- **Body:** Optional, contains the actual data (e.g., HTML, JSON).

Example:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 137

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

Commonly Used Response Headers:

- **Content-Type:** Specifies the media type of the resource.
- **Content-Length:** The size of the response body in bytes.
- **Server:** Provides information about the server handling the request.
- **Set-Cookie:** Sets cookie to be stored by the client.
- **Cache-Control:** Directives for caching mechanisms in both requests and responses.

Default Response Headers in Kestrel

- **Content-Type:** Typically defaults to text/html or application/json depending on the content being served.
- **Server:** Indicates the server software (e.g., Kestrel)
- **Date:** The date and time when the response was generated.

HTTP Status Codes

Overview:

- Status codes are issued by the server in response to the client's request to indicate the result of the request.
- Categories include:
    - **1xx Informational:** Request received, continuing process.
    - **2xx Success:** The request was successfully received, understood, and accepted.
    - **3xx Redirection:** Further action needs to be taken in order to complete the request.
    - **4xx Client Error:** The request contains bad syntax or cannot be fulfilled.
    - **5xx Server Error:** The server failed to fulfil an apparently valid request.

Common Status Codes:

- **200 OK:** The request succeeded.
- **201 Created:** The request succeeded and a new resource was created.
- **204 No Content:** The server successfully processed the request, but is not returning any content.
- **400 Bad Request:** The server could not understand the request due to invalid syntax.
- **401 Unauthorized:** Authentication is required.
- **403 Forbidden:** The client does not have access rights to the content.
- **404 Not Found:** The server cannot find the request resource.
- **500 Internal Server Error:** Ther server encountered an unexpected condition.
- **502 Bad Gateway:** The server was acting as a gateway or proxy and received an invalid response from the upstream server.
- **503 Service Unavailable:** The server is not ready to handle the request.