# Configuration System:

ASP.NET Core uses a new and flexible configuration system based on the .NET Configuration API that allows configuration to be read from various sources such as appsettings.json, environment variables, and command-line arguments. On the other hand, .NET Framework Web API uses the older configuration system based on XML configuration files.

# Hosting Models:

ASP.NET Core - Kestrel, which is cross-platform and lightweight integrated web server. It can also be hosted in other web servers such as IIS or Apache using a reverse proxy.

.NET Framework Web API - only in IIS.

# DI:

.NET Framework - System.Web.Mvc framework for dependency injection, which can be more complex and difficult to manage.

ASP.NET Core uses Microsoft.Extensions.DependencyInjection package, a lightweight and modular approach. It allows to easily swap out implementations or configure different services for different environments.

In .NET Framework, DI you need to:

1. first register services types in DI container and
2. then specify dependency resolver for these services.

This is typically done in the Global.asax file, which is executed when the application starts up. You can register your dependencies using a variety of techniques, such as registering individual services one by one or using a container to automatically resolve dependencies.

Here is an example of how to register a dependency using the DependencyResolver and Autofac:

var builder = new ContainerBuilder();

builder.RegisterType<MyService>().As<IMyService>();

var container = builder.Build();

DependencyResolver.SetResolver(new AutofacDependencyResolver(container));

Once you have registered your dependencies, you can use constructor injection to inject them.

Here is DI with native .NET Framework dependency resolver Unitiy.

var container = new UnityContainer();

container.RegisterType<IMyService, MyService>();

GlobalConfiguration.Configuration.DependencyResolver = new UnityDependencyResolver(container);

Unity container is not as flexible or powerful as some other popular DI containers like Autofac, Ninject, or Simple Injector. It also lacks some of the advanced features that are available in the native DI container in ASP.NET Core.

Testability:

ASP.NET Core's dependency injection framework is designed to make it easy to write unit tests for your application. By injecting dependencies into your classes, you can easily replace implementations with mock objects for testing purposes. This is much harder to do in .NET Framework Web API, which does not have built-in support for dependency injection and requires third-party libraries to implement.

# Middleware

In ASP .NET Framework, middleware is implemented using HttpModules and HttpHandlers, whereas in .NET Core, middleware is implemented using middleware components.

# The Pipeline Process Model

The ASP.NET HTTP pipeline relies on Internet Information Services (IIS) to receive the requests it is going to process (it can also be integrated with other Web servers). When IIS receives an HTTP request, it examines the extension of the file identified by the target URL. If the file extension is associated with executable code, IIS invokes that code in order to process the request. Mappings from file extensions to pieces of executable code are recorded in the IIS metabase. When ASP.NET is installed, it adds entries to the metabase associating various standard file extensions, including .aspx and .asmx, with a library called aspnet\_isapi.dll.