**Generic Host Builder in ASP .NET Core**

By Shahed C on February 18, 2019

[5 Replies](https://wakeupandcode.com/generic-host-builder-in-asp-net-core/#comments)

This is the **seventh**of a [series of posts](https://wakeupandcode.com/aspnetcore/#aspnetcore2019) on ASP .NET Core in 2019. In this series, we’ll cover 26 topics over a span of 26 weeks from January through June 2019, titled **A-Z of ASP .NET Core!**

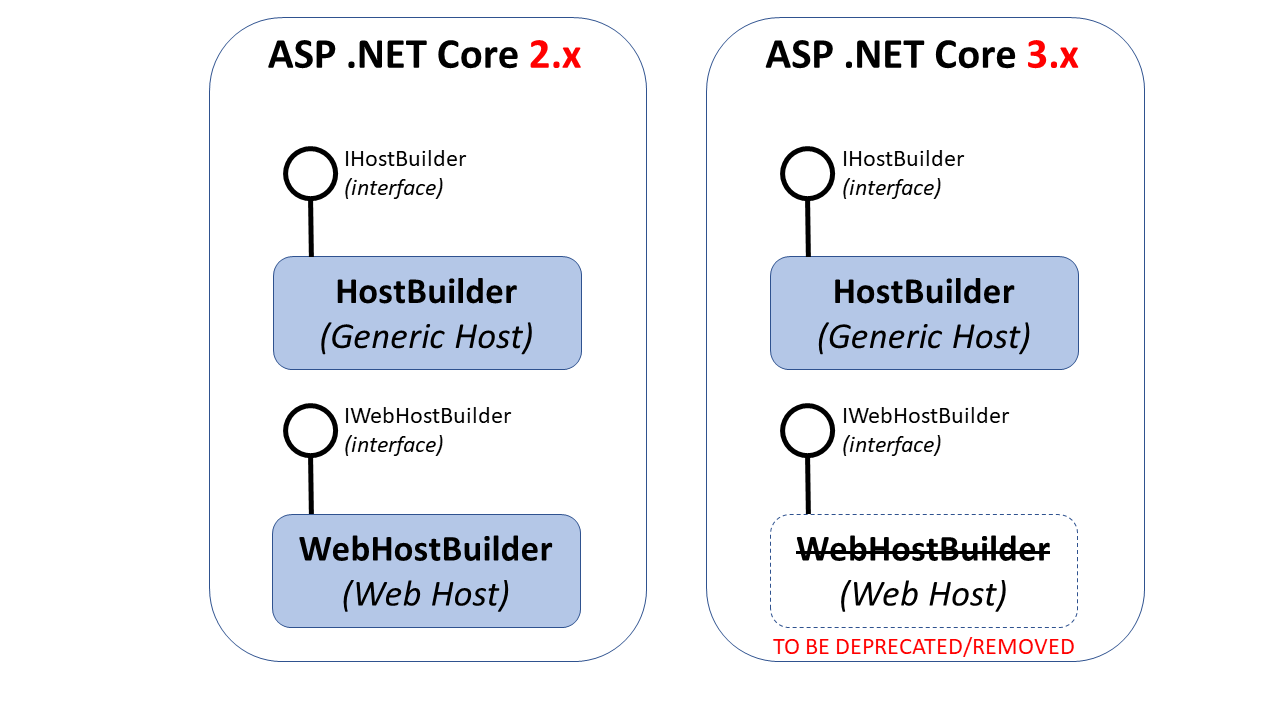
** A – Z of ASP .NET Core!**

**In this Article:**

* G is for Generic Host Builder
* Generic Host Builder in 2.x
* Web Host Builder in 2.x
* Generic Host Builder for Web Apps in 3.x
* References

**G is for Generic Host Builder**

The Generic Host Builder in ASP .NET Core was introduced in v2.1, but only meant for non-HTTP workloads. However, it is intended to replace the Web Host Builder when v3.0 is released in 2019.

[](https://wakeupandcode.com/wp-content/uploads/2019/02/Blog-Diagram-GenericHost.png)

Generic Host Builder in ASP .NET Core 3.0

**Generic Host Builder in 2.x**

So, if the Generic Host Builder isn’t currently used for web hosting in v2.x, what can it be used for? The aforementioned non-HTTP workloads include a number of capabilities according to the [documentation](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/generic-host?view=aspnetcore-2.2), including:

* app config, e.g. set base path, add hostsettings.json, env variables, etc
* dependency injection, e.g. various hosted services
* logging capabilities, e.g. console logging

The **HostBuilder** class is available from the following namespace, implementing the **IHostBuilder** interface:

using Microsoft.Extensions.Hosting;

At a minimum, the **Main**() method of your .NET Core app would look like the following:

public static async Task Main(string[] args)  
{  
 var host = new HostBuilder()  
 .Build();   
  
 await host.RunAsync();  
}

Here, the [Build() method](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.hosting.hostbuilder.build?view=aspnetcore-2.2) initializes the host, so (as you may expect) it can only be called once for initialization. Additional options can be configured by calling the **ConfigureServices**() method before initializing the host with **Build**().

var host = new HostBuilder()  
 .ConfigureServices((hostContext, services) =>  
 {  
 services.Configure<HostOptions>(option =>  
 {  
 // option.SomeProperty = ...  
 });  
 })  
 .Build();

Here, the [ConfigureServices() method](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.hosting.hostbuilder.configureservices?view=aspnetcore-2.2#Microsoft_Extensions_Hosting_HostBuilder_ConfigureServices_System_Action_Microsoft_Extensions_Hosting_HostBuilderContext_Microsoft_Extensions_DependencyInjection_IServiceCollection__) takes in a **HostBuilderContext** and an injected collection of **IServiceCollection** services. The options set in the Configure() can be used to set additional HostOptions. Currently, [HostOptions](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.hosting.hostoptions?view=aspnetcore-2.2) just has one property, i.e. ShutdownTimeout.

You can see more configuration capabilities in the [official sample](https://github.com/aspnet/Docs/blob/master/aspnetcore/fundamentals/host/generic-host/samples/2.x/GenericHostSample/Program.cs), broken down into the snippets below:

***Host Config Snippet:***

.ConfigureHostConfiguration(configHost =>  
{  
 configHost.SetBasePath(Directory.GetCurrentDirectory());  
 configHost.AddJsonFile("hostsettings.json", optional: true);  
 configHost.AddEnvironmentVariables(prefix: "PREFIX\_");  
 configHost.AddCommandLine(args);  
})

***App Config Snippet:***

.ConfigureAppConfiguration((hostContext, configApp) =>  
{  
 configApp.AddJsonFile("appsettings.json", optional: true);  
 configApp.AddJsonFile(  
 $"appsettings.{hostContext.HostingEnvironment.EnvironmentName}.json",   
 optional: true);  
 configApp.AddEnvironmentVariables(prefix: "PREFIX\_");  
 configApp.AddCommandLine(args);  
})

***Dependency Injection Snippet:***

.ConfigureServices((hostContext, services) =>  
{  
 services.AddHostedService<LifetimeEventsHostedService>();  
 services.AddHostedService<TimedHostedService>();  
})

***Logging Snippet:***

.ConfigureLogging((hostContext, configLogging) =>  
{  
 configLogging.AddConsole();  
 configLogging.AddDebug();  
})

**Web Host Builder in 2.x**

The **WebHostBuilder**class is available from the following namespace (specific to ASP .NET Core), implementing the **IWebHostBuilder** interface:

using Microsoft.AspNetCore.Hosting;

The [Web Host Builder in ASP .NET Core](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/web-host?view=aspnetcore-2.2) is currently used for hosting web apps as of v2.x. As mentioned in the previous section, it will be replaced by the Generic Host Builder in v3.0. At a minimum, the **Main**() method of your ASP .NET Core 2.x web app would look like the following:

public class Program  
{  
 public static void Main(string[] args)  
 {  
 CreateWebHostBuilder(args).Build().Run();  
 }   
  
 public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>  
 WebHost.CreateDefaultBuilder(args)  
 .UseStartup<Startup>();  
}

If you’re not familiar with the shorthand syntax of the helper method **CreateWebHostBuilder**() shown above, here’s what it would normally look like, expanded:

public static IWebHostBuilder CreateWebHostBuilder(string[] args)  
{  
 return WebHost.CreateDefaultBuilder(args).UseStartup<Startup>();  
}

**NOTE:** This type of C# syntax is known as an [Expression Body Definition](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/lambda-operator#expression-body-definition), introduced for methods in C# 6.0, and additional features in C# 7.0.

The **CreateDefaultBuilder**() method performs a lot of “magic” behind the scenes, by making use of pre-configured defaults. From the [official documentation](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.webhost.createdefaultbuilder?view=aspnetcore-2.2), here is a summary of the default configuration from the Default Builder:

* use Kestrel as the web server
* configure it using the application’s configuration providers,
* set the [ContentRootPath](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.hosting.ihostingenvironment.contentrootpath?view=aspnetcore-2.2#Microsoft_AspNetCore_Hosting_IHostingEnvironment_ContentRootPath) to the result of [GetCurrentDirectory()](https://docs.microsoft.com/en-us/dotnet/api/system.io.directory.getcurrentdirectory#System_IO_Directory_GetCurrentDirectory),
* load [IConfiguration](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.configuration.iconfiguration?view=aspnetcore-2.2) from ‘appsettings.json’ and ‘appsettings.[[EnvironmentName](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.hosting.ihostingenvironment.environmentname?view=aspnetcore-2.2#Microsoft_AspNetCore_Hosting_IHostingEnvironment_EnvironmentName)].json’,
* load [IConfiguration](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.configuration.iconfiguration?view=aspnetcore-2.2) from User Secrets when [EnvironmentName](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.hosting.ihostingenvironment.environmentname?view=aspnetcore-2.2#Microsoft_AspNetCore_Hosting_IHostingEnvironment_EnvironmentName) is ‘Development’ using the entry assembly,
* load [IConfiguration](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.configuration.iconfiguration?view=aspnetcore-2.2) from environment variables,
* load [IConfiguration](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.configuration.iconfiguration?view=aspnetcore-2.2) from supplied command line args,
* configure the [ILoggerFactory](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.iloggerfactory?view=aspnetcore-2.2) to log to the console and debug output,
* enable IIS integration (if running behind IIS with the ASP .NET Core Module)

For more information on some of the above, here are some other blog posts in this series, you may find useful:

* [Your Web App Secrets in ASP .NET Core](https://wakeupandcode.com/your-web-app-secrets-in-asp-net-core/)
* (Coming soon) IIS In-Process Modules in ASP .NET Core

**Generic Host Builder for Web Apps in 3.x**

Going forward, ASP .NET Core 3.0 will allow you to use the updated Generic Host Builder instead of the Web Host Builder in your web apps. As of [Preview 2](https://blogs.msdn.microsoft.com/webdev/2019/01/29/aspnet-core-3-preview-2/), the templates available in ASP .NET Core 3.0 have already been updated to include the Generic Host Builder.

At a minimum, the **Main**() method of your .NET Core 3.0 web app would now look like the following:

public static void Main(string[] args)  
{  
 CreateHostBuilder(args)  
 .Build()  
 .Run();  
}  
  
public static IHostBuilder CreateHostBuilder(string[] args) =>  
 Host.CreateDefaultBuilder(args)  
 ConfigureWebHostDefaults(webBuilder =>  
 {  
 webBuilder.UseStartup<Startup>();  
 });

Here’s an expanded representation of the **CreateHostBuilder**() method:

public static IHostBuilder CreateHostBuilder(string[] args)  
{  
 return Host.CreateDefaultBuilder(args)  
 ConfigureWebHostDefaults(webBuilder =>  
 {  
 webBuilder.UseStartup<Startup>();  
 });  
}

This **CreateHostBuilder**() method in the 3.0 template looks very similar to the 2.x call to **CreateWebHostBuilder**() mentioned in the previous section. In fact, the main difference is that the call to ***WebHost***.**CreateDefaultBuilder**() is replaced by ***Host***.**CreateDefaultBuilder**(). Using the **CreateDefaultBuilder**() helper method makes it very easy to switch from v2.x to v3.0.

Another difference is the call to **ConfigureWebHostDefaults**(). Since the new host builder is a ***Generic*** Host Builder, it makes sense that we have to let it know that we intend to configure the default settings for a Web Host. The **ConfigureWebHostDefaults**() method does just that.

Going forward, it’s important to know the following:

* **WebHostBuilder** will be *deprecated* and then *removed*in the near future
* However, the **IWebHostBuilder** interface will remain
* You won’t be able to inject just any service into the Startup class…
* … instead, you have IHostingEnvironment and IConfiguration

If you’re wondering about the reason for the limitation for injecting services, this change prevents you from injecting services into the Startup class  *before* **ConfigureServices**() gets called.

**References**

* ASP.NET Core updates in .NET Core 3.0 Preview 2: <https://blogs.msdn.microsoft.com/webdev/2019/01/29/aspnet-core-3-preview-2/>
* .NET Generic Host: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/generic-host?view=aspnetcore-2.2>
* ASP.NET Core Web Host: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/web-host?view=aspnetcore-2.2>
* Using HostBuilder and the Generic Host in .NET Core Microservices: <https://www.stevejgordon.co.uk/using-generic-host-in-dotnet-core-console-based-microservices>
* The ASP.NET Core Generic Host: namespace clashes and extension methods: <https://andrewlock.net/the-asp-net-core-generic-host-namespace-clashes-and-extension-methods/>
* Samples on GitHub: <https://github.com/aspnet/Docs/tree/master/aspnetcore/fundamentals/host/generic-host/samples/>