.NET Generic Host in ASP.NET Core

04/17/2020 • 32 minutes to read • +5

In this article

The ASP.NET Core templates create a .NET Core Generic Host (HostBuilder).

This topic provides information on using .NET Generic Host in ASP.NET Core. For information on using .NET Generic Host in console apps, see .NET Generic Host.

Host definition

A *host* is an object that encapsulates an app's resources, such as:

- Dependency injection (DI)
- Logging
- Configuration
- `IHostedService` implementations

When a host starts, it calls IHostedService.StartAsync on each implementation of IHostedService registered in the service container's collection of hosted services. In a web app, one of the `IHostedService` implementations is a web service that starts an HTTP server implementation.

The main reason for including all of the app's interdependent resources in one object is lifetime management: control over app startup and graceful shutdown.

Set up a host

The host is typically configured, built, and run by code in the `Program` class. The `Main` method:

- Calls a `CreateHostBuilder` method to create and configure a builder object.
- Calls `Build` and `Run` methods on the builder object.

The ASP.NET Core web templates generate the following code to create a Generic Host:

C# Copy

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
```

```
            {
                webBuilder.UseStartup<Startup>();
            });
}
```

The following code creates a Generic Host using non-HTTP workload. The `IHostedService` implementation is added to the DI container:

C# Copy

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureServices((hostContext, services) =>
            {
               services.AddHostedService<Worker>();
            });
}
```

For an HTTP workload, the `Main` method is the same but `CreateHostBuilder` calls `ConfigureWebHostDefaults`:

C# Copy

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

The preceding code is generated by the ASP.NET Core templates.

If the app uses Entity Framework Core, don't change the name or signature of the `CreateHostBuilder` method. The Entity Framework Core tools expect to find a `CreateHostBuilder` method that configures the host without running the app. For more information, see Design-time DbContext Creation.

Default builder settings

The CreateDefaultBuilder method:

- Sets the content root to the path returned by GetCurrentDirectory.
- Loads host configuration from:
  - Environment variables prefixed with `DOTNET_`.
  - Command-line arguments.
- Loads app configuration from:
  - *appsettings.json*.
  - *appsettings.{Environment}.json*.
  - Secret Manager when the app runs in the `Development` environment.
  - Environment variables.
  - Command-line arguments.
- Adds the following logging providers:
  - Console
  - Debug
  - EventSource
  - EventLog (only when running on Windows)
- Enables scope validation and dependency validation when the environment is Development.

The `ConfigureWebHostDefaults` method:

- Loads host configuration from environment variables prefixed with `ASPNETCORE_`.
- Sets Kestrel server as the web server and configures it using the app's hosting configuration providers. For the Kestrel server's default options, see Kestrel web server implementation in ASP.NET Core.
- Adds Host Filtering middleware.
- Adds Forwarded Headers middleware if `ASPNETCORE_FORWARDEDHEADERS_ENABLED` equals `true`.
- Enables IIS integration. For the IIS default options, see Host ASP.NET Core on Windows with IIS.

The Settings for all app types and Settings for web apps sections later in this article show how to override default builder settings.

Framework-provided services

The following services are registered automatically:

- IHostApplicationLifetime
- IHostLifetime
- IHostEnvironment / IWebHostEnvironment

For more information on framework-provided services, see Dependency injection in ASP.NET Core.

IHostApplicationLifetime

Inject the IHostApplicationLifetime (formerly `IApplicationLifetime`) service into any class to handle post-startup and graceful shutdown tasks. Three properties on the interface are cancellation tokens used to register app start and app stop event handler methods. The interface also includes a `StopApplication` method.

The following example is an `IHostedService` implementation that registers `IHostApplicationLifetime` events:

C# Copy

```
internal class LifetimeEventsHostedService : IHostedService
{
    private readonly ILogger _logger;
    private readonly IHostApplicationLifetime _appLifetime;

    public LifetimeEventsHostedService(
        ILogger<LifetimeEventsHostedService> logger,
        IHostApplicationLifetime appLifetime)
    {
        _logger = logger;
        _appLifetime = appLifetime;
    }

    public Task StartAsync(CancellationToken cancellationToken)
    {
        _appLifetime.ApplicationStarted.Register(OnStarted);
        _appLifetime.ApplicationStopping.Register(OnStopping);
        _appLifetime.ApplicationStopped.Register(OnStopped);

        return Task.CompletedTask;
    }

    public Task StopAsync(CancellationToken cancellationToken)
    {
        return Task.CompletedTask;
    }

    private void OnStarted()
    {
        _logger.LogInformation("OnStarted has been called.");

        // Perform post-startup activities here
    }

    private void OnStopping()
    {
        _logger.LogInformation("OnStopping has been called.");
```

```
            // Perform on-stopping activities here
        }

        private void OnStopped()
        {
            _logger.LogInformation("OnStopped has been called.");

            // Perform post-stopped activities here
        }
    }
```

IHostLifetime

The IHostLifetime implementation controls when the host starts and when it stops. The last implementation registered is used.

`Microsoft.Extensions.Hosting.Internal.ConsoleLifetime` is the default `IHostLifetime` implementation. `ConsoleLifetime`:

- Listens for `Ctrl`+`C`/SIGINT or SIGTERM and calls StopApplication to start the shutdown process.
- Unblocks extensions such as RunAsync and WaitForShutdownAsync.

IHostEnvironment

Inject the IHostEnvironment service into a class to get information about the following settings:

- ApplicationName
- EnvironmentName
- ContentRootPath

Web apps implement the `IWebHostEnvironment` interface, which inherits `IHostEnvironment` and adds the WebRootPath.

Host configuration

Host configuration is used for the properties of the IHostEnvironment implementation.

Host configuration is available from HostBuilderContext.Configuration inside ConfigureAppConfiguration. After `ConfigureAppConfiguration`, `HostBuilderContext.Configuration` is replaced with the app config.

To add host configuration, call ConfigureHostConfiguration on `IHostBuilder`. `ConfigureHostConfiguration` can be called multiple times with additive results. The host uses whichever option sets a value last on a given key.

The environment variable provider with prefix `DOTNET_` and command-line arguments are included by `CreateDefaultBuilder`. For web apps, the environment variable provider with prefix `ASPNETCORE_` is added. The prefix is removed when the environment variables are read. For example, the environment variable value for `ASPNETCORE_ENVIRONMENT` becomes the host configuration value for the `environment` key.

The following example creates host configuration:

C# Copy

```
// using Microsoft.Extensions.Configuration;

Host.CreateDefaultBuilder(args)
    .ConfigureHostConfiguration(configHost =>
    {
        configHost.SetBasePath(Directory.GetCurrentDirectory());
        configHost.AddJsonFile("hostsettings.json", optional: true);
        configHost.AddEnvironmentVariables(prefix: "PREFIX_");
        configHost.AddCommandLine(args);
    });
```

App configuration

App configuration is created by calling ConfigureAppConfiguration on `IHostBuilder`. `ConfigureAppConfiguration` can be called multiple times with additive results. The app uses whichever option sets a value last on a given key.

The configuration created by `ConfigureAppConfiguration` is available at HostBuilderContext.Configuration for subsequent operations and as a service from DI. The host configuration is also added to the app configuration.

For more information, see Configuration in ASP.NET Core.

Settings for all app types

This section lists host settings that apply to both HTTP and non-HTTP workloads. By default, environment variables used to configure these settings can have a `DOTNET_` or `ASPNETCORE_` prefix.

ApplicationName

The IHostEnvironment.ApplicationName property is set from host configuration during host construction.

Key: `applicationName`
Type: `string`
Default: The name of the assembly that contains the app's entry point.
Environment variable: `<PREFIX_>APPLICATIONNAME`

To set this value, use the environment variable.

ContentRoot

The IHostEnvironment.ContentRootPath property determines where the host begins searching for content files. If the path doesn't exist, the host fails to start.

Key: `contentRoot`

Type: `string`

Default: The folder where the app assembly resides.

Environment variable: `<PREFIX_>CONTENTROOT`

To set this value, use the environment variable or call `UseContentRoot` on `IHostBuilder`:

C# Copy

```
Host.CreateDefaultBuilder(args)
    .UseContentRoot("c:\\content-root")
    //...
```

For more information, see:

- Fundamentals: Content root
- WebRoot

EnvironmentName

The IHostEnvironment.EnvironmentName property can be set to any value. Framework-defined values include `Development`, `Staging`, and `Production`. Values aren't case-sensitive.

Key: `environment`

Type: `string`

Default: `Production`

Environment variable: `<PREFIX_>ENVIRONMENT`

To set this value, use the environment variable or call `UseEnvironment` on `IHostBuilder`:

Copy

```
C#
Host.CreateDefaultBuilder(args)
    .UseEnvironment("Development")
    //...
```

ShutdownTimeout

HostOptions.ShutdownTimeout sets the timeout for StopAsync. The default value is five seconds. During the timeout period, the host:

- Triggers IHostApplicationLifetime.ApplicationStopping.
- Attempts to stop hosted services, logging errors for services that fail to stop.

If the timeout period expires before all of the hosted services stop, any remaining active services are stopped when the app shuts down. The services stop even if they haven't finished processing. If services require additional time to stop, increase the timeout.

Key: `shutdownTimeoutSeconds`
Type: `int`
Default: 5 seconds
Environment variable: `<PREFIX_>SHUTDOWNTIMEOUTSECONDS`

To set this value, use the environment variable or configure `HostOptions`. The following example sets the timeout to 20 seconds:

```
C#                                                                    Copy

Host.CreateDefaultBuilder(args)
    .ConfigureServices((hostContext, services) =>
    {
        services.Configure<HostOptions>(option =>
        {
```

```
            option.ShutdownTimeout = System.TimeSpan.FromSeconds(20);
        });
    });
```

Settings for web apps

Some host settings apply only to HTTP workloads. By default, environment variables used to configure these settings can have a `DOTNET_` or `ASPNETCORE_` prefix.

Extension methods on `IWebHostBuilder` are available for these settings. Code samples that show how to call the extension methods assume `webBuilder` is an instance of `IWebHostBuilder`, as in the following example:

C# Copy

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

CaptureStartupErrors

When `false`, errors during startup result in the host exiting. When `true`, the host captures exceptions during startup and attempts to start the server.

Key: `captureStartupErrors`
Type: `bool` (`true` or `1`)

Default: Defaults to `false` unless the app runs with Kestrel behind IIS, where the default is `true`.
Environment variable: `<PREFIX_>CAPTURESTARTUPERRORS`

To set this value, use configuration or call `CaptureStartupErrors`:

```
C#
webBuilder.CaptureStartupErrors(true);
```

DetailedErrors

When enabled, or when the environment is `Development`, the app captures detailed errors.

Key: `detailedErrors`
Type: `bool` (`true` or `1`)
Default: `false`
Environment variable: `<PREFIX_>_DETAILEDERRORS`

To set this value, use configuration or call `UseSetting`:

```
C#
webBuilder.UseSetting(WebHostDefaults.DetailedErrorsKey, "true");
```

HostingStartupAssemblies

A semicolon-delimited string of hosting startup assemblies to load on startup. Although the configuration value defaults to an empty string, the hosting startup assemblies always include the app's assembly. When hosting startup assemblies are provided, they're added to the app's assembly for loading when the app builds its common services during startup.

Key: `hostingStartupAssemblies`

Type: `string`

Default: Empty string

Environment variable: `<PREFIX_>_HOSTINGSTARTUPASSEMBLIES`

To set this value, use configuration or call `UseSetting`:

C# Copy

```
webBuilder.UseSetting(WebHostDefaults.HostingStartupAssembliesKey, "assembly1;assembly2");
```

HostingStartupExcludeAssemblies

A semicolon-delimited string of hosting startup assemblies to exclude on startup.

Key: `hostingStartupExcludeAssemblies`

Type: `string`

Default: Empty string

Environment variable: `<PREFIX_>_HOSTINGSTARTUPEXCLUDEASSEMBLIES`

To set this value, use configuration or call `UseSetting`:

C# Copy

```
webBuilder.UseSetting(WebHostDefaults.HostingStartupExcludeAssembliesKey, "assembly1;assembly2");
```

HTTPS_Port

The HTTPS redirect port. Used in enforcing HTTPS.

Key: `https_port`
Type: `string`
Default: A default value isn't set.
Environment variable: `<PREFIX_>HTTPS_PORT`

To set this value, use configuration or call `UseSetting`:

C# Copy

```
webBuilder.UseSetting("https_port", "8080");
```

PreferHostingUrls

Indicates whether the host should listen on the URLs configured with the `IWebHostBuilder` instead of those URLs configured with the `IServer` implementation.

Key: `preferHostingUrls`
Type: `bool` (`true` or `1`)
Default: `true`
Environment variable: `<PREFIX_>_PREFERHOSTINGURLS`

To set this value, use the environment variable or call `PreferHostingUrls`:

C# Copy

```
webBuilder.PreferHostingUrls(false);
```

PreventHostingStartup

Prevents the automatic loading of hosting startup assemblies, including hosting startup assemblies configured by the app's assembly. For more information, see Use hosting startup assemblies in ASP.NET Core.

Key: `preventHostingStartup`
Type: `bool` (`true` or `1`)
Default: `false`
Environment variable: `<PREFIX_>_PREVENTHOSTINGSTARTUP`

To set this value, use the environment variable or call `UseSetting` :

C# Copy

```
webBuilder.UseSetting(WebHostDefaults.PreventHostingStartupKey, "true");
```

StartupAssembly

The assembly to search for the `Startup` class.

Key: `startupAssembly`
Type: `string`
Default: The app's assembly
Environment variable: `<PREFIX_>STARTUPASSEMBLY`

To set this value, use the environment variable or call `UseStartup`. `UseStartup` can take an assembly name (`string`) or a type (`TStartup`). If multiple `UseStartup` methods are called, the last one takes precedence.

C# Copy

```
webBuilder.UseStartup("StartupAssemblyName");
```

C# Copy

```
webBuilder.UseStartup<Startup>();
```

URLs

A semicolon-delimited list of IP addresses or host addresses with ports and protocols that the server should listen on for requests. For example, `http://localhost:123`. Use "*" to indicate that the server should listen for requests on any IP address or hostname using the specified port and protocol (for example, `http://*:5000`). The protocol (`http://` or `https://`) must be included with each URL. Supported formats vary among servers.

Key: `urls`
Type: `string`
Default: `http://localhost:5000` and `https://localhost:5001`
Environment variable: `<PREFIX_>URLS`

To set this value, use the environment variable or call `UseUrls`:

C# Copy

```
webBuilder.UseUrls("http://*:5000;http://localhost:5001;https://hostname:5002");
```

Kestrel has its own endpoint configuration API. For more information, see Kestrel web server implementation in ASP.NET Core.

WebRoot

The IWebHostEnvironment.WebRootPath property determines the relative path to the app's static assets. If the path doesn't exist, a no-op file provider is used.

Key: `webroot`

Type: `string`

Default: The default is `wwwroot`. The path to *{content root}/wwwroot* must exist.

Environment variable: `<PREFIX_>WEBROOT`

To set this value, use the environment variable or call `UseWebRoot` on `IWebHostBuilder`:

C# Copy

```
webBuilder.UseWebRoot("public");
```

For more information, see:

- Fundamentals: Web root
- ContentRoot

Manage the host lifetime

Call methods on the built IHost implementation to start and stop the app. These methods affect all IHostedService implementations that are registered in the service container.

Run

Run runs the app and blocks the calling thread until the host is shut down.

RunAsync

RunAsync runs the app and returns a Task that completes when the cancellation token or shutdown is triggered.

RunConsoleAsync

RunConsoleAsync enables console support, builds and starts the host, and waits for `Ctrl`+`C`/SIGINT or SIGTERM to shut down.

Start

Start starts the host synchronously.

StartAsync

StartAsync starts the host and returns a Task that completes when the cancellation token or shutdown is triggered.

WaitForStartAsync is called at the start of `StartAsync`, which waits until it's complete before continuing. This can be used to delay startup until signaled by an external event.

StopAsync

StopAsync attempts to stop the host within the provided timeout.

WaitForShutdown

WaitForShutdown blocks the calling thread until shutdown is triggered by the IHostLifetime, such as via `Ctrl`+`C`/SIGINT or SIGTERM.

WaitForShutdownAsync

WaitForShutdownAsync returns a Task that completes when shutdown is triggered via the given token and calls StopAsync.

External control

Direct control of the host lifetime can be achieved using methods that can be called externally:

C# Copy

```
public class Program
{
    private IHost _host;

    public Program()
    {
        _host = new HostBuilder()
            .Build();
    }

    public async Task StartAsync()
    {
        _host.StartAsync();
    }

    public async Task StopAsync()
    {
        using (_host)
        {
            await _host.StopAsync(TimeSpan.FromSeconds(5));
        }
    }
}
```

Additional resources

- Background tasks with hosted services in ASP.NET Core