

Upgrading a .NET 5 "Startup-based" app to .NET 6

- [12 Part series](#)

Most of the questions are around the "minimal hosting" changes, and "minimal APIs", and what that means for their existing .NET 5 apps.

WebApplication and WebApplicationBuilder

For the new `WebApplication` and `WebApplicationBuilder` types suggest looking at the [second post in this series](#).

Typical minimal `Program.cs` that uses `WebApplication`

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();
app.MapGet("/", () => "Hello World!");
app.Run();
```

Options

1. Do nothing

- the old way still works, just make change like this =>
`<TargetFramework>net6.0</TargetFramework>`

2. Re-use your Startup class

- You can still use the new `WebApplication` style, but don't to put everything in `Program.cs`.

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

var startup = new Startup(builder.Configuration); // Manually create an instance
startup.ConfigureServices(builder.Services);      // Manually call ConfigureServi

var app = builder.Build();
```

```
//app.MapGet("/", () => "Hello World!");           // does this go away?

startup.Configure(app, app.Lifetime);               // Call Configure(), passing in tl
app.Run();
```



This is probably the simplest approach to re-use your Startup class if you want to shift to the new WebApplication approach. Beware, For example, you can't change settings like the app name or environment after you've created a WebApplicationBuilder. See the [docs](#) for more of these subtle differences.

3. Do Local methods in Program.cs

I probably wouldn't choose to create a Startup class, but I probably would add similar methods into my Program.cs file to give it some structure.

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

ConfigureConfiguration(builder.configuration);
ConfigureServices(builder.Services);

var app = builder.Build();

ConfigureMiddleware(app, app.Services);
ConfigureEndpoints(app, app.Services);

app.Run();

void ConfigureConfiguration(ConfigurationManager configuration) => { }
void ConfigureServices(IServiceCollection services) => { } // BlazorServerSamples.l
void ConfigureMiddleware(IApplicationBuilder app, IServiceProvider services) => { }
void ConfigureEndpoints(IEndpointRouteBuilder app, IServiceProvider services) => { }
```



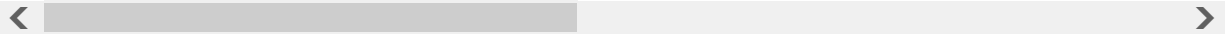
- I have separate methods for setting up middleware (which is sensitive to order) and endpoint (not sensitive to order)
- I used the `IApplicationBuilder` and `IEndpointRouteBuilder` types in the method signatures to enforce it.
- It's easy to update the method signatures or break these out if we need more flexibility.

Notes about the above code block

1. **ConfigurationManager**

- Added in .Net 6 to support ASP.NET Core's new `WebApplication` model

```
void ConfigureConfiguration(ConfigurationManager configuration) => { }  
// Hadn't used this before, see [Looking inside ConfigurationManager in .NET 6](http
```



2. **IServiceCollection**

- code referenced above
- see `ServiceCollectionExtensions.txt`

```
void ConfigureServices(IServiceCollection services) => { }  
  
/*  
  ServiceCollectionExtensions.txt  
  public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
*/
```

3. **ConfigureMiddleware**

- code referenced above
- see `ServiceCollectionExtensions.txt`

```
void ConfigureMiddleware(IApplicationBuilder app, IServiceProvider services) => {
```



4. **ConfigureEndpoints**

- code referenced above

```
void ConfigureEndpoints(IEndpointRouteBuilder app, IServiceProvider services) => {
```



Old .net 5 **Startup.cs**

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapRazorPages();
    endpoints.MapBlazorHub();
    endpoints.MapFallbackToPage("/_Host");
});
```

Part 1 Looking inside ConfigurationManager in .NET 6

ConfigurationManager was added to support ASP.NET Core's new `WebApplication` model, used for simplifying the ASP.NET Core startup code. However ConfigurationManager is very much an **implementation detail**. It was introduced to optimise a specific scenario (which I'll describe shortly), but for the most part, you don't need to (and won't) know you're using it.