/23/goodbye-

/2 getting- (https://dotnetcoretutorials.com/)

/10/basic-

Hosting An ASP.NET Core Web Application In IIS

DECEMBER 23, 2019 (HTTPS://DOTNETCORETUTORIALS.COM/2019/12/23/HOSTING-AN-ASP-NET-CORE-WEB-APPLICATION-IN-IIS/) BY WADE (HTTPS://DOTNETCORETUTORIALS.COM/AUTHOR/ADMIN/) · 11 COMMENTS (HTTPS://DOTNETCORETUTORIALS.COM/2019/12/23/HOSTING-AN-ASP-NET-CORE-WEB-APPLICATION-IN-IIS/#COMMENTS)

/02/using-

/02/us

/19/using-

For the past few years I've been almost exclusively using Azure's PAAS Websites to host my .NET Core applications. Whereby I set up my Azure Devops instance to point to my Azure Website, and at the click of the button my application is deployed and you don't really have to think too hard about "how" it's being hosted.

/20/publishing-

Well, recently I had to set up a .NET Core application to run on a fresh server behind IIS and while relatively straight forward, there were a few things I wish I knew beforehand. Nothing's too hard, but some guides out there are waaayyy overkill and take hours to read let alone implement what they are saying. So hopefully this is a bit more of a straight forward guide.

/11/how-

/04/csv-

You Need The ASP.NET Core Hosting Bundle

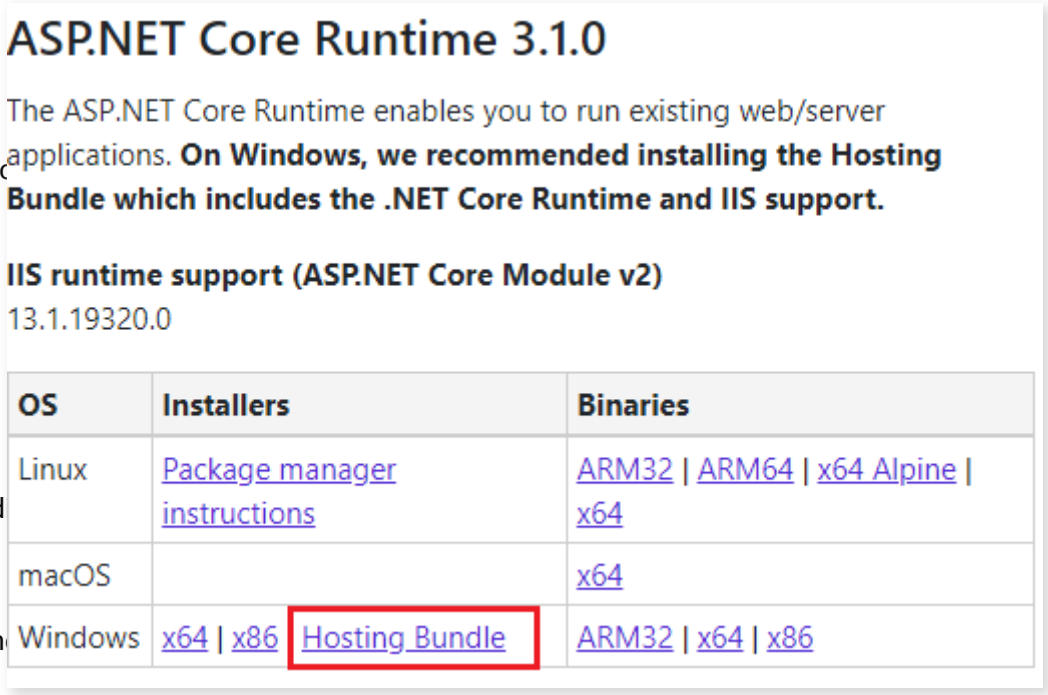
One thing that I got stuck on early on was that for .NET Core to work inside IIS, you actually need to do an install of a “Hosting Module” so that IIS knows how to run your app.

This actually frustrated me a bit at first because I wanted to do “Self Contained” deploys where everything the app needed to run was published to the server. So… If I’m publishing what essentially amounts to the full runtime with my app, why the hell do I still need to install stuff on the server!? But, it makes sense. IIS can’t just magically know how to forward requests to your app, it needs just a tiny bit of help. Just incase someone is skimming this post, I’m going to bold it :

Self contained .NET Core applications on IIS still need the ASP.NET Core hosting bundle

So where do you get this “bundle”. Annoyingly it’s not on the main .NET Core homepage and you need to go to the specific version to get the latest version. For example here : https://dotnet.microsoft.com/download/dotnet-core/3.1 (https://dotnet.microsoft.com/download/dotnet-core/3.1).

It can be maddening trying to find this particular download link. It will be on the right hand side buried in the runtime for Windows details.



ASP.NET Core Runtime 3.1.0

The ASP.NET Core Runtime enables you to run existing web/server applications. **On Windows, we recommended installing the Hosting Bundle which includes the .NET Core Runtime and IIS support.**

IIS runtime support (ASP.NET Core Module v2)
13.1.19320.0

OS	Installers	Binaries
Linux	Package manager instructions	ARM32 | ARM64 | x64 Alpine | x64
macOS		x64
Windows	x64 | x86 Hosting Bundle	ARM32 | x64 | x86

Note that the “bundle” is the module packaged with the .NET Core runtime. So once you’ve installed this, for now atleast, self contained deployments aren’t so great because you’ve just installed the runtime anyway. Although for minor version bumps it’s handy to keep doing self contained deploys because you won’t have to always keep pace with the runtime versions on the server.

After installing the .NET Core hosting bundle you must restart the server OR run an IISReset. Do not forget to do this!

In Process vs Out Of Process

So you've probably heard of the term "In Process" being bandied about in relation to .NET Core hosting for a while now. I know when it first came out in .NET Core 2.2, I read a bit about it but it wasn't the "default" so didn't take much notice. Well now the tables have turned so to speak, so let me explain.

From .NET Core 1.X to 2.2, the default way IIS hosted a .NET Core application was by running an instance of Kestrel (The .NET Core inbuilt web server), and forwarding the requests from IIS to Kestrel. Basically IIS acted as a proxy. This works but it's slow since you're essentially doing a double hop from IIS to Kestrel to serve the request. This method of hosting was dubbed "Out Of Process".

In .NET Core 2.2, a new hosting model was introduced called "In Process". Instead of IIS forwarding the requests on to Kestrel, it serves the requests from within IIS. This is much faster at processing requests because it doesn't have to forward on the request to Kestrel. This was an optional feature you could turn on by using your csproj file.

Then in .NET Core 3.X, nothing changed per-say in terms of how things were hosted. But the defaults were reversed so now In Process was the default and you could use the csproj flag to run everything as Out Of Process again.

Or in tabular form :

Version	Supports Out Of Process	Supports In Process	Default
.NET Core <2.2	Yes	No	N/A
.NET Core 2.2	Yes	Yes	Out Of Process
.NET Core 3.X	Yes	Yes	In Process

Now to override the defaults, you can add the following to your csproj file (Picking the correct hosting model you want).

```
<PropertyGroup>
  <AspNetCoreHostingModel>InProcess/OutOfProcess</AspNetCoreHostingModel>
</PropertyGroup>
```

As to which one you should use? Typically, unless there is a specific reason you don't want to use it, InProcess will give you much better performance and is the default in .NET Core 3+ anyway.

After reading this section you are probably sitting there thinking… Well.. So I'm just going to use the default anyway so I don't need to do anything? Which is true. But many guides spend a lot of time explaining the hosting models and so you'll definitely be asked questions about it from a co-worker, boss, tech lead etc. So now you know!

UseIIS vs UseIISIntegration

There is one final piece to cover before we actually get to setting up our website. Now ***before*** we got the "CreateDefaultBuilder (https://dotnetcoretutorials.com/2019/07/31/what-does-the-createdefaultbuilder-method-do-in-asp-net-core/)" method as the default template in .NET Core, you had to build your processing pipeline yourself. So in your program.cs file you would have something like :

```
var host = new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .UseIISIntegration()
        .UseStartup<Startup>()
        .Build();
```

So here we can actually see that there is a call to UseIISIntegration . There is actually another call you may see out in the wild called UseIIS without the integration. What's the difference? It's actually quite simple. UseIISIntegration sets up the out of process hosting model, and UseIIS sets up the InProcess model. So in theory, you pick one or the other but in practice CreateDefaultBuilder actually calls them both and later on the SDK works out which one you are going to use based on the default or your csproj flag described above (More on that in the section below).

So again, something that will be handled for you by default, but you may be asked a question about.

Web.Config Shenanigans

One issue we have is that for IIS to understand how to talk to .NET Core, it needs a web.config file. Now if you're using IIS to simply host your application but not using any additional IIS features, your application probably doesn't have a web.config to begin with. So here's what the .NET Core SDK does.

If you do not have a web.config in your application, when you **publish** your application, .NET Core will add one for you. It will contain details for IIS on how to start your application and look a bit like this :

```
<configuration>
  <location path="." inheritInChildApplications="false">
    <system.webServer>
      <handlers>
        <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModuleV2" r
      </handlers>
      <aspNetCore processPath="dotnet" arguments=".\MyTestApplication.dll" stdo
    </system.webServer>
  </location>
</configuration>
```

So all it's doing is adding a handler for IIS to be able to run your application (Also notice it sets the hosting model to InProcess – which is the default as I'm running .NET Core 3.X).

If you do have a web.config, it will then append/modify your web.config to add in the the handler on publish. So for example if you are using web.config to configure.. I don't know, mime types. Or maybe using some basic windows authorization. Then it's basically going to append in the handler to the bottom of your own web.config.

There's also one more piece to the puzzle. If for some reason you decide that you want to add in the handler yourself (e.g. You want to manage the arguments passed to the dotnet command), then you can actually copy and paste the above into your own web.config.

But. There is a problem.

The .NET Core SDK will also always try and modify this web.config on publish to be what it *thinks* the handler should look like. So for example I copied the above and fudged the name of the DLL it was passing in as an argument. I published and ended up with this :

```
arguments=".\MyTestApplication.dll .\MyTestApplicationasd.dll"
```

Notice how it's gone "OK, you are running this weird dll called MyTestApplicationasd.dll, but I think you should run MyTestApplication.dll instead so I'm just gonna add that for you". Bleh! But there is a way to disable this!

Inside your csproj you can add a special flag like so :

```
<PropertyGroup>
  <TargetFramework>netcoreapp3.0</TargetFramework>
  <IsTransformWebConfigDisabled>true</IsTransformWebConfigDisabled>
</PropertyGroup>
```

This tells the SDK don't worry, I got this. And it won't try and add in what it thinks your app needs to run under IIS.

Again, another section on “You may need to know this in the future”. If you don’t use web.config at all in your application then it’s unlikely you would even realize that the SDK generates it for you when publishing. It’s another piece of the puzzle that happens in the background that may just help you in the future understand what’s going on under the hood when things break down.

An earlier version of this section talked about adding your own web.config to your project so you could point IIS to your debug folder. On reflection, this was bad advice. I always had issues with projects locking and the “dotnet build” command not being quite the same as the “dotnet publish”. So for that reason, for debugging, I recommend sticking with IIS Express (F5), or Kestrel by using the dotnet run command.

IIS Setup Process

Now you’ve read all of the above and you are ready to actually set up your website. Well that’s the easy bit!

First create your website in IIS as you would a standard .NET Framework site :

You’ll notice that I am pointing to the ***publish*** folder. As described in the section above about web.config, this is because my particular application does not have a web.config of it’s own and therefore I cannot just point to my regular build folder, even if I’m just testing things out. I need to point to the publish folder where the SDK has generated a web.config for me.

You’ll also notice that in my case, I’m creating a new Application Pool. This is semi-important and I’ll show you why in a second.

Once you've create your website. Go to your Application Pool list, select your newly created App Pool, and hit "Basic Settings". From there, you need to ensure that .NET CLR Version is set to "No Managed Code". This tells IIS not to kick off the .NET Framework pipeline for your .NET Core app.

Obviously if you want to use shared application pools, then you should create a .NET Core app pool that sets up No Managed Code.

And that's it! That's actually all you need to know to get up and running using IIS to host .NET Core! In a future post I'll actually go through some troubleshooting steps, most notably the dreaded HTTP Error 403.14 which can mean an absolute multitude of things.

Related Posts

1. **Hosting An ASP.NET Core Web Application As A Windows Service In .NET Core 2 (https://dotnetcoretutorials.com/2018/09/12/hosting-an-asp-net-core-web-application-as-a-windows-service/)**
2. **Hosting An ASP.NET Core Web App As A Windows Service In .NET Core 3 (https://dotnetcoretutorials.com/2019/12/21/hosting-an-asp-net-core-web-app-as-a-windows-service-in-net-core-3/)**
3. **Kestrel vs IIS (https://dotnetcoretutorials.com/2019/12/25/kestrel-vs-iis/)**
4. **Set X-Content-Type-Options in ASP.net Core (https://dotnetcoretutorials.com/2017/01/20/set-x-content-type-options-asp-net-core/)**

11 COMMENTS

Mehdi

December 27, 2019 at 4:58 am (https://dotnetcoretutorials.com/2019/12/23/hosting-an-asp-net-core-web-application-in-iis/#comment-20536)

Hi,
Thank you for your posts. keep it up!

I tried to host my project in IIS, localy (from debug folder). I added web.config to the root of the project with the code you mentioned and do the rest steps. it works nicely. but all static file requests return 404.
I tried the solution which this article https://weblogs.asp.net/imranbaloch/leveraging-iis-static-file-feature-in-aspnetcore (https://weblogs.asp.net/imranbaloch/leveraging-iis-static-file-feature-in-aspnetcore) says. but I got internal server error.

How I can fix this?

Reply

Wade

December 27, 2019 at 11:45 am (https://dotnetcoretutorials.com/2019/12/23/hosting-an-asp-net-core-web-application-in-iis/#comment-20545)

Interesting. And static files work when you just run from Kestrel? I honestly haven't run into this because the only things I build now are SPAs so it's always just an API. I'll have a try and let you know.

Reply

Mehdi

December 27, 2019 at 6:19 pm (https://dotnetcoretutorials.com/2019/12/23/hosting-an-asp-net-core-web-application-in-iis/#comment-20550)

> And static files work when you just run from Kestrel?

Yes.

Actually I back to Kestrel for now, because I didn't find any solution.

> I'll have a try and let you know.

would be great, thanks.

Reply

Wade

December 31, 2019 at 8:54 am (https://dotnetcoretutorials.com/2019/12/23/hosting-an-asp-net-core-web-application-in-iis/#comment-20735)

So I had a bit of a play. Honestly, I think I made a mistake saying it's fine to point IIS at the debug folder because I had a hell of a time with files locking all the time etc by IIS. So… I'd probably not do that.

But the reason your static files don't work is because if you check the debug folder, they won't be in there. The wwwroot is only copied when publishing but not when building it seems. So if you are trying to point IIS at the debug folder from a straight build (Not a publish), things won't work. I'm going to edit the above with this info.

Reply

Mehdi

January 1, 2020 at 2:57 am (https://dotnetcoretutorials.com/2019/12/23/hosting-an-asp-net-core-web-application-in-iis/#comment-20761)

Thanks bro for your time.

IIS express works fine, but I have a problem with it. when I hit Ctrl+F5, IIS express run and open the url (e.g localhost:5235) in browser. now I can change the code, build the project and refresh the browser to see the updated result, so far so good. but if I rebuild, actually clean and build. refreshing browser won't work anymore, I have to hit Ctrl+F5 again (withing visual studio) to see the site again.
Is there any way to avoid this annoying Ctrl+F5 thing?

Wade

January 1, 2020 at 8:54 am (https://dotnetcoretutorials.com/2019/12/23/hosting-an-asp-net-core-web-application-in-iis/#comment-20765)

Yes. The best way is to use Kestrel with the watch feature. I have a really old article here which I'll update shortly with new details on it : https://dotnetcoretutorials.com/2017/01/31/live-coding-net-core-using-dotnet-watch/ (https://dotnetcoretutorials.com/2017/01/31/live-coding-net-core-using-dotnet-watch/)

But in short, in your project folder in a command prompt, you just need to run "dotnet watch", which essentially just watches for changes similar to Angulars "Serve" or Gulp/Grunt watch etc. And will auto recompile and server changes.

Sergio M

January 31, 2020 at 4:00 pm (https://dotnetcoretutorials.com/2019/12/23/hosting-an-asp-net-core-web-application-in-iis/#comment-22290)

Hi Wade,

I'm trying publishing my .net core on Asphostportal, but I receive 403 forbidden error message. Is there anything that I missed? They tried to publish sample .net core and it is working fine, but my application can't work. Any insight?

Thank you

Reply

Wade

January 31, 2020 at 6:22 pm (https://dotnetcoretutorials.com/2019/12/23/hosting-an-asp-net-core-web-application-in-iis/#comment-22300)

If you could upload a minimal example up to Github or similar I can take a look for you (Can also email me at wade (at) dotnetcoretutorials.com ☺ First thought would be either you aren't selecting the right publish folder (It

can be a bit of a maze), or that your API is working but it's returning 403 as designed (e.g. you are hitting an endpoint you shouldn't be or loading the base URL which loads a 403 forbidden etc).

Reply

John

May 2, 2020 at 12:52 pm (https://dotnetcoretutorials.com/2019/12/23/hosting-an-asp-net-core-web-application-in-iis/#comment-25662)

Hi Wade,
Have you every added as asp.net core application as an application inside of say the Default Web Site? My ajax call to a action method will get a 401 error and require to authenticate again to that action method. I could get around this by decorating the action method method with [AllowAnony,mous] but this is not ideal. If the asp.net core application is setup as a website by itself then the ajax calls work fine. Any suggestion on how to fix this issue?

Best,

Reply

Sunder

May 24, 2020 at 6:31 am (https://dotnetcoretutorials.com/2019/12/23/hosting-an-asp-net-core-web-application-in-iis/#comment-26301)

Hi,

I am using ASP.NET Core 3.0 Angular 8. I am trying to publish production build to IIS. All my assets CSS JS are in ClientApp/dist folder when I run the application. All the CSS and js files are fall back to index.html. I tried re-write RULE but no luck. Can you please suggest if anyone got into this issue?

Reply

Wade

May 24, 2020 at 8:52 am (https://dotnetcoretutorials.com/2019/12/23/hosting-an-asp-net-core-web-application-in-iis/#comment-26304)

This one may be useful
https://dotnetcoretutorials.com/2017/04/28/serving-static-files-outside-wwwroot-asp-net-core/

(https://dotnetcoretutorials.com/2017/04/28/serving-static-files-outside-wwwroot-asp-net-core/) I think it's more likely an issue of serving static files outside the regular wwwRoot.

Reply

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Post Comment

Privacy Policy (/privacy-policy/)