ASP.NET 5 Overview

ASP.NET 5 is a significant redesign of ASP.NET. This topic introduces the new concepts in ASP.NET 5 and explains how they help you develop modern web apps.

Introduction to ASP.NET 5

ASP.NET 5 is a lean .NET stack for building modern web apps. We built it from the ground up to provide an optimized development framework for apps that are either deployed to the cloud or run on-premises. It consists of modular components with minimal overhead, so you retain flexibility while constructing your solutions.

ASP.NET 5 includes the following features:

* New flexible and cross-platform runtime
* New modular HTTP request pipeline
* Cloud-ready environment configuration
* Unified programming model that combines MVC, Web API, and Web Pages
* Ability to see changes without re-building the project
* Side-by-side versioning of the .NET Framework
* Ability to self-host or host on IIS
* New tools in Visual Studio 2015
* Open source in GitHub

The changes we made for ASP.NET 5 were based on customer requests and feedback. These changes simplify development, hosting, and maintenance, and are targeted to modern web apps.

Your legacy apps will run on the new version of the ASP.NET without any modifications. However, to take advantage of the new features in ASP.NET 5, you will need to port your existing code to the new framework. You will find many similarities between ASP.NET 5 and earlier versions of ASP.NET, so porting code involves fixing particular issues rather than re-writing the app.

This topic provides an orientation to ASP.NET 5 and explanations of the changes.

Download Visual Studio 2015 CTP 6

You can begin working with ASP.NET 5 by [downloading](http://go.microsoft.com/fwlink/?LinkId=521794) Visual Studio 2015 CTP 6. For more information about what is included in this release, see the [Visual Studio 2015 CTP 6](http://www.visualstudio.com/en-us/news/vs2015-vs).

In this topic

* [Why redesign ASP.NET?](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#redesign)
  + [Need flexible, cross-platform runtime](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#runtime)
  + [Host anywhere](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#host)
  + [Use different versions of .NET side-by-side](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#side)
  + [Simplify dependency management](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#dependency)
  + [Eliminate duplication in MVC, Web API, and Web Pages](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#unify)
  + [Improve HTTP performance](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#http)
  + [Make it cloud-ready](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#cloud)
  + [Integrate dependency injection](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#injection)
  + [Make it open source and transparent](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#open)
  + [Provide agile development environment](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#agile)
* [What about Web Forms?](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#webforms)
* [What about legacy apps?](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#legacy)
* [Next steps](http://www.asp.net/vnext/overview/aspnet-vnext/aspnet-5-overview#next)

Why redesign ASP.NET?

### Need flexible, cross-platform runtime

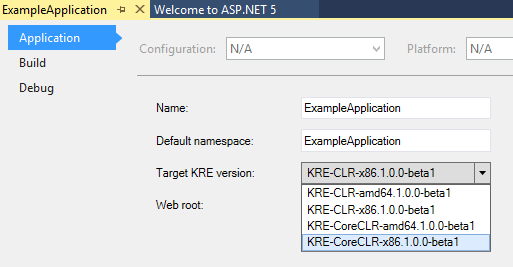
In the past, the .NET Framework was delivered as a single, all-encompassing installation. With each release of .NET, features were added but rarely removed, so the size of the framework continually grew. This approach ensured that a Windows machine with .NET installed could support every type of .NET app, but it also meant that each app depended on features which were not actually being used.

If a critical update was needed to one technology in .NET, you received a notification that an important update was available, even if you were not directly using this part of .NET. You had to decide whether to install this update and endure the disruption to your server or ignore the update and hope that you didn’t actually need it.

ASP.NET 5 gives you greater flexibility by being able to run on three runtimes:

1. **Full .NET CLR**  
   The full .NET CLR is the default runtime for projects in Visual Studio. It provides the entire API set and is your best option for backwards compatibility.
2. **Core CLR (cloud-optimized runtime)**  
   The Core CLR is a lean and completely modular runtime for ASP.NET 5 projects. This CLR has been re-designed into components so you have the flexibility to include only those features that you need in your app. You add the components as NuGet packages. When you are finished, your app is dependent only on required features. By re-factoring the runtime into separate components, we can deliver improvements to the components more quickly because each component is updated on its own schedule. The Core CLR is about 11 megabytes instead of around 200 megabytes for the full .NET CLR. The Core CLR can be deployed with your app and different versions of the Core CLR can run side-by-side (both of these advantages are described in greater detail below).
3. **Cross-Platform CLR**  
   We will release a cross-platform runtime for Linux and Mac OS X. When released, this runtime will enable you to develop and run .NET apps on Mac and Linux devices. We will work closely with the Mono community on this effort. Until its release, you can use the Mono CLR for cross-platform development. For more information, see [Develop ASP.NET vNext applications on a Mac](http://blogs.msdn.com/b/webdev/archive/2014/08/12/develop-asp-net-vnext-applications-on-a-mac.aspx).

By default, new Visual Studio projects use the full .NET CLR. You can specify the Core CLR in the configuration properties for your project.



### Host anywhere

ASP.NET 5 enables you to host your app on IIS or self-host your app in your own process. When you target the Core CLR, you can deploy your app with every dependency bundled within the deployment package. Therefore, your app and its dependencies are completely self-contained and no longer dependent on a system installation of .NET. Any type of device or hosting platform is capable of running the app.

This new capability gives you a lot of freedom. We still recommend IIS as the best option for hosting, but in some cases, you may need to use a different hosting platform. Now, you simply deploy your project to that host. Your hosting preference no longer dictates which development framework to use, and vice versa.

For an example of hosting an app outside of IIS, see [Create a Web API in MVC 6](http://www.asp.net/vnext/overview/aspnet-vnext/create-a-web-api-with-mvc-6).

### Use different versions of .NET side-by-side

When apps on a server depend on a single, system-wide installation of the .NET Framework, all of the apps have to run the same version of .NET. This situation might have created some anxiety for you when considering whether to upgrade to a new version of the .NET Framework. Perhaps, you wanted some of your apps to use the latest version of .NET but you were unsure whether all of your legacy apps would work appropriately with the new version.

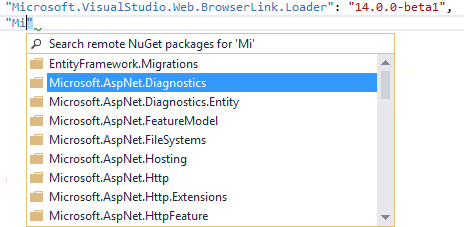
Fortunately, ASP.NET 5 fixes this problem. You can define the dependencies within your deployment package so you can specify for each app which version of .NET to use. You get the benefits of the latest version for some apps and the ease of sticking with an old version for other apps. All of these different versions run side-by-side without any problems. To run different versions side-by-side, you must target the Core CLR.

### Simplify dependency management

ASP.NET 5 introduces a new, lightweight way to manage dependencies in your projects. You no longer add assembly references to your project; instead, you manage dependencies by referencing NuGet packages. You can add NuGet packages through the NuGet Package Manager or you can edit the JSON file (project.json) that lists the NuGet packages and versions used in your project. To add other dependencies, you simply type the name and version number of the NuGet package into your project.json file.



In Visual Studio 2015, IntelliSense assists you with finding the available NuGet packages.



The project.json file only includes NuGet packages that you directly added to your project. If you add a NuGet package that is dependent on other packages, those secondary dependencies are loaded but not listed in the project.json file. This approach keeps your project.json file less cluttered and easier to manage. If you remove a NuGet package from project.json, the secondary dependencies are removed too if no other packages need them.

The JSON format makes it easier for you to manage dependencies even when you do not have an available installation of Visual Studio. You can open the project.json file in any type of text editor and make changes; such as updating dependencies for app deployed to the cloud.

### Eliminate duplication in MVC, Web API and Web Pages

In the past, MVC, Web API, and Web Pages contained overlapping features but the implementations of those features were separate. For example, MVC and Web API both provided routing but the MVC routing classes resided in the System.Web.Mvc.Routing namespace while similar classes for Web API resided in the System.Web.Http.Routing namespace. Or, Web Pages and MVC both used Razor syntax, but some NuGet packages were compatible with only one or the other.

In ASP.NET 5, MVC, Web API, and Web Pages will be merged into a single framework called MVC 6. This merging removes duplication from the framework and makes it easier for you to develop apps that use these programming frameworks. You no longer need to write slightly different code depending on whether you are within an MVC, Web API, or Web Pages context.

For this preview release, MVC and Web API have been unified in MVC 6. **Web Pages will be added to MVC 6 in a later release.**

### Improve HTTP performance

ASP.NET 5 introduces a new HTTP request pipeline that is lean and fast. This pipeline is modular so you can add only the components that you need. By reducing the overhead in the pipeline, your app will experience better throughput. The new pipeline also supports [OWIN](http://owin.org/).

### Make it cloud-ready

When you create a new ASP.NET 5 project, that project is structured for easy deployment to the cloud. Visual Studio 2015 provides a new environment configuration system that replaces the Web.config file. The new system enables you to request named values from a variety of sources (such as JSON, XML, or environment variables). You specify values for each environment, and after deployment your app simply reads the correct values.

We also provide diagnostic and tracing tools that make it easier for you to discover issues with your app in the cloud.

### Integrate dependency injection

Dependency injection is built into ASP.NET 5. You can use your Inversion of Control (IoC) container to register dependencies. Dependency injection facilitates providing the right services for the environment. For more information, see [Dependency Injection in ASP.NET vNext](http://blogs.msdn.com/b/webdev/archive/2014/06/17/dependency-injection-in-asp-net-vnext.aspx).

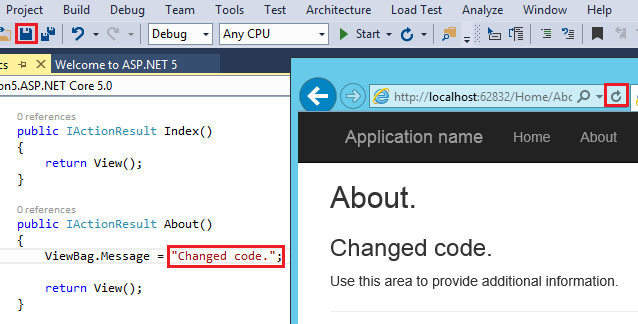
### Make it open source and transparent

All of the code for ASP.NET 5 is available through [GitHub](https://github.com/aspnet). Not only is the code available, but we use GitHub repositories for all of our development. You can see exactly which changes were made and when they were made. You can download the code and submit changes.

By developing ASP.NET 5 in GitHub we make it easier for you to understand the code, understand our intended direction, contribute enhancements, and build your own customized implementations of ASP.NET features.

### Provide agile development environment

Visual Studio 2015 provides a lightweight developer experience for ASP.NET apps. You simply make changes in your code, save the changes, and refresh the browser. You can see code changes in the web browser without re-building the project.



The code you change can be in the web project or in a class library referenced by your project. You must run (CTRL + F5) the project rather than be in debug mode to see the changes dynamically.

Visual Studio uses the [Roslyn compiler](http://msdn.microsoft.com/en-us/library/roslyn.aspx) to enable this dynamic compilation. You still have all of the structure and power of a compiled framework, but the development experience feels more like an interpreted language.

Every function within the Visual Studio user interface is matched with command line operations. You can easily transition between using the interface and writing command line scripts.

Finally, you can use other code editors to work on your ASP.NET 5 projects.

What about Web Forms?

You can continue developing Web Forms apps and have confidence that Web Forms is an essential part of the .NET web development platform. We remain focused on adding new features to Web Forms to improve the development experience and keep the technology up-to-date with web practices.

Web Forms 4.6 includes the following new features for Web Forms:

* HTTP 2
* Async model binding
* Roslyn CodeDOM compilers

Your existing Web Forms apps will continue to run without modification on IIS with .NET 4.6.

For a video about the new features in Web Forms 4.6, see [Web Forms 4.6](http://go.microsoft.com/fwlink/?LinkId=518773). For information about the many recent changes for Web Forms in Visual Studio 2013 Update 2, see [Improvements to ASP.NET Web Forms](http://blogs.msdn.com/b/webdev/archive/2014/05/13/improvements-to-asp-net-web-forms.aspx).

What about legacy apps?

You may be worried that, with the number of changes in ASP.NET 5, you now need to re-write all of your applications. Don’t worry. Applications that you built on earlier versions of ASP.NET will continue to work with the new .NET Framework. You do not need to update or port these applications if you do not need the new features in ASP.NET 5.

For example, your apps that currently use Web Forms, MVC 5, Web API 2, SignalR 2, Web Pages 3 or Entity Framework 6 are fully supported on the new framework without modification. However, you will most likely need to use the full .NET CLR to run legacy apps because only this CLR provides full compatibility with earlier versions.

The cloud-optimized runtime provides a more limited API surface. To use the cloud-optimized runtime, your application must use only types and members that are available in that runtime.

To see if your application can run on the cloud-optimized runtime, use the [API Portability Analyzer](http://go.microsoft.com/fwlink/?LinkID=398760). This tool tells you which platforms your application can target and which dependencies prevent your application from running on other platforms. It helps you understand the scope of required changes and suggests new types or members to use in place of unsupported ones.

MVC 6 and SignalR 3 apps use the new HTTP pipeline, so they are not compatible with apps that use System.Web. To upgrade an existing app to MVC 6 or SignalR 3, you must create a new project through Visual Studio 2015 and then port your code to the new project. When porting, you will need to modify unsupported code.