Hi this is Scott, and for this webinar Pluralsight asked me to talk about ASP.NET Core and the impact this new version of asp.net has on the enterprise. I’ll tell you up front I plan on dumping quite a bit of information from my brain to yours, and I hope you find this valuable in making decisions about the future.

When larger companies with larger development teams ask me about ASP.NET core, I generally frame the conversation in terms of risk and reward. Yes, we’ll talk about the new architecture and new features, but when working on critical business applications with long lifecycles, you want to pay attention to the risks as well as the rewards and be prepared for what you are getting into.

I feel there are six areas to consider in evaluating ASP.NET Core and it’s impact on your business and operations. First is understanding .NET Core and ASP.NET Core’s relationship with .NET Core, which is the new .NET framework. Second there is the new hosting model for ASP.NET. Third is the new HTTP processing pipeline in ASP.NET which you build completely from scratch using what we call middleware components. Fourth is the new design and architecture for implementing security requirements for user authentication and authorization. Fifth would be your data access considerations, because there is a new version of the entity Framework available in this new world. Lastly, there is ASP.NET Core itself. What advantages and disadvantages does the new framework bring ?

# ASP.NET Core and .NET Core

One of the first topics to understand about the new world is we now have two distinct flavors of the .NET framework to choose from. First there is the full .NET framework, this is the mature .NET framework that has been around since the beginning, this is the framework that comes pre-installed with Windows, and the most recent version of the full .NET framework is 4.6.2. The full .net framework includes application level frameworks like windows forms, web forms, wcp, and wpf.

Then we have .NET Core, a brand and modular version of .NET that runs on more than just the Windows operating system.

Where ASP.NET Core fits into the picture is that ASP.NET Core is an application framework that can run on either the full .NET framework or on .NET Core. Selecting a .NET framework flavor is one of your first decisions when making a move to ASP.NET Core. Do you want to run on the full framework, or .NET Core, or will you chose to support both?

When you choose to run on the full .NET framework you are running on the framework you already know today. Although ASP.NET Core will be a bit different from the ASP.NET frameworks of the past, your underlying framework is the same as it has been in the past. In order to understand why you might chose .NET Core instead of the full framework we need to dig into the benefits of .NET Core.

## The Rewards

.NET Core is a cross-platform framework, meaning.NET Core will run on Windows, on the Mac, and on various flavors of Linux. .NET Core also works in a Docker container for those who are using or thinking about using Docker software containers.

I want to make clear that ASP.NET Core does not require .NET Core as the underlying framework. and we’ll circle back to this topic later. But, if you *do* choose to use .NET Core as your underlying framework, you will be able to deploy applications on all of these various platforms. Linux is, of course, a big target for server-side applications. Most enterprises are heterogeneous and already have the IT expertise to run business applications on both Windows and Linux servers, so you can re-use this existing expertise. There is also the opportunity to save money as Linux servers typically run a bit cheaper, particularly when using cloud based infrastructure. On Azure, for example, a single 4 core virtual machine running Linux is around $90 cheaper per month than its Windows counterpart.

What’s not immediately obvious when talking about a cross-platform .NET framework is how the necessary tooling also works across platforms. .NET Core has no hard dependency on Windows or Visual Studio. All of the low level tools you need to do work will run from the command line, and there is an entire new world of text editors and IDEs available that we can now use to develop .NET applications. This includes Visual Studio code from Microsoft, Project Rider from Jet Brains, as well as editors like Sublime and Atom. I’ve worked on more than one project over the years where there is a front-end specialist who uses a Mac and this developer hasn’t been able to install Visual Studio and work with an ASP.NET project the same way a Windows developer would work with the project. This type of scenario is considerably easier with .NET Core because a developer on Apple hardware can write, run, and debug ASP.NET code just as well as a developer on Windows.

.NET Core is also an open source project. On GitHub you can find not just the code for the framework itself, but also the code for the unit tests, and the documentation. You can view bugs in the GitHub issues list and see the current status of a bug. I’ve been telling people that one of the underappreciated advantages of Microsoft’s move towards open source is not in having the source code to a framework or library. We’ve always been able to get source code, even if we had to use a de-compiler. There have been a number of times this year where I’ve found digging into the unit test code to be valuable. The unit tests have given me better insight into how a particular feature works compared to using documentation or the source code itself, because the unit tests can help describe a feature from several different perspectives and also tell me what a piece of software is not designed to do.

In the enterprise, there is often a worry that open source projects do not have the same level of support as closed source commercial products. However, Microsoft has announced a support lifecycle of 3 years for each major and minor release of .NET Core. The .NET Core 1.0.0 release was June 27, 2016, meaning the end of support for 1.0.0 is in June of 2019, or even later if there is a Long Term Support (LTS) release in the future.

Something you’ll notice when looking at the source code on GitHub is how modular the new .NET Core has become. This is no longer a monolithic framework where you have everything or you have nothing at all. With .NET Core the fundamental pieces are smaller and integrated with the NuGet package management ecosystem. You can pull in just the pieces of .NET Core that you need to build an application.

.NET Core also supports an application deployment model known as a self-contained deployment. A self-contained deployment means you can put your asp.net core application into production with all of the .NET core assemblies the application needs to function, and these framework assemblies are all local. There is no global assembly cache, there is no .NET Core installation required on the server. If you have micro-services architecture with a dozen asp.net core services deployment on the server, each service can deploy with it’s own version of the .NET framework and never need to worry about conflicting with the versions of other services or requiring the other services to upgrade. Everyone lives in perfect isolation. Of course there is a downside here,

With all of this goodness about .NET Core, what are the downsides?

## The Risks

Making a new version of the .NET framework that works across platforms required some hard work and sacrifices. To break up the monotony of boring screen shots, let me put it like this. If the full .NET framework was a diet plan, you could have doughnuts for breakfast, a deep dish pizza for lunch, and a bacon cheeseburger with egg on top at dinner time. No human can really consume this many calories in a day, but the calories are there for you if you want to try and eat them all. Core .NET is more of a lite meal with all the essential vitamins included. Depending on your appetite, you might not find enough here.

Where am I going with this analogy?

Well, when I say that .net core makes some sacrifices, I mean there was obviously some technology that has been dropped and some features will never appear in .NET Core. There are generally three reasons for a given feature of .NET to not appear in .NET Core. One reason is that there wasn’t enough time to port a feature to .NET core and the team is still working on expanding coverage. A second reason is that a given feature might not make sense for a cross platform library and only makes sense on a windows desktop. And finally, there are some technologies in .NET that have been around for 15 years now and are just never going to move forward. Unfortunately, many of us working in the enterprise space still use frameworks from the original .NET of 2001, so what are the big pieces that are missing and what do you need to know?

Well, for starters, there are entire frameworks that have not made the move to .net and probably never will. These are frameworks like winforms, webforms, wpf, and wcf. From an asp.net persepective we don’t really care about desktop technologies like winfroms and wpf, but there is a tremendous amount of enterprise software written using webforms, there is a tremendous number of services using wcf, and I know for a fact there is still software around using the predecessor to WCF, which was asmx web services. Applications that rely on these frameworks might never move to .net core as they will require a re-design and a re-write. When you think about WCF in particular, a rewrite might not only impact your business but also your partner’s business.

I do want to point out that when it comes to WCF there is no server side WCF replacement. But, if you are writing a server-side application that consumes WCF services hosted in another service, there is an open source version of the WCF client libraries available, and even some beta tooling for visual studio to add a service reference to a .net core application from a wsdl endpoint, so that might be useful for a few projects.

There are also features that are not in .NET Core, at least not currently, some of these may appear in the future, but these are features like the once higely popular dataset class from system.data, there is also no support for working with xml schemas or transformations in .net core. There are no distributed transactions in the system.transactions namespace, no ability to communicate with LDAP or ActiveDirectory, and no class for interacting with an SMTP server. Some of these features, like LDAP and SMTP can be replaced with HTTP based cloud services or third party providers, but this will still require some work at porting application logic.

There are also some features in this new world that still exist but the APIs or behavior are different. For example, the reflection APIs have changed, the encryption APIs have some slight differences,

When moving to .net core you also need your third party dependencies to support .net core. Many popular libraries do already, libraries like structuremap, automapper, mediatr, json.net, these all have working version for .net core. MongoDB is a technology I use frequenely and a .net core version of the C# driver just made a release date last week. Nevertheless, if you rely on third party code you’ll need to rely on the third partieis to make a port to .net core, or find an alternative, or write your own code to replace the dependency.

By the way, one tool you might find useful in analyzing your applications is the.net portability analyzer. You can run this as a visual studio extension or from the command line, and with this tool you can select a target platform, like .net core or asp.net core or the full .net framework, and the tool can tell you what problems you’ll face.

## Summary

To summarize this part of the conversation, I think we are still a few years away from seeing .net core play a big role in the enterprise. We have so much legacy code that relies on webforms and wcf services and there might not be enough of a return on investment to re-write or port these applciations, at least not yet. But, it is clear that Microsoft’s future direction is in the Core space. Yes, the last update to the full .net framework did include some improvements for asp.net and web forms, but clearly the future innovation and hard work will be in the new core frameworks like .net core, asp.net core, entity framework core, and whatever other cores come along in the future. So I do believe it is a good time now that these technologies are released to start some planning, and start prototyping, and if you have requirements for new server applications, I’d push to try new applications using .NET Core first. If you are into micro-service architecture, it wouldn’t hirt to start writeing some of yoru small services in asp.net core, and micro-services could architecture even help an eventual migration to theis new platform.

Also, I’ll just remind you, you can use ASP.NET Core with the full .NET framework, and this setup might solve some problems like problems with third party dependencies that don’t support .NET Core. Yes, it does feel like a temporary solution you might use in transition, but at least this type of approach is a step forward even if the solution might feel messy. In the enterprise, we always have to deal with messy.

## Hosting

Now let’s start talking about some asp.net core specifics and I want to start with hosting, because hosting brings about some major changes in ASP.NET Core.

ASP.NET Core is really just a collection of .nuget packages now, just a few libraries that you can put together to process HTTP requests. One of packages you’ll use is a package named Microsoft.aspnetcore.server.kestrel. The word server is in the name because this version of asp.net includes it’s own web server named kestrel. A kestrel, by the way in the animal kingdom is a bird, a bird of prey in the falcon family. Kestrel in asp.net is a cross platform web server. We’ll talk more about the benefits of kestrel in a moment, but once you find out that asp.net has its own web server you might think wow, that means we can self host everything and use IIS anymore. And while you can use Kestrel without IIS on Windows, the idea Moicrosoft has, particularly in enterprise applications, is that you’ll really use Kestrel as a server behind a hardened proxy and process manager like IIS.

So the hosting scenario looks like this. HTTP requests will arrive at your proxy, which could be IIS on windows, or could be engine-X or apache on other platforms, and your proxy will deliver those requests to your application which is using kestrel. You can have multiple applications behind a single proxy and configure your proxy to use host headers to figure out where to deliver messages. This isn’t much different than what we do with IIS now, expect IIS has a mich larger role with current asp.net applications.

Really, with asp.net core there will be some confusion because some settings in IIS will have no impact. You might think you need to select the version of .net to use in IIS, but in reality, when an asp.net core application is deployed to IIS the asp.net core application will be in it’s own worker process and selecting a version of .net to use regardless of what IIS says. However, there are many settigns in IIS that will apply to asp.net core, for example all the great process manangement features of IIS to recycle applications, throttle CPU and memory, all the process management features that keep an application up and running, we still use those in IIS. Everthing else though, is pretty much all Kestel.

Let me show you what this looks like in VS. **DEMO**

One is performance. On github, in the aspnet/benchmarks reprository, you can find code and documentation for running benchmark tests against asp.net core. It was amaxzing to watch this repository over the past year as the asp.net team made one optimization after another to make kestrel a screaming fast server for .net. in the readme for the repo they publish nubers for different scrnarios, here I pulled out the numbers just for serving a plain text file, a simple job, but you can see asp.net 4.6 could manage 57000 requests per second while asp.net core on kestrel is doing over 300,000 requests per second. This is an insane improvement that makes the old asp.net look very slow, and even makes nodejs, which we often think of as quick and nimble, the numbers make node look pretty slow, too. So, if you want high performance and efficient server utilization you’ll like using Kestrel even though it does change your production environment and will require some new expertise.

The risk here is that we are running a new type of web server named for a bird, and to many IT departments I’m sure this will sound scary. But you do have to remember that this web server is designed to work with a mature host and proxy in front, a mature host and proy like IIS or nginx or aprache, and the performance numbers are better. The net result being that you’ll need to spend some time with development and operations to achieve the same level of logging and transparency in production that you have today.

## Middleware

The next topic is middleware in asp.net core. Previously in asp.net we used http modules and http handlers to make a processing pipeline and service requests, but these abstractions are gone from asp.net core. Instead we use middleware components to process requests. When a requests arrives at an asp.net core application, ultimately kestrel will receive the requests and then send the request through a middleware pipeline that we configure for the application. Each piece of middleware can inspect the request and either pass the request to the next piece of middleware, or decide to process a request and send control back up out of the pipeline. If you are familiar with the katana project in previous releases of asp.net, middleware will look familiar. Each piece of middleware is typically implemented as a class, and Microsoft provides dozens of middleware components with many basic features that we need to build an application. We can also build our own middleware.

DEMO

Risks

One of risks of middleware is that if you have an investment in custom http modules and custom http handlers, you’ll need to port those pieces of code into middleware components. Many of the custom http modules I’ve seen in the enterprise have been there to support custom authentication and authorization schemes, which is our next topic.

Security

When it comes to security in aspnet core there are a few changes. I’ve already mentioned that http modules are not a part of the design of aspnet core. And that means that older frameworks like the asp.net membership providers, they don’t work in this new world of aspnet core.

What’s new is that you’ll need to rely on middleware for any type of user authentication scenarios. If you rely on windows authentication for an intranet application, you’ll still be required to use IIS to authenticate users, and IIS and Kestrel will work together via middlware, to give you a window’s identity. For nearly everything else there is middleware available to process cookie authentication tickets, tokens, integrate with Azure active directory, and facebook, google, twitter – all the big external identity providers.

What’s also changed in this version is there is now a focus on claims. Previosuly in asp.net we had a focus on roles, but claims are more flexible. I can have a claim for myself that my name is scot allen and my date of birth is june 11th. There is also the new concept of an authorization policy. A policy is a piece of code to check information about an identity, and the information is typically about claims. So, I could have an authorization policy in my application that applies to a controller action and says only users with a birthdate in june and a first name of Scott are authrtoizared to use this functionality.

There is also a new version of the asp.net identity framework for aspnet core. This is the fraemwrok to use if you want to allow users to login with forms based authentication and issue the user a cookie when they sign in. There is a implementation of the framework that works with the entity framework to store user information in a database, ut you can also use asp.net identity to allow users to login with facebook and twitter, and when you create a new project in asp.net all this is set up for you, let’s take a look.

DEMO

The identity framework has some shortcomings when it comes to the enterprise and I strongly suggest you look at the identityserver product from thinktecture as soon as you reach the limitations of the frameworks and middlware provided by Microsoft. Identity Server is a free and open source product from Thinktecture, you can find lots of documentation and videos on how to integrate this product into your enterprise. If you want to provide a single-sign on for your users across tyour applications and API, and you want to support users on the web as well as on mobile devices, and you want to build a service that supports today’s protocols like openid connection and oauth 2.0, then identity server gives this all tto you out of the box.

Id’ suggest for the enterprise you start looking at how you will implement autnentication and authorization as one of the first steps in your move to asp.net core. Because in the enterprise we often have messy scenarios involving legacy systems and strange rules, so you’ll need to get your existing logic ported to middleware and authorization policies and make sure everytting works and is secure. Or, you might need to hide some of the messiness behind identity server and allow your applications to use more modern protocols and concepts. Do this earlY!

Data Access

What about data access in this version of aspnet core? Well we still have the classes in system.data nad system.data.sqlserver, so you still have the ability to work with ADO.NET code like we have since the beginning, although there are some changes there and classes like the DataSet are not available at least in the current .net core.

There is also a brand new re-built from the ground up version of the entitiy framework.

…

Summary

To summarize, what we are seeing is a reboot of .NET. There are many familiar pieces still here, but the reboot has given us leaner, more modular frameworks, and I think we have much better abstractions to work with. Take middleware for example, the design of HTTP modules was good at the time but once you start working with middleware you’ll realize just how inflexible http modules were.

So yes, there is indeed some risk with the new frameworks. Not that I don’t think the frameworks will work properly, they are high quality, but in the enterprise we are going to need to unlearn some of what we’ve learned about .net and asp.net in the past and figure out the new guidelines and best practices for the new frameworks and the new abstractions. We are going to have trouble porting many pieces of enterprise code because the code is tied to frameworks like WCF or WebForms that aren’t making the transition to the core world. We can start a new project in .NET core easily enough, but for many applications a move is going to require a re-design, and some re-architecting, and a re-write of existing, working code.

But there is a reward. Moving entirely to .net core gives us the ability to run applications on differnet platforms, gives us the easy ability to containerize our applications which can be a boost for devops, and we’ll be working with small, modular, high performance components that can save money in the long run by reducing the amounts and types of resources that we need.

For me, I believe most enterprises need to start thinking about .NET core in a 5 year plan. The time span is a long, conservative time span, because I think many organizations want to see how the core fraemworks are going to evolve and grow now that we have the frameworks released and people are starting to use the products in real applications. In the 5 year plan you can first start thinking about how you can start implementing new services using aspnet core as I really believe it is going to be much easier for enterprises to start with a greenfield project, gain some expertise on .net core, and then start thinking about how to port and migrate existing applications and services to .net core.