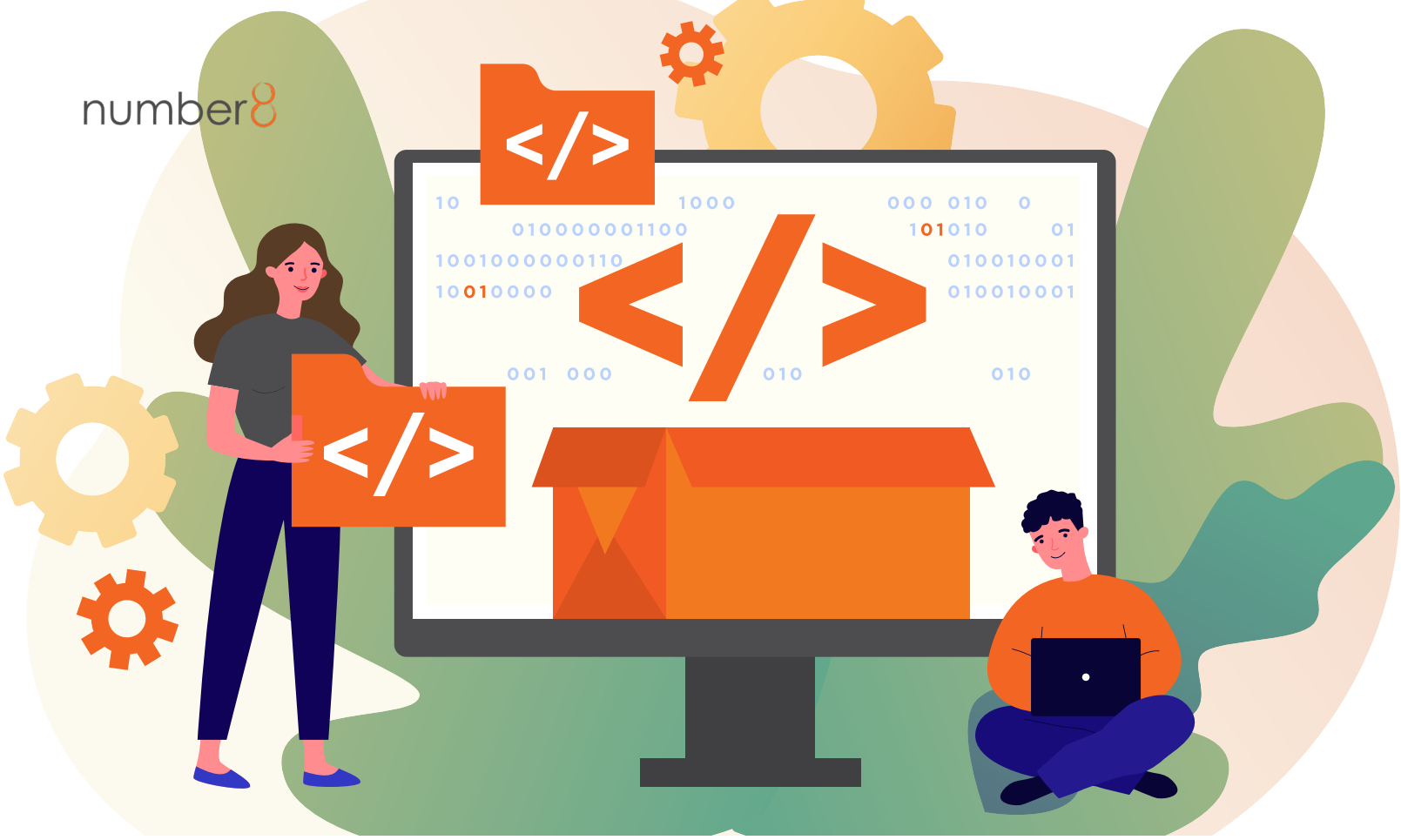
number8



WHY AND HOW TO MIGRATE FROM .NET TO .NET CORE.

- OLIVER RAY, MANAGING DIRECTOR @ NUMBER8

There's no time like the present when it comes to paying down technical debt, investing in high-value features, and keeping up with development house-keeping. These practices allow development velocity to skyrocket. And For most companies still utilizing .NET, migrating to .NET Core is the most impactful technical investment they can make right now.

This transition will simplify your teams' development cycles and have a measurable impact on your bottom-line. However, it can also include unforeseen complexities and dependencies, so it's crucial that the planning and prep work is done throughly.

Our team at number8 took the time to detail a few high-level benefits that every team should consider during this transition. So, let's get started.

WHY MIGRATE FROM .NET TO .NET CORE?

.NET Core 1.0 was announced in 2014 and aimed to bring the .NET framework into the open source world. The ultimate goal of this initiative was to make .NET projects more portable to other platforms. With the explosion of commodity platforms like Linux and ARM64 processors, .NET had to become more open to survive. From it's initial release .NET Core has continued to evolve to leverage a wide range of technologies that allow teams to develop systems in whatever way they see fit.

A significant example of this forward thinking was early support for Docker containers and Kubernetes. Teams everywhere are trying to rethink core enterprise applications to fully see the benefit of cloud tooling. As your data center becomes more and more software defined, it is important to have a framework that can keep up with those demands. .NET Core does. And as new patterns and technologies emerge, .NET Core is continuing to find ways to add additional support.

In the current landscape, with .NET Core 3.1 moving into long term support, version 5.0 is being released at the end of 2020. Now is the time to evaluate, plan, and begin migration efforts. Step one is to start migrating your older applications to .NET Core. You may be aware of the early teams that tried to adopt .NET Core 1.0 and ran into massive problems because their packages did not support .NET Core. However, over the years more tooling and Nuget packages have been added to help in the migration. Now, you would be hard pressed to find a Nuget package that didn't support .NET Core or .NET Standard.

START PLANNING NOW

.NET will cease to be functional in the foreseeable future, making this decision less of a choice and more of a necessity. It's possible that your team is already behind when it comes to developing a realistic plan for the movement of your applications. Richard Lander, Microsoft's Principal Program Manager for .NET, recently announced .NET Core 3.0 closed many of the remaining capability gaps with .NET framework 4.8. Learn more about the transition from Ricard here.

WHY YOU SHOULD CARE ABOUT .NET STANDARD

.NET Standard is a compatible way to share code across all .NET implementations. This allows for maximum flexibility and minimizes code reuse. Your team will have the ability to share code across .NET Core, .NET, Mono, Xamarin, UWP, and Unity. Regardless of the implementation, it will work. The motivation behind .NET Standard is to establish greater uniformity in the .NET ecosystem. This continued emphasis on flexibility makes the adoption of .NET Core a no-brainer.

In the event you need any more convincing, the .NET Standard 2.0 introduced .NET Framework compatibility mode. This compatibility mode allows .NET Standard projects to reference .NET Framework libraries as if they were compiled for .NET Standard. Referencing .NET Framework libraries doesn't work for all projects, such as libraries that use Windows Presentation Foundation (WPF) APIs.

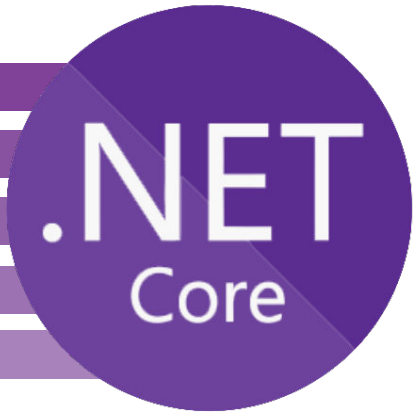
If your enterprise uses any internal Nuget packages, migrate those first to .NET Standard 2.0. Then, you can get up and running with .NET Core with relative ease. Thankfully Microsoft made a chart to help you find the available compatibility. Available Here.

WHAT'S THE POINT IF EVERYTHING IS RUNNING ON LEGACY INFRASTRUCTURE?

A common question we get is, "If I have an on-prem data center not running Linux, why migrate?" The answer is pretty simple - **Why Risk It?**

The .NET Framework only receives security patches that are part of the Windows Server Lifecycle it was shipped with. As those lifecycles quickly approach their end of support, you're left with a massive hole in your security policies. However, there is upside in making the switch. Besides getting the latest features of .NET Core, you also get the tooling surrounding it, which means:

- Easier to collect diagnostic dumps with 3.1
- More internal data exposed by the CLR and GC
- Better ASYNC exception messages
- Ahead of time compiling for faster startup times
- Self contained deployments



HOW DOES THIS IMPACT THE BOTTOMLINE?

MORE THAN JUST TOOLING

Since the CLR is now Open Source, developers are optimizing .NET Core to reduce memory usage and get a higher benchmark ranking in Tech Empower. .NET Core is now achieving over 6 million requests per second in the Plaintext benchmark on a single Dell server. Bing reported a 34% improvement in latency by migration to .NET Core 2.1. Overall, lower memory usage means lower server costs and everyone loves that.

FLEXIBILITY FOR THE FUTURE

.NET Core also has the ability to run across platforms and in a wide range of environments. If you're doing any of the following, Microsoft suggests .NET Core:

- Targeting microservices
- Using Docker containers
- Have a need for high-performance and scalable systems
- Have a need for side-by-side .NET versions per application

The somewhat hidden beauty of .NET Core is the fact that it can run in either a container or serverless environment. The flexibility is almost endless. .NET Core can run in a Docker Container or Kubernetes. Also, .NET Core works in AWS Lambda environments with the latest 3.1 release being fully supported by AWS.

Overall, .NET Core is a pretty strong hedge against the future. Microsoft has put a ton of resources into the development of .NET Core and it has ever increasing popularity among developers. In fact, it was one of the most 'loved' frameworks in the 2019 Stackoverflow Developer Survey. Net Core offers you the ability to adapt with the future. Why not try to capture that future today?