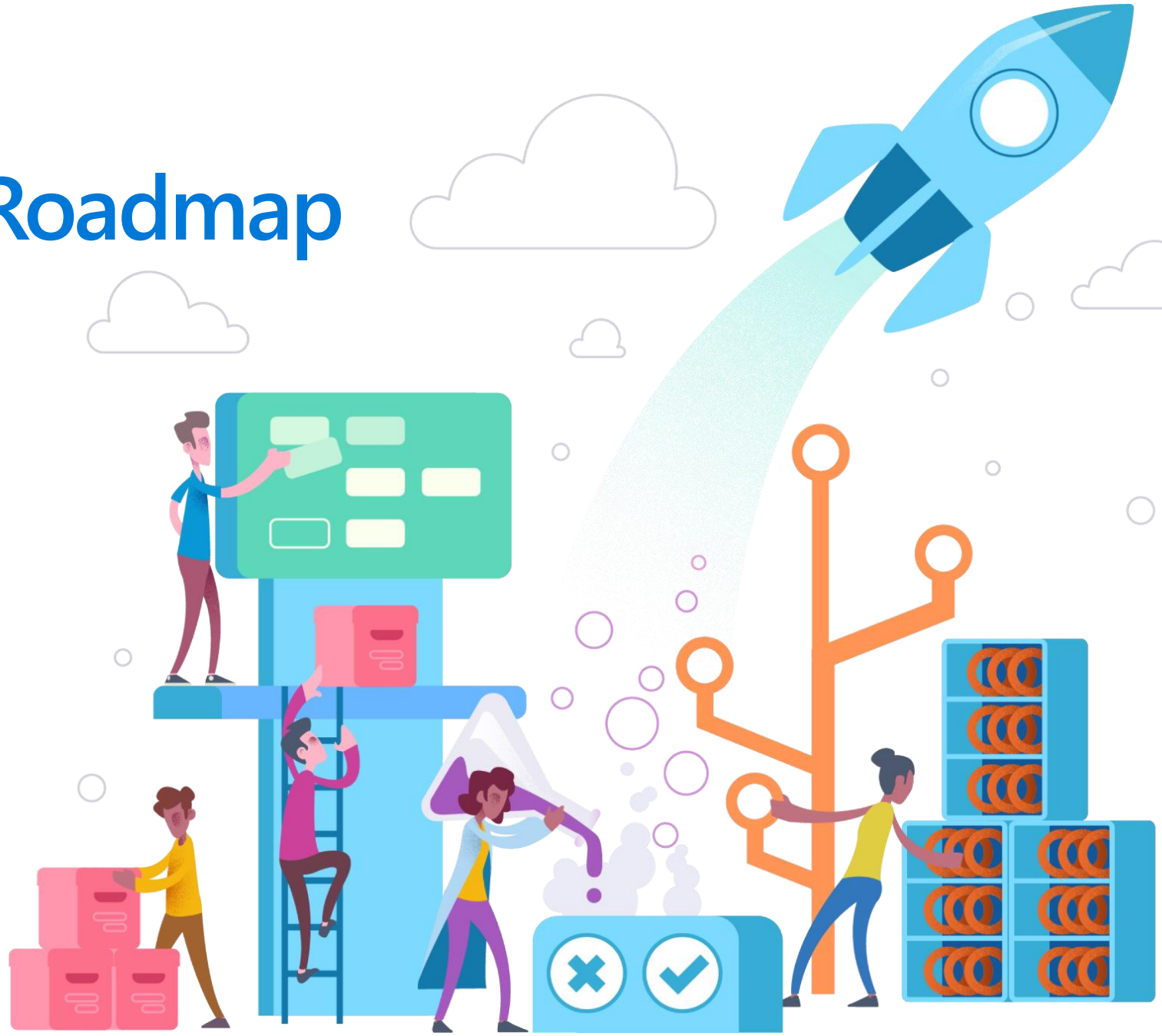
Microsoft

.NET Overview & Roadmap

.NET Framework & .NET Core



Michael Pedersen
TSP – Azure App Dev

01-11-2018

Agenda

Introduction

.NET Overview

Deployment & ASP .NET Core

Migration & Compatibility

SignalR in .NET Core & SignalR Service

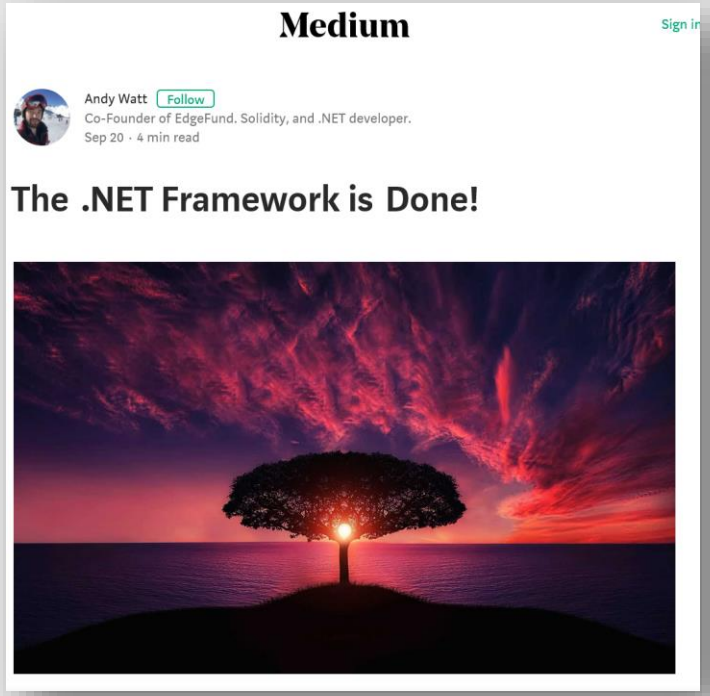
Blazor

Several demos!

Why are we having this meeting

Background:

https://medium.com/@andy.watt83/the-net-framework-is-done-8aec3bbae12d



"The .NET Framework is on it's last release—there will not be another one after 4.8"

The sun is setting on .NET Framework. From now on, .NET Core is king.

~~It is all but confirmed that .NET Framework v4.8 will be the final release, and that all subsequent innovation will be happening on .NET Core. It is time for us all to make the switch.~~

I fully expect an announcement from Microsoft in the coming months to confirm that Framework v4.8 will be the final release, and that from that point onward, the bulk of the development work will be on Core. Meaning

Update on .NET Core vs .NET Framework

Scott Hunter - 2018

https://blogs.msdn.microsoft.com/dotnet/2018/10/04/update-on-net-core-3-0-and-net-framework-4-8/

https://www.youtube.com/watch?time_continue=2003&v=zj495oS5dHc (Jump to 33 minutes in)

> .NET Framework is the implementation of .NET that's installed on over one billion machines and thus needs to remain as compatible as possible. Because of this, it moves at a slower pace than .NET Core. I mentioned above that even security and bug fixes can cause breaks in applications because applications depend on the previous behavior. We will make sure that .NET Framework always supports the latest networking protocols, security standards, and Windows features.

> If you have existing .NET Framework applications, you should not feel pressured to move to .NET Core. Both .NET Framework and .NET Core will move forward, and both will be fully supported, .NET Framework will always be a part of Windows. But moving forward they will contain somewhat different features. Even inside of Microsoft we have many large product lines that are based on .NET Framework and will remain on .NET Framework.

The Language Strategy

https://blogs.msdn.microsoft.com/dotnet/2017/02/01/the-net-language-strategy/

https://blogs.msdn.microsoft.com/vbteam/2017/02/01/digging-deeper-into-the-visual-basic-language-strategy/

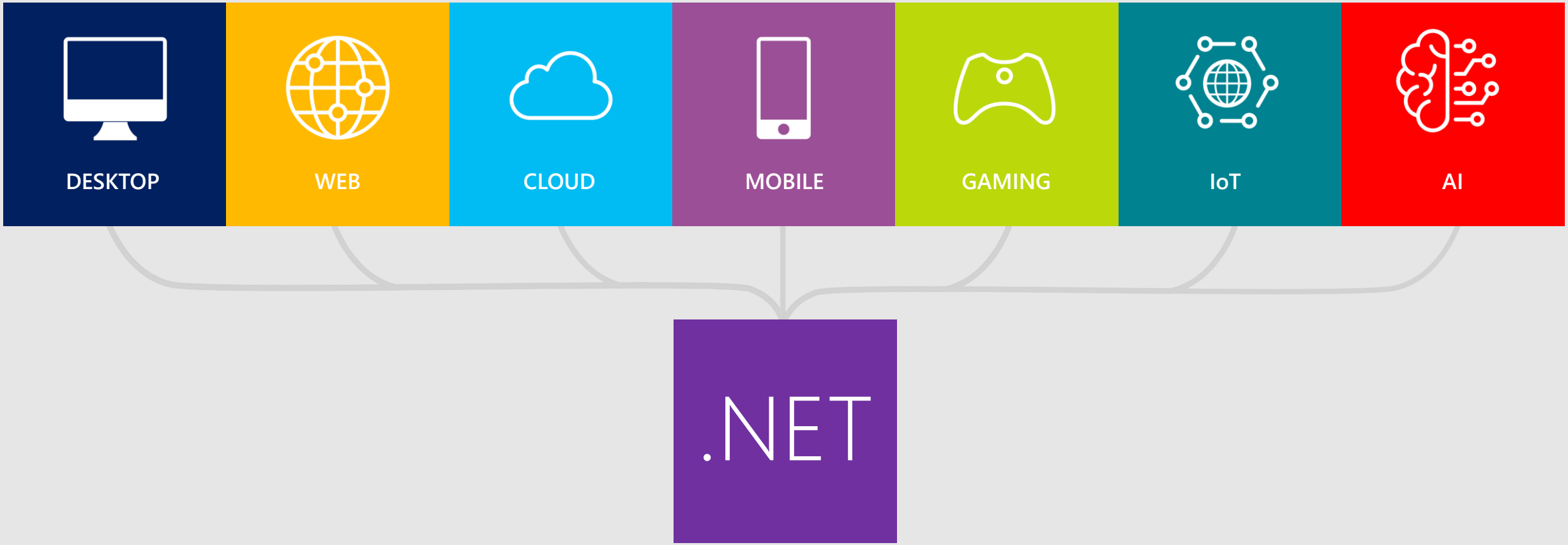
C#

"We will keep growing C# to meet the evolving needs of developers and remain a state of the art programming language. We will innovate aggressively, while being very careful to stay within the spirit of the language. [...] We will continue to empower the broader ecosystem and grow its role in C#'s future, while maintaining strong stewardship of design decisions to ensure continued coherence."

VB

"We will keep Visual Basic straightforward and approachable. We will do everything necessary to keep it a first class citizen of the .NET ecosystem: When API shapes evolve as a result of new C# features, for instance, consuming those APIs should feel natural in VB. We will keep a focus on the cross-language tooling experience, recognizing that many VB developers also use C#. ***We will focus innovation on the core scenarios and domains where VB is popular.***"

Your platform for building **anything**



Why .NET Core?

Crossplatform

Side by side

Simplicity

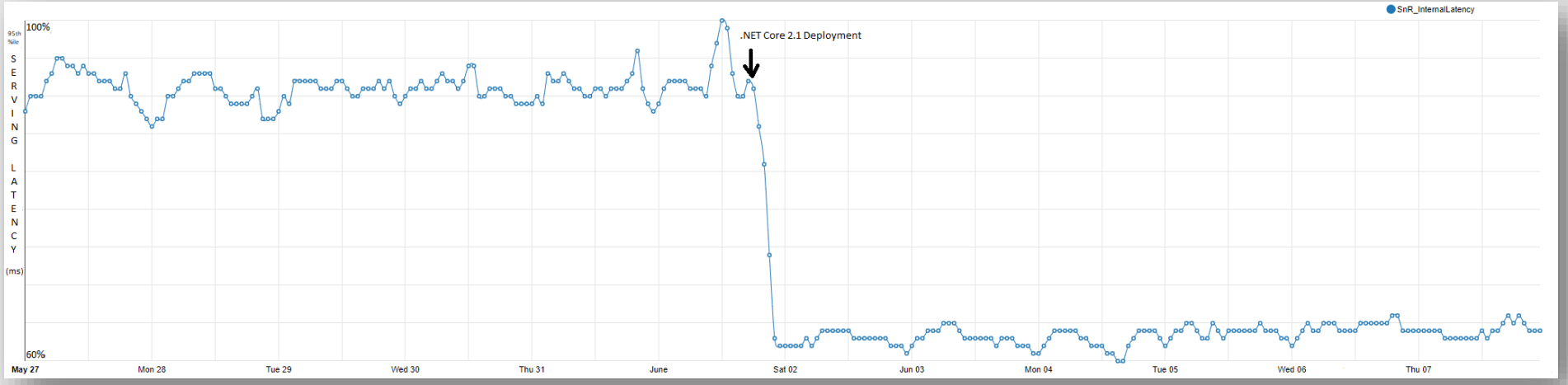
Performance

Easy Built-in: DI, config providers, logging providers etc…

Web & Back-end for now…

https://blogs.msdn.microsoft.com/dotnet/2018/08/20/bing-com-runs-on-net-core-2-1/

https://twitter.com/Nick_Craver/status/1031858480888639488



.NET Standard

.NET Standard is for sharing code!

Defines a set of APIs that all .NET implementations must provide to conform to the standard. This unifies the .NET implementations and prevents future fragmentation.

The following table lists the minimum platform versions that support each .NET Standard version.

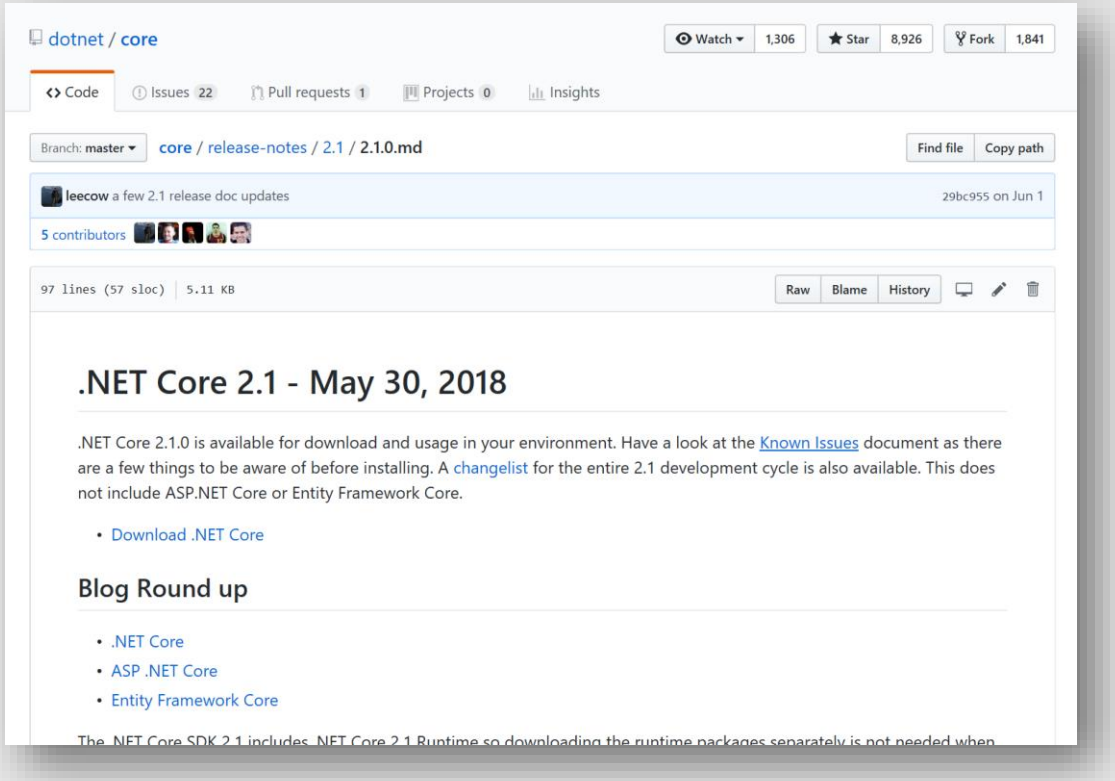
.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0
.NET Framework [1]	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1	4.6.1	4.6.1
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0
Universal Windows Platform	10.0	10.0	10.0	10.0	10.0	10.0.16299	10.0.16299	10.0.16299
Windows	8.0	8.0	8.1					
Windows Phone	8.1	8.1	8.1					
Windows Phone Silverlight	8.0							
Unity	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1

.NET Core Versions

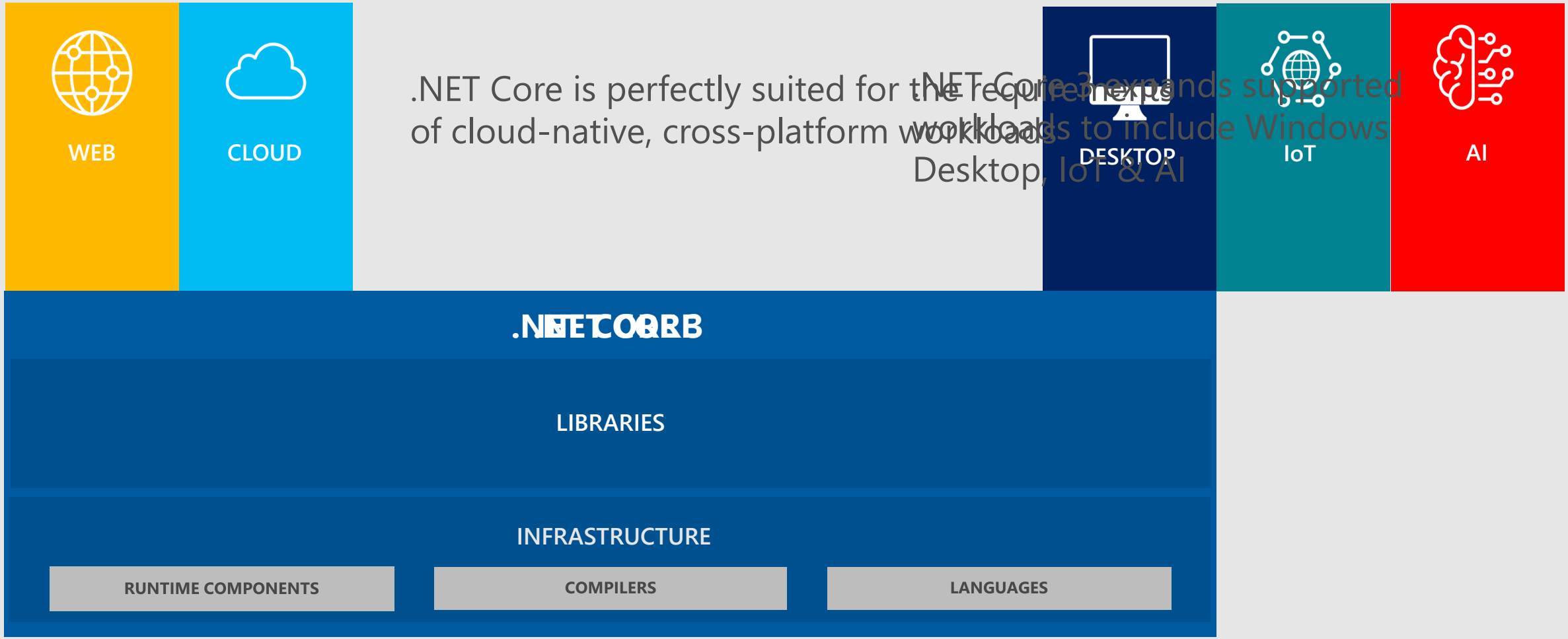
1.0, 1.1, 2.0, 2.1 and 2.2 (preview). 3.0 Announced

Release details:

https://github.com/dotnet/core/tree/master/release-notes



.NET Core 3


WEB
CLOUD
.NET Core is perfectly suited for the requirements of cloud-native, cross-platform workloads
.NET Core 3 expands supported workloads to include Windows Desktop, IoT & AI
DESKTOP
IoT
AI
.NET CORE
LIBRARIES
INFRASTRUCTURE
RUNTIME COMPONENTS
COMPILERS
LANGUAGES

.NET Core 3 Desktop Improvements

- .NET Core 3 supports WinForms and WPF frameworks
  - XAML Islands - WinForms & WPF can host UWP
  - XAML Controls – WinForms & WPF browser and media UWP controls (Edge)
  - Access to all the Windows 10 API's
  - Side by side - Machine global or app local framework
- NET Core App Bundler
  - Precompiled, fast startup
  - Small apps by removing unused dependencies, link away unused IL
  - Single self-contained .exe
- WinForms and WPF in internal builds
  - Visual Studio supports building and debugging now
  - Designers not available yet
- Public Preview later this year

Microsoft Support for .NET Core

Long Term Support (LTS) releases are supported for the following timeframe, whichever is longer:

- Three years after initial release.
- One year after a subsequent **LTS** release.

Current releases include new features that may undergo future change based on feedback:

- Three months after a subsequent Current or LTS release

Both types of releases receive critical fixes throughout their lifecycle, for security, reliability, or to add support for new operating system versions. You must stay up-to-date with the latest patches to qualify for support.

https://github.com/dotnet/core/blob/master/microsoft-support.md

Getting .NET Core

.NET Download Site

https://www.microsoft.com/net/download

Visual Studio SDKs

https://www.microsoft.com/net/download/visual-studio-sdks

API Browser

https://docs.microsoft.com/en-us/dotnet/api

dotnet CLI

.NET Core command-line interface (CLI) is a new cross-platform toolchain for developing .NET applications

Build IDEs, editors, and build orchestrators etc. on top

Extensible

```
dotnet run
dotnet restore
dotnet build
dotnet clean
dotnet test
dotnet publish
```

…

https://docs.microsoft.com/en-us/dotnet/core/tools/?tabs=netcore2x

dotnet CLI Templates

dotnet new console –n console-demo-app

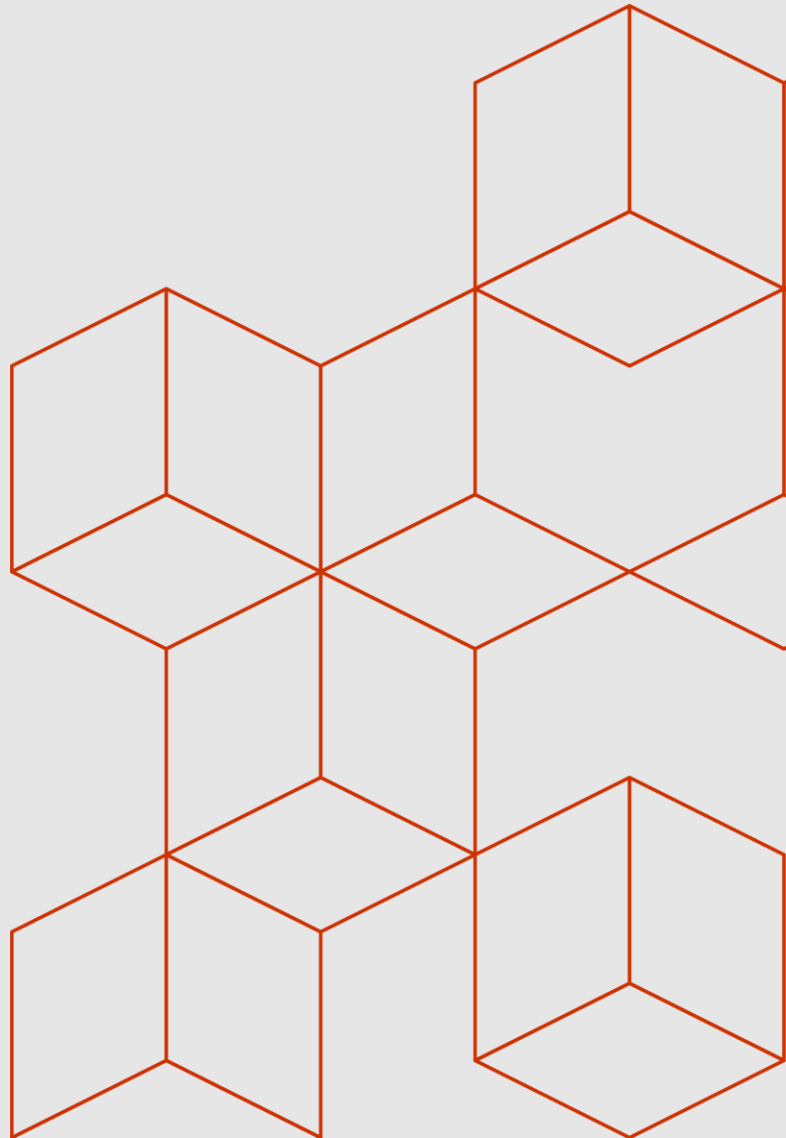
dotnet new -l

```
c:\temp\console1
  dotnet new console -n console1

Welcome to .NET Core!
---------------------
Learn more about .NET Core: https://aka.ms/dotnet-docs
Use 'dotnet --help' to see available commands or visit: https://aka.ms/dotnet-cli-docs

Telemetry
---------
The .NET Core tools collect usage data in order to help us improve your experience. The data
 arguments. The data is collected by Microsoft and shared with the community. You can opt-out
ETRY_OPTOUT environment variable to '1' or 'true' using your favorite shell.

Read more about .NET Core CLI Tools telemetry: https://aka.ms/dotnet-cli-telemetry

ASP.NET Core
------------
Successfully installed the ASP.NET Core HTTPS Development Certificate.
To trust the certificate run 'dotnet dev-certs https --trust' (Windows and macOS only). For e
 the platform specific documentation.
For more information on configuring HTTPS see https://go.microsoft.com/fwlink/?linkid=848054.
Getting ready...
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on console1\console1.csproj...
  Restoring packages for c:\temp\console1\console1\console1.csproj...
  Generating MSBuild file c:\temp\console1\console1\obj\console1.csproj.nuget.g.props.
  Generating MSBuild file c:\temp\console1\console1\obj\console1.csproj.nuget.g.targets.
  Restore completed in 412,87 ms for c:\temp\console1\console1\console1.csproj.

Restore succeeded.
```

```
01-11-2018  08:32    <DIR>          .
01-11-2018  08:32    <DIR>          ..
01-11-2018  08:32               178 console1.csproj
01-11-2018  08:32    <DIR>          obj
01-11-2018  08:32               190 Program.cs
```

https://dotnetnew.azurewebsites.net/

Demo #1: .NET Core



Framework-dependent deployments (FDD)

Deploy only your app and third-party dependencies

App will use the version of .NET Core that's present on the target system

Default deployment model for .NET Core and ASP.NET Core

Why create a framework-dependent deployment?

- You don't have to define the target operating in advance
- The size of your deployment package is small
- Multiple apps use the same .NET Core installation, which reduces both disk space and memory usage on host systems

Disadvantages:

- Your app can run only if the version of .NET Core that you target, or a later version, is already installed on the host system
- It's possible for the .NET Core runtime and libraries to change without your knowledge in future releases. In rare cases, this may change the behavior of your app

Self-contained deployments (SCD)

For a self-contained deployment, you deploy your app and any required third-party dependencies along with the version of .NET Core that you used to build the app

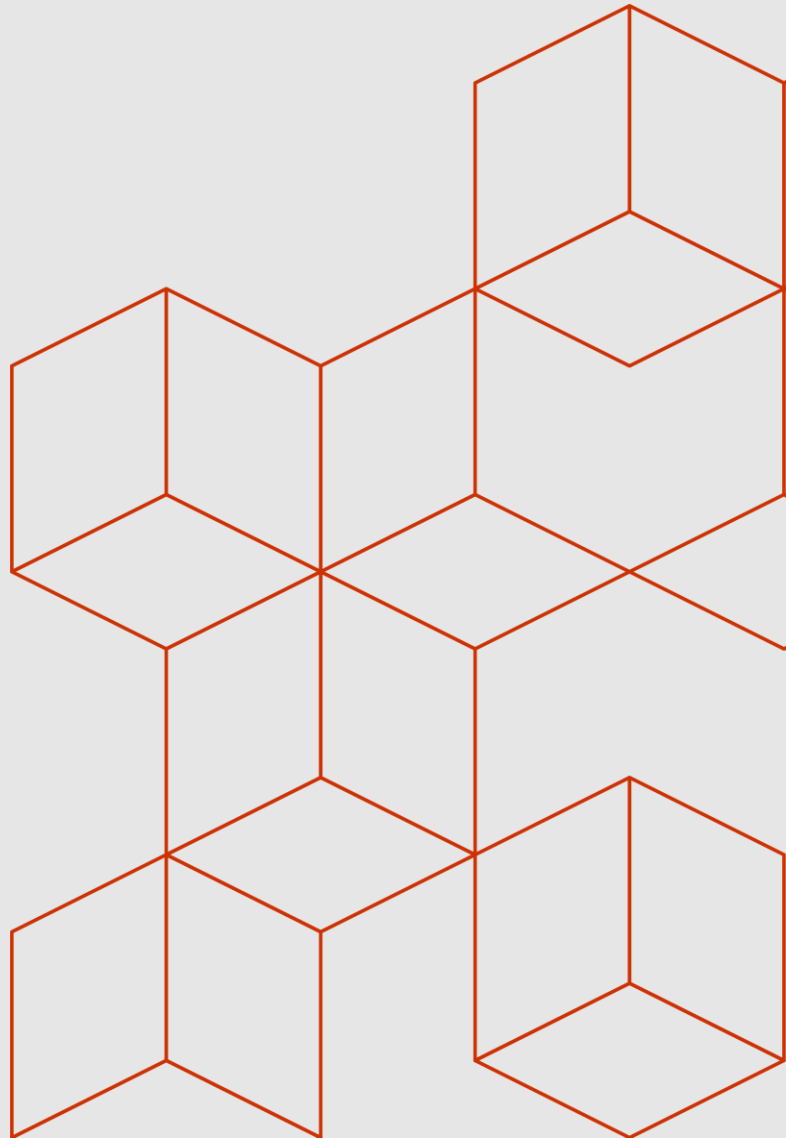
Two major advantages:

- You have sole control of the version of .NET Core that is deployed with your ap
- You can be assured that the target system can run your .NET Core app

It also has a number of disadvantages:

- You must select the target platforms for which you build deployment packages in advance
- The size of your deployment package is relatively large
- Deploying numerous self-contained .NET Core apps to a system can consume significant amounts of disk space, since each app duplicates .NET Core files
- Remember the GAC… ☺

Demo #2: Self-contained deployments



ASP .NET Core

ASP.NET Core is a redesign of ASP.NET 4.x

Architectural changes for a leaner, more modular framework

Integrates seamlessly with e.g. Angular, React, and Bootstrap

Has all the properties of .NET Core plus e.g.

- Lightweight, high-performance, and modular HTTP request pipeline
- Ability to host on IIS, Nginx, Apache, Docker, or self-host in your own process
- Easy to work with HTTPS locally

```
dotnet dev-certs https --trust
```

- Uses Razor

```
@page
@model AboutModel
@{
    ViewData["Title"] = "About";
}
<h2>@ViewData["Title"]</h2>
<h3>@Model.Message</h3>

<p>Hello, world! The time on the server is @DateTime.Now</p>
```

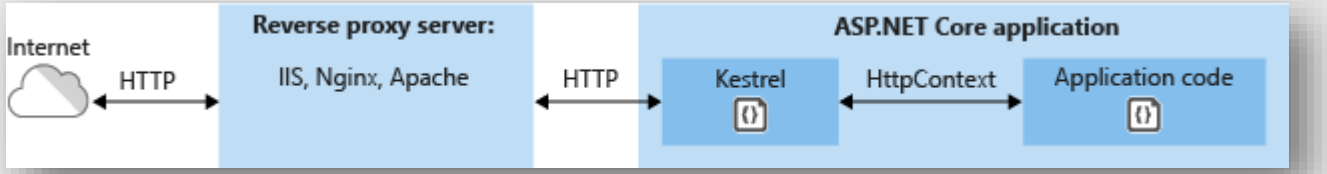
https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1

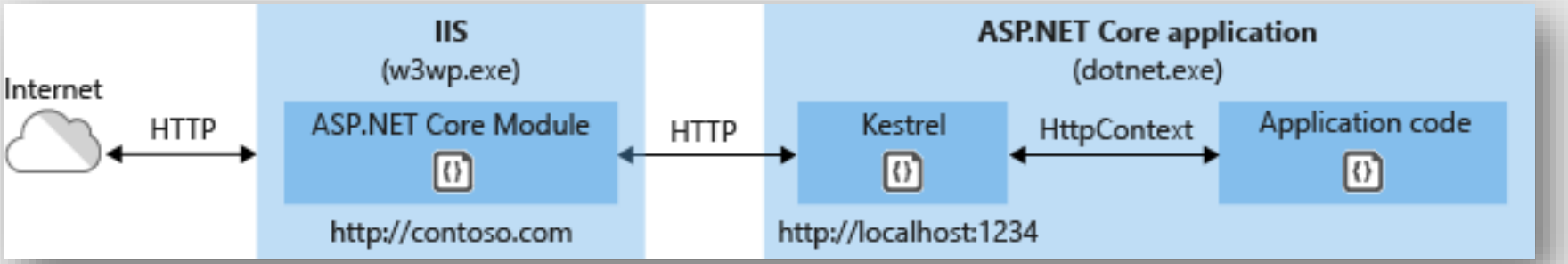
Hosting ASP .NET Core

ASP.NET Core ships three server implementations:

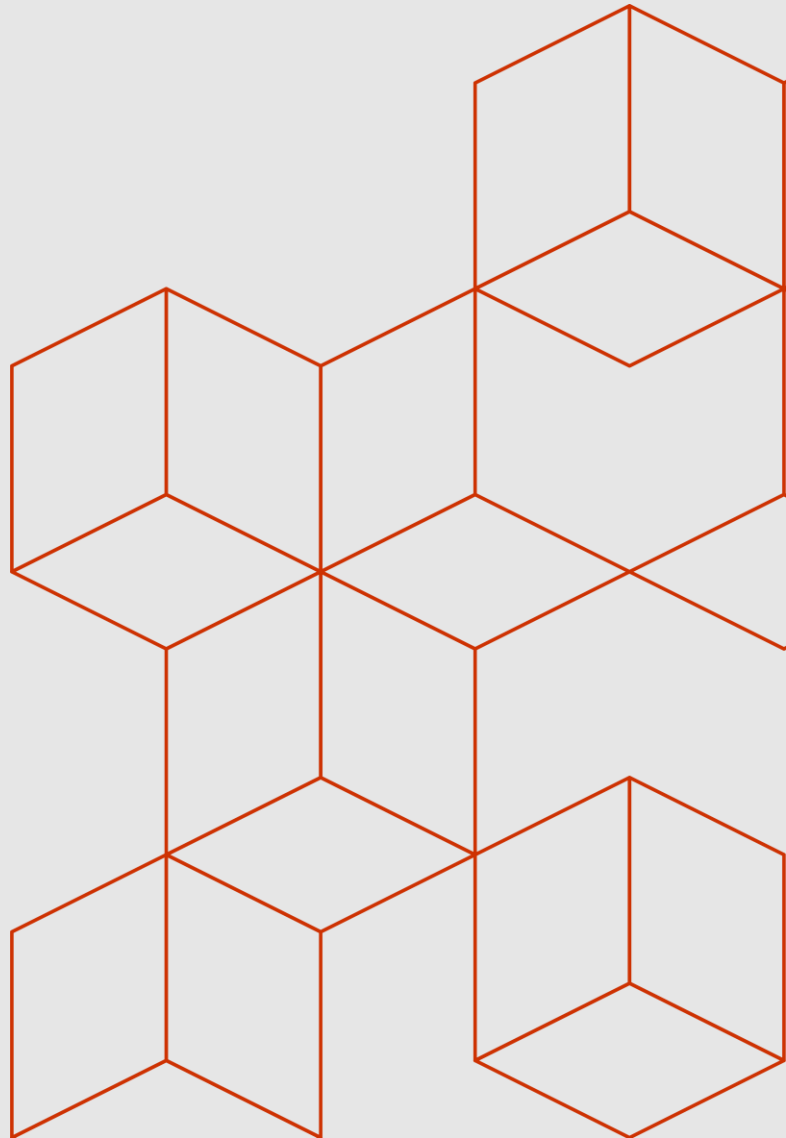
- Kestrel is the default, cross-platform HTTP server for ASP.NET Core
- HTTP.sys is a Windows-only HTTP server based on the HTTP.sys kernel driver + HTTP Server API
- Custom server implementation

IIS, Nginx, and Apache can't be used without Kestrel





Demo #3: Docker Container on Linux with .NET Core ASP.NET



Porting to .NET Core from .NET Framework

1. Identify and account for your third-party dependencies
2. Retarget all projects you wish to port to target .NET Framework of the appropriate version (see .NET Standard)
3. Use the .NET Portability Analyzer to analyze your assemblies and develop a plan to port based on its results
4. Port your tests code.
5. Execute your plan for porting!

Windows Compatibility Pack for .NET Core

20,000 APIs via a single NuGet package

https://www.nuget.org/packages/Microsoft.Windows.Compatibility

This package is meant for developers that need to port existing .NET Framework code to .NET Core.

Some are Windows only e.g. Microsoft.Win32.Registry. You have three options if you want to go cross platform: Remove, Replace or Guard

Just porting to .NET Core because it's a new .NET implementation isn't a good reason!

https://docs.microsoft.com/en-us/dotnet/core/porting/windows-compat-pack

Portability Analyzer

Creates a detailed report on how flexible your program is across .NET implementations by analyzing assemblies

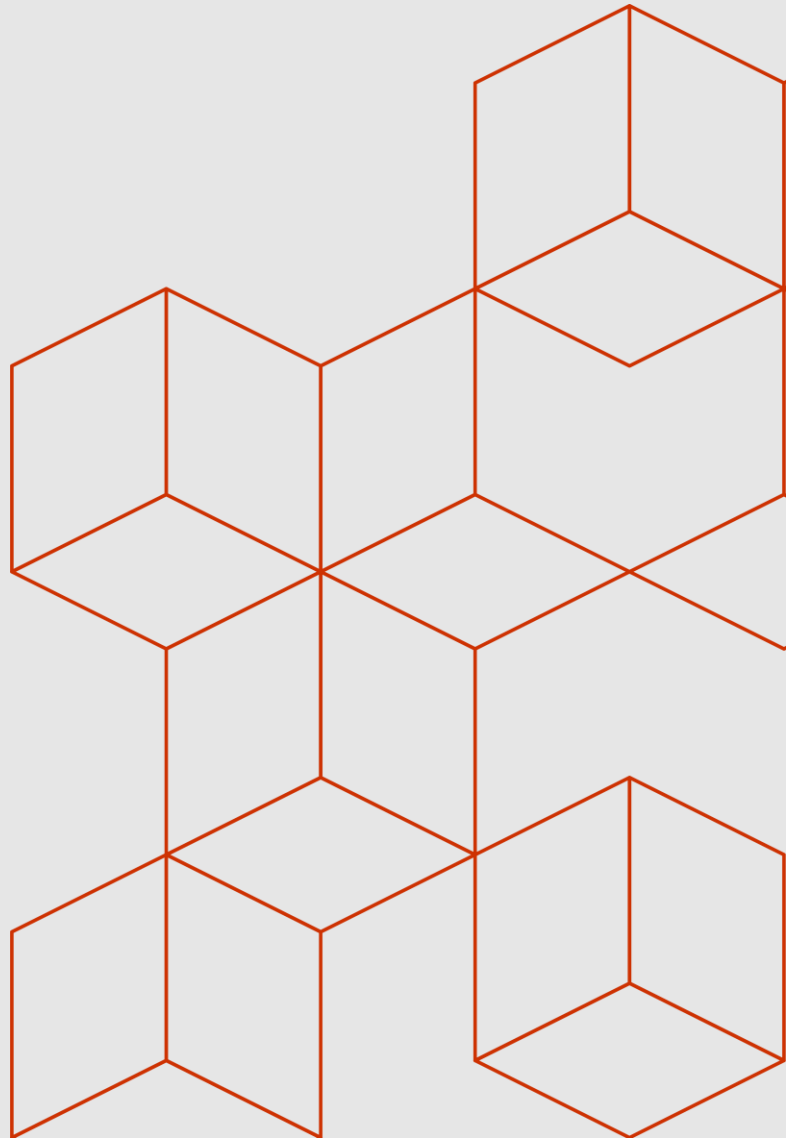
Comes as a Visual Studio Extension, console app and Windows GUI App

Generation of multiple analysis reports in different formats: JSON, HTML or Excel

https://docs.microsoft.com/en-us/dotnet/standard/analyzers/portability-analyzer

https://blogs.msdn.microsoft.com/dotnet/2018/08/08/are-your-windows-forms-and-wpf-applications-ready-for-net-core-3-0/

Demo #4: Portability Analyzer



Announcing Azure SignalR Service GA!

- Add real-time web functionalities easily with Azure and .NET Core
- Enable via “services.AddSignalR().AddAzureSignalR()”

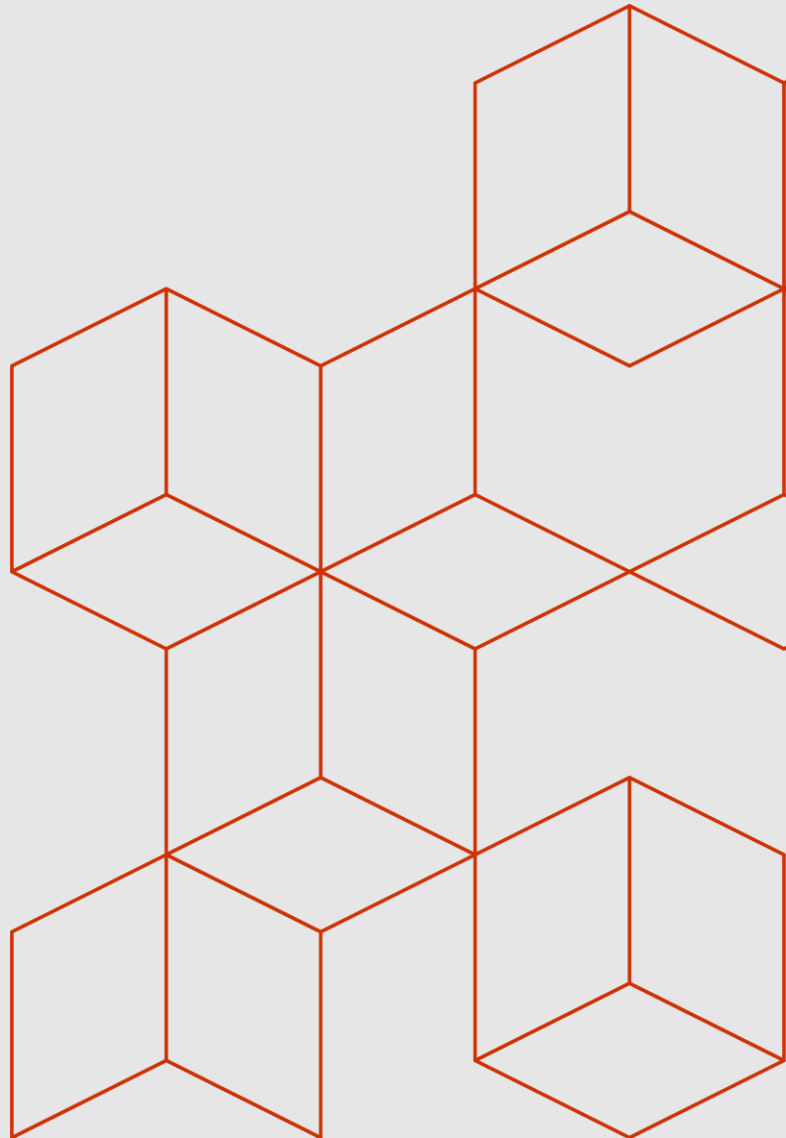
Fully managed service

No more worries about capacity provisioning, scaling, or persistent connections

Native SignalR development

Use ASP.NET Core SignalR to build real-time experiences such as chat, stock tickers, live dashboards, and instant broadcasting

Demo #5: SignalR on .NET Core



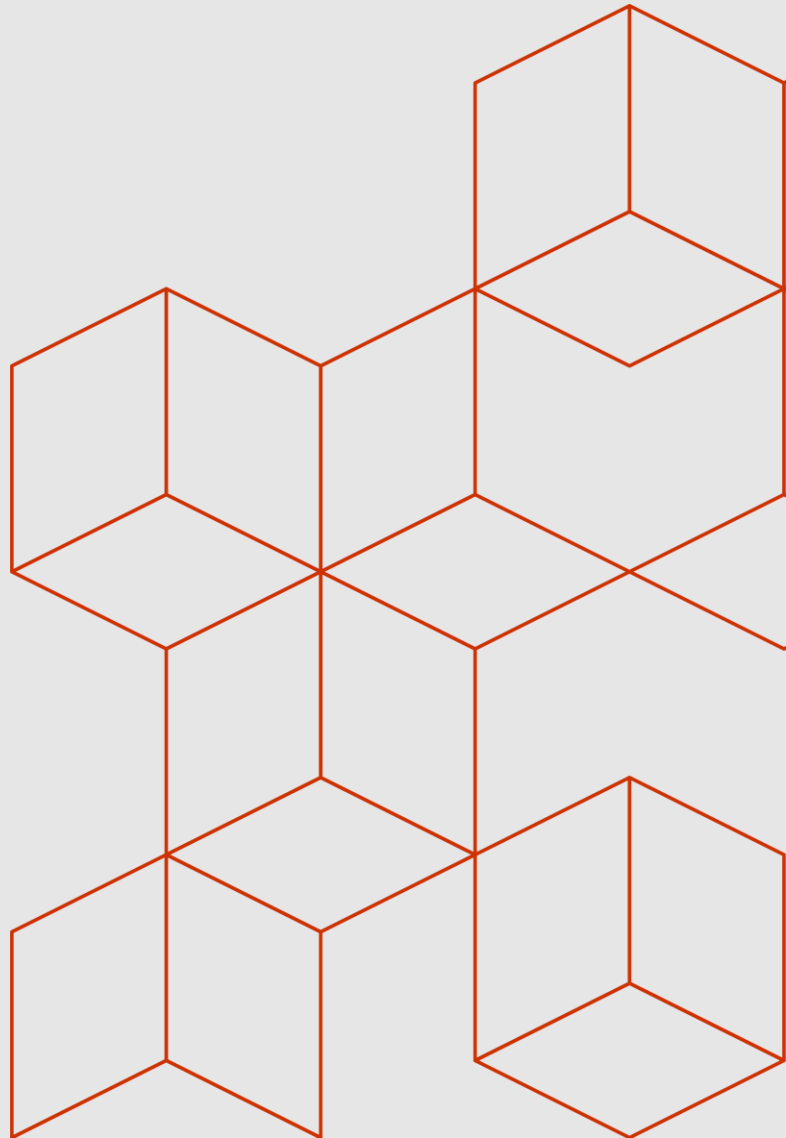
Blazor: .NET in the Browser (Experimental)

- Build client-side Web UI in .NET
  - You don’t need to know AngularJS, React, Vue, Knockoutetc
  - Take advantage of stability and consistency of .NET
  - Uses Razor syntax
- Runs in all browsers
  - Native performance
  - Strongly typed on client and server
  - Requires no plugin or code transpilation
  - Share C# code with the client and the server
  - Runs client-side on WebAssembly and fallback to JS

https://blazor.net/

Demo #6: Blazor

https://blazor-demo.github.io/



Microsoft

Questions?

Code & Slides
https://github.com/mpeder/netcoredemos