

Asp.Net Core 3

Wael Kdoh - @waelkdoh

Senior Consultant



Module 1: Overview

Module Overview

Module 1: Overview

Section 1: Modern Web

Lesson: ASP.NET and Modern
Web

What Is Modern Web?

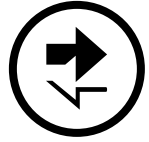
Modern Web

- Web Frameworks
 - Mobile / Tablet First
 - Responsive Design
 - Client Frameworks
 - Cloud Ready
- Web Tooling
 - Standards Based
 - Tooling in Browser
 - Adopting Popular third-party Tools

ASP.NET Core for the Modern Web



Totally Modular



Faster Development Cycle



Seamless transition from on-premises to cloud



Fast



Choose your Editors and Tools

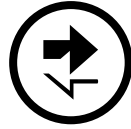



[Open Source with Contributions](#)



Cross-platform

ASP.NET Core - Agility

-  **Faster Development Cycle**
 - Features are shipped as packages
 - Framework ships as part of the application
-  **More Control**
 - Zero day security bugs patched by Microsoft
 - Same code runs in development and production
 - Developer opts to new versions, allowing breaking changes

ASP.NET Core - Fast



Development productivity and low friction

- Edit code and refresh browser
- Flexibility of dynamic environment with the power of .NET Framework
- Develop with Visual Studio, third-party and cloud editors



Runtime Performance

- Faster startup times
- Lower memory / higher density (more than 90% reduction)
- Modular, opt into just features needed
- Use a raw socket, framework or both

ASP.NET Core - Cloud

 Seamless transition from on-premises to Cloud and Cloud Ready

 Cloud Ready

- Configuration, Session and Cache

 Diagnostics

- Run/Debug in Cloud
- Tracing/Logging without re-deploying

ASP.NET Core – Cross Platform

 Open Source with Contributions

 Runtime

- Windows, Mac, Linux (Debian, Ubuntu, CentOS, Fedora, and derivatives)

 Editors

- Visual Studio, Text, and Cloud editors
- No editors (command-line)

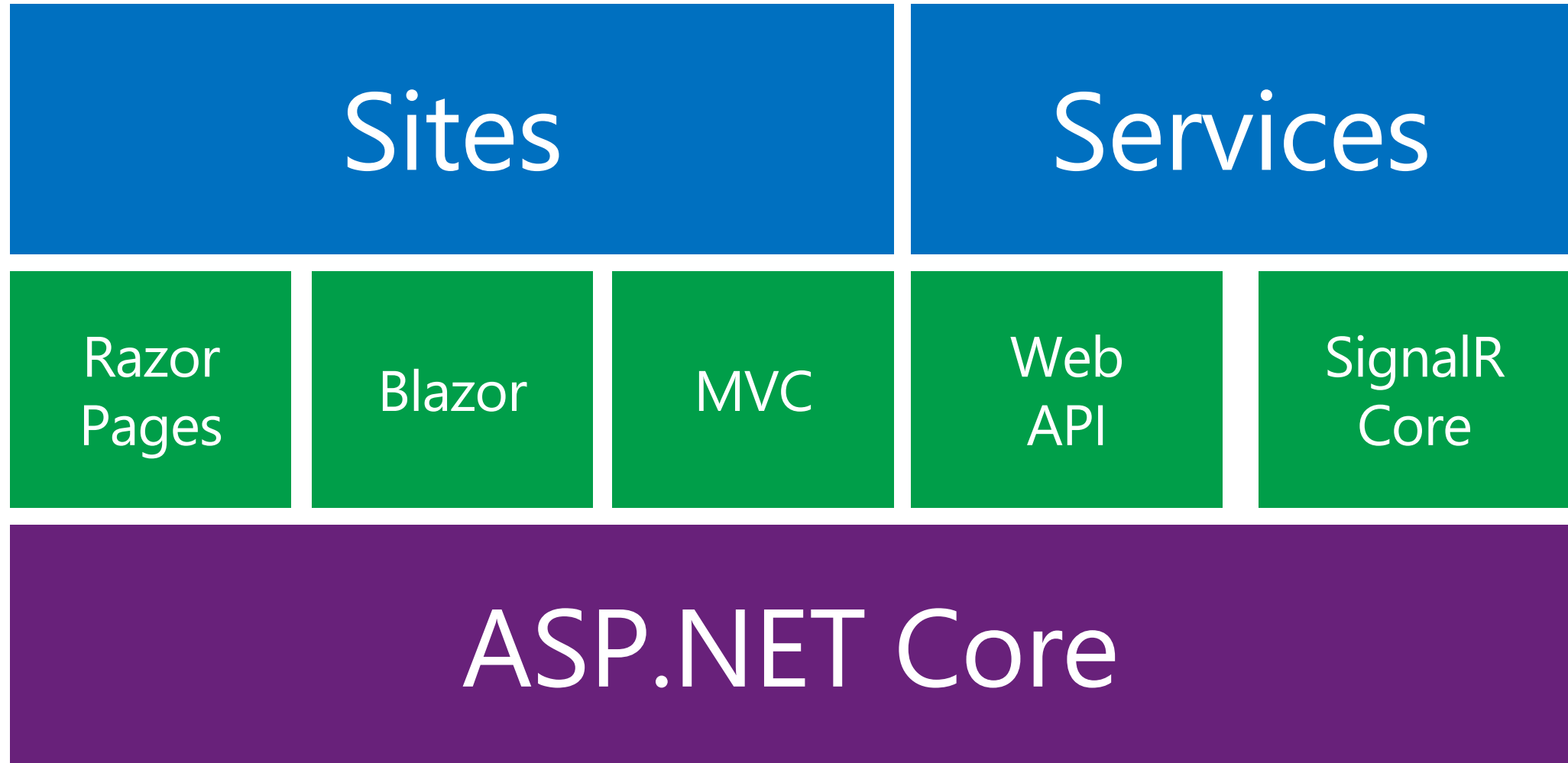
Demo: ASP.NET Core and Visual Studio 2019

Module 1: Overview

Section 1: Modern Web

Lesson: One ASP.NET

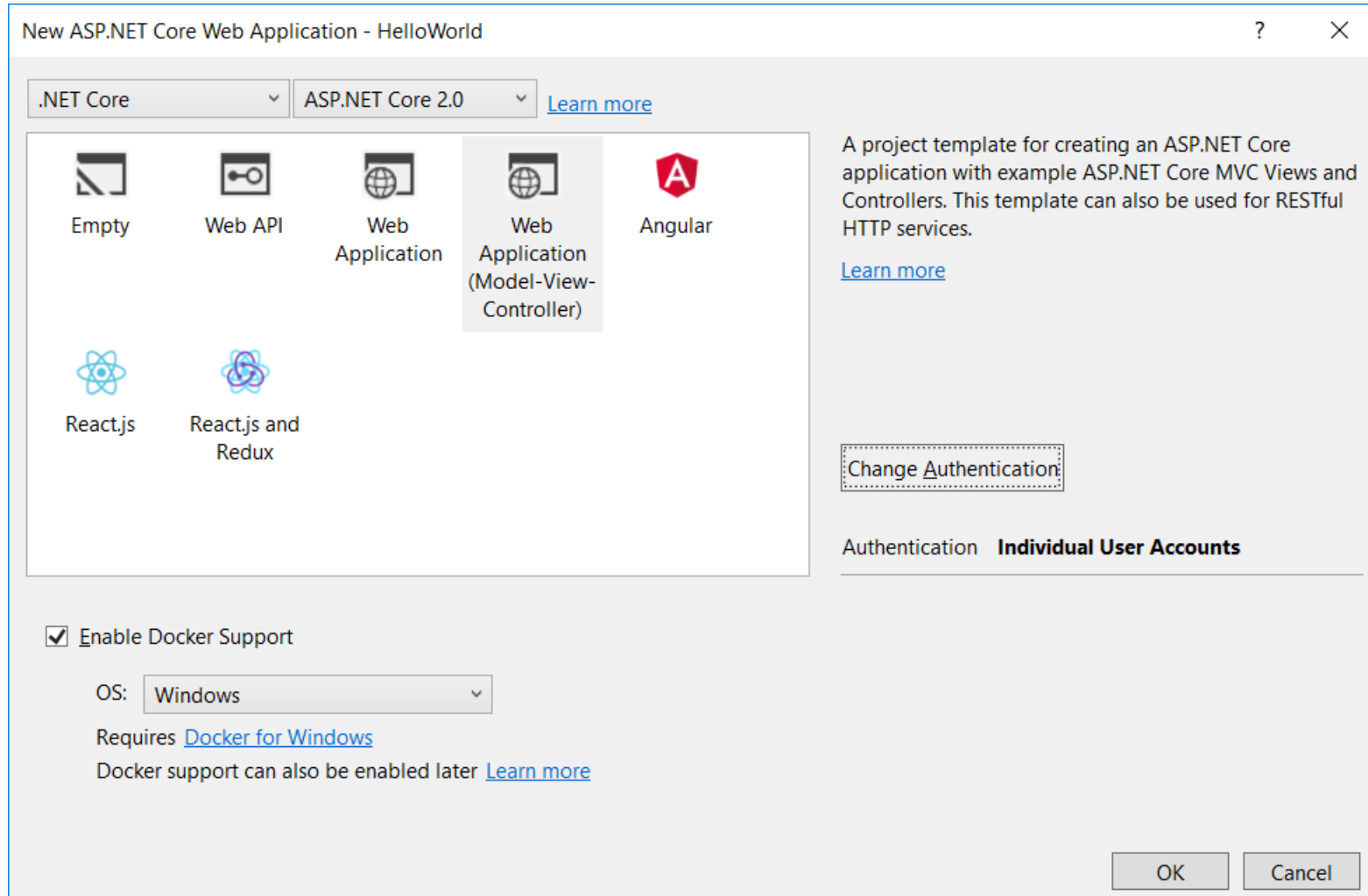
ASP.NET Core



Commonalities

- All programming models have the same Microsoft ASP.NET
 - Authentication/Authorization/Membership
 - Output Caching, Session State, and Configuration
 - AJAX, Deployment, etc.
- All programming models are fully supported and will continue to be supported
- All programming models solve real problems

ASP.NET Core Project System



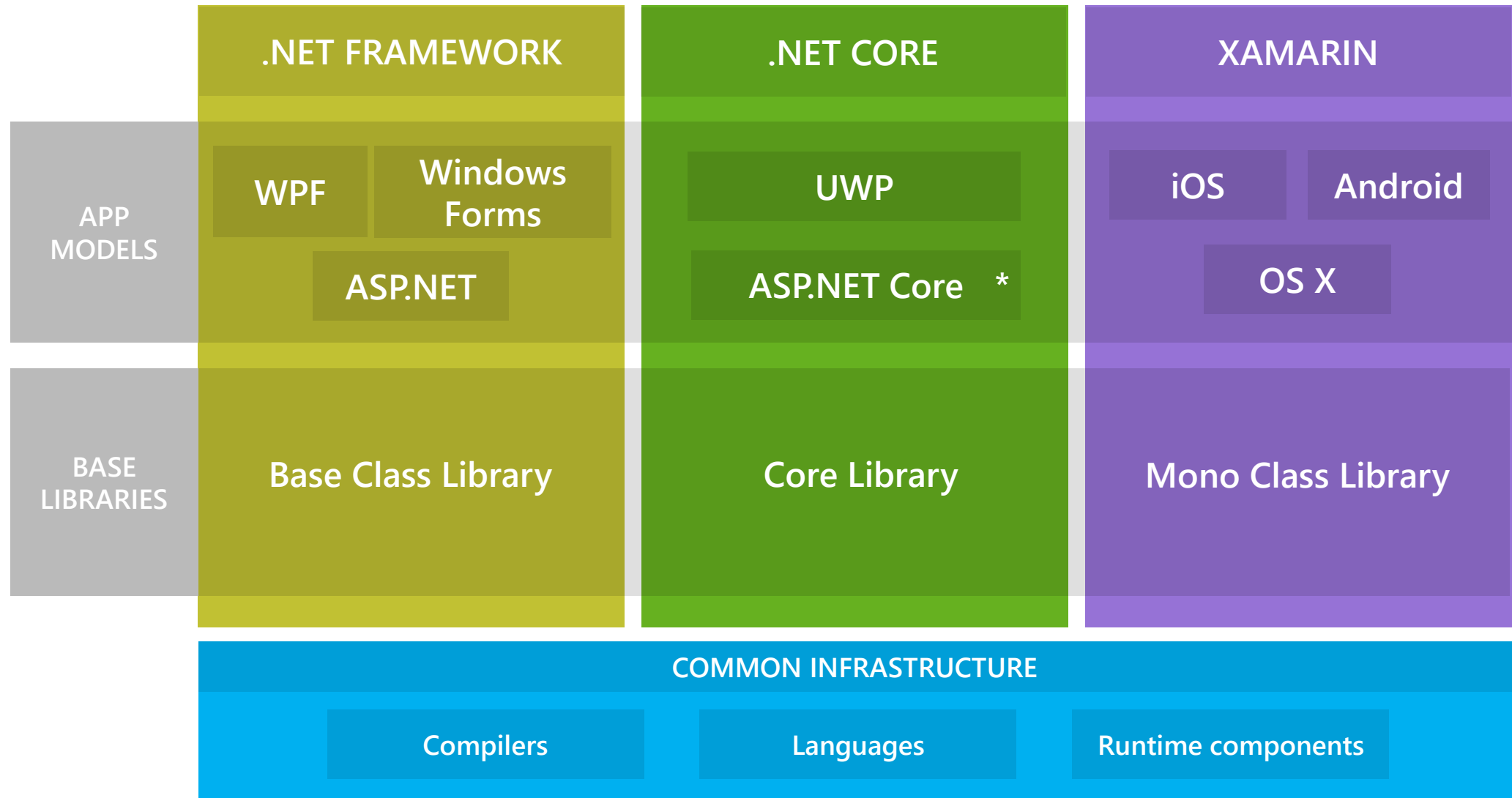
Demo: One ASP.NET

Module 1: Overview

Section 2: .NET Platform

Lesson: Overview

The Open .NET Ecosystem

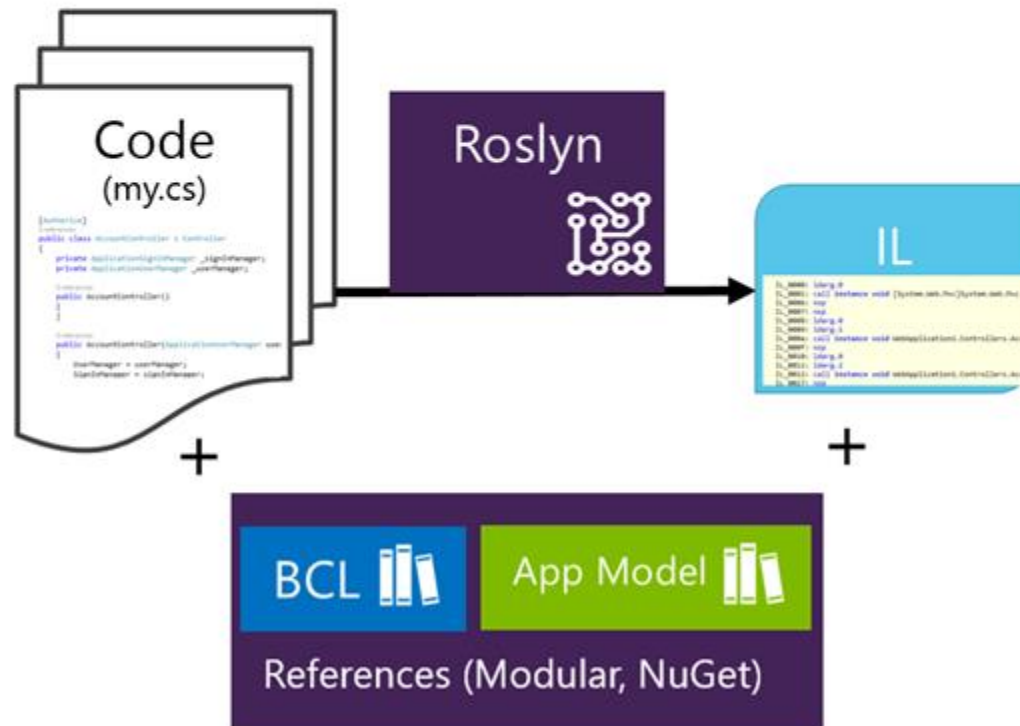


.NET Core

	.NET Core			ASP.NET Core	EF Core
Source License	MIT			Apache 2	Apache 2
Binary License	Microsoft EULA			Microsoft EULA	Microsoft EULA
Acquisition	Installer	Package-Manager	NuGet	NuGet	NuGet
OSes	Windows	macOS	Linux	Same	Same
App Deployment	Runtime-dependent	Self-contained	Docker	Same	
Side-by-side installs	Yes!			Yes!	Yes!

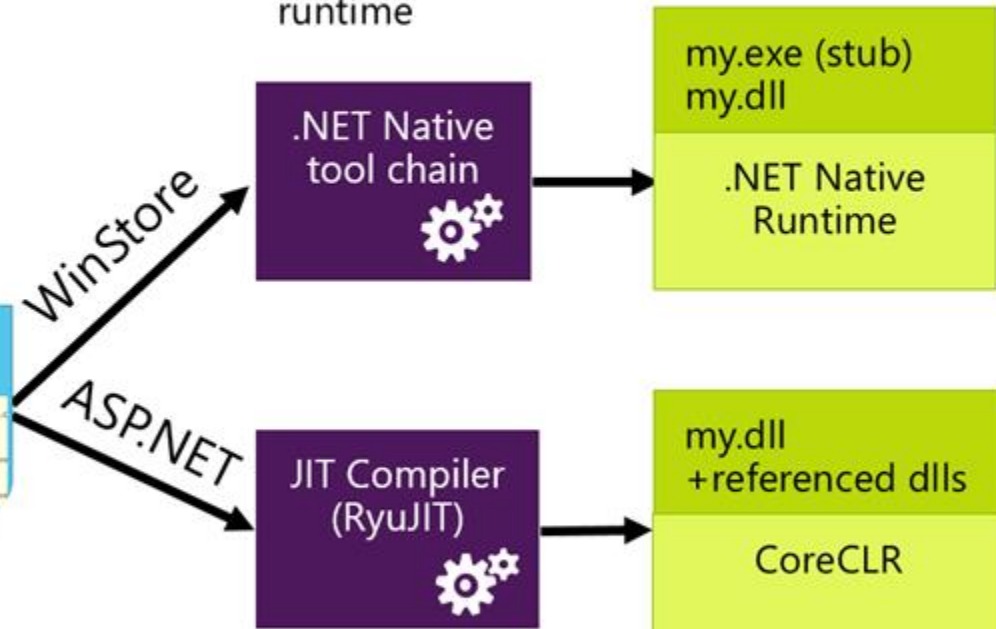
Code / Build / Debug

Roslyn takes your code and compiles it to Intermediate language (IL). You have module references to the BCL and App Model that you are targeting.



Deploy and Run

References are built with your app into one native dll deployed locally with runtime



References and CoreCLR are deployed with app locally, just-in-time (JIT) compilation on start up.

ASP.NET vs. ASP.NET Core

MSBuild/CodeDOM > csc.exe

Loose, GAC, NuGet

FCL, GAC, NuGet

IIS

.NET BCL and FCL

.NET CLR

IIS: WebEngine4.dll; EXE: OS

Windows

Compilation

Libraries

Application Frameworks

Web Server

Platform Libraries

Runtime

Runtime Loader

Operating System

.Net CLI (Roslyn)

NuGet, npm, Bower

NuGet

IIS, HTTP.SYS, Kestrel

.NET BCL and FCL; .NET on NuGet

.NET CLR; .NET Core CLR

.Net CLI

Windows, OSX, Linux

Which One is Right for Me?

ASP.NET Core

Build for Windows, Mac, or Linux

Use [MVC](#), or [Web API](#)

Multiple versions per machine

Develop with Visual Studio or Visual Studio Code using C#

New platform

Ultra performance

[Choose .NET Framework or .NET Core runtime](#)

ASP.NET

Build for Windows

Use [Web Forms](#), [SignalR](#), [MVC](#), [Web API](#), or [Web Pages](#)

One version per machine

Develop with Visual Studio using C#, VB or F#

Mature platform

High performance

Use .NET Framework runtime

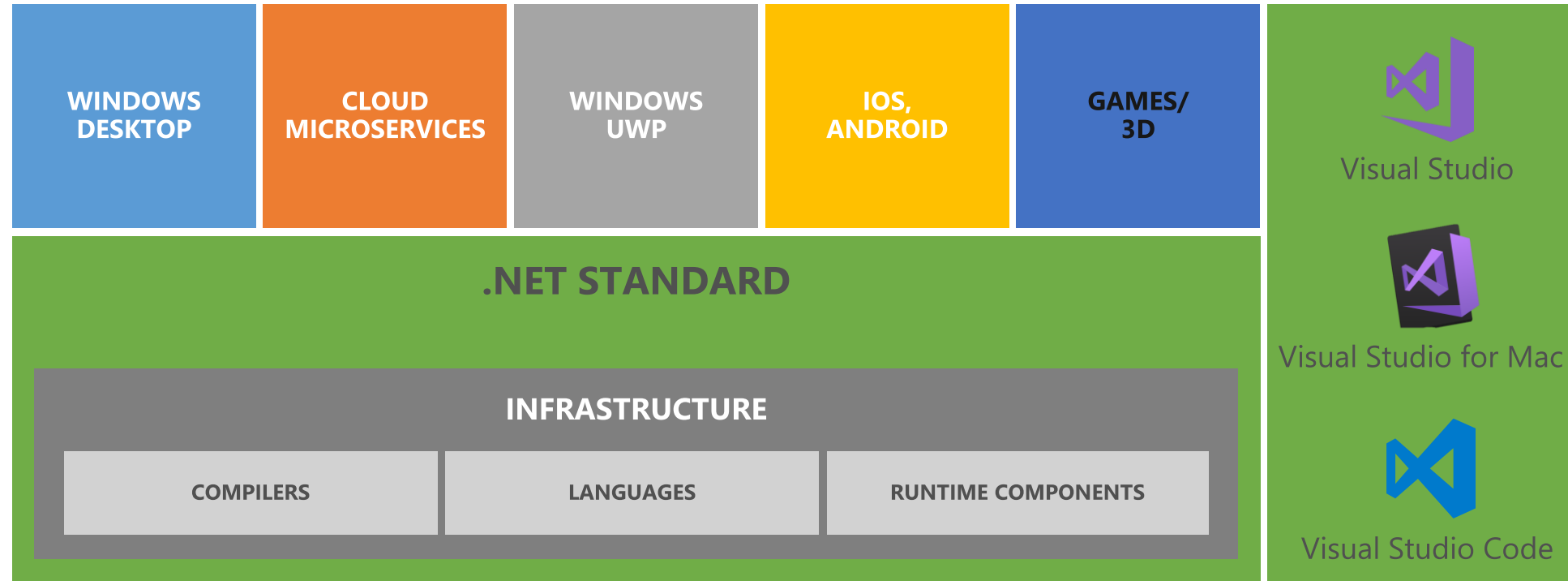
.NET Framework vs. .NET Core (Server Apps)

.NET Framework	.NET Core
Current application runs on .NET framework. Recommended to extend it instead of migrating	Cross-platform needs
Need 3 rd party libraries not available on .NET Core	Targeting microservices
Need .NET technologies not available on .NET Core	Using Docker containers
Need a platform not supported by .NET Core	Need high performance & scalable systems
	Side-by-side .NET versions by application
	Fully open-source

.NET Standard Library

- Goal: Establish greater uniformity in the .NET ecosystem
- A set of APIs that all .NET platforms have to implement
- Unifies the .NET platform and prevents future fragmentation
- .NET Standard will replace Portable Class Libraries (PCLs) as the tooling story for building multi-platform .NET libraries.
- Addresses three main scenarios:
 - Defines uniform set of BCL APIs for all .NET implementations to implement, independent of workload.
 - Enables developers to produce portable libraries that are usable across .NET implementations, using this same set of APIs.
 - Reduces or even eliminates conditional compilation of shared source due to .NET APIs, only for OS APIs.

.NET Standard



.NET Standard allows sharing code, binaries, and skills between .NET client, server, and all flavors

.NET Standard provides a specification for any platform to implement

All .NET runtimes provided by Microsoft implement the standard

.NET Standard Library

.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0	2.1
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	3.0
.NET Framework ¹	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1 ²	4.6.1 ²	4.6.1 ²	N/A ³
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4	6.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14	12.16
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8	5.16
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0	10.0
Universal Windows Platform	10.0	10.0	10.0	10.0	10.0	10.0.16299	10.0.16299	10.0.16299	TBD
Unity	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	TBD

³ .NET Framework won't support .NET Standard 2.1 or later versions. For more details, see the [announcement of .NET Standard 2.1](#).

APIs in .NET Standard 2.0

XML

XLinq • XML Document • XPath • Schema • XSL

SERIALIZATION

BinaryFormatter • Data Contract • XML

NETWORKING

Sockets • HTTP • Mail • WebSockets

IO

Files • Compression • MMF

THREADING

Threads • Thread Pool • Tasks

CORE

Primitives • Collections • Reflection • Interop • Linq

.NET Standard 2.0 coverage and support

Much bigger API Surface

We have more than doubled the set of available APIs from **13k** in [.NET Standard 1.6](#) to **32k** in [.NET Standard 2.0](#). Most of them are existing .NET Framework APIs.

.NET Framework compatibility mode

The vast majority of NuGet packages are currently still targeting .NET Framework. Many projects are currently blocked from moving to .NET Standard because not all their dependencies are targeting .NET Standard yet. We added a compatibility mode that allows .NET Standard projects to reference .NET Framework libraries. Found that [70% of all NuGet packages on nuget.org are API compatible](#) with .NET Standard 2.0. So in practice it unblocks many projects.

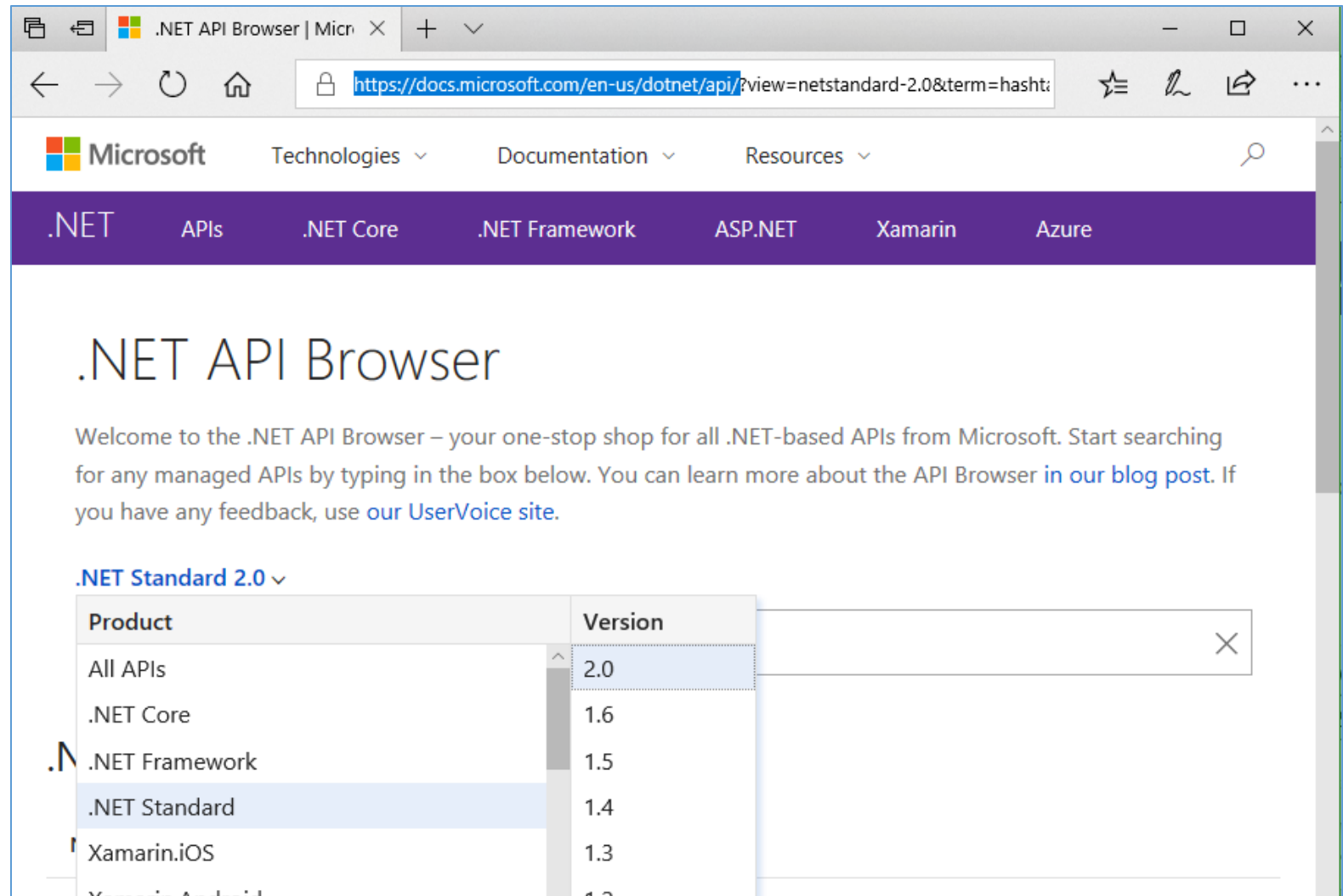
Which version of .NET Standard should I target?

- When choosing a .NET Standard version you should consider this trade-off:
 - The higher the version, the more APIs are available to you.
 - The lower the version, the more platforms you can run on.
- So generally speaking, you should target the lowest version you get away with.

.NET API Browser

Is one-stop shop for all .NET-based APIs from Microsoft. You can search for any managed APIs in it.

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



Demo:

.NET Standard 2.1

Module 1: Overview

Section 3: ASP.NET Core

Lesson: ASP.NET Core Projects

ASP.NET Core Project File

- ***.csproj**
 - Simplified project file
 - Automatically includes all source files in/under the folder containing project.json
- All project folder files shown in Solution Explorer
 - Visual Studio automatically monitors the ASP.NET Core project directory files
- project.json no longer supported
 - Migrated to *.csproj through Visual Studio Migration or through `dotnet migrate` on CLI

ASP.NET Core Project File Contents

```
<Project Sdk="Microsoft.NET.Sdk.Web">
```

```
  <PropertyGroup>
```

```
    <TargetFramework>netcoreapp2.2</TargetFramework>
```

```
    <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
```

```
    <RootNamespace>netcore2._2</RootNamespace>
```

```
  </PropertyGroup>
```

```
  <ItemGroup>
```

```
    <PackageReference Include="Microsoft.AspNetCore.App" />
```

```
    <PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.2.0" PrivateAssets="All" />
```

```
  </ItemGroup>
```

```
</Project>
```

Not required starting
with netcoreapp3.0

```
<Project Sdk="Microsoft.NET.Sdk.Web">
```

```
  <PropertyGroup>
```

```
    <TargetFramework>netcoreapp3.0</TargetFramework>
```

```
  </PropertyGroup>
```

```
  <ItemGroup>
```

```
  </ItemGroup>
```

```
</Project>
```

ASP.NET Core Project File Contents

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.2</TargetFramework>
    <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
    <RootNamespace>netcore2._2</RootNamespace>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.App" />
    <PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.2.0" PrivateAssets="All" />
  </ItemGroup>

</Project>
```

ASP.NET Core shared
framework

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
  </ItemGroup>

</Project>
```

What is Microsoft.AspNetCore.App metapackage?

- **Microsoft.AspNetCore.App** is installed when the .NET Core 3.0 or later SDK is installed. The shared framework is the set of assemblies (.dll files) that are installed on the machine and includes a runtime component and a targeting pack
- **Projects that target the Microsoft.NET.Sdk.Web SDK implicitly reference the Microsoft.AspNetCore.App framework**
<Project Sdk="Microsoft.NET.Sdk.Web">
- **ASP.NET Core 3.0 removes some assemblies that were previously part of the Microsoft.AspNetCore.App package reference. Most notable sub-components**
 - Json.NET (Newtonsoft.Json)
 - Entity Framework Core (Microsoft.EntityFrameworkCore.*)
 - Microsoft.CodeAnalysis (Roslyn)

Shared Framework – Deep Dive

- .NET Core apps run in one of two modes: **framework-dependent** or **self-contained**
- **Framework-dependent** deployment relies on the presence of a shared system-wide version of .NET Core
- **Self-contained** deployment doesn't rely on the presence of shared components on the target system. All components are included with the application

Shared Framework – Deep Dive

- To put it simply, a .NET Core shared framework is a folder of assemblies (*.dll files) that are not in the application folder
- These assemblies version and release together. This folder is one part of the “shared system-wide version of .NET Core”, and is usually found in C:/Program Files/dotnet/shared.
- You can produce both kinds of apps with these command line instructions:
 - `dotnet publish --runtime win10-x64 --output bin/self_contained_app/`
 - `dotnet publish --output bin/framework_dependent_app/`

.NET Runtimes





- Build could be done for “common” platform or for specific platform

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.0</TargetFramework>
    <RuntimeIdentifiers>win10-x64</RuntimeIdentifiers>
  </PropertyGroup>

  <ItemGroup>
  </ItemGroup>

</Project>
```

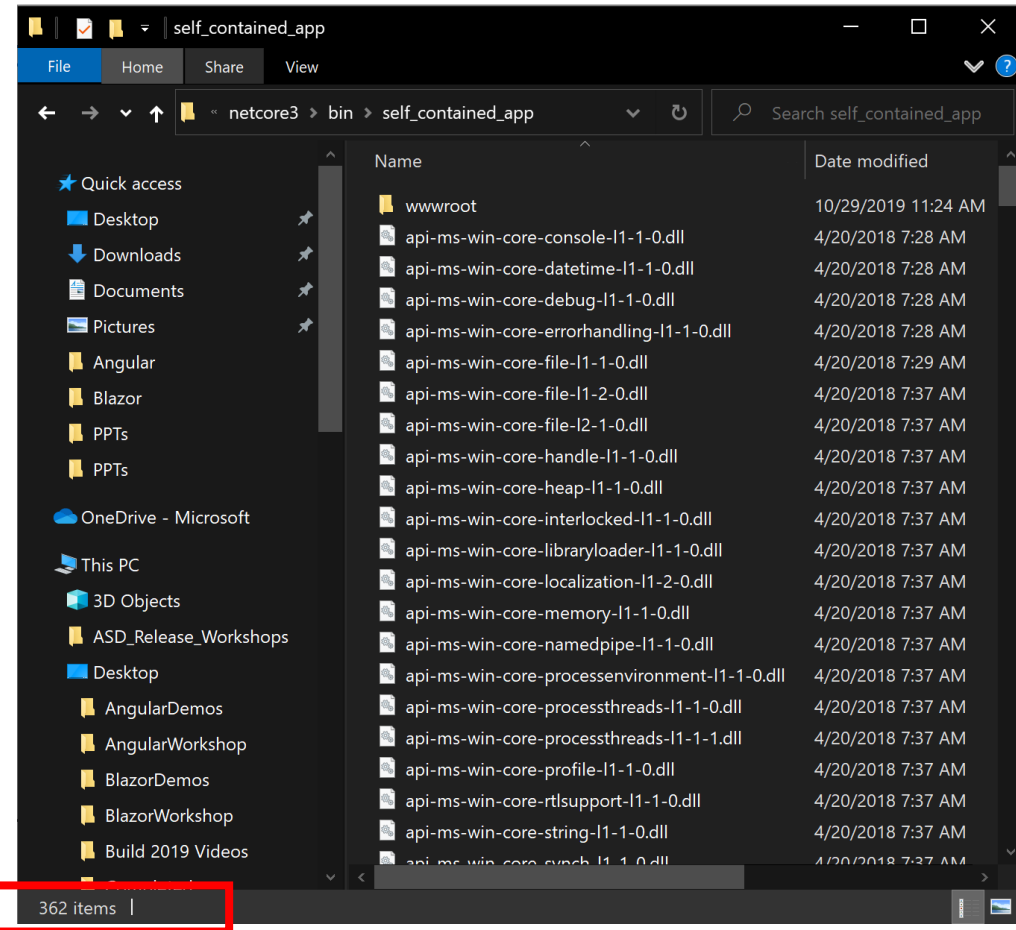
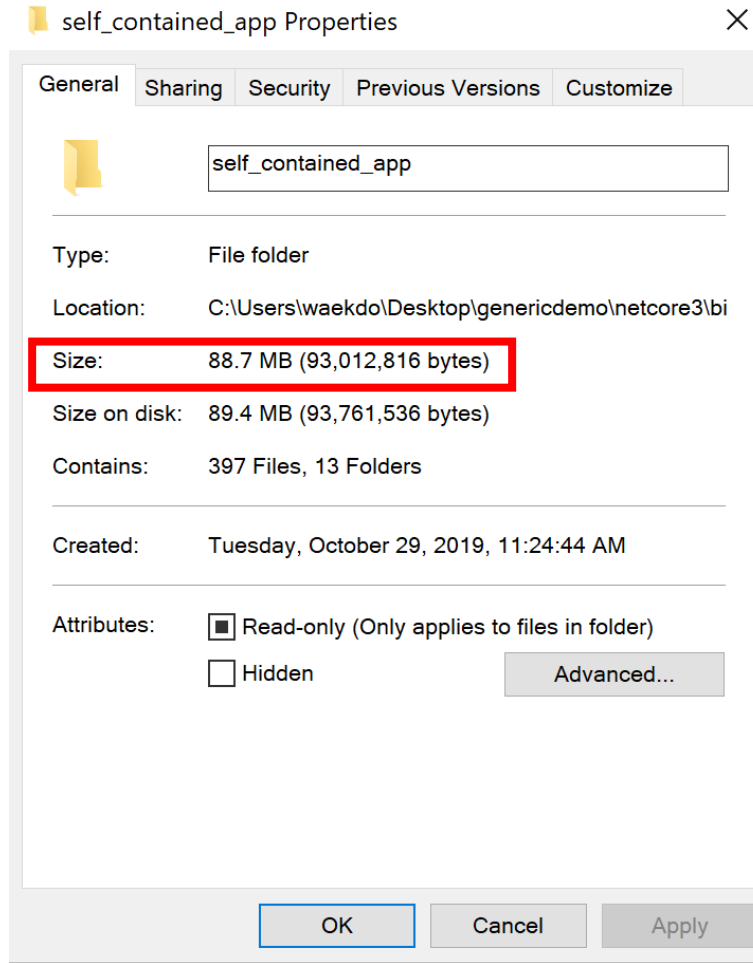
<< BuildForSpecificRuntime > bin > Debug > netcoreapp1.1 >		
<input type="checkbox"/>	Name	Date modified
	ubuntu.16.10-x64	11-12-16 18:50
	win10-x64	11-12-16 18:49
	BuildForSpecificRuntime.dll	11-12-16 18:50
	BuildForSpecificRuntime.pdb	11-12-16 18:50

```
C:\t\ASP.NETCore\Demos\Module 01 - Overview\BuildForSpecificRuntime>dotnet publish --runtime ubuntu.16.10-x64
Publishing BuildForSpecificRuntime for .NETCoreApp,Version=v1.1/ubuntu.16.10-x64
Project BuildForSpecificRuntime (.NETCoreApp,Version=v1.1) was previously compiled. Skipping compilation.
publish: Published to C:\t\ASP.NETCore\Demos\Module 01 - Overview\BuildForSpecificRuntime\bin\Debug\netcoreapp1.1\ubuntu.16.10-x64\publish
Published 1/1 projects successfully

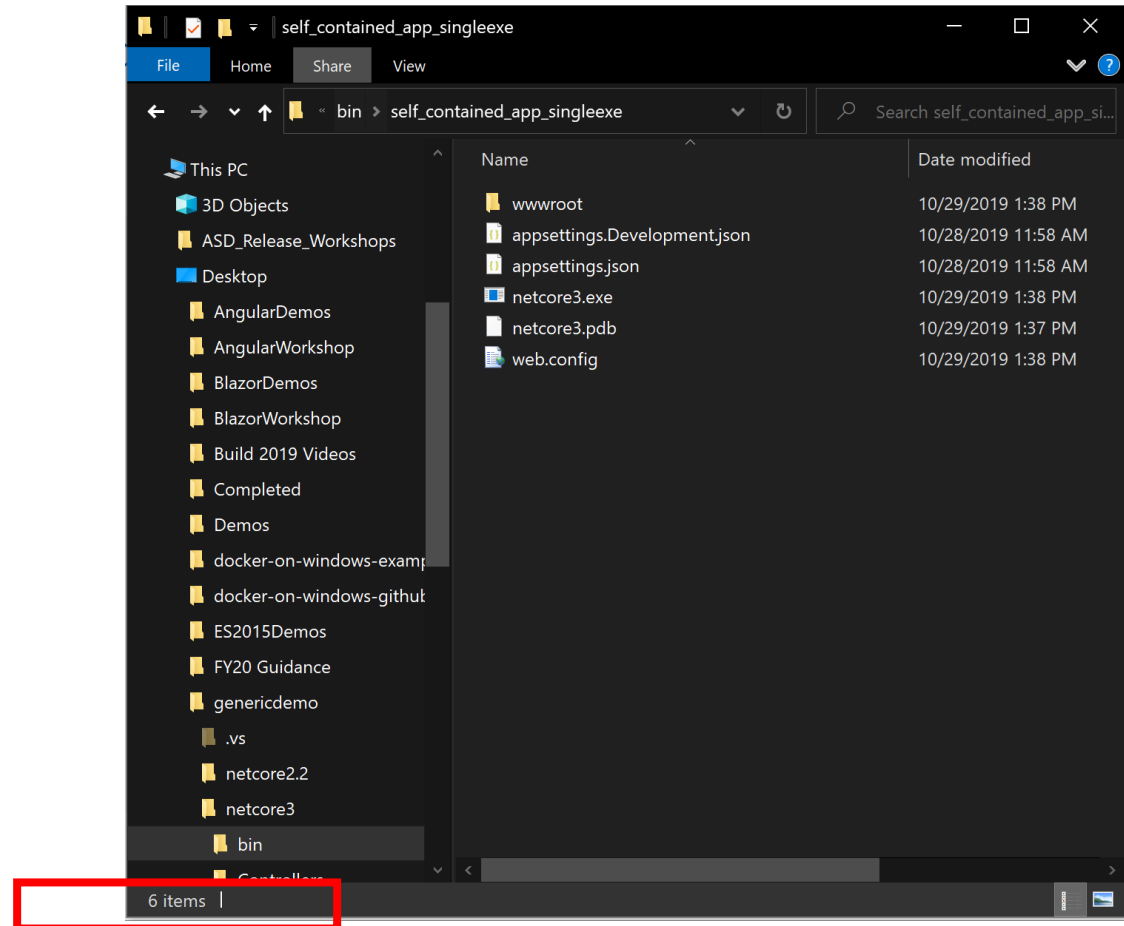
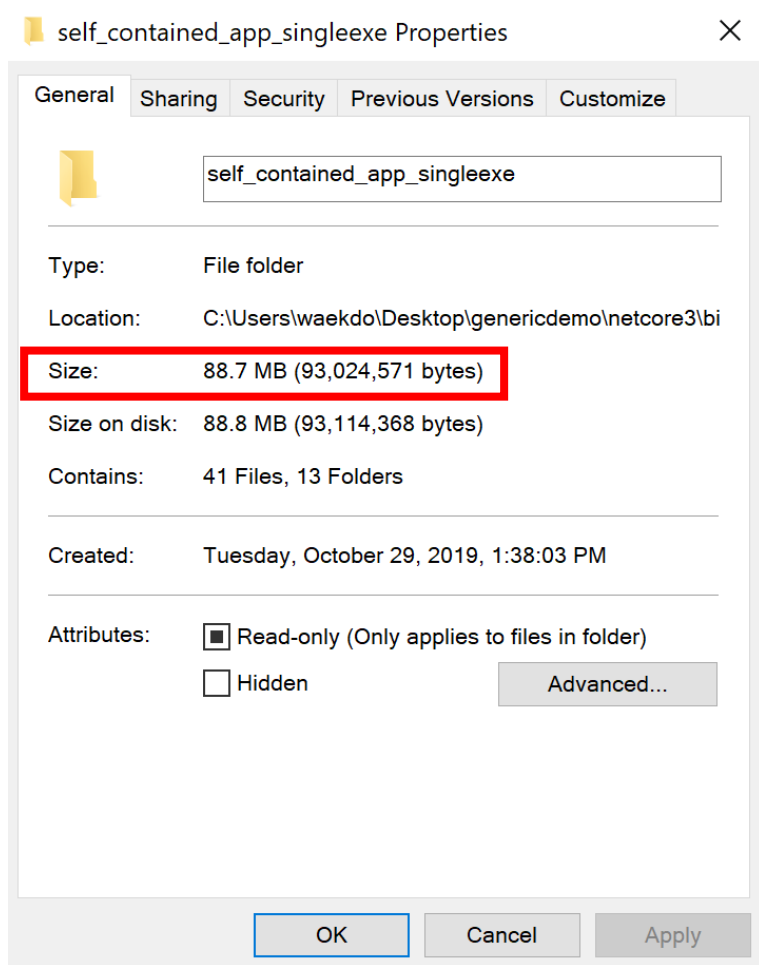
C:\t\ASP.NETCore\Demos\Module 01 - Overview\BuildForSpecificRuntime>dotnet publish --runtime win10-x64
Publishing BuildForSpecificRuntime for .NETCoreApp,Version=v1.1/win10-x64
Project BuildForSpecificRuntime (.NETCoreApp,Version=v1.1) was previously compiled. Skipping compilation.
publish: Published to C:\t\ASP.NETCore\Demos\Module 01 - Overview\BuildForSpecificRuntime\bin\Debug\netcoreapp1.1\win10-x64\publish
Published 1/1 projects successfully
```

Demo: Build On Windows For Different Runtimes

Publishing A Single EXE File In .NET Core 3.0

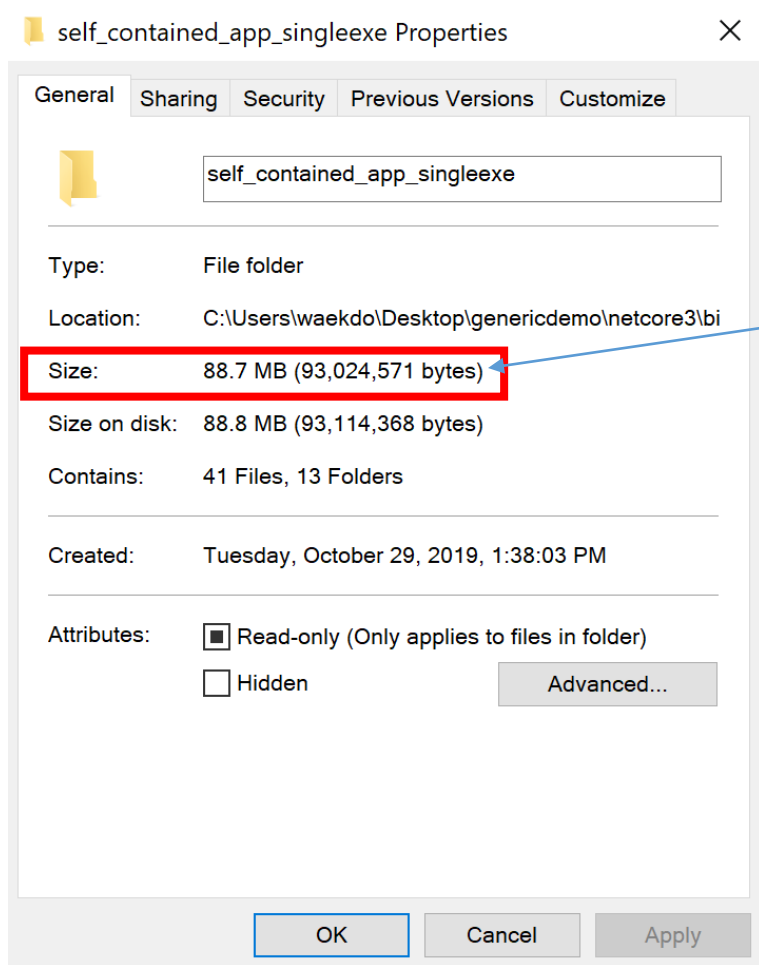


Publishing A Single EXE File In .NET Core 3.0



Demo: Publishing A Single EXE File In .NET Core 3.0

File Size And Startup Cost



This is over 80 MB!!!

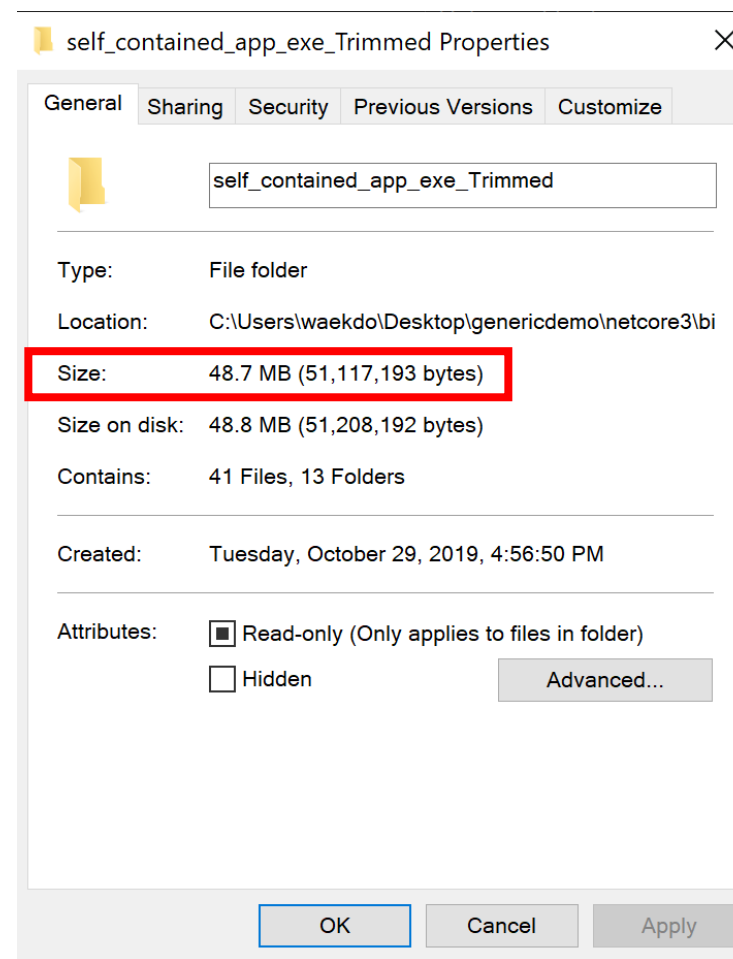
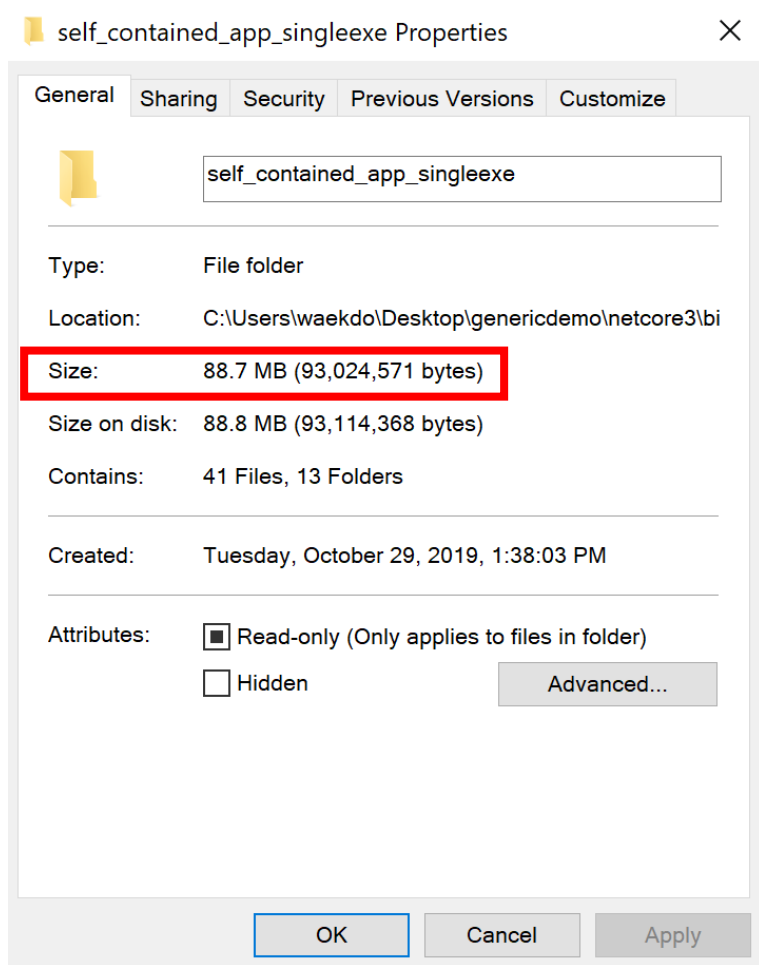
File Size And Startup Cost

- The other issue you may find is that there is a slight startup cost when running the self contained executable for the first time
 - It needs to essentially unzip all dependencies to a temporary directory on first run. It's not too high (5 seconds or so), but it's noticeable
 - Subsequent runs use the already unzipped temp folder and so startup is immediate

The PublishTrimmed Flag With IL Linker

- Starting with .Net Core 3 you ILLink.Tasks is no longer supported and instead the Tree Trimming feature is built into .NET Core directly

The PublishTrimmed Flag With IL Linker



Demo: The PublishTrimmed Flag With IL Linker

The PublishTrimmed Flag With IL Linker

- Reflected Assemblies

- Through various forms of reflection, we may end up loading assemblies at runtime that aren't direct references. Take this (very convoluted) example of loading an assembly at runtime :

```
static void Main(string[] args)
{
    Console.WriteLine(Assembly.Load("System.Security").FullName);
    Console.ReadLine();
}
```

- Now when debugging this locally, and we have .NET Core installed, we ask for System.Security and it knows what that is because we are using the installed .NET Core platform. So running it, we get :

```
System.Security, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
```

The PublishTrimmed Flag With IL Linker

- Reflected Assemblies
 - But if we publish this using the PublishTrimmed flag from the command line, then run it :

Unhandled Exception: System.IO.FileNotFoundException: Could not load file or assembly 'System.Security, Culture=neutral, PublicKeyToken=null'.
The system cannot find the file specified.

Demo: Reflected Assemblies

Using .csproj To Create A Reduced Single Executable

```
<PropertyGroup>  
  <TargetFramework>netcoreapp3.0</TargetFramework>  
  <RuntimeIdentifier>win10-x64</RuntimeIdentifier>  
  <PublishSingleFile>true</PublishSingleFile>  
  <PublishTrimmed>true</PublishTrimmed>  
</PropertyGroup>
```

ASP.NET Core Project File Contents

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.2</TargetFramework>
    <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
    <RootNamespace>netcore2._2</RootNamespace>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.App" />
    <PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.2.0" PrivateAssets="All" />
  </ItemGroup>

</Project>
```

In process vs out of process

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
  </ItemGroup>

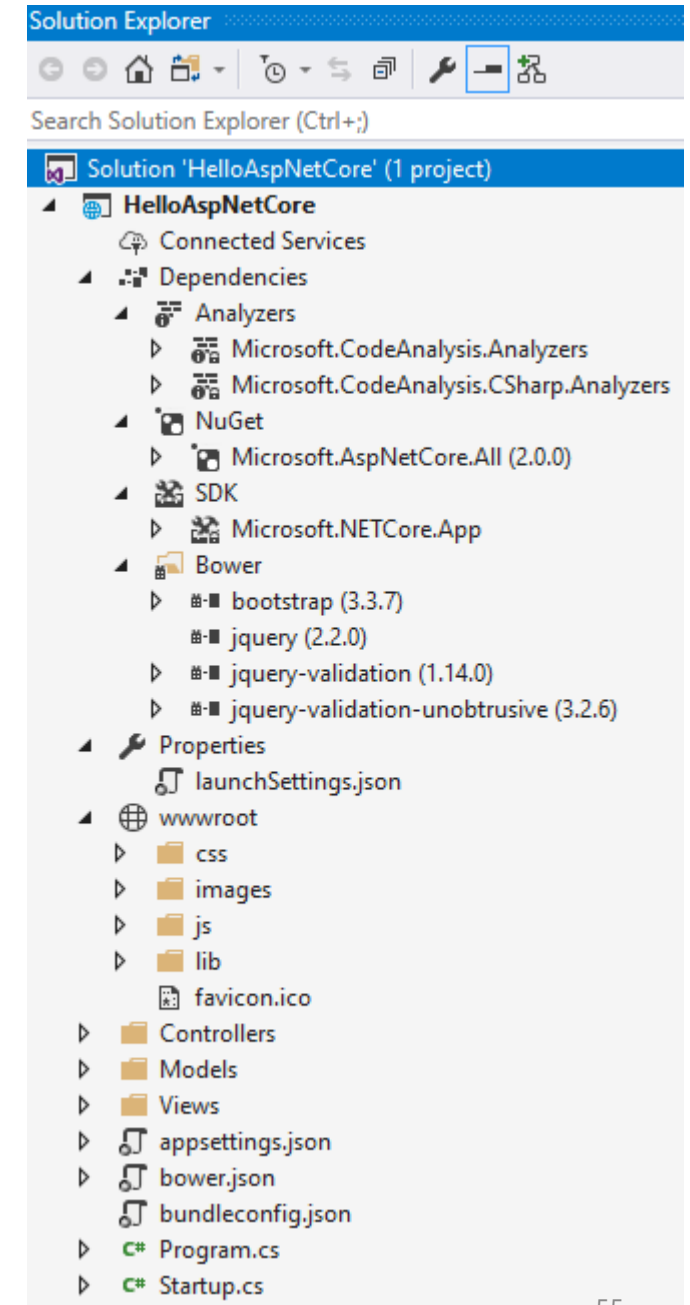
</Project>
```

In Process Vs Out Of Process

- More info [here](#)

ASP.NET Core Project Layout

- wwwroot folder
 - Static file placeholder
- Dependencies
 - Bower
 - NPM
- References
 - NuGet packages
 - No direct DLL references
- Configuration through config.json, appsettings.json, etc.
- No web.config unless app is hosted in IIS or IIS Express



Package Managers

- NuGet

- Package manager for Microsoft development platform
- Great for server-side libraries: more than 55,000 packages available
- .NET Framework and runtime now available through NuGet

- NPM

- Package manager for JavaScript; widely used in JavaScript development community
- First class citizen in ASP.NET and Visual Studio 2017
- More than 250,000 packages are available

- Bower

- Package manager for the web (HTML, JavaScript, and CSS)
- Installed using NPM; suited for web application front-end development
- More than 36,000 packages are available



Configuration System

Config File	Description
*.csproj	<ul style="list-style-type: none">• Main project file;• NuGet package dependencies;• Targeted frameworks; Commands, package includes and excludes
Config.json	<ul style="list-style-type: none">• Application specific configuration, such as database connection strings
AppSettings.json	<ul style="list-style-type: none">• Other application settings, such as Application Insights instrumentation key
Bower.json	<ul style="list-style-type: none">• Lists bower packages used by an app• Bower package configuration
Gruntfile.js	<ul style="list-style-type: none">• Configures grunt tasks – with options, names and operations tasks
Gulpfile.js	<ul style="list-style-type: none">• Configure Gulp tasks
Package.json	<ul style="list-style-type: none">• Lists NPM packages used by an app• NPM package configuration
bundleconfig.json	<ul style="list-style-type: none">• Concatenate (bundle) and minify CSS, JavaScript and HTML files without verbose toolchain.

Configuration files are completely **pluggable**

The default JSON format can be replaced with an XML/.ini file or even take settings from a SQL Server database

Task Runners

- Tasks runners are used to achieve automation for client-side code:
 - LESS/SL compilation to CSS
 - JavaScript minification
 - JSLint/JSHint
 - JavaScript unit tests
- Grunt
 - Most popular task runner for JavaScript
 - More than 5800+ plug-ins are available
- Gulp
 - Task runner for JavaScript
 - Relies on the code logic based on the pipes for simplification
 - More than 2600+ plug-ins

New project templates!

- CLI & VS2019 now share project templates
- New in 2.0:
 - SPA templates for Angular & React
 - Razor Pages with Authentication options
 - Authentication using Azure B2C
 - 2-factor authentication using Authenticator apps
- Simplified project creation experience in VS2019
- You can use Docker Support

Module 1: Overview

Section 3: ASP.NET Core

Lesson: Command Line Interface
(CLI)

.NET Core Command Line Interface (CLI)

- Cross-platform toolchain for developing .NET Core applications
- Primary layer built upon by Visual Studio, editors, build orchestrators, etc.
- Cross-platform with same surface area for supported platforms
- Language agnostic
- Target agnostic

```
dotnet new  
dotnet restore  
dotnet build --output /stuff  
dotnet run /stuff/new.dll
```

CLI Command Examples

<code>dotnet restore</code>	Uses NuGet to restore dependencies as well as project-specific tools that are specified in the project file in parallel.
<code>dotnet build</code>	Restores any dependencies then builds the project and its dependencies into a set of binaries. The binaries include the project's code in Intermediate Language (IL) files with a .dll extension and symbol files used for debugging with a .pdb extension.
<code>dotnet run</code>	It allows you to run your application from the source code with one command. It's useful for fast iterative development from the command line. The command depends on the dotnet build command to build the code. Any requirements for the build, such as that the project must be restored first.
<code>dotnet clean</code>	Cleans the output of the previous build.
<code>dotnet new web</code>	Create a new Empty web application then restores the dependencies/packages for it.

.NET Core Tooling

Visual Studio

VS Code

.NET Core
Command Line
tools

Shared SDK component

CLI dotnet new templates

.NET Core 2 introduce many new templates from the CLI

Example:

```
dotnet new angular -lang c#
```

This will install an ASP.NET Core Web Application which using Angular and C# language

Template description	Template name	Languages
Console application	console	[C#], F#, VB
Class library	classlib	[C#], F#, VB
Unit test project	mstest	[C#], F#, VB
xUnit test project	xunit	[C#], F#, VB
ASP.NET Core empty	web	[C#], F#
ASP.NET Core Web App (Model-View-Controller)	mvc	[C#], F#
ASP.NET Core Web App	razor	[C#]
ASP.NET Core with Angular	angular	[C#]
ASP.NET Core with React.js	react	[C#]
ASP.NET Core with React.js and Redux	reactredux	[C#]
ASP.NET Core Web API	webapi	[C#], F#
global.json file	globaljson	
Nuget config	nugetconfig	
Web config	webconfig	
Solution file	sln	
Razor page	page	
MVC/ViewImports	viewimports	
MVC ViewStart	viewstart	

.NET Core CLI Extensibility

- .NET Core is built for extensibility, you extend the CLI with your own custom commands and tooling
- The CLI tools can be extended in three main ways:
 1. Via NuGet packages on a per-project basis
Per-project tools are contained within the project's context, but they allow easy installation through restoration.
 2. Via NuGet packages with custom targets
Custom targets allow you to easily extend the build process with custom tasks.
 3. Via the system's PATH
PATH-based tools are good for general, cross-project tools that are usable on a single machine.

Example of extensibility is the EF Core commands

Entity Framework Core CLI

- EF Core CLI is considered an extension to .NET Core CLI
- It can be installed through NuGet and will be available through the CLI in this directory
- The EF Core .NET Command Line Tools are installed by manually editing the *.csproj file.
 - Add **Microsoft.EntityFrameworkCore.Tools.DotNet** as a **DotNetCliToolReference**.
 - Execute **dotnet add package Microsoft.EntityFrameworkCore.Design**
 - Execute **dotnet restore**.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="2.0.0" PrivateAssets="All" />
  </ItemGroup>
  <ItemGroup>
    <DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet" Version="2.0.0" />
  </ItemGroup>
</Project>
```

Demo: .NET Core CLI & Visual Studio Code

Module 1: Overview

Section 3: ASP.NET Core

Lesson: ASP.NET Core on .NET
Framework

ASP.NET Core on .NET Framework

- Familiar .NET Framework Application Programming Interfaces (APIs)
- Ecosystem of existing packages
- Installed machine-wide
- Updated with the operating system
- *nix support via mono

Demo: ASP.NET Core on .NET Framework

Module 1: Overview

Section 3: ASP.NET Core

Lesson: ASP.NET Core on .NET
Core

ASP.NET Core on .NET Core

- .NET Core CLR runtime
- Optimized for servers
- Small footprint
- Class libraries in NuGet packages
- Side-by-side and portable
- Cross-platform: Windows, *nix

Demo: ASP.NET Core on .NET Core

Module 1: Overview

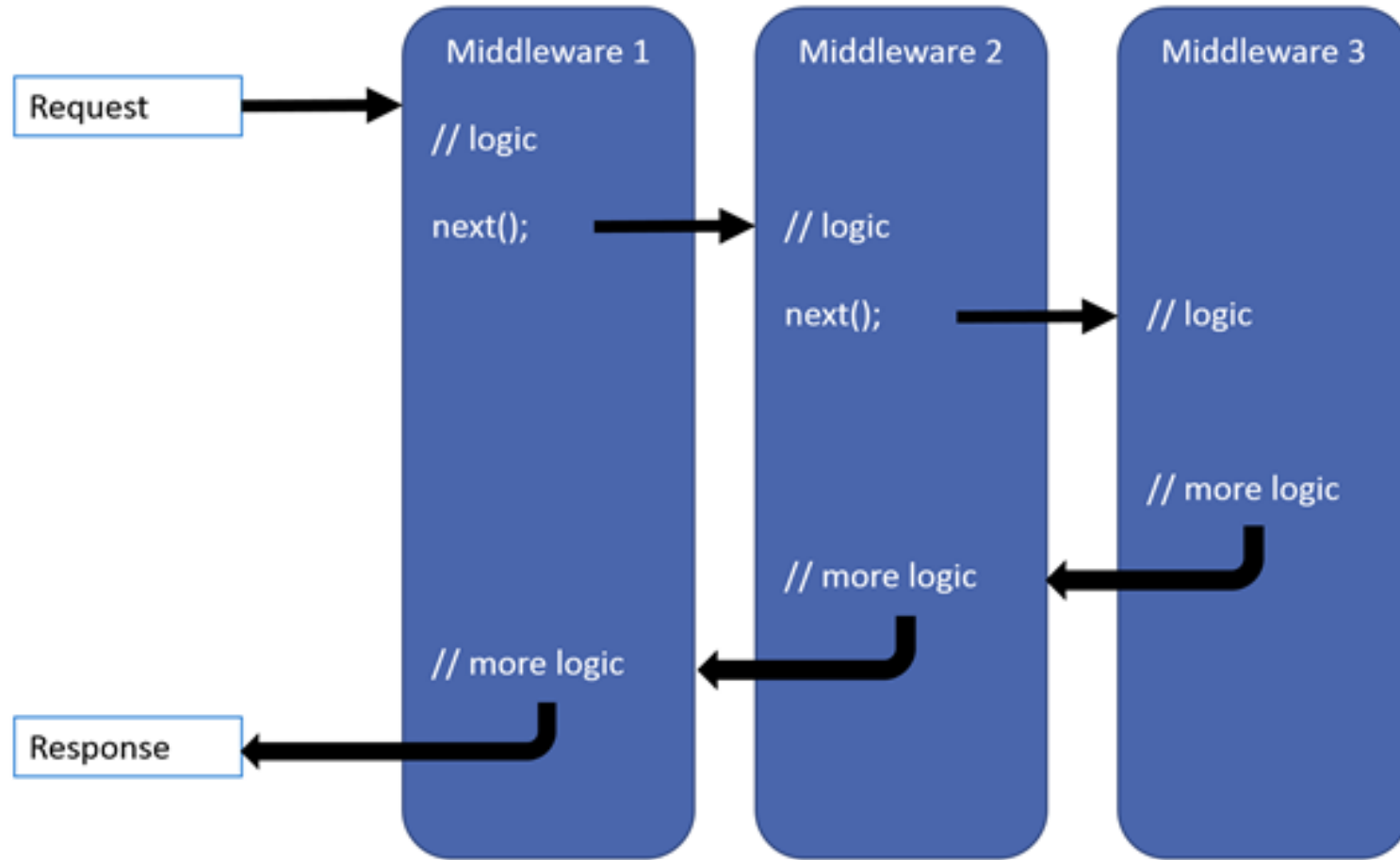
Section 3: ASP.NET Core

Lesson: Middleware

Middleware

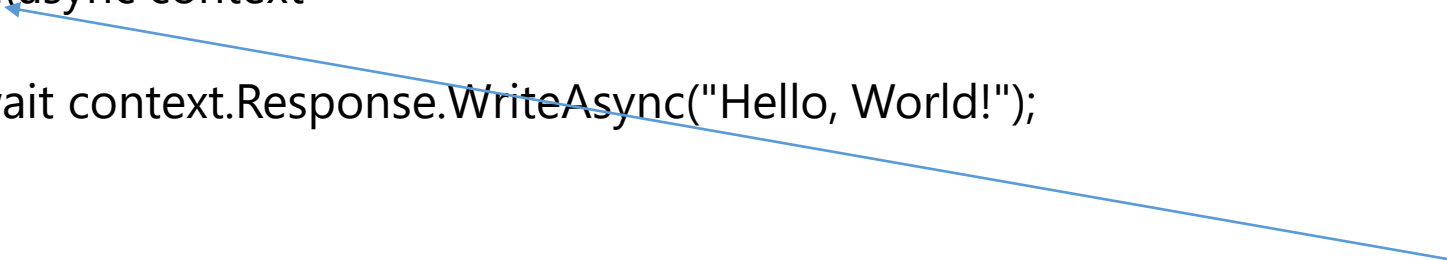
- Small application components assembled into an application pipeline to handle requests and responses
- Integrated support by ASP.NET Core
- Wired up in **Configure** method of **Startup** class
- Either invokes the next component in the chain or short-circuits it
- **Run**, **Map**, and **Use** extension methods
- Implemented in-line as anonymous method, or through a reusable class
- Order of **Use[Middleware]** statements in application's Configure method is very important

Middleware Pipeline



Simple ASP.NET Core Middleware

```
public class Startup {  
    public void Configure(IApplicationBuilder app) {  
        app.Run(async context =>  
        {  
            await context.Response.WriteAsync("Hello, World!");  
        });  
    }  
}
```



Run is a terminal
middleware

The first Run delegate
terminates the pipeline

Chain Multiple Request Delegates Together With *Use*

```
public class Startup
```

```
{
```

```
    public void
```

```
    {
```

```
        app.Use
```

```
    {
```

```
        //
```

```
        await
```

```
        //
```

```
    });
```

```
    app.Run(async context =>
```

```
    {
```

```
        await context.Response.WriteAsync("Hello from 2nd delegate.");
```

```
    });
```

```
    }
```

```
}
```

next parameter

the next
pipeline.

⚠ Warning

Don't call `next.Invoke` after the response has been sent to the client. Changes to `HttpResponse` after the response has started throw an exception. For example, changes such as setting headers and a status code throw an exception. Writing to the response body after calling `next`:

- May cause a protocol violation. For example, writing more than the stated `Content-Length`.
- May corrupt the body format. For example, writing an HTML footer to a CSS file.

`HasStarted` is a useful hint to indicate if headers have been sent or the body has been written to.

circuit the
t calling
meter

Built-in Middleware

Middleware	Description
Authentication	Provides authentication support
CORS	Configures Cross-Origin Resource Sharing
Diagnostics	Includes support for error pages and runtime information
Routing	Define and constrain request routes
Session	Provides support for managing user sessions
Static Files	Provides support for serving static files, and directory browsing

Full list can be found [here](#)

Demo: Writing Middleware

Module 1: Overview

Section 3: ASP.NET Core

Lesson: Hosting

Hosting in ASP.NET Core

- Host is responsible for app startup and lifetime management. At a minimum, the host configures a server and a request processing pipeline.
- Many defaults encapsulated in new API: **WebHost.CreateDefaultBuilder**

```
public class Program
{
    0 references | 0 exceptions
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    1 reference | 0 exceptions
    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}
```

Default Configurations for `WebHost.CreateDefaultBuilder`

- Configures Kestrel as the web server.
- Sets the content root to **`Directory.GetCurrentDirectory`**.
- Loads optional configuration from:
 - `appsettings.json`.
 - `appsettings.{Environment}.json`.
 - User secrets when the app runs in the **`Development`** environment.
 - Environment variables.
 - Command-line arguments.
- Configures logging for console and debug output with log filtering rules specified in a Logging configuration section of an `appsettings.json` or `appsettings.{Environment}.json` file.
- Enables IIS integration by configuring the base path and port the server should listen on when using the ASP.NET Core Module if you're running under IIS.

Host Configuration Values

- **Server URLs:** Indicates the IP addresses or host addresses with ports and protocols that the server should listen on for requests.
`.UseUrls("http://*:5000;http://localhost:5001;https://hostname:5002")`
- **Startup Assembly:** Determines the assembly to search for the Startup class.
`.UseStartup("StartupAssemblyName")`
- **Environment:** Sets the app's environment
`.UseEnvironment("Development")`
- **Contents Root:** This setting determines where ASP.NET Core begins searching for content files, such as MVC views.
`.UseContentRoot("c:\\mywebsite")`

Host Configuration Values

- **Detailed Errors:** Determines if detailed errors should be captured.
`.UseSetting(WebHostDefaults.DetailedErrorsKey, "true")`
- **Capture Startup Errors:** This setting controls the capture of startup errors.
`.CaptureStartupErrors(true)`
- **Web Root:** Sets the relative path to the app's static assets.
`.UseWebRoot("(Content Root)/wwwroot")`

Module 1: Overview

Section 3: ASP.NET Core

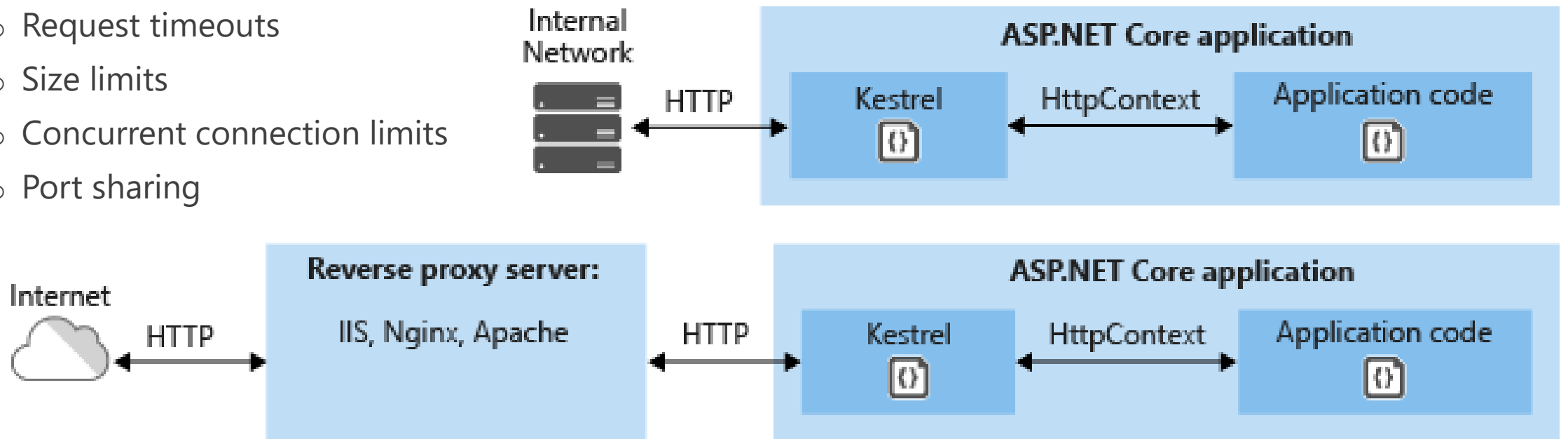
Lesson: Hosting Servers

ASP.NET Core Hosting

- ASP.NET Core is completely decoupled from the web server environment that hosts the application
- ASP.NET Core ships with:
 - **Kestrel**: Cross-platform HTTP server based on libuv, a cross-platform asynchronous I/O library
 - **WebListener**: Windows-only HTTP server based on the Http.Sys kernel driver
- ASP.NET Core defines a number of HTTP Feature Interfaces
 - Used by web servers and middleware to identify supported features

Kestrel

- Supported Features
 - HTTPS
 - WebSockets
 - Unix sockets for high performance behind Nginx
- Kestrel does not yet support:
 - Request timeouts
 - Size limits
 - Concurrent connection limits
 - Port sharing



ASP.NET Core Module

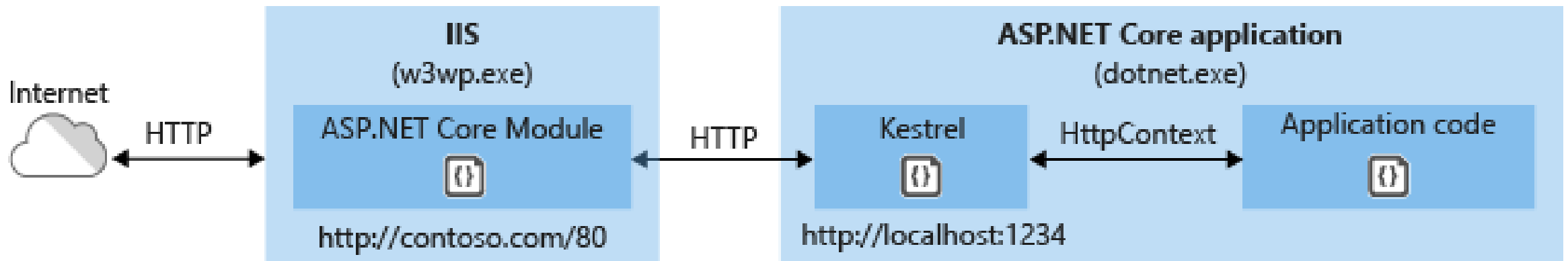
- Native IIS module hooked into IIS pipeline to redirect traffic to backend ASP.NET Core app
- Process management
 - Start dotnet.exe on first request
 - Restarts it when dotnet.exe crashes
- Advantages:
 - IIS App Pool does not run any managed code
 - Existing ASP.NET windows components are not required to be installed
 - Separate process for ASP.NET Core; existing ASP.NET modules can run alongside

```
var builder = new WebHostBuilder()
    .UseContentRoot(Directory.GetCurrentDirectory())
    .UseConfiguration(config)
    .UseStartup<Startup>()
    .UseUrls("http://localhost:5001")
    .UseIISIntegration()
    .UseKestrel(options =>
    {
        if (config["threadCount"] != null)
        {
            options.ThreadCount = int.Parse(config["threadCount"]);
        }
    });

var host = builder.Build();
host.Run();
```

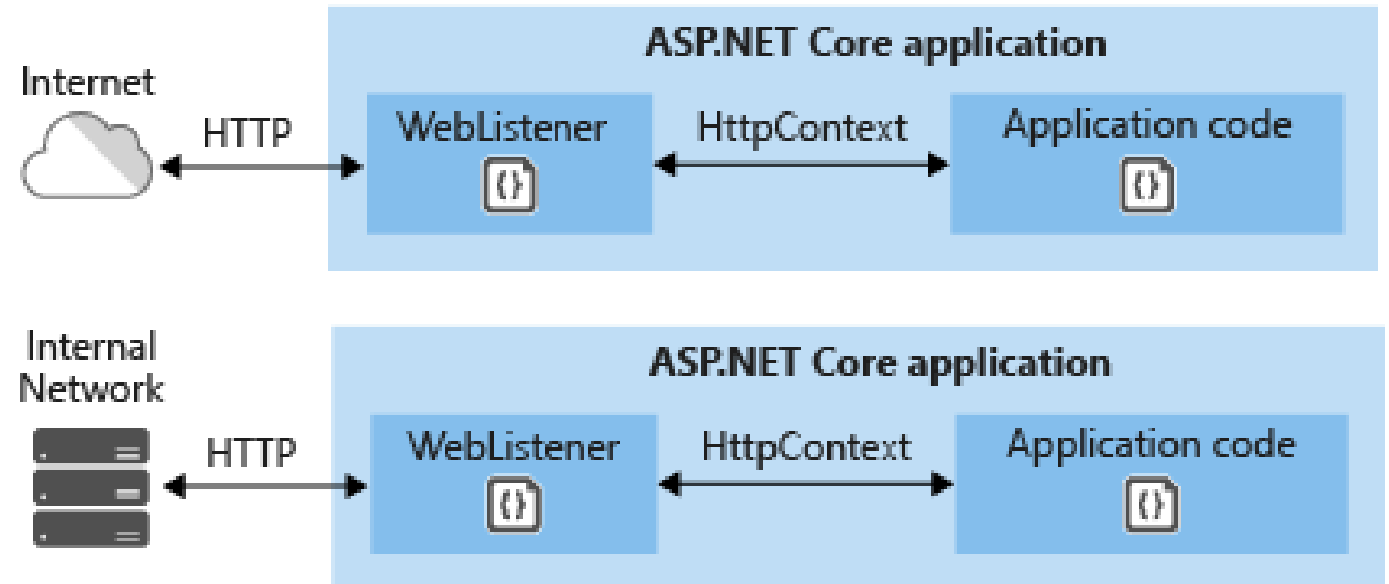
Request flow with ASP.NET Core Module (IIS)

1. Incoming web request is routed to primary port 80/443 through kernel model Http.Sys driver
2. Request forwarded to ASP.NET Core app (on non-80-443 port)
3. Kestrel picks up the request and pushes it into ASP.NET Core middleware pipeline
4. Middleware passes the request to application logic as HttpContext instance
5. Application HTTP response is eventually passed back to IIS













WebListener

- Supported Features
 - Windows Authentication
 - Port sharing
 - HTTPS with SNI
 - HTTP/2 over TLS (Windows 10)
 - Direct file transmission
 - Response caching
 - WebSockets



Which Web Server Should You Use?

Choosing Web Servers

	Windows	Linux/OSX	Development-Ready
IIS			
IIS Express			
WebListener			
Kestrel			
Apache/Nginx			

Module 1: Overview

Section 3: ASP.NET Core

Lesson: Working Environments,
Microsoft Azure, Web Hooks

Working Environments

- Working Environments
 - Development
 - Test
 - Staging
 - Production
- Startup Conventions
 - Startup → Startup{EnvironmentName} – for example, *StartupDevelopment*
 - ConfigureServices() → Configure[Environment]Services()
 - Configure() → Configure[Environment]()
- Applies to Microsoft Azure as well through App Settings in Azure Portal

Working Environment Configuration

Application Configuration: N/A

Build Platform: N/A

Debug

Profile: IIS Express [New... Delete]

Launch: IIS Express

☒ Launch URL:

☐ Use Specific Runtime:

Version	Platform	Architecture
1.0.0-rc1-update1	.NET Framework	x86

Environment Variables:

Name	Value
Hosting:Environment	Development

Add Remove

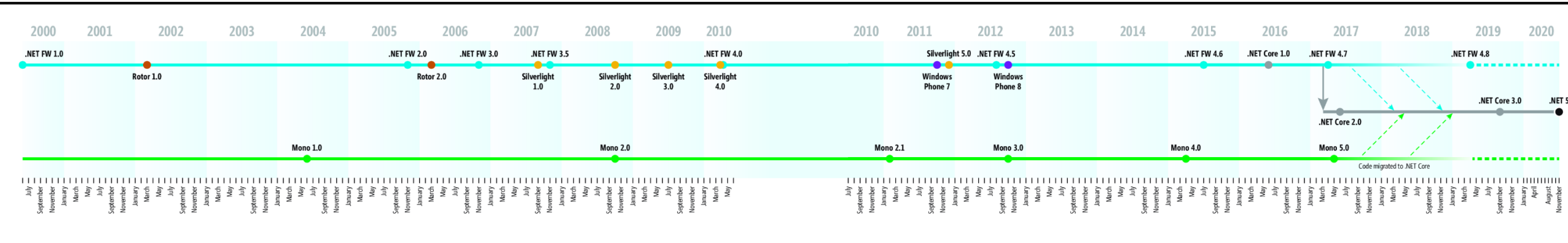
Azure Support

- Following services have CoreCLR libraries:
 - Azure Active Directory
 - Storage
 - SQL Server Database
 - Azure Management libraries (Azure Compute, network, storage, AppService, and Resource Manager, etc.)
- ASP.NET Core is fully supported on Azure App Service (Web Apps)

Major New Advancements in ASP.NET Core

- No-compile developer experience
 - Visual Studio uses Roslyn engine to compile ASP.NET Core projects
 - Pre-compiled projects on build request
- Dependency injection out-of-the-box
 - First class citizen and available throughout the entire stack
 - Minimalistic Dependency Injection (DI) container is included out-of-the-box; Bring Your Own Container(BYOC) is supported
- ASP.NET Core on OSX and Linux
 - .NET Core CLR and Mono runtime
- Task Runner is integrated to Visual Studio
 - Runs tasks for the following events: pre-built, post-built, clean, and project open

.NET Reunified: Microsoft's Plans for .NET 5



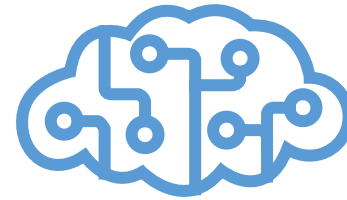
.NET Core 3.0 Themes



**Windows
desktop apps**



**Full-stack Web
Development**

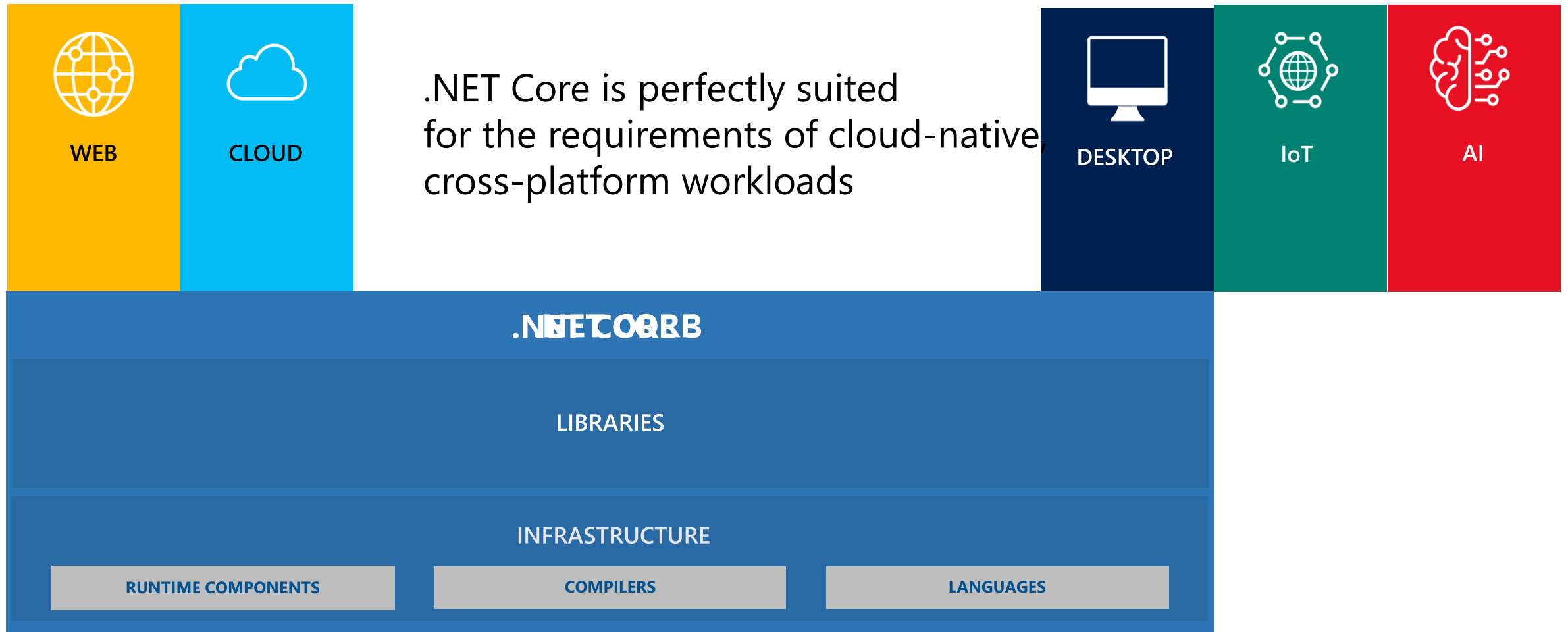


**Artificial
intelligence &
Machine Learning**



Big data

.NET Core 3

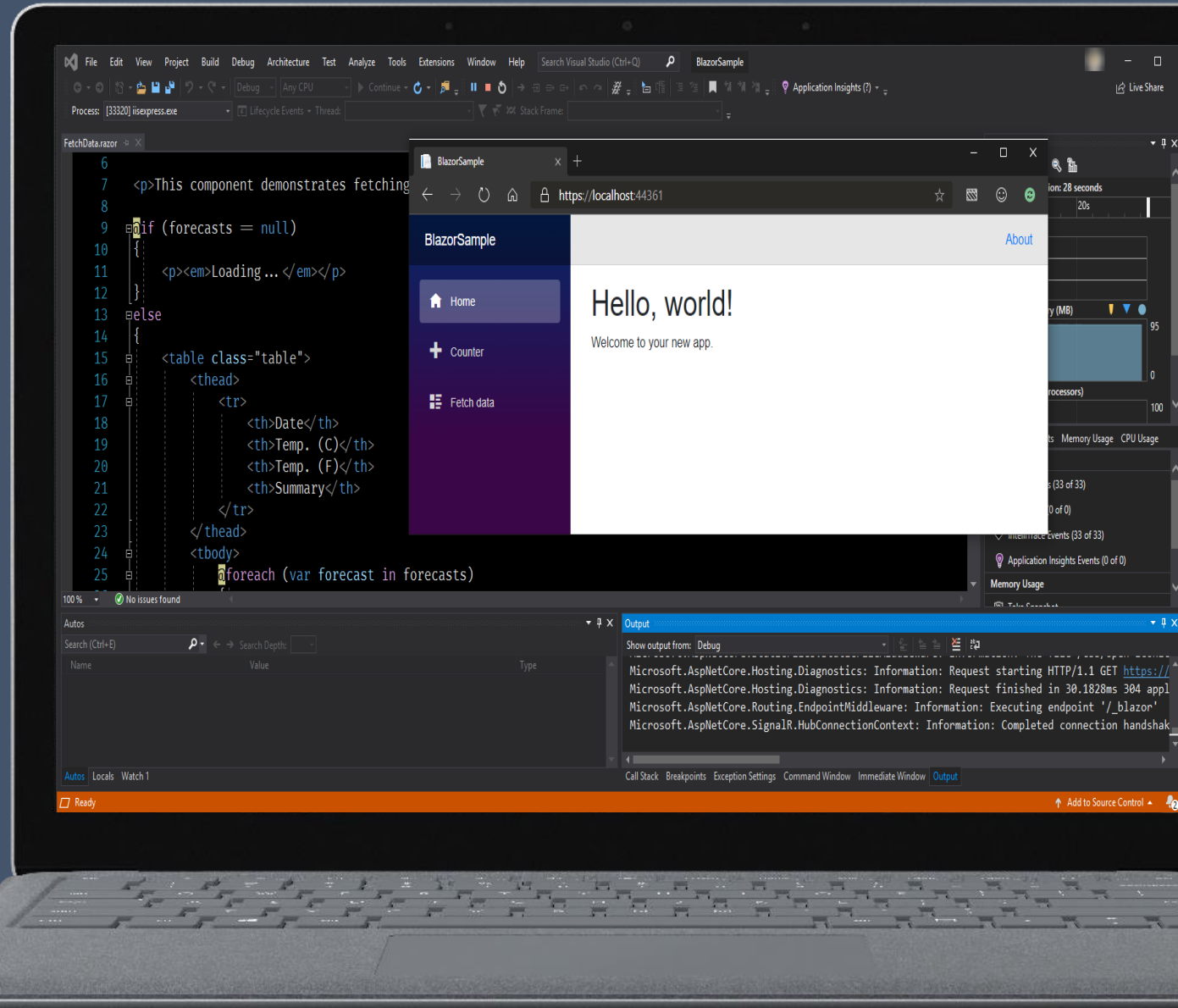


What's new in ASP Core

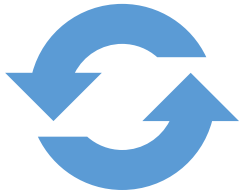
gRPC

Worker Service

Blazor



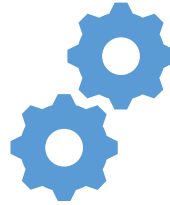
ASP.NET Core 3.0



gRPC

High performance contract-based
RPC services with .NET

Works across many languages and
platforms



Worker service

Starting point for long running back
processes like Windows Server or
Linux daemon

Producing or consuming messages
from a message queue



Web API's + identity

Add security and authentication to
Web API's



What is gRPC?

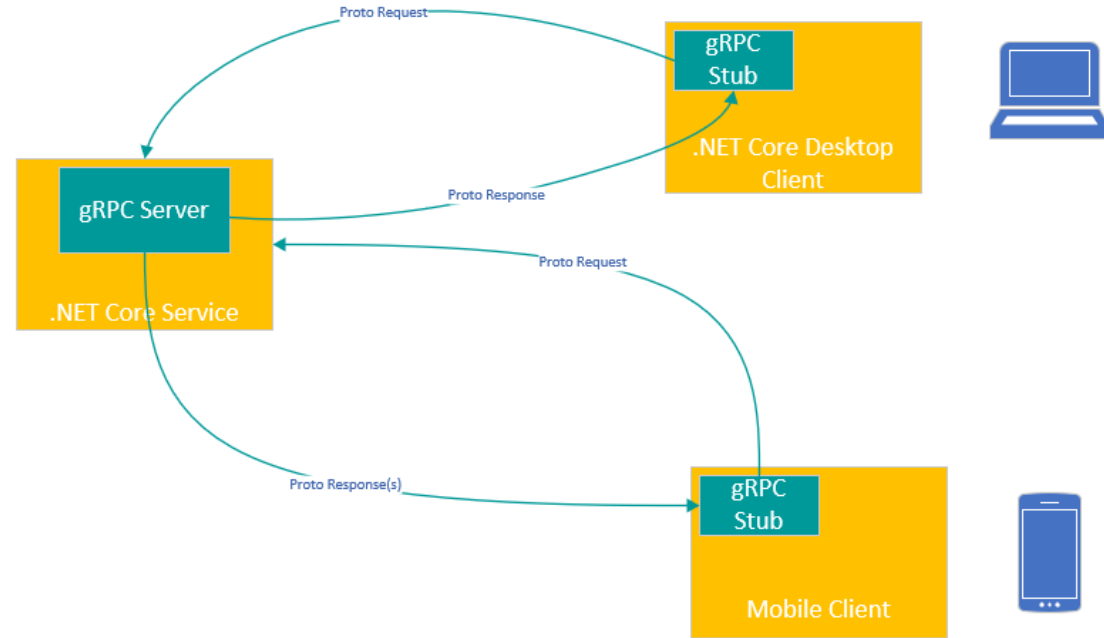
- gRPC stands for **gRPC Remote Procedure Calls**
- High Performance, highly scalable, standards based, open source general purpose RPC Framework
- Binary Data Representation (compact)
- Takes advantage of the HTTP/2 feature set
- Endpoints are contract based so clients know what to expect
- Bi-directional Client/Server Message Streaming

Getting Started

.proto file used to define:

1. Service endpoints
2. Request Message Format
3. Response Message Format

Client and Server can use .proto files to generate code



```
syntax = "proto3";
```

```
service Greeter {  
  rpc SayHello (HelloRequest) returns (HelloReply) {}  
}
```

```
message HelloRequest {  
  string name = 1;  
}
```

```
message HelloReply {  
  string message = 1;  
}
```

.proto file

Worker Service

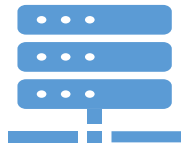
Starting point for long running background processes like a Windows Service or Linux daemon

- Lightweight, small – only the services that you need
- Producing or consuming messages from a message queue
- Microservices

You get all the nice ASP Core services

- Dependency Injection
- Rich Configuration Options
- Logging

ASP.NET Core 3.0 Blazor



Full stack web

development with C#

You don't need to know AngularJS,
React, Vue, etc.

Take advantage of stability and
consistency of .NET



Runs in all browsers

Strongly typed on the client and
server

Share C# code with the client and
server



Web assembly

(optional and in preview)

Native performance

Requires no plugin or code
transpilation

