

# An introduction to .NET Core and ASP.NET Core

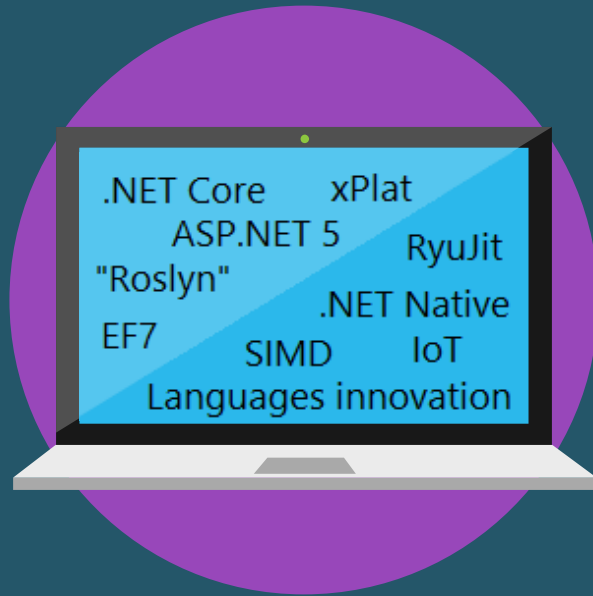
Gill Cleeren

@gillcleeren – [www.snowball.be](http://www.snowball.be)

# Agenda

- Hello .NET Core 3
- Installing .NET Core 3
- Creating an app with .NET Core
  - Console app
- Moving to ASP.NET Core 3
  - An introduction to ASP.NET Core
- ASP.NET Core WebAPI

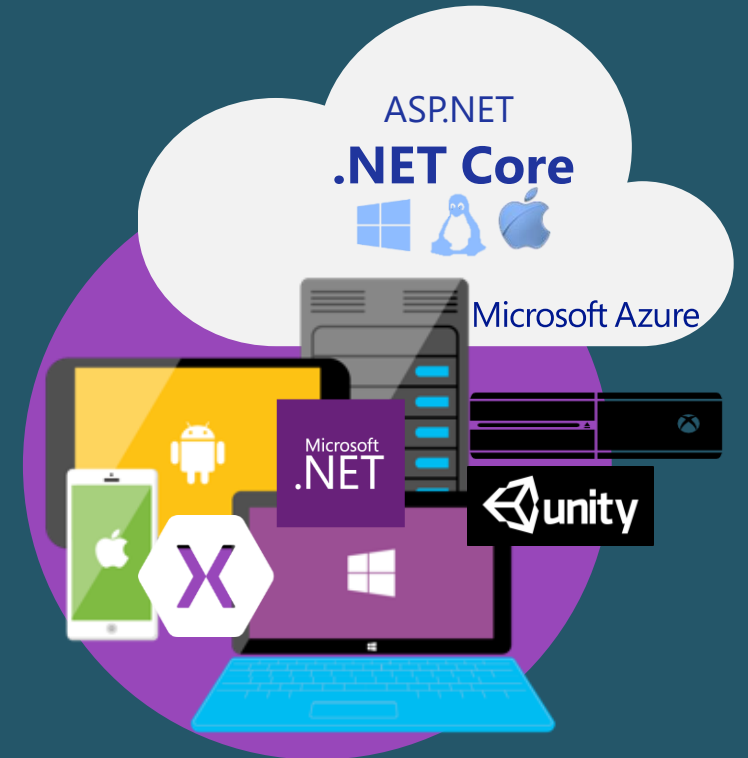




Innovation



Openness



Any app, any  
platform

Hello .NET Core 3

# .NET Core

*“.NET Core is a general purpose development platform maintained by Microsoft and the .NET community on GitHub. It is cross-platform, supporting Windows, macOS and Linux, and can be used in device, cloud, and embedded/IoT scenarios.”*

*source: <https://docs.microsoft.com/en-us/dotnet/articles/core>*

# What is .NET Core?

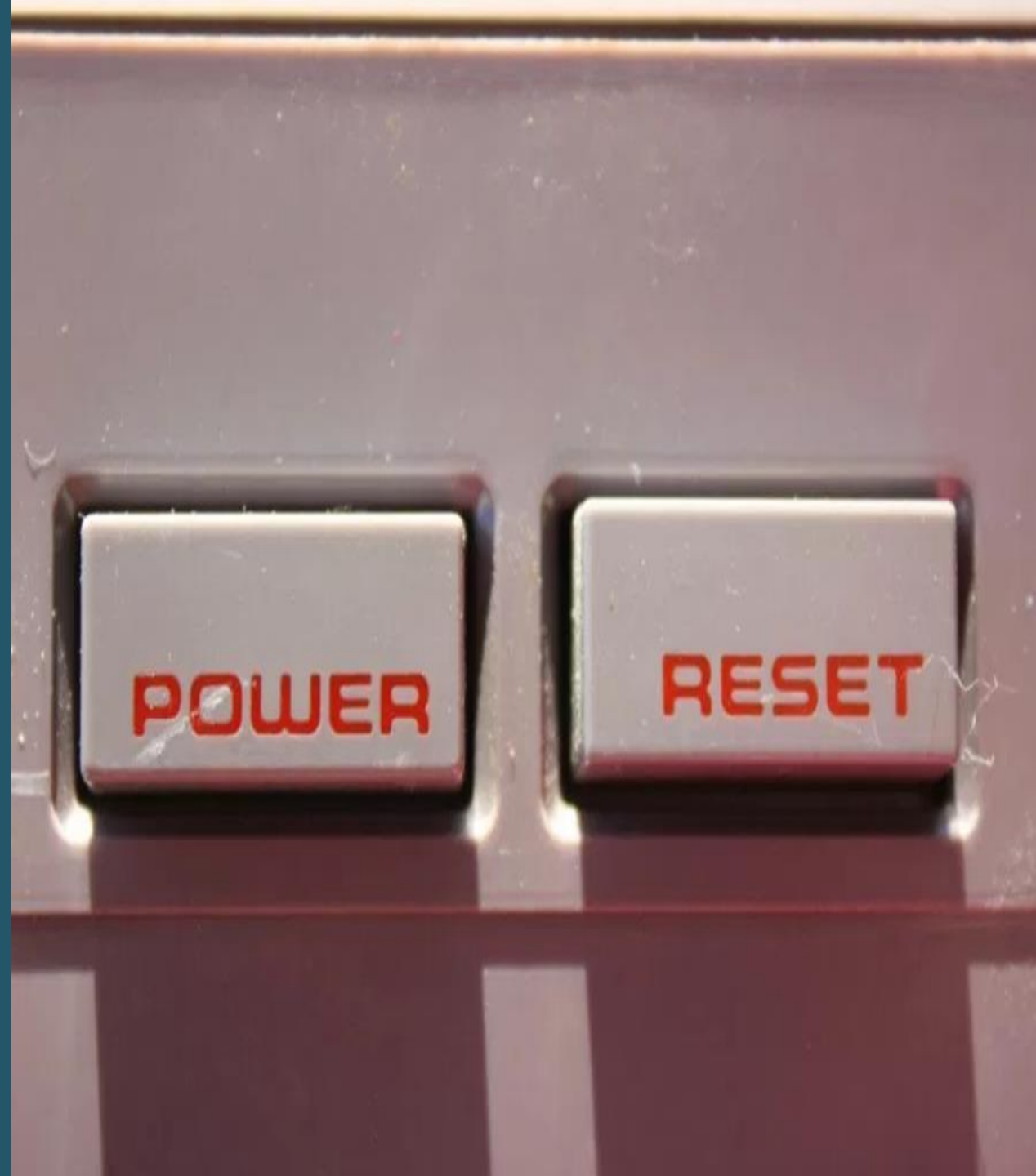
- **Flexible deployment**
  - Can be included in your app or installed side-by-side user- or machine-wide
- **Cross-platform**
  - Runs on Windows, macOS and Linux; can be ported to other OSs
  - The supported Operating Systems (OS), CPUs and application scenarios will grow over time, provided by Microsoft, other companies, and individuals
- **Command-line tools**
  - All product scenarios can be exercised at the command-line

# What is .NET Core?

- **Compatible**
  - .NET Core is compatible with .NET Framework, Xamarin and Mono, via the .NET Standard Library
- **Open source**
  - The .NET Core platform is open source, using MIT and Apache 2 licenses
  - .NET Core is a .NET Foundation project
- **Supported by Microsoft**
  - .NET Core is supported by Microsoft, per .NET Core Support

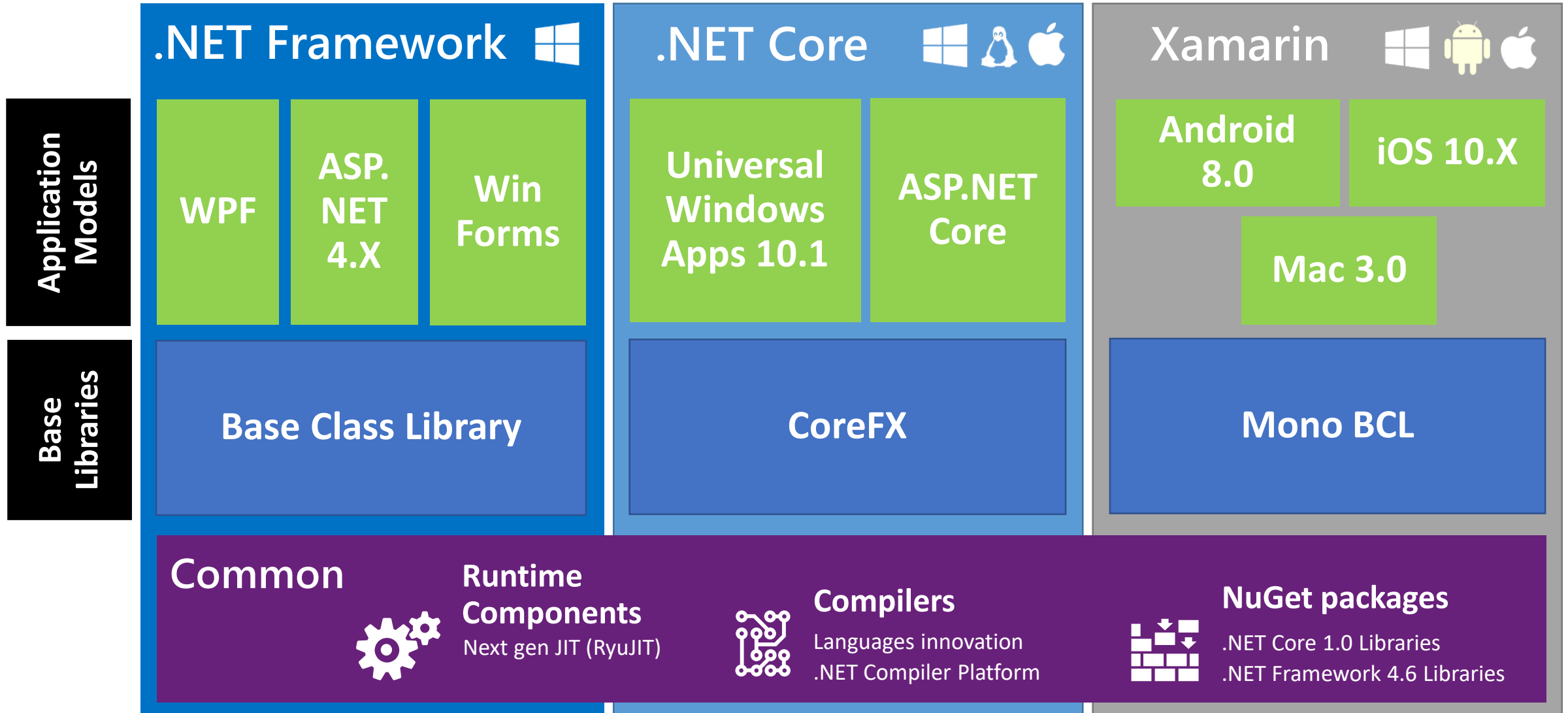
# Why a reboot?

- .NET Framework was too monolithic
- Too many dependencies (preventing cross-platform)
- Maintaining compatibility slowed down Framework development
- Removed dependency on OS
- Faster startup
- New runtime next to full framework (maintaining compatibility)





# The big picture of .NET Platforms



# .NET Core

- **Runtimes**

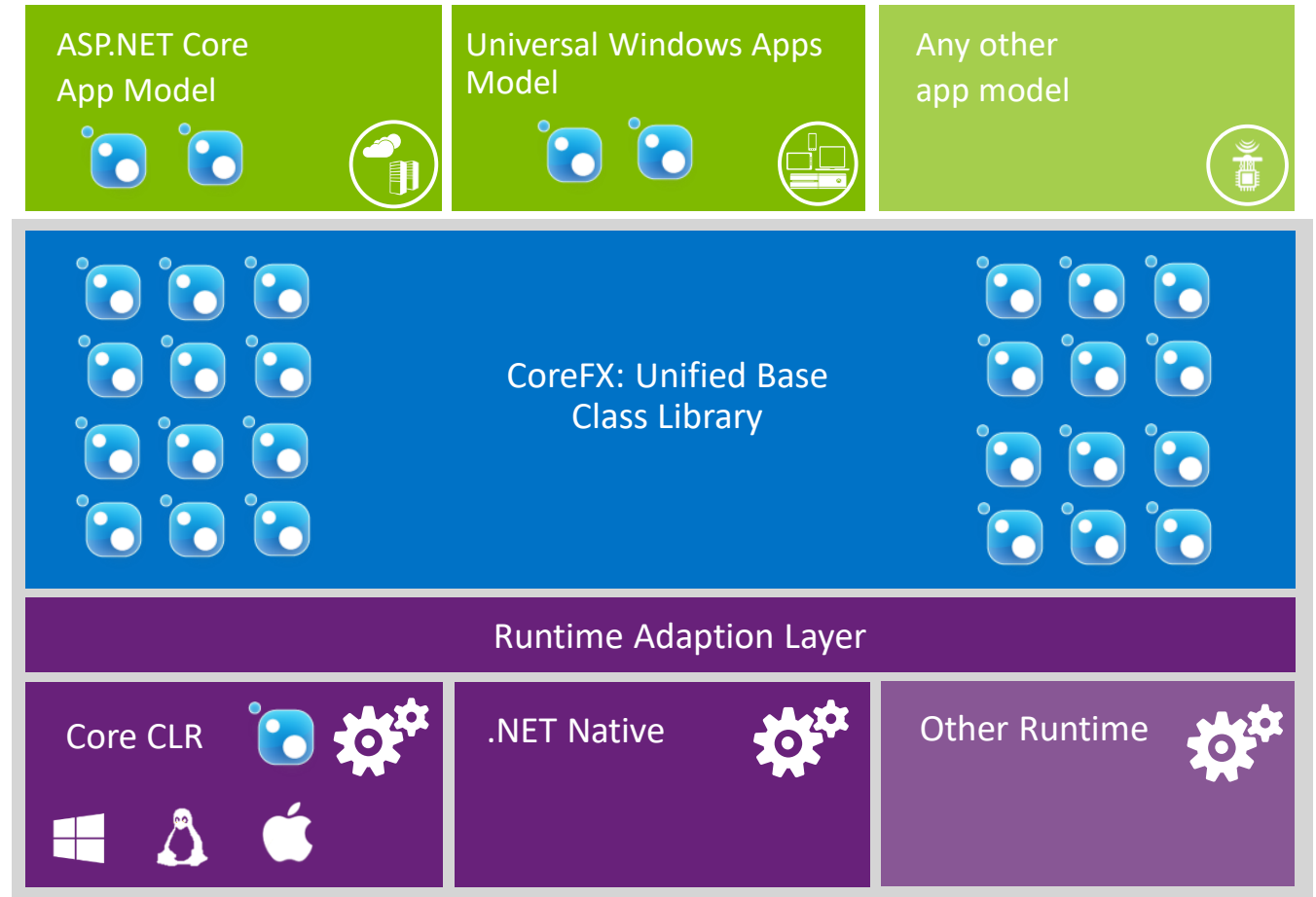
- Cross-platform implementations of CoreCLR
- CoreRT for AOT scenarios

- **CoreFX libraries**

- Standards based BCL
- Platform and OS abstractions
- Delivered in NuGet packages
- Shared Framework

- **SDK**

- Command-line tools
- Side by side installation
- C# and F#
- Driver dotnet.exe



# The [netcore] building blocks

- CoreFX
  - .NET Core Framework
- CoreCLI
  - Command Line Interface
- CoreCLR
  - Common Language Runtime
- CoreRT
  - Native runtime



# CoreFX goals

- Abstract the platform
  - IO will be handled for you, no matter what the platform (Mac, Linux...)
- Authentically .NET
  - BCL... It's all the same, you can reuse what you know
- Open source
  - Good for enterprises
  - Security access
  - Even debugging to GitHub works
- Support NSL
  - .NET Standard Library
  - “The Holy Grail”





# Getting to the core

The .NET Framework

NS

NS

NS

NS

NS

NS

NS

NS

OS specific

Common Logic

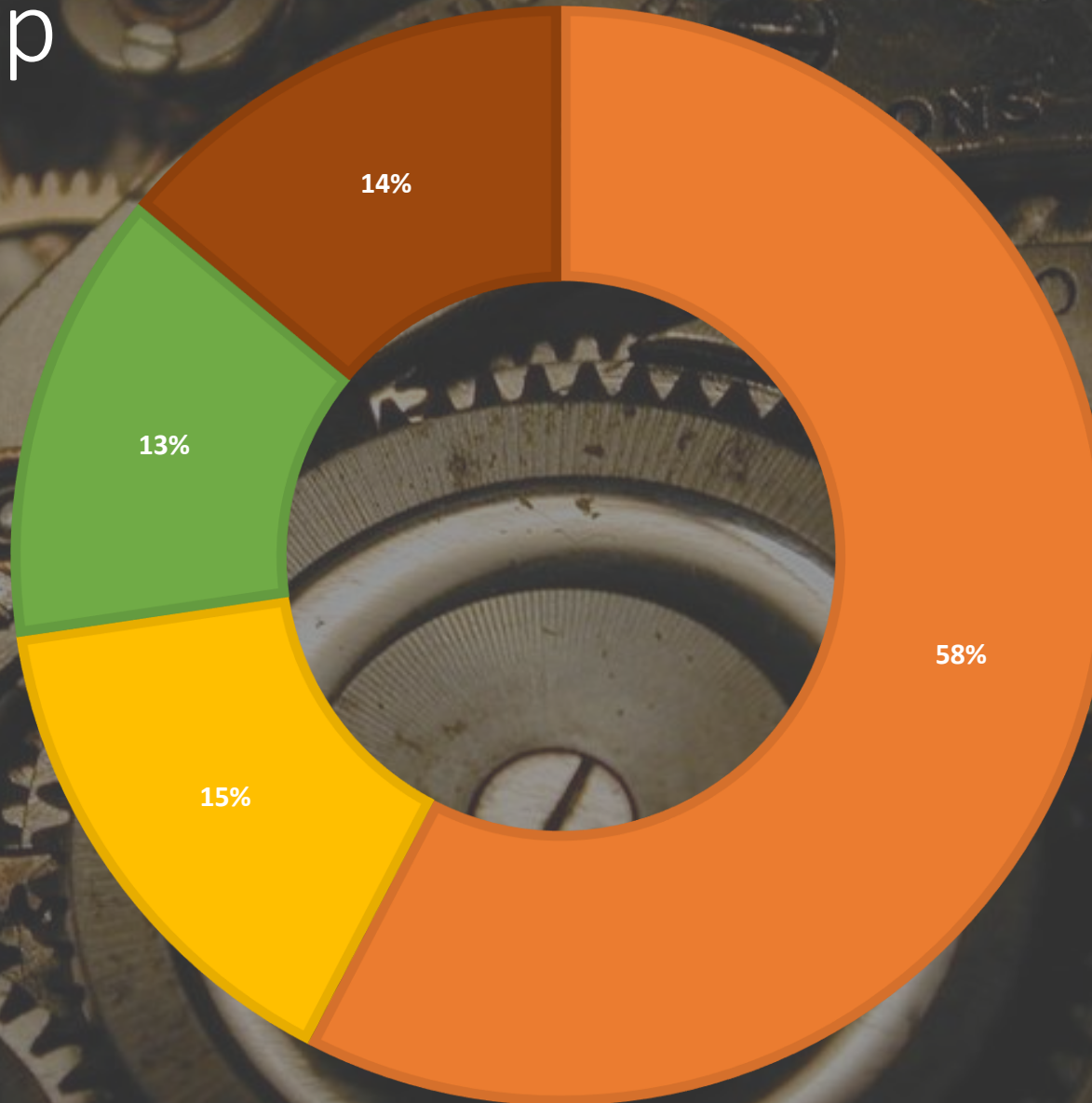
Windows only

.NET Core

A specification and implementation

# System.Net.Http

■ Shared ■ Windows ■ MacOS ■ Linux





A detailed close-up of a mechanical watch movement, showing various gears, screws, and metal plates. The image has a warm, golden-brown color palette. In the upper right, a metal plate is engraved with the text "ADJUSTED TO TEMPERATURE & POSITIONS". Below this, the number "3955098" is visible. The text "Will everything work?" is overlaid in white in the upper left area.

Will everything work?

PlatformNotSupportedException

# Packages and more packages

## NuGet Packages

Building blocks of .NET Core

## Metapackages


Describes a set of packages meaningful and tested together


















Easier than referencing individual packages

**NETStandard.Library**

**Microsoft.NETCore.App**

**Microsoft.NETCore.Portable.Compatibility** [.](#)

 NETStandard.Library (1.6.0)

-  Microsoft.NETCore.Platforms (1.0.1)
- ▷  Microsoft.Win32.Primitives (4.0.1)
- ▷  System.AppContext (4.1.0)
- ▷  System.Collections (4.0.11)
- ▷  System.Collections.Concurrent (4.0.12)
- ▷  System.Console (4.0.0)
- ▷  System.Diagnostics.Debug (4.0.11)
- ▷  System.Diagnostics.Tools (4.0.1)
- ▷  System.Diagnostics.Tracing (4.1.0)
- ▷  System.Globalization (4.0.11)
- ▷  System.Globalization.Calendars (4.0.1)
- ▷  System.IO (4.1.0)
- ▷  System.IO.Compression (4.1.0)
- ▷  System.IO.Compression.ZipFile (4.0.1)
- ▷  System.IO.FileSystem (4.0.1)
- ▷  System.IO.FileSystem.Primitives (4.0.1)
- ▷  System.Linq (4.1.0)





Portable Class Libraries    .NET Standard Library  
Machine generated    Human generated

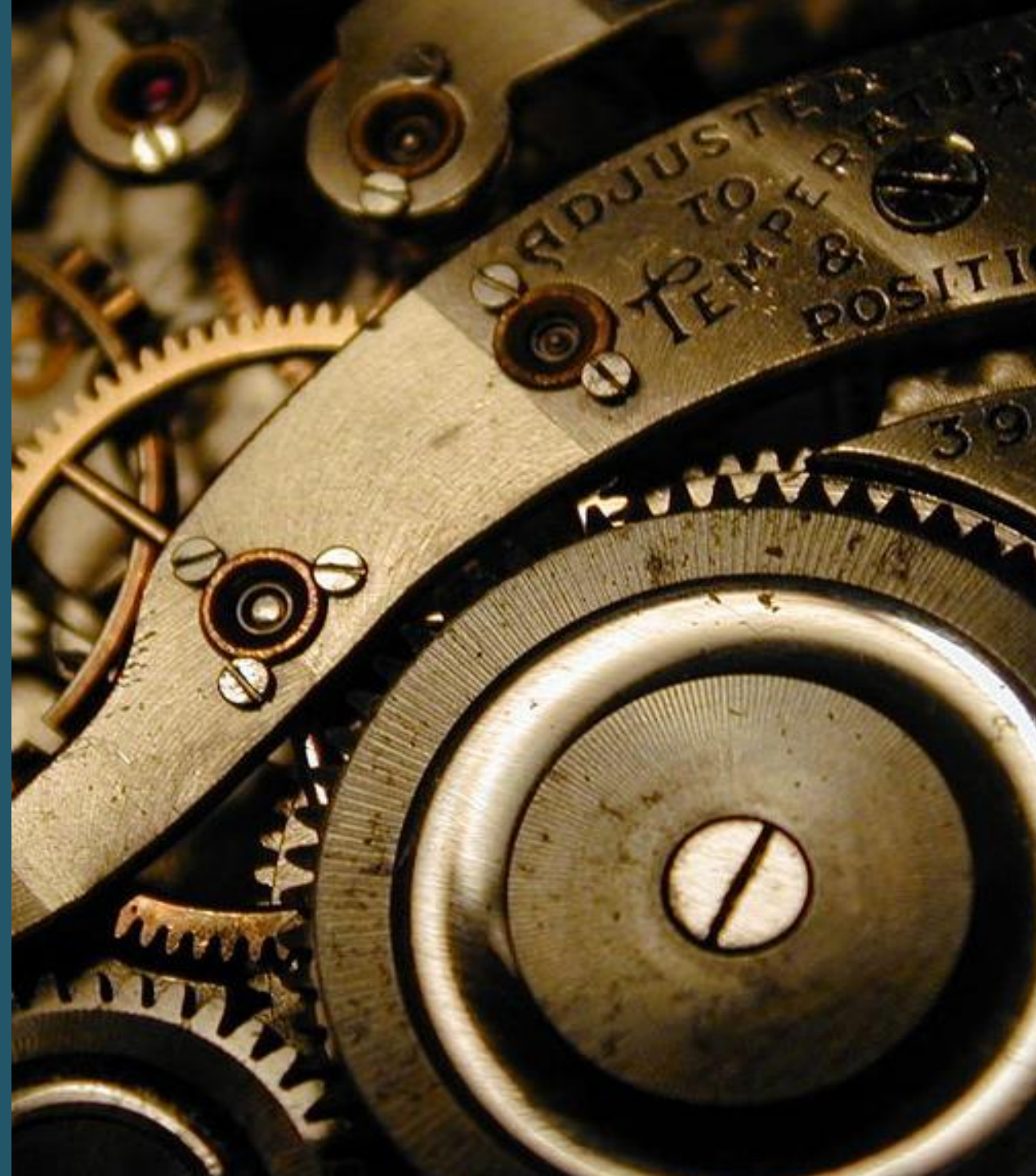


# .NET Standard is the new black!

PCLs shouldn't be used anymore

Target 'least common denominator' API for platforms

Not a sustainable model



# .NET Standard in a nutshell

- .NET Standard is a specification
- Represents a set of APIs that all .NET platforms have to implement
- .NET Core is an implementation of the .NET Standard

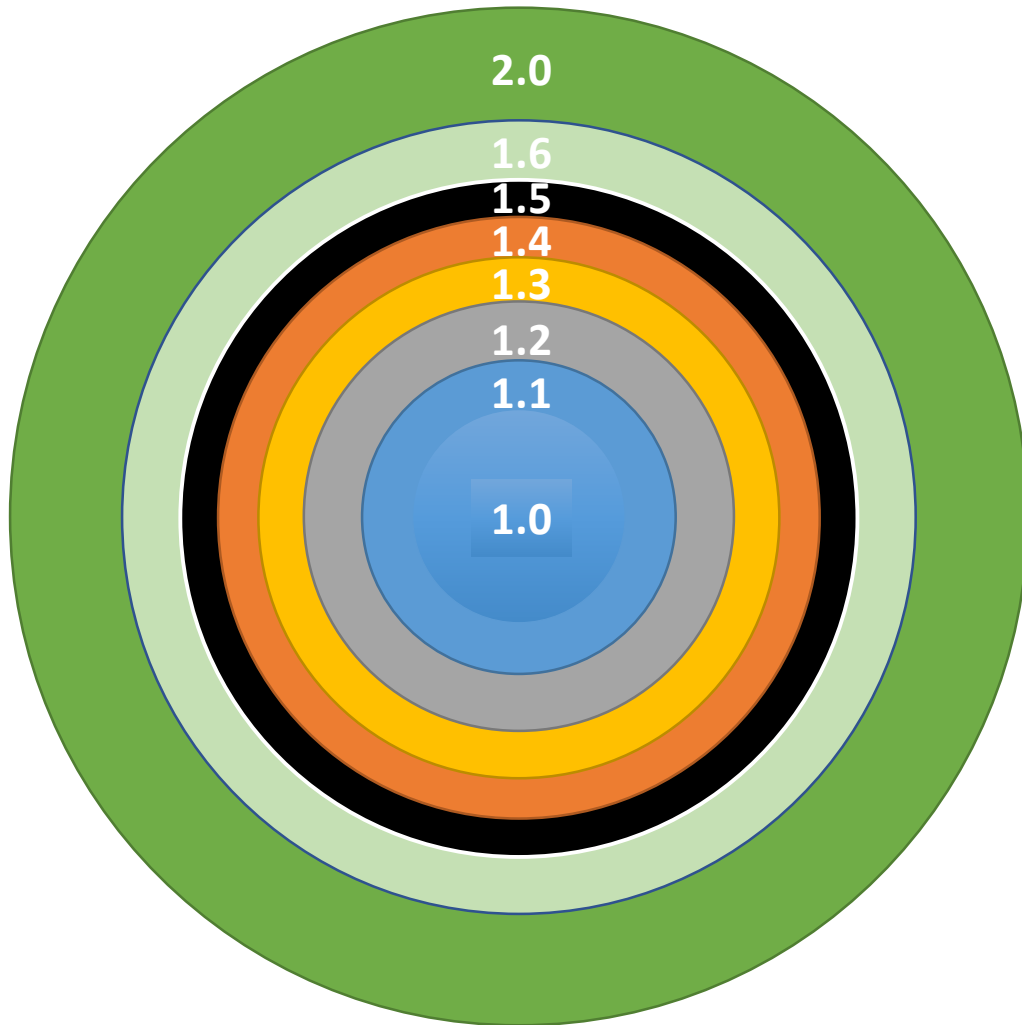
```
public partial class Object
{
    public Object() { }
    public virtual bool Equals(System.Object obj) { throw null; }
    public static bool Equals(System.Object objA, System.Object objB) { throw null; }
    ~Object() { }
    public virtual int GetHashCode() { throw null; }
    public System.Type GetType() { throw null; }
    protected System.Object MemberwiseClone() { throw null; }
    public static bool ReferenceEquals(System.Object objA, System.Object objB) { throw null; }
    public virtual string ToString() { throw null; }
}
```





[Github .NET Standard](#)

# Versioning in .NET Standard



- Higher versions incorporate all APIs from previous versions
- Concrete .NET platforms implement a specific version of .NET Standard

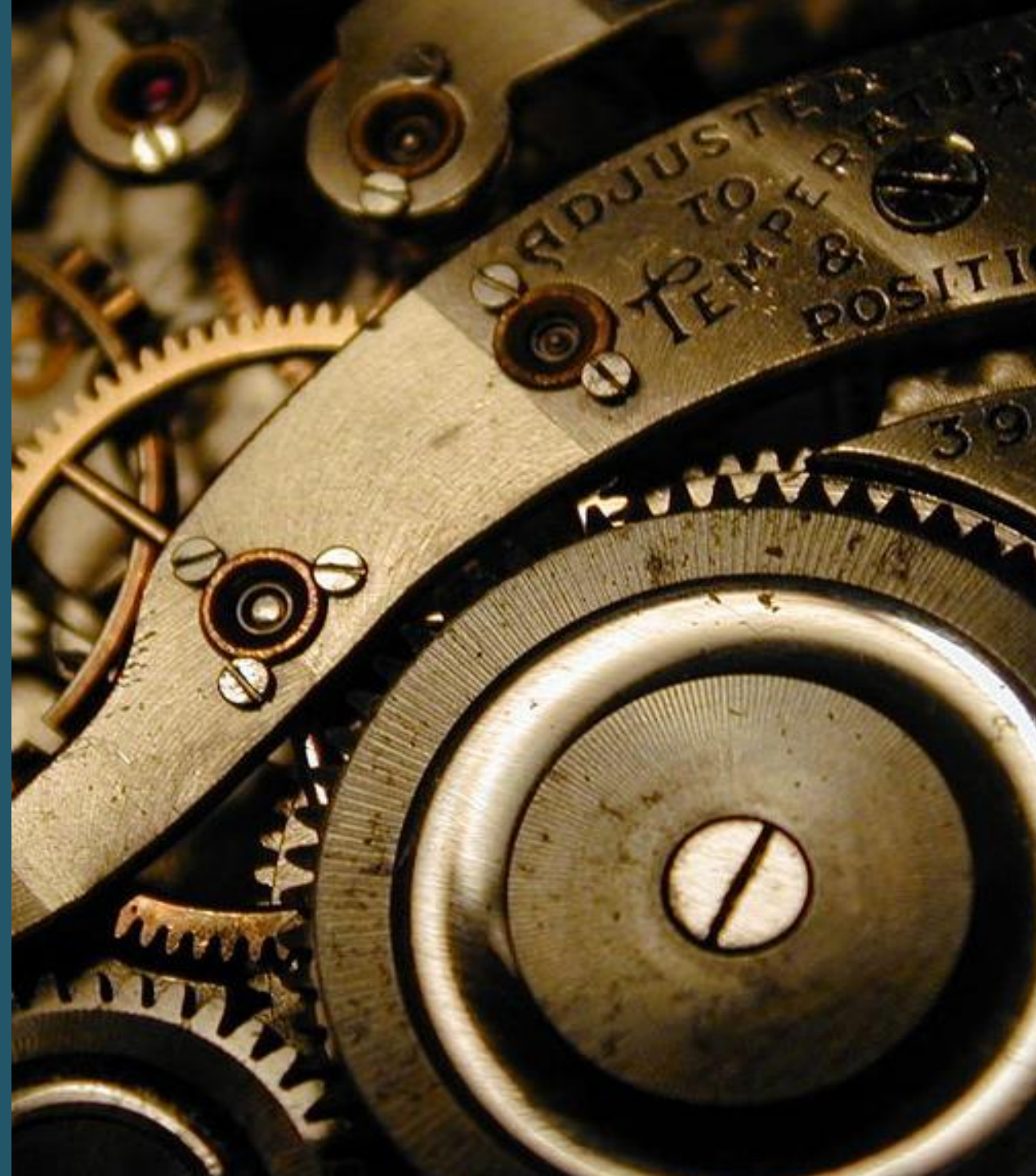


# .NET Standard

A version of the .NET Standard is a definition of a collection of standard Api's

A runtime can support one or more versions of the .NET Standard

A library can target one or more versions of the .NET Standard



# .NET Standard: versions

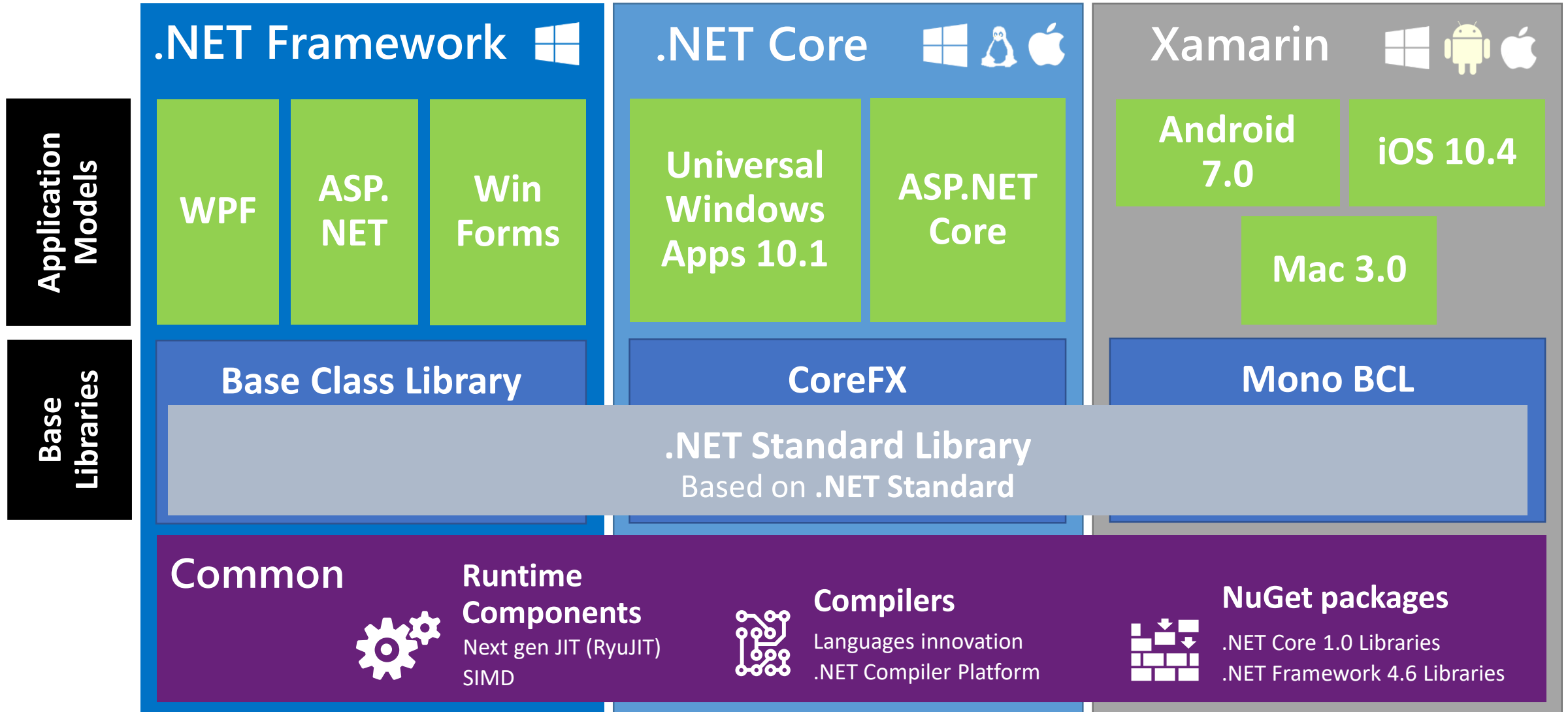
.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0
.NET Framework (with .NET Core 1.x SDK)	4.5	4.5	4.5.1	4.6	4.6.1	4.6.2		
.NET Framework (with .NET Core 2.0 SDK)	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1	4.6.1	4.6.1
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4
Xamarin.iOS	10.0	10.0	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	7.5
Universal Windows Platform	10.0	10.0	10.0	10.0	10.0	vNext	vNext	vNext
Windows	8.0	8.0	8.1					
Windows Phone	8.1	8.1	8.1					
Windows Phone Silverlight	8.0							

# .NET Standard 2.1

.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 <sup>1</sup>	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1 <sup>2</sup>	4.6.1 <sup>2</sup>	4.6.1 <sup>2</sup>	N/A <sup>3</sup>
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



# Revisiting .NET Platforms



# .NET Standard 2.0

Introduces more APIs

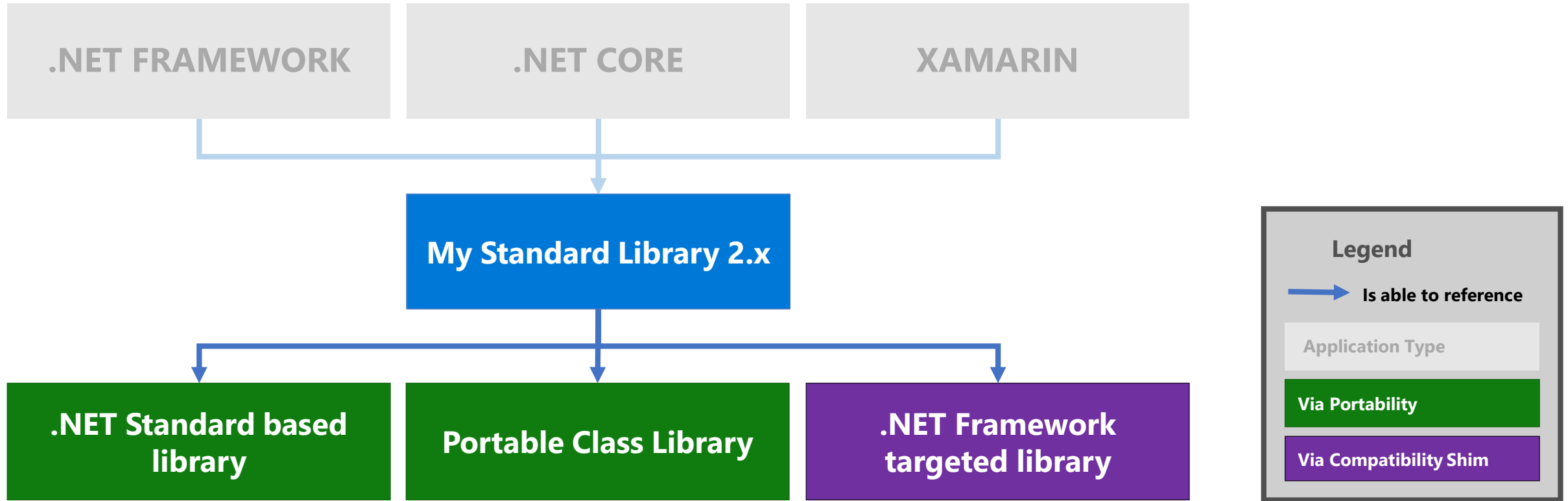
Version	#APIs	Growth %
1.x	13,501	+1%
2.0	32,638	+142%

Creates compatibility with .NET Framework libraries

Compatibility shim bridges .NET Standard based libraries to .NET FX libraries (mscorlib)

<b>XML</b>	XLinq • XML Document • XPath • Schema • XSL
<b>SERIALIZATION</b>	BinaryFormatter • Data Contract • XML
<b>DATA</b>	Abstractions • Provider Model • DataSet
<b>NETWORKING</b>	Sockets • Http • Mail • WebSockets
<b>IO</b>	Files • Compression • MMF
<b>THREADING</b>	Threads • Thread Pool • Tasks
<b>CORE</b>	Primitives • Collections • Reflection • Interop • Linq

# Referencing libraries from .NET Standard



# Breaking changes

- No Global Assembly Cache
  - All assemblies are deployed privately
  - An assembly for each namespace (no more mscorlib.dll)
- No application models
  - WPF
  - Winforms
  - WebForms
- Anything Windows-specific
  - registry, ACLs, perf counters, etc.
- No AppDomains
  - Infrastructure exists but it is no longer usable in terms of API
  - New ApplicationContext object
  - AssemblyLoadContext to dynamically load assemblies
  - Container to isolate code
- No Remoting
- No Binary Serialization
- No Code Access Security (CAS)

# Should I migrate my project?

- Cannot:
  - WinForms, unless UWP
  - ASP.NET WebForms
- Can:
  - WPF and WinForms: with Microsoft.Windows.Compatibility package
- Should (absolutely):
  - General purpose libraries
- Should (maybe):
  - ASP.NET MVC
  - Micro-services
  - Console apps
  - Rewrite

.NET Core installation

# .NET Core installation

Dot.net or [www.microsoft.com/net](http://www.microsoft.com/net)



## .NET Framework

The .NET framework helps you create mobile, desktop, and web applications that run on Windows PCs, devices and servers.



## .NET Core

.NET Core gives you a blazing fast and modular platform for creating server applications that run on Windows, Linux and Mac.

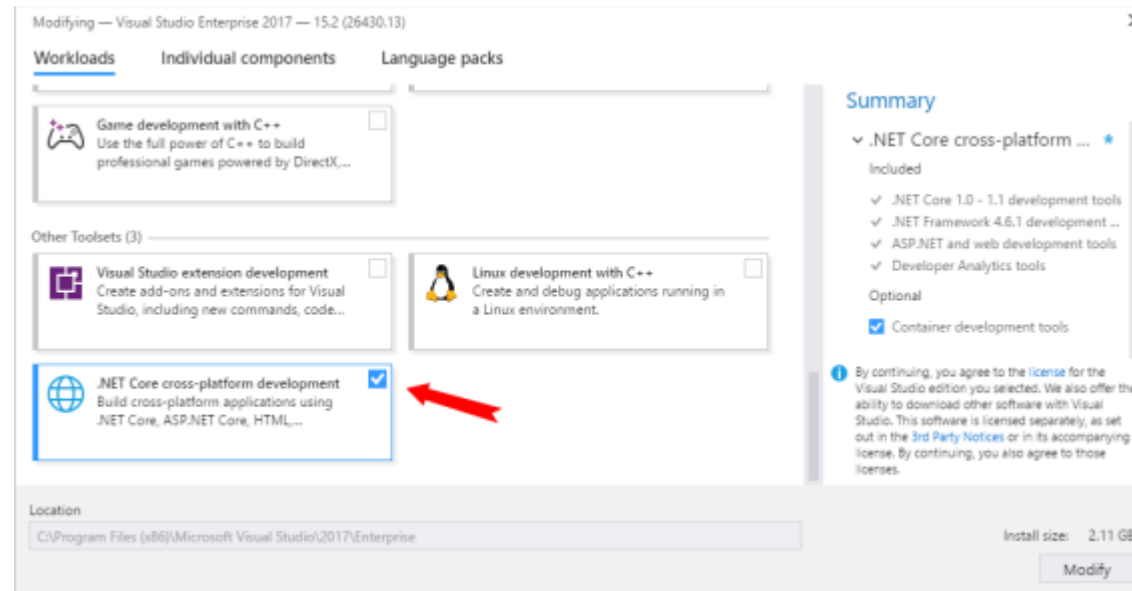


## Xamarin

Xamarin brings the power and productivity of .NET to iOS and Android, reusing skills and code while getting access to the native APIs and performance.

# .NET Core installation

Or from Visual Studio





# Open source

dotnet / cli

Watch

417

Star

2,195

Fork

729

<> Code

Issues 350

Pull requests 20

Projects 4

Wiki

Insights

This repo contains the .NET Core command-line (CLI) tools, used for building .NET Core apps and libraries through your development flow (compiling, NuGet package management, running, testing, ...). <https://github.com/dotnet/core>

7,019 commits

24 branches

21 releases

183 contributors

MIT

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

wli3 committed with wli3 Pass Deb repo secret to the repoapi script ...

Latest commit 414585a 23 hours ago

Documentation	Merge pull request #7225 from GuardRex/guardrex/first-run-info-update	23 hours ago
TestAssets	Use latest command line parser	a month ago
build	Pass Deb repo secret to the repoapi script	2 hours ago
build_projects	Honor UI language override in test runs	21 days ago
packaging	Merge pull request #7225 from GuardRex/guardrex/first-run-info-update	23 hours ago
resources	Return non-zero exit code for test failure in multitargeted test proj...	8 months ago
scripts	Passwords/keys should not be passed in the environment via a docker f...	13 days ago
src	Merge pull request #7225 from GuardRex/guardrex/first-run-info-update	23 hours ago
test	Merge branch 'master' into merges/release/2.0.0-to-master-20170731-07...	8 days ago
tools	Merge branch 'rel/1.1.0' into merge_rel_110	2 months ago
.gitattributes	Add props and targets to text=auto	2 months ago
.gitignore	Add .binlog (MSBuild binary log format) extension to .gitignore	3 months ago
CONTRIBUTING.md	merge master	5 months ago



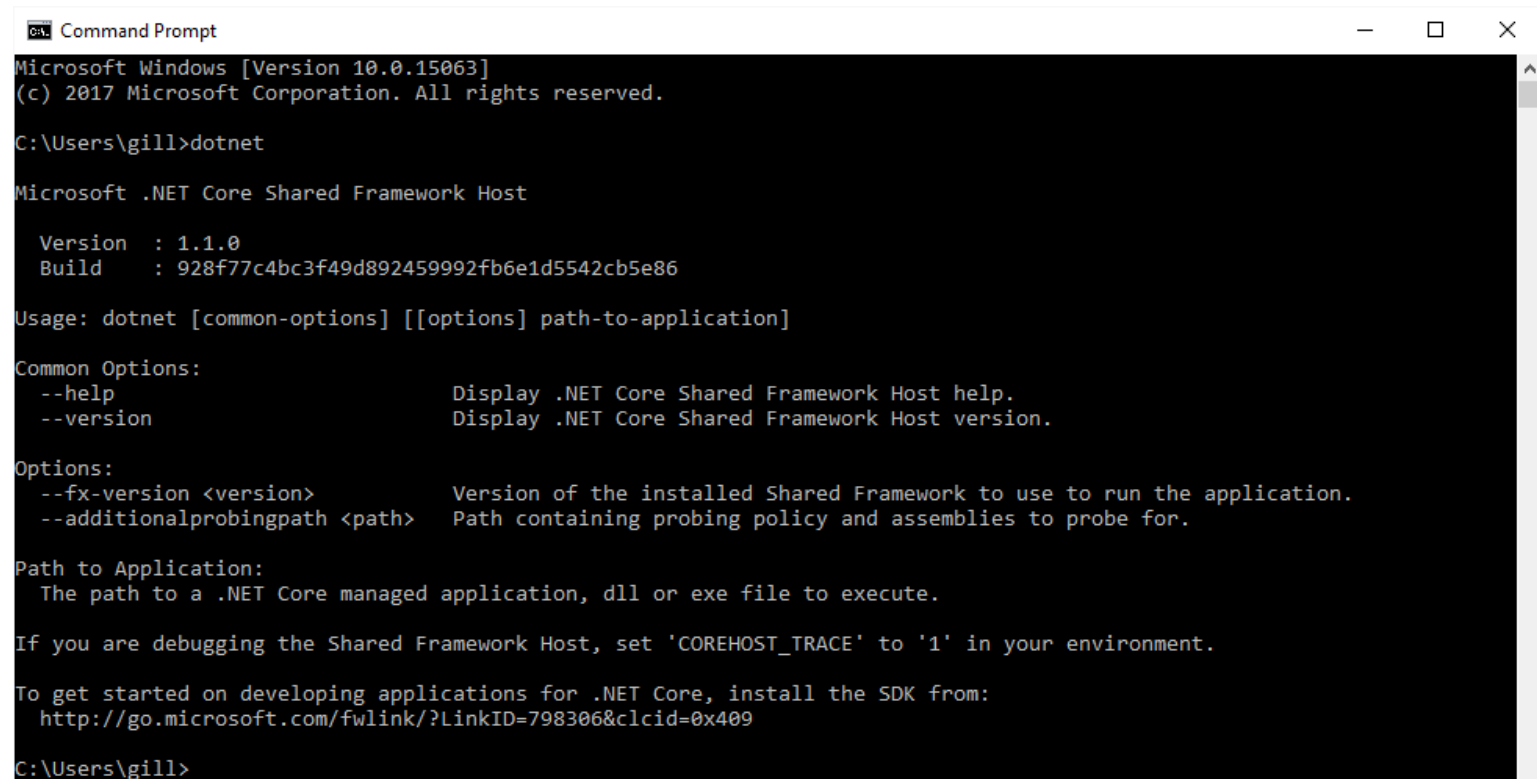
# DEMO

Let's go to [Dot.net](https://dot.net)

Creating an app with .NET Core

# dotnet

- Open console window
- Type dotnet
  - Driver of .NET Core CLI
  - Note that first run will take some time to complete



```
Command Prompt
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\gill>dotnet

Microsoft .NET Core Shared Framework Host

  Version : 1.1.0
  Build   : 928f77c4bc3f49d892459992fb6e1d5542cb5e86

Usage: dotnet [common-options] [[options] path-to-application]

Common Options:
  --help           Display .NET Core Shared Framework Host help.
  --version        Display .NET Core Shared Framework Host version.

Options:
  --fx-version <version>  Version of the installed Shared Framework to use to run the application.
  --additionalprobingpath <path>  Path containing probing policy and assemblies to probe for.

Path to Application:
  The path to a .NET Core managed application, dll or exe file to execute.

If you are debugging the Shared Framework Host, set 'COREHOST_TRACE' to '1' in your environment.

To get started on developing applications for .NET Core, install the SDK from:
  http://go.microsoft.com/fwlink/?LinkID=798306&clcid=0x409

C:\Users\gill>
```

# Command overview

- **new**: new project based on template
- **restore**: restore dependencies
- **build**: build 😊
- **publish**: packs the application and dependencies
- **run**: run
- **test**: runs tests
- **vstest**: runs specific tests
- **pack**: create NuGet package
- **migrate**: version update
- **clean**: like VS
- **sln**: modify parameters in an sln file

Template description	Template name	Languages
Console application	console	[C#], F#
Class library	classlib	[C#], F#
Unit test project	mstest	[C#], F#
xUnit test project	xunit	[C#], F#
ASP.NET Core empty	web	[C#]
ASP.NET Core web app	mvc	[C#], F#
ASP.NET Core web api	webapi	[C#]
Nuget config	nugetconfig	
Web config	webconfig	
Solution file	sln	

# CLI goals

Getting started files

Restore dependencies

Publish app packages

Create NuGet packages

Remember, everything is a package

```
C:\Users\gill>dotnet -h
.NET Command Line Tools (2.0.0)
Usage: dotnet [runtime-options] [path-to-application]
Usage: dotnet [sdk-options] [command] [argument]

path-to-application:
    The path to an application .dll file to execute

SDK commands:
    new                Initialize .NET projects.
    restore            Restore dependencies specified in the project file.
    run               Compiles and immediately executes the application.
    build             Builds a .NET project.
    publish           Publishes a .NET project for distribution.
    test              Runs unit tests using the test runner.
    pack              Creates a NuGet package.
    migrate           Migrates a project.json based project to a csproj.
    clean             Clean build output(s).
    sln              Modify solution (SLN) files.
    add              Add reference to the project.
    remove           Remove reference from the project.
    list             List reference in the project.
    nuget            Provides additional NuGet commands.
    msbuild           Runs Microsoft Build Engine.
    vstest            Runs Microsoft Test Execution Engine.
```

# Creating a new .NET Core Console app

- Start from dotnet new
  - Requires type of application to create
  - Will create in the specific directory
- Example: dotnet new console

```
Directory of C:\Code\core\test
08/08/2017  22:43    <DIR>          .
08/08/2017  22:43    <DIR>          ..
08/08/2017  22:43                186 Program.cs
08/08/2017  22:43                178 test.csproj
                2 File(s)              364 bytes
                2 Dir(s)  531,658,104,832 bytes free

C:\Code\core\test>
```

- Example: dotnet new console -lang C# -n "HelloDotNetCore"

# Creating a class library

- Requires different type parameter
  - `dotnet new classlib -n "CustomLibrary" -lang C#`

```
Directory of C:\Code\core\CustomLibrary
08/08/2017  22:45    <DIR>          .
08/08/2017  22:45    <DIR>          ..
08/08/2017  22:45                90 Class1.cs
08/08/2017  22:45               145 CustomLibrary.csproj
                2 File(s)                235 bytes
                2 Dir(s)  531,658,395,648 bytes free

C:\Code\core\CustomLibrary>
```



# Creating a solution

- Creating the solution
  - `dotnet new sln -n "Demo Solution"`
- Adding projects to the solution
  - `dotnet sln <SolutionName> add <ProjectName>`



# Demo

Using dotnet

Creating a solution and adding projects

# Working with dependencies

- .NET Core works with NuGet for dependencies
  - dotnet restore
- Will restore the packages you have in the project file

```
C:\Code\core\test>dotnet restore
Restoring packages for C:\Code\core\test\test.csproj...
Generating MSBuild file C:\Code\core\test\obj\test.csproj.nuget.g.props.
Generating MSBuild file C:\Code\core\test\obj\test.csproj.nuget.g.targets.
Writing lock file to disk. Path: C:\Code\core\test\obj\project.assets.json
Restore completed in 1.9 sec for C:\Code\core\test\test.csproj.

NuGet Config files used:
  C:\Users\gill\AppData\Roaming\NuGet\NuGet.Config
  C:\Program Files (x86)\NuGet\Config\Microsoft.VisualStudio.Offline.config

Feeds used:
  https://api.nuget.org/v3/index.json
  https://etcrd.pkgs.visualstudio.com/_packaging/ETCRD/nuget/v3/index.json
  C:\Dropbox\Work\Courses\Update to 2017\Demos\VS2017\NuGet Package store
  C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\
```

# Building an application

- Building is done using dotnet build
  - Will by default build the csproj(s) in the directory you're working with
  - You can also specify the sln or csproj you want to build
    - dotnet build "Demo App"
    - dotnet build "Demo App\Demo App.csproj"
- Building requires that you first have performed a restore!

```
C:\Code\core\test>dotnet build
Microsoft (R) Build Engine version 15.1.1012.6693
Copyright (C) Microsoft Corporation. All rights reserved.

test -> C:\Code\core\test\bin\Debug\netcoreapp1.1\test.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:01.98
C:\Code\core\test>
```

# Building an application

- Default build will be a debug build
  - Can be changed using the --configuration parameter
    - `dotnet build --configuration Release`
    - `dotnet build "Demo App" --configuration Release`
- You can also target a different runtime as your target
  - `dotnet clean`
  - `dotnet restore --runtime rhel.7-x64`
  - `dotnet build --runtime rhel.7-x64`
    - Will build for specific “os”
    - First we need to clean, then bring in the dependencies and finally, we build
    - This flow will always be required when we are changing the target
      - Arrives in bin folder

# Running a .NET core app

- Running is done using dotnet run
  - Will perform a build if needed

```
Time Elapsed 00:00:01.80  
C:\Code\core\test>dotnet run  
Hello World!  
C:\Code\core\test>
```

# Publishing a .NET Core app

- Once we have a working app, we need to deploy (or “publish”) it
  - Can be done using `dotnet publish <Project> -c Release`
- 2 different deployment options exist
  - Self-contained Deployment (SCD):
    - does not depend on any .NET Core version installed on the system
    - Contains all that’s needed (.NET Core libraries and runtime) + your application
  - Framework-dependent Deployment (FDD)
    - Requires that a system-wide .NET Core version is installed on the machine



# WARNING

**Pack and Publish aren't the same!**

Pack creates the NuGet package  
Publish prepares the app bundle



# Working with FDD

- This is the default when we publish an application
- Will deploy only app + dependencies
- Uses .NET Core version on the system
- Will ALWAYS create a DLL (no exe)
- Application package size is smaller
- Less disk space is used since it uses the system-wide version
- Builds into Portable Executable
  - Can be run without specifying the OS (note that it's built for an OS!)



# DEMO

Creating an FDD

# Working with SCD

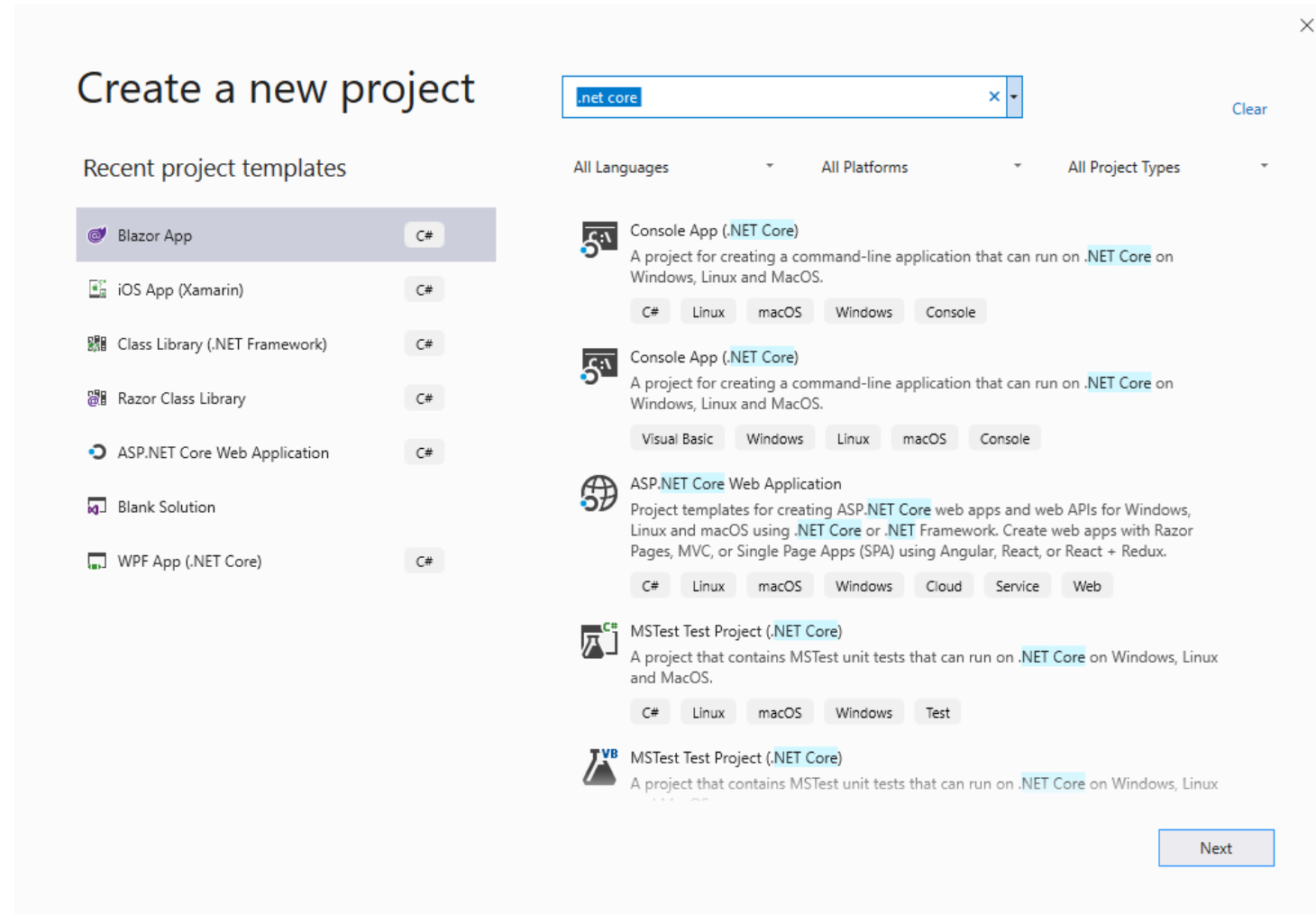
- .NET Core will get deployed along with the application package + dependencies
  - App package size will be larger
  - We can be sure that it will run
  - We can select which platform version we want to use it with
  - Can be made dependent on version of framework that we build app with
- Will create exe, which will load your DLL



# DEMO

Creating an app with SCD

# Creating .NET Core apps with Visual Studio 2019



# Available options

- Console App
  - Class Library
  - Unit Test Project
  - xUnit Test Project
  - ASP.NET Core Web Application
- 
- With .NET Core 3 (2019), you can also target also WPF and WinForms projects
    - Not “cross-platform” though

# Deployment options

- Visual Studio also supports FDD and SCD
- FDD is again the default
  - Will generate executable DLL again
  - Needs .NET Core to be installed
- Runtime must be set to Portable

## Publish


Publish your app to Azure or another host. [Learn more](#)

 FolderProfile ▼

Publish

[Create new profile](#)

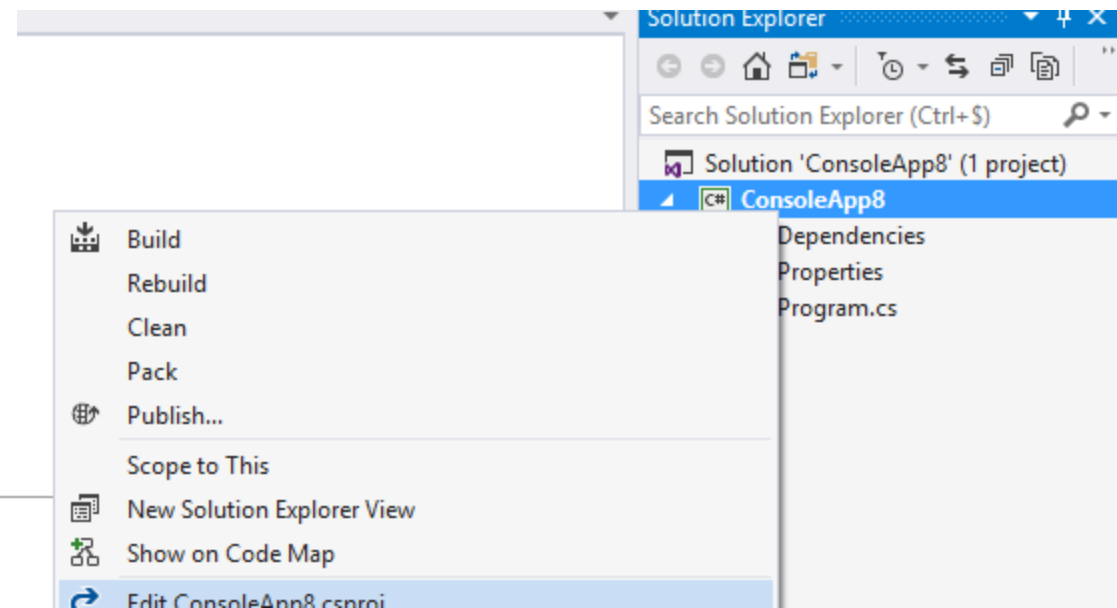
## Summary

Target Location	bin\Release\PublishOutput 	<a href="#">Settings</a>
Configuration	Release	<a href="#">Delete profile</a>
Target Framework	netcoreapp1.1	
Target Runtime	Portable	

# Self-contained Deployments

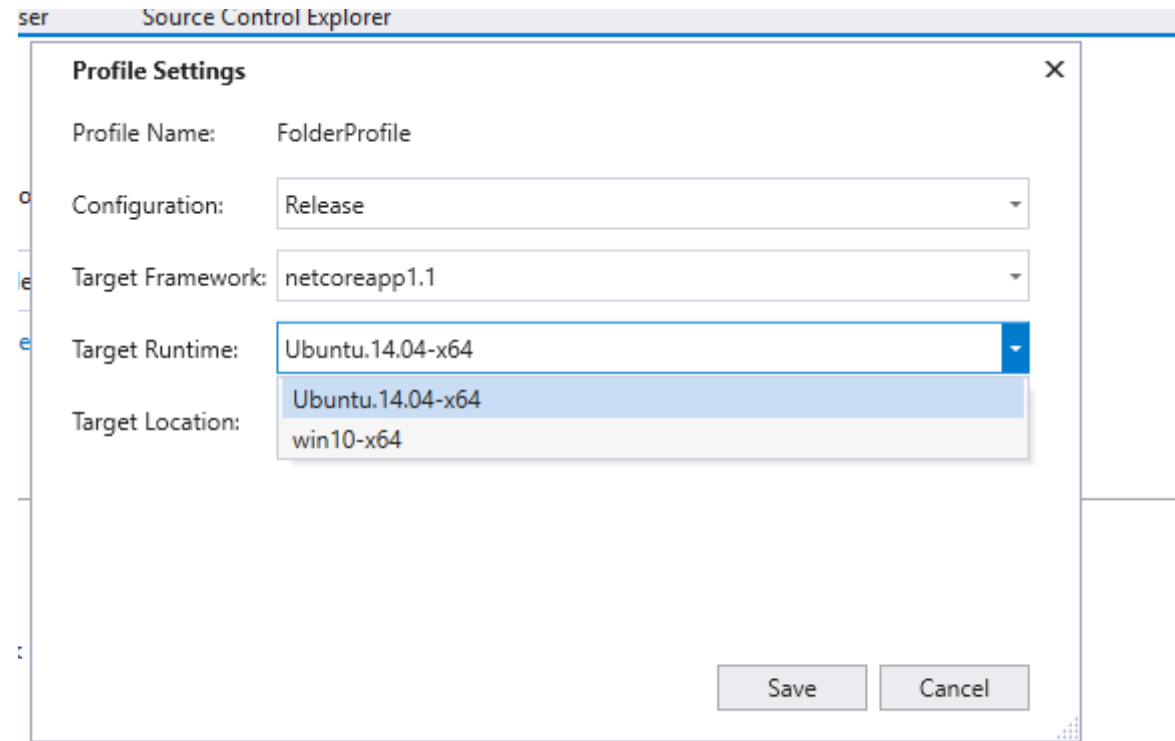
- Requires edit to project file before we can do anything
- Will then create a similar package
  - Edit csproj to add needed runtimes

```
<Project Sdk="Microsoft.NET.Sdk">  
  
  <PropertyGroup>  
    <OutputType>Exe</OutputType>  
    <TargetFramework>netcoreapp1.1</TargetFramework>  
    <RuntimeIdentifiers>win10-x64;Ubuntu.14.04-x64</RuntimeIdentifiers>  
  </PropertyGroup>  
  
</Project>
```





# Publishing the app





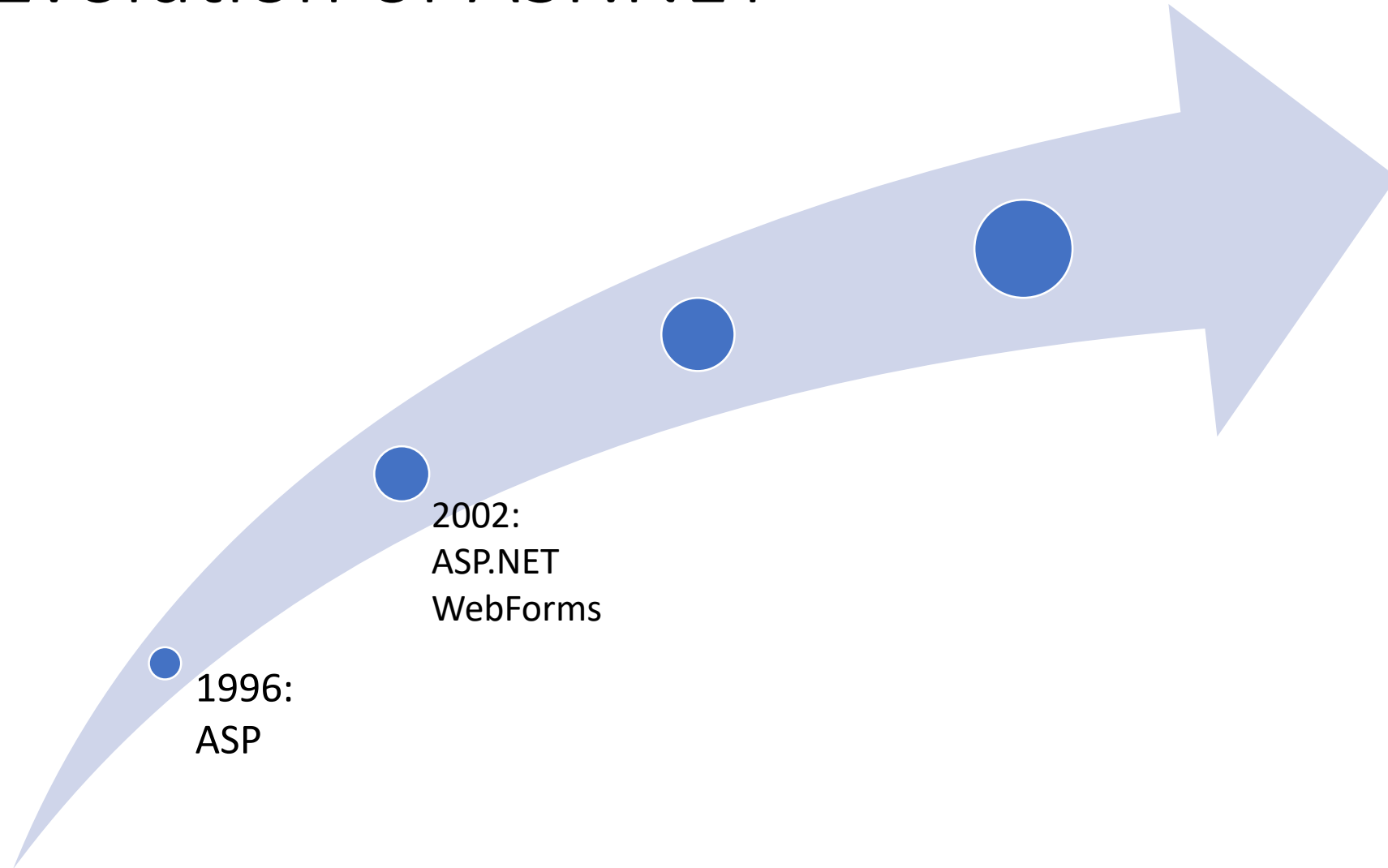
# DEMO

Creating .NET Core apps with Visual Studio  
Publishing options

Moving to ASP.NET Core 3.0

The evolution to ASP.NET Core

# The Evolution of ASP.NET

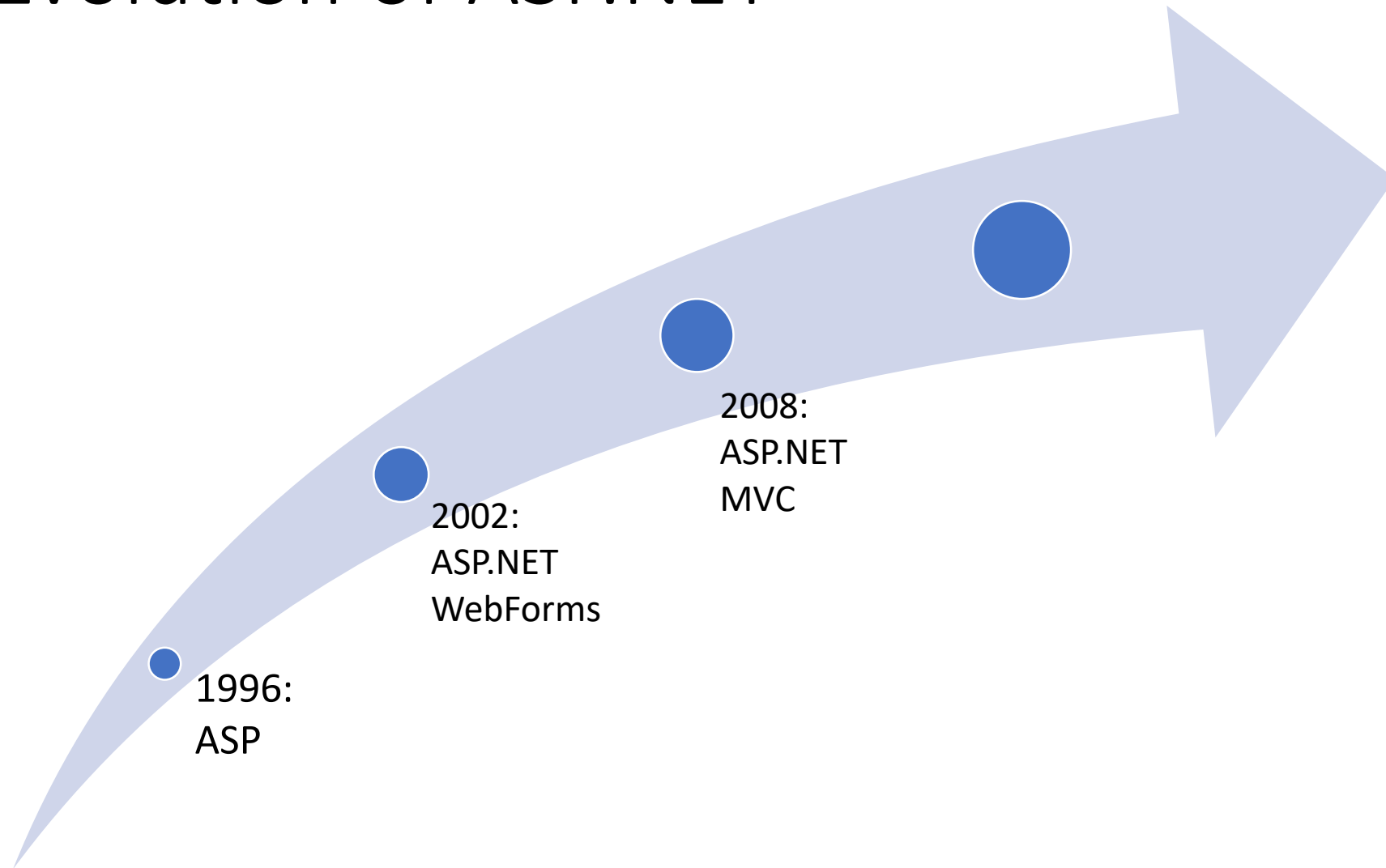


# WebForms had its problems...

- ViewState
- HTML was difficult to manage
- Hard to test
- Tight coupling (MVP)
- Page Life Cycle



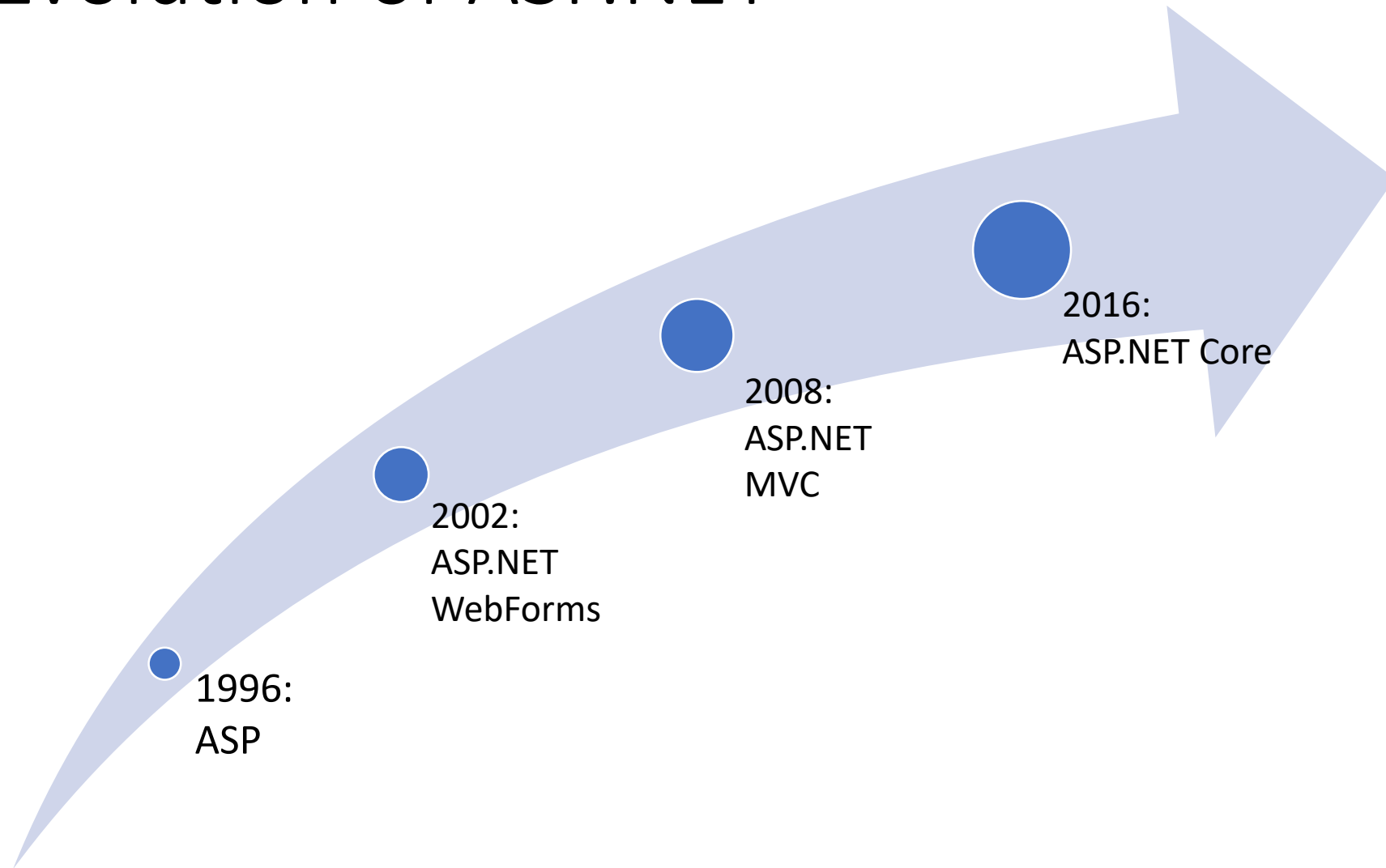
# The Evolution of ASP.NET



# Problems with the Original ASP.NET MVC

- Dependency on IIS (System.Web)
- Dependency on .NET framework
- Web evolves faster than .NET
- Not created with cloud in mind

# The Evolution of ASP.NET



# ASP.NET Core

*“ASP.NET Core is a new open-source and cross-platform framework for building modern cloud based internet connected applications, such as web apps, IoT apps and mobile backends.”*

*source: <https://docs.microsoft.com/en-us/aspnet/core>*

# Welcome to ASP.NET Core 3.0

- Built on top of .NET Core
- Cross-platform
- Not tied to original .NET framework

# What does ASP.NET Core bring us then?

- Unification between MVC and Web API
- Dependency Injection
- Modular HTTP request pipeline
- Based on NuGet
- Cloud-ready
- IIS or self-host
- Open source
- Community focus
- Cross-platform
- New tooling
- Better integration of client-side frameworks
- Command-line support



# What's supported at this point?



Latest version: 3.0 (last week 😊)

- Console applications
- ASP.NET Core MVC
- ASP.NET Core Web API
- .NET Standard Libraries
- Blazor server-side apps

# Main new features in ASP.NET Core MVC

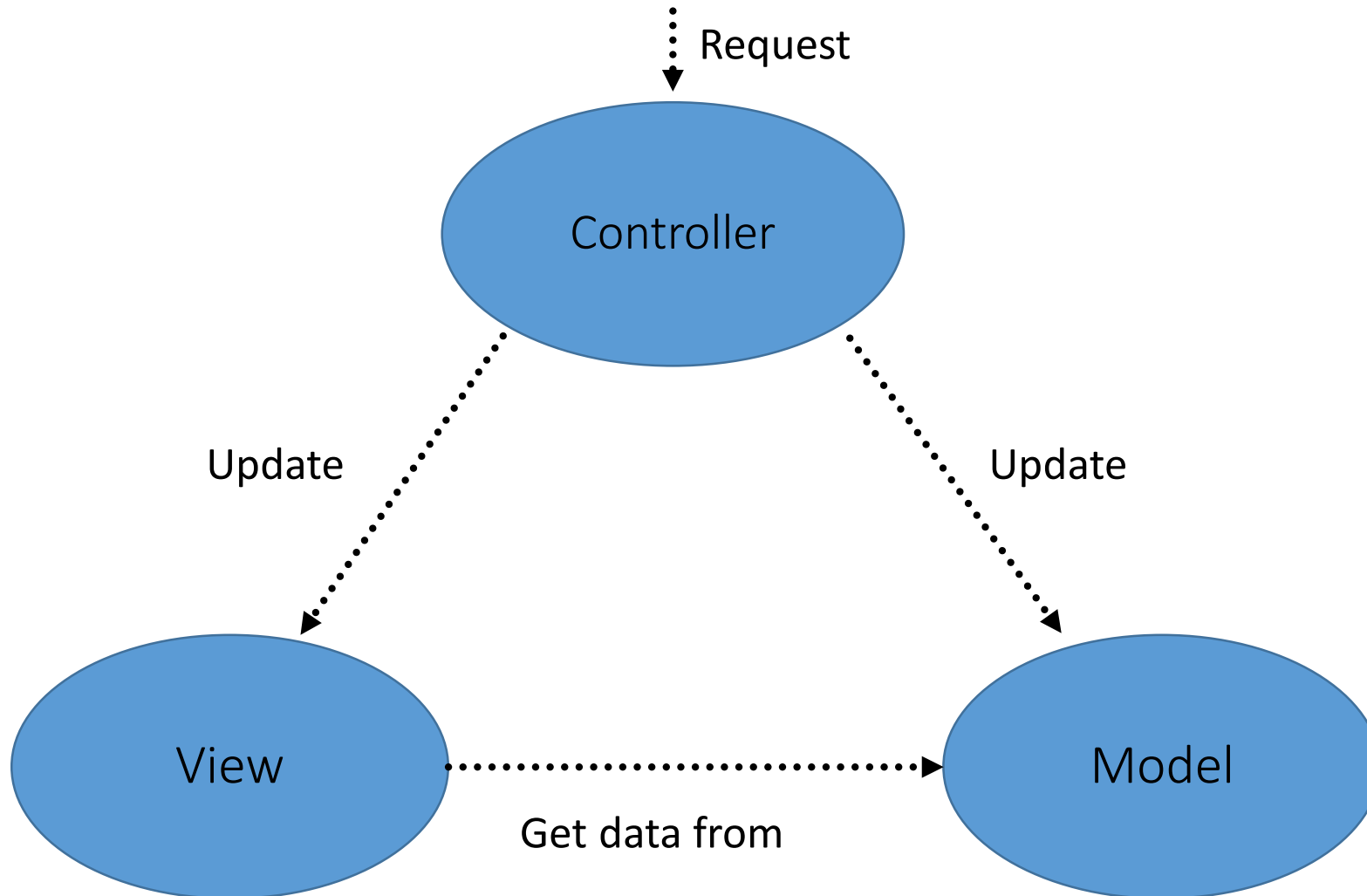
- Hosting
- Middleware
- Dependency Injection
- Configuration
- Identity
- Tag Helpers
- View Components

# The MVC in ASP.NET Core MVC



- Model-View-Controller
  - Architectural pattern
  - Separation of concerns
  - Promotes testability and maintainability

# The MVC in ASP.NET Core MVC



# Creating ASP.NET Core applications

- Using dotnet, we can also generate ASP.NET Core applications
  - Empty web application
  - MVC Web application
  - Web API
- Works with C# and F#

# Creating ASP.NET Core applications

- Different options
  - Empty: `dotnet new web -n <ProjectName>`
  - MVC: `dotnet new mvc -n <ProjectName>`
  - MVC without authentication: `dotnet new mvc -au None -n <ProjectName>`
  - Web API: `dotnet new webapi -n <ProjectName>`
- We can change the default SQLite to LocalDB
  - `dotnet new mvc -n <ProjectName> -uld true -au none`

Name	Date modified	Type	Size
Controllers	09/08/2017 21:05	File folder	
obj	09/08/2017 21:06	File folder	
Views	09/08/2017 21:05	File folder	
wwwroot	09/08/2017 21:05	File folder	
.bowerrc	09/08/2017 21:05	Bower RC Source ...	1 KB
appsettings.Development.json	09/08/2017 21:05	JSON Source File	1 KB
appsettings.json	09/08/2017 21:05	JSON Source File	1 KB
bower.json	09/08/2017 21:05	JSON Source File	1 KB
bundleconfig.json	09/08/2017 21:05	JSON Source File	1 KB
Program.cs	09/08/2017 21:05	Visual C# Source f...	1 KB
Startup.cs	09/08/2017 21:05	Visual C# Source f...	2 KB
webapptest.csproj	09/08/2017 21:05	Visual C# Project f...	1 KB





# DEMO

Creating a web app with ASP.NET Core MVC

# Summary

- .NET Core is a new world for Microsoft developers
- .NET Core 3.0 is an important update
  - API-wise, it's almost the same
- CLI is big addition
- VS2019 will continue to be most important way of working
- ASP.NET Core is major (r)evolution

Questions?

# An introduction to .NET Core 3.0 and ASP.NET Core 3.0

Gill Cleeren

@gillcleeren – [www.snowball.be](http://www.snowball.be)