.NET

Free. Cross-platform. Open source.
A developer platform for building all your apps.

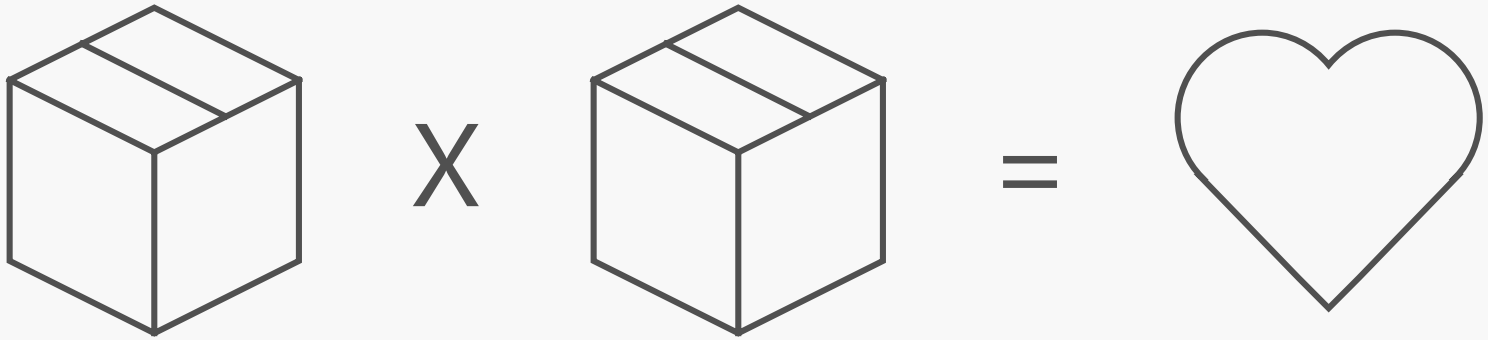
www.dot.net

From .NET Framework to .NET Core

Jon Galloway – Executive Director, .NET Foundation
@JonGalloway

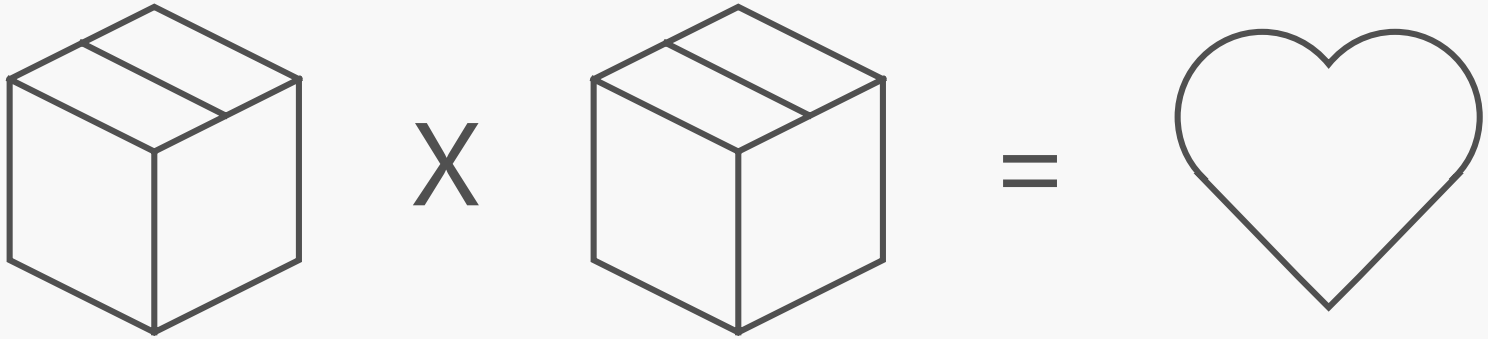
Why Windows Desktop on .NET Core?

- Deployment Flexibility
  - Side-by-side support
  - Machine global or app local framework
  - Self-contained EXEs
- Core runtime and API improvements
- Performance



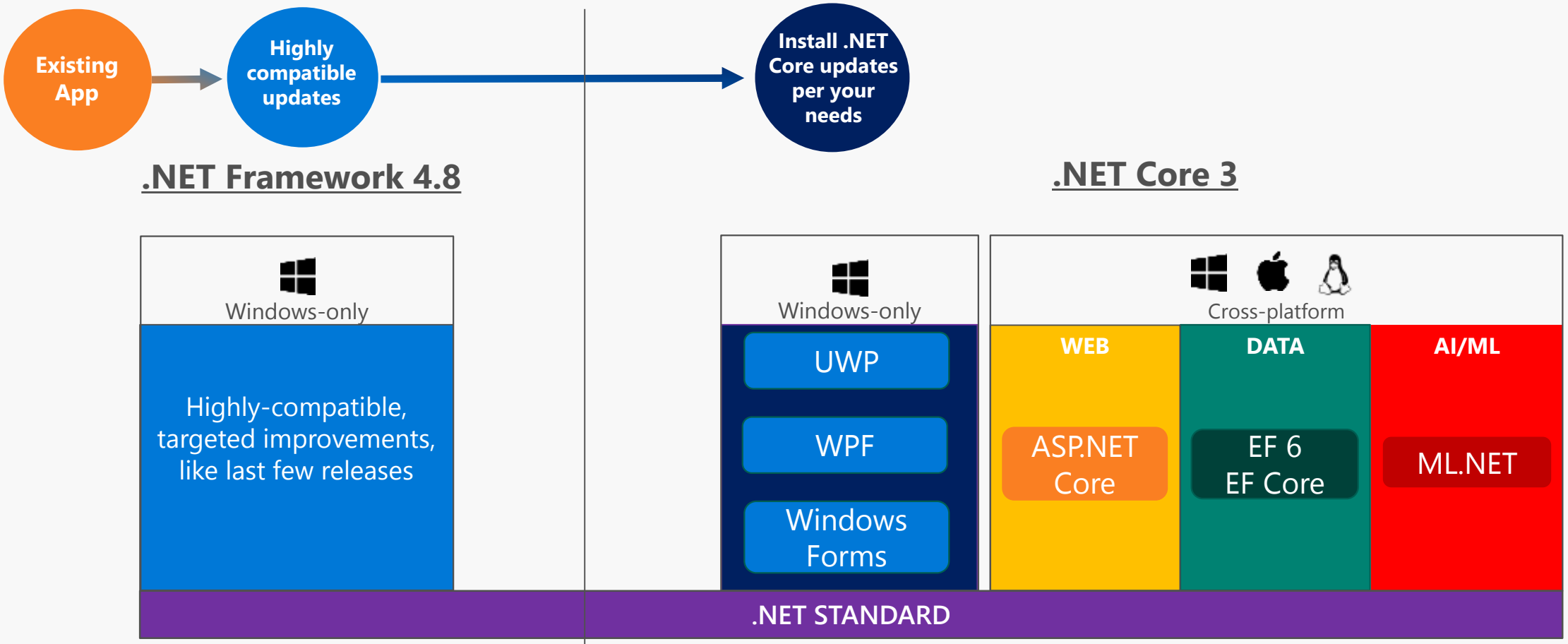
Why Web on .NET Core?

- Deployment Flexibility
  - Side-by-side support
  - Machine global or app local framework
  - Docker
- Core runtime and API improvements
- Performance
- Productivity… Really!



Update .NET Framework Apps

Modernize Desktop Apps with .NET Core 3



- .NET Framework support **unchanged** (supported for life of Windows)

- Side-by-side support
- Self-contained EXEs

- XAML Islands - WinForms & WPF apps can host UWP controls
- Full access to Windows 10 APIs

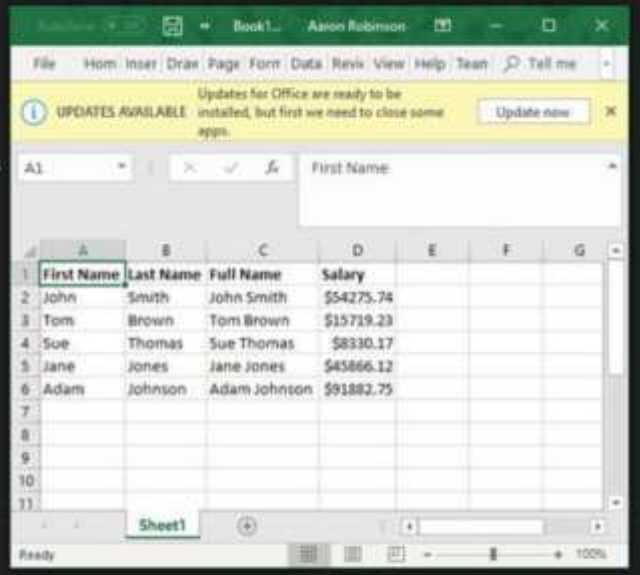
FEATURES IN BOTH FXs

Rich Lander
@runfaster2000

Following

The @dotnet Core team just got COM interop working well enough to drive Excel. Coming with .NET Core 3.

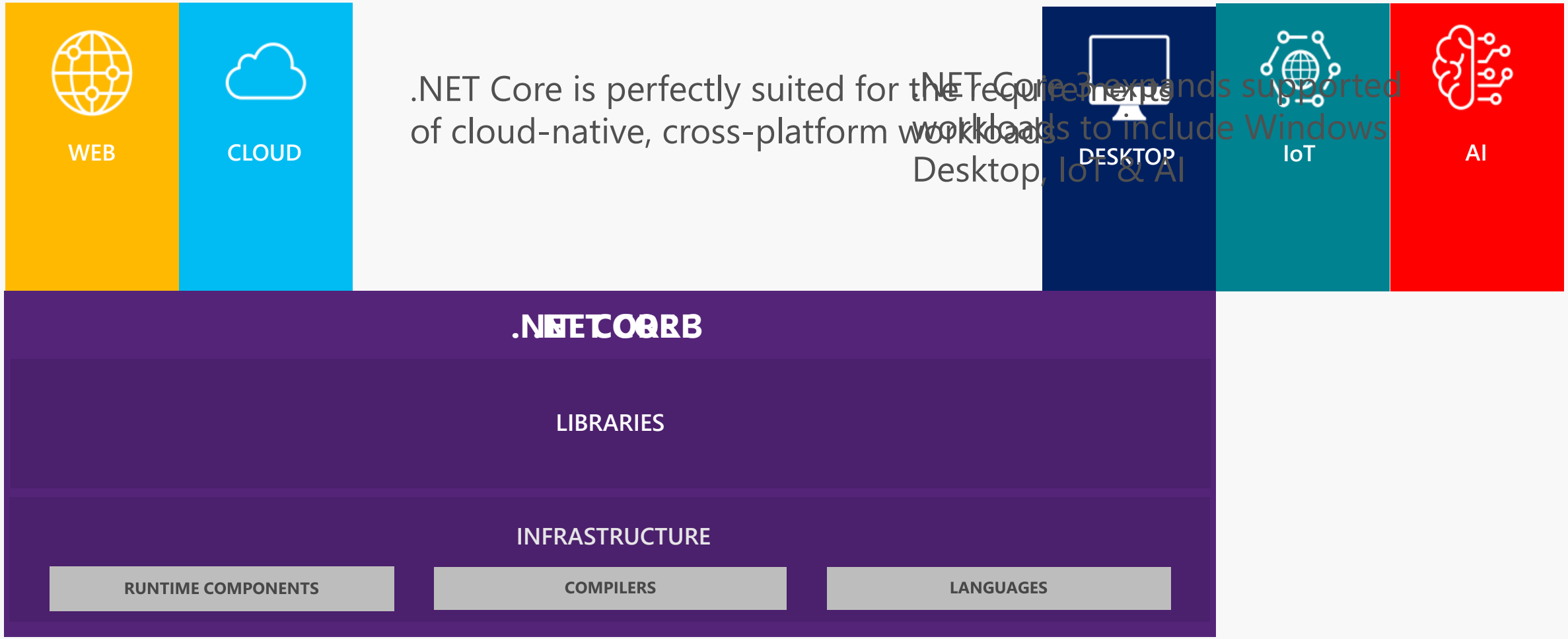
```
static void RunExcel()
{
    Excel.Application oXL;
    Excel._Workbook oWB;
    Excel._Worksheet oSheet;
    Excel.Range oRng;

    try
    {
        //Start Excel and get Application object.
        oXL = new Excel.Application();
        oXL.Visible = true;

        //Get a new workbook.
        oWB = (Excel._Workbook)(oXL.Workbooks.Add(Missing.Value));
        oSheet = (Excel._Worksheet)oWB.ActiveSheet;

        //Add table headers going cell by cell.
        oSheet.Cells[1, 1] = "First Name";
        oSheet.Cells[1, 2] = "Last Name";
        oSheet.Cells[1, 3] = "Full Name";
        oSheet.Cells[1, 4] = "Salary";

        //Format A1:D1 as bold, vertical alignment = center.
        oSheet.get_Range("A1", "D1").Font.Bold = true;
        oSheet.get_Range("A1", "D1").VerticalAlignment =
        Excel.XlVAlign.xlVAlignCenter;

        // Create an array to multiple values at once.
        string[,] saNames = new string[5, 2];

        saNames[0, 0] = "John";
        saNames[0, 1] = "Smith";
        saNames[1, 0] = "Tom";
```



10:46 AM - 20 Oct 2018

350 Retweets **1,037** Likes

.NET Core 3


WEB
CLOUD
DESKTOP
IoT
AI
.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
.NET CORE
LIBRARIES
INFRASTRUCTURE
RUNTIME COMPONENTS
COMPILERS
LANGUAGES

The Future of .NET Framework

- .NET Framework is slowing down
- Why? We want to stop breaking your apps!
  - Billions of installs. Getting much harder to release major features without compatibility issues
  - Can innovate much faster with .NET Core because of its side-by-side nature
- We *will* continue to update
  - Many major product lines within Microsoft depend on .NET Framework
  - We will continue to update .NET Framework for years to come. *High-compat features only*.
  - Things like… new security protocols, bug fixes, Windows features, etc.
- No changes to support policy
  - Support for 10 years (5 mainstream+5 custom) per Windows support policy for each release
  - .NET Framework will remain a Windows component and service policy is the same as Windows
- Recommend new development on .NET Core

The Bottom Line

Don’t Panic!	New development	Existing code
• You don’t have to move!	• .NET Core is a good choice	• Costs and benefits to migrating

Leverage Guidance and Tools

Choosing between .NET Core and .NET Framework for server apps

06/19/2018 • 6 minutes to read • Contributors all

There are two supported implementations for building server-side applications with .NET: .NET Framework and .NET Core. Both share many of the same components and you can share code across the two. However, there are fundamental differences between the two and your choice depends on what you want to accomplish. This article provides guidance on when to use each.

Use .NET Core for your server application when:

- You have cross-platform needs.
- You are targeting microservices.
- You are using Docker containers.
- You need high-performance and scalable systems.
- You need side-by-side .NET versions per application.

Use .NET Framework for your server application when:

- Your app currently uses .NET Framework (recommendation is to extend instead of migrating).
- Your app uses third-party .NET libraries or NuGet packages not available for .NET Core.
- Your app uses .NET technologies that aren't available for .NET Core.
- Your app uses a platform that doesn't support .NET Core.

When to choose .NET Core

https://aka.ms/choose-netcore-or-framework

CONNECT(); 2017

.NET - Introducing the Windows Compatibility Pack for .NET Core

By Immo Landwerth | Connect(); 2017

The Microsoft .NET Framework is still the best choice for certain styles of apps, especially for desktop apps and Web apps that use ASP.NET Web Forms. But if you need highly scalable Web apps, create self-contained deployments using Docker, or if you need to run on Linux, then you want to consider porting to .NET Core. But bringing existing code to .NET Core can be a challenge. In this article, I'll explain how you can use the new Windows Compatibility Pack for .NET Core. It provides access to APIs that were previously available only for .NET Framework (for example, System.Drawing, System.DirectoryServices, ODBC, WMI and many more). Because this includes both cross-platform and Windows-only technologies, it's critical to understand early if you're using APIs that might interfere with your cross-platform goals. I'll address this by showcasing the new API analyzer, which gives you live feedback as you're editing code.

https://msdn.microsoft.com/en-us/magazine/mt814807.aspx

Porting to .NET Core from .NET Framework

06/20/2016 • 2 minutes to read • Contributors all

If you've got code running on the .NET Framework, you may be interested in running your code on .NET Core 1.0. This article covers an overview of the porting process and a list of the tools you may find helpful when porting to .NET Core.

Overview of the Porting Process

The recommended process for porting follows the following series of steps. Each of these parts of the process are covered in more detail in further articles.

1. Identify and account for your third-party dependencies.

   This will involve understanding what your third-party dependencies are, how you depend on them, how to see if they also run on .NET Core, and steps you can take if they don't.

2. Retarget all projects you wish to port to target .NET Framework 4.6.2.

   This ensures that you can use API alternatives for .NET Framework-specific targets in the cases where .NET Core can't support a particular API.

3. Use the .NET Portability Analyzer to analyze your assemblies and develop a plan to port based on its results.

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

Tool: .NET Portability Analyzer

Options

Search Options (Ctrl+E)

- Environment
- Projects and Solutions
- Source Control
- Work Items
- Text Editor
- Debugging
- IntelliTrace
- Performance Tools
- .NET Portability Analyzer
  - General
- Cross Platform
- Database Tools
- F# Tools
- Graphics Diagnostics
- Live Unit Testing
- NuGet Package Manager
- Office Tools
- SQL Server Tools
- Text Templating
- Web
- Web Forms Designer
- Web Performance Test Tools
- Windows Forms Designer
- Workflow Designer
- XAML Designer

General

Default output directory C:\Portability Analysis

Default output name ApiPortAnalysis

Output formats

☐ Json ☐ HTML ☑ Excel

Target Platforms

.NET Core
☐ 1.0 ☐ 1.1 ☐ 2.0

.NET Core + Platform Extensions
☐ 1.0

.NET Framework
☐ 1.1 ☐ 2.0 ☐ 3.0 ☐ 3.5 ☐ 4.0 ☐ 4.5 ☐ 4.5.1 ☐ 4.5.2 ☐ 4.6

.NET Standard
☐ 1.0 ☐ 1.1 ☐ 1.2 ☐ 1.3 ☐ 1.4 ☐ 1.5 ☑ 1.6 ☐ 2.0

.NET Standard + Platform Extensions
☐ 1.6 ☐ 2.0

ASP.NET Core
☐ 1.0

Mono
☐ 2.0 ☐ 3.5 ☐ 4.0 ☐ 4.5

Silverlight
☐ 2.0 ☐ 3.0 ☐ 4.0 ☐ 5.0

Windows
☐ 8.0 ☐ 8.1 ☐ 10.0

Refresh

More information is available at h

.NET Portability Report

Summary

Assembly	Mono 4.5
HubApp1.Windows	94.84%

HubApp1.Windows

Missing assemblies

Windows, Version=255.255.255.255, Culture=neutral, PublicKeyToken=null

Target type	Mono 4.5	Recommended changes
Windows.Foundation.Rect	✖	
get_Width	✖	
get_Height	✖	

Error List

0 Errors | 0 Warnings | 25 Messages

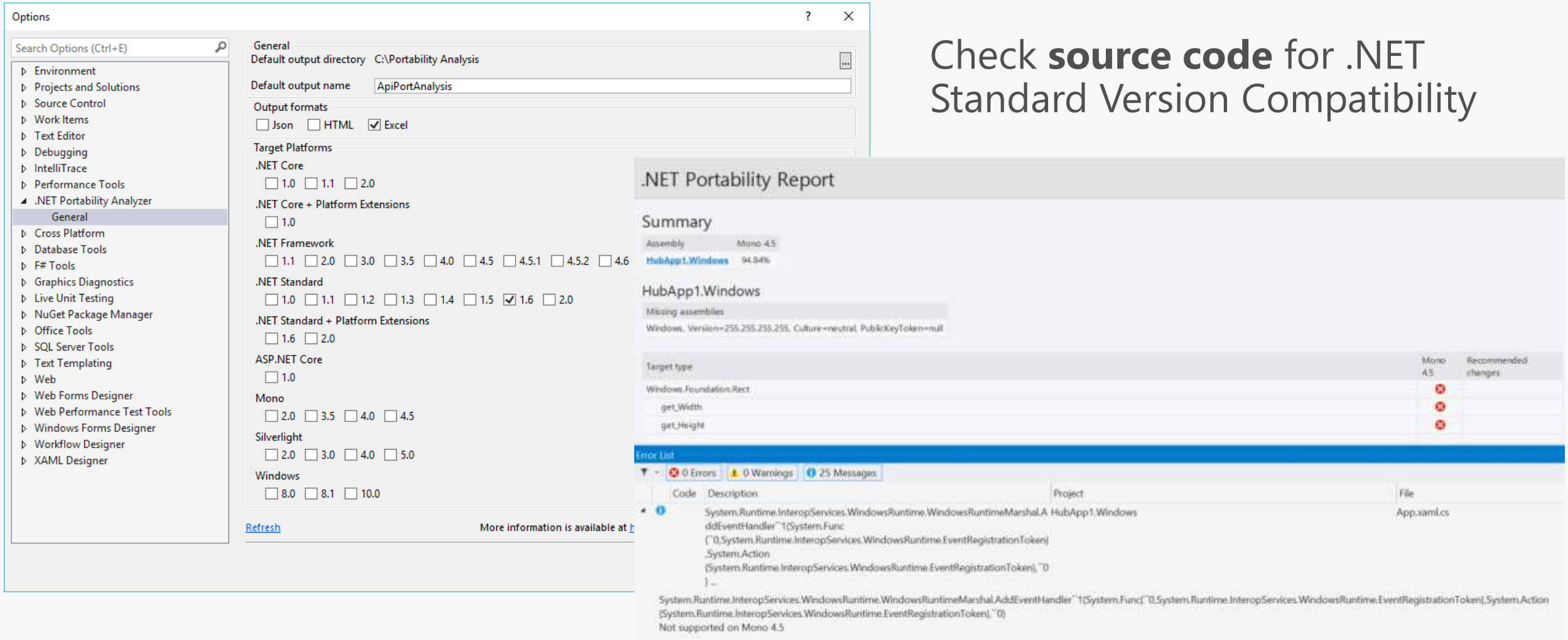
Code	Description	Project	File
	System.Runtime.InteropServices.WindowsRuntime.WindowsRuntimeMarshal.AddEventHandler`1(System.Func(`0,System.Runtime.InteropServices.WindowsRuntime.EventRegistrationToken),System.Action(System.Runtime.InteropServices.WindowsRuntime.EventRegistrationToken),`0) ...	HubApp1.Windows	App.xaml.cs

System.Runtime.InteropServices.WindowsRuntime.WindowsRuntimeMarshal.AddEventHandler`1(System.Func(`0,System.Runtime.InteropServices.WindowsRuntime.EventRegistrationToken),System.Action(System.Runtime.InteropServices.WindowsRuntime.EventRegistrationToken),`0)
Not supported on Mono 4.5

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

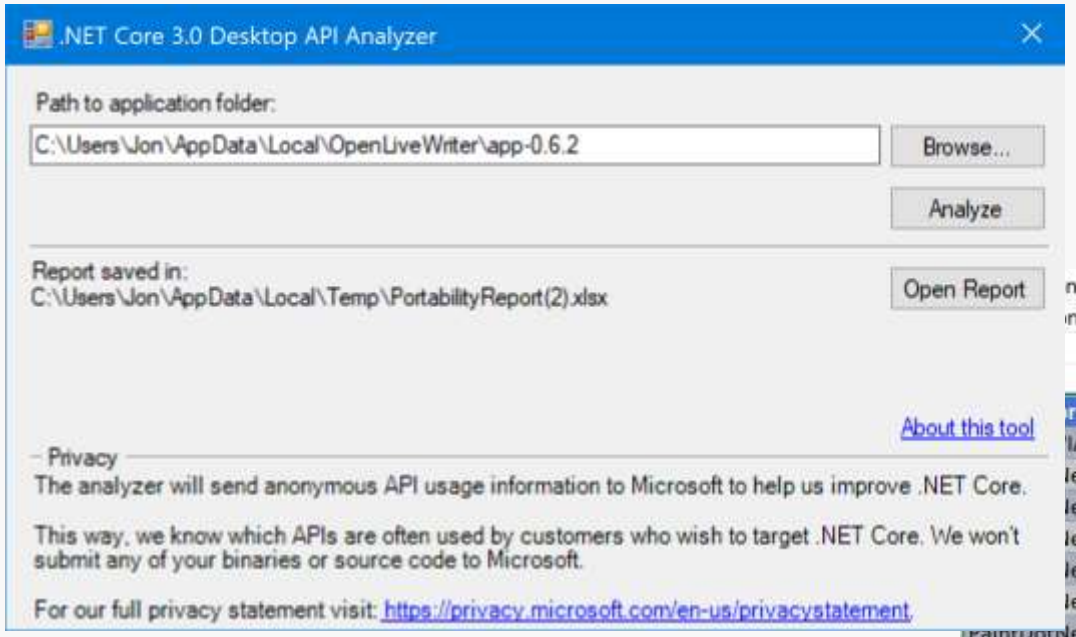
Tool: .NET Portability Analyzer

Check **source code** for .NET Standard Version Compatibility



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

Tool: .NET Core 3.0 Desktop API Analyzer



Check **binaries** for
.NET Core 3.0 Compatibility

n Id	d3c04720-2a8e-4c58-a677-60f3b203c24c
n	
	.NET Core + Platform Extensions,.NET Framework,.NET Standard + Platform Extensions

r assembly name entries	Target Framework	.NET Core + Pla	.NET Framewor	.NET Standard +
IA		100	100	84.13
let	.NETFramework,Version=v4.7	58.5	100	58.5
let.Base	.NETFramework,Version=v4.7	95.3	100	95.3
let.Core	.NETFramework,Version=v4.7	80.73	100	80.73
let.Data	.NETFramework,Version=v4.7	92.23	100	92.23
let.Effects	.NETFramework,Version=v4.7	62.06	100	62.06
PaintDotNet.Framework	.NETFramework,Version=v4.7	45.05	100	45.05
PaintDotNet.Resources	.NETFramework,Version=v4.7	97.52	100	97.52
PaintDotNet.SystemLayer	.NETFramework,Version=v4.7	77.49	100	77.21
PaintDotNet.SystemLayer.Native.x64	.NETFramework,Version=v4.7	99.69	100	99.69
PaintDotNet.SystemLayer.Native.x86	.NETFramework,Version=v4.7	99.69	100	99.69
PdnRepair	.NETFramework,Version=v4.7	100	100	100
SetupNgen	.NETFramework,Version=v4.7	100	100	100
UpdateMonitor	.NETFramework,Version=v4.7	100	100	100

API Catalog last updated on	Monday, May 7, 2018
See 'http://go.microsoft.com/fwlink/?LinkId=397652' to learn how to read this table	

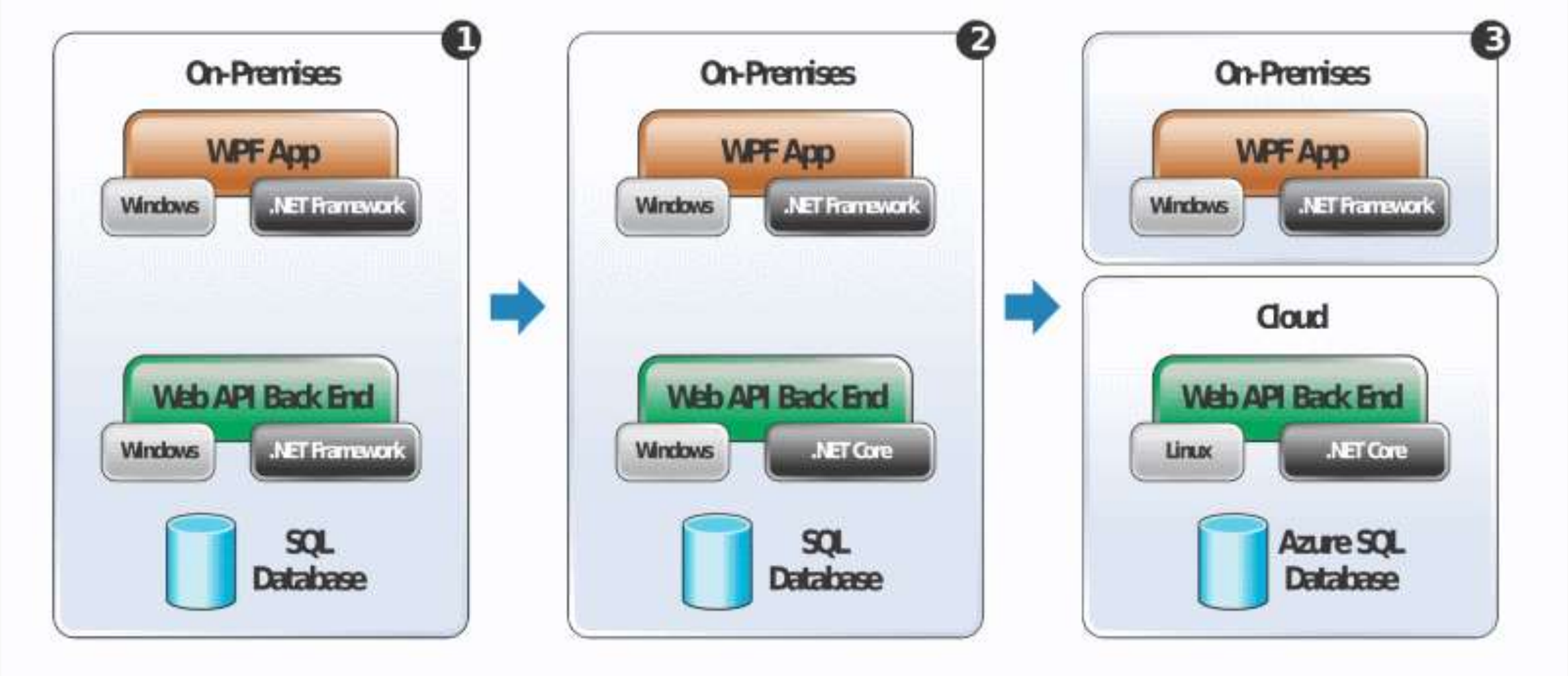
https://blogs.msdn.microsoft.com/dotnet/are-your-windows-forms-and-wpf-applications-ready-for-net-core-3-0/

Strategy: Divide and Conquor

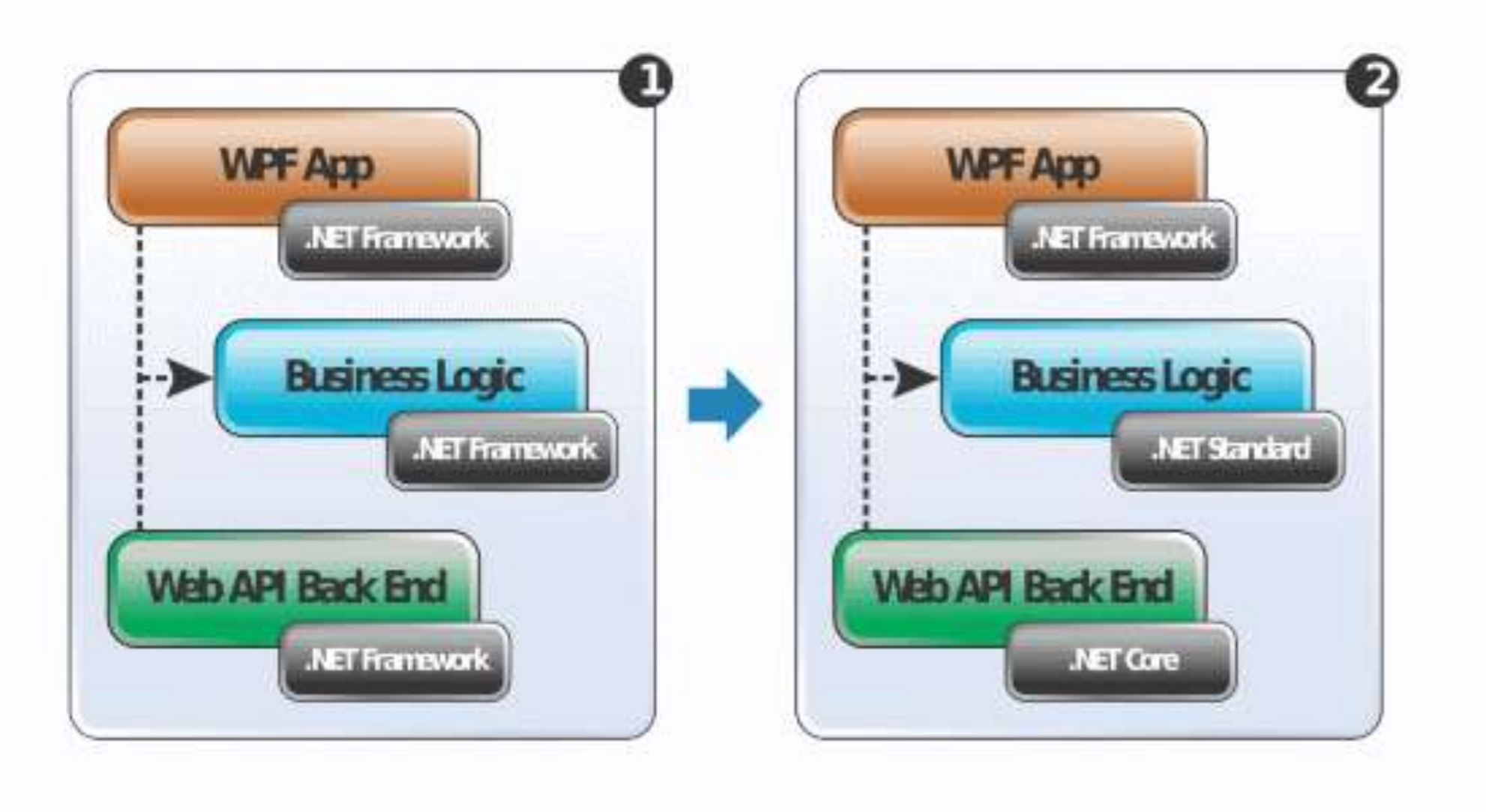
What Goes Where

- Desktop Apps -> .NET Core 3
- Business Logic -> .NET Standard
- Back End APIs -> .NET Core 3
- Front End Web Apps -> Well, this is tougher…

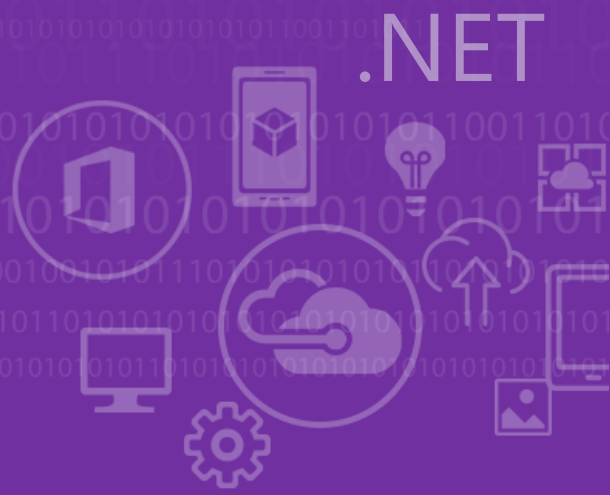
Migrating a Typical .NET App Partially to the Cloud



Handling Shared Code When Targeting Multiple .NET Implementations



Desktop



What to expect in porting desktop apps to Core

- Update project files to target .NET Core 3 and recompile.
- Dependencies will not need to retarget and recompile. There will be additional benefits if you update dependencies.

Business Logic & .NET Standard

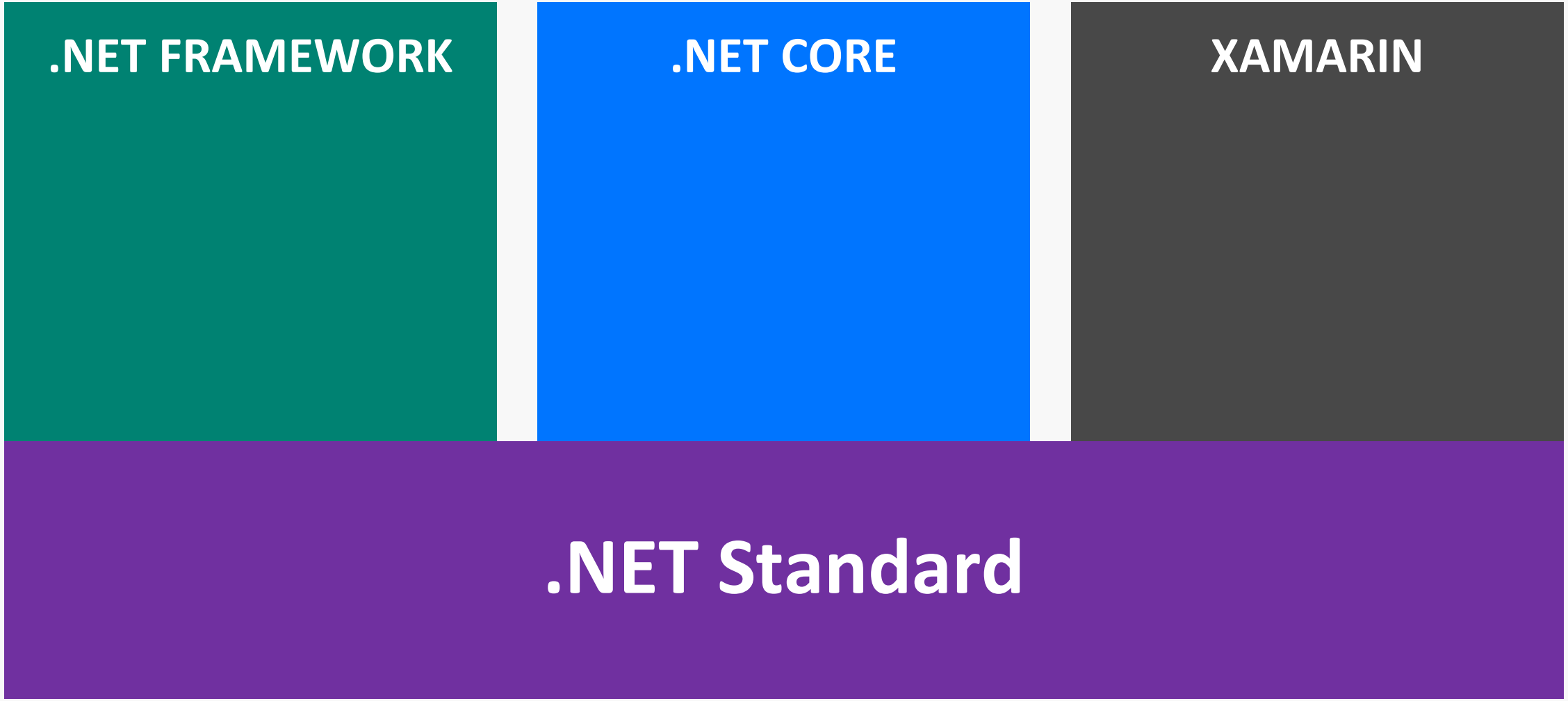
What is .NET Standard?

- .NET Standard is a specification
- A set of APIs all .NET platforms have to implement

.NET Standard	~	HTML specification
.NET Core .NET Framework Xamarin	~	Browsers

```
$ dotnet new classlib -o My.Class.Library
```

.NET Standard in context



What's in .NET Standard?

Microsoft.Win32.SafeHandles
System
System.CodeDom.Compiler
System.Collections
System.Collections.Concurrent
System.Collections.Generic
System.Collections.ObjectModel
System.Collections.Specialized
System.ComponentModel
System.ComponentModel.Design
System.
System.
System.
System.
System.
System.
System.
System.
System.Diagnostics.SymbolStore
System.Diagnostics.Tracing
System.Drawing
System.Dynamic
System.Globalization
System.IO
System.IO.Compression
System.IO.IsolatedStorage
System.IO.MemoryMappedFiles
System.IO.Pipes
System.Linq
System.Linq.Expressions
System.Net
System.Net.Cache
System.Net.Http
System.Net.Http.Headers
System.Net.Mail
System.Resources
System.Runtime
System.Runtime.CompilerServices
System.Runtime.ConstrainedExecution
System.Runtime.ExceptionServices
System.Runtime.InteropServices
System.Runtime.InteropServices.ComTypes
System.Runtime.Serialization
System.Runtime.Serialization.Formatters
System.Runtime.Serialization.Formatters.Binary
System.Runtime.Serialization.Json
System.Runtime.Versioning
System.Security
System.Security.Authentication
System.Security.Authentication.ExtendedProtection
System.Security.Claims
System.Security.Cryptography.X509Certificates
System.Transactions
System.Windows.Input
System.Xml
System.Xml.Linq
System.Xml.Resolvers
System.Xml.Schema
System.Xml.Serialization
System.Xml.XPath
System.Xml.Xsl

All the foundational APIs
~37k APIs in .NET Standard 2.0

Versions of .NET Standard

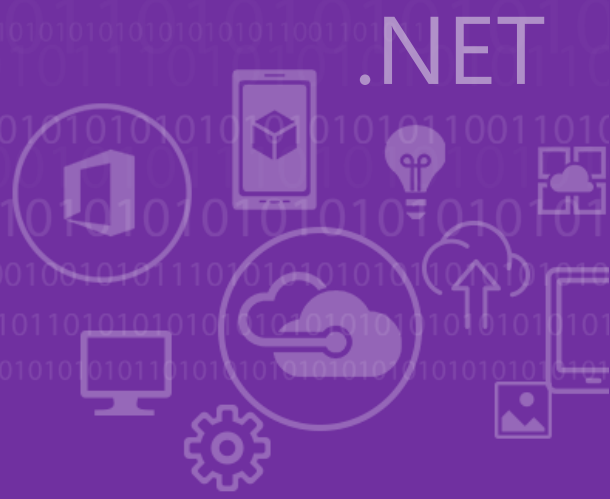
.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0
.NET Framework [1]	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1	4.6.1	4.6.1
Mono	4.6	4.6	4.6	4.6	[illegible]			
Xamarin.iOS								
						[illegible]	10.0.16299	10.0.16299
	[illegible]	8.0	8.1					
Windows Phone	8.1	8.1	8.1					
Windows Phone Silverlight	8.0							

Don't worry about it.
Start with .NET Standard 2.0.

.NET Standard is also Open Source!

- Anybody can propose API additions
- The review board approves the API
  - Has representatives from the .NET Foundation, Microsoft, Xamarin/Mono, & Unity
- Acceptance requires
  - A stable implementation that is shipped in at least one .NET implementation
  - Sponsorship from a board member
- Next version is planned here:
  - https://github.com/dotnet/standard/tree/master/docs/planning/netstandard-2.1

Using platform-specific APIs from .NET Standard



Windows Compatibility Pack

- Microsoft.Windows.Compatibility (NuGet package)
  - Can be referenced from .NET Core as well as from .NET Standard
  - Has ~21k APIs (Windows-only as well as cross-platform)

- Contents

ACLs	Drawing	Ports
Code Pages	EventLog	Registry
CodeDom	MEF v1	Runtime Caching
Configuration	Odbc	WCF
Crypto	Perf Counters	Windows Services
DirectoryServices	Permissions	…

Detecting usage of unsupported APIs

- Use API Analyzer!
  - https://aka.ms/apianalyzer
  - Roslyn analyzer that flags usages of APIs that don't work across all platforms

```
private static string GetLoggingPath()
{
    using (var key = Registry.CurrentUser.OpenSubKey(@"Software\Fabrikam\AssetManagement"))
    {
        if (key?.GetValue("LoggingDirectoryP
            return configuredPath;
    }

    var appDataPath = Environment.GetFolderP                                    pplicationData);
    return Path.Combine(appDataPath, "Fabrik
}
```

RegistryKey RegistryKey.OpenSubKey(string name) (+ 4 overloads)
Retrieves a subkey as read-only.

Exceptions:
ArgumentNullException
ObjectDisposedException
System.Security.SecurityException

RegistryKey.OpenSubKey(string) isn't supported on Linux, MacOSX

Show potential fixes (Alt+Enter or Ctrl+.)

Error List

Entire Solution | 0 Errors | 2 Warnings | 0 Messages | Build + IntelliSense | Search Error List

Code	Description	Project	File	Line	Suppression St...
PC001	RegistryKey.GetValue(string) isn't supported on Linux, MacOSX	Fabrikam.Shared	Logger.cs	19	Active
PC001	RegistryKey.OpenSubKey(string) isn't supported on Linux, MacOSX	Fabrikam.Shared	Logger.cs	17	Active

Testing

More than Unit Tests

- Unit Tests
- Integration Tests
- UI Automation Testing
- Web Automation Testing

Example: WPF Coded UI Test

Microsoft | Visual Studio Docs Tasks Languages Workloads Product Resources More Documentation All Microsoft

Docs / Visual Studio / Testing / UI test automation / Walkthrough: Create, Edit and Maintain a Coded UI Test Feedback Edit Share Dark Sign in

Visual Studio 2017

Filter by title

- testing
- UI test automation
  - Walkthrough: Create, Edit and Maintain a Coded UI Test
  - Test UWP Apps with Coded UI Tests
  - Set a Unique Automation Property for UWP Controls
  - Use HTML5 Controls in Coded UI Tests
  - Test SharePoint 2010 Applications with Coded UI Tests
  - Create a Data-Driven Coded UI Test
  - Wait For Specific Events During Playback
  - Use Different Web Browsers with Coded UI Tests
  - Edit Coded UI Tests Using the

Walkthrough: Create, edit, and maintain a coded UI test

11/04/2016 • 10 minutes to read • Contributors all

In this walkthrough, you'll learn how to create, edit, and maintain a coded UI test to test a Windows Presentation Framework (WPF) app. The walkthrough provides solutions for correcting tests that have been broken by various timing issues and refactoring of controls.

Create a WPF app

1. On the **File** menu, point to **New**, and then select **Project**.

   The **New Project** dialog box appears.

2. In the **Installed** pane, expand **Visual C#**, and then select **Windows Desktop**.

3. Above the middle pane, verify that the target framework drop-down list is set to **.NET Framework 4.5**.

4. In the middle pane, select the **WPF Application** template.

5. In the **Name** text box, type **SimpleWPFApp**.

In this article

- Create a WPF app
- Create a shortcut to the WPF app
- Create a coded UI test for SimpleWPFApp
- Edit and rerun the coded UI test
- Refactor a control in SimpleWPFApp
- Map refactored control rerun the test
- Videos
- FAQ
- See also

https://docs.microsoft.com/en-us/visualstudio/test/walkthrough-creating-editing-and-maintaining-a-coded-ui-test

Example: Create a Coded UI Test using Selenium

Microsoft

PartsUnlimited example e-commerce website

Create a Coded UI Test using Selenium in Visual Studio

If you performed the previous lab, you will have used a screen recorder to create a coded UI test. While screen recorders are convenient, they often do not provide the level of control needed for more sophisticated testing.

In this lab, you will learn to create a coded UI test based on Selenium. The Selenium automation tools speak a language known as Selenese. Rather than issuing Selenese directly to the different types of browsers, web drivers are used to perform the translation. In this lab, you will install the Chrome, IE, and Firefox drivers, then automate the code from within the C# language in Visual Studio.

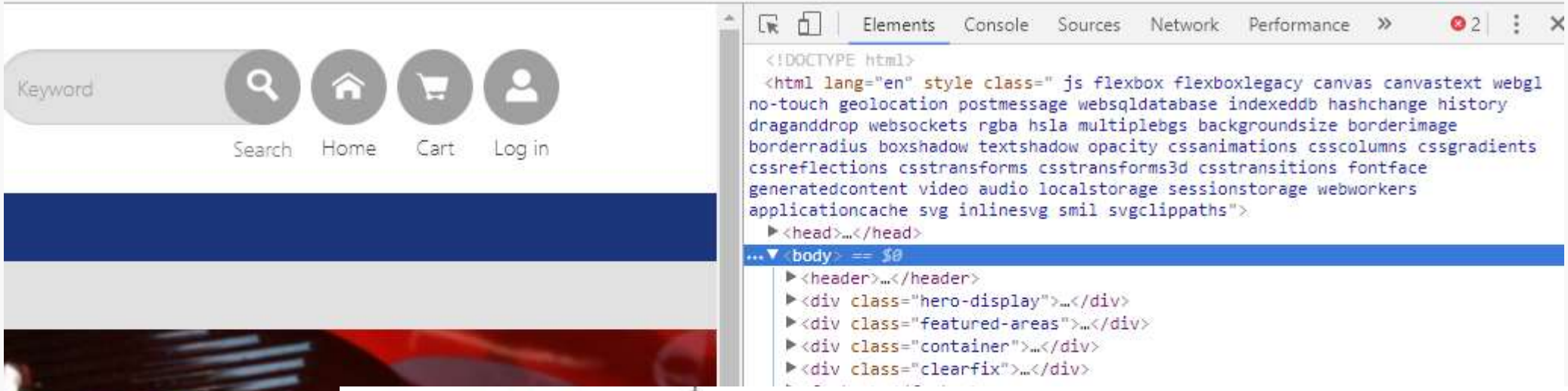
> Note that there is a Selenium IDE that can be used for recording web actions but using it is not currently recommended. It only works with older insecure versions of FireFox (up to version 5.4). In this lab, we will use the drivers and classes directly from code.

For the purposes of the lab, we will use the PartsUnlimited web application hosted on Azure to run this lab.

There have been reports that the keyword search on the home page is intermittently performing poorly. We will investigate how the keyword search on the home page performs, both by entering values into the box and clicking the search icon, and by directly referencing the search term a URL.

https://microsoft.github.io/PartsUnlimited/testing/200.5x-Testing-SeleniumCodedUITestinVS.html

Example: Create a Coded UI Test using Selenium




```
[TestMethod]
[TestCategory("Selenium")]
public void DirectSearchForLights()
{
    string lightSearchQuery = this.websiteURL + @"/Search?q=light";

    browserDriver = new ChromeDriver();
    browserDriver.Manage().Window.Maximize();
    browserDriver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(20));
    browserDriver.Navigate().GoToUrl(lightSearchQuery);
}
```

https://microsoft.github.io/PartsUnlimited/testing/200.5x-Testing-SeleniumCodedUITestinVS.html

Web Front Ends

Case Studies: ASP.NET Web Forms Examples

Contoso Univerity	• Rewrite
Wingtip Toys	• Refactor and Migrate to Razor Pages

Case Studies: Large ASP.NET Web Forms Product

- Update your Web Forms applications to use Model Binding as much as possible, which will provide you with a cleaner architecture for shared models, services, dependency injection, and unit tests
- Refactor Web Forms to MVC where possible, as the MVC model is much easier to migrate to ASP.NET Core
- Build new development in ASP.NET Core using cookie sharing and shared services to interoperate
- Continually build and update a roadmap as you go, focused on business value (e.g. ability to migrate services to Docker, improved performance for key services)

Case Studies: Large ASP.NET Web Forms Product

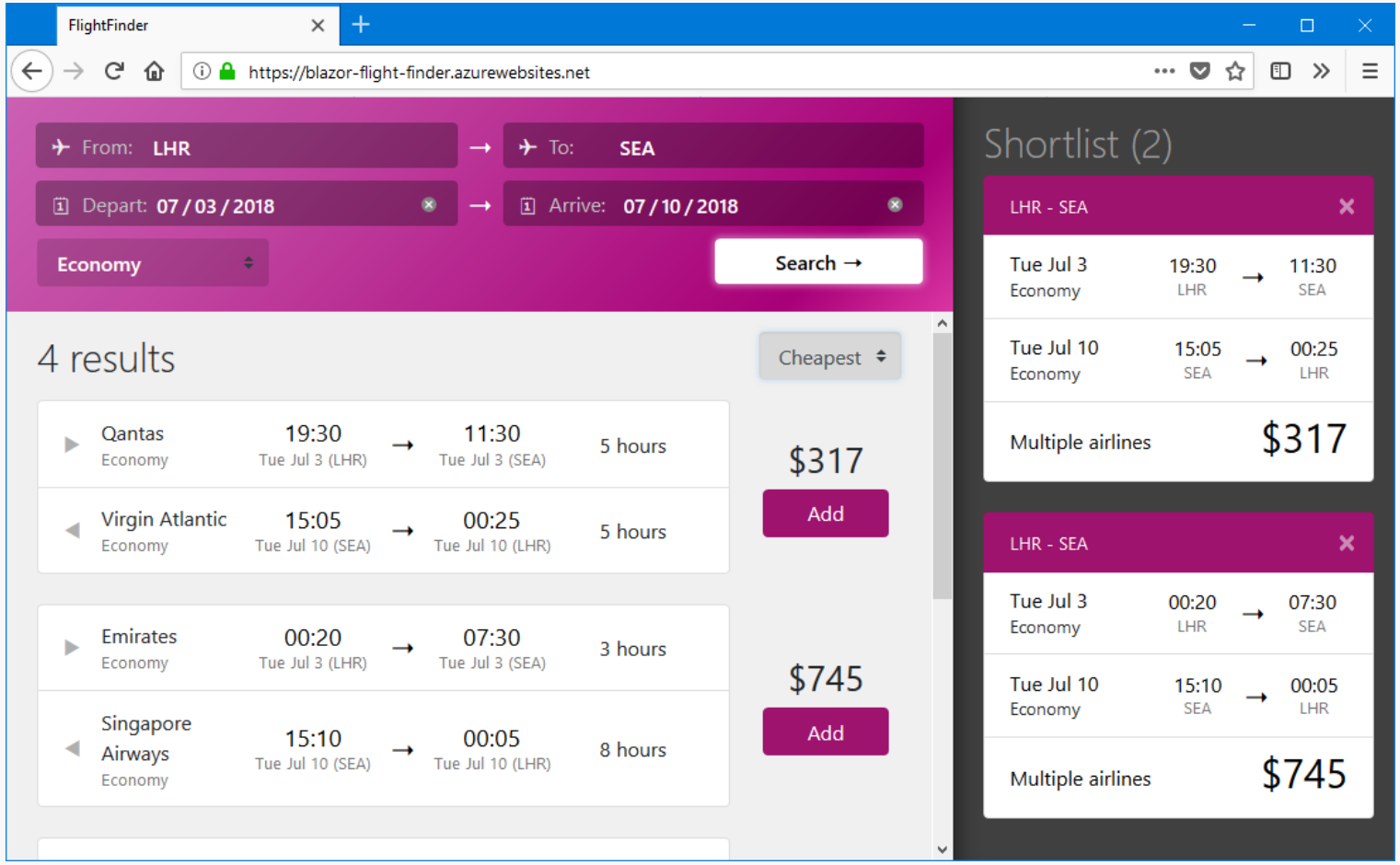
So in all honesty I'd ***move authentication out to a subsystem*** *all on its own using* ***Identity Server****. Then you can use* ***OpenID in the WebForms*** *app and* ***ODIC in the ASP.NET Core app*** *(and if you ever head down that route, mobile apps too). That decoupling means you don't need cookie sharing, which is a limited solution at best.*

Case Study: ASP.NET Web Forms Component-based System


```
@viewModel DotvvmDemo.CalculatorViewModel, Dotvvm

<p>
    Enter the first number:
    <dot:TextBox Text="{value: Number1}" />
</p>
<p>
    Enter the second number:
    <dot:TextBox Text="{value: Number2}" />
</p>
<p>
    <dot:Button Text="Calculate" Click="{command: Calculate()}" />
</p>
<p>
    The result is: {{value: Result}}
</p>
```

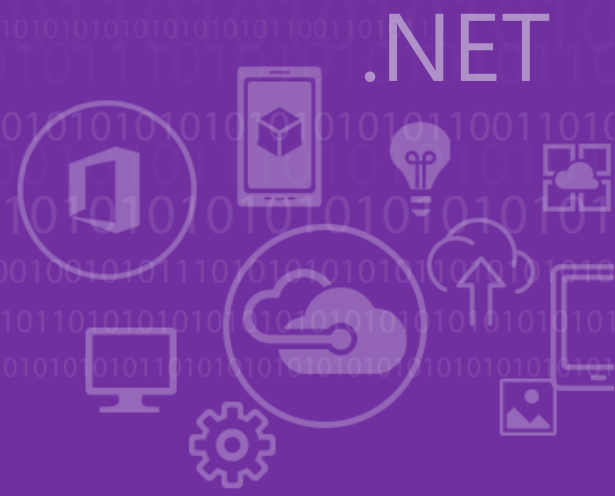
https://www.dotvvm.com/

Case Study: ASP.NET Web Forms Component-based System



https://www.blazor.net/

Summary

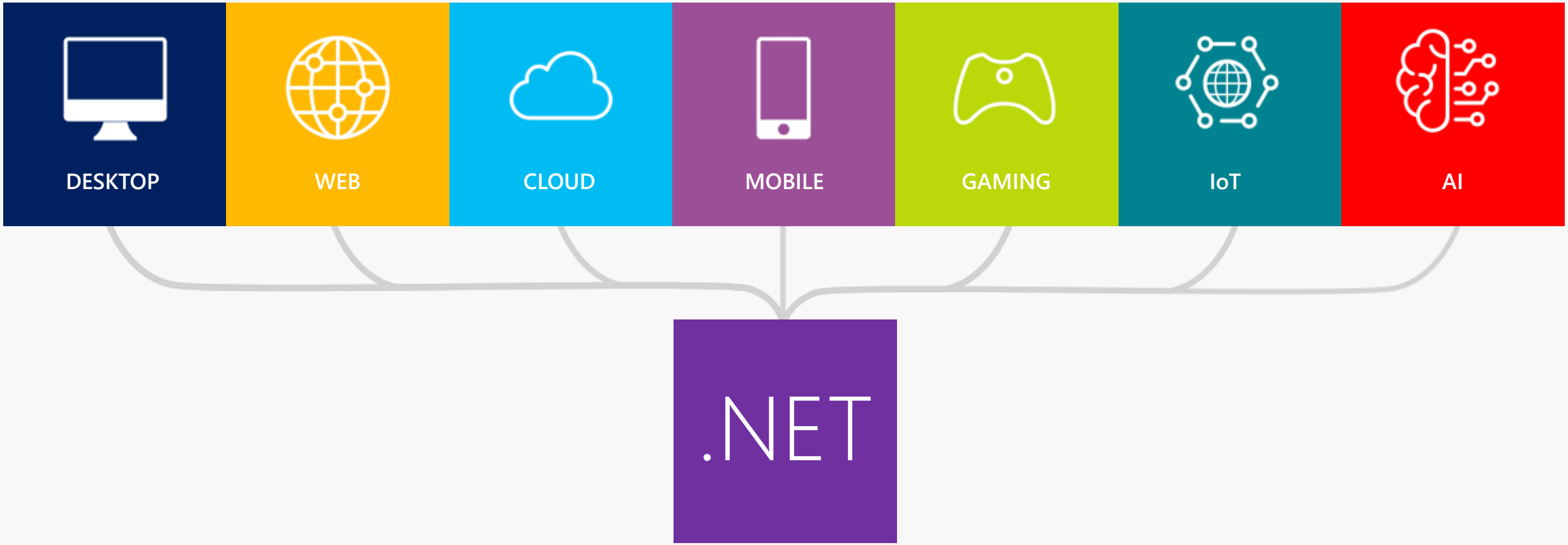


What We Covered

- Make an informed decision *if* and *what* to move
- Use the guidance and tools
- Use tests to handle a phased migration
- Divide and conquer
  - Business Logic -> .NET Standard
  - Windows Apps -> .NET Core 3.0
  - Web Apps -> API / Razor Pages

https://gist.github.com/jongalloway/4003fc3484bb9f22ffcee61464cc2de8

Your platform for building **anything**



www.dot.net