Understanding .NET Framework

Microsoft's .NET Framework is a development platform that includes a **Common Language Runtime** (**CLR**) that manages the execution of code, and provides a rich library of classes to build applications.

Microsoft designed .NET Framework to have the possibility of being cross-platform, but Microsoft put their implementation effort into making it work best with Windows.

Practically speaking, .NET Framework is Windows-only, and a legacy platform.

Understanding the Mono and Xamarin projects

Third parties developed a .NET implementation named the Mono project that you can read more about at the following link:
http://www.mono-project.com/

Mono is cross-platform, but it fell well behind the official implementation of .NET Framework. It has found a niche as the foundation of the **Xamarin** mobile platform.

Microsoft purchased Xamarin in 2016 and now gives away what used to be an expensive Xamarin extension for free with Visual Studio 2017. Microsoft renamed the **Xamarin Studio** development tool to **Visual Studio for Mac**, and has given it the ability to create ASP.NET Core Web API services. Xamarin is targeted at mobile development and building cloud services to support mobile apps.

Understanding .NET Core

Today, we live in a truly cross-platform world. Modern mobile and cloud development has made Windows a much less important operating system. So, Microsoft has been working on an effort to decouple .NET from its close ties with Windows.

While rewriting .NET to be truly cross-platform, Microsoft has taken the opportunity to refactor .NET to remove major parts that are no longer considered core.

This new product is branded as .NET Core, which includes a cross-platform implementation of the CLR known as CoreCLR, and a streamlined library of classes known as CoreFX.

Scott Hunter, Microsoft Partner Director Program Manager for .NET, says, "Forty percent of our .NET Core customers are brand-new developers to the platform, which is what we want with .NET Core. We want to bring new people in."

The following table shows when important versions of .NET Core were released, and Microsoft's schedule for the next major release:

Version	Released
.NET Core RC1	November 2015
.NET Core 1.0	June 2016
.NET Core 1.1	November 2016
.NET Core 1.0.4 and .NET Core 1.1.1	March 2017
.NET Core 2.0	August 2017
.NET Core for UWP in Windows 10 Fall Creators Update	October 2017
.NET Core 2.1	Q1 2018

.NET Core is much smaller than the current version of .NET Framework because a lot has been removed.

For example, Windows Forms and Windows Presentation Foundation (WPF) can be used to build graphical user interface (GUI) applications, but they are tightly bound to Windows, so they have been removed from .NET Core. The latest technology used to build Windows apps is Universal Windows Platform(UWP), and UWP is built on a custom version of .NET Core.

ASP.NET Web Forms and Windows Communication Foundation (WCF) are old web application and service technologies that fewer developers choose to use for new development projects today, so they have also been removed from .NET Core. Instead, developers prefer to use ASP.NET MVC and ASP.NET Web API. These two technologies have been refactored and combined into a new product that runs on .NET Core, named ASP.NET Core.

The Entity Framework (EF) 6 is an object-relational mapping technology to work with data stored in relational databases such as Oracle and Microsoft SQL Server. It has gained baggage over the years, so the cross-platform version has been slimmed down and named Entity Framework Core.

In addition to removing large pieces from .NET Framework to make .NET Core, Microsoft has componentized .NET Core into NuGet packages: small chunks of functionality that can be deployed independently.

Microsoft's primary goal is not to make .NET Core smaller than .NET Framework. The goal is to componentize .NET Core to support modern technologies and to have fewer dependencies, so that deployment requires only those packages that your application n

Understanding .NET Standard

The situation with .NET today is that there are three forked .NET platforms, all controlled by Microsoft:

- .NET Framework
- .NET Core
- Xamarin

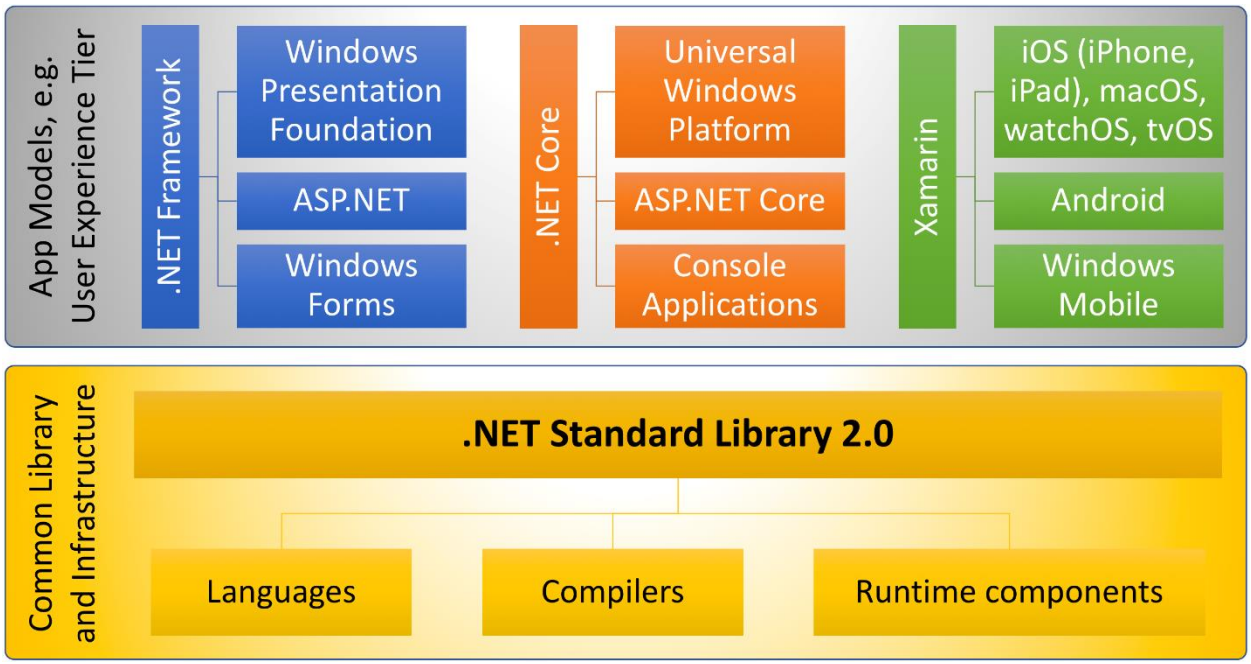
Each have different strengths and weaknesses because they are designed for different scenarios. This has led to the problem that a developer must learn three platforms, each with annoying quirks and limitations.

So, Microsoft defined .NET Standard 2.0: a specification for a set of APIs that all .NET platforms must implement. You cannot install .NET Standard 2.0 in the same way that you cannot install HTML5. To use HTML5, you must install a web browser that implements the HTML5 specification. To use .NET Standard 2.0, you must install a .NET platform that implements the .NET Standard 2.0 specification.

.NET Standard 2.0 is implemented by the latest versions of .NET Framework, .NET Core, and Xamarin. .NET Standard 2.0 makes it much easier for developersto share code between any flavor of .NET.

For .NET Core 2.0, this adds many of the missing APIs that developers need to port old code written for .NET Framework to the cross-platform .NET Core. However, some APIs are *implemented*, but throw an exception to indicate to a developer that they should not actually be used! This is usually due to differences in the operating system on which you run .NET Core.

The following diagram summarizes how the three variants of .NET (sometimes known as App Models) will share the common .NET Standard 2.0 and infrastructure:



Understanding .NET Native

Another .NET initiative is .NET Native. This compiles C# code to native CPU instructions **ahead-of-time** (**AoT**), rather than using the CLR to compile **intermediate language** (**IL**) code **just-in-time** (**JIT**) to native code later.

.NET Native improves execution speed and reduces the memory footprint for applications. It supports the following:

UWP apps for Windows 10, Windows 10 Mobile, Xbox One, HoloLens, and **Internet of Things** (**IoT**) devices such as Raspberry Pi

Server-side web development with ASP.NET Core

Console appications for use on the command line

Comparing .NET technologies

The following table summarizes and compares .NET technologies:

Technology	Feature set	Compiles to	Host OSes
.NETFramework	Both legacy and modern	IL code	Windows only
Xamarin	Mobile only	IL code	iOS, Android, Windows Mobile
.NET Core	Modern only	IL code	Windows, macOS, Linux
.NET Native	Modern only	Native code	Windows, macOS, Linux