Rainer Stropek | time cockpit

C#-Revolution

Your Host

Rainer Stropek

- Developer, Entrepreneur
- MVP for Azure, MVP for Dev Technologies, MS Regional Director, Trainer at IT-Visions

Contact

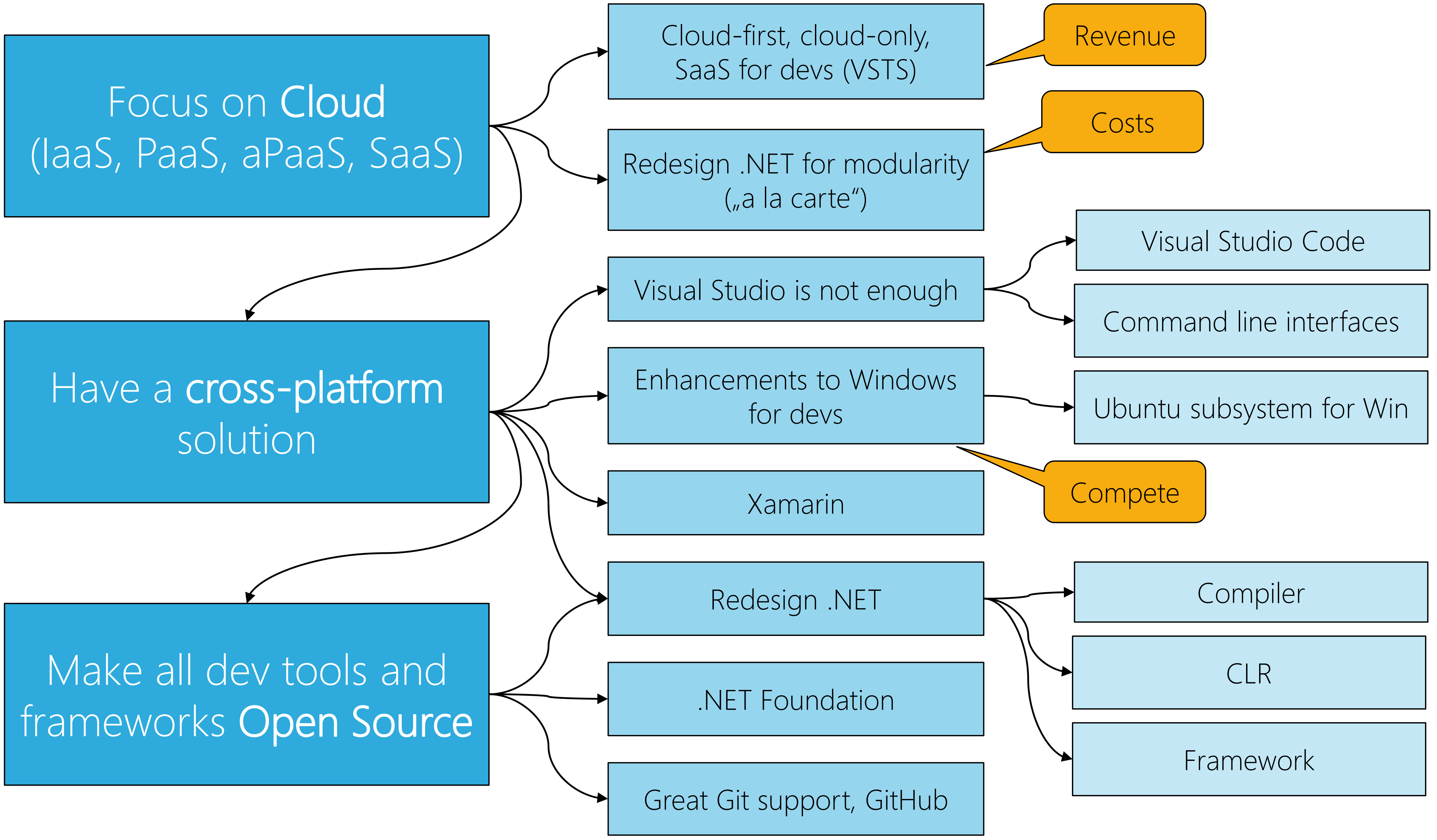
- software architects gmbh
  rainer@timecockpit.com
  Twitter: @rstropek

Agenda

C# und .NET machen einen radikalen Wandel durch. Open Source, Plattformunabhängigkeit, grundlegendes Redesign, neue Compilerplattform – als C#-Entwicklerinnen und -Entwickler gibt es viel Neues zu lernen. Der BASTA!-C#-Workshop von Rainer Stropek ist eine gute Gelegenheit, sich einen Tag Zeit zu nehmen, um auf den neuesten Stand zu kommen. Im Workshop werden unter anderem folgende Themen behandelt:

- Neuerungen in C# und Visual Studio
- Die neue .NET Runtime
- dotnet CLI
- Die neue .NET-Ausführungsumgebung
- Anwendungsbeispiele in ASP.NET Core (Fokus liegt auf der Sprache und .NET-Grundlagen, nicht auf ASP.NET)
- Neue Tools und Libraries.

In der bewährten Art und Weise wird sich Rainer Stropek im Workshop auf Codebeispiele statt Slides konzentrieren.


Focus on Cloud (IaaS, PaaS, aPaaS, SaaS)
Cloud-first, cloud-only, SaaS for devs (VSTS)
Revenue
Redesign .NET for modularity („a la carte")
Costs
Have a cross-platform solution
Visual Studio is not enough
Visual Studio Code
Command line interfaces
Enhancements to Windows for devs
Ubuntu subsystem for Win
Compete
Xamarin
Make all dev tools and frameworks Open Source
Redesign .NET
Compiler
CLR
Framework
.NET Foundation
Great Git support, GitHub

.NET Core

Why .NET Core?

Refactor .NET Framework

- Establish a Standard Library for the various incarnations of .NET
- .NET Core is not 100% compatible with .NET 4.x (details)

Make it a real cross-platform solution

- Windows, Mac OS, Linux (details in .NET Core Roadmap)

Make it open source

- A .NET Foundation project
- MIT License

Details: https://docs.microsoft.com/dotnet/

Components of .NET Core

.NET Runtime (CoreCLR)
- CoreCLR includes Base Class Library (BCL)

.NET Core Foundation Libraries (CoreFX)

.NET Command Line Tools (.NET CLI)
- Including the **`dotnet`** application host

Cross-Platform Compiler (Roslyn)

Status of .NET Core

- .NET Core 1.1 is RTM
  - Download current version
  - 2.0 is scheduled for Summer 2017 (roadmap, overview in docs)
- Visual Studio Tools are in preview
  - Wait for VS2017 (March 2017)
- C# is RTM
  - VB and F# are coming
- X64 Support
  - X86, X64 support on Windows
  - X64 support on many Linux distros

See also: https://github.com/dotnet/core/blob/master/roadmap.md

What can you build?

Console applications

ASP.NET Core applications

UWP applications

Xamarin Forms applications

See also: https://github.com/dotnet/core/blob/master/roadmap.md

Where to get .NET Core?

.NET Core landing page

- With Visual Studio tools (Visual Studio prerequisites)
- Command-line tools (with your own editor, e.g. VSCode, download)

.NET Install Script (details, download)

- You have to care for the prerequisites

NuGet

- Packages and Metapackages

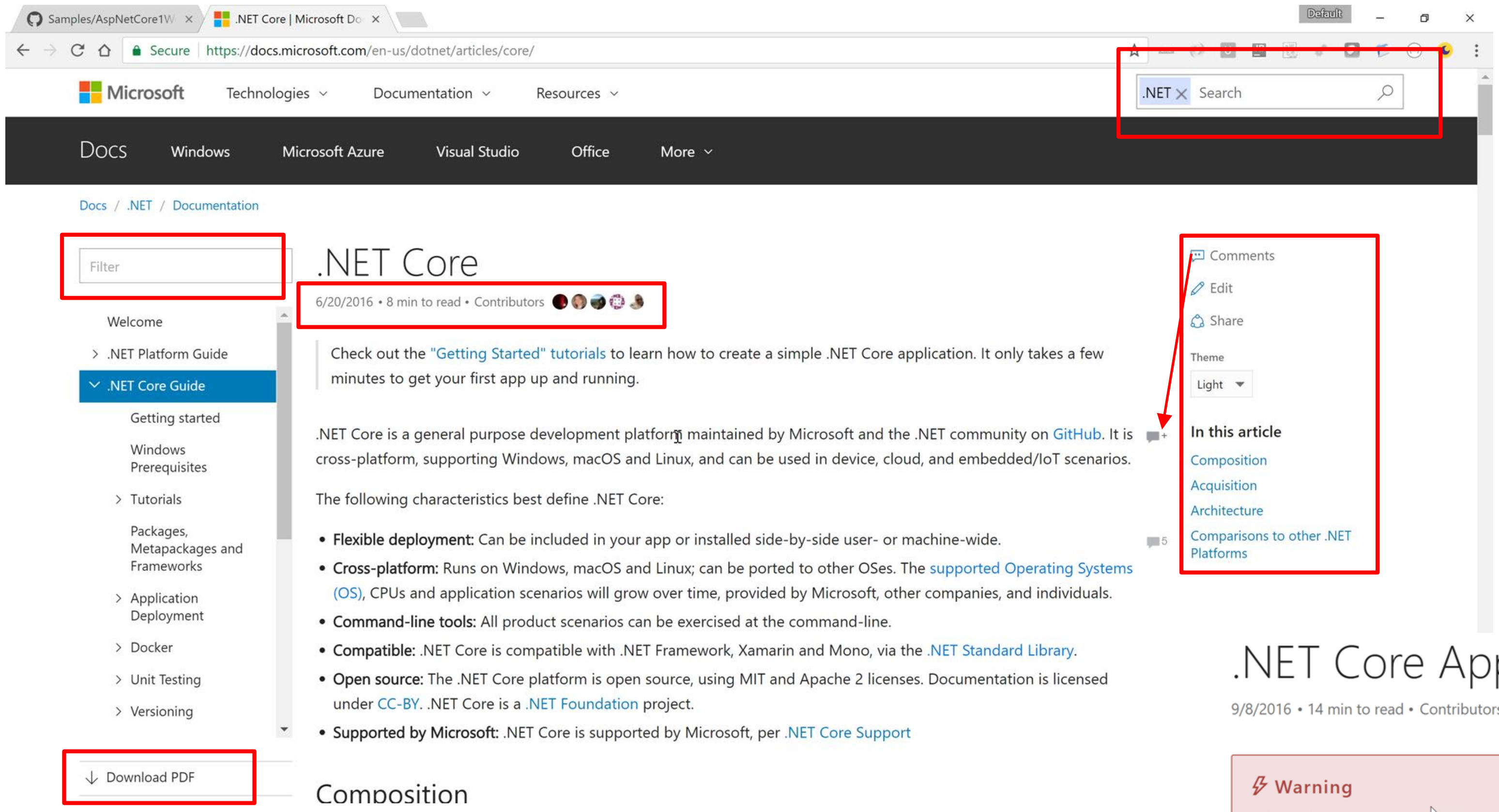
Docker: `microsoft/dotnet` image (details)

.NET Core Source Browser

See also: https://github.com/dotnet/core/blob/master/roadmap.md

Getting Help

New https://docs.microsoft.com



.NET Core

6/20/2016 • 8 min to read • Contributors

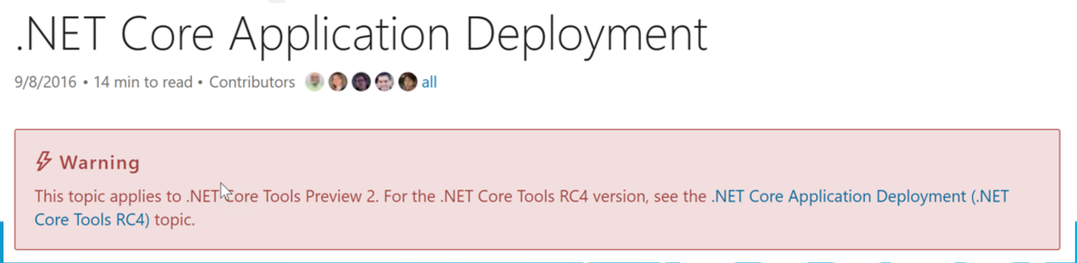
Check out the "Getting Started" tutorials to learn how to create a simple .NET Core application. It only takes a few minutes to get your first app up and running.

.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.

The following characteristics best define .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 OSes. 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.
- **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. Documentation is licensed under CC-BY. .NET Core is a .NET Foundation project.
- **Supported by Microsoft:** .NET Core is supported by Microsoft, per .NET Core Support

Composition



.NET Core Application Deployment

9/8/2016 • 14 min to read • Contributors all

Warning

This topic applies to .NET Core Tools Preview 2. For the .NET Core Tools RC4 version, see the .NET Core Application Deployment (.NET Core Tools RC4) topic.

Packages, Metapackages and Frameworks

Demo

Create console app with CLI

Analyze `.csproj`

Discuss `.csproj` reference

Run app

Further readings

- More about cross-platform libraries
- MSBuild Project File Schema Reference
- Creating new templates

https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/10-console-hello-world

.csproj

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp1.1</TargetFramework>
  </PropertyGroup>
</Project>
```

For executable, not present for class libraries

Target Framework (1.0 or 1.1)

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netstandard1.6</TargetFramework>
  </PropertyGroup>
</Project>
```

Class library based on .NET Standard 1.6

See also MSBuild Project File Schema Reference

Solutions

Demo

Create solution: **`dotnet new sln`**

Add proj.: **`dotnet sln add …`**

Create solution in VS2017

- .NET Standard class library with Json.NET
- .NET Framework console app with reference

Further readings

- [.NET Core Tools MSBuild]()

https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/25-project-references

Cross-platform

Demo

Run app on Linux using Docker

.NET CLI

.NET Core CLI

`dotnet` command

- `new` – create project
- `migrate` – migrates from Preview 2 to RC4 (*project.json* → *.csproj*)
- `restore` – restore dependencies
- `run` – run source code without explicit compile
- `build` – builds project and dependencies
- `test` – runs unit tests
- `pack` – packs code into a NuGet package
- `publish` – packs the app and dependencies for publishing

https://docs.microsoft.com/en-us/dotnet/articles/core/preview3/tools/index

`dotnet run`

Run application from the source code

- Use **`dotnet`** without any command to run a built DLL

Uses **`dotnet build`** in the background

Important parameters

- `--framework`
- `--configuration <Debug|Release>`

https://docs.microsoft.com/en-us/dotnet/articles/core/preview3/tools/dotnet-run

Deployment (`dotnet publish`)

Framework-dependent deployment

- Shared system-wide version of .NET Core must be present on target system
- DLLs are launched using **`dotnet`**
- DLLs are portable

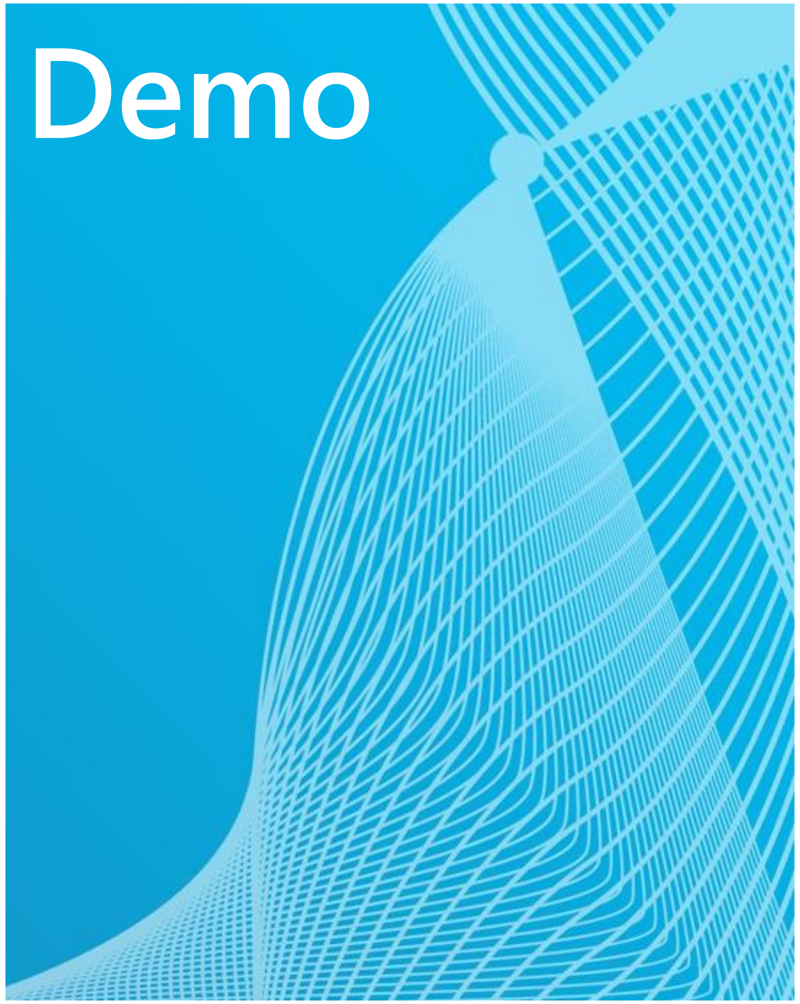
Self-contained deployment

- No prerequisites on target system necessary
- Does *not* contain native prerequisites
- Results in an platform-specific executable

Optional: Use CrossGen for native image generation

https://docs.microsoft.com/en-us/dotnet/articles/core/preview3/deploying/index

Self-contained Deployment



Create self-contained sample

- See following slides

Build and publish SCD

```
dotnet publish -c release
dotnet publish -c release -r win10-x64
dotnet publish -c release -r debian.8-x64
```

Runtime Identifier (RID) (details)

Release instead of debug version (need not ship PDBs)

https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/27-self-contained

Self-contained Deployment

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

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp1.1</TargetFramework>
  </PropertyGroup>

  <PropertyGroup>
    <RuntimeIdentifiers>win10-x64;debian.8-x64</RuntimeIdentifiers>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Newtonsoft.Json" Version="9.0.1" />
  </ItemGroup>

</Project>
```

Details: https://docs.microsoft.com/en-us/dotnet/articles/core/rid-catalog

Versioning

Versioning

Framework version changes when APIs are added

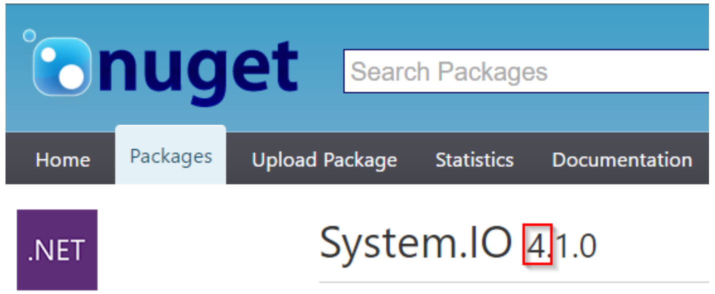
No implementation → no patch numbers

Example: **`netcoreapp1.0`**

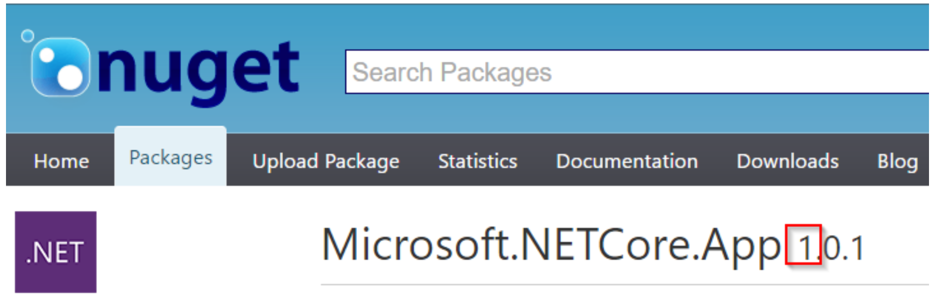
Package versions

`System.*` packages use 4.x numbers (overlap with .NET Framework)

Packages without overlapping with .NET Framework → 1.x



System.IO 4.1.0



Microsoft.NETCore.App 1.0.1

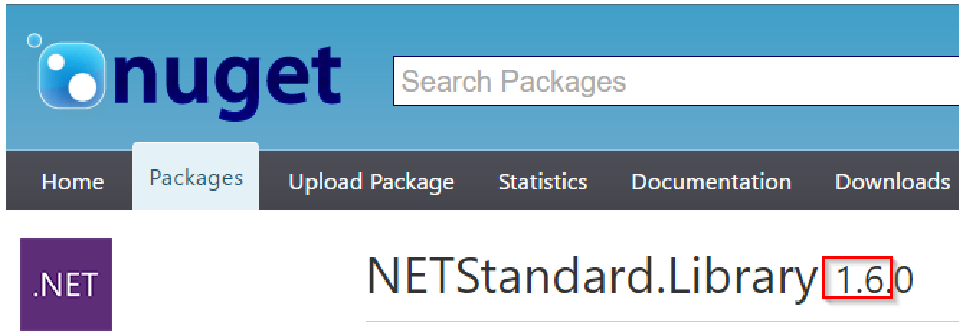
https://docs.microsoft.com/en-us/dotnet/articles/core/versions/index

Versioning

.NET Standard Library

Versioning independent of any .NET runtime, applicable to multiple runtimes

1.6 for .NET Core 1.0



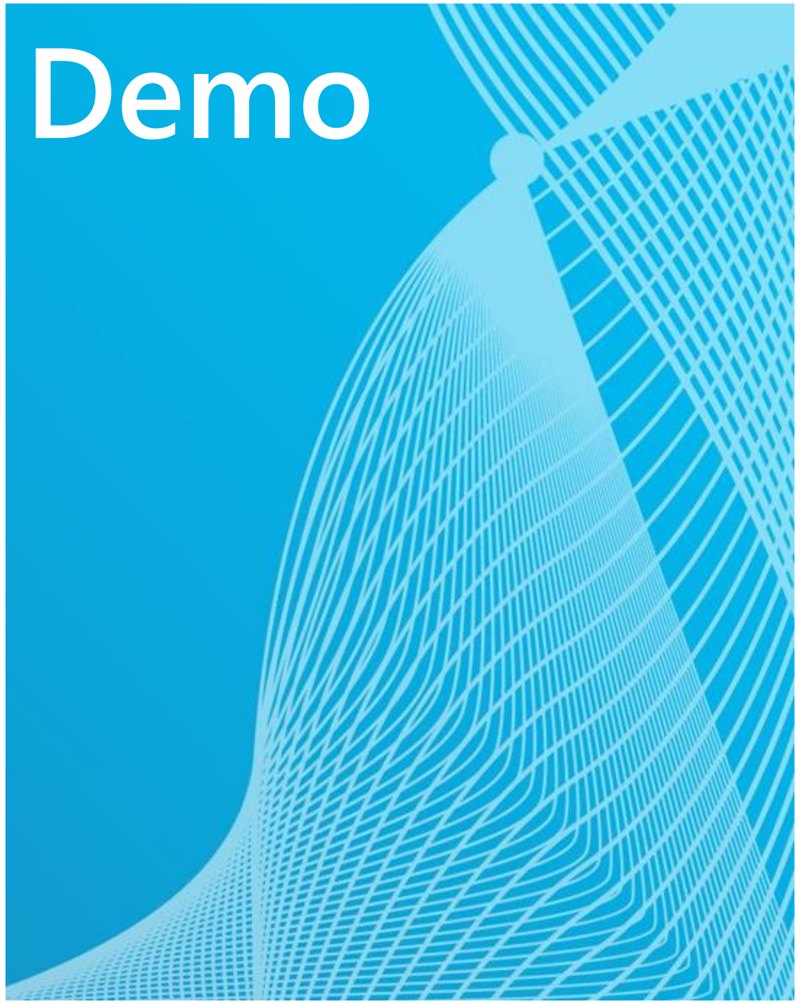
NETStandard.Library 1.6.0

[Examples]

https://docs.microsoft.com/en-us/dotnet/articles/core/versions/index

Libraries

Libraries



Shared files

Libraries

Creating NuGet packages

```
dotnet pack
```

Further readings

- [More about cross-platform libraries]
- [Tools for porting code from .NET Framework]

.NET Standard Library

Why a standard library?

CLR (CLI) has already been standardized (ECMA 334)

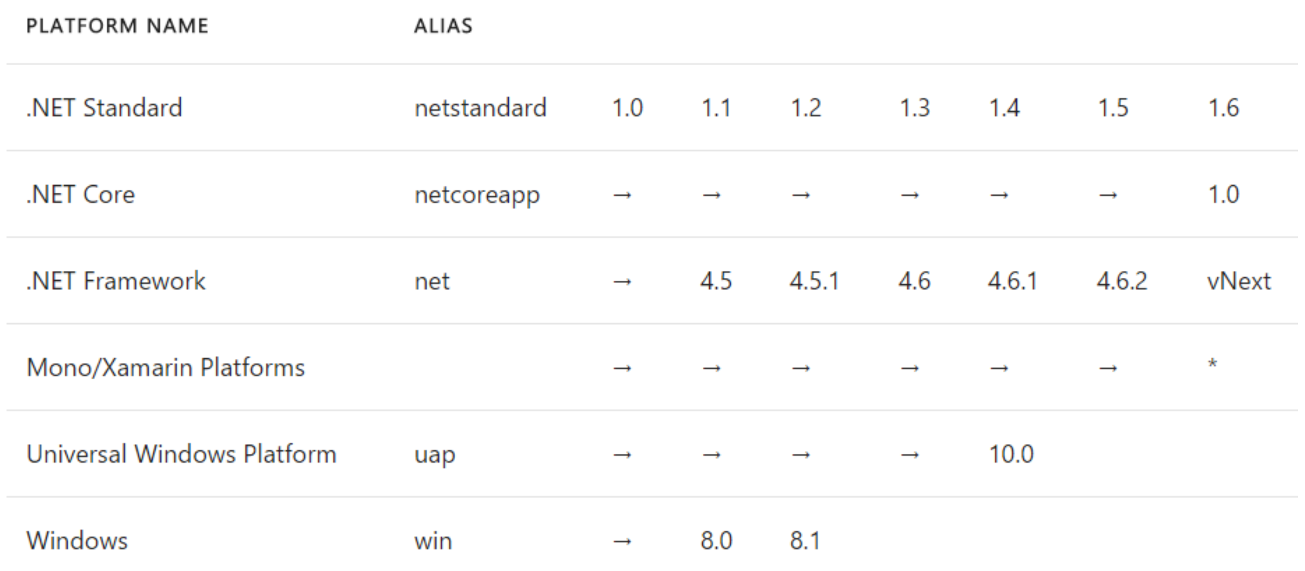
- No standardized BCL prior to .NET Core

Goal: Standard BCL API for all .NET platforms

- Easier to create portable libraries
- Reduce conditional compilation

What about PCLs?

- Well defined API instead of just intersection of platforms
- Better versioning
- Overlapping PCL profiles (details)



PLATFORM NAME	ALIAS							
.NET Standard	netstandard	1.0	1.1	1.2	1.3	1.4	1.5	1.6
.NET Core	netcoreapp	→	→	→	→	→	→	1.0
.NET Framework	net	→	4.5	4.5.1	4.6	4.6.1	4.6.2	vNext
Mono/Xamarin Platforms		→	→	→	→	→	→	*
Universal Windows Platform	uap	→	→	→	→	10.0		
Windows	win	→	8.0	8.1				

Details: https://docs.microsoft.com/en-us/dotnet/articles/standard/library

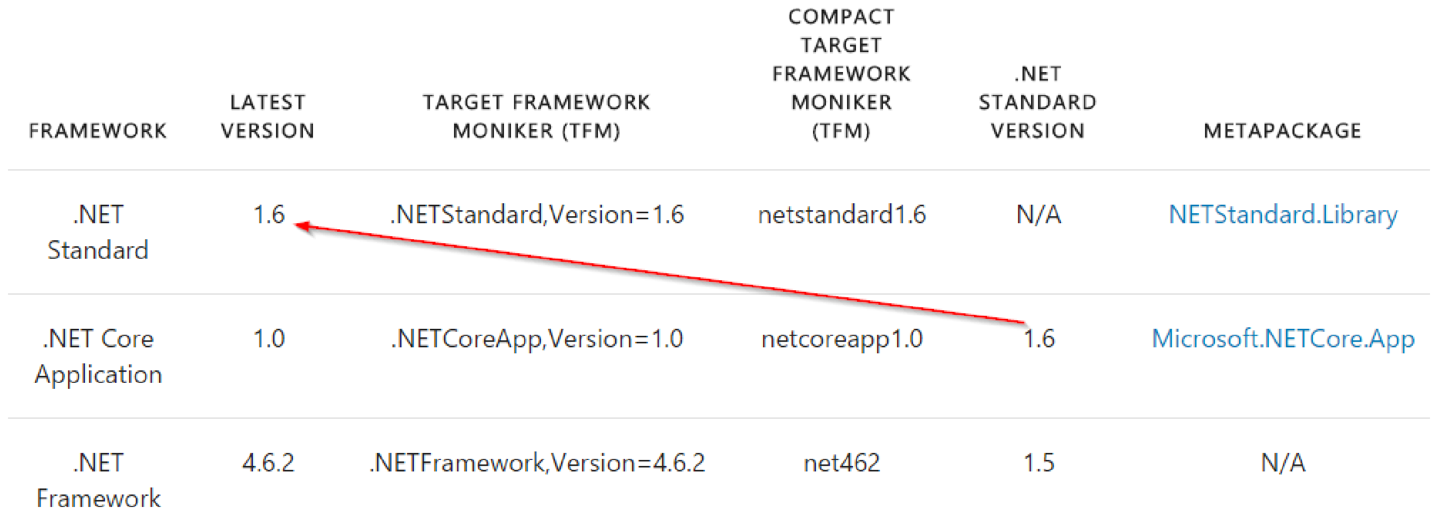
.NET Standard Library

Standard APIs defined as empty C# classes

Example: [`ref` folder in `System.Runtime`]

`NETStandard.Library` ([NuGet])

Metapackage for .NET Standard Library



FRAMEWORK	LATEST VERSION	TARGET FRAMEWORK MONIKER (TFM)	COMPACT TARGET FRAMEWORK MONIKER (TFM)	.NET STANDARD VERSION	METAPACKAGE
.NET Standard	1.6	.NETStandard,Version=1.6	netstandard1.6	N/A	NETStandard.Library
.NET Core Application	1.0	.NETCoreApp,Version=1.0	netcoreapp1.0	1.6	Microsoft.NETCore.App
.NET Framework	4.6.2	.NETFramework,Version=4.6.2	net462	1.5	N/A

Details: https://docs.microsoft.com/en-us/dotnet/articles/standard/frameworks

ASP.NET Core 1 Basics

Practical use of .NET Core

Minimal ASP.NET Core 1

Demo

- ASP.NET Pipeline
- Discuss "a la carte" framework
  - Add static files (sample)
- Kestrel
  - Windows, Linux with Docker
- Visual Studio Code
- Further readings
  - Building middlewares

https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/50-simplest-aspnet

Walkthrough VS “File – New – Project”

Demo

Create web project in VS2015

Walkthrough

- Servers (IIS and Kestrel)
- Environments
- Adding MVC

101 for ASP.NET Core 1

Application Startup

- `Main` Method
- `Startup` class with `ConfigureServices` (DI) and `Configure` (Pipeline)

Static Files

Environments

Servers

- IIS, Kestrel

Configuration

No `web.config` anymore

Key/value pair settings from different providers

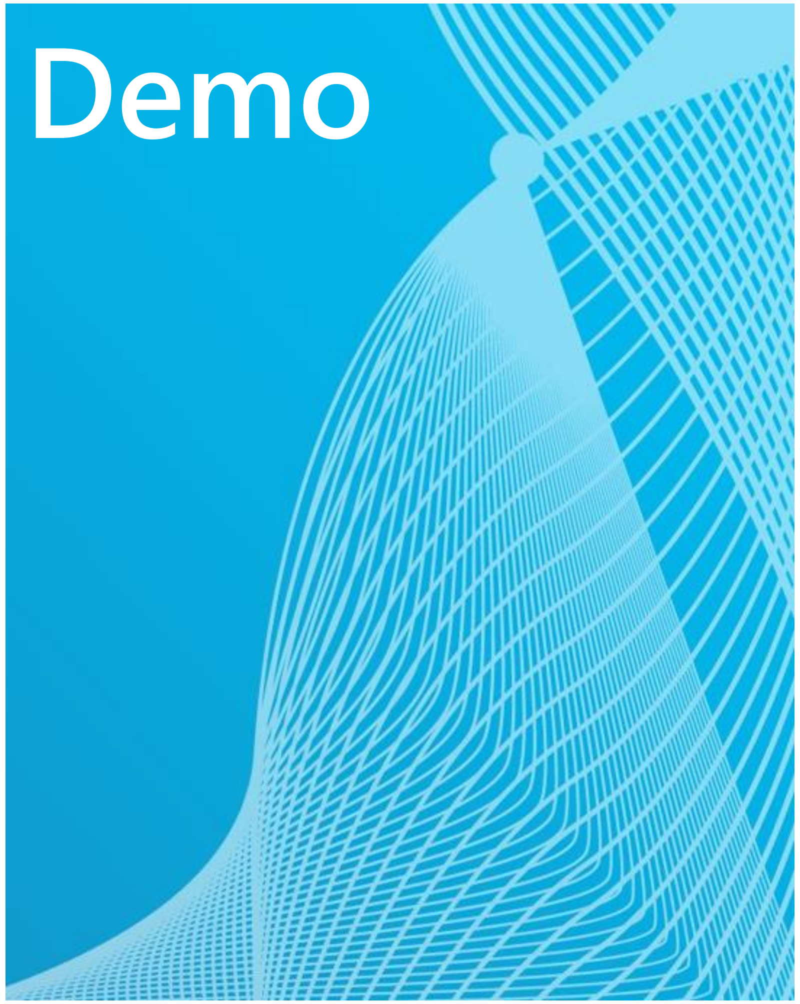
- E.g. memory, environment variables, JSON, INI, XML

Extensible

- Details about writing custom providers

Options pattern for DI integration

Configuration



In-memory configuration

JSON configuration

Configuration via command line

Configuration with environment variables

Options pattern

See practical use in AppInsights

https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/55-configuration/

Logging

Support for logging built into ASP.NET Core

Various logger built in

- E.g. console, NLog

Details about logging

Consider using Application Insights

- Getting started with AppInsights in ASP.NET Core

Logging

Demo

- JSON file to configure logging
  - .NET Core Logging
  - AppInsights
- Custom logging
- AppInsights portal

https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/58-logging/

Dependency Injection

Support for DI built into ASP.NET Core

- Details about DI

Framework-provided services and your own services

Service Lifetime

- Transient, Scoped, Singleton, Instance

Default container can be replaced (details)

Dependency Injection

Demo

Setting up DI

Service Lifetime

https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/60-di-scopes/

.NET Core Automation

Test, build, and release automation

CI with .NET Core apps

VSTS supports building and publishing .NET Core apps

- Details

Azure App Services supports .NET Core apps

- Kudu-support for .NET Core

Ready-made Docker image with `Dockerfile`

- `microsoft/dotnet`

Build Automation

Demo

Build and deploy .NET Core in VSTS

https://www.visualstudio.com/en-us/docs/build/apps/aspnet/aspnetcore-to-azure

Dockerfile for .NET Core app

Demo

Unit Testing

- .NET Core supports multiple test frameworks
  - E.g. XUnit, MSTest
  - Compare XUnit and MSTest

Unit Testing

Demo

Create and run library with tests

- XUnit (sample)
- MSTest (sample)

Run tests with **`VSTest.Console.exe`**

```
vstest.console.exe project.json
/UseVsixExtensions:true /logger:trx
```

Project setup

- Folders, **`project.json`**

https://blogs.msdn.microsoft.com/visualstudioalm/2016/09/01/announcing-mstest-v2-framework-support-for-net-core-1-0-rtm/

C# 7

Live Coding; sample code see
https://github.com/rstropek/Samples/tree/master/CSharp7

Thank you for coming!

Questions?