



Da .NET  
a .NET Core

19 LUGLIO 2016

Raffaele Rialdi

 @raffaeler

 raffaeler@vevy.com

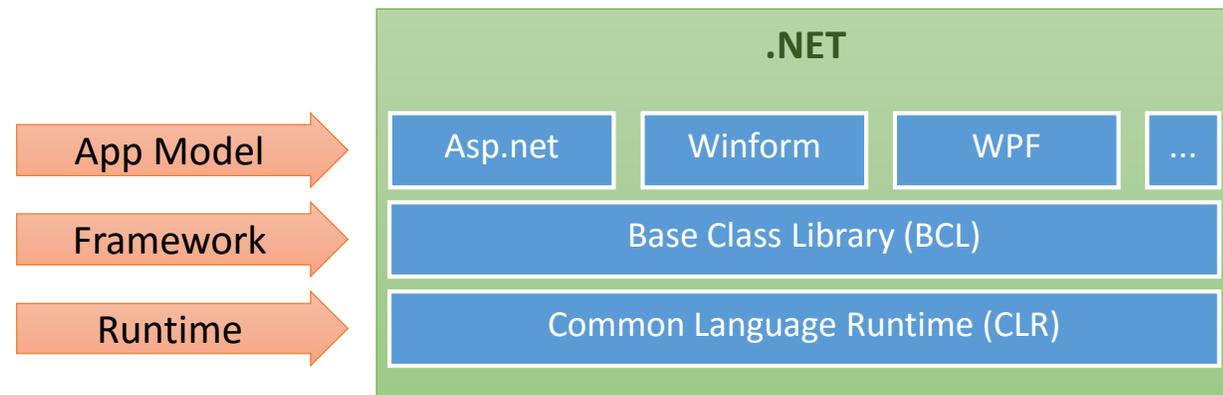
 <http://iamraf.net>



# Il Framework .NET fino ad oggi

- 15 anni, 1.8 miliardi di installazioni
- Diverse codebase/compilazioni
  - Desktop, Silverlight (Intel - Windows)
  - Compact Framework (ARM – Win CE)
  - Micro Framework (ARM)
- Ma anche le versioni .NET non-MS
  - Mono / Xamarin
    - Linux, iOS, MacOS, ...
    - Intel e ARM

Ogni versione replica tutto lo stack

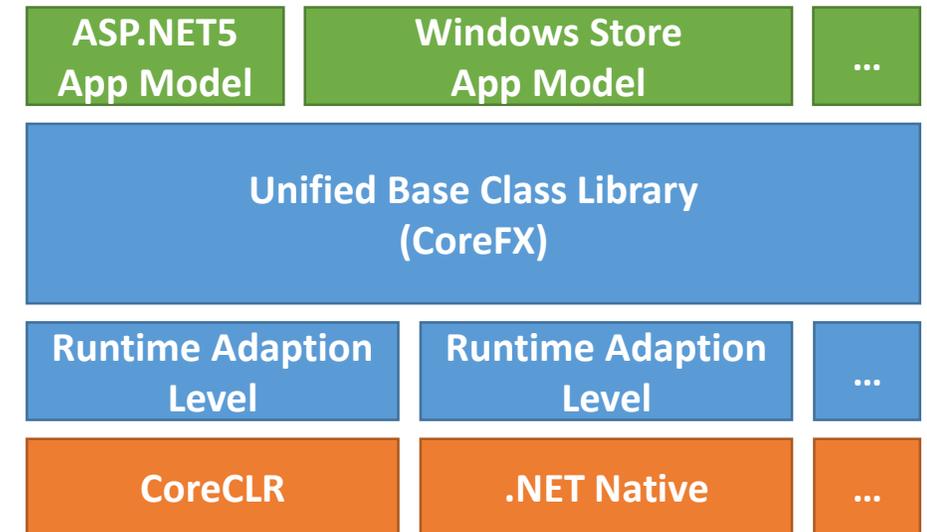


# Cos'è DotNetCore

- È un «fork» (derivato dagli stessi sorgenti) del Framework.NET
  - Nuovo runtime (Common Language Runtime – CLR)
  - Nuove librerie (CoreFX)
  - Alcune breaking changes per far funzionare la storia Cross-Platform
- Il goal di DotNetCore è di abilitare gli scenari Cross-Platform e Cross-Device
  - Supporto per Windows, Linux, MacOS e i container Docker
  - È ufficialmente supportato da Microsoft su tutte le piattaforme testate
- Gli scenari che si avvantaggiano da DotNetCore (attualmente) sono:
  - ASP.NET Core: nuovo stack ASP.NET, riscritto da zero
  - Universal Windows Platform: Windows Store Apps (cross-device)
  - Cloud App: Applicazioni e Microservizi su Azure
  - Console App: Il modo migliore per cominciare!

# L'evoluzione di DotNetCore

- Tutte le librerie che usano CoreFX possono essere usate da tutti gli Application Model (ASP.NET, UWP, Console)
- Ogni «Runtime Adaptation Layer» è specifico ad una data Platform / CPU
  - x86, x64, ARM, altre CPUs sono attese in futuro ...
- Questo modello è pensato per future estensioni
  - Nuovi Application Models
  - Nuovi OS / Platforms
  - Nuove varianti di CLR



# L'ecosistema dotnet

- **CoreCLR** è il **nuovo CLR** usato da ASP.NET e le Console Application
  - Usa il nuovo jitter « **RyuJIT** ». Include il GC e i servizi base e di Interop
  - <https://github.com/dotnet/coreclr>
- **CoreRT** è un altro CLR che usa "**.NET Native**" La UWP usa questo CLR
  - La toolchain .NET Native genera tutto il **codice nativo** ed è privo di tutte le dipendenze
  - UWP usa il CoreCLR durante il debug and .NET Native in Release (generato su Azure alla pubblicazione)
  - <https://github.com/dotnet/coreclr>
- **CoreFX** è la nuova **Base Class Library** ora chiamata «**.NET Core Foundational Libraries**»
  - Contiene codice Intermediate Language (IL) e librerie per specifici runtime. Può essere **usata da tutti i CLR**
  - <https://github.com/dotnet/corefx>
- DotNet Command Line Interface (CLI)
  - <https://github.com/dotnet/cli>

# Dalle Portable Class Libraries alla Standard Library

- PCL = intersezione delle funzionalità disponibili sulle piattaforme scelte
  - Espresse alla **compilazione**
  - In nuget identificate con i "moniker" Es: "portable-net45+sl5+win8+wpa81+wp8"
    - Vedi profiles al link <https://docs.nuget.org/create/targetframeworks>
  - NON funzioneranno con future platform. Sarà necessario ri-deployare.
- La "Standard Library" definisce i contratti della libreria di **runtime**
  - Assicura la compatibilità a livello binario
  - Ogni set viene identificato su nuget tramite "moniker". Es: "netstandard1.6"
  - La direttiva "imports" di Nuget consente di riusare le PCL nella Standard Library
    - Imports forza l'uso di una determinata libreria in assenza del moniker di riferimento

# La .NET Standard Library

- Definisce un set standard di API disponibile per tutti gli Application Model
  - Sono un set di Reference Assemblies (contratti, nessuna implementazione)
  - Sono definite nel repository di "CoreFX" (GitHub)
- La ".NET Platform" o ".NET Runtime" definisce uno specifico framework
  - Es: Framework.NET, Mono/Xamarin, Windows Phone, UWP, etc.
  - Ogni ".NET Platform" implementa una specifica versione di .NET Standard Library
- Le librerie (DLL) indicano la versione della ".NET Standard Library" come loro requisito
  - netstandard1.6, netstandard1.5, ... (più bassa è, maggiore sarà la compatibilità con le varie platform)
- Le App (asp.net, console, ...) indicano solo la ".NET Platform"
  - Che implicitamente corrisponde ad una versione della Standard Library
  - Es: netcoreapp1.0, uap10.0 , net452 , xamarinios

# .NET Migration tool

- Due versioni dello stesso tool
  - La versione "command line"
  - L'estensione di Visual Studio (fornisce più informazioni)
- Che informazioni ci fornisce?
  - Legge un assembly binario via reflection
  - Analizza le dipendenze
  - Spedisce un minimo set di informazioni ad un Web Service Microsoft
    - Il tool è usabile anche in modalità "offline"
  - Crea un report con la lista dei membri che non sono supportati dalle versioni di Framework richieste nell'analisi
- Supporta le versioni di .NET più comuni
  - .NET standard, .NET Core, Xamarin, Mono, Silverlight, etc.

# Nuget

- Può esistere un "Metapackage"
  - Specifica solo le dipendenze
- Può esistere un Reference Assembly
  - L'assembly contiene una cosa del tipo:

```
[assembly: TypeForwardedTo(typeof(Assembly))]  
[assembly: TypeForwardedTo(typeof(AssemblyContentType))]  
[assembly: TypeForwardedTo(typeof(AssemblyName))]  
[assembly: TypeForwardedTo(typeof(BindingFlags))]  
[assembly: TypeForwardedTo(typeof(ConstructorInfo))]
```

- Può contenere il codice "vero"

The image shows a screenshot of Visual Studio. The top part shows the Solution Explorer with a tree view for 'NetCoreLibrary' > 'References' > '.NETStandard,Version=v1.6' > 'NETStandard.Library (1.6.0)'. An orange box labeled 'Metapackage' points to this entry. Below it, the Properties window is open, showing the 'NETStandard.Library' Dependency. The table below shows the properties of this dependency:

Name	NETStandard.Library
Path	C:\Users\raffaeler\.nuget\packages\NETStandard.Library\1.6.0
Type	Package
Version	1.6.0

Below the Properties window, a file explorer view shows the contents of the folder 'C:\Users\raffaeler\.nuget\packages\NETStandard.Library\1.6.0'. The files listed are:

Name	Date modified	Type	Size
dotnet_library_license.txt	11-Jun-16 23:15	Text Document	10 KB
NETStandard.Library.1.6.0.nupkg	15-Jul-16 18:27	NUPKG File	8 KB
NETStandard.Library.1.6.0.nupkg.sha512	15-Jul-16 18:27	SHA512 File	1 KB
NETStandard.Library.nuspec	11-Jun-16 23:15	NUSPEC File	11 KB
ThirdPartyNotices.txt	11-Jun-16 23:15	Text Document	2 KB

A blue arrow points from the 'Type' property in the Properties window to the 'NETStandard.Library.1.6.0.nupkg' file in the file explorer. A red box highlights the 'Size' column in the file explorer table.

# Breaking changes

- AppDomains
  - Esiste a livello di infrastruttura ma non è più usabile in termini di API
  - Info di base reperibili via **AppContext** (che espone ad esempio **BaseDirectory**)
  - Per caricare dinamicamente gli assembly usare **AssemblyLoadContext**
  - Per isolare del codice usare i **container** (Linux o Windows container) ... Docker & C
- Remoting
  - Architettura problematica considerata obsoleta
- Binary Serialization
  - Esistono alternative valide in diverse librerie... ma forse verrà reinserita in un prossimo futuro
- Security
  - Code Access Security (CAS)
  - Security Transparency

# Nuget is king

- CoreFX (BCL) è distribuita via nuget
  - Una dll per ciascun namespace
  - Niente più GAC (niente più 'globali')
  - Il deploy è privato a ciascuna App
  - Ogni App ha la sua propria copia di tutte le dipendenze necessarie
- Le dll sono private
  - O sono parte della solution
  - O parte di un pacchetto nuget
    - I server Nuget possono essere privati
- La CLI di DotNet recupera i packages da Nuget

## App

```
{  
  "version": "1.0.0-*",  
  "buildOptions": {  
    "emitEntryPoint": true  
  },  
  
  "dependencies": {  
    "Microsoft.NETCore.App": {  
      "type": "platform",  
      "version": "1.0.0"  
    }  
  },  
  
  "frameworks": {  
    "netcoreapp1.0": {  
      "imports": "dnxcORE50"  
    }  
  }  
}
```

## Dll

```
{  
  "version": "1.0.0-*",  
  
  "dependencies": {  
    "NETStandard.Library": "1.6.0"  
  },  
  
  "frameworks": {  
    "netstandard1.6": {  
      "imports": "dnxcORE50"  
    }  
  }  
}
```

Demo reference

.net desktop / .net core

# DotNet CLI in pratica

<http://dot.net/>

- Installare la « CLI »  
<http://dot.net/>
- Creare una App  
`mkdir myapp`  
`cd myapp`  
`dotnet new`
- Scrivere il codice
  - ...
- Compilare e lanciare l'App  
`dotnet restore`  
`dotnet run`

```
C:\app1>dotnet --help
.NET Command Line Tools (1.0.0-preview2-003121)
Usage: dotnet [host-options] [command] [arguments] [common-options]

Arguments:
  [command]           The command to execute
  [arguments]         Arguments to pass to the command
  [host-options]      Options specific to dotnet (host)
  [common-options]   Options common to all commands

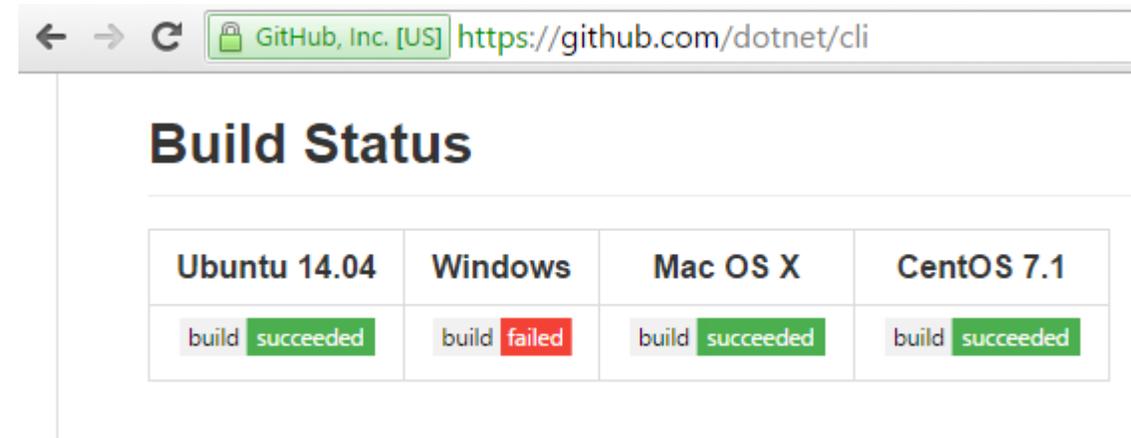
Common options:
  -v|--verbose       Enable verbose output
  -h|--help          Show help

Host options (passed before the command):
  -v|--verbose       Enable verbose output
  --version          Display .NET CLI Version Number
  --info             Display .NET CLI Info

Common Commands:
  new                Initialize a basic .NET project
  restore            Restore dependencies specified in the .NET project
  build              Builds a .NET project
  publish            Publishes a .NET project for deployment (including the runtime)
  run                Compiles and immediately executes a .NET project
  test              Runs unit tests using the test runner specified in the project
  pack               Creates a NuGet package
```

# Cross-Platform

- Le compilazioni notturne su GitHub includono diversi OS
  - Linux: CentOS 7.1, Debian 8.2, FreeBSD 10.1, openSUSE 13.2, RedHat 7.2, Ubuntu 14.04, Ubuntu 15.10
    - x64
  - MacOS: OSX 10.11
    - x64
  - Windows: > Windows 8.1
    - x64, ARM
- Non ci sono OS di "prima classe"
  - Quindi capita di vedere anche questo :)

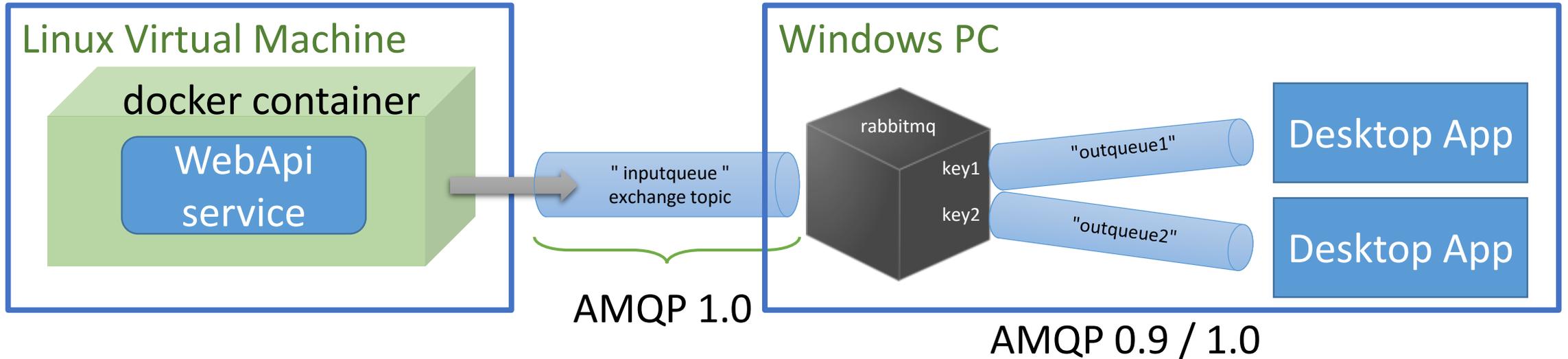


Ubuntu 14.04	Windows	Mac OS X	CentOS 7.1
build succeeded	build failed	build succeeded	build succeeded

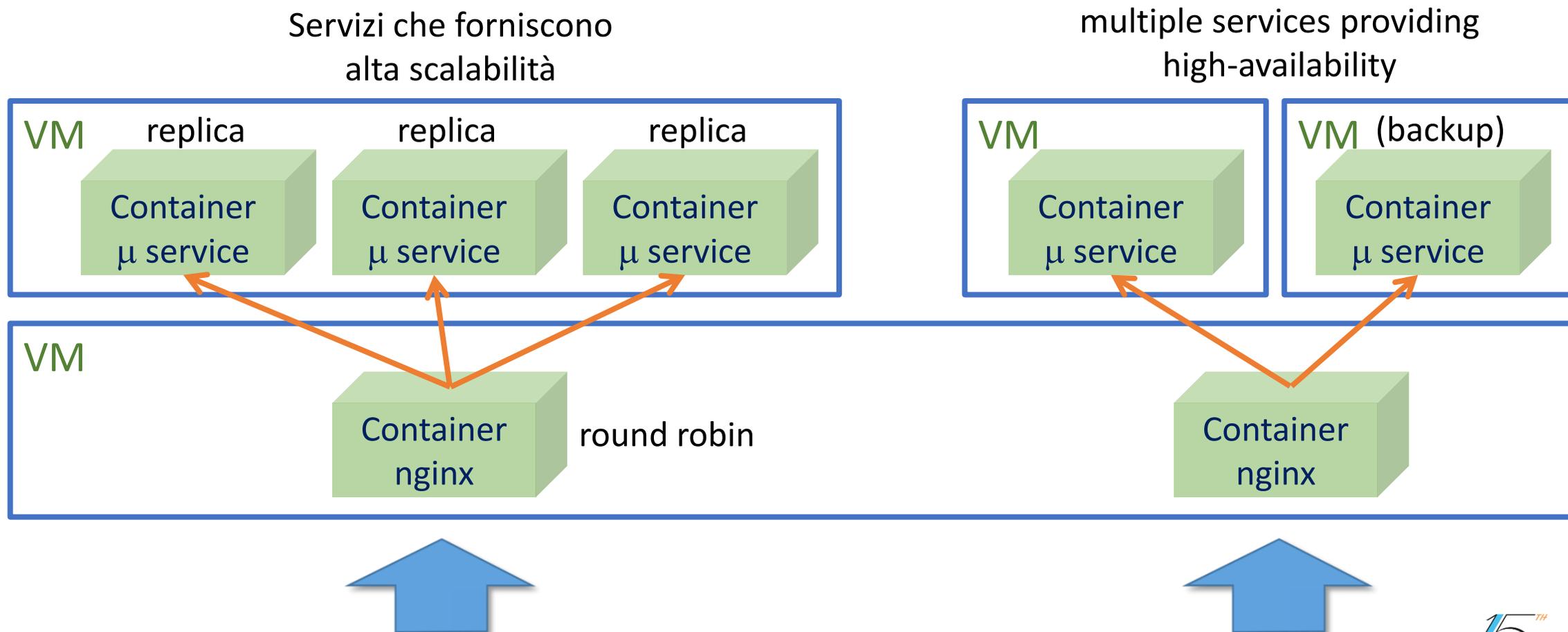
# .NET Native

- A partire dal Framework.NET 4.6 c'è un nuovo JIT compiler chiamato «RyuJIT»
  - È più veloce e produce codice ottimizzato (SIMD, Vectorization, etc.)
- .NET Native è una toolchain o compilatore IL «AOT» Ahead Of Time
  - La compilazione nativa usa il backend compiler di Visual C++
    - Una sorta di NGEN più evoluto
  - Il JIT compiler non è più necessario
    - La versione del CLR flavor che usa la toolchain .NET Native si chiama « CoreRT»
  - UWP è l'application model che usa .NET Native ... altri verranno in futuro
- Benefici:
  - Tempo di boot dell'applicazione più veloce, meno RAM utilizzata (non c'è il JITter caricato)
  - Migliori performance
  - Minore utilizzo di batteria
- Svantaggi:
  - Tempi di compilazione
  - Le Expression Linq sono interpretate

# Scenario DEMO



# Containers: i mattoncini delle architetture moderne



# Perché .NET Core?

- Ricostruire .NET senza compromessi sulla retro-compatibilità
  - Risolvere i problemi di versioning e dipendenze
  - Rendere il deploy delle nuove versioni più semplice (e side-by-side)
  - Evitare spreco di risorse negli scenari che richiedono scalabilità
  - Nessun compromesso: alcune «breaking changes» sono inevitabili
- Vantaggi
  - Scalabilità
  - Load balancing
  - Sfruttare al massimo l'hardware moderno
  - **Sicurezza**: isolare il codice dalle risorse e dalle apps
    - On the desktop: sandbox (UWP)
    - On the server: Containers in place of classic WebApps / Services

# Questions?

<https://twitter.com/ugidotnet>



```
C:\app1>dotnet --help
.NET Command Line Tools
Usage: dotnet [common-options] [command] [arguments]

Common Options (passed before the command):
-v|--verbose      Enable verbose output
--version         Display .NET CLI Version Number
--info           Display .NET CLI Info

Common Commands:
new
restore Restore
build
publish Publishes a .NET project for deployment (including the runtime)
run Compiles and immediately
test Runs unit
pack Creates a NUGET package
```

@raffaeler