



INTRODUZIONE A DotNetCore

Raffaele Rialdi



Twitter: @raffaeler

Email: raffaeler@vevy.com

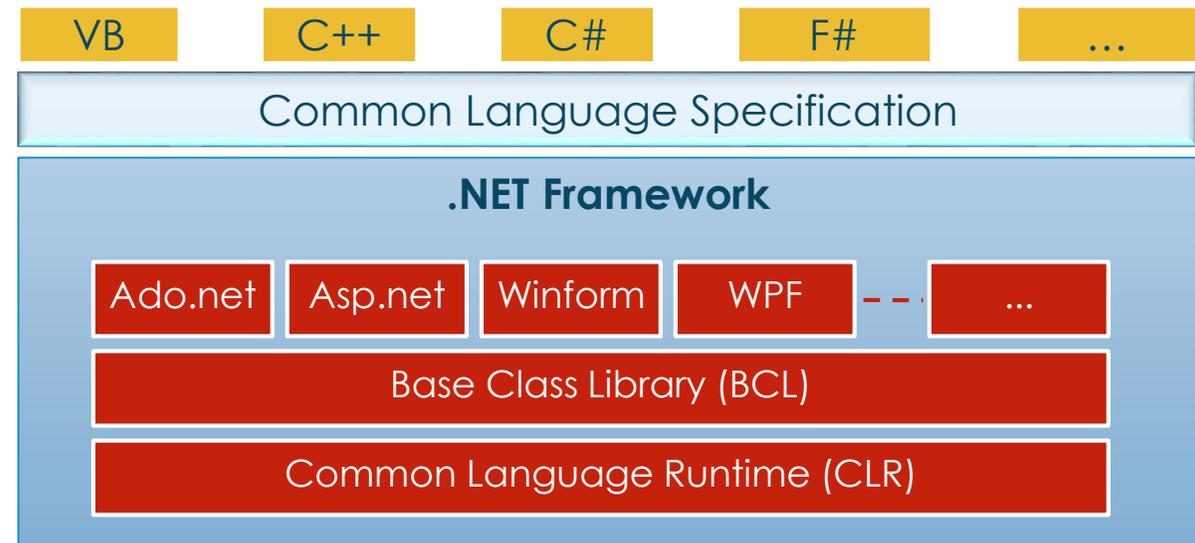
Website: <http://iamraf.net>

RATIONALE

- Ristrutturare la piattaforma .NET senza compromettere la compatibilità
 - Indirizzare i problemi di versionamento e dipendenze
 - Risolvere il problema del deploy → quali sorprese durante l'installazione del prodotto?
 - Come rendere più semplice l'installazione di una nuova versione?
 - Come gestire al meglio le installazioni side-by-side?
 - Come evitare di sprecare risorse negli scenari di scalabilità
 - È necessario comunque eseguire delle «breaking changes»
- Oggi si tende a ridurre il grafo delle dipendenze (librerie e servizi del sistema operativo)
 - Sul desktop: UWP al posto delle classiche applicazioni
 - Sul server: Containers al posto delle webapp classiche
- Vantaggi?
 - Scalabilità
 - Load balancing
 - Migliore sfruttamento dell'hardware moderno
 - Disaccoppiamento dalle versioni di host e librerie installate
 - Sicurezza!

.NET OGGI

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



COS'È DOTNETCORE

- È un « fork » del Framework.NET desktop
 - Nuovo runtime (Common Language Runtime – CLR)
 - Nuova codebase (Base Class Library – BCL)
 - Nuovi ma compatibili perché è un « fork », cioè nasce dagli stessi sorgenti
- Il **goal** di DotNetCore è di essere **Cross-Platform** e **Cross-Device**
 - Supporto per Windows, Linux e MacOS (e i Docker container)
 - *È un prodotto ufficiale e supportato da Microsoft sulle piattaforme testate*
- Le applicazioni che beneficiano di DotNetCore (al momento) sono:
 - **ASP.NET Core**: la nuova architettura di ASP.NET, riscritta da zero
 - **Universal Windows Platform**: le applicazioni Windows cross-device
 - **Cloud Applications**: applicazioni o micro-servizi per la cloud Azure
 - **Console Application**: il modo migliore per provarlo

CROSS-PLATFORM

- Attualmente DotNetCore viene compilato su diversi sistemi operativi
 - 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
- Supporto cross-platform genuino
 - Nessun sistema operativo « first-class »

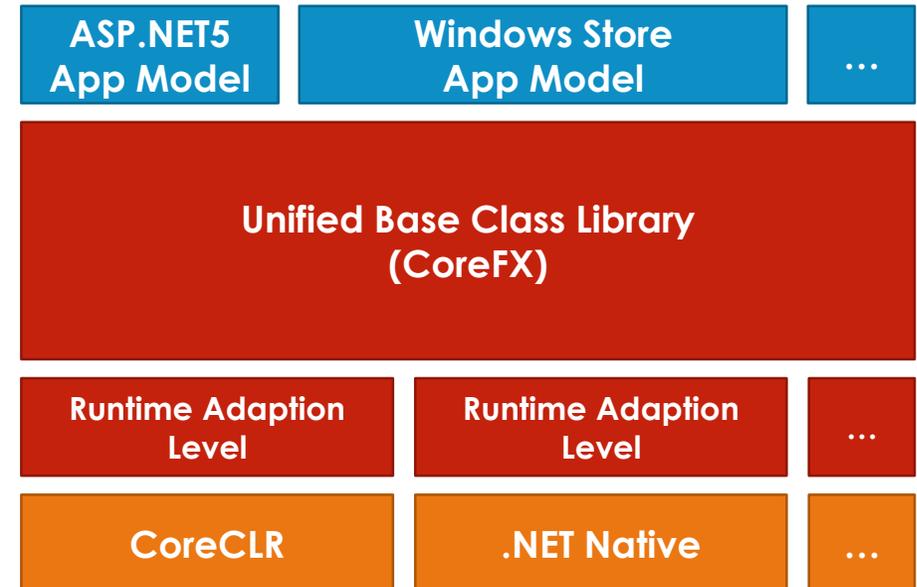
Ubuntu 14.04	Windows	Mac OS X	CentOS 7.1
build succeeded	build failed	build succeeded	build succeeded

L' ECOSISTEMA DI DOTNETCORE

- CoreCLR è il nuovo CLR usato ad esempio da ASP.NET
 - Si basa sul nuovo jitter « **RyuJIT** » e contiene GC, Interop e servizi di base
 - <https://github.com/dotnet/coreclr>
- CoreRT è un altro CLR che usa ".NET Native" usato dalle applicazioni UWP
 - Si basa sulla toolchain .NET Native che genera **solo e tutto codice nativo**
 - Le UWP usano CoreCLR in debug e .NET Native Runtime in deploy (release)
 - <https://github.com/dotnet/corert>
- CoreFX è la nuova Base Class Library o «.NET Core Foundational Libraries»
 - Scritta in puro Intermediate Language (IL) e usabile da tutti i CLR
 - <https://github.com/dotnet/corefx>
- DotNet Command Line Interface (CLI)
 - <https://github.com/dotnet/cli>

BIG PICTURE

- Le librerie che usano solo CoreFX possono essere condivise in tutti gli Application Model (ASP.NET, UWP, Console)
- I « Runtime Adaptation Layer » sono specifici della piattaforma / CPU
 - x86, x64, ARM, altre CPU in arrivo...
- Modularità pensata per estensioni future
 - Nuovi Application Model
 - Nuovi OS / Platform
 - Nuove varianti di CLR



JSON PROJECT

- Nuova struttura di Solution e Progetto, basata sulla struttura del file system
 - I file hanno un nuova struttura, in formato JSON
- Global.JSON è il file che governa la solution
 - È in aggiunta al file .sln usato solo da Visual Studio
- Project.JSON è il file che governa il singolo progetto
 - È in aggiunta al file .xproj usato solo da Visual Studio
 - Contiene la lista dei **framework** su cui potrà girare il binario a runtime
 - Contiene la lista delle **dipendenze/reference** per ciascun framework
- La gerarchia della lista delle dipendenze viene "**esplosa**" e resa "**flat**" nel file project.lock.json per rendere più veloce il lavoro dei tool

IL RUOLO DI NUGET

- CoreFX viene distribuita solo via nuget
 - Ogni namespace ha la sua dll
 - Niente più GAC
 - Il deploy è privato
 - Ogni applicazione ha il suo set di dll
- Le dll private
 - O sono parte della solution
 - O viene creato un pacchetto nuget
 - Il server nuget può essere privato
- Nuget diventa lo strumento di distribuzione via DotNet CLI

```
{
  "version": "1.0.0-*",
  "description": "XmlRpcCore Class Library",
  "authors": [ "RaffaeleR" ],
  "tags": [ "" ],
  "projectUrl": "",
  "licenseUrl": "",
  "frameworks": {
    "net451": {
      "frameworkAssemblies": {
        "System.Xml.Linq": "4.0.0.0"
      },
      "dependencies": {
        "System.Reflection": "4.1.0-beta-23516"
      }
    },
    "dotnet5.4": {
      "dependencies": {
        "Microsoft.CSharp": "4.0.1-beta-23516",
        "System.Collections": "4.0.11-beta-23516",
        "System.Linq": "4.0.1-beta-23516",
        "System.Runtime": "4.0.21-beta-23516",
        "System.Threading": "4.0.11-beta-23516"
      }
    }
  },
  "dependencies": {
    "System.Xml.XDocument": "4.0.11-beta-23516"
  }
}
```

LA COMMAND LINE

- La CLI (Command Line Interface) è tornata
 - È una garanzia di funzionamento per il mondo cross-platform
 - Specie IoT / Server dove non c'è un desktop grafico
 - È un modo per automatizzare i task di deploy
 - Ma questo non significa l'abbandono dei tool visuali
 - È facile installarla a macchina "vuota"
- Nella versione attuale abbiamo (e possiamo dimenticarci)
 - "dnvm" DotNet Version Manager
 - "dnx" DotNet eXecution environment
 - "dnu" DotNet development Utilities
- A breve debutterà una unica CLI: "dotnet"

DOTNET CLI IN PRATICA

1. Install the CLI
 1. MSI, apt-get, OSX pkg, docker
2. Create an App

```
mkdir myapp
cd myapp
dotnet new
```
3. Write some code
 - and the **project.json** file
4. Run

```
dotnet restore
dotnet run
```

```
.NET Command Line Tools (1.0.0-beta-001598)
Usage: dotnet [common-options] [command] [arguments]

Arguments:
  [command]      The command to execute
  [arguments]    Arguments to pass to the command

Common Options (passed before the command):
  -v|--verbose  Enable verbose output
  --version     Display .NET CLI Version 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
  repl         Launch an interactive session (read, eval, print, loop)
  pack         Creates a NuGet package
```

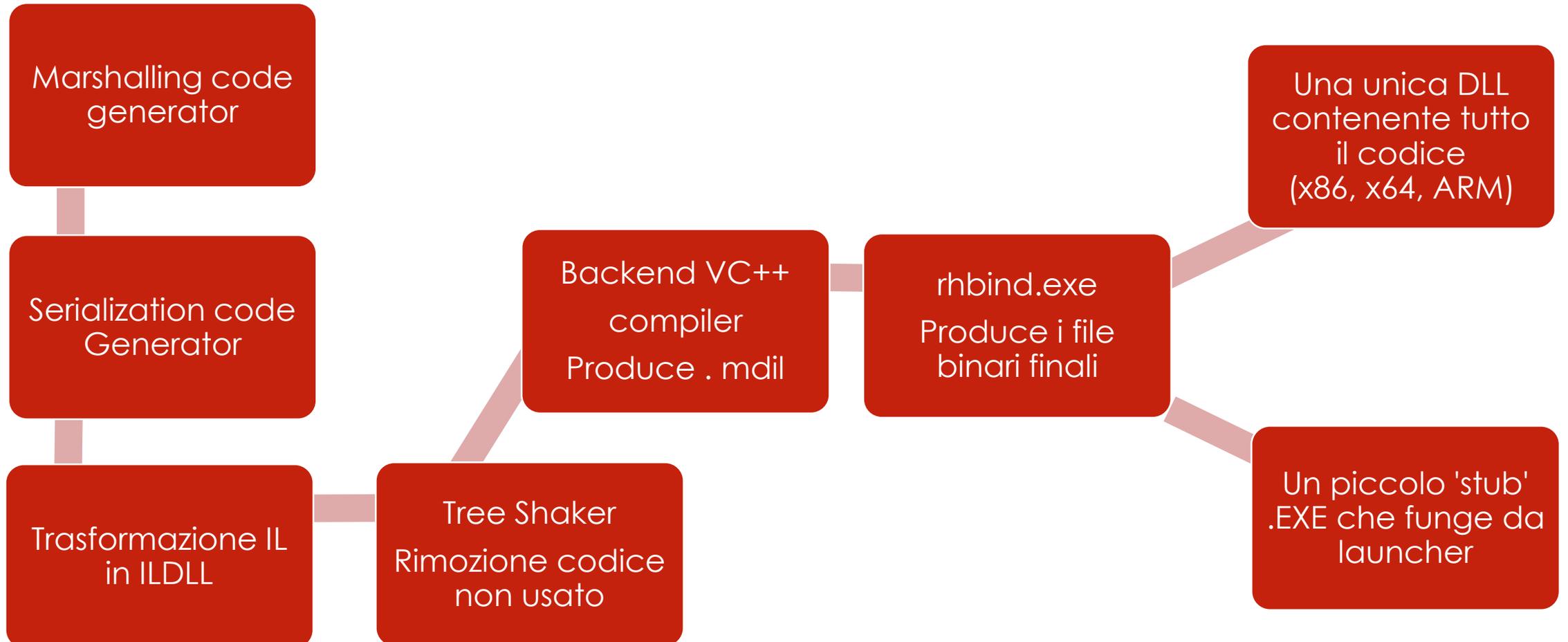
<http://dotnet.github.io/getting-started/>

.NET NATIVE

- A partire dal Framework.NET 4.6, viene usato il nuovo jitter «RyuJIT»
- .NET Native è una toolchain o «AOT» Ahead Of Time IL Compiler
 - Compilazione nativa usando il backend compiler di VC++
 - Una versione molto avanzata del vecchio **ngen**
 - Il jitter e i servizi di "virtual machine" del CLR non sono più necessari
 - Tecnicamente il CLR che usa .NET Native si chiama **CoreRT**
 - L'application model fruitore (non unico) di .NET Native sono le App UWP
- Benefit:
 - Minore tempo di bootstrap
 - Maggiori performance
 - Minore consumo di batteria
- Svantaggi:
 - Tempi di compilazione
 - Interpretazione delle Linq Expressions

TOOLCHAIN .NET NATIVE

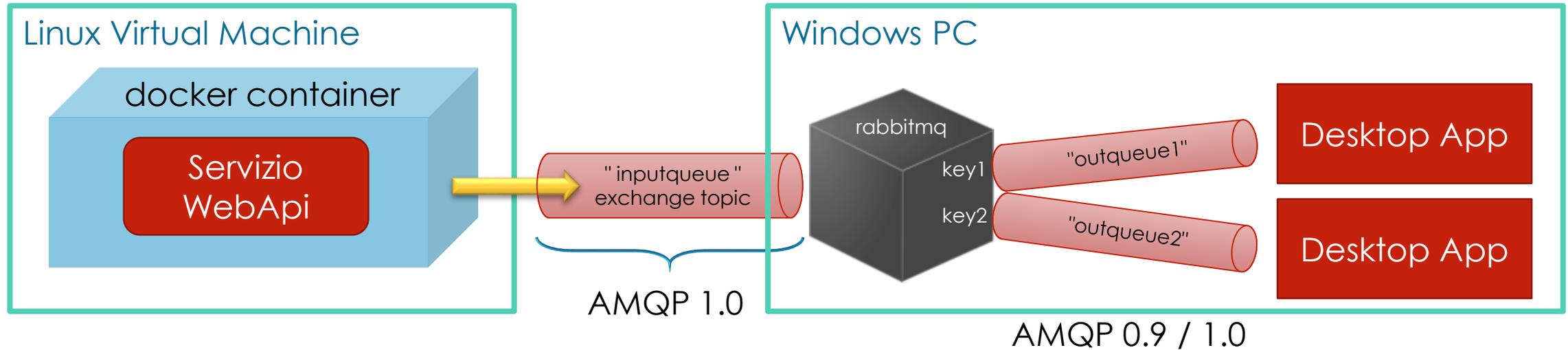
I PASSI DELLA COMPILAZIONE



UN TOOL PER MIGRARE LE APPLICAZIONI ATTUALI

- Il tool esiste in due salse
 - Un tool che può essere lanciato via Command Line
 - Una estensione di Visual Studio (più ricca di informazioni)
- I passi dell'analisi sono i seguenti:
 - Lettura dell'assembly generato alla compilazione
 - Analisi delle dipendenze
 - Chiamata ad un web service Microsoft (può anche funzionare offline)
 - Output di una tabella che indica quali chiamate non sono supportate sulle altre versioni di Framework
- Le analisi riguardano .NET standard, .NET Core, Xamarin, Mono, Silverlight, etc.

SCENARIO DEMO

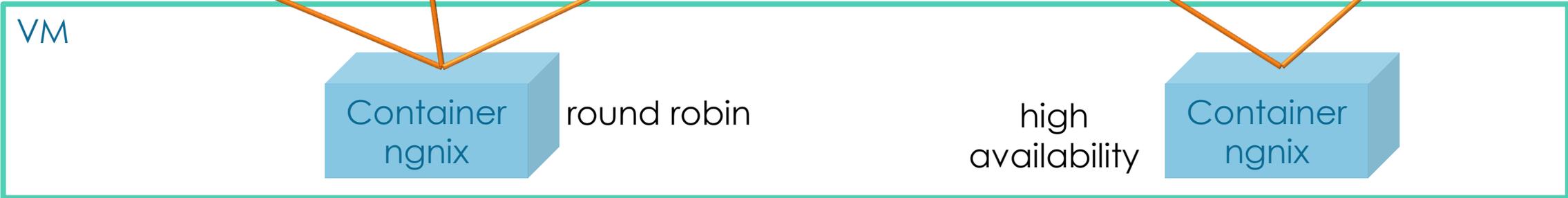


ESEMPI DI ARCHITETTURE ...

servizi replicati per fornire scalabilità



servizi replicati per fornire Alta Disponibilità



DOMANDE?

