

# La rivoluzione di ASP.NET Core nel mondo delle Web API

Raffaele Rialdi  
Senior Software Architect  
Microsoft MVP  
Consultant - Speaker - Teacher

 @raffaeler

 <https://github.com/raffaeler>

 <http://iamraf.net>

 [raffaeler@vevy.com](mailto:raffaeler@vevy.com)

# Perché ASP.NET Core?

1. Host generico per tutte le applicazioni non visuali
2. Performance eccezionali sul ciclo request/response
3. Interamente riscritto da zero per evitare i problemi del passato
4. Cross-platform, cross-architecture e container-ready
5. Nuovo framework di autorizzazione basato su policy
6. Estremamente semplice da estendere (dependency injection)
7. Supporto nativo per servizi in background
8. Un progetto per scenari di self-hosting, service o reverse-proxy
9. Configurazione semplice ed estendibile
10. Ricco ecosistema di middleware (Websockets, SignalR, Swagger, ...)

# Performance!

## Saturating 10GbE with 7+ million requests

<https://www.techempower.com/benchmarks/#section=test&runid=8ca46892-e46c-4088-9443-05722ad6f7fb&hw=ph&test=plaintext>

Best plaintext responses per second, Test environment (331 tests)						Classification	Language	Web Server
Rnk	Framework	Best performance (higher is better)		Errors	Cls	Lng	Plt	FE
1	<a href="#">actix-raw</a>	7,003,320	<div></div> 100.0%	0	Plt	Rus	Non	act
2	<a href="#">wizzardo-http</a>	7,002,116	<div></div> 100.0%	0	Mcr	Jav	Non	Non
3	<a href="#">aspcore</a>	7,000,118	<div></div> 100.0%	0	Plt	C#	.NE	kes
4	<a href="#">ulib</a>	6,998,365	<div></div> 99.9%	1	Plt	C++	Non	ULi
5	<a href="#">ulib-plaintext_fit</a>	6,998,068	<div></div> 99.9%	0	Plt	C++	Non	ULi
6	<a href="#">hyper</a>	6,996,874	<div></div> 99.9%	0	Mcr	Rus	Rus	Hyp
7	<a href="#">libreactor</a>	6,996,726	<div></div> 99.9%	0	Mcr	C	Non	Non
8	<a href="#">tokio-minihttp</a>	6,995,981	<div></div> 99.9%	0	Mcr	Rus	Rus	tok
9	<a href="#">baseio-http-lite</a>	6,983,540	<div></div> 99.7%	0	Plt	Jav	bas	Non
10	<a href="#">aspcore-rhtx</a>	6,975,733	<div></div> 99.6%	0	Plt	C#	.NE	kes
11	<a href="#">rapidoid-http-fast</a>	6,951,751	<div></div> 99.3%	0	Plt	Jav	Rap	Non
12	<a href="#">rapidoid</a>	6,926,431	<div></div> 98.9%	0	Plt	Jav	Rap	Non
13	<a href="#">actix</a>	6,840,894	<div></div> 97.7%	0	Mcr	Rus	Non	act
14	<a href="#">mofuw</a>	6,762,181	<div></div> 96.6%	0	Plt	Nim	Non	Non
15	<a href="#">thruster</a>	6,498,029	<div></div> 92.8%	0	Mcr	Rus	Rus	Non
16	<a href="#">httpbeast</a>	6,464,835	<div></div> 92.3%	0	Plt	Nim	Non	Non

# .NET Framework o .NET Core?

- .NET Framework non verrà aggiornato alle versioni più recenti di C#
  - Le versioni 7.x e 8.0 contengono migliorie importanti
  - La retro-compatibilità guida queste scelte (a scapito della performance)
- .NET Core porta novità importanti per le performance
  - Memory<T> e Span<T> sono esempi clamorosi
- ASP.NET Core e .NET Core
  - ASP.NET Core 2.x può funzionare su .NET Framework e .NET Core
  - ASP.NET Core 3.x può funzionare solo su .NET Core

# Il formato binario 'netstandard'

- Le librerie netstandard sono il territorio neutrale tra i due framework
  - La migrazione è pressoché indolore a partire dal .NET Framework 4.72
  - Trasformare in netstandard 2.0 le librerie è la via migliore per migrare
- Possiamo guardare a netstandard come fosse un'**interfaccia**
  - È un contratto che garantisce di trovare le sue API nei vari framework
- La versione attuale di netstandard è la 2.0
  - Arriverà presto la 2.1 che NON sarà implementata da .NET Framework
  - Mono, Unity, Xamarin e UWP sono altre implementazioni che si adegueranno

# Generic Host (HostBuilder)

- HostBuilder può essere usato da qualsiasi altra app
  - Microsoft.Extensions.Hosting namespace e Nuget package

Templates	Short Name	Language	Tags
Console Application	console	[C#], F#, VB	Common/Console
Class library	classlib	[C#], F#, VB	Common/Library
WPF Application	wpf	[C#]	Common/WPF
Windows Forms Application	winforms	[C#]	Common/WinForms
Worker Service	worker	[C#]	Common/Worker/Web
Unit Test Project	mstest	[C#], F#, VB	Test/MSTest

- Template pre-configurato con:
  - Dependency injection, variabili di ambiente, directory di default, configurazione, logging, supporto a self-hosting e servizio Windows/Linux
- L'utente può aggiungere di Lifetime o Timed services e supporto al Ctrl-C

# WebHost (WebHostBuilder)

- Questo host viene usato per servire richieste HTTP
- La configurazione viene demandata alla classe "Startup"
- Startup.ConfigureServices
  - Aggiunge i servizi al Dependency Injector (MVC, Auth, Authz, Cookie, ...)
  - Configura i servizi (settings application-wide)
- Startup.Configure
  - Inizializza la Pipeline responsabile della gestione Request / Response
  - File statici, Https, Regole di routing verso i controller, etc.

# ASP.NET Core: Quale HTTP server

- ASP.NET Core usa il web server "Kestrel" o IIS ( $\geq$  ASP.NET Core 2.2)
- È molto performante, può essere esposto direttamente o tramite un Reverse Proxy
- Linux:
  - Nginx è il reverse proxy di riferimento (ma non è certamente l'unico)
- Windows e IIS in modalità reverse proxy
  - Si configura il modulo ASP.NET Core di IIS in modalità **out of process**
- Windows e IIS in modalità HTTP server (a partire da ASPNET Core 2.2)
  - Si configura il modulo ASP.NET Core di IIS in modalità **in-process**



ASP.NET Core e security

# ASPNET Core: Security e GDPR

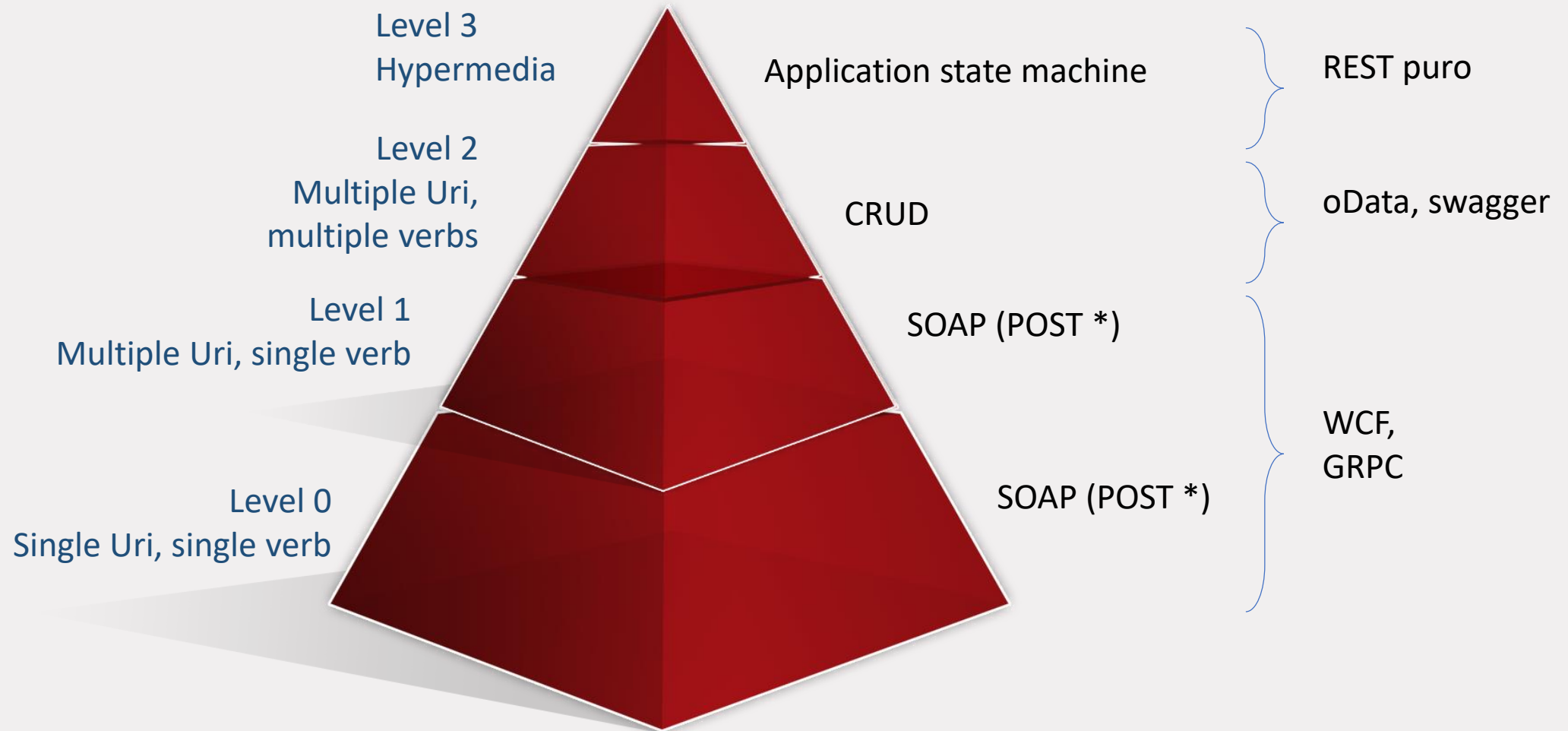
- HTTPS ovunque
  - Tool per generare i certificate self-signed
  - Tip: usare <https://letsencrypt.org/> provider gratuito per produzione  
`app.UseHttpsRedirection()` → server-side redirection verso HTTPS
  - `app.UseHsts()` → client-side redirection verso HTTPS
- I nuovi template forniscono i requisiti base della GDPR
  - "cookie banner"
  - Download dei dati di privacy
  - Cancellazione profilo

# L'autenticazione è più semplice

- L'autenticazione viene aggiunta tramite Dependency Injection
  - Disponibili molti provider su Nuget
  - La creazione di nuovi provider è molto semplice
- Da ASP.NET Core 3.0 SDK Preview 3 integrazione di Identity Server
  - Fornisce l'autenticazione per le SPA che usano WebAPI
  - Si possono gestire scenari di autenticazione più sofisticati
  - Nuovi template (da command line)

Progettare APIs con WebAPI

# Leonard Richardson «maturity level»



*Il verb POST viene usato in quanto idempotente*

# GRPC

<https://grpc.io/>

- È una specifica che definisce il modo in cui dei processi possono interoperare in logica Remote Procedure Call
  - Cross language e cross platform
  - Analogo a SOAP
- Molto utile per comunicare tra processi
  - HTTP è molto performante, per mia esperienza analogo a Named Pipes
- Client e server sono fortemente accoppiati
  - Sconsigliabile quando i client sono fuori controllo (i.e. Fattura elettronica)

# oData (versione $\geq 4.0$ )

- OData è da tempo una specifica OASIS (ceduta da Microsoft)
- Definisce come esporre un servizio che consuma dati
  - Espone il modello di metadati del modello esposto dal servizio
  - Fornisce le modalità di navigazione del modello
  - Fornisce le modalità di CRUD su tutto il modello (limitabile a piacere)
  - Permette di eseguire query sui dati con una sintassi testuale (\$filter)
- Il lato server ritorna IQueryable<T> offrendo query performanti

# Il modello REST puro

<http://www.infoq.com/articles/roy-fielding-on-versioning>

- È basato sulla dissertazione di Roy Fielding
  - «Hypermedia as the engine of application state (HATEOAS)»
  - «Nothing but the entry-point is needed from the client»
- Per scrivere servizi REST puri abbiamo questo magnifico strumento 😊





# Swagger: REST con i piedi per terra

- Conosciuto anche come OpenAPI <https://www.openapis.org/>
- Serve a definire le specifiche di un servizio REST in modalità neutrale rispetto ai linguaggi e alle tecnologie.
- In altre parole vengono prodotti i metadati del servizio esposto
  - Permette di generare il codice client
  - Permette di conoscere tutte le API senza doverle "scoprire dinamicamente"
  - Determina la necessità di versionare le API
- In ASP.NET Core è implementato in due pacchetti Nuget:
  - Swashbuckle.AspNetCore
  - NSwagger
  - E un Analyzer!



**Microsoft.AspNetCore.Mvc.Api.Analyzers** ✓ by Microsoft, 183K d 📦 v3.0.0-preview3-19153-02  
CSharp Analyzers for ASP.NET Core MVC.

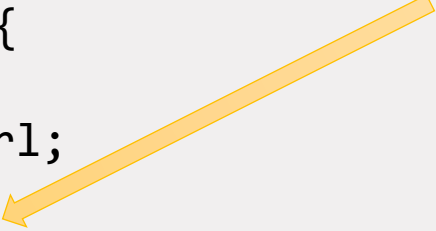
# Eseguire chiamate HTTP da ASP.NET Core

- Step 1: Creare un client tipizzato

Dependency Injection

```
public class CatalogService : ICatalogService {
    private readonly HttpClient _httpClient;
    private readonly string _remoteServiceBaseUrl;

    public CatalogService(HttpClient httpClient) => _httpClient = httpClient;
    public async Task<Catalog> GetCatalogItems(int page, int take,
                                                int? brand, int? type)
    {
        var uri = API.Catalog.GetAllCatalogItems(
            _remoteServiceBaseUrl, page, take, brand, type);
        var responseString = await _httpClient.GetStringAsync(uri);
        var catalog = JsonConvert.DeserializeObject<Catalog>(responseString);
        return catalog;
    }
}
```



# Eseguire chiamate HTTP da ASP.NET Core

- HttpClientFactory si trova in Nuget: **Microsoft.Extensions.Http**
- Aggiungere il client tipizzato alla Dependency Injection

```
services.AddHttpClient<ICatalogService, CatalogService>();
```
- A cosa serve HttpClientFactory
  - Gestisce autonomamente un pool di client (la durata dipende da SetHandlerLifetime appesa ad AddHttpClient)
  - Supporta la Retry Policy e "Circuit Braker" tramite AddPolicyHandler
    - Circuit breaker serve ad isolare le operazioni che **probabilmente** falliranno
    - <https://docs.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker>

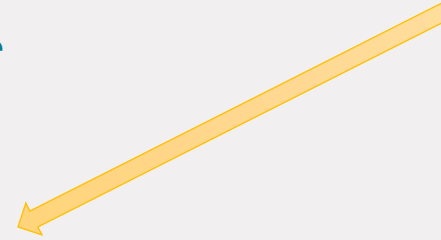
# Come si usa il client HTTP

```
public class CatalogController : Controller
{
    private ICatalogService _catalogSvc;

    public CatalogController(ICatalogService catalogSvc)
    {
        _catalogSvc = catalogSvc;
    }

    // ...
}
```

Dependency Injection

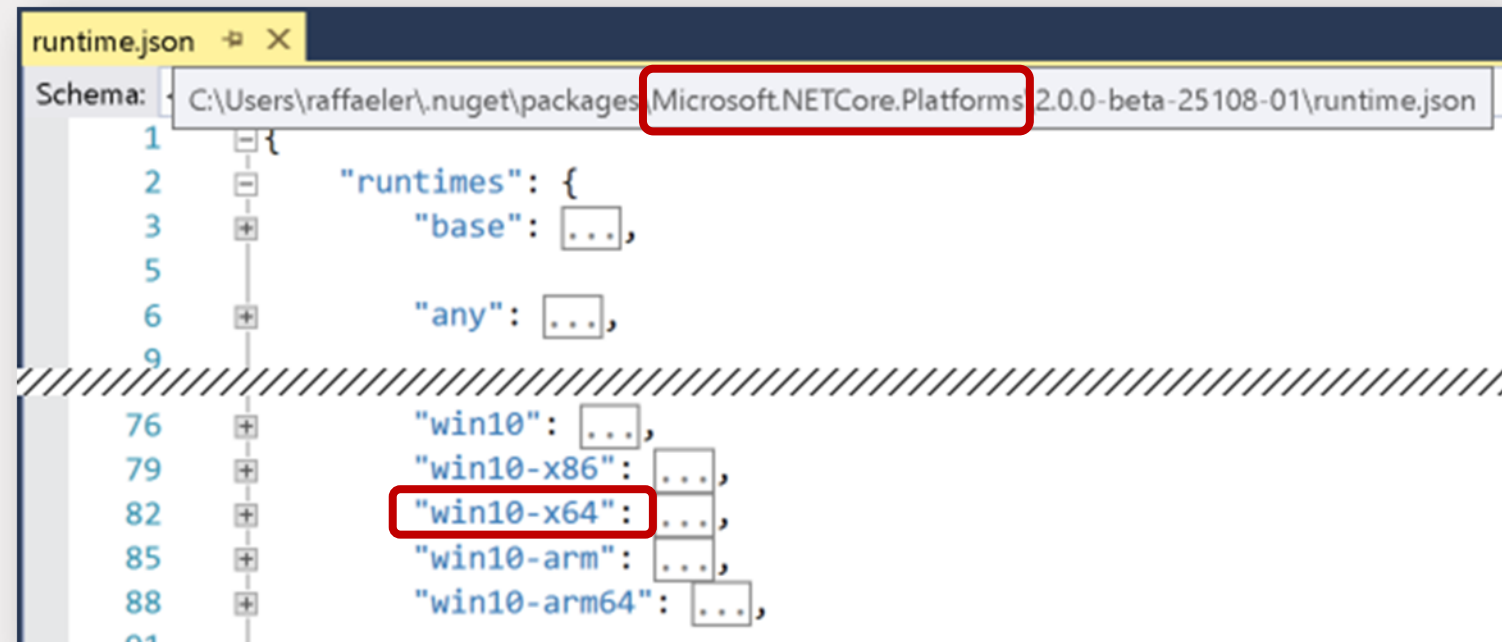


# ASP.NET Core: Creare un Servizio Windows

- È necessario specificare un RID
- Sostituire `host.Run` con `host.RunAsService`
- Specificare la folder base usando `.UseContentRoot(...)`
- Pubblicare l'app (dotnet publish)
- Usare il comando "sc create" per creare il servizio Windows



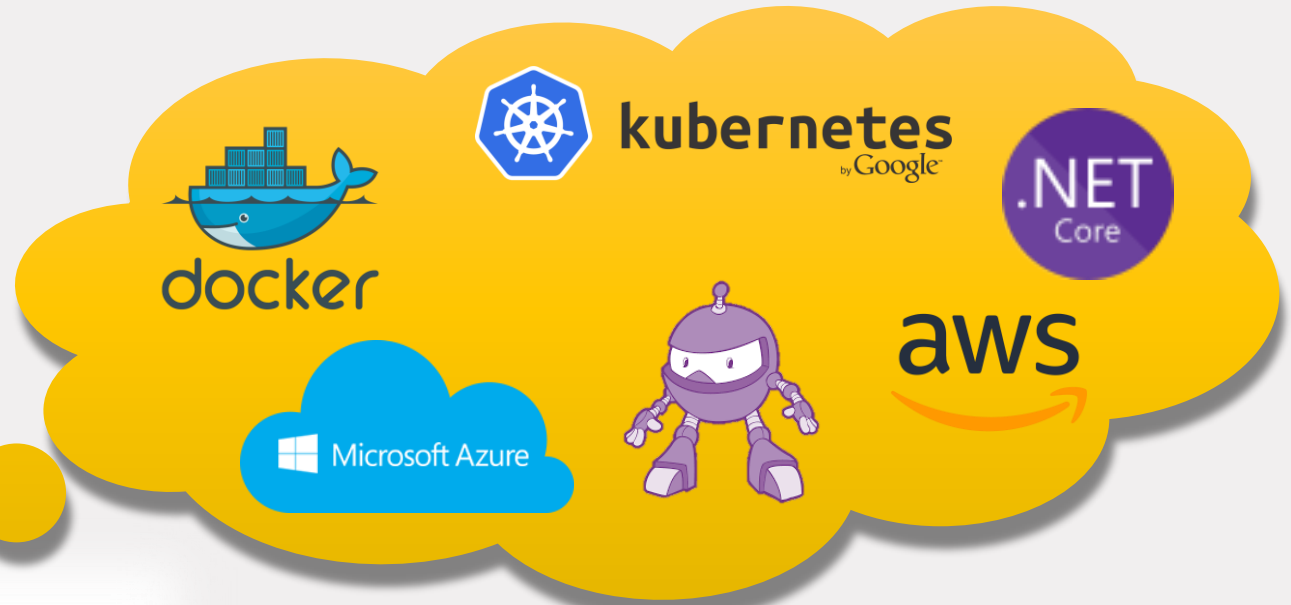
```
<PropertyGroup>  
  <TargetFramework>netcoreapp2.1</TargetFramework>  
  <RuntimeIdentifier>win10-x64</RuntimeIdentifier>  
</PropertyGroup>
```



# Links

- dotnet sdk list (global tool):
  - <https://github.com/jonstodle/DotNetSdkHelpers>
- OData
  - \$orderby, \$top, \$count:
    - <http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.html>
  - \$filter <http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html>
  - Main Repo: <https://github.com/OData/WebApi>
  - Additional Samples <https://github.com/OData/ODataSamples>

# Questions?



## Thank you!

@raffaeler

[raffaeler@vevy.com](mailto:raffaeler@vevy.com)