Online on-demand Project Support

.NET Core 3.0

In a Nutshell

Thorsten Kansy (tkansy@dotnetconsulting.eu)

Thorsten Kansy

Freier Consultant, Softwarearchitekt,
Entwickler, Trainer und Fachautor

DOTNETCONSULTING
by Thorsten Kansy

Agenda

- Verwendete Software
- Einführung
- Konfiguration
- Inversion of Control & Dependency Injection
- Logging
- Deployment
- Migration


pienene
the latvian green studio
pienene
latvian green studio
P
Fabisiak
44
Valvoline

Roadmap

Roadmap

Build 06-08.05.2019

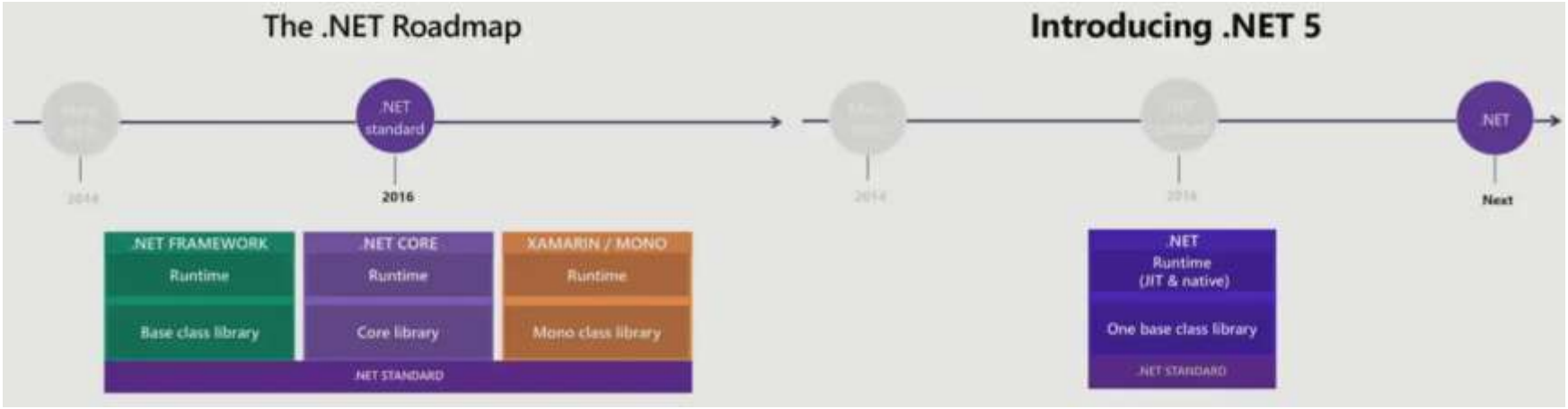
- .Net Core 3.0 (@.NET Conf 23.09.2019)
- .NET 5.0 Nov 2020
- Major releases every year,
- LTS (Long Term Support, 3 Jahre) grade Versionen

```
https://www.microsoft.com/en-us/build
```

DOTNETCONSULTING by Thorsten Kansy

Quelle: MicrosoftBuild

Roadmap



.NET Framework + .NET Core + Mono/ Xamerin

=

.NET 5.0

DOTNETCONSULTING
by Thorsten Kansy

Quelle: MicrosoftBuild

Roadmap

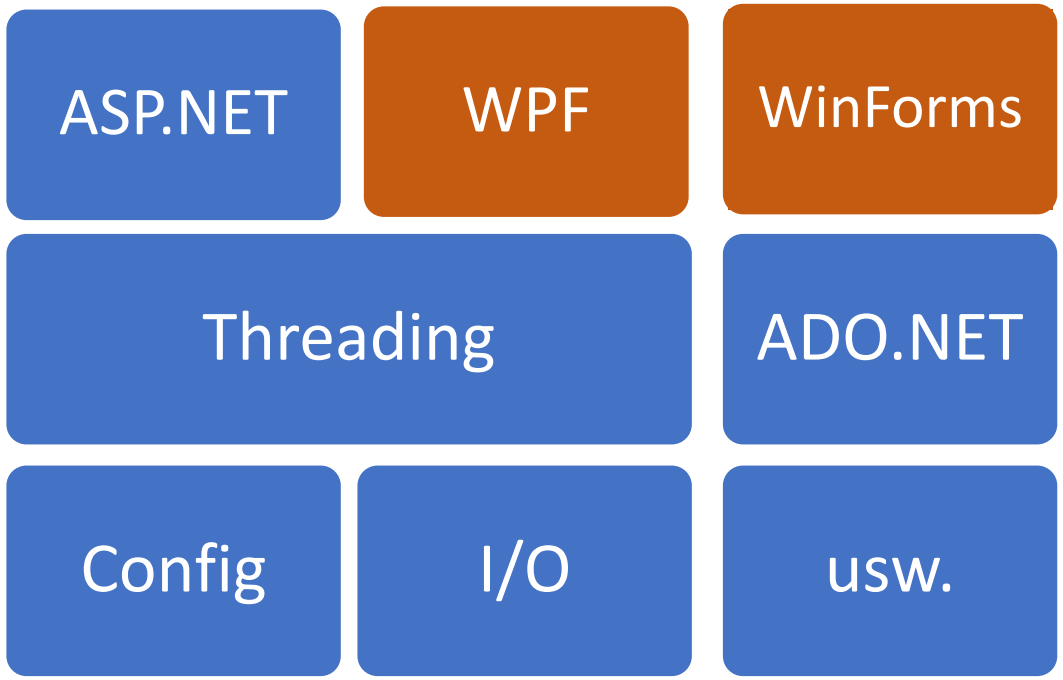
- .NET Framework 4.8 ist die letzte .NET Framework Version
- Migration von .NET Framework App nur bei neuen Features notwendig
- CoreRT (Optimized single File .NET with AOT)

- Sterben wird:
  - ASP.NET Webforms
  - ASP.NET Webservices
  - .NET Remoting
  - WCF (REST & SOAP)
  - Windows Workflow Foundation (WF)

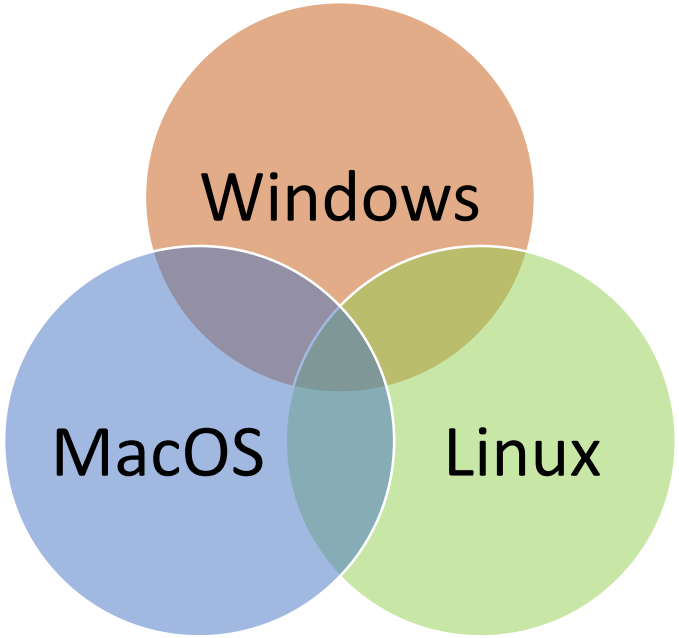
Einführung

.NET-Core-Aufbau

Modular



Multi Plattform



DOTNETCONSULTING
by Thorsten Kansy

.NET Core vs. .NET Framework

.NET Core	.NET Framework
Modularer Aufbau (NuGet)	Systemweites, (fast) monolithisches Framework
WPF, WinForms, ASP.NET, etc.	WPF, WinForms, ASP.NET, etc.
Fast vollständig, nicht überfrachtet	Vollständig, aber überfrachtet
Native Multiplattform (viele Funktionen)	Windows gebundene Plattform
Nur teilweise an Windows gebunden	An Windows gebunden

Was spricht für .NET Core?

- Multiplattform
  - Windows
  - Linux
  - MacOS
- Höhere Performance
  - Modularität
  - ASP.NET, z. B. Entkopplung von `System.Web.dll`
- Geringerer Ressourcenverbrauch
  - Modularität

Konfiguration

Konfigurationsprovider

Provider	NuGet-Paket
JSON-Datei	`Microsoft.Extensions.Configuration.Json`
XML-Datei	`Microsoft.Extensions.Configuration.Xml`
INI-Datei	`Microsoft.Extensions.Configuration.Ini`
Umgebungsvariablen	`Microsoft.Extensions.Configuration.EnvironmentVariables`
Programmargumente	`Microsoft.Extensions.Configuration.CommandLine`
In-Memory-Collection	–
Azure Key Vault	`Microsoft.Extensions.Configuration.AzureKeyVault`
User Secrets (nur ASP.NET Core)	`Microsoft.Extensions.Configuration.UserSecrets`
Custom	–

```
Install-Package Microsoft.Extensions.Configuration.Json (s.o.)
```

Konfiguration verwenden

```
// Konfiguration vorbereiten, Provider nach Bedarf anfügen
IConfigurationBuilder builder = new ConfigurationBuilder()
    // Umgebungsvariablen hinzufügen
    .AddEnvironmentVariables()
    // Json-Datei hinzufügen
    .AddJsonFile("Config.json");

// Konfiguration abschließen, Werte einlesen
IConfigurationRoot config = builder.Build();

// Zugriff
string windowHeight = config["App:Window:Height"];
```

Demo

Eigenen Custom Provider schreiben

Code-Klasse	Aufgabe
`SqlConfigurationExtensions`	Erweiterung der Fluent API
`SqlDatabaseConfigurationSource`	Implementiert `IConfigurationSource`, um eine Konfiguration mittels Fluent API zu ermöglichen.
`SqlDatabaseConfigurationProvider`	Implementiert `IConfigurationProvider` und stellt die eigentliche Programmlogik dar, die die Konfigurationswerte bereitstellt.
`SqlDatabaseChangeToken`	Implementiert `IChangeToken` und bietet die Möglichkeit, Veränderungen bei den Konfigurationswerten zu signalisieren. Hier nicht weiter vertieft, da die Änderungen zwischen dem Lesen zwei zusammenhängender Werte auftreten können.

Demo

Inversion of Control & Dependency Injection

Inversion of Control und Dependency Injection

- Überschaubarer Code (Single Responsibility Principle)
- Wiederverwendbarkeit
- Besser testbarer Code

`IServiceCollection` IoC-Container

`IServiceProvider` DI-Service

```
Install-Package Microsoft.Extensions.DependencyInjection
```

```
Install-Package Microsoft.Extensions.DependencyInjection.Abstractions
```

IoC-Konfiguration

```
static private void ConfigureServices(IServiceCollection serviceCollection)
{
    // Instanz pro GetService<>()-Aufruf/ GetRequiredService<>()-Aufruf
    serviceCollection.AddTransient<IOrderService, SnailMailOrderService>();

    // ODER Instanz als Singelton
    serviceCollection.AddSingleton<IOrderService, SnailMailOrderService>();

    // ODER Instanz per (ASP.NET)-Request (siehe CreateScope())
    serviceCollection.AddScoped<IOrderService, SnailMailOrderService>();
}
```

Lebenszeit von Instanzen

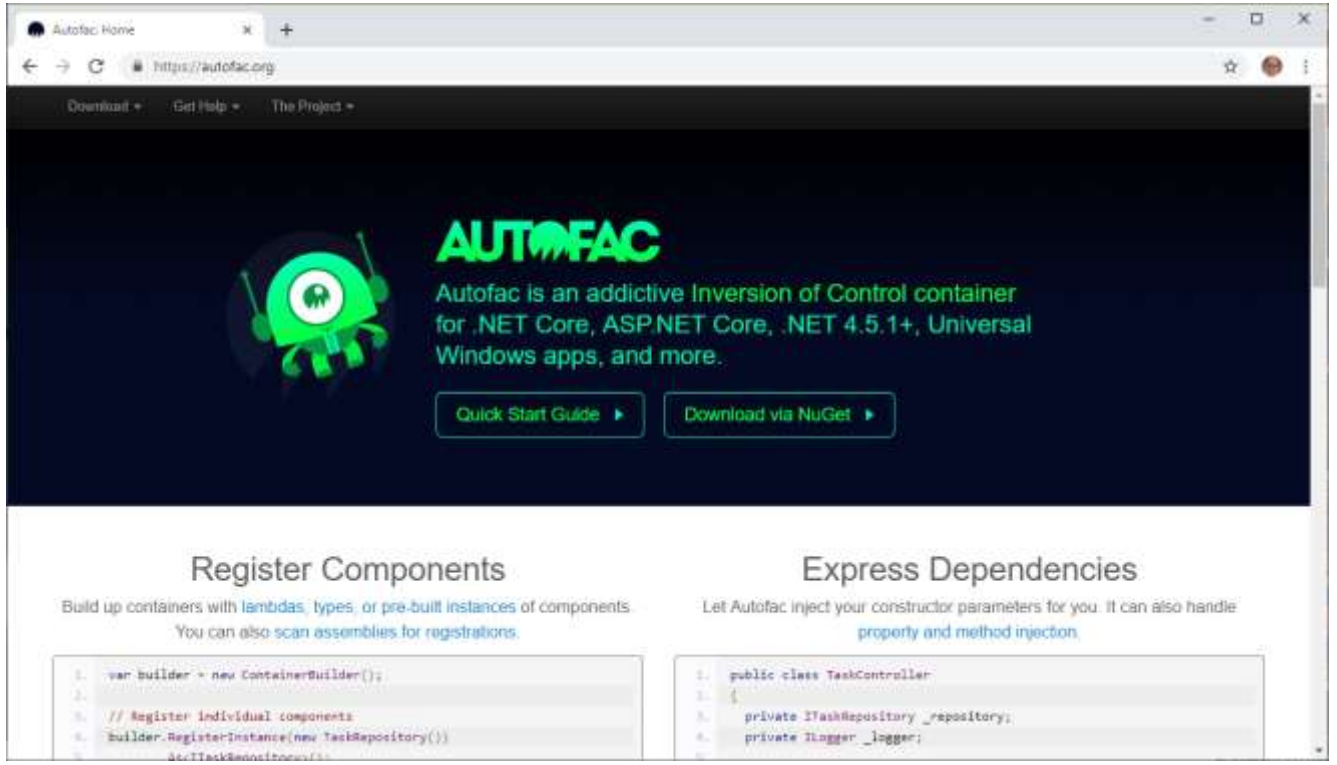
Variante	Lebenszeit
`AddTransient()`	Der IoC-Container liefert jedes Mal eine neu erzeugte Instanz
`AddScoped()`	Während einer Anfrage (z. B. an einen ASP.NET-Controller) wird die gleiche Instanz geliefert. Bei der nächsten Anfrage wird wiederum eine neue Instanz geliefert
`AddSingleton()`	Der IoC-Container liefert jedes Mal die gleiche Instanz zurück

DI-Service

```
IServiceProvider services = serviceCollection.BuildServiceProvider();

// Liefert NULL, wenn der Service nicht geliefert werden kann
services.GetService<IOrderService>()
    ?.PlaceOrder("Wattestäbchen", 10);

// Wirft eine Exception (System.InvalidOperationException)
// statt NULL zu liefern
services.GetRequiredService<IOrderService>()
    .PlaceOrder("Wattestäbchen", 10);
```

Demo

IoC- und DI-Alternative – Autofac



http://autofac.org

```
Install-Package Autofac
```

```
Install-Package Autofac.Extentions.DependencyInjection
```

Logging

Logging

- 3 gute Gründe:
- Fehlersuche, Fehlersuche & Fehlersuche!

`ILoggerFactory` Logger Factory

`ILogger` Logger Service

```
Install-Package Microsoft.Extensions.Logging
```

```
Install-Package Microsoft.Extensions.Logging.Abstractions
```

Logging-Provider

Level	NuGet-Paket
Konsole	`Microsoft.Extensions.Logging.Console`
Debugger-Monitor (Output window)	`Microsoft.Extensions.Logging.Debug`
Trace Listener (Output window)	`Microsoft.Extensions.Logging.TraceSource`
Windows-Ereignisprotokoll	`Microsoft.Extensions.Logging.EventLog`
EventSource/EventListener	`Microsoft.Extensions.Logging.EventSource`
Azure-App-Dienste „Diagnoseprotokolle“ und „Log Stream“	`Microsoft.Extensions.Logging.AzureAppServices`
Datei, Datenbank, SMTP etc.	`Serilog.* // NLog.* // usw.`
Custom	–

Logging konfigurieren

```
static void LoggingDemo1(string scopeName = null)
{
    // Logger Factory konfigurieren
    ILoggerFactory loggerFactory = new LoggerFactory()
        .AddConsole(LogLevel.Trace, true)
        .AddDebug();

    ILogger logger = loggerFactory.CreateLogger<Program>();

    demoLogging(logger, scopeName);
}
```

Logging-Level

Variante	Lebenszeit
`Trace (0)`	Ablaufverfolgung, die auch sensible Informationen enthalten kann
`Debug (1)`	Technische Debug-Informationen für die Fehlersuche
`Information (2)`	Nachverfolgung des allgemeinen Ablaufs der Anwendung
`Warning (3)`	Warnungen bei ungewöhnlichen oder unerwarteten Ereignissen
`Error (4)`	Fehler und Ausnahmen, die vom Code nicht behandelt werden können
`Critical (5)`	Für Fehler, die sofortige Aufmerksamkeit erfordern

```
public enum LogLevel
{
    Trace = 0,
    Debug = 1,
    Information = 2,
    Warning = 3,
    Error = 4,
    Critical = 5,
    None = 6
}
```

DOTNETCONSULTING
by Thorsten Kansy

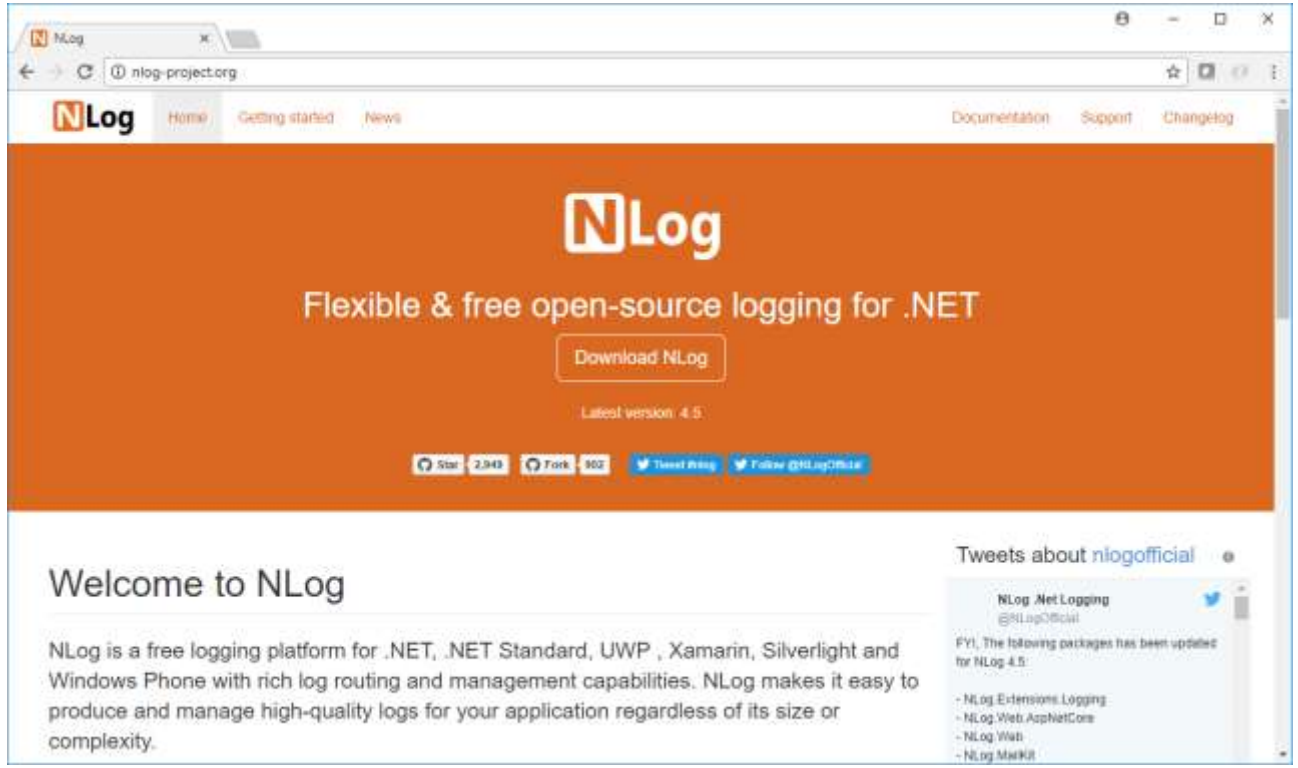
Demo

Logging & Dependency Injection

```
// Factory
serviceCollection.AddSingleton<ILoggerFactory, LoggerFactory>();

// Generischer Logger
serviceCollection.AddSingleton(typeof(ILogger<>), typeof(Logger<>));

public class AppController
{
    private readonly ILogger<AppController> _logger;

    public Application(ILogger<AppController> Logger)
    {
        _logger = Logger;
    }
}
```

Demo

Nlog – Fully-featured Logging Framework



```
http://nlog-project.org
```

```
Install-Package NLog
```

```
Install-Package NLog.Extensions.Logging
```

Demo

Fragen?

Links

```
https://www.visualstudio.com/de/downloads
```