

Online on-demand Project Support



# .NET Core 2.2

## Grundlagen



Thorsten Kansy ([tkansy@dotnetconsulting.eu](mailto:tkansy@dotnetconsulting.eu))

# Thorsten Kansy

THORSTEN KANSY

Freier Consultant, Softwarearchitekt,  
Entwickler, Trainer und Fachautor



# Agenda

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

A photograph showing two farmers from behind, wearing traditional conical hats, working in a vast rice paddy. The field is filled with young rice plants and water. The background shows more terraced fields stretching into the distance.

# Verwendete Software



Lokale Admin-Rechte

# Verwendete Software .NET Core 2.2

- Visual Studio 2017 15.9.3
- .NET Core 2.2



*Fokus trotzdem auf VS 17*

# Alternative Editoren

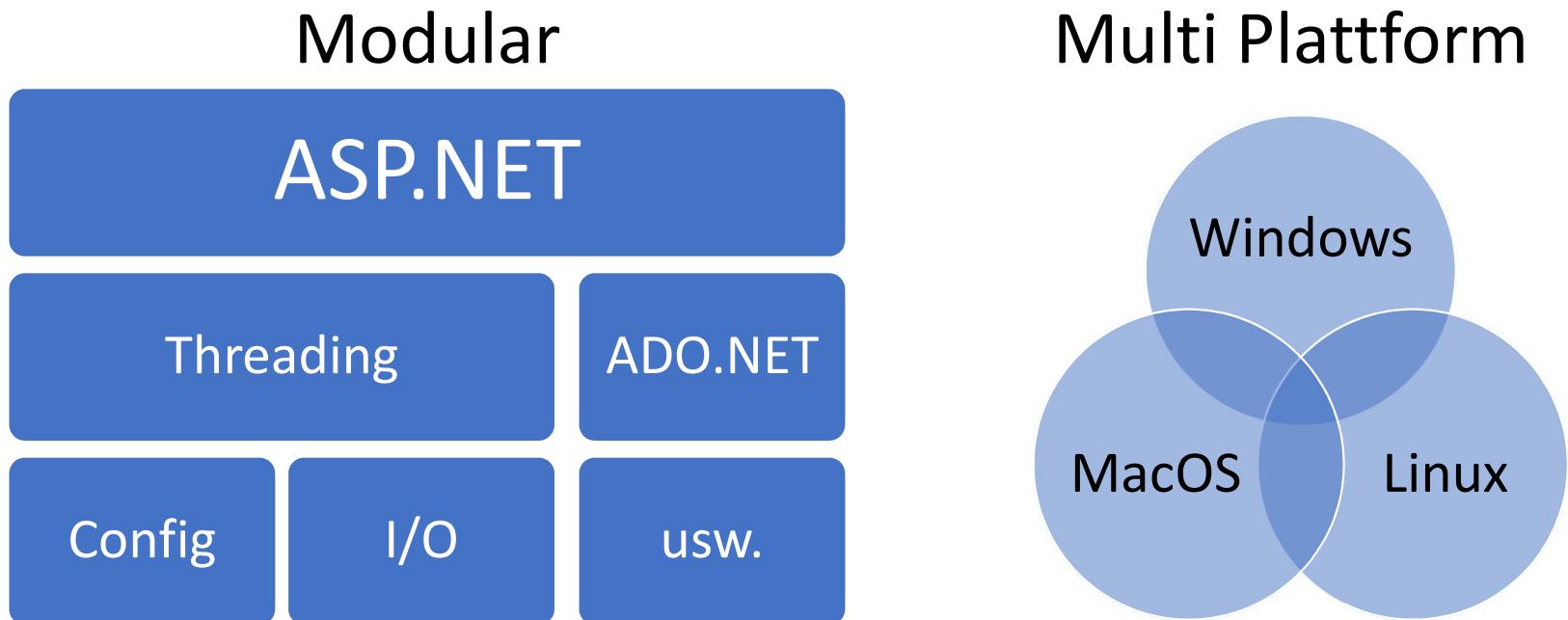
- Windows
  - Visual Studio Code
- MacOS
  - Visual Studio
  - Visual Studio Code
- Linux
  - Visual Studio Code

Und viele mehr, wie z. B. Sublime Text und Notepad

# Einführung



# .NET-Core-Aufbau



# .NET Core vs. .NET Framework

.NET Core	.NET Framework
Modularer Aufbau (NuGet)	Systemweites, (fast) monolithisches Framework
Keine UI (nur Web)	WPF, WinForms, ASP.NET, etc.
Unvollständig	Vollständig, aber überfrachtet
Native Multiplattform (viele Funktionen)	An Windows gebunden

# Was spricht für .NET Core?

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



# Was spricht gegen .NET Core?

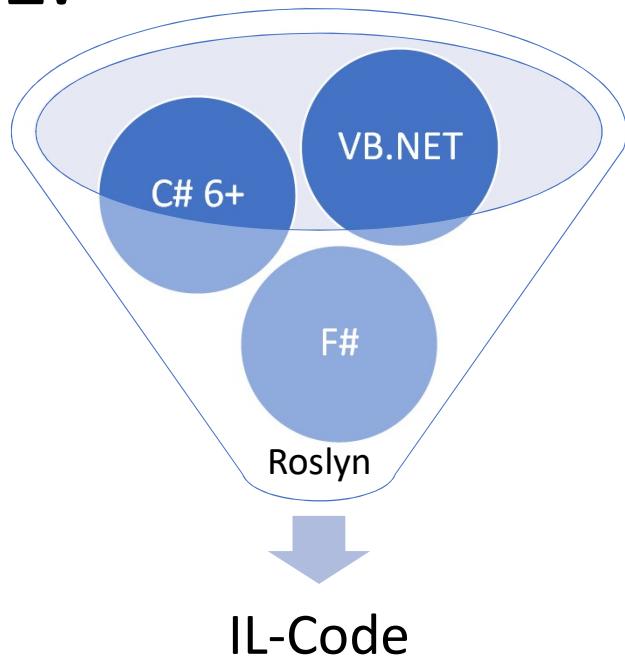
- (Noch) keine Desktop-UI bereitgestellt (zumindest von Microsoft)
  - Electron.NET
  - Avalonia
  - ImGui
  - WPF und Windows Form (.NET Core 3.0, Anfang 2019)
- .NET Core ist nicht vollständig -> wird erweitert entwickelt
  - API Breaks möglich

# Aber was ist mit .NET Standard?

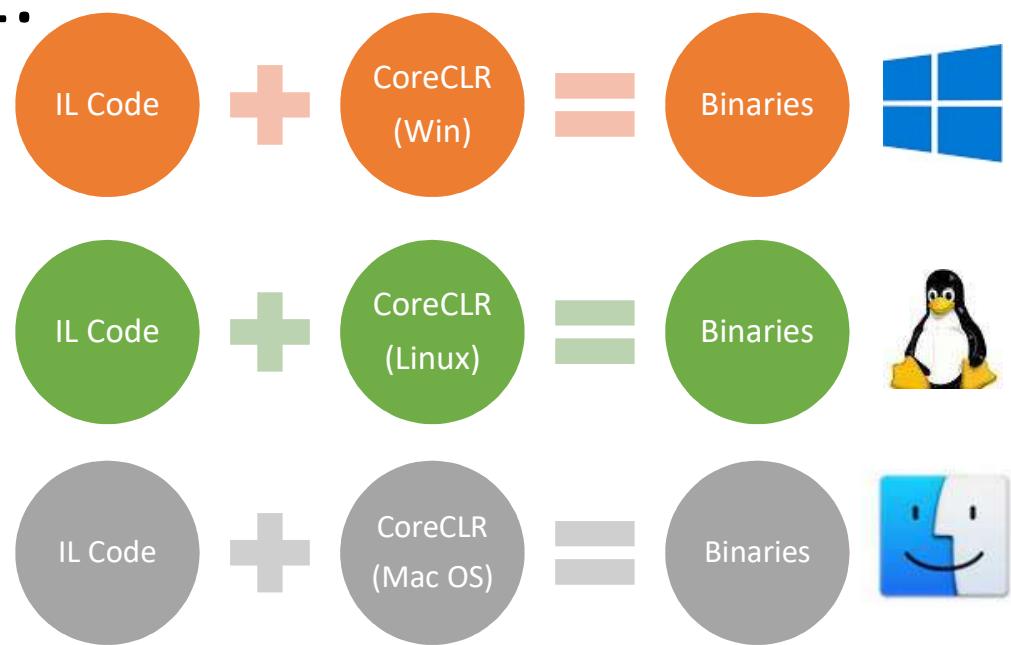
- Gemeinsame Funktionen für alle .NET Branches/Flavors
  - .NET Framework
  - .NET Core
  - Xamarin
- Jeweils in der aktuellen Version
- Erleichtert die Erstellung von Cross-Code

# Compile-Prozess(e)

1.



2.



# .NET Native

Nativer Code für Universal Window Platform (UWP)

- ngen.exe (Native Image Generator)

Vorteile für UWP

- Schnellere Ausführung
- Schnellerer Start
- Optimized app memory usage.



<https://docs.microsoft.com/en-us/dotnet/framework/net-native/>

# Visual Studio in Stellung bringen

## Visual Studio-Downloads

### Visual Studio Community 2017

Kostenlose, mit allen Funktionen ausgestattete IDE für Studenten, Open Source und einzelne Entwickler

### Visual Studio Professional 2017

Professionelle Entwicklertools, Dienste und Abonnementvorteile für kleine Teams

### Visual Studio Enterprise 2017

End-to-End-Lösung, die die anspruchsvollen Qualitäts- und Skalierungsanforderungen von Teams beliebiger Größe erfüllt

### Visual Studio Code

Codebearbeitung neu definiert.  
Kostenlos, Open Source und überall ausführbar.



<https://www.visualstudio.com/de/downloads>



Mit VS nicht notwendig

# .NET Core in Stellung bringen

- Windows
- Linux
- MacOS



<https://www.microsoft.com/net/download/core>

The screenshot shows the Microsoft .NET Core download page. At the top, there's a banner for ".NET Conf - Save the Date" (September 19 - 21, 2017). Below the banner, the .NET logo is on the left, and a large "Download .NET Core" button is on the right. A navigation bar includes links for HOME, DOWNLOADS, LEARN, ARCHITECTURE, CUSTOMERS, COMMUNITY, and SUPPORT. On the left, a sidebar has "Overview" at the top, followed by a blue "SDK" button and a grey "Runtime" button. Under "SDK", there's a section titled ".NET Core 2.0 SDK" with a table of download links:

Platform	File Type
Windows (x64) Installer	.exe download
Windows (x64) Binaries	.zip download
Windows (x86) Installer	.exe download
Windows (x86) Binaries	.zip download
macOS (x64) PKG	.pkg download
macOS Binaries	.tar.gz download
Linux	.tar.gz downloads
Visual Studio 2017 Tools	Installing .NET Core tools in Visual Studio 2017
.NET Standard Support for Visual Studio 2015	.msi download

# .NET Core CLI

- Command Line für .NET Core
  - Entity Framework
  - User Secrets (ASP.NET)
  - ...

```
dotnet --info
```

```
dotnet --version
```

```
dotnet exec <prog>.dll
```

```
dotnet ef
```

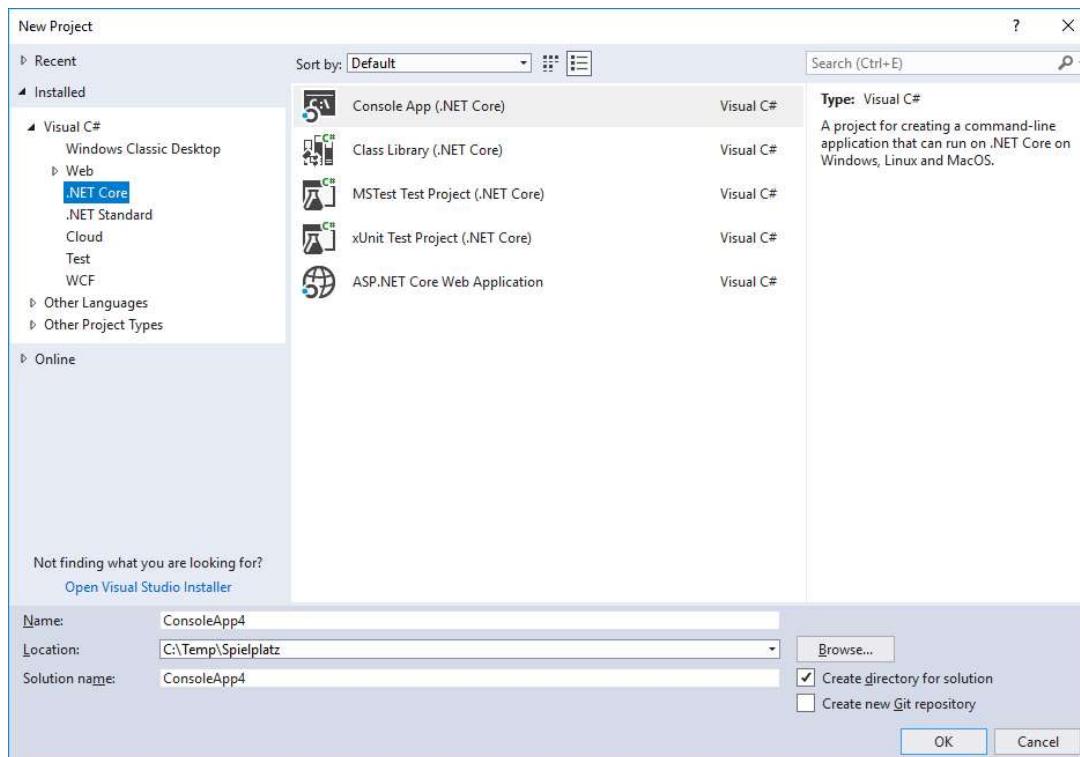
# „Hallo Welt“ mit CLI und Notepad

```
dotnet new console -n HelloWorld  
  
cd HelloWorld  
  
notepad program.cs  
  
dotnet run [<proj>.csproj]  
  
dotnet <prog>.dll
```



 **Demo** 

# „Hallo Welt“ mit Visual Studio 2017



 **Demo** 

# „Hallo Welt“ mit Visual Studio Code

```
md CoreHW
cd CoreHW
dotnet <prog>.dll
dotnet run <proj>.csproj

code .
```

# „Hallo Welt“ mit Visual Studio Code (Ubuntu)

The screenshot shows two instances of Visual Studio Code running on Ubuntu. The left instance is for the 'aspnet' project, which uses .NET Core. It displays the 'Program.cs' file with code for a web application. The right instance is for the 'CoreSample' project, which uses .NET Standard. It also displays the 'Program.cs' file with a simple 'Hello World!' console application. Both instances show the Explorer, Welcome, and Open Editors panes.

```
aspnet Project (Left):
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;

namespace aspnet
{
    public class Program
    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run();
        }

        public static IWebHost BuildWebHost(string[] args)
        {
            return WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .Build();
        }
    }
}
```

```
CoreSample Project (Right):
using System;
namespace CoreSample
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

 **Demo** 



Bye, Bye, Multi-platform

# Windows Compatibility Pack

- Code Pages
- CodeDom
- Configuration
- Directory Services
- Drawing
- ODBC
- Permissions
- Ports
- Windows Access Control Lists (ACL)
- Windows Communication Foundation (WCF)
- Windows Cryptography
- Windows EventLog
- Windows Management Instrumentation (WMI)
- Windows Performance Counters
- Windows Registry
- Windows Runtime Caching
- Windows Services



<https://docs.microsoft.com/en-us/dotnet/core/porting/windows-compat-pack>

# Multi-platform-Code

## Einsatz mit Guarding

```
if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
{
    // Windows Code, Zugriff auf Registry
    using (RegistryKey key = Registry.CurrentUser.OpenSubKey(@"Software\Fabrikam\AssetManagement"))
    {
        if (key?.GetValue("LoggingDirectoryPath") is string configuredPath)
            Console.WriteLine(configuredPath);
    }
}
else if (RuntimeInformation.IsOSPlatform(OSPlatform.Linux))
{
    // Linux Code
    // ...
}
```

Install-Package Microsoft.Windows.Compatibility

Install-Package Microsoft.DotNet.Analyzers.Compatibility

 **Demo** 



# 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** 

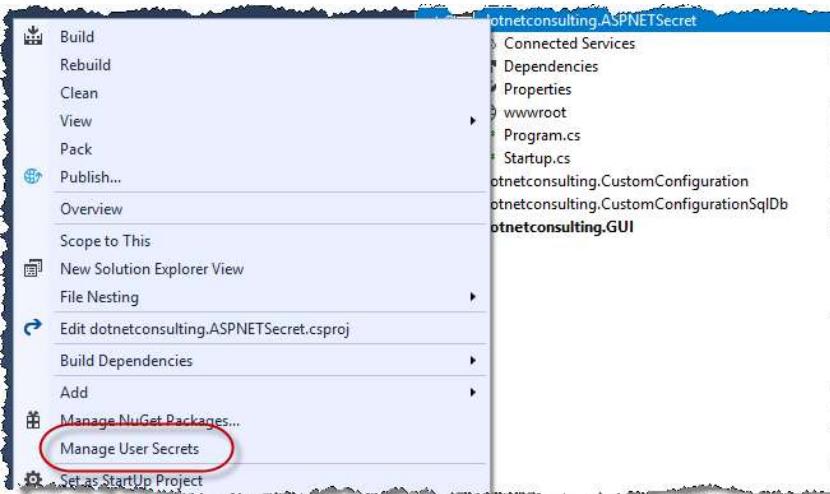
# ASP.NET User Secrets

Sensible Einstellungen auslagern

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <UserSecretsId>576504d1-3c17-4fc0-b0a0-23ee841be7e6</UserSecretsId>
  </PropertyGroup>
</Project>
```

Betriebssystem	Speicherort
Windows	%APPDATA%\microsoft\UserSecrets\<userSecretsId>\secrets.json
Linux	~/.microsoft/usersecrets/<userSecretsId>/secrets.json
MacOS	~/.microsoft/usersecrets/<userSecretsId>/secrets.json

# User Secrets verwalten



Aktion	.NET Core CLI
User Secret setzen	<code>dotnet user-secrets set Geheimnis "My Name is Bond"</code>
User Secrets auflisten	<code>dotnet user-secrets list</code>
User Secret löschen	<code>dotnet user-secrets remove Geheimnis</code>
Alle User Secrets löschen	<code>dotnet user-secrets clear</code>

 **Demo** 

# Eigenen Custom Provider schreiben

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

 **Demo** 

# EntityFramework Core konfigurieren

Nicht in `DbContext.OnConfiguring(...)`

```
private static IServiceProvider createDependencyInjectionContainer(IConfigurationRoot Configuration)
{
    ...
    // EF Context konfigurieren
    .AddDbContext<SamplesContext1>(
        o => o
            .UseLazyLoadingProxies()
            .UseSqlServer(Configuration["ConnectionStrings:EFConString"])
            .ConfigureWarnings(w => w.ThrowRelationalEventId.QueryClientEvaluationWarning())
            .EnableSensitiveDataLogging(true)
    )
    ...
}
```

 **Demo** 

# Eigene Komponenten richtig konfigurieren

## Beispiel „SmtpService“

- SmtpServiceConfig  
Konfiguration mit konkreten Werten, Quelle beliebig
- SmtpServiceConfigBuilder  
Validiert und erzeugt die Konfiguration
- SmtpServiceCollectionExtensions  
Stellt AddSmtpService () -Methode bereit

# Eigene Komponenten richtig konfigurieren

```
static private void ConfigureServices(IServiceCollection serviceCollection)
{
    ...
    // SMTP-Service konfigurieren
    serviceCollection.AddSmtpServer<RealSmtpService>(o =>
    {
        // Konkrete Werte aus Konfiguration, Quelle jedoch beliebig und hier unwichtig
        o.SetHostAndPort("192.168.1.1", 25);
        o.Sender = "produktion@dotnetconsultng.eu";
    });
    ...
}
```

 **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 Singleton
    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 – Konsole

```
IServiceProvider services = serviceCollection.BuildServiceProvider();  
  
services.GetService<IOrderService>().PlaceOrder("Wattestäbchen", 10);
```

# DI Service – Razor Pages

```
public class IndexModel : PageModel
{
    private readonly IOrderService _orderService;

    public IndexModel(IOrderService OrderService)
    {
        _orderService = OrderService;
    }

    public void OnGet()
    {
        // _orderService verwenden
    }
}
```

Gilt für: MVC und Web-API

# DI Service – Controller

```
public class HomeController : Controller
{
    private readonly IOrderService _orderService;

    public HomeController(IOrderService OrderService)
    {
        _orderService = OrderService;
    }

    public IActionResult Index()
    {
        // _orderService verwenden
    }
}
```

 **Demo** 

# IoC- und DI-Alternative – Autofac



<http://autofac.org>

```
Install-Package Autofac
```

```
Install-Package Autofac.Extensions.DependencyInjection
```

# Autofac-Setup

```
public IServiceProvider ConfigureServices(IServiceCollection services)
{
    // Andere Framework-Services hinzufügen
    // services.AddMvc();

    // Autofac hinzufügen und konfigurieren
    ContainerBuilder containerBuilder = new ContainerBuilder();
    containerBuilder.RegisterModule<DefaultModule>();
    containerBuilder.Populate(services);
    IContainer container = containerBuilder.Build();

    return new AutofacServiceProvider(container);
}
```

```
Install-Package Autofac
```

```
Install-Package Autofac.Extensions.DependencyInjection
```

# Autofac-Konfiguration

```
public class DefaultModule : Module
{
    protected override void Load(ContainerBuilder builder)
    {
        builder.RegisterType<EMailOrderService>()
            .As<IOrderService>()
            // Konstruktor-Parameter
            .WithParameter("SmtpHost", "127.0.0.1")
            // Eigenschaft
            .WithProperty("Sender", "tkansy@dotnetconsulting.eu");
    }
}
```



 **Demo** 

# Logging



# Logging

- 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.

# 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);
}
```

 **Demo** 

# 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  
}
```

 **Demo** 

# Logging-Scopes

Bereich im Logging zwecks besserer Leßbarkeit

```
using (logger.BeginScope($"Scope y = {y}"))
{
    for (int z = 0; z < 2; z++)
    {
        logger.LogDebug($"y = {y}, z = {z}");
    }
}
```

 **Demo** 

# Logging in Action I

```
logger.Log(LogLevel.Trace, "Trace");
logger.Log(LogLevel.Debug, "Debug");
logger.Log(LogLevel.Information, "Information");

logger.LogWarning("Warning");
logger.LogError("Error");
logger.LogCritical("Critical");
```

# Logging in Action II

```
// Wenn etwas schief geht
try
{
    throw new Exception("Passierschein A38 nicht gefunden.");
}
catch (Exception ex)
{
    logger.LogError(ex, "Error");
    throw;
}
```

# Logging in Action III

```
// Wenn mal etwas richtig schief geht
try
{
    throw new Exception("Passierschein A38 nicht vorhanden.");
}
catch (Exception ex)
{
    logger.LogCritical(ProgramEventId.PermitNotFound, ex,
                      "Permit A38 not found ");
    throw;
}
```

# Logging in Action IV

```
// Oder wenn etwas nicht dort ist, wo es sein sollte
string importFilename = @"c:\nix\app.config";
try
{
    string import = File.ReadAllText(importFilename);
}
catch (Exception ex)
{
    logger.LogCritical(ProgramEventId.ImportFileFailed, ex,
        "Import file '{0}' not ", importFilename);
}
```

 **Demo** 

# EventId

```
public static class ProgramEventId
{
    public static readonly EventId CriticalSituation201
        = new EventId(201, "CriticalSituation201");

    public static readonly EventId PermitNotFound
        = new EventId(202, "PermitNotFound");

    public static readonly EventId ImportFileFailed
        = new EventId(203, "ImportFileFailed");
}
```

 **Demo** 

# Logging & Dependency Injection

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

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

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

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

 **Demo** 

# Logging in Datei – Serilog

```
// Filename fürs Logging
string fileName = Path.Combine(ApplicationContext.BaseDirectory, "Logging.txt");

// Logger Factory konfigurieren;
ILoggerFactory loggerFactory = new LoggerFactory()
    .AddFile(fileName, LogLevel.Information);

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



<https://github.com/serilog/>

Install-Package Serilog.Extensions.Logging.File

 **Demo** 

# NLog – Fully-featured Logging Framework



The screenshot shows a web browser displaying the official NLog project website at [nlog-project.org](http://nlog-project.org). The page has an orange header with the NLog logo and navigation links for Home, Getting started, News, Documentation, Support, and Changelog. Below the header is a large orange section containing the NLog logo, the text "Flexible & free open-source logging for .NET", a "Download NLog" button, and information about the latest version (4.5). At the bottom of this section are social media links for GitHub, Fork, Twitter, and Facebook. To the right of the main content is a sidebar titled "Tweets about nlogofficial" showing a few tweets from the official account.



<http://nlog-project.org>

```
Install-Package NLog
```

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

 **Demo** 

# Allgemeines Deployment



# Deployment

- Framework-dependent Deployment (FDD)
- Self-contained Deployment (SCD)
- Visual Studio 2017: Publish
- Visual Studio 2017: Pack
  - Nuget-Paket erstellen
- .NET Core CLI

# Visual Studio 2017: Publish

- FDD
  - Project -> Publish
- SCD
  - Project -> Publish

```
== *.csproj-Daten ==  
<PropertyGroup>  
  <RuntimeIdentifiers>win10-x64;osx.10.11-x64;linux-x64</RuntimeIdentifiers>  
</PropertyGroup>
```

2.0?

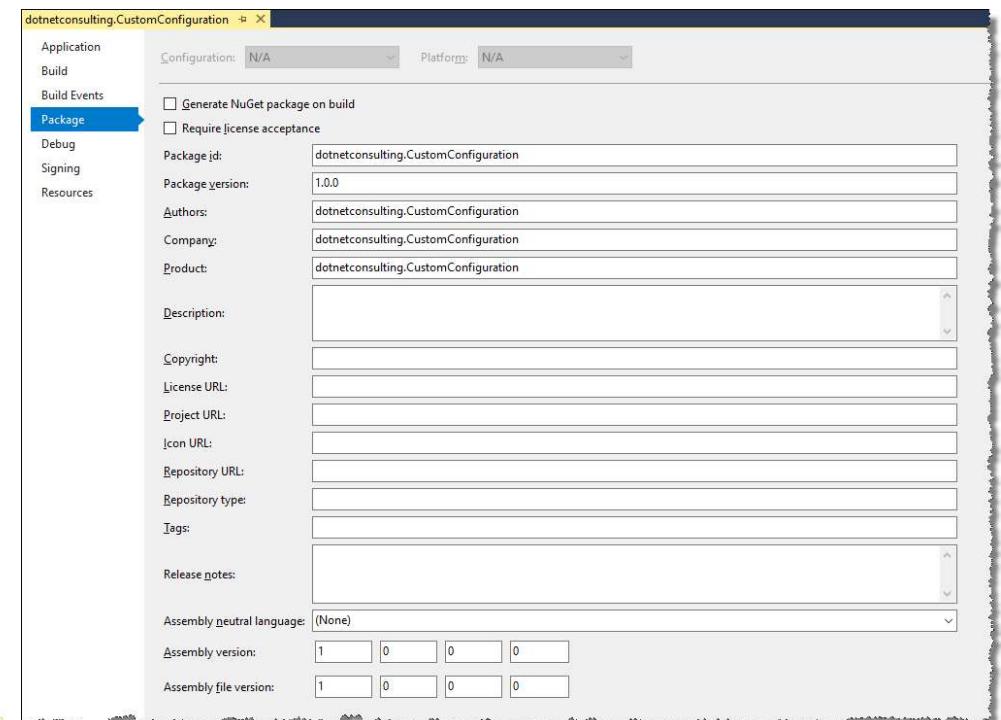


<https://docs.microsoft.com/en-us/dotnet/core/deploying/deploy-with-vs>

 **Demo** 

# Visual Studio 2017: Pack

- Erstellt ein NuGet-Paket
  - nuget.exe aus VS heraus
- Details:
  - Projekt -> Properties -> Package



 **Demo** 

# .NET Core CLI

## Framework-dependent Deployment

```
dotnet publish -c Release
```

## Self-contained Deployment

```
dotnet publish -c Release -r win10-x64  
dotnet publish -c Release -r ubuntu.16.10-x64  
dotnet publish -c Release -r osx.10.11-x64
```

 **Demo** 



# Migration

# Migration zu .NET Core

1. Dritthersteller prüfen
2. Projekte auf .NET Framework 4.6.2 umstellen
3. API Portability Analyzer Tool
4. Testportierung
5. ...



<https://github.com/Microsoft/dotnet-apiport/>



# Fragen?

# Links



<https://www.visualstudio.com/de/downloads>



<https://github.com/ElectronNET/Electron.NET>



<https://github.com/AvaloniaUI/Avalonia>



<https://github.com/mellinoe/ImGui.NET>