SDN MAGAZINE

Nummer
141
Dec 2021

SDN Magazine is
een uitgave van:
SDN



SDN volledig vernieuwd!

- gRPC Code First
- De volgende stap in je carrière
- Service Locator Pattern – In Unity3D


MAKE A DIFFERENCE AS CHAPTER LEAD
#STARTWITHUS
Are you working as a Chapter Lead and are you looking for a challenging next step? Discover what ABN AMRO has to offer you by visiting
werkenbijabnamro.nl/en/
ABN·AMRO

Colofon

Uitgave
Software Development Network
26ste jaargang
Nr. 141 • November 2021

Bestuur van SDN
Marcel Meijer
Martin Smelt
Roy Janssen

Auteurs:
Eric van Olst & Patrick Vroegh, Johan Smarius, Vincent Hendriks, Patrick Bes, Steef-Jan Wiggers, Sander Hoogendoorn, Koen Aerts, Roy Janssen, Arno Peters, Menno Jongerius en Iris den Hoorn

Listings
Zie de website www.sdn.nl voor eventuele source files uit deze uitgave.

Contact
Software Development Network
info@sdn.nl
redactie@sdn.nl

©2021 Alle rechten voorbehouden. Niets uit deze uitgave mag worden overgenomen op welke wijze dan ook zonder voorafgaande schritelijke toestemming van SDN. Tenzij anders vermeld zijn artikelen op persoonlijke titel geschreven en verwoorden zij dus niet noodzakelijkerwijs de mening van het bestuur en/of de redactie. Alle in dit magazine genoemde handelsmerken zijn het eigendom van hun respectievelijke eigenaren.

Adverteren
Informatie over adverteren en de tarieven kunt u opvragen bij Kelly Verschoor:
kelly.verschoor@sdn.nl

Beste SDN-ers,

Eindelijk zien jullie waar wij de afgelopen maanden achter de schermen druk mee bezig waren, het hernieuwde en gerestylede SDN-magazine. Ons frisse nieuwe logo hebben jullie al gezien en deze frisse nieuwe kleurstelling hebben we ook doorgevoerd in het magazine. En dit magazine heb je weer als gedrukt exemplaar van je deurmat of uit je brievenbus geplukt. Niets zo lekker als door een papieren magazine bladeren. Heerlijk.

Wat er aan het magazine niet is veranderd, zijn de mooie verhalen van onze collega's uit het veld! Collega's die de tijd hebben genomen om hun ervaringen en hun mening over veelal technische onderwerpen op te schrijven. Deze verhalen zijn stuk voor stuk met zorg samengesteld door de auteur en ook jij kunt zo'n auteur zijn of worden. Mail je artikel voorstel naar redactie@sdn.nl en we maken er samen een mooi artikel van. Of je nu een beginner of een volleerd boekschrijver bent, iedereen is van harte uitgenodigd om te schrijven. Net zoals Eric van Olst & Patrick Vroegh, Johan Smarius, Vincent Hendriks, Patrick Bes, Steef-Jan Wiggers, Sander Hoogendoorn, Koen Aerts, Roy Janssen, Arno Peters, Menno Jongerius en Iris den Hoorn hebben gedaan. Stuk voor stuk mooie actuele onderwerpen van C# en RPC naar Logging en Versie nummering en keuzes in loopbaan of workplaces, maar ook Logic apps, threads, patterns en frontend terwijl je door een doolhof gaat. De redactiecommissie heeft er veel plezier aangehad om de artikelen te mogen proeflezen.

Heel veel plezier met lezen! Oh en mocht je je magazine kwijtraken aan een andere collega developer op het Microsoft platform, mail ons dan sturen we je een nieuw exemplaar op. Beter nog laat de collega developer zich inschrijven op www.sdnl.nl, dan blijft hij/zij altijd op de hoogte en krijgt hij/zij altijd het magazine thuisgestuurd.

Dit is de eerste van weer velen, hou je staande in de tsunami van SDN-magazines.

Groeten namens de redactie,
Marcel Meijer
Voorzitter en eindredacteur.

Over Marcel Meijer
Marcel Meijer is Freelance (Enterprise) Architect, Teamlead, CTO, CTO Advisor of interim manager. Daarnaast is Marcel voorzitter, bestuurslid, event organisator en eindredacteur bij de SDN. Sinds 2010 mag hij zich MVP noemen.


DJ
DEVITJOBS.NL
DE WEG NAAR EEN NIEUWE TECH JOB
Bezoek de site, solliciteer direct of meld je aan om de nieuwste vacatures te ontvangen.
devitjobs.nl/

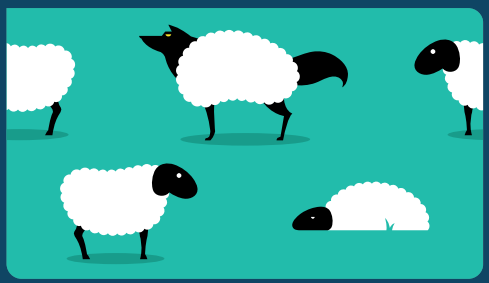
In dit nummer

06 Nieuwe Features in C#9

11 RPC Code First

18 Een introductie van Structured Logging

22 Threat Modeling



30 Realiseren van Integratie oplossingen Met Azure logic Apps

36 Modern workplaces - we are continuously uncovering'

38 Loopbaanstress! Manager, architect of developer?

42 Laat je versienummer aan je pipeline over!

44 Service Locator Pattern



52 Over het doolhof dat front-end ontwikkeling heet

54 De magie is online samenwerking

6 Met de komst van .NET 5 en C# 9.0 zijn er heel veel waardevolle toevoegingen gedaan aan de programmeertaal. In deze whitepaper gaan we in op de belangrijkste toevoegingen en om C# 9.0 te kunnen gebruiken dien je eerst de .NET 5 runtime of SDK te installeren. Ga hiervoor naar https://dotnet.microsoft.com/download. Na installatie zal C# 9.0 de standaard C# versie zijn bij elk nieuw project gebaseerd op .NET 5.

18 Je gooit een app in productie en dat is het dan ... Toch? Vroeger was het vaak dat je eigenlijk geen zicht had. Je applicatie ging als het ware een roadtrip maken waar je niet direct bij kon zijn. Als er dan iets mis was dan hoorde je dat vaak achteraf en indirect. Via gebruikers bijvoorbeeld. Vervolgens kon je de logs achterhalen door ze op te gaan vragen bij een beheerder om vervolgens te gaan ploeteren door een redelijk grote hoeveelheid tekst, maar haalde je er dan ook echt uit wat je nodig had?

42 Het valt me regelmatig op dat versioning van .NET assemblies in CI/CD pipelines niet altijd volledig wordt geautomatiseerd. Ik vind dat altijd ontzettend jammer. Het hele doel van een pipeline is toch om snel en consistent bruikbare pakketjes op te leveren? Dan zou het ook fijn zijn als het juiste versienummer automatisch bepaald kan worden.

Nieuwe features in C#9

Met de komst van .NET 5 en C# 9.0 zijn er heel veel waardevolle toevoegingen gedaan aan de programmeertaal. In deze whitepaper gaan we in op de belangrijkste toevoegingen en om C# 9.0 te kunnen gebruiken dien je eerst de .NET 5 runtime of SDK te installeren. Ga hiervoor naar https://dotnet.microsoft.com/download. Na installatie zal C# 9.0 de standaard C# versie zijn bij elk nieuw project gebaseerd op .NET 5.

Auteurs: Eric van Olst & Patrick Vroegh

PATTERN MATCHING (UITBREIDING)

Waarom?

De syntax van ontwikkeltalen is al sinds jaren het onderwerp van discussies tussen ontwikkelaars. Leesbaarheid, efficiëntie en flexibiliteit is slechts een selectie van de tientallen argumenten die jouw mening vormen over het al dan niet geslaagd zijn van een syntax. Martin Fowler geeft een – zoals we van hem gewend zijn – zeer zinvolle bijdrage in deze discussie met zijn bekende uitspraak: *Any fool can write code that a computer can understand. Good programmers write code that humans can understand.*
Met die uitspraak in het achterhoofd heeft Microsoft een aantal vereenvoudigingen aan de syntax van `if` en `switch` doorgevoerd. Deze onderdelen worden door ons veel gebruikt om complexe condities uit te werken. Sinds C# 7.0 helpt Pattern Matching ons om de code die we opleveren beter leesbaar en dus minder foutgevoelig te maken. Ook in C# 9.0 zijn er weer enkele nieuwe mogelijkheden toegevoegd.

Hoe?

Logical Pattern. Om meer complexe condities goed leesbaar te houden is het Logical Pattern toegevoegd aan C# 9.0. Hiermee kunnen we gebruik maken van `and`, `or` en `not` in onze statements, en het herhalen van de parameter waarmee vergeleken moet worden kan achterwege worden gelaten. **Voorbeeld 1.**
Dit is een krachtig mechanisme dat kan worden gebruikt in if-statements zoals in bijgaand voorbeeld. De volgorde waarin de controles worden uitgevoerd is volstrekt logisch, maar in sommige gevallen zul je toch aan deze volgorde moeten wennen. Vergeet dus niet om unittests toe te voegen met voldoende testsets om er zeker van te zijn dat je functie in alle gevallen het gewenste gedrag vertoont.
Het Logical Pattern komt zelfs beter tot zijn recht wanneer dit wordt gecombineerd in switch expressions, zoals we hierna zullen zien bij het Relational Pattern.

Relational Pattern

In C# 8.0 zijn switch expressions toegevoegd. Hiermee was het mogelijk om direct een returnwaarde als resultaat van een aantal voorwaarden te definiëren. De syntax bevatte veel onnodige herhalingen en was daarom nog niet in alle gevallen even leesbaar, zoals het onderstaande voorbeeld illustreert. **Voorbeeld 2.**
In C# 9.0 kunnen we de bovenstaande syntax verder verfijnen door gebruik te maken van het Relational Pattern. Hierdoor kunnen we rechtstreeks gebruik maken van Relational operators zoals `<` of `>=`. Toepassing leidt tot het volgende resultaat: **Voorbeeld 3.**

```
// Voor de introductie van Logical patterns in C# 9.0
if (price != 0 && price < 10 || price > 20) return true;

// Mogelijk vanaf C# 9.0
if (price is not 0 and < 10 or > 20) return true;
```

1

```
public int CalculateTicketPriceCsharp8(Person person)
{
    return person switch
    {
        Person p when p.Age <= 12 => 5,
        Person p when p.Age > 12 && p.Age <= 60 => 15,
        Person p when p.Age > 60 => 10,
    };
}
```

2

```
public int CalculateTicketPriceCsharp9(PersonRecord person)
{
    return person.Age switch
    {
        <= 12 => 5,
        > 12 and <= 60 => 15,
        > 60 => 10,
    };
}
```
3

```
public static decimal PropertyMatchingTypeExpression(Vehicle vehicle) =>
    vehicle switch
    {
        Car => 6.00m,
        Taxi => 4.50m,
        Bus => 8.00m,
        DeliveryTruck => 12.00m,
        _ => throw new Exception("Cannot calculate fee for provided vehicle type")
    };
```
4

In deze syntax ligt de nadruk op de logica en het resultaat, en veel minder op het achterhalen van de juiste objecten.
We zien ook dat het gebruik van het Logical Pattern ook daadwerkelijk de leesbaarheid van de code vergroot: in één oogopslag is nu duidelijk dat er een selectie gemaakt wordt voor iemand tussen de 12 en 60.
Dit voorbeeld leent zich daarnaast om de *switch expression*, die in C# 8.0 al is geïntroduceerd nogmaals onder de aandacht te brengen.
Bij de switch expression staat de variabele vóór het `switch` keyword, de switch-onderdelen `case` en `:` zijn vervangen door de bekende =>. Als laatste is de body is geen statement meer, maar een expressie.
De expression is in het onderstaand voorbeeld uitgewerkt. Hierbij zien we dat we ons niet hoeven te beperken tot property checking, ook Type checking behoort tot de mogelijkheden. **Voorbeeld 4.**

Type Pattern

Met het Type Pattern kun je met een vereenvoudigde syntax conditie checks op type en properties uitvoeren. De nieuwe schrijfwijze in C# 9.0 is compact en overzichtelijk, zoals in onderstaand voorbeeld wordt getoond. **Voorbeeld 5.**
Wanneer de parameter person niet van het type `PersonRecord` is ontstaat er geen exception maar wordt de evaluatie van het if-statement afgebroken. In het andere geval (person is van het type `PersonRecord`) wordt het resultaat bepaald op basis van de property `Age` van `person`. Wees je er in dit voorbeeld goed van bewust dat deze code géén `NullReferenceException` oplevert wanneer de parameter `person` null is.

```
public bool PatternMatchWithIfStatement(object person)
{
    if (person is PersonRecord { Age: < 20 })
        return true;

    return false;
}
```
5

```
using System;

namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```
6

```
System.Console.WriteLine("Hello World!");
```
7

Verder?

Veel van onze logica in onze applicaties is vormgegeven door implementaties van if-then-else en switch statements. Na verloop van tijd kunnen deze constructies steeds complexer en onoverzichtelijker worden. Met de nieuwe mogelijkheden van pattern matching kunnen we deze bestaande code relatief eenvoudig refactoren naar beter leesbare en dus beter onderhoudbare code.
Maar natuurlijk geldt zoals altijd: bedenk of deze complexe constructies gezien kunnen worden als code smell voor het overtreden van belangrijke SOLID principes, zoals het Open/Closed of Single Responsibility principe. Overweeg in die gevallen vooral ook de toepassing van nette OO oplossingen, zoals het inzetten van encapsulation en de vele design patterns die beschikbaar zijn.

Top level statements

Waarom? Elke .NET assembly die uitvoerbaar is, dient een entrypoint te hebben anders heeft de applicatie geen startpunt. Dit startpunt is de welbekende `Main` methode. Zelfs een simpele console applicatie heeft een `Main` methode nodig om te kunnen functioneren.
Het requirement van zo'n `Main` methode maakt dat C# een ietwat minder toegankelijke programmeertaal is voor beginners. Wanneer beginners de volgende code onder ogen krijgen, kan dat best intimiderend overkomen: **Voorbeeld 6.**
Het enige wat bovenstaande code moet doen is de tekst "Hello World!"

tonen op het scherm, maar voordat de applicatie zover is moet er dus eerst heel veel "ceremonie" plaatsvinden:

> `using` statements.
> `namespace` definitie.
> `class` definitie.
> `Main` methode definitie (het startpunt).

De nieuwe C# 9.0 feature "Top-level Statements" maakt deze ceremonie overbodig en dus hopelijk ook wat meer toegankelijk voor beginners!

Hoe?

De volgende code is vanaf C# 9.0 uitvoerbaar geworden op zichzelf, zonder extra code: **Voorbeeld 7.** Zoals je kan zien is deze code van alle "ceremonie" ontdaan, dus geen `class` definitie en geen `Main` methode definitie meer. Je kunt je vast voorstellen dat dit veel beter te begrijpen is voor een beginner dan de eerdere voorbeeldcode.
`using` statements zijn gewoon nog te gebruiken om te voorkomen dat je steeds de fully qualified type namen moet specificeren: **Voorbeeld 8.**

```
using System;

Console.WriteLine("Hello World!");
```
8

Verder?

Ondanks dat er geen `Main` methode is gedefinieerd, ben je nog steeds in staat om de `args` parameter te gebruiken om de command-line parameters uit te kunnen lezen: **Voorbeeld 9.** Wanneer de Top-level Statements het `await` keyword bevatten, zal de C# compiler onderwater automatisch een `async` entrypoint maken: **Voorbeeld 10.** De mogelijkheid bestaat nog steeds om een `int` waarde terug te geven, welke aangeeft wat de zogenaamde exitcode is van je applicatie. **Voorbeeld 11.**

```
using System;
using System.Text.RegularExpressions;

Console.WriteLine(Regex.IsMatch(args[0], args[1]) ? "Match!" : "No match!");
```
9

```
using System.IO;

await File.WriteAllTextAsync(@"c:\temp\output.txt", "Hello World!");
```
10

```
try
{
    /* ... */
    return 0; // Succes
}
catch
{
    return 1; // Error!
}
```
11

Maar?

Bij het gebruik maken van Top Level Statements zijn er wel een tweetal beperkingen waar je rekening mee moet houden.

> 1. Top Level Statements kunnen slechts in één bestand onder het project aanwezig zijn.
> 2. In principe geen nieuwe regel, maar wanneer er Top Level Statements aanwezig zijn in een project, mogen er geen andere entrypoints bestaan.

Immutability verbeteringen

Stel je programmeert een object waarvan je wilt dat de waardes niet gewijzigd worden nadat het object geïnstantieerd is. Dit was in het verleden lastig in code te realiseren. Je kunt met private setters werken, maar dan zijn objecten vaak niet goed te serialiseren. In de praktijk kiezen de meeste ontwikkelaars er dan maar voor om het object "mutable" te houden. Het vervelende is dat de code dan niet optimaal de intentie van de programmeur weergeeft. Dit was altijd al een probleem bij bijvoorbeeld data transfer objects, maar met de populariteit van domain driven design en de bijbehorende value objects, wordt het nog belangrijker om een betere oplossing te vinden.

Microsoft moet dit ook gedacht hebben, toen ze in C#9 de init-only setters en de recordtypes introduceerde. Deze nieuwe code constructies maken het mogelijk om op eenvoudige manier objecten te maken die immutable zijn, en die op basis van hun waardes met elkaar vergeleken kunnen worden.

Init only properties

Waarom? De Init Only Setters zijn aan C# toegevoegd om de ontwikkelaars een betere ervaring te geven met het maken en gebruiken van immutable classes of structs, door een extra moment te introduceren waarop data gemodificeerd mag worden.
Voorheen werd immutability verkregen door members `readonly` te maken of door properties niet te voorzien van een public setter. Het probleem wat hiermee ontstond, was dat je de data van het object alleen kon initialiseren via constructor parameters. Ook middels de in C# 3.0 geïntroduceerde object initializers, was het niet mogelijk om de data voor het object bij initialisatie in te stellen: **Voorbeeld 12.**

```
var person = new Person()
{
    Id = 1
};
public class Person
{
    public int Id { get; private set; }
}
```
12

```
var person = new Person()
{
    Id = 1
};

public class Person
{
    public int Id { get; init; }
}
```
13

Hoe?

In C# 9.0 is het nu wel mogelijk om immutable properties voor een object te initialiseren middels de object initializer. Die doe je door de setter van de property te vervangen door

```
// Voorheen
public abstract class Vehicle
{
    public int TireCount { get; }
    public Vehicle()
    {
        // Toegestaan sinds C# 6.0
        TireCount = 4;
    }
}

public class Bike : Vehicle
{
    public Bike()
    {
        // Niet toegestaan
        // voor C# 9.0
        TireCount = 2;
    }
}
```

```
// Nu
public abstract class Vehicle
{
    public int TireCount { get; init; }
    public Vehicle()
    {
        TireCount = 4;
    }
}

public class Bike : Vehicle
{
    public Bike()
    {
        // Nu wel toegestaan!
        TireCount = 2;
    }
}
```

14

```
// Voorheen:
public class Animal
{
    public bool _doesBark;
    public string Race { get; }
    public bool DoesBark
    {
        get => _doesBark;
        set
        {
            _doesBark = value;
            if (_doesBark)
            {
                // Niet toegestaan
                // voor C# 9.0
                Race = "Dog";
            }
        }
    }
}
```

```
// Nu:
public class Animal
{
    public bool _doesBark;
    public string Race { get; init; }
    public bool DoesBark
    {
        get => _doesBark;
        init
        {
            _doesBark = value;
            if (_doesBark)
            {
                // Nu wel toegestaan!
                Race = "Dog";
            }
        }
    }
}
```

15

```
public record Racecar(string Name, int LengthInCm);
```

16

het nieuwe `init` keyword: **Voorbeeld 13.** Hiermee wordt dus voorkomen dat je expliciet constructor parameters voor alle properties moet aanmaken. Het geeft de ontwikkelaar dus veel vrijheid in het initialiseren van objecten.

Verder?

Het kunnen instellen van immutable properties middels de object initializer is verreweg het grootste voordeel wat de Init Only Setters feature biedt. De nieuwe feature kan echter breder ingezet worden, onder andere op de volgende manieren:

> Setters van properties kunnen nu in de instance constructor van de afgeleide class ook benaderd worden: **Voorbeeld 14**

> Init Only Setters kunnen ook benaderd worden vanuit de body van een andere Init Only Setter: **Voorbeeld 15**

Record types

Waarom? In deel 2 van deze serie is al gewezen op init-only setters.

Op een eenvoudige manier kun je daarmee aangeven dat properties als immutable behandeld moeten worden.
Met het nieuwe keyword `record` krijg je als programmeur de mogelijkheid om snel en overzichtelijk een complete class immutable te maken. Je kunt dit record dan instantiëren, waarbij de data initieel wordt gezet en daarna niet meer te wijzigen is.
Dit levert je een aantal voordelen op. Denk aan toepassing van deze record types in een multi-threaded applicatie. Omdat het object immutable is, met andere woorden niet kan worden benaderd om wijzigingen door te voeren, kan er geen race condition optreden waarbij thread 1 gegevens leest terwijl thread 2 gegevens schrijft. Je code kan worden vereenvoudigd zonder risico op deadlocks of nog erger, berekeningen met een onverwachte uitkomst.
Een ander groot voordeel van een immutable record is dat je de betrouwbaarheid van je unittesten kunt vergroten. Stel je een applicatie voor waarbij een "ouderwetse" DTO-class door diverse methodes in een keten als parameter wordt gebruikt. Wanneer de inhoud van deze class mutable is, zul je bij het schrijven van een test op één van die methodes uit moeten zoeken of er methodes bestaan die gegevens in jouw DTO kunnen wijzigen. Dit kan namelijk impact hebben op het aantal scenario's dat je zult moeten testen. Een immutable record maakt

> Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

```
var ferrariRacecar = new Racecar(Name: "Ferrari SF1", LengthInCm: 557);
var redbullRacecar = new Racecar("Redbull RB16B", 551);
ferrariRacecar.
```

Deconstruct
Equals
GetHashCode
GetType
LengthInCm — int Racecar.LengthInCm { get; init; }
Name
ToString

17

```
public record PersonRecord
{
    public string Firstname { get; init; } = string.Empty;
    public int Age { get; init; }
}
```

18

```
public void Records_should_implement_value_comparison()
{
    // GIVEN
    var person1 = new PersonRecord { Firstname = "Menno" };
    var person2 = new PersonRecord { Firstname = "Menno" };

    var comparison = person1 == person2;

    // THEN
    Assert.IsTrue(comparison);
    Assert.AreEqual(person1, person2);
}
```

19

```
var person1 = new PersonRecord { Firstname = "Menno", Age = 40 };
var person2 = person1 with { Firstname = "Erik" };
```

20

```
var ferrariRacecar = new Racecar(Name: "Ferrari SF1", LengthInCm: 557);
Console.WriteLine(ferrariRacecar);
// Output:    "Racecar { Name = Ferrari SF1, LengthInCm = 557 }"
```

21

het onmogelijk dat gegevens worden gewijzigd, waardoor je veilig deze uitzoekklus over kunt slaan.

Hoe?

Declareren en instantiëren C# 9.0 biedt twee mogelijkheden om een record te declareren. Het meest eenvoudig is de volgende manier, waarbij je *positional arguments* toepast: **Voorbeeld 16.**
Deze methode is intuïtief en overzichtelijk. De C# compiler voegt boilerplate-code zoals een constructor met parameters en een deconstructor toe zodat je direct een Object van het type Car kunt aanmaken en gebruik kan maken van enkele handige uitbreidingen. Uit de intellisense blijkt dat de aangegeven property LengthInCm gebruikt maakt van `init` als setter en dus inderdaad immutabel is.
Er is géén parameterloze constructor aanwezig, je bent dus verplicht om de parameters via de constructor te vertrekken. **Voorbeeld 17.**
Veel controle over `Racecar` heb je natuurlijk niet, nuttige zaken als het inbouwen van validatie of het aangeven van een default value is zo niet mogelijk. Daarom is ook de meer klassieke ogende methode beschikbaar waardoor je bijvoorbeeld berekende properties of methodes kunt toevoegen. **Voorbeeld 18.**

Vergelijken van records

Een mooie eigenschap van een record is dat het zuiver gezien een reference type is, maar dat het zich gedraagt als een value type. Dit heeft een belangrijk effect op het vergelijken van instances van records. Bij de vergelijking wordt namelijk – net als bij een echt value type – niet gekeken of de instances naar hetzelfde object verwijzen, maar of het type en de inhoud van de record instanties overeenkomen.
De bijgaande test brengt dit mooi in beeld. Er worden 2 aparte instanties van `PersonRecord` aangemaakt en vervolgens vergeleken via de `==-operator` en de `AreEqual` methode. Met classes zijn we gewend dat dit verschillende objecten zijn en daarom verwachten we dat de test faalt. Maar omdat records zich gedragen als value types slaagt deze test. **Voorbeeld 19**.

Muteren van een record

Hoewel een record immutable is, is het wel mogelijk een kopie te maken van een instantie en daarbij kleine wijzigingen aan te brengen met behulp van het keyword `with` op de volgende manier: **Voorbeeld 20.**

Verder?

Bij het record type krijg je nog een aantal andere methodes tot je beschikking.

`ToString()`

Deze methode levert automatisch de inhoud van het type en alle publieke properties. **Voorbeeld 21.**

`Deconstructor()`

De deconstructor geeft je de mogelijkheid om op de volgende manier de gegevens uit je record te verzamelen. **Voorbeeld 22.**

```
var ferrariRacecar = new Racecar(Name: "Ferrari SF1", LengthInCm: 557);
var (naam, lengte) = ferrariRacecar;
```

22

gRPC Code First

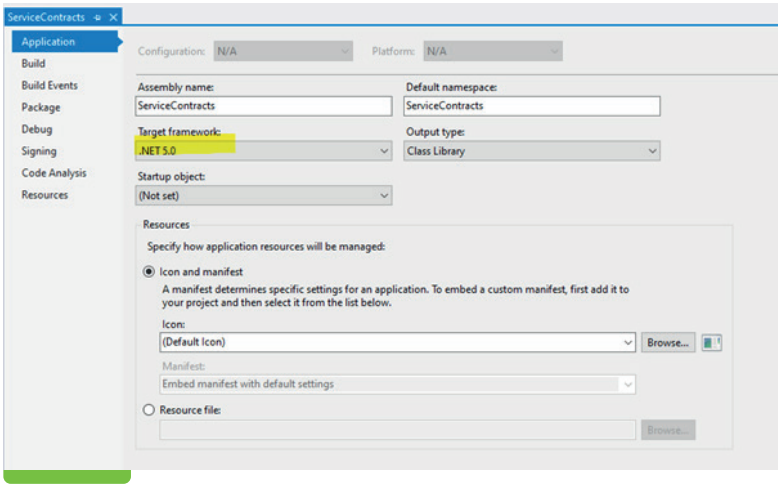
In de laatste twee edities van het magazine heb ik je laten zien hoe je gRPC en gRPC-Web binnen .NET kunt gebruiken op basis van een contract geschreven in protobuf. Dit is alleen niet de enige aanpak. In dit artikel zal ik je laten zien hoe je kunt werken met gRPC met een code first aanpak.

Auteur: Johan Smarius

De NuGet-packages die je hiervoor nodig hebt, zijn niet afkomstig van Microsoft, maar van het open source project "protobuf-net.Grpc" (https://github.com/protobuf-net/protobuf-net.Grpc).

Opzetten van het contract

Bij het gebruik van een code first aanpak moet je het contract in code op gaan nemen. Deze code zal moeten worden gedeeld tussen de server en de client. De gemakkelijkste manier om dit te doen, is om deze code op te nemen in een class library. Vanuit de server en de client kun je dan gemakkelijk een referentie opnemen naar deze class library. Als je je servercode in .NET 5 wilt maken, is het wel handig om ook het target framework voor deze class library op .NET 5 te zetten. Helaas



Figuur 1, Instellen Target Framework.

kun je deze keuze nog niet maken bij het aanmaken van de library, dus dit moet je na het aanmaken even nog instellen bij de properties van je project **(Figuur 1)**.
Voordat je het contract kunt coderen, moet je de NuGet-package "protobuf-net.Grpc" toevoegen aan het nieuw gemaakte project.
Je kunt nu het contract op twee manieren gaan definiëren. Voor contracten kun je gebruik maken van de types DataContract/DataMember **(Figuur 2)** en ServiceContract **(Figuur 3)**. Je kent deze misschien nog wel van WCF.
Bij het datacontract is het belangrijk om het volgnummer van het veld expliciet op te nemen, net zoals je dat ook moet doen als je gebruik maakt van de normale protobuf-aanpak.
In het datacontract kun je gebruik maken van de normale .NET typen.

Voor de interface moet je het attribuut "ServiceContract" gebruiken. De naam van de service wordt normaal bepaald aan de hand van de typenaam van de interface. Je kunt dit echter sturen door expliciet de naam op te geven bij het attribuut, zoals in de code van Figuur 3 gedaan is. In principe worden alle methoden binnen deze interface als beschikbare methoden voor het contract beschouwd. Om dit toch explicieter te maken, wordt in de voorbeeldcode altijd gebruik gemaakt van het attribuut "OperationContract".
Als returntype voor een methode kun je gebruik maken van een DataContract-type, void, Task of zoals in de voorbeeldcode gebruikt is ValueTask.
Bij de normale protobuf-aanpak wordt standaard een synchrone en asynchrone methode gegenereerd op basis van het protobuf-bestand. Bij de code first aanpak wordt er geen code gegenereerd en moet je zelf expliciet een keuze maken door te kiezen voor

BIO

Johan Smarius

Johan Smarius is een ervaren (lead) software developer, architect, spreker en trainer op het Microsoft .NET vlak met meer dan 25 jaar ervaring in de IT. Hij heeft ervaring met .NET sinds versie 1.0. Johan werkt als docent informatica bij Avans Hogeschool in Breda en als technical consultant bij zijn eigen bedrijf JMAC Software Solutions. In zijn vrije tijd leert Johan kinderen programmeren en is hij instructeur EHBO.

Twitter: @JohanSmarius
Website: https://www.johans-marius.com

Figuur 2, DataContract.

```
[DataContract]
2 references | Johan Smarius, 6 days ago | 1 author, 1 change
public class EchoRequest
{
    [DataMember(Order = 1)]
    2 references | Johan Smarius, 6 days ago | 1 author, 1 change
    public string Message { get; set; }
}
```

```
[ServiceContract(Name = "SDN.EchoService")]
2 references | Johan Smarius, 6 days ago | 1 author, 1 change
public interface IEchoService
{
    [OperationContract]
    2 references | Johan Smarius, 6 days ago | 1 author, 1 change
    ValueTask<EchoResponse> EchoMessage(EchoRequest request);
}
```

Figuur 3, ServiceContract.

een synchroon of asynchroon return-type. De code van een service wil je natuurlijk het liefst zo veel mogelijk asynchroon laten uitvoeren, dus in dit geval is Task of ValueTask het meest van toepassing. Om nu expliciet te maken dat een enkele waarde uit de methode wordt geretourneerd, is in deze voorbeeldcode gebruik gemaakt van ValueTask.

Voor simpele contracten waarbij je geen low-level invloed nodig hebt op het contract, voldoet deze aanpak. Als je meer controle nodig hebt, dan kun je in plaats van DataContract gebruik maken van de types ProtoContract en ProtoMember **(Figuur 4)**. Beide contracten lijken heel erg op elkaar. Er wordt alleen gebruik gemaakt van andere attributen en de parameter voor het volgnummer hoef je niet meer expliciet te benoemen.

Voor dit eenvoudige voorbeeld is het gebruik van ProtoContract niet direct nodig, maar als je bijvoorbeeld gebruik wilt maken van DateTime typen in je contract, dan heb je deze constructie wel nodig **(Figuur 5)**.

Met de code first aanpak hoef je geen gebruik meer te maken van het speciale type "google.protobuf.Timestamp", maar moet je wel specificeren dat de omzetting door het protobuf-net.Grpc package moet

Figuur 4, Protocontract.

```
[ProtoContract]
2 references | Johan Smarius, 6 days ago | 1 author, 1 change
public class EchoRequest
{
    [ProtoMember(1)]
    2 references | Johan Smarius, 6 days ago | 1 author, 1 change
    public string Message { get; set; }
}
```

```
[ProtoContract]
public class CreateOrderRequest
{
    [ProtoMember(1)]
    public int CustomerId { get; set; }

    [ProtoMember(2)]
    public CustomerType CustomerType { get; set; }

    [ProtoMember(3, DataFormat =  DataFormat.WellKnown)]
    public DateTime OrderedAt { get; set; }

    [ProtoMember(4)]
    public IEnumerable<CreateOrderLineRequest> OrderLines { get; set; }
}
```

Figuur 5, Protocontract met sturing datatype.

```
1 reference | Johan Smarius, 3 days ago | 1 author, 1 change
public enum CustomerType
{
    Error = 0,
    Private = 1,
    Corporate = 2
}
```

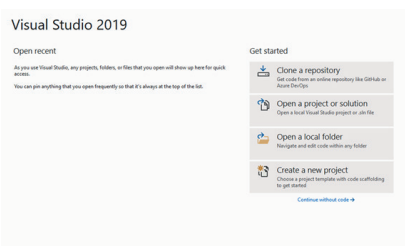
Figuur 6, Enumeratie voor contract.

worden geregeld. Hiervoor gebruik je de parameter DataFormat. Als je deze instelt op DataFormat.WellKnown, dan wordt de mapping van het interne type van en naar het .NET DateTime type automatisch voor je geregeld. In dit contract kun je ook zien dat collecties en enumeraties gewoon ondersteund worden. CustomerType is namelijk een normale C# enum **(Figuur 6)**.

Alle code in dit artikel maakt gebruik van de ProtoContract-variant, omdat ik binnen een codebase graag gebruik maak van dezelfde variant. Door het coderen van je data- en servicecontract ben je klaar met het definiëren van je interface van de service.

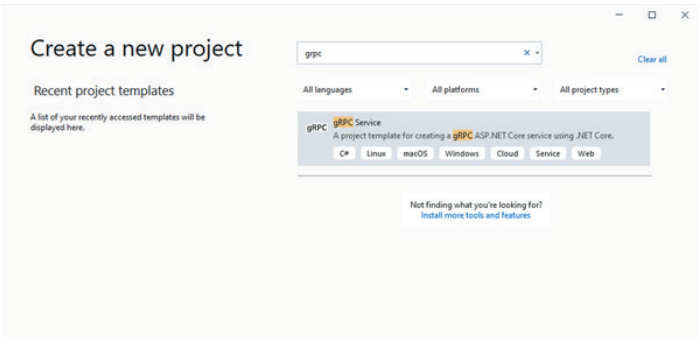
Coderen van de server

Voor het coderen van de server kun je in principe als basis werken met een ASP.NET Core Web Application. De voorbeeldcode maakt echter gebruik van de normale gRPC-template binnen Visual Studio. De normale aanpak via protobuf contracten en de code first aanpak kunnen zonder problemen naast elkaar gebruikt worden en om dit te laten zien, heb ik voor de gRPC-template gekozen. Om je geheugen nog even op te frissen, zal ik de stappen die hiervoor nodig zijn nog even herhalen. Bovendien is een scherm in de template een heel klein beetje veranderd sinds het eerste artikel.

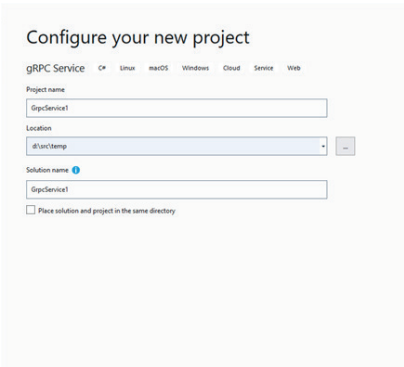


Figuur 7, Startscherm Visual Studio 2019.

In het startscherm kun je kiezen voor "Create a new project" **(Figuur 7)**. In de templatepagina **(Figuur 8)** kun je nu zoeken naar gRPC. Voor gRPC is er op dit moment maar een template beschikbaar. Als je deze template kiest en op "Next" drukt, dan kom je

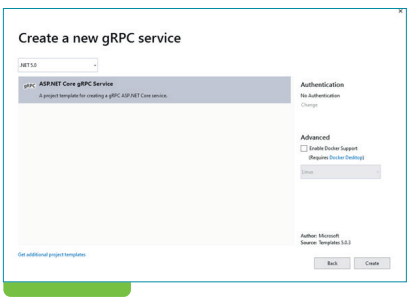


Figuur 8, Create Project Page.

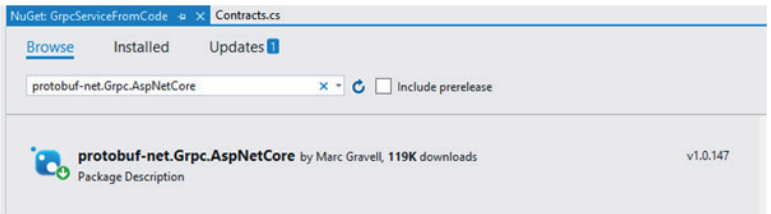


Figuur 9, Projectconfiguratie.

terecht op de pagina die voor alle templates gebruikt wordt om de project naam, locatie en solution name in te stellen **(Figuur 9)**.
Als je nu op "Create" klikt dan kun je nog wat aanvullende opties instellen **(Figuur 10)**.
Het is belangrijk om te controleren dat .NET 5 als framework geselecteerd staat en niet .NET Core 3.1. Als je nu weer op "Create" drukt, dan wordt de code voor je server gegenereerd.

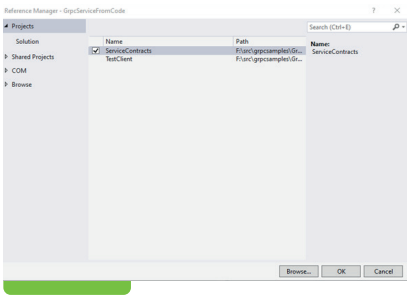


Figuur 10, Service configuratie.



Figuur 11, NuGet package voor server code.

Om de code voor de server te kunnen schrijven, moet je het NuGet-package "protobuf-net.Grpc.AspNetCore" toevoegen **(Figuur 11)**.
De definitie van de interface staat in de Class Library die je aangemaakt hebt, dus daar moet je ook een project referentie naar leggen **(Figuur 12)**.



Figuur 12, Project referentie.

```
public class EchoService : IEchoService
{
    public ValueTask<EchoResponse> EchoMessage(EchoRequest request)
    {
        var response = new EchoResponse()
        {
            Message = $"Echo {request.Message}"
        };

        return new ValueTask<EchoResponse>(response);
    }
}
```

Figuur 13, Implementatie server.

De implementatiecode voor de server kun je het beste plaatsen in de map "Services". Je kunt hiervoor gewoon een nieuw bestand maken van het type "Class".
De hele implementatie van de server bestaat uit het implementeren van de interface **(Figuur 13)**. Voor deze service komt dat vooral neer op het aanmaken van het response bericht.

```
private readonly ILogger<EchoService> _logger;

0 references | 0 changes | 0 authors, 0 changes
public EchoService(ILogger<EchoService> logger)
{
    _logger = logger;
}
```

Figuur 14, Gebruik van dependency injection in server code.

Binnen deze code kun je ook gewoon gebruik maken van dependency injection **(Figuur 14)**.
De server moet natuurlijk nog wel bekend gemaakt worden aan .NET. Hiervoor moeten wat aanpassingen in de Startup.cs worden gedaan. Binnen de ConfigureServices methode moet de ondersteuning voor code first gRPC toegevoegd worden aan de services-collectie. Hiervoor is een aparte methode "AddCodeFirstGrpc" beschikbaar **(Figuur 15)**. De normale gRPC-ondersteuning en de code first variant kunnen gewoon naast elkaar bestaan. In de code zie je dus ook gewoon de normale "AddGrpc" terug die door de standaard Microsoft implementatie gebruikt wordt.
De service zelf moet natuurlijk in de routing opgenomen worden. Net als bij de normale gRPC kun je dit bereiken door het type via de methode MapGrpcService toe te voegen aan

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddGrpc();
    services.AddCodeFirstGrpc();
}
```

Figuur 15, Aanpassingen ConfigureServices.

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapGrpcService<GreeterService>();
    endpoints.MapGrpcService<EchoService>();

    endpoints.MapGet("/", async context =>
    {
        await context.Response.WriteAsync("Communication with gRPC endpoints must be made through a gRPC client.
    });
});
```

Figuur 16, Toevoegen aan routing De code voor de server is hiermee volledig.

```
using var channel = GrpcChannel.ForAddress("https://localhost:5001");
```

Figuur 17, Aanmaken channel.

```
var echoService = channel.CreateGrpcService<IEchoService>();
```

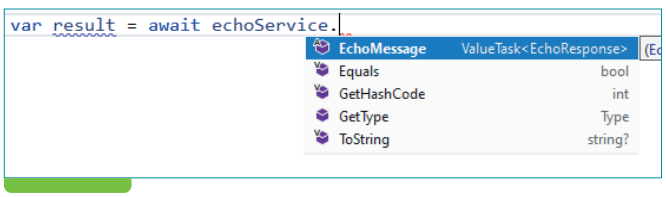
Figuur 18, Aanmaken client.

de endpoints collectie **(Figuur 16)**.

Gebruik maken van een service

Om gebruik te kunnen maken van de service moet je de NuGet-package "protobuf-net.Grpc" toevoegen aan je code en heb je natuurlijk ook een project referentie nodig naar de Class Library waarin de interface staat. Voordat de service aangemaakt kan worden, moet je eerst een channel maken **(Figuur 17)**.

De client kun je nu aanmaken door expliciet de service aan te maken **(Figuur 18)**. Via deze client kun je dan weer de methode aanroepen, die je in het servicecontract op hebt genomen **(Figuur 19)**. Bij het opzetten van het contract was al beschreven dat code first niet automatisch synchrone en asynchrone methoden voor je maakt. In deze afbeelding is dat ook duidelijk te zien.

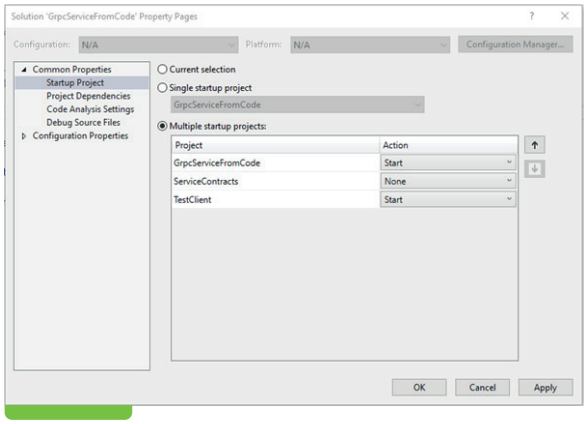


Figuur 19, Beschikbare methoden.

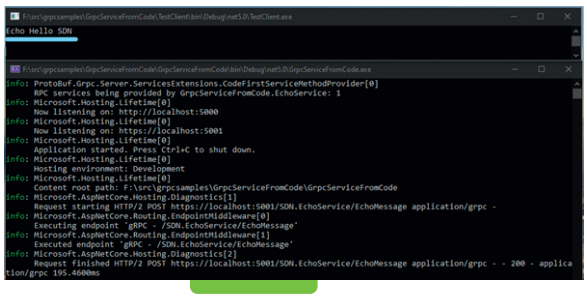
```
var result = await echoService.EchoMessage(new EchoRequest() {Message = "Hello SDN"});

Console.WriteLine(result.Message);
```

Figuur 20, Aanroepen EchoMessage in code.



Figuur 21, Instellen meerdere projecten om te starten.



Figuur 22, Uitvoer van de service.

Voor het aanroepen moet natuurlijk nog wel een EchoRequest aan worden gemaakt **(Figuur 20)**.

Bij het uitvoeren van deze code moet je er natuurlijk wel weer rekening mee houden dat je 2 uitvoerbare projecten hebt. Voor het makkelijk debuggen moet ik dus beide applicaties starten **(Figuur 21)**.

De output van de draaiende applicaties wordt getoond in 2 console windows. In deze consoles kun je zien dat de code inderdaad werkt **(Figuur 22)**. In de logging kun je ook zien dat de naam van de service inderdaad "SDN.EchoService" is zoals opgegeven in de Name-parameter van het ServiceContract **(Figuur 23)**.

Uitgebreidere mogelijkheden

De EchoService is natuurlijk nog redelijk beperkt. Voor het eerste artikel uit deze reeks heb ik een uitgebreider

```
var order = new CreateOrderRequest()
{
    CustomerId = response.Id,
    OrderedAt = DateTime.UtcNow,
    OrderLines = new List<CreateOrderLineRequest>
    {
        new CreateOrderLineRequest()
        {
            ProductId = 100,
            Price = 10,
            Quantity = 2
        },
        new CreateOrderLineRequest()
        {
            ProductId = 200,
            Price = 20,
            Quantity = 3
        }
    }
};
```

Figuur 23, Gebruik initializers binnen code.

voorbeeld gemaakt. Dit voorbeeld heb ik overgezet naar een code first aanpak. Aan de hand van deze code zal ik je nog wat uitgebreidere mogelijkheden laten zien.
In het begin van dit artikel heb ik bij het behandelen van het opzetten van het contract al laten zien dat ook enumeraties en collecties binnen een contract mogelijk zijn. In principe kun je bij het opzetten van het contract complete boomstructuren gebruiken. De enige beperking is dat dit niet mag resulteren in een graaf, omdat er dan cyclische afhankelijkheden ontstaan. Voor WCF-ontwikkelaars is dit geen onbekende beperking.
Bij de code first aanpak kun je gewoon gebruik maken van de .NET collecties die je al gewend bent om te gebruiken. Persoonlijk gebruik ik over het algemeen IEnumerable<T> voor collecties, om de exacte keuze niet in de interface vast te leggen en op die manier wat flexibiliteit te hebben bij de implementatie van de service. Bij het aanmaken van een collectie voor een aanroep, kun je gewoon gebruik maken van object en collection initializers **(Figuur 23)**.
Als je binnen je contracten gebruik wilt maken van void of een lege parameterlijst voor een methode, dan wordt dit gewoon ondersteund. Je hebt hiervoor geen speciaal type nodig om dit aan te geven. Voor een methode die geen resultaat terug hoeft



Johan Smarius

te geven, maar die wel asynchroon moet zijn, kun je gebruik maken van Task of ValueTask **(Figuur 24)**.
Streaming binnen de code first aanpak wordt ook ondersteund door gebruik te maken van het type "IAsyncEnumerable". Je kunt dit type gebruiken als returntype, maar ook als type voor een methodeparameter **(Figuur 25)**.
In de code van de server kun je nu nieuwe returnwaarden aanmaken en via "yield return" teruggeven **(Figuur 26)**.
In principe kan de server eeuwig nieuwe waarden blijven sturen. Om de client een mogelijkheid te geven om aan geven dat de stroom mag stoppen, kun je gebruik maken een CancellationToken. Dit token is opgenomen in de CallContext klasse uit de ProtoBuf.Grpc namespace **(Figuur 27)**.
In de uitwerking van de servicemethode heb ik ervoor gekozen om de code te splitsen in een private methode die alleen weet heeft van een cancellationtoken **(Figuur 28)** en de methode die nodig is vanuit de

```
[OperationContract]
2 references | Johan Smarius, 6 days ago | 1 author, 1 change
ValueTask AddOrder(CreateOrderRequest request);
```

Figuur 24, Retourneren van void async.

```
[OperationContract]
0 references | Johan Smarius, 6 days ago | 1 author, 1 change
IAsyncEnumerable<WeatherForecastStreamElement> GetWeatherForecastStream(CallContext context = default);
```

Figuur 25, Server side streaming.

```
while (!cancellationToken.IsCancellationRequested)
{
    await Task.Delay(500, cancellationToken);

    var forecast = new WeatherForecastStreamElement
    {
        Date = (DateTime.UtcNow.AddDays(index++)),
        Temperature = randomGenerator.Next(-20, 55),
        Summary = Summaries[randomGenerator.Next(Summaries.Length)]
    };

    yield return forecast;
}
```

Figuur 26, Return nieuwe returnwaarden.

```
0 references | Johan Smarius, 6 days ago | 1 author, 1 change
public IAsyncEnumerable<WeatherForecastStreamElement> GetWeatherForecastStream(CallContext context = default)
    => GetWeatherForecastStream(context.CancellationToken);
```

Figuur 27, Implementatie van de interface.

```
private async IAsyncEnumerable<WeatherForecastStreamElement> GetWeatherForecastStream(
    [EnumeratorCancellation]CancellationToken cancellationToken)
{
```

Figuur 28, Cancellationtoken in private implementatiemethode.

```
static async Task Main(string[] args)
{
    var terminateDemo = 0;

    CancellationTokenSource _cancellationTokenSource = new();

    try
    {
        using var channel = GrpcChannel.ForAddress("https://localhost:6001");
        var weatherForecastService = channel.CreateGrpcService<IWeatherForecastStreamingService>();

        await foreach (var weatherData in weatherForecastService.GetWeatherForecastStream(
            context: _cancellationTokenSource.Token).WithCancellation(_cancellationTokenSource.Token))
        {
            Console.WriteLine(
                $"Forecast {terminateDemo} {weatherData.Date.ToLongDateString()} {weatherData.Temperature} {weatherData.Summary}");

            ++terminateDemo;

            if (terminateDemo > 100)
            {
                _cancellationTokenSource.Cancel();
            }
        }
    }
    catch (RpcException ex) when (ex.StatusCode == StatusCode.Cancelled)
    {
        // Save to ignore
    }
}
```

Figuur 29, Client code voor server streaming.

```
public async ValueTask AddOrder(CreateOrderRequest request)
{
    Console.WriteLine($"Received order placed at {request.OrderedAt.ToLongDateString()}");

    foreach (var orderLine in request.OrderLines)
    {
        Console.WriteLine($"Received {orderLine.ProductId}");
    }

    using var channel = GrpcChannel.ForAddress("https://localhost:5003");
    var financialService = channel.CreateGrpcService<IFinancialService>();
    await financialService.AddInvoice(new AddInvoiceRequest
    {
        Amount = request.OrderLines.Sum(line => line.Price * line.Quantity),
        CustomerId = request.CustomerId
    });
}
```

Figuur 30, Aanroepen andere service in servicecode.

interface. Om het cancellationtoken goed te laten werken, moet je bij de parameter het attribuut "EnumeratorCancellation" gebruiken. De compiler geeft hier ook een waarschuwing voor als je dit per ongeluk zou vergeten.
Met deze combinatie van methoden kan de client een context meegeven, maar is dat niet verplicht.
De client code kan gebruik maken van de await foreach constructie om de stroom van nieuwe dataelementen op te vangen. Deze code is opgenomen in een console applicatie dus om het annuleren te simuleren, wordt de foreach na 100 iteraties geforceerd afgebroken **(Figuur 29)**. Bij het stoppen van de async stream wordt wel een RpcException gegooid, maar die kun je veilig negeren. Door het gebruik van een filter, kun je dit negeren ook heel specifiek doen.

Op dit moment ondersteunt protobuf-net.Grpc nog niet de integratie met het DI-framework van ASP.NET Core via de GrpcClientFactory. Als je dus gebruik wilt maken van een andere service binnen je code van een service, dan moet je expliciet de client aanmaken **(Figuur 30)**.
Je kunt deze code natuurlijk wel netter maken door zelf met factory patterns te gaan werken. Het gebrek aan ondersteuning voor de GrpcClientFactory heeft wel tot gevolg dat deze variant niet goed samen kan werken met de gRPC-Web toevoeging aan gRPC.NET. Dit heeft natuurlijk ook gevolgen voor de flexibiliteit die je op dit moment hebt in de keuze van je hosting opties.

Samenvatting

In dit artikel heb ik de belangrijkste aspecten voor het bouwen van een gRPC service op basis van een code first aanpak behandeld. Met deze variant kun je gemakkelijk een contract in je code schrijven. Dit contract kan dan gedeeld worden tussen de server en de client. Het ontwikkelteam hoeft op deze manier veel minder kennis te hebben van protobuf en hoeft bovendien veel minder rekening te houden met een aantal specifieke typen, met een nogal aparte syntax, die bij het gebruik van de standaard protobuf aanpak helaas nodig is. Toch gaat mijn eigen voorkeur uit naar de protobuf aanpak. Vooral door het gebrek aan ondersteuning van GrpcClientFactory en daarmee dus ook gRPC-Web zal ik in de praktijk maar in heel specifieke gevallen voor deze code first aanpak kiezen. Voor ontwikkelteams die komen vanuit WCF kan deze optie wel heel handig en laagdrempelig zijn.
Op mijn github (https://github.com/JohanSmarius) heb ik een aantal gRPC demo projecten staan. De code uit dit artikel staan in de repo's: "GrpcDemoCodeFirst" (port van de gRPC services uit het eerste artikel) en "GrpcServiceFromCode" (EchoService). ●


I have the freedom to experiment with my own ideas
Discover IT at rabobank.jobs
Rabobank
Growing a better world together.

Een introductie van Structured Logging

Je gooit een app in productie en dat is het dan … Toch? Vroeger was het vaak dat je eigenlijk geen zicht had. Je applicatie ging als het ware een roadtrip maken waar je niet direct bij kon zijn. Als er dan iets mis was dan hoorde je dat vaak achteraf en indirect. Via gebruikers bijvoorbeeld. Vervolgens kon je de logs achterhalen door ze op te gaan vragen bij een beheerder om vervolgens te gaan ploeteren door een redelijk grote hoeveelheid tekst, maar haalde je er dan ook echt uit wat je nodig had?

Auteur: Vincent Hendriks

Als er 1 ding is wat tegenwoordig in vrijwel iedere applicatie aanwezig is dan is het wel logging. Als er iets misgaat is dit ook vaak 1 van de eerste dingen waar we naar kijken.

Waarom loggen we eigenlijk?

Voordat we verder gaan, waarom loggen we eigenlijk? Hoe ik er in ieder geval tegenaan kijk is dat we vaak ons best doen om een applicatie zo robuust mogelijk te ontwikkelen, maar er toch altijd iets mis zal gaan. Als dit gebeurt willen we eigenlijk altijd de logs erbij kunnen pakken om te gaan kijken wat er nu precies gebeurt, toch? Vragen die je jezelf zult stellen zijn o.a.:

- Wat was nou precies de foutmelding?
- Wat gebeurde er voordat dit probleem optrad?
- Hadden meerdere gebruikers hier last van?

In dit geval gebruiken we de logs als een diagnostische tool. Een andere reden om logging toe te passen is bijvoorbeeld t.b.v. monitoring/analytics:

- Hoeveel requests per seconde zijn er nu?
- Wat is de gemiddelde responsetijd?
- Hoeveel orders per minuut zijn er nu?

Vervolgens kunnen we deze logging weer gebruiken om alerts af te laten gaan als er bijvoorbeeld enkele van die bovenstaande waardes boven/onder een bepaalde grens komen. Wat de reden ook is, je zult vast ook wel weten hoe lastig dat soms kan zijn als je dit allemaal uit pure tekst zou moeten halen. Dit is waar gestructureerde logging van pas komt.

Gestructureerde logging

Gestructureerde logging (soms ook semantische logging genoemd) doet eigenlijk precies wat de naam al zegt. Het zorgt ervoor dat je niet alleen een platte tekst logt, maar gestructureerde data.
In dit artikel maak ik qua voorbeelden gebruik van Serilog i.c.m. Microsoft's abstractie laag t.b.v. logging, maar dit kan natuurlijk ook op andere manieren ofwel talen/ontwikkelomgevingen.
Laten we eerst eens kijken naar een simpel voorbeeld, hoe het niet moet, waarbij er een foutmelding wordt gelogd:

```
_logger.LogError(exception,
$"An error occurred trying
to save order: {order.Id}")
```

Wat voor data zien wij hier als we dit naar een eenvoudig log bestand zouden wegschrijven?
Typisch zal dit dan het tijdstip, de log level, de foutmelding met stacktrace zijn en een tekst regel met daarin een order id verwerkt. Als we in hetzelfde log bestand zouden gaan zoeken naar een specifieke order id dan is er een grote kans dat we dit gaan vinden, maar wat nou als we iets anders hadden dan het order id? Bijvoorbeeld een klant id of nog beter een error id die we als response melding hebben gegeven.
We kunnen natuurlijk deze ontbrekende informatie toevoegen aan deze log actie, maar daar hebben we op dat moment niks aan.
We weten nou eenmaal niet altijd wat we precies nodig gaan hebben totdat we ergens tegen aan lopen en het blijkt dat we iets missen.
Overigens verdwijnt dit probleem natuurlijk nooit helemaal, maar we kunnen het wel zo veel mogelijk proberen te voorkomen. Los daarvan denk ik ook dat niemand heel erg vrolijk wordt op het moment dat we ellenlange log acties in onze code gaan schrijven alleen maar omdat we allerlei extra data willen gaan loggen.
Een ander probleem wat we hier zien is dat als we de data willen verwerken/doorzoeken dat we de tekst moeten gaan parsen.
Om deze 2 punten te gaan verbeteren gaan we als eerste deze log actie een klein beetje herschrijven naar:

```
_logger.LogError(exception,
"An error occurred trying
to save order: {orderId}",
order.Id)
```

Er zijn nu een aantal dingen die veranderd zijn:

> Op het moment dat de configuratie zegt dat we alleen critical logging willen wegschrijven dan zal er geen string concatenatie plaats vinden, omdat deze log actie een lagere level heeft.

> Een log provider kan de eventuele parameters apart verwerken. In dit geval zou de orderId dus, naast de tekst weergave van de logregel, apart opgeslagen kunnen worden.

Wanneer we bijvoorbeeld Serilog (https://serilog.net/) gebruiken dan geeft Serilog alle parameters door naar de onderliggende "log sinks"/ log providers. Een "sink" bepaald hoe en waarheen een log item weggeschreven gaat worden. Je kunt hier denken aan de Console, Files, SQL Server, etc.
Dit lost, afhankelijk van de gekozen sink, in ieder geval het probleem op van het tekst parsen. We kunnen alleen nog een stapje verder:

```
_logger.LogError(exception,
"An error occurred trying
to save order: {@order}",
order)
```

Er is eigenlijk, qua code, maar 1 klein verschil: In plaats van een order id hebben we het gehele order object meegegeven.
Normaliter zou dit dan d.m.v. een ToString call omgezet worden naar een string wat standaard een type naam zal opleveren, maar door een @ toe te voegen geven we richting Serilog aan dat we dit object als een gestructureerd object willen opslaan.
Zodra je dit dan bijvoorbeeld in de Console/ File logging voorbij ziet komen krijg je typisch de JSON-representatie van de order te zien.
Wat er nu alleen nog ontbreekt is dat je niet eenvoudig door je logs heen kunt zoeken. Dit is waar een "Structured Log Server" van pas komt.

Scoping

We weten nu dat we hele objecten kunnen loggen, maar stel je zou bijvoorbeeld alle logs willen zien gedurende het uitvoeren van 1 specifieke API call? In een traditionele aanpak zou je dan voor iedere log regel een bepaald kenmerk moeten wegschrijven zodat je dit kunt volgen, maar met een moderne aanpak zoals we hier beschrijven kan dit d.m.v. "log scoping". Bij het aanmaken van een nieuwe log scope voeg je data toe aan alle log acties die binnen deze scope worden uitgevoerd.
ASP.NET maakt automatisch al een nieuwe scope voor ieder request en voegt hier o.a. een unieke request id aan toe. Zou je van 1 foutmelding willen weten wat er nog meer gedurende dezelfde request is gebeurd dan hoef je alleen maar te filteren op de request id die bij deze specifieke foutmelding aanwezig is en je krijgt alle log regels te zien die gelogd zijn gedurende de afhandeling van deze request.
Los van het feit dat ASP.NET Core zelf scopes aanmaakt, kun je dit ook zelf doen. Als voorbeeld zou je een taak kunnen hebben draaien binnen je applicatie en een nieuwe scope voor iedere taak run kunnen aanmaken.
Een voorbeeld hoe je zelf een scope aanmaakt:

```
using (_logger.BeginSco-
pe("Running task {task}.
Task Run Id: {taskRunId}",
typeof(TTask), taskRunId))
{
    ...
}
```

In bovenstaand voorbeeld zou dit betekenen dat je op basis van een taskRunId alle bijbehorende log regels kan opzoeken.

Gevoelige informatie

Hopelijk ben je tot nu toe enthousiast, maar voordat je overal volledige objecten gaat loggen moet je wel even bewust worden van een risico wat hierbij komt kijken: **Let goed op wat je naar je logs wegschrijft!**
Het is al heel wat keren voorgekomen dat een bedrijf in het nieuws komt, omdat er bijvoorbeeld wachtwoorden zijn gelekt via logbestanden.
Uiteraard wil je geen wachtwoorden en/of gevoelige informatie wegschrijven! Eigenlijk is dit risico altijd al wel aanwezig geweest, maar met meer data betekent dat dit risico alleen maar groter wordt
Gelukkig zijn er wel een aantal dingen die je kunt doen om dit te voorkomen. Je kunt simpelweg ervoor zorgen dat deze objecten met gevoelige informatie niet gelogd worden of je gaat deze data maskeren/wegpoetsen voordat ze weggeschreven worden.
Als voorbeeld hoe je dit zou kunnen doen kijken we naar hoe een open source library (https://github.

BIO

Vincent Hendriks

Vincent Hendriks is een Software Engineer met meer dan 15 jaar ervaring en een enorme passie voor het vak. Vincent heeft een zeer brede ervaring in onderwerpen zoals C#, .NET/.NET Core, Software achitectuur, CQRS, Event Sourcing, SQL, Azure, Docker en nog veel meer. LinkedIn: https://www.linkedin.com/in/vhendriks



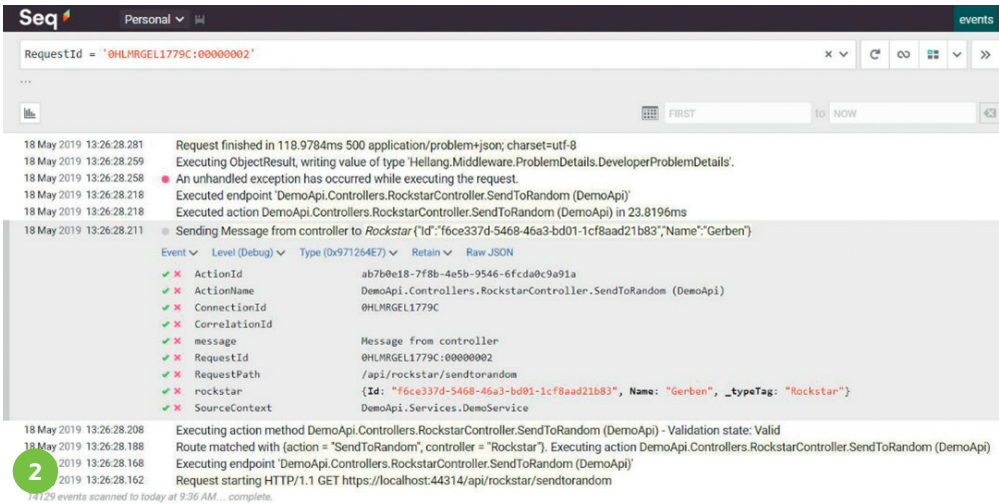
com/destructurama/attributed) dit doet d.m.v. attributes die je boven je properties/ fields kunt plaatsen:

```
public Guid Id { get; }

[NotLogged]
public string Name { get; }

[LogMasked(ShowFirst = 3)]
public string SocialSecurity
Number { get; }
```

Zodra dit object vervolgens gelogd wordt dan zal er geen "Name" aanwezig zijn en zal er maar een gedeelte van het "SocialSecurityNumber" zichtbaar zijn. **Voorbeeld 1.**



Log sinks

Wanneer je gebruik maakt van een logging library die gestructureerde logging ondersteund, maar dit dan wegschrijft naar een formaat waar deze data als platte tekst opgeslagen wordt dan maak je nog niet optimaal gebruik van gestructureerde logging. Hiervoor kun je beter gebruik maken van een opslag plek die deze gestructureerde logging wel volledig opslaan en het mogelijk maakt om door de logs heen te zoeken, te filteren en te visualiseren. Ook zijn er services die vervolgens ook alerts kunnen genereren aan de hand van logs.
Er zijn diverse cloud oplossingen, maar ook oplossingen die je zelf kunt hosten. Welke je gebruikt ligt vooral aan eigen voorkeuren, het bedrijfsbeleid en/ of eventuele wetgevingen waar je je aan moet houden. Enkele voorbeelden zijn:

Seq

Seq is the intelligent search, analysis, and alerting server built specifically for modern structured log data. Seq is dus een logging server waar logging naar toe gestuurd kan worden en tegelijkertijd zeer makkelijk doorheen gezocht kan worden. Perfect voor development purposes, maar ook zeker geschikt voor productiedoeleinden. Seq biedt in ieder geval naast het doorzoeken van logs ook mogelijk om dashboards en alerts aan te maken. Ingestion gaat op basis van een HTTP endpoint, maar er zijn ook tools die file based logs kunnen pushen naar seq. Seq moet je overigens wel zelf beheren. **Voorbeeld 2.**

Azure Monitor

Collect, analyze, and act on telemetry data from your Azure and on-premises environments. Azure Monitor helps you maximize performance and availability of your applications and proactively identify problems in seconds.
Waar je voorheen Application Insights en Azure Log Analytics had zijn deze tegenwoordig ondergebracht bij Azure Monitor. Azure Monitor is een volledige managed omgeving. Bied in ieder geval logging (log analytics), dashboards, alerts, metrics en Application Insights wat allerlei dependencies, metrics, etc inzichtelijk maakt binnen een applicatie. **Voorbeeld 3.**

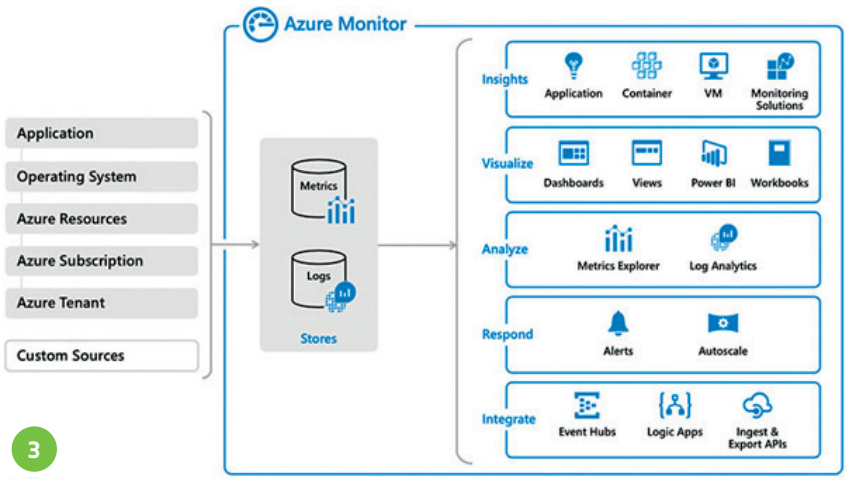
Elastic Stack

That's Elasticsearch, Kibana, Beats, and Logstash (also known as the ELK Stack). Reliably and securely take data from any source, in any format, then search, analyze, and visualize it in real time. De "Elastic Stack" (voorheen ELK stack) is niet zozeer 1 product, maar meerdere in 1:

> Elasticsearch
> Logstash
> Kibana
> Beats
> Logstash

Elasticsearch: Elasticsearch is een gedistribueerd, RESTful search en analytics engine. Hier worden de logs dus in opgeslagen en bied het andere tools de mogelijkheid om hier doorheen te zoeken.
Logstash is het beste te vergelijken met een dataprocessing pipeline. Het pakt verschillende bronnen en stuurt dit door naar iets anders zoals elasticsearch.
Kibana is de UI schil die het mogelijk maakt om met de data in elastic-

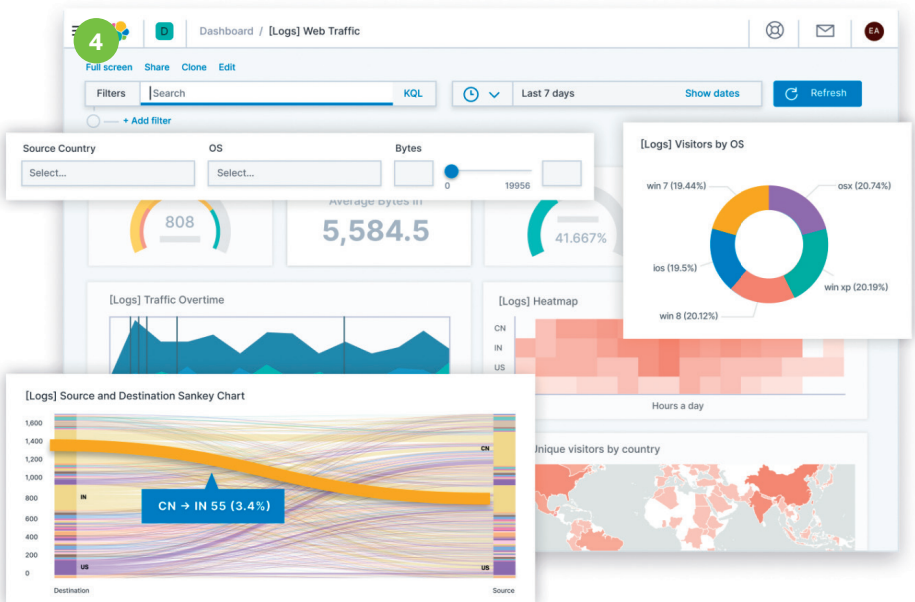




search te kunnen werken.
Logstash is een logging collectie pipeline waarbij logs gefiltered en verstuurd kunnen worden naar andere elasticsearch of andere services.
Beats is bedoeld om logs vanuit honderden of duizenden lichtgewicht machines naar logstash of elasticsearch te sturen.
De elastic stack kan zelf gehost worden of als managed product afgenomen worden bij verschillende partijen.Het is een zeer populair platform met enorm veel mogelijkheden. Denk aan logging, dashboards, alerts en machine learning.
Voorbeeld 4.

Console

Als developers zullen we al best veel logging gezien hebben in onze console via "stdout". Wat nu als ik je vertel dat dit ook gewoon in productie wordt gebruikt? Wanneer je je applicatie in bijvoorbeeld een kubernetes cluster draait wel in ieder geval! Wanneer je veel verschillende applicaties moet beheren dan wil je niet per applicatie bezig zijn met zaken zoals waar de logging heen moet. Dat is iets wat je over kunt laten aan bijvoorbeeld de beheerder van je cluster. Een populaire strategie is dan te loggen naar de console, waarna een andere applicatie deze console output uitleest, parsed en pushed naar een service waar de logs opgeslagen kunnen worden.
Twee populaire tools hiervoor zijn FluentD en FluentBit. Ze "collecteren" logs van verschillende bronnen (file, http, etc), filteren en transformeren dit (indien nodig) en zorgen dat de logs weer doorgestuurd worden naar andere services zoals een Elasticsearch, Azure Monitor, Seq, etc.
Voor wie hier nu Azure Monitor gebruikt op een linux omgeving: Je gebruikt FluentD al zonder dat je dit wellicht al wist.
Om bijvoorbeeld in een basic kubernetes cluster alle logging op te vangen en door te sturen kun je een deamonset definiëren (Een deamonset is iets wat op alle nodes moet draaien) met daarin FluentBit die vervolgens alle console output van alle applicaties in het cluster forward naar bijvoorbeeld Elasticsearch.
Gebruik je AKS (Azure Kubernetes Service) dan is een koppeling met Azure Monitor zeer eenvoudig al out-of-the-box mogelijk zonder dat je hier zelf iets voor hoeft te doen.



Conclusie

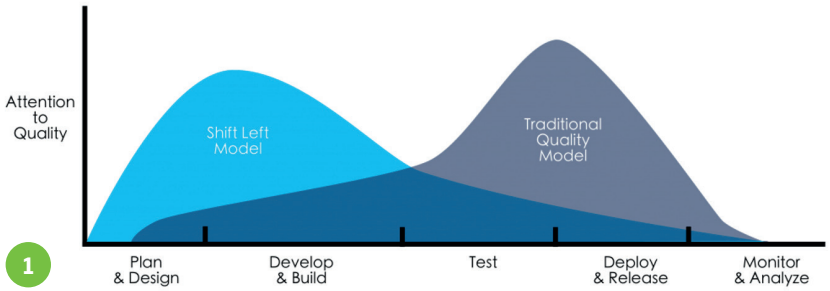
Hopelijk is het nu een beetje duidelijk wat gestructureerd loggen nou precies is en hoe je hier je voordeel uit kunt halen. Ook heb ik enkele services/tools genoemd die je hiervoor kunt gebruiken. Gebruik je nu nog helemaal geen structured logging, maar zou je dit wel willen? Begin dan bij development ... Zorg dat je bijvoorbeeld Seq lokaal draait in een docker container. De toegevoegde waarde van structured logging zal snel gezien gaan worden! ●

Vind je dit verhaal interessant en heb je nog wat vragen? Neem gerust contact met mij op via LinkedIn: https://www.linkedin.com/in/vhendriks



Threat Modeling

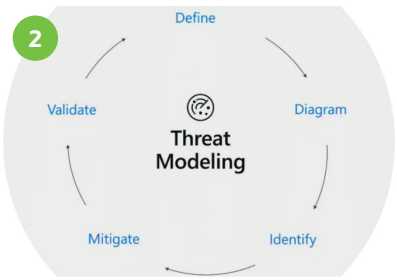
In het huidige software ontwikkelingslandschap is het steeds sneller opleveren van software een belangrijk. Organisaties moeten mee in de DevOps beweging om hun concurrentiepositie te behouden of te verbeteren. Om dit goed en veilig te kunnen doen is het van belang om zo vroeg mogelijk in het ontwikkel-proces al feedback te verzamelen, om snel te kunnen bijsturen. Dit vatten we vaak samen in de zogenaamde Shift Left beweging. Shift Left is een verzameling van kwaliteitsborging methodieken zoals o.a. unittesting, performancetesting & componenttesting.



Auteur: Patrick Bes

Naast deze belangrijke testfeedback is ook op het gebied van security een van belang dat men zo vroeg mogelijk een helder beeld heeft van de potentiele security risico's. De wolf in schaapskleren moet al vroeg worden geïdentificeerd. Hiervoor zijn er al tientallen geleden jaren methodieken bedacht, welke helaas vaak over het hoofd worden gezien. DevOps is een manier om een kortere "*time to market*" te bewerkstelligen, maar daarnaast ook een lager faalpercentage bij opleveringen. Het beperken van security issues in productie is hier natuurlijk ook onderdeel van. **Threat Modeling** is bij uitstek een methodiek om bij dit laatste te helpen **(Figuur 1)**.
Deze methodiek bestaat uit 5 stappen die zich herhalen binnen de Software Applicatie LifeCycle. Het herhalende karakter past ook prima in het herhalende karakter van DevOps,. Of bij het gebruik van deze methodiek wellicht wel DevSecOps, omdat security nu een prominentere rol krijgt. **(Figuur 2)** De stappen die men kan ondernemen zijn:



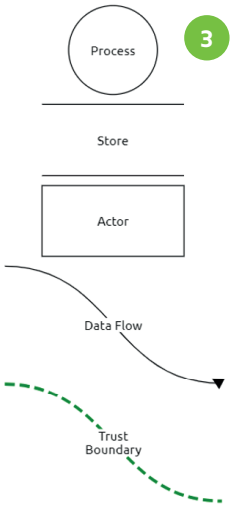


Define: Vastleggen van de securityeisen.

Niet in elke app zullen de security-eisen gelijk zijn. Indien er met persoonlijke data wordt gewerkt zullen er vast eisen zijn vanuit wetgeving c.q. beleid van uit de organisatie zelf. Zorg ervoor dat deze duidelijk zijn.

Diagram: Het maken van een applicatiediagram

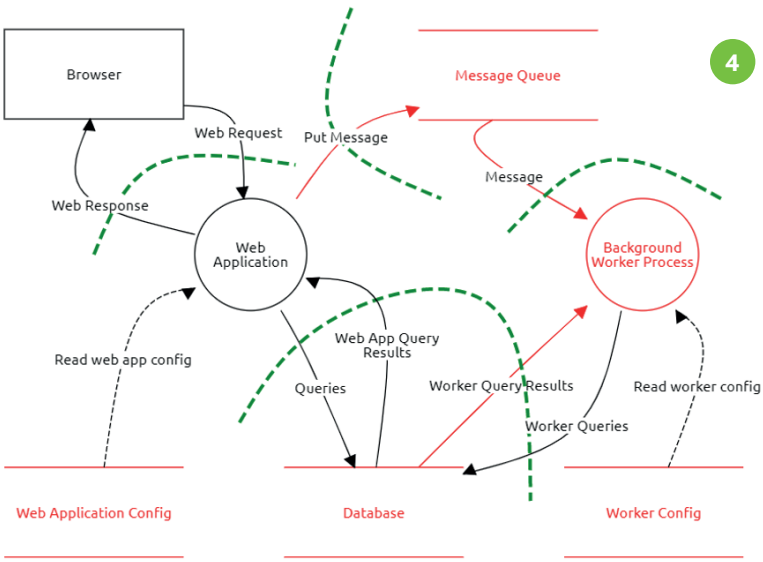
Om inzicht te krijgen in de potentiële dreigingen is het makkelijk om de flow van de betreffende applicatie te modelleren. Dit kan men doen in



specifieke Threat modeling tooling zoals bijvoorbeeld Thread Dragon. https://threatdragon.org/login
Hierin kan men de applicatie weergeven via een eenvoudige Data Flow Diagram bestaande uit Processen, Stores, Actors en/of DataFlows zoals hier weergegeven **(Figuur 3)**.

Identity: Het identificeren van de potentiële bedreigingen

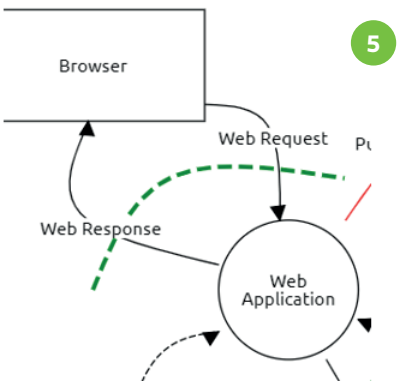
Na het opstellen van het diagram kan men met Trust boundaries koppelvlakken aangeven waar men risico's herkend. Het model kan na deze stap er als volgt uit zien (De groene stippellijnen representeren de Trust Boundaries): **(Figuur 4)**



Mitigate: Het mitigeren van de bedreigingen

In 1999 hebben Praerit Garg and Loren Kohnfelder bij Microsoft het STRIDE Model ontwikkeld. Met dit model kan men per trust boundary de specifieke kwetsbaarheden classificeren. STRIDE is een acroniem voor:

Threat	Schending	Definitie	Voorbeeld
Spoofing identity	Authenticatie	Het verkrijgen van toegang van iemand anders zijn authenticatie informatie	Zich voordoen als Clown Bassie.
Tampering with data	Integriteit	Het ongeoorloofd veranderen van data	Rekeningnummer aanpassen in data.
Repudiation	Niet-afwijzend	Het ontkennen dat men een actie heeft uitgevoerd	"Ik heb die mail niet verzonden"
Information disclosure	Vertrouwelijkheid	Het verkrijgen van data welke niet voor de betreffende persoon is bedoeld	Inzicht in belastingaangifte van de buren.
Denial of service	Beschikbaarheid	Een risico voor de beschikbaarheid en betrouwbaarheid van het systeem	Een crash loop van de website waardoor hij niet beschikbaar is.
Elevation of privilege	Autorisatie	Het toegang krijgen tot functionaliteit zonder de juiste privileges	Mogelijkheid tot uitvoeren van scripts zonder de juiste rechten.



5

Men kan nu per trust boundary inventariseren welke onderdelen van STRIDE hier een bedreiging vormen. In het eerdere voorbeeld kunnen wij aangeven dat Spoofing Identity tussen de browser en webapplicatie een potentieel risico is. Dit risico kan in dit geval bijvoorbeeld gemitigeerd zijn door het gebruik van een signed authenticatie token. Zo kan men alle trust boundaries langs lopen en aan de hand van STRIDE de risico's inventariseren en bepalen of deze adequaat zijn gemitigeerd **(Figuur 5)**.

Validate: Het valideren of alle bedreigingen zijn gemitigeerd;

Na het mitigeren van de potentiële bedreigingen is het nog steeds van belang dat men valideert of er toch geen security leaks of kwetsbaarheden in de applicatie zijn geslopen. Het hebben van geautomatiseerde security scans, dependency checks in de pipeline en real time is nog steeds van belang. Daarnaast kunnen er, nu inzichtelijk is waar de potentiele grootste risico's zitten, abuse cases worden opgesteld. Op basis hiervan kunnen weer specifieke test scenario's worden bedacht om de gemitigeerde bedreigingen te valideren.



Patrick Bes

Tot Slot

Het kan goed zijn om deze modellering techniek in je DevOps proces op te nemen. Hierdoor kunnen potentiële security leaks al vroegtijdig worden geïdentificeerd en kan hierop worden geacteerd. Het neemt initieel niet veel extra tijd in beslag en kan veel opleveren. Het vergroot in ieder geval de security awareness bij alle betrokkenen en dat is altijd winst. ●

Patrick Bes, Bergler Competence Center. Patrick helpt onze klanten als lead developer en consultant bij vragen op het gebied van .NET Core and Full framework, DevOps, Cloud transition, Microservice transition, DDD, Kubernetes, Azure DevOps, Azure platform, Identity & Access management en Shift Left.

CreateTogether.cs ICustomerServices.cs IBetabitServices.cs BetabitServices.cs Extraordi

- advertorial -

CreateTogether.cs

```
using Microsoft.Azure.Functions.Worker;
using Microsoft.Azure.Functions.Worker.Http;

namespace Betabit.Customer.Relationships
{
    public class CreateTogether
    {
        private readonly IBetabitServices _betabitServices;
        private readonly ICustomerServices _customerServices;

        public CreateTogether(ICustomerServices customerServices, IBetabitServices
        {
            _customerServ          tomerServices;
            _betabitServ           itServices;
        }

        [Create
        Ru                    (AuthorizationLevel.Anonymous, "get",
                x

                          ervices.BetabitCreate();
                          omerServices.CustomerCreate();

                          teExtraordinaryTogether(betabitPart, customerPa

            return ex          inary;
        }

        #region          the ma          pens ···
    }
}
```

Join us at betabit.nl

bijzondere code
componeer je samen!

FEATURING

In jouw regio
Nieuwste technologieën
Maatwerk software
Uitdagende opdrachten
Opleiding volledig op maat



betabit {...}

samen bijzonder maken;

Nablik Future Tech Digital 2021

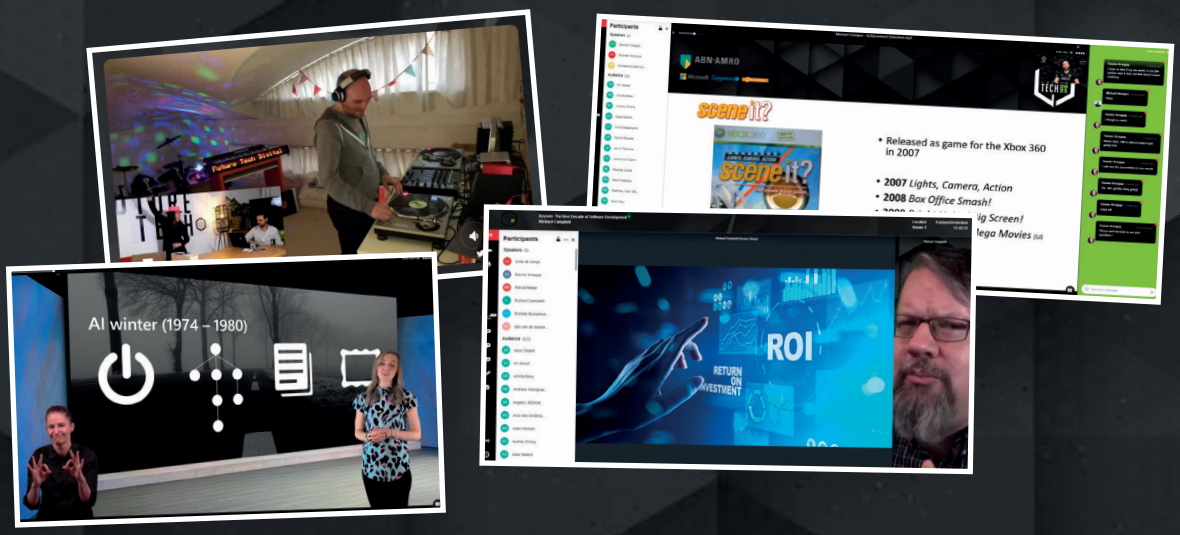
FUTURE TECH Masterclasses Live

Omdat het Corona virus langer aan bleef houden dan we hadden gehoopt, hebben we ook in 2021 Future Tech 'de Developers conferentie voor Microsoft Technologies & .NET' weer digitaal gehouden. En het was weer een succesvolle digitale editie!

2 dagen lang hebben we bezoekers tussen 15:30 – 19:00 deelnemen aan verschillende interactieve sessies van sprekers over de hele wereld.

Met 600 bezoekers waren de sessies van onder andere Michiel Hamers en Roelant Dieben (Ordina), Amy Boyd en Henk Boelman (Microsoft), Arin Roy en Donald Hessing (Capgemini) en Michael R Bentley (ABN AMRO) een groot succes!

Hierbij een kleine terugblik.



Sponsored by:

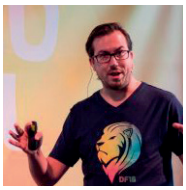
ABN·AMRO

Microsoft Capgemini ORDINA




ORDINA

Ordina:
As a .NET software developer we are constantly broadening our horizon with the exciting new features that are added to the .NET framework and its extended family. Sometimes new techniques come along that can potentially change everything. The Microsoft Power Platform could be such a game-changer, but should we fear this development, or should we embrace this as a .NET software developer? During this session we explored everything you need to know as a .NET software developer how you can leverage this low-code platform while still writing C# code.




Microsoft

Microsoft:
After a tremendous age of innovation and research in the AI field, we are moving towards the age of appliance. But how and where do you get started as a developer and how do you apply machine learning ethically and use it for good.
In this talk we dived into the current state of everyday AI, debated the challanges that come with responsible AI and you got some tips and tricks how you can advantage of AI as a developer.




Capgemini

Capgemini:
GraphQL is a new API standard that provides a more efficient, powerful and flexible alternative to REST. GraphQL enables declarative data fetching where a client can specify exactly what data it needs from an API. Instead of multiple endpoints that return fixed data structures, a GraphQL server only exposes a single endpoint and response with precisely the data a client asked for. The single endpoint can forward the queries to downstreama APIs, merge the data and return it to the client in a single response. This makes GraphQL a great solution in microservices architecture in which the number of endpoints for clients to render the page can become a rigid approach.



ABN·AMRO

ABN-AMRO:
Over "The versatility of azure native logging & monitoring". De oplossingen, voornamelijk gebaseerd op Azure Monitor, die ABN AMRO heeft aangenomen voor het monitoren van applicaties die worden gehost op Azure en on-premises. Onderwerpen zijn onder meer Systeem, End-to-End en Security monitoring, evenals Netwerk monitoring en regelgevende Availability Reporting.

BERGLER

SOFTWARE SOLUTIONS



KENNIS IS MACHT, KENNIS DELEN IS KRACHT!

BERGLER SOFTWARE SOLUTIONS IS OP ZOEK NAAR ERVAREN

LEAD .NET DEVELOPERS.

In de rol van Technical Lead Consultant ben je een meewerkend voorman op het gebied van software development, -kwaliteit, security, Agile processen, SOLID, DevOps, etc.

Samen met je collega's, ben je altijd bezig jezelf verder te ontwikkelen op het gebied van software development binnen de **MICROSOFT STACK**.

Wij geven je 10%-tijd om je in te zetten binnen ons Competence Center (www.bergler.nl/competence-center/).

Kijk ook eens op https://connect.bergler.nl/ om je toekomstige collega's te leren kennen.

BEN JE GEÏNTERESSEERD IN DEZE UITDAGENDE EN AFWISSELENDE FUNCTIE?
Neem dan contact op met Ronald Jansen (HR Manager Bergler Software Solutions):
E: rjansen@bergler.nl. Bellen mag natuurlijk ook: 076-5720200

www.bergler.nl

May the source be with you!

orbi

A MASTERPIECE OF CONNECTIVITY

Orbi® WiFi 6 is perfecte WiFi voor het hele huis. Snelheid. Design. Gemak. Orbi is speciaal ontworpen voor razendsnelle, superieure WiFi in de meest geavanceerde slimme woningen.

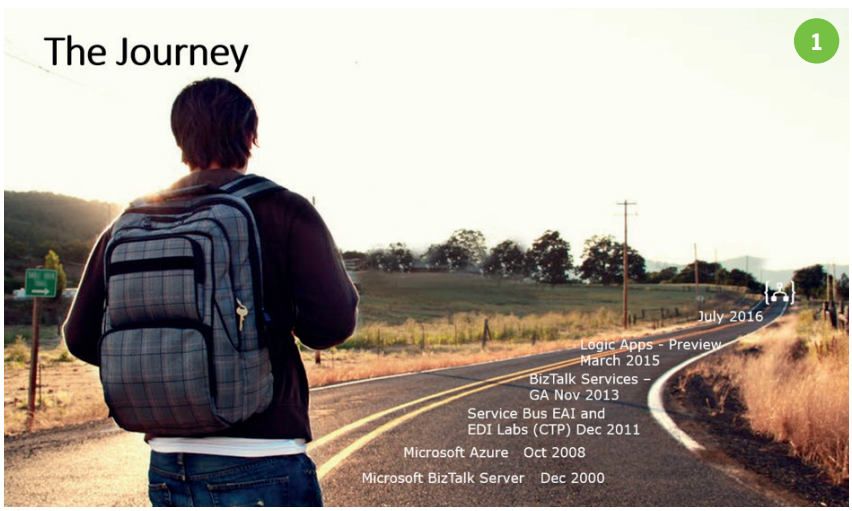


NETGEAR® | Upgrade naar topklasse WiFi via **NETGEAR.com/NL/BestWiFi**

Realiseren van Integratie oplossingen met Azure Logic Apps

Met de opkomst van de Cloud kwam ook de toename in Software-as-a-Service (SaaS) oplossingen zoals Salesforce, Dynamics 365, DropBox, ZenDesk, ServiceNow en Workday. Deze oplossingen stellen organisaties in staat hun bedrijfsprocessen rondom financiën, human-resources, logistiek, en vele andere te borgen. Kortom, het verschuiven van dergelijke processen ondersteund in on-premise systemen naar oplossingen in de Cloud. Het verschuiven wordt ook wel 'ver-Saas-en' genoemd. Echter, kan het ook zijn dat sommige systemen in nieuwbouw oplossingen met platform diensten kan worden gerealiseerd of middels 'Lift-and-Shift' naar de Cloud worden gebracht.

Auteur: Steef-Jan Wiggers



Los van waar bedrijfsprocessen zich afspelen wisselen systemen informatie met elkaar uit. Waarbij in het verleden en zelf vandaag de dag een server product als BizTalk kan worden gebruikt of sinds een aantal jaar Logic Apps in Azure. Met Logic Apps en BizTalk Server kunnen organisaties indien zij een 'best-of-breed' landschap hebben voor hun bedrijfsprocessen deze goed met elkaar laten aansluiten. Dit was en is de kracht van BizTalk Server en eveneens Logic Apps – echter zijn de technologieën niet hetzelfde.

De oorsprong van Logic Apps

Bijna vijf jaar geleden werd Logic App publiekelijk beschikbaar gesteld op het Azure platform en sindsdien is het in twee jaar tijd uitgegroeid tot een leider in de integratie Platform as a Service (iPaaS) Magic Quadrant van Gartner. En vandaag de dag nog steeds.
De publieke Cloud providers eerste poging om een dergelijke dienst in de Cloud te krijgen had de toepasselijk naam BizTalk Services. Deze naam werd na enkele updates van de dienst veranderd in Microsoft Azure BizTalk Services, afgekort MABS wat je ook als een tweede versie van BizTalk Services kunt beschouwen. Echter, bleek deze tweede versie niet afdoende en besloot het bedrijf om de EDI/B2B functionaliteit te verhuizen naar de derde iteratie genaamd Logic Apps. En nu in 2021 is de vierde versie in publieke preview beschikbaar **(Figuur 1)**.

Workflow definitie

Wat zijn Logic Apps nou eigenlijk en hoe verhoudt het zich tot zijn evenknie BizTalk Server. Welnu ontwikkelaars kunnen een Logic App bouwen, d.w.z. een workflow ontwerpen met behulp van de 'designer' in de browser of Visual Studio - gebruikmakend van de Azure Logic Apps Visual Studio Tools (https://marketplace.visualstudio.com/items?item-

BIO

Steef-Jan Wiggers

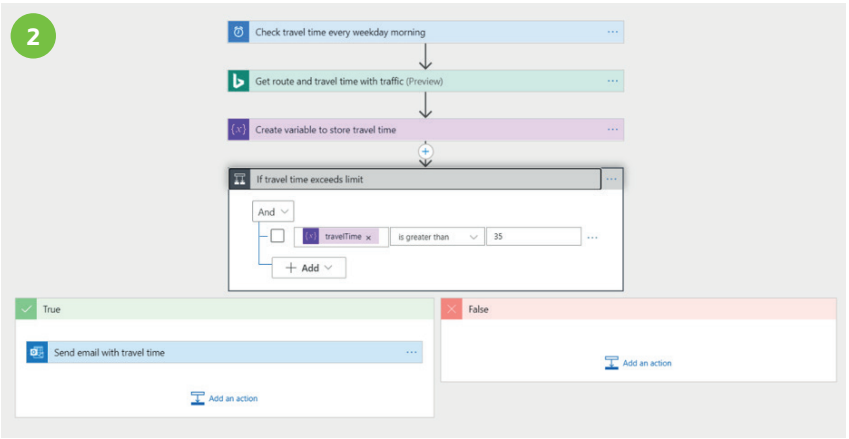
is werkzaam als Technical Integration Architect bij HSO. Hij is zeer actief binnen de Azure en Integratie community, internationaal spreker en auteur van diverse White papers en (e)Books. Verder is hij Cloud journalist bij InfoQ en schrijft hij nieuwsartikelen rondom Azure, AWS en GCP.



Name=VinaySinghMSFT.AzureLogicAppsToolsForVS2019). Bovendien kun je een Logic app zelf zien als een logische container of host van een werkstroom definitie (workflow) - wanneer deze wordt uitgevoerd, worden resources (lees CPU cycles en dergelijke) op het Azure-platform verbruikt. De Logic App zelf is een Platform as a Service (PaaS) dienst in Azure **(Figuur 2)**.

Met de designer kun je een bedrijfsproces workflow of een integratie workflow definiëren door een van de vele vooraf gedefinieerde- of lege sjablonen (templates) te kiezen. De voor gedefinieerde sjablonen bieden zoiets als bijvoorbeeld een 'Peek-Lock' van een Service Bus-bericht. Naast de vooraf gemaakte templates, kun je ook met een lege beginnen en starten met het slepen van een trigger en acties naar het canvas van de designer - elke trigger, actie, conditie die je in de Visual Designer plaatst, wordt vastgelegd als JSON. Het laatste beschrijf ik in de Domain Specific Language voor Logic Apps.

Tot slot is de Visual Designer is geschreven in TypeScript/ React en wordt gehost in een iFrame in de portal. Bovendien is de designer in de Azure Portal dezelfde designer die beschikbaar is via de eerdere genoemde Azure Logic App Tools for Visual Studio. De ervaring is hetzelfde, ware het niet dat je de Logic App in Visual Studio niet kunnen laten draaien. Daar later mee over in dit artikel.




```
{
    "definition": {
        "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
        "actions": {
            "Create_variable_to_store_travel_time": {
                "inputs": {
                    "variables": [
                        {
                            "name": "travelTime",
                            "type": "Integer",
                            "value": "@div(body('Get_route_and_travel_time_with_traffic')?['travelDuration'],60)"
                        }
                    ]
                },
                "runAfter": {
                    "Get_route_and_travel_time_with_traffic": [
                        "Succeeded"
                    ]
                },
                "type": "InitializeVariable"
            },
            "Get_route_and_travel_time_with_traffic": {
                "inputs": {
                    "host": {
                        "connection": {
                            "name": "@parameters('$connections')['bingmaps']['connectionId']"
                        }
                    },
                    "method": "get",
                    "path": "/REST/V1/Routes/Driving",
                    "queries": {
```

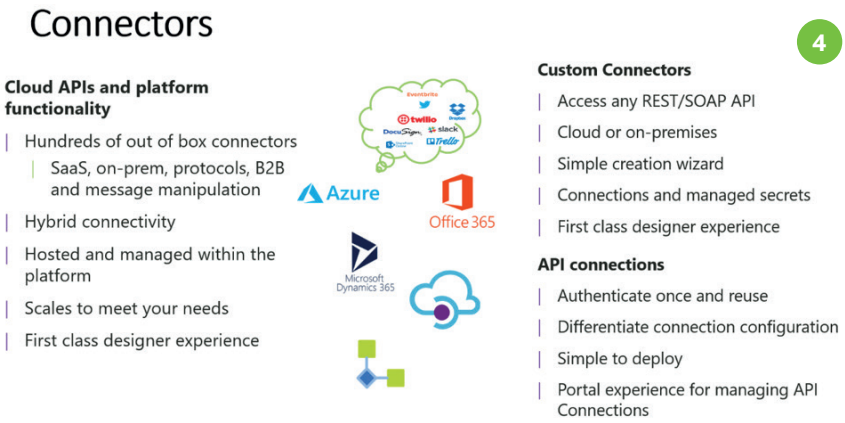
3

De Domain Specific Language (DSL) voor Logic Apps

Zoal eerder aangegeven kun je Logic Apps zien als een 'Workflow as a Service' in Azure, maar het gedraagt zich echter niet als een normale workflow of workflows - die je mogelijk zult kennen van Windows Workflow Foundation (WF). Een dergelijke workflow werkt zoals: doe eerst stap A, dan stap B, dan stap C, en je kunt verder allerlei voorwaarden en loopings hebben – het is een zogenaamde 'foward-chaining' proces. Echte, een Logic App gedraagt zich echter heel anders.

De runtime onder de Logic Apps is eigenlijk een planner (scheduler), die allerlei taken uitvoert op basis van een JSON gebaseerde DSL. En die de taken beschrijft welke uitgevoerd dienen te worden. Wanneer je naar de codeweergave kijkt, ziet je dat er een RunAfter-stap is. En er staat RunAfter step A: Stap B wordt uitgevoerd na stap A - dus wanneer je een workflow maakt, heb je een reeks van taken (acties) **(Figuur 3)**.

Wat de JSON DSL je biedt is een set van taken die je parallel kan laten lopen - elke actie wordt in wezen een taak die op de achtergrond wordt uitgevoerd. Een zogenaamde 'Resource



provider' interpreteert de DSL en laat vervolgens een orchestrator de taken uitvoeren.

Connectors

Eerder hebben we de workflow definitie en de DSL van Logic Apps behandeld en nu gaan we de connectiviteit – verbindingen naar andere systemen en diensten behandelen. De kracht van de Logic Apps zijn de beschikbare connectors voor diverse Azure diensten, en SaaS-diensten. Deze stellen je in staat om verbindingen (connections) aan te gaan. De connectors zijn in feite 'wrappers' om OpenAPI specificaties heen – oftewel de beschrijving van een API, waar Microsoft een implementatie op baseert. Het bedrijf heeft inmiddels meer dan drie honderd van dergelijke connectors gemaakt – welke je in de Logic App gebruik van kan maken. En mocht er geen connector zijn, dan bestaat er nog de mogelijkheid om zelf een connector te maken. Tot slot, kan je ook een proxy van een API aangemaakt in Azure API Management consumeren in de Logic App **(Figuur 4)**.

Bouwen van een Logic App

Een Logic App kun je eenvoudig in de Azure Portal aanmaken om vervolgens met de designer aan de slag te gaan om een workflow te maken. Je sleept een actie op het canvas voor de start van de flow, om vervolgens een of meerdere acties er op te laten volgen. Wat betreft acties zijn er diverse smaken, zoals:

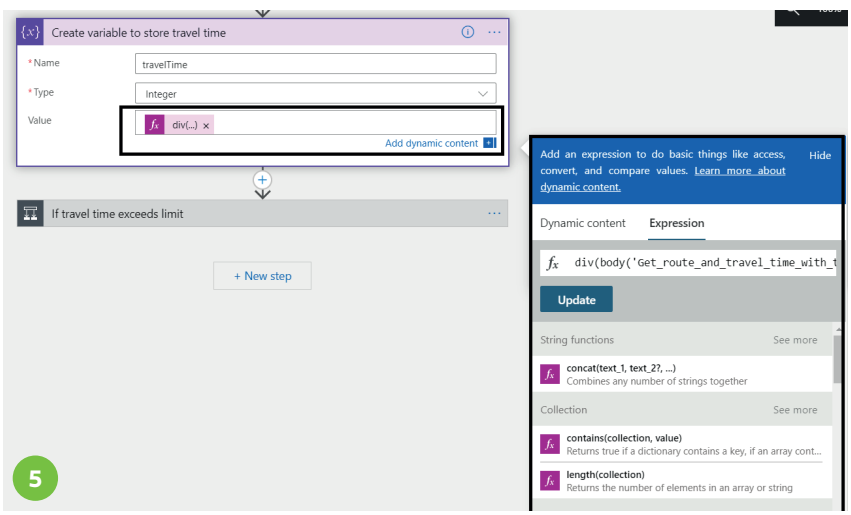
- Acties om een andere systeem of service aan te roepen (connector);
- Acties om controle op de flow te kunnen verkrijgen – denk aan condities ('*if else*'), loopings ('*do while*' en '*for-each*') en stoppen van de flow (*'terminate'*);
- Acties om transformaties uit te voeren.

Verder kun je ook bepaalde configuraties uitvoeren op de acties, zoals:

- Configureren van acties voor opnieuw proberen van een aanroep naar een ander systeem of service (zogenaamde *'retry'*)
- Configureren van al dan niet uitvoeren van een actie door middel van de *'RunAter'* setting – dus voer een actie uit als de vorige gefaald heeft;
- Configureren van acties door eigenschappen ('Tracked properties') aan te brengen voor monitoring doeleinden.
- Configureren van *'Time-outs'*, en in- en outputs van de actie te beveiligen.

Kortom, bij het opbouwen van de flow heb je als ontwikkelaar een aantal mogelijkheden om het robuust te maken. Eveneens kan je met behulp van zogenaamde 'function expressions' data in de berichten, die in de Logic App komen te manipuleren. Je kunt bijvoorbeeld de input (data) van json naar xml formaat transformeren of een rekenkundige toepassing uitvoeren (delen van een waarden door een getal) **(Figuur 5)**.

Door een workflow op te bouwen met alle mogelijkheden die de Logic App biedt met connectors en de designer kun je dus een integratie bouwen – wat je mogelijk gewend bent met BizTalk Server. Echter, biedt de Logic App geen 'publish-subscribe' mechanisme waar je berichten op een bus plaats of in het geval van BizTalk Server in een zogenaamde 'MessageBox' waar afnemers zich op kunnen abonneren. De Logic App is vergelijkbaar met de 'orchestration' functionaliteit



van BizTalk. Andere functionaliteiten van BizTalk zijn terug te vinden in de Azure Service Bus, Event Grid, API Management en de Integration Account. Kortom, deze diensten kun in combinatie met Logic Apps gebruiken om je integratie oplossing in Azure te realiseren wanneer ontkoppelen van systemen en services een eis is. In dit artikel richt ik me meer op een voorbeeld waar dat niet noodzakelijkerwijs hoeft.

Hosting Opties

Voor dat ik een voorbeeld van een Logic App ga beschrijven behandel ik nog een aantal andere zaken zoals hosting opties, uitrollen ('deployment'), monitoring en beveiliging ('security'). Wat hosting opties betreft kun je een Logic App aanmaken in de Portal als een 'Pay-as-you Go' optie. Hierbij wordt de Logic App op de Azure regie naar keuze aangemaakt, waarbij je samen met vele andere gebruikmaakt van de dezelfde infrastructuur van Azure. Je kan echter ook kiezen om dat niet te doen en een deel te reserveren voor eigen gebruik – dit is de 'Integration Service Environment' (ISE) optie. Met een ISE heb je een deel van de Azure infrastructuur voor jezelf alleen en betaal je daar ook voor – de 'Pay-as-you-Go' optie vervalt. Tot slot is er een derde optie met Logic App preview waarbij je wederom een 'Pay-as-you-Go' optie hebt echter nu draait de Logic App in een 'App Service' runtime (dezelfde als Azure Functions). Bij deze optie kun je Logic App ook in een container laten draaien en dus hosten in een Kubernetes cluster met de daarbij komende voordelen **(Figuur 6)**.

Deployment

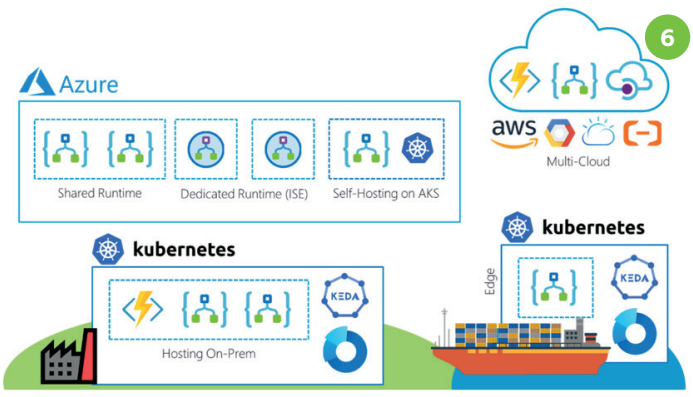
Wanneer je een Logic App hebt gemaakt en deze naar andere omgevingen wilt uitrollen dan kun je gebruikmaken van zogenaamde 'Azure Resource Management' (ARM) templates. Met de eerder beschreven Visual Studio Tools plug-in kun je met de Cloud Explorer in Visual Studio navigeren naar een Logic App, de workflow definitie (json) en bijhorende ARM template downloaden. Deze kun je vervolgens bewerken en van extra parameters voorzien alvorens je het borgt in een Azure repo. Vanuit de repo kun je een pipeline starten, die de ARM template gebruikt om de Logic App naar andere omgevingen uit te rollen. Naast Visual Studio kan je ook Visual Code gebruiken om naar de Logic App te navigeren in Azure om vervolgens de Logic App (JSON) te bewerken, enzovoort. **(Figuur 7)** Het belangrijk je te beseffen dat het uitrollen van Logic Apps je de workflow en ARM gebruikt en niet direct je Logic App publiceert via Visual Studio of de Logic App kopieert en verplaatst naar een andere omgeving.

Monitoring

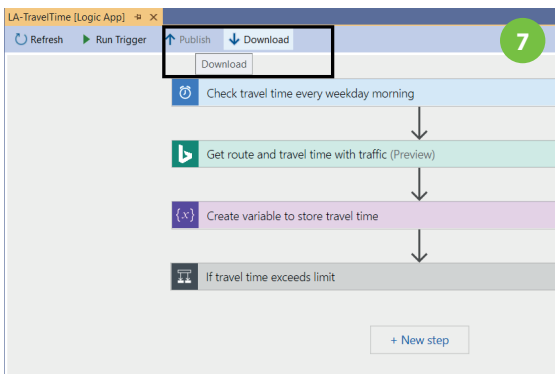
Een ander aspect wat van belang is voor Logic Apps is monitoring. De service biedt net als andere Azure diensten 'metrics', die je inzichtelijk kan krijgen via het tabblad onder monitoring. Je kunt echter ook via 'Diagnostic Settings' of bij aanmaken van de Logic App de 'metrics' van Logic App runs sturen naar een zogenaamde 'Log Analytics Workspace', zodat via de beschikbare functionaliteit dashboards e.d. kunt gebruiken voor monitoring doeleinden. Verder biedt het dashboard voor Logic Apps monitoring ook functionaliteit voor het herstarten van een of meerdere Logic Apps, inzage in eventuele 'tracked properties', en meer.

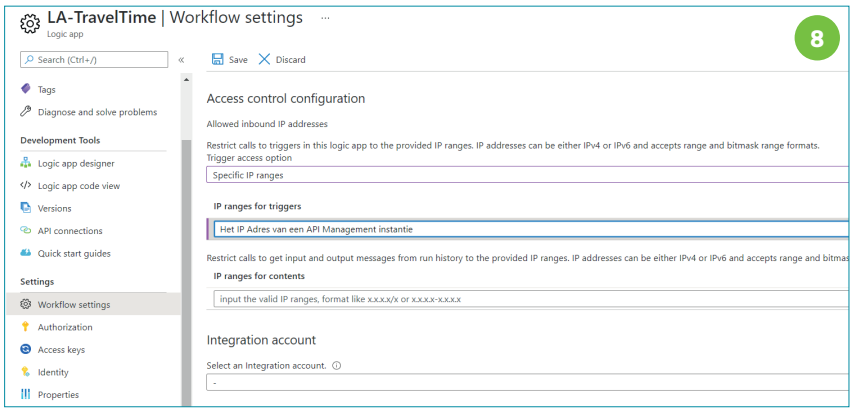
Security

Tot slot voordat ik een voorbeeld van een toepassing met een Logic App ga beschrijven wil ik nog in gaan op beveiliging. Logic Apps kunnen diverse andere Azure diensten benaderen waar je door middel van 'Role Based Access Control' (RBAC) functionaliteit zoals een 'Managed Identity' het een en andere wat betreft beveiliging kunnen regelen. Verder kun je een Logic App met een HTTP-trigger beveiligen via Azure API Management. Een Logic App met een dergelijke trigger krijgt een publiek benaderbaar adres ('endpoint'), waarbij je



Bron: https://www.codit.eu/blog/why-running-azure-logic-apps-anywhere-is-a-game-changer/





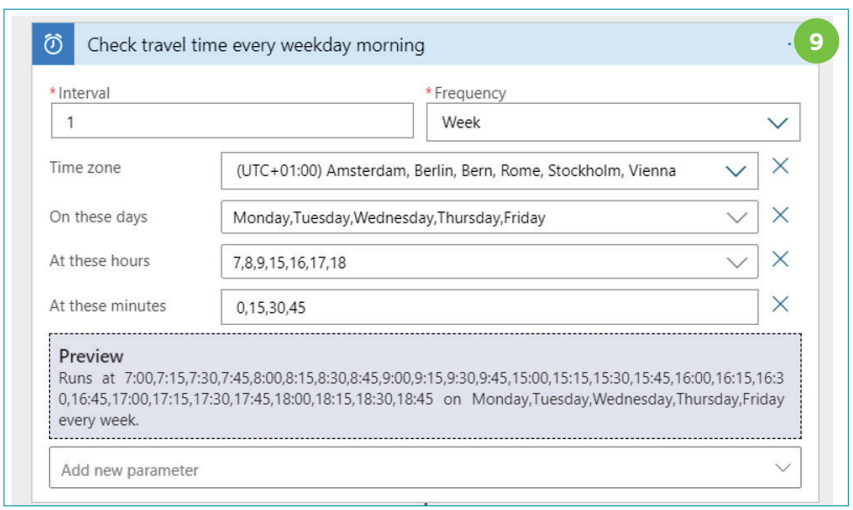
toegang mogelijk wilt reguleren. Door een proxy aan te maken in API-management kan je toegang beperken – ook door het IP-adres van API Management te gebruiken bij toegang tot de Logic App **(Figuur 8)**. Kortom, beveiliging speelt een belangrijke rol bij Logic Apps – niet alleen voor toegang echter ook de data die het binnenkrijgt – dus vergeet dat niet!

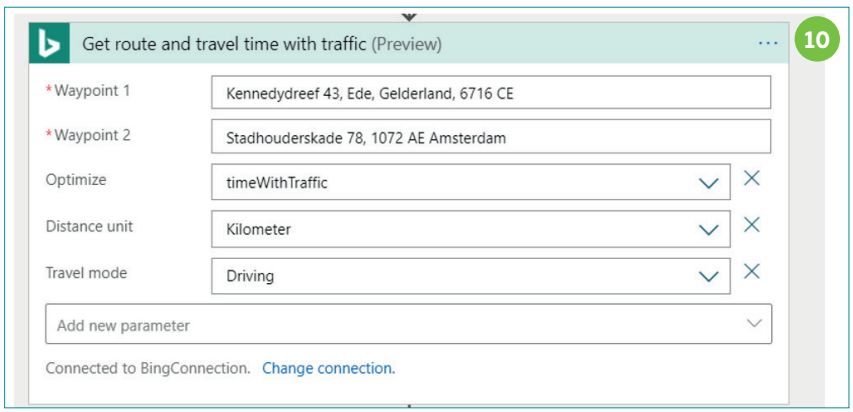
Scenario

Logic Apps bieden je met de connectoren heel veel mogelijkheden om een proces (workflow) of integratie te maken. Stel je wil bijvoorbeeld weten of de reistijd van je huis naar kantoor langer dan gebruikelijk gaat duren op bepaalde tijdstippen (07:00 – 09:00 en 15:00 – 18:00). Je kan Logic App bouwen met een 'Schedule trigger', die om het kwartier op de tijdstippen een flow start **(Figuur 9)**.
De vervolgactie naar de trigger gebruikt een 'Bing Maps' connector, die een methode heeft om de reistijd tussen twee gegeven adressen oplevert op basis van de verkeersdrukte op het tijdstip **(Figuur 10)**.
De uitkomst van de connector is input voor een vervolgactie, die de reistijd opslaat in een variabele met de volgende 'function expression':

```
div(body('Get_route_and_travel_time_with_traffic')?['travelDuration'],60)
```

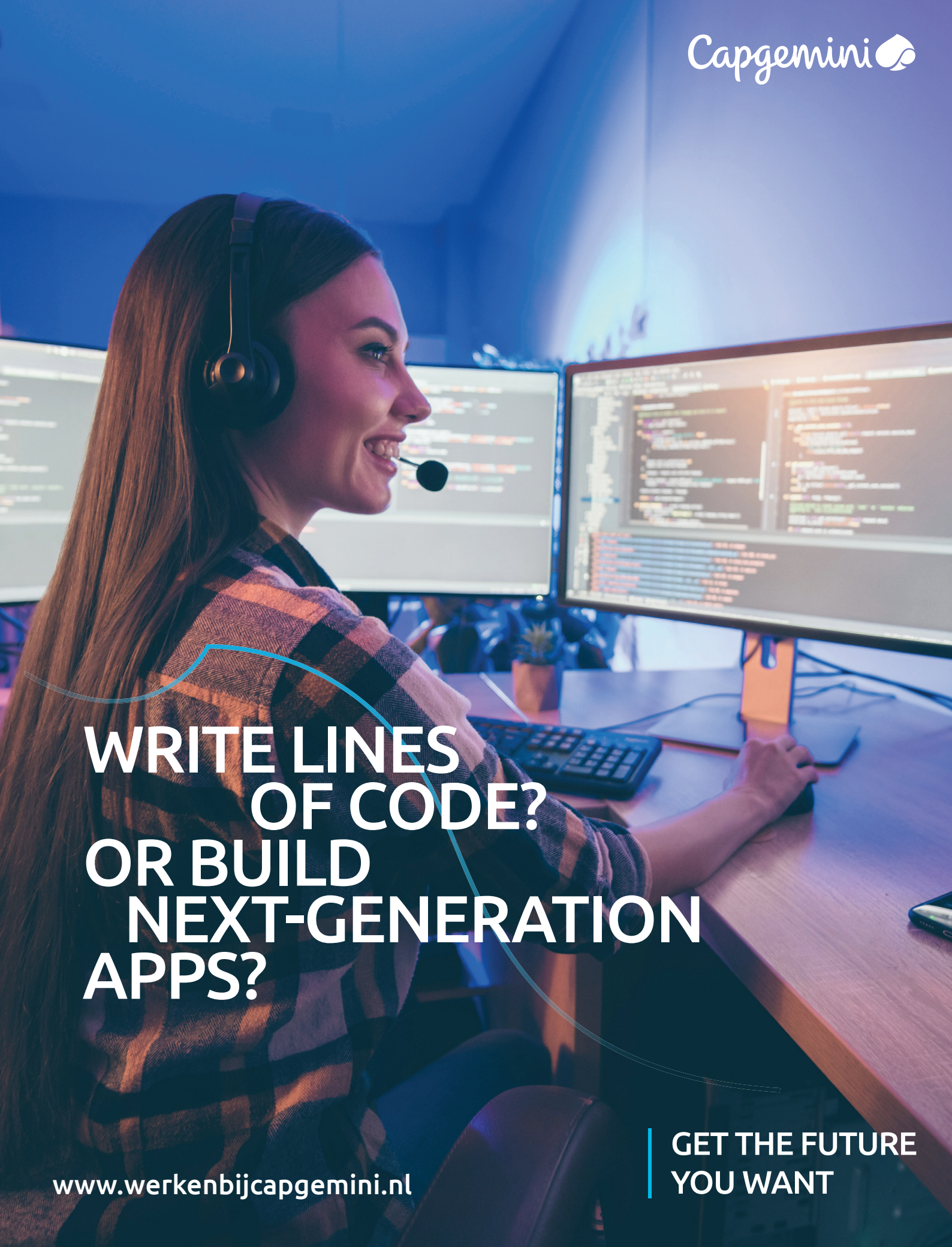
De waarde van de variabele wordt gebruikt in een conditie ('if else'), waarbij de reistijd hoger dan 35 minuten statement is opgenomen. Als dat het geval is zal een er een actie volgen met een email connector, die een mail stuurt **(Figuur 11)**. Dit voorbeeld geeft weer hoe je een proces (flow) kunt maken, die met een 'Bing Maps'- en email connector je op de hoogte kan brengen dat je mogelijk langer onderweg bent naar een bestemming waar je elke dag heen gaat. De flow kun je als een integratie met Bing Maps zien.





Afronding

Logic Apps bieden een mogelijkheid voor je om systemen met elkaar te laten communiceren en mogelijke integraties te bouwen. Logic Apps kunnen ook worden gebruikt om zaken te automatiseren in Azure. De service heeft zich sinds zijn beschikbaarheid vijf jaar geleden duidelijk ontwikkeld tot een volwassen dienst, die ook gebruikt kan worden in combinatie met andere diensten zoals Sentinel. Dit artikel heeft je een inkijk gegeven wat er bij bouwen van Logic Apps komt kijken en hoe je een proces kunt automatiseren en integraties kunt maken. ●


Capgemini
WRITE LINES OF CODE?
OR BUILD NEXT-GENERATION APPS?
www.werkenbijcapgemini.nl
GET THE FUTURE YOU WANT

We are **continuously uncovering**

Auteur: Sander Hoogendoorn

In 1983 I bought my first computer. A Commodore 64. Soon, I wrote my first code. In BASIC. That was thirty-seven years ago. From there, it didn't take long before I earned money from writing code. That was at the age of nineteen, during the first year of my computer science studies.
However, it wasn't until recently, during a conversation with my girlfriend Kim, who is just taking up programming, that I realized that when I wrote my first code, the field of software development had only existed for about forty years. It is a young field.
I learned how to code without automatically refactoring development tools, without being able to run code in the cloud, without unit testing, without automated pipelines, without continuous integration, and even without StackOverflow. We didn't even have the internet. All of this simply didn't exist thirty years ago.
The field of software development has evolved rigorously during its short existence. We went from the very first computers to a ubiquitous worldwide presence of software on a magnitude of devices in about seventy years.
And we have learned so much.
If you consider the methodologies and approaches for software development over the years a similar pattern emerges. Back in 1983, I didn't know much about methodologies. Yes, the waterfall existed, but my projects weren't large enough to apply it. It wasn't until my first job at a consultancy, in 1994, that I first encountered waterfall projects. It wasn't pretty. They went over budget, over time, and under-delivered. Project managers ruled relentlessly, processes were strict, and developers were mere resources.
But we learned. We started to realize that requirements change and feedback was useful, so we'd better take it into account. A new generation of methodologies emerged, such as the Unified Process and the Spiral Model, introducing increments and iterations, albeit usually in cycles of several months.
In the mid-1990s, I got acquainted with the Dynamic Systems Development Method (DSDM). DSDM has increments and iterations at heart and it introduced much shorter cycles. Along with improvements in languages, tools, and frameworks, we now delivered increments in iterations of six or eight weeks. And for the first time, these iterations included all activities necessary to deliver, such as design, coding, testing, integration, and releasing increments to the customer.
We learned again. The shorter the feedback loops, the higher the quality. And, the more rituals and ceremonies we could make redundant, the better and faster we became. Bye-bye big-upfront designs. Bye-bye Gantt charts. So long rigor project plans.
As a result, in 1998, my team and I developed the Smart approach that combined these learnings with again improved tooling, including code generation. It allowed us to shorten our iterations again, to lengths of three to four weeks. Not too long after that, my team and I learned about extreme programming (XP), and with it, we discovered the merits of refactoring, continuous integration, unit testing, and even autonomy, which shortened our iterations again.
Then in 2001, the Agile Manifesto was published.
It is needless to say, almost twenty later, that the Agile Manifesto made an unimaginably big impact on the world. It is a major driver in changing organizations in many different industries. For us, the Manifesto was a strong confirmation that we were on the right track. Smart was an agile approach, and its popularity piggybacked on the evangelization of agile.
Then, we bumped into Scrum, which soon became more popular than all of the other agile approaches together. Scrum almost became

> **We are continuously uncovering better ways of working. I'm already curious about how we will fare in the next decade**

synonymous with agile. Due to its simplicity Scrum attracted people from many other fields to software development. At the same time, our tooling improved again. Git largely improved version control systems. JUnit made unit testing tangible. Refactoring finally got supported out-of-the-box. Clear many impediments from our paths, we again were able to move faster. Our iterations shortened again, to one to two weeks.

Moreover, we learned that the process of making rituals and ceremonies redundant to become better and faster never stops. It is continuous. The ideas, values, and principles of the Agile Manifesto remain extremely valuable. But Scrum is not the end game.

Moving forward continuously means that its rituals and ceremonies will become redundant as soon as we can move beyond them. Exactly following the most important part of the Agile Manifesto. We are uncovering better ways of developing software by doing it and helping others do it.

Where does that leave us today? To cut a decade short. We now reap the benefits of continuous delivery, and being able to automatically test and deploy our code in the cloud. We leverage microservices architectures, DevOps, and infrastructure-as-code, allowing teams to prune even more overhead from their approaches.

For the last seven to eight years, I've helped organizations to move along these lines. Every time, the results were similar. Despite the resistance from traditional agile and Scrum representatives, we made more and more of the rituals and ceremonies redundant that we so happily introduced twenty years ago.

We now deliver individual features. Continuously. Therefore, there's no need to plan and fit our work items into iterations or sprints. We stopped doing sprints but rather delivered features multiple times per day. Without having to fit work into sprints, there was no need anymore for low-level estimates. We dropped those too. And instead of organizing day-long refinement workshops at the start of sprints, we rather briefly discuss each work item on the day we pick it up.

Each of our work items only requires a small subset of the skills of the whole team. Therefore, each work item is picked up exactly when we need to deliver it, but also exactly by who can deliver it. Tiny teams of one to three people work on tiny work items, for maybe one or two days, deliver it, and then disband again, to reform new tiny teams, often in different compositions. I refer to these teams as microteams, as a reference to microservices.

Again we learned to deliver software better and faster. We made the rituals and ceremonies of the current generation of agile approaches redundant, without losing the good parts. So bye-bye sprint planning and refinements. Bye-bye story point estimation. So long sprints. Still, we completely adhere to the Agile Manifesto, which clearly states that our highest priority is to satisfy the customer through early and continuous delivery of valuable software. Today we just deliver even smaller features, in even shorter cycles, with even smaller teams, in smaller components.

As an example, my current team builds a microservices platform for an IoT company. We bring the platform one step further every day. We don't have sprints. We don't estimate our work items. Our testing and pipelines are fully automated, and our infrastructure is immutable. We have very few ceremonies left and simply answer one question every day: what small problem are we solving today? We operate in microteams of constantly varying compositions and deliver work items multiple times per day. And actually, it's a breeze. If there's one thing I've learned from twenty years of agile and thirty-five years in this industry is that all methodologies are temporary, including agile approaches. We are continuously uncovering better ways of working. I'm already curious about how we will fare in the next decade. ●

BIO

Sander Hoogendoorn

Sander is an independent dad, speaker, writer, and traveler. He has been writing code since 1984 and still codes every day. He is a serial CTO, currently at e-commerce iBOOD. Sander is a code philosopher, agilist, and operated as Capgemini's global agile thought leader for a decade before going freelance in 2015.Following his adagio that small steps are the fastest way forward, Sander helps to empower organizations, teams, and individuals, and to disrupt their ways of working, technology, architectures, and code. He has authored various books and published tons of articles. Sander is a well-known and inspiring keynote speaker at international conferences on diverse topics such as disruption, culture, (beyond) agile, continuous delivery, microteams, monads, software architecture, microservices, and writing beautiful code. Tools do not solve problems, thinking does.



Maak je toekomst!

Loopbaanstress! Manager, architect of developer?

Heb je al nagedacht over je volgende stap in je loopbaan? Vind je het een lastige keuze of weet je de juiste argumenten niet te vinden? Hopelijk vind je in dit artikel antwoord in deze lastige keuze. Ik geef aan wat mijn keuze bepaald heeft en wat ik daar nu van vindt.



Auteur: Koen Aerts

Dit artikel schrijf ik eigenlijk met maar een doel, software developers behoeden voor een verkeerde keuze. De afgelopen jaren heb ik regelmatig gezien dat erg goede software developers switchen naar manager. Waarom maken ze deze keuze? Het laatste wat ik wil is een rol als architect of manager afraden, het zijn beide hele leuke rollen. Het hangt erg af van welke zaken je belangrijk vindt in een rol. Het doel van dit artikel is om het een bewuste keuze te laten zijn met hopelijk wat extra handvatten.

Mijn loopbaan keuze

Zelf heb ik informatica (software engineering) gestudeerd en ben vervolgens als software developer aan de slag gegaan. Na eerst veel van senioren geleerd te hebben krijg ik steeds meer de gelegenheid om mijn ideeën (architectuur, design patterns, etc.) in projecten kwijt te kunnen. Het is een super interessante wereld en fascinerend wat met techniek allemaal mogelijk is. Bij mijn werkgever zit ik op dat moment met de beste developers



in een denktank om de lijn van het bedrijf te bepalen en ben ik actief betrokken bij kennisdeling. Ik krijg waardering van mijn werkgever en hij ziet mijn kwaliteiten als software developer.
Door deze erkenning groeit het zelfvertrouwen en daardoor ben ik gaan nadenken over de toekomst. Ik dacht destijds dat het ophield als je senior developer was, inmiddels weet ik dat het daar pas begint. In het volgende hoofdstuk meer over het perspectief van de software developer.
Ik dacht destijds dus dat er weinig ontwikkelmogelijkheden als (senior) developer waren. Daarnaast wil ik graag impact maken. Ik dacht dat dit als manager beter zou lukken. Inmiddels ken ik echter vele software developers die veel meer impact maken dan de meeste managers. Ik ben er nu van overtuigd dat je in beide rollen impact kunt maken.
Een ander belangrijk argument bij mijn afweging is het salaris geweest. Ik probeer een baan te vinden die leuk is en ook goed betaald. Of dit nu een goed argument is of niet, ik heb gemerkt dat het mijn keuze beïnvloedt. Ik was van mening dat een developer niet zoveel kon verdienen als een manager of architect. Vandaag de dag geven diverse bedrijven aan dat de beste developers net zoveel verdienen als de beste managers. Dit is een filosofie waar ik helemaal achter kan staan!
Het laatste argument wat mij erg bezig hield was het continu bij moeten blijven. Nieuwe frameworks, talen, technieken, etc. Hou ik

Whether you think you can, or you think you can't. You're right.
Henry Ford



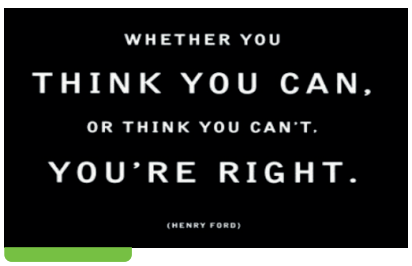
Koen Aerts

dat vol tot mijn 67ste? Ik merk dat best wel veel software developers met deze vraag worstelen en ik vind het lastig om er antwoord op te geven. Wat ik in ieder geval in mijn omgeving zie is software ontwikkelaars van 60+ die nog steeds ontzettend goed zijn. Op de hoogte van alle laatste trends en qua snelheid zeker niet onderdoen voor de jongere collega's. Blijven investeren in je vak is wel een voorwaarde, maar dit geldt ook als je richting management of architectuur zou gaan. Voor velen van ons is IT een hobby, hoe moeilijk is het om hier tijd aan te besteden? Achteraf vind ik dit niet zo'n sterk argument meer, maar het heeft wel invloed op mijn keuze gehad.
Uiteindelijk heb ik de keuze gemaakt om manager (in eerste instantie project manager) te worden. Voor mij is het een prima keuze geweest, want ik vind het vak van manager

echt super leuk. Het is alleen wel apart dat ik de keuze gemaakt heb op basis van argumenten waarvan ik nu de meeste niet meer valide vind. Met de kennis van nu zou ik weer dezelfde keuze maken, maar dan met het argument dat management een leuk vak is.
Uiteraard zijn er nog vele andere argumenten te bedenken, bovenstaande argumenten waren voor mij relevant. Een argument wat voor mij niet meegespeeld heeft is 'het gevleid voelen' als je voor manager of architect gevraagd wordt. Dit voelt goed en uiteraard wil je hier met je werkgever over meedenken. Diezelfde werkgever heeft er echter niets aan als hij straks een goede developer kwijt is en een ongemotiveerde manager/architect terug heeft.

Het perspectief van een software developer

Software development is een heel mooi vak en heeft ontzettend veel ontwikkelmogelijkheden. Jammer dat ik tijdens mijn keuze die mogelijkheden onvoldoende gezien heb. Ik weet niet precies hoe dat komt, wellicht dat het 'gras bij de buren groener lijkt'. Middels dit artikel probeer ik in ieder geval te benadrukken dat 'het gras' van een software ontwikkelaar heel groen is, je hebt een wereld aan mogelijkheden!
Mocht ik nog in het vak zitten, zou ik voor mezelf een ambitieus doel neerzetten, bijvoorbeeld auteur van een bestseller, internationaal bekend



Vertel iedereen wat je volgende stap is en geloof hierin, dan wordt het werkelijkheid!



spreker of contributor aan een veelgebruikt open source project.
Vervolgens zou ik natuurlijk naar dat doel toe gaan werken. Wat voor mij erg goed werkt is om mijn volgende stap tegen iedereen aan te houden en er ook van overtuigd zijn dat dit de volgende stap is (ook al lijkt deze hoog gegrepen). Ik heb gemerkt dat je er op de eerste plaats zelf in gaat geloven, maar ook je omgeving en dan komt het vanzelf uit (selffulfilling prophecy)!
Bij de ontwikkeling wordt vaak gesproken over softskills en hardskills. De meer 'harde' skills is het bekwamen in de programmeertalen, frameworks, tooling, etc. In onze wereld heeft dit een beperkte houdbaarheid omdat vernieuwing hier ontzettend snel gaat. Het bij blijven is essentieel, maar realiseer je wel dat deze kennis tijdelijk houdbaar is. Vaardigheden die 'onbeperkt houdbaar' zijn, zijn de softskills. Vanuit die optiek en ook omdat deze skills helpen om een goede developer te worden is het belangrijk hier ook veel in te investeren.
Op deze manier kun je een hele carrière bezig blijven met je ontwikkeling in de rol van software developer. Je bent best een tijdje bezig voordat jij de volgende Martin Fowler bent!

Werkgever

Zoek een werkgever die ondersteuning geeft aan alle perspectieven dat het vak van software development te bieden heeft. Natuurlijk is het mooi als zij ook de filosofie heeft dat developers en managers net zoveel zouden moeten verdienen.

Mocht je het management- of architectenvak overwegen dan is het belangrijk dat je hier kennis mee kunt maken zonder dat je al een definitieve keuze hoeft te maken. Fijn als je werkgever hier ook invulling aan kan geven.
Zodra je uit de techniek gaat worden de functies algemener en de vraag minder. Het zal nog lang duren voordat het tekort aan developers niet meer bestaat. Managers en architecten zijn er genoeg. Sterker nog, met de komst van zelfsturende teams zijn de project managers overbodig geworden. Heftig als je van software developer naar project manager geswitched bent. Wellicht gebeurd er straks hetzelfde met scrummasters en is het alleen nog een rol voor developers in het team. Wellicht kan dit ook een argument zijn in je overwegingen.

Conclusie

Software development is een ontzettend mooi vak met groeimogelijkheden voor een hele carrière. Creëer een omgeving waar deze groei ook mogelijk voor je wordt en communiceer je ambitie met deze omgeving. Uiteraard zijn andere rollen als architect en/of manager ook ontzettend leuk, maar als je de keuze maakt voor een overstap doe het dan met de juiste argumenten. Mocht je twijfelen dan is het natuurlijk erg prettig als de nieuwe rol kunt 'uitproberen'. Praten met je omgeving en/of mensen die al in die rol zitten helpt natuurlijk ook. Hopelijk heeft dit artikel je richting gegeven in je carrière, heel veel succes!

BRANDMANNEN
Recruitment Marketing

door het vuur voor talent.

job marketing & employer branding



Maak kennis met het team van BRANDMANNEN, een recruitment marketing bureau gespecialiseerd in job marketing en employer branding. Wij gaan door het vuur voor talent.

Bij BRANDMANNEN maken we slimme campagnes voor vacatures èn combineren we dit met sterke content. Zo staat het werkgeversmerk niet alleen optimaal op het netvlies van de doelgroep maar raak je ook nog de juiste kandidaat recht in het hart. Met als resultaat een voortdurende stroom getalenteerde kandidaten die bewust voor jouw bedrijf kiezen.

Wie de doelgroep ook is, de employer branding content die wij maken is spot-on. Onze slimme job marketing campagnes zijn live te volgen via het Brandmannen ATS. We sturen campagnes tussendoor bij voor een optimale conversie. Binnen ons team hebben we ieder ons eigen specialisme, samen opgeteld een ijzersterke formule. De magie ontstaat in de samenwerking met de klant.

Kijk ook eens op brandmannen.nl/vacatures

Laat je versienummer **aan je pipeline over!**

Het valt me regelmatig op dat versioning van .NET assemblies in CI/CD pipelines niet altijd volledig wordt geautomatiseerd. Ik vind dat altijd ontzettend jammer. Het hele doel van een pipeline is toch om snel en consistent bruikbare pakketjes op te leveren? Dan zou het ook fijn zijn als het juiste versienummer automatisch bepaald kan worden.

Auteur: Arno Peters

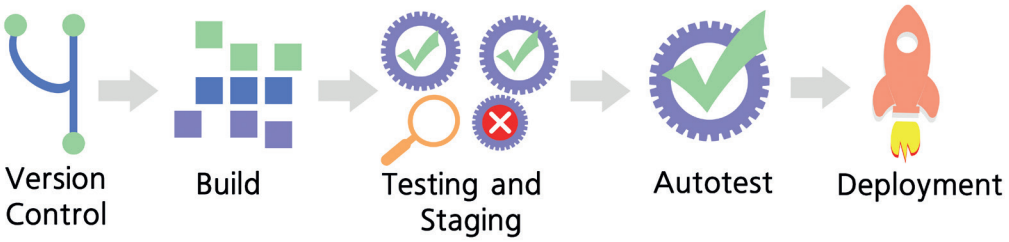
In .NET heb je minimaal de 'AssemblyVersion' property nodig, die is beperkt tot (maximaal) 4 cijfers: [Major].[Minor].[Patch].[Revision]. Door subjectieve keuzes over de waardes van die 4 cijfers, lukt het vaak niet om een versienummer automatisch te bepalen. Een voorbeeld: *"deze change is best groot, dus laten we dan maar het [Minor] cijfer ophogen in plaats van de [Patch]"*, zonder dat "grootte" echt goed te meten valt. Of *"Marketing wil graag een 12 op de doos, dus laten we dat als [Major] kiezen"*. En soms wordt een deel van het versienummer dan wel weer aangevuld met een gegenereerd nummer.
De reden dat me dit stoort, is dat een (deels) handmatig bepaald versienummer ervoor zorgt dat je altijd blijft knoeien met source control en branches. Je moet op het laatste moment vlak voor de release wat in je source zetten, anders klopt het nummer niet, en als je dit op een branch wil doen om iets te kunnen testen en het later wil mergen, krijg je daar weer extra werk van. Een veel voorkomend gevolg is dan weer dat ervoor gekozen wordt om dan maar alleen vanaf de main branch te releasen – een rigide strategie die helemaal niet nodig is.
De oplossing is eigenlijk eenvoudig: maak gebruik van een tweede versienummer! Een 'AssemblyVersion' is noodzakelijk, maar dient puur voor de techniek. .NET faciliteert ook een 'InformationalVersion' property: een string waarin er een label gezet kan worden om aan gebruikers te tonen. 95, 98, maar ook XP en Vista; als er een commercieel gewenste versie is, kan deze daarin geplaatst worden.
Hiermee kan de 'AssemblyVersion' volledig gebruikt worden voor cijfers bepaald vanuit de techniek. Rest de vraag: wat is dan een handig versieschema?
Ik kom daarbij altijd uit op Semantic Versioning (ook wel afgekort tot SemVer). Deze methode gebruikt de eerste 3 posities [Major].[Minor].[Patch] van het versienummer.
Waarom kies ik hiervoor? Vooral omdat het exacte versienummer in de praktijk niets uitmaakt. Ik kan een heel schema verzinnen gebaseerd op allerlei niet meetbare concepten, maar er is in de praktijk niemand die

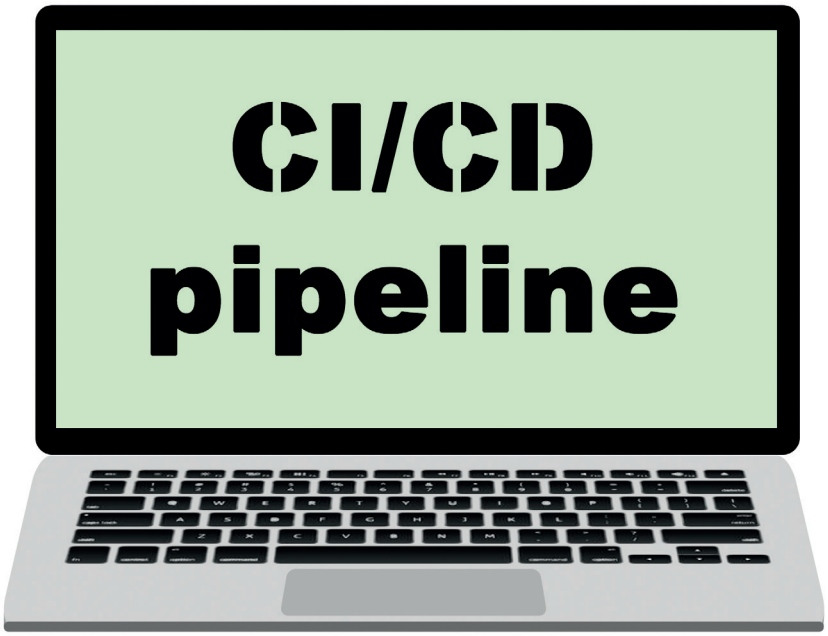
BIO

Arno Peters

Ik ben een software engineer met meer dan 10 jaar ervaring. Naast dat ik me helemaal richt op het Microsoft-landschap en graag mooie dingen maak met C#, vind ik het meeste plezier in het ondersteunen van development-teams. Hierbij heb ik een sterke focus op DevOps en de processen daaromheen, en draag ik met passie een pragmatische aanpak van Agile uit.







daar dan wat mee doet. Gebruikers weten zelden meer van hun software dan de commerciële versie, en als developer kijk ik alleen naar technische versienummers als ik een update wil doorvoeren of er iets niet werkt, dan kan ik de juiste documentatie erbij pakken.
Semantic Versioning lost dit heel elegant op door de nummers van een version eigenlijk los van elkaar te zien. Het kijkt niet naar het formaat van de change, maar enkel naar de betekenis van je commit.

- Als je enkel wat bugs fixt, maar niks nieuws toevoegt: [Patch] +1
- Als je wel wat nieuws toevoegt, maar geen breaking changes hebt: [Minor] +1
- Als je breaking changes hebt: [Major] +1

Er wordt niks vastgelegd over HOEVEEL bugfixes, of (breaking) changes je hebt. SemVer helpt je bij het inschatten hoe ingewikkeld een up- of downgrade tussen twee versies zou kunnen worden, zonder dat je direct de documentatie in moet.
Het geheim om dit automatisch in mijn pipelines te kunnen doen zit in de tool GitVersion. Deze tool is voor een heleboel buildplatforms beschikbaar, en kan als stap in een pipeline opgenomen worden. Je maakt vervolgens de betekenis van je commits meetbaar met behulp van een prefix in je commitmessages, en GitVersion doorloopt je commit history om de actuele versie te berekenen. Er is dus geen magie: als developer ben je volledig in control.
GitVersion kan ook overweg met branches en pre-release versies: er kan naast de 'AssemblyVersion' ook een suffix bepaald worden (met daarin bijvoorbeeld de branchname) die gecombineerd een unieke waarde oplevert. En die past weer keurig als tekst in de 'InformationalVersion', die op dat moment toch niet gebruikt wordt voor commerciële doeleinden. Op deze manier kan ik zonder zorgen mijn pipeline een correct genummerde release laten produceren, vanaf iedere commit, en vanaf iedere branch.
Nog een tip rondom de vindbaarheid van het versienummer in source control: maak gebruik van commit tagging in je pipeline, op het moment dat je echt een release doet. Dan vind je de juiste commit een stuk vlugger dan via een contentsearch in de history van je repository, EN het kan uiteraard prima opgenomen worden in de pipeline.
Als je meer informatie wil over Semantic Versioning vind je op https://SemVer.org, en voor GitVersion is https://gitversion.net/docs een goed startpunt. ●

Het hele doel van een pipeline is toch om snel en consistent bruikbare pakketjes op te leveren?



Service Locator Pattern

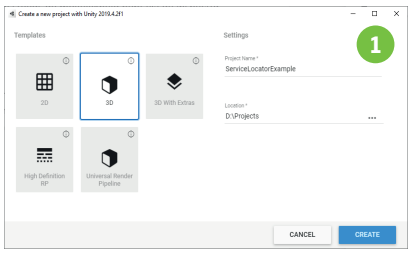
Het 'Service Locator Pattern', hoe kun je deze gebruiken in Unity3D voor het ontwikkelen van een VR applicatie.

Wat is dit pattern precies en waar kunnen we het voor gebruiken. Het 'Service Locator Pattern' is een design pattern dat in softwareontwikkeling gebruikt wordt om afhankelijkheden tussen verschillende classes te verminderen. Door gebruik te maken van een centrale plek om alle services te bewaren, de zogenaamde 'Service Locator' wordt een centrale plek gecreëerd, die de juiste service voor een bepaalde taak kan teruggeven. Dit design pattern bestaat al lang en is goed gedocumenteerd met diverse hulpbronnen op internet. Maar het gebruik van dit pattern binnen Unity is minder bekend.
In Unity wordt voor een 'Service' of centrale 'Controller' meestal gebruik gemaakt van 'singletons', hiervan zijn diverse voorbeelden terug te vinden. Nadeel van Singletons is dat je niet een centrale plek hebt, waar je kunt vinden welke services er binnen de applicatie allemaal beschikbaar zijn. Over het algemeen wordt het gebruik van 'singletons' afgeraden, toch is het in deze situatie handig. Door gebruik te gaan maken van het 'Service Locator Pattern' heb je straks nog maar 1 'singleton' in je hele applicatie nodig, namelijk de 'Service Locator'. In de onlinevoorbeelden over het gebruik van dit pattern binnen Unity3D vond ik met name voorbeelden om classes te gebruiken die je niet binnen Unity3D in hoefde te stellen. Maar vaak wil je juist een aantal instellingen vanuit de 'Inspector' in Unity3D instellen. Om dit toch mogelijk te maken moeten we een aantal dingen net even wat anders aanpakken.
In dit volgende deel wil ik jullie stap voor stap laten zien, hoe je binnen Unity3D gebruik kunt maken van het 'Service Locator Pattern' en nog steeds de 'Inspector' kunt gebruiken om de properties in te stellen.

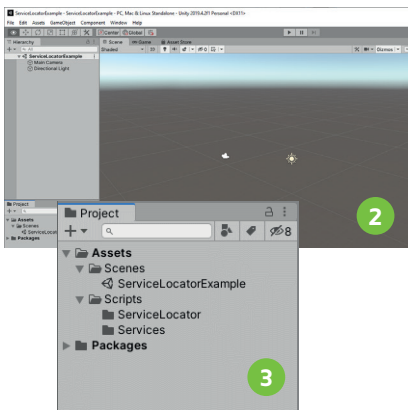
Project binnen Unity3D

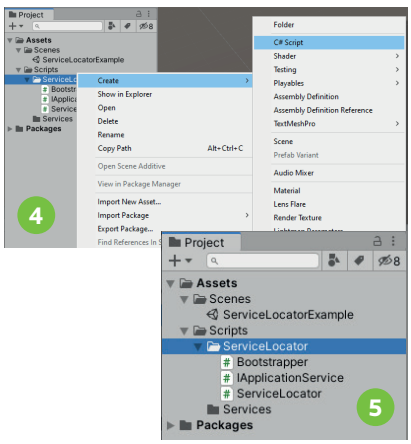
Uiteraard beginnen we met het starten van Unity3D, ik doe dit zelf altijd met behulp van de 'Unity hub' appli-

catie zodat ik eenvoudig meerdere versies van Unity op mijn systeem kan ondersteunen. Hier kies je voor het aanmaken van een nieuw project: **Voorbeeld 1**



Ik kies hier voor een 3D project, en vervolgens geef je jouw project een naam en locatie waar deze kan worden opgeslagen. **Voorbeeld 2**
Je krijgt vervolgens een schoon leeg project met daarin alvast een 'SampleScene' die ik zelf ook direct een duidelijke naam geef. Om het een en ander straks netjes te kunnen





organiseren maak ik in de projectstructuur direct een map aan voor de 'Scripts' met hierin weer een map voor de 'ServiceLocator' scripts en een map voor de 'Services'. **Voorbeeld 3**
Door op de map 'ServiceLocator' met de rechtermuis te klikken heb je de mogelijkheid hier scripts aan toe te voegen. **Voorbeeld 4**
Hier voegen we nu een drietal scripts aan toe, te weten 'Bootstrapper', 'IApplicationService' en 'ServiceLocator'. **Voorbeeld 5**
Nu gaan we deze scripts stuk voor stuk vullen met code. Door op een file te dubbelklikken zal Visual Studio geopend worden en kun je de code bewerken. In de 'IApplicationService' voegen we vervolgens de volgende code toe: **Voorbeeld 6**
Dit is vooral handig om straks eenvoudig te constateren dat een Service geschikt is voor het gebruik van de 'Service Locator'. Vervolgens open je de code voor de 'ServiceLocator', hier voeg je de volgende code in: **Voorbeeld 7**
Het bootstrapper script moet telkens bijgewerkt worden als er een nieuwe Service bijkomt. Deze zal ik dan ook pas gaan vullen zodra zo de eerste services gemaakt zijn. Voor dit voorbeeld heb ik alvast een tweetal services bedacht namelijk de 'PlayerService' en de 'LevelService', deze gaan we nu aanmaken op dezelfde manier als we eerder al scripts hebben toegevoegd. Zodra dit klaar is zal de projectstructuur er als volgt uitzien: **Voorbeeld 8**
Deze nieuwe services gaan we ook van wat code voorzien, aangezien dit slechts een voorbeeld is heb ik wat van mijn fantasie gebruikt wat deze services doen. Ik ga ervan uit dat jullie met veel nuttigere invullingen gaan komen dan ik hier doe :).
De volgende code zet je in de 'LevelService': **Voorbeeld 9**
En de volgende code zet je in de 'PlayerService': **Voorbeeld 10**
In deze 'PlayerService' zie je ook direct hoe je heel eenvoudig de 'ServiceLocator' kunt aanspreken om een bepaalde service op te halen, de code daarvoor staat geel gemarkeerd in bovenstaande code. Nu we twee services tot onze beschikking hebben moeten we (zoals eerder al gezegd) ook de 'Bootstrapper' gaan aanvullen met deze code regels: **Voorbeeld 11**
De oplettende lezer ziet hier al iets terug, wat je in de onlinevoorbeelden niet kunt vinden. Namelijk dat we in de code op zoek gaan naar 'GetComponentInChildren', dit doen we hier zodat we via de 'inspector' instellingen aan de services kunnen configu-

```
/// <summary>
/// Base interface for our service locator to work with. Services implementing
/// this interface will be retrievable using the locator.
/// </summary>
public interface IApplicationService
{
}
```
6

BIO

Roy Janssen

Roy werkt vanuit zijn eigen onderneming Semper IT Services in diverse rollen zoals onder andere technisch architect, lead developer en consultant binnen kleine en grote projecten. Als trainer en spreker deelt hij graag zijn kennis over C#, WPF, XAML, HoloLens, NUI, AR/VR en Make-Code met de community.



```
using System;
using System.Collections.Generic;
using UnityEngine;

/// <summary>
/// Simple service locator for <see cref="IApplicationService"/> instances.
/// </summary>
public class ServiceLocator
{
    private ServiceLocator() { }

    /// <summary>
    /// currently registered services.
    /// </summary>
    private readonly Dictionary<string, IApplicationService> services = new Dictio-
nary<string, IApplicationService>();

    /// <summary>
    /// Gets the currently active service locator instance.
    /// </summary>
    public static ServiceLocator Current { get; private set; }

    /// <summary>
    /// Initalizes the service locator with a new instance.
    /// </summary>
    public static void Initiailze()
    {
        Current = new ServiceLocator();
    }

    /// <summary>
    /// Gets the service instance of the given type.
    /// </summary>
    /// <typeparam name="T">The type of the service to lookup.</typeparam>
    /// <returns>The service instance.</returns>
    public T Get<T>() where T : IApplicationService
    {
        string key = typeof(T).Name;
        if (!services.ContainsKey(key))
        {
            Debug.LogError($"{key} not registered with {GetType().Name}");
            throw new InvalidOperationException();
        }

        return (T)services[key];
    }

    /// <summary>
    /// Registers the service with the current service locator.
    /// </summary>
    /// <typeparam name="T">Service type.</typeparam>
    /// <param name="service">Service instance.</param>
    public void Register<T>(T service) where T : IApplicationService
    {
        string key = typeof(T).Name;
        if (services.ContainsKey(key))
        {
            Debug.LogError($"Attempted to register service of type {key} which is
already registered with the {GetType().Name}.");
            return;
        }

        services.Add(key, service);
    }

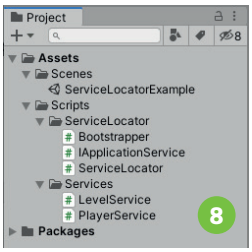
    /// <summary>
    /// Unregisters the service from the current service locator.
    /// </summary>
    /// <typeparam name="T">Service type.</typeparam>
    public void Unregister<T>() where T : IApplicationService
    {
        string key = typeof(T).Name;
        if (!services.ContainsKey(key))
        {
            Debug.LogError($"Attempted to unregister service of type {key} which is
not registered with the {GetType().Name}.");
            return;
        }

        services.Remove(key);
    }
}
```

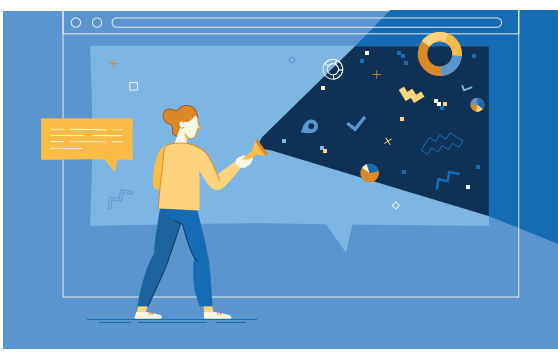
7

> Vind deze en de andere listingen uit het artikel op sdn.nl!

reren. Om dit voor elkaar te krijgen moet je wel de structuur binnen je scene goed inrichten. Dat is dan ook de volgende stap die we gaan nemen. In de 'Hierarchy' gaan we op het hoogste object (root object) met de



rechtermuis klikken en kiezen we ervoor om een Empty GameObject te maken. **Voorbeeld 12**
Onder dit betreffende GameObject gaan we vervolgens nog twee Empty GameObjecten maken. Deze geef je namen, en sorteer je zodat de 'Hierarchy' er als volgt uit komt te zien: **Voorbeeld 13**
Aan deze lege GameObjecten gaan we vervolgens de juiste scripts toevoegen. Dit doe je door eerst in de 'Hierarchy' het juiste GameObject te selecteren en vervolgens in de 'Inpector' te klikken op de knop 'Add component'. Je kunt vervolgens het juiste script zoeken en toevoegen, zie volgende schermafdruk: **Voorbeeld 14**
Nadat je het 'Bootstrapper' script hebt toegevoegd kun je vervolgens het juiste script aan de twee services toevoegen. Als je vervolgens de 'PlayerService' bekijkt in de 'Inspector' zie je het volgende: **Voorbeeld 15**
Hier zie je nu dat je gebruik maakt van een service die via de 'Service Locator' opgehaald kan worden. Maar je ziet ook dat je nog steeds de mogelijkheid hebt om properties zoals 'Player Name' en 'Score' behorende bij het object via de 'Inspector' eenvoudig in te stellen.



Op basis van bovenstaande voorbeeld is het vervolgens erg eenvoudig zelf scripts te gaan schrijven, die de daadwerkelijke game flow gaan uitvoeren. Vanuit deze scripts is het op basis van het gegeven voorbeeld heel eenvoudig zelf via de 'Service locator' een service op te gaan halen.

En ik kan me goed voorstellen dat je zelf de bestaande services nog verder wil uitbreiden en zelfs eigen services toe wil voegen.

Conclusie

Door gebruik te maken van het 'Service Locator Pattern' voorkom je dat je diverse 'Singleton' classes nodig hebt in je project, je kunt je beperken tot het maken van 1 'Singleton' namelijk de 'ServiceLocator'. Deze biedt dan de mogelijkheid diverse services binnen je applicatie beschikbaar te stellen, die je ook nog gewoon kunt instellen via de 'Inspector' binnen Unity3D. Als je alleen de aangeboden services 'Service' in de naam geeft dan zorgt code completion ervoor dat je ze zeer eenvoudig terug kunt vinden op het moment dat je via de 'ServiceLocator' een service terug wil halen. Let wel goed op dat je alle gemaakte services ook opneemt in je 'Bootstrapper' anders kom je tijdens run-time in de problemen! ●

```
using System.Collections.Generic;
using UnityEngine;

public class LevelService : MonoBehaviour, IApplicationService
{
    public Level CurrentLevel;

    private LevelService()
    {
        CurrentLevel = Level.Beginner;
    }

    public void NextLevel()
    {
        if (CurrentLevel != Level.Expert)
        {
            CurrentLevel++;
        }
    }
}

public enum Level
{
    Beginner = 0,
    Intermediate = 1,
    Advanced = 2,
    Expert = 3
}
```

9

```
using UnityEngine;

public class PlayerService : MonoBehaviour, IApplicationService
{
    private Level currentLevel;
    private LevelService currentLevelService;

    public string PlayerName;
    public int Score;


    public void Awake()
    {
        currentLevelService = ServiceLocator.Current.Get<LevelService>();

        currentLevel = currentLevelService.CurrentLevel;
    }

    public void PlayerReachesNextLevel()
    {
        currentLevelService.NextLevel();
    }
}
```

10

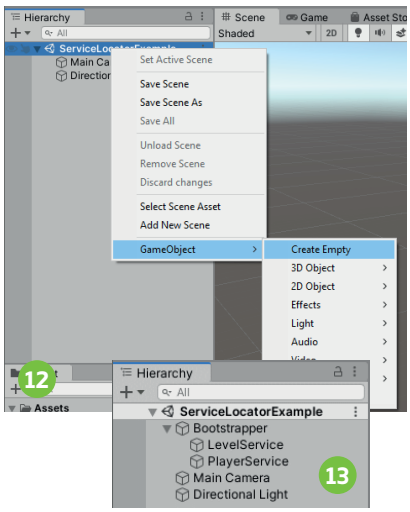
```
using UnityEngine;

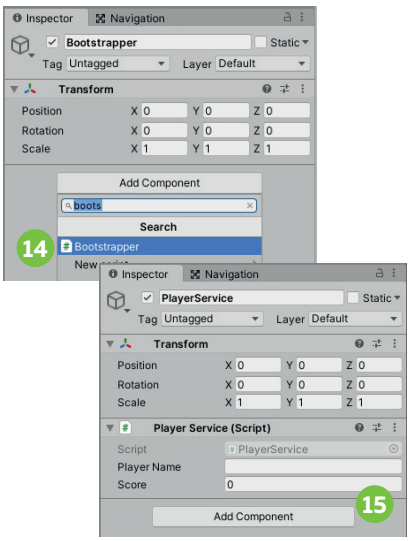
public class Bootstrapper : MonoBehaviour
{
    public void Initiailze()
    {
        // Initialize default service locator.
        ServiceLocator.Initiailze();

        // Register all your services next.
        ServiceLocator.Current.Register(this.GetComponentInChildren<LevelSer-
vice>());
        ServiceLocator.Current.Register(this.GetComponentInChildren<PlayerSer-
vice>());
    }

    private void Awake()
    {
        this.Initiailze();
    }
}
```

11





FUTURE TECH 2022

THE IT CONFERENCE FOR MICROSOFT TECHNOLOGIES

March 17, 2022 Jaarbeurs Utrecht

The Call for Papers has opened!

Submit your paper here: sessionize.com/future-tech-2022/





Maanden geleden zijn wij als SDN de transitie gestart naar een verfrissende, nieuwe en volgende stap in de 30 jaar lange traditie. Met deze verandering willen we nog meer als voorheen een belangrijk onderdeel worden in het Microsoft community eco systeem. Waar we vooral willen binden en verbinden, developers samenbrengen en via allerlei verschillende kanalen kennis met elkaar te delen. Via grote en kleine events, magazine, twitter en de vernieuwde website waarbij bijna alles mogelijk is.

De belangrijkste verandering is ons vernieuwde logo met het schakeltje van verbinding in het hart van de Microsoft development community. Gekoppeld aan een moderne site waar kennis en ervaring gedeeld gaan worden. Ook onze twitter handle hebben we hierop aangepast, @sdn_community geeft beter weer wat wij willen zijn. Hou ons in de gaten, wij waren niet weg en gaan nog lang niet weg.

Zoals jullie in het voorwoord al hebben kunnen lezen maken we weer gebruik van allerlei commissies om de juiste content voor jullie te verzamelen. Voor het magazine doen we dat met een redactiecommissie en voor de events buigt een events commissie zich over de stortvloed aan sessievoorstellen om de krenten uit de pap te halen. Vind je het leuk om mee te doen en mee te beslissen over de inhoud? Stuur dan een mail naar info@sdn.nl voor meer informatie.

De SDN-cast blijft een vast onderdeel en ook daar gaan we de komende tijd nog veranderingen in aanbrengen om hier op een andere manier kennis en informatie te delen. Ook daarvoor zijn we altijd op zoek naar mooie en interessante verhalen. Ken je iemand of wil je zelf graag eens gast zijn in de SDN-cast, meld je dan op info@sdn.nl.

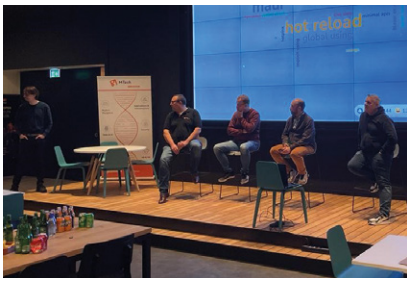


Terugblik SDN Night .NET6 Launch
Powered by Ordina

Op dinsdag 9 november heeft SDN in samenwerking met Ordina MTech een .NET 6 Launch Event bij Ordina in Nieuwegein georganiseerd. In de Plaza van Ordina kwam tijdens deze SDN Night een groep van 40 Microsoft Professionals bijeen voor de release van Microsoft .NET 6 en Visual Studio 2022.

Microsoft heeft maandag 8 november de 2022-release van Visual Studio en dinsdag 9 november de definitieve versie van .NET 6 algemeen beschikbaar gemaakt. Met deze versie van Visual Studio (VS) hebben developers voor het eerst een 64-bit versie van het platform ter beschikking. Deze 64-bit versie moet meer mogelijkheden bieden om alle systeembronnen te benutten. Vooral als developers gedurende langere tijd met bepaalde oplossingen moeten werken. Met VS 2022 kunnen developers bijvoorbeeld dagenlang het platform laten draaien, zelfs met oplossingen die meer dan 700 projecten omvatten.

De avond begon met een hapje en een drankje. Gevolgd door de opening van de meet-up door Roelant Dieben (Sr. Cloud Architect bij Ordina MTech) en Kelly Verschoor (Partner Manager bij SDN). Om 17.00 uur keken we live naar de keynote van Microsoft. Een interessante sessie van een uur waarin Scott Hunter en z'n team alle ins & outs van .NET 6 presenteerden. Na de keynote stond er een heuse pizzafoodtruck buiten waar iedereen een heerlijke verse pizza kon laten bereiden. Vervolgens was het tijd voor een inhoudelijke panelsessie met verscheidene .NET experts aan het woord die de keynote evalueerden. Op interactieve wijze barstte de discussie los over de lancering van .NET 6. De experts en de meet-up deelnemers wisselden hun ervaringen en verwachtingen uit. Zo gingen we in discussie over de voor- en nadelen van .NET 6 ten opzichte van het .NET Framework. Daarnaast hebben we gezien dat Microsoft Blazor nog steeds erg belangrijk vindt en hebben we de nieuwe features besproken. Ten slotte sloten we de avond af met een borrel waarin nog veel werd nagepraat met elkaar over de nieuwe ontwikkelingen binnen Microsoft! ●







nieuw

Gigaset

Smart Home

Gigaset Video Deurbel (1920x1080p Live view)

- Live beelden, pushnotificatie en screenshot op smartphone en smartwatch gratis service
- Twee weg audio comminicatie
- Notificatie via bestaande gong, Philips Hue, Gigaset Plugs en Sirene en Google Home*/ Alexa
- Versleutelde peer-to-peer video verbinding met veilige Duitse Cloud
- Vervanging van bedrade deurbel (dus zonder batterijen)

* na update

HD Buitencamera
Resolutie 1280x720. Met nachtzicht en bewegingsdetectie. Zowel WiFi als LAN aansluiting.

Compacte Full HD camera
(1920x1080p). Geïntegreerde microfoon en speaker voor twee-weg audio en nachtbereik. Werkt op bewegingsdetectie.

Universal
Controleert of deur/raam geopend, gesloten of ge-forceerd wordt. Universal meet ook lucht-druk en temperatuur in de ruimte.

motion
Bewegingsmelder Is er iemand in uw woning? Met motion weet u het meteen.

Siren
Slaat alarm met 85 db bij poging tot inbraak.

SECURITY

SECURITY

button
Zonder smartphone controle over het Gigaset Smart Home systeem.

smoke
Waarschuwt u via alarm en smartphone bij rookontwikkeling.

Works with Google Assistant, Philips Hue en Amazon Alexa.

COMFORT

ELEMENTARY

plug
Regel b.v. uw verlichting via smartphone, waar u ook ter wereld bent.

water
Waarschuwt u bij waterlekkage. Geeft ook lucht-vochtigheid en temperatuur in de ruimte weer.

HEATING

thermostat
Regel de temperatuur in uw kamer op afstand of via tijdschema. Maak van uw radiatorthermostaatknop een slimme thermostaat.

climate
Meet temperatuur en lucht-vochtigheid en communiceert met Thermostaat. Informeert over ventilatie intervallen voor een gezonde leefomgeving.

Verkrijgbaar bij:

coolblue bol.com expert Phone House

gigaset.com

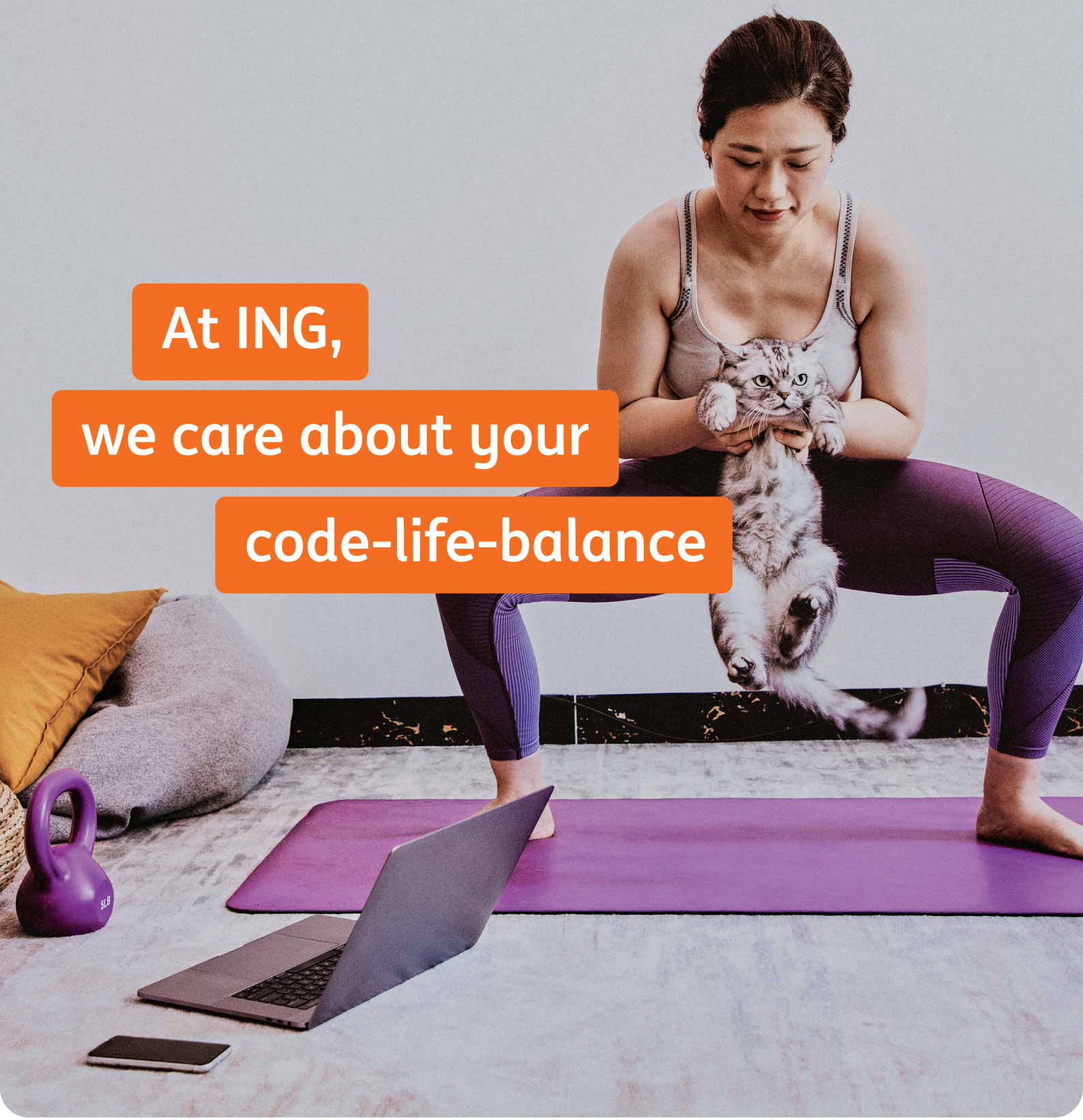
Menno Jongerius

Over het doolhof dat front-end ontwikkeling heet

Er was vroeger een programma op televisie waarbij mensen cavia's door een doolhof lieten lopen. Degene die het eerst bij de uitgang was die won de prijs. Ik weet niet hoe het met jullie is, maar als ontwikkelaar voel ik me ook wel eens zo'n cavia, maar volgens mij heeft iemand de uitgang dichtgemetseld. Het was ergens rond 2003 toen ik mijn eerste applicaties voor het web schreef. Niet gehinderd door enige kennis koos ik een platform dat dicht bij mijn kennis van desktopapplicaties zat: ASP.NET Webforms. Niet de beste technische keuze van Microsoft ooit, maar ik kon het best lang volhouden. In 2008 gingen we vol goede moed over op ASP.NET MVC, dat geeft toch een mooie scheiding van verantwoordelijkheden ... Ja, wel beter, maar wat dacht je van een API met een apart front-end, dan kun je de data beter scheiden van de visualisatie, en je hebt meteen een open systeem. Dus in 2010 maar aan de slag met het bouwen van Web API's. Wat doen we dan met het front-end? Weet je wat. Angular.JS dat lijkt met wel wat. Hmmm, wordt wel een beetje traag als je front-end groter wordt. Gelukkig schrijven ze een nieuwe versie. Klein nadeel is wel dat je zo ongeveer je hele front-end op de schop moet gooien. Nee, dan maar iets met React? Of nee Vue, dat is een prachtige techniek. Ja maar React heeft wel een grote community hoor. Trouwens, ik houd niet zo van javascript. Misschien toch maar overstappen op Blazor, dat ziet er toch veelbelovend uit.
Ondertussen zit de business verwachtingsvol naar ons te kijken. Wat moeten we doen? En als we nu investeren in deze technologie, hoelang blijft het dan draaien? Ik durf er al bijna geen antwoord op te geven. Misschien is er licht aan het eind van de tunnel. Kunnen we met een oplossing komen om onze front-end net zo modulair op te zetten als de backend? En dan bedoel ik met modules systemen die we volledig zelfstandig kunnen ontwikkelen en deployen en niet een of ander NPM package graag. Ik heb het antwoord nog niet gevonden, maar ik hoor graag of iemand al een weg heeft gevonden om uit het doolhof dat front-end ontwikkeling heet te klimmen. ●



Menno Jongerius



71% of our IT colleagues agree

Visit ing.nl/careers/it

do your thing

Iris de Groot

De magie in online samenwerking

Als software engineer binnen Ordina ben ik onderdeel van een 'High Performance'-team. Deze teams worden multidisciplinair gevormd, en de banden binnen het team worden actief versterkt door middel van trainingen en activiteiten. Het leuke aan een dergelijk team binnen detachering is dat je als solide element binnen het ecosysteem van een bedrijf binnenkomt. Je levert direct een complete agile bijdrage, naast slechts de solitaire kennis en ervaring van één developer die zijn draai nog moet vinden. Én je leert elkaar echt goed kennen. Dat komt mede door onze onconventionele manier van werken.

Ik kwam tijdens de lockdown het team binnen. Lang hebben we samengewerkt zonder elkaar in het echt gezien te hebben. Het enige nadeel bleek het niet op de hoogte zijn van elkaars lengte. Af en toe gistten we er naar: "Je lijkt me lang, maar toch geen twee meter?", om vervolgens te horen dat de betreffende collega toch exáct twee meter is.

Naast dit soort 'strubbelingen' bracht de situatie van beginnen in een online team echter veel voordelen met zich mee. Binnen ons team hebben we een manier van werken die in eerste instantie voor iedereen ongewoon klinkt, maar écht een voordeel levert voor het developervak.

Door standaard in een call te zitten simuleren we het naast elkaar werken. De sfeer is informeel. Naast dat je je prima langere tijd stil kunt houden om ook het in de realiteit bestaande 'stilzwijgend naast elkaar zitten' te simuleren, hoef je geen mailtje te sturen om te vragen wat dat ene endpoint nog maar was, of tien minuten op het antwoord te wachten terwijl je collega 'idle' in de chat aanwezig is.

Een grapje is ook snel gemaakt. Je bent op de hoogte van het gewicht van ieders kat en hoort op de achtergrond hoe het vierjarige dochtertje van één van je collega's zich van de trap lanceert, waardoor je direct snapt waarom hij even weg is. (waargebeurd) Toch zeker niet onbelangrijk als je samen wilt werken!

Alle gekkigheid echter daargelaten: die voordelen voor het developervak die ik noemde? Het blijkt ontzettend handig met z'n zessen tegelijkertijd volledig mee te kunnen kijken op één scherm, zonder daarbij achter één bureau te verkrampen. Nooit eerder had ik de mogelijkheid zo makkelijk te pair-programmen of situaties te debuggen. Het komt de kwaliteit van code enorm ten goede. Mocht er iets mis zijn met de machine van collega, ook dan kan deze bijdragen door mee te kijken, oplossingen aan te dragen en feedback te leveren. Ken je de applicatie nog niet waar een collega aan werkt? Je wordt er binnen no-time doorheen geloodst.

Zelf weet ik niet of ik nog terug zou willen naar een andere werkwijze. In de realiteit zal dat tegenvallen, maar ik trek er in ieder geval lering uit voor de toekomst. Het is transparantie en kennisdeling ten top. En voor de echte binnenvetters onders ons: De openheid van dit concept brengt een codekwaliteit met zich met zich mee die menig developerhartje sneller zal doen kloppen.



Iris de Groot

SDN

Software Development Network

Word nu lid voor
maar €69,95 per jaar
en ontvang de volgende voordelen!

Wees er snel bij, het eerste jaar dat je lid bent krijg je 50% korting en betaal je maar €35,-!

> **Gratis toegang tot Future Tech**
> **Gratis toegang tot SDN University sessions**
> **4 keer per jaar een gedrukt exemplaar van het SDN Magazine**

Voor elke serieuze.NET developer en/of bedrijven die Microsoft technologieën gebruiken is het een must om onderdeel te worden van de SDN. Als je up-to-date wilt blijven, is dit de manier om dat te doen.

Ga naar sdn.nl/lidworden/ voor meer informatie

Meet the High Performance Team



Dat klinkt goed toch? Een High Performance Team met alleen jouw eigen collega's!

Maar wat is een High Performance Team precies?
Wat doet zo'n team en waarom zou ik daar meer van willen weten?
Zou dat iets voor mij zijn?

Teams zijn de toekomst en misschien wel jouw toekomst.
Ben je nieuwsgierig hoe Software Development in een High Performance Team eruit ziet? Hoe je samen meer kan bereiken?

Wil jij free format vragen stellen, en het echte verhaal horen van leden uit 3 verschillende teams? Stuur een mailtje naar Cindy.davids@ordina.nl.

ORDINA
Ahead of change