SDN MAGAZINE

Nummer
143
Jun 2022

SDN Magazine is
een uitgave van:
SDN

Van en voor Microsoft & .NET professionals



FUTURE TECH 2022

THE IT CONFERENCE FOR MICROSOFT TECHNOLOGIES

> Distributed Tracing met Application Insights

> Een goede QA Engineer heeft een holistische blik


TIKKIE WON'T BE INNOVATIVE FOREVER
#STARTWITHUS
After successfully launching apps like Tikkie and Grip, it would be easy to say "Okay, we made it". But that's not how innovation works. Innovation is about constantly challenging yourself. Daring to acknowledge that something can always be better. Must be better. Because it's this mindset that makes the difference. And to make a difference we need you. Employees that will continue to motivate others, and us.
abnamro.nl/beginbijons
ABN·AMRO

Colofon

Uitgave
Software Development Network
26ste jaargang
Nr. 143 • juni 2022

Redactiecommissie:
Annejan Barelds, Nadine Wolff, Jan de Vries, John Bruin, Menno Jongerius, Marcel Meijer, Roy Janssen, Roelant Dieben, Vincent Hendriks.

Auteurs:
Kitty van Leeuwen, Florian Schaal, Annejan Barelds, Jasper Sprengers, Patrick Bes & Arjen Kaak, Francois Bouteruche, Maarten Grootonk, Jacob Duijzer, Johan Smarius.

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

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

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

©2022 Alle rechten voorbehouden. Niets uit deze uitgave mag worden overgenomen op welke wijze dan ook zonder voorafgaande schriftelijke 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.

Beste SDN-ers,

Het zonnetje komt zo nu en dan al goed door en de temperaturen stijgen weer. Dat kan natuurlijk maar een ding betekenen: de zomer staat weer voor de deur! Alle vakanties komen er weer aan, maar voordat het zover is, hebben we nog de Future Tech conferentie!

Dit magazine staat dan ook in het teken van de conferentie. Midden in het magazine vindt je de beursspecial met onder andere de plattegrond, timetable en sponsoren. Zo kan iedereen alvast in de stemming komen om samen met ons, op 21 juni 2022 in de Jaarbeurs, Utrecht, er een mooi feestje van te maken!

Uiteraard gaat dit magazine niet alleen over Future Tech, want wij hebben weer een mooie club auteurs die toffe artikelen heeft ingestuurd. Onze redactiecommissie is dan ook druk bezig geweest om er een mooie editie van te maken, en ik denk dat dat zeker weer gelukt is! Hopelijk jullie natuurlijk ook.

Wij danken daarom de volgende auteurs voor hun bijdrage aan dit magazine: Kitty van Leeuwen, Florian Schaal, Annejan Barelds, Jasper Sprengers, Patrick Bes & Arjen Kaak, Francois Bouteruche, Maarten Grootonk, Jacob Duijzer, Johan Smarius.

Heel veel plezier met het lezen van het magazine en de voorpret die jullie mogen beleven aan Future Tech. Hopelijk zien we jullie allemaal op 21 juni in de Jaarbeurs, Utrecht.

En, vergeet niet, als jij ook in het SDN magazine wilt staan, mail dan jouw artikel naar redactie@sdn.nl

Kelly Verschoor
kelly.verschoor@sdn.nl



De beste bedrijfs-kritische software. In Azure.



Marktleider in software-ontwikkeling, -duiding en opleiding in Microsoft Azure





betabit {...}

samen bijzonder maken;

In dit nummer

6 Azure Functions isolated process

11 Distributed Tracing met Application Insights

14 OWASP 2022 In de praktijk

17 Een goede QA Engineer heeft een holistische blik

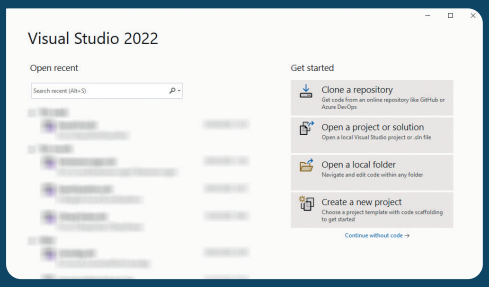
19 Future Tech Special

35 50 Shades of .NET on AWS

39 Externe REST services aanroepen met Refit

42 Visual Studio Solution Filters





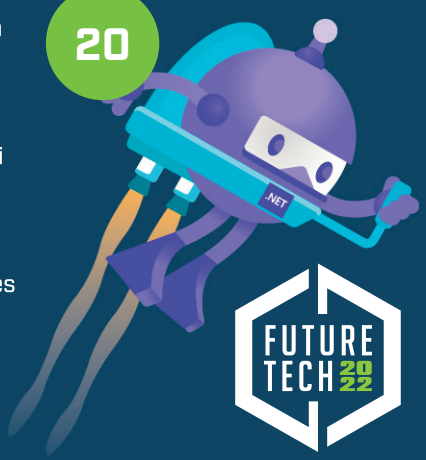
48 Code analyseren en genereren met Roslyn

54 Doorgeschoten vakmanschap

11 De meeste developers die wel eens iets met de Azure cloud doen, zijn wel bekend met Azure Application Insights (AI). AI is onderdeel van Azure Monitor, en is met minimale configuratie te gebruiken als sink voor de logregels vanuit je applicatie. Zeker voor applicaties die op een Azure App Service Plan draaien is dit heel gemakkelijk in te richten. Je zet een verwijzing naar Application Insights in je configuratie, hengelt de .NET Logging extensions naar binnen, en klaar!

39 Je kunt je bijna geen applicatie meer voorstellen die geen gebruik maakt van externe gegevens. Vaak komt deze data uit REST Api's1. Als developer kan het behoorlijk saai zijn om de zoveelste API te moeten ontsluiten maar gelukkig zijn er mogelijkheden om dit een stuk eenvoudiger te maken. Één van die opties is het gebruik van Refit2.

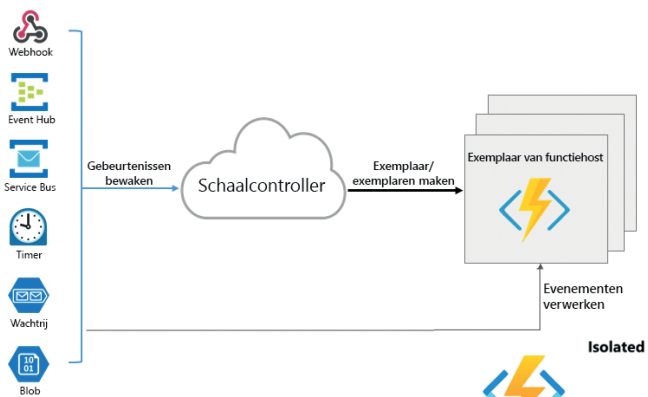
20



Azure Functions **isolated process**

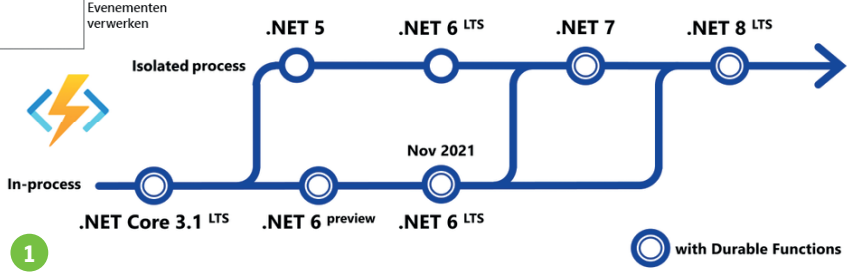
Met de komst van .NET 5 en nu met de LTS versie .NET 6 is er een nieuwe stap gezet voor Azure functions, Azure Functions isolated process. Dit ging echter niet zonder de nodige kritiek en struikelblokken omdat de architectuur van Azure Functions niet gemaakt was om out of the box support te bieden voor .NET 5 en later .NET 6. In dit artikel ga ik in op de veranderingen binnen de architectuur die nodig waren om support voor deze nieuwe versie van .NET te bieden. Daarnaast laat ik zien wat de grootste verschillen zijn tussen in-process (oude architectuur) en isolated process en hoe je met isolated process aan de slag kunt. Tenslotte kijken we nog naar Middleware, wat we al kennen van ASP.NET, maar nu ook beschikbaar is binnen Azure Functions doormiddel van de switch naar isolated process.

Auteur: Florian Schaal



In de oude architectuur van Azure Functions werd er gebruik gemaak van in-process execution, wat betekend dat de code die je schrijft als class library gedraaid wordt binnen hetzelfde process als de host. In het begin leek dit veel voordelen te bieden omdat men zo toegang had tot de host-api en verschillende bindings.

Het grote nadeel wat meteen duidelijk werd met de komst van .NET 5 is dat een koppeling tussen de host en de function code ook betekend dat de .NET versie van onze function code en de host (.NET Core 3 en Azure Function Runtime v3) gelijk moet zijn. Toen .NET 5 aangekondigd werd was er dan ook in de eerste versies nog geen support binnen Azure Functions. Dit omdat support van .NET 5 betekende dat de host ook op .NET 5 moest draaien en dus een rewrite nodig had. Hierdoor ontstond binnen de .NET community veel omdat zelfs AWS had eerder support voor .NET 5 binnen Lambda Functions aangezien bij Lambda functions vanaf het begin de keuze was gemaakt om de host en de function code los te trekken van elkaar.
Door de ophef en omdat een rewrite van de host veel tijd in beslag zou nemen, is er uiteindelijk gekozen om ook binnen Azure Functions de host en function code los te trekken. En zo werd Azure Functions isolated process geboren. Met deze nieuwe en losse architectuur is het mogelijk om de host op .NET Core 3.x te laten draaien en je function code bijvoorbeeld op .NET 5 of .NET 6.
Inmiddels is .NET 6 gereleased maar voelen we nog steeds een aantal nadelen van de oude architectuur. Zo zijn bijvoorbeeld Durable Functions nog steeds niet beschikbaar binnen het isolated process. Deze worden verwacht in .NET 7, wat betekent dat men voor nu bij in-process moet blijven als je gebruik maakt van durable functions. Inmiddels is het wel mogelijk om .NET 6 te gebruiken binnen de in-process architectuur.
Er is geen ondersteuning voor durable functions in .NET 5 omdat dit ook geen LTS versie is. **Voorbeeld 1**.



1

Verschillen

Azure Functions isolated process werd in het begin van van 2021 GA. Zoals je eerder al las is er bijvoorbeeld nog steeds geen support voor Durable Functions binnen Isolate Process. Buiten dit grote gemis zijn er ook andere verschillen waar je zeker rekening mee moet houden als je wilt upgraden of een nieuwe function wilt maken gebaseerd op dit nieuwe model. Ik som ze hieronder even kort voor je op.

Startup vs Program

Binnen in-process functions wordt er gebruik gemaakt van een startup class. Deze is niet nodig om de function te runnen, maar als je zaken zoals dependency injection wilt gebruiken dien je deze toe te voegen. Omdat de function code gedraaid wordt als een class library binnen de host, wijkt deze class af van wat men van andere .NET applicaties gewend is. Ga je hiermee voor het eerst aan de slag, dan kan dit heel verwarrend zijn. **Voorbeeld 2**.
Bij een function met isolated process is dit vervangen door de Program.cs class die standaard wordt toegevoegd wanneer je een nieuw project aanmaakt. Als we deze class bekijken (**voorbeeld 3**) dan zien we dat dit heel erg veel lijkt op wat we gewend zijn van ASP.NET. Het is daarom ook meteen veel duidelijker hoe men met de HostBuilder aan de slag kan.

Dependency Injection

Zoals we net zagen was het bij een in-process function al niet heel intuïtief om een startup class toe te voegen, maar het voelde i.c.m. Dependency injection ook wat hacky. Niet als of het gemaakt was om hier goede support voor te bieden. Begrijp mij niet verkeerd, het werkt prima en deed wat het moest doen. Willen we nu gebruik maken van dependency injection binnen een isolated process function dan kunnen we dit op dezelfde manier doen zoals we dit binnen ASP.NET doen. Dit voelt meteen een stuk beter aan. **Voorbeeld 4**.

ILogger

Bij een in-process function wordt de ILogger geinject binnen de function. Deze kun je dan meteen gebruiken om errors of informatie te loggen. Dit is binnen een function met isolated process niet zo. Binnen isolated process wordt een FunctionContext geinject. Deze context bevat veel informatie over de huidige execution van de function. Binnen deze context is ook de logger beschikbaar. Deze kan je verkrijgen door de GetLogger methode aan te roepen en hier een categorie aan mee te geven. In de meeste gevallen zal dit de naam van de function zijn waarin de logger gebruikt wordt. **Voorbeeld 5**.

Output bindings

Als we kijken naar output bindings heeft hier ook een shift plaatsgevonden, wat je code naar mijn mening een stuk duidelijker maakt. Als je in het verleden met in-process functions een bericht op een queue wilde plaatsen doormiddel van een output binding, dan werd dit gedaan door een IAsyncCollector te injecten in de Run methode. **Voorbeeld 6**.
Met isolated process functions is dit verplaats naar een attribute. Je function werkt dus nu altijd met return values zoals een array, string of serializable class. Wil je dus één item op de queue zetten, dan return je een string. Wil je er meerdere op de queue plaatsten, dan return je bijvoorbeeld een list. **Voorbeeld 7**. Dit maakt de code een stuk duidelijker tegenover het oude model.
Ook meerdere output bindings binnen één function worden ondersteund binnen een isolated process function. Stel we hebben als voorbeeld een string die we op een queue willen plaatsen en we willen een http response geven. Om hier mee aan de slag te gaan moeten we een wrapper class maken waar we de verschillende output types inzetten. **Voorbeeld 8**.

```
[assembly: FunctionsStartup(typeof(Startup))]

namespace inprocessSDN
{
    1 usage
    public class Startup : FunctionsStartup
    {
        public override void Configure(IFunctionsHostBuilder builder)
        {
            builder.Services.AddLogging();
        }
    }
}
```
2

```
public class Program
{
    public static void Main()
    {
        var host = new HostBuilder()
            .ConfigureFunctionsWorkerDefaults() // IHostBuilder
            .Build();

        host.Run();
    }
}
```
3

```
var host = new HostBuilder()
    .ConfigureFunctionsWorkerDefaults()
    .ConfigureServices((serviceCollection) =>
    {
        serviceCollection.AddHttpClient();
    }) // IHostBuilder
    .Build();
```
4

```
[Function(name: "HelloWorld")]
public static HttpResponseData Run([HttpTrigger(AuthorizationLevel.Function, params methods: "get")] HttpRequestData req,
    FunctionContext executionContext)
{
    var logger = executionContext.GetLogger(categoryName: "HelloWorld");
    logger.LogInformation(message: "C# HTTP trigger function processed a request.");
```
5

```
[FunctionName("HelloWorld")]
public static async Task RunAsync(
    [HttpTrigger(AuthorizationLevel.Function, params methods: "get", "post", Route = null)]
    HttpRequest req, [Queue(queueName: "myqueue", Connection = "")] IAsyncCollector<string> outputQueue, ILogger log)
{
    await outputQueue.AddAsync(item: "Hello World");
}
```
6

```
[Function(name: "HelloWorld")]
[QueueOutput(queueName: "output-queue")]
public static List<string> Run([HttpTrigger(AuthorizationLevel.Function, params methods: "get")] HttpRequestData req,
    FunctionContext executionContext)
{
    return new List<string> { "Hello World", "Hello World!" };
}
```
7

```
public class MyResponseBindings
{
    [QueueOutput(queueName: "output-queue")]
    public string MyQueueItem { get; set; }
    public HttpResponseData HttpResponseData { get; set; }
}
```
8

Zoals je ziet hebben we nu onze QueueOutput attribute verplaatst naar de wrapper class om aan te geven dat MyQueueItem naar een queue gestuurd moet worden. Bij een http output moeten we ervoor zorgen dat het return type HttpResponseData is. Op de class zelf hoeven we verder geen attributes te plaatsen. We kunnen nu deze class gebruiken binnen onze function. Wel moeten we het return type van onze function aanpassen naar onze nieuwe class MyResponseBindings. **Voorbeeld 9**.

```
[Function(name: "HelloWorld")]
public static MyResponseBindings Run([HttpTrigger(AuthorizationLevel.Function, params methods: "get")] HttpRequestData req,
    FunctionContext executionContext)
{
    return new MyResponseBindings
    {
        MyQueueItem = "Hello World",
        HttpResponseData = req.CreateResponse(HttpStatusCode.Accepted)
    };
}
```
9

Cold-starts

Als je gebruik maakt van een Azure Function op een consumption plan, krijg je te maken met cold starts. Dit komt door het feit dat je function in een sleep-mode gaat als er geen traffic is geweest binnen een bepaalde periode. Dit kan voor problemen zorgen afhankelijk van hoe de Azure Function gebruikt word.
Bij in-process functions worden de host en de function code in hetzelfde process gedraaid, omdat de host de function code laadt als class library. Als we over performance praten is dit een plus, aangezien er niet eerst een ander process gestart hoeft te worden om de function code in te draaien.
Bij Isolated functions is dit helaas verslechtert. Dit komt vooral door het feit dat je function code nu echt een los process is wat opgestart moet worden als de function getriggered wordt. Als je hier veel hinder door ondervindt zijn er een aantal zaken die je kunt doen om je function starttijd te verbeteren.

Als je echt geen last van cold-starts wilt hebben kun je het beste je function plan omzetten naar een premium plan. Dit kost natuurlijk wel iets extra's maar hierdoor kun je instellen hoeveel instanties er blijven draaien en dus niet gaan slapen.
Een andere improvement die ook door Microsoft zelf wordt aangeraden is om je function te draaien op Linux. Door je function op Linux te draaien heb je minder overhead van het OS. Dit versnelt dus de opstarttijd van je function.

Middleware

Een nieuwe feature die functions met isolated model bieden is middleware. Middleware kennen we al van ASP.NET, maar is nu ook beschikbaar binnen Azure Functions. Binnen isolated functions wordt eerst internal function middleware aangeroepen m.b.t. output binding, function execution en vervolgens onze custom middleware. Net als bij ASP.NET werkt dit als een pipeline en kun je dus meerdere custom middlewares achter elkaar chainen. **Voorbeeld 10**.
We kunnen middleware gebruiken om te loggen, te authentiseren of

BIO

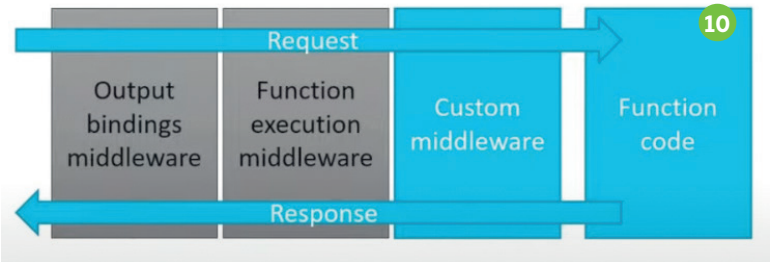
Florian Schaal



Florian Schaal is een Senior Cloud Developer met ruim 10 jaar ervaring in het vak. Zijn specialisaties liggen voornamelijk op Azure, .NET en Docker, maar hij heeft ook ervaring met verschillende front-end frameworks en mobile development met Flutter. Florian heeft een passie voor software development en verdiept zich graag in nieuwe technieken.

Momenteel is Florian werkzaam bij Cloud Republic, waar hij klanten en teams helpt software te ontwikkelen die toekomstbestendig en onderhoudbaar is. Dit doet hij vanuit verschillende rollen, van Team-Lead tot Senior Cloud Developer. Naast zijn werk bij klanten host Florian ook de Dev Talks podcast met collega Dibran Mulder. (devtalks.nl).

LinkedIn: https://www.linkedin.com/in/florian-schaal
Website: https://www.florianschaal.com/




```
public class HelloWorld
{
    [Function(name: "HelloWorld")]
    public async Task<HttpResponseData> Run([HttpTrigger(AuthorizationLevel.Function, params methods: "get")] HttpRequestData req,
        FunctionContext executionContext)
    {
        var logger = executionContext.GetLogger<HelloWorld>();
        logger.LogInformation(message: $"C# HTTP trigger function {executionContext.FunctionDefinition.Name} received request.");

        //logic..

        return req.CreateResponse(HttpStatusCode.OK);
    }
}
```
11

```
public class MyLoggerMiddleware : IFunctionsWorkerMiddleware
{
    public Task Invoke(FunctionContext context, FunctionExecutionDelegate next)
    {
        throw new System.NotImplementedException();
    }
}
```
12

```
public class MyLoggerMiddleware : IFunctionsWorkerMiddleware
{
    public async Task Invoke(FunctionContext context, FunctionExecutionDelegate next)
    {
        var logger = context.GetLogger<HelloWorld>();
        logger.LogInformation(message: $"C# HTTP trigger function {context.FunctionDefinition.Name} received request.");

        await next(context);
    }
}
```
13

om validaties uit te voeren. In het volgende voorbeeld gaan we een simpele middleware opzetten die de function naam logt zodra we een request ontvangen.
Als we dit binnen de function zelf doen zou dit er uit kunnen zien als in **voorbeeld 11**. Willen we deze log regel nu verplaatsen naar een middleware, dan moeten we deze eerst aanmaken. Hiervoor maken we een class die het interface IFunctionsWorkerMiddleware implementeert en daarmee ook de invoke methode die bij een inkomend request iets voor ons kan doen. **Voorbeeld 12**. Omdat we hier weer een FunctionContext binnen krijgen betekent dit dat we onze logger ook weer op dezelfde manier kunnen ophalen en deze kunnen laten loggen. Vervolgens moeten we nog next aanroepen en de context meegeven om ervoor de zorgen dat de pipeline doorgaat. **Voorbeeld 13**.
Als laatste stap voegen we nog onze middleware toe tijdens het configureren van de HostBuilder. **Voorbeeld 14**.
Als we onze function nu runnen, zien we dat voor elke request die binnenkomt via de middleware een log regel wordt aangemaakt. **Voorbeeld 15**. We kunnen dus nu als laatste de log regel uit onze function verwijderen.

```
public class Program
{
    public static void Main()
    {
        var host = new HostBuilder()
            .ConfigureFunctionsWorkerDefaults(builder =>
            {
                builder.UseMiddleware<MyLoggerMiddleware>();
            }) // IHostBuilder
            .Build();

        host.Run();
    }
}
```
14

```
[2022-05-20T12:32:48.105Z] Executing 'Functions.HelloWorld' (Reason='This function was programmatically called via the host APIs.', Id=3eb7f56b-e010-457b-aab9-fab81fbf7664)
[2022-05-20T12:32:48.567Z] C# HTTP trigger function HelloWorld received request.
[2022-05-20T12:32:48.568Z] Executed 'Functions.HelloWorld' (Succeeded, Id=3eb7f56b-e010-457b-aab9-fab81fbf7664, Duration=536ms)
```
15

Voor- en nadelen

Met isolated process wordt de volgende stap gezet in het verhaal van Azure functions. Als developers willen we vaak snel aan de slag met nieuwe technieken zonder een afweging te maken of de stap naar een nieuwe techniek wel echt nodig is. Het is dus altijd goed om even te kijken wat nu de grootste voor- en nadelen zijn en of deze relevant zijn voor ons.

Voordelen

> Volledige controle over function startup process;
> Dependency Injection zoals we dit gewend zijn binnen .NET;
> Middleware support;
> Geen dependency conflicts omdat zoals de naam het zegt het worker process isolated is van de host.

Nadelen

> Nog geen support voor Durable Functions;
> Nog niet alle bindings zijn beschikbaar;
> Meer last van cold-starts;
> Veel breaking changes en bestaande code werkt hierdoor niet zonder een rewrite.

Samenvatting

Met Azure Functions isolated process is een mooie stap gemaakt naar een architectuur die klaar is voor de toekomst. Ga je beginnen met een nieuwe function en hebben de nadelen geen effect op jouw process? Dan zou ik zeker voor isolated process kiezen. Heb je al een bestaande function? Dan is het op het moment nog niet nodig om deze om te gaan schrijven, tenzij er een sterke behoefte is voor middleware support. in-process wordt nog ondersteund tot 8 november 2024. ●

WIJ RAKEN NIET UITGEPRAAT OVER DEVELOPMENT

Daarom praten wij, Florian en Dibran, je in onze podcast Dev Talks bij over de nieuwste ontwikkelingen.

Altijd de laatste cloud updates in je inbox?

Schrijf je in voor Dev News!



@cloudrepublic.nl

Cloud Republic

cloud republic

Distributed Tracing met Application Insights

De meeste developers die wel eens iets met de Azure cloud doen, zijn wel bekend met Azure Application Insights (AI). AI is onderdeel van Azure Monitor, en is met minimale configuratie te gebruiken als sink voor de logregels vanuit je applicatie. Zeker voor applicaties die op een Azure App Service Plan draaien is dit heel gemakkelijk in te richten. Je zet een verwijzing naar Application Insights in je configuratie, hengelt de .NET Logging extensions naar binnen, en klaar!

Auteur: Annejan Barelds

Maar een bakje met alle logregels die een applicatie heeft gegenereerd, is tegenwoordig meestal niet meer genoeg om een probleem te kunnen analyseren. In een modern applicatielandschap worden functionele processen niet door één applicatie afgehandeld, maar door een set aan gedistribueerde services. Daarom is het belangrijk om, over al deze services heen, de diagnostische data met elkaar te kunnen correleren. In dit artikel gaan we onderzoeken hoe Application Insights ons ook daarbij kan helpen.

Het datamodel van Application Insights

Om goed te begrijpen wat Application Insights voor ons kan doen bij het correleren van (log)data, moeten we eerst even goed bekijken wat we eigenlijk nog meer wegschrijven naar Application Insights. Dat is namelijk niet beperkt tot logregels: ook onder andere inkomende requests en uitgaande calls (dependencies) komen terug in Application Insights. Daarnaast kennen we een operation in Application Insights. Dit is geen item type, maar de lijm die alle items (requests, traces, dependencies) bij elkaar houdt. Een operation begint met een request, en wordt gerepresenteerd door het operation_id attribuut. En alle traces en dependencies die binnen de scope van dit request gegenereerd worden, krijgen dezelfde operation_id. **Voorbeeld 1**.
Naast de logregels die je zelf in je code invoegt, worden deze verschillende items vooral gegenereerd door de ingebakken intrumentatie in .NET runtimes, frameworks en SDK's. Voor andere platformen en frameworks zal dat uiteraard anders zijn.

Operations en parents

Met dit datamodel in het achterhoofd, zoeken we dus eigenlijk naar een manier om de operations in een downstream service te relateren aan operations in een upstream service. In de meest eenvoudige gevallen is dit een één-op-één relatie. Wanneer we als voorbeeld communicatie over HTTP nemen, dan hebben we at runtime een client die één call (dependency) uitvoert naar een HTTP endpoint, wat op dat endpoint leidt tot één request.
In Application Insight wordt zo'n HTTP call gemodelleerd als een parent-child relatie. We hebben aan de kant van de client een request (bijvoorbeeld omdat een browser een GET doet op een pagina), met een operation_id. De call naar de upstream service is een dependency met hetzelfde operation_id, én met een verwijzing naar het bovenliggende request: de operation_parentid van de dependency heeft dezelfde waarde als de id van het request. Aan de kant van het HTTP endpoint leidt deze call tot een nieuw request, met óók weer dezelfde operation_id. En in dit geval is de operation_parentid op het request een verwijzing naar de id van de dependency. **Voorbeeld 2**.
Dit model werkt trouwens ook over verschillende Application Insights-instanties heen. De benodigde informatie voor gedistribueerde tracing wordt in het geval van HTTP-communicatie uitgewisseld door middel van headers, dus een enkele operatie kan prima gedistribueerd zijn over meerdere Application Insights-instanties. Voor het uitvoeren van een query die alle gerelateerde telemetry van een operatie ophaalt, is het mogelijk om over meerdere instanties heen te queryen. Sterker nog: als je gebruikt maakt van de Transaction Search in de Azure portal, zal de UI zélf eventuele andere instanties uit dezelfde subscription erbij opzoeken om alle telemetry bij een operatie op te halen.

Van parents naar links

Bovenstaand model werkt heel mooi voor HTTP, en eigenlijk voor alle

itemType	message	operation_Id
request		f071e8a2579e479ea98bec4b510a6939
trace	Executing 'Sink' (Reason='This function was programmatically called via the host...	f071e8a2579e479ea98bec4b510a6939
trace	C# HTTP trigger function is processing a request...	f071e8a2579e479ea98bec4b510a6939
dependency		f071e8a2579e479ea98bec4b510a6939
trace	C# HTTP trigger function processed a request.	f071e8a2579e479ea98bec4b510a6939
trace	Executed 'Sink' (Succeeded, Id=182008e6-890c-4162-b332-3436fac969b5, Durat...	f071e8a2579e479ea98bec4b510a6939
trace	{"requestId":"0cef8eec-2aaf-4292-8d22-bb415f07c535","language":"DotNetAsse...	f071e8a2579e479ea98bec4b510a6939

1

itemType	role	operation_Id	id	operation_ParentId
request	client	f071e8a2579e479ea98bec4b510a6939	6624469b1953b4c5	f071e8a2579e479ea98bec4b510a6939
dependency	client	f071e8a2579e479ea98bec4b510a6939	e46ef8221e26fa9f	6624469b1953b4c5
request	endpoint	f071e8a2579e479ea98bec4b510a6939	9dca37c11bb19d3b	e46ef8221e26fa9f

2

gevallen waarin we een één-op-één relatie hebben tussen downstream en upstream services die participeren in een operatie. Maar het is niet altijd zo simpel. Denk bijvoorbeeld aan architecturen waarin event streaming een rol speelt. Dan kun je enorme aantallen operaties hebben die allemaal een event inschieten, en verderop in de keten leeft een proces die deze events in batch verwerkt. Voor een batchproces dat 200 events per keer afhandelt, zijn er dan 200 parent operations. En slechts één operation_parentid veld. Met andere woorden: het parent-model werkt niet voor complexere ketens.

Meer en meer SDK's gaat daarom over op een ander model, waarin een request een array van links kan bevatten naar andere operations.

De nieuwe Service Bus SDK bijvoorbeeld werkt op die manier. Stel je een workload voor, bijvoorbeeld een API, die als onderdeel van een request (bijvoorbeeld een POST) een bericht op een queue moet zetten. Op deze queue luistert een Azure Function, die iets zinnigs met dat bericht moet doen. Als zowel de producer van het bericht (de API) als de consumer (de Function) naar Application Insights loggen, dan zou dat er uit kunnen zien als onderstaande afbeelding: **Voorbeeld 3**.

Je ziet hier dat de consumer van het bericht niet langer dezelfde operation_id heeft als de producer. Ook de operation_parentid verwijst nu naar zichzelf, en niet meer naar een bovenliggende dependency. Wat daarvoor in de plaats is gekomen, is een array _MS.links in de customDimensions. Hierin staat, in dit geval, één object met daarin verwijzingen naar de bovenliggende operation_id en de bovenliggende dependency. Maar het is een array, dus daar kunnen meerdere objecten in terecht komen. En dat zou het geval zijn bij een Event Hub triggered Function, die in batch meerdere events op een Hub verwerkt, die allemaal afkomstig zijn van een andere overkoepelde operatie.

En ja, ook deze nieuwe manier om gedistribueerd te correleren, wordt volledig ondersteund door de Application Insights Transaction Search, en ook hier kunnen de verschillende participanten in het proces gekoppeld zijn aan verschillende Application Insights resources.

Zelf maken

Zoals besproken zijn een flink aantal SDK's voorzien van out-of-the-box gedistribueerde tracing. We bespraken al hoe dit zit voor HTTP calls, en voor Azure Service Bus. Event Hub hebben we niet gezien, maar wel genoemd: ook hiervoor is dit standaardfunctionaliteit. Maar toch komt het soms voor dat je zou willen correleren over (Azure) services heen, waarbij je daar nog wat extra werk voor moet doen. Dat is bijvoorbeeld het geval bij Azure Cosmos DB.

Denk bijvoorbeeld aan een use case waarbij je Cosmos DB zou willen gebruiken als Event Store in een event-sourced systeem. Je zou dan de Cosmos DB Change Feed willen gebruiken voor event listeners om op events te kunnen reageren. En dan zou het ook mooi zijn als het proces dat leidt tot een event (de producer), te correleren is aan de processen die gaan lopen als gevolg van dat event (de listeners).

Om dit voor elkaar te krijgen moeten we er in de basis voor zorgen dat de diagnostische identifiers van het proces van de producer terechtkomen bij de consumer, zodat deze ze in het gelogde request kan toevoegen aan de

itemType	role	operation_Id	id	operation_ParentId	customDimensions_MS.links
request	producer	ddcc62a12574a5d5d1e2500cad0134fc	5f6766615a8cde3a	ddcc62a12574a5d5d1e2500cad0134fc	
dependency	producer	ddcc62a12574a5d5d1e2500cad0134fc	8b6dc7f27106d780	5f6766615a8cde3a	
request	consumer	5c9525e9a576d538a88d089f8da8ba34	c232bc0149639462	5c9525e9a576d538a88d089f8da8ba34	[{"operation_Id":"ddcc62a12574a5d5d1e2500cad0134fc","id":"8b6dc7f27106d780"}]

3

array met links. In het geval van HTTP gaat dit door middel van request headers, en in het geval van Service Bus door middel van een property op het bericht. In het geval van Cosmos DB hebben we echter niet zo heel veel andere keus dan om dit als JSON property in de body van het document (ofwel het event, in onze event sourcing use case) op te slaan: er is immers verder geen metadata die van producer naar listener gaat, en die we zouden kunnen verrijken.
Dus laten we in elk event een property DiagnosticId opnemen. Hierin nemen we de Id op van de huidige Activity: **Voorbeeld 4**.

```
public class SomeEvent
{
    public string DiagnosticId
    {
        get
        {
            return Activity.Current?.Id;
        }
    }
}
```

4

```
        private void SetLinks(IEnumerable<SomeEvent> events)
        {
            var links = events.Select(BuildLink).Where(link => link != default(Ope-
rationLink));
            Activity.Current?.SetTag("_MS.links", JsonSerializer.Serialize(links));
        }

        private OperationLink BuildLink(SomeEvent @event)
        {
            var parts = @event.DiagnosticId?.Split('-');
            if (parts != null && parts.Length == 4)
            {
                return new OperationLink(parts[1], parts[2]);
            }
            return default(OperationLink);
        }

        private record struct OperationLink(string operation_Id, string id);
```

5

BIO

Annejan Barelds

Annejan werkt als developer, architect en cloud consultant bij 4Dotnet, en in die rol helpt hij klanten om goede software te ontwikkelen en optimaal gebruik te maken van de cloud. Hij werkt nu ruim 15 jaar als developer. Zo'n 10 jaar geleden kwam hij in aanraking met Azure, en het enthousiasme over die grote bouwblokkendoos is sindsdien niet meer weg geweest. Hij houdt van het maken van technische oplossingen, maar ook van het delen van zijn kennis en ervaring met anderen.



Deze Id is als volgt geformatteerd: 00-0af7651916cd43dd8448eb. De 4 delen van deze string hebben elk een betekenis, waarbij deel 2 en 3 voor ons relevant zijn. Dit zijn namelijk de operation_id (deel 2) en de id (deel 3) van de huidige Activity.
Als we dit event opslaan in Cosmos, hebben we effectief de juiste diagnostische informatie ter beschikking gesteld aan de listener. De listener hoeft deze string dan alleen nog maar om te zetten in het juiste format en aan de _MS.links array toe te voegen. Verder hebben we nog wat code nodig om ervoor te zorgen dat we om kunnen gaan met meerdere events die in batch door de listener worden ontvangen: **Voorbeeld 5**.
Wat we doen is, voor elk event, de DiagnosticId opslitsen, waarbij we geïnteresseerd zijn in de eerder genoemde operation_id en id. Deze zetten we in een record struct en vervolgens in een array, die we gebruiken als waarde bij de key _MS.links; het geheel voegen we als tag toe aan de huidige Activity. En het framework doet de rest, wat in dit geval betekent dat deze array van links toegevoegd wordt aan de customDimensions van het request in Application Insights: Met andere woorden: we hebben nu zelf een relatie tot stand gebracht tussen 2 processen, en we hebben dat gedaan op een manier die correct geïnterpreteerd kan worden door bijvoorbeeld de Transaction Search in Application Insights.

Slot

Er is uiteraard nog veel meer uit te zoeken aan gedistribueerde tracing in Application Insights. We hebben bijvoorbeeld weinig aandacht besteed aan hoe de relatie tussen Application Insights en de instrumentatie door middel van System.Diagnostics nu eigenlijk precies zit. Ook hebben we niet stilgestaan bij de beweging naar OpenTelemetry, wat gericht is op een open en interoperabel ecosysteem van instrumentatie en monitoring. Maar die details hoeven we ook niet te kennen, om toch gebruik te kunnen maken van de opties voor gedistribueerde tracing die Application Insights out-of-the-box biedt. En zelfs het aanhaken van je eigen spulletjes op dit model vergt niet meer dan een paar regels code.
Dus vanaf nu weten we: gedistribueerde tracing hoeft niet complex te zijn, en debugging in een gedistribueerd landschap hoeft niet moeilijker te zijn dan in een monoliet. Happy tracing! ●

OWASP 2022: **de praktijk**

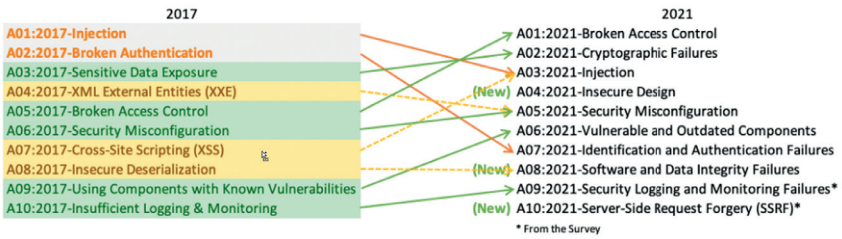
Auteurs: Patrick Bes en Arjen Kraak

Begin dit jaar is voor het eerst in lange tijd een nieuwe versie van de OWASP top 10 uitgebracht. Cybersecurity behoort een vast agendapunt te zijn voor elke ontwikkelaar en organisaties die betrokken zijn bij het ontwikkelen en onderhouden van software. Het adresseren van bedreigingen is niet alleen een technisch onderwerp maar ook een organisatorisch. In dit artikel delen we onze inzichten op beide onderwerpen.

De technische aspecten

De OWASP foundation is een non-profit organisatie met als doelstelling de veiligheid van software te vergroten. Ze hebben naam en faam gekregen door het samenstellen van de OWASP top 10. Hierin publiceren zij het overzicht van de 10 meest voorkomende kwetsbaarheden. Dit doen ze onafhankelijk van de gebruikte programmeeromgeving waardoor het overzicht breed toepasbaar is voor alle software ontwikkelingsprojecten. Voor .NET is een cheatsheet beschikbaar om je snel op weg te helpen: https://cheatsheetseries.owasp.org/cheatsheets/DotNet_Security_Cheat_Sheet.html. De OWASP top 10 geeft een concrete lijst met risico's waarmee je je software project controleert. Dit is iets wat gedurende het gehele ontwikkeltraject in de achterhoofden van ontwikkelaars moet zitten.





De nieuwe top 10

In 2021 hebben ze een update gegeven op de lijst zoals deze beschikbaar was vanaf 2017. Deze bevat een aantal wijzigingen. De lijst is het resultaat van uitgebreid onderzoek dat heeft plaats gevonden. Hierin wordt onder andere gemeten hoe vaak bepaalde kwetsbaarheden voorkomen en wat daarvan de impact is. Op basis van deze getallen is de volgorde binnen de top 10 vastgesteld. Wij zullen hier de belangrijkste aanpassingen en toevoegingen op de top 10 beschrijven.



Broken Access Control is nieuw binnengekomen op nummer 1 van plek 5 in 2017. Dit is een vrij algemene kwetsbaarheid en komt in de praktijk neer op gebruikers die zich meer toegang weten te verschaffen dan de gezette restricties zouden toestaan. Een concreet voorbeeld van deze kwetsbaarheid is het datalek bij Facebook in 2018. Men kon hier de "View As" functie gebruiken om een profiel te bekijken als iemand anders. De video upload functie werd dan sporadisch getoond wat niet de bedoeling was. Indien dit het geval was kreeg men een nieuw access token van de persoon waarmee men keek. Met dit token kon men vervolgens inloggen op het betreffende account. Dit komt dus grofweg neer op het misbruiken van bestaande functionaliteiten die op het eerste gezicht een logisch doel dienden. Het uitvoeren van security integratie testen en threat-modeling sessies kunnen in dergelijke gevallen wellicht een maatregel bieden om risico's in kaart te brengen en te bewaken.

Cryptographic Failures deze kwetsbaarheid is 1 positie gestegen t.o.v. van 2017. Ook dit een vrij algemene kwetsbaarheid gerelateerd aan cryptografie (of het gebrek hieraan). Voorbeelden hiervan zijn b.v. het gebruik van verouderde algoritmes zoals: MD5, SHA1, DES. Zo is cryptografische hash methodiek van SHA-1 in 2017 al succesvol gehackt en sindsdien al niet meer bruikbaar. Helaas zijn er nog veel legacy applicaties in omloop waar deze algoritmes in zitten. De noodzaak om legacy applicaties aan te passen is hoog, maar niet altijd in het vizier. Een goede scan van de huidige code base op het gebruik van deze verouderde algoritmes is hier zeker een aanrader.



Injection is de kwetsbaarheid die jarenlang op nummer 1 heeft gestaan. Het is goed nieuws dat men hier bewuster van is geworden

waardoor deze iets op de lijst is gezakt. Daarnaast is het een combinatie geworden van de oude vertrouwde SQL injection en de Cross-Site Scripting kwetsbaarheid. Het gebrek aan het valideren van invoer is vaak een oorzaak. Na al deze jaren boven aan de top 10 te hebben gestaan zijn er toch nog concrete voorbeelden zoals b.v. de Fortnite hack uit 2019 was een combinatie van Cross-Site scripting en SQL-Injection. In je code base is het zaak rekening te houden met potentiële SQL Injection risico's. Dus let op bij het dynamisch samenstellen van queries, doe aan invoer validatie, implementeer het least priviledge principe. Er zijn legio mogelijkheden om deze kwetsbaarheden te mitigeren en een scan op een legacy code base op dergelijke issue is altijd een aanrader.



Alle kwetsbaarheden die zijn opgenomen in de Top 10 zijn van groot belang. Een aantal hiervan staan al meerdere jaren vast en staan wellicht wel helder op het netvlies. Maar een aantal zijn volledig nieuw op de lijst en verdienen daarom wat extra aandacht.

Insecure Design legt de focus veel meer op de kwetsbaarheden die ontstaan door design en architectuur fouten. Deze kwetsbaarheid vraagt meer om oplossingen in de proces sfeer in plaats van technologische oplossingen. Denk hier b.v. aan Threat Modeling en secure design patterns, verder in dit artikel komen wij hierop terug. Voorbeelden van insecure design zijn scenario's die op de tekentafel een goed idee lijken, maar waar in praktijk misbruik van kan worden gemaakt. Zo klinkt het voor een webshop vrij logisch om zoveel mogelijk klanten te kunnen helpen in zo kort mogelijke tijd. Maar in de praktijk kan dit betekenen dat kwaadwillenden de hele voorraad kunnen opkopen middels geautomatiseerde scripts. Dit is b.v. in praktijk voorgekomen bij de introductie van de Playstation 5. Aangezien dit niet te herleiden is naar een technische onvolkomenheid maar misbruik is van bewust geïmplementeerde features kan het dit lastig zijn om deze vroeg te herkennen. Door als team in de huid te kruipen van de kwaadwillende tijdens een threat modeling sessie kun je het wellicht wel herkennen als risico.

Software and Data Integrity Failures is een categorie die gaat over het doen van aannames gerelateerd aan software updates zonder hiervan de bron te controleren. In de loop van de tijd zijn we steeds meer afhankelijk geworden van externe code die afkomstig is van verschillende bronnen. Denk bijvoorbeeld aan NPM packages, Nuget packages, Docker images enz. Deze packages kunnen allemaal bronnen zijn van potentiële kwetsbaarheden die wij introduceren in onze code. Dit betreft dan supply chain hacks zoals wij hier al beschreven. Een voorbeeld hiervan is de hack van Asus laptop update service in 2019. Hierbij kregen gebruikers een update op hun laptop geïnstalleerd waarbij een kwetsbaarheid werd geïntroduceerd. Inmiddels zijn er legio producten op de markt die dit soort risico voor een groot gedeelte kunnen mitigeren in je build/release pipelines. (b.v. OWASP dependency check, Whitesource, Snyk, Prisma cloud, Sonatype Nexus en nog vele anderen) Nog veel organisaties hebben het gebruik van deze tooling helaas niet afdwongen. Dit is een aandachtspunt wat met relatief weinig effort te verhelpen is.

Server-Side Request Forgery is een nieuwe kwetsbaarheid die gaat over het ophalen van externe resources zonder dat de URL die wordt aangeleverd wordt gevalideerd. Een eenvoudig voorbeeld van deze kwetsbaarheid is bijvoorbeeld het aanleveren van een URL als file:/// etc/passwd waardoor er paswoorden gelekt kunnen worden in het resultaat. Andere scenario's zijn bijvoorbeeld het kunnen scannen van interne poorten om zo een map van een intern netwerk te kunnen maken. Dit soort kwetsbaarheden zijn wat lastiger te vinden. DAST tools (zoals b.v. Tenable.io of Nessus) kunnen je hier wel bij helpen om hier (evt. geautomatiseerd) risico's af te dekken.

Dit is maar een korte lijst van de belangrijkste verschillen maar neem vooral de tijd om goed te kijken naar alle kwetsbaarheden die in de top 10 zijn opgenomen. Daarnaast worden er op de site van OWASP.org nog meer oplossingsrichtingen aangedragen welke je in de juiste richten kunnen wijzen.

OWASP voor teams en organisaties

Het adresseren van bedreigingen kan niet zonder het borgen van maatregelen in de werkprocessen. We beschrijven de methodieken die kunnen worden gebruikt bij de waardeketen van softwareontwikkeling en onderhoud.

De maatregelen worden genomen met verschillende invalshoeken:

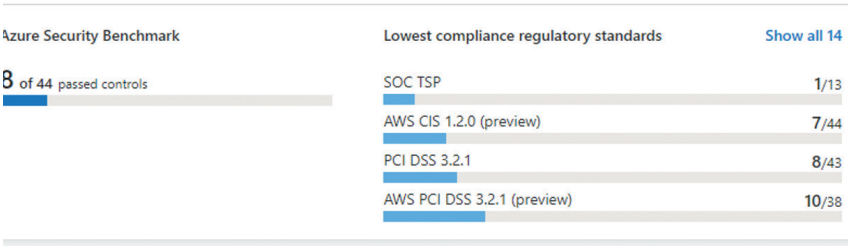
- Kijken naar de toekomst; wat kan gebeuren?
- Kijken naar de actualiteit; welke impact maken we met huidige inspanningen?
- Kijken naar de actualiteit; wat gebeurt er op dit moment in het landschap?

Wat zou er kunnen gebeuren?

Het beperken van beveiligingsrisico's begint al tijdens het nadenken over vernieuwing en onderhoud. Het beleid van 'security by design' is hier van toepassing. Met deze invalshoek kijken de bedenkers van oplossingen (ontwikkelaars) en de gebruikers (klant en stakeholders) samen naar mogelijke bedreigingen.

Dit kan worden gedaan tijdens een plenaire 'Threat modeling' sessie.

Threat Modeling is een methodiek waarbij men bedreigingen in kaart brengt op basis van scenario's en persona's. Eerder is hierover een artikel geschreven.
In het kader van zo vroeg mogelijk inzicht krijgen in de mogelijke kwetsbaarheden is het aan te raden om een security self-assessment uit te voeren. Een dergelijk assessment geeft je veel handvatten om volledig beeld te krijgen van volwassenheid op security gebied. Dit geldt voor organisatorisch tot technisch. Het SAMM assessment van OWASP is hier een goed voorbeeld van SAMM Assessment (owaspsamm.org)
Cloud diensten bieden geautomatiseerde assessments aan tegen gangbare standaarden. Zo biedt Azure in Microsoft Defender for Cloud controles ten opzichte van: ISO 27001, HIPAA. Per standaard wordt een score gegeven:
Microsoft Defender for Cloud geeft direct aanbevelingen voor aanpassingen in de eigen infrastructuur om de score ten opzichte van standaarden te verbeteren. De beheerder wordt direct doorgeleid naar de resources waarop de aanpassingen van toepassing zijn.



Welke impact maken we met de huidige inspanningen?

Als eigenaar van broncode is de organisatie verantwoordelijk voor de doorontwikkeling waarbij het 'security by design' principe als uitgangspunt wordt gehanteerd. Hiervoor worden de volgende middelen ingezet.
Borg beveiligingsoverwegingen in de 'Definition of Done'; tijdens het schrijven van software geeft deze DoD een beleid over wanneer een stuk ontwikkelwerk als gereed kan worden beschouwd. De 'DoD' wordt tijdens discussie over de oplossing en taakverdeling gebruikt om de taken voor beveiliging te adresseren. Op deze manier, worden de maatregelen onderdeel van het reguliere ontwikkelwerk. Voorbeelden van 'checkpoints' zijn:

- Wordt bij het uitvoeren van T-sql in code functie parameters gebruikt?
- Worden foutmeldingen en stacktraces niet aan de gebruiker getoond?
- Zijn controllers (MVC) voorzien van een Authorize attribuut of afgeleide ervan?

Pas het principe van 'fail fast, fail often' toe; dit principe heeft als doel om zo snel mogelijk inzicht te krijgen van de gevolgen van aanpassingen aan code. Dit kan worden gedaan door het toepassen van de volgende:

- Beleid voor het verwerken van pull-requests (pull-request policies)
- Verzekering van herleidbaarheid naar de Product Backlog door middel van links
- Uitvoeren van een 'pre-merge' build
- Automatische scans van de samengevoegde code door gereedschappen als SonarQube en WhiteSource. SonarQube kijkt vooral naar de code die de ontwikkelaar zelf schrijft. WhiteSource kijkt naar code die wordt hergebruikt van anderen (packages)

Wat gebeurt er op dit moment?
Het is cruciaal ook de huidige omgevingen te blijven bewaken. Deze bewaking kan worden uitgevoerd door het gebruiken van technische oplossing als bijvoorbeeld Azure Security Center of Azure Sentinal. Deze oplossingen bieden 'high-end' oplossingen voor het beveiligen van de cloud infrastructuur.
Tegenwoordig hebben veel organisaties een 'hybride' cloud afname. Hierbij bestaat een infrastructuur in eigen beheer naast een oplossing bij een cloud provider als Microsoft of Amazon. Hybride of niet, het is gebruikelijk om het platform bloot te stellen aan een 'pentest'. Deze test, vaak uitgevoerd door een onafhankelijke partner, onderzoekt het platform als een hacker. De bevindingen hieruit worden geïntegreerd in het ontwikkelproces zoals dat eerder is beschreven. Op deze manier is de gehele waardeketen gedekt.



‘Een goede QA Engineer **heeft een holistische blik’**

Als QA Engineer wordt Kitty van Leeuwen graag zo vroeg mogelijk in een proces betrokken. Haar visie: hoe meer je geautomatiseerd test, hoe meer ruimte er overblijft voor creatief handmatig testen.

Auteur: Kitty van Leeuwen

Toen Kitty van Leeuwen in februari 2018 bij Ordina begon binnen de AUTO|Q unit deed ze dat met het idee om op termijn door te stromen naar een andere business unit. Vier jaar later zit ze nog altijd bij dit team van bevlogen testprofessionals. ‘Dat komt doordat ik nog altijd veel uitdaging heb,’ vertelt de QA Engineer, die kunstgeschiedenis studeerde en een carrière in de kunstwereld begon, maar uiteindelijk koos voor de IT. ‘Tijdens mijn traineeship bleek testing veel technischer dan ik van tevoren had gedacht. Dat vond ik leuk. Ons vakgebied is ontzettend in beweging. Bij Ordina volgen we alle ontwikkelingen op de voet en behoren we tot de top van Nederland op het gebied van QA. Dat maakt het ontzettend interessant.’

Q-denken

De Q in AUTO|Q staat voor quality. Alle testers en QA Engineers binnen dit team hebben een sterke focus op kwaliteit. Een mindset die intern het Q-denken wordt genoemd. In haar functie als QA (Quality Assurance) Engineer zorgt Kitty ervoor dat die kwaliteitsfocus gedurende het hele proces aanwezig is. ‘Een goede QA Engineer kijkt met een holistische blik naar het ontwikkelproces,’ zegt Kitty. ‘Een QA Engineer is een generalist. Je moet bijvoorbeeld kennis hebben van automated testing, monitoring en performance. Maar daarnaast ook een agile development proces als behaviour driven development kunnen aandragen. Eigenschappen als nieuwsgierigheid en leergierigheid zijn belangrijk en je moet een kritische blik hebben, niet te snel tevreden zijn. Alleen zo houd je de kwaliteit hoog.’

Beter voorkomen dan genezen

Kitty ziet dat veel opdrachtgevers QA steeds serieuzer nemen. ‘Een paar jaar geleden werden test-gerelateerde zaken regelmatig onderschat. Het was een van de eerste posten waarop werd bezuinigd. Testers werden vaak pas aan het einde van het project ingezet om nog wat bugs eruit te halen. Gelukkig zien steeds meer opdrachtgevers in dat je bugs beter kunt voorkomen dan oplossen. Kwaliteit inbedden op een vroeg moment in het proces loont. Daarom worden testers en QA Engineers steeds eerder betrokken. Niet meer als losse schakels, zoals voorheen, maar als volwaardig onderdeel van een team. Dat vind ik prettig. In mijn functie kan ik meedenken over de requirements en zo vooraf zorgen dat we het juiste gaan bouwen. Dat helpt ook bij het schrijven van geautomatiseerde tests. Het zorgt ervoor dat ontwikkelaars en testers meer dezelfde taal spreken, dat komt het eindresultaat ten goede.’

Tijdbesparing

Testautomatisering is de laatste jaren aan een enorme opmars bezig. ‘Waarom zou je een persoon telkens op een button laten klikken als dit ook geautomatiseerd kan?’, stelt Kitty een retorische vraag. ‘Daarmee bespaar je veel tijd en geld. Bovendien verkleint het de kans op fouten, want handmatig testen is mensenwerk. De tijd die je overhoudt, kan ingevuld worden door op een creatievere manier handmatig te testen, een techniek als Exploratory Testing kan daarbij helpen. Daarmee vind je dingen die je anders niet had gevonden, omdat je alleen maar bezig was met die repetitieve checks.

Alfa meets bèta

Buiten haar dagelijkse werk investeert Kitty ook veel tijd in het up-to-date blijven van alle ontwikkelingen op het gebied van automated testing. ‘Ik volg

Liever bugs voorkomen dan bugs oplossen



trainingen, maar vind het ook leuk om me zelfstandig nieuwe dingen eigen te maken. Zoals nieuwe frameworks.' Maar ze doet nog veel meer. 'Binnen AUTO|Q ben ik ook bezig met het onderwerp technologie en ethiek. Daaruit ontstaan blogs, talks en meet-ups. Met andere collega's werk ik aan een nieuw onboarding-traject voor nieuwe collega's. Een traject waarin zij nog beter kunnen aangeven wat hun ambities zijn en wat ze nodig hebben om daar te komen. Tegelijkertijd volg ik een opleiding aan de Schrijversvak-school, om ook de alfa in mij levend te houden. En wie weet verschijnt er ooit een boek over IT'ers dat niet op waarheid berust 😉. Ik waardeer dat Ordina mij de ruimte geeft om dat soort dingen te doen.' ●

BIO

Kitty van Leeuwen

Kitty van Leeuwen is QA consultant en Codesmith bij Ordina. Ze is gefascineerd door alles wat nieuw is in de wereld van web development en automatisering. Na haar specialisatie in de discipline van software testing, heeft ze zich ontwikkelt als pleitbezorger voor een kwaliteit- en DevOps mindset in teams




FUTURE TECH 2022
THE IT CONFERENCE FOR MICROSOFT TECHNOLOGIES
21st June 2022, Jaarbeurs Utrecht
Tickets: www.futuretech.nl
Are you member of the SDN? Get your free ticket with code: SDNMember

Beste SDN-ers en FutureTech bezoeker

Wij zijn verheugd jullie weer in real life te mogen verwelkomen op ons FutureTech event IRL van Reshift Digital Stichting DotNed, en SDN! Eindelijk kunnen we weer events organiseren om mooie kennisdeel momenten te faciliteren en contacten met collega's mogelijk te maken met koffie en iets lekkers. Hoewel onze digitale uitvoeringen een hoge opkomst hadden en de content even hoogstaand. Er gaat niets boven samen in een ruimte op een sponsorplein te discussiëren over een zojuist gehoorde sessie of kennis te maken met een collega uit een andere industrie.

De afgelopen twee jaar hebben ons wel duidelijk gemaakt dat IT belangrijk is en dat de ontwikkelingen nog sneller gaan. Waar wij als IT-ers al vaak online vergaderingen deden, moest ineens de hele wereld hierin mee. Dat betekent wel iets voor je software en architectuur. Simpel weg dingen in een Cloud zetten is niet voldoende, je code moet goed getest en aan goed architectuur voldoen. En de ontwikkelingen in de taal C#, Clouds en architectuur gaan snel. Waar we in 2008 nog maar 4 services kenden in Microsoft Azure, kennen we er nu boven de 100 services in een relatief korte tijd en met een behoorlijke snelheid. Monolieten zijn niet schaalbaar en service orientatie is de toekomst, maar dan moet je het wel op de juiste manier doen en je teams moeten ook nog bij kunnen houden. Al deze ingrediënten zitten in ons event!

Dit nieuwe FutureTech event heeft een mooi en vol programma, ondersteund door een veelheid aan sponsoren zonder hen was het niet mogelijk geweest. Met vele mooie sprekers uit binnen- en buitenland over een veelheid aan onderwerpen, die direct toepasbaar zijn voor jullie, Architecten en Developers. Ook is er een superleuke workshop over 'SOA done right', perfect om real life en handson bezig te zijn met Microservices. Vergeet de keynotes ook niet van Jonas Johansen en Scott Hunter. Daarnaast staat de beursvloer vol met partners van de SDN en Future Tech, op hun mooie stands maken zij graag een praatje met je en wie weet win jij wel een van de vele mooie prijzen die er die dag gewonnen kunnen worden, of doe jij gewoon mee met de leuke spelletjes die er op de stands te spelen zijn. Maar een tas vol met toffe verzamelde goodies doet het natuurlijk ook al goed.

Wij (Reshift Digital, DotNed, SDN samen FutureTech) wensen jullie allemaal een superfijne conferentie toe! Laat je feedback achter zodat we volgend jaar nog mooier en beter kunnen zijn!

Veel plezier, geniet van de sessies en de artikelen en wij zien jullie graag volgend jaar weer!

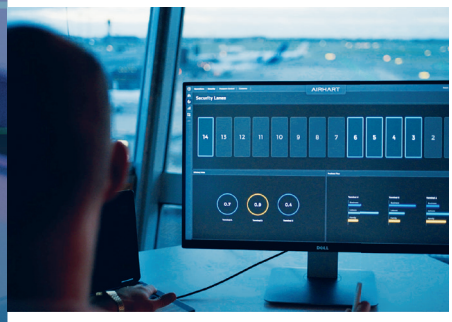
Groeten,

Marcel Meijer
Voorzitter SDN (Software Development Network)
Voorzitter FutureTech



Make a **difference**

At Netcompany we believe that our people can make a real difference to our societies, businesses and the environment. Our culture is open and driven by collaboration. Our flat organizational structure allows for greater agility and a pragmatic approach focused on delivering optimal results.



Technology is at the centre of everything we do at Netcompany. What guides us is a core belief that we can use it to help make a real difference to our common future. We challenge old business models providing new disruptive services. We do this for more than hundreds of clients in the private and public sector. Read more about our projects and the value they create in society on our website: **www.netcompany.com**.

We are Netcompany

Netcompany was founded in 2000 and has its headquarters in Copenhagen, Denmark. Today, we are an international company with more than 6,500 employees working from 17 countries. We serve a wide array of customers in in their digital transformation journeys across Europe. In the Netherlands we are located in Delft with clients such as Studielink, Rijkswaterstaat and the Ministry of Foreign Affairs.
Netcompany is a pure-play IT services company delivering business-critical strategic IT projects. Our goal is to accelerate customers' digital transformation through digital platforms, core systems and infrastructure services.

Our values

It takes talent to apply the opportunities technology gives us at Netcompany and Netcompany invests in talent like no one else. Today we are more than 6000 people. Every Netcompany project is led by project managers with strong IT backgrounds, who have the experience to guide the team and to ensure quality in all deliverables while knowing what responsibility to delegate.
Our ambitions for the future are big: we aim to be the leading digital challenger in Northern Europe.

Meet Netcompany at Future Tech & win a bike

Want to know more? We are eager to meet you and share more about what we do and how you can make a difference. Don't forget to sign up at our booth to win an original Netcompany Veloretti bike.

Join our keynote: AIRHART – The future of Airport Management

How modern architecture and tech can be utilised to solve complex business requirements.
Meet AIRHART - an airport management platform that enables airports and their stakeholders to work and become smarter – together. AIRHART facilitates an efficient eco-system for optimizing operations, passenger experience, commercial excellence and has the ability to adapt to new needs – faster. It is modular and configurable and can be tailored to individual airports along with being a continuous learning system based on trustworthy data and algorithms. AIRHART is implemented in .NET and Angular, hosted in Azure Cloud utilizing Kubernetes and Kafka to orchestrate application and information flow.
Come and listen to the story of our Danish colleagues Jonas Johansen and Morten Ege Jensen, who both worked on this unique project, and will demonstrate how a composable platform like this, can be implemented in a highly complex business environment and will share their learnings about this exciting project.

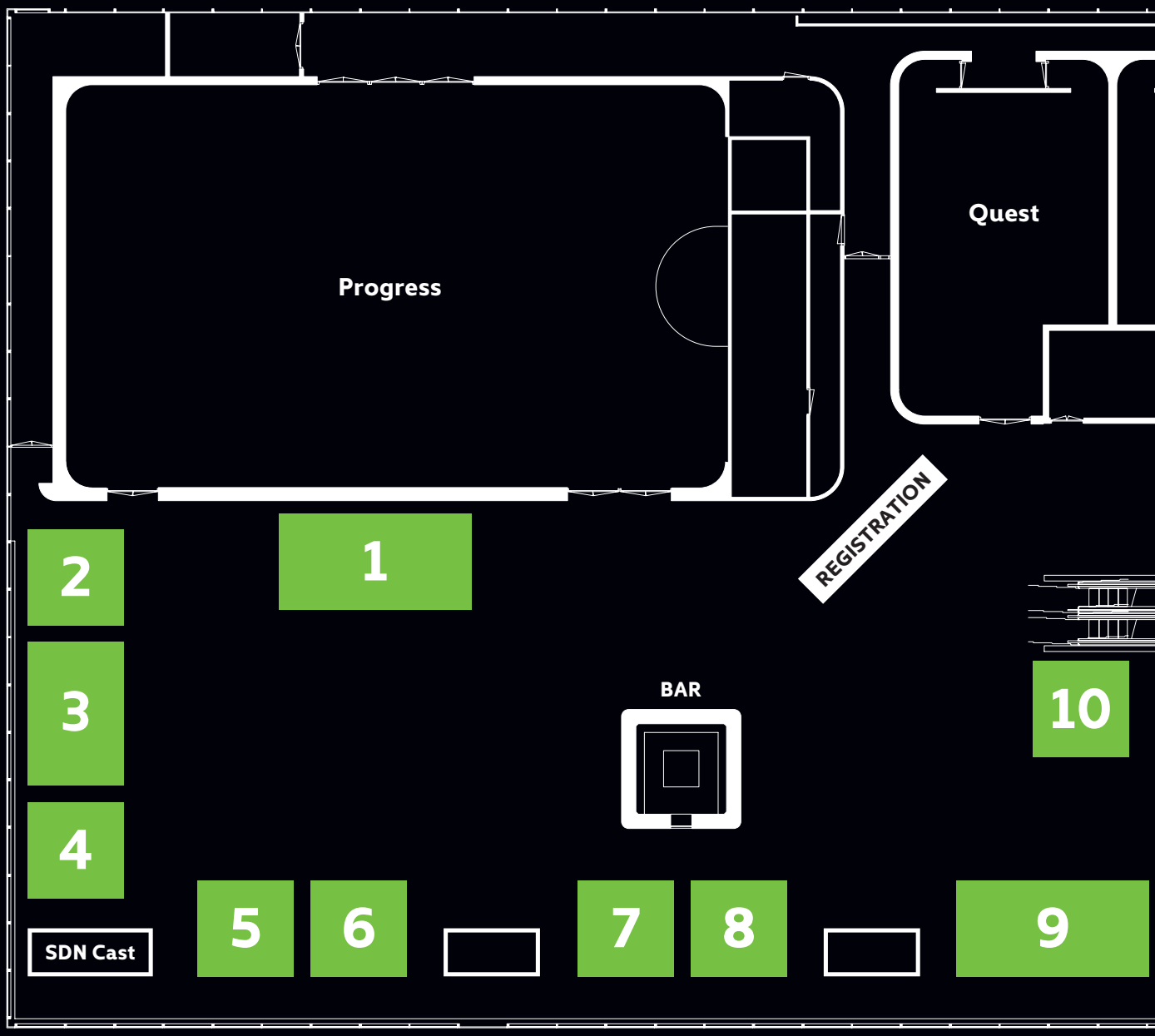




netcompany

Location: Progress
Time: 10:10 am - 10:40 am

Floorplan Future Tech 2022



Netcompany	1	Vitas	6
AWS	2	Betabit	7
Bergler	3	Ordina	8
CoderDojo		ABN Amro	9
Cloud Republic	4	4DotNet	10
Microsoft	5	OGD Diensten	11

*Subject to change


Expedition
Mission 1
Mission 2
lift
Community &
Startup plein
11
Workshops
Achmea

Timetable Future Tech 2022

Room/Time	Progress	Mission 1	
10:00 - 10:10			
10.10-10.40	**Keynote: AIRHART – The future of Airport Management – Jonas Johansen**		
10.40-11:00	**Transit zone: Coffee break**		
11.00-11.45	.NET MAUI Blazor - Build Hybrid Mobile, Desktop, and Web apps **Gerard Versluis**	What is the right "unit" in unit testing and why it is not a class? **Dennis Doomen**	
12.00-12.45	Forget SignalR! Start working real-time with Azure Web PubSub **Eduard Keilholz**	Supercharge your App Service to a Global scale **Tiago Costa**	
12.45-13.45	**Transit zone: Lunchbuffet**		
13.45-14.30	**Keynote: Scott Hunter**		
14.30-15.15	What is new in .NET 7 and C# 11 **Johnny Hooyberghs**	"Micro" front-ends in Blazor webassembly **Menno Jongerius**	
15.15-15.45	**Transit zone: Coffee & tea break**		
15.45-16.30	Implementing a Zero-Trust Architecture with Containers & .NET 6 **Alex de Groot**	The Fifth Element: Azure Secure Devops **Mike Martin**	
16.45-17:30	Better architecture without architects **Geert van der cruijsen**	Containers, Machine Learning, Security and GitHub Actions **Henk Boelman & Sjoukje Zaal**	
17.30-18.30	**Transit zone: Drinks & Music**		

*Subject to change

	Mission 2	Expedition	Quest	Workshop area
	Opening			
	Introduction to Azure Percept - accelerated AI on the edge **Andreas Erben**	Hej! An intro to Backstage **Suzanne Daniels**	Programming for the International Space Station with your kids **Menno Jongerius, Marloes Bijkerk & Sanneke van der Meer**	Check futuretech.nl for the latest info
	How to build a cloud native application with .NET and AWS **François Bouteruche**	Running financial services on DevOps-friendly and secure AKS clusters **William Artero & Laura Rehorst - ABN**	Building and Leading Remote Teams **Amber Vanderburg**	
	Bicep: Infrastructure as Code for Azure Reinvented **Henry Been**	Foundations of the Metaverse: Azure Spatial Anchors and Understanding 3D Space **Andreas Erben**	Securing and migrating older projects to newer technologies **Georgia Zarnomitrou**	SOA Done Right **Dennis van der Stelt**
	HUDs of the Metaverse **James Ashley**	Keeping everyone in the Loop **Michiel Hamers & Roelant Dieben**	5 Things Every Developer Should Know About Azure Service Bus **Annejan Barelds**	
	Securing your .NET application software supply-chain **Niels Tanis**	Will Git Be Around Forever? A List of Possible Successors **Hanno Embregts**	Check futuretech.nl for the latest info	

ABN AMRO is happy to join Future Tech

This time on a physical event! Please stop by our booth and try your luck on our wheel of fortune game. You even could win a big LEGO box (or a small one).

This year we also have a great session on stage called 'Running financial services on DevOps-friendly and secure AKS clusters' given by William Moraes Artero and Laura Rehorst.
Whats the session about? "Containers stormed the market since early 2010s. The creation of toolkits to manage them, such as Docker, simplified its usage and adoption. Benefits such as low footprint and isolation turned it a must among solution designs. Developers and businesses both had to understand and adapt to leverage it ever since. From a business perspective, containers are only a small piece of the puzzle. Standard runtimes are enough for development and small business purposes. Enterprise, on the other side, needs granular control over security, governance and scalability. This is where Kubernetes fits in, providing both functional and non-functional container management.
Such requirements double down for financial services. The finance market has high standards across the globe. EU banks, for instance, must comply to dozens of regulations and directives. These ensure the safety of clients' data and assets.
ABN AMRO currently provides internal DevOps teams a product called Managed Container Platform. This solution enables teams to create their own secure and compliant Kubernetes clusters.
It relies on Azure Pipelines and Terraform to create all components so teams don't have to. It also provisions OPA Gatekeeper in all clusters to enforce a set of policies required by CISO. External component restrictions are then handled through Azure Policies.
This solution design has been in use for over 3 years now, on both AWS and Azure. It enables around 600 DevOps teams to use Kubernetes clusters every day.
As the beginning of 2022, the cloud platform at ABN AMRO is undergoing severe changes. The new operating model aims at removing as much abstraction layers as possible, such as MCP. This will enable DevOps teams to direct consume from the resource providers. As such, MCP is evolving into a new major version.
The new solution will lift the dependency on both Azure Pipelines and Terraform. Instead of those, Azure Policies and FluxCD will ensure all compliancy and security. DevOps teams will be free to use whichever infrastructure-as-code solution they want. Stakeholders will also have faster and better control over the policies.
If you want to know more about the implementation details, make sure to attend to Future Tech 2022!"
And if you have any more questions what ABN AMRO is doing with Microsoft Technologies, stop by our booth and have a chat with one of our crew members! ●



ABN·AMRO

Keeping everyone **in the Loop**

During this demo-heavy session we will take you on a tour of Microsoft Loop. At the Ignite conference Microsoft made some announcements around Loop, silently added live components to Microsoft Teams and it is time to lift the veil and dive into how we can leverage Loop to make our lives a lot more productive. After this session you too can leverage the power of Microsoft Loop.

Michiel Hamers & Roelant Dieben





ORDINA

Programming for the International Space Station **with your kids**

Did you know that the International Space Station has two Astro PI's setup to do real space experiments, for educational purposes? And did you know that CoderDojo enables groups of enthusiastic kids and teenagers to get acquainted with software development using this equipment. This session we will show you how, let you play around with the setup, and who knows maybe we will welcome you and your kids on future sessions.

Sanneke van der Meer & Marloes van der Meulen





BERGLER
SOFTWARE SOLUTIONS

5 Things Every Developer Should Know About Azure Service Bus

Did you ever have a discussion with your team mates on whether to use Azure Service Bus? And do some team members mainly see the decoupling benefits, while others fear that messaging makes everything way more complicated than REST? In this talk, I look back on some experiences I had with development teams that were facing these questions, and I'll go over some of their objections, concerns and uncertainties. I'll cover some gotchas to be aware of when message ordering is important; how to achieve end-to-end diagnostics; distributed alternatives to traditional database transactions; and more. After this talk, you'll be equipped to use Azure Service Bus as a .NET developer, and you'll be able to help your team adopt decoupling through messaging.

Annejan Barelds

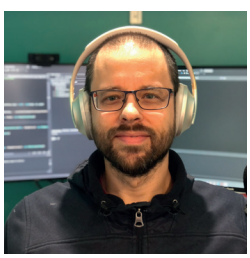


4dotnet

How to build a cloud native application **with .NET and AWS**

.NET has a long history on AWS. It is supported for more than 11 years. The AWS SDK for .NET has been the first AWS SDK released.
When it comes to building cloud native applications with .NET, you can expect to use the full breadth and depth of AWS services to build modern applications. In this session, I will introduce key services you can use with .NET to build cloud native applications. Then, I will discuss the AWS Toolkit for Visual Studio that brings AWS services into our favorite IDE. Finally, I will run a live demo of how to bootstrap quickly a cloud native application leveraging AWS services with .NET.

François Bouteruche



aws

Containers, Machine Learning, Security **and GitHub Actions**

Join Sjoukje and Henk in this session to find out how you can create a Machine Learning pipeline that can train a PyTorch model using GitHub actions and learn how you can deploy and secure the container holding the model in production. There will be a lot of live coding demo's!

Henk is a Cloud Advocate specializing in Artificial intelligence and Azure with a background in application development. He is currently part of the regional cloud advocate team in the Netherlands. Before joining Microsoft, he was a Microsoft AI MVP and worked as a software developer and architect building lots of AI powered platforms on Azure. He loves to share his knowledge about topics such as DevOps, Azure and Artificial Intelligence by providing training courses and he is a regular speaker at user groups and international conferences.

Sjoukje Zaal is a CTO Microsoft, Microsoft Regional Director and Microsoft Azure MVP with over 20 years of experience providing architecture, development, consultancy and design expertise. She works at Capgemini, a global leader in consulting, technology services and digital transformation. She mainly focusses on Cloud, Security, Productivity, and IoT.

She loves to share her knowledge and is active in the Microsoft community as a co-founder of the user groups Tech Daily Chronicle, Global XR Community and the Mixed Reality User Group. She is also a board member of Azure Thursdays and Global Azure. Sjoukje is an international speaker and involved in organizing many events. She wrote several books and writes blogs. Sjoukje is also part of the Diversity and Inclusion Advisory Board.

Henk Boelman & Sjoukje Zaal





Microsoft



Wij zijn OGD. Ruim 1300 inventieve ict'ers, die samen de fullservice-dienstverlening verzorgen van onze klanten. We werken hierbij met diverse technieken, zoals Microsoft Azure IAAS en PAAS, Infra as Code (zoals ARM templates en Bicep) en Azure DevOps. Ons motto is 'Work hard, play hard'. Vanuit onze liefde voor techniek vinden we de best passende oplossing. We werken aan projecten bij klanten als Royal Flora Holland, Rijkwaterstaat, Accare en Festo. We zetten bijvoorbeeld een volledig geautomatiseerde TOPdesk-omgeving op, realiseren werkplekken met Azure Virtual Desktop of richten een Azure-omgeving in volgens het Cloud Adoption Framework. OGD'ers bouwen graag aan oplossingen. Zo geven we bedrijven inzicht in hun data met The Smart Data Grid, een overkoepelend dashboard. Het AI-team detecteert met het ServiceMesh problemen al voordat ze bij de servicedesk zijn aangemeld. Automatiseren doen we niet alleen bij de klant maar ook bij onszelf. Bij ons is er altijd ruimte voor ideeën en feedback. We vinden het tof om te experimenteren en delen kennis actief met elkaar. Bijvoorbeeld in de opleidingen en kennissessies die we zelf opzetten. We verbeteren continu. Ieder jaar organiseren we veel activiteiten, zoals ons eigen technologische festival: Technival! En in coronatijd hebben we ons hoofdkantoor nagebouwd in een game-omgeving, de HUBBUB, om toch een groots kerstfeest te organiseren. Als OGD'er kun je alle kanten op. Van cloud infra specialist tot fullstack-softwareontwikkelaar en van applicatiebeheerder tot business consultant. Je kiest je eigen pad. Je verkent nieuwe wegen en haalt alles uit jezelf, met leuke collega's en een berg plezier!

Barbara

OGD ict-diensten

We zijn volledig Microsoft georiënteerd binnen Vitas, wij ontwikkelen onze zakelijke toepassingen in C#. We richten ons sterk op Microsoft certificeringen en de nieuwste technieken. We houden ons veel bezig met Cloud transities met hoofdzakelijk Azure. We hebben binnen Vitas fullstack ontwikkelaars, dan wel back-end en front-end specialisten.
We zetten actief in op de professionele en persoonlijke ontwikkeling van onze medewerkers. Hierin worden individuele ontwikkel en prestatie doelstellingen gekoppeld aan onze bedrijfsambities, maar blijft persoonlijke aandacht voorop staan. Vitas wordt gekenmerkt door een hoge mate van senioriteit en kennisniveau (Senior developers, architecten, projectleiders en meer). We drukken senioriteit niet uit in aantal jaren, maar in behaalde resultaten van onze professionals. Bovenal staat het individu voorop en onderscheiden onze mensen zich door hun ambitie, enthousiasme en gedrevenheid. Inspanning maar ook tijd voor ontspanning vinden we hierin heel belangrijk.
Daarnaast zijn we een informele, nuchtere club van gedreven professionals waar volop ruimte is voor humor en waar iedereen zichzelf kan zijn. Er wordt op sociaal vlak veel georganiseerd. Binnen Vitas heerst een familiaire sfeer en is de bedrijfscultuur die dit teweegbrengt iets waar mensen zich goed bij voelen.
Het verloop binnen Vitas is nihil en we doen er alles aan om dit te handhaven en onze professionals altijd op de eerste plek te zetten. Dit borgen we door zowel op inhoudelijk vlak, sociaal en financieel vlak persoonlijk aandacht te besteden aan de behoeften van onze medewerkers. En hier vanuit een persoonlijke invalshoek gehoor aan te geven.

VITAS

Wij zijn Cloud Republic, de cloud developers van morgen



We staan bekend als specialisten in .NET / Azure development en alles daaromheen. Alles wat wij doen staat in het teken van passie voor techniek. Wij geloven in 'anders denken' om daardoor voorop te lopen met de nieuwste Cloud technologieën. Daarmee stellen we een aantal grote bedrijven in Nederland dagelijks in staat om maximaal gebruik te maken van de Cloud, DevOps, Data en AI.

Bij Cloud Republic werken echte vakidioten die software bouwen zien als een ambacht. Zelfs nadat we onze laptop dichtklappen, zijn we nog niet uitgepraat over ons werk. Daarom praten we je in onze 'Dev Talks' podcast en 'Beter Worden' YouTube serie wekelijks bij over laatste trends op het gebied van cloud development. Ook een keertje meepraten over jouw expertise? We zijn altijd op zoek naar toffe gastsprekers.

Kom langs op onze stand op Futuretech en ontmoet de gezichten achter Dev Talks, behaal de highscore op onze oldschool arcadekasten en win vette prijzen. We kijken er naar uit om je te ontmoeten!

cloud republic

NON-TECHNICAL SKILLS
EMERGING TECHNOLOGIES
DEVOPS
SECURITY
CLOUD

FUTURE TECH 2022

THE IT CONFERENCE FOR MICROSOFT TECHNOLOGIES

Future Tech thanks her partners

MAIN SPONSOR

netcompany

CO-SPONSOR

ABN·AMRO

PARTNERS

4dotnet

achmea

aws

BERGLER SOFTWARE SOLUTIONS

betabit {···} samen bijzonder maken;

cloud republic

Microsoft

OGD ict-diensten

ORDINA

VITAS

GEORGIA CODES TO HELP STUDENTS INTO THEIR DREAM EDUCATION

What do you want to code for?



Georgia helped to develop a one-stop communication platform for students to store important, personal information. Ultimately, the improved system grants students easier access to enrol or re-enrol at a college or university, making it easier to pursue their dream careers!

How do you want to make a difference?
Have a look at our vacancies:

netcompany

HOOR ALS EERSTE WAT ER IN HET BLAD STAAT!

Luister de Computer Idee Podcast!

LUCHTIG, LAAGDREMPELIG, LEUK EN INFORMATIEF. DAT IS DE COMPUTER IDEE PODCAST. WIL JE IN BEGRIJPELIJK NEDERLANDS OP DE HOOGTE BLIJVEN VAN DE NIEUWSTE ONTWIKKELINGEN IN EN MOGELIJKHEDEN VAN DE DIGITALE SAMENLEVING, DAN IS DEZE PODCAST ECHT IETS VOOR JOU!

CI Podcast

LUISTER NU OP
WWW.COMPUTERIDEE.NL/PODCAST

WIN!

Kom langs in de 4Dotnet stand en maak kans op mooie prijzen!



LEGO Super Mario Nintendo Entertainment System t.w.v. € 230,-



4dotnet ondersteunt bedrijven bij het ontwikkelen van software op basis van Microsoft technologie. Benieuwd wat we voor jou kunnen betekenen? Ga naar www.4dotnet.nl

50 shades of .NET on AWS

One question I hear often from .NET developers is: "how can I host my .NET application on AWS?". I systematically answer: "It depends". In this article, I will walk you through the decision tree to use for deciding the best option to host your .NET application on AWS.

Auteur: François Bouteruche

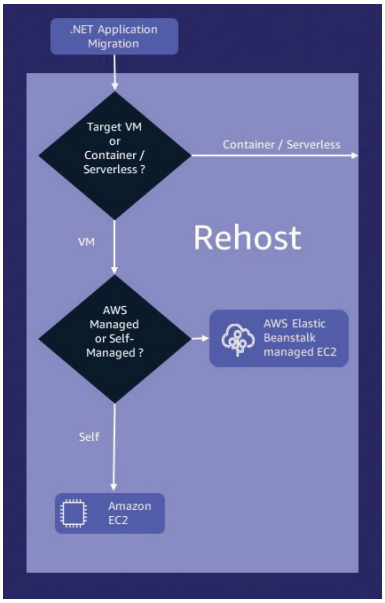
At a first glance, my answer may be frustrating but it is the best and shortest answer I can provide. After all, which .NET are we talking about? Is it a .NET Framework application, a .NET Core or .NET 5+ application? It makes a huge difference. For example, if it is a .NET Core or a .NET 5+ application, you may be able to run your application on Linux and even target our ARM chip, Graviton 2. If it is a .NET Framework application, you will have to stick to Windows Server.

There are many other questions that drive the decision to relocate your application to Amazon EC2 instances (short for Elastic Compute Cloud, basically virtual machines) or, if you want to go in a different direction, to refactor your application to run as serverless functions on AWS Lambda, our serverless, event-driven compute engine. Let's walk through the decision tree together.

Relocate/rehost your .NET application

The first thing you need to assess is if you want to continue to target virtual machines for hosting your applications. There are many reasons for this. Here are a few:

- Your operations team is highly skilled at managing your infrastructure and you want to build on that.



- Your operations and/or development teams have no container or serverless architecture skills.
- Your application has specific requirements that will be easier to meet by using Amazon EC2 instances, e.g. many CPU cores, many gigabytes of memory, spot instance usage.

In this case, your .NET application should be hosted on Amazon EC2 instances. It will be on Windows if it is a .NET Framework application. It can be on Windows or Linux if it is a .NET Core or .NET 5+ application.

But even there, you still have a choice. Do you want to run self-managed Amazon EC2 instances or do you want AWS to manage your EC2 instances for you? If the second option is appealing to you then AWS Elastic Beanstalk is the right service to host your .NET application. It will manage your Amazon EC2 instances for you as well as your load balancer and your auto scaling group (which are optional). The auto scaling group will automatically increase or decrease instances based on a set of rules you define.

It you have a complex application that require DevOps expertise, you may prefer to self-manage your own Amazon EC2 instance fleet, your load balancer and your auto scaling group to have full control and optimize your workload.

BIO

Francois Bouteruche

Francois is currently a Senior Developer Advocate at Amazon Web Services, advocating for .NET and Java developers. He has a PhD in Computer Science and started his career working on handwriting recognition. He has more than 15 years of experience in .NET development and more than 8 years in the Cloud area. He joined AWS in 2019 to help .NET developers using AWS Cloud to build, test and deploy efficiently and securely on AWS Cloud.

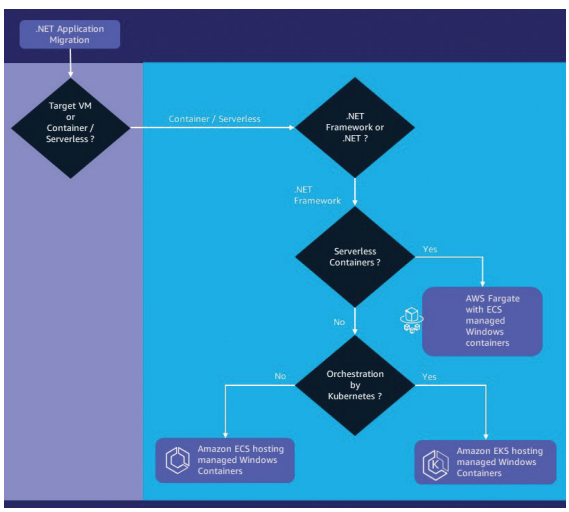


Replatform your .NET application

If your .NET application is containerized or you are open to containerizing it, then a wide range of options are available to you.

Replatform your .NET Framework application

Let's reduce the scope by first addressing the options for a .NET Framework application. In this case, you need to run Windows containers. If you don't want to manage the Amazon EC2 instances that will host your Windows containers, then your option is to use AWS Fargate with Amazon Elastic Container Service (Amazon ECS) managed Windows containers. AWS Fargate is a serverless compute engine for containers that runs your containers without you having to manage underlying compute hosts for your container instances. Amazon ECS Is our built-in container orchestrator. It can orchestrate your container instances on your own Amazon EC2 instance cluster or leverage AWS Fargate to save you much form the heavy lifting of managing compute hosts.



AWS Fargate may not fit your need because you may have specific hardware requirements that AWS Fargate is not able to cover. In this case, you will run your .NET containerized application on Amazon EC2 instances. You still need to decide which container orchestrator you want to use: Kubernetes or Amazon Elastic Container Service. Kubernetes is an open-source container orchestration system managed by the Cloud Native Computing Foundation and widely supported. If you decide to invest in Kubernetes, you can host your .NET application on Amazon Elastic Kubernetes Service (Amazon EKS) with Amazon EC2 Windows instances. Amazon EKS offers you a managed Kubernetes-based container orchestration.

You may have no appetite for learning Kubernetes and prefer to use an orchestrator deeply integrated with the AWS platform. If so, you can host your .NET application on Amazon Elastic Container Service with Amazon EC2 Windows instances.

> At a first glance, my answer may be frustrating but it is the best and shortest answer I can provide

Replatform your .NET Core, .NET 5+ application

If your application is a .NET Core or .NET 5+ application, the question you need to answer is: do you want to run your application on Windows or Linux? There are many reasons for continuing to run your .NET Core/.NET 5+ application on Windows servers. Here are the two most common:

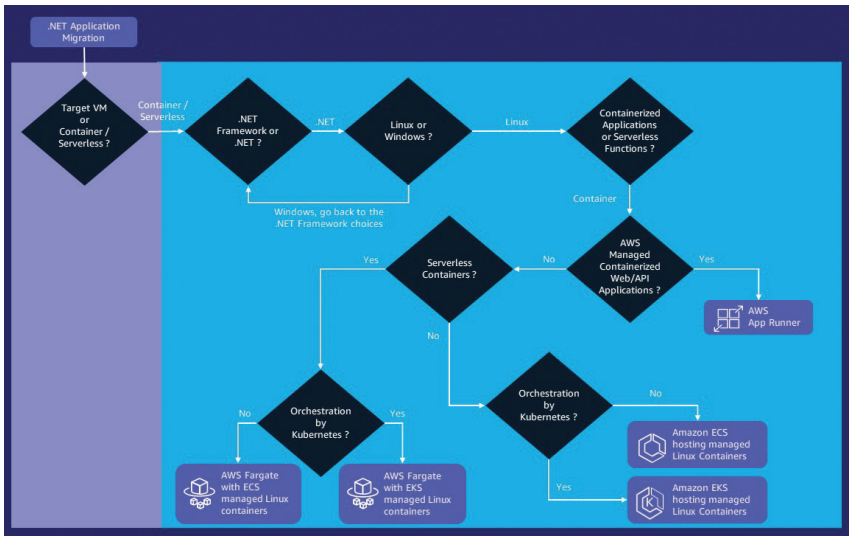
- Your operations and/or development team is highly skilled on Windows operations and has less knowledge on Linux operations.
- Your application has a dependency on a module that can run only on Windows.

In this case, the decision tree you have to walkthrough is basically the same as for .NET Framework applications.

To reduce your need for Windows Server licenses or standardize your operations on one operating system, you may also decide to run your .NET application on Linux. This offers you even more options. If a container is your favorite application packaging format you can continue there. If you are interested by serverless functions, jump directly to the next section.

If container is your choice, the next node in the decision tree is about the nature of your application. If you are building a publicly accessible Web application or API and want to sit in the passenger seat for operating your infrastructure, I would recommend to use AWS App Runner. It is a fully managed service for running your web or API application with no prior infrastructure experience required. Under the hood, it automatically deploys your application, load balances traffic with encryption and scales up and down the number of container instances to meet your traffic needs. But you don't need to worry about all this.

However, not every .NET application is about public Web application



or API. You may want to deploy a private Web application or API. You may want to deploy a background worker process. You may want to deploy batch jobs. Even in these cases you can let AWS manages the infrastructure, health check, scaling, etc. for you.

In this case, you will use AWS Fargate to run your containerized .NET applications and the last question you have to answer is whether you want to use Kubernetes as an orchestrator and thus Amazon EKS or our built-in orchestrator Amazon ECS.

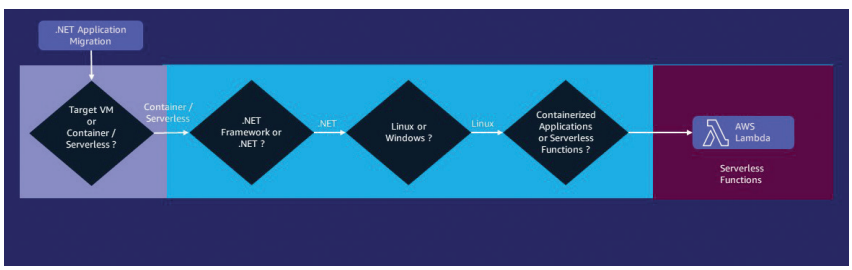
If you have specific hardware or configuration requirements, you may want to consider deploying your .NET containerized application on self-managed compute hosts. It will offer you maximum flexibility on the configuration of your hosts. Here again, you end up by choosing between Amazon EKS or Amazon ECS.

Refactor to serverless functions

Finally, if you want to avoid using containers or want to leverage a fully managed infrastructure or you are developing event-oriented .NET applications, you are a candidate for using serverless functions and AWS Lambda, our serverless, event-driven compute engine.

If you want to leverage the event-driven nature of AWS Lambda compute and still use containers to package your application, you can. AWS Lambda offers you this flexibility.

The .NET Lambda function programming model is pretty straightforward. The code blow shows the default function handler method you get when you create a C# Lambda function for processing Amazon Simple Queue Service (Amazon SQS) events. Amazon SQS is a fully queue service. It can trigger your Lambda function each time a message or a batch of messages reach the queue. Your function handler receives an SQS event that contains the messages and all you have to do is to write the logic for processing the messages. **Listing 1**.




```
/// <summary>
/// This method is called for every Lambda invocation. This method takes in
/// an SQS event object and can be used to respond to SQS messages.
/// </summary>
/// <param name="evt"></param>
/// <param name="context"></param>
/// <returns></returns>
public async Task FunctionHandler(SQSEvent evt, ILambdaContext context)
{
    foreach(var message in evt.Records)
    {
        await ProcessMessageAsync(message, context);
    }
}
```

1

Conclusion

You really have 50 shades of .NET on AWS. Knowing which way to host is the right for you may seem overwhelming at first. If you use the decision tree we have gone through in this article, you will end up with the right option in the vast majority of the use cases.

You may have some use cases that sill don't fit in this decision tree because of hardware, security, compliance, or business requirements.

If so, don't hesitate to reach out to the AWS community or AWS Developer Advocates like myself on social networks or on repost.aws.

If you want to learn more, I encourage you to visit our AWS .NET homepage (https://aws.amazon.com/dotnet) and to read our "Developing and Deploying .NET Applications on AWS" Whitepaper (https://go.aws/39N-Clk2). If you prefer videos, you can visit our .NET on AWS playlist on the AWS YouTube channel (https://bit.ly/dotnetonaws-youtube). ●



> Denk jij dat je het beter weet?

/ Mooi, dan zoeken we jou!

/ Werk aan complexe ict-projecten
én deel jouw kennis met 1300 collega's.

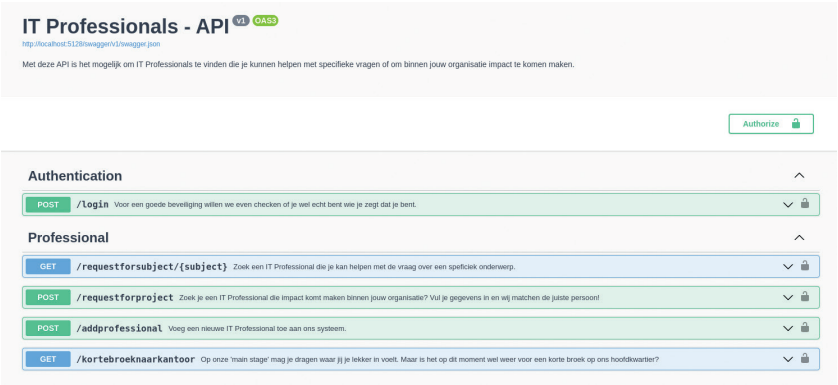
> werkenbij.ogd.nl
Je baan is meer dan werk.

Externe REST services **aanroepen met Refit**

Je kunt je bijna geen applicatie meer voorstellen die geen gebruik maakt van externe gegevens. Vaak komt deze data uit REST Api's[1]. Als developer kan het behoorlijk saai zijn om de zoveelste API te moeten ontsluiten maar gelukkig zijn er mogelijkheden om dit een stuk eenvoudiger te maken. Één van die opties is het gebruik van Refit[2].

Auteur: Jacob Duijzer

Om te laten zien hoe je met behulp van Refit[2] een API aan kunt roepen gebruik ik een een voorbeeeld API met een 5-tal methodes. De voorbeeldcode is ook beschikbaar op GitHub[3]. De API ziet er zo uit:



Beschikbare NuGet packages

Er zijn twee verschillende NuGet packages beschikbaar, de standaard Refit4 package en Refit.HttpClientFactory[5], dezelfde package maar met extra extensies voor Dependency Injection. Voor de meeste voorbeelden maak ik gebruik van een console applicatie met de normale Refit[4] nuget package, alleen bij het laatste voorbeeld gebruik ik ook dependency injection.

De implementatie van Refit

Refit[2] gebruikt een interface als definitie voor het genereren van code om een API aan te kunnen roepen. In deze interface definieer je welke gegevens je terug verwacht, welke parameters je mee wilt geven en of de API authenticatie gebruikt. Je kunt aangepaste headers meegeven of Bearer Tokens voor authenticatie. In dit artikel laat ik een aantal veelgebruikte functies van Refit[2] zien, als je meer wilt weten over de mogelijkheden dan raad ik je aan om de website van Refit[2] te lezen.
De interface die nodig is om de bovenstaande API aan te kunnen roepen ziet er zo uit: **Afbeeling 1**.
De eerste methode is een GET request met als parameter Subject. Als resultaat wordt een POCO terug gegeven met gegevens van de gevonden professional. De tweede methode, een POST geeft geen gegevens terug en verzend een ProjectDescription als body. De code om de API daadwerkelijk aan te roepen ziet er zo uit: **Afbeeling 2**.

Authenticatie

Een van de belangrijkste onderdelen van een API is beveiliging. Hebben de gebruikers van de API wel de juiste rechten en zijn ze wie ze zeggen dat ze zijn?
Er zijn twee methodes beschikbaar in de API, een login-methode om een Bearer token op te halen en een methode, AddProfessional, waaraan dit token meegegeven dient te worden, beide methodes zijn ook opgenomen in de interface voor Refit[2].
De methode AddProfessional heeft een Authorize attribuut om aan te geven dat deze methode alleen door geauthenticeerde gebruikers mag worden aangeroepen: **Afbeeling 3**.
In het onderstaande voorbeeld wordt een nieuwe professional aangemaakt door eerst een token op te halen en deze vervolgens mee te sturen naar de API: **Afbeeling 4**.



1

```
public interface IProfessionalsApi
{
    [Get("/requestforsubject/{subject}")]
    Task<Professional> FindProfessionalForQuestion(string subject);

    [Post("/requestforproject")]
    Task SendRequestForProject([Body] ProjectDescription projectDescription);

    [Post("/login")]
    Task<string> Login(UserDto userDto);

    [Post("/addprofessional")]
    Task AddProfessional([Body] Professional professional, [Authorize("Bearer")] string bearerToken);

    [Post("/kortebroeknaarkantoor")]
    Task<string> CanIWearShortsToTheOffice();
}
```

2

```
// Create an API based on the interface
var api = RestService.For<IProfessionalsApi>("http://localhost:5128");

// Example 1: simple get
var professional = await api.FindProfessionalForQuestion(Subject.Agile.ToString());
Console.WriteLine($"Je kunt met je vragen over {Subject.Agile} terecht bij de deze professional: {professional.FirstName} {profes-
sional.LastName}, {professional.Title} <{professional.EmailAddress}>.");

// Example 2: simple post
ProjectDescription projectDescription = new()
{
    CompanyName = "Some Awesome Company",
    Location = "Eindhoven",
    JobTitle = Title.TeamLead,
    Description = "Voor een zelfstandig Agile team van 4 developers en een scrum master zijn we op zoek naar een Team Lead die kan
helpen het niveau van het team op een hoger niveau te brengen. De tech-stack is .NET, het niveau van de developers is wisselend"
};
await api.SendRequestForProject(projectDescription);
```

3

```
// Demo method, authorized post
[HttpPost("addprofessional"), Authorize]
public IActionResult AddProfessional([FromBody] Professional professional)
{
    return Created(string.Empty, professional);
}
```

4

```
// Example 3: login, using bearer token
Professional newProfessional = new()
{
    FirstName = "Dirk",
    LastName = "Alma",
    EmailAddress = "dirk.alma@gmail.com",
    Title = Title.ProductOwner,
    Technologies = new List<string> { Subject.Agile.ToString(), Subject.AzureDevOps.ToString()}
};

var bearerToken = await api.Login(new UserDto("rock", "star123"));
await api.AddProfessional(newProfessional, bearerToken);
```

5

```
public interface IOpenWeatherApi
{
    [Get("/data/2.5/weather?lat={lat}&lon={lon}&appid={apiKey}&units=metric")]
    Task<WeatherData> WeatherForLocation(string lat, string lon, string apiKey);
}
```

6

```
builder.Services
    .AddRefitClient<IOpenWeatherApi>()
    .ConfigureHttpClient(c =>
        c.BaseAddress = new Uri("https://api.openweathermap.org/"));
```

7

```
[HttpGet("kortebroeknaarkantoor")]
public async Task<IActionResult> KorteBroekNaarKantoor(
    [FromServices] IOpenWeatherApi openWeatherApi,
    [FromServices] WeatherServiceSettings weatherServiceSettings)
{
    var weer = await openWeatherApi.WeatherForLocation(weatherServiceSettings.Latitude, weatherServiceSettings.Longitude, weather-
ServiceSettings.ApiKey).ConfigureAwait(false);
    return Ok(weer.Temperatures.Maximum >= 15f ?
        $"Prima weer om in je korte broek te gaan, het wordt vandaag {Math.Round(weer.Temperatures.Maximum, 0)} graden!" :
        $"Ik zou het niet doen vandaag, het wordt maar {Math.Round(weer.Temperatures.Maximum, 0)} graden!");
}
```

```
[Fact]
public async Task ReturnTheWeatherForTheRequestedLocation()
{
  // ARRANGE
  var mockApi = new Mock<IOpenWeatherApi>();
  mockApi
      .Setup(with => with.WeatherForLocation(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))
      .ReturnsAsync(new WeatherData {Location = "Test Location"});
  var weatherService = new WeatherService(mockApi.Object);

  // ACT
  var weather = await weatherService.GetWeatherForLocation("testlat", "testlong", "api-key");

  // ASSERT
  mockApi.Verify(apiCall => apiCall.WeatherForLocation(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()), Times.Once);
}
```

8

BIO

Jacob Duijzer

Jacob Duijzer is een Software Engineer met een passie voor kwaliteit en samenwerking. Hij is altijd op zoek naar de verbinding tussen diverse disciplines en het verbeteren van teams, vaak door middel van het organiseren van Coding Dojo's. Jacob is werkzaak bij Team Rockstars IT.
LinkedIn: https://linkedin.com/in/jacobduijzer



Dependency Injection

Voor de laatste methode van de API maakt de applicatie gebruik van een externe API de weersvoorspelling op te halen bij OpenWeatherMap[6]. Ook voor deze API gebruik ik Refit[2]: **Afbeeling 5**.
Vervolgens registreer ik deze interface: **Afbeeling 6.**
Bovenstaande code is voldoende om de externe API aan te kunnen roepen vanuit een controller of service: **Afbeeling 7.**

Unit Testen

Het gebruik van interfaces heeft nog een bijkomend voordeel: met behulp van bijvoorbeeld Moq is het eenvoudig om code die gebruik maakt van Refit te testen. **Afbeeling 8**

Conclusie

Refit[2] is een tool waarmee je eenvoudig en snel API's kunt ontsluiten, zowel extern als binnen je eigen netwerk. Ik gebruik het zelf al zo lang dat ik bijna niet meer weet hoe ik het zonder zou moeten doen. Er zijn ook alternatieven voor de HttpClient of Refit, ik kom bijvoorbeeld regelmatig Flurl[7] tegen. ●

1. REST Wikipedia: https://en.wikipedia.org/wiki/Representational_state_transfer
2. Refit: https://github.com/reactiveui/refit
3. Voorbeeldproject: https://github.com/jacobduijzer/RefitVoorbeelden
4. Refit nuget package: https://www.nuget.org/packages/Refit/
5. Refit.HttpClientFactory nuget package: https://www.nuget.org/packages/Refit.HttpClientFactory/
6. OpenWeatherMap website: https://openweathermap.org
7. Flurl website: https://flurl.dev

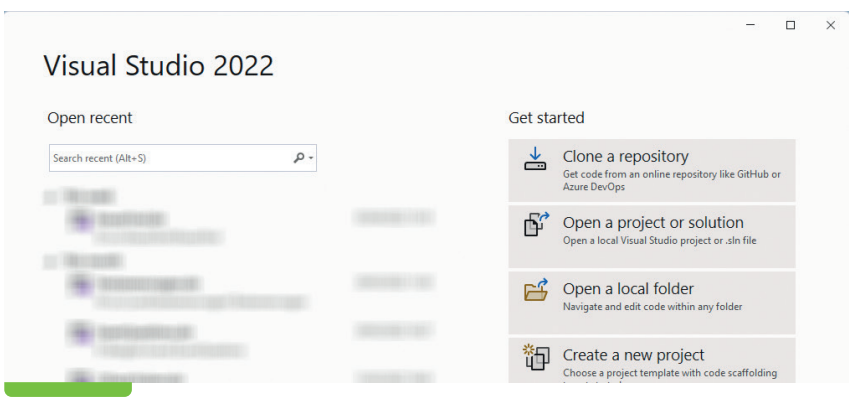
Visual Studio **Solution Filters**

Auteur: Johan Smarius

Visual Studio is een geweldig product, maar de performance van Visual Studio bij grotere solutions wil nog wel eens een punt van aandacht zijn. Ook kan een solution met daarin heel veel projecten de aandacht wel eens afleiden van de code die aangepast moet worden. Dit geldt zeker voor een nieuwkomer op een project die in één keer heel veel code ziet en daardoor moeite zal hebben om door de bomen het bos te zien. In Visual Studio 2019 is een feature geintroduceerd die hierbij kan helpen: solution filters. In dit artikel wordt uitgelegd hoe deze feature gebruikt kan worden.

Instellen solution filter

Om solution filters in te stellen, moet de solution met de optie "Open a project or solution" geopend worden (**Afbeelding 1**).
Bij de bestandsselectie is het belangrijk om de optie "Do not load projects" aan te zetten (**Afbeelding 2**).
De solution wordt nu geopend zonder de projecten te laden (**Afbeelding 3**).
Voor alleen die projecten die te maken hebben met bijvoorbeeld de "WeatherForecast" kan een solution filter worden ingesteld. Hiervoor zijn twee mogelijkheden. Ten eerste kan een project geladen worden inclusief de afhankelijkheden met andere projecten (**Afbeelding 4**).
Het project "WeatherForecastService" is afhankelijk van "WeatherForecaseStreamingService.Contracts" en dit laatste project zal dus ook gelijktijdig geladen worden. De andere optie is om een enkelvoudig project te laden. De "WeatherForecastClient" heeft geen afhankelijkheid van de service en zal dus ook nog apart geladen moeten worden (**Afbeelding 5**).
Binnen de solution zijn op dit moment alleen die projecten geladen die te maken hebben met de WeatherForecast, dus het doel is bereikt (**Afbeelding 6**).
Bij het bouwen van de solution worden nu alleen de drie actieve projecten gebouwd (**Afbeelding 7**).



Afbeelding 1 Open solution.

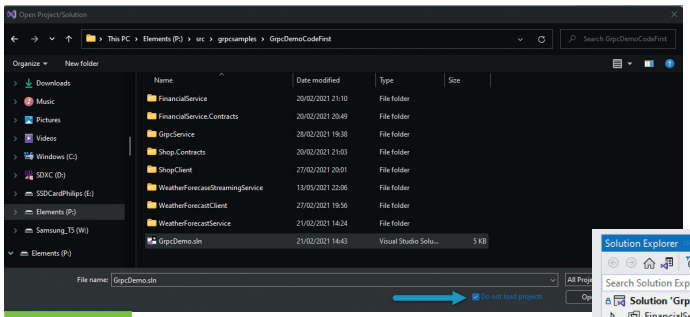
Een code analyse op de solution, neemt ook alleen maar de geladen projecten mee (**Afbeelding 8**).
Ook de unit testen die gedraaid worden, beperken zich tot de geladen projecten (**Afbeelding 9**). Voor dit voorbeeld is een solution filter ingesteld op een solution die gebaseerd is op een Onion architectuur. Binnen de complete solution zitten heel veel unit testen, maar die zijn nu ook niet allemaal zichtbaar.
Bij het opslaan van de solution, wordt de geladen status van de projecten ook bijgehouden. Als de solution dus geopend wordt, zijn weer alleen de geladen projecten direct zichtbaar. Als voor een aanpassing nu andere projecten nodig zijn, dan moeten deze extra projecten weer geladen worden en andere

BIO

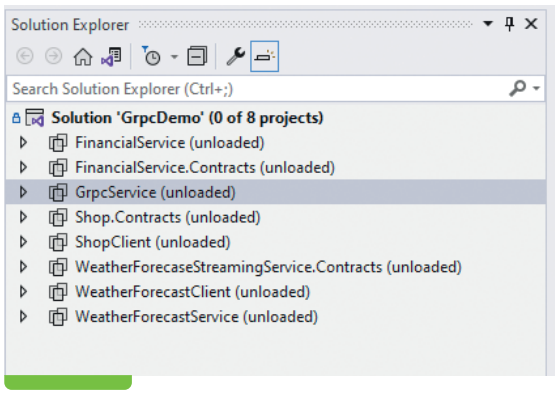
Johan Smarius

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.johansmarius.com

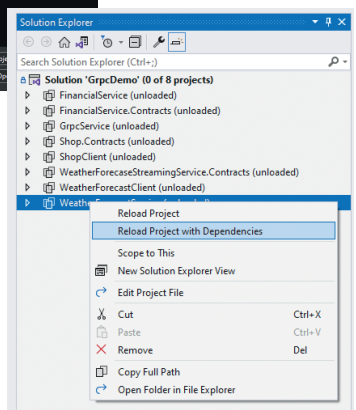




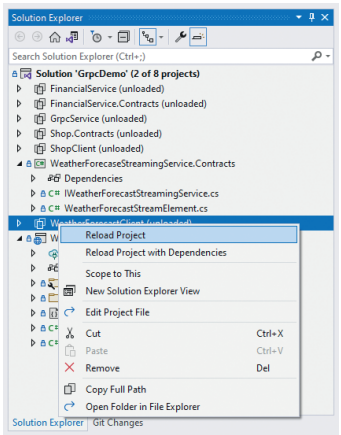
Afbeelding 2 Bestandsselectie.



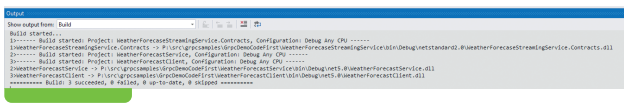
Afbeelding 3 Geladen solution.



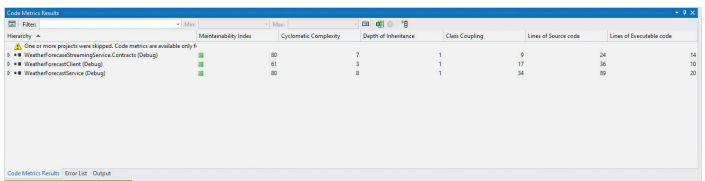
Afbeelding 4 Laden van projecten met afhankelijkheden.



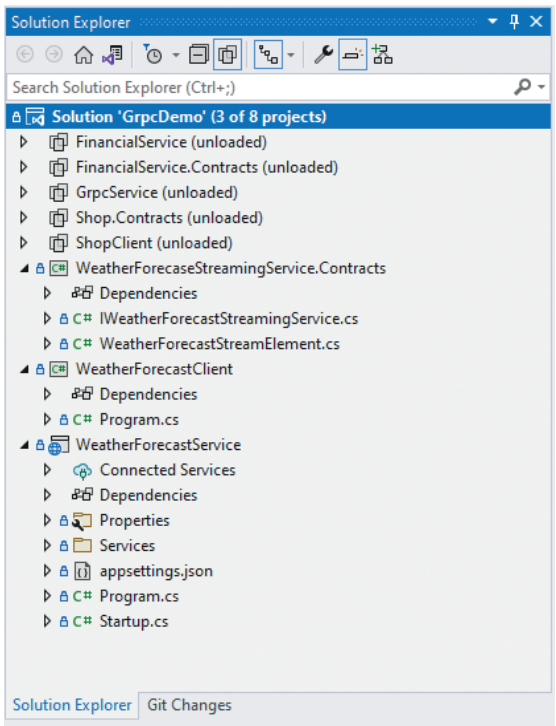
Afbeelding 5 Los laden van een project.



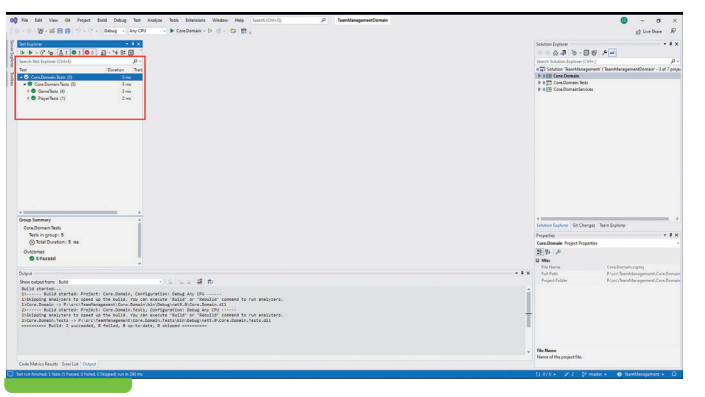
Afbeelding 7 Build output.



Afbeelding 8 Code analyse op geladen projecten.

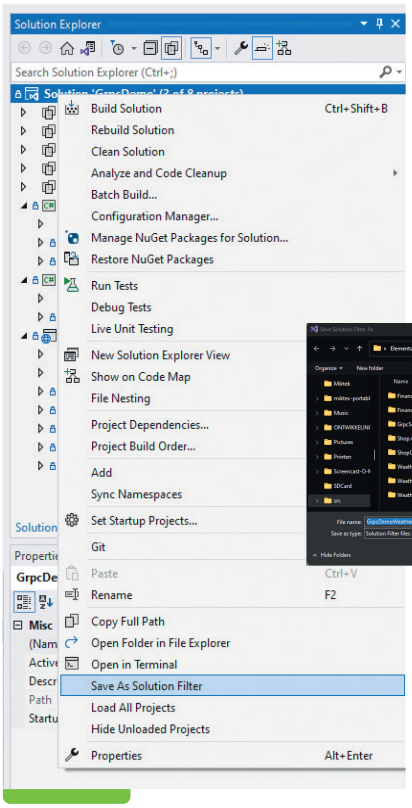


Afbeelding 6 Geladen projecten.

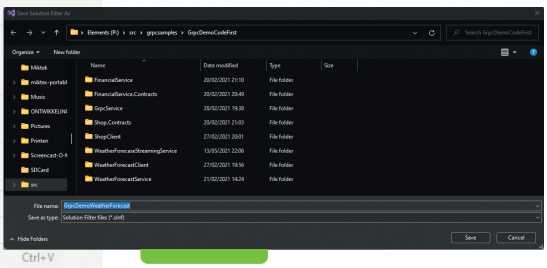


Afbeelding 9 Unit testen en geladen projecten.

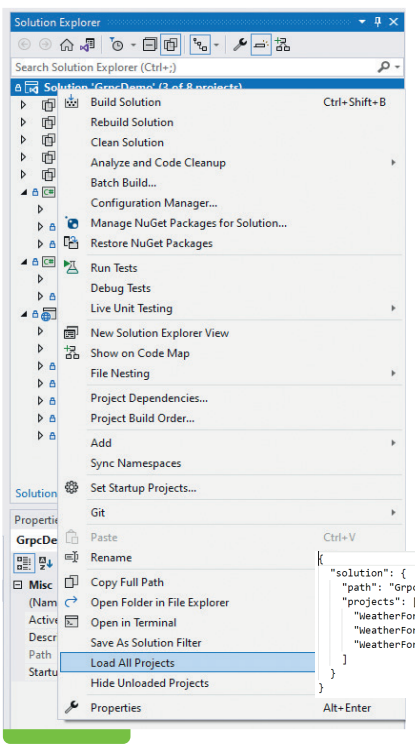
projecten eventueel ontladen. Dit is niet handig, zeker niet in teamverband. De gewenste projecten kunnen natuurlijk in een losse solution op worden genomen, maar dat vereist extra onderhoud als nieuwe projecten aan de solution toe moeten worden gevoegd. Deze nieuwe projecten moeten dan namelijk aan meerdere solutions toe worden gevoegd. In verband met de kans op fouten is dit niet handig. Gelukkig kan de staat van de geladen projecten gemakkelijk opgeslagen worden als een filter op de solution. Hiervoor kan gebruik worden gemaakt van de optie “Save as Solution Filter” in het contextmenu van de solution (**Afbeelding 10**).
Het solution filter moet een bestandsnaam krijgen met de extensie “slnf”. (**Afbeelding 11**).
Na het opslaan van het solution filter bestand is het belangrijk om alle projecten in de solution weer te laden. De solution zelf moet namelijk idealiter alle projecten geladen bevatten en via een of meerdere solution filters kunnen dan selecties worden gemaakt. Voor het herladen van alle projecten kan gebruik worden gemaakt van de optie “Load All Projects” uit het contextmenu van de solution (**Afbeelding 12**). Het herladen van de projecten kan in een grote solution wel even duren.
De solution en het solution filter kunnen nu apart van elkaar worden geopend (**Afbeelding 13**).
De solution explorer geeft aan dat een solution filter geopend is (**Afbeelding 14**). Het totaal aantal projecten in de parent-solution wordt hierbij namelijk ook getoond.
De bestandsindeling van een solution filter is heel anders dan van een solution (**Afbeelding 15**). Binnen het bestand van een solution filter worden alleen maar die projecten genoemd die geladen moeten worden en verder wordt de koppeling met de originele solution gelegd. Hiermee ontstaat dus een oplossing die beter onderhoudbaar is, dan wanneer een



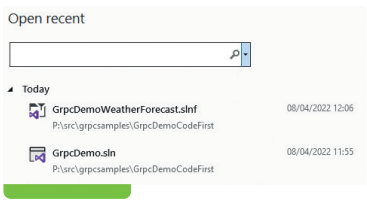
Afbeelding 10 Opslaan als solution filter.



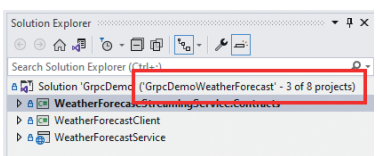
Afbeelding 11 Opslaan van solution filter bestand.



Afbeelding 12 Load all projects.



Afbeelding 13 Recent projects met solution en solution filter.



Afbeelding 14 Geladen solution filter.



Afbeelding 15 Bestandsindeling solution filter.

kopie zou zijn gemaakt van de originele solution file.

Wisselen tussen solution en solution filter

Tijdens het werken met de code moet soms worden gewisseld tussen het solution filter en de complete solution. Voor een GIT push is het bijvoorbeeld handig om vooraf alle unit testen te draaien. Gelukkig biedt Visual Studio hier hele goede ondersteuning voor. Via de switch-optie boven in de Solution Explorer (**Afbeelding 16**) kan gemakkelijk worden gewisseld tussen de solution en het solution filter. Door te dubbelklikken op de gewenste keuze, wordt die keuze geladen.

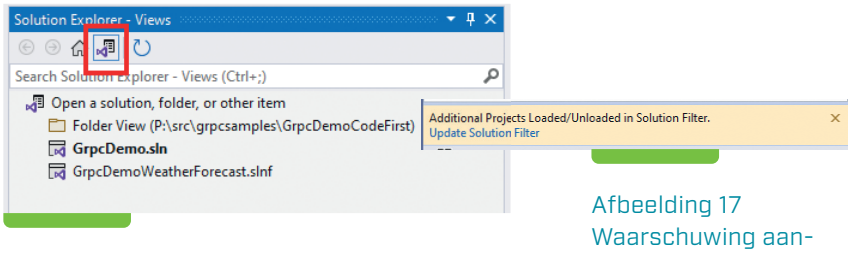
Na het laden van het solution filter geeft Visual Studio ten onrechte soms de melding dat de projecten in het filter aangepast zijn, zonder dat dit het geval is (**Afbeelding 17**). Deze melding kan veilig worden genegeerd. Kiezen voor update kan ook zonder bijwerkingen.

Aanpassen solution filter

Het kan natuurlijkvoorkomen dat een bestaand filter aangepast moet worden. Gelukkig wordt dit ook ondersteund. In het contextmenu van de solution is hiervoor de optie "Show Unloaded Projects" (**Afbeelding 18**) beschikbaar. Deze functionaliteit kan ook worden bereikt door te klikken op de knop van "Show All Files" in de menubalk van de Solution Explorer (**Afbeelding 19**).

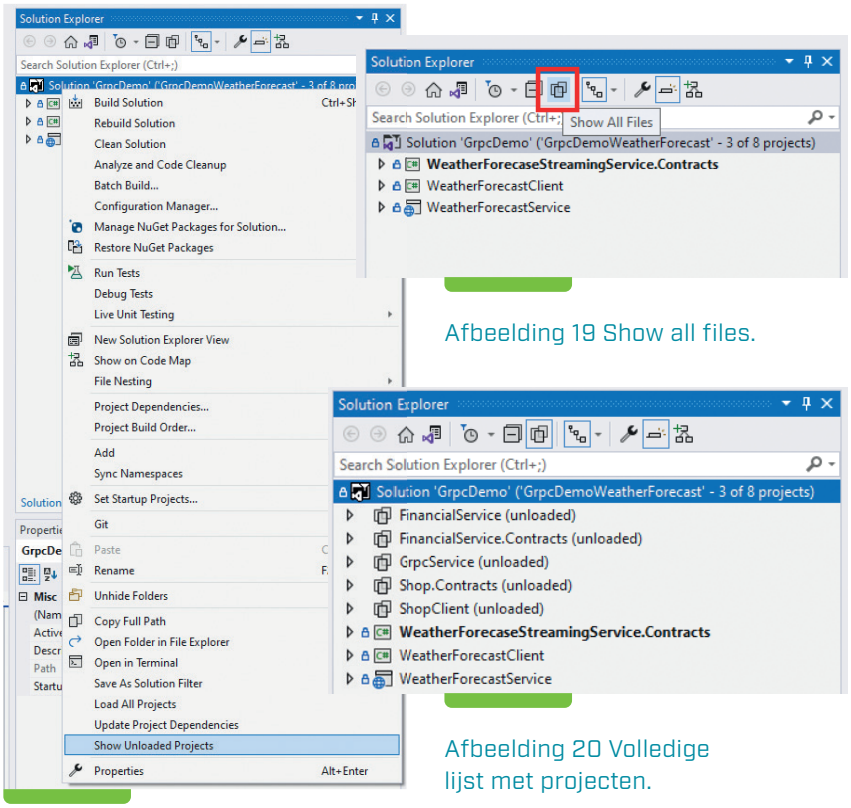
De volledige lijst met projecten wordt nu weer getoond (**Afbeelding 20**). Via de opties "Reload Project" of "Reload project with Dependencies" kunnen nu extra projecten worden toegevoegd aan het filter. Na het herladen wordt nu terecht de melding getoond dat het filter aangepast is en dat deze moet worden geüpdatet (**Afbeelding 21**).

De inhoud van het filterbestand wordt na het klikken op update direct aangepast (**Afbeelding 22**).



Afbeelding 16 Wisselen tussen solutions.

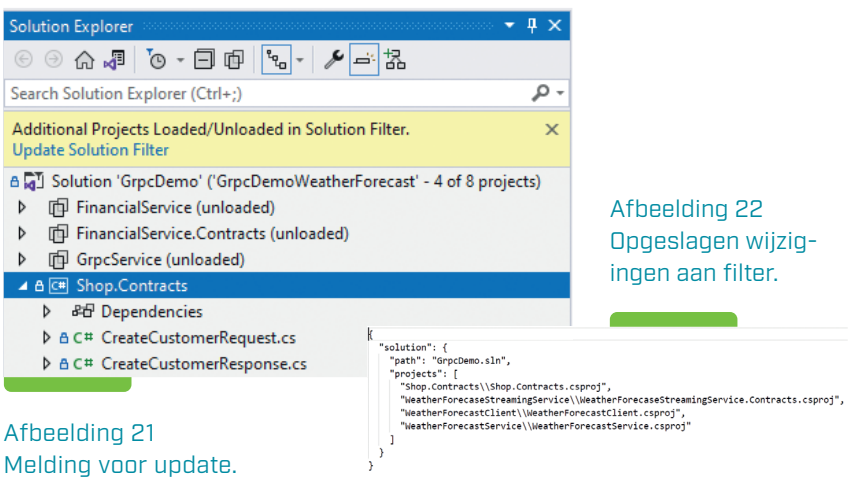
Afbeelding 17 Waarschuwing aanpassingen filter.



Afbeelding 19 Show all files.

Afbeelding 20 Volledige lijst met projecten.

Afbeelding 18 Show unloaded projects.



Afbeelding 21 Melding voor update.

Afbeelding 22 Opgeslagen wijzigingen aan filter.

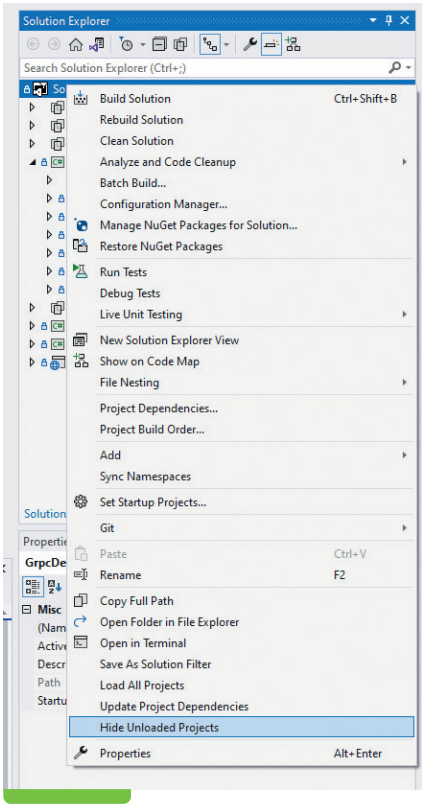
De projecten die niet zichtbaar hoeven te zijn binnen het filter, kunnen weer worden verborgen met de optie "Hide Unloaded Projects" (**Afbeelding 23**) of door gebruik te maken van het knopje in de menubalk van de Solution Explorer.

Ondersteuning in Visual Studio for Mac

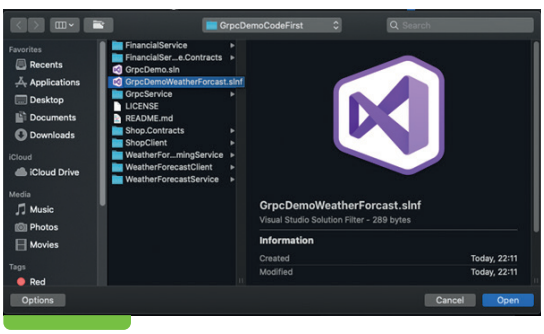
De ondersteuning van solutions filters in Visual Studio for Mac is helaas niet zo compleet als onder Windows. Solution filters kunnen wel geladen worden in Visual Studio for Mac (**Afbeelding 24**) en deze worden ook correct geladen (**Afbeelding 25**). Het aanmaken en aanpassen van de solution filters zelf, kan helaas nog niet met Visual Studio for Mac. Hiervoor is dus echt de Windows-versie nodig of het bestand moet met de hand worden aangepast. Handmatige aanpassing neemt wel een stuk van de handigheid van de solution filters weg en is bovendien natuurlijk heel erg foutgevoelig.

Samenvatting

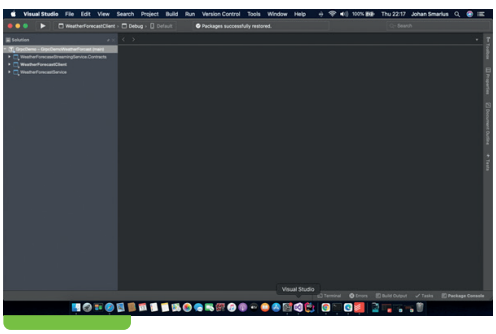
In dit artikel is de werking van solution filters in Visual Studio getoond. De ondersteuning in de Mac-versie is op dit moment nog te beperkt om bruikbaar te zijn als alleen op een Mac wordt gewerkt. Met de filters kan een losse view op een complete solution gemaakt worden die sneller zal laden, maar zeker ook minder complex zal ogen voor een beginnend ontwikkelaar op het team.
Het is wel belangrijk om niet alleen vanuit het solution filter te werken, want regelmatig is het nodig om zeker te weten dat de hele solution ook bouwt en zonder problemen draait. Eventueel kan hiervoor natuurlijk ook de .NET-CLI gebruikt worden.
Op https://github.com/johansmarius staat de code die gebruikt is in dit artikel. De screenshots in dit artikel zijn afkomstig uit deze code. ●



Afbeelding 23 Verbergen van overige projecten.



Afbeelding 24 Laden van Solution Filter in Visual Studio for Mac.



Afbeelding 25 Geladen solution filter in Visual Studio for Mac.



MAY THE SOURCE BE WITH YOU

BERGLER ICT

Liefhebbers van IT

Gold Microsoft Partner

PARTNERSHIP VOOR FLEXIBELE CONTINUÏTEIT

Software teams lopen in de praktijk tegen veel uitdagingen aan. Hierbij is het belangrijk dat voor de lange termijn continuïteit geleverd wordt. Ons doel is om je van dienst te zijn met gerichte kennis en ondersteuning, en je daarmee te helpen je (softwareontwikkel-) roadmap te realiseren.

Bergler Software Solutions brengt kennis en expertise, biedt ontwikkelcapaciteit en bouwt software-applicaties. Bergler onderscheidt zich, al ruim 40 jaar, daarnaast via een eigen "CTO-office on demand" (Competence Center), QA Software (optimaliseren van de kwaliteit), populaire on- en offline kennis-events en een jaarlijkse (openbare) TechRadar waarin wij onze visie delen op de assen techniek, platformen, tools en ontwikkeltalen met betrekking tot zaken waarvan wij geloven dat ze belangrijk zijn.

ONZE MERKWAARDEN: DESKUNDIG, NUCHTER EN SAMEN

We brengen dit in de praktijk door onze kennis en ervaring te delen en door gerichte ondersteuning van onze klanten door ons unieke Team as a Service (TaaS) concept. Dit betekent een slimme en vooral efficiënte combinatie van ondersteuning door onze Lead Developers en capaciteit door onze professionals in Nederland, Servië en Portugal. Het resultaat: de beste prijs-kwaliteit en schaalbaarheid.

De expertise van **Bergler Software Solutions** richt zich op de Microsoft stack. Via **Bergler Flex Solutions** zijn wij naast de juiste expertise ook in staat snel te helpen met bredere IT capaciteitsvraagstukken. Bergler ICT is Microsoft Gold partner. Meer weten? Kijk op www.bergler.nl

CODER(DOJO);

Ook de allerjongste liefhebbers van IT hebben de toekomst!

Bergler ICT werkt nauw samen met CoderDojo Nederland. CoderDojo heeft als doel om jonge mensen enthousiast te maken voor ons prachtige vakgebied. Bij CoderDojo leren kinderen tussen de 7 en 17 jaar programmeren, apps of websites maken en meer. De bijeenkomsten ('dojo's') zijn altijd gratis en worden verzorgd door enthousiaste mentoren. Wil jij namens Bergler ook mentor voor CoderDojo zijn? Neem voor meer informatie contact met ons op!

www.bergler.nl | info@bergler.nl

Code analyseren en genereren met Roslyn

Bij het ontwikkelen van software worden we vaak geholpen door integrated development environments (IDE's) zoals Visual Studio. Visual Studio geeft ons bijvoorbeeld suggesties met de gedefinieerde classes en methodes in de vorm van IntelliSense. Ook kunnen IDE's vaak helpen met code genereren,verplaatsen en refactoren.

Auteur: Maarten Grootoonk

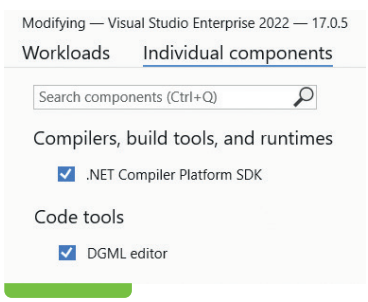
Om deze suggesties te kunnen geven en code acties uit te kunnen voeren moet de IDE de geschreven code begrijpen. Dit wordt meestal gedaan met statische code analyse. Hierbij wordt de code ingelezen en omgevormd naar een abstract syntax tree, afgekort AST.
Een AST bevat een schematische boom van onze gedefinieerde classes, methodes en variabelen, maar bevat deze ook informatie over waar in het bestand de code staat die bij een Node hoort. Zo kun je precies herleiden in welke rij en kolom van het bestand een methode of variabele is gedefinieerd.
Aan de hand van deze informatie kun je zelf programma's maken om code te controleren en/of suggesties te tonen. Ook kun je aan de hand van een AST automatisch code genereren en aanpassen. Zo kun je repetitieve zaken geautomatiseerd aanpakken zonder bang te hoeven zijn voor knip en plak fouten.
Kortom, AST's helpen jou met het automatiseren van repetitief werk en het gebruik ervan voorkomt fouten die kunnen ontstaan door handmatig code schrijven.

Aan de slag

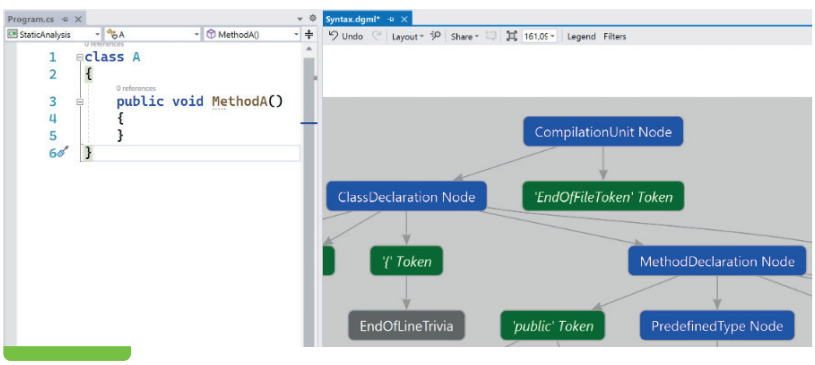
Nu denk je misschien, hoe ziet zo'n AST er nou uit en hoe werk ik ermee? In dit segment gaan we kijken naar tools in het Microsoft landschap die jou bij deze twee vraagstukken helpen. **Afbeelding 1**.
In het Microsoft landschap is Roslyn de open-source C# en Visual Basic compiler. Roslyn biedt naast compilers ook APIs om met deze compilers te werken. Zo is er een API om code om te zetten naar een AST. Roslyn kan als C# library aan .NET projecten worden toegevoegd.
Naast de Roslyn C# library zijn er componenten voor Visual Studio om het werken met Roslyn makkelijker te maken. Zo is er de Syntax Graph en Syntax Visualiser om AST's te visualiseren. Beide kunnen geïnstalleerd door in de Visual Studio Installer onder het "Individual Components" tabblad de volgende componenten te installeren:

De eerste AST

Voor onze eerste introductie met een AST, beginnen we met het bekijken van een stuk C# code in Visual Studio. Naast de standaard editor is Syntax Graph geopend, deze visualiseert de AST van de geschreven code. Als je wijzigingen maakt aan de code is dit meteen in de AST visualisatie terug te zien. **Afbeelding 2**.
De AST bestaat uit Nodes, elk van deze Nodes representeert een deel van de code. Zo is er een ClassDeclaration welke een class definitie



Afbeelding 1 - Visual Studio componenten voor Roslyn ontwikkelen.



Afbeelding 2 - Voorbeeld Abstract Syntax Tree voor C# code.

representeert en is er een MethodDeclaration die een methode representeert. Zoals de naam "Abstract Syntax Tree" doet vermoeden zijn deze Nodes gestructureerd als takken van een boom.
Dit betekent dat elke Node eigen Nodes onder zich kan hebben. Deze worden in de Members property bijgehouden. Bij een C# programma is de CompilationUnit de root Node, helemaal bovenaan de AST. In het code voorbeeld heeft de CompilationUnit Node ook weer Nodes onder zich zoals ClassDeclaration en deze heeft op zijn beurt dan weer MethodDeclaration Nodes onder zich.

Gegevens uit de AST halen

Nu we hebben gezien hoe een AST is opgebouwd, gaan we kijken hoe we er informatie uit kunnen halen. Omdat de AST als een boom met takken is gestructureerd kunnen we namelijk niet zomaar door een lijst van alle Nodes lopen. We moeten er doorheen navigeren.
Voordat we kijken naar de best practice voor het doorlopen van een AST, kijken we eerst naar een basisopzet. In deze opzet construeren we een AST en halen we de namen van de gedefinieerde classes op. Het opstellen van de AST kan met **CSharpSyntaxTree.ParseText**. Afbeelding 3.

```
var programText = @"
    class FirstClass {
        public void MethodOne() {}
    }";

var tree = CSharpSyntaxTree.ParseText(programText);
var compilationUnit = tree.GetCompilationUnitRoot();

foreach (var compilationUnitChild in compilationUnit.Members)
{
    if (compilationUnitChild is ClassDeclarationSyntax node)
    {
        var name = node.Identifier.Text;
        Console.WriteLine($"Found class with name: {name}");
    }
}
```

Afbeelding 3 - C# code voor opbouwen en uitlezen AST.

```
var programText = @"
    class FirstClass {
        public void MethodOne() {}
    }";

var tree = CSharpSyntaxTree.ParseText(programText);
var compilationUnit = tree.GetCompilationUnitRoot();

foreach (var compilationUnitChild in compilationUnit.Members)
{
    if (compilationUnitChild is not ClassDeclarationSyntax classNode) continue;

    foreach (var classChild in classNode.Members)
    {
        if (classChild is not MethodDeclarationSyntax methodNode) continue;

        var name = methodNode.Identifier.Text;

        Console.WriteLine("Found node: " + name);
    }
}
```

Afbeelding 4 - Voorbeeld constructie om Nodes op verschillende niveaus uit de AST uit te lezen.

In bovenstaande voorbeeld bouwen we een AST op basis van C# code in de programText. De ParseText methode bouwt de AST op en geeft de Node bovenaan de boom terug. Dit is bij C# programma's de CompilationUnit Node. Onder deze Node vallen alle andere Nodes van het programma.
Via de Members property van de CompilationUnit Node kunnen we de directe kinderen van de Node uitlezen. In dit geval is dat een ClassDeclaration Node, deze Node heeft de Identifier property waarmee we de naam van de class kunnen uitlezen. Omdat de Members property van een Node alleen zijn directe kinderen bevat, vinden we alleen ClassDeclarations. Als we ook de MethodDeclaration willen ophalen, moeten we ook door de Members van elke ClassDeclaration lopen. Dit zou er als volgt uit zien: Afbeelding 4.
In dit voorbeeld valt het op dat we voor wat basisgegevens ophalen al veel code moeten schrijven. Ook maakt deze code veel aannames, zo werkt deze logica momenteel alleen als het programma geen namespace

Visual Studio geeft ons bijvoorbeeld suggesties met de gedefinieerde classes en methodes in de vorm van IntelliSense

```
public class DemoASTWalker : CSharpSyntaxWalker
{
    public override void VisitClassDeclaration(ClassDeclarationSyntax node)
    {
        Console.WriteLine($"Found class with name: {node.Identifier.Text}");

        base.VisitClassDeclaration(node);
    }

    public override void VisitMethodDeclaration(MethodDeclarationSyntax node)
    {
        Console.WriteLine($"Found method with name: {node.Identifier.Text}");

        base.VisitMethodDeclaration(node);
    }
}
```

Afbeelding 5 - Voorbeeld implementatie CSharpSyntaxWalker.

```
var programText = @"
    class FirstClass {
        public void MethodOne() {}
    }
    ";

var tree = CSharpSyntaxTree.ParseText(programText);
var walker = new DemoASTWalker();

var root = tree.GetRoot();

walker.Visit(root);
```

Afbeelding 6 - Voorbeeld gebruik geïmplementeerde CSharpSyntaxWalker.

declaratie bevat. Dit zou namelijk betekenen dat de NamespaceDeclaration het enige kind is van CompilationUnit, de ClassDeclaration's zijn namelijk dan weer kinderen van NamespaceDeclaration.
Ook gaat het voorbeeldprogramma maar tot twee niveau's diep. Als de MethodDeclaration kinderen zou hebben worden deze nooit langsgelopen omdat hiervoor een extra forloop voorgeschreven moet worden. Kortom, de AST doorlopen met for loops is vaak foutgevoelig en maakt je code lastiger te begrijpen, onderhouden en uit te breiden. Om dit probleem te verhelpen levert de Roslyn library de CSharpSyntaxWalker class aan. Deze class loopt voor jou door de AST en laat je gemakkelijk gegevens uit de AST halen.

Werken met CSharpSyntaxWalker

Door een eigen class de CSharpSyntaxWalker class te laten implementeren kun je aan de hand van het Visitor design pattern door de AST lopen. Bij het Visitor design pattern worden Nodes in een Tree data structuur doorgelopen, elke keer als een Node wordt bezocht wordt de Visit methode aangeroepen.

Bij het schrijven van logica kunnen we zelf een Visit methode implementeren. Zo kunnen we een VisitClassDeclaration methode implementeren die aangeroepen wordt als de CSharpSyntaxWalker een ClassDeclaration "visit" in de AST. **Afbeelding 5 & Afbeelding 6**. Bovenstaande code bereikt hetzelfde doel als het eerste voorbeeld met de for loops. Echter is de code met CSharpSyntaxWalker flexibeler doordat er geen aannames worden gedaan over de structuur van het programma. Hierdoor blijft analyse logica werken ook al worden ClassDeclaration's of MethodDeclarations verplaatst in de te analyseren code.

Gegevens van Nodes bijhouden met CSharpSyntaxWalker

Nu we met de CSharpSyntaxWalker door de AST kunnen lopen gaan we bezig met het schrijven van analyse logica. Hiervoor willen we informatie uit de AST verzamelen, doordat de DemoASTWalker een standaard class is kunnen we met een standaard C# constructie gegevens verzamelen.
In onderstaande voorbeeld maken we gebruik van een Data Transfer Object (DTO) om waardes bij te houden. Deze DTO geven we door

```
public class NodeCountResult
{
    public int AmountOfClasses { get; set; }
}

public class DemoASTWalker : CSharpSyntaxWalker
{
    private readonly NodeCountResult _results;

    public DemoASTWalker(NodeCountResult results)
    {
        _results = results;
    }

    public override void VisitClassDeclaration(ClassDeclarationSyntax node)
    {
        _results.AmountOfClasses++;

        base.VisitClassDeclaration(node);
    }
}
```

Afbeelding 7 - Voorbeeld implementatie gegevens bijhouden.

in de constructor en slaan we op in het _results veld. In de analyse logica zoals VisitClassDeclaration kunnen we waardes schrijven naar de DTO. **Afbeelding 7**.
In onderstaand stuk code geven we de referentie naar de NodeCountResult instantie mee in de constructor. Vervolgens lopen we door AST met de ASTWalker via walker.Visit(root), tijdens het doorlopen wordt de DTO gevuld.
Hierna kunnen de resultaten via de countResult object referentie worden uitgelezen. In dit geval bewaren we een numerieke waarde, maar dit kunnen ook complexe types zoals IEnumerable zijn. **Afbeelding 8**.
Aan de hand van deze opzet met de CSharpSyntaxWalker en de DTO kun je zelf analyse logica schrijven. Deze logica kan zo simpel of uitgebreid zijn als je zelf wilt.

Code aanpassen via CSharpSyntaxRewriter

Nu we hebben gezien hoe we met CSharpSyntaxWalker C# programma's kunnen analyseren, gaan we kijken hoe we bestaande code kunnen aanpassen met Roslyn. In de voorbeelden gaan we logica schrijven om private velden in classes altijd met een underscore te laten starten. Terwijl het mogelijk is om dit soort code wijzigingen met "zoeken en vervangen" of regex te doen, kan dit resulteren in code die niet compileert of ongevraagde code wijzigingen. In onderstaand voorbeeld is te zien dat ook het commentaar wordt aangepast terwijl we alleen de class velden willen aanpassen. **Afbeelding 9**.
Omdat Roslyn semantisch de code begrijpt weet Roslyn het verschil tussen een class veld en een stuk commentaar. Hierdoor kun je veilig geautomatiseerd code aanpassen met veel minder kans op onjuiste resultaten. Om dit te demonstreren gaan we met de CSharpSyntaxRewriter class van Roslyn private velden automatisch prefixen met een underscore. Hieronder is het gewenste verschil uitgewerkt: **Afbeelding 10**.
De CSharpSyntaxRewriter lijkt in veel opzichten op de CSharpSyntaxWalker, zo hebben ze beide methodes zoals VisitVariableDeclarator die je kan overschrijven. Het grootste verschil is dat je bij CSharpSyntaxRewriter een aangepaste versie van een AST Node kan teruggeven. Deze bijgewerkte Node wordt dan in de AST geplaatst en zorgt uiteindelijk voor de gevraagde code-wijzigingen. Hieronder is een voorbeeld implementatie voor het toevoegen van underscores uitgewerkt: **Afbeelding 11**.
In bovenstaand voorbeeld lopen we door alle field declarations van de ingelezen code heen. Per field declaration kijken we of deze private is, en als de declaration private is, halen we de eerste Variable Declaration binnen die Field Declaration op.
Hierbij kijken we of de huidige veldnaam al een underscore heeft. Als dit niet het geval is, voegen we een underscore toe aan de veldnaam.
Hierbij is het goed om te weten dat Nodes in een AST niet direct aangepast kunnen worden.

```
var programText = @"
    class FirstClass {
        public void MethodOne() {}
    }
    ";

var tree = CSharpSyntaxTree.ParseText(programText);

var countResult = new NodeCountResult();
var walker = new DemoASTWalker(countResult);
var root = tree.GetRoot();

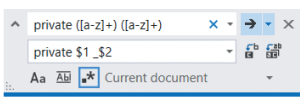
walker.Visit(root);

Console.WriteLine(countResult.AmountOfClasses); // 1
```

Afbeelding 8 - Voorbeeld gebruik geïmplementeerde node gegevens.

```
namespace Example
{
    class FirstClass
    {
        private int counter = 3;

        /// <summary>
        /// This method uses a private field value
        /// </summary>
        public void MethodOne()
        {
        }
```



Afbeelding 9 - Voorbeeld scenario waarin het gebruik van Regex ongewenste resultaten oplevert.

```
class FirstClass
{
    public string Name;
    private int counter = 3;
    private int secondCounter = 5;
    private bool _alreadyPrefixed = true;
}
```

```
class FirstClass
{
    public string Name;
    private int _counter = 3;
    private int _secondCounter = 5;
    private bool _alreadyPrefixed = true;
}
```

Afbeelding 10 - Code aanpassing die we automatisch willen doorvoeren.

```
public class CodeRewriterV1 : CSharpSyntaxRewriter
{
    public override SyntaxNode VisitFieldDeclaration(FieldDeclarationSyntax node)
    {
        if (!node.Modifiers.Any(m => m.IsKind(SyntaxKind.PrivateKeyword)))
        {
            return node;
        }

        var variableDeclaration = node.Declaration.Variables[0];
        var currentName = variableDeclaration.Identifier.ValueText;

        if (currentName.StartsWith("_")) return node;

        var updatedName = $"_{currentName}";

        var updatedVariable = variableDeclaration
            .WithIdentifier(SyntaxFactory.Identifier(updatedName));

        return node.ReplaceNode(variableDeclaration, updatedVariable);
    }
}
```

Afbeelding 11 - Voorbeeld implementatie van CSharpSyntaxRewriter die namen van class velden aanpast.

```
var programText = @"
    class FirstClass {
        public string Name;
        private int counter = 3;
        private int secondCounter = 5;
        private bool _alreadyPrefixed = true;
    }
    ";

var tree = CSharpSyntaxTree.ParseText(programText);
var rewriter = new CodeRewriterV1();

var root = tree.GetRoot();
var updatedRoot = rewriter.Visit(root);

Console.WriteLine(updatedRoot.ToFullString());
```

Afbeelding 12 - Voorbeeld hoe een CSharpSyntaxRewriter implementatie gebruikt kan worden voor een stuk code.

```
class FirstClass {
    public string Name;
    private int _counter= 3;
    private int _secondCounter= 5;
    private bool _alreadyPrefixed = true;
}
```

Afbeelding 13 - Resulteerde code nadat de CodeRewriterV1 de code heeft aangepast.

BIO

Maarten Grootoonk

Maarten Grootoonk is een Software Engineer en trainer bij Ordina die zich specialiseert in het Microsoft landschap. Zo ontwikkelt hij .NET applicaties voor de klant en geeft hij verschillende Azure trainingen en workshops.



We moeten een kopie maken van de Node en deze aanpassen.
Om dit proces te versimpelen levert Roslyn verschillende **With...** Methodes aan. Met deze methodes kun je makkelijk een kopie van de Node maken en deze kopie aanpassen. In dit voorbeeld vervangen we de Identifier met de nieuwe naam van de field declaration.
Als laatste stap vervangen we de oude Node zonder underscore met de nieuwe Node die wel een underscore in de naam heeft. Dit doen we door op de hoofd node(FieldDeclaration) de ReplaceNode methode aan te roepen. Deze methode vervangt de node zelf of een van z'n onderliggende Nodes.

AST omzetten naar Code.

Nu we hebben gezien hoe code aangepast kan worden gaan we kijken hoe je een AST omzet naar code.
Dit kan worden gedaan door de ToFullString methode aan te roepen op de Node die Visit() terug geeft: **Afbeelding 12**.
Bovenstaande programma levert de volgende Console output op: **Afbeelding 13**
Hierin is te zien dat de namen van de private velden zijn aangepast en de velden die al een underscore hadden niet zijn aangepast. Dit is precies het gewenste resultaat, ook weten we zeker dat we geen onjuiste resultaten krijgen omdat Roslyn de semantiek van de code begrijpt.

Zelf aan de slag

We hebben nu gezien hoe je met de Roslyn API analyses kan uitvoeren en bestaande code kan aanpassen. In de voorbeelden zijn een aantal basis scenario's uitgewerkt om een idee te geven wat je met Roslyn kan. Om meer te weten te komen over Roslyn en de API is het volgende artikel een goed startpunt: **https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/.**

"Vitas is een nuchtere informele club van gedreven Microsoft professionals waar volop ruimte is voor inhoudelijke groei en iedereen zichzelf kan zijn. Inhoudelijk mooi werk in een fijne omgeving is waar we het voor doen."



MICROSOFT ENTHUSIASTS

We houden ons bezig met cloud transities met hoofdzakelijk Azure. We zetten actief in op de professionele en persoonlijke ontwikkeling van onze collega's, waarbij persoonlijke aandacht voorop staat.

- FIJNE EN ONGEDWONGEN CULTUUR
- HYBRIDE WERKEN
- FLEXIBELE WERKTIJDEN
- BOVENGEMMIDELDE ARBEIDSVOORWAARDEN
- ONBEPERKT MOGELIJKHEID TOT ONTWIKKELING
- HOOG KENNISNIVEAU EN SENIORITEIT
- SOCIALE TEAMEVENEMENTEN
- EN NOG VEEL MEER

SCAN DE QR CODE VOOR MEER INFORMATIE EN SOLLICITEER DIRECT

MICROSOFT ANGULAR AZURE .NET C#

VITAS

Jasper Sprengers

Doorgeschoten vakmanschap

Eind juni gaat naar verwachting de James Webb telescoop in bedrijf. De opvolger van de Hubble-telescoop was dertig jaar in de maak en kostte tien miljard dollar. De software die straks op 1,5 miljard km van de aarde gaat draaien zal met gekmakend oog voor detail en kwaliteit gebouwd zijn – mag ik aannemen.

Kwaliteit is duur, dus maken we risico-inschattingen en stemmen daar de kwaliteitsbehoefte op af. Hoe groot is de schade als er straks cruciale fouten in de software blijken te zitten? Hoe verkleinen we de kans daarop? Hoe snel kunnen we zulke fouten opsporen en repareren? Bij de Webb telescoop mag het antwoord duidelijk zijn: denk alleen al aan de reputatieschade. Als ernstige bugs einde verhaal kunnen betekenen is een mega-investering in *first time right* gerechtvaardigd. Voor de meeste andere projecten geldt dat zelden.

Wie jarenlang aan de Webb telescoop heeft gesleuteld is waarschijnlijk een clean code purist. In die gemeenschap geldt een principiële liefde voor kwaliteit als het ware kenmerk van vakmanschap. Misschien komt dat wel doordat ons beroep vrij toegankelijk is voor amateurs. Iedereen mag zich ontwikkelaar noemen, in tegenstelling tot tandarts of advocaat. De ruimte voor vakmanschap is grenzeloos als je alle aspecten van softwarekwaliteit even serieus neemt. Die gaat verder dan enkel stabiliteit en correctheid: *readability, adaptability, usability, portability* en tig andere '*ilities*'.

Er bestaat echter wel degelijk ook doorgeschoten vakmanschap, en dat vind je bij clean coders in projecten waar topkwaliteit geen vereiste is. Soms is een zeven goed genoeg. De techneuten kunnen zich er slecht naar schikken dat de prijs voor een negen niet te verantwoorden is, maar dat is dan jammer. Zij betalen niet de rekening. Vakmanschap kan niet enkel gefocust zijn op het maximaliseren van kwaliteit. Je moet ook optimaliseren.

De clean coder pur sang heeft een broertje dood aan zulke concessies, trekt zich weinig aan van deadlines, of werkt zich in een burn-out om het werk goed én op tijd af te krijgen. Dat is geen zegen voor de groep. Perfectionisten zijn vaak matige teamspelers en bepaald niet gezellig in de omgang. Sterrenkok Sergio Herman beschreef althans zichzelf zo in de documentaire net de veelzeggende naam *Fucking Perfect* (2015).

Wie kwaliteit boven klantbelang of opleverdatum stelt benadert het vak als kunst. Kunstenaarschap is een kwestie van instelling, of je nu werkt met penseel, pen of C#. De kunstenaar is ten eerste loyaal aan de liefde voor het vak en hoe je dat hoort te bedrijven, ongeacht wat je bouwt. In een werknemersrelatie zal zij pas onder protest en met tegenzin iets opleveren wat niet aan haar standaard voldoet.

Ik beschrijf hier natuurlijk archetypes. Perfectionisme versus pragmatisme is een spectrum, en een gebalanceerd team heeft ook professionals die vooral loyaal zijn aan de opdrachtgever en voor wie *best practices* niet heilig zijn. Goede pragmatici zijn zeker geen prutsers en werken ook hard en gemotiveerd. Maar kwaliteit is voor hen wel rekbaar en onderhandelbaar, dus moet je ze in hun tomeloze energie om de klant blij te maken er wel af en toe aan herinneren netjes te blijven werken.



Jasper Sprengers

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



code & comedy

The conference for and by developers where code and humor come together.

2022 theme Robotics

June 16th, 2022 | Ordina HQ, Nieuwegein

Sign up now: www.codeandcomedy.com

ORDINA
Ahead of change