ISSN: 2211-6486

SDN MAGAZINE KNOW-LEDGE #138

Nummer 138 Juli 2020 SDN Magazine verschijnt elk kwartaal en is een uitgave van Software Development Network

SDN Magazine is hét lijfblad van en voor programmeurs, met artikelen over ontwikkelen voor Windows, Web, Mobile of Cloud. Of u nu ontwikkelt voor .NET, Delphi of iets anders, als lid van het SDN weet u alles.

ONDERWERPEN:

IN DIT NUMMER BIJDRAGEN VAN O.A.:

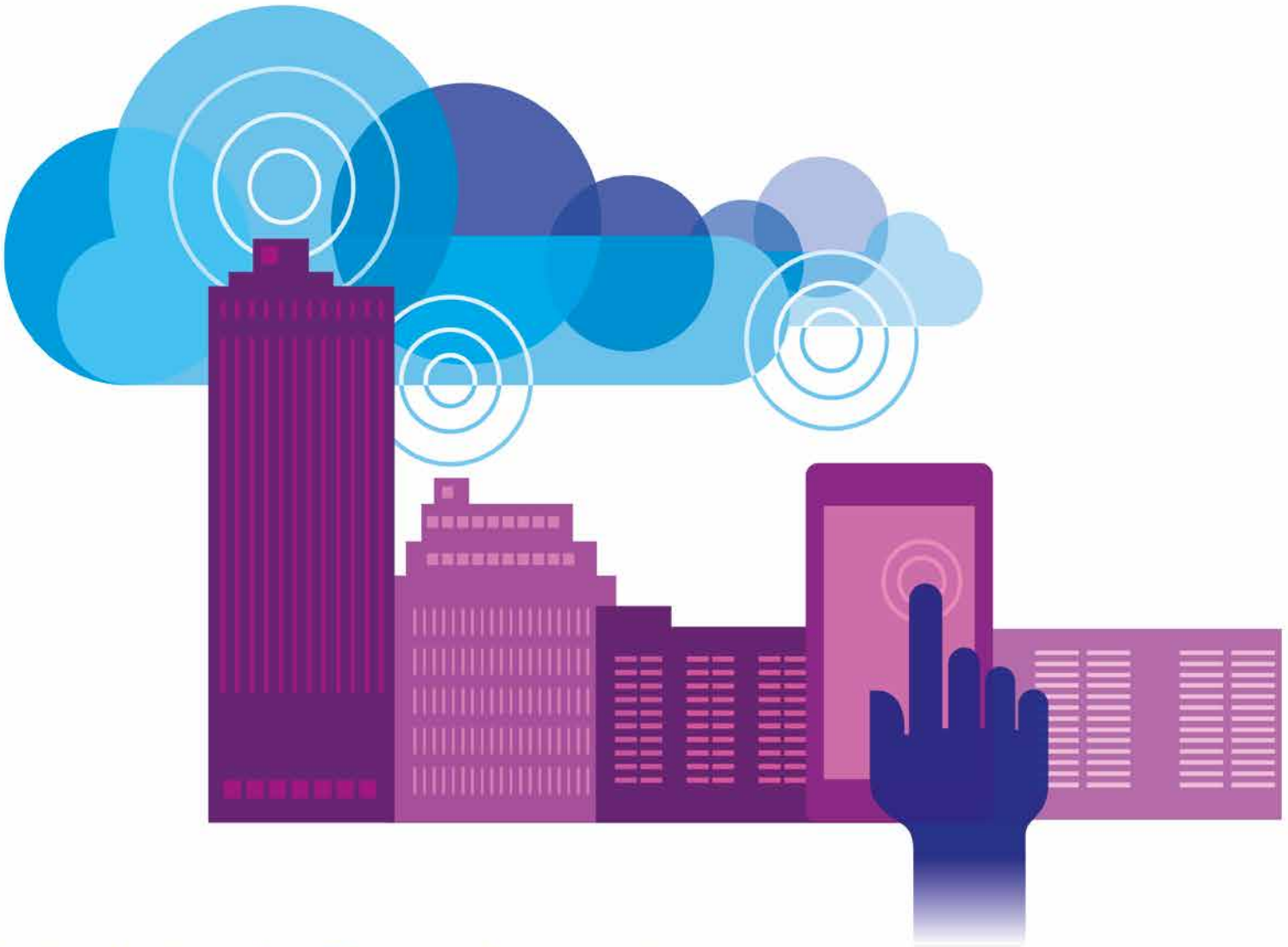
Menno Jongerius

Arjen Kraak

Jan de Vries

Christiaan Nieuwlaat

Visual Studio



Verbind. Synchroniseer. Transformeer de wereld van apps.

Welkom in het tijdperk van buitengewone verwachtingen. Want elke app moet altijd soepel op ieder apparaat kunnen werken. Zonder haperingen en problemen. Visual Studio helpt ontwikkelaars te presteren met de meest geavanceerde én geïntegreerde oplossing die er op de markt is: een ultramoderne set met tools en services die u helpen bij het ontwikkelen, testen en implementeren van servicegerichte apps. De gebruikservaring wordt moeiteloos getransformeerd op alle Windows-apparaten.

Ontdek Visual Studio
www.visualstudio.com

Het zijn mooie tijden voor ontwikkelaars

Microsoft

voorwoord

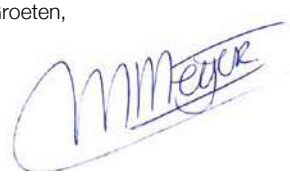
Beste SDN Magazine lezer,

Het jaar 2020 is alweer halverwege en wat een raar jaar! Ineens stond ons hele leven op de kop en gingen we allemaal in een intelligente lockdown. Dit had een enorme impact voor vele beroepsgroepen, zoals sportscholen, kappers, cafés en restaurants etc. Ook de IT-community met zijn vele meetups, in-person events en conferenties moesten omschakelen. Een aantal grote events werden definitief verplaatst naar volgend jaar en een aantal naar later dit jaar, maar een aantal gingen als een online event verder. Zoals bijvoorbeeld Microsoft Build en Apple WWDC. Voordeel van deze online events: er waren bezoekers, die anders onmogelijk aanwezig konden zijn. Wij hebben ons FutureTech event ook moeten verplaatsen naar het najaar van 2020, hopelijk lukt het dan. Ook onze SDN-events van april en juni hebben we helaas moeten cancelen, maar niet getreurd, dit najaar komen we zeker terug of we doen weer een online event.

Voor je ligt het laatste SDN-magazine en dit magazine is als fysiek magazine naar je verstuurt. In dit magazine artikelen over: Introductie naar Domain Driven Design (DDD), Wat doet een Lead Developer, Van Monolithische Applicatie naar Microservices, Kan Cloud Native On-Premises zijn, Multi-Stage Pipelines in Azure gebruiken, Avoiding defensive copies of Structs in C#, Gebruik Managed Identities Voor Inter-Service Communicatie, Hello Blazor, Valkuilen in async/await en Microsoft ML.NET with Jupyter Notebooks. Onze collega's uit het community Arjen Kraak, Mark Monster, Christiaan Nieuwlaat, Patrick Bes, Eduard Keilholz, Fons Sonnemans, Jan de Vries, Gill Cleeren, Menno Jongerius en Daniel Costea geven allemaal een inkijkje in hun ervaringen over de verschillende onderwerpen. Bedankt auteurs, super waardevolle verhalen.

Veel leesplezier en tot ons volgende event of uitzending van de SDN Cast Rebooted. Wil je schrijven of spreken of meedoen aan onze cast meld je dan op redactie@sdn.nl.
Namens de SDN medewerkers alvast fijne vakantie!

Groeten,



Marcel Meijer
Eindredacteur Magazine en Voorzitter SDN

SDN MAGAZINE KNOW-LEDGE#138

Over Marcel Meijer:
Marcel Meijer is voorzitter, bestuurslid, event organisator en eindredacteur bij de SDN, vast teamlid bij de SDN Cast, onderdeel van het Content team bij Techorama, betrokken bij de organisatie van FutureTech en sinds 2010 MVP.

Colofon

Uitgave
Software Development Network
Achtentwintigste jaargang
No. 138 • juli 2020

Bestuur van SDN
Marcel Meijer, voorzitter
Rob Suurland, penningmeester
Roy Janssen, secretaris

Redactie
Marcel Meijer (redactie@sdn.nl)

Aan dit magazine werd meegewerkt door
Aan dit magazine werd meegewerkt door Bob Swart, Maarten van Stam, Arjen Bos, Roy Janssen, Rob Suurland, Marcel Meijer en natuurlijk alle auteurs!

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

Contact
Software Development Network
Postbus 506 7100 AM Winterswijk
Tel. (085) 21 01 310
info@sdn.nl

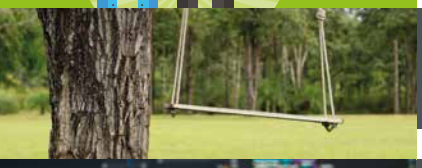
Vormgeving en opmaak
Reclamebureau Bij Dageraad
Winterswijk www.bijdageraad.nl

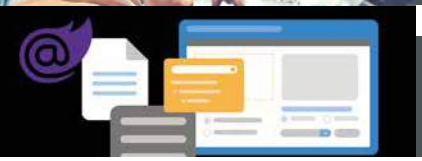
Adverteerders

Adverteren
Informatie over adverteren en de advertentietarieven kunt u vinden op www.sdn.nl onder de rubriek Magazine.

SDN MAGAZINE KNOW-LEDGE #138

Inhoud





Arjen Kraak

Introductie naar Domain Driven Design (DDD)

Als Lead Developer ben ik betrokken bij het opmaken van business cases, uitwerken van ideeën en het implementeren van softwareoplossingen. Een steeds vaker terugkomend thema is dat mijn relaties hun applicatie landschap willen moderniseren zonder opnieuw het wiel te willen uitvinden.Vaak wil men voortbouwen op het goede wat er al is. Meestal is het geen goed idee om de optie “opnieuw beginnen” ter tafel te brengen. Maar hoe kunnen we onze klanten dan wel op weg helpen?

Het opnieuw beginnen is dus geen optie. Werken met een “brown field” is dus het beginpunt. De aanpak waarmee ik start is het in kaart brengen van het probleemgebied. Hierbij wordt het bestaansrecht van de applicatie (het waarom en wat) onderzocht. Via dit model wordt vervolgens onderzocht hoe de applicatie opgeknipt kan worden op basis van thema’s in dit model. Dit ontvlechten van de applicatie wordt in software termen vaak het “ontvlechten van de monoliet” genoemd.In dit artikel, deel ik graag een methodiek die tegenwoordig heel actueel en toepasbaar is. Deze methodiek is Domain Driven Design (DDD).
Helaas is het zo dat bij het beschrijven van deze methodiek (te) snel de route van abstractie en technologische gevolgen wordt bewandeld. Ik pak dat deze keer anders aan.
Zodra je googled op “DDD” krijg je een spervuur van termen, toepassingen en dergelijke tot je beschikking. Hieronder een woordwolk van veel gebruikte termen.

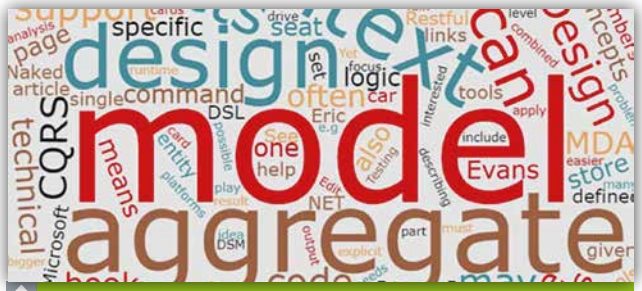


FIGURE 1

Verschillende termen uit deze woordwolk zullen aandacht krijgen in dit en erop volgende artikel(en). Laten we beginnen bij het begin.

HET SWINGTREE PROJECT

Software ontwikkelen is een complexe inspanning die vaak leidt tot onbegrip. Dit onbegrip komt vooral voort uit het vertalen van de wens (het idee) naar de oplossing. Met alle goede bedoelingen en processen (“Scrum“, “Agile” !), blijkt jammer genoeg in de praktijk dat het resultaat lijkt op het beroemde “Swingtree project”.

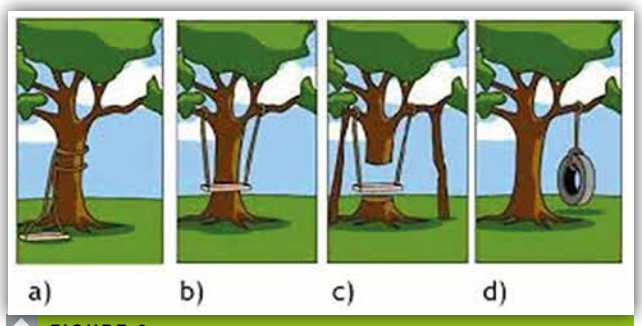


FIGURE 2

Het “Swingtree project” is een metafoor die regelmatig wordt getoond om aan te geven dat er wezenlijke verschillen bestaan in begrip:

- Wat de gebruiker / klant heeft gedeeld over de wens
- Wat de gebruiker / klant werkelijk nodig heeft
- Hoe de architect de wens heeft vertaald naar het ontwerp
- Hoe de ontwikkelaar het ontwerp heeft geïmplementeerd.

Uiteindelijk blijft het ontwikkelen van software mensenwerk. Mensenwerk vereist veel investeren in communicatie en afstemming. Vooral in de fase van een project waar nog geen regel code is geschreven moet er veel worden gecommuniceerd en gedocumenteerd om zoveel mogelijk te voorkomen dat uiteindelijk een product wordt opgeleverd wat lijkt op de schommel. De methodiek Domain Driven Design is zeer bruikbaar zowel in de verkennende fase van een project, als in de realisatie.

WAAROM KIEZEN VOOR DOMAIN DRIVEN DESIGN?

Onderaan de streep is het de bedoeling dat door middel van een gemeenschappelijke taal (“Ubiquitous language“) een ontwerp (“Domain Model”) ontstaat waarin middels afbakening van thema’s (“Bounded Context”) de onderwerpen (“Entities”) worden beschreven. Kernwaarde hierbij is dat dit wordt gedaan zonder verwijzing naar de oplossing zelf (technologie en software architectuur). Domain Driven Design gaat dus over de wens / probleem en niet over de oplossing.

WAT IS EEN “PROBLEM DOMAIN”?

De term “probleemdomein” wordt vaak aangeduid in DDD voor het definieren van een functioneel gebied in een organisatie. Een applicatie zal nimmer alle processen ondersteunen van de organisatie. Denk bijvoorbeeld aan applicaties voor personeelszaken en inkoop. Organisaties van enige omvang hebben hiervoor altijd verschillende applicaties. Personeelszaken is bijvoorbeeld een “probleemdomein”.

ER WORDT EEN “DOMAIN MODEL” OPGESTELD. WAT IS EEN “DOMAIN MODEL”?

Een groot denker en pionier op het terrein van Domain Driven Design, Eric Evans zegt over het “model” het volgende: *“every model represents some aspect of reality or an idea of interest. A model is a simplification. It is an interpretation of reality that abstracts the aspects relevant to solving a problem at hand and ignores extraneous detail”*

Introductie naar Domain Driven Design (DDD)

Een architect en/of analist faciliteert het opstellen van het model. Wat heb je dan? Het model beschrijft het “uiterlijk” en “gedrag” van het gekozen probleem domein. Het “uiterlijk” is de verzameling van gegevens die van belang zijn. Hierbij is ook de relatie tussen deze gegevens interessant. Het “gedrag” beschrijft de processtappen en gebeurtenissen die met/op deze gegevens plaatsvinden. Je kan het ook menselijk maken. Het gedrag is wat van buitenaf wordt gezien. Dit gedrag heeft intentie/motivatie, dat zijn de gegevens. Binnen DDD wordt gedrag beschreven als een verzameling van “events” en de intentie als een verzameling van “entiteiten”. De verzameling van deze beschrijvingen worden geborgd in tekeningen en code.

WAT IS HET DOEL VAN HET DOMAIN MODEL?

In het begin van dit artikel wordt “swingtree project” beschreven. Het middel om dit soort projecten te voorkomen is het domain model. Dit is de “why” van het modelleren. Het model levert documentatie in de taal van de kenniswerker in het probleemdomein. Dit betekent dat het model in een taal is geschreven zonder technische oplossingen en begrijpbaar voor de kenniswerker en andere betrokkenen. Concreet, er moeten termen worden gebruikt van de “werkvloer”. Als meerdere termen voor hetzelfde onderwerp worden gebruikt, moet de best passende term worden gekozen.
Het primaire doel van het model is dus het beschrijven van de “why” en “what” van het probleemdomein.

Door middel van het model wordt:
1. Kennis over het probleemdomein geborgd
2. Spreken toekomstige gebruikers en de ontwikkelaars dezelfde taal
3. De oplossing in code herkenbaar, ook in de toekomst

Een secundaire doelstelling is het hebben van een basis voor het software ontwikkelteam om de oplossing te realiseren van het probleemdomein. Deze basis moet in de code worden herkend zodat ontwikkelaars snel vanuit het probleemdomein, de technische oplossing eigen kunnen maken. Op deze manier wordt code herkenbaar en wordt alleen noodzakelijke code geschreven. Binnen DDD wordt de taal van de werkvloer “ubiquitous language” genoemd.

EEN GESTRUCTUREERDE AANPAK

Zonder het gestructureerd aanpakken van de analyse om tot het model te komen, kan een applicatie ontstaan zoals hieronder weergegeven. Voor een klein probleemdomein hoeft dat geen probleem te zijn. Echter, het probleem domein zal altijd onderhevig zijn aan veranderingen en wellicht uitbreidingen. Het resultaat is een nog complexer plaatje. Binnen DDD worden dergelijke oplossingen “Big ball of mud” genoemd.

Beter is om deelgebieden van het probleemdomein te beschrijven. Dit zijn de “subdomains” van het probleemdomein. Per subdomein worden vervolgens de modellen opgesteld. Dit betekent ook dat de oplossing (de architectuur) bestaat uit subdomeinen. Het realiseren van de oplossing gebeurt met focus en kan zelfs door verschillende teams worden uitgevoerd. De applicatie resulteert dan in een plaatje wat vergelijkbaar is met de onderstaande.

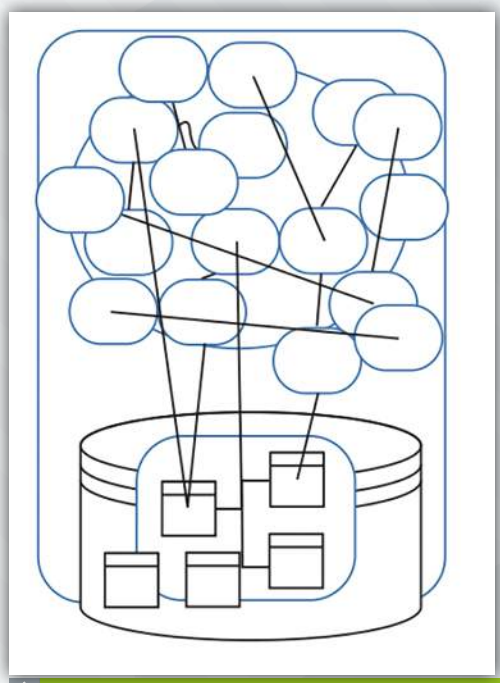


FIGURE 3

Bron: Patterns, Principles and Practices of Domain Driven Design by: Millet & Tune

AGGREGATES, ENTITEITEN EN VALUE OBJECTEN

Om het begrip rond aggregates, entiteiten en value objecten goed te begrijpen is begrip nodig van de zogenaamde “life cycle” van objecten in het algemeen. De DDD manier beschrijft de status van een object zijnde: “actief”, “verwijderd” of “gearchiveerd”.

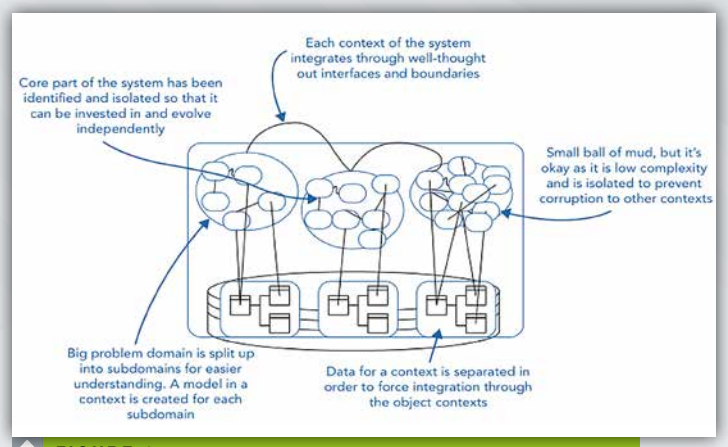


FIGURE 4

Bron: Patterns, Principles and Practices of Domain Driven Design by: Millet & Tune

Een “actief” object is in het geheugen geladen (bijvoorbeeld uit een database) en wordt actief mee gewerkt of gebruikt om entiteiten binnen een bounded context te benaderen.
Een “verwijderd” object krijgt deze status eigenlijk middels een “onzichtbaar” vlag omdat entiteiten eigenlijk nooit verwijderd worden (is overigens een uitdaging in het perspectief van GDPR en “mij vergeten”).

Een “gearchiveerd” object is opgeslagen en niet geladen in het geheugen van de applicatie.
Bovenstaande is door Eric Evans grafisch weergegeven in zijn boek “Domain Driven Design, tackling the complexity in the heart of software“.

Moderne software oplossingen bevatten snel diepgaande structuur van objecten middels verschillende lagen. In aanvulling hierop worden software oplossingen steeds meer gebruikt via verschillende apparaten

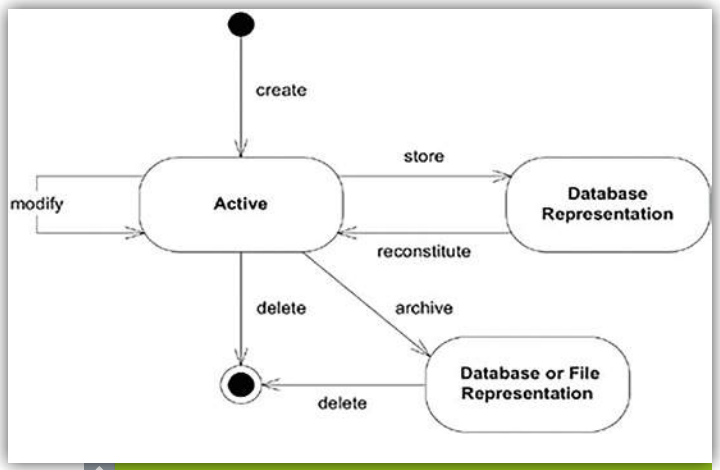


FIGURE 5

en kunnen deze draaien op verschillende besturingssystemen. Veel gebruikers muteren gegevens tegelijkertijd hierdoor. De uitdaging is daarom om de waarheid van de objecten te garanderen. Deze consistentie van veranderingen “eventual consistency” is DE uitdaging van DDD gedreven oplossingen.

Consistentie wordt bepaald via een verzameling van regels, in DDD termen “invariants” genoemd. Deze “invariants” moeten op 1 plek worden vastgelegd. Zij bepalen de samenhang van de objecten in de bounded context en zorgen ervoor dat 1 waarheid blijft bestaan. Die ene plek is het hoofd object in de bounded context. Dit hoofd object wordt de aggregate root genoemd. Doordat dit type object verantwoordelijk is om de “invariants” te bewaken, moet het de rol spelen van de “poortwachter” voor de bounded context.
In andere woorden, alleen via de aggregate root kan de bounded context en haar inhoud worden aangesproken en worden gewijzigd.

AGGREGATES EN HUN BOUNDED CONTEXTS

De volgende regels worden in acht genomen voor aggregates en hun bounded contexts:

- De aggregate zorgt voor consistentie binnen de bounded context. Dit betekent dat het verwerken van gerelateerde objecten binnen 1 transactie moet plaatsvinden
- De aggregate objecten hebben een identiteit die over alle bounded contexten geldt. Op deze manier zijn uitwisselingen mogelijk van gegevens tussen de bounded contexten. Overige entiteiten binnen een bounded context, hebben een lokale identiteit. Er bestaat geen garantie dat voor entiteiten met dezelfde naam en identiteit maar in verschillende bounded contexten, zij ook identiek zijn. Immers, bijvoorbeeld de entiteit “Person” kan in de bounded context van Employee totaal iets anders voorstellen als in de bounded context Sales
- De opbouw van de structuur van objecten uit een opslag, gebeurt altijd via de aggregate. De aggregate heeft kennis van de interne structuur in de bounded contex
- Een aggregate kan referenties hebben naar andere aggregates in andere bounded contexten
- Zodra er wijzigingen worden aangebracht via de aggregate, moeten deze onmiddelijk worden verwerkt in de opslag en moeten alle regels voor consistentie zijn nageleefd.

WAT ZIJN AGGREGATES EN VALUE OBJECTS?

Aggregates is dus een speciaal object in DDD. Conform de “ubiquitous language” is de aggregate dus een entiteit met speciale eigenschappen en gedrag. Andere objecten binnen de bounded context zijn dus of reguliere entieiten of value objects.
Value objects, weer een abstracte term die toelichting vereist. Het verschil tussen een entiteit en een value object is dat de laatste geen unieke identiteit heeft. Ook is een value object onwijzigbaar. Waarom zou je dergelijke objecten gebruiken en hoe pas je deze toe? Een voorbeeld wordt gebruikt om deze vragen te beantwoorden.
Neem de bounded context van OrderManagement (fictief voorbeeld). Binnen deze bounded context is de Order de aggregate en deze heeft een collectie van OrderLine objecten. Het OrderLine object heeft informatie nodig over het Product. Product wordt in dit voorbeeld beschreven door een naam en SKU code. Stel nu dat deze eigenschappen als eenvoudige eigenschappen in de OrderLine worden ondergebracht. Niets weerhoud een onwetende ontwikkelaar ervan de productnaam aan te passen van de Orderline. Echter, dit is niet toegestaan omdat men zich kan voorstellen dat de naam en SKU code bij elkaar horen.
Om dit te voorkomen wordt Product gebruikt als value object. Indien het bestaande Product op de OrderLine aangepast dient te worden, dan passen we niet de eigenschappen van Product aan, maar vervangen we de huidige instantie van Product door een nieuwe instantie van Product.

ARJEN KRAAK

Ik start vanuit de “waarom?” vraag om duurzame software-oplossingen te realiseren voor mijn klanten. Ik zoek hierbij de verbinding met de business en zorg ervoor dat wensen en eisen door business en IT worden begrepen en afgestemd.
Ik speel graag een actieve rol tijdens de definitie van een software vraagstuk. Ik krijg energie wanneer ik deze kan vertalen naar een ontwerp en realisatie van de oplossing met een team van ontwikkelaars.
Software architectuur en ontwikkeling zijn mijn passie. Ik deel mijn ervaring en kennis graag met collega’s binnen Bergler Software Solutions en daarbuiten. Ik sta mijn mannetje en werk graag samen met zowel betrokkenen in de business als met het ontwikkelteam.

Mark Monster

Wat doet een lead developer?

De afgelopen jaren heb ik binnen verschillende organisaties de rol van lead developer mogen bekleden. Soms werd de rol anders genoemd, zoals tech-lead of development coördinator. Maar wat houdt deze rol nu eigenlijk in? Hebben we het over een management rol of meer over zoeken we meer leiderschap? Iedere organisatie geeft een iets andere draai aan de rol lead developer. Hieronder een kleine uiteenzetting over zaken die komen kijken bij het uitvoeren van de rol van lead developer.

LEAD ALS IN LEIDEN

Als lead developer ben je een leider. Het is een rol met verantwoordelijkheid maar zonder mandaat. Het is jouw taak om de afgesproken kwaliteitsstandaarden van het team waar te maken. Dit kun je niet alleen, daar heb je het team - je interne stakeholders - voor nodig. In de basis ben je een rol model, dus geef je als lead developer het goede voorbeeld. Iedereen dient zich te houden aan de kwaliteitsstandaarden, daar is de de lead developer geen uitzondering in. Moedig je teamleden aan om kritisch te zijn, zowel naar oplossingen die je aandraagt als tijdens code reviews. Laat iedereen het gevoel krijgen dat je kritisch kunt kijken naar een oplossing en met opbouwende suggesties kunt komen zonder iemand af te kraken. Spreek elkaar er gerust op aan als één van je teamleden een ander persoonlijk aanspreekt. Het doel is niet om te bewijzen dat er verschil zit tussen de teamleden, maar om van elkaar te leren.

Een lead developer is over het algemeen communicatief vaardiger dan de gemiddelde developer. Dit is een kracht waar je als lead developer gebruik van kunt en zult maken. Hier hoort echter wel een verantwoordelijkheid bij. Zorg ervoor dat er ruimte is voor de andere teamleden om met initiatieven te komen en maai dus niet steeds het gras voor hun voeten weg. Sommigen teamleden hebben die behoefte helemaal niet en blijven liever wat meer op de achtergrond, dat is een prima keuze die niet hoeft te worden geforceerd. Deze teamleden hebben weer andere krachten die het tijdens hun werk tot uiting komen.

Veel organisaties zullen jou als lead developer verantwoordelijk zien voor de gekozen technische oplossingsrichting. Een ieder die zelf software heeft geschreven weet dat de oplossingsrichting nogal eens suboptimaal blijkt te zijn. Idealiter kom je hier achter op het moment dat herstel nog ‘goedkoop’ is. Soms is er echter meer kwaad geschied en hebben klanten er merkbaar last van gehad. Dat is het moment waarop je als lead developer op moet staan en verantwoordelijkheid neemt voor de situatie. Ook als heel duidelijk een probleem is ontstaan door toedoen van een teamlid, de lead developer neemt de verantwoordelijkheid. Uiteindelijk is dit onder jouw supervisie ontstaan. Denk als team samen na over de ontstane situatie en welke maatregelen jullie passend vinden. Zorg ervoor dat de stakeholders niet alleen op de hoogte worden gebracht van het ‘probleem’, maar ook van de maatregelen die jullie als team gaan nemen om dit in de toekomst te voorkomen.
Heel vaak gaat het wel goed en zijn er successen behaald met het team. Neem die ruimte ook om die successen als team te vieren. Als een teamlid zijn werk goed doet en misschien zelfs iets uitzonderlijks heeft gedaan, is dit zijn verdienste en is het belangrijk dat zijn formele manager of teamleider dit ook hoort.

TECHNISCHE KENNIS

Natuurlijk is het belangrijk dat de technische kennis een beetje in orde is, maar je hoeft zeker niet de beste techneut van het team te zijn. Belangrijk is een brede technische kennis, dit zorgt ervoor dat je in complexe situaties weet welke experts je bij elkaar moet brengen. Het is belangrijk dat je complexe zaken eenvoudig kunt uitleggen. Daarbij moet je bewust zijn op welk niveau je moet aansluiten in je team. Zo hebben developers vaak een andere uitleg nodig dan testers. Dat zit hem niet alleen in het verschil in kennis, maar ook in de verschillende kijk die mensen op een oplossing hebben. Zo willen developers vaak interne details weten, waarbij testers de interne werking meer als een gegeven beschouwen en geïnteresseerd zijn in de in- en output van de oplossing.

Gebruikmaken van hulpmiddelen is altijd toegestaan. Ondersteun een uitleg met plaatjes die je tekent op een whiteboard of flipover. Of schrijf een stuk pseudocode.

Toon je team alternatieve, betere manieren om tot een oplossing te komen. Help hen ook om hun skills te verbeteren. Geef daarom ruimte voor ideeën van je teamleden en vaar niet blind op je eigen ervaring en skills.

Helemaal hot is de term DevOps, waarbij development en operations bij elkaar komen. Ja, het is belangrijk om wat affiniteit te hebben met het daadwerkelijk in productie draaien van software. In mijn beleving hoef je niet de systeembeheerder te zijn die de DMZ opzet, mTLS configureert of de firewall ruling regelt. Het is echter wel belangrijk om te weten hoe de applicatie in productie draait. Draait de applicatie achter een loadbalancer? Dan moet je dit als lead developer wel weten. Iedere lead developer heeft dit vast wel eens meegemaakt: een inschatting over de infrastructuur waar de applicatie op zou draaien die niet blijkt te kloppen. Loop dan eens langs operations voor uitleg, dat zorgt meestal voor een soepeler productiegang.

Besef altijd dat, hoe goed je technische kennis ook is, er zijn specialisten die meer weten. Ga daarom altijd respectvol met elkaar om. Probeer als developer de securityspecialist niet te overtuigen dat de regels op jouw applicatie niet van toepassing zijn, omdat je mooie blauwe ogen hebt. Met goed onderbouwde argumenten lukt dit meestal wel.

Wat doet een lead developer?

Binnen grote organisaties is het aantal systemen waaruit het IT landschap bestaat niet op twee handen te tellen. Zeker in de wereld van micro services loopt het aantal services in de tientallen. Veelal zal je team te maken krijgen met integratie van eigen systemen, met systemen van andere teams of zelfs SaaS diensten. Het is als lead developer essentieel om dit landschap van systemen te overzien. Het gaat dan niet om de details maar om de hoofdlijnen. Het helpt het team als er eenvoudige architectuurplaten worden getekend van de eigen systemen en de integratiepunten. Deze architectuurplaten zijn nuttig tijdens refinements waarbij bijvoorbeeld impact op een specifiek deel van het systeem aangegeven kan worden.

COMMUNICATIE

Dat communicatie belangrijk is voor een lead developer spreekt waarschijnlijk voor zich. Naast zaken kunnen uitleggen moet je ook kunnen luisteren. Luisteren tijdens een reguliere standup. Misschien zit iemand wel vast en durft jouw teamlid niet om ondersteuning te vragen. Na de standup vragen of je mee mag kijken helpt vaak goed. Soms hoef je niet meer te zijn dan een Rubber Duck.

Een door de wol geverfde lead developer heeft van nature de neiging om een refinement te leiden. Een refinement heeft een structuur nodig waarin ieder teamlid zijn inbreng kan geven. Gezien de grote diversiteit aan persoonlijkheden heeft ieder teamlid zijn eigen motivatie nodig om inbreng te kunnen leveren. Als er een besluit genomen moet worden over een oplossingsrichting, dan wil je dat zo'n besluit wordt gedragen door het team. Ondanks dat niet iedereen het perse met de oplossing eens hoeft te zijn, dient er wel vertrouwen te zijn.

Naast je eigen team heb je ook te maken met stakeholders buiten je eigen team. Zoals de architect, die meestal op hoofdlijnen kaders stelt waarin verschillende scrum teams kunnen bewegen. In sommige organisaties zijn deze kaders heel erg strikt, al zijn er vaak situaties die om een bijzondere oplossing vragen. Zo'n bijzondere oplossing dien je als lead developer wel even te toetsen bij de architect. In het verleden heb ik vaak gezien dat lead developers moeite hebben met het verdedigen van hun voorgestelde oplossing. Hun oplossing is dan soms het niet meer delen van een oplossing in een vroeg stadium.

Dit heeft een heel naar effect: aanpassingen in een oplossing zijn vaak duurder als de code grotendeels al is geschreven. Met een andere mindset je oplossing presenteren aan de architecten zal zeker helpen. Probeer bijvoorbeeld de architecten bij je concept oplossing te betrekken. Vraag actief om feedback. De kans is natuurlijk groot dat er een kleine aanpassing gewenst is in de voorgestelde oplossing. Dat is helemaal niet erg, we hebben het hier niet over ego-zaken zoals gezichtsverlies, we hebben het over een betere oplossing.

Dat communicatie belangrijk is voor een lead developer spreekt waarschijnlijk voor zich. Naast zaken kunnen uitleggen moet je ook kunnen luisteren.

Verder hebben veel organisaties nog een overlegorgaan waarin de architecten en alle lead developers samen komen. Tijdens zo'n meeting worden meestal team overstijgende technische zaken besproken zoals architectuur, standaarden, maar ook wordt er informatie gedeeld over de worstelingen waar andere teams mee te maken hebben. Een mooi moment om je eigen ervaringen te delen en van elkaar te leren. Maar vergeet nooit een officieel overleg is niet noodzakelijk om met andere lead developers te praten. Als je ergens tegenaan loopt en binnen je eigen team kom je er niet direct uit, wandel dan eens langs een lead developer van een ander team. Soms levert dit direct een oplossing of een stap in de richting van de oplossing en in het slechtste geval een mooi sparringsmoment.

HET SCHAAP MET VIJF POTEN?

Leiderschap, communicatie en technische kennis, gebieden waarmee je als lead developer mee te maken krijgt. Dit lijkt misschien op de rol van een schaap met vijf poten. De rol van lead developer past dan ook niet bij iedereen. Bovenstaande uiteenzetting past bij hoe ik de rol van lead developer invul. Laat je hierdoor inspireren en geef de verschillende aspecten een eigen invulling.

Op zoek naar een lead developer voor je eigen team? Of wil je naar aanleiding van dit artikel van gedachten met mij willen wisselen? Neem dan contact met mij op.

MARK MONSTER

Mark Monster is freelance software architect en lead developer. Hij heeft meer dan 15 jaar ervaring als lead developer en architect en is gespecialiseerd in micro services en cloud omgevingen gebaseerd op .NET. Neem contact op met Mark via mark@monsterconsultancy.nl of via 0654106014.
Volg ook zijn blog op https://monsterconsultancy.nl.

"SOMS HOEF JE NIET MEER TE ZIJN DAN EEN RUBBER DUCK..."

Menno Jongerius

Valkuilen in async/await

Het is inmiddels alweer behoorlijk wat jaren geleden dat Microsoft async/await introduceerde in het .NET Framework. Het gebruik van async/await heeft het mogelijk gemaakt om met heel weinig code asynchroon te programmeren. Helaas betekent dit niet dat het eenvoudig is om asynchroon te programmeren, en ik heb mijzelf meermalen in de voet geschoten doordat ik te kort door te bocht wilde gaan. Bij deze wat valkuilen waar ik tegenaan ben gelopen, maar eerst even de twee belangrijkste redenen om asynchroon te programmeren.

WAAROM ASYNC/AWAIT EEN GOEDE ZAAK IS

Er zijn eigenlijk twee belangrijke redenen om async/await toe te passen. In webapplicaties zal de webserver (afhankelijk van de gekozen technologie) maar een beperkt aantal threads tegelijk in een workerprocess afhandelen. Dit betekent dat bij een groot aantal gelijktijdige verzoeken de webserver processen in de wacht gaat zetten. Door gebruik te maken van async/await, wordt de calling thread van het workerprocess vrijwel direct vrijgegeven, terwijl een background thread de bulk van het werk uitvoert. In client applicaties (Windows en apps) zijn asynchrone processen vooral van belang omdat de UI-thread wordt vrijgegeven en de applicatie dus goed blijft reageren op de gebruiker, terwijl processen op de achtergrond worden afgehandeld. Elke langlopende taak (database, I/O of zware berekening) leent zich om in een asynchrone taak af te handelen.

VALKUIL 1: DEADLOCKS

In het voorbeeld hieronder wordt het resultaat van een taak uitgevraagd. Stel dat dit een asynchrone methode is, die zelf ook gebruik maakt van async/await, dan creëer je een deadlock. Dat komt omdat het aanroepen van Result, de UI-thread in de wacht zet totdat het resultaat van de methode terugkomt. Als de methode zelf een await commando gebruikt, zal deze een thread opstarten en na afloop het werk willen teruggeven aan de UI-thread. Dat kan echter niet omdat deze UI-thread aan het wachten is op het resultaat van de functie die hij nu zelf zou moeten uitvoeren.

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    var processor = new AsyncProcessing();

    this.AsyncLabel.Content = processor.ExecuteAsync().Result;
}
```

FIGUUR 1

Ofschoon dit fenomeen vooral voor zal komen in een Windows applicatie of app, is het niet uit te sluiten dat deadlocks voorkomen in een webapplicatie wanneer je zelf de asynchrone taken beheert en niet gebruik maakt van de asynchrone controllers van ASP.NET.

OPLOSSING

Zorg dat de methode nadat deze gereed is, wordt opgevolgd door een nieuwe asynchrone taak die het resterende werk uitvoert. In de praktijk is het aan te raden om een class op te zetten die alle uitkomsten van een taak correct afhandelt, want er komt nogal wat bij kijken om alle foutafhandeling en cancellation scenario's in te bouwen.

Valkuilen in async/await

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    var processor = new AsyncProcessing();

    processor.ExecuteAsync().ContinueWith((task) =>
    {
        // do all stuff that is required here after the code continuous
        this.AsyncLabel.Content = task.Result;
    });
}
```

FIGUUR 2

VALKUIL 2: UI WERK OP EEN NIET-UI-THREAD

In het voorbeeld hierboven werd het resultaat van een taak gebruikt in op een label de content te veranderen. Dit gaat fout in Windowsapplicaties en apps wanneer een andere thread dan de UI-thread deze code uitvoert.

OPLOSSING

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    var processor = new AsyncProcessing();

    processor.ExecuteAsync().ContinueWith((task) =>
    {
        // do all stuff that is required here after the code continuous
        Dispatcher.Invoke(() =>
        {
            this.AsyncLabel.Content = task.Result;
        });
    });
}
```

FIGUUR 3

Stuur het werk dat naar de UI-thread moet via de Dispatcher naar de UI-thread. Deze is toegankelijk vanaf een window/usercontrol of via het Application object. In de praktijk schijf ik er altijd een proxy met interface voor, zodat ik de code ook kan unittesten.

VALKUIL 3: INCORRECTE/GEEN FOUTAFHANDELING

Binnen een async/await context worden fouten netjes doorgegeven aan de aanroepende thread. Wanneer je aan het einde van de callchain aankomt, is het heel gemakkelijk om de foutafhandeling te vergeten.

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    var processor = new AsyncProcessing();

    processor.ExecuteAsync();
}
```

FIGUUR 4

Bovenstaande code zal zonder problemen draaien, maar indien er in de ExecuteAsync een fout plaatsvindt zal deze niet zichtbaar worden in de aanroepende thread. Gebruik om dezelfde reden geen async void (tenzij je zeker weet wat je doet), maar retourneer altijd een taak zodat eventuele fouten afgehandeld kunnen worden. Wanneer je de code aanpast naar ExecuteAsync().Wait() zal er wel een fout optreden, echter dan krijg je weer deadlock problemen.

OPLOSSING

Schrijf, zoals in een eerder voorbeeld, een juiste continuatie taak waarin eventuele fouten worden afgehandeld.
Schrijf wat er moet gebeuren wanneer er een fout optreedt (bij voorkeur wat charmanter dan MessageBox.Show).

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    var processor = new AsyncProcessing();

    processor.ExecuteAsync().ContinueWith((task) =>
    {
        // schrijf deze melding netjes weg in de logging
        MessageBox.Show(task.Exception.Message);
    }, TaskContinuationOptions.OnlyOnFaulted);
}
```

FIGUUR 5

VALKUIL 4: AWAIT ALLEEN ALS HET NODIG IS

```
private async Task GetFileContents()
{
    await ReadFileAsync();
}

1 reference | 0 changes | 0 authors, 0 changes
private async Task<string> ReadFileAsync()
{
    using (var fs = new FileStream("File path", FileMode.Open))
    using (var sr = new StreamReader(fs))
    {
        return await sr.ReadToEndAsync();
    }
}
```

FIGUUR 6

In het bovenstaande voorbeeld wordt de ReadFileAsync methode aangeroepen. De aanroepende methode zal bij het await keyword een nieuwe thread opstarten om het asynchrone werk op te pakken. De ReadFileAsync methode zal nogmaals een thread opstarten bij het await keyword. Deze laatste is onnodig omdat de waarde van ReadToEndAsync in deze methode niet gebruikt wordt. In dit geval is het beter om geen async/await te gebruiken, maar rechtstreeks een Task te retourneren.

OPLOSSING

```
public async Task GetFileContents()
{
    var content = await ReadFileAsync();
}

1 reference | 0 changes | 0 authors, 0 changes
private Task<string> ReadFileAsync()
{
    return Task.Run(() =>
    {
        using (var fs = new FileStream("File path", FileMode.Open))
        using (var sr = new StreamReader(fs))
        {
            return sr.ReadToEnd();
        }
    });
}
```

FIGUUR 7

Geef een taak terug zodat er slechts een thread opgestart wordt bij de eerste await. Let wel op dat je in dit geval de boundaries van de code goed in acht neemt. De using statements moeten hier binnen de taak liggen en niet erbuiten anders zijn deze objecten al opgeruimd terwijl de taak nog draait.

VALKUIL 5: START NIET ONNODIG THREADS OP

```
public void SendRequest()
{
    urls.AsParallel().ForAll(async url =>
    {
        var client = new HttpClient();
        await client.PostAsync(url, new StringContent("Hello World"));
    });
}
```

FIGUUR 8

Los van de zaken die in bovenstaande code niet juist zijn, zoals het niet afhandelen van fouten en onhandig aanmaken van de HttpClient, wordt er voor elke parallelle actie nu een extra thread gemaakt voor het uitvoeren van de post. Dat betekent een thread vanuit de task parallel library en een vanuit async/await.

```
public void SendRequest()
{
    var tasks = urls.Select(url =>
    {
        var client = new HttpClient();
        return client.PostAsync(url, new StringContent("Hello World"));
    });
    Task.WaitAll(tasks.ToArray());
}
```

FIGUUR 9

OPLOSSING

Bij een groot aantal url's is de beste oplossing om wel de task parallel library te gebruiken, maar hierbinnen de post synchroon te houden.

Bij een klein aantal url's kun je beter een lijst van taken retourneren waar je de code op laat wachten.

MENNO JONGERIUS

Geweldige software maken, dat is waar het uiteindelijk om gaat. Dat je met je applicaties problemen van klanten oplost, maar er komt veel bij kijken voordat je als bedrijf zover bent. Kun je snel schakelen? Is je ontwikkelteam op elkaar ingespeeld? Is je service op orde? Hoe heb je je strategie uitgezet?
Dit zijn de vragen waar ik me graag met collega's op focus en de optimale antwoorden op wil vinden.

> **IN CLIENT APPLICATIES (WINDOWS EN APPS) ZIJN ASYNCHRONE PROCESSEN VOORAL VAN BELANG OMDAT DE UI-THREAD WORDT VRIJGEGEVEN EN DE APPLICATIE DUS GOED BLIJFT REAGEREN OP DE GEBRUIKER, TERWIJL PROCESSEN OP DE ACHTERGROND WORDEN AFGEHANDELD.**

Patrick Bes

Kan Cloud Native On-Premises zijn?

Nee, dit gaat niet over het On-Premises thuiswerken met behulp van Cloud Diensten in onheilspellende tijden van het Coronavirus. Dit gaat over vraag “kun je Cloud Native zijn met een On Premises serverpark?” Vreemde vraag leek mij. Volgens Wikipedia is Cloud Native een softwareontwikkeling aanpak die maximaal de mogelijkheden van Cloud Computing gebruikt. Dit strookt niet direct met een On Premises aanpak. Vanwaar dan toch deze vraag? Dit is naar aanleiding van het rapport van VMWare “The State of Kubernetes 2020. ”

LEAD ALS IN LEIDEN

Dit is het nieuwste onderzoeksrapport van VMWare onder een aantal grote organisaties naar de status van Kubernetes. Het rapport heeft wat verrassende uitkomsten zoals dat 64% van de ondervraagden Kubernetes On-Premises draaien, wat meer is dan vergelijkbare rapporten. Dit heeft waarschijnlijk te maken met het type organisaties dat VMWare in dit onderzoek heeft ondervraagd. Het zijn namelijk alleen organisatie met 1000 of meer medewerkers. Organisaties van deze omvang hebben wellicht zelf al veel infrastructuur staan, waardoor de switch naar de cloudproviders minder snel van belang is. Het ligt voor de hand dat in het kader van de bovenstaande uitslag de onderstaande quote in het rapport is opgenomen:

> *“Cloud native isn’t about where you operate, it’s how you operate.”*
> (Joe Beda, Principal Engineer, VMware and co-creator of Kubernetes)

Dit is een uitspraak waar ik even van op mijn hoofd moest krabben. Is het zo dat de manier waarop je werkt bepaalt of je cloud native bent en niet de plek? Om dit te kunnen beantwoorden moet ik eerst antwoord hebben op de vraag:

WAT IS EIGENLIJK CLOUD NATIVE?

Als we praten over cloud native dan is de Cloud Native Computing Foundation direct de organisatie waar je aan zult denken. Met de gang naar de cloud is het makkelijker geworden om open source projecten van verschillende bronnen te adopteren, waardoor je organisatie meer kan focussen op business logica en minder op het technische loodgieterswerk. De CNCF heeft als doel om deze open source projecten te beoordelen zodat jij als organisatie enige zekerheid hebt voor de continuïteit en dat je geen vendor lock-in hebt. Dit zijn projecten waarmee je comfortabel de cloud in kunt. Hiervoor heeft het CNCF een overzicht van al hun projecten op 1 plek: https://landscape.cncf.io/.
Het CNCF is de autoriteit op het gebied van cloud native en hebben ook de definitie voor cloud native opgesteld. Vrij vertaald is het ongeveer:

Cloud native technologieën geven organisaties de mogelijkheid om moderne schaalbare applicaties te ontwikkelen en te runnen, in dynamische omgevingen zoals de public, private en hybride clouds. Het gebruik van containers, service meshes, microservices, immutable infrastructuur en declaratieve API’s bevestigen deze aanpak.

Dan terug naar de vraag kan Cloud Native On-Premises zijn. Het antwoord zit hem in de Private Cloud definitie.

PRIVATE CLOUD (!)= ON-PREMISES

Private clouds kunnen er in vele verschijningsvormen zijn. Zo zijn er private hosted clouds waarbij niet gedeelde dedicated hardware door een cloud provider beschikbaar is gesteld. Of een internal hosted private cloud waarbij de omgeving op de interne infrastructuur van de organisatie geplaatst is. In dit laatste geval kan men spreken van een On-Premises omgeving.
Als men in een dergelijke omgeving gebruik maakt van cloud native projecten zoals door het CNCF worden ondersteund (bijv. Kubernetes) dan kan men spreken over Cloud Native op een On-Premises omgeving.

WENSELIJK?

De vraag blijft dan: is het wenselijk om een Cloud Native On-Premise oplossing te ontwikkelen? Dit kan in veel gevallen een prima oplossing zijn. Door geen gebruik te maken van de public cloud kan men b.v. lastige vraagstukken voor legal & compliance ontwijken. Het kan een prima startpunt zijn om met cloud native projecten kennis te maken. Uiteindelijk kan het ook als startpunt functioneren om een hybrid cloud scenario te ontwikkelen.
Het heeft ook een keerzijde, zo ontneem jezelf de mogelijkheid tot het gebruik van managed public cloud diensten. Denk bijvoorbeeld aan AKS, GKE of EKS (managed Kubernetes), Azure SQL of Cosmos DB. Dit zorgt voor meer onderhoud op dergelijke producten of ze kunnen überhaupt niet On Premises worden afgenomen. Dit zijn overwegingen die moeite waard zijn om te bekijken maar, uiteindelijk is het hebben van een cloud native strategie in een On Premise omgeving prima mogelijk.

Bronnen:

- https://k8s.vmware.com
- https://www.cncf.io
- https://www.capgemini.com
- https://github.com

PATRICK BES

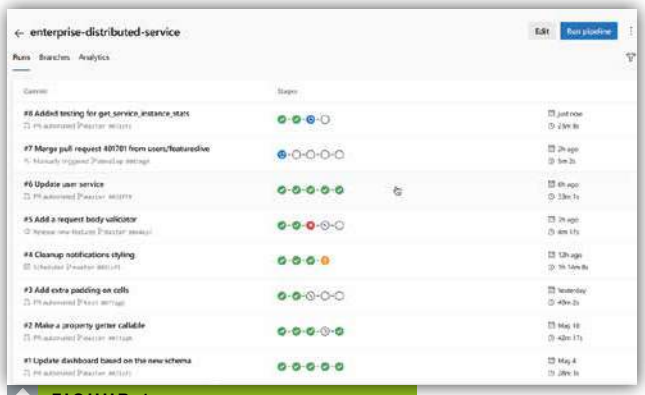
Software architect en lead application developer in diverse applicaties met focus op de Microsoft stack.

Eduard Keilholz

Multi-Stage Pipelines in Azure gebruiken

Al enige tijd, zijn de build en release opties in Azure DevOps van naam gewijzigd naar ‘pipelines’. Dus je hebt nu een build pipeline en een release pipeline. En ook al enige tijd, kun je die build pipeline in code configureren. Of nou ja, in code… YAML. Dit heeft voordelen, en een nadeel. Het nadeel spreekt voor zich, dat is YAML zelf. De notaties doen een beetje JSON-achtig aan, alleen missen de aanhalingstekens en de accolades. Daarnaast heeft YAML de nare eigenschap white-space gevoelig te zijn. Alles moet precies op de juiste manier uitgelijnd zijn en kloppen, anders werkt het niet. Dit levert veel frustraties op bij het schrijven en testen van die YAML-bestanden.

Het grote voordeel is, dat je deze bestanden naast je code in hetzelfde project kunt opnemen en je dus versiebeheer over deze bestanden krijgt. Ook wil het nog wel eens, dat een build- of releasedefinitie mee evolueert met de software. Tijdens het plannen van werkzaamheden kun je makkelijk een taak maken dat de YAML-bestanden bijgewerkt moeten worden, zonder dat een Operations, spookachtige wijzigingen moet gaan maken in Azure DevOps. Deze wijzigingen kunnen wellicht zorgen voor problemen in een build of release van andere opdrachten die simultaan ontwikkeld worden en ook naar de OTAP-straat moeten. Daarnaast is sinds enige tijd al de preview feature voor multi-stage pipelines in te schakelen (Klik in Azure DevOps rechts bovenin op gebruikers instellingen en zie daar de mogelijkheid om preview features in te schakelen).



FIGUUR 1

In de basis houdt het in dat je nu meerdere stages in een pipeline kunt bouwen. Een stage kun je zien als een stap in de het uitrollen van jouw systeem. Er moet dus iets gecompileerd worden, dat moet naar een test omgeving, daarna naar een acceptatie omgeving en uiteindelijk naar een productie omgeving. Vier stappen, vier stages.

BEGINNEN BIJ HET BEGIN

Laten we even beginnen bij het begin, zo’n YAML-pipeline. Als je er wel eens één gemaakt hebt voor bijvoorbeeld een ASP.NET Core Web API, zal hij er ongeveer zo uitzien:

```
trigger:
  - master

pool:
  vmImage: 'Ubuntu-16.04'


variables:
  name: 'value'
  other: 'value'

steps:
  - task: DotNetCoreCLI@2
    displayName: 'Restore Packages'

  - task:

  - task:
```

Een heel erg basis versie, doe een dotnet restore, compileer het systeem, start de unit tests, publiceer het systeem en publiceer deze bestanden vervolgens als artefact voor een release. Wanneer je nu een release pipeline start, zal deze het gepubliceerde artefact downloaden om er vervolgens iets mee te doen. Door verschillende stappen in je release definitie te maken, is het mogelijk om dit artefact op test, acceptatie vervolgens productie te installeren met tussenkomst van ‘Approvals’. Met andere woorden, een tester kan bijvoorbeeld voordat een versie op test komt, de release goedkeuren. Pas dan zal zijn test omgeving geüpdatet worden. Na het testen, kan de tester de versie ‘goedkeuren’ of ‘afkeuren’. Als de versie is afgekeurd, stopt de release straat daar. Wanneer de versie is goedgekeurd, zal deze een stap verder gaan, naar Acceptatie. Daar kan weer een verantwoordelijk persoon goed of afkeuren voordat het systeem weer een stap verder gaat. Deze stappen en de momenten van goed- en/of afkeuren zijn optioneel en allemaal naar eigen wens in te stellen. Op zich werkt dit systeem prima, alleen moet iemand rechten hebben op de ‘Ops’ omgeving, en daar alle stappen en acties definiëren, instellen en goed zetten en dat is een lastig klusje. Daarnaast kan iemand, zonder tussenkomst van een tweede persoon, de release definitie volledig aanpassen en dus ook stuk maken. Gelukkig kan dit ook anders.

Multi-Stage Pipelines in Azure gebruiken

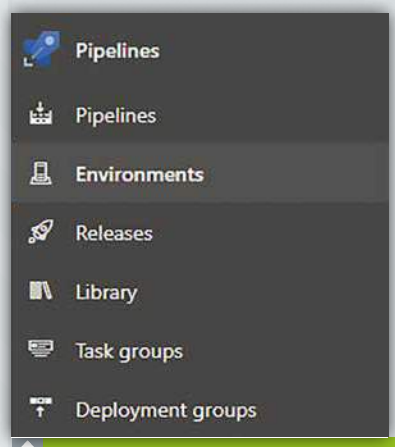
MEERDERE STAGES MAKEN

Waar een YAML-bestand voorheen enkel bestond uit stappen en taken, krijgen we er nu een derde laag bij, stages. De verschillende gebeurtenissen van het maken van een build en het uitrollen van deze build, kunnen zo onderverdeeld worden in verschillende stages.

```
trigger:
  - master

pool:
  vmImage: 'Ubuntu-16.04'

variables:
  buildConfiguration: 'Release'
  projects: '**/*.csproj'
  testProjects: '**/*Tests/*.csproj'

stages:
- stage: A
  jobs:
  - job: A1
  - job: A2

- stage: B
  jobs:
  - job: B1
  - job: B2
```

ENVIRONMENTS

Als je in Azure DevOps, binnen een project, het menu Pipelines openklapt, zie je een kopje 'Environments'. Hier kun je verschillende omgevingen maken, waar 'deployments' naartoe moeten. Dus bijvoorbeeld Test, Acceptatie en Productie. De 'approval' stappen voor het goed- of afkeuren van een bepaalde versie kan nu ingesteld worden op een environment. Zo kun je mooi omgevingen definiëren en instellen wie er verantwoordelijk zijn voor die omgeving.



FIGUUR 2

NIEUWE PIPELINES

Het vorige YAML-bestand, gaan we nu dus aanpassen zodat er ook releases bij zitten. Deze releases komen in een eigen stage, waarmee ze een individuele stap worden in het hele proces. Voor een release moet worden aangegeven naar welke omgeving uitgerold gaat worden. Azure DevOps draagt vervolgens zorg voor de benodigde randvoorwaarden zoals de verplichte goed- of afkeur momenten.

WAT IS ER NU ANDERS?

Het grootste verschil is dat de gehele pipeline, dus de builds en releases, allemaal in code geschreven zijn. Dit heeft tot gevolg, dat als er redelijke 'branch policies' ingesteld zijn, er reviews gedaan worden van eventuele wijzigingen in een build dan wel release definitie.

ARM TEMPLATES

Azure Resource Manager (ARM) Templates zijn ook voor multi-stage pipelines al makkelijk uit te rollen. Met multi-stage blijft deze functionaliteit natuurlijk behouden. Doordat ARM Templates gebruik maken van parameter bestanden kun je makkelijk per stage een afwijkende infrastructuur uitrollen. Zo hoef je test omgeving bijvoorbeeld niet zo zwaar ingericht te worden als je met een productie omgeving zou willen.

Er zijn drie strategieën voor het uitrollen van een ARM Template: Validatie; Incrementeel; en Compleet. Validatie doet niets aan de infrastructuur, maar valideert alleen het JSON-bestand. Incrementeel voegt alleen resources toe (of brengt wijzigingen aan). Met complete geef je aan dat jouw JSON-bestand compleet is. Als er een resource in Azure staat, die niet in het JSON-bestand staat, zal deze verwijderd worden.

Dit kun je makkelijk in zetten door je ARM Template eerst incrementeel uit te rollen. Nieuwe resources worden dan aangemaakt. Eventueel 'oude' of uit gefaseerde resources blijven nog event staan. Dan wordt het systeem uitgerold op de nieuwe omgeving. Daarna rol je nog een keer het ARM Template uit, maar dan de 'complete' stand. Te verwijderen resources worden zo pas verwijdert nadat de gehele omgeving om is naar een nieuwe versie.

CONCLUSIE

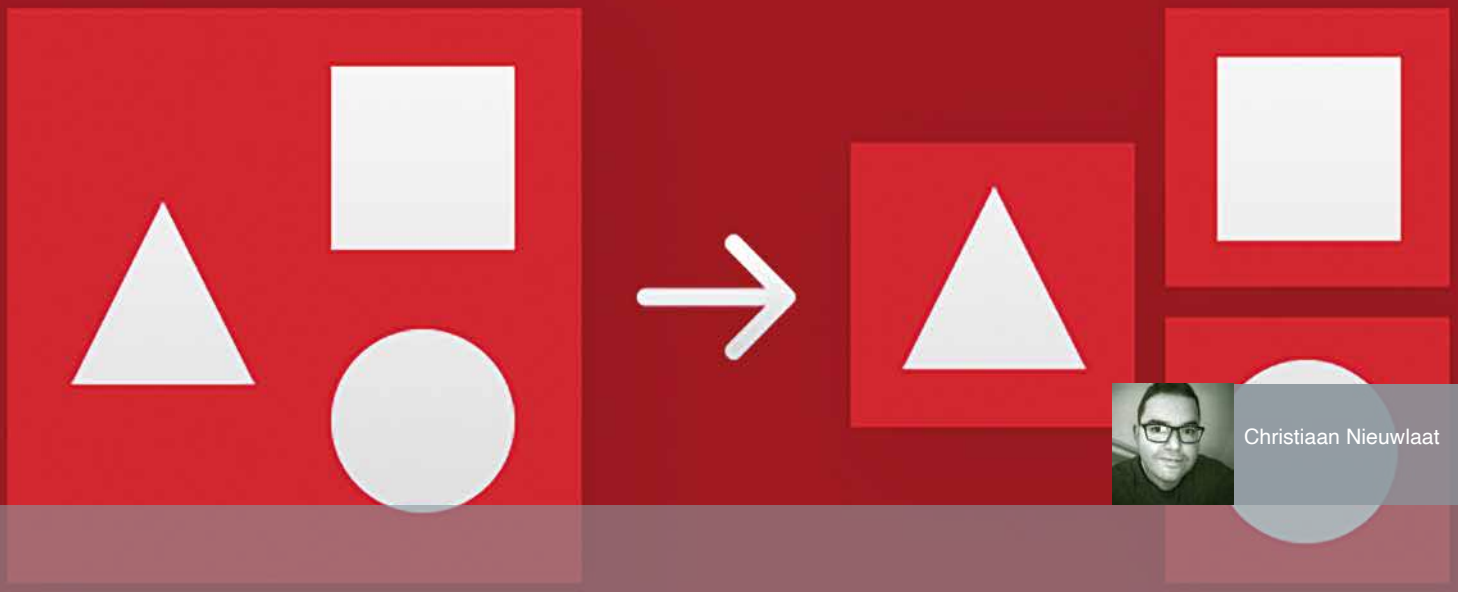
Azure DevOps heeft grote stappen gemaakt met het implementeren van multi-stage pipelines. De syntax (YAML) kan soms een beetje een uitdaging zijn maar dat weegt in mijn optiek niet op tegen de voordelen die je hebt. Mede in combinatie met het beheren van (cloud) resources in code (ARM Templates) ben je nu als ontwikkel team in staat alle benodigdheden voor een geoliede CI/CD-straat op die plek te beheren waar jij het liefst te vinden bent. Je ontwikkelomgeving.

Omdat zowel een ARM Template, als een multi-stage pipeline definitie in YAML een aantal pagina's in beslag zal nemen, heb ik een demo project in GitHub gemaakt. Een simpel Azure Functions project, met een volledige pipeline die een build maakt en deze uitrolt naar een Azure omgeving die ingericht wordt door een ARM-template.
https://github.com/nikneem/azure-functions-multistage-arm-deployment

EDUARD KEILHOLZ

Ik heb al meer dan twintig jaar ervaring met het ontwikkelen van software. Als Cloud Solution Architect bij 4DotNet mag ik bedrijven helpen kwaliteit stappen te maken in hun cloud oplossing. Ik vind het leuk om kennis over te dragen en om mensen te helpen. Naast mijn specialiteit, backend systemen ontwikkelen in de cloud, heb ik ook veel plezier in het maken van Angular front-end oplossingen.

4DotNet: https://4dotnet.nl
Blog: https://hexmaster.nl
Twitter: https://twitter.com/ed_dotnet
LinkedIn: https://www.linkedin.com/in/nikneem/



Christiaan Nieuwlaat

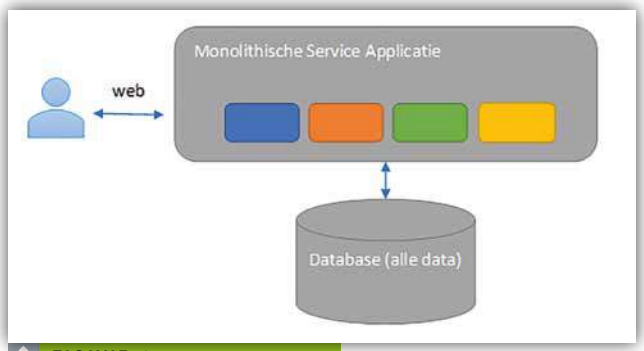
Van Monolithische Applicatie naar Microservices

Microservices, wie kent ze tegenwoordig niet? Relatief kleine applicaties die zeer domeingeoriënteerd een beperkt aantal functies kunnen uitvoeren. Waar we voorheen grote (monolithische) applicaties maakten die heel veel functionaliteit in zich hadden, word er heden ten dagen steeds meer gezocht naar een manier om onze software beter onderhoudbaar, stabieler en beter schaalbaar te maken.

Een van de architectuurvormen waar steeds vaker voor wordt gekozen is een zogenaamde microservices architectuur. Deze architectuur kenmerkt zich doordat een applicatie bestaat uit meerdere kleine applicaties die georkestreerd samenwerken om op die manier de gewenste functionaliteit te leveren.

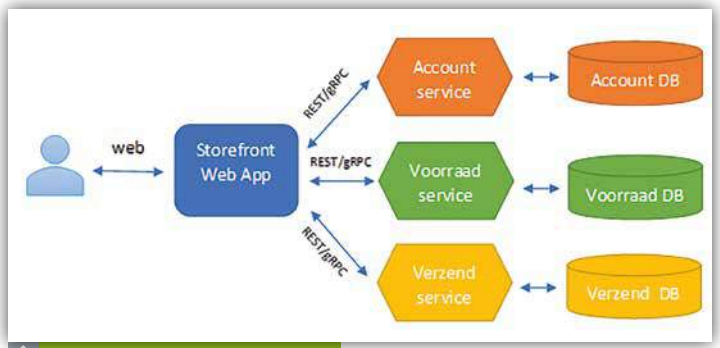
VOORBEELD VAN EEN TRANSFORMATIE NAAR MICROSERVICES

Stel we hebben een e-commerce webwinkel, gebouwd als enkele applicatie, deze applicatie is verantwoordelijk voor de storefront web-applicatie, maar ook voor het accountbeheer, voorraadbeheer en de verzending.



FIGUUR 1

Een dergelijk systeem er zou er als microservice platform als volgt uit kunnen zien.



FIGUUR 2

In dit voorbeeld is er gekozen om de onderdelen op basis van domeinen om te zetten naar losstaande microservices. Elke microservice heeft hierbij ook een eigen database en is ook als enige verantwoordelijk voor deze data.

DE VOORDELEN VAN EEN MICROSERVICE ARCHITECTUUR

Een van de meest voor de hand liggende voordelen van het opknippen is dat de losse onderdelen schaalbaar zijn, stel dat het ontzettend

druk is op de webwinkel, dan is het nu mogelijk om alleen dat deel van de applicatie op te schalen, zonder dat dan ook de andere services mee opgeschaald hoeven worden.

Naast dit voordeel zijn er nog een aantal te noemen, waaronder:

- *Eenvoudiger ontwikkelen / Duidelijkere codebase*
  Eenvoudiger betekent hier niet dat het simpeler is om een applicatie op basis van microservices te maken. Wat hiermee wel wordt bedoeld is dat je omdat je te maken hebt met kleine, maar wel écht losstaande programma's, je je minder zorgen hoeft te maken over onverwachte side-effects in andere delen van de "totale" applicatie. De andere delen leven namelijk als consumer en/of producer in het landschap, en kunnen alleen via een vooraf gespecificeerde API met je applicatie praten.

- *Best tool for the job*
  In een microservice landschap is het zeer goed mogelijk dat er gebruik gemaakt wordt van verschillende ontwikkeltalen om de services in te ontwikkelen. Naast het feit dat op deze manier de meest geschikte ontwikkeltaal voor een bepaalde oplossing gebruikt kan worden, zorgt dit er ook voor dat je als bedrijf een ruimere keuze aan ontwikkelaars zou kunnen aantrekken.

- *Onafhankelijk*
  De componenten kunnen onafhankelijk van elkaar worden bijgewerkt en in gebruik worden genomen.

- *Mogelijkheid voor Continuous Integration / Continouous Delivery*
  Doordat de componenten relatief klein zijn is het mogelijk om deze bij elke code-wijziging automatisch te laten bouwen, testen en bij goed bevonden eventueel packagen voor deployment. Dit is op een grote monolithische applicatie veel minder goed te doen.

- *Herbruikbaarheid*
  Als zowel Applicatie X als Applicatie Y een specifiek stuk functionaliteit gebruiken wat een bepaalde service biedt, dan kan deze service voor beide applicaties hergebruikt worden.

- *Betere deployment*
  Je hoeft alleen de services waar wijzigingen zijn geweest te vernieuwen, dit kan beduidend impact hebben op de mogelijke downtime bij een deployment

DE NADELEN VAN EEN MICROSERVICE ARCHITECTUUR

Zijn er dan alleen maar voordelen bij een microservice architectuur? Nee, zeker niet. Er zijn zeker een aantal punten waar heel goed op gelet moet worden bij het omzetten van een monolithische applicatie naar een microservice gebaseerde applicatie.
Een aantal van deze mogelijke valkuilen zijn:

- *Te veel verschillende programmeertalen*
  Waar het heterogene karakter van het microservice landschap juist heel positief is, werkt een overdaad aan verschillende talen juist tegen. In het kader van onderhoudbaarheid is het beter om een vooraf geselecteerd aantal talen te definiëren waar het platform mee kan worden gebouwd.

- *Goed overdachte opsplitsing*
  Een microservice moet zodanig gemaakt zijn dat deze zo "loosely coupled" mogelijk is, maar niet zo extreem dat een wijziging in een service een domino effect heeft op vele andere services.

- I*n memory / in-process communicatie is sneller dan inter-process communicatie*
  Dit is een valkuil voor vele ontwikkelaars die net aan de eerste microservice architectuur beginnen. De communicatie van programmacode in een en dezelfde applicatie kost nano of microseconden, waar eenzelfde communicatie over tussen losstaande processen al snel milliseconden tot seconden kan duren.

En dan is er nog een punt waar ik wat extra aandacht aan wil besteden. En dat is beschikbaarheid. Alwaar binnen een monolithische applicatie bijna altijd alle componenten standaard beschikbaar (en dus te gebruiken zijn), is dit bij gedistribueerde systemen (zoals een microservice platform) zeer zeker niet het geval. Het kan zo maar voorkomen dat een service niet beschikbaar is, deze service (of een van de services waar deze van afhankelijk is) kan down zijn gegaan, of er is een netwerkprobleem waardoor deze niet bereikbaar is. Daarom is het mijns inziens bij het ombouwen van monolithische applicatie naar een microservice platform erg belangrijk om bij het ontwerp en de bouw van het platform altijd rekening te houden met niet beschikbare services.

HULPMECHANISMEN BIJ HET OMGAAN NAAR EEN MICROSERVICE ARCHITECTUUR

Er zijn een aantal mechanismen die je kunnen helpen om het microservice platform zogenaamd robuuster en weerbaarder te maken in het geval er services niet beschikbaar zijn. Ik heb er een paar uitgekozen welke mijns inziens goed zouden kunnen helpen door deze vanaf de start van het ontwerpen al mee te nemen. Je wilt niet hebben dat je applicatie lelijk onderuit gaat omdat er een onderdeel niet beschikbaar is.

BUFFERING EN EVENTUEEL FALLBACK VIA QUEUES

In een microservices gebaseerde applicatie is er bijna geen enkele actie die niet over meerdere services heen wordt uitgevoerd. Sommige acties vereisen direct response, andere kunnen eventueel asynchroon worden afgehandeld. Voor deze laatste categorie zou het kunnen helpen om de intercommunicatie tussen de service processen door middel van een queue te laten doen. Mocht een verwerkend proces tijdelijk niet beschikbaar zijn of down gegaan zijn, dan kan deze alsnog het verzoek tot verwerking oppakken uit de queue en alsnog de gewenste actie uitvoeren. Dit kan zeker in systemen die zogenaamd eventually consistent moeten zijn een goede oplossing zijn.

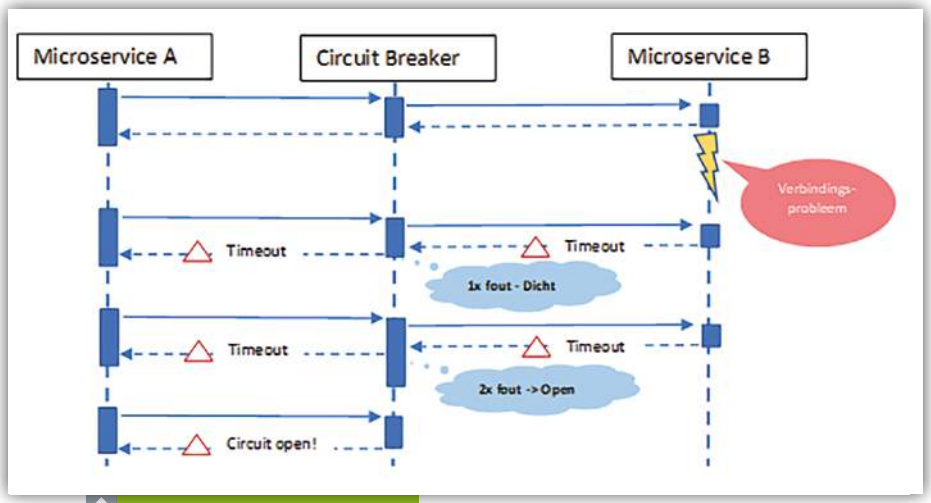
Een andere keuze kan zijn dat bepaalde services als "hydra" worden uitgevoerd. Hiermee bedoel ik dat een service zowel uit een queue als vanuit een directe call verzoeken kan afhandelen. Stel dat Service A een request aan Service B wilt doen om een berekening te maken, alleen Service B is op dat moment te druk bezet, dan kan Service A zijn verzoek in de queue voor Service B zetten, welke in de tussentijd weer beschikbaar is geworden. Deze leest het verzoek uit de queue en stuurt het response naar Service A via een Queue. Als Service A nog aan het wachten is op response, dan kan deze het response uit de Queue lezen en doorgaan met zijn routine.

Bijkomend voordeel van het werken met buffer queues is dat meerdere instanties van de verwerkende service parallel een werkvoorraad kunnen verdelen.

CIRCUIT BREAKER

Voor de services die eigenlijk direct antwoord moeten geven is er ook een mechanisme om te helpen het systeem robuuster te maken. Als microservice A, microservice B synchroon aanroept kan het zijn dat deze laatste niet beschikbaar is, of zodanig hoog belast dat deze niet tijdig kan reageren. Mogelijk zijn er in microservice A resources gereserveerd gedurende de tijd dat deze bezig is met wachten op response van microservice B. Als er maar genoeg calls naar microservice A gedaan worden, is er kan op zogenaamd resource starvation, waardoor microservice A geen enkel verzoek meer kan aannemen. Dit kan resulteren in een domino effect waardoor je applicatie onderuit zou kunnen gaan.

Om dit te voorkomen is het zogenaamde “Circuit Breaker” pattern bedacht. Dit is een intelligente proxy die tussen microservice A en microservice B wordt gebouwd en die in de gaten houd of microservice B nog wel goed werkt.



FIGUUR 3

Als microservice B een aantal keren (in dit plaatje 2 keer) niet goed of niet tijdig heeft geantwoord, zal de proxy voor een X tijd alle inkomende verzoeken direct laten falen. Na de wachtperiode zal de proxy weer een paar verzoeken doorsturen, en als microservice B weer goed werkt gaat de schakelaar weer terug om en begint het monitoren opnieuw.

Als afsluiter wil ik nog een tweetal mechanismen meegeven welke ik in de praktijk heb toegepast en ik zelf nuttig vind. Deze zijn niet specifiek ter verbetering van de robuustheid van je microservices platform, maar zijn mijns inziens zeker waard om mee te nemen in het onwerp.

EXPONENTIAL BACKOFF

Een service kan zichzelf monitoren om te zien of iemand / een process excessief gebruik maakt van de service. Mocht de service merken dat deze te vaak of te snel achtereen door dezelfde bevrager wordt aangeroepen, dan zal deze een response sturen met het verzoek om x tijd te wachten. Mocht dit verzoek niet gehonoreerd worden dan wordt er opnieuw een response gestuurd met een langere wachtijd. Steeds als er niet voldaan wordt aan deze wachttijd wordt deze exponentieel groter. Pas als de wachttijd verstreken is zal de service weer beschikbaar zijn voor verzoeken van deze bevrager.

LOG CORRELATION

Elke applicatie is leuk zolang dat ie werkt, dat is ook zo bij een microservices gebaseerde oplossing. Het wordt alleen lastig als het niet goed gaat. Het debuggen van een microservices platform is een hele andere tak van sport dan het debuggen van een monolithische applicatie. Waar het mijns inziens al compleet anders gaat, is bij het verkrijgen van de juiste diagnostische informatie door logbestanden. Waar een monolithische applicatie over het algemeen redelijk sequentieel logt wat ie doet en waar wat misgaat, is dat bij een microservices applicatie helemaal niet zo duidelijk. De microservices draaien namelijk in basis losstaand en worden mogelijk door verschillende andere services aangeroepen. Het kan hierdoor ontzettend lastig worden om over alle services heen een bepaalde actie in de log bij elkaar te vinden. Gelukkig zijn er bibliotheken die het relatief eenvoudig maken om zogenaamde log-correlation aan je logs toe te voegen. De verzoeker geeft bij het verzoek een unieke token mee, welke door alle services in de keten overgenomen en overgedragen worden, zodat al de services deze token kunnen toevoegen in de logs. Hierdoor krijgt elke actie van begin tot eind een kenmerk, en kan dat kenmerk gebruikt worden om de logs van alle services aan elkaar te kunnen relateren.

CHRISTIAAN NIEUWLAAT

Al vanaf zijn 4e levensjaar is Christiaan Nieuwlaat bezig met programmeren, begonnen met MSX basic listings overtypen vanuit MSX Tips-en-trucs boekjes. Daarna via Assembler, Pascal, Delphi, C en C++, nu, 35 jaar ervaring verder, zowel bedreven op het gebied van Java als .NET. Op dit moment houdt hij zich voornamelijk bezig met AWS, Cloud, Software ontwikkeling, Architectuur en Security. Naast gepassioneerd ontwikkelaar / architect is Christiaan sinds 7 juni 2019 een Certified Ethical Hacker, wat zijn passie voor informatiebeveiliging en softwarebeveiliging alleen maar meer heeft aangewakkerd.
Christiaan werkt op dit moment bij Team Rockstars IT als software engineer / solution architect, en helpt klanten hun complexe IT vraagstukken op te lossen, dit in diverse rollen, variërend van technisch adviseur tot software architect.
Naast zijn werkzaamheden is Christiaan een liefhebbende vader, deejay, danser en houdt hij ervan om te “tinkeren” met micro-elektronica / IoT devices. Hij is tevens bestuurslid van de lokale EHBO vereniging en in opleiding tot instructeur EHBO.

Jan de Vries

GEBRUIK

Managed Identities

VOOR INTER-SERVICE COMMUNICATIE

Het is inmiddels al een paar jaar mogelijk om, in Azure, Managed Identities te gebruiken om jouw service te authentiseren bij een Azure service, zoals een Azure Service Bus of Azure Key Vault.
Een Managed Identity is een automatisch gegenereerde identiteit voor jouw service welke door Azure wordt beheerd in de Azure Active Directory. Door gebruik te maken van deze identiteit & de bijbehorende functionaliteit hoef jij je, als ontwikkelaar of beheerder, niet meer druk te maken over het beheren van Client Secrets.

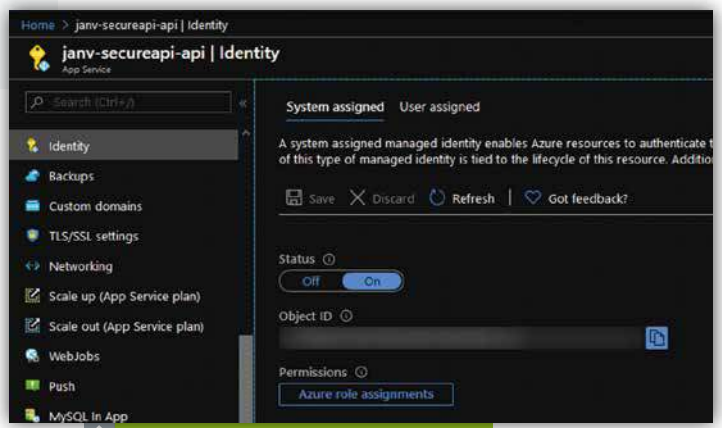
Maar wist je dat je Managed Identities ook kunt gebruiken voor het authentiseren (& autoriseren) in je eigen applicatielandschap? Hoe je dit doet en wat je hiervoor moet instellen ga ik kort in dit artikel beschrijven.

EEN MANAGED IDENTITY MAKEN

Een eenvoudige, maar zeer belangrijke, stap is het aanmaken van de Managed Identity. Het beste is om dit via je ARM-template in te stellen door het volgende stuk JSON toe te voegen.

```
"identity": {
  "type": "SystemAssigned"
}
```

Maar het kan natuurlijk ook via de Azure Portal door in de Identity blade de `Status` op `On` te zetten.



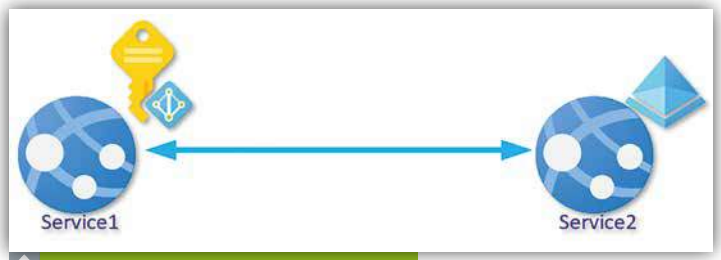
FIGUUR 1

Na het activeren van deze feature zul je zien dat er een nieuwe `Enterprise Application` is aangemaakt binnen de Azure Active Directory met dezelfde naam als jouw service. Dit nieuw aangemaakte object is te vergelijken met een Service Principal.

Nu er een identiteit voor onze service is aangemaakt is het tijd om de andere service(s) te configureren en rollen uit te delen.

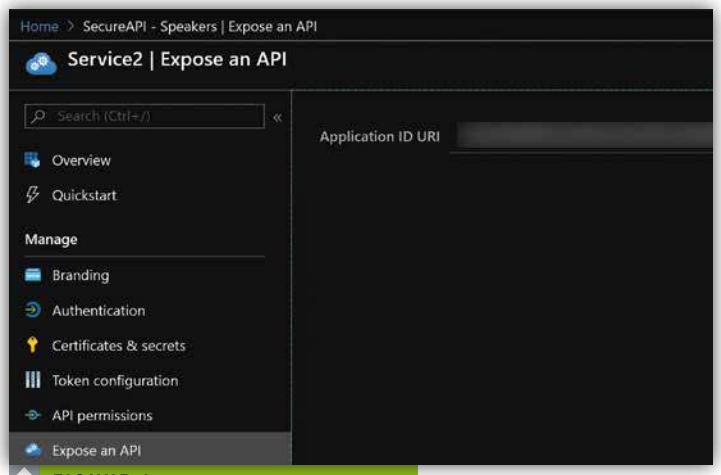
SERVICE CONFIGURATIE

Laten we stellen dat de Managed Identity is ingesteld op `Service1`. Er wordt vanaf deze service een HTTP request gemaakt richting `Service2`.
We willen er nu voor zorgen dat er in `Service2` alleen maar verzoeken binnen komen van gebruikers & applicaties welke zijn geauthentiseerd voor gebruik van deze service. Schematisch ziet dit er dan als volgt uit.



FIGUUR 2

Om dit te bewerkstelligen moet er in de Azure Active Directory een `App Registration` worden aangemaakt voor `Service2`. In de `Expose an API` blade moet een `Application ID URI` worden gemaakt welke later nodig is voor het configureren van `Service2` en het opvragen van een access token in `Service1`.



FIGUUR 3

Via de blade `Manifest` moeten nieuwe `appRoles` worden toegevoegd. Het is hierbij belangrijk dat de `id` en de `value` uniek zijn. Deze twee waarden worden later weer gebruikt.

```
"appRoles": [
{
    "allowedMemberTypes": [
        "Application",
        "User"
    ],
    "description": "Reader Role",
    "displayName": "Speaker service reader",
    "id": "42ee5891-7e50-4db9-a6d9-75ffc8cc1e9b",
    "isEnabled": true,
    "lang": null,
    "origin": "Application",
    "value": "SecureApi.Speaker.Reader"
},
```

Na de configuratie van de App Registration kan `Service2` hieraan worden 'gekoppeld'.
Dit doe je door in de `Startup.cs` de methoden `UseAuthentication()` en `UseAuthorization()` aan te roepen. In de `ConfigureServices` moet worden aangegeven dat je een `JwtBearerDefaults.AuthenticationScheme` wilt gebruiken en wat de valide 'audiences' zijn.

```
services
    .AddAuthentication(o =>
    {
        o.DefaultScheme = JwtBearerDefaults.
AuthenticationScheme;
    })
    .AddJwtBearer(o =>
    {
        o.Authority = Configuration["Authentication:Author
ity"];
        o.TokenValidationParameters = new Microsoft.
IdentityModel.Tokens.TokenValidationParameters
        {
            ValidAudiences = new List<string>
            {
                Configuration["Authentication:AppIdUri"],
                Configuration["Authentication:ClientId"]
            }
        };
    });
```

ROLLEN TOEKENNEN

Voor het toekennen van rollen is op het moment van schrijven nog geen UI beschikbaar, dus moet dit via PowerShell of de Azure CLI worden gedaan. Het commando via de Azure CLI ziet er als volgt uit.

```
az rest `
--method post `
--uri https://graph.microsoft.com/beta/servicePrincipals/
[enterpriseApplicationObjectId]/appRoleAssignments `
--headers "{'content-type': 'application/json'}" `
--body "{
'appRoleId': '[RoleId]', # identifier of your app role
'principalId': '[ManagedIdentityObjectId]', # object id of
the Managed Identity
'principalType': 'ServicePrincipal',
'resourceId': '[enterpriseApplicationObjectId]' #
the identifier Enterprise Application
}"
```

Hier gebruik je dus de object identifier van de Enterprise Application welke dezelfde naam heeft als je App Registration, de object identifier van je Managed Identity en de identifier van de rol die je wilt toekennen. Het maken van een valide access token ziet er nu als volgt uit.

```
var tenantId = this.configuration["ActiveDirectory:Tenan
tId"];
var applicationIdUri = this.configuration["ApplicationId
Uri"];

var azureServiceTokenProvider = new
AzureServiceTokenProvider();
var accessToken = await azureServiceTokenProvider.
GetAccessTokenAsync(
                        applicationIdUri,
                        tenantId: tenantId);
```

Let op dat hier ook weer de Application ID URI van `Service2` wordt gebruikt, omdat je voor die service een token wilt maken.
Dit access token kun je gebruiken in de Authorization header van een HTTP request richting `Service2`. De authenticatie & autorisatie zal plaatsvinden en wanneer die slaagt zal er een valide respons terug komen.

TOT SLOT

Mocht je in een architectuur terecht komen waar er meerdere services aan elkaar zijn geknoopt, zorg dan voor een goede authenticatie & autorisatie tussen deze services. De hierboven beschreven functionaliteit is voor iedereen gratis te gebruiken binnen Azure en is een prima eerste stap om de services voor de buitenwereld te beschermen.
Alle code kan worden teruggevonden op GitHub (https://github.com/Jandev/secure-apis), zodat je daar ook het gehele plaatje kunt zien.

JAN DE VRIES

Jan de Vries heeft in zijn loopbaan verschillende rollen gehad. Tegenwoordig werkt hij bij 4DotNet als een Cloud Solution Architect/Developer en heeft de Microsoft MVP award op het gebied Microsoft Azure gekregen. Zijn focus is daarbij het opzetten van schaalbare en performante software oplossingen, waar Microsoft Azure een primaire rol in speelt. Jan blogt met enige regelmaat op https://jan-v.nl en kunt hem volgen op @Jan_de_V

BERGLER ICT

Zeg wat je doet. Doe wat je zegt.

No-nonsense full-service ICT-dienstverlening. Dat is waar Bergler voor staat.
In de volle breedte van het vakgebied en zonder compromis. Direct inzetbaar om
de continuïteit van de bedrijfsvoering bij de opdrachtgever zeker te stellen.

Bergler Nederland B.V. Minervum 7210 4817 ZJ BREDA
info@bergler.nl T: 076 – 572 02 00

.NET Trainingen voor starters en specialisten
C# - XAML - ASP.NET Core - Visual Studio - SQL Server

Ons eigen lesmateriaal is altijd up-to-date en zeer praktijk gericht. Wij leveren maatwerk, ook in het Engels.

In-Company bij uw bedrijf.
Small-Group, 1-4 personen.
Online via Microsoft Teams.

Fons Sonnemans

MVP Microsoft® Most Valuable Professional

Reflectionit.nl

Fons Sonnemans

Avoiding defensive copies of Structs in C#

C# is a beautiful language which can be used to write high performance code. In general, using a struct value is faster than an object (class). Although you can easily make mistakes which will lead to defensive copies of structs. Sometimes those copies are called hidden or implicit copies because you don't see them easily. These copies will slow down the performance, especially if the struct is larger than 16 bytes. In this article I will discuss how to use the latest C# 7 and C# 8 features to avoid these defensive copies.

WHERE DO DEFENSIVE COPIES OCCUR?

Let's start with our own Point struct. It contains X and Y fields of the type double. It has a Swap() method and a Distance property. The ToString() method is overridden and uses string interpolation to display the (X,Y) value.

```
struct Point {

    public double X, Y;

    public Point(double x, double y) { X = x;  Y = y; }

    public void Swap() => this = new Point(Y, X);

    public double Distance => Math.Sqrt((X * X) + (Y * Y));

    public override string ToString() => $"({X.
ToString()},{Y.ToString()})";
}
```

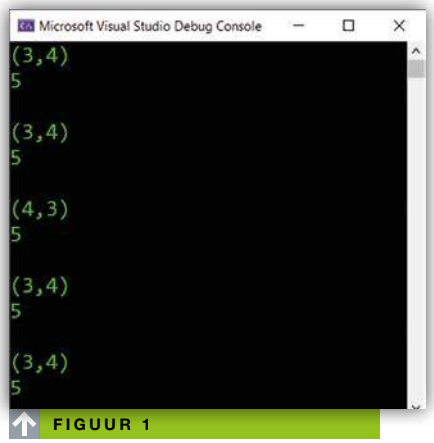
There are 3 places where defensive copies of struct values occur.

1. When you call members off a **readonly field**
2. When you call members of a **readonly local**, new in C# 7.2
3. When you call members of an **in parameter**, new in C# 7.2

The next code sample has all of these issues.

```
class Program {

    private static Point _p1 = new Point(3, 4);
    private static readonly Point _p2 = new Point(3, 4);

    static void Main(string[] args) {
        Console.WriteLine(_p1.ToString()); // (3,4)
        Console.WriteLine(_p1.Distance); // 5
        Console.WriteLine();

        _p2.Swap(); // Defensive Copy, the copy is swapped
        Console.WriteLine(_p2.ToString()); // Defensive Copy:
(3,4)
       Console.WriteLine(_p2.Distance);   // Defensive Copy: 5
        Console.WriteLine();

        Point p3 = _p1; // Value copy
        p3.Swap();
        Console.WriteLine(p3.ToString());  // (4,3)
        Console.WriteLine(p3.Distance);    // 5
        Console.WriteLine();

        ref readonly Point p4 = ref _p1;
        p4.Swap(); // Defensive Copy, the copy is swapped
        Console.WriteLine(p4.ToString()); // Defensive Copy:
(3,4)
        Console.WriteLine(p4.Distance);   // Defensive Copy:
5
        Console.WriteLine();

        Foo(_p1); // (3,4) & 5
    }

    static void Foo(in Point p) {
        p.Swap(); // Defensive Copy, the copy is swapped
        Console.WriteLine(p.ToString()); // Defensive Copy
        Console.WriteLine(p.Distance);   // Defensive Copy
    }
}
```

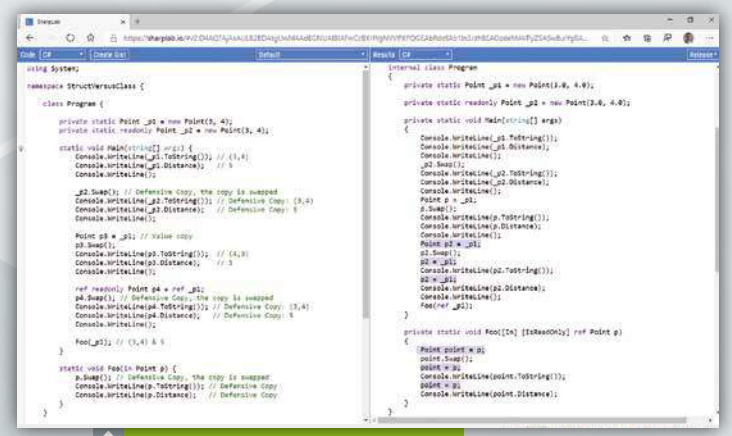
If you run this app you get the following output. The second point (_p2) is not swapped because it is a readonly field. The fourth point is not swapped because it is a readonly local. The fifth point is not swapped because it is an in parameter. Not only the Swap() calls but also the Distance and ToString() calls cause defensive copies.



FIGUUR 1

Avoiding defensive copies of Structs in C#

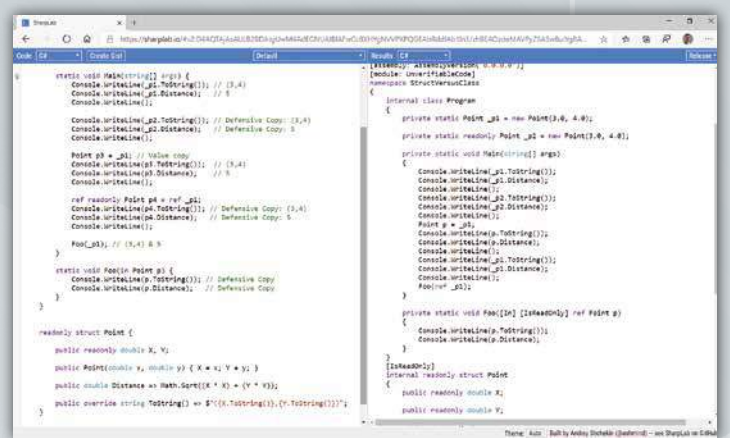
To make these copies visible you can use a decompiler, for example ILSpy, Reflector or DotPeek. I often use the website SharpLab.io. You can write code in it and see the Intermediate Language (IL) which is generated. It can also decompile the IL to C#, for this it uses the ILSpy engine. It is like looking under the hood at what is going on.
The purple marked lines in the right panel of the following screenshot show you all the defensive copies. There are 3 more for the readonly field (_p2) calls. Unfortunately, ILSpy doesn't decompile them correctly, they are really hidden.



FIGUUR 2

SOLUTION 1: MAKE THE STRUCT READONLY
In C# 7.2 there is a feature called 'Readonly structs'. You add the readonly keyword in front of the struct keyword. You also have to add the readonly keyword in front of all fields. You cannot assign to 'this' any more so I removed the Swap() method for now.

When you examine the result of this in SharpLab.io you will see that the defensive copies are gone. This will improve the performance of your code. You could (should) create benchmark for this to validate it. I did this and it is really faster. How much depends on many factors like the size of the struct and what else you have in your code.

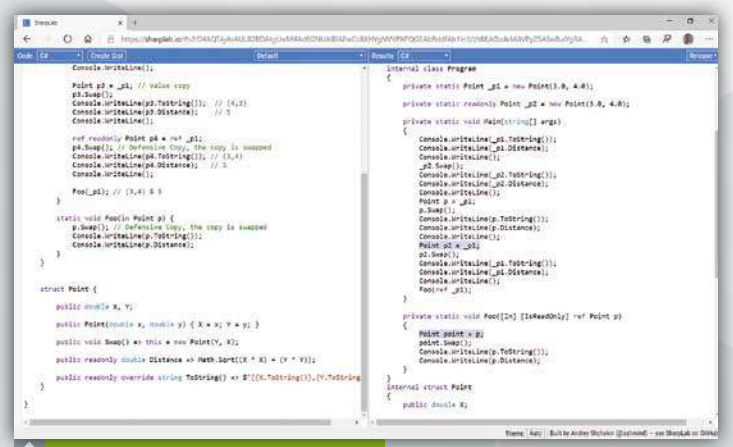


FIGUUR 3

SOLUTION 2: USE READONLY MEMBERS
Solution 1 is not always possible because the struct becomes immutable. This is not always something you want. In C# 8.0 there is a new feature called 'Readonly members'.

With this feature you can keep the struct mutable and only make some of your members readonly. Readonly members are not allowed to change the value of a field. If you call those members, defensive copies are not created any more.

When you examine the result of this in SharpLab.io you will see that Swap() still uses a defensive copy. They are gone for the Distance and ToString() which will improve the performance.



FIGUUR 4

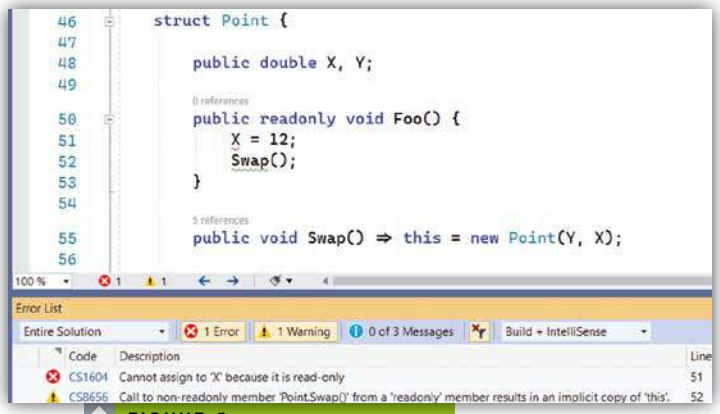
To improve my Point struct I have rewritten the Swap() method. It now returns a new Point instead of assigning it to 'this'. This allows me to make the method readonly which will improve the performance again.

```
struct Point {

    public double X, Y;

    public Point(double x, double y) { X = x; Y = y; }

    public readonly Point Swap() => new Point(Y, X);

    public readonly double Distance => Math.Sqrt((X * X) +
(Y * Y));

    public readonly override string ToString() => $"({X.
ToString()},{Y.ToString()})";
}
```

All callers must be aware of this change and use (store) the return value. It is a breaking change so be careful with this.

```
//p3.Swap();
p3 = p3.Swap();
```

In the next screenshot the Foo() method is readonly, it has an 'error' and a 'warning' because it changes the value of X and calls the Swap() method. In the warning the term 'implicit copy' is used instead of 'defensive copy'.



FIGUUR 5

I hope (expect) that next versions of Visual Studio will have more warnings of implicit copies.

BENCHMARK

In .NET Core 3.0 the DateTime struct is readonly, in .NET Framework it is not. The next micro benchmark will compare the call of the Year property on DateTime fields. One field is readonly the other one not.

```
class Program {

    // NuGet: Install-Package BenchmarkDotNet
    static void Main(string[] args) => BenchmarkRunner.
Run<DateTimeBM>();
}

[SimpleJob(RuntimeMoniker.Net48)]
[SimpleJob(RuntimeMoniker.NetCoreApp31, baseline: true)]
public class DateTimeBM {

    private DateTime _date1;
    private readonly DateTime _date2;

    [Benchmark] public int NotReadonly() => _date1.Year;

    [Benchmark] public int Readonly() => _date2.Year;
}
```

If you look at the results for the 'NotReadonly' method you see that the performance difference between .NET 4.8 and .NET Core 3.1 is about 10% (ratio), Core is just faster. If you look at the 'Readonly' method you get a 21% difference. This is 11% more which is caused by the defensive copy. You may think this is not a large difference. DateTime is only a small struct (8 bytes). If the struct is larger, for example 24 bytes, you get a larger difference. I had 35% for a struct of 24 bytes.

Method	Job	Runtime	Mean	Error	StdDev	Ratio
NotReadonly	.NET 4.8	.NET 4.8	5.914 ns	0.0226 ns	0.0212 ns	1.10
NotReadonly	.NET Core 3.1	.NET Core 3.1	5.372 ns	0.0152 ns	0.0127 ns	1.00
Readonly	.NET 4.8	.NET 4.8	6.480 ns	0.0155 ns	0.0130 ns	1.21
Readonly	.NET Core 3.1	.NET Core 3.1	5.365 ns	0.0211 ns	0.0187 ns	1.00

FIGUUR 6

CLOSURE

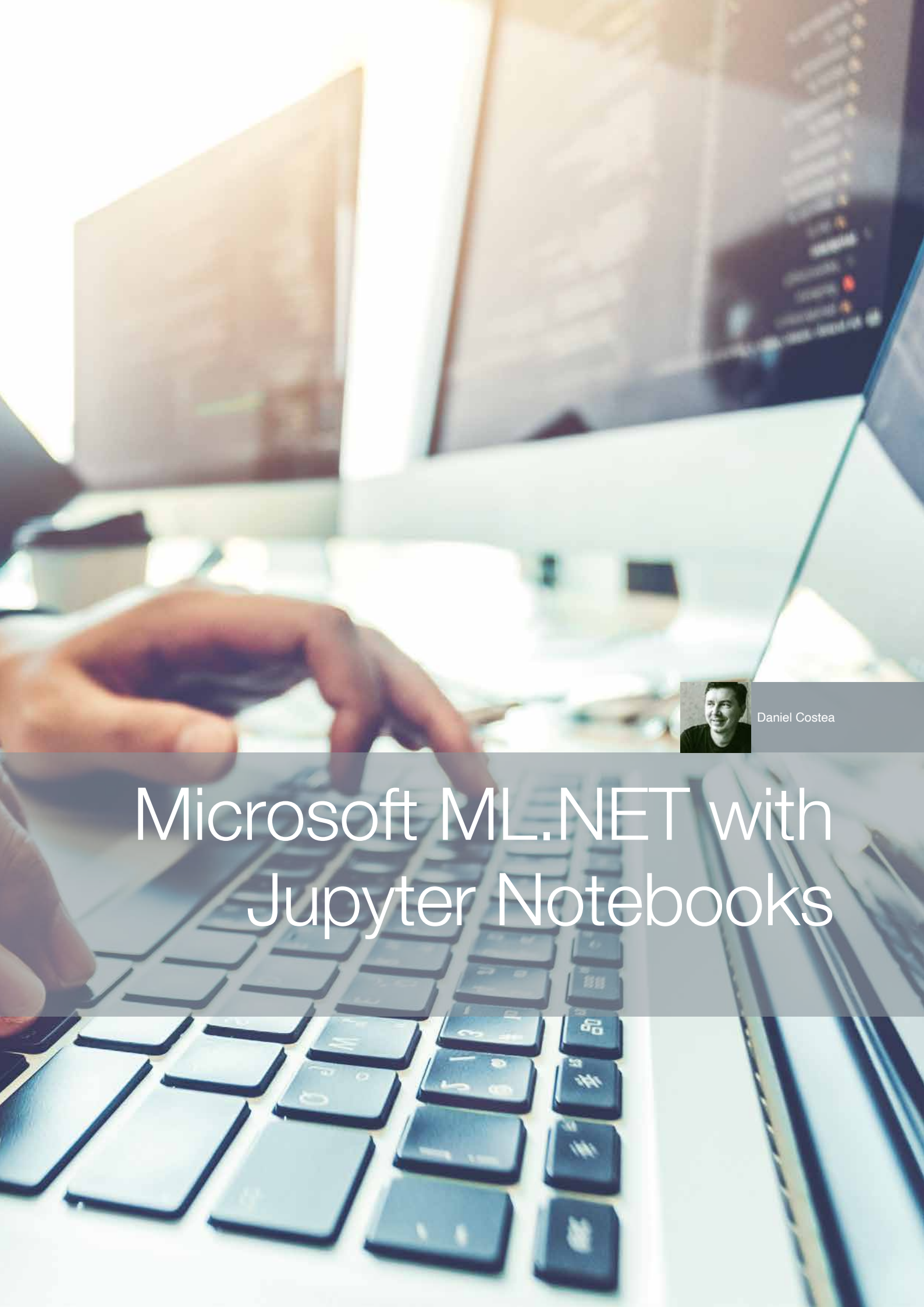
I hope this article improved your awareness of defensive copies. Make sure you make your immutable structs really readonly. Mutable structs should have mainly readonly members. Be careful when using readonly struct fields in .NET Framework and .NET Core pre 3.0. The primitive structs like double, DateTime, long and decimal are not readonly. They will cause defensive copies and slow down the performance.

All source code can be found on https://github.com/sonnemaf/DefensiveCopies

FONS SONNEMANS

Fons Sonnemans is oprichter van Reflection IT, hét maatwerk opleidingscentrum voor Microsoft Software Development sinds 2001.
Als Microsoft MVP is hij een veel gevraagde spreker bij conferenties en seminars. Hij staat bekend om zijn altijd actuele kennis en vakkundige toepassingen van de nieuwste Microsoft technologieën. Hij deelt die kennis en ervaring dagelijks met veel plezier en vakmanschap. De laatste jaren heeft Fons zich verdiept in Writing High Performance C# en .NET code. Als specialist op het gebied van performance wordt hij gevraagd voor consultancy en ontwikkelde hij over dit onderwerp een 2-daagse training.

> USING ILSPY ENGINE IS LIKE LOOKING UNDER THE HOOD AT WHAT IS GOING ON



Daniel Costea

Microsoft ML.NET with Jupyter Notebooks

Is Microsoft ML.NET yet another machine learning framework?
In the world of data science and machine learning, Python programming language is making the rules. In addition to that, the existing frameworks like TensorFlow, Keras, Torch, CoreML, CNTK, are not easy to integrate with .NET projects.
ML.NET is built upon .NET Core and .NET Standard (inheriting the ability to run on Linux, macOS, and Windows) and it's designed as an extensible platform, therefore you can consume models created with other popular ML frameworks (TensorFlow, ONNX, CNTK, Infer.NET). Microsoft ML.NET is more than machine learning, it includes deep learning, and probabilistics and you have access to a large variety of machine learning scenarios, like image classification, object detection.

IS MICROSOFT ML.NET DATA-FRIENDLY ENOUGH?

Python users have Jupyter Notebooks which is a great tool where you can integrate the markdown information, with code and with diagrams.
Until recently ML.NET did not have a good REPL (read–eval–print loop) instrument to play with data, but now, .NET developers can run on-premise interactive machine learning scenarios with Jupyter Notebooks using with C#, F#, or Powershell scripts in a web browser.

PREPARE ML ENVIRONMENT

There are several ways to get started using .NET with Jupyter. If you want to run Jupyter Notebooks locally, you have to install Jupyter Notebook on your machine using Anaconda. If you have successfully installed Jupyter Notebook you can launch it from the Windows menu to open / create a notebook.

DEFINING A PRACTICAL SCENARIO

Let's suppose we have a system able to read temperature, luminosity, infrared and distance to a source (of previously enumerated energy types) and we have a dataset consisting of a few hundred observations. These observations are labeled, that means we have a source for every observation in the dataset. We want to predict the source for a new observation.

LET'S JUMP INTO A JUPYTER NOTEBOOK

One of the most important features of Jupyter, XPlot.Plotly is a data visualisation library responsible for rendering some amazing diagrams.

Let's fast forward and presume we have installed the libraries, initialized the machine learning context and loaded the training and testing datasets (some lines are removed for brevity).

There are several ways to get started using .NET with Jupyter.

```
#r "nuget:Microsoft.ML,1.4.0"
using Microsoft.ML;
using Microsoft.ML.Data;
using XPlot.Plotly;

private static readonly MLContext mlContext = new
MLContext(2020);
```

DISPLAY COMMAND

In a Jupyter Notebook we can use Console.WriteLine to print data, but we will love display command since it's able to print text, html, svn and charts, using DataFrame.

```
var features = mlContext.Data.CreateEnumerable<ModelI
nput>(trainingData, true);
display(features.Take(10));
```

features

index	Temperature	Luminosity	Infrared	Distance	CreatedAt	Source	label
0	32.18	6.84	0	28.69	01/03/2020 10:31:27	Day	
1	115	100	43.85	195.19	01/03/2020 10:26:04	Lighter	
2	30.89	73.44	0	24.32	01/03/2020 10:33:01	FlashLight	
3	81.48	100	93.46	400	01/03/2020 10:27:54	Lighter	
4	32.18	6.64	0	76.97	01/03/2020 10:31:06	Day	
5	24.26	14.45	0	17.49	01/03/2020 10:23:42	FlashLight	
6	27.34	70.02	0	193.76	01/03/2020 10:25:27	FlashLight	observation
7	23.48	87.6	0	343.57	01/03/2020 10:22:23	FlashLight	
8	34.11	100	92.38	7.16	01/03/2020 10:28:07	Lighter	
9	31.86	57.52	0	400	01/03/2020 10:33:25	FlashLight	

FIGURE 1

Of course we don't understand much looking at the tabular data but Jupyter brings up some great diagram types with XPlot.Plotly library able to aggregate the data in a more useful way.

PLOT SEGMENTATION

If we need to see more information about our data we can use plot segmentation.

```
var featuresDiagram = Chart.Plot(new[] {
    new Graph.Box { y = featuresTemperatures, name =
"Temperature" },
    new Graph.Box { y = featuresLuminosities, name =
"Luminosity" },
    new Graph.Box { y = featuresInfrareds, name =
"Infrared" },
    new Graph.Box { y = featuresDistances, name =
"Distance" }
});
display(featuresDiagram);
```

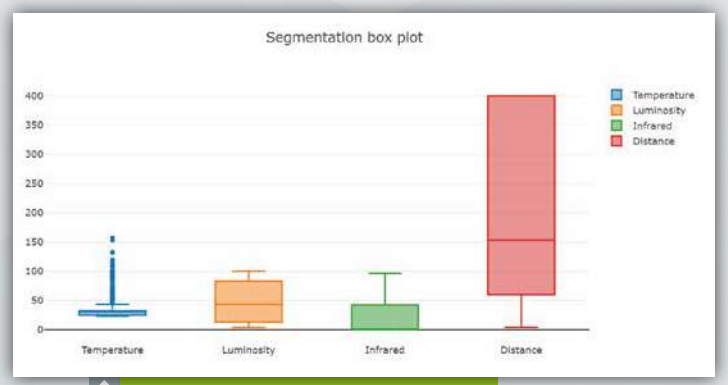


FIGURE 2

Looking at the diagram we can extract valuable information like:

- the median bar from Distance is much higher comparing to the other features
- the min-max values from Temperature and Infrared are not uniformly distributed
- Temperature has many outliers

We can use this information later to improve the model's accuracy, but now there is another thing which doesn't look right, some features look "heavier" than others. Let's see what happens if we try to bring all of them to the same scale, or to normalize them.

```
var preprocessingPipeline = mlContext.Transforms.
Conversion.MapValueToKey("Label")
    .Append(mlContext.Transforms.
CustomMapping(parseDateTime, null))
    .Append(mlContext.Transforms.
Concatenate("Features", featureColumns))
    .Append(mlContext.Transforms.
NormalizeMinMax("Features", "Features"));
```

The features are now normalized (figure 3).

CORRELATION MATRIX

Thinking of data, another question arises, do we really need all the features? Most probably some are less important than others. Correlation matrix is an excellent instrument able to measure the correlation between the features like it follows:

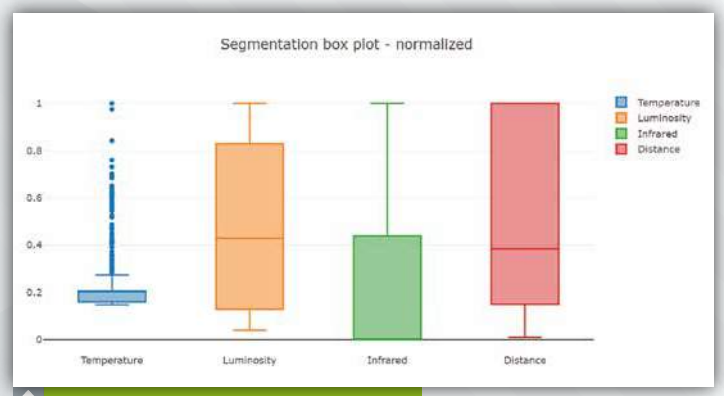


FIGURE 3

- near -1 or 1 indicates a strong relationship (proportionality).
- closer to 0 indicates a weak relationship.
- 0 indicates no relationship.

```
#r "nuget:MathNet.Numerics, 4.9.0"
using MathNet.Numerics.Statistics;

var correlationMatrixHeatmap = Chart.Plot(new Graph.
Heatmap { x = featureColumns, y = featureColumns.
Reverse(), z = z, zmin = -1, zmax = 1 });
display(correlationMatrixHeatmap);
```

FIGURE 4

The strongly correlated features do not convey extra information, therefore they can be removed (it's not the case here!).
For example, our most correlated features are Distance and Infrared (0.48), and Temperature seems to be the most uncorrelated feature compared with the other features.

BUILD THE PREPROCESSING PIPELINE

By convention, ML.NET is expecting to find Features column (as input) and Label column (as output), if you have such columns, you don't have to provide them, otherwise you have to do some data transformations in order to expose these columns to the transformers. In addition, if we need to do binary or multi classification, we have to convert the label to a number using MapValueToKey.

```
var featureColumns = new string[] { "Temperature",
"Luminosity", "Infrared", "Distance" };
var preprocessingPipeline = mlContext.Transforms.
```

```
    Conversion.MapValueToKey("Label")
        .Append(mlContext.Transforms.
Concatenate("Features", featureColumns));
```

In most cases we have more than one relevant feature we might need to train our model and we need to concatenate them into the previously mentioned feature named Features.

BUILD AND TRAIN THE MODEL

Now, we are ready to build the training pipeline using one of the multi-class classifiers, Stochastic Dual Coordinate Ascent (SDCA), but we might want to try other classifiers, as well.

```
var modelPipeline = preprocessingPipeline.
Append(mlContext.MulticlassClassification.Trainers.
SdcaNonCalibrated("Label", "Features"));
var postprocessingPipeline = modelPipeline.
Append(mlContext.Transforms.Conversion.MapKeyToValue("
PredictedLabel"));
var model = postprocessingPipeline.Fit(trainingData);
```

CONFUSION MATRIX

After getting a model, the first thing to concern about should be the model performance and Confusion Matrix is an excellent performance measurement for machine learning classification.
Using the testing dataset we can make predictions and compare the predicted results to the actual results. In the previous diagram we can see the Temperature class was predicted correctly (as Temperature) for 47 times and incorrectly as Infrared for 3 times, and Distance for 2 times, resulting in a 0.9038 precision rate.

Confusion Matrix		Predicted				
		Temperature	Luminosity	Infrared	Distance	Recall
Truth	Temperature	47	0	0	0	1
	Luminosity	0	29	0	1	0.9667
	Infrared	3	0	55	0	0.9483
	Distance	2	0	2	11	0.7333
	Precision	0.9038	1	0.9649	0.9167	total = 150

FIGURE 5

PERMUTATION FEATURE IMPORTANCE

Another useful instrument is the Permutation Feature Importance (PFI) diagram. PFI computes the importance score for each feature by random permutations of the feature, resulting in more or less important features.

```
var pfi = mlContext.MulticlassClassification.Permutat
ionFeatureImportance(linearPredictor, transformedData,
permutationCount: 3);
 var sortedMetrics = pfi.Select((metrics, index) =>
 new { index, metrics.MacroAccuracy }).OrderBy(feature
 => Math.Abs(feature.MacroAccuracy.Mean)).
 Select(feature => feature.MacroAccuracy.Mean);
 var pfiDiagram = Chart.Plot(new[] {
    new Graph.Bar { x = sortedMetrics, y =
featureColumns.Reverse(), orientation = "h", name =
"Permutation Feature Importance" }
});
display(pfiDiagram);
```

In our diagram the Distance feature seems to have a very low score. Dropping it will not result in a significant reduction in our model performance.

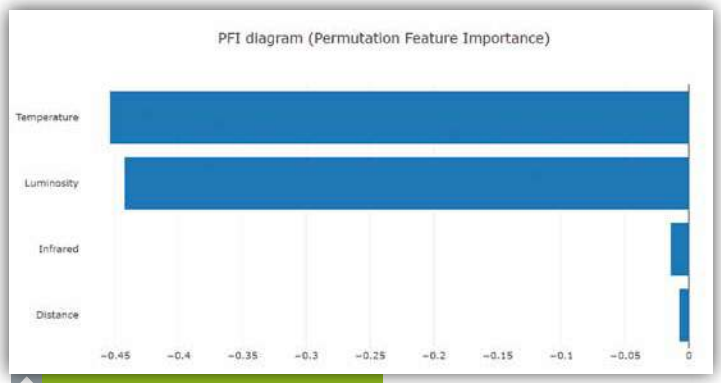


FIGURE 6

CONCLUSION

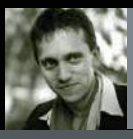
We have a platform and we have a set of instruments to iteratively improve the performance of our model. When we are pleased with the model accuracy, we can start to consume the model, making predictions using new data.

If you want to play with the Jupyter Notebook, you can find it at the address: https://github.com/dcostea/DotnetJupyterNotebooks

DANIEL COSTEA

Trainer. Developer. Speaker. Event organizer. All related to .NET Technologies. These are the ingredients that make Daniel love his job.

> **THINKING OF DATA, ANOTHER QUESTION ARISES, DO WE REALLY NEED ALL THE FEATURES?**



Gill Cleeren

Hello Blazor!

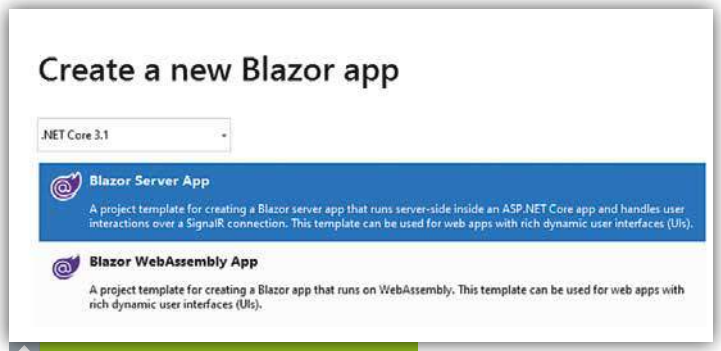
Toen Microsoft in 2019 Blazor de eerste keer voorstelde, was ik meteen verkocht. Blazor laat toe om C# code uit te voeren in de browser en zo kunnen .NET developers hun kennis gaan toepassen voor het bouwen van interactieve web applications. In dit artikel gaan we samen kijken wat Blazor is, welke dingen plots mogelijk worden en hoe het development model in elkaar zit.

Laat ons eerst kijken wat Blazor echt is. Een goeie plaats is daarvoor zeker de Microsoft Docs site en daar zien we dat Blazor beschreven wordt als “a framework to build interactive web UIs using C# and HTML”. Ondanks dat het een korte beschrijving is, heeft deze heel wat lading. Blazor is een framework dat ons als .NET en C# developer toe laat om web apps te bouwen. En dan hebben we het niet over server-side web apps zoals we dat al vele jaren kunnen met ASP.NET. Nee, het gaat hier om interactieve experiences in de browser. Met C#? Inderdaad! Normaal verwachten we hier dat we JavaScript moeten gaan gebruiken. Terwijl JavaScript zowat de meest gebruikte development language is, is ze niet meteen door iedereen graag gezien. Met Blazor wordt het nu dus mogelijk om web apps te bouwen maar dan zonder JavaScript.

Hoe kan dat plots gaan werken? De browsers gaan toch niet plotseling C# code begrijpen? Klopt inderdaad, dat zou niet meteen een interessant optie zijn. De oplossing die gebruikt wordt is een nieuwe standaard genaamd WebAssembly. WebAssembly, of afgekort WASM, laat toe om binary code te draaien in de sandbox van de browser. Die code kan het resultaat zijn van een server-side compile die dan op de client wordt uitgevoerd. Zie het als een DLL (of EXE) die binnen de context van de browser zal draaien op de client. WASM is een standaard die door alle moderne browsers wordt ondersteund en dus geen plugin vereist om uit te voeren. Vermoedelijk heb je tijdens het lezen van bovenstaande paragrafen al gedacht aan Silverlight. En geheel terecht! Terwijl Silverlight ook als target had om rich client apps te bouwen die op de client uitvoerden, was alles hier afhankelijk van een plugin. Wanneer de plugin niet ondersteund wordt, zitten we dan meteen met een probleem. WebAssembly vereist niets, enkel een moderne browser. En wat met mobile browsers? Ook geen probleem: Safari op je iPhone, Chrome op je Android… Ze ondersteunen vandaag al allemaal WebAssembly. Enkel IE valt uit de boot, maar dat is voor de meesten vandaag geen al te groot probleem meer.
Blazor code, C# samen met HTML, zal dus op de client uitvoeren en daar dezelfde opties mogelijkheden krijgen als JavaScript vandaag. Blazor is nog nieuw en niet alles is al voorzien. Sommige integraties die vandaag native kunnen vanuit JavaScript zullen nog moeten verlopen via een JavaScript bridge, maar dat zal mettertijd allemaal aangevuld worden.
Voor we kijken naar een eerste applicatie met Blazor, moeten we het even hebben over de verschillende modellen van Blazor die er vandaag bestaan.

DE BLAZOR MODELLEN

Met Blazor heeft Microsoft een ietwat speciale strategie opgezet voor een werkende versie in de handen van developers te krijgen. Een eerste “echte” preview is verschenen in het eerste kwartaal van 2019. Ondanks heel wat updates tussen toen en nu, zijn heel wat developers aan de slag gegaan met de code-base en zijn er al heel wat applicaties op de markt gebouwd met Blazor.
Er zijn echter vandaag 2 manieren, laat ons zeggen modellen, van de Blazor stack: server-side Blazor en client-side Blazor.



FIGUUR 1

Server-side Blazor werd officieel gereleased samen met .NET Core 3.0 in september. Zoals de naam al weggeeft, draait de code hier op de server en dus niet in het WebAssembly model zoals eerder beschreven. Wanneer de gebruiker naar een Blazor applicatie gaat, zal de code uitvoeren op de server maar door gebruik te maken van een SignalR connectie, lijkt het alsof de code draait op de client. De initiële download is klein aangezien de code zal draaien op de server, maar heel wat voordelen, zoals bv offline gebruik, vallen weg.
Client-side Blazor, waarbij de ganse applicatie dus naar de client wordt gebracht en uitvoert binnen de context van de browser, wordt in mei 2020 verwacht. Hierbij zal de volledige kracht van Blazor naar buiten komen, en voordelen zoals offline werken met een echte in-browser app, zullen beschikbaar worden. Het grote voordeel aan deze manier van vrijgeven van de verschillende versies is dat de code tussen beide versies van Blazor 100% identiek zal zijn. Als je vandaag dus een server-side Blazor app maakt, kan je die met enkele lijnen code om te wisselen, omzetten naar een client-side app. Het eigenlijke programming model van beide modellen is identiek. Ondanks dat het 2 producten lijken, is het echt wel 1 product, dat gewoon in 2 verschillende modi kan uitvoeren.
Laat ons nu eens kijken hoe code voor een Blazor applicatie er uit zal zien.

FILE >NEW PROJECT

Blazor applicaties worden geschreven met C# en HTML en gebruiken een component-gebaseerde approach. De HTML die we gaan gebruiken, is standaard markup waarin ook CSS kan en zal gebruikt worden. Alle kennis die je daarin al hebt, zal dus perfect herbruikbaar

zijn in Blazor. Ook de C# code is verre van speciaal. Er wordt gebruik gemaakt van Razor markup, waarbij C# en HTML door elkaar gebruikt worden en de engine zelf zal de overgang detecteren. De naam Blazor komt overigens van een samentrekking van WebAssembly en Razor. Wat je ziet op onderstaande snippet, is een Blazor component. Een component is de building block van elke Blazor applicatie. Een component bevat zoals gezegd een combinatie van HTML en C#. In de snippet is de functionaliteit vervat in een "code" block maar in de meeste gevallen zal je ook gebruik maken van een code-behind gebaseerde approach. In de HTML markup staan variablen, gedeclareerd in C# en hun waarde, zoals hier van @currentCount, zullen at runtime bepaald worden door de uitgevoerde C# code. Op de HTML button wordt een click event handler toegevoegd die een method zal aanroepen binnen de C# code.

```
@page "/counter"

<h1>Counter</h1>

<p>Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

FIGUUR 2

Alles binnen Blazor is een component, gaande van een pagina tot een onderdeel van een pagina, zoals een menu. Components kunnen genest worden en uiteraard ook op meerdere plaatsen binnen de applicatie gebruikt worden. Dat verhoogt de reusability van de code en componenten.

Wanneer deze applicatie nu draait in de browser, zal je, afhankelijk van het gekozen model, de applicatie volledig in een gecompileerde vorm over de lijn trekken (WebAssembly) of op de server laten draaien in de vorm van server-side Blazor.

CONCLUSIE

Blazor is nog maar net op de markt en de interesse in het product is immens. Het is lange tijd geleden dat er nog zo werd uitgekeken naar de release van een nieuw technologie. Blazor maakt scenario's mogelijk die de C# developer al een tijdje niet meer had gezien: niet langer is JavaScript de enige taal die het mogelijk maakt om interactieve web apps te bouwen! Als je meer wil te weten komen over Blazor, heb ik een interessante Pluralsight cursus waarin je end-to-end een Blazor app gaat bouwen beschikbaar via http://www.pluralsight.com/courses/getting-started-blazor

GILL CLEEREN

Gill Cleeren is CTO bij Xpirit Belgium en runt de Techorama conferenties in België en Nederland. Hij is MVP en Pluralsight auteur.


TJ
TEQJOBS.NL
YOUR NEXT TECH JOB
NEW TEQJOB COMING YOUR WAY
Bezoek de site, solliciteer direct of meld je aan om de nieuwste vacatures te ontvangen.

HELP JIJ ACHMEA #1STAPVOOR TE ZIJN?

Bij Achmea geloven we in een service waarbij wij 24 uur per dag, 7 dagen per week, klaar staan voor onze klanten. Daar hebben we de nieuwste technologie voor nodig. En IT'ers die **vooruitkijken**, **lef tonen** en **#1stapvoor lopen**.

Slimme mensen die kunnen bouwen – soms letterlijk – op onze huidige systemen en platformen, en mensen die ook verder kijken naar andere mogelijkheden. Collega's die samen in hoog tempo de technische ontwikkelingen aankunnen. Iets voor jou? Zet jouw stap op **www.werkenbijachmea.nl**.

Zet jouw stap bij Achmea:

> 1. **.NET/Sitecore Developer Centraal Beheer**
> 2. **Front-end Developer Centraal Beheer**
> 3. **iOS Developer Interpolis**

Enthousiast?

Solliciteer nu!

Centraal Beheer · Zilveren Kruis · Interpolis · FBTO · achmea