SDN MAGAZINE

Nummer
142
Mrt 2022

SDN Magazine is
een uitgave van:
SDN

Van en voor Microsoft & .NET professionals



ARM jezelf met Bicep

- React Hooks - Een korte introductie
- Hoe migreer je .NET apps naar AWS
- Realisatie afsprakenportaal - Testen voor toegang



GEORGIA CODES TO HELP STUDENTS INTO THEIR DREAM EDUCATION

What do you want to code for?

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

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

netcompany

Colofon

Uitgave
Software Development Network
26ste jaargang
Nr. 142 • maart 2022

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

Auteurs:
Eduard Keilholz, Jacob Boer, Vincent Hendriks, Menno Jongerius, Francois Bouteruche, Peter Eijgermans, Koen Aerts, Nadine Wolff en Duco Fronik

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

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

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

©2022 Alle rechten voorbehouden. Niets uit deze uitgave mag worden overgenomen op welke wijze dan ook zonder voorafgaande schriftelijke toestemming van SDN. Tenzij anders vermeld zijn artikelen op persoonlijke titel geschreven en verwoorden zij dus niet noodzakelijkerwijs de mening van het bestuur en/of de redactie. Alle in dit magazine genoemde handelsmerken zijn het eigendom van hun respectievelijke eigenaren.

Beste SDN-ers,

Terwijl het buiten stormt zijn wij binnen hard bezig met weer een mooi nieuw SDN magazine, want waar het weer zijn biceps laat zien, doen wij dat natuurlijk ook! Vooruitkijkend op mooiere dagen, blikken we in ieder geval alvast vooruit op Future Tech, die verplaatst is naar 21 juni - en zoals wij hier steeds horen, de langste dag van het jaar is. Klinkt als een top dag om samen met jullie een mooi event te beleven. Verder vinden jullie in dit magazine een QR-code die je gelijk naar de ticket pagina brengt, zo zorg jij dat je er in ieder geval bij bent. Maar met de verplaatsing van Future Tech bekend dat wel dat TEQnation er nu eerst aankomt! 12 mei zal dit feest weer losbarsten. Hebben jullie je tickets al? (www.teqnation.nl).

Maar nu eerst dit mooie magazine, waar vele auteurs en redactieleden weer enorm hun best voor hebben gedaan om een mooie editie voor jullie neer te zetten. Lijkt het jou ook wat om met jouw artikel in het magazine te staan? Mail dan jouw artikel voorstel naar redactie@sdn.nl.

In deze editie hebben in ieder geval de volgende auteurs hun ervaring en mening gedeeld: Eduard Keilholz, Jacob Boer, Vincent Hendriks, Menno Jongerius, Francois Bouteruche, Peter Eijgermans, Koen Aerts, Nadine Wolff en Duco Fronik. Met stuk voor stuk mooie en actuele onderwerpen hebben wij veel plezier gehad de artikelen te mogen proeflezen. En nu is het plezier aan jullie!

Dus heel veel plezier met lezen! Oh, en mocht je je magazine kwijtraken aan een andere collega developer, geen probleem! Mail ons, dan sturen we je gelijk een nieuw exemplaar op. Of laat de collega developer zich meteen inschrijven op www.sdnl.nl

Kelly Verschoor
kelly.verschoor@sdn.nl




Hybrid work teaches you to listen better
Discover IT & the digital way of working at rabobank.jobs

Rabobank

In dit nummer

6 ARM jezelf met Bicep



12 Realisatie afsprakenportaal - Testen voor toegang

14 Agile software development met feature toggles

18 Migrating our trusty ol' .NET Framework applications to AWS, I couldn't believe it!

22 GDPR voor software developers



26 React Hooks - Een korte introductie

31 Ontwikkel Multicloud Ready Microservices met DAPR

37 Als ik later groot ben

40 Het ijs breken

44 Een sprong in het diepe

46 Tips & Trics

6 Misschien heb je al gehoord dat ergens in maart van dit jaar (2021) een nieuwe DSL (of Domain Specific Language) geïntroduceerd is onder de naam Bicep. Deze nieuw ontwikkelde taal is een oplossing om het schrijven van ARM templates simpeler, makkelijker en sneller te maken. Wat is Bicep?

18 I Love My Local Farmer is a fictional company inspired by customer interactions with AWS Solutions Architects. They started a journey from on-premises to the Cloud and share their learnings through their engineering blog on Medium (https://ilovemylocalfarmer.io). Any stories told in this blog are not related to a specific customer. Similarities with any real companies, people, or situations are purely coincidental. The rest of this article is one of the stories already published on their blog.

31 We weten allemaal hoe snel je een cloud native webapplicatie kunt schrijven en deployen naar elke public cloud. Met een paar klikken heb je zo een leeg .NET Core project gemaakt, dependency injectie geïmplementeerd, een aantal APIs ontsloten en gedeployed naar je favoriete compute service. Maar wat als je over de Microservice route wilt gaan?

ARM jezelf met Bicep

Misschien heb je al gehoord dat ergens in maart van vorig jaar (2021) een nieuwe DSL (of Domain Specific Language) geïntroduceerd is onder de naam Bicep. Deze nieuw ontwikkelde taal is een oplossing om het schrijven van ARM templates simpeler, makkelijker en sneller te maken. Wat is Bicep?

Auteur: Eduard Keilholz

ARM is een afkorting voor Azure Resource Manager. De ARM is in de basis verantwoordelijk voor het uitrollen van resources binnen Azure. Je kunt de ARM aansturen door templates, de zogenaamde ARM templates. Deze templates zijn geschreven in JSON en worden geïnterpreteerd door de ARM waaruit instructies komen. Een ARM template schrijf je op een declaratieve manier. Dit betekent dat je omschrijft hoe je wil dat jouw infrastructuur eruit moet zien. Bijvoorbeeld: Ik wil een blauwe auto met 4 wielen. Als je dit template aanbiedt aan de ARM, gaat het kijken of je al een auto uitgerold hebt in de Cloud. Is het antwoord hierop nee, dan wordt jouw auto aangemaakt. Is het antwoord hierop ja, dan wordt gekeken of de eigenschappen van de auto overeenkomen met de in het template omschreven eigenschappen en naar wens aangepast wanneer nodig. Ook als je in het portal van Azure resources uitrolt, genereert het portal een ARM template en maakt een nieuwe 'deployment' aan.
Nu je in de kern weet hoe de ARM werkt, vraag je je misschien af waar die nieuwe Bicep taal nou voor nodig is. Allereerst wordt infrastructuur als ARM template, geschreven in JSON-formaat. Hierdoor zit je ook automatisch vast aan de JSON-standaarden en kun je dus handige functies als string interpolatie niet gebruiken. Ook heeft een ARM template vaste plekken waar je variabelen en parameters declareert, terwijl je die vaak juist graag in de buurt declareert van waar ze gebruikt worden. JSON heeft ook een hoop ceremonie, zoals de vele accolades, aanhalingstekens en soms schema definities. De tooling voor ARM templates (extensies en IntelliSense) zijn over de afgelopen jaren flink verbeterd, maar nog zeker niet zoals ze moeten zijn. Een mooie opsomming van nadelen waar een team mee aan de slag is gegaan om een DSL bovenop ARM templates te maken.
Bicep, de grootste spierbal van je ARM, is niet voor niets gekozen als naam voor dit project. Niet alleen is de taal compact en relatief eenvoudig te leren, het is ook gelukt om een heldere manier te vinden om infrastructuur te beschrijven als code (Infrastructure as Code (IaC)) en is daarmee een zeer goede aanvulling op ARM templates.

Mijn eerste Bicep

Neem als voorbeeld een Web App in Azure. Een Web App heeft een App Service Plan nodig die je kunt gebruiken om in te stellen hoe krachtig de computer moet zijn waar de website op komt te draaien, en om bijvoorbeeld in te stellen hoe de website moet schalen naarmate er meer of juist minder verkeer is. Daarna kun je de Web App zelf uitrollen. Hieronder staat een ARM template, waarmee je deze twee resources kunt uitrollen in Azure: **Voorbeeld 1**
In dit voorbeeld zie je dat tot en met regel 5 er in feite niets gebeurt.

```
1  {
2    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
3    "contentVersion": "1.0.0.0",
4    "resources": [
5      {
6        "type": "Microsoft.Web/serverfarms",
7        "apiVersion": "2021-02-01",
8        "name": "my-app-service-plan",
9        "location": "[resourceGroup().location]",
10       "sku": {
11         "name": "Standard"
12       }
13     },
14     {
15       "type": "Microsoft.Web/sites",
16       "apiVersion": "2021-02-01",
17       "name": "my-web-app",
18       "location": "[resourceGroup().location]",
19       "properties": {
20         "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', 'my-app-service-plan')]"
21       },
22       "dependsOn": [
23         "[resourceId('Microsoft.Web/serverfarms', 'my-app-service-plan')]"
24       ]
25     }
26   ]
27 }
```

1



Pas vanaf regel zes wordt een resource gedeclareerd, de serverfarms resource (dit is de oude benaming van een App Service Plan en onder die naam in de ARM blijven steken). Op regel 22, zie je een expliciete afhankelijkheid naar het App Service Plan, waarmee de ARM een instructie krijgt om te wachten met het uitrollen van de Web App, tot het App Service Plan klaar is. Dit moet ook, want ze zijn afhankelijk van elkaar. In het volgende stukje code worden exact dezelfde resources gedeclareerd, maar dan in Bicep:

Voorbeeld 2

```
1  resource appServicePlan 'Microsoft.Web/serverfarms@2021-02-01' = {
2    name: 'my-app-service-plan'
3    location: resourceGroup().location
4    sku: {
5      name: 'Standard'
6    }
7  }
8
9  resource webApp 'Microsoft.Web/sites@2021-02-01' = {
10   name: 'my-web-app'
11   location: resourceGroup().location
12   properties: {
13     serverFarmId: appServicePlan.id
14   }
15 }
```

2

Je vindt hier precies dezelfde informatie terug, als in de JSON-variant. Met het sleutelwoord 'resource' geef je aan dat je een resource wil declareren. De daaropvolgende naam, is een variabele naam. Dus 'appServicePlan' op regel 1 en 'webApp' op regel 9 zijn namen van variabelen die je kunt noemen zoals je wil. Wat ook meteen opvalt, is dat het aantal regels is teruggedrongen tot 15 (of 14 als we de lege regel weghalen) en toch staat er precies hetzelfde. Bicep wordt zo overzichtelijker en makkelijker te onderhouden.

Op regel 13, vind je dezelfde afhankelijkheid terug als in het JSON-bestand, echter is deze afhankelijkheid impliciet. Ik heb in het Bicep bestand geen afhankelijkheid geschreven van de Web App, naar het App Service Plan, maar omdat de Web App waardes gebruikt uit de resource van het Service Plan (namelijk het ID), herkent Bicep dit wel als een afhankelijkheid. Je mag in Bicep echter ook een afhankelijkheid expliciet maken zoals in dit voorbeeld op regel 10:

Voorbeeld 3

In de het voorbeeld hierboven vind je ook meteen een nadeel van Bicep en dat is wat mij betreft het enige nadeel, dat is dat Bicep regel gevoelig is. Gevoelsmatig zou ik de array met

```
9  resource webApp 'Microsoft.Web/sites@2021-02-01' = {
10   dependsOn: [
11     appServicePlan
12   ]
13   name: 'my-web-app'
14   location: resourceGroup().location
15   properties: {
16     serverFarmId: appServicePlan.id
17   }
18 }
```

3

afhankelijkheden op regel 10, zeker als er maar één afhankelijkheid is, op een enkele regel willen zetten. Dit zorgt er echter voor dat het Bicep bestand niet meer valide is en niet meer transpiled. Transpiled zeg je? Jazeker! Bicep is zoals eerder beschreven, een DSL en compileert niet echt naar iets. Ook is het niet één op één te vertalen naar een ander formaat en daarom is de term transpilen bedacht. Een samenvoeging van 'to translate' en 'to compile'. Het is belangrijk om te begrijpen dat Bicep een vervanger is van de ontwikkel methode, maar niet een vervanger van het uitrol mechanisme. De ARM accepteert nog altijd alleen maar JSON-bestanden. Bicep bestanden moeten dus omgezet worden naar JSON (dus ARM templates) voor ze uitgerold kunnen worden. Tegenwoordig kan de ARM dit zelf, deze stap kun je dus overslaan als je dat wil. Het is wel handig om te weten dat als er echter een foutmelding optreedt tijdens het uitrollen van een template, deze foutmelding refereert naar het JSON-bestand en dus niet naar het Bicep bestand. Foutmeldingen en regelnummers gaan dus allemaal over het JSON-bestand.

Is het verstandig om het transpilen handmatig te doen? In mijn optiek, ja! In professionele ontwikkelomgevingen met een goed CI/CD proces, zou ik zeggen dat het wenselijk is om ergens tijdens een build fase Bicep templates om te zetten naar ARM templates en deze JSON-bestanden te gebruiken voor het verdere verloop van je CI/CD proces. Stel je gebruikt een Azure DevOps pipeline, of een GitHub Actions workflow. Dan kun je een transpile doen van jouw bicep template waarmee je het omzet naar een ARM template. Dit template kun je dan publiceren als Artifact. In de daaropvolgende stages of jobs, kun je dit Artifact dan weer downloaden voor gebruik (het uitrollen naar een development, test, acceptatie en of productie omgeving). Dus de grootste verschillen zitten hem in het ontwikkelen van je infrastructuur als code bestanden. De tooling voor Bicep is vele malen beter. De feedback en IntelliSense die je krijgt als je de Bicep extensie voor Visual Studio Code installeert is niet te vergelijken met de ervaring die je hebt met het schrijven van ARM templates. Het aantal fouten dat je daardoor maakt tijden het schrijven van infrastructuur als code is daardoor veel minder en het is niet ondenkbaar opeens de volledige infrastructuur voor een website de eerste keer foutloos te schrijven.

Parameters

Parameter bestanden blijven onveranderd als je gebruik maakt van Bicep. Omdat een Bicep bestand voordat het uitgerold wordt, eerst omgezet wordt naar een ARM template, worden ook parameter bestanden nog steeds in JSON geschreven. Hieronder staat een voorbeeld van een parameter bestand: **Voorbeeld 4**

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "queues": {
      "value": ["firstqueue", "secondqueue"]
    }
  }
}
```

4

Dit parameter bestand bevat één parameter met de naam 'queues' en als waarde een array. Deze array bevat twee namen voor een queue. Dit parameter bestand hoort bij het volgende template: **Voorbeeld 5**

```
param queues array

resource serviceBus 'Microsoft.ServiceBus/namespaces@2021-06-01-preview' = {
  name: 'sdndemo'
  location: resourceGroup().location
  sku: {
    name: 'Standard'
  }
}

resource serviceBusQueue 'Microsoft.ServiceBus/namespaces/queues@2021-06-01-preview' = [for queue in queues: {
  name: queue
  parent: serviceBus
}]
```

5

Op regel 1 zie je de parameter gedeclareerd worden. Wanneer dit template wordt uitgerold zonder dat een waarde voor 'queues' gespecificeerd is, faalt het template. Tijdens het uitrollen kun je op verschillende manieren een waarde toekennen aan parameters, en een parameter bestand zoals hierboven is er één van. Meestal maak je per stap in je OTAP-straat, een parameter bestand om zo verschillen te kunnen definiëren tussen bijvoorbeeld een test- en een productieomgeving. Zo wil je misschien de SKU (hoe 'zwaar' de server ingericht moet zijn in termen van RAM en CPU) van een webserver in de test omgeving niet zo zwaar hebben als in de productieomgeving om kosten te besparen. Parameter bestanden zijn hierbij een uitkomst.
Parameters kun je decoreren met aanvullende informatie. Bijvoorbeeld met een omschrijving om meer informatie te geven wat je als waarde verwacht. Ook kun je bijvoorbeeld een lijst van toegestane waardes opgeven. Onderstaand zie je een aantal voorbeelden van decorators voor parameters: **Voorbeeld 6**
Voorgaand template rolt een Service Bus namespace uit, en als dat klaar is, worden gelijk te queues (gedefinieerd in het parameter bestand) aangemaakt. Wanneer je dit template uitrolt zit de service bus in Azure er als volgt uit: **Voorbeeld 7**

```
@allowed([
  'dev'
  'test'
  'prod'
])
param environmentName string

@minValue(1)
@maxValue(10)
param numberOfInstances int

@secure()
param superSecretPassword string
```

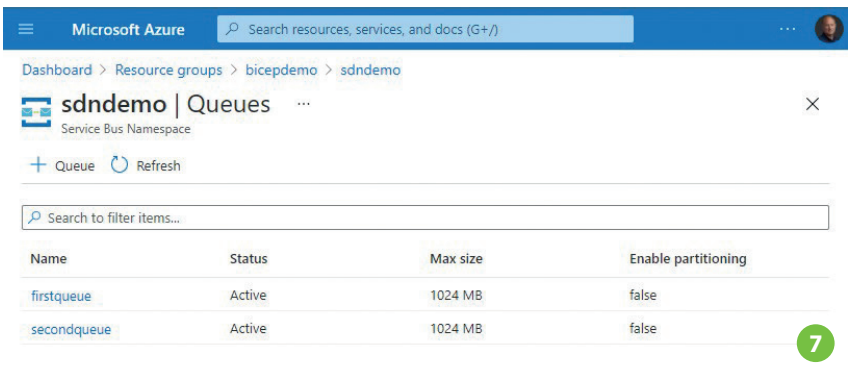
6

Functies

Bicep kent een lijst aan ingebouwde functies die je kunt gebruiken. In de voorgaande voorbeelden zie je bijvoorbeeld de resourceGroup() functie. Met deze functie kun je informatie ophalen over de Resource Group waar je op dat moment een resource in uit aan het rollen bent, in dit voorbeeld wordt een resource in dezelfde locatie uitgerold als de Resource Group. Maar er zijn nog veel meer functies die je kunt gebruiken. Bijvoorbeeld de volgende string functies: contains(), replace(), substring(), toLower() en toUpper() zijn enkele van de string functies die Bicep kent. Het heeft geen zin om alle functies individueel te behandelen want het zijn er simpelweg te veel.

Outputs

Soms heb je informatie nodig die beschikbaar komt door het uitrollen van een resource. Bijvoorbeeld de connectiestring van een storage account. Dit soort informatie kun je als output waarde definiëren in een bestand. **Voorbeeld 8**
Dit voorbeeld is al een stuk ingewikkelder. Eerst de parameters, deze krijgen een waarde van de aanroepende partij, behalve 'administratorUsername'. Die krijgt een default waarde (lege string). Je ziet dat de parameter voor het wachtwoord een '@secure()' decorator heeft. Dat betekent dat de inhoud gevoelig is en wanneer deze waarde bijvoorbeeld naar een log geschreven wordt, zie je sterretjes in plaats van de werkelijke waarde. Dan worden twee variabelen gedeclareerd voor het bepalen van de SQL Server naam en



7

```
1   param defaultResourceName string
2
3   param administratorUsername string = ''
4   @secure()
5   param administratorPassword string
6
7   var resourceName = '${defaultResourceName}-sql'
8   var administratorLogin = administratorUsername ?? '${defaultResourceName}-admin'
9
10  resource sqlServer 'Microsoft.Sql/servers@2021-02-01-preview' = {
11    name: resourceName
12    location: resourceGroup().location
13    properties: {
14      administratorLogin: administratorLogin
15      administratorLoginPassword: administratorPassword
16      publicNetworkAccess: 'Disabled'
17    }
18  }
19
20  output sqlServerName string = sqlServer.name
21  output sqlServerLogin string = administratorLogin
22  output sqlServerPassword string = administratorPassword
```

8

de administrator gebruikersnaam. Als de gebruikersnaam parameter een waarde heeft, dan wordt deze waarde gebruikt. Anders verzint het templates een default waarde ('${defaultResourceName}-admin'). Hier zie je ook meteen een mooi voorbeeld van hoe string interpolatie werkt in Bicep. Onderaan het voorbeeld zie je dat de naam van de SQL Server, de gebruikersnaam en het wachtwoord als output variabele beschikbaar gemaakt worden.

Modules

Als je met Infrastructuur als Code (IaC) jouw Cloud infrastructuur beschrijft, merk je dat deze bestanden, ook met Bicep, toch best groot kunnen worden. Het is dan prettig als je de mogelijkheid hebt om deze infrastructuur in logische blokken op te kunnen delen zodat jouw infrastructuur eenvoudiger leesbaar temaken en te onderhouden is. Met ARM templates gebruikte je hier 'linked templates' voor met alle nadelen van dien. In Bicep, heet deze functionaliteit een module. Ik vind het zelf prettig om per resource type een apart Bicep bestand te maken zodat ik deze in latere projecten makkelijk kan hergebruiken. Dan maak ik een Bicep bestand waar ik deze resources in combineer (bijvoorbeeld een SQL Server en een SQL Server database), en dit bestand roep ik dan aan vanuit mijn hoofd template. Dit is niet per definitie de beste methode, maar wel een handige manier om Bicep templates klein, overzichtelijk, onderhoudbaar en herbruikbaar te maken. Het voorgaande voorbeeld van de SQL Server is hier een mooi voorbeeld van. Het volgende voorbeeld, omschrijft een SQL Server database:

Voorbeeld 9

Ook hier zie je weer parameters die helpen om de naam van de database te bepalen, maar op regel 14, zie je dat in deze naam ook de naam van de SQL Server nodig is om aan te geven op welke SQL Server de database gezet moet worden. Je kunt de bovenstaande voorbeelden als bestand opslaan. Ik vind het prettig om hierbij de namespaces aan te houden die Microsoft voor de resources gebruikt. Dus 'Sql' voor de SQL Server (regel 10 in het vorige voorbeeld) en 'SQL/servers' voor de database (regel 12 in bovenstaand voorbeeld). Dit zijn mappen om Bicep bestanden te organiseren. Ook hier geld dat dit niet hoeft, het is slechts mijn persoonlijke voorkeur. Mocht je dit voorbeeld willen gebruiken, maak dan de volgende bestandsstructuur aan:

> main.bicep
> Sql/servers.bicep
> Sql/servers/databases.bicep

BIO

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/eduard-keilholz/



```bicep
param defaultResourceName string
param sqlServerName string

param sku object = {
  name: 'Standard'
  tier: 'Standard'
  capacity: 10
}

var resourceName = '${defaultResourceName}-sqldb'

resource database 'Microsoft.Sql/servers/databases@2021-02-01-preview' = {
  location: resourceGroup().location
  name: '${sqlServerName}/${resourceName}'
  sku: sku
  properties: {}
}
```

9

Het voorgaande SQL Server voorbeeld, neem je over in Sql/servers.bicep en het bovenstaande SQL Database voorbeeld neem je over in Sql/servers/databases.bicep. In het volgende voorbeeld worden de twee gecombineerd waardoor een SQL Server uitgerold gaat worden, met een SQL Database erop. Deze Bicep code komt in het bestand main.bicep:

Voorbeeld 10

```bicep
param systemName string

@allowed([
  'dev'
  'test'
  'prod'
])
param environmentName string

@secure()
param superSecretPassword string

var defaultResourceName = '${systemName}-${environmentName}'

module sqlServerModule 'Sql/servers.bicep' = {
  name: 'sqlServerModule'
  params: {
    defaultResourceName: defaultResourceName
    administratorPassword: superSecretPassword
  }
}

module sqlDatabaseModule 'Sql/servers/databases.bicep' = {
  name: 'sqlDatabaseModule'
  params: {
    defaultResourceName: defaultResourceName
    sqlServerName: sqlServerModule.outputs.sqlServerName
  }
}
```

Het template hierboven maakt gebruik van de kleinere Bicep bestanden die in de voorgaande voorbeelden gemaakt zijn. Er wordt een standaard resource naam opgemaakt aan de hand van de systeemnaam (bijvoorbeeld 'sdndemo') gecombineerd met een omgeving naam (development, test of productie). Op regel 15 en 23 zie je dat het sleutelwoord 'resource' is vervangen door 'module' om aan te geven dat een module gebruikt wordt. Ook hier ken je een variabele naam aan toe gevolgd door het pad naar het Bicep bestand. Op regel 15 zie je dat de module voor het aanmaken van de SQL Server aangeroepen wordt en op regel 23 zie je dat de module voor het aanmaken van de SQL Database aangeroepen wordt. Op regel 27,

10

word de naam van de SQL Server doorgegeven aan de SQL Database module, door deze uit te lezen uit de SQL Server module. Hier wordt dus gebruik gemaakt van een van de output parameters van de voorgaande module. Het voorgaande voorbeeld van de SQL Server en database dit gecombineerd worden in één Bicep bestand is een mooi voorbeeld van herbruikbaarheid. Die lijn zou je nog verder door kunnen trekken zodat je een overzichtelijke set aan (herbruikbare) Bicep bestanden hebt.

Alternatieven

En hoe het dan zit met alternatieven zoals Terraform en Pulumi? Het antwoord zit al verstopt in de vraag. Het zijn alternatieven, met name als je (alleen) met Azure werkt. Terraform en Pulumi kunnen beide meerdere platformen aan zoals Azure, AWS en Google. Soms kunnen zelfs hosting providers als GoDaddy en zelfs Azure DevOps aangesproken worden. Als je dergelijke oplossingen al gebruikt, is er niet een goede reden om opeens naar Bicep over te gaan. Mocht je nog geen IaC invulling hebben voor jouw project(en), is het echt raadzaam hiermee te starten. Bicep is DE nieuwe oplossing toegespitst op Azure. Wil je zonder een extra andere tool te leren willen starten met IaC, dan is Bicep absoluut aan te raden. Moet je per direct overstappen als je nu al ARM templates ingericht hebt voor al jouw projecten? Nee zeker niet. Het werken met Bicep ervaar ik ontzettend prettig, vooral ten opzichte van ARM templates. Dit betekent echter niet dat je alles opeens om moet zetten. Het is wel handig om te weten dat je met de Azure CLI jouw ARM templates automatisch om kunt zetten naar Bicep.

Conclusie

De syntax, de feedback tijdens het schrijven, het onderhoud en het gemak van modules maken het schrijven van infrastructuur een stuk aangenamer dan voorheen, wanneer je ARM templates gebruikt. Het blijft echter wel zo dat Bicep altijd vertaald wordt naar een ARM template JSON-bestand, dus verwacht niet dat er opeens nieuwe mogelijkheden bijgekomen zijn. De drempel om IaC te gebruiken is nog nooit zo laag geweest, tegen alle voordelen die erbij komen kijken. Soms moet je om te starten even door de zure appel heen, dat is in dit geval echter meer dan de moeite waard! Mocht je echt alles willen weten over het automatisch uitrollen van resources in Azure, kun je overwegen het boek 'Azure Infrastructure as Code' te kopen dat ik samen met Henry Been en Erwin Staal geschreven heb: https://www.manning.com/books/azure-infrastructure-as-code.



Realisatie afsprakenportaal - **Testen voor toegang**

Lang niet alles laat zich plannen en soms komen dingen gewoon op je pad. Voor ons valt in deze categorie zeer zeker het project Realisatie Testen voor toegang portaal en is uitgegroeid tot een geweldig visitekaartje voor Vitas. Veel mensen in Nederland kennen deze site of hebben er inmiddels gebruik van moeten maken. Het equivalent en afgeleide hiervan, Testenvoorjereis.nl hebben we ook mogen realiseren en is operationeel geweest tot 1 oktober 2021.

Auteur: Jacob Boer

Wij, Vitas, een specialist op het gebied van software-implementaties op basis van Microsoft-technologie, kregen eind maart 2021 een telefoontje. Of wij in staat waren om op zeer korte termijn een ontwikkelteam op de been te brengen voor het ontwikkelen van de Testen voor Toegang portal. Uitdagende factor daarbij is dat de overheid een harde deadline heeft afgekondigd (1 mei 2021) waarin we in Nederland in staat moeten worden

gesteld om testen uit te voeren om toegang te krijgen tot evenementen in Nederland. Deze variabelen bij elkaar maakt het, zacht uitgedrukt 🙂, een uitdagend traject om in te stappen én om er uiteindelijk commitment voor te kunnen afgeven.
Vitas krijgt het voor elkaar om in 2 weken tijd een team van 8 personen op de been te brengen. Het team bestaat uit 4 backend ontwikkelaars, 2 front-end ontwikkelaars, een tester en een scrummaster. Vanuit de opdrachtgever is een product owner fulltime onderdeel van het team. Commitment vanuit het team is de eerste stap. Na een intensieve planningssessie waarin hoofd- en bijzaken goed zijn gescheiden komt de klap erop, we gaan ervoor!

Veel mensen in Nederland kennen deze site of hebben er inmiddels gebruik van moeten maken.

Technisch gezien het neusje van de zalm: alles opgetrokken in Microsoft Azure, geschreven in .NET Core met C#, een Domain Driven Design aanpak (waarbij alles in kleine geïsoleerde softwarecomponenten via volledig geautomatiseerde release pipelines in Azure wordt uitgerold), de Webui is geschreven in Vue.JS (waarbij gebruik gemaakt werd van scenario gebaseerde end-to-end unittesten door middel van Cypress), specifieke front-end unittesten in de vorm van JEST en in de geautomatiseerde releaseketen wordt gebruik gemaakt van SonarQube voor statische code analyse.
Alles draaiend in AppServices met daaronder een SQL-database, waarbij de applicatie kan meeschalen met de vraag (en dat is soms hard nodig gezien de piekbelastingen die er zijn).
Het was bikkelen, bikkelen en nog eens bikkelen. Maar met een enthousiast en gemotiveerd team, waarbij iedereen elkaar hielp, het klokje ronddraaide en met een heel duidelijk gedefinieerd sprintdoel om collectief naar toe te werken. Fantastisch om te zien dat de sprintdoelen behaald zijn en dat we als team de livegang in de eerste week van mei in 2021 hebben mogen vieren.
Als Vitas hadden we het idee dat de applicatie vanzelf zou worden uitgefaseerd en dat we de applicatie "op de plank" konden leggen. Immers de vaccinatiegraad stijgt en de druk op de zorg neemt af, Nederland wordt meer vrijgegeven en daardoor kan ook weer meer.
De praktijk is soms weerbarstig weten we inmiddels. Anno februari 2022 wordt er nog steeds met het voltallige team gewerkt aan het project. Tijdens de afgelopen periode zijn ook veel nieuwe functionaliteiten toegevoegd, zoals: een volledige beheermodule om het aansluiten van nieuwe teststraten te vergemakkelijken, decentraal kunnen beleggen van rechten/autorisaties voor de testlocaties om bijvoorbeeld capaciteitsplanningen (met bijbehorende beveiligingsvraagstukken) te maken en te bepalen. Naar verwachting gaat het team aan het einde van Q1 2022 toewerken naar de beheerfase.
De omstandigheden waaronder het ontwikkelteam moet werken, zijn ook verre van eenvoudig. Thuiswerken en nog eens thuiswerken, met Teams als één van de belangrijkste tools. Ondanks dit weet iedereen elkaar te vinden en wordt er goed samengewerkt. Eén van de dingen die helpt om het leuk te houden is de wekelijkse virtuele vrijdagmiddagborrel; even met elkaar (in Teams) kletsen over andere zaken dan werk.
Alles bij elkaar een project waar Vitas heel trots op is. Hoe leuk is het om als bedrijf en medewerker op deze manier te kunnen bijdragen aan een maatschappelijk relevant onderwerp.

BIO

Jakob Boer

Jakob Boer is als senior consultant werkzaam bij Vitas. Hij is begonnen als SharePoint-developer en heeft zich ontwikkelt richting Azure. De laatste tijd richt Jakob zich als projectmanager op de interne projecten binnen Vitas. Bij het project Testen voor Toegang vervult hij de rol van Scrum master. Jakob vindt het fijn om met een team van professionals leuke en uitdagende projecten te doen, een categorie waar Testen voor Toegang zeker in valt!



Agile software development **met feature toggles**

Heel vaak gaat het over agile software development waarmee gesproken wordt over "Scrum", "Kanban", etc. Als het goed is zouden deze populaire werk methodieken ervoor moeten zorgen dat teams gaan reflecteren en verbeteren, maar dan heb je ook de tools nodig om je softwareproduct op een agile manier te ontwikkelen.

Auteur: Vincent Hendriks

Een typisch kenmerk van een softwareproduct wat echt met een Agile mindset ontwikkeld is, is namelijk het feit dat er vaak en regelmatig gereleased kan worden. Dit doe je door zo veel mogelijk te automatiseren op het gebied van testing en deployment, maar ook door te zorgen dat je kleine stappen kunt maken. Wat je dan dus niet wilt hebben zijn langdurige feature branches.
Iedere ontwikkelaar, in een wat groter team, weet dat het hebben van branches die lang openblijven een vervelend iets is. Je krijgt merge conflicten, maar het houdt soms ook grotere changes tegen want vaak heerst er een mindset: "Laten we eerst die branch nou maar afronden voordat we verder gaan met een andere change".
Hoe los je dit op? Het is een mooi streven om branches zo kort mogelijk te laten leven, maar wanneer een feature simpelweg veel tijd kost om helemaal te ontwikkelen dan heb je een uitdaging. Je kunt het werk in kleinere brokken opdelen, maar als het niet acceptabel genoeg is om naar je volledige userbase vrij te geven dan kun je de branch niet mergen naar de branch waarvandaan je released. Feature Toggles bieden hiervoor oplossingen voor. Wat daarnaast ook speelt is dat je zo snel mogelijk resultaten wilt tonen en testen. Natuurlijk kan dit op een test omgeving, maar wat nou als je feedback wilt ophalen van een select aantal gebruikers? Wat perfect zou zijn is wanneer je dit ook gewoon in productie kunt proberen.
Maar breng je dan gewoon releases uit die half af zijn? Nee, zeker niet!

> Hoe langer een branch open blijft staan hoe groter de kans op merge conflicten en simpelweg conflicterende aanpassingen.

Feature toggles als oplossing

Je kunt feature toggles gebruiken om functionaliteiten aan of uit te zetten.

- **Is de feature niet compleet?** Dan staat de feature uit of alleen vanaf specifieke locaties, ip adressen of gebruikers.
- **Is de feature wel compleet?** Dan kun je de feature aan zetten.
- **Is de feature wel compleet, maar zijn we nog niet helemaal zeker van de mogelijke effecten?** Dan stellen we de feature voor een klein gedeelte open. Dit kan op basis van percentages, maar bijvoorbeeld ook op basis van een land.

Feature Management

Microsoft heeft d.m.v. de 2 NuGet's "Microsoft.FeatureManagement" & "Microsoft.FeatureManagement.AspNetCore" een zeer mooie feature management functionaliteit uitgebracht voor ASP.NET ontwikkelaars. Een eenvoudige feature toggle kan ofwel aan ofwel uitstaan. Echter kun je de "feature toggle" wat dynamischer laten gedragen dan zijn er feature filters.
Een feature filter betekent eigenlijk dat elke keer wanneer een feature toggle gecheckt wordt vanuit een applicatie dit afhankelijk van de

Nieuwe features

Op het moment van schrijven zijn we met de development teams binnen Anycoin Direct druk bezig met een grote feature. Dit is niet een feature die binnen enkele dagen gemaakt kan worden. Wat je dan zou kunnen krijgen is dat er een feature branch gemaakt zou worden waar heel lang aan ontwikkeld gaat worden. Als dat het enige was waaraan ontwikkeld zou worden is dat wellicht geen probleem, maar uiteraard gaat alles gewoon door. Er zijn updates om uit te voeren, bug fixes te doen en andere kleinere features die ook ontwikkeld worden.
Wij kiezen ervoor om geen grote, langdurige feature branches te onderhouden. Zo snel mogelijk naar de branch afronden is onze voorkeur. Wij ontwikkelen deze feature stap voor stap en bij iedere release kunnen wij deze feature testen in productie. Zodra de feature op het niveau is om te gaan releasen dan gaan we de feature releasen voor een kleinere beta gebruikersgroep. Komen daar geen bijzonderheden uit? Dan kunnen we ervoor kiezen om een specifiek land of al onze klanten toegang te geven tot deze nieuwe feature.

context aan/uit kan staan. Je kunt dit bijvoorbeeld gebruiken om afhankelijk van de locatie van de gebruiker de feature toggle aan/uit te zetten. Er zijn 3 verschillen Feature filters mogelijk:

Percentage

Met een Percentage filter kun je ervoor kiezen de feature aan te zetten voor een specifiek percentage. Let hierbij wel op dat dit er bij iedere check een andere waarde terug kan komen. Wil je zorgen dat een gebruiker elke keer een feature wel/niet mag zien dan kun je beter kiezen voor een Targeting filter.

Targeting

Een Targeting filter kun je gebruiken om op basis van specifieke kenmerken zoals een gebruikersgroep of locatie feature toggle aan te zetten. Per groep kun je een percentage specificeren waarmee je kunt kiezen dat een bepaald percentage die voldoet aan de groep de feature te zien krijgt.
Deze filter is een zogenaamde Contextual Filter en gebruikt de request context bij het bepalen of de feature toggle aan moet staan ja of nee. Een perfect voorbeeld hiervoor is bij het gebruik van een percentage binnen deze Targeting filter waarin je zegt dat 50% van de gebruikers de feature mogen gebruiken. Het is niet fijn voor gebruikers om iedere keer een ander resultaat te krijgen. Door kenmerken van de request context te gebruiken krijgt in dit geval de eindgebruiker iedere keer hetzelfde oordeel.

TimeWindow

Een TimeWindow filter kun je gebruiken om op bepaalde tijden de feature toggle aan te zetten. Deze kan natuurlijk ook handig zijn bij speciale "acties" die alleen op bepaalde tijdstippen gelden. Hiermee hoef je dus niet zelf op het specifieke moment aanwezig te zijn om de feature aan en weer uit te zetten. **Voorbeeld 1**

Het gebruik van Feature Toggles

Tot nu toe heb ik het alleen maar gehad over hoe je feature toggles inricht, maar hoe weet je nu in je applicatie of de feature aan staat? Hiervoor zijn meerdere mogelijkheden waarbij ik me voor nu focus op de 2 meest voor de hand liggende manieren in een ASP.NET Core scenario.

Views

Wanneer je een MVC view hebt kun je gebruik maken van Tag helpers. Eerst moet je de Tag helper registreren in bijvoorbeeld je _ViewImports.cshtml: **Voorbeeld 2**
Vervolgens kun je de feature tags gebruiken. **Voorbeeld 3**

Controllers / Actions

Bij je controllers en/of specifieke controller actions kun je simpelweg gebruik maken van een FeatureGate attribute: **Voorbeeld 4**

Azure App Configuration

Wanneer je feature toggles in een "appsettings.json" hebt staan kan het al voordelen hebben. Maar je moet er, in dit geval, wel aan denken om dit bij een deployment goed te zetten. Wil je in deze situatie de

```
"FeatureManagement": {
  "FeatureA": true, // Feature flag set to on
  "FeatureB": false, // Feature flag set to off
  "FeatureC": { // Feature flag which is 50% on and 50% off
    "EnabledFor": [
      {
        "Name": "Percentage",
        "Parameters": {
          "Value": 50
        }
      }
    ]
  }
}
```

1

Een voorbeeld hoe dit in je appsettings.json er uit kan zien.

```
@addTagHelper *, Microsoft.FeatureManagement.AspNetCore
```

2

```
<feature name="FeatureA">
    <p>This can only be seen if 'FeatureA' is enabled.</p>
</feature>

<feature name="FeatureA" negate="true">
    <p>This will be shown if 'FeatureA' is disabled.</p>
</feature>
```

3

```
[FeatureGate(MyFeatureFlags.FeatureA)]
```

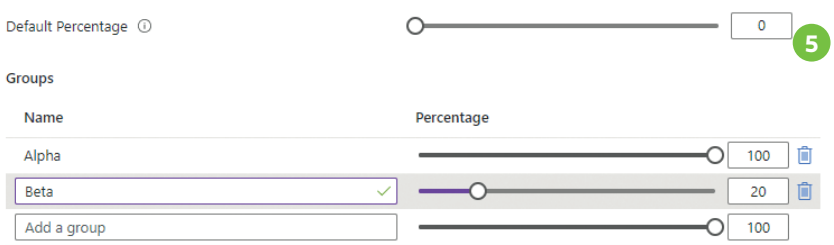
4

feature toggles aanpassen dan moet er eigenlijk een nieuwe deployment gedaan worden. Idealiter zou je geen nieuwe deployment willen doen en dat hoeft ook niet, wanneer je Azure App Configuration gebruikt. Azure App Configuration is een service van Azure die het mogelijk maakt om configuratie in de cloud te beheren. Vanuit een .NET perspectief is het een extra configuration provider. Dynamisch configuratie aanpassingen kunnen doen zonder een deployment is alleen al erg fijn, maar deze service biedt ook de mogelijkheid om specifiek feature toggles te beheren.
Gebruik je Azure App Configuration (wat ik zeker aan raad) dan zou je voor development de appsettings.development.json kunnen gebruiken om features te ontwikkelen en voor bijvoorbeeld test, acceptatie & productie de Azure App Configuration te gaan gebruiken voor een dynamische setting. **Voorbeeld 5**

Conclusie

Feature toggles kunnen een zeer fijne tool zijn om je product agile te blijven ontwikkelen en releasen. Microsoft's Feature Management library i.c.m. Azure App Configuration is een zeer mooie combinatie. Zit je niet in deze stack dan zijn er ook genoeg andere tools en producten van andere leverancies, maar in een .NET minded wereld zit je hier zeker goed.

Als het goed is zouden deze populaire werk methodieken ervoor moeten zorgen dat teams gaan reflecteren en verbeteren.



Vond je dit verhaal interessant en had je nog wat vragen? Neem gerust contact met me op via LinkedIn: https://www.linkedin.com/in/vhendriks. ●

BIO

Vincent Hendriks

Vincent Hendriks is CTO bij Anycoin Direct en een Software Engineer met meer dan 16 jaar ervaring. Vincent heeft een zeer brede ervaring in onderwerpen zoals C#, .NET/.NET Core, Software architectuur, CQRS, Event Sourcing, SQL, Azure, Docker en nog veel meer.



anycoin direct

Wij zijn op zoek naar jou!

Werken met een jonge en gedreven groep developers?

- Nieuwe technologieën
- Positieve sfeer
- Jong team
- Ruimte voor input
- Je collega's zijn je vrienden
- Personeelskorting op crypto
- Potje smash of kart?

Scan de QR code voor video interviews en sfeerbeelden.

Tot snel!



Vincent Hendriks
CTO

Migrating our trusty ol' .NET Framework applications to AWS, **I couldn't believe it!**

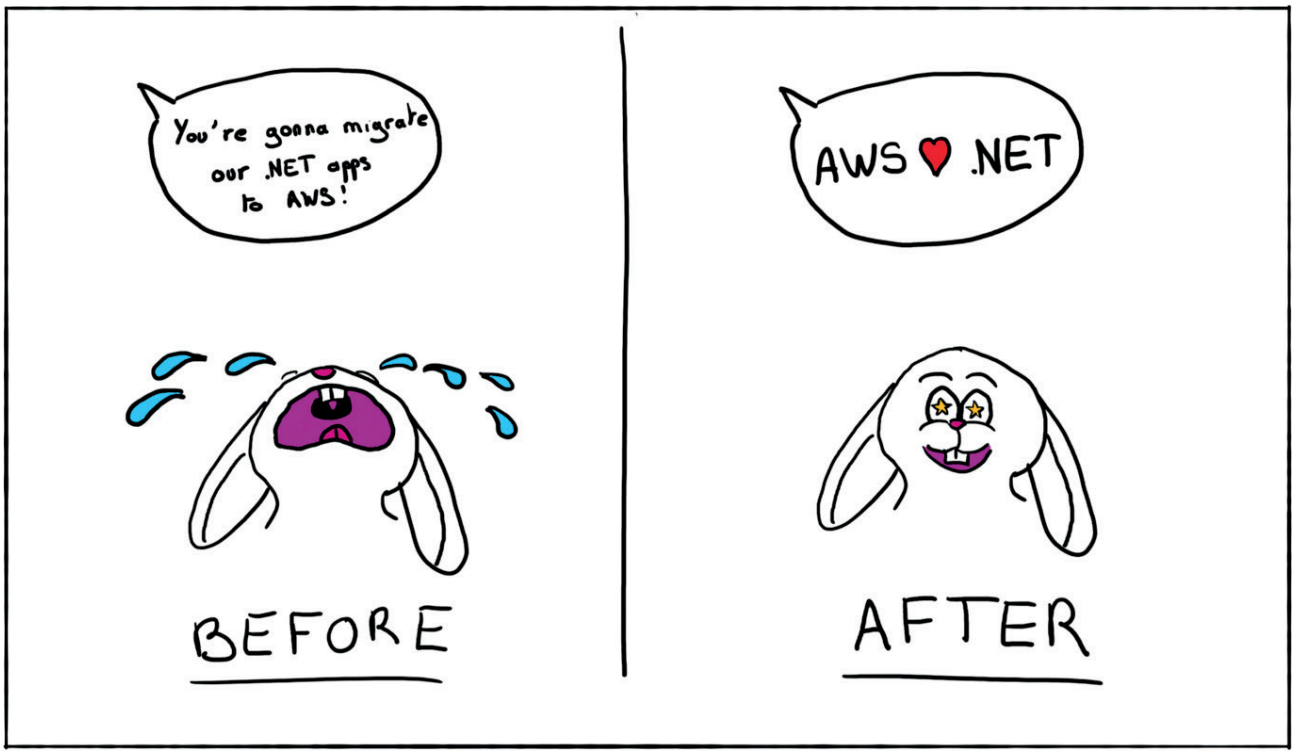
I Love My Local Farmer is a fictional company inspired by customer interactions with AWS Solutions Architects. They started a journey from on-premises to the Cloud and share their learnings through their engineering blog on Medium (https://ilovemylocalfarmer.io). Any stories told in this blog are not related to a specific customer. Similarities with any real companies, people, or situations are purely coincidental. The rest of this article is one of the stories already published on their blog.

Auteur: Francois Bouteruche

When Inès, our CTO, announced that the company has decided to migrate all-in on AWS, as a .NET developer, I was a bit concerned.
My team and I are used to Visual Studio and we are just one click away from another cloud provider since several versions now. Even the .NET documentation offers lots of code examples adapted to this other cloud provider. I was wondering if it was even possible to run .NET applications on AWS.
I think my team and I went through all the steps of the Cycle of Acceptance. We first denied this announcement. Why would we need to migrate our trusty ol' .NET Framework applications to the Cloud? They've worked perfectly well for years. Our platform is very stable and we release new features twice a year (sic). Then we went angry. Why did nobody ask us? We would have told them that if a Cloud migration was really needed, another Cloud would have been easier for us as it is fully integrated in our IDE. I escalated and had a discussion with Inès and our Head of Development. They took time to explain all the good reasons that the company has decided to migrate to AWS. When I got out of this meeting, I was clearly at the depression step of the cycle. How would we handle this?
The light came from, Melina, one of our team members. She attended a .NET online event. An AWS Developer Advocate delivered a talk called "50 shades of .NET on AWS". The key takeaways she has shared with us are that AWS supports Windows Server, SQL Server and .NET since 2008. They also provide a bunch of tools to support .NET developers and numerous hosting options. It was time for us to bargain. We didn't want to be left behind nor become obsolete. So, we started to look into what AWS has to offer us.

Don't take me out of my preferred IDE!

First things first, we love our IDE. We are not alone. Visual Studio was ranked #2 in 2021 StackOverflow survey with 33.03% of respondents using it. We don't want to get out of it. For testing purposes, we love the Right-Click and Publish feature. Of course, we don't use it to deploy into production. We were curious to see if we would be able to deploy our web applications from our IDE directly to AWS.
The answer is "Yes, we can!". AWS provides a Visual Studio extension called AWS Toolkit for Visual Studio. Once you've installed it, you first need to configure your AWS Credentials to access your AWS Account. Then, you get the AWS Explorer view that allows you to interact with the AWS services deployed into your account. Not all AWS Services are supported here, but you will find the services you will most likely use to host your .NET applications. For our .NET Framework web applications, we could Right-Click-Publish to AWS Elastic Beanstalk by default. The first time I saw this name in my Visual Studio IDE, I had no idea what it is but I will come back to this later.



Do you speak my language?

The second thing we explored was how to interact with AWS services from our C# code. We didn't intend to strongly couple our code to AWS services, but we discovered a few services that seemed to offer very low hanging fruits. The most appealing one was Amazon S3. You probably already heard about this cloud object storage service. It looked like an obvious solution to offload serving static content from our IIS servers. But what would have been the point if we had to develop our own .NET Framework client for Amazon S3 API?

We have been surprised to discover that the AWS SDK for .NET was the first official SDK released by AWS more than 10 years ago. It offers client implementations for AWS services with common patterns for authentication and authorization. In addition, AWS provides libraries that extend the integration of AWS services with .NET or popular 3rd party .NET libraries. For example, you can use AWS Logging .NET to integrate popular logging libraries like NLog, Apache log4net, ASP.NET Core Logging or Serilog with Amazon CloudWatch Logs service, a managed logs service.

When the company has decided to migrate all-in on AWS, as a .NET developer, I was a bit concerned.

Power[my]Shell or I bash you

As long time Windows Server users, we learned PowerShell. Our ops team has massively invested in PowerShell to administer and operate our Windows servers environments. Our dev team has learned a bit of PowerShell too to interact with our dev and test Windows Server environments.
We were glad to discover that the AWS Tools for PowerShell provide PowerShell modules. They enable us to script operations on our AWS resources from our usual scripting language. We could use it to create, start and stop our servers for example. Another box checked by AWS that I wouldn't have bet on.

Do you have sweets for us?

It was our very own surprise that AWS fulfills these three basic yet mandatory requirements. At that point, I was a bit ashamed by my own unconscious bias and became hungry for more. Does AWS have sweets for us? Yes, and here are those we love the most.

The curse of choice — our hosting options

Regarding hosting, they offered a full range of options for our .NET Framework applications. It goes from hosting on raw EC2 instances (their name for virtual machines) to deploying Windows containers workload on managed orchestrators. In between, you find managed services like Amazon Lightsail, well-suited for individual developers and small projects, and AWS Elastic Beanstalk that manages your load balancer and your Windows servers for you. The same goes for SQL Server. You can operate it by yourself on EC2 instances or opt in for the fully managed database service Amazon RDS and its SQL Server flavor.

Make me feel at home — support Azure DevOps

Before hosting, you need to deploy. For our CI/CD pipelines, my team has relied on Team Foundation Server for years. They support it! They provide the AWS Toolkit for Azure DevOps, an extension that adds AWS tasks to deploy on AWS from Azure DevOps server. Indeed, the extension works both on premise with Team Foundation Server or on Azure DevOps, the SaaS offering. Mind blowing!

We finally reached the acceptance step. AWS seemed to give everything we needed for migrating our .NET Framework applications.

Back to the future — hello .NET Core / .NET

We know that .NET Framework has no future. Only .NET does. So, porting our applications to .NET is on our roadmap. The release of .NET 6 which is a Long-Term Support version is a compelling event for us to do so. It is now mature enough to taste without any risks. AWS has a nice helper tool here: the Porting Assistant for .NET. It runs a compatibility assessment of your .NET Framework code and assists you while your port your code to .NET.

The one thing which drove me crazy — infrastructure as code in C#

Since we've started our migration to AWS, we have decided to manage our infrastructure as code to follow best practices. AWS offers us to do so by writing C# piece of code using the AWS Cloud Development Kit. For the first time, we can embed in the same Visual Studio solution the code of our application and the code of our infrastructure and get them versioned all together! No need for us to learn a new language! Just plain C# and the relevant NuGet packages. Amazing!

Where are we going now?

We finally reached the acceptance step. AWS seemed to give everything we needed for migrating our .NET Framework applications. So, we've started to work on this. I won't tell you that everything has been a quiet walk in the park, but we made it. And we are more than proud of where we've landed. If you want to know more, follow our adventure on our blog - ilovemylocalfarmer.io.

BIO

Francois Bouteruche

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



aws

AWS Is How Now Go Build

Check out AWS Developer Tools



AWS partners with TEQNATION
teqnation.com/aws-partner-page/

GDPR voor software developers

Toegegeven, het is voor de meeste ontwikkelaars niet het favoriete onderwerp en het is ook niet nieuw, tegelijkertijd is basiskennis van GDPR belangrijk om te voorkomen dat we onszelf in de voet schieten. Dit artikel geeft een samenvatting van onze webinar en we hopen je hiermee op weg te helpen om een basis begrip te hebben van deze belangrijke Europese wetgeving.

Auteur: Menno Jongerius

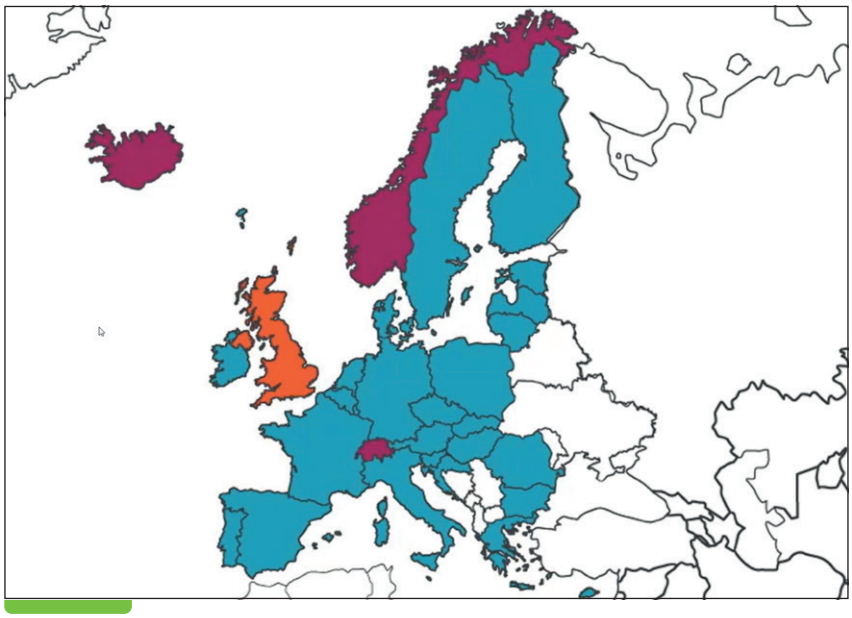
Voor wie is de GDPR van toepassing?

De GDRP wetgeving is bedoeld om personen, lees inwoners van de EU, te beschermen m.b.t. elke vorm van dataprocessing van persoonlijke data. De wetgeving gaat veel verder dan alleen het beschermen van privacy. Zodra je als organisatie gegevens van personen in beheer hebt, dan ben je gebonden aan de GDPR wetgeving. Omdat GDPR gaat over mensen, heeft de wetgeving geen betrekking op gegevens over bedrijven of geanonimiseerde gegevens. Met die laatste categorie moet je voorzichtig zijn, omdat het soms mogelijk is om anonieme data alsnog aan personen te koppelen door databestanden te combineren. In dat geval is de data niet anoniem.

Waar is de GDPR van toepassing?

De GDPR is van toepassing op alle burgers binnen de EU waarbij Noorwegen en IJsland zich aan de wetgeving hebben verbonden. Groot Brittannië heeft inmiddels een zelfstandige, maar identieke versie van de GDPR in gebruik. **Let op** de GDPR is van toepassing op alle verwerking van data van burgers en dus niet van de locatie van het bedrijf dat deze data verwerkt. Dus ook bedrijven die buiten de EU gevestigd zijn, dienen zich te houden aan de GDRP als ze data van burgers van de EU landen verwerken. Als data buiten de EU opgeslagen of verwerkt





Invloedsgebied GDPR.

wordt, moet dit expliciet aan de burger gemeld worden.

Rollen binnen GDPR?

De GDPR kent drie specifieke rollen: de data subject, de data controller en de data processor. De data subject is de formele term voor de levende persoon waar data over verwerkt wordt. De data controller is de rechtspersoon die bepaald hoe en waarvoor data wordt verwerkt. De data processor is een organisatie die in opdracht van de data controller taken uitvoert. Neem een praktisch voorbeeld van data over mijzelf als werknemer. Dan zou ikzelf het data subject zijn, mijn werkgever zou de data controller zijn, en als de werkgever gebruik maakt van een SAAS dienst om mijn personeelsdossier bij te houden, dan is deze SAAS aanbieder de data processor. Data controller en data processor delen binnen GDPR het overgrote deel van de verantwoordelijkheid. De data controller is eindverantwoordelijk, maar een data processor kan zich daar niet achter verschuilen.

Principes binnen GDPR

De GDPR gaat veel verder dan alleen privacy regelgeving. Het gaat over verantwoordelijk omgaan met data van de burgers. Dit betekent:

> Dat data wordt opgeslagen met een rechtmatige reden
> Dat alleen wordt opgeslagen wat nodig is om de taak uit te voeren waarvoor je de data opslaat
> Dat je ervoor verantwoordelijk bent dat de data die je opslaat juist is
> Dat je de data verwijderd (of anonimiseert) als deze niet meer nodig is
> Dat de data veilig wordt bewaard
> Dat je als organisatie aantoonbaar verantwoordelijkheid neemt over de data die je onder je hoede hebt (denk aan ISO certificeringen etc.)

Rechtmatige redenen om data te bewaren

Het is belangrijk om als organisatie na te gaan welke data je beheerd en vooral waarom je die data hebt. Het is alleen toegestaan om data over personen op te slaan als je voldoet aan één of meerdere van de volgende redenen:

> Er is sprake van een contract tussen data subject en data controller
> Er is een wettige reden om data op te moeten slaan (denk aan de belastingdienst die data over burgers moet vastleggen conform wetten die gelden)
> Het vastleggen van de data beschermd een vitaal belang (meestal moet je dan denken aan zaken rondom gezondheidszorg)
> De data dient een publiek belang of is vastgelegd door een overheidsinstantie
> Er is sprake van een zwaarwegend belang om de data vast te leggen (dit is voer voor juristen)

In alle andere gevallen mag data alleen worden vastgelegd op basis van toestemming van de persoon in kwestie. Dit wordt ook wel op basis van "consent" genoemd.

Speciale categorie data

Binnen de GDPR zijn er speciale categorieën van data die ervoor zorgen dat je aanvullende maatregelen moet nemen in de onderbouwing waarom je de data gebruikt, en in de beveiliging van de data. Het gaat om de volgende vier categorieën:

> Data die kan bijdragen aan discriminatie. Dit is van toepassing op data over afkomst of ras, maar ook

basiskennis van GDPR is belangrijk om te voorkomen dat we onszelf in de voet schieten.

bijvoorbeeld data m.b.t. geloofsovertuiging.
> Data die een verhoogd risico geeft op identiteitsfraude. Strikt genomen zijn vooral biometrische of genetische data benoemd, maar zaken als burger-servicenummer of bankgegevens worden ook onder speciale data geschaard.
> Elke vorm van medische data.
> Data over het seksleven of seksuele oriëntatie.

In principe gelden dezelfde rechtmatige redenen, alleen ze worden veel strenger toegepast. Met andere woorden: als je deze data wilt opslaan en verwerken, dan zul je juridisch moeten toetsen of de reden voldoende onderbouwd is. Er is één speciale reden benoemd voor het vastleggen van speciale categorie data en dat is als het hebben van deze data onderdeel is van de natuur van de organisatie. Denk hierbij aan de ledenlijst van een geloofsgemeenschap. Gezien de aard van de organisatie kan het niet anders dan dat speciale data opgeslagen wordt. Overigens ontslaat dat deze organisaties er niet van dat de persoon in kwestie expliciet toestemming geeft dat zijn gegevens worden opgeslagen.



Rechten van de "data subject"

Tot nu toe hebben we gekeken naar geldige redenen om data op te slaan conform de GDPR regels. De wetgeving geeft ook expliciet aan dat de persoon in kwestie waarvan data opgeslagen wordt rechten heeft die nageleefd moeten worden:

> Iemand heeft het recht om geïnformeerd te worden als zijn/haar data ergens gebruikt wordt. Specifiek zijn de volgende zaken benoemd die in zo'n kennisgeving verwerkt moeten zijn:
- Wat is de wettelijke reden waarom data wordt gebruikt (zie rechtmatige redenen eerder in dit document). Indien de wettelijke reden is op basis van toestemming, dan dient ook benoemd te worden dat deze toestemming op elk moment ingetrokken mag worden.
- Wat is de bewaartermijn van de data.
- Benoeming van de rechten onder GDPR die de persoon in kwestie heeft.
- Of de persoon verplicht is de data te verstrekken (dit is bijvoorbeeld het geval bij contracten).
- **Of de data wordt gebruikt voor automatische profilering (bijvoorbeeld machine learning algoritmen).**

> Iemand heeft het recht om op te vragen welke data over hem/haar is opgeslagen.
> Iemand heeft het recht om incorrecte data te laten corrigeren.
> Iemand heeft het recht om te vragen dat data verwijderd wordt (dit is geen absoluut recht, want het kan zijn dat de data niet verwijderd mag worden vanwege wetgeving of omdat sprake is van een contract).
> Iemand heeft het recht om te vragen dat de dataverwerking stopgezet wordt.
> Iemand heeft het recht bezwaar aan te tekenen tegen het gebruik van persoonlijke data.
> Iemand heeft expliciet het recht om bezwaar te maken tegen automatische profilering.

In de laatste versie van GDPR is er ook het recht op overdraagbaarheid van data bijgekomen, maar in de praktijk weet nog niemand goed hoe we hier handen en voeten aan moeten geven omdat er geen standaarden zijn om data van de ene naar de andere organisatie over te dragen. Vanaf het moment dat een burger een verzoek doet op basis van de

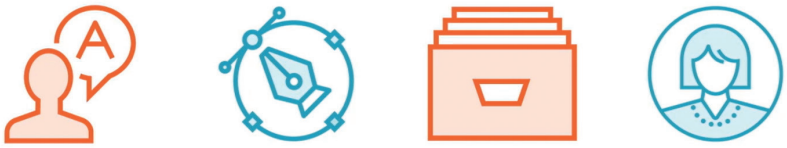


Tell me what you have got (Article 15)

Correct it (if inaccurate) (Article 16)

Delete it (if allowed) (Article 17)

Stop doing that with it (Article 18)



Accountability and Governance (Art. 24)

Data Protection by Design and Default (Art. 25)

Records of Processing Activity (Art. 30)

Cooperate with Regulators (Art. 31)

rechten die hij/zij binnen GDPR heeft, is de organisatie verplicht om binnen 30 dagen te reageren. Dit betekend dat het belangrijk is dat handelingen die nodig zijn geautomatiseerd worden en dat er over nagedacht is in het ontwerp van software.

Plichten voor de data controller / processor

In principe gelden de verplichtingen binnen GDPR vooral voor de data controller, maar in de praktijk zullen een hoop zaken verzorgd worden door de data processor dus deze moet ook voldoen aan de wetgeving. De belangrijkste verplichtingen zijn:

- Je bent verplicht om de verantwoordelijkheid te nemen over de data die je in beheer hebt. In de praktijk betekend dit dat je bestuurlijke organisatie en processen op orde moeten zijn.
- Je bent verplicht om data te beveiligen. Dit gaat breder dan alleen technische beveiliging, het gaat er over dat je kunt aantonen verantwoordelijk met de toegang tot persoonlijke data om te gaan.
- Voor organisaties van meer dan 250 medewerkers ben je verplicht bij te houden wie, wanneer toegang tot data heeft (gehad). Omdat deze eis vaak behoorlijk ingrijpt op software ontwerp, is mijn advies om deze eis altijd mee te nemen in onze ontwerpen, ook als de organisatie nu nog onder de norm van het aantal medewerkers valt.
- Het is een open deur, maar het is verplicht om mee te werken met officiële instanties.

Naast de bovenstaande verplichtingen kan het zijn dat je als organisatie gevraagd wordt een "data protection impact assessment" te doen. Dit is altijd verplicht als je op grote schaal automatische profilering toepast, speciale categorie data verwerkt of beelddata van publiek toegankelijke ruimte verwerkt. Als laatste, kan het zijn dat je een data protection officer in dienst moet hebben. Dit is altijd het geval als je een overheidsinstantie of als je de verplichting tot een data protection impact assessment hebt. Een data protection officer (DPO) kan als rol of functie worden vervuld mits een aantal spelregels in acht worden gehouden:

- De DPO wordt geraadpleegd bij beslissingen rondom persoonlijke data.
- De DPO heeft voldoende mogelijkheden om zijn werk te doen (in tijd en middelen).
- De DPO heeft een beschermde status en kan onafhankelijk werken.
- De DPO heeft kennis van zaken.

Sancties binnen GDPR

Wat organisaties grote zorgen kan baren zijn de sancties op overtredingen van de GDPR. De liegen er niet om met boetes van 20 miljoen tot 4% van de omzet van een organisatie. In de praktijk zijn instanties echter vooral betrokken om organisaties te helpen hun verantwoordelijkheid omtrent data gebruik te nemen. Tegelijk zijn er voorbeelden van boetes. Op de site https://www.timelex.eu/nl/blog/2-jaar-gdpr-een-overzicht-van-handhaving-waar-schuwingen-en-boetes staat een overzicht van een aantal sancties die zijn opgelegd om een beter beeld te krijgen van scenario's waarin boetes zijn opgelegd.

Wat betekend dit nu voor ons als software developers?

Een gedeelte van de GDPR wetgeving zal vooral betrekking hebben op processen, tegelijk zijn er zaken die waar we als ontwikkelaars rekening mee moeten houden bij het bouwen van onze software:

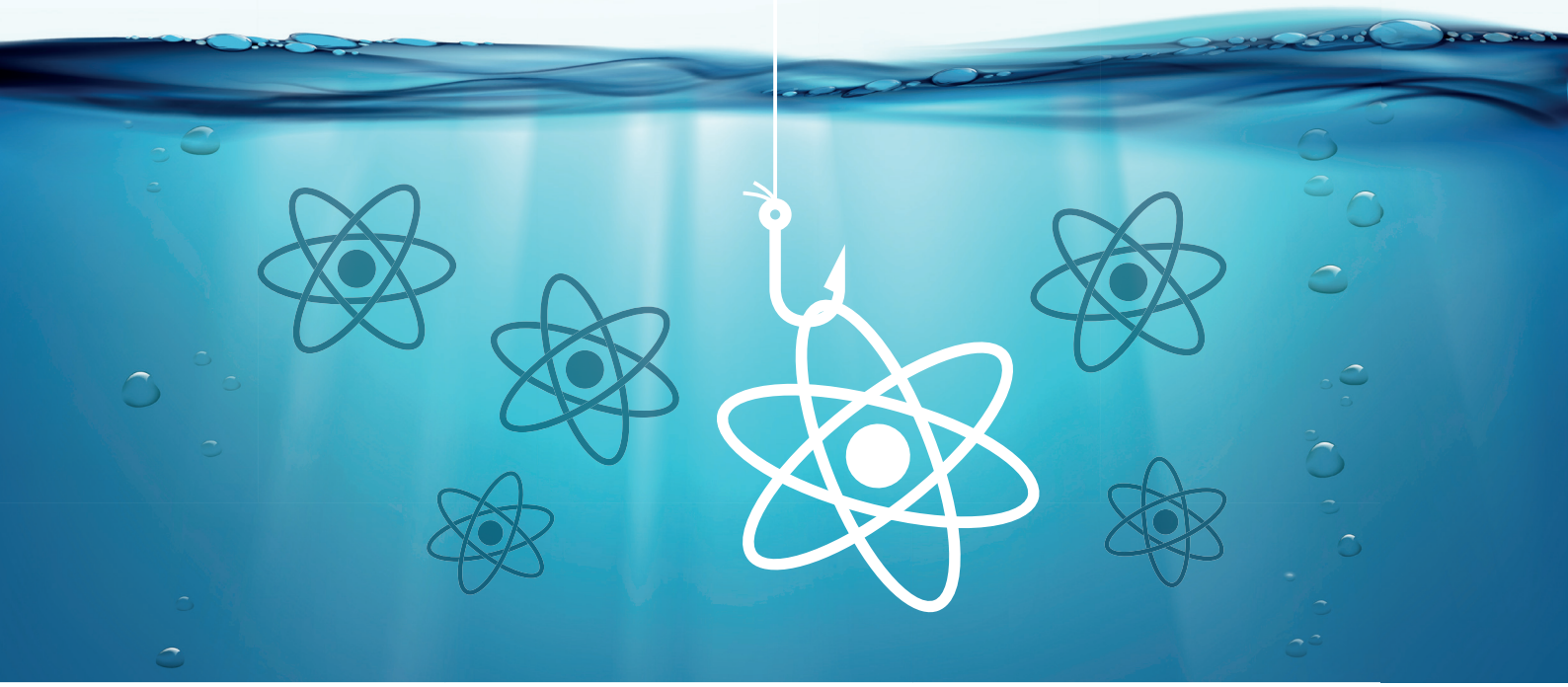
- Zorg dat persoonsdata op te leveren zijn indien een burger een verzoek doet om zijn/haar data te ontvangen (en tegelijk dat de data niet door iedereen zomaar opgevraagd kan worden)
- Sla alleen data op die echt nodig is.
- Zorg dat data verwijderd of geanonimiseerd kan worden en processen die dit ook op tijd doen.
- Zorg voor gedegen autorisatie.
- Monitor en registreer data toegang en wijzigingen.
- Wees bijzonder voorzichtig met allerhande vormen van exporteer en rapportage functies waardoor data uit systemen gehaald kan worden in bulk.
- Versleutel gevoelige data altijd.
- Waarborg dat data correct is, en hersteld kan worden bij calamiteiten.
- Goede beveiliging en regelmatige pentesten zijn een must.
- Voorkom dat data die voor verschillende doelen gebruikt wordt, en/of andere bewaartermijnen heeft, met elkaar verstrengeld raakt.
- Ga voorzichtig om met logging waarin persoonsdata terugkomt.

Bronverwijzing:
Voor dit artikel is gebruik gemaakt van de Pluralsight cursus "GDPR: The Big Picture". Je vind de volledige cursus op Pluralsight: https://app.pluralsight.com/library/courses/gdpr-big-picture/table-of-contents

React Hooks
Een korte introductie

In deze tutorial focussen we ons op het gebruik en begrip van React Hooks. Hiervoor maken we als voorbeeld een simpele productpagina met een shopping cart (zie afbeelding 1). De shopping cart representeert het geheugen (oftewel de 'state') van de productpagina. State verwijst in het algemeen naar applicatie data die moet worden bijgehouden.

Auteur: Peter Eijgermans



Om React hooks beter te doorgronden starten we met het implementeren van de 'useState' hook om de state (concreet: de shopping cart) bij te werken in de applicatie. Tenslotte vervangen we de useState hook door de 'useReducer' hook. De useReducer hook heeft meestal de voorkeur als je complexe state logica hebt. Als uitgangspunt voor de tutorial, **clone** je de volgende Github repository, daar vind je ook de Product_?.js bestanden:

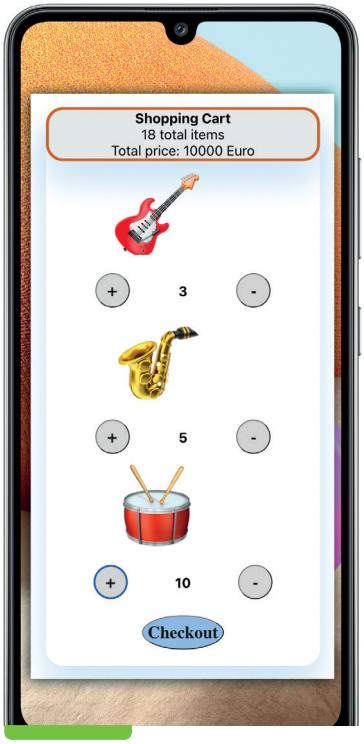
```
git clone https://github.com/petereijgermans11/react-hooks
```

Installeer de dependencies middels: **npm i && npm start** en open de webapp op: http://localhost: 3000

Wat is een component in React?

Voordat we met de implementatie van de productpagina aan de slag gaan, zoom ik even in op wat een component is in React.

Componenten zijn onafhankelijke en herbruikbare stukjes code. Ze hebben hetzelfde doel als JavaScript-functies, maar retourneren HTML. In het geval van React wordt er JSX code geretourneerd. JSX stelt ons in staat om HTML-elementen in JavaScript te schrijven en deze in de DOM te plaatsen zonder de methoden createElement() en/of appendChild() te gebruiken. Zie listing 1 voor een voorbeeld van een functie component die properties (als functie argu-



Afbeelding 1.

ments) ontvangt en JSX retourneert. Zie ook paragraaf: Klasse componenten versus functie componenten voor meer context.
Let op dat om een JavaScript-expressie in JSX in te sluiten, de JavaScript moet worden gewrapped met accolades, zoals `{props.message}`.
Tevens moet de naam van een React component altijd beginnen met een hoofdletter.
Om dit component beschikbaar te maken in de applicatie dien je deze te exporteren (**listing 1**).

Components in Files

Bij React draait alles om het hergebruiken van code, en het is aan te raden om je componenten op te splitsen in aparte bestanden.
Om dat te doen, maak je een nieuw bestand met een **.js**-bestandsextensie en plaats de code voor het component erin. In ons voorbeeld plaatsen we de code in de file Product.js. Deze staat in de folder: *src/Components/Product*. Let op dat de bestandsnaam moet beginnen met een hoofdletter.

Component in een component

Nu heb je een component genaamd **Product**, die een message retourneert. Om dit component in je applicatie te gebruiken, gebruik je een vergelijkbare syntaxis als normale HTML: **<Product />**. Zie **Listing 2** voor een voorbeeld hoe je dit component aanroept met een message-property in een App component.

Klasse componenten versus functie componenten

Er zijn twee soorten componenten in React: klasse componenten en functie componenten. In deze tutorial zullen we ons concentreren op functie componenten, omdat React hooks alleen werken in functie componenten.

Stap 1 - Aanmaken Product component

Na deze korte introductie starten we met het maken van een simpele Product 'functie component' met React (zie **Listing 3**). Het component bestaat uit twee delen:

> de shopping cart, dat het aantal gekozen artikelen en de totale prijs bevat
> en het product, die twee knoppen heeft om het artikel toe te voegen of te verwijderen van de shopping cart.

Voeg de volgende code toe aan de Product.js voor een Product functie component (zie voorbeeld **Product_1.js**).
In deze code heb je JSX gebruikt om de HTML-elementen voor de Product component te maken en te retourneren, met een emoji om het product weer te geven.

Step 2 Implementeer de useState Hook

In dit Product component gaan we twee gegevens bijhouden in de 'state': de 'shopping cart' en de 'totale kosten'. Beiden kunnen in de 'state' worden opgeslagen met behulp van de useState Hook (**Listing 4**).

Wat doet useState?

useState declareert een "state variabele". Onze state variabelen heten *cart* en *total*. Dit is een manier om waarden tussen de functie aanroepen te "bewaren". Normaal gesproken "verdwijnen" variabelen wanneer een functie wordt afgesloten, maar state variabelen worden bewaard door React.

```
function Product(props) {
    return <div> {props.message} </div>
}

export default Product;
```

1

```
import Product from '../Product';

function App()
     { return <Product message="My shopping cart" />
 }

export default App;
```

2

```
export default function Product() {
  return(
    <div>
      <div>Shopping Cart: 0 total items</div>
      <div>Total: 0</div>
      <div><span role="img" aria-label="gitar">🎸</span></div>
         <button>+</button>
         <button>-</button>
    </div>
  )
}
```

3

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

Wat levert useState op?

Het retourneert een array met de volgende twee waarden:

> de huidige state (de variabele 'cart' of 'total' bevatten de huidige state)
> en een functie waarmee je deze state variable kunt bijwerken zoals: 'setCart'.

Wat kan je als argument meegeven aan useState?

Het enige argument dat we door kunnen geven aan useState() is de initiële state. In ons voorbeeld geven we een lege array mee als initiële state voor onze variabele cart. En we geven de waarde 0 mee als initiële state voor onze variabele total.

Wat gebeurt er als de functies setCart of setTotal aangeroepen worden?

Naast het bijwerken van de state variable, zorgen deze functies ervoor dat er een re-render plaatsvindt van dit component, zodra deze aangeroepen worden.

BIO

Peter Eijgermans

Peter Eijgermans is Frontend Developer en Codesmith bij Ordina JsRoots. Hij deelt graag zijn kennis met anderen middels workshops en presentaties.



In **Listing 5** breiden we onze Product component uit met meerdere producten door een product array te definiëren. In de JSX gebruiken we de **.map** methode om over deze producten array te itereren en te tonen (zie **Product_2.js**).
Tevens is er een *add-functie* gedefinieerd om de shopping cart en de totale kosten te kunnen updaten via de Add-button.
In deze add-functie maken we gebruik van de functies *setCart* en *setTotal* die gedefinieerd zijn in de useState hook.
In plaats van het direct doorgeven van het nieuwe product aan de *setCart* en *setTotal* functies, wordt er een anonieme functie doorgegeven die de huidige state aanneemt en een nieuwe bijgewerkte waarde retourneert.
Zorg er echter voor dat je de cart-state niet direct muteert. In plaats daarvan kun je het nieuwe product aan de cart-array toevoegen door de huidige cart-array te **spreaden** (`...current`) en het nieuwe product aan het einde van deze array toe te voegen. Opmerking: door de spread-operator maak je een nieuwe instantie/clone aan van de cart-array en kan je zonder side effects de mutatie doorvoeren.

```
const [cart, setCart] = useState([]);
const [total, setTotal] = useState(0);
```

4

```
import React, { useState } from 'react';

const products = [
   {
       emoji: '\uD83C\uDFB8',
       name: 'gitar',
       price: 500
   }];

export default function Product() {
   const [cart, setCart] = useState([]);
   const [total, setTotal] = useState(0);

   function add(product) {
       setCart(current => [...current, product.name]);
       setTotal(current => current + product.price);
   }
   return(
       <div>
           <div>Shopping Cart: {cart.length} total items</div>
           <div>Total price: {total}</div>
           <div>
               {products.map(product => (
                  <div key={product.name}>
                     <span role="img" aria-label={product.name}>{product.emoji}
                     </span>
                     <button onClick={() => add(product)}>+</button>
                     <button>-</button>
                  </div>
               ))}
           </div>
       </div>
   )
}
```

5

```
import React, { useReducer, useState } from 'react';

function cartReducer(state, product) {
  return [...state, product.name]
}

export default function Product() {
  const [cart, setCart] = useReducer(cartReducer, []);
  const [total, setTotal] = useState(0);

  function add(product) {
    setCart(product);
    setTotal(current => current + product.price);
  }

  return(...)
}
```

6

```
const products = [
   {
       emoji: '\uD83C\uDFB8',
       name: 'gitar',
       price: 500
   }];

function getTotal(cart) {
   return cart.reduce((totalCost, item) => totalCost + item.price, 0);
}

function cartReducer(state, action) {
   switch(action.type) {
       case 'add':
           return [...state, action.product];
       case 'remove':
           const productIndex = state.findIndex(
                 item => item.name === action.product.name);
           if(productIndex < 0) {
               return state;
           }
           const update = [...state];
           update.splice(productIndex, 1)
           return update
       default:
           return state;
   }
}

export default function Product() {
   const [cart, setCart] = useReducer(cartReducer, []);

   function add(product) {
       setCart({ product, type: 'add' });
   }

   function remove(product) {
       setCart({ product, type: 'remove' });
   }

   return(...)
}
```

8

```
function cartReducer(state, action) {
   switch(action.type) {
       case 'add':
           return [...state, action.product];
       case 'remove':
           const productIndex = state.findIndex(
                 item => item.name === action.product.name);
           if(productIndex < 0) {
               return state;
           }
           const update = [...state];
           update.splice(productIndex, 1)
           return update
```

7

Step 3 Implementeer de useReducer Hook

Er is nog een Hook genaamd *'useReducer'* die speciaal is ontworpen om de state bij te werken op een manier die vergelijkbaar is met de *.reduce* array-methode. De useReducer Hook is vergelijkbaar met useState, maar wanneer je deze Hook initialiseert, geef je een functie door die de Hook uitvoert wanneer je de state samen met de initiële gegevens wijzigt. De functie, ook wel de '**reducer**' genoemd, heeft twee argumenten: de state en het product.

Refactor de shoppingcart om de useReducer Hook te gebruiken (**listing 6**). Maak een functie genaamd cartReducer die de state en het product als argumenten gebruikt. Vervang useState door useReducer en geef vervolgens de functie cartReducer door als het eerste argument en een lege array als het tweede argument, wat de initiële state is.

Voer deze wijziging ook door voor setTotal. Het eindresultaat van deze eerste refactoring kan je vinden in component **Product_3.js**.

Nu is het tijd om de 'remove' functie toe te voegen. Om dit te implementeren maken we gebruik van een veel voorkomend patroon in reducer-functies om een *action-object* door te geven als tweede argument. Dit action-object bestaat uit het product en het *action-type* (**listing 7**).

Binnen de cartReducer wordt het totaal bijgewerkt op basis van het action-type. In dit geval voeg je producten toe aan de shoppingcart bij action-type: add. En verwijdert je ze bij action-type: *remove*. De remove actie werkt de state bij door 'splicing out' de eerste instantie van het gevonden product uit de cart-array. Middels de spread-operator maken we een copy van de bestaande state/cart-array, zodat we geen last zullen hebben van side effects tijdens het updaten. Vergeet niet om de final state te retourneren aan het eind van iedere actie.

Na het bijwerken van de cartReducer, maken we een remove-functie aan die de setCart aanroept met een action-object (**listing 8**). Dit action-object bevat het product en het action-type: *remove*.

Pas tevens de add functie aan die de setCart aanroept met een action-type: add. En verwijder de bestaande aanroep naar setTotal uit de add functie.

Maak tenslotte een getTotal functie aan die de totaalprijs berekend op basis van de totale cart-state.

Hier kan je gebruik maken van de 'cart.*reduce()* functie' (**listing 8** en **Product_5.js**).

Tenslotte

Er zijn zeker andere manieren om useState en useReducer toe te passen. useState en useReducer worden niet aanbevolen voor het beheren van de state in zeer grote, complexe projecten, houd daar rekening mee.

Naast de hier behandelde hooks biedt React vele andere hooks, evenals de functionaliteit om je eigen custom Hooks te maken. Voorbeelden van andere hooks zijn:

- useEffect, maakt het mogelijk om side effects af te handelen, zoals fetching data. In component Product_7.js is een useEffect geïmplementeerd om de productgegevens op te halen.
- en useContext, is een manier om de state globaal te onderhouden. ●



> Denk jij dat je het beter weet?

/ Mooi, dan zoeken we jou!

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

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

Ontwikkel Multicloud Ready Microservices met DAPR

We weten allemaal hoe snel je een cloud native webapplicatie kunt schrijven en deployen naar elke public cloud. Met een paar klikken heb je zo een leeg .NET Core project gemaakt, dependency injectie geïmplementeerd, een aantal APIs ontsloten en gedeployed naar je favoriete compute service. Maar wat als je over de Microservice route wilt gaan?

Auteur: David Hoop

Wel dan moet je eerst je Kubernetes cluster opzetten en je Service Bus of Message queue configuratie voor de service-to-service communicatie en dan heb je nog geen regel code geschreven. Heb je wel eens nagedacht over de afhankelijkheid van je software architectuur op je cloud infrastructuur? In de meeste gevallen is je container setup specifiek voor de cloud provider die je op dit moment gebruikt en hebben je microservices een tight coupling met cloud provider specifieke endpoints of met andere PaaS-diensten van deze cloud provider. Wat als je dan wilt switchen naar een andere cloud provider? Wat als je een nieuwe service nodig hebt die enkel wordt aangeboden door de concurrent? Dan heb je een software architectuur nodig welke klaar is voor multicloud!

Het multicloud ready maken van jouw architectuur kan nu vereenvoudigd worden door DAPR te gebruiken. Een middleware laag en applicatie runtime welke is ontworpen om jouw microserviceontwikkeling en deployment cloud agnostisch te maken. Zonder tight coupling tussen je applicatie code en jouw cloud infrastructuur.

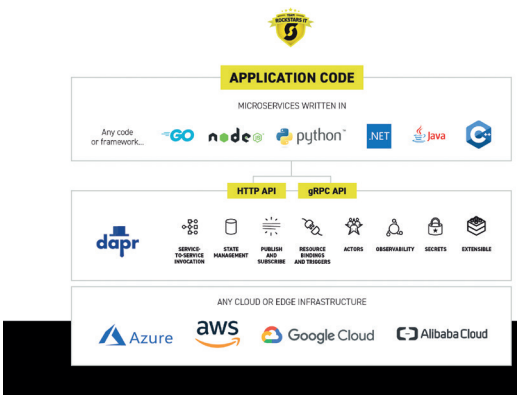
Geschiedenis van DAPR

Als gezegd DAPR is een middleware laag en applicatie runtime, daarom staat de afkorting ook voor "Distributed Application Runtime". Distributed betekent in deze dat DAPR decentraal draait en perfect past binnen een microservices architectuur. De key functionaliteit van DAPR is om ondersteuning te bieden voor cloud native en serverless cloud diensten in een microservices architectuur.

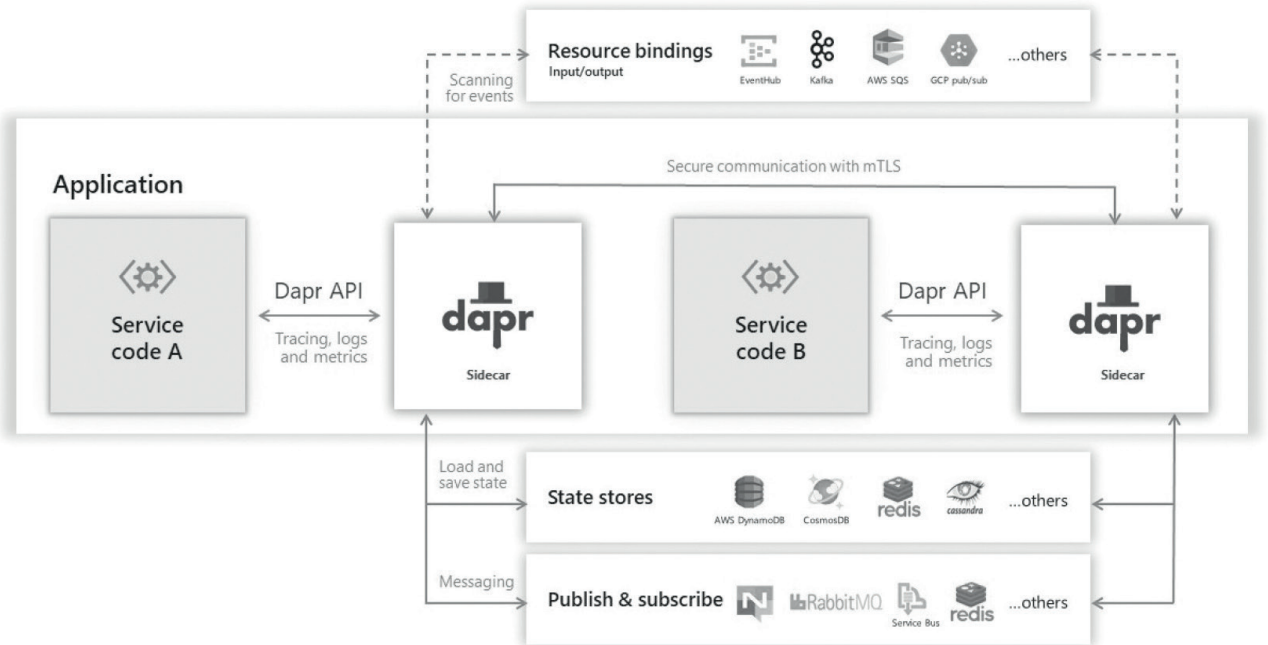
Au contraire wat de meeste mensen denken is DAPR niet ontwikkeld of onderhouden door Microsoft. Het is een volledig open-source project gelanceerd onder de MIT-licentie. Microsoft heeft echter wel redelijk wat publiciteit vergaard met DAPR en draagt ook financieel bij aan het project. Verder worden er ook bijdragen geleverd aan het project door developers van Microsoft en developers die in het verleden voor Microsoft hebben gewerkt. Deze inmenging van Microsoft heeft ertoe geleid dat er binnen het .NET ecosysteem er goede ondersteuning is voor DAPR vanaf eigenlijk begin af aan. Er is een specifieke SDK voor .NET Core, volledige support in Visual Studio en goede documentatie volgens de Microsoft standaarden. Maar DAPR is niet enkel Microsoft specifiek. De runtime zelf is geschreven in Go en is platform agnostisch. Dit geldt voor zowel de infrastructuur laag waar DAPR mee praat als de applicatie laag die je bovenop DAPR draait. Naast .NET Core is er ook ondersteuning voor Node.js, Java, Pyhton, C++ en Go zelf. Op de infrastructuur laag is er out-of-the-box support voor natuurlijk Microsoft Azure maar ook voor AWS, Google Cloud Platform en zelfs nieuwkomers in de markt zoals Alibaba Cloud en Camunda Cloud.

DAPR Bouwstenen

DAPR acteert als een interface tussen je applicatie en je container infrastructuur. Het gebruik van containers



SIDECAR AND COMPONENT ARCHITECTURE



is overigens verplicht als je DAPR wil gebruiken. De onderliggende orchestrator staat vrij. Dit kan zowel Kubernetes zijn maar ook Docker Swarm en de cloud provider die de infrastructuur levert is ook niet van belang.
Het belangrijkste voordeel van DAPR uit oogpunt van de developer is dat het zorgt voor een single-contract interface om the interacteren met elke cloud infrastructuur, elke andere microservice in je cluster en zelfs interacties met externe applicaties of API's op het internet, buiten je cluster. Je hoeft niet meer bang te zijn voor breaking changes in je infrastructuur laag. Zolang je jouw contract met de DAPR runtime respecteert is er niets aan de hand.
De DAPR runtine is modulair ontwikkeld. Dit betekent dat je zelf kunt kiezen welke services in jouw software life cycle door DAPR geregeld moeten worden en welke je zelf regelt. Op deze manier blijft DAPR lichtgewicht en heeft het geen impact op de snelheid van jouw microservices. Ook neemt DAPR zo minder resources in binnen je Kubernetes cluster.
Er zijn onder andere DAPR modules voor: State Management, Service-to-Service Invocation, Pub-Sub, Resource Bindings, Observables en Secrets. Vanwege de open-source licentie van DAPR kun je ook een van de velen modules gebruiken die ontwikkeld zijn en onderhouden worden door de community.

Sidecar Architectuur

DAPR gebruikt het sidecar pattern om zichzelf beschikbaar te maken voor jouw microservice. Dit is hetzelfde pattern dat Service Meshes zoals Istio gebruiken binnen het Java Ecosysteem. Alhoewel DAPR is geen Service Mesh omdat DAPR geen diensten of features aanbiedt op de netwerk laag, het is puur en alleen een applicatie runtime.
Wanneer je een DAPR resource injecteert in je microservice wordt er een sidecar opgespint in een aparte pod of container in jouw cluster. Deze pod wordt van verbonden met de pod welke je microservice draait op eenzelfde manier als meerdere karretjes samenwerken in een trein van een achtbaan. Afhankelijk van de hoeveelheid DAPR diensten die je gebruikt in je microservices kunnen er één of meerdere karretjes aangehaakt worden bij je service pod. De karretjes onderling communiceren over mTLS.

Black-box Interface

Het grote voordeel van DAPR is de enkele interface die je gebruikt vanuit het perspectief van je microservicecode. Je kunt met DAPR communiceren over HTTP(S) of gPRC

```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: demosecrets
  namespace: default
spec:
  type: secretstores.local.file
  metadata:
    - name: secretsFile
      value: secrets.json
```

Code Sample 1. Voorbeeld van een secret component die een secret leest uit het local filesystem.

```
public static IHostBuilder CreateHostBuilder(string[] args)
{
    // Create DAPR Client
    var client = new DaprClientBuilder().Build();

    return Host.CreateDefaultBuilder(args)
        .ConfigureServices((services) =>
        {
            // Add the DAPR Client to Host.
            services.AddSingleton<DaprClient>(client);
        })
        .ConfigureAppConfiguration((configBuilder) =>
        {
            // Add the secret store Configuration Provider to the configuration builder.
            configBuilder.AddDaprSecretStore("demosecrets", client);
        })
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        });
}
```

Code Sample 2. DAPR Client toevoegen aan je ASP.NET Core webapplicatie.

```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: demosecrets
  namespace: default
spec:
  type: secretstores.azure.keyvault
  version: v1
  metadata:
  - name: vaultName
    value: david07-e-kv
  - name: spnClientId
    value: b49cdb50-a668-4c57-913d-21b45bf89e4d
```

Code Sample 3. DAPR-component wijzingen naar Azure Key Vault met MSI-authenticatie.

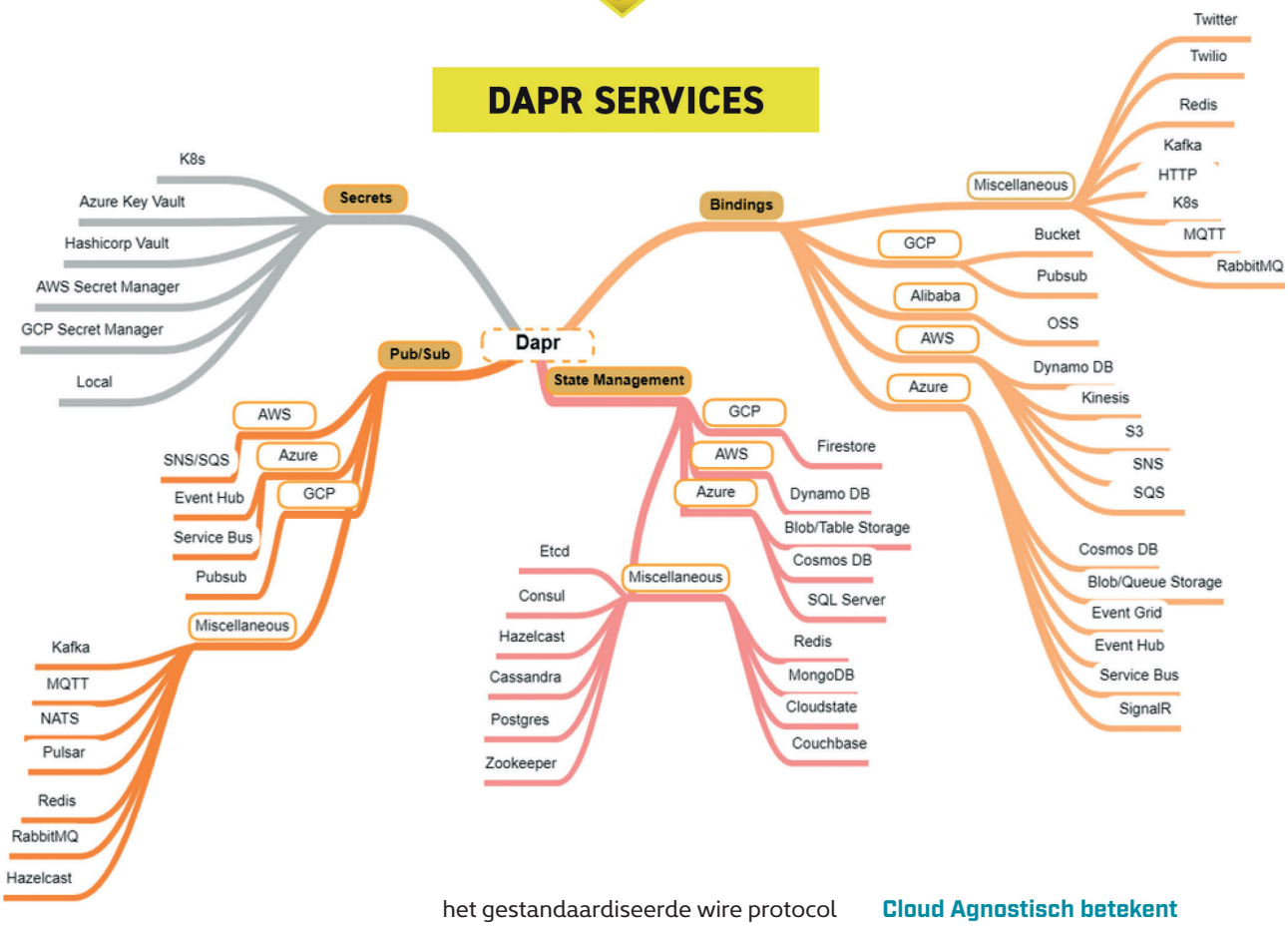
afhankelijk van jouw voorkeur en de stack die je gebruikt. Elke DAPR service die je gebruikt wordt gedefinieerd als een component in je source code. Componenten zijn beschreven in YAML en kunnen gebruikt worden om te interacteren met elk van de DAPR modules. Voor elke module is er een lijst met ondersteunde componenten van elke cloud provider.

Voorbeeld 1

Het DAPR component dat hierboven staat bevat relevante metadata over het type component en de infrastructuur achter het component. In dit geval hebben we het component demosecrets van het type secretstores dat leest uit een lokaal bestand dus een bestandsnaam en pad zijn verplicht. **Voorbeeld 2**

De DAPR client wordt geïnjecteerd als een singleton en daarna voegen we een SecretStore toe aan de client. De DAPR client weet alleen dat hij praat tegen de demosecrets secret store en weet niets over de fysieke locatie van de store of de specifieke implementatie. De client weet alleen dat het secrets kan ophalen van dit specifieke DARP endpoint wat in dit geval http://localhost:<daprPort>/v1.0/secrets/demosecrets/ zou zijn. Dit betekent dat je de inhoud van het DAPR component op elk moment kunt veranderen zonder het contract met de interface van je applicatie te breken. Dus stel ik wil mijn secrets verplaatsen van een lokaal bestand naar zeg Azure Key Vault, dan kan ik dat doen door enkel mijn DAPR component te wijzigen. **Voorbeeld 3**

Mijn applicatie en DAPR-client praten nog steeds tegen de demosecrets secret store. De client weet niet dat de secrets niet langer in het lokale bestandsysteem leven en verplaatst zijn naar Azure Key Vault. Mijn applicatie weet ook niet dat hij nu secrets moet ophalen over het publieke internet en zich eerst tegen Azure moet authentiseren, de DAPR runtime regelt dit allemaal voor mijn microservice.



Dit betekent dat je resources kunt verplaatsen of kunt wisselen tussen publieke cloud providers zonder dat je applicatie hiervan weet en zonder dat je de configuratie van je applicatie hoeft aan te passen of code changes hoeft door te voeren in je microservice.

Componenten voor elke Cloud

DAPR heeft een heleboel off-the-shelf componenten voor alle ondersteunden cloud platformen. Vorr een beter inzicht in deze componenten zie de graph hieronder welke de verschillende componenten van elk cloud platform laat zien. Sommige componenten kunnen gebruikt worden op elk cloud platform zolang je maar het gestandaardiseerde wire protocol gebruikt voor deze componenten. Natuurlijk kun je gebruik maken van Azure native services zoals Cosmos DB, SignalR of Event Grid maar er is ook support voor vergelijkbare diensten in AWS en GCP. Ook kun je kiezen voor diensten die beschikbaar zijn op elke cloud zoals Cassandra, Kafka of RabbitMQ. Mocht het component dat jij wilt gebruiken toch nog niet bestaan dan kun je deze zelf maken en makkelijk toevoegen aan de DAPR Components Contrib.

Cloud Agnostisch betekent Multicloud Ready

Omdat DAPR zelf cloud agnostisch is, i.e. voor elke module is er een component voor bijna alle Publieke Clouds (Azure, AWS, GCP, Alibaba Cloud, Camunda Cloud) en het ervoor zorgt dat jouw microservice niet weet tegen welke cloud hij praat is DAPR ideaal om te gebruiken in een multicloud omgeving of zelfs een must in een multicloud architectuur. Met DAPR kun je jouw infrastructuur wijzigen onafhankelijk

Maar wat als je over de Microservice route wilt gaan?

van je applicatie code en makkelijker switchen naar een andere cloud provider of verschillende cloud providers door elkaar heen gebruiken zonder dat jouw microservice dit hoeft te weten. Verder kan je zelfs een hybride cloud oplossing maken met resources die ook nog steeds on-premises leven.
Voor puur op PaaS gebaseerde applicaties draaiend op Azure App Service of AWS Lambda was dit al tijden mogelijk met de SDKs die er beschikbaar zijn voor deze platformen binnen .NET Core. Maar voor microservices architecturen op de .NET Stack vult DAPR een gat om multicloud ook makkelijk te maken voor deze architectuur. Voor andere talen zoals Java kan DAPR ook een goede keus zijn maar binnen het Java Ecosysteem bestaan ook al andere bewezen oplossingen om service-code te scheiden van de cloud infrastructuur met Service Meshes zoals Istio maar ook met nieuwe ontwikkelingen binnen het Spring framework met Spring Cloud Skipper en ook de Spring Cloud native implementaties op Azre, AWS, GCP en Alibaba cloud zorgen ervoor dat er veel meer concurrentie is voor DAPR binnen de Java stack. Ik voorzie dus een grotere shift in interesse en ondersteuning voor DAPR op het Microsoft platform en de .NET Core community zo dat DAPR de go to middleware layer wordt om op .NET Core gebaseerde microservices te draaien op elke cloud.

Waarom Multicloud belangrijk is

Het public cloud landschap is veranderlijk en als afnemer wil je geen last hebben van vendor lock-in. Op IT-management niveau committeren aan één enkele leverancier voor cloud infrastructuur kan best stressvol zijn. Niet alleen verlies je flexibiliteit maar je moet ook een commitment op de lange termijn aangaan met een specifieke leverancier, helemaal als je software architectuur slechts rekening houdt met één enkel cloud platform. Op deze manier heb je een harde afhankelijkheid op deze leverancier, misschien ben je wel minder blij met de support die deze leverancier biedt, misschien is er wel een service die je wilt gebruiken die deze leverancier nog niet heeft of misschien verhoogd deze leverancier simpelweg haar prijzen omdat je er toch al aan vast zit. In deze gevallen wil je de mogelijkheid hebben om (deels) te veranderen van cloud provider zonder je business continuïteit van je applicaties op het spel te zetten. Het hebben van een multicloud architectuur maakt dit mogelijk omdat je jouw applicatie ontkoppeld van de cloud infrastructuur. Dit zorgt ervoor dat je niet langer overgeleverd bent aan de grillen van die éne cloud provider waarmee je in zee bent gegaan en maakt het mogelijk om nieuwe diensten van andere leveranciers uit te proberen. Het zorgt er ook voor dat jouw CTO niet langer slapeloze nachten krijgt over dat megacontract met die éne cloud provider welke anders een single point of failure zou worden binnen je IT-strategie. Als je overweegt om te migreren richting public cloud, dan moet je ook overwegen om je software architectuur multicloud ready te maken. Doe dit voor het te laat is en je vastzit in de gouden kooi van de cloud provider waarmee je niet meer mee kunt leven, maar ook niet langer zonder kan. ●

BIO

David Hoop

David de Hoop is Special Agent (thought leader) bij Team Rockstars IT. Als solution architect adviseert hij organisaties op het gebied van Public Cloud, DevOps transformatie en Microsoft Azure. David is een fervent Agile aanhanger en DevOps evangelist. Daarnaast haalt hij veel energie uit het geven van workshops en spreken op conferenties, het liefst fysiek op een podium al is dit de laatste twee jaar wat lastig.




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

Later als ik groot ben ...

"Wat wil je later worden als je groot bent? Later wil ik CTO worden papa!" Ik heb het nog nooit een kind horen zeggen. Prinses, brandweer, mama, topvoetballer, zangeres en piloot zijn beroepen die op zo'n moment vaak genoemd worden.

Auteur: Koen Aerts

Als onderdeel van de digitalisering hebben veel bedrijven een CTO-functie ingericht. Maar wat is een CTO nu eigenlijk? Wat zijn je werkzaamheden en verantwoordelijkheden? Hoe word je überhaupt een CTO en wat zijn dan belangrijke stappen om te nemen?
Navraag leert dat het bij software developers een populaire carrière stap is. Dit artikel vertelt meer over de functie CTO, de weg naar CTO en mijn persoonlijke ervaringen.

Wat doet een CTO?

Volgens Wikipedia is de CTO (Chief Technology Officer) "the highest technology executive position within a company and leads the technology or engineering department". Dit betekent dus leidinggeven aan de technische afdeling en de technische visie voor het bedrijf bepalen. Dit zijn belangrijke verantwoordelijkheden!
Uiteraard bepaalt de grootte van de technische afdeling de zwaarte van de functie. De ene keer stuur je zelf het technische team aan en de andere keer zit er nog een management laag (bijvoorbeeld IT managers) tussen. Bij andere bedrijven is het weer zo ingericht dat je vanuit het management team indirect leiding geeft.
Met deze definitie betekent een CTO binnen het ene bedrijf iets anders dan bij het andere bedrijf. De functie heeft altijd als verantwoordelijkheid de (technische) toekomst van het bedrijf richting te geven. Zelf ben ik CTO bij een consultancybedrijf. Het is mijn verantwoordelijkheid om onze software developers te begeleiden richting toekomst (bijvoorbeeld zorgen dat de juiste carrièremogelijkheden geboden worden) en zorg dragen dat wij de expertise hebben om onze klanten verder te helpen. Klanten kunnen van ons (van mij en van onze experts) het juiste IT advies krijgen.
Met name dit laatste zorgt ervoor dat je kennis moet hebben van veel verschillende landschappen. Je krijgt de mogelijkheid om in heel veel verschillende keukens te kijken en ziet veel best practices (overigens ook worst practices) op het gebied van techniek en proces.
Het gaat van Microsoft naar Java, van AI met Python naar mobile met Flutter. Het is, voor mij in ieder geval, niet te doen om op al deze gebieden expert te zijn. Samenwerking met experts is dan ook een erg belangrijk onderdeel van deze functie.
Hoe groei je in zo'n functie? Is hier een vast carrière pad voor?

Carrière pad CTO

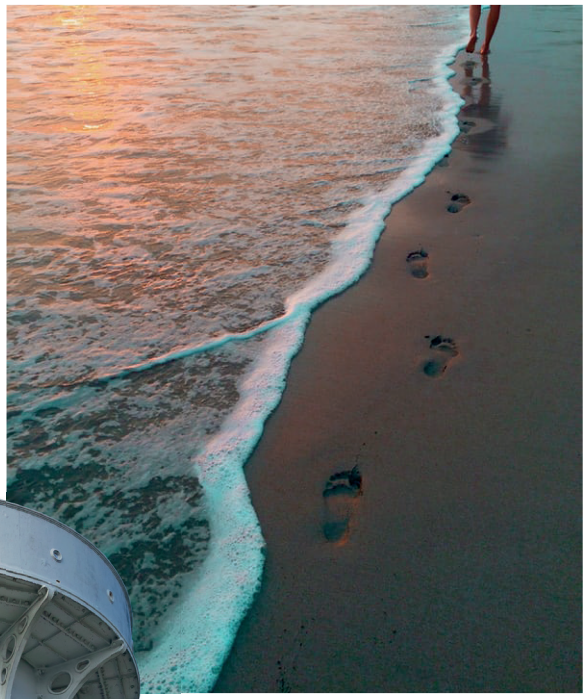
Er bestaat geen opleiding tot CTO, hoe zou je dan richting CTO kunnen groeien? Naar mijn mening zijn twee pijlers belangrijk bij de functie van CTO. Voldoende technische IT-skills en managementervaring.
IT-skills is een enorm breed begrip. De benodigde technische bagage zal bij het ene bedrijf dan ook anders zijn dan bij het andere bedrijf. Hebben ze veel maatwerk software of werken ze met standaard pakketten? Hebben ze eigen servers of draait alles in de Cloud? Zijn de werkplekken in eigen beheer of zijn er productielijnen in de fabriek met veel IT? Uiteraard zullen er ook veel thema's overeenkomen zoals Security, Cloud, Data, AI en DevOps.
De managementervaring is benodigd omdat je ook verantwoordelijk bent voor de technische leiding van het bedrijf. Het is belangrijk om als teamlead of (afdelings)manager al managementervaring opgedaan te hebben.
De werkzaamheden zullen zowel op strategisch, tactisch en operationeel

Prinses, brandweer, mama, topvoetballer, zangeres en piloot zijn beroepen die op zo'n moment vaak genoemd worden.

niveau zijn. Een visie neerzetten, een meerjaren planning maken en beslissingen nemen waar de bedrijfsvoering naartoe gaat zijn voorbeelden van strategische werkzaamheden. Tactisch werk zou portfolio ontwikkeling en het uitbouwen van klantrelaties kunnen zijn. Uiteraard heb je ook veel te maken met operationele werkzaamheden. Gesprekken met collega's, klanten, sollicitatiegesprekken, marketing uitingen en nog veel meer beheersen je agenda.
Soft skills, als onderdeel van managementervaring, zijn enorm belangrijk aangezien de werkzaamheden enorm uiteenlopend zijn. Je communiceert met de techneuten van het bedrijf en adviseert de raad van bestuur. Een hele diverse functie die dan ook een heel divers profiel vraagt.

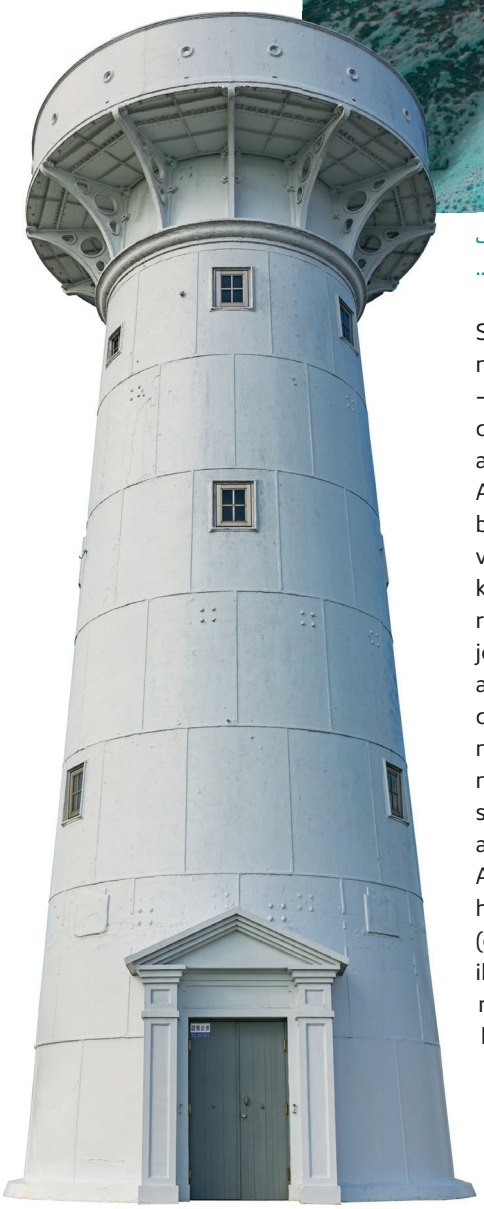
Mijn carrière

Achteraf vind ik het gemakkelijk om bovenstaande (carrière pad CTO) op te schrijven omdat het enorm voor de hand ligt. Toch denk ik dat managementervaring opdoen en daarmee een stuk techniek loslaten een moeilijke stap is.
Achteraf vind ik het gemakkelijk om bovenstaande (carrière pad CTO) op te schrijven omdat het enorm voor de hand ligt. Toch denk ik dat managementervaring opdoen en daarmee een stuk techniek loslaten een moeilijke stap is.
Naar mijn mening moet je technisch een stap terug doen op het moment dat je een managementfunctie krijgt om een tweetal redenen. Enerzijds is management een vak apart en vereist het veel aandacht om je hierin te bekwamen. Anderzijds is het voor de mensen die je aanstuurt fijn als zij mee mogen beslissen over de technische keuzes die gemaakt worden. Niemand wil vanuit een ivoren toren te horen krijgen welke keuzes er gemaakt zijn. De beste keuzes worden samen als één team gemaakt! Belangrijk is het dan ook om de organisatie goed over de technische as ingericht te hebben. Op deze manier kun je samen efficiënt sturing geven aan de organisatie!
In 2000 ben ik gestart als software developer. In 2008 zat ik bij een architectenclub. Om diverse redenen (zie artikel loopbaanstress -



Je kent het pad pas ...
... als je het gelopen hebt

Ivoren toren



SDN Magazine #141 - https://sdn.nl/nieuws/sdn-magazine-2021-141/ - - wat hier verder op ingaat) heb ik destijds gekozen voor een toekomst als manager.
Als je die beslissing neemt heeft dat behoorlijk wat impact. Ik merkte dat vanaf dat moment mijn technische kennis razendsnel aan het verouderen was. Je komt snel op het punt dat jouw team met software developers andere technische keuzes maken dan dat jezelf zou maken. Dat was het moment dat ik realiseerde dat ik me moet laten adviseren door technische experts. Een hele gekke ervaring als je dat kortgeleden nog zelf was.
Alle jaren heb ik kennisdeling als heel waardevol bestempeld. Als XPG (eXPertise Groep) leider organiseerde ik veel sessies en stond ook regelmatig zelf op de zeepkist. Naast de kennis en ervaring die je hier opdoet bouw je ook een netwerk op waar ik tot op de dag van vandaag veel aan heb. Ook in mijn huidige rol als CTO hecht ik nog steeds heel veel waarde aan kennisdeling.

BIO

Koen Aerts

Koen Aerts is CTO bij Team Rockstars IT. Na diverse jaren als software ontwikkelaar en architect gewerkt te hebben heeft hij de stap naar management functies gemaakt. Koen is erg innovatief en creatief en is actief in veel software communities. Daarnaast is hij voorzitter van de Devoxx4Kids stichting.



Vervolgens ben ik tien jaar lang als manager werkzaam geweest in verschillende rollen. In 2020 ben ik door Team Rockstars IT, tevens mijn huidige werkgever, gevraagd of ik daar CTO wilde worden. Toen die vraag gesteld werd zag ik dat ik de perfecte ervaring had voor een dergelijke rol. De recruiter vroeg aan mij of ik wel alles van Cloud en Microservices af wist. Ik gaf aan dat dit met tien jaar managementervaring niet het geval is, maar dat ik dat ook niet als probleem zag. Ik zou juist alle kennis van de organisatie kanaliseren om daar de juiste beslissingen mee te kunnen nemen. Aangezien ik daar vorig jaar mocht beginnen zal het wel het juiste antwoord geweest zijn 😉

De wereld van IT is enorm groot

Toegeven dat je niet alles weet vind ik een kracht! Soms is het moeilijk, zeker als je het gevoel hebt dat jij dit juist zou moeten weten vanuit je functie. Toch is het belangrijk om hulp in te roepen op het moment dat je zelf niet tot het beste resultaat kunt komen. De wereld van IT is te groot om alles zelf te weten.
Wat mij betreft is het de taak van de CTO om zich door de juiste experts te laten adviseren en dan een weloverwogen besluit te nemen. Om een weloverwogen besluit te kunnen nemen zijn al veel IT-skills nodig.

Conclusie

Hopelijk heb je door dit artikel meer inzicht gekregen in de functie van CTO. Zoals al eerder gezegd een hele diverse functie. Mocht je een toekomst als CTO ambiëren, denk dan vooral ook na over het management gedeelte ervan. Het is goed om deze management ervaring op te gaan doen.
Wellicht zal je de techniek voor een deel los gaan laten waardoor je je team nodig hebt om tot de juiste inzichten en besluiten te komen. Wat zou je ervan vinden als je de techniek los zou moeten laten?
Wat mij aanspreekt is de samenwerking en de impact die je bij je eigen organisatie en ook bij andere organisaties (vaak klanten) kunt maken. Het geeft mij veel voldoening om andere mensen te kunnen helpen met de brede kennis die ik bezit!
Succes met al je overwegingen en je technische, management of gecombineerde carrière 😉

Brede kennis leidt tot innovatie.



Mocht je een toekomst als CTO ambiëren, denk dan vooral ook na over het management gedeelte ervan.

Nadine Wolff

Het Ijs Breken

Op de eerste dag van mijn eerste stage van de (technische) Informatica opleiding werd ik voorgesteld aan collega's en kreeg ik mijn plek aangewezen in de kamer. Ik kan mij nog herinneren dat een van de collega's grapte dat ze nu geen vrouwonvriendelijke grapjes konden maken. Ik lachte een beetje schaapachtig en ging op mijn plek zitten. Ik wist niet hoe ik hierop moest reageren, het was voor mij de eerste keer in zo'n omgeving.
Zodra ik op de middelbare school voor de bèta-richting koos, heb ik alleen maar tussen de jongens gezeten. En die jongens gingen toch op een hele andere manier met elkaar, en met mij om. Laat ik het zo verwoorden: ik heb geleerd om mannen (jongens) humor te waarderen. Dus dat er ineens hoffelijk rekening met mijn vrouw-zijn werd gehouden, was nieuw voor me. Die eerste keer was trouwens niet de laatste keer dat er stilte viel zodra ik, vrouw, op kantoor plaats nam tussen volwassen mannen. Ik heb het altijd als een lief gebaar opgevat ook al voel je toch het ongemak. En ondanks dat ik net zoals veel techneuten in bepaalde mate moeite hebben met sociale interactie, heb ik mezelf aangeleerd om het ijs te breken zodat iedereen zich in de ruimte weer comfortabel voelt.
Hoe dan?! Nou, in eerste instantie heb ik de focus gelegd op verbinding. Gesprekstof vinden en dan langzaam de gezelligheid weer introduceren. En met gesprekstof bedoel ik niet over het weer buiten of over het koffieautomaat praten. Maar over gezamenlijke interesses waarbij het geslacht niet uitmaakt zoals... techniek. We zijn allemaal techneuten, vinden het leuk om software te ontwikkelen en hebben liefde voor het vak/techniek.
Nieuwe collega's tijdens de pauze uithoren over hun thuissituatie is een persoonlijke favoriet van mij geworden. Waar woon je? Partner? Kinderen? Hobby's of andere bezigheden tijdens het weekend? Of als je het wat zakelijker wilt houden: werk je al lang hier? Works every time, iedereen praat graag over zichzelf. Vooral als je ze vraagt wat ze leuk vinden om te doen, kan je zelfs de stilste persoon aan het praten krijgen. En zodra je iets hoort dat op toepassing is van jezelf, kan je mooi aanhaken. Hoppa, verbinding en weg ongemak.
Makkelijk toch? Elke nerd kan dit leren. Op een gegeven moment werd ik er zo goed in dat ik van mijn leidinggevende terug kreeg dat ik wel erg 'amicaal' was op kantoor. (Ik heb ook moeten opzoeken wat amicaal betekent). Ach, dat is toch ook met kleine kinderen die een nieuw trucje geleerd hebben? Op een gegeven moment doen ze niets anders meer dan alleen dat ene trucje tot vervelens toe. Alles draait om balans. Zodra ik wat zekerder van mijn zaak werd, ben ik ook humor gaan inzetten. Iets wat voor mijn gevoel toch wat meer bestemd is voor de personen die al een stap hoger zitten op de social skills ladder. Het was in het begin best moeilijk voor mij om in te schatten welke grapjes je kan toepassen bij iemand die je nog niet kent. Iets wat ik met vallen en opstaan heb geleerd. Het grapje "loop je (weer) achter de vrouwtjes aan?" bij een collega die achter je loopt in de wandelgangen wordt niet altijd door iedereen even goed ontvangen. Toch vind ik het proberen waard want humor is een geweldig smeermiddel en ook een geweldige ijsbreker. Het werkt in ieder geval een stuk sneller dan zoeken naar een onderwerp om op in te haken.
Ik daag mijzelf graag uit met mijn grapjes niemand te beledigen. Daarmee hoop ik te laten zien dat je ook lol kan hebben zodra er een vrouw bij het gezelschap komt. Woordgrapjes is daar een mooi voorbeeld van, al is het ook een smaak dingetje. Kwestie van uitproberen. Persoonlijk vind ik het heerlijk om te grappen tussen de mannen vanuit mijn vrouwelijk oogpunt. Als er een (mannelijke) collega een homofobische opmerking of grap maakt, reageer ik enthousiast dat ik een goeie ervaring met mannen heb en zeker aan te raden is!



Ik gebruik dus ook humor om op een luchtige manier mijn grenzen aan te geven. Of rustig en vooral vriendelijk zeggen dat je er niet om kan lachen. Dat werkt stukken beter dan een sneer teruggeven. Geloof me, ik heb het geprobeerd en de situatie werd er niet beter op. Want het werd alleen maar een soort van wedstrijd waarbij de ene vervelende opmerking de andere opvolgde terwijl de rest van de groep mij ontzettend kattig begon te vinden. Als je een gevecht aangaat, vecht dat met je eigen wapens in plaats van die van je tegenstander te kopiëren.

Als ik nu in dezelfde situatie terecht zou komen als bij die eerste dag op mijn eerste stage dan zou ik met een grote glimlach aan ze vragen of dit betekent dat ik dan ook geen man-onvriendelijke grapjes mag maken? ●


FUTURE
TECH 2022
THE IT CONFERENCE FOR
MICROSOFT TECHNOLOGIES
21 juni 2022, Jaarbeurs Utrecht
Tickets zijn nu verkrijgbaar
via www.futuretech.nl of scan de QR code!
Meer info op www.futuretech.nl



Ben je al geabonneerd op het YouTube kanaal van de SDN? De SDN Cast is een tweewekelijkse (dinsdagavond om 20:00) video/online kennissessie over development of IT gerelateerde onderwerpen. Zo komen diep technische thema's aan bod zoals C#, .NET, Chaos engineering en Cloud/Azure. Of juist meer business en proces gerichte zaken waarin je bijvoorbeeld leert hoe je beter kan samenwerken. In de SDN Casts voeren de gastheren (Marcel Meijer en Roy Janssen) gesprekken met IT-specialisten/Developers/Architecten etc.

Daarvoor zijn we altijd op zoek naar gesprekpartners met een mooi en goed verhaal. Dat hoeven geen nieuwe onderwerpen te zijn. Een gesprek waarin je ervaringen deelt van een oplossing waar jij of je team(genoten) mee geworsteld hebben, zijn ook voor het grote publiek interessant. Lijkt het je wat om mee te doen? Dan hoor ik graag welke avonden je kunt (mag ook in het weekend), dan proberen we daar een mouw aan te passen. Kijk anders op ons YouTube kanaal: https://www.youtube.com/sdncast of www.sdncast.nl.



Duco Fronik

Een sprong in het diepe

Een aantal maanden geleden sprong ik in het diepe. Ik begon met een nieuw project met een voor mij nog onbekende technologie. Als mobile .NET developer heb ik veel ervaring met het bouwen van mobiele apps, vooral met Xamarin en Xamarin.Forms. Mijn sprong in het diepe was het bouwen van een mobiele app met cross-platform framework Flutter!

In de mobile wereld kan je niet meer om Flutter heen, overal hoor je enthousiaste geluiden: Flutter is snel, mooi en makkelijk te leren. Die dingen zijn allemaal waar, maar geldt dat niet voor ieder framework?

De eerste stappen zijn vaak makkelijk, volg de "getting started guide" en 5 minuten later knippert "Hello world" op je scherm. De tegenvallers komen vaak pas als je aan de slag gaat met een real world application. Veel van die tegenvallers hebben hun basis in oude kennis: je probeert problemen op te lossen op de manier die je gewend bent met het framework of de taal waar je veel ervaring in hebt. Andere tegenvallers komen wel degelijk door de onbekende technologie. Zo ondersteunt Dart (de programmeertaal voor Flutter) geen reflectie. Gebruik van reflectie zou het optimaliseren van de app binary te lastig maken. Dat betekent bijvoorbeeld dat het (de)serialiseren van JSON handmatig moet worden uitgetikt, of doormiddel van "Code Generation" moet worden gegenereerd. Een horde die makkelijk is te overwinnen na een uurtje speuren naar best practices. Tegenvallers maken het leren van een nieuwe technologie soms frustrerend en ontmoedigend. Maar bestonden de eerste maanden dat je programmeerde ook niet uit een heleboel tegenvallers? Het belangrijkste is dat tegenvallers worden afgewisseld door overwinningen, want die geven energie! Toen ik jaren geleden begon met programmeren op een Arduino genoot ik van het ontdekken en puzzelen. Net zoals ik nu geniet van het ontdekken en puzzelen binnen Flutter en Dart, omdat ik stappen maak en doelen bereik. Nieuwe technologie leer je door uit je comfortzone te stappen met je kennis en ervaring zonder deze voor harde waarheid aan te nemen in de nieuwe omgeving. En het belangrijkste is te genieten van de stappen die je maakt! Ik geloof dat je jezelf als developer dit leerproces eigen moet maken.

We hebben het nou eenmaal niet altijd voor het kiezen: technologie zou nooit leidend mogen zijn maar moet in dienst staan van de toepassing. De enige manier om zeker te weten welke technologie de beste oplossing biedt voor een toepassing, is af en toe een sprong in het diepe te wagen en zoveel mogelijk van een andere technologie te leren. Pas dan kan je een weloverwogen keuze maken.

Mijn ervaringen met Flutter zijn fantastisch, ik leer nog steeds elke dag nieuwe dingen. Flutter is bij uitstek geschikt om apps te bouwen met een prachtige UI.

Toch zou ik voor een "data heavy" mobiele app met veel logica, zonder met mijn ogen te knipperen voor Xamarin kiezen. Wie weet of dat binnenkort .NET MAUI wordt, een veelbelovend redesign van Xamarin.Forms. MAUI is een simplificatie van de huidige architectuur en biedt desktopsupport en performance winst. Gebruik van het Generic Host model (dependency injection zoals in ASP.NET Core) en de UI keuze tussen XAML of declaratief met C# maken het een aantrekkelijk framework. Bouw ik mijn volgende app in .NET MAUI? Dat weet ik pas zeker na mijn volgende sprong in het diepe!



Duco Fronik

ZOOM.NL ACADEMY

Leer je camera van binnen en buiten kennen

Na het volgen van de cursus voorkom je fouten én mis je nooit meer dat ene moment door een verkeerde instelling

~~€ 69,90~~
voor
€ 39,95

Onze klanten beoordelen ons met

8.4



Cursusbundel: Camerabeheersing

Bestel op zoomacademy.nl

✓ Praktijkgericht ✓ Inclusief boek en online cursus ✓ Direct aan de slag

Tip: Denk goed na wanneer je exceptions gebruikt

In onderstaand voorbeeld kun je zien dat het throwen van een exception 10x trager is dan het vooraf testen op de input.

Resultaat met Exception

```
var stopwatch = Stopwatch.StartNew();
try
{
  var resultaat = int.Parse(input);
  stopwatch.Stop();

  Console.WriteLine($"Parsen succesvol:
  {stopwatch.ElapsedTicks} ticks");
}
catch (Exception)
{
  stopwatch.Stop();
  Console.WriteLine($"Parsen niet
  succesvol: {stopwatch.ElapsedTicks}
  ticks");
}
```

Parsen niet succesvol: 143438 ticks
Parsen succesvol: 70 ticks

Resultaat zonder Exception

```
var stopwatch = Stopwatch.StartNew();
var isSucces = int.TryParse(input, out _);
stopwatch.Stop();

if (!isSucces)
{
  Console.WriteLine($"Parsen niet succes-
vol:
  {stopwatch.ElapsedTicks} ticks");

  return;
}

Console.WriteLine($"Parsen succesvol:
{stopwatch.ElapsedTicks} ticks");
```

Parsen niet succesvol: 13149 ticks
Parsen succesvol: 28 ticks

Haci Cayci, Software Engineer bij Ordina MTech

Tip: Stop met diagrammen maken en begin met modelleren!

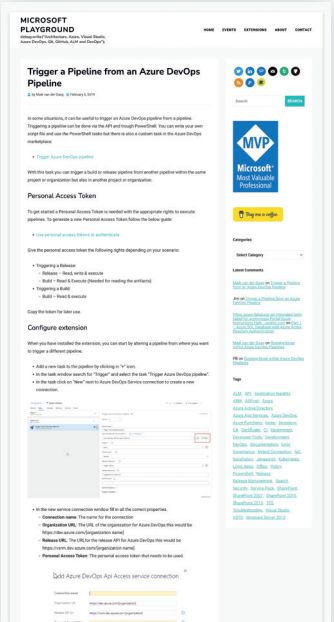
Niet met topzware architectentools, maar vanuit code: https://www.structurizr.com/. Structurizr is een implementatie van C4: een model om software te beschrijven op de 4 abstractieniveaus Context, Containers, Componenten, en Code. Zie https://c4model.com/.
Ieder team dat nu gebruik maakt van whiteboards, PowerPoint, Visio, wiki's of drawio om architectuurdiagrammen te tekenen, kent het probleem: elke tekening is een one-off; een nieuw perspectief betekent een nieuwe tekening, en updates moeten in elke tekening opnieuw verwerkt worden. Met Structurizr maak je een model van je software architectuur - zo simpel als mogelijk of zo uitgebreid als nodig. Op basis van dat ene model maak je meerdere diagrammen. Updates aan het model zie je direct terug in de diagrammen. Onderlinge afhankelijkheden zijn mooi te visualiseren, en de diagrammen zijn te embedden in bijvoorbeeld Confluence. Ook ADR's en documentatie kun je onderbrengen in Structurizr.
Voor elk team dat géén behoefte heeft aan wollige architectuurdocumenten, maar ook het gevoel heeft dat whiteboard-tekeningen wel érg vluchtig zijn: C4 en/of Structurizr vormen een mooie middenweg!

Annejan Barelds

Tip: Gates op de "Deploy" Pipeline' naar Trigger Release pipeline

In Azure Devops willen we een Release pipeline (Deployment van software naar bepaalde omgeving) automatisch aftrappen pas nadat een andere Release Pipeline (Nightly Tests) succesvol gedraaid heeft. Dit artikel heeft ontzettend geholpen: https://msftplayground.com/2019/02/trigger-a-pipeline-from-an-azure-devops-pipeline/ Het is een simpele en stabiele oplossing; er ging vooral veel tijd aan vooraf voordat mijn gedachte een shift maakte van 'Gates op de "Deploy" Pipeline' naar 'Trigger Release pipeline vanuit andere Release Pipeline' heeft.

Nadine Wolff



DEVITJOBS.NL

Software Architect .NET/Azure

Salarisindicatie:
55.000 - 80.000 EUR

Locatie:
Utrecht

Technologieën:
.NET Core, API, ASP.NET, Angular, Architect, Azure, C, C#, Cloud, CSS, DevOps, JavaScript, MVC, React, SQL

cloud republic

.NET Developer

Salarisindicatie:
60.000 - 85.000 EUR

Locatie:
Nieuwegein

Technologieën:
.NET MVC, ASP.NET, .NET Core, .NET Framework, Azure, Backend, C#, Cloud, DevOps, MS-SQL, SQL, Windows Forms, Web, .NET

ORDINA

.NET Developer

Salarisindicatie:
30.000 - 60.000 EUR

Locatie:
Ijsselstein

Technologieën:
.NET Framework, .NET MVC, C#, .NET

centric
connect.engage.succeed.

Medior Back-end Developer

Salarisindicatie::
60.000 - 65.000 EUR

Locatie:
Haarlem

Technologieën:
CI/CD, AWS, API, .NET

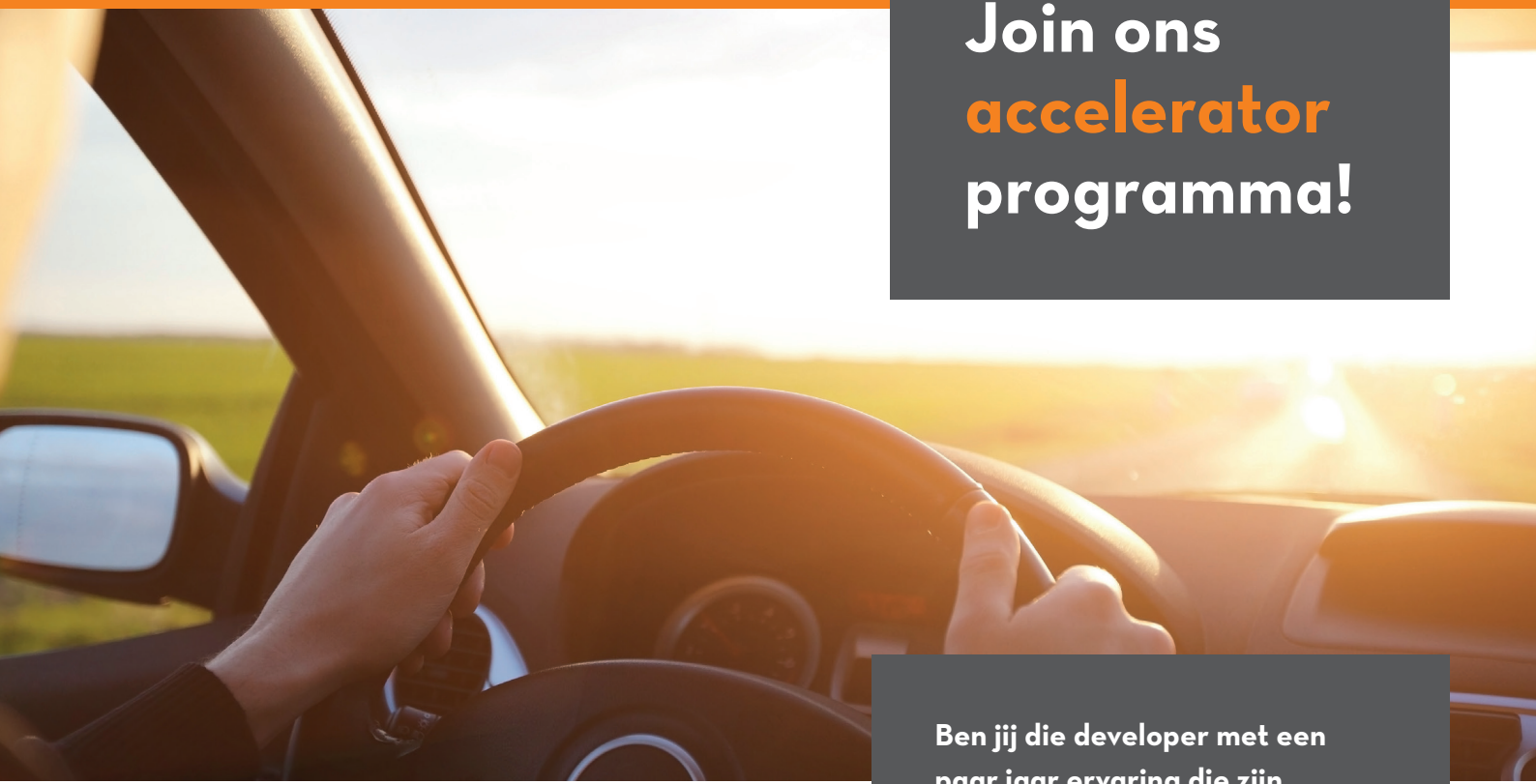
RESHIFT

Find all these vacancies and more

at Devitjobs.nl

“Heb jij graag het stuur in handen?
Met ons op maat gemaakte programma
zet je de volgende stap in je carrière!”

Join ons accelerator programma!



ORDINA

Software Development

- 1 jaar intensieve training voor en door developers
- Uitsluitend moderne technieken
- Combinatie van vakinhoud en soft skills

Ben jij die developer met een paar jaar ervaring die zijn .Net kennis naar het ‘Next Level’ wil brengen?

Word collega bij Ordina Software Development en doe mee aan dit leerzame en leuke traject!

Wil je meer informatie? Kom naar onze stand op Futuretech of mail naar cindy.davids@ordina.nl.

Microsoft • .NET6 • AZURE • DevOps • C# • OpenSource • Cloud • AI/ML • Security • PowerPlatform