Nummer 3 - 2018

JAVA MAGAZINE

Onafhankelijk tijdschrift voor de Java-professional

NEDERLANDSE JAVA USER GROUP
15 JAAR JUBILEUM NLJUG!
nljug

De toekomst in met Java Roadmap

Features

Alle nieuwe functies van Java 10

Terugblikken: J-Spring & TEQnation

Kijkje onder de motorkap met Spring Boot

Java, Polyglot, Full-stack, DevOps, T-Shaped en ...

Vandaag de dag hoor je veel termen in onze Java wereld. Ben je een Full-stack developer? Polyglot? Ben je een DevOps Engineer? Wij doen Scrum en verwachten T-Shape van je. Allemaal zaken die we heel normaal beginnen te vinden. Eén van de meest gestelde vragen die ik krijg van junior ontwikkelaars is: waar moet je beginnen? Ze horen over relationele databases, maar ook over No SQL en wat is dat JPA dan? Waar ben ik?

Nu is een voorwoord wellicht niet de plek om op die existentiële vraag antwoord te geven, maar het moge duidelijk zijn dat onze Java wereld best groot is en ook heel overweldigend kan zijn voor nieuwe ontwikkelaars. Van elke Javaan wordt eigenlijk verwacht dat ze ook weten wat Linux is en hoe je een machine moet inrichten. Leer even een nieuwe JVM taal en dat Front-end gewoon op je CV staat, met de hotste JavaScript frameworks erop, dat is toch normaal? Kotlin is sinds kort niet meer weg te denken en Scala heeft zijn niche ook al lang veroverd. Om nog maar niet over Docker, Kubernetes, cloud, serverless enzo te spreken.

Java Magazine onderkent dit en heeft ook dit blad weer een gaaf scala (woordspeling ;)...) aan boeiende artikelen die passen in deze grote wereld rondom Java. Denk aan Machine learning met Python, React Native, maar ook antwoord op de vraag: waar gaat Java heen?

Kortom, wij hopen weer een editie te hebben opgeleverd, samen met alle fantastische schrijvers, om jullie te inspireren. En ik daag jullie uit om ook een artikel te schrijven! De redactie-commissie van Java Magazine is altijd op zoek naar goede artikelen. Zoals uit dit voorwoord blijkt, hoeft dat niet alleen over Java te gaan. Dus schroom je niet en doe eens gek. Trek de stoute schoenen aan en ga los op een artikel. Je zal er geen spijt van krijgen. Om toch een beetje antwoord te geven op de vraag "waar moet ik beginnen?"
Nou, wat dacht je van hier?

Laat het me weten! ■

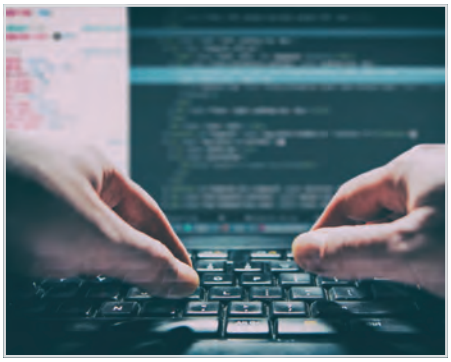
Ivo Woltring
Redactie Java Magazine
Ivo.Woltring@ordina.nl

Colofon Java Magazine 03-2018

Content manager:
Stijn van de Blankevoort
Eindredactie:
Anne Camphuijsen
Projectcoördinator:
Tedje van Gils
Auteurs:
Koen Aerts, Johannes Boneschanscher, Juan Castellanos, Edwin Derks, Erwin de Gier, Mark Hendriks, Johan Janssen, Joop Lanting, Willem Meints, Bouke Nijhuis, Bas Passon, Robert Scholte, Bertjan Schrijver, Ivo Woltring
Redactiecommissie:
Rob Brinkman, Bas Passon, Ben Ooms, Erwin de Gier, Ivo Woltring, Johan Janssen, Nanne Baars, Peter Hendriks
Vormgeving:
Britt Zaal
Uitgever:
Martin Smelt
Traffic & Media order:
Marco Verhoog
Drukkerij:
Senefelder Misset, Doetinchem
Advertenties:
Richelle Bussenius
E-mail: richelle.bussenius@nljug.org
Telefoon: 023 752 39 22
Fax: 023 535 96 27
Marc Post
E-mail: mpost@reshift.nl
Telefoon: 023 543 00 08
Abonnementenadministratie
Tanja Ekel

Lidmaatschap van de NLJUG kost € 44,50 (België € 45,50) per jaar, waarbij Java Magazine gratis verschijnt. Naast het Java Magazine krijgt u gratis toegang tot de vele NLJUG workshops en het J-Fall congres. Het NLJUG is lid van het wereldwijde netwerk van JAVA user groups. Voor meer informatie of wilt u lid worden, zie **www.nljug.org**. Een nieuw lidmaatschap wordt gestart met de eerst mogelijke editie voor een bepaalde duur. Het lidmaatschap zal na de eerste (betalings)periode stilzwijgend worden omgezet naar lidmaatschap van onbepaalde duur, tenzij u uiterlijk één maand voor afloop van het initiële lidmaatschap schriftelijk (per brief of mail) opzegt. Na de omzetting voor onbepaalde duur kan op ieder moment schriftelijk worden opgezegd per wettelijk voorgeschreven termijn van 3 maanden. Een lidmaatschap is alleen mogelijk in Nederland en België. Uw opzegging ontvangen wij bij voorkeur telefonisch. U kunt de Klantenservice bereiken via: 023-5364401. Verder kunt u mailen naar **members@nljug.org** of schrijven naar NLJUG BV, Ledenadministratie, Richard Holkade 8, 2033 PZ Haarlem. Verhuisberichten of bezorgklachten kunt u doorgeven via **members@nljug.org** (Klantenservice).

Wij nemen je gegevens, zoals naam, adres en telefoonnummer op in een gegevensbestand. De verwerking van uw gegevens voeren wij uit conform de bepalingen in de Algemene Verordening Gegevensbescherming. De gegevens worden gebruikt voor de uitvoering van afgesloten overeenkomsten, zoals de abonnementenadministratie en, indien je daar toestemming voor hebt gegeven, om je op de hoogte te houden van interessante informatie en/of aanbiedingen. je kunt uw persoonsgegevens opvragen om inzicht te krijgen in welke gegevens wij van je hebben, deze te corrigeren of, na beëindiging van de abonnee-overeenkomst, te laten verwijderen. Stuur hiertoe een kaartje aan NLJUG BV, afd. klantenservice, Richard Holkade 8, 2033 PZ Haarlem of een e-mail naar **members@nljug.org.nl**.

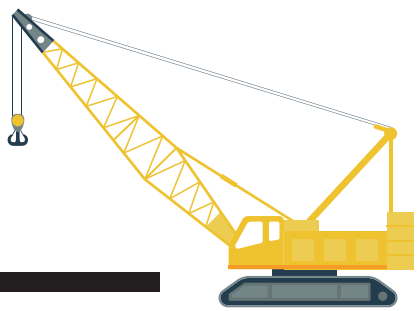


10 Alles over Java 10

Een overzicht van de laatste wijzigingen. Sinds 20 maart is Java 10 officieel afgerond en beschikbaar als kant en klare download. In dit artikel bekijken we alle nieuwe functies van deze nieuwste release. In dit artikel geven we een overzicht van enkele veranderingen qua licentie, ondersteuning, performance en andere extra's.

28 Bouwstraat voor open source projecten

We gebruiken op ons werk allemaal wel een vorm van een bouwstraat. Een vaak gemaakte keuze hiervoor is Jenkins voor het aansturen van het proces en SonarQube voor de codekwaliteit. Hoe mooi zou het zijn als je dat ook voor je eigen open source projecten zou kunnen realiseren? Thuis een eigen bouwstraat opzetten kan, maar daar heb je wel hardware voor nodig. Wat nou als je deze functionaliteit elders zou laat hosten en ook nog eens volledig gratis?



51 TEQNATION terugblik

TEQNATION The Future is Smart was een geweldige dag vol met de hotste onderwerpen zoals AI & Machine Learning, Big Data en Blockchain en sprekers zoals Laurent Picard van Google en Mark Heckler van Pivotal. Was je er helaas niet bij? Op pagina 51 krijg je een impressie en zie je hoe het eraan toeging. Wil je meer info en foto's? Kijk dan op **www.teqnation.nl**.

SCHRIJVEN VOOR JAVA MAGAZINE?

Ben je een enthousiast lid van NLJUG en zou je graag willen bijdragen aan Java Magazine? Of ben je werkzaam in de IT en zou je vanuit je functie graag je kennis willen delen met de NLJUG-community? Dat kan! Neem contact op met de redactie, leg uit op welk gebied je expertise ligt en over welk onderwerp je graag zou willen schrijven. Direct artikelen inleveren mag ook. Mail naar **info@nljug.org** en wij nemen zo spoedig mogelijk contact met je op.

De NLJUG event app(s)

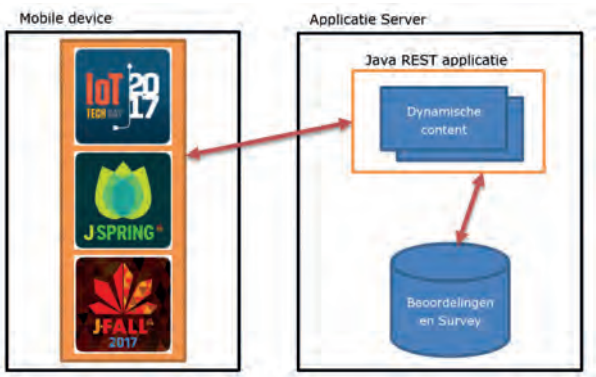
Ontwikkelt door het Mobile Compentence Centrum van de Belastingdienst

Vanaf 2015 levert de Belastingdienst een event app voor de NLJUG evenementen, zoals J-Fall. Inmiddels is deze app geevolueerd tot de breder inzetbare NLJUG event app. Je zult je wel afvragen waarom de Belastingdienst deze app levert, er zijn toch zeker een dozijn event apps in de app-stores te vinden. Voor die vraag moeten we terug in de tijd.

De NLJUG was in 2015 op zoek naar een partij om een nieuwe app te ontwikkelen voor J-Fall 2015. De Belastingdienst had zich juist in dat jaar aangemeld als Business Partner van NLJUG. De overwegingen om sponsor te worden, met 250 Java Developers werkzaam bij de Belastingdienst, laten zich raden. Wat is dan mooier en tastbaarder dan een app te leveren waarmee de organisatie kan laten zien waar het toe in staat is. Voorwaarde is wel dat de app vlekkeloos werkt, want anders krijgt je bij de doelgroep een wat minder goede naam...

De Belastingdienst is direct op de vraag van de NLJUG ingegaan. Er was op dat moment al een event app voor de startersdag gebouwd, die prima als basis kon dienen. Als echte ICT-ers hebben we samen met de NLJUG de eisen en wensen opgesteld. Zo moest de content dynamisch zijn in tegenstelling tot de bestaande app, nog een handje vol aan layout wensen, sessies moeten beoordeeld kunnen worden en een overzichtspagina voor de organisatie met de beoordelingen van de sessies en het evenement als geheel. So far, so good. Oh ja, het is nu 15 september en het moet 5 november live, dat is over anderhalve maand. Hiervan worden 4 weken gereserveerd voor de oplevering naar de app stores, omdat de ervaring leert dat dit veel doorlooptijd kost.

Dus feitelijk maar 2 weken tijd voor (her)ontwerp, bouw, backend-platform en test.
Aan de slag met een eenvoudige en doeltreffende architectuur met de onderstaande onderdelen:



De App:

Met daarin navigatie, statische content zoals logo's en pasfoto's van sprekers. Het deel van de content moet dynamisch zijn, omdat ook onverwachte situaties in de programmering kunnen optreden. De inhoud van het event en de sessie worden bij het opstarten van de app geladen vanaf een server.

Dynamische content:

- Event.json bestand, met daarin informatie over het event en de locatie.
- Schedule.json bestand, met daarin informatie over de sessies, zalen en tijden.

Java Rest service:

Voor het afhandelen van het http verkeer tussen de app en de server. Voornaamste taak is het aanbieden van de gegevens van en naar de database.

Database:

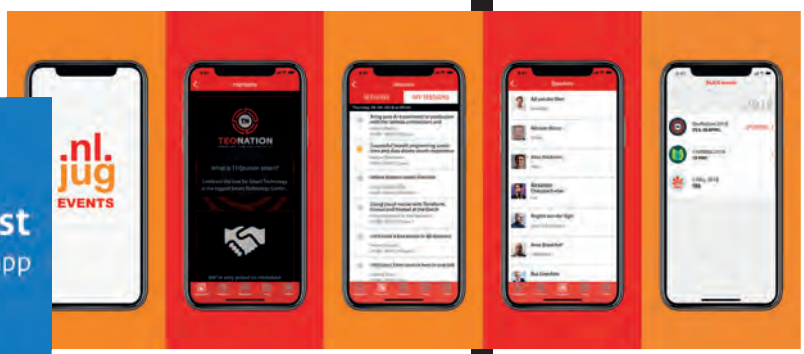
Per sessie kan er door de bezoekers een beoordeling worden gegeven en een eindoordeel

Anton de Ruiter
Competence Unit Manager / Vakpool-manager Java bij de Belastingdienst

over het hele event. Deze gegevens moeten in de database worden vastgelegd.

De rest is geschiedenis, de app draaide als een zonnetje en er zijn veel beoordelingen van de sessies binnen gekomen. De app voor J-Fall 2015 werd goed beoordeeld en vervolgens steeds opnieuw gekopieerd voor de overige events van de NLJUG: IoT Tech Days, TEQnation & J-Spring. Inmiddels staan er al drie versies van de J-Fall app in de app-stores, namelijk die voor 2015, 2016 & 2017.

Toen ontstond het idee om één generieke NLJUG event app te ontwikkelen, die voor alle toekomstige evenementen gebruikt kan gaan worden.



Hiervoor is de app aangepast zodat de app voornamelijk een lege navigatie huls is die wordt gevuld vanuit json bestanden met tekstuele content, zoals voorheen, maar ook met afbeeldingen en styling. Hiermee heeft de Belastingdienst nauwelijks nog werk aan de app en heeft de NLJUG een dynamische app en volledig beheer over de content.

Download de NLJUG event app en zodra het programma voor J-Fall 2018 bekend is verschijnt deze automatisch in de app op jouw mobiel. ■

Belastingdienst

Deep Learning en Python

Deep learning modellen in Java met Microsoft Cognitive Toolkit

Er is veel te doen in de wereld van artificial intelligence op het gebied van deep learning en neurale netwerken. Er zijn steeds betere frameworks beschikbaar waarmee je ook als ontwikkelaar steeds eenvoudiger een deep learning model kunt bouwen. Niet alleen de frameworks voor het trainen van netwerken worden beter. Ook de manier waarop je modellen in je productieomgeving kan toepassen, worden steeds beter. In dit artikel lees je hoe je een deep learning model kan trainen in Python en gebruiken in Java. Zo kan jij straks ook modellen van je collega *data scientists* effectief gebruiken op productie.

Microsoft Cognitive Toolkit als alternatief voor Tensorflow

Je zult ondertussen al wel gehoord hebben van Tensorflow, het deep learning framework van Google. Met dit framework zijn al veel interessante projecten bedacht, waarvan AlphaGo wel een van de bekendste en meest ambitieuze is. Naast Tensorflow bestaan nog meer frameworks om neurale netwerken te bouwen. Waaronder Microsoft Cognitive Toolkit (CNTK).

Ik wil je graag CNTK laten zien om twee redenen. De eerste reden is dat CNTK 8x sneller is dan Tensorflow. De tweede reden is complexer. Het kan heel goed dat je favoriete deep learning framework over vijf jaar niet meer bestaat. De enige oplossing die je dan hebt, is het opnieuw trainen van je model op een nieuw framework.

Een model trainen kost veel tijd. Daarnaast is het lastig om hetzelfde resultaat te krijgen, omdat je te maken hebt met een vorm van kansberekening. Je wilt dus niet je model opnieuw hoeven trainen in het nieuwe framework. In plaats daarvan wil je het model bij de eerste keer trainen opslaan in een open bestandsformaat.

CNTK ondersteund een open bestandsformaat onder de naam Open Neural Network Exchange (ONNX). Het ONNX formaat wordt gebruikt door onder andere Intel, Nvidia, Facebook en Microsoft. Er is nu al ondersteuning voor acht verschillende deep learning frameworks. Het niveau wisselt wel. Op dit moment ondersteunt CNTK het ONNX formaat volledig. InTensorflow kun je alleen modellen importeren. De belangrijkste reden waarom ik je CNTK laat zien, is het open formaat dat het ondersteund. Dat het framework daarnaast sneller is, is natuurlijk weer mooi meegenomen.

Een model trainen in Python

We gaan aan de slag met CNTK. Eerst trainen we een model in Python. Het model dat we gaan trainen, is een neuraal netwerk voor het herkennen van handgeschreven getallen. Hiervoor is op internet een open dataset beschikbaar, die je eenvoudig kan inladen (zie **Listing 1**).

Elk voorbeeld in de dataset bestaat uit een vector van 64 bytes die de originele afbeelding representeert (opgeslagen in de variabele X). Bij elk voorbeeld is aangegeven welk getal het representeert (opgeslagen in de variable y). De labels coderen we hierbij naar zogenaamde one-hot vectoren. We maken gebruik van de OneHotEncoder utility uit de scikit-learn library.

Willem Meints
is een Microsoft MVP en Technical Evangelist voor Info Support.

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.preprocessing import OneHotEncoder

digit_dataset = load_digits(10)

X = digit_dataset.data
y = digit_dataset.target

// Reshape, zodat we een matrix krijgen met 1 kolom
// in plaats van een vector.
y = y.reshape(-1, 1)

encoder = OneHotEncoder()
encoder.fit_transform(labels)
```

Listing 1

Het model opbouwen

Het neurale netwerk dat we gaan bouwen, staat in **Figuur 1** afgebeeld met het bijbehorende aantal neuronen in elke laag. De eerste en de derde laag zijn het belangrijkste. De eerste laag moet overeenkomen met het aantal inputs (64 in ons geval). De derde en laatste laag bestaat uit tien neuronen. Dit betekent dat voor elk getal dat we willen voorspellen, er een neuron is.

Als we straks aan het neurale netwerk vragen om een plaatje te categoriseren, dan is de output neuron met de hoogste waarde het getal dat we zoeken. In **Listing 2** bouwen we het neurale netwerk met behulp van layer functies. De Dense functie definieert een zogenaamde dense layer.

Een dense layer is een laag in het neurale netwerk die alle nodes uit de vorige laag verbindt met alle neuronen in de dense layer. De Sequential functie koppelt meerdere lagen automatisch aan elkaar.

Het model trainen

Je kan het neurale netwerk zien als een complexe wiskundige formule die je probeert te optimaliseren. CNTK laat je deze formule als zodanig opschrijven. Pas als je gaat trainen, wordt deze formule omgezet naar uitvoerbare code.

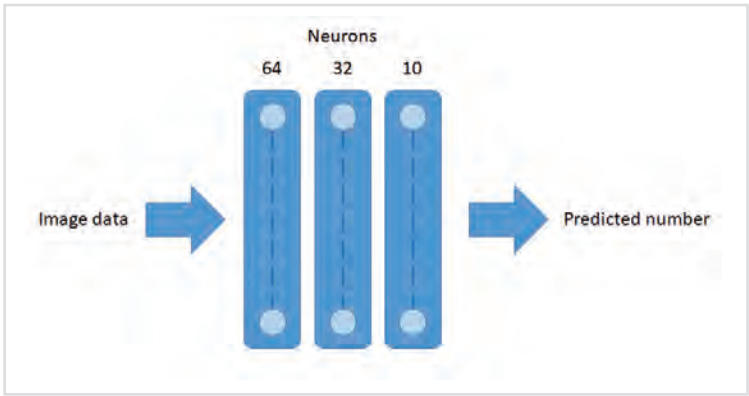
In **Listing 3** definiëren we een input variabele voor ons model. Daarna roepen we ons model aan met deze input variabele. Dit levert een nieuwe functie op, welke we opslaan in de variabele z. Om het model te kunnen trainen, heb je een loss functie nodig. Hiermee meet je of je neurale netwerk het goed doet. Met een loss function meet je hoe ver het neurale netwerk van het goede antwoord af zit.

In **Listing 4** definiëren we de loss als een categorical cross entropy functie. Deze meet het verschil tussen de categorie waarin we een plaatje verwachten te categoriseren en de werkelijke categorie. Als het neurale netwerk een goede voorspelling doet, dan is de output van deze functie laag en dicht bij de waarde 0. Als het model een onduidelijke voorspelling doet (hij valt in geen van de categorieën), dan is de output van deze functie hoog.

Naast de loss definiëren we ook een error functie. Deze gebruiken we later bij het evalueren van ons neurale netwerk. De error functie drukt uit welk percentage van de getallen we goed hebben voorspeld met ons neurale netwerk.

Voor het trainingsproces maken we een learner en een trainer aan. De learner definieert welk algoritme je wilt gebruiken voor het trainen. De trainer coördineert het leerproces. In **Listing 5** zie je hoe de code hiervoor eruit ziet.

Het trainen vindt plaats door alle voorbeelden uit een trainingset meerdere keren aan te bieden aan het neurale netwerk door middel van de trainer.



Figuur 1

```
from cntk.layers import Dense, Sequential

model = Sequential([
    Dense(64),
    Dense(32),
    Dense(10)
])
```

Listing 2

```
features = cntk.input_variable(64)

z = model(features)
```

Listing 3

```
label = cntk.input_variable(10)

loss = cntk.cross_entropy_with_softmax(z, label)
label_error = cntk.classification_error(z, label)
```

Listing 4

```
learning_rate = 0.1
learning_rate_schedule = cntk.learning_parameter_schedule(learning_rate)

learner = cntk.sgd(z.parameters, learning_rate_schedule)

trainer = cntk.Trainer(z, (loss,error_label), [learner])
```

Listing 5

In **Listing 6** splitsen we de dataset in een trainingset en validatieset. Met de trainingset gaan we aan de slag om te trainen. De validatieset gebruiken we later om te controleren of ons model goed werkt. Het trainingproces bestaat verder uit een tweetal loops. De eerste bepaalt hoe vaak we de hele dataset doorlopen voor het trainen. Het volledig doorlopen van de trainingset noemen we ook wel een epoch. In totaal trainen we tien epochs. De tweede loop is de minibatch loop. We sturen steeds kleine subsets van de data op naar de videokaart, zodat het netjes in het videogeheugen past.

Het daadwerkelijk trainen vindt plaats in de de train_minibatch methode. Deze accepteert een dictionary, waarin we een koppeling leggen tussen de input variabelen en de data die erbij hoort. CNTK gaat nu aan de slag en converteert het model naar uitvoerbare code en traint het model met behulp van de trainer op bijvoorbeeld je GPU. Meestal duurt dit proces enkele minuten.

Nadat het model is getraind kun je het valideren door de code uit **Lisitng 7** uit te voeren. Net als bij het trainen, testen we het model door middel van een minibatching proces. We gebruiken de test_minibatch methode voor dit proces. Aan het einde van de minibatch loop, hebben we de uiteindelijke score. Het model dat we hebben gemaakt, levert in dit geval een score op van 0.925, oftewel 92.5%. Dat is zeker niet de hoogste score voor het herkennen van handgeschreven getallen, maar ook niet heel slecht.

Modellen opslaan in een open formaat

Nu je een model hebt, kun je deze exporteren naar ONNX om hem te gaan gebruiken in Java. Het ONNX formaat is gebaseerd op protobuf, een ander bekend open source data formaat. ONNX is dan ook niet meer dan een officiële specificatie voor het opslaan van neurale netwerken en andere machine learning modellen in protobuf files. En zoals dat tegenwoordig gaat, staat ONNX als open source project op Github.

In **Listing 8** zie je hoe het model opgeslagen kan worden in het ONNX formaat. De variabele z bevat het model, zoals we dat eerder hadden gemaakt.

Het model gebruiken in jouw Java applicatie

Nu we het model hebben opgeslagen in ONNX kunnen we het gebruiken vanuit een ander deep learning framework of een andere taal zoals CNTK, die ook een implementatie in Java kent. De Java versie van CNTK is niet gepubliceerd op Maven. Je moet, om CNTK vanuit Java te kunnen gebruiken, een release downloaden van Github (**https://github.com/Microsoft/CNTK/releases**). In de release vind je de file cntk.jar, welke je met Maven kan installeren als een beschikbare dependency op je machine.

Nadat je de jar file lokaal in je Maven cache hebt gezet, kan je hem vanuit Maven gebruiken door hem als dependency op te nemen in je pom.xml. Om het model te kunnen gebruiken, moet je het ONNX bestand inladen met behulp van de Function class. In **Listing 10** zie je hoe dit werkt.

ER IS NU AL ONDERSTEUNING VOOR ACHT VERSCHILLENDE DEEP LEARNING FRAMEWORKS

```
from sklearn.model_selection import train_test_split

minibatch_size = 64

train_X, test_X, train_y, test_y =
    train_test_split(X, y, test_size=0.2)

for _epoch in range(1, 10):
    for batch_offset in range(0, train_X.shape[0], minibatch_size):
        features_batch = train_X[
            batch_offset:batch_offset + minibatch_size]
        labels_batch = train_y[
            batch_offset:batch_offset + minibatch_size]

        input_map = {
            features: features_batch,
            label: labels_batch
        }

        trainer.train_minibatch(input_map)
```

Listing 6

```
for batch_offset in range(0, len(test_X), minibatch_size):
    features_batch = test_X[
        batch_offset:batch_offset + minibatch_size]
    labels_batch = test_y[
        batch_offset:batch_offset + minibatch_size]

    input_map = {
        features: features_batch,
        label: labels_batch
    }

    score = trainer.test_minibatch(input_map)

print("Model score: {}".format(score))
```

Listing 7

```
z.save('model.onnx', cntk.ModelFormat.ONNX)
```

Listing 8

```
mvn install:install-file -Dfile=cntk.jar -DgroupId= com.microsoft.cntk -DartifactId=cntk -Dversion=2.5.1
```

Listing 9

```
DeviceDescriptor device = DeviceDescriptor.getCPUDevice();
Function modelFunction = Function.load(
    "model.onnx", device, ModelFormat.ONNX);
```

Listing 10

Net als in Python kun je CNTK modellen in Java op je CPU of GPU trainen of gebruiken. In **Listing 10** laden we het model in, zodat het op de CPU kan draaien.

Nadat het model is ingeladen, kan je het voeden met data. In **Listing 11** zie je dat we een batch maken die bestaat uit een FloatVectorVector met daarin een FloatVector die de pixels van het plaatje met het getekende getal bevat.

Belangrijk is dat je zorgt dat de afbeelding 8 bij 8 pixels groot is, gelijk aan de grootte van de input van je neurale netwerk. Ik heb hiervoor een losse functie gemaakt die dit regelt met behulp van de BufferedImage API in Java.

Om de voorspelling te kunnen doen, moet je de data doorgeven aan het model. Hiervoor leggen we in **Listing 12** een mapping aan tussen input variabelen en de bijbehorende data. De output geven we in dit geval wel door, maar laten we leeg.

De evaluatie methode vertaalt de input vanuit Java naar native C++ code en voert de werkelijke voorspelling uit op het device dat we eerder hebben gekozen. Om vervolgens de output uit het netwerk terug te halen naar Java, moeten we het extra stukje code toevoegen uit **Listing 13**.

De FloatVector kan je vervolgens uitlezen, om te bepalen welke neuron de hoogste waarde had. De index van de neuron is dan het getal dat in de afbeelding wordt gerepresenteerd.

```
FloatVectorVector batch = new FloatVectorVector();
FloatVector vector = new FloatVector();

for (int index = 0; index < pixels.length; index++) {
    vector.add((float)pixels[index]);
}

batch.add(vector);
```

Listing 11

```
Value inputValue = Value.createDenseFloat(
    features.getShape(), batch, device);

UnorderedMapVariableValuePtr inputMap =
    new UnorderedMapVariableValuePtr();

UnorderedMapVariableValuePtr outputMap =
    new UnorderedMapVariableValuePtr();

inputMap.add(features, inputValue);
outputMap.add(predictedDigit, null);

modelFunction.evaluate(inputMap, outputMap, device);
```

Listing 12

```
FloatVectorVector outputBuffer = new FloatVectorVector();

outputMap.getitem(predictedDigit).copyVariableValueToFloat(
    predictedDigit, outputBuffer);

FloatVector outputRecord = outputBuffer.get(0);

// Om een waarde van een output neuron te bepalen
Float neuronValue = outputRecord.get(1);
```

Listing 13

Conclusie

Er is veel gaande op het gebied van artificial intelligence. Het wordt steeds eenvoudiger om modellen te maken en te hosten in je programma's. Ontwikkelingen, zoals ONNX, dragen hier flink aan bij. De gebruiksvriendelijkheid van libraries, zoals CNTK in Java, is dan misschien nog niet het beste wat je ooit hebt gezien. Maar het laat wel goed zien wat nu al mogelijk is met deep learning in Java.

En terwijl het nu nog primair een feestje is voor *Data Scientists*, binnenkort kan ook jij als ontwikkelaar de vraag krijgen of je niet even kan helpen om een model te maken en in productie te zetten. Nu je gezien hebt dat het met een combinatie van CNTK en ONNX goed mogelijk is om een model te trainen in Python en deze vervolgens in Java te gebruiken, merk je direct dat ook jij als ontwikkelaar prima een deep learning model kan bouwen.

Ben je geïnteresseerd in de code? Deze staat op Github: **https://github.com/wmeints/cntk-deeplearning-java/.** ■

JE KAN HET NEURALE NETWERK ZIEN ALS EEN COMPLEXE WISKUNDIGE FORMULE DIE JE PROBEERT TE OPTIMALISEREN

Java 10: een overzicht van de laatste wijzigingen

De nieuwste varianten en functies

Sinds 20 maart is Java 10 officieel afgerond en beschikbaar als kant en klare download. In dit artikel bekijken we alle nieuwe functies van deze nieuwste release.

Laten we maar direct een overzicht geven van enkele varianten, qua licentie, ondersteuning, performance en extra's zoals bijvoorbeeld:

1. Oracle's Java SE met commerciële onderdelen, zoals de Flight Recorder, Mission Control en een langere periode van ondersteuning en updates.
2. De open source versie van het project OpenJDK[1] (naast de experimentele versies waar nog aan gewerkt wordt zoals bijvoorbeeld JDK 11, Project Valhalla of: Project ZGC) met de HotSpot JVM.
3. Azul's gratis Zulu (gebaseerd op Hotspot en OpenJDK) of de commerciële Zing JVM.
4. De open source Eclipse OpenJ9 JVM[2] (gedoneerd door IBM) met experimentele OpenJDK 10 nightly builds van het AdoptOpenJDK.net project[3].
5. Red Hat's OpenJDK (binnenkort – op dit moment nog Java 9.0.4) gebouwd met behulp van Iced Tea gedistribueerd voor o.a. Fedora en Red Hat Enterprise Linux[4]. Het is ook beschikbaar voor Windows als commercieel aanbod dat gratis is voor software ontwikkeling[5], maar zonder ondersteuning of gegarandeerde security updates.

Wat er veranderd in Java 10 is beschreven in JSR (Java Specification Request)-383 en goedgekeurd door leden van het Java Community Process (JCP)[6][7]. Enkele leden van de JCP zijn: Twitter (dat een eigen team van JVM experts heeft) en de London Java Community[8], bekend om hun initiatief "Adopt a JSR"[9], dat JUG-leden betrekt bij het opstellen van nieuwe JSR's. Voor Java 10 betekent dit dat het bestaat uit een lijst[10] van 12 JEP's (Java Enhancement Proposals).

Java 9 op de voet gevolgd door 10

Java 10 arriveert exact volgens planning een half jaar na de release van Java 9, waarmee Oracle stopt om gratis updates te publiceren voor Java SE 9. Dit in tegenstelling tot het 4 jaar oude Java 8, dat nog minstens tot begin 2019 gratis geüpdatet wordt. Dat is maanden langer dan dat Java 10 zal krijgen. Betekent de korte ontwikkeltijd van 6 maanden dat er weinig veranderd is? Of de korte ondersteuning dat je beter geen gebruik kunt maken van deze versie? Nee, er zijn een hoop verbeteringen doorgevoerd en als je al Java 9 gebruikt, is het heel compatibel.

Als je nog Java 8 gebruikt, moet je er rekening mee houden dat de gratis updates daarvoor ook vanaf januari 2019 afgelopen kunnen zijn. Hierdoor worden kritieke problemen op het gebied van stabiliteit en veiligheid dan niet meer door Oracle gratis opgelost en nieuwe protocollen zullen niet ondersteund worden (zoals bijvoorbeeld HTTP/2 en TLS 1.3).

Johannes Boneschanscher werkt sinds 2006 als ontwikkelaar en beheerder van Java applicaties bij Info Support.

```
java –version

openjdk version “10” 2018-03-20
OpenJDK Runtime Environment 18.3 (build 10+46)
OpenJDK 64-Bit Server VM 18.3 (build 10+46, mixed mode)
```

Listing 1

```
var x = 5; // x krijgt het type int
x = 6L;    // Fout: een long kan niet toegewezen aan een int
```

Listing 2

Dit brengt ons bij de eerste verandering: de versienummering die al werd veranderd in Java 9 (dat niet meer 1.9.x maar 9.x.y.z heet – waarbij niet-significante nullen weggelaten worden) wordt verder verbeterd.

Waar sommige Java-releases jaren op zich lieten wachten, is door de nieuwe halfjaarlijkse versies een nieuwe versienummering handiger. Dit is nu als leverancierspecifieke versie toegevoegd naast de bestaande versienummering en gaat door het leven als system property **“java.vendor.version”.** Dit krijgt de vorm: yy.MM, waardoor Java 10 (18.3) gevolgd zal worden door Java 11 (18.9) in september. Dit zou moeten helpen om snel te kunnen zien of een Java-versie nog actueel is. Dit kun je ook zien met “**java.version.date**”. Concreet ziet dat er momenteel zo uit in OpenJDK (zie **Listing 1**).

Oracle en Azul hebben overigens aangekondigd Java 11 (18.9) als LTS-versie te bestempelen en te blijven ondersteunen voor de komende acht jaar. Gratis is dat niet, maar wel zijn er plannen van adoptopenjdk.net om gratis dezelfde service te bieden voor vier jaar. Aangezien elke drie jaar een nieuwe LTS-versie uitkomt, biedt dat 1 jaar om een grote transitie te maken, in plaats van elke 6 maanden een kleinere.

Nieuwe snufjes voor softwareontwikkeling

De enige taalwijzing is de introductie van ‘var’ als typeaanduiding voor lokale variabelen. Waar je in Java als statisch getypeerde taal altijd zeker kan zijn welk type gebruikt wordt, zoals **int, double** of in wat complexere zaken iets als: **Stream<Optional<String>>** kan dit nu weggelaten worden. Het type wordt dan automatisch bepaald door de toekenning, die erachter staat. Als de compiler automatisch het type van de var bepaald heeft, wordt dit vervolgens ook afgedwongen (zie **Listing 2**).

Als je **Listing 3** en **4** met elkaar vergelijkt, zie je dat met de namen van de variabelen, ze verder vooraan op de regel staan, waardoor de code beter leesbaar is.

Overigens was var geen gereserveerd sleutelwoord in Java en dat is het nog steeds niet. **Listing 5** is dus valide Java 10.

In een aantal gevallen mag je var niet gebruiken volgens de Java Language Specification (zie **Listing 6**), omdat dit verwarring over het bedoelde type zou kunnen veroorzaken.

Er is geen extra term voor automatisch afgeleide types bedacht voor final “variabelen” (zoals val in Scala), maar je mag wel final met var combineren, zoals in **Listing 7.**

```
long klantID = 8_000_000_000L;
Klant klant = klantService.getKlant(klantID);
List<Factuur> recenteFacturen = factuurService.
getRecenteFacturen(klant);
```

Listing 3

```
var klantID = 8_000_000_000L;
var klant = klantService.getKlant(klantID);
var recenteFacturen = factuurService.
getRecenteFacturen(klant);
```

Listing 4

```
for(var var = 1; var <= 10; var++) {
   System.out.println(var);
}
```

Listing 5

```
var a = 2, b = 3.0; // Fout: mag niet meerdere declaraties combineren
var c[] = new int[4]; // Fout: mag geen array definiëren van een var type
var d; // Fout: geen initialisatiecode waarvan het type afgeleid kan worden
var e = null; // Fout: initialisatiecode is null
var f = { 6, 7 }; // Fout: array initialisatie mag niet
var g = (g = 7); // Fout: zelf referentie in de initialisatiecode
```

Listing 6

```
final var valutas = Currency.getAvailableCurrencies();
for (final var valuta : valutas) {
    System.out.printf(“code: %5s symbool: %5s \n”
                      , valuta.getCurrencyCode()
                      , valuta.getSymbol());
}
```

Listing 7

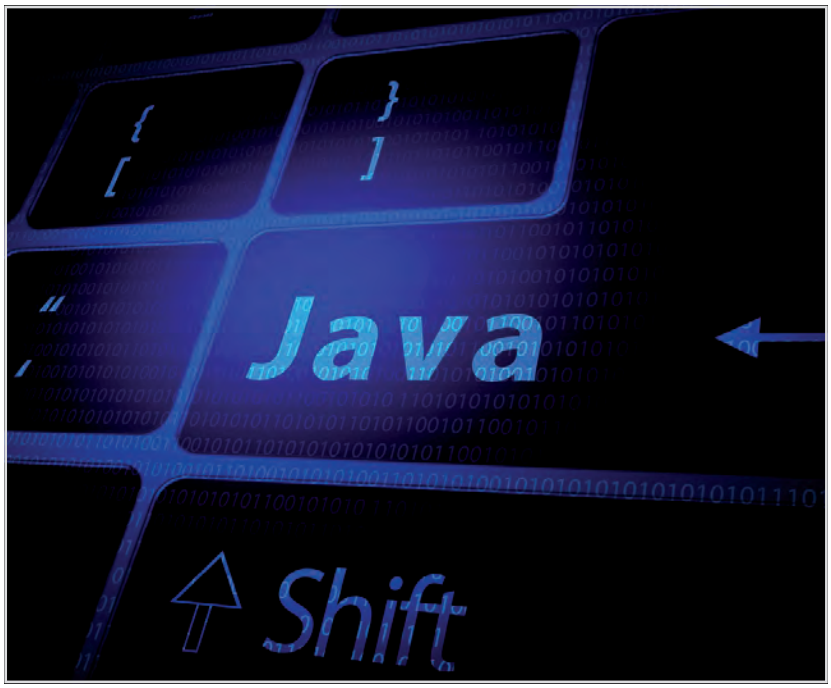
Var kan alleen voor lokale variabelen gebruikt worden, maar niet voor attributen, parameters en dergelijke. Java blijft daarmee een sterk getypeerde taal, waar compilers en IDE's de ontwikkelaar ondersteunen door fouten al tijdens het programmeren te melden.

Performance verbeteringen

Eigenlijk heeft JEP-307 niets te maken met de taal Java, maar vooral met HotSpot: de referentie implementatie van de Java Virtual Machine specificatie. De specificatie zelf beschrijft alleen dat er een automatisch mechanisme is dat oude ongebruikte objecten opruimt[11], beter bekend als de Garbage Collector. Oracle's HotSpot JVM (onderdeel van het OpenJDK project) geeft je de keuze uit verschillende Garbage Collectors, die afhankelijk van de toepassing het beste presteren. Denk hierbij aan het streven naar een zo hoog mogelijke doorvoersnelheid (batch) versus een zo laag mogelijke responsetijd (webserver) of de keuze tussen een zo laag mogelijk geheugengebruik ten koste van extra CPU gebruik. Sinds Java 9 is G1 (Garbage First) de standaard Garbage Collector als je er zelf geen configureert. Deze is vooral bedoeld voor applicaties, die zowel veel geheugen nodig hebben als ook altijd snel moeten reageren. In uitzonderingssituatie waar G1 niet snel genoeg geheugen vrijgeeft, wordt teruggevallen op een veel langer durende "Full GC", die vanaf Java 10 standaard alle CPU's gebruikt en niet meer slechts 1 CPU.

Overigens is G1 niet de enige kandidaat in dit speelveld: alternatieve experimentele Garbage Collector projecten onder de vlag van OpenJDK zijn o.a. Shenandoah[12] en ZGC[13]. Deze projecten zijn gesponsord door respectievelijk Red Hat en Oracle of de commerciële C4 Garbage Collector van de Zing JVM van Azul.

Om Java minder geheugen te laten gebruiken en sneller te laten starten, breidt JEP-310 het bestaande Class-Data Sharing uit. Hierdoor kunnen voortaan niet alleen de modules van de JVM laden, maar ook applicatiecode van het classpath en later ook modules. Dit is eerder als experimenteel onderdeel van Oracle's Java SE geïntroduceerd en wordt nu onderdeel van OpenJDK. Dit is vooral interessant waar veel instanties van dezelfde applicatie(server) draaien of vaak opnieuw opgestart worden.



Bijvoorbeeld in serverless technology, zoals Apache OpenWhisk.

Tot slot streeft JEP-312 met Thread-Local Handshakes naar het zo min mogelijk pauzeren van alle threads tegelijk. In de HotSpot JVM was het tot nu toe gebruikelijk dat als er achter de schermen iets moest worden aangepast of vanuit een andere thread een stacktrace opgevraagd werd, hiervoor alle applicatie threads tijdelijk moesten stoppen. Pas als de laatste ook gestopt was (een 'safe point' bereikt had) kon hiermee begonnen worden. Met deze wijziging wordt het mogelijk om een actie klaar te zetten voor alle threads, die ze kunnen uitvoeren op het eerstvolgende moment dat voor die thread handig is. Daarna kunnen ze door draaien zonder daarbij op andere threads te moeten wachten.

Experimentele wijzigingen

JEP-316 en 317 roepen meer vragen op dan dat ze beantwoorden, maar zijn wel interessant vanwege hun innovatie.

- JEP-316 maakt het mogelijk om Java te starten, waarbij de Heap niet het normale geheugen (DRAM) gebruikt, maar alternatieve technologieën waaronder NV-DIMM's. Deze niet-vluchtige geheugen-modules overleven stroomstoringen, maar zijn nu nog duurder en langzamer

AL DEZE WIJZIGINGEN LATEN ZIEN DAT JAVA 10 IS MEEGEGROEID MET DE NIEUWE TRENDS IN SOFTWARE-ONTWIKKELING, HOSTING EN HARDWARE

dan DRAM. Wellicht als dit soort geheugen toevallig over is op het systeem waar Java op draait, kan het nu in ieder geval gebruikt worden. Echter maakt Java 10 geen gebruik van de niet-vluchtige eigenschappen. Als NV-DIMM's goedkoper worden per gigabyte dan normale DRAM, dan wordt het wellicht interessant voor applicaties, die profiteren van veel geheugen.

- Verder is de Graal compiler om bytecode naar machinecode te vertalen nu als alternatieve JIT compiler toegevoegd. Volgens Chris Seaton van Oracle[14] moet deze Java implementatie de huidige C++ implementatie vervangen om beter te kunnen innoveren.

Voorjaarschoonmaak

Een aantal JEP's kunnen samengevat worden als voorjaarsschoonmaak:

- Het werken aan Java moet eenvoudiger worden door het samenvoegen van de acht verschillende broncode repositories waaruit het bestond.
- Door een reorganisatie van de Garbage Collector code moet het eenvoudiger worden oude implementaties (zoals CMS) te verwijderen en nieuwe (zoals Shenandoah) toe te voegen.
- En tot slot verdwijnen twee tools: policyeditor en javah. Deze laatste werd tot Java 8 gebruikt voor het genereren van C/C++ header bestanden (*.h) voor het implementeren van Java-methodes, die als 'native' gedeclareerd zijn. Sinds Java 8 kan dit tijdens het compileren door javac gedaan worden. Dat scheelt weer een stap in het bouwproces.

Overig nieuws

OpenJDK zal vanaf Java 10 standaard nog bruikbaarder zijn door de meegeleverde lijst van vertrouwde certificaatautoriteiten. Dit maakt het aantal verschillen tussen Java SE van Oracle en OpenJDK nog kleiner. Zonder deze lijst werkt SSL/TLS niet. Gelukkig doet het overgrote deel van de partijen uit het Oracle Java SE Root CA programma mee en waar nodig kunnen nog steeds extra certificaat autoriteiten toegevoegd worden met tools als keytool of Portecle.

Op het gebied van internationalisatie werd de afgelopen jaren gewerkt aan het implementeren van nieuwe standaarden in Java zoals: Common Locale Data Repository (CLDR) en diverse BCP-47 extensies, van het Unicode Consortium. Toegevoegd in JEP-314 zijn: monetaire eenheden, eerste dag van de week, regio's en tijdzones.

Buiten de JEP's om zijn er nog allerlei verdere verbeteringen in Java 10, zoals:

- Integratie met Linux cgroups voor Docker containers om geheugen- en processorgebruik te limiteren. Zonder deze wijziging wordt aangenomen dat je een maximum heap wil van 25% van het systeem waarop de container draait, ongeacht de limieten van de container.
- Nieuwe collectors in java.util.stream. Collector en de methode copyOf in de standaard Collection klassen maken het eenvoudig om collecties te maken, die onveranderbaar (immutable) zijn, vooropgesteld dat de elementen zelf ook onveranderbaar zijn.
- De Optional klasse heeft een nieuwe methode orElseThrow gekregen.

Conclusie

Al deze wijzigingen laten zien dat Java 10 is meegegroeid met de nieuwe trends in softwareontwikkeling, hosting en hardware. Dit is mede te danken aan bedrijven die geloven in Java (zoals Oracle, IBM en Red Hat) en dit gezamenlijk als open source verder ontwikkelen. Wie nog afhankelijk is van een oude Java-versie heeft nu nog meer argumenten om te upgraden naar de beste versie van Java tot nu toe. ■

JAVA 10 IS MEEGEGROEID MET DE NIEUWE TRENDS IN SOFTWARE-ONTWIKKELING, HOSTING EN HARDWARE

REFERENTIES

1 **http://jdk.java.net/10/**
2 **https://www.eclipse.org/openj9/oj9_resources.html**
3 **https://adoptopenjdk.net/nightly.html**
4 **https://fedoraproject.org/wiki/Changes/Shenandoah**
5 **https://developers.redhat.com/products/openjdk/download/**
6 **https://en.wikipedia.org/wiki/Java_Community_Process**
7 **https://www.jcp.org/en/jsr/detail?id=383**
8 **https://www.jcp.org/en/jsr/results?id=6107**
9 **https://community.oracle.com/community/java/jcp/adopt-a-jsr**
10 **http://openjdk.java.net/projects/jdk/10/**
11 **https://docs.oracle.com/javase/specs/jvms/se10/html/jvms-2.html**
12 **https://www.youtube.com/watch?v=qBQtbkmURiQ**
13 **https://www.youtube.com/watch?v=tShc0dyFtgw**
14 **http://chrisseaton.com/truffleruby/jokerconf17/**

Java EE 8

Maak kennis met deze nieuwe release (de laatste onder Oracle)

Met alle nieuwe releases van Java SE zouden we bijna vergeten dat ook een nieuwe versie van Java EE is uitgekomen, namelijk Java EE 8. Dat is meteen een erg bijzondere release gezien het de laatste is onder de paraplu van Oracle. Inmiddels is Java EE verhuisd naar de Eclipse Foundation en heeft het een nieuwe naam gekregen: Jakarta EE. In dit artikel kun je meer lezen over een aantal van de interessante features van Java EE 8.

Java EE 8

De grootste verandering in Java EE 8 is wellicht de ondersteuning voor Java SE 8. Je kan nu in Java EE 8 van alle mooie features gebruik maken die Java SE 8 te bieden heeft. Daarnaast zijn er een aantal volledig nieuwe onderdelen (oftewel specificaties) en hebben andere onderdelen een minor of major upgrade gekregen. Zelfs als je niet direct met Java EE 8 werkt of gaat werken, dan zijn deze features nog steeds interessant. Je kan ze namelijk ook los (zonder een volledige Java EE applicatie) in jouw project gebruiken, bijvoorbeeld binnen Spring.

Bean Validation 2.0

Het Bean Validation project wordt veel gebruikt, zowel binnen Java EE, maar ook bij bijvoorbeeld Spring. In versie 2.0 zijn een aantal nieuwe features toegevoegd. Onder meer om Java 8 features, zoals Optional en de nieuwe datum/tijd functionaliteit van bijvoorbeeld LocalDateTime, te kunnen gebruiken.

Een nieuwe feature is het gebruik van meerdere dezelfde annotaties op één veld. Voorheen kon dat niet zonder de annotaties in een lijst op te nemen, zoals in **Listing 1**.

Inmiddels kan dat een stuk makkelijker, zoals te zien is in **Listing 2**, zonder dat je er een lijst omheen hoeft te zetten.

Bij collecties, zoals in **Listing 3**, was het al mogelijk om validaties op de lijst zelf toe te passen.

Inmiddels is het ook mogelijk om validaties toe te passen op de elementen van een lijst, zoals in **Listing 4** te zien is. Dus alle elementen mogen niet leeg zijn en moeten voldoen aan de reguliere expressie.

Een andere nieuwe mogelijkheid voor collecties is het valideren van keys en values van bijvoorbeeld een Map, zoals in **Listing 5**.

Optional is nieuw in Java 8 en ook daar is met Bean Validation 2.0 ondersteuning voor gekomen. In **Listing 6** is te zien dat het e-mail adres optioneel is.

Johan Janssen
is trainer bij Info Support en bedenker/organisator van JVMCON.

```
@Size.List({
        @Size(min = 8, groups = Gebruiker.class),
        @Size(min = 15, groups = Beheerder.class)
})
private String wachtwoord;
```

Listing 1: de 'oude' manier om meerdere, dezelfde annotaties op één veld te zetten

```
@Size(min = 8, groups = Gebruiker.class)
@Size(min = 15, groups = Beheerder.class)
private String wachtwoord;
```

Listing 2: de 'nieuwe' manier om meerdere dezelfde annotaties op één veld te zetten

```
@NotEmpty
private List<Medewerker> medewerkers;
```

Listing 3: valideren van de lijst zelf

```
private List<@NotEmpty @Pattern(regexp="") String> laatsteWerkplekken;
```

Listing 4: valideren van elementen in de lijst.

```
private Map<@Valid MijnKeyType, @Valid MijnValue> waarden;
```

Listing 5: valideren van elementen in de map

```
Optional<@Email String> getEmail() {
```

Listing 6: een optioneel e-mail adres

```
@Email
@NotEmpty
@NotBlank
@Positive
@PositiveOrZero
@Negative
@NegativeOrZero
@PastOrPresent
@FutureOrPresent
```

Lijst 1: nieuwe ingebouwde constraints van Bean Validation 2.0

Er waren al een aantal ingebouwde contraints, zoals @Size, die je zo kon gebruiken. Echter, die waren niet voldoende voor alle mogelijke scenario's die men wilde valideren. De ontwikkelaars van Bean Validation 2.0 hebben onder andere gekeken naar veel voorkomende constraints die mensen zelf geprogrammeerd hadden in GitHub en die toegevoegd aan de ingebouwde constraints. In **Lijst 1** zijn de nieuw toegevoegde constraints te zien.

Naast al deze mooie features heeft het team ook zogenaamde extension points toegevoegd. Met die extension points kan ondersteuning voor andere collectie frameworks, zoals Google Guave en andere talen zoals Scala, gebouwd worden.

JAX-RS

Twee van de grootste vernieuwingen in JAX-RS zijn server send events en de reactive client API. De reactive client voegt de 'rx' methode toe en levert een zogenaamde CompletionStage op. In **Listing 7** is te zien hoe twee verschillende URL's worden aangeroepen. Daarna worden de resultaten gecombineerd en kun je ermee doen wat je wilt.

Servlet 4.0

Servlets hebben ondersteuning gekregen voor HTTP/2, waarmee de performance verbeterd kan worden. De grootste vernieuwing is de toevoeging van server push. Met server push kunnen resources alvast naar de browser cache van de gebruiker gestuurd worden voordat de browser de resources zelf gaat ophalen. Als de browser vervolgens de resources wil ophalen, dan zijn ze al beschikbaar in de cache. In **Listing 8** is te zien hoe de server alvast een css en png file naar de browser cache stuurt, voordat de browser hier zelf actief om vraagt.

```
CompletionStage<Response> stage1 = ClientBuilder.newClient()
              .target("url1")
              .request()
              .rx()
              .get();

CompletionStage<Response> stage2 = ClientBuilder.newClient()
       .target("url2")
       .request()
       .rx()
       .get();

stage1.thenCombine(stage2, (result1, result2) ->
```

Listing 7: twee URL's worden aangeroepen en het resultaat wordt gecombineerd

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
    PushBuilder builder = request.newPushBuilder();
    builder.path("css/custom.css").push();
    builder.path("images/logo.png").push();
```

Listing 8: een css en png file worden naar de browser cache gepushed

```
@BasicAuthenticationMechanismDefinition(realmName="${'my-realm'}")
@WebServlet("/user")
@DeclareRoles({ "admin", "user"})
@ServletSecurity(@HttpConstraint(rolesAllowed = "user"))
public class UserServlet extends HttpServlet {
```

Listing 9: basic authentication

```
@LdapIdentityStoreDefinition(
                  url = "ldap://url:port/",
                  callerBaseDn = "..",
                  groupSearchBase = ".."
)
```

Listing 10: LDAP Identity Store

Java EE Security API (nieuwe API)

In Java EE 8 is het niet langer nodig om de security configuratie in de web.xml op te nemen en in applicatie server specifieke bestanden. In plaats daarvan is een op annotaties gebaseerde oplossing gebouwd die werkt voor alle applicatie servers die de specificatie implementeren. Authenticatie kan gedaan worden door één van onderstaande annotaties te gebruiken. **@BasicAuthenticationMechanismDefinition**, **@FormAuthenticationMechanismDefinition**, **@CustomFormAuthenticationMechanismDefinition**

In **Listing 9** is bijvoorbeeld te zien hoe je basic authentication implementeert. Natuurlijk moet je daarvoor wel ergens de gebruikers informatie kunnen ophalen, zoals bijvoorbeeld de gebruikersnaam, wachtwoord en in welke groepen de gebruiker zit. Die informatie zit in een zogenaamde identity store. Een identity store kan bijvoorbeeld een database (@DatabaseIdentityStoreDefinition) zijn, of een LDAP server zoals in **Listing 10**.

MET JAVA EE 8 ZIJN WEER EEN HELEBOEL INTERESSANTE FEATURES BIJGEKOMEN

```
String medewerkerJson = JsonbBuilder.create().toJson(origineleMedewerker);
Medewerker medewerker = JsonbBuilder.create().fromJson(medewerkerJson, Medewerker.class);
```

Listing 11: serialization en deserialization met JSON-B

```
public class Medewerker {
        private String naam;
}
{
    “naam”: “James”,
}
```

Listing 12: object en de bijbehorende JSON

```
public class Medewerker {
        @JsonbProperty(“voornaam”)
        private String naam;
}
{
    “voornaam”: “James”,
}
```

Listing 13: het ‘naam’ veld in het object wordt omgezet naar een ‘voornaam’ veld in de bijbehorende JSON

```
JsonbConfig config = new JsonbConfig()
    .withPropertyNamingStrategy(PropertyNamingStrategy.LOWER_CASE_WITH_UNDERSCORES);

JsonbBuilder.create(config);
```

Listing 14: strategie voor de veldnamen aanpassen

Als laatste is een SecurityContext interface gemaakt die met de @Context annotatie geïnjecteerd kan worden. De SecurityContext kan je gebruiken om tegenaan te programmeren en bevat methoden zoals getUserPrincipal en isUserInRole(String role).

JSON Binding API (nieuwe API)

JSON-B ondersteunt serialization en deserialization tussen Java objecten en JSON. Dat kan door gebruik te maken van standaard mappings of door zelf met annotaties te bepalen wat de mapping wordt. In **Listing 11** is te zien hoe eerst een medewerker object naar JSON omgezet wordt met serialization. Vervolgens wordt met deserialization de JSON weer omgezet naar een medewerker object.

Een voorbeeld van de Java class en de bijbehorende JSON is te zien in **Listing 12**.

Dit werkt prima als je tevreden bent met de standaard mapping, maar het kan natuurlijk zo zijn dat je de naamgeving in je JSON berichten anders wilt hebben dan in de Java objecten. Dat kan je gemakkelijk realiseren door gebruik te maken van de @JsonbProperty annotatie. Als je de annotatie op het veld plaatst, dan wordt die gebruikt voor zowel de serialization als de deserialization. Het is ook mogelijk om de annotatie op de getter te plaatsen, zodat die alleen meegenomen wordt voor de serialisatie. Als de annotatie op de setter geplaatst wordt, dan wordt die alleen meegenomen voor de deserialisatie. In **Listing 13** is te zien dat het ‘naam’ veld in het Medewerker Java object wordt omgezet naar een ‘voornaam’ veld in de bijbehorende JSON.

Vaak is het zo dat je alle velden op dezelfde manier wilt weergeven, bijvoorbeeld allemaal in lowercase met een liggende streep ertussen. Dat kan door gebruik te maken van de PropertyNamingStrategy. Standaard zijn al flink wat varianten beschikbaar, maar je kunt ook een eigen strategy maken. In **Listing 14** zie je hoe een JsonbConfig wordt aangemaakt met een specifieke strategie. Deze configuratie wordt vervolgens meegegeven bij het creëren van de JsonbBuilder.

Als je bepaalde velden niet mee wilt nemen, dan kan je gebruik maken van de @JsonbTransient annotatie. Net als bij de JsonbProperty annotatie wordt die gebruikt voor serialization en/of deserialization afhankelijk van waar je de annotatie plaatst. In **Listing 15** is te zien dat de woonplaats niet meegenomen wordt voor de serialization en deserialization.

Naast deze mogelijkheden zijn nog veel meer opties bij JSON-B. Genoeg om nóg een artikel mee te kunnen vullen. Mocht je meer over JSON-B willen weten of er meteen mee aan de slag willen gaan, dan adviseer ik je zeker om hier nog meer over te lezen.

```
public class Medewerker {
        @JsonbProperty(“voornaam”)
        private String naam;
        @JsonbTransient
        private String woonplaats;
}
{
    “voornaam”: “James”,
}
```

Listing 15: gebruik van @JsonbTransient op het woonplaats veld zorgt ervoor dat woonplaats niet meegenomen wordt bij serialization en deserialization.

ZELFS ALS JE NIET DIRECT MET JAVA EE 8 WERKT OF GAAT WERKEN, DAN ZIJN DEZE FEATURES NOG STEEDS INTERESSANT

```
@Inject
private Event<LogEvent> events;

public void saveMedewerker(Medewerker medewerker)
{
        // Code om de medewerker op te slaan
        events.fireAsync(new LogEvent(medewerker));
}
```

Listing 16: versturen van een asynchroon event

```
public void logger(@ObservesAsync LogEvent logEvent) {
        logger.info(logEvent.toString());
}
```

Listing 17: afvangen van een asynchroon event

```
public void loggen(@Observes @Priority(1) Medewerker medewerker) {
        // Log de medewerker information
}

public void opslaan(@Observes @Priority(2) Medewerker medewerker) {
        // Sla de medewerker op, hierbij kan een exceptie optreden
}
```

Listing 18: observers een prioriteit geven om de volgorde van aanroepen te bepalen

Context and Dependency Injection (CDI)

Eén van de grote vernieuwingen voor CDI is het feit dat we nu events asynchroon kunnen verzenden en ontvangen. Het versturen van een asynchroon event is eenvoudig te realiseren met de fireAsync methode, zoals in **Listing 16** te zien is.

Vervolgens kunnen we het event afvangen met de observer annotatie @ObservesAsync, zoals in **Listing 17**.

Met CDI 1.1 worden de observers synchroon aangeroepen als een event wordt gestuurd. De volgorde waarin de observers worden aangeroepen, kan niet vastgelegd worden en is dus willekeurig. Het grootste probleem daarmee is dat als een observer een exceptie gooit, dan ontvangen alle volgende observers het event niet meer. Met CDI 2.0 kun je hier grotendeels omheen werken aangezien je iedere observer een prioriteit kunt geven. De laagste prioriteit wordt als eerste aangeroepen.

In het laatste voorbeeld, **Listing 18**, is te zien dat het event eerst gelogd wordt en pas daarna opgeslagen wordt. Bij het opslaan kan namelijk een exceptie optreden. Deze code zorgt ervoor dat er in ieder geval iets wordt gelogd, voor het geval de exceptie optreedt, en op die manier te voorkomen dat niets gelogd wordt.

Conclusie

Met Java EE 8 zijn weer een heleboel interessante features bijgekomen. Niet alleen voor de Java EE ontwikkelaars, maar ook voor ontwikkelaars die direct of indirect (via bijvoorbeeld Spring) van een van deze features gebruik maken. Deze features maken het een stuk gemakkelijker om goede code te schrijven. Dus zorg ervoor dat je zo snel mogelijk upgrade en gebruik gaat maken van al deze mooie verbeteringen. ■

J-Spring 2018 in beeld

De NLJUG kijkt terug op een fantastische J-Spring 2018 en bedankt daarvoor alle bezoekers en sponsoren.
Tot volgend jaar!

SPONSOREN

topicus

Rabobank

EXPOSANTEN

CarthagoICT
CONSULTING. ICT PROJECTS. WEBSOLUTIONS.

CGI

GROUP9
ELITE JAVA DEVELOPMENT

OVSOFTWARE

Profit4Cloud

Rijksoverheid

sultansofjava

vijfhart
IT-OPLEIDINGEN

YACHT

Hyperledger Fabric & Java @ J-spring

In februari kreeg ik de vraag of ik op J-Spring een presentatie wilde geven namens Topicus. Op dat moment zat ik net in een experiment met de integratie van Hyperledger Fabric in Topicus KeyHub, onze identity and access managementoplossing. Aangezien er op J-Fall al een aantal inleidende presentaties gegeven waren over blockchain, leek me dit een uitstekend moment om eens wat dieper op de materie in te gaan.

In de maanden die volgden heb ik mij meermaals afgevraagd of dit wel zo'n verstandige beslissing was. Het is lang geleden dat ik met een dergelijk complex systeem gewerkt heb. Er waren momenten dat ik dacht dat ik er niet door zou komen, maar de aanhouder wint. Na een Hyperledger Fabric training van 3 dagen en meerdere weken prutsen is het gelukt om het systeem te doorgronden en de proof of concept werkend te krijgen. Het uitwerken van de presentatie zelf viel hierna reuze mee.

De woensdag voor J-Spring waren alle sprekers uitgenodigd voor een rondleiding door TivoliVredenburg. Als echte artiesten mochten wij door het hele pand, hadden we toegang tot onze eigen kleedruimte en konden we zelfs gebruik maken van een eigen lift. Hierna zijn we vertrokken naar Kafé België voor een gezellig etentje. Op het menu stond een heuse beer can chicken en natuurlijk veel Belgisch bier. We hebben een zeer geslaagde avond gehad met z'n allen, die de volgende ochtend ook nog wel te merken was.

Ik heb J-Spring 2018 ervaren als een zeer leuke en leerzame dag. Als ontwikkelaar bij Topicus Security was ik ook zeer verrast met het ruime aanbod aan security gerelateerde presentaties. Het is goed om te zien dat er steeds meer aandacht geschonken wordt aan dit belangrijke onderwerp. Gaandeweg de dag merkte ik wel dat het lastig ging worden om mij te meten aan mijn medesprekers. Het niveau lag hoog.

Om 14:05 was het dan eindelijk zo ver. Het was mijn beurt om spreken. Het zaaltje was goed gevuld. Ik kon merken dat ik toch wel licht gespannen was. Gelukkig verdween dit snel en heb ik iedereen mee kunnen nemen in dit onderwerp dat zeker niet makkelijk was. Ik wil nogmaals iedereen bedanken voor het luisteren naar mijn verhaal en de leuke reacties die ik gehad heb. Via de app is mijn sessie beoordeeld met gemiddeld 3.7 van 5 sterren, een prestatie waar ik trots op ben.

Aan de organisatie van J-Spring wil ik zeggen: bedankt voor het organiseren van deze geweldige dag. Het is mij zo goed bevallen dat ik serieus overweeg om vaker dit soort praatjes te houden. Wie weet zien jullie mij dus weer terug op een volgende conferentie. Er zijn in ieder geval nog genoeg zaken waar over te vertellen valt. ■

Emond Papegaaij
is Java ontwikkelaar bij Topicus Security'

topicus

Most Dev: waar developers elkaar inspireren en versterken

Heb je 'm gezien – of nee, gehoord – op J-Spring? De ultramoderne muziekapplicatie op de retro Commodore 64? Het is een van de vele bijzondere projecten die tot stand komen binnen Most Dev van Yacht, de community voor en door developers.

Gave ideeën realiseren, kennis delen en werken aan je ontwikkeling. Dat zijn de pijlers van Most Dev.

Software-engineers gespecialiseerd in backend-, frontend- en mobile development vinden hier alles om hun carrière te versnellen: uitdagende opdrachten, een schat aan vakkennis en persoonlijke groei. Plus een stevige portie fun. Bijvoorbeeld in de vorm van hackathons en IT-challenges.

Teamspirit

De muziekapplicatie op J-Spring trok de aandacht. "Missie geslaagd", vindt communitymanager Kay van Raak. "Yacht staat in de IT-wereld vooral bekend als detacheerder. Dat beeld klopt niet: de developers in onze community inspireren elkaar, sparren en werken in teamverband aan grote vraagstukken van opdrachtgevers. Ze ontwikkelen zich razendsnel. Onder meer dankzij het Most Dev Talent Program."

Toonaangevende opdrachtgevers

Uitdagende opdrachten typeren Most Dev. Yacht is onderdeel van Randstad Holding en heeft contracten met veel grote spelers. Daardoor zijn er ook op development-gebied volop kansen. Van vaste banen tot interessante opdrachten voor zzp'ers. Most Dev is landelijk actief en groeit hard. "Sinds kort zijn we ook aangesloten bij NLJUG, daar zijn we blij mee", stelt Kay van Raak. "We hebben verschillende bijzondere events op de planning staan waarbij alle developers in Nederland welkom zijn. Je gaat nog veel van ons horen!"

Voor meer informatie: **www.yacht.nl/mostdev.** ■

YACHT

J-Spring 2018 met OVSoftware & RoboTeam Twente

J-Spring 2018 was weer een groot succes! Dit jaar stonden we samen op een stand met het RoboTeam Twente én hun robot voor het WK Robotvoetbal. RoboTeam Twente: 'Voor ons is het geweldig om op de J-Spring te staan samen met OVSoftware. We kunnen ons gezamenlijk profileren aan een grote groep enthousiaste software engineers.'

Uiteraard waren er weer vele interessante sessies. Alle verschillende sessies zorgen ervoor dat onze medewerkers graag een bezoekje brengen aan J-Spring. Eric: 'J-Spring was ook dit jaar weer een leuk evenement met ervaren sprekers. Dit soort evenementen geeft een prima mogelijkheid om in 1 dag bijgepraat te worden over nieuwe ontwikkelingen

en aandachtspunten. Dit keer lag de nadruk op security; van het verkeerd gebruik van 'cookies' in websites tot het hacken van auto's. Daarnaast was er de nodige aandacht voor het gebruik van agile ontwikkelmethoden in de praktijk en de mogelijkheden van moderne cloud diensten. Een ander aspect van een evenement zoals J-Spring is dat je oud-collega's tegenkomt, al met al altijd weer een gezellige dag!'

In onze ogen was het een geslaagde en succesvolle dag! Wij kijken alweer uit naar de J-Fall komende november. Wil je graag meer weten over OVSoftware? Dan ben je altijd welkom voor een lekkere kop koffie. We hebben vestigingen in Enschede, Apeldoorn, Amersfoort en Den Haag. Mail Martijn Klein Haarhuis voor meer informatie: Martijn.Klein.Haarhuis@OVSoftware.com. Bellen of appen mag natuurlijk ook: +31681777175.

Organisatie, bedankt voor weer een mooie J-Spring! ■

OVSOFTWARE

Developing tomorrow's bank today

Rabobank is looking for new IT Colleagues

With our IT team you will be at the heart of tomorrow's Rabobank. Whether it is our Rabobank App, used by over 3 million customers, our responsive web platform or the latest apps and widgets we're creating for smartwatches, you will be improving and mindshifting the way our customers interact with Rabobank.

Being part of this means you have the experts around you to learn, the less experienced to coach and space for your experience to flourish. We believe in having you in the driver's seat and guiding us to the future. Join us and become part of one of the biggest IT teams within the financial market where your work and your next step is all about you.

Interested in our opportunities? Check rabobank.nl/ict

Rabobank

Carthago ICT, altijd (pr)ijs tijdens een zomerse J-Spring in Tivoli Vredenburg!

Na het succes van de J-Spring revival van 2017 was Carthago ICT uiteraard ook bij deze editie weer van de partij. Met een mooie line-up van (inter)nationale sprekers op een inspirerende locatie: pop-podium Tivoli, wilden wij met onze stand niet ontbreken.

En gezien het succes van de ijscokar, toch ook die maar helemaal naar de 6e verdieping gebracht.
Al vroeg zagen wij de altijd enthousiaste Java menigte Tivoli binnenstromen, mooi op tijd voor de opening en daarna de Keynote door Daan Keuper.
Tijdens de breaks was voldoende tijd om de verschillende stands te bezoeken, al was de doorstroming naar de "beursvloer" af en toe wat gestagneerd.
Onze ijsjes vonden uiteraard weer gretig aftrek en het gaf ons de kans om met velen van jullie in gesprek te komen.
De 24BRIGHT Game is nog steeds een uitdaging: slechts een enkeling durfde de uitdaging aan om het aloude Dr. Bibber, in een modern jasje gestoken naar het 24BRIGHT logo van Carthago, te spelen. Wij hadden voor de beste speler van de dag een mooi "Carthago Hoogvliegers Prijs": een vliegles op Vliegveld Teuge.
De beste tijd van deze editie van J-Spring is gescoord door: Daniel (zie foto)
Wil jij ook een Carthago Hoogvlieger worden? Kijk eens op onze website wat wij aan mooie projecten en consultancy opdrachten hebben lopen en neem gerust eens contact met ons op.

Wij kijken in ieder geval terug op een geslaagde J-Spring en kijken nu al uit naar J-Fall. ■

Carthago ICT
CONSULTING. ICTPROJECTS. WEBSOLUTIONS.

Group9 op J-Spring 2018

Een gave J-Spring is helaas weer achter de rug. Een dag lang, samen met een hoop developers, bezig zijn met Java en luisteren naar de talks die gepland stonden, en leuke gesprekken met andere developers hierover. In die talks waren verschillende interessante zaken te horen waar we als GROUP9 dagelijks bij klanten mee bezig zijn: Continuous Delivery, Microservices, Reactive, klanten van Waterval naar Agile brengen, hacken van autos - ok, die doen we eigenlijk niet zo heel vaak bij klanten ;-).

Hot-topics als AI en Blockchain waren natuurlijk ook aanwezig, met interessante verhalen over wat daar mogelijk is. In de praktijk zien we dat het natuurlijk de vraag is wanneer reguliere klanten dit gaan gebruiken. Vaak zie je dat dit soort innovaties tijd nodig hebben om hun toepassing

te vinden. Voor ons als Java developers is het dan wel goed dat we daarvan op de hoogte blijven en al voorlopen op dit soort zaken. Het is ontzettend gaaf als we met veel energie en passie dit soort ontwikkelingen binnen bedrijven kunnen gaan integreren en daarmee business value leveren.

Om dat met veel energie en passie te kunnen doen, is het natuurlijk van belang dat je zelf als ontwikkelaar goed in je vel zit en gezond bent; iets waar we als GROUP9 veel focus op leggen. We hopen daarom ook een steentje bijgedragen te kunnen hebben aan de gezondheid en daarmee aan de energie van vele Java-developers met de gezonde Smoothies en appels op onze stand!

Op naar een energieke en gezonde JFall! ■

GROUP9
ELITE. JAVA. DEVELOPMENT.

Helm package manager:

Kubernetes deployments met Helm

Kubernetes wordt langzamerhand de standaard voor het deployen en draaien van container gebaseerde applicaties. Softwareprojecten leveren geen JAR of WAR files meer op, maar Docker containers. Of liever nog complete deployment packages van container en configuratie die je direct kunt uitrollen op een cluster. Helm is een package en deployment manager voor Kubernetes. Helm maakt het mogelijk om op een geavanceerde manier met Kubernetes deployment configuraties om te gaan door functies te bieden, zoals versionering en dependency management. In dit artikel bespreken we Helm. Ook bespreken we wat de voordelen zijn van Helm boven het gebruik maken de standaard Kuberenetes YAML configuraties.

Kubernetes YAML configuraties

Wanneer je iets wilt deployen op een Kubernetes cluster, bijvoorbeeld een Docker image of een service definitie, dan maak je gebruik van een YAML configuratie bestand**. Listing 1** toont een voorbeeld van zo'n deployment configuratie. De deployment configuratie bevat onder andere een verwijzing naar de repository en versie van de Docker image die je wilt gebruiken. Als je wijzigingen wilt maken in de configuratie, dan zul je dit bestand moeten aanpassen en deze opnieuw deployen. Werken met deze configuraties kan op termijn leiden tot veel bestanden en veel versies. Dit is lastig te beheren en te delen. Bij het uitrollen van een nieuwe deployment, is het bijvoorbeeld nodig om de versie nummer in de YAML aan te passen. Ook kun je met de standaard Kubernetes YAML geen afhankelijkheden definiëren tussen componenten. Tot slot is versie historie van deployments wel beschikbaar, maar niet op zo'n manier dat het mogelijk is om ook services, ingresses en andere Kubernetes artefacten onder te brengen onder 1 versie. Hierdoor is het niet mogelijk een volledige set artefacten te beheren en bijvoorbeeld een rollback uit te voeren op al deze componenten.

Helm

Helm biedt voor al deze zaken een oplossing. Helm maakt gebruik van Charts. Deze zijn vergelijkbaar met bijvoorbeeld Linux packages. Charts zijn yaml bestanden die template code bevatten (met behulp van Mustache). Dit maakt het mogelijk om specifieke waardes uit de configuratie te halen en apart te beheren.

Helm bestaat uit twee componenten: een commandline client en een tiller daemon in Kubernetes. De tiller wordt gebruikt om de status van de releases bij te houden, zodat verschillende cli clients kunnen worden gebruikt. Dit maakt het mogelijk om de status en historie van je deployments te beheren..

Aan de slag met Helm

Het ontdekken van de mogelijkheden van Helm gaat het snelst door ermee aan de slag te gaan.

Voorwaarden:

- Installeer Helm (**https://docs.helm.sh/using_helm/#installing-helm of via homebrew**)
- Zorg dat je toegang hebt tot een Kubernetes cluster, opties hiervoor zijn

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:1.0
        ports:
        - containerPort: 80
```

Listing 1

Erwin de Gier is Software Architect bij Trifork Amsterdam.

Minikube, het geïntegreerde Kubernetes cluster in de native Docker app (edge versie) of een managed Kubernetes bij AWS of Google Cloud.

We installeren de tiller doormiddel van het 'helm init' commando (**Listing 2**).

De chart die we gebruiken voor dit voorbeeld is beschikbaar in de volgende git repository: **https://github.com/erwindeg/k8s-helm**.

Voor dit voorbeeld maken we gebruik van een chart die bestaat uit een drie deployments en drie services:

- Reddis Master
- Reddis Slave
- PHP Frontend

De deployments binnen Kubernetes beschrijven de Docker images die worden gedeployed. De services worden gebruikt om de gedeployde instanties met elkaar te kunnen laten communiceren via een hostname:port mapping. Achter een enkele service kunnen meerdere instanties draaien met behulp van bijvoorbeeld een loadbalancer.

De repository bevat een folder "helm_chart_guestbook" met de volgende bestanden:

- templates/ guestbook-deployments.yaml
- templates/ guestbook-services.yaml
- Chart.yaml
- values.yaml

De template files zijn Kubernetes configuratie bestanden met Mustache templates. Dit maakt het mogelijk om de Kubernetes configuratie bestanden te generen. Zo kunnen ze worden hergebruikt en kan je bijvoorbeeld eenvoudig een versie nummer aanpassen. De waardes die in de template bestanden worden gebruikt, staan in de values.yaml. Daarnaast

```
//check of je verbonden bent met het correcte cluster, je kunt switchen met
config set context: https://kubernetes-v1-4.github.io/docs/user-
guide/kubectl/kubectl_config_set-//context/
$ kubectl config current-context
docker-for-desktop
//Installeer de Tiller daemon in je cluster, het zou kunnen dat je hier een
security warning krijgt. Voor meer informatie hierover:
https://github.//com/kubernetes/helm/blob/master/docs/securing_installation.md
$ helm init
Tiller (the Helm server-side component) has been installed into your
Kubernetes Cluster.
Happy Helming!
```

Listing 2

```
$ git clone https://github.com/erwindeg/k8s-helm
$ cd k8s-helm
$ helm install --name helm-gb-chart --namespace helm-gb
./helm_chart_guestbook
NAME:   helm-gb-chart
LAST DEPLOYED: Fri Apr 20 15:44:13 2018
NAMESPACE: helm-gb
STATUS: DEPLOYED
```

Listing 3

is het mogelijk om deze via cli variabelen mee te geven. Het Chart.yaml bestand bevat versie informatie over de chart zelf.

Om onze voorbeeld guestbook applicatie te deployen, hebben we de chart nodig. Daarom doen we eerst een 'git clone' van de repository met deze chart. Hierna gebruiken we 'helm install' om deze chart te installeren (**Listing 3**). We geven een release name mee (-name) en een namespace binnen Kubernetes (helm-gb). Namespaces binnen Kubernetes zorgen voor een logische scheiding van een cluster. De helm release name is uniek voor een cluster en wordt gebruikt om de historie bij te houden.

Met het commando 'kubectl get pods' kun je de gedeployde instances binnen een namespace zien. Een pod binnen Kubernetes is een set containers die samen op één host deployed wordt en zaken, zoals netwerk en storage, delen. Je kunt meerdere containers op één pod draaien, maar dan zul je verschil-

```
$ kubectl get pods --namespace helm-gb
NAME                              READY     STATUS    RESTARTS   AGE
frontend-788d978f9b-bwm5p         1/1       Running   0          33s
frontend-788d978f9b-hxmj6         1/1       Running   0          33s
frontend-788d978f9b-mpvk5         1/1       Running   0          33s
redis-master-7747787588-wp5cr     1/1       Running   0          33s
redis-slave-865486c9df-8mcx5      1/1       Running   0          33s
redis-slave-865486c9df-pnk5m      1/1       Running   0          33s

$ kubectl get svc --namespace helm-gb
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
frontend       NodePort    10.107.212.37    <none>        80:32080/TCP   5m
redis-master   ClusterIP   10.106.134.71    <none>        6379/TCP       5m
redis-slave    ClusterIP   10.108.147.227   <none>        6379/TCP       5m
```

Listing 4

HELM IS DE MISSENDE SCHAKEL IN HET BEHEREN VAN DE KUBERNETES DEPLOYMENTS

```
$ helm list
NAME            REVISION    UPDATED                     STATUS      CHART
NAMESPACE
helm-gb-chart   1           Fri Apr 20 15:44:13 2018    DEPLOYED
Guestbook-Chart-1.0.0   helm-gb

$ helm upgrade helm-gb-chart helm_chart_guestbook --set
frontend.image=harshals/gb-frontend:1.1
Release "helm-gb-chart" has been upgraded. Happy Helming!
# helm list
NAME            REVISION    UPDATED                     STATUS      CHART
NAMESPACE
helm-gb-chart   2           Fri Apr 20 16:05:51 2018    DEPLOYED
Guestbook-Chart-1.0.0   helm-gb
```

Listing 5

```
  replicas: 3
  image: "gcr.io/google_samples/gb-frontend:1.0"
  node_port: 32080
```

Listing 6

```
...
    spec:
      containers:
      - name: php-redis
        image: {{ .Values.frontend.image }}
...
```

Listing 7

lende porten moeten gebruiken. In dit geval hebben we drie instanties van de frontend container, die elk op een eigen pod draaien. Voor de frontend hebben we één service die gebruikt wordt om te communiceren met de drie pods (**Listing 4**). Een service kun je gebruiken om pods logisch te groeperen en deze via één specificatie ontsluiten.

Met 'helm list' krijgen we een overzicht van alle geïnstalleerde releases (**Listing 5**). De volgende stap is om een nieuwe versie van onze applicatie te deployen. We gebruiken hiervoor 'helm upgrade'. We geven de naam en versie van de frontend image op die we willen upgraden. In de values.yaml (**Listing 6**), is de waarde van de frontend.image variabele 1.0. In ons cli commando is dit 1.1. In het guestbook-deployments.yaml bestand gebruiken een template voor het vullen van de image waarde: {{ .Values.frontend.image }} (**Listing 7**). Op deze manier kan Helm de uiteindelijke Kubernetes yaml bestanden genereren met daarin een versie afkomstig van een values.yaml bestand of een versie nummer vanaf de cli. Dit laatste is bijvoorbeeld handig wanneer je in een build pipeline een nieuwe versie wilt deployen. Na het uitvoeren van het upgrade commando en 'helm list' kunnen we zien dat we nu op revisie 2 van deze release zitten.

```
$ helm history helm-gb-chart
REVISION    UPDATED                     STATUS      CHART
DESCRIPTION
1           Fri Apr 20 15:44:13 2018    SUPERSEDED  Guestbook-Chart-1.0.0
Install complete
2           Fri Apr 20 16:05:51 2018    DEPLOYED    Guestbook-Chart-1.0.0
Upgrade complete

$ helm rollback helm-gb-chart 1
Rollback was a success! Happy Helming!

$ helm history helm-gb-chart
REVISION    UPDATED                     STATUS      CHART
DESCRIPTION
1           Fri Apr 20 15:44:13 2018    SUPERSEDED  Guestbook-Chart-1.0.0
Install complete
2           Fri Apr 20 16:05:51 2018    SUPERSEDED  Guestbook-Chart-1.0.0
Upgrade complete
3           Mon Apr 23 08:49:15 2018    DEPLOYED    Guestbook-Chart-1.0.0
Rollback to 1
```

Listing 8

```
dependencies:
- name: nginx
  version: "1.2.3"
  repository: "https://example.com/charts"
```

Listing 9

Rollback

Het is ook mogelijk om releases terug te draaien. Helm houdt de release historie bij in het cluster, dus het terugdraaien van een release kan vanaf een andere machine dan de machine die gebruikt is om de release uit te rollen. Dit is handig als je bijvoorbeeld releases automatisch deployed in een build pipeline, maar dit handmatig wenst terug te draaien vanaf je ontwikkelomgeving.

In **Listing 8** laten we met 'helm history' zien dat we een install en een upgrade hebben gedaan van onze guestbook applicatie. Met 'helm rollback' kunnen we een rollback uitvoeren naar een eerdere revisie. Deze komt vervolgens in de lijst te staan met een nieuw revisienummer.

Helm Chart repository

Helm charts kun je delen door de Chart bestanden te delen (bijvoorbeeld via git). Dit is vergelijkbaar met het delen van je broncode of Dockerfiles. Als je je Charts wilt verspreiden of ervan gebruik wilt maken in andere Charts, dan wil je waarschijnlijk een geversioneerd pakket opleveren (vergelijk jar file of Docker image). Dit kan door gebruik te maken van een Helm Chart repository. Een Chart repository is een HTTP server met een index.yaml en één of meer gepackagede Charts. Een package kun je maken met het 'helm package' commando. Dit resulteert in een .tgz bestand. Hierin zitten de charts en values.yml bestanden. De charts kunnen refereren naar docker images. Deze zitten niet in de package, maar worden tijdens deployment gedownload in je Kubernetes cluster. Je kunt een lokale Chart repository draaien met "helm serve –repo-path ./charts".

De officiële Chart repository wordt onderhouden in **https://github.com/kubernetes/charts**. Deze repository bevat een veelvoud aan charts, bijvoorbeeld voor Sonarqube, MySQL, Sonatype Nexus, etc. Dit maakt het gemakkelijk om deze via helm in je Kubernetes cluster te deployen. Zo installeert het commando "helm install stable/sonarqube" bijvoorbeeld een instantie van Sonarqube.

Dependencies

Het is mogelijk om afhankelijkheden tussen specifieke Charts te definiëren. Ze kun je meerdere Charts deployen met één commando en kan je afdwingen dat een bepaalde versie van een service beschikbaar is in je cluster. Dependencies definieer je in een requirements.yaml bestand. Dit is vergelijkbaar met bijvoorbeeld een Maven pom.xml bestand voor het beschrijven van Java dependencies. **Listing 9** toont een voorbeeld van een dependency declaratie.

Conclusie

Helm is een package manager voor Kubernetes deployment configuraties. Vergelijkbaar met Maven voor Java of npm voor Javascript. Door gebruik te maken van Mustache templates en een bestand met variabelen genereert Helm de Kubernetes deployment configuraties voor je. Binnen ons project gebruiken we Helm voor versionering. De mogelijkheden van Helm maken het zeer geschikt om steeds weer nieuwe versies te deployen vanuit een build pipeline. Ook het is het eenvoudig om de versie historie in te zien. Dit maakt de deployments naar Kubernetes een stuk betrouwbaarder.

Doordat je charts kunt templaten, kunnen we onze deployment configuraties ook veel eenvoudiger generiek beschrijven. Zo maken we gebruik van één Chart voor al onze microservices. De values.yml bevat de microservice specifieke waardes.

Voor ons was Helm de missende schakel in het beheren van de Kubernetes deployments. Er zijn uiteraard alternatieven, zo is er een Kubernetes plug-in voor Terraform. Deze gebruikt echter een oudere API versie, waardoor niet alle functies beschikbaar zijn. Meer informatie over Helm kan je vinden op de officiële website: **https://helm.sh**. ■

HELM HOUDT DE RELEASE HISTORIE BIJ IN HET CLUSTER

Bouwstraat

Voor open source projecten

We gebruiken op ons werk allemaal wel een vorm van een bouwstraat. Een vaak gemaakte keuze hiervoor is Jenkins voor het aansturen van het proces en SonarQube voor de codekwaliteit. Hoe mooi zou het zijn als je dat ook voor je eigen open source projecten zou kunnen realiseren? Thuis een eigen bouwstraat opzetten kan, maar daar heb je wel hardware voor nodig. Wat nou als je deze functionaliteit elders zou laat hosten en ook nog eens volledig gratis?

Ondanks het feit dat velen van ons op het werk gewend zijn geraakt aan een bouwstraat, wordt dit vaak niet toegepast als het om open source hobbyprojecten gaat. Dat is jammer, want het is helemaal niet zo moeilijk om snel en gratis een werkende bouwstraat voor je projecten op te zetten.

Waarom build automation?

Er zijn een aantal problemen die je met build automation op kunt lossen. Ten eerste: inconsistente builds. Omdat het op jouw machine werkt, betekent dat nog niet automatisch dat het ook op een andere machine werkt. En we willen het *'we ain't shipping your machine*' natuurlijk voorkomen. Ten tweede: falende unit tests. Het komt weleens voor dat een ontwikkelaar vergeet om alle unit tests te draaien, voordat codewijzigingen worden gepushed naar source control. Combineer dit met een tool die jouw code analyseert op veel voorkomende fouten en je hebt een mooie basis waarmee je de kwaliteit van jouw code kan controleren en bewaken.

Wat hebben we allemaal nodig voor een build straat voor jouw open source?

- Repository om jouw source code onder te brengen;
- Dienst om source code automatisch te bouwen;
- Dienst, die de kwaliteit van de source code onder de loep neemt.

Laten we beginnen met het vinden van een geschikte repository.

De tools uitzoeken

Om te starten hebben we een dus repository nodig voor de source code. Er zijn hier verschillende mogelijkheden voor. Zo zijn er bijvoorbeeld de volgende websites, die deze dienst aanbieden: AWS CodeCommit, Bitbucket, GitHub en Gitlab. Omdat AWS CodeCommit alleen privé repositories host, valt deze gelijk al af. Bitbucket biedt ook het bouwen van jouw source code aan, maar dan ben je echter wel beperkt tot 50 'build' minuten per maand. Gitlab ondersteunt Continuous Integration middels hun eigen CI/CD integratie. Voor de gratis variant is de feature set echter erg beperkt.

Mijn voorkeur gaat uit naar GitHub. Maar waarom?

- GitHub zet zich in voor open source. In het kort komt het erop neer dat alles wat je in een publiekelijk toegankelijke repository plaatst, door anderen ingezien mag worden. Ook mag elke gebruiker een "fork" maken van jouw gedeelde repository. Voor een completer beeld van hun beleid, raad ik je ten sterkste aan de "terms of service" door te lezen;
- Veel succesvolle open source projecten worden gehost op GitHub. Denk hierbij bijvoorbeeld aan: Spring framework, Mockito en Kotlin;
- Door gebruik te maken van GitHub als jouw open source repository, kun je op een gemakkelijke manier gebruik maken van een aantal tools, die jouw project kunnen voorzien van build automation functionaliteit en codekwaliteit checks. Als bijko-

Mark Hendriks:
Chapter lead, ELK goeroe/lover/adopt, JTech, trainer Ordina Academy

Figuur 1: Overzicht van de werking van de automatische bouwstraat

mend voordeel kunnen deze tools - indien ingezet voor open source projecten - gratis gebruikt worden.

Het volgende waar we naar op zoek moeten, is een dienst om onze source code automatisch te bouwen. Als je op GitHub naar de marketplace gaat, is daar een hele verzameling diensten voor jouw open source repository te vinden. Kijk je vervolgens onder 'Continuous integration', dan zijn daar op het moment van schrijven negen verschillende diensten beschikbaar, die dienst kunnen doen als build automation service.

Tot nu toe heb ik van deze negen allen gebruik gemaakt van travis-ci en circleci. Mijn ervaring leert dat travis-ci gemakkelijk op te zetten is, vandaar dat ik voor de voorbeelden in dit artikel dan ook van travis-ci gebruik zal maken. Dit wil uiteraard niet zeggen dat de andere mogelijkheden minderwaardig zijn of dat travis-ci de beste op de markt is.

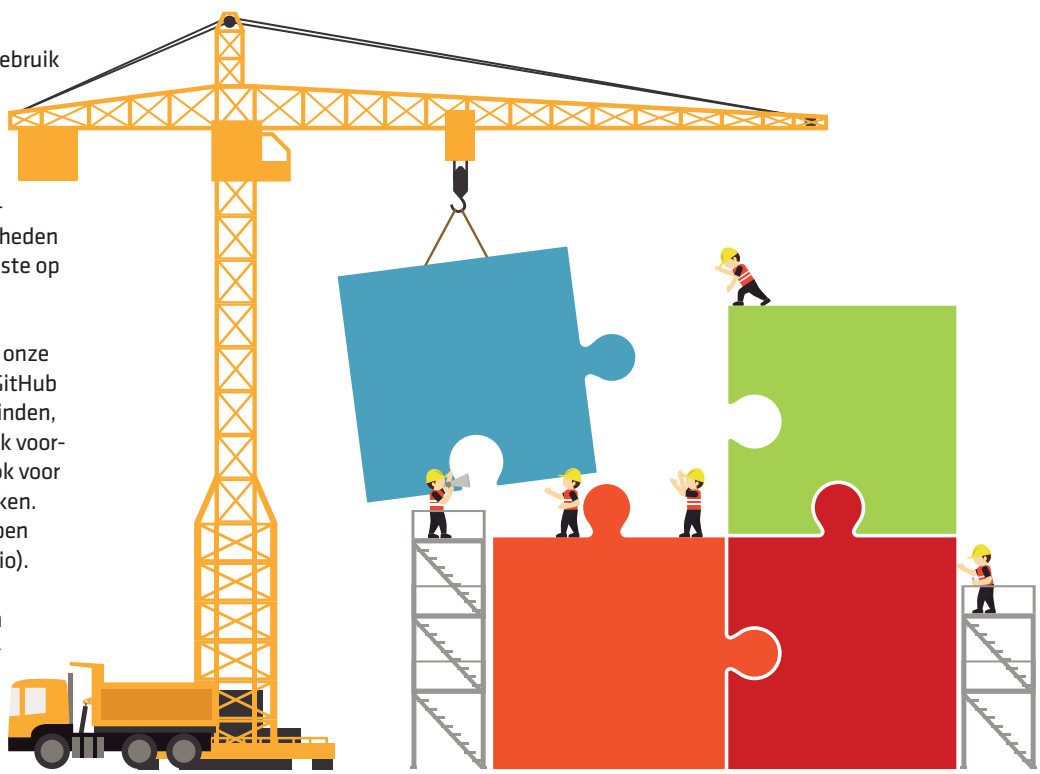
Natuurlijk willen wij ook de kwaliteit van onze source code kunnen waarborgen. Op de GitHub marketplace zijn een aantal services te vinden, die hiervoor in te zetten zijn. Aangezien ik vooral ervaring met SonarQube heb, zou ik ook voor mijn open source projecten willen gebruiken. Gelukkig is ook deze dienst gratis voor open source, middels SonarCloud (sonarcloud.io).

Nu wij voor ons open source project een selectie van de benodigde services hebben gemaakt, kunnen wij beginnen met het opzetten van onze eigen 'gratis' build straat.

Het opzetten van je project

Mocht je nog geen GitHub-account hebben, dan is dit het moment om het aan te maken. Mocht je niet weten hoe je jouw source code op GitHub krijgt, dan verwijs ik je graag naar hun documentatie.

De source code van dit voorbeeld is te vinden op mijn GitHub repository (**github.com/rovingeye/javamag-opensource-bouwstraat**). Zodra jouw code op GitHub staat, kunnen we travis-ci aan jouw repo koppelen.



Versie 1

```
language: java

jdk:
  - oraclejdk8
```

Versie 2

```
addons:
  sonarcloud:
    organization: "rovingeye-github"

cache:
  directories:
    - '$HOME/.m2/repository'
    - '$HOME/.sonar/cache'

script:
  - mvn clean org.jacoco:jacoco-maven-plugin:prepare-agent install sonar:sonar
```

Figuur 2: Versie 1 en 2 van .travis.yml

Ga naar **www.travis-ci.org** en log in met jouw GitHub credentials. Geef tijdens het aanmelden aan dat het open source betreft. Zodra je dit proces hebt doorlopen kun je een repository naar keuze aan travis-ci hangen. Dit doe je door op de '+' linksboven te klikken. Vervolgens zie je al jouw openbare repositories verschijnen. Is dit niet het geval, druk dan op de 'Sync account' knop links op het scherm. Kies jouw betreffende repository door op de checkbox te klikken.

Vervolgens moeten we een extra bestand aan het project toevoegen. Maak in de root van jouw project een bestand aan met de naam '.travis.yml'. Dit bestand zal travis-ci vertellen wat het allemaal moet doen. In **Figuur 2** zie de eerste versie van het '.travis.yml' bestand. Om te beginnen, vertellen we travis-ci alleen dat het de taal 'java' betreft en dat we onze source code willen bouwen met de Oracle JDK 8. Als travis-ci een 'pom.xml' in het project vindt, zal er gebruik gemaakt worden van Maven. Mocht het project een mvnw wrapper bevatten, dan zal deze gebruikt worden. Vindt travis-ci een 'build.gradle' bestand, dan zal Gradle gebruikt worden. Mocht je ANT willen gebruiken, dan zal dat wat extra configuratie vereisen. Dit is in de documentatie van travis-ci terug te vinden.

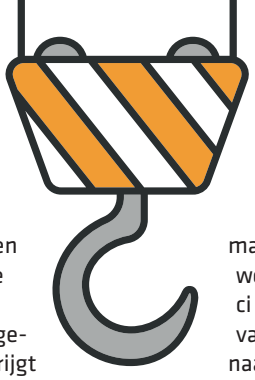
Nu we travis-ci hebben ingesteld en het project van de benodigde configuratie hebben voorzien, kunnen we de source code naar Github pushen. Na enige tijd zal travis-ci doorhebben dat er nieuwe source code beschikbaar is en beginnen met het bouwen van jouw project. Hoe lang dit duurt hangt af of er resources beschikbaar zijn om te kunnen starten. Omdat dit de gratis variant betreft, kan het zijn dat je even geduld moet hebben. Het resultaat van de bouw poging is terug te vinden op het dashboard.

Als alles goed is verlopen, zal de build groen zijn. Mocht deze rood zijn, dan valt er in de logs terug te lezen waar het mis is gegaan. De volgende stap is het koppelen van de automatische build aan SonarCloud.

Sonarcloud koppelen

Ga naar de website van SonarCloud (**sonarcloud.io**) en log in met jouw Github credentials. Geef ook hier bij het aanmelden aan dat het open source betreft. Om de automatische build te kunnen koppelen aan jouw SonarCloud hebben we de naam van de 'organization' nodig. Deze is rechtsboven achter het woord 'key' te vinden. In mijn geval is dit 'rovingeye-github'. Tevens hebben we een token nodig om daadwerkelijk te kunnen verbinden met SonarCloud. Druk rechtsboven op '+' en klik op 'Analyze new

ER ZIJN EEN AANTAL PROBLEMEN DIE JE MET BUILD AUTOMATION OP KUNT LOSSEN: INCONSISTENTE BUILDS, FALENDE UNIT TESTS EN SLECHTE CODEKWALITEIT



project'. Volg de wizard en geef een naam voor jouw token op. Voor de overzichtelijkheid raad ik aan om hier de naam van jouw project te gebruiken. Kopieer de token die je krijgt voor het gemak naar een tekstbestand.

Vervolgens moeten we deze token aan travis-ci toevoegen. Klik op het dashboard op jouw project. Klik rechts boven op 'More options' en vervolgens op 'Settings'. Onder het kopje 'Environment Variables' kunnen we de token toevoegen. Geef als naam 'SONAR_TOKEN' op en kopieer jouw token in het 'value' veld. Sla deze vervolgens op door op 'Add' te klikken.

Het laatste dat er moet gebeuren, is het eerder aangemaakte '.travis.yml' bestand aan te passen. Wij moeten travis-ci namelijk vertellen dat SonarCloud aangeroepen dient te worden. **Figuur 2** laat tevens versie 2 van het '.travis.yml' bestand zien. Door gebruik te maken van de SonarCloud add-on van travis-ci, hoeven wij geen credentials op te geven. De eerder door ons toegevoegde 'SONAR_TOKEN' environment Variable zal gebruikt worden voor de authenticatie. Wel moet je de organization, die binnen SonarCloud staat gedefinieerd, opgeven.

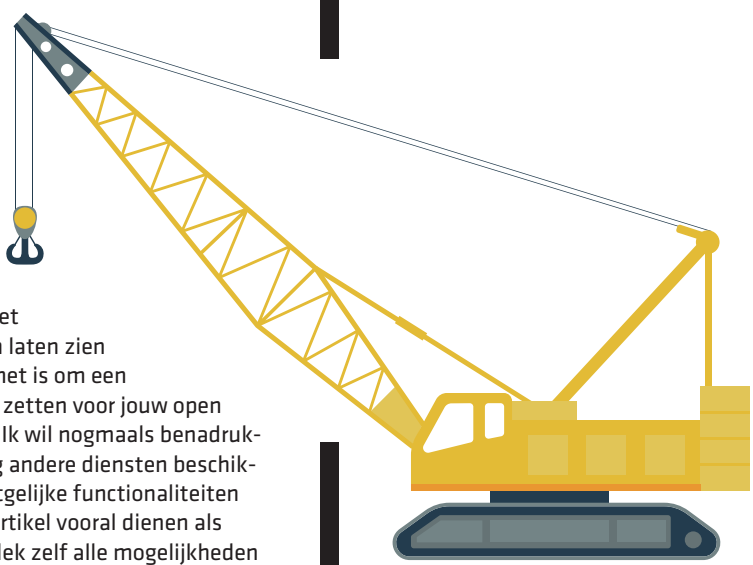
Onder 'script' geven we het Maven commando mee, dat ervoor zorgt dat SonarCloud voorzien wordt van alle gegevens om jouw project te analyseren. Push de nieuwe configuratie opnieuw naar GitHub. Zodra de automatische build in travis-ci klaar is, zal ook de analyse van SonarCloud op het dashboard van SonarCloud te vinden zijn.

Badges

Om het geheel af te maken, zullen we nu de zogeheten badges toevoegen aan de 'README.md' van ons project. Mocht dit bestand nog niet in de root van jouw project staan, maak deze dan aan. Als eerste zullen we de build status badge van travis-ci toevoegen. Ga naar het dashboard van jouw project. Rechtsboven staat naast de naam van jouw project een badge. Klik hierop. Kies vervolgens uit het tweede dropdown menu voor 'Markdown'. Kopieer de getoonde tekst en plak deze bovenin jouw 'README.md' bestand in je project.

Om de quality gate badge toe te voegen, gaan we naar de projectpagina in SonarCloud. Rechtsonder staat de knop 'Get project badges'. Als je hierop klikt, krijg je een keuzemenu. De standaardselectie is precies wat we nodig hebben. Kopieer ook hier de tekst. Deze badge behoeft wat meer werk. Plak de tekst onder de eerder geplakte tekst in jouw 'README.md' bestand. Gebruik vervolgens de markdown notatie van de travis-ci badge als template. Vergeet niet de tweede URL te vervangen door de URL die naar jouw projectanalyse verwijst. In **Figuur 3** zie je de uitwerking.

Als je deze wijzigingen naar GitHub pushed, zullen de badges op jouw GitHub repository pagina verschijnen.



Afsluitend

Hopelijk heb ik met dit artikel kunnen laten zien hoe gemakkelijk het is om een build straat op te zetten voor jouw open source projecten. Ik wil nogmaals benadrukken dat er genoeg andere diensten beschikbaar zijn die soortgelijke functionaliteiten bieden. Laat dit artikel vooral dienen als inspiratie en ontdek zelf alle mogelijkheden die er te vinden zijn. ■

```
[![Build Status](https://travis-ci.org/rovingeye/javamag-opensource-bouwstraat.svg?branch=master)]
  (https://travis-ci.org/rovingeye/javamag-opensource-bouwstraat)
[![Quality Gate](https://sonarcloud.io/api/project_badges/measure?project=nl
  .markhendriks%3Ajavamag-opensource-bouwstraat&metric=alert_status)](https://sonarcloud.io/dashboard?id=nl
  .markhendriks%3Ajavamag-opensource-bouwstraat)
```

Figuur 3: Uitwerking van badge Markdown en resulterende badges.

Google Assistant

Een voice adventure

Sinds een half jaar heb ik een Google Home in de woonkamer staan. Ondertussen ben ik redelijk bekend met de mogelijkheden, die out-of-the-box geboden worden. Het is dus hoog tijd om eens zelf functionaliteit toe te gaan voegen. Dit heb ik gedaan in de vorm van een 'voice adventure'. Jullie zijn vast wel bekend met het fenomeen text adventure, waarbij je tekst gebruikt om de spelwereld te manipuleren. In het geval van een voice adventure gebruik je dus je stem in plaats van tekst.

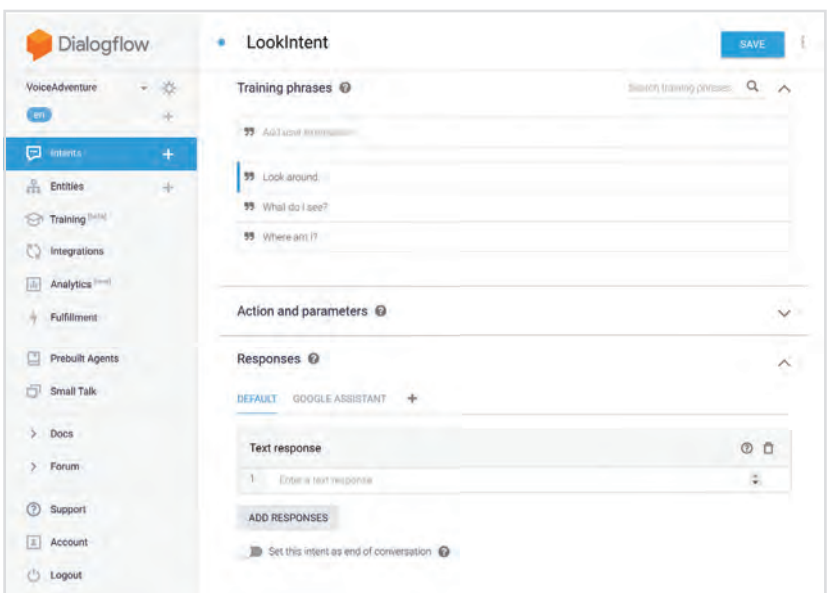
De Google Home is één van de slimme speakers van Google. Het brein van deze speaker is de Google Assistent. Dit is ongeveer dezelfde assistent die je aantreft op moderne Androidtelefoons. Tevens is de Google Assistant beschikbaar voor iPhone in de App Store. Er zijn wat kleine verschillen, maar deze zijn niet relevant voor dit artikel. Het spel werkt dan ook op alle smaken.

Een uitbreiding op de Google Assistent wordt een Assistant app genoemd. Deze kan je aanmaken op een platform genaamd **Actions on Google** (https://console.actions.google.com). Hier maak je een project aan en vervolgens dien je daar Actions aan toe te voegen. De makkelijkste manier om dit te doen, is gebruik maken van bestaande templates. Deze stellen je in staat om heel snel en zonder enige programmeerkennis een Action aan te maken. Aangezien dit een artikel is in het Java Magazine, is dit natuurlijk niet de optie die we gaan kiezen. Tevens biedt deze optie ook te weinig vrijheid om het einddoel te bereiken. Oftewel, je dient hier te kiezen voor een custom app. Er zijn weer meerdere mogelijkheden, maar voor het maken van een voice adventure kiezen we voor de Dialogflow optie. Dit is een simpele speech interaction builder, waarin je relatief eenvoudig een app in elkaar kunt klikken. Deze gebruiken we om een eerste opzet van het spel te maken.

Voordat we hier echter mee aan de slag gaan, moeten we eerst even kort schetsen hoe het spel in elkaar zit. Er zijn vijf locaties waartussen we kunnen bewegen. Op deze vijf plekken kan je interacteren met de omgeving door het geven van commando's. Je kunt hierbij denken aan 'look around', 'go north', 'pick up key' en 'use sword'.

Afbeelding 1 geeft een indruk van de Dialogflow interface. Aan de linkerkant zie je de twee belangrijkste concepten waarmee je kunt werken: **intents** en **entities**. Een intent is vergelijkbaar met een request reply paar. Het request is wat de gebruiker zegt en de reply is de reactie, die daarop volgt. Het request kan worden gespecificeerd onder het kopje **Training phrases**. Het reply staat dan onder het kopje **Response**. We negeren nu een aantal opties, maar dit is in essentie een **intent**. Een voorbeeld van een request is, vragen waar

Bouke Nijhuis
is Managing Consultant bij CINQ ICT.



Afbeelding 1: Het LookIntent in Dialogflow

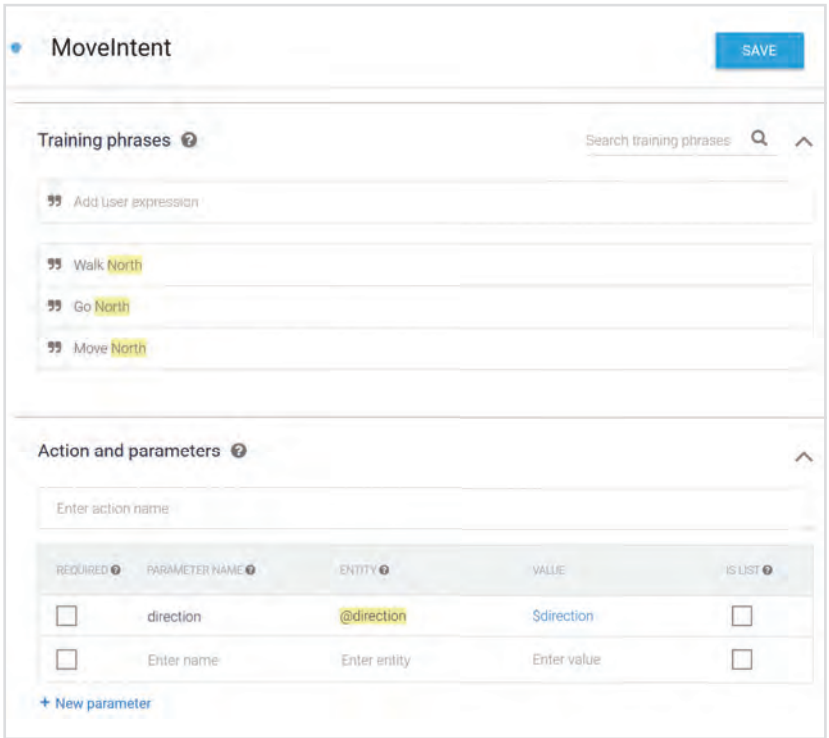
je bent en het antwoord zou vervolgens een beschrijving van de omgeving zijn.

Een entity is een variabele. Deze wordt hoofdzakelijk gebruikt om variabele waarden uit een request te halen. We gaan deze bijvoorbeeld gebruiken om de windrichting te bepalen in het commando 'go north'. In **Afbeelding 2** zie je hoe dit eruit ziet in Dialogflow. In dit geval wordt 'north' toegewezen aan de entity @direction. Je kunt dus ook 'go south' of 'go east' zeggen. Dit geeft een redelijk overzicht wat je allemaal kunt bereiken in Dialogflow. Niet alle concepten zijn behandeld, maar wel de belangrijkste.

Voor het spel moeten we een toestand (state) kunnen bijhouden. We moeten de locatie van de speler bijhouden en kunnen aanpassen. Hetzelfde geldt voor de items in de inventory. Deze gegevens kunnen we opslaan in een entity, maar het manipuleren van deze entities is vrij beperkt in Dialogflow. De gemakkelijkste manier om de locatie op te slaan, is door middel van een x- en y-coördinaat. Als de speler vervolgens beweegt, dan zullen we een coördinaat moeten aanpassen. Echter, dit soort simpele berekeningen kun je niet uitvoeren binnen deze speech interaction builder.

Het probleem dat Dialogflow niet krachtig genoeg is, komen we op veel meer plekken tegen. Je kunt hierbij denken aan onder andere het bepalen van de grenzen van de speelwereld en het bijhouden van de inventory. Eigenlijk in alle gevallen waarin je de toestand moet bijhouden, is custom Java-code nodig. Gelukkig is het niet moeilijk om eigen code toe te voegen aan een Assistant app.

Het is nu tijd om de scheidslijn tussen Dialogflow en custom code te bepalen. De eerste is erg goed in het vertalen van spraak naar tekst en het herkennen van een request. Hiervoor gaan we dus de speech interaction builder gebruiken. Voor alle andere zaken hebben we custom code nodig. Het toevoegen van custom code gaat door middel van een webservice. Eerder is uitgelegd dat een intent vergelijkbaar is met een request reply paar. In Dialogflow kun je een statisch reply definiëren. Je kunt het antwoord echter ook laten geven door een webservice. Dit wordt een fullfilment genoemd. Per intent kun je aangeven of je gebruik wilt maken van het statische antwoord of van de fullfilment. Bij de fullfilment kan je één URL definiëren waar de requests worden heen gestuurd aan



Afbeelding 2 - het MoveIntent in Dialogflow

```
public void handleRequest(InputStream inputStream, OutputStream
outputStream, Context context) throws IOException
{
   // handle request
}
```

Listing 1

de hand van een HTTP POST. Dit endpoint is vervolgens verantwoordelijk voor het geven van een antwoord.

Voor het afhandelen van deze requests zijn serverless function frameworks uiterst geschikt. Hierbij kun je denken aan AWS Lambdas, Google CloudFunctions & Azure Functions. Voor de implementatie van dit spel is er gebruik gemaakt van een AWS Lambda. Je kunt hier een simpele .jar uploaden en vervolgens aangeven welke methode er moet worden aangeroepen. Deze "main-methode" staat in **Listing 1**.

Het InputStream object bevat het request dat is verstuurd vanuit Dialogflow. Het antwoord wordt uiteindelijk gegeven via de OutputStream. Het Context object kan worden gebruikt om AWS-zaken aan te spreken. Hierbij kun je denken aan het uitvragen van informatie over de Lambda of het verkrijgen van een AWS Logger object. De requests komen binnen in de vorm van een JSON met daarin allerlei (context) informatie over het request.

De belangrijkste velden hierin zijn:

- **resolvedQuery**: het exacte commando van de gebruiker.
- **intentName**: de naam van het herkende intent (bijvoorbeeld een LookIntent, zoals in **Afbeelding 1**).
- **parameters**: ingevulde entities (bijvoorbeeld 'north' als windrichting in **Afbeelding 2**).
- **contexts**: opgeslagen gegevens over voorgaande interacties.

In de praktijk maak je geen gebruik van de resolvedQuery, maar handel je alles af op basis van de herkende intent in combinatie met de parameters en de contexts. Voor het maken van een Voice Adventure heb je ongeveer tien verschillende intents nodig. De meest sprekende voorbeelden zijn 'bewegen' en 'gebruiken'. Deze twee intents zullen verderop in detail worden uitgewerkt, maar voor dit mogelijk is, moet er eerst dieper worden ingegaan op de spelwereld.

De spelwereld bestaat uit locaties met daarin voorwerpen. Elke locatie heeft een positie en een omschrijving. De positie is nodig voor het bepalen van aan elkaar grenzende locaties. De omschrijving wordt gebruikt om de omgeving te beschrijven. De voorwerpen zijn iets complexer. Deze hebben naast een locatie en beschrijving ook een indicator of ze op te pakken zijn. Tot slot is er nog een datastructuur, die aangeeft welke voorwerpen op elkaar gebruikt kunnen worden en wat de effecten daarvan zijn.

De eerste intent die we in detail gaan bespreken, is de **MoveIntent**. In **Afbeelding 2** zie je met welke commando's deze intentie wordt getriggerd. Wanneer je 'move' of 'go' combineert met een windrichting, dan herkent Dialogflow dit als een MoveIntent. Alles wat hierop lijkt, zal als een MoveIntent worden geïnterpreteerd. Zo zal bijvoorbeeld 'run south' ook als een MoveIntent worden opgepakt. Als gevolg hiervan wordt er een request naar het ingestelde endpoint gestuurd. Een voorbeeld van een versimpeld request zie je in **Listing 2**.

Het eerste wat de software doet, is achterhalen om welk intent het gaat. In dit geval is het een MoveIntent en wordt de zogenoemde MoveHandler class gebruikt om het request af te handelen. Voor elke intent is er een bijpassende handler class. Deze is dan verantwoordelijk voor het bijwerken van de toestand (state) en het formuleren van een antwoord.

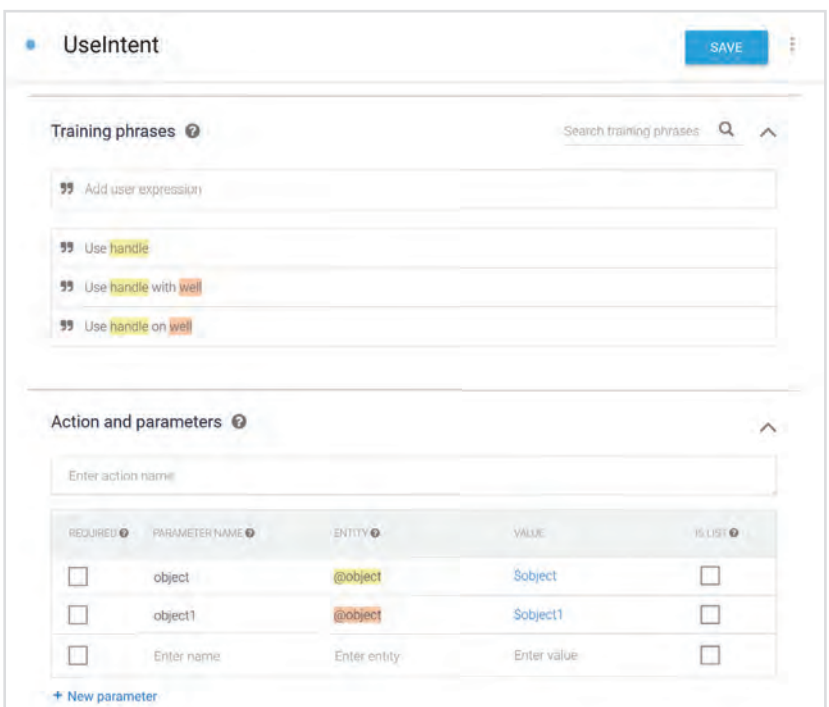
In het geval van het bovenstaande request zal de MoveHandler allereerst kijken of de verplaatsing mogelijk is. Hiervoor wordt eerst gekeken naar de huidige locatie (van de speler) en de gekozen richting. Als de verplaatsing mogelijk is, dan wordt de huidige locatie aangepast en krijgt de speler een beschrijving van de nieuwe locatie. Als dit niet mogelijk is,

```
{
 "result": {
   "resolvedQuery": "move north",
   "parameters": {
     "direction": "N"
   },
   "contexts": [],
   "metadata": {
     "intentName": "MoveIntent"
   }
 }
}
```

Listing 2

```
{
 "speech": "You are standing in front of a big castle.",
 "displayText": "You are standing in front of a big castle.",
 "contextOut": [
   {
     "name": "state",
     "parameters": {
       "posx": 0,
       "posy": 1
     }
   }
 ]
}
```

Listing 3



Afbeelding 3: het UseIntent in Dialogflow

dan krijgt hij hier een melding van (en wordt de huidige locatie natuurlijk niet aangepast). Het bijbehorende (versimpelde) reply staat in **Listing 3**.

De waarde van de attributen **speech** en **displayText** zijn gelijk. Dit houdt in dat je het spel kunt spelen met voice- als text-interactie. De toestand wordt bijgehouden in een context(Out) object, genaamd state. Hierin zie je de locatie van de speler als een coördinaat. Deze was in het request niet gedefinieerd en wordt default vervolgens op (0, 0)gezet. Door een succesvolle beweging naar het noorden is het (0, 1) geworden.

Een ander interessant intent is de **UseIntent**. Deze wordt gebruikt wanneer er voorwerpen op elkaar worden gebruikt. Een voorbeeld hiervan is het zwaard op de trol gebruiken door middel van'use sword on troll'. In dit geval zijn er dus twee parameters (namelijk sword en troll). In **Afbeelding 3** zie je hoe dit werkt in Dialogflow. Er zijn dan twee parameters van de entiteit @object gemaakt.

Deze twee (ingevulde) parameters zie je vervolgens weer terug in het versimpelde request in **Listing 4**. Wanneer je dit request vergelijkt met het vorige request in **Listing 2,** dan zie je dat nu het contexts-veld ook is gevuld. De inhoud lijkt erg op de inhoud van het veld contextOut in **Listing 3**. Het is wederom het context object waarin de state wordt bijgehouden. Er zijn twee nieuwe attributen bij gekomen, namelijk inventory en removedItems. De eerste houdt bij welke objecten de speler heeft opgepakt en de tweede bevat de objecten, die de speler heeft gebruikt in het spel, maar nu niet meer bruikbaar zijn.

Wanneer er een UseIntent binnenkomt, dan wordt er eerst gekeken of beide objecten gevuld zijn. Als dit niet het geval is dan krijgt de speler een melding dat de actie niet kan worden uitgevoerd. Vervolgens wordt er gekeken of beide items op elkaar gebruikt kunnen worden. Hiervoor worden een aantal controles uitgevoerd. Daarbij kun je denken aan zaken als: heeft de speler het object ooit opgepakt, staat de speler op de juiste locatie en heeft het zin om beide objecten op elkaar te gebruiken. Als alle controles positief zijn, dan wordt de actie uitgevoerd en heeft de speler een nieuw object verkregen. Er wordt dan een nieuw object toegevoegd aan het veld inventory en het gebruikte object verhuist van inventory naar removedItems. In alle andere gevallen krijgt de gebruiker een melding met daarin de reden waarom de actie niet mogelijk is.

```
{
 "result": {
   "resolvedQuery": "use sword on troll",
   "parameters": {
     "object": "sword",
     "object1": "troll"
   },
   "contexts": [
     {
       "name": "state",
       "parameters": {
         "posx": -1,
         "posy": 0,
         "inventory": ["SWORD"],
         "removedItems": ["HANDLE"]
       }
     }
   ],
   "metadata": {
     "intentName": "UseIntent"
   }
 }
}
```

Listing 4

Tot zover de uitleg over het maken van een Assistant app door middel van Dialogflow en custom Java-code. Tot slot wil ik jullie graag uitnodigen om het spel te spelen. Dit doe je door de QR-code in **Afbeelding 4** te scannen. Het kan gespeeld worden met zowel tekst- als spraakinvoer. Uiteraard is de laatste optie de leukste manier en dit doe je door rechts onderin op de microfoon te klikken. Mocht je geen idee hebben hoe te beginnen, dan kun je simpelweg om hulp vragen in het spel. Veel plezier met spelen!

Mocht je na het spelen geïnteresseerd zijn in de bijbehorende Java-code, dan staat er in de referenties een URL naar het GitHub project. ■

Afbeelding 4: QR code

REFERENTIES

Actions on Google: **https://console.actions.google.com**
Dialogflow: **https://console.dialogflow.com/api-client/**
WebDemo: **https://bot.dialogflow.com/4df6f607-bedc-4393-bd59-7485757110ba**
Github: **https://github.com/BoukeNijhuis/google-home-demo**

Java Roadmap

Hoe ziet de toekomst van Java eruit?

De Java SE 10 Development Kit is pas net uitgebracht door Oracle en Java SE 11 staat alweer bijna klaar. Sinds Java SE 10 heeft Oracle een 6 maanden releaseschema aangekondigd voor de standaardeditie van Java. In zijn blog heeft Mark Reinhold een nieuw voorstel gedaan voor versionering. Hij stelt voor om vanaf het moment van de 6 maanden release (vanaf versie 10) over te gaan op een YY.MM release versie. Dit zou Java SE 11 versie 18.9 maken, omdat het uit moet komen in september 2018. Er is een hoop te doen rondom deze manier van versioneren en dit kan dus nog veranderen. Nu zie je vaak de volgende notatie staan "Java SE 11 (18.9)". De tijd zal het leren.

Tot nu toe zijn pas een paar nieuwe features aangekondigd voor Java SE 11 (18.9). Ook gaan een paar features verloren in deze versie, zoals CORBA en Java EE (recentelijk hernoemd naar Jakarta EE, hierover later meer) modules. Ook zal JavaFX verwijderd worden.

Op dit moment zitten we op Java SE 10 (18.3) en de laatste versie van de Enterprise Edition is uitgebracht (Java EE 8).

Java SE 11 zal een zogenaamde LTS (Long Term Support) release zijn. Java SE 11 zal de hoogste level van ondersteuning genieten van Oracle tot september 2023 en aanvullende ondersteuning, zoals patches en beveiligingswaarschuwingen, tot 2026. Het is de planning van Oracle om elke drie jaar een LTS release uit te brengen. Dat betekent dus dat elke feature release, zoals Java SE 9 en Java SE 10, maar een half jaar ondersteuning van Oracle krijgen, omdat dit short-term releases zijn.

In JDK 11 zijn tot nu toe drie nieuwe features gepland, maar er worden er wellicht meer verwacht. Dit wordt bevestigd op de JEP site voor Java SE 11.

De Epsilon "No-Op" Garbage collector, zoals beschreven in JEP 318, handelt wel geheugenallocatie af, maar implementeert geen terugvorderingsmechanisme. Zodra de Java Heap "vol" is, zal de JVM stoppen. De motivatie achter deze triviale no-op garbage collector ligt vooral op het gebied van testen, JVM Interface, geheugendruk en extreem kort lopende jobs.

Nieuwe features

De Local-Variable syntax for Lambda Parameters, zoals beschreven in JEP 323, is een van de nieuwe features gepland voor Java SE 11 en zal toestaan dat het woord 'var' wordt gebruikt bij het declareren van de formele parameters van impliciet getypeerde Lambda expressies. Sinds Java SE 10 is het mogelijk om var te gebruiken voor implicit typing van lokale variabelen, maar dit kan binnen Lambda expressies nog niet worden gebruikt. Deze JEP wil dat rechttrekken (zie **Listing 1**).

Deze twee statements zijn dan gelijk aan elkaar. In deze JEP is het nog niet toegestaan om expliciete declaratie en impliciet (var) door elkaar te gebruiken (zie **Listing 2**).

```
(x, y) -> x.process(y)          // implicitly typed lambda expression
(var x, var y) -> x.process(y)  // implicit typed lambda expression
```

Listing 1

```
(var x, y) -> x.process(y)
// Cannot mix 'var' and 'no var' in implicitly typed lambda expression
(var x, int y) -> x.process(y)
// Cannot mix 'var' and manifest types in explicitly typed lambda expression
```

Listing 2

Ivo Woltring is Software Architect en Codesmith bij Ordina JTech.

Edwin Derks is Solutions Architect en CodeSmith bij Ordina JTech en heeft naast Java een passie voor cloud-driven development en serverless architecture.

JEP 309 beschrijft een uitbreiding op het class-file formaat voor ondersteuning van een nieuwe constant-pool variant. Deze variant maakt het mogelijk om de creatie te delegeren naar een zogenaamde bootstrap methode. Deze JEP is vooral gericht op de JVM en zou taalontwerpers en compiler builders meer opties moeten geven voor expressiviteit en performance.

De Java EE features, die voor het gemak voor ontwikkelaars sinds Java SE 6 ook in de Standard Edition zitten, zoals JAX-WS (Java API voor XML-gebaseerde webservices), JAXB (XML Binding), JAF (Java Activation Framework) en Common annotations, zullen verwijderd worden. In Java SE 9 zijn ze deprecated verklaard om in Java SE 11 (18.9) verwijderd te worden.

De motivatie hierachter is dat het onderhouden van deze modules in twee Java versies lastig is, doordat de verschillende versies ook verschillende evoluties doormaken. Oracle zegt eigenlijk dat met twee onafhankelijke versies er geen noodzaak meer is om deze features in Java SE te hebben. Dit kan gevolgen hebben voor applicaties, die nu afhankelijk zijn van deze "out-of-the-box" ondersteuning van de EE features. De code zal niet meer compileren op Java SE 11 (18.9).

Volgens Oracle is er geen significante interesse meer voor het ontwikkelen van nieuwe applicaties met CORBA en zijn de kosten van het onderhouden daarvan niet langer rendabel. Ook hier is er een risico voor applicaties die een (sub-)set aan CORBA API's gebruiken, dat ze niet meer zullen werken als het gecompileerd is op Java SE 11+. Voor alle EE modules, die verwijderd worden, zijn ook third party oplossingen te vinden, maar niet voor CORBA. Meerdere keren nadenken dus voor applicaties, die CORBA gebruiken, als je wilt upgraden.

Java FX gaat uit de Standaard Editie van Java verwijderd worden, zodat het niet meer mee hoeft te draaien in het halfjaarlijkse release schema.

Ondertussen... bij Java EE

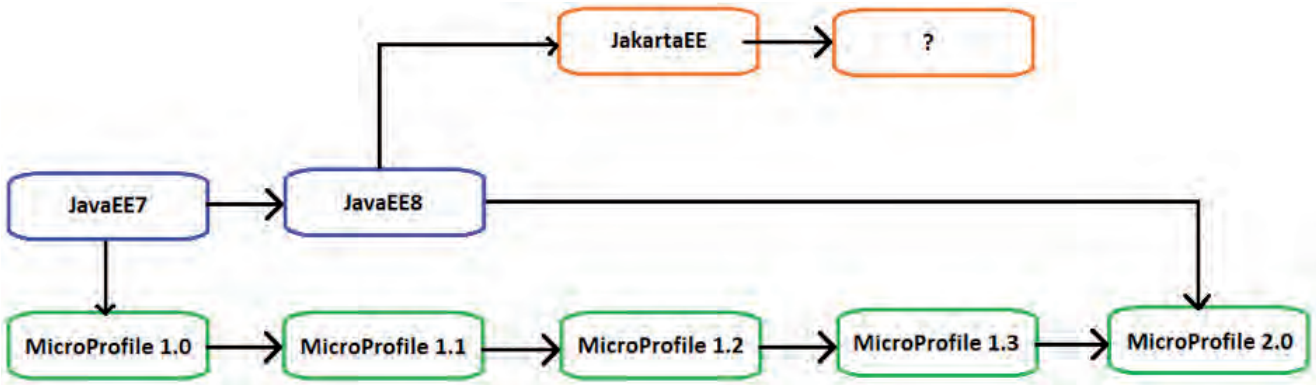
Jarenlang is de Java EE specificatie eigendom van Oracle geweest. Dat hoor je goed: geweest. Afgelopen najaar heeft Oracle de Java EE specificatie overgedragen aan de Eclipse Foundation, in dezelfde tijd dat Oracle eindelijk de definitieve en langverwachte versie van de Java EE 8 specificatie had uitgebracht.

De ontwikkeling van de Java EE specificatie maakte namelijk een moeilijke tijd door onder het bewind van Oracle. Dit had als reden dat de specificatie niet bijbleef met de ontwikkelingen van de enterprise softwaremarkt om Java EE heen. Hierdoor zag Oracle er waarschijnlijk geen heil meer in en wilden ze dit project onderbrengen bij een partij, die hier verandering in moet brengen.

Hiermee is de Java EE specificatie een open source project geworden. Men hoopt dat de community het project oppakt en deze in een hoger tempo doorontwikkelt naar moderne maatstaven voor enterprise software. Dit is helaas onder het bewind van Oracle niet gelukt, ondanks dat grote bedrijven als RedHat en IBM hier actief hun steentje aan bij hebben gedragen.

Een ander issue was de komst van Java SE 9 met zijn modulesysteem. Dit fenomeen heeft een grote impact op de adoptie van Java SE 9+ in de Java EE implementaties. Dit komt o.a. doordat de applicatieservers, die de Java EE specificatie implementeren vaak ook intern een eigen modularisatie toepassen om bijvoorbeeld snelle starttijden en lage memory footprints te realiseren. Tevens moeten ook alle referentie implementaties van de specificaties hierop aangepast worden. Ondersteuning voor Java SE 9 is daarom nog steeds niet volledig of optimaal.

Maar dat is nog niet alles: Oracle heeft dan wel afstand gedaan van de Java EE specificatie, maar niet van de naam. De community heeft de specificatie omgedoopt naar Jakarta EE, met als interne project naam: EE4J. De applicatieserver Glassfish (tevens gedoneerd en verder doorontwikkeld als Payara) blijft overigens (althans voorlopig) de referentie implementatie van de Jakarta EE specificatie.




```
if (obj instanceof Integer) {
   int intValue = ((Integer) obj).intValue();
   // use intValue
}
```

Listing 3

```
if (x matches Integer i) {
   // gebruik i hier
}
```

Listing 4

Naast Java EE en Jakarta EE is er ook nog het project MicroProfile om microservices mee te bouwen in Java EE stijl. Het huidige MicroProfile 1.3 is gebaseerd op Java EE 7 en vult met zijn specificaties het gat op, dat Java EE 7 open liet om naar moderne maatstaven microservices te bouwen. Versie 2.0 wordt uitgebracht in juni 2018 en bevat updates voor een Java EE 8 basis en gloednieuwe JSON-B specificatie.

Jakara EE heeft als doel om cloud native Java te supporten. In hoeverre dit project gaat doorpakken op zowel de bestaande Java EE als mogelijk de MicroProfile specificatie zullen we in de nabije toekomst zien.

Maar hoe nu verder?

Tot dit punt van het artikel zaten we nog redelijk op wat daadwerkelijk al gepland is voor de nabije toekomst en wat de concrete plannen zijn. Maar hoe nu verder? Een aantal zaken zitten al in de pipeline. Zo is er project

```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello World.</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

Listing 5

```
String html = `<html>
                 <body>
                   <p>Hello World.</p>
                 </body>
               </html>
               `;
```

Listing 6

Amber, dat als doel heeft om kleinere, op productiviteit georiënteerde Java-taalfuncties te verkennen, die zijn geaccepteerd als kandidaat JEP's. JEP 323 uit project Amber is ondertussen gepromoot naar feature van Java SE 11 (18.9).

Zo wordt in Project Amber onder andere gekeken naar **Pattern Matching** (JEP 305), **Raw String Literals** (JEP 326) en **Data Classes** (Nog geen JEP).

Bij Pattern Matching wordt onder andere bekeken of veel voorkomende constructies, (zoals in **Listing 3**) gemakkelijker gemaakt kunnen worden met een constructie als in **Listing 4**.

Bij **Raw String Literals** wordt gekeken of het gebruiken van String gemakkelijker kan.

Vervang bijvoorbeeld **Listing 5** door **Listing 6**.

Ook wordt hier gekeken of het dubbel escapen weg kan. Vervang:

```
"this".matches("\\w\\w\\w\\w");
```

Door:

```
"this".matches(`\w\w\w\w`);
```

Bij **Data Classes** wordt onderzocht of een stuk verbositeit van Java geëlimineerd kan worden door het introduceren van Data Classes. Het idee is om classes als **Listing 9** te vervangen door:

```
record Point(int x, int y) { }
```

Dit lijkt een no-brainer, waarbij de meeste Javanen zullen denken: "ja, doe maar!". Het is echter moeilijk om consensus te krijgen over zaken als muteerbaarheid, uitbreidbaarheid en constructors.

Als stukje onderhoud van de JVM is er project **Graal**. Graal biedt bijvoorbeeld ondersteuning voor het gebruik van een custom JIT (Just-in-Time) compiler. Zo'n JIT compiler is nu beschikbaar voor Java, maar eventueel ook voor Scala, Clojure of Kotlin. De Java JIT compiler is tevens geschreven in Java en niet in C++, zoals de conventionele C1 en C2 compiler (die we nu gebruiken).

Dus we krijgen een JIT Compiler, geschreven in Java, die we als zodanig kunnen ontwikkelen en onderhouden. Of een JIT Compiler nu geschreven is in Java of C++, het gaat om wat deze compiler runtime interpreteert naar machinecode. Daarom wil men vanaf nu een JIT Compiler in Java. De C1 en C2 zijn te oud en niet meer te onderhouden (zeker niet met C++). Met Java hopen ze hiermee een stap te zetten in de toekomstbestendigheid van de JVM.

Toekomstvisie

Brian Goetz is van mening dat Java op deze manier op de juiste weg is. Oracle loopt inderdaad niet voorop met het introduceren van nieuwe baanbrekende features, maar kijkt naar de ervaringen, die zijn opgedaan bij features in andere talen. Vooralsnog lijkt Java daarmee zijn kracht te behouden. Wij zijn zelf in ieder geval heel benieuwd naar de toekomst van Java en zullen deze ook op de voet blijven volgen. ■

```
final class Point {
   public final int x;
   public final int y;

   public Point(int x, int y) {
       this.x = x;
       this.y = y;
   }
   // implementaties van equals, hashCode, toString
   // verder niets
}
```

Listing 9

AFGELOPEN NAJAAR HEEFT ORACLE DE JAVA EE SPECIFICATIE OVERGEDRAGEN AAN DE ECLIPSE FOUNDATION

REFERENTIES

http://openjdk.java.net/projects/graal/
http://chrisseaton.com/truffleruby/jokerconf17/
https://mreinhold.org/blog/forward-faster
http://openjdk.java.net/jeps
https://jakarta.ee
https://projects.eclipse.org/projects/technology.microprofile/releases/microprofile-2.0
http://openjdk.java.net/projects/amber/
http://openjdk.java.net/projects/jdk/11/

React native

Wat is React Native en wat maakt het anders?

React Native is een JavaScript library voor het ontwikkelen van hybride mobiele applicaties. Dit project is ontwikkeld door Facebook tijdens een interne hackathon in 2013 en is publiek gemaakt in 2015. Het doel van het project was het ontwikkelen van mobiele applicaties die een native performance hebben met webtechnologieën. React Native was in eerste instantie met alleen iOS ondersteuning gestart, maar is sindsdien uitgegroeid tot een project met meer dan 59k sterren op Github.

Sinds de release van React Native verschenen ook veel interessante forks van het project, zoals ReactXP (cross platform development voor iOS, Android, Web en Windows 10) en React Native MacOS. In dit artikel wordt de basis van React en React Native geïntroduceerd. Alle code voorbeelden en de todo applicatie die zijn gemaakt voor dit artikel zijn te vinden op de volgende repository: **https://github.com/jdcas89/todoApp**.

React.js

Reactjs (**https://reactjs.org**) is een Javascript library voor het maken van Web UI's. De belangrijkste aspecten van React worden beschreven, aangezien React Native en React veel op elkaar lijken. Deze beschrijvingen gelden dus grotendeels voor zowel React als React Native.

React Nodes, de Virtual DOM en JSX

Een UI in React is een verzameling van React Nodes, deze Nodes zijn representaties van DOM elementen. De volledige representatie van de DOM heet de Virtual DOM. Met behulp van React algoritmes worden vergelijkingen gedaan met de echte DOM om selectief updates te kunnen voeren. React nodes kan je creëren met behulp van de helper functie createElement.

In **Listing 1** zie je hoe een lijst van twee elementen met behulp van de createElement functie wordt geschreven. We zien gelijk hoe inefficiënt dit kan zijn bij het schrijven van geneste elementen. Hiervoor kunnen we JSX gebruiken(JavaScript XML), een XML syntax extensie van ECMAScript die het mogelijk maakt om HTML en Javascript samen te schrijven (**https://facebook.github.io/jsx/**).

Om JSX te kunnen gebruiken, heb je een JavaScript preprocessor nodig die de JSX naar standard ECMAScript omzet. Deze preprocessors zijn nodig, omdat JSX niet ondersteund wordt door browsers. Een voorbeeld van dezelfde ul element uit **Lising 1** in JSX ziet er als in **Listing 2**.

Components

Webapplicaties gemaakt met React bestaan uit verschillende componenten. Een component bestaat uit meerdere React nodes en is een representatie van een deel van de UI. Er zijn twee soorten componenten: class componenten (stateful) en functionele componenten(stateless). Class componenten hebben een interne state, levenscyclus methodes en moeten de render functie aanroepen die je JSX bevat. Stateless compo-

Juan Castellanos is een gedreven software-ontwikkelaar met een passie voor het oplossen van problemen en het delen van zijn kennis met anderen en werkt bij Quintor Den Haag. In zijn vrije tijd coacht hij een developer groep en maakt hij muziek.

```
var liElement0 = React.createElement('li', { id: 'li0' }, 'zero');
var liElement1 = React.createElement('li', { id: 'li1' }, 'one');
var ul = React.createElement('ul', { id: 'ul' }, liElement0, liElement1);
ReactDOM.render(ul, document.getElementById('container'));
```

Listing 1: Renderen van React Nodes

```
const JSX = function () {
    return (
        <ul>
            <li>zero</li>
            <li>one</li>
        </ul>
    )
}

ReactDOM.render(<JSX />, document.getElementById('container'));
```

Listing 2: React met JSX

nenten hebben geen state en krijgen alles via props (**https://reactjs.org/docs/components-and-props.html**).

State

Om te weten wanneer React de DOM elementen opnieuw moeten renderen (weergeven), wordt de state van het bijbehorende component bijgehouden. De state is niets anders dan een object met eigenschappen die leeft binnen het component. Bij een verandering van deze eigenschappen wordt er selectief een re-render gedaan op het component en zijn child components. Het is belangrijk te weten dat de state van een component niet rechtstreeks veranderd kan worden. Dit gebeurd via de functie setState.

Props

Props zijn eigenschappen (properties) die je mee kan geven aan een component. Deze props kunnen benaderd worden via het props object in het component zelf. Een voorbeeld van hoe props gebruikt kunnen worden, is te zien op **Listing 3**.

Event handlers

Event handlers zijn functies die in componenten leven en worden meestal gebruikt om de state te manipuleren. Deze handlers kunnen ook doorgegeven worden via props.

Styling

CSS eigenschappen in React componenten kunnen declareren, net zoals in HTML elementen. In dat geval wordt className gebruikt voor CSS classes in plaats van class. Dit komt, omdat de class keyword in JavaScript gereserveerd is. Dit betekent dat libraries, zoals Bootstrap, gebruikt kunnen worden met React componenten. Een component met styling ziet uit als in **Listing 4**.

Wat is de kracht van React Native?

Ten eerste doe je al je ontwikkeling met JavaScript en React. Dit betekent dat het maken van een mobiele applicatie met React Native niet veel verschilt als je al bekend bent met webapplicatie ontwikkeling met React. Het grootste deel van je code is ook te gebruiken voor react webapplicaties, omdat weinig verschil is tussen React en React Native.

```
const StatelessComponentWithProps = (props) => (
    <div>
        <p>Component without any state, but with props {props.value} </p>
    </div>
);

<StatelessComponentWithProps value="this is a value in a prop" />
```

Listing 3: Props in React Native

```
const ReactComponent = () => {
    <div className="container-fluid">
        <h1 className="heading">Styled React Component</h1>
        <p className="lead">Paragraphs with style</p>
    </div>
}
```

Listing 4: React Native component

React Native gebruikt geen Webview. Alles wat je ontwikkelt met React Native heeft het gevoel en prestatie van een echte native applicatie. Dit gebeurt mede door react native bridge (zie **Listing 5**) die de JavaScript code verbindt met de overeenkomende native module. Ten slotte kan je native code en React Native code combineren als je dit nodig hebt of je eigen native bridges ontwikkelen voor je components.

Verschillen tussen React Native en React

Wat zijn de grootste verschillen tussen React Native en React?

- React Native maakt geen gebruik van HTML elements. De React Native library heeft React components die gekoppeld zijn aan de native API (zie **Tabel 1** op de volgende pagina).
- Styling gebeurt mede door style prop. Voor styling is ook conventie om deze in het component zelf te hebben voor volledige self-contained components.

Listing 5: React Native flow

HTML element	React Native component
div	View
p	Text
image	Image
input	TextInput

Tabel 1: Elementen

- React Native gebruikt flexbox (**https://css-tricks.com/snippets/css/a-guide-to-flexbox/**) als standard voor styling en een camelCase voor de "css" classes.

Development ervaring

Ontwikkeling in React Native is vergelijkbaar met webapplicatie ontwikkeling. Features zoals live-reload werken out-of-the-box,waardoor er bijna geen wachttijd is om je veranderingen te zien op je simulator of device. React Native biedt ook de mogelijkheid om je JavaScript code te debuggen met behulp van Chrome Developer tools. Verder kan je met React Native de inspector tool gebruiken om eigenschappen van je componenten in te zien.

Voordelen

- Native prestatie;
- Herbruikbaarheid van je codebase;
- Applicaties updaten zonder goedkeuring van app stores;
- Static Type checking met behulp van Flow (**https://flow.org/**);
- Gebruikt en ondersteunt door grote partijen;
- Debugging met chrome developer tools is mogelijk;
- Onderhoud mogelijkheden;
- Uitstekende documentatie.

Nadelen

- Je moet de basis van React kennen;
- Combinatie JavaScript en HTML (JSX);
- Je moet native code kunnen schrijven ten behoeve van de functionaliteit die je wilt;
- Het wordt niet ondersteund door de React Native API;
- State management voor React applicaties blijft een uitdaging. Hiervoor kan je een state management library, zoals Redux gebruiken(**https://redux.js.org/**);
- Best practices zijn soms moeilijk te volgen, omdat de library heel flexibel is en wordt regelmatig geüpdatet.

Hoe werkt het in de praktijk?

Door middel van een applicatie worden de aspecten van het werken met React Native in de praktijk in kaart gebracht. Deze applicatie is heel simpel en kan het volgende doen:

- Todos weergeven;
- Todos toevoegen;
- Todos markeren als gedaan;
- Todos verwijderen.

De requirements voor het installeren van de packages die je nodig hebt om aan de slag te gaan met React Native staan op de repository. De officiële handleiding staat kun je hier vinden: **https://goo.gl/TLBJr2**.

Listing 6: Component schets

Als je alles geïnstalleerd hebt, dan kan je met de React Native CLI een project starten met het volgende command:

```
react-native init todoApp.
```

Als eerste stap wordt een schets gemaakt van de component compositie van de applicatie (zie **Listing 7**). Het maken van een schets

```
export default class App extends component {
    render(){
        return(
            <View>
                <Text>Todos</Text>
                <TodoList />
                <CreateTodoModal />
            </View>
        )
    }
}
```

Listing 7: Initiële code voor de root component

```
constructor() {
  super();
  this.state = {
    todos: [],
    createModalVisible: false,
    newTodo: '',
    errorMessage: '',
    loading: true,
  };
}
```

Listing 8: State voor de root component

```
// → Modal component
toggleModal = () => { }
createTodo = (todoText) => { }

// → TodoList → TodoItem component
deleteTodo = (key) => { }
toggleDone = (key) => { }
```

Listing 9: Event handlers

helpt bij het schrijven van de componenten. In **Listing 6** zien we bijvoorbeeld dat de root component twee componenten bevat: (1) de List en (2) de CreateTodoModal component. Verder bestaat de List component uit meerdere TodoItem componenten. De basis voor de root component is te zien in **Listing 8**.

We kunnen nu beginnen met het maken van lege componenten voor TodoList , TodoItem en CreateTodoModal. Deze componenten hoeven geen state te hebben, omdat we alles via props gaan doorgeven. Voor de state kunnen we een initiële state definiëren voor de root component. Het is belangrijk om een lijst te hebben van de todos, de zichtbaarheid van de modal en een error message voor input validatie. De volledige applicatie state ziet uit als in **Listing 9**.

De volgende stap is om de event handlers van de app te definiëren (zie **Listing 10**). Deze event handlers - samen met de benodigde state - kunnen via props aan de betreffende componenten doorgeven worden.

Wanneer de componenten de benodigde props en state bevatten, kan je beginnen aan de implementatie van de event handlers en logica van de applicatie. Als voorbeeld wordt de TodoList (zie **Listing 11**) component toegelicht. Dit (functionele) component krijgt de state van de App component. Met behulp van de FlatList (**https://facebook.github.io/react-native/docs/flatlist.html**) component worden TodoItems gerenderd. Er wordt ook een tekst toegevoegd voor het geval er geen todos zijn. Verder worden de toggleDone en deleteTodo event handlers doorgegeven aan elke TodoItem.

Hoe de rest van de applicatie is afgerond, is op de repository te zien. De applicatie werkt op iOS en Android. Als we deze applicatie naar web moesten migreren, dan zouden we een groot percentage van de code zo over kunnen nemen.

```
export default TodoList = props => (
  <View style={styles.listContainer}>
    { props.todos.length === 0 &&
    <CustomText style={styles.noTasksText}>You have no tasks!</CustomText>
    }
    <FlatList
      data={props.todos}
      extraData={props.rootState}
      renderItem={({ item }) => (
        <TodoItem
          key={item.key}
          todo={item.todo}
          done={item.done}
          doneHandler={() => props.toggleDone(item.key)}
          deleteHandler={() => props.deleteTodo(item.key)}
        />
      )}
    />
  </View>
);
```

Listing 10: Implementatie TodoList component

```
export default TodoList = props => (
  <View style={styles.listContainer}>
    { props.todos.length === 0 &&
    <CustomText style={styles.noTasksText}>You have no tasks!</CustomText>
    }
    <FlatList
      data={props.todos}
      extraData={props.rootState}
      renderItem={({ item }) => (
        <TodoItem
          key={item.key}
          todo={item.todo}
          done={item.done}
          doneHandler={() => props.toggleDone(item.key)}
          deleteHandler={() => props.deleteTodo(item.key)}
        />
      )}
    />
  </View>
);
```

Listing 11: Implementatie TodoList component

Conclusie

Het kan zijn dat je dit artikel leest en denkt dat je veel over React hebt gelezen en weinig over React Native. Dit is precies wat zo aantrekkelijk is aan de library. Als je React eenmaal kent, kan je met weinig extra kennis alle mobiele applicaties (en zelfs native desktop applicaties) met React Native ontwikkelen. React Native biedt een kans aan ontwikkelaars die geen native development (willen) doen om toch mobiele applicatie ontwikkeling te doen met native prestaties. Als je niet bekend met React bent, dan is het een goede kans het te leren, aangezien het steeds meer gebruikt wordt. ■

Spring Boot

Een blik onder de motorkap

In een wereld waarin we steeds vaker microservices bouwen, kom je Spring Boot regelmatig tegen. Spring Boot is een opiniated framework voor het bouwen van Spring applicaties, die je eenvoudig kunt starten vanaf de commandline. Spring Boot maakt gebruik van convention over configuration om zonder al te veel werk snel een applicatie in elkaar te zetten. In dit artikel wil ik je meenemen achter de schermen van Spring Boot.

Als je voor het eerst met een Spring Boot applicatie aan de gang gaat, lijkt alles magisch te gaan. Een Spring Boot applicatie, die je start vanaf de commandline heeft een main class, die er ongeveer als volgt uit ziet (zie **Listing 1**).

```
@SpringBootApplication
public class SampleApplication {

    public static void main(String[] args) {
        SpringApplication.run(SampleApplication.class, args);
    }
}
```

Listing 1

Hier is op zich niks raars aan, want het is een typische Java class met een static main method. Er zijn hier twee zaken die opvallen en te maken hebben met Spring Boot. Ten eerste de annotatie **`@SpringBootApplication`** en daarnaast het gebruikt van de class **`SpringApplication`**. Deze class wordt gebruikt om een Spring applicatie te bootstrappen vanuit een Java main method. De annotatie is een samengestelde annotatie, die het effect heeft van **`@Configuration`**, **`@ComponentScan`** en **`@EnableAutoConfiguration`**. De eerste twee annotaties zijn standaard Spring annotaties. De laatste is speciaal voor Spring Boot en activeert de autoconfiguratie: de magie.

Autoconfiguratie

De **`@EnableAutoConfiguration`** annotatie vertelt Spring Boot dat er gebruik gemaakt gaat worden van de autoconfiguratie mogelijkheid. Dit start met het laden van de **`AutoConfigurationImportSelector`** class. Deze class maakt gebuikt van de standaard **`SpringFactoriesLoader`** class om het hele classpath te scannen voor **`META-INF/spring.factories`** bestanden. Vervolgens worden alle gevonden bestanden gescand op de aanwezigheid van de key **`org.springframework.boot.autoconfigure.EnableAutoConfiguration`**. Als we de inhoud van de spring-boot-autoconfigure JAR bekijken, dan zien we dat er inderdaad een spring.factories bestand is met de genoemde autoconfiguratie key (zie **Listing 2**).

In het spring.factories bestand zien we een hele lijst aan classes staan achter de autoconfiguratie key. Dit zijn allemaal standaard spring **`@Configuration`** classes, die geladen worden door de **`AutoConfigurationImportSelector`**. Kijken we bijvoorbeeld naar een deel van de **`RabbitAutoConfiguration`** class, dan zien we een normale Spring configuratieclass met wat extra annotaties op class en methods (zie **Listing 3**).

In de **`RabbitAutoConfiguration`** zien we bijvoorbeeld de **`@ConditionalOnClass`** annotatie. Deze vertelt Spring Boot dat de autoconfiguratie voor RabbitMQ alleen geladen moet worden als de classes **`RabbitTemplate.class`** en **`Channel.class`** in het classpath gevonden worden. Dit is slechts een van de annotaties, die gebruikt worden bij de autoconfiguratie. Een aantal veel gebruikte condities zijn:

- @ConditionalOnBean
- @ConditionalOnClass
- @ConditionalOnMissingBean
- @ConditionalOnProperty
- @ConditionalOnWebApplication

Bas Passon is werkzaam als software architect en Java consultant bij First8 BV. Open source Java combineren met containers is zijn passie. Hij specialiseert zich daarbij op het snijvlak waar container orkestratie samen komt met de applicatie.

```
# Initializers
org.springframework.context.ApplicationContextInitializer=\
org.springframework.boot.autoconfigure.SharedMetadataReaderFactoryContextInitializer,\
org.springframework.boot.autoconfigure.logging.ConditionEvaluationReportLoggingListener

...

# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration,\
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\
org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\
org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\
...
```

Listing 2

```
@Configuration
@ConditionalOnClass({ RabbitTemplate.class, Channel.class })
@EnableConfigurationProperties(RabbitProperties.class)
@Import(RabbitAnnotationDrivenConfiguration.class)
public class RabbitAutoConfiguration {

    @Configuration
    @ConditionalOnMissingBean(ConnectionFactory.class)
    protected static class RabbitConnectionFactoryCreator {

        @Bean
        public CachingConnectionFactory rabbitConnectionFactory(
                RabbitProperties properties) throws Exception {
                ...
        }
        ...
    }

    ...

    @Configuration
    @ConditionalOnClass(RabbitMessagingTemplate.class)
    @ConditionalOnMissingBean(RabbitMessagingTemplate.class)
    @Import(RabbitTemplateConfiguration.class)
    protected static class MessagingTemplateConfiguration {

        @Bean
        @ConditionalOnSingleCandidate(RabbitTemplate.class)
        public RabbitMessagingTemplate rabbitMessagingTemplate(
                RabbitTemplate rabbitTemplate) {
            return new RabbitMessagingTemplate(rabbitTemplate);
        }

    }
}
```

Listing 3

```
@Target({ ElementType.TYPE, ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Conditional(OnMondayCondition.class)
public @interface ConditionalOnMonday {
}
```

Listing 4

```
@Order(Ordered.HIGHEST_PRECEDENCE)
public class OnMondayCondition extends SpringBootCondition {

    @Override
    public ConditionOutcome getMatchOutcome(ConditionContext context,
AnnotatedTypeMetadata metadata) {
        DayOfWeek today  = LocalDate.now().getDayOfWeek();
        ConditionMessage message = ConditionMessage
                .forCondition(ConditionalOnMonday.class, "expected today
to be a monday")
                .foundExactly(today);
        return new ConditionOutcome(today == DayOfWeek.MONDAY, message);
    }
}
```

Listing 5

Er zijn nog meer condities die gebruikt worden. Kijk hiervoor in het package **`org.springframework.boot.autoconfigure.condition`**. In dit package vind je ook de implementaties van de conditionals. Handig als startpunt als je zelf condities wilt maken, iets wat uiteraard mogelijk is. Stel, je wilt de configuratie om een of andere duistere reden alleen maar laden als de Spring Boot applicatie gestart wordt op maandagen, dan kun je een conditie annotatie definiëren als **`@ConditionalOnMonday`** (zie **Listing 4**).

Deze annotatie verwijst via **`@Conditional(OnMondayCondition.class)`** naar de implementatie van de conditie. De OnMondayCondition class is vrij simpel. Er hoeft alleen maar gecontroleerd te worden of het maandag is (zie **Listing 5**).

De nieuwe conditie kan je dan in je eigen configuratie gebruiken, door hem op een method of class te zetten in één of meerdere van je @Configuration classes (zie **Listing 6** op de volgende pagina).

Uiteraard kun je naast eigen condities ook eigen autoconfiguratie maken. Zoals al uitgelegd, zijn autoconfiguratie classes normale Spring-configuratie classes. Ze worden alleen niet gevonden via de componentscanner, maar via de **`AutoConfigurationImportSelector`**, die spring.factories bestanden doorzoekt op autoconfiguratie informatie. Als we bijvoorbeeld onze **`MondayConfiguration`** willen omvormen tot autoconfiguratie, dan moeten we de class in een aparte JAR plaatsen en deze voorzien van een META-INF/spring.factories bestand, met daarin de al eerder genoemde **`org.springframework.boot.autoconfigure.EnableAutoConfiguration`** key.

Aan deze key voegen we onze autoconfiguratie class toe.

```
org.springframework.boot.autoconfigure.
EnableAutoConfiguration=\
    com.first8.springboot.config.
MondayAutoConfiguration
```

Het is een goed gebruik om autoconfiguratie classes te suffixen met AutoConfiguration. Dan kun je ze direct onderscheiden van de andere configuratie classes. De conventie volgend, hernoemen we de MondayConfiguration class naar MondayAutoConfiguration. Als de JAR nu in het classpath van een Spring Boot applicatie gevonden wordt, zal de configuratie automatisch geladen worden. Als het maandag is op het moment dat de applicatie start, zal er een MondayReporter bean geladen worden.

Soms komt het voor dat de volgorde waarin configuratie door Spring geladen wordt belangrijk is. Dit wil nog wel eens voorkomen in eigen gemaakte autoconfiguratie waar je standaard Spring Boot autoconfiguratie stukken wilt overschrijven. Om dit in goede banen te leiden, wordt er gebruik gemaakt van de **@AutoConfigureAfter**, **@AutoConfigureBefore** en **@AutoConfigureOrder**.

Condities en autoconfiguratie kunnen best complex worden. Het wordt al heel snel onduidelijk waarom de configuratie niet uitpakt, zoals je zou verwachten. Om te achterhalen wat er tijdens de autoconfiguratie gebeurd is, kun je Spring Boot een rapport laten genereren, waarin staat welke configuratie om welke reden wel of niet geladen is. Als je een Spring Boot applicatie start met **-Ddebug** of de property **debug=true** in de application.properties zet, wordt het rapport zichtbaar (zie **Listing 7**).

Configuration Properties

Spring Boot gebruikt een application.properties (of yaml) bestand voor de configuratie van je applicatie. De waarden uit de application.properties worden naar Java objecten geladen en waar nodig wordt typeconversie gedaan. De autoconfiguratie heeft op veel plaatsen configuratie instellingen gedefinieerd. Via de **@EnableConfigurationProperties** annotatie wordt er een class gedefinieerd waar de configuratie in geladen moet worden. Door Spring Boot wordt er een singleton bean beschikbaar gesteld in de context.

Als we even terugkijken naar de autoconfiguratie van RabbitMQ, dan zien we dat daar ook **@EnableConfigurationProperties** anno-tatie gebruikt is. De class **RabbitProperties** wordt gebruikt voor het definiëren van de instellingen (zie **Listing 8**).

Elke class die we willen gebruiken voor configuratie instellingen moet voorzien worden

```
@Configuration
public class MondayConfiguration {

    @ConditionalOnMonday
    @Bean
    public MondayReporter mondayReporter() {
        return new MondayReporter();
    }
}
```

Listing 6

```
Positive matches:
-----------------

   DataSourceAutoConfiguration matched:
      - @ConditionalOnClass found required classes 'javax.sql.DataSource',
'org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseType';
@ConditionalOnMissingClass did not find unwanted class (OnClassCondition)

   DevToolsDataSourceAutoConfiguration matched:
      - DevTools DataSource Condition found auto-configured DataSource
(DevToolsDataSourceAutoConfiguration.DevToolsDataSourceCondition)0

   DispatcherServletAutoConfiguration matched:
      - @ConditionalOnClass found required class
'org.springframework.web.servlet.DispatcherServlet';
@ConditionalOnMissingClass did not find unwanted class (OnClassCondition)
      - found 'session' scope (OnWebApplicationCondition)
   ...

Negative matches:
-----------------

   ActiveMQAutoConfiguration:
      Did not match:
         - @ConditionalOnClass did not find required classes
'javax.jms.ConnectionFactory',
'org.apache.activemq.ActiveMQConnectionFactory' (OnClassCondition)

   BatchAutoConfiguration:
      Did not match:
         - @ConditionalOnClass did not find required class
'org.springframework.batch.core.launch.JobLauncher' (OnClassCondition)

   MondayConfiguration#mondayReporter:
      Did not match:
         - @ConditionalOnMonday today to be a monday found SUNDAY
(OnMondayCondition)
   ...
```

Listing 7

```
@ConfigurationProperties(prefix = "spring.rabbitmq")
public class RabbitProperties {
    ...
    private String host = "localhost";

    private int port = 5672;

    private String username = "guest";
    ...
```

Listing 8

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-activemq</artifactId>
</dependency>
```

Listing 9

van de **@ConfigurationProperties** annotatie. Alle attributen van de class zullen via de application.properties van een waarde voorzien kunnen worden. De hard gecodeerde waarde wordt gezien als de default instelling. De annotatie geeft je ook de mogelijkheid om een prefix voor de instellingen op te geven. De key in de application.properties is gedefinieerd als *prefix + attribuut*. In het geval van RabbitMQ kunnen we dan onder andere de volgende instellingen doen:

- spring.rabbitmq.host=<hostname>
- spring.rabbitmq.port=<port>
- spring.rabbitmq.username=<port>

Je kunt de combinatie **@EnableConfigurationProperties** en **@ConfigurationProperties** ook gebruiken in je eigen code om applicatie instellingen beschikbaar te maken via de application.properties. Het mooie daarvan is dat ze meteen meeliften op het hele configuratiemechanisme van Spring. Dus ook alle mogelijke manieren om waardes te definieren via environment variabelen of system properties werken dan.

Starters

Het managen van dependencies binnen een project is tegenwoordig met alle beschikbare keuzes een complexe aangelegenheid geworden. Dependencies hebben vaak ook weer dependencies en die willen nog wel eens met elkaar clashen. Het handmatig uitzoeken van dependencies is niet echt iets waar je veel tijd mee kwijt wilt zijn. Om het je wat gemakkelijker te maken, beschikt Spring Boot over zogenaamde *starters*. Starters kun je zien als meta dependencies. Het zijn JAR's die niks anders bevatten dan een dependency descriptor. Starters zijn over het algemeen gelinkt aan specifieke Spring of third party technologie en definiëren alle dependencies, die nodig zijn om de betreffende technologie te kunnen gebruiken. De versies van de dependencies zijn zorgvuldig uitgezocht om te voorkomen dat ze conflicteren met de dependencies van andere starters. Zo kun je eenvoudig functionaliteit aan je project toevoegen zonder dat je meteen tegen conflicterende dependencies aan loopt. Er is bijvoorbeeld een starter met de naam **spring-boot-starter-activemq** voor het gebruik van ActiveMQ. In het geval je ActiveMQ wil gebruiken, voeg je de volgende dependency toe (zie **Listing 9**) aan je project en je kunt van start.

Het is niet nodig om een versie op te geven. Spring Boot projecten hebben over het algemeen **spring-boot-starter-parent** als parent gedefinieerd en de versie daarvan bepaalt de versie van de starters. Bij elke release van Spring Boot worden de versies van de dependencies, die een starter beheert, bijgewerkt. In totaal zijn er op dit moment meer dan dertig officiële starters. Daarnaast zijn er nog talloze community starters beschikbaar. Gebruik je in je project dependencies waarvoor nog geen starter beschikbaar is, dan kun je overwegen om er zelf een te maken en te doneren aan de community.

Conclusie

Met behulp van starters en autoconfiguratie kun je developers heel veel werk uit handen nemen bij het opzetten van projecten. Een project dat gebruik maakt van microservices zou bijvoorbeeld een volledige basisset van starters kunnen maken. Hiermee kun je heel eenvoudig een nieuwe microservice beginnen en voorzien van integratie met services, zoals messaging en persistence. Starters hiervoor zouden dan kunnen zijn: **acme-spring-boot-starter-service, acme-spring-boot-starter-persistence**, **acme-spring-boot-starter-messaging** of **acme-spring-boot-starter-test**. De verschillende starters voorzie je dan natuurlijk van de nodige autoconfiguratie om bijvoorbeeld de connectie naar de message broker, database of remote logging server te regelen. Wil je nog meer weten over Spring Boot? Dan is de referentiedocumentatie een goed startpunt. Deze kan je vinden op **https://docs.spring.io/spring-boot/docs/current/reference/html**.

Als laatste is het overigens goed om op te merken dat alles wat je met Spring Boot kunt doen, ook bereikt kan worden met Spring en extra configuratie. Met Spring Boot krijg je misschien meer dependencies dan noodzakelijk voor je project en daarmee ook een groter artifact. ■

ALS JE VOOR HET EERST MET EEN SPRING BOOT APPLICATIE AAN DE GANG GAAT, LIJKT ALLES MAGISCH TE GAAN

Devoxx4Kids

De nieuwe generatie komt eraan!

We werken in een supermooie branche: elke dag innovatie, veel werk voor iedereen en goede perspectieven. ICT zal de wereld de komende decennia alleen nog maar meer gaan beïnvloeden en daar hebben we veel goede mensen voor nodig. Het is belangrijke om kinderen kennis te laten maken met deze wereld vol kansen en innovatie. Dat is dan ook het doel van Devoxx4Kids: kennismaken met de wereld van ICT op een leuke manier.

In 2012 is Devoxx4Kids ontstaan vanuit de Devoxx-organisatie. De eerste Devoxx4Kids werd georganiseerd in Gent en 65 kinderen tussen de 10 en 14 jaren deden hieraan mee. De tieners speelden Scratch, programmeerden de Lego Mindstorms en ontdekten de interessante wereld van de Mars Rovers en de Nao robot. In de zomer van 2013 is de eerste Devoxx4Kids in Nederland (Amsterdam) georganiseerd met ongeveer dezelfde modules. Vanaf die tijd hebben al veel bedrijven Devoxx4Kids-dagen georganiseerd.

Stichting

In Nederland hebben we een Devoxx4Kids stichting. Zoals het hoort heeft de stichting een bestuur met voorzitter, penningmeester en secretaris. Op de website **www.devoxx4kids.org/nederland/** vind je informatie over de stichting. De stichting is in het leven geroepen om ondersteuning te bieden en de formule te bewaken bij de verschillende Devoxx4kids initiatieven, die worden georganiseerd in Nederland. Ondersteuning kan zijn vanuit financieel oogpunt, van communicatieve en/of organisatorische aard of via levering van bijvoorbeeld spullen en (educatieve) materialen.

Materiaal

Op de Devoxx4Kids website is veel materiaal beschikbaar:

- Arduino
- Lego Mindstorms
- Minecraft
- Scratch
- AppInventor
- CS Unplugged

Vaak hebben de bedrijven, die Devoxx4Kids dagen organiseren, ook eigen modules. Vanuit Ordina ontwikkelen we modules, die betaalbaar zijn. De gedachten hierachter is dat de ouders de spullen zelf aan moeten kunnen schaffen op het moment dat het kind enthousiast is. Dat is bijvoorbeeld een nadeel van de Lego Mindstorms, want dit pakket kost al snel €300. Daarom hebben we de module 'clash of robots' ontwikkeld. Dit is een bestuurbare auto op basis van een Arduino en kost ongeveer €50. De aansturing gaat via een app en werkt op iedere Android telefoon. Voor de toekomst zijn we al bezig met een nieuwe module. Het idee is om een app te maken met Kotlin. De kinderen vinden het geweldig als ze zelf een app maken voor hun mobiele telefoon.

De stichting heeft t-shirts, banners en een Nao robot, die gebruikt worden voor de Devoxx4Kids dagen. Iedereen die een Devoxx4Kids dag organiseert, kan deze materialen gratis gebruiken.

Koen Aerts is werkzaam binnen Ordina JTech. Sinds 1999 is hij actief in de ICT-sector, eerst zelf als software ontwikkelaar en momenteel aan de organisatorische kant.

Devoxx4Kids girls edition

Op zaterdag 31 maart hebben de JDuchess en Ordina een Devoxx4Kids girls edition georganiseerd. De reguliere Devoxx4Kids dagen zijn zowel voor jongens als meisjes toegankelijk, dus waarom zou je dan een speciale 'girls edition' organiseren?

Het blijkt nog niet vanzelfsprekend te zijn dat meisjes ook geïnteresseerd kunnen zijn in de wereld van ICT. Dat bleek vorig jaar nog maar eens bij een Devoxx4Kids dag. Een moeder komt met haar zoon en dochter en laat haar zoon achter. Ik vraag haar of de dochter ook blijft en ze reageerde: "ik wist niet dat het ook voor meisjes was". Dat was voor mij de druppel! Dan maar een speciale 'girls edition' organiseren om het nog maar eens te benadrukken dat meiden ook welkom zijn.
De opkomst was erg goed en ook de inschrijving voor deze editie was helemaal vol! 50 enthousiaste meisjes hebben 31 maart een enorm leuke dag beleefd. We hadden drie modules samengesteld:

- **Clash of robots**: in deze eerste workshop bouw je je eigen auto en gaat deze vervolgens met software besturen. 10 auto's worden in een arena geplaatst en jouw auto moet als langste blijven rijden om de winnaar van dit spel te worden.
- **Muziek maken met Sonic Pi**: we gaan aan de slag met een aantal commando's, waarmee je op de computer muziek kunt maken. Stap voor stap leer je hoe je instrumenten samples en ritmes kunt gebruiken om tot een waar muziekstuk te komen. Let op: er zal herrie gemaakt worden!
- **Codecombat**: Codecombat is een gaaf spel, dat zich afspeelt in de kerkers van Kithgard. Je kunt hier alleen ontsnappen door te leren programmeren. Je kunt kiezen uit diverse programmeertalen.

Het was een zeer geslaagde dag en ik hoop dat we in de toekomst net zoveel meisjes als jongens op de Devoxx4Kids dagen mogen verwelkomen.

Oproep

Het is erg fijn dat er al zoveel bedrijven Devoxx4Kids dagen organiseren. Er zijn er inmiddels ongeveer 10 per jaar in Nederland. Maar er zijn nog veel meer kinderen, die hier naartoe willen komen. Iedere keer moeten we weer kinderen teleurstellen, omdat de dag dan volgeboekt is. Daarom zou het mooi zijn als er nog meer bedrijven Devoxx4Kids dagen willen organiseren!

Wil je ook een keer als vrijwilliger met een Devoxx4Kids dag meedoen of wil je samen met jouw bedrijf een Devoxx4Kids dag organiseren? Neem dan contact met ons op via info@devoxx4kids.nl. ■

"IK WIST NIET DAT HET OOK VOOR MEISJES WAS"

Kom naar de NLJUG Masters of Java 2018

Powered by First8!

Het officieuze NK Java Programmeren Masters of Java 2018 vindt dit jaar plaats op woensdag 7 november in Veenendaal. We beginnen na de lunch, en rond 17.30 weten we wie zich Master of Java 2018 mag noemen. Wordt het voor de 3e achtereenvolgende keer Team Faalhaas of is er een verrassende nieuwkomer?

Voor wie het event nog niet kent: de Masters of Java is een "funprogging contest", toegankelijk voor iedere Java ontwikkelaar. In deze wedstrijd wordt programmeervaardigheid onder hoge druk getest. Met de unieke Masters of Java competitie-software gaan we de hele middag leuke en uitdagende programmeerpuzzels oplossen. Weg van integratieproblematiek, framework keuzes en versiebeheer.

Het ervaren quizmasters team van First8 heeft dit jaar weer 6 nieuwe en verfrissende opdrachten voorbereid. Lukt het jou om Master of Java 2018 te worden? Schrijf je in en ga de uitdaging aan!

Code challenges

In een team van maximaal 2 personen werk je aan uitdagende, verrassende en grappige code challenges. De opdrachten duren maximaal 30 minuten, je krijgt een aantal hints en het is de bedoeling is dat je binnen de gestelde tijd tot een oplossing komt. Via kant-en-klare unittesten kun je checken of je code de goede kant op gaat. Iedere seconde die je eerder klaar bent, levert extra punten op. Heb je de oplossing ingestuurd? Dan verlaat je de zaal en via het scherm in de hal zie je hoe de anderen vorderen en na iedere ronde is er een tussentijdse ranking. Een uitstekend recept voor leuke interactie tussen de deelnemers. Je krijgt direct feedback en tips over de verschillende manieren om het probleem op te lossen. Uiteraard zorgen we ook goed voor de inwendige mens.

In 2017 deden er een record aantal deelnemers en teams mee, wat zorgde voor een verhitte strijd . We hopen ook dit jaar weer veel vuurwerk te zien tijdens dit officieuze NK Java programmeren.

Masters of Java is wederom onderdeel van de pre-conference van J-Fall, doe je dus mee met de Masters of Java dan ben je automatisch verzekerd van toegang tot J-Fall en uiteraard wordt het winnende team daar ook nog in het zonnetje gezet.

Ben jij Java specialist en durf je de uitdaging aan? Deelname is kosteloos. Voor meer informatie of deelname aan de wedstrijd check: https://nljug.org/masters-of-java-2018/

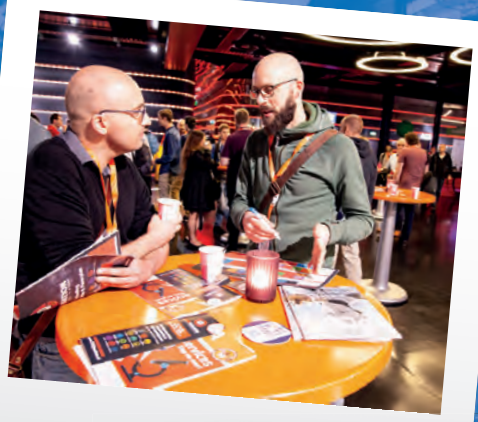
We zien je graag op woensdagmiddag 7 november in Veenendaal!

2018 MASTERS OF JAVA nljug

FIRST8

TEQNATION terugblik

TEQNATION
THE FUTURE IS SMART



TEQNATION The Future is Smart was een geweldige dag vol met de hotste onderwerpen zoals AI & Machine Learning, Big Data en Blockchain en sprekers zoals Laurent Picard van Google en Mark Heckler van Pivotal.

Was je er helaas niet bij? Dan kan je hier een impressie krijgen en zien hoe het eraan toeging. Wil je meer info en foto's? Kijk dan op www.teqnation.nl

Platinum Partners

ING
Rabobank

Partners

Axxerion
IBM
ICU
ORACLE
snow
redhat
accenture technology
CGI
kangaroot
NCIM GROEP
Quintor
topicus
SUSE
TESLA
ORDINA

Hogeschool van Amsterdam

BRENG DE NEXT GEN NAAR EEN NEXT LEVEL

De opleiding **HBO-ICT** van de **Hogeschool van Amsterdam** (**HvA**) is op zoek naar professionals die uitblinken in het vakgebied **Cyber security** of **Software Engineering**. Experts met ervaring, die graag de uitdaging aangaan om jongeren te motiveren en te inspireren.

Ben jij de bevlogen expert die van onze studenten ICT-professionals maakt?

Ga naar **www.hva.nl/hbo-ict-vacatures** en ontdek de mogelijkheden als docent aan de HvA.

Van het bestuur

Terugkijken en vooruitkijken, 15 jaar NLJUG

J-Spring

We kijken als bestuur terug op een zeer succesvolle editie van J-Spring met uitstekende sprekers, Tivoli als toplocatie en ruim 500 bezoekers. De gemiddelde beoordeling van de sessies lag -met 4 uit 5 sterren- erg hoog. Natuurlijk zijn er altijd verbeterpunten die we graag oppakken om volgend jaar een nóg betere editie neer te zetten.

Vooruitkijken naar J-Fall

Tijdens J-Spring hebben we ook de volgende editie van J-Fall aangekondigd. Deze vindt plaats op donderdag 8 november 2018 in Ede. Ook dit jaar is er een pre-conference op 7 november 2018, inclusief een aantal workshops en natuurlijk een spetterende editie van de Masters of Java.

Met de aankondiging van J-Fall is ook de CFP geopend. Grijp jouw kans om hier te spreken en schiet je voorstel in via: **https://nljugcfp.nl/**. Lijkt dit je nét iets te spannend? Ook dit jaar is er weer een mentoring programma voor potentiële sprekers. Houd de website en Twitter in de gaten om je daarvoor aan te melden.

NLJUG 15 jaar

De NLJUG bestaat dit jaar 15 jaar. Hier zullen we tijdens J-Fall uitgebreid bij stil staan. We kijken terug op een hele mooie periode waarin we samen de kennis en passie over ons vakgebied hebben gedeeld. De afgelopen 15 jaar waren zes edities van J-Fall en J-Spring, zijn meer dan 60 edities van Java Magazine uitgekomen hebben we elkaar op vele momenten ontmoet. We kijken dan ook zeker uit naar de volgende 15 jaar!

Vernieuwde site www.nljug.org

Waarschijnlijk heb je het al gezien... Er staat een nieuwe versie van de NLJUG site online. Op de site is meer ruimte voor boeiende content: de talks van vorige edities van J-Fall en J-Spring, de artikelen uit Java Magazine en een overzicht van de events van onze Partners. Heb je tips of mis je nog iets? Feedback voor verdere verbeteringen is natuurlijk altijd van harte welkom.

Jouw artikel in Java Magazine

De redactiecommissie is altijd op zoek naar relevante en boeiende content voor het Java Magazine.

Heb je zelf een onderwerp waar je over wilt schrijven? We horen het graag! Stuur een mail met daarin het onderwerp en een korte beschrijving van het artikel naar **rob.brinkman@nljug.org**. We kijken uit naar jouw bijdrage!

Wil je op een andere manier bijdrage aan Java Magazine? We zijn op zoek naar versterking voor de redactiecommissie. Heb je hier interesse in? Dan maken we graag kennis met je. ■

Rob Brinkman
bestuurslid NLJUG,
redactievoorzitter
Java Magazine

Maven build information

Meer met Maven

In elke editie zal Robert Scholte een probleem voorleggen en deze oplossen met behulp van Apache Maven om meer inzicht te geven in Maven zelf en de vele beschikbare plugins.

Robert Scholte
Freelance ICT consultant, Java architect en chairman Apache Maven.
Twitter: @rfscholte

Met de 3.5.x reeks van Apache Maven wordt flink ingezet op de verbetering van weergave van informatie. Een direct opvallend verschil is uiteraard de toevoeging van kleuren aan de output, maar er is meer. Om te beginnen, is de header van elke module aangepast. Het ziet er voortaan ongeveer als volgt uit (zie **Listing 1**).

```
----------------------< groupId:artifactId >----------------------
Building ProjectName Version                                [2/5]
--------------------------[ packaging ]---------------------------
```

Listing 1

```
Project Name Version ..................... SUCCESS [  0.617 s]
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 4.136 s
Finished at: 2018-05-10T15:52:16+02:00
------------------------------------------------------------------------
```

Listing 2

```
System.gc();
Runtime r = Runtime.getRuntime();
long mb = 1024 * 1024;
logger.info( "Final Memory: " + ( r.totalMemory() -
r.freeMemory() ) / mb + "M/" + r.totalMemory() / mb + "M" );
```

Listing 3

Maven staat redelijk bekend om zijn verbositeit, dus we hebben gezocht naar een compacte opmaak waarbij deze extra informatie getoond wordt.

Met de toevoeging van de groupId + artifactId is het eenvoudiger om te achterhalen welk coördinaat bij een build-blok hoort. Daarnaast kan het helpen bij het analyseren van de build log als je de artifactId kent. De packaging is toegevoegd om een indicatie te krijgen van het eindproduct en welke plugins aangeroepen zullen worden om dit te bereiken. Bijvoorbeeld: als packaging de waarde 'pom' is, worden minstens de maven-install-plugin en maven-deploy-plugin aangeroepen, waarschijnlijk niet veel meer.

Verder wordt in het geval van een multimodule project een index getoond van de module dat gestart wordt. Dit is puur een indicatie en moet een beter gevoel geven over hoe ver de build ondertussen gevorderd is. Ook de presentatie van het resultaat is lichtelijk aangepast (zie **Listing 2**).

De eerste regel van dit overzicht zal voortaan altijd de versie bevatten, zodat je niet meer hoeft te scrollen om te versie te achterhalen. Als het om een multimodule gaat waarbij de versies onderling verschillen, dan zullen alle afwijkende versies ook getoond worden. Verder eindigde tot voorheen elke build met een Final Memory-regel, dat er ongeveer als volgt uit zag:

```
[INFO] Final Memory: 596M/2016M
```

Dit is vast menig developer opgevallen, maar weinig mensen zullen beseffen wat deze getallen inhouden. Om een idee te krijgen wat deze getallen betekenen, kan je het beste naar de onderliggende code kijken (zie **Listing 3**).

De intentie van deze code was waarschijnlijk goed, maar met name de garbage collection call kan tegenwoordig resulteren in onnodige kosten zodra men per CPU-cycle betaald. Het lijkt erop dat deze info ondertussen niet meer relevant is, dus hebben we besloten om het weg te halen.

Tot slot zal de uitput met de volgende Maven release ook de output van "mvn --version" lichtelijk aangepast worden. De regel "Java Home" zal verplaatst en hernoemd worden, omdat deze regelmatig tot verwarring leidt. Wat hier getoond wordt, is niet de waarde van JAVA_HOME, maar de waarde van de system property "java.home", welke altijd naar de directory van de runtime wijst. Dit zijn weliswaar kleine aanpassingen, maar hopelijk dragen ze bij aan een betere ervaring. ■

(On-)menselijke Trekjes

Joop Lanting is vaste columnist voor Java Magazine.

Ik was ±12. Onze kat was ziek. Opgegeven. Mijn broer en ik stopten hem in een mandje en namen tram 7 naar de dierenarts in het centrum van Amsterdam om hem (de kat dus) te laten inslapen.
Daar aangekomen brachten we eerst afschuwelijke minuten door in de wachtkamer tussen al die andere ellendige gevallen. We kwamen aan de beurt. De arts greep de kat, bekeek hem van boven tot onder, pakte een spuit en prikte die in de kat zijn kont. Op een toon als van een kruidenier die nog een koekje op de weegschaal legt voegde hij ons toe: " Zo, neem maar weer mee, die kan nog 10 jaar mee!"

Buitengekomen barsten we in huilen uit, alle spanning kwam vrij. De kat begreep er niks van, maar onze moeder ook niet. Het is ook niet makkelijk uit te leggen.
Ik voelde me helemaal niet lekker, ik had al eens een hartaanval gehad en de angst voor een volgende keer raak je niet kwijt. Maar later op de avond werd het ondraaglijk. Dokter gebeld. Die kwam, vergezeld van een assistente, kijken. "Mogelijk een hartaanval, opname!" Binnen het uur stonden er maar liefst twee ambulances voor de deur, een grote en een kleine. Op mijn pantoffels kroop ik in de grote en werd meteen beplakt met de nodige kabeltjes. Terwijl we naar het ziekenhuis reden verkondigde de broeder dat hij een ernstige afwijking op de monitor zag. Voor mijn gevoel was ik al overleden. Na bloedafname en diverse onderzoeken kwam de dienstdoende cardiologe mij met een pokerface vertellen dat ik GEEN hartaanval had maar een koortsaanval.

Niks aan de hand? Ik mocht (moest) maar weer naar huis. Blij? Helemaal niet blij! Het was al na middernacht. In de hal van het uitgestorven ziekenhuis stond ik een uur te kleumen op mijn pantoffels tot de taxi kwam. Ik deed EHBO en toen een collega in elkaar zakte en zo bleef liggen op kantoor werd ik erbij geroepen. Hij was gauw bij kennis en bleek eerder depressief dan gewond of ziek. De aap kwam spoedig uit de mouw: ze deden met zijn drieën een automatiseringsproject en alles stond op één disk. Enigszins verward had hij de (verwisselbare) disk in de drive gezet en gedachteloos geformat, gewist dus. En .... natuurlijk hadden ze geen backup gemaakt. De ongelukkige wist van radeloosheid niet meer wie hij aan moest kijken. Dat zijn hoogste baas ook wel eens een natuurramp veroorzaakte was niet relevant. Zonder dat iemand het voor hem op had genomen ging hij naar huis en ik heb hem niet meer teruggezien op de zaak.

Ik had cursus gelopen in Engeland en op de terugweg verbleef ik een paar dagen in een bed-and-breakfast. Aan het ontbijt maakte ik een enorme blunder: de 'Landlady' kwam met een krant onder de arm informeren of ik nog "relatives had in Amsterdam". "Nou", zei ik, grappig, "er zijn er niet veel meer over." Ik ging door de grond toen ze me de voorpagina toonde van de krant: een ELAL jumbo was op de Bijlmer gestort. Terug op kantoor kwam ik een vrouwelijke collega tegen die in de Bijlmerflat woonde naast de rampenflat. Ze was helemaal kapot: ze had het vliegtuig vanuit het keukenraam zien neerstorten.

Maar, vertelde ze me, het vliegtuig kwam van rechts en in de Telegraaf (krant) stond dat het van links kwam. Ze kon dit niet verwerken en niemand toonde daarvoor enig begrip. Kort daarna meldde ze zich ziek .... Mijn baas stelde Fred aan ons voor. Hij zou de PC afdeling doen. Apart vertelde hij ons dat Fred een opgegeven kankerpatiënt was, ze gaven hem nog 2 jaar hooguit. Dat bleek angstwekkend juist, het werden 23 maanden.

Het nieuws verspreidde zich snel, want ICT-ers willen ook wel eens over iets anders roddelen dan hardware of software. Fred kreeg veel belangstelling. Maar na een jaar was ik de enige die nog met hem optrok. Iemand uitte zijn 'belangstelling' in de vorm van een terloops "goh, leeft ie nog?". Fred werkte dubbel zo hard als vroeger, maar ging zienderogen achteruit. Hij ging van de verkoopondersteuning over naar een afdeling voor softwareontwikkeling. Zijn nieuwe baas gaf hem een werkplek met een verweesde verouderde PC. Kort daarna kwam er budget vrij en de hele afdeling kreeg nieuwe, grotere en snellere PC's. Maar niet Fred. "Dat heeft toch geen zin meer" zei de baas tegen Fred .... De week daarna meldde Fred zich ziek en bleef weg. Onze wereld is vol kwetsbare individuen. Geen reden om ze dan ook nog te kwetsen. Geen reden om ze in hun nood aan hun lot over te laten .... en ICT-ers zijn ook niet van ijzer!

;JOOP!