Nummer 2 - 2020

JAVA MAGAZINE

Onafhankelijk tijdschrift voor de Java-professional

.nl. jug

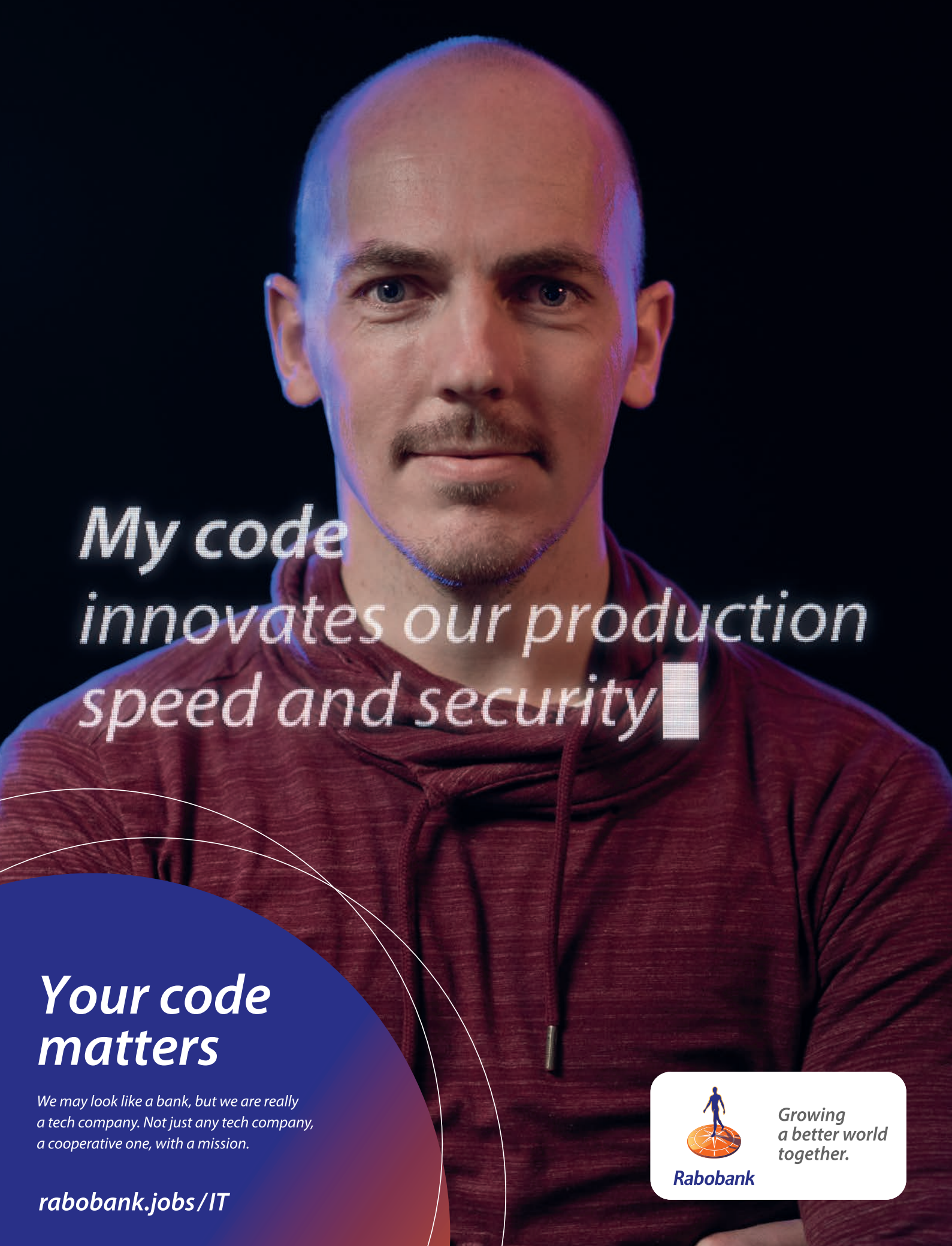
J-Spring Digital is coming!

Features

Load testing met JMeter en AWS

One team, one computer, one codebase Power to the Mob!

Bouw zelf Progressive Selfies Web Apps



Your code matters

We may look like a bank, but we are really a tech company. Not just any tech company, a cooperative one, with a mission.

rabobank.jobs/IT

Rabobank

Growing a better world together.

Colofon Java Magazine 01-2020

Content Manager & Project Coördinatie:
Stijn van de Blankevoort
Eindredactie:
Tedje van Gils
Auteurs:
Nanne Baars, Rob Brinkman, Evertson Croes, Peter Eijgermans, Bas Knopper, Benjamin Komen, Mitchel Kuijpers, Pim Otte, Robert Scholte, Mitchel Schudel, Magdolna Szilagyi, Ko Turk, Jeroen Willemsen, Ivo Woltring
Redactiecommissie:
Stijn van de Blankevoort,
Peter Hendriks, Ben Ooms, Ted Vinke, Mark van der Walle Hedzer Westra, Ivo Woltring, Thomas Zeeman
Vormgeving:
Wonderworks, Haarlem
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 € 49,50 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).

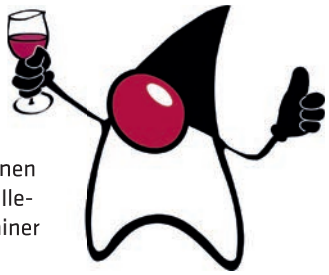
Wij nemen je gegevens, zoals naam, adres en telefoonnummer op in een gegevensbestand. De verwerking van uw gegevens voeren wij uit conform de bepalingen in de Algemene Verordening Gegevensbescherming. De gegevens worden gebruikt voor de uitvoering van afgesloten overeenkomsten, zoals de abonnementenadministratie 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**

Voorwoord

Het nieuwe normaal

Afgelopen periode stond de wereld op zijn kop. Ondertussen zijn velen aan deze nieuwe situatie gewend geraakt. Van thuiswerken, tot zo min mogelijk de straat op gaan en het beperken van sociale contacten. Niet alleen individuen moesten wennen maar ook communities zoals de NLJUG maken hierdoor veranderingen door. Hierover heeft Rob Brinkman een mooie column geschreven op bladzijde 54.
NLJUG blijft staan voor kennisdeling, ongeacht de situatie en externe omgevingsfactoren. Hier zullen we dan ook altijd een weg in vinden. Zo hebben we 29 april de eerste (zeer geslaagde!) NLJUG Masterclass Live mogen geven waarin Sander Mak de nieuwe features van Java 14 besprak. Heb je 'm gemist of wil je rustig teruglezen wat je allemaal kan verwachten van Java 14? Lees dan het artikel van onze Ivo Woltring op pagina 6.
Softskills & methodologie zijn onderwerpen waarover nog maar weinig artikelen zijn verschenen in Java Magazine. Sinds heden proberen we hierover elke editie een artikel te plaatsen, omdat het erg belangrijk is voor je persoonlijke én professionele ontwikkeling. "One team, one computer, one codebase"... Klinkt vreemd, maar het kan écht. Lees Magdolna's artikel over Mob programming.
Maar wat kan je allemaal nog meer vinden in deze tweede editie van Java Magazine in 2020? Er is voor ieder wat wils. Zoals Crux, een document store die indexes legt waarmee je graph queries kan uitvoeren op je data. Dit klinkt indrukwekkend, toch? Maar wat betekent het voor jou? Blader hiervoor door naar pagina 10. Of leer op pagina 14 hoe je zelf een Progressive Selfies Web App kan bouwen in deel twee van Peter Eijegermans artikel. We nemen ook een kijkje naar de programmeertaal Rust en wat we ervan kunnen leren als Javanen met Pim Otte.
Nu we toch allemaal zo veel mogelijk thuis moeten zitten, kan je net zo goed een keer proberen een artikel te schrijven om zo jouw kennis te delen! Naast dat het leuk staat op je CV is het natuurlijk ook hartstikke tof dat je vereeuwigd wordt binnen het Java-domein. Stuur mij een mailtje en misschien sta jij wel in Java Magazine #3 2020.
En om af te sluiten: We mogen Mark van der Walle verwelkomen bij de Java Magazine redactiecommissie! Stuur hem vooral je (digitale) felicitaties! 😊

Stijn van de Blankevoort
Content-/Communitymanager NLJUG
svdblankevoort@reshift.nl



06 Java 14

In Java 14 zijn 16 nieuwe features meegenomen. Om met wat Java 14 features te kunnen spelen, zonder de early access echt te hoeven installeren, is alles uitgeprobeerd binnen een Docker container met OpenJDK 14. Dit artikel zal in drie hoofdsecties verdeeld worden. Het eerste deel zal gaan over nieuwe standaard features. Het tweede deel zal ingaan op preview en incubator features en het laatste deel over deprecated / verwijderde onderdelen. Oftewel: Kom alles te weten over Java 14!

22 Load testing met JMeter en AWS

Over het thema "testen" wordt steeds vaker geschreven en gesproken. Frameworks zoals "Robot" die gebruikt kunnen worden voor end-to-end testing komen frequenter voor in projecten. Dit in combinatie met unit testing zorgt voor een goede bewaking van de codekwaliteit tegen regressie. Deze technieken testen verschillende flows van jouw applicatie als één gebruiker, maar hoe goed werkt jouw applicatie als tien gebruikers tegelijkertijd dezelfde flow doorlopen? En honderd? En duizend? Al verwacht je dat er maar twintig mensen jouw applicatie gaan gebruiken, ook dan is het de moeite waard om een load test uit te voeren om er zeker van te zijn dat alles goed gaat draaien onder die omstandigheden.

40 Power to the mob!

I remember a time, when at every code review, I was screaming inside and dying bit by bit at every line of code. Another piece of code that has no structure and no consistency; another layer is leaking into another one—the nth solution for the same problem inside one application. Put in the mix the long development time and switching team members, and we have the perfect formula to build monsters.

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.

Java 14

Oracle heeft sinds Java SE 10 een 6 maanden releaseschema aangekondigd. De deadline voor nieuwe features in Java is fixed gemaakt. Dit betekent dat het aantal JDK Enhancement Proposals (JEP) dat geïmplementeerd kan worden per release variabel is geworden. Eens in de drie jaar zal een zogenaamde Long-term support (LTS) versie worden gemaakt. Java SE 11 is zo'n LTS versie en zal door Oracle nog tot september 2023 ondersteund worden. Dit zou betekenen dat Java 17 ook weer een LTS wordt. De versies die tussen LTS versies uitkomen krijgen alleen support tot een nieuwe versie uitkomt, dus zo ook Java 14. Dit artikel beschrijft welke nieuwe features in Java 14 geïmplementeerd zijn.

In Java 14 zijn 16 nieuwe features (JEP's) meegenomen. Om met wat Java 14 features te kunnen spelen, zonder de early access echt te hoeven installeren, is alles uitgeprobeerd binnen een Docker container met OpenJDK 14 (listing 1).

Dit artikel zal in drie hoofdsecties verdeeld worden. Het eerste deel zal gaan over nieuwe standaard features. Het tweede deel zal ingaan op preview en incubator features en het laatste deel over deprecated / verwijderde onderdelen. Bij elke feature zal het JEP-nummer vermeld worden.

NIEUWE STANDAARD FEATURES

358: Helpful NullPointerExceptions

Deze JEP verbetert de bruikbaarheid van `NullPointerExceptions` door meer informatie te geven. Gegeven de volgende code (listing 2). Krijg je normaal gesproken een error zoals in Listing 3.
Maar als je hem draait met optie `-XX:+ShowCodeDetailsInExceptionMessages` dan krijg je een specifieke foutmelding te zien (Listing 4).

361: Switch Expressions

De Switch had status preview in Java 12 en 13 en wordt nu standaard onderdeel van de taal. De uitbreiding houdt in dat de switch gebruikt kan worden zowel als een statement als een expressie. Code voorbeelden zijn te vinden in de vorige artikelen (**http://ivo2u.nl/oz**).

345: NUMA-Aware Memory Allocation for G1

Non-uniform memory access (NUMA) is een manier om een cluster van microprocessors te configureren in een systeem met meerdere processors, zodat geheugen lokaal kan worden gedeeld, de prestaties kunnen worden verbeterd en de mogelijkheden van het systeem kunnen worden uitgebreid. In Java 14 is NUMA-bewuste geheugentoewijzing geïmplementeerd om de G1-prestaties op grote machines te verbeteren. Als de `+XX:+UseNUMA` parameter wordt meegegeven als de JVM geïnitieerd wordt, zal een gelijke verdeling plaats gaan vinden over het totale aantal NUMA nodes.

349: JFR Event Streaming

De JDK Flight Recorder is verbeterd om continue monitoring van JDK Flight Recorder-gegevens te ondersteunen. Dus in plaats van het

Ivo Woltring is Software Architect en Codesmith bij Ordina JTech en houdt zich graag bezig met nieuwe ontwikkelingen in de softwarewereld.

```
$ docker run -it --rm \
      -v $(pwd)/src/main/java:/src \
      openjdk:14 /bin/bash
$ cd /src
```

Listing 1.

```
public class NPEApp {
    public static void main(String[] args) {
        final String[] s = new String[2];
        s[1].length();
    }
}
```

Listing 2.

opnemen (recording) en dan parsen van een events-bestand, kunnen events nu realtime gestreamd worden zonder een bestand te gebruiken, of ze kunnen vanuit een bestand worden gestreamd. Een code voorbeeld is hier (**http://ivo2u.nl/oz**) te vinden.

352: Non-Volatile Mapped Byte Buffers

Deze JEP voegt nieuwe JDK-specifieke bestandstoewijzingsmodi toe, zodat de FileChannel API kan worden gebruikt om `MappedByteBuffer`-instanties te maken die verwijzen naar non-volatile memory (NVM) (persistent geheugen). Deze functie wordt alleen ondersteund in Linux op dit moment.

364: ZGC on macOS / 365: ZGC on Windows

De Z Garbage Collector (ZGC) is een "scalable low latency" garbage collector.
ZGC was geïntroduceerd in Java 11, maar alleen voor Linux. Nu ook voor macOS en Windows.

PREVIEW EN INCUBATOR FEATURES

Features die in preview zijn kunnen nog compleet wijzigen in volgende versies. Het is niet te adviseren om deze in productie te gebruiken. Preview features moeten speciaal geactiveerd worden. Dit doe je met de `–source 14 ––enable–preview` parameters. De `–Xlint:preview` parameter is niet nodig, maar geeft output over wat er precies allemaal preview in de code is. Incubator modules zijn een middel om niet-definitieve API's en niet definitieve tools in handen te geven van ontwikkelaars, terwijl de API's / Tools in een toekomstige release leiden naar acceptatie of verwijdering.

305: Pattern Matching for instanceof (Preview)

`instanceof` kan gebruikt worden met een naam erachter. Deze naam functioneert dan als locale variabele als de instanceof expressie waar (true) is. De `if` statement in listing 5 leest als volgt: Als `obj` een instantie van String is cast `obj` dan naar een string en geef het aan `s` (zie listing 5).

359: Records (Preview)

Records bieden een compacte syntaxis voor het declareren van klassen die houders zijn voor onveranderlijke (immutable) gegevens (zie Listing 6).

```
$ javac NPEApp.java && java NPEApp
Exception in thread "main" java.lang.NullPointerException
    at NPEApp.main(NPEApp.java:4)
```

Listing 3.

```
$ javac NPEApp.java && java -XX:+ShowCodeDetailsInExceptionMessages
NPEApp
Exception in thread "main" java.lang.NullPointerException: Cannot in-
voke "String.length()" because "<local1>[1]" is null
    at NPEApp.main(NPEApp.java:4)
```

Listing 4.

```
public class App {
    public static void main(String[] args) {
        //Flip comment op de volgende twee regels om de else tak in te
gaan
        final Object obj = "Het Werkt!!!";
        // final Object obj = 42;
        if(obj instanceof String s) {
            // s is direct te gebruiken als: s = (String) obj;
            System.out.println(s + " <- heeft lengte: " + s.length());
        } else {
            // s is hier niet beschikbaar
            System.out.println("Niet een string");
        }
    }
}
$ javac -source 14 --enable-preview -Xlint:preview App.java
App.java:6: warning: [preview] pattern matching in instanceof is a
preview feature and may be removed in a future release.
        if(obj instanceof String s){
                          ^
1 warning
$ java --enable-preview App
Het Werkt!!! <- heeft lengte: 12
```

Listing 5.

```
record Point(int x, int y) {};
public class JEP359 {
    public static void main(String[] args) {
        final Point p1 = new Point(1, 2);
        final Point p2 = new Point(1, 2);
        if (p1.equals(p2)) {
                System.out.println(p1);
        }
    }
}
$ javac -source 14 --enable-preview -Xlint:preview JEP359.java
JEP359.java:1: warning: [preview] records are a preview feature and
may be removed in a future release.
record Point(int x, int y) {};
^
1 warning
$ java --enable-preview JEP359
Point[x=1, y=2]
```

Listing 6.

368: Text Blocks (Second Preview)

De Text blocks waren al een Preview in Java 13 (JEP 355) en het is nu nog steeds een preview in Java 14, maar nu uitgebreid met twee nieuwe escape sequences: de `\` als line-terminator en de `\s` escape sequence.

De `\s` aan het eind van de zinnen met kleuren (listing 7) garanderen dat de lengte van die zinnen precies 7 zijn en de spaties achter de kleuren blijven bestaan. Andere code voorbeelden zijn te vinden in het Java 13 artikel (**http://ivo2u.nl/oz**).

343: Packaging Tool (Incubator)

De Packaging Tool is een command line tool (`jpackage`) om platform-specifieke packages te maken van Java applicaties. Dit betekent `deb` of `rpm` bestanden op Linux, `pkg` of `dmg` bestanden om macOS en `msi` of `exe` bestanden op Windows. De code uit listing 5 wordt hergebruikt in listing 8.

In de test Docker image was ik niet in staat een `rpm` te bouwen, maar wel een zogenaamde `app-image`.
Dit levert de directory structuur op (Listing 9) met een executable in de bin folder.

370: Foreign-Memory Access API (Incubator)

Deze JEP biedt een API waarmee Java-programma's veilig en efficiënt toegang kunnen krijgen tot geheugen buiten de Java-heap. Dit kan handig zijn om:

- De kosten en de onvoorspelbaarheid van de Garbage collections te vermijden (helemaal als grote caches worden bijgehouden).
- Geheugen te delen tussen meerdere processen.
- Geheugen-content te serializeren en deserializeren door bestanden naar geheugen te laden (via bijvoorbeeld `mmap`)

Een voorbeeld is te zien in listing 10. Voor deze JEP moet een module handmatig toegevoegd moet worden met: `--add-modules jdk.incubator.foreign`

Deprecated en verwijderde Features

Deze sectie beschrijft JEP's die gemarkeerd worden als deprecated of verwijderd zijn. Deprecated betekent dat ze gevlagd worden voor toekomstige verwijdering.

```
public class JEP368 {
    public static void main(String[] args) {
        final String s = """
                Deze regel heeft geen enter hier \
                en een enkele spatie tussen de quotes: "\s"
                groen \s
                rood  \s
                blauw \s
                """;
        System.out.println(s);
    }
}
$ javac -source 14 --enable-preview JEP368.java && java --enable-pre-
view JEP368
Note: JEP368.java uses preview language features.
Note: Recompile with -Xlint:preview for details.
Deze regel heeft geen enter hier en een enkele spatie tussen de
quotes: " "
groen
rood
blauw
```

Listing 7.

```
# Compile de voorbeeld code
$ javac -source 14 --enable-preview App.java
Note: App.java uses preview language features.
Note: Recompile with -Xlint:preview for details.
# Maak een manifest file zoals deze...
$ cat App.mf
Manifest-Version: 1.0
Main-Class: App
# Maak de jar file...
$ jar cmf App.mf app.jar App.class App.java
# Package de applicatie naar een zogenaamde app-image...
$ jpackage --name app --input . --main-jar app.jar \
  --type app-image --java-options "--enable-preview"
WARNING: Using incubator modules: jdk.incubator.jpackage
# Kijken of het werkt...
$ ./app/bin/app
Het Werkt!!! <- heeft lengte: 12
```

Listing 8.

```
.
└── app
    ├── bin
    └── lib
        ├── app
        └── runtime
            ├── conf
            │   ├── management
            │   ├── sdp
            │   └── security
            │       └── policy
            ├── legal
            │   ├── java.base
            │   ├── [nog veel meer items hier ...]
            └── lib
                ├── jfr
                ├── security
                └── server
```

Listing 9.

```
import jdk.incubator.foreign.MemoryAddress;
import jdk.incubator.foreign.MemorySegment;
public class JEP370 {
    public static void main(String[] args) {
        try (MemorySegment segment = MemorySegment.allocateNative(1024)) {
                MemoryAddress base = segment.baseAddress();
                System.out.println(base);
        }
    }
}
$ javac -source 14 --enable-preview --add-modules jdk.incubator.foreign JEP370.java && java --enable-preview
--add-modules jdk.incubator.foreign JEP370
warning: using incubating module(s): jdk.incubator.foreign
1 warning
WARNING: Using incubator modules: jdk.incubator.foreign
MemoryAddress{ region: MemorySegment{ id=0x683ceb2a limit: 1024 } offset=0x0 }
```

Listing 10.

362: Deprecate the Solaris and SPARC Ports

Solaris/SPARC, Solaris/x64, and Linux/SPARC ports zijn deprecated verklaard en zullen in een toekomstige release verwijderd worden.

363: Remove the Concurrent Mark Sweep (CMS) Garbage Collector

CMS Garbage Collection was deprecated verklaard in Java 9 en is nu in Java 14 weggehaald.

366: Deprecate the ParallelScavenge + SerialOld GC Combination

Deze zijn deprecated verklaard omdat ze weinig gebruikt werden en zullen in een toekomstige release verwijderd worden.

367: Remove the Pack200 Tools and API

De Pack200 Tools en API zijn in Java 11 deprecated verklaard en in Java 14 verwijderd.

Conclusie

Een hoop JEP's zijn de revue gepasseerd en een paar leuke features zijn erbij gekomen. Voor de developers is de switch nu standaard geworden en we kunnen nu betere NPE tracing doen. De multi-line string is aan het evolueren en een hoop oud zeer is weggehaald. Al met al een mooie release.

REFERENTIES:

http://openjdk.java.net/projects/jdk/14/
http://ivo2u.nl/oz
https://www.oracle.com/technetwork/java/java-se-support-roadmap.html

Crux

Een bitemporal document database

De eerste Alpha versie van Crux is op 19 april 2019 uitgekomen. Crux is een nieuwe bitemporal database van JUXT. Het is een document store die indexes legt waarmee je graph queries kan uitvoeren op je data. Dit klinkt indrukwekkend, maar wat betekent dit nu?

Om een database bitemporal te mogen noemen, moet deze een dingen kunnen opslaan met een valid-time en een transaction-time. Om dit wat duidelijker te maken, zal ik een paar voorbeelden geven waarbij we langzaam opbouwen naar een bitemporal database. Bij de meeste projecten waar je gebruikmaakt van SQL zul je waarschijnlijk op veel van je tabellen een "Created" kolom toevoegen zoals te zien in Tabel 1.

Dit is een redelijk pijnloze kolom om toe te voegen, want deze timestamp hoef je alleen tijdens de create in te vullen. De volgende stap is om dan een "updated" kolom toe te voegen aan je data model zoals te zien in Tabel 2.

Deze updated kolom wordt al vervelender in je data model. Want dit betekent dat je goed moet opletten in je code dat deze kolom iedere keer geüpdatet wordt. Ook zie je dat je op bijna iedere tabel deze kolommen toe gaat voegen.

Voor sommige werkvelden moet je alle versies van je data bewaren zoals voor de medische of bank industrie. Om dit te bewerkstelligen kan je je tabel opzetten zoals in Tabel 3.

In dit voorbeeld heeft iedere versie van het document een "Valid from" en een "Valid to" kolom om aan te geven wat in een bepaald versie is van een bepaalde entiteit tussen twee tijdspunten. Misschien valt het hier op dat de 10003 geen "Valid to" datum heeft, dit betekent dat dit de live versie is. Je kan je voorstellen dat deze tabellen toevoegen aan je SQL database ervoor zorgt dat je project heel snel groeit in complexiteit vooral als je deze tabellen gaat querien. Om dit op te lossen zijn er al Temporal databases die dit voor je wegnemen:

- Datomic: bij datomic kan je as-of aanroepen op je database om te querien op je data in verschillende punten in tijd.
- SQL:2011: In de SQL 2011 standaard voor SQL hebben ze support toegevoegd voor temporal databases.

Je merkt hierboven dat ik het heb over Temporal databases en nog niet over Bitemporal databases heb. Bij Temporal databases gebruiken ze de "transactie time" van wanneer je iets in de database stopt. Dus dit betekent dat iedere keer als je iets nieuws toevoegt aan de database de transactie tijd omhoog gaat, zie figuur 1 voor een voorbeeld.

Bij Bitemporal databases voegen ze ook een "valid time" toe. Dit zorgt ervoor dat je dingen die in het verleden waren makkelijk kan toevoegen aan de database. Als je kijkt naar Figuur 2. dan zie je dat nog steeds de

Mitchel Kuijpers is als senior full stack developer werkzaam bij Avisi. Hij heft veel kennis over UI development en focust zich met name op development round trips.

ID	*Created*		
10001	2019-08-07T09:39:30		
10002	2019-10-14T09:39:30		
10003	2019-09-20T09:39:30		

Tabel 1.

ID	*Created*	*Updated*	
10001	2019-08-07T09:39:30	2019-08-07T09:39:30	
10002	2019-10-14T09:39:30	2019-10-15T09:39:30	
10003	2019-09-20T09:39:30	2019-09-25T09:39:30	

Tabel 2.

ID	*Version ID*	*Valid from*	*Valid to*
10001	100001	2019-01-07...	2019-03-07...
10001	100002	2019-03-07...	2019-08-07...
10001	100003	2019-08-07	

Tabel 3.

"transaction time" altijd oploopt terwijl de "valid time" ook terug in de tijd kan.

Het voorbeeld van Figuur 2 is wat een bitemporal database is. Het grote voordeel van bitemporal databases is dat je vanzelf een transactietijd bijhoudt, wat heel erg handig is voor CQRS architecturen. Zo wordt het bijvoorbeeld triviaal om naar een transactielog te luisteren en schrijven en lezen op te splitsen. Maar door de "valid time" abstractie kan je nog steeds wel gebruik maken van tijd in je domein. Crux is voor zover wij weten de enige open-source bitemporal database.

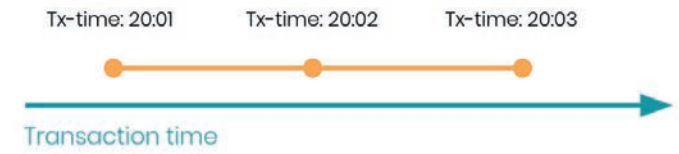
Unbundled

Crux noemt zichzelf een "Unbundled database" (gebaseerd op een artikel van Martin Kleppman). Het is gebouwd met de unix filosofie in het achterhoofd. Het is zo gebouwd dat elk onderdeel doet wat dat onderdeel echt heel erg goed doet. Als je kijkt naar de architectuur van Crux in Figuur 3 dan zie je al dat je voor de Document store en de Bitemporal Graph Indexes kan kiezen voor RocksDB of LMDB. Je ziet dat de transaction en document log gebruik maken van Apache Kafka maar je kan hiervoor bijvoorbeeld ook een JDBC implementatie voor gebruiken. Deze stukken zijn pluggable, wij hebben voor een eigen project twee eigen open-source implementaties geschreven namelijk: **crux-active-objects** en **crux-xodus**, zie Figuur 4 voor een voorbeeld van hoe dat eruit ziet. Wij hadden dit nodig, omdat we bepaalde constraints hadden in dit project. Iets met OSGI, maar daar kunnen we beter niet te diep op ingaan. Je krijgt bij alle implementaties dezelfde API.

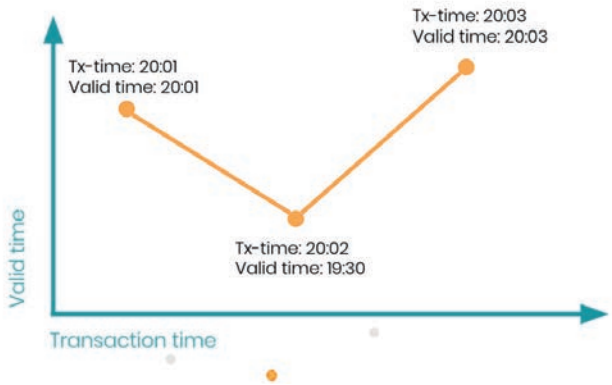
Hoe werkt Crux?

Crux is een schema-less document store, waarbij je gebruikmaakt van transacties om documenten op te slaan. Het opslaan van documenten en het indexeren van documenten is helemaal losgekoppeld. In het voorbeeld van Figuur 3 zie je dat Kafka gebruikt wordt om documenten op te slaan. Je ziet dat de pijl van "Transact" rechtstreeks naar Kafka gaat en dat de "Crux Node" zelf de transacties uit Kafka haalt om deze te indexeren voor queries op de data.

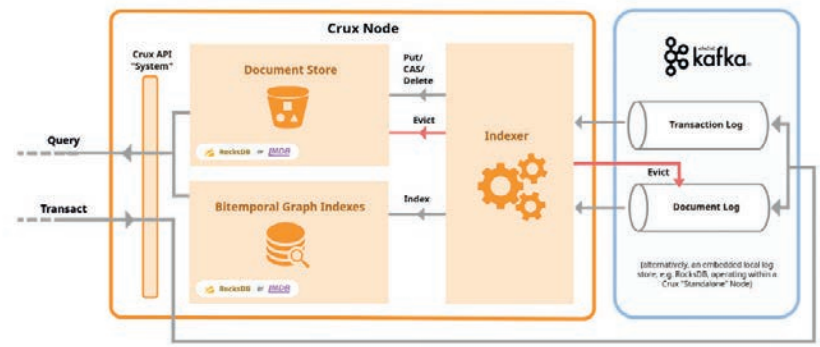
Je ziet bij het Kafka gedeelte van Figuur 3 dat er een "Transaction Log" en een "Document Log" is. In de "Transaction Log" worden alleen transacties opgeslagen met referenties naar content die in de aparte "Document Log" worden bijgehouden. In de "Transaction Log"



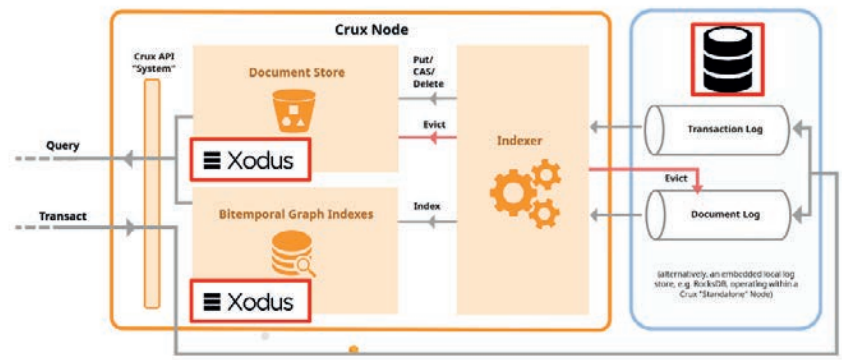
Figuur 1.



Figuur 2.



Figuur 3.



Figuur 4.

verwijzen ze naar content checksums die in de "Document Log" staan. Dit is gedaan zodat de transactie log immutable kan blijven. Ze hebben deze dingen ook losgekoppeld omdat we vanwege de AVG wet soms dingen moeten verwijderen uit onze data stores. Ze doen dit door alle versies van een document te vervangen door een marker dat het is

verwijderd en ze hoeven niks te veranderen in de transaction log. Standaard houdt Crux alle mogelijk versies van een document bij, maar je kan door middel van een "evict" operatie documenten met al hun versies compleet verwijderen.

Het indexeren van data zodat er queries gedaan kunnen worden is geregeld door Crux. Maar ze hebben niet zelf een nieuwe datastore ontwikkeld. Ze maken hiervoor gebruik van bestaande Key Value stores. Ze leggen de juiste indexes zodat je datalog queries kan doen op de geïndexeerde documenten, hier gaan we later nog dieper op in. Maar simpel gezegd sla je maps op met data en worden er standaard indexes op alle properties gelegd.

De API

Crux is geschreven in Clojure maar ze bieden ook een Java en REST API aan. We gaan hier in op de Clojure API omdat ze nog bezig zijn om de Java API gebruiksvriendelijker (lees bruikbaar) te maken. Bij Crux ondersteunen ze de volgende operaties:

- PUT
- DELETE
- CAS
- EVICT

Deze operaties worden in EDN (Extensible Data Notation) geschreven. Zie **https://learnxinyminutes.com/docs/edn/** voor een sneListl overzicht van EDN.

Put

Een Put operatie ziet er als volgt uit. Iedere operatie is een vector waarbij het eerste element de operatie is. In dit geval is de operatie "crux.tx/put". Het tweede element in een operatie is de map met data die je wilt opslaan. In deze map kan alles wat valide EDN is opgeslagen worden. De enige voorwaarde is dat er een id wordt gespecifieerd. Dit doe je met de key "crux.db/id", zie ook Listing 2.

Delete

De delete operatie delete documents bij ID en doet hierbij altijd een soft delete. Dus als ik je iets verwijdert, kan je het nog wel in het verleden deze documenten vinden. Je specificeert deletes zoals in Listing 3.

CAS

De CAS operatie doet een compare and swap. Dus het voert een wijziging alleen door als het expected document precies de verwachte waarde heeft. Zie voor een voorbeeld Listing 4.

Evict

Evict verwijdert een document en alle eerdere versies. Dit is toegevoegd zodat het makkelijker is om je aan de AVG wet te voldoen, zie Listing 5 voor een voorbeeld.

```
[:crux.tx/put ;; The operation
 {:crux.db/id :john-mccarthy ;; The id is required
  :first-name "John" ;; Any valid EDN
  :last-name "McCarthy"
  :birth-date "04-09-1927"
```

Listing 2.

```
[:crux.tx/delete ;; The operation
 :john-mccarthy ;; The id of the document to delete
 #inst "2011-10-24T09:21:52.151-00:00"] ;; Optional: Deletes the document at the given valid time. If you do not specify this this will be the transaction time.
```

Listing 3.

```
[:crux.tx/cas ;; The operation
{..} ;; Expected document
{..} ;; New document
#inst "2018-05-18T09:21:31.846-00:00"] ;; Optional valid time
```

Listing 4.

```
[:crux.tx/evict ;; The operation
 :john-mccarthy] ;; The id of the document to evict
```

Listing 5.

```
[[:crux.tx/put ...]
 [:crux.tx/put ...]
 [:crux.tx/delete ...]
 [:crux.tx/cas ...]]
```

Listing 6.

```
[{:crux.db/id :ivan
  :name "Ivan"
  :last-name "Ivanov"}

 {:crux.db/id :petr
  :name "Petr"
  :last-name "Petrov"}

 {:crux.db/id :smith
  :name "Smith"
  :last-name "Smith"}]
```

Listing 7.

```
{:find [?id]
 :where [[?id :name ?n]
         [?id :last-name ?n]
         [?id :name "Smith"]]}
```

Listing 8.

Transacties

Een transactie in Crux is een lijst van operaties zie Listing 6.

Queries

Voor het querien van data wordt er gebruikt gemaakt van Datalog. Datalog is een logic based query language. Een datalog query bestaat uit een set van variabelen en een set van clausules. Stel je voor dat we de data van Listing 7 in Crux hebben gestopt door middel van drie put operaties.

Dan kunnen we een query schrijven zoals in Listing 8. Met de query gebeurt het volgende:

`[?id :name ?n]` Eerst wordt er door de eerste clausule in the ":where" statement gevraagd om alle documenten die een :name property hebben.
`[?id :last-name ?n]` Vervolgens zoekt die verder waarbij "?id" wordt hergebruikt uit het vorige statement naar alle documenten die dus een ":name" en een ":last-name" hebben.
`[?id :name "Smith"]` Vervolgens maken we set nog kleiner door alleen de documenten te pakken waarbij ":name" een waarde heeft van "Smith"

Nu vraag je je misschien af wat dan het resultaat is van deze query. Het resultaat is alles wat gedefinieerd is in de ":find" clausule. In dit geval dus alleen een :id zie Listing 9 voor het daadwerkelijke resultaat.

```
#{[:smith]}
```

Listing 9.

Er is nog veel meer mogelijk met Crux en Datalog, maar over alleen deze queries zouden we een compleet artikel kunnen schrijven. Het is in ieder geval ook mogelijk om dat Crux graph queries te schrijven en om terug in de tijd te gaan. De makers van Crux hebben een hele gave tutorial geschreven in Nextjournal die je in de browser kan volgen deze kan je vinden op: **https://nextjournal.com/crux-tutorial/start**

Waarom Crux?

Het is altijd leuk om over nieuwe technologieën te lezen, maar ik ben altijd benieuwd waar je deze dan het beste zou kunnen toepassen. De makers van Crux zijn van origine een consultancy bedrijf. Ze zien bij veel klanten incidentele complexiteit veroorzaakt door ad-hoc temporal en bitemporal oplossingen. Denk aan gigantisch complexe SQL queries of SQL databases die bijna niet op te schalen zijn. Je ziet vaak dat tijd toch wel zo'n essentieel ding is in je datamodellen dat je het vroeg of laat toch moet introduceren in je datamodel. Als een database deze complexiteit kan wegnemen dan kan je je focussen op het probleem dat je probeert op te lossen voor de klant. Ik denk dat de meeste evidente uses cases zijn voor bedrijfstakken waarbij het verplicht is om alle versies van data op te slaan zoals bijvoorbeeld: ziekenhuizen, notarissen, overheid en banken.

De introductie van de AVG wet maakt temporal databases nog een stuk lastiger. In een ideale wereld zou het verleden nooit meer veranderen. Maar in de post AVG wet wereld moet je alle referenties naar bepaalde persoonsgegevens kunnen verwijderen, wat kan zorgen voor een gigantisch aantal updates in de database.

Je krijgt bij Crux standaard een "Event Log" waar je naar kan luisteren om andere views op je data te maken. Wij gebruiken dit bijvoorbeeld zelf om ook bepaalde data te indexeren in Lucene zodat we full-text searches kunnen doen op data. Ze bieden ook sinds kort een Kafka Connect plugin waarmee je de transacties kan publiceren naar een ander topic om bijvoorbeeld rapporten te genereren of data in Elasticsearch te indexeren.

Wij gebruiken Crux zelf voor CRM data, waar het mogelijk is om te zien wanneer een bedrijf van adres veranderd is. Of om te kijken wanneer een persoon ergens anders in dienst is gegaan. Als je geïnteresseerd bent in hoe we nu exact Crux gebruiken, zie dan ook onze blogpost: **https://www.avisi.nl/blog/crux-our-final-database-migration** ■

DE INTRODUCTIE VAN DE AVG WET MAAKT TEMPORAL DATABASES NOG EEN STUK LASTIGER

REFERENTIES

Crux: **https://opencrux.com/**
Crux Roadmap: **https://github.com/juxt/crux/projects/8**
Crux kafka connect: **https://www.confluent.io/hub/juxt/kafka-connect-crux**
Crux Active objects transaction log implementatie:
https://github.com/avisi-apps/crux-active-objects
Crux Xodus key value store implementatie:
https://github.com/avisi-apps/crux-xodus
Turning the database inside out Martin Kleppman:
https://martin.kleppmann.com/2015/03/04/turning-the-database-inside-out.html

PWA

Deel 2: Bouw zelf een Progressive Selfies Web App

Als vervolg op het eerste artikel over PWA, gaan we ons richten op het inzetten van API's die toegang hebben tot de hardware op je device. Omgevingslichtsensoren, accelerometers en 'face and shape detectie' zijn voorbeelden van API's die momenteel worden ontwikkeld.

Als je een krachtige PWA wilt bouwen die gebruik maakt van de hardware op een device, wordt het alleen maar beter. In dit artikel ga ik in op enkele PWA-features die toegang geven tot je hardware API's:

- Media Capture API, om een foto (in dit artikel een 'selfie') te kunnen maken met je camera https://developer.mozilla.org/en-US/docs/Web/API/Media_Streams_API
- GeoLocation API, om de locatie van je selfie te bepalen https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API
- en de veelgebruikte Background Sync API, om de selfies tussentijds in een wachtrij of database te plaatsen, zodat de gebruiker ook een selfie naar de server kan zenden als deze geen connectie heeft. **https://developers.google.com/web/updates/2015/12/background-sync**

Project Setup

Om te starten dien je eerst *node* te installeren: https://nodejs.org/en/download/
Als uitgangspunt voor de tutorial, **clone** je deze Github repository:

```
git clone https://github.com/
petereijgermans11/progressive-web-app
```

Ga vervolgens in je terminal naar de directory:

```
cd pwa-article/pwa-app-native-
features-init
```

Installeer de dependencies middels:
`npm i && npm start` en open de webapp op:
http://localhost:8080

Media Capture API

Met de Media Capture API hebben webapps toegang tot de streams via de audio- en video-opname-interfaces van je device. De streams die door de API worden weergegeven, kunnen rechtstreeks worden gekoppeld aan de HTML-elementen <audio> of <video>, of kunnen worden gelezen en gemanipuleerd in de code. Inclusief verdere, meer specifieke verwerking via de Media Recorder API of Real-Time Communication API.

API uitleg

```
navigator.mediaDevices.
getUserMedia(constraints)
```

Hiermee wordt de gebruiker gevraagd om toegang tot de media-interface (video of audio).

```
stream.getAudioTracks()
```

Retourneert een verzameling audiotracks die door de microfoon van je device worden geleverd.

```
stream.getVideoTracks()
```

Retourneert een verzameling videotracks die door de camera van je device worden geleverd.

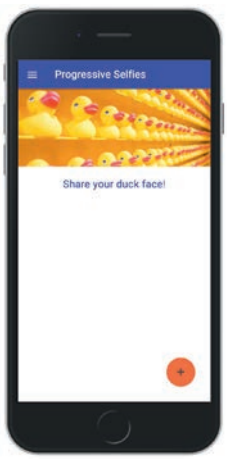
```
mediaElement.srcObject = stream
```

Stelt een stream in die moet worden gerenderd in het meegeleverde <audio> of <video> HTML-element.

De Progressive Selfies-app aanpassen om selfies te kunnen maken

Voeg code van listing 1 toe in je **index.html**, direct onder de tag: `<div id="create-post">`. In deze code wordt de 'Capture button' gede-

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



Afbeelding 1.

Afbeelding 2.

finieerd om een snapshot (=selfie) te maken van je videotracks. Tevens is de <video> en <canvas> tag gedefinieerd voor het respectievelijk tonen van je video en je snapshot (selfie) in je webpagina.

Voeg de code van listing 2 toe in **feed.js**, voor het definiëren van je benodigde variabelen. Hierbij bevat bijvoorbeeld de variabele `videoPlayer` het HTML-element <video> met het id = "player". Hierin worden je videotracks gerenderd. Het `canvasElement` is voor het renderen van je selfie,de `captureButton` is er om een selfie te kunnen maken.

Voeg de **initializeMedia-functie** van listing 3 toe in **feed.js**, om je camera te initialiseren.

Deze code initialiseert de camera. Eerst wordt gecontroleerd of de API van de 'mediaDevices' en 'getUserMedia' beschikbaar is in de **navigator property** *van het window object*. Het window object representeert de browser window (Javascript en ook de *navigator property* is onderdeel van het window object). Als de 'mediaDevices' en 'getUserMedia beschikbaar zijn in de navigator, wordt in bovenstaande code de gebruiker gevraagd om toegang tot de camera, middels de call:

```
navigator.mediaDevices.
getUserMedia(constraints)
```

De call: `videoPlayer.srcObject = stream`, stelt een stream (of videotracks) in, die wordt gerenderd in het meegeleverde <video> HTML-element.

Voeg de code van listing 4 toe in **feed.js,** voor het definiëren van je 'modal' om een selfie maken. Hierin wordt ook de bovenstaande **initializeMedia-functie** aangeroepen**.**

Voeg een *click* event handler toe aan de 'shareImageButton' in **feed.js** (zie listing 5). Deze button (afbeelding 2) *opent* de hierboven gedefinieerde 'openCreatePostModal' om een selfie te maken.

```
shareImageButton.
addEventListener('click',
openCreatePostModal);
```

Listing 5.

Voeg tenslotte een click event handler toe voor 'Capture Button' in **feed.js** (listing 6). Met deze 'Capture Button' kan je een snapshot/selfie maken van je videotracks (zie afbeelding 3). Deze snapshot wordt gerenderd in het *canvasElement* en wordt geconverteerd

```
<video id="player" autoplay></video>
<canvas id="canvas" width="320px" height="240px"></canvas>
<button id="capture-btn"
        class="mdl-button mdl-js-button mdl-button--raised mdl-
button--colored">
     Capture
</button>
```

Listing 1.

```
const videoPlayer = document.querySelector('#player');
const canvasElement = document.querySelector('#canvas');
const captureButton = document.querySelector('#capture-btn');
let picture;
```

Listing 2.

```
const initializeMedia = () => {
   if (!('mediaDevices' in navigator)) {
       navigator.mediaDevices = {};
   }
   if (!('getUserMedia' in navigator.mediaDevices)) {
       navigator.mediaDevices.getUserMedia = (constraints) => {
           const getUserMedia = navigator.webkitGetUserMedia ||
                                navigator.mozGetUserMedia;
           if (!getUserMedia) {
               return Promise.reject(new Error('getUserMedia is not
implemented!'));
           }
           return new Promise((resolve, reject) =>
                  getUserMedia.call(navigator, constraints, resolve,
reject));
       };
   }

   navigator.mediaDevices.getUserMedia({video: {facingMode: 'user'},
audio: false})
       .then(stream => {
           videoPlayer.srcObject = stream;
           videoPlayer.style.display = 'block';
           videoPlayer.setAttribute('autoplay', '');
           videoPlayer.setAttribute('muted', '');
           videoPlayer.setAttribute('playsinline', '');
       })
       .catch(error => {
           console.log(error);
       });
};
```

Listing 3.

```
const openCreatePostModal = () => {
   setTimeout(() => createPostArea.style.transform = 'translateY(0)',
1);
   initializeMedia();
};
```

Listing 4.

naar een Blob (zie listing 7) via de functie: *dataURItoBlob* (voor eventuele opslag in een Database).
Restart eventueel de server met **npm start** en maak een selfie middels de Capture button.

Afbeelding 3.

De Progressive Selfies-app aanpassen om je locatie te bepalen

Geolocation

Met de Geolocation API hebben webapplicaties toegang tot de locatiegegevens die door het apparaat worden verstrekt - verkregen via GPS of via de netwerkomgeving. Afgezien van de eenmalige locatie vraag, biedt het ook een manier om de app op de hoogte te stellen van locatie wijzigingen.

API uitleg

```
navigator.geolocation.
getCurrentPosition(callback)
```

Voert een eenmalige zoekopdracht uit voor de locatie met coördinaten, nauwkeurigheid, hoogte en snelheid, indien beschikbaar.

```
navigator.geolocation.
watchPosition(callback)
```

Locatie wijzigingen worden hiermee geobserveerd.

Laten we de Geolocation API gebruiken om de positie te bepalen van je selfie.
Voeg onderstaande code toe in je **index.html**, direct onder de tag: **div#manual-location**.
In deze code wordt de *'Get Location button'* gedefinieerd om de locatie te bepalen waar je de selfie hebt genomen (listing 8).

```
<div class="input-section">
   <button
       id="location-btn"
       type="button"
       class="mdl-button mdl-js-
button mdl-button mdl-button--
colored">
       Get Location
   </button>
   <div
       id="location-loader"
       class="mdl-spinner mdl-js-
spinner is-active">
   </div>
</div>
```

Listing 8.

Voeg listing 9 toe in **feed.js,** voor het definiëren van je benodigde variabelen om de locatie te bepalen.
Voorafgaand aan **initializeMedia()**-functie in **feed.js** (listing 10):
Voeg de volgende **initializeLocation()**-functie toe in **openCreatePostModal**, direct na **initializeMedia()** in **feed.js** (zie listing 11).

```
captureButton.
addEventListener('click', event => {
   canvasElement.style.display = 'block';
   videoPlayer.style.display = 'none';
   captureButton.style.display = 'none';
   const context = canvasElement.getContext('2d');
   context.drawImage(
       videoPlayer, 0, 0, canvasElement.width,
       videoPlayer.videoHeight / (videoPlayer.videoWidth /
canvasElement.width)
   );
   videoPlayer.srcObject.getVideoTracks().forEach(track => track.
stop());
   picture = dataURItoBlob(canvasElement.toDataURL());
});
```

Listing 6.

In utility.js:

```
const dataURItoBlob= dataURI => {
 const byteString = atob(dataURI.split(',')[1]);
 const mimeString = dataURI.split(',')[0].split(':')[1].split(';')[0];
 const ab = new ArrayBuffer(byteString.length);
 const ia = new Uint8Array(ab);
 for (let i = 0; i < byteString.length; i++) {
     ia[i] = byteString.charCodeAt(i);
 }
 const blob = new Blob([ab], {type: mimeString});
 return blob;
};
```

Listing 7.

```
const locationButton = document.querySelector('#location-btn');
const locationLoader = document.querySelector('#location-loader');
let fetchedLocation = {lat: 0, lng: 0};
```

Listing 9.

```
const initializeLocation = () => {
   if (!('geolocation' in navigator)) {
       locationButton.style.display = 'none';
   }
};
```

Listing 10.

```
const openCreatePostModal = () => {
   setTimeout(() => createPostArea.style.transform = 'translateY(0)',
1);
   initializeMedia();
   initializeLocation();
};
```

Listing 11.

Voeg een click event handler toe voor de **'locationButton'** in **feed.js**. Met deze '*Location button*' wordt de locatie bepaald waar je de selfie hebt genomen (listing 12).

Deze code controleert of de API van de 'geolocation' beschikbaar is in de **navigator property** *van het window object*. Als dit het geval is dan voert deze code een eenmalige zoekopdracht uit om de locatie te bepalen (met coördinaten), via de `navigator.geolocation.getCurrentPosition()`. Vervolgens wordt met deze coördinaten via '*openstreetmap*', het adres erbij gezocht.

Restart eventueel de server met **npm start** en haal de locatie op middels de 'GET LOCATION' button.

Selfies Online versturen met BackgroundSync API

Wat als je wilt dat de gebruiker gegevens naar de server stuurt terwijl de gebruiker offline is, waardoor dit niet mogelijk is? Met de BackgroundSync API kunnen gebruikers gegevens in de wachtrij zetten die naar de server moeten worden verzonden, terwijl een gebruiker offline werkt. Zodra deze weer online is, worden de gegevens in de wachtrij naar de server verzonden.

Voorbeeld:
Stel dat iemand die onze Progressive Selfies app gebruikt een selfie wil maken en versturen, maar de app is offline. Met Background-Sync API kan de gebruiker een selfie in de wachtrij zetten terwijl hij offline is. Zodra deze weer online is, verstuurt de **Service Worker** de gegevens naar de server. In ons geval gebruiken we een **IndexedDB** om gegevens tussentijds op te slaan (afbeelding 4)**.** De library voor deze database kan je vinden in de map **lib/idb.js**. Wat en hoe een **Service Worker** werkt, kun je nalezen in mijn eerste artikel over PWA.

Clone de server

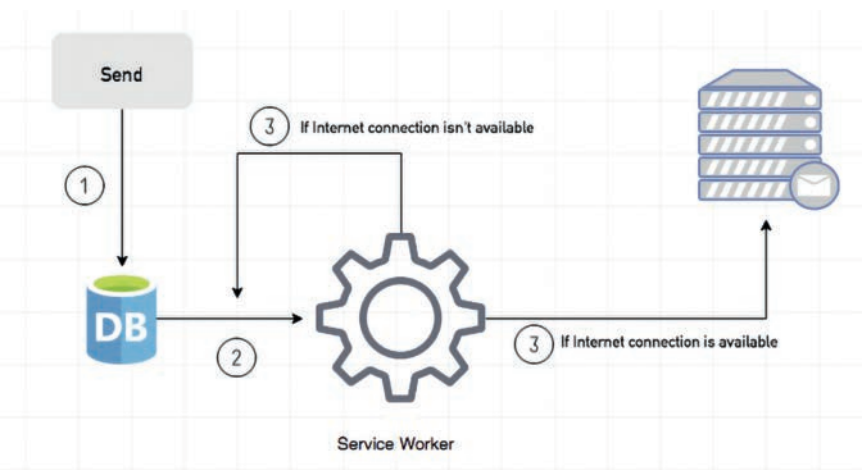
Clone eerst de server via: https://github.com/petereijgermans11/progressive-web-app-server. Naar deze server gaan we je de selfies versturen. Installeer de dependencies en start deze server middels: `npm i && npm start`
De server draait op localhost:3000.

Sync selfies

Om **BackgroundSync** op onze app toe te passen, moeten we een 'store' in onze **IndexedDB**-*database* maken om onze "te synchroniseren selfies" te bewaren (listing 13). We doen dat in **utility.js**. Deze utility.js bevat de code die nodig is voor zowel de **Service Worker** als de app zelf.

```
locationButton.addEventListener('click', event => {
   if (!('geolocation' in navigator)) {
       return;
   }
   let sawAlert = false;
   locationButton.style.display = 'none';
   locationLoader.style.display = 'block';
   navigator.geolocation.getCurrentPosition(position => {
       locationButton.style.display = 'inline';
       locationLoader.style.display = 'none';
       fetchedLocation = {lat: position.coords.latitude, lng:
position.coords.longitude};
       const reverseGeocodeService = 'https://nominatim.openstreetmap.
org/reverse';
                   fetch(`${reverseGeocodeService}?
                        format=jsonv2&lat=${fetchedLocation.
lat}&lon=${fetchedLocation.lng}`)
           .then(response => response.json())
           .then(data => {
               locationInput.value = `${data.address.country}, ${data.
address.state}`;
               document.querySelector('#manual-location').classList.
add('is-focused');
           })
           .catch(error => {
               console.log(error);
               locationButton.style.display = 'inline';
               locationLoader.style.display = 'none';
               if (!sawAlert) {
                   alert('Couldn\'t fetch location, please enter
manually!');
                   sawAlert = true;
               }
               fetchedLocation = {lat: 0, lng: 0};
           });
   }, error => {
       console.log(error);
       locationButton.style.display = 'inline';
       locationLoader.style.display = 'none';
       if (!sawAlert) {
           alert('Couldn\'t fetch location, please enter manually!');
           sawAlert = true;
       }
       fetchedLocation = {lat: 0, lng: 0};
   }, {timeout: 7000});
});
```

Listing 12.



Afbeelding 4.

We dienen de BackgroundSync API in te zetten als we gegevens naar de server zenden, via de **submit** event. Voeg een **submit event handler** toe in **feed.js** (Listing 14).

In het eerste deel van de code wordt de **id** van de te versturen selfie gegenereerd. Eerst doen we een eenvoudige controle om te zien of de browser Service Worker en SyncManager ondersteunt. Als dit het geval is en de **Service Worker** klaar is, registreer dan een synchronisatie (**sync**) met de tag **sync-new-selfies**. Dit is een eenvoudige string die wordt gebruikt om deze **sync-event** te herkennen. Je kunt deze sync-tags beschouwen als eenvoudige labels voor verschillende acties.

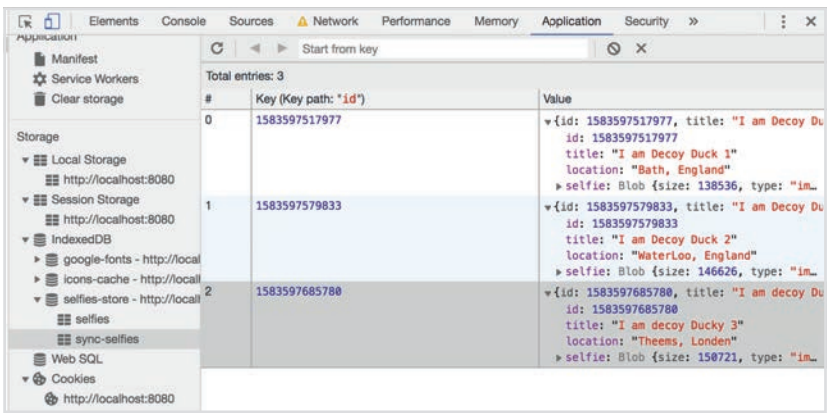
Vervolgens slaan we de selfie op in de **IndexedDB** middels **writeData**.
Tenslotte, worden alle opgeslagen selfies uitgelezen middels **readAllData('sync-selfies')** en worden getoond in je PWA via de functie: **updateUI(syncSelfies)**. Er wordt ook nog een message uitgestuurd: 'Your Selfie was saved for syncing!

De Service Worker

Om de **Service Worker** correct te laten werken, dient er een **sync event-listener** gedefinieerd te worden in **sw.js** (Listing 15).

```
const dbPromise = idb.openDb('selfies-store', 1, upgradeDB => {
   if (!upgradeDB.objectStoreNames.contains('selfies')) {
       upgradeDB.createObjectStore('selfies', {keyPath: 'id'});
   }
   if (!upgradeDB.objectStoreNames.contains('sync-selfies')) {
       upgradeDB.createObjectStore('sync-selfies', {keyPath: 'id'});
   }
});
```

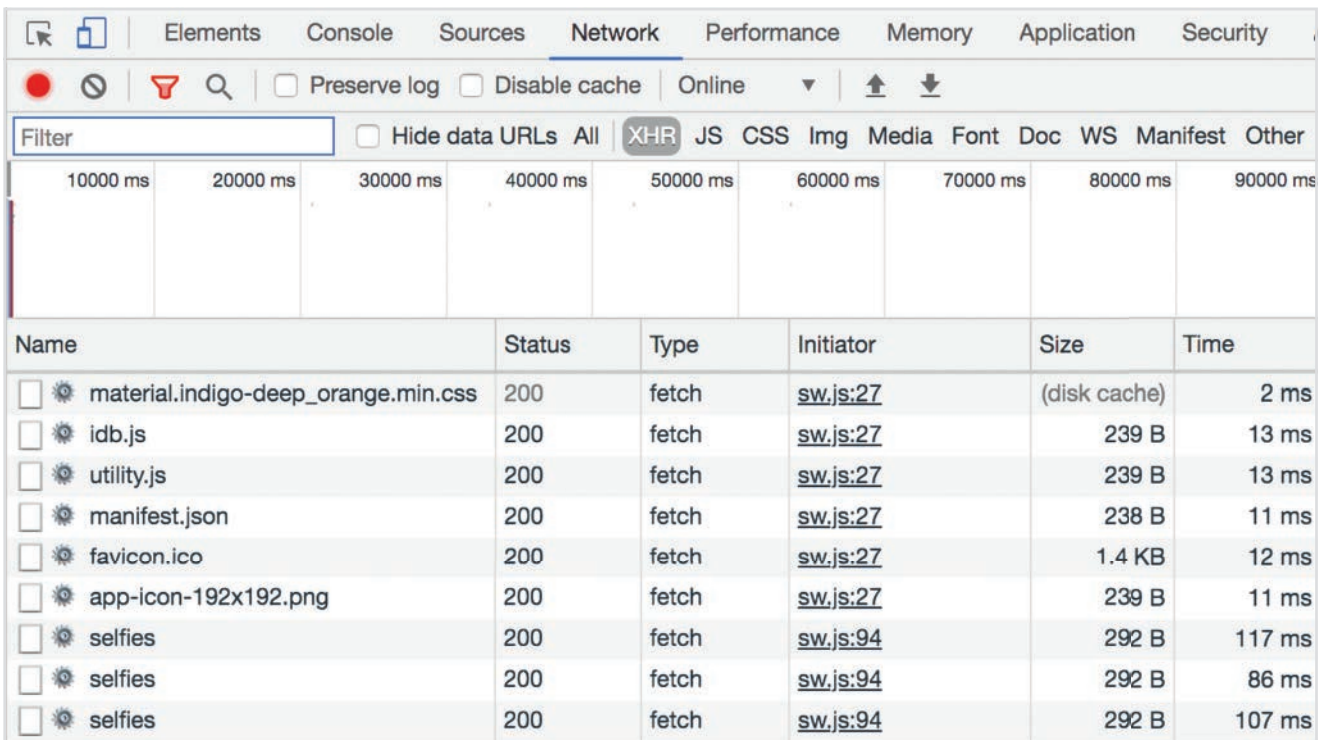
Listing 13.



Afbeelding 5.

Bovenstaande **sync-event** wordt alleen geactiveerd als de browser weer *verbinding* heeft. De Service Worker kan omgaan met dit soort events (zie mijn eerste artikel over PWA). Er

Afbeelding 6.



wordt ook gecheckt of het huidige *sync-event* een **tag** heeft die overeenkomt met de string **'sync-new-selfies'**. Als we deze tag niet zouden hebben, wordt het *sync-event* telkens geactiveerd wanneer de gebruiker verbinding heeft. Vervolgens halen we de selfies op die zijn opgeslagen in de **IndexedDB** toen de gebruiker op de submit-button klikte (zie listing 14). Hierna wordt de **fetch-API** gebruikt om de selfies naar de server te verzenden, middels de API_URL: **http://localhost:3000/selfies**. Tenslotte worden de reeds verstuurde selfies verwijderd uit de **IndexedDB.**

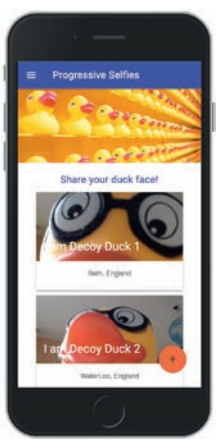
Testing

Geloof het of niet, dit alles testen is eenvoudiger dan je denkt: als je de pagina eenmaal hebt bezocht en de **Service Worker** actief is, hoeft je alleen maar de verbinding met het netwerk te verbreken. Iedere Selfie die je offline probeert te versturen zal nu opgeslagen worden in de **IndexedDB** (zie afbeelding 5). Tevens zullen alle opgeslagen selfies getoond worden in je hoofdscherm (zie afbeelding 7).

Zodra je weer **online** bent zal de **Service Worker** ervoor zorgen dat de selfies verstuurd worden naar de server (bekijk hiervoor je *Netwerk-tab* in je Chrome Developer Tools in afbeelding 6). Daar zie je dat er drie maal een selfie 'gepost' is naar de server. Deze server bevat een folder met de naam *'images'*, waar de verstuurde selfies in staan.

Tenslotte

Na deze introductie kun je verder gaan met een uitgebreide tutorial die je kan vinden in: *https://github.com/petereijgermans11/progressive-web-app/tree/master/pwa-workshop*. In een volgend artikel ga ik verder in op andere API's als Web Streams API, Push API (voor het ontvangen van push notifications) en de web Bluetooth API. ■



Afbeelding 7.

```
form.addEventListener('submit', event => {
   event.preventDefault();
   if (titleInput.value.trim() === '' || locationInput.value.trim()
=== '' || !picture) {
       alert('Please enter valid data!');
       return;
   }
   closeCreatePostModal();

   const id = new Date().getTime();

   if ('serviceWorker' in navigator && 'SyncManager' in window) {
       navigator.serviceWorker.ready
           .then(sw => {
               const selfie = {
                   id: id,
                   title: titleInput.value,
                   location: locationInput.value,
                   selfie: picture,
               };
               writeData('sync-selfies', selfie)
                   .then(() => sw.sync.register('sync-new-selfies'))
                   .then(() => {
                       const snackbarContainer =
                           document.querySelector('#confirmation-toast');
                       const data = {message: ' Selfie saved for syncing!'};
                     snackbarContainer.MaterialSnackbar.showSnackbar(data);
                       readAllData('sync-selfies')
                           .then(syncSelfies => {
                             updateUI(syncSelfies);
                           })
                   })
                   .catch(function (err) {
                       console.log(err);
                   });
           });
   }
});
```

Listing 14.

```
self.addEventListener('sync', event => {
 console.log('[Service Worker] Background syncing', event);
 if (event.tag === 'sync-new-selfies') {
     console.log('[Service Worker] Syncing new Posts');
     event.waitUntil(
         readAllData('sync-selfies')
             .then(syncSelfies => {
                 for (const syncSelfie of syncSelfies) {
                     const postData = new FormData();
                     postData.append('id', syncSelfie.id);
                     postData.append('title', syncSelfie.title);
                     postData.append('location', syncSelfie.location);
                     postData.append('selfie', syncSelfie.selfie);
                     fetch(API_URL, {method: 'POST', body: postData})
                         .then(response => {
                             console.log('Sent data', response);
                             if (response.ok) {
                                 response.json()
                                     .then(resData => {
                                       deleteItemFromData('sync-selfies',
                                          parseInt(resData.id));
                                     });
                             }
                         })
                         .catch(error =>
                           console.log('Error while sending data', error));
                 }
             })
     );
 }
});
```

Listing 15.

Wat ik als Javaan van Rust heb geleerd

"Once you stop learning, you start dying." Met deze wijze woorden van Albert Einstein in mijn hoofd besloot ik om een nieuwe programmeertaal op te pakken. Ik ben ooit begonnen met Visual Basic gevolgd door Python, waarna ik in een professionele omgeving Java, JavaScript en een beetje Go heb kunnen toepassen.

Omdat ik graag wat meer low-level talen wilde proberen, was ik al een keer aan C++ begonnen. Ik was er echter nooit door gegrepen. Doordat Rust zich onder andere als alternatief voor C++ positioneert, leek mij dat een geschikte taal om te proberen. In dit artikel beschrijf ik wat mij het meest is opgevallen in mijn reis door Rust, en hoe dat mij als Javaan heeft beïnvloed.

Het feit dat ik Rust heb leren kennen, heeft in grote lijnen twee gevolgen gehad. Het eerste is dat ik een nieuwe waardering heb voor het gebruik van onveranderlijkheid en invarianten in code. Het tweede gevolg is dat ik extra uitkijk naar en al geniet van een aantal features die met name in de context van Project Amber aan de Java-taal worden toegevoegd.

Geheugengebruik

Rust als taal heeft als doel om programmeurs in staat te stellen om efficiënte en betrouwbare software te schrijven. Het meest kenmerkende is de manier waarop Rust correct gebruik van geheugen afdwingt. Waar talen als Java vertrouwen op een garbage collector om het geheugengebruik netjes bij te houden en vrij te geven wanneer het niet meer gebruikt wordt, en talen als C vertrouwen op de programmeur om handmatig (de)allocaties te doen, is het bij Rust de compiler die problemen als memory leaks voorkomt. Als de compiler niet kan bewijzen dat een stuk code correct gebruikmaakt van het geheugen, dan volgt een compilatiefout. De programmeur kan nog wel de compiler de pas afsnijden door het **unsafe** keyword te gebruiken, maar dit middel wordt doorgaans alleen ingezet als het echt nodig is. Soms kan de compiler namelijk niet aantonen dat een stuk code geldig is, maar kan de programmeur dat wel.

In feite voorkomt de compiler dat de poten onder een stoel weggezaagd worden terwijl er nog iemand op zit. Zolang er alleen mensen op de stoel willen zitten, doet de compiler niks. Zodra er iemand met een zaag aankomt, gaat de compiler controleren of er niemand meer op de stoel kan zitten. Pas als dat met zekerheid gezegd kan worden, mag er aan de stoel geklust worden. Op de stoel zitten komt overeen met leestoegang tot een stuk geheugen; aan de stoel zagen is schrijftoegang tot datzelfde stuk.

Onveranderlijkheid

Ondanks het strenge toezicht van de compiler blijkt dat het nog prima mogelijk is om code te schrijven. De grootste hulp daarbij is onveranderlijkheid. Zonder verdere instructie zijn alle data die door een stuk Rust-code gebruikt worden onveranderlijk, iets wat in Java het **final** keyword vereist. Op het moment dat de programmeur iets wil muteren, moet dat altijd expliciet aangegeven worden met het **mut** keyword, waar mut staat voor "mutable". Dit maakt het ook meteen makkelijker om de code te lezen. De programmeur weet immers ook dat als een stuk data onveranderlijk is, er dan op geen enkele plek wijzigingen aangebracht kunnen worden. Naast het effect op het veilig gebruik van geheugen, blijkt de bescherming die de compiler biedt ook nut te hebben bij concurrency. Ook daar ontstaan problemen doordat data gewijzigd worden op een onverwacht moment. Wederom brengt de door de compiler afgedwongen onveranderlijkheid dan een zekerheid die steun biedt aan

Pim Otte is een consultant/software developer bij Quintor. Bij ICTU bouwt hij binnen het Discipl team oplossingen om de dienstverlening van overheid naar burger te versoepelen. Daarnaast heeft hij de minoren Full-Stack Java en Blockchain aan de Hogeschool Rotterdam gegeven.

de programmeur. Rust formaliseert het delen van data tussen verschillende threads en kan daarmee sommige typen van race conditions (data races) voorkomen.

Toen ik begon met Rust was juist deze onveranderlijkheid iets waar ik mee worstelde. Enerzijds kwam de error `cannot borrow 'variable' as mutable, as it is not declared as mutable` regelmatig terug, wat in eerste instantie tot frustratie heeft geleid. Echter, na meer ervaring begon ik van tevoren na te denken over of ik een variabele zou willen veranderen, en zo ja, dan gaf ik dat aan. Anderzijds kwam ik `variable does not need to be mutable` als error tegen, terwijl ik dacht dat ik deze wel aanpaste. Inderdaad bleek dat ik een paar regels later een typfoutje had gemaakt. Merk op dat voor Java IntelliSense ook een heel eind komt in het signaleren van dit soort problemen. Naar mijn mening zit de echte winst van dit besef van veranderlijkheid in dat het tot een 'nieuw normaal' leidt: onveranderlijk, tenzij anders aangegeven. Dit maakt ook dat ik in Java eerder mijn Lombok `@Data` annotatie splits, en expliciet alleen setters genereer als een field ook daadwerkelijk aangepast zou moeten worden na het instantiëren van het object.

Pattern matching

Rust heeft het **match** keyword, wat de mogelijkheid biedt tot pattern matching. Pattern matching lijkt in de basis op een switch-case constructie. Het patroon waartegen gematcht wordt komt overeen met datgene waarover geswitcht wordt. De verschillende patronen die getest worden komen overeen met de cases. Neem bijvoorbeeld de code in **Listing 1**. Als we hier het geval van het dubbeltje weglaten, zal de compiler ons er met een foutmelding op attenderen dat we een geval missen. Door een tweetal andere mogelijkheden van Rust worden match-expressies nog krachtiger. Ten eerste is het mogelijk om data op te slaan in de varianten van een enum. Ten tweede kunnen die data weer uitgepakt worden met behulp van destructuring. In **Listing 2** definiëren we een enum die een optional integer representeert: ofwel de waarde is `Some` en in dat geval is er ook een bijbehorende integer, ofwel de waarde is `None`. Daarna definiëren we een functie die pattern matching gebruikt om te bepalen of er al dan niet een getal in de optional zit, en vervolgens 1 toevoegt als dat zo is. Om dit te doen wordt de integer in het `Some(i)` geval daadwerkelijk uitgepakt, en is deze benaderbaar onder de naam i.

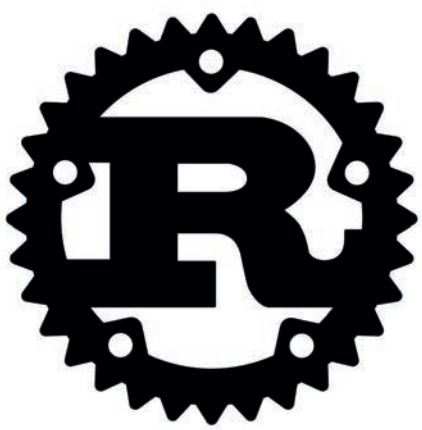
```
enum Munt {
    Cent,
    Dubbeltje,
    Kwartje,
    Gulden,
}
fn waarde_in_cent(munt: Munt) -> u8 {
    match munt {
        Munt::Cent => 1,
        Munt::Dubbeltje => 10,
        Munt::Kwartje => 25,
        Munt::Gulden => 100,
    }
}
```

Listing 1.

```
enum OptionalInt {
    Some(i64),
    None,
}
fn plus_een(x: OptionalInt) -> OptionalInt {
    match x {
        None => None,
        Some(i) => Some(i + 1),
    }
}
```

Listing 2.

ZONDER VERDERE INSTRUCTIE ZIJN ALLE DATA DIE DOOR EEN STUK RUST-CODE GEBRUIKT WORDEN ONVERANDERLIJK



The Rust Programming Language

Project Amber

Deze voorbeelden lichten maar een tipje van de sluier op van wat er allemaal mogelijk wordt met pattern matching. Gelukkig is er voor de Java-taal goed nieuws. Pattern matching is een van de speerpunten van Project Amber (1). Project Amber legt de focus op kleinere features die de productiviteit van de programmeur verhogen. Een voorbeeld hiervan is dat de match-expressie zoals die in Rust bestaat, beetje bij beetje wordt geïntroduceerd. De mogelijkheid om vanuit een switch-case een waarde terug te geven, is geïntroduceerd als preview in JDK 12 in de vorm van switch expressions met JEP 325 (2), en is vervolgens nog iets verfijnd met JEP 354 (3). Pattern matching voor instanceof bestaat in JDK 14 door JEP 305, die het als preview introduceert. Voor uitbreiding van pattern matching bestaan er ook plannen (4) die onder andere de mogelijkheden voor deconstruction beschrijven. Ten slotte worden de mogelijkheden van de enums van Rust ook beschikbaar gesteld met behulp van records en sealed types (5). Records zijn als preview in JDK 14 beschikbaar met JEP 359. Sealed classes zijn nog onderweg.

De bovenstaande verzameling aan previews geeft aan dat er al volop gewerkt wordt om pattern matching naar Java te brengen, en dat het team dat aan Project Amber werkt, goede stappen maakt. Ik heb ook niet het idee dat er in Rust mogelijkheden bestaan op dit gebied die voor Java niet in (voor)ontwikkeling zijn. Ik zie het feit dat Java deze previews heeft ook als een van de vruchten die afgeworpen is door de snellere releases. Hierdoor kan feedback van developers meegenomen worden door middel van preview features, en dit heeft dus daadwerkelijk geleid tot het bijschaven van de switch-expression feature. Je merkt hier ook de invloed van functionele talen zoals Scala, waar pattern matching al lang een veelgebruikte functionaliteit is. Overigens is het concept van onveranderlijkheid ook in het geval van functionele talen iets wat erg gewaardeerd wordt.

Developer-vriendelijke foutmeldingen

In Rust is de compiler streng: er wordt weinig toegelaten. De compiler kan daarentegen ook behulpzaam zijn: als er een error optreedt, volgt er doorgaans ook een goede hint

```
mismatched types
expected reference, found struct 'std::rc::Rc'
note: expected type '&std::rc::Rc<std::cell::RefCell<parser::Letter>>'
         found type 'std::rc::Rc<std::cell::RefCell<parser::Letter>>'
help: consider borrowing here: '&sequence'rustc(E0308)
parser.rs(52, 30): expected reference, found struct 'std::rc::Rc'
For more information about this error, try 'rustc --explain E0308'.
```

Listing 3.

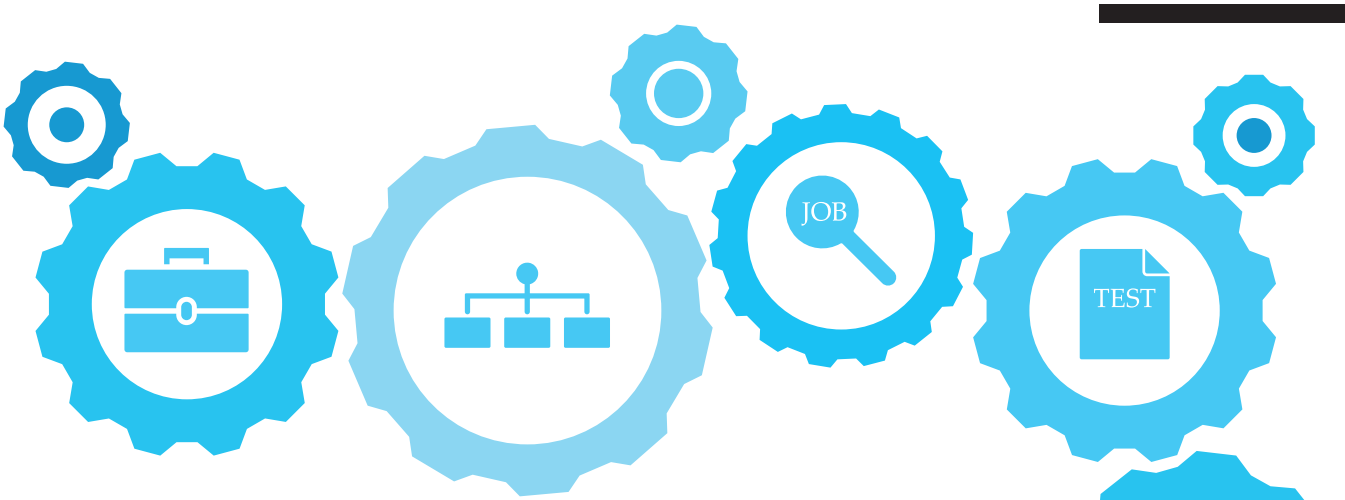
over wat er zou moeten gebeuren. Zoals in **Listing 3**, volgt soms een stukje "help" met een suggestie hoe het te corrigeren, en een commando om meer uitleg te verkrijgen over dit type error. Het fijne hieraan is dat je vaak snel compilatiefouten kunt corrigeren. Het nadeel is dat het verleidelijk is om op automatische piloot de suggesties op te volgen. Voor een ontwikkelaar die net met Rust is begonnen, valt het aan te raden om het Rust-boek (6) erbij te houden. Als we kijken naar Java zou je kunnen vermoeden dat de foutmeldingen al duidelijk genoeg zijn, maar uit JEP 358 (7) blijkt dat er nog steeds verbeteringen doorgevoerd worden. In deze JEP zijn de meldingen bij NullPointerExceptions verbeterd.

Kortom, door Rust heb ik een hernieuwde waardering voor het toepassen van onveranderlijkheid in mijn code, in welke taal dan ook. Daarnaast kijk ik erg uit naar de toevoeging van pattern matching en gerelateerde functionaliteiten in Java. Als jij ook niet kunt wachten, probeer ze dan uit met de `--enable-preview` flag en de nieuwste JDK. Als je ook kennis wilt maken met Rust is de getting-started pagina (8) een goed begin. Het lezen van het boek geeft een brede achtergrond. Voor een aanpak met veel voorbeelden is Rust by Example (9) ook een aanrader. ■

IN RUST IS DE COMPILER STRENG: ER WORDT WEINIG TOEGELATEN

LINKS

[1] **https://openjdk.java.net/projects/amber/**
[2] **https://openjdk.java.net/jeps/325**
[3] **https://openjdk.java.net/jeps/354**
[4] **https://cr.openjdk.java.net/~briangoetz/amber/pattern-match.html**
[5] **https://cr.openjdk.java.net/~briangoetz/amber/datum.html**
[6] **https://doc.rust-lang.org/book/**
[7] **https://openjdk.java.net/jeps/358**
[8] **https://www.rust-lang.org/learn/get-started**
[9] **https://doc.rust-lang.org/stable/rust-by-example/**



ArchUnit

Unittest je architectuur met ArchUnit

Als ontwikkelaar zal je het wel herkennen: je start op een groot project dat al een tijdje draait, en je leest de architectuurdocumentatie die ooit is opgesteld in de vorm van UML-diagrammen. Vervolgens duik je de code in en je vindt, behalve in grote lijnen, weinig meer van die originele architectuur terug. Conventies en afhankelijkheden tussen componenten blijken met voeten getreden te worden, en nieuwe inzichten zijn nooit meer aan de oorspronkelijk bedachte architectuur toegevoegd.

Ralph Johnson omschrijft architectuur als het gemeenschappelijk begrip onder ontwikkelaars over het ontwerp van een systeem[1]. Omdat inzichten veranderen, evolueert de architectuur van een systeem ook. Nu bewaken we de functionaliteit van een evoluerend systeem al met automatische tests; waarom zouden we de architectuur dan ook niet met tests bewaken?

In dit artikel nemen we ArchUnit [2] onder de loep, een open-source-library waarmee je architectural-constraints voor software-architectuur kunt vastleggen als executable code, als unittest.

Waarom ArchUnit?

Er zijn een aantal tools, zoals CheckStyle [4], SpotBugs [5] (voorheen FindBugs) en AspectJ [6] die je in staat stellen om code-constraints vast te leggen. Deze tools hebben wel nadelen: het zijn lastig te begrijpen modellen, hanteren xml als constraintspecificatie, en kennen een vrij technische setup voor custom rules.

Tests in ArchUnit zijn gewone Java-unittests, geschreven via een fluent api. De api is gericht op gebruikelijke constraints in een architectuur, zoals layering, toegang van component A naar component B, en naming-constraints. En omdat het hier unittests betreft, draaien ze gewoon mee met een normale projectbuild.

Let wel, ArchUnit beperkt zich tot de interne structuur van Java-applicaties. De tool ondersteunt bijvoorbeeld geen regels tussen JVM's of constraints die over http/netwerk-grenzen heen gaan.

Aan de slag

ArchUnit gebruiken is een kwestie van de dependency aan je project toevoegen, zie listing 1 voor de Maven-dependency. JUnit 4 en 5 worden allebei ondersteund. In dit artikel gaan we uit van JUnit 5 [7].

Michel Schudel is als software engineer van Craftsmen werkzaam bij de Rabobank. Hij deelt graag kennis over Java in blogs, workshops, en talks op conferenties.

```
<dependency>
   <groupId>com.tngtech.archunit</groupId>
   <artifactId>archunit-junit5</artifactId>
   <scope>test</scope>
   <version>0.13.1</version>
</dependency>
```

Listing 1: JUnit 5 Maven dependency voor ArchUnit.

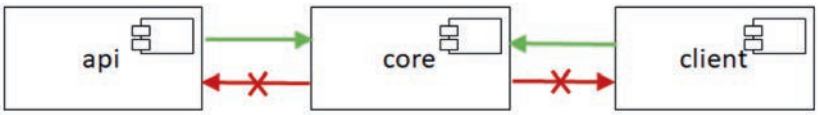
Testvoorbeelden

Neem een Spring Boot-applicatie met drie packages:

- api - bevat de REST-controllers
- core - bevat business-logica
- client - bevat uitgaande calls naar andere systemen

Package rules

We willen vastleggen dat de business-logica in de *core*-package geen afhankelijkheden heeft naar de *api* en *client*-packages. Zie figuur 1.



Figuur 1: Toegestane en verboden package-dependencies.

Listing 2 laat zien hoe je zo'n constraint afdwingt in een ArchUnit-test. Je geeft de root-package van je applicatie op met de annotatie *@AnalyzeClasses*, en stelt vervolgens met de annotatie *@ArchTest* in de fluent api een regel op. Vervolgens run je de unittest met je IDE of een projectbuild. Je kunt bovendien van wildcards gebruikmaken, zodat je niet elke keer de gehele packagenaam hoeft te definiëren. Wat gebeurt er als er toch een niet-toegestane package-import plaatsvindt? Dan faalt de test, met een bericht: *ArchUnitJunit5Test Architecture Violation [Priority: MEDIUM] - Rule 'no classes that reside in a package '..core..' should depend on classes that reside in a package '..api..'' was violated.* Je krijgt dus heel snel feedback over overtreden regels.

```
@AnalyzeClasses(packages = "nl.craftsmen.archunitdemo")
public class ArchUnitJunit5Test {

   @ArchTest
   public static final ArchRule packageRule = noClasses()
           .that()
           .resideInAPackage("..core..")
           .should()
           .dependOnClassesThat()
           .resideInAnyPackage("..api..", "..client..");
}
```

Listing 2: ArchUnit-testvoorbeeld.

Layer rules

Je kunt packages ook groeperen in layers. Op deze manier breng je de test op een abstracter/gelaagder niveau. Zie listing 3.

```
@ArchTest
public static final ArchRule rule = Architectures.layeredArchitec-
ture()
       .layer("Api").definedBy("..api..")
       .layer("Core").definedBy("..core..")
       .layer("Client").definedBy("..client..")

       .whereLayer("Api").mayNotBeAccessedByAnyLayer()
       .whereLayer("Core").mayOnlyBeAccessedByLayers("Api", "Client")
       .whereLayer("Client").mayNotBeAccessedByAnyLayer();
```

Listing 3: Layered architecture rule.

Annotation en naming rules

Het is ook mogelijk te controleren op naamgeving van classes en het correcte gebruik van annotaties. Zoals Listing 4 waarin we controleren of de controller classes wel een *@RestController*-annotatie hebben.

```
@ArchTest
public static final ArchRule namingRule = classes()
       .that()
       .resideInAPackage("..api..")
       .and()
       .haveSimpleNameEndingWith("Controller")
       .should()
       .beAnnotatedWith(RestController.class);
```

Listing 4: Annotation rule.

TESTS IN ARCHUNIT ZIJN GEWONE UNITTESTS, GESCHREVEN VIA EEN FLUENT API

Overige rules
Er zijn nog meer rules voorhanden. Zie de uitgebreide beschrijving van de fluent api in de ArchUnit user guide [3] bijvoorbeeld, maar het beste is om je te laten leiden door de code completion van je IDE. Zo ontdek je het snelst de uitgebreide mogelijkheden.

Custom rules

Waar de standaard api niet expressief genoeg is, zijn er gelukkig ook nog custom rules. Zo is Listing 5 een voorbeeld van een custom rule die in de fluent api controleert of er geen classes zijn met deprecated fields. Helaas ondersteunt ArchUnit (nog) geen lambda-style declaraties.

```
public static ArchCondition<JavaClass> notHaveFieldsAnnotatedWithDe-
precated =
       new ArchCondition<JavaClass>("should not have a field annotated
with @Deprecated") {
           @Override
           public void check(JavaClass item, ConditionEvents events) {
               //if class has a deprecated field
               events.add(SimpleConditionEvent.violated(item, "should
not have a field annotated with @Deprecated"));
           }
       };

@ArchTest
public static final ArchRule customRule = classes().should(notHaveFiel
dsAnnotatedWithDeprecated);
```

Listing 5: Custom rule.

Tests als module

Je kan ArchUnit-tests ook opnemen in een los Maven artifact, zodat diverse applicaties in een project dezelfde architectuur tests kunnen draaien. Vooral in projecten met veel microservices is dat handig om consistentie te bewaken.

Figuur 2 laat zien hoe je dit bereikt. Je creëert een artifact *nl:myarchtests* dat je tests bevat. Vervolgens neem je dit artifact als test-dependency op in je applicatie. Je moet de surefire-plugin van je applicatie wel zo configureren dat de tests die in het artifact staan, meedraaien.

Samengevat

ArchUnit is een krachtige open-source-library om Java-projecten te toetsen aan architecture-constraints.

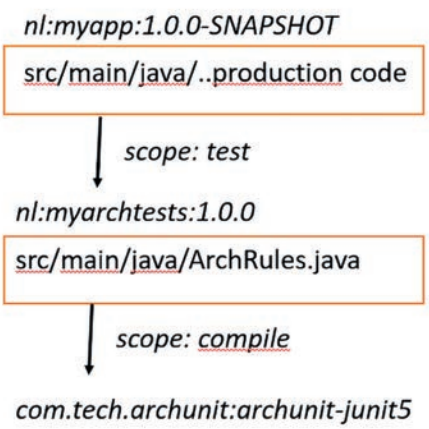
De voordelen op een rij:

- Unittests, dus meteen vanuit IDE te draaien;
- Onderdeel van de code;
- Evolutie van de architecturele constraints en versionering mogelijk, via Maven artifacts.

Voor wie zelf met ArchUnit aan de slag wil, is er een voorbeeldproject beschikbaar op GitLab: **https://gitlab.com/craftsmen/archunit-demo**. Happy testing! ■

REFERENTIES

[1] **https://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf**
[2] ArchUnit: **https://www.archunit.org/**
[3] ArchUnit user guide: **https://www.archunit.org/userguide**
[4] CheckStyle: **https://checkstyle.sourceforge.io/**
[5] SpotBugs: **https://spotbugs.github.io/**
[6] AspectJ: **https://www.eclipse.org/aspectj/**
[7] JUnit 5: **https://junit.org/junit5/**



Figuur 2: ArchUnit-tests als losse module.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
        <dependenciesToScan>
            <dependency>nl:myarchtests:1.0.0</dependency>
        </dependenciesToScan>
    </configuration>
</plugin>
```

Load testing met JMeter en AWS

Hoeveel kan jouw applicatie aan?

Over het thema "testen" wordt steeds vaker geschreven en gesproken. Frameworks zoals "Robot" die gebruikt kunnen worden voor end-to-end testing komen frequenter voor in projecten. Dit in combinatie met unit testing zorgt voor een goede bewaking van de codekwaliteit tegen regressie. Deze technieken testen verschillende flows van jouw applicatie als één gebruiker, maar hoe goed werkt jouw applicatie als tien gebruikers tegelijkertijd dezelfde flow doorlopen? En honderd? En duizend? Al verwacht je dat er maar twintig mensen jouw applicatie gaan gebruiken, ook dan is het de moeite waard om een load test uit te voeren om er zeker van te zijn dat alles goed gaat draaien onder die omstandigheden.

Bij onze projecten hebben wij onverwachte problemen gevonden met onze netwerkinfrastructuur en configuratie, applicatielogica en meer, door het uitvoeren van load tests. Hierdoor konden we ze oplossen voordat ze grote impact hadden op onze projecten. Vaak wordt hier niet naar gekeken en als het aan bod komt, is het dan te laat. In dit artikel wil ik een eenvoudige en goedkope manier delen om load tests te gaan draaien.

JMeter

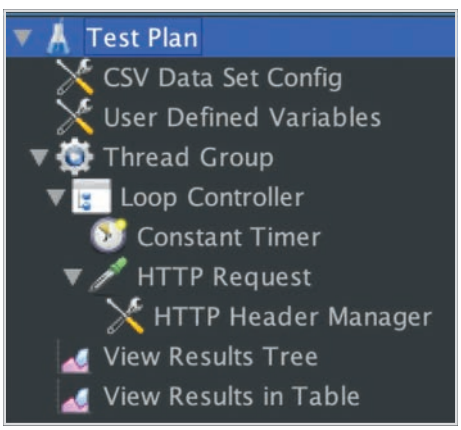
JMeter is een open source Java-applicatie die gebruikt kan worden voor load testing. Als je JMeter voor het eerst opstart, zie je alleen een leeg Test Plan component. Rechtsklikken op de Test Plan geeft je de mogelijkheid om je test plan uit te breiden. Hier gaan wij componenten aan toevoegen om onze load test te definiëren.

Voorbeeldplan

Een voorbeeldtestplan voor het load testen van een REST API ziet eruit zoals in afbeelding 1.

Test plan: De root component van de test plan

- **CSV data set config:** Hiermee kan je een csv file gebruiken als input voor je test. Denk aan een lijst van auth tokens voor de headers.
- **User defined variables:** Hier kan je een lijst van variabelen definiëren die overal in de test gebruikt kunnen worden. Variabelen kunnen gebruikt worden met deze notatie: $(variabeleNaam).
- **Thread group:** Hier geef je aan hoeveel threads parallel moeten gaan draaien. In het geval van dit plan: hoeveel HTTP requests parallel moeten gaan draaien.
- **Loop controller:** Hiermee kan je een "loop" instellen, waardoor alle stappen binnen de loop controller een aantal keer herhaald wordt.
- **Constant timer:** Gebruik dit om een delay te zetten. Als je niet een delay hier zet zal JMeter de HTTP requests allemaal direct achter elkaar afvuren.
- **HTTP request:** Hier staat alle informatie over een specifieke HTTP request zoals de hostname.



Afbeelding 1.

Evertson Croes is een Software Engineer bij Luminis en komt oorspronkelijk uit Aruba. Hij is ook een AWS Certified Solutions Architect

- **HTTP header manager:** Deze gebruik je om headers toe te voegen aan de HTTP requests zoals auth tokens.
- **View results in table:** Deze kan gebruikt worden om de resultaten te zien op een hoog niveau over de hele test. Je kan informatie zien zoals aantal succesvolle en gefaalde calls, en gemiddelde latency.
- **View results in tree:** Deze kan gebruikt worden om de resultaten per HTTP request te zien. Hier kan je zien per request hoe lang het geduurd heeft of waarom het niet succesvol was.

Met behulp van de thread group, loop controller en constant timer kunnen wij ervoor zorgen dat wij een aantal requests per seconde kunnen sturen naar een applicatie. Al deze configuraties kunnen wij geven aan de "user defined variables" zodat wij dat allemaal op één plek kunnen veranderen per acceptatie criteria.

Testplan draaien

Zorg dat je in de HTTP sampler naar een server verwijst die je wilt gaan testen. Je kan om te beginnen ook naar een server verwijzen die lokaal draait. De test plan draaien kan in de GUI via de "Play" knop. De test gaat op dat moment draaien met de ingestelde configuraties. Het resultaat kan je vinden in de "View results in tree/table" component. In afbeelding 2 zie je een voorbeeld met 1 thread en 10 messages (loops).

Dit was een test met een kleine load. Als je een echte load test wilt draaien, dan wordt er aangeraden om de test te draaien vanaf de command line, omdat dit minder resources kost. De test plan wordt opgeslagen in een .jmx file die aangeroepen zoals in Listing 1.

De "-n" parameter geeft aan dat je JMeter wilt draaien in de "Non-GUI" mode. Daarna geef je de locatie aan van de test script en waar de log en xml file opgeslagen mag worden. De volgende drie parameters geven aan wat in de xml file komt te staan. Vanaf dit punt geven wij dezelfde variabelen aan die wij in het testplan bij de "User Defined Variables" hebben opgegeven. Dit kan alleen als je variabelen in de test definieert met deze notatie:

```
jmeter -n \
   -t=test.jmx \
   -j=test.log \
   -l=test.xml \
   -Jjmeter.save.saveservice.output_format=xml \
   -Jjmeter.save.saveservice.response_data=true \
   -Jjmeter.save.saveservice.samplerData=true \
   -JnumberOfThreads=1 \
   -JrampUpTime=1 \
   -Jprotocol=https \
   -Jhost=<<enter host server name here>> \
   -Jpath=<<enter path here, example "/resource">> \
   -JnumberOfMessages=10 \
   -JdelayBetweenMessages=1000 \
```

Listing 1.

```
${__P(numberOfThreads,10)}
```

Waarbij "numberOfThreads" de parameter die gegeven kan worden gegeven vanaf de command line is en 10 de default waarde als dit niet doorgegeven is in de command line.

Als je de test via de command line draait, wordt er in de logging aangegeven welk percentage van de requests geslaagd is. Voor meer informatie zou je de xml file kunnen bekijken, die informatie per request bevat.

Acceptatiecriteria

Het is belangrijk om de acceptatiecriteria te bepalen voor je test. Hoeveel users per seconde wil je testen? Stel dat wij 100 users per seconde willen testen, kunnen wij de numberOfThreads (Thread group) op 100 zetten en de delayBetweenMessages (Constant timer) op 1000 (milliseconde). Om de test langer te laten draaien dan 1 seconde, kunnen wij de numberOfMessages (Loop Controller) zetten op 10. Met deze configuraties hebben wij een test die 100 requests per seconde gaat sturen naar onze server gedurende 10 seconden.

Wat is een acceptabele performance van je applicatie? In de "advanced" instellingen van de HTTP sampler kunnen wij een timeout aan-

HET IS BELANGRIJK OM DE ACCEPTATIE-CRITERIA TE BEPALEN VOOR JE TEST

Afbeelding 2.

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	13:21:22.154	Thread Group 1-1	HTTP Request	42		216	140	42	29
2	13:21:23.198	Thread Group 1-1	HTTP Request	1		216	140	1	0
3	13:21:24.204	Thread Group 1-1	HTTP Request	4		216	140	4	2
4	13:21:25.210	Thread Group 1-1	HTTP Request	1		216	140	1	0
5	13:21:26.213	Thread Group 1-1	HTTP Request	2		216	140	2	1
6	13:21:27.220	Thread Group 1-1	HTTP Request	2		216	140	1	0
7	13:21:28.227	Thread Group 1-1	HTTP Request	2		216	140	2	1
8	13:21:29.236	Thread Group 1-1	HTTP Request	1		216	140	1	0
9	13:21:30.242	Thread Group 1-1	HTTP Request	2		216	140	2	1
10	13:21:31.246	Thread Group 1-1	HTTP Request	1		216	140	1	0

teria hebben: “Bij 100 gebruikers per seconde mogen de requests maximaal 500ms duren”. Dan kunnen wij de timeout in de HTTP sampler zetten op 500. Het succespercentage van de load test geeft in dit geval aan in hoeverre onze applicatie aan onze criteria voldoet.

Amazon Web Services (AWS)

Nu hebben wij een test die wij via de command line kunnen draaien met verschillende instellingen. De mogelijkheid bestaat dat de gewenste load op het systeem zo groot wordt dat je de test niet meer vanaf één machine/PC/laptop kan draaien. Je zou kunnen gaan schalen door meerdere laptops de test tegelijkertijd te laten draaien. Een alternatief is om de Cloud te gebruiken om je JMeter instanties te schalen. In dit onderdeel leg ik uit hoe je dit op AWS kan doen.

Wij gaan Amazon Elastic Container Service (ECS) hiervoor gebruiken. Voordat wij met ECS aan de gang kunnen, moeten wij eerst een Docker image bouwen en deze pushen naar de Elastic Container Registry (ECR).

Docker image

In listing 2 zie je een voorbeeld van een Dockerfile om mee te beginnen.

Deze Dockerfile maakt een image die al onze JMeter properties als environment variables aanmaakt. Vervolgens gaat hij JMeter zelf downloaden en in de container zetten. Als laatste gaat hij de JMeter script draaien met alle parameters. De log en xml files worden ook gelogd.

Elastic Container Registry (ECR)

Nu dat wij een Dockerfile hebben die alle dependencies installeert en de test draait, kunnen wij een Docker image naar AWS sturen. Dit kan met de Elastic Container Registry (ECR) service. In de ECR console, klik op “create repository” en kies een naam. Vervolgens is de repository aangemaakt. Klik op de naam van de repository en klik op “view push commands”. Hier staat precies in wat je moet doen (copy paste van commando’s) om een Docker image te sturen naar ECR. Het resultaat is dat je een image krijgt te zien in jouw repository, zie afbeelding 3.

Elastic Container Service (ECS)

De Elastic Container service is een AWS managed service om containers te draaien. Met behulp van deze service gaan wij meerdere containers naast elkaar draaien die dezelfde JMeter test gaan afvuren. Deze containers zijn gebaseerd op de image die wij in de vorige stap hebben gepusht naar ECR.

```
FROM openjdk:8-jre-alpine3.7
RUN apk update && apk add ca-certificates wget && update-ca-
certificates
ENV JMETER_HOME=/usr/share/apache-jmeter \
   JMETER_VERSION=5.2.1 \
   TEST_SCRIPT_FILE=/var/jmeter/test.jmx \
   TEST_LOG_FILE=/var/jmeter/test.log \
   TEST_RESULTS_FILE=/var/jmeter/test-result.xml \
   TOKEN_FILE=/var/jmeter/tokens.csv \
   PATH="~/.local/bin:$PATH" \
   NUMBER_OF_THREADS=1 \
   RAMP_UP_TIME=1 \
   PROTOCOL=https \
   HOST=<<enter host server name here>> \
   RESOURCE_PATH=test/test \
   NUMBER_OF_MESSAGES=10 \
   DELAY_BETWEEN_MESSAGES=1000

RUN wget http://archive.apache.org/dist/jmeter/binaries/apache-jmeter-
${JMETER_VERSION}.tgz && \
   tar zxvf apache-jmeter-${JMETER_VERSION}.tgz && rm -f apache-
jmeter-${JMETER_VERSION}.tgz && mv apache-jmeter-${JMETER_VERSION}
${JMETER_HOME}

COPY test.jmx ${TEST_SCRIPT_FILE}
COPY tokens.csv ${TOKEN_FILE}
EXPOSE 443

CMD echo -n > $TEST_LOG_FILE && \
   echo -n > $TEST_RESULTS_FILE && \
   $JMETER_HOME/bin/jmeter -n \
   -t $TEST_SCRIPT_FILE \
   -j $TEST_LOG_FILE \
   -l $TEST_RESULTS_FILE \
   -Jjmeter.save.saveservice.output_format=xml \
   -Jjmeter.save.saveservice.response_data=true \
   -Jjmeter.save.saveservice.samplerData=true \
   -JnumberOfThreads=$NUMBER_OF_THREADS \
   -JrampUpTime=$RAMP_UP_TIME \
   -Jprotocol=$PROTOCOL \
   -Jhost=$HOST \
   -Jpath=$RESOURCE_PATH \
   -JnumberOfMessages=$NUMBER_OF_MESSAGES \
   -JdelayBetweenMessages=DELAY_BETWEEN_MESSAGES \
   -JtokenFile=$TOKEN_FILE && \
   echo -e "\n\n===== TEST LOGS =====\n\n" && \
   cat $TEST_LOG_FILE && \
   echo -e "\n\n===== TEST RESULTS =====\n\n" && \
   cat $TEST_RESULTS_FILE
```

Listing 2.

Task definition

Om deze containers te draaien, moeten wij een “task” definiëren in ECS. Klik op Task Definition in de ECS console om te beginnen. Maak een nieuwe aan en kies voor “Fargate”.

Het verschil tussen de Fargate en EC2 mode

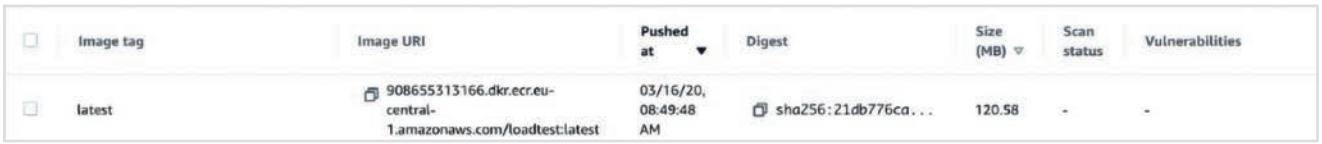
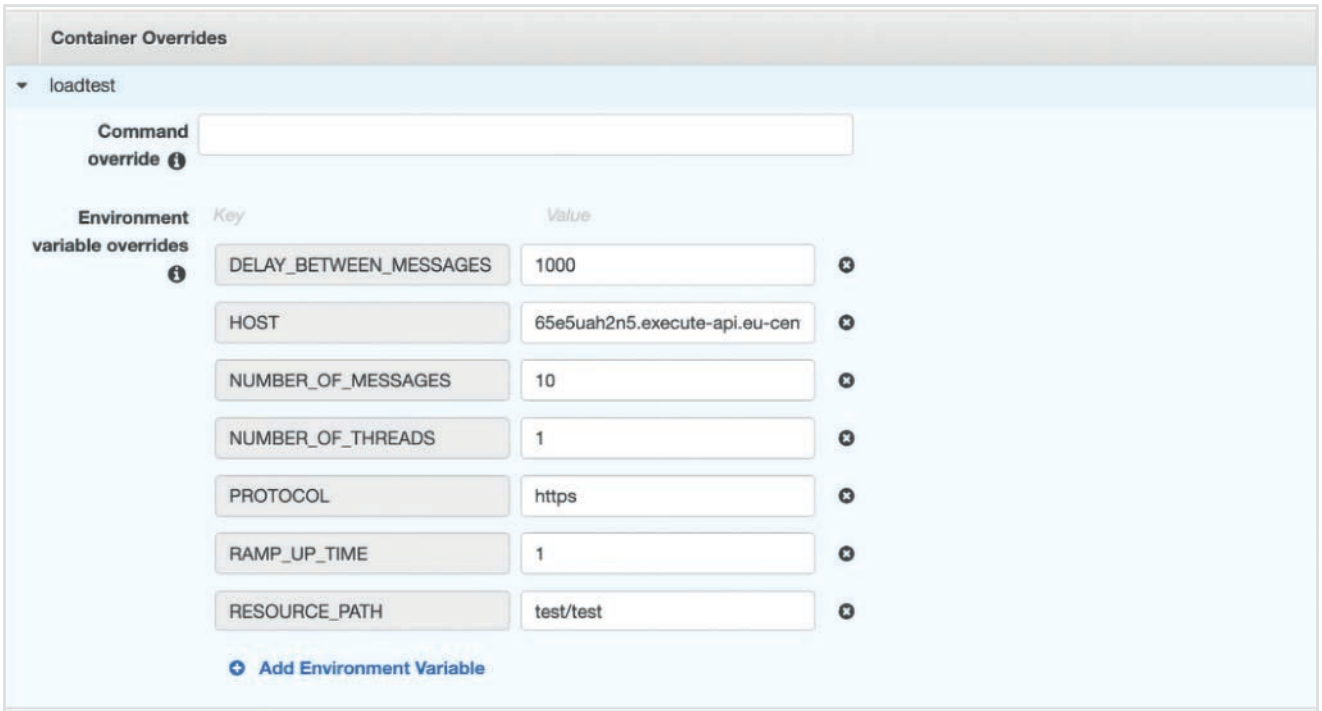


Image tag	Image URI	Pushed at	Digest	Size (MB)	Scan status	Vulnerabilities
latest	908655313166.dkr.ecr.eu-central-1.amazonaws.com/loadtest:latest	03/16/20, 08:49:48 AM	sha256:21db776ca...	120.58	-	-

Afbeelding 3.



Afbeelding 4.

is het volgende. Bij de EC2 variant moet je zelf ervoor zorgen dat je eerst een aantal AWS EC2 instanties opstart waar je je test op wilt gaan draaien. Met Fargate geef je alleen aan hoeveel memory en CPU de taak nodig heeft en hoeveel van de taken je wilt draaien. Amazon regelt zelf dat er genoeg resources opgestart wordt om jouw taken te draaien.

Wij gaan onze taak op de volgende manier instellen (als een instelling niet hier gemeld wordt, kan je het negeren):

- Task Definition Name: loadtest
- Task memory (GB): 2GB
- Task CPU (vCPU): 1vCPU
- Add container:
  - Container name: load testing
  - Image: Ga naar je image die je geupload hebt in ECR en plak hier de URI
  - Memory Limits (MiB): Soft limit, 2048
  - Port mappings: 443, tcp
  - CPU units: 1
  - Environment variables: Voeg alle environment variables die je in je Dockerfile hebt gedefinieerd. Voorbeeld: NUMBER_OF_THREADS met value 1.

De task memory en CPU kunnen iets hoger dan wat wij nu hebben ingesteld. Na het uitvoeren van de test kan je de metrics inzien van je test en beslissen of je meer of minder memory of CPU wilt gebruiken.

Cluster

Op dit punt hebben wij een JMeter test, een Docker image die de JMeter test bevat en een Task Definition die naar de Docker image refereert. Nu moeten wij een Fargate cluster gaan opzetten waar de containers op gaan draaien. Klik op clusters in ECS en klik op “Create cluster” en kies voor “Networking only”. Geef de cluster een naam en klik op “Create”. Nu zijn we klaar om te gaan testen!

Testen

Klik op de net aangemaakte cluster en selecteer de “Tasks” tab. Klik op “Run new Task”. Selecteer de aangemaakte Task definition en cluster. Selecteer de default VPC en Subnet. Klap vervolgens de “Advanced Options” uit om alle instelbare environment variables te zien en aan te passen.
Hiermee kan je verschillende configuraties van load uitproberen zonder dat je de JMeter

Afbeelding 5.

test, Dockerfile en Task definition hoeft te wijzigen. Klik op "Run Task" om de test te starten.

In de cluster krijg je een nieuwe taak te zien, zie afbeelding 5.

Of de load test gelukt is kunnen wij hier niet zien. Wij moeten naar de logging gaan kijken. Alle logging wordt automatisch naar Cloudwatch gestuurd.

Cloudwatch

De logging van de test wordt automatisch gestuurd naar Cloudwatch. Ga naar de CloudWatch console op AWS, klik op "log groups" en zoek op "/ecs/<naam van je container>/". Daar wordt alle JMeter logging, de log file en de .xml file getoond. De JMeter logging geeft al initieel aan welk percentage voor die container geslaagd is.

Als je geïnteresseerd bent in het resultaat van meerdere containers, kan je gebruik maken van "Insights". Met Cloudwatch Insights kan je queries schrijven die over meerdere logs gaan en statistieken berekenen. Stel dat je de gemiddeld latency wilt weten voor een test, zou je de volgende query kunnen draaien:

```
fields @latency
| filter (@message like 'lb="HTTP Request"')
| parse @message 't="*" ' as @latency
| stats avg(@latency) as @averageLatency by bin(1s)
```

Voor meer informatie zie de Cloudwatch Insights documentatie.

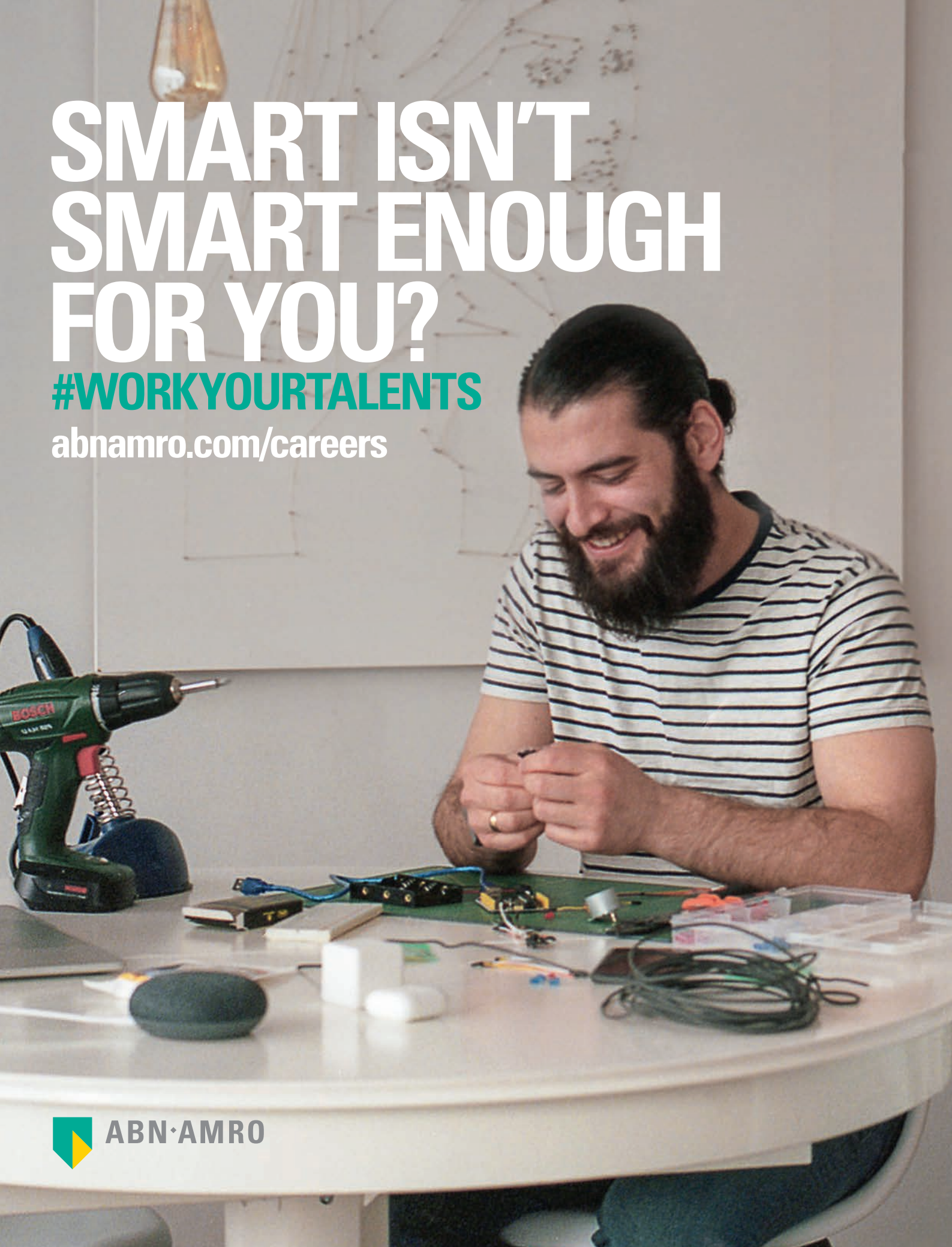
Conclusie

In dit artikel hebben wij een JMeter test aangemaakt. Vervolgens hebben wij een Docker image aangemaakt die alle dependencies installeert voor het draaien van de JMeter test en de JMeter test draait. Deze image hebben wij gepusht naar AWS ECR. Daarna hebben wij een Task Definition aangemaakt die gebruik maakt van de Docker image. Als laatste hebben wij de taak op een ECS cluster gedraaid en de resultaten ingezien op CloudWatch.

Deze setup is instelbaar op verschillende manieren en elke component kan vervangen worden door jouw favoriete tool en Cloud provider. Het belangrijkste is dat je gaat nadenken over load testen. Daar hebben jij en je klant veel aan. Ik hoop dat dit artikel jou heeft gemotiveerd om meer aandacht te gaan geven aan load testing. ■

MEER INFORMATIE

Heb je behoefte aan meer informatie? Ik heb een meer uitgebreide blog op DZone (**https://dzone.com/articles/stop-hoping-for-the-best-and-load-performing-tests**) en een video van Luminis DevCon 2019 op YouTube (**https://www.youtube.com/watch?v=QRxesbPSM_c&t=169s**).
Er is ook een GitLab repository met voorbeelden die gebruikt zijn voor dit artikel.
https://gitlab.com/evertson90/load-test-blog/-/tree/master/jmeter


SMART ISN'T
SMART ENOUGH
FOR YOU?
#WORKYOURTALENTS
abnamro.com/careers
BOSCH
ABN·AMRO

Introductie tot cryptografie in Java

Wanneer je aan de slag moet met cryptografie in Java komen er allerlei zaken op je af: hoe bereik je wat je wilt? Welke mechanismen kun je gebruiken? Hoe doe je dat ongeveer? In dit artikel nemen we je mee door een aantal basisbegrippen. Wil je meer weten? Dan zien we je terug in de volgende uitgave van het Java Magazine met do's and don'ts

Even terug naar wat basisbegrippen. We komen cryptografie overal tegen in het dagelijks leven: wanneer we een website openen met HTTPS, wanneer we een moderne smartphone aanzetten met versleutelde opslag tot en met wanneer we berichtjes sturen naar onze vrienden of familie met WhatsApp/Signal/Telegram. Maar hoe werkt het nu? En belangrijker: hoe passen we het toe in Java? Voordat we naar de code rennen, laten we eerst even wat basisbegrippen uitleggen die in dit artikel vaak hergebruikt worden. Er zijn drie security attributen die je graag wil bereiken met cryptografie:
Confidentiality: hoe hou je geheimen geheim?

- Integrity: hoe voorkom je ongeautoriseerde aanpassingen?
- Non-repudiation: hoe kan je met zekerheid zeggen dat iemand een bericht daadwerkelijk verstuurd heeft?

Cryptografie kent een aantal soorten begrippen:

- Plaintext: de originele niet versleutelde content. Plaintext hoeft niet per se tekst te zijn: dit kan ook gewoon een serie bytes zijn die bijvoorbeeld bestaat uit gecompileerde applicatiecode.
- Ciphertext: de tekst waarop een encryptie operatie heeft plaatsgevonden. De inhoud is versleuteld of encrypted.
- Cipher: een geïnitialiseerd encryptie/decryptie algoritme.
- Encryption: het versleutelen van plaintext naar ciphertext.
- Decryption: het ontsleutelen van ciphertext naar plaintext.
- Signing: het ondertekenen van een bericht
- Signature: de handtekening als output van een sign operatie.

Nu we de basisbegrippen even wat uitleg hebben gegeven, kunnen we aan de slag. Van bijna alle hieronder genoemde vormen van encryptie hebben we in https://github.com/nbaars/java-magazine-article/ een aantal voorbeelden opgenomen. Deze voorbeelden zijn voorzien van uitgebreid commentaar zodat je deze code kunt hergebruiken.

Streaming en block-operations

Voor veel encryptie algoritmes moet de computer allerlei rekenwerk doen op een serie aaneengesloten bits. Vaak is het dan niet efficiënt om deze bits allemaal tegelijk te verwerken. In plaats daarvan is het efficiënter om de serie bits in blokken op te delen van gelijke lengte (bijv. 128 bit) zodat operaties efficiënt uitgevoerd kunnen worden. Op die manier kan je dan iedere keer die 128 bits op dezelfde manier door elkaar halen (transponeren) of als een getal weergeven om er rekenwerk mee te doen. Ciphers die op die manier werken noemen we een "block cipher".

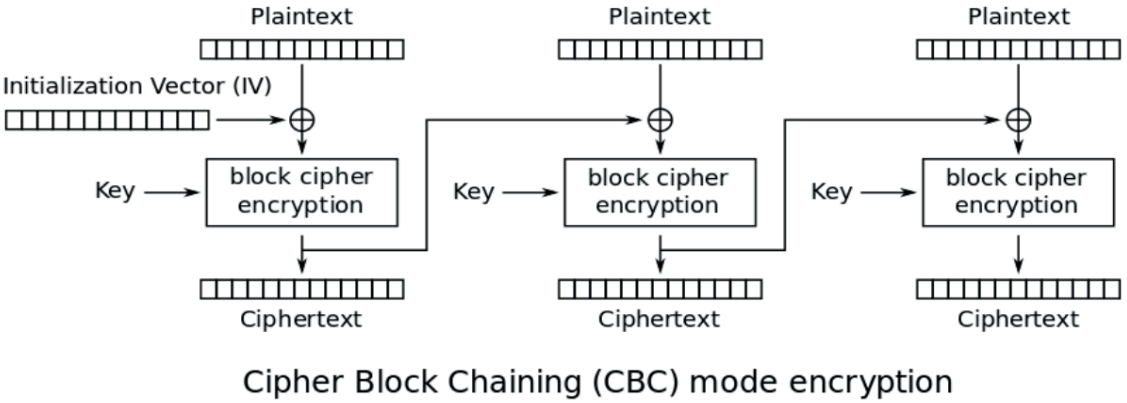
Daarentegen zijn er ook ciphers die bit-voor-bit kunnen werken. Dit noemen we zogenaamde "stream cipher". Daarin wordt er vaak per bit een operatie gedaan: bijvoorbeeld een XOR tussen de plaintext bit en een stream aan random bits.
Maar wat nu als we bits overhouden? Stel je voor dat je in blokjes van 128 bits werkt en opeens is het laatste stukje nog te versleutelen plaintext maar 32 bit. Wat doe je dan? Dan moet je de rest van het blokje opvullen, dit noemen we padding.

Symmetrische encryptie

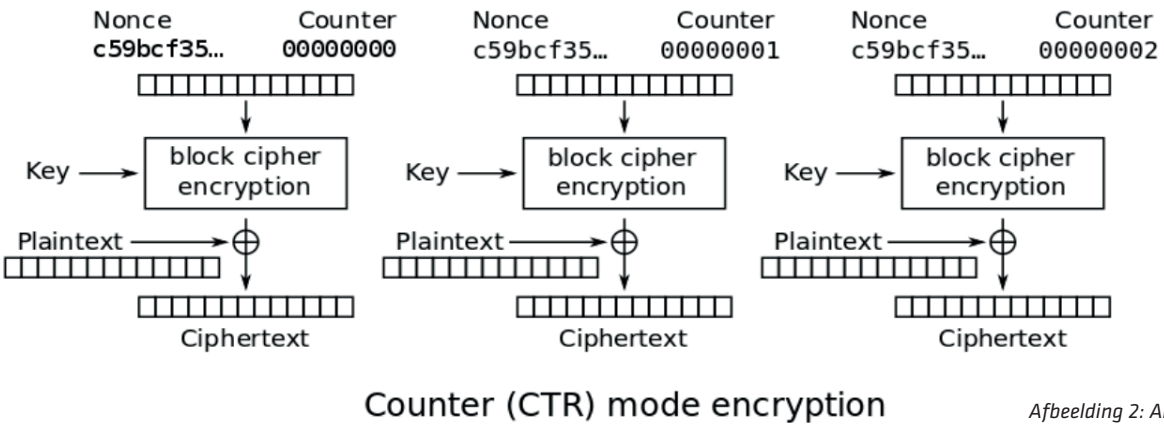
In het geval dat de encryptie en de decryptie met dezelfde sleutel gebeuren, hebben we het over symmetrische encryptie. Er zijn al-

Nanne Baars is software developer bij Xebia.

Jeroen Willemsen is een Principal Security Architect bij Xebia Security.



Afbeelding 1: AES CBC (https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation).



Afbeelding 2: AES CTR (https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation).

lerlei block en stream ciphers die dit principe toepassen. Laten we naar twee voorbeelden kijken: Advanced Encryption standard (AES) en ChaCha20.

AES

AES is één van de bekendste symmetrische encryptie algoritmes. AES beschrijft een encryptie en decryptie algoritme dat een sleutel van 128, 192 of 256 bit neemt en de plaintext encrypt of de ciphertext decrypt. Hoe het precies werkt gaat net even te ver voor dit artikel. Belangrijker is hoe dit algoritme wordt toegepast. Deze verschillende manieren noemen we "modes". Een aantal bekende modes zijn AES-CBC, AES-GCM en AES-CTR. Laten we eens kijken naar AES-CBC (AES Cipher Block Chaining), in afbeelding 1 zie je een schematische afbeelding.

In ieder blok wordt het bericht ge-XOR'ed met de uitkomst uit het vorige blok. Om het eerste blok uniek en onvoorspelbaar te maken, wordt er in het begin geen ciphertext of plaintext gebruikt, maar een block random bits. Dit block noemen we een "Initialization Vector" (IV). Aan het einde van alle operaties kom je bij het laatste blok. Hier moet het blok uiteindelijk ook 128 bit lang zijn. Is de te verwerken plaintext daar korter? Dan komt ook hier de padding om te hoek kijken.

Dit is weer helemaal anders bij AES-CTR (AES-Counter). AES-CTR is een voorbeeld van een stream cipher. Bij een stream cipher wordt er gebruik gemaakt van een keystream die per bit ge-XOR-ed wordt met de plaintext bits bij de encryptie. De keystream wordt verkregen door de block cipher toe te passen op de counter. Doordat er uiteindelijk bit voor bit geXOR-ed wordt, is er geen padding nodig (zie figuur 2). Een voorbeeld van AES-CBC kun je vinden in de Github repository.

ChaCha20

Deze stream cipher, beschikbaar in Java 11 [1] is ontwikkeld als alternatief op AES. Het voordeel van ChaCha20 is dat deze in de pure software implementatie (dus zonder specifieke hardware AES-ondersteuning) sneller is dan AES. Dit maakt de cipher dus uitermate geschikt in omgevingen zonder krachtige processor. ChaCha20 werkt op basis van 256

bits sleutels en 96 bits nonces, het is een stream cipher wat intern op blocks vertrouwt, net als AES-CTR. Zie ChaCha20 in de Github repository voor een voorbeeld.

Authenticated Encryption with Associated Data (AEAD)

Bij veel verschillende ciphers heb je nog geen integriteitsbescherming. Bijvoorbeeld bij AES-CBC: indien je een aantal bits in het versleutelde bericht aanpast, kan het best zo zijn dat je nog steeds tot een geldig, maar aangepast bericht kan komen na ontcijfering. Wil je dat voorkomen? Dan moeten we de cipher text voorzien van een message authentication code (MAC) waardoor een ontvanger kan detecteren dat het bericht is aangepast. Een veelgebruikte manier is bijvoorbeeld het toepassen AES-CBC met een HMAC, bij ChaCha20 wordt Poly1305 gebruikt hiervoor.

Bij AES-GCM (AES Gallois Counter Mode) zit deze integriteit automatisch ingebakken en hoef je geen extra MAC meer toe te passen. AES-GCM is een voorbeeld van AEAD. Dit staat voor “authenticated encryption” met “associated data”, deze vorm van encryptie biedt naast confidentiality van je bericht ook integriteit en echtheid te controleren. AEAD specificeert een aantal algoritmes waarbij de volgende input wordt verwacht:
Key: sleutel voor de encryptie

- Nonce (number used only once) een getal dat slechts eenmalig gebruikt mag worden onder dezelfde sleutel!
- Plaintext
- Associated data wordt niet opgenomen in de ciphertext maar wordt wel meegenomen in de integriteitscontrole, een aanvaller kan dit niet maar zo aanpassen.

Tijdens de ontsleuteling van het bericht wordt allereerst gekeken of de integriteit van het bericht klopt. Is er iets aangepast? Dan stopt het cipher. Is er niets aangepast? Dan wordt het bericht verder ontcijferd.
Als aanvulling kan associated data worden gebruikt om een bepaalde context aan te geven. Bijvoorbeeld: als authenticated associated data kan je een app-identifier gebruiken zodat data alleen te decrypten is op de app-instance met dat app-id.
Let wel op! Een Nonce is géén IV. Een Nonce mag je maar écht 1 keer gebruiken! Gebruik je hem vaker, dan kom je in de problemen. Meer hierover kan je lezen in [2]. Aan de andere kant: gebruik echt random IVs als je een IV nodig hebt! Indien je wel een counter gebruikt als IV, dan kan je in de problemen komen. Een voorbeeld hiervan is te vinden in onze Github repository [3].

Uitdaging: maar hoe krijg je de sleutel over de lijn?

Het grote probleem van symmetrische encryptie is: op welke veilige manier kun je de sleutel delen over een medium zoals het internet? Hier kan asymmetrische encryptie bij helpen.

Asymmetrische encryptie

Bij deze vorm encryptie hebben de verzender (voor nu even Alice) en de ontvanger (noemen we Bob) allebei twee sleutels: 1 publieke sleutel en 1 geheime privésleutel. Deze sleutels vormen een keypair. De publieke sleutel kunnen Alice en Bob met elkaar delen. Als Alice een bericht naar Bob wil sturen, gebruikt Alice de publieke sleutel van Bob en versleutelt hiermee het bericht. Vanaf dat moment is Bob de enige die het bericht kan ontcijferen omdat Bob de privésleutel heeft.

Hoe de sleuteluitwisseling in de praktijk op een veilige manier moet gebeuren, is buiten de scope van dit artikel. Je kan je voorstellen dat ook hier van alles mis kan gaan. Stel je voor dat Alice haar publieke sleutel naar Bob wil sturen en een derde partij (voor nu even Mallory) het bericht onderschept. Mallory geeft dan haar publieke sleutel in plaats van die van Alice aan Bob. Als Bob dan een sleutel wil uitwisselen met Alice, gebruikt hij de publieke sleutel van Mallory, die vanaf dan dus met de conversatie kan meeluisteren omdat ze de bijbehorende private sleutel heeft.
Hoe de sleuteluitwisseling in de praktijk op een veilige manier moet gebeuren is buiten de scope van dit artikel. Want ook hier kan van alles misgaan. Voor nu hebben we twee voorbeelden van asymetrische cryptografie: RSA (Ron Rivest, Adi Shamir, and Len Adleman) en ECC (Elliptic Curve Cryptography).

RSA & ECC

RSA is ontwikkeld in 1978 en gebruikt priemgetallen en vermenigvuldigingen mod N. Het principe is gebaseerd op het feit dat het ontbinden van priemgetallen een rekenkundig moeilijk probleem is. Het vinden van de juiste grote priemgetallen wordt gelukkig door Java voor je opgelost, zodat je uiteindelijk tot een publieke exponent en de juiste publieke N komt als publieke sleutel. RSA-X staat ook eigenlijk voor de lengte van N in bits: bij RSA-1024 is N 1024 bits lang, bij RSA-4096 is N 4096 bits lang.

OP WELKE VEILIGE MANIER KUN JE DE SLEUTEL DELEN OVER EEN MEDIUM ZOALS HET INTERNET?

```
public static byte[] signRsaPssSha512(byte[] privateKey, byte[] msg) {
   PSSSigner signer = new PSSSigner(new RSAEngine(), new SHA512Digest(), new SHA512Digest(), new SHA512Digest().getDigestSize());
   RSAPrivateCrtKeyParameters key = (RSAPrivateCrtKeyParameters) PrivateKeyFactory.createKey(privateKey);
   signer.init(true, key); //true means: sign
   signer.update(msg, 0, msg.length);
   return signer.generateSignature();
}
```

Listing 1.

ECC maakt gebruik van elliptische krommen over eindige velden en discrete logaritmes, dat net als bij RSA een moeilijk probleem is. Deze krommen zijn vastgesteld en worden gevalideerd [4]. Eén van de voordelen van ECC is dat de grootte van de sleutel kleiner is, maar wel sterker. Dit maakt ECC efficiënter en beter te gebruiken in het geval van beperkte rekenkracht. Het aantal valkuilen bij het vinden van een curve is ook groter, in het tweede artikel zullen we hier meer aandacht aan besteden.

Praktijk

Met een asymmetrisch cipher kun je per keer slechts een beperkt aantal bits versleutelen. Bijvoorbeeld met RSA-2048 kan het bericht uit maximaal 2048 bits bestaan (minus de padding). Bij ECC wordt de grootte bepaald door het veld van de curve. Hoe wordt dit nu gebruikt? Wanneer je sleutels uitwisselt kan je RSA of EC gebruiken om de symmetrische sleutel te versleutelen om deze uit te wisselen. Een voorbeeld hiervan is Elliptic-curve Diffie–Hellman (ECDH), dit is een 'key agreement protocol' waarbij de symmetrische sleutel over een onveilig medium toch uitgewisseld kan worden. Deze symmetrische sleutel wordt dan gebruikt om het bericht vervolgens te versleutelen.

Hashing

Stel je voor: je verstuurt een bericht via een onbetrouwbaar medium, hoe kan je dan een indicatie krijgen of deze niet is aangetast onderweg? In andere woorden: hoe krijg je een indicatie van de integriteit van een bericht? In onze Github repository [5] kan je het voorbeeld ChangeCipher vinden. Hierin is te zien hoe je een bericht kan aanpassen als attacker. Wil je de integriteit kunnen controleren? Dan kan dit onder andere door het gebruik van een hashing methode. In feite wordt er over een plaintext met een hashfunctie een hash berekend: H(Plaintext) = hash. De plaintext kan oneindig lang zijn, terwijl de hash altijd een vaste lengte heeft. Je voelt hem wel aankomen: als iedere plaintext in de wereld door de hash functie heen tot een hash komt met een vaste lengte, dan heb je dus ergens wel twee berichten die allebei dezelfde hash hebben. Dit noemen we een collision. Om te voorkomen dat je collisions krijgt, moet je een hash-algoritme kiezen dat een zo hoog mogelijke collision resistance heeft. De SHA (Secure Hash Algorithm) familie is een groep aan hashes die een steeds hogere collision resistance heeft. Op dit moment kunnen we dan ook aanbevelen om SHA-2 (256 bits of hoger) of SHA-3 (256) te gebruiken.

Ondertekenen van een bericht

Waar je met een hash vooral keek of de integriteit in orde was, ga je met een signature een stap verder: je valideert de integriteit van een bericht en je controleert of het bericht ook op die manier is verstuurd door de afzender. Een signature wordt namelijk gemaakt door een private key die alleen de verstuurder heeft. Je kan de signature dan weer valideren met de public key. Signatures zijn operaties die je niet op grote blokken plaintext direct kan zetten. In plaats daarvan wordt de hash van een bericht ondertekend. De ondertekening daarvan controleer je vervolgens door met de public key te valideren dat de signature klopt. Hoe gaat dit in zijn werk? Bekijk de code in listing 1: De plaintext `msg` in de code wordt hier ondertekend. Om dit te doen wordt er eerst een `PSSSigner` klasse in het leven geroepen die een hashfunctie meekrijgt om een hash over het bericht te berekenen. De andere kant kan met de publieke sleutel de signature valideren. We hebben nu alle bouwblokken beschreven en in het volgende artikel zullen we een aantal constructies uitlichten waar je op moet letten als je encryptie gaat gebruiken in productiecode. ■

VERWIJZINGEN

[1] **https://tools.ietf.org/html/rfc7539#section-1.1**
[2] **https://tools.ietf.org/id/draft-irtf-cfrg-gcmsiv-08.html**
[3] **https://github.com/nbaars/java-magazine-article/**
[4] **https://safecurves.cr.yp.to/**
[5] **https://github.com/nbaars/java-magazine-article/**

Google Cloud Run

Eenvoudig serverless webapplicaties draaien

Wil je een applicatie in de cloud te draaien, maar weet je niet goed waar je moet beginnen? Met Cloud Run heb je je applicatie in een handomdraai op de Google Cloud draaiend.

Tegenwoordig zijn er tal van mogelijkheden om je webapplicatie in de cloud te draaien. Dit vereist echter vaak kennis van infrastructuur om clusters met virtuele machines te configureren, wat ook nog eens flink geld kan kosten om draaiend te houden. Bij oplossingen met een hogere abstractie qua infrastructuur, kun je enigszins beperkt worden in welke programmeertaal en versie je kunt gebruiken, zoals bij Google App Engine of Amazon Elastic Beanstalk het geval is. Andere beperkingen kunnen zijn dat je applicatie in slaapstand wordt gezet na 30 minuten van inactiviteit, wat bijvoorbeeld bij Heroku het geval is [1]. Daarnaast is het lastig om een applicatie lokaal te testen, omdat het niet mogelijk is alle cloud diensten lokaal te emuleren.

Cloud Run is een relatief nieuwe dienst binnen het Google Cloud Platform, die veel van deze problemen oplost.

Hoe werkt Cloud Run precies?

Bij Cloud Run verpak je je service in een Docker container, waarin je je webapplicatie laat opstarten en laat luisteren naar netwerkverkeer op een bepaalde poort. Google regelt alle onderliggende infrastructuur voor je, wat deze dienst serverless maakt. Daarnaast worden een aantal zaken goed geregeld, zoals autoscaling. Als er geen verkeer binnenkomt wordt je service teruggeschaald naar nul instanties, alsmede de kosten. Mocht er plotseling veel netwerkverkeer naar je service zijn, dan wordt er automatisch opgeschaald tot maximaal 1000 servers.

Wanneer te gebruiken?

Belangrijk is wel te vermelden dat je service stateless moet zijn, omdat achtereenvolgende netwerkverzoeken door verschillende instanties afgehandeld moeten kunnen worden. Zo zul je sessies van gebruikers en andere state dus buiten de containers op moeten slaan. Verder kan je service alleen CPU gebruiken terwijl er een netwerkverzoek wordt afgehandeld. Om toch achtergrondprocessen te draaien kun je vanuit andere Google diensten zoals Cloud Pub/Sub of Cloud Tasks een verzoek naar je service laten doen.

Google Cloud voorbereiden

Om Cloud Run te proberen is het nodig een Google Cloud account te hebben. Als je deze nog niet hebt, kun je deze gratis aanmaken. Je krijgt dan 300 dollar tegoed, wat ruim voldoende is om te experimenteren. Maak een nieuw project aan, zodat je dit complete project met alle aangemaakte *resources* kunt verwijderen als je klaar bent met experimenteren. Dan weet je zeker dat er achteraf geen kosten in rekening worden gebracht.

Om de CLI te gebruiken moet je de Google Cloud SDK installeren op je computer. Als alternatief kun je de Cloud Shell [2] gebruiken, waarmee je alles vanuit de Google Cloud Console in je browser kunt doen en niks lokaal hoeft te installeren. Zorg dat je nieuw aangemaakte project geselecteerd is, en de benodigde APIs, Cloud Build, Container Registry en Cloud Run, aan staan (listing 1).

Docker image bouwen

Op Cloud Run kan een willekeurige Docker container met webserver draaien. Je kunt dus bijvoorbeeld een Java project met Spring Boot, Micronaut of Thorntail maken. Maar je kunt ook een andere programmeertaal gebruiken. Ter illustratie gaan we een simpel Spring Boot

Benjamin Komen is Software Development Consultant bij Xebia en werkt graag aan Cloud-Native applicaties.

```
gcloud config set project [PROJECT-ID]
gcloud services enable cloudbuild.googleapis.com containerregistry.
googleapis.com run.googleapis.com
```

Listing 1.

project gebruiken. Clone of download deze vanaf **https://github.com/benjaminkomen/cloudrun** en navigeer naar de map in je terminal of de Cloud Shell.

Dit project bevat een Spring Boot applicatie met slechts een hello world endpoint. Bouw deze met Docker (listing 2).

```
docker build -t cloudrun ./cloudrun
```

Listing 2.

Dit bouwt een Docker image vanuit de map "cloudrun" en geeft hem een tag met de naam "cloudrun" mee. De Dockerfile (listing 3), maakt gebruik van de zogenaamde *multi-stage build*. In de eerste fase wordt vanuit je bronbestanden met behulp van Maven een JAR gemaakt. In de tweede fase wordt deze JAR uitgevoerd.

Als Cloud Run deze Docker image draait, wordt er een PORT variabele geïnjecteerd. Als je deze Docker image zelf lokaal wilt draaien, zul je deze variabele mee moeten geven (listing 4).

```
# Fase 1: compileer een JAR bestand.
FROM maven:3.6.3-jdk-13 as builder

# Kopieer lokale code naar de container image.
WORKDIR /app
COPY pom.xml .
COPY src ./src

# Bouw een te publiceren artifact.
RUN mvn package -DskipTests

# Fase 2: voer de eerder gebouwde JAR uit.
FROM adoptopenjdk/openjdk13:alpine-slim

COPY --from=builder /app/target/cloudrun-*.jar /cloudrun.jar

# Draai de web service bij het opstarten van de container.
CMD ["java","-Djava.security.egd=file:/dev/./urandom","-Dserver.
port=${PORT}","-jar","/cloudrun.jar"]
```

Listing 3.

```
docker run -it -e PORT=8080 -p 8080:8080 cloudrun
```

Listing 4.

Je ziet in je console hoe Spring Boot opstart. Je applicatie is nu benaderbaar in je browser via **http://localhost:8080**, of als je niet lokaal maar in de Cloud Shell [2] draait, kun je klik-

ken op het 'Web Preview' icoontje naast het potloodje rechtsboven en vervolgens 'Preview on port 8080'. Om de Docker container te stoppen, type je Ctrl + C in je terminal of shell.

Cloud Build

Bij het lokaal bouwen van een Docker image, is deze alleen lokaal beschikbaar. Om de Docker image beschikbaar te maken voor Cloud Run, is het handig als deze in de Container Registry van je Google Cloud project staat. Om dit eenvoudig te doen kun je gebruik maken van Cloud Build, waarmee een Docker image gebouwd en gepubliceerd wordt in de Container Registry (listing 5). Docker images die je hier publiceert, zijn standaard niet publiek toegankelijk, maar alleen binnen je project te gebruiken.

Hierbij is PROJECT_ID het id van je project en IMAGE_NAME kun je zelf kiezen, bijvoorbeeld "cloudrun". Je kunt de gebouwde image vervolgens vinden in de Container Registry (**https://console.cloud.google.com/gcr?project=PROJECT_ID**).

Draaien op Cloud Run

Nu we een Docker image in de Container Registry gepubliceerd hebben, kunnen we een container gaan draaien op de Cloud Run (listing 6).

Hierbij is SERVICE een naam die je zelf kunt verzinnen, bijvoorbeeld "cloudrun". De PROJECT_ID en IMAGE_NAME zijn hetzelfde als wat we bij het Cloud Build commando hebben gebruikt. De *platform managed* vlag geeft aan dat we Cloud Run fully managed willen gebruiken. Dit betekent dat Google de onderliggende infrastructuur volledig regelt. Als alternatief kun je Cloud Run gebruiken in combinatie met een eigen Google Kubernetes cluster. De vlag *region*, waar "europe-west1" staat, geeft aan in welke regio hij wordt gedraaid, wat je uiteraard zo dicht mogelijk bij je eindgebruikers wilt hebben. Binnen deze regio wordt je applicatie gedupliceerd over meerdere zones [3]. De vlag *allow-unauthenticated* geeft aan dat de service publiekelijk toegankelijk via het internet beschikbaar is. De *memory* vlag is in het geval van Java en Spring Boot aan te bevelen. Deze instelling kan op maximaal 2 gibibyte geheugen gezet worden.

Je kunt de applicatie dan vervolgens bezoeken via de URL in de output van het commando.

Afbeelding 1: Stappenplan van Cloud Run deployment.

```
gcloud builds submit --tag gcr.io/[PROJECT_ID]/[IMAGE_NAME] ./cloudrun
```

Listing 5.

```
gcloud run deploy [SERVICE] \
 --image gcr.io/[PROJECT_ID]/[IMAGE_NAME] \
 --platform managed \
 --region europe-west1 \
 --allow-unauthenticated \
 --memory 1Gi
```

Listing 6.

Automatisch nieuwe versies bouwen

Een nieuwe revisie van het project publiceren hoeft niet steeds met handmatige commando's. Dit kan namelijk ook met behulp van een GitHub trigger [4]. Dit zorgt er voor dat als je een nieuwe commit naar GitHub pusht, de trigger afvuurt en er een nieuwe versie van je Cloud Run service gebouwd wordt. Hiervoor moet er een bestand genaamd *cloudbuild.yaml* aan het project worden toegevoegd, waar de volledige build pipeline in geconfigureerd staat: het bouwen met Cloud Build, releasen van Docker image naar de Container Registry en het publiceren van een nieuwe versie op Cloud Run (zie afbeelding 1). Deze YAML configuratie kan ook gebruikt worden als instructie om handmatig te deployen.

Eigen domeinnaam kiezen

De URL die je standaard krijgt voor je applicatie is niet makkelijk te onthouden. Echter, het is eenvoudig een eigen domeinnaam te gebruiken. Als je naar het dashboard in de Google Cloud gaat (**https://console.cloud.google.com/run?project=PROJECT_ID**) en boven op 'Manage custom domains' klikt, kun je binnen een aantal kliks bij Google een domeinnaam registreren en wordt deze toegevoegd. Dit kan al vanaf ongeveer 12 euro per jaar. Daarnaast kun je een elders geregistreerde domeinnaam toevoegen met behulp van DNS records.

Bekijken en beheren via dashboard

Nu je applicatie op Cloud Run draait, kun je enkele zaken op het dashboard bekij-

ken (**https://console.cloud.google.com/run?project=PROJECT_ID**). Je ziet hier een tabel met je *services*. Binnen een Google Cloud project kun je dus meerdere Cloud Run services hebben, bijvoorbeeld een backend en een frontend service die met elkaar communiceren.

Elke service heeft een lijst van revisies. Bij elke publicatie van een nieuwe versie, wordt er automatisch een nieuwe revisie aangemaakt. Dit kun je gebruiken om terug te gaan naar een eerdere revisie, of om een gedeelte van het verkeer naar de ene revisie te leiden, en een gedeelte naar een andere revisie. Daarnaast kun je performance metrics zien, zoals hoeveelheid requests, CPU en geheugengebruik. In het logging tabblad kun je de logs zien. Deze worden automatisch verstuurd naar Stackdriver, de logging dienst van de Google Cloud.

Kosten

Bij Cloud Run betaal je niet voor de onderliggende infrastructuur, maar per 100 milliseconden dat er requests worden afgehandeld door je service. Tenzij je service continue aangeroepen wordt, zal dit dus beduidend minder zijn dan betalen voor een virtuele machine die continu aan staat.

Een voorbeeld van een rekening zie je in afbeelding 2. Je ziet dat er voor CPU-gebruik, geheugengebruik, opslag (van de Docker images) en netwerkverkeer een bedrag in rekening wordt gebracht.

Product	Resource Type	Interval	Usage	Amount(€)
Cloud Run	CPU Allocation Time	Jan 1 - Jan 31	1.234 Hours	0.10
Cloud Storage	Multi-Regional Storage US	Jan 1 - Jan 31	0.659 Gibibyte-months	0.02
Cloud Run	Cloud Run Network Internet Egress Intercontinental (Excl Oceania and China)	Jan 1 - Jan 31	0.05 Gibibytes	0.01
Cloud Run	Memory Allocation Time	Jan 1 - Jan 31	0.74 Gibibyte-hours	0.01

Afbeelding 2: Voorbeeld van een maandelijkse rekening van Google Cloud Platform.

Vergelijking met concurrentie

In onderzoek van Gartner [5] wordt Google samen met Amazon en Microsoft als leider op het gebied van Cloud technologie genoemd. Het is dus interessant om af te vragen hoe Cloud Run zicht verhoudt ten opzichte van de concurrentie.

Amazon AWS

Een vergelijkbare dienst van Amazon is AWS Fargate. Hiermee kun je een Docker container draaien op infrastructuur die Amazon voor jou beheert. Hoewel AWS Fargate vergelijkbaar is met Google Cloud Run, is er wel meer configuratie nodig. Zo moet je een *Task Definition* maken met CPU, geheugen en netwerk configuratie en een *Service* die deze draaiende taak beheert. Wil je gebruik maken van bijvoorbeeld load balancing, zul je apart een AWS load balancer moeten configureren.

Microsoft Azure

Een vergelijkbare dienst van Microsoft is Azure Container Instances. Hierbij draai je een Docker container op een *Resource group* waar je CPU, geheugen en netwerkinstellingen beheert. Je betaalt voor de gealloceerde CPUs en geheugen per seconde dat je *container instance* draait. Er vindt geen *autoscaling* plaats. Om dat te bereiken kun je bijvoorbeeld Azure Kubernetes Services (AKS) gebruiken. Als je alleen ACI gebruikt, zul je dus handmatig containers moeten stoppen als je ze niet wilt gebruiken en handmatig moeten opstarten als je er meer nodig hebt.

Conclusie

Met Cloud Run kun je eenvoudig een web service in de cloud draaien, zonder kennis van onderliggende infrastructuur. Je betaalt alleen voor daadwerkelijk gebruik en kunt snel op- en neerschalen bij meer of minder vraag. Cloud Run onderscheidt zich hiermee ten opzichte van concurrentie en er is nauwelijks sprake van *vendor lock-in*. Een Docker container die op Cloud Run draait, kan je eenvoudig ook op een ander platform laten draaien. Hierdoor is Cloud Run een goede oplossing om zo snel mogelijk containers te laten draaien, waar je later nog alle kanten mee op kunt. ■

MET CLOUD RUN KUN JE EENVOUDIG EEN WEB SERVICE IN DE CLOUD DRAAIEN

REFERENTIES

[1] **https://devcenter.heroku.com/articles/free-dyno-hours**
[2] **https://cloud.google.com/shell/docs/using-cloud-shell**
[3] **https://cloud.google.com/docs/geography-and-regions**
[4] **https://cloud.google.com/cloud-build/docs/create-github-app-triggers**
[5] **https://cloud.google.com/gartner-cloud-infrastructure-as-a-service/**
[6] **https://cloud.google.com/run/docs/quickstarts/build-and-deploy**
[7] **https://github.com/ahmetb/cloud-run-faq**
[8] *Mastering Serverless Applications with Google Cloud Run* (W. Venema)

Power to the mob

One team, one computer, one codebase

I remember a time, when at every code review, I was screaming inside and dying bit by bit at every line of code. Another piece of code that has no structure and no consistency; another layer is leaking into another one—the nth solution for the same problem inside one application. Put in the mix the long development time and switching team members, and we have the perfect formula to build monsters.

This is what many software developers and I have experienced during the years. How could we eliminate at least one element from this equation?
Can we make the current team to work as a team, instead of a group of individuals that work on the same thing in parallel? Because, more often than not, team members express their different views through code, which leads to an inconsistent, unstructured, and unmaintainable project.

Then I read for the first time about mob programming: a software development approach, where the whole team works together on the same task, behind the same computer at the same time. Could this be the way to enable the team to collaborate as one mind and one soul? Is mob programming here to the rescue? Can we build a better team and make the codebase clean?
So I challenged the team to go mob!

The thought of putting 5 strongly opinionated developers working behind one laptop, at the same time, working on the same thing, made the team anxious. "That sounds ridiculous!" - they said. Well...we still did it!

Worst-case scenario, we will get an opportunity to know each other better and to shake things up from our comfortable chairs and daily routines.

The rules

We scheduled the first mob programming session, we booked a meeting room for a couple of hours, and we started as initially developed with a few easy rules.

- One computer - There is only one computer in use for programming.
- Being open and considerate toward new views and ideas and treating each other with kindness and respect.

Magdolna Szilágyi is a Software Engineer at Code Nomads with a passion for software architecture, clean code and effective teams.

- Driver/Navigator pattern: The Driver sits at the keyboard and writes the code. The Navigators discuss the idea being coded and guide the Driver in creating the code. The Driver is not involved in the discussion.
- Timed rotation: We rotate the Driver every 15 minutes, so no one is attached to the keyboard for very long.

We started our session by choosing a story to work on. Then the first Driver connected his laptop to a big screen for everybody to see it. The Navigators started the discussion on how to solve the story. There was only one problem. The Driver, who was not involved in the discussion, had the most knowledge about that specific topic. This was getting us nowhere, so early on, we decided to customize the rules to our needs. The first change was that the Driver could be involved in the discussion as well. However - essential - the Driver when typing must follow the Navigators. Otherwise, the Driver might "hijack" the keyboard and make changes without consulting with the rest of the team.

We navigated the Driver, and this was fine until another developer started typing. The typing went slow due to trying to figure out how to work with somebody else's workstation and shortcuts. We had different operating systems, different development environments and configurations. So, everybody started bringing their own laptop and worked on that.

Every 15 minutes, we were changing drivers, and because everybody was using their laptop, we were pushing and pulling the work in progress code to the repository. We found the 15 minutes rotation quite disruptive and we were not comfortable with messing up the git history. So instead of timed rotation, we switched to tasked rotation. We split the story into mini-tasks, for example: writing a unit test, refactoring a method or a class. We found this more effective, also considering that code writing to discussion ratio at times was low and we produced no code every 15 minutes.

The first session left us with mixed feelings. We had our fair share of struggles, but we have also seen the potential to increase team collaboration. One session wasn't enough to reach a conclusion. After all, we needed time and practice to learn how to do mob programming. We decided to make this a regular thing. Every 2 weeks a couple of hours or a full day, we were going mob!

The benefits

Some of the benefits were clear from the first session, and some we saw after a couple of sessions. For instance, it was evident from the first moment that this is an excellent knowledge sharing tool. We were exchanging handy tips and shortcuts to make us more productive with our IDEs. We were learning new things ranging from business rules to programming paradigms or cool new tech.

On top of that, it was a great team exercise!

THE FIRST SESSION LEFT US WITH MIXED FEELINGS. WE HAD OUR FAIR SHARE OF STRUGGLES, BUT WE HAVE ALSO SEEN THE POTENTIAL TO INCREASE TEAM COLLABORATION

We had a chance to know and work better with each other. For a new team, it is especially beneficial because it will speed up the bonding process. It gives you a feeling of belonging and an assurance that you are not alone. It will help to work more like a team instead of together in parallel. Spending time together, sharing knowledge and frustrations builds stronger connections and makes members more committed to each other and to the common goal.

During the sessions, we had moments when we realized that we had different ideas about the scope of the project, and everybody was trying to push their own "agenda" on how to build it. It was the right moment of collective reflection, especially if we were under high pressure being focused on getting things done. It is a good time to slow down and see if the team is still on the same page and is going in the right direction. It helped us to bring the team on the same page regarding the vision and plans of the project.

Our biggest struggles were communication and listening. The most challenging part was to listen and to hear others' opinion, rather than just running over our response in our head. But when we were put on the spot, we had to make our communication more efficient. We had to listen and think twice about how to express our ideas to be heard and understood. We needed to think twice and experiment on how to formulate your instructions for a junior, a senior developer, and non-techies as well.

And to my happiness, we were improving and cleaning our codebase! How? Because, as a team, we started finally picking up those low hanging fruits that we always postponed. We were discussing points and issues which otherwise were neglected during our day to day work. For example, code styles, best practices, possible improvements, and technologies. When discussing and making some agreements, we ended up expressing a unified opinion and decisions in code. The project started to look more like one great developer wrote it.

The learnings

It was not all sunshine and rainbows. Some sessions were better than the others, however, we have learned something new each time. As we progressed, we got a lot of insights on what can go wrong, how we can spend our time to get the most value out of it, and how we can improve.

For example, it was no fun to debug together. It was frustrating, difficult to follow, and disengaging. So we have learned to choose issues to solve where there is space for creativity, for instance, issues where we needed to build a new feature. It was more fun, we learned a lot and enabled us to collaborate more and better.

However, more space for creativity and freedom brings in more discussions and disagreements, which is not bad because this makes people think. But, sometimes, the conversation can get heated and frustrating. Try to moderate the discussion, and keep it civil, organized, and focused. Pay attention to interpersonal dynamics and people's reactions. If somebody is hurt, you can point it out and discuss how to make arguments without getting personal. If somebody is confused, revise the previously discussed points and comments. Being aware of these dynamics and reactions can avoid unnecessary misunderstandings and, instead, the energy can be used to find the right solution.

AS WE PROGRESSED, WE GOT A LOT OF INSIGHTS ON WHAT CAN GO WRONG, HOW WE CAN SPEND OUR TIME TO GET THE MOST VALUE OUT OF IT, AND HOW WE CAN IMPROVE

Try to involve everybody in the discussion. To include those who are naturally more quiet or shy, you might ask directly for their opinion and encourage them with body language. For instance, smile when they say something, look toward them often, and show that you value their participation.

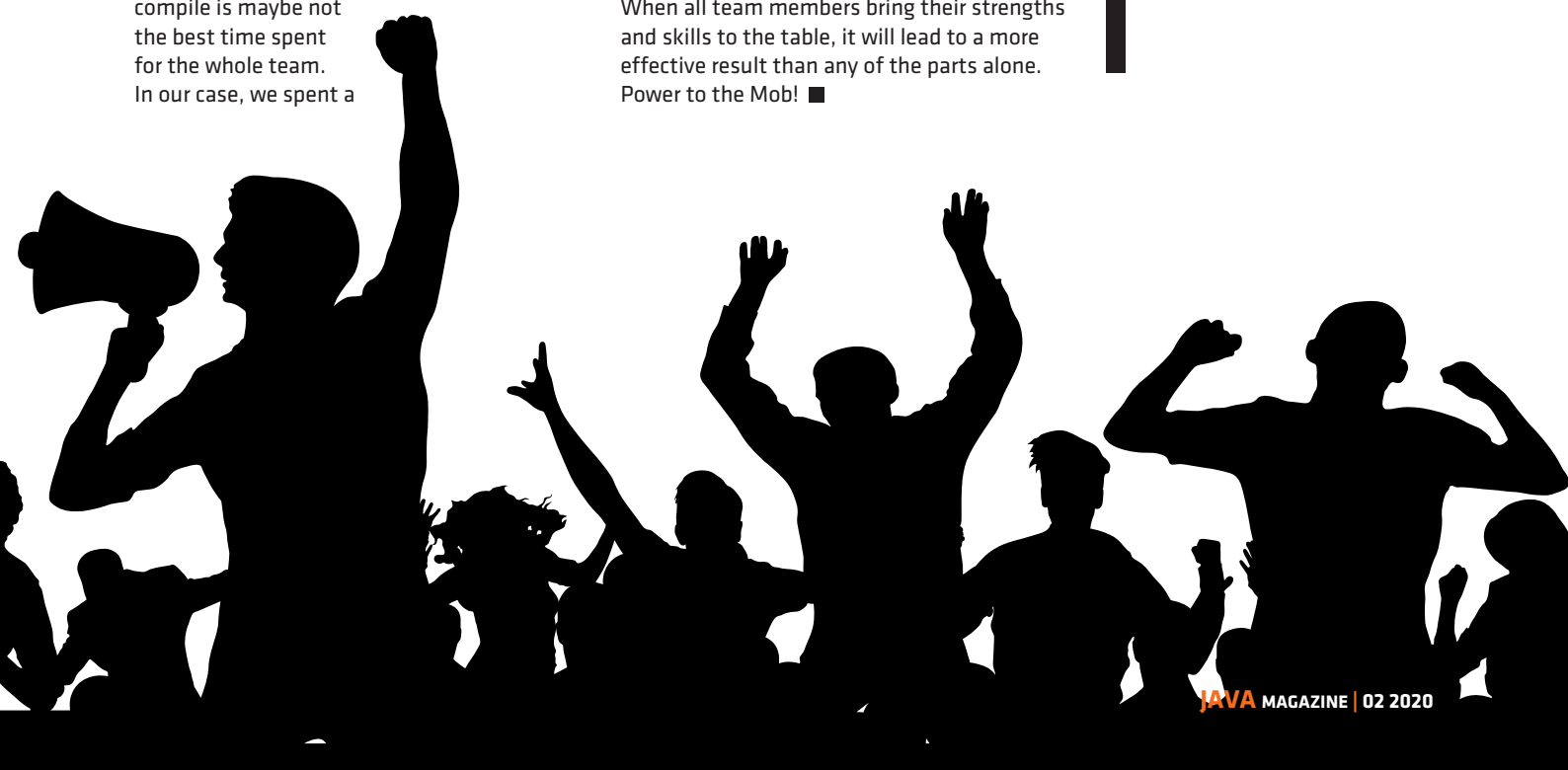
Some of the discussions can be tiring; that is why it is essential to have regular breaks. If, as a developer, you are also the facilitator and you are involved in the discussions, you might forget about the breaks. No breaks made us tired at the end of the session and less tolerant of each other. What worked for us is to set a timer for 50 minutes, and after the timer went off, we took a 10 minutes break. An excellent way to start up after the break is with some fun games! To keep the energy levels high, have some icebreakers and energizers prepared. This can light up the mood, boost team morale, and make people open to discussion. Games, as such, can be guessing each other's favorite music, untangling yourselves or any other you find suitable and fun. If nothing else works, then bring some cookies! Everyone is happy when cookies are involved, and they are there as ultimate comfort.

After you have had all the discussions and you start typing, I recommend keeping your code changes scoped. If you intend to do a refactoring, think twice about what value brings to the session. For example, making the code compile is maybe not the best time spent for the whole team. In our case, we spent a considerable amount of time making the code compile due to a change to a variable type of a heavily used domain class. What worked for us was test-driven development (TDD). Since we are doing it for a short amount of time and we have a lot of discussions, most of the time, we do not manage to finish a full story. However, by doing TDD and seeing the test going green, it gives us a feeling of accomplishment

If your first mob session wasn't as expected, don't give up. If it is uncomfortable, that's great! It means you are out of your comfort zone and starting to learn something new. Keep doing them regularly. The flow of the session and team collaboration will improve from session to session. Repeat, customize, and enforce the rules. They really do work. And always ask for feedback at the end of the session. This will help you to identify the problems, and for the next time, you can find a solution.

Conclusion

Programming is more about thinking than actually writing code. The mob sessions allow a variety of ideas to be expressed and discussed. All members have the opportunity to think about and respond to them. When there is a mix of ideas on the table it is more likely to come up with a great solution. This leads to group ownership and commitment because everyone has a chance to contribute to the discussion and to be heard. The final result feels like it belongs to everyone. When all team members bring their strengths and skills to the table, it will lead to a more effective result than any of the parts alone. Power to the Mob! ■



JSR-381: A Standard Java API for Visual Recognition

Using Machine Learning

At this point, most of us know Machine Learning (ML) as recognizing patterns in your data, anticipating the likely future behavior of similar data and taking steps to maximize benefits based on those predictions. It represents a massive change in how we solve problems and has long-term effects to the very foundations of computing and to the world in general.

Most ML toolkits require you to learn Python. Python is one of the leading programming languages in data science and ML. But there are 10 million Java developers on the planet that need to build world-class production ML applications that leverage our years of expertise in development and deployment.
This article will cover a deep learning introduction and create a ML model that will be able to classify Star Wars aircrafts in just a few lines of code using Java Specification Request #JSR-381.

JSR-381

The goal of JSR-381 is to develop a standardized Java-friendly API which can be used to solve small and complex tasks by applying ML specifically to visual recognition use cases. Each ML engine (e.g. TensorFlow & Deep Netts) has its own way of accepting input. It is a relatively large effort to standardize this type of input and therefore it won't be covered in this JSR. For now, the following features will be included into JSR-381 to simplify training and using the ML model:

- **Binary classification** to classify input which has the outcome true/false, yes/no, dark/bright, etc;
- **Image classification** to classify an image as input and the outcome of two or more trained labels, depending on the model;
- **Object detection** to detect and classify one or more fragments in the image as input and one or more trained labels as outcome;
- **Training tools** to build the ML model for binary classification, image classification and object detection.

At the same time as the development of the API, the first implementation of the API will be developed as well. This is the so called "Reference Implementation" (RI) which will meet all the requirements of the API as prescribed by the Java Community Process (JCP). The RI of JSR-381 uses Deep Netts, an open source deep learning engine written in pure Java. Amazon has its own implementation of JSR-381 which will support TensorFlow, Apache MXNets and other upcoming engines using their abstraction API called Deep Java Library (DJL).

Supported algorithms

Not every developer is a mathematician. Neither am I. With JSR-381 you won't have to understand the underlying math of these algorithms because it's developed to allow you to use common Java design patterns instead of generated wrappers for native libraries. JSR-381 will only support neural networks by default and exposes a high level API to use. Each implementation of JSR-381 will have to support neural networks but may also expose other algorithms and implementation specific algorithms to solve complex problems, which are better solved by algorithms other than neural networks. This way JSR-381 can be used by beginners and advanced ML Java developers alike.

Quick dive into understanding classification and deep learning

Classification is commonly used to determine whether the given input is answer A, B or even C. For each classification model, it needs a given set of labels and a prepared dataset. A part of preparing the dataset is labelling the data. For example: a picture of a cat is labeled as cat and a picture of the dog is labeled as dog. This way the ML model will be able to

Kevin Berendsen is software engineer at OpenValue, maintainer of JSR-381 and passionate about bringing Java and ML together.

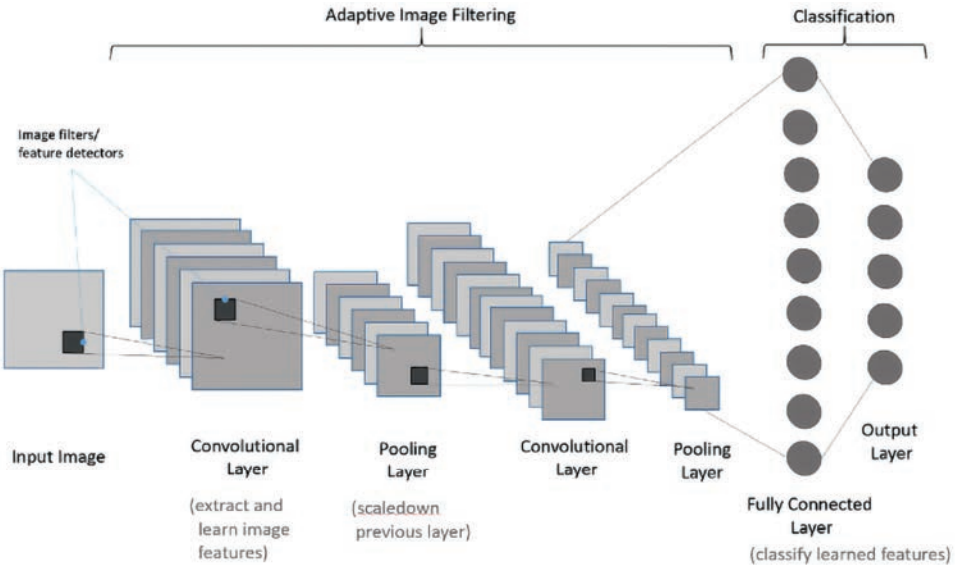


Image 1: Architecture of convolutional neural network

learn and tell whether a newly given picture has to be classified as a cat or a dog. As previously explained, binary classification will only contain two labels which will be complete opposites of each other. The scenario of the dog and the cat is binary classification as well. But it might also need to classify all bird pictures as birds. Afterwards the dataset and labels will be expanded with new data and the model will be retrained to classify cats, dogs and birds. This is called multiclass classification. MNIST is the best-known multiclass classification ML model. MNIST is able to classify numeric characters from 0 to 9. Every ML engine has an example using MNIST because the dataset is free to download. Furthermore, the MNIST dataset has already been prepared to be trained with high accuracy.

Deep learning

Deep learning is a specific ML technique based on using multiple neural network layers. There are different algorithms but the one that's often applied for image classification is called **convolutional neural network**. This type of algorithm has adaptive image filtering through network layers, then performs the classification on the trained model and results into labeled output like you would with any classifier.

Use the Force to Learn about ML

Now that we have a basic understanding of classification and deep learning, let's create a critically important application example in which it identifies two aircrafts from the Star Wars franchise. In this article we're building a binary classifier to solve this problem. The reason why we're not using a multiclass classifier, is because we're only identifying the input image as the Millennium Falcon or the Tie Fighter. For those who are not familiar these names, shame on you! These are the most iconic aircrafts that I'm talking about ➡

Dataset preparation

As mentioned, each ML model will start by gathering the dataset. This can be a very time consuming process, therefore we will skip this part for now. We'll use a prepared dataset containing only 60~ images for each label. Normally you'd want to create a dataset that contains many more images but it will work just fine for our use case. The general rule of preparing the dataset is to keep the amount of images equal to each label, otherwise you might create a biased ML model. Our dataset contains a labels file, a training file which maps the images to a label and a folder of images.

Defining the neural network architecture

The image classifier demands a network architecture to train the ML model. Picking the right architecture is important. Depending on the needs of the project, you might want to pick a smaller architecture to reduce training time and still train a model that will have a good accuracy. Because we have a really small dataset and picking the largest architecture won't make any difference in the resulting accuracy, we will use the architecture that is used in the MNIST example.

Tie Fighter

Millennium Falcon

DEPENDING ON THE NEEDS OF THE PROJECT, YOU MIGHT WANT TO PICK A SMALLER ARCHITECTURE TO REDUCE TRAINING TIME

Training the ImageClassifier

Using JSR-381 exposes the ImageClassifier using neural networks. We start by first creating a new builder instance:

```
ImageClassifier<?> classifier =
NeuralNetImageClassifier.builder()
        …
        .build();
```

It's important to let the builder know which class type you wish to perform classification on. Because not all implementations will be able to classify that input class with the exception of the BufferedImage class. That one is supported by all implementations so we can safely use it:

```
ImageClassifier<BufferedImage>
classifier =
NeuralNetImageClassifier.builder()
        .inputClass(BufferedImage.
class)
        …
        .build();
```

You can see the difference between the snippets where the generic type of the resulting ImageClassifier is changed. There's no need to cast the resulting ImageClassifier yourself because that's been taken care of within the API.

The dataset configuration is coming next. All images within the dataset should be normalized. This means that all images should have the same width and height. Depending on the implementation, the implementation might normalize the images as a preparation step but this will definitely slow the process of training the actual ML as it's a part of preparing the dataset. Increasing the size will allow better accuracy most of the time but it also requires more computing time so training the model will take longer.

The MNIST example has an image width and height of 28 pixels. In this use case with these few images in our dataset, setting the width and height to 128 pixels was showing better results. Bigger resolutions will contain more content and information that might improve the ML model while training the neural network (Listing 1).

Afterwards, we specify the location of two files. One file for all the labels that you want to use and one that contains the mapping of all images with the corresponding label (Listing 2).

```
ImageClassifier<BufferedImage> classifier = NeuralNetImageClassifier.
builder()
        .inputClass(BufferedImage.class)
        .imageWidth(128)
        .imageHeight(128)
        ...
        .build();
```

Listing 1

```
ImageClassifier<BufferedImage> classifier = NeuralNetImageClassifier.
builder()
        .inputClass(BufferedImage.class)
        .imageWidth(128)
        .imageHeight(128)
        .labelsFile(new File("..."))
        .trainingFile(new File("..."))
        ...
        .build();
```

Listing 2

```
ImageClassifier<BufferedImage> classifier = NeuralNetImageClassifier.
builder()
        .inputClass(BufferedImage.class)
        .imageWidth(128)
        .imageHeight(128)
        .labelsFile(new File("...txt"))
        .trainingFile(new File("...txt"))
        .networkArchitecture(new File("...json"))
        .exportModel(Paths.get("starwars.dnet"))
        .maxEpochs(1000)
        .maxError(0.3f)
        .learningRate(0.01f)
        .build();
```

Listing 3

The architecture of the neural network is defined in a JSON file and copied from the MNIST example. Because we don't want to change the architecture, there are two layers which have to be changed. The input layer has to be set to the same image width and height of 128 pixels and the output layer should return two labels, because we're only classifying the Millennium Falcon and the Tie Fighter. We pass the architecture file to the builder (See image 2).

```
        …
        .networkArchitecture(new
File("...json"))
        …
```

Next, we export our model to a file and set that parameter as well. It's completely optional, but training the model may take some time and you don't want to retrain the model each time you wish to use it.

```
        …
        .exportModel(Paths.
get("starwars.dnet"))
        ....
```

Finally, we have to set the training parameters. These are parameters that have to be experimented with to "guess" which settings suits the model in order to get the best results in a decent amount of training time. Training this model only took a couple of minutes. This is relatively fast and training it with more iterations didn't show any significant difference in the accuracy.

The maxEpochs setting sets the maximum amount of iterations before stopping the model's training process (Listing 3). The maxError setting is used to stop the training once it hits the error rate of the training process. The learningRate setting is how fast you wish to train the model. With a lower rate it takes more time to train the model but that doesn't have to improve the accuracy. Meaning, it will have more iterations/epochs and takes more time to hit the maximum error rate.

Put the trained model to use

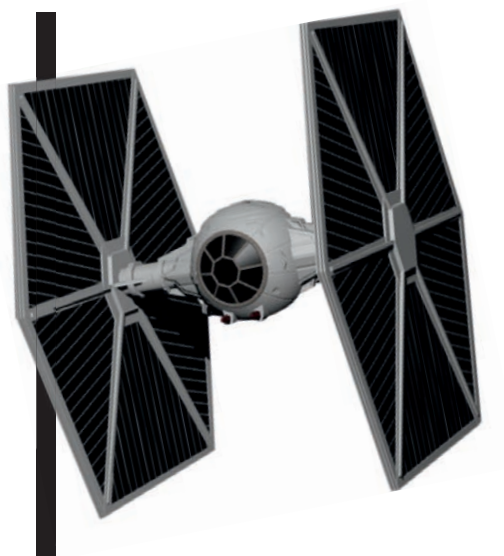
Finally, we've trained the model! Now I'd like to classify an image that was not used in the dataset and see if my model can identify it. I randomly took images of the Tie Fighter and the Millennium Falcon from Google search and ran it into the model. The ImageClassifier accepts the generic type, InputStream or File as input:

```
BufferedImage tieFighter = …;
Map<String, Float> results = imageClassifier.classify(tieFighter);
System.out.println(results)
```

The console would output similarly to:

```
{ "tie_fighter": 0.78f, "millennium_falcon": 0.22f }
```

The accuracy is approximately 80 percent which shows a decent distinction between a Tie Fighter and a Millennium Falcon. I wouldn't want to rely on 80 percent accuracy during an intergalactic dogfight, but it's good enough for our learning purposes. In a future version of the API, additional statistical evaluation should be performed after training the model to determine the accuracy.



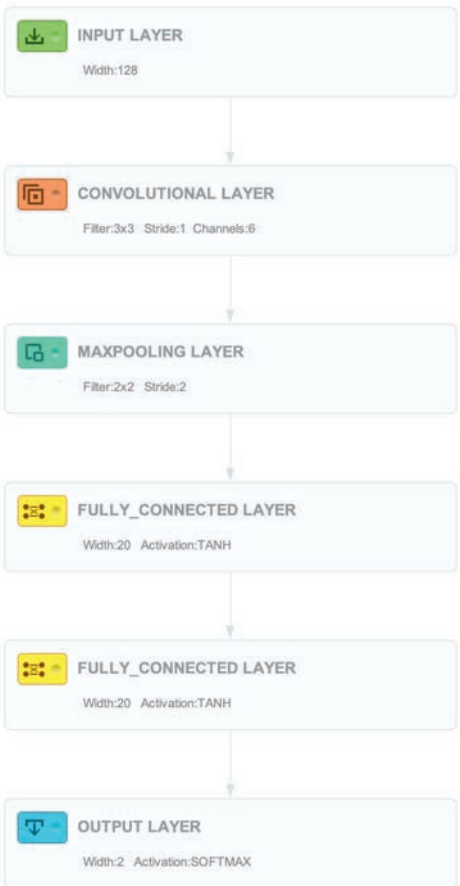


Image 2: Architecture visualized using DeepNetts Platform

Summarizing what's learned and what's upcoming

The article described the goal of JSR-381, taught a basic understanding of image classification and deep learning and allowed you to create a ML model step by step by using only JSR-381. In case you only want to run the code, go to the GitHub repository at **github.com/JavaVisRec/jsr381-starwars**

What's to be expected in the future

Many features beyond the current set are in the planning stages for JSR-381. The following major features are very likely to be added in the upcoming versions.

- Export models as ONNX (Open Neural Network Exchange);
- Model evaluation;
- Object detection;
- Transfer learning.

Become a part of the VisRec community

The VisRec working group is always looking for anyone to help, test and develop features for the next versions of the API. Feedback is also always welcome. Visit **https://groups.io/g/visrec** ■

Atlanta DevNexus 2020

Grootste Java-conferentie van Amerika

Het wordt wel de grootste Java-conferentie van Amerika genoemd: DevNexus in Atlanta (Georgia, VS)! Veel ervaren sprekers, alle grote partijen aanwezig, en een gezellige groep collega's (Blue4IT) om de stad mee te verkennen. En zelf mocht ik als spreker ook bijdragen met mijn talk over IDEs. Alle elementen waren dus aanwezig voor een gave trip!

Dag 1: Workshops

Er zijn nog zoveel features die we nog te weinig kennen. Jeanny Boyarsky (bekend van de OCA 11 boeken) ging met haar talk 'Java Idioms' in op de meest voorkomende herhalende condities. Van simpele maar effectieve methoden `repeat()`, tot aan de manieren om een collectie aan te maken met `Set.of()` en `List.of()`. Daarnaast ging ze dieper in op (advanced) streams. Had je ooit wel eens gebruik gemaakt van `iterate()` of `generate()`? Nee? Genoeg te leren dus!

Dag 2: Conference

Op het podium vonden we veel sprekers die bij Pivotal (VMWare) werken. Eén van de topics die zij behandelden was Spring Security. De main takeaways:
1. Gebruik geen `csrf.disable()` in je Spring Boot configuratie.
2. Schakel geen headers uit, anders kunnen hackers met behulp van (verborgen) iframes je HTTP methoden aanroepen (met jouw autorisaties), ook wel 'Clickjacking' genoemd.
3. Wees zuinig op je sessie data, pas op met gedeelde computers (Session Fixation).
Ook vanuit Snyk (Brian Vermeer) werd er gehamerd op security. Wist je bijvoorbeeld dat er een plugin in je IDE (IntelliJ IDEA en Eclipse) is die je hele dependency tree kan scannen op security vulnerabilities? Het is belangrijk om je code secure te blijven houden developers!

Dag 3: Conference

Wat ons opviel vanuit een volle zaal, was de talk over Quantum Computing. In plaats van de welbekende skippybal van Roy van Rijn bij JFall, gebruikte Jessica Pointer dit keer een donut bij de uitleg van een qubit. De donut kan meerdere "states" hebben. Een normale computer houdt alleen rekening mee met twee posities (voorkant en achterkant). Een Quatum Computer ziet daarnaast ook een super positie (denk dan aan een draaiende donut), dit noemen we dus een qubit. Door deze andere manier van "denken" zijn berekeningen sneller gemaakt, dit belooft veel voor de toekomst!

Een ander onderwerp als we kijken naar de toekomst is die van het gebruik van IDEs. Waar nu IntelliJ IDEA vrijwel de markt domineert voor de JVM Developer**, lijkt het erop dat Eclipse een andere koers gaat varen. Zij proberen de markt te veroveren met diverse cloud oplossingen.Zo draait Eclipse Che geheel in de cloud, en bouw je met Eclipse Codewind je gehele code tegen een Kubernetes container die identiek is aan je productie omgeving.

De after party (en verder)

Een leuke afterparty kon daarnaast niet ontbreken in 'Downtown' Atlanta. In een café (vol met Arcade kasten) hadden we de eer om met een drankje alle gave spellen uit het verleden te spelen. En natuurlijk om al die mooie ervaringen te delen met je collega's en mede-sprekers. Mooi toch?! En hier stopt de lol nog niet eens. Want in het weekend dat volgde, gingen we naar een NBA basketbalwedstrijd (die vanuit achterstand, in de laatste 5 minuten door de Atlanta Hawks werd gewonnen) en naar het grootste aquarium van de wereld. Al met al een onvergetelijke ervaring. Tot volgend jaar!

Ko Turk, Java Developer bij Blue4IT

NLJUG Academy

Als NLJUG willen we graag het onderwijs (organisatie, student en docent) meer betrekken bij de prachtige community die de NLJUG heet. Daarnaast willen we ook graag verkennen hoe we als vakgroep meer kunnen betekenen voor het onderwijs, dat natuurlijk de mooie uitdaging heeft om de dynamische development wereld te vatten in een onderwijsprogramma. Om dit te kunnen bereiken, hebben we vanuit de NLJUG het initiatief de **NLJUG Academy** opgezet. Bas geeft als NLJUG Community Developer en kartrekker van de NLJUG Academy elke Java Magazine een update over dit mooie initiatief.

Het doel van de NLJUG Academy is om onze Java community, het bedrijfsleven en onderwijs dichter bij elkaar te brengen en daarmee het gat tussen bedrijfsleven en onderwijs te verkleinen. De basis om dat voor elkaar te krijgen is het in het leven geroepen Kennispartnerschap van de NLJUG. Waarbij ik overigens trots kan vermelden dat er in de afgelopen periode twee nieuwe Kennispartners zijn aangesloten bij de NLJUG: Codam en de Haagse Hogeschool! Samen met de Hogeschool Utrecht, de Hogeschool Amsterdam, de Hogeschool Rotterdam en NOVI Hogeschool hebben we daarmee als NLJUG Academy inmiddels al een flinke groep Kennispartners!

Via die Kennispartners kunnen we op meerdere manieren studenten, docenten, developers, vakgebied en bedrijfsleven bij elkaar brengen. Zo kunnen we als NLJUG Academy:
Studenten en docenten betrekken bij onze gave events, waarbij ze op een leuke manier actuele inhoud meekrijgen en in contact komen met bedrijven;
Faciliteren dat docenten met bedrijven in contact komen om te sparren over curricula om te zorgen dat die nog beter aansluit;
Gastcolleges organiseren via developers uit de community of via Business Partners;
Dit magazine delen met studenten en docenten zodat zij de toffe artikelen uit dit magazine lezen en meekrijgen!

De Kennispartners spelen daarmee een belangrijke rol in het verkleinen van het gat tussen onze mooie Java community en het onderwijs. Daarom zijn we altijd op zoek naar meer Kennispartners om aan te laten sluiten. Vandaar deze oproep: mochten jullie iemand kennen, of zelf iemand zijn die bij een onderwijsinstelling aangesloten is, laat het me weten. Dan kunnen we kijken of we de mooie groep Kennispartners van de NLJUG uit kunnen breiden, en daarmee meer studenten en docenten kunnen laten aanhaken bij onze gave community!

Met deze mooie ontwikkelingen is de kans groot dat je binnenkort (wanneer het weer kan) studenten gaat tegenkomen op de NLJUG events. Neem onze toekomstige collega's vooral enthousiast mee in ons mooie vakgebied, zou ik zeggen.

We hebben twee nieuwe Kennispartners mogen verwelkomen de afgelopen periode! Codam (codam.nl) in Amsterdam en de Haagse Hogeschool (https://www.dehaagsehogeschool.nl/).

De huidige Kennispartners van de NLJUG:

Vanuit de NLJUG Academy geven we ook gastcolleges bij onze Kennispartners. Ook een keer een gastcollege geven? Laat het weten door een mailtje te sturen naar Bas.knopper@nljug.org.

Bas W. Knopper is NLJUG Community Developer

NLJUG INNOVATION AWARD 2020

JAVA DEVELOPERS MAKEN HET VERSCHIL
IN INNOVATIEVE TECHNOLOGISCHE ONTWIKKELINGEN

SCHRIJF JE NU IN

De NLJUG reikt dit jaar voor de derde keer tijdens de 17e editie van J-Fall de NLJUG Innovation Award uit. De businesspartners van de NLJUG kunnen tot de sluitingsdatum: 7 oktober 2020 weer innovatieve projecten en/of kandidaten voordragen voor een award. Het kan daarbij gaan om projecten die succesvol, innovatief, verantwoord en/of omvangrijk zijn, welke uitgevoerd door onze businesspartners zelfstandig of in samenwerking/opdracht van haar relaties.

INSCHRIJVING

Vanaf nu tot en met zaterdag 7 oktober 2020 kan een project en/of persoon worden voorgedragen. Ken jij een bedrijf, project en/of persoon die dit jaar de NLJUG Innovation Award zou moeten winnen?

Schrijf je in op nljug.org/nljug-innovation-award/

Dilemma's

Joop Lanting
is vaste columnist
voor Java Magazine

Een dilemma is een situatie die dwingt tot een beslissing. Er zijn zwakke dilemma's die je kunt negeren omdat niets doen onschuldig lijkt of omdat je de eventuele gevolgen daarvan accepteert. Voorbeeld: je komt na een bezoek aan de supermarkt tot de ontdekking dat je het statiegeld bonnetje niet hebt ingeleverd. Dat kan alleen dezelfde dag nog, dus je moet terug. Maar je kunt ook doorlopen naar huis.

En er zijn harde dilemma's waarbij je gewoon niet om een beslissing heen kunt, soms doordat de tijd beperkt is, soms omdat niets doen fatale gevolgen zal hebben.
Voorbeeld: je hebt hevige pijn op de borst. Je kunt daarmee naar de huisarts, die kan oordelen dat je naar het ziekenhuis moet, wat je hele planning in de war gooit. Je kunt ook wachten op een hartinfarct, dan is de overlevingskans gering en volledig herstel uitgesloten. Ik spreek uit ervaring.

Het begint al zodra je kunt lopen. Al jong moet je kiezen tussen risico's nemen of uitgelachen worden. Als programmeur maak je ook dilemma's mee, bijvoorbeeld: leveren we het project op tijd op, zonder compleet ontwerp en documentatie, of maken we die zaken eerst in orde? In het eerste geval is zowel de projectleider als de klant aanvankelijk tevreden, maar owee wanneer de gebruikelijke wijzigingen of
uitbreidingen zijn verkocht: dan begrijpt niemand dat je zonder goede beschrijvingen vruchteloos door duizenden regels programmatekst heen moet zoeken naar structuur en aanknopingspunten en je gaat over elke deadline heen. Hoe leg je uit dat geen mens alle sources en gebruikte componenten kan onthouden? 'Weet je niet meer hoe het in elkaar steekt? je hebt er toch zelf aan gewerkt?'
In het tweede geval ben je zelf als enige tevreden, want je bent voorbereid op het verwachte meerwerk. Maar wanneer de klant om welke reden ook niets meer bijbestelt 'komt het door jouw late oplevering'.

Tot zover is het de dagelijkse gang van zaken, dat geldt praktisch voor iedereen, maar het kan erger:

- In mijn jeugd schijnt het volgende verhaal in de Telegraaf te hebben gestaan: Een kind moest een operatie aan het onderlichaam ondergaan. Daarvoor werd hij plaatselijk verdoofd. Na de operatie zei hij tegen zijn ouders: "De dokter was stom dronken". Dat werd afgedaan als fantasie ten gevolge van de doorstane emoties. En men durfde ook niet te informeren bij het ziekenhuis. Wat later werd de chirurg ontslagen wegens alcoholverslaving en het kind bleek te zijn verminkt.

- Op 3 Juni 1998 reed een ICE (hogesnelheidstrein) ter hoogte van het Duitse Eschede. Plotseling boorde een stalen balk, zo leek het, zich door de vloer tussen twee stoelen. De ontstelde reizigers waren aanvankelijk verlamd van schrik. Vervolgens rende een hunner door de trein en haalde een conducteur. Deze wist ook niet wat hij ermee aanmoest en ondernam geen actie. Een paar kilometer verder stak het ondereind van de balk in een wissel en ontspoorde de trein waarbij de voorste rijtuigen te pletter sloegen tegen een viaduct. Resultaat: 101 doden (waaronder een paar arbeiders buiten de trein) en 88 gewonden. De oorzaak was een gebroken wielband. Treinwielen bestaan uit een ijzeren schijf waaromheen een stalen band is aangebracht waarop de trein feitelijk rijdt. Dilemma: niemand, ook de conducteur niet, had aan de noodrem durven trekken (misbruik wordt gestraft) en in Duitsland (gezag is gezag) is men té gelaten om zo'n beslissing te nemen.

- Op 10 februari wierp een Fatah-terrorist een handgranaat in het busje dat ELAL passagiers naar een gereedstaand vliegtuig, vlucht AI-435, van München naar London zou brengen. Een van de passagiers, Aryeh Katzenstein, bedacht zich geen seconde en wierp zich op de granaat, die hem doodde en de anderen verwondde. Een onmogelijke, heldhaftige beslissing: wachten tot iemand anders zo moedig is betekent zeker de dood van alle passagiers, kortom de keuze tussen dood en dood. Een duivels dilemma.

;JOOP!

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 plug-ins.

Bills of Materials (BOM)

Een aantal keren heb ik naast Ray Tsang op heb podium gestaan om hem tijdens zijn presentatie "Surviving Dependency Hell With Apache Maven" te ondersteunen. Hierin komen tal van voorbeelden voorbij van dependency gerelateerde problemen. Doel is voornamelijk om inzicht te geven in hoe dependency resolution en de classpath werken.
Eén van de adviezen is om ervoor te zorgen dat versies van een bepaalde groep dependencies altijd in sync zijn. Denk bijvoorbeeld aan de dependencies van org.springframework, deze horen allemaal dezelfde versie te hebben.
De klassieke manier om dit op te lossen is met een placeholder voor de versie:

```
<properties>
   <spring.version>1.0.0</ spring.version >
</properties>

<dependencyManagement>
 <dependencies>
   <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-beans</artifactId>
     <version>${spring.version}</version>
   </dependency>
   <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-core</artifactId>
     <version>${spring.version}</version>
   </dependency>
  …
 </dependencies>
</dependencyManagement>
```

Het goede aan deze oplossing is dat je op één plek voor alle dependencies kunt bepalen welke versie je wilt gebruiken. Toch kan het helaas nog steeds gebeuren dat je dependencies van org.springframework met een andere versie binnen krijgt. Stel een dependency gebruik spring-jdbc versie 2.5.6, terwijl jij spring.version op 5.2.4.RELEASE hebt en spring-jdbc niet in dependencyManagement hebt staan. In zo'n geval krijg je de oude versie van de spring-jdbc jar.

Kern van het probleem is, dat je nooit zeker kunt weten of de lijst met managed dependencies compleet is. Er zijn uiteindelijk twee oplossingen mogelijk: met de requireSameVersion rule van de maven-enforcer-plugin kun je de build laten falen als de geconfigureerde dependencies van versie verschillen.

```
<requireSameVersions>
  <dependencies>
    <dependency>org.springframework</dependency>
  </dependencies>
</requireSameVersions>
```

Dit is echter meer een pleister dan een solide oplossing. Een mooiere oplossing in de Bills Of Materials, kortweg bom-file. Zo'n bom-file is een pom, maar dan met een uitputtende lijst van managed dependencies die qua versie bij elkaar horen. Je zou deze lijst kunnen copy/pasten in je eigen pom, maar eenvoudiger is het om de bom-file als volgt op te nemen:

```
<dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-framework-bom</artifactId>
        <version>5.2.4.RELEASE</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
```

Let er vooral op dat dit een managed dependency is. Naast de groupdId, artifactId en version zie je 2 elementen: type en scope. Voor dependencies is de default type 'jar', maar die willen we niet: vandaar dat hier pom staat. En de scope moet 'import' zijn, zodat Maven weet dat hij het blok 'dependencyManagement' uit de pom-file moet overnemen.

Dit betekent wel, dat de ontwikkelaars van dat project (in dit geval springframework) tevens een bom-file moeten leveren. En zo'n bom bevat alleen(!) de set van modules, die bij elkaar horen. De volgorde van deze import-scoped dependencies maakt uit, dus als er ineens een gedeelde library in staat (bijv. slf4j-api) dan kan het vervelende gevolgen hebben.

Voor meer tips, kijk op https://jlbp.dev/

Java-developers delen kennis binnen en buiten Rijksoverheid

‘Innovaties op het gebied van Java houden we bij de Belastingdienst nauwlettend in de gaten. Nieuwe technieken testen we. Developers kijken of ze geïntegreerd kunnen worden. Maar omdat we werken voor burgers en ondernemingen kiezen we wel voor ‘proven technology’. De Java-developers van de Belastingdienst werken in meer dan 100 multidisciplinaire scrumteams aan complexe systemen. Voor het opleveren van een eindproduct is veel afstemming nodig. Dat maakt het werk ook zo uitdagend. We vinden het heel belangrijk dat onze Java-ontwikkelaars zich verder kunnen specialiseren. Daarom kunnen zij opleidingen volgen bij de IT Academie. Ontwikkelen gebeurt ook door kennis te delen. 300 Java-developers van de Belastingdienst ontmoeten elkaar in een ‘community of interest’, de vakgroep Web-Java. Rijksbreed gebeurt dat ook. 4 keer per jaar komen ‘Javanen’ van o.a. de Dienst Uitvoering Onderwijs (DUO) in Groningen, het Centraal Justitieel Incassobureau (CJIB) in Leeuwarden en Defensie in Den Haag samen rond de Rijksbrede JavaTafel om inzichten en ervaringen met elkaar te delen. En zelfs buiten de Rijksoverheid weten ‘Javanen’ elkaar steeds beter te vinden. Samen met het Kadaster, Luminis en Profit4Cloud organiseert de Belastingdienst als Apeldoorn JUG regelmatig (online) kennisdelingsessies. Het platform dat we hiervoor gebruiken, Meetup, telt inmiddels bijna 900 leden. Ook trekken we op met de Nederlandse Java User Group, de NLJUG. Kennisdelen is zó belangrijk. Stel dat we bij de Belastingdienst overstappen op een nieuwe tool. De ervaringen die we opdoen kunnen we heel gemakkelijk met elkaar delen. Daar worden we allemaal beter van.’ ■

Anton de Ruiter
is vakcoördinator Java bij de Belastingdienst

Van het bestuur

Het gerammel van de brievenbus, de lichte plof van dit Java Magazine op je deurmat, gelijk oprapen, snel doorbladeren en ernaar uitkijken om hem straks uitgebreid te lezen. Even een vertrouwd moment in deze bijzondere en bizarre tijden.

In korte tijd is er om ons heen een hoop veranderd. Het enige dat vaststaat is dat we voorlopig zullen moeten leren omgaan met deze constante verandering.

Als NLJUG bestuur proberen we hier op de juiste manier op in te spelen. Aan de ene kant door goed te kijken naar de impact van deze veranderingen op bijvoorbeeld de NLJUG events zoals J-Spring en J-Fall. Aan de andere kant door vast te houden aan waar we voor staan en nieuwe manieren te zoeken om onze gezamenlijke passie voor het Java-vakgebied te delen.

Dit betekent dat de fysieke editie van J-Spring op 1 juli niet door gaat. We werken hard aan een online alternatief. Volg het @nljug Twitter-account om op de hoogte te blijven van de laatste initiatieven en ontwikkelingen.

Voor het laatste nieuws kun je ook terecht op de NLJUG website. Wie weet is dat bezoek gelijk een mooi moment om wat afleiding te zoeken. Door terug te bladeren in oude edities van het Java Magazine of een aantal video's van eerdere edities van J-Spring of J-Fall te bekijken.

Verder is er online genoeg te doen. Er zijn veel online trainingen beschikbaar, bijvoorbeeld bij Plurarsight, Udemy of Coursera en de sessies van de Virtual JUG zijn ook echt een aanrader.

Eigenlijk zijn we allemaal op zoek naar een eigen manier om met deze situatie om te gaan en onze gedachten te verzetten. Vaak zoeken we afleiding gerelateerd aan ons mooie vakgebied. Door het volgen van een online training, het schrijven van een blog of een artikel voor Java Magazine (mail even naar **info@nljug.org** 😊), een bijdrage te leveren aan je favoriete open source project of toch (weer) dat hobbyproject af te stoffen.

Mijn tip: vergeet ook niet af en toe heel iets anders en zelfs soms helemaal niets te doen. Zelf ben ik bijvoorbeeld weer begonnen met pianospelen. Echt muzikaal is het niet, maar het is wel een prima afleiding.

Namens het bestuur, tot snel online of op een later moment bij een NLJUG event.

Zorg goed voor jezelf en voor elkaar.

Rob Brinkman

Rob Brinkman is een bestuurslid van de NLJUG.

ORDINA
Ahead of change

NU **SOFTWARE ONTWIKKELEN**

MORGEN DE KLANT **VOORSPRONG GEVEN**

Bij Ordina JTech werk je samen met ruim 150 gedreven en creatieve developers bij de gaafste Javaclub van Nederland.

Wil jij voorop lopen in je vak?
Bekijk de mogelijkheden op onze website:

WWW.ORDINA.NL/VAKGEBIEDEN/JAVA