JAVA MAGAZINE
.nl. jug
Onafhankelijk tijdschrift voor de Java-professional
04 > 2022

J-FALL IS HERE!

›DEVELOPEN OP DE IPAD
MET GITPOD

›FLUTTER
WAAROM EN HOE?

›CYBERSECURITY
IN DEVOPS

›JAVA 19
ALLE NIEUWE FEATURES


We think there is a place for everyone within the bank. Including employees with autism.
Discover IT & how Rabobank creates an inclusive culture at **rabobank.jobs**

Rabobank

{ COLOFON } JAVA MAGAZINE 04-2022

Content Manager & Project Coördinatie:
Stijn van de Blankevoort

Eindredactie:
Eveline Kroese

Auteurs:
Rogier van Apeldoorn, Raymond Jetten, Laurens van der Kooi, Bjorn van der Laan, Julien Lengrand-Lambert, Fernanda Machado, Maja Reißner, Maarten Smeets, Brian Vermeer, Arjen Wiersma, Ivo Woltring

Redactiecommissie:
Stijn van de Blankevoort, Julien Lengrand-Lambert, Maja Reißner, Ted Vinke, Mark van der Walle, 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 € 59,50 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, Nijverheidsweg 18, 2031 CP 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, Nijverheidsweg 18, 2031 CP Haarlem of een e-mail naar **members@nljug.org**.

VOORWOORD

Tijd voor J-Fall!

Nog een paar dagen en dan begint J-Fall 2022! Ditmaal waren we door de tickets heen binnen 24 uur. Geen ticket bemachtigd? Kijk snel of er nog plek is voor 1 van de preconference workshops of de Masters of Java (beide de dag ervoor). Want als je meedoet van een van deze toffe events, krijg je ook een ticket voor J-Fall!

Met meer dan 60 top sprekers, vele deep dive sessies, get-your-hands-dirty workshops, inspirerende keynotes en meest gezellige marktvloer mag je dit natuurlijk niet missen. Vind in dit magazine vanaf bladzijde 25 o.a. de timetable en plattegrond. Mis je de abstracts? Dat zou heel goed kunnen! Onder het mom van duurzaamheid, vind je dit jaar alle sessies alleen online op de website of in de app. Een hartelijk dankjewel namens het event team en het NLJUG bestuur voor alle partijen die hebben bijgedragen aan deze editie van J-Fall... Maar bovenal natuurlijk: jij! Zonder jou was er geen J-Fall.

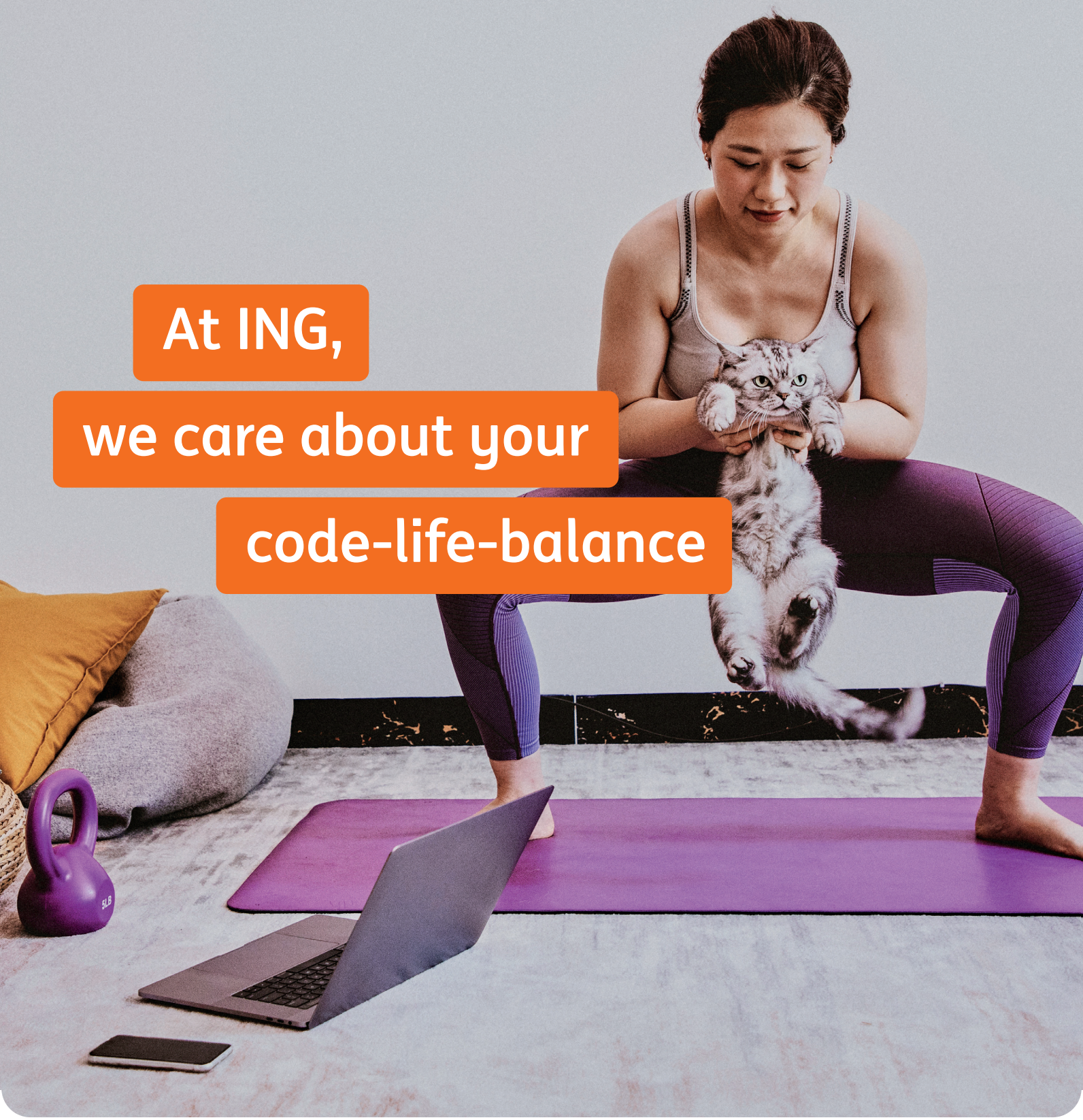
Maar wat hebben we nog meer voor je in petto? Ondertussen is Java 19 tot je beschikking. Benieuwd wat er allemaal nieuw is? Blader dan door naar pagina 6. Natuurlijk hebben we nog veel meer interessante artikelen. Duik bijvoorbeeld in de wereld van algoritmes of verdiep je in property-based testing. Zorg dat je je beveiliging in orde hebt aan de hand van het artikel van Raymon Jetten. Leer hoe je kan programmeren vanaf je iPad of ontdek Apache NiFi met Maarten Smeets en zie wat Flutter voor je kan betekenen.

Natuurlijk zijn we altijd op zoek naar meer toffe artikelen. Heb je ervaring met een leuk/interessant Java-gerelateerde tool en zou je deze kennis willen delen? Waag dan een poging en stuur een email naar info@nljug.org. Ook als het je eerste artikel is, is dat geen probleem, we begeleiden je graag!

Ditmaal nog een andere oproep: We hebben nog een stoel te vergeven in de redactiecommissie! Dus wil je bijdragen aan dit gave magazine en meebeslissen over de content en de koers die we bevaren? Kijk dan snel op pagina 49.

Veel leesplezier en tot ziens op J-Fall!

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



71% of our IT colleagues agree

Visit ing.nl/careers/it

do your thing

INHOUD

Op pad met Dijkstra

09 Je staat op het punt om naar J-Fall te gaan. Eerst wil je wel zeker zijn dat je niet in de file terechtkomt, dus je controleert een online routeplanner. Helaas staat er bij Amersfoort een file, maar gelukkig komt de routeplanner met een alternatieve route. Het soort algoritme dat de routeplanner gebruikt, heet een 'kortste pad algoritme' en het gebruikt een 'graaf' als datastructuur. De basis voor dit algoritme is in 1956 gelegd door Edsger Dijkstra en staat bekend als Dijkstra's Algoritme. In dit artikel geef ik een Java-implementatie die je kan gebruiken.

CCybersecurity in DevOps

20 Als DevOps-engineer ben jij verantwoordelijk voor het ontwikkelen en onderhouden van de IT-systemen van je opdrachtgever. Alsof die taken nog niet genoeg zijn, is er ineens een cybersecurity-incident op een van de systemen die onder jouw verantwoordelijkheid vallen. Nu blijkt dat je niet alleen verantwoordelijk bent voor ontwikkeling en onderhoud, je bent ook nog eens de eerste verdedigingslinie.

iPad driven development using Gitpod

40 Most of us Java developers use beefy machines to get our work done every day. They're powerful, can run dozens of docker images every day and keep us fresh by blowing their fans at full force when compiling. And still, there are many places where even the beefiest machine isn't the best option at hand. Maybe you're on a train with a spotty connection and need to update your dependencies. Or maybe you quickly want to fix a bug in that library you used today.

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 19

Jawel hoor, we zijn weer een half jaar verder en het is tijd voor een nieuwe versie van Java. In Java SE 19 staan zeven features (JEP's) gepland.

Om met wat Java SE 19 features te kunnen spelen (zonder de early access echt te hoeven installeren) is alle code in dit artikel uitgevoerd binnen een dockercontainer met OpenJDK 19 [2], zie Listing 1.

L1

```
$ docker run -it --rm \
      -v $(pwd)/src:/src \
      openjdk:19-slim /bin/bash
$ cd /src/java19
```

Dit artikel is in twee hoofdsecties verdeeld. Het eerste deel gaat over nieuwe standaardfeatures. Het tweede deel gaat in op preview en incubator features. Normaal gesproken is er nog een derde sectie, waarin we spreken over de features die uitgefaseerd (gaan) worden, maar er zijn er geen aangekondigd voor deze versie. Bij elke feature zal het JEP-nummer vermeld worden.

{ NIEUWE STANDAARD FEATURES }

In Java 19 is maar één nieuwe feature aangekondigd.

422: Linux/RISC-V Port

RISC-V (uitgesproken in het Engels als 'Risk-five') is een oorspronkelijk aan de Berkley Universiteit van California ontwikkelde RISC-instructiesetarchitectuur (ISA). De toenemende beschikbaarheid van RISC-V-hardware maakt een poort van de JDK waardevol. In Java 19 zal deze poort voltooid zijn en onderdeel worden van de JDK.

{ PREVIEW EN INCUBATOR FEATURES }

424: Foreign Function & Memory API (Preview)

Deze JEP vervangt twee eerdere incubatie-API's: de Foreign Memory Access API (JEP's 370, 383 en 393) en Foreign Linker API (JEP 389). De eerdere incubaties faalden. Het doel van deze JEP is om een gebruiksvriendelijkere en algemenere API te bieden om met code en gegevens buiten JVM te werken.

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

426: Vector API (Fourth Incubator)

Deze JEP is de vierde incubatiefase van een API om vectorberekeningen te compileren tot optimale vectorinstructies op de ondersteunde CPU-architecturen. Deze fase richt zich vooral op verbeteringen uit feedback en op verbeterde implementatie en

L2

```
package java19;
public class JEP405 {
    record Point(int x, int y) {}
    static void printSumOld(Object o) {
        if (o instanceof Point p) {
            int x = p.x();
            int y = p.y();
            System.out.println(x + y);
        }
    }
    static void printSumNew(Object o) {
        if (o instanceof Point(int x,int y)) {
            System.out.println(x + y);
        }
    }
    public static void main(String[] args) {
        printSumOld(new Point(22, 20));
        printSumNew(new Point(20, 22));
    }
}
$ java --enable-preview --source 19 JEP405.java
42
42
```

performance. Deze JEP bouwt voort op JEP 417 uit Java SE 18, JEP 414 uit Java 17 en JEP 338 die in Java SE 16 is geïntroduceerd. Zie referentie [2] voor voorbeeldcode.

405: Record Patterns (Preview)
Java 16 is middels JEP 394 uitgebreid met een 'typepatronen test' (Engels: 'Type Pattern'). In Java 17 en 18 is ook het switch-case-statement hiermee uitgebreid, via respectievelijk JEP 406 en 420, zie referentie [3] voor codevoorbeelden.

Het gebruik van Type Pattern zal in de meeste gevallen de noodzaak van typecasten verwijderen. Dit is echter pas de eerste stap naar een meer declaratieve, datagerichte programmeerstijl. Aangezien Java nu met records een meer expressieve manier ondersteunt om gegevens te modelleren, kan pattern matching het gebruik van gegevens makkelijker maken door ze in staat te stellen de semantische bedoeling in hun modellen uit te drukken, zie Listing 2.

427: Pattern Matching for switch (Third Preview)
Dit is de derde preview van pattern matching voor switch-statements die voor het eerst is uitgebracht in Java 17 in JEP 406 en zijn tweede preview kreeg in Java 18 in JEP 420. In deze derde preview zijn vooral kleine verbeteringen doorgevoerd aan de hand van gebruikersfeedback en gebruikservaring. Bekijk de voorbeeldcode [2] en het Java 17 artikel [3].

425: Virtual Threads (Preview)
Virtual Threads zijn onderdeel van project Loom [4]. Project Loom is gericht op het verbeteren van de concurrency-prestaties in Java door de ontwikkelaar concurrency-toepassingen met bekende API's te laten schrijven, te onderhouden en hardwarebronnen efficiënter te laten gebruiken.

Virtuele Threads zijn nieuwe, lichtgewicht implementaties van Java's thread-klasse die worden ingepland door de JDK, in plaats van door het besturingssysteem (OS), zoals tot nu toe het geval is geweest in Java. Voorbeeldcode is hier achterwege gelaten, omdat je er in het volgende Java Magazine een heel artikel over kan lezen.

428: Structured Concurrency (Incubator)
Het idee achter Structured Concurrency is om de levensduur van een of meerdere threads hetzelfde te laten werken als codeblokken in gestructureerd programmeren. Structured Concurrency behandelt meerdere taken in verschillende threads als een enkele unit of work (eenheid van werk), waardoor foutafhandeling en afbreken worden gestroomlijnd, wat de betrouwbaarheid en ook de observeerbaarheid (debugging) verbetert.

Listing 3 geeft drie voorbeelden van een `foo`-method. Een method waar de code sequentieel wordt aangeroepen. Een waar het aangeroepen wordt op een multithreaded manier en eentje waar Structured Concurrency wordt toegepast. In de `fooSequential`-methode is het voor de gemiddelde ontwikkelaar overduidelijk wat er gebeurt als er in een van de statements een exception plaatsvindt. `fooSequential` zal falen op dat statement.

Hoe `fooThreaded` faalt als bijvoorbeeld in `baz()` een exception wordt gegooid, is een stuk moeilijker te begrijpen. De threads moeten namelijk eerst helemaal resolven, voordat de `foo`-method in de fout zal propageren. Deze zal namelijk `bar()` eerst evalueren en die zal pas na twee seconden terugkomen. Dit komt, omdat de `baz` en `bar`-calls in isolatie draaien. Het gaat nog verder. Stel dat deze `foo`-method zelf in de fout gaat voordat de joining calls worden gedaan. Dan zal `foo` al falen, maar de threads gewoon doorlopen.

In de `fooStructured`-method worden de aangemaakte threads als één unit of work gezien en zal de `foo`-method meteen terugkomen als een van de andere calls falen.

In Listing 4, waar de code gedraaid is zonder dat een exception gegooid wordt, kan je zien dat de Sequential-call meer dan 2500 ms duurt, omdat `bar` en `baz` na elkaar aangeroepen worden.

L3

```
import jdk.incubator.concurrent.*;
import java.io.IOException;
import java.util.concurrent.*;
public class JEP428 {
    String bar() throws InterruptedException {
        Thread.sleep(2000);
        return "bar";
    }
   String baz() throws IOException,
                       InterruptedException {
        Thread.sleep(500);
        return "baz";
//        throw new IOException("baz");
    }
    String fooSequential() throws
    InterruptedException, IOException {
        String bar = bar(); // kan een Exception
        opleveren
        String baz = baz(); //ditto
        return baz + bar;
    }
    String fooThreaded() throws
    ExecutionException, InterruptedException {
        var executorService =
        new ForkJoinPool(2);
        Future<String> bar =
        executorService.submit(this::bar);
```

Vervolg op de volgende pagina

L3

```
        Future<String> baz = executorService.
        submit(this::baz);
        return bar.get() + baz.get();
    }
    String fooStructured() throws
    ExecutionException, InterruptedException {
        try (var scope = new
        StructuredTaskScope.ShutdownOnFailure()) {
            Future<String> bar =
            scope.fork(this::bar);
            Future<String> baz =
            scope.fork(this::baz);
            scope.join();
            scope.throwIfFailed();
            return bar.resultNow() +
            baz.resultNow();
        }
    }
    public static void main(String[] args) {
        var self = new JEP428();
        long start = System.currentTimeMillis();
        try {
            System.out.println(
self.fooSequential());
        } catch (InterruptedException |
IOException e) {
            System.out.println("Error:
            sequential");
        }
        long end = System.currentTimeMillis();
        System.out.println("Sequential took " +
        (end - start) + " ms");
        start = System.currentTimeMillis();
        try {
            System.out.println(self.
            fooThreaded());
        } catch (ExecutionException |
                InterruptedException e) {
            System.out.println("Error:
            threaded");
        }
        end = System.currentTimeMillis();
        System.out.println("Threaded took " +
        (end - start) + " ms");
        start = System.currentTimeMillis();
        try {
            System.out.println(self.
            fooStructured());
        } catch (ExecutionException |
        InterruptedException e) {
```

L3

```
            System.out.println("Error:
            structured");
        }
        end = System.currentTimeMillis();
        System.out.println("Structured took " +
        (end - start) + " ms");
    }
}
```

Tussen Threaded en Structured zit niet veel verschil. Echter het grote verschil wordt duidelijk, zodra er wel wat fout gaat (Listing 5). Dan kan je zien dat bij het gebruik van Structured Concurrency de `foo`-methode faalt, zodra de eerste exception gegooid wordt (in de `baz`-method).

{ CONCLUSIE }

Veel incubator en preview features in deze versie van Java, maar ook wel echt potentiële features met een hoop belofte. Als de Virtuele Threads en de Structured Concurrency het halen om tot de core van Java te behoren, dan belooft dat veel goeds. Multi threaded werken alsof je gewoon gestructureerd aan het programmeren bent. Ik ben vóór! <

L4

```
$ java –source 19 –enable-preview \
–add-modules jdk.incubator.concurrent JEP428.java
bazbar
Sequential took 2504 ms
barbaz
Threaded took 2008 ms
barbaz
Structured took 2062 ms
```

L5

```
$ java –source 19 –enable-preview \
–add-modules jdk.incubator.concurrent JEP428.java
Interrupted in sequential
Sequential took 2512 ms
Interrupted in threaded
Threaded took 2019 ms
Interrupted in structured
Structured took 531 ms
```

{ REFERENTIES }

1 https://openjdk.org/projects/jdk/19/
2 http://ivo2u.nl/Vy
3 http://ivo2u.nl/oz
4 https://openjdk.org/projects/loom/

OP PAD MET DIJKSTRA

Je staat op het punt om naar J-Fall te gaan. Eerst wil je wel zeker zijn dat je niet in de file terechtkomt, dus je controleert een online routeplanner. Helaas staat er bij Amersfoort een file, maar gelukkig komt de routeplanner met een alternatieve route, zodat je op tijd bent voor de officiële opening. Het soort algoritme dat de routeplanner gebruikt, heet een 'kortste pad algoritme' en het gebruikt een 'graaf' als datastructuur. De basis voor dit algoritme is in 1956 gelegd door Edsger Dijkstra en staat al jaren bekend als Dijkstra's Algoritme. In dit artikel neem ik je mee in dit algoritme en geef ik een Java-implementatie die je kan gebruiken.

Arjen Wiersma is Hoofddocent bij NOVI hogeschool in Utrecht. Hij verzorgt geavanceerde vakken, zoals Datastructuren & Algoritmen en Software Security. NOVI Hogeschool is kennispartner van de NLJUG.

{ WAT IS EEN GRAAF? }

Een graaf is een verzameling van knopen en lijnen. In het Engels heten ze 'Vertexes' en 'Edges'. Een knoop kan met andere knopen verbonden worden met lijnen. Een lijn kan een richting aangeven en kan ook een gewicht hebben. Met het voorbeeld van de routeplanner zou een gerichte lijn een straat met een enkele richting kunnen aangeven. Het gewicht kan de afstand van de ene knoop naar de andere knoop weergeven. Het gewicht is een arbitrair iets; je mag zelf kiezen wat het betekent. In plaats van afstand zou het ook de maximale snelheid kunnen zijn. In dit artikel wordt een gerichte graaf besproken waarbij de waarde van het gewicht de afstand tussen de knopen weergeeft.

Er wordt actief onderzoek gedaan naar grafen, voornamelijk hoe deze op verschillende probleemgebieden kunnen worden toegepast. Al deze onderzoeken vallen onder de noemer 'grafentheorie'. In de grafentheorie worden nog steeds nieuwe algoritmen ontdekt die op innovatieve manieren gebruik maken van de eigenschappen van een graaf. De algoritmen zijn over het algemeen elegant te noemen. Ze bestaan uit een verzameling van ogenschijnlijk eenvoudige stappen die zeer complexe problemen weten op te lossen.

{ TOEPASSINGEN VAN GRAFEN }

Wellicht denk je dat een graaf enkel een toepassing heeft in het navigeren van het ene punt naar het andere punt, maar niets is minder waar. Je vindt grafen echter overal; van het verwerken van taal tot het bestuderen van moleculen in scheikunde. Er zijn ook geavanceerde databases, zoals Neo4J die data opslaan in grafen.

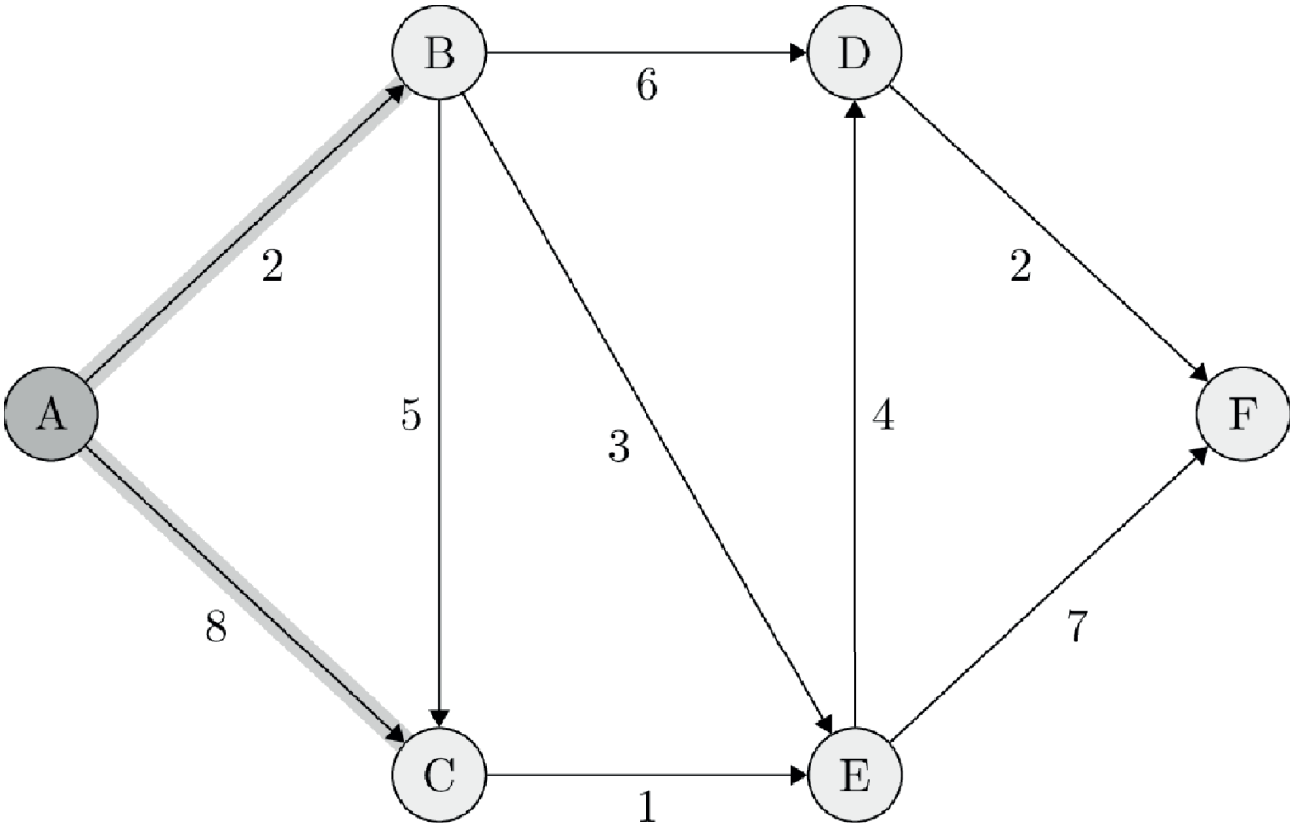
Als voorbeeld kun je de Java-code die je dagelijks schrijft nemen. De compiler leest de code en bouwt een boomstructuur, een graaf. Op basis van het bouwen en analyseren van deze graaf wordt er feedback gegeven aan de programmeur en wordt duidelijk of het programma correct geschreven is. Ook kunnen er, door toepassing van algoritmes, verbeteringen worden voorgesteld.

{ HET ALGORITME }

Dijkstra's algoritme neemt een verzameling van knopen en lijnen, waarbij de lijn gericht is en voorzien van een gewicht wat de afstand van de ene knoop naar de andere knoop weergeeft. Vanaf een bronknoop wordt de graaf systematisch geëvalueerd. Bij toepassing van het algoritme ontstaat er een tabel waarbij van elke knoop bekend is wat de kortste route tot de knoop is en vanuit welke knoop deze te bereiken is. Het algoritme maakt het mogelijk om van elke knoop te weten wat het kortste pad is vanaf de bronknoop.

Om het algoritme toe te passen is het nodig om eerst alle knopen als 'onbezocht' aan te merken. Dit kan als een eigenschap van een object of een element in een lijst. Daarnaast hebben alle knopen een oneindige afstandswaarde. Het is bij de start nog niet bekend wat de daadwerkelijke afstand is tot de knoop. Kies de bronknoop als de huidige knoop; vanuit deze knoop worden alle routes bepaald. Vervolgens worden er drie stappen systematisch uitgevoerd:

1. Bekijk alle verbonden, onbezochte knopen en bereken de voorlopige afstandswaarde van de route door de huidige knoop. Als de afstandswaarde lager is dan de al opgeslagen waarde, sla deze waarde en de huidige knoop op.
2. Als alle verbonden, onbezochte knopen berekend zijn, markeer dan de huidige knoop als 'bezocht'.
3. Selecteer de knoop met de laagste afstandswaarde die nog niet bezocht is en herhaal vanaf stap 1. Als er geen onbezochte knopen meer zijn, stopt het proces.



Afbeelding 1: Een verzameling knopen en lijnen met gewicht.

Als er enkel nog knopen zijn met een oneindige afstandswaarde, dan zal er geen pad zijn vanuit de bronknoop en is deze onbereikbaar. Wanneer het algoritme stopt, is van elke bereikbare knoop bekend wat de meest minimale afstandswaarde tot deze knoop is vanuit de bronknoop. Het algoritme is 'hebberig' te noemen. Het stopt alleen als alle opties verwerkt zijn. Als enkel het pad naar een specifieke knoop bepaald moet worden, is het mogelijk om te stoppen op het moment dat de route tot die specifieke knoop bezocht is. De eigenschap van het algoritme dat altijd eerst het kortste pad bezocht wordt, zorgt ervoor dat er geen korter pad zal zijn.

{ EEN VOORBEELD }

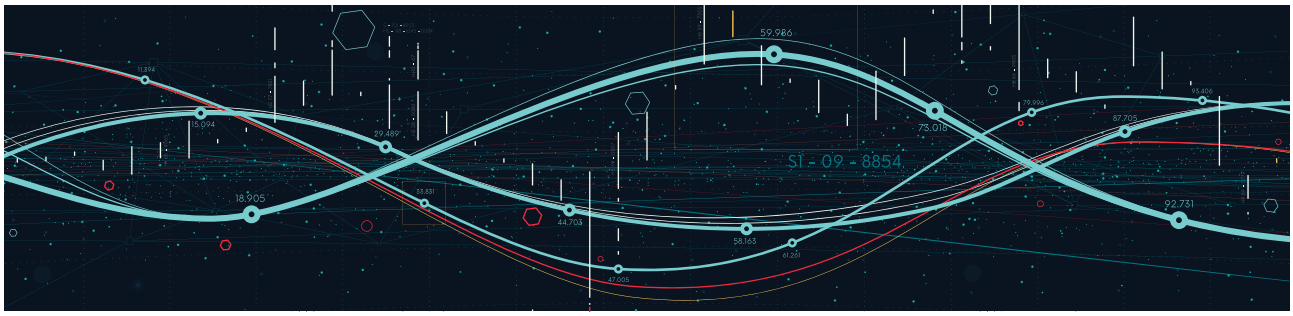
Hoewel het algoritme drie eenvoudige stappen kent, is het niet voor iedereen even gemakkelijk te begrijpen. Als voorbeeld is in Afbeelding 1 een graaf afgebeeld met zes knopen; A tot en met F.

Dit is een kleine graaf en puur op intuïtie wordt duidelijk dat als het kortste pad van `A` naar `F` gezocht wordt, deze via `B` en `D` gaat. Door Dijkstra's Algoritme toe te passen kan dit ook bewezen worden.

Als bronknoop wordt knoop `A` genomen. Vervolgens worden alle verbonden, onbezochte knopen bekeken. In dit geval zijn dat de knopen `B` en `C` (stap 1). Vanuit knoop `A` is de afstand tot knoop `B` 2. Omdat 2 lager is dan oneindig wordt de tabel bijgewerkt voor knoop `B` met de afstandswaarde 2 en de 'vanuit'-knoop naar `A`. Hetzelfde geldt voor knoop `C`; deze krijgt de waarde 8 en 'vanuit' wordt bijgewerkt naar `A`. Omdat nu alle verbonden, onbezochte knopen bekeken zijn, wordt `A` gemarkeerd als bezocht (stap 2).

Om de volgende knoop te bepalen, wordt de knoop met de laagste afstandswaarde uit de tabel genomen. Enkel `B` en `C` hebben een niet oneindige waarde en van de twee knopen heeft `B` de laagste waarde; 2. Stap 1 wordt opnieuw uitgevoerd. De knopen `C`, `D` en `E` zijn verbonden en zijn nog niet bezocht. De afstand naar knoop `C` vanuit `B` is 5. De afstandswaarde van knoop `B` was 2. De afstand naar `C` wordt 2 + 5 = 7. Omdat 7 lager is dan de huidige waarde van 8 wordt de tabel weer bijgewerkt. De afstandswaarde van `C` wordt 7 en de 'vanuit'-waarde wordt `B`. Het kortste pad vanuit `A` naar `C` is op dit moment 7 via knoop `B`. Dezelfde toepassing voor `D` geeft een afstandswaarde van 8 vanuit `B` en 5 voor `E` vanuit `B`. Uiteindelijk wordt `B` gemarkeerd als bezocht.

De volgende onbezochte knoop om te bekijken wordt `E`, omdat deze een afstandswaarde van 5 heeft. De procedure wordt herhaald totdat er geen verbonden, onbezochte knopen over zijn. In Tabel 1 staat het resultaat van de gehele toepassing van het algoritme. Vanuit deze tabel is het nu mogelijk om vanuit elke knoop in



de graaf te bepalen wat de kortste route is naar de bronknoop `A`. In de tabel is te lezen dat de kortste route naar `F` de afstandswaarde 10 heeft. Om bij `F` te komen met deze waarde kan dat enkel via knoop `D`. Van `D` is te lezen dat het kortste pad een afstandswaarde van 8 heeft en via `B` komt. `B` heeft vervolgens een afstandswaarde 2 en komt via `A`, wat ook de bronknoop is. De kortste route is dan ook `A`, `B`, `D` naar `F`.

Knoop	Afstand	Vanuit
A	0	
B	2	A
C	7	B
D	8	B
E	5	B
F	10	D

Tabel 1: Resultaat van het algoritme.

{ JAVA-IMPLEMENTATIE }

Een implementatie in Java vraagt een aantal objecten; eentje voor de knoop (Vertex) en eentje voor de lijn (Edge). Het is niet nodig om een tabel bij te houden, het is namelijk mogelijk om data op te slaan in objecten. De Edge is een simpele POJO met een start-Vertex en een doel-Vertex. De Vertex heeft een lijst met Edge-objecten die verbonden zijn met deze Vertex (`adjecenciesList`). Om te beginnen wordt de afstandswaarde in de Vertex op `Integer.MAX_VALUE` gezet als oneindigheidswaarde. Als laatste houdt de Vertex bij of deze al bezocht is.

Een datastructuur die goed past bij het algoritme is een PriorityQueue. Hiermee is altijd de Vertex met de kortste afstand het eerste element in de rij. Door de bronknoop als enige element in de rij te plaatsen met een afstand van 0, kan een eenvoudige `while`-lus gemaakt worden die door gaat tot de rij leeg is. Elke iteratie wordt het eerste element uit de queue opgehaald, worden de verbonden, onbezochte knopen bekeken en wordt deze bijgewerkt als de nieuwe afstand kleiner is dan de al gevonden afstand. De volledige en gedocumenteerde broncode vind je op Gitlab (https://gitlab.com/credmp/algorithms).

Het kortste pad in Java:

```
public void calculate(Vertex start) {
    start.setDistance(0);

    Queue<Vertex> queue = new PriorityQueue<>();
    queue.add(start);

    while (!queue.isEmpty()) {
        Vertex current = queue.poll();

        for (Edge edge : current.
        getAdjacenciesList()) {
            Vertex v = edge.targetVertex();
            if (!v.isVisited()) {
                int newDistance = current.
                getDistance() + edge.weight();

                if (newDistance < v.getDistance()) {
                    queue.remove(v);
                    v.setDistance(newDistance);
                    v.setPredecessor(current);
                    queue.add(v);
                }
            }
        }
        current.setVisited(true);
    }
}
```

{ AFSLUITEND }

Dijkstra's Algoritme is een elegante manier om in een graaf de kortste route vanuit een bronknoop naar elke andere knoop te vinden. Het algoritme vormt de basis van veel applicaties die dagelijks gebruikt worden en nu kan ook jij het algoritme toepassen. <

PROPERTY-BASED TESTING

Met Kotlin en Kotest

Bjorn van der Laan is backend software engineer bij Xebia. Daarnaast geeft hij training in Kotlin en is Bjorn betrokken bij de organisatie van Kotlin Dev Day.

Het schrijven van unittests gebeurt vaak op basis van voorbeelden. We spreken dan van 'example-based testing'. Bij het testen van een functie met een getal als inputparameter gebruiken we dan naast een aantal typische voorbeelden bijvoorbeeld een negatief getal, het getal nul en een aantal grote getallen. Vervolgens voeren we de testsuite uit en een lijst van groene vinkjes vertelt ons dat de functie naar behoren werkt. In dit artikel laat ik je zien waarom traditionele example-based tests tekort kunnen schieten en hoe property-based testing dit kan aanvullen (maar niet vervangen!). De voorbeeldcode in dit artikel is geschreven in Kotlin met het Kotest [1] testing framework.

{ EXAMPLE-BASED TESTEN }

Example-based tests specificeren het verwachte gedrag van een stuk code door voorbeeldscenario's te definiëren. Elke test is een enkel concreet scenario en stelt bij uitvoering vast of het resultaat overeenkomt met de verwachtingen. Neem bijvoorbeeld de `PaymentSettlementService` en `Euro`-classes in Listing 1. Een groep van bijvoorbeeld vrienden of collega's kan met deze service betalingen verrekenen die zij voor elkaar doen, bijvoorbeeld wanneer zij samen uitgaan.

De `settlePayment`-methode van `PaymentSettlementService` verhoogt de balans van de persoon aangegeven door `String paidBy` en verdeelt het betaalde bedrag over de personen in lijst `paidFor`. Als iemand nog niet bekend is, wordt er eerst een neutrale balans aangemaakt. Alle balansen worden opgeslagen in een map.

We willen deze methode testen om zeker te weten dat het naar behoren werkt. Het gaat hier om geld, dus dat moet niet mis gaan natuurlijk. Een testsuite aan unittests kan eruit zien als in Listing 2, waarbij de volledige code gevonden kan worden in mijn GitHub repo [2].

In dit geval hebben we voorbeelden waarbij een persoon alleen voor anderen betaalt of ook voor zichzelf. Daarnaast hebben we een grote betaling, een negatieve betaling en meerdere betalingen. In alle gevallen controleren we of de balansen van de deelnemers samen optellen tot 0. Als dit niet het geval is zou er geld gecreëerd of juist verdwenen zijn.

Wanneer alle concrete voorbeelden van een example-based testsuite slagen, wekt dat de indruk dat de methode het gewenste gedrag vertoont voor ook andere mogelijke inputwaarden. Dat is echter helemaal niet gegarandeerd. Daar komt bovenop dat een programmeur voor een testsuite vaak een aantal standaard voorbeeldobjecten definieert.

In het voorbeeld gebruiken we bijvoorbeeld altijd de namen 'Willem', 'Henk' en 'Jan'. De relevant geachte eigenschappen worden weliswaar per test gevarieerd, maar de testsuite is, op een subtiele en onbedoelde manier deels afhankelijk van deze door de programmeur gekozen standaard voorbeelden. Daarmee worden de tests ook beïnvloed door impliciete en misschien zelfs onbewuste aannames van deze programmeur.

In het voorbeeld hierboven zit bijvoorbeeld de aanname verstopt dat de namen van de personen niet uitmaken. Zelfs als dat nu nog niet zo is, kan dat in de toekomst altijd nog veranderen. Verder schrijven we vaak een veelvoud aan tests om er voldoende zeker van te zijn dat inputwaarden die binnen bereik, buiten bereik, geldig of ongeldig zijn voldoende zijn afgedekt. Dit is allemaal code die we moeten onderhouden. Property-based tests kunnen aan al deze bovengenoemde tekortkomingen tegemoetkomen.

{ PROPERTY-BASED TESTING }

Property-based tests (PBT) zijn ontworpen om kenmerken (properties) van een stuk code te testen die altijd waar zouden moeten zijn. Waar bij example-based testing de programmeur verantwoordelijk is voor het verzinnen van verschillende voorbeelden, maakt property-based testing gebruik van zogenoemde 'generators' om automatisch inputwaarden te genereren, en past deze toe op door de programmeur bedachte regels. Hierdoor hoef je als programmeur niet meer na te denken of je voorbeelden wel representatief zijn voor alle waarden die als input zouden kunnen worden gegeven. Property-based tests werken in dat opzicht op een hoger abstractieniveau dan example-based tests.

Ons testing framework Kotest kent twee soorten generators. ‘Exhaustives’ genereren alle mogelijke waarden in een bepaald bereik, zoals alle positieve integers of alle waarden van een enum. Vaker gebruiken we echter het tweede type generators genaamd ‘Arbs’ (kort voor Arbitraries), die een te configureren aantal (standaard duizend) waarden genereren en daarbij zowel pseudo-willekeurige waarden als voorgedefinieerde edgecases meenemen. De positive integer Arb genereert bijvoorbeeld duizend integers groter dan nul, waarvan standaard in ieder geval twee procent een edgecase zal zijn. Edgecases voor positieve integers zijn onder andere `1` en `Int.MAX_VALUE`.

Naast de ingebouwde Arbs kunnen we ook onze eigen Arb bouwen. Vaak bouw je zo’n custom Arb voor een eigen geschreven class door het combineren van ingebouwde Arbs. Je vindt een voorbeeld van een custom Arb voor onze `Euro`-class in mijn GitHub repo [2]. Listing 3 laat zien hoe we onze eerdere example-based tests kunnen vervangen door een property-based test. Als onze property kiezen we hierbij dat de som van alle balansen altijd optelt tot nul.

We zouden verwachten dat deze nieuwe test wel zal slagen, omdat de example-based tests eerder ook allemaal geen probleem opleverden. Maar schijn bedriegt, want bij het uitvoeren zien we dat deze property-based test faalt. Om ons te helpen het probleem te lokaliseren heeft property-based testing een functionaliteit genaamd ‘shrinking’ (zie Listing 4). Het testing framework neemt daarbij de meestal complexe willekeurige input van een gefaalde run en maakt dit steeds kleiner om zo het minimale voorbeeld te vinden waarbij de test faalt. De definitie van kleiner is hierbij per Arb gedefineerd.

L1

```
class PaymentSettlementService {
    val balances: MutableMap<String, Euro> =
mutableMapOf()

    fun settlePayment(paidBy: String, paidFor:
        List<String>, totalAmount: Euro) {
        val totalMembers = paidFor.size
        val amountPerMember = totalAmount /
        totalMembers

        balances[paidBy] = balances.
        getOrDefault(paidBy, Euro(0)) + totalAmount

        paidFor.forEach { member ->
            balances[member] = balances.
            getOrDefault(member, Euro(0)) -
            amountPerMember
        }
    }
}
```

In dit geval komen we erachter dat balansen niet altijd optellen tot nul wanneer het te verrekenen bedrag niet op gehele centen uitkomt. Deze imprecisie wordt veroorzaakt door het gebruik van integer division in de `div`-methode van `Euro`. Onze property geldt dus niet in alle gevallen en dit was ons bij de example-based tests niet opgevallen.

L2

```
class EuroExampleBasedTest : FeatureSpec({
    infix fun Euro.shouldBe(other: Euro) =
        this.toCents() shouldBeExactly other.
toCents()

    feature(“Sum of all balances should always
be zero”) {
        scenario(“Willem pays for all including
himself”) {
            val payees = listOf(“Willem”,
“Henk”, “Jan”)
            val payer = “Willem”

            val settleService =
PaymentSettlementService()
            settleService.settlePayment(payer,
payees, Euro.fromCents(90))

            val sumOfBalances = settleService.
balances.values
                .fold(Euro(0)) { sum, balance ->
sum + balance }

            sumOfBalances shouldBe Euro(0)
        }

        scenario(“Willem pays for all excluding
himself”) {...}

        scenario(“Willem pays a negative amount
for all”) {...}

        scenario(“Willem pays a big amount for
all”) {...}

        scenario(“Willem and Henk both pay
something for the group”) {...}
    }
})
```

Om onze code te verbeteren zouden we, afhankelijk van de requirements, nu bijvoorbeeld kunnen kiezen om de methode aan te passen of de hele `Euro`-class omschrijven van integers naar doubles. Als het gaat om het verrekenen van betalingen tussen vrienden zouden we in dit soort gevallen bijvoorbeeld een willekeurige balans een cent minder kunnen verlagen, omdat een cent niet zoveel uitmaakt.

{ CONCLUSIE }

In dit artikel hebben we gezien wat property-based testing is en hoe het verschilt van het meer gangbare example-based testing. Waar het definiëren van representatieve voorbeelden bij example-based testing de verantwoordelijkheid van de programmeur is, laten we dit bij property-based testing over aan generators, waardoor de tests niet deels afhankelijk worden van de gekozen voorbeelden.

Property-based testing stelt ons in staat om concrete voorbeelden te generaliseren, zodat we ons kunnen focussen op de kenmerken van de code die aan deze voorbeelden ten grondslag liggen. Dit resulteert in een schonere en compactere testsuite die makkelijk te onderhouden is en subtiele bugs beter blootlegt. Zo brengen we datgene wat we daadwerkelijk testen dichterbij wat we claimen te testen.

Het is echter ook belangrijk om op te merken dat property-based testing niet bedoeld is om unittests op basis van voorbeelden compleet te vervangen. Het formuleren van de juiste property's is lastiger dan het opstellen van een aantal goede voorbeeldscenario's en het uitvoeren van property-based tests kost in het algemeen meer tijd. Dit maakt property-based tests minder geschikt voor test-driven development.

Wanneer de implementatie stabieler is, kan het waardevol zijn om een example-based testsuite te vervangen door property-based tests. Het is een toevoeging aan ons testarsenaal, waarmee we met een enkele test elke denkbare inputwaarde kunnen afdekken. Dit kan bugs aan het licht brengen waar we misschien niet direct aan hadden gedacht. ‹

L3

```
class EuroPropertyBasedTest : FeatureSpec({
    infix fun Euro.shouldBe(other: Euro) =
        this.toCents() shouldBeExactly other.
        toCents()

    feature("Sum of all balances should always
    be zero") {
        checkAll(
            Arb.positiveInt(23039), Arb.list
           (Arb.firstName(), 2..4)
        ) { totalAmount, firstNames ->
            val payees = firstNames.map { it.name }
            val payer = payees.random()

            val settleService =
            PaymentSettlementService()
            settleService.settlePayment(payer,
            payees, Euro.fromCents(totalAmount))

            val sumOfBalances = settleService.
            balances.values
                .fold(Euro(0)) { sum, balance ->
                sum + balance }

            sumOfBalances shouldBe Euro(0)
        }
    }
})
```

{ REFERENTIES }

1 Kotest: https://kotest.io/
2 Voorbeeldcode: https://github.com/BjornvdLaan/Property-BasedTesting-Kotest-Kotlin-Demo

L4

```
Property test failed for inputs

0) 9578
1) [FirstName(name=Lorri),
FirstName(name=Amber), FirstName(name=Franklin),
FirstName(name=Madonna)]

Attempting to shrink arg [FirstName(name=Lorri),
FirstName(name=Amber), FirstName(name=Franklin),
FirstName(name=Madonna)]
Shrink #1: [FirstName(name=Lorri),
FirstName(name=Amber), FirstName(name=Franklin)]
pass
Shrink #2: [FirstName(name=Lorri),
FirstName(name=Amber)] fail
Shrink result (after 2 shrinks) =>
[FirstName(name=Lorri), FirstName(name=Amber)]

Caused by java.lang.AssertionError: 1 should be
equal to 0 at
[...]
```

POLYMORFISME TOEGEPAST OP EEN REST API MET SPRING BOOT

In een object-georiënteerde taal als Java is het gebruikelijk om overerving toe te passen. Het feit dat een superklasse de vorm kan hebben van een subklasse (en daarmee verschillende vormen kan aannemen), wordt 'polymorfisme' genoemd. Een van de krachtige middelen die polymorfisme in Java biedt, is het gebruik van polymorfe parameters in methoden. Dit houdt in dat een methode bijvoorbeeld een superklasse als parameter kan krijgen en dat deze methode vervolgens met een subklasse aangeroepen kan worden. Dit artikel laat zien hoe je dit krachtige middel op een REST API in een Spring Boot-applicatie kunt toepassen.

Laurens van der Kooi is Java Developer bij Ordina JTech. Hij vindt het gaaf om kennis te delen en is groot fan van het Spring Framework.

{ DATA TRANSFER OBJECTS }

Bij het implementeren van een REST API wordt er over het algemeen gebruik gemaakt van Data Transfer Objects (DTO's). Dit zijn simpele objecten (Listing 1) die alleen de data bevatten die nodig is in de informatie-uitwisseling bij het aanroepen van een endpoint.

Vaak zijn deze objecten een representatie van een veel complexer domeinobject in de backend van een applicatie. Listing 2 toont een versimpelde controller die de `VehicleDto` uit Listing 1 kan ontvangen en alle opgeslagen Vehicles kan retourneren. Om de DTO naar het bijbehorende domeinobject te mappen en weer terug, wordt er gebruik gemaakt van een mapper (deze is voor dit artikel verder niet relevant). Hiermee worden er over en weer alleen DTO's gecommuniceerd.

L1

```
public class VehicleDto {
    private Long id;
    private VehicleType vehicleType;
    private String brand;
    private String model;

    // ... getters and setters
}
```

{ POLYMORFE DTO'S }

Wanneer een domeinobject een datamodel kent met meerdere subklassen, kan het wenselijk zijn om je DTO een dito structuur te geven. In ons voorbeeld wordt `VehicleDto` een abstracte basisklasse met een aantal implementaties voor verschillende voertuigtypes (Listing 3). Ook wordt er een abstracte tussenlaag geïntroduceerd: `MotorizedVehicleDto`. Deze bevat een aantal velden die voor gemotoriseerde voertuigen van toepassing zijn.

Wanneer deze DTO's door dezelfde endpoints als in Listing 2 verwerkt moeten kunnen worden, dan ontstaat een use case voor het toepassen van polymorfisme op een REST API.

{ SERIALISATIE EN DESERIALISATIE }

Het onderwerp dat om de hoek komt kijken bij het toepassen van polymorfisme op REST is: 'serialisatie'. Serialisatie is in deze context het omzetten van een Java-object in een vorm die geschikt is voor verzending. 'Deserialisatie' is het tegenovergestelde: het maken van een Java-object vanuit een geserialiseerde vorm. In de voorbeelden in dit artikel communiceren we via JSON met de REST API. Met serialisatie wordt in de komende voorbeelden dus het omzetten van een DTO naar JSON bedoeld; met deserialisatie het omzetten van JSON naar een DTO.

L2

```
@RestController
@RequestMapping(“/vehicles”)
public class VehicleController {
    private final VehicleService service;
    private final VehicleMapper mapper;

    // ... constructor

    @PostMapping
    public VehicleDto postVehicle(@RequestBody
    VehicleDto dto) {
        var vehicle = mapper.
        mapToVehicleEntity(dto);
        service.saveVehicle(vehicle);
        return mapper.mapToVehicleDto(vehicle);
    }

    @GetMapping
    public List<VehicleDto> getAllVehicles() {
        return service.getVehicles()
                .stream()
                .map(mapper::mapToVehicleDto)
                .collect(Collectors.toList());
    }
}
```

L4

```
{
   “vehicleType”: “CAR”,
   “brand”: “Toyota”,
   “model”: “C-HR”,
   “licensePlate”: “07-88-RE”,
   “horsePower”: 180,
   “fuelType”: “HYBRID”,
   “numberOfDoors”: 4,
   “bodyStyle”: “SUV”
}
```

{ JACKSON }

In een Spring Boot-applicatie is Jackson de default JSON-mapper. Jackson verzorgt in zo’n applicatie de serialisatie en deserialisatie wanneer er via REST met de applicatie wordt gecommuniceerd. Over het algemeen lukt het de Jackson-library wel om een DTO te interpreteren wanneer deze verstuurd wordt via een REST API. Voor de situaties waarin er afwijkend gedrag nodig is in dit proces kent Jackson verschillende annotaties om dit te configureren. Onze use case met de polymorfe DTO’s uit Listing 3 is daar een voorbeeld van.

L3

```
public abstract class VehicleDto {
    private Long id;
    private VehicleType vehicleType;
    private String brand;
    private String model;

    // ... getters and setters
}

public abstract class MotorizedVehicleDto
extends VehicleDto {
    private String licensePlate;
    private int horsePower;
    private FuelType fuelType;

    // ... getters and setters
}

public class CarDto extends MotorizedVehicleDto
{
    private int numberOfDoors;
    private BodyStyle bodyStyle;

    // ... getters and setters
}

public class BusDto extends MotorizedVehicleDto
{
    private BusType busType;
    private int litersLuggageCapacity;

    // ... getters and setters
}
```

{ POLYMORFISME EN JACKSON }

Als je in Java een methode definieert die een `VehicleDto` als parameter heeft, kun je deze methode aanroepen met een `CarDto` (een `CarDto` is immers een subklasse van `VehicleDto`, zie Listing 3). Zou het dan niet logisch zijn dat bij het versturen van een `CarDto` als JSON (Listing 4), deze JSON wordt gedeserialiseerd in een `CarDto` waarmee onze controller uit de voeten kan? Niet vanzelf! Als we geen configuratie zouden toevoegen aan onze use case, dan interpreteert Jackson de subtypes van de `VehicleDto` niet. Jackson kent in dat geval alleen de `VehicleDto`, omdat de REST-controller uit Listing 2 alleen methodes kent met de `VehicleDto` als parameter en return type. Een POST naar het /vehicles-endpoint met een `CarDto` resulteert in de deserialisatie van een `VehicleDto`, waarbij dus de `CarDto`-specifieke velden verloren gaan. Bovendien komt er in dit voorbeeld vanuit Jackson een foutmelding zoals in Listing 5, omdat een abstracte klasse niet geïnstantieerd kan worden.

L5

```
com.fasterxml.jackson.databind.exc.
InvalidDefinitionException: Cannot construct
instance of `nl.kooi.vehicle.api.dto.VehicleDto`
(no Creators, like default constructor, exist):
abstract types either need to be mapped to
concrete types, have custom deserializer, or
contain additional type information
```

L6

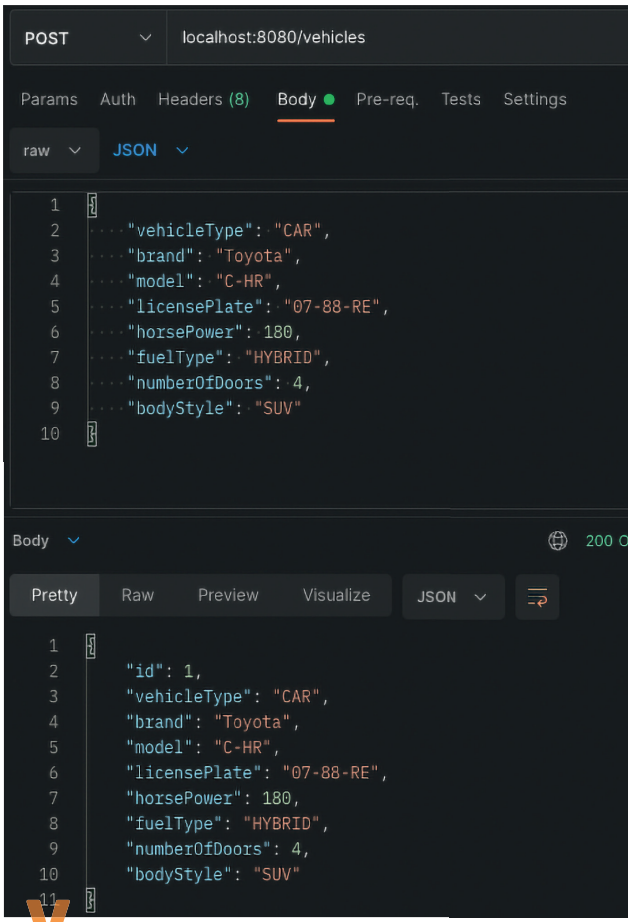
```
@JsonTypeInfo(
        include = JsonTypeInfo.As.EXISTING_
PROPERTY,
        property = "vehicleType",
        use = JsonTypeInfo.Id.NAME,
        visible = true
)
@JsonSubTypes({
        @JsonSubTypes.Type(value = BusDto.class,
        name = "BUS"),
        @JsonSubTypes.Type(value = CarDto.class,
        name = "CAR")
})
public abstract class VehicleDto {
    private Long id;
    private VehicleType vehicleType;
    private String brand;
    private String model;

    // ... getters and setters
}
```

`VehicleDto` is immers een abstracte klasse, waarin de 'basis'-velden van een voertuig gedefinieerd staan en die concrete implementatie nodig heeft om gebruikt te kunnen worden. Dit zou dan betekenen dat er voor iedere concrete subklasse van de `VehicleDto` een nieuw endpoint gedefinieerd moet worden. Voor de use case waarin het nodig is om alle (toekomstige) implementaties van `VehicleDto` via dezelfde endpoints te kunnen verwerken, is dit niet wenselijk.

De foutmelding in Listing 5 geeft al een aantal oplossingen:
- Werken met concrete klassen;
- het werken met een custom deserializer;
- het toevoegen van aanvullende type informatie.

Het laatste punt gaan we uitwerken, omdat dit een simpele en leesbare vorm is van het configureren van polymorfisme in Jackson.



Afbeelding 1.

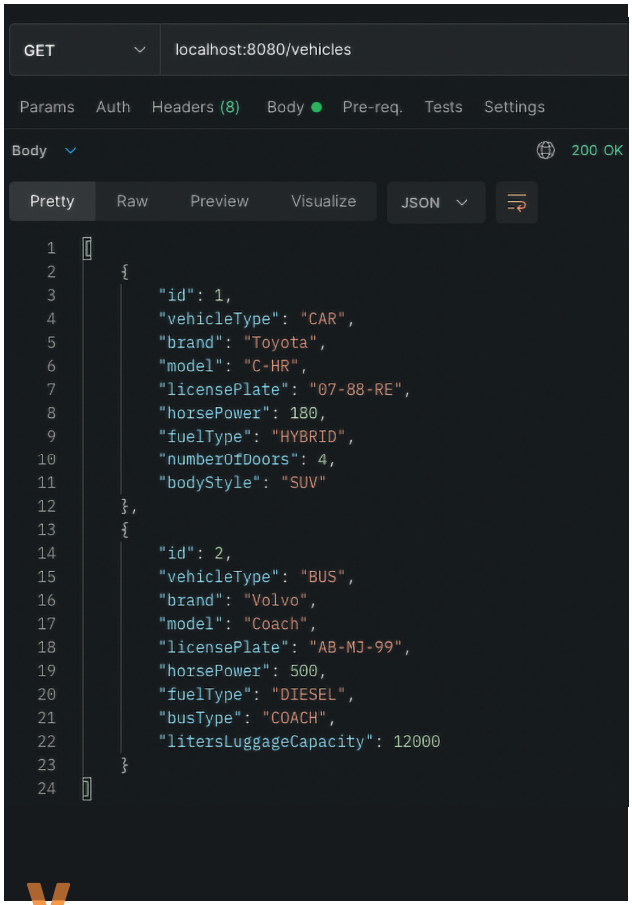
{ TOEVOEGEN VAN TYPE INFORMATIE }

Bij het voorzien van Jackson van type informatie, komen een tweetal Jackson-annotaties kijken. Deze annotaties zijn `@JsonTypeInfo` en `@JsonSubTypes`. De eerste beschrijft hoe de type informatie wordt bepaald, en de tweede beschrijft welke subklassen er worden onderkend. Logischerwijs worden deze annotaties op de superklasse geplaatst (in ons voorbeeld is dit de `VehicleDto`).

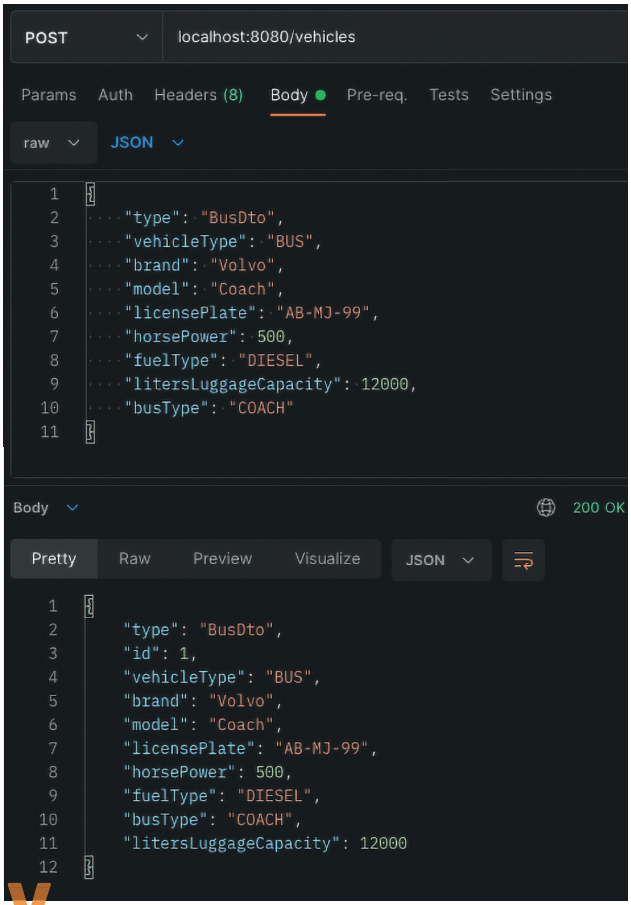
Omdat de `VehicleDto` een enum-veld bevat met `vehicleType` (deze bevat twee constanten: `BUS`, `CAR`), is dat een goede kandidaat om het onderscheid te kunnen maken tussen de verschillende subklassen. De implementatie hiervan staat in Listing 6.

Met `@JsonTypeInfo`-annotatie is aangegeven dat er op basis van een bestaand veld, met de naam 'vehicleType' het onderscheid gemaakt kan worden. Met `use = JsonTypeInfo.Id.NAME` is geconfigureerd dat het onderscheid gemaakt wordt op basis van de waarde die 'vehicleType' heeft. Ook wordt met `visible = true` aangegeven dat het betreffende veld geserialiseerd moet worden.

De `@JsonSubTypes`-annotatie geeft aan welke subtyperingen er worden onderkend (`BusDto` en `CarDto`), en welke waarde



Afbeelding 2.



Afbeelding 3.

het veld uit de `@JsonTypeInfo` omschrijving (vehicleType) in de betreffende situatie heeft (de zogenaamde discriminator).

{ HET CONSUMEREN VAN POLYMORFE DTO'S }

Wanneer de JSON uit Listing 4 nu via REST wordt aangeboden, wordt deze geaccepteerd en als `CarDto` verder de applicatie in gestuurd (zie bovenste JSON uit Afbeelding 1). Het endpoint retourneert nu ook de geserialiseerde versie van de `CarDto` (zie onderste JSON uit Afbeelding 1).

Wanneer er meerdere voertuigtypen in de applicatie zijn opgeslagen en worden opgehaald via het GET-endpoint dat alle voertuigen ophaalt, dan wordt ieder voertuig met de bijbehorende velden geserialiseerd (Afbeelding 2).

Wanneer er een nieuwe implementatie van de `VehicleDto` wordt geïntroduceerd (zoals Listing 7), is het nu slechts een kwestie van de `VehicleType`-enum uitbreiden en de `@JsonSubTypes`-annotatie aanvullen met de nieuwe informatie (Listing 8).

Zoals je ziet, is de controller uit Listing 2 verder ongemoeid gebleven bij het toevoegen van de nodige configuratie om Jackson om te kunnen laten gaan met subklassen. De controller heeft namelijk geen weet van het serialisatieproces dat zich afspeelt vóór en nádat een van de controllermethoden wordt aangeroepen.

{ ANDERE MANIEREN VAN TYPEBEPALING MET @JSONTYPEINFO }

In het voorgaande voorbeeld werd het type van de DTO bepaald aan de hand van een veld dat onderdeel is van de `VehicleDto`: vehicleType. Dit werd aangeduid door in de `@JsonTypeInfo`-annotatie `include = JsonTypeInfo.As.EXISTING_PROPERTY` op te nemen. De `@JsonTypeInfo`-annotatie kan ook op andere manieren geconfigureerd worden.

Wanneer de include property achterwege wordt gelaten, zal Jackson de standaardmanier van typebepaling gebruiken: `JsonTypeInfo.As.PROPERTY`. Het verschil met `EXISTING_PROPERTY` is dat er nu gebruik gemaakt wordt van een veld dat alleen voor serialisatie-doeleinden wordt toegevoegd. Dit veld wordt als meta-informatie gezien en is geen onderdeel van de DTO. De veldnaam moet nog wel gedefinieerd worden in het property-veld van de annotatie. In Listing 9 is te zien hoe deze

L7

```
public class BoatDto extends VehicleDto {
    private PropulsionType propulsionType;
    private int lengthMeters;
    private int draftMeters;

    // ... getters and setters
}
```

configuratie eruit ziet. In dit voorbeeld is de include property ter illustratie expliciet opgenomen. De `VehicleDto` is verder inhoudelijk niet aangepast. De `@JsonSubTypes` zijn ook van nieuwe namen voorzien (`"BusDto"` en `"CarDto"`).

In Afbeelding 3 is te zien hoe de aanroep van het POST-endpoint met deze configuratie gaat. In de JSON die gebruikt wordt om de POST te doen (bovenste JSON in Afbeelding 3), wordt nu een veld `"type"` meegegeven om aan te geven dat dit om een `BusDto` gaat. In de response (onderste JSON in Afbeelding 3), wordt dit veld ook mee teruggegeven, zodat deze meta-informatie door een aanroepende partij gebruikt kan worden.

Jackson faciliteert nog een aantal manieren waarmee de typebepaling geconfigureerd kan worden met de `@JsonTypeInfo`-annotatie. Hier wordt in dit artikel verder niet op ingegaan.

{ CONCLUSIE }

Jackson biedt een aantal mogelijkheden om subklassen te (de)serialiseren. Door een superklasse van deze configuratie te voorzien, kun je je REST API met verschillende subtypes van een DTO om laten gaan. Dit kan zonder het contract van je API te wijzigen. Aan de ene kant zorgt dit ervoor dat je API goed uitbreidbaar is. Aan de andere kant, is het wel nodig om goede documentatie over de API te verschaffen om de aanroepende partij goed op de hoogte te stellen van wat er in welke situatie aan velden nodig is. Goed om te vermelden is dat CodeGen-tools waarmee API's en API-documentatie gegenereerd kunnen worden (zoals Swagger CodeGen [1]) vaak support bieden voor het opzetten en documenteren van subtyperingen [2].

Wanneer object-oriëntatie op deze manier consequent front-to-back in een backend-applicatie wordt gehanteerd, krijg je een goed onderhoudbaar en uitbreidbaar geheel.

De code die in dit artikel gebruikt is, is te vinden op mijn GitHub-pagina [3].

L8

```
@JsonTypeInfo(
        include = JsonTypeInfo.As.EXISTING_
PROPERTY,
        property = "vehicleType",
        use = JsonTypeInfo.Id.NAME,
        visible = true
)
@JsonSubTypes({
        @JsonSubTypes.Type(value = BusDto.class,
        name = "BUS"),
        @JsonSubTypes.Type(value = CarDto.class,
        name = "CAR"),
        @JsonSubTypes.Type(value = BoatDto.
        class, name = "BOAT")
})
public abstract class VehicleDto {…}
```

L9

```
@JsonTypeInfo(use = JsonTypeInfo.Id.NAME,
        include = JsonTypeInfo.As.PROPERTY,
        property = "type",
        visible = true)
@JsonSubTypes({
        @JsonSubTypes.Type(value = BusDto.class,
        name = "BusDto"),
        @JsonSubTypes.Type(value = CarDto.class,
        name = "CarDto")
})
public abstract class VehicleDto {…}
```

{ REFERENTIES }

1 https://swagger.io/docs/specification/data-models/inheritance-and-polymorphism/
2 https://github.com/LvdKooi/jackson-vs-multiple-subclasses/blob/codegen-example/vehicle/src/main/resources/api/api.yaml
3 https://github.com/LvdKooi/jackson-vs-multiple-subclasses

CYBERSECURITY IN DEVOPS

Als DevOps-engineer ben jij verantwoordelijk voor het ontwikkelen en onderhouden van de IT-systemen van je opdrachtgever. Alsof die taken nog niet genoeg zijn, is er ineens een cybersecurity-incident op een van de systemen die onder jouw verantwoordelijkheid vallen. Nu blijkt dat je niet alleen verantwoordelijk bent voor ontwikkeling en onderhoud, je bent ook nog eens de eerste verdedigingslinie.

Raymond Jetten is cybersecurity engineer bij Team Rockstars IT. Volgens Raymond is cybersecurity een onmisbaar onderdeel van moderne softwareontwikkeling.

Hoe kan het dat je niet alleen onderhoud en ontwikkeling moet doen, maar ook nog eens de beveiliging van je systemen? Je bent niet de enige die met dit probleem worstelt. Veel DevOps-teams zitten met eenzelfde vraag of zullen hierop enig moment tegenaan lopen, simpelweg omdat ze vroeg of laat met een cybersecurity-incident geconfronteerd worden. De wereld is namelijk aan het veranderen: landen voeren digitaal oorlog, criminelen kunnen online enorm cashen en iedereen kan een (niet zo ethische) hacker zijn. Het is geen vraag meer óf je cybersecurity-incidenten kunt verwachten, maar wanneer je ze moet verwachten. Dit artikel helpt jou als DevOps-engineer om je voor te bereiden op de onvermijdelijke cybersecurity-incidenten.

{ SCOPE, INVENTARISEER EN LEER JE SYSTEMEN KENNEN }

Voordat je aan de slag kunt, moet je eerst weten wat je scope is. Het maakt niet uit of je gehele infrastructuur in een cloudomgeving hangt of in een serverpark op kantoor staat. Begin met het afbakenen van de scope van de infrastructuur die onder jouw (of je teams) verantwoordelijkheid valt. Zorg ook dat iedereen, inclusief je baas, het eens is met deze scope. Vermijd grijze gebieden waarbij meerdere partijen (of niemand!) verantwoordelijk zijn. Ben je alleen verantwoordelijk voor de applicaties die jij en je team ontwikkelen en beheren? Of wordt er ook van je verwacht dat je de Macbook verdedigd waar Harold van sales op werkt, en die hij 's avonds mee naar huis neemt om stiekem films mee te downloaden?

"JE KUNT JE SYSTEMEN NIET VERDEDIGEN, ALS JE NIET WEET WAT JE PRECIES HEBT."

Zodra je scope duidelijk is kan je gaan inventariseren. Waarschijnlijk kan je 80% uit je hoofd benoemen, maar hoe zit het met die hele specifieke details of wanneer je nieuwe collega met twee maanden ervaring een incident moet afhandelen? Zorg ervoor dat je een up-to-date overzicht van al je assets tevoorschijn kunt halen, zodra je het nodig hebt. Dus niet alleen fysieke assets, maar ook virtuele assets, software in gebruik, dataflows, enzovoort. Voor kleine omgevingen kom je misschien nog weg met een Excel Sheet, maar voor grotere en meer dynamische omgevingen ben je beter af met een van de vele beschikbare tools voor assetmanagement die op de markt te krijgen zijn (enterprise of opensource). Zorg ervoor dat je dit overzicht ook beschikbaar hebt als je productieomgeving niet beschikbaar is en dat het zo compleet mogelijk is.

Nu je weet wat je moet beschermen en hoe het eruit ziet, is het tijd om te leren hoe het écht werkt. Natuurlijk weet je wat het product doet, maar weet je ook wat er onder de spreekwoordelijke 'motorkap' gebeurt? Gebruik je opensource containerimages die telemetrie naar de uitgever sturen? Is de spring-boot applicatie die op poort 8080 luistert, ook echt de enige service op je virtuele machine die verbindingen accepteert?

Wat je ook vindt in deze eerste stappen, documenteer het. Het maakt niet uit of de resultaten precies zijn zoals je verwacht of dat je wellicht wat verrassingen gevonden hebt. De informatie die je hier verzamelt gaat jou en je collega's enorm helpen in het detecteren van mogelijke incidenten, maar ook in het analyseren en oplossen ervan.

{ BESCHERM JE SYSTEMEN }

Wellicht ben je geschrokken van wat je in de scoping-fase gevonden hebt en wil je zo snel mogelijk zorgen voor een betere beveiliging. Je kunt natuurlijk delen van je infrastructuur loskoppelen van het internet (air-gappen), multi-factor authentication afdwingen en super complexe wachtwoorden die elke week wisselen verplichten, maar is dat écht de beste manier om de server die statische website content levert te beveiligen? Waarschijnlijk niet.

Goede beveiliging begint bij het leren kennen van jezelf en je tegenstanders. Is de organisatie of het bedrijf waarvoor je werkt een grote multinational die diensten voor kritische infrastructuur levert, of een mkb'er die kinderspeelgoed verkoopt? Beide hebben te maken met een heel verschillend dreigingsbeeld (threat-environment). De multinational zal waarschijnlijk opgewassen moeten zijn tegen 'Advanced Persistent Threats' (APT's, zeer geavanceerde hackersgroepen) die uit zijn op maatschappelijke schade, de speelgoedverkoper zal vooral te maken hebben met criminelen die snel willen scoren.

Ga voordat je zelf aan de slag gaat op zoek naar beschikbare informatie. Sommige organisaties hebben interne threat-environments of threat-models beschikbaar die zijn gemaakt door de security-afdeling, anderen kunnen wellicht terecht bij een CERT (Computer Emergency Response Team) dat een specifieke sector bedient. Zo is er bijvoorbeeld een sectoraal CERT voor de zorg.

Als je weet wie je tegenover je hebt, kun je aan de slag met de implementatie van beschermende maatregelen die relevant zijn voor jouw specifieke situatie. Uiteraard zijn er een aantal 'best-practices'. Zo kunnen lange wachtwoorden, encryptie van data en multi-factor authentication nooit kwaad. Wat daarop aanvullend nog

relevant is zul je zelf moeten bepalen. Denk bijvoorbeeld eens na over hoe kwetsbaarheden in je systeem tot een minimum kunnen worden beperkt en snel kunnen worden verholpen.

Hoe dan ook, het is belangrijk om een goede balans te vinden tussen de gebruiksvriendelijkheid van het systeem en de security. Een overdreven voorbeeld is het air-gappen van de statische content van je website. Het neemt vrijwel alle reguliere manieren om je website aan te vallen weg, maar maakt normaal gebruik ervan ook onmogelijk. Maak je risico's inzichtelijk en zoek een goede balans tussen risico-acceptatie en risico-mitigatie. Er zijn frameworks en standaarden die je hierbij kunnen helpen zoals de ISO 27000/27001, het NIST Cybersecurity Framework, het NIST Risk Management Framework of OWASP SAMM.

{ AFWIJKINGEN DETECTEREN }

De voorbereiding is klaar en beschermende maatregelen zijn genomen, dus nu kan het verdedigen echt beginnen. Aangezien je DevOps-team geen volledig Security Operations Center (SOC) is, zul je zo efficiënt mogelijk moeten zijn. De meeste DevOps-teams hebben al een log-verzameling-tool in gebruik (DataDog, ElasticSearch, etc.). Dit soort tools richten zich steeds meer op security, dus om de eerste stappen op het gebied van detectie te zetten, hoef je vaak maar een paar keer te klikken om de relevante features (SIEM, Security Information & Event Management) en logs aan te zetten. De sleutel tot succes is in dit geval echter: Less is more. Ga selectief te werk met het verzamelen van securitylogs. Je wil namelijk voorkomen dat je een overdaad aan informatie krijgt waardoor incidenten in de logs staan, maar niemand ze opmerkt.

Houd ook in deze stap weer rekening met het dreigingsbeeld dat op jouw situatie van toepassing is. Verzamel vooral logs die je helpen om deze dreigingen te detecteren. Heb je vooral te maken met generieke malware? Die is vaak snel te herkennen in DNS-logs. Maak je je zorgen om ransomware? Ga dan aan de slag met Endpoint Detectie & Response (EDR) software. Ben je een mogelijk doelwit voor APT's? Zorg dan voor up-to-date Intrusion Detection/Protection Systemen (IDS/IPS) en houd de logs daarvan scherp in de gaten.

Uiteraard heeft het weinig zin om security gerelateerde logs alleen te verzamelen, je zal er ook iets mee moeten doen. Zorg ervoor dat de oplossing die je gebruikt om je logs te verzamelen jou een alert geeft op het moment dat zich een situatie voordoet die je wil voorkomen. Bijvoorbeeld: EDR-software die een ransomware-infectie voorkomt, DNS-query's naar top-level domains die je niet verwacht, of services die verkeer ontvangen op onverwachte poorten. Voor het makkelijk detecteren van dit soort events zijn SIEM-oplossingen ontwikkeld. Als je al een log-oplossing in gebruik hebt met SIEM-functionaliteit, gebruik die dan.

"ALS JE DE STATISCHE CONTENT VAN JE WEBSITE GAAT AIR-GAPPEN KUN JE UITDAGINGEN VERWACHTEN."

Natuurlijk kun je niet 24/7 naar de output van je monitoring kijken. Zorg er daarom voor dat de tool die je hiervoor gebruikt, jou op de hoogte stelt van specifieke alerts. Of dat nou een push-melding, Jira-ticket of Slack-melding is maakt niet uit. Dit is de fase waarin jij als first responder kunt shinen! Wil je nog iets verder gaan? Neem zo nu en dan even de tijd om net wat dieper in je logs en systemen te duiken. Denk eens na over hoe jij jouw eigen systeem zou aanvallen en ga vervolgens na of deze aanvallen ook daadwerkelijk plaatsvinden. Dit concept heet 'Threat Hunting' en wordt normaal gesproken door level 2 of 3 cybersecurity-analisten gedaan. Het is echter een geweldige manier om meer te leren over cybersecurity en je systemen beter te leren kennen.

{ REAGEREN OP EEN INCIDENT }

Wat als je concludeert dat we daadwerkelijk met een cybersecurity-incident te maken hebben? Ga je getroffen containers opnieuw deployen en hopen dat het niet nog eens gebeurt? Het is belangrijk om een plan klaar te hebben voor de momenten waarop het mis gaat. Het zorgt ervoor dat je een probleem gestructureerd

aanpakt en de chaos die een incident met zich meebrengt de baas kunt blijven.

Bedenk een aantal (realistische) scenario's op basis van jouw threat environment. Wat gaat er mogelijk mis en wie moet er dan op reageren? Kun je het betreffende scenario zelf afhandelen of heb je ondersteuning nodig van een incident-responseteam? Wie moet je informeren, hoe herstel je je systemen en hoe voorkom je dat een incident terugkeert net nadat je het hebt opgelost? Het zijn allemaal vraagstukken waar je vooraf over nagedacht moet hebben om chaos tijdens een incident te voorkomen.

Je kunt online veel templates en voorbeelden vinden voor het maken van een incident-response en recovery plan. Geen van deze voorbeelden zijn echter allesomvattend en specifiek voor jouw situatie geschreven. Het is daarom belangrijk om je plannen zo nu en dan te testen. Zet een component met sleutelrol uit in je testomgeving en behandel het alsof het onderdeel is van een cybersecurity-incident. Hoe reageert je team? Doet het plan wat het moet doen? Wat kan er beter?

Als laatste moet je ervoor zorgen dat je plannen mee veranderen met je infrastructuur en systemen. Het updaten van documentatie

"HET INCIDENT WAS DUIDELIJK VINDBAAR. GEWOON, IN DE LOGS."

is vaak onderdeel van de 'definition of done' van development teams. Zorg ervoor dat incident-responseplannen bijwerken daar ook onderdeel van is. Als er nieuwe componenten worden toegevoegd, zullen je plannen daar rekening mee moeten houden.

{ CONCLUSIE }

Zodra je het bovenstaande meeneemt in je DevOps-werkzaamheden, zul je aanmerkelijk beter in staat zijn om cybersecurity-incidenten te detecteren en erop te reageren. Maak daarbij gebruik van procedures en tools die je organisatie al heeft geïmplementeerd. Het verdedigen van IT-systemen tegen cyberaanvallen is namelijk een teamsport, waarbij de verdedigers alle mogelijke manieren om aangevallen te worden moeten verdedigen, terwijl de aanvallers genoeg hebben aan een enkele treffer. Maak het onderwerp daarom bespreekbaar binnen je team, management en de rest van je organisatie en ben realistisch in wat je wil bereiken. Kleine, structurele stapjes zijn ontzettend waardevol en beter vol te houden dan in één klap alles goed willen doen. <

ZIEN WE JOU BIJ ONZE GEPEPERDE CHALLENGE OP J-FALL?

CHILIT

CHILIT: VOOR DEVELOPERS MET PIT!

Ben jij een developer die méér brengt? Méér waard is? Méér pit heeft? Ontdek dan wat CHILIT voor jou kan betekenen.

Zo geniet jij straks van;

- ☑ een vast salaris,
- ☑ schaalbare bonussen,
- ☑ veel vrijheid,
- ☑ volop zekerheid!

Chilit B.V.
Utrechtseweg 75, 3702AA Zeist
088 224 5480, hello@chilit.nl
Check onze beloningsstructuur op chilit.nl

Je vindt ons op J-Fall 2022 op stand A2
(1e verdieping, bij de keynote zaal)

TIMETABLE

FLOORPLAN

J·FALL 2022

J-FALL 2022 SPECIAL

Android

IOS

Download the J-Fall app right here!

MAIN SPONSOR

ABN·AMRO

CO-SPONSORS

Capgemini

CHILIT

Rabobank

Red Hat

Rijksoverheid

Welcome to J-Fall

On behalf of the NLJUG Board, our partners, and sponsors, I am happy to welcome you to a new edition of the biggest Java conference in the Netherlands. After months of preparation, we are proud that we can host more than 1500 eager Java practitioners at our flagship event.

Last year we were very lucky to have J-Fall as an in-person conference. With some restrictions, we could host our event for 1300 Java developers. It felt great, but I am happy that we are back to a full-scale Java event without any restrictions.

This year is the 19th edition of J-Fall and the 7th that we host in the cinema of Pathé Ede. For me, this marks the 10th J-Fall in person. Many of them I joined as an attendee, some as a speaker, and this is the second time I am part of the amazing team organizing it.

The Program

This year we had an enormous amount of papers submitted for J-Fall. With over 360 submissions, this was the most successful CFP we ever had. Many of these submissions were from seasoned speakers who want to join J-Fall again, but we also received a large number of submissions from new speakers. Many of these speakers were part of our speaker mentoring program, and some of them made it into the final program, which makes us incredibly proud. I want to thank the program committee for grading every submission and creating this diverse and balanced program for J-Fall 2022. The cherry on the cake for me personally is that we have Venkat Subramaniam doing a preconference masterclass, a technical session, and the community keynote this year. Last year was great, but now we are fully back!

Early Bird sessions

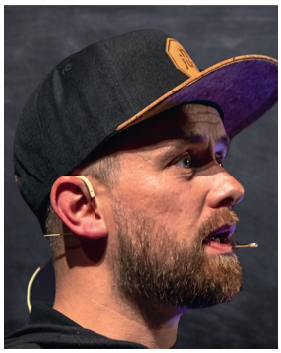
Pro tip: make sure you attend the early bird sessions. The first round starts at 8:00, before the opening keynote. The session topics of the 5 early birds are great and usually well-attended. As a bonus, you probably beat the morning traffic, and there is still enough parking space available.

Hands-on labs, Ignite sessions and Byte sizes

Next to the regular conference-type sessions, we have 2 hands-on labs in parallel. These workshops are designed in a way, so you can jump in halfway or move out early if you do not want to spend the full-time slot and see a talk. We also have 2 dedicated timeslots for byte size sessions, where you will see 3 shorter-size presentations in one timeslot. Last but not least, during lunch, we will have the ignite sessions, where the presenters will do a 5-minute timed presentation with auto-forwarded slides every 20 seconds. They might nail it or fail it, but it will be fun.

Finally

We are very honored with all the positive feedback in the community. Having J-Fall sold out in less than 24 hours is simply amazing. Don't forget to rate the session you visited in the NLJUG app and leave some written feedback, which is now possible. Also, feel free to share the good moment on Twitter using the #jfall hashtag and contact us on Twitter during the day with any questions. Alternatively, you can always reach out to the people in the NLJUG shirts if you need something.
Thank you all for your confidence. We are going to make this edition a memorable one.



Brian Vermeer *is board member of the NLJUG.*

J-Fall 2022 Highlights

This J-Fall we've got a ton of high quality content for you. You can find a few of them right here!

Keynote: The Art of Simplicity

Venkat Subramaniam
Location: Room 9
Time: 09:45-10:10

We've been told to keep things simple. It turns out, that's easily said than done. Creating something simple is, well, not really that simple. If simple was sitting next to us, would we even recognize it? Is my design simple, is yours simple? How can we tell? That's a simple question, but the answer to it is... well come to this keynote to find out.

Java Next - From Amber to Loom, from Panama to Valhalla

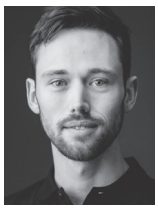
Nicolai Parlog
Location: Room 9
Time: 11:40-12:30

Java's four big projects are entering the home stretch: Amber and Panama have already incubated, previewed, and even finalized some features, Loom and Valhalla are on track to follow soon. Time to take a closer look at how...

- Project Amber makes the language more expressive and ready for today's and tomorrow's problems
- Project Panama cuts through the isthmus separating Java from native code
- Project Loom enables hassle-free and efficient structured concurrency
- Project Valhalla mends the rift in Java's type system and improves performance

After this talk, you will know what to expect from Java in the next few years.

Workshop: Choose your own adventure with Spring Security



Tim te Beek & Willem Cheizoo
Location: Hands-on Lab
Time: 14:40-17:45

Learn and apply Spring Security 5+ in a collection of individual challenges, for all levels. From picking the right dependency, to enforcing security constraints. Choose from a series of self-guided exercises around authorization to suit your needs. Topics include: OpenID Connect, Keycloak and integrating with Spring Cloud Gateway, Spring Data and Testing. Walk away with increased confidence in Spring Security and what it can do for your projects.

Modern and Lightweight Cloud Application Development with Jakarta EE 10

Ivar Grimstad
Location: Room 1
Time: 15:50-16:40

Jakarta EE 10 is packed with new features for simple development of modern, lightweight enterprise Java applications for the Cloud. The new Jakarta EE Core Profile enables developers to develop microservices with Jakarta EE technologies with runtimes smaller than ever. Jakarta EE Core Profile even makes it possible to compile Jakarta EE applications to native images to reduce the footprint even further. In this session, we will explore the new features of Jakarta EE 10 in an interactive way packed with live code demos. Among the features we will go through are the new reflection-free programming model introduced in *CDI 4.0 Lite* and the new Java SE Bootstrap API in *Jakarta RESTful Web Services 3.1*. But there will be more, so make sure you join this session to get it all! We will even take a peek into the looking glass to see what we can expect from **Jakarta EE 11**.

Visit the ABN AMRO booth at J-Fall

Dear J-Fall visitor, ABN AMRO is very proud to be the Main Sponsor of J-Fall this year! You will find us near room 8 with our stand and yes you have to come by. Why? We are giving some great LEGO prizes away. How? We distribute ABN AMRO Event cards that have a NFC chip. Try to get your hands on one and come to our booth to activate this card. After activating you will play a game and see on the big screen if you made it to the leadingboard. Come by at the end of the day around 17.30 because we will hand out all prizes to the top 10 players at that time. If you get hot from the tension, we have a solution for that too! So no excuses to play our game!These ABN AMRO Event cards can also be used on your mobile phone. Try it out!Furthermore, we have some great speakers. Check out our Mystery Keynote speaker on the main stage. Our developer Andre Groeneveld is also on stage with an interesting talk. Title: Automate your home by building your own Zigbee bridge. Info: With the huge advancements in technology over the last few decades, automating things around the house has become not only possible, but these days also easy. We have thousands of different devices to choose from, but installing them becomes more of a nuisance, as each company has their own app and bridge that you need to place somewhere in your house.
Join this talk as I'll share my journey in setting up a Zigbee bridge in my house, using a Raspberry Pi 4 and a Conbee 2 that will cater for different Zigbee enabled devices like Phillips hue and Ikea lamps. This session is not only for people familiar with home automation, but also for those who want to start automating things around the house, but wants to start and do not know where. After this session you'll be able to setup your own home automation system or make your existing system more private and remove the excess bridging devices in your home.So see you at J-Fall this year!!

Andre Groeneveld

ABN·AMRO

If I have to do this one more time, I quit!

Location: Room 7
Time: 10:35 – 11:25

Jeroen van der Wal - Full-stack software engineer @ Rabobank

You also have these moments right!? Doing the same thing for the 100th time, thinking, NOT AGAIN! Going to that same application, pressing the same buttons over and over again. Going to the same website, filling in the same data, over and over again. So... Do we really have to do this again...!? I will demonstrate how personal automation, using macro's, Alfred, but also a Stream Deck, can make your day to day work easier and more fun! I will be constantly demonstrating techniques that can be used.

So, are you curious how we do things at Rabobank? Come and visit our talk and stand!

Jeroen is a senior Full-stack software engineer at the Rabobank. He loves to share, with a lot of pasion, whatever makes him better at his work, having more fun and doing the right stuff. Jeroen started his work as an OPS engineer, and traveled via DEV/OPS to the full-stack Software Engineer postion he has now.

Rabobank

How to migrate a mission critical railway system

Location: Room 7
Time: 11:40 – 12:30

Marije de Heus - Senior software engineer @ NS

Arjen Jansen – Solution Architect @ NS

This is simple, just moving trains around through the Netherlands. Making the schedule and execute this plan. Right, of course solving the schedule jigsaw is a real hard job, but when it's done, drive around. But just being on the way, a catenary cable breaks, a tree falls down on the railway, or a broken train blocks the other train traffic. How to take care that travelers are still being served. This requires action for planning adjustments in near real-time. And systems keeping track of the actual railway network state from many data sources. Use all this data to choose the best option for optimal traveler service. Additionally, zero tolerance for downtime and continuously informing many other parties about any change in train movements.

To fully support this, we develop micro-services using contemporary and innovative technologies. Our services are deployed on a RedHat Openshift environment within clusters stretched over multiple data centers. Everything in a redundant setup, scalable and the strictest HA/HP requirements.

We will show our pipeline from test to production, which allows us to add new functionality and fix defects in production several times a day.

ING at J-Fall

We are present at J-Fall with a keynote, session and stand. Come visit us!

Location: Room 9
Time: 13:55 – 14:20

BULLET-PROOFING APIS TO BUILD RELIABLE SYSTEMS

Misunderstandings frequently happen in spoken speech. Typos in written messages are paramount. Humans can often perceive the incorrect communication's intended meaning and adjust their actions accordingly, but machines are naturally extraordinarily literal. Mistyped instructions cannot be executed. Many programming languages solved this problem by offering tools that perform exhaustive verification before any code execution is attempted. What about application programming interfaces in a world made of increasingly interconnected systems? We keep human miscommunications away from APIs by adapting ideas borrowed from the same principles found in programming language tools: compilation, assembly and linkage. This case study highlights how ING combines these techniques to bring their API quality to the next level.

Vasco Veloso - Senior Software Engineer at ING Bank NL

Location: Room 1
Time: 14:20 – 15:30

SPRING KAFKA BEYOND THE BASICS - LESSONS LEARNED ON OUR KAFKA JOURNEY AT ING BANK

You know the fundamentals of Apache Kafka. You are a Spring Boot developer and working with Apache Kafka. You have chosen Spring Kafka to integrate with Apache Kafka. You implemented your first producer, consumer, and maybe some Kafka streams, it's working... Hurray! You are ready to deploy to production what can possibly go wrong?

In this talk, Tim will take you on a journey beyond the basics of Spring Kafka and will share his knowledge, pitfalls, and lessons learned based on real-life Kafka projects that are running in production for many years at ING Bank in the Netherlands.

Tim van Baarsen - Senior software engineer @ ING

- Can you survive consuming a poison pill from a Kafka topic without writing gigabytes of log lines?
- Do you understand how to deal with exceptions in different cases?
- Do you validate incoming data consumed from a topic?
- Should you do integration testing for Kafka? And what other options do I have?
- Can you incorporate Apache Kafka as part of your distributed tracing?
- Are you able to monitor the performance of your Kafka applications to understand whether or not you are lagging behind?

In case you answer one or more above questions with 'NO' join this talk!

ING

From Monolith to Microservices at the Belastingdienst

Location: Room 1
Time: 10:35 – 11:25

This is session is about an on-going journey of a department called InkomensHeffing at the Belastingdienst towards a more adaptable, scalable and maintainable software architecture. We transform our landscape from a Cobol/Cool:Gen monolith architecture towards distributed case-wise computing with Java microservices. We have to combine this with a transformation of our software development methodology from waterfall to scrum and implementation of frequent new functional requirements. I cover successes, failures and learning experiences: How do we govern our services? How do we standardize our services? How do we transform our organization? How do we motivate and involve colleagues?
Bram is a software architect with a technical background at the Belastingdienst (Dutch Tax and Customs Administration). He believes in professional freedom and responsbility to enable individuals and organisations to flourish. He stimulates collegues to share their knowledge and experience by organizing different activities next to his work as an architect.

Bram Starmans-van den Hout - Architect at Dutch Tax and Customs Administration

Rijksoverheid

de Rechtspraak
IVO Rechtspraak

GraphQL-ify your APIs

Location: Room 6
Time: 11:40 – 12:30

Soham Dasgupta - Oracle ACE & Java Expert @ Capgemini

GraphQL is query language for APIs, but what are the advantages and how would one implement such in their microservices/APIs. In this session, I will go through the basics of GraphQL, different aspects of GraphQL and architecture of such APIs. There will be demo/live-coding on, how 4 different ways we can implement GraphQL for a Springboot microservice/API. Lots of examples, live coding and helpful comparison on structure, usage and implementations of GraphQL in Springboot & Java world.

Soham Dasgupta is by role a Solution Architect at Capgemini. He has over 16 years of IT experience in software programming, designing and architecture. He is an tech-enthusiastic and Java, Javascript expert. He writes blogs and speaks at various global technical conferences on various topic, for example Java, Javascript, Chatbots and AI. He has also created an open-source bot testing framework. His extensive work with Oracle products earned him the title "Oracle ACE".

Capgemini

Be 'Mr. Miyagi' or find one! A talk about the power of mentoring

Location: Room 8
Time: 16:55 – 17:45

Kelly Jille - Java Software Engineer at Ordina

Mr. Who? Besides one of my childhood heroes, Mr. Miyagi is a fictional karate master from Okinawa, Japan, in the movie serie 'The Karate Kid'. He was the karate mentor of several people and made them worthy champions. I wanted that too! So began my own karate journey with my own sensei (mentor). Later, I became a lawyer and got a so-called 'patroon' (another mentor). Both mentoring relationships have paid off. When I took my first steps in my career switch to the IT world, I discovered that mentorship is not a given. What a pity! Fortunately, I managed to find a mentor again. I experienced it as a difference as night and day.

I would like to talk about the mentorship's 'what', 'who', 'how' and 'why'. I would also like to explain the importance of mentorship in our field and what it could offer us as a community. In doing so, I would like to share my own experiences and ideas. To add weight to it, I make a link with research. I have looked at whether I can find my own experiences and ideas back in society, science or both. I think that we as a community, but also as an employer and client, should not want anything else and I would like to explain why. After this talk, you'll understand the power of mentoring and how you can make a difference yourself!

ORDINA

.nl. jug

NLJUG thanks our sponsors for making J-Fall 2022 possible!

MAIN SPONSOR

ABN·AMRO

CO-SPONSORS

Capgemini · CHILIT · Rabobank · Red Hat · Rijksoverheid

EXHIBITORS

adyen · a.s.r. · ASML · aws · blueriq · CAMUNDA · CGI · CONSPECT CONSULTING & ICT · FIRST8 · DEVOXX 4KIDS · GROUP9 ELITE JAVA DEVELOPMENT · IKEA · ilionx experts in eenvoud · InfoSupport Solid Innovator · ICT GROUP InTraffic Mastering Mobility · JCORE · j-next member of the human network · kadaster · luminis Conversing worlds · Microsoft · NATIONALE POSTCODE LOTERIJ · ORDINA · pan company proven IT experts · PICNIC · PLANON AIM FOR THE OPTIMUM · profit4cloud · QUALOGY · QUAD · Quintor · randstad · sogeti Part of Capgemini · SynTouch · Sytac · ROCKSTARS IT · THALES · topicus · TrailBlazers. · VX COMPANY achieve more · Xebia

TIMETABLE J-FALL 2022

	Room 9	Room 8	Room 7	Room 6
08:00 - 08:50		To be announced **Andre Groeneveld**		Spice Up Asciidoctor Documentation With Java **Hubert Klein Ikkink**
9:00 - 9:20	Opening (NLJUG)			
9:20 - 9:45	Keynote: Bullet-proofing APIs to build reliable systems **Vasco Veloso**			
9:45 - 10:10	Keynote: To be announced			
10:10 - 10:35	Coffee Break			
10:35 - 11:25	Design Patterns in the Light of Lambdas **Venkat Subramaniam**	I built my own JVM language. Here's why and how **Nataliia Dziubenko**	If I have to do this one more time, I quit! **Jeroen van der Wal**	The hidden gems of distributed tracing **Ana Maria Mihalceanu**
11:40 - 12:30	Java Next - From Amber to Loom, from Panama to Valhalla **Nicolai Parlog**	Byte size sessions **Bram Patelski, Julien Lengrand-Lambert & Daniël Spee**	How to migrate a mission critical railway system **Marije de Heus & Arjen Jansen**	GraphQL-ify your APIS **Soham Dasgupta**
12:30 - 13:30	Lunch Break			Ignite sessions **Host: Tba**
13:30 - 13:55	Keynote: NLJUG Update			
13:55 - 14:20	Keynote: Bullet-proofing APIs to build reliable systems **Vasco Veloso**			
14:40 - 15:30	Serverless Java Apps in the Cloud: MicroProfile, Quarkus, and Cloud Run **Rustam Mehmandarov & Mads Opheim**	Dekompiled **Jakob Löhnertz**	Innovation Award Sessions **Tba**	Effective Developer Testing: taking it to the next level **Maurício Aniche**
15:30 - 15:50	Coffee Break			
15:50 - 16:40	The Hitchhiker's Guide to a Great Developer Career **Sven Peters & Helen Scott**	Supercharge your Native Image applications 🚀 **Alina Yurenko**	Using Data Orchestration for speed, reliability and near real-time content **Bonnie Nti Mensah**	Canary releases with Infrastructure as Code in Java **Fernanda Machado**
16:55 - 17:45	Event-Driven Micro-services = Microservices Done Right **Urs Peter**	Be 'Mr. Miyagi' or find one! A talk about the power of mentoring **Kelly Jille**	Panel: Exploring Sustai-nability in Tech Without the Guilt-Trips **Hanno Embregts, Jan Ouwens, Jan-Hendrik Kuperus & Julien Lengrand-Lambert**	HTTP/3 and Reactive Streaming **Berwout de Vries Robles**
17:45 - 19:00	Drinks & Music			

Room 5	Room 4	Room 1	Hands-on labs 1
Pattern Matching: Small Enhancement or Major Feature? **Hanno Embregts & Peter Wessels**	Very serious gaming **Tom Eugelink**	A practical guide to designing your AWS Architecture **Steffan Norberhuis**	
What's Cooking in Maven? **Maarten Mulders**	Realtime Warehouse Analytics at Picnic with TimescaleDB **Sander Moelands**	From Monolith to Microservices at the Belastingdienst **Bram Starmans-van den Hout**	Hands-on lab: Enter Serverless Functions Journey with Quarkus **Daniel Oh**
Software archaeology: Learning from the landing on the moon! **Tobias Voß**	Co-creating with UX and Software **Simone de Gijt & Thamar Swart**	Spring Boot 3 / Spring 6: What's new? **Lennart ten Wolde**	
Premises and promises of code review **Robert Scholte & Mary Gouseti**	"Show Me the Garbage!", Garbage Collection a Friend or a Foe **Haim Yadid**	Spring Kafka beyond the basics - Lessons learned on our Kafka journey at ING Bank **Tim van Baarsen**	Hands-on lab: Choose your own adventure with Spring Security **Tim te Beek**
Byte size sessions: **Marcel Ton, Michael Krimgen & Jeroen Reijn**	Pro-Code vs Low-Code in Microservice Orchestration **Niall Deehan**	Modern and Lightweight Cloud Application Development with Jakarta EE 10 **Ivar Grimstad**	
Keep your dependencies in check **Marit van Dijk**	To be announced	Building Smart Devices using Java on the RaspberryPi: An intro to Pi4J. **Frank Delporte**	

For the latest version of the timetable, check the website or app!

Download the J-Fall app right here!

Android IOS

FLOORPLAN

GROUND FLOOR

1ST FLOOR

GROUND FLOOR

D1	Randstad	D10	AWS	D18	
D2	J-Next	D11		E1	Alliander
D3	VX Company	D12	Pancompany	E2	ING
D4	Sogeti	D13	IKEA	E3	NS
D5	JCore	D14	Picnic	E4	Intraffic
D6	Group9	D15	Qualogy	E5	Adyen
D7	Xebia	D16	Postcode Loterij	E6	Camunda
D9	First8	D17	Quintor	F	Rijksoverheid

1ST FLOOR

A1	Microsoft	B6	Conspect	B19	CGI
A2	Chilit	B7	Planon	B20	Red Hat
A3	Devoxx4kids	B8	Blueriq	B24	ABN AMRO
A4	Sytac	B9	Quad Solutions	B25	Capgemini
A5	Rabobank	B10		B27	Info Support
A6	NLJUG	B11	Topicus	B29	Rijksoverheid
C7	Luminis	B12	SynTouch	B30	Thales
C6	Ordina	B14	Team Rockstars	B33	Trailblazers
C5	Ilionix	B16	Profit4cloud		
C4	ASML	B18	Kadaster		

MAIN SPONSOR

ABN·AMRO

CO-SPONSORS

Capgemini CHILIT

Rabobank Red Hat Rijksoverheid


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

IPAD DRIVEN DEVELOPMENT USING GITPOD

Most of us Java developers use beefy machines to get our work done every day. They're powerful, can run dozens of docker images every day and keep us fresh by blowing their fans at full force when compiling. And still, there are many places where even the beefiest machine isn't the best option at hand. Maybe you're on a train with a spotty connection and need to update your dependencies. Or maybe you quickly want to fix a bug in that library you used today, but don't want to spend the time setting everything up as mentioned in the README.

Julien Lengrand-Lambert is leading Developer Relations at Adyen, a Kotlin GDE and a Gitpod Hero.

{ WHAT'S GITPOD ANYWAY AND HOW DOES IT WORK? }

Gitpod [1] solves all those problems by spinning up fresh, automated dev environments for each task in the cloud in seconds. It also has a large free offering, letting you run 50 hours even for private repos a month.

Let's try it with the infamous (Gitpod clone of the) Spring petclinic [2]. On the GitHub project page, the README hints us that the project supports Gitpod and we can simply click the 'Open in Gitpod'-button in Image 1. It may ask you to sign up (using your GitHub / Gitlab credentials), then will open an IDE in your browser and straight give you access to the code and run the application (see result in Image 2 on the next page).

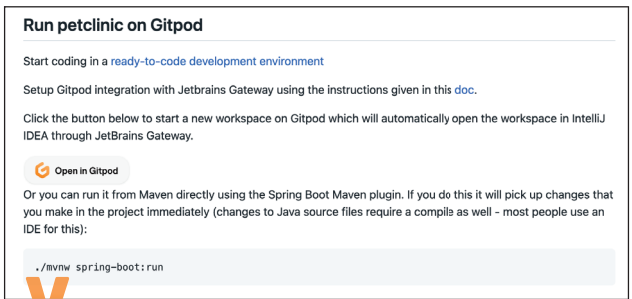


Image 1: the 'Open in Gitpod'-button on the example repository.

```yaml
tasks:
- init: ./mvnw -B dependency:go-offline package -DskipTests
  command: java -jar target/*.jar
  name: Run PetClinic app

# exposed ports
ports:
- port: 8080
  onOpen: open-preview

vscode:
  extensions:
    - pivotal.vscode-spring-boot
    - vscjava.vscode-java-pack

jetbrains:
  intellij:
    plugins:
      - com.haulmont.jpab
    prebuilds:
      version: both
```

Image 3: the .gitpod.yml file for the spring petclinic.

Running a full project and being able to modify it in a single click? Is there honestly any way to provide your users with a better developer experience than this? The magic behind this is the .gitpod.yml file that you can find at the root of the repository. The code is visible in Image 3. There actually isn't so much in it.

The `init` allows Gitpod to create 'prebuilds'. They reduce wait time, by installing dependencies or running builds before you start a new workspace.
The command is what is run every time you create a new workspace (by for example clicking on the 'open in Gitpod'-button). It does two things: checking that your environment variables are set, and then start the Spring Boot application.

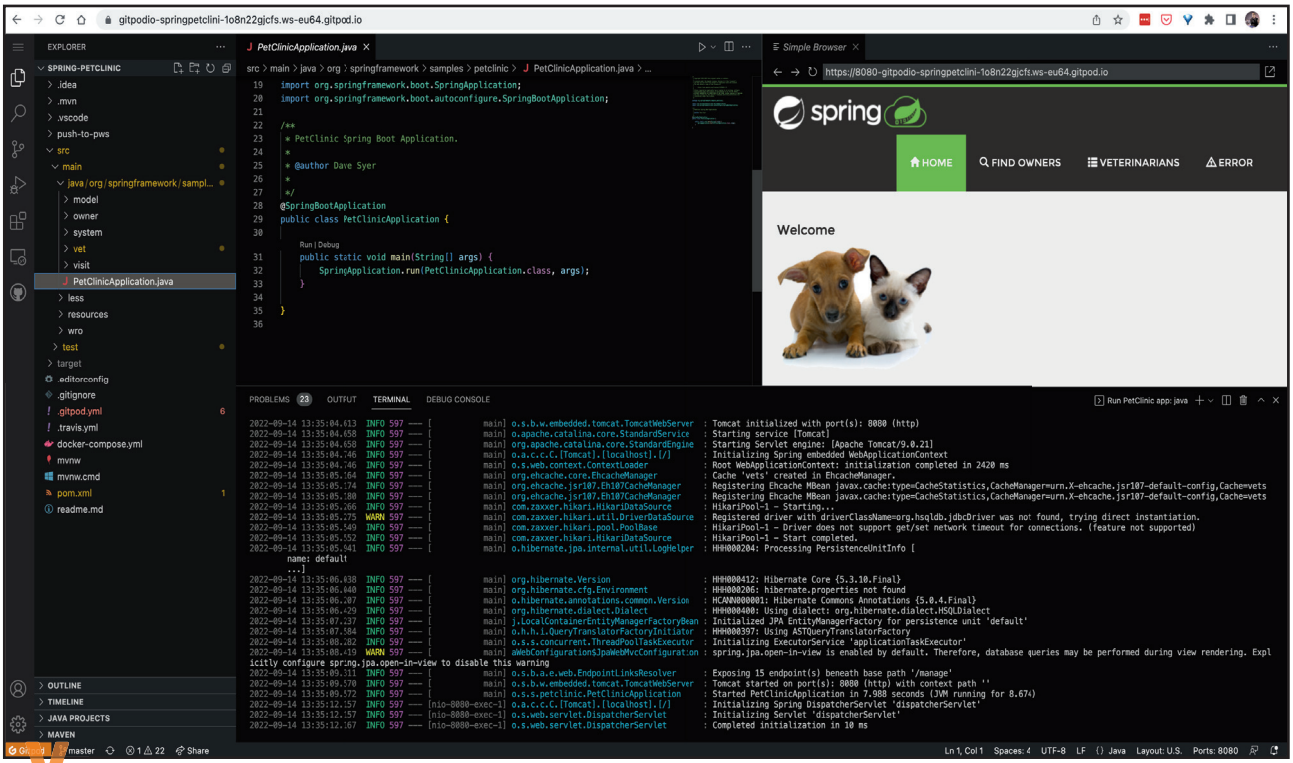


Image 2: VScode in the browser running the petclinic.

We expose the 8080 port, make it public so webhooks can reach it and to make it easy to demo it to your product owner for example. We also ask Gitpod to open a preview once the application is started.
Finally, we propose a few useful extensions for installation in the cloud instance of VSCode (and Intellij).

There are many more things you can do in the configuration file, such as defining custom docker images, or even clone additional repositories before running the code. You can read more about this in the reference page about .gitpod.yml [3].

Note that you can 'gitpodify' any project by prepending the project URL (yes, that works with Gitlab and Bitbucket as well) with https://gitpod.io/#. In our case, that would be https://gitpod.io/#https://github.com/gitpod-io/spring-petclinic. If no .gitpod.yml file is present, a template file will be generated for you, making it easy to get started.

{ OH SO LIKE GITHUB CODESPACES, RIGHT? }

If you're tech savvy, you must have heard about GitHub Codespaces (if you haven't yet, press "." on any GitHub repository for a surprise 😊).

I won't dive into much details here, Gitpod offers a whole page to describe the differences between the two products [4]. A few of the interesting things to me is that Gitpod supports Gitlab and Bitbucket on top of GitHub. We'll dive more into it in a minute, but Gitpod also allows you to keep your most loved IntelliJ IDE, which Codespaces doesn't allow at this time. Last but not least, much of Gitpod is open source [4], which is always nice to have. You never know what'll happen tomorrow (Copilot anyone?)

{ I DON'T LIKE CODING IN MY BROWSER AND I HAVE MY SETUP }

I hear you! I'm also not a big fan of developing in my browser. What's more, I mostly do Kotlin and VSCode just doesn't cut it (yet) compared to IntelliJ.

Thing is, you can absolutely keep your local tools and still use Gitpod everyday! First of all, Gitpod allows you to set many things, such as your GIT_AUTHOR, GIT_AUTHOR_EMAIL, environment variables but even provide a repository of all of your dotfiles so that you keep all of all the goodies you're used to locally.

Additionally, Gitpod allows you to pick your IDE of choice when firing up repositories. You can select VsCode or IntelliJ and it will use your local installation seamlessly (though keep in mind your workspace will still be in the cloud). Please note that IntelliJ support currently is in beta and works via the Jetbrains Gateway [6] but it works flawlessly for me.

What this means is you can actually use Gitpod without having to compromise at all with the developer experience you are used to.

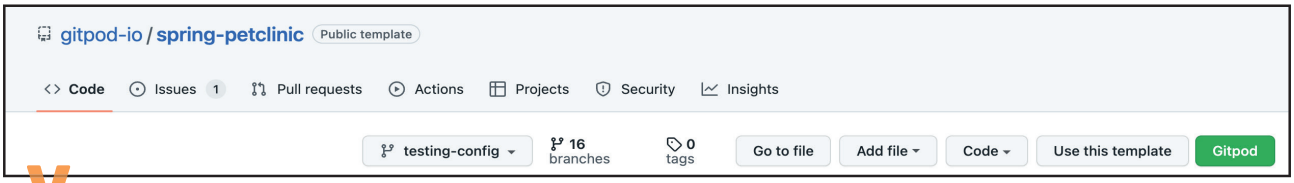


Image 4: the browser extension adds 'Gitpod'-buttons that start workspace in branches, commits and issues as well.

This is crucial to me and what really convinced me to love Gitpod so much. Up to now, my experience with remote programming was limited to connecting to virtual machines which I found unacceptable for my comfort.

{ INTEGRATING GITPOD IN YOUR WORKFLOW }

Now that I've hopefully gotten you interested in Gitpod as a development tool, let's see what it brings to the table on top of remote capabilities. One of the things I really like with the folks of Gitpod is that they are really thinking hard about developer experience and actively try to make our lives easier. Let me introduce you to 'Gitpod contexts' [7] (note that for this you need to install the browser extension [8]).

During a single working day, you may be:

- Creating an issue and starting to work on it;
- finding and fixing a typo in the README;
- reviewing a merge request or checking the status of a branch;
- adding a test to one of your project dependencies.

Because Gitpod allows you to have up to four concurrent workspaces open, you can actually do all of this at the same time, without having to switch context like you usually would.
A colleague is quickly asking us to check the 'testing-config' branch to report a problem. No problem, we navigate to the branch and click on the 'Gitpod'-button that the browser conveniently adds for us (see Image 4)!
We reproduce the problem, create a GitHub issue assigned to us and open a new workspace clicking on the 'Gitpod'-button. Gitpod is nice enough to start the project as we expect, but in a new branch that contains the issue context (Image 5).
We create a local fix, commit and push and click the 'share'-button to let our colleague know the issue is fixed. We had direct access to our workspace :) (Image 6).
It appears the issue in 'testing-config' is due to a bug in the latest version of Lombok. Let's move to that repository, create a quick issue and ... well by now you know the drill!
And just like that, we have four different workspaces open in different contexts, concurrently. Of course, this is an over the top example but I'm sure you're no stranger to having to deal with two different things at once at the office. I very much appreciate being able to fix that quick typo in the README using a new workspace rather than creating a sticky note. Both take just the same amount of time!

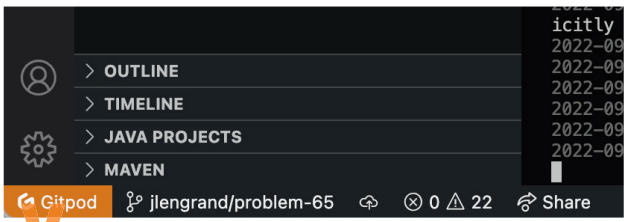


Image 5: Gitpod is smart enough to understand we want to work in a branch, and offers us to share the workspace as well.

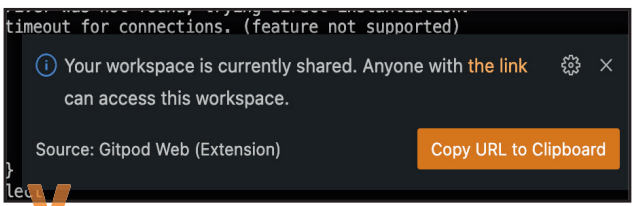


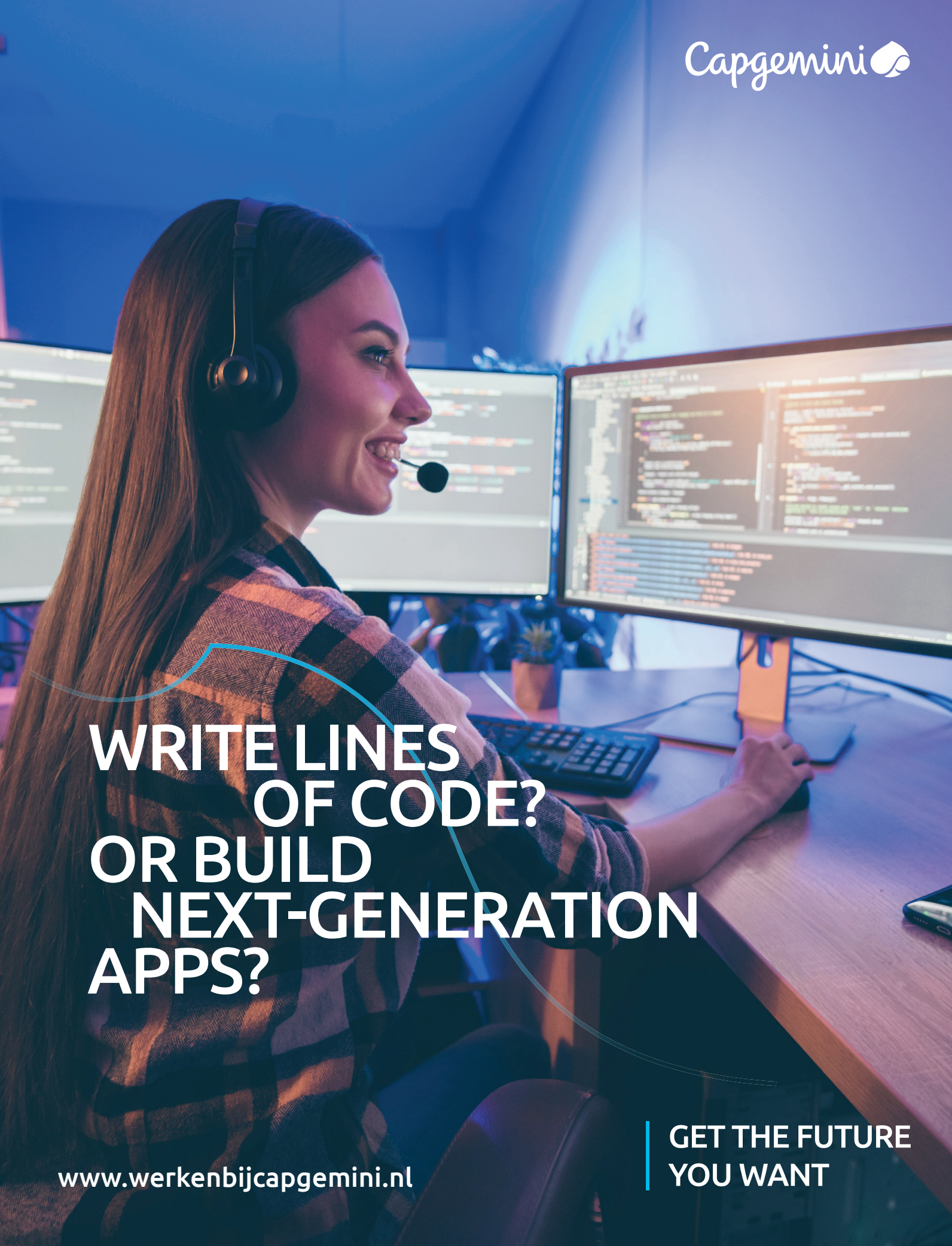
Image 6: sharing a workspace is nothing more than sharing a link.

{ A WORD OF CONCLUSION }

When you really love a project, it's hard to not make your article look like an ad. Here, I've actively tried to focus on the features that make me love Gitpod so much. And everything I have presented here comes for free. There are many more things to talk about, like the fact that you can self-host Gitpod or that some of the plans include additional collaboration features or even the CLI. In the meantime, I recommend you give Gitpod a try on one of your side projects and make it visible in the README. I'm sure the people visiting your repository will thank you, because I know I do!

{ REFERENCES }

1 https://www.gitpod.io/docs/getting-started
2 https://github.com/gitpod-io/spring-petclinic
3 https://www.gitpod.io/docs/references/gitpod-yml
4 https://www.gitpod.io/vs/github-codespaces
5 https://github.com/gitpod-io/gitpod
6 https://www.jetbrains.com/help/idea/remote-development-a.html#gateway
7 https://www.gitpod.io/docs/context-urls
8 https://chrome.google.com/webstore/detail/gitpod-always-ready-to-co/dodmmooeoklaejobgleioelladacbeki


Capgemini
WRITE LINES OF CODE?
OR BUILD NEXT-GENERATION APPS?
www.werkenbijcapgemini.nl
GET THE FUTURE YOU WANT

Volop mogelijkheden én bijdragen aan een betere wereld bij de Postcode Lottery Group

De Postcode Lottery Group is een internationale, marketing gedreven organisatie met een sociaal doel: we organiseren loterijen om geld in te zamelen voor een rechtvaardigere, groenere en gezondere wereld. Deelnemers van de Loterij winnen geweldige prijzen, en steunen tegelijkertijd goede doelen. Hoe het is om bij deze organisatie te werken? Hoofd People & Culture Reineke Ekering en Java Developer Gerben Schepers geven een tipje van de sluier.

"Ons concept zorgt voor een win-win situatie: deelnemers van de Postcode Loterij winnen geweldige prijzen én goede doelen ontvangen geld. Sinds 1989 zo'n 12 miljard.
Om dat te bereiken is internationale samenwerking en innovatie van belang." aldus Reineke. "Heb je een goed idee? Dan kun je het realiseren. Zelfs op internationaal niveau, want onze Loterijen verspreiden zich over vijf landen. Ook voor ontwikkeling, plezier en vitaliteit is veel aandacht, met onder andere incompany trainingen en lezingen, gratis ontbijt en sportmogelijkheden. Maar bovenal is de Loterij een plek met een fijne sfeer waar iedereen welkom is. Onze deur staat altijd open voor enthousiaste nieuwe collega's."

Gerben is in januari begonnen als Java Developer bij de Postcode Lottery Group. "In deze negen maanden heb ik de kans gekregen om in meerdere teams te werken, waaronder een internationaal team met verschillende disciplines. Samen werken we aan herbruikbare elementen, zoals een designsysteem en verschillende services voor al onze Loterijen. We draaien alles serverless via AWS en kunnen daardoor volop focussen op de businesslogica in plaats van beheer. Door alle mogelijkheden om te leren en de goede sfeer binnen het team voel ik mij hier helemaal op mijn plek."

Postcode Lottery Group

NATIONALE POSTCODE LOTERIJ · SVENSKA POSTKOD LOTTERIET · PEOPLE'S POSTCODE LOTTERY · DEUTSCHE POSTCODE LOTTERIE · NORSK POSTKODE LOTTERI

Advertorial

Zes tips voor het schrijven van een goede blog

Of je nu een informatiejunkie bent, nieuwe collega's wilt aanspreken of graag schrijft; bloggen helpt je jezelf persoonlijk en zakelijk te ontwikkelen. Maar als je begint, blijkt het toch niet zo eenvoudig. Hoe start je, wat is je onderwerp en hoe spreek je de doelgroep aan? Deze tips van How2Blog wakkeren jouw creativiteit aan!

1. Onderwerp: Start bijvoorbeeld met een blog over je ervaringen op J-Fall.
2. Kop: Kies een goede oneliner. Bijvoorbeeld: 'Vijf dingen die ik leerde op J-Fall'.
3. Maak je blog scanbaar met een een intro, een middenstuk en een conclusie. En vergeet de subkoppen niet.
4. Wees goudeerlijk, positief kritisch en gebruik sarcasme en humor.
5. Zonder input geen output; maak notities van de sessies en bespreek jouw bevindingen in je blog.
6. Maak veel foto's die je meteen in je blog kan verwerken.

Nu is het slechts een kwestie van de blaren op je vingers typen. Heb je toch nog moeite met bloggen, dan helpen Tirza van Hengstum en Daphne Keislair je graag op weg. Met onze trainingen voor IT'ers zorgen we ervoor dat je het bloggen zo onder de knie hebt.

Meer weten? Scan de QR Code. We zijn zelf ook op de conferentie (te herkennen aan ons t-shirt), dus we zien je daar.

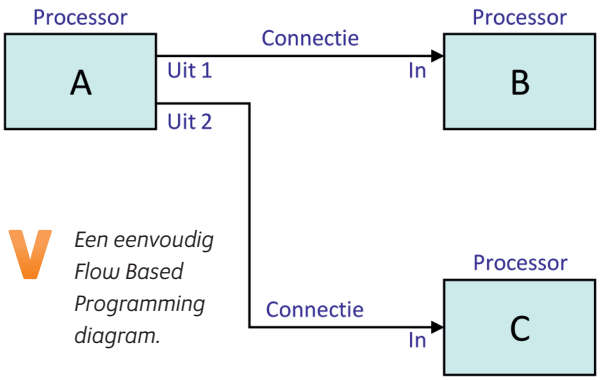
www.how2blog.nl/j-fall

Scan mij!

APACHE NIFI
flow gebaseerde integratie

Integratieuitdagingen kunnen complex zijn. Naast dat er veel aspecten spelen, zoals security en performance, is er ook vaak sprake van geregelde veranderingen in bijvoorbeeld gekoppelde systemen en berichtdefinities. Apache NiFi [1] is een in Java gebouwd open source flow gebaseerd integratieproduct die het opzetten, onderhouden en real-time wijzigen van gegevensstromen eenvoudig maakt. Zogenaamde 'flows' worden gedefinieerd in een grafische omgeving waar designtime en runtime hetzelfde zijn. Het product heeft uitgebreide, ingebouwde securitymogelijkheden en is gemaakt voor het omgaan met grote datasets. Loop je tegen een integratieuitdaging aan? Kijk ook eens naar Apache NiFi!

Maarten Smeets is een ervaren software architect met focus op data, integratie, security en performance, hoewel hij andere uitdagingen ook met veel plezier oppakt.



Een eenvoudig Flow Based Programming diagram.

NiFi vindt zijn oorsprong bij de NSA waar de eerste versie in 2006 gereleased werd. Het werd daar gebruikt om te zorgen dat informatie uit diverse bronnen snel terecht kwam waar deze het meest van waarde was. In 2014 is NiFi vanuit de NSA aan de Apache foundation gedoneerd. De mensen die aan NiFi gewerkt hebben bij de NSA zijn uiteindelijk grotendeels bij de start-up Onyara Inc terechtgekomen. Dit bedrijf is overgenomen door Hortonworks in 2015. Hortonworks is vervolgens gefuseerd met Cloudera in 2019. Cloudera biedt NiFi commercieel aan als Cloudera Flow Management [2].

{ FLOW BASED PROGRAMMING }

NiFi is gebaseerd op de principes van Flow Based Programming (FBP) [3]. J. Paul Morrison heeft dit paradigma in 1969 bij IBM ontwikkeld. Hij definieerde applicaties als netwerken van black-boxprocessoren die data uitwisselen door middel van connecties. Deze connecties zijn vastgelegd buiten de processoren. FBP is dus van nature component gebaseerd.

Het flow gebaseerd programmeren heeft een aantal voordelen die je in NiFi terugziet, zoals:

Foutafhandeling

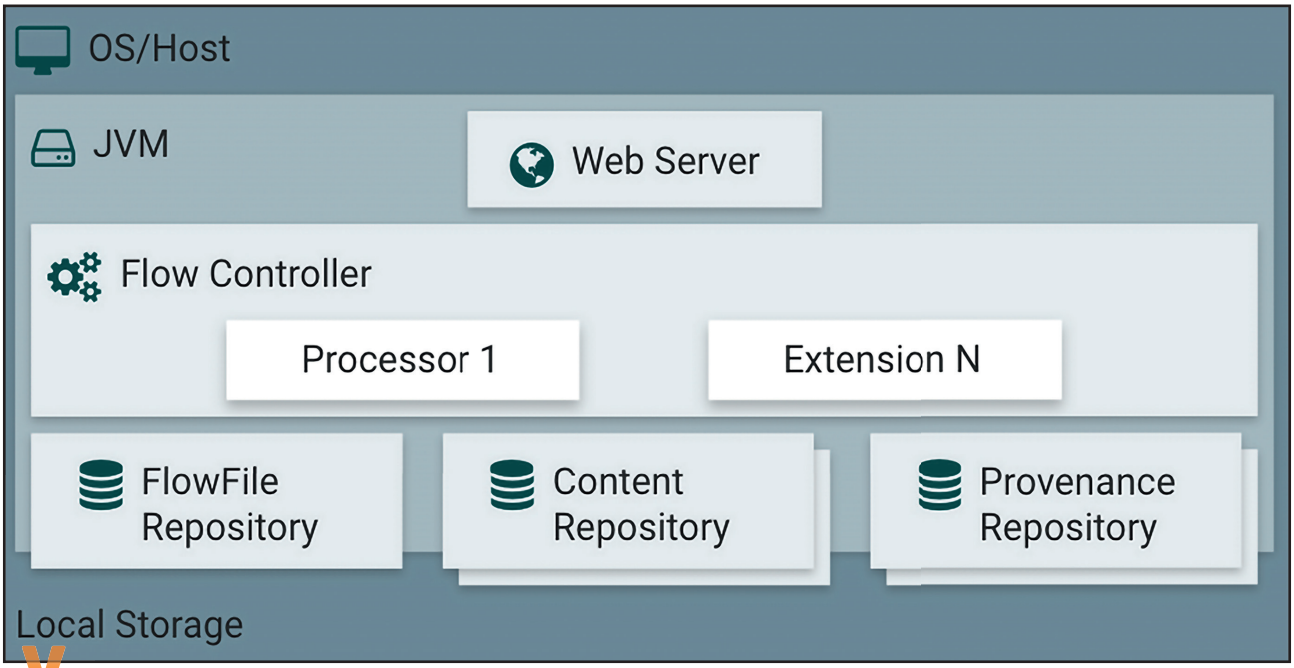
Foutafhandeling is heel expliciet. Een processor heeft vaak een of meerdere connecties die binnenkomen en een of meerdere die eruit kunnen komen. De 'failure'-connectie moet expliciet ergens naartoe worden geleid of bij de desbetreffende processor afgebroken worden, omdat anders de processor 'invalid' wordt en niet geactiveerd kan worden.

Eenvoudige concurrency

Berichten stromen over het algemeen asynchroon en onafhankelijk van elkaar door NiFi. Mede hierdoor is het bijna vanzelfsprekend dat er grote hoeveelheden berichten gelijktijdig verwerkt kunnen worden, zonder dat je hier expliciet maatregelen voor hoeft te nemen.

Back Pressure

Processoren zijn verbonden met connecties. Een connectie kan je zien als een queue. Per connectie is instelbaar hoeveel berichten



NiFi architectuur van een enkele node.

deze mag bevatten en hoeveel schijfruimte hij in mag nemen. Als dit bereikt is, wordt de aanroepende processor geïnstrueerd geen nieuwe berichten meer aan te bieden. Uiteindelijk zal de invoer geen nieuwe berichten meer oppakken, totdat er weer ruimte beschikbaar komt. Hier hoef je niets voor te doen. Dit is out-of-the-box-functionaliteit die eenvoudig te configureren is.

{ ARCHITECTUUR }

NiFi maakt gebruik van zogenaamde 'FlowFiles'. Dat zijn de dataobjecten of berichten die door een NiFi-flow stromen. Een FlowFile bestaat uit content (het bericht) en attributen (metadata). Een flow is een aaneenschakeling van processoren en connecties die vaak gebundeld worden in zogenaamde 'process groups'. Op een process group kunnen rechten worden toegekend en een process group kan als eenheid geversioneerd worden.
NiFi draait op een JVM en maakt gebruik van lokale storage voor zijn repository's.

De FlowFile-repository wordt gebruikt om metadata bij te houden van FlowFiles, terwijl ze onderweg zijn in NiFi. De content-repository bevat de inhoud van een bericht en wijzigingen aan de inhoud. De provenance-repository houdt bij wat er met een FlowFile gebeurt en de toestand van de metadata op dat moment. Het bevat een pointer naar de content in de content-repository. Je hebt dus geen database nodig om NiFi te kunnen gebruiken. Een NiFi-instantie is, vanwege gebruik van de local storage, niet stateless.

NiFi kan ook geclusterd worden. Bij het clusteren wordt gebruik gemaakt van Apache ZooKeeper. Elke node bevat dezelfde flow-definities. De individuele nodes verwerken echter wel verschillende FlowFiles en maken daarbij gebruik van hun eigen local storage. Dit is van belang als je de oplossing voor hoge beschikbaarheid in wil richten en bijvoorbeeld ook als je het wil draaien op Kubernetes.

{ ONTWIKKELEN }

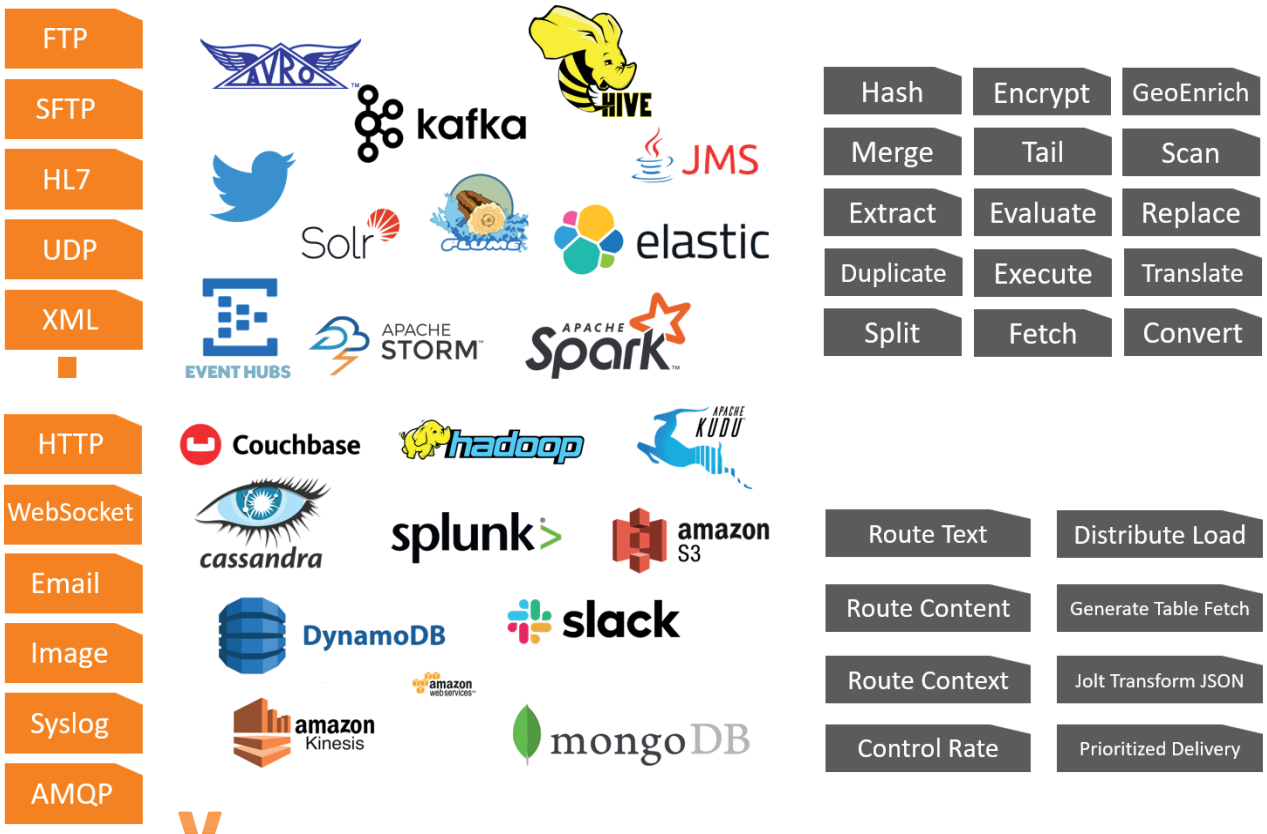
Ontwikkelen in NiFi doe je in de webinterface die beschikbaar is. Hierin kun je grafisch de processoren en connecties op een canvas tekenen en vervolgens configureren. Bij het ontwikkelen van flows kan je gebruik maken van templates.

{ PARAMETERS }

Processoren moeten geconfigureerd worden. Voor de configuratie kan je parameters gebruiken. Deze gebruik je bijvoorbeeld voor eigenschappen die specifiek zijn voor een omgeving of die toegepast kunnen worden op een template om deze specifiek te maken. Parameters worden gegroepeerd in 'parameter contexts' en deze contexts ondersteunen (vanaf NiFi 1.15) overerving. Een parameter context kan je vervolgens toewijzen aan een process group.

{ PROCESSOREN }

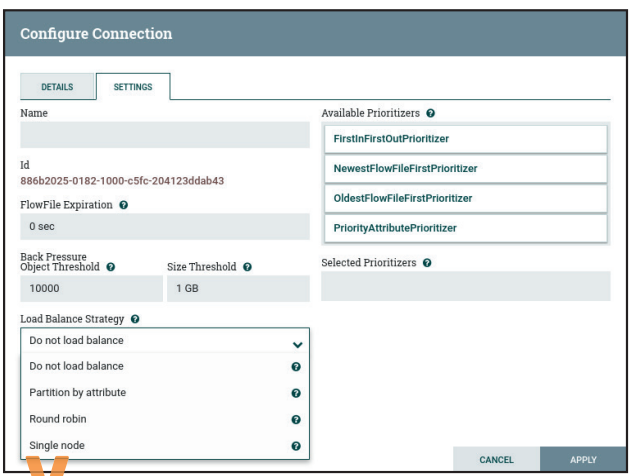
De processoren doen het zware werk. Er zijn er meer dan 300 in de laatste versie out-of-the-box beschikbaar. Je gebruikt ze voor het lezen of schrijven van bron- en doelsystemen, het uitvoeren van acties op data zoals bijvoorbeeld transformaties, routering, logging en validaties. Er zijn zelfs processoren die je als (query gebaseerde) change-data-capture-oplossing in kan zetten om zo wijzigingen in een bronsysteem op te kunnen pakken.



NiFi processoren doen het zware werk.

{ CONNECTIES }

In de configuratie van een connectie kan je prioritering van berichten configureren met zogenaamde 'Prioritizers'. Dit kan bijvoorbeeld op basis van een veld in het bericht of op basis van wanneer het bericht is binnengekomen. Ook kun je instellen na hoeveel tijd berichten komen te vervallen en niet meer verwerkt hoeven te worden ('FlowFile Expiration'). Back Pressure is in te stellen op basis van het aantal berichten ('Object Threshold') en/of de grootte van de queue ('Size Threshold'). Connecties zijn ook het middel om load te verdelen over nodes in een cluster.



Configuratie van prioritering, back pressure en load balancing.

{ DE NIFI REGISTRY }

De NiFi Registry kan je gebruiken voor versiebeheer. Default gebruikt deze een eigen filestorage. Als alternatief kan hij flows in een Git-repository opslaan. Process groups kun je versioneren. Als er een nieuwere versie van een process group in de registry gecommit is, zie je dat terug in de webinterface en kan je op andere omgevingen die aan dezelfde registry gekoppeld zijn, de process group met de nieuwe versie upgraden. Ook is het eenvoudig om te zien welke lokale wijzigingen er zijn. Als er bij lokale wijzigingen ook een nieuwere versie beschikbaar is, is er sprake van een conflict en kan je de versie niet bijwerken zonder lokale wijzigingen eerst terug te draaien. Dit werkt goed in een klein team, maar branching en intelligente conflictresolutie zijn bijvoorbeeld niet beschikbaar, wat uitdagingen kan geven als het NiFi-team groter wordt.

{ UITBREIDINGSMOGELIJKHEDEN }

Hoewel er veel standaard beschikbaar is in NiFi, is het ook zeer geschikt om uit te breiden:

- Er zijn scriptingmogelijkheden binnen processoren voor Clojure, ECMAScript, Groovy, Lua, Python en Ruby.
- Je kan custom processoren, controller services, reporting tasks,

prioritizers maken in Java. Deze kan je eenvoudig bouwen met Maven.
- De uitgebreide API maakt het mogelijk om elke actie die je via de gebruikersinterface uit kan voeren ook te scripten. Dit wordt veel gebruikt om taken te automatiseren.

{ ECOSYSTEEM }

Veel is al onderdeel van NiFi, maar bepaalde zaken zijn geen onderdeel van het Apache NiFi-project en heb je wel vaak samen met NiFi nodig. Een aantal voorbeelden hiervan:
- Een SDK om makkelijk programmatisch van de NiFi API gebruik te kunnen maken. NiPyAPI is een populaire Python SDK hiervoor. Voor Java is er bijvoorbeeld nifi-swagger-client.
- Deployment-scripts. Ansible-scripts zijn beschikbaar en worden regelmatig gebruikt. Er zijn ook Chef- en Puppet-scripts beschikbaar.
- Deployment-scripts om NiFi uit te rollen op containerplatformen. Er zijn Kubernetes-operators beschikbaar, Helm-charts en OpenShift-templates.

{ USE CASES }

Zoals jullie hebben kunnen lezen, is er erg veel mogelijk met Apache NiFi, maar waar zet je het vooral voor in en wanneer is het beter om een ander integratieproduct te kiezen?

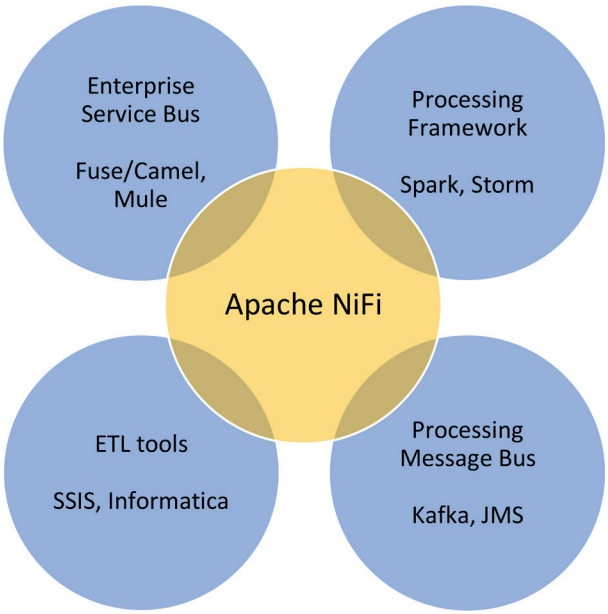
NiFi heeft als product overlap met ETL-oplossingen, service-bus-producten, processing frameworks en message brokers. Het wordt ook wel 'the swiss army knife of data movement' genoemd en een 'big data accelerator'. In die hoek zal je het product ook vooral tegenkomen.

NiFi is goed in snel, betrouwbaar, traceerbaar grote hoeveelheden data van A naar B te brengen en daar eenvoudige zaken op te doen als routering of transformatie. Het is niet geschikt om uitgebreide businesslogica in te bouwen of om bijvoorbeeld complexe berekeningen in te doen. Daar zijn andere oplossingen over het algemeen meer geschikt voor.

{ WAAR WORDT HET IN DE PRAKTIJK VOOR INGEZET? }

Je ziet NiFi veel in combinatie met big-data-trajecten. Hier is vaak een requirement dat data uit veelal technologisch diverse bronnen geaggregeerd moet worden in bijvoorbeeld een data lake, zodat er analyses op uitgevoerd kunnen worden. Een voorbeeld hiervan is waarvoor de NSA het product initieel heeft opgesteld.

Iets dichter bij huis gebruikt een grote energieleverancier NiFi om de brug te slaan tussen specifieke technische pakketten die vaak beperkte mogelijkheden tot integratie bieden met hun Kafka-platform. Een spoorwegmaatschappij zet NiFi in om op een gecontroleerde manier efficiënt data tussen cloud en on-premises uit te wisselen. Een grote overheidsorganisatie maakt gebruik van NiFi om gegevens vanuit verschillende bronnen beschikbaar te maken in hun analyticsplatform.



NiFi positionering.

{ WELKE UITDAGINGEN ZIJN ER? }

Hoewel NiFi een prachtig product is waar je erg veel mee kan, zijn er natuurlijk ook uitdagingen. Een belangrijke is CI/CD en dan met name op het vlak van de uitrol van flows naar andere omgevingen en het geautomatiseerd testen van flows.

{ TOT SLOT }

Apache NiFi is een krachtig open source Java-gebaseerd integratieproduct dat zeer geschikt is om grote hoeveelheden data op een gecontroleerde en veilige manier van A naar B te krijgen. Het kan draaien on-premises of in cloudomgevingen, op bare metal, in een container of op Kubernetes onder zowel Linux als Windows.

NiFi heeft een actieve community en er komen zeer regelmatig nieuwe versies uit met nieuwe features en bugfixes. Er zijn mogelijkheden om met zeer uiteenlopende systemen te kunnen koppelen. Het is ook eenvoudig NiFi zelf uit te breiden mocht het nodig zijn.

Als je op zoek bent naar een laagdrempelige maar toch krachtige en flexibele integratieoplossing waar goed over is nagedacht, zou ik vooral eens kijken of Apache NiFi een oplossing voor je use case zou kunnen zijn!

{ REFERENTIES }

1 Apache NiFi: https://nifi.apache.org
2 Cloudera Flow Management: https://docs.cloudera.com/cfm
3 Flow Based Programming: https://en.wikipedia.org/wiki/Flow-based_programming

SNEL PIPELINES OPTUIGEN MET AZURE DEVOPS

Nieuwe dingen leren, werken aan de Cloudtransitie of gezellig karten met collega's. Bij IVO Rechtspraak hoeft software engineer Jeroen zich geen dag te vervelen. Met zijn DevOps-team werkt hij aan het Digitaal Werkdossier (DWD). Dat is een belangrijke applicatie waarmee rechters en andere medewerkers van de Rechtspraak digitaal kunnen samenwerken. 'Bewaring gebruikt het al, en de andere afdelingen kunnen niet wachten. Een goed teken natuurlijk.'

Jeroen is software engineer bij IVO Rechtspraak . Toen hij vader werd, trok de vastigheid en flexibiliteit die de Rijksoverheid te bieden had hem aan.

Binnen software development ligt de focus momenteel op de tooling voor de delivery van de applicatie, dus als het systeem van de ontwikkelomgeving naar de productieomgeving gaat. Jeroen bijt zich het liefst vast in complexe functionaliteiten. 'In mijn werk kijk ik niet alleen naar de software, maar ook naar de processen erachter. En ik praat met gebruikers over wat ze willen. Dat maakt wat ik doe heel divers. Wat mij blij maakt? Als collega's iets goeds zeggen over mijn werk, en het naar de productie mag.'

{ VERNIEUWEN MET OPENSHIFT EN AZURE }

Jeroen ontwikkelt zijn kennis en ervaring graag. Dat komt mooi uit, want IVO Rechtspraak is volop aan het migreren naar een nieuw platform en nieuwe technieken. 'Zo hebben we een nieuwe versie van het OpenShift-platform, dat brengt veel veranderingen met zich mee. Daarnaast zijn we bezig om al onze pipelines om te bouwen van Jenkins naar de DevOps-omgeving van Microsoft Azure in de cloud. Het zijn er een heleboel, want bij IVO werken we met microservices.'

{ FLEXIBELE MICROSERVICES }

Het Digitaal Werkdossier bestaat uit tientallen verschillende microservices. Bijvoorbeeld voor het inlezen, verwerken en doorzoekbaar maken van bestanden of het maken van annotaties in dossiers. Jeroen is ontzettend enthousiast over structuren met microservices. Het is een van de redenen dat hij graag bij IVO wilde werken. 'Ook hebben we een heel leuk team. Onze backend kan zich echt aan elkaar optrekken. Ik werk veel thuis, maar we hebben de hele dag door contact via Slack. En na het werk gaan we regelmatig karten of boulderen.'

{ PIPELINES BOUWEN }

Azure DevOps is upcoming en wordt door veel bedrijven omarmd. Je kunt er snel pipelines mee optuigen doordat je met herbruikbare objecten werkt, een soort bouwblokken. Jeroen: 'Het kost je een paar dagen om een pipeline voor een microservice te bouwen, maar daarna heb je er binnen een paar minuten eentje klaar voor een vergelijkbare service. Ook kun je heel snel updates doorvoeren in je pipelines. Dat gaat centraal, per object. Je hoeft dus niet alle pipelines door voor een wijziging. Wil je als IT'er relevant blijven, dan moet je met dit soort nieuwe technieken uit de voeten kunnen.'
Nieuwsgierig naar de nieuwe technieken die IVO Rechtspraak gebruikt? Bezoek de stand van de Rijksoverheid tijdens J-Fall.

{ CHECK DE VACATURES }

Meer verhalen van IT'ers bij de Rijksoverheid en de impact op hun werk op Nederland vind je op Werkenvoornederland.nl. Scan de QR-code voor de meest actuele vacatures <

de Rechtspraak
IVO Rechtspraak

Rijksoverheid

HOW IT'S MADE: REDACTIELID ANNO 2022

"Pling!" Er staat een nieuw berichtje in het Java Magazine Slack-kanaal. Ik ben eigenlijk aan het werk, maar als Slack open staat, lees ik wel vlug door de berichtjes. Het berichtje is van Stijn, onze contentmanager. Iemand heeft een abstract naar de NLJUG gemaild en wil een artikel schrijven. Ik markeer Stijns bericht als 'ongelezen'. Dit kan tot vanavond wachten. Dan ga ik de abstract lezen, de auteur half stalken en een mening vormen of zo'n artikel goed in het Java Magazine zou passen. Met Slack-duimpjes en hier en daar wat discussies in de chat vindt de redactie consensus of en in welke vorm we het artikel vinden passen. Stijn houdt in een Excel Sheet het overzicht over de vier edities die per jaar verschijnen.

Het is een paar dagen later en we willen het artikel hebben, maar wie gaat er nu mee verder? Eigenlijk heel eenvoudig: wie er wil. De vraag wordt openbaar in de groep gesteld en er is altijd iemand te vinden die én de tijd heeft én het onderwerp inhoudelijk goed kan reviewen. Deze keer zal ik dit zijn en neem ik vervolgens contact op met de (potentiële) auteur. Gewoon met een mailtje, waarin ik me voorstel en wat informatie deel, zoals de editie en deadline die we voor het artikel in gedachten hebben. And then we wait. ;)

"Pling!" Dit keer is het een mail van de auteur die zijn artikel nu via Google Docs met me deelt. Weer tijdens werktijd, zucht. Dit mag wel even en ik stuur een kort antwoord dat ik binnenkort naar het artikel ga kijken. En dan staat het op mijn to-do-lijst en doe ik mijn best, zodat ik toch wel binnen een week naar het artikel kijk. Ik lees het artikel gewoon door. Dan begin ik met Googlen, verdiep ik mij in het onderwerp en maak ik wat aantekeningen. Het artikel moet natuurlijk inhoudelijk correct zijn, maar vooral ook in z'n geheel een afgerond verhaal worden en voor de lezer interessant en leuk zijn om te lezen.

Natuurlijk ben ik maar een enkele lezer, maar ik geef vervolgens feedback op het artikel over hoe ik het ervaar en waar het misschien duidelijker of anders kan. Ik schrijf ook wel eens wat en weet hoeveel moeite schrijven kan kosten. Daar hou ik in mijn commentaar rekening mee. Soms is het erg lastig om de vinger erop te leggen, waarom iets niet lekker leest. En juist dát vind ik persoonlijk geweldig: de oorzaak vinden, dit verwoorden en een mogelijk alternatief verzinnen.

En "plong", daar gaat uiteindelijk mijn reactie weer terug naar de auteur. Die verwerkt mijn feedback naar eigen inzicht. Wat ook betekent dat een suggestie soms niet wordt overgenomen en dat is oké. Het blijft de tekst van iemand anders. Meestal komt dan nog een kleine extra ping-pong-ronde met wat nuances in de tekst, bijvoorbeeld woordherhalingen die nog even bijgeschaafd worden.

Nu is het tijd ook. De deadline voor de volgende editie was namelijk gisteren al. Gelukkig is het artikel af en kan het door richting eindredactie, wat concreet betekent dat ik de Google Docs-link in de Excel Sheet plak en Stijn een berichtje via Slack stuur dat het klaar is.

"Pling!" Oh nee, weer een berichtje van Stijn. Was ik toch weer de biografie van de auteur vergeten bij te werken. Op de valreep dit laatste stukje nog rechttrekken en het artikel is nu ECHT klaar voor de print.

Wil jij ook lid worden van de Java Magazine redactie? Dat kan! Wij hebben nog plek.

{ JAVA MAGAZINE EDITORIAL COMMITTEE VACANCY }

First of all, thank you for your interest in contributing to the NLJUG. Everyone is welcome to reply to this vacancy, but do note: we are looking to increase the diversity of the editorial committee. This vacancy closes the 30th of November.

{ WHAT IS JAVA MAGAZINE? }

Java Magazine is a magazine about the programming language Java and related technologies. The magazine is published by the NLJUG in collaboration with publisher Reshift Digital, and arrives at the homes of the NLJUG members and its business partners four times a year.

Java Magazine contributes to the professional development of IT experts who work with Java. Furthermore, it helps to keep track of the hottest new technologies and offers a moment of rest away from the computer screen. This is made possible by a physical magazine which contains workshops, reviews of interesting or new technologies, columns and much more.

{ WHAT DOES IT MEAN TO BE A JAVA MAGAZINE EDITORIAL COMMITTEE MEMBER? }

- Everything is decided in a democratic manner.
- You think about the current state and future of Java Magazine, as in content, look & feel and brand.
- You get a vote in the selection of articles.
- You coach (new) authors in writing articles, meaning you will help with the content, writing style and inform them about deadlines and submitting the articles.
- Actively recruit (new) authors.
- Join 4 in-person Java Editorial Committee meetings per year.

{ WHAT DO WE EXPECT FROM YOU? }

- Knowledge about Java (at least 2 years of experience with Java and related technologies);
- you have a sufficient professional network;
- you clear time to give input and react quickly;
- an excellent mastery of the Dutch language;
- pre: excellent mastery of the English language;
- pre: you have written multiple articles (or blogs);
- pre: you have at least 5 years of experience with Java and related technologies;
- you are proactive & enthusiastic! :)

Interested? Send an email with motivation to **info@nljug.org**.

FIRST8 | CONCLUSION

MASTERS OF JAVA 2022

Funprogging contest voor alle Java Developers

De Masters of Java is een roemruchte “funprogging contest” (gebaseerd op Java SE), die toegankelijk is voor iedere Java ontwikkelaar. In deze wedstrijd wordt niet de API-kennis, maar de echte programmeer-vaardigheid getest. In voorgaande edities heeft Masters of Java al heel wat verhitte strijd opgeleverd!

Er zijn 5 zeer diverse programmeeropdrachten voor de teams. Deze moeten binnen de tijdslimiet worden opgelost. Voor elke seconde die je overhoudt, scoor je een punt. Ook kun je extra punten scoren met bepaalde testen. Het team met de meeste punten wint.

Een team bestaat uit maximaal 2 ontwikkelaars en het team dat aan het einde van de dag de meeste punten heeft, mag zich “Master of Java 2022” noemen. Er zal naast eeuwige roem natuurlijk gestreden worden voor mooie en spectaculaire prijzen. De afgelopen jaren is de beslissing steeds gevallen tijdens de laatste opdracht. Het belooft dus een zinderende wedstrijd te worden!

Schrijf je nu in voor de Masters of Java 2022.

EEN NLJUG EVENT

2022 MASTERS OF JAVA nl.jug

Wanneer: woensdag 2 november
Tijd: 13.00u tot 17.30u
Waar: De Reehorst, Ede
Kosten: Gratis

Als trotse hoofdsponsor zorgt First8 | Conclusion elk jaar voor 5 uitdagende opdrachten en een robuuste game server. Met veel enthousiasme, bevlogenheid en passie werken wij aan deze en andere gave projecten. Iets voor jou? We maken graag kennis met je.

Benieuwd naar eerdere opdrachten?
Check https://github.com/First8/mastersofjava

www.first8.nl

Aanmelden: nljug.org/masters-of-java-2022

WAT IS FLUTTER EN HOE KUN JE HET LEREN?

Flutter is al enige tijd beschikbaar en wordt steeds vaker gebruikt voor het bouwen van apps. Het wordt niet alleen gebruikt op het gebied van mobile development, maar ook op het gebied van web- en desktopapplicaties. Tijd om eens te kijken hoe het er voor staat en wat er nieuw is in Flutter 3.0.

Rogier van Apeldoorn is een CodeSmith bij Ordina JS Roots en heeft een passie voor mobile app development.

Ik krijg nog regelmatig de vraag van mensen “Wat is Flutter?”, daarom zal ik in het kort uitleggen wat Flutter precies is. Uiteindelijk wil ik met dit artikel een eerste aanzet geven hoe je het beste kan beginnen met het leren van Flutter.

{ WAT IS FLUTTER? }

“Write once, run everywhere” (WORE) was in 1995 een slogan waarmee Sun Microsystems illustreerde hoe Java-crossplatform gebruikt kon worden. Deze slogan kan ook heel goed toegepast worden op Flutter. Flutter is een opensource framework geschreven in de programmeertaal Dart. Zowel Flutter als Dart zijn ontwikkeld door Google en worden actief door Google zelf gebruikt.

Met Flutter kan je apps bouwen voor de volgende platformen en dat alles met één codebase:

- Android
- iOS
- Web Chrome/Firefox/Safari/Edge
- Linux
- macOS
- Windows

{ DART }

Omdat Flutter gebruik maakt van Dart, hier eerst nog wat meer over Dart. Dart is class defined, garbage-collected en objectgeoriënteerd. De codesyntaxis lijkt op die van Java/Kotlin/C. In Dart hebben we interfaces, mixins, abstract classes, generics, static typing en een sound-typingsysteem (hierover later meer). Het mooie aan de Dart-compiler is dat we de beschikking hebben over een Just-in-Time (JIT)-methode en een Ahead-of-Time (AOT)-methode.

Just-in-Time

De ontwikkelervaring met Dart is echt ongekend. Als je een app ontwikkelt, kan je deze compileren en laten draaien op een simulator, een device, in een browser of op de machine zelf. Doordat Dart gebruik kan maken van de JIT-methode zie je wijzigingen die je doorvoert in de code, direct terug in de app die je aan het ontwikkelen bent. Je hoeft de app niet opnieuw te compileren. Met uitzondering van een webapp blijft de state waarin de app op dat moment is behouden. Dit voelt een beetje als magie.

Stel je voor: je moet een berekening maken in een boodschappenmandje en er zit een fout in de berekening. Als je dan de code aanpast en deze wijziging opslaat, zie je dit direct terug. Je hoeft de app niet opnieuw te compileren en je hoeft niet opnieuw de boodschappen in het mandje te stoppen. Deze manier van ontwikkelen is zo prettig en verhoogt de snelheid enorm en daarmee ook het plezier in het ontwikkelen van de app.

Ahead-of-Time

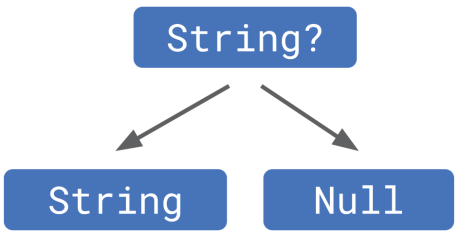
Naast Just-in-Time-compilatie kan Dart ook Ahead-of-Time (AOT) compileren.
De AOT-methode zal de Dart-code compileren naar binaire code voor het platform waarop de app gaat draaien. Het voordeel van AOT is dat de opstarttijden en performance van de app gelijk zullen zijn aan die van een native app, die ontwikkeld is voor het platform.

Sound null safe

Sinds Flutter 2.0 kunnen apps gebouwd worden die ‘sound null safe’ zijn. Veel talen hebben al ondersteuning voor null safety, zoals Kotlin, Swift en Rust, maar
Dart is samen met Swift de enige taal die sound null safe zijn.

Wanneer we null safety aanzetten in Dart, dan zijn de types standaard niet null. Verwachten we ergens een string als parameter?

Dan moet daar ook altijd een string in staan en kan er geen null in staan. Dit wordt afgedwongen door de Dart-compiler. Het voordeel hiervan is dat we geen onnodige null-checks hoeven te schrijven in de code. Dit voorkomt veel null-pointer-exceptions. Het kan natuurlijk dat we toch een string of null willen terugkrijgen als parameter. Dan kunnen we dit eenvoudig aangeven aan de compiler door achter het type een vraagteken te plaatsen(optional).



Op een optional kunnen we een null-check doen, voordat we de inhoud gaan bekijken. Door 'late' voor de type-aanduiding te zetten, leggen we aan de compiler uit dat de variabele na creatie van de klasse zal worden ingevuld.

Door het volledige project null safe te maken, is de compiler nog beter in staat de code te optimaliseren. Dit is het 'sound'-gedeelte van sound null safe. De optimalisatie die je krijgt als je volledige project null safe is. Dit resulteert in minder bugs kleinere, binary's en snellere executie.

Toekomst van Dart

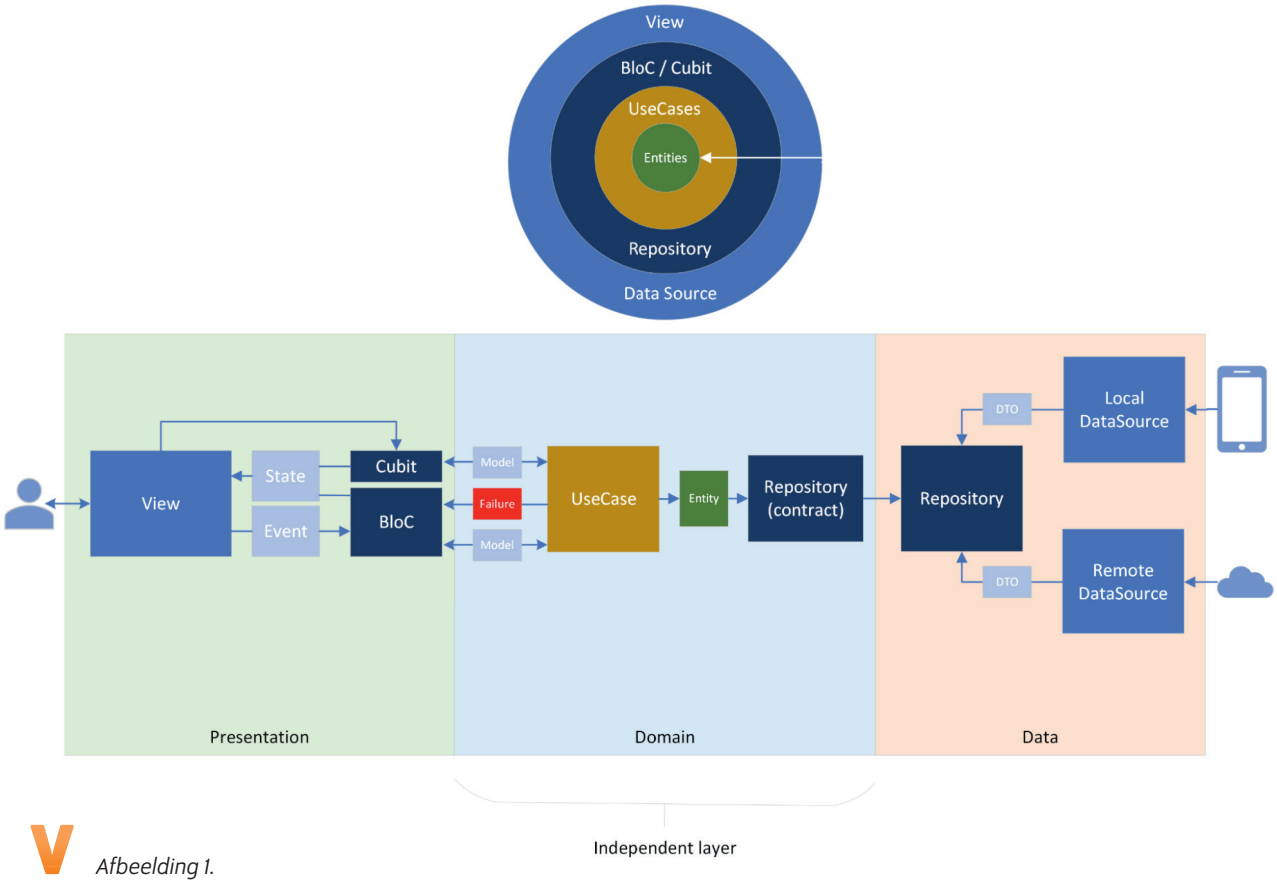
Dart is vooral een nieuwe taal die kijkt wat andere talen goed doen en dit vervolgens overneemt. Een voorbeeld hiervan is sound null safety. Daarnaast krijg je als gebruiker van Dart regelmatig een vragenlijst. Hierin kan je aangeven wat je graag in de volgende versie van Dart verbeterd wil zien. Ik denk dat Dart wel kan uitgroeien tot één van de mooiste talen.

{ HOE WERKT FLUTTER? }

Als mensen aan mij vragen: "wat is Flutter?" Dan stel ik altijd de vraag:
"Ken je Unity of Unreal?" Dit zijn game-engines. Met een game-engine hoef je als ontwikkelaar niet meer na te denken over physics, collision detection en andere zaken. Flutter is hetzelfde alleen niet voor games, maar voor apps.

Flutter maakt geen gebruik van de componenten die het platform levert. Dus niet een invoerveld of tableview van Android of iOS, maar Flutter rendert alles op het scherm zelf met minimaal 60 frames per seconde.

Het mooie aan Flutter is dat je geen HTML, CSS of JavaScript hoeft te leren om een interface te bouwen. Dit doe je allemaal in één taal: Dart.
Met Flutter heb je volledige controle over elke pixel op het scherm. Het voordeel is ook dat je een interface bouwt die op alle plat-



Afbeelding 1.

formen gelijk is. Vaak zie je bedrijven veel moeite stoppen in het bouwen van een interface die op alle platformen gelijk is. Door de opzet van Flutter en het zelf renderen van de componenten heb je hier geen last van. Wil je afwijken door bijvoorbeeld platform-specifieke componenten te renderen? Dan is hier de mogelijkheid voor. Denk hierbij aan iOS, Windows, macOS of andere platform-specifieke widgets.

Wat geweldig is aan Flutter is dat je in bestaande projecten Flutter ook kan gebruiken voor het bouwen van nieuwe features. Dit geeft ontwikkelaars de kans om Flutter uit te proberen in bestaande apps en een app niet volledig opnieuw te hoeven schrijven. Dit zien we al bij veel bedrijven gebeuren, zoals bijvoorbeeld bij Funda die grote delen van hun native app aan het vervangen is met features gebouwd in Flutter.

{ FLUTTER 3.0 RELEASE OP DE GOOGLE I/O }

Op 11 en 12 mei is de Google I/O-conferentie weer geweest en daarin werd voor het eerst in de developer keynote ruimte gemaakt voor Flutter. Het lijkt daarmee dan ook dat Flutter binnen Google een steeds grotere rol gaat spelen. Op dit moment zijn er 500.000 apps gebouwd met Flutter en tijdens de conferentie werd Flutter 3.0 gereleased.

Met Flutter 3.0 zijn nu ook Linux en macOS stable en kunnen we dus apps bouwen voor Android, iOS, Web, Windows, macOS en Linux. En dat komt mede door de renderengine die Flutter gebruikt.

Doordat Flutter steeds meer platformen kan ondersteunen, gaat de complexiteit van het bouwen van apps met Flutter wel omhoog. Dit is een keuze die het Flutter-team heeft gemaakt. Een voorbeeld hiervan is de routing. Doordat Flutter meer platformen is gaan ondersteunen, moest de routing ook aangepast worden. Je kan nu in Flutter op twee manieren navigeren. Navigator 1.0 is een imperative API, waarmee je pagina's op een navigatie-stack plaatst. Navigator 2.0 is een declarative API, waarmee er volledige controle is over de navigatie-stack.

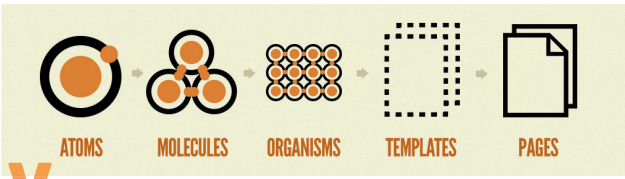
Google liet superlist.com zien (een voorbeeldapp). Deze app ondersteunt alle platformen. Als app-developer is het altijd leuk om aan een game te werken. Met Flutter 3.0 kan je gebruik maken van de hardware-accelerated-graphics-ondersteuning van Flutter. Daarnaast kan je gebruik maken van de opensource game-engine Flame. Samen met Flutter 3.0 is de Casual Games Toolkit aangekondigd. In deze toolkit zitten templates en 'best practices', zoals in-app purchase, credits, ads en cloudservices, zoals gamecenter integratie voor het bijhouden van scores.
Hiermee kan je als app-developer focussen op de gameplay en de soms wel saaie en complexe delen van een game hergebruiken vanuit het template.

Een voorbeeld van de Casual Games Toolkit is de Flutter pinball demo. Flutter wordt steeds groter binnen Google. Dit wordt mede bevestigd door het feit dat Firebase volledige ondersteuning biedt voor Flutter.

{ HOE START JE MET FLUTTER? }

Ben je enthousiast, maar weet je nog niet hoe je met Flutter moet beginnen? Met Flutter ben je volledig vrij in het kiezen van je applicatie-architectuur. Flutter kan in het begin daarom lastig zijn. Er is niet een duidelijk voorbeeld vanuit Google hoe je een app kan bouwen. Binnen Ordina hebben we een opleidingstraject, die elke toekomstige Flutter ontwikkelaar moet doorlopen.

- Udemy Flutter-cursus van Maximillian Schwarzmüller.
- Clean Architecture en TDD van Reso Coder.
- Bloc from Zero to Hero van Flutterly.
- Atomic Design.



Atomic design principe.

Zoals al eerder gezegd, ben je vrij in hoe je een app ontwerpt in Flutter. Dit kan starten met Flutter wat ingewikkelder maken. Hier een voorbeeld om je te laten zien wat het probleem hiermee is, als je nog niet veel Flutter-ervaring hebt. Je kunt in Flutter op verschillende manieren de state van de app managen. Hiervoor zijn verschillende packages beschikbaar: Provider, Riverpod, setState, InheritedWidget, Redux, Fish-Redux, Bloc / RX, GetIt, MobX, Flutter Commands, Binder, GetX, states_rebuilder, Tripple Pattern. Er is genoeg keuze, maar wat is nu de beste? Voor veel grote projecten wordt bloc als state-management-oplossing gebruikt.

De architectuur die wij veel gebruiken binnen projecten en die echt een aanrader is zodra je een groot Flutter-project start, is 'The Clean Architecture' van Robert C. Martin (Uncle Bob). Deze architectuur is vertaald naar een Flutter-opzet en deze hebben wij bij Ordina verder geoptimaliseerd in de architectuur, zoals te zien in Afbeelding 1. Het sterke van deze architectuur is de scheiding tussen de datasources, businesslogica en presentatielaag en de mogelijkheid tot het schrijven van unittesten voor elke laag.

Naast deze architectuur is het sterk aan te raden om structuur aan te brengen in de opbouw van de presentatielaag. Dit kan gedaan worden met het 'atomic design' principe.
Door widgets te structureren volgens het atomic design principe kunnen we widgets op meerdere plaatsen hergebruiken en blijft de presentatielogica overzichtelijk en wordt hergebruik van widgets bevordert.

{ COMMUNITY }

Natuurlijk is het super leuk om samen met andere te sparren over Flutter. Dit kan heel goed tijdens meet-ups van de Flutter Netherlands Community. Dit is een snel groeiende community

Afbeelding 2.

van Flutter-enthousiastelingen, die in coronatijd is opgericht en inmiddels een kleine 750 leden telt. Er worden regelmatig meet-ups georganiseerd door het hele land, waarbij bedrijven een kijkje in de keuken geven hoe ze Flutter binnen hun organisatie gebruiken. Daarnaast is er meestal een onderdeel 'in de spotlight', waarbij ontwikkelaars door middel van een 'lightning talk' in vijf minuten wat over hun project kunnen vertellen. Dit is altijd erg interessant en inspirerend. Het is een levendige community waar veel gebeurt en iedereen bereid is elkaar verder te helpen.

{ TOT SLOT }

Je ziet steeds meer bedrijven de overstap maken naar Flutter voor hun cross-platform-oplossing, of native development vervangen door Flutter. Het unieke aan Flutter is dat het een andere opzet heeft als alle andere cross-platform-oplossingen, namelijk de render engine.

Je bent vrij in de architectuurkeuze binnen Flutter. Dit kan voor nieuwe ontwikkelaars best lastig zijn om te beginnen met Flutter. Concurrenten (zoals Xamarin) zijn erg onder de indruk van Flutter. Dat blijkt wel uit de tweet van Miguel de Icaza (co-founder van Xamarin), zie Afbeelding 2. <

{ REFERENTIES }

1 https://flutter.dev/
2 https://resocoder.com/
3 https://atomicdesign.bradfrost.com/
4 https://pinball.flutter.dev/

V

COLUMN

People who plan better, procrastinate less

Maja Reißner *is software developer bij SIDN.*

I procrastinate. It can be annoying and exhausting not to do what I actually want to or should do. Usually when breaking procrastination I'll be making a plan, not too complicated but a real plan on how to do whatever I need to do. Sometimes it's a complete plan and sometimes I just need to figure out how to begin. I thought that typing out a plan was my personal coping mechanism to deal with procrastination but recently I've read articles that suggest that this is actually a feature, not a bug [1].

Procrastination is often thought to be due to fear of failure, fear of not being good enough and fear of making mistakes. However, evolutionary science linked procrastination to goal-management [2]. As a consequence, this could mean that procrastinating is unconsciously not being okay with the plan yet, indicating that you should plan better.

The reasoning behind this is as follows: complex planning is a skill that apparently differentiates us from other species (such as Homo Neanderthalensis). It means we can make a plan, think about possible consequences and evaluate our plan, resulting in good decisions and better survival. Procrastination can be thought of unconsciously evaluating a plan and refusing to go forward with a plan we do not feel comfortable with yet. Therefore, it prevents us from doing things we haven't properly planned yet.

Let's think bigger here for a moment: if you procrastinate everything, maybe you should re-evaluate everything. Maybe you're just in very general terms not fine with what you're currently doing and how this will improve and secure your future.

Though this is a possible reason, there's an extra check you may want to do before you throw your life away: delaying steps for long-term goals (such as studying, building up a future, losing weight) is not necessarily procrastination. It may just as well be that you only have trouble with postponing gratifications. A simple example would be that it is more satisfying to have a pizza now and watch TV than to go for a run and eat healthy until you are in good shape again.

If you know you have a gratification-delay problem, you can make a plan that introduces small gratifications throughout the path towards the long-term goal. In that sense, overcoming gratification-delay problems is similar to overcoming procrastination: You need to make a decent plan.

Interestingly enough I did not procrastinate writing this. The thought was simple enough, the resources easily available and the specifications clear enough. And who knows, maybe writing this is actually procrastinating something else that I haven't worked out yet. So what are you procrastinating when reading this article? ;)

I actually believe that this is an unfriendly and not very helpful question. You read this article because right now YOU made it your top priority. That's okay. And if there's something else in your life that should have more priority (by your own judgment), what would now be the best next step in your plan to get that thing done? That's what you need to focus on. :)

References

[1] https://www.calnewport.com/blog/2011/07/10/the-procrastinating-caveman-what-human-evolution-teaches-us-about-why-we-put-off-work-and-how-to-stop/

[2] https://journals.sagepub.com/doi/10.1177/0956797614526260

BellSoft Released Alpaquita Cloud Native Platform for Java Developers

New offering includes the only optimized Java Linux distribution, delivers up to 50% lower RAM consumption

BellSoft, the creator of Liberica JDK, a progressive Java runtime for the most complete Java experience and a leading OpenJDK contributor, announced on September 27th the release of the BellSoft Alpaquita Cloud Native Platform and Alpaquita Linux. The offerings are designed to meet the growing demand for more efficient, highly secure and supported Java software that performs better and reduces cloud computing costs.

"Our mission at BellSoft is to deliver the most complete Java experience to developers and IT managers. We are passionately committed to the modernization of Java for a cloud-native world, and we believe the best way to achieve this is through open source collaboration, and it's why we are one of the top contributors to OpenJDK," said BellSoft CEO and co-founder Alex Belokrylov. "With today's introduction of the Alpaquita Cloud Native Platform and Alpaquita Linux we are extending our leadership in the Java community to address pain points around efficiency and cost, so developers can focus on creating innovative, scalable, cloud-native applications."

The Alpaquita Cloud Native Platform is a compilation of a brand new Alpaquita Linux with other BellSoft tools that empower container-based software development. Alpaquita Linux is the only Linux distribution tuned for a Java runtime. Based on Alpine Linux, it delivers better performance, security, smaller size, and flexibility. It is also the smallest microcontainer on the market, consuming less energy and memory space, thus reducing cloud computing costs.

Available in several packages for developers and development teams, the Alpaquita Cloud Native Platform includes the new Alpaquita Linux distribution, Liberica Lite, a version of Liberica JDK specifically optimized for cloud environments, and Liberica Native Image Kit. It delivers a new level of efficiency and performance for cloud-based applications by providing a secure, performant, and convenient Java runtime and deployment environment.

Alpaquita Cloud Native Platform is purpose built for large enterprises looking to improve sustainable development practices, and decrease costs and energy consumption. It is ideal for extensive Java workloads and optimizes cloud and hardware resources to deliver improved ROI on cloud investments. Features include:

- Optimal RAM consumption for Java deployment;
- Java and GraalVM support to run in any infrastructure and on any cloud;
- Top-notch security enhanced with modern features and CVE-tracking;
- Support for Linux and Java Runtime with SLA;
- Minimal static footprint for containerized applications.

About BellSoft

BellSoft delivers the most complete Java experience. Our Java platform, Liberica JDK, provides a unified Java experience across the organization, delivering a more secure, reliable, and cost-effective approach to application development. Our flexible approach empowers developers and architects to build, run, and manage Java applications in any platform and any environment. BellSoft is one of the largest contributors to OpenJDK and is the runtime of choice for VMware, Pivotal, Spring, JetBrains, and more. BellSoft serves millions of use cases and global brands across every industry, helping companies build for the future every day. For more information, visit www.bellsoft.com.

DEVITJOBS.NL

Java Programmeur Electronic Trading

Salarisindicatie:
40.000 - 50.000 EUR

Locatie:
Rotterdam

Technologieën:
Java, Java EE, Kafka, React, Gradle, MySQL, MariaDB, Docker, Kubernetes, Prometheus, Grafana, GitLab

transtrend

Senior Java Developer

Salarisindicatie:
55.000 - 85.000 EUR

Locatie:
Apeldoorn

Technologieën:
Java, JavaScript, Azure

profit4cloud

Java / API Developer Wealth

Salarisindicatie:
60.000 - 90.000 EUR

Locatie:
Amsterdam

Technologieën:
API, Azure, Backend, Camunda, CI/CD, Cloud, DevOps, H2, Hibernate, JPA, JWT, Java, Mobile, Nexus, REST

ABN·AMRO

Aspiring Expert

Salarisindicatie:
35.000 - 65.000 EUR

Locatie:
Breda

Technologieën:
Java, Spring, Spring Boot, React, React Native, Scala, TypeScript, MongoDB, Maven, Jenkins, Git, GitHub, GitLab, Azure, AWS

codecentric

Find all these vacancies and more

at Devitjobs.nl

CANARY RELEASES WITH INFRASTRUCTURE AS CODE IN JAVA

Many teams building modern Java applications often implement it with modular architectural patterns, serverless computing, and agile development processes. When building serverless web applications, it is common to use AWS Lambda functions as the compute layer for business logic. In this article, you'll create and automatically rollout a Java application with canary deployment strategy using AWS Serverless and Infrastructure as Code (IaC) in Java.

Fernanda Machado works as a Developer Specialist Architect at the AWS Developer Acceleration (DevAx) team.

With incremental releases, canary deployment is a practice to rollout software into production to a small set of users before replacing it entirely. Computing serverless technologies [1], such as AWS Lambda and AWS Fargate [2,3], allow you to focus on writing business logic, removing the complexity of managing the underlying infrastructure. AWS Cloud Development Kit (AWS CDK) empowers you to reuse your Java skills to deploy cloud technology using IaC in Java.

AWS Lambda versions allow you to manage the deployment of your function. For example, you can publish a new AWS Lambda version for beta testing without affecting users of the stable production version. First, you need to point an AWS Lambda alias to two function versions. Second, configure the percentage of traffic that is routed to each version. This allows you to gradually shift user traffic, reduce the risk and limit the blast radius of the new function version.

AWS CodeDeploy allows you to automatically manage the rollout of new function versions using AWS Lambda traffic shifting capabilities. AWS CodeDeploy lets you automate pre-deployment tests that a function must pass before it begins taking traffic. You can also set alarms that automatically trigger rollbacks in the event of errors.

To easily build these deployment patterns into your serverless applications, you can include AWS CodeDeploy within your AWS Serverless Application Model (AWS SAM) templates or AWS CDK with AWS SAM Construct Library [4]. AWS SAM and AWS CDK are both open source frameworks for building applications using IaC. AWS SAM is template-based using JSON or YAML, while the AWS CDK uses languages such as Python and Java.

{ IMPLEMENTATION }

To start, we'll setup AWS CDK on your workstation. Next, create the AWS CDK skeleton application. Import required libraries and add business logic to our application. Finally, we'll deploy our application and implement new changes to test the canary deployment.

{ PREREQUISITES }

Before starting this guide, you will need:

- Node.js 16 or later.
- AWS CDK Toolkit 2.37 or later.
- Java Development Kit (JDK) 11 or later.
- Apache Maven 3.5 or later.
- AWS CLI 2.
- Unix-based systems or Windows Subsystem for Linux (WSL).

If you don't have AWS CDK installed and your AWS account bootstrapped, please follow the 'Getting Started with AWS CDK' guide [5]. The bootstrap gives AWS CDK permissions to deploy AWS resources on your behalf and upload your source code. Once you have the AWS CDK toolkit installed. The AWS Account

and region combination bootstrapped, you can start writing and deploying some infrastructure.

{ CREATE THE CDK DEMO APPLICATION }

We will now create a blank project for Java with CDK:

```
mkdir aws-cdk-canary-lambda-demo
cd aws-cdk-canary-lambda-demo
cdk init app --language java
```

Bash.

This will be the initial project structure:

```
├── README.md
├── cdk.json
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── com
    │           └── myorg
    │               ├── AwsCdkCanaryLambdaDemoApp.java
    │               └── AwsCdkCanaryLambdaDemoStack.java
    └── test
        └── java
            └── com
                └── myorg
                    └── AwsCdkCanaryLambdaDemoTest.java
```

Structure 1: bash.

Here are some of the important files and what they are used for:
AwsCdkCanaryLambdaDemoApp.java – This is the entry point to your AWS CDK demo application. It creates the stack resource defined in the file src/main/java/com/myorg/ AwsCdkCanaryLambdaDemoStack.java. A resource stack is a set of AWS resources, which CDK will deploy into a specific account and region.
AwsCdkCanaryLambdaDemoStack.java – This contains your application stack, resources and properties definition.
cdk.json – This holds the AWS CDK toolkit settings and parameters to run your project.
pom.xml – This defines your project dependencies. The AWS CDK includes a collection of constructs called the AWS Construct Library, containing constructs for every AWS service. The application skeleton generated by the AWS CDK Toolkit is a Maven project and includes the AWS Construct Library dependencies.

{ CREATE THE AWS LAMBDA FUNCTION RESOURCE }

Let's create your function resource. Go to the file src/main/java/com/myorg/ AwsCdkCanaryLambdaDemoStack.java and open up your stack definition. When first opening up the file you should see something like this:

```
public class AwsCdkCanaryLambdaDemoStack extends
Stack {
    public AwsCdkCanaryLambdaDemoStack (final
    Construct scope, final String id) {
        this(scope, id, null);
    }

    public AwsCdkCanaryLambdaDemoStack (final
    Construct scope, final String id, final StackProps
props) {
        super(scope, id, props);
        // here you can define you AWS resources
    }
}
```

Java.

If you would run AWS CDK now, no resources would be created as we don't have any defined yet. Let's define the AWS Lambda function configuration and source code:

```
Function function = Function.Builder.create(this,
"CdkLambdaFunction")
        .code(Code.fromAsset("./lambda-application/
target/lambda-application-0.1.jar"))
        .handler("com.myorg.
AwsCdkDemoLambdaHandler")
        .functionName("myFunction")
        .runtime(Runtime.JAVA_11).memorySize(256)
        .timeout(Duration.minutes(5)).build();
```

Java.

The next step is to add a tag to the current state of our function using AWS Lambda alias and version to better manage the deployment:

```
Version version = function.getCurrentVersion();
Alias alias = Alias.Builder.create(this, "alias")
        .aliasName("prod")
        .version(version)
        .build();
```

Java.

Now, we'll define a custom application deployment strategy to be used by AWS CodeDeploy. Five percent of our application traffic is immediately shifted to our new version. After two minutes, all traffic is shifted to the new version:

```
CustomLambdaDeploymentConfig config =
CustomLambdaDeploymentConfig.Builder.create(this,
"CdkLambdaCustomConfig")
```

```
        .type(CustomLambdaDeploymentConfigType.
        CANARY)
        .interval(Duration.minutes(2))
        .percentage(5)
        .build();
```

Java.

The next step is to define our function deployment automatic rollout, by associating the Lambda alias function and the deployment strategy:

```
LambdaDeploymentGroup deploymentGroup =
LambdaDeploymentGroup.Builder.create(this,
“CdkLambdaCanaryDeployment”)
        .alias(alias)
        .deploymentConfig(config)
        .build();
```

Java.

{ CREATE THE LAMBDA FUNCTION APPLICATION }

Using your favourite IDE, create the lambda-application Maven module within the root project directory. Within the new module, create a new class src/main/java/com/myorg/AwsCdkDemoLambdaHandler.java. Again, within the project directory, create the lambda-resource-stack Maven module. In this new module, replace the src and test directories with the ones from the root project directory. The final project structure looks like Structure 2.

We need to add the required libraries to use AWS Lambda with Java. Within the new lambda-application module, go to the pom.xml file and add the following dependencies:

```
<dependency>
   <groupId>com.amazonaws</groupId>
   <artifactId>aws-lambda-java-core</artifactId>
   <version>1.2.1</version>
</dependency>
```

Maven.

Let’s add the business logic to the function. It needs to generate a random number between 1 and 5, and evaluate if it’s even or odd. Go to the file src/main/java/com/myorg/AwsCdkDemoLambdaHandler.java and add the following handler method:

```
public class AwsCdkDemoLambdaHandler implements
RequestHandler<Object, String> {

    Random random = new Random();

    @Override
    public String handleRequest(Object o, Context
    context) {
        System.out.println(“Alpha version in
        production”);
        int number = random.nextInt(6);
        if(number % 2 == 0)
            System.out.println(number + “ is
            even”);
        else
            System.out.println(number + “ is odd”);
        return “success”;
    }
}
```

Java.

The CDK Toolkit needs to know how to execute your AWS CDK app. Let’s update the app key within the cdk.json file by adding the following value (continued next page):

```
├── README.md
├── cdk.json
├── cdk.out
├── lambda-application
│   ├── pom.xml
│   └── src
│       ├── main
│       │   ├── java
│       │   │   └── com
│       │   │       └── myorg
│       │   │           └── AwsCdkDemoLambdaHandler.java
│       │   └── resources
│       └── test
│           └── java
├── lambda-resource-stack
│   ├── pom.xml
│   └── src
│       ├── main
│       │   ├── java
│       │   │   └── com
│       │   │       └── myorg
│       │   │           ├── AwsCdkCanaryLambdaDemoApp.java
│       │   │           └── AwsCdkCanaryLambdaDemoStack.java
│       │   └── resources
│       └── test
│           └── java
│               └── com
│                   └── myorg
│                       └── AwsCdkCanaryLambdaDemoTest.java
└── pom.xml
```

Structure 2: bash.

```
"app": "mvn exec:java -pl lambda-resource-
stack -Dexec.mainClass=com.myorg.
AwsCdkCanaryLambdaDemoApp",
```

JSON.

{ DEPLOY THE STACK }

Time to test our deployment. To see if your code is valid, you can run `mvn install` which will download the required dependencies to compile the Java code. Don't forget to import the used classes in both modules. If that is successful, you can run the deploy command:

```
cdk deploy
```

Bash.

If your code has security implications, you'll see a summary of these and need to confirm them before deployment proceeds. This isn't the case in our stack. Enter 'y' to continue with the deployment and create the resources. This will create an AWS CloudFormation change set to deploy this change. AWS CDK manages all of this for you, along with uploading the template file to an Amazon S3 bucket and using CloudFormation to run it. This is why we had to run bootstrap before.

After a few minutes, your new AWS Lambda should be ready. Every time you change the business logic of your AWS Lambda function, re-deploy it by running `mvn package` and `cdk deploy` will generate a new AWS Lambda version.

```
6:50:47 PM | UPDATE_IN_PROGRESS   |
AWS::Lambda::Alias                | alias
```

Bash.

You can continually see the Lambda invocation output using the AWS CLI:

```
while true; do aws lambda invoke --function-name
myFunction:prod out --log-type Tail \
--query 'LogResult' --output text |  base64 -d;
done
```

Bash.

The output should be something like this:

```
START RequestId: af260d47-f8ac-4046-b3e5-
1cc3921158b3 Version: 1
Alpha version in production
3 is odd
```

Bash.

Now, business requirements have changed and we need to change the function logic to check numbers until 10. Update the code accordingly. Deploy it again, by running `mvn package` and `cdk deploy`. The deployment will shift 95% of traffic from the current version to the replacement version at approximately two minutes after the deployment started [6]. The lambda invocation output should include the new range:

```
START RequestId: af260d47-f8ac-4046-b3e5-
1cc3921158b3 Version: 1
Beta version in production
10 is odd
```

Bash.

{ CLEAN UP RESOURCES (OPTIONAL) }

If you wish to remove your newly created resources, you can run the `cdk destroy` command and it will remove all the resources via the CloudFormation stack it created earlier.

```
cdk destroy
```

Bash.

You should receive the below prompt. After pressing 'y' and 'Enter', it will start removing all the infrastructure and provide updates. Once completed, you will see the following:

```
Are you sure you want to delete:
AwsCdkCanaryLambdaDemoStack (y/n)? y
 ✅  AwsCdkCanaryLambdaDemoStack: destroyed
```

Bash.

Congratulations! In this article, you learned how to create and automatically rollout a Java application with canary deployment strategy using AWS Serverless and IaC in Java. If you would like to learn the common serverless applications scenarios and best practices, I encourage you to go through the AWS Well-Architected Serverless Lens [7]. To follow the latest blogs, videos, and training for AWS Serverless, you can check out the Serverless Land site [8].

{ REFERENCES }

1 https://aws.amazon.com/serverless/
2 https://docs.aws.amazon.com/cdk/v2/guide/ecs_example.html
3 https://aws.amazon.com/modern-apps/faqs/
4 https://aws.amazon.com/cdk/features/
5 https://docs.aws.amazon.com/cdk/v2/guide/getting_started.html
6 https://docs.aws.amazon.com/codedeploy/latest/userguide/deployments-view-details.html#deployments-view-details-console
7 https://serverlessland.com/
8 https://docs.aws.amazon.com/wellarchitected/latest/serverless-applications-lens/

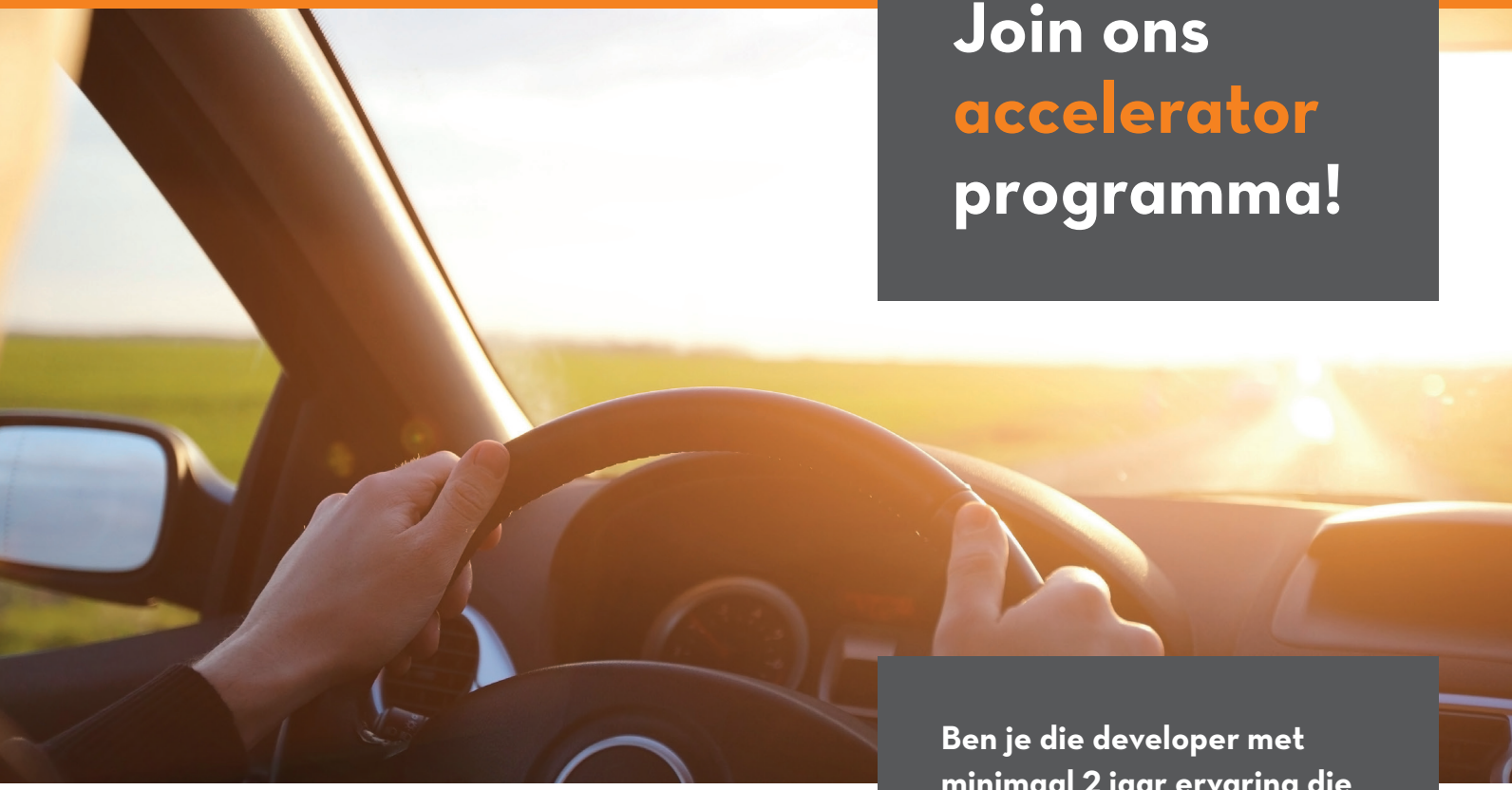
Vandaag versnel ik het incheckproces, zo zijn reizigers morgen nog sneller thuis.

Shirani, Developer

Lees meer op
werkenbijns.nl

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

Join ons accelerator programma!



ORDINA
Ahead of change

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

Ben je die developer met minimaal 2 jaar ervaring die zijn developer kennis naar een next level wil brengen?

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

Kijk voor meer informatie op:
www.ordina.nl/werkenbij

Cloud Days • Security Days • DevOps Days • Testing Days • Frontend Days • Framework Days