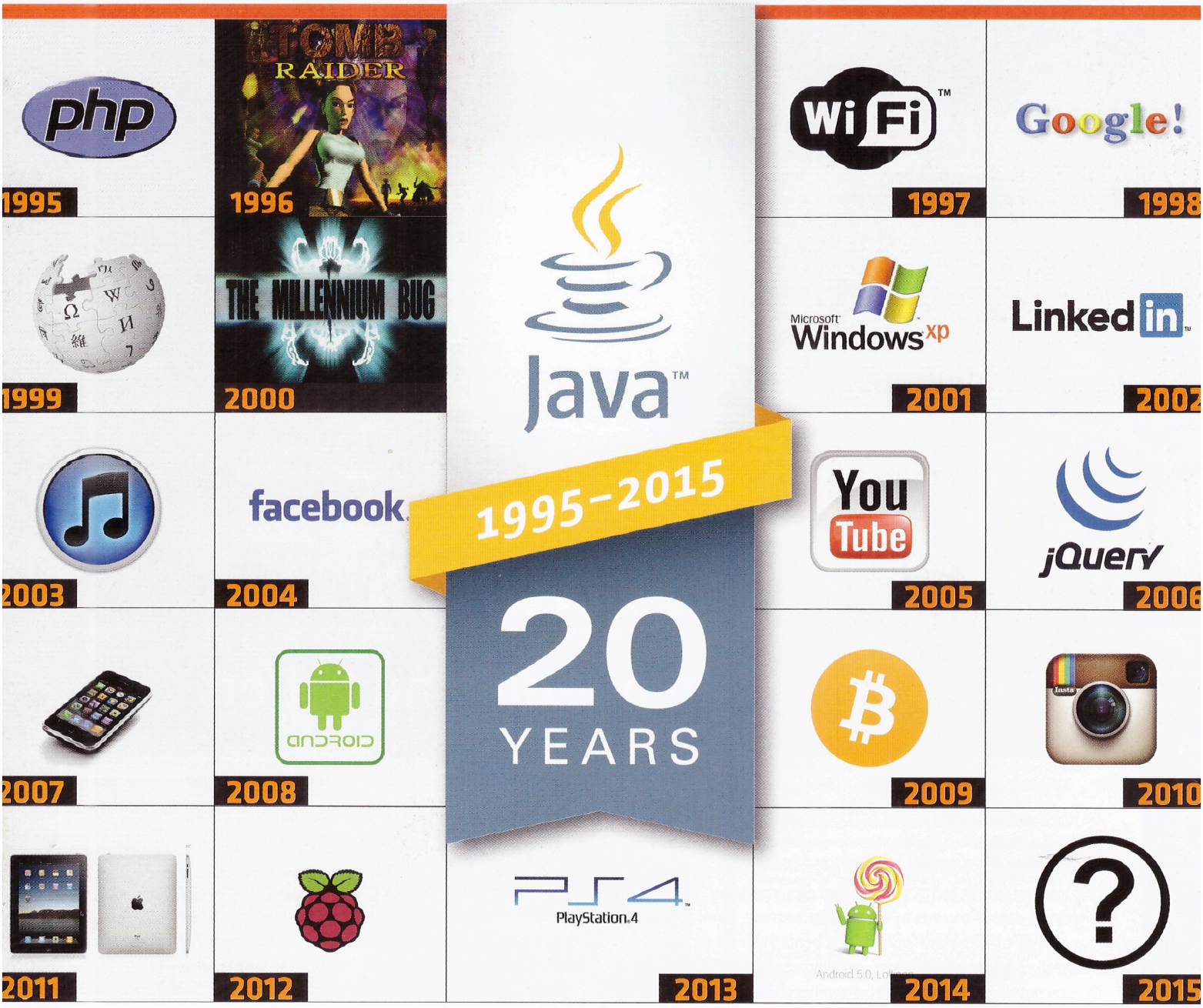
Nummer 3 2015

JAVA MAGAZINE

.nl jug

Onafhankelijk tijdschrift voor de Java-professional



Features

Grip op je buildproces met Gradle

Terugblik IoT Developers Day 2015

Smarthome expert Kai Kreuzer

Grip op je buildproces met Gradle

En hoe je een migratie vanaf Ant of Maven aanpakt in een enterprise-omgeving

"Wat is het verschil tussen Ant en Maven? De maker van Ant heeft zijn excuses aangeboden." Dit grapje, waarmee Mark Reinhold de lachers op zijn hand kreeg op Devoxx, geeft treffend aan hoe impopulair buildscripts zijn onder ontwikkelaars. Meestal zijn die liever met de inhoud bezig dan met de build-infrastructuur, hoe belangrijk die ook is. Bovendien hebben industriestandaarden als Ant en Maven elk hun eigen problemen en maken ze verbose XML onvermijdelijk. Voor ons project bij de Nederlandse Spoorwegen, dat leunt op ruim 25.000 regels Ant-scripting, onderzochten we daarom de mogelijkheden om Gradle in te zetten als vervangend buildsysteem. En wat bleek? We kwamen op onze bestemming aan, maar wel met enige vertraging. Dit artikel deelt onze ervaringen.

Hanno Embregts is als software engineer werkzaam bij Info Support voor de Nederlandse Spoorwegen. Hij houdt zich graag bezig met Java (EE), Continuous Delivery, Behavior-Driven Development en agile werken.

Jan Ouwens was als senior software engineer bij Sogeti werkzaam voor de Nederlandse Spoorwegen. Hij houdt zich graag bezig met Java, Scala, softwarekwaliteit en zijn open source-tool EqualsVerifier.

Ant...

Bij de NS vinden we *continuous delivery* belangrijk, dus de Ant-scripting doet aanzienlijk meer dan alleen compileren, testen en packagen. Het ondersteunt het volledige software-ontwikkelproces van IDE tot productie-omgeving. Het wordt onderhouden door ontwikkelaars met de speciale rol van 'integrator'. Zij voeren verbeteringen door en coachen andere teamleden in het nemen van verantwoordelijkheid voor een optimaal presterende buildstraat.

Juist omdat we zoveel vragen van onze buildscripting, zien we graag dat onderhoud snel en eenvoudig uitgevoerd kan worden. Ant is flexibel, maar onze scripting heeft al in meerdere projecten dienst gedaan en is in tien jaar tijd uitgegroeid tot een complex geheel. Nieuwe teamleden vinden het moeilijk om structuur te ontdekken en te achterhalen welke targets ze moeten gebruiken. Bovendien zijn de verschillende onderdelen van het systeem in de buildscripting nauw met elkaar verbonden. Mocht je een component los willen opleveren, dan kun je je voorbereiden op een complexe klus. In de wetenschap dat het systeem dat we bouwen 20 jaar mee moet kunnen, en waaraan door 35 ontwikkelaars wordt gewerkt aan een *codebase* van bijna een miljoen regels, besloten we dat onze buildscripting aan vervanging toe was.

Maven?

Natuurlijk dachten we allereerst aan Maven. Door Mavens structuur en het *lifecycle-concept* is een bestaand buildscript voor ontwikkelaars snel te doorgronden. Maven heeft bovendien goede ondersteuning voor gemodulariseerde projecten.

Maven levert het grootste voordeel op als je project kan aansluiten bij de voorgestelde structuur. Echter bleken onze Ant-targets daar ongeschikt voor. We konden de situatie alleen werkend krijgen, als we om Mavens conventies heen zouden werken met allerlei *custom* XML-configuratie. Dit omvangrijke extra werk deed ons besluiten om Maven links te laten liggen.

Gradle!

Gradle[2] is een modern buildsysteem, dat de flexibiliteit van Ant combineert met de structuur van Maven. Buildscripts worden opgesteld in een op Groovy gebaseerde Domain Specific Language (DSL), waarin je eenvoudig je eigen buildtaken kunt definiëren.

Die buildtaken hoef je overigens lang niet altijd zelf te bedenken. Gradle levert een vaste structuur door *plugins* aan te bieden. Door een plugin in je buildscript te activeren, komen verschillende voorgedefinieerde taken beschikbaar.

Plugins voeren een standaardprogramma uit, dat zonder extra configuratie de Maven-conventies volgt. Maar afwijken van deze conventies is eenvoudig en vereist weinig regels code. Zo kun je *custom* buildtaken bijvoorbeeld een plek geven in de reguliere buildvolgorde.

In een eerder artikel in Java Magazine heeft Hubert Klein Ikkink de kracht van Gradle al uitgebreid toegelicht[3]. Wij richten ons in de rest van dit artikel op de voordelen van Gradle voor de NS, de uitgevoerde migratie en de uitdagingen die we daarbij tegenkwamen.

Keuze voor Gradle

We kozen voor Gradle als vervangend buildsysteem vanwege Gradles focus op het aanbrengen van structuur, terwijl er ruimte blijft voor flexibiliteit. Flexibiliteit was er genoeg in onze Ant-implementatie en dat moest zo blijven. Structuur is wat er miste en wat toegevoegd moest worden.

Maar er waren nog meer redenen om Gradle te kiezen. Op de buildserver gedroeg ons project zich tamelijk monolithisch. Een wijziging in *source control* triggerde een build van het hele project, terwijl er misschien maar één kleine wijziging was gedaan. We zouden de buildtijd flink kunnen verkorten door het project op te splitsen in verschillende functionele componenten. Het kwam ons daarom goed uit dat Gradle bekend staat om de goede ondersteuning voor *multi-project-builds*[4]. Gradles ondersteuning voor incrementele builds kon ook verder bijdragen aan significante performanceverbeteringen.

Migratiestrategie

We begonnen de migratie bij een klein, afgezonderd onderdeel zonder afhankelijkheden, om daar vervolgens steeds een onderdeel aan toe te voegen. Na elke stap zouden we vervolgens verifiëren dat de nieuwe scripting hetzelfde resultaat opleverde als voorheen.
Verdere uitgangspunten in onze aanpak waren:

- Vervang niet direct alle Ant-scripting, maar alleen het deel dat ontwikkelaars dagelijks gebruiken;
- Voer de migratie uit naast de reguliere ontwikkelwerkzaamheden, dus breek geen bestaande scripting;
- Laat de migratie uitvoeren door 'integrator'-ontwikkelaars uit minstens twee teams.

Uitdagingen

Ons eerste component was een generiek onderdeel uit een dienstregelingsapplicatie, dat zoals eerder aangegeven niet afhankelijk was van andere onderdelen. Dit component bestond uit ruim zestig subprojecten, waarvan de helft testproject was en bedoeld om de overige projecten te testen.

Nu konden we Gradle gaan gebruiken. De Java-plugin voor Gradle[5] draaide al snel de eerste succesvolle *compile*-taak. Vervolgens maakten we het component tot een *multi-project*-build door alle relevante submappen toe te voegen als subproject (zie **Listing 1**). We waren erg tevreden over de elegante syntax waarmee we Gradle konden

```
def moduleProjectsPath = "."

new File(getRootDir(), moduleProjectsPath).eachDir(
    { dir ->
        if (dir.name.startsWith('nl.ns.dienstregeling.generiek')) {
            include ":${dir.name}"
        }
    }
)
```

Listing 1: Een settings.gradle-bestand dat subprojecten toevoegt voor alle submappen die beginnen met 'nl.ns.dienstregeling.generiek'.

GRADLE LEVERT EEN VASTE STRUCTUUR DOOR PLUGINS AAN TE BIEDEN

vertellen welke projecten de source code bevatten en welke voor de testcode waren (zie **Listing 2**).

Een soepele start, maar gedurende de migratie kwamen we ook uitdagingen tegen (zij het niet allemaal Gradle-gerelateerd). Drie daarvan zullen we toelichten.

Dependencyvervlechting

Van tevoren wisten we al dat de benodigde *libraries* plus versies in de Ant-scripting op meerdere plekken waren gedefinieerd. In Gradle brachten we deze definities samen op een centrale plek in het buildscript. Subprojecten konden naar deze dependencies refereren en zelf kiezen welke ze nodig hadden.

Helaas kwamen we in het vervolg de JAR's die bij de dependencies hoorden op allerlei plekken in het project in verschillende versies tegen. Tot overmaat van ramp bleek de Ant-scripting deze JAR's te pas en te onpas te verzamelen in tijdelijke 'workspace-directories', om vanuit die context een buildstap uit te voeren. En in Nexus (onze *artifact repository*) was het een oerwoud van verdwaalde artifacts, zonder bijbehorende POM-bestanden en soms zelfs zonder versienummer.

De hoofdoorzaak van dit alles? Onze dependencies waren niet transitief. Die aanpak stamde uit de tijd dat we Ant nog zonder Ivy gebruikten en we onze eigen dependencies moesten beheren.

Om dit op te lossen zijn we de dependencies één voor één afgegaan en hebben we transitiviteit geïntroduceerd. Bovendien kreeg elk artifact het juiste versienummer en stelden we zeker dat Nexus voortaan van elk artifact maar één versie had.

Samenwerking met bestaande scripting

Zoals eerder al aangegeven, was een uitgangspunt om niet direct alle 25.000 regels Ant-scripting te vervangen, maar alleen het deel dat ontwikkelaars dagelijks gebruiken. Dit betrof het deel dat het ontwikkelproces ondersteunde van programmeren tot publiceren naar Nexus. Zo konden we het migratietraject afbakenen en ons richten op het onderdeel waar potentieel de meeste tijdswinst te halen was. Dat nam niet weg dat de Gradle-scripting moest kunnen samenwerken met de Ant-scripting die zou overblijven. Gelukkig biedt Gradle goede ondersteuning voor het aanroepen van Ant-targets[6].

```
configure(javaProjects()) {
    // Pas configuratie toe voor alle Java-projecten.
}

configure(testProjects()) {
    // Pas configuratie toe voor alle testprojecten.
}

def javaProjects() {
    subprojects - testProjects() - rootProject()
}

def testProjects() {
    subprojects.findAll { it.name.endsWith('.test') }
}

def rootProject() {
    [project]
}
```

Listing 2: In onze codebase staan tests in aparte projecten. Gradle biedt een elegante manier om zowel de Java- als de testprojecten apart te configureren.

De uitdaging was om de artifacts die Gradle opleverde exact gelijk te maken aan de artifacts uit Ant. Bestaande Ant-scripting ging namelijk uit van specifieke waarden in de MANIFEST.MF-bestanden en die waren vanuit Gradle blijkbaar gewijzigd of zelfs afwezig. Deze en andere minieme verschillen ontdekten we door de JAR's op regelniveau te vergelijken met ZipDiff[7]. Correcties konden vervolgens snel doorgevoerd worden.

Ook hebben we de resultaten van de unittests vergeleken met dezelfde resultaten uit Ant. Gradle kon op een elegante manier de logging van de JUnit-task voor Ant nabootsen (zie **Listing 3**). Dat stelde ons in staat om de logging van beide varianten in een *diff-viewer* te bekijken en zeker te stellen dat de resultaten identiek waren.

Vanaf dat moment hadden we een goede samenwerking met het overblijfsel van de Ant-scripting.

```
configure(testProjects()) {
    test {
        afterSuite { desc, result ->
                println "    [junit] Running ${desc.className}"
                println "    [junit] Tests run: " +
"${result.testCount - result.skippedTestCount}, Failures: " +
"${result.failedTestCount}, Errors: ${result.exceptions.size}"
        }
    }
}
```

Listing 3: Na het uitvoeren van de tests logt Gradle de resultaten in de stijl van de JUnit-task uit Ant, om zo eventuele verschillen makkelijk te vinden.

Samenwerking met de buildstraat

Wanneer je van buildsysteem verandert, heeft dat gevolgen voor de buildstraat. Buildjobs in Jenkins werden nu via Gradle uitgevoerd. Het uitrekenen van de code coverage ging zoals voorheen met Jacoco, maar werd voortaan door Gradle aangestuurd. En ook voor het gebruik van FindBugs en Sonar waren wijzigingen nodig.

De ondersteuning was gelukkig naar wens. Jenkins biedt een Gradle-plugin die *out-of-the-box* prima werkt. Voor de andere drie tools konden we plugins in Gradle zelf gebruiken. De uitdaging lag in het feit dat de standaardinstellingen van die Gradle-plugins niet aansloten bij de manier waarop we de tools vroeger vanuit Ant aanriepen. Gradle is flexibel genoeg om die configuratie in detail aanpasbaar te maken, maar de Gradle-plugins – die door de *community* worden gemaakt – zijn een stuk minder goed gedocumenteerd dan Gradle zelf. Om die reden kostte het wat *trial and error*, maar uiteindelijk lukte het om onze buildstraat voor het generieke component van programmeren tot publicatie in Nexus werkend te krijgen.

Conclusie

En dat was precies waar we wilden komen. Zoals we in de inleiding al aangaven, was het zeker geen reis zonder ongeregeldheden. Meer nog dan dat: het werk is nog niet klaar. We zijn momenteel bezig om naast het generieke component een tweede component naar Gradle te migreren. In totaal verwachten we op zo'n twintig componenten uit te komen, die dan elk met Gradle zullen werken. Met de opgedane ervaring verwachten we die wel een stuk sneller te kunnen migreren.

Ondanks het openstaande werk en de vertraging die we opliepen, is onze ervaring met de Gradle-migratie positief. Ons project is door de opdeling in componenten sneller te doorgronden voor nieuwe teamleden. En de verwachte performanceverbeteringen zijn al duidelijk zichtbaar: builds gaan niet meer onnodig af en het incrementele karakter zorgt voor korte doorlooptijden. Bovendien kan Gradle onze unit-tests zonder handmatige configuratie parallel uitvoeren.

We sporen daarom graag andere projecten aan om een migratie naar Gradle te overwegen. De meeste tijd ging bij ons zitten in het wegwerken van *technical debt* uit het verleden: zaken die met Gradle zelf niet veel te maken hebben. Het loont dus om hier alvast aandacht aan te besteden voordat je de migratie start. Een snelle *check-up*:

1. Maak dependencies transitief en gebruik één versie van elk artifact;
2. Doorgrond de bestaande scripting en wees op de hoogte van afwijkende constructies die in Gradle mogelijk problemen gaan opleveren;
3. Gebruik in de samenwerking met de buildstraat zo veel mogelijk standaardinstellingen.

Als deze zaken goed op de rails staan, dan staat niets je meer in de weg om de migratie te starten. Begin bij componenten die niet afhankelijk zijn van andere onderdelen en werk vervolgens 'omhoog' in de structuur van je project. Let er bij het vertalen op dat je steeds kleine stappen neemt en verifieer de correcte werking na elke stap. Op die manier stel je zeker dat de nieuwe situatie probleemloos aansluit bij eventuele overblijfselen uit het verleden.

Met deze aanpak zul je merken dat Gradle je op allerlei vlakken tegemoetkomt met structuur én flexibiliteit, en dat je en *passant* weer grip hebt gekregen op je buildproces. ■

MET DEZE AANPAK ZUL JE MERKEN DAT GRADLE JE OP ALLERLEI VLAKKEN TEGEMOETKOMT MET STRUCTUUR ÉN FLEXIBILITEIT, EN DAT JE EN PASSANT WEER GRIP HEBT GEKREGEN OP JE BUILDPROCES

REFERENTIES

1 *Devoxx 2011 Fireside Chat - www.parleys.com/tutorial/devoxx-fireside-chat*
2 *Gradle: Build Automation for the JVM, Android, and C/C++ - www.gradle.org*
3 *Hubert Klein Ikkink / Gradle is klaar voor de enterprise - www.nljug.org/databasejava/gradle-is-klaar-voor-de-enterprise*
4 *Gradle User Guide / Multi-project Builds - gradle.org/docs/current/userguide/multi_project_builds.html*
5 *Gradle User Guide / The Java Plugin - gradle.org/docs/current/userguide/java_plugin.html*
6 *Gradle User Guide / Using Ant from Gradle - gradle.org/docs/current/userguide/ant.html*
7 *ZipDiff - zipdiff.sourceforge.net*