Pipeline as code

Continuous Delivery pipelines met Jenkins 2

In de afgelopen tien jaar heeft Jenkins zich ontpopt tot de standaardtool voor automatisering in software development. Dit jaar is de eerste grote Jenkins release sinds tijden uitgekomen: Jenkins 2.0. In dit artikel lees je wat er allemaal nieuw is.

Op JavaOne vertelde Kohsuke Kawaguchi, de maker van Jenkins, dat er meer dan 120.000 actieve installaties zijn. Voor meer dan 90% van de gebruikers is Jenkins "mission critical". Jenkins is dus niet zomaar een hobbyproject. Een belangrijke factor in het succes van Jenkins is de enorme verzameling plug-ins, die beschikbaar zijn. Er is een heel ecosysteem ontstaan, waarmee Jenkins in vrijwel elke omgeving kan worden ingezet.

Nieuw in Jenkins 2

Jenkins 2 is een drop-in upgrade, volledig backward compatible met versie 1.6, met drie grote wijzigingen.

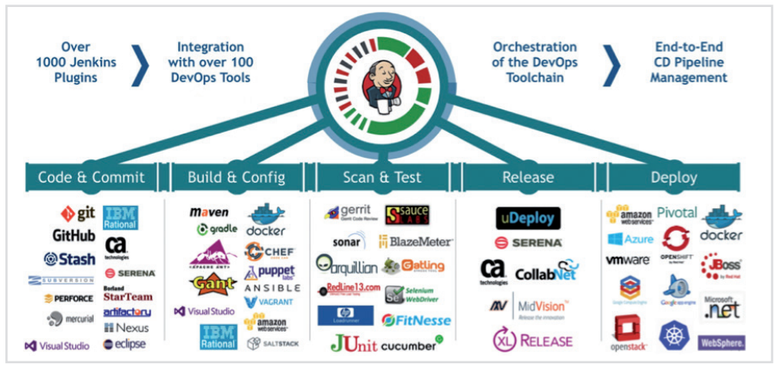
Betere out-of-the-box ervaring
Vroeger was Jenkins standaard vrij beperkt. Je moest veel plug-ins installeren, voordat je aan de gang kon. Dat is nu beter geregeld. Je kunt tijdens de installatie kiezen voor een aanbevolen set plug-ins, waardoor je direct aan de slag kunt.

Security is nu standaard. Bij de eerste keer opstarten wordt er een initieel admin wachtwoord gezet, dat je uit de logs moet vissen. Dat is iets meer werk vergeleken met vroeger waar alles open stond, maar het is wel een stuk veiliger. Zeker als je te weten komt, dat er bots zijn die het internet af scannen naar onbeveiligde Jenkins-instanties.

Opgefriste GUI
De GUI van Jenkins is licht verbeterd ten opzichte van Jenkins 1.x. Bij installatie en upgrade krijg je nu een wizard voorgeschoteld, die je helpt bij de configuratie. Eenmaal geïnstalleerd zijn de wijzigingen marginaal. Het aanmaken van een nieuwe job ziet er iets anders uit en bij het configureren van jobs heb je nu tabjes, waarmee je snel naar een bepaalde sectie kunt springen. De echt grote GUI-wijzigingen zitten in project "Blue Ocean", waarover je verderop meer kunt lezen.

Pipeline as code
Veruit de grootste wijziging in Jenkins 2 is het *pipeline as code* concept. Een pipeline is een geautomatiseerde reeks van stappen, die software vanuit versiebeheer naar de handen van je gebruikers brengt. Met *pipeline as code* kun je pipelines in tekstuele vorm beschrijven en in versiebeheer bewaren. Deze pipelines schrijf je met een flexibele Groovy DSL.

Bert Jan Schrijver is software craftsman bij JPoint en is momenteel werkzaam bij de Nationale Politie



Afbeelding 1: Het Jenkins ecosysteem. Bron: [1]

In Jenkins 1 werkte het maken van pipelines niet echt optimaal, zeker niet in een microservice-omgeving als je veel losse jobs hebt voor build, test en deploy. Een project met 20 services leverde dan al gauw een stuk of 100 Jenkins jobs op.

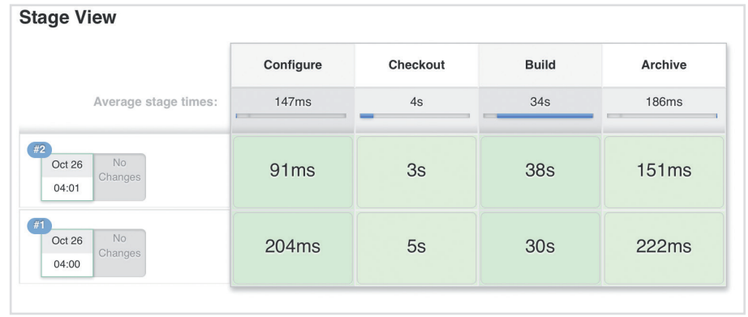
Met Jenkins 2 pipelines werkt dat een stuk praktischer om twee redenen:

1. Je kunt jobs precies zo samenstellen als je zelf wilt. Een enkele job kan een complete pipeline van commit tot en met productie-deployment afhandelen.
2. Hergebruik van onderdelen van jobs is veel eenvoudiger. Hierover verderop meer.

Jenkins wordt hiermee gepositioneerd voor continuous delivery use cases en voor andere meer complexe automatiserings-scenario's.

Pipelines

Je kunt een Jenkins 2 pipeline op twee manieren definiëren: door de pipeline in de Jenkins GUI in te tikken of door een bestand met daarin de pipeline-definitie in versiebeheer te plaatsen. Voor de laatste optie is het een conventie om een "Jenkinsfile" in de root van het project te zetten waar de job betrekking op heeft. Dit is vergelijkbaar met de Dockerfile of een Maven pom.xml: een standaard configuratie-file op een standaard plek.



Afbeelding 2: visualisatie van een Jenkins pipeline

Als je in je IDE aan Jenkinsfiles werkt, dan kun je een GDSL file importeren [2]. Dit bestand regelt de highlighting en code completion in je IDE.

Een voorbeeld
Het volgende pipeline script bouwt een eenvoudig Maven project (zie **Listing 1**).

De eerste stap is het kiezen van het type Jenkins node waar de build op moet gaan draaien. In het eenvoudigste scenario heb je alleen een Jenkins "master" node. In wat complexere omgevingen zie je vaak een master node en een aantal tot zelfs tientallen slave nodes om veel builds tegelijkertijd te kunnen draaien.

In dit voorbeeld geven we aan dat de job op een node met label 'java8' moet draaien. Jenkins kiest dan de eerstvolgende beschikbare node waar dit label aan toegekend is. De hele pipeline draait nu op deze 'java8' node. Je kunt er ook voor kiezen om verschillende onderdelen op verschillende nodes en zelfs parallel te draaien.

```
node('java8') {

    stage('Configure') {
        env.PATH = "${tool 'maven-3.3.9'}/bin:${env.PATH}"
    }

    stage('Checkout') {
        git 'https://github.com/bertjan/spring-boot-sample'
    }

    stage('Build') {
        sh 'mvn -B -V -U -e clean package'
    }

    stage('Archive') {
        junit allowEmptyResults: true,
              testResults: '**/target/**/TEST*.xml'
    }

}
```

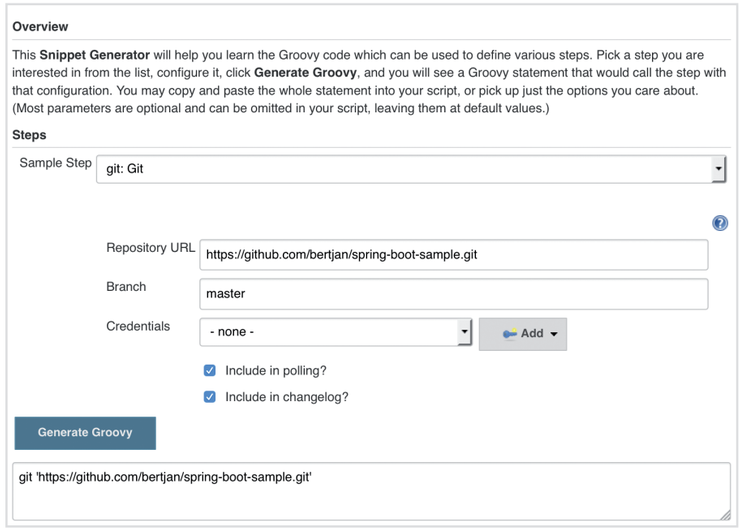
Listing 1: pipeline script voor een Maven project

De pipeline in dit voorbeeld kent vier "stages". Een stage is een stap in een pipeline, die als zodanig herkenbaar is in de Jenkins GUI. In de eerste stage zoeken we geïnstalleerde tool 'maven-3.3.9' op en maken deze beschikbaar door hem in de *path*-variabele van het OS te zetten. De tweede stap is een Git checkout, de derde een Maven build en de vierde archiveert de resultaten van de unit tests.

Als we de pipeline uitvoeren, zien we de volgende resultaten terug in de Jenkins GUI (zie **Afbeelding 2**).

De praktijk leert dat pipelines meegroeien van eenvoudig naar complex. Met gecodeerde pipelines kun je meerdere jobs definiëren zonder jezelf te herhalen. Dit is een groot voordeel ten opzichte van traditionele, point-and-click pipelines.

Pipelines overleven een herstart van Jenkins. Dat is handig als je af en toe een upgrade van Jenkins wilt uitvoeren, waarvoor een herstart nodig is. Jenkins bevat daarnaast de mogelijkheid om een pipeline te "replayen", waarbij je via de GUI kleine wijzigingen kunt doen en de pipeline vervolgens opnieuw kan uitvoeren. Dat is erg handig bij het ontwikkelen van gecodeerde pipelines.



Afbeelding 3: Pipeline snippet generator

Pipeline syntax

Als je nog niet eerder met de Groovy DSL van Jenkins hebt gewerkt, dan is de pipeline syntax even wennen. Om de overgang wat gemakkelijker te maken, vind je in de Jenkins GUI een ingebouwde pipeline snippet generator. Hiermee kun je de meest voorkomende build-acties alsnog op de traditionele manier bij elkaar klikken en van daaruit de bijbehorende pipeline DSL code genereren.

De pipeline referentie-documentatie [3] is ook goed bruikbaar en geeft een behoorlijk compleet overzicht van de mogelijkheden. Je merkt dat de pipeline DSL nog vrij nieuw is, want nog niet alle plug-ins zijn via de DSL te configureren. Mocht je er met de plugin-documentatie niet uitkomen, dan kun je altijd nog de broncode induiken. Jenkins componenten en plug-ins zijn Java- en Groovy code en daardoor meestal ook rechtstreeks te integreren in je pipeline. Daarvoor moet je meestal wel op de hoogte zijn van de interne werking van Jenkins.

Workflow libs repository

Jenkins biedt zelf een aparte Git-repository aan om herbruikbare pipeline scripts in te plaatsen. Deze scripts zijn vervolgens voor al je builds beschikbaar. Het feit dat je deze interne Git-repository moet gebruiken, vind ik wat onhandig. Het voelt niet echt natuurlijk wanneer al je code en Jenkins-files al op een andere Git-server hebt staan. Verderop in dit artikel vind je een voorbeeld van het gebruik van deze repository.

Voorbeelden

De beste manier om bekend te raken met de pipeline syntax is door er eenvoudigweg mee aan de slag gaan. In **Listing 2**, **3**, **4** en **5** vind je een aantal voorbeelden van de meer geavanceerde constructies, die mogelijk zijn. Voor wat uitgebreidere voorbeelden, zie [5].

Blue Ocean

Het Blue Ocean project is een complete make-over van de Jenkins user experience. Initieel richt het project zich op de GUI voor developers rondom pipelines. Het uiteindelijke doel is om de gehele GUI geleidelijk te vervangen.

HET BLUE OCEAN PROJECT IS EEN COMPLETE MAKE-OVER VAN DE JENKINS USER EXPERIENCE

```
try {
    // build-stappen hier
} catch (e) {
    currentBuild.result = 'FAILED'
    mail to: '<to>', subject: '<subject>',
         body: '<body>', attachLog: true
    throw e
}
```

Listing 2: Versturen van een e-mail wanneer er iets fout gaat in een build

```
// In bestand common/Constants.groovy:
class Constants {
  static final SONAR_URL =
    'https://sonar.company.com';
}
return this;

// In bestand Jenkinsfile:
load 'common/Constants.groovy'
sh "mvn -B -V -U -e sonar:sonar " +
   "-Dsonar.host.url='${Constants.SONAR_URL}'"
```

Listing 3: Gebruik van klassen, constanten en includes

```
// In repo ssh://<user>@<jenkins>:2222/workflowLibs.git,
// bestand my.company.MyWorkflowSteps:
package my.company
def someBuildStep() {
  // implementatie van een build-stap
}

// In Jenkinsfile:
def mySteps = new my.company.MyWorkflowSteps()
mySteps.someBuildStep()
```

Listing 4: Definiëren van een herbruikbare workflow-stap in het interne workflowLibs-repository van Jenkins

```
parallel 'unit': {
  sh 'mvn test'
}, 'integration': {
  sh 'mvn integration-test'
}
```

Listing 5: Parallel uitvoeren van één of meerdere onderdelen van een build

Blue Ocean beta is als plug-in beschikbaar. Je kunt het installeren via de Jenkins plug-in manager. Ga naar de “Advanced” tab en voer bij “Update Site” de URL van de experimentele plug-in registry in [4]. De laatste stap is het uitvoeren van een update van de lijst beschikbare plug-ins.

Nu kun je Blue Ocean installeren. Na een herstart van Jenkins zie je bovenin het scherm een knop "Try Blue Ocean UI". Een klik daarop brengt je bij het overzicht van je builds (zie **Afbeelding 4**).

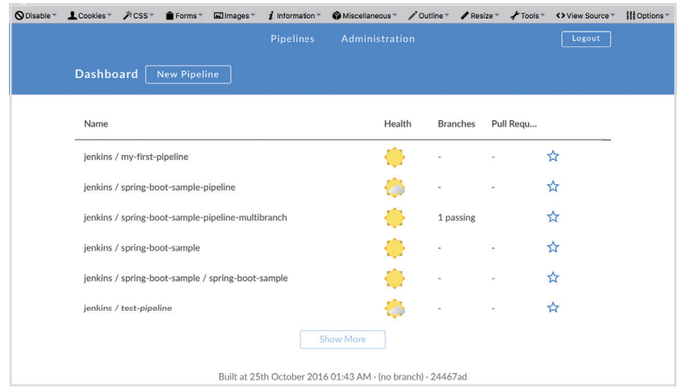
Voor de doorgewinterde Jenkins-gebruiker zal dit een lichte schok zijn. Jenkins ziet er opeens namelijk enigszins sexy uit. Bij de details van een pipeline zie je een soort spoorboekje met stations voor elke pipeline stage. Per stage kun je de logs van de build uitklappen (zie Afbeelding 5).

Ik vind het een hele verbetering. Functioneel gezien is het nog niet echt heel anders, maar het ziet er wel duidelijker en vooral een stuk mooier uit dan de traditionele interface.

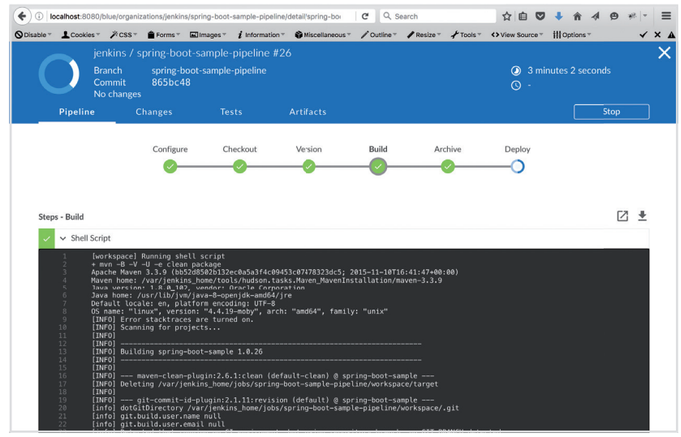
Multibranch pipeline

Een laatste noemenswaardige feature is de multibranch pipeline plug-in. Hiermee kun je een pipeline aan een Git-project koppelen. Jenkins maakt dan automatisch voor alle branches in het project pipelines aan. Dit is erg handig als je veel met pull requests en feature branches werkt. Het scheelt je een hoop onderhoud aan je jobs.

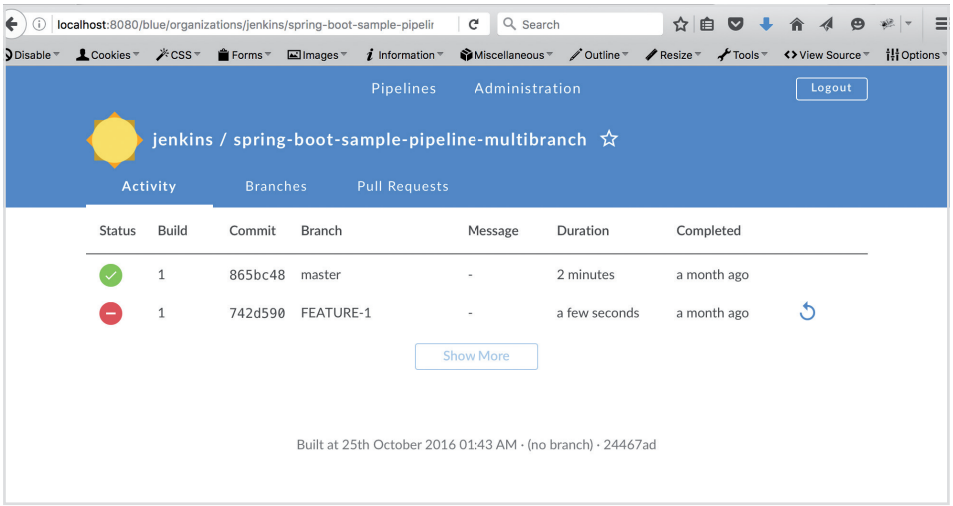
De GitHub Organization Folder Plugin gaat nog een stapje verder. Die maakt automatisch pipelines aan voor alle branches van alle repositories in een Github organisatie. Het is wel een voorwaarde dat elke repository netjes een Jenkinsfile bevat.



Afbeelding 4: Overzicht builds in de Blue Ocean GUI



Afbeelding 5: Details van een pipeline build



Afbeelding 6: Multibranch pipeline

in het generiek opzetten van de jobs met herbruikbare stappen en anderzijds in de bestaande jobs. Ook al deden die allemaal ongeveer hetzelfde, er zaten toch nog best wat verschillen tussen. Per job moesten we inschatten of die verschillen bewust of onbewust waren.

We hebben ervoor gekozen om één Git-repository te gebruiken met daarin alle Jenkins-file definities voor alle jobs. Nadeel daarvan is dat de Jenkins-file en de code van een project niet bij elkaar staan. Voordeel is dat je al je builds bij elkaar hebt en dat je makkelijk in een keer een refactorslag door al je builds kunt doen.

We maken minimaal gebruik van de interne workflow repository in Jenkins. Wij gebruiken hem eigenlijk alleen om de generieke componenten van onze builds te bootstrappen. Deze componenten staan dan weer in losse Groovy DSL files in de repository met al onze builds erin. We onderscheiden twee typen componenten: low-level stappen, die een klein onderdeel van de build doen en meer high-level stappen, die een complete pipeline definiëren. Het voordeel van deze aanpak is dat jobs die vrijwel hetzelfde zijn een high-level definitie kunnen hergebruiken, terwijl jobs die net iets anders zijn alsnog op een lager niveau hergebruik kunnen doen.

Het resultaat: onze builds zijn nu veel consistenter. Het is bijna geen werk meer om nieuwe jobs toe te voegen en we kunnen op één plek de definitie van vrijwel al onze jobs beheren.

Toekomstige verbeteringen

Op JavaOne gaf Kohsuke Kawaguchi een kijkje in de toekomst van Jenkins. De eerstvolgende wijzigingen zijn gericht op gebruikersgemak. Er komt een eenvoudiger pipeline model aan dat minder op programmeren lijkt en meer declaratief is. Jenkins wil zowel point-and-click als tekst-editor-gebruikers bedienen. Een wijziging waar ik zelf naar uitkijk: pipeline DSL verwerking faalt bij het inlezen en niet bij het uitvoeren. Nu kom je pas bij het draaien van de job achter typo's in je Jenkins-file. Dat kost onnodige trial & error-tijd.

Conclusie

Jenkins 2.0 is een krachtig continuous delivery platform. De nieuwe versie is een drop-in upgrade voor 1.6-installaties, bevat GUI-verbeteringen en een beter verzorgde gebruikerservaring. De kern van 2.0 is pipeline as code, waarmee je je jobs in een DSL beschrijft en in versiebeheer kunt zetten. Algemeen gezegd: minder clicks, meer code. Als je het mij vraagt is er geen reden om nog op 1.6 te blijven hangen. Upgraden maar! ■

REFERENTIES

[1] **http://www.slideshare.net/asotobu/jenkins-20-65705621**
[2] **http://st-g.de/2016/08/jenkins-pipeline-autocompletion-in-intellij**
[3] **https://jenkins.io/doc/pipeline/steps**
[4] **http://updates.jenkins-ci.org/experimental/update-center.json**
[5] **http://www.slideshare.net/BertJanSchrijver/javaone-2016-pipeline-as-code**