



Pipeline as code

Building Continuous Delivery Pipelines with Jenkins 2.0

Bert Jan Schrijver

bertjan@jpoint.nl



 @[@bjschrijver](https://twitter.com/bjschrijver)

Let's meet



Bert Jan Schrijver

jpoint

OPEN VALUE

POLITIE

.nl.
jug

Outline

- Definitions
- Jenkins 1.x
- What's new in Jenkins 2.0?
- Pipeline as code
- Pipelines in depth
- Jenkins 2.0 in the wild

*Thanks to @alexsotob
and @kohsukekawa!*

Definitions

Who's who in CI & CD

Continuous Integration

Team members integrate their work frequently. Commits are verified by automated builds and tests.

Continuous Deployment

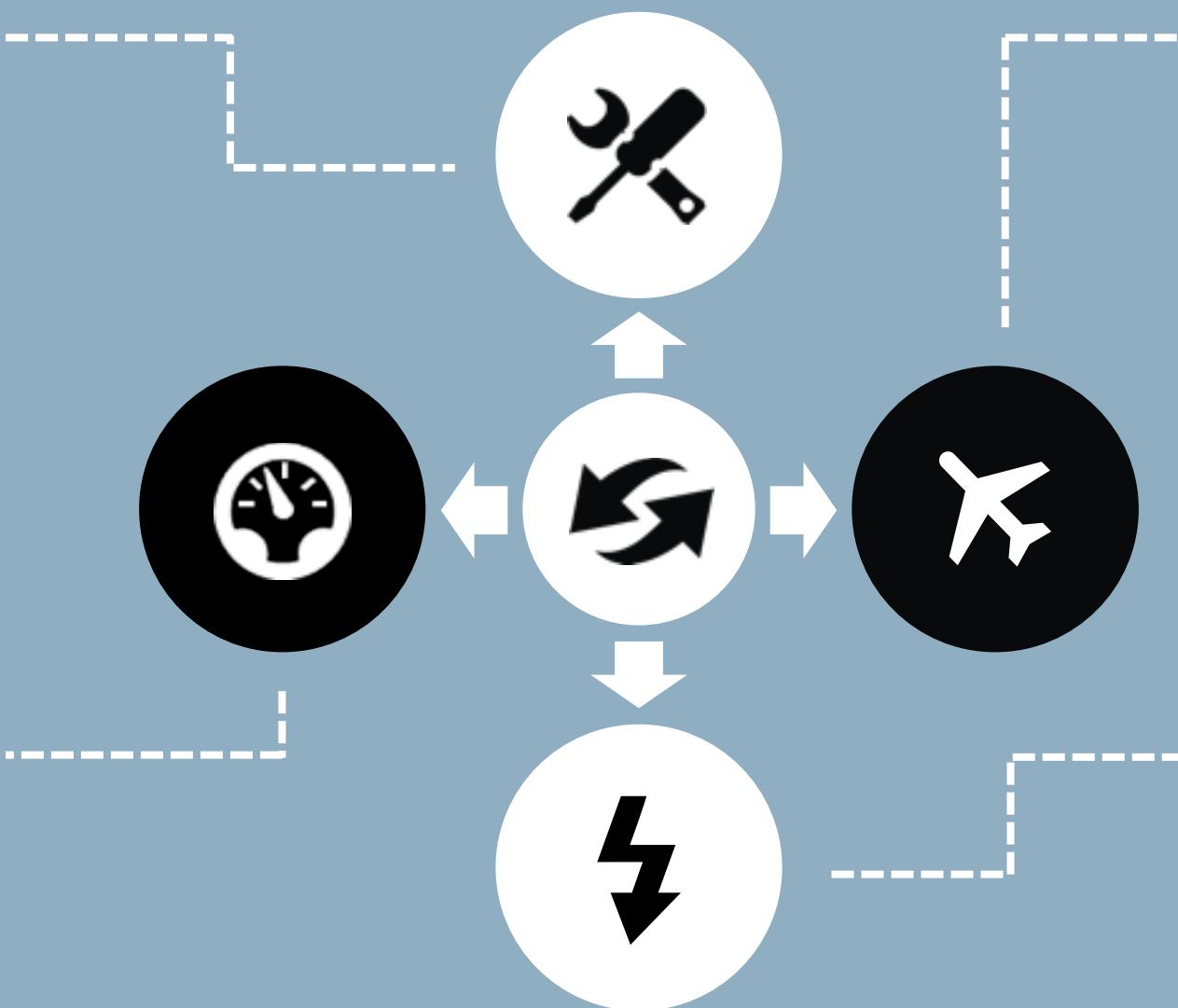
Every change goes through the build/test pipeline and automatically gets put into production.

Continuous Delivery

Building and testing software in such a way that the software can be released to production at any time.

Pipeline

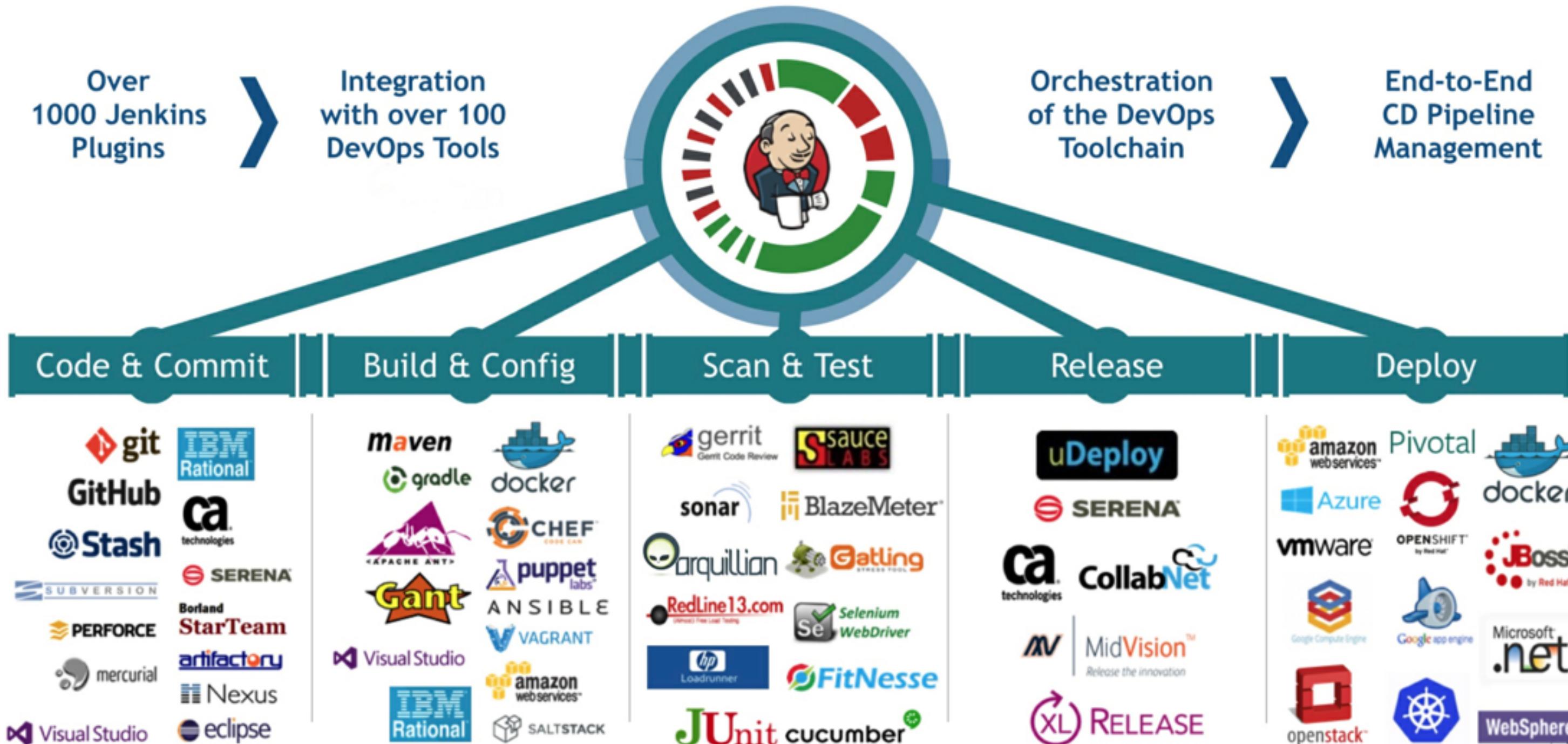
An automated sequence of stages to deliver software from version control to your users.



About Jenkins

- De-facto standard tool for automation in software development and beyond
- Around for 10+ years, millions of users
- Over 120.000 active installations
- Jenkins is mission critical for >90% of its users
- Version 2.0: first major release in years

Jenkins: an overview



Source: <http://www.slideshare.net/asotobu/jenkins-20-65705621>

DEMO



What's new in Jenkins 2?

- Better out-of-the-box experience
 - Default set of plugins
 - Secured by default
- Revamped UI
- Pipeline as code
- In general: more code, less GUI, less state
- Drop-in upgrade, backwards compatible w/1.6

Pipeline as code

- Key new feature
- Positions Jenkins for:
 - continuous delivery use cases
 - other more complex automations of today
- Allows to:
 - describe chain of automation in textual form and put it in version control

Pipeline as code

- Grows with you from simple to complex
- Handle lots of jobs without repetition
- Survives Jenkins restarts
- Brings next level of reuse to Jenkins

My first pipeline

```
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'  
    }  
  
}
```

DEMO



Pipeline syntax

- Built-in syntax and snippet generator
- Groovy DSL definition (GDSL file) for IDE
- Pipeline reference:
<https://jenkins.io/doc/pipeline/steps>
- Plugin documentation
- If all else fails: dive into the source

SNIPPETS



Archive build artifacts

```
archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true
```

Cleanup old builds

```
properties(  
    [buildDiscarder(logRotator(  
        artifactDaysToKeepStr: '',  
        artifactNumToKeepStr: '',  
        daysToKeepStr: '',  
        numToKeepStr: '10'))) ] )
```

E-mail notification

```
try {  
    // build steps here  
}  
catch (e) {  
    currentBuild.result = "FAILED"  
    def subject = 'Build \' + env.JOB_NAME + '\' (branch \' + branch + '\') ' +  
        'failed in Jenkins'  
    def body = 'Build log is attached. Details: ' + env.BUILD_URL  
    def to = 'email@domain.com'  
    mail to: to, subject: subject, body: body, attachLog: true  
    throw e  
}
```

Including files, using constants

```
// File common/Constants.groovy:

class Constants {
    static final MAJOR_VERSION_NUMBER = '3.2.1';

    static final SONAR_URL = 'https://sonar.my.company.com';
}

return this;

// Jenkinsfile:

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

Re-usable workflow steps

```
// In repo ssh://<username>@<jenkins-url>:2222/workflowLibs.git,  
// file my.company.package.MyWorkflowSteps  
  
package my.company.package  
  
def someBuildStep() {  
    // Some build step  
}  
  
// In Jenkinsfile:  
  
def mySteps = new my.company.package.MyWorkflowSteps()  
mySteps.someBuildStep()
```

Parallel run on multiple nodes

```
stage('Checkout') {
    git 'https://github.com/bertjan/spring-boot-sample'
    stash excludes: 'build/', includes: '**', name: 'source'
}

stage ('Test') {
    parallel 'unit': {
        node {
            // perform unit test
            unstash 'source'
            sh 'mvn test'
            junit '**/build/test-results/*.xml'
        }
    }, 'integration': {
        node {
            // perform integration test
            unstash 'source'
            sh 'mvn integration-test'
            junit '**/build/test-results/*.xml'
        }
    }
}
```

DEMO



Jenkins 2.0 in the wild

- Upgraded a 1.6 instance with about 50 builds
- Replaced all builds with pipelines
- Minimal use of Jenkins workflow internal repo
- One single Git repo for all builds
- Re-usable pipelines and steps
- Builds are far more consistent

Jenkins 2.0: Moving forward

- Upcoming changes: focused on ease of use
 - simplified pipeline model
 - look less like programming, more declarative
 - cater both point-and-click and editor people
 - fails when Jenkins reads it, not when it runs it

SUMMARY



Summary

- Jenkins 2.0: powerful continuous delivery platform
- UI improvements, more curated experience
- Pipeline as code: less clicks, more code

Resources



<https://github.com/bertjan/spring-boot-sample>

- Article for Java magazine (Dutch), online soon at <https://github.com/bertjan/javamagazine>

The screenshot shows a magazine spread. The left page features a large title 'Pipeline as code' and a subtitle 'Continuous Delivery pipelines met Jenkins 2'. It includes several columns of text, a photo of Bert Jan Schrijver, and a diagram illustrating the Jenkins ecosystem. The right page contains a section titled 'Stage View' showing pipeline execution times, a 'Listing 1: pipeline script voor een Maven project' showing Groovy DSL code, and a 'Listing 2: visualisatie van een Jenkins pipeline' showing a Jenkins pipeline configuration interface.

Pipeline as code

Continuous Delivery pipelines met Jenkins 2

In de afgelopen tien jaar heeft Jenkins zich ontwikkeld 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 hobbyprojectje. 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 drukke 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 gegeven.

Over 1000 Jenkins Plugins

Integration with over 100 DevOps Tools

Orchestration of the DevOps Toolchain

End-to-End CD Pipeline Management

Code & Commit

Build & Config

Scan & Test

Release

Deploy

git GitHub Stash Jenkins Ansible Chef Sonar JBossFuse Deploy Serena C2 CollabNet Jenkins UUnit eucumbel RELEASE

Stage View

	Configure	Checkout	Build	Archive
Average stage times	147ms	4s	34s	169ms
Oct 26 04:01	91ms	3s	38s	151ms
Oct 26 04:00	204ms	5s	30s	222ms

Afbeelding 2: visualisatie van een Jenkins pipeline

pom.xml is een standaard configuratiefile op een standaard plek.

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

Een voorbeeld:

```
pom.xml</pre>
```

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

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 productiedeploy afhandelen.
2. Hergebruik van onderdelen van jobs is veel eenvoudiger. Hierover verderop meer.

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 geinstalleerde tools.

"maven-3.3.9" op en maken deze beschikbaar door hem in de PATH-variable 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 meegenomen kunnen worden 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.

Listing 1: pipeline script voor een Maven project

```
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 -U -e clean package'  
    }  
    stage('Archive') {  
        junit allowEmptyResults: true,  
            testResults: '**/target/**/TEST*.xml'  
    }  
}
```

Listing 2: visualisatie van een Jenkins pipeline

```
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 -U -e clean package'  
}  
stage('Archive') {  
    junit allowEmptyResults: true,  
        testResults: '**/target/**/TEST*.xml'  
}
```

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



Questions?

Thanks for your time!

Got feedback? Tweet it!

