De Jenkins Maven / Freestyle à Pipeline

Speaker : Adrien Lecharpentier (CloudBees)
Format : Tools in action
Slides : https://speakerdeck.com/alecharp/freestyle-a-pipeline

Constant dans Jenkins : 1 job par livrable => 10 plugins à configurer
Pipeline permet de simplifier la configuration

Proposé dans Jenkins 2.0 qui est actuellement en RC1, Pipeline offre :

- Un DSL pour décrire la construction du projet
- De la récupération des sources à la livraison
- Une forêt de plugins

Cas d'utilisation en 2 étapes :

1. Projet maven avec Spring Boot : construit un JAR auto-exécutable
2. Livraison d'une image Docker

Actuellement, il faut 2 jobs :

1. Job Maven qui build le JAR
2. Si Job 1 OK, déclenchement d'un build d'image Docker

Adrien laisse place à une démo de Pipeline :

Récupération d'un nœud (master ou slave)
Découpé en plusieurs étapes

```
node {
 stage 'Checkout'
 git url : 'https://github.com/alecharp/simple-app.git', branch : 'develop'

// Récupération artisanale du SHA-1 court en 3 lignes dans la variable short_commit
….
```

```
 stage 'Build'
 withMaven (‘mvn clean verify')

// Rapport de tests
step([$class :’JUnitResultArchiver’, testResults : ‘target/surefire-reports/*.xml’
step([$class :’JUnitResultArchiver’, testResults : ‘target/surefire-reports/*.xml’

dir(‘target’) { archive ‘*.jar’}
// Mise en mémoire du JAR construit précédemment (utilisation du stash)
stash name : ‘binary’, includes : ‘target/*.jar’

dir(‘src/main/docker’) { stash name : ‘docker’, includes : ‘Dockerfile’

}

node(‘docker’) {
 unstash :’docker’
unstash : ‘binary’

stage ‘Build Docker img’
image  docker.build(« alecharp/simpleapp :${short_commit} »)

// On démarre le conteneur afin de la valider manuellement
container = image.run('-P’)
sh ‘docker port ${container.id} 8080 > DOCKER_IP’
ip = readFile(‘DOCKER_IP’).trim()
sh (rm DOCKER _IP)
}

// Validation manuelle
stage ‘Validation’
try {
input message : http://{ip}, Is it ok? » n ik : ‘Publish it’
} finally {
 node(‘docker’) { container.stop}
}

node(‘docker’) {
        stage ‘Publishing’
        docker.withRegistry(‘http://localhost:5000’, ‘’)
}

Pour se garantir que le nœud Jenkins a JDK et Maven ;

def withMaven(def body) {
  def java = tool ‘oracle-8u77’
```

```
  def maven = tool ‘maven-3.3.9’

withEnv([« JAVA _HOME =${java}, « PATH+MAVEN=${maven/bin]) {
 body.call()
}
```

Il existe un **snippet generator** qui assiste le développeur pour, par exemple, générer le rapport de tests d’intégration et unitaires (failsafe, surefire)

Même lorsque Jenkins est en attente de la réponse de l’utilisateur, le nœud est disponible pour d’autres pipeline. Tant que l’étape n’est pas validée, l’utilisateur attend. Possibilité d’envoyer un mail.

Adrien imagine scénario dans lequel il serait nécessaire de passer à Java 9. Avant Pipeline, il était nécessaire d’aller modifier la conf Jenkins. Il n’y avait pas de synchro entre Jenkins et le besoin du projet.
A présent, on peut commiter un fichier **Jenkinsfile** dans le projet.
Ressemble au travis.yml

Multibranch pipeline : pipeline as code
Cliquer sur Branch Indexing => Jenkins recherche les Jenkinsfile et lance le build.

Démo : création d’une nouvelle branche foobar
Indexation relancée => crée un nouveau Job
Suppression de la branche foobar => Jenkins détruit le job
Plus de clonage de build / suppression de branches inutiles.