Intégration Continue : Jenkins et Sonar

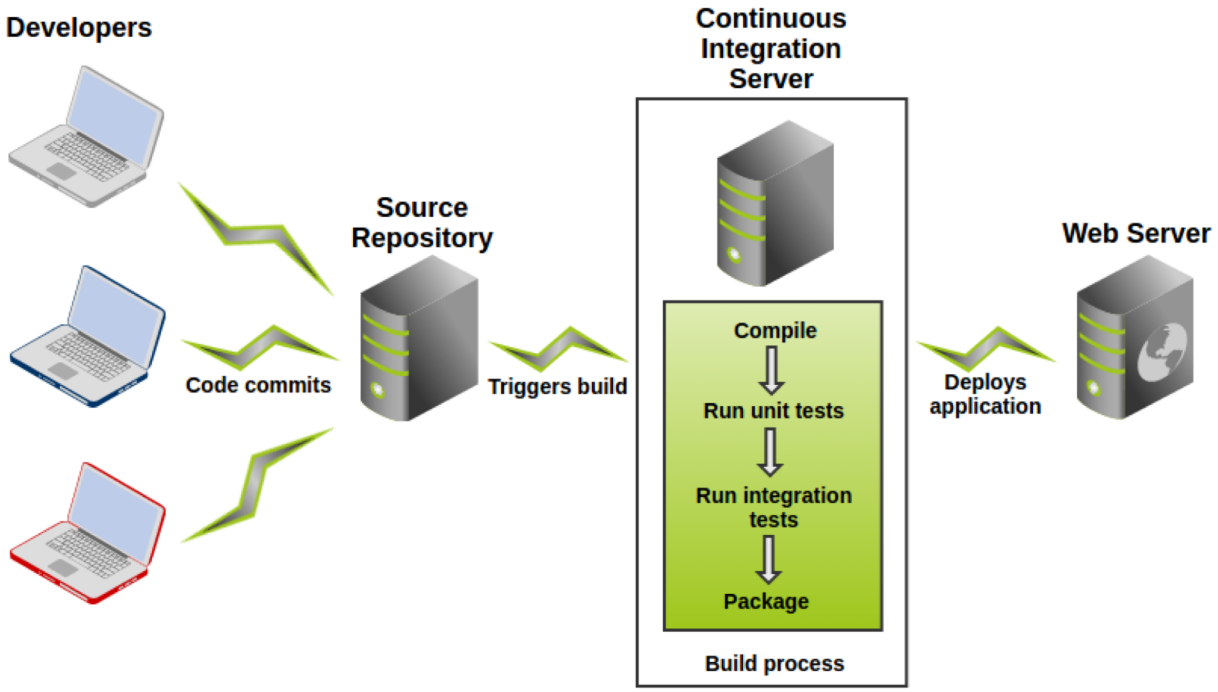
Intégration Continue avec Jenkins, Sonar et Nexus

Présentation

Présentation

- ➔ Sans usine logicielle
  - Développeur récupère le projet du SCM
  - Développe une fonctionnalité, corrige un bug
  - Push ses modifications
  - …
  - Et puis que fait-on ?
  - Où est-ce qu'on redéploie l'application ?
  - Où sont les environnements de test, préproduction et production ?
  - Les tests lancés localement suffisent-ils ?
  - Le code respecte-il les conventions établies ? si non qui m'en informe ?
- ➔ Le développeur et son IDE ne suffisent pas au développement d'un projet

Présentation



Présentation

- ➔ Usine logicielle
  - ◆ L'environnement de travail complémentaire au développeur
  - ◆ Le développeur développe, "l'usine logicielle" s'occupe du reste
  - ◆ Outils (installable)
    - *Nexus, Artifactory, Archiva (Gestion des artefacts maven)*
    - *Jenkins, Hudson (Intégration Continue)*
    - *SonarQube (Contrôle Qualité du code)*
  - ◆ SaaS
    - *TravisCI, CircleCI (Intégration Continue)*
    - *Cloudbees (utilise Jenkins)*

Concept

Concept

- ➔ Intégration Continue
  - ◆ Maintenir le bon état de santé d'un projet à chaque modification
  - ◆ Détecter au plus tôt les problèmes
  - ◆ Automatisation du build, déploiements, …
  - ◆ Seul moyen de lancer des tests qui ne seraient pas exécutable sur un poste développeur
- ➔ DevOps
  - ◆ Le monde qui fait le lien entre le développement le déploiement
    - *Docker, Google cloud, Amazon Web Services, …*
  - ◆ Travail d'automatisation, scripts, ...

Outils

Outils

- Jenkins
  - Build et déploie un projet
  - Exécute les phases maven dès modification sur le projet
- SonarQube
  - Calculer la dette technique
  - Effectue des rapports sur la qualité du code
  - Configuration de règles de codage

Jenkins

Jenkins

- Jenkins
  - Outil de build
  - Plugins d'intégration avec Maven, Gradle, Git,...
  - Création de job (tâche de construction d'un projet maven, script de déploiement, ...)
    - *Utiliser le 'Freestyle project' en précisant les goals de build*
      - 'mvn clean deploy sonar:sonar'

Config projet

Globalement, le processus permettant de compiler un projet est le suivant :

- récupération du projet à partir du serveur de contrôle de version ;
- création des fichiers de compilation ;
- compilation ;
- application des tests automatiques ;
- préparation de la version à distribuer ;
- mise en place de la version à distribuer.

Ce processus doit donc être configuré dans Jenkins. Deux méthodes de configuration s'offrent à vous :

- par le biais d'un pipeline, qui définit la configuration d'un projet grâce à du code ;
- la méthode historique (*freestyle*) où la configuration du projet se réalise à travers un formulaire à remplir. Dans ce cas, certaines étapes sont évidentes, d'autres se font grâce à l'exécution de scripts et ne sont donc pas guidées.

Job Freestyle

Il suffit de cliquer sur « Nouveau Item », donner un nom puis de sélectionner le type de job (ici, projet free-style). De là, vous serez redirigé vers la page de configuration de votre nouveau job.

La première partie de la page de configuration est négligeable. Elle permet d'indiquer une description du projet (affichée sur la page du projet), de choisir si la tâche a des **paramètres**, ou encore de gérer comment le projet est compilé ou est-ce que les anciennes compilations doivent être conservées.

Ensuite arrive la gestion du code source, ou comment nous allons récupérer le code source du projet. Les différents outils disponibles ici dépendent des modules installés.

Après la configuration de la récupération du code, vous pouvez configurer la périodicité de cette récupération, ou autrement dit, qu'est-ce qui déclenche l'exécution de cette tâche. Cela peut être un déclenchement manuel, à la suite d'autres tâches, périodique ou simplement parce que le code a changé (vérification périodique du code sur le serveur de gestion des versions).

Enfin arrive l'étape de compilation. Vous pouvez la configurer en ajoutant des scripts shell ou batch afin de lancer les commandes de compilation du projet. Les commandes sont évidemment exécutées sur la machine esclave. Donc, si votre esclave est une machine Linux, utilisez les scripts shell, sinon, les scripts batch.

Finalement arrivent les étapes suivant la réussite de la tâche. C'est ici que l'on pourra, par exemple, notifier par courriel de l'échec de la tâche ou faire un rapport.

Job pipeline

Pipeline est un module de plus en plus populaire permettant de configurer les tâches au travers d'un code. Ce dernier peut être sauvegardé avec le reste du code source de votre projet, sur votre serveur de contrôle de version.

Encore une fois, cela passe par « Nouveau Item ». Évidemment, le type de tâche à prendre sera « Pipeline ».

La page de configuration reprend des éléments de celle pour les tâches free-style tout en offrant une version épurée. En effet, mise à part la configuration des paramètres, des options globales de la tâche et de la répétitivité de la tâche, on arrive tout de suite à la configuration du pipeline et cela, à travers un simple éditeur de script.

Les scripts Pipeline apportent les grandes notions suivantes :

- un pipeline est l'ensemble du processus à exécuter ;
- un nœud (*node*) représente un environnement pouvant exécuter un pipeline (une machine esclave) ;
- un niveau (*stage*) représente un ensemble d'étapes de votre processus (par exemple, la récupération des sources, la compilation…) ;
- une étape (*step*) représente ce qu'il y a à faire à un moment donné (l'action à proprement parler, comme make).

Syntaxe

```
node {
   stage('Build') { // }
   stage('Test') { // }
    stage('Deploy') { // }
}
```

OU

```
pipeline {
   agent any
   stages {
      stage('Build') { steps { // } }
      stage('Test') { steps { // } }
      stage('Deploy') { steps { // } }
} }
```

Restreindre l’exécution d’une tâche

Tout comme avec les jobs freestyle, vous avez la possibilité de restreindre l’exécution d’une tâche à une machine (ou un lot de machines) précise grâce aux labels :

syntaxe script :

node (label: 'slave') { ... }

syntaxe déclarative :

pipeline { agent {label 'slave'} stages { ... } }

Paralléliser des tâches

La force du pipeline est aussi de permettre la parallélisation des tâches. Pour cela, le mot clé parallel a été implémenté :

syntaxe script :

parallel ("stream 1" : { node ("windows") { ... "stream 2" : { node ("linux") ...

syntaxe déclarative :

```
pipeline {
  agent none
  stages {
    stage('Run Tests') {
      parallel { stage('Test On Windows') { agent { label "windows" } steps { bat "run-tests.bat" } post { always { junit "**/TEST-*.xml" } } }
      stage('Test On Linux') { agent { label "linux" } steps { sh "run-tests.sh" } post { always { junit "**/TEST-*.xml" } } } } } }
```

Lancer la compilation par web

Il est possible de lancer la compilation sans passer par l'interface graphique. Voici l'URL permettant de lancer une tâche :

http://serveur:8080/job/NomDuJob/build

Vous pouvez aussi ajouter une attente avec le paramètre delay :

http://serveur:8080/job/NomDuJob/build?delay=30sec

La gestion des paramètres de compilation nécessite de passer par la page buildWithParameters :

http://serveur:8080/job/NomDuJob/buildWithParameters?param1=param1value

TP

TP

- Jenkins 1/2
  - Télécharger apache-tomcat-8.0.32.zip, jenkins.war (latest)
  - Unzip tomcat, mettre jenkins.war dans webapp
  - Lancer bin/startup.sh (ou équivalent sous Windows)
  - http://localhost:8080/jenkins
  - Suivre les étapes...

TP

- ➔ Jenkins 2/2
  - Créer un Job Jenkins pour ‘sample4CI’ (cloner https://gitlab.com/ctamisier/sample4CI.git)
  - Cron de build: toutes les 5 minutes (H/5 * * * *)
  - Build Maven: mvn clean install sonar:sonar

TP

- ➔ SonarQube
- ➔ Ajouter le tools Sonaqube à jenkins (Manage **Jenkins > Global Tool Configuration)**
- ➔ Corriger les problèmes signalés par Sonar suite au build sur le projet précendent.
- ➔ Prendre un projet android et intégrer SonarQube : https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-gradle/
  - ➔ Ou dl le projet de démo https://github.com/SonarSource/sonar-scanning-examples/tree/master/sonarqube-scanner-gradle