TP Intégration continue

Table of Contents

Démarrage Jenkins

Objectif: Installer et démarrer Jenkins

1. Aller dans /usr/share/jenkins ou télécharger jenkins.war
   a. https://jenkins.io/download/, choisir Long Term Support 2.150 / Generic java package (.war)
2. Lancer jenkins :
   a. Avec JDK 8:

```
java -jar jenkins.war --httpPort=8081
```

   b. Avec JDK 10:

```
java --add-modules java.xml.bind -jar jenkins.war --enable-future-java
--httpPort=8081
```

   c. Avec JDK 11: voir https://jenkins.io/blog/2018/06/17/running-jenkins-with-java10-11/
3. Aller dans le navigateur à l'adresse : http://localhost:8081
4. Récuperer le mot de passe Administrateur soit:
   a. Dans les logs affichés dans le terminal
   b. Dans le fichier ${JENKINS_HOME}/secrets/initialAdminPassword
5. **Passer l'installation des plugins** pour le moment: Cliquer sur la croix en haut à droite

Création d'un premier Job

Déclenchement périodique

Objectif: Découvrir les bases d'un job

1. Cliquer sur « Nouveau Item »
2. Choisir « Construire un projet free-style »
3. Ajouter une étape de build exécutant un "script shell" ou un "batch windows" (en fonction de l'environnement)
4. Ecrire un script créant un fichier 'jenkins.log' contenant le numéro du build et la date (le but est juste de tracer les exécutions du job)
   a. Utiliser les variables proposées par Jenkins (syntaxe $NOM_VARIABLE sous Linux, %NOM_VAR% sous windows)
   b. Afficher la date sous Linux: `date +"%d/%m/%y %H:%M"` et sous Windows `%date% %time%`
   c. Ajouter à la fin d'un fichier avec 'echo' et '>>'

5. Sur le build (une exécution), consulter la sortie de la console pour voir la trace de l'exécution
6. Sur le job, aller voir l'espace de travail qui doit contenir le fichier 'jenkins.log'

Objectif: Déclencher automatiquement l'exécution

1. Configurer le build pour démarrer toutes les 2 minutes
2. Aller voir l'historique des builds (en bas à gauche) et vérifier l'appel toutes les 2 minutes
3. Aller voir le contenu du fichier de trace 'jenkins.log' qui doit contenir l'historique

Build en échec

Objectif: Identifier un build en échec

1. Modifier le script shell en mettant volontairement une commande invalide
2. Attendre la prochaine exécution ou la déclencher manuellement
3. Constater l'échec du build dans
    a. le tableau de bord
    b. l'historique
    c. la console d'exécution

Désactivation du job

Objectif: Désactiver le job

Pour éviter d'avoir un job qui se lance toutes les minutes et surcharger inutilement les machines, on va le désactiver.

1. Sur la page du projet, cliquer sur le bouton "Désactiver le projet".

Activation sur commit

Installation du plugin Git

Le principe de l'intégration continue est de faire l'intégration au plus près des modifications. Généralement, elle est déclencher du code est commité dans le gestionnaire de source.

Objectif: Installation de plugins

Les plugins peuvent être installés directement depuis Internet. Ils peuvent également être installés manuellement à partir de fichier .hpi.

Configuration du proxy

1. Aller à la racine de Jenkins (le tableau de bord)
2. Aller sur « Administrer jenkins » / « Gestion des plugins » / « Avancé »

3. Renseigner
    a. Serveur: proxy-etu
    b. Port: 3128
    c. Ne pas mettre de nom utilisateur ni de mot de passe
4. Dans avancé tester avec l'url http://www.google.fr
5. En bas de la page, cliquer sur "vérifier maintenant"

Si l'accès réseau fonctionne depuis Jenkins

1. Aller sur « Administrer jenkins » / « Gestion des plugins »
2. Aller sur « disponible »: une liste de plugins dois s'afficher
3. Chercher le plugin « Git » (Sur la fin de la liste)
4. Cocher la case (ne pas cliquer sur le nom qui envoi sur la documentation)
5. Faire « installer sans redémarrer »

Si l'accès réseau ne fonctionne pas depuis Jenkins (la liste des plugins reste vide)

1. Récupérer les fichiers :
    a. scm-api.hpi
    b. git-client.hpi
    c. git.hpi
2. Aller sur « Administrer jenkins » / « Gestion des plugins » / « Avancé »
3. Dans la partie « Soumettre un plugin », aller prendre chacun des fichiers et faire « soumettre » Le plugin est maintenant disponible pour utiliser Git

Création du job lié à Git

Objectif: Déclencher le job à la suite d'un Commit

On va créer un pseudo projet sous Git. Il sera composé d'un simple fichier texte. Par soucis de simplicité, on va scruter le repository git local. La commande 'push' ne sera donc pas nécessaire.

- Créer un répertoire et créer un fichier texte à l'intérieur
- Créer un repository git à partir de ce répertoire

```
cd [MON REPERTOIRE]
git init
git add [MON FICHIER]
git commit -m 'version initiale du projet'
```

- Créer un nouveau job
- Faire pointer le job sur ce repository

```
La syntaxe est : file:///[Chemin vers le répertoire git]
```

Lorsque le chemin est correct, le message d'erreur s'efface dans la configuration du job.

1. Programmer la scrutation du repository toutes les minutes
2. Regarder si le job est exécuté et quand le git a été interrogé pour la dernière fois (log de scrutation)
3. Modifier le fichier du projet et faire un simple commit
4. Vérifier que le build se relance

Création d'un projet Maven

On va utiliser un projet sous git pour montrer l'interaction avec le gestionnaire de configuration. On pointera sur le repository git local pour des raisons de simplicité. La configuration classique est de pointer sur un repository partagé.

Configuration Maven dans Jenkins

Objectif: Configuration de Maven

1. Installer le plugin : 'Maven Integration'
2. Aller dans « Administration Jenkins » / « Configuration globale des outils »
3. Aller dans la zone 'Maven' (et pas 'Configuration Maven')
   a. Nom : Maven
   b. Décoché installation (maven est normalement déjà installé)
   c. Mettre dans MAVEN_HOME : /usr/share/maven

Création du job

Objectif: Construire un projet Maven

1. Créer un projet git au choix :

   Utiliser un projet personnel (projet, tp) utilisant Maven et compilant (de préférence avec des tests)

   ou cloner un repository existant : git clone https://github.com/sfauvel/cours_ic.git
2. Faire un nouveau job de « Construire un projet Maven » (nouvelle option disponible)
3. Définir le planning de déclenchement toutes les minutes
4. Attendre la première exécution et aller voir dans les logs les étapes réalisées
5. Aller vérifier la présence du .jar dans le « dernier build stable » ou « modules »/ « nom « projet »

Les tests

Objectif: Détection d'un problème dans le code

1. Modifier le code de la classe « Inn » pour introduire un bug et faire échouer les tests
    a. modifier une valeur ou une condition
    b. faire un commit
2. Aller voir le rapport détaillé des tests et les logs de la console
3. Corriger le code et commiter
4. Vérifier l'état du build, regarder le graph des résultats de tests

Ajouter des actions sur le projet

Objectif: Mise en place d'étapes annexes

Analyse de code avec Checkstyle

1. Installer le plugin 'Checkstyle'
2. Dans le job, ajouter le goal 'checkstyle:checkstyle' à maven (spécifier en plus 'install' qui est le goal par défaut lorsqu'il n'y en a pas de précisé)
3. Dans 'Configuration du build', cocher 'Publier les résultats Checkstyle'
4. Relancer le job et aller voir les résultats ('Résultats Checkstyle' dans le menu) et les défauts constatés

Publication de la javadoc

Le plugin Javadoc a du être installé avec l'installation du plugin Maven Integration Plugin.

Pour des raisons de sécurité, l'utilisation des frames de la javadoc de sont pas autorisés par défaut. Relancer Jenkins avec l'option suivante:

```
java -Dhudson.model.DirectoryBrowserSupport.CSP="'none'; img-src 'self'; style-src
'self'; child-src 'self'; frame-src 'self';" -jar jenkins.war --httpPort=8081
```

1. Ajouter dans les directives « Goals et option » de maven, le goals : javadoc:javadoc
2. Démarrer une construction manuellement
3. La javadoc est disponible sur l'interface

Construction d'une release note

Objectif: Produire une release note à partir des messages de commit

Pour lister les différents commit, on va utiliser la commande `git log`.

On va utiliser l'option `--pretty=format:'<FORMAT>'` pour configurer le format de sortie avec la date et le message ('Author date' et 'Subject').

En ajoutant l'option `--date=format:'<DATE FORMAT>'` on va configurer le format de la date.

Voir: https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History

1. Ajouter un script shell qui construit un fichier contenant les logs git.

Archiver des artifacts

Objectif: Conserver les derniers build stable

Dans les "Actions à la suite du build", on va "ajouter une action à la suite du build" en choisissant "Archiver des artefacts".

On va spécifier le fichier de releasenote (donner le nom du fichier des logs Git) ainsi que les .jar générés (utiliser une expression régulière).

Construction d'une livraison

Objectif: Préparer une livraison

1. Contruire un .zip contenant la release note et le .jar.
2. Archiver le .zip construit.
3. Récupérer le fichier et vérifier son contenu.

Organisation des builds

Objectif: Optimiser le temps d'exécution des builds

On va créer des builds pour différent usage

1. Créer un job nommé « FastBuild »
    a. il ne contient que la compilation et les tests
    b. il est exécuté toutes les minutes
2. Renommer le job précédent en « FullBuild »
    a. Modifier la fréquence pour ne démarrer que toutes les 15 minutes

Mise en place des pipelines

Objectif: Scripter les étapes d'intégration continues

1. Installer le plugin 'Blue Ocean'
2. Créer un job de type Pipeline
3. Dans Pipeline, choisir comme définition: Pipeline script from SCM
4. Choisir le SCM: Git
5. Renseigner le répertoire Git du projet
6. Créer un fichier Jenkinsfile à la racine du projet. Ce fichier contiendra le script jenkins.

```
pipeline {
    agent any
    stages{
        stage('Checkout') {
          steps {
              echo "Récupération des sources"
          }
        }
        stage('Build and test') {
          steps {
              echo "Compilation et tests"
          }
        }
        stage('Post Build') {
            steps {
                parallel(
                  javadoc: {
                    echo "Génération de la documentation"
                  },
                  checkstyle: {
                    echo "Analyse de code"
                  }
                )
            }
        }
    }
}
```

7. Lancer un build du job
8. Aller voir dans 'Blue Ocean' le déroulement du job

Objectif: Identifier une erreur sur le pipeline

1. Insérer une erreur dans le script (par exemple, en mettant 'xxxx' à la place de 'echo' dans la partie checkstyle)

2. Commiter le fichier
3. Relancer le Job
4. Aller voir le pipeline et identifier l'étape en échec

Objectif: Executer des commandes Maven dans le Pipeline

1. Modifier le Jenkinsfile pour construire le projet

```
pipeline {
    agent any
    tools {
        maven 'Maven'
    }
    stages{
        stage('Build and test') {
          steps {
              echo "Compilation et tests"
              sh "mvn install"
          }
        }
        stage('Post Build') {
            steps {
                parallel(
                  javadoc: {
                    echo "Génération de la documentation"
                    sh "mvn javadoc:javadoc"
                    publishHTML target: [
                        reportDir: 'target/site/apidocs',
                        reportFiles: 'index.html',
                        reportName: 'Javadoc'
                      ]
                  },
                  checkstyle: {
                    echo "Analyse de code"
                    sh "mvn checkstyle:checkstyle"
                    publishHTML target: [
                        reportDir: 'target/site',
                        reportFiles: 'checkstyle.html',
                        reportName: 'Checkstyle'
                      ]
                  }
                )
            }
        }
    }
}
```

ANNEXES

Ces étapes ne sont pas réalisable sur les environnements à dispositon (manque de ressources ou d'autorisations)

Notification

Pour pouvoir envoyer des mails, il faut configurer le serveur SMTP.

1. Aller dans « Administrer Jenkins » / « configurer le système »
2. Renseigner le serveur SMTP pour l'envoi de mail.
3. Faire un test en envoyant un mail.
4. Dans le job, ajouter un destinataire en cas d'échec
5. Provoquer un nouveau build en échec (erreur de compilation ou de test)
6. Vérifier la réception du mail
7. Corriger le problème pour recevoir une notification de retour à la normale.

Analyse de code avec Sonar

Installation

1. Installer Sonar en récupérant le .zip sous /usr/local/opt
2. Aller dans «~/ Documents »
3. Faire « unzip /usr/local/opt/sonarqube-5.1.zip »
4. Aller dans le répertoire sonarqube-5.1/bin/linux-x86-64
5. Exécuter : . ./sonar.sh console
6. Aller à l'adresse : http://localhost:9000

Configuration de l'analyse

1. Configurer le job en ajoutant une action à la suite du build : SonarQube
2. Démarrer une construction manuellement
3. L'analyse Sonar est exécutée à la fin de la construction
    a. Aller voir des les logs l'analyse Sonar
    b. Accéder au rapport Sonar à travers Jenkins