Ateliers
« Groovy pour les pipelines Jenkins»

Pré-requis :

- Bonne connexion Internet
- JDK17+
- Eclipse
- Docker
- Git

Table des matières

Atelier 1 : Mise en place

1.1 Installation Groovy

Choisir une installation en fonction de votre OS. (recommandé SDK)

Créer un fichier *HelloWorld.groovy* contenant la ligne suivante :

```
println "Hello, ${this.args[0]}, the time is ${new Date()}"
```

Tester l'installation en utilisant les différents outils

- Exécuteur de script
- CLI interactif
- Console

1.2 Mise en place Eclipse et plugin Groovy

Installer le plugin Groovy dans l'IDE Eclipse

Voir : *https://github.com/groovy/groovy-eclipse/wiki#how-to-install*

Test de l'IDE

- Création de projet *Groovy :*
  New → Project → Groovy Project
- Création de script :
  New → Other → Groovy → Groovy Type
  Script → Hello
- Éditeur et complétion
- Exécution
  Run As Groovy Script, Java Application

Atelier 2 : Types de base et collections

Reprendre le script ***Indexer.groovy*** et le placer dans un package ***org.formation***

1. Écrire un script qui à partir d'un texte donné, remplit une *Map* avec comme clé les mots du texte et comme valeur leur nombre d'occurrences.

2. Améliorer le programme pour ne pas prendre en compte les mots de 2 caractères ou moins

3. Trier la map par les mots-clés

4. Construire la map inversé : la clé étant le nombre d'occurrence, les valeurs les listes de mots

5. Créer un intervalle avec les valeurs possibles des occurrences. Boucler sur l'intervalle pour afficher les mots clés de cette occurrence

Atelier 3 : POO - Classes, Scripts

3.1 Organisation en classes et packages

Nous réorganisons la logique du TP précédent avec des classes, des packages et un script principal.

Le projet contiendra donc :

- Une classe ***Index*** (package *model*) qui encapsule les données d'un index :
  - Une source (contenu source associé à l'index)
  - Dates de création et d'indexation
  - l'index : Une map contenant les mots et leur nombre d'occurrence
- Une classe ***Indexer*** (package *service*) qui permettra de déclencher l'indexation d'un index. Avec comme propriétés de configuration :
  - Un *tokenizer* : le délimiteur de mots-clé
  - *minimalSize* ; La taille minimale des mots à mettre dans l'index

  Et comme méthode
  - *Index buildIndex(Object source)* qui déclenche l'indexation
- Une classe ***MainScript*** utilisant les classes précédentes

Atelier 4 : Closure, contexte et délégué

4.1 Utilisation des closures

Nous transformons notre classe *Indexer*.

Un indexeur est désormais modélisé comme :

- Un *Tokenizer* responsable de découper un texte en un ensemble de mots
- Une liste de filtres : Chaque filtre traite une collection de mots et retourne une autre collection de mots. Les filtres sont donc traités comme des closures prenant en entrée une liste de *String* et retournant une autre liste de *String*

Transformer la classe *Indexer* dans ce sens et réécrire la méthode qui renseigne un index.

Dans la classe Main, créer un Index et lui ajouter 3 filtres :

- Un filtre transformants la liste de mots en liste de mots en minuscule
- Un filtre supprimant les mots inférieurs à 3 caractères
- Un filtre supprimant les noms propres Delhi, Londres, Pékin, Chine

4.2 Classe déléguée

Afficher dans une Closure :

- La stratégie de résolution
- La classe propriétaire
- La classe délégué
- L'attribut thisObject

Faire en sorte que la classe déléguée pour chaque Closure soit l'indexer et que chaque Closure accède à un attribut de l'indexer.

Atelier 5 : Compléments

5.1 Annotations et I/O

Annoter la classe Indexer avec @Log et utiliser la variable log pour afficher des messages

Modifier le code précédent afin de parcourir un répertoire à la recherche de fichiers *.txt* et de créer des index pour chaque fichier

5.2 Intégration

Récupérer le programme Java fourni et l'importer dans votre IDE.
Consulter les 2 classes principales :

- MainEval
- MainGroovyShell

Exécuter les

Ateliers 6: Mise en place Jenkins

Objectifs

- Prendre en main la distribution de Jenkins
- Avoir un aperçu des configurations administrateur
- Concrétiser l'architecture Maître/Esclave de Jenkins

6.1 Script de démarrage

Télécharger la distribution générique de Jenkins : https://www.jenkins.io/download/

Écrire un script de démarrage qui positionne en variables d'environnement :

- JAVA_HOME
- JENKINS_HOME
- la mémoire de la JVM
- Éventuellement, le port d'écoute de jenkins
- La propriété *hudson.plugins.git.GitSCM.ALLOW_LOCAL_CHECKOUT* permettant de travailler avec des dépôts localux

Voici un exemple d'un tel script :

```
#!/bin/sh
export JAVA_HOME=/usr/lib/jvm/java-1.17.0-openjdk-amd64
export JENKINS_BASE=/home/dthibau/Formations/Jenkins/MyWork/jenkins
export JENKINS_HOME=${JENKINS_BASE}/.jenkins

java -Dhudson.plugins.git.GitSCM.ALLOW_LOCAL_CHECKOUT=true -Xmx2048m \
-jar ${JENKINS_BASE}/jenkins.war --httpPort=8082
```

Faire un premier démarrage, se connecter à l'adresse d'écoute et compléter l'installation en :

- Installant les plugins recommandés
- Définissant un login/password administrateur (*admin/admin* par exemple)

Visualiser ensuite l'arborescence de JENKINS_HOME

6.2 Configuration générale et outils

6.2.1 Configuration système : l'exemple du serveur de mail

Si nécessaire installer le plugin ***mailer***

Ensuite aller dans la configuration système :

« Administrer Jenkins → Configurer le système»

Renseigner les paramètres du serveur smtp

login : stageojen@plbformation.com

password : stageojen

smtp : smtp.plbformation.com (port smtp 587)

Tester l’envoi de mail

6.2.2 Configuration des outils (JDK, Maven, Git)

Aller dans la configuration globale des outils :
« Administrer Jenkins → Configuration Globale des outils »

- Dans la section JDK, indiquer un JAVA_HOME pré-installé
- Dans la section Maven, utiliser l’installation automatique
- Dans la section Git, indiquer l’installation par défaut

Ateliers 7 : Pipelines, syntaxe déclarative, script et librairies

Objectifs

- Distinguer et prendre en main les 2 syntaxes
- Mettre au point son environnement de développement de JenkinsFile
- Utiliser les multi-branches
- Visualiser les pipelines avec le plugin *BlueOcean*

7.1 Déclaratif / Script

Installer le plugin ***Blue Ocean***

Créer un job Pipeline « *First_Pipeline* »

Dans le champ d'édition de la pipeline, choisir un exemple avec la syntaxe déclarative

Modifier le script afin qu'il effectue les tâches suivantes :

- Checkout du dépôt des sources de l'atelier précédent
- Définition de l'outil Maven défini dans notre installation de Jenkins
- Exécution de la cible Maven ***clean package***
- Étape de Post-build de Publication des résultats de tests
- Archivage des artefacts

Modifier ensuite l'exemple de la version script et effectuer les mêmes opérations

7.2 Jobs multibranches et implémentation déclarative

7.2.1 Multibranch Job

Pour la mise en place de la pipeline, vous pouvez :

- Utiliser la fonction Replay
- Installer l'extension Jenkins ***Pipeline Linter Connector*** dans Vscode qui permet de valider le JenkinsFile avant de committer

Créer la branche ***dev*** dans votre repository GIT :

```
git checkout -b dev
```

Récupérer le fichier ***Jenkinsfile*** fourni et le committer dans la branche *dev*

Créer un job multi-branch pipeline ***multi-module*** pointant sur le dépôt dans Jenkins

Observer l'exécution automatique des JenkinsFile dans la branche dev

7.2.2 Implémentation déclarative

Pré-requis : Démarrage d'un serveur SonarQube et intégration Maven

```
docker run -d --name sonarqube -p 9000:9000 sonarqube
```

- Se connecter à localhost:9000 avec admin/admin
- Changer le mot de passe
- Dans le Menu en haut à droit ***My Account → Security***
- Générer un nouveau jeton,
- Le déclarer dans Jenkins sous forme de crédentiels :
  - Type « Secret Text »

Le fichier *JenkinsFile* déclare 3 phases ou stage.

Dans la première phase « Build and Tests » effectuer la commande maven

```
./mvnw -Dmaven.test.failure.ignore=true clean package
```

Puis dans les étapes de post-build :

- Publier toujours les résultats des tests
- En cas de succès : Archiver les artefacts
- En cas d'erreur : Envoyer un mail

Dans la seconde phase, effectuer en parallèle :

- Une tâche exécutant une analyse des dépendances du projet :

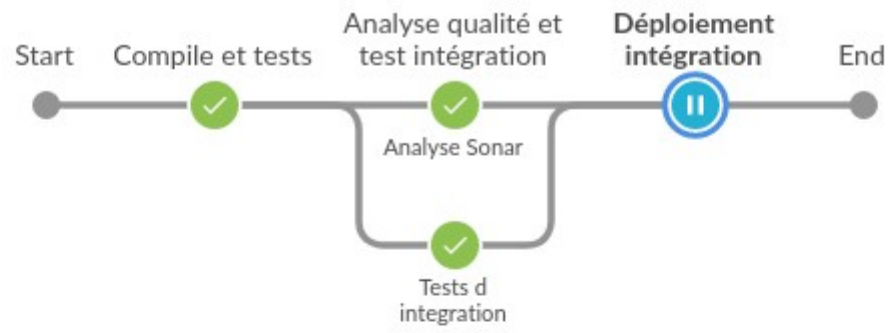
```
./mvnw -DskipTests verify
```

- Une tâche exécutant une analyse SonarQube

```
./mvnw -Dsonar.login=${SONAR_TOKEN} clean integration-test sonar:sonar
```

Dans la dernière phase :

- Poser une question à l'utilisateur :
  Dans quel Data Center, voulez-vous déployer l'artefact ?
- Lui proposer 3 choix. Par exemple : Paris, Lille, Lyon
- Récupérer le paramètre saisi et l'utiliser pour déployer l'artefact de la première phase dans un répertoire dynamique
- Finalement faire en sorte que cette phase ne s'exécute que sur la branche principale



7.3 Bloc script

Installer le plugin ***Pipeline Utility Steps***.

7.3.1 Construction boucle et variable globale

Nous voulons maintenant déployer vers plusieurs dataCenters.

La liste des datacenters sera fournie dans un fichier projet au format JSON.

Veillez également à que l'étape en attente d'approbation n'utilise pas d'agent.

Dans la phase de déploiement, effectuer une boucle sur la liste des datacenters et effectuer un déploiement sur chaque

7.3.2 Fonction et appel API

Visualiser la fonction Groovy fournie dans le fichier ***wait4Quality.txt***.

La comprendre, la déclarer dans la pipeline et l'utiliser à bon escient

7.4 Librairies

Référence :

https://dev.to/jalogut/centralise-jenkins-pipelines-configuration-using-shared-libraries

7.4.1 Tutoriel

Créer un nouveau dépôt Git nommé ***GlobalLib***

Y créer un répertoire ***vars*** et y déposer le fichier ***standardPipeline.groovy*** fourni.

Committer

Configuration Jenkins

Dans Jenkins, Configurer une Global Pipeline Library pointant sur le précédent dépôt.

Utilisation

Créer ensuite un *Jenkinsfile* utilisant la librairie définie dans un nouveau projet.

7.4.2 Ajout de fonctions Groovy

En vous inspirant du tutoriel précédent, créer une nouvelle fonctions dans la librairie partagée ***tarGz.groovy*** . La fonction récupérera dans sa configuration :

- Un sous-répertoire du projet
- Une liste de suffixe de fichiers
- Un répertoire de sortie

Elle produira un tar qui contiendra tous les fichiers respectant les extensions

```
// my-shared-library/vars/createTarGz.groovy

def call(Map config) {
    def sourceDir = config.sourceDir
    def extensions = config.extensions
    def outputDir = config.outputDir

    echo "Création d'une distribution tar.gz à partir de $sourceDir avec les
extensions : ${extensions.join(', ')}"

    // Créez une chaîne de filtres pour rechercher plusieurs extensions
    def extensionFilter = extensions.collect { ext -> "--include=\"*.
$ext\"" }.join(' ')

    // Utilisez des commandes shell pour créer le tar.gz
    sh "mkdir -p $outputDir"
```

```
    sh "tar -czvf $outputDir/my_distribution.tar.gz -C $sourceDir
$extensionFilter ."
}
```

Utilisez la fonction dans la pipeline