

Cahier de TPs

Gradle

Pré-requis :

Poste développeur avec accès réseau Internet libre

Linux, Windows 10, Mac OS

Pré-installation de :

- Git
- JDK11+
- IDE : Eclipse , IntelliJ Idea, VSCode
- Docker

TP1 : Démarrage

1.1 Installation

En fonction de votre système d'exploitation, installer Gradle soit via le gestionnaire de paquets, soit manuellement

1.2 HelloWorld

Écrire la tâche *HelloWorld* comme décrite dans le support.

Exécuter la en mode quiet ou non

Effectuer un ***gradle tasks***

Le démon est-il démarré ?

1.3 Thérapie de groupe

Récupérer le script fourni, le comprendre

Exécuter la tâche *groupTherapy*

Exécuter les tâches *yayGradle0* et *groupTherapy* ; des changements ?

Afficher les tâches disponibles du projet

Utiliser une abréviation en « camelCase »

Exclure la tâche *yayGradle0* lors de l'exécution de *groupTherapy*

TP2 : Groovy, String, Maps, Regexp

Installation Groovy

Voir : <https://groovy-lang.org/install.html>

Installation Groovy dans l'IDE

<https://groovy-lang.org/ides.html>

Utilisation String, Maps, Regexp

Reprendre le script ***Indexer.groovy*** et compléter le code.

Tester au fur et à mesure en appelant :
`groovy Indexer.groovy`

TP3 : Concepts : Projets, Tasks, Properties

3.1 Propriétés projet, tâches

Écrire une tâche **printProject** qui :

- Affiche sur la console le nom et la description du projet

Ajoute la propriété **profile** au projet, affecter et afficher cette nouvelle propriété projet

Externaliser la propriété *profile* dans *gradle.properties*, surcharger la via la ligne de commande

3.2 Code de configuration

Faire en sorte que la tâche *printProject* soit listée par la commande *gradle tasks* (un groupe doit être défini)

Définir dans *build.gradle* une classe *ProjectVersion* composée de 3 champs :

- *majorNumber* (Integer)
- *minorNumber* (Integer)
- *classifier* (String)

Écrire une tâche de configuration qui à partir d'un fichier *version.properties* à la racine du projet instancie un objet *ProjectVersion* et le stocke dans la propriété **version** du projet.

Compléter la tâche **printProject** qui affiche l'instance de l'objet *ProjectVersion*

Exécuter la tâche *printProject* avec les différents niveau de log (quiet, info, debug)

3.3 Graphe de tâches

Définir 3 nouvelles propriétés projet :

- **srcDirectory** : L'emplacement des sources
- **outputDirectory** : Emplacement des fichiers « compilés »
- **distDirectory** : Emplacement pour la distribution

Définir 4 nouvelles tâches à partir des types prédéfinis de Gradle :

- **clean** : Suppression du répertoire *outputDirectory*
- **copyVersion** : Copie du fichier *version.properties* dans *outputDirectory*
- **compile** : Copie des fichiers *.c présents dans *srcDirectory* vers *outputDirectory* en les renommant en *.o
- **createDistribution** : Création d'une archive à partir des fichiers présents dans *outputDirectory*

createDistribution dépend de *compile* qui dépend de *copyVersion*.

Tester les différentes tâches et visualiser les effets du cache de Gradle

Input property

Écrire une tâche **makeRelease** qui définit :

- une propriété d'entrée classifier initialisée à *project.version.classifier*
- en sortie le fichier **version.properties**

L'action effectuée consiste à positionner le *classifier* dans *version.properties* à **final**

Exécuter une première fois l'action et vérifier que le fichier properties a bien été mis à jour.

Exécuter une seconde fois et vérifier que la tâche est « UP-TO-DATE »

Dépendance implicite

Créer une tâche **deploy** de type *Copy* qui prend en entrée la sortie de *createDistribution* et la copie en dehors de l'arborescence projet.

Observer la dépendance implicite en exécutant **gradle -i deploy**

3.4 Tâche custom et **buildSrc**

Tâche custom

Définir une tâche personnalisée *UpdateVersionTask* qui prend en entrée :

- En entrée un objet *ProjectVersion* (l'objet doit être *Serializable*)
- En sortie un fichier

qui effectue une action qui modifie les différentes clés du fichier avec les différentes propriétés de *ProjectVersion*

Utiliser ce type de tâche dans *makeRelease*

Tâche finale

Écrire une tâche **Release** qui produit la séquence suivante :

- *clean*
- Change le classifier de *version.properties* à *Final*
- *deploy*
- Change les clés du fichier *version.properties* en incrémentant la version mineure et en positionnant SNAPSHOT dans le classifier

Utilisation de **buildSrc**

Réorganiser le code en utilisant le répertoire **buildSrc** :

Utiliser l'arborescence **src/main/groovy** et un package **org.formation**

- y placer la classe *ProjectVersion*
- y placer la tâche custom *updateVersionTask*

Importer le package *org.formation* dans *build.gradle*

3.5 Hooks

Écrire un hook qui lorsque le graphe est créé, vérifie que si le graphe contient la tâche ***makeRelease*** la propriété *profile* est égale à ***prod***. Lancer une exception sinon

TP4 : Java et C++

4.1 Plugin *init* pour Java

Utiliser le plugin *init* pour créer une application de type **basic**.

- Observer les fichiers générés
- Visualiser les tâches disponibles

Appliquer le plugin java et visualiser les tâches disponibles, les task rules

Utiliser le plugin *init* pour créer une application Java.

- Observer les fichiers générés
- Visualiser les tâches disponibles
- Fixer la version de Java correspondant à votre environnement et créer une distribution
- Exécuter l'application Java générée

4.2 Dépendances et configurations

Créer un nouveau répertoire de travail et reprendre les sources fournis. C'est une application web Java monolithique autonome qui utilise le framework *SpringBoot*.

Le framework fournit 2 plugins Gradle:

- ***org.springframework.boot*** : Il ajoute principalement une tâche ***bootRun*** permettant de générer un exécutable qui lance l'application web.
- ***io.spring.dependency-management*** : Il permet de gérer les versions de toutes les dépendances utilisées par le framework (Équivalent à la balise *dependencyManagement* de Maven)

Appliquer ces 2 plugins en indiquant la version **2.7.8** pour *org.springframework.boot* et **1.0.15.RELEASE** pour *io.spring.dependency-management*

Spécifier les propriétés groupe et version pour le projet ainsi que la propriété *sourceCompatibility*

Déclarer ensuite le repository Maven et les dépendances suivantes :

- Dépendances du framework :
 - Pré-processing des annotations :
 - *org.springframework.boot:spring-boot-configuration-processor*
 - Nécessaires pour la compilation :
 - *org.springframework.boot:spring-boot-starter-validation*
 - *org.springframework.boot:spring-boot-starter-data-jpa*
 - *org.springframework.boot:spring-boot-starter-thymeleaf*

- `org.springframework.boot:spring-boot-starter-web`
- `org.springframework.boot:spring-boot-starter-security`
- Test
 - `org.springframework.boot:spring-boot-starter-test`
 - `org.springframework.security:spring-security-test`
 - `net.sourceforge.htmlunit:htmlunit`
- Exécution uniquement :
 - `org.springframework.boot:spring-boot-starter-actuator`
 - `org.springframework.boot:spring-boot-devtools`
 - `org.hsqldb:hsqldb`
- Dépendances non gérées par le framework
 - Pour la compilation :
 - `io.jsonwebtoken:jjwt` (Trouver une version récente sur Maven Central)
 - `io.springfox:springfox-swagger2 :2.9.2`
 - Pour l'exécution
 - `io.springfox:springfox-swagger-ui :2.9.2`
 - `org.webjars:bootstrap` (Indiquer une version 4.x)

Une fois indiquées les dépendances, exécuter successivement les tâches *compileJava*, *test*, puis *bootRun*

Exécuter la tâche générant la distribution

4.3 Init pour projets C++

Créer un répertoire de travail et y exécuter *gradle init*

Suivre l'assistant pour démarrer un projet de type Application C++

Visualiser les fichiers générés :

- Le wrapper
- *settings.gradle*
- *build.gradle* : plugins utilisés et machine cible
- la structure du projet

Visualiser les tâches disponibles et Exécuter un build

Refaire la même chose pour une librairie C++

4.4 Projet Librairie C++

Récupérer les sources fournis

Mettre au point le fichier Gradle pour appliquer les plugins :

- *cpp-library*

- *cpp-unit-test*
- *maven-publish*

Définit plusieurs machines cibles pour cette librairie.

Spécifier une propriété groupe et version pour le projet

Les fichiers de test dépendent de la librairie :

org.gradle.cpp-samples:googletest:1.9.0-gr4-SNAPSHOT

présent dans le dépôt Maven :

<https://repo.gradle.org/gradle/libs-snapshots-local/>

Indiquer cette dépendance dans *build.gradle*

Essayer d'exécuter la tâche *test*

Les tests utilisent la librairie *pthread* qui nécessite l'option *-lpthread* lors de la phase de link.

Configurer cette option pour les variantes correspondant à une machine Linux utilisant *gcc*

Définir un dépôt Maven local et publier vers ce dépôt, regarder les méta-données associées

TP5 : Multi-projets

5.1 Mise en place de la structure projet

Créer un projet de type *basic*

Reprendre les sources fournis et les dézipper dans le répertoire parent

Le projet comprend 2 modules :

- Un module librairie qui contient du code utilitaire
- Une module application application monolithique qui dépend du module précédent

Inclure les 2 modules dans le projet parent

Exécuter `./gradlew projects`

Appliquer les plugins :

- `java`
- `org.springframework.boot`
- `io.spring.dependency-management`

Exécuter `./gradlew task`

Par défaut, le plugin `org.springframework.boot` crée une application exécutable, ce qui n'est pas souhaitable pour le module *library*

Pour le désactiver et activer la création de jar nécessaire à la dépendance :

```
bootJar {
    enabled = false
}
jar {
    enabled = true
}
```

5.2 Gestion des dépendances

Les 2 projets utilisent mavenCentral

Les 2 projets utilisent le framework SpringBoot version 2.1.6.RELEASE

Les 2 projets ont comme dépendances :

- `org.springframework.boot :spring-boot-starter-test` dans la configuration *test*

Le projet *library* a comme dépendance :

- *org.springframework.boot :spring-boot-starter* dans la configuration *implementation*

Le projet *application* a comme dépendance :

- Le module *library*
- *org.springframework.boot :spring-boot-starter-web*
- *org.springframework.boot :spring-boot-starter-actuator*

Une fois après avoir indiqué ces dépendances, construire :

- Le module *library*
- Le module *application*
- L'intégralité du projet

Exécuter l'application en exécutant la tâche *bootRun* sur le module *application*

5.3 Projet C++

Récupérer les sources fournies qui est un projet multi-module contenant une application dépendante du modules *utilities* qui dépend du module *list*

Ecrire les fichiers *gradle* pour construire ce projet

TP6 : Pipeline

Ce projet utilise le projet Java du TP4

6.1 Tests unitaires

S'assurer d'utiliser JUnit4 pour les tests unitaires

Exécuter les tests et visualiser les rapports de tests

Modifier la configuration du logging

- Pour afficher la sortie standard
- Pour afficher les événements tests

6.2 Tests d'intégration

Définir un nouveau *sourceSet* `src/intTest/java`

Y déplacer la classe de test *MemberRestControllerIntegrationTest.java* et s'assurer que les tests d'intégration s'exécutent lors de *gradle check*

6.3 Jacoco et Sonar

Appliquer le plugin *jacoco*

- Visualiser les tâches

Configurer afin que la tâche *jacocoTestReport* s'exécute systématiquement après *test*

Déclarer une nouvelle tâche *jacocoIntTestReport* qui génère un rapport pour les tests d'intégration

Installer Sonar ou démarrer le par docker

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

Appliquer le plugin *org.sonarqube :3.3* et configurer le projet

6.4 Publishing

Installer ou démarrer Nexus via docker

Configurer le DSL publications et repositories afin de permettre des publications vers le dépôt snapshot ou release de Nexus

6.5 Intégration Jenkins

Installer Jenkins mettre en place une pipeline en s'inspirant du fichier *Jenkinsfile* fourni