Chapitre 4
Scala Build Tool

1. Définition

sbt (*Scala Build Tool*) est l'outil de gestion et de construction de projet le plus utilisé en Scala, utilisable également en Java.

Grâce au fichier `build.sbt`, on peut configurer son projet et le compiler de façon incrémentale grâce aux informations extraites du compilateur. Il propose de nombreuses fonctionnalités comme le packaging, la publication ou encore le lancement des tests d'un projet.

En plus des fonctionnalités classiques, sbt propose également de construire son projet sur de multiples versions de Scala.

2. Installation

Avant de pouvoir installer sbt, un JDK est nécessaire. Le site officiel recommande d'installer la version 8 ou 11. Pour installer sbt, on peut télécharger le .zip ou le .tgz disponibles sur la page officielle puis suivre les étapes d'installation.

Il est préférable d'opter pour un gestionnaire de paquets propre à son système d'exploitation pour pouvoir gérer les versions plus efficacement.

2.1 Sur Mac/Linux

2.1.1 Homebrew

brew est un utilitaire de paquets disponible sur Linux et Mac. Il permet d'installer les paquets disponibles dans sa librairie avec cette commande :

```
$ brew install sbt
```

Remarque

On ne peut pas récupérer simplement une version spécifique d'un programme ; brew installe toujours la dernière version d'un programme à condition que le code Github de Homebrew ait été mis à jour.

Les programmes sont installés directement dans le dossier d'installation de brew et des liens symboliques vers le dossier `/usr/local` sont créés.

```
$ find / -type d -name sbt
/usr/local/Cellar/sbt

$ ls -l /usr/local/bin
sbt -> ../Cellar/sbt/1.3.13/bin/sbt
```

2.1.2 SDKMAN!

SDKMAN! est un utilitaire de paquets disponible sur Linux, Mac et Windows. Il permet d'installer les paquets disponibles dans sa librairie avec cette commande :

```
$ sdk install sbt
```

Les programmes sont installés dans le dossier d'installation de SDKMAN! mais contrairement à brew, aucun lien symbolique n'est généré.

```
$ find / -type d -name sbt
/.sdkman/candidates/
```

Remarque

Lors de l'installation, si une autre version du programme existe, un message s'affiche pour demander si vous souhaitez que cette version devienne la version par défaut.

Pour installer une version donnée d'un programme, il suffit de l'ajouter à la fin de la commande.

```
$ sdk install sbt 1.3.0
```

2.2 Sur Windows

2.2.1 Scoop

Scoop détecte automatiquement la version du programme à installer selon votre système d'exploitation (32 ou 64 bits). Pendant l'installation, il n'y aura pas de pop-up pour demander une autorisation quelconque.

```
$ scoop install sbt
```

Pour installer une version donnée d'un programme, il suffit d'ajouter @ suivi de la version souhaitée au nom du programme.

```
$ scoop install sbt@1.3.0
```

2.2.2 Chocolatey

Chocolatey possède la plus grande librairie en ligne de paquets Windows. En plus de proposer une commande en ligne de code, il met à disposition une page web recensant tous les paquets disponibles.

```
$ choco install sbt
```

Pour installer une version donnée d'un programme, il suffit d'ajouter l'option `--version` suivie de la version souhaitée.

```
$ choco install sbt --version 1.3.0
```

3. Création d'une application

3.1 Architecture du projet

sbt utilise une architecture similaire à Maven pour les fichiers source en remplaçant le dossier java par le dossier scala.

```
build.sbt
project/
src/
-- main/
  |-- resources/
  |-- scala/
|-- test/
  |-- resources/
  |-- scala/
target/
```

On retrouve des dossiers `src` et `test` pour les classes de production et de tests. L'équivalent du `pom.xml` est le fichier `build.sbt` contenant toutes les informations nécessaires pour compiler le projet. Le dossier `target` contient toutes les classes compilées et le dossier `project` des fichiers utilitaires.

Remarque

Si vous travaillez dans un projet git, il est conseillé d'inclure le dossier `target/` dans le `.gitignore`.

3.2 Définition de construction

3.2.1 Définition de la version de sbt

Tout d'abord, pour déterminer quelle version de sbt utiliser pour compiler le projet, il faut la spécifier dans un fichier de configuration. Pour cela, on inscrit dans le fichier `build.properties` présent dans le dossier `project` les informations suivantes :

```
sbt.version=1.3.13
```

Si la version spécifiée n'est pas présente sur le poste, elle sera téléchargée par le lanceur sbt.

Remarque

Si le fichier `build.properties` n'existe pas, l'application pourra tout de même être lancée mais dans une version arbitraire. Il est donc recommandé de toujours spécifier ce fichier pour éviter des erreurs de compilation et pour s'assurer d'avoir un même environnement de travail sur tous les postes.

3.2.2 Fichier build.sbt

Avec sbt, la définition de construction est définie dans le fichier `build.sbt` et correspond à la définition d'un ou plusieurs projets de type `Project`.

La définition d'un projet se fait en spécifiant le dossier dans lequel se trouvent les sources, par défaut le dossier courant et trois informations par défaut : le nom et la version du projet et la version de Scala à utiliser. Chaque paramètre se définit avec une clé et une valeur séparées par l'opérateur :=.

Cela donne le résultat suivant :

```
lazy val projet = (project in file("."))
 .settings(
   name := "Projet Scala",
   scalaVersion := "2.12.7",
   version := "1.0.0"
 )
```

On peut également définir des valeurs par défaut pour tous les projets présents dans le répertoire courant en ajoutant un paramètre à la racine du fichier. Pour accentuer le fait que ce paramètre est commun à tous les projets, on peut ajouter le préfixe `ThisBuild /`.

Ainsi, si on définit un projet secondaire dans un dossier `secondaire` sans préciser de version, la version utilisée sera celle définie par défaut.

```
ThisBuild / scalaVersion := "2.13.3"

lazy val projet = (project in file("."))
 .settings(
   name := "Projet Scala",
   scalaVersion := "2.12.7",
```

```
    version := "1.0.0"
  )

lazy val projetSecondaire = (project in file("secondaire"))
  .settings(
    name := "Projet Scala Secondaire",
    version := "1.2.0"
  )
```

Cela générera un dossier `secondaire` qui compilera avec la version 2.13.3 de Scala.

Remarque

Pour le reste des explications, nous nous baserons sur un projet unique et tous ses paramètres seront déterminés à la racine du fichier.

3.3 Import de dépendances

3.3.1 Définition

Les librairies importées par un projet Scala sont définies dans la variable `libraryDependencies`. Cette variable étant une liste, pour ajouter une dépendance au projet, il suffit de rajouter un élément dans cette liste.

Une dépendance se définit avec trois éléments séparés par un `%` :

- `idGroup` : l'identifiant du groupe proposant la librairie.
- `idArtefact` : l'identifiant de l'artefact à importer.
- `revision` : la version souhaitée de la librairie.
- `configuration` : la configuration utilisée pour ajouter la librairie (paramètre optionnel).

Ainsi, pour ajouter la version 1.2.3 de la librairie *logback-classic* du groupe *ch.qos.logback*, il suffit d'ajouter la ligne suivante :

```
libraryDependencies += "commons-net" % "commons-net" % "3.6"
```

Pour obtenir la version de la librairie qui correspond à la version de Scala, on peut remplacer **%** par **%%** entre l'identifiant du groupe et l'identifiant de l'artefact. Ainsi, sbt ajoutera automatiquement la version de Scala au nom de l'artefact.

Par exemple, pour ajouter la version 2.1.0 de la librairie *emailaddress* du groupe *uk.gov.hmrc* dépendante de la version de Scala, il suffit d'ajouter la ligne suivante :

```
libraryDependencies += "uk.gov.hmrc" %% "emailaddress" % "2.1.0"
```

3.3.2 Dépendance de test

Par défaut, les dépendances sont ajoutées à la compilation des classes de production. Pour les ajouter aux classes de test, il faut ajouter la configuration à la fin de la ligne.

Par exemple, pour ajouter la version 5.0.0 de la librairie *scalatestplus-play* du groupe *org.scalatestplus.play* dépendante de la version de Scala, il suffit d'ajouter la ligne suivante :

```
libraryDependencies += "org.scalatestplus.play" %% "scalatestplus
play" % "5.0.0" % Test
```

3.3.3 Ajout d'un répertoire de librairies

Par défaut, sbt recherche les librairies dans le répertoire standard Maven2. Si vous essayez d'importer une librairie qui n'existe pas dans ce répertoire, vous obtiendrez une exception de type `ResolveException`.

Pour résoudre ce problème, il est possible d'ajouter de nouveaux répertoires. Ces derniers sont définis dans la variable `resolvers`.

Un répertoire est défini par son nom et son URL d'accès. Pour ajouter un nouveau répertoire, il suffit d'ajouter à la variable `resolvers` son nom et son URL séparé par le mot-clé `at`.

Par exemple, pour ajouter un répertoire Sonatype, il suffit d'ajouter la ligne suivante :

```
resolvers += "Sonatype OSS Snapshots" at
"https://oss.sonatype.org/content/repositories/snapshots"
```