
Gradle

Plan

Plan

Gradle ?

Un *moteur de production* logiciel

- logiciel libre, 2007, Licence Apache 2.0 ▸ gradle.org
- fonctionnalités similaires à Maven / Ant / Apache Ivy
- compilation / test / intégration continue / déploiement. . .
- gère des projets écrits en Java, Scala, Groovy, C++ et d'autres
- multi-projets : e.g. une application + ses librairies
- intégré aux IDE courants
- utilisé pour de très gros projets : Android, NETFLIX. . .

Gradle

Principales caractéristiques

- ⇒ principe *zéro-conf* : *convention plutôt que configuration*
- ⇒ gestion des dépendances et de leurs versions
- ⇒ accélère la compilation/exécution (cache/démon)
- ⇒ fait de manière à être étendu facilement : plugin

Installation

- Nécessite une JVM 8+
- présent dans tous les dépôts principaux Linux
- ▸ gradle.org/install
- ⇒ *Gradle user home* : `~/.gradle` (dep, tools, cache. . .)

Plan

Création d'un projet

Possible de créer un projet géré par Gradle via l'IDE, mais la ligne de commande reste plus flexible. > *gradle init* :

```
fab@boid ~/ztmp/demo
$ gradle init

Select type of project to generate:
  1: basic
  2: application
  3: library
  4: Gradle plugin
Enter selection (default: basic) [1..4] 2

Select implementation language:
  1: C++
  2: Groovy
  3: Java
  4: Kotlin
  5: Scala
  6: Swift
Enter selection (default: Java) [1..6]

Split functionality across multiple subprojects?:
  1: no - only one application project
  2: yes - application and library projects
Enter selection (default: no - only one application project) [1..2]

Select build script DSL:
  1: Groovy
  2: Kotlin
Enter selection (default: Groovy) [1..2]

Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no] yes

Select test framework:
  1: JUnit 4
  2: TestNG
  3: Spock
  4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4]

Project name (default: demo):

Source package (default: demo):

> Task :init
Get more help with your project: https://docs.gradle.org/7.5.1/samples/sample_building_java_applications.html
```

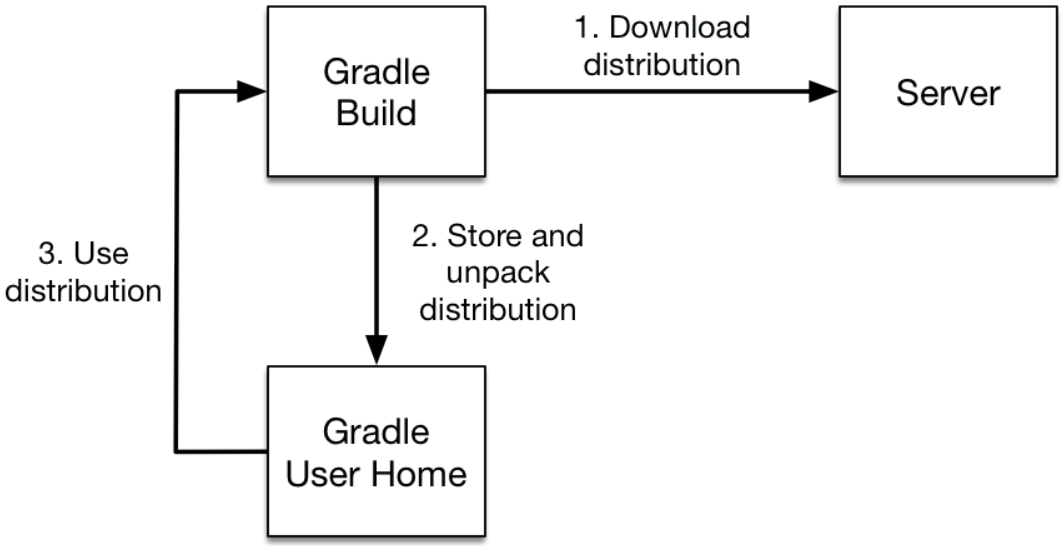
Groovy Kotlin

```
├── gradle (1)
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew (2)
├── gradlew.bat (2)
├── settings.gradle (3)
└── app
    ├── build.gradle (4)
    └── src
        ├── main
        │   └── java (5)
        │       └── demo
        │           └── App.java
        └── test
            └── java (6)
                └── demo
                    └── AppTest.java
```

1. Generated folder for wrapper files
2. Gradle wrapper start scripts
3. Settings file to define build name and subprojects
4. Build script of `app` project
5. Default Java source folder
6. Default Java test source folder

Gradle Wrapper

Uutilise la version spécifiée par le projet pour les build ▸ wrapper



- Le programme `gradlew` se trouve à la racine
- La bonne version sera téléchargée sur l'hôte si besoin
- ⇒ à la racine : `./gradlew <task>`
- ⇒ dans un sous-projet : `../gradlew <task>`

settings.gradle

Groovy | Kotlin

settings.gradle

```
rootProject.name = 'demo'
include('app')
```

- **rootProject.name** : nom global du build/projet (pas nécessairement le nom du dossier contenant)
- **include('app')** : nom des sous-projets inclus, un seul ici : app

app/build.gradle

Groovy | Kotlin

app/build.gradle

```
plugins {
    id 'application' (1)
}

repositories {
    mavenCentral() (2)
}

dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter:5.8.2' (3)

    implementation 'com.google.guava:guava:31.0.1-jre' (4)
}

application {
    mainClass = 'demo.App' (5)
}

tasks.named('test') {
    useJUnitPlatform() (6)
}
```

(1) Apply the application plugin to add support for building a CLI application in Java.

(2) Use Maven Central for resolving dependencies.

(3) Use JUnit Jupiter for testing.

(4) This dependency is used by the application.

(5) Define the main class for the application.

app/src/main/java/demo/App.java

```java
/*
 * This Java source file was generated by the Gradle 'init' task.
 */
package demo;

public class App {
    public String getGreeting() {
        return "Hello World!";
    }

    public static void main(String[] args) {
        System.out.println(new App().getGreeting());
    }
}
```

app/src/test/java/demo/AppTest.java

```java
/*
 * This Java source file was generated by the Gradle 'init' task.
 */
package demo;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class AppTest {
    @Test void appHasAGreeting() {
        App classUnderTest = new App();
        assertNotNull(classUnderTest.getGreeting(), "app should have a greeting");
    }
}
```

./gradlew run et ./gradlew build

```
fab@boid ~/ztmp/demo
$ ./gradlew run

> Task :app:run
Hello World!

BUILD SUCCESSFUL in 588ms
2 actionable tasks: 2 executed
fab@boid ~/ztmp/demo
$ ./gradlew build

BUILD SUCCESSFUL in 1s
7 actionable tasks: 6 executed, 1 up-to-date
```

./gradlew build

- ⇒ app/build/distributions/app.tar
- ⇒ app/build/distributions/app.zip

Plan

Gestion des dépendances

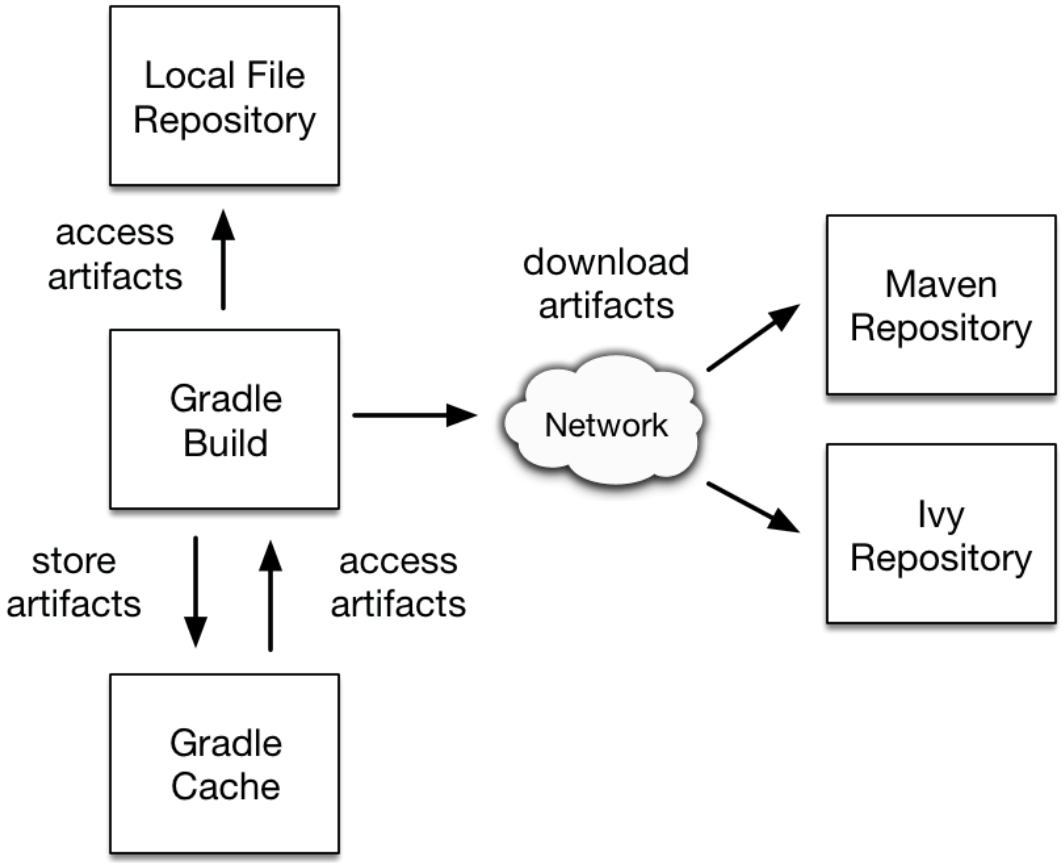
Principes et intérêts

- Généralement un projet nécessite d'autres librairies : fonctionnalités avancées / test / intégration continue . . .
- Gradle $\Rightarrow$ automatisation des tâches de déclaration / résolution / utilisation

Fonctionnement

- déclaration : spécifie la version de la librairie
- résolution : récupère la librairie depuis un dépôt ou localement
- utilisation : spécifie le niveau d'usage : compilation / test / API / runtime. . .

Résolution


Local File Repository
access artifacts
Gradle Build
store artifacts
access artifacts
Gradle Cache
download artifacts
Network
Maven Repository
Ivy Repository

Paramétrage des dépôts

Example 1. Adding central Maven repository

Groovy Kotlin

build.gradle

```
repositories {
    mavenCentral()
}
```

Paramétrage des dépôts

Example 3. Declaring multiple repositories

Groovy Kotlin

build.gradle

```
repositories {
    mavenCentral()
    maven {
        url "https://repo.spring.io/release"
    }
    maven {
        url "https://repository.jboss.org/maven2"
    }
}
```

Paramétrage des dépôts

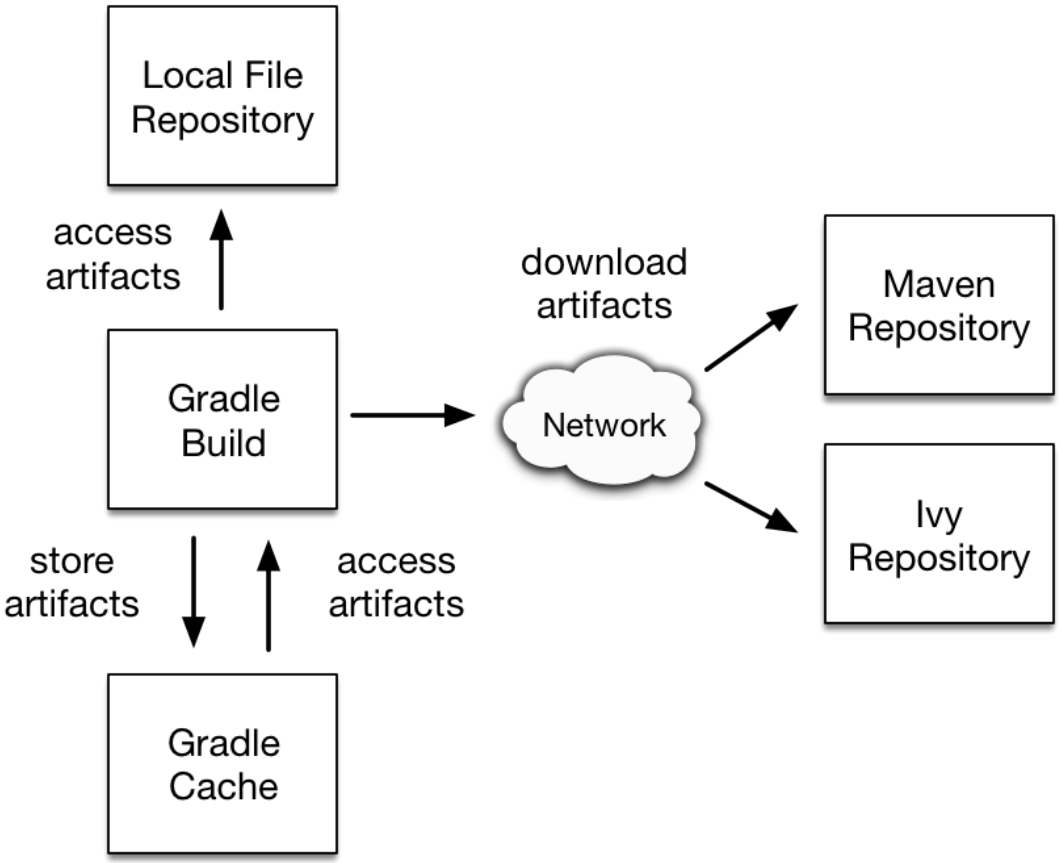
Example 4. Flat repository resolver

Groovy Kotlin

build.gradle

```
repositories {
    flatDir {
        dirs 'lib'
    }
    flatDir {
        dirs 'lib1', 'lib2'
    }
}
```

Résolution


Local File Repository
access artifacts
Gradle Build
store artifacts
access artifacts
Gradle Cache
download artifacts
Network
Maven Repository
Ivy Repository

JavaFX

Parfois, les dépendances sur des projets plus complexes sont configurées grâce à des plugins dédiés.
Exemple avec `org.openjfx.javafxplugin` qui rajoute une tâche `javafx` spécifique

```
plugins {
    // Apply the application plugin to add support for bu
    id 'application'
    id 'org.openjfx.javafxplugin' version '0.0.13'
    id 'org.javamodularity.moduleplugin' version "1.8.12"
}

apply plugin: 'eclipse'

repositories {
    // Use Maven Central for resolving dependencies.
    mavenCentral()
}

javafx {
    version = "19"
    // Found by eclipse
    modules = [ 'javafx.controls', 'javafx.fxml' ]
}
```

JavaFX

Lorsqu'on suit le principe des modules, il peut être nécessaire de déplacer à la main les fichiers jar dans le module path : bouton droit → Build path → configure build path :