Gradle

Open Source Build Automation

Gradle

- Popular build automation system
- Alternative to Ant and Maven
- Supports:
  - Java
  - Android
  - HTML/CSS/Javascript
  - C/C++
  - And more

Gradle

- Flexible, general purpose
- Groovy-based DSL
- Supports build-by-convention
  - But you can override conventions
- Supports Maven and Ivy infrastructure

Gradle Wrapper

- The preferred way of starting a Gradle build
- Run "`sh gradlew <tasks>`"
- The wrapper will download and install Gradle automatically
- Each project can use a different version of Gradle
- Simplifies setup for developers and continuous integration

Installing the Gradle Wrapper

- When the project is first created, you will need a regular installation of Gradle
- Run the wrapper task

```
gradle wrapper —gradle-version 2.4
```

- Generates four files:

```
gradlew
gradlew.bat
gradle/wrapper/gradle-wrapper.jar
gradle/wrapper/gradle-wrapper.properties
```

- These files are checked in to source control

Gradle Daemon

- Starting Java processes is time consuming
- Gradle has a daemon mode where it runs continuously in the background
- Enable it by editing `~/.gradle/gradle.properties`

  ```
  org.gradle.daemon=true
  ```

- Run Gradle normally
- The daemon starts automatically

A Simple Java Project

- Run "`sh gradlew build`"
- Other useful tasks:
  - clean
  - assemble
  - check

Tasks and Customization

- Default tasks can be customized
- You can define your own tasks
- The Gradle documentation lists the available tasks and their properties

Dependencies

- Supports Maven and Ivy
- Dependency sets:
  - compile
  - test
  - runtime
- Lacks Maven's notion of "provided"
  - But you can implement it yourself

Recipe: Fat JAR

- Build a single JAR file with all of the dependencies
- There is an existing 3rd-party plugin for this
- We'll rebuild it here because we will customize it for the next recipe
- We use a Groovy closure to collect all of the runtime dependencies into our JAR file

Recipe: Provided

- Exclude dependencies provided by a container environment
- Again, there are 3rd-party plugins to do this
  - WAR plugin
- We modify the previous Groovy closure to subtract the “provided” set of dependencies from the “runtime” set

Recipe: Dependency Insight

- There are several popular logging APIs and our project's components each use a different one
- We want to know what modules introduce dependencies on the various logging APIs so that we can exclude them
- Dependency Insight is built directly into Gradle
- Helps to find and resolve dependency conflicts
- Run "`sh gradlew dependencyInsight –dependency <name>`"

Recipe: Dependency Mapping

- Another strategy for resolving the logging API mess
- We want to unify all dependencies (including transitive ones) to use SLF4J over Logback
- We use a Groovy closure to map logging dependencies to SLF4J and Logback