Gradle - Training

Mithun Technologies
devopstrainingblr@gmail.com +91-9980923226

- Agenda

- ✓ Introduction
- ✓ Installation
- ✓ Content –.gradle files
- ✓ Examples (Standalone, Web, Enterprise Projects)
- ✓ Integration with SonarQube
- ➢ Integration with Nexus/Jfrog Artifactory

- Introduction

Type	
Vendor	Gradle Inc
Is Open Source?	Yes
Initial release	v0.7(Jul 20, 2009)
Stable release	V4.9 (July 16, 2018)
Developers	Hans Dockter, Adam Murdoch, Szczepan Faber, Peter Niederwieser, Luke Daley, Rene Gröschke, Daz DeBoer, Steve Appling
Software Download URL	https://gradle.org/releases/
Is executable software?	No, download as zip, extract and use it.
Reference Websites	https://docs.gradle.org/current/userguide/userguide.html

- Introduction

Before discussing about Gradle, first we will go through some shortcomings of ant and maven. Then we will discuss about why we need Gradle build tool to our projects.

Apache Ant Benefits

Ease of Use – The tool provides a wide range of tasks that almost fulfils all the build requirements of the user.
Platform Independent – Ant is written in Java thus is a platform-independent build tool. The only requirement for the tool is JDK.
Extensibility – As the tool is written in Java and the source code is freely available, a user is leveraged with the benefit to extend the tool's capabilities by writing java code for adding a task in Ant Libs.

Drawbacks of Ant

1) We need to write ant build scripts using XML. If we want to automate a complex project, then we need to write a lot of logic in XML files.
2) There is no built-in ant project structure. Since we can use any build structure for our projects, new developers find it hard to understand the project struct and build scripts.
3) It is very tough to write some complex logic using if-then-else statements.
4) We need to maintain all jars with required version in version control. There is no support for dependency management.

- Introduction

Because of these many drawbacks, now-a-days most of the projects have restructured and they are using maven build tool. Even though Maven provides the following benefits compare to ant, but still it has some drawbacks.

Advantages of Maven

1) Maven is an expressive, declarative, and maintainable build tool.
2) All maven projects follow pre-defined structure. Even new developers find it easy to understand the project structure and start development quickly.
3) We don’t need to maintain all jars with required version in version control. Maven will download all required jars at build time. Maven support for dependency management is one of the best advantage it has over ant.
4) Maven gives us very modularized project structure.

Drawbacks of Maven

1) Maven follows some pre-defined build and automation lifecycle. Sometimes it may not fit to our project needs.
2) Even though we can write custom maven life cycle methods, but it’s a bit tough and verbose.
3) Maven build scripts are a bit tough to maintain for very complex projects.
4) Maven has solved most of the ant build tool issues, but still it has some drawbacks. To overcome these issues, we need to use Gradle build tool. Now we can start our discussion on Gradle.

- Introduction

Gradle is an open-source build automation tool that builds upon the concepts of Apache Ant and Apache Maven and introduces a Groovy-based domain-specific language (DSL) instead of the XML form used by Apache Maven for declaring the project configuration.

Supports java, groovy, scala, antlr, c, cpp programing languages ..et.c

At a glance Gradle Open source: Apache License v2 First release 2009, latest release July 2018 (4.9) Backed by Gradleware... ooops, I mean Gradle Inc.
Used by: Android, Spring IO, LinkedIn, Netflix, Twitter and more...

Gradle uses a directed acyclic graph ("DAG") to determine the order in which tasks can be run.

Gradle is implemented using Groovy, and Gradle build scripts are actually Groovy scripts. Groovy is a programming language that runs on the Java Virtual Machine (JVM).

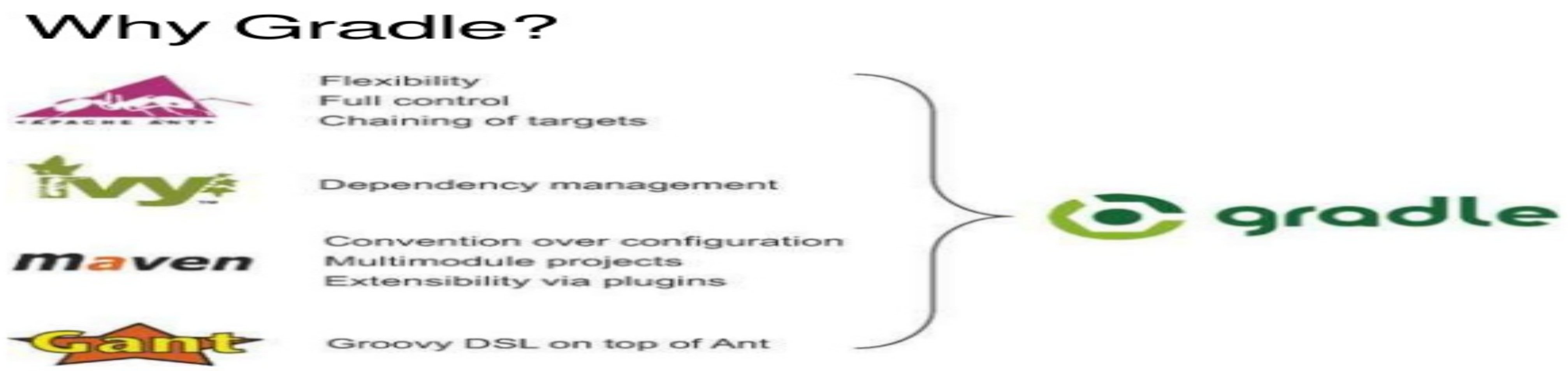
- Introduction

Gradle advantages

Gradle will provide the following advantages compared to ant and maven. That’s why all new projects are using Gradle as build tool.

1) Gradle’s build scripts are more readable, expressive and declarative.
2) Gradle’s build scripts are written in simple Groovy, no XML. That means it use it’s own DSL based on Groovy script.
3) Gradle provides very scalable and high-performance builds.
4) Unlike Maven’s pom.xml, No need to write boilerplate XML Code.
5) It’s very easy to use gradle tool and implement custom logic in our project.
6) Gradle provides standard project layout and lifecycle, but it’s full flexibility. We have the option to fully configure the defaults. This is where it’s better than maven.
7) It is very easy to maintain even for multi-module complex projects.
8) It is very easy to integrate/migrate existing ant/maven project with Gradle.

- Introduction



Gant is a Groovy based build system that internally uses ant tasks. In Gant, we need to write build scripts in Groovy, but no XML.
Apache Ivy sub-project of the Apache Ant project, with which Ivy works to resolve project dependencies. An external XML file defines project dependencies and lists the resources necessary to build a project. Ivy then resolves and downloads resources from an artifact repository: either a private repository or one publicly available on the Internet.
To some degree, it competes with Apache Maven, which also manages dependencies. However, Maven is a complete build tool, whereas Ivy focuses purely on managing transitive dependencies.

- Introduction

Why Gradle?

- Installation

Prerequisites

Gradle runs on all major operating systems and requires only a Java JDK version 7 or higher to run. To check, run java -version. You should see something like this:

```
❯ java -version
java version "1.8.0_151"
Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)
```

Gradle ships with its own Groovy library, therefore Groovy does not need to be installed. Any existing Groovy installation is ignored by Gradle.

Gradle uses whatever JDK it finds in your path. Alternatively, you can set the JAVA_HOME environment variable to point to the installation directory of the desired JDK.

- Installation

Install with a package manager

SDKMAN! is a tool for managing parallel versions of multiple Software Development Kits on most Unix-based systems.

```
$ sdk install gradle 4.9
```

Homebrew is “the missing package manager for macOS”.

```
$ brew install gradle
```

Scoop is a command-line installer for Windows inspired by Homebrew.

```
$ scoop install gradle
```

- Installation

Installing manually

Step 1. Download the latest Gradle distribution
https://gradle.org/releases/
The distribution ZIP file comes in two flavors:
Binary-only (bin)
Complete (all) with docs and sources

Step 2. Unpack the distribution

Linux & MacOS users
Unzip the distribution zip file in the directory of your choosing, e.g.:

```
❯ mkdir /opt/gradle
❯ unzip -d /opt/gradle gradle-4.9-bin.zip
❯ ls /opt/gradle/gradle-4.9
LICENSE  NOTICE  bin  getting-started.html  init.d  lib  media
```

- Installation

Microsoft Windows users
Create a new directory C:\Gradle with File Explorer.

Open a second File Explorer window and go to the directory where the Gradle distribution was downloaded. Double-click the ZIP archive to expose the content. Drag the content folder gradle-4.9 to your newly created C:\Gradle folder.

Alternatively you can unpack the Gradle distribution ZIP into C:\Gradle using an archiver tool of your choice.

Step 3. Configure your system environment
For running Gradle, firstly add the environment variable GRADLE_HOME. This should point to the unpacked files from the Gradle website. Next add GRADLE_HOME/bin to your PATH environment variable. Usually, this is sufficient to run Gradle.

Linux & MacOS users
Configure your PATH environment variable to include the bin directory of the unzipped distribution, e.g.:

```
❯ export PATH=$PATH:/opt/gradle/gradle-4.9/bin
```

- Installation

Microsoft Windows users

In File Explorer right-click on the This PC (or Computer) icon, then click Properties → Advanced System Settings → Environmental Variables.
Under System Variables select Path, then click Edit. Add an entry for C:\Gradle\gradle-4.9\bin. Click OK to save.

Verifying installation
Open a console (or a Windows command prompt) and run gradle -v to run gradle and display the version, e.g.:

```
C:\Users\BalajiReddyLachhanna>gradle -v
------------------------------------------------------------
Gradle 4.9
------------------------------------------------------------
Build time:   2018-07-16 08:14:03 UTC
Revision:     efcf8c1cf533b03c70f394f270f46a174c738efc
Kotlin DSL:   0.18.4
Kotlin:       1.2.41
Groovy:       2.4.12
Ant:          Apache Ant(TM) version 1.9.11 compiled on March 23 2018
JVM:          1.8.0_152 (Oracle Corporation 25.152-b16)
```

- Core Concepts

Core Concepts

Build script: a build configuration script supporting one or more projects.
Project: a component that needs to be built. It is made up of one or more tasks.
Task: a distinct step required to perform the build. Each task/step is atomic (either succeeds or fails).
Publication: the artifact produced by the build process.

Dependency Resolution

Dependencies: tasks and projects depending on each other (internal) or on third-party artifacts (external).
Transitive dependencies: the dependencies of a project may
themselves have dependencies.
Repositories: the “places” that hold external dependencies
(Maven/Ivy repos, local folders)
DAG: the directed acyclic graph of dependencies (what depends on what).
Dependency configurations : named sets (groups) of dependencies (e.g. per task).

.

- Core Concepts

The Build Lifecycle

1. **Initialization**:

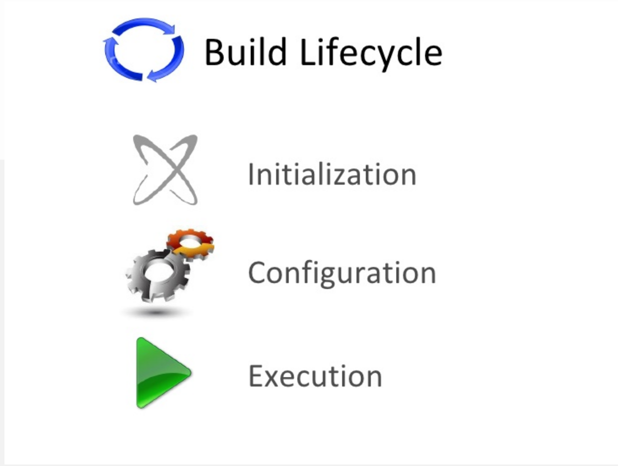
   Gradle supports single and multi-project builds. During the initialization phase, Gradle determines which projects are going to take part in the build, and creates a Project instance for each of these projects.

.2. **Configuration**:

   During this phase the project objects are configured. The build scripts of all projects which are part of the build are executed. Computes the DAG.

3. **Execution**:

   Gradle determines the subset of the tasks, created and configured during the configuration phase, to be executed. The subset is determined by the task name arguments passed to the gradle command and the current directory. Gradle then executes each of the selected tasks.



.

Groovy & Gradle

gradle task != ant task
gradle task == ant target

build.gradle

```
task count{
 4.times {print "$it "}
}
```

Run Gradle
Syntax : gradle <taskName> [orgs1 orgs 2]

➢ gradle count

```
gradle –q count
gradle count --warning-mode all
gradle count –info
gradle count –-dry-run
gradle count –console=plain
```

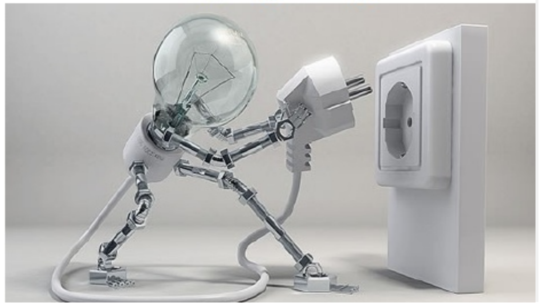
Plugins

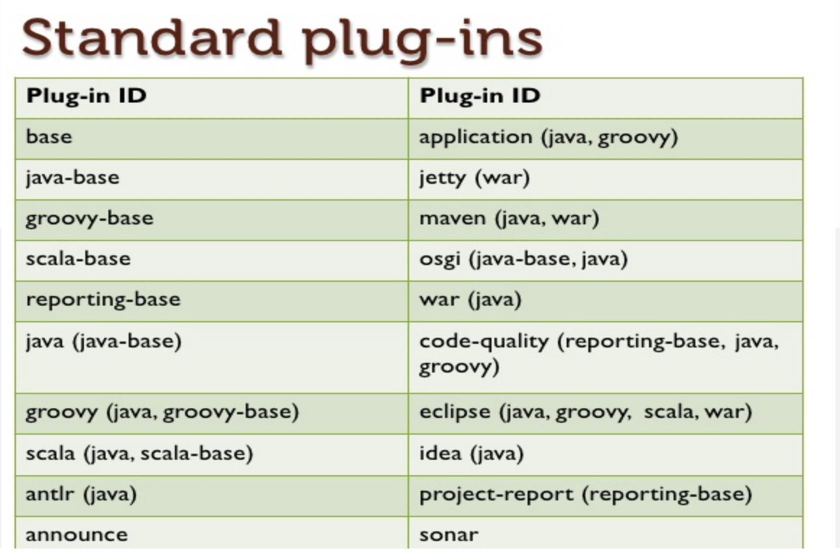
A plugin applies a set of extensions to the build process.

- Add tasks to a project.
- Pre-configure these tasks with reasonable defaults.
- Add dependency configurations.
- Add new properties and methods to existing objects.

Gradle Plugins

java scala groovy cpp antrl
checkstyle findbugs pmd sonar
ear war osgi jetty maven





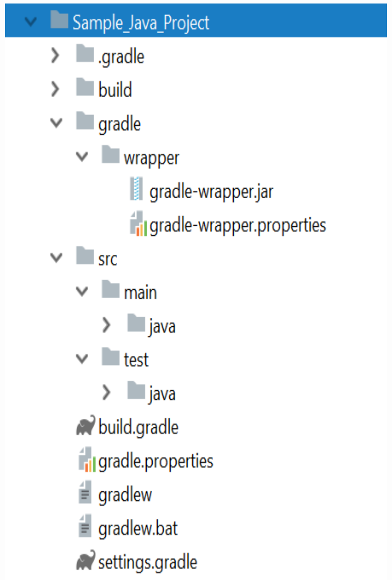
Standard plug-ins

Plug-in ID	Plug-in ID
base	application (java, groovy)
java-base	jetty (war)
groovy-base	maven (java, war)
scala-base	osgi (java-base, java)
reporting-base	war (java)
java (java-base)	code-quality (reporting-base, java, groovy)
groovy (java, groovy-base)	eclipse (java, groovy, scala, war)
scala (java, scala-base)	idea (java)
antlr (java)	project-report (reporting-base)
announce	sonar

Gradle – Create Java project structure automatically

To quick start a new Gradle Java project, type gradle init --type java-library

```
mkdir Sample_Java_Project
cd Sample_Java_Project
gradle init --type java-library
```



```
Sample_Java_Project
  .gradle
  build
  gradle
    wrapper
      gradle-wrapper.jar
      gradle-wrapper.properties
  src
    main
      java
    test
      java
  build.gradle
  gradle.properties
  gradlew
  gradlew.bat
  settings.gradle
```

- Sample build.gradle

The simple example

The contents of build.gradle (aka the build script), placed in the root folder of the project.

```
apply plugin: ”java”

//Introduces a set of Maven style conventions
//(tasks, source locations, dependency configurations, etc)
group     = ”com.balu”
version = ”1.0 SNAPSHOT”
repositories {
//resolve all external dependencies via Maven central
  mavenCentral()
 }
dependencies {
//each name (compile, testCompile, etc) refers to
//a configuration introduced by the java plugin
compile”commons io:commons io:2.4”
testCompile ”junit:junit:4.11”
runtime  files(”test/foo.jar”, ”lib/utils.jar”)
}
```

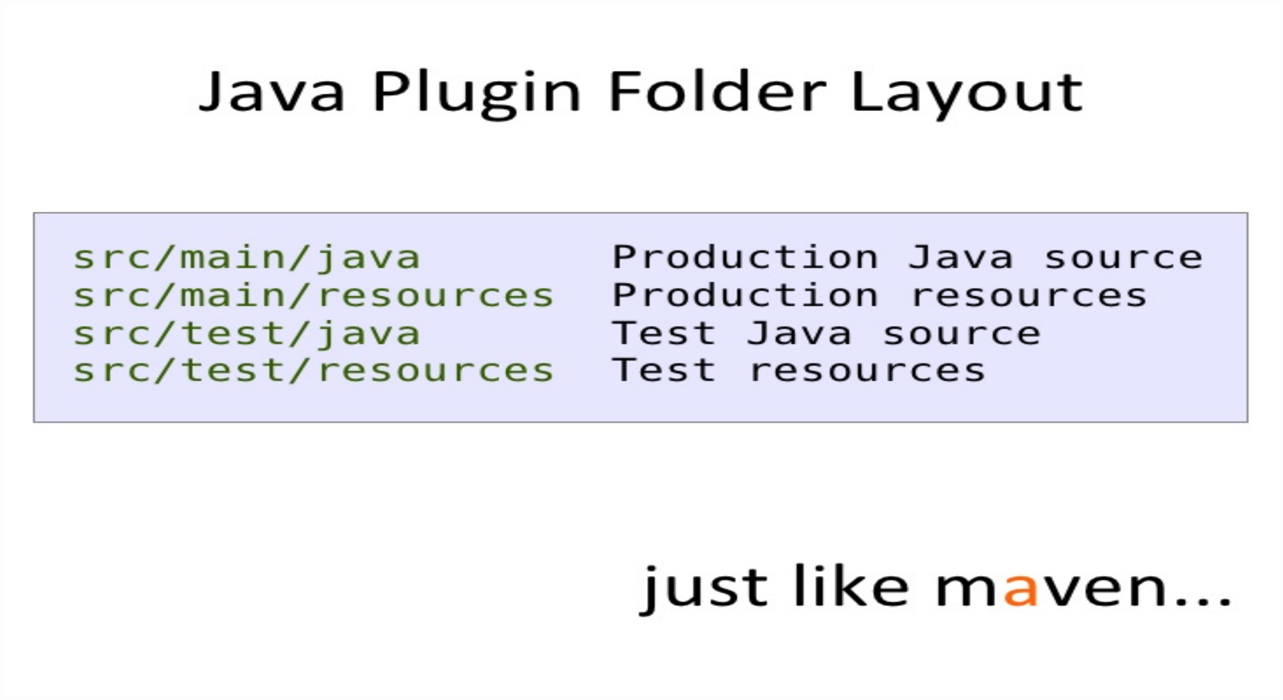
Java Plugin

Java Plugin

buid.gradle

apply plugin: ‘java’

- ➢ **gradle build**



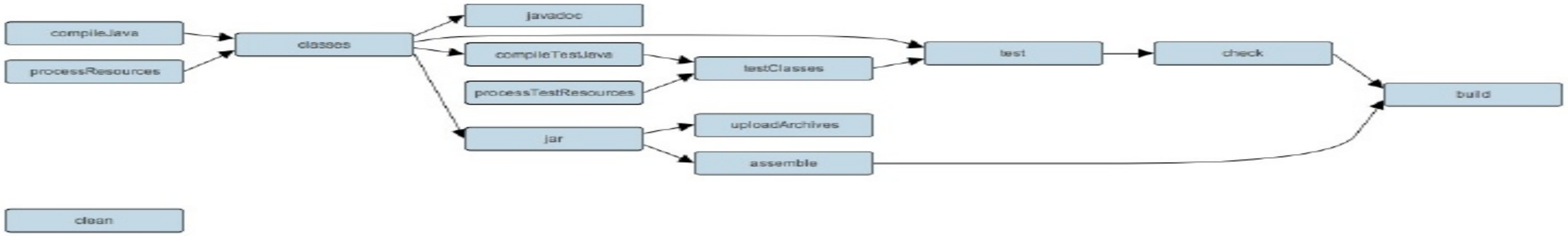
Java Plugin Folder Layout

```
src/main/java           Production Java source
src/main/resources      Production resources
src/test/java           Test Java source
src/test/resources      Test resources
```

just like maven...

.

Java Plugin Tasks Flow



War Plugin

War Plugin:
The War plugin extends the Java plugin to add support for assembling web application WAR files. It disables the default JAR archive generation of the Java plugin and adds a default WAR archive task.
Usage
To use the War plugin, include the following in your build script (build.gradle)

```
apply plugin: "war"
or
plugins {
   id 'war'
}
```

commands

- ➢ gradle war (or)
- ➢ gradle clean build

Project layout
In addition to the standard Java project layout, the War Plugin adds:
src/main/webapp
Web application sources
For more info refer: https://docs.gradle.org/current/userguide/userguide_single.html#war_plugin

SonarQube

Analyzing with SonarQube Scanner for Gradle

The SonarQube Scanner for Gradle provides an easy way to start SonarQube analysis of a Gradle project.
The ability to execute the SonarQube analysis via a regular Gradle task makes it available anywhere Gradle is available (developer build, CI server, etc.), without the need to manually download, setup, and maintain a SonarQube Runner installation.

Usage

To use the SonarQube plugin, include the following in your build script (build.gradle)

```
plugins {
  id "org.sonarqube" version "2.6.2"
}
sonarqube {
    properties {
        property 'sonar.host.url', 'http://localhost:9000'
        property 'sonar.login', 'balaji'
        property 'sonar.password', 'password'
    }
}
```

Commands: > gradle sonarqube or gradle clean install sonarqube

Integration With Nexus

Integration With Nexus
The Maven Publish Plugin provides the ability to publish build artifacts to an Apache Maven or any custom repository like Nexus or JFrog repository. A module published to a Maven repository can be consumed by Maven, Gradle.

Usage
To use the SonarQube plugin, include the following in your build script (build.gradle)

```
plugins {
    id 'maven-publish'
}
or
apply plugin: 'maven-publish’
```

Integration With Nexus

```
publishing {
       publications {
        maven(MavenPublication) {
            artifactId publishName
            groupId 'com.rts'
            version project.version
            // "boot" jar
            artifact ("$buildDir/libs/$publishName-$version-boot.jar") {
                classifier = 'boot'
            }
        }
    }
    repositories {
        maven {
            credentials {
                username 'admin'
                password 'admin123'
            }
            url "http://localhost:8081/repository/maven-releases"

        }
    }
}
```

Commands: > gradle clean build publish or gradle publish

Multi-Module Project

A multi-module project has one main module and many submodules. It has this layout:

```
(root)
  +- gradle.properties     # optional
  +- settings.gradle
  +- build.gradle
+-- ...
+-- sub-a/
  |     +- build.gradle
  |     +- src/
  +-- sub-b/
        +- build.gradle
        +- src/
```

settings.gradle
The main role of settings.gradle is to define all included submodules and to mark the directory root of a tree of modules, so you can only have one settings.gradle file in a multi-module project.

```
rootProject.name = 'project-x'
include 'sub-a', 'sub-b’
```
The settings file is also written in groovy, and submodule lookup can be adapted alot.

gradle.properties
This is optional, it's main purpose is to provide startup options to use for running gradle itself, e.g.

```
org.gradle.jvmargs=-Dfile.encoding=UTF-8 ...
org.gradle.configureondemand=true
```

build.gradle
There is one such file per module, it contains the build logic for this module.
In the build.gradle file of the main module, you can use allprojects {} or subprojects {} to define settings for all other modules.
In the build.gradle file of the submodules, you can use compile project(':sub-a') to make one submodule depend on the other.

Questions ?

Mithun Technologies

Thank you

Mithun Technologies