How to - Build Gradle

GRADLE

Gradle is a build tool similar to Maven and Ant.It is an open source build automation system that builds up on ANT, Maven and Ivy repositories**.**

It supports groovy based Domain Specific Language (DSL) over the XML**.**

Maven	Gradle
Maven is a software project management and comprehension tool. It is mostly used with Java-based projects.	Gradle is a build automation system that uses a domain-specific language based on the programming language Groovy.
Maven uses an XML file to describe the project, its dependencies, the build order, and its required plug-ins.	Gradle does not use an XML file to describe a project that is being built, instead, it uses the domain-specific language **Groovy DSL**; a Java-based DSL that supports Groovy language features.
Maven Build scripts are not as short or clean	Gradle scripts are short, clean and easy to understand
Makes build process easier	Allows Structuring the build and supports Multi-project builds

Gradle Core Concepts

- Projects
- Tasks
- Build scripts

Project

Project is a representation of what needs to be built and leads to an artifact at the end of the build.A project typically has a build script in which its tasks are defined.

- Projects are registered in the ***settings.gradle*** file
- Projet has a ***build.gradle*** file

Tasks

Tasks are actions needed to build the project. A task could be compiling the source code, generating JavaDoc, zipping the compiled classes into a JAR file etc. A Gradle project contains one or more tasks to execute in order to build the project.

Build Phases

Every Gradle build goes through 3 different lifecycle phases

The initialization phase

In this phase, Gradle tries to identify all the projects involved in the build process. It is very important for Gradle to know whether it's a Single-project build or a Multi-project build. In a Multi-project build there are several projects to evaluate. Hence, several build scripts. Gradle looks at the ***settings.gradle*** file in order to identify the different projects. At the end of the initialization phase, Gradle creates an instance of ***org.gradle.api.Project*** corresponding to each of these projects.

The configuration phase

Gradle executes the build script of each project identified in the previous phase.

The execution phase

Gradle identifies the tasks that need to be executed based on the DAG of task objects created in the previous phase, and executes them according to their dependency order.

How to Install Gradle on linux

Prerequisites

- Java JDK installed
- Linux/centos/Ubuntu

Step 1: System update

Before installing any packages on your CentOS server instance, it is recommended to update the system. Login using the sudo user and run the following commands to update the system.

```
yum -y install epel-release
yum -y update
```

Step 2 :Download Gradle

```
wget https://services.gradle.org/distributions/gradle-3.4.1-bin.zip
```

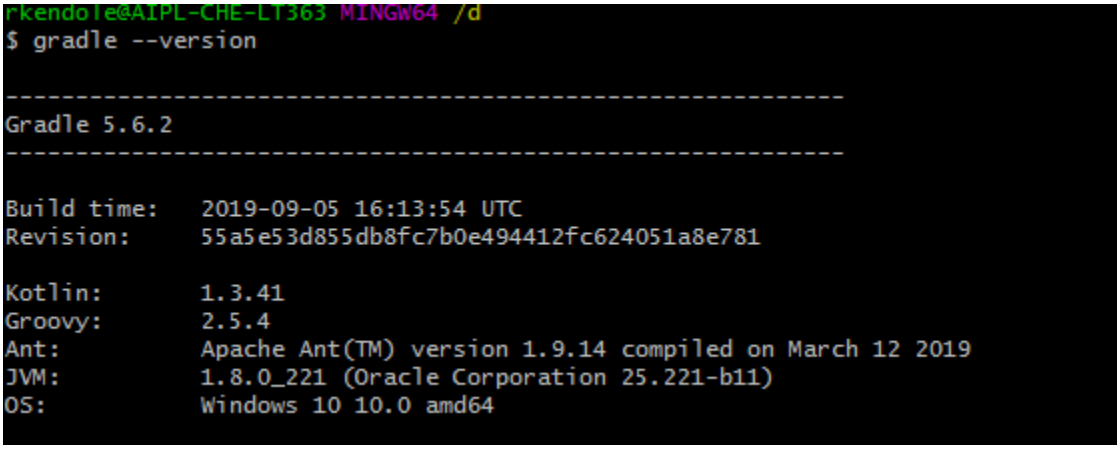
Step 3 : Install Gradle

Extract the downloaded archive to the newly created directory

Configure the `PATH` environment variable so that the `gradle` executable can be directly executed anywhere on the system.

You can run the following command to check if the Gradle install was successful.

Gradle -V



Building Java Projects with Gradle

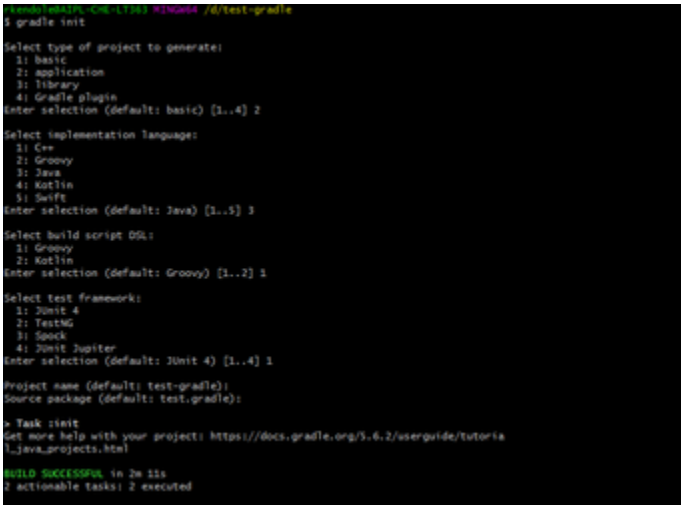
Initiating a Java Project with Gradle

Gradle comes with a built-in plugin called the Build Init plugin.The plugin provides a task, called `init`, that generates the project.The plugin also uses the Wrapper task to create gradle wrapper script **gradlew.**

Run the init task

Run the `init` task and select `java-application` project type when prompted.

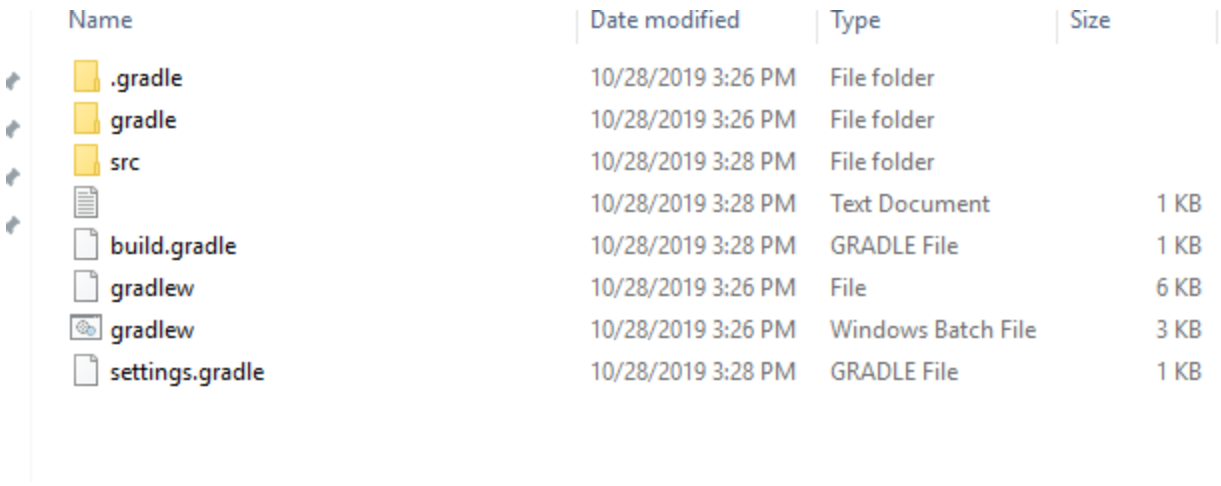
```
############################
Gradle cmd

Run the init task
############################

# gradle init
```



Since we selected **application** as our project type,Gradle lets us chose from one of four programming languages to build our project.

Next, we're able to select either Groovy or Kontlin as the language in which our *build scripts* will be written in.

Folder structure:



Gradle has started our project by generating the barebones of our project as well as some Gradle-related configuration files. Not only does Gradle create our main module within /src, but it also creates the corresponding folder structure needed to test said module.

Let's take a look at the files related to using Gradle in our project:

- **build.gradle**: This is the file we'll use to configure our builds. This is where we specify things like dependencies to download, tasks to run, projects to import, etc. This is the file we'll do the most work in.
- **settings.gradle**: Additional settings for our Gradle build.
- **gradlew**: `gradlew` stands for "Gradle wrapper" and is the file we'll use to execute our builds (for example: `$ ./gradlew build`). When we initialize Gradle in a project we actually decouple our project's Gradle from our system's Gradle, which means we could hand off our source to somebody who doesn't have Gradle installed and they'd still be able to build our project

Run Gradle tasks:

Before you even create a build.gradle file for the project, you can ask it what tasks are available:

You should see a list of available tasks. Assuming you run Gradle in a folder that doesn't already have a *build.gradle* file, you'll see some very elementary tasks such as this:

Build Java code:

- Open `build.gradle` file and add this line `apply plugin: 'java'`.

Create a very basic `build.gradle` file in the <project folder> you created at the beginning of this guide

```
####################################
build.gradle
apply plugin: 'java'
####################################
```

This single line in the build configuration brings a significant amount of power. Run gradle tasks again, and you see new tasks added to the list, including tasks for building the project, creating JavaDoc, and running tests.

- Now Run this command `gradle build`.

To see the results of the build effort, take a look in the build folder. Therein you'll find several directories, including these three notable folders:

- classes. The project's compiled .class files.
- libs. Assembled project libraries (usually JAR and/or WAR files).
- *reports*. Reports produced by the build (such as test reports).

```
######################

Gradle command to  build the code

gradle build



######################
```