Release engineering using Maven and Jenkins

This page describes how release engineering is currently performed for Commons REST (CREST) and the OpenDJ LDAP SDK. Both projects are multi-module Maven projects. Since they are components of other projects it is important that both of them are maintained regularly and released frequently. Therefore, the release engineering process must be as simple and lightweight as possible.

This page is not a guide nor a "best practices": it is just a description of what we are doing currently and any feedback on how we can improve the process is more than welcome. The main purpose is to share the knowledge and hopefully save some time for other projects.

Ok, let's get started...

Version control

NOTE: I'm assuming that many of the concepts below will apply equally to Git when we migrate.

Many of the tools discussed below are "convention" based and therefore work much better if your Subversion repository conforms to the standard project layout convention of trunk/branches/tags. Deviate at your own risk! Let's look at these in more detail:

- **`<`*`project`*`>/trunk`**

  Example SVN: https://svn.forgerock.org/commons/forgerock-rest/trunk
  Example pom version: 2.1.0-SNAPSHOT

  Description: this is where the next "dot zero" version of the project is being developed. I think we all know that, so no more details required. Projects should never be released from the trunk. Instead, a branch should be created first and the release performed from the branch.

  Maven versioning: the version assigned to the trunk (via the various pom.xml files) is of the form "x.y.0-SNAPSHOT", where "x.y" is greater than the most recent branch.

  Jenkins: we only have a single "post commit" job for the trunk, which is triggered periodically, or when dependencies are updated, or when changes are committed. Release engineering is disabled for this job.

- **`<`*`project`*`>/branches/<`*`major`*`>.<`*`minor`*`>`**

  Example SVN: https://svn.forgerock.org/commons/forgerock-rest/branches/2.0
  Example pom version: 2.0.2-SNAPSHOT

  Description: this is where existing versions of the project are maintained and prepared for release. Typically changes are cherry picked from the trunk and back-ported to the appropriate branch(es). New versions of the project are always released from a branch, and never the trunk. In particular, a "dot-zero" release developed on the trunk, is first branched, and then released.

  Maven versioning: the version assigned to a branch is of the form "x.y.z-SNAPSHOT" where "x.y" is older than the trunk version, and "z" is more recent than the most recent tag associated with the branch.

  Jenkins: we have two jobs per branch. The first is a "post commit" job, identical to the trunk "post commit" job and differing only in the Subversion checkout URL which refers to the branch instead of the trunk. The second job is the job which will be used for performing release engineering. It has a very different configuration compared to the post commit jobs. More on that below...

  `<`*`project`*`>/tags/<`*`major`*`>.<`*`minor`*`>.<`*`micro`*`>`

  Example SVN: https://svn.forgerock.org/commons/forgerock-rest/tags/2.0.1
  Example pom version: 2.0.1

  Description: this is where released versions of the project are located. Although not enforced, tags should be considered read-only and, therefore, should never be updated once they have been created. A released product is built from its tag and deployed to our Maven repository (Artifactory). If a release fails for some reason, the tag should be rolled back (i.e. deleted) - a tedious process which should rarely occur because the project should have already been stabilized on the branch.

  Maven versioning: the version assigned to a tag is of the form "x.y.z" where "x.y.z" is older than the associated branch version. In addition, a released project MUST NOT have any dependencies on unreleased (SNAPSHOT) components.

  Jenkins: we do not have any jobs for building tags. Typically a tag is built once while performing a release from a branch.

Configuring Maven for branching and releasing

We use the Maven Release Plugin for creating branches and release engineering. Although slower than the Artifactory release plugin, I have found it to be more "rigorous". In particular, the plugin will verify that there are no SNAPSHOT dependencies as well as building the release from the tag and not the branch (something the Artifactory does and which is very bad).

There are two steps to configuring your project to use this plugin:

1. Ensure that your project's parent `pom.xml` has a valid "scm" configuration. Note that this configuration does not need to be repeated in the sub-modules, it is only required in the parent. The "scm" configuration tells the release plugin where it should check code out from and where updates, such as tagging, branching, and updating pom versions, should be performed.

   It is absolutely critical that the URLs are valid and correspond to the project version's trunk, branch, or tag. In particular, the "scm" configuration must be updated when creating a branch from the trunk, a process which is handled automatically by the Maven release plugin's "branch" goal (see below).

Example: CREST has the following "scm" configurations:

Trunk:

```
    <scm>
        <url>https://svn.forgerock.org/commons/forgerock-rest/trunk/</url>
        <connection>scm:svn:https://svn.forgerock.org/commons/forgerock-rest/trunk/</connection>
        <developerConnection>scm:svn:https://svn.forgerock.org/commons/forgerock-rest/trunk/<
/developerConnection>
    </scm>
```

Branch:

```
    <scm>
        <url>https://svn.forgerock.org/commons/forgerock-rest/branches/2.0</url>
        <connection>scm:svn:https://svn.forgerock.org/commons/forgerock-rest/branches/2.0</connection>
        <developerConnection>scm:svn:https://svn.forgerock.org/commons/forgerock-rest/branches/2.0<
/developerConnection>
    </scm>
```

Tag:

```
    <scm>
        <url>https://svn.forgerock.org/commons/forgerock-rest/tags/2.0.1</url>
        <connection>scm:svn:https://svn.forgerock.org/commons/forgerock-rest/tags/2.0.1</connection>
        <developerConnection>scm:svn:https://svn.forgerock.org/commons/forgerock-rest/tags/2.0.1<
/developerConnection>
    </scm>
```

2. **This step is NOT necessary if you are using forgerock-parent 1.2.0 or newer.** If you are using a version of forgerock-parent older than 1.2.0 then you will need to configure the Maven release plugin.

   For some reason (I don't know why), our ForgeRock parent pom defines some pretty strange defaults that I can never get to work. Both CREST and the OpenDJ LDAP SDK override these settings and revert them back to something more aligned with the plugin's default settings. Here's what the CREST configuration looks like (note that it is defined in a plugin management section):

```
<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-release-plugin</artifactId>
                <inherited>true</inherited>
                <configuration>
                    <!-- Disable inherited configuration -->
                    <autoVersionSubmodules>true</autoVersionSubmodules>
                    <mavenExecutorId>forked-path</mavenExecutorId>
                    <useReleaseProfile>true</useReleaseProfile>
                    <suppressCommitBeforeTag>false</suppressCommitBeforeTag>
                    <goals>deploy</goals>
                    <arguments>-Penforce</arguments>
                </configuration>
            </plugin>
        </plugins>
    </pluginManagement>
</build>
```

Creating project branches

It is tempting to use a command like "svn cp" in order to create a branch. However, this is not enough since it will not update the "scm" configuration before creating the branch. If you forget to update the "scm" configuration then release engineering will go horribly wrong and in a very confusing way.

Fortunately, the Maven release plugin makes the process very easy! With the above plugin configuration, all we do is issue the following command from a checked out copy of the trunk:

```
mvn release:branch -DbranchName=<major.minor> -DdevelopmentVersion=<major.minor.0>-SNAPSHOT
```

Where the "branchName" is the **SVN branch name**, e.g. "2.0", and the "developmentVersion" is the next development **Maven version**. For example:

```
mvn release:branch -DbranchName=2.0 -DdevelopmentVersion=2.1.0-SNAPSHOT
```

Configuring Jenkins jobs for release engineering

As discussed earlier we have "post commit" Jenkins jobs for the trunk and each branch. These do not require any special configuration, except that they should not include any release engineering settings. Here our OpenDJ LDAP SDK 2.6 branch post commit job:

https://builds.forgerock.org/view/OpenDJ/job/OpenDJ%20SDK%20-%20branch%202.6%20-%20Snapshot/

Unfortunately, configuring Jenkins to perform release engineering is not straightforward. You would hope that it would be a simple case of enabling the Jenkins M2 release plugin for the branch job, but it doesn't work for a couple of reasons:

- One of our Jenkins build agents uses SVN 1.6 whereas the others use SVN 1.7. Jenkins requires that all workspaces are checked out using the same SVN workspace version, so we have a global "lowest common denominator" setting of 1.6. During a release we check out the working copy as 1.6 but the Maven release plugin attempts to interact with the workspace using the SVN tool's native version, which fails due to incompatibility on all but one of our build agents (the one whose native version is 1.6). The workaround is to lock the Job to a single build agent - the one using 1.6, but this applies to all builds - post-commit and release. In addition, I have found various incompatibilities between the 1.6 based build agent and various Jenkins plugins (e.g. code coverage is not possible).
- The Jenkins post job deploy plugins (Artifactory and default) are dumb and don't recognize a release build. Upon completion of a successful release the workspace is left in a state where it contains released artifacts but pom files which have been updated to the next development version. The Jenkins deploy actions then kick in and attempt to redeploy the artifacts which Artifactory correctly refuses due to the version mismatch. The only workaround is to disable the Jenkins deploy actions, and instead deploy directly from Maven during the build (i.e. clean install deploy).

Here are the steps to configure a Jenkins release engineering job:

1. lock the job to the SVN 1.6 build agent by selecting "Restrict where this project can be run" and specifying the label expression "solaris && x86"

   Also, scroll back up the page to the top section of the configuration to sure that you are using the correct JDK, e.g. set JDK to JDK.

2. ensure that you have the correct Subversion URL which should point to the branch, e.g: https://svn.forgerock.org/opendj/branches/b2.6-sdk

3. deselect all build triggers - these builds are triggered manually

4. configure default build settings, e.g. Maven 3.0.5. The build goals and options are not important since the Maven release plugin will override them I think

5. enable the M2 release plugin by selecting "Maven release build" in "Build Environment". The goals should look like: "`-Dresume=false release:prepare release:perform`". Note that you should not enable "Enable Artifactory release management". Set the following "Environment Variables":

   ```
   PATH=/opt/SUNWspro/bin:/usr/sfw/bin:/usr/ccs/bin:/usr/local/bin:/usr/bin:/usr/ucb:/bin:/opt/CollabNet_Subversion/bin:/opt/jdk/bin
   ```

6. Remove all "Post Build Actions", especially any that look like they "deploy artifacts".

Once configured you should see a "Perform Maven Release" option on the job's main page.

Releasing a project using Jenkins

Releasing the next version of the project is now easy once everything has been configured correctly! 😎

Releases are always performed from a branch, so make sure that a branch has been created first and a Jenkins release job created for it as well. Before performing a release, make sure that the branch builds successfully and reliably. It is a pain to roll-back partially failed releases.

To perform a release go to the Jenkins release job's main page and click on "Perform Maven Release":

1. Release Version: type in the tag version, e.g. "2.0.1"
2. Development version: type in the next development SNAPSHOT version for the branch, e.g. "2.0.2-SNAPSHOT"
3. Specify custom SCM tag: type in the SVN tag that you want to use, e.g. "2.0.1". This will usually be the same as the "release version", but may differ in some cases. For example, our SDK uses tags of the form "b<version>-sdk".

For the first release you may get some strange "NaN" defaults. On subsequent releases you may get some sensible defaults, although I think that you will always need to specify the SCM tag (I haven't found a way to specify a default).
Once you have verified all the settings press "Schedule Maven Release Build" and cross your fingers...