Building a maven2 project

Jenkins provides a job type dedicated to Maven 2/3. This job type integrates Jenkins deeply with Maven 2/3 and provides the following benefits compared to the more generic free-style software project.

- Jenkins parses Maven POMs to obtain much of the information needed to do its work. As a result, the amount of configuration is drastically reduced.
- Jenkins listens to Maven execution and figures out what should be done when on its own. For example, it will automatically record the JUnit report when Maven runs the test phase. Or if you run the javadoc goal, Jenkins will automatically record javadoc.
- Jenkins automatically creates project dependencies between projects which declare SNAPSHOT dependencies between each other. See below.

Thus mostly you just need to configure SCM information and what goals you'd like to run, and Jenkins will figure out everything else.

TODO

Talk about how maven modules gets represented in sub projects, and how build numbers among sub-projects are controlled. Use some screenshots.

Automatic build chaining from module dependencies

Jenkins reads dependencies of your project from your POM, and if they are also built on Jenkins, triggers are set up in such a way that a new build in one of those dependencies will automatically start a new build of your project. Jenkins understands all kinds of dependencies in POM. Namely,

- parent POM
- <dependencies> section of your project
- <plugins> section of your project
- <extensions> section of your project
- <reporting> section of your project

This process takes versions into account, so you can have multiple versions/branches of your project on the same Jenkins and it will correctly determine dependencies. Note that dependency version ranges are not supported, see https://issues.jenkins-ci.org/browse/JENKINS-2787 for the reason.

This feature can be disabled on demand - see configuration option *Build whenever a SNAPSHOT dependency is built*

Environment Variables (since 2.1)

Maven project type exposes the following environment variables, allowing you to use them as variable expansions in build configuration. The sampe values are the values of the variables if you are building this tree(https://github.com/jenkinsci/jenkins/tree/jenkins-1.536)

Name	Example	Meaning
POM_DISPLAYNAME	Jenkins main module	Taken from <name> in POM
POM_VERSION	1.536	Taken from <version> in POM
POM_GROUPID	org.jenkins-ci.main	Taken from <groupId> in POM
POM_ARTIFACTID	pom	Taken from <artifactId> in POM
POM_PACKAGING	pom	Taken from <packaging> in POM

The Maven Integration Plugin

The Maven 2 project type is contained in the Maven Integration plugin, which is bundled with Jenkins. If you are running any version of Jenkins earlier than the current release, it may show up in the list of plugins having available updates. It is recommended not to upgrade the Maven Integration plugin separately from Jenkins itself. While it is technically a plugin, it is developed, tested, and released as part of the Jenkins core.

Maven Surefire Test Results

The Maven Integration plugin understands the POM and knows about specific Maven testing plug-ins, such as org.apache.maven.plugins:maven-surefire-plugin, org.eclipse.tycho:tycho-surefire-plugin, etc. The definitive list of supported test plug-ins can be found at https://github.com/jenkinsci/maven-plugin/blob/master/src/main/java/hudson/maven/reporters/TestMojo.java. For such known test plug-ins the Maven Integration plugin is able to collect test results from their default or POM-configured reports directory. Test results from Maven test-capable plug-ins unknown to the Maven Integration plugin can be collected if their execution goal is 'test', 'test-run', 'integration-test' and they have a 'reportsDirectory' configuration property containing the location of their test results. Other than that results from unknown test plug-ins (such as org.codehaus.mojo:exec-maven-plugin) will not be collected, even if they are returned in the configured test results location (default target/surefire-reports).

Collecting Test Results from arbitrary test plugins

As of org.jenkins-ci.main:maven-plugin:2.13, there is a mechanism to inform the Maven Integration plugin of the location of test results produced by an unknown Maven test plug-in. If the POM declares a property with a name matching the pattern 'jenkins.<plugin-execution-id>.reportsDirectory', where <plugin-execution-id> is the unknown test plug-in's execution identifier, the Maven Integration plugin will resolve the value of this property against the project base directory and collect any test results that it finds in the resulting directory.

Example

In order to have Jenkins pick up the test results produced by an unknown plugin with a plugin execution ID 'e2eTests' which generates JUnit-compatible XML reports in the directory 'target/protractor-reports' in the project workspace, add the following property to your project POM:

```
<properties>
    <jenkins.e2eTests.reportsDirectory>target/protractor-reports</jenkins.e2eTests.reportsDirectory>
</properties>
```