mediadriver

Media Driver Consulting
Blog Post

DevOps: Maven Release Plugin

Author: Matt Pavlovich

October 14, 2016

About the author

Matt Pavlovich

Matt Pavlovich is a Founding Partner and Technical Practice Lead of Media Driver, a premier Solutions Provider and Systems Integrator with technologies for Integration, Business Process Management and DevOps. Media Driver provides consulting services, training courses and software products to assist organizations in their adoption of these technologies.

Matt directs the Media Driver Product Development Team. Media Driver products accelerate Enterprise application development while also increasing supportability. Media Driver tools provide a wide range of business users with greater access to Middleware solutions.

Matt has a lengthy resume of work within the Open Source Software community and is a Committer on the Apache ActiveMQ project. Matt is a hands-on technical leader who has led some of the largest and most well-known JBoss Fuse and JBoss A-MQ implementations in the world.

Background

Enterprise Customers often ask for measurable ways to show benefit by adopting Application DevOps (App DevOps) also known as Continuous Integration and Continuous Delivery (CI/CD). Much of the benefit of App DevOps (or CI/CD) is hard to quantify and most organizations do not have a line item in their time tracking system for activities such as *"fixing cowboy coder mistakes"* or *"up all night finding the correct source code revision to start writing the 1 line fix in order to get orders flowing again without unintentionally merging in a bunch of new code that has not been tested"*. Tracking total hours spent performing those activities over the course of a year across all development teams can be very difficult. Additionally, with a flood of Enterprise DevOps tools promising to deliver massive ROI, it is challenging for organizations to prioritize financial resources to solve problems that have nebulous value.

My recommendation in these cases is always the same—the biggest bang for your DevOps buck doesn't cost a dollar. Enterprises can realize immediate time savings by having an automated code artifact release process using the Maven Release Plugin. The benefits of adoption is that the Maven Release Plugin provides a straightforward way to measure the amount of time that is spent performing release tasks, while also eliminating the opportunity for human error during manual processes. Many enterprises dedicate a significant portion of their development time and resources to tasks related to code artifact release.

Automating these steps provides measurable value and increases quality by having a consistent and reliable release process. Additionally, having an automated code artifact release process is critical for organizations looking to adopt Micro Service Architecture or use Linux Containers (such as Docker or OpenShift), where the number of released code artifacts and release frequency will be greatly increased.

What is Maven Release Plugin?

The Maven Release Plugin automates common steps taken when releasing a version of a code artifact:

1. Update the version number to a stable release number
2. Create a “tag” in the source control management system (such as SVN or Git) to identify the point in time when the release occurred
3. Deploy released artifacts into a Maven Repository (such as Nexus or Artifactory) to make the code artifacts available for deployment to servers or to be used by other developers
4. Update the source code to the new version number for use in development

What’s with the blog posts complaining about the Maven Release Plugin?

The complaints I’ve seen about the Maven Release Plugin can be grouped into two core issues:

1. Version Numbering (attempting to use a non-Maven version numbering scheme)
2. Complexity in rolling back or “undoing” a release if there is an issue

Both of these issues can be traced back to same issue: the desire to have a specific version number at a given time. Adopting an organizational approach to match the behavior of the tool eliminates both of these issues and allows organizations to move forward very efficiently.

Many organizations have used a version numbering scheme for years. The most common process is to determine a target version number ahead of time and scheduling a specific version to release with specific

Sprint activity or Project intervals. While there is nothing inherently wrong with this practice, it complicates the process and does not work well with the version numbering inherent with the Maven Release Plugin.

The Maven Release Plug defines version numbers as

[major].[minor].[incremental]-[qualifier]

For example: The version number "3.0.1-SNAPSHOT" is used for all development build created ahead of a "3.0.1" release.

Understanding how Maven versioning works and how to best utilize it will eliminate a lot of common issues.

What's the Best Practice for versioning when using the Maven Release Plugin?

A stated goal should be that code artifacts move unchanged from QA to Production, limiting the risk of an error creeping into Production during the release cycle

1. Enterprise IT Projects should identify milestones as releases containing features or bug fixes, **and not focus on a specific version number.**

   Recommended Best Practice:
   Sprint 12 Release: Updates to include support for widget Y and bug fixes for widget X

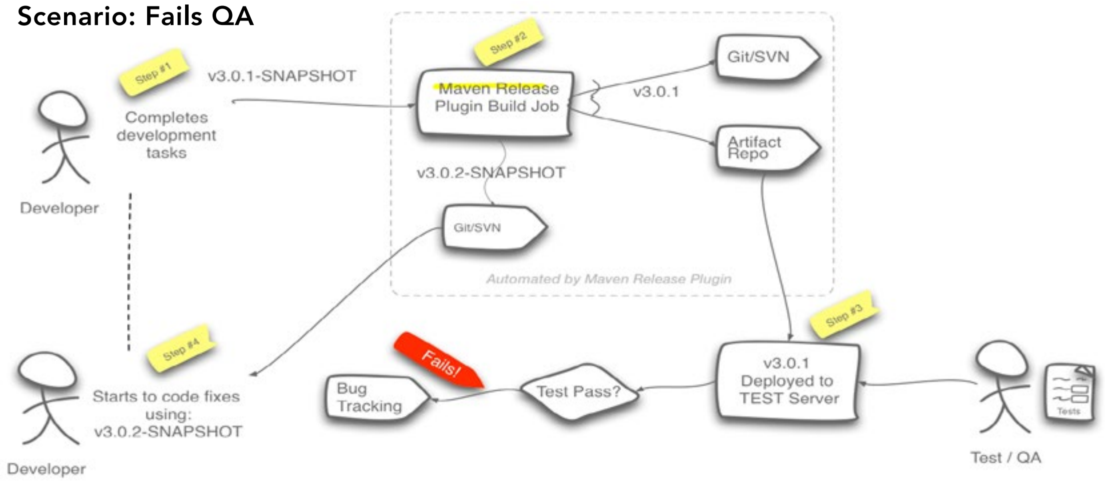
   Not Recommended:
   Sprint 12 Release: v3.0.1 updates

   Note: The version number should be an output of the Sprint activity or Project interval, not decided ahead of time.

2. Each release of a code artifact does not necessarily get deployed to production.

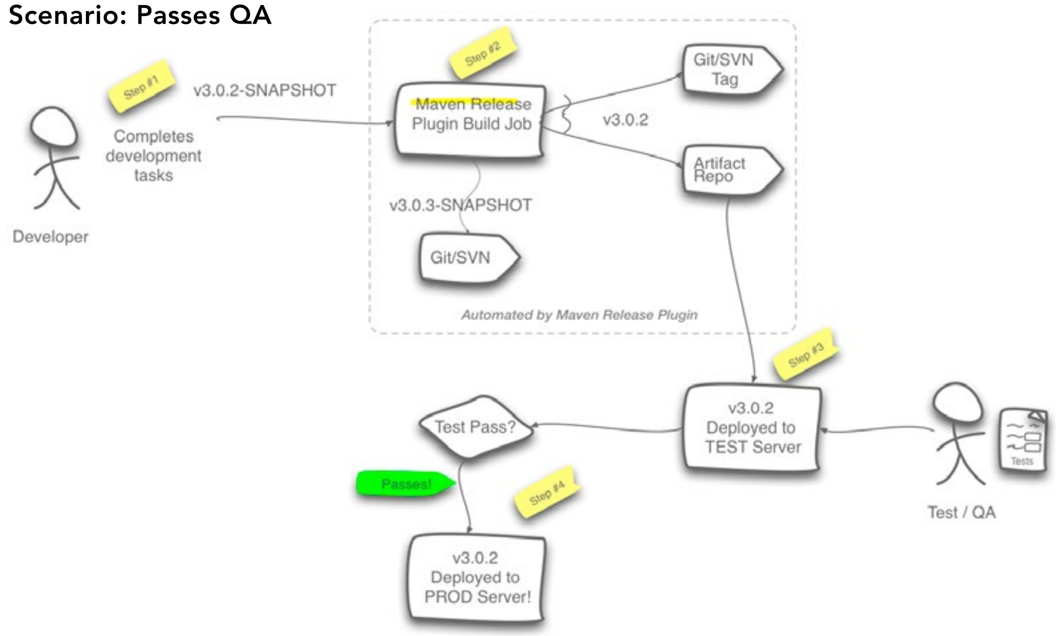
   In the examples below, when QA finds a bug in v3.0.1, development needs to provide a v3.0.2 release for certification testing. Once v3.0.2 passes QA, it would be suitable to release to Production, bypassing any production release of v3.0.1.

Scenario: Fails QA


Step #1
v3.0.1-SNAPSHOT
Completes development tasks
Developer
Step #2
Maven Release Plugin Build Job
v3.0.1
Git/SVN
Artifact Repo
v3.0.2-SNAPSHOT
Git/SVN
Automated by Maven Release Plugin
Step #3
v3.0.1 Deployed to TEST Server
Test / QA
Tests
Test Pass?
Fails!
Bug Tracking
Step #4
Starts to code fixes using: v3.0.2-SNAPSHOT
Developer

A stated goal should be that code artifacts move unchanged from QA to Production, limiting the risk of an error creeping into Production during the release cycle.

Scenario: Passes QA


Step #1
v3.0.2-SNAPSHOT
Completes development tasks
Developer
Step #2
Maven Release Plugin Build Job
v3.0.2
Git/SVN Tag
Artifact Repo
v3.0.3-SNAPSHOT
Git/SVN
Automated by Maven Release Plugin
Step #3
v3.0.2 Deployed to TEST Server
Test / QA
Tests
Test Pass?
Passes!
Step #4
v3.0.2 Deployed to PROD Server!

3. DevOps teams should create automated code artifact build jobs in order to standardize the process and not burden developers with having to become experts in the Maven Release Plugin.

   Below is a sample of standard build jobs that should be created:

Recommended Build Jobs for a code project called "widget.service":

widget.service-SNAPSHOT	This is the standard build for development tasks
widget.service-SNAPSHOT-no-tests	A fast build that by-passes unit testing
widget.service-RELEASE-dry-run	A build that simulates a release, but does not perform the steps. Should be run before the RELEASE builds in order to identify problems ahead of time.
widget.service-RELEASE	The release build
widget.service-SNAPSHOT-update-version	- A build that allows the user to enter the new version number for development. - This is needed when updating major or minor version numbers. For example moving from "3.0.4-SNAPSHOT" to "4.0.0-SNAPSHOT"

What are the gotchas?

Choosing not to utilize the Maven version numbering scheme will backfire, and not produce the benefits that could be attained. The Maven Release Plugin is best utilized with a standard Maven versioning scheme.

1. Development tasks occur with qualifier "–SNAPSHOT for example, 3.0.1-SNAPSHOT
2. Release tasks have clean version numbers: 3.0.1
3. Attempts at using non-Maven versioning schemes create more work and generally provide negative business value

What is hype?

For automated code release processes, the hype factor is zero. The benefits are very real when adopting an automated code artifact process. An automated release process is practically a requirement for organizations looking to adopt Micro Service Architecture and Containers. The Maven Release Plugin provides this with no upfront cost.

How do I ensure success?

1. Learn and adopt the Maven version numbering scheme.
2. Adopt a project plan or Sprint planning process that does not target a specific release version number, but rather a set of features or bug fixes.
3. Perform QA testing against released versions of artifacts, not "SNAPSHOT" development builds.

Summary

Enterprises need to transform and modernize their IT Applications in order to keep pace with competition and provide better value to customers. DevOps practices help enterprises stay competitive by providing measurable ways for organizations to scale their application development efforts through automation. The most impactful automation to implement with the least time commitment is an automated artifact release process. The Maven Release Plugin provides this automation with no upfront software costs.

For more blog posts, go to https://www.mediadriver.com/blog/

References

[1] Apache Maven Release Plugin website: http://maven.apache.org/maven-release/maven-release-plugin/

Media Driver
3824 Cedar Springs Road
Suite 430
Dallas, TX 75219

Phone	855-4-SOA-TEAM
Email	info@mediadriver.com
Web	www.mediadriver.com

Media Driver Consulting
Blog Post

mediadriver