Customizing Versioning Logic

This lesson explains how to implement a customized versioning logic using Jenkins X.

Sometimes we do want to use semantic versioning, but we want to add additional information. We usually do that by adding **prefixes** or **suffixes** to the three versions (*major*, *minor*, and *patch*). As a matter of fact, we are already doing that. So far, our pipelines use *"pure"* semantic versioning, except for GitHub releases. As a refresher, the step that creates release notes is `jx step changelog --version v\$(cat ../../VERSION)`. You can see that `v` is added as a prefix to the version.

Kubernetes release versioning

If we look at Kubernetes releases, we can see those like `v1.14.0-rc.1`, `v1.14.0-beta.2`, `v1.14.0-alpha.3`, and `v1.14.0-alpha.2`. They are using a variation of semantic versioning. Instead of increasing patch version with each release, the Kubernetes community is creating `alpha`, `beta`, and release candidate (`rc`) releases leading to the final `v1.14.0` release. Once `v1.14.0` is released, batch versions are incremented following semantic versioning rules until a new minor or major release.

How can we achieve this in Jenkins X?

Jenkins X does not support such a versioning scheme out of the box. Nevertheless, it should be a relatively simple change to `jenkins-x.yaml` to accomplish that or some other variation of semantic versioning. We won't go into practical exercises that would demonstrate that since I assume you should can make the necessary changes on your own.

One of the ways we could modify how versioning works is by changing `Makefile`. Instead of having a fixed entry like `VERSION := 1.0.0`, we could use a function as a value. Many of the variable declarations in the existing `Makefile` are getting dynamically generated values, and I encourage you to explore them. As long as you can script the generation of the version, you should have no problem implementing your own logic. Just as with custom prefixes and suffixes, we won't go into practical examples.

Finally, you might choose to ignore the existence of the `jx-release-version` and replace the parts of your pipeline that use it with something completely different.

The choice of whether to **adopt Jenkins X implementation of semantic versioning as-is**, to **modify it to suit your needs**, or to **change the logic to an entirely different versioning scheme** is up to you. Just remember that the more you move away from a standard, the more difficulty you will have in adopting tools and their out-of-the-box behavior.

Sometimes there is a good reason not to use conventions, while in others, we invent our own for no particular reason. If you do choose not to use semantic versioning, I hope you are in the former group.

Next, let's see how you can handle versioning using **Maven**, **Gradle**, or any other build and packaging tools.