How to Use Jenkins Less

Jesse Glick (CloudBees)

Jenkins World

A global DevOps event

2017

How to Use Jenkins Less

Jesse Glick
CloudBees, Inc.
@tyvole
@jglick on GitHub

Who would want to use Jenkins less?!

- admins: learn about a new way of evaluating features
- feature contributors: get a new perspective on value

Plugins! So many plugins!

On a good day

- So easy to get started! And inline help shows me how to select the right choices.
- Lots of features ready to use before I even anticipated needing them.
- Integrated smoothly with other plugins and Jenkins features.
- Jenkins update center constantly offers me improvements and fixes. I just “select all”!

On a not-so-good day

- This bug is a showstopper! Am I the only one to use this mode? Why did no one fix this yet?
- Someone here know enough Java to work on this?
- Did anyone take a look at my pull request yet? I need it to support the newest protocol.
- One team using my server is happier with the old version, can I just update the plugin for everyone else?

The “build vs. buy” questions

- Do you care more about getting started quickly, or long-term maintainability of your CI/CD system?
- Are your feature needs complex & idiosyncratic, or straightforward & common?
- Are the plugins you use maintained well? If not, do you have in-house expertise?
- Do you mind copying & pasting from stackoverflow.com?
- How big can your “attack surface” be?

Scoreboards

- 0: not integrated with Jenkins at all, but stable
- 5: some special features in use, but manageable
- 10: tight integration, but way too much baked in

Using Builders & Publishers Less

- OH: “How do I build a Ruby project in Jenkins?”

Steps vs. wrappers vs. project types

- many plugins offer build steps
  - some run unique functionality, others package up options
- “wrappers” set an environment for nested steps
  - details of how to run actual step left to job definition
- Pipeline jobs can interleave these in any order
  - many plugin freestyle features are `if` statements with a UI
- steer clear of dedicated project types!
  - CloudBees Jenkins Enterprise job templates: plugin-like UI

How do I run Maven in a Jenkins job?

- Pipeline project: `sh 'mvn install'`; `withMaven` wrapper
  - colorizing output? custom `settings.xml`? tool, JDK, publishers, snapshot deps?
- freestyle: shell step `mvn install`; Maven build step
- Maven project: obvious & convenient, but a bad choice:
  - nonstandard execution classpath, breaks some extensions
  - Jenkins classes in build JVM, including Remoting channel

Using Pipeline libraries

- alternative to DSLs
  - similar flexibility, but pulled by `Jenkinsfile`, not pushed by plugin
- available directly from SCM, as branch/tag/revision
  - extension point could support artifact repositories, etc.
  - possible to self-publish on GitHub
- “paired PR” tactic for proposing & testing changes

Demo: different ways of running Maven

https://github.com/jglick/jk--

Scoreboard

- `freestyle-shell, simple-sh: 2`
- `using-lib: 3`
- `freestyle-builder: 4`
- `withMaven: 6`
- `dedicated-project: 10`

Generic vs. specific publishers

- “notifiers” can be treated much like builders
  - example: mail or chat notification
  - for Pipeline, really there is no difference at all
- “recorders” more often integrate with Jenkins APIs
  - but some are more general than others
- some are hybrids (e.g., Testopia)
  - partly sends/receives data from an external server
  - also special build result, similar to JUnit/XUnit plugins

Demo: HTML Publisher vs. Javadoc

https://issues.jenkins-ci.org/browse/JENKINS-32619

Scoreboard

- `generic: 3`
- `javadoc: 6`

Using Groovy Less

What does Jenkins use Groovy for?

- standard Jenkins scripting language since 2006 at the latest
- extends Java syntax; familiar to many Java devs, less so to other users
- `/script` console, freestyle post-build, & many other uses
  - a custom resumable runtime used for Pipeline
- runs inside Jenkins JVM, can access same APIs as plugins

The pain of Groovy security

- Jenkins admins can run any script they like, but this does not scale
- so, enter the “sandbox”
  - most workaday scripts just run
  - but less-common API calls have to be approved before use
  - and fidelity to command-line semantics is imperfect
  - and this is a *very* complex language → regressions are common

And the pain of in-process scripting

- sandboxing is mandatory but adds runtime overhead
- unterminated loops, big allocations, etc. stop master from scaling
  - *Pipelines At Scale: How Big, How Fast, How Many?*
- scripts are tied to the master's version of Java and Groovy
- Job DSL vs. Jenkins Job Builder

Preferring external processes

- can run on an agent, taking load off the master
- container tech can apply quotas, networking rules, …
- each user picks the exact language and runtime they need
- embrace Docker!
- in-process vs. command-line API clients

Using SCMs Less

What does an SCM plugin do?

- check out / update sources (tool selection, credentials)
- generate changelog compared to previous build
- poll remote server for new changes
- scan for branches or repositories
- advanced:
  - identify “change requests” incl. metadata, trust status
  - “lightweight” checkouts
  - check out named revision (e.g., for a library)

What can you do with no SCM plugin?

- check out / update sources
  - `withCredentials` (or freestyle equivalent) to authenticate
  - `tool` (or freestyle equivalent) to select installation, if any
- that's all folks!

What about a meta-plugin?

- *Script SCM* plugin more or less works: polling, changelogs
  - no Pipeline support, only freestyle
  - you need to know Apache Ant (do you?) and Groovy (?)
  - oops, Groovy scripting here is **unsafe**
- *Shell Script SCM* works less: some polling, no changelogs
  - also no Pipeline support
  - shell scripts must be portable on all agents

Demo: userspace vs. dedicated Mercurial SCM

Scoreboard

- `userspace: 4`
- `dedicated: 7`

Using DSLs Less

OH: “A bug in this plugin is blocking me. You said this is the best way to use Docker!”

Pipeline DSL plugins

- some Pipeline integrations expose DSL beyond step definitions
- prettier `Jenkinsfile` usage, but
  - little or no **Pipeline Syntax** support
  - cannot use basic Pipeline APIs, need to depend on Groovy details
  - generally unusable from Declarative Pipeline

The saga of Docker Pipeline (part I)

- offers DSL rooted at `docker` to build & run Docker images
- `Image.inside`: major feature to run build steps inside container
  - used by Declarative Pipeline's `agent docker`
  - very convenient when it works
  - but has common, severe, and nonobvious limitations

The saga of Docker Pipeline (part II)

- some Credentials and “tool” (Docker CLI) integrations
  - also available as plain steps
- rest of DSL is sugar for `sh 'docker ...'` commands
  - some Jenkins fingerprint integration, but no consumer
  - subject of endless PRs, like tweaking args to `build`

Demo: DSL vs. hand-rolled

https://github.com/jenkinsci/docker-workflow-plugin/pull/105/files

Scoreboard

- original: 7
- revised: 2

Jenkins World

A global DevOps event

2017