Complex Build Workflows and Jenkins

Andrew Bayer, Cloudera Inc.

Introduction: Who Am I?

- Andrew Bayer
- Cloudera Kitchen team (QA, build, packaging, etc)
- Build architect
- Board member and committer for Jenkins
- Committer on Apache Bigtop, Apache Flume, Apache Sqoop, Apache Whirr, jclouds

Not all software builds can be contained in a single Jenkins job.

Nor should they!

If your build process needs to generate artifacts for multiple platforms, aggregate the output from other builds, etc...

You will probably end up with multiple Jenkins jobs.

Jenkins provides an amazing array of plugins and tools for tying those multiple jobs together into a coherent workflow

An Example

- Cloudera's CDH build
  - RPM/Debian packages for an array of Linux distributions/platforms.
  - Binary tarballs and source RPM/Debian packages.
  - Deployed Maven artifacts.
  - All that for 10+ component projects.
  - RPM/Debian package repositories for all of the above.

How We Used To Build

- One monolithic Jenkins job, calling a python script.
  - Check out all 10+ component's git repos and a parent repo with common makefiles, etc, using a tool similar to Android's repo.
  - Build source packages and deploy Maven artifacts.
  - Push source packages to S3.

How We Used To Build, continued

- Launch an EC2 instance for each platform
- Build native packages from the source packages on each EC2 instance.
- Push the native packages to S3
- Generate package repositories from the packages on S3

- Reminder: **all** of that is done through a single Python script.

Why We Changed

- Not possible to poll for changes, since we weren't using the Jenkins git plugin.
- No meaningful record of source code changes from build to build, for the same reason.
- No way to restart the build partway through – it's all or nothing.
- Can't build just one component or one platform – again, all or nothing.

Why We Changed, continued

- No visibility into build progress while on EC2 phase.
- Many, many distinct moving parts, glued together by confusing custom scripts.
- So many, many places things could go wrong, and so hard to fix when they do!

Our New Build Process

- Each component has a separate Jenkins job
  - Git repos cloned/checked out using Jenkins Multi-SCM plugin, not custom scripts
  - Component job build source packages, deploys Maven artifacts
  - Copies source packages to a binary staging area, and then calls generic native packaging builds for all needed platforms

Our New Build Process, continued

- Generic native packaging jobs
  - One for each platform
  - Parameters specify which component to build, and where to find the source packages
  - No need for component or even release specific behavior in native packaging jobs – just environmental setup and rpmbuild/debuild!
  - Built RPM/Debian packagings pushed to binary staging area.

Our New Build Process, continued

- After all generic native packaging jobs for a component finish, current package repositories updated with newly built packages
- Current package repositories always have the latest-and-greatest packages for all components in a release.

Our New Build Process, continued

- Full build of all components runs weekly
  - Builds components in dependency order, parallelizing when possible
  - Normal package repository update skipped
  - Instead, at end of full build, when all components have completed, new repositories are created

How The New Process Helps

- Individual components can be built on their own
- Component builds poll for changes, giving us automatic builds whenever code changes
- Full builds can be restarted from a failure point easily

How The New Process Helps, continued

- Separate Jenkins jobs for each step in the process make it much easier to see what's going on, what went wrong, where, etc.
- Parallelized full builds mean build time drops by ~25%, with more room for improvement to come.

How The New Process Helps, continued

- Widely-used Jenkins plugins used instead of custom scripts whenever possible, providing significant improvement in reliability.
- Cheaper than building on EC2!

How We Did It:

...Plugins, Plugins, Plugins!

Plugins We're Using

- Parameterized Trigger Plugin
  - Triggers downstream builds with parameters.
  - Allows us to have a single generic packaging job per platform, rather than one for each component/platform combo.

Plugins We're Using

- Parameterized Trigger Plugin, continued
  - Run downstream builds as a blocking build step.
  - Can run multiple downstream builds at the same time, with the same parameters.
  - Lets us have component and full build jobs that run downstream builds, wait until they complete, and then do further actions.

Plugins We're Using

- Conditional Build Step Plugin
  - Invoke different sets of build steps depending on conditions
  - Possible conditions include checks against parameters, time/day of week, and anything else using the Token Macro plugin's token functionality.

Plugins We're Using

- Conditional Build Step Plugin, cont.
  - Makes builds themselves scriptable, without having to use shell script build steps for all logic.
  - We use this to invoke our package repository update script with the Parameterized Trigger plugin, but only when a component build is not part of a full build.

Plugins We're Using

- jclouds plugin
  - Like EC2 plugin, but for any cloud API/provider supported by jclouds 1.5.
  - Spin up slaves on the fly from existing templates/images.
  - Can also create multiple instances as part of a build, tearing them down at end, for use in cluster testing, etc.

Plugins We're Using

- jclouds plugin, cont.
  - Saves us money and hassle by letting us spin up package build slaves on our internal cloud, rather than EC2.
  - In-development node pooling feature will allocate new slaves from a pool of already-running fresh instances, dropping provisioning time immensely.

Plugins We're Using

- Multi-SCM plugin
  - Use multiple SCM plugins or multiple instances of a single SCM plugin in an individual Jenkins job.
  - Full Jenkins SCM integration when using multiple git repositories in a single build, without needing to use something like git submodules.

Plugins We're Using

- Description Setter plugin
  - Sets the Jenkins build description based on regex matching in the build log.
  - Lets us mark what component was built in a generic packaging build easily and obviously.

Plugins We're Using

- Associated Files plugin
  - Marks files and/or directories outside of Jenkins archived artifacts as being connected to a build.
  - Records and displays staging area and package repositories for our builds.
  - With delete option enabled, associated directories will get deleted when the build gets deleted, simplifying cleanup.

Non-Jenkins-specific

- Not using S3 means we had to do our own binary staging area
  - Considered using S3 regardless, or using something like MongoDB or HDFS for storing files
  - Opted for simple NFS – less overhead, easier to access, etc
  - Not using Jenkins archiving due to size of artifacts – hundreds of megabytes, hundreds of files

Future Considerations

- CloudBees Jenkins Enterprise Template plugin
  - Define a job in XML, with Jelly or Groovy for variable replacement and logic
  - Jobs based on the template automatically inherit the template's job configuration, changing when the template changes
  - Would allow us to more easily have nearly identical jobs for multiple components and multiple releases with less maintenance

Future Considerations, cont.

- All Changes plugin
  - Aggregates change reports from downstream builds, including those launched via the Parameterized Trigger plugin
  - Would let us see all changes for a full build of all components in one place

Future Considerations, cont.

- Github plugin
  - Allows triggering builds via Github post-receive hooks, rather than polling
  - Kohsuke will tell you – that's better!