Large-Scale Automation with Jenkins

Kohsuke Kawaguchi / kk@kohsuke.org / @kohsukekawa
Architect, CloudBees, Inc.
Have you met Jenkins?

http://jenkins-ci.org/
x 53,000
x 600
Context

http://www.flickr.com/photos/spidermandragon5/2922128673/
Workflow!
Parameterized Builds

- Plain jobs can be thought of like a procedure without any input
  
  ```
  void buildAcmeLibrary() { ... }
  ```

- Ability to pass parameters make it more useful
  
  ```
  void buildAcmeLibrary(targetPlatform) { ... }
  ```
Three Things You Need To Do

• #1: Define Parameters

  - This build is parameterized
  - **String Parameter**
    - Name: Browser
    - Default Value: firefox
    - Description: The browser to run the tests with

Lots of different parameter types
Three Things You Need To Do

• #2: Refer to parameter values
  – As variable expansions: `${Browser}`
  – As environment variables from your builds
  – Some parameter types expose data in different ways
    • File parameter
Three Things You Need To Do

• #3: Specify actual values when you run

![Project foo](image-url)
Parameterized Trigger Plugin

https://wiki.jenkins-ci.org/display/JENKINS/Parameterized+Trigger+Plugin

• Call other jobs (with parameters)
  – Wait for their completions (optional)
Other Simple Choreography Tools

https://wiki.jenkins-ci.org/display/JENKINS/Join+Plugin

- Join Plugin
When Jobs Start Working Together…

https://wiki.jenkins-ci.org/display/JENKINS/Copy+Artifact+Plugin

- Copy Artifact Plugin
  - Copy artifacts into a workspace
  - By using various criteria
Copy Artifact vs External Repository

• Almost as if artifacts are versioned

- acme-build #10
- acme-build #11
- acme-build #12
- acme-build #13
- acme-build #14
- acme-build #15

foo.jar

Last saved build

Last stable build
Labeling Builds Is Useful

• Especially when labels have semantics
Labeling Builds Is Useful

• More so when you automate them
• Take “Signed off to QA” label for example
Introducing Promoted Builds Plugin

https://wiki.jenkins-ci.org/display/JENKINS/Promoted+Builds+Plugin

• Promotion = act of giving a build a label
• You specify:
  – Promotion criteria
  – what happens after promotion

• Label is a nice hand-off between teams
  – It’s like sausage making process
Maven Repository Plugin

https://wiki.jenkins-ci.org/display/JENKINS/Jenkins+Maven+Repository+Server

- Virtual Maven repository
  - Expose artifacts from specific build
  - And its upstream builds
Challenge: visualization
Challenge: Visualization

- Edge traversal breaks down on large workflow
Dependency Graph

https://wiki.jenkins-ci.org/display/JENKINS/Dependency+Graph+View+Plugin
Build Pipeline Plugin
https://wiki.jenkins-ci.org/display/JENKINS/Build+Pipeline+Plugin

• It shows how far each change has gone
Fingerprint

- Yet another angle to look at data
- Think of it as a lifelog for a blob

<table>
<thead>
<tr>
<th>Air travel</th>
<th>Jenkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airline check-in/TSA/immigration</td>
<td>Jobs and workflow</td>
</tr>
<tr>
<td>My going through immigration</td>
<td>Build #13</td>
</tr>
<tr>
<td>My travel experience</td>
<td>Fingerprint view</td>
</tr>
</tbody>
</table>
Fingerprint: Mechanism

- **MD5 checksum of a file**
  - Recorded against builds that it appeared
  - (And actions that were taken)
Fingerprint: Why?

• Track down where it came from
  – My component integrates to product XYZ, and a bug was reported against XYZ 3.0.5. Which build of the component did it contain?

• Cross-correlate jobs that aren’t directly related
Next Step in Workflow

- Aggregation of results

? Aggregation of definitions
Build Flow Plugin

https://wiki.jenkins-ci.org/display/JENKINS/Build+Flow+Plugin

- Groovy DSL for kicking builds
  - High-level primitives
  - Ability to define abstractions

```groovy
b = build("acme-build")
guard {
    parallel {
        { build("acme-test1", param1:b.number) },
        { build("acme-test2", param1:b.number) }
    }
} rescue {
    build("acme-teardown")
}
```
Jenkow Plugin

https://wiki.jenkins-ci.org/display/JENKINS/Jenkow+Plugin

• Embed BPMN workflow engine in Jenkins
  – Timeout, fork, join, ...

• Workflow is version controlled in Git
  – Push to Jenkins to load them up
Demo
Next Step in Workflow

✓ Choreography defined in one place

? Everything defined in one place
Job DSL Plugin
https://wiki.jenkins-ci.org/display/JENKINS/Job+DSL+Plugin

• Groovy DSL for defining jobs

```groovy
def project = "jenkinsci/jenkins"
def branchApi = new URL("https://api.github.com/repos/${project}/branches")
def branches = new JsonSlurper().parse(branchApi.newReader())
branches.each { b ->
    job {
        name "${project}-${b.name}".replaceAll('/', '-')
        scm {
            git("git://github.com/${project}-${b.name}.git", b.name)
        }
        steps {
            maven("install")
        }
    }
}
```
Or More Likely…

• Take Existing Job, Make Adjustments

```java
def project = "jenkinsci/jenkins"
def branchApi = new URL("https://api.github.com/repos/${project}/branches")
def branches = new JsonSlurper().parse(branchApi.newReader())
branches.each { b ->
  job {
    using "jenkins-build"
    name "${project}-${b.name}".replaceAll('/', '-')
    scm {
      git("git://github.com/${project}.git", b.name)
    }
  }
}
```
Job DSL Plugin

• You can go down to XML definitions

• The program itself executes as Jenkins job
  – Control over when it executes
  – Store definitions in VCS
Or just a bit of Perl/Python/Ruby scripts

• Programmatically CRUD jobs

```
$ ssh jenkins get-job foo \\
| sed -e 's/old.gitserver.com/new.gitserver.com/g' \\
| ssh jenkins update-job foo
```
Templates (in Jenkins Enterprise by CloudBees)

http://www.cloudbees.com/jenkins-enterprise-by-cloudbees-overview.cb

• Share some traits with Job DSL
  – Define job once, generate many variations
  – Update definition, and reflect it everywhere

• But different
  – Templates are defined in GUI, not in a program
  – Individual variations are manually updated by users
Conclusion

• Lots of useful building blocks for automating even more
  – That means many people are doing this

• Take your automation to the next level
Multi-Configuration Project

• You often do the same thing with slight variations
  – Compile C++ code for different platforms
  – Test with different browsers
  – Or more generally, think of it as

```java
for (x in [a,b,c]) {
    for (y in [d,e,f]) {
        doSomethingWith(x,y,...);
    }
}
```
Model

- Define axes
  - One axis ≈ one for loop

- Choose from pre-defined types of Axis
  - Generic axis: arbitrary values exposed as environment variables
  - Slave axis: pick slaves by their names or their labels
    - e.g., linux, solaris, and windows
  - JDK axis
Multi-Configuration Project Gimmicks

• Filtering
  – Otherwise combinations increase exponentially
  – Not all combinations make sense
  – Use boolean expression to trim down the size
    
    \[
    \text{(label=="windows").implies(browser=="iexplore") &&}
    \text{(label=="mac").implies(browser=="safari")}
    \]
  – Or tell Jenkins to cut the workload to N%
    • Jenkins will thin out the combinations by itself