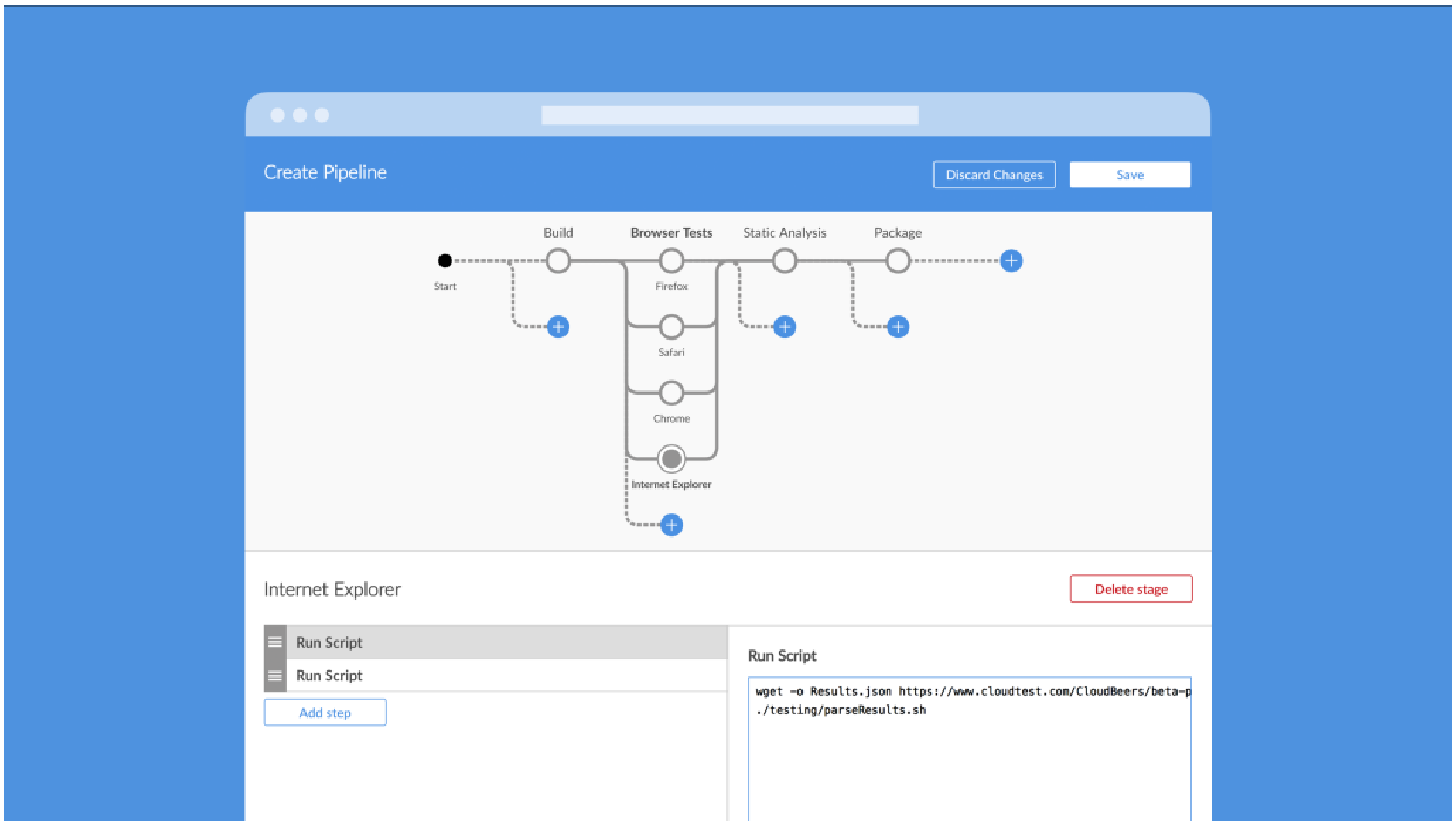
Ten Things I Wish I'd Known Before Using Jenkins Pipelines



```
wget -o Results.json https://www.cloudtest.com/CloudBeers/beta-p
./testing/parseResults.sh
```

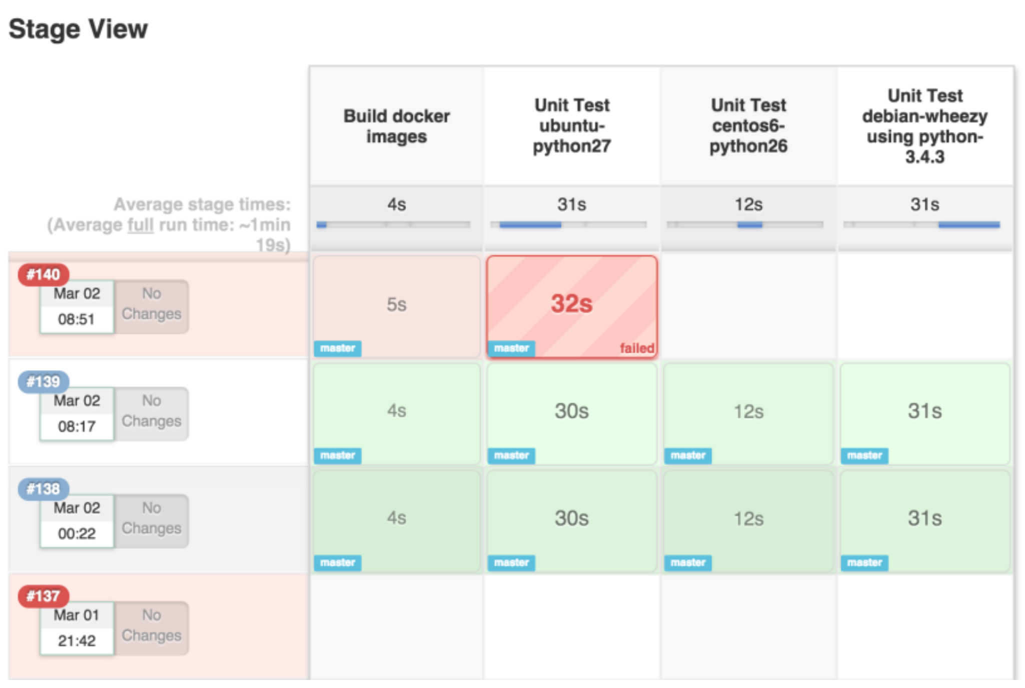
Jenkins Pipelines are becoming the standard way to programmatically specify your CI flow.

I started playing with Jenkins Pipelines using the web interface, then hit a block as I didn't really know the ropes.

Here's some things I wish I'd known first:

1) Wrap work in stages

Want those neat stages to show up in the Jenkins job homepage?


Stage View

	Build docker images	Unit Test ubuntu-python27	Unit Test centos6-python26	Unit Test debian-wheezy using python-3.4.3
Average stage times: (Average full run time: ~1min 19s)	4s	31s	12s	31s
#140 Mar 02 08:51 No Changes	5s master	32s master failed		
#139 Mar 02 08:17 No Changes	4s master	30s master	12s master	31s master
#138 Mar 02 00:22 No Changes	4s master	30s master	12s master	31s master
#137 Mar 01 21:42 No Changes				

Then wrap your stuff in stages, eg:

```
[...]
stage('setupenv') {
  node(nodename) {
    sh 'mkdir -p ' + builddir
    dir(builddir) {
      checkout([$class: 'GitSCM', branches: [[name: '*/master']], doGenerateSubmoduleConfigurations: false,
extensions: [[$class: 'SubmoduleOption', disableSubmodules: false, parentCredentials: false,
recursiveSubmodules: true, reference: '', trackingSubmodules: false]], submoduleCfg: [], userRemoteConfigs:
[[url: 'https://github.com/ianmiell/shutit']]])
    }
  }
}
stage('shutit_tests') {
  node(nodename) {
    dir(builddir + '/shutit-test') {
      sh('PATH=$(pwd)/..:${PATH} ./run.sh -s tk.shutit.shutit_test shutit_branch master -l info 2>&1')
    }
  }
}
[...]
```

It's a good idea to keep stages discrete, as they can be isolated from one another – for example, you could switch one stage to another **node** (see below) if you want (but then you might want to look into **stash()**ing files…).

2) Wrap Steps in a Node

All code that does steps in a pipeline should be wrapped in a node block:

```
node() {
  sh('env')
}
```

If the node is not specified, eg:

```
def myvariable='blah'
```

then by default it will run on the master.

You can specify the node by supplying an argument:

```
node('mynode') {
  [...]
}
```

If the pipeline code is not in a node block, then it's run on the master in some kind of lightweight node/thread.

3) Checkout scm doesn't work in browser UI!

This was a gotcha for me. 'checkout scm' is a great single line to add to your script that checks out the source the Jenkinsfile is taken from.

But when updating the script in the Jenkins browser interface this won't work!

When storing your source in source control, you can then switch to using 'checkout scm'. Otherwise use the 'git' function.

If you like this, you might like my book Learn Bash the Hard Way, available for $5:

---

4) You can use functions

Oh yeah, and **functions** are available to you too.

Handy, eg for seeing whether a node is available (see next tip)…

5) Use try / catch to dynamically decide what to do

Your code can be wrapped in a try/catch block.

I use this along with a **function** and **timeout()** to see whether a node is available before using it:

```
def nodetest() {
  sh('echo alive on $(hostname)')
}
// By default we use the 'welles' node, which could be offline.
usenode='welles'
try {
  // Give it 5 seconds to run the nodetest function
  timeout(time: 5, unit: 'SECONDS') {
    node(usenode) {
      nodetest()
    }
  }
} catch(err) {
  // Uh-oh. welles not available, so use 'cage'.
  usenode='cage'
}
// We know the node we want to use now.
node(usenode) {
  [...]
}
```

6) WTF is the Deal with Pipeline Syntax vs Groovy?

Pipeline syntax may be preferable to groovy, but is newer. See here:

https://jenkins.io/blog/2016/12/19/declarative-pipeline-beta/

The docs confusingly assume a familiarity with both, and it's not clear to the casual user why there's these two ways of doing the same thing.

7) It's Still a Bit Buggy

In one memorable evening I tried to change the branch a Jenkinsfile was pulled from, but the old branch persisted in my Jenkins build. It wouldn't pick up changes from the branch I'd change to.

I ended up having to create a new job and delete the old one.

Sometimes you have to kick off a job that fails in order to get the 'new' pipeline to be picked up by Jenkins.

8) Input

Want to force user input before continuing?

Simple:

```
input('OK to continue?')
```

But – seemed to work better for me when I had **defined stages first**!

9) Locking

If you're running jobs in branches and want to ensure they don't interfere with each other on the same Jenkins node, then locks are a simple way to ensure the serial running of jobs.

Here's an example from a Jenkinsfile I wrote:

```
 lock('cookbook_openshift3_tests') {
   stage('setupenv') {
     node(nodename) {
       sh 'mkdir -p ' + builddir
       dir(builddir) {
       ////when in source...
         checkout([$class: 'GitSCM', branches: [[name: '*/' + env.BRANCH_NAME]],
doGenerateSubmoduleConfigurations: false, extensions: [[$class: 'SubmoduleOption', disableSubmodules: false,
parentCredentials: false, recursiveSubmodules: true, reference: '', trackingSubmodules: false]],
submoduleCfg: [], userRemoteConfigs: [[url: 'https://github.com/IshentRas/cookbook-openshift3']]])
       }
     }
   }
[...]
}
```

10) Parameterized Jobs

These are the bomb. They allow you to parameterize your job so you can do more surgical builds when necessary. You can allow for defaults (like the BRANCH_NAME below), while giving users an interface to run their builds.

We use this internally to build test environments on demands.

Here's an extract from the same Jenkinsfile:

```
#!groovy
try {
  properties([parameters([
    string(name: 'BRANCH_NAME', defaultValue: env.BRANCH_NAME, description: 'Branch to build'),
    string(name: 'builddir', defaultValue: 'cookbook-openshift3-test-' + env.BUILD_NUMBER, description:
'Build directory'),
    string(name: 'nodename', defaultValue: 'cage', description: 'Node to build on'),
    string(name: 'CHEF_VERSION', defaultValue: '12.16.42-1', description: 'Chef version to use, eg 12.4.1-
1'),
    string(name: 'OSE_VERSIONS', defaultValue: '1.3 1.4 1.5', description: 'OSE versions to build, separated
by spaces'),
    string(name: 'CHEF_IPTABLES_COOKBOOK_VERSION', defaultValue: 'latest', description: 'iptables cookbook
version, eg 1.0.0'),
    string(name: 'CHEF_SELINUX_COOKBOOK_VERSION', defaultValue: 'latest', description: 'selinux cookbook
version, eg 0.7.2'),
    string(name: 'CHEF_YUM_COOKBOOK_VERSION', defaultValue: 'latest', description: 'yum cookbook version, eg
3.6.1'),
    string(name: 'CHEF_COMPAT_RESOURCE_COOKBOOK_VERSION', defaultValue: 'latest', description:
'compat_resource cookbook version'),
    string(name: 'CHEF_INJECT_COMPAT_RESOURCE_COOKBOOK_VERSION', defaultValue: 'false', description:
'whether to inject compat_resource cookbook version (eg true for some envs)'),
    booleanParam(name: 'dokitchen', defaultValue: true, description: 'Whether to run kitchen tests'),
    booleanParam(name: 'doshutit', defaultValue: true, description: 'Whether to run shutit tests')
   ])])
   lock('cookbook_openshift3_tests') {
     stage('setupenv') {
       node(nodename) {
[...]
     if (dokitchen) {
       stage('kitchen') {
[...]
```

Again, I had issues getting Jenkins to 'pick up' the fact that the job was parameterized when I added it to a Jenkinsfile

Examples

There are Jenkinsfile examples here but they look a bit unloved.

Some gists were more useful to me.

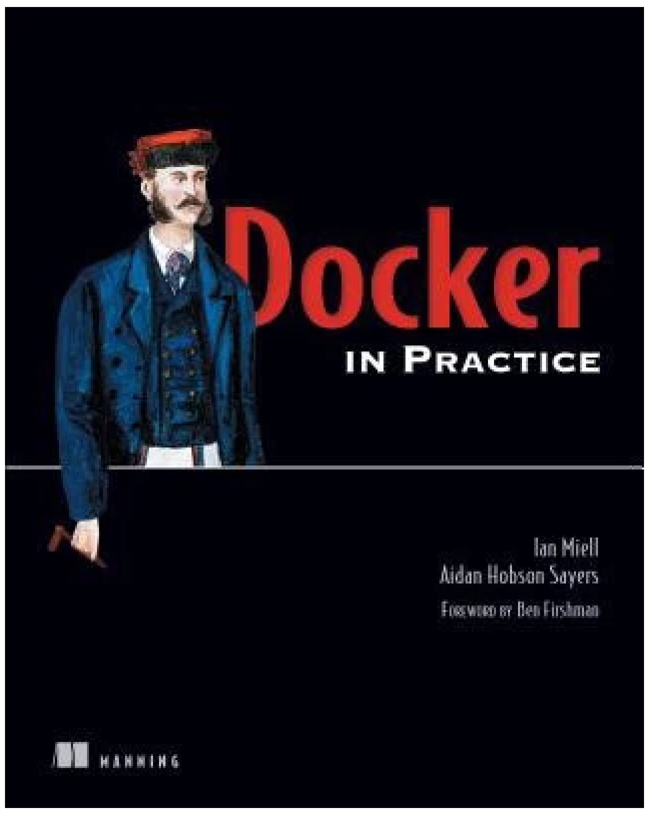
This intro was pretty good too.

And the canonical reference is here.

Docker in Practice

This is a work in progress from the second edition of Docker in Practice

Get 39% off with the code: 39miell2



Share this:

Email Share 307 20 points Tweet tumblr.

Related:

CI as Code Part I: Stateless Jenkins Deployments Using Docker

With 5 comments

CI as Code Part II: Stateless Jenkins With Dynamic Docker Slaves

With 6 comments

Ten Things I Wish I'd Known Before Using Vagrant

With 6 comments

zwischenzugs / April 23, 2017 / Uncategorized / CI, devops, jenkins

5 thoughts on "Ten Things I Wish I'd Known Before Using Jenkins Pipelines"