No, You Shouldn't Do That!
Lessons from Using Pipeline

Robert Sandell
James Nord

Jenkins World
2016

#JenkinsWorld

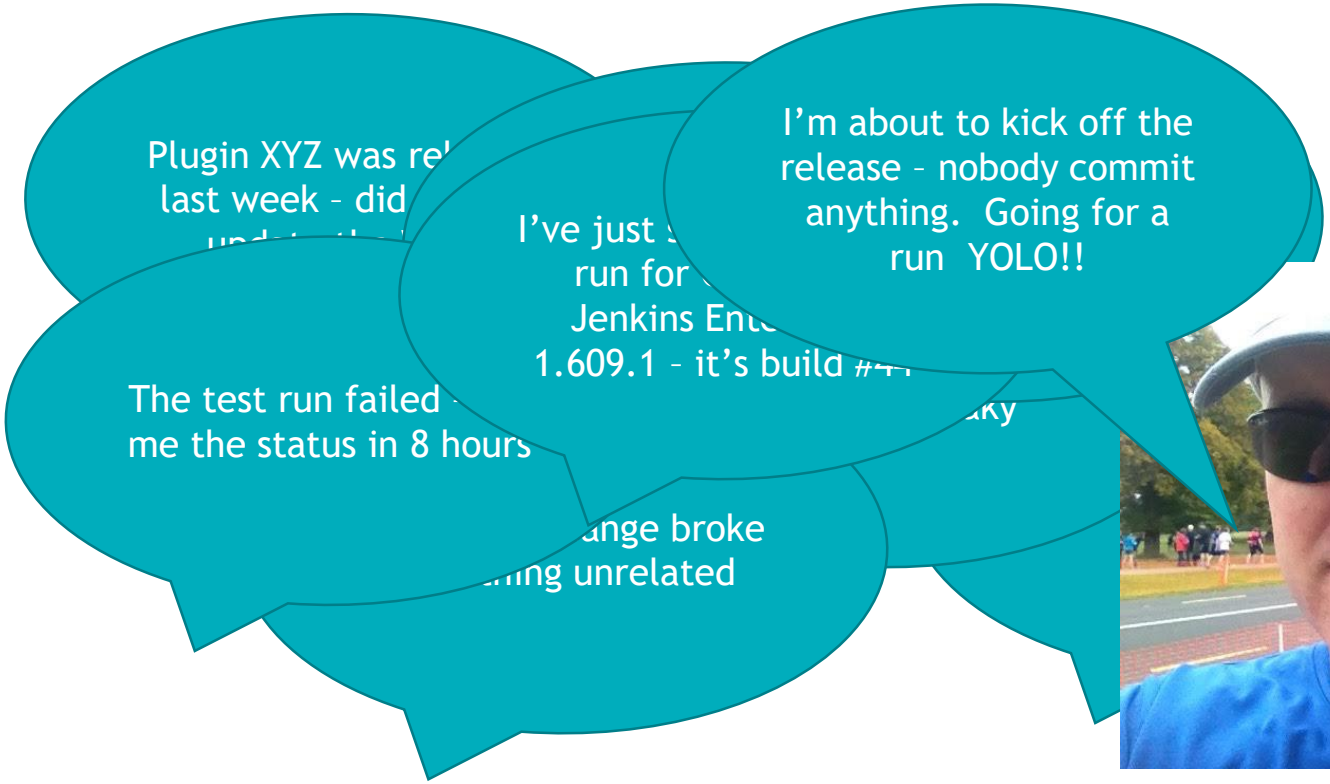
No, You Shouldn't Do That! Lessons from Using Pipeline

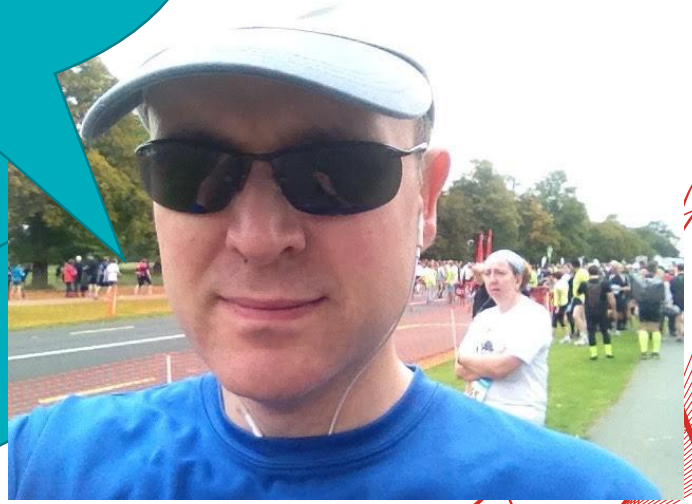
Robert Sandell, CloudBees, Inc.
James Nord, CloudBees, Inc.

#JenkinsWorld


No, You shouldn’t
do that!
NO SIGNAL

The Reality (mid 2015)





The Vision


Jenkins core change
OSS plugin change
Proprietary plugin change
Build product and run test suites

Run the tests, produce the installers.

Build Packages and Test
(3 products, 5 packages/product)

Plugins and dependencies (100+)			OSS Jenkins (2 + security 😒)			Test suites (3-5ish)		
release ☑	commit ☑	PR ☑	release ☑	commit ☑	PR ☑	Release ☑	commit ☑	PR ☑

And at the same time…

The Start

Requirements:

- Code review of the pipelines
- Re-use of the pipelines
- Running on our CI server (DEV@cloud)
- Need to support OSS plugins as well

Solution:

- GitHub
- Global pipeline library

The Start

But:

- The global library is not available on DEV@cloud
- Global Variables or passing `Objects`

Solution:

- Store all the flows and libraries on GitHub and have a common loader to load the correct flow for each job.
- Avoid religious wars during prototyping ☺

The Start


```groovy
withLib { def environment, def functions, def cjeFlow, def flow ->
  flow.doStandardBranchPipeline(environment, functions, cjeFlow, currentBuild, git_branch, cje_war_url)
}

def withLib(body) {
    stage "Load Lib"
    def environment
    def functions
    def cjeFlow
    def flow
    node {
        dir('lib') {
            deleteDir()
            git changelog: false, poll: false, url: 'git@github.com:cloudbees/cjp-champagne-lib.git', branch: 'master'

            environment = load 'lib/environment.groovy'
            functions = load 'lib/functions.groovy'
            cjeFlow = load 'flows/cje-war-pipelines.groovy'
            flow = load 'flows/cjoc-war-pipeline.groovy'
        }
    }
    body(environment, functions, cjeFlow, flow)
}
```

The Start

But:

[Pipeline Remote Loader Plugin]

```
stage 'Load a file from GitHub'
def helloworld = fileLoader.fromGit('examples/fileLoader/helloworld',
    'https://github.com/jenkinsci/workflow-remote-loader-plugin.git',
    'master', null, '')

stage 'Run method from the loaded file'
helloworld.printHello()
```

But this week:

```
@Library('somelib@1.0')
```

Story #1 The Native Installers...

https://github.com/jenkinsci/packaging

- Runs on multiple platforms (Debian, OSX, Windows)
- Runs linearly (builds one platform then the next)
- Needs snowflakes (including the Debian!)
- Uses distfork-plugin
- Need to be able to get the installers for testing (a web server)
- Need approval from the elite before uploading to production site

Solution:

- Change the installer creation scripts
- Use `node('some_label')` to run on arbitrary nodes
- Can do the builds in `parallel`
- Jenkins even has `ToolInstaller`s
- Jenkins is a web server!

Story #1 The native installers...

But:

- Changing the packaging scripts proved to be a mammoth task
  - The packaging scripts are heavily templated with `make` & `bash` and some `ruby` thrown in for good measure.
  - so was never completed ☹
- Custom tools plugin was not pipeline friendly

Story #1 The native installers...

So:

- Give up on being good, resort to being OK:
- Just run `make dist` -> get approval -> `make publish`...

But:

- We have no group support on our auth system
- we want more than 1 approver
  - `input` can take only take a single user or group.
- Dist-fork plugin ignores node properties (for `PATH` etc)
- `make publish` needs the files - otherwise it rebuilds everything again
  - so what you publish may not be what you tested

Story #1 The native installers...

So:

- Give up on being OK, resort to being pragmatic:
  - Snowflakes slaves 😱
  - Use `input` outside of the `node`.
    - Don't limit the approval to a single user
    - ask for a magic token (from a Jenkin's secret) and loop until someone guesses correctly 😱
  - Use `stash`/`unstash` to store and restore the workspace either side of the `input` to prevent rebuilding
  - Keep `make` happy (I don't want to build stuff again).
    - `sh 'find . -exec touch -d "`date`" {} \;'`

Story #1 The native installers...

Story #2 Build Orchestration

Rule#1

- Don’t put build logic in the pipeline

But:

- We need to control flow so we need some logic for that…
- And we need information from the build to utilize elsewhere in the pipeline
  - Version (war, plugin…)

Story #2 Build Orchestration

So:

- Can use `sh` steps and commands (`unzip`) with `readFile`

But:

- Windows

So:

- `if (isWindows()) { batch 'blah' } else { sh 'blah'}`

Story #2 Build Orchestration

But:

- `unzip` on windows?

So:

- Use `powershell`
- Install Cygwin

But:

- No `powershell` support
  1. Write a temp file (the `powershell`)
  2. Run a batch step with a command to load the `powershell` script
- I am the only one that knows `powershell`
- Cygwin = Snowflake servers
- Docker slaves?


Use shell steps!
NO SIGNAL

Story #2 Build Orchestration

So:

- When it is true utility functions
- When it is quick to run
- When it is just easier…

- Write a plugin with some custom steps
  - https://github.com/jenkinsci/pipeline-utility-steps-plugin/

Those forking test suites

Story #3 (forking test suite logic)

Needed to make the test suite faster

1. Run fewer tests
2. Make the tests run faster
3. Get faster hardware
4. **Run tests in parallel**

Started with option 4, looking at option 1.

At the same time DEV@cloud has moved to faster instances.
Soon we are moving from DEV@cloud to a dedicated CJOC cluster in AWS.

But we had 3 different test suites...

Story #3 ATH parallelisation

https://github.com/jenkinsci/acceptance-test-harness

Community project to run “black box” tests on Jenkins itself with Selenium. Could take more than 8 hours to run.

https://wiki.jenkins-ci.org/display/JENKINS/Parallel+Test+Executor+Plugin

The `splitTests` step analyzes test results from the last successful build of this job, if any. It returns a set of roughly equal “splits”, each representing one chunk of work. Typically you will use the `parallel` step to run each chunk in its own node, passing split information to the build tool in various ways.

Story #3 ATH parallelisation

But:

The JUnit publisher archives all test results it collects into one big list and adds to that list each time the step is run in a pipeline.

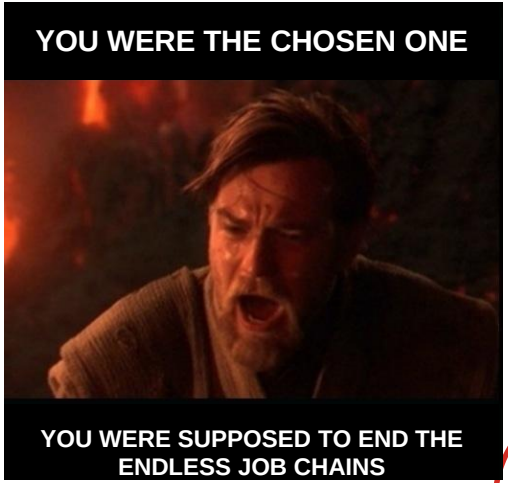
The Parallel Test Executor splits on that big list. Resulting in a big mash of splits from all the various test executions.

Solution:

A new pipeline Job for each Test suite 😒

Using the `build` step and copy artifacts, the traditional way.

But it has been said that matrix support is coming… for a long time...soon...ish...



Story #3 PCT parallelisation

Plugin Compat Tester - https://github.com/jenkinsci/plugin-compat-tester

Runs the tests of plugins against a specific core version to provide an indication of how well the plugin will work on that Jenkins version.
I've seen build times up to 16 hours on just the plugins we package in CJE.

Solution:

Write a similar algorithm as `splitTests`, but split on plugins instead of tests. The pipeline DSL is a Groovy script after all so I can code whatever I want, right!?

A short while later I had about 20 lines of Groovy code that did so...

- Find an archived report.xml from a previous build
- Copy it to the workspace with CopyArtifacts plugin
- Slurp it up and extract all the short-names
- Divide them up into equal size buckets and invert to blacklist
- Use the buckets as --excludes in a parallel step.

Story #3 PCT parallelisation

- XmlSlurper.parseText(xml).document.plugin.each {...}

Nope!

Story #3 PCT parallelisation

Annotating a function with `@NonCPS` will make pipeline to not run the function in GroovyCPS.

In that function you can then use each, collect and the other nice Groovy thingies.

But the code will still be processed by the sandbox*, so any quirks it has is still in effect.

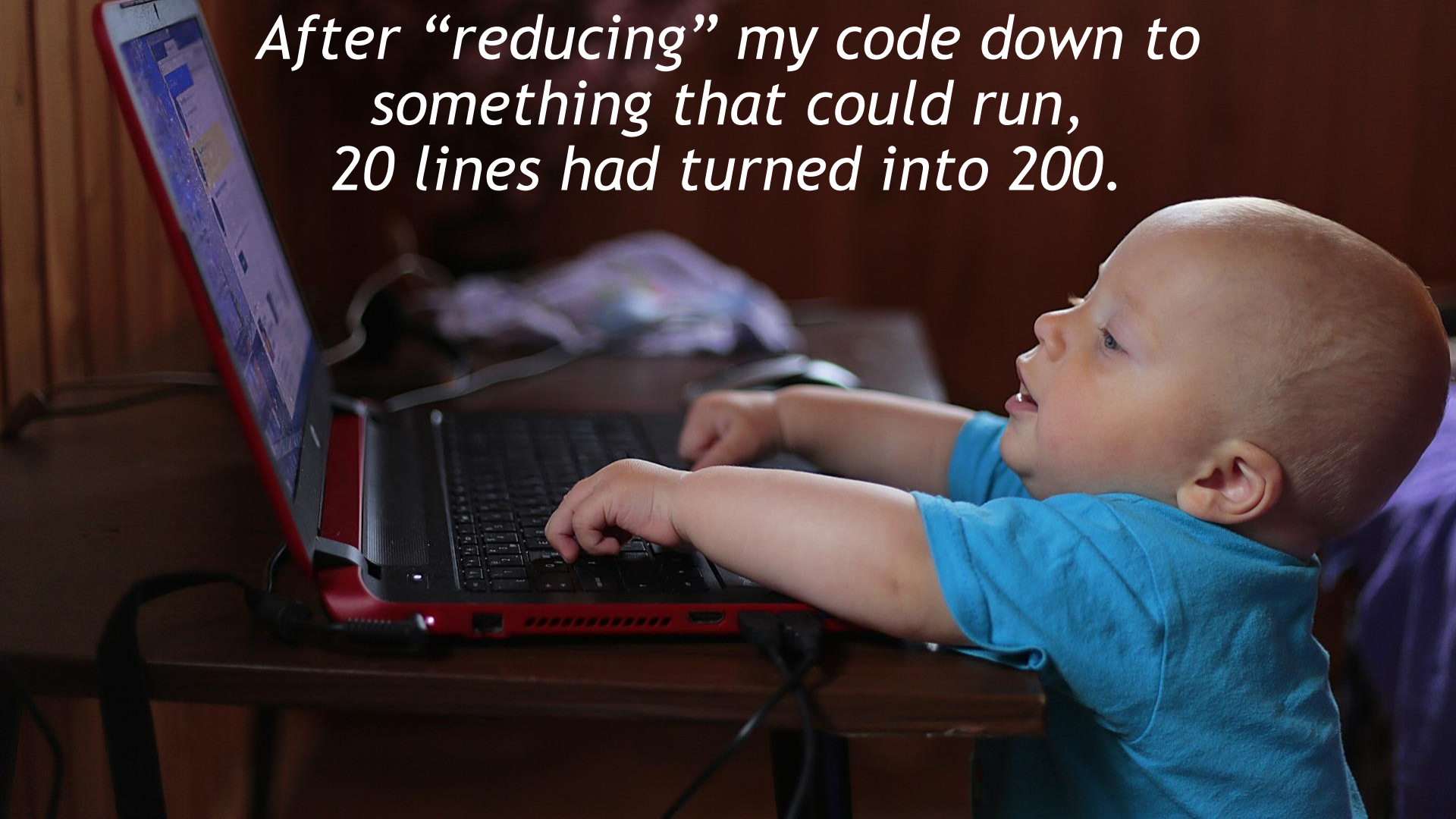
If you call any build steps from inside that function, the `@NonCPS` annotation will be “void”.

```
@NonCPS
boolean hasApples(String basket) {
  return basket.split(",").find {it == "apple"} != null
}

String basket = readFile file: "basket.csv"
if (hasApples(basket)) {
  dir("apples") {
    sh "mvn clean package"
  }
}
```

```
@NonCPS
boolean hasApples() {
  String basket = readFile file: "basket.csv"
  return basket.split(",").find {it == "apple"} != null
}

if (hasApples(basket)) {
  dir("apples") {
    sh "mvn clean package"
  }
}
```

#JenkinsWorld

**Unless it is a trusted global library, or you are running without the sandbox for some other reason.*


After “reducing” my code down to something that could run, 20 lines had turned into 200.

Story #3 PCT parallelisation

So..

- Don’t try to be fancy!
  - Pipeline is an orchestration layer first and foremost.
  - Keep logic in external scripts if possible.
  - When not possible, KISS!
- Multiple jobs are the current workaround for matrix
  - Don’t re-use these job – keep them triggered by the pipeline only

When Things Go Wrong….

And they will

Bug hunting, reporting

- No good saying the build failed at the end in an email
  - Your flow is not doing just one thing any more
  - Why did it fail and when
    - `try catch/finally`

- Track your library version
  - Use `changelog: true` with all your checkouts
  - `echo` the library version when loaded

- Report flaky tests so you can fix them (or remove them)
- Use smaller functions in your library to make it quicker to test

Optimisation

- Don't do `input`s inside a `node` - as you lock the executor.
- Similar with `timeout`s outside of nodes when script is coming back from restart
- `stage` “`concurrency`” -> great in theory in practice not expected behaviour. ☹
  - Use “`lock`”s to gate execution
  - Use `milestone`s to prevent build backlog; but it’s not available ☹
- Use `stash` over `archive` for moving files between `stage`s or `node`s.

So

Repeat after me

Lessons learned

- Convert what you have as is, *then* adapt and improve.
- Know your build environment and try to keep it consistent.
- Don’t try to be fancy, *unless you have lots of extra time*
  - **Keep the pipeline simple**, it is built to be an orchestration layer not a build script.
  - Put build logic in external scripts if possible, to avoid things like `@NonCPS` and method whitelisting problems.
  - If you *really* need to build libraries use the new shared libraries feature

Thanks for Listening

Jenkins World

2016

#JenkinsWorld

Questions?

Jenkins World

2016

#JenkinsWorld

Thank you!
DFTBA

Jenkins World
2016

#JenkinsWorld


Pipeline is awesome!
NO SIGNAL

Jenkins World

2016

#JenkinsWorld

Jenkins World

2016

#JenkinsWorld

Jenkins World

2016

#JenkinsWorld

Input approval


```
def magicValue
node {
  withCredentials([[$class: 'StringBinding', credentialsId: 'production_magic', variable:
  'tmpMagicValue']]) {
    magicValue = env.tmpMagicValue
  }
}

while (true) {
  def pass = input id: 'PushToProduction', message: 'Please enter the magic value for publishing to
  production', parameters: [[$class:
  'com.michelin.cio.hudson.plugins.passwordparam.PasswordParameterDefinition', defaultValue:
  'wibble', description: 'The magic token to show you have rights to push to production', name:
  'authentication token']]
  if (magicValue != pass) {
    echo "incorrect value entered"
  }
  else {
    echo "Push to production approved by magic"
    break
  }
}
```