Declarative Pipelines in Jenkins

Andrew Bayer and Robert Sandell

Jenkins World

A global DevOps event

2017

Who are we?

- We're both long-time contributors to Jenkins.
- We've both worked extensively on Declarative Pipelines, among many other things.
- And we're both engineers at CloudBees, working on making Jenkins better every day!

What is Pipeline?

- Pipeline as Code - define your build along with your project code, committed in your source control
- Scripted Pipeline lets you use nearly the full power of Groovy to script your Pipeline.
- Declarative Pipeline provides a more structured and opinionated syntax, allowing for validation and a visual editor

The Past, Present, and Future of Declarative Pipelines

Since Jenkins World 2016...

- Declarative Pipelines 1.0 in February 2017!
- First Blue Ocean Pipeline Editor beta in February 2017 too!

Since Jenkins World 2016...

- Declarative Pipelines 1.1 in March 2017
  - `anyOf`, `allOf`, `not` conditions for `when`
  - `libraries` directive for including shared libraries

Since Jenkins World 2016...

- Kubernetes `agent` support added

Since Jenkins World 2016...

- Frequent minor releases
  - Bug fixes
  - Polish to user experience
  - More test coverage

Coming Soon…

- Declarative Pipelines 1.2
  - Planned for shortly after Jenkins World
  - Much improved syntax for parallelization of `stages`
  - Complete rewrite of parser
  - Source control-provided variables included in environment automatically
    - When on Jenkins 2.60+ with newer versions of SCM plugins
  - `when` conditions using the SCM changelog

Parallel in Declarative before 1.2

```
pipeline {
    agent none
    stages {
        stage("foo") {
            steps {
                parallel(first: {
                    node("for-first-branch") {
                        echo "First branch"
                    }
                },
                second: {
                    node("for-second-branch") {
                        echo "Second branch"
                    }
                })
            }
        }
    }
}
```

Parallel stages!

```
pipeline {
    agent none
    stages {
        stage("foo") {
            parallel {
                stage("first") {
                    agent { label "for-first-branch" }
                    steps { echo "First branch" }
                }
                stage("second") {
                    agent { label "for-second-branch" }
                    steps { echo "Second branch" }
                }
            }
        }
    }
}
```

Parallel `stages!`

- A `stage` can have either `parallel` or `steps` - but not both!
- A `stage` inside a `parallel` cannot have `parallel` itself, only `steps.`
- A `stage` containing `parallel` cannot have `agent` or `tools` specified for it, since it doesn't do any actual work itself.

Parser rewrite

- Major behind the scenes changes to how we actually run a Declarative Pipeline.
- Huge improvements in `environment` and `when expression` variable and method handling - 9+ bugs related to `environment` alone fixed by the rewrite!
- More stable and maintainable going forward - less chance of weird regressions on variable handling.

Changelogs in `when`

- `changelog`
  - Examines the commit log messages with a regular expression.
- `changeset`
  - Examines the affected file paths in the changelog.

```
stage('some') {
    agent none
    when {
        changelog ".*Signed-off-by:\\s+(.*)$"
    }
    steps {
        ...
    }
}
```

```
stage('some') {
    agent none
    when {
        changeset "src/main/webapp/**/*"
    }
    steps {
        ...
    }
}
```

Changelogs in `when`

Caveats

- Jenkins must have first collected the changelog
  - e.g. `checkout scm` *(done by default)*
- Git plugin does not collect the changelog for the first build of a branch, including pull requests.
- Somewhat rigid logic

Changelogs in `when`

To somewhat fix that;

- For GitHub and BitBucket PRs*, we look at the full history for all builds.
  - This can cause some false positives - say, a commit in build #2 that gets reverted in build #3

** There is an extension point for that*

Declarative 1.2 beta available now!

1.2-beta-4 available in Experimental Update Center now

Give it a try, let us know if you have any problems

On The Roadmap

- No timeframe for any of this yet, but it’s all in the works.

- Restart a run from a failed stage
- Validation and inclusion in editor of shared library steps
- More Editor coverage of Declarative syntax
- Matrix!

Declarative 1.2 Demo/Showcase

Demo Expectations

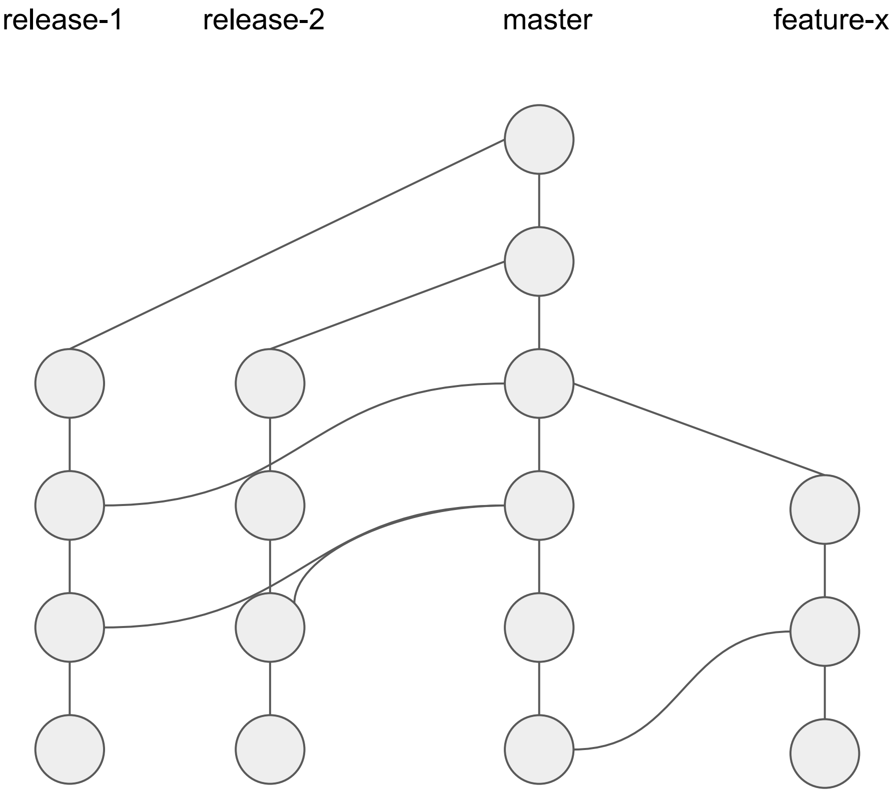
- NOT a CD Best Practices Demo
  - I’ve modeled the workflow and pipeline based on the cool features
- I’m not intending to throw you in the deep end
  - But we’re not going to splash in the kiddie pool either

Sample App

I picked a little sample application from JHipster.
It uses a couple of different technologies

Backend: Java & Maven
Frontend: Node.js & Gulp
Performance Tests in Gatling
Packaged as a Docker Container

Made up workflow



Planning Stage(s)

- Build
- Static Analysis
- Test
  - Unit
  - Performance
  - Frontend
- Deployment

Special Considerations

Our build resources are limited and the developers are impatient.
So I want to only run a limited set of tests on the feature branches and then more on master.

We need to handle deployment credentials in some secure way.

And we are cowards, so only deploy to production if a human says it’s ok .

Refined Plan

- Build Backend
- Test Backend
  - + Findbugs
- More Tests *(master & release)*
  - Frontend *(changes in UI)*
  - Performance
- Deploy to Staging *(master & release)*
- Deploy to Production *(release)*

Show me the code!

```
pipeline {
    stages {
        stage('Checkout') {
        }
        stage('Build Backend') {
        }
        stage('Test Backend') {
        }
        stage('Test More') {
        }
        stage('Deploy to Staging') {
        }
        stage('Archive') {
        }
        stage('Deploy to Production') {
        }
    }
}
```

Checkout

```
stage('Checkout') {
    agent any
    steps {
        checkout scm
        stash(name: 'ws', includes: '**', excludes: '**/.git/**')
    }
}
```

Checkout

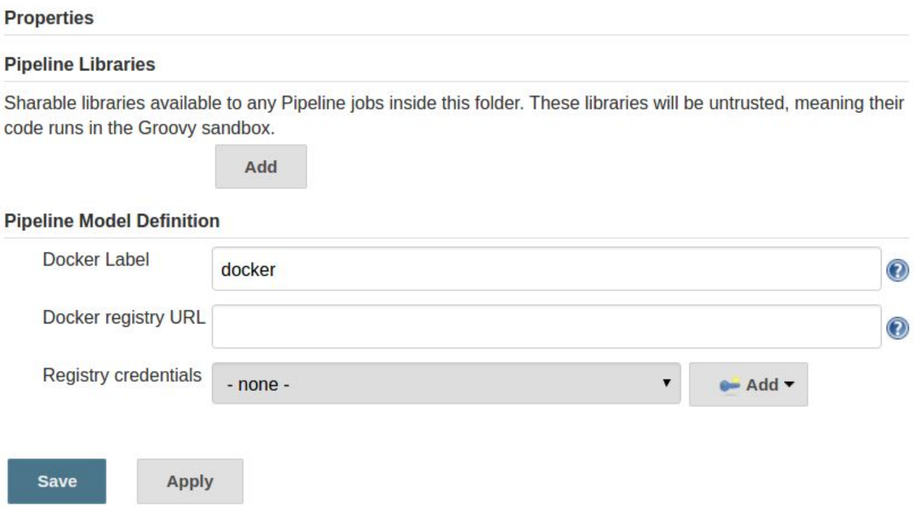
```
pipeline {
    agent none
    options {
        skipDefaultCheckout()
    }
    stages {
        stage('Checkout') {
            agent any
            steps {
                checkout scm
                stash(name: 'ws', includes: '**', excludes: '**/.git/**')
            }
        }
...
```

Build Backend

```
stage('Build Backend') {
    agent {
        docker {
            image 'maven:3-alpine'
            args '-v $HOME/.m2:/root/.m2'
        }
    }
    steps {
        unstash 'ws'
        sh 'mvn -B -DskipTests=true clean package'
        stash name: 'war', includes: 'target/**/*'
    }
}
```

Default Docker Label

Jenkins World
A global DevOps event
2017



Properties

Pipeline Libraries

Sharable libraries available to any Pipeline jobs inside this folder. These libraries will be untrusted, meaning their code runs in the Groovy sandbox.

Pipeline Model Definition

Docker Label: docker

Docker registry URL:

Registry credentials: - none -

Test Backend

```
stage('Test Backend') {
    agent {
        docker {
            image 'maven:3-alpine'
            args '-v $HOME/.m2:/root/.m2'
        }
    }
    steps {
        unstash 'ws'
        unstash 'war'
        sh 'mvn -B test findbugs:findbugs'
    }
    post {
        success {
            junit '**/surefire-reports/**/*.xml'
            findbugs pattern: 'target/**/findbugsXml.xml', unstableTotalAll: '0'
        }
        unstable {
            junit '**/surefire-reports/**/*.xml'
            findbugs pattern: 'target/**/findbugsXml.xml', unstableTotalAll: '0'
        }
    }
}
```

Test More

```
stage('Test More') {
    parallel {
        stage('Frontend') {

        }
        stage('Performance') {

        }
    }
}
```

Test More - Frontend

```
stage('Frontend') {
    when {
        anyOf {
            branch "master"
            branch "release-*"
            changeset "src/main/webapp/**/*"
        }
    }
    agent {
        dockerfile {
            args "-v /tmp:/tmp"
            dir "docker/gulp"
        }
    }
    steps {
        unstash 'ws'
        unstash 'war'
        sh '. target/scripts/frontEndTests.sh'
    }
}
```

Test More - Performance

```
stage('Performance') {
    when {
        anyOf {
            branch "master"
            branch "release-*"
        }
    }
    agent {
        docker {
            image 'maven:3-alpine'
            args '-v $HOME/.m2:/root/.m2'
        }
    }
    steps {
        unstash 'ws'
        unstash 'war'
        sh 'mvn -B gatling:execute'
    }
}
```

Deploy to Staging

```
stage('Deploy to Staging') {
    agent any
    environment {
        STAGING_AUTH = credentials('staging')
    }
    when {
        anyOf {
            branch "master"
            branch "release-*"
        }
    }
    steps {
        unstash 'war'
        sh '. target/scripts/deploy.sh staging -v $REL_VERSION -u $STAGING_AUTH_USR -p $STAGING_AUTH_PSW'
    }
    //Post: Send notifications; hipchat, slack, send email etc.
}
```

Update global environment

```
pipeline {
    environment {
        REL_VERSION = "${BRANCH_NAME.contains('release-') ?
BRANCH_NAME.drop(BRANCH_NAME.lastIndexOf('-')+1) + '.' + BUILD_NUMBER : 'M.' + BUILD_NUMBER}"
    }
...
```

Archive - for devs

```
stage('Archive') {
    agent any
    when {
        not {
            anyOf {
                branch "master"
                branch "release-*"
            }
        }
    }
    steps {
        unstash 'war'
        archiveArtifacts artifacts: 'target/**/*.war', fingerprint: true, allowEmptyArchive: true
    } //Perhaps push to a docker registry as well
}
```

Deploy to Production

```
stage('Deploy to Production') {
    agent none
    environment {
        PROD_AUTH = credentials('production')
    }
    when {
        branch "release-*"
    }
    steps {
        timeout(15) {
            input message: 'Deploy to production?', ok: 'Fire zee missiles!'
            node("linux") {
                unstash 'war'
                sh '. target/scripts/deploy.sh production -v $REL_VERSION -u $PROD_AUTH_USR -p $PROD_AUTH_PSW'
            }
        }
    }
}
```

Resources

- Documentation at https://jenkins.io/doc/book/pipeline/
- Some examples in https://github.com/jenkinsci/pipeline-examples/tree/master/declarative-examples

Resources

- Real world examples:
  - Mozilla’s shared libraries: https://github.com/mozilla/fxtest-jenkins-pipeline
  - Mozilla’s Declarative Pipelines:
    - https://github.com/mozilla/mozillians-tests/blob/master/Jenkinsfile
    - https://github.com/mozilla/treeherder/blob/master/tests/jenkins/Jenkinsfile
    - https://github.com/mozilla/Addon-Tests/blob/master/Jenkinsfile
    - and more!

Questions?

We’re also on the floor and at the Community Booth!

Thanks!

Jenkins World

A global DevOps event

2017