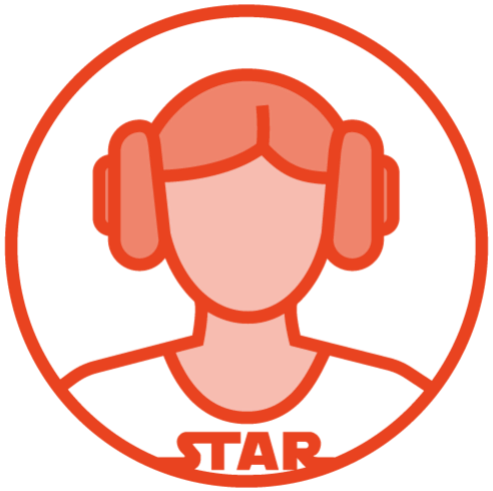
Creating Builds with Groovy

Chris B. Behrens
SOFTWARE ARCHITECT

@chrisbbehrens

Problem A



This is all in version control, right?

Problem B

Code locally

Commit changes to feature branch

Build and deploy to test environment

Commit to develop, promote to shared testing

Problem B

The build compiles a series of libraries

Your changes contain a new library for the build to compile

When do you modify the build?

The wrong answer breaks everyone else's work

The build should reflect the state of code on your branch

Demo

Where build configuration is stored

Get a grasp on how the rest of the information is stored

A way forward

Content is in version control

Solve our problems

The Solution: Jenkinsfile

Jenkinsfiles are Groovy scripts

Declarative and scripted

A Sample Jenkinsfile

```
node {
    stage('Build') {
        echo 'Building....'
    }
    stage('Test') {
        echo 'Testing....'
    }
    stage('Deploy') {
        echo 'Deploying....'
    }
}
```

A Sample Jenkinsfile

```
Node(‘DOTNET’) {
    stage('Build') {
        echo 'Building....'
    }
    stage('Test') {
        echo 'Testing....'
    }
    stage('Deploy') {
        echo 'Deploying....'
    }
}
```

Demo

Configure our pipeline build in Jenkins

Pulling from GitHub

Look at how the build gets executed

Dip our toe in making the build actually do something useful

Making Our Jenkinsfile Do Some Real Work

Our Workspace

docker exec -it 1005adeb0699f0188d35b14642071e927bf25472a88bbf8e92d4b006e3ef1872 /bin/sh

```
$ cd /var/jenkins_home/workspace/HelloPipeline
$ ls
1_config.groovy       music.json   researchlinks.txt
1_config.groovy.txt   jenkinsfile  scheduleBuild.groovy
$
```

SCM from Our Sample Maven Build

```
git 'https://github.com/jglick/simple-maven-project-with-tests.git'
```

Jenkins Step Documentation

Jenkins | Blog | Documentation | Plugins | Community | Subprojects | About | English | Download

Jenkins User Documentation
Home

Guided Tour

- Getting started
- Creating your first Pipeline
- Running multiple steps
- Defining execution environments
- Using environment variables
- Recording test results and artifacts
- Cleaning up and notifications
- Deployment

Tutorials

Pipeline Steps Reference

The following plugins offer Pipeline-compatible steps. Each plugin link offers more information about the parameters for each step.

Read more about how to integrate steps into your Pipeline in the Steps section of the Pipeline Syntax page.

- 360 FireLine Plugin
  - `step([$class: 'FireLineBuilder'])`: Execute FireLine
- AbsInt Astrée Plugin for Jenkins
  - `step([$class: 'AstreeBuilder'])`: Astrée Analysis Run
- AbsInt a³ Jenkins Plugin
  - `step([$class: 'A3Builder'])`: a³ Analysis Run
- Acunetix
  - `step([$class: 'BuildScanner'])`: Acunetix
- Acunetix 360 Scan Plugin
  - `NCScanBuilder`: Acunetix 360 Scan
- Agiletestware Pangolin Connector for TestRail
  - `pangolinTestRail`: Pangolin: Upload test results into TestRail

https://jenkins.io/doc/pipeline/steps

A Grab Warning

@grab does not work in pipeline scripts

Install a plug-in...more on this later

Demo

Whip up a quick .NET Core application

Commit it to our GitHub project

Modify our pipeline script to build it

Execute our build

Take a look at the results

A Useful Comparison

Build system works just like Jenkinsfiles

With YAML, instead of Groovy

More like the declarative format

Get the build into version control

Or else

Summary

Scripted Pipeline Builds

- Composed of Groovy statements
- Nodes
- Stages
- Steps within the stages

Echo build with stages

Building a simple .NET Core application

Stubbed error handling