CI Pipeline with Jenkinsfile/Jenkins 2.0

Jacob Aae Mikkelsen - Greach 2017

Agenda

- Jenkins
- Pipeline Fundamentals
- Defining Pipelines
- Editor Support
- Security
- Case Study: Implementing part of our pipeline
- Demo

Jacob Aae Mikkelsen

- Senior Engineer at LEGO
  - Microservice based architechture on JVM
- GR8Conf EU - Organizing Team Member
- External Associate Professor - University of Southern Denmark
- @JacobAae
- Blogs The Grails Diary

Jenkins

- Continous Integration
- Continous Delivery/Deployment

Continous Integration

Continuous Integration (CI) is the practice of merging all developer working copies to a shared mainline several times a day.

Continous Delivery/Deployment

Continuous delivery (CD) is an approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time.

Pipeline Fundamentals

Why

- Easy to **build all branches**
- CI Pipeline as **code** (stored with project in repo)
- Pipelines can survive restarts of the Jenkins master.
- Can optionally stop and wait for human input or approval
- Ability to fork/join, loop, and perform work in parallel.
- supports custom extensions to its DSL

Why Not

- Other CI systems do this as well
- Syntax confusion
  - Declarative Pipeline
  - Scripted Pipeline

Why

It Groovy!

Scripted Pipeline

Components

```
node { (1)
    stage('Build') { (2)
        sh './gradlew build' (3)
    }
}
```

1 **node** - allocate an executor and workspace for this part of the Pipeline.

2 **stage** - a stage of this Pipeline.

3 **step** using sh Execute shell command

Node

1. Schedules the steps in the Jenkins queue
2. Creates a workspace where work can be done on files checked out from source control.

Stage

Defining a conceptually distinct subset of the entire Pipeline

Stage

Stage View

	checkout	Unit Tests	Static Code Analysis	Coverage	Docker Release	Documentation	DEV Deploy	Integration Testing	Finish Deploy
Average stage times: (Average full run time: ~3min 14s)	1s	1min 15s	17s	11s	2min 28s	30s	53s	1min 28s	8s
#25 Jan 02 15:15 No Changes	3s master								
#24 Jan 02 14:50 1 commits	2s master	2min 34s master	22s master	10s master	3min 33s (paused for 20min 19s) master	50s master	58s master	2min 34s master	11s master
#23 Nov 14 11:40 No Changes	1s master								
#22 Nov 14 10:57 3 commits	1s master	48s master	10s master	11s master	2min 18s (paused for 39min 44s) master	29s master	51s master	1min 3s master	6s master
#21 Nov 14 09:27 No Changes	1s master								
#20 Nov 14 09:25 2 commits	1s master	49s master	11s master	11s master	2min 3s (paused for 19s) master	20s master	51s master	1min 17s master	7s master

Step

A single task

Fundamentally steps tell Jenkins what to do.

More steps exposed by various plugins

Declarative Pipeline

Components

```
pipeline {    (1)
    agent any    (2)
    stages {
        stage('Example') {     (3)
            steps {     (4)
                echo 'Hello World'
            }
        }
    }
    post {     (5)
        always {     (6)
            echo 'I will always say Hello again!'
        }
    }
}
```

Declarative vs Scripted

Scripted

- Scripted is real Groovy (almost)
- Very flexible
- Imperative programming model

Declarative

- Simpler and more opinionated syntax
- Smaller learning curve (for non-Groovy programmers)

Fundamentally the same

Writing Pipelines

Writing Pipelines

1. Script in web UI (pipeline project)
2. Storing Jenkinsfile in project

After writing it in UI, it should be stored with the project in VC

Snippet generator

Build in support for many steps.

Snippet generator

Jenkins

søg

Jenkins ▸ some-project-integration-tests ▸ Pipeline Syntax

- Back
- Snippet Generator
- Step Reference
- Global Variables Reference
- Online Documentation
- IntelliJ IDEA GDSL

Overview

This **Snippet Generator** will help you learn the Groovy code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Generate Groovy**, and you will see a Groovy statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

Steps

Sample Step: build: Build a job

Project to Build: some-project-integration-tests

☑ Wait for completion

☑ Propagate errors

Quiet period: 10

Parameters: some-project-integration-tests is not parameterized

Generate Groovy

```
build job: 'some-project-integration-tests', quietPeriod: 10
```

Global Variables

There are many features of the Pipeline that are not steps. These are often exposed via global variables, which are not supported by the snippet generator. See the Global Variables Reference for details.

Notable features

Not covered later

Parallel steps

```
parallel(
    longerTests: {
        sh "./gradlew functional-tests"
    },
    quickerTests: {
        sh "./gradlew unit-tests"
    }
)
```

Locking

```
lock(resource: 'staging-server', inversePrecedence: true)
```

Node labels

```
node('linux') {}
node('windows') {}
```

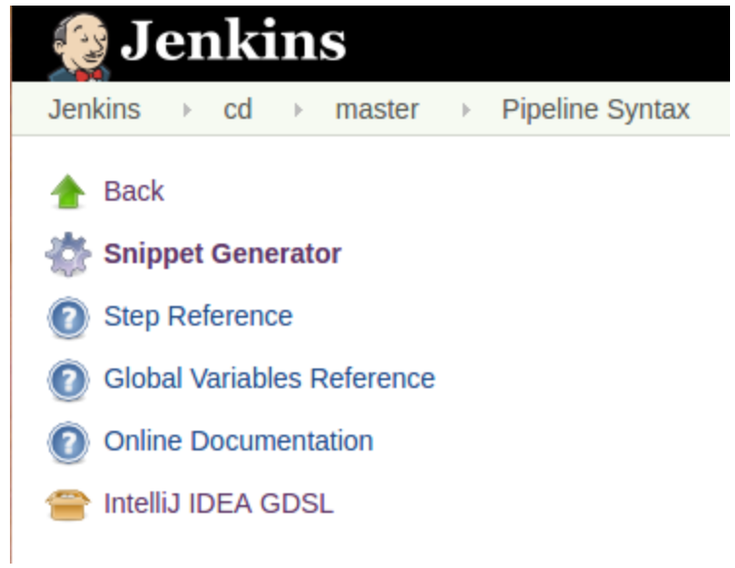
IntelliJ Support

IntelliJ Support

```
#!groovy
```

gsdl file

Place `jenkins.gdsl` file in `src/main/resources`



Autocomplete

```
} catch( Exception e) {
    currentBuild.result = "FAILURE"

    mai
    mail(Map args)                                  Object
    main(String[] args)                             void
Press Ctrl+Space to see non-imported classes >>
```

Only autocomplete for defined elements

Security

Missing Permissions

You will not have priveledges to write any code by default in the Jenkinsfile, but need to grant access for priveledged instructions

Plugin: Script Security Plugin

In-process Script Approval
Allows a Jenkins administrator to review proposed scripts
approval.

Easy overview

Approve / Approve assuming permission check / Deny signature : `method java.lang.Class getName`

Signatures already approved:

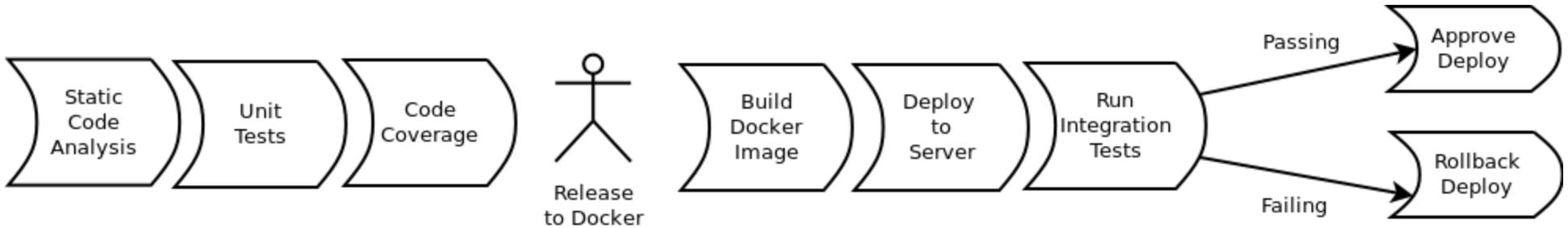
```
method hudson.model.Actionable getAllActions
method org.jenkinsci.plugins.workflow.support.steps.build.RunWrapper getRawBuild
method org.jenkinsci.plugins.workflow.support.steps.input.ApproverAction getUserId
staticMethod org.codehaus.groovy.runtime.DefaultGroovyMethods dump java.lang.Object
```

Global Library

Global libraries runs as trusted code, and have much less restrictions

Case Study - Deployment Pipeline

Pipeline


Static Code Analysis
Unit Tests
Code Coverage
Release to Docker
Build Docker Image
Deploy to Server
Run Integration Tests
Passing
Approve Deploy
Failing
Rollback Deploy

Code Checkout

Checkout code, and check for **ci-skip**

Code Checkout

```
node {
    stage "checkout"
        // get source code
        checkout scm

        // Respect ci-skip message
        def lastMessage = sh(returnStdout: true, script:'git log --format=%s -1')
        if (lastMessage.contains('ci-skip') ) {
            currentBuild.result = 'ABORTED'
            //error("Not building due to ci-skip")
        }
}

if( currentBuild.result == 'ABORTED') {
    return
}
```

Clean and Stash

```
stage 'Clean'
    sh "./gradlew clean"
    // save source code so we don't need to get it every time and also avoids conflicts
    stash excludes: 'build/', includes: '**', name: 'source'
```

Static Code Analysis

```
stage 'Static Code Analysis'
    unstash 'source'
    sh "./gradlew codenarcMain codenarcTest"
    publishHTML(target: [allowMissing: false, alwaysLinkToLastBuild: false,
        keepAll: true, reportDir:'build/reports/codenarc', reportFiles: 'main.html',
         reportName: 'Codenarc Main'])
    publishHTML(target: [allowMissing: false, alwaysLinkToLastBuild: false,
        keepAll: true, reportDir:'build/reports/codenarc', reportFiles: 'test.html',
        reportName: 'Codenarc Test'])
```

Unit tests

```
stage 'Unit Tests'
  sh "./gradlew test"

  // publish JUnit results to Jenkins
  step([$class:'JUnitResultArchiver', testResults:'**/build/test-results/test/*.xml'])

  // save coverage reports for being processed during code quality phase.
  stash includes: 'build/jacoco/*.exec', name: 'codeCoverage'
```

Coverage

```
stage 'Coverage'
    unstash 'source'
    unstash 'codeCoverage'
    sh "./gradlew jacocoTestReport"
    publishHTML(target: [reportDir:'build/reports/jacoco', reportFiles: 'index.html',
         reportName: 'Code Coverage'])
```

Error Handling

Above 3 steps are wrapped in try/catch

In the catch block we will sent an email with info

Error Handling

```
currentBuild.result = "FAILURE"

mail body: "Project build error: ${e}" ,
    from: 'jenkins@grydeske.com',
    replyTo: 'jenkins@grydeske.com',
    subject: "Initial test/coverage or codenarc failed for  ${env.JOB_NAME}"+
       " - Build # ${env.BUILD_NUMBER}",
    to: "${env.CHANGE_AUTHOR_EMAIL}"

throw e
```

User Input for release

Some user should approve if we will release this change as a docker image

This should be outside a node block, as we don't wan't to block a build executor in Jenkins

User Input

```
stage 'Docker Release?'
def inputValue
timeout(time:2, unit:'HOURS') {
    inputValue = input( message: 'Release Docker Image?', ok: 'Yes',
        submitterParameter: 'approver',
        parameters: [[$class: 'ChoiceParameterDefinition',
            choices: 'Minor\nPatch\nMajor', description: 'Major,Minor or Patch',
            name: 'releaseType']])
    echo "${inputValue.approver} ${inputValue.releaseType}"
}
```

User Input - Not long ago

```
@NonCPS
def getLatestApprover() {
    def latest = null
    def acts    // this returns a CopyOnWriteArrayList, safe for iteration
    acts = currentBuild.rawBuild.getAllActions()
    for (act in acts) {
        if (act instanceof
                org.jenkinsci.plugins.workflow.support.steps.input.ApproverAction) {
            latest = act.userId
        }
    }
    latest
}
```

Releasing a docker image

Gradle jobs for handling docker makes it simple

Releasing a docker image

```
unstash 'source'
sh "git checkout ${env.BRANCH_NAME} && git pull"
```

Releasing a docker image

```
switch( releaseType ) {
    case 'Patch':
        sh "./gradlew releasePatch"
        break
    case 'Minor':
        sh "./gradlew releaseMinor"
        break
    case 'Major':
        sh "./gradlew releaseMajor"
        break
    default:
        currentBuild.result = "FAILURE"
}
```

Deploy to Dev

Deploy the newly release docker image to DEV environment, and wait for it to be ready

Deploy to Dev

```
stage 'Deploy'
    sh "./gradlew deploy${inputValue.releaseType}"
```

Deploy to Dev

```
stage 'Wait til ready'
    timeout(time:5, unit:'MINUTES') {
        sh "./gradlew waitForDeploy"
    }
    sleep 5 // Wait for application to startup after deployment
```

Integration Tests

Run integration tests against the entire deployed stack with the new component

Integration Tests

```
stage 'Integration Tests'
    def integrationTestResult = build( job: '/demoapp-integration-tests/master',
        wait: true, propagate: false)
    echo "Result: ${integrationTestResult.result}"
```

Finish

Act accordingly on result of integration tests (finish upgrade or rollback)

Finish

```
stage 'Finalize'
    if( integrationTestResult.result == 'SUCCESS' ) {
        echo "Tests passed - finishing upgrade"
        sh "./gradlew -i approveDeploy"
    } else {
        echo "Tests failed with status ${integrationTestResult.result}"
        sh "./gradlew -i rollbackDeploy"

        currentBuild.result = "FAILURE"
        // TODO Send error message
    }
```

Global Libraries

Make Jenkinsfile DRY

Code can be shared, to avoid duplication

Global Library

Global Pipeline Libraries

Sharable libraries available to any Pipeline jobs running on this system. These libraries will be trusted, meaning they run without “sandbox” restrictions and may use `@Grab`.

Library

Name: global-lib

Default version: master

Currently maps to revision: c408b390a809f1fc4dd12871b5461d4680e1b0e7

Load implicitly: ☑

Allow default version to be overridden: ☑

Retrieval method

◉ Modern SCM

Source Code Management

◉ Git

Project Repository: /home/jacob/repositories/greach/jenkins-shared-libs

Credentials: - none - | Add

Ignore on push notifications: ☐

Repository browser: (Auto)

Additional Behaviours: Add

Advanced...

○ GitHub

○ Legacy SCM

Delete

Shared Lib

```
.
├── README.adoc
└── vars
    └── ciskip.groovy
```

Shared Lib

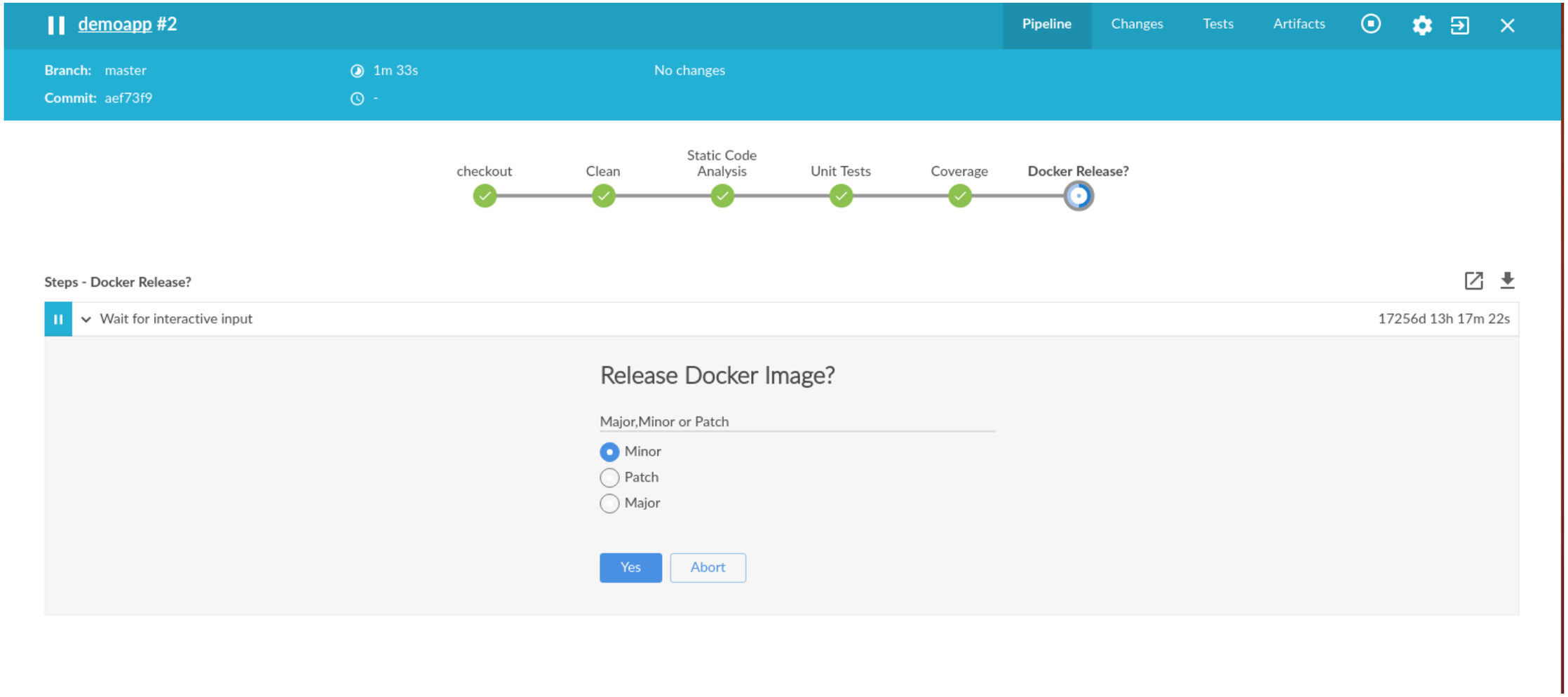
```
def call() {
    node {
        stage "checkout"
        // get source code
        checkout scm

        // Respect ci-skip message
        def lastMessage = sh(returnStdout: true, script: 'git log --format=%s -1')
        if (lastMessage.contains('ci-skip')) {
            currentBuild.result = 'ABORTED'
            //error("Not building due to ci-skip")
        }
    }
}
```

Code Checkout

```
ciskip()
```

Blue Ocean


demoapp #2
Pipeline
Changes
Tests
Artifacts
Branch: master
Commit: aef73f9
1m 33s
-
No changes
checkout
Clean
Static Code Analysis
Unit Tests
Coverage
Docker Release?
Steps - Docker Release?
Wait for interactive input
17256d 13h 17m 22s
Release Docker Image?
Major,Minor or Patch
Minor
Patch
Major
Yes
Abort

Resources

Try it Yourself

Docker demo:

```
docker run -p 8080:8080 -p 8081:8081 -p 9418:9418 -ti jenkinsci/workflow-demo
```

Go to http://localhost:8080/

Literature

- Getting started https://jenkins.io/doc/book/pipeline/
- Nice article with examples https://dzone.com/refcardz/continuous-delivery-with-jenkins-workflow
- Compatibility for plugins: https://github.com/jenkinsci/pipeline-plugin/blob/master/COMPATIBILITY.md
- Tutorial on Github https://github.com/jenkinsci/pipeline-plugin/blob/master/TUTORIAL.md
- Examples: https://github.com/jenkinsci/pipeline-examples/tree/master/pipeline-examples

Literature

- Steps exposed from plugins: https://jenkins.io/doc/pipeline/steps/
- Best practices: https://www.cloudbees.com/blog/top-10-best-practices-jenkins-pipeline-plugin
- Comparison: http://daanhorn.nl/post/jenkins-workflow/
- Docker tryout: https://jenkins.io/blog/2015/12/03/pipeline-as-code-with-multibranch-workflows-in-jenkins/

Questions