

# Important Prerequisites and Setup

- Have VirtualBox installed and ready

<https://www.virtualbox.org/>

- Download VM from

<https://www.dropbox.com/s/tptrgoc528exwu6/BDPJ2.ova?dl=0>

- See instructions at

<https://github.com/brentlaster/safaridocs/blob/master/BDPJ2-setup.pdf>

# Building a Deployment Pipeline with Jenkins 2

Brent Laster

© 2018 Brent Laster

2

# About me

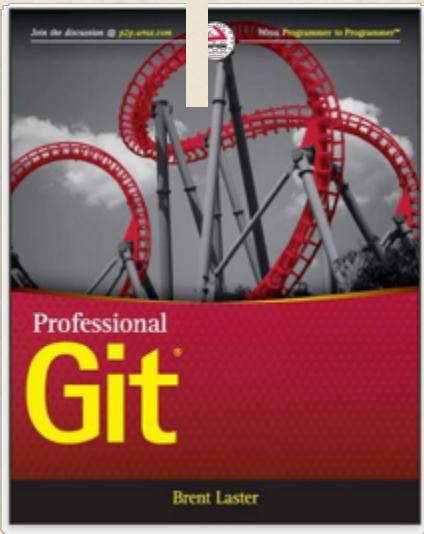
- Senior Manager, R&D
- Part-time trainer
- Git, Gerrit, Gradle, Jenkins
- Author - NFJS magazine, Professional Git book
- <https://www.linkedin.com/in/brentlaster>
- @BrentCLaster

## Professional Git 1st Edition

by Brent Laster • (Author)

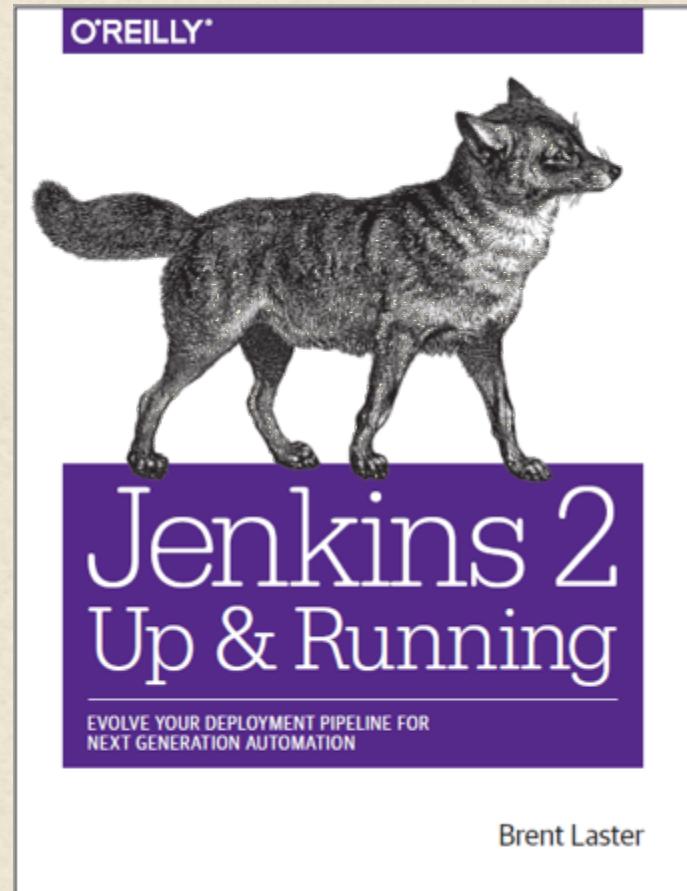
4.5 stars 1,000+ customer reviews

Look Inside

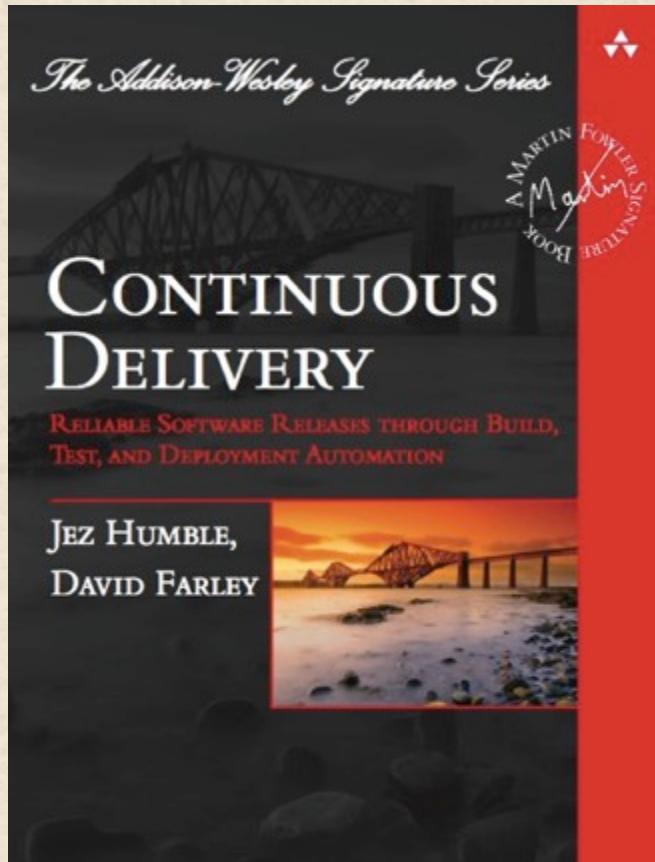


# Jenkins 2 Book

- Jenkins 2 – Up and Running
- O'Reilly – available April 2018
- Pre-order on Amazon



# Continuous Delivery – the book



“An extremely strange, but common, feature of many software projects is that for long periods of time during the development process the application is not in a working state. In fact, most software developed by large teams spends a significant proportion of its development time in an unusable state.”

# Continuous Delivery

## Goals

- Deliver useful, working software to users as quickly as possible
  - Low cycle time
    - » Get to users sooner
    - » Verify changes quickly
  - High quality
- Supported by frequent, automated releases
  - Repeatable
  - Discipline vs. art
  - Frequency means deltas between releases is small – reduces risk
  - Faster feedback

## Benefits

- Less risk
- Less stress
- Quicker ROI
- Improved response to business needs/challenges
- Improved quality
- Fail-fast
- Higher confidence

# Continuous Delivery

## Practice

- Theme is automation of software production process
- Combines 3 core practices/disciplines
  - Continuous Integration
  - Continuous Testing
  - Continuous Deployment (if desired)
- Includes Configuration Management



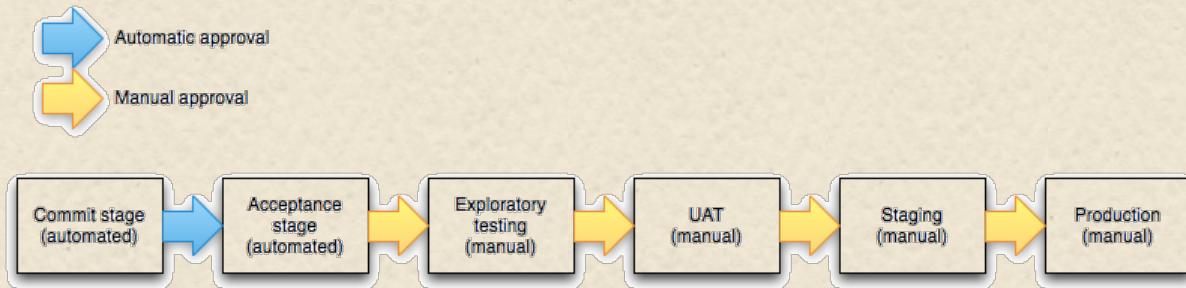
Carl Caum published on 30 August 2013

Continuous Delivery doesn't mean every change is deployed to production ASAP. It means every change is proven to be deployable at any time

— Carl Caum (@ccaum) August 28, 2013

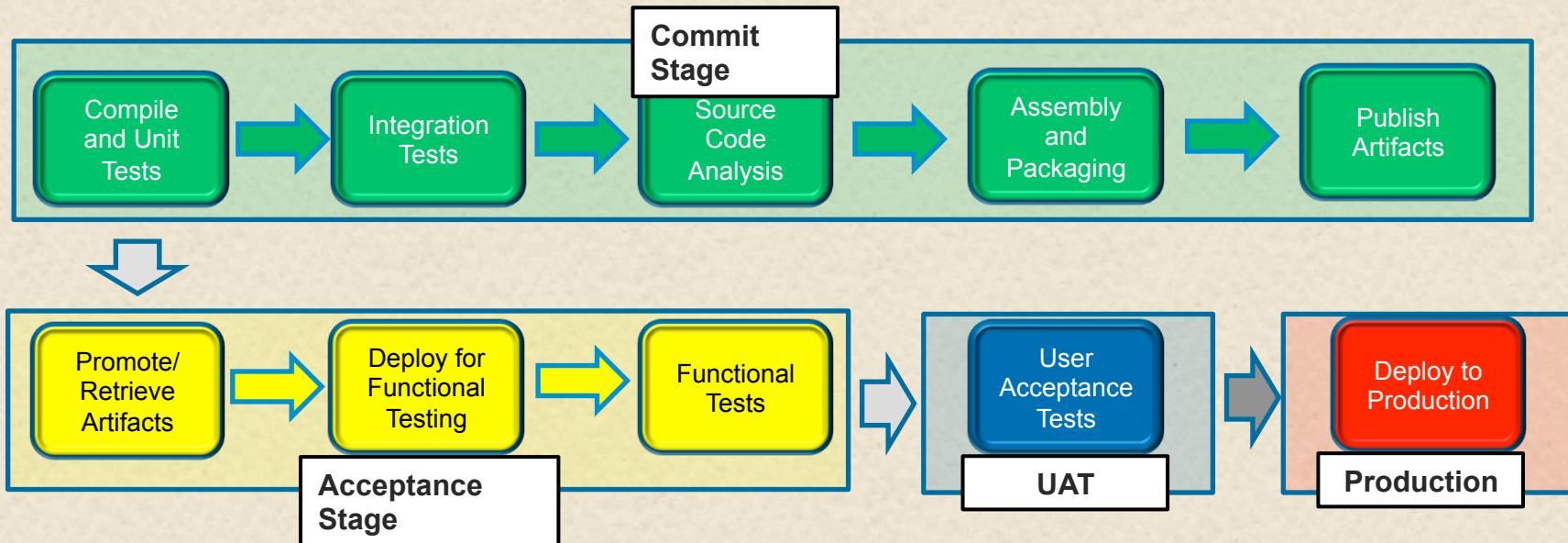
# Deployment Pipeline

- "... an automated implementation of your application's build, deploy, test, and release process.



- Every change made to configuration, source code, environment or data triggers a new instance of the pipeline.
- Change motivates production of binaries and then a series of tests are done to prove it is releasable.
- Levels of testing provide successive levels of confidence.

# Deployment Pipeline Stages



- **Approach**
  - Model steps in pipeline as stages in Jenkins pipeline script
  - Some Linux-specific implementations
  - Using Jenkins wherever possible
  - Using free versions of products
  - Light treatment of each area
  - Selected set of technologies

# What is Jenkins 2 (2.0+)?

- Features
  - Next evolution of Jenkins
  - Includes more integrated support for pipelines-as-code
    - Pipelines-as-code is not new with 2.0
  - Pipeline DSL improvements
  - Support for pipeline scripts stored in source control - Jenkinsfiles
  - Automatic project creation based on Jenkinsfile presence in branches
  - Improved DSL structure and processing via Declarative Pipelines
  - Advanced interface - Blue Ocean
  - Still supports FreeStyle
- Motivations
  - Treat pipelines as a first class citizen
  - Build support around them as a construct
  - Allow to express in coding
  - Use programming logic
  - Treat like code
    - » Store in SCM
    - » Reviewable
  - Easy to test
  - Text-based
  - Handle exceptional cases
  - Restarts

# Technologies used

- Jenkins - workflow management
- Git - source management
- Gradle - build engine and automation
- SonarQube - code analysis and metrics
- Jacoco - code coverage
- Artifactory - binary artifact storage & management
- Docker - container and image creation



**Jenkins**



# Home Screen -> Types of Projects

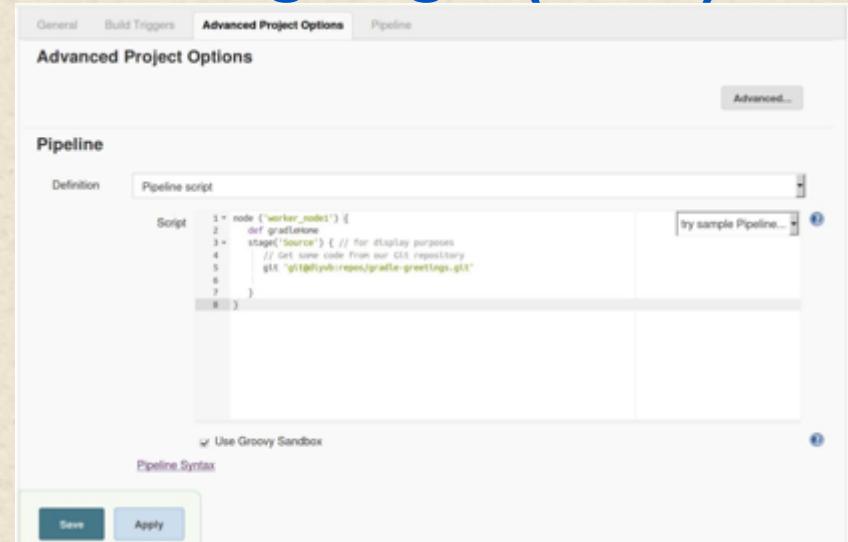
The screenshot shows the Jenkins home page at [localhost:8080](http://localhost:8080). The left sidebar includes links for New Item, People, Build History, Manage Jenkins, My Views, and Credentials. The Build Queue section indicates "No builds in the queue." The Build Executor Status section shows 1 Idle and 2 Idle executors. The main content area features a "Welcome to Jenkins!" message and a search bar. A central panel titled "Enter an item name" has "simple-pipe" typed into it. Below this, a list of project types is displayed:

- Freestyle project**: This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
- Maven project**: Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
- Pipeline**: Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project**: Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- Ivy project**: Build an ivy project. Hudson takes advantage of your ivy module descriptor files to provide additional functionality.
- External Job**: This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See the documentation for more details.
- Folder**: Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
- Github Organization**: Scans a Github organization (or user account) for all repositories matching some defined markers.
- Multibranch Pipeline**: Manages a set of Pipeline projects according to detected branches in one SCM repository.

At the bottom of the central panel, there are "ok" and "Add to current view" buttons.

# Jenkins Domain Specific Language (DSL) 13

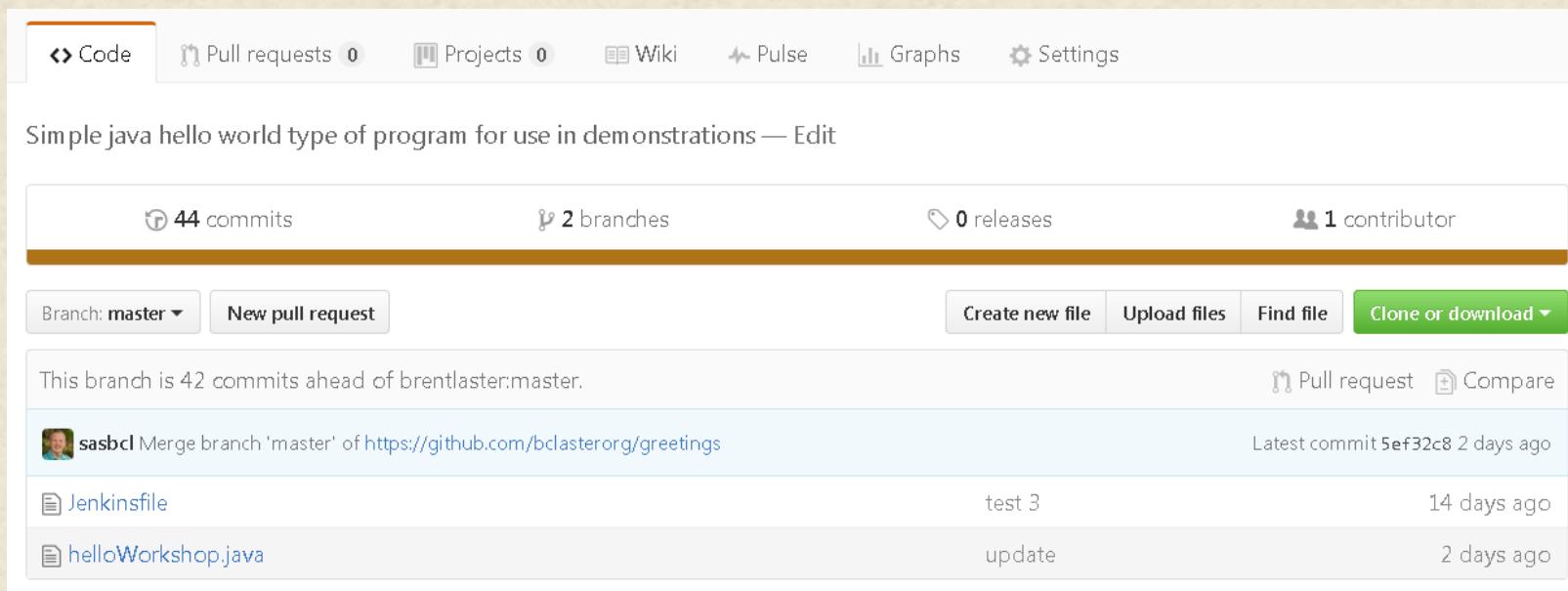
- Groovy-based
- Allows for orchestrating process steps
- Can be written and stored as pipeline script in job itself or as an external Jenkinsfile stored in the repository



The screenshot shows the Jenkins Pipeline configuration page. At the top, there are tabs for General, Build Triggers, Advanced Project Options, and Pipeline. The Pipeline tab is selected. Below the tabs, it says "Advanced Project Options". The main area is titled "Pipeline" and has a "Definition" section. Under "Definition", there is a "Pipeline script" section containing the following Groovy code:

```
1 node('worker_node1') {  
2     def gradleone  
3     stage('source') { // for display purposes  
4         git "git@github.com:pratik-greetings/greetings.git"  
5     }  
6 }  
7  
8 }
```

Below the script, there is a checkbox labeled "Use Groovy Sandbox" which is checked. Underneath the script, it says "Pipeline Syntax". At the bottom of the pipeline configuration are two buttons: "Save" and "Apply".



The screenshot shows a GitHub repository page for a Jenkinsfile. At the top, there are navigation links: Code (selected), Pull requests 0, Projects 0, Wiki, Pulse, Graphs, and Settings.

The main content area shows a message: "Simple java hello world type of program for use in demonstrations — Edit". Below this, there are summary statistics: 44 commits, 2 branches, 0 releases, and 1 contributor.

At the bottom of the commit list, there are buttons for Branch: master ▾, New pull request, Create new file, Upload files, Find file, and Clone or download ▾.

The commit list shows the following entries:

- This branch is 42 commits ahead of brentlaster:master.
- sasbcl** Merge branch 'master' of <https://github.com/bclasterorg/greetings> Latest commit 5ef32c8 2 days ago
- Jenkinsfile** test 3 14 days ago
- helloWorkshop.java** update 2 days ago

On the far right of the commit list, there are "Pull request" and "Compare" buttons.

# Working with the pipeline script editor

- Syntax error highlighting - not always valid

A screenshot of the Pipeline Script Editor interface. The code editor shows a pipeline script with several syntax errors highlighted by red boxes with a white 'X'. Line 20 has an error under the 'echo' command, and line 22 has an error under the 'stage' command. The code itself includes a try-catch block and a stage named 'Notify'.

```

16      }
17    }
18  } // end try
19  -> catch(err) {
20    |   echo "Caught: ${err}"
21  }
22  -> stage('Notify') {

```

- Hover over X to see explanation

A screenshot of the Pipeline Script Editor showing a tooltip for a syntax error. The tooltip text reads: "illegal string body character after dollar sign; solution: either escape a literal dollar sign "\\$5" or bracket the value expression "\${5}"". The cursor is hovering over the error icon on line 20.

```

18  } // end try
19  -> catch(err) {
20    |   echo "Caught: ${err}"
21
2 illegal string body character after dollar sign;
2 solution: either escape a literal dollar sign "\$5" or bracket the value expression "${5}"
24
25  }

```

- Automatic pair generation for () and {}
- Nice, but can trip you up
- Type “stage (“ and you get “stage()”
- Type node {, hit return, and closing } is auto-generated

A screenshot of the Pipeline Script Editor illustrating automatic brace generation. On the left, the user types 'node {' on line 19. On the right, the editor automatically completes the line with a closing brace '}' on line 21.

```

18
19 -> node {
20
21
22
18
19 -> node {
20
21   }
22

```

# Working with the pipeline script editor

- Quotes are significant; Double check after copy and paste from Word or PDF
- Rule of thumb: “If the text isn’t green, the quotes aren’t clean.”

```
stage('Compile') {
    // Run gradle to execute compile
    gbuild3 'clean compileJava -x test'
}
```

vs.

```
stage('Compile') {
    // Run gradle to execute compile
    gbuild3 "clean compileJava -x test"
}
```

- Blocks can be collapsed/expanded

```
1 @Library('Utilities2') _
2 node('worker_node1') {
3     stage('Source') {
4         // Get code from our git repos
5         git 'git@diyvb2:/home/git/repo'
6     }
}
```

```
1 @Library('Utilities2') _
2 node('worker_node1') {
3     stage('Source') { ... }
4     stage('Compile') {
5         // Run gradle to execute compile
6         gbuild3 'clean compileJava -x test'
7     }
}
```

```
1 @Library('Utilities2') _
2 node('worker_node1') { ... }
3
```

# Jenkins Pipelines – Scripted vs. Declarative

- Scripted
  - Nodes
    - » Stages
      - » DSL
      - » Groovy code
  - Use any legitimate groovy code
  - Groovy error checking/reporting
  
- Declarative
  - Agent
    - » Stages
      - » Stage
        - Steps
          - » DSL
    - Predefined Sections
    - No imperative logic
    - DSL-oriented checking/reporting
    - Direct Blue-Ocean compatibility

```

1 // Scripted Pipeline
2 node('worker') {
3   stage('Source') { // Get code
4     // Get code from our git repository
5     git 'git@diyvb2:/home/git/repositories/workshop.git '
6   }
7   stage('Compile') { // Compile and do unit testing
8     // Run gradle to execute compile and unit testing
9     sh "gradle clean compileJava test"
10  }
11 }
12
13
14

```

```

1 pipeline { // declarative pipeline
2   agent label:'worker'
3   stages {
4     stage('Source') { // Get code
5       steps {
6         // Get code from our git repository
7         git 'git@diyvb2:/home/git/repositories/workshop.git '
8       }
9     }
10    stage('Compile') { // Compile and do unit testing
11      steps {
12        // Run gradle to execute compile and unit testing
13        sh "gradle clean compileJava test"
14      }
15    }
16  }
17 }

```

# Terminology - Node

- Binds code to a particular system (agent) to run via a label
- Has defined number of executors - slot for execution
- As soon as an executor slot comes open on that agent, the task is run
- Code between {} forms program to run
- A particular agent can be specified in node(agent)
- Creates an associated workspace (working directory) for duration of the task
- Schedules code steps to run in build queue
- Best practice: Do any significant work in a specific node
  - Why? The default is for a pipeline script to run on the master with only a lightweight executor - expecting limited use of resources.

```
node {  
    // stages  
}  
  
node ('agent_1') {  
    // stages  
}
```

```
parallel (  
    win: { node ('win64') {  
        ...  
    }},  
    linux: { node ('ubuntu') {  
        ...  
    }},  
)
```

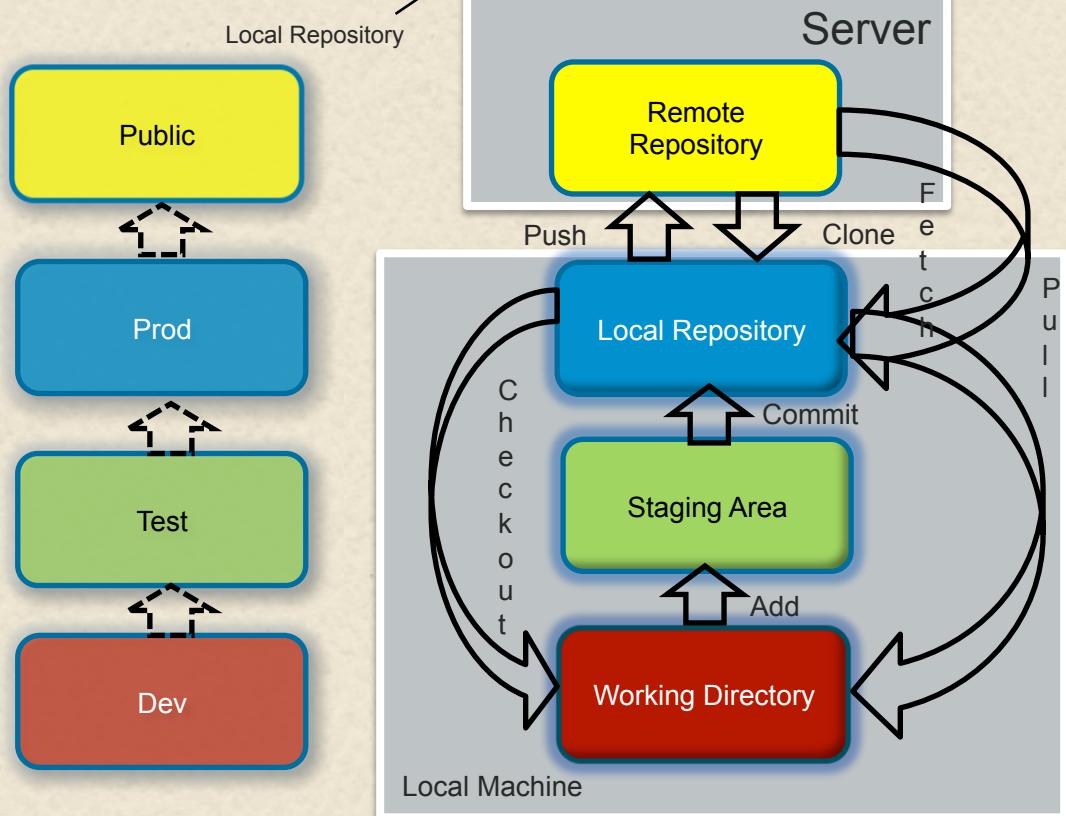
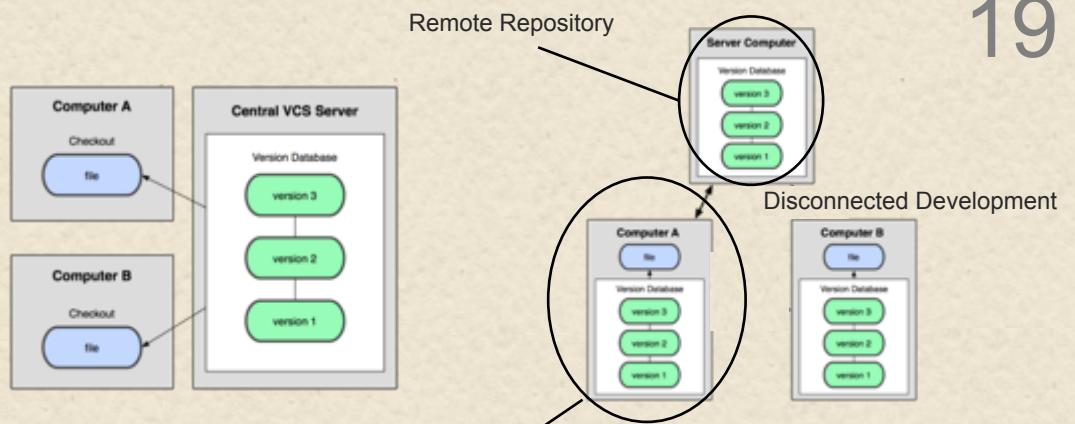
# Lab 1 – Node setup

# Git

- Distributed Version Control System
- Open source
- Available for multiple platforms
- Roots in source control efforts for Linux Kernel (2005)
- Primary developer – Linus Torvalds

## workflow

```
git init or git clone
<create or edit content>
git add .
git commit -m "<msg>"
git push origin master
```

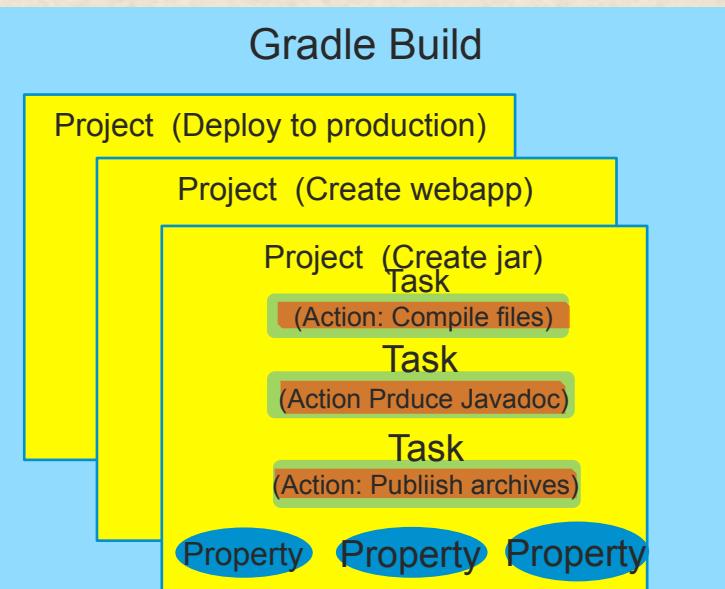


# Gradle

- General Purpose Build *Automation* System
- Middle ground between ANT and Maven
  - ANT – maximum flexibility, no conventions, no dependency management
  - Maven – strong standards, good dependency management, hard to customize, not flexible
- Offers useful “built-in” conventions, but also offers ease of extension
- Useful for developing build standards
- Rich, descriptive language (DSL – Groovy) – Java core (or Kotlin)
- Built-in support for Groovy, Java, Scala, OSGI, Web, etc.
- Multiple dependency management models
  - Transitive – can declare only top-level and interface with Maven or Ivy
  - Manual – can manage by hand
- Toolkit for domain-specific build standards
- Offers useful features – upstream builds, builds that ignore non-significant changes, etc.
- Derives a lot of functionality from plugins

# Gradle Structure

- Two basic concepts – projects and tasks
- Project
  - Every Gradle build is made up of one or more projects
  - A project is some component of software that can be built (examples: library JAR, web app, zip assembled from jars of other projects)
  - May represent something to be done, rather than something to be built. (examples: deploying app to staging area or production)
- Task
  - Represents some atomic piece of work which a build performs (examples: compiling classes, creating a jar, generating javadoc, publishing archives to a repository)
  - List of actions to be performed
  - May include configuration



```

KEYWORD   TASK NAME TYPE SPECIFICATION
task createTarget(type: DefaultTask) {
  API      doLast {
    ACTION
    destDirs.mkdirs()
  }
}

} SHORTHAND FOR DefaultTask , doLast
task createTarget <<
destDirs.mkdirs()
}

```

# Configure System vs. Global Tool Config

[Configure System \[Jenki...\]](#)

localhost:8080/configure

Jenkins configuration

**Global properties**

- Environment variables
- Tool Locations
- Help make Jenkins better by sending anonymous usage statistics and crash reports to the Jenkins project.

**Timestamper**

System clock time format: '<b>HH:mm:ss</b> '

Elapsed time format: '<b>HH:mm:ss.S</b> '

**Jenkins Location**

Jenkins URL: http://localhost:8080/

Please set a valid host name, instead of localhost

System Admin e-mail address: address not configured yet <nobody@nowher>

**SSH Server**

SSHD Port: Fixed :  Random  Disable

**Save** **Apply**

[Global Tool Configuration...](#)

localhost:8080/configureTools/

Jenkins > Global Tool Configuration

## Global Tool Configuration

**Maven Configuration**

Default settings provider: Use default maven settings

Default global settings provider: Use default maven global settings

**JDK**

JDK installations: Add JDK

List of JDK installations on this system

**Git**

Git installations:

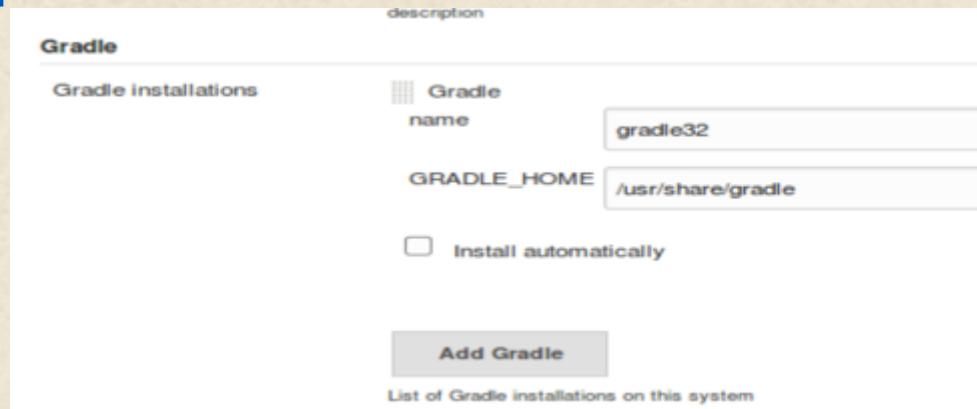
<b>Git</b>	Name: Default
Path to Git executable:	git

There's no such executable git in PATH: /usr/local/bin, /usr/bin, /bin, /usr/local/games, /usr/games.

**Save** **Apply**

# Global Tools and Pipeline

- Tools defined in global configuration
- DSL keyword ‘tool’
- In pipeline script
  - def variable for tool location
  - assign variable to “tool <toolname>” from global configuration
  - Use variable in place of location in script



```

1 -> node {
2     def gradleLoc = tool 'gradle32'
3     stage ('Build Source') {
4         git 'git@diyvb2:/opt/git/gradle-demo.git'
5         sh "${gradleLoc}/bin/gradle clean build"
6     }
7 }
```

```

[Pipeline] sh
[test-script] Running shell script
+ /usr/share/gradle/bin/gradle clean build
Starting a Gradle Daemon (subsequent builds will be faster)
:clean UP-TO-DATE
:compileJava
:processResources UP-TO-DATE
:classes
--
```

# Pipeline DSL Syntax

- Many functions (per Groovy) expect closures (code blocks enclosed in {})
- Many steps can only run inside of a node block (on an agent)
- Steps in DSL always expect mapped (named) parameters
  - Example: `git branch: 'test', url: 'http://github.com/brentlaster/gradle-greetings.git'`
  - Two parameters: "branch" and "url"
- Shorthand for `git([branch: 'test', url: 'http://github.com/brentlaster/gradle-greetings.git'])`
  - **Groovy permits not putting parentheses for parameters to functions**
  - **Named parameter syntax is shorthand for Groovy mapping syntax of [ key: value, key: value]**
- Default object is "script", so if only passing one parameter or required parameter, can omit that.
  - `bat([script: 'echo hi']) = bat 'echo hi'`

# Snippet Generator

- Facilitates generating Groovy code for typical actions
- Select the operation
- Fill in the arguments/parameters
- Push the button
- Copy and paste

The screenshot shows the Jenkins Snippet Generator interface. The left sidebar includes links for Back, Snippet Generator (selected), Step Reference, Global Variables Reference, Online Documentation, and IntelliJ IDEA GDSL. The main area has a title 'Overview' with a description of the tool's purpose. A search bar labeled 'Sample Step' contains the text 'git: Git'. Below this, configuration fields are shown: 'Repository URL' set to 'git@dlvb2.repositories:gradle-greetings.git', 'Branch' set to 'master', 'Credentials' set to '- none -', and two checked checkboxes: 'Include in polling?' and 'Include in changelog?'. At the bottom, a large button labeled 'Generate Pipeline Script' is highlighted in blue, and the resulting Groovy code 'git 'git@dlvb2.repositories:gradle-greetings.git'' is displayed in a text area.

# Shared Pipeline Libraries

- Allows you to create a shared SCM repository of pipeline script code and functions.
- Two modes:
  - General mode: Any SCM repository (“external library”)
  - Legacy mode: Git repository hosted by Jenkins (“internal library”)
- Repository Structure



- src directory is added to classpath when pipeline is executed
- vars directory is for global variables or scripts accessible from pipeline scripts; can have corresponding txt file with documentation
- resources directory can have non-groovy resources that get loaded from a “libraryResource” step in external libraries

# Defining Global Shared External Pipeline Libraries in Jenkins

27

- Defined in Manage Jenkins->Configure System

The screenshot shows the 'Global Pipeline Libraries' configuration page. It includes fields for 'Library Name' (set to 'Utilities'), 'Default version' (set to '1.5'), and checkboxes for 'Load implicitly' (unchecked), 'Allow default version to be overridden' (checked), and 'Include @Library changes in job recent changes' (checked). A note states: 'Cannot validate default version until after saving and reconfiguring.' Below this is a 'Retrieval method' section with a radio button selected for 'Modern SCM'.

- Usable by any script
- Trusted - can call any methods
  - Could put potentially unsafe methods in libraries here to encapsulate them
  - Limit push access
- Name - will be used in script to pull in library
- Version = any version identifier understood by SCM
  - example: Git: branch, tag, commit SHA1
- Modern SCM - plugin with newer api to pull specific identifier/version of library
- Legacy SCM - existing plugins
  - To pull specific identifier version, include \${library.LibraryName.version} in SCM configuration

27

# Global Shared External Pipeline Libraries

## SCM Configuration Example - Git

Modern SCM

Git

Source Code Management

Project Repository

Credentials

Ignore on push notifications

Repository browser

Additional Behaviours

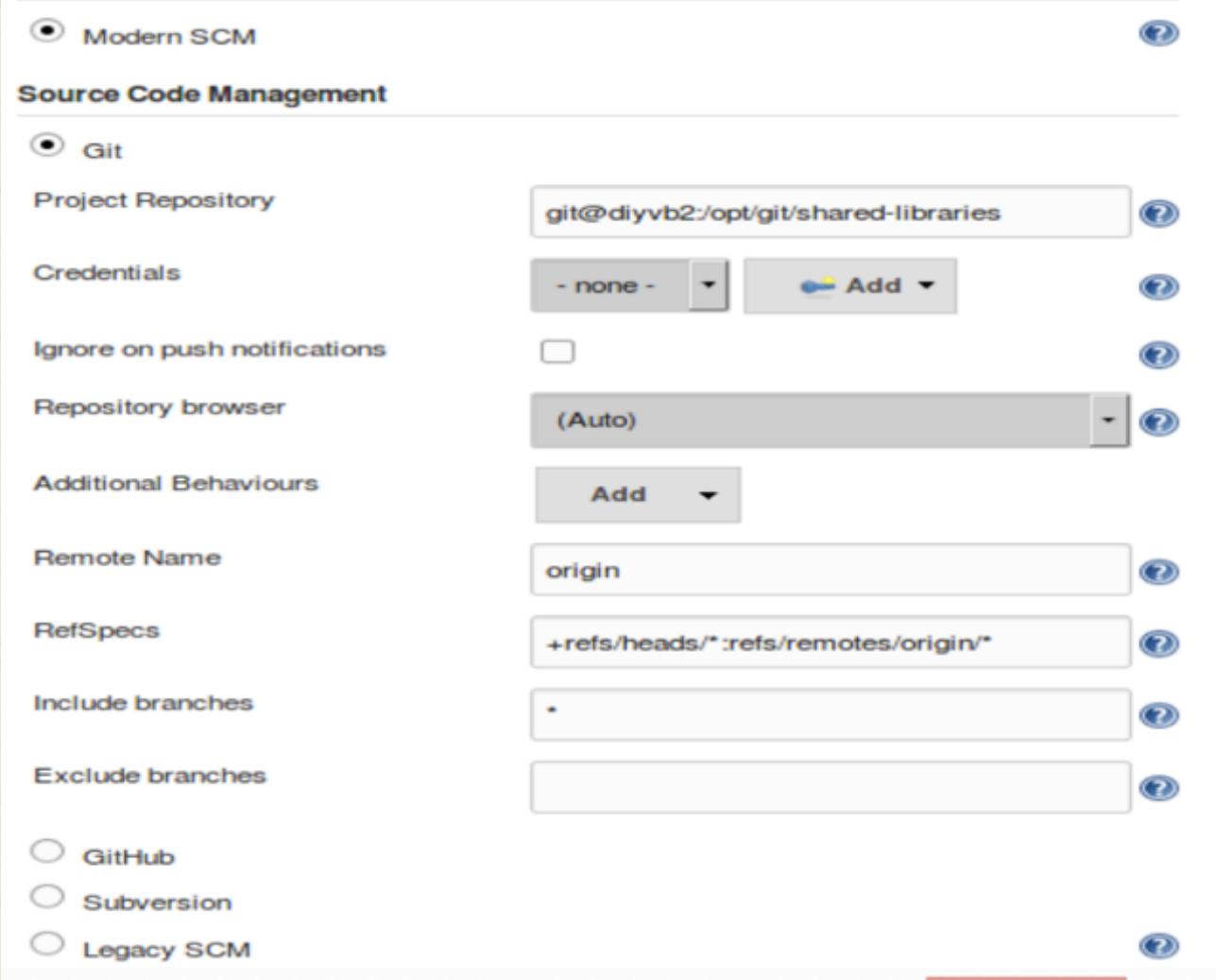
Remote Name

RefSpecs

Include branches

Exclude branches

GitHub  
 Subversion  
 Legacy SCM



# Using Libraries and Resources

- External Libraries
  - Default version: @Library('LibraryName')\_ (\_ or import)
    - » @Library('Utilities')
  - Override version: @Library('LibraryName@version')
    - » @Library('Utilities@1.4')
- Resources for External Libraries
  - use libraryResource to load from /resources directory in shared library structure
  - def input = libraryResource 'org/foo/bar/input.json'
- Third-party Libraries
  - @Grab('location:name:version')
    - » @Grab('org.apache.logging.log4j:log4j-api:2.+')
  - Does not work in Groovy Sandbox
- Direct load of code
  - def verifyCall = load("/home/diyuser/shared\_libraries/src/verify.groovy")

# Terminology - Stage

- Aggregates build steps into sections
- Stages are inside of a node block
- Stages take a name (usually corresponding to the function)

```
node ('worker_node1') {
    def gradleHome
    stage('Source') { // for display purposes
        // Get some code from our Git repository
        git 'jenkins@localhost:8091/projects/gradle-greetings.git'
    }

    stage('Results') {
        junit '**/target/surefire-reports/TEST-*.xml'
        archive 'target/*.jar'
    }
}
```

# Stage View

```

1 ~ node ('worker_node1') {
2   def gradleHome
3   stage('Source') { // for display purposes
4     // Get some code from our Git repository
5     git 'git@diyvb/repos/gradle-greetings.git'
6   }
7 }
8 ~ stage('Build') {
9   // Run the gradle build
10  gradleHome = tool 'gradle27'
11  sh "'$gradleHome/bin/gradle' clean buildAll"
12 }
13
14 }
```

Jenkins > simple-pipe >

- [Back to Dashboard](#)
- [Status](#)
- [Changes](#)
- [Build Now](#)
- [Delete Pipeline](#)
- [Configure](#)
- [Open Blue Ocean](#)
- [Full Stage View](#)
- [Pipeline Syntax](#)

**Build History** [trend](#)

find	<a href="#">X</a>
#2	Jul 27, 2017 8:57 AM
#1	Jul 27, 2017 8:50 AM
<a href="#">RSS for all</a> <a href="#">RSS for failures</a>	

## Pipeline simple-pipe



### Stage View



# Stage View and Logs

The screenshot shows the Jenkins interface for a pipeline named "simple-pipe". On the left, there's a sidebar with various options like Changes, Build Now, Delete Pipeline, Configure, Move, Full Stage View, and Pipeline Syntax. Below that is a "Build History" section with two entries: #2 (Nov 7, 2016 4:46 PM) and #1 (Nov 7, 2016 3:50 PM). At the bottom of the sidebar are RSS feed links.

The main content area is titled "Stage Logs (Build)". It shows a list of steps:

- Use a tool from a predefined Tool Installation (self time 1s)
- Shell Script (self time 49s)

The "Shell Script" step is highlighted with a red background. The log output for this step is:

```
[simple-pipe] Running shell script
+ /opt/gradle-2.7/bin/gradle clean buildAll

FAILURE: Build failed with an exception.

* What went wrong:
Task 'buildAll' not found in root project 'simple-pipe'. Some candidates are: 'build'.

* Try:
Run gradle tasks to get a list of available tasks. Run with --stacktrace option to get
trace. Run with --info or --debug option to get more log output.
```

Below the logs, it says "BUILD FAILED".

At the bottom, it shows "Total time: 42.56 secs".

On the right side, under "RECENT BUILDS", there's a list of recent builds:

- Last build (#2), 8 min 55 sec ago
- Last stable build (#1), 1 hr 5 min ago
- Last successful build (#1), 1 hr 5 min ago
- Last failed build (#2), 8 min 55 sec ago
- Last unsuccessful build (#2), 8 min 55 sec ago
- Last completed build (#2), 8 min 55 sec ago

# Replay - try out a change to a previous run<sup>33</sup>

- Can be done only after a completed run
- Done from the screen of a particular run
- Editable script window with most recent script
- Changes can be made and then “Run”
- Creates new execution as another run but changes are not saved

The screenshot shows the Jenkins interface for a pipeline named "simple-pipe". The current build is #2, and the "Replay" option is selected. The main content area is titled "Replay #2" and contains a script editor with the following Groovy code:

```
1 node ('worker_node1') {  
2     def gradleHome  
3     stage('Source') { // for display purposes  
4         // Get some code from our Git repository  
5         git 'git@diyvb2:/opt/git/gradle-demo.git'  
6     }  
7     stage('Build') {  
8         // Run the gradle build  
9         gradleHome = tool 'gradle32'  
10        sh "${gradleHome}/bin/gradle clean buildAll"  
11    }  
12}  
13}
```

Below the script editor is a link labeled "Pipeline Syntax". At the bottom of the page is a "Run" button.

# Lab 2 - Adding stages, configuring tools, and using shared libraries

# Running steps in Parallel

- DSL has a “parallel” operation
- parallel takes a map as an argument
- Value of map should be closure (pipeline steps defined in {})
- Use nodes in closures if available to get best parallelism
- If specific nodes aren’t specified, Jenkins will run on unused nodes
- Can’t use a stage step inside a parallel block

# Parallel Example 1

- Simple script to demo parallel step
- Creates map of steps
- Automatically executes on unused nodes

```
node ('worker_node1') {
    stage("Parallel Demo") {
        // Run steps in parallel

        // The map we'll store the steps in
        def stepsToRun = [:]

        for (int i = 1; i < 5; i++) {
            stepsToRun["Step${i}"] = { node {
                echo "start"
                sleep 5
                echo "done"
            }}
        }
        // Actually run the steps in parallel
        // parallel takes a map as an argument,
        parallel stepsToRun
    }
}
```

```
[Pipeline] stage
[Pipeline] { (Parallel Demo)
[Pipeline] parallel
[Pipeline] [Step1] { (Branch: Step1)
[Pipeline] [Step2] { (Branch: Step2)
[Pipeline] [Step3] { (Branch: Step3)
[Pipeline] [Step4] { (Branch: Step4)
[Pipeline] [Step1] node
[Step1] Running on master in /var/lib/jenkins/jobs/externlib-test/workspace
[Pipeline] [Step2] node
[Step2] Running on master in /var/lib/jenkins/jobs/externlib-test/workspace@2
[Pipeline] [Step3] node
[Step3] Running on worker_node2 in /home/jenkins/worker_node2/workspace/externlib-test
[Pipeline] [Step4] node
[Pipeline] [Step1] {
[Pipeline] [Step2] {
[Pipeline] [Step3] {
[Pipeline] [Step1] echo
[Step1] start
[Step1] Sleeping for 5 sec
[Pipeline] [Step1] sleep
[Pipeline] [Step2] echo
[Step2] start
[Pipeline] [Step2] sleep
[Step2] Sleeping for 5 sec
[Pipeline] [Step3] echo
[Step3] start
[Pipeline] [Step3] sleep
[Step3] Sleeping for 5 sec
[Pipeline] [Step1] echo
[Step1] done
[Pipeline] [Step1] }
[Step4] Running on master in /var/lib/jenkins/jobs/externlib-test/workspace
[Pipeline] [Step1] // node
[Pipeline] [Step1] }
[Pipeline] [Step4] {
[Pipeline] [Step4] echo
[Step4] start
[Pipeline] [Step4] sleep
[Step4] Sleeping for 5 sec
[Pipeline] [Step2] echo
[Step2] done
[Pipeline] [Step2] }
[Pipeline] [Step2] // node
[Pipeline] [Step2] }
[Pipeline] [Step3] echo
[Step3] done
[Pipeline] [Step3] }
[Pipeline] [Step3] // node
[Pipeline] [Step3] }
[Pipeline] [Step4] echo
[Step4] done
[Pipeline] [Step4] }
[Pipeline] [Step4] // node
[Pipeline] [Step4] }
[Pipeline] // parallel
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

# Parallel Example 2

- Parallel block includes explicit mapping to other nodes

```
stage ('Test') {
// execute required unit tests in parallel

    parallel (
        master: { node ('master'){
            sh '/opt/gradle-2.7/bin/gradle -D test.single=TestExample1 test'
        }},
        worker2: { node ('worker_node2'){
            sh '/opt/gradle-2.7/bin/gradle -D test.single=TestExample2 test'
        }},
    )
}
```

- Failure on other nodes due to files not being present - only on initial node

```
[master]
[master] FAILURE: Build failed with an exception.
[master]
[master] * What went wrong:
[master] Task 'test' not found in root project 'workspace@2'.
[master]
[master] * Try:
[master] Run gradle tasks to get a list of available tasks. Run with
option to get more log output.
[master]
[master] BUILD FAILED
[master]
[master] Total time: 22.374 secs
```

# stash and unstash Functions

- Allows for saving (stashing) and retrieving (unstashing) files for different stages and/or nodes in a build
- Different from Git stash
- Intended for mostly small files - use artifact repository for large files (or external workspace manager plugin)
- Format
  - stash name: "<name>" [includes: "<pattern>" excludes: "<pattern>"]
  - unstash "<name>"

# Pipeline Example 1 (with stash)

- Stash after source management step
- Stash only necessary build script and test area
- Unstash on other nodes

```
stages {  
  
    stage('Source') {  
        git branch: 'test', url: 'git@diyvb:repos/gradle-greetings'  
        stash name: 'test-sources', includes: 'build.gradle,src/test/'  
    }  
    ...  
    stage ('Test') {  
        // execute required unit tests in parallel  
  
        parallel {  
            master: { node ('master'){  
                unstash 'test-sources'  
                sh '/opt/gradle-2.7/bin/gradle -D test.single=TestExample1 test'  
            }},  
            worker2: { node ('worker_node2'){  
                unstash 'test-sources'  
                sh '/opt/gradle-2.7/bin/gradle -D test.single=TestExample2 test'  
            }},  
        }  
    }  
}
```

```
[Pipeline] stage  
[Pipeline] { (Test)  
[Pipeline] parallel  
[Pipeline] [master] { (Branch: master)  
[Pipeline] [worker2] { (Branch: worker2)  
[Pipeline] [master] node  
[master] Running on master in /var/lib/jenkins/jobs/parallel-test-stash-no-  
[Pipeline] [worker2] node  
[worker2] Running on worker_node2 in /home/jenkins/worker_node2/workspace/p  
[Pipeline] [master] {  
[Pipeline] [worker2] {  
[Pipeline] [master] unstash  
[Pipeline] [worker2] unstash  
[Pipeline] [master] sh  
[master] [workspace@2] Running shell script  
[Pipeline] [worker2] sh  
[master] + /opt/gradle-2.7/bin/gradle -D test.single=TestExample1 test  
[worker2] [parallel-test-stash-no-clean-workspace] Running shell script  
[worker2] + /opt/gradle-2.7/bin/gradle -D test.single=TestExample2 test  
[master] :compileJava UP-TO-DATE  
[master] :processResources UP-TO-DATE  
[master] :classes UP-TO-DATE  
[worker2] :compileJava UP-TO-DATE  
[worker2] :processResources UP-TO-DATE  
[worker2] :classes UP-TO-DATE  
[worker2] :compileTestJava  
[worker2] :processTestResources UP-TO-DATE  
[worker2] :testClasses  
[master] :compileTestJava  
[master] :processTestResources UP-TO-DATE  
[master] :testClasses  
[worker2] :test  
[worker2]  
[worker2] TestExample2 > example2 FAILED  
[worker2]      org.junit.ComparisonFailure at TestExample2.java:10  
[worker2]  
[worker2] 1 test completed, 1 failed  
[worker2] :test FAILED  
[worker2]  
[worker2] FAILURE: Build failed with an exception.  
[worker2]  
[worker2] * What went wrong:  
[worker2] Execution failed for task ':test'.  
[worker2] > There were failing tests. See the report at: file:///home/jenki  
[worker2]
```

# Cleaning out Workspaces

- Jenkins does not automatically clean out workspaces
- No guarantees that workspaces will persist
- Can clean up before or after
- Multiple approaches
  - `deleteDir()`
  - Install Workspace Cleanup Plugin
    - » `cleanWs()` step
    - » `Add step([$class: 'WsCleanup'])`

```
[worker2] Running on worker_node2 in /home/jenkins/worker_node2/workspace/parallel-test
[Pipeline] [master] step
[master] [WS-CLEANUP] Deleting project workspace... [WS-CLEANUP] done
[Pipeline] [worker2] {
[Pipeline] [master] unstash
[Pipeline] [master] sh
```

# Pipeline Example 2 (with stash and workspace cleanup)

```
stage('Source') {
    git branch: 'test', url: 'git@diyvb:repos/gradle-greetings'
    stash name: 'test-sources', includes: 'build.gradle,src/test/*'
}
...
stage ('Test') {
// execute required unit tests in parallel

parallel (
    master: { node ('master'){
        // always run with a new workspace
        step([$class: 'WsCleanup'])
        unstash 'test-sources'
        sh '/opt/gradle-2.7/bin/gradle -D test.single=TestExample1 test'
    }},
    worker2: { node ('worker_node2'){
        // always run with a new workspace
        step([$class: 'WsCleanup'])
        unstash 'test-sources'
        sh '/opt/gradle-2.7/bin/gradle -D test.single=TestExample2 test'
    }},
)
}
```

```
[Pipeline] { (Test)
[Pipeline] parallel
[Pipeline] [master] { (Branch: master)
[Pipeline] [worker2] { (Branch: worker2)
[Pipeline] [master] node
[master] Running on master in /var/lib/jenkins/jobs/parallel-test/workspace
[Pipeline] [worker2] node
[Pipeline] [master] {
[worker2] Running on worker_node2 in /home/jenkins/worker_node2/workspace
[Pipeline] [master] step
[master] [WS-CLEANUP] Deleting project workspace... [WS-CLEANUP] done
[Pipeline] [worker2] {
[Pipeline] [master] unstash
[Pipeline] [master] sh
[master] [workspace@2] Running shell script
[Pipeline] [worker2] step
[worker2] [WS-CLEANUP] Deleting project workspace... [WS-CLEANUP] done
[Pipeline] [worker2] unstash
[master] + /opt/gradle-2.7/bin/gradle -D test.single=TestExample1 test
[Pipeline] [worker2] sh
[worker2] [parallel-test] Running shell script
[worker2] + /opt/gradle-2.7/bin/gradle -D test.single=TestExample2 test
[worker2] :compileJava UP-TO-DATE
[worker2] :processResources UP-TO-DATE
[worker2] :classes UP-TO-DATE
[master] :compileJava UP-TO-DATE
[master] :processResources UP-TO-DATE
[master] :classes UP-TO-DATE
[worker2] :compileTestJava
[worker2] :processTestResources UP-TO-DATE
[worker2] :testClasses
[master] :compileTestJava
[master] :processTestResources UP-TO-DATE
[master] :testClasses
[worker2] :test
[worker2]
[worker2] TestExample2 > example2 FAILED
[worker2]      org.junit.ComparisonFailure at TestExample2.java:10
[worker2]
[worker2] 1 test completed, 1 failed
[worker2] :test FAILED
[worker2]
[worker2] FAILURE: Build failed with an exception.
[worker2]
[worker2] * What went wrong:
[worker2] Execution failed for task ':test'.
```

# Working with credentials - username and password

- Need to have Credentials Binding plugin installed
- Define username/password credentials in Jenkins
- Use “withCredentials” block
  - plugs in username and password to specified environment variables

T	P	Store	Domain	ID	Name
key	user	Jenkins	(global)	b9681314-2590-4175-bf6b-35875b650508	diyuser/******** (diyuser - username + pw)
key	user	Jenkins	(global)	b92cd6dd-4857-45b8-857b-d44e625ecf54	jenkins (ssh)
key	user	Jenkins	(global)	03463f1a-bddd-48cc-b1e0-575bfa9b721f	diyuser (ssh (workshop key))
key	user	Jenkins	(global)	gitcreds	brentlaster/******** (Personal Credentials for brentlaster for GitHub access)

```

1 node {
2   def urlRepo = "github.com/brentlaster/dbdeploy-example.git"
3
4   stage("Get Source") {
5     git url: 'https://'+urlRepo
6   }
7   stage("Update Source") {
8
9     sh "git config user.name diyuser"
10    sh "git config user.email diyuser@localhost"
11
12  withCredentials([
13    [class: 'UsernamePasswordMultiBinding', credentialsId: 'gitcreds',
14     usernameVariable: 'GIT_USER', passwordVariable: 'GIT_PASS']])
15
16    sh "git tag -a ${env.BUILD_TAG} -m 'demostrate push of tags'"
17    sh "git push http://${env.GIT_USER}:${env.GIT_PASS}@${urlRepo} --tags"
18  }
19}

```

# Working with credentials - ssh keys

- Need to have SSH Agent plugin installed
- Setup ssh key credentials in Jenkins
- Use sshagent block, passing in ID of ssh credentials

Credentials

T	P	Store	Domain	ID	Name
		Jenkins	(global)	b9681314-2590-4175-bf6b-35875b650508	diyuser/******** (diyuser - username + pw)
		Jenkins	(global)	b92cd6dd-4857-45b8-857b-d44e625ecf54	jenkins (ssh)
		Jenkins	(global)	03463f1a-bddd-48cc-b1e0-575bfa9b721f	diyuser (ssh (workshop key))
		Jenkins	(global)	gitcreds	brentlaster/******** (Personal Credentials for brentlaster for GitHub access)

Icon: S M L

Pipeline script

```

1  node {
2      def sshReodef = "git@diyvb/repos/shared_libraries.git"
3
4      stage ("Get Source") {
5          git url: sshReodef
6      }
7
8      stage ("Update Source") {
9
10         sh "git config user.name diyuser"
11         sh "git config user.email diyuser@localhost"
12
13         sshagent(['03463f1a-bddd-48cc-b1e0-575bfa9b721f']) {
14             sh "git tag -a ${env.BUILD_TAG} -m 'demonstrate push of tags'"
15             sh "git push ${sshReodef} --tags"
16         }
17     }
18 }
```

# Integration Testing

- Gradle java plugin has built-in conventions for tests

src/test/java	Test Java source
src/test/resources	Test resources

- We leverage this for our unit tests
- In order to separate out testing for integration tests (and functional tests) we need to define new groupings for these types of tests in the build.gradle file
- We do that using Gradle sourcesets

# Gradle Source Sets

- Source set - a collection of source files that are compiled and executed together
  - Used to group together files with a particular meaning in a project without having to create another project
  - In Java plugin, two included source sets
    - main (default for tasks below – assumed if <SourceSet> omitted)
    - test
  - Plugin adds 3 tasks for source sets
    - compile<SourceSet>Java
    - process<SourceSet>Resources
    - <SourceSet>Classes
  - Source sets also provide properties such as
    - allSource – combination of resource and Java properties
    - java.srcDirs – directories with source
- Can define your own source sets via sourceSets property of the project

```
sourceSets {  
    externApi  
}
```
  - Get 3 new tasks: externApiClasses, compileExternApiJava, and processExternApiResources
  - Custom configuration (also how you would define structure for new ones)

```
sourceSets {  
    main {  
        java {  
            srcDir 'src/java'  
        }  
        resources {  
            srcDir 'resources/java'  
        }  
    }  
}
```

# Source Sets for Integration and Functional Tests (build.gradle)

46

```
sourceSets {  
    integrationTest {  
        java {  
            compileClasspath += main.output + test.output  
            runtimeClasspath += main.output + test.output  
            srcDir file('src/integrationTest/java')  
        }  
        resources.srcDir file('src/integrationTest/resources')  
    }  
}  
  
sourceSets {  
    functionalTest {  
        java {  
            compileClasspath += main.output + test.output  
            runtimeClasspath += main.output + test.output  
            srcDir file('src/functionalTest/java')  
        }  
        resources.srcDir file('src/functionalTest/resources')  
    }  
    tasks.withType(Test) {  
        reports.html.destination = file("${reporting.baseDir}/${name}")  
        outputs.upToDateWhen { false }  
    }  
}  
  
configurations {  
    integrationTestCompile.extendsFrom mainJavaCompile  
    integrationTestRuntime.extendsFrom mainJavaRuntime  
}  
  
configurations {  
    functionalTestCompile.extendsFrom mainJavaCompile  
    functionalTestRuntime.extendsFrom mainJavaRuntime  
}  
  
task integrationTest(type:Test) {  
    environment 'MYSQL_ENV_MYSQL_DATABASE', 'registry_test'  
    testClassesDir = sourceSets.integrationTest.output.classesDir  
    classpath = sourceSets.integrationTest.runtimeClasspath  
}  
  
task jacocoIntegrationTestReport(type: JacocoReport) {  
    sourceDirectories = files(sourceSets.main.allSource.srcDirs)  
    classDirectories = files(sourceSets.main.output)  
  
    sourceDirectories += files(sourceSets.integrationTest.allSource.srcDirs)  
    classDirectories += files(sourceSets.integrationTest.allSource.srcDirs)  
    executionData integrationTest  
}
```

# Controlling which database to use

```

mysqlDS = new MysqlDataSource();

strMySQLHost = System.getenv("MYSQL_PORT_3306_TCP_ADDR");
if(strMySQLHost == null || strMySQLHost.isEmpty()) {
    strMySQLHost = "localhost";
}
strMySQLPort = System.getenv("MYSQL_PORT_3306_TCP_PORT");
if(strMySQLPort == null || strMySQLPort.isEmpty()) {
    strMySQLPort = "3306";
}
strMySQLDatabase = System.getenv("MYSQL_ENV_MYSQL_DATABASE");
if(strMySQLDatabase == null || strMySQLDatabase.isEmpty()) {
    strMySQLDatabase = "registry";
}
mysqlDS.setURL("jdbc:mysql://"+strMySQLHost+":"+strMySQLPort+"/"+strMySQLDatabase
);

```

`docker-compose-roar-containers.yml`

```

roar-web-container:
  image: roar-web-image
  ports:
    - "8089:8080"
  links:
    - "roar-db-container:mysql"
roar-db-container:
  image: roar-db-image
  ports:
    - "3308:3306"
  environment:
    MYSQL_USER: admin
    MYSQL_PASSWORD: admin
    MYSQL_DATABASE: registry
    MYSQL_ROOT_PASSWORD: root+1

```

`build.gradle`

```

// set an environment variable that we use to point to the database we want
task integrationTest(type:Test){
    environment 'MYSQL_ENV_MYSQL_DATABASE', 'registry_test'
    testClassesDir = sourceSets.integrationTest.output.classesDir
    classpath = sourceSets.integrationTest.runtimeClasspath
}

```

# Example Integration Test Output

Test results - Test Summary - Mozilla Firefox

localhost:8080/job/roar-analysis/ws/dataaccess/build/reports/integrationTest/index.html

## Test Summary

6 tests	0 failures	0 ignored	2.476s duration
---------	------------	-----------	-----------------

**100% successful**

[Packages](#) [Classes](#)

### Packages

Package	Tests	Failures	Ignored	Duration	Success rate
<a href="#">com.demo.dao</a>	6	0	0	2.476s	100%

### Classes

Class	Tests	Failures	Ignored	Duration	Success rate
<a href="#">com.demo.dao.MyDataSourceTest</a>	1	0	0	0.759s	100%
<a href="#">com.demo.dao.SchemaRegistryTest</a>	5	0	0	1.717s	100%

Workspace of roar-analysis on jenkins\_slave1

[dataaccess / build / reports / integrationTest /](#)

- [classes](#)
- [css](#)
- [ls](#)
- [packages](#)
- [index.html](#) 2.49 KB [view](#)
- [\(all files in zip\)](#)

# Lab 3 – Adding in testing

# Technology - SonarQube ([sonarsource.org](http://sonarsource.org))<sup>50</sup>

- Open platform for helping manage code quality in 7 areas:
  - Architecture and Design
  - Comments
  - Coding rules
  - Potential bugs
  - Duplications
  - Unit tests
  - Complexity
- Allows configuration of rules, alerts, thresholds, exclusions, etc.
- Extensible via plugins
- Example site: <http://nemo.sonarqube.org>

# Sonar - drilling in

<http://localhost:9000>

The screenshot shows the SonarQube Main Dashboard for the 'ROAR :: Pipeline Demo' project. The dashboard includes the following sections:

- Lines Of Code:** 47 Java files, 1 directory, 67 lines.
- Duplications:** 0.0% (0 lines, 0 blocks, 0 files).
- Complexity:** 6 cyclomatic complexity units across 2.0 functions, 6.0 classes, and 6.0 files. A bar chart shows complexity distribution: 1 function has 2 cycles, 2 functions have 1 cycle, and 4 functions have 0 cycles.
- SOALE Rating:** A (Green)
- Technical Debt Ratio:** 2.1%
- Debt:** 30min
- Issues:** 6 (Breakdown: 0 Blocker, 2 Critical, 0 Major, 4 Minor, 0 Info).
- File Tangle Index:** 0.0%
- Suspect File Dependencies:** 0

At the bottom, there is a footer with links to SonarQube technology details and a copyright notice for SonarSource SA.

# Sonar - looking at issues

The screenshot shows the SonarQube web interface for a project named "ROAR :: Pipeline Demo". The user is viewing the "Issues" tab under the "Components" section. A search bar at the top right contains the text "Search". The date "May 3 2016 12:25 PM" is also visible.

**Severity Summary:**

Severity	Count
Blocker	0
Critical	2
Major	0
Minor	4
Info	0

**Unresolved Issues:**

Type	Count	Status	Assignee	Priority	Last Updated	File
Unresolved	6	Fixed		0	2 months ago	V1_status.java
False Positive	1	Won't fix		0	2 months ago	L13
Removed	0					

**Resolved Issues:**

- Replace all tab characters in this file by sequences of white-spaces. (Minor, Open, Not assigned, Not planned, 2min debt) - 2 months ago, L13, convention
- Rename this class name to match the regular expression "[A-Z][a-zA-Z0-9]\*\$". (Minor, Open, Not assigned, Not planned, 5min debt) - 2 months ago, L15, convention
- Rename this constant name to match the regular expression "[A-Z][A-Z0-9]\*([A-Z0-9]+)\*\$". (Minor, Open, Not assigned, Not planned, 2min debt) - 2 months ago, L16, convention
- Either log or rethrow this exception. (Critical, Open, Not assigned, Not planned, 10min debt) - 2 months ago, L56, error-handling
- Use a logger to log this exception. (Critical, Open, Not assigned, Not planned, 10min debt) - 2 months ago, L57, error-handling
- All most one statement is allowed per line, but 2 statements were found on this line. (Minor, Open, Not assigned, Not planned, 1min debt) - 2 months ago, L60, convention

# Sonar - responses

ROAR :: Pipeline Demo api/src/main/java/com/demo/pipeline/status/V1\_status.java

Replace all tab characters in this file by sequences of white-spaces. Minor  
Minor Open Not assigned Not planned 2min debt Comment

Rename this class name to match the regular expression "[A-Z][a-zA-Z0-9]\*\$". Minor  
Minor Open Not assigned Not planned 5min debt Comment

Rename this constant name to match the regular expression "[A-Z][A-Z0-9]\*[A-Z0-9]\*\$". Minor  
Minor Open Not assigned Not planned 2min debt Comment

Either log or rethrow this exception. Critical  
Critical Open Not assigned Not planned 10min debt Comment

This exception. Minor  
Minor Open Not assigned Not planned 10min debt Comment

At most one statement is allowed per line, but 2 statements were found on this line. Minor  
Minor Open Not assigned Not planned 1min debt Comment

Blocker Critical Major Minor Info

ROAR :: Pipeline Demo api/src/main/java/com/demo/pipeline/status/V1\_status.java

Replace all tab characters in this file by sequences of white-spaces. Minor  
Minor Open Not assigned Not planned 2min debt Comment

Rename this class name to match the regular expression "[A-Z][a-zA-Z0-9]\*\$". Minor  
Minor Open Not assigned Not planned 5min debt Comment

Rename this constant name to match the regular expression "[A-Z][A-Z0-9]\*[A-Z0-9]\*\$". Minor  
Minor Open Not assigned Not planned 2min debt Comment

Either log or rethrow this exception. Critical  
Critical Open Not assigned Not planned 10min debt Comment

Use a logger to log this exception. Critical  
Critical Open Not assigned Not planned 10min debt Comment

At most one statement is allowed per line, but 2 statements were found on this line. Minor  
Minor Open Not assigned Not planned 1min debt Comment

Markdown Help "Build" "Code" "Comment" Cancel

At most one statement is allowed per line, but 2 statements were found on this line. Minor  
Minor Open Not assigned Not planned 1min debt Comment

ROAR :: Pipeline Demo api/src/main/java/com/demo/pipeline/status/V1\_status.java

Replace all tab characters in this file by sequences of white-spaces. Minor  
Minor Open Not assigned Not planned 2min debt Comment

Rename this class name to match the regular expression "[A-Z][a-zA-Z0-9]\*\$". Minor  
Minor Open Not assigned Not planned 5min debt Comment

Rename this constant name to match the regular expression "[A-Z][A-Z0-9]\*[A-Z0-9]\*\$". Minor  
Minor Open Not assigned Not planned 2min debt Comment

Either log or rethrow this exception. Critical  
Critical Open Not assigned Not planned 10min debt Comment

Use a logger to log this exception. Critical  
Critical Open Not assigned Not planned 10min debt Comment

At most one statement is allowed per line, but 2 statements were found on this line. Minor  
Minor Open Not assigned Not planned 1min debt Comment

ROAR :: Pipeline Demo api/src/main/java/com/demo/pipeline/status/V1\_status.java

Replace all tab characters in this file by sequences of white-spaces. 2 months ago Minor  
Minor Open Not assigned Not planned 2min debt Comment

Rename this class name to match the regular expression "[A-Z][a-zA-Z0-9]\*\$". 2 months ago L13 Minor  
Minor Open Not assigned Not planned 5min debt Comment

Rename this constant name to match the regular expression "[A-Z][A-Z0-9]\*[A-Z0-9]\*\$". 2 months ago L13 Minor  
Minor Open Not assigned Not planned 2min debt Comment

Either log or rethrow this exception. 2 months ago L56 Critical  
Critical Open Not assigned Not planned 10min debt Comment

Use a logger to log this exception. 2 months ago L56 Critical  
Critical Open Not assigned Not planned 10min debt Comment

At most one statement is allowed per line, but 2 statements were found on this line. 2 months ago L56 Minor  
Minor Open Not assigned Not planned 1min debt Comment

Search... error-handling bad-practice brain-overload bug code clumsy

ROAR :: Pipeline Demo api/src/main/java/com/demo/pipeline/status/V1\_status.java

Replace all tab characters in this file by sequences of white-spaces. Minor  
Minor Open Not assigned Not planned 2min debt Comment

Rename this class name to match the regular expression "[A-Z][a-zA-Z0-9]\*\$". Minor  
Minor Open Not assigned Not planned 5min debt Comment

Rename this constant name to match the regular expression "[A-Z][A-Z0-9]\*[A-Z0-9]\*\$". Minor  
Minor Open Not assigned Not planned 2min debt Comment

Either log or rethrow this exception. Critical  
Critical Open Not assigned Not planned 10min debt Comment

Use a logger to log this exception. Critical  
Critical Open Not assigned Not planned 10min debt Comment

At most one statement is allowed per line, but 2 statements were found on this line. Minor  
Minor Open Not assigned Not planned 1min debt Comment

Return to List ROAR :: Pipeline Demo api/src/main/java/com/demo/ 1 / 1 Reload New Search Bulk Change

```
query.close();  
  
returnString = "<p>Database Status</p>" +  
    "<p>Database Date/Time return: " + myString + "</p>";  
}  
catch (Exception e) {  
  
    Either log or rethrow this exception. 2 months ago L56 Critical  
    Critical Open Not assigned Not planned 10min debt Comment  
    error-handling  
  
    Use a logger to log this exception. 2 months ago L56 Critical  
    Critical Open Not assigned Not planned 10min debt Comment  
  
    At most one statement is allowed per line, but 2 statements were found on this line. 2 months ago L56 Minor  
    Minor Open Not assigned Not planned 1min debt Comment  
  
    e.printStackTrace();  
}  
finally {  
    if (conn != null) conn.close();  
}
```

# Sonar - Quality Profiles

The screenshot shows the SonarQube interface for managing Quality Profiles. The top navigation bar includes tabs for Dashboards, Issues, Measures, Rules, Quality Profiles (which is selected), Quality Gates, Settings, and More. A search bar and various navigation icons are also present. On the left, a sidebar lists 'Quality Profiles' and 'Java Profiles'. The main content area displays a table for 'Sonar way' with the following details:

NAME	RULES	PROJECTS	DEFAULT	OPERATIONS
Sonar way	192		✓	Back up Rename Copy

Buttons for 'Compare Profiles', 'Restore Profile', 'Restore Built-in Profiles', and 'Create' are located above the table. The URL in the browser is `localhost:9000/sonar/profiles`.

# Sonar - rules

The image displays two side-by-side screenshots of the SonarQube web application, specifically the 'Coding Rules' section for Java.

**Left Screenshot:** Shows a list of Java coding rules. One rule is highlighted: "compareTo" should not return "Integer.MIN\_VALUE".

Rule Description	Language	Severity	Count
"equals()" should not be used to test the values of "Atomic" classes	Java	bug	271
@Override annotation should be used on any method overriding (since Java 5) or implementing (since Java 6) another one	Java	bad-practice	33
"BigDecimal(double)" should not be used	Java	bug, cert	31
"CHECKSTYLE-OFF" suppression comments should not be used	Java	bad-practice	27
"Cloneables" should implement "clone"	Java	bug	21
"compareTo" results should not be checked for specific values	Java	bug	17
"compareTo" should not return "Integer.MIN_VALUE"	Java	bug	16
"ConcurrentLinkedQueue.size()" should not be used	Java	performance	16
"deleteOnExit" should not be used	Java	performance	14
"Double.longBitsToDouble" should not be used for "int"	Java	bug	14
"equals" methods should be symmetric and work for subclasses	Java	bug	13
"equals(Object obj)" and "hashCode()" should be overridden in pairs	Java	bug, cert, cwe	11
"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method	Java	bug	10
"final" classes should not have "protected" members	Java	confusing	10
"finalize" should not set fields to "null"	Java	cluney, performance	9
"FIXME" tags should be handled	Java		8
"for" loop incrementers should modify the variable being tested in the loop's stop condition	Java	bug	7
"hashCode" and "toString" should not be called on array instances	Java	bug	7

**Right Screenshot:** Provides a detailed view of the rule "compareTo" should not return "Integer.MIN\_VALUE". It includes the rule ID (SQ2187), severity (Critical, bug), and a note about its reliability. It also shows a 'Noncompliant Code Example' and a 'Compliant Solution'.

**Noncompliant Code Example:**

```
public int compareTo(MyClass) {
    if (condition) {
        return Integer.MIN_VALUE; // Noncompliant
    }
}
```

**Compliant Solution:**

```
public int compareTo(MyClass) {
    if (condition) {
        return -1;
    }
}
```

**Quality Profiles:** Shows the rule is assigned to the 'Sonar way' profile at the Critical level.

# Sonar - quality gates

The screenshot shows the SonarQube interface for managing quality gates. The top navigation bar includes links for Dashboards, Issues, Measures, Rules, Quality Profiles, Quality Gates (which is the active tab), Settings, and More. A search bar and user account information are also present.

The main content area is titled "SonarQube way". It displays a section titled "CONDITIONS" with a note: "Only project measures are checked against thresholds. Sub-projects, directories and files are ignored." Below this, a legend explains project health icons:

- Green checkmark: at least one threshold is defined, no threshold is reached.
- Yellow warning icon: at least one warning threshold is reached, no error threshold is reached.
- Red error icon: at least one error threshold is reached.

Under the "Add Condition" dropdown, there are eight rows of configuration for various metrics:

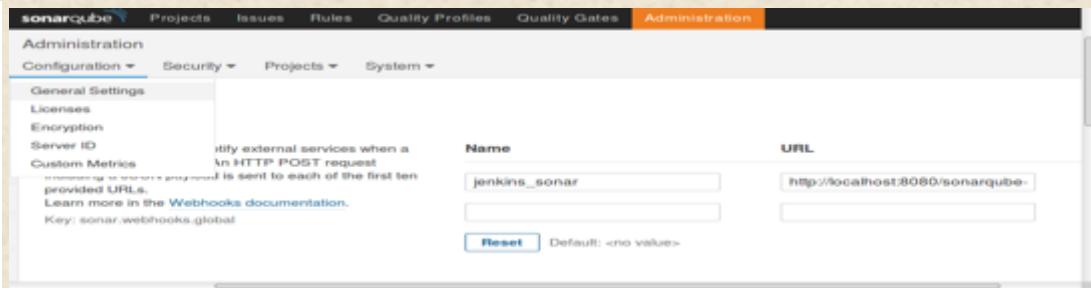
Metric	Value	Operator	Threshold	Action
Blocker issues	Value	is greater than	0	Update Delete
Critical issues	Δ since previous version	is greater than	0	Update Delete
Coverage on new code	Δ since previous version	is less than	80	Update Delete
Open issues	Value	is greater than	0	Update Delete
Reopened issues	Value	is greater than	0	Update Delete
Skipped unit tests	Value	is greater than	0	Update Delete
Unit test errors	Value	is greater than	0	Update Delete
Unit test failures	Value	is greater than	0	Update Delete

Below the conditions, there is a "PROJECTS" section with three buttons: "With", "Without", and "All". A checkbox for "ROAR : Pipeline Demo" is selected. A search bar is also present in this section.

# SonarQube Integration with Pipeline

- Globally install and configure server and scanner (optional scanners for Gradle, Maven)
- In pipeline, invoke scanner
  - Use `withSonarQubeEnv` DSL method to automatically attach SonarQube task id to pipeline context (injects environment variables)
- (Optional) Define webhook in Sonar to notify Jenkins when analysis is done
- (Optional) Define wait in pipeline to wait for webhook and quality gate
  - Uses `waitForQualityGate` DSL method
  - Pauses Pipeline execution and wait for previously submitted SonarQube analysis to complete and returns quality gate status.

```
stage('SonarQube Analysis') {
    // requires Scanner 2.8+
    def scannerLoc = tool 'SonarQube Scanner 2';
    withSonarQubeEnv('SonarQube Server') {
        sh "${scannerLoc}/bin/sonar-scanner"
    }
}
```



```
stage("Quality Gate") {
    timeout(time: 30, unit: 'MINUTES') { // Set maximum time we'll wait for this
        def qgate = waitForQualityGate() // Reuse taskId previously collected by withSonarQubeEnv
        if (qgate.status != 'OK') {
            error "Pipeline aborted due to not passing quality gate criteria: ${qgate.status}"
        }
    }
}
```

# Pipeline Flow Control

- input - stop and wait for user response
  - code: input 'Continue to next stage?'
  - default options: proceed or abort
  - optionally can take parameters to provide additional information
  - consider wrapping with timeout to ensure continuation
- retry - if an exception happens in code block, retry
  - code: retry(5) { // processing }
  - if retry limit is reached and exception occurs, aborts processing (if not handled)
- sleep - pauses pipeline until provided time expires
  - code: sleep 30 (if seconds)
  - code: sleep time: 1, unit: 'MINUTES' (if other unit like minutes)
- timeout - execute code in block with timeout
  - code: timeout(time: 600, unit: 'NANOSECONDS') { // processing }
  - if timeout is hit, throws exception - aborts processing (if not handled)
- waituntil - wait until processing in block returns true
  - code: waituntil { // processing }
  - if processing executes and returns false, wait a bit and try again
  - exceptions in processing exit immediately and throw error
  - good to wrap with a timeout

# Jacoco (Java Code Coverage)

- Derived from instrumenting Java class files
- Instruction coverage - info about how much code has been executed
- Branch coverage - if/switch statements; counts total number of branches in a method; figures out number of executed or missed branches
- Cyclomatic Complexity - minimum number of paths that can generate all possible paths through a method (McCabe 1996). Can suggest number of unit tests to completely cover a piece of code.
- Lines (requires debug info, default in gradle), methods, classes



- For source lines with executable code
  - Fully covered code - green
  - Partially covered code - yellow
  - Haven't been executed - red
- Diamonds on left are for decision lines
  - All branches executed- green
  - Part of branches executed - yellow
  - No branches execute - red

```
//loop through all the columns and place them into the JSON Object
for (int i=1; i<numColumns+1; i++) {

    String column_name = rsmd.getColumnName(i);

    if(rsmd.getColumnType(i)==java.sql.Types.ARRAY){
        obj.put(column_name, rs.getArray(column_name));
    }
    else if(rsmd.getColumnType(i)==java.sql.Types.BIGINT){
        obj.put(column_name, rs.getInt(column_name));
    }
}
```

```
step([$class: 'JacocoPublisher',
      execPattern:'**/**.exec',
      classPattern: '**/classes/main/com/demo/util,**/classes/main/com/demo/dao',
      sourcePattern: '**/src/main/java/com/demo/util,**/src/main/java/com/demo/dao',
      exclusionPattern: '**/*Test*.class'])
```

# Lab 4 - Analysis

# Semantic Versioning

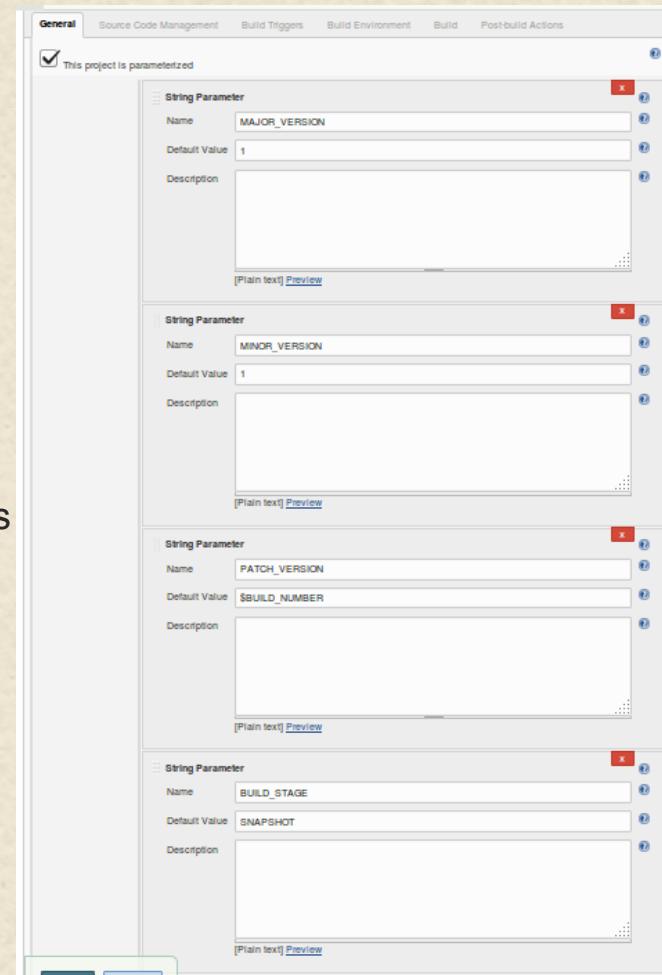
- Format - **Major.Minor.Patch-<label>**
- Increment Major version for incompatible API changes.
- Increment Minor version for adding functionality in a backwards-compatible manner.
- Increment Patch version for backwards-compatible bug fixes.
- Labels for pre-release or build metadata can be added as an extension to the version number (in “<label>”).

**Project ref-assemble**

This build requires parameters:

MAJOR_VERSION	1
MINOR_VERSION	1
PATCH_VERSION	\$BUILD_NUMBER
BUILD_STAGE	SNAPSHOT
PERSISTENT_WORKSPACE	/home/jenkins/ref_worker/workspace/ref-compile

**Build**



```
// if we have the semantic versions values set then use those

if (hasProperty('MAJOR_VERSION') && hasProperty('MINOR_VERSION') && hasProperty('PATCH_VERSION') && hasProperty('BUILD_STAGE'))
    {setVersion = MAJOR_VERSION + '.' + MINOR_VERSION + '.' + PATCH_VERSION + '-' + BUILD_STAGE}

// otherwise if have the SOURCE_BUILD_NUMBER set (as would be in Jenkins) then use that (and assume that we are running in Jenkins)

else if (System.env.SOURCE_BUILD_NUMBER != null)
    {setVersion = '0.0.' + System.env.SOURCE_BUILD_NUMBER + '-SNAPSHOT'
     runningInJenkins = 1
    }
```

# Parameters and Pipelines

- Ways to access parameters in a pipeline
  - env.<Parameter Name> - returns the string value of the environment variable
  - params.<Parameter Name> - returns the “strongly typed” parameter
  - \${<Parameter Name>} – returns the interpolated value

The screenshot shows the 'General' configuration page of a Jenkins project. A checkbox labeled 'This project is parameterized' is checked. Below it, there are two 'String Parameter' entries:

- String Parameter**
  - Name: MAJOR\_VERSION
  - Default Value: 1
  - Description: (empty)
- String Parameter**
  - Name: MINOR\_VERSION
  - Default Value: 1
  - Description: (empty)

Each parameter entry has 'Plain text' and 'Preview' buttons at the bottom.

```
def workspace = env.WORKSPACE
```

```
"${params.MAJOR_VERSION}",  
"${params.MINOR_VERSION}",  
"${params.PATCH_VERSION}",  
"${params.BUILD_STAGE}"
```

# Workflow Remote Loader Plugin

- Allows loading Groovy code from Git and SVN
- Provides a way to maintain logic in remote files in SCMs and load them on-demand
- Adds global fileLoader DSL variable with associated methods
  - fromGit (String libPath, String repository, String branch, String credentialsId, String labelExpression) – loads a single Groovy file from the Git repository
    - » libPath – relative path to file. “.groovy” extension implied
    - » repository – supports all forms supported by Git Plugin
    - » branch – can also be labels; default is “master”
    - » credentialsId – optional: default value: null (no authorization)
    - » labelExpression – optional: node label to specify node for checkout; default is empty string (runs on any node)
  - Function in Git should always have “return this;”(supplies scope to pipeline script to make calls to function)

# Loading code directly

- Can load code directly via load operation
- Usage:

```
def myProc = load("/path/to/proc/code/proc.groovy")  
myProc("argument")
```

- Assumes proc.groovy has form:

```
def call(String <argument>) { <processing> }
```

# Assemble Job - assemble task

```
project(':web') {  
    <...>  
    apply plugin:'war'  
    <...>  
    // create a simple info file for packaging with our war  
    task createInfoFile << {  
        def InfoFile = new File("web/app-info.txt")  
        Properties props = new Properties()  
        props.setProperty('version',version)  
        props.setProperty('disclaimer',"Covered under the Use At Your Own Risk guarantee. For workshop demos only. All others beware!")  
        props.store(InfoFile.newWriter(),null)  
    }  
    apply plugin: 'maven'  
    task createPom << {  
        pom {  
            project {  
                groupId 'com.demo.pipeline'  
                artifactId project.name  
                version "$version"  
            }  
        }.writeTo("pom.xml")  
    }  
    war {  
        dependsOn createInfoFile, createPom  
        basePath = "web"  
        from('..') {  
            include 'app-info.txt'  
            into('WEB-INF/classes')  
        }  
    }  
}
```

# Lab 5 - Assemble

# Publishing

- Plugins usually define what needs to be published
- Need to tell Gradle where to publish artifacts
- Done by attaching a repository to uploadArchives task

```
uploadArchives {  
    repositories {  
        ivy {  
            credentials {  
                username = "name"  
                password = "pass"  
            }  
            url = 'http://intranet/IUOCustom'  
        }  
    }  
}
```

- Note: Can define an artifact using an archive task

```
artifacts {  
    archives someFile (or jar)  
}
```

# What is Artifactory?

- Binary repository manager
  - We have source control for our source
  - Artifactory provides version control for binary artifacts (jars, wars, etc.)
- Why use it?
  - If you rebuild from source each time, that can introduce change and failure for consumers.
  - By having a versioned copy of a (tested) artifact, everyone knows what they are getting.
  - Having multiple versions stored and versioned clearly allows different consumers to use different versions (i.e. current, last release, etc.)
  - Integrates with CI servers (such as Jenkins) so that if build is clean, automatically gets published with metadata about build
  - Allows virtual repositories which can aggregate multiple well-known or internal repositories

# Artifactory - Artifacts

The screenshot shows the JFrog Artifactory web interface. The top navigation bar includes tabs for 'roar-publish-artifact #3...', 'Artifactory', and 'Apache Tomcat/8.0.33'. The main title is 'Artifactory - Mozilla Firefox'. The left sidebar has links for Home, Artifacts, Builds, and Admin. The Artifacts section is selected. The main content area is titled 'Artifact Repository Browser' and shows the 'libs-snapshot-local' repository. The tree view on the left shows a hierarchy under 'libs-snapshot-local': ext-snapshot-local, libs-release-local, and libs-snapshot-local, which is expanded to show com/demo/pipeline, api, dataaccess, roarv2, util, and web. The web folder contains several artifact versions: 0.0.1-SNAPSHOT, 1.0.0-SNAPSHOT, 1.1.27-SNAPSHOT, 1.1.28-SNAPSHOT, and 1.1.29-SNAPSHOT. The right panel displays the 'libs-snapshot-local' repository details. The 'General' tab is selected, showing the following information:

Name:	libs-snapshot-local
Package Type:	Maven
Repository Path:	libs-snapshot-local/ ⓘ
Repository Layout:	maven-2-default
Description:	Local repository for in-house snapshots
Artifact Count:	34
Created:	04-10-15 00:22:51 -04:00 ⓘ

The 'Properties' tab is also visible. Below the general info, there is a section for 'Virtual Repository Associations' with a link to 'libs-snapshot'.

# Artifactory - Builds

The screenshot shows a web browser window with three tabs: 'roar-publish-artifact #3...' (active), 'Artifactory', and 'Apache Tomcat/8.0.33'. The main content is the 'Build Browser' for the 'roar-publish-artifact' build. The left sidebar has links for Home, Artifacts, Builds (selected), and Admin. The top navigation bar shows the URL 'localhost:8082/artifactory/webapp/#/builds/roar-publish-artifact' and includes a search bar, user info ('Welcome, Diyuser ( Log Out )'), and help links.

**Build Browser**

History for Build 'roar-publish-artifact'

32 Builds

Build ID	CI Server	Status	Build Time
36	http://localhost:8080/job/roar-publish-artifact/36/		03-05-16 12:26:15 -04:00
35	http://localhost:8080/job/roar-publish-artifact/35/		03-05-16 12:22:50 -04:00
34	http://localhost:8080/job/roar-publish-artifact/34/		03-05-16 12:09:15 -04:00
33	http://localhost:8080/job/roar-publish-artifact/33/		03-05-16 12:03:15 -04:00
32	http://localhost:8080/job/roar-publish-artifact/32/		03-05-16 11:56:45 -04:00
31	http://localhost:8080/job/roar-publish-artifact/31/		02-05-16 16:26:20 -04:00
30	http://localhost:8080/job/roar-publish-artifact/30/		02-05-16 16:20:14 -04:00
29	http://localhost:8080/job/roar-publish-artifact/29/		02-05-16 16:15:34 -04:00
28	http://localhost:8080/job/roar-publish-artifact/28/		02-05-16 16:04:54 -04:00
27	http://localhost:8080/job/roar-publish-artifact/27/		02-05-16 15:47:57 -04:00
26	http://localhost:8080/job/roar-publish-artifact/26/		02-05-16 15:08:58 -04:00

70

# Artifactory - Editing/Defining Repositories

The screenshot shows the 'Edit jcenter Repository' page in the JFrog Artifactory web interface. The left sidebar is dark grey with white text, showing navigation links like Admin, Back to Main, Repositories, Local, Remote, Virtual, Layouts, Configuration, Security, Services, Import & Export, and Advanced. The main area has a light grey background with a green header bar at the top. The title 'Edit jcenter Repository' is in the header. Below it is a tab navigation bar with 'Basic' (selected), 'Advanced', and 'Replications'. The 'Basic' tab contains fields for 'Package Type \*' (set to 'Maven'), 'Repository Key \*' (set to 'jcenter'), and 'URL \*' (set to 'http://jcenter.bintray.com'). A 'Test' button is next to the URL field. Below these are sections for 'General' settings (Repository Layout: 'maven-2-default', Remote Layout Mapping: 'Select Remote Layout') and 'Maven Settings' (Checksum Policy: 'Generate if absent', Max Unique Snapshots: '0'). There are also sections for 'Include Patterns' ('\*\*/\*') and 'Exclude Patterns' (empty). At the bottom right are buttons for 'Cancel', 'Back', 'Next >', and a large green 'Save & Finish' button.

# Artifactory - Virtual Repository

The screenshot shows the JFrog Artifactory web interface for managing a virtual repository. The left sidebar navigation includes Admin, Back to Main, Repositories, Local, Remote, Virtual, Layouts, Configuration, Security, Services, Import & Export, and Advanced. The main content area is titled "Edit libs-release Repository". It has tabs for Basic and Advanced, with Basic selected. Under Basic, the Package Type is set to Maven. The General section includes fields for Repository Key (set to "libs-release"), Public Description, Internal Description, Include Patterns (containing "\*\*/\*"), and Exclude Patterns. To the right, the "Repositories" section shows a list of available repositories: ext-snapshot-local, libs-snapshot-local, plugins-release-local, plugins-snapshot-local, jcenter, mavenCentral, mavenLoc, and libs-snapshot. A "Selected Repositories" panel on the right lists three repositories: libs-release-local, ext-release-local, and remote-repos. At the bottom are buttons for Cancel, Back, Next, and Save & Finish.

# Artifactory Integration with Jenkins (and Gradle)<sup>73</sup>

- Artifactory server and plugin installed and configured – provides Artifactory object
- Create instance pointing to installed Artifactory - def server = Artifactory.server "<name>"
- Create new Artifactory Gradle object and point to installed Gradle
  - def artifactoryGradle = Artifactory.newGradleBuild()
  - artifactoryGradle.tool = "<gradle tool name in Jenkins>"
- Set publish/deploy repositories
  - artifactoryGradle.deployer repo:'libs-snapshot-local', server: server
  - artifactoryGradle.resolver repo:'remote-repos', server: server
- Tell Jenkins whether or not Gradle is already including Artifactory plugin
  - artifactoryGradle.usesPlugin = false
- Set options to capture build info (if desired)
  - def buildInfo = Artifactory.newBuildInfo(), buildInfo.env.capture = true
- Set deploy/publish options
  - artifactoryGradle.deployer.deployMavenDescriptors = true
  - artifactoryGradle.deployer.artifactDeploymentPatterns.addExclude("\*.jar")
- Invoke artifactoryGradle object as you would for Gradle
  - artifactoryGradle.run rootDir: "/", buildFile: 'build.gradle', tasks: ...
- Publish the build info (if desired) - server.publishBuildInfo buildInfo

# Build info in Artifactory

Deploying build descriptor to: <http://localhost:8082/artifactory/api/build>  
Build successfully deployed. Browse it in Artifactory under <http://localhost:8082/artifactory/webapp/builds/ref-publish-artifact/41>

The screenshot shows the JFrog Artifactory web interface. The URL in the browser bar is <http://localhost:8082/artifactory/webapp/#/builds/ref-publish-artifact/41/1463147123186/general/>. The page title is "JFrog Artifactory". The left sidebar has links for Home, Artifacts, Builds, and Admin. The main content area is titled "Build Browser" and shows "Build #41". Below the title, there are tabs: General Build Info, Published Modules (which is selected), Environment, Issues, and Licenses. Under "Published Modules", it says "4 Modules". A table lists the modules:

Module ID	Number Of Artifacts	Number Of Depend...
com.demo.pipeline:api:1.1.38-SNAPSHOT	1	54
com.demo.pipeline:dataaccess:1.1.38-SNAPSHOT	1	53
com.demo.pipeline:util:1.1.38-SNAPSHOT	1	52
com.demo.pipeline:web:1.1.38-SNAPSHOT	2	55

At the bottom left, there is a logo for "JFrog Artifactory OSS 4.1.3 rev 40020 © Copyright 2016 JFrog Ltd". The footer also shows the URL [localhost:8082/artifactory/webapp/](http://localhost:8082/artifactory/webapp/).

# War deployed to Artifactory

Deploying artifact: <http://localhost:8082/artifactory/libs-snapshot-local/com/demo/pipeline/web/1.1.38-SNAPSHOT/web-1.1.38-SNAPSHOT.war>

The screenshot shows the JFrog Artifactory interface. The left sidebar has links for Home, Artifacts, Builds, and Admin. The main area is titled "Artifact Repository Browser". On the left, there's a tree view of repositories: ext-release-local, ext-snapshot-local, libs-release-local, and libs-snapshot-local. Under libs-snapshot-local, there's a folder for com/demo/pipeline containing api, dataaccess, roarv2, util, and web. The web folder contains two sub-folders: 0.0.1-SNAPSHOT and 1.0.0-SNAPSHOT. The 1.0.0-SNAPSHOT folder is selected and highlighted in blue. Inside it, there are files: maven-metadata.xml, web-1.0.0-20160428.202916-5.pom, web-1.0.0-20160428.202916-5.war, web-1.0.0-20160428.222018-6.pom, and web-1.0.0-20160428.222018-6.war. To the right, a detailed view of the 1.0.0-SNAPSHOT artifact is shown. It has tabs for General (selected), Effective Permissions, and Actions. The General tab displays information: Name: 1.0.0-SNAPSHOT, Repository Path: libs-snapshot-local/com/demo/pipeline/web/1.0.0-SNAPSHOT/, Deployed by: diyuser, Artifact Count: Show, and Created: 07-04-16 23:16:13 -04:00 (35d 10h 58m 33s ago). Below this is a section for Virtual Repository Associations with a single entry: libs-snapshot.

# resources

- Files in this directory can be non-groovy
- Can be loaded via the libraryResource step in an external library; loaded as a string
- Intended to allow external libraries to load up additional non-groovy files they may need
- Examples: datafile (xml, json, etc.)
- Syntax: 

```
def datafile = libraryResource 'org/conf/data/lib/datafile.ext'
```
- libraryResource can also be used to load up any resource needed in a script (requires caution!)

```
def myExternalScript = libraryResource 'externalCommands.sh'  
sh myLatestScript
```

- can be useful to separate out non-pipeline code or programmatically specify different files to load based on conditions

# Getting the latest (highest number) artifact

```
# remove any existing wars in workspace
if [ -e *.war ]; then rm *.war; fi

# Artifactory location
server=http://localhost:8082/artifactory
repo=libs-snapshot-local

# Maven artifact location
name=web
artifact=com/demo/pipeline/$name
path=$server/$repo/$artifact
version=`curl -s $path/maven-metadata.xml | grep latest | sed "s/.*/<latest>\1</latest>.*\1/"` version =1.1.3-SNAPSHOT
build=`curl -s $path/$version/maven-metadata.xml | grep '<value>' | sort -t- -k2,2nr | head -1 | sed "s/.*/<value>\1</value>.*\1/"` war=$name-$build.war
url=$path/$version/$war

# Download
echo $url
wget -q -N $url
```

Jenkins Build Step

cleanup

path =<http://localhost:8082/artifactory/libs-snapshot-local/com/demo/pipeline/web>

version =1.1.3-SNAPSHOT

The screenshot shows the JFrog Artifactory interface. On the left, the sidebar has 'Artifacts' selected. The main area is titled 'Artifact Repository Browser'. It shows a tree view of repositories: 'libs-snapshot-local' contains 'com/demo/pipeline' which further contains 'api', 'dataaccess', 'roarv2', and 'web'. The 'web' folder has three sub-folders: '1.1.0-SNAPSHOT', '1.2.5-SNAPSHOT', and '1.1.3-SNAPSHOT'. The '1.1.3-SNAPSHOT' folder is highlighted with a yellow box. Inside it, there are files: 'maven-metadata.xml' (also highlighted), 'web-1.1.3-20160509.202726-1.pom' (highlighted), 'web-1.1.3-20160509.202726-1.war' (highlighted), 'ivy-1.0.0-SNAPSHOT.xml' (highlighted), and 'maven-metadata.xml' (highlighted). A tooltip '33' is shown over the '1.1.3-SNAPSHOT' folder. On the right, a detailed view of the '1.1.3-SNAPSHOT' artifact is shown. The 'Info' section includes:

Name:	1.1.3-SNAPSHOT
Repository Path:	libs-snapshot-local/com/demo/pipeline/web/1.1.3-SNAPSHOT/
Deployed by:	diyuser
Artifact Count:	Show
Created:	09-05-16 16:27:48 (0d 0h 12m 6s ago)

The 'Virtual Repository Associations' section lists 'libs-snapshot'.

# Lab 6 - Artifactory

# What is Docker?

- (Marketing) From docker.com

An open platform for distributed applications for developers and sysadmins.

Open source project to ship any app as a lightweight container

- (Technical)

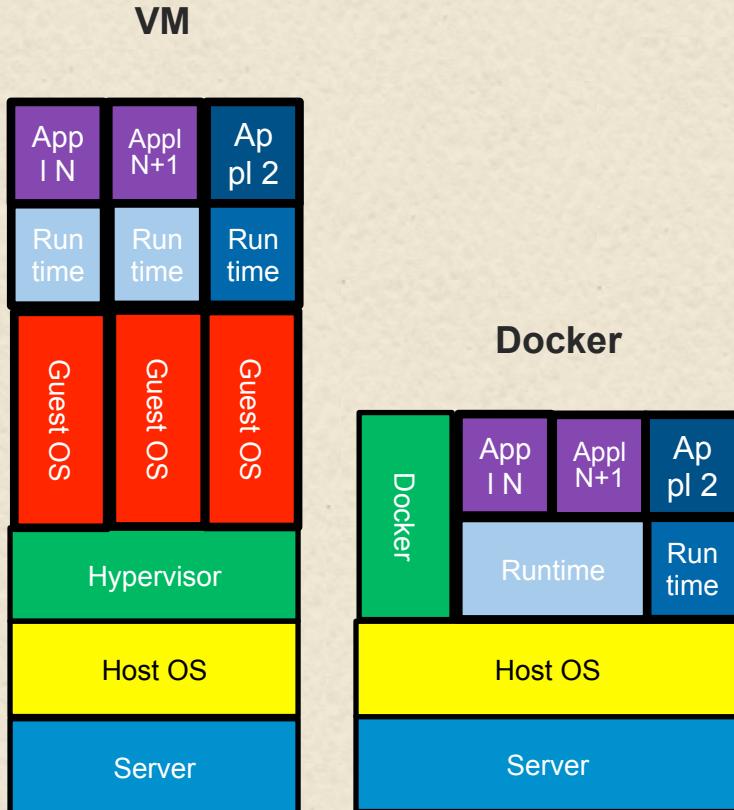
Thin wrapper around Linux Container technology

Leverages 3 functionalities from Linux to provide isolated environment in the OS

- » union filesystem - data
- » namespaces - visibility (pid)
- » cgroups - control groups (resources)

- Provides restful interface for service
- Provides description format for containers
- Provides API for orchestration

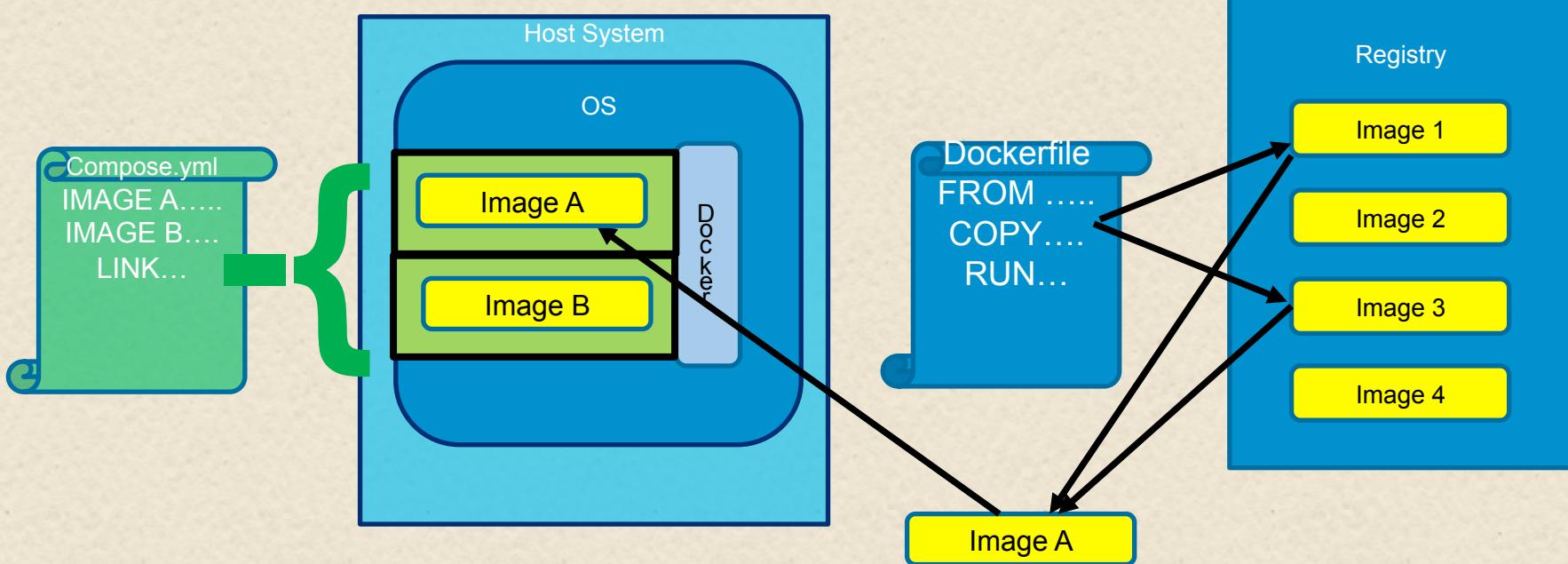
# How Docker differs from a VM



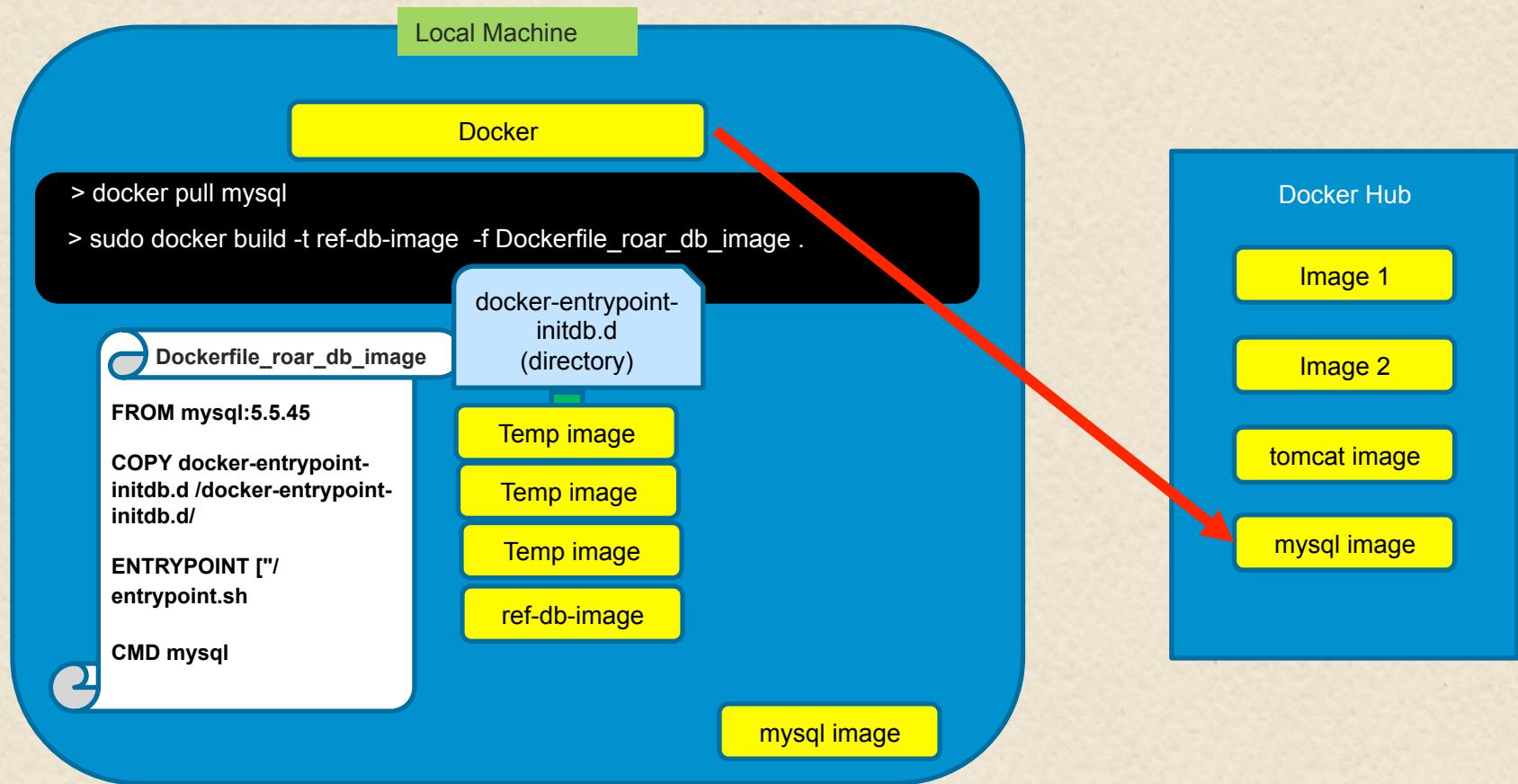
- A VM requires a Hypervisor and a Guest OS to create isolation; Docker uses Linux container technologies to run processes in separate spaces on the same OS
- Because it doesn't require the Hypervisor and a Guest OS, Docker is:
  - Faster to startup
  - More portable (can run an image unchanged in multiple environments)

# Docker Flows

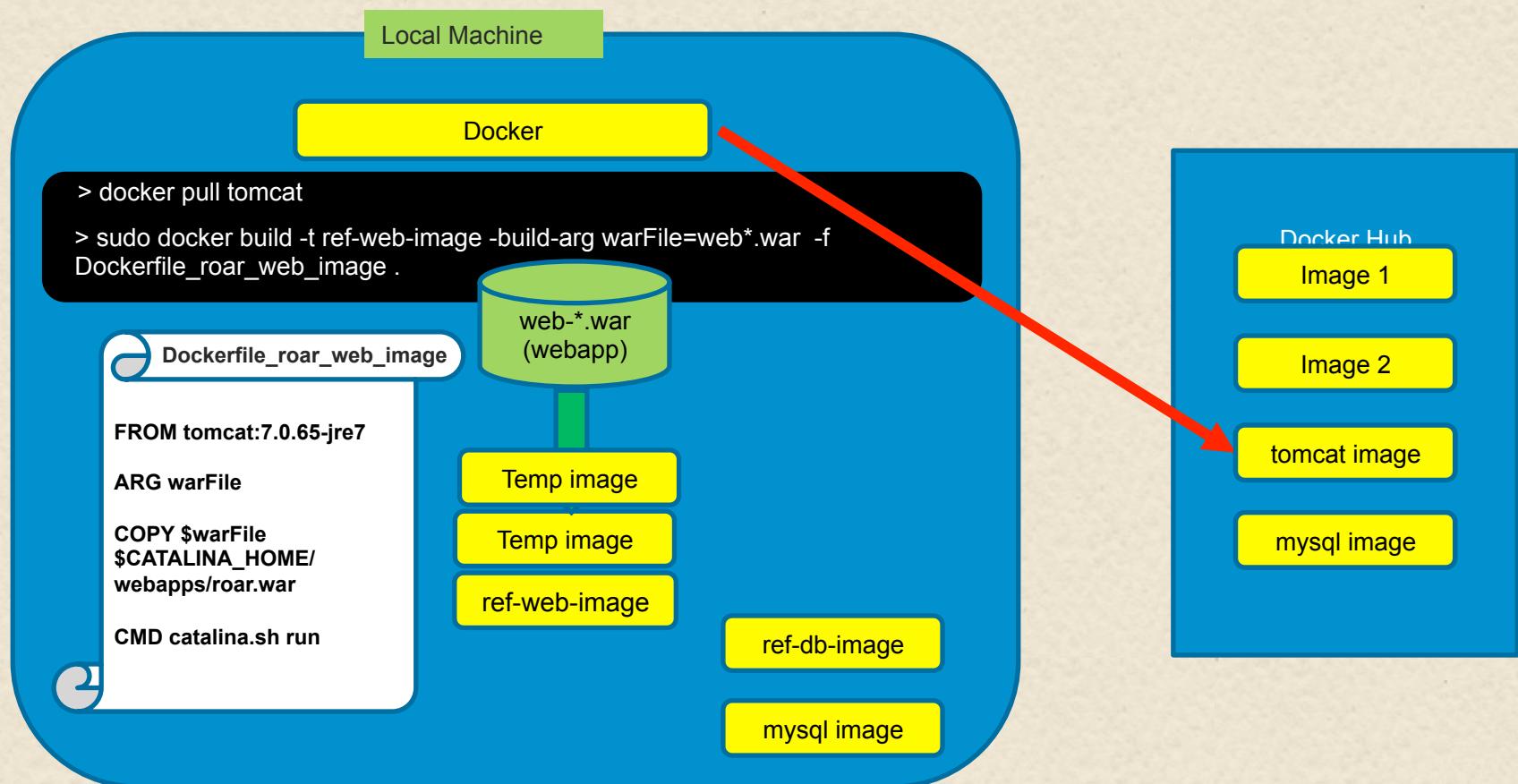
- Docker images are stored in a registry
- A Dockerfile describes how to create a new image
- docker build creates the new image
- docker run starts up a container with an image
- A docker compose file describes how to create containers and link multiple ones together
- docker-compose executes a compose file and runs the unit



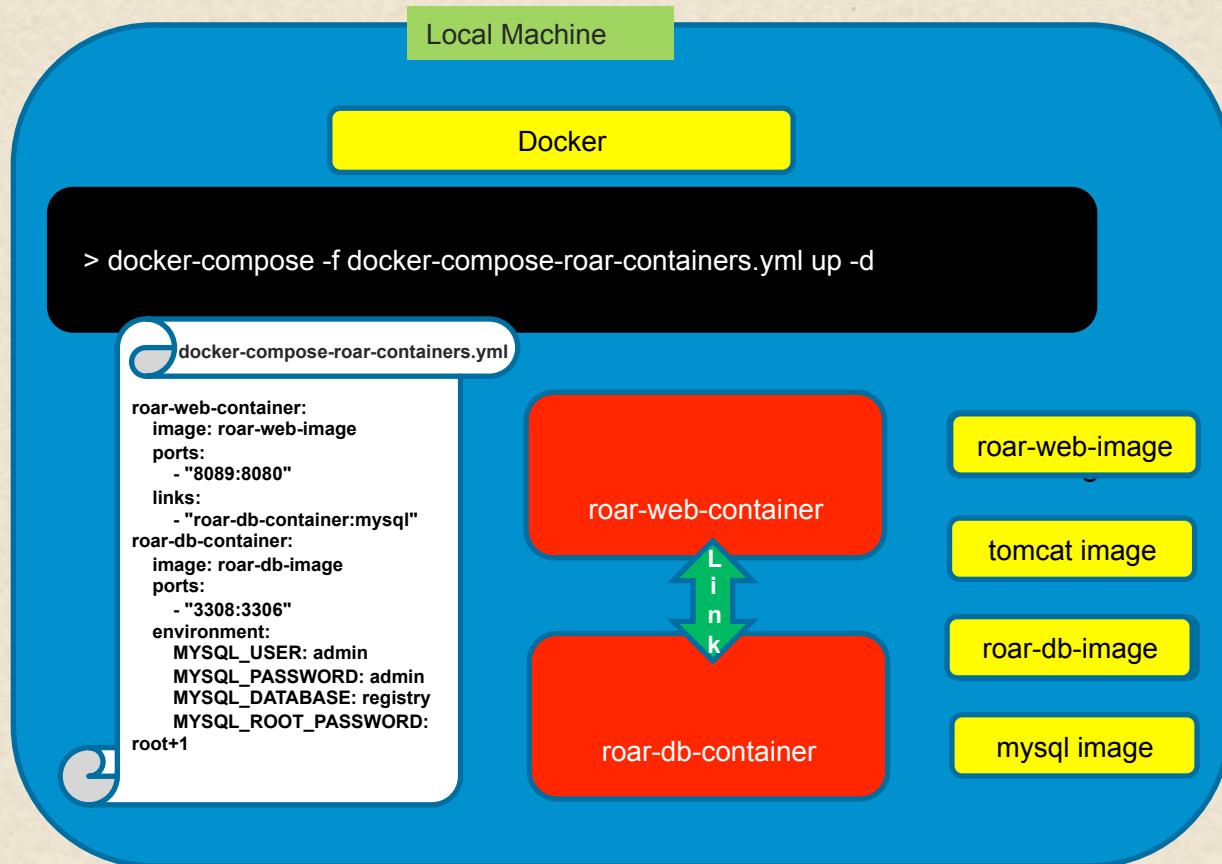
# Creating the docker database image



# Creating the web image



# Composing the combined images and containers



# Jenkins 2 and Docker

- 3 ways of running Docker via Jenkins
  - Configured as a “cloud” - standalone Jenkins agent
  - DSL Run a command such as “inside” of any docker image
  - Directly vs shell call
- Cloud
  - Provides completely self-contained environment
  - Option provided by having Docker plugin (not Docker Pipeline Plugin) installed
  - Requires Docker image that can function as “standalone agent”
    - » has slave.jar, java runtime, probably ssh

The screenshot shows the Jenkins configuration interface for a Docker cloud. It includes fields for Name, Docker URL, Docker API Version, Credentials, Connection Timeout, Read Timeout, Container Cap, and a dropdown for Images. A 'Test Connection' button is also present.

**Cloud**

Docker

Name:

Docker URL:

The URL to use to access your Docker server API (e.g: `tcp://172.16.42.43:4243` or `unix:///var/run/docker.sock`). (from Docker plugin)

Docker API Version:

Credentials:

Connection Timeout:

Read Timeout:

Container Cap:

Images:

List of Images to be launched as slaves

# Jenkins 2 and Docker

86

- Run a command “inside” of any Docker image (via Docker Pipeline Plugin)
    - Plugin provides global “Docker” variable
    - Choose image and use “inside” DSL command to execute build step in Docker image
    - Options to pass to Docker can be specified in () - `image.inside('-v ...')`
    - Inside command will:
      - » **Get an agent and workspace**
        - » **Node block not required**
      - » **If not already present, pull image**
      - » **Start container with that image**
      - » **Mount workspace from Jenkins**
        - » **as volume inside container**
        - » **appears as same filepath**
        - » **must be on same filesystem**
      - » **Executes build steps**
        - » **sh commands wrapped with “docker exec” to run in container**
      - » **Stop container and get rid of storage**
      - » **Create record that image was used for this build - for awareness for image updates, traceability, etc.**

```
stage ("Get Source") {
    // run a command to get the source code down
    myImg.inside('-v /home/git/repos:/home/git/repos') {
        sh "rm -rf gradle-greetings"
        sh "git clone --branch test /home/git/repos/gradle-greetings.git"
    }
}
stage ("Run Build") {
    myImg.inside() {
        sh "cd gradle-greetings && gradle -g /tmp clean build -x test"
    }
}

[Pipeline] stage
[Pipeline] { (Get Source)
[Pipeline] sh
[workspace] Running shell script
+ docker inspect -f . my-image:snapshot
.
[Pipeline] withDockerContainer
$ docker run -t -d -u 1002:1002 -v /home/git/repos:/home/git/repos -w /var/lib/jenkins/jobs/docker-test2/workspace:rw -v /var/lib/jenkins/jobs/docker-test2/workspace@tmp:/var/***** -e ***** -e ****
--entrypoint cat my-image:snapshot
[Pipeline] {
[Pipeline] sh
[workspace] Running shell script
+ rm -rf gradle-greetings
[Pipeline] sh
[workspace] Running shell script
+ git clone --branch test /home/git/repos/gradle-greetings.git
Cloning into 'gradle-greetings'...
done.
[Pipeline] }
$ docker stop --time=1 21aefe948bc96b55543d58fb3d45ad711582ae75b34e9b511bc0a3b83eb87f34
$ docker rm -f 21aefe948bc96b55543d58fb3d45ad711582ae75b34e9b511bc0a3b83eb87f34
[Pipeline] // withDockerContainer
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
```

# Jenkins 2 and Docker - withDockerContainer

87

- Alternate way to run a set of steps in a predefined container
- Takes an image as an argument
- Used implicitly by method calls on Docker global variable
- Starts up a container and runs any sh commands in the container
- workspace is automatically mounted R/W into container

```
worker2: { node('worker_node2') {
    // always run with a new workspace
    step([$class: 'WsCleanup'])
    unstash 'test-sources'
    withDockerContainer('gradle:3.4.1') {
        gbuild2 '-D test.single=TestExample1 test'
    }
},
```

# Declarative Pipelines - Docker agents

88

- **agent { docker '<image>' }** - pull the given image from Docker Hub and run the pipeline or stage in a container based on the image - on a dynamically provisioned node.
- **agent docker { <elements> }** - allows for defining more specifics about the docker agent. 3 additional elements can be in the declaration (within {} block).
  - » **image '<image>'** - pull the given image and use it to run the pipeline code
  - » **label '<label>'** - optional - tells Jenkins to instantiate the container and "host" it on a node matching <label>.
  - » **args '<string>'** - optional - tells Jenkins to pass these arguments to the docker container; uses same docker syntax as you would normally use
- **agent { dockerfile true }** - Note that "dockerfile" here is a literal. Used when the source code repository that you retrieve that has a Dockerfile in its root. Tells Jenkins to build a docker image using that Dockerfile (from the root of the SCM repo), instantiate a container, and then run the pipeline or stage code in that container.
- **agent dockerfile { <elements> }** - long syntax allows for defining more specifics about the docker agent you are trying to create from a dockerfile. 3 additional elements that can be added in the declaration (within the {} block).
  - » **filename '<path to dockerfile>'** - Specify an alternate path to a dockerfile including directories and a different name. Jenkins will try to build an image from the dockerfile, instantiate a container, and use it to run the pipeline code.
  - » **label '<label>'** - optional - tells Jenkins to instantiate the container and "host" it on a node matching <label>.
  - » **args '<string>'** - optional - tells Jenkins to pass these arguments to the docker container; same syntax as normally used for Docker
- **reuseNode** - tells Jenkins to reuse the same node and workspace that was defined for the original pipeline agent to "host" the resulting docker container.

```
agent {  
    docker {  
        image "image-name"  
        label "worker-node"  
        args "-v /dir:dir"  
    }  
}
```

```
agent {  
    dockerfile {  
        filename "<subdir/docker file name>"  
        label "<agent label>"  
        args "-v /dir:dir"  
    }  
}
```

```
pipeline {  
    agent label 'linux'
```

```
stage 'abc' {  
    agent {  
        docker {  
            image 'ubuntu:16.6'  
            reuseNode true  
        }  
    }  
}
```

# Jenkins 2.0 and Docker - Other Operations

89

The screenshot shows the Jenkins Pipeline Syntax documentation for the 'docker' step. It includes a navigation bar with links to Step Reference, Global Variables Reference, Online Documentation, and IntelliJ IDEA DSL. The main content is titled 'Global Variable Reference' under the 'Variables' section. The 'docker' variable is described as offering convenient access to Docker-related functions from a Pipeline script. It includes several methods:

- withRegistry(url[, credentialsId]) { ... }**: Specifies a registry URL such as `https://docker.mycorp.com/`, plus an optional credentials ID to connect to it.
- withServer(url[, credentialsId]) { ... }**: Specifies a server URL such as `tcp://swarm.mycorp.com:2376`, plus an optional credentials ID to connect to it.
- withTool(toolName) { ... }**: Specifies the name of a Docker installation to use, if any are defined in Jenkins global configuration. If unspecified, docker is assumed to be in the \$PATH of the slave agent.
- Image(id)**: Creates an Image object with a specified name or ID. See below.
- build(image[, args])**: Runs docker build to create and tag the specified image from a Dockerfile in the current directory. Additional args may be added, such as `-f Dockerfile.foobar --pull`. Object. Records a FINGERPRINT in the build.
- Image.id**: The image name with optional tag (`mycorp/myapp`, `mycorp/myapp:latest`) or ID (hexadecimal hash).
- Image.run([args], command)**: Uses docker run to run the image, and returns a Container which you could stop later. Additional args may be added, such as `-p 8080:8080 --memory swap=1`. Options `withRun([args], command)` { ... } like run but stops the container as soon as its body exits, so you do not need a try-finally block.
- Image.inside([args]) { ... }**: Like withRun but starts this as a container for the duration of the body, but all external commands (STD) launched by the body run inside the container rather than on the host. These commands are run in a shell.
- Image.tag([tagname])**: Runs docker tag to record a tag of this image (defaulting to the tag it already has). Will rewrite an existing tag if one exists.
- Image.push([tagname])**: Pushes an image to the repository after tagging it as with the tag method. For example, you can use `Image.push 'latest'` to publish it as the latest version in its repository.
- Image.pull()**: Runs docker pull. Not necessary before run, withRun, or inside.
- Image.imageName()**: The ID prefixed as needed with registry information, such as `docker.mycorp.com/mycorp/myapp`. May be used if running your own Docker commands using STD.

```
node() {
    def myImg
    stage ("Build image") {
        // download the dockerfile to build from
        git 'git@diyvb:repos/dockerResources.git'

        // build our docker image
        myImg = docker.build 'my-image:snapshot'
    }
    stage ("Get Source") {
```

```
[Pipeline] stage
[Pipeline] { (Build image)
[Pipeline] git
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url git@diyvb:repos/dockerResources.git
Fetching upstream changes from git@diyvb:repos/dockerResources.git
> git --version # timeout=10
> git fetch --tags --progress git@diyvb:repos/dockerResources.git
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit}
Checking out Revision 742b984c53e96e7d1465d9442af6c6606757e845
> git config core.sparsecheckout # timeout=10
> git checkout -f 742b984c53e96e7d1465d9442af6c6606757e845
> git branch -a -v --no-abbrev # timeout=10
> git branch -D master # timeout=10
> git checkout -b master 742b984c53e96e7d1465d9442af6c6606757e845
> git rev-list 742b984c53e96e7d1465d9442af6c6606757e845 # timeout=10
[Pipeline] sh
[workspace] Running shell script
+ docker build -t my-image:snapshot .
Sending build context to Docker daemon 289.8 kB

Step 1 : FROM java:8-jdk
--> 861e95c114d6
Step 2 : MAINTAINER B. Laster (bclaster@nclusters.org)
--> Using cache
--> 48b4694fbab0
Step 3 : ENV GRADLE_VERSION 2.14.1
--> Using cache
--> c84de3a28e12
Step 4 : RUN cd /opt && wget https://services.gradle.org/distributions/gradle-2.14.1-bin.zip && ln -s "/opt/gradle-$GRADLE_VERSION/bin/gradle" /usr/local/bin/gradle
--> Using cache
--> df50ff638f0d
Step 5 : ENV GRADLE_HOME /opt/gradle
```

# Jenkins 2.0 and Docker - Running via shell

90

```
try {
    stage ("Run Tests") {
        sh "docker run --privileged --rm -v '${env.WORKSPACE}:${env.WORKSPACE}' --name '${env.BUILD_TAG}' ${myImg.id} /bin/sh -c 'cd ${env.WORKSPACE}/gradle-greetings && gradle test'"
    }
} finally {
    sh "docker rmi -f ${myImg.id} ||:"
}
```

```
[Pipeline] stage
[Pipeline] { (Run Tests)
[Pipeline] sh
[workspace] Running shell script
+ docker run --privileged --rm -v /var/lib/jenkins/jobs/docker-test2/workspace:/var/lib/jenkins/jobs/docker-test2/workspace --name jenkins-docker-test2-20 my-image:snapshot /bin/sh -c cd /var/lib/jenkins/jobs/docker-test2/workspace/gradle-greetings && gradle test
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:compileTestJava
Download https://repo1.maven.org/maven2/junit/junit/4.10/junit-4.10.pom
Download https://repo1.maven.org/maven2/org/hamcrest/hamcrest-core/1.1/hamcrest-core-1.1.pom
Download https://repo1.maven.org/maven2/org/hamcrest/hamcrest-parent/1.1/hamcrest-parent-1.1.pom
Download https://repo1.maven.org/maven2/junit/junit/4.10/junit-4.10.jar
Download https://repo1.maven.org/maven2/org/hamcrest/hamcrest-core/1.1/hamcrest-core-1.1.jar
:processTestResources UP-TO-DATE
:testClasses
:test

TestExample2 > example2 FAILED
  org.junit.ComparisonFailure at TestExample2.java:10

4 tests completed, 1 failed
:test FAILED
```

10

# Lab 7 – Deploy to Docker

# That's all - thanks!

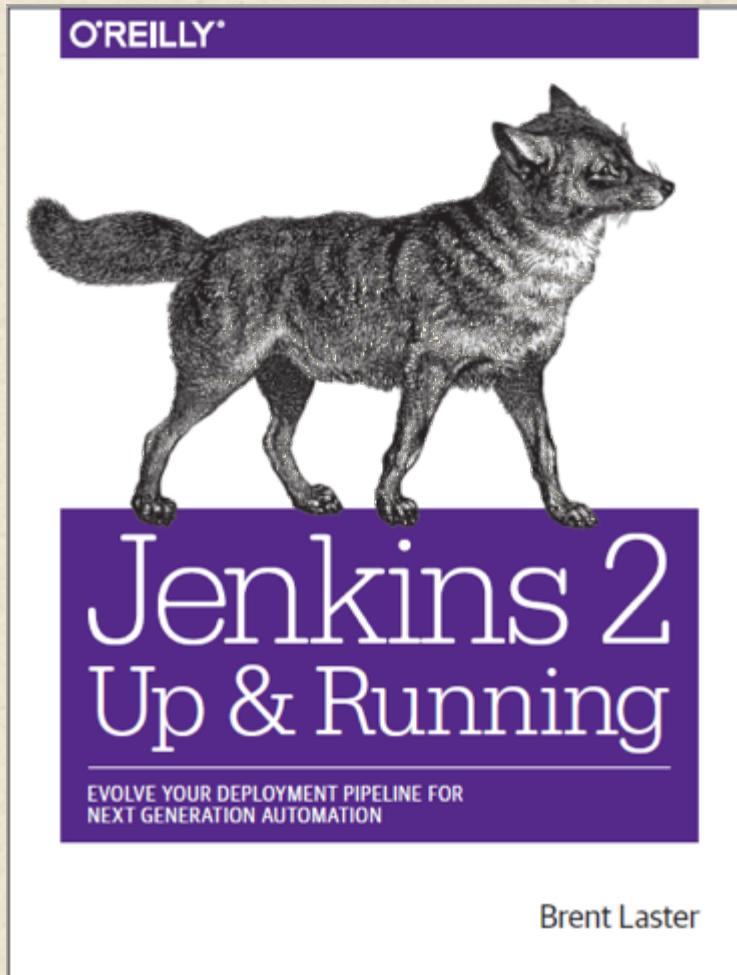
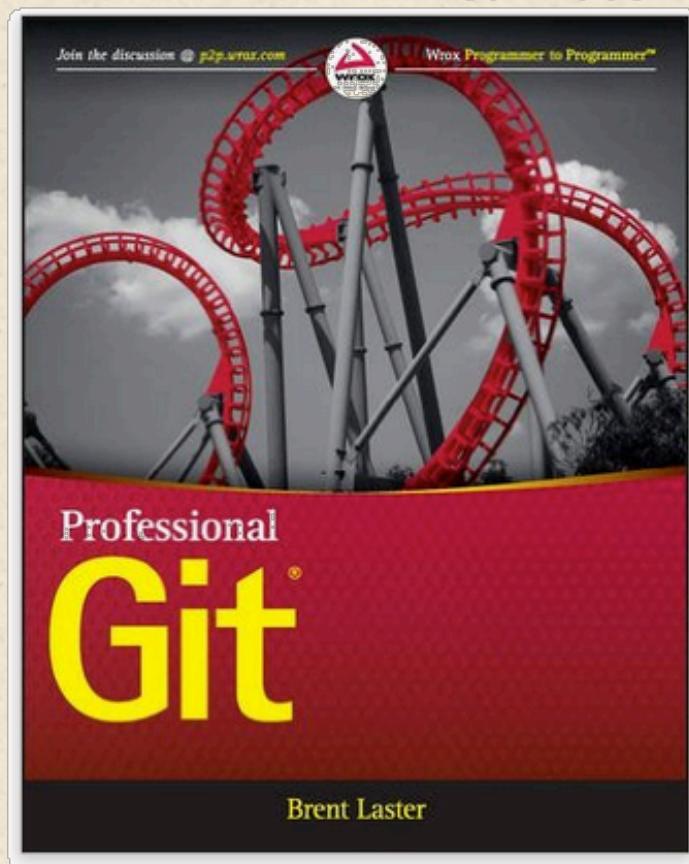
## Professional Git 1st Edition

by Brent Laster ▾ (Author)



3 customer reviews

[Look inside](#) ↓



© 2018 Brent Laster

92